On the Tape Complexity of Deterministic Context-Free Languages

I H SUDBOROUGH

Northwestern University, Evanston, Illinois

ABSTRACT Let DSPACE($L(n)$) denote the family of languages recognized by deterministic $L(n)$-tape bounded Turing machines The principal result described in this paper is the equivalence of the following statements (1) The deterministic context-free language $L_0^{(2)}$ (described in the paper) is in DSPACE(log(n)) (2) The simple LL(1) languages are in DSPACE(log(n)) (3) The simple precedence languages are in DSPACE(log(n)). (4) DSPACE(log(n)) is identical to the family of languages recognized by deterministic two-way multihead pushdown automata in polynomial time These results are obtained by constructing a deterministic context-free language $L_0^{(2)}$ which is log(n)-complete for the family of deterministic context-free languages In other words, a tape hardest deterministic context-free language is described The best upper bound known on the tape complexity of (deterministic) context-free languages is $(\log(n))^2$

KEY WORDS AND PHRASES deterministic context-free languages, tape complexity, simple precedence languages, simple LL(1) languages, Turing machine, log(n)-tape reducibility, log(n) complete, polynomial time bounded multihead pushdown automata, Dyck languages

CR CATEGORIES 5 23, 5 25

Lewis, Stearns, and Hartmanis [23] have described an algorithm to recognize every context-free language by a deterministic $(\log(n))^2$-tape bounded Turing machine. In a recent paper [26] the author has shown that, if the context-free languages could be recognized by a deterministic log(n)-tape bounded Turing machine, then nondeterministic and deterministic $L(n)$-tape bounded complexity classes are identical, for $L(n) \geq \log(n)$. In this paper the tape complexity of deterministic context-free languages is considered. It is shown that there is a single deterministic context-free language $L_0^{(2)}$ which is recognized by a deterministic [nondeterministic] log(n)-tape bounded Turing machine if, and only if, all deterministic context-free languages can be recognized by deterministic [nondeterministic] log(n)-tape bounded Turing machines. Also, $L_0^{(2)}$ may be log(n)-tape reduced to a simple LL(1) language, as discussed by Korenjak and Hopcroft [22] and Aho and Ullman [2], and to a simple precedence language, as discussed by Wirth and Weber [28] and Aho and Ullman [2]. It is shown, therefore, that these proper subfamilies of the deterministic context-free languages are as difficult to recognize as the complete family of deterministic context-free languages. Furthermore, it is shown that $L_0^{(2)}$ is recognized by a deterministic log(n)-tape bounded Turing machine if, and only if, the family of languages recognized by deterministic multihead two-way pushdown automata in polynomial time is identical to

General permission to make fair use in teaching or research of all or part of this material is granted to individual readers and to nonprofit libraries acting for them provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery To otherwise reprint a figure, table, other substantial excerpt, or the entire work requires specific permission as does republication, or systematic or multiple reproduction

Some of these results were presented at the Eighth Annual ACM Symposium on the Theory of Computing held in Hershey, Pennsylvania, May 3–5, 1976

This work was supported in part by the National Science Foundation under Grant GJ-43228

Author's address: Computer Science Department, The Technological Institute, Northwestern University, Evanston, IL 60201

© 1978 ACM 0004-5411/78/0700-405 $00 75

the family of languages recognized by deterministic log(n)-tape bounded Turing machines. Since we believe it is unlikely that these families are identical, we conjecture that $L_0^{(2)}$ cannot be recognized in log(n)-space.

As indicated our results provide evidence to support the conjecture that the deterministic context-free languages are not recognized in log(n)-space. In this respect they are similar to the results of Jones and Laaser [19], Galil [11], Cook [8], and Book [3], which describe evidence that the family of languages recognized in polynomial time by deterministic Turing machines is not contained in the family recognized in $(\log(n))^k$-space, for any $k \geq 1$. The results are also similar in this respect to the description of complete problems for the question of whether or not nondeterministic polynomial time bounded algorithms can be replaced by equivalent deterministic polynomial time bounded algorithms as in Cook [5] and Karp [20]. A context-free language which is hardest with respect to tape and time complexity measures has also been recently described by Greibach [14]

It is shown, moreover, that the family of languages log-tape reducible to context-free languages [deterministic context-free languages] is exactly the set of languages recognized by nondeterministic [deterministic] log(n)-tape bounded auxiliary pushdown automata within polynomial time. Cook [4] has shown that every language in these two families can be recognized in $(\log(n))^2$-space by a deterministic Turing machine. It follows from the characterization given here that Cook's result follows from the algorithm given by Lewis, Stearns, and Hartmanis [23] for context-free languages via log-tape reducibilities.

A Tape Hardest Deterministic Context-Free Language

Our results are best described using the notions of log(n)-tape reducibility and log(n)-completeness as defined recently by Jones [18] and Meyer and Stockmeyer [24] We include these definitions here.

Definition. Let Σ, Δ be alphabets and $f:\Sigma^* \to \Delta^*$ be a function. f is *log(n)-tape computable* if there is a deterministic Turing machine with a two-way read-only input tape, a one-way output tape, and a two-way read-write work tape, which when started with $x \in \Sigma^*$ on its input tape will halt having written $f(x) \in \Delta^*$ on its output and having visited at most $\log(|x|)$ tape squares on its work tape.

Definition. Let $A \subseteq \Sigma^*$ and $B \subseteq \Delta^*$ be arbitrary sets of words. A is *log(n)-tape reducible to* B, denoted $A \leq_{log} B$, if there is a log(n)-tape computable function f such that $\forall x \in \Sigma^*$ $(x \in A \Leftrightarrow f(x) \in B)$.

The following two lemmas are contained either explicitly or implicitly in Jones [18] and Meyer and Stockmeyer [24]:

LEMMA 1. *$\leq_{log}$ is a transitive relation.*

Definition. Let *DSPACE($L(n)$)* [*NSPACE($L(n)$)*] denote the family of languages recognized by a deterministic [nondeterministic] $L(n)$-tape bounded offline Turing machine. (These devices have a two-way read-only input tape and a separate read-write work tape on which at most $L(n)$ tape cells are visited.)

LEMMA 2. *Let $A \subseteq \Sigma^*$ and $B \subseteq \Delta^*$. If $A \leq_{log} B$ and $B \in DSPACE(\log^d(n))$ [$B \in NSPACE(\log^d(n))$], then $A \in DSPACE(\log^d(n))$ [$A \in NSPACE(\log^d(n))$] for any real number $d \geq 1$.*

Definition. Let $\mathscr{L}$ be a family of languages $L \subseteq \Sigma^*$ is *log(n)-complete for $\mathscr{L}$* if (1) $L \in \mathscr{L}$, and (2) for every language L' in $\mathscr{L}$, $L' \leq_{\log} L$.

We shall describe a language $L_0^{(2)}$ which is log(n)-complete for the family of deterministic context-free languages. The deterministic context-free languages are simply the languages recognized by deterministic (one-way) pushdown automata and have often been discussed in the literature because of their application to compiler design; for example, see Ginsburg and Greibach [12], Knuth [21], and Aho and Ullman [2].

A two-way deterministic (nondeterministic) multihead pushdown automaton is a recognition device with a finite state control, a fixed number of heads scanning a read-only input tape, and a pushdown (last-in-first-out) storage tape. These devices have been

considered previously by Harrison and Ibarra [15], Cook [7], Ibarra [17], Galil [11], and others. Cook [7] has shown that the family of languages recognized by two-way nondeterministic and deterministic multihead pushdown automata are identical and that it is the same as the family of languages recognized by deterministic polynomial time bounded Turing machines. It is well known that this family is also identical to the family of languages recognized by deterministic [nondeterministic] log(n)-tape bounded Turing machines which have an additional unbounded pushdown store. (These are the log(n)-tape bounded auxiliary pushdown automata of Cook [7].) The deterministic [nondeterministic] two-way pushdown automata considered by Gray, Harrison, and Ibarra [13], Cook [5], Aho, Hopcroft, and Ullman [1], and Galil [11], for example, are simply the special case when only one head is allowed to process the input tape. A deterministic [nondeterministic] pushdown automaton is obtained by the further restriction that the single input tape head proceed from left to right only.

We shall assume that the input tape of a two-way multihead pushdown automaton is delimited by a left and a right endmarker. We shall say that an n-head pushdown automaton accepts a string written on the input tape, if when started in its initial state with all n heads scanning the left endmarker and the initial tape symbol in the pushdown store, there exists a sequence of transitions which takes the pushdown automaton to a final state. Such a device is said to be deterministic if there is at most one possible transition for any given configuration the device may enter. An e-move by a one-way single-head pushdown automaton is a transition which does not move the input head to the right. In the case of deterministic pushdown automata it is known that if e-moves are not allowed, then the family of languages recognized is a proper subset of the deterministic context-free languages.

Let LOG($\mathscr{L}$) = {L | L $\leq_{\log}$ L′ for some L′ in $\mathscr{L}$} denote the family of languages which are log(n)-tape reducible to the family $\mathscr{L}$. Let 2NPDA(k) [2DPDA(k)] denote the family of languages recognized by k-head nondeterministic [deterministic] two-way pushdown automata. (We shall often omit the one, in the case $k = 1$, and refer to 2NPDA and 2DPDA.) We show that the family of languages recognized in polynomial time by nondeterministic [deterministic] multihead pushdown automata is LOG(CFL) [LOG(DCFL)]. (Here CFL [DCFL] denotes the family of context-free languages [deterministic context-free languages].) In the following we shall use 2NPDA(k) and 2DPDA(k) to denote the automata as well as the family of languages they recognize (Context should clarify any ambiguity.)

The proof of the following lemma is similar to the proof of Theorem 1 in [26, p. 67]. The only difference is that here we are concerned with polynomial time bounded pushdown automata and not with multihead finite automata. However, the basic construction is to reduce within log(n) tape any language recognized by a two-way automaton to a language recognized by a one-way automaton of the same general type.

LEMMA 3. *If L is recognized by a 2NPDA [2DPDA] in polynomial time, then L $\leq_{\log}$ L′ for some context-free [deterministic context-free] language L′*

PROOF. Let q(n) be a polynomial and M = (S, Σ, Γ, δ, p_0, Z_0, F) be a 2NPDA [2DPDA] such that $x \in$ L if and only if M has a computation of at most q(|x|) steps on input x which causes it to enter a final state. We assume, without any loss of generality, that M reverses the direction of motion of its input head only when it is scanning an endmarker symbol and that each transition moves the input head to the left or to the right. Let ¢ and \$ be the left and right endmarker symbols bounding the input tape of M

Construct the one-way pushdown automaton M′ = (S ∪ {f}, Σ ∪ {¢, \$}, Γ, δ′, p_0, Z_0, {f}), where:

(q, γ) ∈ δ′(p, a, Z) ⇔ (q, γ, λ) ∈ δ(p, a, Z) for any λ ∈ {L, R} (indicating a move to the left or right, respectively), p ∈ S − F, a ∈ Σ ∪ {¢, \$}, Z ∈ Γ, and γ ∈ Γ*;
(f, Z) ∈ δ′(p, a, Z) if p ∈ F, a ∈ Σ ∪ {¢, \$}, and Z ∈ Γ;
(f, Z) ∈ δ′(f, a, Z) for all a ∈ Σ ∪ {¢, \$}, and Z ∈ Γ.

(Observe that M′ is deterministic if M is deterministic.) Let h be the function from Σ^* to $(\Sigma \cup \{\not{\epsilon}, \$\})^*$ defined by $h(x) = (\not{\epsilon}x\$x^R)^{q(|x|)}$. It follows from the construction of M′ that $x \in L$ if and only if $h(x)$ is recognized by M′. It is straightforward to verify that h is $\log(n)$-tape computable. Thus, let L′ be the context-free [deterministic context-free language] recognized by M′. □

The following lemma is a straightforward adaptation of Lemma 2.3 of [11, p. 173] which was based in turn on results in [17]. The following definition is used in the lemma.

Definition. Let a and b be two symbols not in Σ. For any language $L \subseteq \Sigma^*$ let $T(L) = \{(axb)^{|axb|} | x \in L\}$

LEMMA 4. *L is recognized by a 2NPDA(2k) [2DPDA(2k)] in polynomial time if and only if T(L) is recognized by a 2NPDA(k) [2DPDA(k)] in polynomial time.*

Combining Lemma 3 and Lemma 4, we obtain:

LEMMA 5. *For any $k \geq 1$, if L is recognized by a 2NPDA(k) [2DPDA(k)] in polynomial time, then there is a context-free language L′ such that $L \leq_{log} L'$.*

PROOF It is straightforward to verify that $L \leq_{\log} T(L)$. Therefore, by a finite number of applications of Lemma 4 and the transitivity of $\leq_{\log}$, $L \leq_{\log} L''$ for some L″ recognized by a 2NPDA [2DPDA] in polynomial time. By Lemma 3 we conclude that there is a context-free [deterministic context-free] language L′ such that $L'' \leq_{\log} L'$. The result follows from the transitivity of the relation $\leq_{\log}$. □

It has been shown that the languages recognized in polynomial time by nondeterministic [deterministic] multihead pushdown automata are contained in LOG(CFL) [LOG(DCFL)]. Containment in the other direction is also true.

LEMMA 6. *If $L \leq_{log} L'$ for some context-free [deterministic context-free] language L′, then L is recognized by a 2NPDA(k) [2DPDA(k)] in polynomial time, for some $k \geq 1$.*

PROOF. By well-known techniques, see for example [16], it can be shown that $\bigcup_{k\geq 1}$ 2NPDA(k) [$\bigcup_{k\geq 1}$2DPDA(k)] is identical to the family of languages recognized by nondeterministic [deterministic] $\log(n)$-tape bounded auxiliary pushdown automata. (The equivalence is also true with the polynomial time constraint.) Thus it is sufficient to show recognition by a nondeterministic [deterministic] $\log(n)$-tape bounded auxiliary pushdown automaton in polynomial time.

Since $L \leq_{\log} L'$, there is a $\log(n)$-tape computable function f such that $x \in L$ if and only if $f(x) \in L'$. Let P be the auxiliary PDA which on input x computes $f(x)$ symbol by symbol, using the $\log(|x|)$ bounded auxiliary tape, and simulates the one-way pushdown automaton P′ that recognizes L′ on the symbols of $f(x)$ as they are generated. (P needs, in general, to use the pushdown store to simulate P′.) If P′ enters a final state after processing all of $f(x)$, then P enters a final state. Clearly P operates in polynomial time, since $f(x)$ is produced in polynomial time and P′ can be assumed, without any loss of generality, to operate in linear time. It is also straightforward to verify that P recognizes x if and only if $f(x)$ is in L′. □

We summarize the results of Lemma 5 and Lemma 6 in the following:

THEOREM 1. *LOG(CFL) [LOG(DCFL)] is identical to the family of languages recognized by nondeterministic [deterministic] log(n)-tape bounded auxiliary pushdown automata in polynomial time.*

Let $DCFL_e$ denote the family of languages recognized by empty store by deterministic pushdown automata which are allowed no e-moves. Although $DCFL \neq DCFL_e$, the next lemma shows that for tape complexity considerations it is sufficient to consider $DCFL_e$.

LEMMA 7. *For every L in DCFL there is an L′ in $DCFL_e$ such that $L \leq_{log} L'$*

PROOF. Let P be a one-way deterministic pushdown automaton that recognizes $L \subseteq \Sigma^*$. We shall assume that c_1 and c_2 are integers such that on any input x P makes at most $c_1|x| + c_2$ transitions. (This may be assumed, without any loss of generality, by the proof of Theorem 2.22 of [2]). Let B and $\not{\#}$ be new symbols not in Σ and, for each positive integer n, define the homomorphism $H_n : \Sigma^* \to (\Sigma \cup \{B\})^*$ by $H_n(a) = aB^n$, for all $a \in \Sigma$. Let f be the function from Σ^* to $(\Sigma \cup \{B\})^*$ defined by $f(x) = H_n(x)\not{\#}$, where $n = c_1|x| + c_2$. It is straightforward to verify that f is log-tape computable Let P′ be the one-way deterministic pushdown automaton that on an input of the form

$$a_1B^{n_1}\ a_2B^{n_2}\ a_3B^{n_3}\ \cdots\ a_kB^{n_k}\#,$$

where $n_1, n_2, \ldots, n_k$ are nonnegative integers, simulates P on the input $a_1a_2\cdots a_k$. Each time P executes an e-move P′ moves its head to the right instead. If P′ in simulating e-moves of P moves its head to the right and does not scan an occurrence of the symbol B, then P′ stops and rejects the input. On the other hand, if the input head of P′ is scanning an occurrence of the symbol B and the next transition of P to be simulated is not an e-move, then P′ will move its input head to the right until encountering the next symbol from Σ. Furthermore, P′ can accept by empty store in the following ways: (1) by adding a special symbol to the bottom of the store initially, (2) whenever P enters a final state, P′ will empty its store down to the special symbol while preceding to the right, and (3) when the symbol # is encountered on the input P′ will erase the special symbol and stop.

Let L′ be the language recognized by P′. It follows that $x \in L \Leftrightarrow f(x) \in L'$ and L′ is in $DCFL_e$. □

Definition. Let $Z_0, Z_1, \ldots, Z_m, \bar{Z}_0, \bar{Z}_1, \ldots, \bar{Z}_m$ by distinct symbols The Dyck language D_{m+1} is the language generated by the context-free grammar with the rules: $S \to SS$, $S \to Z_0S\bar{Z}_0$, $S \to Z_1S\bar{Z}_1, \ldots, S \to Z_mS\bar{Z}_m$, $S \to e$.

The reader will not go astray in identifying D_{m+1} with the language of all well-formed expressions of nested parentheses of $m + 1$ types. In this vein the symbol $\bar{Z}_i$ is said to cancel the corresponding symbol Z_i. Hence the last uncanceled symbol in the string $Z_1Z_2Z_1Z_2\bar{Z}_2\bar{Z}_1$ is Z_2. Let Σ be an alphabet and], [, and # be new symbols not in Σ. In the following we shall refer to strings of the form $[x_1 \# x_2 \cdots \# x_m]$, where $x_1, x_2, \ldots, x_m$ are in Σ^* as "blocks" and each of the x_i as "choice strings" of the block.

Definition. Let $L_0^{(m+1)}$ be the set of all strings of the form:

$$\gamma_0[\bar{Z}_0\gamma_0^{(1)}\#\bar{Z}_1\gamma_1^{(1)} \cdots \#\bar{Z}_m\gamma_m^{(1)}][\bar{Z}_0\gamma_0^{(2)}\#\bar{Z}_1\gamma_1^{(2)} \cdots \#\bar{Z}_m\gamma_m^{(2)}] \cdots [\bar{Z}_0\gamma_0^{(k)}\#\bar{Z}_1\gamma_1^{(k)} \cdots \#\bar{Z}_m\gamma_m^{(k)}]\#, \qquad (*)$$

where γ_0 and $\gamma_j^{(i)}$ are in $\{Z_0, Z_1, \ldots, Z_m\}^*$, for all $1 \le i \le k$ and $0 \le j \le m$, and there exist integers $0 \le i_1, i_2, \ldots, i_k \le m$ such that

(1) $\gamma_0\bar{Z}_{i_1}\gamma_{i_1}^{(1)}\bar{Z}_{i_2}\gamma_{i_2}^{(2)} \cdots \bar{Z}_{i_k}\gamma_{i_k}^{(k)}$ is in D_{m+1}, and

(2) for all $1 \le l \le k$, $i_l = j$ if and only if $\gamma_0\bar{Z}_{i_1}\gamma_{i_1}^{(1)}\bar{Z}_{i_2}\gamma_{i_2}^{(2)} \cdots \bar{Z}_{i_{l-1}}\gamma_{i_{l-1}}^{(l-1)}$ is in $\mathrm{INIT}(D_{m+1}) \cdot \{Z_j\} \cdot D_{m+1}$, i.e. the last uncanceled symbol in the concatenation of previous choice strings is Z_j.

It should be clear that $L_0^{(m)}$, for any $m \ge 1$, is a deterministic context-free language. A deterministic pushdown automaton P_m to recognize $L_0^{(m)}$ can be described as follows[1]:

(1) P_m stores the string $\gamma_0\#$ in its pushdown store and enters step (2) Let i be equal to one

(2) If the top symbol on the pushdown store is Z_j, then P_m moves its input head to the jth string, of the form $\bar{Z}_j\gamma_j^{(i)}$ in block i, and replaces Z_j by the string $\gamma_j^{(i)}$ ($\gamma_j^{(i)}$ may be the empty string) If the pushdown store at this point contains only the symbol # that was placed at the bottom of the store initially, then P_m moves to the right until encountering the pair of symbols]# and accepts the input On the other hand, if the top symbol on the store is not #, then P_m moves its head to the end of block i without changing the contents of the pushdown store If there is another block in the input string, then P_m reenters step (2) with i replaced by $i + 1$. Otherwise, P_m stops and rejects the input

For any string x of the form (*) given in the previous definition and any sequence $i_1, i_2, \ldots, i_l$, let $x[i_1, i_2, \ldots, i_l]$ be the string $\gamma_0\bar{Z}_{i_1}\gamma_{i_1}^{(1)}\bar{Z}_{i_2}\gamma_{i_2}^{(2)} \cdots \bar{Z}_{i_l}\gamma_{i_l}^{(l)}$. Recall Lemma 7; our next result shows that $L_0^{(3)}$ is a hardest deterministic context-free language with respect to the space (or tape) complexity measure.

LEMMA 8. *For every L in DCFL$_e$, $L \le_{log} L_0^{(3)}$*

PROOF. Let P be a one-way deterministic pushdown automaton without e-moves that

[1] We assume that P_m checks also that the string is in the right form, as given in (*). This may be done since the set of strings of this form is a regular set and DCFL is closed under intersection with regular sets

accepts L by empty store. We assume, without any loss of generality, that P has two symbols in its pushdown store alphabet, say Z_1 and Z_2, and that P reenters its initial state when it accepts by emptying its store. (It is evident from the proof of Lemma 7 that we may assume this last property, since if the pushdown automaton constructed there has the special "bottom-of-stack" symbol alone on its store it may enter any state it chooses before erasing it. Thus, every DCFL can be reduced in $\log(n)$ space to an L in $\mathrm{DCFL_e}$, which can be recognized by a pushdown automaton which empties the store and simultaneously reenters the initial state.) Let P have k states: $p_1, p_2, \ldots, p_k$, where p_1 is the initial state. Let Z_1 be the initial pushdown store symbol.

Corresponding to each symbol a we define a sequence of k blocks, denoted by $B(a)$, as follows:

$$B(a) = [\bar{Z}_0 \# \bar{Z}_1 \gamma_{11} Z_0^{k+m(1,1)-2} \# \bar{Z}_2 \gamma_{12} Z_0^{k+m(1,2)-2}][\bar{Z}_0 \# \bar{Z}_1 \gamma_{12} Z_0^{k+m(2,1)-3} \# \bar{Z}_2 \gamma_{22} Z_0^{k+m(2,2)-3}]$$
$$[\bar{Z}_0 \# \bar{Z}_1 \gamma_{k2} Z_0^{m(k,1)-1} \# \bar{Z}_2 \gamma_{k2} Z_0^{m(k,2)-1}],$$

where, for $1 \le i \le k$, P has the transitions

$$\delta(p_i, a, Z_1) = (p_{m(i,1)}, \gamma_{i1}), \qquad \delta(p_i, a, Z_2) = (p_{m(i,2)}, \gamma_{i2}).$$

(The notation $\delta(p, a, Z) = (q, \gamma)$ means that P in state p, scanning a on the input tape, and scanning Z on the pushdown store, enters next state q, moves the input head to the next symbol, and replaces the symbol on top of the store by the (possibly empty) string γ.) If $x = a_1 a_2 \cdots a_n$, where each a_i is an individual input symbol, then define $B(x)$ to be the string $B(a_1)\, B(a_2) \cdots B(a_n)$. It is claimed that x is accepted by P if and only if $Z_1 B(x) \#$ is in $L_0^{(3)}$.

If $x = a_1 a_2 \cdots a_n$ is accepted by P, then there exists a sequence of n steps that cause P to move from its initial configuration on input x to a configuration in which the pushdown store is empty and the state of P is p_1 (the initial and final state). Let the configuration entered after l steps be denoted by $(p_{r(l)}, Z_1^{(l)} Z_2^{(l)} \cdots Z_{s(l)}^{(l)})$. We show by induction on $l \ge 0$ that there is a sequence $t_1, t_2, \ldots, t_{nk}$ such that

$$Z_1 B(x) \#[t_1, t_2, \ldots, t_{lk}] \in D_3 Z_1^{(l)} D_3 Z_2^{(l)} \cdots D_3 Z_{s(l)}^{(l)} Z_0^{r(l)-1} \quad \text{for all } 0 \le l \le n. \qquad (**)$$

(That is, after "canceling" symbols, $Z_1 B(x) \#[t_1, t_2, \ldots, t_{lk}]$ is the string in the pushdown store of P after l steps followed by $r(l) - 1$ occurrences of Z_0 (which corresponds to the state $p_{r(l)}$ of P after l steps).) For $l = 0$ the result is easily verified, since for a sequence ξ of length zero $Z_1 B(x) \#[\xi]$ is Z_1 and P starts in state p_1 with Z_1 in the pushdown store. We assume next the validity of the statement for $l = m$ and show it is true for $m + 1$. Thus we are assuming that a sequence $t_1, t_2, \ldots, t_{mk}$ has been constructed so that (**) is true for $l = m$. One can then (1) select the symbol $\bar{Z}_0$ from the next $r(m)$-1 blocks of $B(x)$, (2) select the string $\bar{Z}_{s(m)}^{(m)} \gamma Z_0^{k+r(m+1)-r(m)-1}$ from the next block (which corresponds to the transition $\delta(p_{r(m)}, a_m, Z_{s(m)}^{(m)}) = (p_{r(m+1)}, \gamma)$), and (3) select the symbol $\bar{Z}_0$ from the next $k - r(m)$ blocks of $B(x)$. In so doing one constructs a sequence $t_1, t_2, \ldots, t_{(m+1)k}$ such that statement (**) is true for $l = m + 1$. For $l = n$, it follows that $Z_1 B(x) \#[t_1, t_2, \ldots, t_{nk}] \in D_3$, since P has an empty pushdown store and has reentered state p_1 after n steps. Therefore, if x is accepted by P, then $Z_2 B(x) \#$ is in $L_0^{(3)}$.

In an analogous manner it can be shown that if $Z_1 B(x) \#$ is in $L_0^{(3)}$, then x is recognized by P.

It is straightforward to observe that the transformation from x into $Z_1 B(x) \#$ can be done in $\log(|x|)$ space. In fact, there is a homomorphism that will transform x into $B(x)$. (The homomorphism simply maps a symbol a into $B(a)$.) Thus, the transformation does not require any additional workspace. □

The family of simple precedence languages, defined originally by Wirth and Weber [28], is known to be a proper subfamily of the family of deterministic context-free languages (see Fischer [9]). Are the simple precedence languages as difficult to recognize as arbitrary

deterministic context-free languages? Likewise, the family of simple LL(1) languages is a proper subfamily of DCFL [9]. Is it as difficult to recognize as DCFL? We provide an affirmative answer to these questions by describing a language which is simultaneously simple precedence and simple LL(1) and is $\log(n)$-complete for DCFL. We refer the reader to an excellent discussion of precedence grammars and simple LL(1) grammars in [2] for the necessary definitions and fundamental properties.

Definition. Let $\Sigma = \{Z_0, Z_1, \bar{Z}_0, \bar{Z}_1\}$ and let c and d be two distinct symbols. Define the substitution mapping[2] from $P(\Sigma^*)$ to $P((\Sigma \cup \{c, d\})^*)$ by.

$$\begin{aligned}
&\sigma(Z_0) = \{Z_0\},\\
&\sigma(Z_1) = \{Z_1\},\\
&\sigma(\bar{Z}_0) = (\{c\} \cup \{dxc \mid x \in \Sigma^*\})^* \{\bar{Z}_0\},\\
&\sigma(\bar{Z}_1) = (\{cxd \mid x \in \Sigma^*\} \cup \{dx_1cx_2d \mid x_1, x_2 \in \Sigma^*\})^* \{\bar{Z}_1\}.
\end{aligned}$$

For example, $Z_0Z_1Z_0cd\bar{Z}_0Z_1c\bar{Z}_0d\bar{Z}_1Z_0c\bar{Z}_0Z_1d\bar{Z}_1\bar{Z}_0$ is in $Z_0Z_1Z_0\sigma(\bar{Z}_0)\sigma(\bar{Z}_1)\bar{Z}_0 \subseteq \sigma(Z_0D_2\bar{Z}_0)$. Intuitively, the last "uncanceled" symbol in a prefix of a string in $\sigma(Z_0D_2\bar{Z}_0)$ provides the necessary information about where next to choose a string of symbols to form a string in $Z_0D_2\bar{Z}_0$. If the last uncanceled symbol is Z_0, then the next string of symbols may be chosen after an occurrence of the symbol c to the right. If the last uncanceled symbol is Z_1, then the next string of symbols may be chosen after an occurrence of d to the right.

LEMMA 9. $L_0^{(3)} \leq_{log} \sigma(Z_0D_2\bar{Z}_0)$.

PROOF. We show first that $L_0^{(3)} \leq_{\log} L_0^{(2)}$. For this define a homomorphism e from $\{Z_0, Z_1, Z_2\}^*$ to $\{Z_0, Z_1\}^*$ by: $e(Z_0) = Z_0$, $e(Z_1) = Z_0Z_1$, $e(Z_2) = Z_0Z_1Z_1$. Transform any block B of the form $[\bar{Z}_0\gamma_1\#\bar{Z}_1\gamma_2\#\bar{Z}_2\gamma_3]$ into the sequence of blocks $E(B) = [\bar{Z}_0e(\gamma_1)Z_1\#\bar{Z}_1]$ $[\bar{Z}_0e(\gamma_2)Z_1\#\bar{Z}_1][\bar{Z}_0e(\gamma_3)\#\bar{Z}_1]$. Then $\gamma_0B_1B_2 \cdots B_n\#$ is in $L_0^{(3)}$ if and only if $e(\gamma_0)E(B_1)E(B_2) \cdots E(B_n)\#$ is in $L_0^{(2)}$.

Then for any string x in $L_0^{(2)}$ of the form $\gamma_0[\bar{Z}_0\gamma_0^{(1)}\#\bar{Z}_1\gamma_1^{(1)}][\bar{Z}_0\gamma_0^{(2)}\#\bar{Z}_1\gamma_1^{(2)}] \cdots [\bar{Z}_0\gamma_0^{(k)}\#\bar{Z}_1\gamma_1^{(k)}]\#$, let $f(x) = Z_0\gamma_0c\bar{Z}_0\gamma_0^{(1)}d\bar{Z}_1\gamma_1^{(1)}c\bar{Z}_0\gamma_0^{(2)}d\bar{Z}_1\gamma_1^{(2)}c \cdots c\bar{Z}_0\gamma_0^{(k)}d\bar{Z}_1\gamma_1^{(k)}c\bar{Z}_0$. (The transformation essentially maps occurrences of the symbol # to d, bracket symbols to the symbol c, strings over Σ to themselves, and places a Z_0 in front and a $\bar{Z}_0$ behind the resulting string.) It is straightforward to verify that x is in $L_0^{(2)}$ if and only if $f(x)$ is in $\sigma(Z_0D_2\bar{Z}_0)$. Clearly, $f(x)$ is computable in $\log(|x|)$ space. □

It should be noted that although $f(x)$, in the proof of the previous lemma, is very similar to x the language $\sigma(Z_0D_2\bar{Z}_0)$ differs considerably from $L_0^{(2)}$. These differences are desirable in the following sense. Although it seems to be very difficult to construct a simple precedence grammar or a simple LL(1) grammar for $L_0^{(2)}$, it is not so difficult to construct such grammars for $\sigma(Z_0D_2\bar{Z}_0)$.

LEMMA 10. *$\sigma(Z_0D_2\bar{Z}_0)$ is a simple LL(1) language.*

PROOF. Consider first the following grammar for the language $Z_0D_2\bar{Z}_0$:

$$\begin{aligned}
&S \rightarrow Z_0A\\
&A \rightarrow \bar{Z}_0 \mid Z_0AA \mid Z_1BA\\
&B \rightarrow \bar{Z}_1 \mid Z_0AB \mid Z_1BB
\end{aligned}$$

This grammar can be modified in a straightforward manner to obtain one for $\sigma(Z_0D_2\bar{Z}_0)$, such as the abbreviated grammar described below.

$$\begin{aligned}
&S \rightarrow Z_0A\\
&A \rightarrow T_1\bar{Z}_0 \mid Z_0AA \mid Z_1BA\\
&B \rightarrow T_2\bar{Z}_1 \mid Z_0AB \mid Z_1BB
\end{aligned}$$

where T_1 is a nonterminal which generates the regular set $(\{c\} \cup \{dxc \mid x \in \Sigma^*\})^*$ and T_2 is a nonterminal which generates the regular set $(\{cxd \mid x \in \Sigma^*\} \cup \{dx_1cx_2d \mid x_1, x_2 \in$

[2] For any set A, let $P(A)$ denote the set of all subsets of A. For each element a of an alphabet Σ, let $\sigma(a) \in P(\Sigma^*)$. Let $\sigma(e) = e$ and $\sigma(a_1a_2 \cdots a_r) = \sigma(a_1)\sigma(a_2) \cdots \sigma(a_r)$. Then the function σ is called a *substitution mapping*. For any set $L \subseteq \Sigma^*$, let $\sigma(L) = \bigcup_{x \in L}\sigma(x)$.

$\Sigma^*\})^*$. Finally, this abbreviated grammar can be expanded into an equivalent simple LL(1) grammar as follows:

$$\begin{aligned}
S &\to Z_0T_A \\
T_A &\to \bar{Z}_0|Z_0T_AT_A|Z_1T_BT_A|cA|dB \\
T_B &\to \bar{Z}_1|Z_0T_AT_B|Z_1T_BT_B|cC|dE \\
A &\to cA|dB|\bar{Z}_0 \\
B &\to Z_0B|Z_1B|\bar{Z}_0B|\bar{Z}_1B|cA \\
C &\to Z_0C|Z_1C|\bar{Z}_0C|\bar{Z}_1C|dD \\
D &\to \bar{Z}_1|cC|dE \\
E &\to Z_0E|Z_1E|\bar{Z}_0E|\bar{Z}_1E|cF \\
F &\to Z_0F|Z_1F|\bar{Z}_0F|\bar{Z}_1F|dD
\end{aligned}$$

(The reader should observe that the nonterminals T_A and T_B in this last grammar generate the same set of terminal strings as the nonterminals A and B respectively, in the previous grammar. Also, the nonterminals A, B, C, D, E, and F have been added to generate the regular sets mentioned previously.) Thus, $\sigma(Z_0D_2\bar{Z}_0)$ is a simple LL(1) language. □

LEMMA 11. *$\sigma(Z_0D_2\bar{Z}_0)$ is a simple precedence language.*

PROOF. Consider initially the following grammar for the language $Z_0D_2\bar{Z}_0$:

$$\begin{aligned}
S &\to A\bar{Z}_0 \\
A &\to Z_0|AA\bar{Z}_0|AB\bar{Z}_1 \\
B &\to Z_1|BA\bar{Z}_0|BB\bar{Z}_1
\end{aligned}$$

This grammar can be modified in a straightforward manner to obtain one for $\sigma(Z_0D_2\bar{Z}_0)$, such as the abbreviated grammar described below:

$$\begin{aligned}
S &\to AT_1\bar{Z}_0 \\
A &\to Z_0|AAT_1\bar{Z}_0|ABT_2\bar{Z}_1 \\
B &\to Z_1|BAT_1\bar{Z}_0|BBT_2\bar{Z}_1
\end{aligned}$$

where T_1 is a nonterminal which generates the regular set $(\{c\} \cup \{dxc|x \in \Sigma^*\})^*$ and T_2 is a nonterminal which generates the regular set $(\{cxd|x \in \Sigma^*\} \cup \{dx_1cx_2d|x_1, x_2 \in \Sigma^*\})$.* Finally, this abbreviated grammar can be expanded into an equivalent simple precedence grammar as follows:

$$\begin{aligned}
S &\to T'_A\bar{Z}_0 \\
T'_A &\to T_A \\
T'_B &\to T_B \\
A &\to Z_0|AT'_A\bar{Z}_0|AT'_B\bar{Z}_1 \\
B &\to Z_1|BT'_A\bar{Z}_0|BT'_B\bar{Z}_1 \\
T_A &\to T_Ac|T_1c|A \\
T_B &\to T_2d|T_3d|B \\
T_1 &\to T_1Z_0|T_1Z_1|T_1\bar{Z}_0|T_1\bar{Z}_1|T_Ad \\
T_2 &\to T_2Z_0|T_2Z_1|T_2\bar{Z}_0|T_2\bar{Z}_1|T_Bc \\
T_3 &\to T_3Z_0|T_3Z_1|T_3\bar{Z}_0|T_3\bar{Z}_1|T_4c \\
T_4 &\to T_4Z_0|T_4Z_1|T_4\bar{Z}_0|T_4\bar{Z}_1|T_Bd
\end{aligned}$$

(The reader should observe that the nonterminals T'_A and T'_B in this last grammar generate the same set of terminal strings as AT_1 and BT_2, respectively, in the previous grammar. Also, the nonterminals T_1, T_2, T_3, and T_4 are used to generate the regular sets mentioned above. It is left to the reader to construct the table of precedence relations and to verify that the relations $<$, $\doteq$, and $>$ defined by this last grammar are disjoint.) Thus, $\sigma(Z_0D_2\bar{Z}_0)$ is a simple precedence language. □

It is not known whether or not there is an operator precedence grammar (see Floyd [10] or Aho and Ullman [2]) which generates a language L′ such that L′ is $\log(n)$-complete for DCFL. It is also not known whether the family of operator precedence languages is contained in DSPACE($\log(n)$) [NSPACE($\log(n)$)]. The following operator precedence grammar generates the language $\tau(D_2)$, where τ is the substitution defined by $\tau(Z_0) = Z_0$, $\tau(Z_1) = Z_1$, $\tau(\bar{Z}_0) = \bar{Z}_1^*\bar{Z}_0$, and $\tau(\bar{Z}_1) = \bar{Z}_0^*\bar{Z}_1$:

$$\begin{aligned}
S &\to Z_0\bar{Z}_0|Z_1\bar{Z}_1|Z_0\bar{Z}_0S|Z_1\bar{Z}_1S|Z_0T\bar{Z}_0|Z_1U\bar{Z}_1|Z_0T\bar{Z}_0S|Z_1U\bar{Z}_1S \\
T &\to T\bar{Z}_1|\bar{Z}_1|S \\
U &\to U\bar{Z}_0|\bar{Z}_0|S
\end{aligned}$$

It is conjectured that $\tau(D_2)$ is an example of an operator precedence language not in NSPACE($\log(n)$). (The reader may verify that the above grammar is an operator grammar

and that disjoint relations $<$, $=$, and $>$ are defined on the terminal symbols.)

In conclusion we have established the following:

THEOREM 2. *The following statements are equivalent:*

(1) *The language $L_0^{(2)}$ is in DSPACE(log(n)).*

(2) *The simple LL(1) languages are contained in DSPACE(log(n)).*

(3) *The simple precedence languages are contained in DSPACE(log(n)).*

(4) *The family of languages recognized by deterministic log(n)-tape bounded auxiliary pushdown automata in polynomial time is identical to DSPACE(log(n)).*

That is, $L_0^{(2)}$ is a space or tape hardest deterministic context-free language.[3] It follows from [18, 24] that $L_0^{(2)}$ can be recognized in DSPACE$((\log(n))^d)$, for any $1 \leq d < 2$, if and only if DCFL $\subseteq$ DSPACE$((\log(n))^d)$. The hardest context-free language has been described by Greibach [14]. (Greibach's language is hardest with respect to both time and tape complexity measures. Since all deterministic context-free languages are linear time recognizable, they are all equally hard with respect to the time complexity measures.)

An interesting open question is whether or not there is a language in DCFL which is $\log(n)$-complete for NSPACE(log n) or for CFL. It follows from the results presented that if a language in DCFL exists to which all languages in NSPACE(log n) or CFL can be reduced within $\log(n)$ space, then $L_0^{(2)}$ is one such language. Furthermore, such a language exists if and only if NSPACE(log n) $\subseteq$ LOG(DCFL) or LOG(CFL) = LOG(DCFL). Whether or not LOG(CFL) = LOG(DCFL) is equivalent, by what has been shown, to the question of whether nondeterminism adds power to polynomial time bounded and log tape bounded auxiliary PDAs, i.e. a P $\neq$ NP question for automata of this kind. Whether or not LOG(DCFL) $\subseteq$ NSPACE(log n) is also an interesting question that, as yet, remains open.

REFERENCES

(Note References [6, 25, 27] are not cited in the text.)

1 AHO, A V, HOPCROFT, J E, AND ULLMAN, J D Time and tape complexity of pushdown automaton languages *Inform and Control 13,* 3 (1968), 186–206

2 AHO, A V, AND ULLMAN, J D *The Theory of Parsing, Translation, and Compiling, Vol I Parsing* Prentice-Hall, Englewood Cliffs, N J, 1972

3 BOOK, R V Translational lemmas, polynomial time, and $(\log n)^j$-space *Theoret Comptr Sci 1* (1976), 215–226

4 COOK, S A Path systems and language recognition Proc Second Annual ACM Symp Theory of Computing, 1970, pp. 70–72

5 COOK, S A. The complexity of theorem proving procedures Proc Third Annual ACM Symp. Theory of Computing, 1971, pp 151–158

6. COOK, S A Linear time simulation of deterministic two-way pushdown automata Information Processing 71, Vol 1, North-Holland Pub Co, Amsterdam, pp 75–80

7 COOK, S A Characterizations of pushdown machines in terms of time-bounded computers *J ACM 18,* 1 (Jan 1971), 4–18

8 COOK, S A An observation on time-storage trade off *J Comptr Syst Sci 9* (1974), 308–316

9 FISCHER, M.J Some properties of precedence languages Proc First Annual ACM Symp Theory of Computing, 1969, pp 181–190

10 FLOYD, R W Syntactic analysis and operator precedence *J ACM 10,* 3 (July 1963), 316–333

11 GALIL, Z Two-way deterministic pushdown automata and some open problems in the theory of computation Proc Fifteenth Annual IEEE Symp Switching and Automata Theory, 1974, pp 170–177

12 GINSBURG, S, AND GREIBACH, S A Deterministic context-free languages *Inform and Control 9,* 6 (1966), 620–648

13 GRAY, J N, HARRISON, M A, AND IBARRA, O H. Two-way pushdown automata *Inform and Control 11,* 1 (1967), 30–70

14 GREIBACH, S A The hardest context-free language *SIAM J Computing 2,* 4 (1973), 304–310

15 HARRISON, M A, AND IBARRA, O H Multitape and multihead pushdown automata *Inform and Control 13,* 5 (1968), 433–470

16. HARTMANIS, J On nondeterminacy in simple computing devices *Acta Informatica 1* (1972), 336–344

17 IBARRA, O H On two-way multihead automata *J Comptr Syst Sci 7* (1973), 28–36

[3] It should be noted that $L_0^{(2)}$ is $\log(n)$-complete for the family DCFL by Lemma 8 and the proof of Lemma 9

18 JONES, N D. Space-bounded reducibility among combinatorial problems *J Comptr Syst Sci 11* (1975), 68–85.
19. JONES, N.D , AND LAASER, W.T Complete problems for deterministic polynomial time *Theoret Comptr Sci. 3* (1977), 105–117.
20. KARP, R M Reducibility among combinatorial problems In *Complexity of Computer Computations,* R.E Miller and J N Thatcher, Eds , Plenum Press, N Y , 1972, pp. 85–104.
21 KNUTH, D E On the translation of languages from left to right *Inform and Control 8*, 6 (1965), 607–639
22. KORENJAK, A J , AND HOPCROFT, J E Simple deterministic languages Conf Rec Seventh Annual IEEE Symp Switching and Automata Theory, 1966, pp 36–46
23 LEWIS, P M , STEARNS, R E , AND HARTMANIS, J Memory bounds for the recognition of context-free and context-sensitive languages Proc Sixth Annual IEEE Symp Switching Circuit Theory and Logical Design, 1965, pp 191–212
24 MEYER, A R , AND STOCKMEYER, L J Word problems requiring exponential time Proc Fifth Annual ACM Symp Theory of Comptng , 1973, pp 1–9
25 SAVITCH, W J Relationships between nondeterministic and deterministic tape complexities *J Comptr and Syst Sci 4* (1970), 177–192
26 SUDBOROUGH, I H On tape-bounded complexity classes and multihead finite automata *J Comptr and Syst Sci 10,* 1 (1975), 62–76
·27 SUDBOROUGH, I H On deterministic context-free languages, multihead automata, and the power of an auxiliary pushdown store Proc Eighth Annual ACM Symp Theory of Comptng , 1976, pp 141–148
28 WIRTH, N , AND WEBER, H EULER—A generalization of ALGOL and its formal definition, Pts 1, 2 *Comm. ACM 9,* 1 (1966), 13–23, and 9, 2 (1966), 89–99

RECEIVED JULY 1975, REVISED OCTOBER 1977