THE TIME AND TAPE COMPLEXITY OF DEVELOPMENTAL LANGUAGES (†)

by

I. H. Sudborough
Department of Computer Sciences
The Technological Institute
Northwestern University
Evanston, Illinois 60201

Abstract

The following results are established:

(1) EDOL $\subseteq$ DSPACE (log n)

(2) EOL $\subseteq$ DSPACE ((log n)^2)

(3)* EDTOL $\subseteq$ NSPACE (log n)

(4) EDTOL $\subseteq$ DSPACE (log n) if and only if NSPACE (log n) $\subseteq$ DSPACE (log n)

Statement (4) follows from statement (3) above, the fact that all linear context-free languages are EDTOL languages [21], and the existence of a linear context-free language which is log-tape complete for NSPACE (log n) [15]. Furthermore, it is shown that all EOL languages are log-tape reducible to context-free languages. Hence, EOL $\subseteq$ DSPACE (log n) if and only if every context-free language is in DSPACE (log n).

Introduction

In [1] the author has shown that the set of languages which are log-tape reducible to context-free languages (deterministic context-free languages) is identical to the set of languages recognized in polynomial time by non-deterministic (deterministic) log(n)-tape bounded auxiliary pushdown automata. This characterization enables one to show that many families which properly contain the context-free

(†) This work is supported by NSF Grant GJ-43228

(*) This result has been independently obtained by Tero Harju [18] and Neil Jones and Sven Skyum [20] using essentially the same algorithm as stated here. In fact, the author has recently become aware of an earlier paper by Harju [19] again with essentially the same algorithm, which demonstrates that EDTOL $\subseteq \bigcup_{k>0}$ DTIME (n^k).

languages have the same tape complexity as the family of context-free languages [2]. It is known that context-free languages can be recognized in $(\log n)^2$ space deterministically [3]. It is not known if this result is optimal. Greibach has described a context-free language whose time or tape complexity is the least upper bound on the complexity of all CFL's [4]. Moreover, it is known that if all CFL's can be recognized in deterministic log space, then nondeterministic L(n)-tape complexity classes are identical to deterministic L(n)-tape complexity classes, for $L(n) \geq \log n$ [5].

In this paper we consider the tape and time complexity of developmental languages [6]. It is shown that every EOL language is log tape reducible to a context-free language. Thus, EOL languages can be recognized in $(\log n)^2$ space. In fact, the algorithm described in [3] for CFL's is the basic component of our EOL algorithm. It is shown that the EDOL languages are recognizable in log n space (and in $O(n^2)$ time). It is also shown that EDTOL languages can be recognized in log n space nondeterministically. As a corollary we obtain Harju's earlier result [19] that every EDTOL language is recognizable deterministically in polynomial time. Furthermore, since every linear CFL is an EDTOL language [21], we obtain that EDTOL $\subseteq$ DSPACE(log n) if and only if NSPACE(L(n)) = DSPACE(L(n)), for all $L(n) \geq \log n$ [15]. This work was principally motivated by the results of Van Leeuwen which show that EOL $\subseteq$ DSPACE($(\log n)^3$) [7] and that the membership problem for ETOL is NP- complete [8].

We shall employ the following terminology in the remainder of this paper. Let Σ be an alphabet of symbols. Σ^* will denote the set of all sentences of strings over the alphabet Σ. The string consisting of zero symbols, called the empty string, is denoted by e. The number of symbols in a string x, called the length of x, is denoted by $|x|$. The reversal of a string x is denoted by x^R.

We shall assume that the reader is familiar with the basic concepts of formal language theory and computational complexity as contained, for example, in [9]. Let DSPACE(L(n)) and NSPACE(L(n)) denote the families of languages recognized by deterministic (nondeterministic) L(n)- tape bounded off-line Turing machines, respectively. Let P and NP denote the families of languages recognized by deterministic (nondeterministic) multitape Turing machines in polynomial time. We shall employ the concept of a log tape reduction as discussed, for example, in [10, 11].

Definition. Let Σ and Δ be alphabets and f be a function from Σ^* to Δ^*. f is log-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 after having written f(x) in Δ^* on its output tape 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 strings. A is log-tape reducible to B, denoted by $A \leq_{\log} B$, if there is a log-tape computable function f such that, for all x in Σ^*, x is in A if and only if f(x) is in B.

The following lemmas are from [10, 11].

Lemma. $\leq_{\log}$ is transitive

Lemma. Let A and B be sets of strings. If $A \leq_{\log} B$ and B is in DSPACE $((\log n)^k)$, for $k \geq 1$, then A is in DSPACE$((\log n)^k)$.

Definition. For any family of languages $\mathcal{L}$ let LOG($\mathcal{L}$) denote the set of all languages which are log-tape reducible to some language in $\mathcal{L}$.

In [1] the families LOG(CFL) and LOG(DCFL), where CFL (DCFL) denotes the family of context-free languages (deterministic context-free languages), have been characterized as the families of languages recognized by nondeterministic and deterministic log(n)- tape bounded auxiliary pushdown automata in polynomial time, respectively. L(n)- tape bounded auxiliary pushdown automata were first considered in [12]. It was shown there that nondeterministic and deterministic L(n)- tape bounded auxiliary pushdown automata recognize the same family of languages, for $L(n) \geq \log n$. However, if we restrict these families by insisting that each string accepted must be accepted in some number of steps bounded by a fixed polynomial in the length of that string, then the equivalence of nondeterministic and deterministic families is an open question. Also, it is well known that the family of languages recognized by log(n)- tape bounded auxiliary pushdown automata is identical to the family of languages recognized by two-way multihead pushdown automata. We shall use both type of machine models interchangeably.

Let $APDA_p(\log n)$ ($ADPDA_p(\log n)$) denote the family of languages recognized by nondeterministic (deterministic) polynomial time bounded and log(n)- tape bounded auxiliary pushdown automata, respectively. Thus, $LOG(CFL) = APDA_p(\log n)$ and $LOG(DCFL) = ADPDA_p(\log n)$. Since $CFL \subseteq DSPACE((\log n)^2)$ [3], it follows that $LOG(CFL) \subseteq DSPACE((\log n)^2)$. In [12] Cook has shown that P is identical to the family of languages recognized by log(n)- tape bounded auxiliary pushdown automata (with no time restriction). Thus, $LOG(CFL) = APDA_p(\log n) \subseteq P$. A language L_0 is log-tape complete for a family $\mathcal{L}$ if (1) L_0 is in $\mathcal{L}$, and (2) for all L in $\mathcal{L}$, $L \leq_{\log} L_0$. It follows that if a language L_0 is log-tape complete for NSPACE(log n), then L_0 is in DSPACE(log n) if and only if $NSPACE(L(n)) = DSPACE(L(n))$, for all $L(n) \geq \log n$ [13].

Context-Independent Developmental Languages

In [7] Van Leeuwen described an algorithm to recognize any EOL language and demonstrated that it could be implemented within $(\log n)^3$ space on a deterministic Turing machine. It is shown here that the family of EOL languages is contained in LOG(CFL). Therefore, the EOL languages are essentially of the same tape complexity as context-free languages and the algorithm given by Lewis, Stearns, and Hartmanis in [3], which requires $(\log n)^2$ space on a deterministic Turing machine, is also applicable to the family of EOL languages.

The reader is referred to [6] for motivation and background information relevant to the study of developmental systems. The basic idea is to describe a mathematical model of the growth patterns of simple cellular organisms. For brevity, and to avoid repetition, we shall describe here only the formal definitions of the grammars and languages under consideration. We follow basically the approach described in [7].

Definition An EOL-grammar is a four-tuple $G = (V, \Sigma, \delta, S)$, where V is a finite set (of symbols), $\Sigma \subseteq V$ (a set of terminal symbols), S is an element of V (the initial symbol), and δ is a function which maps elements of V to finite subsets of V^* (the set of productions).

If $\delta(A) = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, then we say that the symbol A can in one step become any one of the strings of symbols α_i, for $1 \le i \le n$. We may extend δ to map V^* into V^*, in the customary way, by $\delta(e)=e, \delta(\beta A) = \delta(\beta)\delta(A)$, for $\beta \in V^*$ and $A \in V$. For $n \ge 0$, let δ^n denote the composition of δ with itself n times. (Note that δ^0 is considered to be the identify function on V^* and $\delta^1 = \delta$.)

Instead of $\alpha \in \delta(A)$ we shall often write $A \to \alpha$ and call such an item a production. For x and y in V^* we shall often write $x \Rightarrow y$ for $y \in \delta(x)$. Let $\overset{*}{\Rightarrow}$ be the transitive, reflexive closure of $\Rightarrow$. (In this vein the reader will note that an EOL-grammar is a context-free grammar in which the usual notion of "β derives γ", i.e. $\beta \Rightarrow \gamma$, has been altered. In a context-free grammar β derives γ means one of the symbols in β, say A, has been replaced by the right side of a production of the form $A \to \alpha$ to obtain γ. In an EOL-grammar β derives γ means all of the symbols in β have been replaced by the right hand side of a production involving that symbol.)

<u>Definition</u> An <u>EDOL-grammar</u> is an EOL-grammar $G = (V, \Sigma, \delta, S)$ such that, for all $a \in V$, $\delta(a)$ contains exactly one element.

<u>Definition</u> Let $G = (V, \Sigma, \delta, S)$ be an EOL-grammar. The <u>language generated by G</u>, denoted by L(G), is:

$$L(G) = \bigcup_{n \ge 0} \delta^n(S) \cap \Sigma^* = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

A language L is called an <u>EOL-language</u> if there exists an EOL-grammar G such that L = L(G). A language L is called an <u>EDOL-language</u> if there exists an EDOL-grammar G such that L = L(G). The family of context-free languages is properly contained in the family of EOL-languages [6].

To show that every EOL-language is in LOG(CFL) we describe an algorithm to recognize EOL-languages which may be implemented on a log(n)-tape bounded auxiliary pushdown automaton that operates in polynomial time. The algorithm is basically similar to the usual algorithm for recognizing context-free languages using a nondeterministic pushdown store automaton (see, for example, pages 176-77 of [14]). However, the derivation tree corresponding to a derivation by an EOL-grammar must satisfy the property that all paths from the root of the tree to a leaf are of the same length. This additional requirement can be checked by adding to the symbols stored in the pushdown store, during the usual context-free language algorithm, an integer which denotes the level in the corresponding derivation tree at which the symbol resides. Thus, if the symbol A on top of the store is at level i then the pushdown store will contain a representation of the pair (A,i). If $A \to BC$ is a production in the grammar, then (A,i) will be removed from the top of the store and (B,i+1) and (C,i+1) will be added to the store. For some particular value k, which denotes the length from the root of the derivation tree to each leaf, each pair of the form (A,k) as it appears on top of the store will initiate a verification that the next input symbol is A and will then be deleted from the store. By Lemma 3.1, page 207, of [7], for each word x generated by an EOL-grammar there is a derivation tree such that the distance from the root to each leaf is some integer k less than or equal to $c|x|+1$.

The pair (A,i) consisting of the symbol A and the integer i will be represented by the string [A * N(i)], where [,*, and] are new symbols not occurring in the symbol alphabet of the EOL-grammar, and N(i) is the binary representation of i. In order to perform the replacement of the pair (A,i) by the pairs (B,i+1) and (C,i+1) indicated in the production $A \to BC$ we shall need to use the worktape space of the auxiliary pushdown automaton. The algorithm is incorporated into the proof of the following theorem:

<u>Theorem</u> Every EOL-language is in LOG(CFL).

<u>Proof</u>. Let $G = (V,\Sigma,\delta,S)$ be an EOL-grammar. Let M be the nondeterministic log(n)-tape bounded auxiliary pushdown automaton which performs the following steps on an input string w of length n:

(1) M nondeterministically writes an integer k ($0 \le k \le c|w|+1$) on its worktape. (k denotes the length of each path from the root to a leaf in a derivation tree for w.)

(2) M places the string consisting of the representation of the pair (S,0) and the special symbol # on its pushdown store. (The symbol # is on the bottom of the store.)

(3) For $0 \le i < k$, if the representation of (A,i) is the top string on the pushdown store, then M may replace that string with the representations of $(B_1,i+1)$, $(B_2,i+1)$, ... ,$(B_m,i+1)$, if $B_1B_2\ldots B_m$ is an element of $\delta(A)$.

(4) If the representation of (A,k) is the top string on the pushdown store, then M determines whether A is the current input symbol or not and whether A is in Σ or not. If A is not the current symbol or A is not in Σ, then M stops without accepting. Otherwise, M deletes the representation of (A,k) from the store, moves the input head right, and executes step (5).

(5) If the top symbol on the pushdown store is not #, then M executes steps (3)-(5) again. If the top symbol on the store is # but the input head is not scanning the right endmarker, then M stops without accepting. Otherwise, if the top symbol is # and the input head is scanning the right endmarker, then M stops and accepts.

M can execute step (3) or (4) by (1) writing the representation [A*N(i)] for the pair (A,i) on its worktape, (2) determining whether or not N(i) is equal to N(k) (which is stored permanently on the worktape), and if $i < k$, then (3) incrementing i to obtain N(i+1) and writing sequentially the representations for $(B_m,i+1)$, ..., $(B_1,i+1)$, as specified in step (3), on the pushdown store. Clearly, this process can be performed using only $d \cdot \log(|w|)$ cells on the worktape, for some constant $d > 0$, since $k < c|w|=1$. Moreover, M is polynomial time bounded, since M executes at most $c_1 k$ steps between movements of the input head, for some $c_1 > 0$. (That is, after each execution of step (3) the level number in the top pair on the pushdown store is increased by one and when the level number in the top pair becomes k the input head moves right or M stops.) Therefore, M operates in time $O(n^2)$. That M accepts w if, and only if, w is in L(G) can easily be verified.

Therefore, L(G) is in LOG(CFL) by Theorem 2.1. $\square$

It should be noted that the use of the index i in the symbols (A,i) stored in the pushdown store of the preceding algorithm is mainly for convenience. That is, the level of a symbol in a derivation tree could be determined from the symbol's position in the pushdown store. Thus, we need only use the worktape space to record the fixed path length k and to count the appropriate level number of a sumbol in the pushdown store. It follows that the algorithm we have described can be viewed as an implementation of a pre-set pushdown automaton [22]on an auxiliary pushdown automaton. It follows that every pre-set pushdown automaton language is in LOG(CFL). It is known that the family of pre-set pushdown automata languages properly contains the family of EOL languages [22].

For EDOL languages we can obtain a better result. That is, every EDOL-language can be recognized by a deterministic log(n)-tape bounded Turing machine. The basic idea, in this case, is that for each string $w \in V^*$ from an EDOL-grammar G $= (V,\Sigma,\delta,S)$ and each integer $i, \delta^i(w)$ is a single string in V^* and the length of that string can be easily obtained. In fact, the length of that string can be ascertained in an amount of space bounded by the logarithm of max $\{|\delta^j(w)| \mid 0 \leq j \leq i\}$. Such an algorithm, denoted by LENGTH(w,i), is described below: (Let $\#_a(w)$ denote the number of occurrences of the symbol a in the string w)

LENGTH(w,i)

(1) If i=0, then set LENGTH to $|w|$. Otherwise, for each $a \in V$, set NUMBER (a) = $\#_a(w)$;

(2) For each $a \in V$, set NUMBER (a) $= \sum_{b \in V} [\text{NUMBER (b)} \times \#_a\delta(b)]$

(3) Set i to i-1.

(4) If i = 0, then set LENGTH $= \sum_{a \in V}$ NUMBER(a) and stop. Otherwise, re-execute step (2).

For example, if $\delta(a)$ = bab and $\delta(b)$ = abaa, then LENGTH(abb,2) = 37. Since the above algorithm need only represent, say in binary notation, a finite set of integers, namely NUMBER(a), for each $a \in V$, each of which is never larger than m = max $\{|\delta^j(w)| \mid 0 \leq j \leq i\}$, the space needed for execution is bounded by $c \cdot \log_2 m$, for some $c > 0$ dependent only upon the grammar G.

Let $G = (V,\Sigma,\delta S)$ be an EDOL grammar. As in [24], we shall say that a symbol $a \in V$ is <u>mortal</u> if $\delta^i(a) = e$, for some $i > o$. Those symbols in V which are not mortal will be called <u>vital</u>. Let $||w||$ denote the number of occurrences of vital symbols in the string w. We observe that there is a constant $c > 0$, depending only upon G, such that for all w in V* and $i \geq o$,

$$(*) \qquad |\delta^i(w)| \leq c \cdot \max \{||\delta^i(w)||, w\}.$$

In order to see this, let $p = \max (\{|\delta(a)| \mid a \in V\} \cup \{1\})$ and let q be the smallest integer such that, for all $a \in V$, if a is mortal, then $\delta^q(a) = e$. Let $c = p^q$. We separate two cases:

a) if $i < q$, then $|\delta^i(w)| \le p^i \cdot |w| \le c \cdot |w|$.

b) if $i \ge q$, then consider $y = \delta^{i-q}(w)$. Since, for all x in V^*, $\| x \| \le \|\delta(x)\|$, it follows that $\| y \| \le \| \delta^q(y) \| = \| \delta^i(w)\|$. It follows also from the definition of p that $|\delta(x)| \le p \cdot |x|$ and, hence, $|\delta^q(x)| \le p^q \cdot |x|$. Since all of the mortal symbols occuring in x will map to e in $\delta^q(x)$, we have,in fact,that $|\delta^q(x)| \le p^q \cdot \| x \|$. Therefore,

$$|\delta^i(w)| = |\delta^q(\delta^{i-q}(w))| = |\delta^q(y)| \le p^q \cdot \| y \| \le p^q \cdot \| \delta^i(w) \|.$$

Therefore, the claim in line (*) is established. That is, the length of any string of symbols generated from a string w in a deviation of $\delta^i(w)$ is bounded by a constant multiple of the length of either w or $\delta^i(w)$.

The next lemma from [23] is needed to bound the size of integers represented in the algorithm.

Lemma. Let L be generated by an EDOL grammar G. There is a constant $c > 0$ dependent only upon G such that each word w in L has a derivation of length less than or equal to $c \cdot \max\{|w|, 1\}$.

The next procedure is used to determine whether or not a symbol B is the j-th symbol in the string $\delta^i(A)$,for any symbol A in V and $i \ge 0$. It is denoted by VERIFY(B,j,A,i):

VERIFY(B,j,A,i)

(1) If $i > 0$, then go to (3);

(2) If A = B, then stop and answer "true"; if $A \ne B$, then stop and answer "false";

(3) Let $\delta(A) = A_1A_2\ldots A_p$ (where $p \ge 1$ and each A_i is in V) and let ℓ be the least integer ($1 \le \ell \le p$) such that $j \le \mathrm{LENGTH}(A_1A_2\ldots A_\ell, i-1)$. Then call VERIFY ($B, j-\mathrm{LENGTH}(A_1A_2\ldots A_{\ell-1}, i-1), A_\ell, i-1$). (If $\delta(A) = e$, then stop and answer "false".)

The above algorithm requires space only to record the four arguments and to execute the LENGTH routine described earlier. Let n be the length of $\delta^i(A)$. Then, by the previous lemma, $i \le cn$, for some $c > 0$. Since we may assume that the argument j is between 1 and n, all four arguments may be represented in $d \cdot \log_2 n$ space,for some constant d. Furthermore, by our previous analysis, since the LENGTH routine is called only to evaluate the length of $\delta^j(w)$ where w is a subword of $\delta^{i-j}(S)$, for $0 \le j \le i$, the space needed to execute this routine is bounded by $d_2 \cdot \log_2 (|\delta^i(S)|) = d_2 \cdot \log_2 n$, for some constant $d_2 > 0$.

The following algorithm, denoted by MEMBER(x), determines whether $x \ne e$ is generated by a given EDOL grammar G or not. It is an iterative algorithm which calls the previous VERIFY algorithm in succession to determine whether the i-th symbol of x is the i-th symbol of $\delta^j(S)$, for $1 \le i \le |x|$, and for increasing values of j. (One can determine whether e is generated by an EDOL grammar or not by a table look up procedure. That is, the answer may be prerecorded.)

MEMBER(x)

(Let $x = a_1a_2...a_n$, where $n \geq 1$ and a_i is in Σ, for $1 \leq i \leq n$, and let c be the constant from the previous lemma.)

(1) Set i = 0. If LENGTH (S,i) $\neq$ n, then go to (5).

(2) Set j = 1

(3) If VERIFY (a_j,j,S,i) is true, then go to (4). Otherwise, go to (5)

(4) If j = n, then stop and answer "true". Otherwise, set j = j+1 and go to (3).

(5) If i = cn, then stop and answer "false". Otherwise, set i = i+1 and go to (6).

(6) If LENGTH (S,i) = n, then go to (2). Otherwise, go to (5).

It should be clear from the previous discussion that MEMBER(x) correctly determines whether x is generated by some specific EDOL grammar or not and that it does not require more than $\log(|x|)$ space. Therefore, we have established the following theorem:

Theorem The family of EDOL languages is contained in DSPACE (log n).

EDTOL languages

In [8] Van Leeuwen has shown that the membership problem for ETOL languages is NP-complete. This implies that the ETOL languages are in P if and only if P = NP and, for some $k \geq 1$, are in DSPACE($(\log n)^k$) if and only if NP $\subseteq$ DSPACE ($(\log n)^k$) [10,11]. It is shown here that the membership problem for EDTOL languages is solvable nondeterministically in log n space. Thus, Harju's result [19] that EDTOL $\subseteq$ P follows as a corollary. Furthermore, since all linear CFL's are in EDTOL [21], and there is a linear CFL which is log-tape complete for NSPACE (log n) [15], EDTOL $\subseteq$ DSPACE (log n) if and only if NSPACE (L(n)) $\subseteq$ DSPACE (L(n)), for all $L(n) \geq \log n$.

An EDTOL-grammar differs from an EDOL-grammar in that there are a finite set of tables of productions. At any time within a derivation an arbitrary table may be selected and only productions within that table may be applied. The tables in an EDTOL-grammar must, moreover, satisfy the property that no two productions within a table have the same left side. (In other words, the tables are deterministic.)

Definition. An ETOL-grammar is a four-tuple $G = (V,\Sigma,\mathcal{P},S)$, where V is a finite set (of symbols), $\Sigma \subseteq V$ (a set of terminal symbols), $\mathcal{P}$ is a finite set $\{\delta_1,\delta_2,\ldots,\delta_k\}$, for some $k \geq 1$, of functions which map elements of V to finite subsets of V* (each δ_i is a set of productions), and S is an element of V (the start symbol). An EDTOL-grammar is an ETOL-grammar $G = (V,\Sigma,\mathcal{P},S)$ such that, for every δ in $\mathcal{P}$ and every $a \in V$, $\delta(a)$ contains exactly one element.

As in the case of an EOL grammar we may extend the functions δ in $\mathcal{P}$ to map V* into V* by specifying that $\delta(e)=e$ and $\delta(\beta A) = \delta(\beta)\ \delta(A)$ for $\beta \in V^*$ and $A \in V$. Instead of $\alpha \in \delta(A)$ we shall often write $A \rightarrow \alpha$ and call such an item a production. For x and y in V* and $\delta_i \in \mathcal{P}$ we shall often write $x \underset{i}{\Rightarrow} y$ for $y \in \delta_i(x)$. Let $\overset{*}{\Rightarrow}$ be transitive reflexive closure of the union of the relations $\underset{i}{\Rightarrow}$. We may consider $\mathcal{P}$ to be a finite set of tables of productions. At any step in a derivation some table is chosen and only productions in that table are applied.

Definition. Let $G = (V,\Sigma,\mathcal{P},S)$ be an ETOL grammar. The language generated by G, denoted by L(G), is:

$$L(G) = \{\delta_1\delta_2\cdots\delta_p(S) \mid p \geq 1\ \&\ \delta_1,\delta_2,\ldots,\delta_p \in \mathcal{P}\} \cap \Sigma^* = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

Let $G = (V,\Sigma,\mathcal{P},b_1)$ be an EDTOL grammar with $V = \{b_1,b_2,\ldots,b_m\}$. Let $x = a_1a_2\cdots a_n$ be a string over Σ of length n. Our algorithm stores initially a representation of the initial string b_1. A representation of an intermediate string α in a derivation of x will be accomplished by the array T(i), with $1 \leq i \leq m$, in our algorithm. For each $1 \leq i \leq m$, T(i) will either contain the empty set $\emptyset$ or will contain a pair of natural numbers (s_i,t_i). If $T(i) = \emptyset$, then b_i does not occur in the intermediate string α. If $T(i) = (s_i,t_i)$, then the algorithm has guessed that each b_i in the intermediate string α generates the string of length (t_i-s_i) starting at cell s_i of the input tape. (The algorithm is nondeterministic; it will verify that its guesses are correct before accepting the input string.) Since an EDTOL grammar is deterministic, each occurrence of b_i in α must generate the same string after any finite number of steps. Thus, it is sufficient to represent α by recording which variables occur within it and for each of those variables record what portion of the input string it should generate. The representation of a intermediate string β such that $\alpha \Rightarrow \beta$ and $\beta \overset{*}{\Rightarrow} x$ is accomplished by (1) guessing which table of productions is used at this step of a derivation of x and (2) updating the array information to represent δ. For example, if $b_i \to b_j b_j$ were a production in the current table and $T(i) = (s_i, t_i)$, then our algorithm will "guess" a natural number v such that $s_i \leq v \leq t_i$ and determine whether the string of length $(v-s_i)$ that begins at cell s_i of the input is identical to the string of length (t_i-v) that begins at cell v. If so, then T(j) will be set to (s_i,v) and the algorithm continues. If not, then the algorithm stops without accepting the input.

The algorithm terminates when all of the non-empty elements of the array T have been verified. That is, when it is verified that $T(i) = (s_i, t_i)$, where $t_i = s_i+1$, and b_i is the symbol (string of length one) on cell s_i of the input tape, for all $1 \leq i \leq m$. The algorithm also uses an array $T_0(i)$, $1 \leq i \leq m$, for temporary storage of new values for the array T during some steps of the process. A more complete description of the algorithm follows.

Let $G = (V,\Sigma,\mathcal{P},b_1)$ be an EDTOL grammar such that $V = \{b_1,b_2,\ldots,b_m\}$. Let $x = a_1a_2\ldots a_n$ be a string of length n over Σ. The following steps are executed:

(1) Set T(1) to the pair (1,n+1) and each of T(2),...,T(m) to ϕ.

(2) Select nondeterministically a function δ from the set $\mathcal{P}$. For $1 \leq i \leq m$, set $T_0(i) = \phi$.

(3) For each $1 \leq i \leq m$, such that $T(i) \neq \phi$, do the following:

(a) If $\delta(b_i)$ is the empty string, then determine whether $s_i = t_i$ or not. If $s_i \neq t_i$, then stop without accepting the input. If $s_i = t_i$, then continue.

(b) If $\delta(b_i) = b_{i_1} b_{i_2} \ldots b_{i_\ell}$, with $\ell \geq 1$, then choose nondeterministically integers $v_1, v_2, \ldots, v_{\ell-1}$ such that $v_0 = s_i \leq v_i \leq v_2 \ldots \leq v_{\ell-1} \leq t_i = v_\ell$. (If $\ell = 1$, then there is nothing to choose.) Then, for $1 \leq j \leq \ell$, do the following:

(i) If $T_0(i_j) = \phi$, then set $T_0(i_j)$ to the pair (v_{j-1}, v_j);

(ii) If $T_0(i_j) \neq \phi$, say $T(i_j) = (m_j, n_j)$, determine if the input tape has the same string on cells m_j through $n_j - 1$ as on cells v_{j-1} through $v_j - 1$. If not, then stop without accepting the input. If so, continue.

(4) For $1 \leq i \leq m$, set T(i) to $T_0(i)$. If $t_i = s_i+1$, for all $1 \leq i \leq m$ such that $T(i) \neq \phi$, then go to step (5); otherwise, go to step (2).

(5) For $1 \leq i \leq m$, if $T(i) \neq \phi$, determine whether b_i is the symbol in cell s_i of the input tape. If not, then go to (2). If for all $1 \leq i \leq m$, such that $T(i) \neq$

ϕ, b_i is the symbol in cell s_i of the input tape, then stop and accept the input.

It should be noted that this algorithm does not require more than log (n) space for inputs of length n. That is, the number of elements in the arrays T and T_0 is dependent only upon the number of symbols in the EDTOL grammar (not on the length of the input string). Furthermore, each element of these two arrays is either the empty set or a pair of natural numbers between 1 and n+1. Thus, at most log n space is required for, say, a binary representation of each element of the array. Thus, at most $c \cdot \log n$ cells are sufficient storage for the arrays, for some $c > 0$. Since constant factors are irrelevant for general tape bounded Turing machines, we have:

<u>Theorem</u>. EDTOL $\subseteq$ NSPACE (log n).

<u>Fact</u>. EDTOL $\subseteq$ DSPACE (log n) if and only if NSPACE (log n) = DSPACE (log n).

Proof. It is known that every linear CFL is an EDTOL language [21]. In [15] a linear CFL which is log tape complete for NSPACE (log n) was described. It follows that if EDTOL $\subseteq$ NSPACE (log n), then NSPACE (log n) = DSPACE (log n). The converse follows directly from the preceding theorem, <u>i.e.</u> EDTOL $\subseteq$ NSPACE (log n).

Acknowledgements

I am indebted to Jan Van Leeuwen for carefully reading an earlier version of this paper and his comments which helped improve the presentation. An earlier version of the theorem showing that EOL is contained in LOG(CFL) is contained in the paper [17], which was jointly authored with a graduate student at Northwestern University, A. K. Arora.

REFERENCES

(1). I.H. Sudborough, On the Tape Complexity of Deterministic Context-Free Languages, to appear. Some of these results are in "On Deterministic Context-Free Languages, Multihead Automata, and the Power of an Ausiliary Pushdown Store," Proceedings of the 8th Annual ACM Symposium on Theory of Computing (1976), 141-148.

(2). I.H. Sudborough, The Complexity of the Membership Problem for Some Extensions of Context-Free Languages, Intern. J. Computer Math., to appear.

(3). P.M. Lewis, R.E. Stearns, and J. Hartmanis, Memory Bounds for the Recognition of Context-Free and Context-Sensitive Languages, Proceedings of the Sixth Annual IEEE Symposium on Switching Circuit Theory and Logical Design (1965), 199-212.

(4). S.A. Greibach, The Hardest Context-Free Language, SIAM J. on Computing 2 (1973), 304-310.

(5). I.H. Sudborough, On Tape-Bounded Complexity Classes and Multi-Head Finite Automata, JCSS (1975), 62-76.

(6). G.T. Herman and G. Rozenberg, Developmental Systems and Languages, North Holland Publishers, Amsterdam, 1975.

(7). J. Van Leeuwen, The Tape Complexity of Context-Independent Developmental Languages, JCSS (1975), 203-211.

(8). J. Van Leeuwen, The Membership Questions for ETOL-languages is Polynomial Complete, Info. Processing Letters 3 (1975), 138-143.

(9). J.E. Hopcraft and J.D. Ullman, Formal Languages and Their Relation to Automata, Addison-Wesley Publishing Co., Reading, Mass., 1969.

(10). N.D. Jones, Space-Bounded Reducibility among Combinational Problems, JCSS 11 (1975), 62-85.

(11). A.R. Meyer and L.J. Stockmeyer, Word Problems Requiring Exponential Time, Proceedings of Fifth Annual ACM Symposium on Theory of Computing (1973), 1-9.

(12). S.A. Cook, Characteristics of Pushdown Machines in Terms of Time-Bounded Computers, JACM 18 (1971), 4-18.

(13). W.J. Savitch, Relationships Between Nondeterministic and Deterministic Tape Complexities, JCSS 4,2 (1970), 177-192.

(14). A.V. Aho and J.D. Ullman, The Theory of Parsing, Translation, and Compiling, Vol. I, Prentice-Hall Publishing Co., Englewood Cliffs, N.J., 1972.

(15). I.H. Südborough, On Tape-Bounded Complexity Classes and Linear Context-Free Languages, JACM (1975), 500-501.

(16). S.A. Cook, Path Systems and Language Recognition, Proceedings of Second Annual ACM Symposium on Theory of Computing (1970), pp. 70-72.

(17). A.K. Arora and I.H. Sudborough, On Languages log-tape reducible to context-free languages, Proceedings of the 1976 Conference on Information Sciences and Systems, Johns Hopkins University, Baltimore, Maryland, 1976.

(18). T. Harju, personal communication.

(19). T. Harju, A polynomial recognition algorithm for the EDTOL languages, Elektron. Informationsverarbeit. Kybernetik, to appear.

(20). N.D. Jones and S. Skyum, Recognition of deterministic ETOL languages in polynomial time, Technical Report DAIMI PB-63 (October, 1976), Institute of Mathematics, University of Aarhus, 8000 Aarhus C, Denmark.

(21). A. Salomaa, Parallelism in rewriting systems, in Automata, Languages and Programming, J. Loeckx (ed.), Springer-Verlag Lecture Notes in Computer Science Series 14 (1974), pp. 523-533.

(22). J. Van Leeuwen, Notes on pre-set pushdown automata, in L Systems, G. Rozenberg and A. Salomaa (eds.), Springer-Verlag Lecture Notes in Computer Science Series 15 (1974), pp. 177-188.

(23). G. Rozenberg and P. Doucet, on OL-Languages, Information and Control 19, 1971, pp. 302-318.

(24). P.M.B.Vitanyi, On the size of DOL languages, in L systems, G. Rozenberg and A. Salomaa (eds.), Springer-Verlag Lecture Notes in Computer Science Series 15 (1974), pp. 78-92.