Theoretical Computer Science 4 (1977) 143–153.

ECONOMY OF DESCRIPTION BY PARSERS, DPDA'S, AND PDA'S*

Matthew M. GELLER[1]
University of Michigan, Ann Arbor, MI 48105, U.S.A.

Harry B. HUNT, III[2]
Harvard University, Cambridge, MA 02138, U.S.A.

Thomas G. SZYMANSKI[3] and Jeffrey D. ULLMAN[3]
Princeton University, Princeton, NJ 08540, U.S.A.

Communicated by A. Meyer
Received May 1976
Revised September 1976

Abstract. It is shown that there is a sequence of languages $E_1, E_2, \ldots$ such that every correct prefix parser (one which detects errors at the earliest possible moment, e.g., LR or LL parsers) for E_n has size 2^{cn}, yet a deterministic PDA recognizing E_n exists and has size $O(n^2)$. There is another easily described sequence of languages $N_1, N_2, \ldots$ for which N_n has a nondeterministic PDA of size $O(n^2)$, but no deterministic PDA of size less than 2^{cn}. It is shown moreover, that this latter gap can be made arbitrarily large for different sequences of languages.

1. Introduction

Meyer and Fischer [10] attempted to analyze with respect to size certain systems for expressing languages. They obtained various results showing how specification of certain languages was far more economical in one system of specification than another. For example, they exhibit a family of languages $\{I_n \mid n \geq 1\}$ for which the size of finite automata needed to recognize languages of this family grows doubly exponentially in n, whereas the size of DPDA's recognizing I_n grow as $O(n^3)$.

In this paper, we exhibit several results of this flavor that relate to pushdown automata (PDA's), deterministic pushdown automata (DPDA's), and parsers. The results relating to parsers are particularly interesting, as we show there can be an

* A preliminary version of this report appeared in the Proceedings of the IEEE 16th Annual Symposium on Foundations of Computer Science.

[1] Work supported in part by NSF Grant MCS-76-07744.
[2] Work supported in part by U.S. Army Contract DA-31-124-ARO-D-462.
[3] Work supported in part by NSF Grant DCR-74-21939.

exponential difference between the size of a minimal DPDA for a language and the size of any DPDA which behaves as an LR(k) parser for the same language. The technique involved in the proof makes use of an unusual "closure property" which LR parsers possess but general DPDA's do not.

Geller and Harrison [3] present a model for comparing the size of the tables required by different bottom-up parsing algorithms for a given language. In [4] it is shown that a family of grammars, $\{G_n \mid n \geq 1\}$, exists, such that the size of production prefix parsers for G_n grows as $O(n^2)$, yet the size of LR(0) parsers grows as $O(2^n)$. The following question, however, still remained. Can we transform each grammar G_n to another grammar G'_n, generating the same language, such that the size of LR(0) parsers recognizing G'_n grows polynomially in n? In this paper, we answer that question in the negative. That is, a family of languages, L_n, is given for which there exists a family of grammars G_n, with the size of production prefix parsers (or precedence parsers or strict deterministic parsers) growing as $O(n^2)$. However, for any family of grammars generating L_n, the size of LR(0) parsers (in fact, the size of any correct prefix parser) grows as $O(2^{cn})$ for some $c > 0$. The *correct prefix* parsers include all parsers that halt as soon as an error has provably occurred. These include LL(k), SLR(k), LALR(k) and LR(k) parsers for all k. The correct prefix property and its relation to error detection and recovery in compilers is discussed in [1, 6].

We obtain several other results that relate the economy of description of certain families of languages to PDA's and DPDA's. A simple sequence of languages with an exponential size difference between PDA's and DPDA's recognizing them will be exhibited, and a result of [10] is generalized to show that there is, for example, no recursive bound relating the size of DPDA's and PDA's for the same regular set.

2. A family of languages that need exponentially growing PDA's for recognition

Definition 2.1. A *scanning PDA* is the standard PDA of Ginsburg [5] with the following modifications:

(1) To each input string we add an endmarker, \$.

(2) Acceptance occurs with only Z_0, the bottom of stack marker, on the pushdown store, the machine in a final state and the input tape empty.

(3) The stack can grow at most one symbol at a time. If it grows a symbol, the previous top stack symbol is not changed. That is, in one move, stack symbol X can be replaced by the empty string, by some other symbol Y or by XZ for some symbol Z. We refer to the language accepted by A as $T(A)$.

A *configuration* of a PDA will be denoted by a triple (q, α, w), where q is the current state of the PDA, α is the contents of the pushdown store (with the top of the store on the right) and w is the "unprocessed" portion of the input tape. The empty string will be denoted by Λ. By (2) above, a string w is accepted by a scanning PDA if and only if

$$(q_0, Z_0, w\$) \overset{*}{\vdash} (q_f, Z_0, \Lambda),$$

where q_0 is the intitial state and q_f is a final state of the machine.

By the *size* of an automaton or grammar, we mean the number of symbols used to specify it. However, when dealing with PDA's, the use of the state-symbol product will be far more convenient. Since we are dealing with exponential gaps in this paper, we will be able to interchange these measures, as the following lemma indicates.

Lemma 2.2. *For a given input alphabet there are constants $c_1 > 0$ and c_2 such that for any scanning PDA with standard description of length n and state-symbol product m, we have $c_1\sqrt{m} \leqslant n \leqslant c_2m^2$.*

Proof. Let there be s states and t stack symbols, so $m = st$. The standard list of alternates for each state, input and stack symbol of a scanning PDA can have at most $s(2t + 1)$ entries, since there are, by condition (3) of Definition 2.1, only $2t + 1$ stack moves performable in any situation. Thus, $n = O((st)[s(2t + 1)]) = O(m^2)$.

For the lower bound on m simply observe that s and t are each no greater than n, since each state and symbol must be mentioned in the standard representation of the PDA. □

Lemma 2.3. *There is a constant c such that for any PDA with standard description of length n there is an equivalent scanning PDA with description of length at most cn^2.*

Proof. The standard constructions of Ginsburg [5] suffice. □

From here on, we shall use "PDA" to mean scanning PDA and "size" to mean state-symbol product.

We next introduce the notion of a scan. A scan is a special kind of sequence of moves in which the symbols below that symbol which was on top of the pushdown store initially, have no effect on the behavior of the machine.

Definition 2.4. Let A be a (scanning) PDA with state set Q, input alphabet Σ, pushdown alphabet Γ, initial state q_0, final state set F and bottom of stack marker Z_0. If p and q are in Q, X and Y are in Γ and w is in $\Sigma^* \cup \Sigma^*\$$, we say that A makes a *qXpY scan* on w provided there exist $q_f \in F$, $\alpha \in \Gamma^*$ and t, $z \in \Sigma^*$ for which

$$(q_0, Z_0, twz) \overset{*}{\vdash} (q, \alpha X, wz) \overset{*}{\vdash} (p, \alpha Y, z) \overset{*}{\vdash} (q_f, Z_0, \Lambda)$$

and in the sequence of moves

$$(q, \alpha X, wz) \overset{*}{\vdash} (p, \alpha Y, z)$$

the stack always contains at least $|\alpha X|$ elements.

We next prove a technical lemma that allows us to infer that if some sufficiently long string causes a scan, then it has some substring of smaller length that also causes a scan. In particular, we can find in any computation of a PDA a scan whose length is within a factor of two of some desired length. This lemma will be necessary for a future combinatorial argument, and it is a generalization of a lemma originally appearing in Lewis, Stearns and Hartmanis [9].

Lemma 2.5. *Let A be a PDA and let c be any constant between* 0 *and* 1. *Then if x is any input of length at least* $4/c$ *which is accepted by A, we may write* $x\$ = x_1x_2x_3\$$ *such that* $\frac{1}{2}c|x| \leq |x_2| \leq c|x|$ *and A makes a scan on* x_2 *or* $x_2\$$.

Proof. We constuct x_2 by the following recursive algorithm. At all times we have a substring y of $x\$$, with $|y| > c|x|$, on which A makes a scan. Initially $y = x\$$. Whenever the algorithm calls itself, it does so on a string with a shorter scan than the given scan for y.

The scan on y can be of two types, depending on whether the first move grows the stack or not.

Case 1. The scan of y uses an input symbol before growing its stack. Then $y = ay'$, for some $a \in \Sigma$, and

$$(q, \alpha X, y) \overset{*}{\vdash} (q', \alpha X', y') \overset{*}{\vdash} (p, \alpha Y, \Lambda).$$

Then A makes a scan on substring y' of x. If $|y'| \leq c|x|$, then x_2 is y'. Note in the case where $|y'| \leq c|x|$, we must have $|y'| \geq \frac{1}{2}c|x|$. For $|y| > c|x|$, so $|y'| > c|x| - 1$. If $|y'| < \frac{1}{2}c|x|$, a contradiction of the hypothesis $|x| \geq 4/c$ is immediate. If, on the other hand, $|y'| > c|x|$, recursively apply the algorithm to y'.

Case 2. (The stack grows before the first use of an input symbol). Then we may write $y = b_1y_1b_2y_2$, with b_1 and b_2 each either in Σ or equal to Λ, and the scan of y may be written

$$(q, \alpha X, y) \overset{*}{\vdash} (q', \alpha X'Z, y_1b_2y_2) \overset{*}{\vdash} (q'', \alpha X''Z', b_2y_2) \vdash (q''', \alpha X'', y_2) \overset{*}{\vdash} (p, \alpha Y, \Lambda),$$

where scans are made on $b_1y_1b_2$, y_1 and y_2.

If $|y_1| > c|x|$, call the algorithm recursively on y_1. If $\frac{1}{2}c|x| \leq |y_1| \leq c|x|$, then choose $x_2 = y_1$, and we are done. The only case remaining is where $|y_1| < \frac{1}{2}c|x|$. Suppose $|b_1y_1b_2| \geq \frac{1}{2}c|x|$. Given the hypothesis $|x| \geq 4/c$, together with the assumption $|y_1| < \frac{1}{2}c|x|$ it is easy to show $|b_1y_1b_2| \leq c|x|$. Thus, we may pick $b_1y_1b_2$ for x_2. Thus assume $|b_1y_1b_2| < \frac{1}{2}c|x|$. Then $|y_2| > \frac{1}{2}c|x|$. If $|y_2| \leq c|x|$ we are done; if not, apply the algorithm to y_2. Note that y_2 may be y, but the scan $(q''', \alpha X'', y_2) \overset{*}{\vdash} (p, \alpha Y, \Lambda)$ is shorter than the given scan for y, so the algorithm will converge. □

We now restrict ourselves to a special family of languages. We let $\Sigma_n = \{a_1, \ldots, a_n\}$, and let L_n be the set of permutations of Σ_n, $n \geq 1$.

We next wish to show that distinct permutations establish distinct scans when one of the languages L_n is being recognized.

Lemma 2.6. *Let P be a PDA accepting L_n. If P makes pXqY scans on two strings x and y, then x and y are permutations of one another.*

Proof. Otherwise, substitute y for x and accept a word not in L_n. ☐

We are now ready to prove that PDA's accepting L_n grow as 2^{cn} in size.

Theorem 2.7. *There exists a constant $c > 0$ such that for $n \geq 6$, any PDA P accepting L_n has size at least 2^{cn}.*

Proof. It suffices to show that for some $d > 0$ there are 2^{dn} distinct strings, not permutations of one another, such that while accepting some word, P makes scans on those strings. For then by Lemma 2.6, there must be 2^{dn} quadruples $qXpY$ such that $qXpY$ scans are performed by P. Therefore, the size of P is at least $2^{dn/2}$.

Construct a maximal collection of sets $S_1, S_2, \ldots, S_s$ each included in Σ_n, such that

(1) for each S_i, there is some string w_i, a permutation of S_i, such that P performs a scan on w_i or $w_i\$$, and

(2) $\frac{1}{3}n \leq k_i \leq \frac{2}{3}n$ for all i, where k_i is the size of S_i.

The number of strings in L_n that contain the symbols of S_i as a substring is $k_i!(n - k_i + 1)!$. By condition (2), this number is at most $(n/3)!((2n/3) + 1)!$. Thus, if

$$s < \frac{1}{n+1}\binom{n+1}{n/3},$$

there is some string w in L_n which contains none of $S_1, S_2, \ldots, S_s$ as a substring. By Lemma 2.5 with $c = 2/3$, w causes P to perform a scan on a substring w' or $w'\$$, where w' is of length between $\frac{1}{3}n$ and $\frac{2}{3}n$. By hypothesis, the set S of symbols of w' is none of $S_1, S_2, \ldots, S_s$. Thus $\{S_1, S_2, \ldots, S_s\}$ was not maximal as supposed. Hence

$$s \geq \frac{1}{n+1}\binom{n+1}{n/3} \geq 2^{dn}$$

for some $d > 0$. ☐

We also wish to consider another family of languages, namely $Q_n = \{x \# x \mid x \in \{0, 1\}^n\}$. Note that words in Q_n are of length $2n + 1$. We get a similar result, namely:

Theorem 2.8. *There exists a constant $c > 0$ such that for $n \geq 8$, any PDA recognizing Q_n has size at least 2^{cn}.*

Proof. As in Lemma 2.6, we can show that a PDA P accepting Q_n cannot make $qXpY$ scans on two distinct strings whose lengths do not exceed $n+1$. (Note that this result does not hold for strings of greater length, as the strings could contain corresponding symbols of the two copies of x in $x \# x\$$).

Thus, consider a maximal set of strings $y_1, y_2, \ldots, y_s$ in $(0+1)^*$ that

(1) For each y_i there is a string in Q_n on which P makes a scan, and

(2) $(n+1)/2 \leq |y_i| \leq n+1$ for all i.

Any word in Q_n is determined by knowing any $n+1$ consecutive positions. Each y_i is therefore a substring of at most $2^{n+1-|y_i|} \leq 2^{(n+1)/2}$ words of Q_n. By Lemma 2.5 with $c=(n+1)/(2n+1)$ we may use the technique of Theorem 2.7 to show $s \geq 2^{(n+1)/2}$ for $n \geq 8$. The balance of the argument follows Theorem 2.7. □

We now need a lemma relating a closure property to the growth rate of sequences of PDA's.

Lemma 2.9. *Let $\{M_n \mid n \geq 1\}$ be some family of languages and $\{R_n \mid n \geq 1\}$ a family of regular sets. Assume that there exists a constant $c > 0$ such that for sufficiently large n, any PDA recognizing $M_n \cap R_n$ is of size greater than 2^{cn}. Also assume there exists some function $f(n)$, such that for sufficiently large n some finite automaton recognizing $R_n\$$ has at most $f(n)$ states. Then for sufficiently large n, any PDA recognizing M_n is of size greater than $2^{cn}/f(n)$.*

Proof. If not, then the standard "cross product of states" construction [5] provides a PDA recognizing $M_n \cap R_n$ with size less than 2^{cn}. □

We now give two examples where this lemma is applied.

Example 2.10. We let $M_n = \{$the set of strings in Σ_n^* containing at least one instance of each symbol$\}$. We know $L_n = M_n \cap (\Sigma_n)^n$, and $(\Sigma_n)^n\$$ is recognized by an $n+2$ state automaton. Therefore, it follows from Theorem 2.7 and Lemma 2.9 that there exists a constant $c>0$ such that for sufficiently large n, any PDA recognizing M_n is of size greater than 2^{cn}. This example will be useful later on in examining the applications of the results of this section to parsing.

Example 2.11. Let $P_n = \{x_1 2 x_2 2 \cdots 2 x_k 22 x_i \mid x_j \in \{0,1\}^n$ for $1 \leq j \leq k$, $1 \leq i \leq k$, and regarded as binary integers, $x_j < x_{j+1}$ for $1 \leq j < k\}$. Consider $P_n \cap (0+1)^n 22 (0+1)^n$, which is essentially Q_n. Therefore, it follows from Theorem 2.8 and Lemma 2.9 that there exists a constant $c>0$ such that for sufficiently large n, any PDA recognizing P_n is of size greater than 2^{cn}. This example resolves a conjecture of Meyer and Fischer [10].

3. The size of minimal PDA's recognizing languages compared with the size of minimal DPDA's and other deterministic devices

In this section, we first demonstrate a particular family of languages N_n for which the size of PDA's accepting N_n grows polynomially in n, yet the size of DPDA's recognizing N_n grows exponentially in n. We then show that no recursive function can bound the gain in enconomy of PDA's over DPDA's.

Lemma 3.1. *Let $L \subseteq \Sigma^*$ be a language accepted by a DPDA P of size m. Then $\bar{L}$ is accepted by a DPDA of size $3m$.*

Proof. Delete from the description of P any transition $\delta(q, \Lambda, X)$ if for every i, there exists a state p and non-empty string γ such that $(q, X, \Lambda) \vdash (p, \gamma, \Lambda)$. The test for such transitions can in fact be performed in polynomial time, although this fact is irrelevant to the present proof. The resulting DPDA has no loops. Then use the construction of [5] to complement the language accepted by the DPDA. □

This result leads to the following theorem, which shows that there exists a natural sequence of languages for which small PDA's exist, yet for which DPDA's must be large.

Theorem 3.2. *There exists a constant $c > 0$ such that for sufficiently large n, any DPDA P accepting*

$$N_n = \{a_{i_1} a_{i_2} \cdots a_{i_n} b_{j_1} b_{j_2} \cdots b_{j_m} \mid 1 \leq i_k \leq n,\ 1 \leq j_k \leq n \text{ and } (\exists r)(\forall s)\ i_r \neq j_s\}$$

has size at least 2^{cn}.

Proof. By Lemma 3.1, given P we may construct P', of size polynomial in the size of P, accepting $\overline{N_n}$. Construct DPDA P'' to simulate P' on (imaginary) input $a_1 a_2 \cdots a_n$ and then read a (real) input string of b's, again simulating P'. The size of P'' is also polynomial in the size of P. But P'' accepts the language M_n of Example 2.10, with a's recoded as b's, which we showed requires exponentially sized PDA's. □

Comparison of the size of DPDA's and PDA's for context free grammars reveals a property displayed by Meyer and Fischer [10] between finite automata and context free grammars. That is, the gain in economy can be arbitrary[3]. We prove a considerably stronger result, in fact.

[3] A.R. Meyer points out that this result follows from Meyer and Fischer [10] and the result of Stearns [12], which showed a recursive relationship between the sizes of minimal finite automata and DPDA's for a given language.

Lemma 3.3. [10]. *Let f be any recursive function. Then there is a Turing machine* T_f *which on any input of length n accepts after a sequence of at least* $f(n)$ *moves. Moreover, there is a constant k such that for each input x of length n there is a context free grammar* $G_{f,x}$ *of size at most kn, which generates the noncomputations of* T_f *on input x.*

Note that $\overline{L(G_{f,x})}$ is a single string of length at least $f(n)$, whenever $f(n)$ is defined.

Lemma 3.4. *Let* $\mathcal{D}$ *be any class of language descriptors, e.g., DPDA's. Suppose that for any* $D \in \mathcal{D}$ *there exists* $D' \in \mathcal{D}$ *defining the complement of* $L(D)$*, the language defined by D, and* $c(n)$ *is a total recursive function such that* $\text{size}(D') \leq c(\text{size}(D))$*. Further, let* $z(n)$ *be a total recursive function such that if* $L(D)$ *has a word x of length greater than* $z(\text{size}(D))$*, then* $L(D)$ *has some word besides x, i.e., x can be "pumped." Then there is no total recursive function g such that for every CFG G of size m generating a regular set there is a descriptor* $D \in \mathcal{D}$ *defining* $L(G)$*, with* $\text{size}(D) \leq g(m)$.

Proof. Suppose there were such a g, and let $f(n) = zcg(n^2)$. By Lemma 3.3, there is an integer k and a sequence of context free grammars $G_1, G_2, \ldots$, where G_i is of size at most ki and generates a language whose complement is one string of length at least $f(i)$. By hypothesis, there is a descriptor D_1 such that $L(D_1) = L(G_k)$ and $\text{size}(D_1) \leq g(k^2)$. Then there is a descriptor D_2 such that $L(D_2) = \overline{L(G_k)}$, and $\text{size}(D_2) \leq cg(k^2)$.

If $zcg(k^2) \leq f(k)$, then $\overline{L(G_k)}$ contains more than one string, so $f(k) < zcg(k^2)$. But $f(k) = zcg(k^2)$ by definition. Hence, g does not exist. □

Theorem 3.5. *For regular sets there is no recursive relationship between the sizes of a context-free grammar (or PDA) and the smallest equivalent finite automaton, DPDA or 1 way deterministic stack automaton.*

Proof. Closure of one way deterministic stack automata under complement is easy. "Pumping" for these devices follows from [11].

Independently, Valiant [13] has obtained a related result, that there is no recursive relationship between the sizes of unambigious context free grammars and DPDA's for the same language. Close examination of Valiant's construction reveals that the unambiguous grammars involved are actually deterministic grammars, that is, grammars which are LR. He has thus shown that no recursive relationship holds between the sizes of a deterministic grammar and the smallest equivalent DPDA. □

4. Applications to parsing

We now wish to apply the results we have obtained to parsing. Geller, Graham and Harrison [4] show that there exist families of grammars $\{G_n \mid n \geq 1\}$ for which the size of production prefix parsers grows as cn^2, while SLR(k) parsers grow as 2^{cn}. In this section, we shall show that for this family of grammars, for any grammars generating the same language, SLR(k) parsers must grow as 2^{cn}.

This result will follow from the fact that by nature of the correct prefix property (cf. Graham and Rhodes [6]) correct prefix parsers have the task of recognizing two distinct languages. That is, the parsers must halt with the correct parse after reading a correct input, and must also halt and declare error on an input as soon as the input has been found incorrect. We shall exhibit a seequence of languages which can be recognized by a sequence of DPDA's growing polynomially in size, yet any sequence of PDA's recognizing the set of input strings on which the parser first declares error grows exponentially in size. First, we need some definitions.

Definition 4.1. Let $L \subseteq \Sigma^*$ be a language. Then $x \in \Sigma^*$ is a *correct prefix* of L if and only if there exists some $z \in \Sigma^*$ such that $xz \in L$.

Definition 4.2. If L is a deterministic CFL, let $I(L) = \{x \mid x = ya$ for some symbol a, y is a correct prefix of L, but x is not$\}$.

Definition 4.3. Let A be a DPDA acting as a parser for L. Then A is a *correct prefix parser* if by changing the set of final states of A we produce a recognizer for $I(L)$. Note that many types of parsers are correct prefix parsers. Important examples are LR(k), LL(k), SLR(k) and LALR(k).

It follows by definition that:

Lemma 4.4. *Let L be a deterministic CFL. Then the smallest correct prefix parser for L is no smaller than the smallest DPDA for I(L).*

We can apply Lemma 4.4 to exhibit a specific sequence of languages having an exponential gap between the sizes of their smallest DPDA's and smallest correct prefix parsers.

Theorem 4.5. *There is a constant c such that for sufficiently large n, every correct prefix parser for the language*

$$E_n = \{a_{i_1}a_{i_2}\cdots a_{i_n}b_j \mid 1 \leq i_k, j \leq n \text{ and } j \neq i_r \text{ for any } r\}$$

has size at least 2^{cn}.

Proof. We see that $I(E_n) \cap \Sigma_{n-1}^* \cdot a_n = M_{n-1} \cdot a_n$, where M_{n-1} is the language of Example 2.10. By an argument similar to that of Example 2.10, there is a constant c such that for sufficiently large n, every DPDA recognizing $M_{n-1} \cdot a_n$ has size at least 2^{cn}. But, by Lemmas 2.9 and 4.4, every correct prefix property parser for E_n is at least as large as any DPDA recognizing $M_{n-1} \cdot a_n$. □

There are, however, strict deterministic and production prefix parsers for each E_n, $n \geq 1$ with size $O(n^2)$.

Theorem 4.6. *Consider the sequence of grammars*

$$G_n = (V_n, \Sigma_n, P_n, S) \quad \text{where } n \geq 1,\ \Sigma_n = \{a_i, b_i \mid 1 \leq i \leq n\} \quad \text{and}$$

$$P_n = \{A_i \to a_j A_i \mid 1 \leq i, j \leq n,\ j \neq i\} \cup \{A_i \to b_i \mid 1 \leq i \leq n\} \cup \{S \to A_i \mid 1 \leq i \leq n\}.$$

Then:

(1) $L(G_n) = E_n$ *for all* $n \geq 1$.

(2) *There exist production prefix and strict deterministic parsers for* G_n *with size* $O(n^2)$.

Proof. (1) Is easy to show.

(2) Is found in [4]. □

Remark. It is easy to show the converse to Theorems 4.5 and 4.6, that is, for every DPDA X there is an equivalent correct prefix parser at most exponentially larger than X. By way of proof, consider the construction of the "predicting machine" in [8].

5. Conclusions

In summary, we have exhibited families of languages with the following behaviors:

	minimum size correct prefix parser	minimum size DPDA	minimum size PDA or CFG
L_n, M_n, Q_n	2^{cn}	2^{cn}	2^{cn}
N_n	2^{cn}	2^{cn}	$\leq cn^2$
E_n	2^{cn}	$\leq cn^2$	$\leq cn^2$

We have shown one of the conjectures of Meyer and Fischer [10] to be true. In addition, we have shown that the gain in economy of description of a PDA over a DPDA or even over a one way deterministic stack automaton can be arbitrary.

Acknowledgement

The authors would like to thank the referees and Albert Meyer for their helpful comments on an early draft of this paper.

References

[1] A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation, and Compiling*, Vols. I and II (Prentice Hall, Englewood Cliffs, NJ, 1972 and 1973).

[2] M.M. Geller, Compact parsers for deterministic langauges, Ph.D. Thesis, University of California, Berkeley (1975).

[3] M.M. Geller and M.A. Harrison, Strict deterministic versus LR(0) parsing, *Conf. Record ACM Symp. Principles of Programming Languages* (1973) 22–32.

[4] M.M. Geller, S.L. Graham and M.A. Harrison, Production prefix parsing, *Second Colloq. Automata, Languages and Programming* (1974) 232–241.

[5] S. Ginsburg, *The Mathematical Theory of Context-Free Languages* (McGraw-Hill, NY, 1966).

[6] S.L. Graham and S.P. Rhodes, Practical syntactic error recovery, *Comm. ACM* **18** (11) (1975) 639–650.

[7] M.A. Harrison and I.M. Havel, Strict deterministic grammars, *J. Comput. System Sci* **7** (1973) 237–277.

[8] J.E. Hopcroft and J.D. Ullman, *Formal Languages and their Relation to Automata* (Addison-Wesley, Reading, MA, 1969).

[9] P.M. Lewis II, R.E. Sterns and J. Hartmanis, Memory bounds for recognition of context-free and context-sensitive languages, *IEEE Conf. Record 6th. Ann. Symp. Switching Circuit Theory and Logical Design* (1966) 191–202.

[10] A.R. Meyer and M.J. Fischer, Economy of description by automata, grammars, and formal systems, *Conf. Record* 1971 *Ann. Symp. Switching and Automata Theory*, (1971) 188–191.

[11] W. Ogden, Intercalation theorems for stack languages, *Proc. ACM Symp. Theory of Computing* (1969) 31–42.

[12] R.E. Stearns, A regularity test for pushdown machines, *Information and Control* **11** (3) (1967) 323–340.

[13] L.G. Valiant, A succinctness result for descriptions of deterministic languages, TR 70, University of Leeds, (July, 1975).