LOWER BOUNDS ON THE SIZE OF DETERMINISTIC PARSERS

by

Esko Ukkonen

Memorandum No. UCB/ERL M81/98

15 October 1981

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

/

Lower bounds on the size of deterministic parsers*

Esko Ukkonen

Department of Computer Science, University of Helsinki
Tukholmankatu 2, SF-00250 Helsinki 25, Finland
and
Computer Science Division - EECS, University of California
Berkeley, California 94720, USA

ABSTRACT

Worst-case lower bounds on the size of deterministic parsers as a function of the size of the grammar are studied. It is shown first that there is no recursive function bounding the succinctness gained using parsable context-free grammars instead of parsers. Also is shown that there exists an infinite family of LL(2) grammars such that the size of *every* left or right parser for these grammars must be $\geq 2^{cm}$ for some $c>0$, where m is the size of the grammar. Similarly, it is shown that there exists an infinite family of LR(0) grammars such that the size of *every* right parser for these grammars must be $\geq 2^{c\sqrt{m}}$. Hence for all $k \geq 0$, the class of the LR(k) grammars cannot be parsed using right parsers whose size is polynomially bounded in the size of the grammar, and for all $k \geq 2$, the class of the LL(k) grammars cannot be parsed using left parsers whose size is polynomially bounded in the size of the grammar.

Key Words: size complexity, parsable grammar, left parser, right parser, LL(k) parsing, LR(k) parsing.

CR Categories: 5.25, 5.23, 5.27.

* This work was partially carried out when I was visiting the Computer Science Division of the University of California at Berkeley. I am indebted to Professor Michael A. Harrison for providing me with this opportunity. The work was supported by the Academy of Finland and by the Finnish Cultural Foundation. Partial support was provided by the National Science Foundation Grant MCS 79-15763 (Univ. of California).

The main results of Sections 4 and 5 were reported at the Eighth Colloquium on Automata, Languages and Programming [14].

1. Introduction

In this paper we examine the relationship between the sizes of parsable context-free grammars and parsers for them. By a parser we mean a deterministic pushdown transducer which produces the translation from terminal strings to parses. Parsers that use mechanisms more powerful than a single pushdown stack are not considered.

First we study the most general form of the problem: How much larger must a deterministic left or right parser be than the context-free grammar it parses? We show that for certain parsable grammars the parsers must always be enormously larger than the grammars. Specifically, in Section 3 we show that there cannot be any recursive function bounding the size of a smallest parser as a function of the size of the grammar it parses. This result is obtained as a corollary to a result of Valiant [15] that in an infinite class of deterministic context-free languages the size of a deterministic pushdown automaton for a language is not recursively bounded by the size of the smallest unambiguous context-free grammars for the same language.

Although this nonrecursiveness result means that there cannot be any general purpose algorithm for constructing single-stack parsers for all parsable grammars, by imposing suitable restrictions on parsability we get classes of grammars parsable by small parsers. For example, different classes of precedence grammars as well as bounded context grammars (see e.g. [1]) and strict deterministic grammars [6] have parsers of polynomial size. Grammars in these classes must satisfy strong restrictions.

The LL(k) and LR(k) parsers and grammars [9,13] are more problematic. The LL(k) and LR(k) grammars contain all the context-free grammars that can be parsed top-down and, respectively, bottom-up using a stack and k symbol lookahead. In this sense LL(k) and LR(k) parsers are as powerful as we may hope. However, the known LL(k) and LR(k) parser constructions, as described e.g. in [1], may produce parsers that are non-polynomially larger than the grammars. For the LR(k) method this size gap was observed in [2]. We will give an example of such a gap for the LL(k) parsers when $k \geq 2$.

Thus we are forced to ask, are such non-polynomial size differences necessary or might it be possible to find a parser construction for LL(k) or LR(k) grammars always producing parsers with polynomially bounded size? In some special cases such an improvement is known: In [3] a family of grammars is given for which the LR(0) parsers are non-polynomially larger than the production prefix parsers or simple precedence parsers. This result can be explained by the observation that the LR(0) parsers are correct prefix parsers (i.e., the string read by the parser is always a prefix of some correct string in the language), while the production prefix parsers or simple precedence parsers are not. The correct prefix property radically increases parser size for the grammars of [3].

In this paper we show that in the general case such improvements are impossible because independently of the parsing method used, the non-polynomial size gaps cannot be totally avoided for the classes of LL(k) and LR(k) grammars. This means that the correct prefix property is not the only reason for the non-polynomial size of LL(k) and LR(k) parsers.

To prove our results on LL(k) grammars we give in Section 4 an infinite sequence of LL(2) grammars and show that the size of *any* left or right parser for such a grammar must be at least an exponential function of the size of the grammar. This implies that if $k \geq 2$, grammatical classes LL(k) and LR(k) cannot be parsed using (left or right) parsers of polynomial size. On the other hand, since an LL(1) grammar is always strong LL(1) and thus has a left parser of polynomial size, in the LL(1) case the size difference is only polynomial.

We also analyze a sequence of LR(0) grammars mentioned already by Earley [2] (and attributed by him to John Reynolds) as an example of a grammar family for which the LR(k) construction gives non-polynomially large parsers. We show in Section 5 that *all* right parsers for these grammars must be at least of the same size as an LR(0) parser. A non-polynomial difference between the parser size and grammar size therefore exists for LR(k), SLR(k) and LALR(k) grammars when $k \geq 0$.

In the literature, most closely related to this paper is a recent work by Pittl [11]. He independently gives an example family of LR(1) grammars with no polynomial size right parsers but does not consider left parsers. Also a work [4] by Geller, Hunt, Szymanski and Ullman investigates the size of different pushdown automata (but not parsers, i.e. pushdown transducers emitting a parse). For example, they generalized the result from [3] mentioned above by giving a family of languages such that there is an exponential difference between the size of a minimal deterministic pushdown automaton (dpda) for a language and the size of any dpda with the correct prefix property for the same language. They also gave a family of languages $\{N_n\}$ such that there is an exponential difference between the size of a minimal context-free grammar for a language and the size of any dpda for the same language. It is easily seen that this exponential difference is not preserved if parsers in our sense are considered. This is because every LL(k) or LR(k) grammar for N_n must be exponentially larger than the minimum size context-free grammar for N_n.

Let us conclude the present introduction by giving our notational conventions for strings, context-free grammars and parsers. We mainly follow the notation of [1] with the exception that the *length* of a string s is denoted by $lg(s)$. Recall that e denotes the empty string and $\#W$ the number of elements of a set W. The ***reversal*** of a string s is denoted by s^R.

The ***size*** of a (context-free) grammar $G = (N, \Sigma, P, S)$ is defined by

$$|G| = \sum_{A \to \alpha \in P} lg(A\alpha).$$

As noted in [5], the ***norm*** of G given by $||G|| = |G| \cdot \log_2 \#(N \cup \Sigma)$ would be a more realistic measure. We have, however, that $||G|| \leq |G| \cdot \log_2 |G| \leq |G|^2$, which means that the more convenient measure $|G|$ can be used in this paper *because* we are interested in proving larger than polynomial gaps.

If there is in G a leftmost derivation $S \Rightarrow_L^{\pi} \alpha$ where π is the sequence of productions applied in the derivation, then π is called a *left parse* of α in G. Similarly, if there is a rightmost derivation $S \Rightarrow_R^{\pi} \alpha$ then π^R is a *right parse* of α in G.

2. Parsability and deterministic parsers

Our purpose is to prove some lower-bound results which should be valid for all parsers. We will use deterministic pushdown transducer as the formal parser model. The analysis will be restricted to left parsers and to right parsers. Obviously, all different deterministic (or non-backtracking) left and right parsers and parsing methods proposed in the literature can be abstracted as deterministic pushdown transducers which translate input strings to left parses or to right parses. To be precise, this is true only if the parsing method is based on only one pushdown stack and one left-to-right scan of the string to be parsed. For example, the parsing method for LR-regular grammars and for some other analogous classes needs two scans. We leave such parsers out of consideration.

The 'real' parsers which are used for parsing context-free grammars are usually not represented in the pushdown transducer form. But because they can be transformed into this form with at most a polynomial increase in the size, our results on nonpolynomial size gaps between a parser and the grammar are valid also for them.

Let us next give the formal definitions. A *parser* for G is a deterministic pushdown automaton accepting language $L(G)$ generated by G and giving for each w in $L(G)$ a parse of w as an output. In general, a deterministic pushdown automaton with output is called a *deterministic pushdown transducer* (dpdt) and defined as an 8-tuple $T = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F)$, where Q denotes states, Σ input alphabet, Γ pushdown alphabet, Δ output alphabet, δ transition function (a partial function) from $Q \times (\Sigma \cup \{e\}) \times \Gamma$ to $Q \times \Gamma^* \times \Delta^*$ satisfying the following determinism condition: if $\delta(q,e,t)$ is defined then $\delta(q,a,t)$ is undefined for all $a \in \Sigma$. Furthermore, q_0 denotes initial state, Z_0 initial pushdown element and $F \subset Q$ the final states. By deleting all the occurrences of the output alphabet Δ in the definition of a dpdt we get definition of a deterministic pushdown automaton (dpda).

A *configuration* of T is denoted by (q,x,α,π) where $q \in Q$ is the current state, $x \in \Sigma^*$ is the unused portion of the input, $\alpha \in \Gamma^*$ is the current content of the pushdown stack (with the top of the stack to the left), and $\pi \in \Delta^*$ is the output string emitted to this point. Each pair (q,Z) in $Q \times \Gamma$ is called a *mode* of T. The mode of a configuration $(q,x,Z\alpha,\pi)$ is (q,Z) where Z is the topmost stack symbol. The *next move relation* $\vdash$ among configurations is defined in the usual way, that is,

$$\text{if } \delta(q,a,Z)=(q',\alpha',\pi'), \text{ then } (q,ax,Z\alpha,\pi) \vdash (q',x,\alpha'\alpha,\pi\pi'), \text{ and}$$
$$\text{if } \delta(q,\varepsilon,Z)=(q',\alpha',\pi'), \text{ then } (q,x,Z\alpha,\pi) \vdash (q',x,\alpha'\alpha,\pi\pi'),$$

for all x in Σ^*, α in Γ^* and π in Δ^*.

Let now $\$$ be a symbol not in $\Sigma \cup N$. A (*deterministic*) *left parser* for grammar $G = (N,\Sigma,P,S)$ is formally defined as a dpdt $T = (Q,\Sigma \cup \{\$\},\Gamma,P,\delta,q_0,Z_0,F)$ such that T accepts by final state and empty stack the language $L(G)\$$ and for each accepted input $x\$$, T outputs a left parse of x in G. Thus G has a derivation $S \Rightarrow_L^{\pi} x$ for some terminal string x if and only if T has a move sequence $(q_0,x\$,Z_0,e) \vdash^* (q,e,e,\pi)$ where q is in F. Similarly, a (*deterministic*) *right parser* for grammar $G = (N,\Sigma,P,S)$ is a dpdt T which is like a left parser but outputs a right parse of x for each input $x\$$ where x is in $L(G)$.

A grammar is called *left parsable* if it has a left parser and *right parsable* if it has a right parser [1]. It follows that a parsable grammar always generates a *deterministic language*, that is, a language recognized by some dpda.

Parsing some parsable grammars G is not possible without using unbounded *lookahead*. To announce the next production appearing in the parse, say $A \rightarrow \omega$, every parser for such a grammar G must sometimes scan the input string an unbounded amount beyond the point where the string produced from this particular production $A \to \omega$ begins (in left parsing) or ends (in right parsing). Such parsers are inconvenient. Moreover, in the next section we will see that there cannot be any algorithm that constructs a parser for every parsable grammar.

These difficulties disappear if we consider grammars that are parsable with lookahead whose length is always bounded. To state this restricted form of parsability more formally, let $k \geq 0$ be an integer. Grammar G is called ***left parsable with lookahead k***, if G has a left parser T such that if an input $z\$$ where $z \in L(G)$ leads in T to a computation

$$(q_0, z\$, e, e) \vdash^* (q, uv\$, \gamma, \pi) \vdash (q', v\$, \pi(A \to \omega)) \qquad (1)$$

then the leftmost derivation of z in G is

$$S \Rightarrow_L^* xA\alpha \Rightarrow_L x\omega\alpha \Rightarrow_L^* xy = z$$

and $|y| - |v| \leq k$. This simply says that at the moment of announcing $A \to \omega$, T has scanned at most k symbols beyond the beginning of the string produced from $A \to \omega$.

Analogously, G is called ***right parsable with lookahead k***, if G has a right parser T such that if an input $z\$$ where $z \in L(G)$ leads in T to a computation (1) then the rightmost derivation of z in G is

$$S \Rightarrow_R^* \alpha Ay \Rightarrow \alpha\omega y \Rightarrow_R^{*R} xy = z$$

and $|y| - |v| \leq k$.

Because the standard LL(k) parser construction yields for LL(k) grammars a left parser which works with lookahead k and the standard LR(k) parser construction yields for LR(k) grammars a right parser which works with lookahead k (and these parsers can further be transformed into the dpdt form), we have the following expected characterization:

Proposition 1. *A grammar is LL(k) if and only if it is left parsable with lookahead k, and a grammar is LR(k) if and only if it is right parsable with lookahead k.* ■

The *size* of a dpdt T is defined by (c.f. [5])

$$|T| = \sum_{\delta(q,a,Z)=(q',\alpha,\gamma)} (3+lg(a)+lg(\alpha)+lg(\gamma)).$$

Thus the size means the length of a string listing the transition function. As for grammars, the norm $||T|| = |T|\cdot\log_2\#(Q\cup\Sigma\cup\Gamma\cup\Delta)$ is a more realistic measure, but because again $||T|| \leq |T|\cdot\log_2|T| \leq |T|^2$, the more convenient measure $|T|$ can be used in proving non-polynomial gaps between the sizes of T and G.

Since $\#Q\leq|T|$ and $\#\Gamma\leq|T|$, the number of different modes satisfies

$$\#(Q\times\Gamma) \leq |T|^2. \tag{2}$$

To show that $|T|$ grows faster than polynomially in $|G|$ it therefore suffices to prove that $\#(Q\times\Gamma)$ is not polynomially bounded in $|G|$.

A parser or a dpdt T is called *moderate* if its transition function δ is such that whenever $\delta(q,a,Z) = (q',\alpha,\gamma)$ then $lg(\alpha) \leq 2$. Assuming this normal form makes some proofs simpler. If $lg(\alpha) > 2$, by adding new states we may easily replace this transition step by $lg(\alpha)-1$ equivalent moderate steps. This makes the description of the step less than 5 times as long as originally. So we have:

Lemma 2. *For each parser T, there is an equivalent moderate parser T' such that* $|T'| < 5\,|T|$. ■

3. Parsers with unbounded lookahead

In [15], Valiant shows that the relative succinctness that may be achieved by describing deterministic context-free languages by unambiguous context-free grammars rather than by deterministic pushdown automata is not bounded by any recursive function. What Valiant proved, can be restated as follows:

Proposition 3 [15]. *There is an infinite family $Q = \{G_i\}$ of unambiguous grammars such that each $L(G_i)$ is a deterministic language with the following property: If F is a function such that, for each i, there is a dpda D_i accepting the language $L(G_i)$ and $|D_i| \leq F(|G_i|)$, then F cannot be recursive.*

This is proved using an idea (by Hartmanis [7]) of encoding large Turing machine computations in small context-free grammars. To construct an element of Q, let M be a (deterministic) Turing machine that eventually halts when started on a blank input tape. If x and y are instantaneous descriptions (ID) of T such that y follows from x by application of the transition function of M, then we write $y = Next_M(x)$. Let x_0 be the ID for the starting configuration with blank input tape and let $\$$, a and b be symbols outside the alphabet describing the ID's. Let L' be the set of strings of the form

$$\$x_0\$x_1\$ \cdots \$x_n\$,$$

where x_n is a halting ID of M, such that $x_{2k+1}^R = Next_M(x_{2k})$ for all k, $0 \leq k \leq (n-1)/2$. Let L'' be the set of strings of the same form under the different restriction that $x_{2k} = Next_M(x_{2k-1}^R)$ for all k, $1 \leq k \leq n/2$. Finally let $L = L'a \cup L''b$.

Since L', L'' are both recognized by dpda's of size recursive in the size of M, languages $L'a$ and $L''b$ are also both so recognizable and therefore both generated by unambiguous grammars (say G_a and G_b, respectively) of similar size. Since these languages are disjoint their union L is also generated by an unambiguous grammar G of size recursive in the size of M. For each M, set Q contains the grammar G.

There is only one string x such that xa and xb both are in L. String x describes the halting computation of M with blank input. Hence

$$x = \$y_0\$y_1^R\$ \cdots \$y_z^{(R)}\$ \tag{3}$$

where the $y_0, y_1, \cdots, y_z$ are the ID's in the halting computation. This implies (see [15] for the details) that L can be recognized by a dpda M'. Let y be the shortest string such that M' accepts both ya and yb. Then the number of modes of M' must be $\geq(\log|y|)^k$ for some constant k [15: Lemma 4]. However, $y=x$ and thus $|y|>z$. Then the number of modes of M', and hence the size of M', cannot be recursively bounded in the size of G since z is not so bounded. This completes the proof of Proposition 3.

By modifying this proof we can show that there is no recursive function bounding the succinctness gained if parsable grammars are used instead of parsers for them.

Theorem 4. *There is an infinite family $R = \{G_i\}$ of (left and right) parsable grammars with the following property: If F is a function such that for each i, there is a left or right parser D_i for grammar G_i and $|D_i| \leq F(|G_i|)$, then F cannot be recursive.*

Proof. Clearly, instead of parsers it suffices to consider dpda's D_i for languages $L(G_i)$. The proof proceeds as that of Proposition 3 if we first show that each grammar G included in the set Q of Proposition 3 can be chosen to be left and right parsable.

It is not difficult to see how to generate L' and L'' by two LL(1) grammars (c.f. the grammar given in [8, p. 713]). Thus we can find LL(1) grammars, say G' and G'', for $L'a$ and $L''b$. We may assume that the sets of nonterminals of these grammars do not intersect. Let G be the union of G' and G''. Grammar G has productions $\{S \to S', S \to S''\} \cup P' \cup P''$ where S' and S'' are the start symbols and P' and P'' are the production sets of G' and G'', respectively.

Grammar G is LL(k) when $k>|x|$ where x is as in (3). For each Turing machine M that eventually halts when started on a blank input, set R contains the grammar G. Then each grammar G in R is both left and right parsable because G is LL(k) and therefore also LR(k). This completes our proof since the size of G is again recursively (even polynomially) bounded in the size of M. ■

In the above proof of Theorem 4 we constructed grammars G such that each G is LL(k) and thus also LR(k) for some k, but k varies with G, of course. It is easy to see that the length of the lookahead needed to parse grammars in R is not recursively bounded in the the size of the grammars. On the other hand, to obtain the result of Theorem 4 we needed no grammars that are outside the class LL.

4. Left parsability with bounded lookahead

Our results on the size of left parsers for grammars that are left parsable with bounded lookahead, that is, for LL(k) grammars, are based on properties of a sequence of grammars $G_n = (N_n, \Sigma_n, P_n, A_0)$, $n=1,2,\cdots$, where

$$\begin{array}{lll} P_n\!: & A_i \rightarrow a_{i+1}A_{i+1}B_{i+1} \mid d_{i+1}A_{i+1}C_{i+1} & (0 \le i \le n-1) \\ & A_n \rightarrow b_i \mid e & (1 \le i \le n) \\ & B_i \rightarrow b_i c_i \mid e & (1 \le i \le n) \\ & C_i \rightarrow c_i \mid e & (1 \le i \le n) \end{array}$$

Grammar G_n is of size $|G_n| = 17n+2$. In addition:

Lemma 5. *For every $k \ge 2$, each G_n is an LL(k) grammar.* ■

The proof of the lemma is left to the reader. Also note that G_n is not LL(1) or LR(1) or strong LL(2) or LALR(2) and that language $L(G_n)$ is finite. The standard LL(2) parser construction algorithm [1] gives more than 2^n LL(2) tables because the nonterminal A_n alone occurs in 2^n different LL(k) contexts.

We will show that the size of *every* left parser for G_n must be at least an exponential function of $|G_n|$. This will be done by proving the stronger result that every right parser for G_n must be at least exponentially larger than G_n, and by noting the following lemma:

Lemma 6. *If G_n has a left parser of size t then it has a right parser of size $< 2t$.*

Proof. The Lemma is true because G_n left-to-right covers itself, that is, there is a homomorphism h between production sequences of G_n such that the right parse of a string in $L(G_n)$ is a homomorphic image of the left parse of the same string. Therefore, to get a right parser we must only augment a left parser for G_n with evaluation of h. A suitable cover homomorphism h on the productions of G_n is

$$\begin{aligned}
&h(A_n \to b_i) = (A_n \to b_i),\\
&h(A_n \to e) = (A_n \to e),\\
&h(B_i \to b_i c_i) = (B_i \to b_i c_i,\ A_{i-1} \to a_i A_i B_i),\\
&h(B_i \to e) = (B_i \to e,\ A_{i-1} \to a_i A_i B_i),\\
&h(C_i \to c_i) = (C_i \to c_i,\ A_{i-1} \to d_i A_i C_i),\\
&h(C_i \to e) = (C_i \to e,\ A_{i-1} \to d_i A_i C_i),
\end{aligned}$$

where $i = 1,2,\dots,n$. For the remaining productions the value of h equals e.

Let dpdt T be a left parser for G_n. Modify T as follows: Whenever $\delta(q,a,Z) = (q',\alpha,\gamma)$ in T, replace γ by $h(\gamma)$. The resulting dpdt T' is a right parser for G_n. Since always $lg(h(\gamma)) \le 2 \cdot lg(\gamma)$, it finally follows that $|T'| < 2 \cdot |T|$. ■

Theorem 7. *There exists a constant $c > 0$ such that, when $n > 10$, any moderate right parser for G_n has size at least 2^{cm} where $m = |G_n|$.*

Proof. The proof is based on the following simple idea. We consider parsing strings of the form $x_1 x_2 \cdots x_n b_i c_i$ in $L(G_n)$ where x_i equals a_i or d_i. The right parse of such a string begins with $A_n \to b_i$ or $A_n \to e$ depending on whether $x_i = d_i$ or $x_i = a_i$. If the current state q and the topmost stack symbol Z after reading x_n can always tell whether $x_i = d_i$ or $x_i = a_i$, then the number of different modes (q,Z) is immediately seen to be exponentially large in $|G_n|$, and the Theorem follows. Otherwise the parser must exponentially often consult the stack below the topmost element. But then it turns out that the information popped from the stack cannot be ignored because it is needed later in parsing. To save this information the number of modes should be exponentially large.

We now formalize the above argument. Let $T = (Q, \Sigma_n \cup \{\$\}, \Gamma, P_n, \delta, q_0, Z_0, F)$ be a moderate right parser for G_n. We will show that $\#(Q \times \Gamma) > 2^{n/10}$ when $n > 10$. This proves the Theorem since then $|T| > 2^{n/20}$ from (2), which means, noting that $|G_n| = m$ is less than $18n$, that $|T| > 2^{m/360}$.

So we claim that $\#(Q \times \Gamma) > 2^{n/10}$. To derive a contradiction, assume

$$\#(Q \times \Gamma) \leq 2^{n/10}. \tag{4}$$

Denote by $C_{1/3}$ the set of those (state, stack content)-pairs that T reaches after reading a prefix of length $\lfloor n/3 \rfloor$ of some string in $L(G_n)$; note that the output of T must be empty at this moment. More formally, let $L_{1/3} = \{x_1 x_2 \cdots x_{\lfloor n/3 \rfloor} \mid x_i = a_i \text{ or } d_i\}$. Then

$$C_{1/3} = \{(q,\alpha) \mid (q_0, x, e, e) \vdash^* (q, e, \alpha, e) \text{ where } x \in L_{1/3}\}.$$

Clearly, for different x the corresponding elements (q,α) of $C_{1/3}$ must be different because then the languages accepted starting from state q and stack content α must differ. Hence

$$\# C_{1/3} \geq 2^{\lfloor n/3 \rfloor}. \tag{5}$$

Let $c = (q,\alpha)$ be a fixed element of $C_{1/3}$ and let $L_{2/3} = \{y_{\lfloor n/3 \rfloor + 1} \cdots y_n \mid y_i = a_i \text{ or } d_i\}$. Consider a move sequence

$$(q, y, \alpha, e) \vdash \cdots \vdash (q', e, \beta, e) \tag{6}$$

where y is in $L_{2/3}$. Sequence (6) is assumed to be maximal, that is, no transition is possible from the last configuration of (6). Again note that the output must still be empty in (6). Let now $(r,v,Z\gamma,e)$ be the configuration in sequence (6) whose stack $Z\gamma$ is of the lowest height; if there are more than one such configuration in (6), we let $(r,v,Z\gamma,e)$ be the first of them. The mode of this configuration, (r,Z), is called the *bottom* of (6) and denoted as $bottom(q,y,\alpha)$. Then sequence (6) can uniquely be written as

$$(q,y'v,\alpha,e) \vdash \cdots \vdash (r,v,Z\gamma,e) \vdash \cdots \vdash (q',e,\beta,e).$$

where $y'v=y$. The sequence starting from $(r,v,Z\gamma,e)$ is uniquely determined by r, v and Z, since the height of the stack remains at least as high as $lg(Z\gamma)$. Hence we have $\beta = \beta'\gamma$ for some β'. We say that string y' is the prefix of y that *leads* to the bottom. Denote then by $L_{(r,Z)}$ the set of of strings that lead to the bottom (r,Z), that is,

$$L_{(r,Z)} = \left\{ y' \;\middle|\; \begin{array}{l} \text{for some } y \in L_{2/3},\ bottom(q,y,\alpha) = (r,Z) \text{ and} \\ y' \text{ is the prefix of } y \text{ that leads to the bottom} \end{array} \right\}.$$

Lemma 8. $\#L_{(r,Z)} < 2^{n/10}$.

Proof. To derive a contradiction, assume that $\#L_{(r,Z)} \geq 2^{n/10}$. Given y' in $L_{(r,Z)}$ we may write for some v such that $y'v \in L_{2/3}$ and for some $x \in L_{1/3}$

$$\begin{aligned} (q_0,xy'v,e,e) &\vdash \cdots \vdash (q,y'v,\alpha,e) \\ &\vdash \cdots \vdash (r,v,Z\gamma,e) \\ &\vdash \cdots \vdash (q',e,\beta,e). \end{aligned} \tag{7}$$

Here y' is the prefix of $y'v$ that leads to the bottom. Since $lg(\alpha) \geq lg(Z\gamma)$, there must be before configuration $(q,y'v,\alpha,e)$ in (7) a last configuration

$$(s,w,Y\delta,e) \tag{8}$$

such that $lg(Y\delta) = lg(Z\gamma)$. It may happen that $(s,w,Y\delta,e) = (q,y'v,\alpha,e)$. Here the assumption that T is moderate is needed because it implies that every stack height $\leq lg(\alpha)$ must occur in (7). If $\#L_{(r,Z)} \geq 2^{n/10}$ and if we recall (4), we realize that then there must be two different elements y' and y'' of $L_{(r,Z)}$ such that the corresponding modes (s,Y) of (8) are the same for y' and y''. Hence for some suffixes x', x'' of the string x occurring in (7) we get

$$(s,x'y',Y\delta') \vdash \cdots \vdash (q,y',\alpha,e) \vdash \cdots \vdash (r,e,Z\delta',e) \tag{9}$$

and

$$(s,x''y'',Y\delta'') \vdash \cdots \vdash (q,y'',\alpha,e) \vdash \cdots \vdash (r,e,Z\delta'',e) \tag{10}$$

and the height of the stack in (9) is always $\geq lg(Y\delta') = lg(Z\delta')$ and in (10) always $\geq lg(Y\delta'') = lg(Z\delta'')$. Hence we may replace $x'y'$ in (9) by $x''y''$ and still obtain the same final configuration $(r,e,Z\delta',e)$. But this is a contradiction, since $y' \neq y''$ and therefore a replacement of $x'y'$ by $x''y''$ in (9) should also change the final configuration of (9) because the languages accepted as well as the parses omitted should change. But, as we noted, this did not occur, which completes the proof of Lemma 8.

The next lemma considers sets $L'_{(r,Z)}$ given by

$$L'_{(r,Z)} = \left\{ v \;\middle|\; \begin{array}{l} \text{for some } y',\ y = y'v \text{ is in } L_{2/3},\ bottom(q,y,\alpha) = (r,Z) \\ \text{and } y' \text{ is the prefix of } y \text{ that leads to the bottom} \end{array} \right\}.$$

While $L_{(r,Z)}$ was the set of prefixes that lead to bottom (r,Z), $L'_{(r,Z)}$ is the set of suffixes that lead from bottom (r,Z). Suppose that u and v are elements of $L'_{(r,Z)}$ such that $lg(u)=lg(v)$. Let y' and y'' be the prefixes of $y'u$ and $y''v$ that lead to bottom (r,Z). Then we clearly may choose $y'=y''$. This observation comes into use after proving the following lemma which gives an lower bound for the maximum number of elements in $L'_{(r,Z)}$ that have mutually the same length.

Lemma 9. *If $n > 10$, there exists a mode (r,Z) such that $L'_{(r,Z)}$ contains $> 2^{n/10}$ elements having the same length.*

Proof. We may represent $L_{2/3}$ as

$$L_{2/3} = \bigcup_{(r,Z)} L''_{(r,Z)}$$

where for each (r,Z), $L''_{(r,Z)} = \{uv \mid u \in L_{(r,Z)}, v \in L'_{(r,Z)}, lg(uv) = \lceil 2n/3 \rceil\}$ is a subset of $L_{(r,Z)}L'_{(r,Z)}$. Noting Lemma 8 we therefore obtain

$$\#L_{2/3} \le \sum_{(r,Z)} \#L''_{(r,Z)} < 2^{n/10} \cdot \sum_{(r,Z)} \#L'_{(r,Z)}.$$

If the Lemma were not true, then we should have $\#L'_{(r,Z)} \le \lceil 2n/3 \rceil \cdot 2^{n/10}$ for all (r,Z), since the length of all elements in $L'_{(r,Z)}$ is $\le \lceil 2n/3 \rceil$. But this leads to a contradiction since now, if $n > 10$,

$$\#L_{2/3} < 2^{2n/10} \cdot \lceil 2n/3 \rceil \cdot 2^{n/10} = 2^{3n/10 + \log_2 \lceil 2n/3 \rceil}$$
$$< 2^{\lceil 2n/3 \rceil}$$

which by the definition of $L_{2/3}$ cannot be true. This proves Lemma 9.

All the above considerations are with respect to a fixed element $c = (q,\alpha)$ of $C_{1/3}$. Thus we may conclude from Lemma 9 that for every such $c \doteq (q,\alpha)$, there exists a mode (r_c, Z_c) such that $L'_{(r_c,Z_c)}$ contains $> 2^{n/10}$ elements of the same length. That is, there is a string y' and more than $2^{n/10}$ disjoint strings v of equal length such that $y = y'v$ is in $L_{2/3}$ and y' is the prefix of y that leads to *bottom*$(q,y,\alpha) = (r_c, Z_c)$. Denote the common length of strings v by I_c. Observe that the moves reading v do not depend on the stack content below Z_c.

Thus we have chosen for each c in $C_{1/3}$ a triple (r_c, Z_c, I_c). Since the number of different triples is at most $2^{n/10} \cdot \lceil 2n/3 \rceil < 2^{\lceil n/3 \rceil}$, it follows from (5) that we must have $(r_c, Z_c, I_c) = (r_{c'}, Z_{c'}, I_{c'})$ for some disjoint $c = (q,\alpha)$, $c' = (q',\alpha')$ in $C_{1/3}$.

By the construction, then, there are strings x, x' in $L_{1/3}$, $x \ne x'$, and strings y, y' such that, for more than $2^{n/10}$ distinct v, strings yv and $y'v$ are in $L_{2/3}$ and

$$(q_0, xyv, e, e) \vdash^* (r_c, v, Z_c\alpha, e) \vdash^* (s, e, \beta\alpha, e), \tag{11}$$

$$(q_0, x'y'v, e, e) \vdash^* (r_c, v, Z_c\alpha', e) \vdash^* (s, e, \beta\alpha', e). \tag{12}$$

Since $x = x_1 \cdots x_{\lfloor n/3 \rfloor}$ and $x' = x_1' \cdots x_{\lfloor n/3 \rfloor}'$ are different, x_i and x_i' must differ for some i. Then we may assume, by symmetry, that $x_i = a_i$ and $x_i' = d_i$.

Suppose that the input to be read after (11) or (12) is $b_i c_i$. Then the first output following (11) should be production $A_n \to e$ and the first output following (12) should be production $A_n \to b_i$. But because the last configurations of (11) and (12) can differ only in stack portions α and α', the parser must always pop β from the stack to be able to choose between $A_n \to e$ and $A_n \to b_i$. However, s or β must vary with v because the output following $A_n \to e$ or $A_n \to b_i$ varies with v. Thus after popping β the mode of T must be different for each v. This is not possible because there are $> 2^{n/10}$ different strings v but by (4), only $\le 2^{n/10}$ different modes. We have a contradiction which completes the proof of Theorem 7. ∎

The restriction to moderate right parsers can be removed from Theorem 7.

Corollary 10. *There is a constant* $c' > 0$ *and an integer* n_0 *such that, when* $n > n_0$, *any right parser for* G_n *has size at least* $2^{c'm}$ *where* $m = |G_n|$.

Proof. If T is a right parser for G_n then, by Lemma 2, we can find a moderate right parser T' for G_n such that $|T| > |T'|/5$. Then, by Theorem 7, $|T| > 2^{cm}/5 = 2^{\log_2(1/5)+cm} \ge 2^{c'm}$ for some $0 < c' < \log_2(1/5)/m + c$. Such a constant c' clearly exists when m, that is n, is large enough. ∎

Our main technical result for left parsers follows in a similar way from Corollary 10 and Lemma 6:

Theorem 11. *There is a constant* $c'' > 0$ *and an integer* n_0' *such that when* $n > n_0'$, *any left parser for* G_n *has size at least* $2^{c''m}$ *where* $m = |G_n|$. ∎

Noting Lemma 5, this finally gives:

Theorem 12. *For each* $k \ge 2$, *there is an infinite family of LL(k) grammars such that the size of every left parser for these grammars grows at least* *exponentially in the size of the grammar.* ∎

It is well-known that the strong LL(k) grammars have left parsers with size polynomial in the size of the grammar. The strong LL(k) parser construction (e.g. [1]) gives parsers with parsing tables of size of the order $|G|^{k+1}$. The parsing table has a row for each nonterminal and terminal of G, totaling $<|G|$ rows, and a column for each of the k-symbol lookahead strings, totaling $<|G|^k$ columns.

If the table is interpreted as a dpdt, table entries correspond to transition function values. The length of each entry is $<|G|$ since an entry can be at most as long as the longest right hand side of the productions of G. Then the description of a strong LL(k) parser is of length $<|G|^{k+2}$, a polynomial in $|G|$. So we see that it is not possible to extend the result of Theorem 12 to the case $k \leq 1$, since if a grammar is LL(k) for $k \leq 1$, then it is also strong LL(k).

In addition to the strong LL(k) grammars we have another subclass of the LL(k) grammars having left parsers of polynomial size. It follows from a remark in [13, p. 230] that if an LL(k) grammar G has no e-productions (productions of the form $A \rightarrow e$), then G has a left parser with size polynomially bounded in $|G|$.

Theorem 12 does not say anything about the dependence of the lower bound of parser size on the length k of the lookahead. The only requirement is that $k \geq 2$. However, it is plausible that the lower bound should increase with k. As the minimum effect of k, there should be different states for encoding the different lookahead strings. The number of such strings grows exponentially in k.

We have no formal results in this direction. Only an example will be given where k has a strong effect on the size of (canonical) LL(k) and LR(k) parsers. Let grammars $G_{k,n} = (N_{k,n}, \Sigma_{k,n}, P_{k,n}, A_n)$ where $k \geq 2$, $n \geq 1$, have productions (1^{k-2} denotes a string of $k-2$ 1's)

$$
\begin{array}{lll}
P_{k,n}: & A_n \to A1 & \\
& A \to A_0 \mid 1^{k-2}B \mid e & \\
& A_i \to a_{i+1}A_{i+1}B_{i+1} \mid d_{i+1}A_{i+1}C_{i+1} & (0 \le i \le n-1) \\
& B \to b_i & (1 \le i \le n) \\
& B_i \to b_i c_i \mid e & (1 \le i \le n) \\
& C_i \to c_i \mid e & (1 \le i \le n).
\end{array}
$$

Grammar $G_{k,n}$ is of size $|G_{k,n}| = 17n+k+6$. Each $G_{k,n}$ is an LL(k) grammar. When $k \ge 3$, $G_{k,n}$ is not LR(k') for $k'<k$.

If the standard (canonical) LL(k) parser construction [1] is applied on $G_{k,n}$, we obtain a parser with more than $2^{(k-1)n}$ LL(k)-tables. The canonical LR(k) parser construction gives parsers also with more than $2^{(k-1)n}$ LR(k)-tables. This is because merely the nonterminal A occurs in $2^{(k-1)n}$ different LL(k) and LR(k) contexts.

Unfortunately, these parsers are considerably larger than necessary. Because grammars $G_{k,n}$ are both strong LL(k) and SLR(k), the special parser constructions for these classes can be applied. The strong LL(k) parser construction gives for $G_{k,n}$ a parser in which the number of LL(k) tables is only of the order n, and the SLR(k) parser construction gives a parser in which the number of LR(k) tables is only of the order $n+k$. We also note that the k-symbol lookahead is needed only when deciding whether or not the next production to be announced is $A \to e$: If the lookahead at this point is not any of the strings $1^{k-2}b_i1$ where $1 \le i \le n$, then the production $A \to e$ should be announced. Otherwise the next action is shifting a terminal. All the other parsing decisions can be based on at most one symbol lookahead. This means that the number of states of a dpdt needed to realize the use of the lookahead is only polynomially bounded in $n+k$. We conclude that the minimal left or right parser for $G_{k,n}$ is of polynomial size in $n+k$.

5. Right parsability with bounded lookahead

The grammars G_n of the previous section are LR(2). Then we immediately obtain from Corollary 10 the following counterpart of Theorem 12 for right parsers.

Theorem 13. *For each $k \geq 2$, there is an infinite family of LR(k) grammars such that the size of every right parser for these grammars grows at least exponentially in the size of the grammar.* ■

Grammars G_n are not LR(0) or LR(1). To complete our study of parser size in the LR(0) and LR(1) cases we use a sequence of right regular grammars $Q_n = (N_n, \Sigma_n, P_n, S)$ where

$$
\begin{array}{lll}
P_n: & S \to A_i & (1 \leq i \leq n) \\
 & A_i \to a_j A_i & (1 \leq i \neq j \leq n) \\
 & A_i \to a_i B_i \mid b_i & (1 \leq i \leq n) \\
 & B_i \to a_j B_i \mid b_i & (1 \leq i, j \leq n)
\end{array}
$$

The size $|Q_n| = m$ equals $6n^2 + 5n$. Earley [2] established grammars Q_n as an example where the size of the standard LR(0) parser grows exponentially with n. An LR(0) parser for Q_n has at least 2^{cn} states for some $c > 0$. The size of every LR(k) as well as SLR(k) or LALR(k) parser for Q_n is thus $\geq 2^{c'\sqrt{m}}$ for some $c' > 0$.

We will show that the size of *any* right parser for Q_n must be at least of the order $2^{\sqrt{m}}$. First a result analogous to Theorem 7 is given.

Theorem 14. *There is a constant $c > 0$ and an integer n_0 such that when $n > n_0$, any moderate right parser for Q_n has size at least $2^{c\sqrt{m}}$ where $m = |Q_n|$.*

Proof. The idea of the proof resembles to that of the proof of Theorem 7. We restrict ourselves to considering strings of the form $x_1 x_2 \cdots x_{n-1} a_n^j b_i$ in $L(Q_n)$, where each $x_1, \cdots, x_{n-1}$ is in $\{a_1, \cdots, a_{n-1}\}$, j is ≥ 0, and $1 \leq i \leq n-1$. The right parse of such a string begins with $B_i \to b_i$ or $A_i \to b_i$ depending on whether or not a_i occurs in $x_1 \cdots x_{n-1}$. Any parser cannot emit this first parse element earlier than when reading b_i. If after reading the last a_n the current mode of a parser can always tell whether or not $x_1 \cdots x_{n-1}$ contains a_i for any $1 \le i \le n-1$, the number of modes must be at least of the order $2^{\sqrt{m}}$ and the Theorem follows. Otherwise the parser must sometimes consult the stack below the topmost element. Then the parser should pop the stack portion corresponding to a_n^j. However, the popped information, which depends on j, cannot be ignored since it is needed later in parsing. Hence the number of modes is again at least of the order $2^{\sqrt{m}}$.

To formalize the above argument, let $T = (Q, \Sigma_n \cup \{\$\}, \Gamma, P_n, q_0, Z_0, F)$ be a moderate right parser for G_n. Because of (2) it suffices to show that $\#(Q \times \Gamma) > 2^{n/2}$ when $n>1$. To derive a contradiction, assume

$$\#(Q \times \Gamma) \le 2^{n/2}. \tag{13}$$

Denote by X a maximal subset of $\{x_1 x_2 \cdots x_{n-1} \mid x_i \in \{a_1, \cdots, a_{n-1}\}, 1 \le i \le n-1\}$ such that if x and x', $x \ne x'$, are in X then some symbol a_i occurs in x or x' but not both in x and x'.

First fix a string $x \in X$. Then $x a_n^j b_i$ is in $L(G_n)$ for each $j \ge 0$, $1 \le i \le n$. Hence T has a move sequence $(q_0, x, e, e) \vdash^* (q, e, \alpha, e)$, and a subsequent sequence

$$(q, a_n^j, \alpha, e) \vdash \cdots \vdash (q_j, e, \beta_j, e) \tag{14}$$

for each $j \ge 0$. The *bottom* of (14) is defined as in the proof of Theorem 7. Then, since a_n^j is a prefix of $a_n^{j'}$, $j < j'$, there is j_0 such that for all $j > j_0$, the bottom of (14) is the same and is achieved after reading $a_n^{j_0}$ but before reading $a_n^{j_0+1}$. Denote this particular bottom, which is unique for each x, by (r_x, Z_x).

Now, since $\#X > 2^{n/2}$, set X must contain distinct strings x and x' such that $(r_x, Z_x) = (r_{x'}, Z_{x'})$. Then, by the construction, there are for each $k \ge 0$ move sequences

$$(q_0, xa_n^{j_0}a^k, e, e) \vdash^* (r_x, a^k, Z_x\alpha, e) \vdash^* (s, e, \beta\alpha, e) \tag{15}$$

$$(q_0, x'a_n^{j_0'}a^k, e, e) \vdash^* (r_x, a^k, Z_x\alpha', e) \vdash^* (s, e, \beta\alpha', e) \tag{16}$$

Suppose that symbol a_i occurs in only one of strings x, x', say in string x, and assume that the input to be read after (15) and (16) is b_i.

Then the first output following (15) should be production $B_i \to b_i$ and the first output following (16) should be production $A_i \to b_i$. But because the last configurations of (15) and (16) can differ only in stack portions α and α', the parser must always pop β from the stack to be able to to choose between $B_i \to b_i$ and $A_i \to b_i$. However, s or β must vary with k because the output following $B_i \to b_i$ or $A_i \to b_i$ varies with k. Thus after popping β the mode of T must be different for each $k=0,1,\cdots$. This is not possible because by (13), there are only $\leq 2^{n/2}$ different modes. This contradiction proves the Theorem. ∎

Using Lemma 2 we again get:

Corollary 15. *There is a constant $c'>0$ and an integer n_0' such that when $n>n_0'$, any right parser for Q_n has size at least $2^{c'\sqrt{m}}$ where $m = |Q_n|$.* ∎

Since each Q_n is an LR(0) grammar, the main result of this section follows.

Theorem 16. *For each $k\geq 0$, there is an infinite family of LR(k) grammars (as well as SLR(k) and LALR(k) grammars) and a constant c such that the size of every right parser for these grammars grows at least as $2^{c\sqrt{m}}$ where m is the size of the grammar.* ∎

The growth rate $2^{c\sqrt{m}}$ given in this theorem is the largest we have been able to derive for parsers of LR(0) and LR(1) grammars. Note, however, that there are LR(0) grammars for which the standard LR(0) parser construction gives parsers with size growing exponentially in m (and not only in $\sqrt{m}$). An example of such a family is $\{R_n\}$, $n=1,2,\cdots$, where grammar R_n has productions (A_0 is the start symbol)

$$\begin{array}{ll} A_{i-1} \to 1A_i a_{i-1} & (1 \le i \le n) \\ A_n \to 1A_0 a_n & \\ A_i \to 0A_i a_i & (1 \le i \le n) \\ A_i \to 0A_0 a_i & (1 \le i \le n) \\ A_0 \to a & \end{array}$$

Each R_n is an LR(0) grammar of size $12n+6$. Grammars R_n are closely related to an example of [10] which shows an exponential succinctness between non-deterministic and deterministic finite automata.

The LR(0) parser construction gives for R_n more than 2^{n+1} LR(0) tables. This can be seen by showing that for any nonempty subset of

$$\{[A_0 \to 1 \cdot A_1 a_0], [A_1 \to 1 \cdot A_2 a_1], \cdots, [A_n \to 1 \cdot A_0 a_n]\}$$

there must be an LR(0) table whose essential items are exactly the items appearing in the subset.

On the other hand, R_n has a strict deterministic parser or a production prefix parser with size only linear in $|R_n|$. Such parsers do not have the correct prefix property. Thus $\{R_n\}$ is an extreme example of a grammar family where the correct prefix property strongly increases the parser size. The increase is of the order 2^{cm} where m denotes the size of the grammar. In the literature, Geller, Graham and Harrison [3] give examples where the increase is $2^{c\sqrt{m}}$.

6. Conclusion

We have shown first, that the size of a parser cannot be recursively bounded in the size of the grammar to be parsed. Hence there is no algorithm that constructs a parser for every parsable context-free grammar.

Using certain example families of LL(k) and LR(k) grammars, we have also shown that no matter what parsing method is used, the size of a parser cannot always be polynomially bounded in the size of the grammar although the grammar were parsable with lookahead of fixed length. The following table summarizes these results.

Grammar class	$k=0,1$	$k \geq 2$
LL(k)	*poly*	2^{cm}
strong LL(k)	*poly*	*poly*
e –free LL(k)	*poly*	*poly*
LR(k)	$2^{c\sqrt{m}}$	2^{cm}
e –free LR(k)	$2^{c\sqrt{m}}$	$2^{c\sqrt{m}}$
SLR(k)	$2^{c\sqrt{m}}$	$2^{c\sqrt{m}}$
LALR(k)	$2^{c\sqrt{m}}$	$2^{c\sqrt{m}}$

Table 1. Lower bounds on parser size

Each table entry gives a bound for parser size. The bounds for the class LL(k) and for its subclasses concern with left parsers, the bounds for the class LR(k) and for its subclasses concern with right parsers. The entry '*poly*' means that the specified class has parsers of polynomial size. The other entries give a worst-case lower bound for the parser size. Thus $2^{c\sqrt{m}}$ in the last entry of the SLR(k) row means that for each $k \geq 2$, there exists a constant $c > 0$ and an infinite family of SLR(k) grammars such that the size of every right parser for these grammars grows at least as $2^{c\sqrt{m}}$ where m is the size of the grammar. All bounds on the first three rows as well as the last bound on the LR(k) row are from Section 4. The remaining results are proved in Section 5.

It is obvious that the grammatical structures essential for our results are probably useless in context-free grammars that describe real programming languages. Purdom [12] even gives evidence that such grammars have LR-type parsers whose size grows only linearly. To establish this formally it would be interesting to find grammatical conditions for polynomial size parsability, that are more precise than those given in Table 1. Another problem for further research is to improve the lower bounds in Table 1 or to show that they are best possible.

Acknowledgment. I am indebted to Seppo Sippu for turning my attention to the problem considered in this work.

References

[1] Aho,A.V. and J.D.Ullman: *The Theory of Parsing, Translation, and Compiling. Vol. I: Parsing.* Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[2] Earley, J.: An efficient context-free parsing algorithm. Ph.D. Thesis, Carnegie-Mellon Univ., Pittsburgh, 1968.

[3] Geller,M.M., S.L.Graham and M.A.Harrison: Production prefix parsing (extended abstract). Automata, Languages, and Programming (J.Loeckx, ed.), Lecture Notes in Computer Science 14, pp. 232-241, Springer-Verlag, 1974.

[4] Geller,M.M., H.B.Hunt,III, T.G.Szymanski and J.D.Ullman: Economy of description by parsers, DPDA's and PDA's. Theoretical Computer Science 4 (1977), 143-153.

[5] Harrison,M.A.: *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Mass., 1978.

[6] Harrison,M.A. and I.M.Havel: On the parsing of deterministic languages. Journal of the ACM 21 (1974), 525-548.

[7] Hartmanis,J.: Context-free languages and Turing machine computations. Proc. Symposium Appl. Math., Vol. 19, Am. Math. Soc., Providence, RI, 1967, pp. 42-51.

[8] Hunt,H.B.,III, T.G.Szymanski and J.D.Ullman: On the complexity of LR(k) testing. Comm. ACM 18:12 (1975), 707-716.

[9] Knuth,D.E.: On the translation of languages from left to right. Information and Control 8 (1965), 607-639.

[10] Meyer,A.R. and M.J.Fisher: Economy of description by automata, grammars and formal systems. Proc. of the 12th Ann. IEEE Symp. on Switching and Automata Theory (1971), 188-190.

[11] Pittl,J.: Negative results on the size of deterministic right parsers. Proc. 10th Int. Symp. on Mathematical Foundations of Computer Science. Springer Lect. Notes in Computer Science, 1981.

[12] Purdom,P.: The size of LALR(1) parsers. BIT 14 (1974), 326-337.

[13] Rosenkrantz,D.J. and R.E.Stearns: Properties of deterministic top-down grammars. Information and Control 17 (1970), 226-256.

[14] Ukkonen,E.: On size bounds for deterministic parsers. Proc. 8th Int. Colloquium on Automata, Languages and Programming (Acre). Lect. Notes in Computer Science, Vol. 115, Springer-Verlag, 1981, pp. 218-228.

[15] Valiant,L.G.: A note on the succinctness of descriptions of deterministic languages. Information and Control 32 (1976), 139-145.