Theoretical Computer Science 242 (2000) 59–69

Theoretical Computer Science

www.elsevier.com/locate/tcs

On the size of parsers and LR(k)-grammars

Hing Leung[a,*,1], Detlef Wotschke[b]

[a] *Department of Computer Science, New Mexico State University, NM 88003, USA*

[b] *Fachbereich Informatik, J.W. Goethe-Universität, 60054 Frankfurt am Main, Germany*

Received August 1996; revised July 1998

Communicated by K. Culik II

Abstract

In this paper, we consider two tradeoff results regarding the economy of description in parsing. One result is on the tradeoff between the size of a parser and its ability to detect an error early. The other result is on the tradeoff between the size of an LR(k)-grammar and the length k of the lookahead. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Parsing; Grammar; Descriptional complexity

1. Introduction

When constructing or building a compiler, parser *size* is one of the most important considerations. As is to be expected and as is well-known from practical experience, parser size and parser performance are often conflicting features. For example, early error recovery is an important and highly desired feature. However, very often early error recovery has to blow up the size of the parser. The theoretical foundation for this experimentally well-known fact was established by Geller et al. [2] by showing that the price for early error recovery might be exponential compared to a parser which can delay error recovery arbitrarily long.

Specifically, Geller et al. [2] compared the sizes of unrestricted parsers and correct prefix parsers. A correct prefix parser is a parser which detects an error as soon as the symbols read so far, together with the next symbol in the input buffer, do not form a prefix of any string in the language. Intuitively, a correct prefix parser always detects an error at the earliest moment and will not attempt to read past the point where the error first occurred. In contrast, an unrestricted parser which is not a correct prefix parser is allowed to detect an error at a position that is arbitrarily far away from

* Corresponding author.
E-mail address: hleung@nmsu.edu (H. Leung).
[1] The research was supported by an Alexander von Humboldt research fellowship and was done while this author was visiting the J.W. Goethe-Universität, Germany.

the point where the error first occurred. It was shown that the size of correct prefix parsers could be exponentially larger than that of unrestricted parsers for parsing the same languages. Another study in the sizes of syntax driven parsers with early error detection capabilities is by Moura [3].

In this paper we try to lay the theoretical foundation for the intuitively expected fact that, between immediate and arbitrarily delayed error recovery, there must be a hierarchy of trade-offs between parser size and delay in error recovery. In other words, there must be examples where one has a choice between the two extremes of immediate and arbitrarily delayed error recovery and thus between exponentially larger or smaller parsers. This hierarchy lays the theoretical foundation that, as is often the case in practice, there is a hierarchy of choices of slowly increasing the speed of error recovery at the expense of slowly increasing parser size.

Given a parser for a certain language, we say that the parser detects an error with position delay d if for each invalid string, the position where the parser detects an error lies at most d positions behind the position where the error first occurred. Thus, a correct prefix parser is the same as a parser with delay 0.

In Section 2, we present a sequence of languages $\mathscr{L}_n$ such that there is an increasing tradeoff in the size of the parsers as the delay in error detection position varies.

In particular, our result shows that when there is no delay requirement to satisfy, the smallest parser for $\mathscr{L}_n$ can be constructed with size at most n^3, whereas the smallest correct prefix parser can be constructed with size at most n^{n+3} and to require at least $n^{n/2}/(\sqrt{2}(n+1))$ size. In other words, this special case of our result is very close in nature to the result obtained by Geller et al. [2]. Moreover, the method used in [2] is a difficult combinatorial analysis on the behavior of correct prefix parsers. In comparison, our method is a lot simpler technically.

Fuessel [1] studied other measures of delays in error detection on the basis of so-called “rejecting” DPDAs.

The second part of this paper deals with LR(k)-grammars and LR(k)-parsing.

LR(k)-grammars are important and basic concepts for parsers. There are many advantages of LR(k)-parsers over other kinds of parsers [4]. They can parse in linear time. LR(k)-parsers are correct prefix parsers which detect an error at the earliest moment. The class of LR(1)-grammars denotes the set of all deterministic context-free languages. It is known that the use of a longer lookahead k cannot increase the expressive power of LR(k)-grammars. They still denote the set of deterministic context-free languages.

We would like to ask whether the use of a longer lookahead could reduce the size of the grammar for denoting the same language. This question will be answered affirmatively in the sense that there do exist some languages for which such trade-offs between grammar size and lookahead are possible. Admittedly, the languages used here are of somewhat theoretical nature. On the other hand, they look very much like palindrome languages and thus carry a lot of resemblance to “real-life” context-free languages.

In Section 3, we present a sequence of languages L_n such that there is a progressive tradeoff in the size of the LR(k)-grammars for denoting the same language as the length of the lookahead varies. A conclusion is given in Section 4.

2. Parsers

Given a context-free grammar, we consider a parser for the grammar to be a deterministic pushdown transducer that outputs a parse trees on valid inputs, and returns error messages otherwise.

Formally, the size of a parser is defined to be the length of the parser description, that is, the number of symbols used to specify it [2].

In order to simplify the technical discussions in the following subsections, we identify a parser with its corresponding deterministic pushdown automaton (DPDA). Furthermore, we assume that the stack can grow at most one symbol in one move. Note that to convert any DPDA into a DPDA that grows at most one symbol in one move will only increase the size by a linear factor.

Let m denote the product of the number of states in the finite state control and the cardinality of the stack alphabet. It has been shown [2] that the size of a parser that can grow at most one symbol on the stack in one move is $O(m^2)$. Since we are dealing with bigger than polynomial tradeoff results, for technical reasons, we redefine the size of a parser to be m instead of measuring the size of a parser strictly by the number of symbols used in the specification.

Let $\Sigma_n = \{a_1, \ldots, a_n\}$. Let $\mathscr{L}_n$ be the language

$$\{a_{i_1} \ldots a_{i_n} a_j a_{i_j} a_{i_n} \ldots a_{i_1} \mid 1 \leqslant i_1, \ldots, i_n,\ j \leqslant n\} \subseteq \Sigma_n^{2n+2}.$$

Remark. Σ_n^{2n+2} denotes the set of all strings over Σ_n that are of length $2n+2$.

Theorem 1. *For $d \in \{0, 1, \ldots, n\}$, there exists a parser accepting $\mathscr{L}_n$ with position delay d in error detection and size $= O(n^{n-d+3})$.*

Proof. Consider the context-free grammar $G_n = (N, \Sigma_n, P, S_1)$, where

$$N = \{S_i \mid 1 \leqslant i \leqslant n\} \cup \{S_{i,j,k} \mid 1 \leqslant j < i \leqslant n+1, 1 \leqslant k \leqslant n\}$$

and the set of productions P are

(i) $S_i \to a_k S_{i+1} a_k$, for $1 \leqslant i \leqslant n-1$ and $1 \leqslant k \leqslant n$,
(ii) $S_i \to a_k S_{i+1,i,k} a_k$, for $1 \leqslant i \leqslant n$ and $1 \leqslant k \leqslant n$,
(iii) $S_{i,j,k} \to a_h S_{i+1,j,k} a_h$, for $1 \leqslant j < i \leqslant n$ and $1 \leqslant h, k \leqslant n$,
(iv) $S_{n+1,j,k} \to a_j a_k$ for $1 \leqslant j \leqslant n$ and $1 \leqslant k \leqslant n$.

It can be easily verified that G_n generates $\mathscr{L}_n$. We are going to construct parsers for G_n.

Let us first focus on the design of a parser with delay n. By the definition of $\mathscr{L}_n$, an error can occur at the earliest in the $(n+2)$-th position. A delay n will allow the parser to report an error after the whole input string of length $2n+2$ has been read. That is, there is no special requirement with respect to the delay in the position of error detection that the parser has to satisfy.

We can try to modify a “usual” design of a parser for accepting the language of a palindrome $\{a_{i_1} \ldots a_{i_n} a_{i_n} \ldots a_{i_1}\}$ to accept our language $\mathscr{L}_n$. The modifications involve remembering the $(n+2)$-th symbol in the finite state control and setting up a "counting process" when the $(n+1)$-th symbol a_j has been read. While the parser is performing the job of verifying the palindrome structure of the input, the counting process will locate the symbol a_{i_j} as the stack is being popped. The $(n+2)$-th symbol remembered will then be checked against the symbol a_{i_j} found. The modifications presented can be implemented in a parser with $O(n^2)$ states and n stack symbols. Hence, the size of the parser is $O(n^3)$.

For parsers with delay $d \in \{0, 1, \ldots, n-1\}$, we add more features into the previous design with delay n for $\mathscr{L}_n$. The new parser will have to remember the first $n-d$ symbols of the input in the finite state control besides pushing them onto the stack. Recall that the $(n+1)$-th symbol is denoted by a_j. If j is not bigger than $n-d$, then the parser will verify, as soon as the $(n+2)$-th symbol has been read, whether the $(n+2)$-th symbol is the same as symbol a_{i_j} which has been remembered in the finite state control. However, if j is bigger than $n-d$ then the parser will operate just as in the previous design. The parser contructed is of delay d. The number of states in the finite state control will be blown up $(n-d)$ times by a factor of n. Hence, the size is $O(n^{n-d+3})$. $\square$

We want to show that any parser (DPDA) detecting an error with position delay $d \in \{0, 1, \ldots, n\}$ and recognizing $\mathscr{L}_n$ requires $\Omega(n^{(n-d-2)/2})$ size.

Let us define the term *reduced configuration*. Given a configuration of a parser which is the ordered triple (current state, stack contents, unprocessed input), a corresponding "reduced configuration" is defined to be (current state, top stack symbol, length of unprocessed input).

Consider a given string $x = a_{i_1} \ldots a_{i_{n-d}}$. We define $T_x = \{y \in \Sigma_n^{2d+2} \mid x\,y\,x^R \in \mathscr{L}_n\}$.

For each $y \in T_x$, consider the sequence of configurations that the parser enters for processing the accepted string $x\,y\,x^R$. We say that a configuration in the sequence is "working" on x if the corresponding unprocessed input consists of a suffix of x (which may not have to be proper, and which could also be an empty string) followed by $y\,x^R$; and a configuration in the sequence is "working" on y if the corresponding unprocessed input consists of a *proper* suffix of y (which could be an empty string) followed by x^R. Note: the subsequence of configurations that are "working" on x is the same independent of the choice of y. We define $C_{x,y}$ to be the configuration with the minimum stack height over all the configurations in the sequence which are "working" on y. If there is more than one candidate for the minimum, then just pick an arbitrary one.

We define C_x to be the configuration with the minimum stack height over all $C_{x,y}$ for $y \in T_x$. Again, pick any one if there is more than one candidate. let $C_{x,y'}$ be the winning configuration, that is $C_x = C_{x,y'}$. We also denote y' by y_x. Let us focus our attention on those configurations that are "working" on x during the processing of the accepted string $x\,y_x\,x^R$ and which have a *shorter or equal* stack height as that of C_x; we then define D_x to be the *last* such configuration according to the ordering defined by the sequence of configurations.

Let $\mathscr{D}_x$ and $\mathscr{C}_x$ be the reduced configurations of D_x and C_x respectively.

Lemma 1. *For* $x_1 \neq x_2 \in \Sigma_n^{n-d}$, $(\mathscr{D}_{x_1}, \mathscr{C}_{x_1}) \neq (\mathscr{D}_{x_2}, \mathscr{C}_{x_2})$.

Proof. Assume the contrary. Suppose $x_1 \neq x_2$ and $(\mathscr{D}_{x_1}, \mathscr{C}_{x_1}) = (\mathscr{D}_{x_2}, \mathscr{C}_{x_2})$. Let x_1 be represented by $a_{i_1} \dots a_{i_{n-d}}$ and x_2 be represented by $a_{j_1} \dots a_{j_{n-d}}$. Let the length of the unprocessed input in D_{x_1} (and in D_{x_2}) be $2n+2-p$, where $p = 0, 1, \dots, n-d$, and the length of the unprocessed input in C_{x_1} (and in C_{x_2}) be $n-d+q$, where $q = 0, 1, \dots, 2d+1$.

Case 1: $a_{i_1} \dots a_{i_p} \neq a_{j_1} \dots a_{j_p}$.

Let x_1 and x_2 differ in the hth index, that is $a_{i_h} \neq a_{j_h}$, where $1 \leqslant h \leqslant p$. Then given

$$a_{i_1} \dots a_{i_p} \ (a_{j_{p+1}} \dots a_{j_{n-d}} z \, a_h \, a_{j_h} \, z^R) x_2^R,$$

where $z \in \Sigma_n^d$, the parser, which is a DPDA, will not be able to report the error which occurs at the $(n+2)$-th symbol, a_{j_h}, after the first $n+d+2$ input symbols have been processed. This is because of the ways $\mathscr{D}_x$'s are defined and the given condition $\mathscr{D}_{x_1} = \mathscr{D}_{x_2}$ that the processing of the substring $(a_{j_{p+1}} \dots a_{j_{n-d}} z \, a_h \, a_{j_h} \, z^R)$ in the above given input by the parser would not differ from the processing of the same substring in the input $x_2 \, z \, a_h \, a_{j_h} \, z^R \, x_2^R$ which is in $\mathscr{L}_n$. But the parser is supposed to have a delay of at most d only; hence, a contradiction.

Case 2: $a_{i_1} \dots a_{i_p} = a_{j_1} \dots a_{j_p}$ and $a_{i_{p+1}} \dots a_{i_{n-d}} \neq a_{j_{p+1}} \dots a_{j_{n-d}}$.

Since the parser can grow at most one symbol on the stack in one move and $p < n-d$ by the assumption of case 2, the stack height of D_{x_1} and the stack height of C_{x_1} must be the same; similarly, the stack height of D_{x_2} is the same as that of C_{x_2}. Let y_{x_1} and y_{x_2} be denoted by $a_{\alpha_1} \dots a_{\alpha_{2d+2}}$ and $a_{\beta_1} \dots a_{\beta_{2d+2}}$ respectively. Then the parser will accept the input

$$a_{i_1} \dots a_{i_p} \ (a_{j_{p+1}} \dots a_{j_{n-d}} a_{\beta_1} \dots a_{\beta_{2d+2-q}}) a_{\alpha_{2d+3-q}} \dots a_{\alpha_{2d+2}} a_{i_{n-d}} \dots a_{i_1}$$

because it cannot distinguish this input from $x_1 y_{x_1} x_1^R \in \mathscr{L}_n$ by the facts that $\mathscr{D}_{x_1} = \mathscr{D}_{x_2}$, $\mathscr{C}_{x_1} = \mathscr{C}_{x_2}$, the stack height of D_{x_1} equal to the stack height of D_{x_2} and the stack height of C_{x_1} equal to the stack height of C_{x_2}. However, this is a contradiction since the above input should not be in the language $\mathscr{L}_n$ since $a_{i_{n-d}} \dots a_{i_1} \neq (a_{i_1} \dots a_{i_p} a_{j_{p+1}} \dots a_{j_{n-d}})^R$. $\square$

Theorem 2. *Any parser with position delay* $d \in \{0, 1, \dots, n\}$ *in error detection and recognizing* $\mathscr{L}_n$ *requires* $\Omega(n^{(n-d-2)/2})$ *size.*

Proof. Let us denote the size of a given parser by S. Given a reduced configuration $\mathscr{D}_x$, the number of possible values for the length of the unprocessed input is $n-d+1$; hence the number of different possible $\mathscr{D}_x$ is $(n-d+1)S$, which is bounded by $(n+1)S$. Correspondingly, the number of possible values for the length of the unprocessed input in $\mathscr{C}_x$ is $2d+2$; hence the number of different possible $\mathscr{C}_x$ is $(2d+2)S$, which is bounded by $(2n+2)S$. From Lemma 1, the number of distinct $(\mathscr{D}_x, \mathscr{C}_x)$ must be greater than or equal to the number of different x's in Σ_n^{n-d}. Therefore,

$$(n+1)S(2n+2)S \geqslant n^{n-d}.$$

Thus $S \geqslant n^{n-d/2} / (\sqrt{2}(n+1))$ and the theorem follows. $\square$

The tradeoff results of Theorems 1 and 2 are summarized in the following corollary.

Corollary 1. *Any parser for $\mathscr{L}_n$ with position delay $n-k$ in error detection has size $n^{\Theta(k)}$.*

Proof. Let $d=n-k$. Then, by Theorem 1, the parser size with position delay d is $\mathrm{O}(n^{n-d+3})=\mathrm{O}(n^{k+3})=n^{\mathrm{O}(k)}$. By Theorem 2, the parser has size $\Omega(n^{(n-d-2)/2})=\Omega(n^{k/2-1})=n^{\Omega(k)}$. □

3. LR(k)-grammars

The size of a production in a grammar is defined to be the number of symbols on the right-hand side of the production plus the number of symbols on the left-hand side. In the case of a context-free production, the left-hand side has only one symbol.

The size of a grammar is defined to be the summation over the size of all productions. Therefore, an upper bound on the size of a grammar can be obtained by computing the product of the number of productions and the length of the longest production.

Define the language $L_n \subseteq \{0,1\}^{3n+1}$ to be $L_n^0 \cup L_n^1$ where

$$L_n^0=\{a_1\dots a_n a_n \dots a_1 0 b_1 \dots b_n \mid a_1,\dots,a_n,b_1,\dots,b_n \in \{0,1\}\},$$

$$L_n^1=\{a_1\dots a_n b_1 \dots b_n 1 a_n \dots a_1 \mid a_1,\dots,a_n,b_1,\dots,b_n \in \{0,1\}\}.$$

To generate the language L_n^1, we can construct a grammar G_n^1 with the start symbol S, the set of nonterminal symbols $\{S,D_1,\dots,D_{2n+1}\}$ and the following productions:
(i) $S \to D_1$,
(ii) $D_i \to aD_{i+1}a$, for $1 \leqslant i \leqslant n$ and $a \in \{0,1\}$,
(iii) $D_{n+i} \to bD_{n+i+1}$, for $1 \leqslant i \leqslant n$ and $b \in \{0,1\}$,
(iv) $D_{2n+1} \to 1$.
The grammar is unambiguous. The corresponding right parser needs to produce its first output only when the $(2n+1)$-th symbol of the input is read. Also, it is easy to see that the grammar is LR(0). The number of productions is $4n+2$.

Theorem 3. *There exists an LR(0)-grammar $G_{n,0}$ generating L_n with the number of productions being 2^n+6n+3 and the longest production having length $2n+1$ on the right hand side.*

Proof (*sketch*). Let $G_{n,0}$ be $G_{n,0}^0 \cup G_n^1$ where $G_{n,0}^0$ is the grammar with the start symbol S, the set of nonterminal symbols $\{S,C,A_1,\dots,A_n\}$ and the following productions:
(i) $S \to CA_1$,
(ii) $C \to a_1 \dots a_n a_n \dots a_1 0$, for $a_1,\dots,a_n \in \{0,1\}$
(iii) $A_i \to bA_{i+1}$, for $1 \leqslant i \leqslant n-1$ and $b \in \{0,1\}$,
(iv) $A_n \to b$, for $b \in \{0,1\}$. □

Theorem 4. *For $1 \leqslant k \leqslant n$, there exists an LR$(k)$-grammar $G_{n,k}$ generating L_n with the number of productions being $2^{n-k+1} + 6n + 2k + 1$ and the longest production having length max$(2(n-k+1), 3)$ on the right hand side.*

Proof (*sketch*). Let $G_{n,k}$ be $G^0_{n,k} \cup G^1_n$ where $G^0_{n,k}$ is the grammar with the start symbol S, the set of nonterminal symbols $\{S, C_1, \dots, C_k, A_1, \dots, A_n\}$ and the following productions:

(i) $S \to C_1 0 A_1$,
(ii) $C_i \to aC_{i+1}a$, for $1 \leqslant i \leqslant k-1$ and $a \in \{0, 1\}$,
(Note: when $k = 1$, this group of productions is empty.)
(iii) $C_k \to a_k \dots a_n a_n \dots a_k$, for $a_k, \dots, a_n \in \{0, 1\}$,
(iv) $A_i \to bA_{i+1}$, for $1 \leqslant i \leqslant n-1$ and $b \in \{0, 1\}$,
(v) $A_n \to b$, for $b \in \{0, 1\}$. $\square$

Theorem 5. *Let $f(n,k)$ be $(1/n^2)2^{\frac{1}{4}(n-k)}$. For any LR$(k)$-grammar with $k = 0, 1, \dots, n-1$ generating L_n, where $n \geqslant 3$, the number of nonterminal symbols must be at least $f(n,k)$ or there exists a nonterminal symbol A such that the number of different productions with A on the left hand side must be at least $f(n,k)$.*

Proof. Assume the contrary that the number of nonterminal symbols is less than $f(n,k)$ and, for all nonterminals A, the number of different productions with A on the left hand side is also less than $f(n,k)$.

Let us introduce some terminology. Given a nonempty string x, we write $x[i]$ to denote the ith symbol of x. For $i \leqslant j$, we write $x[i:j]$ to denote the substring $x[i]x[i+1]\dots x[j]$ of x. If $i > j$, we write $x[i:j]$ to denote the string $x[i]x[i-1]\dots x[j]$. Thus x^R is the same as $x[|x|:1]$.

Given a parse tree of a string generated by an LR(k)-grammar and an internal node n of the tree, we can talk about "the subtree n" as the subtree consisting of n and the edges connecting to *all* its descendents. Moreover, when we say a subtree of the parse tree, we mean the complete subtree that can be identified by some node n; otherwise, we would call it a "partial subtree" instead.

For each string $w \in \{0,1\}^n$, we consider the parse tree constructed by the LR(k)-parser for the string $s(w) = ww^R00^n = w[1:n]w[n:1]00^n \in L_n$.

We are going to define two subtrees (see Fig. 1) in this parse tree of $s(w)$. The first one is the *biggest* subtree such that its leaves cover terminal symbols of $s(w)$ beginning from some symbol (nonempty) of the first n symbols of $s(w)$, that is from $w[1:n]$, to some symbol (nonempty) from the substring $w[n:k+1]$ of the second n symbols of $s(w)$, that is from $w[n:1]$; but the leaves do not cover any other symbols from the rest of $s(w)$, that is from $w[k:1]00^n$. Note that such a subtree may not exist. If it exists, then it is unique and we identify this subtree by $T_1(w)$ which can be interpreted as the root node of the subtree. We denote the nonterminal symbol at $T_1(w)$ by $N_1(w)$. We denote the number of symbols that $T_1(w)$ covers on w, the first n symbols of $s(w)$,

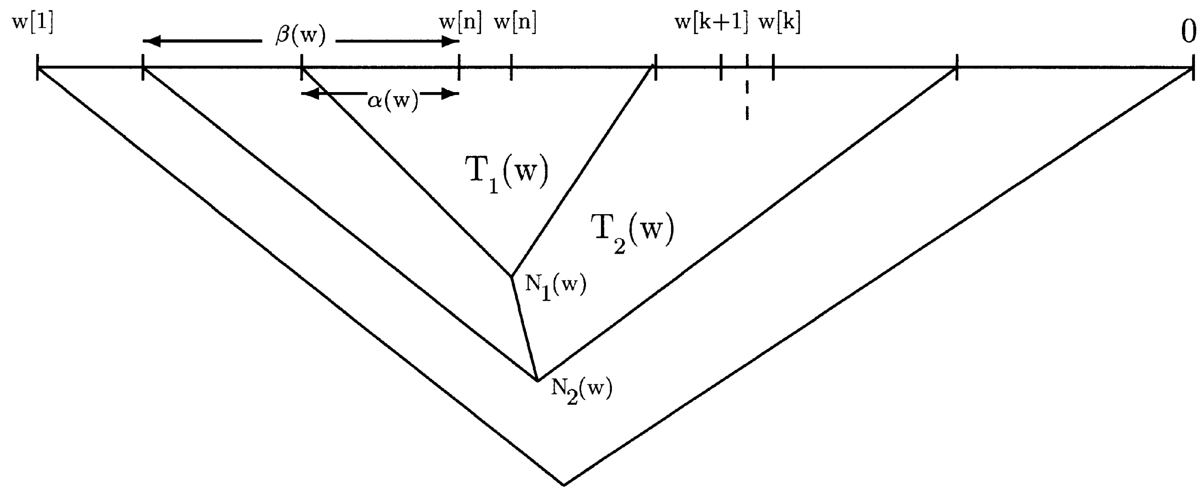

Fig. 1. Parse tree for string $s(w) = w[1:n]w[n:1]00^n$.

by $\alpha(w)$ where $\alpha(w) \leqslant n$. In case $T_1(w)$ is not defined, for technical reason, we would still want to define $\alpha(w)$ to be of the value 0.

As the second subtree we define the *smallest* subtree such that its leaves cover terminal symbols beginning from some symbol (nonempty) of the first n symbols of $s(w)$, that is from $w[1:n]$, and cover all the terminal symbols from the substring $w[n:k+1]$ of the second n symbols of $s(w)$; the leaves may or may not cover terminal symbols from the rest of $s(w)$, that is, from $w[k:1]00^n$. Unlike $T_1(w)$, such a subtree always exists. It is also unique. We identify the subtree by $T_2(w)$ with the corresponding nonterminal symbol being $N_2(w)$. $T_1(w)$, if it exists, is indeed a subtree of $T_2(w)$. It is possible that $T_1(w)$ and $T_2(w)$ are the same. We denote the number of symbols that $T_2(w)$ covers on w, the first n symbols of $s(w)$, by $\beta(w)$ where $\alpha(w) \leqslant \beta(w) \leqslant n$; that is, the symbols in the substring $w[n-\beta(w)+1:n]$ of the first n symbols of $s(w)$ are covered.

Therefore, each $w \in \{0,1\}^n$ corresponds to an ordered pair $(\alpha(w), \beta(w))$. By simple counting, taking into account that $\alpha(w)$ could be 0, the total number of different possible ordered pairs is $n + n(n+1)/2$, which is less than or equal to n^2 for $n \geqslant 3$. With the cardinality of $\{0,1\}^n$ being 2^n, there must exist an ordered pair which corresponds to at least $2^n/n^2$ strings in $\{0,1\}^n$. We denote the value $2^n/n^2$ by $g(n)$. We denote the ordered pair corresponding to at least $g(n)$ strings in $\{0,1\}^n$ by (α_0, β_0). We define S_0 to be the set $\{w \in \{0,1\}^n \mid (\alpha(w), \beta(w)) = (\alpha_0, \beta_0)\}$. Thus, $|S_0| \geqslant g(n)$.

Case 1: $\alpha_0 \geqslant (n-k)/4$.

Note that $\alpha_0 > 0$. Define $S_2 = \{w_2 \in \{0,1\}^{\alpha_0} \mid w = w_1 w_2 \in S_0\}$. Then $|S_2| \geqslant g(n)/2^{n-\alpha_0}$. Since $\alpha_0 \geqslant (n-k)/4$, we have $|S_2| \geqslant f(n,k)$. However, the number of nonterminal symbols is assumed to be less than $f(n,k)$. Therefore, there must exist two different strings $w', w'' \in S_0$ such that $N_1(w') = N_1(w'')$ and $w'[n-\alpha_0+1:n] \neq w''[n-\alpha_0+1:n]$. Let us focus on the actions of the LR(k) parser on the strings $s(w') = w'w'[n:1]00^n$ and $w'w'[n:1]1w'[n:1]$. Both strings are in the language L_n. Hence, there exist valid parse trees for the two strings. Since the strings match up to the first $2n$ symbols, the parser actions on both strings should be the same until the $(2n+1)$-th symbol is "seen" by the k-lookahead. We know that the subtree $T_1(w')$ has already been constructed after the shifting of the first $2n-k$ symbols of $s(w')$. Yet the $(2n+1)$-th symbol of $s(w')$ could only be "seen" after the $(2n-k+1)$-th symbol has been shifted. Thus, with respect to the string $w'w'[n:1]1w'[n:1]$, the same subtree $T_1(w')$ would have also been constructed after the first $2n-k$ symbols are shifted. Therefore, by replacing the subtree $T_1(w')$ in the parse tree of the string $w'w'[n:1]1w'[n:1]$ by the subtree $T_1(w'')$ from the parse tree of $s(w'')$, which is possible since $N_1(w')=N_1(w'')$, we obtain a parse tree. Consider the string obtained from the leaves of the parse tree. We claim that this string is not in the language and hence a contradiction. First, the string may not have $3n+1$ symbols. If the string is of length $3n+1$, then note that the replacement only affects the first $(2n-k)$ symbols of the string and the $(2n+1)$-th symbol is still the symbol 1. However, the last n symbols of the string are not a reversal of the first n symbols by the condition $w'[n-\alpha_0+1:n] \neq w''[n-\alpha_0+1:n]$.

Case 2: $\beta_0 < 3(n-k)/4$.

Define $S_4 = \{w_4 \in \{0,1\}^{n-k-\beta_0} \mid w = w_3w_4w_5 \in S_0, w_5 \in \{0,1\}^{\beta_0}\}$. Note that $n-k-\beta_0 > 0$. Then $|S_4| \geqslant g(n)/2^{k+\beta_0}$. Since $\beta_0 < 3(n-k)/4$, we have $|S_4| > f(n,k)$. However, the number of nonterminal symbols is assumed to be less than $f(n,k)$. Therefore, there must exist two different strings $w', w'' \in S_0$ such that $N_2(w') = N_2(w'')$ and $w'[k+1:n-\beta_0] \neq w''[k+1:n-\beta_0]$. Therefore, by replacing the subtree $T_2(w')$ in the parse tree of the string $s(w')$ by the subtree $T_2(w'')$ from the parse tree of $s(w'')$, which is possible since $N_2(w') = N_2(w'')$, we obtain a parse tree. As in case 1, we claim that this string is not in the language and hence a contradiction. Again the string may not have $3n+1$ symbols. If the string is of length $3n+1$, then note that the $(2n+1)$-th symbol is still the symbol 0. However, the last $n-k$ symbols of the first n symbols of the string are not a reversal of the first $n-k$ symbols of the second n symbols of the string after the replacement.

Case 3: $\alpha_0 < (n-k)/4$ and $\beta_0 \geqslant 3(n-k)/4$.

Define $S_7 = \{w_7 \in \{0,1\}^{\beta_0-\alpha_0} \mid w = w_6w_7w_8 \in S_0, w_8 \in \{0,1\}^{\alpha_0}\}$. Note that if α_0 is 0, then w_8 is the empty string. Then $|S_7| \geqslant g(n)/2^{n-(\beta_0-\alpha_0)}$. Since $\beta_0 - \alpha_0 > (n-k)/2$, we have $|S_7| > 2^{(n-k)/4} f(n,k)$. However, the number of nonterminal symbols is assumed to be less than $f(n,k)$. Therefore, there must exist a nonterminal symbol A such that the number of elements in the set $S_{7,A} = \{w_7 \in S_7 \mid w = w_6w_7w_8 \in S_0, w_8 \in \{0,1\}^{\alpha_0}, N_2(w) = A\}$ is greater than $2^{(n-k)/4}$, hence greater than $f(n,k)$. By assumption, the number of productions with A as the left hand side is less than $f(n,k)$. Thus, there exist two different strings $w', w'' \in S_0$ such that $w'[n-\beta_0+1:n-\alpha_0] \neq w''[n-\beta_0+1:n-\alpha_0]$ and the same production appears at the roots of the subtrees $T_2(w')$ and $T_2(w'')$.

Let the production at the root of the subtrees $T_2(w')$ and $T_2(w'')$ be denoted by $A \to A_1 \dots A_m$, where $m \geqslant 2$ and for $1 \leqslant p \leqslant m$, A_p could either be a nonterminal or terminal symbol; note that m cannot be 1, otherwise $T_2(w')$ and $T_2(w'')$ could not be the *smallest* trees that satisfy their definitions. We use the term "the subtree A_p" of $T_2(w')$ to denote the subtree identified by the node with label A_p, which is the p-th immediate descendant of $T_2(w')$. Similarly, we can define the subtree A_p of $T_2(w'')$.

Case 3.1: $T_1(w')$ and $T_1(w'')$ exist.

$T_1(w')$ is a subtree of $T_2(w')$. Moreover, by the assumption of case 3, $T_1(w')$ is then a proper subtree of $T_2(w')$. Consider the path p_1 from the root of the subtree $T_2(w')$ to the $(n-\alpha_0)$-th terminal symbol of $s(w')$, and the path p_2 from the root of the subtree $T_2(w')$ to the $(n-\alpha_0+1)$-th terminal symbol of $s(w')$. Note that the $(n-\alpha_0+1)$-th symbol of $s(w')$ belongs to both the subtrees $T_1(w')$ and $T_2(w')$, whereas the $(n-\alpha_0)$-th symbol of $s(w')$ only belongs to $T_2(w')$ but not $T_1(w')$.

We claim that the paths p_1 and p_2 intersect only at the root node. If the paths p_1 and p_2 intersect at a node which also belongs to the subtree $T_1(w')$, then $T_1(w')$ should cover the $(n-\alpha_0)$-th terminal symbol of $s(w')$, which is a contradiction. If the paths p_1 and p_2 intersect at a node which does not belong to the subtree $T_1(w')$ and which is also not the root node of $T_2(w')$, then either $T_1(w')$ is not the biggest subtree or $T_2(w')$ is not the smallest subtree satisfying their definitions.

Therefore, there exists a $1 \leqslant q' < m$ such that the partial subtree which consists of the root $T_2(w')$ and the subtrees $A_1, \ldots, A_{q'}$ of $T_2(w')$ as the immediate descendents, would cover exactly the terminal symbols beginning from the $(n-\beta_0+1)$-th symbol to the $(n-\alpha_0)$-th symbol of $s(w')$ as the leaves. Similarly, there exists a $1 \leqslant q'' < m$ such that the partial subtree which consists of the root $T_2(w'')$ and the subtrees $A_1, \ldots, A_{q''}$ of $T_2(w'')$ as the immediate descendants, would cover exactly the terminal symbols beginning from the $(n-\beta_0+1)$-th symbol to the $(n-\alpha_0)$-th symbol of $s(w'')$ as the leaves.

Consider the parse tree for $s(w')$. With respect to the subtree $T_2(w')$ within the parse tree for $s(w')$, we are going to replace the subtrees $A_1, \ldots, A_{q''}$ of $T_2(w')$ by the corresponding subtrees $A_1, \ldots, A_{q''}$ from $T_2(w'')$. We then obtain a new parse tree. However, the string generated is not in the language L_n. This could be due to two possible reasons. If $q' \neq q''$, it is possible that the generated string is not of length $3n+1$, hence not in L_n. Another possibility is that the first n symbols are not a reversal of the second n symbols. This is because we have replaced the substring from the $(n-\beta_0+1)$-th symbol to the $(n-\alpha_0)$-th symbol of $s(w')$ by the corresponding portion in $s(w'')$, and it has already been established that $w'[n-\beta_0+1 : n-\alpha_0] \neq w''[n-\beta_0+1 : n-\alpha_0]$.

Case 3.2: $T_1(w')$ and $T_1(w'')$ do not exist.

The arguments are analogous to that in case 3.1 except that we omit the discussion of $T_1(w')$ and $T_1(w'')$. For example, we now consider the path p_1 from the root of the subtree $T_2(w')$ to the n-th terminal symbol of $s(w')$, and the path p_2 from the root of the subtree $T_2(w')$ to the $(n+1)$-th terminal symbol of $s(w')$. Again, we can argue that the paths p_1 and p_2 intersect only at the root node of subtree $T_2(w')$; if not, we can show that either $T_1(w')$ exists or $T_2(w')$ is not the smallest subtree satisfying its definition. □

Note that in the proof above, we do not rely on any sophisticated properties of LR(k) parsing. We only need to recall the fact that LR(k) parsing is deterministic and outputs a right parse in reverse order.

The tradeoff results of Theorems 3, 4 and 5 are summarized in Corollary 2.

Corollary 2. *Let $n \geqslant 2$ and $0 \leqslant k \leqslant n - 9\lg n$. Any LR($k$)-grammar for L_n has size $2^{\Theta(m)}$ where $m = n - k$.*

Proof. Combining Theorems 3 and 4, there exists an LR(k)-grammar generating L_n with the number of productions at most $2^{n-k+1} + 6n + 2k + 3$ and each production having length at most $2(n-k)+3$. Given that $k \leqslant n - 9\lg n$, we have $m = n - k \geqslant 9\lg n$ and thus $k \leqslant n = 2^{\lg n} = O(2^m)$. The grammar size is bounded above by $(2(n-k)+3)(2^{n-k+1}+6n+2k+3) = (2(n-k)+3)(2\cdot 2^{(n-k)}+6n+2k+3) = O(m)(O(2^m)+O(2^m)+O(2^m)) = O(m)O(2^m) = 2^{O(m)}$. By Theorem 5, the size of an LR(k)-grammar generating L_n has size at least $f(n,k) = 2^{(n-k)/4-2\lg n} = 2^{((n-k)-8\lg n)/4} = 2^{((n-k)/9+8(n-k)/9-8\lg n)/4} \geqslant 2^{((n-k)/9)/4} = 2^{m/36} = 2^{\Omega(m)}$ since $n - k \geqslant 9\lg n$. □

4. Conclusion

In this paper, we have presented tradeoff results in economy of description for parsers when the ability for early error detection varies and for LR(k)-grammars when the length of the lookahead varies. One main contribution of this paper are the new proof techniques developed.

Note that $\mathscr{L}_n$ (Section 2) and L_n (Section 3) are finite languages for all positive integers n. Tradeoff results are obtained for these finite languages. Let us consider two families of infinite languages, $\mathscr{L}_n^*$ and L_n^*. It can be easily seen that the tradeoff results presented in Section 2 still hold for $\mathscr{L}_n^*$, and tradeoff results presented in Section 3 still hold for L_n^*.

As further study, we would recommend a similar study as in Section 3 for LL(k)-grammars.

Acknowledgements

We would like to thank Rainer Kemp, Andreas Weber and Hans-Martin Fuessel for their valuable comments and discussions. We also thank the anonymous referees for their valuable suggestions which helped improve the presentation of this paper.

References

[1] H.M. Fuessel, Descriptional complexity of early error detection for parsing deterministic context-free languages (German title: Beschreibungskomplexitaet einer fruehen Fehlerkennung beim Parsen von deterministisch kontextfreien Sprachen), Master Thesis (Diplomarbeit), University of Frankfurt, 1992.

[2] M. Geller, H. Hunt III, T. Szymanski, J. Ullman, Economy of description by parsers, DPDAs, and PDAs, Theoret. Comput. Sci. 4 (1977) 143–153.

[3] A. Moura, Early error detection in syntax-driven parsers, IBM J. Res. Dev. 30 (1986) 617–626.

[4] S. Sippu, E. Soisalon-Soininen, Parsing Theory, Vol. II: LR(k) and LL(k) Parsing, EATCS Monographs on Theoretical Computer Science, vol. 20, Springer, Berlin, 1990.