Proof of Time Bounds for Recognizing Unambiguous and LR(k) Grammars Using Earley's Algorithm

Matthew E. Wong
Yale University
wongm@cs.yale.edu

May 13, 1995

Introduction

This paper is meant to present a different proof of the time bounds for Earley's algorithm for parsing context-free grammars. Often these proofs are difficult to follow, and it is helpful to the reader to be able to see different approaches to a proof. These time bounds are proved in Earley's dissertation, but in many popular textbooks— including *Compilers: Principles, Techniques, and Tools* by Aho, Sethi, and Ullman, *The Theory of Parsing, Translation, and Compiling* by Aho and Ullman, and *The Language of Machines: An Introduction to Computability and Formal Languages* by Floyd and Beigel— the proofs are either incomplete or omitted altogether.

Earley's algorithm is an important example of dynamic programming. Like the Cocke-Younger-Kasami algorithm, Earley's algorithm breaks a parsing problem into many subproblems. Like the CYK algorithm, it is a general recognizer that works on any context-free grammar. Unlike the CYK algorithm, Earley's algorithm only attempts to solve subproblems that are feasible in the parsing. For this reason, the algorithm solves fewer subproblems when parsing unambiguous and $LR(k)$ grammars. This makes sense intuitively, since the algorithm outputs all of the parse trees for its input, and, given an unambiguous grammar, it will only output one tree. Furthermore, when parsing $LR(k)$ grammars, the algorithm only solves those subproblems that will actually be in the parse tree for its input (using a lookahead of at least k). In a sense, it builds the parse tree without wasting time building parts of trees that will not be in final parse tree; thus it runs in time directly proportional to the size of the parse tree.

In the following section, the terminology used in this paper is defined. Then, a simplified version of the algorithm is presented. This version does not use lookahead, which is only necessary to prove the linear time bound for $LR(k)$ grammars. Using the simplified algorithm, we will prove the $O(n^2)$ time bound for unambiguous grammars. Then, the complete algorithm will be introduced, which we will use to prove the $O(n)$ time bound for $LR(k)$ grammars.

Terminology

It is assumed that the reader is familiar with context-free languages. This section describes terminology that is specific to this paper.

We use capital letters for nonterminals. Strings of either terminals or nonterminals are represented by Greek letters. The empty string is λ. The end-of-file is denoted $\dashv$. α^k represents

$$\overbrace{\alpha \ldots \alpha}^{\text{k times}}$$

$|\alpha|$ denotes the number of symbols in α. For a production q, $|q|$ denotes the number of symbols on the right-hand side of q. The productions with a particular nonterminal D on their left sides are called the *alternatives* of D. D_q denotes the left-hand nonterminal of the production q, and C_{q_y} denotes the y^{th} character, terminal or nonterminal, of the right-hand side of q.

We write $\alpha \Rightarrow \beta$ if $\exists \gamma, \delta, \eta, A$ such that $\alpha = \gamma A \delta$ and $\beta = \gamma \eta \delta$ and $A \rightarrow \eta$ is a production in G. We write $\alpha \overset{*}{\Rightarrow} \beta$ (β is *derived* from α) if $\exists$ strings $\alpha_0, \alpha_1, \ldots, \alpha_m (m \geq 0)$ such that

$$\alpha = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_m = \beta$$

The sequence $\alpha_0, \ldots, \alpha_m$ is called a derivation (of β from α).

The *root* of a grammar is a designated nonterminal from which all strings in the language defined by the grammar can be derived. A *sentential form* of a grammar G is a string α such that α is derived from the root R of G. A *sentence* is a sentential form consisting entirely of terminal symbols. We may represent any sentential form in at least one way as a *parse tree*, reflecting the steps made in deriving it (though not the order of the steps).

A *recognizer* is an algorithm which takes as input a string and either *accepts* or *rejects* depending on whether the string is a sentence of the grammar. A *parser* is a recognizer which also outputs the set of all legal parse trees for the string. Input is written in boldface letters. Hereon the variables X and n are reserved for input: X denotes the input string $X_0 \dots X_{n-1}$ of length n.

A sentence is *unambiguous* if it can only be parsed in one way; that is, if it has only one parse tree. A grammar is *unambiguous* if each of its sentences is unambiguous. A grammar is *reduced* if every nonterminal appears in some derivation of some sentence. Hereafter we use "grammar" to mean "reduced context-free grammar without null productions".

Description of the Algorithm

The following is a simplified version of Earley's algorithm as a recognizer. The only difference between this algorithm and Earley's algorithm is that this algorithm does not use lookahead, while Earley's algorithm does. This simplified algorithm is the specific case of Earley's algorithm when the length of lookahead is set to zero.

The recognizer scans the input string $X_0 \dots X_{n-1}$ from left to right. As each symbol X_i is scanned, a list of states S_{i+1} is constructed which represents the condition of the recognition process at that point in the scan. Each state in the set holds information representing: (1) a production such that we are currently scanning a character that is part of a portion of the input string which is derived from the production's right side, (2) a point in that production which shows how much of the production's right side we have recognized so far, and (3) a pointer back to the position in the input string at which that instance of the production began.

This triple is represented (q, j, f), where q is a production, j is an integer representing the current position within the right side of q $(0 \leq j \leq |q| - 1)$, and f is the integer such that X_f is the first character in that instance of the production q.

To save time, we will keep an array S of length n to hold each state list, and each state list will be separated into $(N+1)$ sublists, where N is the number of nonterminals in the grammar. The sublist for a nonterminal D in state set S_i will be written S_{i_D}, and the extra sublist is written S_{i_t}. From hereon, whenever a state (q,j,f) is "added to S_i" it means that: if $j=|q|$ or $C_{q_{j+1}}$ is a terminal symbol, then add (q,j,f) to S_{i_t} (if it is not already there), else add (q,j,f) to S_{i_D}, where $D=C_{q_{j+1}}$ (if it is not already there).

To begin the algorithm, we put the single state $(\phi \rightarrow R, 0, f)$ in state set 0, where R is the root of the grammar. We then apply three operations, the *completer*, the *predictor*, and the *scanner*, in that order, to each state set, starting with S_0, until the predictor stops and either accepts or rejects. These three operations operate on a state set S_i as follows:

Completer: For each state (q,j,f) in S_i, if $j=|q|$ then, for each state $(p,y,f') \in S_{f_D}$ where $D=D_q$, add the state $(p,y+1,f')$ to S_i. As with the predictor, the completer is applicable to new states that it adds to S_i.

Predictor: If S_i is empty, then reject. If $i=n+1$ (i.e. the entire input has already been scanned), and $(\phi \rightarrow R, 1, 0) \in S_i$ then accept.

For each state (q,j,f) in S_i, if C_{q_j} is nonterminal, then, for each alternative q' of C_{q_j}, the state $(q',0,i)$ is added to S_i. Note that the predictor will also be applied to new states that it adds to S_i.

Scanner: For each state (q,j,f) such that $X_{i+1}=C_{q_{j+1}}$, add the state $(q,j+1,f)$ to S_{i+1}.

Proof of $O(n^2)$ Bound on Unambiguous Grammars

Lemma 1 *The number of states in any state set S_i is $O(i)$.*

Proof: For each state (q,j,f) there are $|G'|$ possible values for q; i possible values for f; and b possible values for j, where b is some number such that $b \geq |p|$ for all productions $p \in G'$. Thus there are ci possible states, where c is the constant $b(|G'|)(|\Sigma|^k)$.

Lemma 2 (String Completion) *On input string X, for any state (q,j,f) in some state set S_i, there exists a string β such that, on input $X'=X_0 \ldots X_{i-1}\beta$, the recognizer will accept, and in a parsing for X', $D_q \stackrel{*}{\Rightarrow} X'_f \ldots X'_{f'}$ via the production q, for some $f' \geq f$.*

That is, if the recognizer has only scanned up through X_i, and S_i contains any state (q, j, f), then there is always some string β such that, if the input string is $X_0 \ldots X_i\beta$, the recognizer will accept, and, in its parse tree, the production q derives some substring beginning with X_f.
Proof: The proof is by induction. The inductive hypothesis is that, for any state that has already been added to a state set, the lemma holds. We need to prove that the lemma holds for the next state to be added.

Base Case: Consider the first state $(\phi \rightarrow S, 0, 0)$ in S_0, the first state set. Since the language $L(G)$ is nonempty, there must exist at least one string X that can be parsed. Since ϕ is the root of the language, any parse of any string in G must have $\phi \overset{*}{\Rightarrow} X_0 \ldots X_{n-1}$.

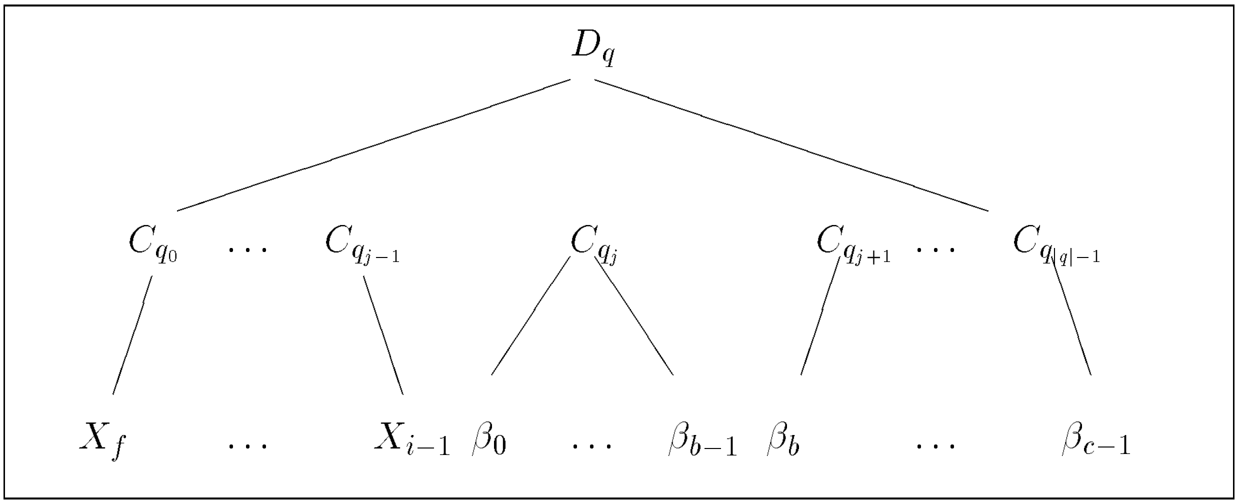
Inductive Case: The inductive case breaks into three subcases, depending on whether the next state is added by the predictor, the scanner, or the completer.

Subcase Predictor: Consider any state $(q', 0, i)$ added to state set S_i by the operation of the predictor on state (q, j, f) in S_i. By the inductive hypothesis, there exists a string β such that $X' = X_0 \ldots X_{i-1}\beta$ is in G, and $D_q \overset{*}{\Rightarrow} X'_f \ldots X'_{f'}$ for some $f' \geq f$. Then, in the parse tree for $X_0 \ldots X_{i-1}\beta$, for some numbers b, and c, we have:

$$
\begin{array}{rcll}
D_{q'} & = & C_{q_j} & \\
D_q & \Rightarrow & C_{q_0} \ldots C_{q_j} \ldots C_{q_{|q|-1}} & \\
C_{q_0} \ldots C_{q_{j-1}} & \overset{*}{\Rightarrow} & X_f \ldots X_{i-1} & \\
C_{q_j} & \overset{*}{\Rightarrow} & \beta_0 \ldots \beta_{b-1} & \text{via some production } \hat{q} \\
C_{q_{j+1}} \ldots C_{q_{|q|-1}} & \overset{*}{\Rightarrow} & \beta_b \ldots \beta_{c-1} & \\
D_q & \overset{*}{\Rightarrow} & X_f \ldots X_{i-1}\beta_0 \ldots \beta_{c-1} &
\end{array}
\qquad (1)
$$

If q' is the same production as $\hat{q}$ in (1), then $C_{q_j} \overset{*}{\Rightarrow} \beta_0 \ldots \beta_{b-1}$ via q', therefore $D_{q'} \overset{*}{\Rightarrow} X_i \ldots X_{i+b}$, and the theorem holds. If q' is different than the production in equation (1), then we need to find a string β' to satisfy the theorem. Let ω be some string of length s such that $C_{q_j} \overset{*}{\Rightarrow} \omega$ via the production q'. We simply replace $\beta_0 \ldots \beta_{b-1}$ ω, so $C_{q_j} = D_{q'} \overset{*}{\Rightarrow} \omega$. Then $\beta' = \omega\beta_b \ldots \beta_{|\beta|-1}$. This does not change the rest of the parsing, so the string $X'' = X_0 \ldots X_{i-1}\beta'$

Figure 1: Predictor Subcase: Conditions guaranteed by the inductive hypothesis



is in G, and we have $D_{q'} \stackrel{*}{\Rightarrow} X''_i \ldots X''_{i+|\omega|-1}$ via q'.

Subcase Scanner: Consider any state (q, j, f) added to S_i by operation of the scanner on state $(q, j-1, f)$ from the set S_{i-1}. By the inductive hypothesis, we have a string β such that $X_0 \ldots X_{i-2}\beta$ is in G, and in the parsing, $D_q \stackrel{*}{\Rightarrow} X'_f \ldots X'_{f'}$ via production q, where $X' = X_0 \ldots X_{i-2}\beta$. The production q begins with X_f, and ends at some $X_{f'}$ where $f' > i-1$, thus $f' = i+s$ for some $s \leq |\beta|$. Since it was added by the scanner, C_{q_j} must be a non-terminal that matches the next input character, $X_{i-1} = \beta_0$. Thus, letting $\beta' = \beta_1 \ldots \beta_{|\beta|-1}$, and $X'' = X_0 \ldots X_{i-1}\beta' = X'$ we have $X'' \in G$, and $D_q \stackrel{*}{\Rightarrow} X''_f \ldots X''_{f'}$ via q.

Subcase Completer: Consider any state (q, j, f) added to set S_i by the operation of the completer on some state $(p_1, |p_1|, f_1, X_i \ldots X_{i+k-1})$. By the inductive hypothesis, we have β such that $X' = X_0 \ldots X_{i-1}\beta \in G$ and $D_{p_1} \stackrel{*}{\Rightarrow} X'_{f_1} \ldots X'_{f'_1}$ via production p_1. Since $D_{p_1} = C_{q_j}$, then the production q implicitly must be used in the parse for X', and the production q began at X_f. Thus, letting $\beta' = \beta$, $X'' = X'$, we have $X'' \in G$, and $D_q \stackrel{*}{\Rightarrow} X''_f \ldots X_{f''}$

Figure 2: Predictor Subcase: The parse tree after substituting β' for β

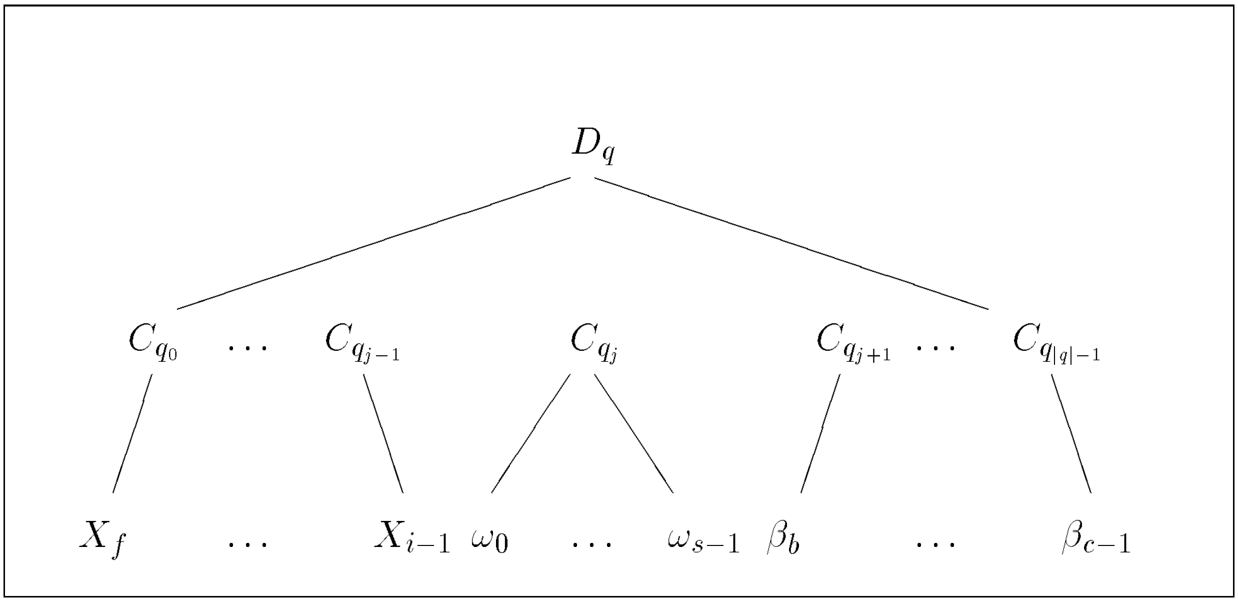


Figure 3: Scanner Subcase: Conditions guaranteed by the inductive hypothesis

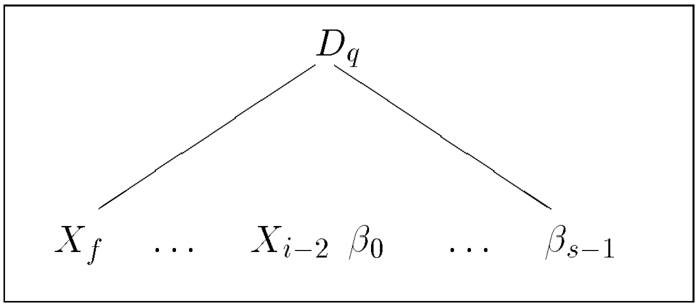
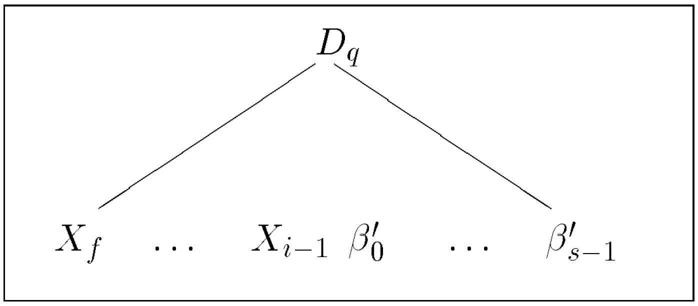


Figure 4: Scanner Subcase: The parse tree after substituting β' for β



for $f'' > f$. This completes the proof of the lemma.

Theorem 1 *For any unambiguous grammar G and any input $X \in G$, the completer will never, while processing a state set, attempt to add a state more than once.*

Proof: The proof is by contradiction. Suppose that, in processing state set S_i, the completer tried to add some state $(q, j+1, f)$ more than once. Then the following conditions must be true:
1. S_i contains the following two states:
   $s_1 = (p_1, |p_1|, f_1)$
   $s_2 = (p_2, |p_2|, f_2)$ such that $f_1 \leq f_2$.
2. $D_{p_1} = C_{q_j} = D_{p_2}$. Call this nonterminal A.
3. S_{f_1} and S_{f_2} both contain state (q, j, f). Call this state s_0. Thus we have

S_{f_1}	S_{f_2}	S_i
$(D_q \rightarrow C_{q_0} \ldots C_{q_{j-1}} A \ldots, j, f)$	$(D_q \rightarrow C_{q_0} \ldots C_{q_{j-1}} A \ldots, j, f)$	$(A \rightarrow \alpha_1, \lvert p_1 \rvert, f_1)$ $(A \rightarrow \alpha_2, \lvert p_2 \rvert, f_1)$

and either $f_1 \neq f_2$ or $p_1 \neq p_2$, or else s_1 and s_2 would be the same state. Now, let β be the string promised by String Completion, such that $X_0 \ldots X_{i-1}\beta$ is in G, and $D_q \stackrel{*}{\Rightarrow} X'_f \ldots X'_{f'}$. Then $C_{q_{j+1}} \ldots C_{q_{|q|-1}} \stackrel{*}{\Rightarrow} \beta_0 \ldots \beta_{r-1}$ for some r, $0 \leq r \leq |\beta|$. Then, in the completed parse tree for $X_0 \ldots X_{i-1}\beta$, we have

$$\begin{aligned} D_{p_1} = \quad D_{p_2} &= \quad C_{q_j} \\ D_q \quad &\stackrel{*}{\Rightarrow} \quad X_f \ldots X_{i-1}\beta_0 \ldots \beta_{r-1} \\ A = D_{p_1} \quad &\stackrel{*}{\Rightarrow} \quad X_{f_1} \ldots X_{i-1} \end{aligned}$$

$$A = D_{p_2} \quad \stackrel{*}{\Rightarrow} \quad X_{f_2} \dots X_{i-1}$$

Case 1: $p_1 \neq p_2$. Then there are two possible parsings for input $X_0 \dots X_{i-1}\beta$:

$$\begin{aligned} C_{q_j} &\stackrel{*}{\Rightarrow} X_{f_1} \dots X_{i-1} \qquad \text{via } p_1\text{, and} \\ D_q &\stackrel{*}{\Rightarrow} X_f \dots X_{i-1}\beta_0 \dots \beta_{r-1} \end{aligned}$$

or

$$\begin{aligned} C_{q_j} &\stackrel{*}{\Rightarrow} X_{f_2} \dots X_{i-1} \qquad \text{via } p_2\text{, and} \\ D_q &\stackrel{*}{\Rightarrow} X_f \dots X_{i-1}\beta_0 \dots \beta_{r-1} \end{aligned}$$

Case 2: $p_1 = p_2$, and therefore $f_1 \neq f_2$. Then we have the following two parsings for the substring $X_f \dots X_{i-1}$ on input $X\beta$:

$$\begin{aligned} D_q &\stackrel{*}{\Rightarrow} X_f \dots X_{i-1}\beta_0 \dots \beta_{r-1} \\ C_{q_0} \dots C_{q_{j-1}} &\stackrel{*}{\Rightarrow} X_f \dots X_{f_1-1} \\ C_{q_j} &\stackrel{*}{\Rightarrow} X_{f_1} \dots X_{i-1} \end{aligned}$$

or

$$\begin{aligned} D_q &\stackrel{*}{\Rightarrow} X_f \dots X_{i-1}\beta_0 \dots \beta_{r-1} \\ C_{q_0} \dots C_{q_{j-1}} &\stackrel{*}{\Rightarrow} X_f \dots X_{f_2-1} \\ C_{q_j} &\stackrel{*}{\Rightarrow} X_{f_2} \dots X_{i-1} \end{aligned}$$

This concludes the proof.

Theorem 2 *For any unambiguous grammar G, the algorithm recognizes $L(G)$ in time $O(n^2)$.*

Proof: From the previous theorem, we know that the completer will never attempt to add a state twice. Since the number of states in a state set is bounded by i, and the completer never attempts to add a state more than once, the completer adds $O(i)$ states per state set, each of which takes constant time to add, for a run time of $O(i)$ per state set. The predictor operation runs in time $O(k)$ per state set, because there is a fixed number of productions— thus a fixed number of states it can add to any set— and each addition takes constant time. The scanner runs in at most time $O(i)$ per state set, because it performs at most one operation per state, and the number of states that it scans is $O(i)$. Summing the operations over n state sets, the recognizer runs in $O(n^2)$ time.

Figure 5: Theorem 1, Case 1: ($p_1 \neq p_2$) The two parse trees. In the top tree, $A \stackrel{*}{\Rightarrow} X_{f_1} \ldots X_{i-1}$ via p_1. In the bottom tree, $A \stackrel{*}{\Rightarrow} X_{f_1} \ldots X_{i-1}$via p_2.

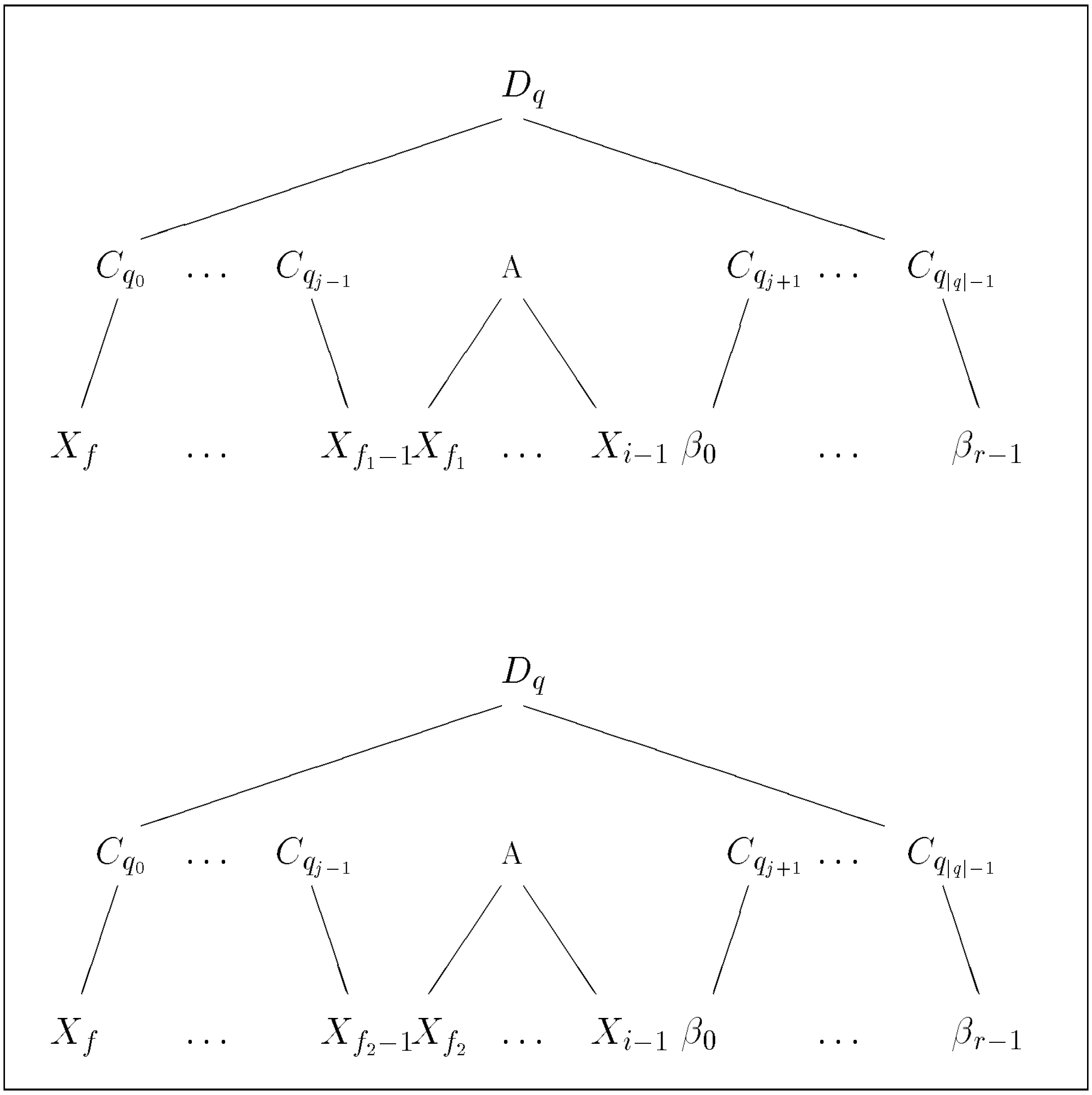
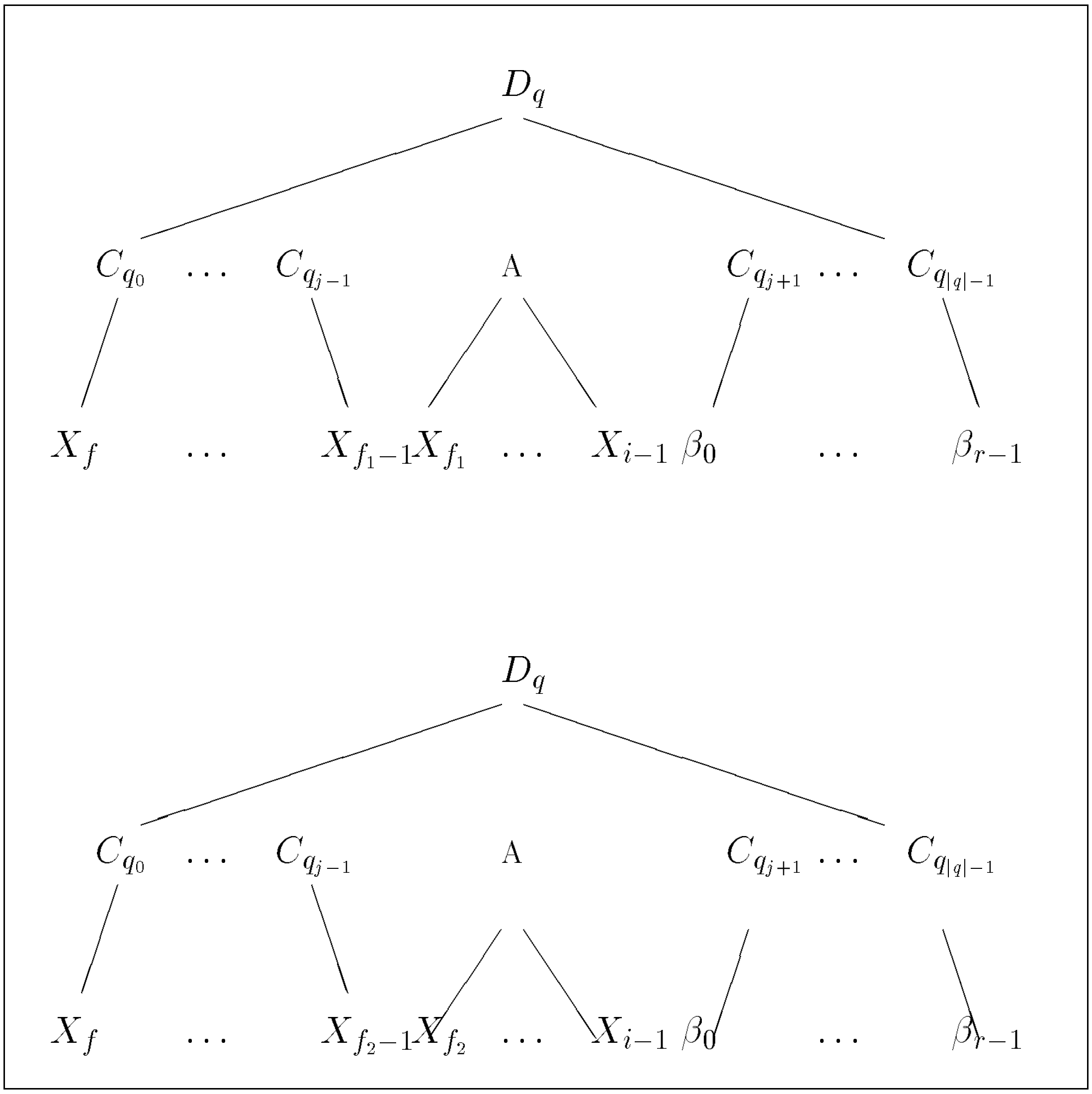


Figure 6: Theorem 1, Case 2: ($f_1 \neq f_2$) The two parse trees.



Adding Lookahead to the Algorithm

Earley's algorithm cannot achieve the linear time bound without utilizing its lookahead feature. Note that the lookahead must be at least as large as k, when recognizing an $LR(k)$ grammar, to achieve the linear time bound. If the lookahead is less than k, it might recognize some $LR(k)$ grammar in time $\Omega(n^2)$ (for more details, see Earley's dissertation).

To use lookahead, out simplified algorithm is modified as follows: A state is now represented as (q, j, f, α), where α is a string of length k representing a possible lookahead in order for the production q to be viable in the parse tree.

We append k end-of-file markers to the input string. The first state to be added to S_0 is $(\phi \rightarrow R, 0, f, \dashv^k)$, instead of $(\phi \rightarrow R, 0, f)$. The scanner operation remains the same. The predictor will now add a lookahead value to each state that it adds, and the completer will now check the lookahead and make sure it matches the next k input characters before it completes a state. The new predictor and completer operations are:

Completer: For each state (q, j, f, α) in S_i, if $j = |q|$ and $X_i \ldots X_{i+k-1} = \alpha$, then for each state $(p, y, f', \alpha') \in S_{f_D}$ where $D = D_q$, add the state $(p, y+1, f', \alpha')$ to S_i.

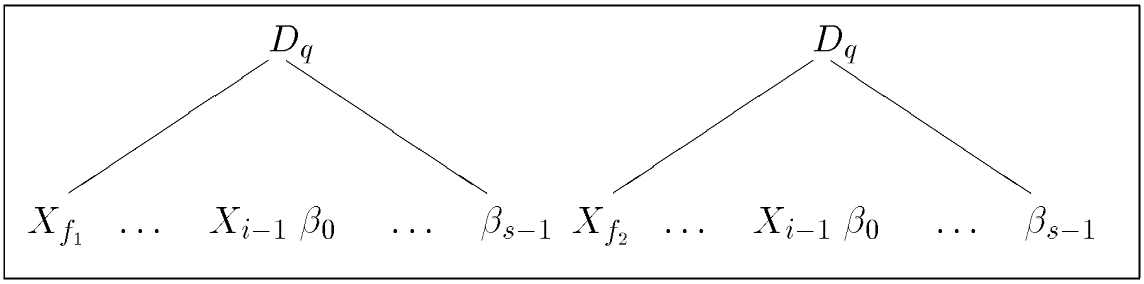
Predictor: If S_i is empty, then reject. If $i = n+1$ (i.e. the entire input has already been scanned), and

$$S_i = \{ \quad (\phi \rightarrow R, 1+k, 0, \dashv^k) \quad \}$$

then accept.

For each state (q, j, f, α) in S_i, if C_{q_j} is nonterminal, do the following two operations: (1) for each alternative q' of C_{q_j}, if C_{q_j} is the last symbol in the production q, add the state $(q', 0, i, \alpha)$ to S_i; (2) if C_{q_j} is not the last symbol in the production q, then for each string α' of length k such that $C_{q_{j+1}} \ldots C_{q_{|q|}} \stackrel{*}{\Rightarrow} \alpha'\beta$ (for any string β), and for each alternative q' of C_{q_j}, add the state $(q', 0, i, \alpha')$ to S_i.

Figure 7: The two parse subtrees if the completer adds the two states (q, j, f_1, α) and (q, j, f_2, α) to S_i



Proof of $O(n)$ Time Bound on $LR(k)$ Grammars

Given an $LR(k)$ grammar, and using a lookahead of length k, we will assume that the recognizer always makes a correct decision about what productions to complete. Thus every state that is completed by the completer will appear in the parse tree for X; i.e. the completer will complete as many states as there are nodes in the parse tree for X.

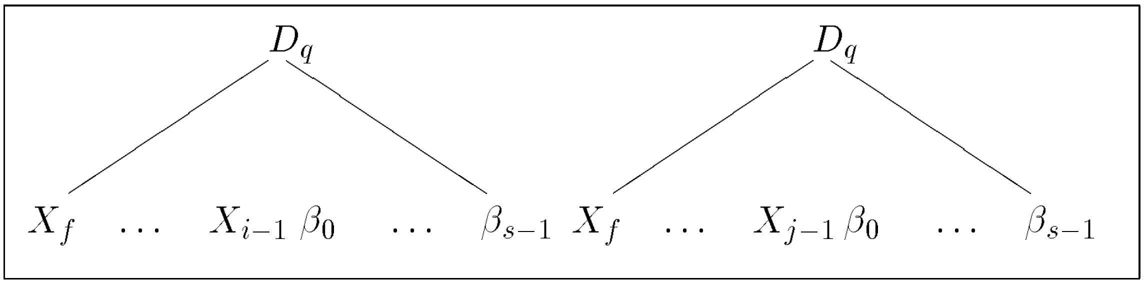
Lemma 3 *Given an unambiguous grammar, the completer adds $O(k)$ states for each state it completes.*

Proof: Suppose this were not true. Let's first assume that the completer never adds a state (q, j, f, α) to two different state sets. Since the number of values for q, j and α is bounded, then there must exist a state set in which the completer added the two states (q, j, f_1, α) and (q, j, f_2, α) where $f_1 \neq f_2$. Then, given β by String Completion, we have the two possible subparsings for $X_f \ldots X_{i-1}\beta_0 \ldots \beta_{s-1}$ as shown in Figure 7.

This, of course, is a contradiction, since the grammar is unambiguous. Now, consider the possibility that the completer adds the same state (q, j, f, α) to two different state sets, S_i and S_j, where $i \neq j$. Then we have the two possible subparsings for $X_f \ldots X_{i-1}\beta_0 \ldots \beta_{s-1}$ as shown in Figure 8, which is also a contradiction.

Lemma 4 *Given an $LR(k)$ grammar, the number of states the completer adds during the entire parsing for some input X is $O(n)$.*

Figure 8: The two parse subtrees if the completer adds the state (q, j, f, α) to two state sets S_i and S_j



Proof: Since the grammar is $LR(k)$, the completer only completed those states that are nodes in the parse tree for X. For each of these states, it runs in $O(k)$ time. The number of nodes in the parse tree for X is at most $(2n-1)$, the size of a binary tree with n leaves. Thus the completer runs in time $O(n)$ overall.

Lemma 5 *Given an $LR(k)$ grammar, the scanner operates overall in time $O(n)$.*

Proof: We already know that the scanner operates in time $O(k)$ for each state that it scans. The only remaining uncertainty is in the number of states that it scans, which is the total number of states kept in the array. The number of states that the completer adds is bounded by the number of nodes in the parse tree for X. The number of states that the predictor adds per state set is bounded by the number of productions in the grammar. The number of states that the scanner adds per state set is $O(k)$, because, as proved above, there cannot exist two states (q, j, f_1, α) and (q, j, f_2, α) in the same state set, and there cannot exist the same state in different state sets. Thus the scanner itself adds $O(k)$ states per state set. Summing all of these states together, the total number of states kept in the array is $O(n)$. Thus the scanner runs in $O(n)$ time overall.

Lemma 6 *The predictor operates, overall, in time $O(n)$.*

Proof: Again, since the predictor operates in $O(k)$ time for each state that it scans, we need only prove that the number of states it scans is $O(n)$, which we just did.

Theorem 3 *Given $LR(k)$ grammar, the algorithm operates in time $O(n)$.*

Proof: The completer, predictor, and scanner all operate in time $O(n)$ overall; thus the algorithm operates in time $O(n)$ as well.

Acknowledgements

I am indebted to Richard Beigel of Yale University for his guidance and counsel in writing this paper. I would also like to thank Professor Gasarch of the University of Maryland for his reviews and helpful advise.

References

[1] Aho, Alfred; Sethi, Ravi; and Ullman, Jeffrey. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Publishing Co.; Reading, MA. 1986

[2] Aho, Alfred; and Ullman, Jeffrey. *The Theory of Parsing, Translation, and Compiling, Vol. I: Parsing*. Prentice Hall, Inc.; Englewood Cliffs, N.J. 1972

[3] Earley, J. *An Efficient Context-Free Parsing Algorithm.* Ph.D. Thesis, Computer Science Dept., Carnegie-Mellon University, Pittsburg, PA. 1968

[4] Earley, J. *An Efficient Context-Free Parsing Algorithm.* In *Communications of the ACM 13, 2* (Feb. 1970), 94-102.

[5] Floyd, Robert; and Beigel, Richard. *The Language of Machines: An Introduction to Computability and Formal Languages*. Computer Science Press; New York. 1994