PARSING REGULAR GRAMMARS WITH FINITE LOOKAHEAD.

by: Thomas J. Ostrand,* Marvin C. Paull* and Elaine J. Weyuker**

DCS-TR-71
May 1978

* - from Dept. Computer Science
Rutgers University
New Brunswick, N.J.

** - from Dept. of Computer Science
Courant Inst. of Mathematical Sciences
New York University,
New York, N.Y. 10012

May 5, 1978
ABSTRACT

Not every unambiguous regular grammar can be parsed by a finite state machine, even if a lookahead facility is added to the machine's capabilities. Those which can be parsed with a fixed lookahead of \( k \) are said to be FL(\( k \)). If such a grammar has \( n \) non-terminals, it never needs more than \( (n(n-1)/2)+1 \) lookahead, and there exist grammars which do require this much. An algorithm is presented for determining whether a grammar is fixed lookahead parsable, and if so, for finding the minimum lookahead needed. The algorithm sets up and solves a set of \( O(n^2) \) equations using \( O(n^4) \) steps. Two parsing methods for FL(\( k \)) grammars are discussed. One uses a large precomputed parsing table, and operates in real time. The second parses an input string in time proportional to its length, while using approximately \( 3n \) storage locations.
PARSING REGULAR GRAMMARS WITH FINITE LOOKAHEAD.

by: Thomas J. Ostrand,* Marvin C. Paull* and Elaine J. Weyuker**

DCS-TR-71
May 1978

* - from Dept. Computer Science
   Rutgers University
   New Brunswick, N.J.

** - from Dept. of Computer Science
    Courant Inst. of Mathematical Sciences
    New York University,
    New York, N.Y. 10012

May 5, 1978
ABSTRACT

Not every unambiguous regular grammar can be parsed by a finite state machine, even if a lookahead facility is added to the machine's capabilities. Those which can be parsed with a fixed lookahead of \( k \) are said to be FL(k). If such a grammar has \( n \) non-terminals, it never needs more than \( (n(n-1)/2)+1 \) lookahead, and there exist grammars which do require this much. An algorithm is presented for determining whether a grammar is fixed lookahead parsable, and if so, for finding the minimum lookahead needed. The algorithm sets up and solves a set of \( O(n^2) \) equations using \( O(n^4) \) steps. Two parsing methods for FL(k) grammars are discussed. One uses a large precomputed parsing table, and operates in real time. The second parses an input string in time proportional to its length, while using approximately \( 3n \) storage locations.
1. Introduction

Grammars which allow an input string to be parsed with a fixed amount of scanning ahead in the string are particularly useful for programming language definition. In this paper, we define and study a class of regular grammars with this property. Regular grammars are generally used in the lexical analysis phase of compilation, as a means of defining the tokens of the programming language. While extensive study has been made of fixed lookahead parsing methods for context-free grammars (e.g., see Aho and Ullman [1]), similar methods for regular grammars have been neglected. This may be due to the fact that every regular grammar can be converted to an equivalent deterministic grammar which is parsable with lookahead 1. In general, however, the deterministic grammar will have many more nonterminals than its equivalent nondeterministic form, and the structure of the original grammar, which is essential for parsing, will be lost.

We use the following notation. $G = (N, \Sigma, P, S)$ is a grammar with nonterminals $N$, terminals $\Sigma$, productions $P$, and start symbol $S$. Strings of terminals are denoted by $w, x, y, z$; individual terminal symbols by $a, b, c$; and individual nonterminals by $A, B, C$. We also use lower case Greek letters $\alpha, \beta, \gamma, \delta$ as variables ranging over $N$. For
x \in \Sigma^+ and 1 \leq i \leq j \leq |x|, x[i:j] is the substring of x consisting of the symbols in positions i through j of x, inclusive. We use \( x[i] \) for \( x[i:i] \). FIRST\(_k\)(x) is \( x[1:k] \) if \( |x| \geq k \), and \( x \) if \( |x| < k \).

A suitable type of lookahead parser for regular grammars is a finite state machine augmented with a lookahead register which contains the \( k \) most recent input symbols. After reading the first \( t+1 \) symbols of input \( w \), the machine will be in state \( q(t) \) and the lookahead register will contain \( w[t:t+k-1] \). Reading the next input symbol \( w[t+k] \) enables the machine to determine its next state \( q(t+1) \). The states correspond to symbols from the nonterminal alphabet of the regular grammar being parsed. Informally, we say that the grammar is parsable by finite state machine with fixed lookahead \( k \).

Our definition of fixed lookahead parsing is given in terms of grammars rather than machines, since parsing is an activity normally associated with grammars and to emphasize the analogy to the fixed lookahead parsable context-free LL(k) grammars [2].

A regular grammar is parsable with fixed lookahead \( k \) if it is possible to determine uniquely the next production in a derivation by examining the nonterminal in the current sentential form and the next \( k \) symbols in the input string. Let \( G = (N, \Sigma, P, S) \) be a right-linear grammar. We say that \( G \) is parsable by finite state machine with fixed lookahead \( k > 0 \), or that \( G \) is FL(k) if whenever there are two derivations
\[ S \Rightarrow wT \Rightarrow waA \Rightarrow wx \]

and
\[ S \Rightarrow wT \Rightarrow waB \Rightarrow wy \]

such that \( \text{FIRST}_k(x) = \text{FIRST}_k(y) \), then \( A = B \). \( G \) is an FL grammar if it is FL(k) for some \( k \).

If \( G \) is an FL grammar, then the required lookahead for \( G \) is the minimum \( k \) such that \( G \) is FL(k).

We frequently present a grammar \( G \) by giving a nondeterministic finite automaton equivalent to \( G \). Figure 1 shows a regular grammar \( C_1 \) and equivalent automaton \( A_1 \). A lookahead of four is both necessary for \( C_1 \) (as shown by the input abba) and sufficient to parse any input.

A grammar can fail to be fixed-lookahead parsable either because it is ambiguous, or because its lookahead requirement for parsing some set of accepted inputs is unbounded. The automaton \( A_2 \) in Figure 2 is a simple modification of \( A_1 \) which is ambiguous. The strings ab and abb are each accepted on more than one path in \( A_2 \). A different modification of \( A_1 \) produces \( A_3 \) in Figure 3, which is unambiguous, but requires unbounded lookahead. \( A_3 \) accepts the union of the sets \( a(ba)^*b \) and \( a(ba)^*bb \). Since a proper prefix of any accepted input could come from a word in either set, and the sets are accepted on disjoint paths, an unbounded lookahead is required to parse inputs.

In Section 2 we give an alternative characterization of FL(k) grammars and show that if a grammar is FL, there is an upper bound on the lookahead necessary to parse it.
\[ S \rightarrow aA_1 \mid aA_2 \]
\[ A_1 \rightarrow bA_2 \mid bA_3 \mid b \]
\[ A_2 \rightarrow AS \mid bA_4 \]
\[ A_3 \rightarrow aA_1 \]
\[ A_4 \rightarrow bA_3 \mid b \]

(a) FL(4) Grammar \( G_1 \)  
(b) Equivalent Finite Automaton \( A_1 \)

Figure 1
Figure 2  Ambiguous Automaton $A_2$

Figure 3  Unbounded Lookahead Automaton $A_3$
We also show that there are grammars whose lookahead requirements meet this bound. Because of the bound, it is decidable whether an arbitrary regular grammar $G$ is $FL(k)$ for some $k$. In Section 3, we present an efficient algorithm for determining whether $G$ is $FL$ and if so, for determining the minimum $k$.

Section 4 discusses several ways of designing a parser for an $FL(k)$ grammar $G$. One is the finite state machine with lookahead register described above. Its moves are based on a parsing table which contains an entry for each pair consisting of a nonterminal $A$ of $G$ and a length-$k$ sequence $z$ of inputs. The entry gives the correct production to be used to generate the first symbol in $z$. The machine's state set is the set of nonterminal symbols of $G$, with $S$ the initial state. A move consists of consulting the parsing table to determine the next state and production, and shifting the lookahead register one position left to read the next input into its rightmost position. This machine can recognize and parse $L(G)$ in real-time, but the size of the parsing table may be exponential in the amount of lookahead.

We also present an alternative parsing algorithm which operates in linear time, and which has a much more modest storage requirement. In addition this algorithm can be used to parse grammars with unbounded lookahead requirements, although its efficiency may drop in such cases.
2. Bounds for Finite Lookahead

It is undecidable whether an arbitrary context-free grammar $G$ is parsable with fixed lookahead, that is, whether there is an integer $k$ such that $G$ is LL($k$). However, the analogous question is decidable for arbitrary regular grammars; the following theorem provides a decidable characterization of PL grammars and also gives an upper bound on the required lookahead.

The theorem essentially says that the only way an unambiguous grammar can require arbitrary lookahead is by having two identical derivation loops starting at distinct nonterminals.

**Theorem 1.** An unambiguous regular grammar $G$ with $n$ nonterminals is not parsable with fixed lookahead if and only if there are nonterminals $A$ and $B$, and words $w_1 \in \Sigma^*$, $w_2 \in \Sigma^+$, with $|w_2| \leq n(n-1)/2$, such that

$$S \xrightarrow{*} w_1A \xrightarrow{+} w_1w_2A'$$

and

$$S \xrightarrow{*} w_1B \xrightarrow{+} w_1w_2B',$$

where either $A' = A$ and $B' = B$, or $A' = B$ and $B' = A$.

**Proof:** The condition is clearly sufficient for nonparsability with fixed lookahead, since the loops can be repeated arbitrarily often.
For necessity, let G be unambiguous, not PL, and have n nonterminals. In particular, G is not parsable with lookahead \( r = \frac{n(n-1)}{2} + 1 \). Thus, there exist derivations

\[ S \Rightarrow^* wT \Rightarrow^* w\alpha \Rightarrow^* wx \quad (1) \]

\[ S \Rightarrow^* wT \Rightarrow^* w\beta \Rightarrow^* wy \quad , \quad (2) \]

such that \( \text{FIRST}_r(x) = \text{FIRST}_r(y) \) and \( \alpha \neq \beta \). Since G is unambiguous, \( |x| > r \) and \( |y| > r \). Consider the first \( r \) steps past \( wT \) of the two derivations as proceeding in parallel from the points \( w\alpha \) and \( w\beta \). At each step there appears a pair of nonterminals, one in each derivation. Since there are \( \frac{n(n-1)}{2} \) distinct pairs of nonterminals in G, at some step a previously occurring pair, say \( A \) and \( B \), must appear.

The forms of the two derivations are therefore

\[ S \Rightarrow^* wT \Rightarrow^* w_{1}A \Rightarrow^* w_{1}w_{2}A' \Rightarrow^* wx \quad (1') \]

\[ S \Rightarrow^* wT \Rightarrow^* w_{1}B \Rightarrow^* w_{1}w_{2}B' \Rightarrow^* wy \quad (2') \]

Since the repetition occurs within \( r-1 \) steps, and \( \text{FIRST}_r(x) = \text{FIRST}_r(y) \), it follows that \( w_1 = u_1 \), \( w_2 = u_2 \), and \( |w_2| \leq r-1 \). \( \Box \)

**Corollary.** If an unambiguous regular grammar G with n nonterminals is parsable with required lookahead \( k \), then \( k \leq \left( \frac{n(n-1)}{2} \right) + 1 \).
Proof: If the required lookahead were greater than 
\((n(n-1)/2) + 1\), then the pair of nonterminals of the 
theorem would exist, and G would not be parsable. □

The next result shows that the lookahead bound is as 
tight as possible, by exhibiting for every \(n > 2\) a grammar 
with \(n\) nonterminals and required lookahead \((n(n-1)/2)+1\).

Theorem 2. For every \(n > 2\), there exists a regular grammar 
\(G = (N, \Sigma, P, A_1)\) with \(|N| = n\), such that \(G\) is parsable with 
required lookahead \((n(n-1)/2)+1\).

Proof: Let \(N = \{A_1, \ldots, A_n\}\), \(\Sigma = \{a_0, \ldots, a_k\}\), where 
k = \(\left\lceil \frac{n}{2} \right\rceil\). \(P\) consists of the following productions:

\[A_1 \rightarrow a_0 A_1 | a_0 A_2\]
\[A_{n-1} \rightarrow a_1\]
\[A_2 \rightarrow a_2\]

for each \(i = 1, 2, \ldots, n-1\) , \(A_i \rightarrow a_1 A_{i+1}\)

for each \(i = 2, 3, \ldots, k\) , \(A_n \rightarrow a_1 A_1\)

for each \(i = 1, 2, \ldots, k-1\) , \(A_{n-i} \rightarrow a_{1+i} A_{2+i}\)

for each \(i = 1, 2, \ldots, k-1\) , \(A_{1+i} \rightarrow a_{1+i} A_{n+1-i}\)

Note that the only nondeterminism in the grammar is from 
the start symbol \(A_1\). \(G\) is unambiguous, since no two of 
its productions have identical right-hand sides. (Note that 
the existence of identical right-hand sides in different 
productions is necessary, but not sufficient for ambiguity.)
We first show that \( G \) is fixed lookahead parsable. Suppose not, and let
\[
A_1 \rightarrow^{*} wA \rightarrow^{*} w w_1 A' \\
A_1 \rightarrow^{*} wB \rightarrow^{*} w w_1 B'
\]
be two derivations as in Theorem 1, where \( \{A',B'\} = \{A,B\} \) is the first pair of repeated nonterminals in the derivations. From the productions of \( G \), note that
(a) the nonterminals \( A_1, A_2 \) can appear in parallel only at the first step of the derivations.
(b) for every other pair \( \{\alpha, \beta\} \) there is a unique pair \( \{\gamma, \eta\} \) which can precede \( \{\alpha, \beta\} \) in parallel derivations.
For the pair \( \{A_i, A_j\} \), the predecessor pair is \( \{A_{i-1}, A_{j-1}\} \) if \( i, j \neq 1 \) and \( \{A_n, A_{n-i+2}\} \) if \( j = 1 \).

Thus, whatever pair precedes \( \{A', B'\} \) must also precede \( \{A, B\} \), contradicting the assumption that \( \{A', B'\} \) is the first repetition, and that the looping derivations exist.

To see that a lookahead of \( (n(n-1)/2) + 1 \) is required for \( G \), let
\[
y = a_0 a_1^{n-2} a_2 a_1^{n-3} a_3 \ldots a_{n-k} a_k
\]
and
\[
z = a_1^{k-2} a_k^{-1} a_{k-1} a_1^{k-3} \ldots a_{k-2} a_k
\]
Using \( w_1 = y a_1^{k-1} a_k z \) if \( n \) is even and \( w_2 = y z \) if \( n \) is odd, note that any word in \( \Sigma^* \) beginning with \( w_1 \) or \( w_2 \) requires lookahead at least \( (n(n-1)/2) + 1 \), since the last \( a_2 \) could
be produced by either $A_n \rightarrow aA_1$ or $A_2 \rightarrow a_2A_n$, and the first $a_0$ could be produced by either $A_1 \rightarrow a_1A_1$ or $A_1 \rightarrow a_0A_2$. □

Figure 4 shows the automaton corresponding to $G$ for the case $n = 8$, and the input prefix which requires a lookahead of 29.

It might seem as though the high lookahead requirement for the grammar of Theorem 2 is due to the relatively large size of the terminal alphabet, and that a grammar with a low ratio of terminal to nonterminal symbols could not require such a large lookahead. However, the next result shows that an unambiguous grammar with $n+1$ nonterminals and as few as 2 terminals can require lookahead proportional to $n^2$.

**Theorem 3:** For each $n > 2$, there is a regular grammar $G = (N, \Sigma, \mathcal{F}, S)$ with $|N| = n+1$ and $\Sigma = \{a, b\}$ such that $G$ is fixed-lookahead parsable, with required lookahead at least $(n^2+3)/4$.

**Proof:** Let $N = \{S, A_1, \ldots, A_n\}$, $\Sigma = \{a, b\}$. $\mathcal{F}$ consists of the following productions:

$S \rightarrow aA_1|aA_2$

$A_{n-1} \rightarrow a$

$A_3 \rightarrow b$ \hspace{1cm} if $n$ is odd

$A_i \rightarrow aA_{i+1}$, \hspace{.5cm} $i = 1, \ldots, n-1$

$A_n \rightarrow bA_1$

$A_{n-k} \rightarrow bA_{k+3}$, \hspace{.5cm} $k = 1, 3, 5, \ldots, 2\lfloor n/2 \rfloor - 3$

The grammar is unambiguous despite the rules $S \rightarrow aA_2$ and $A_1 \rightarrow aA_2$, since the start symbol can occur only at the beginning of a derivation. An argument similar to the one
in the previous proof shows that \( \mathcal{G} \) is fixed-lookahead parsesable.

The maximum lookahead requirement for \( \mathcal{G} \) is for one of the following strings, depending on whether \( n \) is odd or even.

\[
\begin{align*}
  w_1 &= a^{n-1}ba^{n-4}ba^{n-6}b\ldots a^4ba^2b \\
  w_2 &= a^{n-1}ba^{n-4}ba^{n-6}b\ldots a^3b
\end{align*}
\]

For \( n \) even, the sentential forms \( w_1A_1 \) and \( w_1A_n \) are derivable, and one additional terminal symbol resolves the parse uncertainty. For \( n \) odd, the derivable forms are \( w_2A_2 \) and \( w_2A_n \).

In either case, one additional terminal symbol resolves the parse ambiguity. Since \(|w_1| = n^2/4\) and \(|w_2| = (n^2-1)/4\), the required lookaheads are, respectively, \((n^2+4)/4\) and \((n^2+3)/4\).

Figure 5 shows the 11 state automaton on \( \{a,b\} \) which has a lookahead requirement of 26, and the appropriate input prefix.
3. Determining the Lookahead Required by a Regular Grammar

Throughout this section we use the equivalent nondeterministic finite automaton $M$ for a given regular grammar $G$. Each nonterminal of $G$ becomes a state of $M$, with the start symbol $S$ being the initial state of $M$. A special final state $F$ is also introduced. The inputs for $M$ are the terminal symbols of $G$. For each production $A \rightarrow aB$ of $G$, there is a transition in $M$ from state $A$ to state $B$ labeled with $a$. For each production $A \rightarrow a$, there is a transition from state $A$ to the introduced final state $F$, labeled with $a$. (The automaton $A_1$ of Figure 1 has one state less than the automaton produced by the above standard construction from grammar $G_1$.) A derivation in the grammar $S \xrightarrow{*} w \in \Sigma^*$ corresponds to a path through $M$, starting at state $S$, passing through each state corresponding to the nonterminals appearing in the derivation, and ending at state $F$. The transitions of the path are labeled with the individual symbols of $w$.

The problem of parsing a string $w$ in language $L(G)$ is now expressed as the problem of finding the path through $M$ produced by $w$. If $G$ is parsable with lookahead $k$, this is equivalent to being able to decide, for a given state of $M$ and sequence of $k$ input symbols, what the next state of $M$ is. We use the same terminology for automata that was introduced previously for grammars, and say that $M$ is FL($k$)
Figure 4  FL(29) Automaton.
Prefix \( a_0 a_1^5 a_2 a_1^5 a_3 a_1^4 a_4 a_1^3 a_4 a_1^2 a_3 a_1 a_2 \) leaves uncertainty.

Figure 5  FL(26) Automaton.
Prefix \( a^9 b a^6 b a^4 b a^2 b \) leaves uncertainty
or parsable with fixed lookahead \( k \). The required lookahead for \( M \) is the minimum \( k \) such that \( M \) is \( PL(k) \). We present in this section an algorithm for deciding whether a nondeterministic finite automaton is parsable with fixed lookahead, and if so, for determining the required amount of lookahead.

The algorithm must be able to provide answers for the following two questions. First, during the process of accepting an input string, at what points does \( M \) have a choice of possible next states? Second, whenever \( M \) has such a choice, how many additional input symbols must be examined to reduce the choice to a single state?

The concept of resolvability of a set of states is the means by which the second question is answered, and the key to the algorithm.

**Definition.** The set of states \( P = \{ p_1, \ldots, p_n \} \) is implied by the set \( Q = \{ q_1, \ldots, q_m \} \) under input \( a \) iff for every \( 1 \leq i \leq m \) there is a \( j, 1 \leq j \leq n \) such that \( p_j \in \delta(q_i, a) \) and for every \( 1 \leq j \leq n \) there is an \( i, 1 \leq i \leq m \) such that \( p_j \in \delta(q_i, a) \).

\( Q \) implies \( P \) iff \( P \) is implied by \( Q \) under some input.

**Definition.** Input sequence \( w \) is applicable to state \( q \) iff \( \delta(q, w) \neq \emptyset \).

**Definition.** A set \( Q \) of states is ambiguous if there are states \( q_i, q_j \in Q \) and input \( a \) such that \( \delta(q_i, a) \cap \delta(q_j, a) \neq \emptyset \), i.e., under some input \( a \) two different states can lead to the same next state.
Definition. A set \( Q \) of states is \( 0 \)-resolvable iff \( Q \) contains exactly one state. For \( k > 0 \), \( Q \) is \( k \)-resolvable iff \( Q \) is not ambiguous and every input sequence of length \( k \) is applicable to at most one state of \( Q \). The \textit{resolvability} of set \( Q \) is the minimum \( k \) such that \( Q \) is \( k \)-resolvable. If there is no such \( k \), then \( Q \) is not resolvable. Lemma 2 below characterizes the types of state sets which are not resolvable. Note that if \( Q \) has resolvability \( k \), then there is some sequence of length \( (k-1) \) which is applicable to two or more states of \( Q \).

The resolvability of a set \( Q \) of states can obviously be determined by pairwise examination of members of \( S \).

Lemma 1. Set \( Q \) of states has resolvability \( k \) if and only if the resolvability of every pair of states of \( Q \) is less than or equal to \( k \), and there is some pair whose resolvability is \( k \). □

Resolvability of a set of states is dependent on the resolvability of the set's successor states, and the nonambiguity of the set itself.

Lemma 2. The resolvability of set \( Q \) is \( k > 0 \) if and only if the resolvability of every set implied by \( Q \) is less than or equal to \( (k-1) \); the resolvability of at least one set implied by \( Q \) is \( (k-1) \), and \( Q \) is not ambiguous.

Proof: If \( Q \) has resolvability \( k \), then the resolvability of any set implied by \( Q \) cannot be greater than \( (k-1) \). Let \( w = au \)
have length (k-1) and be applicable to states q₁ and q₂ of Q. Let R be the set implied by Q under a, p₁ = δ(q₁, a) and p₂ = δ(q₂, a). Since Q is not ambiguous, p₁ ≠ p₂. Since u is applicable to p₁ and p₂ and |u| = k-2, the resolvability of R is at least (k-2), and hence is (k-1).

Now suppose the three conditions on the right of the lemma hold. Let w = au be an arbitrary sequence of length k, and assume it is applicable to states q₁ and q₂ of Q. Then u is applicable to p₁ = δ(q₁, a) and to p₂ = δ(q₂, a). Since Q is not ambiguous, p₁ ≠ p₂ and both are in the set implied by Q under a. But this set has resolvability less than or equal to (k-1), so u could not be applicable to p₁ and p₂; therefore w could not be applicable to q₁ and q₂, and the resolvability of Q is less than or equal to k.

Since Q implies some set with resolvability (k-1), it is clear that there is a sequence of length (k-1) applicable to two states of Q. Thus the resolvability of Q is k. □

Now consider the problem of parsing an input string w, i.e., determining the sequence of states followed by the automaton under w. The initial state is unique and therefore known. Suppose the automaton is in state q after some initial portion of w, and the next input symbol is a. The possible next states are δ(q, a). If the resolvability of δ(q, a) is (k-1), then the (k-1) symbols following a in w will be applicable to only one state in δ(q, a), and the next state is determined by a total lookahead of k.
Conversely, if a total lookahead of \( k \) suffices to
determine the next state, then the sequence of length \((k-1)\)
which follows a is applicable only to that one state.
If lookahead \( k \) is necessary to determine the next state,
then the \((k-2)\) length following sequence must be applicable
to at least two states. Thus the resolvability of all sets
\( \delta(q,a) \) is at most \((k-1)\), and there is some \( \delta(q,a) \) with
resolvability \((k-1)\).

Thus we have the following.

**Theorem 4.** The automaton \( M \) is parsable with required
lookahead \( k \) if and only if the maximum resolvability of
\( \delta(q,a) \), for \( q \in Q, a \in \Sigma \), is \( k-1 \). \( \square \)

Note the special case of Theorem 4 which arises when
\( M \) is deterministic. All sets \( \delta(q,a) \) are singletons, and
their resolvability is therefore 0. Hence a deterministic
automaton is parsable with lookahead 1.

Lemmas 1 and 2 and Theorem 4 are the basis of the
following algorithm for determining the lookahead required
for parsing \( M \).

**Algorithm 1 (Determination of lookahead requirement).**

**Input:** The transition table of an \( n \)-state nondeterministic
finite automaton \( M \).

**Output:** The lookahead required for parsing \( M \), or an indica-
tion that \( M \) is not FL.
(1) (a) Find all immediately implied pairs of M, that is, all pairs \((q_i, q_j)\) which are implied by a single state under any single input. Initialize a list \(L\) to these pairs.

(b) If there are no immediately implied pairs, then \(M\) is deterministic and \(FL(1)\); Exit Algorithm 1.

(2) For each pair \((q_i, q_j)\) in the list \(L\):

(a) Find all pairs of states which are implied by \((q_i, q_j)\) and add to \(L\) all such pairs not already on \(L\). Continue finding implied pairs for all new pairs added to \(L\).

(b) If \((q_i, q_j)\) implies a single state, that is, if for some input \(a\), \(\delta(q_i, a)\) and \(\delta(q_j, a)\) contain a common state, then Exit Algorithm 1.

(3) If \(M\) was not found to be ambiguous, then set up equations for the resolvability \(R_{ij}\) of each pair \((q_i, q_j)\) in \(L\), as follows:

For a pair \((q_i, q_j)\) which does not imply any pair,

\[ R_{ij} = 1. \]

For a pair \((q_i, q_j)\) which implies the pairs \((q_i, q_j)^1, \ldots, (q_i, q_j)^t\),

\[ R_{ij} = 1 + \max(R_{ij}^1, \ldots, R_{ij}^t). \]
(4) Solve for all $R_{ij}$, by back substitution. If a substitution ever results in an equation with the same $R_{ij}$ on each side, the presence of two parallel loops in $M$ is indicated, and by Theorem 1 $M$ is not parsable with fixed lookahead. (Note that such an occurrence would lead to an equation of the form $R = 1 + \max(R, \ldots)$, implying $R > R$. No finite value could satisfy this condition.)

(5) If there is a finite solution for each $R_{ij}$, then $M$ is parsable with fixed lookahead $k = \max(R_{ij}) + 1$. □

Algorithm 1 must only be applied to finite automata with a single final state since it will not detect ambiguous paths which terminate at different final states. The standard automaton always has a single final state, and it is easy to convert any arbitrary f.a. into another with one final state.

Appendix 1 contains an example of the behavior of the lookahead determination algorithm, applied to the grammar of Figure 1.

The algorithm requires time at most proportional to $n^4$ for a f.a. with $n$ states. Step (4), solving the equations, is the dominant part of the algorithm. Since there are $n(n-1)/2$ pairs of states, Step (3) will produce at most this many equations. If the f.a. is F.L, then either there are no immediately implied pairs because the automaton is deterministic, or else at least one implied pair of states has resolvability 1. In the latter case, the set of equations...
is at most upper triangular, since no pair of states can
be its own successor. Back substitution is therefore
adequate for solving the set, and the number of substitu-
tions and comparisons is at most proportional to \( n^4 \); this
is in contrast to Strassen's algorithm [3] which requires
\( O(n^2 \log 7) \) time for solving a general set of \( n^2 \) linear
equations.
4. Constructing a Fixed-Lookahead Parser

There are many ways to take advantage of a grammar's FL(k) property in constructing a parser. The fastest possible parser is based on a parsing table, which contains an entry for each combination of nonterminal symbol and length-k input sequence. Such a table is essentially the next-state table for the finite-state parser. The parsing table's entry for nonterminal A and input sequence x is the correct next nonterminal and number of the next production used in the input string's derivation. The main drawback of a parsing table is its potential size. For an FL(k) grammar with n nonterminals, over an r-symbol terminal alphabet, there can be as many as nr^k entries. If, as is frequently the case with grammars encountered in practice, only a small portion of the grammar actually requires the full lookahead, the parsing table can be considerably smaller than this. Since construction of the parsing table is straightforward once the required lookahead is known, we do not discuss this method further.

A second parsing algorithm requires more time than the parsing table method, but uses considerably less storage. To describe this algorithm we again use the finite automaton corresponding to the grammar G; the parser produces the state sequence which the automaton passes through under the input string. From the state at time t and the (t+1)-st input symbol, the algorithm determines the set
\( N = \delta(q(t), w[t+1]) \) of possible next states. The parser identifies the actual next state as the unique element of \( N \) to which the next \((k-1)\) input symbols are applicable. The parser actually looks ahead only as far as necessary to eliminate all but one state from \( N \); in many cases this will be less than the automaton's maximum of \( k-1 \).

Algorithm 2 keeps track of the states reachable from each element of \( N \) by the inputs \( \{w[t+2:t+j]\}, j = 2, \ldots, k \). That is, the sets \( \delta(q, w[t+2:t+j]), j = 2, \ldots, k \) are computed for each \( q \in N \). Because of the PL(k) property, for some \( j \leq k \), all these sets except one must be empty. The \( q \) of the remaining nonempty set is the desired next state.

Because the f.a. being parsed is assumed to be unambiguous, an efficient method of storing the sets of reachable states can be used. For \( q \) and \( q' \in N \), \( \delta(q, w[t+2:t+j]) \) and \( \delta(q', w[t+2:t+j]) \) must be disjoint, for otherwise the input \( w[t+1:t+j] \) could take the f.a. through two distinct state sequences, starting with \( q(t) \) and ending with the common state in these two sets.

To record the reachable states at time \( (t+j) \), Algorithm 2 uses a vector \( V[1:n] \), where \( V(i) = 1 \) iff state \( q_i \) is in \( \delta(q, w[t+2:t+j]) \).

Algorithm 2. Fixed lookahead parsing for an PL(k) finite automaton \( M \) with \( n \) states.

Input: String \( w[1:k] \)
Output: Sequence of states \( (q(0), q(1), \ldots, q(\ell)) \) M passes through as it accepts \( w \), or an indication that \( w \) is not accepted.

(1) /* Initialize */

\[
\begin{align*}
S & \rightarrow q_0 \\
& /* q_0 is start state q(0) */
\end{align*}
\]

\[
\begin{align*}
t & \rightarrow 0 \\
& /* t points to current input */
\end{align*}
\]

(2) Parseloop

\[
\text{while (} t < \ell \text{)}
\]

\[
\text{do:}
\]

(2.1) for \( i = 1 \) to \( n \) \( V(i) \rightarrow 0 \)

\[
N \rightarrow \delta(S, w[t+1]) /* N is the set of states accessible from present state by next input */
\]

for each \( q_i \in N \) \( V(i) \rightarrow i \)

(2.2) /* Lookahead to resolve \( V \) to one state */

\[
j \rightarrow 2
\]

\[
\text{while (} V \text{ contains at least 2 unequal nonzero entries)}
\]

\[
\text{do: for } i = 1 \text{ to } n \quad Y(i) \rightarrow 0
\]

\[
\text{for } i = 1 \text{ to } n
\]

\[
\text{if } V(i) \neq 0 \text{ then } N \rightarrow \delta(q_i, w[t+j])
\]

\[
\text{for each } q_r \in N
\]

\[
Y(r) \rightarrow V(i)
\]

fi

\[
V \rightarrow Y
\]

\[
j \rightarrow j+1
\]

\[
\text{od}
\]

21
(2.3) if $V(i) = 0$ for every $i = 1, \ldots, n$
    then print 'w not accepted by M'
    exit from Algorithm 2
fi

(2.4) /* Next state $q(t+1)$ is identified as the
    only nonzero number in $V$ */
S = $q_r$, where $r$ is nonzero number in $V$
t = t+1
print $S$, $w[t]$

(2.5) od /* End of Parseloop */

(3) /* End of input; check whether w is accepted */
if $S$ is not accepting state
    then print 'w not accepted by M' □

The algorithm uses storage vectors $V$ and $Y$ of length $n$, a list $N$ whose length could be as much as $n$, plus a few additional locations. Most of the time is spent in the Parseloop of Section 2. The body of this loop is performed $k$ times for an input of length $k$. The critical section within the body is (2.2). For each input character, this loop body is performed at most $k-1$ times. The assignment statement $Y(r) = V(i)$ inside the second for-loop is performed at most $n$ times for each value of $j$ (i.e., for each additional lookahead character), since the automaton could be in at most all $n$ states after reading each character. Algorithm 2 thus uses space proportional to $n$ and time proportional to $kn$ to parse an input string of length $k$.  

22
The computation of a nondeterministic f.a. on input \( w \) from state \( q_0 \) can be described by a tree whose nodes are states and whose branches are labeled with symbols from \( w \). The root is the initial state \( q_0 \), and all nodes a distance \( t \) from the root are the states accessible from \( q_0 \) by the first \( t \) symbols of \( w \). The successors of node \( q \) at level \( t \) are the states in \( \delta(q,w[t+1]) \); the branches from level \( t \) to \( t+1 \) are labeled with \( w[t+1] \).

For an unambiguous f.a., the entire tree produced by any accepted string \( w \) has exactly one path of length \( |w| \). For an \( FL(k) \) finite automaton, the tree of an accepted string has the additional property that every node has at most one descendant which is the root of a subtree containing a path of length \( k \) or greater.

Algorithm 2 keeps track of the root and frontier of each descendant subtree of the present state, and removes a root when its frontier becomes empty. The next state is uniquely identified when only one subtree root remains. During the lookahead process the interior of the selected subtree is not kept. An alternate approach would be to maintain the entire subtree structure during lookahead, thus obviating the need to compute the subtrees for each new input symbol. This can be done using either a \( k \times n \) array or a linked tree structure whose breadth at any level is never more than \( n \) and whose depth is at most \( k \). Although this might seem to be faster than Algorithm 2, a straightforward implementation still requires time proportional to \( kn \) for each input symbol.
Because Algorithm 2 uses a fixed amount of storage, it can be applied without modification to the parsing of an unambiguous grammar which requires unbounded lookahead. For such a grammar, processing symbol w[i] in an input string of length l may require a lookahead of as much as l-i+1. The maximum time required is therefore proportional to n.l^2.

Acknowledgement: We would like to thank Paul Abrahams for making several helpful suggestions.
Appendix 1  Use of Algorithm 1 (Determination of lookahead requirement)

The algorithm's behavior is illustrated on the automaton $A_1$ of Figure 1.

1. The immediately implied pairs are $(2, 3)$ from state 1 and $(3, 4)$ from state 2.
2. $(2, 3)$ implies $(3, 5)$ and $(4, 5)$, which are added to the list of pairs.
   - $(3, 4)$ implies $(1, 2)$
   - $(3, 5)$ implies $(4, 5)$
   - $(4, 5)$ implies nothing
   - $(1, 2)$ implies nothing
3. The equations are
   \[
   R_{23} = 1 + \max(R_{35}, R_{45})
   \]
   \[
   R_{34} = 1 + R_{12}
   \]
   \[
   R_{35} = 1 + R_{45}
   \]
   \[
   R_{45} = 1
   \]
   \[
   R_{12} = 1
   \]
4. The solution is $R_{34} = R_{35} = 2$, $R_{23} = 3$
5. The maximum required lookahead for parsing an input to $M$ is therefore 4.

The automata in Figures 2 and 3 show what occurs when Algorithm 1 is applied to a machine which is not FL. The ambiguity of $A_2$ is discovered in Step (2), when the implied pair $(2, 3)$ leads to the single state 4 under input a.

Notice that the original automaton $A_1$ contains state pairs such as $(1, 4)$ which lead to a single state, but $A_1$ is not
ambiguous because none of these pairs are themselves implied.

The unbounded lookahead required by \( A_3 \) is discovered in Step (4) of the algorithm. There are only two equations: \( R_{23} = 1 + R_{34} \) and \( R_{34} = 1 + R_{23} \). If a substitution is made into either one, the unbounded requirement is immediately apparent.
References

