AN N^2-RECOGNIZER FOR CONTEXT FREE GRAMMARS

by

Jay Earley

1967

Carnegie-Mellon University
Pittsburgh, Pennsylvania
September, 1967

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146) and is monitored by the Air Force Office of Scientific Research. Distribution of this document is unlimited.

ABSTRACT

An algorithm is described which is a recognizer for any context-free grammar. It is shown that the bound on the time the algorithm takes to recognize any string with respect to an unambiguous grammar is proportional to n^2, where n is the length of the string, that a bound on the time for recognizing any string is proportional to n^3, and that a bound on the space required is proportional to n^2.

INTUITIVE EXPLANATION

Let the grammar be in the form of a sequence of productions $D \rightarrow C_1 \ldots C_n$, where this means that D may be rewritten as the string $C_1 \ldots C_n$. The symbols to the left of '$\rightarrow$' are <u>non-terminals</u>, the symbols of the language itself are called <u>terminals</u>, and the non-terminal which stands for 'sentence' is called the <u>root</u> of the grammar. The set of all productions with D on the left side are called the <u>alternatives</u> of D. We will work with this example grammar of simple arithmetic expressions, grammar AE:

$$E \rightarrow T$$

$$E \rightarrow E + T$$

$$T \rightarrow P$$

$$T \rightarrow T * P$$

$$P \rightarrow a$$

The terminal symbols are {a, +, *}, the non-terminals are {E, T, P}, and the root is E.

The basic idea of the algorithm is as follows: It scans an input string from left to right. As each symbol is scanned, a set of states is constructed which represents the condition of the recognition process at that point in the scan. Each state in the set represents (1) a production such that we are currently scanning an instance of it, (2) a point in that production which shows how much of the production we have scanned, and (3) a pointer back to the position of the input string at which we began to look for that instance of the production. We will represent this triple as a production, with a dot in it, followed by an integer.

For example, if we are scanning a * a and we have scanned the first a, we would be in state set consisting of the following states:

$$P \rightarrow a. \quad 0$$

$$T \rightarrow P. \quad 0$$

$$T \rightarrow T.*P \quad 0$$

$$E \rightarrow T. \quad 0$$

$$E \rightarrow E.+T \quad 0$$

Each state represents a possible parse for the beginning of the string, given that we have seen only the a. All the states have 0 as a pointer, since all the productions represented must have begun at the beginning of the string. There will be one such state set for each position in the string. To aid in recognition, we place a right terminator '⊣' (a symbol which doesn't appear elsewhere in the grammar) at the right and of the input string.

To begin the algorithm we put the state

$$\phi \rightarrow .R⊣ \quad 0$$

into state set 0 (at position 0), where R is the root of the grammar and where ϕ is a new non-terminal. Then we compute the <u>closure</u> of the state set as follows: For each state in the set with a non-terminal N immediately to the right of the dot, add one state for each alternative of N. Put the dot in this state immediately to the left of C_1 and make the pointer point to the current position. This adds to the state set all productions which we might begin to look for at this point.

In grammar AE, state set 0 starts with

$$\phi \rightarrow .E⊣ \quad 0$$

Then we add to it

$$E \rightarrow .E+T \ 0$$

$$E \rightarrow .T \quad 0$$

The process is recursive, however, so we must add more productions for T:

$$T \rightarrow .T*P \ 0$$

$$T \rightarrow .P \quad 0$$

and for P:

$$P \rightarrow .a \ 0$$

This completes the closure for state set 0. The closure operation also involves another operation (which is not applicable to state set 0) which will be explained shortly.

After computing the closure of a state set, we scan the next character in the string (call it X) and construct a new state set for that new position as follows: For each state in the old state set which has X immediately to the right of the dot, put a state in the new state set which is the same as the old state except that the dot is moved 1 symbol to the right, so that X is now immediately to the left of the dot. If there are none with X to the right of the dot, an error has been detected since the input string doesn't match any of our current productions. Otherwise, we have gone into a new state set which contains all of the old states which still match the input string.

In grammar AE, if we scan an 'a' first, we construct state set 1 from state set 0 as follows:

0		1
$\phi \rightarrow .E\dashv$	0	
$E \rightarrow .E+T$	0	
$E \rightarrow .T$	0	
$T \rightarrow .T*P$	0	
$T \rightarrow .P$	0	
$P \rightarrow .a$	0	$P \rightarrow a.$ 0

Now we must compute the closure of state set 1. Since there are no states with non-terminals to the right of the dot, the previous closure operation is not applicable. However, another kind, which operates on states in which the dot is to the right of the production, is applicable.

For each such state

$$D \rightarrow C_1 \ldots C_n. \quad f$$

take each state in the closure of the state set to which f points of the form

$$E \rightarrow \alpha.D\beta \quad k \text{ (where } \alpha \text{ and } \beta \text{ are strings)}$$

Add

$$E \rightarrow \alpha D.\beta \quad k$$

to the closure of state set i.

Intuitively, state set i is the state set we were in when we went looking for that D. We have now found it, so we go to all the states in that state set which caused us to look for a D and we move the dot over the D.

In the example, we add to state set 1,

$$T \rightarrow P. \quad 0$$

and this causes us to compute its closure, adding

$T \rightarrow T.*P \quad 0$

$E \rightarrow T. \quad 0$

and finally

$E \rightarrow E.+T \quad 0$

$\phi \rightarrow E.\dashv \quad 0$

We now scan the next symbol and repeat the operations. If we come up with the state set consisting of the single state

$\phi \rightarrow R\dashv. \quad 0$

then we have scanned the string and it is legal.

This completes the algorithm.

We will try it on grammar AE and the following string: a + a * a

State Set	Part of String Scanned	Original States	Added in Closure
0		$\phi \rightarrow .E\dashv$ 0	$E \rightarrow .E+T$ 0
			$E \rightarrow .T$ 0
			$T \rightarrow .T*P$ 0
			$T \rightarrow .P$ 0
			$P \rightarrow .a$ 0
1	a	$P \rightarrow a.$ 0	$T \rightarrow P.$ 0
			$T \rightarrow T.*P$ 0
			$E \rightarrow T.$ 0
			$E \rightarrow E.+T$ 0
			$\phi \rightarrow E.\dashv$ 0
2	a +	$E \rightarrow E+.T$ 0	$T \rightarrow .P$ 2
			$T \rightarrow .T*P$ 2
			$P \rightarrow .a$ 2

3	a + a	P → a.	2	T → P.	2
				T → T.*P	2
				E → E+T.	0
				E → E.+T	0
				ϕ → E.⊣	0
4	a + a *	T → T*.P	2	P → .a	4
5	a + a * a	P → a.	4	T → T*P.	2
				T → T.*P	2
				E → E+T.	0
				E → E.+T	0
				ϕ → E.⊣	0
6	a + a * a ⊣	ϕ → E⊣.	0		

The technique of using state sets and the first closure operation are derived from Knuth's work on LR(k) languages [1]. In fact, our algorithm reduces to Knuth's algorithm for LR(0) grammars except for the fact that we don't do reductions since ours is a recognition algorithm. Our algorithm actually can be modified to do parsing without unnecessary loss of efficiency. It can also be modified to include Knuth's k symbol look-ahead in the cases where that would increase its efficiency.

The algorithm also seems to be able to recognize in time proportional to n, a large class of grammars including LR(k) [1], bounded context [2], finite unions of these, and some others. These include the grammars for which there exists a fixed bound on the size of any state set. We will attempt to characterize the grammars for which the time or space is proportional to n is a later paper.

DEFINITIONS

We have two disjoint sets of symbols, the non-terminals and the terminals. We will use capitals for non-terminals, lower case letters for terminals, Greek letters for strings of symbols, and X's for either terminals or non-terminals. Λ is the empty string.

A production is of the form $A \rightarrow \alpha$. A grammar is a set of productions with a certain one of its non-terminals designated the root R of the grammar. Most of the rest of the definitions are understood to be with respect to a particular grammar. We write $\alpha \Rightarrow \beta$ if $\exists\ \gamma, \delta, \epsilon, A$ such that $\alpha = \gamma A \delta$ and $\beta = \gamma \epsilon \delta$ and $A \rightarrow \epsilon$ is a production. 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

$$\alpha = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_n = \beta.$$

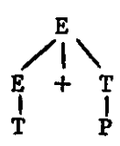
A sentential form is a string α such that $R \overset{*}{\Rightarrow} \alpha$. A sentence is a sentential form consisting entirely of terminal symbols. The language defined by a grammar is the set of its sentences. We may represent any sentential form in at least one way as a derivation tree reflecting the steps made in deriving it (though not the order of the steps). For example, in grammar AE, either derivation

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow T + P$$

or

$$E \Rightarrow E + T \Rightarrow E + P \Rightarrow T + P$$

is represented by



A sentence is unambiguous if it has a unique derivation tree.

A grammar is unambiguous if each of its sentences is unambiguous.

Number the productions arbitrarily 1, ..., g where each production is of the form

$$D_p \rightarrow C_{p1} \ldots C_{pn_p} \qquad 1 \leq p \leq g$$

Add a 0th production

$$D_0 \rightarrow R \dashv$$

where R is the root of the grammar and $\dashv$ is a new terminal symbol.

A state is a triple of integers (p, j, f). Intuitively we will consider p to be a production number, j the number of symbols in the production we have scanned, and f is a state set number. A final state is a state of the form (p, n_p, f).

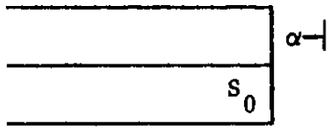
At any particular point in the recognition, when we have scanned i symbols, we represent the condition of the input string and the state sets by

$X_1 \ldots X_i$	$\beta\dashv$
$S_0\ S_1 \ldots S_i$	

where $X_1 \ldots X_i$ are the symbols we have scanned, β is the rest of the input string and $S_0 \ldots S_i$ are the state sets.

THE ALGORITHM

Start the algorithm in condition:



where α is the input string and S_0 is $\{(0,0,0)\}$. Set i to 0. Go to A.

A: Scan the head symbol of α. Call it X_{i+1}.

Construct the closure S_i' of S_i.

Let $S_{i+1} = \{(p,j+1, f) \mid \exists(p, j, f) \text{ in } S_i' \text{ and } C_{p(j+1)} = X_{i+1}\}$.

If S_{i+1} is empty reject the input string.

If $S_{i+1} = \{(0,2,0)\}$ accept the input string.

Otherwise increase i by 1 and go to A.

S_i' is saved for later use; S_i need not be.

The closure S_i' of a state set S_i is computed as follows: Add all states of S_i to S_i'. Scan the states in S_i' in some order performing the following operations on each state s:

(1) If $s = (p, j, f)$ is non-final, then for each q such that $C_{p(j+1)} = D_q$ add $(q, 0, i)$ at the end of S' (so that it will be scanned) unless it is already a member.

(2) If $s = (p, n_p, f)$ is a final state, then for each q such that (q, j, k) is in S_f' and $C_{q(j+1)} = D_p$, add $(q, j+1, k)$ at the end of S_i' unless it is already a member. Call (q, j, k) the parent of (p, n_p, f)

TIME BOUND

Theorem 1: Let G be a grammar with g-1 productions. Let m-1 be the maximum n_p, $0 \leq p \leq g-1$. Let α be a string of length n-1. Let an operation be an algorithm which takes a fixed amount of time independent of the size of the grammar or the string being processed.

(1) There is an upper bound of the form $g^2mn^3/3 + O(n^2)$ on the number of operations in recognizing α with respect to G.

(2) If G is unambiguous, there is an upper bound at the form $gmn^2+O(n)$ on the number of operations in recognizing α with respect to G.

Proof:

(1) There are at most gmi states in any state set S_i.

(2) The following are operations:

a. Test whether a state is in a state set: For each state set we construct a 3-dimensional boolean matrix to represent the existence or absence of any triple.

b. Find a non-terminal which appears immediately to the right of a dot in a state set: For each state set we keep a list of all such non-terminals.

c. Find an alternative of a non-terminal: For each non-terminal we keep a list of each of its alternatives.

d. Find a state in a state set which is a parent of another state: For each state set and for each non-terminal we keep a list of the states which have that non-terminal immediately to the right of the dot.

(3) To compute the first closure function on a state set S_i, it takes at most g operations. There is one step for each non-terminal,

and each such step can involve one operation for each alternative of the non-terminal.

(4) a. To compute the second closure function on a state set S_i, it takes at most [gi] [gmi] operations. There are gi steps, one for each final state in S_i, and each such step can involve up to gmi operations (one for each parent state).

b. However, at most gmi states can be added by the second closure operation. Assume that more than one of the second closure operations produced the same state. Let two of them be as follows:

$$(p_1, n_{p_1}, f_1) \in S_i \ , \ (p_2, n_{p_2}, f_2) \in S_i,$$
$$(q, j, k) \in S_{f_1} \ , \ (q, j, k) \in S_{f_2}, \ D_{p1} = C_{q(j+1)} = D_{p2},$$

and either $p_1 \neq p_2$ or $f_1 \neq f_2$ so that $(q, j+1, k)$ is added to S_i in both ways.

So we have

$$X_1 \ldots X_k C_{q1} \ldots C_{q(j+1)} \overset{*}{\Rightarrow} X_1 \ldots X_{f_1} C_{p_1 1} \ldots C_{F_1 n_{p_1}} \overset{*}{\Rightarrow} X_1 \ldots X_i$$

and

$$X_1 \ldots X_k C_{q1} \ldots C_{q(j+1)} \overset{*}{\Rightarrow} X_1 \ldots X_{f_2} C_{p_2 1} \ldots C_{p_2 n_{p_2}} \overset{*}{\Rightarrow} X_1 \ldots X_i$$

and since $p_1 = p_2$ and $f_1 = f_2$ can't both be true, the above 2 derivations of $X_1 \ldots X_i$ are represented by different derivation trees. So there exists an ambiguous sentence $X_1 \ldots X_i \alpha$ for some α. Thus, in the case that the grammar is unambiguous, the second closure function takes at most gmi operations.

(5) To compute S_{i+1}, from S_i' it takes at most gmi operations, one for each state in S_i'.

(6) Combining (3), (4), and (5), it takes at most

$$g + [gi][gmi] + gmi = g + g^2mi^2 + gmi$$

operations to compute S_{i+1} from S_i in general, and it takes at most

$$g + gmi + gmi = g + 2gmi$$

operations in the unambiguous case.

(7) Summing each of these for i = 1, ..., n gives

general:

$$\sum_{i=1}^{n} [g + g^2mi^2 + gmi]$$

$$= ng + g^2m \sum_{i=1}^{n} i^2 + gm \sum_{i=1}^{n} i$$

$$= ng + g^2mn(n+1)(2n+1)/6 + gmn(n+1)$$

$$\approx g^2mn^3/3$$

unambiguous:

$$\sum_{i=1}^{n} [g + 2gmi]$$

$$= ng + 2gm \sum_{i=1}^{n} i$$

$$= ng + 2gmn(n+1)/2$$

$$\approx gmn^2$$

SPACE BOUND

Theorem 2: Under the assumptions of Theorem 1, there is an upper bound proportional to $gmn^2/2 + O(n)$ on the space required in recognizing α with respect to G.

Proof:

Since the maximum number of states in a state set S_i is gmi, we can sum that over the number of state sets to get a bound on the total number of states:

$$\sum_{i=1}^{n} gmi = gmn(n+1)/2 \approx gmn^2/2$$

The storage required for the matrices and lists of (2)a through (2)d of Theorem 1 (and of course the storage for the state sets themselves) is at most proportional to the number of possible states.

RECOGNIZER PROOF

Theorem 3: If state set S'_i contains state (p, j, f) then

$C_{p1} \cdots C_{pj} \overset{*}{\Rightarrow} X_{f+1} \cdots X_i$.

Proof:

By induction on i.

It is vacuously true in S_0 since j=0 and f=i=0. All states in S'_0 must also have j=0 and f=i=0, so the theorem is also vacuously true for S'_0. Now, assume that it is true for S'_f, for all $f \leq i$. So for $(p, j, f) \in S'_i$,

$$C_{p1} \cdots C_{pj} \overset{*}{\Rightarrow} X_1 \cdots X_i$$

All states in S_{i+1} are of the form (p, j+1, f) where $C_{p(j+1)} = X_{i+1}$, so

$$C_{p1} \cdots C_{pj}C_{p(j+1)} \overset{*}{\Rightarrow} X_1 \cdots X_iX_{i+1}$$

Therefore, the theorem holds for S_{i+1}. Any state in S'_{i+1} was computed from a state in S'_{i+1} for which the theorem holds by either of the closure operations. In the case of closure operation 1, only states for which j=0 and f=i+1 are added, so the theorem holds vacuously for them. In the case of operation 2, suppose that that (q, j+1, k) is added because (p, n_p, f) is in S'_{i+1}, (q, j, k) is in S'_f, and $C_{q(j+1)} = D_p$. We have

$$C_{p1} \cdots C_{pn_p} \overset{*}{\Rightarrow} X_{f+1} \cdots X_{i+1}$$

and so

$$D_p \overset{*}{\Rightarrow} X_{f+1} \cdots X_{i+1}$$

so

$$C_{q(j+1)} \overset{*}{\Rightarrow} X_{f+1} \cdots X_{i+1}$$

Combining these we get

$$C_{q1} \cdots C_{qj}C_{q(j+1)} \overset{*}{\Rightarrow} X_{k+1} \cdots X_fX_{f+1} \cdots X_{i+1}$$

So the theorem holds for S'_{i+1}

This proves the theorem.

Corollary:

If the algorithm accepts an input string, then it is a sentence.

Proof:

If the algorithm accepts α, then $S_n = \{(0,2,0)\}$, and by Theorem 3,

$$R \dashv \overset{*}{\Rightarrow} \alpha \dashv$$

so α is a sentence.

Theorem 4: If S'_i contains (p,j,f) and

$$C_{p(j+1)} \overset{*}{\Rightarrow} X_{i+1} \cdots X_k$$

then S'_k contains $(p,j+1,f)$.

Proof:

By induction on m in the definition of '$\overset{*}{\Rightarrow}$'.

If m=0, then k=i+1 and $C_{p(j+1)} = X_{i+1}$ so S_{i+1} contains $(p,j+1,1)$. Otherwise, $\exists$ q such that

$$C_{p(j+1)} = D_q \rightarrow C_{q1} \cdots C_{qn_q}$$

and $\exists\ k_0 \le k_1 \le \cdots \le k_{n_q}$ such that $k_0=i$, $k_{n_q} = k$ and

$$C_{q1} \overset{*}{\Rightarrow} X_{k_0+1} \cdots X_{k1}$$

$$C_{q2} \overset{*}{\Rightarrow} X_{k_1+1} \cdots X_{k_2}, \ \ldots, \ C_{qn_q} \overset{*}{\Rightarrow} X_{k_{n_q-1}+1} \cdots X_{k_{n_q}}$$

Because of the first closure operation on S_i, $(q,0,i)$ will be in S_i. And by inductive hypothesis we know that if (q,r,i) is in S_{k_r}, then $(q,r+1,i)$ is in S_{k_r}, $r=0,\ldots,n_q-1$, since

$$C_{q_r} \overset{*}{\Rightarrow} X_{k_r+1} \cdots X_{k_{r+1}}$$

So by induction on r, $S_{k_{n_q}} = S_k$ contains (q, n_q, i). But the second closure operation, acting on this state, adds $(p, j+1, f)$ to S'_k. This completes the proof.

Corollary:

If the input string is a sentence, it will be accepted by the algorithm.

Proof:

S'_0 contains $(0,0,0)$ and

$$R \overset{*}{\Rightarrow} X_1 \ldots X_n$$

so by Theorem 4, S'_n contains $(0,1,0)$. And since $\dashv = C_{02}$, S_{n+1} contains $(0,2,0)$. This is the only state it can contain, since $\dashv$ appears nowhere else in the grammar. So the algorithm accepts the input string.

The corollaries to Theorems 3 and 4 together show that the algorithm is a recognizer for all context-free grammars.

ACKNOWLEDGEMENT

I am indebted to Professor Robert W. Floyd for his help in preparing this paper.

REFERENCES

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

[2] Floyd, R. W., "Bounded context syntax analysis," Comm.ACM 7, 1964, 62-66.