MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence
Memo. No. 155

MAC-M-367
February 1968

A Left to Right then Right to Left Parsing Algorithm

by William A. Martin

ABSTRACT

Determination of the minimum resources required to parse a language generated by a given context free grammar is an intriguing and yet unsolved problem. It seems plausible that any unambiguous context free grammar could be parsed in time proportional to the length, n, of each input string. Early (2) has presented an algorithm which parses "many" grammars in time proportional to n, but requires n^2 on some. His work is an extension of Knuth's (4) algorithm, which leads to a very efficient parse proportional to n of deterministic languages. This Memo. presents a different extension of Knuth's method. Knuth's method fails when more than one alternative must be examined by a push down automaton making a left to right scan of the input string. Early's extension takes all possible alternatives simultaneously without duplication of effort at any given step. The method presented here continues through the string in order to gain information which will resolve the conflict in the ensuing right to left pass, which is made on the symbols accumulated on the stack of the automaton. The algorithm is probably more efficient than Early's on certain grammars; it will fail completely on others. The essential idea may be interesting to those attacking the general problem.

I. Introduction

I will assume that the reader is familiar with context free languages and the notation given in Knuth's paper (4).

The need to design computer languages makes it important to understand which languages can be easily parsed. No general unambiguous context free language parsing algorithm has been proposed which does not require time at least proportional to n^2, where the string to be parsed is n symbols long. On the other hand, we have not yet been able to find an unambiguous grammar which could not be parsed in time proportional to n by some algorithm. The algorithm below handles a wide class of grammars.

The handle of a sentential form is defined as the left most string of characters in the sentential form which equals the right side of some rule. Knuth has treated grammars where every handle can be found by considering the characters to its left, the characters in the handle, and some fixed number for all handles, k, of characters to the right of the handle. Such a grammar can be parsed in time proportional to n by a deterministic PDA, scanning the string from left to right. For example, the grammer G1:

$$S \rightarrow A$$
$$S \rightarrow cB$$
$$A \rightarrow aAb$$
$$A \rightarrow ab$$
$$B \rightarrow aBbb$$
$$B \rightarrow abb$$

has the sentential forms

1. $\underline{A}$
2. $a^n\underline{aAb}b^n$
3. $a^n\underline{ab}b^n$
4. $\underline{cB}$
5. $ca^n\underline{aBbbb}^{2n}$
6. $ca^n\underline{abbb}^{2n}$

where the handle has been underlined in each case.

The handle in 6. is distinguished from the handle in 3. by the "c" at the left end of the string. It is not necessary to look at characters to the right of the handle, so k=0, and the grammar is said to be LR(0). On the other hand, grammar G1 cannot be parsed in this manner from the right, for one must look arbitrarily far ahead to find the "c" in order distinguish 6. from 3. By similar arguments we conclude that a grammar which generates the language with strings of the form

$$ca^nb^nb^na^n$$
$$a^nb^nb^{2n}a^nd$$
$$ca^nb^{2n}b^na^nd$$
$$a^nb^{2n}b^{2n}a^n$$

can be parsed from neither the left nor the right. In this memo, we extend Knuth's method to handle languages of this type.

Knuth thinks of the parser as a finite state machine with a push down stack. At each step the parser must either add an input symbol to its

stack or must reduce the stack by recognizing that the symbols at the top of the stack correspond to a handle and replace these symbols on the stack by the left side of the corresponding rule. In general, this will require information about characters to the left, such as the "c" in example G1. Knuth showed that all the useful information about characters, or reductions already made, to the left can be represented by one of a finite number of states.

The state is stored on the stack after each symbol which is added to the stack. The new state is computed from the old state on the top of the stack and the symbol added. When the stack is reduced state symbols for the symbols in the handle are thrown away.

The method gets into trouble when it is not possible to determine a unique handle with the allowable information. For example, parsing G1 from the right, we would reach the point $abbb^n$, the handle could be either ab or abb. Early (2) allows his parser to explore both possibilities by maintaining a two-dimensional array instead of a linear stack. The method used here will be to let the finite state language which summarizes what has been found thus far become conceptually non-deterministic. We will take a path for each of the possibilities which could arise if the stack was reduced. The stack will not be reduced so that no information will be lost. Without reducing the stack we cannot see what state symbol would have ended up on top of the stack after the reduction, so more information will have to be brought along by the non-deterministic finite state summarizing

language. Even though we conceptually take several paths, we often constrain the possible parsings for the rest of the string to the right. Sometimes as the parser moves farther to the right, input symbols will be found which show that some of the paths cannot apply to the string. The stack can be reduced whenever all current paths indicate the same reduction. Once the right end of the string has been reached the stack is parsed from the right using Knuth's algorithm; except that the state symbols from the left to right pass are used to restrict the possibilities which the algorithm considers.

II. Description of the Algorithm

We follow Knuth in the description, except that productions of the form $A \to \varepsilon$ will not be treated since they add even more complexity. We define the algorithm to be LR(k1) and RL(k2) for the k1 and k2 used below.

First, let $H_{k1}(\sigma)$ be the set of all k1-letter strings β over $T \cup \{\#\}$ such that $\sigma \overset{*}{\Rightarrow} \beta\alpha$, for some α. Next, suppose the p^{th} production of the grammar to be parsed has the form $A \to X_1 \ldots X_{n_p}$. A <u>state</u> $[p,j;\alpha,m,t]$ represents the first j letters of the p^{th} production, $0 \le j \le n_p$ and a string α which could follow the left side of the p^{th} production in some sentential form. We also introduce a <u>virtual state</u>, $(p,j;\alpha,m,t)$. Virtual states are interpreted the same as states, but the algorithm uses them differently. m is any integer identifying this state or virtual state from all others created during the parse and t is a list of integers identifying

other states or vitual states. During the translation process we maintain a stack, denoted by

$$\#^{k2}S_0X_1S_1X_2S_2\ldots X_nS_n|Y_1\ldots Y_k\omega\#^{k1} \tag{1}$$

The portion to the left of the vertical line consists alternately of state sets and characters; this is the portion of the string which has been considered by the left to right pass. The state sets contain both regular and virtual states. The steps for the left to right pass are given below. We start the stack with k2 #'s and then enter the following loop, with n=0 and $S_0 = \{[0,0;\ \epsilon\ ,1,()]\}$. At each step, assume the stack contents are as shown in (1) and S=S_n.

Step 1. Compute the "closure" S' of S, which is defined recursively as the smallest set satisfying the following equation:

$S' = S \cup \{[q,0;\beta,m,(m1)] |$ there exists $[p,j;\alpha,m1,t]$ in S', $j<n_p$, or there exists $(p,j;\alpha,m1,t)$ formed by the previous application of Step 3, $j<n_p$, and $X_{p(j+1)} = A_q$, and β in $H_{k1}\ (X_{p(j+2)}\ldots X_{pn_p}\alpha)]$

(We thus have added to S all productions that might apply in addition to those we are already working on.)

Step 2. Compute the following sets of k1 letter strings:

$Z = \{\beta |$ there exists $[p,j;\alpha]$ in S', $j<n_p$,

β in $H_k\ (X_{p(j+1)}\ldots X_{pn_p}\alpha)]$

$Z_p = \{\alpha \mid [p,n_p;\alpha]$ in $S'\}$ $0 \le p \le s$, the number of productions

$Z, Z_0, \ldots, Z_s$ must all be disjoint or the grammer is not LR(k1) and virtual states must be formed. a) If $Y_1 \ldots Y_{k1}$ lies in Z and not in any Z_p, shift the stack left:

$$\#^{k2} S_0 X_1 S_1 \ldots S_n Y_1 | Y_2 \ldots Y_{k1} \omega \#^{k1}$$

and rename its contents by letting $X_{n+1} = Y_1$, $Y_1 = Y_2, \ldots$, and go to Step 3. b) If $Y_1 \ldots Y_{k1}$ lies in exactly one Z_p and not in Z, then the characters forming the right side of production p are on the top of the stack. Check to see if state sets $S_{n-(n_p-1)} \ldots S_{n-1}$ contain any states $[q,j;\beta]$ for which $j=n_q$. If this is not so, the stack can be reduced. Let $r=n-n_p$; the stack now contains $X_{r+1} \ldots X_n, S_n$, replace this string by A_p to obtain $\#^{k2} S_0 X_1 S_1 \ldots X_r S_r A_p | Y_1 \ldots Y_k \omega$ and let $n=r$, $X_{n+1} = A_p$. Now go to Step 3. Otherwise, shift the stack left as in a) above and go to Step 3.

Step 3. The stack now has the form

$$S_0 X_1 S_1 \ldots X_n S_n X_{n+1} | Y_1 \ldots Y_k \omega$$

Let $S''_{n+1} = \{[p,j+1;\alpha,r,t] \mid [p,j;\alpha,r,t]$ in S'_n, $j<n_p$, and $X_{n+1} = X_{p(j+2)}\}$

$$\cup\{(p,j+1;\alpha,m,t) \mid (p,j;\alpha,r,t) \text{ in } S_n', \; j<n_p,$$
$$\text{and } X_{n+1} = X_{p(j+1)}\}$$

$$\cup\{(p,j+1;\alpha,m,t) \mid (p,j;\alpha,r,t) \text{ or } \lfloor p,j;\alpha,r,t \rfloor \text{ in } S_n', \; j<n_p,$$
$$\text{and } (q,n_q;\beta,u,(\ldots r\ldots)) \text{ or } \lfloor q,n_q;\beta,u,(\ldots r\ldots) \rfloor \text{ in } S_{n+1}\}$$

$$\cup\{(p,j;\alpha,m,t) \mid (p,j;\alpha,m,t) \text{ in } S_n', \; j<n_p,$$
$$\text{and } (q,i;\beta,u,(\ldots m\ldots)) \text{ or } \lfloor q,i;\beta,u,(\ldots m\ldots) \rfloor \text{ in } S_{n+1}, i<n_q\}$$

Now form S_{n+1} from S_{n+1}'' by (a) merging all states $\lfloor p,j;\alpha,m,t \rfloor$, $\lfloor p,j;\alpha,n,s \rfloor$ by forming a state $\lfloor p,j;\alpha,m,s\cup t \rfloor$ and changing the references to n to state m, and (b) doing the same for all such pairs of virtual states. I indicate the merge as a separate step because it is thus easier to explain. If the first character to the right of the bar is "#", begin the right to left pass through the stack, otherwise, go to Step 1.

The right to left pass is done in the same manner as the left to right pass with three exceptions. First, no virtual states are formed; if the algorithm reaches a point where a virtual state would be formed, it reports failure. Second, since we are going right to left, it is necessary to sequence through the characters of the right-hand side of a production in the opposite direction. When a new state is formed by Step 1, the form is $\lfloor q, n_{q+1};\beta,m,(m1) \rfloor$. Then, in Step 3, j is decremented instead of incremented.

Third, after forming the state S_n' by Step 1, called $\bar{S}_n$ in the example, $\bar{S}_n$ is replaced by $\bar{\bar{S}}_n$ formed as follows. Let S_m be the left-to-right state set at the top of the input.

$$\bar{\bar{S}}_n = \{[p,j;\alpha] \mid [p,j;\alpha] \text{ is in } \bar{S}_n \text{ and } [p,j-1,\beta] \text{ is in } S_m\}$$

III. Example

The grammar

0. $\bar{S} \rightarrow S\#$
1. $S \rightarrow cBaA$
2. $S \rightarrow cAA$
3. $S \rightarrow BaB$
4. $S \rightarrow AB$
5. $A \rightarrow aAb$
6. $A \rightarrow ab$
7. $B \rightarrow aBbb$
8. $B \rightarrow abb$

produces the strings

1. $ca^n b^{2n} a^{n+1} b^n$
2. $ca^n b^n a^n b^n$
3. $a^n b^{2n} a^{n+1} b^{2n}$
4. $a^n b^n a^n b^{2n}$

We take k1=1 and k2=0.

The major steps in parsing a string of the first type are shown on the following pages. The zero production is not shown in the state sets. Since the stack can expand and contract, the state sets have been given two subscripts. The second is the subscript used in Section II. The first counts the number of times that the stack has reached that length.

In the notation for states or vitual states, the set t has been omitted when it is empty. If t is a set of one element, that element is shown.

Steps to Parse String 1.

$\#S_{00}|\ ca^nb^{2n}a^{n+1}b^n\#$

$\#S_{0,0}cS_{0,1}|\ a^nb^{2n}a^{n+1}b^n\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}|\ a^{n-1}b^{2n}a^{n+1}b^n\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}|\ b^{2n}a^{n+1}b^n\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}|\ b^{2n-1}a^{n+1}b^n\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n+1}|\ a^{n+1}b^n\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n+1}aS_{0,3n+2}|\ a^nb^n\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n+1}aS_{0,3n+2}\cdots aS_{0,4n+1}|\ b^n\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n+1}aS_{0,3n+2}\cdots \underline{aS_{0,4n+1}bS_{0,4n+2}}|\ b^{n-1}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n+1}aS_{0,3n+2}\cdots aS_{0,4n}AS_{1,4n+1}|\ b^{n-1}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n+1}aS_{0,3n+2}\cdots \underline{aS_{0,4n}AS_{1,4n+1}bS_{1,4n+2}}|b^{n-2}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n+1}aS_{0,3n+2}AS_{1,3n+3}\quad|\bar{S}_{0,0}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n+1}aS_{0,3n+2}AS_{1,3n+3}\quad|\bar{\bar{S}}_{0,0}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n+1}aS_{0,3n+2}\quad|\bar{\bar{S}}_{0,1}A\bar{\bar{S}}_{0,0}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n+1}\quad|\bar{\bar{S}}_{0,2}a\bar{\bar{S}}_{0,1}A\bar{\bar{S}}_{0,0}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}bS_{0,n+2}\cdots bS_{0,3n}\quad|\bar{\bar{S}}_{0,3}b\bar{\bar{S}}_{0,2}a\bar{\bar{S}}_{0,1}A\bar{\bar{S}}_{0,0}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n+1}\quad|\bar{\bar{S}}_{0,2n+2}b\cdots\bar{\bar{S}}_{0,3}b\bar{\bar{S}}_{0,2}a\bar{\bar{S}}_{0,1}A\bar{\bar{S}}_{0,0}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n}\quad|\underline{\bar{\bar{S}}_{0,2n+3}a\bar{\bar{S}}_{0,2n+2}b\bar{\bar{S}}_{0,2n+1}}b\cdots\bar{\bar{S}}_{0,3}b\bar{\bar{S}}_{0,2}a\bar{\bar{S}}_{0,1}A\bar{\bar{S}}_{0,0}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n}\quad|\bar{\bar{S}}_{1,2n+1}B\bar{\bar{S}}_{0,2n}b\cdots\bar{\bar{S}}_{0,3}b\bar{\bar{S}}_{0,2}a\bar{\bar{S}}_{0,1}A\bar{\bar{S}}_{0,0}\#$

$\#S_{0,0}cS_{0,1}aS_{0,2}\cdots aS_{0,n-1}\quad|\underline{S_{1,2n+2}a\bar{\bar{S}}_{1,2n+1}B\bar{\bar{S}}_{0,2n}b\bar{\bar{S}}_{0,2n-1}}b\cdots\bar{\bar{S}}_{0,3}b\bar{\bar{S}}_{0,2}a\bar{\bar{S}}_{0,1}A\bar{\bar{S}}_{0,0}\#$

$\#S_{0,0}cS_{0,1}\quad|\bar{\bar{S}}_{1,3}B\bar{\bar{S}}_{0,2}a\bar{\bar{S}}_{0,1}A\bar{\bar{S}}_{0,0}\#$

$\#S_{0,0}\quad|\underline{\bar{\bar{S}}_{1,4}c\bar{\bar{S}}_{1,3}B\bar{\bar{S}}_{0,2}a\bar{\bar{S}}_{0,1}}A\bar{\bar{S}}_{0,0}\#$

$\#S_{0,0}\quad|S\bar{\bar{S}}_{0,0}\#$

Representative State Sets to Parse String 1.

$S_{0,0}$ = ([1,0;#,1] [2,0;#,2] [3,0;#,3] [4,0;#,4] [5,0;a,5,4] [6,0;a,6,4] [7,0;a,7,3] [8,0;a,8,3])

$S_{0,1}$ = ([1,1;#,1] [2,1;#,2] [5,0;a,11,2] [6,0;a,12,2] [7,0;a,13,1] [8,0;a,14, 1])

$S_{0,2}$ = ((1,1;#,1) (2,1;#,2)[5,1;a,11,2] [6,1;a,12,2] [7,1;a,13,1] [8,1;a,14,19] [5,0;b,19,11] [6,0;b,20,11] [7,0;b,21,13] [8,0;b,22,13])

$S_{0'n+1}$ = ((1,1;#,1) (2,1;#,2) (5,1;a,15,2) (7,1;a,17,1) (5,1;b,23,(15,23)) (7,1;b,24,(17,24)) [5,1;b,25,23] [6,1;b,26,23] [7,1;b,27,24] [8,1;b,28,24] [5,0;b,29,25] [6,0;b,30,25] [7,0;b,31,27] [8,0;b,32,27])

$S_{0,n+2}$ = ((1,1;#,1) (2,1;#,2) (5,1;a,15,2) (7,1;a,17,1) (5,1;b,23,(15,23)) (7,1;b,24,(17,24)) [6,2;b,26,23] [8,2;b,28,24] (5,2;b,35,(15,23))

$S_{0,n+3}$ = ((1,1;#,1) (2,1;#,2) (5,1;a,15,2) (7,1;a,17,1) (5,1;b,23,(15,23)) (7,1;b,24,(17,24)) [8,3;b,28,24] (5,3;b,36,(15,23)) (5,2;b,37,(15,23)) (7,2;b,38,(17,24))

$S_{0,n+4}$ = ((1,1;#,1) (2,1;#,2) (5,1;a,15,2) (7,1;a,17,1) (5,1;b,23,(15,23)) (7,1;b,24,(17,24)) (5,2;a,42,9) (5,2;b,41,(15,23)) (5,3;b,40,(15,23)) (7,3;b,39,(17,24))

$S_{0,n+5}$ = ((1,1;#,1) (2,1;#,2) (5,1;a,15,2) (7,1;a,17,1) (5,1;b,23,(15,23)) (7,1;b,24,(17,24)) (5,3;a,47,9) (5,3;b,46,(15,13)) (7,2;a,45,1) (7,2;b,44,(17,24)) (7,4;b,43,(17,24))

$S_{0,3n+1}$ = ((1,1;#,1) (2,1;#,2) (5,1;a,15,2) (5,2;a,50,2) (5,1;b,23,(15,23))
(5,2;b,52,(15,23)) (5,3;b,53,(15,23)) [5,0;#,62,63] [6,0;#,63,61]
(1,2;#,60) (2,2;#,61) (7,1;a,17,1) (7,2;a,54,1) (7,3;a,55,1)
(7,4;a,56,1) (7,1;b,24,(17,24) (7,2;b,57,(17,24)) (7,3;b,58,(17,24))
(7,4;b,59,(17,24)(5,3;a,51,9))

$S_{0,3n+2}$ = ((1,3;#,66) [5,1;#,62,61] [6,1;#,63,61] [5,0;b,64,62] [6,0;b,65,62]
[5,0;#,67,66] [6,0;#,68,66] (2,2;#,61))

$S_{0,3n+3}$ = ((1,3;#,66) (2,2;#,61) (5,1;#,62,61) [5,1;#,67,66] [6,1;#,68,66]
[5,1;b,64,62] [6,1;b,65,62] [5,0;b,69,(67,64)] [6,0;b,70,(67,64)])

$S_{0,4n+1}$ = ((1,3;#,66) (2,2;#,61) (5,1;#,62,(61,66)) (5,1;b,64,(62,64))
[5,1;b,69,64] [6,1;b,70,64] [5,0;b,71,69] [6,0;b,72,69])

$S_{0,4n+2}$ = ((1,3;#,66) (2,2;#,61) (5,1;#,62,(61,66)) (5,1;b,64,(62,64))
(5,2;b,73,(62,64)) [6,2;b,70,64])

$S_{1,4n+1}$ = ((1,3;#,66) (2,2;#,61) (5,1;#,62,(61,66)) (5,1;b,64,(62,64))
(5,2;b,75,(62,64)) [5,2;b,69,64])

$S_{1,4n+2}$ = ((1,3;#,66) (2,2;#,61) (5,1;#,62,(61,66)) (5,1;b,64,(62,64))
(5,3;b,76,(62,64)) [5,3;b,69,64] (5,2;b,75,(62,64)))

$S_{1,3n+3}$ = ((1,4;#,66) (2,2;#,61) [5,2;#,62,61])

$\bar{S}_{0,0}$ = ([1,5;ε] [2,4;ε] [3,4;ε] [4,3;ε] [5,4;ε] [6,3;ε] [7,5;ε] [8,4;ε])

$\bar{\bar{S}}_{0,0}$ = ([1,5;ε])

$\bar{\bar{S}}_{0,1}$ = ([1,4;ε])

$$\bar{\bar{S}}_{0,2} = ([1,3;_{\varepsilon}]\ [7,5;_{\varepsilon}])$$

$$\bar{\bar{S}}_{0,3} = ([7,4;_{\varepsilon}])$$

$$\bar{\bar{S}}_{0,2n+2} = ([8,2;_{\varepsilon}])$$

$$\bar{\bar{S}}_{0,2n+3} = ([8,1;_{\varepsilon}])$$

$$\bar{\bar{S}}_{1,2n+1} = ([7,2;_{\varepsilon}])$$

$$\bar{\bar{S}}_{1,2n+2} = ([7,1;_{\varepsilon}])$$

$$\bar{\bar{S}}_{1,3} = ([1,2;_{\varepsilon}])$$

$$\bar{\bar{S}}_{1,4} = ([1,1;_{\varepsilon}])$$

IV Discussion

First, note that the non-deterministic language is indeed finite state. Since there are a finite number of rules in the grammer and each has a finite number of characters on its right side, there are only a finite number of states of the form $(p,j;\alpha,m,t)$ or $[p,j;\alpha,m,t]$ which have distinct p,j and α. Therefore, the set t must be finite and there are also only a finite number of possible state sets.

Since the method of constructing the state sets is quite complex, one might think that the algorithm cannot be very fast. This is not so. The state sets need to constructed only once, then each state set and string of k input charcters determines a new state set. Thus, the important thing is the number of state sets, and whether this number can be reduced for the grammer in question.

In the example, a character at the left end of the string to be parsed gives information needed to make a reduction at the right end. Only after this reduction is made can information needed to make further reductions at the left end be obtained. The example could be expanded so that it could be parsed only if the same type of algorithm made three passes through the string, or so that the algorithm would have to make an arbitrarily large number of passes.

If the algorithm succeeds after some number of passes, it means that the information needed to find the correct reductions could be obtained without having to unwind the pushdown stack in more than one way. Efficiency

is lost because the algorithm which makes a complete pass in one direction and then in the other may have to consider some characters several times. This could perhaps be omitted if the trouble spots were somehow remembered on the first pass.

The language with strings of the form $a^n b^{2n}$ and $a^n b^n$ cannot be parsed by this algorithm. The a's must be counted against the b's in both ways. On the other hand, the language can still be parsed in time proportional to n by Early's scheme. My algorithm will fail on a palindrom language and Early's will require effort proportional to n^2. A palindrom language can be efficiently parsed both ends toward the middle.

It should be possible to find more general methods of parsing in time proportional to the string length.

Bibliography

1. A. Colmerauer, "Relations de Precedence Totale," Institut de Mathematiques Appliquees, Universite de Grenoble, April, 1967.

2. J. Early, "An N^2-Recognizer for Context Free Grammars," Department of Computer Science Report, Carnegie-Mellon University, Pittsburgh, Pennsylvania, September, 1967.

3. S. Ginsburg, The Mathematical Theory of Context Free Languages, McGraw Hill, 1966.

4. D. E. Knuth, On the Translation of Languages from Left to Right," Information and Control 8, 1965.