Computer Sciences Department
The University of Wisconsin
Madison, Wisconsin 53706

AN IMPROVEMENT TO DOMELKI'S ALGORITHM*

by

David S. Wise

Technical Report #94

August, 1970

*Sponsored by the Mathematics Research Center, United States Army, Madison, Wisconsin, under Contract Number: DA-31-124-ARO-D-462.

AN IMPROVEMENT TO DOMELKI'S ALGORITHM

by

David S. Wise

I. INTRODUCTION

Domelki (1) has given an algorithm for parsing sentences of a reduced context free grammar which takes advantage of the commands on a binary computer which extract logical functions on strings of bits (the computer word) in parallel. Lynch (2) has applied it to his overlap resolvable grammars (3) and it is his version of the algorithm which is improved here. The savings can be as much as 50% for the parsing of ALGOL-like languages. An extension efficiently parses the Simple LR(1) languages as described by DeRemer (4).

II. DEFINITIONS

We shall adopt much of the notation of Colmerauer (5) which is summarized here for flavor rather than conciseness.

For any binary relation ρ on a set, let ρ^+ be the transitive completion of ρ, and ρ^* be the reflexive and transitive completion of ρ.

We can define the inverse of ρ, ρ^{-1}, by $a \rho b \iff b \rho^{-1} a$; if ρ and ρ^{-1} are represented as Boolean matrices, then ρ^{-1} is represented by the transpose (not the inverse) of ρ's matrix.

For a set of symbols V, called a <u>vocabulary</u>, let the set of all strings over V be denoted V^*, and let V^+ be all of V^* but the empty string, ε. If

ξ and η are strings then the <u>length</u> of ξ will be denoted $|\xi|$ and the concatenation of ξ and η by $\xi\eta$.

A <u>context free grammar</u> is a quadruple: $G = (V_T, V_N, s, P)$ where

V_T is the <u>terminal vocabulary</u>; $V_N \cap V_T = \emptyset$.

V_N is the <u>non-terminal vocabulary</u>;

From V_N and V_T we get the vocabulary, $V = V_N \cup V_T$.

$s \in V_N$ is the <u>axiom</u>.

$P \subset V_N \times V^*$ is the set of <u>rules</u>.

The projection of P along the first co-ordinate gives the set of <u>phrases</u> of G. We denote the cardinality of P by p, and index P by $1 \leq n \leq p$. (a_n, ξ_n) is the n^{th} rule; ξ_n the n^{th} phrase.

Lower case Roman characters have been used for symbols and indices:

a,b,c,d,e,x, are symbols in V,

i,j,k,m,n,p are indices (or integers),

and lower case Greek for strings and relations.

$\delta, \sigma, \xi, \eta, \zeta$ are strings;

$\alpha, \beta, \gamma, \lambda, \rho$ are relations.

P yields a relation over $V^+ \times V^*$ denoted by $\rightarrow$ where

$$\eta \rightarrow \zeta \iff \eta = \delta\,\alpha\,\sigma \text{ and } \zeta = \delta\,\xi\,\sigma \text{ where}$$

$$\delta, \sigma \in V^* \text{ and } (a, \xi) \in P.$$

This relation extends to $\overset{*}{\rightarrow}$ and $\overset{+}{\rightarrow}$ as noted above.

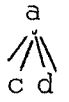
We assume that G is reduced; that is

$$\forall\, a \in V_N \quad \exists \xi \in V_T^* \ \ni\ a \overset{+}{\rightarrow} \xi$$

$$\forall\, b \in V \quad \exists \sigma, \delta \in V^* \ \ni\ s \overset{*}{\rightarrow} \sigma\, b\, \delta.$$

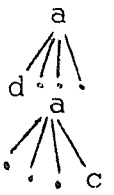
The language defined by G, L(G), is $\{\delta \in V_T^* \mid s \overset{*}{\rightarrow} \delta\}$. The set of sentential forms of G is $\{\delta \in V^* \mid s \overset{*}{\rightarrow} \delta\}$. We shall define three relations, α, λ, and ρ, on V which are determined by P:

$$c\,\alpha\,d \iff \exists a \in V_N \text{ and } \exists \sigma, \delta \in V^* \ \ni\ a \rightarrow \delta c d \sigma;$$



$$a\,\lambda\,d \iff \exists \sigma \in V^* \ \ni\ a \rightarrow d\sigma;$$

$$c\,\rho\,a \iff \exists \delta \in V^* \ \ni\ a \rightarrow \delta c.$$



Note the asymmetry of λ and ρ. It is possible that $(a, \varepsilon) \in P$. (This is an instance of an ε-rule.) Then we must extend α, λ and ρ to allow for 'ghosts':

$$c\,\bar{\alpha}\,d \iff \exists a \in V_N,\ \exists \sigma, \delta \in V^*,\ \exists \eta \in V_N^* \ \ni$$

$$a \rightarrow \delta c \eta d \sigma \text{ and } \eta \overset{*}{\rightarrow} \varepsilon;$$

$$a\,\bar{\lambda}\,d \iff \exists \sigma \in V^*,\ \exists \eta \in V_N^* \ \ni$$

$$a \rightarrow \eta d \sigma \text{ and } \eta \overset{*}{\rightarrow} \varepsilon;$$

$$c\,\bar{\rho}\,a \iff \exists \delta \in V^*,\ \exists \eta \in V_N^* \ \ni$$

$$a \rightarrow \delta c \eta \text{ and } \eta \overset{*}{\rightarrow} \varepsilon.$$

Thence we define the relations β and $\bar{\beta}$:

$$a \beta b \iff a \rho^* \alpha \lambda^* b;$$

$$a \bar{\beta} b \iff a \bar{\rho}^* \bar{\alpha} \bar{\lambda}^* b.$$

The relations α, λ, ρ, and β are read "adjacent to," "left derives," "right derived from," and "by" respectively. We assume that all these relations are extended to the set $V \cup \{ \vdash, \dashv \}$ in the standard manner, where $\vdash$ and $\dashv$ are the left and right endmarkers not in V introduced by an implicit 0^{th} rule in P. ($\xi_0 = \vdash s \dashv$). Let $J = \max_{0 \le n \le p} |\xi_n|$.

From any string η and any integer $0 \le i \le |\eta|$ we adopt the notation $i:[\eta]$ for the prefix string of the first i characters of η, and $[\eta]:i$ for the suffix string of the last i characters. Thus $0:[\eta] = \varepsilon = [\eta]:0$ and $|\eta| : [\eta] = \eta$. If i is out of bounds the meaning is undefined (4), and we shall take relations on such undefined arguments to be false.

Parsing is the mapping of a sentence $\eta \in L(G)$ onto the derivation tree whereby $s \overset{*}{\rightarrow} \eta$. In all the following read "parse any sentence of a grammar" for "parse a grammar."

We redefine Lynch's overlap resolvable grammars (3) by considering members of $P \times P \times \{0 \le i \le J\}$. Let the triple (m,n,i) denote the element $((a_m, \xi_m), (a_n, \xi_n), i)$ from that set, and consider only those triples which satisfy one of the four following conditions:

1) $|\xi_m| \ge i > 0 = |\xi_n|$;

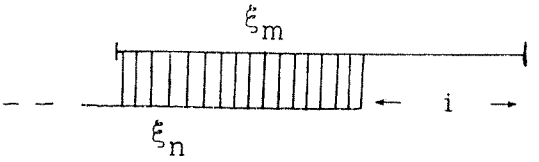
2) $|\xi_m| = i = 0 = |\xi_n|$;

3) $|\xi_m| \ge |\xi_n| > i = 0$;

4) $|\xi_m| > i > 0 < |\xi_n|$;

and the overall restriction (relevant to 2 and 3 only) that $i = 0 \wedge |\xi_m| = |\xi_n|$ implies $m < n$.

Then we shall below define (m,n,i), meeting one of these conditions, to be an overlap when "lining up" ξ_n with ξ_m, so that ξ_m has i of its characters extending to the right beyond the end of ξ_n, causes the "overlapping" characters to match.



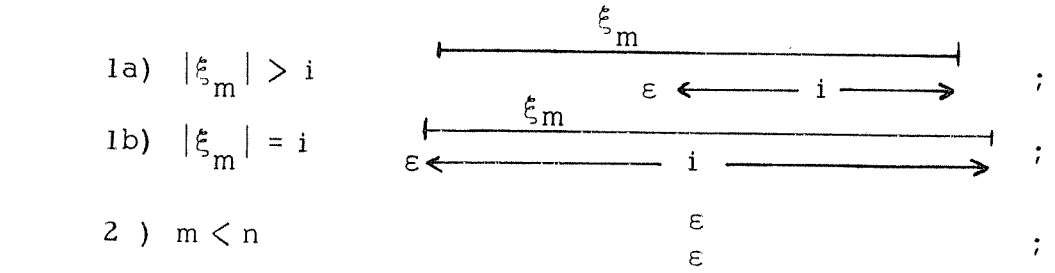
Conditions 1 and 2 define overlaps completely; since $\xi_n = \varepsilon$ the match is vacuous. A triple satisfying condition 3 is defined to be an overlap when $[\xi_m]:|\xi_n| = \xi_n$. A triple satisfying condition 4 is defined to be an overlap when either

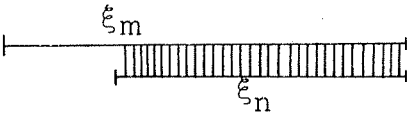
$$|\xi_n| : [[\xi_m]:(|\xi_n| + i)] = \xi_n$$

or $$(|\xi_m| - i):[\xi_m] = [\xi_n] : (|\xi_m| - i).$$

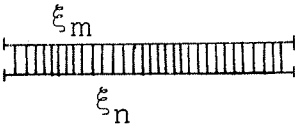
(Note that one of these two disjuncts will be false because of an undefined operand, unless $|\xi_m| = |\xi_n| + i$ in which case they are identical.)

We are interested in resolving every overlap of a grammar. An overlap is resolved if it is resolvable on the right or resolvable on the left. Before defining left resolvability we will further elaborate and illustrate the four conditions:

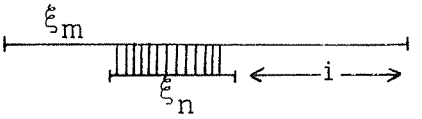


3a) $|\xi_m| > |\xi_n|$ 

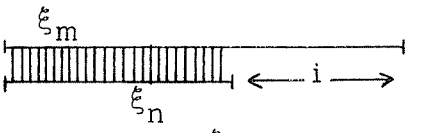
;

3b) $|\xi_m| = |\xi_n|$, $m < n$ 

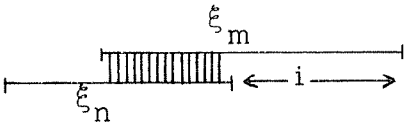
;

4a) $|\xi_m| > |\xi_n| + i$ 

;

4b) $|\xi_m| = |\xi_n| + i$ 

;

4c) $|\xi_m| < |\xi_n| + i$ 

.

As elaborated above 1b, 2, 3b and 4b are even left overlaps, and will be covered only by the first disjunct in the predicate in the definition below. Conditions 1a, 3a, and 4a are one form of uneven left overlap to which only the second disjunct applies; condition 4c is an uneven left overlap covered only by the third disjunct.

An overlap (m,n,i) is defined to be <u>resolvable on the left</u> if

$$\neg\ (|\xi_m| = |\xi_n| + i \quad \wedge \quad a_m\ \lambda^{*-1}\ \alpha^{-1}\ \alpha\ \lambda^*\ a_n$$
$$\vee\ 1{:}[[\xi_m]{:}(|\xi_n| + i + 1)]\quad \alpha\ \lambda^*\ a_n$$
$$\vee\ 1{:}[[\xi_n]{:}(|\xi_m| - i + 1)]\quad \alpha\lambda^* a_m).$$

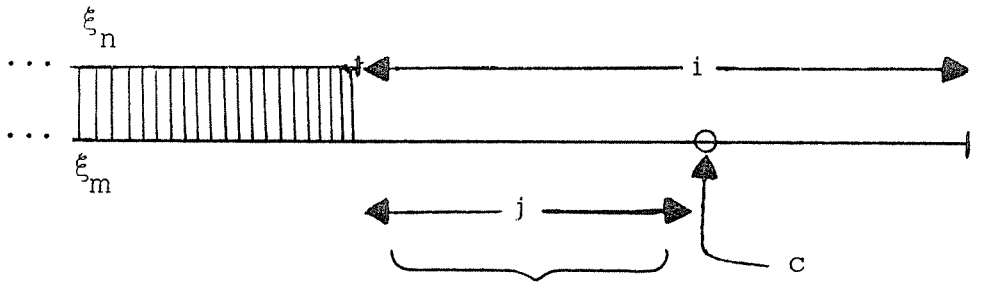
Note that left resolvability is described without regard to the effect of ε-rules; i.e. α and λ are used rather than $\bar{\alpha}$ and $\bar{\lambda}$. In a left to right parse all ε-rules to the left of a phrase will have been parsed before the left resolvability of the overlaps on that phrase are of interest.

Conditions 2 and 3 are even right overlaps; conditions 1 and 4 are uneven

right overlaps. Resolvability on the right may be concisely stated once we describe a predicate:

$$R(m,n,i,j) = j{:}[[\xi_m]{:}i] \overset{*}{\rightarrow} \varepsilon \ \wedge$$
$$(i=j \wedge a_n \bar{\beta}\bar{\beta}^{-1} a_m$$
$$\vee a_n \bar{\beta}\bar{\lambda}^{*-1} 1{:}[[\xi_m]{:}(i-j)]).$$

The interpretation of this predicate will be understood from the illustration of the overlap (m,n,i) and the knowledge that $0 \leq j \leq i$:



where $j{:}[[\xi_m]{:}i] \overset{*}{\rightarrow} \varepsilon.$

The character of interest in phrase ξ_m is $c = 1{:}[[\xi_m]{:}(i-j)]$. Everything between the end of the match with ξ_n and c could possible disappear ($\overset{*}{\rightarrow} \varepsilon$) in a sentential form, so that c could appear adjacent to the end of the match. If this uneven right overlap occurs we are interested in the validity of $a_n \bar{\beta}\bar{\lambda}^{*-1}c$.

When $i = j$, c is undefined. This indicates an even-right overlap whose resolvability will hinge upon $a_n\bar{\beta}\bar{\beta}^{-1} a_m$. For any one overlap, (m,n,i), we shall check $R(m,n,i,j)$ for all j, $0 \leq j \leq i$, because the various substrings of ξ_m following the match with ξ_n may each have potential to disappear in a sentential form.

An overlap (m,n,i), is defined to be resolvable on the right if

$$\neg \exists j (0 \leq j \leq i \quad \wedge \quad R(m,n,i,j))$$

A grammar is generalized overlap resolvable (GOR) if all overlaps are resolved. It is restricted overlap resolvable (ROR) iff it is GOR and has no ε-rules. Lynch's overlap resolvable grammars are the ROR. In testing for ROR we can replace $\bar{\beta}$ and $\bar{\lambda}$ by β and λ in the above, and we note that conditions 1 and 2 never arise and that the definition of right resolvability of (m,n,i) collapses to $\neg$ R(m,n,i,0).

In proposing languages to test for the GOR property, it is clear that overlaps on ε-rules (satisfying condition 1 above) occur "everywhere." Therefore, inclusion of even one ε-rule will generate many overlaps and decrease the chances of resolving every one. It is known how to eliminate ε-rules (6), but semantic associations with rules may make this alternative undesirable compared to more powerful parsing techniques. So the GOR definition might be useful, but in practical situations we shall find the ROR easier to create.

The algorithm described below can be modified to parse much weaker (larger) classes of grammars, but in its basic form it can parse any GOR grammar, and even some non-GOR when the only unresolved overlaps satisfy condition 4c; e.g.

s → tra
s → pbg
a → ig
b → ri .

III. THE ALGORITHM

In describing the algorithm the naming conventions of Lynch (2), whose method appears below as ALGORITHM 2, are emulated. We alter our previous view of the rules of P. We had $(a_n, \xi_n) \in P$ for $1 \leq n \leq p$ which we picture as

$$\xi_n = b_{n,1}\ b_{n,2}\cdots b_{n,|\xi_n|}$$

so we have

$$a_n \rightarrow b_{n,1}\ b_{n,2}\cdots b_{n,|\xi_n|}$$

Now we shall reconstruct the phrases of P, padding with some $e \notin V$ on the left of every phrase. Let

$$\begin{aligned}\eta_n &= e^{J-|\xi_n|}\,\xi_n \\ &= \underbrace{e\ldots\ldots e}_{J-|\xi_n|}\, b_{n,1}\cdots b_{n,|\xi_n|} \\ &= c_{n,1}\ c_{n,2}\cdots c_{n,J}\end{aligned}$$

Thus, we now picture a new set of rules $P' = \{(a_n, \eta_n) \mid (a_n, \xi_n) \in P\}$. Then the following three Boolean matrices are calculated for G.

$$R_{x,n} = 1 \iff a_n\ \bar{\beta}\ x$$

for $x \in V_T \cup \{\dashv\}$; $1 \le n \le p$.

$${}_jL_{x,n} = 1 \iff x\ \alpha\ \lambda^* a_n \ \underline{\text{and}}\ j = J - |\xi_n|$$

for $x \in V \cup \{\vdash\}$; $1 \le n \le p$; $0 \le j \le J$.

$${}_jM_{x,n} = 1 \iff c_{n,j} = x$$

for $x \in V$; $1 \le n \le p$; $1 \le j < J$.

R contains information on right compatability; L describes left compatibility and phrase length; M contains information for matching the phrase. In the following x_{i+1} will refer to the next character on the input stream which is inspected but not removed from that stream.

ALGORITHM 1 uses a stack indexed by i to store matrices $S^i_{j,n}$, where $1 \le j \le J$ and $1 \le n \le p$, defined at steps 1 or 5. The algorithm follows.

1) Set $i := 0$;

$x_0 := \vdash$ the left end marker;

$S^0_{j,n} := {}_jL_{\vdash,n}$ for $1 \le j < J$ and $1 \le n \le p$;

$T_n := {}_JL_{\vdash,n} \wedge R_{x_1,n}$ for $1 \le n \le p$.

If any $T_n = 1$ go to 7.

Note: These elements of L are referenced nowhere else. Only ${}_oL_{\vdash,n}$ might be repeatedly accessed, at step 5. So space can be saved if these unused elements of L indexed by $\vdash$ are not retained after this initialization.

2) Set $i := i + 1$, stacking S^{i-1} (and x_{i-1} if desired).

Let x_i be the next character on the input stream, which is removed from that stream. If $x_i = \dashv$ (right end marker), quit with (hopefully) $i = 2$ and $x_1 = s$ which indicate a successful parse.

3) Compute for $1 \le n \le p$

$$T_n := S^{i-1}_{J-1,n} \wedge {}_JM_{x_i,n} \wedge R_{x_{i+1},n};$$

If, for all n, $T_n = 0$ go to 5.

4) Let n be such that $T_n = 1$. We have identified a parse of

$$a_n \rightarrow x_{1-|\xi_n|+i} \cdots x_i = \xi_n \neq \varepsilon .$$

Perform any desired action associated with that parse.

Set $i := i + 1 - |\xi_n|$;

$x_i := a_n$;

Go to 3.

5) Compute for $1 \le n \le p$ and $2 \le j \le J-1$

$$Q_n := S^i_{1,n} := {}_0L_{x_{i-1},n} \wedge {}_1M_{x_i,n} \vee {}_1L_{x_i,n};$$

$$S^i_{j,n} := S^{i-1}_{j-1,n} \wedge {}_jM_{x_i,n} \vee {}_jL_{x_i,n};$$

$$Q_n := Q_n \vee S^i_{j,n}.$$

If all $Q_n = 0$ an error has been found, unless $i = 1$, $x_1 = s$, and $x_2 = \dashv$ in which case quit successfully.

6) For $1 \le n \le p$ set

$$T_n := {}_JL_{x_i,n} \wedge R_{x_{i+1},n}$$

and if all $T_n = 0$ go to 2.

7) Let n be such that $T_n = 1$. We have identified a parse of $a_n \rightarrow \varepsilon$ between x_i and x_{i+1}.

Perform any desired action associated with that parse.

Set $i := i + 1$;

$x_i := a_n$.

Go to 3.

In the pathologies when $J=0$ or $J=1$, the computations at steps 3 and 5 may have undefined operands and are taken to be 0.

In order to understand the algorithm and its use of the stack, it is only necessary to interpret each bit in an S matrix:

$S^i_{j,n} = 1$ if and only if $|\xi_n| \ge J - j$

and $\qquad x_{i-|\xi_n|+J-j} \; \alpha\lambda^* a_n$

and the two strings

$x_{i+1-|\xi_n|+J-j} \; x_{i+2-|\xi_n|+J-j} \cdots x_i$

$b_{n,1} \; b_{n,2} \cdots b_{n,|\xi_n|-J+j}$

are identical.

It is possible that these two strings are identical and empty ($|\xi_n| = J - j$), in which case $S^i_{j,n}$ indicates left compatability and that a match may just now begin. If all of $S^i = 0$ for any i, then all $Q_n = 0$ at step 5. This situation is an error because no future phrase match could ever include the current x_i; so the parse tree can never be completed. The algorithm will quit at step 2 when $s\lambda^+ s$ and at step 5 otherwise, where s is the axiom.

In the particular case when G is ROR several simplifications occur. The relation $\bar{\beta}$ reduces to β and steps 6 and 7 are replaced by "Go to 2." Obvious simplifications appear in step 1. These eliminate the need for ${}_J L_{x,n}$ for $x \in V \cup \{\vdash\}$, $1 \le n \le p$.

At steps 4 and 7 at most one of the T_n can be non-zero. Hence we get a unique parse tree. However, we can make the algorithm more efficient (as Domelki described) by adding more entries into M and sacrificing the uniqueness of n at step 4.

Consider the strict partial ordering on V_N induced by the $\xrightarrow{+}$ relation:

-transitive	(clearly)
-anti-symmetric	(if $b \xrightarrow{+} a$ and $a \xrightarrow{+} b$ then the reduced grammar is ambiguous and hence not GOR)
-irreflexive	(similarly)

Now define

$$ {}_j\overline{M}_{x,n} = {}_jM_{x,n} \vee \bigvee_{y \in V_N} ({}_jM_{y,n} \wedge y \stackrel{+}{\rightarrow} x) $$

$$ = {}_{,i}M_{x,n} \vee \bigvee_{y \in V_N} ({}_j\overline{M}_{y,n} \wedge y \rightarrow x) $$

If we use $\overline{M}$ in place of M in the algorithm, there may be more than one value of n such that $T_n = 1$ at step 4. It is still true that each choice of $n \ni T_n = 1$ identifies a valid parse, but the most powerful parse is chosen by selecting n such that

$$ |\xi_n| > 1 $$

or, if not possible, such that

$$ a_n \stackrel{*}{\rightarrow} a_m \text{ for all } T_m = 1 $$

It can be seen that if any other value was chosen the algorithm would loop through steps 4 and 3 until the described value of n was forced.

We, therefore, define a strict partial ordering on P which we shall call γ:

$$ (a_n, \xi_n)\ \gamma\ (a_m, \xi_m) \iff |\xi_n| > |\xi_m| \text{ or } |\xi_n| = |\xi_m| \text{ and } a_n \stackrel{+}{\rightarrow} a_m $$

Then we pick the most powerful rule to parse on the basis of the extension of γ to a total ordering.

IV. IMPROVEMENT

For comparison it is necessary to state Lynch's version (2) of the algorithm in a manner parallel to that above.

He reconstructed the phases of P, "padding" with some $e \notin V$ on the right of every phrase. Let $\zeta_n = \xi_n \; e^{J-|\xi_n|} = d_{n,1} \; d_{n,2} \cdots d_{n,J}$. So $d_{n,j} = b_{n,j}$ for $1 \le j \le |\xi_n|$ and $d_{n,j} = e$ for $|\xi_n| < j \le J$. Then

$$P'' = \{(a_n, \zeta_n) \mid (a_n, \xi_n) \in P\}.$$

We then construct three matrices

$$R_{x,n} = 1 \iff a_n \bar{\beta} x$$

for $x \in V_T \cup \{\dashv\}$; $1 \le n \le p$ as before.

$$L_{x,n} = 1 \iff x \alpha \lambda^* \alpha_n$$

for $x \in V \cup \{\vdash\}$; $1 \le n \le p$.

$$_j M_{x,n} = 1 \iff d_{n,j} = x$$

for $x \in V \cup \{e\}$; $1 \le n \le p$; $1 \le j \le J$.

Then ALGORITHM 2 proceeds

1) Set $S^0_{j,n} := 0$ for $1 \le j \le J$ and $1 \le n \le p$;

$i := 0$;

$x_i := \vdash$.

2) Set $i := i + 1$.

Let x_i be the next character on the input stream which we now remove from that stream. If $x_i = \dashv$ (right endmarker) quit.

3) Let x_{i+1} refer to the following character on the input stream, which we peek at but do not remove.

Compute for $1 \le n \le p$ and $2 \le j < J$

$$S^i_{1,n} := L_{x_{i-1},n} \wedge ({}_1M_{e,n} \vee {}_1M_{x_i,n});$$

$$S^i_{j,n} := S^{i-1}_{j-1,n} \wedge {}_jM_{x_i,n} \vee S^i_{j-1,n} \wedge {}_jM_{e,n};$$

$$T_n := \left[S^{i-1}_{J-1,n} \wedge {}_jM_{x_i,n} \vee S^i_{J-1,n} \wedge {}_JM_{e,n}\right] \wedge R_{x_{i+1},n};$$

If all $T_n = 0$ go to 2.

4) Let n be such that $T_n = 1$. Perform any action associated with a parse of $a_n \rightarrow \xi_n$.

Set $i := i + 1 - |\xi_n|$;

$x_i := a_n$;

Go to 3.

The same comments about ROR grammars and the $\overline{M}$ matrix and γ relation applies to this form of the algorithm--indeed, the latter antedate it.

The power of the improved ALGORITHM 1 lies in viewing the padding of e as on the left of the phrases. This results in phrase matches for all rules ending at the S^i_{J*} row so that no cascading to collect these indicators is necessary.

(This cascade is effected by the $S^{i}_{j-1,k} \wedge {}_{j}M_{e,k}$ term in the second formula of step 3 of ALGORITHM 2.) Since the core of the algorithm is computing the S matrices, this improvement results in two significant time savings:

1) Since T can be computed for x_i before S^i, we save the computation of one S matrix (the last) for every non-ε phrase parsed. In a language like ALGOL where the average phrase parsed is of at most length 4, we have a $\frac{3}{4}$ factor improvement or better.

2) In each computation within step 5 we save one logical operation (and indexing associated with the operand). Depending on the computer we can save perhaps a factor of $\frac{2}{3}$ for every computation of S.

Hence we can expect around a 50% time improvement, depending on the grammar and machine.

However, there is a cost in space. While M is reduced by the elimination of provision for e, L is expanded to be as large as M. This virtually doubles the space requirements for constant tables. The stack of S matrices is one matrix shorter.

The entire value of the algorithm and its predecessors is that each matrix entry in L, R, M, T or S is only one bit, and that in a binary computer word we have space for many values as we let n wander $1 \leq n \leq p$. The computations here are all done in parallel with respect to n, so that this packing takes advantage of the parallel logical operations (and and or) on full computer words. Further, if we order P as suggested by γ, the choice of the most powerful rule to parse becomes that of finding the high order one-bit in the few words representing the T vector.

V. EXTENSION TO SLR(1)

If we sacrifice this parallelism with respect to n, the algorithm can be easily modified to handle the weaker Simple LR(1) (denoted SLR(1)) grammars as formally defined by DeRemer (4). For $k > 0$, $LR(0) \subsetneq SLR(k) \subsetneq LR(k)$, where these are the three classes of grammars satisfying Knuth's LR(0), DeRemer's SLR(k), and Knuth's LR(k) conditions respectively (7). In terms of the parsing discussed here, in SLR(k) the left compatibility check (effected by the L matrix in ALGORITHM 1) is replaced by a verification that the string already scanned is a legitimate prefix for some sentence in the language. This amounts to a test for the LR(0) property, which is accomplished in the modification below by abandoning the α relation as part of the phrase-match starter in favor of a stronger "enabler" based on the string already scanned. The independent right compatability check of k characters beyond the end of a phrase match is precisely that effected by the R matrix of ALGORITHM 1 when $k = 1$.

The following five Boolean matrices are claculated from G.

$$R_{x,n} = 1 \iff a_n \bar{\beta} x$$

for $x \in V_T \cup \{\dashv\}$; $1 \le n \le p$.

$$_jM_{x,n} = 1 \iff c_{n,j} = x$$

for $x \in V$; $1 \le n \le p$; $1 \le j \le J$.

$$L_{x,n} = 1 \iff x \lambda^* a_n$$

for $x \in V_N$; $1 \le n \le p$.

$$H_{j,n} = 1 \iff j = J - |\xi_n|$$

for $0 \le j \le J$; $1 \le n \le p$.

$$N_{j,n} = 1 \iff c_{n,j+1} \in V_N$$

for $0 \le j < J$; $1 \le n \le p$.

R and M, identical to those of ALGORITHM 1, contain information on right compatibility and matching. L has been redefined to describe left derivability; H gives the lengths of phrases indicating the height of the row in S where a match should begin; N gives information about the next character being sought for matching.

The non-zero entries in the S matrices will have the same meaning as in ALGORITHM 1 except that every entry indicates a match has started. The E^i_n vector will enable ξ_n to begin matching on x_{i+1}, rather that allowing an entry in $S^i_{j-|\xi_n|,n}$ to indicate a match may begin with x_{i+1}.

ALGORITHM 3, using a stack indexed by i for the S^i and E^i matrices proceeds as follows.

1) Set i:= 0;

$E^0_n := L_{s,n}$ for $1 \le n \le p$ where s is the axiom of G;

$S^0_{j,n} := 0$ for $1 \le j \le J$, $1 \le n \le p$;

Go to 6.

2) Set i:= i+1 stacking S^{i-1} and E^{i-1}.

If x_i, the next character from the input stream, is $\dashv$ (right end marker) then quit.

3) Compute for $1 \le n \le p$

$$T_n := (E_n^{i-1} \wedge H_{J-1,n} \vee S_{j-1,n}^{i-1}) \wedge {}_jM_{x_i,n};$$

If, for all n, $T_n = 0$ go to 5.

4) Same as step 4 of ALGORITHM 1.

5) Compute for $1 \le n \le p$ and $2 \le j \le J-1$

$$E_n^i := 0;$$

$$Q_n := S_{1,n}^i := E_n^{i-1} \wedge H_{o,n} \wedge {}_1M_{x_i,n};$$

$$S_{j,n}^i := (E_n^{i-1} \wedge H_{j-1,n} \vee S_{j-1,n}^{i-1}) \wedge {}_jM_{x_i,n};$$

$$Q_n := Q_n \vee S_{j,n}^i.$$

Whenever $S_{j,n}^i \wedge N_{j,n} = 1$ then for $1 \le m \le p$ set $E_m^i := E_m^i \vee L_{c_{n,j+1},m}$.

If all $Q_n = 0$ an error has occurred, unless $x_{i+1} = \dashv$, $i = 1$ and $x_1 = s$ in which case quit successfully.

6) For $1 \le n \le p$ set

$$T_n := E_n^i \wedge H_{J,n} \wedge R_{x_{i+1},n}$$

and if all $T_n = 0$ go to 2.

7) Same as step 7 of ALGORITHM 1.

The same comments about ROR, $\overline{M}$, and γ described for ALGORITHM 1 apply to ALGORITHM 3 as well.

The parallelism disappears because of the information required to compute the E vector at step 5. If, as suggested, we have designed the matrices so that

traversing a row, $1 \leq n \leq p$ or $1 \leq m \leq p$, traverses the bits of only a few computer words, then finding $c_{n,j+1}$ requires precisely locating a bit within those words. In ALGORITHM 1 it was necessary to locate a '1' bit (the high-order one if $\overline{M}$ was used), but only after it was certain a parse had to occur. Because all machines can trivially test an entire word to determine existence of '1' bits (often called a "jump on zero" instruction) that algorithm can run through all steps but 4 and 7, the parsing steps, without bothering where '1' bits exist. However, ALGORITHM 3 must look up the precise position of many '1' bits in each S matrix. This intermediate work will slow it considerably, but since there is a much stronger left test (for LR(0) property) we can hope that each S^i matrix will remain sparse, certainly much sparser than the S^i of ALGORITHM 1 if it were run instead.

At least there is some space saving as L returns to about its size in ALGORITHM 2 and N and H are not too cumbersome. For stack control we can view E^i as but another row of S^i.

DeRemer defined SLR(k) in terms of the ability of his parsing techniques. Therefore it is not startling to suggest, in light of the closing statement of section II above, that ALGORITHM 3 is sufficient to parse a grammar if and only if it is SLR(1).

APPENDIX

An example of a GOR grammar and the calculations necessary to prepare for ALGORITHM 1 with a sample parse traced.

$$G_1 = (\{+,-,*,(,),z\} \ , \ \{s,t,f\} \ , \ s \ , \ P)$$

where P is the set of rules

0 $\quad \vdash s \dashv$

1 $\{ \quad [s \ , \ s + t] \ ,$

2 $\quad [s \ , \ s - t] \ ,$

3 $\quad [t \ , \ t * f] \ ,$

4 $\quad [f \ , \ (\ s \ )] \ ,$

5 $\quad [s \ , \ t \ \ ] \ ,$

6 $\quad [t \ , \ f \ \ ] \ ,$

7 $\quad [f \ , \ z \ \ ] \ ,$

8 $\quad [f \ , \ \varepsilon \ \ ] \ \}$

P is listed in the linear order determined by γ. The 0^{th} phrase is included for completeness. For G_1 $p = 8$ and $J = 3$. G_1 is GOR; omitting rule 8 would make it ROR.

In the Boolean matrices below the following convention are adopted: If a matrix is labelled σ, we will represent σ, σ^*, $\bar{\sigma}$, and $\bar{\sigma}^*$ all in one matrix by entering either 0,1,*,=, or # with meanings as below.

table of interpretations

label appearing \ matrix wanted	σ	σ^*	$\bar{\sigma}$	$\bar{\sigma}^*$
0	0	0	0	0
1	1	1	1	1
*	0	1	0	1
=	0	0	1	1
#	0	0	0	1

With this table we can interpret each entry in one of the following tableaus to get any one of the four matrices it may describe.

α	s	t	f	+	-	*	(	)	z	$\dashv$
s	0	0	0	1	1	0	0	1	0	1
t	0	0	0	0	0	1	0	0	0	0
f	0	0	0	0	0	0	0	0	0	0
+	0	1	0	0	0	0	0	0	0	0
-	0	1	0	0	0	0	0	0	0	0
*	0	0	1	0	0	0	0	0	0	0
(	1	0	0	0	0	0	0	=	0	0
)	0	0	0	0	0	0	0	0	0	0
z	0	0	0	0	0	0	0	0	0	0
$\vdash$	1	0	0	0	0	0	0	0	0	=

Matrix of α and $\bar{\alpha}$ for G_1

λ	s	t	f	+	-	*	(	)	z	$\vdash$	$\dashv$
s	1	1	*	=	=	#	*	0	*	0	0
t	0	1	1	0	0	=	*	0	*	0	0
f	0	0	*	0	0	0	1	0	1	0	0
+	0	0	0	*	0	0	0	0	0	0	0
-	0	0	0	0	*	0	0	0	0	0	0
*	0	0	0	0	0	*	0	0	0	0	0
(	0	0	0	0	0	0	*	0	0	0	0
)	0	0	0	0	0	0	0	*	0	0	0
z	0	0	0	0	0	0	0	0	*	0	0
$\vdash$	0	0	0	0	0	0	0	0	0	*	0
$\dashv$	0	0	0	0	0	0	0	0	0	0	*

Matrix of λ, $\bar{\lambda}$, λ^*, and $\bar{\lambda}^*$ for G_1

ρ	s	t	f	+	-	*	(	)	z	$\vdash$	$\dashv$
s	*	0	0	0	0	0	0	0	0	0	0
t	1	*	0	0	0	0	0	0	0	0	0
f	*	1	*	0	0	0	0	0	0	0	0
+	=	0	0	*	0	0	0	0	0	0	0
-	=	0	0	0	*	0	0	0	0	0	0
*	#	=	0	0	0	*	0	0	0	0	0
(	0	0	0	0	0	0	*	0	0	0	0
)	*	*	1	0	0	0	0	*	0	0	0
z	*	*	1	0	0	0	0	0	*	0	0
$\vdash$	0	0	0	0	0	0	0	0	0	*	0
$\dashv$	0	0	0	0	0	0	0	0	0	0	*

Matrix of ρ, $\bar{\rho}$, ρ^*, and $\bar{\rho}^*$ for G_1

β	s	t	f	+	-	*	(	)	z	⊣
s	0	0	0	1	1	0	0	1	0	1
t	0	0	0	1	1	1	0	1	0	1
f	0	0	0	1	1	1	0	1	0	1
+	0	1	1	=	=	=	1	=	1	=
-	0	1	1	=	=	=	1	=	1	=
*	0	0	1	=	=	=	1	=	1	=
(	1	1	1	=	=	=	1	=	1	0
)	0	0	0	1	1	1	0	1	0	1
z	0	0	0	1	1	1	0	1	0	1
⊢	1	1	1	=	=	=	1	0	1	=

Matrix of $\dot{\beta}$ and $\bar{\beta}$ for G_1

The (resolved) overlaps of G_1:

Triple	meets condition	resolvable on
(1,5,0)	3a	left
(2,5,0)	3a	left
(3,6,0)	3a	left
(1,8,3)	1b	right
(1,8,2)	1a	left
(1,8,1)	1a	right
(2,8,3)	1b	right
(2,8,2)	1a	left
(2,8,1)	1a	right
(3,8,3)	1b	right
(3,8,2)	1a	left
(3,8,1)	1a	right
(4,8,3)	1b	right
(4,8,2)	1a	right
(4,8,1)	1a	left
(5,8,1)	1b	right
(6,8,1)	1b	right
(7,8,1)	1b	right

$R_{x,n}$ x \ n	1	2	3	4	5	6	7	8
+	1	1	1	1	1	1	1	1
-	1	1	1	1	1	1	1	1
*	0	0	1	1	0	1	1	1
(	0	0	0	0	0	0	0	0
)	1	1	1	1	1	1	1	1
z	0	0	0	0	0	0	0	0
⊣	1	1	1	1	1	1	1	1

R matrix for G_1 derived from $\bar{\beta}$

${}_jL_{x,n}$	j = 0								j = 1								j = 2								j = 3							
x \ n	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
+	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1
-	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1
*	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
(	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1
)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
⊢	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1

Note: ${}_1L_{x,k} = 0$ since G_1 has no rules of length 2.

$x \in V_N \Rightarrow {}_jL_{x,n} = 0$ since G_1 is an operator grammar.

L matrix for G_1

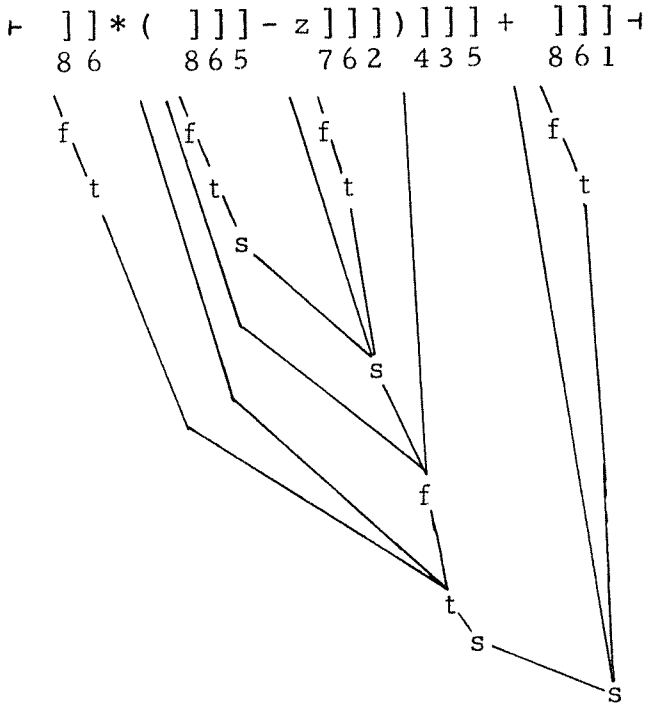
${}_jM_{x,n}$	j = 1	j = 2	j = 3
x \ n	1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8
s	1 1 0 0 0 0 0 0	0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0
t	= = 1 0 0 0 0 0	0 0 0 = 0 0 0 0	1 1 0 0 1 0 0 0
f	= = = 0 0 0 0 0	0 0 0 = 0 0 0 0	= = 1 0 = 1 0 0
+	0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
-	0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0	0 0 0 0 0 0 0 0
*	0 0 0 0 0 0 0 0	0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0
(	0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0
z	= = = 0 0 0 0 0	0 0 0 = 0 0 0 0	= = 0 0 = = 1 0

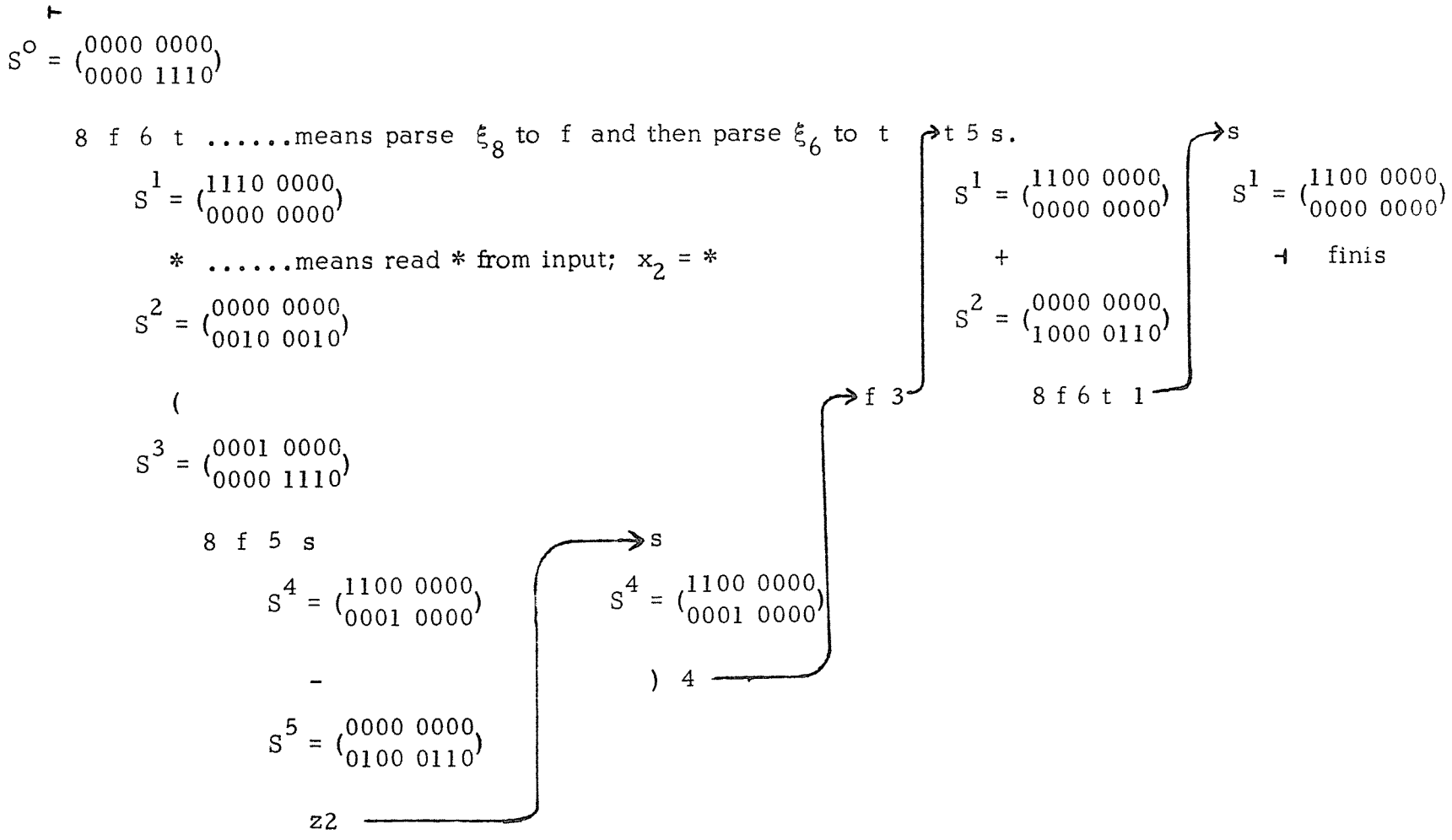
M and $\overline{M}$ matrix for G_1. $\overline{M}$ is used in the example.

Example sentence to be parsed by ALGORITHM 1 using $\overline{M}$ and G_1:

$$\vdash * (- z) + \dashv .$$

The correct parse, and derivation tree appear below.





Parse of the sentence *(-z)+ according to G_1 by ALGORITHM 1.

Bibliography

1) Domelki, B. Algorithms for the recognition of properties of sequences of symbols. Zhurnal Vychislitel'noi Matematiki i matematicheskoi Fiziki 5,1 (1965), 77-97, translation: USSR Computational & Mathematical Physics 5,1, Pergammon Press, Oxford, 1967, 101-130.

2) Lynch, W. C. A high speed parsing algorithm for ICOR grammars. Andrew Jennings Computer Center, Report No. 1097, Case Western Reserve University, Cleveland, 1968.

3) Lynch, W. C. Ambiguities in Backus Normal Form Languages, Ph.D. Thesis, University of Wisconsin, Madison, 1963.

4) DeRemer, Franklin Lewis. Practical translators for LR(k) languages. Project MAC TR-65, Massachusetts Institute of Technology, Cambridge, 1969, (U.S. Dept. of Commerce Clearinghouse, AD 699-501).

5) Colmerauer, Alain. Total precedence relations. J.ACM 17, 1 (January, 1970), 14-30.

6) Hopcroft, J. E. and Ullman, J. D. Formal Languages and their Relation to Automata, Addison Wesley, Reading, Mass., 1969, 62-63.

7) Knuth, Donald E. On the translation of languages from left to right. Information and Control 8, 6 (December, 1965), 607-639.