CONFERENCE RECORD

SEVENTH ASILOMAR CONFERENCE ON CIRCUITS, SYSTEMS, AND COMPUTERS

EDITED BY

Sydney R. Parker
Naval Postgraduate School
Monterey, California

PAPERS PRESENTED

TUESDAY THROUGH THURSDAY NOVEMBER 27-29, 1973
ASILOMAR HOTEL & CONFERENCE GROUNDS – PACIFIC GROVE, CALIFORNIA

Sponsored by

NAVAL POSTGRADUATE SCHOOL
Monterey, California

UNIVERSITY OF SANTA CLARA
Santa Clara, California

With The Participation Of The

UNIVERSITY OF CALIFORNIA
Davis, California

UNIVERSITY OF CALIFORNIA
Los Angeles, California

UNIVERSITY OF CALIFORNIA
Santa Barbara, California

CALIFORNIA STATE UNIVERSITY
Sacramento, California

STANFORD UNIVERSITY
Stanford, California

UNIVERSITY OF SOUTHERN CALIFORNIA
Los Angeles, California

UNIVERSITY OF ARIZONA
Tucson, Arizona

WASHINGTON STATE UNIVERSITY
Pullman, Washington

In Cooperation With

IEEE SAN FRANCISCO SECTION, IEEE CIRCUITS AND SYSTEMS SOCIETY, AND IEEE CONTROL SYSTEM SOCIETY

CONFERENCE COMMITTEE

Chairman:	J. Choma, Jr., Hewlett-Packard Company
Technical Program:	T.E. Fanshier, Lockheed Missiles and Space Company
	T.J. Higgins, University of Wisconsin
	L.P. Huelsman, University of Arizona
	L.P. McNamee, University of California, Los Angeles
	T. Mills, National Semiconductor Corporation
	J.G. Simes, California State University, Sacramento
	A. Willson, University of California, Los Angeles
Publication/Finance:	S.R. Parker, Naval Postgraduate School
Publicity:	S.G. Chan, Naval Postgraduate School

CONFERENCE RECORD

SEVENTH ASILOMAR CONFERENCE ON CIRCUITS, SYSTEMS, AND COMPUTERS

EDITED BY

Sydney R. Parker
Naval Postgraduate School
Monterey, California

PAPERS PRESENTED

TUESDAY THROUGH THURSDAY NOVEMBER 27-29, 1973
ASILOMAR HOTEL & CONFERENCE GROUNDS – PACIFIC GROVE, CALIFORNIA

Sponsored by

NAVAL POSTGRADUATE SCHOOL
Monterey, California

UNIVERSITY OF SANTA CLARA
Santa Clara, California

With The Participation Of The

UNIVERSITY OF CALIFORNIA
Davis, California

UNIVERSITY OF CALIFORNIA
Los Angeles, California

UNIVERSITY OF CALIFORNIA
Santa Barbara, California

CALIFORNIA STATE UNIVERSITY
Sacramento, California

STANFORD UNIVERSITY
Stanford, California

UNIVERSITY OF SOUTHERN CALIFORNIA
Los Angeles, California

UNIVERSITY OF ARIZONA
Tucson, Arizona

WASHINGTON STATE UNIVERSITY
Pullman, Washington

In Cooperation With

IEEE SAN FRANCISCO SECTION, IEEE CIRCUITS AND SYSTEMS SOCIETY, AND IEEE CONTROL SYSTEM SOCIETY

ON THE REDUCTION OF THE SIZE OF TRANSITION MATRICES

L.F. de Almeida Cunha
and
S.R.P. Teixeira

Computer Science Department
Pontifícia Universidade Católica do Rio de Janeiro

Abstract

Although the general context free grammar has shown to be a suitable model for programming language, it has not been used in practice due to time and memory space limitations. More restricted models which guarantee parsing time proportional to n (where n is the length of the input string) have been used instead. Among these schemes, the use of transition matrices has proved to be remarkably efficient. The great speed of this method lies in the fact that for each table reference it does, it makes a syntatic reduction on the sentential form, no further searching being necessary.
The transition matrix is accessed at each step of compilation given the element at the top of the stack and the next element on the input string. Some entries in the matrix specify reductions to be made while others specify error conditions.
The main disadvantage of the method is the large size of the transition matrices for practical programming languages.
This paper presents a method to partition the original grammar into several grammars so that several transition matrices are used instead of just one. Sufficient conditions are given to allow the processor to switch matrices as it is parsing a sentence of the original grammar. This method has all the advantages of the transition matrices technique (mainly speed), while sensibly reducing the memory space required (the most serious drawback of the original method).
Besides, for some grammars and partitions the new method is more powerful than the original method, as shown.
The method presented in this paper is being used in the parser of a fast resident PL/I compiler for the IBM 370/165, being developed at Pontificia Universidade Católica do Rio de Janeiro. It has shown great speed and moderate memory requirements.

1. INTRODUCTION

The use of formal syntax allows for the algorithmic construction of mechanical analyzers (or recognizers) for a significant class of languages. Althought some available algorithms encompass a large class of these languages, their practical use has been restricted to somewhat particular cases. This restriction has been imposed by storage medium and program execution time limitations in practical compilers. For this reason, although the general context free has shown to be a suitable model for programming language, it has not been used in practice.
The most efficient algorithm know to date for parsing a general context free languages[1], guarantees time proportional to at most n^2, where n is the length of the string being parsed (in the case of unambignous grammars).
In practice more restricted models which guarantee time proportional to n have been used. Among these schemes, the use of transition matrices[2] has proved to be remarkably efficient. The great speed of this method lies in the fact that for each table reference it does, it makes a syntax reduction on the input string, no further searching being necessary. Besides, it allows for good error detection too. These properties are consequence of the very nature of a transition matrix. The matrix is accessed at each step of compilation given the element at the top of the stack and the next symbol on the input string. Some entries in the matrix specify reductions to be made. Others specify error conditions.
The class of languages parsable by processors using transition matrices is a subclass of (1,1) bounded

context languages[3]. The outline of an algorithm to test whether a grammar generates a language which belongs to this subclass is given in [2]. The main disadvantage of the method is the large size of the transition matrices for practical programming languages. This paper presents a method to partition the original grammar into several grammars so that several transition matrices are used intead of a single one. This may result in a great reduction of the total storage area used, depending on the partition chosen. The present method also increases the power of the original method.

2. BASIC DEFINITIONS

An alphabet is any finite nonempty set. For the alphabet V, V^* is the set of all strings of finite-length (words) over V, including the null word Λ. V^+ is defined as $V - \{\Lambda\}$.

If X and Y are sets of words, $X.Y = \{\alpha\beta / \alpha \in X, \beta \in Y\}$ where $\alpha\beta$ is the concatenation of α and β, $X^0 = \{\Lambda\}$ and $X^{i+1} = X^i.X$, for $i = 0,1,2,\ldots$

A context free grammar is a 4-tuple $G=(V_N,V_T,S,P)$ where:

(i) V_N is the alphabet of nonterminal symbols

(ii) V_T is the alphabet of terminal symbols, $V_N \cap V_T = \phi$

(iii) $S \in V_N$ is the start symbol of G.

(iv) P is a set of productions of the form $A\to\alpha$, $A \in V_N$, $\alpha \in V^+$, where $V=V_N \cup V_T$. $A \to \alpha$ is an A-production.

Unless otherwise specified, uppercase Latin letters (A,B,C,...) are used for nonterminal symbols, lower case Latin letters at the beginning of the alphabet (a,b,c,...) for terminal symbols, lower case Latin letters at the end of the alphabet (t,u,v,...) for words in V_T^*, lower case Greek letters $(\alpha,\beta,\gamma,\ldots)$ for words in V^*.

For $\alpha,\beta \in V^+$, we say $\alpha \underset{G}{\Longrightarrow} \beta$, or simply $\alpha \Longrightarrow \beta$ when G is understood, if $\alpha = \alpha_1 A \alpha_2$, $\beta=\alpha_1 \gamma \alpha_2$, $\alpha_1,\alpha_2 \in V^*$ $A \to \alpha \in P$. $\alpha \underset{R}{\Longrightarrow} \beta$ if $\alpha \underset{R}{\Longrightarrow} \beta$ as above and $\alpha_2 \in V_T^*$.

If $\alpha_1 \Longrightarrow \alpha_2 \Longrightarrow \ldots \Longrightarrow \alpha_n$, $n > 1$, we say that $\alpha_1 \overset{+}{\Longrightarrow} \alpha_n$, and that $(\alpha_1,\alpha_2,\ldots,\alpha_n)$ is a derivation of α_n from α_1. Similary if $\alpha_1 \underset{R}{\Longrightarrow} \alpha_2 \underset{R}{\Longrightarrow} \ldots \underset{R}{\Longrightarrow} \alpha_n$, $n > 1$, then $\alpha_1 \overset{+}{\underset{R}{\Longrightarrow}} \alpha_n$, and $(\alpha_1,\alpha_2,\ldots,{}_n)$ is a canonical derivation of α_n from α_1. $(\alpha_n,\ldots,\alpha_2,\alpha_1)$ is said to be a canonical parse of α_n to α_1. When α_1 is not mentioned, it is assumed that $\alpha_1 = S$.

Define $\alpha_1 \overset{*}{\Longrightarrow} \alpha_n$ if $\alpha_1 \overset{+}{\Longrightarrow} \alpha_n$ or $\alpha_1 = \alpha_n$. Analogously we define $\alpha_1 \overset{*}{\underset{R}{\Longrightarrow}} \alpha_n$.

If $S \overset{+}{\Longrightarrow} \alpha$ ($S \overset{+}{\underset{R}{\Longrightarrow}} \alpha$), α is a (canonical) sentential form. If $\alpha \in V_T^*$ then α is a sentence.

G is said to be reduced if for each $A \in V_N$, each A-production $A \to \gamma \in P$, $S \overset{*}{\Longrightarrow} \alpha_1 A \alpha_2 \Longrightarrow \alpha_1 \gamma \alpha_2 \overset{*}{\Longrightarrow} x$ for $\alpha_1,\alpha_2 \in V^*$, $x \in V_T^*$.

The language generated by G, is: $L(G)=\{x/S \Longrightarrow, x \in V_T^*\}$.

For each context free grammar G there is a context free grammar G' (called the reduced form of G) such that L(G) = L(G'), and G' is reduced[4].

If $S \overset{*}{\Longrightarrow} \alpha_1 A \alpha_2$, $A \overset{+}{\Longrightarrow} \beta$, then $S \overset{+}{\Longrightarrow} \alpha_1 \beta \alpha_2 = \gamma$ and β is said to be a phrase (an A-phrase) of γ. In this case β is a maximal A-phrase of γ if it is not properly contained in another A-phrase of γ. β is a simple phrase if $A \Rightarrow \beta$.

G in said to be unambiguous iff each sentence has a unique canonical derivation.

If G is unambiguous, γ is a sentential form of G, and β_1 and β_2 are phrases of γ then β_1 and β_2 have no symbol in common or one includes the other[4].

Next, we define three functions:

for each $A \in V_N$

$\text{head}_G(A) = \{a/a \in V_T, \text{there exists } \alpha \in V^*, A \overset{+}{\underset{G}{\Longrightarrow}} a\,\alpha\}$

$\text{suc}_G(A) = \{X/X \in V, \text{ there exist } \alpha_1,\alpha_2 \in V^*,$
$S \overset{+}{\underset{G,R}{\Longrightarrow}} \alpha_1 A X \alpha_2\}$

$\text{pred}_G(A) = \{X/X \in V, \text{ there exist } \alpha_1,\alpha_2 \in V^*,$
$S \overset{+}{\underset{G,R}{\Longrightarrow}} \alpha_1 X A \alpha_2\}$

G is an operator grammar iff no production is of the form $A \to \alpha_1 B C \alpha_2$, $\alpha_1, \alpha_2 \in V^*$, $B, C \in V_N$.

3. THE ORIGINAL PARSING METHOD

It is necessary to repeat some of the important results of the method of transition matrices[2] in order to clarify this paper. The results of this section are proved either directly or indirectly in[2].

A context free grammar $G = (V_N,V_T,S_0,P)$ is an augmented operator grammar (A.O.G) iff:

(i) $V_N = N_* \cup N$, $N_* \cap N = \phi$.
N_* is the set of starred nonterminal symbols (SNTS) and N is the set of unstarred nonterminal symbols (UNTS). Elements of N_* will be denoted as $A^*,B^*,\ldots$, elements of N as: $A,B,\ldots$, and elements of $N_* \cup N$ as $A^-,B^-,\ldots$

(ii) $\perp \in V_T$ is an endmarker.

(iii) $\perp^* \in N_*$, is a starred endmarker

(iv) The only S_0 production in P

$S_0 \rightarrow \perp^* S \perp$, and $\perp^* \rightarrow \perp \varepsilon P$ are the only productions in which S_0, $\perp^*$ or appear.

(v) Each production in P is in one of the forms:

$$A \rightarrow C \tag{3.1}$$
$$A \rightarrow B^* \tag{3.2}$$
$$A \rightarrow B^*C \tag{3.3}$$
$$A \rightarrow a \tag{3.4}$$
$$A^* \rightarrow C\ a \tag{3.5}$$
$$A^* \rightarrow B^*a \tag{3.6}$$
$$A^* \rightarrow B^*C\ a \tag{3.7}$$

Furthemore for each $A^* \varepsilon N_*$ there is at most one A^*-production in P.
The right part α, of an A^*-production $A^* \rightarrow \alpha$ appears as the right part of no other production.

(vi) For each $A \varepsilon N$, $A \xRightarrow[G]{+} A$ does not hold. If $A_0 \xRightarrow[G]{+} A_k$ the sequence $A_0 \xRightarrow[G]{} A_1 \xRightarrow[G]{} \dots$ $\dots \xRightarrow[G]{} A_k$ is unique, $k \geq 1$.

In[5] an algorithm is given to transform any context free grammar into one satisfying (vi) above. In[2] an algorithm is given to transform an operator grammar satisfying (vi) above into an A.O.G., such that the A.O.G. is unambiguous iff the original operator grammar is unambiguous.

<u>Lemma</u> 3.1 - If G is an A.O.G., then:

(a) No sentential form α has the form $\alpha_1 A\ B\ \alpha_2$, $\alpha_1, \alpha_2\ \varepsilon\ V^*$, $A, B\ \varepsilon\ N$.

(b) No sentential form α has the form $\alpha_1 \beta\ \gamma\ \alpha_2$ where β is an A-phrase, γ is a B-phrase. $\alpha_1, \alpha_2\ \varepsilon\ V^*$ $A, B\ \varepsilon\ N$.

(c) If $\alpha_1\ X\ \gamma\ \alpha_2$ is a canonical sentential form, and γ is an A-phrase, $A\ \varepsilon\ N$ $X\ \varepsilon\ V$, then $X\ \varepsilon\ N_*$

(d) If $B\ \varepsilon\ N$, $x\ \varepsilon\ V_T^+$ and $B \xRightarrow[R]{+} x$, the last production used in this derivarion of B to x is of the form $A^* \rightarrow a$.

If α is a sentential form of an A.O.G., a <u>prime phrase</u> of α is a phrase such that:

(a) It contains at least one terminal symbol or one STNS

(b) It contains no other phrase satisfying (a) other than itself.

When parsing a sentential form of an A.O.G.we will detec and reduce a prime phrase according to the shortest derivation that generates it. This derivation takes one of the forms:

$$A \Longrightarrow B^* \tag{3.8}$$
$$A \xRightarrow{+} B^*C \text{ for } A \Longrightarrow B^*D \text{ and } D \xRightarrow{*} C \tag{3.9}$$
$$A^* \Longrightarrow a \tag{3.10}$$
$$A^* \xRightarrow{+} C\ a \text{ for } A^* \Longrightarrow B\ a \text{ and } B \xRightarrow{*} C \tag{3.11}$$
$$A^* \Longrightarrow B^*a \tag{3.12}$$
$$A^* \xRightarrow{+} B^*C\ a \text{ for } A^* \Longrightarrow B^*D\ a \text{ and } D \xRightarrow{*} C \tag{3.13}$$

Note that a phrase C, produced by a derivation of the type $A \xRightarrow{+} C$ is not considered as a separate case. This saves steps during the parse since in practice derivations of this kind do not involve any semantic evaluation.

<u>Theorem 3.1</u> - A canonical sentential form α of and A.O.G. has one of the forms:

$$\text{(a)}\quad B_1^*B_2^* \dots B_\ell^* a_1 a_2 \dots a_m \quad \ell \geq 0,\ m \geq 1 \tag{3.14}$$
$$\text{(b)}\quad B_1^*B_2^* \dots B^*C\ a_1 a_2 \dots a_m \quad \ell \geq 1,\ m \geq 1 \tag{3.15}$$

where $B_1^* = \perp^*$ when $\ell \geq 1$, $a_m = \perp$

Furthemore a leftmost prime phrase of α has one of the forms:

B_ℓ^*, $B_\ell^*\ C$, a_1, $C\ a_1$, $B_\ell^*\ a_1$, $B_\ell^*\ C\ a_1$

When parsing a sentence x of an A.O.G. G we will reach canonical sentential form $\alpha, \alpha \xRightarrow[R]{*} x$. The symbols $B_1^*, B_2^*, \dots, B_\ell^*$ are stored in a pushdown stack. B_ℓ^* is the top of the stack. There is a position called E which stores C in case (3.15). For (3.14) E=empty. $a_1, a_2, \dots, a_m$ are the input symbols to be processed yet. G is said to be <u>parsable by a transition matrix processor</u> if from the top of the stack B_ℓ^*, E and a_1 the leftmost prime phrase β of α is uniquely determined, and the nonterminal A^- such that

$$S_0 \xRightarrow[R]{*} \alpha_1\ A^-\ \alpha_2 \xRightarrow[R]{+} \alpha_1 \beta\ \alpha_2 = \alpha$$

$$\alpha_1\ \varepsilon\ V^*,\quad \alpha_2\ \varepsilon\ V_T^*$$

is also uniquely determined. Note that the derivation of A^- to β which makes

$$\alpha_1\ A^-\ \alpha_2 \xRightarrow[R]{+} \alpha_1\ \beta\ \alpha_2$$

has one the forms (3.8), (3.9),...,(3.13). To complete this parsing step, each of the following three symbols: B_ℓ^*, value of E, a_1, which is part of β is deleted. If $A^- \varepsilon N_*$ then it becomes the new top of the stack. Otherwise A^- will be the new value of E. The process is repeated until E=S_0.

Note that a sentence of an A.O.G. G, is of the form $\perp \beta \perp$ where β is an S-phrase. The parser starts with $\perp^*$ in the stack, and $\beta\perp$ as the input string to be processed.

A transition matrix M is a matrix which has a row assigned to each SNTS B^* , and a column assigned to each terminal a. So, with the top of the stack B^*_ℓ and the next input symbol a_1 , we will access the element $M_{B^*_\ell, A1}$ which specifies for each value of E the unique leftmost prime phrase β, and the unique nonterminal A^- which generates β.

An A.O.G. G is a T-grammar iff Conditions 1 and Condition 2 below hold.

Condition 1 - For each $B^* \varepsilon N_*$, $a \varepsilon V_T$ at most one of the following three statements is true:

(a) There exists $A \to B^* \varepsilon$ P, $a \varepsilon \mathrm{suc}_G(A)$ (3.17)

(b) There exists $A^* \to B^* a \varepsilon P$, (3.18)

(c) There exists $A^* \to a \varepsilon P$, $A^* \varepsilon \mathrm{suc}_G(B^*)$ (3.19)

Furthemore if (a) holds the UNTS A is unique.

Condition 2 - For each $C \varepsilon N$, $B^* \varepsilon N_*$, $a \varepsilon V_T$ at most one of the following three statements is true:

(a) There exists $A \to B^* D \varepsilon P$, $a \varepsilon \mathrm{suc}_G(A)$ (3.20)
$D \overset{*}{\underset{G}{\Longrightarrow}} C$

(b) There exists $A^* \to B^* D$ aεP, $D \overset{*}{\underset{G}{\Longrightarrow}} C$ (3.21)

(c) There exists $A^* \to D$ aεP, $A^* \varepsilon \mathrm{suc}_G(B^*)$ (3.22)
$D \overset{*}{\underset{G}{\Longrightarrow}} C$

Furthemore if (a) holds both A and D are unique.If (b) or (c) hold D is unique.

Before the next Theorem, it is worth to note that the procedure that transforms any context free grammar G into a reduced context free grammar G' , L(G') = L(G), when applied to an A.O.G. G,produces another A.O.G. G'.

Theorem 3.2 - If an A.O.G. G is a T-grammar then it is unambiguous.

Theorem 3.3 - If an A.O.G. G is parsable by a transition matrix processor then its reduced form G' is a T-grammar.

Theorem 3.4 - If an A.O.G. G is a T-grammar then it is parsable by a transition matrix processor.

The method of transition matrices is very fast,its disadvantage is the large size of the transition matrix. In the next section we will see how to reduce the size of the matrix.

4. THE PARTITIONING METHOD

Let $G = (V_N, V_T, S_0, P)$ be a reduced A.O.G. for which we want to construct a transition matrix processor. We will partition G into two grammars and construct transition matrices for both. When parsing a sentence we will detect conditions which will tell us when to switch matrices. This scheme allows for a sensible reduction of the total memory space occupied by the transition matrices without reducing the speed of the method noticeably.

Let $S_0 \to \perp^* S \perp$ be the only S_0- production in P. Let $S_1 \varepsilon V_N$, $S \neq S_0$, $S_1 \neq S$.

$H' = (V'_N, V'_T, S_0, \bar{P}')$ is the reduced form of $(V_N, V_T \cup \{t_1\}, S_0, \bar{P})$ where $t_1 \notin V$

$\bar{P} = \{A \to \bar{\alpha}$ / $A \to \alpha \varepsilon P$, $A \neq S_1$, $\bar{\alpha}$ is obtained from α by replacing each occurrence of S_1 by $t_1\}$

$G = (V_{N0}, V_{T0}, S_0, P_0)$ is the A.O.G.

$(V'_N \cup \{S_1\}, V'_T - \{t_1\}, S_0, P')$ where

$P' = \{A \to \underline{\alpha}$ / $A \alpha \varepsilon \bar{P}'$, $\underline{\alpha}$ is obtained from α by replacing each occurrence of t_1 by $S_1\}$.

Note that G_0 is an A.O.G. but it is not reduced.

$H'_1 = (V'_N, V'_T, S_1, P'_1)$ is the reduced form of

$H_1 = (V_N, V_T, S_1, P)$. $G_1 = (V_{N1}, V_{T1}, X_1, P_1)$ where:

$X_1 \notin V_N, V_{N1} = V_{N1} \cup \{\perp^*, X_1\}$ $V_{T1} = V'_T \cup \{\perp\}$

$P_1 = P'_1 \cup \{X_1 \to \perp^* S_1 , \perp^* \to \perp\}$

There is a complete example in the appendix.

Note that since G_0 and G_1 are A.O.G.'s we will try to construct a transition matrix for each. We will succeed only if both G_0 and G_1 are T- grammars. In case this does not happen the next two lemmas say that G is not a T-grammar.

Lemma 4.1 - If G is a T-grammar then G_0 is a T-grammar.

Proof - The result follows by observing that $P_0 \subseteq P$.

Lemma 4.2 - If G is a T-grammar then G_1 is a T-grammar.

Proof - The proof of this and subsequent Lemmas and Theorems is omitted for lack of space.

The converse of the two previous Lemmas is not true. Both G_0 and G_1 may be T-grammars while G is not. This case is illustrated by the example in the appendix.

We will now proceed to explain the method to detect the conditions to switch tables. Consider the A.O. G. G and assume both G_0 and G_1 are T-grammars.

Let $x = a_0 a_1 \dots a_n a_{n+1}$ be a sentence generated by G. We have $a_0 = a_{n+1} = \perp$.

Let $\beta = a_{i1}\, a_{i1+1} \dots a_{i1+n1-1}$, $n_1 \geq 1$, $i_1 \geq 1$, $i_1 + n_1 - 1 \leq n$ be the leftmost maximal S_1- phrase in x. Since x is a sentence, β is a prime phrase also. It is essential to consider a maximal phrase because we may have a S_1-phrase within another S_1-phrase.

Assume we start a canonical parse of x using a transition matrix processor for G_0. We will be able to proceed correctly until we reach the canonical sentential form $A_1^* \ldots A_j^* a_{i1} \ldots a_{n+1}$ (see Lemma 3.1 (c) and Theorem 3.1), $A_1^* = \perp^*$, $j \geq 1$. Then we will look at the transition matrix for G_0 (denoted as M_0), for the action is the case E = empty, for the pair (A_j^*, a_{i1}). Because a S_1-phrase is about to begin, the correct action in this case should be to reduce the prime phrase a_{i1} (see Lemma 3.1 (d)) to the unique A_{j+1}^*, such that $A_{j+1}^* \to a_{i1} \in P$. One possibility is that column a_{i1} may not exist in M_0(row A_j^* certainly exists in M_0), or in case it does there is no action for (A_j^*, a_{i1}) and E = empty. In this case we add to M_0 for (A_j^*, a_{i1}) and E = empty, the action to reduce a_{i1} to A_{j+1}^* followed by an order to switch (denoted SW) to the transition matrix of G_1. The other possibility is that there is already an action in M_0 for (A_j^*, a_{i1}) and E = empty. In this case there is a conflict and we cannot resolve the ambiguity with the context used with the transition matrix. Consequently our method will not work (see Lemma 4.3).

What was said above for a particular pair (A_j^*, a_{i1}) has to be done for each pair (A^*, a), where $A^* \in \text{pred}_{G0}(S_1)$ and $a \in \text{head}_{G1}(S_1)$. If no conflicts occur we will then obtain a new matrix we will call M_0'.

Now, we want to modify the transition matrix for G_1 (denoted M_1) to allow us to continue the canonical parse correctly until we reach a canonical sentential form as:

(a) $A_1^* \ldots A_j^*\ S_1 a_{i1+n1} \ldots a_{n+1}$ (4.1)

(b) $A_1^* \ldots A_j^*\ B\ a_{i1+n1} \ldots a_{n+1}$ where (4.2)

$$S_1 \xRightarrow[G_1]{+} B$$

We have to consider the case (b) above because productions of the form $A \to C$ (3.1) do not correspond to prime phrases and consequently are not parsed separately. Note that M_1 will not work above because it uses only $\perp$ as a right delimiter to the maximal S_1-phrase, and will not use a_{i1+n1}.

We say that two elements ξ, ϕ of transitions matrices are <u>compatible</u> iff there is no value <u>v</u> of E (including <u>empty</u>) for which the sequence of actions (or orders) in ξ (if any) differs from the sequence of actions in ϕ (if any). In this case, to <u>merge</u> ϕ into ξ will add to ξ all sequences of actions in ϕ which did not exist in ξ. If ξ and ϕ are not compatible, the merging will produce a conflict.

We will now proceed to modify M_1. For each $A^* \varepsilon V_{N1}$ $A^* \neq \perp^*$ the entry in M_1 for $(A^*, \perp)$ is deleted from M_1 (has all actions erased in M_1) and the original entry is merged into the entry for (A^*, a) for each $a \in \text{suc}_{G0}(S_1)$ (note that we may have $a = \perp$). Similarly, for each $a \in V_{T1}$, $a \neq \perp$ the entry in M_1 for $(\perp^*, a)$ is deleted from M_1, and merged into the entry for (A^*, a) for each $A^* \in \text{pred}_{G0}(S_1)$. (note that we may have $A^* = \perp^*$). Finally, the entry in M_1 for $(\perp^*, \perp)$ is deleted, and the original entry with all actions substituted for SW's is merged into the entry for (A^*, a), for each $A^* \in \text{pred}_{G0}(S_1)$ $a \in \text{suc}_{G0}(S_1)$. If no conflicts occur in the above process then the matrix M_1 with all the modifications in called M_1'.

With M_1' (instead of M_1) we will reach the situation (4.1) or (4.2) and then we will execute SW and switch back to M_0' correctly. The return to M_0' is done at the right time since no two maximal S_1-phrases may be adjacent (see Lemma 3.1 (b)). If there is any conflict in the construction of M_1', then our method will not work (see Lemmas 4.4, 4.5, 4.6).

The transition matrix processor starts with M_0' and switches between M_0' and M_1' whenever it executes an order SW. If no action (or order) is found in one of the matrices, then an error has occurred. This modified processor is called a <u>segmented transition matrix processor</u>.

<u>Lemma 4.3</u> - If there exist $A^* \varepsilon \text{pred}_{G0}(S_1)$, $a \in \text{head}_{G1}(S_1)$, and in M_0 for (A^*, a) and E = empty, there is an action which is different from the action to reduce <u>a</u> to the unique B^* such that $B^* \to a \in P$, then G is not a T-grammar.

<u>Lemma 4.4</u> - If there exist $A^* \varepsilon V_{N1}$, $A^* \neq \perp^*$, $a \in \text{suc}_{G0}(S_1)$ such that in M_1 the entry for $(A^*, \perp)$ is not compatible with the entry for (A^*, a) then G is not a T-grammar.

<u>Lemma 4.5</u> - If there exist $a \in V_{T1}$, $a \neq \perp$, $A^* \in \text{pred}_{G0}(S_1)$ such that in T_1 the entry for $(\perp^*, a)$, is not compatible with the entry for

(A^*,a), then G is not a T-grammar.

Lemma 4.6 - If there exist $A^* \varepsilon\ pred_{G0}(S_1)$, $a\ \varepsilon\ suc_{G0}(S_1)$ $A^* \neq \perp^*$, $a \neq \perp$ such that in T_1 the entry for $(\perp^*,\perp)$ with all actions substi - tuted for SW orders is not compatible with the entry for (A^*, a), then G is not a T-grammar.

Theorem 4.1 - If M_0' and M_1' can be constructed without any conflixt then G is unambiguous and the language parsable by the segmented transition matrix processor is precisely L(G).

A algorithm to produce a segmented transiton matrix processor for G using G_0 and G_1 will follow the steps detailed in this paper. First it tests whether G_0 and G_1 are T-grammars in order to construct M_0 and M_1 [2] . Then it constructs M_0' and M_1' , if any conflict occurs the method fails. In this case Lemmas 4.3, 4.4, 4.5 and 4.6 provide conditions to determine whether the original grammar G was a T-grammar. In the appendix an example is given where G is not a T-grammar, but nevertheless there is a segmented transition matrix processor for G. Furthemore, the product of the number of SNTS and the number of terminal symbols of G is 154. But M_0' is only 9 x 7 = 56 and M_1' is 7 x 6 = 42. So, the savings of memory positions as compared to 154 is over 30%.

5. EXTENSION OF THE PARTITION METHOD

The method described in the previous section may be extended when the grammar G is partitioned in more than two parts. In this case we will have the grammars $G_0,G_1,\ldots,G_i$ with matrices $M_0,M_1,\ldots,M_i$ respectively. Analogously to what has been done when i=1, we construct matrices $M_0',M_1',\ldots,M_i'$.The processor starts with matrix M_0' and may switch to M_j', $1 \leq j \leq i$. Then it may switch to M_k' , $1 \leq k \leq i$, $k \neq j$ and so forth. In this general case we will need a pushdown stack to be able to remember the matrix (M_j' above) to return to when we finish processing using M_k' . This was not necessary when i=1.

The results, Lemmas, etc... for this general case are analogous to the case when i=1, and the details will not be given here.

6. CONCLUSIONS

The method presented has all the advantages of the transition matrices technique (mainly speed),while sensibly reducing the memory space requirements (the most serious drawback of the original method [2]). Besides the partition of the original grammar may increase the power of the original method in some cases.

These effects are much similar to these obtained by Karenjack [6] with his method for constructing LR(k) processors.

The problem that remains to be studied is that of the choice of the most convenient partitioning nonterminals, from the point of view of memory space requirements, and number of switches executed when parsing using a partitioned matrix.

The method presented in this paper is being used in the parser for a fast resident PL/I compiler for the IBM 370/165, being developed at Pontificia Universidade Católica do Rio de Janeiro. It has shown great speed and moderate memory requirements.

REFERENCES

1. Earley, J. - An Efficient Context Free Parsing Algorithm. Comm. ACM 13 (Feb. 1970),94-102.
2. Gries, D. - The Use of Transition Matrices in Compiling - Comm. ACM 11 (Jan.1968),26-34.
3. Floyd, R.W. - Bounded Context Syntatic Analysis Comm. ACM 7 (Feb. 1964), 62-67.
4. Hopcroft, J. and Ullman, J. - Formal Languages and their Relation to Automata - Addison Wesley, New York, 1969.
5. Ginsburg, S. - An Introduction to Mathematical Theory of Context - Free Languages. McGraw - Hill Book Company, N.Y., 1966.
6. Korenjack, A.J. - A Practical Method for Constructing LR(k) Processors - Comm. ACM 12 (Nov. 1969), 613-623.