Quasi-sequential Grammars and their Parsing Algorithms

Koichiro OCHIMIZU*, Masaharu MIZUMOTO*, Junichi TOYODA*, Kohkichi TANAKA*

1. INTRODUCTION

A class of quasi-sequential grammars is defined and a parsing algorithm is given. This class of grammars can describe the syntax of FORTRAN IV programming language directly.

When the syntax of FORTRAN IV is represented by the form of the Chomsky's generative grammars, all variables can be ordered, $A_1, A_2, \cdots, A_n$ and all productions can be given either form of $A_i \to A_i \alpha$ where $\alpha \in V^*$ and α contains no A_j with $j \leq i$ or $A_i \to \beta$ where $\beta \in V^+$ and β contains no A_i with $j \leq i$ except for the cases of <primary>→(<arithmetic expression>), <logical primary>→(<logical expression>), <format specification list group>→(<format specification I>) and <Do index part>→(<I/O variables list>, <Do index part I>). Thus this grammar which we call regular-type sequential grammar generates the regular set regarding <primary>, <logical primary>, <format specification list group> and <Do index part> as the temporary terminal symbols.

Quasi-sequential grammars are defined as an extension of regular-type sequential grammars by attaching the productions with self-embedding properties by a special method described in this paper.

The parsing process for these grammars is executed as follows: The set of productions is partitioned into ordered subsets, and scanning-reduction process is repeated k times where k is the number of the ordered subsets. At each step, only one subset in the set of productions is used.

Quasi-sequential grammars are proposed as a model which is applicable to a compiler-compiler for the FORTRAN-type programming languages.

2. REGULAR-TYPE SEQUENTIAL GRAMMARS AND QUASI-SEQUENTIAL GRAMMARS

Definition 1 If a string α includes a string β (i.e. β is a substring of α), we denote it by $\alpha \supset \beta$. If α does not include β, we denote it by $\alpha \not\supset \beta$.

This paper first appeared in Japanese in Joho−Shori (Journal of the Information Processing Society of Japan), Vol. 14, No. 12 (1973), pp. 925∼934.

* Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. Toyonaka, Osaka, Japan 560.

Definition 2[1)] A context-free grammar $G=(V_N,V_T,P,A_1)$ is said to be sequential if the variables in V_N, $A_1,A_2,\cdots,A_n$ can be ordered in such a way that if $A_i\to\alpha$, $\alpha\in V^*$ $(V=V_N\cup V_T)$ is a production in p, then $\alpha\not\ni A_j$ with $j<i$.

Slightly modifying this grammar the concept of the grammar with rank(k) is introduced in the next definition.

Definition 3 Let $G=(V_N,V_T,P,A_1)$ be a context-free grammar such that variables in V_N, $A_1,A_2,\cdots,A_n$, are ordered and each element in P is either

$$A_i\to A_i\alpha\,,\ \alpha\in V^*,\ \alpha\not\ni A_j,\ j\leqq i \qquad\cdots(1)$$

or

$$A_i\to\beta,\beta\in V^+(=V-\{\varepsilon\}),\ \beta\not\ni A_j,\ j\leqq i \qquad\cdots(2)$$

we partition P into k subsets as follows:

(1) The productions whose left-hand side variables are A_i are collected and the set of these productions is denoted by $P(A_i)$.

Therefore P is divided into $P(A_1)$, $P(A_2),\cdots,P(A_n)$. Suppose that $A_i\to C_{ih}\alpha_{ih}$ is the h-th production in $P(A_i)$ and C_{ih} means the leftmost symbol of the right-hand side of the production and α_{ih} is its remaining part.

(2) For $P(A_i)\subset P$, a set S_i is defined as follows:

$$S_i=\{A_p\,|\,A_p\subset\alpha_{ih},\ A_p\in V_N,\ 1\leqq h\leqq m_i\} \qquad\cdots(3)$$

where m_i is the cardinality of $P(A_i)$.

Every $P(A_i)$ is collected by S_i by the algorithm shown in (3).

(3) (i) Set $i=l=m=1$, $S=S_i$, $P(1)=\phi$.

(ii) If $A_{i+1}\in S$ then $P(m)=P(m)\cup P(A_i)$, $m=m+1$, $P(m)=\phi$, $l=l+1$, $S=S_l$ and go to (iii) else $P(m)=P(m)\cup P(A_i)$, $l=l+1$, $S=S\cup S_l$ and go to (iii).

(iii) $i=i+1$ and if $i<n$ then go to (ii) else $P(m)=P(m)\cup P(A_i)$, set $k=m$ and stop.

In this case, grammar G is said to be rank(k).

Proposition 1 $P(i)\cap P(j)=\phi$ for all $i\neq j$. $P(1)\cup P(2)\cup\cdots\cup P(k)=P$.

Definition 4 Suppose a context-free grammar $G=(V_N,V_T,P,A_1)$ with rank(k) and this grammar satisfying the conditions (1)~(4), then G is said to be a regular-type sequential grammar with rank(k).

(1) For any $A_i\in V_N$, there exists $A_i\to\alpha$, $\alpha\in V^+$ and at least for a production, $A_i \overset{*}{\underset{G}{\Rightarrow}} w$, $w\in V_T{}^+$.

(2) There does not exist the pair of productions such that $A_i\to\alpha$, $A_j\to\alpha$, $A_i\neq A_j$, $\alpha\in V^*$.

(3) For any m, consider the $A_j\to\alpha$ in P(m) and Q being a set of productions in P(m) whose right-hand side string α' satisfies $\alpha'\supset\alpha$. Then any sentential form β which is

derived from A_1 by using $(P(1)\cup P(2)\cup\cdots\cup P(m))-(\{A_j\to\alpha\}\cup Q)$ satisfies $\beta \not\Rightarrow \alpha$.

(4) There does not exist the pair of productions in $P(i)$, such that $A_l\to u$, $A_h\to uv$ where u, $v\in V^+$.

Definition 5 A quasi-sequential grammar $G'=(V_N', V_T', P', A_1)$ with rank(k+1) is defined as follows. Suppose $G=(V_N, V_T, P, A_1)$ being a regular-type sequential grammar with rank(k) and satisfying the conditions that there exist $A_{i_1}, A_{i_2}, \cdots, A_{i_r}, A_j\in V_N$ $(i_1\leq j, i_2\leq j, \cdots, i_r\leq j)$ and $A_l \underset{G}{\overset{*}{\not\Rightarrow}}{}^* A_j$ for $A_l\in V_N (l<j)$. A production $A_j\to\alpha_1 A_{i_1}\alpha_2 A_{i_2}\cdots\alpha_r A_{i_r}\alpha_{r+1}\notin P$ to be attached is split into productions such as

$$\begin{cases} A_j\to\alpha_1 A_{q_1}\alpha_2 A_{q_2}\cdots\alpha_r A_{q_r}\alpha_{r+1} \\ A_{q_1}\to\vdash A_{i_1}\dashv,\ A_{q_2}\to\vdash A_{i_2}\dashv,\ \cdots,\ A_{q_r}\to\vdash A_{i_r}\dashv \end{cases} \qquad \cdots(4)$$

where $\alpha_1, \alpha_2, \cdots, \alpha_{r+1}\in V^*$, $A_j\to\alpha_1\alpha_2\cdots\alpha_r\alpha_{r+1}$ satisfies the (1) or (2) in definition 3 and $\vdash, \dashv \notin V_T$. $q_r=q_{r-1}+1, \cdots, q_2=q_1+1$ and q_1 is by one larger than the greatest subscript of variables in V_N. We construct V_N', V_T' and P' as follows.

$V_N'=V_N\ \{A_{q_1}, \cdots, A_{q_r}\}$, $V_T'=V_T\cup\{\vdash, \dashv\}$, $P'=P\cup\{A_j\to\alpha_1 A_{q_1}\alpha_2\cdots\alpha_r A_{q_r}\alpha_{r+1}\}\cup\{A_{q_1}\to\vdash A_{i_1}\dashv, \cdots, A_{q_r}\to\vdash A_{i_r}\dashv\}$. Then $G'=(V_N', V_T', P', A_1)$ is a quasi-sequential grammar with rank(k+1). Remark that $A_j\to\alpha_1 A_{q_1}\alpha_2\cdots\alpha_r A_{q_r}\alpha_{r+1}$ is added to $P(A_j)$ and $\{A_{q_1}\to\vdash A_{i_1}\dashv, \cdots, A_{q_r}\to\vdash A_{i_r}\dashv\}$ is added to $P(0)$.

Proposition 2 A regular-type sequential grammar generates a regular set.

3. PARSING ALGORITHMS FOR QUASI-SEQUENTIAL GRAMMARS

In this section, a parsing algorithm for quasi-sequential grammars is given.

Various sorts of parsers have been proposed for context-free grammars such as top-down analyzer and bottom-up analyzer, LR(k) parser for LR(k) grammars, the parser for precedence grammars, bounded context analyzer. Our parsing algorithm is more simple than any others. For the regular-type sequential grammars, a set of reductions $R=R(1)\cup R(2)\cup\cdots\cup R(k)$ (i.e. the rule which is obtained by changing the right-hand side and left-hand side of a production) are constructed corresponding to the set of productions $P=P(1)\cup P(2)\cup\cdots\cup P(k)$ and, at first, an input terminal string is scanned and reduced from left to right by using only R(k) without looking-ahead symbols and backtracking. The result string from R(k) is analyzed in the same way as in R(k) by using R(k-1). This procedure is repeated by k times. If a parse is finished by using R(1) and the start symbol A_1 is finally obtained, the parse successes and in other cases the parse fails.

---

* $\alpha \underset{G}{\overset{*}{\not\Rightarrow}} \beta$ means α does not derive β by G.

For quasi-sequential grammars with rank(k+1), substrings generated by self-embedding rules are enclosed by ⊢ and ⊣ and then a procedure is related to that part is added. In more detail, the parsing algorithm for a quasi-sequential grammar consists of two phases. One is to recognize the pair of symbols ⊢ and ⊣ , the another is the same as the parsing algorithm for regular-type sequential agrammars.

(1) Suppose a production $A_i \to \vdash A_j \dashv (i \geq j)$ being added to the grammar, the parser scanning a input terminal string from left to right until the first "⊣" is found. If it is found, from that point to the left, a scanning pointer is moved until the first "⊢" is found. The string which is consisted of the pair of ⊢ and ⊣ founded and the string bracketed by them is analyzed by the parsing algorithm for regular-type sequential grammars including ⊢ and ⊣. Then the substring including ⊢ and ⊣ is replaced by A_i.

(2) By the same algorithm as mentioned in (1), the innermost pair of symbols ⊢ and ⊣ is found in the resulting string and the substring including them is reduced. Finally if the sentential form does not include the pairs of symbols ⊢ and ⊣ , then the sentential form is analyzed by the algorithm for the regular-type sequential grammars and the parsing is finished.

Theorem The parsing algorithm for regular-type sequential grammars is deterministic and the parsing algorithm for quasi-sequential grammars is also deterministic.

A construction method of tables used by the parsing algorithm is shown in the following section.

3.1 Transformation of the representation of reductions.

The reductions are transformed as follows: (1) For R(1), if there is a common prefix among the left-hand sides of the reductions, the these reductions with the prefix are combined by bracketing the prefix (i.e. $\alpha_1\beta \to A_i, \alpha_1\beta' \to A_j, \alpha_1\beta'' \to A_k$ become $\alpha_1[\beta \to A_i | \beta' \to A_j | \beta'' \to A_k]$). This algorithm is repeatedly executed until every common prefixes are united. (2) The same algorithm as (1) is executed to R(2),R(3),$\cdots$,R(k).

Example 1 $\{\alpha_1\beta_1\alpha_1 \to A_1, \alpha_1\beta_1\gamma_2 \to A_2, \alpha_1\beta_3 \to A_3, \alpha_4 \to A_4, B \to A_5\}$ becomes $\{\alpha_1[\beta_1[\gamma_1 \to A_1 | \gamma_2 \to A_2] | \beta_3 \to A_3],$ $\alpha_4 \to A_4, B \to A_5\}$

3.2 Symbol tables, syntax tables and rank tables

The construction method of the tables is shown with reffering to the table 1 as an example. Suppose the same forms of the reductions as in 3.1.

(1) At first, every table of R(1) is made (for quasi-sequential grammars, of R(0)).
(*symbol table*) The SI column of the symbol table is filled with the leftmost symbol

of the left-hand side of a reduction. If the left-hand side of the reduction consists of only one symbol then the LI column of that row is filled with a zero and the RI column is filled by the right-hand side of the reduction. In other case, remaining symbols of the left-hand side of the reductions sequentially fill in the SII column of the syntax table. In that case, the LI column of the symbol table is filled with the row number of the syntax table where the first symbol of the remaining symbols is placed and the RI column is filled with a zero.

(*syntax table*) The M column is filled with the symbol "k" if the content of the SII column is the immediate left-hand side symbol of "→" and in that case the RII column is filled with the immediate right-hand side of the "→". If the content of the SII column is the immediate right-hand side of "[", the ALT column of that row is filled with the row number in which the immediate right-hand side symbol of "|" is placed. However if there exists "[" before "|" then the ALT column is filled with the row number which contains the immediate right-hand side symbol of "|" which appears just after the corresponding bracket "]". Furthermore if "|"s occur in the pair of "[" and "]", the ALT column of the row which contains the immediate right-hand side symbol of the (i-1)-th "|" is filled with the row number which contains the immediate right-hand side symbol of the i-th "|". All other spaces are filled with zeros.

(*rank table*) L_1 is filled with the maximum row number of the symbol table.

(2) By the same algorithm as (1), the tables for R(2),R(3),$\cdots$,R(k) are sequentially made, and the tables for R(i) are attached under the tables for R(i-1) according to the type of tables.

Now we explain the meanings of tables. The maximum value of the subscript i of L_i shows the rank number of the grammar. In the i-th reduction process, we use the part of the symbol table whose row numbers are given by the content of L-i+k+1 (k is rank number) and the content of L-i+k plus one. The LI column is filled with the number of the row in which the next symbol to be compared is

r	SII	M	ALT	RII
1	$H_2(\alpha_1)$	0	0	0
⋮	⋮	⋮	⋮	⋮
l_1	$T(\alpha_1)$	⋮		⋮
l_1+1	$H_1(\beta_1)$	⋮	l_4+1	⋮
		⋮	0	⋮
⋮	⋮	⋮	⋮	⋮
l_2	$T(\beta_1)$	⋮	⋮	⋮
l_2+1	$H_1(\gamma_1)$	⋮	l_3+1	⋮
		⋮	0	⋮
⋮	⋮	⋮		⋮
l_3	$T(\gamma_1)$	K	⋮	A_1
l_3+1	$H_1(\gamma_2)$	0	⋮	0
⋮	⋮	⋮	⋮	⋮
l_4	$T(\gamma_2)$	K	⋮	A_2
l_4+1	$H_1(\beta_3)$	0	⋮	0
⋮	⋮	⋮	⋮	⋮
l_5	$T(\beta_3)$	K	⋮	A_3
l_5+1	$H_2(\alpha_4)$	0	⋮	0
⋮	⋮	⋮	⋮	⋮
l_6	$T(\alpha_4)$	K	⋮	A_4
		0	⋮	0
⋮	⋮	⋮	⋮	⋮

syntax table

g	SI	LI	RI
1	$H_1(\alpha_1)$	1	0
2	$H_1(\alpha_4)$	l_5+1	0
3	B	0	A_5
⋮	⋮	⋮	⋮

symbol table

$H_1(\alpha)$: the leftmost symbol of a string α

$H_2(\alpha)$: the direct right-hand side of $H_1(\alpha)$

$T(\alpha)$: the rightmost symbol of a string α

L_0	L_1	L_2	……	L_k
0	3	………	……	……

rank table

Table 1 The symbol table, the syntax table and the rank table for example 1 placed. If the content of the LI column equals to zero, it means that the content of the LI column is to be reduced to the content of the RI column. The content of the M column of the syntax table indicates the next action (i.e. if it is zero, the scanning pointer moves to right by one and if it is "k", the matched substring is reduced to the content of the RII column in that row. The content of the ALT column shows the number of the row to be compared alternatively when the symbol indicated by the scanning pointer does not match with the content of the SII column. The symbol table, the syntax table and the rank table for example 1 are shown in table 1.

4. DESCRIPTION OF FORTRAN IV BY QUASI-SEQUENTIAL GRAMMARS.

The syntax of FORTRAN IV is described by a grammar with rank(26) by regarding <primary>, <logical primary>, <format specification list group> and <Do index part> as temporary terminal symbols. However it is not a regular-type sequential grammar with rank(26) because there exist pairs of productions that violate the conditions in definition 4. It is mainly caused by the existence of the indistinguishable terminal symbols which cause a parse to be non-deterministic and back-tracked. We have proved that these terminal symbols can be identified by lexical analysis. Then on the assumption of this fact; the syntax of FORTRAN IV is described by a regular-type sequential grammar with rank(26). Furthermore it can be described by a quasi-sequential grammar with rank(27) by attaching productions with self-embedding properties. The pair of $\vdash$ and $\dashv$ is inserted into a FORTRAN source program by lexical analysis before a parse is executed. More detailed explanation is omitted owing to limited space.

5. CONCLUSION

It is shown that the syntax of FORTRAN IV can be described by a quasi-sequential grammar with rank(27). The features of the parsing algorithm of this grammar are that the parsing process is devided into 27-phase, in each phase, only one subset of the set of reductions is used and scanning-reduction process is deterministic and without back-tracking and looking-ahead symbols.

REFERENCES

1) J.E.Hopcroft and J.D.Ullman: Formal languages and their Relation to Automata, Addison-Wesley, Reading, Mass, 1969.