On the Generating Method of an Efficient Top-down Syntax Analysis Program

Kazuma Yoshimura*

Abstract

In this paper, an algorithm which automatically generates a Syntactic Analysis Program for an extended LL (k) grammar is presented. In addition, an optimization algorithm for the generated program is proposed.

1. INTRODUCTION

Over the past decade many attempts have been made to develop syntactic analysis programs of LL (k)[1] grammars automatically for compilers. This is because syntactic analysis programs for LL (k) grammars can be built top-down, and are easy to verify, and modify.[2]

Backes' method,[2] though easy to understand, retains some redundancy such as comparing nonterminal symbols in the pattern matching process.

In this report, an algorithm which automatically generates a Syntactic Analysis Program for extended LL (k) grammars is presented. In addition, an optimization algorithm is proposed for the generated programs. This optimization will produce syntactic analysis programs which aside from the look ahead symbols need only the currently scanning input symbol for a pattern matching process which calls for the next input symbol at its success.

2. REPRESENTATION OF A SYNTAX ANALYSIS ALGORITHM

Introduced in this chapter, is a language describing syntax analysis programs. By using this language a method of making syntax analysis programs for extended LL (k) grammars is presented.

2.1 A Language for Syntax Analysis Description

The extended Floyd Production language is defined in this section as:

〈label〉 ‖ 〈pattern〉 [〈look ahead string〉] (〈S-part〉)(〈F-part〉)

Here, 〈label〉 is the name of this statement. Pattern matching and look ahead strings are entered in the 〈pattern〉 and 〈look ahead string〉 columns. If a connected string in the 〈pattern〉 and 〈look ahead string〉 columns in this order coincides with a corresponding part of an input string, the process defined by the 〈S-part〉 will be performed, if not, the process defined by the 〈F-part〉 will be done.

The 'S-part' and 'F-part' are the abbriviations of success action part and failure action part. If the 〈pattern〉 columns is σ, the S-part is performed. The 〈S-part〉 and 〈F-part〉 are of the same form. Labels, numbers, #, *, ERROR and RETURN

This paper first appeared in Japanese in Joho−Shori (Journal of the Information Processing Society of Japan), Vol. 16, No. 3 (1975), pp. 195~204.

* Central Research Laboratory, Hitachi, Ltd.

may be written there separated by commas. Then, actions corresponding to them are performed in left-to-right order. Label means to call a statement labelled by this and # means to call the next statement. RETURN indicates the completion of the call corresponding to this RETURN. Label and # written in the last part of the S column or F column mean a jump to that label and the next statement respectively. The symbol * means to read one symbol from input string, and ERROR means to call the error management routine. Number represents a call of the corresponding semantic routine. The symbol * or a number may not be written at the end of the action part.

2.2 A Transformation algorithm from Backus Naur Form to the Extended Floyd Production Language

2.2.1. Definitions

Let $G = (V, \Sigma, P, S)$ be a context free grammar

Here: V = a finite set of symbols.

Σ = a subset of V, of which elements are called terminal symbols. Elements of $V-\Sigma$ are called nonterminal symbols.

P = a set of production rules of the grammar. An element of P is written in the form $\xi \rightarrow w$; here $\xi \in V-\Sigma$, $w \in V^*$.

S = the start symbol of G.

A head terminal string with length k derived from $w \in V^*$ is as:

Definition 1. $h^k(w) = \{\alpha \mid \quad |\alpha| = k, \quad \alpha \in (\Sigma \cup \vdash)^*,$

$$S \overbrace{\vdash \ldots \vdash}^{k} \overset{*}{\Rightarrow} \beta w \gamma \overset{*}{\Rightarrow} \beta\alpha\delta \}$$

Here, $|\alpha|$ means the number of symbols in the string α.

Definition 2. G is called a ALL(k) grammar, if the following conditions are satisfied. ALL(k) is an extension of LL(k).

(1) There is no B such that

$A \overset{*}{\Rightarrow} Bv \overset{*}{\Rightarrow} Av'$, $v, v' \in V^*$, $B \neq A$, $B \in V-\Sigma$ for $A \in V-\Sigma$.

(2) We classify the element of P according to the left side symbol of $\rightarrow$. A class is represented as:

$$\left.\begin{array}{l} A \rightarrow u_1 \\ \vdots \\ A \rightarrow u_{m_A} \end{array}\right\} (2.1) \qquad \left.\begin{array}{l} A \rightarrow Au_1' \\ \vdots \\ A \rightarrow Au_{n_A}' \end{array}\right\} (2.2)$$

Here head symbols of $u_1, \ldots, u_{m_A}$ are not A.

Let $D^k(A) \equiv \bigcup_{i \neq j} (h^k(u_i) \cap h^k(u_j))$ $\quad i, j = 1, 2, \ldots, m_A$ (2.3)

$E^k(A) \equiv \bigcup_{i \neq j} (h^k(u_i') \cap h^k(u_j'))$ $\quad i, j = 1, 2, \ldots, n_A$ (2.4)

$N^k(A) \equiv \{\alpha \mid S \overset{*}{\Rightarrow} \gamma A \delta, \quad \alpha \in h^k(\delta)$ here, A of $\gamma A \delta$ could not be generated using (2.2)$\}$.

Then, for all $A \in V-\Sigma$,

$D^k(A) = \phi$, $E^k(A) = \phi$, $\bigcup_{i=1}^{n_A} h^k(u_i') \cap N^k(A) = \phi$. Here ϕ means the empty set.

$D^k(A) = \phi$ is a condition sufficient to decide which production rule of (2.1) should be applied since a head string of the remaining input string should be reduced to A. $E^k(A) = \phi$ is a condition sufficient to decide which production rule of (2.2) should be applied after reduction to A has done.

The expression $h^k(u_i') \cap N^k(A) = \phi$ is a condition sufficient to decide whether the reduction to A must be continued or not.

Definition 3.

If $W_i \in 2^{V^*}$ and $S_i \in 2^{\Sigma^*}$ $(i = 1, 2, \ldots, n)$ satisfy the following condition, S_i is called a characteristic string set for separating W_i.

For all $w_i \in W_i$, there exists $w \in S_i$ such that $w \in h^{|w|}(w_i)$ and $w \notin h^{|w|}(w_j)$ for all $w_j \in W_j$, $j \neq i$; There exists $w_j \in W_j$ such that $w' \in h^{|w'|}(w_j)$, w' is an arbitrary head string of w with the length more than 1, if $|w| \geqq 2$, $i \neq j$.

2.2.2 An algorithm of transformation

Here, an algorithm which generates Floyd Production statement from ALL(k) grammar is presented.

First the characteristic sets S_i of $W_i \in 2^{V^*}$ are calculated as follows:

1) Let $S_j = \{ a \mid a \in h^1(w_j),\ w_j \in W_j \}$, put $k = 1$.

2) If there is a string α which is in two or more elements of class $\{ S_j \}$ and $|\alpha| \leqq k$, let $S_{j1}, \ldots, S_{jm}$ be all sets which have α as an element. Delete α from S_{jl} ($l = 1, \ldots, m$), insert string β into S_{jl} such that $w_{jl} \in W_{jl}$, $\beta \in h^{k+1}(w_{jl})$, and β has α as one of its head string. Do 2) as far as possible.

3) If for some i, j, $S_i \cap S_j \neq \phi$, put $k = k + 1$ and do 2).

If there is a characteristic string, the above procedure must be ended in finite steps. The characteristic string set $C^k(A, u_j)$, $j = 1, \ldots, m_A$, is computed in order to decide which production rule must be applied in (2.1); and the characteristic string set $L^k(A, u_j')$ and $M^k(A)$ in order to separate sets $u_1', \ldots, u'_{n_A}$, $N^k(A)$, $j = 1, \ldots, n_A$.

$$\text{Let: } C^k(A) = \bigcup_{i=1}^{m_A} C^k(A, u_i) \qquad L^k(A) = \bigcup_{i=1}^{n_A} L^k(A, u_i').$$

(1) The procedure $P_1(A)$ producing the program for selection of an applicable production rule from (2.1), when it is known that some head string of the remaining input string should be reduced to A, is given here.

Let $a_1, \ldots, a_l$ be all of the different head symbols of strings in $C^k(A)$ and $a_i v_{i1}, \ldots, a_i v_{it_i}$ be all of the strings which belong to $C^k(A)$ and have a_i as a head symbol. Then, make

$$\begin{array}{lllll} A\| & a_1\ [v_{11}] & (g_{11}) & (\#) & \\ & \vdots & & & \\ & a_i\ [v_{ii_j}] & (g_{ii_j}) & (\#) & \\ & \vdots & & & \\ & a_l\ [v_{lt_l}] & (g_{lt_l}) & (\text{ERROR}) & \end{array} \qquad (2.5)$$

Here, g_{ii_j} is the label and stands for Ap if $a_i v_{i_j} \in C^k(A, u_p)$.
The program starting at the label, A_p, is made by the following procedure.

If G is ALL(k), there exist only one u_p, where $a_i v_{i_j} \in C^k(A, u_p)$.

(2) The following procedure $P_2(A,U_i)$ produces a program which confirms that the head part of the remaining input string is really reduced by the production rule $A \rightarrow u_i$ when it is expected. Let $u_i = X_1 X_2 \ldots X_m$, $X_j \in V$. Make

$$\begin{array}{ll} Ai \| & S_1 \\ & \vdots \\ & S_m \end{array} \qquad (2.6)$$

Here, S_j $(1 \leqq j < m)$ represents X_j (*, #) (ERROR) or $\sigma(X_j, \#)$ depending on $X_j \in \Sigma$ or $\in V-\Sigma$ respectively, and S_m represents

X_m (*, k, \$) (ERROR) or σ $(X_m, k, \$)$

depending on $X_m \in \Sigma$ or $\in V-\Sigma$ respectively. Here $A \rightarrow u_i$ is the kth rule of G and k represents to execute the output corresponding to $A \rightarrow u_i$. The symbol \$ is replaced by RETURN or a label A# if $n_A = 0$ in (2.2) or not respectively.

(3) The procedure $P_3(A)$ which produces the program deciding to continue or end the left recursion is given here. Let $b_1, \ldots, b_s$ be all the head symbols of strings in $L^k(A) \cup M^k(A)$, and $b_j W_{j1}$, $b_j W_{j2}$, $\ldots$, $b_j W_{j\alpha_j}$ be all the strings having b_j as the head symbol in $L^k(A) \cup M^k(A)$. Then we make the following program:

$$\begin{array}{lll} A\# \| \quad b_1[\,W_{11}\,] & (g_{11}) & (\#) \\ \qquad \vdots & & \\ \qquad b_i[\,W_{ii_j}\,] & (gii_j) & (\#) \\ \qquad \vdots & & \\ \qquad b_s[\,W_{s\alpha_s}\,] & (gs\alpha_s) & (\text{ERROR}) \end{array} \qquad (2.7)$$

$i = 1, 2, \ldots, S \qquad i_j = 1, 2, \ldots, \alpha_i$

The symbol gii_j stands for the label $A\#l$ if $b_i W_{ii_j} \in L^k(A, u'_l)$ or RETURN if $b_i W_{ii_j} \in M^k(A)$ respectively. By assumption, $b_i W_{ii_j}$ belongs to only one of either $L^k(A, u'_l)$ or $M^k(A)$.

(4) Finally, let the procedure $P_4(A, u'_j)$ be the program which checks u'_j. This procedure is similar to $P_2(A)$.

For $u'_j = Y_1 Y_2 \ldots Y_n$, the following program is produced:

$$\begin{array}{ll} A\#j \| & S_1 \\ & \vdots \\ & S_n \end{array} \qquad (2.8)$$

where, S_i $(1 \leqq i < n)$ stands for Y_i (*, #) (ERROR) or $\sigma(Y_i, \#)$ if $Y_i \in \Sigma$ or $V-\Sigma$ respectively. S_n stands for Y_n (*, q, A#) (ERROR) or $\sigma(Y_n, q, A\#)$ if $Y_n \in \Sigma$ or $V-\Sigma$ respectively. Here q signifies that $A \rightarrow Au'_j$ is the q-th rule in G.

Theorem 1. A program that consists of

$$\begin{array}{ll} START \| & \sigma(S, \#) \\ & \vdash (EXIT) \quad (ERROR) \end{array}$$

and programs produced by the procedures $P_1(A)$, $P_2(A,U_i)$, $P_3(A)$ and $P_4(A,u'_j)$ for all A of $V-\Sigma$, becomes a syntax analysis program of the grammar G with the entry point

START.

2.3 A Method of Optimization

2.3.1 Unification of statements

Suppose there are some statements in (2.5) or (2.7) which have the same symbol in their pattern column but different look ahead strings.

Here, look ahead strings are used for branching only; By postponing the check of look ahead strings until they are read, these statements can be unified. The statements in (2.5) having a_i as the <pattern> column are classified by S-parts. Then erase all of the statements in the class which has the maximum number of elements. (if there are many maximum classes, choose any one).

Put one of the erased statement, eliminating its look ahead string, after the last statement which has a_i as <pattern> column. By this procedure the statements which have different look ahead symbols but the same S-parts are unified. At the beginning ERROR in the F-part of the last statement in (2.5) is replaced by #. Then perform this unifying procedure for the statements which have a_i as <pattern> part for $i = 1, \ldots, \ell$. After that, the F-part of the last statement is replaced by ERROR.

In (2.7), symbols in <pattern> column are checked again at the action part of the successfully pattern matched statement. Hence, the information of the <pattern> column is used in order to decide which statement will be executed next. Let g be one of the most overlapped labels in $g_{11}, \ldots, g_{s\alpha_s}$. Erase all the statements which have g in the S-part, change the label in F-part of the last statement to #, and add $\sigma(g)$ as the next statement.

Lemma 1. The program unified by this procedure is equivalent to the one which is made in paragraph 2.2.

2.3.2 Postponing the pattern matching

After a pattern matching has successfully been done it often happens that the same pattern matching is done at the S-part of the statement. Consequently the statement is unified by postponing the pattern matching. The sequence of the statements is called a production list if all of F-parts are # except the last one, and the last F-part is ERROR or last statement is $\sigma(S_n)$. The following expression is one example of the production list.

$$\begin{array}{lll} W_1 & (S_1) & (\#) \\ \vdots & & \\ W_{n-1} & (S_{n-1}) & (\#) \\ W_n & (S_n) & (\text{ERROR}) \end{array} \qquad (2.9)$$

Let S-part of the i-th statement of (2.9) be $(S_{i_1}, S_{i_2}, \ldots)$. Here, S_{i_1} assumed to be a label. If W_i is the pattern part of a sentence in the production list labelled S_{i_1}, the following procedure is performed according to the last statement of (2.9).

1) Not σ-statement: change F-part of the last statement to #, and change <pattern> column of the i-th statement to σ and place the i-th statement after the last statement.

2) σ-statement: if the S-part of the last statement is S_i, erase the i-th statement. If not, let Q be the S-part of the statement which has W_i as

< pattern >column in the production list labelled S_{i1}, and replace (S_i) in the i-th statement with $(Q, S_{i2}, \ldots)$. Procedure 1) is performed from the statement which has the most overlapped label in $S_{11}, S_{21}, \ldots, S_{n1}$.

Lemma 2. The program unified by this procedure is equivalent to the previous one.

2.3.3 An efficient use of pattern matching information

By the use of the information provided by pattern matching, the number of pattern matchings can be reduced.

(1) Let $W_i\ (S_{i1}, \ldots, S_{im})(Q_{i1}, \ldots)$ (2.10)

be a statement. If program S_{i1} has the statement s which has W_i at the < pattern > column and s has $S_{j1}, \ldots, S_{jk}$ at S-part column, (2.10) is replaced by

$$W_i\ (S_{j1}, \ldots, S_{jk}, S_{i2}, \ldots, S_{im})(Q_{i1}, \ldots). \qquad (2.11)$$

Do this procedure as far as possible.

(2) Suppose there is a sequence of label ... $S_{k1}, \ldots, S_{k\ell}$, ... and each of S_{kj} has the form:

$$S_{kj} \| \quad W_j \quad (S_j) \quad \text{(RETURN)} \quad j = 1, \ldots, \ell\ .$$

Then, add a new program labelled $S_{k1} \ldots . S_{k\ell}$ to the original program and replace all of the sequence of labels:

$S_{k1}, \ldots, S_{k\ell}$ by $S_{k1} \cdots S_{k\ell}$, here

$S_{k1} \cdots S_{k\ell} \| \quad W_1 \quad (S_1, S_{k2}, \ldots, S_{k\ell}) \quad (\#)$

$W_2 \quad (S_2, S_{k3}, \ldots, S_{k\ell}) \quad (\#)$

$\vdots$

$W\ell \quad (S\ell) \qquad \text{(RETURN)}$

Lemma 3. The program unified by this procedure is equivalent to the previous one.

2.3.4 Erasing σ-statement

A statement in which the pattern column consists of σ is called a σ-statement. Let a σ-statement be $S \| \sigma(S_1, \ldots, S_n)$ and let the action part of the statement which calls this σ-statement be the form:

$$(Q_1, \ldots, Q_k, \#, Q_{k+1}, \ldots) \text{ or } (T_1, \ldots, T_r, S, T_{r+1}, \ldots)\ .$$

Then, replace these action parts by $(Q_1, \ldots, Q_k, S_1, \ldots, S_n, Q_{k+1}, \ldots)$ or $(T_1, \ldots, T_r, S_1, \ldots, S_n, T_{r+1}, \ldots)$ respectively

If there are #'s in $S_1, \ldots, S_n$, replace # by some new label e.g. NS, and attach this label to the next statement of σ-statement.

Repeat this procedure for all the action parts which all σ-statements and then erase all σ-statements.

Lemma 4. The program unified by this procedure is equivalent to the previous one.

2.3.5 Other optimization procedures

(1) Erase all the statements which can not be reached from START.

(2) If there is a statement $S_k \parallel W_k \quad (Q, \text{RETURN}) \quad (Q')$ or

$S_k \parallel W_k \quad (Q, \text{RETURN}) \quad (\#)$

$Q' \parallel \ldots$

and there are some action parts which have the sequence of the labels ..., S_ℓ, S_k, S_m, ... , replace this sequence of labels by "..,S_ℓ, S_{km}..." and add the following program to the original one.

$$S_{km} \ldots \parallel \quad W_k \quad (Q, S_m, \ldots) \; (Q', S_m, \ldots)$$

From lemma 1, 2, 3 and 4 the following theorem is derived.

Theorem 2. The program produced by the procedure in 2.2 remains equivalently after taking procedure 2.3.1 - 2.3.5

3. EXAMPLES

By using an Earley's example the above algorithm is demonstrated. The Syntax is given in Table 1 and a program made by the procedure in 2.2.2 and then optimized using 2.3. is given in Table 2.

Table 1 A Syntax of Simple Arithmetic Expression

S → i ← E	1	T → T * F	6
E → T	2	F → P	7
E → ± T	3	F → F ↑ P	8
E → E ± T	4	P → i	9
T → F	5	P → (E)	10

Table 2 A Simplified Program

START	‖	σ	(S,#)	
		⊢	(Exit)	(ERROR)
S	‖	i	(*,#)	(ERROR)
		←	(*,E,1,RETURN)	(ERROR)
E	‖	±	(*,P,7,F#,5,T#,3,E#)	(P,7,F#,5,T#,2,E#)
E#	‖	±	(*,P,7,F#,5,T#,4,E#)	(RETURN)
T#	‖	*	(*,P,7,F#,6,T#)	(RETURN)
F#	‖	↑	(*,P,8,F#)	(RETURN)
P	‖	i	(*,9,RETURN)	(#)
		(	(*,E,P4)	(ERROR)
P4	‖	)	(*,10,RETURN)	(ERROR)

4. CONCLUSION

Presented here has been an algorithm which automatically generates syntactic analysis programs for extended LL(k) grammars. This method offers the following features:

i) It is not necessary to use a stack at the pattern matching process of the syntactic analysis programs for input symbols.

ii) Nonterminal symbols do not appear in the analysis programs.

iii) It is only necessary to do pattern matching for strings with the length one except look ahead strings.

5. REFERENCE

1) Lewis P. M. and Stearns, R. E: Syntax-directed transductions. J. ACM 15. 465 - 488 (1968)

2) Backes, S.: Top-down syntax analysis and Floyd Evans Production Language, IFIP '71