An Algorithm for Constructing a Semi-LL(2) Grammar's Parsing Table

KEIICHI YOSHIDA* and YOSHIKO TAKEUCHI**

Most of the researches on constructing parsing tables for LL(k) grammars are those for $k=1$, in short, for LL(1) grammars. There are few researches for LL(k) grammars in the case of $k \geqq 2$ except for Aho and Ullman's. This results from the fact that every LL(1) grammar is strong but in the case of $k \geqq 2$ there are two grammar classes, non-strong LL(k) grammars and strong LL(k) grammars.

When we construct tables for strong LL(k) grammars, where $k \geqq 2$, we can apply the same methods with LL(1) grammars. On the other hand, entirely different methods should be applied to construct tables for non-strong LL(k) grammars, where $k \geqq 2$, because the context problems are involved.

This paper presents an algorithm on constructing parsing tables for semi-LL(k) grammars (where $k=2$) derived from giving a few restrictions to LL(k) grammars, and its validity and evaluation. There are also strong and non-strong grammars in the semi-LL(k) grammars but this algorithm is relatively straighforward, and the same constructing method can be applied for both strong and non-strong semi-LL(k) grammars. An experimentation on the performance of the algorithm using some example data shows that constructing time is about 1/10, the memory size of the tables for parsing and production rules ranges about 1/120–1/400 of Aho and Ullman's algorithm. The memory size for codes, which are not affected by grammars, is larger than theirs approximately by 7%. The memory size for codes occupies about 29% of the whole program in the case of PASCAL– and about 10% in the case of ISO PASCAL. Generally, the more the number of the production rules increases, the more the rate of the codes part in the total memory decreases. Therefore, this slight increase in our case is thought to be negligible in the cases of languages in practical use.

1. Introduction

We proposed the class of semi-LL(k) grammars and a parsing method for these grammars in the article [1]. Expressive power of grammars in this class for programming languages lies between those of LL(k) and strong LL(k) grammars [1].

Until now, several construction methods of parsing tables for LL(1) grammars and parsing methods using these tables have been proposed and researched [2–6]. However, since these construction methods involve many difficulties for the case of $k \geqq 2$ than for $k=1$, no practical method was presented for this case with the exception of one in the article [5], by which constructed tables are used in table-driven type parsing. These difficulties are due to the fact that there are two types of grammars, namely, strong LL(k) and non-strong LL(k) for $k \geqq 2$, and for non-strong LL(k) grammars, it may happen that the parsing tables must be constructed so as to decide applicable productions context-dependently.

This paper proposes a table construction method adding a new idea to the original method proposed in the articles [4] and [6]. Comparing our method to Aho-Ullman's method with regard to their performance, our experimentation showed that table construction time and table size by the former are about 1/10 and about 1/120–1/400 of those by the latter, respectively. Furthermore, another strong features of our method are that parsing tables are constructed mainly by table-manipulation almost without set calculation and the construction is much easier than by Aho-Ullman's method.

This paper describes an algorithm to construct parsing tables for semi-LL(2) grammars which have the outstanding features described above, and also shows the validity and evaluation of the algorithm.

2. Outline of Aho-Ullman's Method

Aho-Ullman's method consists of two phases. One is to convert the productions of a given LL(2) grammar to strong-typed ones, and the other is to construct the parsing table of this converted grammar. For this conversion it is necessary to produce new nonterminal symbols

This is a translation of the paper that appeared originally in Japanese in Transactions of IPSJ, Vol. 31, No. 6 (1990), pp. 916–929.

*Department of Computer Science, College of Engineering, Shizuoka University.

**Department of Computer Science, Hamamatsu Polytechnic College.

T_1, T_2, $\cdots$, and T_n for an original nonterminal, for example A, when the nonterminal A has n distinct right contexts on sentential forms including A. Because an original nonterminal symbol can usually have two or more right contexts, the number of newly produced nonterminals is guessed to become very large and to be time-consuming to generate them. According to their method implemented by authors, the numbers of nonterminals of PASCAL- (PASCAL minus) grammar increased from 54 to 400 or more for the case of $k=2$.

Furthermore, the conversion of productions from the given grammar to a strong LL(2) grammar using these newly produced non-terminals increases the numbers of the production rules from 98 to 800 or more, according to our experimentation for Aho-Ullman's method.

After converting a given grammar to strong LL(2), a parsing table for the grammar is constructed. Our experimentation following their method shows that memory area requires 7.4 MB for the tables of parsing and converted production rules in the case of PASCAL- grammar.

3. Definitions and Notations

This section describes symbols used in this paper. Other concepts and symbols used without any definition are based on the article [1].

[Definition 1]

A context free grammar (CFG for short) G is defined by:

$$G=(N, \Sigma, P, S)$$

where the symbols N and Σ denote finite sets of nonterminals and terminals, respectively, and P denotes a finite set of productions. The symbol S denotes the start symbol in G.

[Notation 1]

A, B, and C denote elements in N; a, b, and c denote elements in Σ; X, Y, and Z denote elements in $N \cup \Sigma$; s, t, and u denote elements in Σ^*; α, β, and γ denote elements in $(N \cup \Sigma)^*$. ε and ϕ denote a null string and an empty set, respectively. Each symbol except ε and ϕ is allowed to have subscripts if necessary.

[Definition 2]

Set $\text{FIRST}_k(\alpha)$ is defined by:

$$\text{FIRST}_k(\alpha)=\{u \mid (\alpha \overset{*}{\Rightarrow} u\beta,\ \|u\|=k) \text{ or } (\alpha \overset{*}{\Rightarrow} u,\ \|u\|<k)\}$$

where α and $\beta \in (N \cup \Sigma)^*$, $u \in \Sigma^*$, and $\|u\|$ denotes the length of string u. $\alpha \overset{*}{\Rightarrow} u\beta$ denotes a derivation using productions zero or more times, and the symbol $\overset{k}{\Rightarrow}$ denotes a derivation using productions just k times. Finally, the symbol $\Rightarrow$ denotes a derivation using just a production. All derivations in this paper are done in the leftmost way unless otherwise stated.

[Definition 3]

Set END-FOLLOW(X) is defined by:

$$\text{END-FOLLOW}(X)=\{A \mid (A \overset{*}{\Rightarrow} \alpha X\beta,\ X \in N,\ \beta \overset{*}{\Rightarrow} \varepsilon) \text{ or } (A \overset{*}{\Rightarrow} \alpha BX,\ X \in N)\}$$

[Definition 4] [2]

Let L_1 and L_2 be subsets of Σ^*, then an operator $\oplus_k$ is defined by:

$$L_1 \oplus_k L_2=\{w \mid \text{for some } x \in L_1 \text{ and } y \in L_2, \text{ if } \|xy\|<k \text{ then } w=xy, \text{ if } \|xy\|>k \text{ then } w=u, \text{ where } xy=uv, \text{ and } \|u\|=k\}$$

[Definition 5]

Let $G=(N, \Sigma, P, S)$ be a CFG. If $S \overset{*}{\Rightarrow} u_i AX\xi_i$, then PF(partial-FOLLOW) is defined by:

$$\text{PF}_k(A, X)=\cup_i \text{FIRST}(X\xi_i)$$

[Definition 6]

Let $G=(N, \Sigma, P, S)$ be a CFG. We say that G is a semi-LL(k) grammar if $(\text{FIRST}_k(\alpha) \oplus_k \text{PF}_k(A, X)) \cap (\text{FIRST}_k(\beta) \oplus_k \text{PF}_k(A, X))=\phi$ holds for all $uAXv$ such that $S \overset{*}{\Rightarrow} uAXv$, where $A \rightarrow \alpha$ and $A \rightarrow \beta$ are distinct productions in P.

4. Structure of a Parsing Table

4.1 Fundamental Definitions

Several definitions are given, being necessary to describe our algorithm constructing parsing tables.

[Definition 7]

Let $G=(N, \Sigma, P, S)$ be a CFG. An augmented grammar G' is defined by:

$$G'=(N', \Sigma', P', S')$$

where $N'=N \cup \{S'\}$, $\Sigma'=\Sigma \cup \{\$\}$, and $P'=P \cup \{S' \rightarrow S\$\$\}$. From now on, our discussion is developed with this augmented grammar G'. For convenience, the symbols N', Σ', and P' are replaced by N, Σ, and P, respectively.

[Notation 2]

p in $A \underset{p}{\rightarrow} \alpha$ denotes the proper index, a positive integer, affixed to production $A \rightarrow \alpha$, and p in $\alpha \underset{p}{\Rightarrow} \beta$ denotes the index of production used in the leftmost derivation $\alpha \Rightarrow \beta$.

[Definition 8]

Let p and q be production indices. Notation $[\ \ ]p(q)$ and $[X]p(q)$ are called π type production indices. Here, p and q are the indices of productions used in the following derivation:

$$S' \overset{*}{\Rightarrow} u'Y\alpha' \underset{q}{\Rightarrow} u'\xi A\gamma\alpha' \overset{*}{\Rightarrow} uA\alpha \underset{p}{\Rightarrow} u\beta\alpha$$

where $\alpha=\gamma\alpha'$, and X denotes the leftmost symbol of α. π type production indices are used only for constructing parsing tables. In π type production index $[X]p(q)$, if $X=\varepsilon$ and $q=\varepsilon$, then it is equivalent to $[\ \ ]p(\ \ )$.

[Definition 9]

A production index $[X]p$, which is produced by deleting (q) from $[X]p(q)$, is called τ type production index. And also, if $X=\varepsilon$ in $[X]p$, then it is equivalent to [ ]p.

[Definition 10]

A parsing table T is a matrix whose rows and columns are named by elements of set $(N-\{S'\})\cup\Sigma$ and set Σ, respectively. Notation $T(A, a)$ and $T(a, b)$ denote the cell of row A and column a, and the cell of row a and column b of table T, respectively. Each entry in these cells is either a set of τ type production index or nil.

[Notation 3]

The notation ${}^{(i)}\beta$ indicates the symbol on the i-th position from the leftend of the string β.

4.2 Some Properties of Parsing Table

A parsing table should have the following properties from the definition on semi-LL(2) grammars [1]:

[Property 1]

$$S' \overset{*}{\Longrightarrow} uAv \underset{p}{\Longrightarrow} u\alpha v \overset{*}{\Longrightarrow} uab\xi v$$
$$\leftrightarrow [\ \ ]p \in T(A, a) \text{ and } [\ \ ]p \in T(a, b)$$

[Property 2]

$$S' \overset{*}{\Longrightarrow} uAv \underset{p}{\Longrightarrow} u\alpha v \overset{*}{\Longrightarrow} uav \overset{*}{\Longrightarrow} uab\zeta$$
$$\leftrightarrow [\ \ ]p \in T(A, a) \text{ and } [{}^{(1)}v]p \in T(a, b)$$

[Property 3]

$$S' \overset{*}{\Longrightarrow} uAv \underset{p}{\Longrightarrow} u\alpha v \overset{*}{\Longrightarrow} uv \overset{*}{\Longrightarrow} uab\psi$$
$$\leftrightarrow [{}^{(1)}v]p \in T(A, a) \text{ and } [{}^{(1)}v]p \in T(a, b)$$

[Property 4]

There exists no other entry in cells of the parsing table except for the ones described above.

4.3 Several Tables Necessary in the Construction of the Parsing Table

This section describes the correspondence of the definitions stated in Section 3 and the structure of several tables necessary in the construction of parsing tables with the properties presented in Section 4.2.

(1) FIRST table: although symbol FIRST originally denotes the set defined by [Definition 2], it is also used as the name of a table corresponding to the set, in Algorithm θ described in the next section. The table is a matrix whose rows are named by the elements of set $N\cup\Sigma$, and the columns are named by the elements of set $N\cup\Sigma$. The entry in cell of the table is either a set of π type production indices or nil. The symbols FT_1 and FT_2 indicate the parts of $N\times(N\cup\Sigma)$ and $\Sigma\times(N\cup\Sigma)$ of FIRST table, respectively. A set $FIRST_2$ corresponds to the sum of the parts of $N\times\Sigma$ in FT_1 and $\Sigma\times\Sigma$ in FT_2. The following shows their correspondence:

$$ab \in FIRST_2(A) \leftrightarrow [X]p(q) \in FT_1(A, a)$$
$$\text{and } [Y]p(r) \in FT_2(a, b)$$

where p, q, and r denote the production indices, and X and Y are elements of set $N\cup\Sigma\cup\{\varepsilon\}$.

(2) PF table: although symbol PF denotes a set defined by [Definition 5], it is also used as the name of table corresponding to the set, in Algorithm θ. Its table structure is similar to FIRST table. The entry in cell of the table is either a set of π type production indices or nil. The symbols PL_1 and PL_2 denote the parts of $N\times(N\cup\Sigma)$ and $\Sigma\times(N\cup\Sigma)$ of PF table, respectively. A set PF_2 corresponds to the sum of the parts of $N\times\Sigma$ in PL_1 and $\Sigma\times\Sigma$ in PL_2. The following shows their correspondence:

$$ab \in PF_2(A, X) \leftrightarrow [X]p(q) \in PL_1(A, a)$$
$$\text{and } [Y]p(r) \in PL_2(a, b)$$

where p, q, and r are production indices, and, X and $Y \in N\cup\Sigma\cup\{\varepsilon\}$.

(3) END-FOLLOW table: although symbol END-FOLLOW denotes the set defined by [Definition 3], it is also used as the name of the table corresponding to the set, in Algorithm θ. The table is a matrix whose rows are named by elements of set $N\cup\Sigma$, and the columns are named by elements of set N. The entry in cell of the table is either a set of π type production indices or nil. Representing END-FOLLOW table by symbol EF, the correspondence of set END-FOLLOW and table EF is as follows:

$$A \in \text{END-FOLLOW}(Y) \leftrightarrow [X]p(q) \in EF(Y, A)$$

where p and q are production indices, $Y \in N\cup\Sigma$, and $X \in N\cup\Sigma\cup\{\varepsilon\}$. The required parsing table is the part of $(N\cup\Sigma)\times\Sigma$ of table FIRST, being obtained from Step 10 of Algorithm θ. As mentioned before, the entry of cell of this table is either a set of π type production indices or nil.

5. Algorithm

For convenience, the algorithm proposed here is named Algorithm θ. First, the definitions of notation and sets used in Algorithm θ are described.

[Notation 4]

In Algorithm θ the same notations as proposed in [Definition 10] are used to represent every cell of tables.

[Definition 11]

Sets Q, R, and Γ are defined by:

$$Q = \{(A, p) \mid A \underset{p}{\Longrightarrow} \alpha \overset{*}{\Longrightarrow} \varepsilon\}$$
$$R = \{(A, a, p) \mid A \underset{p}{\Longrightarrow} \alpha \overset{*}{\Longrightarrow} a\}$$
$$\Gamma = \{p \mid p \text{ is the index of a production rule}\}.$$

Next, the details of Algorithm θ are given as follows:

[STEP 1]

Initializing FIRST-table. That is, [ ]$p(p)$ is added to $FT_1(A, Y_i)$ if $A \underset{p}{\longrightarrow} \alpha Y_i \beta$ and $\alpha \overset{*}{\Longrightarrow} \varepsilon$, and [ ]$p(p)$ is added to $FT_1(A, Y_i)$ and $FT_2(Y_i, {}^{(1)}\beta)$ if $Y_i \in \Sigma$. [ ]$p(p)$ is added to $FT_2(a, {}^{(1)}\beta)$ if $Y_i \underset{p_i}{\Longrightarrow} \gamma \overset{*}{\Longrightarrow} a$, or $(Y_i, a, p_i) \in R$.

First, let every cell of the FIRST-table be nil.

begin

```
for each production such that A →_p Y_1Y_2···Y_n
and Y_1Y_2···Y_n≠ε do
        begin
          i←0;
          repeat
            i←i+1;
            FT_1(A, Y_i)←FT_1(A, Y_i)∪{[  ]p(p)};
          /* finding the second symbol Y_j on a str-
            ing derived from A  */
        if i+1≦n then
          begin
            if Y_i∈Σ then X←'Y_i'
            else if (Y_i, a, p_i)∈R then X←'a'
              else goto l;
            j←i;
            repeat
              j←j+1;
              FT_2(X, Y_j)←FT_2(X, Y_j)∪{[  ]p(p)};
            until (Y_j, p_j)∉Q or j+1>n
          end;
        l:
      until (Y_i, p_i)∉Q or i+1>n
    end
  end.
```

[STEP 2]

Computing the closure of FIRST-table. In the first half of this step, [ ]$p(p)$ is added to $\mathrm{FT}_1(A, C)$ if $A \underset{p}{\Rightarrow} \alpha B\beta$, $B \overset{*}{\Rightarrow} \gamma C\delta$, and $\alpha\gamma \overset{*}{\Rightarrow} \varepsilon$. In the second half of this step, nonterminal, for example, B in $\mathrm{FT}_1(A, B)$, is replaced with terminal as follows:

```
  begin
    repeat
      for each A, B, C∈N do
        if [  ]p(p) is in FT_1(A, B) and [  ]q(q) is in
        FT_1(B, C) then
          FT_1(A, C)←FT_1(A, C)∪{[  ]p(p)};
    until no change occurs in the FIRST-table;
    repeat
      for each A, B∈N, and a∈Σ do
        if FT_1(B, a)≠φ then
          begin
            for each [  ]p(p)∈FT_1(A, B) do
              begin
                FT_1(A, a)←FT_1(A, a)∪{[  ]p(p)};
                /* finding the second symbol X on a
                string derived from A  */
                for each X∈N∪Σ do
                  if FT_1(B, a)∩FT_2(a, X)≠φ then
                    FT_2(a, X)
                          ←FT_2(a, X)∪{[  ]p(p)}
              end
          end
    until no change occurs in the FIRST-table;
    if R={  } and Q={  } then skip Step 3 through
    Step 7
  end.
```

[STEP 3]

Initializing PF-table. That is, $[^{(1)}\xi]p(p)$ is added to $\mathrm{PL}_1(Y_i, Y_j)$ if $A \underset{p}{\rightarrow} \alpha Y_i\beta Y_j\gamma$ and $\beta \overset{*}{\Rightarrow} \varepsilon$, and in addition, $[^{(1)}\xi]p(p)$ is added to $\mathrm{PL}_1(Y_i, Y_j)$ and $\mathrm{PL}_2(Y_j, {}^{(1)}\gamma)$ if $Y_j\in\Sigma$, where $\xi=\beta Y_j$.

First, let every cell of the PF-table be nil.

```
  begin
    PL_1(S, $)←'[$]0(0)'; /* 0 is the index of produc-
                             tion S'→S$$  */
    PL_2($, $)←'[$]0(0)';
    for each production such that A →_p Y_1Y_2···Y_n
    and Y_1Y_2···Y_n≠ε do
      begin
        i←1;
        repeat
          j←i+1;
          if Y_i∈N and j≦n then
            begin
              PL_1(Y_i, Y_j)←PL_1(Y_i, Y_j)∪{[Y_j]p(p)};
              k←j;
              if Y_k∈N then
                begin
                  while (Y_k, p_k)∈Q and k+1≦n do
                    begin
                      k←k+1;
                      PL_1(Y_i, Y_k)
                      ←PL_1(Y_i, Y_k)∪{[Y_j]p(p)}
                    end
                end;
          /* finding the second symbol Y_m on a str-
          ing derived from Y_i  */
              if Y_k∈Σ then
                if k+1≦n then
                  begin
                    m←k+1;
                    PL_2(Y_k, Y_m)
                    ←PL_2(Y_k, Y_m)∪{[Y_j]p(p)};
                    while (Y_m, p_m)∈Q and m+1≦n
                    do
                      begin
                        m←m+1;
                        PL_2(Y_k, Y_m)
                        ←PL_2(Y_k, Y_m)∪{[Y_j]p(p)}
                      end
                  end
            end;
          i←i+1
        until i≧n
      end
  end.
```

[STEP 4]

Initializing END-FOLLOW-table. That is, for such a production as $A \underset{p}{\rightarrow} Y_1Y_2\cdots Y_n$ and $Y_1Y_2\cdots Y_n\neq\varepsilon$, [ ] $p(p)$ is added to $\mathrm{EF}(Y_n, A)$ if $Y_n\in N$, or $[Y_n]p(p)$ is added to $\mathrm{EF}(Y_n, A)$ if $Y_n\in\Sigma$ and $Y_{n-1}\in N$.

First, let every cell of the END-FOLLOW-table be nil.

```
  begin
```

for each production such that $A \underset{p}{\rightarrow} Y_1Y_2\cdots Y_n$ and $Y_1Y_2\cdots Y_n \neq \varepsilon$ do
if $Y_n \in N$ then
begin
$\mathrm{EF}(Y_n, A) \leftarrow \mathrm{EF}(Y_n, A) \cup \{[\ \ ]p(p)\}$;
$i \leftarrow n$;
while $(Y_i, p_i) \in Q$ and $i > 1$ do
begin
if $Y_{i-1} \in N$ then
$\mathrm{EF}(Y_{i-1}, A)$
$\leftarrow \mathrm{EF}(Y_{i-1}, A) \cup \{[Y_i]p(p)\}$;
$i \leftarrow i-1$
end
end
else if $Y_{n-1} \in N$ then
$\mathrm{EF}(Y_n, A) \leftarrow \mathrm{EF}(Y_n, A) \cup \{[Y_n]p(p)\}$
end.

[STEP 5]

Computing the closure of END-FOLLOW-table. That is, when $B \underset{q}{\Rightarrow} \alpha_1B_1\beta_1$, $B_1 \Rightarrow \alpha_2B_2\beta_2, \cdots, B_{n-1} \underset{p}{\Rightarrow} \alpha_nX\beta_n$, and $C \underset{h}{\Rightarrow} \gamma_1C_1\delta_1$, $C_1 \Rightarrow \gamma_2C_2\delta_2, \cdots, C_{n-1} \underset{g}{\Rightarrow} \gamma_nB\delta_n$, $[V]p(h)$ is added to $\mathrm{EF}(X, C)$ if $X \in N$, or $[a]p(p)$ is added to $\mathrm{EF}(a, C)$ if $X = a \in \Sigma$. Here $\beta_n\beta_{n-1}\cdots\beta_1 \overset{*}{\Rightarrow} \varepsilon$, $\delta_n\delta_{n-1}\cdots\delta_1 \overset{*}{\Rightarrow} \varepsilon$, and symbols V, Z, and Y denote the leftmost symbol of strings $\beta_n\beta_{n-1}\cdots\beta_1\delta_n\delta_{n-1}\cdots\delta_1$, $\beta_n\beta_{n-1}\cdots\beta_1$, and $\delta_n\delta_{n-1}\cdots\delta_1$, respectively.

begin
repeat
for each $B, C \in N$, Y and $Z \in N \cup \{\varepsilon\}$, and $X \in N$ do
if $[Z]p(q)$ is in $\mathrm{EF}(X, B)$ and $[Y]g(h)$ is in $\mathrm{EF}(B, C)$ then
begin
if $Z \neq \varepsilon$ then $V \leftarrow$ 'Z' else $V \leftarrow$ 'Y';
$\mathrm{EF}(X, C) \leftarrow \mathrm{EF}(X, C) \cup \{[V]p(h)\}$
end
until no change occurs in the END-FOLLOW-table;
for each $B, C \in N$, $Y \in N \cup \{\varepsilon\}$, and $X \in \Sigma$ do
if $[X]p(p)$ is in $\mathrm{EF}(X, B)$ and $\mathrm{EF}(B, C) \neq \phi$ then
$\mathrm{EF}(X, C) \leftarrow \mathrm{EF}(X, C) \cup \{[X]p(p)\}$
end.

[STEP 6]

Completing PF-table using END-FOLLOW-table. First, $[a]p(p)$ is added to $\mathrm{PL}_2(a, Y)$ if $C \Rightarrow \alpha_1B\xi Y\beta$, $B \Rightarrow \gamma_1X_1\delta_1$, $X_1 \Rightarrow \gamma_2X_2\delta_2, \cdots$, and $X_{n-1} \underset{p}{\Rightarrow} \gamma_nXa$, where $\delta_{n-1}\cdots\delta_1 \overset{*}{\Rightarrow} \varepsilon$. Secondly, $[V]p(r)$ is added to $\mathrm{PL}_1(A, Y)$ if $B \underset{s}{\Rightarrow} \alpha_1A_1\beta_1$, $A_1 \Rightarrow \alpha_2A_2\beta_2, \cdots, A_{n-1} \Rightarrow \alpha_nA\beta_n$, $C \underset{r}{\Rightarrow} \gamma_1X_1\xi_1Y\delta$, $X_1 \Rightarrow \gamma_2X_2\xi_2, \cdots$, and $X_{n-1} \underset{q}{\Rightarrow} \gamma_nB\xi_n$, where $\xi_n\xi_{n-1}\cdots\xi_1 \overset{*}{\Rightarrow} \varepsilon$. $[V]p(r)$ is added to $\mathrm{PL}_1(A, Y)$ and $\mathrm{PL}_2(Y, W)$ if $Y \in \Sigma$. (cf. Fig. Appendix A). Here, symbols V and W indicate the leftmost symbol of strings $\beta_n\beta_{n-1}\cdots\beta_1\xi_n\xi_{n-1}\cdots\xi_1$ and δ, respectively.

begin
for each $B \in N$, Y and $Z \in N \cup \Sigma$, $X \in N \cup \Sigma \cup \{\varepsilon\}$, and $a \in \Sigma$ do
if $[a]p(p)$ is in $\mathrm{EF}(a, B)$ and $\mathrm{PL}_1(B, Y) \neq \phi$ then
$\mathrm{PL}_2(a, Y) \leftarrow \mathrm{PL}_2(a, Y) \cup \{[a]p(p)\}$;
for each $A, B \in N$, $Y \in N \cup \Sigma$, and $X, Z \in N \cup \Sigma \cup \{\varepsilon\}$ do
if $[X]p(s)$ is in $\mathrm{EF}(A, B)$ and $[Z]q(r)$ is in $\mathrm{PL}_1(B, Y)$ then
begin
if $X \neq \varepsilon$ then $V \leftarrow$ 'X' else $V \leftarrow$ 'Z';
$\mathrm{PL}_1(A, Y) \leftarrow \mathrm{PL}_1(A, Y) \cup \{[V]p(r)\}$;
/* finding second symbol W on a string derived from A */
if $Y \in \Sigma$ then
for each $W \in N \cup \Sigma$
if $\mathrm{PL}_1(B, Y) \cap \mathrm{PL}_2(Y, W) \neq \phi$ then
$\mathrm{PL}_2(Y, W) \leftarrow \mathrm{PL}_2(Y, W) \cup \{[V]p(r)\}$
end;
if $R = \{\ \ \}$ then goto Step 8
end.

[STEP 7]

Processing the case of $A \overset{*}{\Rightarrow} a$. That is, $[Z]p(r)$ is added to $\mathrm{FT}_2(a, X)$ if $A \underset{p}{\Rightarrow} \alpha \overset{*}{\Rightarrow} a$, $B \underset{r}{\Rightarrow} \alpha_1X_1Y\beta_1 \overset{*}{\Rightarrow} \alpha_1\cdots\alpha_{n-1}X_{n-1}\beta_{n-1}\cdots\beta_2Y\beta_1 \underset{q}{\Rightarrow} \alpha_1\alpha_2\cdots\alpha_nA\beta_n\beta_{n-1}\cdots\beta_2Y\beta_1$, and $\beta_n\beta_{n-1}\cdots\beta_2 \overset{*}{\Rightarrow} \varepsilon$, where Z indicates the leftmost symbol of string $\beta_n\beta_{n-1}\cdots\beta_2$.

begin
for each $(A, a, p) \in R$ do
for each $X, Z \in N \cup \Sigma$
if $[Z]q(r)$ is in $\mathrm{PL}_1(A, X)$ then
$\mathrm{FT}_2(a, X) \leftarrow \mathrm{FT}_2(a, X) \cup \{[Z]p(r)\}$
end.

[STEP 8]

Replacing A in $\mathrm{FT}_2(a, A)$ with a terminal, if $A \overset{*}{\not\Rightarrow} \varepsilon$.

begin
for each $a \in \Sigma$, $A \in N$, and $X \in N \cup \Sigma \cup \{\varepsilon\}$ do
if $[X]p(q)$ is in $\mathrm{FT}_2(a, A)$ and $\mathrm{FT}_1(A, b) \neq \phi$ then
$\mathrm{FT}_2(a, b) \leftarrow \mathrm{FT}_2(a, b) \cup \{[X]p(q)\}$;
if $Q = \{\ \ \}$ then skip Step 9 through Step 10
end.

[STEP 9]

Computing $\mathrm{PL}_1(A, a)$ and $\mathrm{PL}_2(a, b)$ for the case of $u_0D\beta_0 \underset{q}{\Rightarrow} u_0\alpha_1D_1\beta_1\beta_0 \overset{*}{\Rightarrow} u_1D_1\beta_1\beta_0 \overset{*}{\Rightarrow} u_{n-1}D_{n-1}\beta_{n-1}\beta_{n-2}\cdots\beta_1\beta_0 \overset{*}{\Rightarrow} u_{n-1}\alpha_nA\gamma B\beta_n\beta_{n-1}\cdots\beta_1\beta_0 \overset{*}{\Rightarrow} uA\gamma B\beta_n\beta_{n-1}\cdots\beta_1\beta_0 = uA\gamma B\beta \underset{h}{\Rightarrow} u\alpha\gamma B\beta \overset{*}{\Rightarrow} u\gamma B\beta \overset{*}{\Rightarrow} uB\beta \underset{p}{\Rightarrow} u\delta\beta \overset{*}{\Rightarrow} uab\xi$. That is, if $\mathrm{PL}_1(A, B) \neq \phi$ and $A \overset{*}{\Rightarrow} \varepsilon$, first, the case of $B \overset{*}{\Rightarrow} ab\xi$ is processed, secondly, the case of $B \overset{*}{\Rightarrow} a$ and $\beta \overset{*}{\Rightarrow} b\xi$ is processed. In addition, if ${}^{(1)}\beta$ in $\mathrm{PL}_2(a, {}^{(1)}\beta)$ is a nonterminal, it is replaced with a terminal.

begin
for each $(A, h) \in Q$, $B \in N$, $X \in N \cup \Sigma$, $Z \in N \cup \Sigma \cup \{\varepsilon\}$, and $a, b \in \Sigma$ do
if $\mathrm{PL}_1(A, B) \neq \phi$ then
begin
for each $[\ \ ]p(p)$ in $\mathrm{FT}_1(B, a)$ and $[Z]p'(q)$ in $\mathrm{FT}_2(a, b)$ do
if $p = p'$ then

```
        if p=q then
          begin /* Bβ =*=> abζβ=abξ */
            for each [X]r(s) in PL1(A, B) do
              begin
                PL1(A, a)
                  ←PL1(A, a)∪{[X]r( )};
                PL2(a, b)
                  ←PL2(a, b)∪{[X]r( )}
              end
          end
        else
          begin /* Bβ =*=> aβ =*=> abξ */
            for each C∈N∪Σ and Y∈N∪Σ
            do
              for each [Y]t(u) in PL1(B, C)
              and [X]r(s) in PL1(A, B) do
                if s=t and u=q then
                  begin
                    PL1(A, a)
                      ←PL1(A, a)∪{[X]r( )};
                    PL2(a, b)
                      ←PL2(a, b)∪{[X]r( )}
                  end
          end
    end;
    /* Replacing nonterminal as the second symbol
    with terminal symbol */
    for each B∈N, Z∈N∪Σ, and a, b∈Σ do
      if [Z]p(q) is in PL2(a, B) and FT1(B, b)≠φ then
        PL2(a, b)←PL2(a, b)∪{[Z]p(q)}
  end.
```

[STEP 10]

The case of $A \underset{p}{\Rightarrow} \alpha \overset{*}{\Rightarrow} \varepsilon$. That is, computing $FT_1(A, a)$ and $FT_2(a, b)$ when $S' \overset{*}{\Rightarrow} uA\gamma \underset{p}{\Rightarrow} u\alpha\gamma \overset{*}{\Rightarrow} u\gamma \overset{*}{\Rightarrow} uab\xi$.

```
  begin
    for each A such that (A, p)∈Q do
      for each a, b∈Σ do
        for each [X]q(r) in PL1(A, a)∩PL2(a, b) do
          for each X∈N∪Σ do
            begin
              FT1(A, a)←FT1(A, a)∪{[X]p( )};
              FT2(a, b)←FT2(a, b)∪{[X]p( )}
            end
  end.
```

[STEP 11]

Constructing the required parsing table. That is, π type production indices are converted to τ type production indices by deleting (r) from $[Z]p(r)$.

```
  begin
    for each A∈N, and a, b∈Σ do
      delete (r) from [Z]p(r) in FT1(A, a) and FT2(a,
        b)
  end.
```

[End of Algorithm]

6. Proof

Here, we prove $T(=T_1+T_2)$ is the semi-LL(2)parsing table for the given G, where T_1 and T_2 are $N\times\Sigma$ part and $\Sigma\times\Sigma$ part of FIRST table constructed by Step 11 of Algorithm θ from G, respectively. π type production indices are necessary in constructing parsing tables, and in the final step of Algorithm θ (Step 11), they are converted to τ type production indices. Thus, π type production indices are used only in the construction of the parsing table, and they are equivalent to τ type production indices as far as related to parsing.

Thus, to show that the constructed parsing table is valid, it is enough to prove the following:

$$S' \overset{*}{\Rightarrow} uAv \underset{p}{\Rightarrow} u\alpha v \overset{*}{\Rightarrow} uab\xi \leftrightarrow [X]p\in T_1(A, a), \text{ and } [Y]p\in T_2(a, b) \quad (1)$$

where X, $Y\in N\cup\Sigma\cup\{\varepsilon\}$, and if $X=\varepsilon$ then $Y={}^{(1)}v$ or $Y=\varepsilon$, and if $X\neq\varepsilon$ then $Y=X={}^{(1)}v$. FT_1^i, FT_2^i, PL_1^i, PL_2^i, and EF^i indicate the values (namely, nil or a set of π type production) added by the application of Step i of Algorithm θ to FT_1, FT_2, PL_1, PL_2, and EF, respectively. $FT_1^{i/j}$, $FT_2^{i/j}$, $PL_1^{i/j}$, $PL_2^{i/j}$, and $EF^{i/j}$ indicate the values (namely, nil or a set of π type production) added by the application of Step i or Step j to FT_1, FT_2, PL_1, PL_2, and EF, respectively.

Since the given grammar is a semi-LL(2), to prove the relation (1), it is enough to prove the next relations:

Case (i) $A \underset{p}{\rightarrow} \alpha$, and $\alpha \overset{*}{\Rightarrow} ab\xi$

$$S' \overset{*}{\Rightarrow} uAv \underset{p}{\Rightarrow} u\alpha v \overset{*}{\Rightarrow} uab\xi v$$
$$\leftrightarrow FT_1^{1/2}(A, a)\ni [\ \ ]p, \text{ and}$$
$$FT_2^{1/2/8}(a, b)\ni [\ \ ]p,$$

Case (ii) $A \underset{p}{\rightarrow} \alpha$, $\alpha \overset{*}{\Rightarrow} a$, and $v \overset{*}{\Rightarrow} b\zeta$

$$S' \overset{*}{\Rightarrow} uAv \underset{p}{\Rightarrow} u\alpha v \overset{*}{\Rightarrow} uav \overset{*}{\Rightarrow} uab\zeta$$
$$\leftrightarrow FT_1^{1/2}(A, a)\ni [\ \ ]p, \text{ and}$$
$$FT_2^{7/8}(a, b)\ni [{}^{(1)}v]p,$$

Case (iii) $A \underset{p}{\rightarrow} \alpha$, $\alpha \overset{*}{\Rightarrow} \varepsilon$, and $v \overset{*}{\Rightarrow} ab\psi$

$$S' \overset{*}{\Rightarrow} uAv \underset{p}{\Rightarrow} u\alpha v \overset{*}{\Rightarrow} uv \overset{*}{\Rightarrow} uab\psi$$
$$\leftrightarrow FT_1^{10}(A, a)\ni [{}^{(1)}v]p, \text{ and}$$
$$FT_2^{10}(a, b)\ni [{}^{(1)}v]p.$$

As mentioned before, there are cases to decide context-dependently an applicable production, that is, dependently on a grammar symbol on the immediate right of the nonterminal to apply the production. (the above cases, (ii) and (iii)). Therefore, we must use π type production indices in the following proof.

Proof of Case (i)

Since G does not produce any cyclic derivation, the following derivation is done in finite steps:

$$A \underset{p}{\Rightarrow} \alpha \overset{*}{\Rightarrow} abv$$

where $a, b \in \Sigma$, and $v \in (N \cup \Sigma)^*$. If the number of steps in the derivation is $k_0 (=k+1)$, then the above derivation can be rewritten as follows:

$$A \underset{p}{\Rightarrow} \alpha \overset{k}{\Rightarrow} abv$$

where k is a non-negative integer. This derivation can generally be rewritten as follows:

$$A \underset{p}{\Rightarrow} \alpha = \mu_1 X_1 v_1 \overset{*}{\Rightarrow} X_1 v_1 \overset{k_1}{\Rightarrow} ab\eta_1, \text{ and } \eta_1 = v \qquad (2)$$

where $X_1 \overset{*}{\not\Rightarrow} \varepsilon$ and $k_1 < k_0$. Then, grammar G must have the following production:

$$A \underset{p}{\rightarrow} \mu_1 X_1 v_1$$

where $\mu_1 \overset{*}{\Rightarrow} \varepsilon$. Thus, the following relation can be obtained from Step 1 of Algorithm θ:

$$A \underset{p}{\rightarrow} \mu_1 X_1 v_1, \text{ and } \mu_1 \overset{*}{\Rightarrow} \varepsilon$$
$$\longleftrightarrow FT_1^1(A, X_1) \ni [\ \ ]p(p), \text{ and}$$
$$FT_2^1(X_1, {}^{(1)}v_1) \ni [\ \ ]p(p) \text{ if } X_1 \in \Sigma.$$

Moreover, the derivation from symbol X_1 of (2) can be rewritten as follows:

$$X_1 \underset{p_1}{\Rightarrow} \mu_2 X_2 v_2 \overset{*}{\Rightarrow} X_2 v_2 \overset{k_2}{\Rightarrow} ab\eta_2$$

where $X_2 \overset{*}{\not\Rightarrow} \varepsilon$ and $k_2 < k_1$. Thus, the following must be a production of grammar G:

$$X_1 \underset{p_1}{\rightarrow} \mu_2 X_2 v_2, \text{ and } \mu_2 \overset{*}{\Rightarrow} \varepsilon$$

That is, we get the following result from Step 1 of Algorithm θ:

$$X_1 \underset{p_1}{\rightarrow} \mu_2 X_2 v_2, \text{ and } \mu_2 \overset{*}{\Rightarrow} \varepsilon$$
$$\longleftrightarrow FT_1^1(X_1, X_2) \ni [\ \ ]p_1(p_1), \text{ and}$$
$$FT_2^1(X_2, {}^{(1)}v_2) \ni [\ \ ]p_1(p_1) \text{ if } X_2 \in \Sigma.$$

The above procedure can be expressed in the general form as follows:

$$X_i \underset{p_i}{\Rightarrow} \mu_{i+1} X_{i+1} v_{i+1} \overset{*}{\Rightarrow} X_{i+1} v_{i+1} \overset{k_{i+j}}{\Rightarrow} ab\eta_{i+j}, \; i=0, 1, \cdots$$

where $X_{i+1} \overset{*}{\not\Rightarrow} \varepsilon$ and $k_{i+1} < k_i$. Here, $X_0 = A$ and $p_0 = p$. Thus, the following holds:

$$X_i \underset{p_i}{\rightarrow} \mu_{i+1} X_{i+1} v_{i+1}, \text{ where } v_{i+1} \overset{*}{\Rightarrow} \varepsilon$$
$$\longleftrightarrow FT_1^1(X_i, X_{i+1}) \ni [\ \ ]p_i(p_i), \text{ and}$$
$$FT_2^1(X_{i+1}, {}^{(1)}v_{i+1}) \ni [\ \ ]p_i(p_i) \text{ if } X_{i+1} \in \Sigma. \qquad (3)$$

Since all derivations are done in finite steps, for some integer i the following holds:

$$X_{i+1} v_{i+1} = a v_{i+1}. \qquad (4)$$

In addition, for some integer $j (\geq 1)$ the following holds:

$$a v_{i+1} \overset{k_{i+j}}{\Rightarrow} a X_{i+j} v_{i+j} = ab\eta_{i+j}. \qquad (5)$$

By (3), (4), (5), Step 2 and Step 8 of Algorithm θ the following relation is obtained:

$$\text{A} \underset{p}{\Rightarrow} \alpha \overset{*}{\Rightarrow} abv \longleftrightarrow FT_1^{1/2}(A, a) \ni [\ \ ]p(p), \text{ and}$$
$$FT_2^{1/2/8}(a, b) \ni [\ \ ]p(p).$$

Thus, the following relation equivalent to that of Case (i) holds:

$$S' \overset{*}{\Rightarrow} uAv \underset{p}{\Rightarrow} u\alpha v \overset{*}{\Rightarrow} uab\xi v$$
$$\longleftrightarrow FT_1^{1/2}(A, a) \ni [\ \ ]p(p), \text{ and}$$
$$FT_2^{1/2/8}(a, b) \ni [\ \ ]p(p).$$

Proof of Case (ii)

The derivation $S' \overset{*}{\Rightarrow} uAv$ (where $v \in (N \cup \Sigma)^+$) can be decomposed in detail:

$$S' \overset{*}{\Rightarrow} vXy \underset{q}{\Rightarrow} v\beta Av \overset{*}{\Rightarrow} uAv \underset{p}{\Rightarrow} u\alpha v$$
$$\overset{*}{\Rightarrow} uav \overset{*}{\Rightarrow} uab\zeta.$$

It is represented that, after applying Step 8 of Algorithm θ, the relation between this derivation and FT_1 and FT_2 is given as follows:

$$S' \overset{*}{\Rightarrow} vXy \underset{q}{\Rightarrow} v\beta Av \overset{*}{\Rightarrow} uAv \underset{p}{\Rightarrow} u\alpha v$$
$$\overset{*}{\Rightarrow} uav \overset{*}{\Rightarrow} uab\zeta$$
$$\longleftrightarrow FT_1^{1/2}(A, a) \ni [\ \ ]p(p), \text{ and}$$
$$FT_2^{7/8}(a, b) \ni [{}^{(1)}v]p(q).$$

In order to prove the above relation, the next several preparatory proofs should be given:

Case (ii-a)

First, it is shown that the following relation holds:

$$X_i \underset{p_i}{\Rightarrow} \alpha_{i+1} X_{i+1} \beta_{i+1} \overset{*}{\Rightarrow} u_{j-1} X_{j-1} \xi_{j-1} \underset{p_{j-1}}{\Rightarrow} u_{j-1} \alpha_j X_j \xi_j$$
$$\longleftrightarrow EF^{4/5}(X_j, X_i) \ni [{}^{(1)}\xi_j] p_{j-1}(p_i) \qquad (6)$$

where $u_{j-1} \in \Sigma^*$, $\beta_{i+1} \overset{*}{\Rightarrow} \varepsilon$, $\xi_{j-1} \overset{*}{\Rightarrow} \varepsilon$, $\xi_j = \beta_j \beta_{j+1} \cdots \beta_{i+2} \beta_{i+1} \overset{*}{\Rightarrow} \varepsilon$. Further if $\xi_j = \varepsilon$, $X_j \in \Sigma$, and $\alpha_j^{(1)} \in N$,

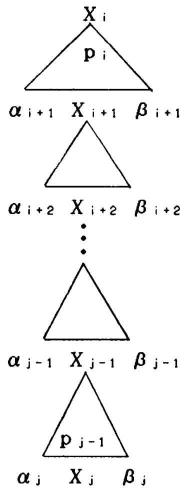

Fig. 1 If $\xi_j = \beta_j \beta_{j-1} \cdots \beta_{i+1} \overset{*}{\Rightarrow} \varepsilon$, then $[{}^{(1)}\xi]p_{j-1}(p_i)$ is added to $EF(X_j, X_i)$. However, if $\beta_j = \varepsilon$, $X_j \in \Sigma$, and $\alpha_j^{(1)} \in N$, then $[X_j]p_{j-1}(p_{j-1})$, instead of $[{}^{(1)}\xi]p_{j-1}(p_i)$, is added to $EF(X_j, X_i)$. Here $\alpha_j^{(1)}$ denotes the rightmost symbol of string α_j.

then $[^{(1)}\xi_j]$ is replaced with $[X_j]$. Here, the notation $\alpha_j^{(1)}$ denotes the rightmost symbol of the string α_j. (cf. Fig. 1). Since all derivations are done in finite steps, if the number of derivation steps of (6) is $k_0(=k+2)$, then we get the following:

$$X_i \underset{p_i}{\Rightarrow} \alpha_{i+1}X_{i+1}\beta_{i+1} \overset{k}{\Rightarrow} u_{j-1}X_{j-1}\xi_{j-1} \underset{p_{j-1}}{\Rightarrow} u_{j-1}\alpha_j X_j \xi_j.$$

In order that the derivation $X_i \underset{p_i}{\Rightarrow} \alpha_i X_{i+1}\beta_{i+1}$ holds, the grammar G must have the following production:

$$X_i \underset{p_i}{\rightarrow} \alpha_{i+1}X_{i+1}\beta_{i+1}$$

where $\beta_{i+1} \overset{*}{\Rightarrow} \varepsilon$. Thus, from Step 4 of Algorithm θ, we get the following:

$$X_i \underset{p_i}{\rightarrow} \alpha_{i+1}X_{i+1}\beta_{i+1} \longleftrightarrow EF^4(X_{i+1}, X_i) \ni [^{(1)}\beta_{i+1}]p_i(p_i) \qquad (7)$$

where $\beta_{i+1} \overset{*}{\Rightarrow} \varepsilon$. Furthermore, for the derivation from X_{i+1}, we also obtain the following derivation:

$$X_{i+1} \underset{p_{i+1}}{\Rightarrow} \alpha_{i+2}X_{i+2}\beta_{i+2}$$
$$\overset{k_1}{\Rightarrow} u'_{j-1}X_{j-1}\xi_{j-1} \underset{p_{j-1}}{\Rightarrow} u'_{j-1}\alpha_j X\xi'_j$$

where $\beta_{i+2} \overset{*}{\Rightarrow} \varepsilon$ and $k_1 < k_0$. Similarly, the grammar G must have the following production because of $X_{i+1} \underset{p_{i+1}}{\Rightarrow} \alpha_{i+2}X_{i+2}\beta_{i+2}$:

$$X_{i+1} \underset{p_{i+1}}{\rightarrow} \alpha_{i+2}X_{i+2}\beta_{i+2}$$

where $\beta_{i+2} \overset{*}{\Rightarrow} \varepsilon$. Then, from the relation between the above production and Step 4 of Algorithm θ, we get the following:

$$X_{i+1} \underset{p_{i+1}}{\rightarrow} \alpha_{i+2}X_{i+2}\beta_{i+2}$$
$$\longleftrightarrow EF^4(X_{i+2}, X_{i+1}) \ni [^{(1)}\beta_{i+2}]p_{i+1}(p_{i+1}) \qquad (8)$$

where $\beta_{i+2} \overset{*}{\Rightarrow} \varepsilon$. In like manner, the following is obtained:

$$X_{i+2} \underset{p_{i+2}}{\rightarrow} \alpha_{i+3}X_{i+3}\beta_{i+3}$$
$$\longleftrightarrow EF^4(X_{i+3}, X_{i+2}) \ni [^{(1)}\beta_{i+3}]p_{i+2}(p_{i+2}) \qquad (9)$$

where $\beta_{i+3} \overset{*}{\Rightarrow} \varepsilon$. Thus, in general, the following representation is valid:

$$X_{i+m} \underset{p_{i+m}}{\rightarrow} \alpha_{i+m+1}X_{i+m+1}\beta_{i+m+1}$$
$$\longleftrightarrow EF^4(X_{i+m+1}, X_{i+m}) \ni [^{(1)}\beta_{i+m+1}]p_{i+m}(p_{i+m})$$

where $\beta_{i+m+1} \overset{*}{\Rightarrow} \varepsilon$. Since all derivations are done in finite steps, for some integer m the following holds:

$$X_{i+m+1} = X_{j-1}.$$

Thus, X_j can be denoted as follows:

$$X_i \overset{k_m}{\Rightarrow} u_{j-1}X_{j-1}\xi_{j-1} \underset{p_{j-1}}{\Rightarrow} u_{j-1}\alpha_j X_j \xi_j$$

where $\xi_j = \beta_j \xi_{j-1} \overset{*}{\Rightarrow} \varepsilon$ and $k_m < k_{m-1}$. Thus, the following holds:

$$X_{j-1} \underset{p_{j-1}}{\rightarrow} \alpha_j X_j \beta_j \longleftrightarrow EF^4(X_j, X_{j-1}) \ni [^{(1)}\beta_j]p_{j-1}(p_{j-1}) \qquad (10)$$

where $\beta_j \overset{*}{\Rightarrow} \varepsilon$. By the procedure of Step 5 of Algorithm θ, we get the following from (7) and (8):

$$X_i \underset{p_i}{\Rightarrow} \alpha_{i+1}X_{i+1}\beta_{i+1} \overset{*}{\Rightarrow} u_{i+1}X_{i+1}\beta_{i+1}$$
$$\underset{p_{i+1}}{\Rightarrow} u_{i+1}\alpha_{i+2}X_{i+2}\xi_{i+2}$$
$$\longleftrightarrow EF^{4/5}(X_{i+2}, X_i) \ni [^{(1)}\xi_{i+2}]p_{i+1}(p_i) \qquad (11)$$

where $u_{i+1} \in \Sigma^*$ and $\xi_{i+2} = \beta_{i+2}\beta_{i+1} \overset{*}{\Rightarrow} \varepsilon$. In additioin, by the procedure of Step 5 of Algorithm θ, the following holds from (11) and (9):

$$X_i \overset{*}{\Rightarrow} u_{i+2}X_{i+2}\xi_{i+2} \underset{p_{i+2}}{\Rightarrow} u_{i+2}\alpha_{i+3}X_{i+3}\xi_{i+3}$$
$$\longleftrightarrow EF^{4/5}(X_{i+3}, X_i) \ni [^{(1)}\xi_{i+3}]p_{i+2}(p_i)$$

where $u_{i+2} \in \Sigma^*$ and $\xi_{i+3} = \beta_{i+3}\beta_{i+2}\beta_{i+1} \overset{*}{\Rightarrow} \varepsilon$. By the repetition of the application of Step 5 of Algorithm θ, we get the following result:

$$X_i \underset{p_i}{\Rightarrow} \alpha_{i+1}X_{i+1}\beta_{i+1} \overset{*}{\Rightarrow} u_{j-1}X_{j-1}\xi_{j-1} \underset{p_{j-1}}{\Rightarrow} u_{j-1}\alpha_j X_j \xi_j$$
$$\longleftrightarrow EF^{4/5}(X_j, X_i) \ni [^{(1)}\xi_j]p_{j-1}(p_i) \qquad (12)$$

where $\xi_j = \beta_j \xi_{j-1} = \beta_j \beta_{j-1} \cdots \beta_{i+1} \overset{*}{\Rightarrow} \varepsilon$. In the production of (6.7), if $\beta_{i+1} = \varepsilon$, $X_{i+1} = a \in \Sigma$, and $\alpha_{i+1}{}^{(1)} \in N$, we get the following by applying Step 4 of Algorithm θ:

$$X_i \underset{p_i}{\rightarrow} \alpha_{i+1}a \longleftrightarrow EF^4(a, X_i) \ni [a]p_i(p_i). \qquad (13)$$

Therefore, if $\beta_j = \varepsilon$, $X_j \in \Sigma$, and $\alpha_j^{(1)} \in N$, then $[^{(1)}\xi_j]$ $p_{j-1}(p_i)$ in (12) is replaced with $[X_j]p_{j-1}(p_{j-1})$.

Case (ii-b)

Secondly, it is shown that the following holds using the result of Case (ii-a):

$$A \underset{q}{\rightarrow} \delta_i X_i \gamma_i \text{ and } X_i \overset{*}{\Rightarrow} u_{j-1}X_{j-1}\gamma' \underset{p_{j-1}}{\Rightarrow} u_{j-1}\delta_j X_j \gamma$$
$$\longleftrightarrow PL_1^{3/6}(X_j, {}^{(1)}\gamma_i) \ni [^{(1)}v]p_{j-1}(q), \text{ and}$$
$$PL_2^6({}^{(1)}\gamma_i, {}^{(2)}\gamma_i) \ni [^{(1)}v]p_{j-1}(q) \text{ if } {}^{(1)}\gamma_i \in \Sigma \qquad (14)$$

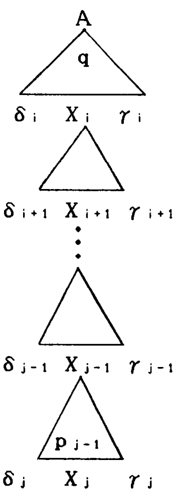

Fig. 2 If $\gamma = \gamma_j \gamma_{j-1} \cdots \gamma_{i+1} \overset{*}{\Rightarrow} \varepsilon$, then $[^{(1)}v]p_{j-1}(q)$ is added to $PL_1(X_j, {}^{(1)}\gamma_i)$. Further if ${}^{(1)}\gamma_i \in \Sigma$ then $[^{(1)}v]p_{j-1}(q)$ is added to $PL_2({}^{(1)}\gamma_i, {}^{(2)}\gamma_i)$, where $v = \gamma\gamma_i$.

where $\gamma=\gamma_j\gamma'=\gamma_j\gamma_{j-1}\cdots\gamma_{i+1}\overset{*}{\Longrightarrow}\varepsilon$, $v=\gamma\gamma_i$, and $u_{j-1}\in\Sigma^*$. (cf. Fig. 2). Clearly, the following is valid from Step 3 of Algorithm θ:

$$A\underset{q}{\longrightarrow}\delta_i X_i\gamma_i$$
$$\longleftrightarrow PL_1^3(X_i,\ ^{(1)}\gamma_i)\ni[^{(1)}\gamma_i]q(q),\text{ and}$$
$$PL_2^3(^{(1)}\gamma_i,\ ^{(2)}\gamma_i)\ni[^{(1)}\gamma_i]q(q)$$
$$\text{if }^{(1)}\gamma_i\in\Sigma\text{ and }^{(2)}\gamma_i\neq\varepsilon. \quad (15)$$

By applying Step 6 of Algorithm θ to (6) and (15), we can obtain the result (14). Finally, using the result of Case (ii-b) it is shown that the relation described in the earlier part of (ii) is valid. From Case (ii-b), the following holds:

$$X\underset{q}{\longrightarrow}\beta B\delta,\ B\overset{*}{\Longrightarrow}v_{j-1}B_{j-1}\xi'\underset{p_{j-1}}{\Longrightarrow}v_{j-1}\beta_j A\xi,$$
$$\text{and }\xi\overset{*}{\Longrightarrow}\varepsilon$$
$$\longleftrightarrow PL_1^{3/6}(A,\ ^{(1)}\delta)\ni[^{(1)}v]p_{j-1}(q),\text{ and}$$
$$PL_2^{3/6}(^{(1)}\delta,\ ^{(2)}\delta)\ni[^{(1)}v]p_{j-1}(q)\text{ if }^{(1)}\delta\in\Sigma$$

where $v=\xi\delta$ and $v_{j-1}\in\Sigma^*$. Thus, we can obtain the following:

$$S'\overset{*}{\Longrightarrow}u_iX\gamma\underset{q}{\Longrightarrow}u_i\beta B\delta\gamma\overset{*}{\Longrightarrow}u_jB\delta\gamma\overset{*}{\Longrightarrow}u_kB_{j-1}\xi'\delta\gamma$$
$$\underset{p_{j-1}}{\Longrightarrow}u_k\beta_jA\xi\delta\gamma$$
$$\longleftrightarrow PL_1^{3/6}(A,\ ^{(1)}\gamma)\ni[^{(1)}\zeta]\quad p_{j-1}(q)$$
$$PL_2^{3/6}(^{(1)}\gamma,\ ^{(2)}\gamma)\ni[^{(1)}\zeta]p_{j-1}(q)\text{ if }^{(1)}\gamma\in\Sigma \quad (16)$$

where $\zeta=\xi\delta\gamma$, $\xi\delta\overset{*}{\Longrightarrow}\varepsilon$, and u_j and $u_k\in\Sigma^*$. Now, assume that

$$A\underset{p}{\Longrightarrow}\alpha\overset{*}{\Longrightarrow}a\quad\text{and}\quad\xi\delta\gamma\overset{*}{\Longrightarrow}b\eta$$

in (6.16). Since the derivation $A\overset{*}{\Longrightarrow}a$ is done in finite steps, this derivation can be rewritten as follows:

$$A\underset{p}{\Longrightarrow}\alpha\overset{k}{\Longrightarrow}a$$

where k is a non-negative integer. By using Step 1 and Step 2 of Algorithm θ, we get the following:

$$A\underset{p}{\Longrightarrow}\alpha\overset{k}{\Longrightarrow}a\longleftrightarrow FT_1^{1/2}(A,a)\ni[\quad]p(p). \quad (17)$$

In addition, applying Step 7 of Algorithm θ to (16) and (17), the following can be obtained:

$$S'\overset{*}{\Longrightarrow}u_iX\gamma\underset{q}{\Longrightarrow}u_i\beta B\delta\gamma\overset{*}{\Longrightarrow}u_jB\delta\gamma\overset{*}{\Longrightarrow}u_k\beta_jA\xi\delta\gamma$$
$$\overset{*}{\Longrightarrow}u_mA\xi\delta\gamma\underset{p}{\Longrightarrow}u_m\alpha\xi\delta\gamma\overset{*}{\Longrightarrow}u_m\alpha\xi\delta\gamma$$
$$\longleftrightarrow FT_2^7(a,\ ^{(1)}\gamma)\ni[^{(1)}\zeta]p(q) \quad (18)$$

where $\zeta=\xi\delta\gamma$, and $\xi\delta\overset{*}{\Longrightarrow}\varepsilon$.

Now, the derivation $\xi\delta\gamma\overset{*}{\Longrightarrow}b\eta$ described before can be rewritten as follows:

$$\xi\delta\gamma\overset{*}{\Longrightarrow}\gamma\overset{*}{\Longrightarrow}b\eta.$$

Thus, applying Step 1 and Step 2 of Algorithm θ to the above derivation, we get the following:

$$\gamma\underset{r}{\Longrightarrow}\gamma'\overset{*}{\Longrightarrow}b\eta\longleftrightarrow FT_1^{1/2}(^{(1)}\gamma,b)\ni[\quad]r(r). \quad (19)$$

Applying Step 8 of Algorithm θ to (18) and (19), the following holds:

$$u_ma\xi\delta\gamma\overset{*}{\Longrightarrow}u_ma\gamma\overset{*}{\Longrightarrow}u_mab\eta$$
$$\longleftrightarrow FT_2^{7/8}(a,b)\ni[^{(1)}\zeta]p(q) \quad (20)$$

where $\zeta=\xi\delta\gamma$. Thus, from (16), (17), and (20) we get the following:

$$S'\overset{*}{\Longrightarrow}u_iX\gamma\underset{q}{\Longrightarrow}u_i\beta B\delta\gamma\overset{*}{\Longrightarrow}u_jB\delta\gamma$$
$$\overset{*}{\Longrightarrow}u_mA\xi\delta\gamma\underset{p}{\Longrightarrow}u_m\alpha\xi\delta\gamma$$
$$\overset{*}{\Longrightarrow}u_m\alpha\xi\delta\gamma\overset{*}{\Longrightarrow}u_ma\gamma\overset{*}{\Longrightarrow}u_mab\eta$$
$$\longleftrightarrow FT^{1/2}(A,a)\ni[\quad]p(p),\text{ and}$$
$$FT_2^{7/8}(a,b)\ni[^{(1)}\zeta]p(q)$$

where $\zeta=\xi\delta\gamma$.

Taking into consideration that q is the index of a production causing the appearance of A onto sentential form, and p is the index of the first production to be applied to A for deriving a terminal a, the next relation described in the beginning of (ii) can be obtained:

$$S'\overset{*}{\Longrightarrow}vX\gamma\underset{q}{\Longrightarrow}v\beta Av\overset{*}{\Longrightarrow}uAv$$
$$\underset{p}{\Longrightarrow}u\alpha v\overset{*}{\Longrightarrow}uav\overset{*}{\Longrightarrow}uab\zeta$$
$$\longleftrightarrow FT_1^{1/2}(A,a)\ni[\quad]p(p),\text{ and}$$
$$FT_2^{7/8}(a,b)\ni[^{(1)}v]p(q).$$

Proof of Case (iii)

Next, it is shown that

$$S'\overset{*}{\Longrightarrow}uAv\underset{p}{\Longrightarrow}u\alpha v\overset{*}{\Longrightarrow}uv\overset{*}{\Longrightarrow}uab\psi$$
$$\longleftrightarrow FT_1^{10}(A,a)\ni[^{(1)}v]p,\text{ and}$$
$$FT_2^{10}(a,b)\ni(^{(1)}v]p. \quad (21)$$

In this case, the following hold:

$$A\underset{p}{\Longrightarrow}\alpha\overset{*}{\Longrightarrow}\varepsilon\text{ and }v\overset{*}{\Longrightarrow}ab\psi. \quad (22)$$

The proof of (22) is reduced to either case (i) or (ii). That is, if $v=X_1X_2\cdots X_n$ (where, $X_i\in N\cup\Sigma$), $X_1X_2\cdots X_{i-1}\overset{*}{\Longrightarrow}\varepsilon$, $X_i\overset{*}{\Longrightarrow}\varepsilon$, and $v'=X_{i+1}X_{i+2}\cdots X_n$, then the relation (6.22) is classified to the following two cases of (iii-a) or (iii-b):

$$\text{(iii-a)}\quad X_iv'\underset{p_1}{\Longrightarrow}\zeta_1v'\overset{*}{\Longrightarrow}ab\psi v'$$
$$\longleftrightarrow FT_1^{1/2}(X_i,a)\ni[\quad]p_1(p_1),\text{ and}$$
$$FT_2^{1/2}(a,b)\ni[\quad]p_1(p_1),$$
$$\text{(iii-b)}\quad X_iv'\underset{p_2}{\Longrightarrow}\zeta_2v'\overset{*}{\Longrightarrow}av'\overset{*}{\Longrightarrow}av''\overset{*}{\Longrightarrow}abv'''$$
$$\longleftrightarrow FT_1^{1/2}(X_i,a)\ni[\quad]p_2(p_2),\text{ and}$$
$$FT_2^{7/8}(a,b)\ni[^{(1)}v']p_2(q). \quad (23)$$

The proof of cases (iii-a) and (iii-b) is reduced to the same ones as these of cases (i) and (ii), respectively. Assume that $v=X_1X_2\cdots X_n$, $X_1X_2\cdots X_{i-1}\overset{*}{\Longrightarrow}\varepsilon$, and $^{(i)}v=X_i$ in (21). Then, the proof on $PL_1(A,X_i)$ and $PL_2(X_i,X_{i+1})$ is reduced to the same ones as Case (ii-b). That is, in the following relation (24), if $X_i\in N$ or $X_{i+1}\in N$, then it must be replaced with a terminal, using

Step 9 of Algorithm θ:

$$S' \overset{*}{\Longrightarrow} vX\gamma \underset{q}{\Longrightarrow} uAv$$
$$\longleftrightarrow PL_1^3(A, X_i) \ni [^{(1)}v]q(q), \text{ and}$$
$$PL_2^{3/6}(X_i, X_{i+1}) \ni [^{(1)}v]q(q) \text{ if } {}^{(1)}v \in \Sigma \qquad (24)$$

where $v = X_1X_2 \cdots X_n$ and $X_1 \cdots X_{i-1} \overset{*}{\Longrightarrow} \varepsilon$. That is, applying Step 9 to the cases (iii-a), (iii-b), and the relation (24), we get the following:

$$S' \overset{*}{\Longrightarrow} vX\gamma \underset{q}{\Longrightarrow} uAv \overset{*}{\Longrightarrow} uv \overset{*}{\Longrightarrow} uab\psi$$
$$\longleftrightarrow PL_1^{3/6/9}(A, a) \ni [^{(1)}v]q(q), \text{ and}$$
$$PL_2^{3/6/9}(a, b) \ni [^{(1)}v]q(q). \qquad (25)$$

Further, because of $A \underset{p}{\Longrightarrow} \alpha \overset{*}{\Longrightarrow} \varepsilon$, by applying Step 10 of Algorithm θ to (25), the following result can be obtained:

$$S' \overset{*}{\Longrightarrow} \zeta \underset{q}{\Longrightarrow} uAv \underset{p}{\Longrightarrow} u\alpha v \overset{*}{\Longrightarrow} uv \overset{*}{\Longrightarrow} uab\psi$$
$$\longleftrightarrow FT_1^{10}(A,a) \ni [^{(1)}v]p(\quad), \text{ and}$$
$$FT_2^{10}(a, b) \ni [^{(1)}v]p(\quad). \qquad \text{[Q.E.D]}$$

7. An Example

The tables Fig. 3 through Fig. 10 show the content of tables at each step of Algorithm θ applied to the non-strong semi-LL(2) grammar represented below.

[Example] Non-strong semi-LL(2) grammar

1. $S \rightarrow aAaa$
2. $S \rightarrow bAba$
3. $S \rightarrow Aa$
4. $A \rightarrow b$
5. $A \rightarrow \varepsilon$

	S	A	a	b	$
S		[ ]3(3)	[ ]1(1) [ ]3(3)	[ ]2(2) [ ]3(3)*	
A				[ ]4(4)	
a		[ ]1(1)	[ ]1(1)		
b		[ ]2(2)	[ ]3(3)	[ ]2(2)	
$					

Fig. 3 FIRST-table for G_3 constructed from Step 2 through Step 1 of Algorithm θ. The entry marked with * denotes one entered in Step 2, and the other entries are entered in Step 1.

	S	A	a	b	$
S					[$]0(0)
A			[a]1(1) [a]3(3)	[b]2(2)	
a			[a]1(1)		
b			[b]2(2)		
$					[$]0(0)

Fig. 4 PF-table for G_3 constructed from Step 3 of Algorithm θ.

The sets Q and R for the above sample grammar are given:

$Q = \{(A, 5)\}$

$R = \{(S, a, 3), (A, b, 4)\}$

8. Evaluations

To evaluate the capabilities of Algorithm θ, we compared Algorithm θ with Aho-Ullman's method under the following conditions:

	S	A
S		
A		
a	[a]3(3)	
b		
$		

Fig. 5 END-FOLLOW-table for G_3 constructed from Step 5 through Step 4 of Algorithm θ.

	S	A	a	b	$
S					[$]0(0)
A			[a]1(1) [a]3(3)	[b]2(2)	
a			[a]1(1)		[a]3(3)*
b			[b]2(2)		
$					[$]0(0)

Fig. 6 PF-table for G_3 constructed from Step 6 through Step 3 of Algorithm θ. The entry marked with * denotes one entered in Step 6.

	S	A	a	b	$
S		[ ]3(3)	[ ]1(1) [ ]3(3)	[ ]2(2) [ ]3(3)	
A				[ ]4(4)	
a		[ ]1(1)	[ ]1(1)	[ ]1(1)**	[$]3(0)*
b		[ ]2(2)	[ ]3(3) [a]4(1)* [a]4(3)*	[ ]2(2) [b]4(2)*	
$					

Fig. 7 FIRST-table for G_3 constructed from Step 8 through Step 1 of Algorithm θ. The entries marked with * denote ones entered in Step 7, and the entry marked with ** is entered in Step 8.

	S	A	a	b	$
S					[$]0(0)
A			[a]1(1) [a]3(3)	[b]2(2)	
a			[a]1(1)		[a]3(3)
b			[b]2(2)		
$					[$]0(0)

Fig. 8 PF-table for G_3 constructed from Step 9 through Step 3 of Algorithm θ.

	S	A	a	b	$
S		[ ]3(3)	[ ]1(1) [ ]3(3)	[ ]2(2) [ ]3(3)	
A			[a]5( )*	[ ]4(4) [b]5( )*	
a		[ ]1(1)	[ ]1(1) [a]5( )*	[ ]1(1)	[$]3(0) [a]5( )*
b		[ ]2(2)	[ ]3(3) [a]4(1) [a]4(3) [b]5( )*	[ ]2(2) [b]4(2)	
$					

Fig. 9 FIRST-table for G_3 constructed from Step 10 through Step 1 of Algorithm θ. The entries marked with * denote ones entered in Step 10.

	a	b	$
S	[ ]1 [ ]3	[ ]2 [ ]3	
A	[a]5	[ ]4 [b]5	
a	[ ]1 [a]5	[ ]1	[$]3 [a]5
b	[ ]3 [a]4 [b]5	[ ]2 [b]4	
$			

Fig. 10 Parsing table for G_3. π type production indices are transformed to τ type.

(a) computer used: SUN-3 Model 60M,
(b) OS and language used: UNIX/PASCAL,
(c) sample grammars: PASCAL- (PASCAL minus) [7] and ISO PASCAL [8].
(PASCAL- grammar is a subset of ISO PASCAL.)

All the programs required for parsing table construction are programmed by a single programmer to minimize differences based on the way of programming.

Table 1 The experimental construction time of parsing tables for PASCAL- (time unit: sec).

Algorithm θ (Authors) X		Aho-Ullman Y		X:Y
Calculation for set Q	0.26	Calculation for set FIRST	0.24	
Calculation for set R		Calculation for T_i	128.54	
Step 1 | Step 11	11.61	Conversion of productions		
		Constructing tables		
Total	11.87	Total	128.78	1:11

Table 2 The experimental construction time of parsing tables for ISO PASCAL (time unit: sec).

Algorithm θ (Authors) X		Aho-Ullman Y		X:Y
Calculation for set Q	1.47	Calculation for set FIRST	1.38	
Calculation for set R		Calculation for T_i	2388.55	
Step 1 | Step 11	190.69	Conversion of productions		
		Constructing tables		
Total	192.16	Total	2389.93	1:12

Speed of Table Construction

First, PASCAL- and ISO PASCAL grammars are converted to semi-LL(2) grammars by hand. After this, we made five experiments by Algorithm θ and Aho-Ullman's method, respectively. Each value in Table 1 and Table 2 indicates the average time resulted from these experiments.

For both cases of PASCAL- and ISO PASCAL, the times required by Algorithm θ are about 1/10 or less than those by Aho-Ullman's. Since the parsing table for ISO PASCAL, constructed by Aho-Ullman's method, is too large to store on the memories of the used computer, total time of Aho-Ullman's method in Table 2 does not include the time required to fill table with computed values. Thus, in practice, the total time by Aho-Ullman's method will be larger than indicated on Table 2.

The time required for calculating the sets Q and R is negligible since each value is only 1.8-2.14% of the total values in Table 1 and Table 2. The set Q is calculated by using the algorithm proposed in the article [6]. The set R can be calculated easily by using an algorithm similar to one for set Q.

Memory Area Required for Constructing Tables

Memory space required in constructing tables can be divided roughly into two parts, namely, the code portion and table portion. The latter is greatly subjected to the grammar to be processed, but the former is not.

Tables to be considered are a parsing table, a production table, and some more tables used temporarily in the construction of the parsing table. The temporary tables used by authors are a part of FIRST-table, PF-table, and END-FOLLOW-table etc. While the size of parsing table, in the case of authors' method, is decided by the numbers of non-terminals and terminals of the grammar to be processed, in the case of Aho-Ullman's method, it is decided by the number of newly generated non-terminals, T_i, and terminals. (cf. Fig. 11 and Fig. 12). Table 3 illustrates the number of nonterminals and productions increase to 8 and 16 times the original number in both cases of PASCAL- and ISO PASCAL grammars, respectively.

Table 4 indicates the actual memory area occupied by parsing tables constructed for PASCAL- and ISO PASCAL grammars. An entry of the parsing table by authors may include two or more τ type production indices, which is implemented by a linked-list using pointers.

The values on the row I of Table 4 include memory area used for these pointers, and also they include memory area required for the parsing table, the productions, and all temporary tables used for constructing parsing table.

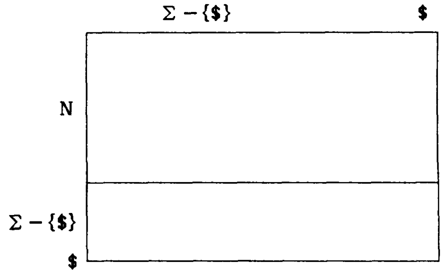

Fig. 11 The structure of authors' parsing table. Rows and Columns are labeled with elements of the sets $(N\cup(\Sigma-\{\$\})\cup\{\$\})$ and $((\Sigma-\{\$\})\cup\{\$\})$, respectively.

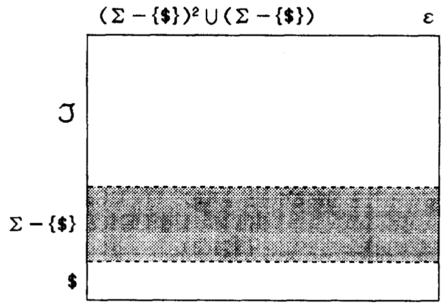

Fig. 12 The structure of Aho and Ullman's parsing table. We can't construct the oblique portion of this figure as a part of array because authors' computer does not have enough memory size, but we can code this part of tables in the parsing steps. Rows and Columns are labeled with elements of the sets $(\mathcal{T}\cup(\Sigma-\{\$\})\cup\{\$\})$ and $((\Sigma-\{\$\})^2\cup(\Sigma\cup\{\$\})\cup\{\varepsilon\})$, respectively. $(\Sigma-\{\$\})^2$ denotes a set of terminal string of length 2.

Table 3 Actual numbers of production rules, terminals, and non-terminals resulted from rewriting Wirth's production rules for applying Aho's algorithm (in the cases of PASCAL- and ISO PASCAL).

	PASCAL- grammar				ISO PASCAL grammar			
	semi-LL(2) X_1	Authors	Aho-Ullman Y_1	X_1:Y_1	semi-LL(2) X_2	Authors	Aho-Ullman Y_2	X_2:Y_2
Numbers of productions	98	98	804	1:8	247	247	3997	1:16
Numbers of nonterminals	54	54	427	1:8	151	151	2369	1:16
Numbers of terminals	46	46	46		61	61	61	

Table 4 Actual memory area required in the construction of parsing tables for PASCAL- and ISO PASCAL (memory unit: byte).

	PASCAL- grammar			ISO PASCAL grammar		
	Authors X_1	Aho-Ullman Y_1	X_1:Y_1	Authors X_2	Aho-Ullman Y_2	X_2:Y_2
Memory for code (A)	65560	60896	1:0.93	65560	60896	1:0.93
Tables I. all tables* (B)	161552**	7530080		558896**	71947144	
Tables II. a parsing table and productions***	60228	7439496	1:124	179732	71805620	1:400
Total (A)+(B)	227112	7590976	1:33	624456	72008040	1:115

*The values on the row consist of memory area for the parsing table, production-rules, and all the tables required for constructing a parsing table.

**In authors' parsing table, it is possible to enter multiple τ type production indices into a single cell of tables. In such the case, it is implemented by a linked list linked from the cell. The values marked with ** involve memory area for these linked lists.

***The values on this row are for only the memory area of parsing table and production-rules used for parsing.

As far as a parsing table and productions are concerned, memory area required by authors' method is reduced to about 1/120 to 1/400 of Aho-Ullman's. This tendency of the decrease in memory area will be presumed for other programming languages, also.

The memory area for code portion by authors' method is slightly larger than Aho-Ullman's. However, because the ratio of the code portion to the whole required memory area is about 10% to 20%, it does not matter.

In the case of Aho-Ullman, the more the size of grammar becomes large, the more rapidly the number of converted productions and converted nonterminals increases. As illustrated in Table 3, in the case of productions, comparing the number of original productions of PASCAL- with that of ISO PASCAL, the latter is about two times and a half of the former. After the conversion, however, this ratio becomes to about five times. Furthermore, in the case of nonterminals, although the ratio of number of original ones is about two point and eight, after the conversion, this ratio becomes to about five and a half. That is, by the conversion, the ratio increases to two times the original one in both cases of nonterminals and productions. As illustrated in Fig. 12, Aho-Ullman's table size varies in proportion to the number of converted nonterminals. Thus, it is guessed that Aho-Ullman's table becomes rapidly larger than the authors' when the size of grammar becomes larger.

9. Conclusions

As mentioned in the introduction, there are many difficulties in constructing parsing tables of LL(k) grammars, $k \geqq 2$. One of these is that the constructing method itself is very complicated. Another one is that the very large area of memory is needed for the construction.

This paper proposed a constructing algorithm of parsing tables in the case of $k=2$ of semi-LL(k) grammars that is slightly restricted LL(k) grammars, and showed the validity of the algorithm. Moreover, we compared results by our algorithm with ones by Aho-Ullman's method using PASCAL- and ISO PASCAL grammars. The former's table-construction time is about 1/10 of that of the latter, and the memory area for parsing table and productions is about 1/120 to 1/400 of that of the latter.

Furthermore, from its property, Aho-Ullman's method is necessary to convert productions, by which the number of productions increases into 8–16 times the original one.

The above results of comparison are guessed to hold generally for all grammars other than ones used as the examples, from the property of Aho-Ullman's method.

Although Aho-Ullman's method has an advantage to be applied not only to semi-LL(2) grammars but also to LL(2) grammars, Algorithm θ is practically quite useful because it is easy to implement and most of LL(2) grammars can be converted into semi-LL(2) grammars.

The outline of a parsing method, which is based on the parsing table constructed by Algorithm θ proposed in this paper, is given in Appendix B. The detail of the parsing algorithm is given in the article [1]. Moreover, the properties of semi-LL(k) grammars are given in the article [1], also.

Acknowledgments

Many thanks are due to Prof. Kenzo INOUE at Science Univ. of Tokyo for his useful comments and help. Without his advice this paper would be far from complete.

Thanks also to the referees for their useful comments.

References

1. Yoshida, K. and Takeuchi, Y. The structure of the Parsing Table and the Parsing Algorithm for Semi-LL(2) Grammars (in Japanese), *Trans. IPS Japan*, **31**, 9 (Sep. 1990), 1354–1365.
2. Aho, A. V. and Ullman, J. D. The Theory of Parsing, Translation, and Compiling, I. Prentice-Hall (1972), 348–356.
3. Lewis, P. M. and Stearns, R. E. Compiler Design Theory, Addison-Wesley (1976), 262–276.
4. Yoshida, K. and Takeuchi, Y. An Algorithm for Constructing LL(1) Parsing Table Using Production Indices (in Japanese), *Trans. IPS Japan*, **27**, 11 (Nov. 1986), 1095–1105.
5. Backhouse, R. C. Syntax of Programming Languages (Theory and Practice), Prentice-Hall (1979), 92–152.
6. Yoshida, K. and Takeuchi, Y. Some Properties of an Algorithm for Constructing LL(1) Parsing Tables Using Production Indices, *J. Inf. Process.*, **11**, 4 (Mar. 1988), 258–262.
7. Hansen, P. B. Brinch Hansen on Pascal Compilers, Prentice-Hall (1985), 15–16.
8. Jensen, K. and Wirth, N. PASCAL-User Manual and Report (3rd ed.), ISO PASCAL standard, Springer-Verlag (1975), 215–220.

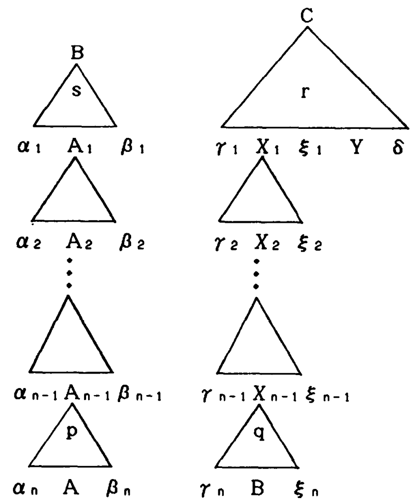

Fig. Appendix A

In this case, if $\mathrm{EF}(A, B) \ni [X]p(s)$ and $\mathrm{PL}_1(B, Y)$

$\ni [Z]q(r)$, then $[V]p(r)$ is added to $PL_1(A, Y)$. In addition, if $Y \in \Sigma$ then $[V]p(r)$ is added to $PL_2(Y, W)$, where $W = {}^{(1)}\delta$, $\beta_n\beta_{n-1}\cdots\beta_1 \overset{*}{\Rightarrow} \varepsilon$, and $\xi_n\xi_{n-1}\cdots\xi_1 \overset{*}{\Rightarrow} \varepsilon$. X denotes the leftmost symbol of a string $\beta_n\beta_{n-1}\cdots\beta_1$, and Z the leftmost symbol of a string $\xi_n\xi_{n-1}\cdots\xi_1$. If $X \neq \varepsilon$ then $V = X$, otherwise $V = Z$.

Appendix B

We describe the outline of a parsing algorithm based on the parsing table proposed in this paper. The detailed algorithm is discussed in the article [1].

First, a stack, an array, another region, variables, symbols, and so on used in parsing, are described:

(1) R: stack storing parsing history.

(2) M: one dimensioned array storing a program text.

(3) $CURRENT_1$ and $CURRENT_2$: variables storing the terminal symbol looked currently on the text for parsing and the terminal symbol being right adjacent to the symbol in $CURRENT_1$ on the text, respectively.

(4) TOP and NEXT: symbols indicating the top symbol and the second symbol from the top in the stack R, respectively.

(5) i: positive integer used as the subscript of $M(i)$, the i-th element, of array M.

(6) Y: variable storing production index.

Next, several actions executed in parsing are described:

(i) POP: taking out the symbol in TOP from stack R. The length of R is reduced by its one element, TOP is replaced by NEXT, and the next of NEXT becomes NEXT.

(ii) PUSH(p): Stack R is extended toward the top by the length of righthand side of the production with index p and the right-hand side is stored on the extended part of R, such that the leftmost symbol of the righthand side comes onto new TOP.

Finally, set operation $\bar{\cap}$,

$U = T(TOP, CURRENT_1) \bar{\cap} T(CURRENT_1, CURRENT_2)$,

is defined as follows.

[Definition]

For any $A \in N$, a and $b \in \Sigma$, and $T(X, Y)$, set operation $\bar{\cap}$ is defined by:

① If $([\ \ ]p \in T(A, a)) \wedge ([\ \ ]p \in T(a, b))$, then $[\ \ ]p \in U$.

② If $([\ \ ]p \in T(A, a)) \wedge ([X]p \in T(a, b))$, then $[\ \ ]p \in U$.

③ If $([X]p \in T(A, a)) \wedge ([X]p \in T(a, b))$, then $[\ \ ]p \in U$.

④ Otherwise, $U = \phi$.

The value of set U is classified into three cases:

(a) Case $|U| = 0$.

$|U| = 0$ means $U = \phi$. In this case, there is no production to be applied. Thus, the parser concludes that the program text is wrong.

(b) Case $|U| = 1$.

For example, if $U = \{[\ \ ]p\}$ then production p should be applied regardless of the symbol in NEXT. If $[\ \ ]$ has any symbol, for example, $[X]p$, then the parser must decide whether it selects production p or not depending on NEXT.

(c) Case $|U| \geqq 2$.

① Case $[\ \ ]p \in U$.

In this case, from (i) and (ii) of Theorem described later, U becomes as follows:

$$U = \{[\ \ ]p, [X_1]p_1, [X_2]p_2, \cdots, [X_n]p_n\},$$

where $n \geqq 2$. This means that there exist the following derivations:

$$S' \overset{*}{\Rightarrow} u_1A\gamma_1 \underset{p}{\Rightarrow} u_1\alpha\gamma_1 \overset{*}{\Rightarrow} u_1ab\zeta\gamma_1$$
$$S' \overset{*}{\Rightarrow} u_2A\gamma_2 \underset{p}{\Rightarrow} u_2\alpha\gamma_2 \overset{*}{\Rightarrow} u_2a\gamma_2 \overset{*}{\Rightarrow} u_2ab\gamma_2'$$
$$S' \overset{*}{\Rightarrow} u_3A\gamma_3 \underset{p}{\Rightarrow} u_3\alpha\gamma_3 \overset{*}{\Rightarrow} u_3\gamma_3 \overset{*}{\Rightarrow} u_3ab\gamma_3',$$

and so on. However, since all the above derivations begin with production $A \rightarrow \alpha$, the parser selects production p as the first step of the derivations.

② Case $[\ \ ]p \notin U$.

In this case, from (iii) of Theorem, U becomes as follows:

$$U = \{[X_1]p_1, [X_2]p_2, \cdots, [X_n]p_n\},$$

where $n \geqq 2$, $X_i \neq X_j$, and $1 \leqq i, j \leqq n$. If NEXT $= X_i$ then there is no production applying to TOP except p_i.

③ If parsing is neither case ① nor ②, from Theorem the parser concludes that the program text is wrong.

[Theorem]

Let a grammar be semi-LL(2). If set U has two or more elements, then none of the following cases hold:

(i) $[\ \ ]p$, $[\ \ ]q$, $p \neq q$

(ii) $[\ \ ]p$, $[X]q$, $p \neq q$

(iii) $[X]p$, $[X]q$, $p \neq q$

where $U = T(A, a) \bar{\cap} T(a, b)$.

Parsing Algorithm

```
begin
  R←'S$$'; /* Initialization of stack R */
  M←text '$$'; /* Initialization of array M */
  i←1;
  CURRENT1←M(i);
  CURRENT2←M(i+1);
    repeat
      if TOP=CURRENT1 and NEXT=CURRENT2
      then
        begin
          POP; POP;
          i←i+2;
```

```
        CURRENT1←M(i);
        CURRENT2←M(i+1);
      end
    else
      if TOP=CURRENT1 then
        begin
          POP;
          i←i+1;
          CURRENT1←CURRENT2;
          CURRENT2←M(i+1);
        end
      else
        if TOP∈Σ then text error
        else
          begin
            find U such that
            U=T(TOP, CURRENT1) ∩̄
                  T(CURRENT1, CURRENT2);
            case
              |U|=0: text error;
              |U|=1:
                if there exists [   ]p in U then
                  select p and Y←'p'
                else
                  if there exists [NEXT]p in U then
                    select p and Y←'p'
                  else text error;
              |U|≧2:
                if there exists [   ]p in U then
                    select p and Y←'p'
                else
                  if there exists [NEXT]pi in U then
                    select pi and Y←'pi'
                  else text error;
            end of case;
            POP; PUSH(Y);
          end
  until TOP='$' and CURRENT1='$'
end.
```

[End of Algorithm]