Some Properties of an Algorithm for Constructing LL(1) Parsing-Tables Using Production Indices

KEIICHI YOSHIDA* and YOSHIKO TAKEUCHI**

This paper reveals five valuable properties of Algorithm H discussed in the paper [2]. Algorithm H can construct parsing-tables for LL(1) grammars only by table-handling using the indices given to the productions of the grammars without set-calculations, which are needed by conventional methods.

Furthermore, based on the property revealed in this paper, it has been shown that Algorithm H can be revised into an algorithm to point out the non-LL(1)ness of input grammars.

1. Introduction

If taking into account the fact that Pascal is a good and practical example of the grammar which can be written in LL, the usefulness of LL grammar is underestimated just because it is a subset of LR. In addition, the latest article [1] reports that an LL parser generator, LLgen, was developed, that it generated a new C compiler, and that it is being used in a natural-language parser project.

We proposed an algorithm for constructing LL(1) parsing-tables using production indices in the paper [2]. Outstanding features of our algorithm, that no other algorithm has, are:

1. Implementation is very easy. Conventional methods[3]~[5] need set-calculations to find the values for FIRST and/or FOLLOW, and then construct parsing-tables with these values. Our algorithm can construct parsing-tables without set-calculations, but only by table-handling using the indices given to the productions.
2. The time spent to construct parsing-tables can be reduced to 1/100, and the required memory size saved to 1/3 through 1/4 compared with the conventional methods.

Conventional algorithms can detect non-LL(1)ness by either testing if following formula is true:

$$\mathrm{FIRST}_1(\beta\mathrm{FOLLOW}_1(A)) \cap \mathrm{FIRST}_1(\gamma\mathrm{FOLLOW}_1(A))=\phi$$

where $A\rightarrow\beta$ and $A\rightarrow\gamma$ are distinct A-production, or checking whether multi-definitions occur for the same entry in constructing parsing-table. On the other hand, it is not necessary for algorithm to apply the above mentioned formula since our algorithm makes parsing-table directly. Both conventional algorithms and ours can detect non-LL(1)ness by multi-definitions. But our process to detect non-LL(1)ness is different from that of conventional algorithms because their ways of constructing parsing-tables are different.

This paper describes five properties of our algorithm, and by the properties our algorithm can be easily revised into an algorithm to point out non-LL(1)ness for a CFG. For convenience, we name our new algorithm Algorithm H. It does not require the grammar to be in the precise form of LL(1).

2. Symbols and Definitions

According to conventions, Σ, and N denote a set of terminals, a set of nonterminals of a given CFG G, respectively. A, B, and C are elements in N; X, Y, and Z are elements in $N\cup\Sigma$; a, b, and c are elements in Σ; s and t are elements in Σ^*; α, β, γ, and δ are elements in $(N\cup\Sigma)^*$ unless otherwise specified. Each symbol is able to have a subscript if necessary. ε, and ϕ are a null string, and an empty set, respectively. Finally, p in $A \underset{p}{\rightarrow} \alpha$ or $\alpha \underset{p}{\Rightarrow} \beta$ denotes an index of the corresponding production. All derivations in this paper are done by the leftmost. Other concepts and symbols used without any definition are based on the paper [3].

Definitions for FIRST, FOLLOW, and END-FOLLOW are:

[Definition 1]
FIRST of X is defined by:

$$\mathrm{FIRST}(X)=\{a \mid a\in\Sigma,\ X \overset{*}{\Rightarrow} a\alpha,\ \alpha\in(N\cup\Sigma)^*\}$$

This definition form is not common, but we use it because all ε-productions are deleted before the calculation of FIRST(X) in Algorithm H and the same definition form can be seen in article [6].

[Definition 2]
FOLLOW of A is defined by:

$\mathrm{FOLLOW}(A)=\{a \mid a\in\Sigma,\ S \overset{*}{\Rightarrow} \alpha A\beta,$ S is the start

*College of Engineering, Shizuoka University, 3-5-1, Jyohoku, Hamamatsu City, Shizuoka Pref., 432, Japan
**Hamamatsu Polytechnic College

symbol of G,
$\beta \overset{*}{\Rightarrow} av,\ v \in (N \cup \Sigma)^*\}$

For constructing an efficient parsing-table, \$ is used as an element of FOLLOW(S), where \$ is an end mark of an input text.

[Definition 3]
END-FOLLOW of B is defined by:

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

In our definition, FOLLOW does not contain ε. In the case of $\beta \overset{*}{\Rightarrow} \varepsilon$ where $S \overset{*}{\Rightarrow} \alpha A \beta$, FOLLOW of A can be found by END-FOLLOW, FOLLOW, and FIRST. For example, in the following derivation

$$S \overset{*}{\Rightarrow} t_1 B X \overset{*}{\Rightarrow} t_2 A \beta X,\quad \beta \overset{*}{\Rightarrow} \varepsilon$$

FOLLOW of A can be calculated by using END-FOLLOW(A), FOLLOW(B), and FIRST(X).

[Definition 4]
P and Q are defined by:

P = {p | p is a unique index for a production represented by a positive integer}

$$Q = \{(A, p) \mid A \underset{p}{\Rightarrow} \alpha \overset{*}{\Rightarrow} \varepsilon\}$$

3. Algorithm

This section describes algorithm for detecting non-LL(1)ness. Before going further, a brief definition to it is provided.

If $A \to \beta$ and $A \to \gamma$ are distinct A-productions and the following condition holds, then it is called that the grammar with such the production has non-LL(1)ness:

$$\text{FIRST}_1(\beta \text{FOLLOW}_1(A)) \cap \text{FIRST}_1(\gamma \text{FOLLOW}_1(A)) \neq \phi.$$

When a grammar has non-LL(1)ness, two or more production indices are defined for the same entry while constructing parsing-table.

It is assumed that all grammars input to Algorithm H do not contain any useless productions. A set Q of (A, p) such that $A \underset{p}{\Rightarrow} \alpha \overset{*}{\Rightarrow} \varepsilon$ is required in Algorithm H, and the set could be calculated by R. Hunter's Algorithm [7]. Instead, here, we use Algorithm K developed by us and it has a much simpler procedure described below. Algorithm K works as a preprocessor for Algorithm H.

By close observation on K, it is clear that K is capable of detecting if an input grammar G contains a non-LL(1)ness caused by A such that $A \underset{p_i}{\Rightarrow} \alpha \overset{*}{\Rightarrow} \varepsilon$ and $A \underset{p_j}{\Rightarrow} \beta \overset{*}{\Rightarrow} \varepsilon$, where $\alpha \neq \beta$ and $i \neq j$.

Algorithm K: {finding set Q}

[Step 1] {finding subset of Q, or {(A, p)} such that $A \underset{p}{\Rightarrow} \varepsilon\}$

```
begin
  Q←φ;
  for each A such that A —p→ ε
    Q←Q ∪ {(A, p)};
end
```

[Step 2] {adding {(A, p_i)} such that $A \underset{p_i}{\Rightarrow} \alpha \overset{+}{\Rightarrow} \varepsilon$ to Q}

```
repeat
  for each production such that A —pi→ Yi1Yi2. . .Yin do
    begin
      k←1;
      repeat
        if (Yik, pk)∈ Q
          then delete Yik from the righthand side of pi
          else go to l; {A ⇏* ε because Yik ⇏* ε}
        k←k+1
      until k>n;
      {checking if an element (A, pj) such that A =pj⇒
      α' =*⇒ ε is already in Q, where i≠j}
      if Q∌ (A, pj) such that A =pj⇒ α' =*⇒ ε and i≠j
           then Q←Q ∪ {(A, pi)}
           else begin write 'G is not LL(1)';
                  end of program
                end;
l: end
until no change occurs in Q.
```

Before discussing Algorithm H, we explain some additional symbols used in it.

(1) FIRST: although this symbol is defined to represent a set in section 2, in Algorithm H it is also used as a name of the table corresponding to the set. The rows of the table are named by non-terminals, and the columns are named by both terminals and non-terminals. Each entry of the table has either a value 0 or a member of P(i.e., a production index).

(2) FOLLOW: this symbol, in H, is also defined to represent a name of the table corresponding to the set, FOLLOW. Its table structure is similar to FIRST except for its entries. Each entry of the table is either nil or *. The symbol * indicates that there is FOLLOW-relation between symbols naming the column and row including the entry, but nil indicates that there is no such relation.

(3) END-FOLLOW: this symbol, in H, is also used to represent a name of the table corresponding to the set, END-FOLLOW. The rows and columns of the table are named by nonterminals only, and their entries are either nil or *.

The following is an outline of Algorithm H and its detailed steps.

Algorithm H consists of the following three parts with eight steps.

PART I: Steps for constructing FIRST-table

Step 1: Initializing FIRST-table by the production such that $A \to \alpha (\alpha \neq \varepsilon)$. T_{11} denotes the part of the FIRST-table filled in by Step 1.

Step 2: Computing the closure of entries filled in by Step 1. T_{12} denotes the part of the FIRST-table filled in by Step 2 alone.

Note: $T_1 = T_{11} + T_{12}$

Part II: (II-a) Steps for constructing FOLLOW-table

Step 3: Initializing FOLLOW-table. The part added to the FOLLOW-table by Step 3 alone is referred to F_{11}.
Step 4: Computing the closure of entries filled in the FOLLOW-table by Step 3. The part constructed by Step 4 alone is referred to F_{12}.
Note: $F_1=F_{11}+F_{12}$

(II-b) Steps for constructing END-FOLLOW-table
Step 5: Initializing END-FOLLOW-table. If $A\rightarrow\alpha B\beta$, and $\beta \overset{*}{\Rightarrow} \varepsilon$, then END-FOLLOW(B, A)='*'. Here, END-FOLLOW(B, A) is the entry on the row B and the column A of the table END-FOLLOW. In the following steps, the same notations are used for the tables FIRST and FOLLOW. The part constructed by Step 5 alone are referred to E_1.
Step 6: Computing the closure of the entries constructed by Step 5 alone. The part constructed by Step 6 alone is referred to E_2.

(II-c) Completing the FOLLOW-table using END-FOLLOW-table
Step 7: If END-FOLLOW(A, B)='*' and FOLLOW (B, a)='*' then FOLLOW(A, a) is filled with * for any $a\in\Sigma$, $A\in N$ and $B\in N$. The part of the FOLLOW-table filled in by Step 7 alone is referred to F_2.
Note: $F=F_1+F_2$

Part III: Completing FIRST-table using FOLLOW-table when $A \overset{*}{\Rightarrow} \varepsilon$
Step 8: For any $A\in N$ and $a\in\Sigma$, if FOLLOW(A, a) ='*' then FIRST(A, a) is filled with p such that (A, p) in Q. The part added to FIRST-table by Step 8 alone is referred to T_2. $T=T_1+T_2$. Then, terminal (T), the part on columns of T named by terminals, is the required parsing-table.

It is assumed that all productions are such that $A\rightarrow\varepsilon$ is excluded before applying Algorithm H.

Algorithm H:
Construction of FIRST-table
[Step 1]
{Let every entry of the FIRST-table be 0 before the following operations are performed.}

```
begin
  for each production such that A -p-> Y1Y2. . .Yn do
    begin j←1;
      repeat FIRST(A, Yj)←p;
        j←j+1;
      until (Yj-1, pj-1)∉Q or j>n; {pj-1 is an index
given to a production Yj-1→α}
    end;
  end.
```

[Step 2]

```
  begin
    repeat
      for each A∈N do
          for each B∈N do
            if FIRST(A, B)∈P then
              for each C∈N do
                if FIRST(B, C)∈P then
                  FIRST(A, C)←FIRST(A, B);
    until no change occurs in the FIRST-table;
    for each A∈N do
      for each B∈N do
        if FIRST(A, B)∈P then
          for each a∈Σ do
            if FIRST(B, a)∈P then
              FIRST(A, a)←FIRST(A, B);
    if Q is empty then skip Step 3 through Step 8
  end.
```

Construction of local-FOLLOW-table
[Step 3]
{Let every entry of the FOLLOW-table be nil before the following operations are performed.}

```
  begin
    for each production such that A→Y1Y2. . .Yn do
      begin j←1; k←j+1;
        while j<n do
          begin if Yj∈N then
            repeat FOLLOW(Yj, Yk)←'*';
                k←k+1;
            until (Yk-1, pk-1)∉Q or k>n;
            j←j+1; k←j+1
          end;
        end;
  end.
```

[Step 4]

```
  begin
    for each A∈N do
      for each B∈N do
        if FOLLOW(A, B)='*' then
          for each a∈Σ do
            if FIRST(B, a)∈P then
              FOLLOW(A, a)←'*';
  end.
```

Construction of END-FOLLOW-table
[Step 5]
{Let every entry of the END-FOLLOW-table be nil before the following operations are performed.}

```
  begin
    for each production such that A→Y1Y2. . .Yn do
      j←n;
      repeat if Yj∈N then
          END-FOLLOW(Yj, A)←'*';
        j←j-1;
      until (Yj+1, pj+1)∉Q or j=0;
  end.
```

[Step 6]

```
  begin
    repeat
      for each A∈N do
        for each B∈N do
          if END-FOLLOW(A, B)='*' then
            for each C∈N do
```

if END-FOLLOW(B, C)='*' then
END-FOLLOW(A, C)←'*'
until no change occurs in the END-FOLLOW table
end.

Construction of FOLLOW-table
[Step 7]

```
begin
  FOLLOW(S, $)←'*'{S: start symbol,
                   $: end mark of an input text}
  for each A∈ N do
     fot each B∈ N do
        if END-FOLLOW(A, B)='*' then
           for each a∈ Σ do
              if FOLLOW(B, a)='*' then
                 FOLLOW(A, a)←'*'
end.
```

Construction of Parsing-table
[Step 8]

```
begin
  for each A∈ N do
     if (A, p)∈ Q then
        for each a∈ Σ do
           if FOLLOW(A, a)='*' then
              FIRST(A, a)←p
end.
```

4. Properties of Algorithm H

We explain properties of Algorithm H in this section. Based on its properties, we show Algorithm H can detect non-LL(1)ness for any input grammar CFG.

[Property 1]

When Step 1 is applied to CFG G, if $T_{11}(A, X)=\{p_1, p_2, \ldots, p_n\}$, $n \geqq 2$, then G cannot be LL(1), where $p_i \in P$, $1 \leqq i \leqq n$.

[Proof]

$T_{11}(A, X)=\{p_1, p_2, \ldots, p_n\}$, $n \geqq 2$
$\Longleftrightarrow$ there exist n productions such that $A \xrightarrow[p_1]{} \alpha_1 X\beta_1$, $\alpha_1 \overset{*}{\Longrightarrow} \varepsilon$, ..., $A \xrightarrow[p_n]{} \alpha_n X\beta_n$, $\alpha_n \overset{*}{\Longrightarrow} \varepsilon$, $n \geqq 2$. However, $\alpha_j X\beta_j \neq \alpha_k X\beta_k$ when $j \neq k$, $1 \leqq j$, $k \leqq n$, $n \geqq 2$.

Therefore, Property 1 is concluded. [Q.E.D]

[Property 2]

When Step 2 is applied to G to construct T_1, if $T_1(A, X)=\{p_1, p_2, \ldots, p_n\}$, and $n \geqq 2$, then G cannot be LL(1), where $p_i \in P$, $1 \leqq i \leqq n$.

[Proof]

$T_1(A, X)=\{p_1, p_2, \ldots, p_n\}$, $n \geqq 2$
$\Longleftrightarrow$ there exist n derivations such that $A \underset{p_1}{\Longrightarrow} \alpha_1 \overset{*}{\Longrightarrow} \beta_1 X\gamma_1 \overset{*}{\Longrightarrow} X\gamma_1, \ldots,$ $A \underset{p_n}{\Longrightarrow} \alpha_n \overset{*}{\Longrightarrow} \beta_n X\gamma_n \overset{*}{\Longrightarrow} X\gamma_n$, $n \geqq 2$. However, $\alpha_j \neq \alpha_k$ when $j \neq k$, $1 \leqq j$, $k \leqq n$, $n \geqq 2$.

Therefore, Property 2 is concluded. [Q.E.D]

[Property 3]

Even if a step from Step 3 to 7 in H constructs a table with a multi-defined entry at some time while applying these steps to G, it cannot be concluded that G is not LL(1).

[Proof]

Suppose that at least one step from Step 3 to 7 constructs a table with a multi-defined entry in any M(X, Y) of the table. M denotes a part of the table local-FOLLOW or END-FOLLOW produced by one of those steps.

Since these steps calculate FOLLOW, such entry simply means that there exists XY at several places on a derived strings or right-hands of productions. Therefore, Property 3 is concluded. [Q.E.D]

[Property 4]

In Step 8 of H, if $T(X, a)=\{p_i, p_j\}$ for some G, then G is not LL(1), where $p_i, p_j \in P$, $i \neq j$.

[Proof]

No pair of (X, a) such that
$T_1(X, a)=\phi$
$T_2(X, a)=\{p_i, p_j\}$
exists in Step 8. If such a pair of (X, a) exists, there could exist an X such that $X \underset{p_i}{\Longrightarrow} \delta_i \overset{*}{\Longrightarrow} \varepsilon$, $X \underset{p_j}{\Longrightarrow} \delta_j \overset{*}{\Longrightarrow} \varepsilon$ in the derivation such that $S \overset{*}{\Longrightarrow} \alpha X\beta\gamma$, and $\beta \overset{*}{\Longrightarrow} at$. Such an X, however, cannot be found at the stage of Step 8, because G is checked by Algorithm K at the calculation for set Q.

If G is LL(1), then the following cannot be true:

$$T_1(X, a)=\{p_i, p_j\} \tag{4.1}$$

because in Property 1 and Property 2 it has been proved that the G is not LL(1) when Eq. (4.1) holds. Therefore, if $T(X, a)=\{p_i, p_j\}$, Eqs. (4.2) should hold because $T=T_1+T_2$.

$$T_1(X, a)=\{p_i\} \quad \text{and} \quad T_2(X, a)=\{p_j\} \tag{4.2}$$

Based on Eqs. (4.2), there exists a derivation $X \underset{p_i}{\Longrightarrow} \delta_i \overset{*}{\Longrightarrow} at_i$ for $T_1(X, a)=\{p_i\}$. Moreover, for $T_2(X, a)=\{p_j\}$, the following should be true:

$$S \overset{*}{\Longrightarrow} \alpha X\beta,\ \beta \overset{*}{\Longrightarrow} at_j, \text{ and } X \underset{p_j}{\Longrightarrow} \delta_j \overset{*}{\Longrightarrow} \varepsilon$$

Hence, G cannot be LL(1).[Q.E.D]

[Property 5]

G is not LL(1) $\Longleftrightarrow$ at least, one of properties among Property 1, Property 2, and Property 4 is true.

[Proof]

$\Longleftarrow$: obvious

$\Longrightarrow$: Iff G is not LL(1), at least, one of the equations below is true for an X in Algorithm H:

(1) $X \underset{p_1}{\Longrightarrow} \alpha_1 \overset{*}{\Longrightarrow} at_1, \ldots, X \underset{p_n}{\Longrightarrow} \alpha_n \overset{*}{\Longrightarrow} at_n$, and $n \geqq 2$
(2) $X \underset{p_i}{\Longrightarrow} \beta_i \overset{*}{\Longrightarrow} at_i$ in addition to $S \overset{*}{\Longrightarrow} \alpha X\beta$, $X \underset{p_j}{\Longrightarrow} \gamma \overset{*}{\Longrightarrow} \varepsilon$, and $\beta \overset{*}{\Longrightarrow} at_j$
(3) $X \underset{p_1}{\Longrightarrow} \alpha_1 \overset{*}{\Longrightarrow} \varepsilon, \ldots, X \underset{p_n}{\Longrightarrow} \alpha_n \overset{*}{\Longrightarrow} \varepsilon$, and $n \geqq 2$.

For the case (3), such an X has been already detected by K during the calculation for a set Q. Therefore, we have only to discuss (1) and (2) mentioned above. The proof of non-LL(1)ness of G in (1) is based on Property 1 and Property 2, and the proof in (2) based on Property 4.

Case (1):

The case (1) consists of the next two cases.

(1)-(a): a grammar G has a set such that $\{p_i | X \underset{p_i}{\Rightarrow} \alpha_i a t_i, \alpha_i \overset{*}{\Rightarrow} \varepsilon, 1 \leqq i \leqq n, n \geqq 2\}$. In this case, Property 1 can be applied, since $T_{11}(X, a) = \{p_1, p_2, \ldots, p_n\}$, and $n \geqq 2$ in Step 1 of Algorithm H.

(1)-(b): a grammar G has a set such that $\{p_i | X \underset{p_i}{\Rightarrow} \alpha_i \overset{+}{\Rightarrow} a t_i, 1 \leqq i \leqq n, n \geqq 2\}$. In this case, Property 2 can be applied, since $T_1(X, a) = \{p_1, p_2, \ldots, p_n\}$, and $n \geqq 2$ in Step 2 of Algorithm H.

Case (2):

For derivation such that $X \underset{p_i}{\Rightarrow} \beta_i \overset{*}{\Rightarrow} a t_i$,

$$T_1(X, a) = \{p_i\} \tag{4.3}$$

is true in Step 1 through 2. Now, being $F = F_1 + F_2$, the case is divided into the following two.

(i) Whenever $S \overset{*}{\Rightarrow} tD\beta \Rightarrow t\alpha X\gamma\beta$, and $\gamma \overset{*}{\Rightarrow} a\delta$, then $F_1(X, a) = \{*\}$ by Step 3 through 4.

(ii) Whenever $S \overset{*}{\Rightarrow} tD\beta \Rightarrow t\alpha X\gamma\beta$, $\gamma \overset{*}{\Rightarrow} \varepsilon$, and $\beta \overset{*}{\Rightarrow} a\delta$, then $F_1(D, a) = \{*\}$, and $E_2(X, D) = \{*\}$ by Step 3 through 4, and Step 5 through 6, respectively. Thus, $F_2(X, a) = \{*\}$ by Step 7. Therefore, according to $F = F_1 + F_2$

$$F(X, a) = \{*\} \tag{4.4}$$

According to Eq. (4.3), and the calculation by Step 8 for Eq. (4.4) and $X \underset{p_j}{\Rightarrow} \delta_j \overset{*}{\Rightarrow} \varepsilon$, we can obtain $T(X, a) = \{p_i, p_j\}$. Thus, Property 4 can be applied.

Since the cases, (1)-(a), (1)-(b), and (2), have no interactions when they appear, Property 5 can be true. [Q.E.D]

5. Conclusion

This paper has revealed five valuable properties of Algorithm H in the paper [2]. Furthermore, based on its properties, it has been shown that Algorithm H can be revised into an algorithm to point out the non-LL(1) ness of input grammars. That means Algorithm H can detect the non-LL(1)ness for input grammars without any set-operation, except finding set Q in Algorithm K.

We will pursue our further research, into applying the idea in Algorithm H to LL(k), $k \geqq 2$ and LR grammar.

Acknowledgment

The authors would like to thank Prof. Kenzo INOUE at Science Univ. of Tokyo who gave us a lot of valuable advices and thank the referees for their valuable suggestions.

References

1. GRUNE, D. and JACOBS, C. J. H. A Programmar-friendly LL(1) Paser Generator, *Software-Practice and Experimence*, **18**, Jan. (1988), 29-38.
2. YOSHIDA, K. and TAKEUCHI, Y. An Algorithm for Constructing LL(1) Parsing-Tables Using Production Indices (in Japanese), *Trans. IPS Japan* **11** (1986), 1095-1105.
3. AHO, A. V. and ULLMAN, J. D. The Theory of Parsing, *Translation, and Compiling*, **I**, Prentice-Hall, Inc., Englewood Cliffs, N. J. (1972), 333-367.
4. GRIFFITHS, M. LL(1) Grammars and Analysers, In GOOS and HARTMANIS ed., Compiler Construction, Lecture Notes in Computer Science, Springer-Verlag, **21** (1976), 57-84.
5. LEWIS II, P. M., ROSENKRANTZ, D. J. and STEARNS, R. E. Compiler Design Theory, Addison-Wesley, (1976), 262-276.
6. BACKHOUSE, R. C. Syntax of Programming Languages, Theory and Practice, Prentice-Hall, (1979), 117.
7. HUNTER, R. The Design and Construction of Compilers, John Wiley & Sons, (1981), 71-73.

(Received May 6, 1987; revised March 22, 1988)