On LLC(k) Parsing Method of LR(k) Grammars

KENZO INOUE* and FUKUMI FUJIWARA**

This paper firstly describes the motivation of developing LLC(k) parsing method which is a kind of mixed strategies for parsing LR(k) grammars. Secondly the structure and the generation-algorithm of parsing tables for LLC(k) method are described and the parsing algorithm using these generated tables is given. Finally some experimental results for several practical programming languages got by an LLC(1) parser-generator are shown and analyzed in comparison with LR(1) parsers for these languages.

1. Introduction

Since context free grammars (CFG's) have been applied to describe the syntax of programming languages, many deterministically parsable sub-classes of CFG's were developed. The largest one of them is the class of LR(k) grammars [1] and among rather small ones there are the classes of LL(k) [2], simple precedence grammars [3], and so on.

An objective of developing a sub-class, by defining the class properly to cover languages to treat, is to make possible, in the pertinent memory size, the generation of parsers by the method combining a parsing algorithm proper to the class and parsing tables of these languages. For a larger class, the syntax of languages are naturally described, but the parsers generated by the above method are large and wasteful in terms of memory space. On the contrary, for a smaller class, unnatural and clumsy rewriting of syntax is necessary to include in the class many languages, but the generated parsers are generally small and compact. Moreover, a grammar does not generally have uniform complexity. For example, for a CFG in LR(k), its LR(k)-ness is not necessarily caused by all combinations of productions. On the contrary, a considerable part of the combinations is in a range of LL(k) and only a part of these breaks the LL(k)-ness. Then if the complex part of the grammar is not rewritten, the generation of a wasteful parser is unavoidable for a usual language. From the above situation for parser-generation, it will be favourable to use a composite parsing algorithm which consists of an algorithm making parsable the complex part of syntax and another one making parsable the remaining simpler part. If this is possible, it will give a parsing method, in spite of the acceptance of a wider class of grammars, producing rather small parsers.

There have been several methods of algorithm composition up to now. There are, for example, the mixed strategy precedence [4], the weak precedence [5], the right precedence [6], and so on.

In the mixed strategy precedence, three kinds of algorithm are combined. Firstly (1, 1)-precedence is used to recognize the right end of a handle, next (2, 1)-precedence is used if (1, 1)-precedence is ambiguous and at last the method of so-called bounded context is used to decide the handle itself, the right end of which was just recognized. In the weak precedence and right precedence, (1, 1)-precedence is used to recognize the right end of a handle, for the determination of handle itself the bounded context method is used by relating the multiplicity of precedences to the left context of a phrase. Although the latter two methods are simplification of precedence algorithm, they accept super classes of precedence grammars. Especially the right precedence method is able to expand the class of grammars it accepts, without any modification of its parsing algorithm. However the above methods are not able to give so much widely expanded class from the original precedence grammar-one, because almost the same kind of algorithms are combined.

As another examples of composite parsing algorithms, there are almost-top-down parsing algorithm [7] and left corner grammars (LC(k)) [8] which use top-down algorithms as basis and are partially supplemented by bottom-up methods. Generally top-down parsing methods are able to have good perspective on parsing status, but accept only small classes of grammars. These characteristics are just reversed for bottom-up parsing methods. Especially LR(k) grammar class is the largest one deterministically parsable of CFG's. Then the combination of top-down and LR(k) algorithms is expected to give good perspective on parsing and a smaller parser for a wide class of CFG's.

Generalized LC(k)parsing method (GLC) [9] is a generalization of LC(k) one and an applicable production is decided when the subtrees of its leftmost i symbols have been recognized by LR(k)-like method ($0 \leq i \leq n_p =$the length of production). Therefore GLC(k) equals LL(k), LC(k), or LR(k) according to $i=0$, 1, or n_p respectively for every production. For GLC, however, the transfer from bottom-up to top-down algorithm must be indicatde by a special symbol ($\cdot$) between the i-th and $i+1$-th symbol of production. The difference of the GLC-method from LC-one is not only in increasing the number of left-corner symbols to be LR-likely recognized, but is in attempting equivalently to treat the LL- and LR-

*Science University of Tokyo.
**Fuji Xerox Co. Ltd.

methods in a parsing.

Least LC(k) parsing algorithm (LLC(k)) described in this paper is such a combined parser of LL(k) and LR(k) as GLC(k), but the fundamental aspect is to realize the smallest parser for a LR(k) grammar, by applying LR(k) algorithm for only parts of productions that are unparsable by LL(k) algorithm. Because these parts of productions are automatically recognized, special symbols on productions to indicate the transfer from bottom-up parsing to top-down are not necessary in this method.*

In the case of combined algorithm, the size of a parser also depends on the method of transfer from one algorithm to another in a parsing state. If there is a big difference between the context information required by the two algorithms, a large quantity of computation, a large table, or both are needed for the transfer, or the transfer might be impossible. At this point, it should be noted that the methods of mixed strategy, weak precedence, and right precedence combine algorithms requiring almost the same context information. LLC(k) parsing method adds a typical example about matching of context informations required by algorithms combined.

Basic definitions, general symbolisms, and so on are described in 2, and the informal and formal descriptions of the LLC-parser and its parsing algorithm are given in 3 and 4 respectively. In 5, some experimental results are discussed and conclusions are given.

2. Definitions, Symbolisms, and Others

The knowledge of LL(k) and LR(k) grammars, symbols, procedures, and functions not given in this paper are based on section 5.1 and 5.2 of Aho and Ullman [11]. The parsing tables and their generation algorithms of LL(k) and LR(k) of [11] are referred as canonical ones, and by LL(k) and LR(k) parsers and so on, without any notice, canonical ones are meant in this paper. The supplementary definitions are given here.

(1) A CFG, G, is represented by $G=(N, \Sigma, P, S)$, where N, Σ, and P are finite sets of non-terminal symbols, terminal symbols, and productions respectively. S is the start symbol of G. G is not ambiguous and does not include a useless nonterminal and a cycle.

(2) Following ordinary conventions,

$$A, B, C, \cdots \in N, \quad a, b, c, \cdots \in \Sigma, \quad X, Y, Z \in (N \cup \Sigma),$$
$$\alpha, \beta, \gamma, \cdots \in (N \cup \Sigma)^*, \quad s, t, u, \cdots \in \Sigma^*$$

are used with or without subscripts, but e denotes a null string and ϕ an empty set.

(3) In G, the productions are properly indexed: that is, in i: $A \to \alpha \in P$, i is the index of $A \to \alpha$ sometimes represented by p_A or $p_{A \to \alpha}$. The production of start symbol is unique and the index 0. $I=\{i\}$.

*LLC-algorithm was developed independently from GLC-method [10].

(4) The length of $\alpha = |\alpha|$, the number of elements in a set $J = |J|$.

(5) When $n=|\alpha|$, $m=|\beta|$, and $\gamma=\alpha\beta$,
$\alpha = {}^{(n)}\gamma = \gamma^{-(m)}$ and $\beta = \gamma^{(m)} = {}^{(n)-}\gamma$.
However if $n, m \geq |\gamma|$, then ${}^{(n)}\gamma = \gamma^{(m)} = \gamma$, and ${}^{(n)-}\gamma = \gamma^{-(m)} = e$.

(6) The local follow set of A with prefix u is $LFOLLOW_k(A) = \cup_i FIRST_k(v_i)$ for all possible i, where

$$S \overset{*}{\underset{lm}{\Rightarrow}} uAv_i \text{ and } FIRST_k(v_i) = \{{}^{(k)}t \mid v_i \overset{*}{\Rightarrow} t\}.$$

Because there are cases of

$$LFOLLOW_k^{u_1}(A) = LFOLLOW_k^{u_2}(A),\ u_1 \neq u_2,$$

different local follow sets of A are denoted $LFOLLOW_k^j(A)$ using $j=1, 2, \cdots$, but with the omission of j where there will be no confusion.

$$FOLLOW_k(A) = \cup_j LFOLLOW_k^j(A).$$

If G is an LL(k) grammar,

$$LFOLLOW_k(A) = FIRST_k(v_i),$$

because of the uniqueness of the leftmost derivation generating prefix uA.

(7) If there is no confusion, $U \oplus V$ stands for $U \oplus_k V$, where

$$U \oplus_k V = \{w \mid w = {}^{(k)}(xy) \text{ for } x \in U \text{ and } y \in V\}.$$

Similarly, $FIRST(v)$, LL, LR, and so on are substituted for $FIRST_k(v)$, LL(k), LR(k) and so on respectively.

(8) If U is the LR-item set for viable prefix $\mu\alpha_1$ and V is $LFOLLOW(A)$ with u being $S \overset{*}{\underset{rm}{\Rightarrow}} \mu A v \Rightarrow \mu\alpha_1\alpha_2 v$ and $\mu \overset{*}{\Rightarrow} u$, then

$$[A \to \alpha_1 \cdot \alpha_2, x] \in U \text{ for } x \in V$$
and
$$[A \to \alpha_1 \cdot \alpha_2, x] \notin U \text{ for } x \notin V.$$

In this paper the item set with a production $A \to \alpha_1\alpha_2$ is denoted by $[A \to \alpha_1 \cdot \alpha_2, V]$ using $LFOLLOW$ of A, V, in the given context, that is

$$[A \to \alpha_1 \cdot \alpha_2, V] = \{[A \to \alpha_1 \cdot \alpha_2, x] \mid x \in V\}.$$

And then U is treated as a set of these item sets.
$U = \{[A \to \alpha_1 \cdot \alpha_2, V] \mid \text{all possible } A \to \alpha_1\alpha_2 \text{ for viable prefix } \mu\alpha_1\}$.

(9) $Ker[U_{\mu X}]$ is a kernel set from which set $U_{\mu X}$, with viable prefix μX, is generated by a closuring operation [11].

(10) $L = \{A \mid A \overset{+}{\Rightarrow} A\gamma\}$, $N_l(A) = \{B \mid A \overset{*}{\Rightarrow} B\gamma\}$.

(11) A parse is an ordered sequence of the indices of productions used in parsing an input text.

3. Informal Description of LLC-Parsing and Parser

An LLC-parser executes the leftmost derivation by LL-parsing method for an input text as far as possible and then transfers to actions similar to LR-parsing, when an applicable production cannot be uniquely determined by LL-parsing.

In Fig. 3.1, let us assume that a prefix u of uv has been recognized by LL-parsing method, that is

$$S \overset{*}{\underset{lm}{\Rightarrow}} uBv,$$

but

$$B \to \beta_i,\ \cap_i FIRST(\beta_i) \oplus V_B \neq \phi,\ i = 1, 2, \cdots \tag{3.1}$$

with the *LFOLLOW* of B, V_B, in the given context. In this situation, the LLC-parser transfers its parsing mode to the next LR-like parsing for the partial text v_1,

$$B \overset{*}{\underset{rm}{\Rightarrow}} v_1,\ v_1 v_2 = v.$$

The range of v_1 is determined by the condition ${}^{(k)}v_2 \in V_B$, when the parsing of B is completed. This requires that the starting state of LR-parsing of v_1 is determined using the string Bx ($x \in V_B$). However it is uneconomical and unnecessary to execute completely the LR-parsing of B. LLC-parser is able to transfer to the LL-parsing for γ_2 of

$$C \overset{*}{\underset{lm}{\Rightarrow}} u_1 \gamma_2 \tag{3.2}$$

when an LR-item $[C \to \gamma_1 \cdot \gamma_2, y]$ is uniquely selected by a lookahead string ${}^{(k)}(u_2 v_0 x)$ at the LR-like parsing stage

$$Bx \overset{*}{\underset{rm}{\Rightarrow}} \xi \gamma_1 u_2 v_0 x,\ \gamma_1 \overset{*}{\underset{rm}{\Rightarrow}} u_1 \tag{3.3}$$

(cf. Fig. 3.1). This starting status of LL-parsing is uniquely given by selecting $[C \to \gamma_1 \cdot \gamma_2, y]$ from an item set of C with a particular *LFOLLOW(C)* included in the set of LR-item sets for viable prefix $\xi\gamma_1$. In the LL-parsing for γ_2, LR-like parsings are repeatedly executed if necessary. When the completion of LL-parsing of C, the status of parser again transfers to LR-like parsing with the LR-state for viable prefix ξC and *LFOLLOW(C)*, regarding as if $\xi u_1 u_2 v_0$ was reduced to $\xi C v_0$ by LR-like parsing (cf. (3.3)).

After the completion of LR-like parsing of v_1 to B, the parser returns to the leftmost derivation of v with the LL-state for uv_1v and *LFOLLOW(B)*, regarding as if derivation $uBv \overset{*}{\underset{lm}{\Rightarrow}} uv_1v$ was completed (cf. (3.1)).

From the above explanation, it is understood that

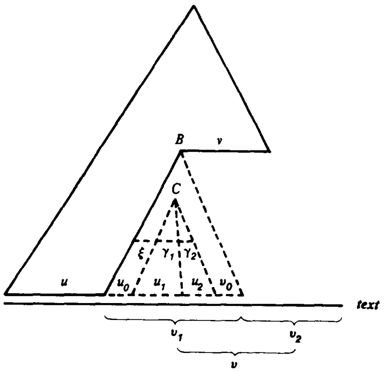

Fig. 1 Parsing tree for Eq. (3.1)

LLC-parser utilizes two kinds of tables, M and M_R for LL-parsing and LR-like respectively. Table M has almost the same structure as a canonical LL-parser table, but M_R is different from canonical LR-one. Several notations must be prepared now, for the explanation of table structure in the following.

An LLC-parser uses a left-open stack. Corresponding to the leftmost derivation from $\alpha\beta$ to $u\beta$ in

$$A \Rightarrow \alpha\beta \overset{*}{\underset{lm}{\Rightarrow}} u\beta,\quad \beta = x_{j-1} B_j x_j \cdots x_{m-1} B_m x_m$$
$$(\alpha = e \text{ when } j = 1),$$

which may be represented in LL-parsing as

$$T_{AV_A} \Rightarrow \bar{\alpha}\beta \overset{*}{\underset{lm}{\Rightarrow}} u\beta,$$
$$\text{where } \bar{\beta} = x_{j-1} T_{B_j V_j} x_j \cdots x_{m-1} T_{B_m V_m} x_m,$$

the leftmost derivation of LLC-parser is denoted by

$$T_{AV_A} \Rightarrow \bar{\alpha}\bar{\beta} p_{A \to \alpha\beta} \overset{*}{\underset{l}{\Rightarrow}} u\bar{\beta} p_{A \to \alpha\beta} = u\bar{v},$$
$$\text{where } \bar{v} = x_{j-1} T_{B_j V_j} x_j \cdots x_{m-1} T_{B_m V_m} x_m p_{A \to \alpha\beta},$$

reduced to $A \Rightarrow \alpha\beta \overset{*}{\underset{lm}{\Rightarrow}} u\beta$ with a homomorphism $h(T_{AV_A}) = A$, $h(T_{B_h V_h}) = B$, $h(\bar{a}) = a$, and $h(p_{A \to \alpha\beta}) = e$. Here T_{AV_A} and $T_{B_h V_h}$ are the LL-states of A with *LFOLLOW* V_A and B_h with *LFOLLOW* V_h respectively, called l-states in LLC-parsing. The l-state of the start symbol denoted by T_{S0} is unique because of the uniqueness of the *LFOLLOW* of S, that is $V_S = \{e\}$.

Corresponding to the reduction from u to α in

$$S \overset{*}{\underset{rm}{\Rightarrow}} \mu A w \Rightarrow \mu\alpha\beta w \overset{*}{\underset{rm}{\Rightarrow}} \mu\alpha v w \overset{*}{\underset{rm}{\Rightarrow}} \mu u v w,\ \alpha = X_1 X_2 \cdots X_j,$$

that is, the reduction from LR-state S_μ to $S_{\mu j}$ of LR-parser in

$$\begin{aligned} S_e S &\overset{*}{\underset{rm}{\Rightarrow}} S_e \cdots S_\mu A S_{\mu A} w \\ &\Rightarrow S_e \cdots S_\mu X_1 S_{\mu 1} X_2 S_{\mu 2} \cdots X_j S_{\mu j} \cdots S_{\mu\alpha\beta} w \\ &\overset{*}{\underset{rm}{\Rightarrow}} S_e \cdots S_\mu X_1 S_{\mu 1} X_2 S_{\mu 2} \cdots X_j S_{\mu j} v w \overset{*}{\underset{rm}{\Rightarrow}} S_e \cdots S_\mu u v w, \end{aligned}$$

the LR-like parsing of LLC-parser is denoted by

$$U_0^A \overset{*}{\underset{r}{\Rightarrow}} u\tilde{\mu} U_0^A,$$
$$\tilde{\mu} = U_j^A \cdots U_2^A U_1^A = inverse(U_1^A U_2^A \cdots U_j^A).$$

Here S_e, S_μ, $S_{\mu A}$, $S_{\mu h} (1 \leq h \leq j)$, and $S_{\mu\alpha\beta}$ are LR-states corresponding to viable prefixes e, μ, μA, $\mu X_1 \cdots X_h$, and $\mu\alpha\beta$ respectively. U_h^A is an r-state with viable prefix $X_1^A \cdots X_h^A$ of the LR-like parsing to the goal A with *LFOLLOW* V_A, and U_0^A is the starting state of this parsing. U_0^A and U_h^A correspond to S_μ and $S_{\mu h}$ respectively.

In general, because LL-parsing and LR-like are alternatively executed

$$T_{AV_A} \overset{*}{\underset{l}{\Rightarrow}} u_1 \bar{v}_1 \overset{*}{\underset{r}{\Rightarrow}} u_1 u_2 \tilde{v}_2 \bar{v}_1 \overset{*}{\underset{l}{\Rightarrow}} u_1 u_2 u_3 \bar{v}_3 \tilde{v}_2 \bar{v}_1 \cdots$$

or

$$T_{AV_A} \overset{*}{\underset{lr}{\Rightarrow}} u\tilde{\bar{v}},\ u = u_1 u_2 u_3 \cdots,\ \tilde{\bar{v}} = \cdots \bar{v}_3 \tilde{v}_2 \bar{v}_1.$$

where $\tilde{\bar{v}}$ is the content of the stack. It is understood later that symbols other than $X \in \Sigma \cup \Upsilon \cup \Psi \cup I$ are entered into the stack without any modification for the above symbolism of stack content, where Υ and Ψ are the sets of l-states and r-states respectively. In the following, parsing of LLC-parser is called *llc*-parsing, and it consists of l- and r-parsing.

The next example grammar is used for the illustration of table structure.

Example 3.1

$$G=(\{S, F, A, B, C, D\}, \{a, b, c, d, f\}, P, S)$$

$$P=\begin{Bmatrix} 0: S\to F,\ 1: F\to aA,\ 2: F\to bB,\ 3: F\to Ab, \\ 4: F\to Ba,\ 5: F\to CD,\ 6: A\to cA,\ 7: A\to c, \\ 8: B\to cB,\ 9: B\to c,\ 10: C\to fc,\ 11: D\to Dd, \\ 12: D\to d \end{Bmatrix}$$

End of Example 3.1

In Example 3.1 and the succeeding ones, symbols used in them do not have any relation with ones at the outside of them.

Let us assume that the *llc*-parsing of the input text uv has proceeded to

$$T_{S0}\overset{*}{\underset{lr}{\Rightarrow}}u'T_{AV_A}\tilde{\bar{v}}'\underset{l}{\Rightarrow}u'\bar{\alpha}_1T_{BV_B}z\overset{*}{\underset{lr}{\Rightarrow}}uT_{BV_B}\tilde{\bar{v}},\ \text{where } A\to\alpha_1B\alpha_2. \tag{3.4}$$

(1) The structure of table M.

A row of M is named by an l-state of A ($\in N$), T_{AV_A}, with an *LFOLLOW* V_A, a column is named by x ($\in\Sigma^{*k}$), and there are four kinds of entries: TD-, NTD-, SNTD-, and *error*-elements.

(L1) TD-element. For T_{BV_B} in (3.4), when an applicable production p_B: $B\to\beta$ is uniquely determined, the entry of $M(T_{BV_B}, {}^{(k)}v)$ is the same as the canonical one. That is, if

$$\beta=y_0C_1y_1\cdots C_my_m$$

then

$$M(T_{BV_B}, {}^{(k)}v)=\bar{\beta}p_B,\ \bar{\beta}=y_0T_{C_1V_1}y_1\cdots T_{C_mV_m}y_m, \tag{3.5}$$

where V_i ($1\le i\le m$) is the *LFOLLOW* of C_i in the given context.

(L2) NTD-element. This is the case of (3.1) which gives rise to r-parsing with the starting state generated from

$$\{[B\to\cdot\beta_i, V_B]\,|\,i=0, 1,\cdots\},\ {}^{(k)}v\in\cap_i(FIRST(\beta_i)\oplus V_B).$$

If U_0 represents the starting state,

$$M(T_{BV_B}, {}^{(k)}v)=U_0^B. \tag{3.6}$$

Example 3.1.1

If the parsing table for Example 3.1 is constructed with $k=1$ using the canonical LL-method, the productions 3 and 4 are applicable for $M(T_{F\{e\}}, c)$. In this case,

$$M(T_{F\{e\}}, c)=U_0^{F_1}=U_1=\{[F\to\cdot Ab,\{e\}],\ [F\to\cdot Ba,\ \{e\}]\}$$

for the LLC-parser (see Table 3.1), where it is superfixed with F_1 to discriminate from one more starting r-state of F (cf. Example 3.1.2).

End of Example 3.1.1

(L3) SNTD-element. This is the case of $B\in L$ on (3.4), in which it is necessary to prepare r-parsing in state T_{AV_A}. That is, using U_0^A generated from kernel

$$\{[A\to\alpha_1\cdot B\alpha_2, V_A]\},$$

$$M(T_{AV_A}, {}^{(k)}(u''v))=\bar{\alpha}_1(U_0^A)^{m+1}, m=|\alpha_1|, \alpha_1\overset{*}{\underset{l}{\Rightarrow}}u''. \tag{3.7}$$

The next parsing actions are executed on the basis of (3.7). Firstly, for the content of stack $T_{AV_A}\tilde{\bar{v}}'$, the topmost symbol T_{AV_A} is replaced by $\bar{\alpha}_1(U_0^A)^{m+1}$ and, by

Table 3.1 Parsing table for example 3.1.

M

		a	b	c	d	f	e
$T_{S\{e\}}$	T_1	T_20	T_20	T_20		T_20	
$T_{F\{e\}}$	T_2	aT_31	bT_42	U_1		$T_5U_2U_2$	
$T_{A\{e\}}$	T_3			U_3			
$T_{B\{e\}}$	T_4			U_4			
$T_{C\{d\}}$	T_5					$fc10$	

M_R

		action						*goto*									
		a	b	c	d	f	e	F	A	B	C	D	a	b	c	d	f
$U_0^{F_1}$	U_1			s				r	U_5	U_6					U_7		
$U_0^{F_2}$	U_2				$0,d12\,¢\,D$			r				U_8					
U_0^A	U_3			s					r						U_9		
U_0^B	U_4			s						r					U_{10}		
$U_1^{F_1}$	U_5		$1,b3\,¢\,F$														
$U_2^{F_1}$	U_6	$1,a4\,¢\,F$															
$U_3^{F_1}$	U_7	$1,9\,¢\,B$	$1,7\,¢\,A$	s					U_{11}	U_{12}					U_7		
$U_1^{F_2}$	U_8				$1,d11\,¢\,D$		$2,5\,¢\,F$										
U_1^A	U_9			$1,T_36\,¢\,A$			$1,7\,¢\,A$										
U_1^B	U_{10}			$1,T_48\,¢\,B$			$1,9\,¢\,B$										
$U_4^{F_1}$	U_{11}		$2,6\,¢\,A$														
$U_5^{F_1}$	U_{12}	$2,8\,¢\,B$															

Notes: s and r are shortenings of *shift* and *return* respectively.

the l-mode parsing following the replacement, $\bar{\alpha}_1$ is removed from the stack and $(U^A)^{m+1}$ is exposed at the stack top. At this point, the parsing transfers to r-mode. In the final step of this r-mode parsing, the content of the stack becomes $\tilde{\alpha}_2\tilde{\bar{B}}(U_0^A)^{m+1}\tilde{\bar{v}}'$. Here the parser executes the following actions corresponding to the reduction by $A\rightarrow\alpha_1B\alpha_2$, that is, after $|\alpha_1B\alpha_2|$ symbols are popped out from the stack, the parser returns to l-parsing by *goto* (U_0^A, A) (ref. R3).

Example 3.1.2

In Example 3.1, production 5 including D in the right hand side gives an SNTD-element, then

$$M(T_{F\{e\}}, f)=T_{C\{d\}}U_2U_2,\ U_2=U_0^{F_2}=\{[F\rightarrow C\cdot D, \{e\}]\}$$

End of Example 3.1.2

Unlike LL-parsing tables, M have no row for several kinds of non-terminals, as described in the following.
(L4) No T_{BV_B} for $B\in L$, as understood from (L3).
(L5) In (3.4), if there is an NTD-element on the row of T_{BV_B},

$$B\rightarrow\alpha C\beta_1,\ B\rightarrow\alpha X\beta_2,\quad C, X\notin L \text{ and } C\neq X,$$

and

$$W=\{FIRST(C\beta_1)\oplus V_B\}\cap\{FIRST(X\beta_2)\oplus V_B\}\neq\phi,$$

then this is not a case that is able to discriminate the following two item sets using a lookahead string, when arrived to a parsing state including the item sets

$$[C\rightarrow\cdot\gamma, V_C], [B\rightarrow\alpha\cdot X\beta_2, V_B].$$

That is, it requires further r-parsing in this case. Then, the row from C on table M is unnecessary so long as it is not required from other context.

Example 3.1.3

About the grammar of Example 3.1, the rows of A and B in M are not made from the context of

$$U_1=T_0^{F_1}=\{[F\rightarrow\cdot Ab, \{e\}], [F\rightarrow\cdot Ba, \{e\}]\}$$

described at Example 3.1.1. That is, there is no row of $T_{A\{b\}}$ and $T_{B\{a\}}$ in M.

End of Example 3.1.3

(2) The structure of table M_R.
Table M_R is a merged table of tables $M_R^{B'}$'s constructed from starting r-states U_0^B's appearing in NTD- and SNTD-elements of table M. Table M_R is divided to *action*- and *goto*-tables, and a row of both table is named by r-state U_i^B, a column is named by $x\in\Sigma^{*k}$ in the case of *action*-table and by $X\in(N\cup\Sigma)$ in the case of *goto*-table.
A state U_i^B is evaluated, starting from U_0^B, by a closuring operation which differs in the following point from the canonical method for generation of LR-parsing tables. That is, when $Ker[U_i^B]$ is given, the closuring operation is stopped at the point discriminating a unique item by a lookahead string.

Example 3.1.4

In Table 3.1,

$$U_9=U_1^A=\{[A\rightarrow c\cdot A, \{e\}], [A\rightarrow c\cdot, \{e\}]\}.$$

For a lookahead symbol c, $[A\rightarrow c\cdot A, e]$ is uniquely given. Then, we need not execute further closuring operation. When the lookahead string is c at the state U_1^A, the parser selects production $A\rightarrow cA$, then transfers to the l-parsing of A following c.

End of Example 3.1.4

(R1) After the above restricted closuring operation of $Ker[U_j^B]$, if

$$[C\rightarrow\gamma_1\cdot\gamma_2, \{x\}]\subset U_j^B$$

is a unique item for a lookahead string w, let

$$action(U_j^B, w)=(|\gamma_1|, \bar{\gamma}_2p_{C\rightarrow\gamma_1\gamma_2}¢\,C). \tag{3.8}$$

The meaning of (3.8) is as follows: Let us assume the parser transferred from the r-state U_i^B to U_j^B having a stacked state sequence $\tilde{\gamma}_1$ (that is $U_j^B=^{(1)}\tilde{\gamma}_1$). In the state the action (3.8) is decided by a lookahead to string w. First $\tilde{\gamma}_1$ is popped out from the stack, then $\bar{\gamma}_2p_{C\rightarrow\gamma_1\gamma_2}¢C$ is pushed and the parser transfers to l-parsing.
If, by the l-parsing continued, $\bar{\gamma}_2$ is popped out from the stack, it means the completion of the application of production $C\rightarrow\gamma_1\gamma_2$. At this point, there is $p_{A\rightarrow\gamma_1\gamma_2}¢CU_i^B$ on the stack top. $p_{A\rightarrow\gamma_1\gamma_2}$ is appended to the current partial parse and $goto(U_i^B, C)$ decides the next stack state.

Example 3.1.5

$Action(U_9, c)$ and $action(U_9, e)$ of Table 3.1 are examples for this case.

End of Example 3.1.5

(R2) The case of no unique item on the closuring operation. For example, for

$$[C\rightarrow\gamma_1\cdot X\gamma_2, \{x\}]\in U_j^B,\ w\in FIRST(X\gamma_2x),$$

another item set with the lookahead string w exists in U_j^B. In this case, $[C\rightarrow\gamma_1X\cdot\gamma_2, \{x\}]$ is entered into Ker of $U_f^B=goto(U_j^B, X)$. Further, if X is a terminal symbol, let $action(U_j^B, w)=shift$.

Example 3.1.6

On Table 3.1,

$$U_1=U_0^{F_1}$$
$$=\left\{\begin{matrix}[F\rightarrow\cdot Ab, \{e\}], [F\rightarrow\cdot Ba, \{e\}], [A\rightarrow\cdot cA, \{b\}],\\ [A\rightarrow\cdot c, \{b\}], [B\rightarrow\cdot cB, \{a\}], [B\rightarrow\cdot c, \{a\}].\end{matrix}\right\}.$$

All items in U_1 have a lookahead string c, then

$goto(U_1, A)=U_5=U_1^{F_1}$, $goto(U_1, B)=U_6=U_2^{F_1}$,
$goto(U_1, c)=U_7=U_3^{F_1}$, $action(U_1, c)=shift$,
$Ker[U_5]=\{[F\to A\cdot b, \{e\}]\}$, $Ker[U_6]=\{[E\to B\cdot a, \{e\}]\}$,
$Ker[U_7]=\{[A\to c\cdot A, \{b\}], [A\to c\cdot, \{b\}],$
$[B\to c\cdot B, \{a\}], [B\to c\cdot, \{a\}]\}$.

End of Example 3.1.6

(R3) Let be $goto(U_0^B, B)=return$.

4. Implementation of Parsing Tables and Parsing Algorithm

In this section, an algorithm producing the tables M and M_R, described in Informal Description 3, and a parsing algorithm using these tables are represented.

The table producing algorithm consists of a preparation algorithm A1, and generation ones, A2 and A3. In the following explanation, (*L1*), (*R1*), and so on are comments on the algorithms and indicate to refer the paragraphs with the same labels in Informal Description 3.

In A1, set $\Omega=\{(A, V_A, Q_{AV_A}, t_{AV_A})\}$ is generated, where $A\in N$, V_A is an *LFOLLOW* of A, t_{AV_A} is '*yes*' or '*no*' and if it is '*yes*' then row T_{AV_A} of M is generated using the corresponding triple (A, V_A, Q_{AV_A}). Q_{AV_A} is a set defined as follows:

$$\begin{aligned}
&Q_{AV_A}=\{(W_j, q(W_j))\mid 1\le j\le n_{AV_A}=|Q_{AV_A}|\},\\
&q(W_j)\\
&\quad=\{A\to\gamma_{ji}\langle V_{i1}, V_{i2},\cdots, V_{im_{ji}}\rangle \mid 1\le i\le n_j=|q(W_j)|\},\\
&W_j=\cap_i(FIRST(\gamma_{ji})\oplus V_A),\ 1\le i\le n_j,\ W_j\cap W_{j'}=\phi,\\
&1\le j, j'\le n_{AV_A}. \qquad (4.1)
\end{aligned}$$

Here, $V_{ih}(1\le h\le m_{ji})$ is the *LFOLLOW* of B_{ih} where $\gamma_{ji}=x_{i0}B_{i1}x_{i1}B_{i2}\cdots B_{im_{ji}}x_{im_{ji}}$ and *LFOLLOW* of A is V_A. W_j is the set of valid lookaheads selecting the same production set $q(W_j)$ of A with *LFOLLOW* V_A. Set Ω' is used to memorize a triple (B, V_B, t_{BV_B}) newly generated in the process generating an element of Ω.

For convenience' sake, hereafter k '\$'s are catenated to the right of a sentential form.

The input of A1 is a grammar $G=(N, \Sigma, P, S)$ and the output is Ω for G.

A1. Preparation Algorithm Evaluating Ω

(1) Initialization: $\Omega':=\{(S, \{\$^k\}, \text{'yes'})\}$, $\Omega:=\phi$.
(2) If $\Omega'=\phi$, then the end of A1, otherwise let

$$\Omega':=\Omega'-\{(A, V_A, t_{AV_A})\}$$

and execute the following calculation for this selected

$$(A, V_A, t_{AV_A}).$$

(3) Preliminary evaluation of *LFOLLOW* B_h's: For all productions of A, make

$$A\to x_0B_1x_1\cdots B_hx_h\cdots B_mx_m\langle V_1,\cdots, V_h,\cdots, V_m\rangle,$$

where

$$V_h=FIRST(x_hB_{h+1}\cdots B_mx_m)\oplus V_A,\ 1\le h\le m.$$

(4) Evaluation of Q_{AV_A}:
(i) Evaluate

$$W_i=\{x\mid x\in FIRST(\gamma_i)\oplus V_A\},\ W=\cup_iW_i,\ 1\le i\le n_A,$$

where $\gamma_i=x_0B_1x_1\cdots B_mx_m$, n_A=the number of A-productions.
(ii) Making production sets $q(W_l)$:
(a) For all $x_t\in W$, let

$$q(x_t)=\{A\to\gamma_i\langle\cdots\rangle\mid \text{all } A\to\gamma_i\text{'s being } \bar{x}_t\in W_i\}.$$

(b) Let

$$W_l=\{x_t\mid \text{all } x_t\text{'s with the same } q(x_t)\}.$$

(c) For all $(W_l, q(W_l))$, let

$$Q_{AV_A}:=\{(W_l, q(W_l))\mid \text{for all } l\text{'s}\},$$

where $q(W_l)=q(x_t)$, $x_t\in W_l$.

(iii) Modification of *LFOLLOW*'s:
(a) If $A\to\alpha_1B_h\alpha_2\langle\cdots, V_h,\cdots\rangle\in q(W_l)$ and $B_h\in L$ then let $V_h:=V_h\cup(FIRST(\zeta\alpha_2)\oplus V_A)$ for all ζ to be $B_h\overset{+}{\Rightarrow}B_h\zeta$.
(b) If $A\to\alpha_1B_h\alpha_2\langle\cdots, V_h,\cdots\rangle$, $A\to\beta_1B_h'\beta_2\langle\cdots, V_h',\cdots\rangle\in q(W_l)$, and $\alpha_1B_h=\beta_1B_h'$, then let $V:=V_h\cup V_h'$, $V_h:=V_h':=V$.

(5) Adding (B_h, V_h, t_h) to Ω': For all pairs of B_h and V_h evaluated by (3) and (4), do the following if $B_h\ne B$ or $V_h\nsubseteq V_B$ for all B and V_B being $(B, V_B, Q_{BV_B}, t_{BV_B})\in\Omega$ and $(B, V_B, Q_{BV_B})\in\Omega'$.
(i) If $B_h\in L$, then $t_h:=$'*no*'. (*L2*)
(ii) If

$$A\to\alpha_1B_h\alpha_2\langle\cdots, V_h,\cdots\rangle,\ A\to\beta_1X\beta_2\langle\cdots\rangle\in q(W_l),$$
$$\alpha_1=\beta_1,\ B_h\ne X,$$

and

$$(FIRST(B_h)\oplus V_h)\cap(FIRST(X\beta_2)\oplus V_A)\ne\phi,$$

then $t_h:=$'*no*'. (*L3*)
(iii) $t_h:=$'*yes*' for t_h's other than cases (i) and (ii).
(iv) $\Omega':=\Omega'\cup\{(B_h, V_h, t_h)\}$.

(6) $\Omega:=\Omega\cup\{(A, V_A, Q_{AV_A}, t_{AV_A})\}$.
(7) Return to (2).

End of A1

In the process generating M, kernels of start states of M_R are generated and entered into set Ψ together with their contextual informations and the pairs of the start state names and the corresponding goal symbols are added in set Γ.

An element U_s of set Ψ is a set of triples, that is,

$$\{([A \to \alpha_1 \cdot \alpha_2,\ V_A],\ W_l,\ W_l)\},$$

where

$$W_l \subseteq FIRST(\alpha_1\alpha_2) \oplus V_A \text{ and } W_l \subseteq FIRST(\alpha_2) \oplus V_A.$$

The input of algorithm A2 is Ω generated by A1 and the outputs are M, Ψ, Γ, and Ω_R, where Ω_R is about the same one as Ω, and used as auxiliary information for the generation of M_R.

A2. Generation Algorithm of Table M

(1) Initialization: $\Psi := \Gamma := \Omega_R := \phi$, $r := 0$, and execute (2) for $A = S$.

(2) If $\Omega = \phi$ then the end of A2, otherwise let

$$\Omega := \Omega - \{(A,\ V_A,\ Q_{AV_A},\ t_{AV_A})\},$$
$$\Omega_R := \Omega_R \cup \{(A,\ V_A,\ Q_{AV_A})\}.$$

If $t_{AV_A} =$ '*no*' for the selected $(A,\ V_A,\ Q_{AV_A},\ t_{AV_A})$ then repeat step (2). If it is not so, advance to (3).

(3) The evaluation of q_l:
While $Q_{AV_A} \neq \phi$, do (i) to (iv).

(i) $Q_{AV_A} := Q_{AV_A} - \{(W_l,\ q(W_l))\}$.

(ii) The case of $|q(W_l)| = 1$. Execute the next (a) and (b) for $A \to x_0B_1x_1 \cdots B_hx_h \cdots B_mx_m \langle V_1, \cdots, V_h, \cdots, V_m \rangle \in q(W_l)$.

(a) If $B_h \notin L$ $(h \leq m)$, then let

$$q_l = x_0 T_{B_1V_1} x_1 \cdots T_{B_mV_m} x_m p_{A\to\alpha} = \bar{\alpha} p_{A\to\alpha}. \quad (\text{*L1*})$$

(b) If $B_h \notin L$ $(h \leq n-1)$ and $B_n \in L$ $(n \leq m)$, then let

$$q_l = x_0 T_{B_1V_1} x_1 \cdots x_{n-1} U_s^{n'} U_{s'},$$
$$n' = |x_0B_1x_1 \cdots B_{n-1}x_{n-1}| \quad (\text{*L2*})$$

using U_s and $U_{s'}$ evaluated by the next method. Let $r := r+1$ and

$$U_r := \{([A \to x_0B_1 \cdots \cdot B_n \cdots B_mx_m,\ V_A],\ W_l,\ FIRST(B_n) \oplus V_n)\}.$$

Let

$$U_s = U_r,\ \Psi = \Psi \cup \{U_r\} \quad (\text{*L2*})$$

and if $A \notin N_l(B_n)$ then

$$U_{s'} = U_r,\ \Gamma = \Gamma \cup \{(U_r,\ A)\}$$

else

$$r := r+1,\ U_{s'} = U_r = \phi,\ \Gamma = \Gamma \cup \{(U_r,\ A)\}.$$

(iii) The case of $|q(W_l)| \geq 2$. For

$$A \to \gamma_{li} \langle V_{i1},\ V_{i2}, \cdots, V_{im_i} \rangle \in q(W_l),\ 1 \leq i \leq n_l,\ n_l \geq 2,$$

let $q_l = U_r$, using U_r evaluated by the next method.

$$r := r+1,$$
$$U_r := \{([A \to \cdot \gamma_{li},\ V_A],\ W_l,\ W_l) | 1 \leq i \leq n_l\},$$
$$\Psi := \Psi \cup \{U_r\},\ \Gamma := \Gamma \cup \{(U_r,\ A)\}.$$

(iv) Entries on row T_{AV_A} of M: Let $M(T_{AV_A}, x) = q_l$ using q_l evaluated by (i) to (iii) for all $x \in W_l$.

(4) $M(T_{AV_A}, x) = error$ for all $x \notin \cup_l W_l$ and return to (2).

End of A2

In the next, algorithm A3 for the generation of table M_R is described.

Among elements of *action*, there are three kinds *shift*, *error*, and (i, v) generated using Ω_R.

In $U_s \in \Psi$, only the kernel of item sets is included. By a restricted closuring operation for this kernel, sets K_1 and K_2 are evaluated, where K_1 is the set of item sets uniquely decided using lookahead strings. This operation expands U_s to $K_1 \cup K_2$.

When $([A \to \alpha_1 \cdot \alpha_2,\ V_A],\ W_l,\ W_i) \in J$, operation

$$J := J - \{([A \to \alpha_1 \cdot \alpha_2,\ V_A],\ W_l,\ W)\}$$

means

$$J := (J - \{([A \to \alpha_1 \cdot \alpha_2,\ V_A],\ W_l,\ W_i)\}) \cup \{([A \to \alpha_1 \cdot \alpha_2,\ V_A],\ W_l,\ W_i - W)\}.$$

Similarly,

$$J := J \cup \{([A \to \alpha_1 \cdot \alpha_2,\ V_A],\ W_l,\ W)\}$$

is used in the place of

$$J := (J - \{([A \to \alpha_1 \cdot \alpha_2,\ V_A],\ W_l,\ W_i)\}) \cup \{([A \to \alpha_1 \cdot \alpha_2,\ V_A],\ W_l,\ W_i \cup W)\}.$$

The inputs of A3 are Ψ, Γ, r (the maximum number of elements of Ψ), and Ω_R, and the output is M_R.

A3. Generation Algorithm of Table M_R

(1) Initialization: $s := 0$.

(2) If $s = r$ then go to (6), otherwise $s := s+1$ and go to the next.

(3) For $U_s \in \Psi$, execute the next operation:

(i) $H := U_s$, $K_1 := K_2 := \phi$.

(ii) If $H = \phi$ then go to (4), otherwise for

$$([A_h \to \alpha_{h1} \cdot \alpha_{h2},\ V_{A_h}],\ W_{A_hl},\ W_{A_hi}) \in H$$

and $\cap_h W_{A_hi} = W \neq \phi$ where $1 \leq h \leq n$ and $n \geq 1$ ($\cap_h W_{A_hi} = W_{A_hi} = W$ when $n = 1$), let

$$H := H - \{([A_h \to \alpha_{h1} \cdot \alpha_{h2},\ V_{A_h}],\ W_{A_hl},\ W) | 1 \leq h \leq n\}$$

and

$$J := \{([A_h \to \alpha_{h1} \cdot \alpha_{h2},\ V_{A_h}],\ W_{A_hl},\ W) | 1 \leq h \leq n\}.$$

(iii) If $|J| = |\{([A \to \alpha_1 \cdot \alpha_2,\ V_A],\ W_l,\ W)\}| = 1$ and ${}^{(1)}\alpha_2 \notin L$ then

$$K_1 := K_1 \cup J$$

and return to (ii).

(iv) If $([A\to\alpha_1\cdot C\alpha_2, V_A], W_{Al}, W_{Ai})\in J$ and $C\in L$, let

$$J:=(J-\{([A\to\alpha_1\cdot C\alpha_2, V_A], W_{Al}, W_{Ai})\}) \cup\{([C\to\cdot\gamma_j, V_C], W_{Cl}, W_{Ai}\cap W_{Cl})\},$$

$$K_2:=K_2\cup\{([A\to\alpha_1\cdot C\alpha_2, V_A], W_{Al}, W_{Ai})\}$$

using triple $([C\to\cdot\gamma_j, V_C], W_{Cl}, W_{Cl})$, where

$$([C\to\cdot\gamma_j, V_C], W_{Cl}, W_{Ai}\cap W_{Cl}) \neq([A\to\alpha_1\cdot C\alpha_2, V_A], W_{Al}, W_{Ai}),$$

which is decided as follows:

(a) $(W_{Al}, q(W_{Al}))\in Q_{AV_A}$ is uniquely decided using W_{Al} from $(A, V_A, Q_{AV_A})\in\Omega_R$ uniquely decided using A and V_A.

(b) Next $(C, V_C, Q_{CV_C})\in\Omega_R$ is uniquely decided from $A\to\alpha_1C\alpha_2\langle\cdots, V_C,\cdots\rangle$ uniquely decided using $A\to\alpha_1C\alpha_2$ and the above $(W_{Al}, q(W_{Al}))$.

(c) Finally all triples $([C\to\cdot\gamma_j, V_C], W_{Cl}, W_{Cl})$ are decided from all pairs $(W_{Cl}, C\to\gamma_j\in q(W_{Cl}))$ using V_C and all $(W_{Cl}, q(W_{Cl}))\in Q_{CV_C}$.

(v) If

$$([A_h\to\alpha_{h1}\cdot X_h\alpha_{h2}, V_{A_h}], W_{A_hl}, W_{A_hi})\in J,$$
$$1\leq h\leq n,\ n\geq 2,$$

and

$$\cap_h W_{A_hi}=W\neq\phi,$$

then execute

$$J:=(J-\{([A_h\to\alpha_{h1}\cdot X_h\alpha_{h2}, V_{A_h}], W_{A_hl}, W)|1\leq h\leq n\}) \cup\{([X_h\to\cdot\gamma_{hj}, V_{X_h}], W_{X_hj}, W\cap W_{X_hj})\},$$

$$K_2:=K_2\cup\{([A_h\to\alpha_{h1}\cdot X_h\alpha_{h2}, V_{A_h}], W_{A_hl}, W)| 1\leq h\leq n\}$$

in the following order, and with $[X_h\to\cdot\gamma_{hj}, V_{X_h}]$ and W_{X_hj} decided by (iv, a–c) when $X_h\in N(X_h$ and W_{X_hj} in the places of C and W_{Cl} respectively) or $[X_h\to\cdot\gamma_{hj}, V_{X_h}]=\phi$ when $X_h\in\Sigma$.

(a) First for all $X_h\notin N_l(X_{h'})$, $1\leq h, h'$.

(b) Secondly for all $X_h=X_{h'}$, $1\leq h, h'$, with $[X_h\to\cdot\gamma_{hj}, V_h]=\phi$ when $X_h\in\Sigma$.

(vi) Let $K_1:=K_1\cup J$ for the remaining elements of J and return to (ii).

(4) The generation of row U_s of table M_R from K_1 and K_2:

(i) Action-elements.

(a) For $([A\to\alpha_1\cdot\alpha_2, V_A], W_{Al}, W_{Ai})\in K_1$. Pick up

$$A\to\alpha_1\alpha_2\langle V_1,\cdots, V_n,\cdots, V_m\rangle\in q(W_{Al}),$$

using $(W_{Al}, q(W_{Al}))\in Q_{AV_A}$ decided from $(A, V_A, Q_{AV_A})\in\Omega_R$ and W_{Al}. When $\alpha_1=x_0B_1\cdots B_{n-1}x_{n-1}$ and $\alpha_2=y_{n-1}B_n\cdots B_mx_m$, generate action-elements for U_s and $x\in W_{Ai}$ in the next method.

The case of $B_n,\cdots, B_m\notin L$,

$$action(U_s, x)=(|\alpha_1|, y_{n-1}T_{B_nV_n}\cdots T_{B_mV_m}y_mp_{A\to\alpha_1\alpha_2}\cent A). \quad (*R1*)$$

The case of $B_n,\cdots, B_{h-1}\notin L$ and $B_h\in L$,

$$action(U_s, x)=(|\alpha_1|, y_{n-1}T_{B_nV_n}\cdots y_{h-1}(U_s')^g), \quad (*R1*)$$

$$g=|\alpha_1y_{n-1}B_n\cdots y_{h-1}|$$

where U_s' is decided as follows.
Let

$$U=\{([A\to\alpha_1y_{n-1}B_n\cdots y_{h-1}\cdot B_h\cdots B_my_m, V_A], W_{Al}, W_{Ak})\},$$

$$W_{Ak}=FIRST(B_h\cdots B_my_m)\oplus V_A.$$

If Ψ includes U_s' equal to this U, use the U_s', otherwise let

$$s':=r:=r+1,\ U_r:=U,\ \Psi:=\Psi\cup\{U_r\},$$

and use this U_r as U_s'.

(b) For $([A\to\alpha_1\cdot a\alpha_2, V_A], W_{Al}, W_{Ai})\in K_2$.

$$action(U_s, x)=shift,\ x\in W_{Ai},\ {}^{(1)}x=a. \quad (*R2*)$$

(ii) Goto-elements.
Let

$$goto(U_s, X)=U_s' \quad (*R2*)$$

for $([A_h\to\alpha_{h1}\cdot X\alpha_{h2}, V_{A_h}], W_{A_hl}, W_{A_hi})\in K_2$, where U_s' is decided as follows.
Let

$$U=\{([A_h\to\alpha_{h1}X\cdot\alpha_{h2}, V_{A_h}], W_{A_hl}, W_{A_hj})| \text{all possible } h\},$$

where

$$W_{A_hj}=\{x|\text{all } x\in(FIRST(\alpha_{h2})\oplus V_{A_h})\}.$$

If Ψ includes U_s' equal to this U, then use the U_s', otherwise

$$s':=r:=r+1,\ U_r:=U,\ \Psi:=\Psi\cup\{U_r\},$$

and use this U_r as U_s'.

(5) Return to (2).

(6) For all U_t being $(U_t, A)\in\Gamma$, let

$$goto(U_t, A)=return. \quad (*R3*)$$

(7) Let *error* all action- and goto-elements not decided in the above method.

End of A3

Next a parsing algorithm which uses table M and M_R generated by A1, A2, and A3 is represented in a form of program using a description language not explained here but easily understandable.

The head symbol of stack is placed in X and the remaining string is in R, t is an input text catenated with $\$^k$ at the right end and a right parse is generated on Ξ. The next procedures are defined.

pop = **begin** $X:={}^{(1)}R; R:={}^{(1)-}R$ **end**,
push(α) = $R:=\alpha R$.

About an element of type (i, v), decided by (4.i.a) of Algorithm A3, on line U_s and column x of M_R table, the fields of i and v are subsequently referred to as $actionA(U_s, x)$ and $actionB(U_s, x)$ respectively.

A4. LLC-Parsing Algorithm

```
begin
  var
    $X$:  union of $\Sigma$, $\Upsilon$, $\Psi$, $I$, {'¢'}, {'\$'}, and $N$;
    $R$:  string of elements of $\Sigma$, $\Upsilon$, $\Psi$, $I$, {'¢'}, {'\$'}, and
          $N$;
    $t$:  string of $a \in (\Sigma \cup \{'\$'\})$;
    $\Xi$:  sequence of $p \in I$;
    $s$:  integer;
  initialization: $t := $ text '\$···\$'; $X := T_{S0}$; $R := e$; $\Xi :=$
                  $e$;
  case $X$ of
    $X \in \Sigma$:  if $X = {}^{(1)}t$ then begin pop; $t := {}^{(1)-}t$ end
                     else fail;
    $X \in \Upsilon$:  if $M(X, {}^{(k)}t) \neq error$ then
            begin
              push($M(X, {}^{(k)}t)$); pop
            end
          else fail;
    $X \in \Psi$:  if action($X, {}^{(k)}t$) = shift then
            begin push($X$); $X := goto(X, {}^{(1)}t)$; $t := {}^{(1)-}t$
            end
          else
            if action($X, {}^{(k)}t$) $\neq$ error then
              begin
                $s := actionA(X, {}^{(k)}t)$; $R := {}^{(s-1)-}R$;
                push(actionB($X, {}^{(k)}t$)); pop
              end
            else fail;
    $X \in I$:  begin $\Xi := \Xi X$; pop end;
    $X =$ '¢': begin pop; $X := goto({}^{(1)}R, X)$;
            if $X = return$ then
              begin pop; pop end
            else
              if $X = error$ then fail
          end;
    $X =$ '\$': if $X = {}^{(1)}t$ then accept and halt else fail;
  end case
end A4                                              End of A4
```

When G is an SLL grammar, because it is possible to use *FOLLOW*'s instead of *LFOLLOW*'s, the generation method of parsing tables for SLL-grammars is simpler than for LL-grammars. In the case of LLC-method too, if G is an SLR(k) grammar or LALR(k), *FOLLOW*'s are used in the place of *LFOLLOW*'s. As a result of this replacement, table M is generated in the same algorithm as one for SLL-grammars. Therefore, an l-state T_{AV_A} of $A \in N$ is A itself because one needs not consider the difference between contexts of A. Further, in these cases, because it is possible to use for the generation of M_R techniques similar to SLR(k), LALR(k), and so on, described in [11], the table generation algorithm becomes very much easier for these grammars [12]. Although this simplified algorithm is not explained here, it will be referred to as AS hereafter.

For grammar G, it is proven by the comparison of the behaviors of LLC- and LR-parsers for G that if an input text t is a sentence on G then the LLC-parser generates the right parse of t, the same as one by LR-parser, with the number of elementary steps proportional to $|t|$ and if it is not a sentence the parser halts on the recognition of the leftmost error-symbol [12].

5. Experiments, Discussions, and Conclusions

In this section, LLC-parsing tables for small-sized grammars are examined, in comparison with their LR-parsers. After this, a similar comparison is done for the parsers of more practical languages, XPL, EULAR, FORTRAN 7000, and ALGOL 60, generated using an LLC-parser generator.

Example grammar 3.1, in Section 3, is really an LALR(1) grammar. It is possible, for implementing its parser, to use the simplified algorithm AS and the tables generated by AS are shown in Table 5.1(a), which are the same as ones in Table 3.1 except non-terminal names are substituted for the corresponding l-state names in Tables M and M_R. Table 5.1(b) shows LALR(1) parsing table for the same grammar.

Because the occurrence of r-states is originated in the deviation from LL-ness of the grammar, it is reasonable to represent the complexity of the grammar by

(*the number of r-states*)/((*the number of l- and r-states*) -1) (5.1)

In comparison with this formula,

$1-$(*the number of l- and r-states*)/(*the number of LR-states*) (5.2)

represents state-reducibility of LLC-technique from LR-one. These values for Table 5.1 are 0.75 and 0.26 respectively, as shown in Table 5.4 together with the numbers of terminals, non-terminals, productions, and the numbers of l-, r-, and LR-states.

In Table 5.4, the results of the following examples are also represented, and their parsers are shown in table 5.2 and 5.3.

Example 5.1

$G=(\{S, E, T, F\}, \{+, *, (,), i\}, P, S)$,

$$P=\begin{Bmatrix} 0\colon S\to E,\ 1\colon E\to E+T,\ 2\colon E\to T,\ 3\colon T\to T*F, \\ 4\colon T\to F,\ 5\colon F\to(E),\ 6\colon F\to i \end{Bmatrix}$$

End of Example 5.1

Example 5.2

$G=(\{S_0, S, A, B\}, \{a, b, c, d\}, P, S_0)$,

$$P=\begin{Bmatrix} 0\colon S_0\to S,\ 1\colon S\to Aa,\ 2\colon S\to dAb,\ 3\colon S\to Bb, \\ 4\colon S\to dBa,\ 5\colon A\to c,\ 6\colon B\to c \end{Bmatrix}$$

End of Example 5.2

Table 5.1(a) LLC-parsing table for example 3.1.

M	a	b	c	f
S	$F0$	$F0$	$F0$	$F0$
F	$aA1$	$bB2$	U_1	CU_2U_2
A			U_3	
B			U_4	
C				$fc10$

M_R	*action*					*goto*				
	a	b	c	d	e	F	A	B	D	c
U_1			s			r	U_5	U_6		U_7
U_2				0,d12 ¢ D		r			U_8	
U_3			s				r			U_9
U_4			s					r		U_{10}
U_5		1,b3 ¢ F								
U_6	1,a4 ¢ F									
U_7	1,9 ¢ B	1,7 ¢ A	s				U_{11}	U_{12}		U_7
U_8				1,d11 ¢ D	2,5 ¢ F					
U_9			1,A6 ¢ A		1,7 ¢ A					
U_{10}			1,B8 ¢ B		1,9 ¢ B					
U_{11}		2,6 ¢ A								
U_{12}	2,8 ¢ B									

Table 5.1(b) LR-parsing table for example 3.1.

	action					*goto*										
	a	b	c	d	f	e	F	A	B	C	D	a	b	c	d	f
S_1	s	s	s		s		$S_{1.1}$	S_5	S_6	S_2		S_3	S_4	S_7		$S_{1.2}$
$S_{1.1}$						a										
S_2				s							S_8				$S_{2.1}$	
S_3			s					$S_{3.1}$						S_9		
S_4			s						$S_{4.1}$					S_{10}		
S_5		s											$S_{5.1}$			
S_6	s											$S_{6.1}$				
S_7	9	7	s					S_{11}	S_{12}					S_7		
$S_{1.2}$			s											$S_{1.21}$		
S_8				s		5									$S_{8.1}$	
$S_{2.1}$				12		12										
$S_{3.1}$						1										
S_9			s			7		$S_{9.1}$						S_9		
$S_{4.1}$						2										
S_{10}			s			9			$S_{10.1}$					S_{10}		
$S_{5.1}$						3										
$S_{6.1}$						4										
S_{11}		6														
S_{12}																
$S_{1.21}$				10												
$S_{8.1}$				11		11										
$S_{9.1}$						6										
$S_{10.1}$						8										

Example 5.1 is an SLR(1) grammar and Example 5.2, taken from Example 7.27 of [11], is a non-LALR(1) grammar, an action-conflict of which is resolved by splitting state 5 to states 5 and 9, based on Algorithm AS [12, 11]. Because the grammars of Example 5.1 and 5.2 have special complexities and these all are small sized ones, then it is natural to have rather low reducibilities as shown in Table 5.4.

To see what tendency for the grammars of languages for practical usage are shown, LLC(1) parsers of XPL [4], EULER [3], FORTRAN 7000 [13], and ALGOL 60 [14] were generated using an LLC(1) parser generator based on Algorithm AS. At the same time, LALR(1) parsers for the above languages were generated using an LALR(1) parser generator.

The figures based on the generated parsers are shown Table 5.5. In comparison with Table 5.4, table reducibility of LLC-parsing tables to LR-parsing ones (1 − *size-ratio of both tables*) is shown in Table 5.5.

The generated parsing tables are a list type with excluded error-elements, and contracted with several techniques superposing elements with the same values. Of course, these contraction techniques are equally applied to both LLC- and LR-parsing tables. The size of a table

Table 5.2(a) LLC-parsing table for example 5.1.

M

	(	i
S	U_1	U_1
F	(U_5U_6	$i6$

M_R	*action*						*goto*			
	+	*	(	)	i	e	S	E	T	F
U_1			0,(U_5		0,$i6$¢ F		r	U_2	U_3	U_4
U_2	1,+ U_8U_8					1,0¢ S				
U_3	1,2¢ E	1,*$F3$¢ T		1,2¢ E		1,2¢ E				
U_4	1,4¢ T	1,4¢ T		1,4¢ T		1,4¢ T				
U_5			0,(U_5		0,$i6$¢ F			U_7	U_3	U_4
U_6										r
U_7	1,+ U_8U_8			2,)5¢ F						
U_8			0,(U_5		0,i¢ F				U_9	U_4
U_9	3,1¢ E	1,*$F3$¢ T		3,1¢ E		3,1¢ E				

Table 5.2(b) LR-parsing table for example 5.1.

	action						*goto*								
	+	*	(	)	i	e	S	E	T	F	+	*	(	)	i
S_1			s		s			S_2	S_3	S_4			S_5		$S_{1.1}$
S_2	s					a					S_8				
S_3	2	s		2		2						$S_{3.1}$			
S_4	4	4		4		4									
S_5			s		s			S_7	S_3	S_4			S_5		$S_{1.1}$
$S_{1.1}$	6	6		6		6									
S_7	s			s							S_8			$S_{7.1}$	
S_8			s		s				S_9	S_4			S_5		$S_{1.1}$
$S_{3.1}$			s		s					$S_{3.2}$			S_5		$S_{1.1}$
S_9	1	s		1		1						$S_{3.1}$			
$S_{3.2}$	3	3		3		3									
$S_{7.1}$	5	5		5		5									

Table 5.3(a) LLC-parsing tables for example 5.2.

M

	c	d
S_0	$S0$	$S0$
S	U_1	U_2

M_R	*action*				*goto*				
	a	b	c	d	S	A	B	c	d
U_1			s		r	U_3	U_4	U_5	
U_2				s	r				U_6
U_3	1,$a1$¢ S								
U_4		1,$b3$¢ S							
U_5	1,5¢ A	1,6¢ B							
U_6			s			U_7	U_8	U_9	
U_7		2,$b2$¢ S							
U_8	2,$a4$¢ S								
U_9	1,6¢ B	1,5¢ A							

Table 5.3(b) LR-parsing table for example 5.2.

	action					*goto*						
	a	b	c	d	e	S	A	B	a	b	c	d
S_1			s	s		S_2	S_3	S_4			S_5	S_6
S_2					a							
S_3	s									$S_{3.1}$		
S_4		s									$S_{4.1}$	
S_5	5	6										
S_6			s				S_7	S_8			S_9	
$S_{3.1}$					1							
$S_{4.1}$					3							
S_7		s								$S_{7.1}$		
S_8	s								$S_{8.1}$			
$S_{7.1}$					2							
$S_{8.1}$					4							
S_9	6	5										

keeping the non-terminals of the left-hand sides and the lengthes of the right-hand sides of each productions is added in the size of LR-table. In the case of LLC-parser, there is not such a table, because of including all informations of productions in M and M_R tables.

From Table 5.5, it is understood that the value of table-reducibility is about the same as one of state-reducibility for each grammar and these values represent grammar-dependency. It is very intresting that the complexity is remarkably low and both reducibilities are high for EULER with a simple precedence grammar.

Table 5.4 Complexity and comparison between LLC- and LR-parsers of example grammars 3.1, 5.1 and 5.2.

Grammars	Example 3.1	Example 5.1	Example 5.2
Terminals	5	5	4
Non-terminals	6	4	4
Productions	13	7	6
l-states	5	2	2
r-states	12	9	9
LR-states	23	12	13
Complexity	0.75	0.90	0.90
State-reducibility	0.26	0.08	0.15

Table 5.5 Complexity and comparison between LLC- and LR-parsers of XPL, EULER, FORTRAN 7000 and ALGOL 60.

Grammars	XPL	EULER	FORTRAN	ALGOL 60
Terminals	47	74	63	66
Non-terminals	51(13)	45(12)	77(20)	99(22)
Productions	108	121	172	205
l-states	38	33	57	77
r-states	101	59	153	190
LR-states	180	193	322	337
LLC-table (bytes)	1,476	1,288	2,426	3,154
LR-table (bytes)	2,041	2,587	3,662	4,264
Complexity	0.73	0.65	0.73	0.71
State-reducibility	0.24	0.52	0.35	0.21
Table-reducibility	0.28	0.50	0.34	0.26

As a summary, the following should be said.

(1) The LLC parsing algorithm accepting sentences on an LR grammar and the generation algorithm of the parser which combined to LL-parsing the least amount of LR-like parsing were described.

(2) For transitions between *l*- and *r*-states in LLC-parsing, it is not necessary to have any additional context information, because *LFOLLOW*'s correctly give information for both *l*- and *r*-states (*FOLLOW*'s when the SLL and SLR combination).

(3) If SLL grammars are used as the basis of LLC technique, LALR-like techniques resolving action-conflicts for LR grammar should also be usable.

(4) LLC-parsing correctly generates the right parse for an input text, with the number of steps proportional to its length, when the text is a sentence on the grammar *G* on which the parser is generated, and the parser recognizes the leftmost error and halts at the error position when the text is not a sentence on *G* [12].

(5) The parsing speed of LLC technique might be thought to be slower than a LR parser for the same sentence because of expansion (case of $X \in \Upsilon$ in A4) processes occurring at *l*-states of the former. However, because the final state of an expansion sequence is known at the time of table generation, it is easy to generate a table replacing by one step-action an expansion sequence.

(6) About the languages for practical usages, XPL, EULER, FORTRAN 7000, and ALGOL 60, the reduction of table size from LR-parsing table (SLR or LALR) is about 35 percents in average, although the reduction value is language-dependent and is the largest for EULER with a simple precedence grammar.

(7) An objective measure was given to represent the complexity being deviation from LL-ness of grammars.

References

1. Knuth, D. E. On the Translation of Languages from Left to Right, *Information and Control* **8**, 6 (1965), 607–639.
2. Lewis, P. M. II and Stearns, R. E. Syntax Directed Transduction, *J. ACM* **15**, 3 (1968), 464–488.
3. Wirth, N. and Weber, H. EULER—a Generalization of ALGOL and Its Formal Definition, Part I and Part II, *C. ACM* **9**, 1, 13–23 and 2 (1966), 89–99.
4. McKeeman, W. M., Horning, J. J. and Wortman, D. B. A Compiler Generator, Prentice-Hall Inc., Englewood Cliffs, N.J. (1970).
5. Ichbiah, J. D. and Morse, S. P. A Technique for Generating Almost Optimal Floyd-Evans Productions for Precedence Grammars, *C. ACM* **13**, 8 (1970), 501–508.
6. Inoue, K. Right Precedence Grammars (in Japanese), *Johoshori* **11**, 8 (1970), 449–456: Inoue, K. Right Precedence Grammars, *Information Processing in Japan* **11**, (1971), 24–29.
7. Kral, J. Almost Top-Down Analysis for Generalized LR(*k*) Grammars, Lecture Notes in Computer Sciences **47**, (1977), 149–172.
8. Rosenkrantz, D. J. and Lewis. II, P. M. Deterministic Left Corner Parsing, IEEE Conf. Record 11th Annual Symposium on Switching and Automata Theory, (1970), 139–152.
9. Demers, A. J. Generalized Left Corner Parsing, Conf. Record of the Fourth Annual ACM Symposium on Principles of Programming Languages, (1977), 170–181.
10. Fujiwara, F. and Inoue, K. An Extension of *LL*(*k*) Parsing Method by LR(*k*)-like Method (in Japanese), *Proc. of ECSJ*, (1977), 1304: Fujiwara, F. and Inoue, K. A Mixed Approach of Parsing Method for SLR(*k*) Grammars, *Proc. of IPSJ* **18**, (1977), 599–600: Fujiwara, F. A Flexible-Top-Down Analysis for LR(*k*) Grammars, Master Thesis, Dept. of Information Sciences, Tokyo Institite of Technology (1978).
11. Aho, A. V. and Ullman, J. D. The Theory of Parsing, Translation, and Compiling I and II, Prentice-Hall, Inc., Englewood Cliffs, N.J. (1972).
12. Inoue, K. and Fujiwara, F. On LLC(*k*)-parsing Method of LR(*k*) Grammars, Research Reports on Information Sciences, Series C, C-37, Tokyo Institute of Technology (1981).
13. Kojima, T., Kato, M. and Nakata, I. An LR(*k*)-Parser Generation System and Its Application to FORTRAN-Compiler (in Japanese), *Johoshori* **15**, 2 (1974), 93–100.
14. Naur, P. et al. Report on the Algorithmic Language ALGOL **60**, *C. ACM* 3, 1 (1963), 1–17.

(Received May 6, 1982; revised July 25, 1983)