<table>
<thead>
<tr>
<th>Title</th>
<th>Unrestricted $LR(k)$ Grammars and its Parser, where $k=0,1$ (New Developments of Theory of Computation and Algorithms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Shiina, Hiromitsu; Masuyama, Shigeru</td>
</tr>
<tr>
<td>Citation</td>
<td>数理解析研究所講究録 (2001), 1205: 154-159</td>
</tr>
<tr>
<td>Issue Date</td>
<td>2001-05</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2433/41017">http://hdl.handle.net/2433/41017</a></td>
</tr>
<tr>
<td>Type</td>
<td>Departmental Bulletin Paper</td>
</tr>
<tr>
<td>Textversion</td>
<td>publisher</td>
</tr>
</tbody>
</table>

Kyoto University
Unrestricted $LR(k)$ Grammars and its Parser, where $k = 0, 1$

1 Introduction

Many parsers were developed for context-free languages. However, we often need to parse noncontext-free languages in natural language processing. Some parsers [1][2][4] were developed for classes larger than that of context-free languages. Vol'v'man[1] and Harris[2] developed parsers for the unrestricted grammar.

Along this line, we proposed the unrestricted $LR(k)$ grammar and the unrestricted $LR(k)$ parser [4]. The conventional $LR(k)$ grammar [3] has lookahead strings which consists of terminal symbols. On the other hand, in the unrestricted $LR(k)$ grammar, we proposed to regard lookahead strings as nonterminal symbols corresponding to roots of subtrees of deductive parse trees. Note that the unrestricted $LR(k)$ parser also works deterministically by using lookahead strings and parses the language $L(G)$ where it is in the class of recursive languages.

In this paper, we shall show how to transform an unrestricted $LR(k)$ grammar to an unrestricted $LR(1)$ grammar for any $k \geq 2$, which implies that there is an equivalent unrestricted $LR(1)$ grammar for any unrestricted $LR(k)$ grammar, $k \geq 2$, in that they generate the same language. A conventional $LR$ grammar where generation capacity of $LR(k)$, $k \geq 2$, grammar also coincides with $LR(1)$ grammar when $k$ is finite[3]. However, it is interesting that this is also true even when $k = \infty$ when we consider the unrestricted $LR$ grammar. Moreover, we clarify that a language generated by any recursive phrase structure language can be generated by some unrestricted $LR(k)$ grammar, where $k = 0, 1$.

2 Definitions

We define the unrestricted $LR(k)$ grammar, $G = (N,T,P,S)$, where $N$ is a finite set of nonterminal symbols, $T$ is a finite set of terminal symbols, $P$ is a finite set of production rules and $S$ is a start symbol. As preliminaries, we define the extended $LR(0)$ state transition diagram (LTD for short), the reachability state transition diagram (RTD for short) and lookahead strings where LTD, RTD and lookahead strings are used in the definition of the unrestricted $LR(k)$ grammar.

2.1 Definition of an LTD

An extended $LR(0)$ state transition diagram (LTD for short) is constructed from production rules of a phrase structure grammar $G = (N,T,P,S)$, which specifies a control part of a pushdown automaton. This LTD extends the $LR(0)$ state transition diagram to the phrase structure grammar. Basically, the creation algorithm is the same.

An LTD is constructed by a 6-tuple $M_L = (Q_L, \delta, \sigma_0, \sigma_{acc}, S')$. Here, $Q_L = \{s_0, s_1, \ldots, s_{n-1}, s_n\}$ is a set of states, $\delta$, is a set of shift functions satisfying $\delta_r(s,X) = s'$, $s,s' \in Q_L$, $X \in N \cup T$, $\delta$ is a set of reduce functions satisfying $\delta_e(s,X) = \{\lambda \rightarrow \mu, s'\}$, $s' \in Q_L$, $\delta$ is an initial state, $\sigma_{acc}$ is a final state and for convenience sake let $s_{acc} = s_n$, and $S'$ is a dummy symbol to construct an LTD in an initial step.

A state $s_i \in Q_L$, $i = 0,1,\ldots,n$, is constructed from items called $LR(0)$ items. An $LR(0)$ item consists of production rules with a dot (.) like $[\lambda \rightarrow \mu_1\mu_2]$, $[\lambda \rightarrow \mu_1 \cdot \mu_2]$ and $[\lambda \rightarrow \mu_1 \mu_2]$. A set of states and functions of LTD are constructed by the following five actions. Note that the length of string $\mu$ is shown as $|\mu|$, and an empty word is shown as $\varepsilon$.

[Action A: Initial action] As an initial action, we append a dummy rule $S' \rightarrow S$ to $P$, define an initial state $s_0$ and a final state $s_{acc}$

\[
\begin{align*}
\delta_r(s_0, S) &= \delta_e(s_0, S) := s_{acc}.
\end{align*}
\]

[Action B] If $[\lambda \rightarrow \mu_1 \cdot X \cdot \mu_2] \in s_i$ where $s_i \in Q_L$, $\mu_1, \mu_2 \in (N \cup T)^*$, then

- for each $X \xi \rightarrow \eta \in P$ where $\xi, \eta \in (N \cup T)^*$, $s_i := s_i \cup \{X \xi \rightarrow \eta\}$.

- for each $s_j \in Q_L$ such that $[\lambda \rightarrow \mu_1 X \cdot \mu_2] \notin s_j$. 
create a new state $s_k$,
$$Q_L' := Q_L \cup \{s_k\}$$
and
$$s_k := s_k \cup \{[\lambda \rightarrow \mu_1 X \cdot \mu_2]\}.$$

[Action C] If $[\lambda \rightarrow \mu_1 X \cdot \mu_2] \in s_i$, $[\lambda \rightarrow \mu_1 X \cdot \mu_2] \in s_j$ where $s_i, s_j \in Q_L$ and $\mu_1, \mu_2 \in (N \cup T)^*$, then
$$\delta _s(s_i, X) := s_j.$$

[Action D] If $[\lambda \rightarrow \mu X] \in s_i$, $[\lambda \rightarrow \mu X] \in s_j$
and
$$\delta _s(\{\ldots \delta _s([s_i, \mu_1, \mu_2] \ldots \mu_{|\mu|}], X\}) = s_i$$
where $s_i, s_j, s_k \in Q_L$, $\mu = \mu_1 \mu_2 \cdots \mu_{|\mu|}$,
$$\mu_1, \mu_2, \ldots, \mu_{|\mu|} \in N \cup T,$$ then
$$\delta _s(s_i, X) := \delta _s(s_i, X) \cup \{[\lambda \rightarrow \mu X, s_k]\}.$$

[Action E] If $[\lambda \rightarrow \epsilon] \in s_i$ where $s_i \in Q_L$, then
$$s_i := s_i \cup \{[\lambda \rightarrow \epsilon]\}$$
and
$$\delta _s(s_i, \epsilon) := \delta _s(s_i, \epsilon) \cup \{[\lambda \rightarrow \epsilon, s_i]\}.$$

Here, we use a grammar $G_1 = (\{S, A, B, C, D, E\}, \{S \rightarrow DE, DE \rightarrow ED, E \rightarrow ABB, A \rightarrow CB, A \rightarrow a, C \rightarrow CA, C \rightarrow a, B \rightarrow b, D \rightarrow d\}, \{a, b, d\}, S)$ as a working example. We can construct an LTD $M_L = (Q_L, \delta _s, \delta _r, s_0, s_{acc}, S')$ for the grammar $G_1$ where $Q_L = \{s_0, s_1, \ldots, s_{14}, s_{acc}\}$ and function $\delta _s$ and $\delta _r$ are shown in Tables 1 and 2.

In addition, contents of states are the following $LR(0)$ items,

$$s_0 = \{[S \rightarrow \cdot S], [S \rightarrow \cdot DE], [DE \rightarrow \cdot ED], [E \rightarrow \cdot ABB], [E \rightarrow \cdot CB], [A \rightarrow \cdot AA], [A \rightarrow \cdot a], [C \rightarrow \cdot AC], [C \rightarrow \cdot a], [D \rightarrow \cdot d]\},$$

$$s_1 = \{[A \rightarrow \cdot a], [C \rightarrow \cdot a\cdot]\},$$

$$s_2 = \{[S \rightarrow D \cdot E], [E \rightarrow \cdot ABB], [E \rightarrow \cdot CB], [A \rightarrow \cdot AA], [A \rightarrow \cdot a], [C \rightarrow \cdot AC]\},$$

$$s_{acc} = \{[S \rightarrow \cdot S]\}.$$

### 2.2 Definition of an RTD

In order to construct an RTD, we consider a push-down automaton which begins with $s_i$ of an LTD by a nonterminal symbol $X$ and stops at $s_j$ of an LTD. Then we create a transition from $s_i$ to $s_j$ by $X$ in an RTD. In short, we write $s_i \xrightarrow{X} s_j$. An RTD is constructed by this transition for all states of an LTD and nonterminal symbols, and is nondeterministic automaton.

<table>
<thead>
<tr>
<th>Table 1: Function $\delta _s$ for $G_1$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta <em>s(s_0, S) := s</em>{acc}$</td>
</tr>
<tr>
<td>$\delta _s(s_0, A) := s_1$</td>
</tr>
<tr>
<td>$\delta _s(s_0, E) := s_3$</td>
</tr>
<tr>
<td>$\delta _s(s_0, C) := s_7$</td>
</tr>
<tr>
<td>$\delta _s(s_2, D) := s_5$</td>
</tr>
<tr>
<td>$\delta _s(s_2, A) := s_7$</td>
</tr>
<tr>
<td>$\delta _s(s_2, D) := s_5$</td>
</tr>
<tr>
<td>$\delta _s(s_4, D) := s_5$</td>
</tr>
<tr>
<td>$\delta _s(s_4, A) := s_4$</td>
</tr>
<tr>
<td>$\delta _s(s_4, C) := s_11$</td>
</tr>
<tr>
<td>$\delta <em>s(s_4, B) := s</em>{10}$</td>
</tr>
<tr>
<td>$\delta <em>s(s_5, B) := s</em>{10}$</td>
</tr>
<tr>
<td>$\delta <em>s(s_7, A) := s</em>{13}$</td>
</tr>
<tr>
<td>$\delta <em>s(s_7, B) := s</em>{13}$</td>
</tr>
<tr>
<td>$\delta <em>s(s_7, B) := s</em>{13}$</td>
</tr>
</tbody>
</table>

An RTD $M_R$ is a 4-tuple $(Q_R, \delta _R, s_0, s_{acc})$ where $Q_R$ is a finite set of states of $M_R$, $\delta _R$ is a finite set of functions, $s_0$ is an initial state, $s_{acc}$ is a final state. A set of states $Q_R$ and a set of functions $\delta _R$ is defined by the following [shift action] and [reduce action], respectively.

[shift action] If $(s_i, X) \vdash (s_j, \epsilon)$ where $s_i, s_j \in Q_L$ and $X \in N$, then
$$Q_R := Q_R \cup \{s_i, s_j\},$$
$$\delta _R(s_i, X) := \delta _R(s_i, X) \cup \{s_j\}.$$ 

[reduce action] If $(s_i, X) \vdash (s_j, \alpha) \vdash (s_k, \epsilon)$ where $s_i, s_j, s_k \in Q_L$, $(\alpha \rightarrow \beta X, s_j) \in \delta _s(s_i, X)$, $\alpha \rightarrow \beta X \in P$, $\alpha \in (N \cup T)^+$,
$$\beta = \beta_1 \beta_2 \cdots \beta_{|\beta|}, \beta_i \in N \cup T \text{ and } X \in N \cup \{\epsilon\},$$
then
$$Q_R := Q_R \cup \{s_i, s_k\},$$
$$\delta _R(s_i, X) := \delta _R(s_i, X) \cup \{s_k\}.$$ 

Among the above actions, we use the following three relations $(s_i, \alpha)$, $\vdash$ and $\vdash$:

- $(s_i, \alpha)$ denotes that the current state of an LTD is $s_i$ and a next input symbol $\alpha \in (N \cup T)^+$.
- $\vdash$ denotes a transition of a pair of a state of an LTD and next input symbols. If $\delta _s(s_i, X) = s_j$ and $(\alpha \rightarrow \beta X, s_j) \notin \delta _s(s_i, X)$, then $(s_i, X, \alpha) \vdash (s_j, \alpha)$. On the other hand, if $(\alpha \rightarrow \beta X, s_j) \in \delta _s(s_i, X)$, then $(s_i, X, \gamma) \vdash (s_j, \alpha \gamma)$.
- $\vdash$ denotes a finite number of transitions of a pair consisting of a state of an LTD.

As an example of the RTD, we illustrate an RTD $M_R = (Q_R, \delta _R, s_0, s_{acc})$ of the grammar $G_1$.

First, we have $Q_R = \{s_0, s_2, s_4, s_7, s_8, s_9, s_{11}, s_{acc}\}$. 
lookahead string to reduce \( \alpha_1 \rightarrow \alpha_2 \), a nonterminal string \( w \) is necessary for a transition from a state \( s_j \) to a final state \( s_{acc} \) on an RTD.

Here \( head_k(\lambda) \) consists of the prefix string of length \( k \), and is defined by the following formula:

\[
head_k(\lambda) = \begin{cases} 
\xi, & \text{if } \lambda = \xi \eta, |\xi| = k, |\eta| \geq 0, \\
\lambda, & \text{if } |\lambda| \leq k,
\end{cases}
\]

where \( \lambda, \xi, \eta \in (N \cup T)^* \).

In addition, \( s_i \xrightarrow{a} s_k \) means a transition from \( s_i \) to \( s_k \) by a nonterminal string \( a \), it denotes \( \delta_R(s_i, A_1) \supseteq s_{i_1}, \delta_R(s_{i_1}, A_2) \supseteq s_{i_2}, \ldots, \delta_R(s_{i_{|\alpha|}}, A_{|\alpha|}) \supseteq s_k \) where \( \alpha = A_1 A_2 \ldots A_{|\alpha|} \).

For example, lookahead string \( LK_2 \) with length 2 is calculated as follows.

\[
LK_2([DE \rightarrow \cdot ED], s_0) = \left\{ \begin{array}{l} head_2(w^{\ast}) | s_0 \xrightarrow{DE} s_{acc} \\ s_0 \xrightarrow{\varepsilon} s_{acc} \end{array} \right\} = \{\varepsilon\}.
\]

The transition from a start state \( s_0 \) with the input string \( DE \) in the above formula reaches a final state \( s_{acc} \). Therefore, an unrestricted \( LR(k) \) parser does not need an input string, that is, its lookahead string is sufficient with an empty word.

On the other hand, the transition from a start state \( s_0 \) with input string \( E \) reaches a state \( s_4 \) in the following formula. Therefore, \( w \) is necessary for the transition from the state \( s_4 \) to the final state \( s_{acc} \).

\[
LK_2([E \rightarrow \cdot CB], s_0) = \left\{ \begin{array}{l} head_2(w^{\ast}) | s_0 \xrightarrow{E} s_4, \\ s_0 \xrightarrow{\varepsilon} s_{acc} \end{array} \right\} = \{AA, AB, AC, CB, D\$, EA, EC, EE \}.
\]

For each \( LR(0) \) item of \( s_0 \), if each length of lookahead strings is two, then we get lookahead strings \( LK_2 \)'s as follows.

\[
\begin{array}{l}
\delta(s_0, a) = \{(A \rightarrow a, s_0)\}
\delta(s_2, E) = \{(S \rightarrow DE, s_0)\}
\delta(s_2, a) = \{(A \rightarrow a, s_0)\}
\delta(s_4, E) = \{(DE \rightarrow ED, s_0), (DE \rightarrow ED, s_1)\}
\delta(s_4, a) = \{(A \rightarrow a, s_4), (C \rightarrow a, s_4)\}
\delta(s_4, d) = \{(D \rightarrow d, s_4)\}
\delta(s_7, A) = \{(A \rightarrow AA, s_0), (A \rightarrow AA, s_2),
\text{ (}A \rightarrow AA, s_3\text{)}\}
\delta(s_7, a) = \{(A \rightarrow a, s_7), (C \rightarrow a, s_7)\}
\delta(s_7, b) = \{(B \rightarrow b, s_7)\}
\delta(s_7, C) = \{(C \rightarrow AC, s_0),
\text{ (}C \rightarrow AC, s_2\text{)}, (C \rightarrow AC, s_4)\}
\delta(s_8, A) = \{(A \rightarrow AA, s_0), (A \rightarrow AA, s_2),
\text{ (}A \rightarrow AA, s_3\text{)}\}
\delta(s_8, a) = \{(A \rightarrow a, s_8), (C \rightarrow a, s_8)\}
\delta(s_8, C) = \{(C \rightarrow AC, s_0), (C \rightarrow AC, s_4)\}
\delta(s_9, B) = \{(E \rightarrow ABB, s_0), (E \rightarrow ABB, s_2),
\text{ (}E \rightarrow ABB, s_4\text{)}\}
\delta(s_9, b) = \{(B \rightarrow b, s_9)\}
\delta(s_{11}, B) = \{(E \rightarrow CB, s_0), (E \rightarrow CB, s_2),
\text{ (}E \rightarrow CB, s_4\text{)}\}
\delta(s_{11}, b) = \{(B \rightarrow b, s_{11})\}
\end{array}
\]

- For \( s_0 \in Q_R \), we are able to consider the transitions \((s_0, S) \vdash \{s_{acc}, e\}, (s_0, D) \vdash \{s_2, e\}, (s_0, E) \vdash \{s_4, e\}, (s_0, A) \vdash \{s_7, e\}\) and \( (s_0, C) \vdash \{s_{11}, e\} \).

We have five function values of \( \delta_R \) from the above five transitions, \( \delta_R(s_0, S) = \{s_{acc}\}, \delta_R(s_0, D) = \{s_2\}, \delta_R(s_0, E) = \{s_4\}, \delta_R(s_0, A) = \{s_7\}, \delta_R(s_0, C) = \{s_{11}\} \).

- For \( s_j \in Q_R \), we are able to consider transitions \((s_j, A) \vdash \{s_4, e\}, (s_j, A) \vdash \{s_7, e\}, (s_j, A) \vdash \{s_{11}, e\}\) and \( (s_j, B) \vdash \{s_9, e\} \).

And, we have the following two function values of \( \delta_R \), \( \delta_R(s_7, A) = \{s_4, s_7, s_6\}, \delta_R(s_7, B) = \{s_9\} \).

We illustrate a RTD in Fig. 1. Note that \( \delta_R(s_1, X) \in s_j \) is shown as a path from \( s_i \) to \( s_j \) with label \( X \) (\( s_i \xrightarrow{X} s_j \)).

### 2.3 Definition of lookahead strings

A set of lookahead strings of length \( k \) is constructed as follows, using an RTD \( M_R \).

\[
LK_k([\alpha_1 \rightarrow \cdot \alpha_2], s_i) = \left\{ \begin{array}{l} head_k(w^{\ast}) | s_i \xrightarrow{\alpha_1} s_j, s_j \xrightarrow{\alpha_2} s_{acc} \end{array} \right\}
\]

The above formula shows that an unrestricted \( LR(k) \) parser needs a nonterminal string \( w \) as a
$LK_2([S \rightarrow \cdot DE], s_0) = \{\}$

$LK_2([DE \rightarrow \cdot ED], s_0) = \{\}$

$LK_2([E \rightarrow \cdot ABB], s_0) = \{AA, AB, AC, CB, D\}$

$EE, EA, EC, ED, EE\}$

$LK_2([E \rightarrow \cdot CB], s_0) = \{AA, AB, AC, CB, D\}, EA, EC, ED, EE\}$$

$LK_2([A \rightarrow \cdot AA], s_0) = \{AA, AB, AC, BB, CB\}$

$LK_2([A \rightarrow \cdot a], s_0) = \{AA, AB, AC, BB, CB\}$

$LK_2([C \rightarrow \cdot AC], s_0) = \{BA, BC, BD, BE\}$

$LK_2([C \rightarrow \cdot a], s_0) = \{BA, BC, BD, BE\}$

On the other hand, for the case of each length of the lookahead strings is one, we have the following equations.

$LK_1([A \rightarrow \cdot a], s_0) = \{A, B, C\}$ and $LK_1([C \rightarrow \cdot a], s_0) = \{B\}$.

2.4 Definitions for an unrestricted $LR(k)$ grammar

First, for a phrase structure grammar $G = (N, T, P, S)$, we construct a LTD. And if all states of an LTD does not have the item like $[\alpha_1 \rightarrow \cdot a_2]$ or $[\beta_1 \rightarrow \cdot a_2]$ where $\dot{\cdot}$ is the end of a production rule in the same state of an LTD, then the grammar $G$ is defined as an unrestricted $LR(0)$ grammar. On the other hand, if a state of an LTD has an $LR(0)$ item, the grammar $G$ is defined as an unrestricted $LR(k)$ grammar, $k \geq 1$, which satisfies the following two conditions:

- (Condition 1) Let an LTD $M_L$ of an unrestricted $LR(0)$ grammar be $(Q_R, \delta_R, s_0, s_{\text{acc}}, S')$. Each production in $P$ has either of the following three forms:
  
  (a) $A \rightarrow a, A \in N, a \in T$.
  
  (b) $A \rightarrow \epsilon, A \in N$, where $\epsilon$ is the empty word.
  
  (c) $\lambda \rightarrow \mu, \lambda, \mu \in N^+$.

- (Condition 2) Each production $\alpha_i \rightarrow \beta \in P$, $\alpha_i \in N^+$, exists in the same state $s_j \in Q_L$, and $LK_k$ satisfies

  $$\bigcap_{\alpha_i \rightarrow \beta \in P} LK_k([\alpha_i \rightarrow \cdot \beta], s_j) = \emptyset.$$

Condition 1 is not essential as transformation, because any phrase structure grammar can be transformed to a grammar satisfying condition 1 which generates the same language.

For the grammar $G_1$ defined in section 2.1, $LK_1([A \rightarrow \cdot a], s_0) \cap LK_1([C \rightarrow \cdot a], s_0) = \{B\}$. Therefore, $G_1$ is not an unrestricted $LR(1)$ grammar. On the other hand, no string is shared among lookahead strings of length 2 (see Table 3). Therefore, $G_1$ is not an unrestricted $LR(1)$ grammar but an unrestricted $LR(2)$ grammar.

Table 3: The set of the lookahead strings $LK_2$ for $G_1$

| $LK_2([S \rightarrow \cdot DE], s_0) = \{\}$ |
| $LK_2([DE \rightarrow \cdot ED], s_0) = \{\}$ |
| $LK_2([E \rightarrow \cdot ABB], s_0) = \{AA, AB, AC, CB, D\}$ |
| $LK_2([E \rightarrow \cdot CB], s_0) = \{AA, AB, AC, CB, D\}$ |

3 The transformation from an unrestricted $LR(k)$ grammar, $k > 1$, to an unrestricted $LR(1)$ grammar

The transformation [3] of a conventional $LR(k)$ grammar replaces a pair of a nonterminal symbol corresponding to a root of a subtree of deducible parse trees and a lookahead string of its nonterminal symbol with a nonterminal symbol. On the other hand, the transformation in this paper replaces a head of lookahead strings with a nonterminal symbol, and its replacement is repeated until the length of lookahead strings becomes 1. In this section, we illustrate how to transform an unrestricted $LR(k)$
grammar, \( k > 1 \), to an unrestricted \( LR(1) \) grammar.

For an unrestricted \( LR(k) \) grammar \( G, k > 1 \), a set of lookahead strings satisfies

\[
LK_{k}[\{\lambda_{1} \rightarrow \cdot \mu], s_{i}\} \cap LK_{k}[\{\lambda_{2} \rightarrow \cdot \mu], s_{j}\} = \emptyset
\]

and

\[
LK_{k}[\{\lambda_{1} \rightarrow \cdot \mu], s_{i}\} \cap LK_{k}[\{\lambda_{2} \rightarrow \cdot \mu], s_{j}\} \ni \alpha
\]

where \( \alpha = A\alpha', \alpha, \alpha' \in (N \cup T)^+ \) and \( A \in N \).

Clearly, we have \( LK_{k}[\{\lambda_{1} \rightarrow \cdot \mu], s_{i}\} \ni B \) and \( LK_{k}[\{\lambda_{2} \rightarrow \cdot \mu], s_{j}\} \ni B \).

To reduce from a lookahead string \( \alpha \) to a nonterminal symbol, we prepare a new nonterminal symbol \( B \notin N \) and replace production rules so that \( LK_{k}[\{\lambda_{1} \rightarrow \cdot \mu], s_{i}\} \ni B \) and \( LK_{k}[\{\lambda_{2} \rightarrow \cdot \mu], s_{j}\} \ni B \).

The following transformation describes precise conditions and replacement of production rules.

**[Reduction of multiple symbols]**

If \( LK_{k}[\{\mu_{1} \rightarrow \cdot \beta], s_{i}\} \ni B \) and \( LK_{k}[\{\mu_{2} \rightarrow \cdot \beta], s_{j}\} \ni B \) where \( \mu_{1} \rightarrow \beta, \mu_{2} \rightarrow \beta \in \mathcal{P}, s_{i} \in Q_{R}, s_{j} \in \{s_{k}, s_{t} \in Q_{R}\}, \{[\alpha \rightarrow \alpha_{1} \cdot A_{\gamma}] \} \in s_{j} \)

and \( \gamma_{i} = \alpha_{i+1} \gamma' \), then

\[
\text{(1) } P := P - \{\alpha \rightarrow \alpha_{1}A_{\gamma}\},
\text{(2) } P := P \cup \{\alpha \rightarrow \alpha_{1}A_{\gamma}B_{\alpha_{i+1}} \rightarrow \gamma_{i+1},
B_{\alpha_{i+2}} \rightarrow \gamma_{i+2}, \ldots, B_{\alpha_{n_{2}}} \rightarrow \gamma_{n_{2}}\},
\text{(3) } N := N \cup \{B\}.
\]

The above transformation is repeated until the grammar \( G \) becomes an unrestricted \( LR(1) \) grammar, i.e., until the following condition is satisfied:

For each \( \beta \in N^+, s_{i} \in Q_{R}, \)

\[
\bigcap_{\alpha: \alpha \rightarrow \alpha_{1} \beta} LK_{k}[\{\alpha \rightarrow \cdot \beta], s_{i}\} = \emptyset.
\]

Note that the above transformation does not depend on the length \( k \) of a lookahead string and works even if \( k = \infty \).

### 4 Properties of a transformation

In the above section, we showed the transformation of an unrestricted \( LR(k) \) grammar. In this section, we shall show some properties of the transformation.

**Theorem 1:** The transformation of the grammar introduced in section 3 does not change the language \( L \) generated by the grammar if \( L \) is in the class of recursive languages.

**Proof.** Let \( G' \) be the resulting grammar obtained from a grammar \( G \), by an application of

**[Reduction of multiple symbols]** in section 3, where \( \alpha \rightarrow \alpha_{1}A_{\gamma} \) is deleted and \( \alpha \rightarrow \alpha_{1}B_{\gamma} \) and \( B_{\alpha_{i+1}} \rightarrow \gamma_{i+1} \) are appended for production rules

\[
\alpha \rightarrow \alpha_{1}A_{\gamma}, A_{\alpha_{i+1}} \rightarrow \gamma_{i+1} \text{ of } G,
\]

Then we shall show that \( L(G) = L(G') \).

We consider the following derivation from a string \( w_{1}\alpha w_{2} \), by production rules \( \alpha \rightarrow \alpha_{1}A_{\gamma} \) and \( A_{\alpha_{i+1}} \rightarrow \gamma_{i+1} \) of \( G' \).

\[
w_{1}\alpha w_{2} \Rightarrow w_{1}\alpha_{1}A_{\gamma}w_{2} = w_{1}\alpha_{1}A_{\alpha_{i+1}}\gamma'w_{2}
\]

We also consider the derivation shown as follows by production rules \( \alpha \rightarrow \alpha_{1}B_{\gamma} \) and \( B_{\alpha_{i+1}} \rightarrow \gamma_{i+1} \) of \( G' \).

\[
w_{1}\alpha w_{2} \Rightarrow w_{1}\alpha_{1}B_{\gamma}w_{2} = w_{1}\alpha_{1}B_{\alpha_{i+1}}\gamma'w_{2}
\]

Hence, the same strings are derived from \( w_{1}\alpha w_{2} \) by \( G \) and \( G' \), respectively. Thus we conclude that

\( L(G) = L(G') \) if \( L(G) \) is in the class of recursive languages. By repeating the above discussion \( m \) times where \( m \) is the number of applications of

**[Reduction of multiple symbols]** in section 4 to obtain an unrestricted \( LR(1) \) grammar, we have shown that this theorem holds.

**Theorem 2:** Any phrase structure grammar \( G \) where \( L(G) \) is in the class of recursive languages can be transformed to either an unrestricted \( LR(0) \) grammar or an unrestricted \( LR(1) \) grammar.

**Proof.** By the definition of the unrestricted \( LR(k) \) grammar, it is easy to see that we can construct an unrestricted \( LR(k) \) grammar for any phrase structure grammar which generates the same language if we allow \( k \) to be \( \infty \). The transformation from an unrestricted \( LR(k) \), \( k > 1 \), grammar to an unrestricted \( LR(1) \) grammar introduced in section 4 works even if \( k = \infty \), as the transformation procedure does not depend on \( k \). Thus consider the unrestricted \( LR(k) \) grammar \( G \) which generates the same language as the given phrase structure grammar. If \( k > 1 \), then, by Theorem 1, there is an unrestricted \( LR(1) \) grammar which generates the same language as the given phrase structure language when it is in the class of recursive languages. The case of \( k = 0 \) is trivial. Thus we have proved the theorem.

### 5 An example of the transformation and parsing

The lookahead strings \( LK_{1} \)'s of the grammar \( G_{1} \) consist of

\[
LK_{1}[\{A \rightarrow \cdot \alpha], s_{0}\} = \{A, B, C\}
\]

\[
LK_{1}[\{C \rightarrow \cdot \alpha], s_{0}\} = \{B\}.
\]
Figure 2: The LTD of G

From the above lookahead strings, we have the following equation.

\[ LK_1([A \rightarrow \alpha], s_0) \cap LK_1([C \rightarrow \alpha], s_0) = \{B\} \]

Hence, to reduce each length of the lookahead strings, we prepare a new nonterminal symbol \( F \notin N_1 \) and replace B with F such that

\[
\begin{align*}
LK_1([A \rightarrow \alpha], s_0) & = \{A, F, C\}, \\
LK_1([C \rightarrow \alpha], s_0) & = \{B\}.
\end{align*}
\]

From the RTD of \( G_1 \) (see Fig. 1), we have \( \delta_R(A, s_0) = s_7 \) and \( \delta_R(C, s_0) = s_{11} \). Therefore, there is B followed by a dot (\( \cdot \)) in an \( LR(0) \) item of \( s_7 \). In fact, there is the \( LR(0) \) item \([E \rightarrow A \cdot BB]\) in \( s_7 \). Consequently, we replace a production rule \( E \rightarrow ABB \) with production rules \( E \rightarrow AFB \) and \( F \rightarrow b \). Here a new nonterminal symbol \( F \) is followed by a dot (\( \cdot \)) in an \( LR(0) \) item.

- \( P := P - \{E \rightarrow ABB\} \)
- \( P := P \cup \{E \rightarrow AFB, F \rightarrow b\} \)
- \( N := N \cup \{F\} \)

By the above transformation, we have the following grammar \( G_2 = (\{S, A, B, C, D, E, F\}, \{a, b, d\}, \{S \rightarrow DE, DE \rightarrow ED, E \rightarrow CB, A \rightarrow AA, A \rightarrow a, C \rightarrow CA, C \rightarrow a, B \rightarrow b, E \rightarrow AFB, F \rightarrow b, D \rightarrow d\}, \{S\}) \). In addition, we define an example of the LTD in Fig. 2 and the RTD in Fig. 3. Clearly, \( LK_1([A \rightarrow \alpha], s_0) \cap LK_1([C \rightarrow \alpha], s_0) = \emptyset \), in short, the grammar \( G_2 \) satisfies

\[ \bigcap_{\alpha: \alpha \rightarrow \beta \in P} LK_1([\alpha \rightarrow \beta], s_t) = \emptyset. \]

The resulting grammar \( G_2 \) by the transformation is an unrestricted \( LR(1) \) grammar.

6 Concluding Remarks

In this paper, we showed how to transform an unrestricted \( LR(k) \) grammar to an unrestricted \( LR(1) \) grammar without changing the generated language which is in th class of recursive languages. Although a conventional \( LR(k), k \geq 1 \), grammar can be transformed to \( LR(1) \) only when \( k \) is a finite integer. Even if \( k = \infty \) as was shown in Theorem 2, we can transform an unrestricted \( LR(k) \), \( k \geq 1 \), grammar to an unrestricted \( LR(1) \) grammar.

References