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

(New Developments of Theory of Computation and Algorithms)

Author(s)
Shiina, Hiromitsu; Masuyama, Shigeru

Citation
数理解析研究所講究録

Issue Date
2001-05

URL
http://hdl.handle.net/2433/41017

Type
Departmental Bulletin Paper

Textversion
publisher

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 non-context-free languages in natural language processing. Some parsers [1,2,4] were developed for classes larger than that of context-free languages. Vold'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 deductible 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 $M_L = (Q_L, \delta_s, \delta_t, s_0, s_{acc}, S')$. Here, $Q_L = \{s_0, s_1, \ldots, s_{n-1}, s_n\}$ is a set of states, $\delta_s$ is a set of shift functions satisfying $\delta_s(s, X) = s', s, s' \in Q_L, X \in N \cup T$, $\delta_t$ is a set of reduce functions satisfying $\delta_t(s, X) = \{(\lambda \rightarrow \mu, s') | s' \in Q_L, \lambda \rightarrow \mu \in P\}$, $s_0$ is an initial state, $s_{acc}$ 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 $[\lambda \rightarrow \mu_1 \mu_2]$, $[\lambda \rightarrow \mu_1]$, and $[\lambda \rightarrow \mu_2]$. 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 $s_{acc}$.

$s_0 := \{(S' \rightarrow \cdot S)\}$, $s_{acc} := \{(S' \rightarrow \cdot S)\}$ and $\delta_t(s_0, S) := s_{acc}$.

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

- for each $X\xi \rightarrow \eta \in P$ where $\xi, \eta \in (N \cup T)^*$, $s_i := s_i \cup \{(X\xi \rightarrow \cdot \eta)\}$.
- for each $s_j \in Q_L$ such that $[\lambda \rightarrow \mu_1 X \cdot \mu_2] \not\in s_j$,
create a new state $s_k$,
$$Q_L := Q_L \cup \{s_k\} \quad \text{and} \quad s_k := s_k \cup \{[\lambda \rightarrow \mu_1 X \cdot \mu_2]\}.$$  

**[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 $\delta_s(\delta_s(\ldots \delta_s(s_k, \mu_1), \mu_2) \ldots \mu_\|\mu\|), X) = s_i$ where $s_i, s_j, s_k \in Q_L$, $\mu = \mu_1 \mu_2 \ldots \mu_\|\mu\|$, $\mu_1, \mu_2, \ldots, \mu_\|\mu\| \in N \cup T$, 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
$$\delta_r(s_i, \epsilon) := \delta_r(s_i, \epsilon) \cup \{[\lambda \rightarrow \epsilon, s_i]\}.$$  

Here, we use a grammar $G_1 = \{S, A, B, C, D, E\}$, where $Q_L = \{s_0, s_1, s_2, s_{\text{ace}}\}$. The functions $\delta_r$ and $\delta_s$ are shown in Tables 1 and 2.

In addition, contents of states are the following $LR(0)$ items,
$$s_0 = \{[S' \rightarrow S], [S \rightarrow DE], [DE \rightarrow ED], [E \rightarrow ABB], [E \rightarrow CB], [A \rightarrow A], [A \rightarrow B], [C \rightarrow AC], [C \rightarrow d]\},$$
$$s_1 = \{[A \rightarrow a], [C \rightarrow a\alpha], \ldots, [S \rightarrow D \cdot E], [E \rightarrow ABB], [E \rightarrow CB], [A \rightarrow AA], [C \rightarrow AC]\},$$
$$s_2 = \{[S' \rightarrow S]\},$$
$$s_{\text{ace}} = \{[S' \rightarrow S]\}.$$

### 2.2 Definition of an RTD

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

An RTD $M_R$ is a 4-tuple $(Q_R, \delta_R, s_0, s_{\text{ace}})$ 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, $s_{\text{ace}}$ 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 $(s_i, X) \vdash (s_j, \epsilon)$, 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 $(s_i, X) \vdash (s_j, \alpha)$ or $(s_i, X) \vdash (s_j, \alpha)$, where $s_i, s_j \in Q_L$, then
$$Q_R := Q_R \cup \{s_i, s_j\},$$
$$\delta_R(s_i, X) := \delta_R(s_i, X) \cup \{s_k\}.$$  

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

- $(s_i, \alpha)$ denotes the current state of an LRT is $s_i$ and a next input symbol $\alpha \in (N \cup T)^+$.

- $\vdash$ indicates a transition of a pair of a state of an LRT and next input symbols. If $\delta_s(s_i, X) = s_j$ and $(\alpha \rightarrow \beta X, s_j) \notin \delta_r(s_i, X)$, then $(s_i, \alpha) \vdash (s_j, \alpha)$. On the other hand, if $(\alpha \rightarrow \beta X, s_j) \in \delta_r(s_i, X)$, then $(s_i, \alpha) \vdash (s_j, \alpha \gamma)$.

- $\vdash$ denotes a finite number of transitions of a pair consisting of a state of an LRT.

As an example of the RTD, we illustrate an RTD $M_R = (Q_R, \delta_R, s_0, s_{\text{ace}})$ of the grammar $G_1$.

First, we have $Q_R = \{s_0, s_2, s_4, s_7, s_8, s_9, s_{11}, s_{\text{ace}}\}$.
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, \( s_i \xrightarrow{\alpha} s_k \) means a transition from \( s_i \) to \( s_k \) by a nonterminal string \( \alpha \), it denotes \( \delta_R(s_i, A_1) \ni s_k \), \( \delta_R(s_i, A_2) \ni s_k \), \ldots, \( \delta_R(s_i, A_{|\alpha|}) \ni s_k \) where \( \alpha = A_1A_2 \ldots A_{|\alpha|} \), \( A_1, A_2, \ldots, A_{|\alpha|} \in N \) and \( s_i, s_{i1}, \ldots, s_{i|\alpha|}, s_k \in Q_R \).

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

\[
LK_2([DE \rightarrow \cdot ED], s_0) = \begin{cases} 
head_2(w^\epsilon) & |s_0| \xrightarrow{DE} \in s_{acc}, \\
s_0 & \epsilon \xrightarrow{s_{acc}} s_{acc}
\end{cases} = \{\$\}.
\]

The transition from a start state \( s_0 \) with 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 \( s_{acc} \).

\[
LK_2([E \rightarrow \cdot CB], s_0) = \begin{cases} 
head_2(w^\epsilon) & |s_0| \xrightarrow{E} s_4, \\
s_4 & \epsilon \xrightarrow{s_{acc}} s_{acc}
\end{cases} = \{AA, AB, AC, CB, D$, $EA, EC, EE\}.
\]

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.
$LK_2([S \rightarrow DE], s_0) = \{\}$
$LK_2([DE \rightarrow \cdot DE], s_0) = \{\}$
$LK_2([E \rightarrow \cdot ABB], s_0) = \{AA, AB, AC, CB, D\} \\
\text{EA, } EC, ED, EE \} \]
$LK_2([E \rightarrow \cdot CB], s_0) = \{AA, AB, AC, CB, D\} \\
\text{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 an LTD. And if all states of an LTD does not have the item like $[\alpha_1 \rightarrow \alpha_2]$ or

$[[\beta_1 \rightarrow \beta_2]]$ where dot($) 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 $(Q_R, \delta_R, s_0, s_{acc}, S')$. Each production in $P$ has either of the following three forms:

  (a) $A \rightarrow \alpha, A \in N, \alpha \in T$.

  (b) $\lambda \rightarrow \alpha, A \in N$, where $\epsilon$ is the empty word.

  (c) $\alpha \rightarrow \mu, \lambda, \mu \in N^+$. 

- **(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 $G_1$.

<table>
<thead>
<tr>
<th>Condition 1</th>
<th>Condition 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$LK_2([S \rightarrow DE], s_0) = {}$</td>
<td>$LK_2([DE \rightarrow \cdot DE], s_0) = {}$</td>
</tr>
<tr>
<td>$LK_2([E \rightarrow \cdot ABB], s_0) = {AA, AB, AC, CB, D} \text{EA, } EC, ED, EE }$</td>
<td>$LK_2([E \rightarrow \cdot CB], s_0) = {AA, AB, AC, CB, D} \text{EA, } EC, ED, EE }$</td>
</tr>
<tr>
<td>$LK_2([A \rightarrow \cdot AA], s_0) = {AA, AB, AC, BB, CB }$</td>
<td>$LK_2([A \rightarrow \cdot a], s_0) = {AA, AB, AC, BB, CB }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot AC], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([E \rightarrow \cdot ABB], s_0) = {}$</td>
</tr>
<tr>
<td>$LK_2([E \rightarrow \cdot CB], s_0) = {}$</td>
<td>$LK_2([E \rightarrow \cdot CB], s_0) = {}$</td>
</tr>
<tr>
<td>$LK_2([A \rightarrow \cdot AA], s_0) = {AA, AB, AC, AD, AE, BB, CB }$</td>
<td>$LK_2([A \rightarrow \cdot a], s_0) = {AA, AB, AC, AD, AE, BB, CB }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot AC], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([E \rightarrow \cdot \cdot \cdot], s_0) = {}$</td>
</tr>
<tr>
<td>$LK_2([DE \rightarrow \cdot ED], s_0) = {}$</td>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
</tr>
<tr>
<td>$LK_2([E \rightarrow \cdot ABB], s_0) = {AA, AB, AC, CB, D} \text{EA, } EC, ED, EE }$</td>
<td>$LK_2([E \rightarrow \cdot CB], s_0) = {AA, AB, AC, CB, D} \text{EA, } EC, ED, EE }$</td>
</tr>
<tr>
<td>$LK_2([A \rightarrow \cdot AA], s_0) = {AA, AB, AC, AD, AE, BB, CB }$</td>
<td>$LK_2([A \rightarrow \cdot a], s_0) = {AA, AB, AC, AD, AE, BB, CB }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot AC], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
</tr>
<tr>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
<td>$LK_2([C \rightarrow \cdot a], s_0) = {BA, BC, BD, BE }$</td>
</tr>
</tbody>
</table>

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

\[
LK_k([\lambda_1 \rightarrow \cdot \mu], s_i) \cap LK_k([\lambda_2 \rightarrow \cdot \mu], s_i) = \emptyset \quad \text{and} \quad LK_k([\lambda_1 \rightarrow \cdot \mu], s_i) \cap LK_k([\lambda_2 \rightarrow \cdot \mu], s_i) \ni \alpha
\]

where \( \alpha = \delta_1, \delta_1' \in (N \cup T)^+ \) and \( A \in N \). Clearly, we have \( LK_k([\lambda_1 \rightarrow \cdot \mu], s_i) \ni B \) and \( LK_k([\lambda_2 \rightarrow \cdot \mu], s_i) \ni A \). The following transformation describes precise conditions and replacement of production rules.

**[Reduction of multiple symbols]**
If \( LK_k([\lambda_1 \rightarrow \cdot \beta], s_i) \cap LK_k([\lambda_2 \rightarrow \cdot \beta], s_i) \ni A \)
where \( \delta_1 \rightarrow \beta, \delta_2 \rightarrow \beta \in P \), \( s_i \in Q, s_j \in \{s_1, s_2, \ldots, s_k\} \cap Q \), \( \{\alpha \rightarrow \alpha_1 \cdot \gamma_i\} \ni s_j \) and \( \gamma_i = \alpha_1 \gamma_i \), then

\[
\begin{align*}
(1) P & := P - \{\alpha \rightarrow \alpha_1 \gamma_i\}, \\
(2) P & := P \cup \{\alpha \rightarrow \alpha_1 B \gamma_i, \beta \rightarrow \gamma_{i+1}, \\
& \quad B \delta_{i+2} \rightarrow \gamma_{i+2}, \ldots, B \delta_{n_2} \rightarrow \gamma_{n_2}\}, \\
(3) N & := N \cup \{B\}.
\end{align*}
\]

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 \( \beta \in N^* \), \( s_i \in Q \),

\[
\bigcap_{\alpha: \alpha \rightarrow \beta \in P} LK_k([\alpha \rightarrow \cdot \beta], s_i) = \emptyset.
\]

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 \( \alpha \rightarrow \alpha_1 A \gamma_i \) is deleted and \( \alpha \rightarrow \alpha_1 B \gamma_i \) and \( B \alpha_{i+1} \rightarrow \gamma_{i+1} \) are appended for production rules \( \alpha \rightarrow \alpha_1 A \gamma_i, A \alpha_{i+1} \rightarrow \gamma_{i+1} \) of \( G \), where \( \gamma_i = \alpha_1 \gamma_i \). 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 \( \alpha \rightarrow \alpha_1 A \gamma_i \) and \( A \alpha_{i+1} \rightarrow \gamma_{i+1} \) of \( G \).

\[
\begin{align*}
& w_1 \alpha w_2 \Rightarrow w_1 \alpha_1 A \gamma_i w_2 = w_1 \alpha_1 A \alpha_{i+1} \gamma_i' w_2 \\
& \Rightarrow w_1 \alpha_1 \gamma_{i+1} \gamma_i' w_2
\end{align*}
\]

We also consider the derivation shown as follows by production rules \( \alpha \rightarrow \alpha_1 B \gamma_i \) and \( B \alpha_{i+1} \rightarrow \gamma_{i+1} \) of \( G' \).

\[
\begin{align*}
& w_1 \alpha w_2 \Rightarrow w_1 \alpha_1 B \gamma_i w_2 = w_1 \alpha_1 B \alpha_{i+1} \gamma_i' w_2 \\
& \Rightarrow w_1 \alpha_1 \gamma_{i+1} \gamma_i' w_2
\end{align*}
\]

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. \( \square \)

**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. \( \square \)

5 An example of the transformation and parsing

The lookahead strings \( LK_1 \)'s of the grammar \( G_1 \) consist of

\[
LK_1([A \rightarrow \cdot a], s_0) = \{A, B, C\} \quad \text{and} \quad LK_1([C \rightarrow \cdot a], s_0) = \{B\}.
\]
From the above lookahead strings, we have the following equation.

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

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

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

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, \( LK_1([A \rightarrow \cdot a], s_0) \cap LK_1([C \rightarrow \cdot a], s_0) \) = \( \emptyset \), in short, the grammar \( G_2 \) satisfies

\[ \bigcap_{\alpha: \alpha \rightarrow \beta, \beta \in P} \{Sarrow \alpha \}
\]

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 \( \text{th} \) 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


