A Method of Incorporating Bigram Constraints into an LR Table and Its Effectiveness in Natural Language Processing

Hiroki Imai and **Hozumi Tanaka**
Graduate School of Information Science and Technology
Tokyo Institute of Technology
2-12-1 O-okayama, Meguro, Tokyo 152 Japan
{imai,tanaka}@cs.titech.ac.jp

Abstract

In this paper, we propose a method for constructing bigram LR tables by way of incorporating bigram constraints into an LR table. Using a bigram LR table, it is possible for a GLR parser to make use of both bigram and CFG constraints in natural language processing.
Applying bigram LR tables to our GLR method has the following advantages:
(1) Language models utilizing bigram LR tables have lower perplexity than simple bigram language models, since local constraints (bigram) and global constraints (CFG) are combined in a single bigram LR table.
(2) Bigram constraints are easily acquired from a given corpus. Therefore data sparseness is not likely to arise.
(3) Separation of local and global constraints keeps down the number of CFG rules.
The first advantage leads to a reduction in complexity, and as the result, better performance in GLR parsing.
Our experiments demonstrate the effectiveness of our method.

1 Introduction

In natural language processing, stochastic language models are commonly used for lexical and syntactic disambiguation (Fujisaki et al., 1991; Franz, 1996). Stochastic language models are also helpful in reducing the complexity of speech and language processing by way of providing probabilistic linguistic constraints (Lee, 1989).

N-gram language models (Jelinek, 1990), including bigram and trigram models, are the most commonly used method of applying local probabilistic constraints. However, context-free grammars (CFGs) produce more global linguistic constraints than N-gram models. It seems better to combine both local and global constraints and use them both concurrently in natural language processing. The reason why N-gram models are preferred over CFGs is that N-gram constraints are easily acquired from a given corpus. However, the larger N is, the more serious the problem of data sparseness becomes.

CFGs are commonly employed in syntactic parsing as global linguistic constraints, since many efficient parsing algorithms are available. GLR (Generalized LR) is one such parsing algorithm that uses an LR table, into which CFG constraints are precompiled in advance (Knuth, 1965; Tomita, 1986). Therefore if we can incorporate N-gram constraints into an LR table, we can make concurrent use of both local and global linguistic constraints in GLR parsing.

In the following section, we will propose a method that incorporates bigram constraints into an LR table. The advantages of the method are summarized as follows:

First, it is expected that this method produces a lower perplexity than that for a simple bigram language model, since it is possible to utilize both local (bigram) and global (CFG) constraints in the LR table. We will evidence this reduction in perplexity by considering states in an LR table for the case of GLR parsing.

Second, bigram constraints are easily acquired from smaller-sized corpora. Accordingly, data sparseness is not likely to arise.

Third, the separation of local and global constraints makes it easy to describe CFG rules, since CFG writers need not take into account tedious descriptions of local connection constraints within th CFG[1].

2 CFG, Connection Matrix and LR table

2.1 Relation between CFG and Connection Constraints

Figure 1 represents a situation in which a_i and b_j are adjacent to each other, where a_i belongs to Set_I ($i = 1, \cdots, I$) and b_j belongs to Set_J ($j = 1, \cdots, J$). Set_I

[1](Tanaka et al., 1997) reported that the separate description of local and global constraints reduced the CFG rules to one sixth of their original number.

Hiroki Imai and Hozumi Tanaka (1998) **A Method of Incorporating Bigram Constraints into an LR Table and Its Effectiveness in Natural Language Processing.** In D.M.W. Powers (ed.) *NeMLaP3/CoNLL98: New Methods in Language Processing and Computational Natural Language Learning,* ACL, pp 225-233.

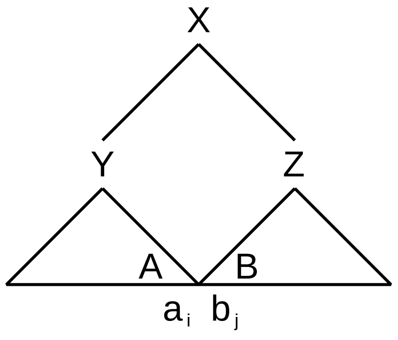

Figure 1: Connection check by CFG

and Set_J are defined by *last1* (Y) and *first1* (Z) (Aho et al., 1986), respectively. If $a \in Set_I$ and $b \in Set_J$ happen not to be able to occur in this order, it becomes a non-trivial task to express this adjacency restriction within the framework of a CFG.

One solution to this problem is to introduce a new nonterminal symbol A_i for each a_i and a nonterminal symbol B_j for each b_j. We then add rules of the form $A \rightarrow A_i$ and $A_i \rightarrow a_i$, and $B \rightarrow B_j$ and $B_j \rightarrow b_j$. As a result of this rule expansion, the order of the number of rules will become $I \times J$ in the worst case. The introduction of such new nonterminal symbols leads to an increase in grammar rules, which not only makes the LR table very large in size, but also diminishes efficiency of the GLR parsing method.

The second solution is to augment $X \rightarrow Y\ Z$ with a procedure that checks the connection between a_i and b_j. This solution can avoid the problem of the expansion of CFG rules, but we have to take care of the information flow from the bottom leaves to the upper nodes in the tree, Y, Z, and X.

Neither the first nor the second solution are preferable, in terms of both efficiency of GLR parsing and description of CFG rules. Additionally, it is a much easier task to describe local connection constraints between two adjacent terminal symbols by way of a connection matrix such as in Figure 2, than to express these constraints within the CFG.

The connection matrix in Figure 2 is defined as:

$$\mathit{Connect}(a_i, b_j) = \begin{cases} 1 & \text{if } b_j \text{ can follow } a_i \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

The best solution seems to be to develop a method that can combine both a CFG and a connection matrix, avoiding the expansion of CFG rules. Consequently, the size of the LR table will become smaller and we will get better GLR parsing performance. In the following section, we will propose one such method. Note that we are considering connections between preterminals rather than words. Thus, we will have $\mathit{Connect}(a_i, b_j) = 0$ in the preterminal connection matrix similarly to the case of words.

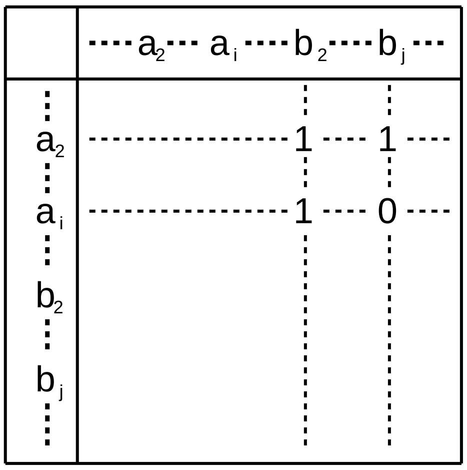

Figure 2: Connection matrix

2.2 Relation between the LR Table and Connection Matrix

First we discuss the relation between the LR table and a connection matrix. The action part of an LR table consists of lookahead symbols and states. Let a shift action *sh m* be in state l with the lookahead symbol a. After the GLR parser executes action *sh m*, the symbol a is pushed onto the top of the stack and the GLR parser shifts to state m. Suppose there is an action A in state m with lookahead b (see Figure 3). The action A is executable if $\mathit{Connect}(a, b) \neq 0$ (b can follow a), whereas if $\mathit{Connect}(a, b) = 0$ (b cannot follow a), the action A in state m with lookahead b is not executable and we can remove it from the LR table as an invalid action. Removing such invalid actions enables us to incorporate connection constraints into the LR table in addition to the implicit CFG constraints.

In section 3.2, we will propose a method that integrates both bigram and CFG constraints into an LR table. After this integration process, we obtain a table called a bigram LR table.

3 Integration of Bigram and CFG Constraints into an LR Table

3.1 The Definition of a Probabilistic Connection Matrix

A close relation exists between bigrams and connection matrices, in that the bigram probability $P(b|a)$ corresponds to the matrix element of $\mathit{Connect}(a, b)$. A connection matrix incorporating bigram probabilities is called a probabilistic connection matrix, in which $\mathit{Connect}(a, b) = 0$ still means b cannot follow a, but instead of connection matrix entries having a binary value of 0 or 1, a probability is associated with each element. This is then used to construct a probabilistic LR table.

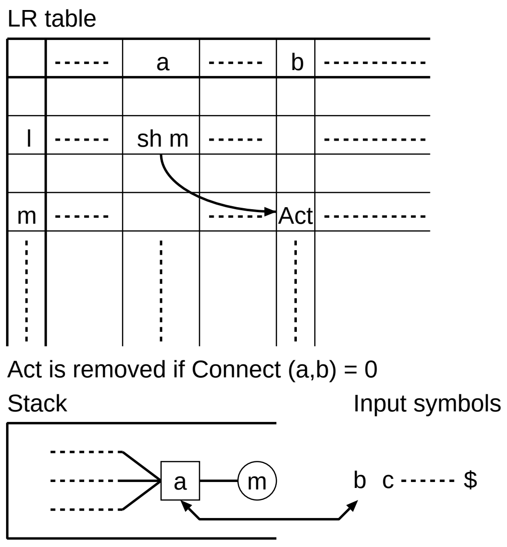

Figure 3: LR table and Connection Constraints

The N-gram model is the most commonly used probabilistic language model, and it assumes that a symbol sequence can be described by a higher order Markov process. The simplest N-gram model with $N = 2$ is called a bigram model, and approximates the probability of a string $\mathbf{X} = x_1x_2x_3 \cdots x_n$ as the product of conditional probabilities:

$$P(\mathbf{X}) = P(x_1|\#)P(x_2|x_1) \cdots P(x_n|x_{n-1})P(\$|x_n) \quad (2)$$

In the above expression, "#" indicates the sentence beginning marker and "\$" indicates the sentence ending marker. The above bigram model can be represented in a *probabilistic connection matrix* defined as follows.

DEFINITION 1 (*probabilistic connection matrix*)
Let $G = (V_N, V_T, P, S)$ be a context-free grammar. For $\forall a, b \in V_T$ (the set of terminal symbols), the *probabilistic connection matrix* named *PConnect* is defined as follows.

$$PConnect(a, b) = P(b|a) \quad (3)$$

where $P(b|a)$ is a conditional probability and $\sum_{b \in V_T} P(b|a) = 1$.

$PConnect(a, b) = 0$ means that a and b cannot occur consecutively in the given order. $PConnect(a, b) \neq 0$ means b can follow a with probability $P(b|a)$.

3.2 An algorithm to construct a bigram LR table

An algorithm to construct a probabilistic LR table, combining both bigram and CFG constraints, is given in Algorithm 1:

Algorithm 1

Input: A CFG $G = (V_N, V_T, P, S)$ and a probabilistic connection matrix *PConnect.*

Output: An LR table T with CFG and bigram constraints.

Method:

```
Step 1 Generate an LR table $T_0$ from the given CFG $G$.
Step 2 Removal of actions:
    For each shift action sh $m$ with lookahead $a$ in the LR table $T_0$, delete actions in the state $m$ with lookahead $b$ if $PConnect(a, b) = 0$.
Step 3 Constraint Propagation (Tanaka et al., 1994):
    Repeat the following two procedures until no further actions can be removed:
        1. Remove actions which have no succeeding action,
        2. Remove actions which have no preceding action.
Step 4 Compact the LR table if possible.
Step 5 Incorporation of bigram constraints into the LR table:
    For each shift action sh $m$ with lookahead $a$ in the LR table $T_0$, let
        $P = \sum_{i=1}^{N} PConnect(a, b_i)$
    where $\{b_i : i = 1, \cdots, N\}$ is the set of lookaheads for state $m$. For each action $A_j$ in state $m$ with lookahead $b_i$, assign a probability $p$ to action $A_j$:
        $p = \frac{P(b_i|a)}{P \times n} = \frac{PConnect(a, b_i)}{P \times n}$
    where $n$ is the number of conflict actions in state $m$ with lookahead $b_i$. The denominator is clearly a normalization factor.
Step 6 For each shift action $A$ with lookahead $a$ in state 0, assign $A$ a probability $p = P(a|\#)$, where "#" is the sentence beginning marker.
Step 7 Assign a probability $p = 1/n$ to each action $A$ in state $m$ with lookahead symbol $a$ that has not been assigned a probability, where $n$ is the number of conflict actions in state $m$ with lookahead symbol $a$.
Step 8 Return the LR table $T$ produced at the completion of Step 7 as the Bigram LR table.
```

As explained above, the removal of actions at Step 2 corresponds to the operation of incorporating connection constraints into an LR table. We call Step 3 Constraint Propagation, by which the size of the LR table is reduced (Tanaka et al., 1994). As many

(1)	$S \rightarrow X\ Y$	(6)	$A \rightarrow a1$
(2)	$X \rightarrow A$	(7)	$A \rightarrow a2$
(3)	$X \rightarrow A\ B$	(8)	$B \rightarrow b1$
(4)	$Y \rightarrow A$	(9)	$B \rightarrow b2$
(5)	$Y \rightarrow b1\ A$		

Figure 4: Grammar G_1

	a1	*a2*	*b1*	*b2*	$
#	*0.6*	*0.4*	*0.0*	*0.0*	*0.0*
a1	*0.0*	*0.0*	*0.0*	*1.0*	*0.0*
a2	*0.0*	*0.0*	*0.3*	*0.0*	*0.7*
b1	*0.0*	*0.1*	*0.9*	*0.0*	*0.0*
b2	*0.0*	*0.0*	*1.0*	*0.0*	*0.0*

Figure 5: Probabilistic connection matrix M_1

actions are removed from the LR table during Steps 2 and 3, it becomes possible to compress the LR table in Step 4. We will demonstrate one example of this process in the following section.

It should be noted that the above algorithm can be applied to any type of LR table, that is a canonical LR table, an LALR table, or an SLR table.

4 An Example

4.1 Generating a Bigram LR Table

In this section, we will provide a simple example of the generation of a bigram LR table by way of applying Algorithm 1 to both a CFG and a probabilistic connection matrix, to create a bigram LR table. Figure 4 and Figure 5 give a sample CFG G_1 and a probabilistic connection matrix M_1, respectively.

Note that grammar G_1 in Figure 4 does not explicitly express local connection constraints between terminal symbols. Such local connection constraints are easily expressed by a matrix M_1 as shown in Figure 5.

From the CFG given in Figure 4, we can generate an LR table, Table 1, in Step 1 using the conventional LR table generation algorithm.

Table 2 is the resultant LR table at the completion of Step 2 and Step 3, produced based on Table 1. Actions numbered *(2)* and *(3)* in Table 2 are those which are removed by Step 2 and Step 3, respectively.

In state 1 with a lookahead symbol *b1*, *re6* is carried out after executing action *sh1* in state 0, pushing *a1* onto the stack. Note that *a1* and *b1* are now consecutive, in this order. However, the probabilistic connection matrix (see Figure 5) does not allow such a sequence of terminal symbols, since $PConnect(a1, b1) = 0$. Therefore, the action *re6* in state 1 with lookahead *b1* is removed from Table 1 in Step 2, and thus marked as *(2)* in Table 2. For this same reason, the other *re6*s in state 1 with lookahead symbols *a1* and *a2* are also removed from Table 1.

On the other hand, in the case of *re6* in state 1 with lookahead symbol *b2*, as *a1* can be followed by *b2* ($PConnect(a1, b2) \neq 0$), action *re6* cannot be removed. The remaining actions marked as *(2)* in Table 2 should be self-evident to readers.

Next, we would like to consider the reason why action *sh9* in state 4 with lookahead *a1* is removed from Table 1. In state 9, *re6* with lookahead symbol $ has already been removed in Step 2, and there is no succeeding action for *sh9*. Therefore, action *sh9* in state 3 is removed in Step 3, and hence marked as *(3)*.

Let us consider action *re3* in state 8 with lookahead *a1*. After this action is carried out, the GLR parser goes to state 4 after pushing X onto the stack. However, *sh9* in state 4 with lookahead *a1* has already been removed, and there is no succeeding action for *re3*. As a result, *re3* in state 8 with lookahead symbol *a1* is removed in Step 3. Similarly, *re9* in state 7 with lookahead symbol *a1* is also removed in Step 3. In this way, the removal of actions propagates to other removals. This chain of removals is called Constraint Propagation, and occurs in Step 3. Actions removed in Step 3 are marked as *(3)* in Table 2.

Careful readers will notice that there is now no action in state 9 and that it is possible to delete this state in Step 4. Table 3 shows the LR table after Step 4.

As a final step, we would like to assign bigram constraints to each action in Table 3. Let us consider the two *re8*s in state 6, reached after executing *sh6* in state 4 by pushing a lookahead of *b1* onto the stack. In state 6, P is calculated at Step 5 as shown below:

$$\begin{aligned} P &= PConnect(b1, a2) + PConnect(b1, b1) \\ &= 0.1 + 0.9 \\ &= 1 \end{aligned}$$

We can assign the following probabilities p to each *re8* in state 6 by way of Step 5:

$$p = \begin{cases} \frac{PConnect(b1,a2)}{P \times n} = \frac{0.1}{1 \times 1} = 0.1 & \text{for } re8 \text{ with lookahead } a2 \\ \frac{PConnect(b1,b1)}{P \times n} = \frac{0.9}{1 \times 1} = 0.9 & \text{for } re8 \text{ with lookahead } b1 \end{cases}$$

After assigning a probability to each action in the LR table at Step 5, there remain actions without probabilities. For example, the two conflict actions (*re2*/*sh6*) in state 3 with lookahead *b1* are not assigned a probability. Therefore, each of these actions is assigned the same probability, 0.5, in Step 7. A probability of 1 is assigned to remaining actions, since there is no conflict among them.

Table 4 shows the final results of applying Algorithm 1 to G_1 and M_1.

state	**action**					**goto**				
	a1	*a2*	*b1*	*b2*	*$*	*A*	*B*	*X*	*Y*	*S*
0	*sh1*	*sh2*				*3*		*4*		*5*
1	*re6*	*re6*	*re6*	*re6*						
2	*re7*	*re7*	*re7*	*re7*						
3	*re2*	*re2*	*re2/sh6*	*sh7*			*8*			
4	*sh9*	*sh10*	*sh11*			*12*			*13*	
5					*acc*					
6	*re8*	*re8*	*re8*							
7	*re9*	*re9*	*re9*							
8	*re3*	*re3*	*re3*							
9					*re6*					
10					*re7*					
11	*sh9*	*sh10*				*14*				
12					*re4*					
13					*re1*					
14					*re5*					

Table 1: Initial LR table for G_1

state	**action**					**goto**				
	a1	*a2*	*b1*	*b2*	*$*	*A*	*B*	*X*	*Y*	*S*
0	*sh1*	*sh2*				*3*		*4*		*5*
1	*re6(2)*	*re6(2)*	*re6(2)*	*re6*						
2	*re7(2)*	*re7(2)*	*re7*	*re7(2)*						
3	*re2(3)*	*re2*	*re2/sh6*	*sh7*			*8*			
4	*sh9(3)*	*sh10*	*sh11*			*12*			*13*	
5					*acc*					
6	*re8(2)*	*re8*	*re8*							
7	*re9(3)*	*re9(2)*	*re9*							
8	*re3(3)*	*re3*	*re3*							
9					*re6(2)*					
10					*re7*					
11	*sh9(3)*	*sh10*				*14*				
12					*re4*					
13					*re1*					
14					*re5*					

Table 2: LR table after Steps 2 and 3

state	**action**					**goto**				
	a1	*a2*	*b1*	*b2*	*$*	*A*	*B*	*X*	*Y*	*S*
0	*sh1*	*sh2*				*3*		*4*		*5*
1				*re6*						
2			*re7*							
3		*re2*	*re2/sh6*	*sh7*			*8*			
4		*sh10*	*sh11*			*12*			*13*	
5					*acc*					
6		*re8*	*re8*							
7			*re9*							
8		*re3*	*re3*							
10					*re7*					
11		*sh10*				*14*				
12					*re4*					
13					*re1*					
14					*re5*					

Table 3: LR table after Step 4

state	**action**					**goto**				
	a1	*a2*	*b1*	*b2*	*$*	*A*	*B*	*X*	*Y*	*S*
0	*sh1* *0.6*	*sh2* *0.4*				*3*		*4*		*5*
1				*re6* *1.0*						
2			*re7* *1.0*							
3		*re2* *1.0*	*re2/sh6* *0.5/0.5*	*sh7* *1.0*			*8*			
4		*sh10* *1.0*	*sh11* *1.0*			*12*			*13*	
5					*acc* *1.0*					
6		*re8* *0.1*	*re8* *0.9*							
7			*re9* *1.0*							
8		*re3* *1.0*	*re3* *1.0*							
10					*re7* *1.0*					
11		*sh10* *1.0*				*14*				
12					*re4* *1.0*					
13					*re1* *1.0*					
14					*re5* *1.0*					

Table 4: The Bigram LR table constructed by Algorithm 1

4.2 Comparison of Language Models

Using the bigram LR table as shown in Table 4, the probability $P1$ of the string "*a2 b1 a2*" is calculated as:

$$\begin{aligned}
P1 &= P(a2\ b1\ a2) \\
&= \sum_{i=1}^{T} P(\mathrm{Tree}_i) \\
&= P(0, a2, sh2) \times P(2, b1, re7) \\
&\quad \times P(3, b1, \underline{re2}) \times P(4, b1, sh11) \\
&\quad \times P(11, a2, sh10) \times P(10, \$, re7) \\
&\quad \times P(14, \$, re5) \times P(13, \$, re1) \\
&\quad \times P(5, \$, acc) \\
&\quad + P(0, a2, sh2) \times P(2, b1, re7) \\
&\quad \times P(3, b1, \underline{sh6}) \times P(6, a2, re8) \\
&\quad \times P(8, a2, re3) \times P(4, a2, sh10) \\
&\quad \times P(10, \$, re7) \times P(12, \$, re4) \\
&\quad \times P(13, \$, re1) \times P(5, \$, acc) \\
&= 0.4 \times 1.0 \times 0.5 \times 1.0 \times 1.0 \\
&\quad \times 1.0 \times 1.0 \times 1.0 \times 1.0 \\
&\quad + 0.4 \times 1.0 \times 0.5 \times 0.1 \times 1.0 \\
&\quad \times 1.0 \times 1.0 \times 1.0 \times 1.0 \times 1.0 \\
&= 0.2 + 0.02 \\
&= 0.22
\end{aligned}$$

where $P(\mathrm{Tree}_i)$ means the probability of the i-th parsing tree generated by the GLR parser and $P(S, L, A)$ means the probability of an action A in state S with lookahead L.

On the other hand, using only bigram constraints, the probability $P2$ of the string "*a2 b1 a2*" is calculated as:

$$\begin{aligned}
P2 &= P(a2\ b1\ a2) \\
&= P(a2|\#) \times P(b1|a2) \times P(a2|b1) \times P(\$|a2) \\
&= \times 0.3 \times 0.1 \times 0.7 \\
&= 0.0084
\end{aligned}$$

The reason why $P1 > P2$ can be explained as follows. Consider the beginning symbol *a2* of a sentence. In the case of the bigram model, *a2* can only be followed by either of the two symbols *b1* and \$ (see Figure 5). However, consulting the bigram LR table reveals that in state 0 with lookahead *a2*, *sh2* is carried out, entering state 2. State 2 has only one action *re7* with lookahead symbol *b1*. In other words, in state 2, \$ is not predicted as a succeeding symbol of *a1*. The exclusion of an ungrammatical prediction in \$ makes $P1$ larger than $P2$.

Perplexity is a measure of the complexity of a language model. The larger the probability of the language model, the smaller the perplexity of the language model. The above result ($P1 > P2$) indicates that the bigram LR table model gives smaller perplexity than the bigram model. In the next section, we will demonstrate this fact.

Language model	Perplexity
Bigram	6.50
Bigram LR table	5.99
Trigram	4.92

Table 5: Perplexity of language models

5 Evaluation of Perplexity

Perplexity is a measure of the constraint imposed by the language model. *Test-set perplexity* (Jelinek, 1990) is commonly used to measure the perplexity of a language model from a test-set. *Test-set perplexity* for a language model L is simply the geometric mean of probabilities defined by:

$$Q(L) = 2^{H(L)}$$

where

$$H(L) = \frac{1}{N} \sum_{i=1}^{M} \log P(S_i)$$

Here N is the number of terminal symbols in the test set, M is the number of test sentences and $P(S_i)$ is the probability of generating i-th test sentence S_i.

In the case of the bigram model, $P_{bi}(S_i)$ is:

$$\begin{aligned}
P_{bi}(S_i) &= P(x_1, x_2, \cdots, x_n) \\
&= P(x_1|\#)P(x_2|x_1) \cdots P(x_n|x_{n-1})P(\$|x_n)
\end{aligned}$$

And in the case of the trigram model, $P_{tri}(S_i)$ is:

$$\begin{aligned}
P_{tri}(S_i) &= P(x_1, x_2, \cdots, x_n) \\
&= P(x_1|\#)P(x_2|\#, x_1) \cdots \\
&\quad P(x_n|x_{n-2}, x_{n-1})P(\$|x_{n-1}, x_n)
\end{aligned}$$

Table 5 shows the test-set perplexity of preterminals for each language model. Here the preterminal bigram models were trained on a corpus with 20663 sentences, containing 230927 preterminals. The test-set consists of 1320 sentences, which contain 13311 preterminals. The CFG used is a phrase context-free grammar used in speech recognition tasks, and the number of rules and preterminals is 777 and 407, respectively.

As is evident from Table 5, the use of a bigram LR table decreases the test-set perplexity from 6.50 to 5.99. Note that in this experiment, we used the LALR table generation algorithm[2] to construct the bigram LR table. Despite the disadvantages of

[2]In the case of LALR tables, the sum of the probabilities of all the possible parsing trees generated by a given CFG may be less than 1 (Inui et al., 1997).

LALR tables, the bigram LR table has better performance than the simple bigram language model, showing the effectiveness of a bigram LR table.

On the other hand, the perplexity of the trigram language model is smaller than that of the bigram LR table. However, with regard to data sparseness, the bigram LR table is better than the trigram language model because bigram constraints are more easily acquired from a given corpus than trigram constraints.

Although the experiment described above is concerned with natural language processing, our method is also applicable to speech recognition.

6 Conclusions

In this paper, we described a method to construct a bigram LR table, and then discussed the advantage of our method, comparing our method to the bigram and trigram language models. The principle advantage over the bigram language model is that, in using a bigram LR table, we can combine both local probabilistic connection constraints (bigram constraints) and global constraints (CFG).

Our method is applicable not only to natural language processing but also speech recognition. We are currently testing our method using a large-sized grammar containing dictionary rules for speech recognition.

Su et al. (Su et al., 1991) and Chiang et al. (Chiang et al., 1995) have proposed a very interesting corpus-based natural language processing method that takes account not only of lexical, syntactic, and semantic scores concurrently, but also context-sensitivity in the language model. However, their method seems to suffer from difficulty in acquiring probabilities from a given corpus.

Wright (Wright, 1990) developed a method of distributing the probability of each PCFG rule to each action in an LR table. However, this method only calculates syntactic scores of parsing trees based on a context-free framework.

Briscoe and Carroll (Briscoe and Carroll., 1993) attempt to incorporate probabilities into an LR table. They insist that the resultant probabilistic LR table can include probabilities with context-sensitivity. Inui et. al. (Inui et al., 1997) reported that the resultant probabilistic LR table has a defect in terms of the process used to normalize probabilities associated with each action in the LR table.

Finally, we would like to mention that Klavans and Resnik (Klavans and Resnik, 1996) have advocated a similar approach to ours which combines symbolic and statistical constraints, CFG and bigram constraints.

Acknowledgements

We would like to thank Mr. Toshiyuki Takezawa and Mr. Junji Etoh for providing us the dialog corpus and the grammar for our experiments. We would also like to thank Mr. Timothy Baldwin for his help in writing this paper.

References

A.V. Aho, S. Ravi, and J.D. Ullman. 1986. *Compilers: Principle, Techniques, and Tools.* Addison Wesley.

T. Briscoe and J. Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

T.H. Chiang, Y.C. Lin, and K.Y. Su. 1995. Robust learning, smoothing, and parameter tying on syntactic ambiguity resolution. *Computational Linguistics*, 21(3):321–349.

A. Franz. 1996. *Automatic Ambiguity Resolution in Natural Language Processing.* Springer.

T. Fujisaki, F. Jelinek, J. Cocke, E. Black, and T. Nishino. 1991. A probabilistic parsing method for sentence disambiguation. In M. Tomita, editor, *Current Issues in Parsing Technologies*, pages 139–152. Kluwer Academic Publishers.

K. Inui, V. Sornlertlamvanich, H. Tanaka, and T. Tokunaga. 1997. A new formalization of probabilistic GLR parsing. In *International Workshop on Parsing Technologies.*

F. Jelinek. 1990. Self-organized language modeling for speech recognition. In A. Waibel and K.F. Lee, editors, *Readings in Speech Recognition*, pages 450–506. Morgan Kaufmann.

J.L. Klavans and P. Resnik. 1996. *The Balancing Act: Combining Symbolic and Statistical Approaches to Language.* The MIT Press.

D.E. Knuth. 1965. On the translation of languages left to right. *Information and Control*, 8(6):607–639.

K.F. Lee. 1989. *Automatic Speech Recognition: The Development of the SPHINX System.* Kluwer Academic Publishers.

K.Y. Su, J.N. Wang, M.H. Su, and J.S. Chang. 1991. GLR parsing with scoring. In M. Tomita, editor, *Generalized LR Parsing.* Kluwer Academic Publishers.

H. Tanaka, H. Li, and T. Tokunaga. 1994. Incorporation of phoneme-context-dependence into LR table through constraints propagation method. In *Workshop on Integration of Natural Language and Speech Processing*, pages 15–22.

H. Tanaka, T. Takezawa, and J. Etoh. 1997. Japanese grammar for speech recognition considering the MSLR method. In *Information Processing Society of Japan, SIG-SLP-15*, pages 145–150. (in Japanese).

M. Tomita. 1986. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers.

J.H. Wright. 1990. LR parsing of probabilistic grammars with input uncertainty for speech recognition. *Computer Speech and Language*, 4(4):297–323.