Proc. of SNLP'95, Thailand, 1995 1~18頁

A New Technology of GLR Parsing and its Applications to Speech and Natural Language Processing

Hozumi Tanaka, Hui Li and Takenobu Tokunaga

Department of Computer Science
Tokyo Institute of Technology
2-12-1 Ôokayama Meguro Tokyo 152 Japan

Abstract

The processing of language and speech requires a lot of information which can be divided into two type of constraints, global and local. For example, the context-free grammar(CFG) that is commonly used for syntactic analysis can be seen a global constraint and the connectability between two adjacent morphonemes can be seen a local constraint. These kind of connection constraints are often expressed by a connection matrix. While these can be expressed by context-free grammar(CFG), it makes the grammar very complex.

Generalized LR(GLR) parsing techniques have been widely used for parsing sentences. The GLR parser is guided by an LR parsing table generated from the CFG. In this paper we propose a method called CPM(Constraints Propagation Method) that generates LR tables by using local and global constraints. The former are encoded in a connection matrix while the latter are containted in a CFG. We will also discuss the relative advantages and disadvantages of our CPM.

This paper is divided into two parts. Section I discusses a method how CPM is used for allophone-based speech processing. The connection constraint in this case is expressed by an allophone connection matrix. Although we illustrate our method only in Japanese, the method is general enough to be applied to any other spoken language. Section II discusses how CPM is used in order to integrate morphological and syntactic constraints into an LR table. The method can be used for languages such as Japanese, Chinese, Korea and Thai where segmentation is a major problem as adjacent words are not necessarily separated by a space.

Part I: Incorporation of Phoneme-Context-Dependence into LR Tables through Constraint Propagation

1 Introduction

It is obvious that successful speech recognition requires the use of linguistic information. To this end, a generalized LR (GLR) parser provides an exceptionally good tool, as it combines in a very flexible way linguistic and phonological information.

One problem with continuous speech recognition of real world texts containing large vocabulary is the reduction of the search space. GLR parsers can meet this requirement[4] by using linguistic constraints. In the phone-based speech recognition system, the GLR parser has been used as a phoneme predictor. The GLR parser[12] is guided by an LR table, automatically generated from context-free grammar (CFG) rules. It proceeds left-to-right without backtracking. In order to make phone predictions, the lookahead symbols in LR table are phones rahter than the usual grammatical categories. Lexical rules are thus expressed as follows:

$$<\textit{grammatical category}> \rightarrow < \textit{sequence of phones}>.$$

Several experiments[2][5][7][9] have shown that the performance of speech recognition systems could be improved by using allophones rather than phones as recognition units. Allophone models (such as triphone models) are context-dependent phone models that take into consideration both the left and right neighboring phones to model the major coarticulatory effects in continuous speech.

The combination of allophone models and a GLR parser[2][7] is desirable in order to achieve better performance in continuous speech recognition. The main problem of integrating GLR parsing into an allophone-based recognition system is how to solve the word juncture problem, that is, how to express the phones at a word boundary with allophone models.

In this part, we propose a new method to generate an allophone-based LR table that can solve the word juncture problem., By combining allophone rules with syntactic and lexical rules, this method generates an LR table, which it modifies then on the basis of an allophone connection matrix by using constraint propagation methods(CPM).

The organization of this part is as follows. Section 2 provides an overview of the past allophone-based GLR parsing methods and points out problems in those methods. After discussing the advantages of using the canonical LR table for speech recognition, section 3 describes our method to generate an allophone-based LR table by applying CPM. Section 4 compares LR tables before and after having applied CPMs; and Section 5 points towards future developments.

In the following sections, we will use several examples from Japanese . It should be noted, however, that the method proposed is not language specific, in principal, it can be applied to any spoken language.

2 Overview of Allophone-Based GLR Parsing Method

The main problem of integrating GLR parsing into an allophone-based recognition system is solving the word juncture problem. Consider a Japanese word "a k i(autumn)", which is a sequence of three phones in the lexical rule that follows:

$$adj \rightarrow a\ k\ i\ .$$

The allophone corresponding to the phone "k", can be determined by the left and right context which are known in advance, namely "a" and "i" respectively. On the other hand, the phones "a" and "i" are located at the word boundary. For the beginning phone "a" there is no left context, and for the final phone "i", there is no right context. Hence for the phones located at the word boundaries, it is difficult to anticipate their respective left or right contexts. we called this the word juncture problem.

Several allophone-based GLR parsing methods have been proposed[2][7] in order to tackle this problem.

Itou et al.[2] have used the lexical rules as the following:

$$adj \rightarrow a(*,k)\ k2\ i(a,*)$$

where "k2" is an allophone of "k", which has the left and right context "a" and "i"; "a(*,k)" is a special phone of "a" whose right context is known, and "i(k,*)" is a special phone of "i" whose left context is known. The LR table is built from a set of syntactic and lexical rules. The allophones at word boundaries are determined dynamically during the recognition process, namely when the preceding or succeeding words are obtained.

In this method, some changes in a GLR parsing algorithm are required to take into account the phoneme-context-dependence at word boundaries. Furthermore, for the final phones "i(k,*)" in the example given above, the system makes unnecessary allophone predictions because of the absence of context on the right hand side.

Nagai et al.[7] have proposed the following three approaches :

(1)Grammar level realization

Like in Itou's approach[2], word intrinsic phones are changed into allophones. Phones at a word boundary are changed into possible allophones, which generate new grammatical categories. For instance, if "a(*,k)" and "i(k,*)" have two allophones, "a1" and "a2", "i1" and "i2" respectively, then the four lexical rules are created from a word such as "a k i":

$$a1\_noun\_i1 \rightarrow a1\ k2\ i1, \quad a2\_noun\_i1 \rightarrow a2\ k2\ i1$$
$$a1\_noun\_i2 \rightarrow a1\ k2\ i2, \quad a2\_noun\_i2 \rightarrow a2\ k2\ i2$$

As one can see, too many lexical rules are created from each word. The creation of new grammatical categories will produce many new syntactic rules. Furthermore, nonterminal symbols of RHS (right hand side) in the syntactic rule must take into account the word juncture problem, considering newly created nonterminal symbols. For example, the nonterminal symbols, "i_cat_j" and "j_cat'_k" can be adjacent in this order, and so on. Thus, phoneme-context-dependence is expressed by a large number of lexical and grammar rules. For instance, in the case of 1353, phoneme-context-independent CFG rules and 300 allophone models, the number of CFG rules amounts to 61507[7].[1] Although this method requires no change of a GLR parsing algorithm, the combinatorial explosion may occur.

(2)Table level realization

[1]According to our method explained later, the number of CFG rules is only 1353 + 300(=1653).

From a set of syntactic and phoneme-context-independent lexical rules, this method generates an LR table, it introduces then stepwise a set of allophones by adding incrementally new states and new nonterminal symbols in the LR table in order to incorporate phoneme-context-dependence. The table modification process is complex and requires changes to the parsing algorithm in order to maintain the left and right context of allophones. Thus, phoneme-context-dependence must be dynamically recognized in the parsing process. This method increases unnecessarily the number of states in the LR table yielding inefficiency of the parsing process.

(3) Parser level realization

In this method the phoneme context dependence is not incorporated in an LR table. The determination of allophones is completely handled by a GLR parser, which has to include a method of the phoneme-context-dependence in a procedural way. This makes the original GLR parsing algorithm more complex and inefficient.

The CPM proposed in this paper, compiles right from the beginning the phoneme-context-dependence into an LR table. It does so by introducing a set of allophone rules and by using methods of constraint propagation. In the CPM, the dynamic processing during the parsing, like that in the table and the parser level realizations[7], is not required, therefore, no change in the GLR parsing algorithm is required. Unlike grammar level realization[7], the CFG rules used in our method is equal to the number of phoneme-context-independent CFG rules plus the number of the allophone models. Hence there is no explosion of CFG rules.

3 Algorithm for Generating Allophone-based LR Table through Constraint Propagation

In this section we propose a new method called CPM (Constraint Propagation Method) in order to generate a phoneme-context-dependent(allophone-based) LR table. This method allows to solve the word juncture problem and to predict allophones.

The outline of our method is shown in Fig. 1.

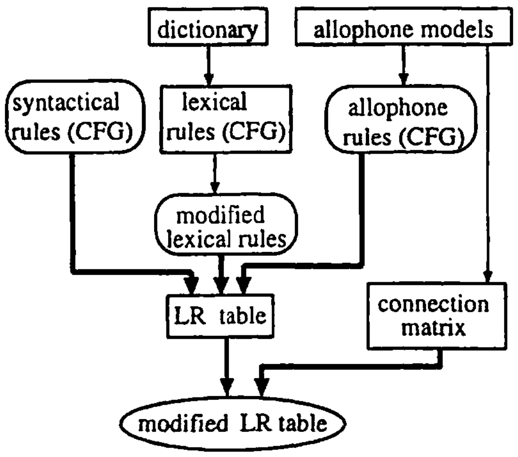


Fig. 1 Outline of CPM

In this method, an allophone connection matrix and a set of allophone rules are constructed on the basis of an allophone model. Lexical rules determine how word intrinsic phones are changed into allophones. From a set of syntactic, lexical and allophone rules(CFG), an initial allophone-based LR table is generated, which is then modified by using the allophone connection matrix and constraint propagation methods.

Before explaining the details of each step, we would like to discuss the types of LR tables. All methods mentioned in Section 2 have used the SLR or LALR table. Compared with the canonical LR table, the SLR and LALR table can not provide the precise phoneme predictions because the SLR and LALR table have fewer states due to merging of several states in an LR table[1]: yet merging several states transforms many actions into a state in which they produce many predictions. This is why our method uses the canonical LR table. Generally, the canonical LR table has more states than the SLR or LALR table, but by applying CPM, we can achieve substantial reduction of the table's size.

In this section, we will illustrate our method by using an example of simple syntactic and lexical rules(CFG) of Japanese(see Fig. 2).

```
(1)  S → N BE              (3)  N → ch i ch i (father)
(2)  N → h a h a (mother)  (4)  BE → d a (be)
```

Fig. 2 An example of the CFG rules and lexical rules

3.1 Allophone connection matrix

The allophone context of an allophone "x" is defined as:

<*left context*> *x* <*right context*>

where <*left context*> and <*right context*> are a set of phones. We assume that there is only one allophone context per allophone. An allophone connection matrix is created from a set of allophone contexts.

Let us consider the allophones "i2" and "d1" for the phones "i" and"d", and the allophone contexts shown here below.

$$\left\{\begin{array}{c}\vdots\\ ch\\ \vdots\end{array}\right\}\mathrm{i2}\left\{\begin{array}{c}\vdots\\ d\\ \vdots\end{array}\right\},\quad \left\{\begin{array}{c}\vdots\\ i\\ \vdots\end{array}\right\}\mathrm{d1}\left\{\begin{array}{c}a\\ \vdots\\ \vdots\end{array}\right\}$$

According to the allophone contexts shown here above in the form of triphones, "d1" can follow "i2" because its right and left hand side contain respectively the phones "d" and "i".

If a connection matrix is expressed as an array of *Connect*[left_allophone, right_allophone], we can fill *Connect*[i2,d1] with symbol "1" to signal that "i2" and "d1" can follow each other in this specific order. Therefore, we can construct an allophone connection matrix from a set of allophone contexts. Note: an entry in the matrix, which can not be filled with "1", is marked by "0". This is to indicate the fact that the counterparts of two adjacent allophones cannot be connected.

Fig. 3 is an example of the allophone connection matrix partially filled. We will use this connection matrix to incorporate allophone connection constraints into the LR table at Section 3.4 by applying CPM.

		RIGHT											
		h1	h2	a1	a2	ch1	ch2	i1	i2	d1	d2	d3	$
LEFT	h1			1	0								
	h2			0	0								
	a1	1	1							1	0	1	0
	a2	0	0							0	0	0	1
	ch1							0	0				
	ch2							0	1				
	i1					0	0			0	0	0	
	i2					1	1			0	1	1	
	d1			0	1								
	d2			0	1								
	d3			0	0								

Fig. 3 An example of the allophone connection matrix partially filled

3.2 Modification of the lexical rules

The phones within a word are automatically converted into the allophones. This is done by using allophone contexts. Please note that we can not change the phones at the word boundaries because of the word juncture problem. Therefore, the lexical rules(rules 2 to 4 in the figure 2 are changed into those shown in the figure here below:

```
(2)'  N → h a1 h1 a    (4)'  BE → d a
(3)'  N → ch i2 ch2 i
```

Fig. 4 Conversion of the lexical rules

3.3 Allophone-based LR table

The allophone rules are derived by associating a phone with the corresponding allophones:

$$<phone> \rightarrow <allophone1> \mid < allophone2 > \mid \cdots$$

Assume the following couples: {h1, h2} for "h", {a1, a2} for "a", {ch1, ch2} for "ch", {i1, i2} for "i", {d1, d2} for "d", on the basis of these data set of allophone rules (see (5) to (14) in Fig. 5) can be produced.

(5) h → h1	(9) ch → ch1	(13) d → d1
(6) h → h2	(10) ch → ch2	(14) d → d2
(7) a → a1	(11) i → i1	(15) d → d3
(8) a → a2	(12) i → i2	

Fig. 5 A set of the allophone rules

Now we have a set of syntactic, lexical and allophonic rules as shown in Fig. 6

(1) S → N BE	(6) h → h2	(11) i → i1
(2)' N → h a1 h1 a	(7) a → a1	(12) i → i2
(3)' N → ch i2 ch2 i	(8) a → a2	(13) d → d1
(4)' BE → d a	(9) ch → ch1	(14) d → d2
(5) h → h1	(10) ch → ch2	(15) d → d3

Fig. 6 A set of the CFG, lexical and allophone rules

From these extended CFG rules we can generate a canonical LR table (see Fig. 7). Note that all the lookahead symbols of this table are allophones.

	ACTION												GOTO							
state	a1	a2	ch1	ch2	d1	d2	d3	h1	h2	i1	i2	$	BE	N	S	a	ch	d	h	i
0			s4(c)	s5				s1	s2(c)					7	8		6		3	
1	r5																			
2	r6(a)				(c)															
3	s9																			
4											r9(a)									
5											r10									
6											s10									
7					s11	s12	s13(c)						15					14		
8												acc								
9								s16												
10				s17																
11	r13(a)	r13																		
12	r14(a)	r14																		
13	r15(a)	r15(a)						(b)												
14	s18(e)	s19														20				
15												r1								
16	s21	s22(b)														23				
17										s24(b)	s25									26
18												r7								
19												r8(c)								
20												r4								
21					r7	r7(a)	r7(d)													
22					r8(a)	r8(a)	r8(a)	(d)												
23					r2	r2(e)	r2(d)													
24					r11(a)	r11(a)	r11(a)													
25			(e)		r12(a)	r12	r12(d)													
26					r3(e)	r3	r3(d)													

Fig. 7 Canonical LR(CLR) table generated from rules in Fig. 6

In the extended CFG rules in Fig. 6, the information about connectability represented in Fig. 3 is not included for the phones at the word boundaries.

3.4 Modifications of LR Table

In order to incorporate allophone connection constraints into the LR table, we combine our CPM with the method developped by Tanaka et al.[11]. In doing so we modify the original canonical LR table. Initial constraints are imposed on the LR table by using allophone connection matrix. These constraints propagate then throughout the LR table producing a modified allophone-based LR table.

3.4.1 Connection check

At first, the constraints on connectability are introduced into the LR table by deleting illegal actions, that is, actions that violate the connection constraints represented in the connection matrix.

(a). Allophone rules for deletable reduce action

At this step we check every reduce action with an allophone rule by using the connection matrix in Fig. 3. In a similar way as Tanaka et al.[11], the illegal reduce actions with allophone rules, which violate connection constraints, are marked "deletable".

Fig. 8 shows this procedure.

```
for each reduce action R with an allophone rule
 in each entry of LR table {
     if (Connect[x, y] = 0) {
        mark R "deletable";
     }
}
where
 x: the RHS of the allophone rule used by R.
 y: the lookahead symbol of R.
 Connect: the allophone connection matrix.
```

Fig. 8 Check the reduce actions with allophone rules

Consider, for example, re6 with the lookahead symbol "a1" at state 2 in Fig. 7. The allophone connection matrix indicates that the connection between "h2" (RHS of rule 6) and "a1" is not possible in this order(*Connect*[h2,a1] = 0), hence the reduce action("re6") is illegal. This being so it gets marked as "deletable".

In Fig. 7, all the deletable reduce actions found by this step are marked (a).

(b). Deletable shift action whose predecessors are shift actions

Let us consider the left hand side of the last phone of a word. If the left hand side phone is an allophone, we can easily check the connectability between the left allophone and allophones belonging to the last phone. In this case, after shifting the left allophone, we have to shift all the end allophones allowed by allophone conection matrix. Consecutive shift actions will occur.

Consider a Japanese word "ch i2 ch2 i", and assume that there are two possible allophones "i1" and "i2" for the end phone "i". The left allophone of the end phone "i" is "ch2". After shifting "ch2" by sh17 in state 10, we can shift "i1" and "i2" at state 17 in Fig. 7. However, *Connect*[ch2,i1]=0 means that shifting "i1" is not allowed, and sh24 in state 17 with lookahead symbol "i1" is marked as"deletable". On the contrary, sh25 with lookahead symbol "i2" in state 17 is not "deletable", because "ch2" and "i2" is connectable(*Connect*[ch2,i2]=1). Fig. 9 shows this procedure.

```
for each shift action S in each entry of LR table {
  if (the action prior to S is a shift action) {
    if (Connect[x, y] = 0) {
      mark S "deletable";
    }
  }
}
where
 x: the lookahead symbol of the shift action prior to S.
 y: the lookahead symbol of S.
```

Fig. 9 Check the shift actions

In Fig. 7, all the deletable shift actions found by this step are marked (b).

3.4.2 Constraints Propagation

Secondly, the above constraints propagate throughout the LR table by deleting all the other illegal actions.

(c). Deletable shift action that lead to an empty state

An empty state is defined as a state whose actions are all marked "deletable". For an empty state, the shift actions that lead to this empty state should be marked "deletable". Meanwhile, for a state, if all of its preceding actions are marked "deletable", all the actions in this state should also be marked as "deletable".

For example, in Fig. 7, sh2 in state 0 with the lookahead symbol "h2" should be marked "deletable", because state 2 is an empty state(the unique action re6 in state 2 has been marked "deletable" at step(a)).

In Fig. 7, all the deletable actions found by this step are marked (c).

(d). Deletable reduce action whose successors are all "deletable" actions

After a reduce action has been carried out, the parser will move on to the next state according to the goto graph, and the following actions with the same lookahead symbol in the next state will be carried out next. If all the following actions of the next states have been marked as"deletable", the reduce action *R* should also be marked as "deletable".

This procedure is summarized in Fig. 10.

```
for each reduce action R in each entry of LR table {
  if (every action that R will lead to has
    already been marked "deletable") {
        mark R "deletable";
  }
}
```

Fig. 10 Deletable reduce actions whose succeeding actions have been all marked "deletable"

Fig. 11 illustrates a part of the goto graph reconstructed from the LR table in Fig. 7.

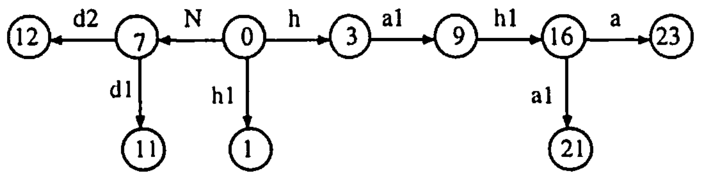


Fig. 11 A part of goto graph

Consider re7 in state 21 with a lookahead symbol "d3" in Fig. 7. According to the above goto graph, the parser will move on to state 23 after re7, applying rule 7 (a $\rightarrow$ al) in state 21(keeping the lookahead symbol "d3" during the reduce action). Thus, in state 23, the next reduce action becomes re2 (N $\rightarrow$ h al hl a) with the same lookahead symbol "d3". Therefore, re7(lookahead symbol "d3") in state 21 will lead to re2(lookahead symbol "d3") in state 23.

After re2, the parser will move to state 7, since from state 23 we can traverse the graph in a reverse order such as "a", "hl", "al", "h". In doing so we will finally reache state 0 from where we move to state 7 by shifting N. In state 7, the action with the same lookahead symbol "d3" is sh13, so re2(lookahead symbol "d3") in state 23 will lead to sh13 in state 7.

However, sh13 has already been marked "deletable" (see step (c)) in state 7. Since, re2(lookahead symbol "d3") in state 23 is marked as"deletable", re7(lookahead symbol "d3") in state 21 should be marked as"deletable "too.

In Fig. 7, all the deletable reduce actions found by this step are marked (d).

(e). Deletable action whose predecessors are all "deletable" actions

If all the actions that will lead to an action called *Act* have been marked "deletable", *Act* should be marked "deletable" too. Fig. 12 shows this procedure.

```
for each action A in each entry of LR table {
  if (all the preceding actions of A have been
      marked "deletable") {
        mark A "deletable";
  }
}
```

Fig. 12 Remove the actions whose preceding actions have been all marked "deletable"

Consider the reduce action re3 with the lookahead symbol "dl" in state 26 is reached from rell(in state 24) and rel2(in state 25) with lookahead symbol "dl". These two reduce actions(rell and rel2) have been marked "deletable" by step(a), so the reduce action re3(lookahead symbol "dl") should also be marked "deletable".

This procedure of propagating constraints (step(c)-(e)) should be repeated until no more "deletable" actions are found. The LR table is then compressed by deleting all the "deletable" actions and all the empty states in order to reduce the size of the table.

Fig. 13 summariez our algorithm.

```
CPM()
{
  /* connection check */
    step (a) and (b);
  /* constraints propagation */
  while(1) {
    step (c) to (e);
    if(no new "deletable" action comes out)
      break;
  }
  compact LR table;
}
```

Fig. 13 A top level procedure of CPM

The above procedure enables us to introduce the phoneme-context-dependence into the phones at word boundaries and solve the word juncture problem.

Fig. 14 is the modified allophone-based canonical LR(MCLR) table from Fig. 7 after applying CPM.

	ACTION								GOTO							
state	a1	a2	ch2	d1	d2	h1	i2	$	BE	N	S	a	ch	d	h	i
0			s5			s1				7	8		6		3	
1	r5															
3	s9															
5							r10									
6							s10									
7				s11	s12				15					14		
8								acc								
9						s16										
10			s17													
11		r13														
12		r14														
14		s19										20				
15								r1								
16	s21											23				
17							s25									26
18								r7								
20								r4								
21				r7												
23				r2												
25					r12											
26					r3											

Fig. 14 The modified canonical LR(MCLR) table after CPM

4 The Effects of CPM

Compared with the table of Fig. 7, the lookahead symbols in Fig. 14 include only "a1", "a2", "ch2", "d1", "d2", "h1" and "i2". The allophones at word boundaries are restricted by CPM through deleting the illegal actions. This very simple example suggests how the word juncture problem can be solved by applying CPM.

For a task with 64 syntactic rules and 120 lexical rules, the canonical LR (CLR) table is compared in terms of the number of states and actions before and after constraint propagation.

The syntactic rules, lexical rules and allophone models we used are the same as in [2], which have already been used in a speech recognition system.

Table 1: Comparison of table size before and after CPM

allophones	table	state	shift	reduce	goto
128	CLR	1722	2260	9254	421
	MCLR	1016	873	1015	421
256	CLR	2959	4760	37831	421
	MCLR	1061	930	1072	421
512	CLR	5627	10211	161127	421
	MCLR	1096	971	1135	421
1024	CLR	11157	21057	701370	421
	MCLR	1139	1003	1212	421

Table 1 shows the table's size before and after CPM for 128, 256, 512 and 1024 allophone models. After applying CPM, in the case of 128 allophone models, the number of states, shift actions, and reduce actions decreases to 61.4%, 38.6%, and 11% of the original canonical LR table, and in the case of 1024 allophone models, the number of states, shift actions, and reduce actions decreases to 11%, 4.9%, and 0.2% of the original canonical LR table. The greater the number of allophones are, the greater the benefits in terms the table size shrinking.

Reduction of the reduce and shift actions implies that the allophone predictions in speech recognition become more accurate.

5 Discussions and Conclusions

We have proposed a new method to generate an allophone-based LR table that solves the word juncture problem. Furthermore it allows us to predict allophones precisely in speech recognition. By combining allophone rules with syntactic and lexical rules, an initial LR table is generated. Next on the basis of an allophone connection matrix the LR table is modified by using CPM.

The characteristics of our approach can be summarized as follows:

(1) Generation of allophone-based LR table is performed at two levels, this enables us to use the existing LR table generation algorithms to generate an initial allophone-based LR table.

(2) By introducing an allophone connection matrix and applying CPM, a large number of illegal states and actions of initial LR table can be deleted, which reduces consderably the tables' size. In addition is solves the word juncture problem. Last but not least, the reduction of actions and states provides us with accurate allophone predictions in speech recognition.

(3) Our method allows to solve the word juncture without any changes in the GLR parsing algorithm: the connection constraints between two adjacent allophones have been incorporated into an LR table.

While our method allows to use existing LR table generation methods in order to get the initial LR table, the number of states and actions of initial LR table often grow exponentially as the number of syntactic and lexical rules or allophone models increases. In order to improve this situation, we can modify the LR table generation algorithm slightly in order to incorporate the allophone connection constraints(step (a) and (b) in Section 3) during the generation process. In the case of 1024 allophones for the above example grammar, the number of the states of the original LR table decreases to 1/6 by changing the LR table generation algorithm.

Future works will address this problem by incorporating the LR table generated by CPM into an allophone-based continuous speech recognition system.

Part II: Integration of Morphological and Syntactic Analysis based on GLR Parsing Algorithm

1 Introduction

Morphological analysis of Japanese is very different from that of English, because no spaces are placed between words. This is also the case in many other Asian languages such as Korean, Chinese, Thai and so forth. In the Indo-European family, some languages such as German have the same phenomena in the form of complex noun phrases. Processing such languages requires the identification of the word and its corresponding morphosyntactic correct category. This process is often called *segmentation*. Segmentation is one of the most important tasks of morphological analysis for these languages, since the wrong segmentation causes fatal errors in the later stages such as syntactic, semantic and contextual analysis. However, correct segmentation is not always possible only on the basis of morphological information. Syntactic, semantic and contextual information are also necessary in order to resolve ambiguities during the process of segmentation.

From a procedural pointof view , it is preferable to integrate the morphological and syntactic analysis into a single framework, since some syntactic constraints are useful for morphological analysis and vice versa. There have been several attempts to develop CFG that covers both the morphological and syntactic constraints [3][8]. However, it is empirically difficult to describe both constraints by using only CFG. In order to have CFG rules include morphological constraints, nonterminal symbols have to bear the morphological attributes which are used for checking connectabilities between morphemes. In other words, nonterminals should be more precisely subcategorized. This increases the number of nonterminals, hence the number of grammar rules.

Using augmented context free grammar (ACFG) instead of CFG may remedy this problem. However, this may cause the delay of connectability checking. For example, in Figure 1, in order to check the connectability between adjacent words, w_i and w_{i+1}, the morphological attributes of each word should be propagated up to their mother nodes B and C, and the check is delayed until the application of the rule A → B C.

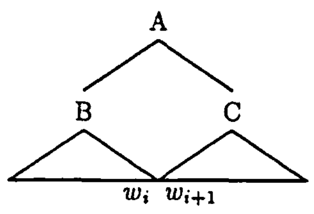


Fig. 1 Connectability check by CFG

Our method represents the morphological constraints in connection matrices and the syntactic constraints in CFGs, then compiling both constraints into an LR table [1]. An already existing, efficient GLR parsing algorithms would be used with minor modifications, enabling us to use concurrently both the morphological and syntactic constraints.

2 Generating LR table

To illustrate our method, we use a simple Japanese grammar as shown in Figure 2.

(1)	s → v	(4)	v → vs ve
(2)	s → v ax	(5)	pp → n p
(3)	s → pp s		

Fig. 2 An example of CFG for Japanese

A connection matrix shown in Figure 3 gives us the connectabilities between adjacent morphological categories (mcat).

		RIGHT																
		n_1	p_1	vs_k	vs_r	vs_w	ve_k^2	ve_k^{2i}	ve_k^3	ve_w^2	ve_w^{2t}	ve_w^3	ve_r^2	ve_r^{2t}	ve_r^3	ax_1	ax_2	\$
	n_1		1															
	p_1			1	1	1												
	vs_k						1	1	1									
	vs_w									1	1	1						
	vs_r												1	1	1			
L	ve_k^2															1		
E	ve_k^{2i}																1	
F	ve_k^3	1																1
T	ve_w^2															1		
	ve_w^{2t}																1	
	ve_w^3	1																1
	ve_r^2															1		
	ve_r^{2t}																1	
	ve_r^3	1																1
	ax_1																	1
	ax_2																	1

Fig. 3 An example of a connection matrix

In order to combine connection matrices and CFG rules, the first step we have to take is to extend the CFG rules by relating the syntactic categories in the CFG rules with the morphological categories in a connection matrix. This is realized by adding CFG rules called morphological rules each of which is a unit production rule with a syntactic category in the LHS and a morphological category in the RHS.

(6)	n $\rightarrow n_1$	(14)	ve $\rightarrow ve_w^2$
(7)	p $\rightarrow p_1$	(15)	ve $\rightarrow ve_w^{2t}$
(8)	vs $\rightarrow vs_k$	(16)	ve $\rightarrow ve_w^3$
(9)	vs $\rightarrow vs_w$	(17)	ve $\rightarrow ve_r^2$
(10)	vs $\rightarrow vs_r$	(18)	ve $\rightarrow ve_r^{2t}$
(11)	ve $\rightarrow ve_k^2$	(19)	ve $\rightarrow ve_r^3$
(12)	ve $\rightarrow ve_k^{2i}$	(20)	ax $\rightarrow ax_1$
(13)	ve $\rightarrow ve_k^3$	(21)	ax $\rightarrow ax_2$

Fig. 4 A set of morphological rules

entry	cat	mcat	meaning
かお	n	n_1	face
かおる	n	n_1	person's name
に	p	p_1	(dative)
あ	vs	vs_k	open
き	ve	ve_k^2	(connect to verb)
い	ve	ve_k^{2i}	(connect to verb)
く	ve	ve_k^3	(connect to nominal)
あ	vs	vs_w	meet
い	ve	ve_w^2	(connect to verb)
っ	ve	ve_w^{2t}	(connect to verb)
く	ve	ve_w^3	(connect to nominal)
かお	vs	vs_r	smell sweet
り	vs	ve_r^2	(connect to verb)
っ	ve	ve_r^{2t}	(connect to verb)
る	ve	ve_r^3	(connect to nominal)
ます	ax	ax_1	(polite form)
た	ax	ax_2	(past form)

n:	noun	ve:	verb ending
p:	case marker	ax:	auxiliary verb
vs:	verb stem		

Fig. 5 An example of Japanese dictionary

We would generate an LR table as shown in Figure 6 from the extended CFG rules (1) through (10) of Figure 2 and 4. The extended CFG rules as well as the LR table do not include any connection constraints in the connection matrix of Figure 3. In order to include the connection constraints, CPM is applied to the LR table in Figure 6. Because of the small size of the grammar and connection matrix, CPM deletes only three actions marked with "*" in Figure 6.

It is possible to incorporate some steps of CPM into the LR table generation process[6], however, it is better to keep them apart in case the case of a small grammar. Generally speaking, the size of the LR table ghrows exponentially with the number of rules of the grammar. Introducing the morphological rules into the syntactic rules can cause an increase in the number of states in the LR table, thereby increasing exponentially the size of the overall LR table in very unfavorable cases.

In our method, the increase of the number of states is equal to that of the morphological rules introduced in the case of SLR table. Suppose we add a morphological rule $X \rightarrow x$ to the grammar. Only the items in the form of $[A \rightarrow \alpha \cdot X\beta]$ would produce a single new item $[X \rightarrow \cdot x]$ from which only a single new state $\{[X \rightarrow x \cdot]\}$ would be created. Thus the increase of the number of the states is equal to that of the morphological rules introduced, hence the size of the LR table does not grow exponentially.

3 Algorithm

The LR parsing algorithm with the modified LR table is principally the same as Tomita's generalized LR parsing algorithm. The only difference is that Tomita's algorithm assumes a sequence of preterminals as an

A C T I O N

state	n_1	vs_k	vs_w	vs_r	ax_1	ax_2	ve_k^2	ve_k^{2i}	ve_k^3	ve_w^2	ve_w^{2i}	ve_w^3	ve_r^2	ve_r^{2i}	ve_r^3	p_1	$
0	s6	s7	s8	s9													
1																	acc
2					s11	s12											r1
3	s6	s7	s8	s9													
4							s15	s16	s17	s18	s19	s20	s21	s22	s23		
5																s25	
6																r6	
7							r8	r8	r8	$\boxed{\text{r8}}$	$\boxed{\text{r8}}$	$\boxed{\text{r8}}$	$\boxed{\text{r8}}$	$\boxed{\text{r8}}$	$\boxed{\text{r8}}$		
8							$\boxed{\text{r9}}$	$\boxed{\text{r9}}$	$\boxed{\text{r9}}$	r9	r9	r9	$\boxed{\text{r9}}$	$\boxed{\text{r9}}$	$\boxed{\text{r9}}$		
9							$\boxed{\text{r10}}$	$\boxed{\text{r10}}$	$\boxed{\text{r10}}$	$\boxed{\text{r10}}$	$\boxed{\text{r10}}$	$\boxed{\text{r10}}$	r10	r10	r10		
10																	r2
11																	r20
12																	r21
13																	r3
14					r4	r4											r4
15					r11	$\boxed{\text{r11}}$											$\boxed{\text{r11}}$
16					$\boxed{\text{r12}}$	r12											$\boxed{\text{r12}}$
17					$\boxed{\text{r13}}$	$\boxed{\text{r13}}$											r13
18					r14	$\boxed{\text{r14}}$											$\boxed{\text{r14}}$
19					$\boxed{\text{r15}}$	r15											$\boxed{\text{r15}}$
20					$\boxed{\text{r16}}$	$\boxed{\text{r16}}$											r16
21					r17	$\boxed{\text{r17}}$											$\boxed{\text{r17}}$
22					$\boxed{\text{r18}}$	r18											$\boxed{\text{r18}}$
23					$\boxed{\text{r19}}$	$\boxed{\text{r19}}$											r19
24	r5	r5	r5	r5													
25	$\boxed{\text{r7}}$	r7	r7	r7													

G O T O

state	s	v	pp	vs	n	ax	ve	p
0	1	2	3	4	5			
1								
2						10		
3	13	2	3	4	5			
4							14	
5								24

Fig. 6 LR table generated from rules (1)-(21)

input, while our algorithm assumes a sequence of characters. Thus the dictionary reference process needs to be slightly modified. Figure 7 illustrates the outline of our parsing algorithm.

```
(1)  initialize stack
(2)  for CS = 0 ... N {
(3)    for each stack top node in stage CS {
(4)      Look-aheads = lookup-dictionary(CS);
(5)      for each look ahead preterminal LA in Look-aheads {
(6)        do reduce while "reduce" is applicable;
(7)        if "shift" is applicable {
(8)          do shift creating a new node in stage (CS + length(LA));
(9)        }
(10)       if "acc" { accept }
(11)       if no action { reject }
(12)     }
(13)   }
(14) }
```

Fig. 7 Outline of the parsing algorithm

In Figure 7 the stage number (CS) indicates how many characters have been processed. The procedure begins at stage 0 and ends at stage N, the character length of an input sentence. In stage 0, the stack is initialized and only the node with state 0 exists (step (1)). In the outer-most loop (2)–(14), each stack top in the current stage is selected and processed. In step (4), the dictionary is consulted and look-ahead symbols are obtained. An important point here is that look-ahead symbols may have different character lengths. A new node is introduced by a shift action at step (8) and is placed into a stage which is ahead of the current stage by the length of the look-ahead word.

4 A worked example

The following example well illustrates the algorithm in Figure 7. The input sentence is "かおるにあいます$ " (meet Kaoru). We assign position numbers between adjacent characters.

Input:	か	お	る	に	あ	い	ま	す	$
Position:	0	1	2	3	4	5	6	7	8 9

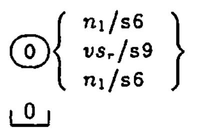
In the following trace, the numbers in circles denote state numbers, and the numbers in squares denote the subtree number. The symbols enclosed by curly brackets denote a look ahead symbol followed by the next applicable action, separated by a slash (/). The stage numbers are shown below the stacks.

Stage: 0

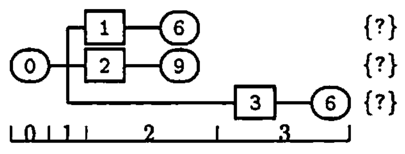
Dictionary reference:

[n_1, "かお"]	at 0–2
[vs_r, "かお"]	at 0–2
[n_1, "かおる"]	at 0–3

We find three look ahead symbols, n_1, vs_r, and n_1 by consulting the dictionary in Figure 5. A shift actions is applied for each of them according to the LR table in Figure 6.



After the shift actions, three new nodes are created at stage 2 or stage 3 depending on the length of look ahead words. At the same time subtrees $\boxed{1}$–$\boxed{3}$ are constructed. The current stage is updated from 0 to 2, since there is no node in stage 1. The look ahead symbols are unknown at this moment.



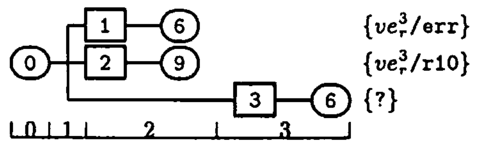
$\boxed{1}$: [n_1, "かお"]
$\boxed{2}$: [vs_r, "かお"]
$\boxed{3}$: [n_1, "かおる"]

Stage: 2

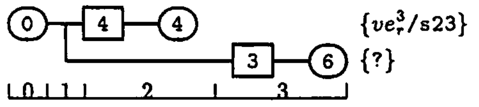
Dictionary reference:

[ve_r^3, "る"]	at 2–3

Dictionary reference gives one look ahead symbol from position 2. Since the current stage number is 2, only the first two stack tops are concerned at this stage. No action is taken of the first stack, because the LR table has no action in the entry for state 6 and a look ahead symbol ve_r^3. As the result, the first stack is rejected. The reduce action (r10) is taken for the second stack.

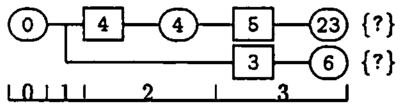


After r10, a shift action (s23) is carried out for the first stack.



$\boxed{4}$: [vs, $\boxed{2}$]

After s23, we would proceed to stage 3.



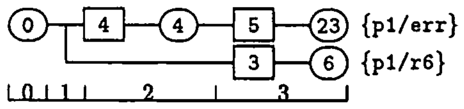
$\boxed{5}$: [ve_r^3, "る"]

Stage: 3

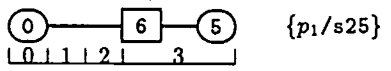
Dictionary reference:

[p_1, "に"]	at 3–4

We obtain symbol p_1 by consulting the dictionary. Because the first stack can take no more action, it is rejected. The reduce action (r6) is then applied to the second stack.

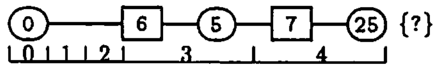


The shift action (s25) is applied to the following stack.



$\boxed{6}$: [n, $\boxed{3}$]

After the shift action (s25), new nodes are created in stage 4.



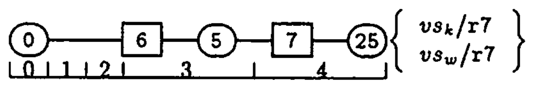
$\boxed{7}$: [p_1, "に"]

Stage: 4

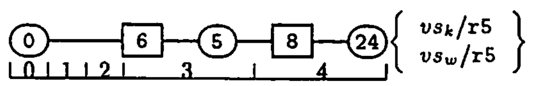
Dictionary reference:

[vs_k, "あ"]	at 4–5
[vs_w, "あ"]	at 4–5

Dictionary reference provides two look ahead symbols for the next word.

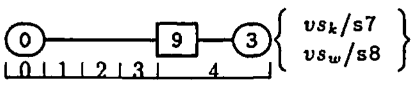


After the two reduce actions (r7), we get two nodes with the same state 24, and they would be merged. This is possible because these two reduce actions give the same structure as well. If the structures are different, we would not able to merge the stacks. We would see such an example later at stage 5.

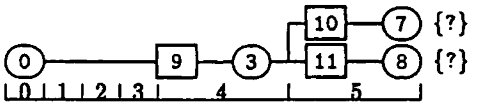


$\boxed{8}$: [p, $\boxed{7}$]

The process in stage 4 continues as follows.



$\boxed{9}$: [pp, $\boxed{6}$, $\boxed{8}$]



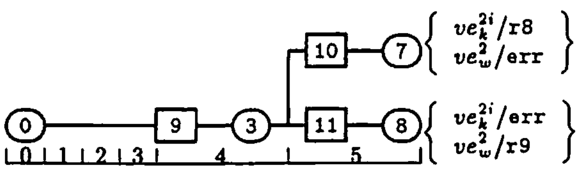
$\boxed{10}$: [vs_k, "あ"]
$\boxed{11}$: [vs_w, "あ"]

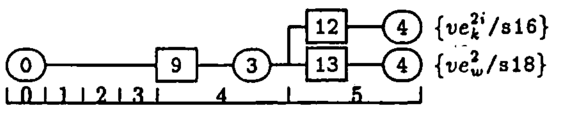
Stage: 5

Dictionary reference:

[ve_k^{2i}, "い"]	at 5–6
[ve_w^2, "い"]	at 5–6

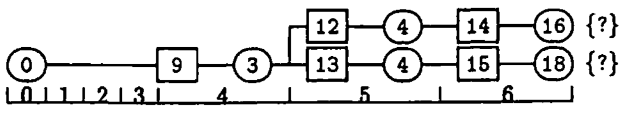
We have two look ahead symbols for each stack top. The reduce actions (r8 and r9) are performed.





$\boxed{12}$: [vs, $\boxed{10}$]
$\boxed{13}$: [vs, $\boxed{11}$]

Note that we are not able to merge the stack tops even with the same state 4 since the structure of $\boxed{12}$ and $\boxed{13}$ are different. If two stack tops are merged here and then different shift actions (s16 and s18) are carried out, we might have invalid combinations of structure such as ($\boxed{12}$, $\boxed{15}$) and ($\boxed{13}$, $\boxed{14}$).



$\boxed{14}$: [ve_k^{2i}, "い"]
$\boxed{15}$: [ve_w^2, "い"]

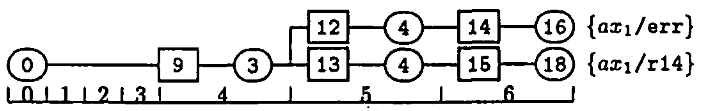
After the shift actions (s16 and s18), we proceed to stage 6.

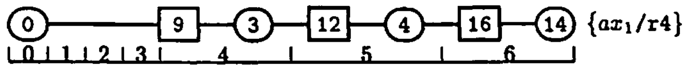
Stage: 6

Dictionary reference:

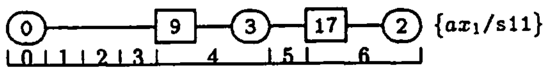
[ax_1, "ます"]	at 6–8

The process in stage 6 proceeds as follows.

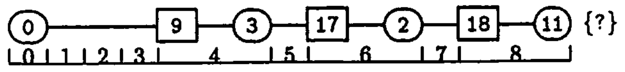




$\boxed{16}$: [ve, $\boxed{15}$]



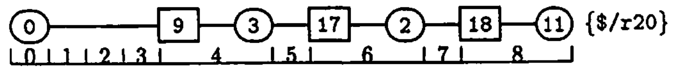
$\boxed{17}$: [v, $\boxed{12}$, $\boxed{16}$]

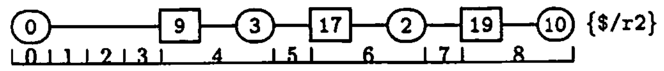


$\boxed{18}$: [ax_1, "ます"]

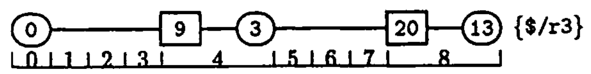
Stage: 8

Dictionary reference:
"$"   at 8–9



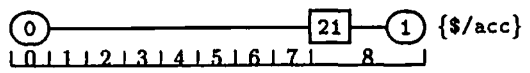


$\boxed{19}$: [ax, $\boxed{18}$]



$\boxed{20}$: [s, $\boxed{17}$, $\boxed{19}$]

The input sentence is automatically segmented and accepted, giving a final parse result 21 as shown in Figure 8.



$\boxed{21}$: [s, $\boxed{9}$, $\boxed{20}$]

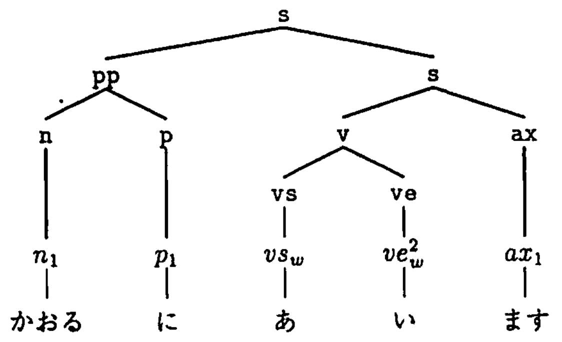


Fig. 8 An analysis of "かおるにあいます"

5 Conclusion

We have proposed a method representing the morphological constraints in connection matrices and the syntactic constraints in CFGs, then compiling both constraints into an LR table. The compiled LR table enables us to make use of the already existing, efficient generalized LR parsing algorithms through which integration of both morphological and syntactic analysis is obtained.

Advantages of our approach would be summarized as follows:

- Morphological and syntactic constraints are represented separately, and it makes easier to maintain and extend them.

- The morphological and syntactic constraints are compiled into a uniform representation, an LR table. We can use the already existing efficient algorithms for generalized LR parsing for the analysis.
- Both the morphological and syntactic constraints can be used at the same time during the analysis.

We have implemented our method using the EDR dictionary with 300,000 words from which 437 morphological rules are derived. This means only 437 new states are introduced to LR table and this does not cause an explosion in the size of the LR table. The method proposed in this part is also applicable to integrate phonological and syntactic analysis. The detail is described elsewhere [10].

References

[1] A.V. Aho, S. Ravi, and J.D. Ullman. *Compilers: Principle, Techniques, and Tools.* Addision Wesely, 1986.

[2] K. Itou, S. Hayamizu, and H. Tanaka. Continuous speech recognition by context-dependent phonetic HMM and an efficient algorithm for finding N-best sentence hypotheses. In *Proc. ICASSP-92*, pages I–21–I–24, 1992.

[3] K. Kita. Study on language modeling for speech recognition. In *PhD thesis, Waseda University*, 1992.

[4] K. Kita, T. Kawabata, and H Saito. HMM continuous speech recognition using predictive LR parsing. In *Proc. ICASSP-89*, pages 703–706. IEEE, 1989.

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

[6] H. Li, K.G. Suresh, and H. Tanaka. Incorporation of connection constraints into the generation process of allophone-based LR table. In *Proc. of the Information Processing Society of Japan(IPSJ) spring meeting*, pages 463–464, March 1995.

[7] A. Nagai, S. Sagayama, K. Kita, and H. Kikuchi. Three different lr parsing algorithms for phoneme-context-dependent hmm based continuous speech recognition. *IEICE Trans. Inf. & Syst.*, E76-D(1):29–37, 1993.

[8] H. Sano and F. Fukumoto. On a grammar formalism, knowledge bases and tools for natural language processing in logic programming. In *Proceedings of FGCS92*, 1992.

[9] R. Schwartz, Y. Chow, O. Kimball, S. Roucos, M. Krasner, and J. Makhoul. Context dependent modeling for acoustic phonetic recognition of continuous speech. In *Proc. ICASSP-85*, pages 1205–1208. IEEE, 1985.

[10] H. Tanaka, H. Li, and T. Tokunaga. Incorporation of phoneme-context-dependence into LR table through constraints propagation method. In *Proc. AAAI-94 Workshop on Integration of Natural Language and Speech Processing*, pages 15–22, Seattle, 1994.

[11] H. Tanaka, T. Tokunaga, and M. Aizawa. Integration of morphological and syntactic analysis based on LR parsing algorithm. In *Proc. International Workshop on Parsing Technologies*, pages 101–109, Tilburg, 1993.

[12] M. Tomita. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers, Boston, MA, 1986.