A Method for Integrating the Connection Constraints into an LR Table

LI Hui and TANAKA Hozumi
Department of Computer Science
Tokyo Institute of Technology
2-12-1 Ôokayama, Meguro, Tokyo 152, Japan
{li, tanaka}@cs.titech.ac.jp

Abstract

This paper presents an algorithm for integrating local constraints, in the form of a connection matrix, into an LR parsing table generated from a set of CFG rules. The algorithm first incorporates connection constraints into the generation process of LR items to construct an initial LR table, and then modifies the initial LR table through constraint propagation. With this algorithm, an LR table that compiles both global constraints in CFG and local constraints in a connection matrix can be constructed efficiently.

1 Introduction

Generalized LR(GLR) parsing technique (Tomita, 1986) has been widely used to parse context-free grammars (CFGs), in natural language processing. Recently, GLR parsing has also been successfully applied in continuous speech recognition to provide grammatical constraints (Kita et al., 1989).

CFGs, as a global constraint, are usually used for syntactic analysis. In addition to global constraints, local constraints are also used in some applications. For example, in natural language processing, the morphological analysis of many Asian languages, such as Japanese, Chinese and Korean, is very different from that of English, because no spaces are placed between words. For these languages, the local constraint between two adjacent words is often used for morphological analysis. In order to integrate the morphological and syntactic analyses, a method that combines both morphological constraints and syntactic constraints(CFGs) into a GLR parsing has been proposed (Tanaka et al., 1993).

As an another example, in continuous speech recognition, the search space can be reduced by way of grammatical constraints based on CFGs (Kita et al., 1989). On the other hand, phoneme-context-dependent phone models (allophone models) can be used to improve the recognition accuracy with the connection constraint between two allophones represented in a connection matrix. Compiling both grammatical constraints and allophonic constraints into GLR parsing will enable us to drive an allophone-based speech recognition system by way of GLR parsing (Nagai et al., 1993)(Tanaka et al., 1994).

The GLR parser is guided by an LR parsing table (Aho et al., 1986), which is automatically created from CFGs, and proceeds left-to-right without backtracking. The heart of the LR parsing technique is the parse table construction algorithm, which is the most complex and computationally expensive aspect of LR parsing. For a GLR parser that integrates both grammatical and connective constraints, an LR parsing table that combines both grammatical and connective constraints is necessary in order to use the usual GLR parsing methods such as Tomita's algorithm (Tomita, 1986).

We have previously proposed a method, called CPM(constraint propagation method), for generating an LR table that incorporated the local constraint in the form of a connection matrix (Tanaka et al., 1994). In CPM, the LR table is constructed in two steps: 1) An initial LR table from the given CFGs is generated; 2) The initial LR table is mod-

ified by using the given connection matrix through constraint propagation. The major problem with CPM is that for a large scale grammar, the explosion of the initial LR table size may occur.

In this paper, we propose a method to integrate local constraints, represented in the form of a connection matrix, into the generation process of the initial LR table. By using this method, the explosion of the initial LR table size can be avoided.

In the following sections, we first describe the language model used in this paper. We then point out the drawbacks of the CPM method, and describe an algorithm to incorporate connection constraints into the generation process of an LR table. Finally, experimental results of LR table generation are presented.

2 CFG and Connection Matrix

A "context-free grammar(CFG)" is a 4-tuple $G =< V_N, V_T, P, S >$, where V_N is a nonterminal symbol set including the start symbol S, V_T is a terminal symbol set, and P is a set of production rules each of which is of form $< A \rightarrow \alpha >$, with $A \in V_N$, $\alpha \in (V_N \cup V_T)$.

Fig. 1 shows an artificial CFG, G_1, where $V_N = \{S, X, Y, A, B, C\}$ and $V_T = \{a1, a2, b1, b2, c1, c2\}$ correspond to the nonterminal and terminal symbol set, respectively.

(1)	S → X Y	(6)	A → a2
(2)	X → A	(7)	B → b1
(3)	X → A B	(8)	B → b2
(4)	Y → b1 C	(9)	C → c1
(5)	A → a1	(10)	C → c2

Figure 1: An example of CFG rules

The connection constraint between two terminal symbols can be represented in a connection matrix, which is of the form:

$$Connect(a,\ b) = \begin{cases} 1 & \text{if } b \text{ can follow } a \\ 0 & \text{if } b \text{ can not follow } a \end{cases} \quad (1)$$

where $a,\ b \in V_T$.

Fig. 2 shows an example of a connection matrix for the terminal symbols of Fig. 1.

Certainly, connection constraints can be incorporated into an CFG through introducing new nonterminal symbols and modifying the original CFG rules. For example, in Fig. 3, if $\{b_i, i = 1, \cdots, I\}$ corresponds to the last functions of B, and $\{c_j, j = 1, \cdots, J\}$ corresponds to the first functions of C, in order to represent the connectability between b_i and c_j in CFG, we should introduce the new nonterminal symbols $\{B_i, i = 1, \cdots, I\}$, where B_i corresponds to that B whose last function is b_i, and the new nonterminal symbols $\{C_j, j = 1, \cdots, J\}$, where C_j corresponds to that C whose first function is c_j. The production rule $< A \rightarrow B\ C >$ will become a set of rules of the form $\{< A \rightarrow B_i\ C_j >, \forall B_i, C_j$, if b_i and c_j are connectable$\}$. It is obvious that for a large scale grammar, if the connection constraint is integrated in this way, the explosion of CFG rules, and consequently the explosion of LR table size may occur. For example, for a CFG with 1653 rules, after the connection constraint was integrated, the number of CFG rules increased to 61507, and the number of LALR table states increased from 12207 to 71030 (Nagai et al., 1993). Accordingly, for a large scale grammar, it is impractical to incorporate the connection constraints into an CFG.

		RIGHT					
		a1	a2	b1	b2	c1	c2	$
LEFT	a1	0	0	1	0	0	0	0
	a2	0	0	0	1	0	0	0
	b1	0	0	1	0	0	1	0
	b2	0	0	0	1	1	0	0
	c1	0	0	0	0	0	0	0
	c2	0	0	0	0	0	0	1

Figure 2: An example of a connection matrix

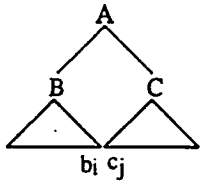

Figure 3: Connectability check by CFG

In fact, it is possible to integrate connection constraints into the LR table, but not change the CFGs. This can be done through modifying the LR table generated from the CFGs by using connection constraints. Since no change occurs in the CFG, the explosion of the CFGs and LR table size will not occur. We will describe this method in the next section.

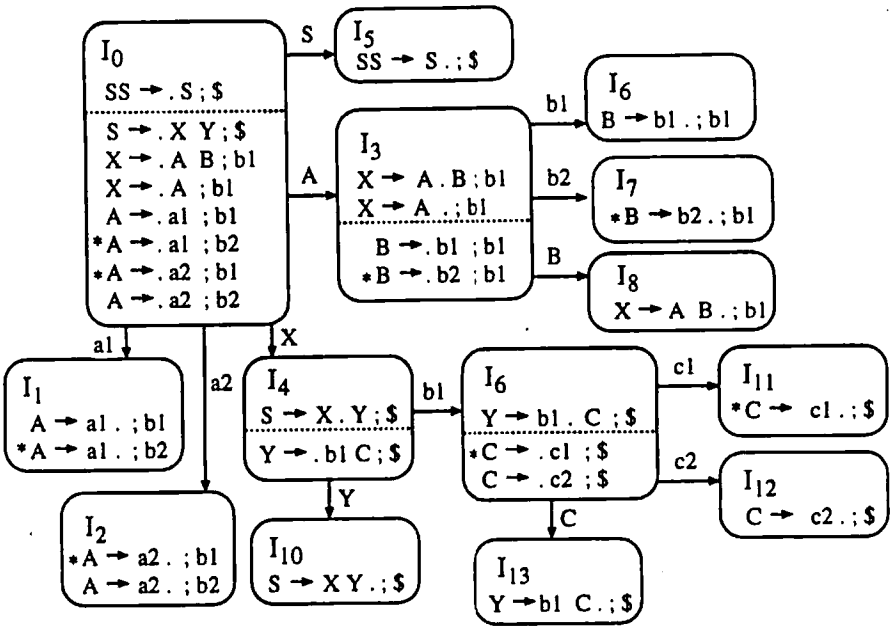

Figure 4: LR(1) items for the grammar in Fig. 1

3 Integration of the connection constraint into the LR table

In this section, we review the CPM (Tanaka et al., 1994) and point out its problems, following which we propose an algorithm to integrate connection constraints into the generation process of LR items.

3.1 CPM Method and its Problems

In CPM, given a set of CFG rules and a connection matrix between two terminal symbols, the LR table that combines both grammatical and connective constraints is constructed in two steps:

1. From the given CFGs, an LR table, called the initial LR table, is constructed.
2. On the basis of the given connection matrix, the initial LR table is modified by deleting all the illegal actions that violate connection constraints.

In many applications, the connection matrix tends to be a sparse matrix. In such a case, for a large scale grammar, although CPM can use the existing method to generate the initial LR table, the explosion of the size of the initial LR table may occur. For a grammar with 2655 CFG rules which we used in a speech recognition system, for instance, the necessary memory space of the LALR table was about 25MB. The necessary memory space usually increases very rapidly with the size of the grammar. Therefore, in the case of a large scale grammar, CPM is impractical in terms of memory requirements. This problem originates in the fact that connection constraints are not considered when constructing the initial LR table. One of the solutions to this problem is modifying the algorithm of LR table generation, avoiding constructing any illegal items that violate connection constraints, by using the given connection matrix, that is, incorporating connection constraints into the generation process of LR items. With this method, a large reduction in the necessary memory for generating LR table can be achieved.

For simplicity, we use the generation of LR(1) items (corresponding to the canonical LR table) as an example to illustrate our method.

3.2 Generation of LR(1) items with connection constraint

3.2.1 LR(1) item

The generation of a canonical LR parsing table is a two stage process which involves first the generation of a finite number of item sets from the grammar, and second the generation of the action table from these sets. An LR(1) item takes the form $< A \rightarrow \alpha \cdot \beta,\ a >$, where "$a$" is one of a list of terminal symbols which can occur as the first symbol immediately following the string derived using the rule $< A \rightarrow \alpha\ \beta >$ for that particular

state.

Fig. 4 shows the canonical collection of sets of LR(1) items for the grammar in Fig. 1.

3.2.2 *Algorithm for generating LR(1) items, incorporating connection constraints*

The basic idea to incorporate connection constraints into the generation of LR(1) items is: when a new item is to be added to an item set, the connectability between it and the preceding or succeeding symbol should be checked; if the connection constraint is not fulfilled, this item will not be added to the item set.

As we know, the items in an item set can be divided into two classes: kernel items and nonkernel items. The kernel items are from one or more preceding states by goto functions, and the nonkernel items are created from the kernel items.

For an item set I, we define the preceding symbol as follows.

Definition 1 (*preceding symbol*)

For an LR(1) item set I, if the kernel item of I is of form $[A \rightarrow \alpha \cdot \beta, a]$, the symbol α, which is immediately before the dot, is called the preceding symbol of I.

For example, the preceding symbol of the item set I_3 in Fig. 4 is nonterminal symbol "A".

The closure operation which defines how a state is constructed can be described as follows.

(i) Suppose that α be the preceding symbol of the item set I, and the computation of closure(I) from the given grammar requires that item $< B \rightarrow \cdot\gamma,\ b >$ be added to I.

(ii) Let $\{c_i, i = 1, \cdots, L\}$ be the last functions of α, $\{d_j, j = 1, \cdots, M\}$ the first functions of γ, $\{e_k,\ k = 1,\ \cdots,\ N\}$ the last functions of γ,

$$Connect(c_i, d_j) = 0,\ for\ \forall\ i,\ j \qquad (2)$$

means that α and γ are not connectable, and

$$Connect(e_k, b) = 0,\ for\ \forall\ k \qquad (3)$$

means that γ and b are not connectable.

(iii) Add the item "$< B \rightarrow \cdot\gamma,\ b >$" to I when both expressions (2) and (3) are not fulfilled.

In the above algorithm, when α or β is a nonterminal symbol, it is necessary to compute the connectability dynamically by using first or last functions. For a large scale grammar, there is a]the potential for the connectability between two symbols to be recomputed many times during the generation process of LR items. To avoid this overlap computation, the connectability between two symbols can be computed in advance by extending the connection matrix from between only terminal symbols, to between any symbols.

Definition 2 (*extended connection matrix*)

For $\alpha,\ \beta \in (V_T \cup V_N)$, let $\{a_i, i = 1, \cdots, k\}$ be the last functions of α and $\{b_j, j = 1, \cdots, m\}$ the first functions of β. The *extended connection matrix* $Connect'(\alpha,\ \beta)$ between α and β is defined as follows.

$$Connect'(\alpha,\ \beta) = \begin{cases} 0, & \text{if } Connect(a_i,\ bj) = 0 \text{ for } \forall \text{ i, j} \\ 1, & \text{otherwise} \end{cases} \qquad (4)$$

As an example, the extended connection matrix constructed from the grammar in Fig. 1 and the connection matrix in Fig. 2 is shown in Fig. 5.

		RIGHT											
		a1	a2	b1	b2	c1	c2	$	A	B	C	X	Y	S
LEFT	a1	0	0	1	0	0	0	0	0	1	0	0	1	0
	a2	0	0	0	1	0	0	0	0	1	0	0	1	0
	b1	0	0	1	0	0	1	0	0	1	1	0	1	0
	b2	0	0	0	1	1	0	0	0	1	1	0	0	0
	c1	0	0	0	0	0	0	0	0	0	0	0	0	0
	c2	0	0	0	0	0	0	1	0	0	0	0	0	0
	A	0	0	0	0	0	0	0	0	1	0	0	1	0
	B	0	0	0	0	0	0	0	0	1	1	0	1	0
	C	0	0	0	0	0	0	0	0	0	0	0	0	0
	X	0	0	0	0	0	0	0	0	0	0	0	1	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0
	S	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5: An extended connection matrix

With the definitions about the extended connection matrix and the preceding symbol, the algorithm for constructing the closure is described in Fig. 6. Using this algorithm, the generation of illegal items that violate the connection constraint can be avoided. As an example, consider the item sets in Fig. 4,

1. In I_0, since $Connect'(a1, b2) = 0$ and $Connect'(a2, b1) = 0$, the following two items

$$A \rightarrow \cdot a1;\ b2$$
$$A \rightarrow \cdot a2;\ b1$$

will not be created according to the procedure of closure construction. Moreover, since the

```
function closure(I);
begin
  δ := preceding symbol of I ;
  repeat
    for each item [A → α · Bβ, a] in I
      for each production B → γ in G'
        for each terminal b in FIRST(βa)
          if [B → ·γ, b] is not in I then
            if Connect'[δ, γ] = 1
              and Connect'[γ, b] = 1 then
                add [B → ·γ, b] to I;
  until no more items can be added to I;
  return I
end;
```

Figure 6: Computation of closure

above two items are not generated, the item "$A \rightarrow a1\cdot$; $b2$" in I_1 and the item "$A \rightarrow a2\cdot$; $b1$" in I_2 will vanish automatically.

2. In I_9, the preceding symbol of this item set is "b1", since $Connect'(b1, c1) = 0$, the item

$$C \rightarrow \cdot c1;\ \$$$

will not be generated, and then the item set I_{11} that transferred from above item by shifting "c1" will vanish automatically.

In Fig. 4, all items that are not created according to our method are marked "*". As a result, the LR table from the modified item sets is shown in Fig. 7.

3.3 *Constraint propagation*

By using the above closure computation algorithm, the generation of a large number of illegal items, namely, a large number of illegal actions of the LR table, has been avoided.

However, some useless actions still exist in the LR table generated by the above method. Consider the action "re6" with lookahead "b2" in state 2, after it has been carried out, the parser will transfer to state 3, but in state 3, no action with the same lookahead "b2" exists. This means no succeeding action for "re6". In fact, during the generation process, the item $< B \rightarrow \cdot b2\ ;\ b1 >$ in I_3 was not constructed since $Connect(b2, b1) = 0$ (see Fig. 4). For actions like this, since no succeeding action exists, carrying out these actions only leads to an error for a GLR parser. On the other hand, for some actions, there may be no

	Action							Goto					
	a1	a2	b1	b2	c1	c2	$	A	B	C	X	Y	S
0	s1	s2						3			4		5
1			r5										
2				r6									
3			s6/r2						8				
4			s9									10	
5							acc						
6			r7										
7*													
8			r3										
9						s12				13			
10							r1						
11*													
12							r10						
13							r4						

Figure 7: The canonical LR table, incorporating connection constraints

preceding actions, meaning that these actions will not be carried out for a GLR parser. In the above two circumstances, actions that have no succeeding or preceding actions are useless actions for a GLR parser. To save memory space and improve the parsing efficiency, we modify the LR table to delete these useless actions through constraint propagation, this procedure shown in Fig. 8.

```
function modify(Table);
begin
  repeat
    for each action A in Table
      if no preceding actions then
        delete A
      else if no succeeding actions then
        delete A
  until no more actions can be deleted;
  return Table
end;
```

Figure 8: Procedure of constraint propagation

In the above example, action "re6" (lookahead "b2") in state 2 has no succeeding actions. and as such, it can be deleted according to the procedure in Fig. 8.

Although we only discussed how to incorporate the connection constraint into a canonical LR table, the proposed method can be applied to both an SLR and LALR table. In the case of an SLR ta-

ble, instead of checking the connectability between the item and its lookahead symbol, we should check the connectability between the item and its follow functions.

4 Experiments

A grammar with 1208 rules for a continuous speech recognition system was used to test the effects of the proposed method. This grammar consisted of 1024 terminal symbols. The connection constraint between the terminal symbols was represented in a connection matrix.

Table 1 shows the comparisons of the initial LR table size, where (1) indicates the initial canonical LR table that did not combine the connection constraint, (2) indicates the initial canonical LR table that combined connection constraints with the method proposed in this paper. For the sake of comparison, the final LR table processed by constraint propagation is listed in (3).

method	state	shift	reduce	goto
(1)	10,715	20,573	633,409	421
(2)	1,141	1,005	5,724	421
(3)	1,139	1,005	1,211	421

Table 1: Comparisons of the initial LR table size

It is clear that the generation of a large number of illegal actions was avoided through incorporating connection constraints into the generation process of the LR table. The number of states, shift actions, and reduce actions of the LR table in (2) is decreased to 11%, 5%, and 1% of those of the LR table in (1), and the number of reduce actions decreased further through constraint propagation.

Since a large number of illegal actions were not generated with method (2), much less CPU time was required for generating the LR table compared with method (1). This is helpful when a grammar is being designed. Here, the grammar is not yet completely fixed, and after each change of the grammar, a new LR table must be generated.

5 Conclusion

This paper proposes a method to integrate connection constraints into an LR table. Through this method, connection constraints are incorporated into the generation process of LR item sets initially, and then all useless actions are deleted through constraint propagation to reduce the memory space of the resulting LR table. The experiments showed that by using the proposed method, the explosion of the size of the initial LR table was avoided. The proposed method has been applied to the integration of Japanese syntactic and morphological analyses and the addition of grammatical constraint into the allophone-based continuous speech recognition system.

Future works will include incorporating two or more connection matrices into an LR table. This is useful in some applications, for example, in an allophone-based continuous speech recognition system, in which allophonic connection constraints, as well as morphological connection constraints should be considered.

References

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

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

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

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

H. Tanaka, H. Li, and T. Tokunaga. 1994. 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.

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