인 지 과 학
Korean Journal of Cognitive Science
Vol.6, No. 3(1995)

트리 접합 문법의 LR파싱 알고리즘

A LR Parsing Algorithm for Tree Adjoining Grammar

한성국†
Sung-Kook Han

요 약

트리접합문법의 LR 상향식 파싱 방법을 제시한다. 먼저 트리접합문법의 형식특성을 기술하기 위한 접합규칙 시스템을 도입하여 파싱과정을 효율적으로 수행할 수 있게 한다. 트리접합문법은 문맥의존성을 갖고 있는데, 접합 순간은 문맥자유문법 체계로 기술할 수 있음을 보이고, 이러한 특성을 기반으로 상향식 파싱방법을 유도한다. 본 논문에서 제시한 LR 상향식 파싱방법은 트리접합문법에 문맥자유문법의 파싱방법을 변형하여 적용할 수 있음을 보인다.

주제어 통사 처리 과정, 구문구조 분석

ABSTRACT

We present a LR, bottom-up parsing algorithms for TAG. We will introduce the adjoining rules system to handle the formal properties of TAG and to describe the parsing process more effectively. We will consider the context-free behavior of TAG at

† 원광대학교 컴퓨터공학과 교수
Won Kwang University, Department of Computer Eng., 344-2, Shin-Young, Ik-San, Chen-Buk, 570-749
e-mail: skhan@wonnms.wonkwang.ac.kr

the adjoining instant. Then we will present the LR bottom up parsing algorithm for TAG by using this property. The basic idea behind a LR bottom up parsing algorithm can be applied to parsing TAG with other conventional algorithms.
Keyword syntactic processing, parsing

1. Introduction

A Tree Adjoining Grammar(TAG) introduced by Joshi et al.[1975] and Joshi[1985] is proved that TAG's are more powerful than CFG, but mildly so. TAG is a tree generating system with an unique operation, adjoining(adjunct). It supports the recursion and localization of dependencies which are essential to describe language structures. This point makes TAG applicable to describe natural language structures even though TAG was studied initially only for its mathematical properties. For further, Joshi[1985], Kroch and Joshi[1985] or Vijay-Shanker[1988] can be referred. Many attempts to apply TAG to Korea are recently increased[Han 1991] [Lee 1995].

Several attempts have been made to parse TAG. In the first practical parser for TAG, especially, Early's algorithm is used to parse TAG on account of its advantages to deal with substitution and feature structures for TAG[Schabes 1988]. In parsing TAG, since TAG is a tree-based system, we do not need to build the whole parsing tree as the conventional parsers do for rewriting systems. If we know the adjoining sequences and the adjoined nodes or their addresses for the given input sentence, we can build the parsing tree by generation or adjunction of trees in TAG. In other words, we already know the parts of parsing tree. Hence, parsing TAG implies to find out the adjoining status. It is not necessary to examine all nodes since only the adjoinable nodes concern the generation of parsing structures.

We will present the LR, bottom-up parsing algorithm for TAG in this sense. In Section 2, we present the adjoining rules system to represent TAG into mathematical expressions like rewriting rules. The adjoining

rule system is convenient to handle the trees in TAG. In Section 3, we will consider the context-free behavior of TAG at the adjoining instant. We also describe parsing algorithm in detail in this section. In Section 4, we will show the extension of parsing algorithm to capture substitution and feature based TAG.

2. TAG and Adjoining Rules System

A tree adjoining grammar(TAG) G=(I, A) is specified by a finite set of elementary trees. The trees in I and A are called the initial trees and the auxiliary trees, respectively. A tree can be an initial tree if it has a form in Fig. 2-1(a): that is, the root and internal nodes of a tree are nonterminal and the frontiers are all terminals. A tree is auxiliary tree if it is of the form in Fig. 2-1(b): that is, the root node of a tree is nonterminal labeled X and the frontier nodes are terminals except one which is nonterminal labeled X, called a foot node. The auxiliary trees that support the unique recursions are used to construct more complex sentences from the skeletal sentential structure.

The major operation, adjoining or adjunction, in TAG is defined as follows[Joshi 1987]. Let α be a tree (α is any tree) containing a node n labeled by X and let β be an auxiliary tree whose root is also

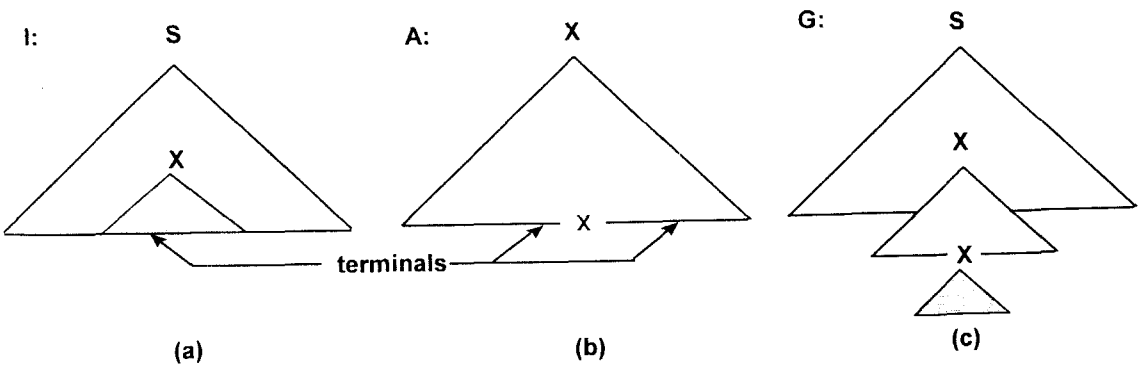

Fig. 2-1 TAG and Adjoining operation

labeled by X. Then the adjunction β to α at node n will be tree γ built as follow:

1.The subtree of α dominated by n, called as t, is excised, leaving a copy of n behind.
2.The auxiliary tree β is attached at n and its root node is identified with n.
3.The subtree t is attached to the foot node of β and the root node n of t identified with foot node of β.

The resulting tree γ called a derived tree, is shown in Fig. 2-1(c). In TAG, all mathematical power come from the adjoining operations. Whenever a new tree is generated by adjoining operation - though the structure represented by trees may have the significant meaning to represent a certain linguistic phenomena - the main operational nodes will be the adjoinable nodes which can be expanded with other trees in TAG. This properties of the adjoinable nodes are similar to those of nonterminal symbols in rewriting systems.

The tree structure are regarded as the efficient representation to describe the structural organizations. However, since the tree structure may be complex or cumbersome to handle the mathematical operations defined on trees on account of their graphical nature of the trees, it may be convenient to use mathematical expressions for representing the tree structures. Such the mathematical expressions should be able to describe all the properties innated or derived from tree operations mathematically. We will define the adjoining rules system for TAG similar to the traditional rewriting systems so that we may treat the trees in TAG more formally.

Definition 2.1 (Adjoining Rules System : ARS) *A adjoining rules system is denotes* $T = (N, \Sigma, P, S, T_n)$, *where N and* Σ *are finite sets of nonterminals and terminals, respectively. P is a finite set of tree adjoining rules; each rule is of the form* $A \rightarrow \alpha$, *where A is a tree name in* T_n, α *is consisted of* $(N \cup \Sigma)^+$ *and S is a starting symbol, i. e., initial tree name in* T_n, *and* T_n *is a finite set of tree names with the empty tree name* ϕ.

The nonterminals in adjoining rules system, which are the adjoinable nodes, root and foot

nodes, are of the form of Φ^{k}_{θ}:(x), where $\theta \subseteq T_n$ represents the adjoining constraints which is the finite set of the adjoinable tree names at node Φ [1]. k is a single tree name adjoined at this nodes and x is the terminal string under Φ, that is, a subtree of Φ. The physical meaning of Φ is a node name or address that θ can be adjoined.

All nonterminals must have their subtree since the nonterminals defined on trees of TAG are the internal nodes[2]. The adjoining rules P for Fig.2-2 will be as follows:

$$I_1 \rightarrow S^{\phi}_{(A1,A2)}:(e)$$
$$A_1 \rightarrow S^{\phi}_{\phi}:(a\ S^{\phi}_{(A1,A2,\phi)}:(b\ S^{\phi}_{\phi}\ c))$$
$$A_2 \rightarrow S^{\phi}_{\phi}:(f\ S^{\phi}_{\phi})$$

We can perceive all information about the corresponding trees through the adjoining rules. Note that the root and the foot are always appeared in the rules, even though there are not adjoinable nodes. Now, let us define the basic operation of TAG, adjoining, on ARS.

Definition 2.2 (Adjoining) *If* $Y_i \rightarrow Y{:}(\beta)$ *is in the adjoining rules P, the adjoining operation on* $X \overset{*}{\Rightarrow} \ldots Y^{k}_{\theta}{:}(\alpha)\ldots$ *will be*

$$X \overset{*}{\Rightarrow} \ldots\ Y^{k}_{\theta}{:}Y^{Y_i}_{\pi}{:}(\beta){:}(\alpha)\ \ldots$$

where $Y_i \in \theta$ *and the nonterminal without subtree in* β *must filled be with* α.

By the definition of adjoining in TAG, the adjoining operation is applied between nonterminal and its subtree. Furthermore the resultant constraints and other properties

1) There are three types of adjoining constraints in TAG: SA, OA and NA. Since the internal nodes with NA, however, cannot be expanded with other trees, we need not to regard these nodes as nonterminals. The nodes with SA will be regarded as the special case of OA constraints with empty tree ϕ.

2) In case of foot node, there is no subtree under Φ at the first time. In the course of adjoining, however, all nonterminals will have their subtrees, That is, foot node is unstable.

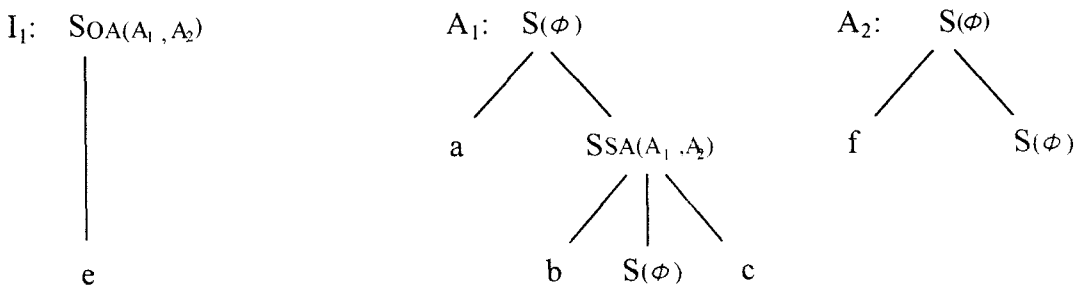

Fig. 2-2 An Example of TAG

after adjoining will be represented by the last nonterminal of cluster of nonterminals. For example, adjoining A_1 and A_2 with I_1 in Fig. 2-2, then the derived elementary tree will be

Therefore the terminal string *afbec* will be obtained from the derived expressions and by the observation of constraints on nonterminals we

$$
\begin{aligned}
I_1 &\rightarrow S^{\phi}_{(A1,A2)} : (e) \\
&\rightarrow S^{\phi}_{(A1,A2)} : S^{A1}_{\phi} : (aS^{\phi}_{(A1,A2,\phi)} : (bS^{\phi}_{\phi} c)) : (e) \\
&= S^{\phi}_{(A1,A2)} : S^{A1}_{\phi} : (aS^{\phi}_{(A1,A2,\phi)} : (bS^{\phi}_{\phi} : (e)c)) \\
&\rightarrow S^{\phi}_{(A1,A2)} : S^{A1}_{\phi} : (aS^{\phi}_{(A1,A2,\phi)} : S^{A2}_{\phi} : (fS^{\phi}_{\phi}) : (bS^{\phi}_{\phi} : (e)c)) \\
&= S^{\phi}_{(A1,A2)} : S^{A1}_{\phi} : (aS^{\phi}_{(A1,A2,\phi)} : S^{A2}_{\phi} : (fS^{\phi}_{\phi} : (bS^{\phi}_{\phi} : (e)c)))
\end{aligned}
$$

can recognize no further adjoining is possible. Above all things, we can realize that the adjoining with A_1 and A_2 are occurred at the specified nodes that can be adjoined[3]. Furthermore, we can construct the derived tree structure from the sentential representation of ARS.

3. Bottom-up Parsing of TAG

We will present an efficient, bottom-up parsing technique for TAG. The proposed parsing method in fact is almost similar to the traditional LR parsing except some minor modification to handle the subtree for foot node. We will explain the underlying principle of LR parsing for TAG with the context-free behavior of TAG at the adjoining instant.

3.1 Context-Freeness of TAG

Let us compare the adjoining operation on TAG to the substitution on the general rewriting rules system. Adjoining I_1 with A_1, the uniqueness of adjoining are in the treatment of subtree for foot node after adjoining. The adjoining must accompany this attachment of the excised subtree for foot node. At this adjoining instant, however, we may regard the trees participating in adjoining operation as context-free rewriting rules[4]. The trees in Fig. 3-1, for example, will be identical to the following context-free rewriting rules.

(3-1) $S \rightarrow a_1 X a_3$
$X \rightarrow b_1 X_f b_3$
$X_f \rightarrow a_2$

This implies that at adjoining instant we can convert TAG to CFG. However, the converted rewriting rules are dynamically changed in terms of the given situation at each adjoining instant. If we adjoin A_1

3) We can easily check out whether the adjoining constraints are satisfied or not in the course of derivation. We can simplify the derived sentential form as follows:

$$X \overset{*}{\Rightarrow} \ldots\ S^{Y_1}_{\theta_1} : S^{Y_2}_{\theta_2} : \ldots : S^{Y_i}_{\theta_i} : (\alpha)\ \ldots$$

can be $X \overset{*}{\Rightarrow} \ldots\ S^{(Y_1, Y_2 \ldots Y_i)}_{\theta_i} : (\alpha)\ \ldots$

In this case(Y1,Y2 ... Yi) is the ordered sequence.

4) TAG is context-sensitive, but mildly so. See Joshi[1987]

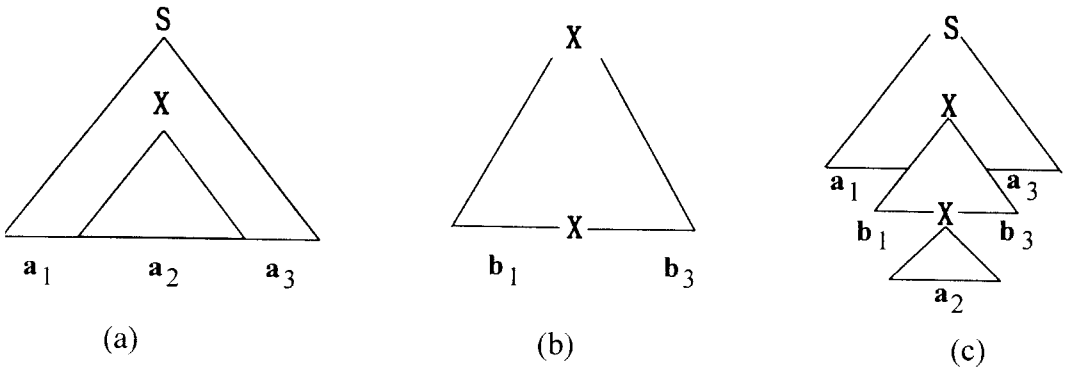

Fig. 3-1 Example of TAG.

with the derived elementary tree in Fig. 3-1(c) once more, then at this moment the rewriting rules will be as follows:

(3-2) $S \rightarrow a_1 X a_3$
$X \rightarrow b_1 X_f b_3$
$X_f \rightarrow b_1 a_2 b_3$

Through the comparison of TAG and CFG, we can realize that at the adjoining instant TAG can be converted to the identical CFG if we know the exact subtree for foot node which is dynamically changed during adjoining.

Theorem 1 (Context-Freeness of TAG) *At the adjoining instant, a TAG G is represented by the equivalent CFG G′.*

We will not give the proof of this theorem. Since the only terminal string is generated by each adjoining, it is simple to construct the equivalent CFG at the adjoining instant. By the definition of adjoining of TAG, it is obvious that the TAG G in Fig. 3-1 is equivalent to the CFG $G' = (\{ S, X, X_f\}, \Sigma, S, P)$ that has the production rules in (3-1), where $a_i (1 \leq i \leq 3)$, b_1 and $b_3 \in \Sigma^*$.

We can easily verify that the derived language of TAG and CFG are equivalent at the adjoining instant. This fact implies that TAG is context sensitive in the whole sense and at the adjoining instant, however, it operates as context-free grammars. Now, let us consider the formal properties of context-free

rules organized at the adjoining moment.

Theorem 2 (LR property of CFG for TAG) *A CFG G = (N,Σ,S,P) that the production rules in P are as followings is LR(2)*

$S \rightarrow a\,X\,b$

$X \rightarrow c_1\,X_f\,d_1 \mid c_2\,X_f\,d_2 \mid \ldots \mid c_n\,X_f\,d_n$

$X_f \rightarrow e$

where a,b,c_i,d_i $(1 \leq i \leq n)$ and $e \in \Sigma^*$, and $S, X, X_f \in N$

proof: Every pair of rightmost derivation of G will be given

$S \rightarrow a\,X\,b \rightarrow a\,c_i\,X_f\,d_i\,b \rightarrow a\,c_i\,e\,d_i\,b$

$S \rightarrow a\,X\,b \rightarrow a\,c_j\,X_f\,d_j\,b \rightarrow a\,c_j\,e\,d_j\,b$

It is obvious that $a\,c_i\,X_f\,d_i\,b : (|a\,c_i\,X_f| + 2) = a\,c_j\,X_f\,d_j\,b : (|a\,c_j\,X_f| + 2)$ or $a\,c_i\,e\,d_i\,b : (|a\,c_i\,e| + 2) = a\,c_j\,e\,d_j\,b$ imply that $a\,c_i = a\,c_j$. If G is not redundant, no same complete sentential forms are derived.

By the Theorem 2, the two lookahead symbols are required when we try to parse language L(G) in CFG G with LR parsing methods. However, since the generated sentence from G is simple and obvious, the parsing of L(G) is straightforward and lookahead symbols are usually not necessary except three cases. Let us consider a typical CFG G1 given as follows to consider the formal properties of CFG for TAG at the adjoining instant.

(3-3) $S \rightarrow a\,X\,b$

$X \rightarrow c_1\,X_f\,d_1 \mid c_2\,X_f \mid X_f\,d \mid X_f$

$X_f \rightarrow e$

We will find an inadequate state at I_{11}. In addition to this, the same problems occurs in case of $c_1=c_2$, and $e=c_1$ or c_2. Since terminals for state transition are generally not disjoint from FOLLOW set, for example, $d_3 \cap FOLLOW(X) \neq \phi$ at I_{11}, this local ambiguity cannot be solved by examining one lookahead symbol. The lookahead information, however, are not necessary for each state except these three cases. We can parse G_1 with LR(0) items provided that the inadequate states is resolved.

We assume that the subtree for foot node at any adjoining instant can be calculated exactly, which is dynamically changed during state transition. To calculate the subtree, let us consider how the subtree for foot node is decided by adjoining.

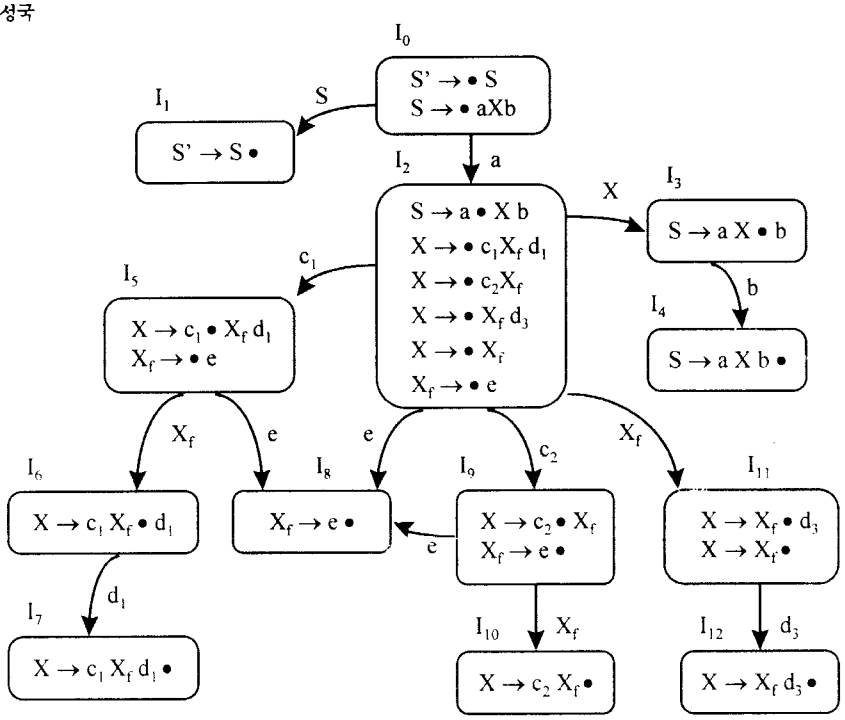

Fig. 3-2. Set of LR(0) items for CFG

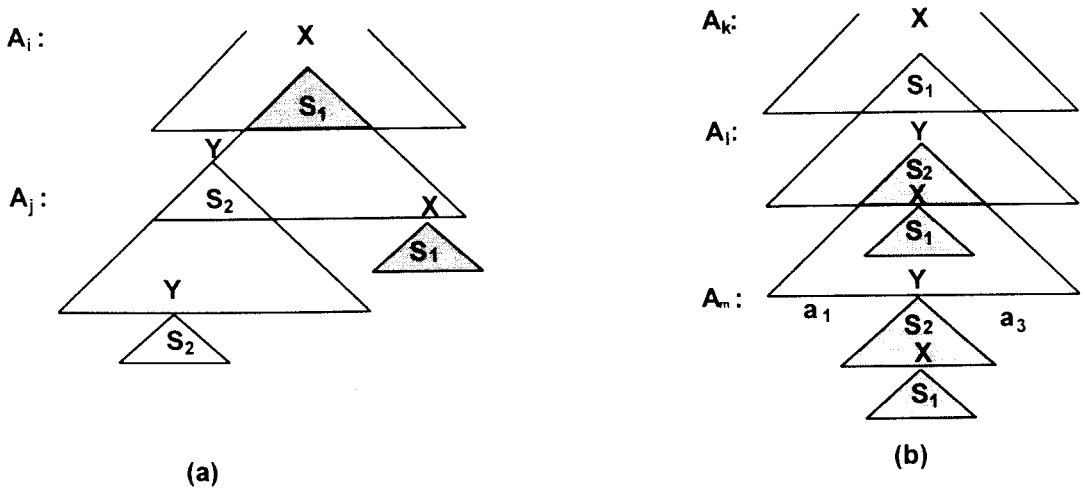

Fig. 3-3. Propagation of subtree for foot node

In Fig. 3-3 which show some typical adjoining, when A_j adjoins on A_i, A_i should pass the subtree for foot node of A_j. In this case, since the subtree S_2 does not contain foot node of A_i, the foot node which is nonterminal without subtree in adjoining rules is simply substituted with the passed subtree and need not to be concerned of S_1. That is, all information for foot node of tree X come from tree that are adjoined with tree X. Let the foot node of A_j be A_{jf}. Then a now context free rule $A_{jf} \rightarrow S_2$ will be applied to decide the subtree. On the other hand, when A_m is adjoined on A_i the foot node A_{mf} of A_m is expanded with context-free rule $A_{mf} \rightarrow a_1 A_{if} a_3$ where A_{if} is foot node of A_i. Note that subtree S_1 should be passed to A_{mf} in the form of contest-free rule $A_{if} \rightarrow S_1$. This process for building up subtree for foot node can be generalized with under context-free system.

Theorem 3 (Context-freeness of subtree formation) *At the adjoining instant, the subtree for foot node Y_f of adjoining tree Y can be determinded with*

$Y_f \rightarrow \alpha$

where α is in the adjoined tree X which is

$X \rightarrow \ldots \phi\theta$: *(α)...and* $Y \in \theta$.

The context-freeness of subtree formation at each adjoining instant is obvious in terms of the definition of adjoining. Theorem 3 can be applied for the whole grammar, not only for adjoining instant, by maintaining related subtree rules in the course of subsequent adjoining.

Consequencely, we can realize that two context free systems are composited in one TAG: one for adjoining of the initial/auxiliary trees and the other for subtree for foot node. Note that a tree play a different role when it receives different subtrees for foot node by left-recursion at the adjoining instant.

3.2 Parsing Table Construction

For parsing TAG by using LR parsing, we define the set of items used in LR parsers. We can obtain *closure(I)*, a set of items constructed from *I*, by means of the following three rules, where *I* is a set of items for TAG G. To compute the subtree for foot node, we should derive the context-free rules for foot node at adjoining states.

1. Every item in I is in *closure(I)*.
2. If $X \rightarrow \alpha \cdot Y_{(Y_1 Y_2 \ldots\ldots Y_n)}:(\beta)\gamma$ is in *closure(I)* and $Y_i \rightarrow Y_{(z_1 z_2 \ldots\ldots z_k)}:(\delta)$ is in adjoining rules P, then $Y_i \rightarrow \cdot Y_{(z_1 z_2 \ldots\ldots z_k)}:(\delta')$ are added to closure*(I)*, where δ' is δ that S_ϕ without subtree in δ is substituted with Y_{if}. And the rules for foot node $Y_{if} / X \rightarrow \beta$ generated at this state.
3. If $X \rightarrow \alpha \cdot Y_\phi:(\beta)\gamma$ is in I, then $X \rightarrow \alpha \cdot \beta\gamma$ is added to closure*(I)*.

The meaning of the dot notation of item sets is identical to that of LR parser. We can define the transition between I by grammar symbols $\beta \in \{N', \Sigma\}$ with the usual sense of LR parser. Note that the set of nonterminals is different from that of adjoining rules, because at adjoining instant context-free rules to determine the subtree for node require additional nonterminals. However, the state transition by *goto(I,β)* is also defined identically to LR parser, i.e, the closure of the set of all items $X \rightarrow \alpha \cdot \beta\gamma$ such that $X \rightarrow \alpha \beta \cdot \gamma$ is in I. In contrast to the general LR grammars, function goto accompanys the inheritance of subtree, that is, context-free rules for subtree at state $X \rightarrow \alpha \cdot \beta\gamma$ is transfered to state $X \rightarrow \alpha\beta \cdot \gamma$. The transfered rules may be used to calculate subtree for foot node at that state.

Thus, since each state is to have all information to realize adjoining operation, a TAG G can be parsed with LR(0) items by means of making a parsing algorithm solved the multiply-defined entries for inadequate states. The algorithm to construct parsing table is given in PTABLE.

Algorithm : PTABLE
/* constructing an LR parsing table */
input : adjoining rules for TAG G
output : parsing table with actions

1. Construct $C=I_0, I_1, \ldots\ldots, I_n$ the collection of sets of items for G.
2. /* State I is constructed from closure set I_i. The parsing actions for state i determined as follows; */

2.1 If $A \rightarrow \alpha \cdot \beta\gamma$ is in I_i and *goto* $(I_i, \beta)=I_j$, then set action $[I,\beta]$ to *"shift j"*, where β must be terminal.

2.2 If $A \rightarrow \alpha \cdot \beta\gamma$ is in I_i, then set *action[I]* to *"reduce* $A \rightarrow \alpha$*"*, i.e. , state

5) This entry will be single unless TAG is left-recursive. The state with single

I become reduction state.[5]

2.3 If [I′→I ·] in I_i, set action *[I,$]* to *"accept"*, where *I* is the initial tree.

3. /* Transition by nonterminals */
For all nonterminals A, if *goto* $(I_i,A)=I_j$, then set *action [I,A]=goto j.*

Certainly, all entries not filled with action mean that errors happen. Let us consider some examples to show how the parsing table is constructed by algorithm PTABLE.

Example 1

The grammar for the language related to the Swiss-German example which shows the cross-serial dependency explained under context-sensitive formalism is represented by the adjoining rules as follows:

$I \rightarrow S_{(A1,A2,\phi)}$: (e)

$A_1 \rightarrow S_\phi$: (a $S_{(A1,A2,\phi)}$: (S_ϕ b))

$A_2 \rightarrow S_\phi$: (b $S_{(A1,A2,\phi)}$: (S_ϕ a))

The state transition diagram for these adjoining rules are derived as shown in Fig. 3-4 and the parsing table is given in Table 3-2.

Note that in transition diagram shown in Fig.3-4, state I_{11} and I_{12} have several different reductions and they become inadequate states. Though the conflicts happen at inadequate states, they will be selected deterministically by their own FOLLOW set.

3.3 LR Parsing Algorithm for TAG

The main problem of parsing TAG is how the inadequate state resolution is solved. We can resolve the inadequate states in terms if FOLLOW set as mentioned above. Now a LR bottom-up parsing algorithm for TAG is given in BOTTOMUP.

Algorithm : BOTTOMUP
/* LR parsing algorithm for TAG */
input : input string to be parsed
output : parsing tree or sentential string to construct the parsing tree in successful
completions otherwise error indication
data used : parsing table by PTABLE

---

reduction is education state. In this case it is not necessary to compute FOLLOW set since errors can be checked during state transition. The states with multiple entries, however, is treated as inadequate states.

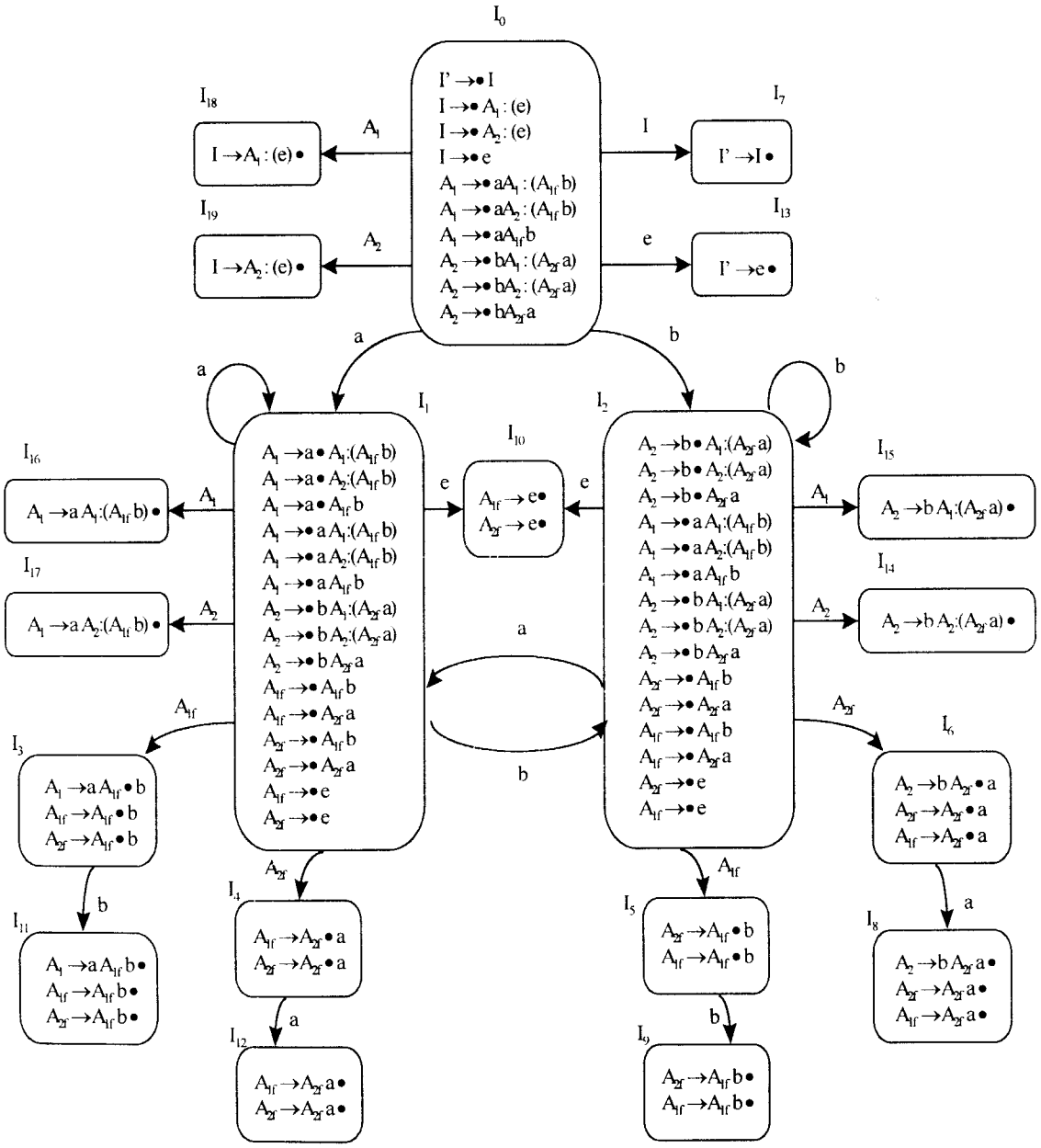

Fig.3-4. State Transition Diagram for Example 1.

Table 3-1. Parsing table for the Swiss-German grammar

state	a	b	e	I	A_1	A_2	A_{1f}	A_{2f}	$
I_0	sh 1	sh 2	sh 13	g 7	g 18	g 19			
I_1	sh 1	sh 2	sh 10		g 16	g 17	g 3	g 4	
I_2	sh 1	sh 2	sh 10		g 15	g 14	g 6	g 5	
I_3		sh 11							
I_4	sh 12								
I_5		sh9							
I_6	sh 8								
I_7									accept
I_8	$A_2f \rightarrow A_1fa$	$A_1f \rightarrow A_2fa$							$A2 \rightarrow bA_2f$
I_9	$A_2f \rightarrow A_1fb$	$A_1f \rightarrow A_1fb$							
I_{10}	$A_2f \rightarrow e$	$A_1f \rightarrow e$							
I_{11}	$A_2f \rightarrow A_1fb$	$A_1f \rightarrow A_1fb$							$A_1 \rightarrow aA_1fb$
I_{12}	$A_2f \rightarrow A_1fa$	$A_1f \rightarrow A_2fa$							
I_{13}									$I \rightarrow e$
I_{14}									$A_2 \rightarrow bA_2$
I_{15}									$A_2 \rightarrow bA_1$
I_{16}									$A_1 \rightarrow aA_1$
I_{17}									$A_1 \rightarrow aA_2$
I_{18}									$I \rightarrow A_1$
I_{19}									$I \rightarrow A_2$

```
begin
 set index to point the first terminal
 of input string and set stack to I0.
 while (t) do
  begin
   let α be the top stack symbol and
   a the symbol pointed by index
   /* reduction state */
   if α is the state with A →β • then
    begin
     pop 2*|β| symbols off the
     stack
     let α' be the state on the top of
     stack
     push A and state in α',A ]
     output A: (β)
    end
 /* shift state */
  else if [α,a] = shift α', then
   begin
    pushα and α', on the top of
    stack
    advance index to |α|
   end
  /* acceptance state */
  else if [α,a] = accept then
  return(accept)
  else return(error)
 end
end
```

As we see in algorithm BOTTOMUP, parsing process is almost identical to the conventional LR parser. During the parsing process, we will obtain the sentential form that contains all adjoining information as the output of parser. We can construct parsing tree by adjoining trees in accordance with these adjoining information or obtain directly parsing tree through the parsing process.

Example 2.

On input string *aabaebbab$* of the Swiss-German example shown in Example 1, the trace of parser for this input is given in Table 3-2.

From the parser, we obtain sentential string as follows :

I : (A_1:(a A_1:(a A_2:(b A_1:(a A_{1f}:(A_{2f}:(A_{1f}:(A_{1f}:(e)b)b)a)b)))))

The interpretation of this expression is definite for construction parsing tree.

4.Extension of LR Parsing Algorithm

It is known that substitution is useful for obtaining the appropriate structural descriptions of natural language[Abeille 1988]. And the feature structures are introduced

Table 3-2. Trace of Parser for input *aabaebbab$*

stack	input	action
I_0	aabaebbab$	shift 1
I_0aI_1	abaebbab$	shift 1
$I_0aI_1aI_1$	baebbab$	shift 2
$I_0aI_1aI_1bI_2$	aebbab$	shift 1
$I_0aI_1aI_1bI_2aI_1$	ebbab$	shift 10
$I_0aI_1aI_1bI_2aI_1eI_{10}$	bbab$	reduce $A_{1f}\rightarrow e$
$I_0aI_1aI_1bI_2aI_1A_{1f}I_3$	bbab$	shift 11
$I_0aI_1aI_1bI_2aI_1A_{1f}I_3bI_{11}$	bab$	reduce $A_{1f}\rightarrow A_{1f}b$
$I_0aI_1aI_1bI_2aI_1A_{1f}I_3$	bab$	shift 11
$I_0aI_1aI_1bI_2aI_1A_{1f}I_3bI_{11}$	ab$	reduce $A_{2f}\rightarrow A_{1f}b$
$I_0aI_1aI_1bI_2aI_1A_{2f}I_4$	ab$	shift 12
$I_0aI_1aI_1bI_2aI_1A_{2f}I_4aI_{12}$	b$	reduce $A_{1f}\rightarrow A_{2f}a$
$I_0aI_1aI_1bI_2aI_1A_{1f}I_3$	b$	shift 11
$I_0aI_1aI_1bI_2aI_1A_{1f}I_3bI_{11}$	$	reduce $A_1\rightarrow aA_{1f}b$
$I_0aI_1aI_1bI_2A_1I_{15}$	$	reduce $A_2\rightarrow bA_1$
$I_0aI_1aI_1A_2I_{17}$	$	reduce $A_2\rightarrow aA_2$
$I_0aI_1A_1I_{16}$	$	reduce $A_1\rightarrow aA_1$
$I_0A_1I_{18}$	$	reduce $I\rightarrow aA_1$
$I_0\ II_7$	$	accept

into TAG to describe linguistic objects. In this section we will show how adjoining rules system and LR parser are extended to capture these two extension of TAG.

4.1 Substitution in Adjoining Rules System

Sometimes it is convenient to use substitution to describe the linguistic structures although substitution can be realized by

adjoining operation[Abeille 1988]. It should be emphasized that the introduction of substitution in TAG does not change the original generative capacity of TAG since substitution is not powerful than adjunction. The substitution is the basic operation used in CFG. Since are similar to rewriting rules and LR parsers work on CFG, it is easy to introduce substitution to ARS and a proposed parser. The definition of substitution in TAG is given as follows[Schabes 1988] :

Definition 4.1 (Substitution in TAG) *A tree with substitution node on the frontier is replaced with the initial tree that can place on this node. The substitution is always mandatory.*

The substitution is illustrated in Fig. 4-1. The tree with substitution node is very similar to auxiliary tree. In ARS it will be represented as the form of

$$S \rightarrow X : (a_1\ S \downarrow_{(I_1,\ \phi)}\ a_3)$$

By the definition of substitution, though no adjoining can take place on substitution

node, we still be able to put some adjoining constraints on the root node of substituting tree in ARS[6] . It is also possible to impose adjoining

6) A downarrow is used to as the marker to represent substitution node[Schabes 1988]

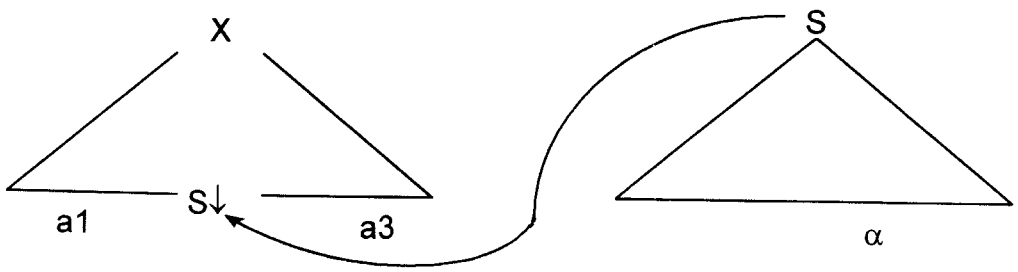

Fig. 4-1. Substitution in TAG.

constraints on any internal nodes of tree with substitution node. When substitution is occurred, tree will be expanded as adjunction happens.

$$S \rightarrow X : (a_1\, S\downarrow_{(I_1,\phi)}\, a_3)$$
$$\rightarrow X : (a_1\, S\downarrow_{(I_1,\phi)} a_3 : S'^{\uparrow}_{\phi}(\alpha)\, a_3)$$

We need not to restrict tree that can substitute or operation conditions even if the last nonterminal holds all properties after operations.

The closure sets of LR(0) items in case of substitution node is obvious. LR(0) item set in Fig. 4-1, for example, is given as Fig. 4-2.

From Fig. 4-2, we will realize that no additional functions or modification is necessary to parse TAG with substitution node with BOTTOMUP. It is natural that LR parsing algorithm handle the substitution. Since substitution node does not have its subtree for foot node, context-free rules for foot node are not generated at substitution state.

4.2 Extension to Feature Structure

It is shown that TAG can be embedded in a feature structure based framedwork[Vijay- Schanker 1988]. Every adjoining node has two

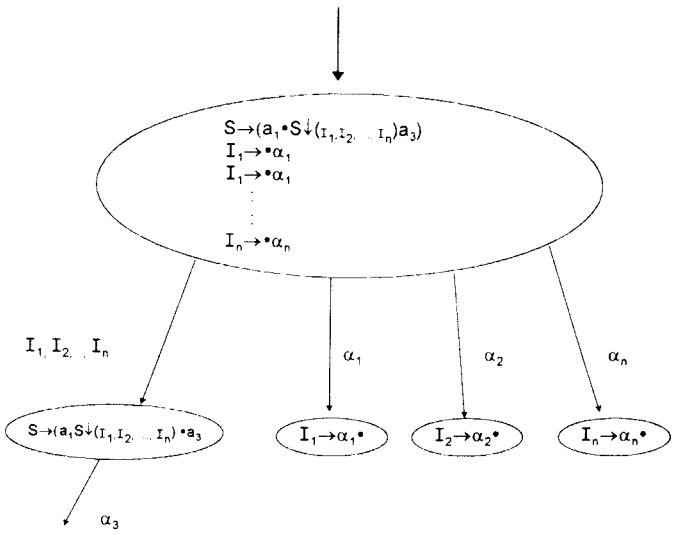

Fig. 4-2 LR(0) items at substitution node

feature structures: a top and a bottom feature represented *t* and *b*, respectively. When derivation is completed, the top and the bottom node is unified as Fig. 4-3.

under node X.

Every nonterminal in adjoining rules should have its subtree. Since the foot node of auxiliary tree had

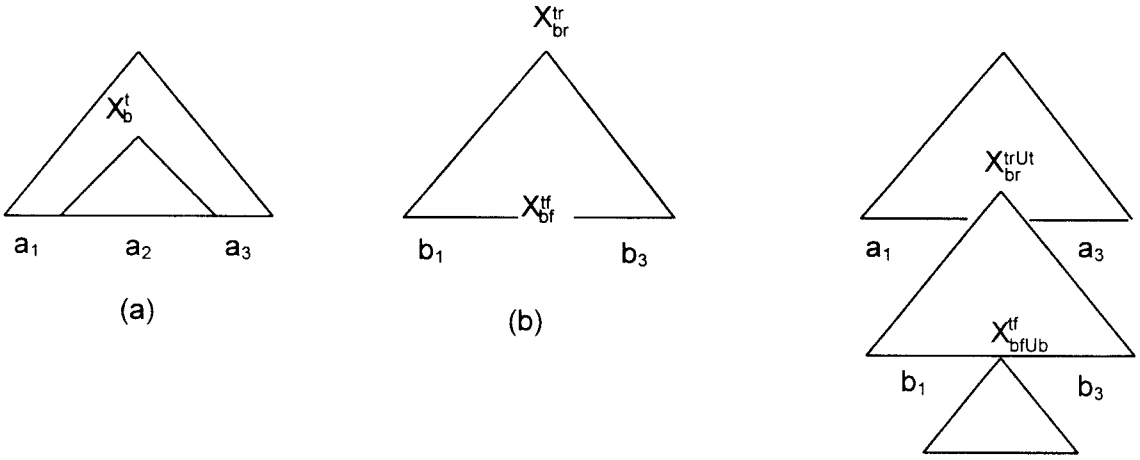

Fig. 4-3. Feature structure of TAG

From the meaning feature structures we can write the adjoining expression for *I* in Fig. 4 -3.

$$I \rightarrow (a_1\ X_t:\ (a_2)_b\ a_3)$$

In this expression the feature *t* is related with top node and *b* is specified the feature of subtree

no subtree at the beginning, we assume it has a fictitious subtree Φ[7] . The auxiliary tree A in Fig. 4-3 is

7) The subtree, of course, must be substituted with the real subtree during derivation. Recall than foot node is unstable.

represented by

$$A \rightarrow X_{tr} : (b_1 X_{tf} : \Phi_{bf} b_3)_{br}$$

The unification of feature structures when adjunction takes place is decribed with ease in ARS as follow:

$$\begin{aligned} I &\rightarrow (a_1 X_t : (a_2)_b\ a_3) \\ &\rightarrow (a_1 X_t : X_{tr} : (b_1 X_{tf} : \Phi_{bf} b_3)_{br} : (a_2)_b\ a_3) \\ &= (a_1 X_{t \cup tr} : (b_1 X_{tf} : (a_2)_{b \cup bf} b_3)_{br}\ a_3) \end{aligned}$$

The definition of adjoining in ARS, as it is, is applicable to capture the feature structure. The nonterminals in feature structures in TAG is represented with the pair of top and bottom feature. And the adjunction is applied between these pairs.

We can obtain the closure set of LR(0) items using unification. Assume that $X \rightarrow \alpha \bullet Y_t : (\beta_b)$ is in *closure(I)* and $Y \rightarrow Z_{tr} : (\gamma W_{tf} : \Phi_{bf} \delta)_{br}$ is in adjoining rules. Then *t* and *tr* are compatible as are *b* and *bf*, $Y \rightarrow \cdot\ Z_{tr \cup t} : (\delta W_{tf} : \Phi_{bf \cup b} \delta)$ added to closure set. The unification to derive to the closure set will easily accomplished in ARS. However, this requires further research.

5. Conculsion

We present a LR bottom-up parsing algorithm for TAG. We introduce the tree adjoining rules system to formalize the grammatical properties. The adjoining rules system makes it possible to handle TAG as the traditional rewriting rules systems. We consider the context-free behaviors of TAG at adjoining instant. We realize that two context-free mechanisms are composite in TAG: one for adjoining trees and the other for foot nodes. This fact is evident by the definition of adjoining in TAG. The context-free rules for adjoining trees act as LR(2) while the context-free rules for foot nodes the previous adjoining states. We merge these two mechanisms by maintaining the dynamic change of foot nodes so that we extend context-free behaviors to the whole grammar. A LR bottom-up parsing algorithm that uses LR(0) items is presented with some typical examples. The algorithm is quite simple and works fine. We also show how ARS can be extended to deal with substitution and feature structures.

When parsing TAG it may be

possible to use the parser generator such as YACC. The representation of tree by adjoining rules and the concepts about context-freeness of TAG will give some intuition to do this. We know that LL are generated at each adjoining state and require their associated rules that are generated at top-down parsing is also possible in case of no-recursion in TAG.

References

(1)Abeille, Anne(1988),"A Lexicalized Tree Adjoining Grammar for French: the General Framework", TR Dept. of Computer and Information Science, Univ. of Pennsylvania

(2)Aho, A.V. and S.C Johnson(1974), "LR parsing", Computing Surveys of the ACM, Vol.6, No.2

(3) Aho, A.V., R. Sathi and Ullman, J.D(1986), Compiler: Principles, Techniques, and Tools, Reading; Mass-Addison-Wesley

(4) Joshi A.K, et al(1975),"Tree Adjunct Grammars", J. Compt. Syst. Sci.

(5) Joshi A.K.(1987), An Introduction to Tree Adjoining Grammars. In Mathematics of Language edited by A.Manaster-Ramer, John Benjamins Publ. Co., PA

(6) Kroch, A. and Joshi A.K.(1985), "An Early-type parser for Tree Adjoining Grammars:", TR MS-CIS-85-18, Dept. of Computer and Information Science, Univ. of Pennsylvania

(7) Schabes, Y. and Joshi A.K. (1988), "An Early-type parser for Tree Adjoining Grammars", TR MS-CIS-88-36, Dept. of Computer and Information Science, Univ. of Pennsylvania

(8) Vijay-Schanker, K.(1986), "A study of Tree Adjoining Grammar", Ph. D. thesis, Dept. of Computer and Information Science, Univ. of Pensylvania

(9) Vijay-Schanker, K. and Joshi A.K.(1988), "Feature Structure based Tree Adjoining Grammars", In Proc. of 12th COLING 88

(10) Tomita, M.(1991), Issues in Parsing Technology, Kluwer Academic Pub

(11) Schehes, Y., and Vijay-Schamker, K. (1990), "Deterministic left to right parsing of Tree Adjoining Languages", Proc of 28th Meeting of ACL.

(12) Kong-Ju Lee and Kil-Chang, Kim(1995), "Tree Transformation

Rules for Korean Lexicalized Multi-Component TAG Parser," Proc. Of 1st Intelligent Technologies. pp.100-105.
(13)Sung-Kook Han, "Word Order and Its variations in Korea: A TAG's Approach", Language Research Vol. 27, No. 1, pp. 19-40.