SOME COMPUTATIONAL PROPERTIES OF TREE ADJOINING GRAMMARS*

K. Vijay-Shankar and Aravind K. Joshi

Department of Computer and Information Science
Room 268 Moore School/D2
University of Pennsylvania
Philadelphia, PA 19104

ABSTRACT

Tree Adjoining Grammar (TAG) is a formalism for natural language grammars. Some of the basic notions of TAG's were introduced in [Joshi,Levy, and Takahashi 1975] and by [Joshi, 1983]. A detailed investigation of the linguistic relevance of TAG's has been carried out in [Kroch and Joshi,1985]. In this paper, we will describe some new results for TAG's, especially in the following areas: (1) parsing complexity of TAG's, (2) some closure results for TAG's, and (3) the relationship to Head grammars.

1. INTRODUCTION

Investigation of constrained grammatical systems from the point of view of their linguistic adequacy and their computational tractability has been a major concern of computational linguists for the last several years. Generalized Phrase Structure grammars (GPSG), Lexical Functional grammars (LFG), Phrase Linking grammars (PLG), and Tree Adjoining grammars (TAG) are some key examples of grammatical systems that have been and still continue to be investigated along these lines.

Some of the basic notions of TAG's were introduced in [Joshi, Levy, and Takahashi,1975] and [Joshi,1983]. Some preliminary investigations of the linguistic relevance and some computational properties were also carried out in [Joshi,1983]. More recently, a detailed investigation of the linguistic relevance of TAG's were carried out by [Kroch and Joshi,1985].

In this paper, we will describe some new results for TAG's, especially in the following areas: (1) parsing complexity of TAG's, (2) some closure results for TAG's, and (3) the relationship to Head grammars. These topics will be covered in Sections 3, 4, and 5 respectively. In section 2, we will give an introduction to TAG's. In section 6, we will state some properties not discussed here. A detailed exposition of these results is given in [Vijay-Shankar and Joshi,1985].

2. TREE ADJOINING GRAMMARS--TAG's

We now introduce tree adjoining grammars (TAG's). TAG's are more powerful than CFG's, both weakly and strongly.[1] TAG's were first introduced in [Joshi, Levy, and Takahashi,1975] and [Joshi,1983]. We include their description in this section to make the paper self-contained.

We can define a tree adjoining grammar as follows. A tree adjoining grammar G is a pair (I,A) where I is a set of initial trees, and A is a set of auxiliary trees.

A tree α is an initial tree if it is of the form

```
α =

         S
        / \
       /   \              w ∈ Σ*
      /     \
     /       \
    ---------
         w
```

That is, the root node of α is labelled S and the frontier nodes are all terminal symbols. The internal nodes are all non-terminals. A tree β is an auxiliary tree if it is of the form

```
β =         X
           / \
          /   \
         /     \      w1w2 ∈ Σ*
        /       \
        -----X-----
      w1         w2
```

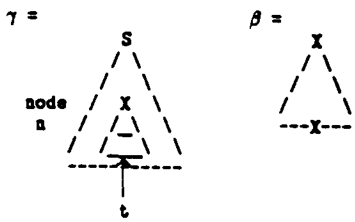
That is, the root node of β is labelled with a non-terminal X and the frontier nodes are all labelled with terminals symbols except one which is labelled X. The node labelled by X on the frontier will be called the foot node of β. The frontiers of initial trees belong to Σ^*, whereas the frontiers of the auxiliary trees belong to $\Sigma^* N \Sigma^+ \cup \Sigma^+ N \Sigma^*$.

We will now define a composition operation called adjoining, (or adjunction) which composes an auxiliary tree β with a tree γ. Let γ be a tree with a node n labelled X and let β be an auxiliary tree with the root labelled with the same symbol X. (Note that β must have, by definition, a node (and only one) labelled X on the frontier.)

[1]Grammars G1 and G2 are weakly equivalent if the string language of G1, L(G1) = the string language of G2, L(G2). G1 and G2 are strongly equivalent if they are weakly equivalent and for each w in L(G1) = L(G2), both G1 and G2 assign the same structural description to w. A grammar G is weakly adequate for a (string) language L, if L(G) = L. G is strongly adequate for L if L(G) = L and for each w in L, G assigns an "appropriate" structural description to w. The notion of strong adequacy is undoubtedly not precise because it depends on the notion of appropriate structural descriptions

*This work was partially supported by NSF Grants MCS-8219116-CER, MCS-82-07294. We want to thank Carl Pollard, Kelly Roach, David Searl, and David Weir. We have benefited enormously by valuable discussions with them.

Adjoining can now be defined as follows. If β is adjoined to γ at the node n then the resulting tree γ_1' is as shown in Fig. 2.1 below.



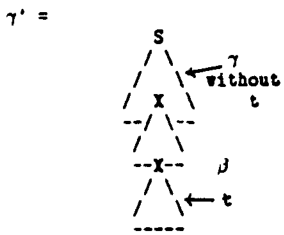


Figure 2.1

The tree t dominated by X in γ is excised, β is inserted at the node n in γ and the tree t is attached to the foot node (labelled X) of β, i.e., β is inserted or adjoined to the node n in γ pushing t downwards. Note that adjoining is not a substitution operation.

We will now define

T(G): The set of all trees derived in G starting from initial trees in I. This set will be called the tree set of G.

L(G): The set of all terminal strings which appear in the frontier of the trees in T(G). This set will be called the string language (or language) of G. If L is the string language of a TAG G then we say that L is a Tree-Adjoining Language (TAL). The relationship between TAG's , context-free grammars, and the corresponding string languages can be summarized as follows ([Joshi, Levy, and Takahashi, 1975], [Joshi, 1983]).

Theorem 2.1: For every context-free grammar, G', there is an equivalent TAG, G, both weakly and strongly.

Theorem 2.2: For every TAG, G, we have the following situations:

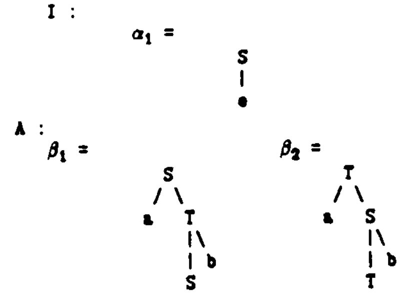
a. L(G) is context-free and there is a context-free grammar G' that is strongly (and therefore weakly) equivalent to G.

b. L(G) is context-free and there is no context-free grammar G' that is equivalent to G. Of course, there must be a context-free grammar that is weakly equivalent to G.

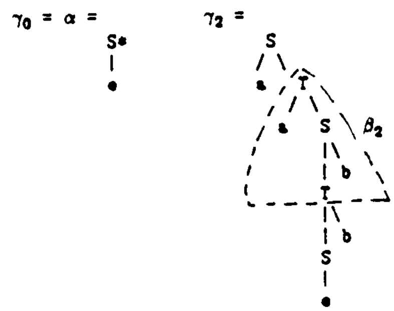
c. L(G) is strictly context-sensitive. Obviously in this case, there is no context-free grammar that is weakly equivalent to G.

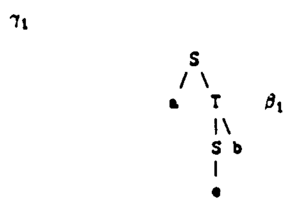
Parts (a) and (c) of Theorem 2.2 appear in ([Joshi, Levy, and Takahashi, 1975]). Part (b) is implicit in that paper, but it is important to state it explicitly as we have done here because of its linguistic significance. Example 2.1 illustrates part (a). We will now illustrate parts (b) and (c).

Example 2.2: Let G = (I,A) where



Let us look at some derivations in G.





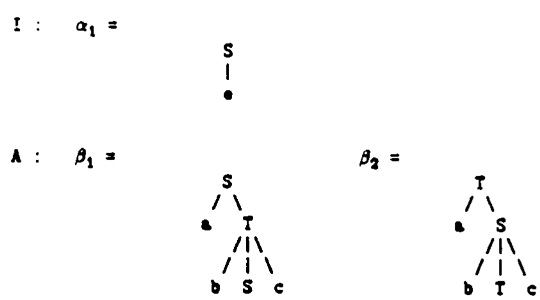
$\gamma_1 = \gamma_0$ with β_1 adjoined at S as indicated in γ_0. $\gamma_2 = \gamma_1$ with β_2 adjoined at T as indicated in γ_2.

Clearly, L(G), the string language of G is

$$L = \{a^n e b^n / n \geq o\}$$

which is a context-free language. Thus, there must exist a context-free grammar, G', which is at least weakly equivalent to G. It can be shown however that there is no context-free grammar G' which is strongly equivalent to G, i.e., T(G) = T(G'). This follows from the fact that the set T(G) (the tree set of G) is non-recognizable, i.e., there is no finite state bottom-up tree automaton that can recognize precisely T(G). Thus a TAG may generate a context-free language, yet assign structural descriptions to the strings that cannot be assigned by any context-free grammar.

Example 2.3: Let G = (I,A) where



The precise definition of L(G) is as follows:

$L(G) = L_1 = \{w\, e\, c^n \,/\, n \geq o$, w is a string of a's and b's such that

(1) the number of a's = the number of b's = n, and

(2) for any initial substring of w, the number of a's $\geq$ the number of b's. $\}$

L_1 is a strictly context-sensitive language (i.e., a context-sensitive language that is not context-free). This can be shown as follows. Intersecting L with the regular language $a^* b^* e\, c^*$ results in the language

$$L_2 = \{ a^n b^n e\, c^n \,/\, n \geq o \} = L_1 \cap a^* b^* e\, c^*$$

L_2 is well-known strictly context-sensitive language. The result of intersecting a context-free language with a regular language is always a context-free language; hence, L_1 is not a context-free language. It is thus a strictly context-sensitive language. Example 2.3 thus illustrates part (c) of Theorem 2.2.

TAG's have more power than CFG's. However, the extra power is quite limited. The language L_1 has equal number of a's, b's and c's; however, the a's and b's are mixed in a certain way. The language L_2 is similar to L_1, except that a's come before all b's. TAG's as defined so far are not powerful enough to generate L_2. This can be seen as follows. Clearly, for any TAG for L_2, each initial tree must contain equal number of a's, b's and c's (including zero), and each auxiliary tree must also contain equal number of a's, b's and c's. Further in each case the a's must precede the b's. Then it is easy to see from the grammar of Example 2.3, that it will not be possible to avoid getting the a's and b's mixed. However, L_2 can be generated by a TAG with local constraints (see Section 2.1) The so-called copy language.

$$L = \{ w\, e\, w \,/\, w \in \{a,b\}^* \}$$

also cannot be generated by a TAG, however, again, with local constraints. It is thus clear that TAG's can generate more than context-free languages. It can be shown that TAG's cannot generate all context-sensitive languages [Joshi ,1984].

Although TAG's are more powerful than CFG's, this extra power is highly constrained and apparently it is just the right kind for characterizing certain structural descriptions. TAG's share almost all the formal properties of CFG's (more precisely, the corresponding classes of languages), as we shall see in section 4 of this paper and [Vijay-Shankar and Joshi,1985]. In addition,the string languages of TAG's can also be parsed in polynomial time, in particular in $O(n^6)$. The parsing algorithm is described in detail in section 3.

2.1. TAG's with Local Constraints on Adjoining

The adjoining operation as defined in Section 2.1 is "context-free". An auxiliary tree, say,

β =

```
       X
      / \
     /   \
    /     \
   ---X---
```

is adjoinable to a tree t at a node, say, n, if the label of that node is X. Adjoining does not depend on the context (tree context) around the node n. In this sense, adjoining is context-free.

In [Joshi ,1983], local constraints on adjoining similar to those investigated by [Joshi and Levy ,1977] were considered.These are a generalization of the context-sensitive constraints studied by [Peters and Ritchie ,1969]. It was soon recognized, however, that the full power of these constraints was never fully utilized, both in the linguistic context as well as in the "formal languages" of TAG's. The so-called proper analysis contexts and domination contexts (as defined in [Joshi and Levy ,1977]) as used in [Joshi ,1983] always turned out to be such that the context elements were always in a specific elementary tree i.e., they were further localized by being in the same elementary tree. Based on this observation and a suggestion in [Joshi, Levy and Takahashi ,1975], we will describe a new way of introducing local constraints. This approach not only captures the insight stated above, but it is truly in the spirit of TAG's. The earlier approach was not so, although it was certainly adequate for the investigation in [Joshi ,1983]. A precise characterization of that approach still remains an open problem.

G = (I,A) be a TAG with local constraints if for each elementary tree $t \in I \cup A$, and for each node, n, in t, we specify the set β of auxiliary trees that can be adjoined at the node n. Note that if there is no constraint then all auxiliary trees are adjoinable at n (of course, only those whose root has the same label as the label of the node n). Thus, in general, β is a subset of the set of all the auxiliary trees adjoinable at n.

We will adopt the following conventions.

1. Since, by definition, no auxiliary trees are adjoinable to a node labelled by a terminal symbol, no constraint has to be stated for node labelled by a terminal.

2. If there is no constraint, i.e., all auxiliary trees (with the appropriate root label) are adjoinable at a node, say, n, then we will not state this explicitly.

3. If no auxiliary trees are adjoinable at a node n, then we will write the constraint as (ϕ), where ϕ denotes the null set.

4. We will also allow for the possibility that for a node at least one adjoining is obligatory, of course, from the set of all possible auxiliary trees adjoinable at that node.

Hence, a <u>TAG with local constraints</u> is defined as follows. G = (I, A) is a TAG with local constraints if for each node, n, in each tree t, be specify one (and only one) of the following constraints.

1. <u>Selective Adjoining (SA:)</u> Only a specified subset of the set of all auxiliary trees are adjoinable at n. SA is written as (C), where C is a subset of the set of all auxiliary trees adjoinable at n.

   If C equals the set of all auxiliary trees adjoinable at n, then we do not explicitly state this at the node n.

2. <u>Null Adjoining (NA:)</u> No auxiliary tree is adjoinable at the node N. NA will be written as (ϕ).

3. <u>Obligating Adjoining (OA:)</u> At least one (out of all the auxiliary trees adjoinable at n) must be adjoined at n. OA is written as (OA), or as O(C) where C is a subset of the set of all auxiliary trees adjoinable at n.

Example 2.4: Let G = (I,A) be a TAG with local constraints where

I: α =

```
              S    (φ)
             / \
      β1    S   S    (β2)
            |   |
            a   b
```

A: β_1 = β_2 =

```
     S   (β1)           S    (β2)
    / \                / \
   /   \              /   \
  a     S (φ)    (φ)  S     b
```

In α_1 no auxiliary trees can be adjoined to the root node. Only β_1 is adjoinable to the left S node at depth 1 and only β_2 is adjoinable to the right S node at depth 1. In β_1 only β_1 is adjoinable at the root node and no auxiliary trees are adjoinable at the foot node. Similarly for β_2.

We must now modify our definition of adjoining to take care of the local constraints. given a tree γ with a node, say, n, labelled A and given an auxiliary tree, say, β, with the root node labelled A, we define adjoining as follows. β is adjoinable to γ at the node n if $\beta \in \vartheta$, where ϑ is the constraint associated with the node n in γ. The result of adjoining β to γ will be as defined in earlier, except that the constraint C associated with n will be replaced by C', the constraint associated with the root node of β and by C*, the constraint associated with the foot node of β. Thus, given

γ = β =

```
       S                                A (C')
      / \    node n                    / \
     / A (C)                          /   \
    / / \                            /     \
   / /   \ \                        /       \
  / /     \ \                      /         \
  -----------                      -----A-----
                                          (C*)
```

The resultant tree γ' is

γ' =

```
          S
         / \
        /   \
       /     \
      /  A (C')
     /  / \   \
     ---/   \---
       /     \
      /  A (C*)
     /  / \   \
     ---/   \---
       /     \
       -------
```

We also adopt the convention that any derived tree with a node which has an OA constraint associated with it will not be included in the tree set associated with a TAG, G. The string language L of G is then defined as the set of all terminal strings of all trees derived in G (starting with initial trees) which have no OA constraints left-in them.

Example 2.5: Let G = (I,A) be a TAG with local constraints where

I : α =

```
      S
      |
      e
```

A : β =

```
      S  (φ)
     /|
    / |
   a  S
     /|\
    / | \
   b  |  c
      S (φ)
```

There are no constraints in α_1. In β no auxiliary trees are adjoinable at the root node and the foot node and for the center S node there are no constraints.

Starting with α_1 and adjoining β to α_1 at the root node we obtain

γ =

```
      S (φ)
     /|
    / |
   a  S
     /|\
    / | \
   b  |  c
      S (φ)
      |
      e
```

Adjoining β to the center S node (the only node at which adjunction can be made) we have

γ' =

```
          S (φ)
         /|
        / |
       a /S\ (φ)
       / /| \
      / / |  \
     / a  S   \      ←   β
    /    /|\   \
   /    / | \   \
  /    b  |  c   \
  L - - - S (φ)- -J
         /|\
        / | \
       b  |  c
          S (φ)
          |
          e
```

It is easy to see that G generates the string language

$$L = \{ a^n b^n e c^n / n \geq 0\}$$

Other languages such as $L' = \{a^{n^2} | n \geq 1\}$, $L^* = \{a^{n^2} | n \geq 1\}$ also cannot be generated by TAG's. This is because the strings of a TAL grow linearly (for a detailed definite of the property called "contact growth" property, see [Joshi ,1983].

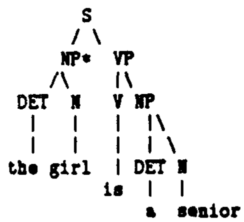
For those familiar with [Joshi, 1983], it is worth pointing out that the SA constraint is only abbreviating, i.e., it does not affect the power of TAG's. The NA and OA constraints however do affect the power of TAG's. This way of looking at local constraints has only greatly simplified their statement, but it has also allowed us to capture the insight that the 'locality' of the constraint is statable in terms of the elementary trees themselves!

2.2. Simple Linguistic Examples

We now give a couple of linguistic examples. Readers may refer to [Kroch and Joshi, 1985] for details.

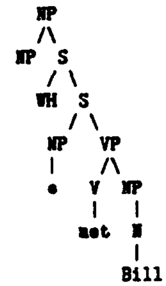
1. Starting with $\gamma_1 = \alpha_1$ which is an initial tree and then adjoining β_1 (with appropriate lexical insertions) at the indicated node in α_1, we obtain γ_2.

$\gamma_1 = \alpha_1 =$

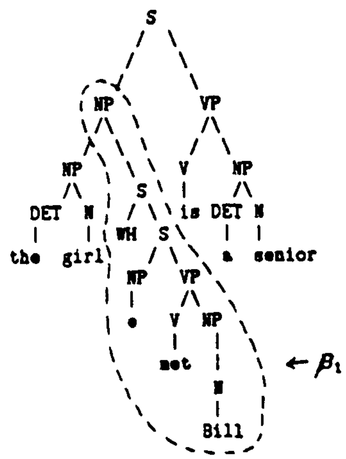


the girl is a senior

$\beta_1 =$



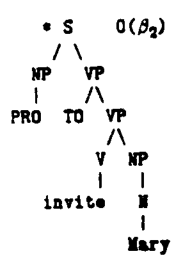
$\gamma_2 =$



The girl who met Bill is a senior

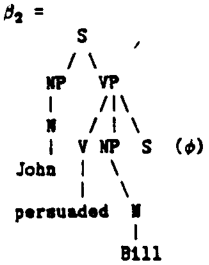
2. Starting with the initial tree $\gamma_1 = \alpha_2$ and adjoining β_2 at the indicated node in α_2 we obtain γ_2.

$\gamma_1 = \alpha_2 =$



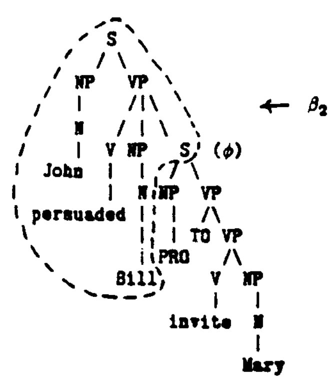
PRO to invite Mary

$\beta_2 =$



John persuaded Bill S

$\gamma_2 =$



John persuaded Bill to invite Mary

Note that the initial tree α_2 is not a matrix sentence. In order for it to become a matrix sentence, it must undergo an adjunction at its root node, for example, by the auxiliary tree β_2 as shown above. Thus, for α_2 we will specify a local constraint $O(\beta_2)$ for the root node, indicating that α_2 requires for it to undergo an adjunction at the root node by an auxiliary tree β_2. In a fuller grammar there will be, of course, some alternatives in the scope of O( ).

3. PARSING TREE-ADJOINING LANGUAGES

3.1. Definitions

We will give a few additional definitions. These are not necessary for defining derivations in a TAG as defined in section 2. However, they are introduced to help explain the parsing algorithm and the proofs for some of the closure properties of TAL's.

DEFINITION 3.1 Let γ,γ' be two trees. We say $\gamma \vdash \gamma'$ if in γ we adjoin an auxiliary tree to obtain γ'.
$\vdash^*$ is the reflexive, transitive closure of $\vdash$.

DEFINITION 3.2 γ' is called a derived tree if $\gamma \vdash^* \gamma'$ for some elementary tree γ.
We then say $\gamma' \in D(\gamma)$.

The frontier of any derived tree γ belongs to either $\Sigma^* N \Sigma^+ \cup \Sigma^+ N \Sigma^*$ if $\gamma \in D(\beta)$ for some auxiliary tree β, or to Σ^* if $\gamma \in D(\alpha)$ for some initial tree α. Note if $\gamma \in D(\alpha)$ for some initial tree α, then γ is also a sentential tree.

If β is an auxiliary tree, $\gamma \in D(\beta)$ and the frontier of γ is w_1 X w_2 (X is a nonterminal, $w_1, w_2 \in \Sigma^*$) then the leaf node having this non-terminal symbol X at the frontier is called the foot of γ.

Sometimes we will be loosely using the phrase "adjoining with a derived tree" $\gamma \in D(\beta)$ for some auxiliary tree β. What we mean is that suppose we adjoin β at some node and then adjoin within β and so on, we can derive the desired derived tree $\in D(\beta)$ which uses the same adjoining sequence and use this resulting tree to "adjoin" at the original node.

3.2. The Parsing Algorithm

The algorithm, we present here to parse Tree-Adjoining Languages (TALs), is a modification of the CYK algorithm (which is described in detail in [Aho and Ullman, 1973]), which uses a dynamic programming technique to parse CFL's. For the sake of making our description of the parsing algorithm simpler, we shall present the algorithm for parsing without considering local constraints. We will later show how to handle local constraints.

We shall assume that any node in the elementary trees in the grammar has atmost two children. This assumption can be made without any loss of generality, because it can be easily shown that for any TAG G there is an equivalent TAG G_1 such that any node in any elementary tree in G_1 has atmost two children. A similar assumption is made in CYK algorithm. We use the terms ancestor and descendant, throughout the paper as a transitive and reflexive relation, for example, the foot node may be called the ancestor of the foot node.

The algorithm works as follows. Let $a_1...a_n$ be the input to be parsed. We use a four-dimensional array A; each element of the array contains a subset of the nodes of derived trees. We say a node X of a derived tree γ belongs to A[i,j,k,l] if X dominates a sub-tree of γ whose frontier is given by either $a_{i+1}...a_j$ Y $a_{k+1}...a_n$ (where the foot node of γ is labelled by Y) or $a_{i+1}...a_n$ (i.e., j = k. This corresponds to the case when γ is a sentential tree). The indices (i,j,k,l) refer to the positions between the input symbols and range over 0 through n. If i = 5 say, then it refers to the gap between a_5 and a_6.

Initially, we fill A[i,i+1,i+1,i+1] with those nodes in the frontier of the elementary trees whose label is the same as the input a_{i+1} for $0 \leq i \leq n-1$. The foot nodes of auxiliary trees will belong to all A[i,i,j,j], such that $i \leq j$.

We are now in a position to fill in all the elements of the array A. There are five cases to be considered.

Case 1. We know that if a node X in a derived tree is the ancestor of the foot node, and node Y is its right sibling, such that X $\in$ A[i,j,k,l] and Y $\in$ A[l,m,m,n], then their parent, say, Z should belong to A[i,j,k,n], see Fig 3.1a.

Case 2. If the right sibling Y is the ancestor of the foot node such that it belongs to A[l,m,n,p] and its left sibling X belongs to A[i,j,j,l], then we know that the parent Z of X and Y belongs to A[i,m,n,p], see Fig 3.1b

Case 3. If neither X nor its right sibling Y are the ancestors of the foot node (or there is no foot node) then if X $\in$ A[i,j,j,l] and Y $\in$ A[l,m,m,n] then their parent Z belongs to A[i,j,j,n].

Case 4. If a node Z has only one child X, and if X $\in$ A[i,j,k,l], then obviously Z $\in$ A[i,j,k,l].

Case 5. If a node X $\in$ A[i,j,k,l], and the root Y of a derived tree γ having the same label as that of X, belongs to A[m,i,l,n], then adjoining γ at X makes the resulting node to be in A[m,j,k,n], see Fig 3.1c.

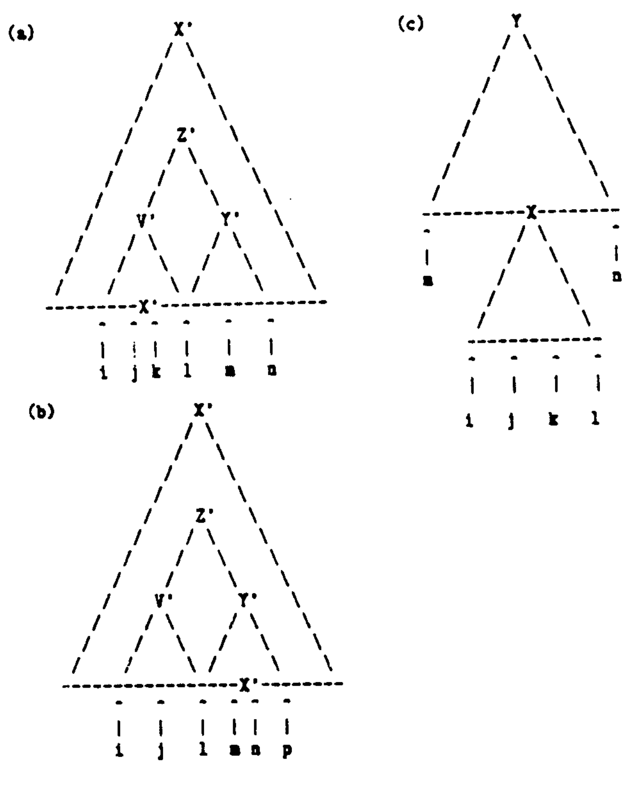


Figure 3.1

Although we have stated that the elements of the array contain a subset of the nodes of derived trees, what really goes in there are the addresses of nodes in the elementary trees. Thus the the size of any set is bounded by a constant, determined by the grammar. It is hoped that the presentation of the algorithm below will make it clear why we do so.

3.3. The algorithm

The complete algorithm is given below

```
Step 1 For i=0 to n-1 step 1 do

Step 2    put all nodes in the frontier of elementary
trees whose label is a_{i+1} in A[i,i+1,i+1,i+1].

Step 3 For i=0 to n-1 step 1 do

Step 4    for j=i to n-1 step 1 do

Step 5       put foot nodes of all auxiliary trees in
A[i,i,j,j]

Step 6 For l=0 to n step 1 do

Step 7    For i=l to 0 step -1 do

Step 8       For j=i to l step 1 do

Step 9          For k=l to j step -1 do

Step 10            do Case 1

Step 11            do Case 2

Step 12            do Case 3

Step 13            do Case 5

Step 14            do Case 4

Step 15 Accept if root of some initial tree ∈ A[0,j,j,n],
 0 ≤ j ≤ n
```

where,

(a) Case 1 corresponds to situation where the left sibling is the ancestor of the foot node. The parent is put in A[i,j,k,l] if the left sibling is in A[i,j,k,m] and the right sibling is in A[m,p,p,l], where $k \leq m < l$, $m \leq p$, $p \leq l$. Therefore Case 1 is written as

```
   For m=k to l-1 step 1 do

     for p= m to l step 1 do

        if there is a left sibling in A[i,j,k,m] and the
right sibling in A[m,p,p,l] satisfying appropriate
restrictions then put their parent
in A[i,j,k,l].
```

(b) Case 2 corresponds to the case where the right sibling is the ancestor of the foot node. If the left sibling is in A[i,m,m,p] and the right sibling is in A[p,j,k,l], $i \leq m < p$ and $p \leq j$, then we put their parent in A[i,j,k,l]. This may be written as

```
  For m=i to j-1 step 1 do

    For p=m+1 to j step 1 do

       for all left siblings in A[i,m,m,p] and right
siblings
in A[p,j,k,l] satisfying appropriate restrictions put
their parents
in A[i,j,k,l].
```

(c) Case 3 corresponds to the case where neither children are ancestors of the foot node. If the left sibling $\in A[i,j,j,m]$ and the right sibling $\in A[m,p,p,l]$ then we can put the parent in $A[i,j,j,l]$ if it is the case that ($i < j \leq m$ or $i \leq j < m$) and ($m < p \leq l$ or $m \leq p < l$). This may be written as

```
for m = j to l-1 step 1 do
    for p = j to l step 1 do
        for all left siblings in A[i,j,j,m] and
right siblings in A[m,p,p,l] satisfying the appropriate
restrictions put their parent in A[i,j,j,l].
```

(e) Case 5 corresponds to adjoining. If X is a node in $A[m,j,k,p]$ and Y is the root of a auxiliary tree with same symbol as that of X, such that Y is in $A[i,m,p,l]$ (($i \leq m \leq p < l$ or $i < m \leq p \leq l$) and ($m < j \leq k \leq p$ or $m \leq j \leq k < p$)). This may be written as

```
for m = i to j step 1 do
    for p = m to l step 1 do
        if a node X ∈ A[m,j,k,p] and the root of
auxiliary tree is in A[i,m,p,l] then put X in A[i,j,k,l]
```

Case 4 corresponds to the case where a node Y has only one child X If $X \in A[i,j,k,l]$ then put Y in $A[i,j,k,l]$. Repeat Case 4 again if Y has no siblings.

3.4. Complexity of the Algorithm

It is obvious that steps 10 through 15 (cases a-e) are completed in $O(n^2)$, because the different cases have at most two nested for loop statements, the iterating variables taking values in the range 0 through n. They are repeated atmost $O(n^4)$ times, because of the four loop statements in steps 6 through 9. The initialization phase (steps 1 through 5) has a time complexity of $O(n + n^2) = O(n^2)$. Step 15 is completed in $O(n)$. Therefore, the time complexity of the parsing algorithm is $O(n^6)$.

3.5. Correctness of the Algorithm

The main issue in proving the algorithm correct, is to show that while computing the contents of an element of the array A, we must have already determined the contents of other elements of the array needed to correctly complete this entry. We can show this inductively by considering each case individually. We give an informal argument below.

Case 1: We need to know the contents of $A[i,j,k,m]$, $A[m,p,p,l]$ where $m < l$, $i < m$, when we are trying to compute the contents of $A[i,j,k,l]$. Since l is the variable itererated in the outermost loop (step 6), we can assume (by induction hypothesis) that for all $m < l$ and for all p,q,r, the contents of $A[p,q,r,m]$ are already computed. Hence, the contents of $A[i,j,k,m]$ are known. Similarly, for all $m > i$, and for all p,q, and $r \leq l$, $A[m,p,q,r]$ would have been computed. Thus, $A[m,p,p,l]$ would also have been computed.

Case 2: By a similar reasoning, the contents of $A[i,m,m,p]$ and $A[p,j,k,l]$ are known since $p < l$ and $p > i$.

Case 3: When we are trying to compute the contents of some $A[i,j,j,l]$, we need to know the nodes in $A[i,j,j,p]$ and $A[p,q,q,l]$. Note $j > i$ or $j < l$. Hence, we know that the contents of $A[i,j,j,p]$ and $A[p,q,q,l]$ would have been computed already.

Case 5: The contents of $A[i,m,p,l]$ and $A[m,j,k,p]$ must be known in order to compute $A[i,j,k,l]$, where ($i \leq m \leq p < l$ or $i < m \leq p \leq l$) and ($m \leq j \leq k < p$ or $m < j \leq k \leq p$). Since either $m > i$ or $p < l$, contents of $A[m,j,k,p]$ will be known. Similarly, since either $m < j$ or $k < p$, the contents of $A[i,m,p,l]$ would have been computed.

3.6. Parsing with Local Constraints

So far,we have assumed that the given grammar has no local constraints. If the grammar has local constraints, it is easy to modify the above algorithm to take care of them. Note that in Case 5, if an adjunction occurs at a node X, we add X again to the element of the array we are computing. This seems to be in contrast with our definition of how to associate local constraints with the nodes in a sentential tree. We should have added the root of the auxiliary tree instead to the element of the array being computed, since so far as the local constraints are concerned,this node decides the local constraints at this node in the derived tree. However, this scheme cannot be adopted in our algorithm for obvious reasons. We let pairs of the form (X,C) belong to elements of the array, where X is as before and C represents the local constraints to be associated with this node.

We then alter the algorithm as follows. If (X,C_1) refers to a node at which we attempt to adjoin with an auxiliary tree (whose root is denoted by (Y,C_2)), then adjunction would determined by C_1. If adjunction is allowed, then we can add (X,C_2) in the corresponding element of the array. In cases 1 through 4, we do not attempt to add a new element if any one of the children has an obligatory constraint.

Once it has been determined that the given string belongs to the language, we can find the parse in a way similar to the scheme adopted in CYK algorithm.To make this process simpler and more efficient, we can use pointers from the new element added to the elements which caused it to be put there. For example, consider Case 1 of the algorithm (step 10). If we add a node Z to $A[i,j,k,l]$, because of the presence of its children X and Y in $A[i,j,k,m]$ and $A[m,p,p,l]$ respectively, then we add pointers from this node Z in $A[i,j,k,l]$ to the nodes X, Y in $A[i,j,k,m]$ and $A[m,p,p,l]$. Once this has been done, the parse can be found by traversing the tree formed by these pointers.

A parser based on the techniques described above is currently being implemented and will be reported at time of presentation.

4. CLOSURE PROPERTIES OF TAG's

In this section, we present some closure results for TALs. We now informally sketch the proofs for the closure properties. Interested readers may refer to [Vijay-Shankar and Joshi,1985] for the complete proofs.

4.1. Closure under Union

Let G_1 and G_2 be two TAGs generating L_1 and L_2 respectively. We can construct a TAG G such that $L(G)=L_1 \cup L_2$.

Let $G_1 = (I_1, A_1, N_1, S)$, and $G_2 = (I_2, A_2, N_2, S)$. Without loss of generality, we may assume that the $N_1 \cap N_2 = \phi$. Let $G = (I_1 \cup I_2, A_1 \cup A_2, N_1 \cup N_2, S)$. We claim that $L(G) = L_1 \cup L_2$.

Let $x \in L_1 \cup L_2$. Then $x \in L_1$ or $x \in L_2$. If $x \in L_1$, then it must be possible to generate the string x in G , since I_1 , A_1 are in G. Hence $x \in L(G)$. Similarly if $x \in L_2$, we can show that $x \in L(G)$. Hence $L_1 \cup L_2 \subseteq L(G)$. If $x \in L(G)$, then x is derived using either only I_1, A_1 or only I_2, A_2 since $N_1 \cap N_2 = \phi$. Hence, $x \in L_1$ or $x \in L_2$. Thus, $L(G) \subseteq L_1 \cup L_2$. Therefore, $L(G) = L_1 \cup L_2$.

4.2. Closure under Concatenation

Let $G_1 = (I_1, A_1, N_1, S_1)$, $G_2 = (I_2, A_2, N_2, S_2)$ be two TAGs generating L_1, L_2 respectively, such that $N_1 \cap N_2 = \phi$. We can construct a TAG $G = (I, A, N, S)$ such that $L(G) = L_1 . L_2$. We choose S such that S is not in $N_1 \cup N_2$. We let $N = N_1 \cup N_2 \cup \{S\}$, $A = A_1 \cup A_2$. For all $t_1 \in I_1$, $t_2 \in I_2$, we add t_{12} to I, as shown in Fig 4.2.1. Therefore, $I = \{ t_{12} / t_1 \in I_1, t_2 \in I_2 \}$, where the nodes in the subtrees t_1 and t_2 of the tree t_{12} have the same constraints associated with them as in the original grammars G_1 and G_2. It is easy to show that $L(G) = L_1 . L_2$, once we note that there are no auxiliary trees in G rooted with the symbol S, and that $N_1 \cap N_2 = \phi$.

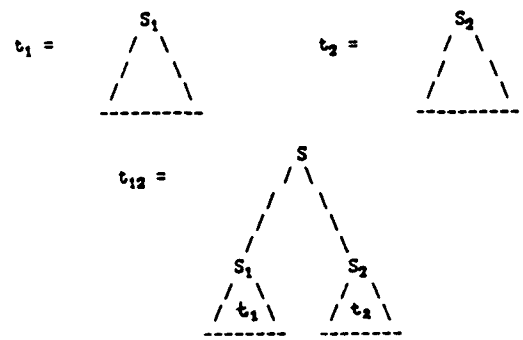


Figure 4.2.1

4.3. Closure under Kleene Star

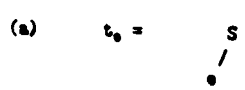
Let $G_1 = (I_1, A_1, N_1, S_1)$ be a TAG generating L_1. We can show that we can construct a TAG G such that $L(G) = L_1^*$. Let S be a symbol not in N_1, and let $N = N_1 \cup \{S\}$. We let the set I of initial trees of G be $\{t_e\}$, where t_e is the tree shown in Fig 4.3a. The set of auxiliary trees A is defined as

$$A = \{t_{1A} / t_1 \in I_1\} \cup A_1.$$

The tree t_{1A} is as shown in Fig 4.3b, with the constraints on the root of each t_{1A} being the null adjoining constraint, no constraints on the foot, and the constraints on the nodes of the subtrees t_1 of the trees t_{1A} being the same as those for the corresponding nodes in the initial tree t_1 of G_1.

To see why $L(G) = L_1^*$, consider $x \in L(G)$. Obviously, the tree derived (whose frontier is given by x) must be of the form shown in Fig 4.3c, where each t_i' is a sentential tree in G_1, such $t_i' \in D(t_i)$, for an initial tree t_i in G_1. Thus, $L(G) \subseteq L_1^*$.

On the other hand, if $x \in L_1^*$, then $x = w_1 ... w_n$, $w_i \in L_1$ for $1 \leq i \leq n$. Let each w_i then be the frontier of the sentential tree t_i' of G_1 such that $t_i' \in D(t_i)$, $t_i \in I_1$. Obviously, we can derive the tree T, using the initial tree t_e, and have a sequence of adjoining operations using the auxiliary trees t_{iA} for $1 \leq i \leq n$. From T we can obviously obtain the tree T' the same as given by Fig 4.3c, using only the auxiliary trees in A_1. The frontier of T' is obviously $w_1 ... w_n$. Hence, $x \in L(G)$. Therefore, $L_1^* \in L(G)$. Thus $L(G) = L_1^*$.



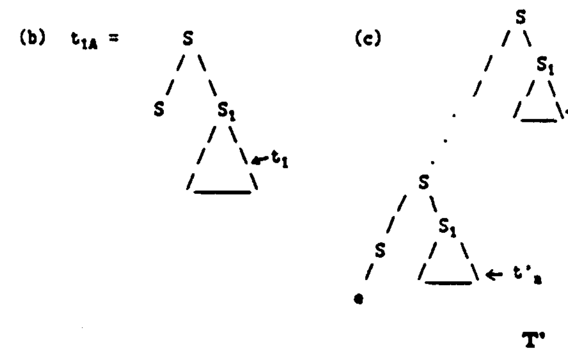


Figure 4.3

4.4. Closure under Intersection with Regular Languages

Let L_T be a TAL and L_R be a regular language. Let G be a TAG generating L_T and $M = (Q, \Sigma, \delta, q_0, Q_F)$ be a finite state automaton recognizing L_R. We can construct a grammar G and will show that $L(G_1) = L_T \cap L_R$.

Let α be an elementary tree in G. We shall associate with each node a quadruple (q_1, q_2, q_3, q_4) where $q_1, q_2, q_3, q_4 \in Q$. Let (q_1, q_2, q_3, q_4) be associated with a node X in α. Let us assume that α is an auxiliary tree, and that X is an ancestor of the foot node of α, and hence, the ancestor of the foot node of any derived tree γ in $D(\alpha)$. Let Y be the label of the root and foot nodes of α. If the frontier of γ (γ in $D(\alpha)$) is $w_1\ w_2\ Y\ w_3\ w_4$, and the frontier of the subtree of γ rooted at Z, which corresponds to the node X in α is $w_2\ Y\ w_3$. The idea of associating (q_1, q_2, q_3, q_4) with X is that it must be the case that $\delta^*(q_1, w_2) = q_2$, and $\delta^*(q_3, w_3) = q_4$. When γ becomes a part of the sentential tree γ' whose frontier is given by $u\ w_1\ w_2\ v\ w_3\ w_4\ w$, then it must be the case that $\delta^*(q_2, v) = q_3$. Following this reasoning, we must make $q_2 = q_3$, if Z is not the ancestor of the foot node of γ, or if γ is in $D(\alpha)$ for some initial tree α in G.

We have assumed here, as in the case of the parsing algorithm presented earlier, that any node in any elementary tree has atmost two children.

From G we can obtain G_1 as follows. For each initial tree α, associate with the root the quadruple (q_0, q, q, q_f) where q_0 is the initial state of the finite state automaton M, and $q_f \in Q_F$. For each auxiliary tree β of G, associate with the root the quadruple (q_1, q_2, q_3, q_4), where q, q_1, q_2, q_3, q_4 are some variables which will later be given values from Q. Let X be some node in some elementary tree α. Let (q_1, q_2, q_3, q_4) be associated with X. Then, we have to consider the following cases

Case 1: X has two children Y and Z. The left child Y is the ancestor of the foot node of α. Then associate with Y the quadruple (p, q_2, q_3, q), and (q, r, r, s) with Z, and associate with X the constraint that only those trees whose root has the quadruple (q_1, p, s, q_4), among those which were allowed in the original grammar, may be adjoined at this node. If $q_1 \neq p$, or $q_4 \neq s$, then the constraint associated with X must be made obligatory. If in the original grammar X had an obligatory constraint associated with it then we retain the obligatory constraint regardless of the relationship between q_1 and p, and q_4 and s. If the constraint associated with X is a null adjoining constraint, we associate (q_1, q_2, q_3, q), and (q, r, r, q_4) with Y and Z respectively, and associate the null adjoining constraint with X. If the label of Z is a, where $a \in \Sigma$, then we choose s and q such that $\delta(q, a) = s$. In the null adjoining constraint case, q is chosen such that $\delta(q, a) = q_4$.

Case 2: This corresponds to the case where a node X has two children Y and Z, with (q_1,q_2,q_3,q_4) associated at X. Let Z (the right child) be the ancestor of the the foot node the tree α. Then we shall associate (p,q,q,r), (r,q_2,q_3,s) with Y and Z. The associated constraint with X shall be that only those trees among those which were allowed in the orignal grammar may be adjoined provided their root has the quadruple (q_1,p,s,q_4) associated with it. If $q_1 \neq p$ or $q_4 \neq r$ then we make the constraint obligatory. If the original grammar had obligatory constraint we will retain the obligatory constraint. Null constraint in the original grammar will force us to use null constraint and not consider the cases where it is not the case that $q_1 = p$ and $q_4 = s$. If the label of Y is a terminal 'a' then we choose r such that $\delta^*(p,a) = r$. If the constraint at X is a null adjoining constraint, then $\delta^*(q_1,a) = r$.

Case 3: This corresponds to the case where neither the left child Y nor the right child Z of the node X is the ancestor of the foot node of α or if α is a initial tree. Then $q_2 = q_3 = q$. We will associate with Y and Z the quadruples (p,r,r,q) and (q,s,s,t) resp. The constraints are assigned as before , in this case it is dictated by the quadruple (q_1,p,t,q_4). If it is not the case that $q_1 = p$ and $q_4 = t$, then it becomes an OA constraint. The OA and NA constraints at X are treated similar to the previous cases, and so is the case if either Y or Z is labelled by a terminal symbol.

Case 4: If (q_1,q_2,q_3,q_4) is associated with a node X, which has only one child Y, then we can deal with the various cases as follows. We will associate with Y the quadruple (p,q_2,q_3,s) and the constraint that root of the tree which can be adjoined at X should have the quadruple (q_1,p,s,q_4) associated with it among the trees which were allowed in the original grammar, if it is to be adjoined at X. The cases where the original grammar had null or obligatory constraint associated with this node or Y is labelled with a terminal symbol, are treated similar to how we dealt with them in the previous cases.

Once this has been done, let $q_1,\ldots,q_m$ be the independent variables for this elementary tree α, then we produce as many copies of α so that $q_1,\ldots,q_m$ take all possible values from Q. The only difference among the various copies of α so produced will be constraints associated with the nodes in the trees. Repeat the process for all the elementary trees in G_1. Once this has been done and each tree given a unique name we can write the constraints in terms of these names. We will now show why $L(G_1) = L_T \cap L_R$.

Let $w \in L(G_1)$. Then there is a sequence of adjoining operations starting with an initial tree α to derive w. Obviously, $w \in L_T$, also since corresponding to each tree used in deriving w, there is a corresponding tree in G, which differs only in the constraints associated with its nodes. Note, however, that the constraints associated with the nodes in trees in G_1 are just a restriction of the corresponding ones in G, or an obligatory constraint where there was none in G. Now, if we can assume (by induction hypothesis) that if after n adjoining operations we can derive $\gamma' \in D(\alpha')$, then there is a corresponding tree $\gamma \in D(\alpha)$ in G, which has the same tree structure as γ' but differing only in the constraints associated with the corresponding nodes, then if we adjoin at some node in γ' to obtain γ_1', we can adjoin in γ to obtain γ_1 (corresponding to γ_1'). Therefore, if w can be derived in G_1, then it can definitely be derived in G.

If we can also show that $L(G_1) \subseteq L_R$, then we can conclude that $L(G_1) \subseteq L_T \cap L_R$. We can use induction to prove this. The induction hypothesis is that if all derived trees obtained after $k \leq n$ adjoining operations have the property P then so will the derived trees after n + 1 adjoinings where P is defined as,

Property P: If any node X in a derived tree γ has the foot-node of the tree β to which X belongs labelled Y as a descendant such that w_1 Y w_2 is the frontier of the subtree of β rooted at X, then if (q_1,q_2,q_3,q_4) had been associated with X, $\delta^*(q_1,w_1) = q_2$ and $\delta^*(q_3,w_2) = q_4$, and if w is the frontier of the subtree under the foot node of β in γ is then $\delta^*(q_2,w) = q_3$. If X is not the ancestor of the foot node of β then the subtree of β below is of the form w_1w_2. Suppose X has associated with it (q_1,q,q,q_2) then $\delta^*(q_1,w_1) = q$, $\delta^*(q,w_2) = q_2$.

Actually what we mean by an adjoining operation is not necessarily just one adjoining operation but the minimum number so that no obligatory constraints are associated with any nodes in the derived trees. Similarly, the base case need not consider only elementary trees, but the smallest (in terms of the number of adjoining operations) tree starting with elementary trees which has no obligatory constraint associated with any of its nodes. The base case can be seen easily considering the way the grammar was built (it can be shown formally by induction on the height of the tree) The inductive step is obvious. Note that the derived tree we are going to use for adjoining will have the property P, and so will the tree at which we adjoin; the former because of the way we designed the grammar and assigned constraints, and the latter because of induction hypothesis. Thus so will the new derived tree. Once we have proved this, all we have to do to show that $L(G_1) \subseteq L_R$ is to consider those derived trees which are sentential trees and observe that the roots of these trees obey property P.

Now, if a string $x \in L_T \cap L_R$, we can show that $x \in L(G)$. To do that, we make use of the following claim.

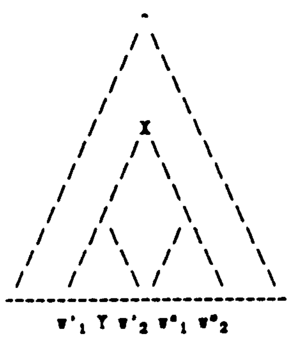
Let β be an auxiliary tree in G with root labelled Y and let $\gamma \in D(\beta)$. We claim that there is a β' in G_1 with the same structure as β, such that there is a γ' in D(beta())') where γ' has the same structure as γ, such that there is no OA constraint in γ'. Let X be a node in β_1 which was used in deriving γ. Then there is a node X' in γ' such that X' belongs to the auxilliary tree β_1' (with the same structure as β_1. There are several cases to consider -

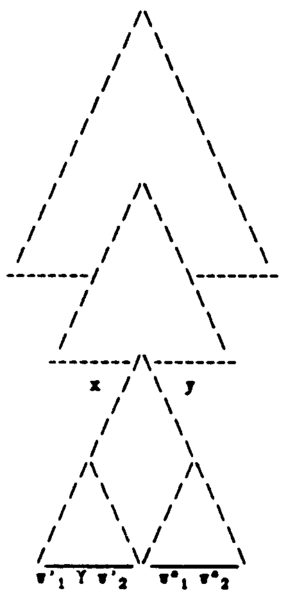
Case 1: X is the ancestor of the foot node of β_1, such that the frontier of the subtree of β_1 rooted at X is w_3Yw_4 and the frontier of the subtree of γ rooted at X is $w_3w_1Zw_2w_4$. Let $\delta^*(q_1,w_3) = q$, $\delta^*(q,w_1) = q_2$, $\delta^*(q_3,w_2) = r$, and $\delta^*(r,w_4) = q_4$. Then X' will have (q_1,q,r,q_4) associated with it, and there will be no OA constraint in γ'.

Case 2: X is the ancestor of the foot node of β_1, and the frontier of the subtree of β_1 rooted at X is w_3Yw_4. Let the frontier of the subtree of γ rooted at X is $w_3w_1w_2w_4$. Then we claim that X' in γ' will have associated with it the quadruple (q_1,q,r,q_4), if $\delta^*(q_1,w_3) = q$, $\delta^*(q,w_1) = p$, $\delta^*(p,w_2) = r$, and $\delta^*(r,w_4) = q_4$.

Case 3: Let the frontier of the subtree of β_1 (and also γ) rooted at X is w_1w_2. Let $\delta^*(q,w_1) = p$, $\delta^*(p,w_2) = r$. Then X' will have associated with it the quadruple (q,p,p,r).

We shall prove our claim by induction on the number of adjoining operations used to derive γ. The base case (where $\gamma = \beta$) is obvious from the way the grammar G_1 was built. We shall now assume that for all derived trees γ, which have been derived from β using k or less adjoining operations, have the property as required in our claim. Let γ be a derived tree in β after k adjunctions. By our inductive hypothesis we may assume the existence of the corresponding derived tree $\gamma' \in D(\beta')$ derived in G_1. Let X be a node in γ as shown in Fig. 4.4.1. Then the node X' in γ' corresponding to X will have associated with it the quadruple (q_1',q_2',q_3',q_4'). Note we are assuming here that the left child Y' of X' is the ancestor of the foot node of β'. The quadruples (q_1', q_2', q_3', p) and (p, p_1, p_1, q_4^*) will be associated with Y' and Z' (by the induction hypothesis). Let γ_1 be derived from γ by adjoining β_1 at X as in Fig. 4.4.2. We have to show the existence of β_1' in G_1 such that the root of this auxiliary tree has associated with it the quadruple (q, q_1', q_4^*, r). The existence of the tree follows from induction hypothesis ($k = 0$). We have also got to show that there exists γ_1' with the same structure as γ' but one that allows β_1' to be adjoined at the required node. But this should be so, since from the way we obtained the trees in G_1, there will exist γ_1^* such that X_1' has the quadruple (q, q_2', q_3', r) and the constraints at X_1' are dictated by the quadruple (q, q_1', q_4^*, r), but such that the two children of X_1' will have the same quadruple as in γ'. We can now adjoin β_1' in γ_1^* to obtain γ_1'. It can be shown that γ_1' has the required property to establish our claim.





$\delta^*(q'_1, v'_1) = q'_2 \quad \delta^*(p, v''_1) = p_1$

$\delta^*(q'_3, v'_2) = p \quad \delta^*(p_1, v''_2) = q''_4 \quad \delta^*(q, x) = q'_1 \quad \delta^*(q''_4, y) = r$

Figure 4.4.1

Figure 4.4.2

Firstly, any node below the foot of β_1' in γ_1' will satisfy our requirements as they are the same as the corresponding nodes in γ_1^*. Since β_1' satisfies the requirement, it is simple to observe that the nodes in β_1' will, even after the adjunction of β_1' in γ_1^*. However, because the quadruple associated with X_1' are different, the quadruples of the nodes above X_1' must reflect this change. It is easy to check the existence of an auxiliary tree such that the nodes above X_1' satisfy the requirements as stated above. It can also be argued on the basis of the design of grammar G_1, that there exists trees which allow this new auxiliary tree to be adjoined at the appropriate place. This then allows us to conclude that there exist a derived tree for each derived tree belongin to $D(\beta)$ as in our claim. The next step is to extend our claim to take into account all derived trees (i.e., including the sentential trees). This can be done in a manner similar to our treatment of derived trees belonging to $D(\beta)$ for some auxiliary tree β as above. Of course, we have to consider only the case where the finite state automaton starts from the initial state q_0, and reaches some final state q_f on the input which is the frontier of some sentential tree in G. This, then allows us to conclude that $L_T \cap L_R \subseteq L(G_1)$. Hence, $L(G_1) = L_T \cap L_R$.

5. HEAD GRAMMARS AND TAG's

In this section, we attempt to show that Head Grammars (HG) are remarkably similar to Tree Adjoining Grammars. It appears that the basic intuition behind the two systems is more or less the same. Head Grammars were introduced in [Pollard,1984], but we follow the notations used in [Roach,1984]. It has been observed that TAG's and HG's share a lot of common formal properties such as almost identical closure results, similar pummping lemma.

Consider the basic operation in Head Grammars - the Head Wrapping operation. A derivation from a non-terminal produces a pair $(i, a_1 \ldots a_i \ldots a_n)$ (a more convenient representation for this pair is $a_1 \ldots a_i \uparrow a_{i+1} \ldots a_n$). The arrow denotes the head of the string, which in turn determines where the string is split up when wrapping operation takes place. For example, consider X->LL_2(A,B), and let $A \Rightarrow^* wh \uparrow x$ and $B \Rightarrow^* ug \uparrow v$. Then we say, $X \Rightarrow^* whug \uparrow vx$.

We shall define some functions used in the HG formalism, which we need here. If A derives in 0 or more steps the headed string whx and B derives ugv, then

1) if X -> LL_1(A,B) is a rule in the grammar then
   X derives whugvx
2) if X -> LL_2(A,B) is a rule in the grammar then
   X derives whugvx
3) if X -> LC_1(A,B) is a rule in the grammar then
   X derives whxugv
4) if X -> LC_2(A,B) is a rule in the grammar then
   X derives whxugv

Now consider how a derivation in TAGs proceeds -

Let β be an auxilliary tree and let α be a sentential tree as in Fig 5.1. Adjoining β at the root of the sub-tree γ gives us the sentential tree in Fig 5.1. We can, now see how the string whx has "wrapped around" the sub-tree i.e.the string ugv. This seems to suggest that there is something similiar in the role played by the foot in an auxilliary tree and the head in a Head Grammar how the adjoining operations and head-wrapping operations operate on strings. We could say that if X is the root of an auxilliary tree β and $a_1 \ldots a_i$ X $a_{i+1} \ldots a_n$ is the frontier of a derived tree $\gamma \in D(\beta)$, then the derivation of γ would correspond to a derivation from a non-terminal X to the string $a_1 \ldots a_i \uparrow a_{i+1} \ldots a_n$ in HG and the use of γ in some sentential tree would correspond to how the strings $a_1 \ldots a_i$ and $a_{i+1} \ldots a_n$ are used in deriving a string in HL.

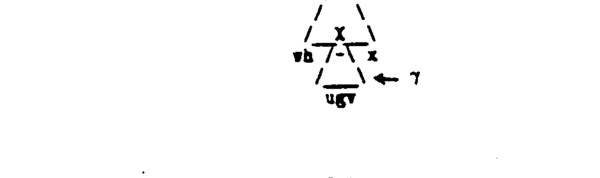


Figure 5.1

Based on this observation, we attempt to show the close relationship of TAL's and HL's. It is more convinient for us to think of the headed string $(i, a_1 \ldots a_n)$ as the string $a_1 \ldots a_n$ with the head pointing in between the symbols a_i and a_{i+1} rather than at the symbol a_i. The definition of the derivation operators can be extended in a straightforward manner to take this into account. However, we can acheive the same effect by considering the definitions of the operators LL,LC,etc. Pollard suggests that cases such as $LL_2(\bar{x}, \bar{\lambda})$ be left undefined. We shall assume that if $\bar{x} = w\underset{\uparrow}{h}y$ then $LL_2(\bar{x}, \bar{\lambda}) = wh\underset{\uparrow}{\lambda}y$, $LL_2(\bar{\lambda}, \bar{x}) = \bar{x}$, $LC_2(\bar{x}, \bar{\lambda}) = x\underset{\uparrow}{\lambda}$, $LC_2(\bar{\lambda}, \bar{x}) = \bar{x}$, $LC_1(\bar{x}, \bar{\lambda}) = \bar{x}$, and $LC_1(\bar{\lambda}, \bar{x}) = \underset{\uparrow}{\lambda}x$.
We, then say that if G is a Head Grammar, then $w_1 = whx$ belongs to L(G) if and only if S derives the headed string $w\underset{\uparrow}{h}x$ or $wh\underset{\uparrow}{\lambda}x$. With this new definition, we shall show, without giving the proof, that the class of TAL's is contained in the class of HL's, by systematically converting any TAG G to a HG G'. We shall assume, without loss of generality, that the constraints expressed at the nodes of elementary trees of G are -

1) Nothing can be adjoined at a node (NA),

2) Any appropriate tree (symbols at the node and root of the auxilliary tree must match) can be adjoined (AA), or

3) Adjoining at the node is obligatory (OA).

It is easy to show that these constraints are enough, and that selective adjoining can be expressed in terms of these and additional non-terminals. We know give a procedural description of obtaining an equivalent Head Grammar from a Tree-Adjoining Grammar. The procedure works as follows. It is a recursive procedure (Convert_to_HG) which takes in two parameters, the first representing the node on which it is being applied and the second the label appearing on the left-hand side of the HG productions for this node. If X is a nonterminal, for each auxiliary tree β whose root has the label X, we obtain a sequence of productions such that the first one has X on the left-hand side. Using these productions, we can derive the string $w_1 \underset{\uparrow}{\lambda} w_2$ where a derived tree in $D(\beta)$ has a frontier $w_1 Y w_2$. If Y is a node with with label X in some tree where adjoining is allowed, we introduce the productions

Y' -> LL_2(X,N') {so that a derived tree with root label X may wrap around the string derived from the subtree below this node}

N' -> $LC_i(A_1, \ldots, A_j)$ {assuming that there are j children of this node and the i^{th} child is the ancestor of the foot node. By calling the procedure recursively for all the j children of Y with A_k,k ranging from 1 through j, we can derive from N' the frontier of the subtree below Y}

Y' -> N' { this is to handle the case where no adjunction takes place at Y}

If G is a TAG then we do the following –

Repeat for every Initial tree

Convert_to_HG(root,S') {S' will be the start symbol of the new Head Grammar}.

Repeat for each Auxilliary tree

Convert_to_HG(root,rootsymbol)

where Convert_to_HG(node,name) is defined as follows

if node is an internal node then

case 1 If the constraint at the node is AA

add productions Sym->LL_2(node symbol,N').

N'->$LC_i(A_1', \ldots, A_i', \ldots, A_j')$

Sym->$LC_i(A_1', \ldots, A_i', \ldots, A_j')$

where $N', A_1', A_2', \ldots A_j'$ are new non-terminal symbols, $A_1, \ldots, A_j$ correspond to the j children of the node and i=1 if foot node is not a descendant of node else =l such that the l^{th} child of node is ancestor of foot node, j=number of children of node

for k=1 to j step 1 do

Convert_to_HG(k^{th} child of node,A_k').

Case 2 The constraint at the node is NA.

Same as Case 1 except don't add the productions
Sym->LL_i(node symbol,N'),
N'->$LC_i(A_1', \ldots, A_j')$.

Case 3 The constraint at the node is OA.

Same as Case 1 except that we don't add
Sym->$LC_i(A_1', \ldots A_j')$

else if the node has a terminal symbol a,

then add the production Sym ->$\bar{a}$

else {it is a foot node }

if the constraint at the foot node is AA then add the productions
Sym ->LL_2(node symbol,$\bar{\lambda}$)/$\bar{\lambda}$

if the constraint is OA then add only the production
Sym ->LL_2(node symbol,$\bar{\lambda}$)

if the constraint is NA add the production
Sym ->$\bar{\lambda}$

We shall now give an example of converting a TAG G to a HG. G contains a single initial tree α, and a single auxiliary tree β as in Fig. 5.2.

```
        S                        S (φ)
α =     |              β =      / \
        e                      /   \
                              a     S
                                  / | \
                                 /  |  \
                                /   |   \
                               b   S(φ)  c
```

Figure 5.2

Obviously, $L(G) = \{a^n b^n c^n \ / \ n \geq 0\}$

Applying the procedure Convert_to_HG to this grammar we obtain the HG whose productions are given by-

$$
\begin{aligned}
S' &\rightarrow LL_2(S,A)\\
A &\rightarrow \bar{\lambda}\\
S &\rightarrow LC_2(B,C)\\
B &\rightarrow \bar{a}\\
C &\rightarrow LL_2(S,D)/D\\
D &\rightarrow LC_2(E,F,G)\\
E &\rightarrow \bar{b}\\
F &\rightarrow \bar{\lambda}\\
G &\rightarrow \bar{c}
\end{aligned}
$$

which can be rewritten as

$$
\begin{aligned}
S' &\rightarrow S/\bar{\lambda}\\
S &\rightarrow LC_2(a,A')\\
A' &\rightarrow LL_2(S,b\underset{\uparrow}{\lambda}c) \text{ or } A' \rightarrow LL_2(S,\underset{\uparrow}{b}c)
\end{aligned}
$$

It can be verified that this grammar generates exactly L(G).

It is worth emphasising that the main point of this exercise was to show the similarities between Head Grammars and Tree Adjoining Grammars. We have shown how a HG G' (using our extended definitions) can be obtained in a systematic fashion from a TAG G. It is our belief that the extension of the definition may not necessary. Yet, this conversion process should help us understand the similarities between the two formalisms.

6. OTHER MATHEMATICAL PROPERTIES OF TAG's

Additional formal properties of TAG's have been discussed in [Vijay-Shankar and Joshi,1985]. Some of them are listed below

1) Pumping lemma for TAG's
2) TAL's are closed under substitution and homomorphisms
3) TAL's are not closed under the following operations
    a) intersection with TAL's
    b) intersection with CFL's
    c) complementation

Some other properties that have been considered in [Vijay-Shankar and Joshi,1985] are as follows

1) closure under the following properties
    a) inverse homomorphism
    b) gsm mappings
2) semilinearity and Parikh-boundedness.

References

1. Aho,A.V., and Ullman,J.D., 1973 "Theory of Parsing, Translation, and Compiling, Volume 1: Parsing, Prentice-Hall, Englewood Cliffs, N.J., 1973.

2. Joshi,A.K., 1983 "How much context-sensitivity is necessary for charecterizing structural descriptions - tree adjoining grammars" in Natural Language Processing - Theoretical, Computational, and Psychological Perspectives (ed. D.Dowty, L.Karttunen, A.Zwicky), Cambridge University Press, New York, (originally presented in 1983) to appear in 1985.

3. Joshi,A.K., and Levy,L.S., 1977 "Constraints on Structural Descriptions: Local Transformations", SIAM Journal of Computing June 1977.

4. Joshi,A.K., Levy,L.S., and Takahashi, M., 1975 "Tree adjoining grammars", Journal of Computer Systems and Sciences, March 1975

5. Kroch, T., and Joshi, A.K., 1985 "Linguistic relevance of tree adjoining grammars", Technical Report, MS-CIS-85-18, Dept. of Computer and Information Science, University of Pennsylvania, April 1985

6. Pollard, C., 1984 "Generalized Phrase Structure Grammars, Head Grammars, and Natural language", Ph.D dissertation, Stanford University, August 1984

7. Roach, K., 1984 "Formal Properties of Head Grammars", unpublished manuscript, Stanford University, also presented at the Mathematics of Languages workshop at the University of Michigan, Ann Arbor, Oct. 1984.

8. Vijay-Shankar,K., Joshi,A.K., 1985 "Formal Properties of Tree Adjoining Grammars", Technical Report, Dept. of Computer and Information Science, University of Pennsylvania, July 1985.