March 1991

Tree-Adjoining Grammars and Lexicalized Grammars

Aravind K. Joshi
University of Pennsylvania

Yves Schabes
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science, Technical Report No. MS-CIS-91-22.

Tree-Adjoining Grammars and Lexicalized Grammars

Abstract

In this paper, we will describe a tree generating system called tree-adjoining grammar(TAG)and state some of the recent results about TAGs. The work on TAGS is motivated by linguistic considerations. However, a number of formal results have been established for TAGs, which we believe, would be of interest to researchers in tree grammars and tree automata. After giving a short introduction to TAG, we briefly state these results concerning both the properties of the string sets and tree sets (Section 2). We will also describe the notion of lexicalization of grammars (Section 3) and investigate the relationship of lexicalization to context-free grammars (CFGs) and TAGS (Section 4).

Comments

University of Pennsylvania Department of Computer and Information Science, Technical Report No. MS-CIS-91-22.

Tree-Adjoining Grammars and Lexicalized Grammars

MS-CIS-91-22
LINC LAB 197

Aravind K. Joshi
Yves Schabes

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

March 1991

Tree-Adjoining Grammars and Lexicalized Grammars*

Aravind K. Joshi and Yves Schabes
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389, USA

1 Introduction

In this paper, we will describe a tree generating system called tree-adjoining grammar (TAG) and state some of the recent results about TAGs. The work on TAGs is motivated by linguistic considerations. However, a number of formal results have been established for TAGs, which we believe, would be of interest to researchers in tree grammars and tree automata. After giving a short introduction to TAG, we briefly state these results concerning both the properties of the string sets and tree sets (Section 2). We will also describe the notion of lexicalization of grammars (Section 3) and investigate the relationship of lexicalization to context-free grammars (CFGs) and TAGs (Section 4).

The notion of lexicalization is linguistically very significant. Formally, this notion leads to a tree representation for CFGs which may be regarded as a *stronger* version of the Greibach Normal Form for CFGs, in the sense that the structures are preserved and not just the string sets.

The motivations for the study of tree-adjoining grammars (TAG) are of linguistic and formal nature. The elementary objects manipulated by a TAG are trees, i.e., structured objects and not strings. Using structured objects as the elementary objects of a formalism, it is possible to construct formalisms whose properties relate directly to the strong generative capacity (structural description) which is more relevant to linguistic descriptions than the weak generative capacity (set of strings).

TAG is a tree-generating system rather than a string generating system. The set of trees derived in a TAG constitute the object language. Hence, in order to describe the derivation of a tree in the object language, it is necessary to talk about derivation 'trees' for the object language trees. These derivation trees are important both syntactically and semantically. It has also turned out that some other formalisms which are weakly

*This paper will appear in Nivat and Podelski (editors), *Definability and Recognizability of Sets of Trees*, Elsevier, 1991. This work was partially supported by NSF grants DCR-84-10413, ARO Grant DAAL03-87-0031, and DARPA Grant N0014-85-K0018.
We are grateful to Anne Abeillé, Bruno Courcelle, Anthony Kroch, Mitch Marcus, Fernando Pereira, Stuart Shieber, Mark Steedman and Marilyn Walker for providing valuable comments.

equivalent to TAGs are similar to each other in terms of the properties of the derivation 'trees' of these formalisms (Weir, 1988; Joshi et al., forthcoming 1991).

Another important linguistic motivation for TAGs is that TAGs allow factoring recursion from the statement of linguistic constraints (dependencies), thus making these constraints strictly local, and thereby simplifying linguistic description (Kroch and Joshi, 1985).

Lexicalization of grammar formalism is also one of the key motivations, both linguistic and formal. Most current linguistic theories give lexical accounts of several phenomena that used to be considered purely syntactic. The information put in the lexicon is thereby increased in both amount and complexity[1].

On the formal side, lexicalization allows us to associate each elementary structure in a grammar with a lexical item (terminal symbol in the context of formal grammars). The well-known Greibach Normal Form (CNF) for CFG is a kind of lexicalization, however it is a *weak* lexicalization in a certain sense as it does not preserve structures of the original grammar. Our tree based approach to lexicalization allows us to achieve lexicalization while preserving structures, which is linguistically very significant.

2 Tree-Adjoining Grammars

TAGs were introduced by Joshi, Levy and Takahashi (1975) and Joshi (1985). For more details on the original definition of TAGs, we refer the reader to (Joshi, 1987; Kroch and Joshi, 1985). It is known that tree-adjoining languages (TALs) generate some strictly context-sensitive languages and fall in the class of the so-called 'mildly context-sensitive languages' (Joshi et al., forthcoming 1991). TALs properly contain context-free languages and are properly contained by indexed languages.

Although the original definition of TAGs did not include substitution as a combining operation, it can be easily shown that the addition of substitution does not affect the formal properties of TAGs.

We first give an overview of TAG and then we study the lexicalization process.

Definition 1 (tree-adjoining grammar)
A tree-adjoining grammar (TAG) consists of a quintuple (Σ, NT, I, A, S), where

(i) Σ is a finite set of terminal symbols;
(ii) NT is a finite set of non-terminal symbols[2]: $\Sigma \cap NT = \emptyset$;
(iii) S is a distinguished non-terminal symbol: $S \in NT$;
(iv) I is a finite set of finite trees, called *initial trees*, characterized as follows (see tree on the left in Figure 1):

- interior nodes are labeled by non-terminal symbols;

[1]Some of the linguistic formalisms illustrating the increased use of lexical information are, lexical rules in LFG (Kaplan and Bresnan, 1983), GPSG (Gazdar et al., 1985), HPSG (Pollard and Sag, 1987), Combinatory Categorial Grammars (Steedman, 1987), Karttunen's version of Categorial Grammar (Karttunen, 1986), some versions of GB theory (Chomsky, 1981)), and Lexicon-Grammars (Gross, 1984).

[2]We use lower-case letters for terminal symbols and upper-case letters for non-terminal symbols.

- the nodes on the frontier of initial trees are labeled by terminals or non-terminals; non-terminal symbols on the frontier of the trees in I are marked for substitution; by convention, we annotate nodes to be substituted with a down arrow ($\downarrow$);

(v) A is a finite set of finite trees, called *auxiliary trees*, characterized as follows (see tree on the right in Figure 1):

- interior nodes are labeled by non-terminal symbols;
- the nodes on the frontier of auxiliary trees are labeled by terminal symbols or non-terminal symbols. Non-terminal symbol on the frontier of the trees in A are marked for substitution except for one node, called the *foot node*; by convention, we annotate the foot node with an asterisk ($*$); the label of the foot node must be identical to the label of the root node.

In *lexicalized TAG*, at least one terminal symbol (the anchor) must appear at the frontier of all initial or auxiliary trees.

The trees in $I \cup A$ are called *elementary trees*. We call an elementary tree an X-type elementary tree if its root is labeled by the non-terminal X.

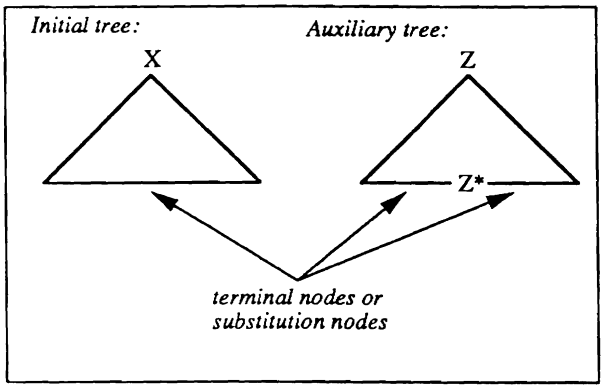


Figure 1: Schematic initial and auxiliary trees.

A tree built by composition of two other trees is called a *derived tree.*

We now define the two composition operations that TAG uses: *adjoining* and *substitution.*

Adjoining builds a new tree from an auxiliary tree β and a tree α (α is any tree, initial, auxiliary or derived). Let α be a tree containing a non-substitution node n labeled by X and let β be an auxiliary tree whose root node is also labeled by X. The resulting tree, γ, obtained by adjoining β to α at node n (see top two illustrations in Figure 2) is built as follows:

- the sub-tree of α dominated by n, call it t, is excised, leaving a copy of n behind.
- the auxiliary tree β is attached at the copy of n and its root node is identified with the copy of n.
- the sub-tree t is attached to the foot node of β and the root node of t (i.e. n) is identified with the foot node of β.

The top two illustrations in Figure 2 illustrate how adjoining works. The auxiliary tree β_1 is adjoined on the VP node in the tree α_2. α_1 is the resulting tree.

Substitution takes only place on non-terminal nodes of the frontier of a tree (see bottom two illustrations in Figure 2). An example of substitution is given in the fourth illustration (from the top) in Figure 2. By convention, the nodes on which substitution is allowed are marked by a down arrow ($\downarrow$). When substitution occurs on a node n, the node is replaced by the tree to be substituted. When a node is marked for substitution, only trees derived from initial trees can be substituted for it.

By definition, any adjunction on a node marked for substitution is disallowed. For example, no adjunction can be performed on any NP node in the tree α_2. Of course, adjunction is possible on the root node of the tree substituted for the substitution node.

2.1 Adjoining Constraints

In the system that we have described so far, an auxiliary tree β can be adjoined on a node n if the label of n is identical to the label of the root node of the auxiliary tree β and if n is labeled by a non-terminal symbol not annotated for substitution. It is convenient for linguistic description to have more precision for specifying which auxiliary trees can be adjoined at a given node. This is exactly what is achieved by *constraints on adjunction* (Joshi, 1987). In a TAG $G = (\Sigma, NT, I, A, S)$, one can, for each node of an elementary tree (on which adjoining is allowed), specify one of the following three constraints on adjunction:

- *Selective Adjunction* ($SA(T)$, for short): only members of a set $T \subseteq A$ of auxiliary trees can be adjoined on the given node. The adjunction of an auxiliary is not mandatory on the given node.
- *Null Adjunction* (NA for short): it disallows any adjunction on the given node.[3]
- *Obligatory Adjunction* ($OA(T)$, for short): an auxiliary tree member of the set $T \subseteq A$ must be adjoined on the given node. In this case, the adjunction of an auxiliary tree is mandatory. OA is used as a notational shorthand for $OA(A)$.

If there are no substitution nodes in the elementary trees and if there are no constraints on adjoining, then we have the 'pure' (old) Tree Adjoining Grammar (TAG) as described in (Joshi et al., 1975).

The operation of substitution and the constraints on adjoining are both needed for linguistic reasons. Constraints on adjoining are also needed for formal reasons in order to obtain some closure properties.

[3]Null adjunction constraint corresponds to a selective adjunction constraint for which the set of auxiliary trees T is empty: $NA = SA(\emptyset)$

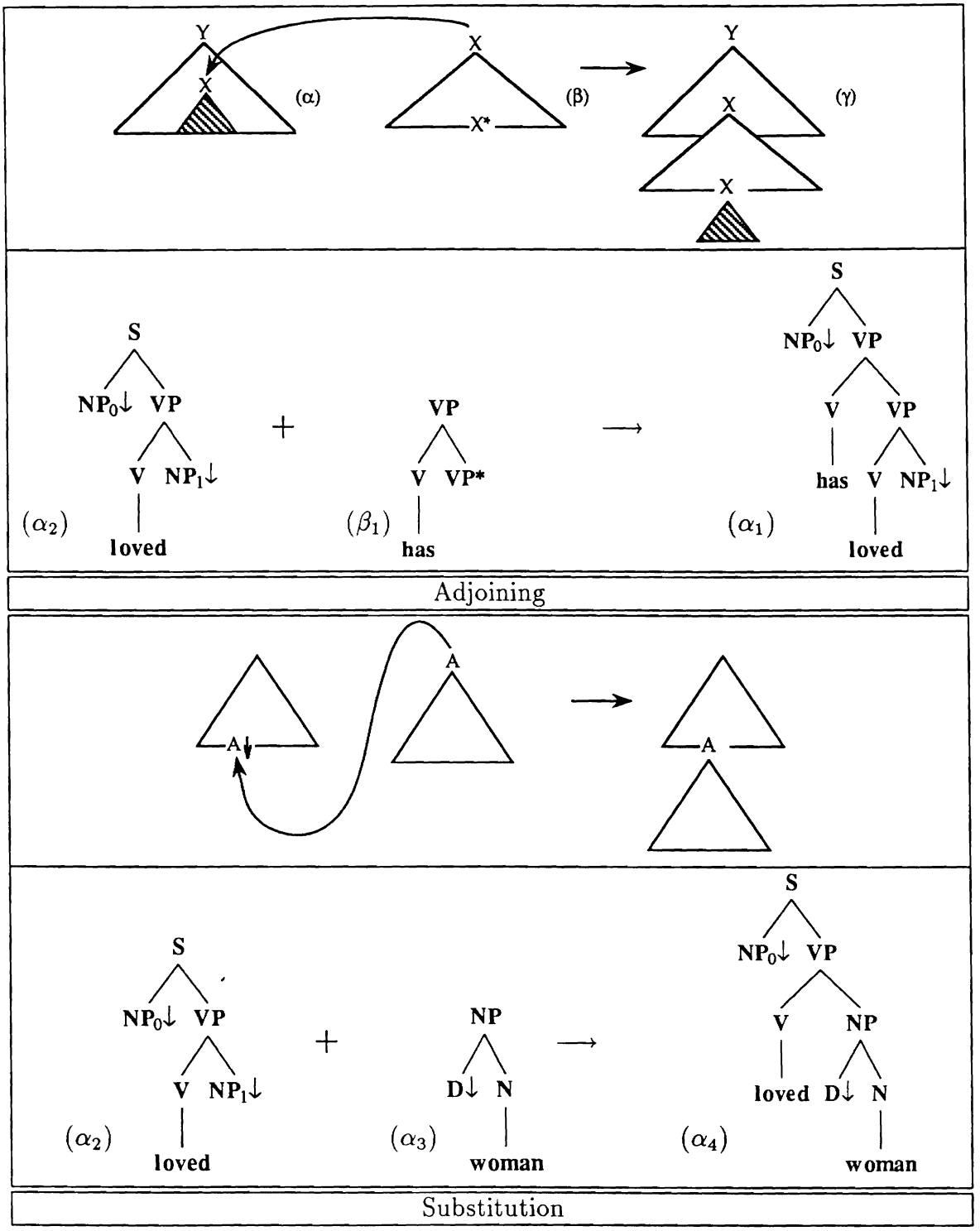


Figure 2: Combining operations: adjoining and substitution

2.2 Derivation in TAG

We now define by an example the notion of derivation in a TAG. Unlike CFGs, the tree obtained by derivation (the *derived tree*) does not give enough information to determine how it was constructed. The *derivation tree* is an object that specifies uniquely how a derived tree was constructed. Both operations, adjunction and substitution, are considered in a TAG derivation. Take for example the derived tree α_5 in Figure 3; α_5 yields the sentence *yesterday a man saw Mary.* It has been built with the elementary trees shown in Figure 4.

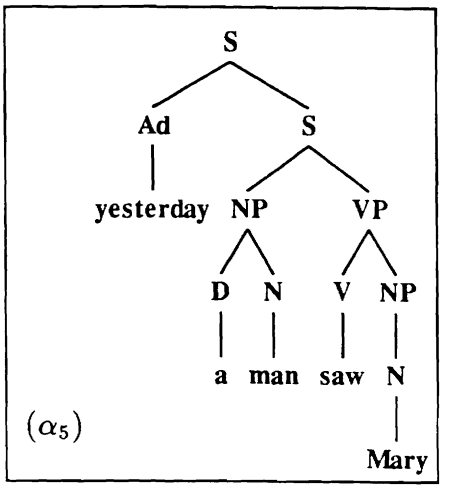


Figure 3: Derived tree for: *yesterday a man saw Mary.*

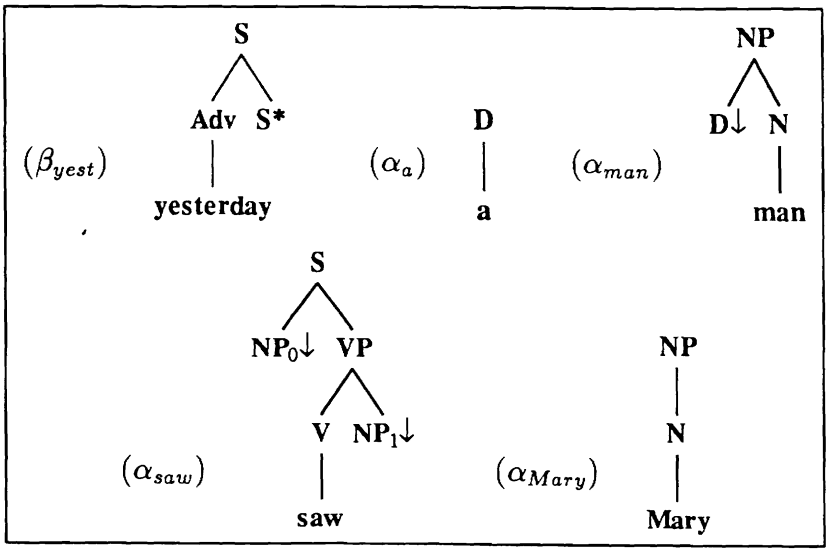


Figure 4: Some elementary trees.

The root of a derivation tree for TAGs is labeled by an S-type initial tree. All other nodes in the derivation tree are labeled by auxiliary trees in the case of adjunction or initial trees in the case of substitution. A tree address is associated with each node (except the root node) in the derivation tree. This tree address is the address of the

node in the parent tree to which the adjunction or substitution has been performed. We use the following convention: trees that are adjoined to their parent tree are linked by an unbroken line to their parent, and trees that are substituted are linked by a dashed line.[4] Since by definition, adjunction can only occur at a particular node one time, all the children of a node in the derivation tree will have distinct addresses associated with them.

The derivation tree in Figure 5 specifies how the derived tree α_5 pictured in Figure 3 was obtained.

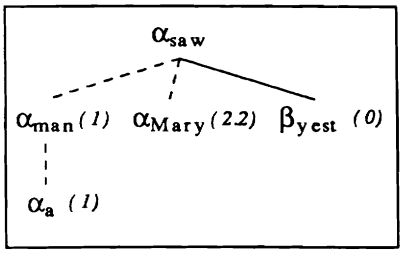


Figure 5: Derivation tree for *Yesterday a man saw Mary.*

This derivation tree (Figure 5) should be interpreted as follows: α_a is substituted in the tree α_{man} at the node of address 1 (D), α_{man} is substituted in the tree α_{saw} at address 1 (NP_0) in α_{man}, α_{Mary} is substituted in the tree α_{saw} at node $2 \cdot 2$ (NP_1) and the tree β_{yest} is adjoined in the tree α_{saw} at node 0 (S).

The order in which the derivation tree is interpreted has no impact on the resulting derived tree.

2.3 Some properties of the string languages and tree sets

We summarize some of the well known properties of tree-adjoining grammar's string languages and of the tree sets.

The *tree set* of a TAG is defined as the set of completed[5] initial trees derived from some S-rooted initial trees:

$$T_G \doteq \{t \mid t \textit{ is 'derived' from some S-rooted initial tree}\}$$

The string language of a TAG, $L(G)$, is then defined as the set of yields of all the trees in the tree set:

$$L_G = \{w \mid w \textit{ is the yield of some } t \textit{ in } T_G\}$$

Adjunction is more powerful than substitution and it generates some context-sensitive languages (see Joshi [1985] for more details). [6]

[4] We will use Gorn addresses as tree addresses: 0 is the address of the root node, k is the address of the k^{th} child of the root node, and $p \cdot q$ is the address of the q^{th} child of the node at address p.

[5] We say that an initial tree is *completed* if there is no substitution nodes on the frontier of it.

[6] Adjunction can simulate substitution with respect to the weak generative capacity. It is also possible to encode a context-free grammar with auxiliary trees using adjunction only. However, although the languages correspond, the possible encoding does not directly reflect the tree set of original context-free grammar since this encoding uses adjunction.

Some well known properties of the string languages follow:

- context-free languages are strictly included in tree-adjoining languages, which themselves are strictly included in indexed languages;

$$CFL \subset TAL \subset \textit{Indexed Languages} \subset CSL$$

- TALs are semilinear;
- All closure properties of context-free languages also hold for tree-adjoining languages. In fact, TALs are a full abstract family of languages (full AFLs).
- a variant of the push-down automaton called embedded push-down automaton (EPDA) (Vijay-Shanker, 1987) characterizes exactly the set of tree-adjoining languages, just as push-down automaton characterizes CFLs.
- there is a pumping lemma for tree-adjoining languages.
- tree-adjoining languages can be parsed in polynomial time, in the worst case in $O(n^6)$ time.

Some well know properties of the tree sets of tree-adjoining grammars follow:

- the tree sets of recognizable sets (regular tree sets)(Thatcher, 1971) are strictly included in the tree sets of tree-adjoining grammars, $\mathcal{T}(TAG)$;

$$\textit{recognizable sets} \subset \mathcal{T}(TAG)$$

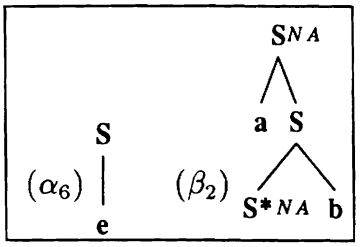
- the set of paths of all the trees in the tree set of a given TAG, $\mathcal{P}(T(G))$, is a context-free language;

$$\mathcal{P}(T(G)) \textit{ is a CFL}$$

- the tree sets of TAG are equivalent to the tree sets of linear indexed languages. Hence, linear versions of Schimpf-Gallier tree automaton (Schimpf and Gallier, 1985) are equivalent to $\mathcal{T}(TAG)$;
- for every TAG, G, the tree set of G, $T(G)$, is recognizable in polynomial time, in the worst case in $O(n^3)$-time, where n is the number of nodes in a tree $t \in T(G)$.

We now give two examples to illustrate some properties of tree-adjoining grammars.

Example 1 Consider the following TAG $G_1 = (\{a, e, b\}, \{S\}, \{\alpha_6\}, \{\beta_2\}, S)$



G_1 generates the language $L_1 = \{a^n e b^n | n \geq 1\}$. For example, in Figure 6, α_7 has been obtained by adjoining β_2 on the root node of α_6 and α_8 has been obtained by adjoining β_2 on the node at address 2 in in the tree α_7.

Although L_1 is a context-free language, G_1 generates cross serial dependencies. For example, α_7 generates, $a_1 e b_1$, and α_8 generates, $a_1 a_2 e b_2 b_1$. It can be shown that $T(G_1)$ is not recognizable.

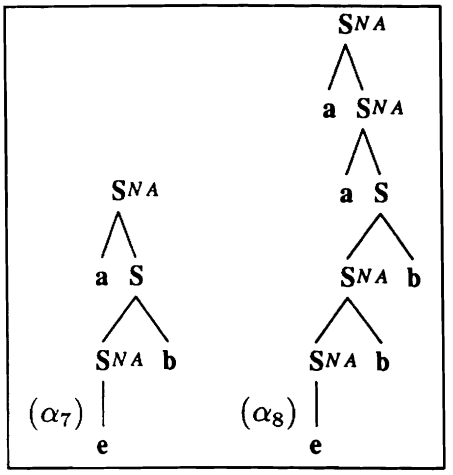
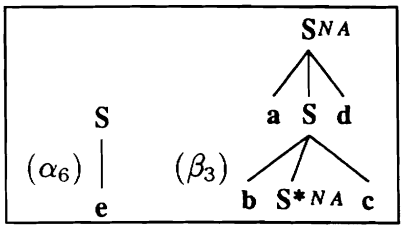


Figure 6: Some derived trees of G_1

Example 2 Consider the following TAG, $G_2 = (\{a, b, c, d, e\}, \{S\}, \{\alpha_6\}, \{\beta_3\}, S)$



G_2 generates the context-sensitive language $L_2 = \{a^n b^n e c^n d^n | n \geq 1\}$. For example, in Figure 7, α_9 has been obtained by adjoining β_3 on the root node of α_6 and α_{10} has been obtained by adjoining β_3 on the node at address 2 in in the tree α_9.

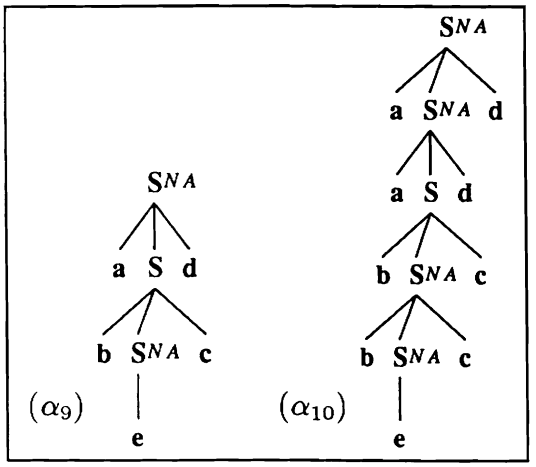


Figure 7: Some derived trees of G_2

One can show that $\{a^n b^n c^n d^n e^n | n \geq 1\}$ is not a tree-adjoining language.

We have seen in Section 2.2 that the derivation in TAG is also a tree. For a given

TAG, G, it can be shown that the set of derivations trees of G, $D(G)$, is recognizable (in fact a local set). Many different grammar formalisms have been shown to be equivalent to TAG, their derivation trees are also local sets.

3 Lexicalized Grammars

We define a notion of "lexicalized grammars" that is of both linguistic and formal significance. We then show how TAG arises in the processes of lexicalizing context-free grammars.

In this "lexicalized grammar" approach (Schabes et al., 1988; Schabes, 1990), each elementary structure is systematically associated with a lexical item called the *anchor*. By 'lexicalized' we mean that in each structure there is a lexical item that is realized. The 'grammar' consists of a lexicon where each lexical item is associated with a finite number of structures for which that item is the anchor. There are operations which tell us how these structures are composed. A grammar of this form will be said to be 'lexicalized'.

Definition 2 (Lexicalized Grammar) A grammar is 'lexicalized' if it consists of:

- a finite set of structures each associated with a lexical item; each lexical item will be called the *anchor* of the corresponding structure;
- an operation or operations for composing the structures.

We require that the anchor must be an overt (i.e. not the empty string) lexical item.

The *lexicon* consists of a finite set of structures each associated with an anchor. The structures defined by the lexicon are called *elementary structures*. Structures built by combination of others are called *derived structures*.

As part of our definition of lexicalized grammars, we require that the structures be of finite size. We also require that the combining operations combine a finite set of structures into a finite number of structures. We will consider operations that combine two structures to form one derived structure.

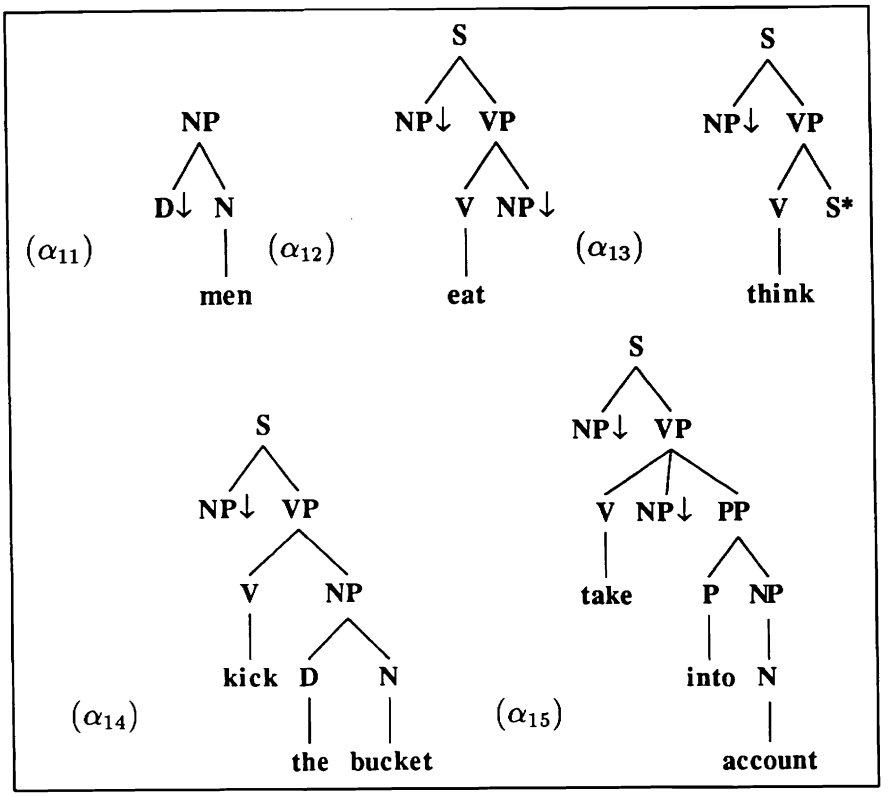
Other constraints can be put on the operations. For examples, the operations could be restricted not to copy, erase or restructure unbounded components of their arguments. We could also impose that the operations yield languages of constant growth (Joshi [1985]). The operations that we will use have these properties.

Categorial Grammars (Lambek, 1958; Steedman, 1987) are lexicalized according to our definition since each basic category has a lexical item associated with it.

As in Categorial Grammars, we say that the *category* of a word is the entire structure it selects. If a structure is associated with an anchor, we say that the entire structure is the *category* structure of the anchor.

We also use the term 'lexicalized' when speaking about structures. We say that a structure is *lexicalized* if there is at least one overt lexical item that appears in it. If more than one lexical item appears, either one lexical item is designated as the anchor or a subset of the lexical items local to the structure are designated as *multi-component anchor*. A grammar consisting of only lexicalized structures is of course lexicalized.

For example, the following structures are lexicalized according to our definition:[7]



Some simple properties follow immediately from the definition of lexicalized grammars.

Proposition 1 *Lexicalized grammars are finitely ambiguous.*

A grammar is said to be finitely ambiguous if there is no sentence of finite length that can be analyzed in an infinite number of ways.

The fact that lexicalized grammars are finitely ambiguous can be seen by considering an arbitrary sentence of finite length. The set of structures anchored by the words in the input sentence consists of a set of structures necessary to analyze the sentence; while any other structure introduces lexical items not present in the input string. Since the set of selected structures is finite, these structures can be combined in finitely many ways (since each tree is associated with at least one lexical item and since structures can combine to produce finitely many structures). Therefore lexicalized grammars are finitely ambiguous.

Since a sentence of finite length can only be finitely ambiguous,the search space used for analysis is finite. Therefore, the recognition problem for lexicalized grammars is decidable

Proposition 2 *It is decidable whether or not a string is accepted by a lexicalized grammar.*[8]

[7]The interpretation of the annotations on these structures is not relevant now and it will be given later.

[8]Assuming that one can compute the result of the combination operations.

Having stated the basic definition of lexicalized grammars and also some simple properties, we now turn our attention to one of the major issues: can context-free grammars be lexicalized?

Not every grammar is in a lexicalized form. Given a grammar G stated in a formalism, we will try find another grammar G_{lex} (not necessarily stated in the same formalism) that generates the same language and also the same tree set as G and for which the lexicalized property holds. We refer to this process as lexicalization of a grammar.

Definition 3 (Lexicalization) We say that a formalism F can be lexicalized by another formalism F', if for any finitely ambiguous grammar G in F there is a grammar G' in F' such that G' is a lexicalized grammar and such that G and G' generate the same tree set (and a fortiori the same language).

The next section discusses what it means to lexicalize a grammar. We will investigate the conditions under which such a 'lexicalization' is possible for CFGs and tree-adjoining grammars (TAGs). We present a method to lexicalize grammars such as CFGs, while keeping the rules in their full generality. We then show how a lexicalized grammar naturally follows from the extended domain of locality of TAGs.

4 'Lexicalization' of CFGs

Our definition of lexicalized grammars implies their being finitely ambiguous. Therefore a necessary condition of lexicalization of a CFG is that it is finitely ambiguous. As a consequence, recursive chain rules obtained by derivation (such as $X \stackrel{*}{\Rightarrow} X$) or elementary (such as $X \rightarrow X$) are disallowed since they generate infinitely ambiguous branches without introducing lexical items.

In general, a CFG will not be in lexicalized form. For example a rule of the form, $S \rightarrow NP\ VP$ or $S \rightarrow S\ S$, is not lexicalized since no lexical item appears on the right hand side of the rule.

A lexicalized CFG would be one for which each production rule has a terminal symbol on its right hand side. These constitute the structures associated with the lexical anchors. The combining operation is the standard substitution operation.[9]

Lexicalization of CFG that is achieved by transforming it into an equivalent Greibach Normal Form CFG, can be regarded as a *weak* lexicalization, because it does not give us the same set of trees as the original CFG.[10] Our notion of lexicalization can be regarded as *strong* lexicalization.

In the next sections, we propose to extend the domain of locality of context-free grammar in order to make lexical item appear local to the production rules. The domain of

[9]Variables are independently substituted by substitution. This standard (or first order) substitution contrasts with more powerful versions of substitution which allow to substitute multiple occurrences of the same variable by the same term.

[10]To our knowledge, there is no known tree transducer that transforms the derivation of a context-free grammar transformed in Greibach normal form to its original derivation. We strongly suspect that such a tree transducer can very probably be found. However, the transducer will probably be quite complex and will need to work on distant pieces since leaves of the derivation tree must be put back to higher positions in the tree.

locality of a CFG is extended by using a tree rewriting system that uses only substitution. We will see that in general, CFGs cannot be lexicalized using substitution alone, even if the domain of locality is extended to trees. Furthermore, in the cases where a CFG could be lexicalized by extending the domain of locality and using substitution alone, we will show that, in general, there is not enough freedom to choose the anchor of each structure. This is important because we want the choice of the anchor for a given structure to be determined on purely linguistic grounds. We will then show how the operation of adjunction enables us to freely 'lexicalize' CFGs.

4.1 Substitution and Lexicalization of CFGs

We already know that we need to assume that the given CFG is finitely ambiguous in order to be able to lexicalized it. We propose to extend the domain of locality of CFGs to make lexical items appear as part of the elementary structures by using a grammar on trees that uses substitution as combining operation. This tree-substitution grammar consists of a set of trees that are not restricted to be of depth one (rules of context-free grammars can be thought as trees of depth one) combined with substitution.[11]

A finite set of elementary trees that can be combined with substitution define a tree-based system that we will call a *tree substitution grammar.*

Definition 4 (Tree-Substitution Grammar)
A Tree-Substitution Grammar (TSG) consists of a quadruple (Σ, NT, I, S), where

(i) Σ is a finite set of terminal symbols;
(ii) NT is a finite set of non-terminal symbols[12]: $\Sigma \cap NT = \emptyset$;
(iii) S is a distinguished non-terminal symbol: $S \in NT$;
(iv) I is a finite set of finite trees whose interior nodes are labeled by non-terminal symbols and whose frontier nodes are labeled by terminal or non- terminal symbols. All non-terminal symbols on the frontier of the trees in I are marked for substitution. The trees in I are called *initial* trees.

We say that a tree is *derived* if it has been built from some initial tree in which initial or derived trees were substituted. A tree will be said of *type* X if its root is labeled by X. A tree is considered *completed* if its frontier is to be made only of nodes labeled by terminal symbols.

Whenever the string of labels of the nodes on the frontier of an X-type initial tree t_X is $\alpha \in (\Sigma \cup NT)^\star$, we will write: $Fr(t_X) = \alpha$.

As for TAG, the derivation in TSG is stated in the form of a tree called the *derivation tree* (see Section 2.2).

It is easy to see that the set of languages generated by this tree rewriting system is exactly the same set as context-free languages.

We now come back to the problem of lexicalizing context-free grammars. One can try to lexicalize finitely ambiguous CFGs by using tree-substitution grammars. However we will exhibit a counter example that shows that, in the general case, finitely ambiguous

[11] We assume here first order substitution meaning that all substitutions are independent.
[12] We use lower-case letters for terminal symbols and upper-case letters for non-terminal symbols.

CFGs cannot be lexicalized with a tree system that uses substitution as the only combining operation.

Proposition 3 *Finitely ambiguous context-free grammars cannot be lexicalized with a tree-substitution grammar.*

Proof[13] of Proposition 3
We show this proposition by a contradiction. Suppose that finitely ambiguous CFGs can be lexicalized with TSG. Then the following CFG can be lexicalized:[14]

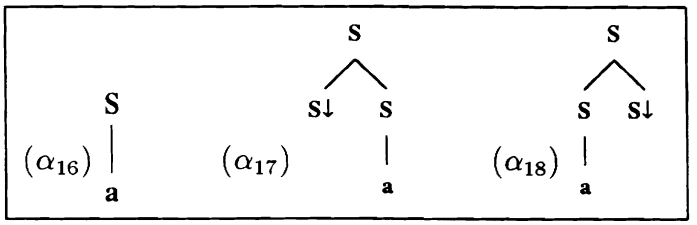
Example 3 (counter example)

$$S \rightarrow S\ S$$
$$S \rightarrow a$$

Suppose there were a lexicalized TSG G generating the same tree set as the one generated by the above grammar. Any derivation in G must start from some initial tree. Take an arbitrary initial tree t in G. Since G is a lexicalized version of the above context-free grammar, there is a node n on the frontier of t labeled by a. Since substitution can only take place on the frontier of a tree, the distance between n and the root node of t is constant in any derived tree from t. And this is the case for any initial tree t (of which there are only finitely many). This implies that in any derived tree from G there is at least one branch of bounded length from the root node to a node labeled by a (that branch cannot further expand). However in the derivation trees defined by the context-free grammar given above, a can occur arbitrarily far away from the root node of the derivation. Contradiction.
□

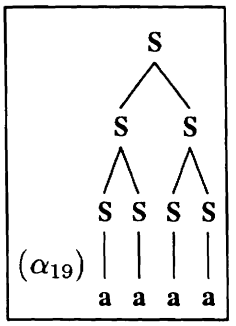
The CFG given in Example 3 cannot be lexicalized with a TSG. The difficulty is due to the fact that TSGs do not permit the distance between two nodes in the same initial tree to increase.

For example, one might think that the following TSG is a lexicalized version of the above grammar:



[13]The underlying idea behind this proof was suggested to us by Stuart Shieber.
[14]This example was pointed out to us by Fernando Pereira.

However, this lexicalized TSG does not generate all the trees generated by the context-free grammar; for example the following tree (α_{19}) cannot be generated by the above TSG:



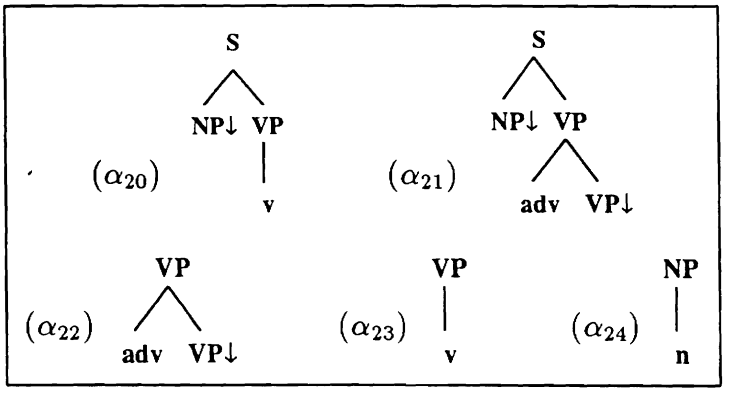
We now turn to a less formal observation. Even if some CFGs can be lexicalized by using TSG, the choice of the lexical items that emerge as the anchor may be too restrictive, for example, the choice may not be linguistically motivated.

Consider the following example:

Example 4

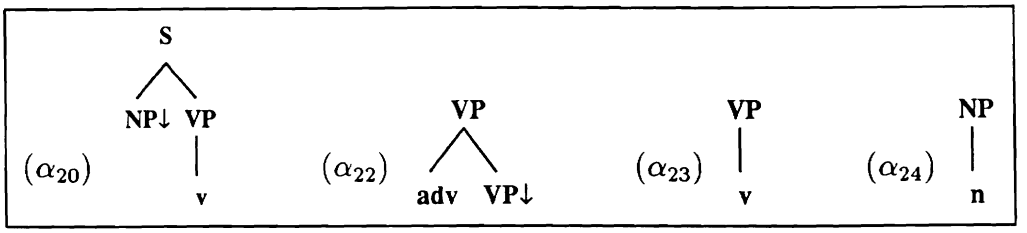
$$\begin{aligned} S &\rightarrow NP\ VP \\ VP &\rightarrow adv\ VP \\ VP &\rightarrow v \\ NP &\rightarrow n \end{aligned}$$

The grammar can be lexicalized as follows:

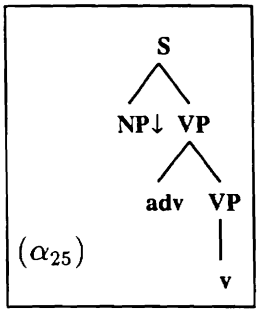


This tree-substitution grammar generates exactly the same set of trees as in Example 4, however, in this lexicalization one is forced to choose *adv* (or *n*) as the anchor of a structure rooted by S (α_{21}), and it cannot be avoided. This choice is not linguistically motivated. If one tried not to have an S-type initial tree anchored by n or by *adv*, recursion on the VP node would be inhibited.

For example, the grammar written below:



does not generate the tree α_{25}:



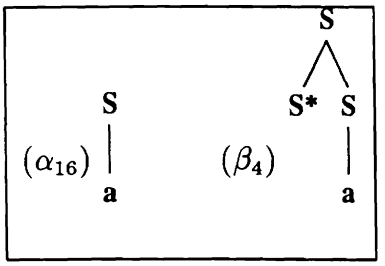
This example shows that even when it is possible to lexicalize a CFG, substitution (TSG) alone does not allow us to freely choose the lexical anchors. Substitution alone forces us to make choices of anchors that might not be linguistically (syntactically or semantically) justified. From the proof of proposition 3 we conclude that a tree based system that can lexicalize context-free grammars must permit the distance between two nodes in the same tree to be increased during a derivation. In the next section, we suggest the use of an additional operation when defining a tree-based system in which one tries to lexicalize CFGs.

4.2 Lexicalization of CFGs with TAGs

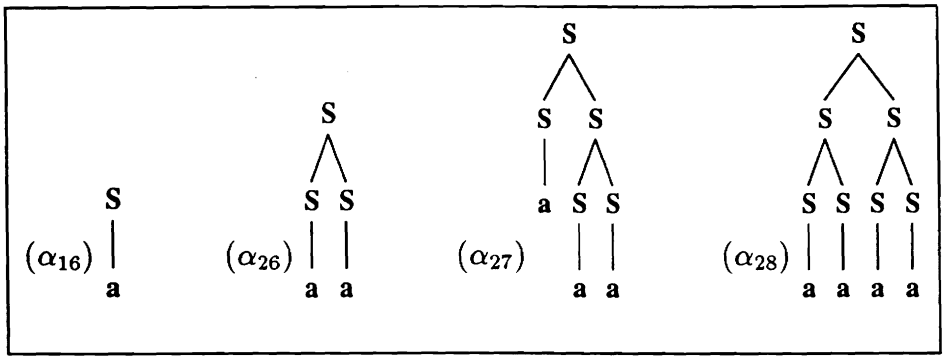
Another combining operation is needed to lexicalize finitely ambiguous CFGs. As the previous examples suggest us, we need an operation that is capable of inserting a tree inside another one. We suggest using adjunction as an additional combining operation. A tree-based system that uses substitution and adjunction coincides with a tree-adjoining grammar (TAG).

We first show that the CFGs in examples 3 and 4 for which TSG failed can be lexicalized within TAGs.

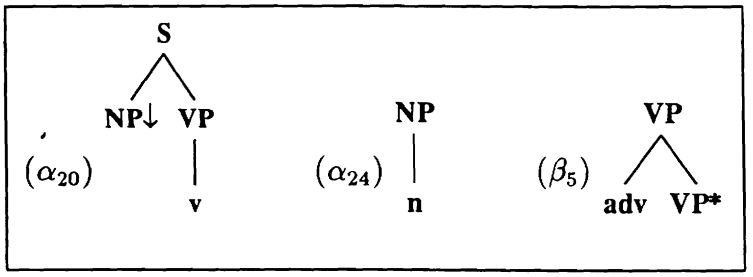
Example 5 Example 3 could not be lexicalized with TSG. It can be lexicalized by using adjunction as follows: [15]



The auxiliary tree β_4 can now be inserted by adjunction inside the derived trees. For example, the following derived trees can be derived by successive adjunction of β_4:



Example 6 The CFG given in Example 4 can be lexicalized by using adjunction and one can choose the anchor freely:[16]

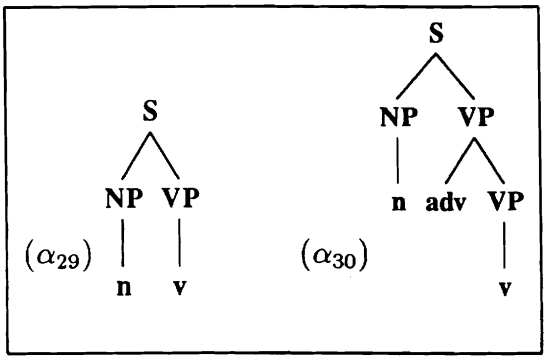


The auxiliary tree β_5 can be inserted in α_{20} at the VP node by adjunction. Using adjunction one is thus able to choose the appropriate lexical item as anchor. The following trees (α_{29} and α_{30}) can be derived by substitution of α_{24} into α_{20} for the NP node and

[15] a is taken as the lexical anchor of both the initial tree α_{16} and the auxiliary tree β_4.

[16] We chose v as the lexical anchor of α_{20} but, formally, we could have chosen n instead.

by adjunction of β_5 on the VP node in α_{20}:



We are now ready to prove the main result: any finitely ambiguous context-free grammar can be lexicalized within tree-adjoining grammars; furthermore adjunction is the only operation needed. Substitution as an additional operation enables one to lexicalize CFGs in a more compact way.

Proposition 4 *If $G = (\Sigma, NT, P, S)$ is a finitely ambiguous CFG which does not generate the empty string, then there is a lexicalized tree-adjoining grammar $G_{lex} = (\Sigma, NT, I, A, S)$ generating the same language and tree set as G. Furthermore G_{lex} can be chosen to have no substitution nodes in any elementary trees.*

We give a constructive proof of this proposition. Given an arbitrary CFG, G, we construct a lexicalized TAG, G_{lex}, that generates the same language and tree set as G. The construction is not optimal with respect to time or the number of trees but it does satisfy the requirements.

The idea is to separate the recursive part of the grammar G from the non-recursive part. The non-recursive part generates a finite number of trees, and we will take those trees as initial TAG trees. Whenever there is in G a recursion of the form $B \stackrel{*}{\Rightarrow} \alpha B \beta$, we will create an B-type auxiliary tree in which α and β are expanded in all possible ways by the non-recursive part of the grammar. Since the grammar is finitely ambiguous and since $\lambda \notin L(G)$, we are guaranteed that $\alpha\beta$ derives some lexical item within the non-recursive part of the grammar. The proof follows.

Proof of Proposition 4

Let $G = (\Sigma, NT, P, S)$ be a finitely ambiguous context-free grammar s.t. $\lambda \notin L(G)$. We say that $B \in NT$ is a *recursive symbol* if and only if $\exists \alpha, \beta \in (\Sigma \cup NT)^\star$ s.t. $B \stackrel{*}{\Rightarrow} \alpha B \beta$. We say that a production rule $B \rightarrow \delta$ is recursive whenever B is recursive.

The set of production rules of G can be partitioned into two sets: the set of recursive production rules, say $R \subseteq P$, and the set of non-recursive production rules, say $NR \subseteq P$; $R \cup NR = P$ and $R \cap NR = \emptyset$. In order to determine whether a production is recursive, given G, we construct a directed graph $\mathcal{G}$ whose nodes are labeled by non-terminal symbols and whose arcs are labeled by production rules. There is an arc labeled by $p \in P$ from a node labeled by B to a node labeled by C whenever p is of the form $B \rightarrow \alpha C \beta$, where $\alpha, \beta \in (\Sigma \cup NT)^\star$. Then, a symbol B is recursive if the node labeled by B in $\mathcal{G}$ belongs

to a cycle. A production is recursive if there is an arc labeled by the production which belongs to a cycle.

Let $L(NR) = \{w | S \stackrel{*}{\Rightarrow} w$ using only production rules in $NR\}$. $L(NR)$ is a finite set. Since $\lambda \notin L(G)$, $\lambda \notin L(NR)$. Let I be the set of all derivation trees defined by $L(NR)$. I is a finite set of trees; the trees in I have at least one terminal symbol on the frontier since the empty string is not part of the language. I will be the set of initial trees of the lexicalized TAG G_{lex}.

We then form a base of minimal cycles of $\mathcal{G}$. Classical algorithms on graphs gives us methods to find a finite set of so-called 'base-cycles' such that any cycle is a combination of those cycles and such that they do not have any sub-cycle. Let $\{c_1 \cdots c_k\}$ be a base of cycles of $\mathcal{G}$ (each c_i is a cycle of $\mathcal{G}$).

We initialize the set of auxiliary trees of G_{lex} to the empty set, i.e. $A := \emptyset$. We repeat the following procedure for all cycles c_i in the base until no more trees can be added to A.

For all nodes n_i in c_i, let B_i be the label of n_i,
  According to c_i, $B_i \stackrel{*}{\Rightarrow} \alpha_i B_i \beta_i$,
  If B_i is the label of a node in a tree in $I \cup A$ then
    for all derivations $\alpha_i \stackrel{*}{\Rightarrow} w_i \in \Sigma^\star$, $\beta_i \stackrel{*}{\Rightarrow} z_i \in \Sigma^\star$
    that use only non-recursive production rules
      add to A the auxiliary tree corresponding to all derivations:
      $B_i \stackrel{*}{\Rightarrow} \alpha_i B_i \beta_i \stackrel{*}{\Rightarrow} w_i B_i z_i$ where the node labeled B_i on the frontier is the foot node.

In this procedure, we are guaranteed that the auxiliary trees have at least one lexical item on the frontier, because $\alpha_i \beta_i$ must always derive some terminal symbol otherwise, the derivation $B_i \stackrel{*}{\Rightarrow} \alpha_i B_i \beta_i$ would derive a rule of the form $B_i \stackrel{*}{\Rightarrow} B_i$ and the grammar would be infinitely ambiguous.

It is clear that G_{lex} generates exactly the same tree set as G. Furthermore G_{lex} is lexicalized.

□

We just showed that adjunction is sufficient to lexicalize context-free grammars. However, the use of substitution as an additional operation to adjunction enables one to lexicalize a grammar with a more compact TAG.

5 Closure of TAGs under Lexicalization

In the previous section, we showed that context-free grammars can be lexicalized within tree-adjoining grammars. Only adjunction is necessary but the addition of substitution gives us the possibility to have a more compact representation of the lexicalized grammar. We now ask ourselves if TAGs are closed under lexicalization: given a finitely ambiguous TAG, G, ($\lambda \notin L(G)$), is there a lexicalized TAG, G_{lex}, which generates the same language and the same tree set as G? The answer is yes. We therefore establish that TAGs are closed under lexicalization. The following proposition holds:

Proposition 5 (TAGs are closed under lexicalization)
If G is a finitely ambiguous TAG that uses substitution and adjunction as combining operation, s.t. $\lambda \notin L(G)$, then there exists a lexicalized TAG G_{lex} which generates the same language and the same tree set as G.

The proof of this proposition is similar to the proof of proposition 4 and we only give a sketch of it. It consists of separating the recursive part of the grammar from the non-recursive part. The recursive part of the language is represented in G_{lex} by auxiliary trees. Since G is finitely ambiguous, those auxiliary trees will have at least one terminal symbol on the frontier. The non-recursive part of the grammar is encoded as initial trees. Since the empty string is not generated, those initial trees have at least one terminal symbol on the frontier. In order to determine whether an elementary tree is recursive, given G, we construct a directed graph $\mathcal{G}$ whose nodes are labeled by elementary trees and whose arcs are labeled by tree addresses. There is an arc labeled by ad from a node labeled by β to a node labeled by α whenever β can operate (by adjunction or substitution) at address ad in α. Then, an elementary tree δ is recursive if the node labeled by δ in $\mathcal{G}$ belongs to a cycle. The construction of the lexicalized TAG is then similar to the one proposed for proposition 4.

6 Conclusion[17]

The elementary objects manipulated by a tree-adjoining grammar are trees, i.e., structured objects and not strings. The properties of TAGs relate directly to the strong generative capacity (structural description) which is more relevant to linguistic descriptions than the weak generative capacity (set of strings). The tree sets of TAGs are not recognizable sets but are equivalent to the tree sets of linear indexed languages. Hence, tree-adjoining grammars generate some context-sensitive languages. However, tree-adjoining languages are strictly contained in the class of indexed languages.

The lexicalization of grammar formalisms is of linguistic and formal interest. We have taken the point of view that rules should not be separated totally from their lexical realization. In this "lexicalized" approach, each elementary structure is systematically associated with a lexical anchor. These structures specify extended domains of locality (as compared to Context Free Grammars) over which constraints can be stated.

The process of lexicalization of context-free rules forces us to use operations for combining structures that make the formalism fall in the class of mildly context sensitive languages. Substitution and adjunction give us the freedom to lexicalize CFGs. Elementary structures of extended domain of locality, when they are combined with substitution and adjunction, yield Lexicalized TAGs. TAGs were so far introduced as an independent formal system. We have shown that they derive from the lexicalization process of context-free grammars. We also have shown that TAGs are closed under lexicalization.

It is still an open problem whether or not adjoining is the 'minimal' operation needed for lexicalizing CFGs, i.e., whether there exists a tree-gluing operation say Φ such that substitution and Φ can lexicalize any CFG, and such that the tree sets of the tree system with substitution and Φ are properly contained in the tree sets of TAG.

Bibliography

Anne Abeillé, Kathleen M. Bishop, Sharon Cote, and Yves Schabes. 1990. A lexicalized tree adjoining grammar for english. Technical Report MS-CIS-90-24, Department of Computer and Information Science, University of Pennsylvania.

Anne Abeillé. 1988. Parsing french with tree adjoining grammar: some linguistic accounts. In *Proceedings of the* 12^{th} *International Conference on Computational Linguistics (COLING'88)*, Budapest, August.

N. Chomsky. 1981. *Lectures on Government and Binding.* Foris, Dordrecht.

G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag. 1985. *Generalized Phrase Structure Grammars.* Blackwell Publishing, Oxford. Also published by Harvard University Press, Cambridge, MA.

[17]There are several important papers about TAGs describing their linguistic and formal properties. Some of these are: Joshi (1987), Joshi, Vijay-Shanker and Weir (forthcoming 1991), Vijay-Shanker (1987), Weir (1988), Schabes (1990; 1991), Schabes and Joshi (1988; 1989), Kroch (1987), Kroch and Joshi (1985), Abeillé, Bishop, Cote and Schabes (1990), Abeillé (1988). A reader interested in TAGs will find these papers very useful.

Maurice Gross. 1984. Lexicon-grammar and the syntactic analysis of french. In *Proceedings of the* 10^{th} *International Conference on Computational Linguistics (COLING'84)*, Stanford, 2-6 July.

Aravind K. Joshi, L. S. Levy, and M. Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1).

Aravind K. Joshi, K. Vijay-Shanker, and David Weir. forthcoming, 1991. The convergence of mildly context-sensitive grammatical formalisms. In Peter Sells, Stuart Shieber, and Tom Wasow, editors, *Foundational Issues in Natual Language Processing.* MIT Press, Cambridge MA.

Aravind K. Joshi. 1985. How much context-sensitivity is necessary for characterizing structural descriptions—Tree Adjoining Grammars. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing—Theoretical, Computational and Psychological Perspectives.* Cambridge University Press, New York. Originally presented in a Workshop on Natural Language Parsing at Ohio State University, Columbus, Ohio, May 1983.

Aravind K. Joshi. 1987. An Introduction to Tree Adjoining Grammars. In A. Manaster-Ramer, editor, *Mathematics of Language.* John Benjamins, Amsterdam.

R. Kaplan and J. Bresnan. 1983. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge MA.

Lauri Karttunen. 1986. Radical lexicalism. Technical Report CSLI-86-68, CSLI, Stanford University. Also in *Alternative Conceptions of Phrase Structure*, University of Chicago Press, Baltin, M. and Kroch A., Chicago, 1989.

Anthony Kroch and Aravind K. Joshi. 1985. Linguistic relevance of tree adjoining grammars. Technical Report MS-CIS-85-18, Department of Computer and Information Science, University of Pennsylvania, April.

Anthony Kroch. 1987. Unbounded dependencies and subjacency in a tree adjoining grammar. In A. Manaster-Ramer, editor, *Mathematics of Language.* John Benjamins, Amsterdam.

Joachim Lambek. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170.

Carl Pollard and Ivan A. Sag. 1987. *Information-Based Syntax and Semantics. Vol 1: Fundamentals.* CSLI.

Yves Schabes and Aravind K. Joshi. 1988. An Earley-type parsing algorithm for Tree Adjoining Grammars. In 26^{th} *Meeting of the Association for Computational Linguistics (ACL'88)*, Buffalo, June.

Yves Schabes and Aravind K. Joshi. 1989. The relevance of lexicalization to parsing. In *Proceedings of the International Workshop on Parsing Technologies*, Pittsburgh, August. To also appear under the title *Parsing with Lexicalized Tree adjoining Grammar* in *Current Issues in Parsing Technologies*, MIT Press.

Yves Schabes, Anne Abeillé, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the* 12^{th} *International Conference on Computational Linguistics (COLING'88)*, Budapest, Hungary, August.

Yves Schabes. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars.* Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, August. Available as technical report (MS-CIS-90-48, LINC LAB179) from the Department of Computer Science.

Yves Schabes. 1991. The valid prefix property and left to right parsing of tree-adjoining grammar. In *Proceedings of the second International Workshop on Parsing Technologies*, Cancun, Mexico, February.

K. M. Schimpf and J. H. Gallier. 1985. Tree pushdown automata. *Journal of Computer and System Sciences*, 30:25–39.

Mark Steedman. 1987. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5:403–439.

J. W. Thatcher. 1971. Characterizing derivations trees of context free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, 5:365–396.

K. Vijay-Shanker. 1987. *A Study of Tree Adjoining Grammars.* Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

David J. Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms.* Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.