August 1988

Parsing Strategies With 'Lexicalized' Grammars: Application to Tree Adjoining Grammars

Yves Schabes
University of Pennsylvania

Anne Abeillé
University of Paris

Aravind K. Joshi
University of Pennsylvania, joshi@cis.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-88-65.

Parsing Strategies With 'Lexicalized' Grammars: Application to Tree Adjoining Grammars

Abstract
In this paper, we present a parsing strategy that arose from the development of an Earley-type parsing algorithm for TAGs (Schabes and Joshi 1988) and from some recent linguistic work in TAGs (Abeillé: 1988a).

In our approach, each elementary structure is systematically associated with a lexical head. These structures specify extended domains of locality (as compared to a context-free grammar) over which constraints can be stated. These constraints either hold within the elementary structure itself or specify what other structures can be composed with a given elementary structure. The 'grammar' consists of a lexicon where each lexical item is associated with a finite number of structures for which that item is the head. There are no separate grammar rules. There are, of course, 'rules' which tell us how these structures are composed. A grammar of this form will be said to be 'lexicalized'.

We show that in general context-free grammars cannot be 'lexicalized'. We then show how a 'lexicalized' grammar naturally follows from the extended domain of locality of TAGs and examine briefly some of the linguistic implications of our approach.

A general parsing strategy for 'lexicalized' grammars is discussed. In the first stage, the parser selects a set of elementary structures associated with the lexical items in the input sentence, and in the second stage the sentence is parsed with respect to this set. The strategy is independent of nature of the elementary structures in the underlying grammar. However, we focus our attention on TAGs. Since the set of trees selected at the end of the first stage is not infinite, the parser can use in principle any search strategy. Thus, in particular, a top-down strategy can be used since problems due to recursive structures are eliminated.

We then explain how the Earley-type parser for TAGs can be modified to take advantage of this approach.

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-88-65.

PARSING STRATEGIES WITH 'LEXICALIZED' GRAMMARS: APPLICATIONS TO TREE ADJOINING GRAMMARS

Yves Schabes, Anne Abeille
and Aravind K. Joshi

MS-CIS-88-65
LINC LAB 126

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

August 1988

Revised version of a paper in COLING 88 (International Conference on Computational Linguistics), Budapest, Hungary, August 1988

Acknowledgements: This research was supported in part by DARPA grant NOOO14-85-K-0018, NSF grants MCS-8219196-CER, IRI84-10413-AO2 and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars *

Yves SCHABES, Anne ABEILLE**and Aravind K. JOSHI

Department of Computer and Information Science
University of Pennsylvania
Philadelphia PA 19104-6389 USA
schabes@linc.cis.upenn.edu abeille@cis.upenn.edu joshi@cis.upenn.edu

ABSTRACT

In this paper, we present a parsing strategy that arose from the development of an Earley-type parsing algorithm for TAGs (Schabes and Joshi 1988) and from some recent linguistic work in TAGs (Abeillé 1988a).

In our approach, each elementary structure is systematically associated with a lexical head. These structures specify extended domains of locality (as compared to a context-free grammar) over which constraints can be stated. These constraints either hold within the elementary structure itself or specify what other structures can be composed with a given elementary structure. The 'grammar' consists of a lexicon where each lexical item is associated with a finite number of structures for which that item is the head. There are no separate grammar rules. There are, of course, 'rules' which tell us how these structures are composed. A grammar of this form will be said to be 'lexicalized'.

We show that in general context-free grammars cannot be 'lexicalized'. We then show how a 'lexicalized' grammar naturally follows from the extended domain of locality of TAGs and examine briefly some of the linguistic implications of our approach.

A general parsing strategy for 'lexicalized' grammars is discussed. In the first stage, the parser selects a set of elementary structures associated with the lexical items in the input sentence, and in the second stage the sentence is parsed with respect to this set. The strategy is independent of nature of the elementary structures in the underlying grammar. However, we focus our attention on TAGs. Since the set of trees selected at the end of the first stage is not infinite, the parser can use in principle any search strategy. Thus, in particular, a top-down strategy can be used since problems due to recursive structures are eliminated.

We then explain how the Earley-type parser for TAGs can be modified to take advantage of this approach.

*This work is partially supported by ARO grant DAA29-84-9-007, DARPA grant N0014-85-K0018, NSF grants MCS-82-191169 and DCR-84-10413. The second author is also partially supported by J. W. Zellidja grant. We have benefitted immensely from our discussions with Mitch Marcus. We want to thank Anthony Kroch, Fernando Pereira and William Rounds, and also Kathy Bishop, Robert Frank, Ellen Hays, Remo Pareschi and Ramesh Subrahmanyam for their valuable comments.

**Visiting from University of Paris VII.

1 'Lexicalization' of grammar formalisms

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 both in amount and complexity: for example, lexical rules in LFG (Kaplan and Bresnan, 1983), GPSG (Gazdar, Klein, Pullum and Sag, 1985), HPSG (Pollard and Sag, 1987), Combinatory Categorial Grammars (Steedman 1985, 1988), Karttunen's version of Categorial Grammar (Karttunen 1986, 1988), some versions of GB theory (Chomsky 1981), and Lexicon-Grammars (Gross 1984).

In this paper, we will discuss what it means for 'lexicalizing' a grammar. We present a method to 'lexicalize' grammars such as CFGs, while keeping the rules in their full generality.[1]

We say that a grammar is 'lexicalized' if it consists of: [2]

- a finite set of structures associated with each lexical item, which is intended to be the head of these structures;
- an operation or operations for composing the structures. The finite set of structures define the domain of locality over which constraints are specified, and these are local with respect to their lexical heads.

Not every grammar in a given form is in a 'lexicalized' form.[3] For example, a CFG, in general, will not be in a 'lexicalized' form. However, if we extend its domain of locality, it can be 'lexicalized' in specific cases. We require that the 'lexicalized' grammar produce not only the same language as the original grammar, but also the same structures (or tree set).[4]

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

2 'Lexicalization' of CFGs

The domain of locality of CFGs can be easily extended by using a tree rewriting grammar. This tree rewriting grammar consists of a set of trees that are not restricted to be of depth one (as in CFGs). It uses only substitution as a combining operation. Substitution can take place only on non-terminal nodes of the frontier of each tree. The language is defined to be the set of strings on the frontier of trees whose roots are labeled by a distinguished symbol S. It is easy to see that the set of languages generated by this tree rewriting grammar is exactly the same set as context-free languages.

One can try to 'lexicalize' CFGs by using this tree rewriting grammar.[5] However, in the general

[1] We plan to discuss the topic of 'lexicalized' grammars in much more detail in a forthcoming paper.

[2] By 'lexicalization' we mean that in each structure there is a lexical item that is realized. We do not mean just adding features (such as head) and unification equations to the rules of the formalism.

[3] Notice the similarity of the definition of 'lexicalized' grammar with the offline parsibility constraint (Kaplan and Bresnan 1983). As a consequence of our definition, each structure has at least one lexical item (its head) attached to it. Furthermore, we assume that an input sentence cannot be infinitely ambiguous. As a consequence, recursive chain CF rules (such as $X \rightarrow Y, Y \rightarrow X$) are disallowed.

[4] Categorial grammars (as used for example by Ades and Steedman, 1982 and Steedman, 1985 and 1988) are 'lexicalized' according to our definition. However, they do not correspond in a simple way to a rule-based system that could be used for top-down recognition or for generation.

[5] Note that a CFG in Greibach normal form can be 'lexicalized' trivially. But since the Greibach normal form of a given CFG will not necessarily generate the same tree set as the original grammar, it cannot be used as a general method for 'lexicalization'.

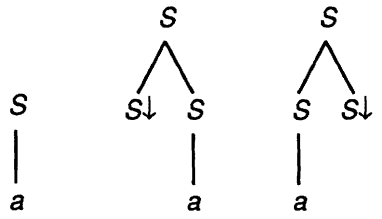
case, CFGs cannot be 'lexicalized', **if only substitution is used**. One necessary condition for the 'lexicalization' of a CFG is the absence of recursive chain rules (such as $X \rightarrow Y, Y \rightarrow X$), since recursive chain rules introduce unbounded structures with no lexical items attached to them. Even if this condition is satisfied, the following example shows that, in general, CFGs can not be 'lexicalized':

Example 1:[6]

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

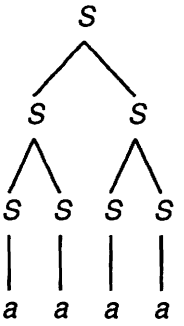
This grammar cannot be 'lexicalized'. The difficulty is due to the fact that if we expand the first rule, then as soon as the lexical item a is introduced, the recursion is inhibited. The set of structures generated by the CFG given in Example 1 cannot be generated by a tree rewriting grammar that uses substitution only. This can be seen in the following grammar ($1a$) which one might think was a lexicalized version of the above grammar: [7]

($1a$)



However, this lexicalized grammar does not generate all the trees generated by the grammar in example 1; for example the tree :

($1b$)



cannot be generated by the 'lexicalized' version.

Even if some CFGs can be 'lexicalized' by using trees, it might force a lexical item to emerge as the 'head' when, in fact, the choice is linguistically totally unmotivated.

[6] This example was pointed out to us by Fernando Pereira.

[7] We use the convention of marking substitution nodes by a down arrow (↓).

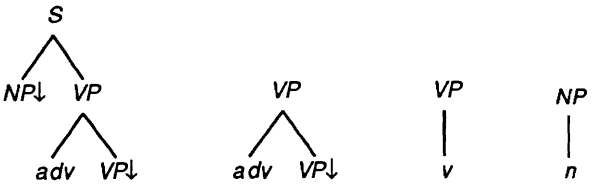
Consider the following example:

Example 2:

$$S \rightarrow NP\ VP$$
$$VP \rightarrow adv\ VP$$
$$VP \rightarrow v$$
$$NP \rightarrow n$$

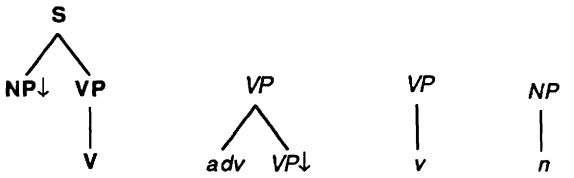
The grammar can be 'lexicalized' as follows:

(2a)



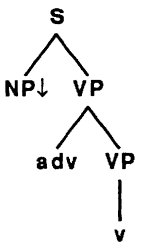
This grammar generates exactly the same set of trees as in Example 2 (using substitution only), however, in this 'lexicalization' one is forced to choose *adv* as the head of the structure given in the first tree. It is not possible to choose the verb v as the head of this structure. If one tried to do so, recursion on the substitution of the VP node would be inhibited. This can be seen as follows:

(2b)



This grammar would not generate:

(2c)



This example shows that although it is possible to 'lexicalize' some CFGs, **substitution alone does not allow us to freely choose the lexical heads**. Substitution alone forces us to make choices of heads that might not be linguistically justified.

Tree adjoining grammars (TAGs) are also a tree-based system. However, the major composition operation in TAGs is **adjoining** or **adjunction**. It builds a new tree from an auxiliary tree β and a tree α (α is any tree, initial, auxiliary or derived by adjunction). The resulting tree is called a **derived tree**. Let α be a tree containing a node n labeled by X and let β be an auxiliary tree whose root node is also labeled by X. Then the adjunction of β to α at node n results a tree γ as shown in Figure 1. Adjunction enables us to factor recursion from local dependencies.[8]

[8] In the original TAG only adjunction is used, whereas in the TAGs presented here we use both adjunction and substitution, although adjunction is more powerful than substitution and therefore substitution can be simulated by adjunction. For the use of both operations see Abeillé 1988(a).

The full definition of TAG requires specifications of constraints on adjunction at each node of each elementary tree. These constraints are obligatory adjunction (OA), selective adjunction (SA), null adjunction (NA) and they refer to what auxiliary trees can be adjoined, if any at each node. For simplicity in our paper we will not show the constraints of adjunction. These constraints are implicit in a feature structure based TAG (Vijay-Shanker and Joshi, 1988) in the

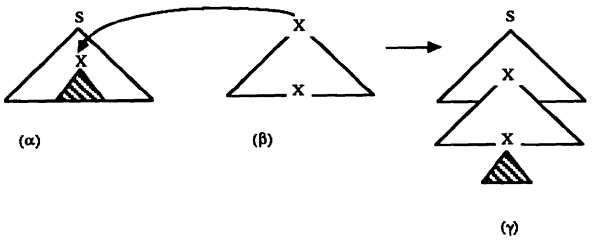
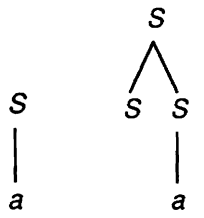


Figure 1: Adjoining

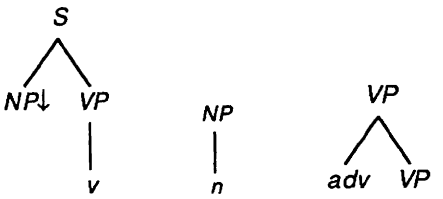
The CFG given in Example 1 can be 'lexicalized' by using adjunction as follows: [9]

(3)



The CFG given in Example 2 can be 'lexicalized' by using adjunction as follows:[10]

(4)



The auxiliary tree rooted by VP can be inserted in the first tree (rooted in S) at the VP node by adjunction. Using adjunction one is thus able to choose the appropriate lexical item as head. **This example illustrates the fact that a CFG with no recursive chain rules can be 'lexicalized' in TAGs, and that if this is done the head can be freely chosen.**

3 TAGs and 'lexicalization'

TAGs are 'naturally' lexicalized because they use an extended domain of locality. TAGs were first 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 (1985), Kroch and Joshi (1985) or Vijay-Shanker (1987). It is known that Tree Adjoining Languages (TALs) are mildly context sensitive. TALs properly contain context-free languages.[11]

Adjunction is more powerful than substitution and adjunction can simulate substitution[12]. Some

---

sense that they appear in the feature structure associated with the nodes and in the failure or success of unification during the composition.

[9] a is taken as the lexical head of both the initial tree and the auxiliary tree.

[10] We chose v as the lexical head of the first tree (rooted in S) but, formally, we could have chosen n instead (but that is not linguistically motivated).

[11] In some earlier work of Joshi (1969, 1973), the use of the two operations 'adjoining' and 'replacement' (a restricted case of substitution) was investigated both mathematically and linguistically. However, these investigations dealt with string rewriting systems and not tree rewriting systems.

[12] It is also possible to encode a context-free grammar with auxiliary trees using adjunction only. However, although

recent linguistic work in TAGs (Abeillé 1988a) has used substitution in addition to adjunction, in order to obtain appropriate structural descriptions in certain cases, such as verbs taking two sentential arguments (e.g. "John equates solving this problem with doing the impossible").[13]

We describe very briefly Tree Adjoining Grammars with adjunction and substitution.

A **Tree Adjoining Grammar** is a tree-based system that consists of two finite sets of trees: I and A . The trees in $I \cup A$ are called **elementary trees**.

The trees in I are called **initial trees**. Initial trees represent minimal linguistic structures which are defined to have at least one terminal at the frontier (the head) and all non-terminal nodes at the frontier to be filled by substitution. We call an initial tree an X-type initial tree if its root is labeled with type X. All basic categories or constituents which serve as arguments to more complex initial or auxiliary trees are X-type initial trees. A particular case is the S-type initial trees (e.g. the left tree in Figure 2). They are rooted in S, and it is a requirement of the grammar that a valid input string has to be derived from at least one S-type initial tree.

The trees in A are called **auxiliary trees**. They can represent constituents that are adjuncts to basic structures (e.g. adverbials). They can also represent basic sentential structures corresponding to verbs or predicates taking sentential complements. Auxiliary trees (e.g. the right tree in Figure 2) are characterized as follows:

- internal nodes are labeled by non-terminals;
- leaf nodes are labeled by terminals or by non-terminal nodes filled by substitution except for exactly one node (called the **foot node**) labeled by a non-terminal on which only adjunction can apply; furthermore the label of the foot node is the same as the label of the root node.

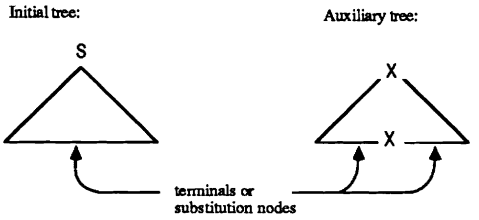


Figure 2: Schematic initial and auxiliary trees

As noted in Section 2, the major composition operation in TAGs is **adjunction**.

We define **substitution** in TAGs to take place on specified nodes on the frontiers of elementary trees. When a node is marked to be substituted, no adjunction can take place on that node. Furthermore, substitution is always mandatory. Any tree derived from a initial tree rooted by the same label as the given node can be substituted. The resulting tree is obtained by replacing the node by the derived tree. Substitution is illustrated in Figure 3. In case of substitution on a node labeled by S (sentential complement), only trees derived from S-type initial trees (therefore rooted by S) can be substituted.

We define the **tree set** of a TAG G, $\mathcal{T}(G)$ to be the set of all derived trees starting from S-type initial trees in I. Furthermore, the **string language** generated by a TAG, $\mathcal{L}(G)$, is defined to be the set of all terminal strings of the trees in $\mathcal{T}(G)$.

If we have a 'lexicalized' grammar, the parser works with a set of structures whose nature depends on the input string and whose cardinality is proportional to the length of the sentence (since we assume that the number of structures associated with a lexical item is finite). We might think of these structures as 'rules' to be used by the parser. Rules are now differentiated by their realizations

the languages correspond, the possible encoding does not directly reflect the original context-free grammar since this encoding uses adjunction.

[13] Adding substitution does not change the mathematical properties of TAGs.

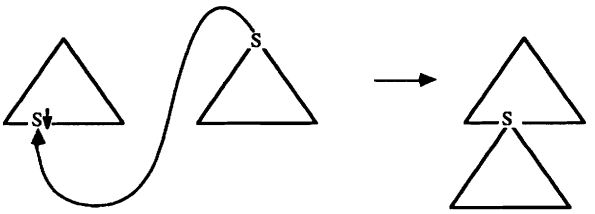


Figure 3: Mechanism of substitution

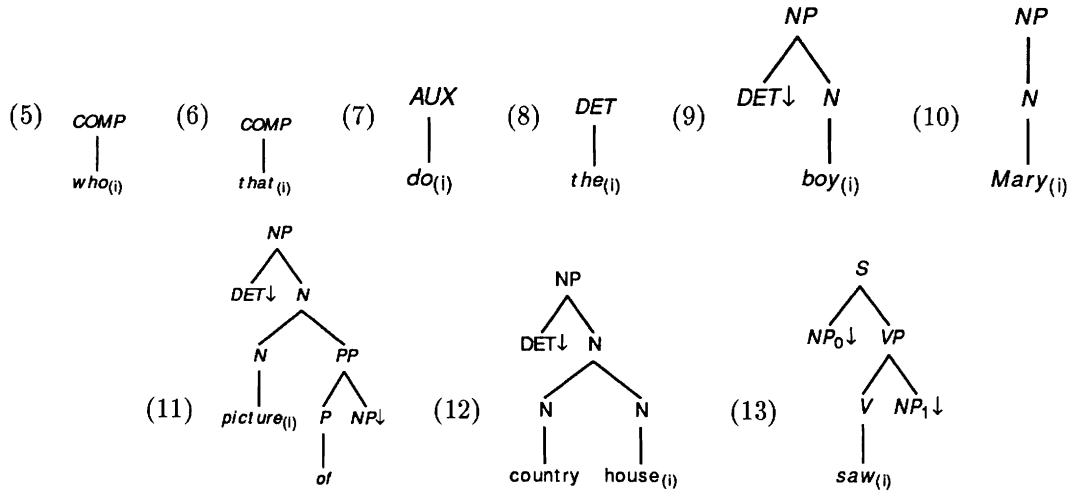
in the sentence. The number of 'rules' that can be used for a given sentence is bounded and is proportional to the length of the sentence.

The lexical item is directly associated with the structure corresponding to the parser 'rule', and such a 'rule' can only occur once. Lexical items are differentiated by their realizations in the input sentence and also by their positions in the sentence. Therefore, a given 'rule' corresponds to exactly one lexical item in the input sentence.

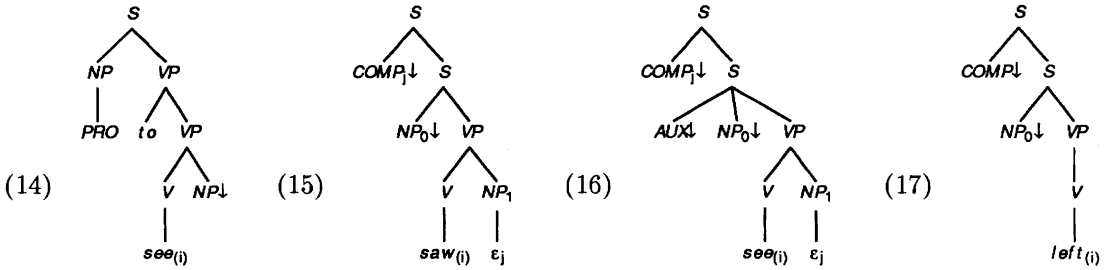
The elementary structures are projections of lexical items which serve as heads. We recall that tree structures in TAGs correspond to linguistically minimal but complete structures: the complete argument structure in the case of a predicate, the maximal projection of a category in the case of an argument or an adjunct. If a structure has only one terminal, the terminal is the head of the structure; if there are several terminals, the choice of the head for a given structure is linguistically determined, e.g. by the principles of $\overline{X}$ theory if the structure is of $\overline{X}$ type. The head of NP is N, that of AP is A. S also has to be considered as the projection of a lexical head, usually V. As is obvious, the head must always be lexically present in all of the structures it produces.

In the TAG lexicon each item is associated with a structure (or a set of structures), and that structure can be regarded as its category, linguistically speaking. Each lexical item has as many entries in the lexicon as it has possible category or argument structures. We will now give some examples of structures that appear in this lexicon.

Some examples of initial trees are (for simplicity, we have omitted the constraints associated with the nodes) :[14]

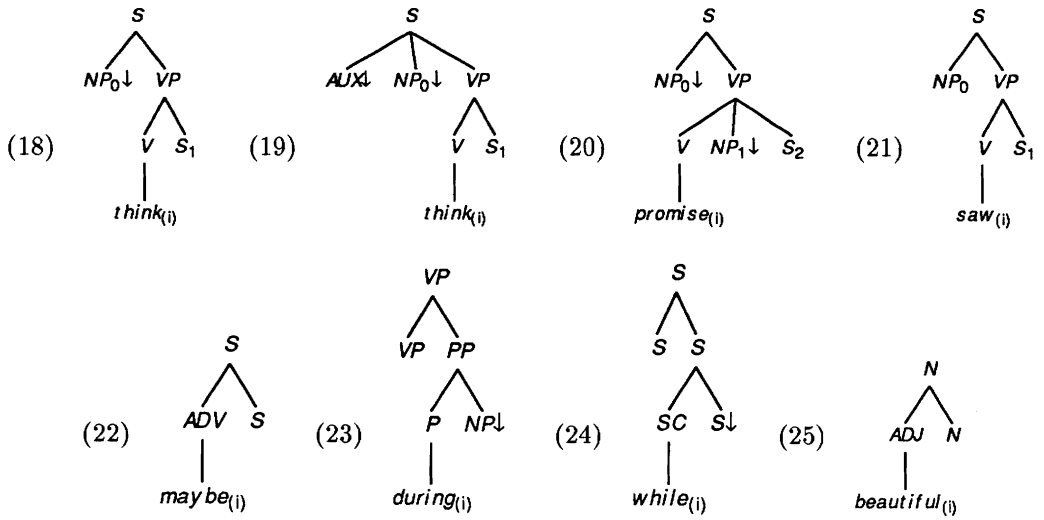


[14]The index in parentheses on a lexical item that produces the structure encodes the position of the lexical item in the string. We put indices on some non-terminals to express syntactic roles (0 for subject, 1 for first object, etc.). The index shown on the empty string (ϵ) and the corresponding filler in the same tree is for the purpose of indicating the filler-gap dependency.



For $X \neq S$, X-type initial trees correspond to the maximal projection of the category X of the head. They are reduced to a pre-terminal node in the case of simple categories such as $COMP$, AUX or DET (trees 5, 6, 7 and 8) and are expanded into more complex structures in the case of categories taking arguments (trees 9-11) and in the case of compound categories (tree 12). They correspond to the maximal projection of a category in the case of simple phrases, to the entire compound, in the case of compound categories. They correspond to trees which will be systematically substituted for one of the argument positions of one of the elementary structures. Trees 13-17 are examples of S-type initial trees: they are usually considered as projections of a verb and usually take nominal complements. The NP-type tree 'Mary' (tree 10), and the NP-type tree 'John' (similar to tree 10), for example, will be inserted by substitution in the tree 13 corresponding to 'NP_0 saw NP_1' to produce 'John saw Mary'.

Examples of auxiliary trees (they are predicates taking sentential complements or modifiers):



The auxiliary tree 18 corresponds to:

$$_0\ \texttt{Bill}\ _1\ \texttt{thought}\ _2\ S$$

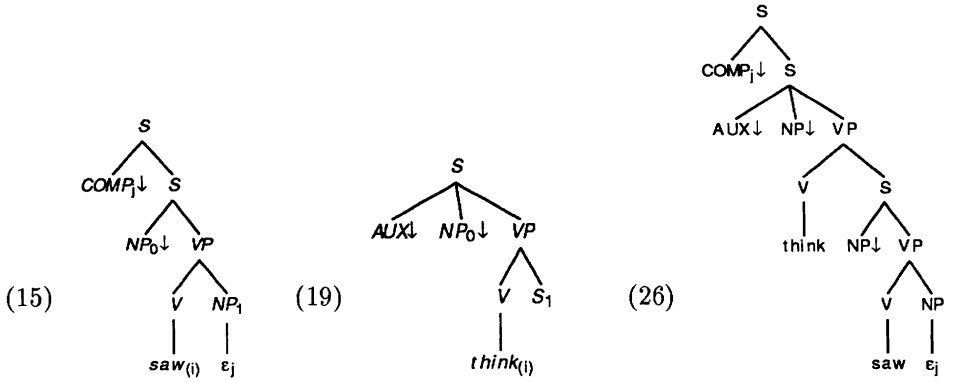
It adjoins to 'John saw Mary' (derived from tree 13), to produce: 'Bill thought John saw Mary'.

The auxiliary tree 'John promised Mary S' (tree 20) adjoins to 'PRO to see Bill' (tree 14), which is one of the initial trees selected by the item 'see' in the lexicon to produce: 'John promised Mary to see Bill'.

What are linguistically called adjuncts are represented as auxiliary trees rooted by the category of the node they are adjoined to. They are the maximal projection for a given category and are selected by a lexical head (for example, adjectives or adverbs taking complements, trees 22, 23, 24, 25

in the above figure).

In our discussion, we consider verbs, adjectives, and certain nouns, as possible predicates yielding sentences. These predicates can take either nominal or sentential arguments. All arguments are always initial trees. They enter the derivation either by being substituted into the structure of the predicate, or, in the case of sentential arguments, by having the predicate structure adjoined to them. It is this very feature that reduces long distance dependencies to localized dependencies (Kroch and Joshi, 1985): **who do you think John saw?**. The gap and the filler are part of the same elementary tree, 'who$_j$ John saw ϵ_j' (tree 15), and the tree corresponding to 'do you think' (tree 19) is adjoined in the interior node S (which will carry an obligatory adjunction constraint not shown here):



In this approach, the **argument structure** is not just a list of arguments. It is the syntactic structure constructed with the lexical value of the predicate and with all the nodes for its arguments, which eliminates the redundancy often noted between phrase structure rules and subcategorization frames. The argument structure for a predicate is its maximally projected structure.[15]

A simple case of an argument structure is a verb with its subcategorized arguments. For example, the verb **saw** (at position i) generates the following structures (among others):

$_0$ **John** $_1$ **saw** $_2$ **Mary** $_3$ ($i = 2$, tree 13)

$_0$ **John** $_1$ **saw** $_2$ **that** $_3$ **Mary** $_4$ **left** $_5$. ($i = 2$, tree 21)

An argument structure can correspond to either one or a set of syntactic surface structures. The lexical head will then produce a set of possible trees, one for **NP$_0$ saw NP$_1$** (tree 12) and another for **who$_i$ did NP$_0$ see** ϵ_i **?** (tree 15), for example. If one defines principles for building such sets of trees, these principles will correspond to syntactic rules in a derivation-based theory of grammar.

4 Parsing 'lexicalized' grammars

We assume that the input sentence is not infinite and that it cannot be syntactically infinitely ambiguous. 'Lexicalization' simplifies the task of a parser in the following sense. The first pass of the parser filters the grammar to a grammar corresponding to the input string. It also puts constraints on the way that adjunctions or substitutions can be performed since each structure has a head whose position in the input string is recorded. The 'grammar' of the parser is reduced to a set of structures whose cardinality is proportional to the length of the input sentence. Furthermore, since each rule can be used once, recursion does not lead to the usual non-termination problem.

[15] Optional arguments are stated in the structure.

Once a structure has been chosen for a given token, the other possible structures for the same token do not participate in the parse. Of course, if the sentence is ambiguous, there may be more than one choice.

If one adopts an off-line parsing algorithm, the parsing problem is reduced to the following two steps:

- In the first step the parser will select, as described in Section 3, the set of structures corresponding to each word in the sentence. Each structure can be considered as encoding a set of 'rules'.
- Then the parser tries to see whether these structures can be combined to obtain a well-formed structure. In particular, it puts the structures corresponding to arguments into the structures corresponding to predicates, and adjoins, if needed, the auxiliary structures corresponding to adjuncts to what they select (or are selected) for.

In principle, any parsing strategy can be applied in the second step, since the number of structures produced is finite, and since each of them corresponds to a token in the input string, the search space is finite and termination is guaranteed. In principle, one can proceed inside out, left to right or in any other way. Of course, standard parsing algorithms can be used, too. In particular, we can use the top-down parsing strategy without encountering the usual problems due to recursion. Problems in the prediction step of the Earley parser used for unification-based formalisms no longer exist. Because we are working with a 'lexicalized' grammar, which results in an extended domain of locality, and because the number of structures is bounded, we do not need a notion like restrictors, as has been used by Shieber (1985) in the parsing of unification-based formalisms.

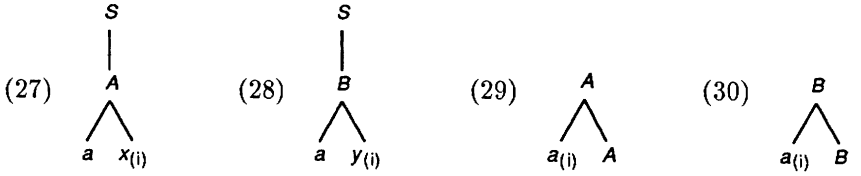
By assuming that the number of structures associated with a lexical item is finite, since each structure has a lexical item attached to it, we implicitly make the assumption that an input string of finite length cannot be syntactically infinitely ambiguous.

Since the trees are selected by the input string, the parser can use information that might be non-local to guide the search. For example, consider the language generated by the following CFG (example due to Mitch Marcus):

$$S \rightarrow A|B$$
$$A \rightarrow aA|ax$$
$$B \rightarrow aB|ay$$

This grammar generates the language:$\{a^\star x\} \cup \{a^\star y\}$. In a standard CFG parsing algorithm, As and Bs will be built until the last token in the input (x or y) is recognized. It would require unbounded look-ahead to decide which rule ($S \rightarrow A$ or $S \rightarrow B$) to choose.

One can encode the grammar in a lexicalized TAG as follows:



Suppose that the heads of the initial trees (27 and 28) are respectively x and y and that a is the head of both auxiliary trees (29 and 30). Then, if the elementary trees are selected by the input string, and if a top-down strategy is used, when the tree 27 (resp. 28) is selected, only the corresponding auxiliary tree 29 (resp. 30) will be processed by the parser (although both trees 29 and 30 will be selected). [16]

[16] This use of non-local information allows parsing of discontinuous constituents in the 'lexicalized' TAGs. They are recognized even if there are unbounded insertions between their components and even if their 'head' is the last element of the string.

After the first step, we have a grammar whose size is proportional to the length of the input string. The size of the grammar to be taken into consideration in the analysis of the parsing complexity of grammar formalisms has been reduced to an amount proportional to the length of the input. We have not yet investigated the implication of this approach for complexity analysis. Worst case analysis, in general, will not be affected. By placing some constraints on the form of the elementary trees and pursuing a variety of parsing strategies, it may be possible to improve these results. Also average case complexity might be easy to investigate in this case. However, all these are still open questions.

5 Expressing the parsing problem in a deduction system

It is possible to express the parsing problem in a terminating deduction system on trees (similar to Lambek's deduction system on categories (1958 and 1961)).[17] The grammar can be thought of as a quintuple $(V_N, \Sigma, \Theta, S, Lex)$ where:

- V_N is a finite set of non-terminal symbols,
- Σ is a finite set of alphabet symbols,
- Θ is the set of trees constructed with $\Sigma^\star$ and V_N (the elements of $\Sigma^\star$ having rank 0).
- Lex is the lexicon, i.e. a function from lexical items to finite subsets of Θ: $\Sigma^\star \rightarrow 2^\Theta(finite)$.

A sequent is defined to be of the form:

$$\tau_1, \cdots, \tau_n \longrightarrow A, \text{ where } \tau_i \in \Theta \text{ and } A \in V_N$$

Two inference rules combine two trees of the lefthand side to form a new one. One inference rule corresponds to adjunction and the other to substitution (substitution and adjunction can only be performed when the head positions are consistent with each other). Once two trees are combined, they are replaced by the resulting tree in the lefthand side of the sequent. This takes into account the fact that each elementary tree corresponds to a single lexical item in the input string. Therefore each tree can be used only once. Axioms of the system are of the form:

$$\tau \longrightarrow A$$

where τ is a 'completed' tree (i.e., a derived tree with no obligatory adjoining constraint) rooted by A.

The sequent

$$\tau_1, \cdots, \tau_n \longrightarrow A$$

is said to be provable if the sequent can be reduced (by the inference rules) to an axiom; we write:

$$\vdash \tau_1, \cdots, \tau_n \longrightarrow A$$

Since there are finitely many ways to combine a finite number of trees with each other, the system is terminating.

The language generated by such system is defined to be:

$$\mathcal{L} = \{a_1, \cdots, a_n | \exists \tau_i \in Lex(a_i), 1 \leq i \leq n, \text{ s. t. } \vdash \tau_1, \cdots, \tau_n \longrightarrow S\}$$

Also, one can state a necessary condition on the correctness of a sentence similar to the category count theorem of van Benthem (1985 and 1986).

Joshi (1988) has studied a categorial-like formulation of TAGs related to the deductive system described here. Joshi and Schabes are currently investigating a deduction system for this categorial-like formulation of TAGs which allows a more elegant combination of syntax and semantics.

6 Extending the Earley-type parser for TAGs

An Earley-type parser for TAGs has been proposed by Schabes and Joshi (1988a). It takes as input a TAG and a sentence to be parsed. It places no restrictions on the grammar. The algorithm

[17] As Categorial Grammars are naturally parsed in two steps, Pareschi (1988) also proposes a decidable deduction system for a definite clause version of Categorial Grammars.

is a bottom-up parser that uses top-down filtering. It is able to parse constraints on adjunction, substitution and feature structures for TAGs as defined by Vijay-Shanker (1987) and Vijay-Shanker and Joshi (1988). It is able to parse CFGs and TAGs directly. Thus it embeds the essential aspects of PATR-II as defined by Shieber (1984 and 1986). Its correctness was proven in Schabes and Joshi (1988b). The concepts of dotted rule and states have been extended to TAG trees. The algorithm as described by Schabes and Joshi (1988a) manipulates states of the form:

$$s = [\alpha, dot, side, pos, l, f_l, f_r, star, t_l^*, b_l^*, subst?]$$

where α is a tree, *dot* is the address of the dot in the tree, *side* is the side of the symbol the dot is on (left or right), *pos* is the position of the dot (above or below the symbol), *star* is an address in α and $l, f_l, f_r, star, t_l^*, b_l^*$ are indices of positions in the input string. The variable *subst*? is a boolean that indicates whether the tree has been predicted for substitution.

The algorithm uses nine processes:

- The **Scanner** allows lexical items to be recognized.
- **Move dot down** and **Move dot up** perform a tree traversal that allow the parser to scan the input from left to right.
- The **Left Predictor** predicts an adjunction if it is possible.
- Suppose that the auxiliary tree that we left-predicted has been recognized as far as its foot, then the **Left Completor** tries to recognize what was pushed under the foot.
- Once the subtree pushed under the foot has been recognized, the **Right Predictor** tries to recognize the other half of the auxiliary tree.
- If the auxiliary tree has been totally recognized, the **Right Completor** tries to recognize the rest of the tree in which the auxiliary tree has been adjoined.
- The **Substitution Predictor** performs the same operations as Earley's original predictor. It predicts for substitution (when appropriate) initial trees that could be substituted.
- If the tree that we predicted for substitution has been totally recognized, the **Substitution Completor** tries to recognize the rest of the tree in which we predicted a substitution.

The Earley-type parser can be extended to take advantage of the lexicon-based strategy proposed earlier. Once the input string has been scanned and the corresponding elementary trees have been selected, the parser will proceed bottom-up using the top-down filtering from the initial trees that have been selected. In order to take into account that each tree is unique and therefore can be used only once, a new component Γ is added to the states. A state is now defined to be:

$$s = [\alpha, dot, side, pos, l, f_l, f_r, star, t_l^*, b_l^*, subst?, \Gamma]$$

Γ encodes the trees corresponding to the input string that have not yet been used:

$$\Gamma = \{\{\gamma_{11}, \cdots, \gamma_{1k}\}, \cdots, \{\gamma_{m1}, \cdots, \gamma_{ml}\}\}$$

where $\{\gamma_{i1}, \cdots, \gamma_{ij}\}$ is the set of trees selected by the lexical item a_i.

The left predictor must be modified so that it predicts only trees that are in the set Γ of the given state. As soon as one tree (say γ_{iu}) is used, the entire set of trees corresponding to the same token ($\{\gamma_{i1}, \cdots, \gamma_{ij}\}$) cannot be used later on. Of course, all competitive paths are taken in parallel as in the usual Earley parser.

The way that Γ is modified by the Left Predictor is illustrated in the following figure:

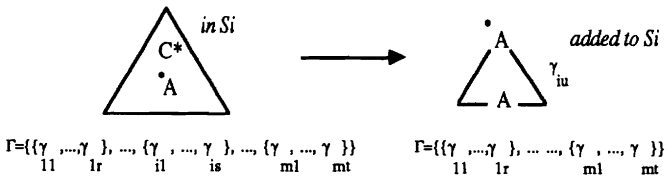


Figure 4: Update of Γ in the Left Predictor

The tree γ_{iu} is predicted and therefore the trees corresponding to the token a_i ($\{\gamma_{i1}, \cdots, \gamma_{is}\}$) are removed from Γ. Furthermore, the Left Predictor must only predict trees in which the position of the head is coherent with the state on which the Left Predictor applies. The Substitution Predictor is modified in a similar way.

The scanner must also be slightly modified since the head of the structure is differentiated not only by its lexical value but also by its position in the string.

7 Current State of Implementation

We have written two grammars, one for English (Bishop, Cote and Abeillé, 1988) and one for French (Abeillé 1988b). Each of them currently comprises more than 300 elementary trees. These trees are gathered in tree families when an element of a certain type (e.g. a verb) is said to select more than one tree. We have 25 such tree families that correspond to most of the basic argument structures of each language. A tree family consists on average of 12 trees. The other elementary trees correspond to arguments (that select initial trees) and to adjuncts (that select auxiliary trees).

The English grammar currently covers the basic argument structures, optional arguments, light verb constructions, and verb particle combinations. Wh-movement and unbounded dependencies, subjacency and some island constraint violations are also accounted for. We are in the process of adding transitivity alternations (such as dative shift or the so-called ergative alternation), PRO binding, and some word order variation. These should add at most 10 trees to each tree family.

The current size of the lexicon is approximately 1000 words for English and approximately 2000 words for French.[18] The expected size in the near future of the lexicon is approximately 4000 words for each language. The next goal for the grammar coverage will be coordination and coreference. A number of approaches for these problems have been proposed within the TAG framework and we are exploring these at present.

In addition to the Earley-type parser, we have implemented a DCG-like parser (purely top-down) for 'lexicalized' TAGs in Prolog (Symbolics Prolog and C-Prolog).

8 Conclusion

In this paper, we have presented a general parsing strategy based on 'lexicalized' grammars. We have briefly described the notion of lexicalization of a grammar. We have shown that CFGs, in general, cannot be lexicalized using substitution alone. Even when they can be lexicalized, using substitution, the lexicalization will often force linguistically unmotivated lexical items to emerge as 'heads'. We have also shown that CFGs can always be lexicalized, if one allows both substitution and adjunction, and further that we can always make the 'right' lexical items emerge as the heads. TAGs are shown to be naturally 'lexicalized'. Lexicalization of a grammar suggests a two-step parsing strategy. The first step selects the set of structures corresponding to each word in the sentence. The second step puts the argument structures into predicate structures. In the first step, structures, rather than non-terminals, are associated with lexical items. This strategy allows us to use non-local information in the input string. The 'grammar' for the parser is reduced to a set of structures whose cardinality is proportional to the length of the input sentence. Furthermore, the parsing strategy applies to any parsing algorithm; in particular top-down, without encountering problems due to recursion. It can be formalized into a terminating deduction system that has finite search space for a sentence of finite length. The Earley-type parser for TAGs has been extended to take advantage of this strategy. We have briefly described the current state of the implementation and the size of the associated grammar.

[18] The lexicalized TAGs associated with these lexicons are feature structure based TAGs (Vijay-Shanker and Joshi, 1988).

References

Abeillé, Anne, August 1988 (a). Parsing French with Tree Adjoining Grammar: some Linguistic Accounts. In *Proceedings of the* 12^{th} *International Conference on Computational Linguistics (Coling'88).* Budapest.

Abeillé, Anne, 1988 (b). *A Lexicalized Tree Adjoining Grammar for French: the General Framework.* Technical Report, Department of Computer and Information Science, University of Pennsylvania.

Ades, A. E. and Steedman, M. J., 1982. On the Order of Words. *Linguistics and Philosophy* 3:517–558.

Bishop, Kathleen M.; Cote, Sharon; and Abeillé, Anne, 1988. *A Lexicalized Tree Adjoining Grammar for English: some Basic Accounts.* Technical Report, Department of Computer and Information Science, University of Pennsylvania.

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

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

Gross, Maurice, 2-6 July 1984. Lexicon-Grammar and the Syntactic Analysis of French. In *Proceedings of the* 10^{th} *International Conference on Computational Linguistics (Coling'84).* Stanford.

Joshi, Aravind K., August 1969. Properties of Formal Grammars with Mixed Type of Rules and their Linguistic Relevance. In *Proceedings of the International Conference on Computational Linguistics.* Sanga Saby.

Joshi, Aravind K., 1973. A Class of Transformational Grammars. In M. Gross, M. Halle and Schutzenberger, M.P. (editors), *The Formal Analysis of Natural Languages.* Mouton, La Hague.

Joshi, Aravind K., 1985. How Much Context-Sensitivity is Necessary for Characterizing Structural Descriptions—Tree Adjoining Grammars. In Dowty, D.; Karttunen, L.; and Zwicky, A. (editors), *Natural Language Processing—Theoretical, Computational and Psychological Perspectives.* Cambridge University Press, New York. Originally presented in 1983.

Joshi, Aravind K., 1988. TAGs in Categorial Clothing. Unpublished manuscript, University of Pennsylvania.

Joshi, A. K.; Levy, L. S.; and Takahashi, M., 1975. Tree Adjunct Grammars. *J. Comput. Syst. Sci.* 10(1).

Kaplan, R. and Bresnan, J., 1983. Lexical-functional Grammar: A Formal System for Grammatical Representation. In Bresnan, J. (editor), *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge MA.

Karttunen, Lauri, 1986. *Radical Lexicalism.* Technical Report CSLI-86-68, CSLI, Stanford University. To also appear in *New Approaches to Phrase Structures*, University of Chicago Press, Baltin, M. and Kroch A., Chicago, 1988.

Kroch, A. and Joshi, A. K., 1985. *Linguistic Relevance of Tree Adjoining Grammars.* Technical Report MS-CIS-85-18, Department of Computer and Information Science, University of Pennsylvania.

Lambek, Joachim, 1958. The Mathematics of Sentence Structure. *American Mathematical Monthly* 65:154–170.

Lambek, Joachim, 1961. On the Calculus of Syntactic Types. In *Proceedings of the Symposium on Applied Mathematics*, pages 166–178.

Pareschi, Remo, June 1988. A Definite Clause Version of Categorial Grammar. In 26^{th} *Meeting of the Association for Computational Linguistics.* Buffalo.

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

Schabes, Yves and Joshi, Aravind K., June 1988. An Earley-Type Parsing Algorithm for Tree Adjoining Grammars. In 26^{th} *Meeting of the Association for Computational Linguistics.* Buffalo.

Schabes, Yves and Joshi, Aravind K., 1988 (b). *An Earley-type Parser for Tree Adjoining Grammars.* Technical Report MS-CIS-88-36, Department of Computer and Information Science, University of Pennsylvania.

Shieber, Stuart M., July 1984. The Design of a Computer Language for Linguistic Information. In 22^{nd} *Meeting of the Association for Computational Linguistics.* Stanford.

Shieber, Stuart M., July 1985. Using Restriction to Extend Parsing Algorithms for Complex-feature-based Formalisms. In 23^{rd} *Meeting of the Association for Computational Linguistics.* Chicago.

Shieber, Stuart M., 1986. *An Introduction to Unification-Based Approaches to Grammar.* Center for the Study of Language and Information, Stanford, CA.

Steedman, M. J., 1985. Dependency and Coordination in the Grammar of Dutch and English. *Language* 61:523–568.

Steedman, M., 1988. Combinatory Grammars and Parasitic Gaps. *To appear in Natural Language and Linguistic Theory* .

van Benthem, Johan, 1985. Lambek Calculus. Manuscript, Filosofisch Instituut, Rijks Universiteit, Groningen.

van Benthem, Johan, 1986. *Essays on Logical Semantics*, Chapter 7, pages 123–150. D. Reidel Publishing Company.

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

Vijay-Shanker, K. and Joshi, A.K., August 1988. Feature Structure Based Tree Adjoining Grammars. In *Proceedings of the* 12^{th} *International Conference on Computational Linguistics (Coling'88).* Budapest.