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 general parsing strategy that arose from the development of an Earley-type parsing algorithm for TAGs (Schabes and Joshi 1988) and from recent linguistic work in TAGs (Abeille 1988).

In our approach elementary structures are associated with their lexical heads. 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.

We state the conditions under which context-free based grammars can be 'lexicalized' without changing the linguistic structures originally produced. We argue that even if one extends the domain of locality of CFGs to trees, using only substitution does not give the freedom to choose the head of each structure. We show how adjunction allows us to 'lexicalize' a CFG freely.

We then show how a 'lexicalized' grammar naturally follows from the extended domain of locality of TAGs and present some of the linguistic advantages of our approach.

A novel general parsing strategy for 'lexicalized' grammars is discussed. In a first stage, the parser builds a set structures corresponding to the input sentence and in a second stage, the sentence is parsed with respect to this set. The strategy is independent of the linguistic theory adopted and of the underlying grammar formalism. However, we focus our attention on TAGs. Since the set of trees needed to parse an input sentence is supposed to be finite, 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. The parser is also able to use non-local information to guide the search.

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. The authors would like to thank Mitch Marcus for his helpful comments about this work. Thanks are also due to Ellen Hays.

**Visiting from University of Paris VII.

1 'Lexicalization' of grammar formalisms

Most of the current linguistics theories tend to give lexical accounts of several phenomena that used to be considered purely syntactic. The information put in the lexicon is therefore increased and complexified (e.g. lexical rules in LFG, used also by HPSG, or Gross 1984's lexicon-grammar). But the question of what it means to 'lexicalize' a grammar is seldom addressed. The possible consequences of this question for parsing are not fully investigated. We present how to 'lexicalize' grammars such as CFGs in a radical way, while possibly keeping the rules in their full generality. If one assumes that the input sentence is finite and that it cannot be syntactically infinitely ambiguous, the 'lexicalization' simplifies the task of a parser.

We say that a grammar formalism is 'lexicalized' if it consists of:

- a finite set of structures to be associated with lexical items, which usually will be heads of these structures,
- an operation or operations for composing the structures.[1] 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 formalism in a given form is in a 'lexicalized' form. For example, a CFG, in general, will not be in a 'lexicalized' form. However, by extending its domain of locality, it can be 'lexicalized'. We require that the 'lexicalized' grammar produces not only the same language as the original grammar, but also the same structures (or tree set).[2]

We propose to study 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 only uses substitution. We state the conditions under which CFGs can be 'lexicalized' without changing the structures originally produced. We argue that even if one extends the domain of locality of CFGs to trees, using only substitution does not give the freedom to choose the head of each structure. We then

[1] 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.

[2] Categorial grammars 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.

show how adjunction enables one to freely 'lexicalize' a CFG.

2 'Lexicalization' of CFGs

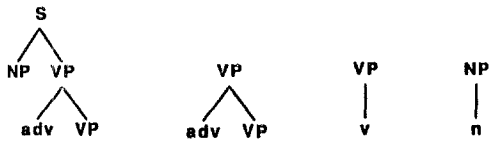
The domain of locality of CFGs can be easily extended by using a tree rewriting grammar. This tree rewriting grammar consists of a set of trees that are not restricted to be of depth one (as in CFGs). It uses only substitution as a combining operation. Substitution can take place only on non-terminal nodes of the frontier of each tree. The language is defined to be the set of strings on the frontiers 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.

If no recursive chain rules exist, it is formally possible to 'lexicalize' a CFG with this tree rewriting grammar.[3] Recursive chain rules are disallowed since they introduce unbounded structures with no lexical items attached to them.

Although a CFG can be 'lexicalized' by using trees, it is not possible to choose freely the lexical item that plays the role of the head for each structure. Consider the following example:

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

The grammar can be 'lexicalized' as follows:



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 example shows that although it is possible to 'lexicalize' CFGs, substitution alone does not allow us to freely choose the lexical heads. Substitution alone forces us to make choices that might not be syntactically and semantically 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 to factor recursion from local dependencies.

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

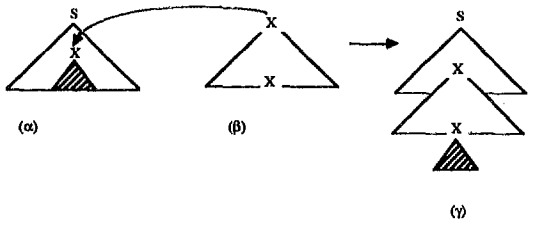
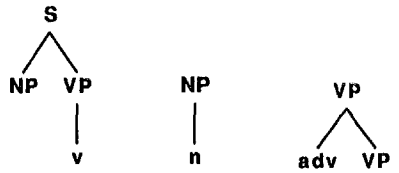


Figure 1: The mechanism of adjunction

The previous CFG can be 'lexicalized' by using adjunction as follows:[4]



The auxiliary tree rooted by VP can be inserted in the S tree on 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 that is done the head can be freely chosen.

3 TAGs and 'lexicalization'

TAGs are 'naturally' lexicalized because they used 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. 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 original context-free grammar since this encoding uses adjunction.

Although adjunction is more powerful than substitution and could be used to simulate it, in recent linguistic work in TAG (Abeillé 1988) substitution has been used 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"). Adding substitution does not change the mathematical properties of TAGs.

We describe very briefly the Tree Adjoining Grammar formalism with adjunction and substitution.

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

The trees in I are called **initial trees**. Initial trees represent basic sentential structures. They are usually considered as projections of the verb and they take nominal

[4]We chose v as lexical head of the S tree but we could have chosen n instead (although it is not motivated).

complements. Initial trees (see the left tree in Figure 2) are rooted in S and their frontier consists of terminal symbols (including the empty string) and non-terminal nodes to be substituted for.

The trees in A are called **auxiliary trees**. They can represent constituents which are adjuncts to basic structures (adverbial). They can also represent basic sentential structures corresponding to verbs or predicates taking sentential complements. Auxiliary trees (see 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 to be substituted 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.

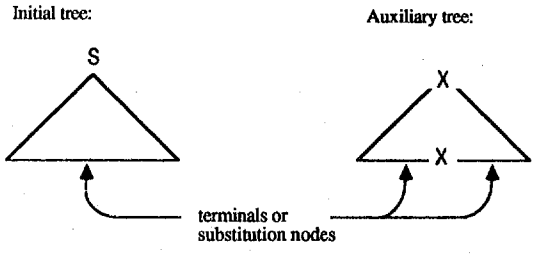


Figure 2: Schematic initial and auxiliary trees

The trees in L are called **lexical trees**. They represent basic categories or constituents which serve as arguments to initial or auxiliary trees. They are reduced to a pre-terminal node in the case of simple categories or are expanded into tree structures in the case of compounds. Structurally they are characterized the same way as initial trees except that they are not necessary rooted by S.

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. In case of substitution on a node labeled by S (sentential complement), only trees derived from initial trees (therefore rooted by S) can be substituted. In all other cases, any tree derived from a lexical 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.

We conventionally mark substitution nodes by a down arrow ($\downarrow$).

We define the **tree set** of a TAG G, $\mathcal{T}(G)$ to be the set of all derived trees starting from 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 $\mathcal{T}(G)$.

Grammar rules defined by the linguistic theory are not the same as the rules used by the parser—let us refer to them as **parser rules**. A parser rule is defined to be a structure encoding a rule of the grammar (or a set of rules) instantiated by the parser when it comes to a lexical item (considered to 'yield' the rule(s)). It is thus a unique object. It is individualized by the lexical item, which is itself individualized by its position in the input string. The lexical item is directly inserted into the structure corresponding to the parser rule, and such a rule can only occur once. **Lexical items** are differentiated by their realization in the input sentence and also their position in the sentence. Therefore a given rule corresponds to exactly one lexical item in the input sentence.

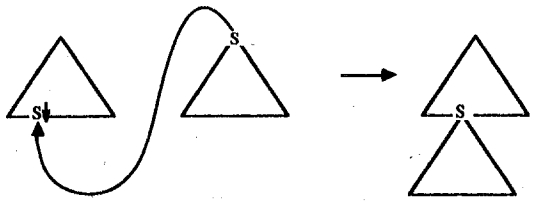


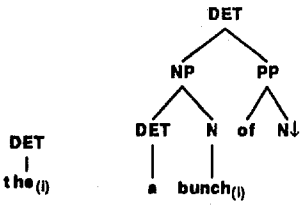
Figure 3: Mechanism of substitution

The structures are produced by lexical items which serve as heads. 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 is linguistically motivated, e.g. by the principles of $\overline{X}$ theory. S also has to be considered as the projection of a lexical head, usually V. Each lexical item corresponds to as many entries as there are possible category or argument structures.

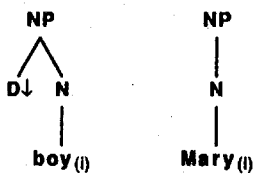
The **category structure** is a lexical tree that is not necessarily reduced to a single category. It corresponds to the maximal projection of a category in the case of simple phrases, to the entire compound, in the case of compound categories.

Category structures can be of two different kinds:

- lexical trees reduced to a single category:[5]



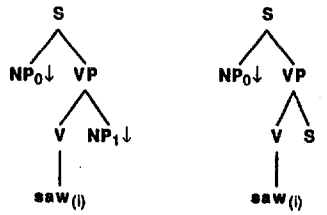
- lexical trees that consist of a phrase:



The **argument structure** is not reduced to a list of arguments as the usual subcategorization frames. It is the syntactic structure constructed with the lexical value of the predicate and with all the nodes for its arguments. The argument structure for a predicate is its maximal structure. An argument is present in the argument structure even if it is optional and its optionality is stated in the structure.

[5]The index in parentheses on a lexical item that produces the structure encodes the position of the lexical item in the string.

A simple case of a argument structure is a verb with its subcategorized arguments. For example, the verb **saw** (at position i) generates the following structures (among others):[6]



The left structure corresponds to:

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

and the other to:

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

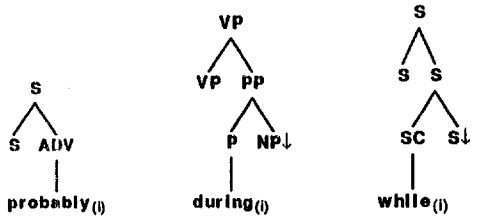
An argument structure can correspond to either one or a set of syntactic surface structures. The lexical head will then produce a set of possible trees, one for $\mathbf{NP}_0$ **saw** $\mathbf{NP}_1$ and another for **who**$_i$ **did** $\mathbf{NP}_0$ **see** ϵ_i **?**, for example. If one defines principles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

Category and argument structures thus instantiated as the parser scans the input string are combined together in a sentence structure by adjoining or substituting.

As Gross (1984), we consider verbs, nouns, and adjectives as predicates yielding sentences. They can take nominal or sentential arguments. If the predicate takes nominal arguments it produces an initial tree. If it takes a sentential argument then it produces an auxiliary tree. Putting arguments into predicates is done by substituting nominal arguments or by adjoining a predicate structure to its sentential argument.

Adjuncts are represented as auxiliary trees rooted by the category of the node they are adjoined to. They are also produced by a head. They can be reduced to a basic category or take nominal or sentential arguments introduced by substitution.

Examples of Adjuncts:



4 Parsing 'lexicalized' grammars

If we have a 'lexicalized' grammar, the grammar of the parser can be reduced to a set of structures whose nature depends on the input string and whose size is proportional to the length of the sentence (if we suppose that the number of structures associated with a lexical item is finite). Since each structure ('rule') corresponds to a token in the sentence, it can be used only once. Rules are now differentiated by their realization 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. Since each rule can be used once, recursion does not lead to the usual non-termination problem. 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:

- First produce the set of structures corresponding to each word in the sentence. This step performs the role of an expanded morphological analysis (or tagging).
- Then put the argument structures into the predicate structures. This step performs a modified syntactic analysis.

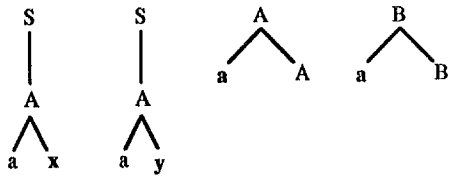
In principle any parsing strategy can be applied to execute the second step, since the number of structures produced is finite and since each of them corresponds to a token in the input string, the search space is finite and termination is guaranteed. In principle, one can proceed inside out, left to right or in any other way. Of course, standard parsing algorithm 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. The use of restrictors as proposed by Shieber (1985) is no longer necessary and the difficulties caused by treating subcategorization as a feature is no longer a problem.

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 produced 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:$\{a^*x\} \cup \{a^*y\}$. In a standard CFG parsing algorithm, A's and B's 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 TAG as follows:



Suppose that the heads of the initial trees are respectively x and y and that a is the head of both auxiliary

[6] We put indices on categories to express syntactic roles (0 for subject, 1 for object).

trees. Then, if the elementary trees are built according to the input string, and if a top-down strategy is used, only A or B trees will be built.

An application concerns the parsing of discontinuous constituents. They are recognized even if there are unbounded insertions between their components and even if their 'head' is the last element of the string.

In the two-step strategy described here, before the first step is taken, there is no grammar. 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. Although we have not yet investigated the implication of this approach on some complexity results, we feel that some of them might be improved.

It is possible to express the parsing problem in a decidable deduction system on trees (similar to Lambek's deduction system on categories (1958 and 1961)). The grammar can be thought as a five-tuple $(V_N, \Sigma, \Theta, S, Lex)$ 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 Σ^* and V_N (the elements of Σ^* having ranked 0).
- Lex is the lexicon, i.e. a function from lexical items to finite subsets of Θ: $\Sigma^* \rightarrow 2^{\Theta}(finite)$.

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 left hand side to form a new one. One inference rule corresponds to adjunction of two trees and the other to substitution of a node in one tree by the other tree. Once two trees are combined, they are replaced by the resulting tree in the left hand side of the sequent. This facts takes into account that each 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 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 decidable.

The language generated by such system is defined to be: $\mathcal{L} = \{a_1, \cdots, a_n | \exists \tau_i \in Lex(a_i) \text{ s. t. } \vdash \tau_1, \cdots, \tau_n \longrightarrow S\}$ 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).

5 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 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 directly CFGs and TAGs. 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), *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) all lexical trees or 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 built, the parser will proceed bottom-up using the top-down filtering from the initial trees that have been built. 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:

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

where $\{\gamma_{i1}, \cdots, \gamma_{ij}\}$ is the set of trees generated 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:

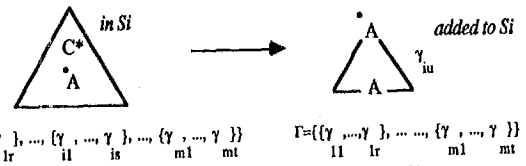


Figure 4: Update of Γ in the Left Predictor

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.

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.

6 Conclusion

In this paper we presented a general parsing strategy based on 'lexicalized' grammar. We defined the notion of lexicalization of a grammar. We showed how a CFG can be 'lexicalized' by using only substitution. But the use of adjunction permits 'lexicalization' with linguistically motivated structures. TAGs have been shown to be naturally 'lexicalized'. Then we gave an overview of the specific lexicon of TAGs. The 'lexicalization' of grammar lead us to introduce a two step parsing strategy. The first step picks up the set of structures corresponding to each word in the sentence. The second step puts the argument structures into predicate structures. Therefore, the relationship between the morphological and syntactic analyses has been modified. In the first step, structures instead of categories are associated with lexical items. The strategy has been shown to be able to use non-local information in the input string. Also problems due to recursion are eliminated. The grammar of the parser has been reduced to a set of structures whose size is proportional to the length of the input sentence. Furthermore, the parsing strategy applies to any parsing algorithm; in particular top-down. It can be formalized into a decidable deduction system that has finite search space for a sentence of finite length. The Earley-type parser for TAGs has been easily extended to take advantage of this strategy.

References

Abeillé, Anne, 1988. Parsing French with Tree Adjoining Grammar: some Linguistic Accounts. In *Proceeding of the* 12^{th} *International Conference on Computational Linguistics.*

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.

Gross, Maurice, 1984. Lexicon-Grammar and the Syntactic Analysis of French. In *Proceeding of the* 10^{th} *International Conference on Computational Linguistics.*

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

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.

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

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

Shieber, Stuart M., 1984. The Design of a Computer Language for Linguistic Information. In 22^{nd} *Meeting of the Association for Computational Linguistics*, pages 362–366.

Shieber, Stuart M., 1985. Using Restriction to Extend Parsing Algorithms for Complex-feature-based Formalisms. In 23^{rd} *Meeting of the Association for Computational Linguistics*, pages 82–93.

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

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., 1988. Feature Structure Based Tree Adjoining Grammars. In *Proceedings of the* 12^{th} *International Conference on Computational Linguistics.*