AN EFFICIENT PARSING ALGORITHM FOR TREE ADJOINING GRAMMARS

Karin Harbusch
DFKI - Deutsches Forschungszentrum für Künstliche Intelligenz
Stuhlsatzenhausweg 3, D-6600 Saarbrücken 11, F.R.G.
harbusch@dfki.uni-sb.de

ABSTRACT

In the literature, Tree Adjoining Grammars (TAGs) are propagated to be adequate for natural language description — analysis as well as generation. In this paper we concentrate on the direction of analysis. Especially important for an implementation of that task is how efficiently this can be done, i.e., how readily the word problem can be solved for TAGs. Up to now, a parser with $O(n^6)$ steps in the worst case was known where n is the length of the input string. In this paper, the result is improved to $O(n^4 \log n)$ as a new lowest upper bound. The paper demonstrates how local interpretion of TAG trees allows this reduction.

1 INTRODUCTION

Compared with the formalism of *context-free grammars* (*CFGs*), the rules of *Tree Adjoining Grammars* (*TAGs*) can be imagined intuitively as parts of context-free derivation trees. Without paying attention to the fact that there are some more restrictions for these rules, the recursion operation (*adjoining*) is represented as replacing a node in a TAG rule by another TAG rule so that larger derivation trees are built.

This close relation between CFGs and TAGs can imply that they are equivalent. But TAGs are more powerful than context-free grammars. This additional power — characterized as *mildly context-sensitive* — leads to the question of whether there are efficient algorithms to solve the word problem for TAGs.

Up to now, the algorithm of *Vijay-Shanker* and *Joshi* with a time complexity of $O(n^6)$ for the worst case was known, in addition to several unsuccessful attempts to improve this result. This paper's main emphasis is on the improvement of this result. An efficient parser for Tree Adjoining Grammars with a worst case time complexity of $O(n^4 \log n)$ is discussed.

All known parsing algorithms for TAGs use the close structural similarity between TAGs and CFGs, which can be expressed by writing all inner nodes and all their sons in a TAG as the rule set of a context-free grammar (the *context-free kernel* of a TAG). Additionally, the constraint has to be tested that all further context-free rules corresponding to the same TAG tree must appear in the derivation tree, iff one rule of that TAG tree is in use. Therefore, it is clear that a context-free parser can be the basis for extensions representing the test of the additional constraint.

On the basis of the two fundamental context-free analysers, the different approaches for TAGs can be divided into two classes. One class extends an *Earley* parser and the second class extends a *Cocke-Kasami-Younger* (*CKY*) parser for CFGs. Here, we focus on the approaches with a CKY basis, because the relation between the resulting *triangle matrix* and the encoded derivation trees is closer than for the *item lists* of an Earley parser.

In particular, the paper is divided into the following sections. First, a short overview of the TAG formalism is given in order to have a common terminological basis with the reader.

In the second section, the approach of *Vijay-Shanker* and *Joshi* is presented as the natural way of extending the CKY algorithm for context-free grammars to TAGs. As a precondition for that analysis, it has to be proven that each TAG can be transformed into *two form*, a normal form restricting the outdegree of a node to be less three.

In section 4, the main section of this paper, a *normal form* is defined as a precondition for a new and more efficient parsing algorithm. This form is more restricted than the two form, and is closely related to the *Chomsky normal form* for CFGs. The main emphasis lies on the description of the new parsing approach. The general idea is to separate the context-free parsing and the additional testing so that the test can run locally. On the triangle matrix which is the result of the CKY analysis with the context-free kernel, all complete TAG trees encoded in the triangle matrix are computed recursively. It is intuitively motivated that this approach needs fewer steps than the strategy of *Vijay-Shanker* and *Joshi*, which stores all intermediate states of TAG derivations, because the locally represented elementary trees can be interpreted as TAG derivations where equal parts are computed exactly once instead of individual representations in each derivation.

In the summary, our experience with an implementation in CommonLISP on a Hewlett Packard machine is mentioned to illustrate the response time in an *average case*. Finally, different approaches for TAG parsing are characterized and compared with the approaches presented here.

2 TAGS BRIEFLY REVISITED

First of all, the basic definitions for TAGs are revisited in order to have a common terminology with the reader (even though not defined explicitly here, CFGs are used as described, e.g., in [Hopcroft, Ullman 79]).

In 1975, the formalism of Tree Adjoining Grammars (TAGs) was introduced by *Aravind K. Joshi*, *Leon S. Levy* and *Masako Takahashi* ([Joshi et al. 75]). Since then, a wide variety of properties — formal properties as well as linguistically relevant ones — have been studied (see, e.g., [Joshi 85] for a good overview).

The following example describing the crossed dependencies in Dutch should illustrate the formalism (see Figure 1, where the node numbers written in slanted font should be ignored here; they make sense in combination with the description of the new algorithm, especially step (**tag2**)). A TAG is a tree generation system. It consists, in addition to the set of *nonterminals* N, the set of *terminals* T and the *start symbol* S, an extraordinary symbol in N, of two different sets of trees, which specify the rules of a TAG. Intuitively, the set I of *initial trees* can be seen as context-free derivation trees. This means the start symbol is the root node, all inner nodes are nonterminals and all leaves are terminals (e.g., in Figure 1 tree α). The second set A, the *auxiliary trees*, which can replace a node in an initial tree (which is possibly modified by further adjoinings) during the recursion process, must have a form, so that again a derivation tree results. The trees β_1 and β_2 demonstrate that restriction. A special leaf (the *foot node*) must exist, labelled with the same nonterminal as the root node. Further, it is obligatory that an auxiliary tree derives at least one terminal. The union of the initial and the auxiliary trees, so to speak the rule set of a TAG, is called the set of *elementary trees.*

Tree γ in Figure 1 shows a *TAG derivation tree*, which means an initial tree with an arbitrary number of adjoinings (here β_1 is adjoined at the node S* in α and β_2 at the node S* in the adjoined tree β_1). During the recursion process (*adjoining*), a node X in an initial tree α, which can be modified by further adjoinings, is replaced by an auxiliary tree β with the same nonterminal label at root and foot node, that X is labelled with. The incoming edge in X (if it exists; this is true if X is not the root node of α) now ends in the root node of β, and all outgoing edges of X in α now start at the foot node of β.

The set of all initial trees modified by an arbitrary number of adjoinings (at least zero) is called T(G), the *tree set* of a TAG G. The elements in this set can also be specified by building a series of triples (α_i, β_i, X_i) $(0 \leq i \leq n)$ — the *derivation* — where $\alpha_0 \in I$, α_i $(1 \leq i \leq n)$ is the result of the adjoining of β_{i-1} in node X_{i-1} in α_{i-1}, β_i $(0 \leq i \leq n\text{-}1)$ is the auxiliary tree, which is adjoined in node X_i in tree α_i and X_i $(0 \leq i \leq n\text{-}1)$ is a unique node number in α_i. This description has the advantage that structurally equal trees in T(G) which result from different adjoinings can be uniquely represented.

L(G), the *language* of a TAG, is defined as the set containing all leaf strings of trees in T(G), respectively all trees which can be constructed by adjoining as described in the corresponding derivation. Here, a leaf string means all labels of leaves in a tree are concatenated in order from left to right. In the tree γ in Figure 1 *'Jan Piet Marie ϵ ϵ zag laten zwemmen'* is in L(G).

The relation between TAGs and CFGs can be characterized by defining the *context-free kernel* K of a TAG G. K is a CFG and consists of the same sets N, T and S of G, but P(K) is the set of all inner nodes of all elementary trees in G interpreted as the lefthand side of a rule, where all sons in their order from left to right build the righthand side of that rule. E.g., in Figure 1 β_2 has the corresponding context-free rules: (S, NP VP), (NP, N), (N, Jan), (VP, S V_1), (V_1, zag).

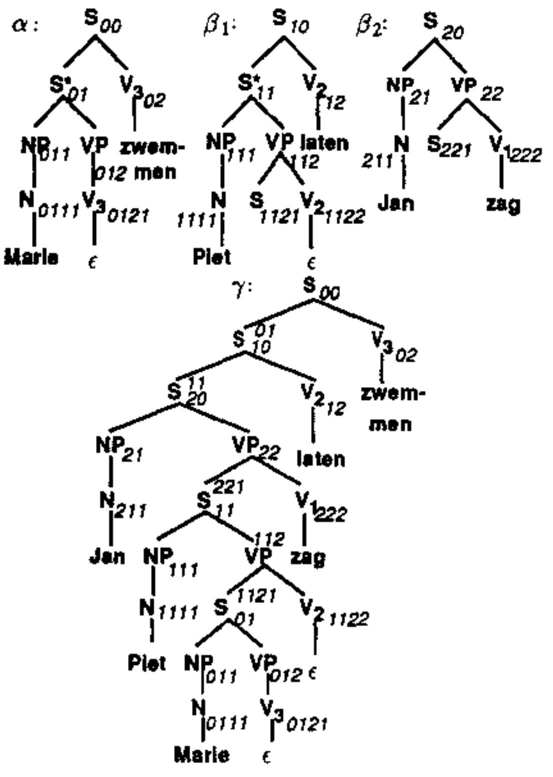


Figure 1: A small sample TAG demonstrating the process of ADJOINING

It is clear that having a context-free derivation tree (on the basis of the context-free kernel K of a TAG G) is a necessary, but not sufficient property for an input string, which is tested to be an element in L(G). In the following, this property motivates the extension of context-free parsing algorithms to accept TAGs as well.

The following parsing algorithms are able to accept some extensions of the pure TAG definition without changing the upper time bound. Here, only *TAGs with Constraints* are mentioned (for more information about other extensions, e.g., *TAGs with Links, with Unification* or *Multi Component TAGs* — some extending the generative capacity — see, e.g., [Joshi 85]).

The motivation for *TAGs with Constraints* (*TAGCs*) is to restrict the recursion operation of TAGs. Each node X in an elementary tree labelled with a nonterminal has an associated *constraint set* C, which has one of the following forms:

- NA stands for *null adjoining* and means that at node X no adjoining can take place,
- SA(B) stands for *selective adjoining* and means that at X the adjoining of an auxil-

iary tree ($\in$ B) can take place (where each tree in B has the same root and foot node label as X) or

- OA(B) stands for *obligatory adjoining* and means that at X the adjoining of an auxiliary tree ($\in$ B) must take place (where each tree in B has the same root and foot node label as X).

When TAGs are mentioned in the following, the same result can be shown for TAGs with Constraints, which it is not explicitly outlined. Only the property of generative power is illustrated, to make clear that finding a parsing algorithm is not a trivial task. For more information about the linguistic relevance of TAGs, the reader is referred, e.g., to [Kroch, Joshi 85].

A first impression comparing the generative power of TAGs and CFGs can be that they are equivalent, but TAGs are more powerful, e.g., the famous language a^n b^n e c^n can be produced by a TAG with Constraints (the main idea in constructing this grammar is to represent the production of an a, a b and a c in one auxiliary tree). Thinking of the application domain of natural language processing, the discussion in the linguistic community becomes relevant as to how powerful a linguistic formalism should be (see, e.g., [Pullum 84] or [Shieber 85]). TAGs are *mildly context-sensitive*, which means that they can describe some context-sensitive languages, but not all (e.g., www with w $\in \{a,b\}^*$, but ww is acceptable for a TAG). One thesis holds that natural language can be described very well by a mildly context-sensitive formalism. But this can only be empirically confirmed by describing difficult linguistic phenomena (here, the example in Figure 1 can only give an idea of the appropriateness of TAGs for natural language description).

This property leads to the question of whether the *word problem* is solvable and if so, how efficiently. In the following section, two different polynomial approaches are presented in detail. The property of efficiency becomes important when a TAG should be used in the application domain mentioned above, e.g., one can think of a syntax description encoded in TAG rules which is part of a natural language dialogue system. The execution time is responsible for the acceptance of the whole system. Later on, our experience with the response time of an implementation of the new algorithm is described.

3 THE VIJAY-SHANKER AND JOSHI APPROACH

First, the approach of *Vijay-Shanker* and *Joshi* (see [Vijay-Shanker, Joshi 85]) is discussed as the natural way of extending the context-free CKY algorithm (see, e.g., [Hopcroft, Ullman 79]) to analyze TAGs as well. As for the context-free analysis with CKY, the grammar is required in normal form as a precondition for the TAG parser. Therefore, first the two form is defined and the idea of the constructive proof for transforming a TAG into two form is given. The TAG parser is then presented in more detail.

3.1 TWO FORM TRANSFORMATION

The parsing algorithm of *Vijay-Shanker* and *Joshi* uses a special CKY algorithm for CFGs which requires fewer restrictive constraints than the *Chomsky normal form* for the ordinary CKY algorithm does. Here, the righthand side of all rules of the grammar should have at most two elements. This definition has to be adapted for TAG rules to extend this CKY parser to analyze TAGs as well.

A TAG G is in *two form*, iff each node in each elementary tree has at most two sons. It can be proven that each TAG G can be transformed into a TAG G' in two form with L(G) = L(G').

The proof of that theorem uses the same techniques as in the context-free case which allow the reduction of the number of elements on the righthand side to build the Chomsky normal form. If there are more than two sons, the second and all additional sons are replaced by a new nonterminal which becomes the lefthand side of a new rule with all replaced symbols on the righthand side (for more details see [Vijay-Shanker, Joshi 85]). We always refer to a TAG in two form, even when it is not explicitly confirmed.

3.2 THE STEPS OF THE ALGORITHM

Now the idea of extending each context-free analysis step by additional tests to ensure that whole TAG trees are in use (sufficient property) is motivated. This approach was proposed to be natural because it tries to build TAG derivation trees at once. In contrast, a two level approach is presented which constructs all context-free derivation trees before the TAG derivations are computed in a second step.

In the CKY analysis used here, a cell [row i,column j] in the triangle matrix ($1 \leq i$, $j \leq n$, the length of the input string w = $t_1 \ldots t_n$, where without loss of generality $n \geq 1$, because the test for ϵ, the empty string, $\in$ L(G) simply consists of searching for initial trees with all leaves labelled with ϵ) contains an element X ($\in$ N) iff there are rules to produce the derivation for $t_{i+1} \ldots t_{j-1}$. This invariant is extended to represent a TAG derivation for $t_{i+1} \ldots t_{j-1}$ iff X $\in [i,j]$. Therefore additional information of each nonterminal in a cell has to be stored as to which elementary trees are under completion and what subtrees have been analyzed up to now. Important to note is that the list of trees which are under completion, can be longer than one. E.g., think of adjoinings which have taken place in adjoined trees as described in Figure 1.

For realization of that information, a *stack* can be imagined. Here, the different stack elements are stored separately to use intermediate states in common. A stack element contains the information of exactly one auxiliary tree which is under construction, and a pointer to the next stack element. This pointer is realized by two additional positions for each cell in the triangle matrix ($[i,j,k,l]$), where k and l in the third and fourth position characterize the fact that from t_{k+1} to

t_{l-1} no information about the structure of the TAG derivation is known in this element and has to be reconstructed by examination of all cells $[k,l,v,w]$ ($k \leq v \leq w \leq l$). The stack cells which the elements point at must also be recursively interpreted until the whole subtree is examined (left and right stack pointer are equal). It is clear that in interpreting these chains of pointers the stack at each node X in the triangle matrix represents all intermediate states of TAG derivations with X as root node in an individual cell of the triangle matrix.

The algorithm starts initializing cells for all terminal leaves ($X \in [i\text{-}1,i,i,i]$ for t_i with father X, $1 \leq i \leq n$) and all foot nodes which can be seen as nonterminal leaves ($X \in [i,j,i,j]$ where X is a foot node in an auxiliary tree, $0 \leq i < j \leq n\text{-}1$).

Just as the CKY algorithm tests all combinations of neighboring strings, here new elements of cells are computed together with the context-free invariant computation, e.g., iff (Z,X Y) is a rule in the context-free kernel of the input TAG and $X \in [i,j,k,l]$, $Y \in [j-1,m,p,p]$ and X and Y are root nodes of neighboring parts in the same elementary tree, then Z is added to $[i,m,k,l]$). With the additional test to determine whether the rule (in the example (Z,X Y)) is in the same TAG tree as the two sons (X and Y) and whether the same holds for the subtrees below X and Y, it is clear that a whole TAG tree can be detected. If this is the case, i.e., that two neighboring stack elements should be combined, all elements of cells $[k,l,m,p]$ are added to $[i,j,m,p]$ iff $X \in [i,j,k,l]$ is the root of an identified auxiliary tree.

The time complexity becomes obvious when the range of the loops for all four dimensions of the array is described explicitly (see [Vijay-Shanker, Joshi 85]). From a more abstract point of view, the main difference between the CKY analysis for a CFG and a TAG is that, the subtrees below the foot nodes are stored. This fact extends the input of length n to n^2 to describe the two additional dimensions. On the basis of that input, the ordinary CKY analysis can be done, and so the expected time complexity is $O((n^2)^3) = O(n^6)$. With the explicitly defined ranges of the four dimensions for the positions in the array, it is clear that the worst case and the best case for this algorithm are equal.

4 A NEW AND MORE EFFICIENT APPROACH

A time bound of $O(n^6)$ in the best and worst case must be seen as a more theoretical result, because an implementation of the algorithm shows that the execution time is unacceptable. In order to use the formalism for any application domain, this result should be improved. In this section, a TAG parser with an upper bound of $O(n^4 \log n)$ in the worst case is presented. The best case is $O(n^3)$, because a CKY analysis has to at least be done.

4.1 NORMAL FORM TRANSFORMATION

As precondition of the new parsing algorithm, the TAG has to be transformed into a *normal form* which contains only trees with nodes and their sons, following the *Chomsky normal form* definition. This means that the following three conditions hold for a TAG G:

1. $\epsilon \in L(G)$ iff a tree with root node S (NA), the start symbol, which allows no further adjoinings (null adjoining), and a single terminal son ϵ is element in the set of initial trees I (this tree is called the *ϵ tree*),
2. except the ϵ tree, no leaf in another elementary tree is labelled with ϵ, and
3. for each node in each elementary tree, the condition holds that either the node has two sons both labelled with a nonterminal or that the node has one son labelled with a terminal.

In a first step, each TAG is transformed in two form so that condition 3 can be satisfied easier. This transformation is accomplished by the constructive proof for the theorem that for each TAG (or TAG with Constraints for which the definition holds as well) there exists an equivalent TAG with Constraints in normal.

Important to note is that the idea of the transformation into Chomsky normal form for CFGs cannot be adopted further on because this construction allows the erasure of nonterminal symbols if their derived structure is added to the grammar. In TAGs, a nonterminal not only represents the derivation of its subtree in an elementary tree, but can be replaced by an adjoining. Therefore, the general idea of the proof is to erase parts of elementary trees which are not in normal form, and represent those parts as new auxiliary trees. After this step, the original grammar is in normal form and therefore the encoded auxiliary trees can be used for explicit adjoinings, always producing structures in normal form. *Explicit adjoinings* mean adjoinings in the new auxiliary trees which were built out of the erased parts of the original grammar. These adjoinings replace the nodes which are not in normal form. Since the details of the different steps are of no further interest here, the reader is referred to [Harbusch 89] for the complete proof.

4.2 THE STEPS OF THE NEW PARSING ALGORITHM

The input of the new parser consists of a TAG G in normal form, and a string $w = t_1 \ldots t_n$. With condition one in the normal form definition, the test for $\epsilon \in L(G)$ is trivial again. From now on this case is ignored, i.e., $n \geq 1$.

The algorithm is divided into two steps. First a CKY analysis is done with the context-free kernel K of the input TAG G. Here, the standard CKY algorithm as described in [Hopcroft, Ullman 79] is taken, which requires a CFG in Chomsky normal form. K satisfies the requirement that the TAG G is in normal form. One can think that it would be sufficient to simply transform the context-free kernel into Chomsky normal form instead of transforming the input TAG. But with this strategy one would loose the one-to-one mapping of context-free rules in the CFK and father-son-relations in a TAG rule which becomes important for finding complete TAG rules in the second step of the new parser.

Here the invariant of the CKY analysis is X $\in [i, j]$ iff there are rules to produce a derivation for $t_i \ldots t_{j+i-1}$. This information is slightly extended to recognize complete subtrees of elementary trees in the triangle matrix. In the terminology of *Vijay-Shanker* and *Joshi*, a stack element is constructed. But it's important to note that the pointers are not interpreted, so that here local information is computed relative to an elementary tree.

Actually, the correspondence between an element in the triangle matrix and a TAG tree is represented as a pointer from the node in a triangle cell to a node in an elementary tree as described in Figure 2 (ignore the dotted lines at the moment). An equivalent description is presented in Figure 3 by storing the unique node number at which the pointer ends in the elementary tree and additionally a flag indicating whether the TAG tree is initial (I) or auxiliary (A) and whether the node is root node (T) of the tree or not (L). E.g., in Figure 2 the NP-son of the root node S in tree γ carries the flag TA.

In this terminology, the special case that the subtree contains the foot node has to be represented explicitly, because the foot node is a leaf in the sense of elementary trees, but not in the sense of a derivation tree. To know where this leaf is positioned in the triangle matrix, a *foot node pointer* (*FP*) is defined from the root of the subtree to the foot node if one exists in that tree (in Figure 2 the dashed arc).

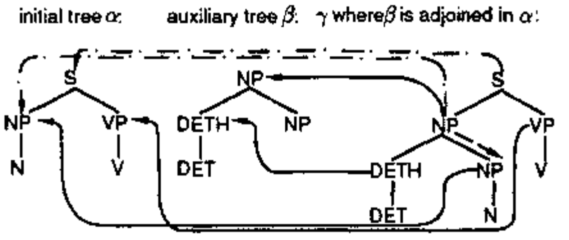


Figure 2: Example illustrating the inductive basis of the new invariant

So, the invariant in the first step of the new parsing algorithm is computed during the CKY analysis — in our second terminology — by recursively defining *extended node numbers* (*ENNs*) by triples (NN,TK,FP) as follows:

Initialization

Each element X in level 1 (father of a terminal t) is initialized with an ENN, where NN is the *node number* of X in a father-son-relation in an elementary tree α ($X \longrightarrow t$), the *tree kind* TK := LU (U=I,A) iff $\alpha \in U$ and NN doesn't end with zero (X is not the root of α) else TK := TU, and the *foot node pointer* FP := nil, because the father of a terminal is never a foot node in the same auxiliary tree.

For each node X in level 1 the ENN := (NN=node number of a foot node in an auxiliary tree, LA, pointer to that ENN) is added iff X is the label of the foot node with node number NN — to describe foot node leaves.

Recursion along the CKY analysis

For each new context-free element Z ($Z \longrightarrow X\ Y$), the following tests are done in addition:

If X has an ENN (NN_1,TK_1,FP_1) and Y has an ENN (NN_2,TK_2,FP_2) and NN_1-(1 in the last positition) = NN_2-(2 in the last position) and $TK_1 = TK_2$ and at least FP_1 or FP_2 = nil then for Z an ENN (NN_1-1,TK,FP) is added where TK = TK_1 if Z is not the root node of the whole tree (in this case TK = TK_1-(L+T in the first position)); FP = FP_i (i=1,2), which is not equal nil, else it is nil.

If an auxiliary tree with Z the label of the foot node exists, the ENN (NN=node number of the foot node in that tree, LA, pointer to that element) is added to Z in the currently manipulated triangle cell — to represent the possibility of an adjoining in that node.

It is obvious that this invariant consisting of the nonterminal in a cell of the triangle matrix to represent the context-free invariant, the pointers to elementary trees, and the foot node pointers to represent which part of an elementary tree is analyzed computes less information than an array cell in the approach of *Vijay-Shanker* and *Joshi* does, where whole subtrees of the derivation tree are stored not stopping at a foot node as we do. Also, it is clear that this invariant can be computed recursively during the ordinary CKY steps within the upper time bound of $O(n^3)$. The number of pointers to elementary trees at each node can be restricted by the number of occurences of a nonterminal as the left-hand side symbol of a rule in the context-free kernel (which is a constant). The number of foot node pointers is restricted by the outdegree of each cell in the triangle matrix ($\leq n$), because only for such an edge can an FP be recursively defined.

In the second step, whole TAG derivations are computed by combining the subtrees of elementary trees (represented by the invariant after step 1), according to the adjoining definition interpreted inversely. *Inversely* means that the equivalence in the adjoining definition is not interpreted in the direction that a node is replaced by a tree, but in the opposite direction, where trees have to be detected and are *eliminated.*

Since all TAG derivation trees of a string w and a TAG G are encoded in the triangle matrix (necessary condition $w \in CFK(G)$) and have to be found in the triangle matrix, the derivation definition has to be modified as well to support the 'inverse' adjoining definition. It means that a string $w \in L(G)$ iff there exists a tree, where recursively all complete auxiliary trees can be detected and replaced by the label of the root node of the auxiliary tree until this process terminates in an initial tree.

The second step formulates the algorithm for exactly this definition. An auxiliary tree in the derivation tree which contains no further adjoinings is called an *innermost tree*. As long as the termination condition isn't satisfied, at least one innermost tree must exist in the derivation tree.

Returning to the invariant in the first step, innermost trees are characterized as a pointers to the root node of an auxiliary tree or in the representation of ENNs as the node number of the root node (in our numbering algorithm visible by the end number zero) and the tree kind flag TA (total auxiliary).

These trees are *eliminated* by identifying the root and the foot nodes of innermost trees, so to speak, as interpretation of the foot node pointers as ϵ edges. This can be represented simply as propagation of the pointers from the foot node to the root node. This information is sufficient because the strategy of the algorithm checks whether an incoming edge in a node and the information of an outgoing edge (without loss of generality represented at the start node of the edge) belong to the same elementary tree. Note that this bottom-up interpretation of the derivation trees (*propagation*) realizes that the finding of larger subtrees is computed only once (the father-son relation is only interpreted in the upward direction). In Figure 2 the dotted line from the NP node in γ describes the elimination of β by propagation of the information from the foot node to the root node.

Since it doesn't matter in the algorithm what history an information in a node has (especially how much and exactly what trees are eliminated) all possibilities of producing new extended node numbers — representing the new invariant — are simply called *elimination*. The information in a node represents what further parts of the same elementary tree are expected to be found in the triangle matrix above that node. A subclassification differentiates what kinds of incoming edges should be compared to find these parts. One class describes whether such a further piece is detected — by interpreting incoming and outgoing edges of the same node (*simple elimination*). E.g., this is the case in the inductive basis of the invariant definition. The second class realizes the elimination of a detected innermost tree, where its foot node pointer ends in that node. Then the neighborhood of the incoming edges in the root node of the innermost tree and the outgoing edges in the foot node (the currently examined node where the invariant contains the information of the outgoing edges from this node) has to be tested (*complex elimination*). By this classification, each neighborhood — the explicitly represented ones in the triangle matrix as well as the neighborhoods via ϵ respectively foot node pointer edges — is examined exactly once during the algorithm.

The fact that a derivation tree again results after an elimination, which is encoded in the triangle matrix as well, becomes clear by looking at the invariant after an elimination. In the first step the invariant describes complete subtrees of elementary trees. If a complete innermost tree is eliminated by propagating the complete subtrees of elementary trees derived by the foot node to the root node, this represents the fact that the root node can derive both trees, but the subtrees below the foot node have to be completed. This can be done again by elimination (in Figure 2 the dotted line from node S represents the computation of a TAG tree after an elimination).

Since this is not the place to present the algorithm in detail, it is described in informal terms:

(tag1) Treatment of the Empty String

```
ACCEPT := false;
if w = ε then if ε tree ∈ I
                  then ACCEPT := true; fi;
              goto (tag7); fi;
```

From now on, G is interpreted without the ϵ tree.

(tag2) Definition of Unique Node Numbers

$\forall$ nodes X in $\alpha \in (I \cup A)$ a unique node number NN is defined recursively as follows:

- α has a unique number k all over the grammar (starting with zero),
- if X is root node NN := k0, for X the left or only son of the root NN := k1, for X the right son of the root (if existing) NN := k2, and
- for the left or only son of a node with node number kx ($x \in \{1,2\}^+$) NN := kx1, for the right son of kx NN := kx2.

(tag3) Computation of the Context-Free Kernel for The TAG (CFK)

Each inner node of an elementary tree in G and its sons are interpreted as a context-free rule where the node number and the constraints are represented as well.

(tag4) Cocke-Kasami-Younger-Analysis with CFK and w

The slightly extended CKY algorithm is applied to w and CFK. The result is a triangle matrix if the following holds:

```
if w ∉ L(CFK) then goto (tag7)
              else goto (tag5); fi;
```

(tag5) Computation of the Initial State

All possible extended node numbers are computed, which means that all auxiliary trees, or respectively all subtrees of elementary trees, are computed on the triangle matrix and gathered in SAT, the **set of active trees.**

(tag6) Iteration on the Elimination and the Initial State

NEWSAT1 and NEWSAT2 are empty sets and COUNT := 1.

(it0) **if** an extended node number with tree kind TK = TI $\in$ [1,n] **then** ACCEPT := true and COUNT := n; **fi**;

(it1) **if** COUNT = n **then goto (tag7); fi**;

(it2) $\forall$ nodes k with extended node number ENN $\in$ SAT and tree kind of ENN = TA : propagate the extended node number of the node which the foot node pointer points at to the root node and add this information to NEWSAT1;

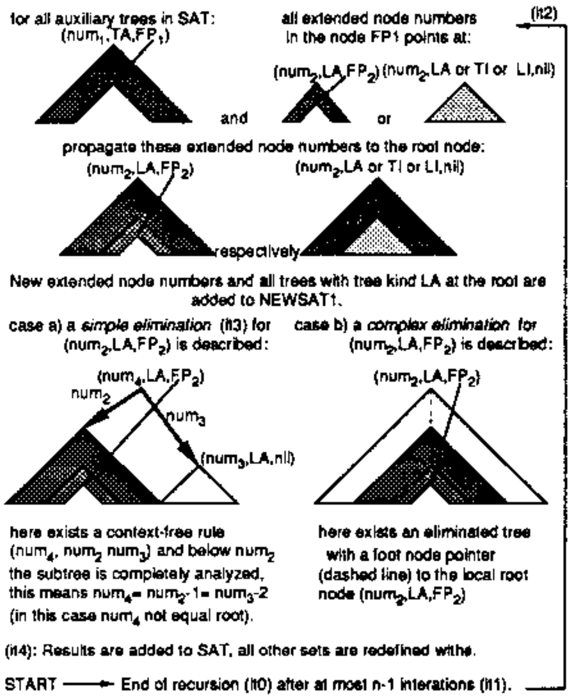
(it3) ∀ nodes k ∈ NEWSAT1 : do all *simple* and *complex eliminations* and add the new extended node numbers to NEWSAT2;

(it4) SAT := NEWSAT2; NEWSAT1 and NEWSAT2 := ∅, COUNT := COUNT+1 and **goto** (it0).

(tag7) Output of the Result

If ACCEPT = true **then** $w \in L(G)$ **else** $w \notin L(G)$; **fi.**

Figure 3 illustrates the recursion step (tag6) for a single, but arbitrary innermost tree representing an auxiliary tree with the root node number num_1.



for all auxiliary trees in SAT:
(num1,TA,FP1)
all extended node numbers in the node FP1 points at:
(it2)
(num2,LA,FP2)
(num2,LA or TI or LI,nil)
and
or
propagate these extended node numbers to the root node:
(num2,LA,FP2)
(num2,LA or TI or LI,nil)
respectively
New extended node numbers and all trees with tree kind LA at the root are added to NEWSAT1.
case a) a simple elimination (it3) for (num2,LA,FP2) is described:
case b) a complex elimination for (num2,LA,FP2) is described:
(num4,LA,FP2)
num2
num3
(num3,LA,nil)
(num2,LA,FP2)
here exists a context-free rule (num4, num2 num3) and below num2 the subtree is completely analyzed, this means num4= num2-1= num3-2 (in this case num4 not equal root).
here exists an eliminated tree with a foot node pointer (dashed line) to the local root node (num2,LA,FP2)
(it4): Results are added to SAT, all other sets are redefined with∅.
START ← End of recursion (it0) after at most n-1 interations (it1).

Figure 3: Illustration of the step of recursion

Here, the question of correctness is not discussed in more detail (see [Harbusch 89]). It should be intuitively clear with the correspondence between the derivation definition and it's interpretation in the recursion step.

Actually, the main emphasis lies on the explanation of the time complexity (for the formal proof see [Harbusch 89]). A good intuition can be won by concentrating for a first glance on a single, but arbitrary TAG derivation tree δ for $w = t_1 ... t_n$ in the triangle matrix after step one. It is clear that δ contains at most n-1 adjoinings, because each TAG tree must produce at least one terminal. Therefore the recursion, which finds independent (unnested) adjoinings simultaneously (after elimination of nested adjoinings identified in the last recursion step), terminates definitively after n-1 loops.

At the beginning, at most $O(n^2)$ innermost trees can exist in the triangle matrix. Each terminal can be a leaf in a constant number of elementary trees and with an indegree of O(n-1) in row 1 of the triangle matrix, the number of occurences of elementary trees containing the input symbol t_i $(1 \leq i \leq n)$ encoded in the invariant after step one is restricted.

Since an elimination is defined along the path between root and foot node of an auxiliary tree, which has at least length 1 (i.e., root and foot node are father and son), the foot node information is always propagated to a higher row in the triangle matrix. The triangle matrix has depth n so that the information of a node in δ — our explicitly chosen derivation tree — can only be passed to O(n-1) nodes because each node has indegree 1 in a derivation tree. The passing of information (propagation) stands for the elimination of O(n-1) innermost trees along the path to the root node. So, the invariant of that node (a constant number of ENNs) can be propagated to O(n) nodes. As a result, the number of invariants at a node increases to O(n). This must be done for all nodes ($O(n^2)$) so that the overall number of steps to find a special, but arbitrary TAG derivation tree is $O(n^3)$.

These suggestions can be used as a basis for finding all derivation trees in parallel instead of a single, but arbitrary one, because all intermediate states in the triangle matrix are shared. The only difference is that the indegree of a node cannot be restricted to 1, but to O(n) so that the exponent 3 increases to 4. The extension "log n" results from storing the foot node pointers, where addresses have to be represented instead of numbers of other cells as in the *Vijay-Shanker-Joshi* approach.

In other words, an intuition for an upper time bound of the algorithm is that the recursion step can be seen as a CKY analysis, because particularly neighboring subtrees are combined to build a larger structure, where the constant number of nonterminals in a cell has to be replaced by O(n) candidates ($O(n^3) \times n$).

Another intuition gives a comparison with the *Vijay-Shanker* and *Joshi* approach. It is obvious that our new approach has a different time bound for the best and the worst case, because all possibilities violating the necessary condition to have a context-free derivation are filtered out before step two is started. In the *Vijay-Shanker* and *Joshi* approach for all context-free subtrees of the triangle matrix, the invariant is computed. But this fact doesn't modify the upper time bound. The main difference lies in the execution time for the two different invariants. In the *Vijay-Shanker-Joshi* approach, all different TAG derivations for a subtree are gathered in the stack of a node in a cell. For all these possibilities, the building process of larger structures is done separately, although the differences in the derivation tree doesn't concern the auxiliary tree actually mentioned. Our local invariant always handles an auxiliary tree with no further information about the derivation. Therefore each elimination of an auxiliary tree is done once only for all derivation trees. From this point of view, the different exponent results from the existence of $O(n^2)$ stack pointers at each node in the triangle matrix.

For both approaches, the integration of TAGs with Constraints is mentioned in common. For the new approach, this extension is obligatory because the normal form transformation produces a TAGC. Anyway, this additional computation doesn't change the upper time bound, because constraints are local and their satisfaction has only to be tested iff an innermost tree should be eliminated (i.e., a stack pointer has to be extended). In this case it had to be checked whether all obligatory constraints in the eliminated tree are satisfied and whether the adjoining was allowed (by analyzing to which tree the rule of the incoming edge in the root node belongs and what constraint the end point of that edge has).

5 SUMMARY

In the application domain of natural language processing, the execution time in an average case is of great interest as well. For the new parsing algorithm, a result is not yet known, but in basic considerations the main idea is to take the depth of analyzed parts of derivation trees as a constant term to come up with a result of $O(n^3)$.

Actually, an implementation of the presented formalism exists written in Common LISP on a Hewlett Packard machine of the 9000 series (for more details about the implementation see [Buschauer et al. 89]). To give an idea of the response time, the analysis of a sentence of about 10 to 15 words and a grammar of about 20 to 30 elementary trees takes at most 6 milliseconds. Currently, this implementation is extended to build a workbench supporting a linguist in writing and testing large TAG grammars (respectively TAGs with Unification).

Finally, other approaches for TAG parsing should be mentioned and compared with the presented result. In the literature, the two Earleybased approaches of *Schabes* and *Joshi* (see [Schabes, Joshi 89]) and of *Lang* ([Lang 86]) are proposed. The lowest upper time bound for the *Schabes-Joshi* approach is $O(n^9)$ and for the approach of *Lang* $O(n^6)$. But both algorithms come up with better results in the best and in the average case. In the framework of parallel parsing, results for TAGs are also proposed. In [Palis et al. 87] a linear time approach on $O(n^5)$ processors and in [Palis, Shende 88] a sublinear ($O(\log^2 n)$) algorithm is described.

One future perspective is to parallelize the new approach by the same method so that the expected result should be a linear time bound on $O(n^2)$ processors. More concretely, an optimal layout for two processors is looked for, where independent subtrees have to be specified (candidates are not always total innermost trees, e.g., if only one TAG derivation exists where all innermost trees are nested).

Further on, we concentrate on appropriate extensions of the TAG formalism for analysis as well as *generation* of natural language with the ambitious aim to verify that TAGs (in some extension) are appropriate for a *bidirectional and integrated description of syntax, semantics and pragmatics.*

ACKNOWLEDGEMENTS

This paper is based on thesis work done under supervision of *Wolfgang Wahlster* and *Günther Hotz*. I would like to gratefully acknowledge *Hans Arz, Bela Buschauer, Günther Hotz, Paul Molitor, Peter Poller, Anne Schauder* and *Wolfgang Wahlster* for their valuable interactions.
I would like to thank *Aravind Joshi* for his helpful comments in earlier discussions and especially on this paper.

REFERENCES

B. Buschauer, P. Poller, A. Schauder, K. Harbusch. 1989. *Parsing von TAGs mit Unifikation.* Saarbrücken, F.R.G.: "AI-Laboratory" Memo, Dept. of Computer Science, Univ. of Saarland.

K. Harbusch. 1989. *Effiziente Strukturanalyse natürlicher Sprache mit Tree Adjoining Grammars.* PhD Thesis, Saarbrücken, F.R.G.: Dept. of Computer Science, Univ. of Saarland.

J. E. Hopcroft, J. D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts.

A. K. Joshi. 1985. *An Introduction to Tree Adjoining Grammars.* Philadelphia, Pennsylvania: Technical Report MS-CIS-86-64, Dept. of Computer and Information Science, Moore School, Univ. of Pennsylvania.

A. K. Joshi, L. S. Levy, M. Takahashi. 1975. *Tree Adjoining Grammars.* Journal of Computer and Systems Science 10:1, Seite 136-163.

T. Kroch, A. K. Joshi. 1985 *Linguistic Relevance of Tree Adjoining Grammars.* Philadelphia, Pennsylvania: Technical Report MS-CIS-85-16, Dept. of Computer and Information Science, Moore School, Univ. of Pennsylvania.

B. Lang. 1989 forthcoming. *A Uniform Framework for Parsing,* Proceedings of the International Workshop on Parsing Technologies in Pittsburgh, 28th-31rd of August.

G. Pullum. 1984. *On Two Recent Attempts to Show That English Is Not a CFL.* Computational Linguistics 10(4): 182-186.

M. A. Pallis, S. Shende, D. S. L. Wei. 1987. *An Optimal Linear-Time Parallel Parser for Tree Adjoining Languages.* Philadelphia, Pennsylvania: Technical Report MS-CIS-87-36, Dept. of Computer and Information Science, Moore School, Univ. of Pennsylvania.

Y. Schabes, A. Joshi. 1988. *An Earley-Type Parsing Algorithm for Tree Adjoining Grammars.* Philadelphia, Pennsylvania: Technical Rep.MS-CIS-88-36, Dept. of Computer and Information Science, Moore School, Univ. of Pennsylvania.

S. M. Shieber. 1985. *Evidence against the Context-Freeness of Natural Language.* Linguistics and Philosophy 8: 333-343.

K. Vijay-Shanker. 1987. *A Study of Tree Adjoining Grammars.* Philadelphia, Pennsylvania: PhD Thesis, Dept. of Computer and Information Science, Moore School, Univ. of Pennsylvania.

K. Vijay-Shanker, A. K. Joshi. 1985. *Some Computational Properties of Tree Adjoining Grammars.* Chicago, Illinois: Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics: 82-93.