BIDIRECTIONAL PARSING OF LEXICALIZED TREE ADJOINING GRAMMARS*

Alberto Lavelli and Giorgio Satta
Istituto per la Ricerca Scientifica e Tecnologica
I - 38050 Povo TN, Italy
e-mail: lavelli/satta@irst.it

Abstract

In this paper a bidirectional parser for Lexicalized Tree Adjoining Grammars will be presented. The algorithm takes advantage of a peculiar characteristic of Lexicalized TAGs, i.e. that each elementary tree is associated with a lexical item, called its *anchor*. The algorithm employs a mixed strategy: it works bottom-up from the lexical anchors and then expands (partial) analyses making top-down predictions. Even if such an algorithm does not improve the worst-case time bounds of already known TAGs parsing methods, it could be relevant from the perspective of linguistic information processing, because it employs lexical information in a more direct way.

1. Introduction

Tree Adjoining Grammars (TAGs) are a formalism for expressing grammatical knowledge that extends the domain of locality of context-free grammars (CFGs). TAGs are tree rewriting systems specified by a finite set of *elementary trees* (for a detailed description of TAGs, see (Joshi, 1985)). TAGs can cope with various kinds of unbounded dependencies in a direct way because of their extended domain of locality; in fact, the elementary trees of TAGs are the appropriate domains for characterizing such dependencies. In (Kroch and Joshi, 1985) a detailed discussion of the linguistic relevance of TAGs can be found.

Lexicalized Tree Adjoining Grammars (Schabes *et al.*, 1988) are a refinement of TAGs such that each elementary tree is associated with a lexical item, called the *anchor* of the tree. Therefore, Lexicalized TAGs conform to a common tendency in modern theories of grammar, namely the attempt to embed grammatical information within lexical items. Notably, the association between elementary trees and anchors improves also parsing performance, as will be discussed below.

Various parsing algorithms for TAGs have been proposed in the literature: the worst-case time complexity varies from $O(n^4 \log n)$ (Harbusch, 1990) to $O(n^6)$ (Vijay-Shanker and Joshi, 1985, Lang, 1990, Schabes, 1990) and $O(n^9)$ (Schabes and Joshi, 1988). As for Lexicalized TAGs, in (Schabes *et al.*, 1988) a two step algorithm has been presented: during the first step the trees corresponding to the input string are selected and in the second step the input string is parsed with respect to this set of trees. Another paper by Schabes and Joshi (1989) shows how parsing strategies can take advantage of lexicalization in order to improve parsers' performance. Two major advantages have been discussed in the cited work: grammar filtering (the parser can use only a subset of the entire grammar) and bottom-up information (further constraints are imposed on the way trees can be combined). Given these premises and starting from an already known method for bidirectional CF language recognition (Satta and Stock, 1989), it seems quite natural to propose an *anchor-driven* bidirectional parser for Lexicalized TAGs that tries to make more direct use of the information contained within the anchors. The algorithm employs a mixed strategy: it works bottom-up from the lexical anchors and then expands (partial) analyses making top-down predictions.

2. Overview of the Algorithm

The algorithm that will be presented is a recognizer for Tree Adjoining Languages: a parser can be obtained from such a recognizer by additional processing (see final section). As an introduction to the next section, an informal description of the studied algorithm is here presented. We assume the following definition of TAGs.

Definition 1 A *Tree Adjoining Grammar (TAG)* is a 5-tuple $G=(V_N, V_\Sigma, S, I, A)$, where V_N is a finite set of non-terminal symbols, V_Σ is a finite set of terminal symbols, $S \in V_N$ is the start symbol, I and A are two finite sets of trees, called *initial trees* and *auxiliary trees* respectively. The trees in the set $I \cup A$ are called *elementary trees*.

We assume that the reader is familiar with the definitions of adjoining operation and foot node (see (Joshi, 1985)).

The proposed algorithm is a tabular method that accepts a TAG G and a string w as input, and decides whether $w \in L(G)$. This is done by recovering (partial) analyses for substrings of w and by combining them. More precisely, the algorithm factorizes analyses of derived trees by employing a specific structure called *state*. Each state retains a pointer to a node n in some tree $\alpha \in I \cup A$, along with two additional pointers (called *ldot* and *rdot*) to n itself or to

*Part of this work was done while Giorgio Satta was completing his Doctoral Dissertation at the University of Padova (Italy). We would like to thank Yves Schabes for his valuable comments. We would also like to thank Anne Abeillé. All errors are of course our own.

its children in α. Let α_n be a tree obtained from the maximal subtree of α with root n, by means of some adjoining operations. Informally speaking and with a little bit of simplification, the two following cases are possible. First, if *ldot, rdot*$\neq n$, state s indicates that the part of α_n dominated by the nodes between *ldot* and *rdot* has already been analyzed by the algorithm. Second, if *ldot=rdot=n*, state s indicates that the whole of α_n has already been analyzed, including possible adjunctions to its root n.

Each state s will be inserted into a recognition matrix T, which is a square matrix indexed from 0 to n_w, where n_w is the length of w. If state s belongs to the component $t_{i,j}$ of T, the partial analysis (the part of α_n) represented by s subsumes the substring of w that starts from position i and ends at position j, except for the items dominated by a possible foot node in α_n (this is explicitly indicated within s).

The algorithm performs the analysis of w starting from the anchor node of every tree in G whose category is the same as an item in w. Then it tries to extend each partial analysis so obtained, by climbing each tree along the path that connects the anchor node to the root node; in doing this, the algorithm recognizes all possible adjunctions that are present in w. Most important, every subtree γ of a tree derived from $\alpha \in I \cup A$, such that γ does not contain the anchor node of α, is predicted and analyzed by the algorithm in a top-down fashion, from right to left (left to right) if it is located to the left (right) of the path that connects the anchor node to the root node in α.

The combinations of partial analyses (states) and the introduction of top-down prediction states is carried out by means of the application of six procedures that will be defined below. Each procedure applies to some states, trying to "move" outward one of the two additional pointers within each state.

The algorithm stops when no state in T can be further expanded. If some state has been obtained that subsumes the input string and that represents a complete analysis for some tree with the root node of category S, the algorithm succeeds in the recognition.

3. The Algorithm

In the following any (elementary or derived) tree will be denoted by a pair (N, E), where N is a finite set of *nodes* and E is a set of ordered pairs of nodes, called *arcs*. For every tree $\alpha=(N, E)$, we define five functions of N into $N \cup \{\perp\}$,[1] called *father, leftmost-child, rightmost-child, left-sibling*, and *right-sibling* (with the obvious meanings). For every tree $\alpha=(N, E)$ and every node $n \in N$, a function $domain_\alpha$ is defined such that $domain_\alpha(n)=\beta$, where β is the maximal subtree in α whose root is n.

[1]The symbol "$\perp$" denotes here the undefined element.

For any *TAG* G and for every node n in some tree in G, we will write $cat(n)=X, X \in V_N \cup V_\Sigma$, whenever X is the symbol associated to n in G. For every node n in some tree in G, such that $cat(n) \in V_N$, the set *Adjoin(n)* contains all root nodes of auxiliary trees that can be adjoined to n in G. Furthermore, a function τ is defined such that, for every tree $\alpha \in I \cup A$, it holds that $\tau(\alpha)=n$, where n indicates the anchor node of α. In the following we assume that the anchor nodes in G are not labelled by the null (syntactic) category symbol e. The set of all nodes that dominate the anchor node of some tree in $I \cup A$ will be called *Middle-nodes* (anchor nodes included); for every tree $\alpha=(N, E)$, the nodes $n \in N$ in *Middle-nodes* divide α in two (possibly empty) left and right portions. The set *Left-nodes* (*Right-nodes*) is defined as the set of all nodes in the left (right) portion of some tree in $I \cup A$. Note that the three sets *Middle-nodes, Left-nodes* and *Right-nodes* constitute a partition of the set of all nodes of trees in $I \cup A$. The set of all foot nodes in the trees in A will be called *Foot-nodes*.

Let $w=a_1 \ldots a_{n_w}$, $n_w \geq 1$, be a symbol string; we will say that n_w is the length of w.

Definition 2 A *state* is defined to be any 8-tuple $[n, ldot, lpos, rdot, rpos, f_l, f_r, m]$ such that:
$n, ldot, rdot$ are nodes in some tree $\alpha \in I \cup A$;
$lpos, rpos \in \{left, right\}$;
f_l, f_r are either the symbol "-" or indices in the input string such that $f_l \leq f_r$;
$m \in \{-, rm, lm\}$.

The first component in a state s indicates a node n in some tree α, such that s represents some partial analysis for the subtree $domain_\alpha(n)$. The second component (*ldot*) may be n or one of its children in α: if $lpos=left$, $domain_\alpha(ldot)$ is included in the partial analysis represented by s, otherwise it is not. The components *rdot* and *rpos* have a symmetrical interpretation. The pair f_l, f_r represents the part of the input string that is subsumed by the possible foot node in $domain_\alpha(n)$. A binary operator indicated with the symbol $\oplus$ is defined to combine the components f_l, f_r in different states; such an operator is defined as follows: $f \oplus f'$ equals f if $f' = -$, it equals f' if $f = -$, and it is undefined otherwise. Finally, the component m is a marker that will be used to block expansion at one side for a state that has already been subsumed at the other one. This particular technique is called *subsumption test* and is discussed in (Satta and Stock, 1989). The subsumption test has the main purpose of blocking analysis proliferation due to the bidirectional behaviour of the method.

Let I_S be the set of all possible states; we will use a particular equivalence relation $Q \subseteq I_S \times I_S$ defined as follows. For any pair of states s, s', sQs' holds if and only if every component in s but the last one (the m component) equals the corresponding

component in s'.

The algorithm that will be presented employs the following function.

Definition 3 A function F is defined as follows:[2]

$F: V_\Sigma \rightarrow \mathcal{P}(I_S)$

$F(a) = \{s \mid s=[father(n), n, left, n, right, -, -, -], cat(n)=a$ and $\tau(\alpha)=n$ for some tree $\alpha \in I \cup A\}$

The details of the algorithm are as follows.

Algorithm 1

Let $G=(V_N, V_\Sigma, S, I, A)$ be a *TAG* and let $w=a_1 \ldots a_{n_w}$, $n_w \geq 1$, be any string in V_Σ^*. Let T be a recognition matrix of size $(n_w+1)\times(n_w+1)$ whose components $t_{i,j}$ are indexed from 0 to n_w for both sides. Develop matrix T in the following way (a new state s is added to some entry in T only if sQs_q does not hold for any state s_q already present in that entry).

1. For every state $s \in F(a_i)$, $1 \leq i \leq n_w$, add s to $t_{i-1,i}$.
2. Process each state s added to some entry in T by means of the following procedures (in any order): *Left-expander(s)*, *Right-expander(s)*, *Move-dot-left(s)*, *Move-dot-right(s)*, *Completer(s)*, *Adjoiner(s)*; until no state can be further added.
3. if $s=[n, n, left, n, right, -, -, -] \in t_{0,n_w}$ for some node n such that $cat(n)=S$ and n is the root of a tree in I, then *output(true)*, else *output(false)*. ❑

The six procedures mentioned above are defined in the following.

Procedure 1 *Left-expander*

Input A state $s=[n, ldot, lpos, rdot, rpos, f_l, f_r, m]$ in $t_{i,j}$.

Precondition $m \neq lm$, $ldot \neq n$ and $lpos=right$.

Description

Case 1: $ldot \in V_N$, $ldot \notin Foot\text{-}nodes$.

Step 1: For every state $s''=[ldot, ldot, left, ldot, right, f_l'', f_r'', -]$ in $t_{i',i}$, $i' \leq i$, add state $s'=[n, ldot, left, rdot, rpos, f_l \oplus f_l'', f_r \oplus f_r'', -]$ to $t_{i',j}$; set $m=rm$ in s if left-expansion is successful;

Step 2: Add state $s'=[ldot, ldot, right, ldot, right, -, -, -]$ to $t_{i,i}$. For every state $s''=[n'', n'', left, n'', right, f_l'', f_r'', -]$ in $t_{i',i}$, $i' < i$, $n'' \in Adjoin(ldot)$, add state $s'=[ldot, ldot, right, ldot, right, -, -, -]$ to $t_{f_r'' f_r''}$.

Case 2: $ldot \in V_\Sigma$.[3]

If $a_i=cat(ldot)$, add state $s'=[n, ldot, left, rdot, rpos, f_l, f_r, -]$ to $t_{i-1,j}$ (if $cat(ldot)=e$, i.e. the null category symbol, add state s' to $t_{i,j}$); set $m=rm$ in s if left-expansion is successful.

Case 3: $ldot \in Foot\text{-}nodes$.

Add state $s'=[n, ldot, left, rdot, rpos, i', i, -]$ to $t_{i',j}$, for every $i' \leq i$, and set $m=rm$ in s. ❑

Procedure 2 *Right-expander*

Input A state $s=[n, ldot, lpos, rdot, rpos, f_l, f_r, m]$ in $t_{i,j}$.

Precondition $m \neq rm$, $rdot \neq n$ and $rpos=left$.

Description

Case 1: $rdot \in V_N$, $rdot \notin Foot\text{-}nodes$.

Step 1: For every state $s''=[rdot, rdot, left, rdot, right, f_l'', f_r'', -]$ in $t_{j,j'}$, $j \leq j'$, add state $s'=[n, ldot, lpos, rdot, right, f_l \oplus f_l'', f_r \oplus f_r'', -]$ to $t_{i,j'}$; set $m=lm$ in s if left-expansion is successful;

Step 2: Add state $s'=[rdot, rdot, left, rdot, left, -, -, -]$ to $t_{j,j}$. For every state $s''=[n'', n'', left, n'', right, f_l'', f_r'', -]$ in $t_{j,j'}$, $j < j'$, $n'' \in Adjoin(rdot)$, add state $s'=[rdot, rdot, left, rdot, left, -, -, -]$ to $t_{f_r'' f_r''}$.

Case 2: $rdot \in V_\Sigma$.[4]

If $a_{j+1}=cat(rdot)$, add state $s'=[n, ldot, lpos, rdot, right, f_l, f_r, -]$ to $t_{i,j+1}$ (if $cat(rdot)=e$, i.e. the null category symbol, add state s' to $t_{i,j}$); set $m=lm$ in s if right-expansion is successful.

Case 3: $rdot \in Foot\text{-}nodes$.

Add state $s'=[n, ldot, lpos, rdot, right, j, j', -]$ to $t_{i,j'}$, for every $j \leq j'$, and set $m=lm$ in s. ❑

Procedure 3 *Move-dot-left*

Input A state $s=[n, ldot, lpos, rdot, rpos, f_l, f_r, m]$ in $t_{i,j}$.

Precondition $m \neq lm$, and $ldot \neq n$, $lpos=left$, or $ldot=n$, $lpos=right$.

Description

Case 1: $lpos=right$.

Add state $s'=[n, rightmost\text{-}child(n), right, rdot, rpos, f_l, f_r, -]$ to $t_{i,j}$; set $m=rm$ in s;

Case 2: $lpos=left$, $left\text{-}sibling(n) \neq \perp$.

Add state $s'=[n, left\text{-}sibling(ldot), right, rdot, rpos, f_l, f_r, -]$ to $t_{i,j}$; set $m=rm$ in s.

Case 3: $lpos=left$, $left\text{-}sibling(ldot)=\perp$.

Add state $s'=[n, n, left, rdot, rpos, f_l, f_r, -]$ to $t_{i,j}$ and set $m=rm$ in s. ❑

Procedure 4 *Move-dot-right*

Input A state $s=[n, ldot, lpos, rdot, rpos, f_l, f_r, m]$ in $t_{i,j}$.

Precondition $m \neq rm$, and $rdot \neq n$, $rpos=right$, or $rdot=n$, $rpos=left$.

Description

Case 1: $rpos=left$.

Add state $s'=[n, ldot, lpos, leftmost\text{-}child(n), left, f_l, f_r, -]$ to $t_{i,j}$; set $m=lm$ in s;

Case 2: $rpos=right$, $right\text{-}sibling(n) \neq \perp$.

Add state $s'=[n, ldot, lpos, right\text{-}sibling(rdot), left, f_l, f_r, -]$ to $t_{i,j}$; set $m=lm$ in s.

Case 3: $rpos=right$, $right\text{-}sibling(ldot)=\perp$.

Add state $s'=[n, ldot, lpos, n, right, f_l, f_r, -]$ to $t_{i,j}$ and set $m=lm$ in s. ❑

---

[2] Given a generic set $\mathcal{A}$, the symbol $\mathcal{P}(\mathcal{A})$ denotes the set of all the subsets of $\mathcal{A}$ (the *power set* of $\mathcal{A}$).

[3] We assume that a_0 is undefined.

[4] See note 3.

Procedure 5 *Completer*
Input A state $s=[n, n, left, n, right, f_l, f_r, m]$ in $t_{i,j}$.
Precondition n is not the root of an auxiliary tree.
Description
Case 1: $n \in$ *Middle-nodes*.
Add state $s'=[father(n), n, left, n, right, f_l, f_r, -]$ to $t_{i,j}$.
Case 2: $n \in$ *Left-nodes*.
For every state $s''=[n'', ldot'', right, rdot, rpos, f_l'', f_r'', m'']$ in $t_{j,j'}$, $j'>j$, such that $ldot''=n$ and $m'' \neq lm$, add state $s'=[n'', ldot'', left, rdot, rpos, f_l \oplus f_l'', f_r \oplus f_r'', -]$ in $t_{i,j'}$; if left-expansion is successful for state s'', set $m''=rm$ in s''.
Case 3: $n \in$ *Right-nodes*.
For every state $s''=[n'', ldot, lpos, rdot'', left, f_l'', f_r'', m'']$ in $t_{i',i}$, $i'<i$, such that $rdot''=n$ and $m'' \neq rm$, add state $s'=[n'', ldot, lpos, rdot'', right, f_l \oplus f_l'', f_r \oplus f_r'', -]$ in $t_{i',j}$; if right-expansion is successful for state s'', set $m''=lm$ in s''. ❑

Procedure 6 *Adjoiner*
Input A state $s=[n, n, left, n, right, f_l, f_r, m]$ in $t_{i,j}$.
Precondition Void.
Description
Case 1: apply always.
For every state $s''=[n'', n'', left, n'', right, i, j, -]$ in $t_{i',j'}$, $i' \leq i$, $j' \geq j$, $n'' \in Adjoin(n)$, add state $s'=[n, n, left, n, right, f_l, f_r, -]$ to $t_{i',j'}$.
Case 2: n is the root of an auxiliary tree.
Step 1: For every state $s''=[n'', n'', left, n'', right, f_l'', f_r'', -]$ in t_{f_l,f_r}, such that $n \in Adjoin(n'')$, add state $s'=[n'', n'', left, n'', right, f_l'', f_r'', -]$ to $t_{i,j}$.
Step 2: For every state $s''=[n'', ldot'', right, rdot, rpos, f_l'', f_r'', m'']$ in $t_{j,j'}$, $j'>j$, such that $n \in Adjoin(ldot'')$ and $m'' \neq lm$, add state $s'=[ldot'', ldot'', right, ldot'', right, -, -, -]$ to t_{f_r,f_r}.
Step 3: For every state $s''=[n'', ldot, lpos, rdot'', left, f_l'', f_r'', m'']$ in $t_{i',i}$, $i'<i$, such that $n \in Adjoin(rdot'')$ and $m'' \neq rm$, add state $s'=[rdot'', rdot'', left, rdot'', left, -, -, -]$ to t_{f_l,f_l}. ❑

4. Formal Results

Some definitions will be introduced in the following, in order to present some interesting properties of Algorithm 1. Formal proofs of the statements below can be found in (Satta, 1990).

Let n be a node in some tree $\alpha \in I \cup A$. Each state $s=[n, ldot, lpos, rdot, rpos, f_l, f_r, m]$ in I_S identifies a *tree forest* $\phi(s)$ composed of all maximal subtrees in α whose roots are "spanned" by the two positions *ldot* and *rdot*. If $ldot \neq n$, we assume that the maximal subtree in α whose root is *ldot* is included in $\phi(s)$ if and only if *lpos*=left (the mirror case holds w.r.t. *rdot*). We define the *subsumption relation* $\leq$ on I_S as follows: $s \leq s'$ iff state s has the same first component as state s' and $\phi(s)$ is included in $\phi(s')$. We also say that a forest $\phi(s)$ derives a forest ψ ($\phi(s) \stackrel{*}{\Rightarrow} \psi$) whenever ψ can be obtained from $\phi(s)$ by means of some adjoining operations. Finally, E denotes the immediate dominance relation on nodes of $\alpha \in I \cup A$, and $\pi(\alpha)$ denotes the foot node of α (if $\alpha \in A$). The following statement characterizes the set of all states inserted in T by Algorithm 1.

Theorem 1 Let n be a node in $\alpha \in I \cup A$ and let n' be the lowest node in α such that $n' \in$ *Middle-nodes* and $(n, n') \in E^*$; let also $s=[n, ldot, lpos, rdot, rpos, f_l, f_r, m]$ be a state in I_S. Algorithm 1 inserts a state s', $s \leq s'$, in $t_{i-h_1,j+h_2}$, $h_1, h_2 \geq 0$, if and only if one of the following conditions is met:

(i) $n \in$ *Middle-nodes* ($n'=n$) and $\phi(s) \stackrel{*}{\Rightarrow} \psi$, where ψ spans $a_{i+1} \ldots a_j$ (with the exception of string $a_{f_l+1} \ldots a_{f_r}$ if $\pi(\alpha)$ is included in $\phi(s)$) (see Figure 1);

(ii) $n \in$ *Left-nodes*, $s=s'$, $h_1=h_2=0$ and $\phi(s) \stackrel{*}{\Rightarrow} \psi'$, where ψ' spans $a_{i+1} \ldots a_j$ (with the exception of string $a_{f_l+1} \ldots a_{f_r}$ if $\pi(\alpha)$ is included in $\phi(s)$). Moreover, n' is the root of a (maximal) subtree τ in α such that $\tau \stackrel{*}{\Rightarrow} \psi$, ψ strictly includes ψ' and every tree $\beta \in A$ that has been adjoined to some node in the path from n' to n spans a string that is included in $a_1 \ldots a_i$ (see Figure 2);

(iii) the symmetrical case of (ii).

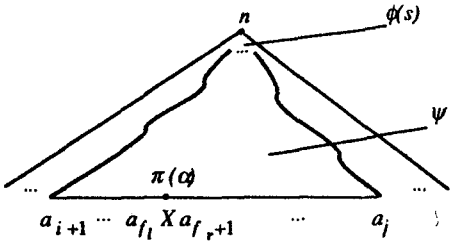

Figure 1.

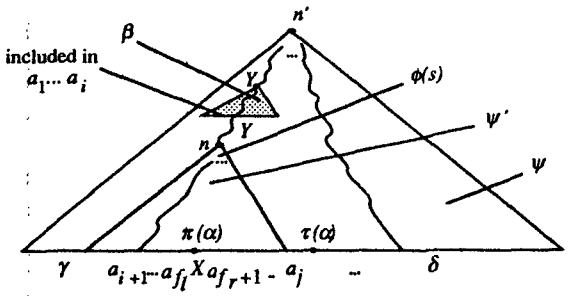

Figure 2.

In order to present the computational complexity of Algorithm 1, some norms for TAGs are here introduced. Let $\mathcal{A}$ be a set of nodes in some trees of a TAG G; we define

$$|G|_{\mathcal{A},k} = \sum_{n \in \mathcal{A}} |children(n)|^k .$$

The following result refers to the *Random Access Machine* model of computation.

Theorem 2 If some auxiliary structures (vector of lists) are used by Algorithm 1 for the bookkeeping of all states that correspond to completely analyzed auxiliary trees, a string can be recognized in $O(n^6 \cdot |A| \cdot \max\{|G|_{\mathcal{N},\mathcal{M},1}+|G|_{\mathcal{M},2}\})$ time, where $\mathcal{M}$ =*Middle-nodes* and $\mathcal{N}$ denotes the set of all nodes in the trees of G.

5. A Linguistic Example

In order to gain a better understanding of Algorithm 1 and to emphasize the linguistic relevance of TAGs, we present a running example. In the following we assume the formal framework of X-bar Theory (Jackendoff, 1977). Given the sentence:

(1) Gianni incontra Maria per caso
lit. Gianni meets Maria by chance

we will propose here the following analysis (see Figure 4):

(2) $[_{CP}\,[_{C'}\,[_{IP}\,[_{NP}$ Gianni$]\,[_{I'}$ incontra$_i\,[_{VP^*}\,[_{VP}\,[_{V'}\,e_i\,[_{NP}$ Maria$]]]]\,[_{PP}$ per caso$]]]]]]$

Note that the Verb *incontra* has been moved to the Inflection position. Therefore, the PP adjunction stretches the dependency between the Verb *incontra* and its Direct Object *Maria*. These cases may raise some difficulties in a context-free framework, because the lack of the head within its constituent makes the task of predicting the object(s) rather inefficient.

Assume a TAG $G=(V_N, V_\Sigma, S, I, A)$, where V_N={IP, I', VP, V', NP}, V_Σ={*Gianni*, *Maria*, *incontra*, PP}, $I=\{\alpha\}$ and $A=\{\beta\}$ (see Figure 3; each node has been paired with an integer which will be used as its address). In order to simplify the computation, we have somewhat reduced the initial tree α and we have considered the constituent PP as a terminal symbol. In Figure 4 the whole analysis tree corresponding to (2) is reported.

Let $\tau(\alpha)=5$, $\tau(\beta)=13$; from Definition 3 it follows that:

$F(5)=\{[4, 5, \mathit{left}, 5, \mathit{right}, -, -, -]\}$,
$F(13)=\{[11, 13, \mathit{left}, 11, \mathit{right}, -, -, -]\}$.

A run of Algorithm 1 on sentence (1) is simplified in the following steps (only relevant steps are reported).
First of all, the two anchors are recognized:

1) s_1=[4, 5, *left*, 5, *right*, -, -, -] is inserted in $t_{1,2}$ and s_2=[11, 13, *left*, 13, *right*, -, -, -] is inserted in $t_{3,4}$, by line 1 of the algorithm.

Then, auxiliary tree β is recognized in the following steps:

2) s_3=[11, 12, *right*, 13, *right*, -, -, -] is inserted in $t_{3,4}$ and m is set to *rm* in state s_2, by Case 2 of the *move-dot-left* procedure;
3) s_4=[11, 12, *left*, 13, *right*, 2, 3, -] is inserted in $t_{2,4}$ and m is set to *rm* in state s_3, by Case 3 of the *left-expander* procedure;
4) s_5=[11, 11, *left*, 13, *right*, 2, 3, -] is inserted in $t_{2,4}$ and m is set to *rm* in state s_4, by Case 3 of the *move-dot-left* procedure;
5) s_6=[11, 11, *left*, 11, *right*, 2, 3, -] is inserted in $t_{2,4}$ and m is set to *lm* in state s_5, by Case 3 of the *move-dot-right* procedure.

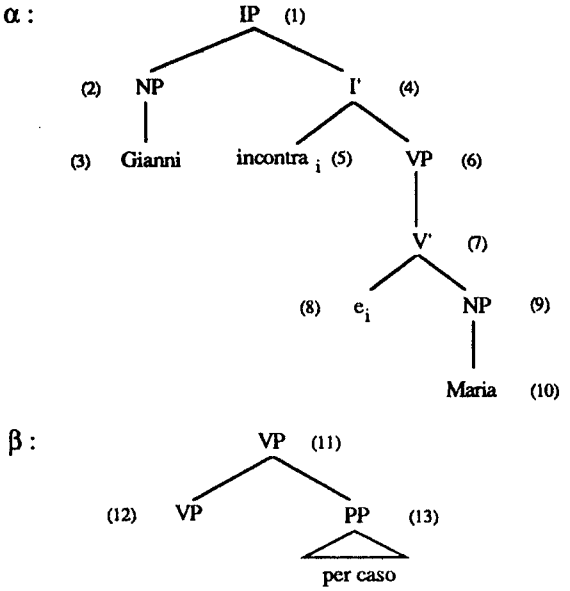

Figure 3.

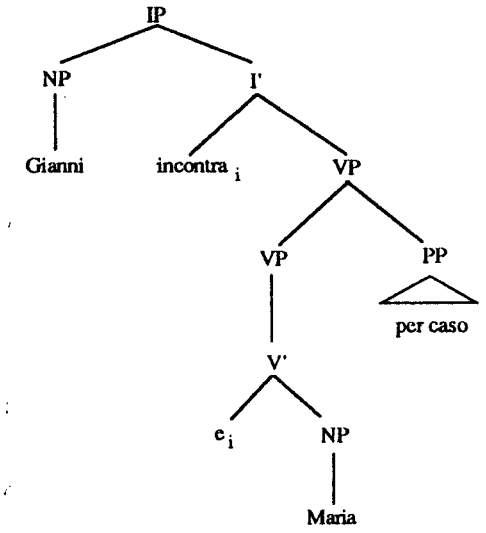

Figure 4.

After the insertion of state s_7=[4, 5, *left*, 6, *left*, -, -, -] in $t_{1,2}$ by Case 2 of the *move-dot-right* procedure, the VP node (6) is hypothesized by Case 1 (Step 2, *via* state s_6) of the *right-expander* procedure with the insertion of state s_8=[6, 6, *left*, 6, *left*, -, -, -] in $t_{2,2}$. The whole recognition of node (6) takes place with the insertion of state s_9=[6, 6, *left*, 6, *right*, -, -, -] in $t_{2,3}$. Then we have the following step:

6) s_{10}=[6, 6, *left*, 6, *right*, -, -, -] is inserted in

$t_{2,4}$, by the *adjoiner* procedure.
The analysis proceeds working on tree α and reaching a final configuration in which state s_{11}=[1, 1, *left*, 1, *right*, -, -, -] belongs to $t_{0,4}$.

6. Discussion

Within the perspective of Lexicalized TAGs, known methods for TAGs recognition/parsing present some limitations: these methods behave in a left-to-right fashion (Schabes and Joshi, 1988) or they are purely bottom-up (Vijay-Shanker and Joshi, 1985, Harbusch, 1990), hence they cannot take advantage of anchor information in a direct way. The presented algorithm directly exploits both the advantages of lexicalization mentioned in the paper by Schabes and Joshi (1989), i.e. grammar filtering and bottom-up information. In fact, such an algorithm starts partial analyses from the anchor elements, directly selecting the relevant trees in the grammar, and then it proceeds in both directions, climbing to the roots of these trees and predicting the rest of the structures in a top-down fashion. These capabilities make the algorithm attractive from the perspective of linguistic information processing, even if it does not improve the worst-case time bounds of already known TAGs parsers.

The studied algorithm recognizes auxiliary trees without considering the substring dominated by the foot node, as is the case of the CYK-like algorithm in Vijay-Shanker and Joshi (1985). More precisely, Case 3 in the procedure *Left-expander* nondeterministically jumps over such a substring. Note that the alternative solution, which consists in waiting for possible analyses subsumed by the foot node, would prevent the algorithm from recognizing particular configurations, due to the bidirectional behaviour of the method (examples are left to the reader). On the contrary, Earley-like parsers for TAGs (Lang, 1990, Schabes, 1990) do care about substrings dominated by the foot node. However, these algorithms are forced to start at each foot node the recognition of all possible subtrees of the elementary trees whose roots can be the locus of an adjunction.

In this work, we have discussed a theoretical schema for the parser, in order to study its formal properties. In practical cases, such an algorithm could be considerably improved. For example, the above mentioned guess in Case 3 of the procedure *Left-expander* could take advantage of look-ahead techniques. So far, we have not addressed topics such as substitution or on-line recognition. Our algorithm can be easily modified in these directions, adopting the same proposals advanced in (Schabes and Joshi, 1988).

Finally, a parser for Lexicalized TAGs can be obtained from Algorithm 1. To this purpose, it suffices to store elements in I_S into the recognition matrix T along with a list of pointers to those entries that caused such elements to be placed in the matrix. Using this additional information, it is not difficult to exhibit an algorithm for the construction of the desired parser(s).

References

Harbusch, Karin, 1990. An Efficient Parsing Algorithm for TAGs. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*. Pittsburgh, PA.

Jackendoff, Ray, 1977. *X-bar Syntax: A Study of Phrase Structure*. The MIT Press, Cambridge, MA.

Joshi, Aravind K., 1985. Tree Adjoining Grammars: How Much Context-Sensitivity Is Required to Provide Reasonable Structural Descriptions?. In: D. Dowty *et al.* (eds). *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*. Cambridge University Press, New York, NY.

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

Lang, Bernard, 1990. The Systematic Construction of Earley Parsers: Application to the Production of $O(n^6)$ Earley Parsers for Tree Adjoining Grammars. In *Proceedings of the 1st International Workshop on Tree Adjoining Grammars*. Dagstuhl Castle, F.R.G..

Satta, Giorgio, 1990. Aspetti computazionali della Teoria della Reggenza e del Legamento. Doctoral Dissertation, University of Padova, Italy.

Satta, Giorgio and Stock, Oliviero, 1989. Head-Driven Bidirectional Parsing: A Tabular Method. In *Proceedings of the 1st International Workshop on Parsing Technologies*. Pittsburgh, PA.

Schabes, Yves, 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD Thesis, Department of Computer and Information Science, University of Pennsylvania.

Schabes, Yves; Abeillé, Anne and Joshi, Aravind K., 1988. Parsing Strategies for 'Lexicalized' Grammars: Application to Tree Adjoining Grammars. In *Proceedings of the 12th International Conference on Computational Linguistics*. Budapest, Hungary.

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

Schabes, Yves and Joshi, Aravind K., 1989. The Relevance of Lexicalization to Parsing. In *Proceedings of the 1st International Workshop on Parsing Technologies*. Pittsburgh, PA. To also appear under the title: Parsing with Lexicalized Tree Adjoining Grammar. In: M. Tomita (ed.). *Current Issues in Parsing Technologies*. The MIT Press.

Vijay-Shanker, K. and Joshi, Aravind K., 1985. Some Computational Properties of Tree Adjoining Grammars. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*. Chicago, IL.