The Computational Complexity of the Correct-Prefix Property for TAGs

Mark-Jan Nederhof*
German Research Center for Artificial Intelligence

A new upper bound is presented for the computational complexity of the parsing problem for TAGs, under the constraint that input is read from left to right in a way that errors in the input are observed as soon as possible, which is called the "correct-prefix property."

The former upper bound was $\mathcal{O}(n^9)$, which is now improved to $\mathcal{O}(n^6)$, which is the same as that of practical parsing algorithms for TAGs without the additional constraint of the correct-prefix property.

1 Introduction

Traditionally, parsers and recognizers for regular and context-free languages process input from left to right. If a syntax error occurs in the input they often detect that error immediately after its position is reached. The position of the syntax error can be defined as the rightmost symbol of the shortest prefix of the input which cannot be extended to be a correct sentence in the language L.

In formal notation, this prefix for a given erroneous input $w \notin L$ is defined as the string va, where $w = vax$, some x, such that $vy \in L$, for some y, but $vaz \notin L$, for any z. (The symbols v, w, ... denote strings, and a denotes an input symbol.) The occurrence of a in w indicates the error position.

If the error is detected as soon as it is reached, then all prefixes of the input that have been processed at preceding stages are **correct** prefixes, or more precisely, they are prefixes of some correct strings in the language. Hence, we speak of the **correct-prefix property**.[1]

An important application can be found in the area of grammar checking: upon finding an ungrammatical sentence in a document, a grammar checker may report to the user the presumed position of the error, obtained from a parsing algorithm with the correct-prefix property.

For context-free and regular languages, the correct-prefix property can be enforced without additional costs of space or time. Surprisingly, it has been claimed by Schabes and Waters (1995) that this property is problematic for the mildly context-sensitive languages represented by tree-adjoining grammars (TAGs): the best practical parsing algorithms for TAGs have time complexity $\mathcal{O}(n^6)$ (Vijay-Shankar and Joshi, 1985) (see Satta (1994) and Rajasekaran and Yooseph (1995) for lower theoretical upper bounds), whereas the only published algorithm with the correct-prefix property, viz. that by Schabes and Joshi (1988), has complexity $\mathcal{O}(n^9)$.

In this paper we present an algorithm that fulfils the correct-prefix property and

* DFKI, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany. E-mail: nederhof@dfki.de

1 We adopt this term from Sippu and Soisalon-Soininen (1988). In some publications, the term **valid prefix property** is used.

operates in $\mathcal{O}(n^6)$ time. This algorithm merely recognizes input, but it can be extended to be a parsing algorithm, with the ideas from Schabes (1994), which also suggests how it can be extended to handle substitution in addition to adjunction. The complexity results carry over to linear indexed grammars, combinatory categorial grammars and head grammars, since these formalisms are equivalent to TAGs (Vijay-Shanker and Weir, 1993; Vijay-Shanker and Weir, 1994).

Section 3 presents the actual algorithm, after the necessary notation has been discussed in Section 2. The correctness proofs are discussed in Section 4. Section 5 presents the time complexity. The ideas from this paper give rise to a number of open questions, as discussed in Section 6.

2 Definitions

Our definition of TAGs simplifies the explanation of the algorithm, but differs slightly from standard treatment such as that by Joshi (1987).

A tree-adjoining grammar is a 4-tuple (Σ, NT, I, A), where Σ is the set of **terminals**, I is the set of **initial trees** and A is the set of **auxiliary trees**. We refer to the trees in $I \cup A$ as **elementary trees**. The set NT, the set of **nonterminals**, does not play any role in this paper.

We refer to the root of an elementary tree t as R_t. Each auxiliary tree has exactly one distinguished leaf, which is called the **foot**. We refer to the foot of an auxiliary tree t as F_t.

We use variables N and M to range over nodes in elementary trees. We assume that the sets of nodes belonging to distinct elementary trees are pairwise disjoint.

For each leaf N in an elementary tree, except when it is a foot, we define $label(N)$ to be the label of the node, which is either a terminal from Σ or the empty string ϵ. For all other nodes, *label* is undefined.

For each node N that is not a leaf or that is a foot, $Adj(N)$ is the set of auxiliary trees that can be adjoined at N, plus possibly the special element **nil**. For all other nodes, *Adj* is undefined. If a set $Adj(N)$ contains **nil**, then this indicates that adjunction at N is not obligatory.

For each non-leaf node N we define $children(N)$ as the (non-empty) list of daughter nodes. For all other nodes, *children* is undefined.

An example of a TAG is given in Figure 1.

The language described by a TAG is given by the set of strings that are the yields of **derived trees**. A derived tree is obtained from an initial tree, by performing the following operation on each node N, except when it is a leaf. The tree is excised at N, and between the two halves a fresh instance of an auxiliary tree is inserted which is taken from the set $Adj(N)$, or the element **nil** is taken from $Adj(N)$ in which case no new nodes are added to the tree. Insertion of the new auxiliary tree, which will from now on be called **adjunction**, is done in such a way that the bottom half of the excised tree is connected to the foot of the auxiliary tree. The new nodes that are added to the tree as a result, are recursively subjected to the same operation.

This process ends in a complete derived tree once all nodes have been treated.

An example of the derivation of a string is given in Figure 2. We start with initial tree $a1$ and treat R_{a1}, for which we find $Adj(R_{a1}) = \{b2, \mathbf{nil}\}$. We opt to select **nil**, so that no new nodes are added. However in the picture we do split R_{a1} in order to mark it as having been treated. Next we treat N^1_{a1}, and we opt to adjoin $b1$, taken from $Adj(N^1_{a1}) = \{b1, b3\}$. After another "**nil**-adjunction" at R_{b1}, we adjoin $b2$ at N^1_{b1}. Note that this is an obligatory adjunction, since $Adj(N^1_{b1})$ does not contain **nil**. Some more **nil**-adjunctions lead to a derived tree with yield *acdb*, which is therefore in the language

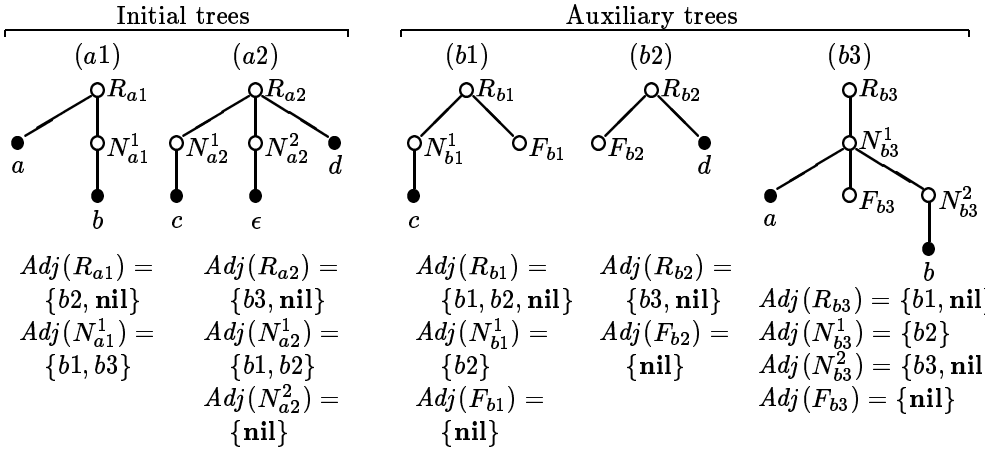


Figure 1
A tree-adjoining grammar

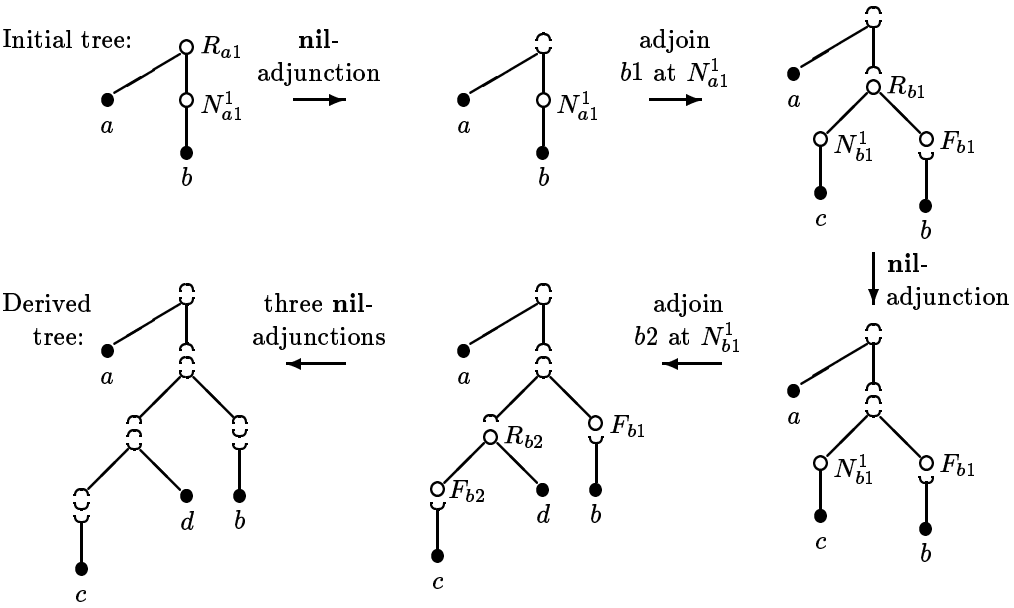


Figure 2
Derivation of the string *acdb*

described by the TAG.

In order to avoid cluttering the picture with details, we have omitted the names of nodes at which (**nil**-)adjunction has been applied. We will reintroduce these names later. A further point worth mentioning is that here we treat the nodes in preorder: we traverse the tree top-down and left-to-right, and perform adjunction at each node during its first visit.[2] Any other strategy would lead to the same set of derived trees, but we chose preorder treatment since this matches the algorithm we will present below.

2 The tree that is being traversed grows in size during the traversal, contrary to traditional usage of the notion of "traversal."

3 The Algorithm

The input to the recognition algorithm is given by the string $a_1 a_2 \cdots a_n$, where n is the length of the input. Integers i such that $0 \leq i \leq n$ will be used to indicate "positions" in the input string. Where we refer to the input between positions i and j we mean the string $a_{i+1} \cdots a_j$.

The algorithm operates by means of least fixed-point iteration: a table is gradually filled with elements derived from other elements, until no more new ones can be found. A certain collection of "steps" indicate how table elements are to be derived from others.[3]

For the description of the steps we use a pseudo-formal notation. Each step consists of a list of antecedents and a consequent. The antecedents are the conditions under which an incarnation of the step is executed. The consequent is a new table element that the step then adds to the parse table, unless of course it is already present. An antecedent may be a table element, in which case the condition that it represents is membership in the table.

The main table elements, or **items**, are 6-tuples $[h,\ N \rightarrow \alpha \bullet \beta,\ i,\ j,\ f_1,\ f_2]$. Here, N is a node from some elementary tree t, and $\alpha\beta$ is the list of the daughter nodes of N. The daughters in α together generate the input between positions i and j. The whole elementary tree generates input from position h onwards.

Internal in the elementary tree there may be adjunctions; in fact, the traversal of the tree (implying (**nil**-)adjunctions at all nodes) has been completed up to the end of α. Furthermore, tree t may itself be an auxiliary tree, in which case it is adjoined in another tree. Then, the foot may be dominated by one of the daughters in α, and the foot generates the part of the input between positions f_1 and f_2. When the tree is not an auxiliary tree, or when the foot is not dominated by one of the daughters in α, then f_1 and f_2 both have the dummy value "$-$".

Whether t is an initial or an auxiliary tree, it is part of a derived tree of which everything to the left of the end of α generates the input between positions 0 and j. The traversal has been completed up to the end of α.

See Figure 3 for a pictorial representation of the meaning of items. We assume R_t and F_t are the root and foot of the elementary tree t to which N belongs; F_t may not exist, as explained above. R is the root of some initial tree. The solid lines indicate what has been established; the dashed lines indicate what is merely predicted. If F_t exists, the subtree below F_t indicates the lower half of the derived tree in which t was adjoined.

The nodes in the shaded areas labelled by I, II and III have not been visited yet by the traversal. In particular it has not yet been established that these parts of the derived tree together generate the input between positions j and n.

For technical reasons, we assume an additional node for each elementary tree t, which we denote by $\top$. This node has only one daughter, viz. the actual root node R_t. We also assume an additional node for each auxiliary tree t, which we denote by $\bot$. This is the unique daughter of the actual foot node F_t; we set $\mathit{children}(F_t) = \bot$.

In summary, an item indicates how a part of an elementary tree contributes to the recognition of some derived tree.

Figure 4 illustrates the items needed for recognition of the derived tree from the running example. We have simplified the notation of items by replacing the names of leaves (other than foot nodes) by their labels.

There is one special kind of item, with only 5 fields instead of 6. This is used as

3 A "step" is more accurately called an "inference rule" in the literature on deductive parsing (Shieber, Schabes, and Pereira, 1995). For the sake of convenience we will apply the shorter term.

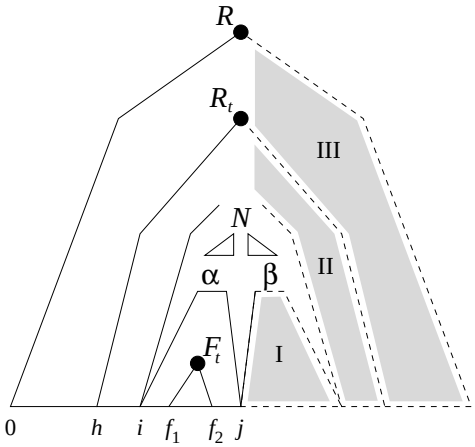
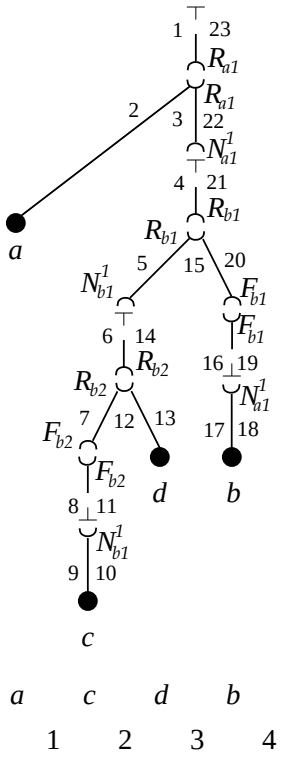


Figure 3
An item $[h,\ N \to \alpha \bullet \beta,\ i,\ j,\ f_1,\ f_2]$



1:	$[0, \top \to \bullet\ R_{a1},\ 0,0,-,-]$	$=$ (**Init**)
2:	$[0, R_{a1} \to \bullet\ a\ N^1_{a1},\ 0,0,-,-]$	$=$ (**Pred 2**)1
3:	$[0, R_{a1} \to a \bullet N^1_{a1},\ 0,1,-,-]$	$=$ (**Scan 1**)2
4:	$[1, \top \to \bullet\ R_{b1},\ 1,1,-,-]$	$=$ (**Pred 1**)3
5:	$[1, R_{b1} \to \bullet\ N^1_{b1}\ F_{b1},\ 1,1,-,-]$	$=$ (**Pred 2**)4
6:	$[1, \top \to \bullet\ R_{b2},\ 1,1,-,-]$	$=$ (**Pred 1**)5
7:	$[1, R_{b2} \to \bullet\ F_{b2}\ d,\ 1,1,-,-]$	$=$ (**Pred 2**)6
8:	$[1, F_{b2} \to \bullet \perp,\ 1,1,-,-]$	$=$ (**Pred 2**)7
9:	$[1, N^1_{b1} \to \bullet\ c,\ 1,1,-,-]$	$=$ (**Pred 3**)8 $+$ 5
10:	$[1, N^1_{b1} \to c \bullet,\ 1,2,-,-]$	$=$ (**Scan 1**)9
11:	$[1, F_{b2} \to \perp \bullet,\ 1,2,1,2]$	$=$ (**Comp 1**)10 $+$ 8 $+$ 5
12:	$[1, R_{b2} \to F_{b2} \bullet d,\ 1,2,1,2]$	$=$ (**Comp 2**)11 $+$ 7
13:	$[1, R_{b2} \to F_{b2}\ d \bullet,\ 1,3,1,2]$	$=$ (**Scan 1**)12
14:	$[1, \top \to R_{b2} \bullet,\ 1,3,1,2]$	$=$ (**Comp 2**)13 $+$ 6
15*a*:	$[N^1_{b1} \to c \bullet,\ 1,\ 3,-,\ -]$	$=$ (**Adj 0**)14 $+$ 10
15:	$[1, R_{b1} \to N^1_{b1} \bullet F_{b1},\ 1,3,-,-]$	$=$ (**Adj 2**)15*a* $+$ 5
16:	$[1, F_{b1} \to \bullet \perp,\ 3,3,-,-]$	$=$ (**Pred 2**)15
17:	$[0, N^1_{a1} \to \bullet\ b,\ 3,3,-,-]$	$=$ (**Pred 3**)16 $+$ 3
18:	$[0, N^1_{a1} \to b \bullet,\ 3,4,-,-]$	$=$ (**Scan 1**)17
19:	$[1, F_{b1} \to \perp \bullet,\ 3,4,3,4]$	$=$ (**Comp 1**)18 $+$ 16 $+$ 3
20:	$[1, R_{b1} \to N^1_{b1}\ F_{b1} \bullet,\ 1,4,3,4]$	$=$ (**Comp 2**)19 $+$ 15
21:	$[1, \top \to R_{b1} \bullet,\ 1,4,3,4]$	$=$ (**Comp 2**)20 $+$ 4
22*a*:	$[N^1_{a1} \to b \bullet,\ 1,\ 4,-,\ -]$	$=$ (**Adj 0**)21 $+$ 18
22:	$[0, R_{a1} \to a\ N^1_{a1} \bullet,\ 0,4,-,-]$	$=$ (**Adj 2**)22*a* $+$ 3
23:	$[0, \top \to R_{a1} \bullet,\ 0,4,-,-]$	$=$ (**Comp 3**)22 $+$ 1

Figure 4
The items needed for recognition of a derived tree

intermediate result in the adjunctor steps to be discussed in Section 3.5.

3.1 Initializer

The initializer step predicts initial trees t starting at position 0; see Figure 5.

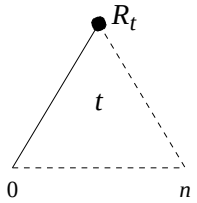


Figure 5
The initialization

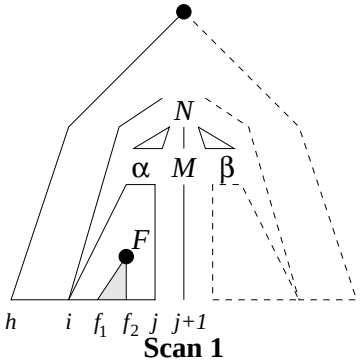


Figure 6
The first scanner step

$$\frac{t \in I}{[0,\ \top \rightarrow \bullet\ R_t,\ 0,\ 0,\ -,\ -]} \qquad \textbf{(Init)}$$

For the running example, item 1 in Figure 4 results from this step.

3.2 Scanner

The scanner steps try to shift the dot rightward in case the next node in line is labelled with a terminal or ϵ, which means the node is a leaf but not a foot. Figure 6 sketches the situation with respect to the input positions mentioned in the step. The depicted structure is part of at least one derived tree consistent with the input between positions 0 and $j+1$, as explained earlier.

$$\frac{\begin{array}{l}[h,\ N \rightarrow \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2],\\ \mathit{label}(M) = a_{j+1}\end{array}}{[h,\ N \rightarrow \alpha M \bullet \beta,\ i,\ j+1,\ f_1,\ f_2]} \qquad \textbf{(Scan 1)}$$

$$\frac{\begin{array}{l}[h,\ N \rightarrow \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2],\\ \mathit{label}(M) = \epsilon\end{array}}{[h,\ N \rightarrow \alpha M \bullet \beta,\ i,\ j,\ f_1,\ f_2]} \qquad \textbf{(Scan 2)}$$

For the running example in Figure 4, **Scan 1** derives among others item 3 from item 2, and item 13 from item 12.

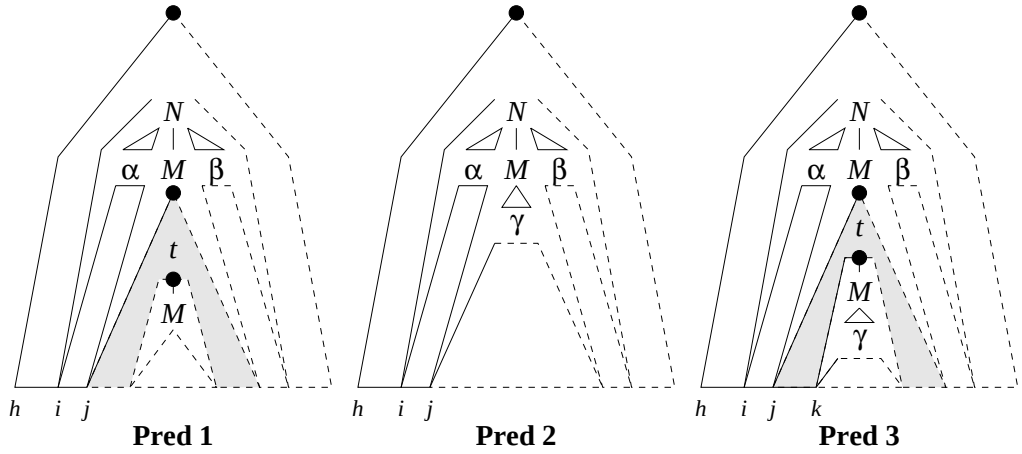


Figure 7
The three predictor steps

3.3 Predictor

The first predictor step predicts a fresh occurrence of an auxiliary tree t, indicated in Figure 7. The second predicts a list of daughters γ lower down in the tree, abstaining from adjunction at the current node M. The third predicts the lower half of a tree in which the present tree t was adjoined.

$$\frac{[h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2],\quad t \in Adj(M)}{[j,\ \top \to \bullet\ R_t,\ j,\ j,\ -,\ -]} \qquad (\textbf{Pred 1})$$

$$\frac{[h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2],\quad \textbf{nil} \in Adj(M),\quad children(M) = \gamma}{[h,\ M \to \bullet\ \gamma,\ j,\ j,\ -,\ -]} \qquad (\textbf{Pred 2})$$

$$\frac{[j,\ F_t \to \bullet\ \bot,\ k,\ k,\ -,\ -],\quad [h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2],\quad t \in Adj(M),\quad children(M) = \gamma}{[h,\ M \to \bullet\ \gamma,\ k,\ k,\ -,\ -]} \qquad (\textbf{Pred 3})$$

For the running example, **Pred 1** derives item 4 from item 3 and item 6 from item 5. **Pred 2** derives among others item 5 from item 4. **Pred 3** derives item 9 from items 8 and 5, and item 17 from items 16 and 3.

3.4 Completer

The first completer step completes recognition of the lower half of a tree in which an auxiliary tree t was adjoined, and asserts recognition of the foot of t; see Figure 8. The second and third completer steps complete recognition of a list of daughter nodes γ, and initiate recognition of the list of nodes β to the right of the mother node of γ.

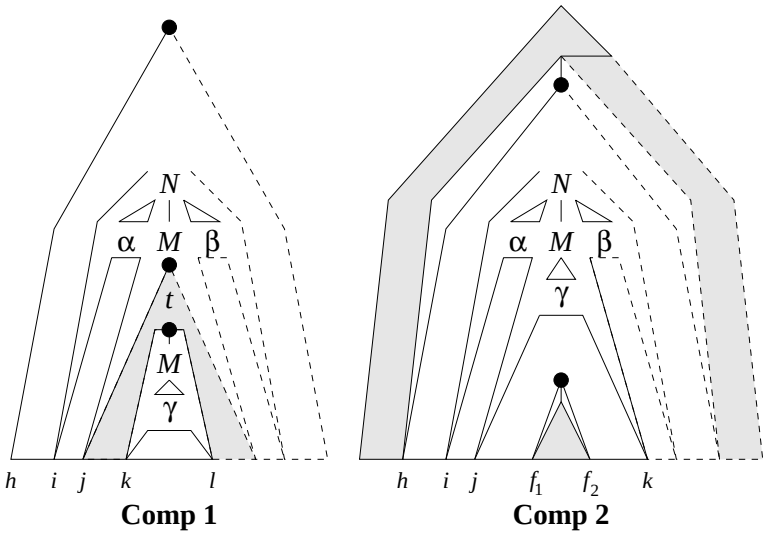


Figure 8
Two of the completer steps

$$\frac{\begin{array}{l}[h,\ M \to \gamma\ \bullet,\ k,\ l,\ f_1',\ f_2'],\\ t \in Adj(M),\\ [j,\ F_t \to \bullet\ \bot,\ k,\ k,\ -,\ -],\\ [h,\ N \to \alpha\ \bullet\ M\beta,\ i,\ j,\ f_1,\ f_2]\end{array}}{[j,\ F_t \to \bot\ \bullet,\ k,\ l,\ k,\ l]} \quad (\textbf{Comp 1})$$

$$\frac{\begin{array}{l}[h,\ M \to \gamma\ \bullet,\ j,\ k,\ f_1,\ f_2],\\ [h,\ N \to \alpha\ \bullet\ M\beta,\ i,\ j,\ -,\ -],\\ M \text{ dominates foot of tree}\end{array}}{[h,\ N \to \alpha M\ \bullet\ \beta,\ i,\ k,\ f_1,\ f_2]} \quad (\textbf{Comp 2})$$

$$\frac{\begin{array}{l}[h,\ M \to \gamma\ \bullet,\ j,\ k,\ -,\ -],\\ [h,\ N \to \alpha\ \bullet\ M\beta,\ i,\ j,\ f_1,\ f_2]\end{array}}{[h,\ N \to \alpha M\ \bullet\ \beta,\ i,\ k,\ f_1,\ f_2]} \quad (\textbf{Comp 3})$$

See Figure 4 for use of these three steps in the running example.

3.5 Adjunctor

The adjunctor steps perform the actual recognition of an adjunction of an auxiliary tree t in another tree at some node M. The first adjunctor step deals with the case that the other tree is again adjoined in a third tree (the two darkly shaded areas in Figure 9) and M dominates the foot node. The second adjunctor step deals with the case that either the other tree is an initial tree, or has the foot elsewhere, i.e. not dominated by M.

The two respective cases of adjunction are realised by step **Adj 0** plus step **Adj 1**, and by step **Adj 0** plus step **Adj 2**. The auxiliary step **Adj 0** introduces items of a somewhat different form than those considered up to now, viz. $[M \to \gamma\ \bullet,\ j,\ k,\ f_1',\ f_2']$. The interpretation is suggested in Figure 10: at M a tree has been adjoined. The adjoined

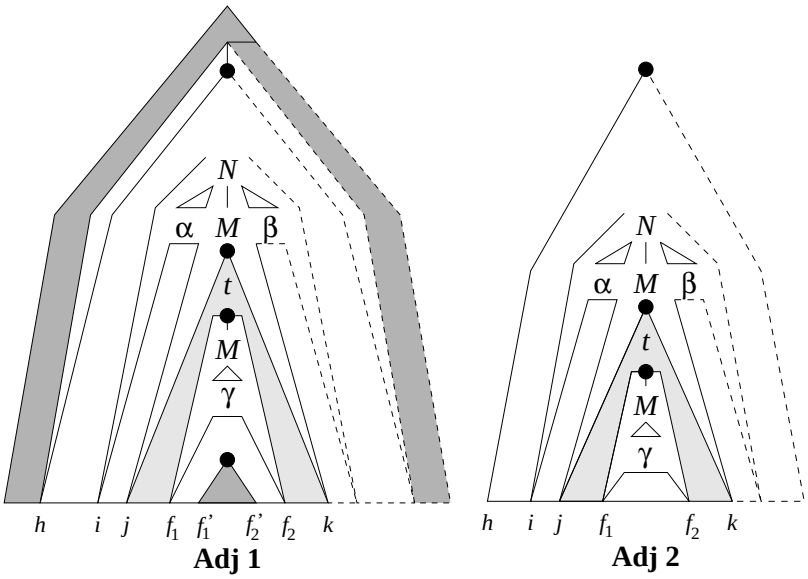


Figure 9
The two adjunctor steps, implicitly combined with **Adj 0**

tree and the lower half of the tree that M occurs in together generate the input from j to k. The depicted structure is part of at least one derived tree consistent with the input between positions 0 and k. In the case that M dominates a foot node, as suggested in the figure, f_1' and f_2' have a value other than "–".

$$\frac{\begin{array}{l}[j,\ \top \to R_t\ \bullet,\ j,\ k,\ f_1,\ f_2],\\ [h,\ M \to \gamma\ \bullet,\ f_1,\ f_2,\ f_1',\ f_2'],\\ t \in Adj(M)\end{array}}{[M \to \gamma\ \bullet,\ j,\ k,\ f_1',\ f_2']} \qquad (\textbf{Adj 0})$$

$$\frac{\begin{array}{l}[M \to \gamma\ \bullet,\ j,\ k,\ f_1',\ f_2'],\\ M \text{ dominates foot of tree } t',\\ [h,\ F_{t'} \to \bot\ \bullet,\ f_1',\ f_2',\ f_1',\ f_2'],\\ [h,\ N \to \alpha\ \bullet\ M\beta,\ i,\ j,\ -,\ -]\end{array}}{[h,\ N \to \alpha M\ \bullet\ \beta,\ i,\ k,\ f_1',\ f_2']} \qquad (\textbf{Adj 1})$$

$$\frac{\begin{array}{l}[M \to \gamma\ \bullet,\ j,\ k,\ -,\ -],\\ [h,\ N \to \alpha\ \bullet\ M\beta,\ i,\ j,\ f_1',\ f_2']\end{array}}{[h,\ N \to \alpha M\ \bullet\ \beta,\ i,\ k,\ f_1',\ f_2']} \qquad (\textbf{Adj 2})$$

For the running example, **Adj 0** derives the intermediate item 15a from items 14 and 10 and from this and item 5, **Adj 2** derives item 15. Similarly, **Adj 0** and **Adj 2** together derive item 22. There are no applications of **Adj 1** in this example.

An alternative formulation of the adjunctor steps, without **Adj 0**, could be the

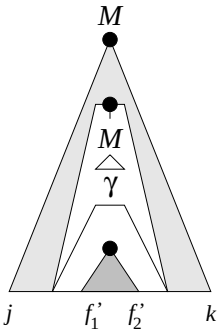


Figure 10
An item $[M \rightarrow \gamma \bullet,\ j,\ k,\ f_1',\ f_2']$

following.

$$\frac{\begin{array}{l}[j,\ \top \rightarrow R_t \bullet,\ j,\ k,\ f_1,\ f_2],\\ [h,\ M \rightarrow \gamma \bullet,\ f_1,\ f_2,\ f_1',\ f_2'],\\ t \in Adj(M),\\ M \text{ dominates foot of tree } t',\\ [h,\ F_{t'} \rightarrow \bot \bullet,\ f_1',\ f_2',\ f_1',\ f_2'],\\ [h,\ N \rightarrow \alpha \bullet M\beta,\ i,\ j,\ -,\ -]\end{array}}{[h,\ N \rightarrow \alpha M \bullet \beta,\ i,\ k,\ f_1',\ f_2']} \qquad \textbf{(Adj 1')}$$

$$\frac{\begin{array}{l}[j,\ \top \rightarrow R_t \bullet,\ j,\ k,\ f_1,\ f_2],\\ [h,\ M \rightarrow \gamma \bullet,\ f_1,\ f_2,\ -,\ -],\\ t \in Adj(M),\\ [h,\ N \rightarrow \alpha \bullet M\beta,\ i,\ j,\ f_1',\ f_2']\end{array}}{[h,\ N \rightarrow \alpha M \bullet \beta,\ i,\ k,\ f_1',\ f_2']} \qquad \textbf{(Adj 2')}$$

That this formulation is equivalent to the original combination of the three steps **Adj 0**, **Adj 1** and **Adj 2** can be argued in two stages.

First, the h in $[h,\ M \rightarrow \gamma \bullet,\ f_1,\ f_2,\ f_1',\ f_2']$ or $[h,\ M \rightarrow \gamma \bullet,\ f_1,\ f_2,\ -,\ -]$ occurring as second antecedent of **Adj 1'** or **Adj 2'**, respectively, can be replaced by a fresh variable h' without affecting the correctness of the algorithm. In particular, the occurrence of h in the second antecedent of **Adj 1'** is redundant because of the inclusion of the fifth antecedent $[h,\ F_{t'} \rightarrow \bot \bullet,\ f_1',\ f_2',\ f_1',\ f_2']$. Note that conversely this fifth antecedent is redundant with respect to the second antecedent, since existence of an item $[h,\ M \rightarrow \gamma \bullet,\ f_1,\ f_2,\ f_1',\ f_2']$, such that M dominates the foot of a tree t', implies the existence of an item $[h,\ F_{t'} \rightarrow \bot \bullet,\ f_1',\ f_2',\ f_1',\ f_2']$. For further explanation, see Section 4.

Second, the first three antecedents of **Adj 1'** and **Adj 2'** can be split off to obtain **Adj 0**, **Adj 1** and **Adj 2**, justified by principles that are the basis for optimization of database queries (Ullman, 1982).

The rationale for the original formulation of the adjunction steps as opposed to the alternative formulation by **Adj 1'** and **Adj 2'** lie in considerations with respect to the time complexity, as will be discussed in Section 5.

4 Properties

The first claim we make about the algorithm pertains to its correctness as a recognizer:

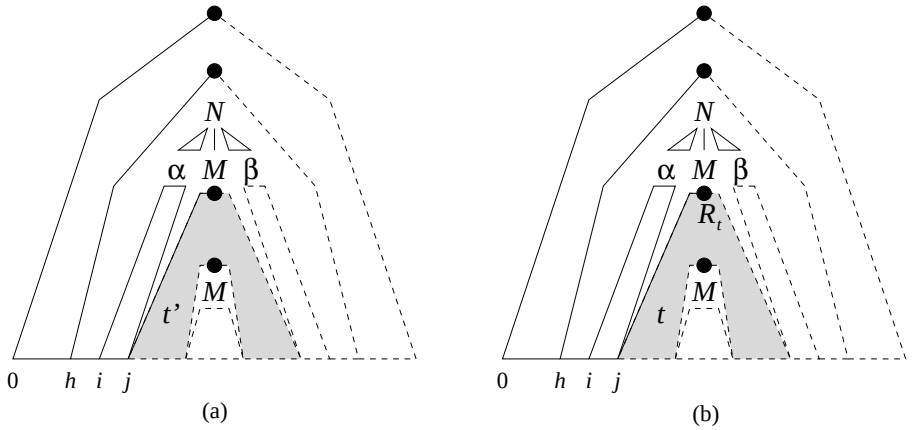


Figure 11
Pred 1 preserves the invariant

Claim
After completion of the algorithm, the item $[0,\ \top \rightarrow R_t \bullet,\ 0,\ n,\ -,\ -]$, for some $t \in I$, is in the table if and only if the input is in the language described by the grammar.

Note that the input is in the language if and only if the input is the yield of a derived tree.

The idea behind the proof of the "if" part is that for any derived tree constructed from the grammar we can indicate a top-down and left-to-right tree traversal that is matched by corresponding items that are computed by steps of the algorithm. The tree traversal and the corresponding items are exemplified by the numbers $1, \ldots, 23$ in Figure 4.

For the "only if" part, we can show for each step separately that the invariant suggested in Figure 3 is preserved. To simplify the proof one can look only at the last five fields of items $[h,\ N \rightarrow \alpha \bullet \beta,\ i,\ j,\ f_1,\ f_2]$, h being irrelevant for the above claim. We do however need h for the proof of the following claim:

Claim
The algorithm satisfies the correct-prefix property, provided the grammar is reduced.

A TAG is reduced if it does not contain any elementary trees that cannot be part of any derived tree. (One reason why an auxiliary tree might not be a part of any derived tree is that at some node it may have obligatory adjunction of itself, leading to "infinite adjunction.")

Again, the proof relies on the invariant sketched in Figure 3. The invariant can be proven correct by verifying that if the items in the antecedents of some step satisfy the invariant, then so does the item in the consequent.

A slight technical problem is caused by the obligatory adjunctions. The pertains to the shaded areas in Figure 3, which represent not merely subtrees of elementary trees, but subtrees of a derived tree, which means that at each node either adjunction or **nil**-adjunction has been performed.

This issue arises when we prove that **Pred 1** preserves the invariant. Figure 11(a) represents the interpretation of the first antecedent of this step, $[h,\ N \rightarrow \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2]$; without loss of generality we only consider the case that $f_1 = f_2 = -$. We may assume that below M some subtree exists, and that at M itself either adjunction with auxiliary tree t' or **nil**-adjunction has been applied; the figure shows the former case.

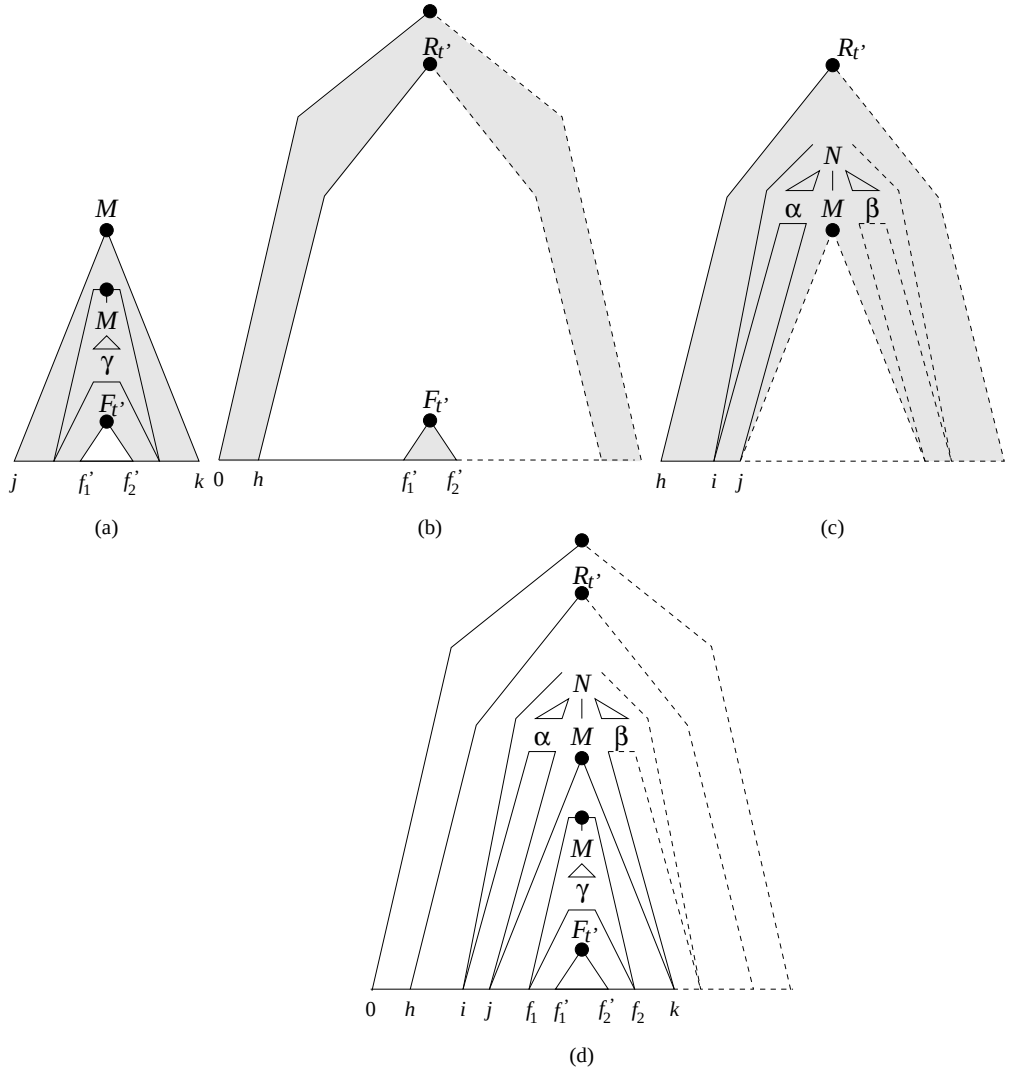


Figure 12
Adj 1 preserves the invariant

In order to justify the item from the consequent, $[j,\ \top \rightarrow \bullet\ R_t,\ j,\ j,\ -,\ -]$, we construct the tree in Figure 11(b), which is the same as that in Figure 11(a), except that t' is replaced by auxiliary tree t, which has been traversed so that at all nodes either adjunction or **nil**-adjunction has been applied, including the nodes introduced recursively through adjunctions. Such a finite traversal must exist since the grammar is reduced.

For the other steps we do not need the assumption that the grammar is reduced in order to prove that the invariant is preserved. For example, for **Adj 1** we may reason as follows. The item $[M \rightarrow \gamma \bullet,\ j,\ k,\ f_1',\ f_2']$, the first antecedent, informs us of the existence of the structure in the shaded area of Figure 12(a). Similarly, the items $[h,\ F_{t'} \rightarrow \bot \bullet,\ f_1',\ f_2',\ f_1',\ f_2']$ and $[h,\ N \rightarrow \alpha \bullet M\beta,\ i,\ j,\ -,\ -]$ provide the shaded areas of Figures 12(b) and 12(c). Note that in the case of the first or third item, we do not use all the information that the item provides. In particular, that the structures are part of a derived tree consistent with the input between positions 0 and k (in the case of (a)) or

j (in the case of (c)) is not needed.

The combined information from these three items ensures the existence of the derived tree depicted in Figure 12(d), which justifies the consequent of **Adj 1**, viz. $[h,\ N \to \alpha M \bullet \beta,\ i,\ k,\ f_1',\ f_2']$.

The other steps can be proven to preserve the invariant in similar ways.

Now the second claim follows: if the input up to position j has been read resulting in an item of the form $[h,\ N \to \alpha a \bullet \beta,\ i,\ j,\ f_1,\ f_2]$, then there is a string y such that $a_1 \cdots a_j y$ is in the language. This y is the concatenation of the yields of the subtrees labelled I, II, and III in Figure 3.

The full proofs of the two claims above are straightforward but tedious. Furthermore, our new algorithm is related to many existing recognition algorithms for TAGs (Vijay-Shankar and Joshi, 1985; Schabes and Joshi, 1988; Lang, 1988; Vijay-Shanker and Weir, 1993; Schabes and Shieber, 1994; Schabes, 1994), some of which were published together with proofs of correctness. Therefore including full proofs for our new algorithm does not seem justified.

5 Complexity

The steps presented in pseudo-formal notation in Section 3 can easily be composed into an actual algorithm (Shieber, Schabes, and Pereira, 1995). This can be done in such a way that the order of the time-complexity is determined by the maximal number of different combinations of antecedents per step. If we restrict ourselves to the order of the time-complexity expressed in the length of the input, this means that the complexity is given by $\mathcal{O}(n^p)$, where p is the largest number of input positions in any step.

However, a better realization of the algorithm exists that allows us to exclude the variables for input positions that occur only once in a step, which we will call **irrelevant** input positions. This realization relies on the fact that an intermediate step

$$\frac{I}{I'}$$

may be applied that reduces an item I with q input positions to another item I' with $q' \leq q$ input positions, omitting those that are irrelevant. That reduced item I' then takes the place of I in the antecedent of the actual step. This has a strong relationship to optimization of database queries (Ullman, 1982).

For example, there are 9 variables in **Comp 1**, of which i, f_1, f_2, f_1', f_2' are all irrelevant, since they occur only once in that step. An alternative formulation of this step is therefore given by the combination of the following three steps:

$$\frac{[h,\ M \to \gamma \bullet,\ k,\ l,\ f_1',\ f_2']}{[h,\ M \to \gamma \bullet,\ k,\ l,\ ?,\ ?]} \qquad \textbf{(Omit 5-6)}$$

$$\frac{[h,\ N \to \alpha \bullet M\beta,\ i,\ j,\ f_1,\ f_2]}{[h,\ N \to \alpha \bullet M\beta,\ ?,\ j,\ ?,\ ?]} \qquad \textbf{(Omit 3-5-6)}$$

$$\frac{\begin{array}{l}[h,\ M \to \gamma \bullet,\ k,\ l,\ ?,\ ?],\\ t \in Adj(M),\\ [j,\ F_t \to \bullet \perp,\ k,\ k,\ -,\ -],\\ [h,\ N \to \alpha \bullet M\beta,\ ?,\ j,\ ?,\ ?]\end{array}}{[j,\ F_t \to \perp \bullet,\ k,\ l,\ k,\ l]} \qquad \textbf{(Comp 1')}$$

The question marks indicate omitted input positions. Items containing question marks are distinguished from items without them, and from items with question marks in different fields.

In **Comp 1'** there are now only 4 input positions left. The contribution of this step to the overall time-complexity is therefore $\mathcal{O}(n^4)$ rather than $\mathcal{O}(n^9)$. The contribution of **Omit 5-6** and **Omit 3-5-6** to the time complexity is $\mathcal{O}(n^5)$.

For the entire algorithm, the maximum number of relevant input positions per step is 6. Thereby, the complexity of left-to-right recognition for TAGs under the constraint of the correct-prefix property is $\mathcal{O}(n^6)$. There are five steps that contain 6 relevant input positions, viz. **Comp 2**, **Comp 3**, **Adj 0**, **Adj 1** and **Adj 2**.

In terms of the size of the grammar G, the complexity is $\mathcal{O}(|G|^2)$, since at most 2 elementary trees are simultaneously considered in a single step. Note that in some steps we address several parts of a single elementary tree, such as the two parts represented by the items $[h,\ F_{t'} \rightarrow \perp \bullet,\ f'_1,\ f'_2,\ f'_1,\ f'_2]$ and $[h,\ N \rightarrow \alpha \bullet M\beta,\ i,\ j,\ -,\ -]$ in **Adj 1**. However, the second of these items uniquely identifies the second field of the first item, and therefore this pair of items amounts to only one factor of $|G|$ in the time complexity.

The complexity of $\mathcal{O}(n^6)$ that we have achieved depends on two ideas: first, the use of **Adj 0**, **Adj 1** and **Adj 2** instead of **Adj 1'** and **Adj 2'**, and second, the exclusion of irrelevant variables above. Both are needed. The exclusion of irrelevant variables alone, in combination with **Adj 1'** and **Adj 2'**, leads to a complexity of $\mathcal{O}(n^8)$. Without excluding irrelevant variables, we obtain a complexity of $\mathcal{O}(n^9)$ due to **Comp 1**, which uses 9 input positions.

The question arises where the exact difference lies with the algorithm from Schabes and Joshi (1988), and whether that algorithm could be improved to obtain the same time-complexity as ours, using techniques similar to those discussed above. This question is difficult to answer precisely because of the significant difference between the types of item that are used in the respective algorithms. However, some general considerations suggest that the algorithm from Schabes and Joshi (1988) is inherently more expensive.

First, the items from the new algorithm have 5 input positions, which implies that storage of the parse table requires a space-complexity of $\mathcal{O}(n^5)$. The items from the older algorithm have effectively 6 input positions, which leads to a space-complexity of $\mathcal{O}(n^6)$.

Second, the "Right Completor" from Schabes and Joshi (1988), which roughly corresponds with our adjunctor steps, has 9 relevant input positions. This step can be straightforwardly broken up into smaller steps that each have fewer relevant input positions, but it seems difficult to reduce the maximal number of positions to 6.

A final remark on Schabes and Joshi (1988) concerns the time-complexity in terms of the size of the grammar that they report, viz. $\mathcal{O}(|G|^2)$. This would be the same upper bound as in the case of the new algorithm. However, the correct complexity seems to be $\mathcal{O}(|G|^3)$, since each item contains references to 2 nodes of the same elementary tree, and the combination in "Right Completor" of two items entails the simultaneous use of 3 distinct nodes from the grammar.

6 Further Research

The algorithm in the present paper operates in a top-down manner, being very similar to Earley's algorithm (Earley, 1970), which is emphasized by the use of the "dotted" items. As shown by Nederhof and Satta (1994), a family of parsing algorithms (viz. top-down, left-corner, PLR, ELR, and LR parsing (Nederhof, 1994)) can be carried over to head-driven parsing. An obvious question is whether such parsing techniques can also be used to produce variants of left-to-right parsing for TAGs. Thus, one may conjecture, for example, the existence of an LR-like parsing algorithm for arbitrary TAGs that operates

in $\mathcal{O}(n^6)$ and that has the correct-prefix property.

Note that LR-like parsing algorithms were proposed by Schabes and Vijay-Shanker (1990) and Nederhof (1998). However, for these algorithms the correct-prefix property is not satisfied.

Development of advanced parsing algorithms for TAGs with the correct-prefix property is not at all straightforward. In the case of context-free grammars, the additional benefit of LR parsing, in comparison to, for example, top-down parsing, lies in the ability to process multiple grammar rules simultaneously. If this is to be carried over to TAGs, then one needs to decide in what way multiple elementary trees can be handled simultaneously. However, this is difficult to reconcile with the mechanism we used to ensure the correct-prefix property, since enforcing the correct-prefix property relies on filtering out hypotheses with respect to "left context," and this requires detailed investigation of that left context, which precludes treating multiple elementary trees simultaneously. An exception may be the case when a TAG contains many, almost identical, elementary trees. It is not clear whether this case occurs often in practice.

Therefore, further research is needed not only to precisely define advanced parsing algorithms for TAGs with the correct-prefix property, but also to determine whether there are any benefits for practical grammars.

Acknowledgments

Most of the presented research was carried out within the framework of the Priority Programme Language and Speech Technology (TST), while the author was employed at the University of Groningen. The TST-Programme is sponsored by NWO (Dutch Organization for Scientific Research).

An error in a previous version of this paper was found and corrected with the help of Giorgio Satta.

References

Earley, Jay. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, February.

Joshi, Aravind K. 1987. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins Publishing Company, Amsterdam, pages 87–114.

Lang, Bernard. 1988. The systematic construction of Earley parsers: Application to the production of $\mathcal{O}(n^6)$ Earley parsers for tree adjoining grammars. Unpublished paper, December.

Nederhof, Mark-Jan. 1994. An optimal tabular parsing algorithm. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 117–124, Las Cruces, New Mexico, USA, June.

Nederhof, Mark-Jan. 1998. An alternative LR algorithm for TAGs. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 2, pages 946–952, Montreal, Quebec, Canada, August.

Nederhof, Mark-Jan and Giorgio Satta. 1994. An extended theory of head-driven parsing. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 210–217, Las Cruces, New Mexico, USA, June.

Rajasekaran, Sanguthevar and Shibu Yooseph. 1995. TAL recognition in $O(M(n^2))$ time. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 166–173, Cambridge, Massachusetts, USA, June.

Satta, Giorgio. 1994. Tree-adjoining grammar parsing and Boolean matrix multiplication. *Computational Linguistics*, 20(2):173–191.

Schabes, Yves. 1994. Left to right parsing of lexicalized tree-adjoining grammars. *Computational Intelligence*, 10(4):506–524.

Schabes, Yves and Aravind K. Joshi. 1988. An Earley-type parsing algorithm for tree adjoining grammars. In *26th Annual Meeting of the Association for Computational Linguistics*, pages 258–269, Buffalo, New York, June.

Schabes, Yves and Stuart M. Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124.

Schabes, Yves and K. Vijay-Shanker. 1990. Deterministic left to right parsing of tree adjoining languages. In *28th Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Pittsburgh, Pennsylvania, USA, June.

Schabes, Yves and Richard C. Waters. 1995. Tree insertion grammar: A cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–513.

Shieber, Stuart M., Yves Schabes, and Fernando C.N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36.

Sippu, Seppo and Eljas Soisalon-Soininen. 1988. *Parsing Theory, Vol. I: Languages and Parsing*, volume 15 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag.

Ullman, Jeffrey D. 1982. *Principles of Database Systems*. Computer Science Press.

Vijay-Shankar, K. and Aravind K. Joshi. 1985. Some computational properties of tree adjoining grammars. In *23rd Annual Meeting of the Association for Computational Linguistics*, pages 82–93, Chicago, Illinois, USA, July.

Vijay-Shanker, K. and David J. Weir. 1993. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.

Vijay-Shanker, K. and David J. Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–546.