PREDICTIVE LEFT-TO-RIGHT PARSING OF A RESTRICTED VARIANT OF TAG(LD/LP) *

Guido Minnen
Sonderforschungsbereich 340
University of Tübingen
Wilhelmstr. 113
72074-Tübingen
Germany
E-mail: minnen@sns.neuphilologie.uni-tuebingen.de

ABSTRACT

By means of a factorization of linear precedence (LP) information in Tree Adjoining Grammar (TAG), the possibility to capture syntactic generalizations can be enhanced, especially with respect to languages which exhibit relative free word-order. In TAG(LD/LP) LP-information is factorized in a way reminiscent of the ID/LP format of CFGs. However, in contrast to its context-free counterpart, in TAG(LD/LP) entanglement of constituents is allowed by admitting not only the permutation of sister nodes, but also of non-sister nodes in a structural description. In this paper a generalization of the predictive left-to-right parser for TAG is presented which is able to handle a restricted variant of TAG(LD/LP), which allows the permutation of sister nodes and non-sister nodes within elementary structures, the basic building blocks of TAG(LD/LP).

Keywords: factorization of linear precedence information, Earley-like parsing of TAG, entanglement of constituents.

*A preliminary version of this paper was presented at the second TAG+ Workshop, June 24-26, 1992, Philadelphia, PA. The presented research was carried out at the Institute for Language Technology an Artificial Intelligence, Tilburg University, The Netherlands and Tübingen University, Germany, as part of the Sonderforschungsbereich 340 of the German Science Foundation Teilprojekt B4 "Constraints on Grammar for Efficient Generation".

1 INTRODUCTION

Tree Adjoining Grammar (TAG; [8, 9]) is a tree-based grammar formalism in which dependencies are factored following from primitive dependencies in elementary trees. Recursion possibilities are factored as a consequence of the composition operation of adjunction. As a result the domain of locality in TAGs is larger than for CFG-based formalisms (such as Lexical-Functional Grammar, or Generalized or Head-driven Phrase Structure Grammar). By means of an additional factorization of linear precedence (LP) information in TAG, the possibility to capture syntactic generalizations can be enhanced, especially with respect to languages which exhibit relative free word-order.

Two extensions of TAG which focus on the factorization of LP-information have been proposed in the literature: Local Dominance/Linear Precedence TAG (TAG(LD/LP); [11, 7]) and Free Order TAG (FO-TAG; [2, 3]). In TAG(LD/LP) LP-information is factorized in a manner reminiscent of the ID/LP format of CFG (ID/LP CFG; [6]). A TAG(LD/LP) consists of elementary structures: domination structures (LD-structures) over which a set of LP-constraints can be defined. Different from its context-free counterpart, TAG(LD/LP) allows entanglement of constituents by admitting not only the permutation of sister nodes, but also the permutation of non-sister nodes both within and between elemenatry structures. The possibilty of entanglement of constituents in TAG(LD/LP) allows an elegant description of discontinuous constituents, as shown in [11, 7]. FO-TAG constitutes a further extension of the TAG(LD/LP) formalism in that it increases the possibilities for elementary structures to inherit LP-constraints from other elementary structures.

An interesting question is whether the increase of strong generative capacity with respect to TAG embodied in TAG(LD/LP) and FO-TAG[1] can be dealt with by generalizations of existing parsing algorithms for TAGs without a significant increase of computational costs. A partial (positive) answer to this question is given in this paper by showing that the predictive left-to-right parser for TAGs (91-parser), as proposed by Schabes in [15], can be generalized to handle a restricted variant of TAG(LD/LP) without influencing the time complexity due to the length of the input string. The restricted variant of TAG(LD/LP) in addition to the permutation of sister nodes allows entanglement of constituents, but only within elementary structures. As a result of this restriction on entanglement of constituents the presented algorithm does not need to deal with inheritance of LP-constraints between elementary structures. The specific problems posed by the possibilities for inheritance of LP-constraints as proposed in the context of FO-TAG are not addressed in this paper. The generalization of the 91-parser is based on a refinement of the notion of entanglement of constituents.

[1] It is not known yet whether TAG(LD/LP) (or FO-TAG) exceed the generative capacity of TAG ([7]).

2 TAG(LD/LP)

2.1 TAG

A TAG[2] is a 5-tuple $G = (V_N, V_T, R, I, A)$ where V_N is a finite set of nonterminal symbols, V_T is a finite set of terminal symbols possibly including the empty string symbol (ϵ), R is a distinguished nonterminal symbol, I is a finite set of trees called *initial trees* and A is a finite set of trees called *auxilairy trees*. The trees $I \cup A$ are called *elementary trees*.

Initial trees (see 1 in figure 1) are characterized as follows: internal nodes are labeled by nonterminal symbols; leaf nodes are labeled by terminal symbols.

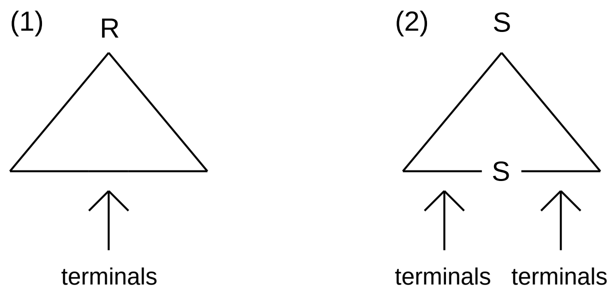

Figure 1 . Schematic initial and auxiliary trees.

Auxiliary trees (see 2 in figure 1) are characterized as follows: internal nodes are labeled by nonterminal symbols; leaf nodes are labeled by terminal symbols except for exactly one node (called the *foot node* of the auxiliary tree) labeled by a nonterminal symbol; the foot node and the root node of an auxiliary tree carry identical labels.

In TAG, trees are combined by means of the composition operation *adjunction*. By means of adjunction, from an auxiliary tree and another tree, which is either initial, auxiliary or the result of the previous combination of two elementary trees, a new tree is built, which is called a *derived tree*. The adjunction operation is illustrated in figure 2.

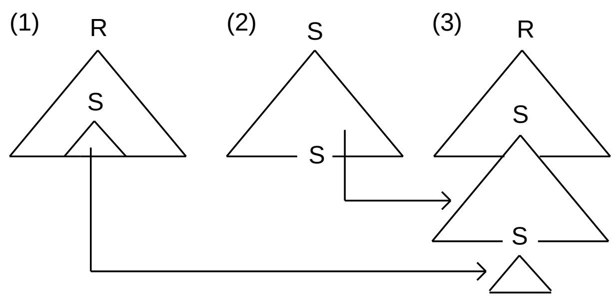

Figure 2 . The composition operation adjunction.

Initial tree 1 is combined with auxiliary tree 2 into the derived tree 3 by means of adjoining 2 at node n in 1, which is labeled by the nonterminal symbol S, as follows:

[2]For an introduction to TAG see [10]

- The subtree of 1 dominated by node n, call it t, is excised, leaving a copy of n behind.
- The auxiliary tree 2 is attached at node n and its root node is identified with n.
- Subtree t is attached to the foot node of 2 and the root node n of t is identified with the foot node of 2.

For linguistic descriptions it is convenient and sometimes necessary to be specific as to which elementary trees can be adjoined at a given node. This is exactly what is achieved by *local constraints on adjunction.* In TAG one can, for each node n in an elementary tree, specify one of the following three constraints on adjunction:

1. selective adjunction: only a specified subset of the set of all elementary trees is adjoinable at node n;
2. null adjunction: no elementary trees are adjoinable at node n;
3. obligatory adjunction: at least one elementary tree (of all the elementary trees adjoinable at n) must be adjoined at n.

A hypothesis that underlies the linguistic intuitions in TAGs is that all dependencies are captured within elementary trees. This is based on the assumption that elementary trees are the appropriate domain (of locality) upon which to define dependencies rather than rewrite rules in CFG-based formalisms.

2.2 TAG(LD/LP)

The LD/LP format of TAG is basically similar to ID/LP CFG. However, there are three important differences:

1. The domain of locality is the elementary structures which are the TAG(LD/LP) equivalent of elementary trees.
2. LP-constraints are defined over LD-structures: in contrast to the LP-rules in ID/LP CFGs which are globally defined, the LP-constraints in TAG(LD/LP) are defined locally. Furthermore, LP-constraints can be defined over non-sister nodes as long as they belong to the same elementary structure.
3. By relaxing the LP-constraints, instances of elementary structures can exhibit entanglement of constituents[3].

As an example of entanglement within elementary structures, consider the representation of the initial structure in figure 3, in which a node is labeled by a (non)terminal symbol and an

[3] As a result of the possibility of entanglement of constituents in TAG(LD/LP) structural descriptions do not necessarily preserve the tree property. Therefore, the basic building blocks of TAG(LD/LP) are referred to as elementary structures.

address referred to by the set of LP-constraints[4]. Due to the fact that x (1) is not stipulated to precede S (2) or b (2.1) constituents *within* an instance of this elementary structure are allowed to entangle, i.e. the structural description corresponding to the yield bxc is allowed.

In case of entanglement of two constituents x and y the following terminology is used: if the leftmost leaf node of the structure corresponding to x precedes the leftmost leaf node corresponding to y, then x and y are referred to as the *intruded constituent* and the *intruding constituent*, respectively[5]. Applying this to the previous example, S (2) is called the intruded constituent and x (1) is the intruding constituent.

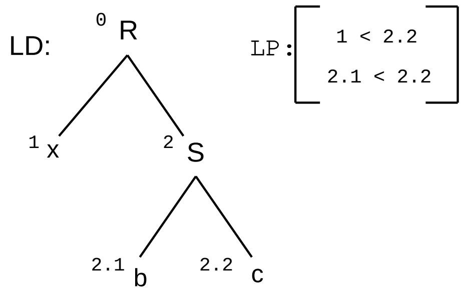

Figure 3 . Entanglement within elementary structures.

TAG(LD/LP) also allows entanglement *between* instances of elementary structures as illustrated in figure 4. In elementary structure α (alpha) the set of LP-constraints enforces the standard tree-ordering upon the corresponding LD-structure. However, if elementary structure α is combined with β (beta) the resulting set of LP-constraints allows the entanglement of the constituents x (1) and S (2) corresponding to the yield $axbc$. Here again, S (2) is called the intruded constituent and x (1) is the intruding constituent.

The entanglement of constituents is not only constrained by LP-constraints, but also by the *Consistency Condition* ([11]):

Consistency Condition (CC): Given an elementary structure or a derived structure with an LP-constraint [X < Y]: if X dominates Z1 and Y dominates Z2 then Z1 < Z2.

The CC accomplishes a default inheritance of LP-constraints. Intuitively understood, the condition stipulates that if two nodes are constrained to appear in a particular order, the LD-structures rooted at these nodes are not allowed to entangle (see also section 4.1).

[4]The addresses of nodes in an LD-structure are Gorn-positions as defined by the following inductive definition: 0 is the address of the root node, k ($k \in N$) is the address of the k^{th} daughter of the root node, $x.y$ (x is an address, $y \in N$) is the address of the y^{th} daughter of the node at address x.

[5]A reminiscent way to refer to entanglement of constituents is employed in [4].

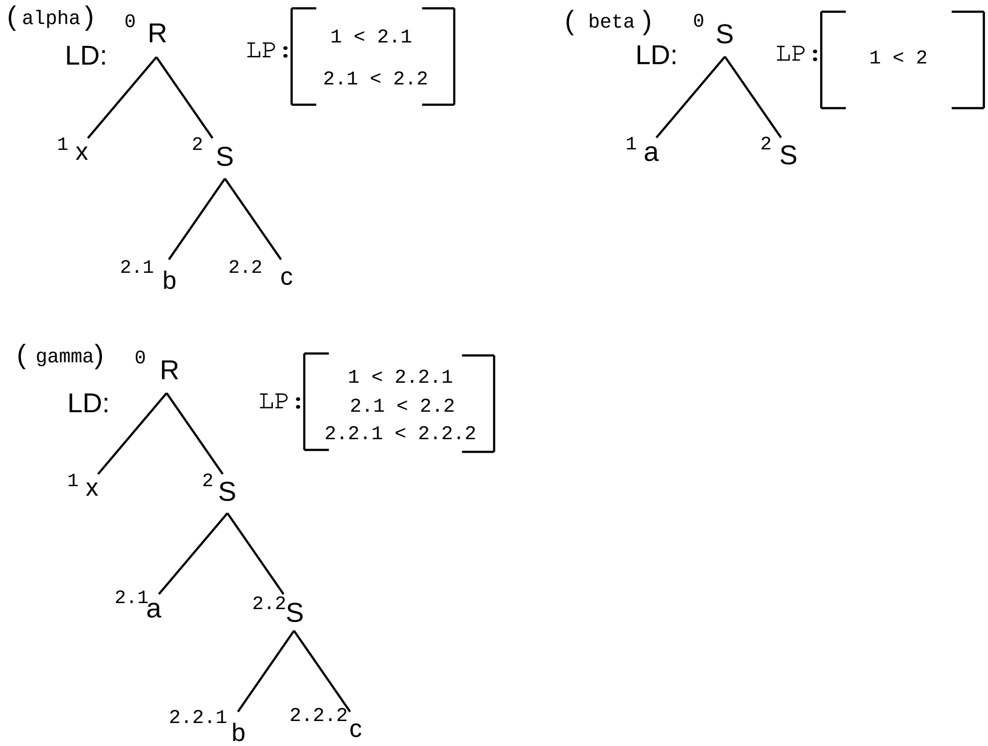

Figure 4 . Entanglement between elementary structures.

3 TAG(LD/TLP) PARSING

3.1 Earley-like Parsing of TAG

There are two parsing algorithms for TAGs proposed in the literature which closely relate to Earley's algorithm for parsing CFGs ([5]): the Earley-type parser for TAGs (88-parser; [14]) and the predictive left-to-right parser for TAGs ([15]).

The 91-parser is taken as a starting point for the development of a parsing algorithm for the ID/TLP TAG formalism, which is a restricted variant of the TAG(LD/LP) formalism where the tree-property (T) of the structures assigned to the input string is preserved by excluding any kind of entanglement, i.e. only sister nodes are allowed to permute. In TAG(LD/TLP), the previously introduced CC is satisfied automatically[6].

The generalization of the 91-parser to cope with the features of the TAG(LD/TLP) formalism are based on proposals by Shieber ([17]) for direct parsing of ID/LP CFGs. Shieber generalized Earley's algorithm, such that it is not necessary to expand an ID/LP CFG to a normal CFG, before it can be used to parse a sentence. The main idea underlying this extension is that an ID-rule can get linearized in accordance with the globally stated LP-rules gradually during the parsing process.

[6]The 91-parser is taken as a starting point, because it has a better worst case behaviour than the 88-parser due to the fact that the earlier parser does not maintain the valid prefix property ([15]). Due to the fact that it does not the valid prefix property the 91-parser is not truly an Earley-type parser.

An alternative two-step parser for TAG(LD/TLP) (2step-parser) was recently proposed in [13]. The 2step-parser applies the Shieber/Earley algorithm directly to compute a set of relevant context-free derivations taking advantage of the close relation between TAG and CFG and subsequently, rules out all context-free derivations which do not correspond to a valid TAG-solution. The time complexity due to the length of the input string of the parser presented in this section is claimed to be identical to the time complexity of the 2step-parser (see section 7). It is unclear whether it is possible to further extend the 2step-parser in order to deal with the permutation of non-sister nodes within an elementary structure, because the representation of elementary structures is scattered over various states in the chart.

3.2 Gradual Linearization and State Representation

Like the 88-parser, the recognizer for TAG(LD/TLP), stores intermediate results of the recognition process in a chart which is divided in state sets. The state sets correspond to the recognition of a particular part of the input string. In the states a division is made between a linearized and an unlinearized structural representation, like in Shieber's algorithm. The linearized structural representation is constructed during the recognition process by means of adding nodes to it from the unlinearized structural representation, i.e. the LD-structure from the grammar, in accordance with the applicable set of LP-constraints. When all nodes from the elementary LD-structure are attached to the linearized structural representation, this linearization is a complete LP-acceptable instance of the elementary structure, i.e. a complete structural description that does not violate any of the applicable LP-constraints. For example, the gradual linearization of the elementary structure in figure 3 corresponding to the yield xbc proceeds as follows:

$[[R(0), []]],$
$[[R(0), [[x(1), []]]]],$
$[[R(0), [[x(1), []], [[S(2), []]]]],$
$[[R(0), [[x(1), []][[S(2), [[b(21), []]]]]],$
$[[R(0), [[x(1), []][[S(2), [[b(21), []], [c(22), []]]]]],$

Figure 5 . Example of gradual linearization.

A linearized structural representation is represented by means of a list consisting of two elements: the label of the root node of the LD-structure (R(0), in this case) and a list specifying possible daughter nodes, which are represented by a list consisting of a node label and a list specifying possible daughter nodes. In case a node label is associated with an empty list, this indicates that the label refers to a node of which no daughter nodes have been recognized or suggested for recognition yet.

The linearized structural representation is a *dotted tree*, which is an extension of Earley's notion of a dotted rule. A dotted tree is a tree in which exactly one node is associated with a dot. For internal nodes, four dot-positions are identified: left-above, left-below, right-below and right-above. For leaf nodes, two dot-positions are identified: left-above and right-above.

The dot-positions are used to mimic a *tree traversal* which is necessary to scan the fringe of the tree from left-to-right while trying to recognize possible adjunctions. A dotted tree is constructed during the recognition process and subsequently traversed as illustrated in figure 6.

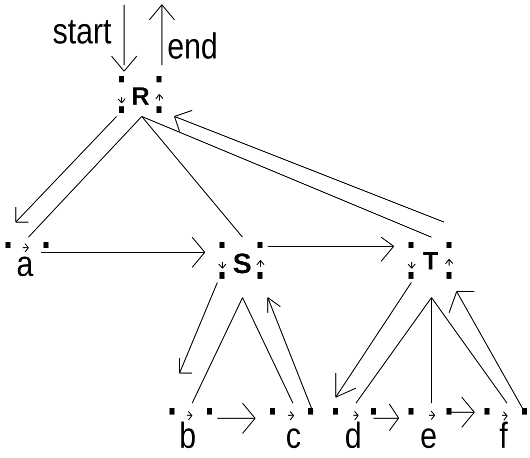

Figure 6 . Example of a tree traversal.

Given a dotted tree with the dot left-above of the root node, the tree traversal of a dotted tree is defined as follows:

- if the dot is at position left-above of an internal node, it is moved down to position left-below of the same internal node;
- if the dot is at position left-below of a internal node, it is moved down to the position left-above of the leftmost daughter node of this internal node;
- if the dot is at position left-above of a leaf node, it is moved to the right to the position right-above the leaf node;
- if the dot is at position right-below of an internal node, it is moved up to the position right-above of the same internal node;
- if the dot is at position right-above of a node, there are two cases:
  - if the node is not the rightmost daughter of its mother node, it is moved up to the position left-above the left sister node of this node;
  - if the node is the rightmost daughter of its mother node, the dot is moved up to the position right-below of its mother.

If a dotted tree carries the dot right-above its root node it is completely recognized.

The concept of a dotted tree is also used in the 88-parser and the 91-parser, there, however, it is used to mimic the traversal of a complete elementary tree. In case of TAG(LD/TLP) recognition the dotted tree in a state is in many cases an incomplete instance of an elementary structure.

It is now possible to give an exact definition of a state. A state is defined as a 9-tuple:

$$[LD_\alpha, LP_\alpha, dot, pos, i, j, k, sat?, a],$$

where

- LD_α is the LD-structure of elementary structure α.
- LP_α is the set of LP-constraints of elementary structure α.
- *dot* is an address in the dotted tree a (see below) indicating which part of α has been linearized so far.
- *pos* is the position of the dot at address *dot*: *pos* $\in$ {*la,lb,ra,rb*}, where la, lb, ra and rb refer to the dot-positions left-above, left-below, right-above and right-below, respectively.
- i, j and k are indices of positions in the input string ranging over {*0,...,n*} $\cup${-}, n being the length of the input string and - indicating that an index is not bound. Index i indicates the position in the input string where the recognition of a started. In case α is an auxiliary structure the indices j and k indicate which part of the input string is recognized as the foot node of a.
- *sat?* is a boolean: *sat?* $\in$ {*true,nil*}. *sat?* is used to signal successful adjunctions.
- a is the dotted tree which constitutes a (partial) representation of an LP_α-acceptable instance of LD_α.

3.3 Main Recognition Algorithm

The recognition process can be informally described as follows: initialization adds a state

$$[LD_\delta, LP_\delta, 0, la, 0, -, -, nil, d]$$

for all initial structures δ in the grammar to state set S_0, where the representation of the partial linearization d, i.e. a dotted tree, in the states resulting from initialization looks as follows:

$$d = [[R(0), []]],$$

where R is the distinguished nonterminal symbol: the label of the root node of δ is added to d. No daughter nodes of the node labeled R (0) are recognized or predicted for recognition yet, therefore the list specifying the daughter nodes of R is empty. Depending on the states resulting from initialization new states are added to the state sets in the chart by six operations until no more states can be added. The six operations are: *Scanner*, *Predictor*, *Completor*, *Adjunctor*, *MoveDotUp* and *MoveDotDown*. If state set S_n (n being the length of the input string) contains a state of the form:

$$[LD_\delta, LP_\delta, 0, ra, 0, -, -, nil, d],$$

where d is a complete LP_δ-acceptable linearization of an initial structure, i.e. a dotted tree with the dot to the right and above the root node in d which spans the input from position 0 to n the input string is completely recognized.

Described more formally, this looks as follows[7]:

Let $G = (V_T, V_N, R, I, A)$ be a TAG(LD/TLP).
Let $i_1...i_n$ be the input string.

program recognizer
begin
$S_0 \;=\; \{[LD_\delta, LP_\delta, 0, la, 0, -, -, nil, d] \;\mid\; \delta \;\in\; I\}$
For x := 0 to n do
    begin
    Process the states of S_x, preforming one of the following
    six operations on each state until no more states can be added:
    1. Scanner
    2. Predictor
    3. Completor
    4. Adjunctor
    5. MoveDotDown
    6. MoveDotUp
    If S_{x+1} is empty and x < n, return rejection.
    end.
If in S_n a state of the form $[LD_\delta, LP_\delta, 0, ra, 0, -, -, nil, d]$ with $\delta \;\in\; I$ then return acceptance.
end.

3.4 Example of TAG(LD/TLP) Recognition

The main recognition algorithm is illustrated by means of an example. The sample TAG(LD/TLP) used for the example is given in figure 7. The sample grammar consists of two elementary structures α (alpha) and β (beta). Both the root and foot node of β carry a *null adjunction constraint* (na), which indicates that no adjunction is allowed at these nodes.

The state sets constituting the chart resulting from the recognition of the input string *aebcd* is given in figure 8. The states constituting state set S_0 are the predicted recognitions with no elements of the input string read yet. In state set S_x the intermediate results up to and including reading the x^{th} element of the input string are stored. The input string is recognized since the state

$$[LD_\alpha, LP_\alpha, 0, ra, 0, -, -, nil, [[s(0), [[e(1), []]]]]\;]$$

is contained in state set S_5.

For each state in a state set it is indicated which operation added it to a state set (see §3.5). Between parentheses the applicable operation is given and between square brackets it is indicated on the basis of which state(s) a particular operation was performed.

[7]Note that A and I in the case of TAG(LD/TLP) refer to the sets of initial and auxiliary *structures* instead of *trees*.

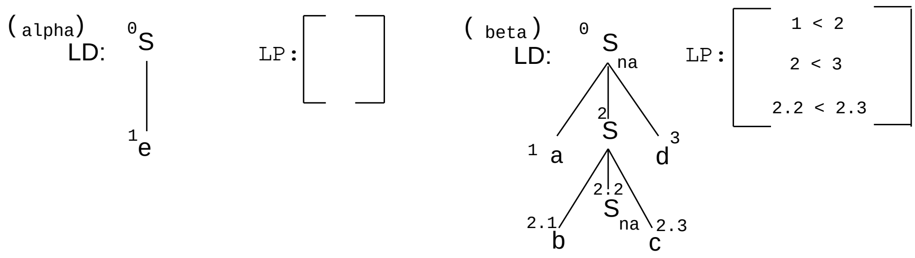

Figure 7 . Sample TAG(LD/TLP).

State set S_0:

1. (Initialization:) $[LD_\alpha, LP_\alpha, 0, la, 0, -, -, nil, [[s(0), []]]\ ]$
2. (Predictor[1]:) $[LD_\alpha, LP_\alpha, 0, lb, 0, -, -, nil, [[s(0), []]]\ ]$
3. (MoveDotDown[2]:) $[LD_\alpha, LP_\alpha, 1, la, 0, -, -, nil, [[s(0), [[e(1), []]]]]\ ]$
4. (Predictor[1]:) $[LD_\beta, LP_\beta, 0, la, 0, -, -, nil, [[s(0), []]]\ ]$
5. (Predictor[4]:) $[LD_\beta, LP_\beta, 0, lb, 0, -, -, nil, [[s(0), []]]\ ]$
6. (MoveDotDown[5]:) $[LD_\beta, LP_\beta, 1, la, 0, -, -, nil, [[s(0), [[a(1), []]]]]\ ]$

State set S_1:

7. (Scanner[6]:) $[LD_\beta, LP_\beta, 1, ra, 0, -, -, nil, [[s(0), [[a(1), []]]]]\ ]$
8. (MoveDotUp[7]:) $[LD_\beta, LP_\beta, 2, la, 0, -, -, nil, [[s(0), [[a(1), []], [s(2), []]]]]\ ]$
9. (Predictor[8]:) $[LD_\beta, LP_\beta, 2, lb, 1, -, -, nil, [[s(0), [[a(1), []], [s(2), []]]]]\ ]$
10. (MoveDotDown[9]:) $[LD_\beta, LP_\beta, 21, la, 1, -, -, nil, [[s(0), [[a(1), []], [s(2), [[b(21), []]]]]]]\ ]$
11. (MoveDotDown[9]:) $[LD_\beta, LP_\beta, 22, la, 1, -, -, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []]]]]]]\ ]$
12. (Predictor[11]:) $[LD_\beta, LP_\beta, 22, lb, 1, -, -, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []]]]]]]\ ]$
13. (Predictor[12+2]:) $[LD_\alpha, LP_\alpha, 0, lb, 1, -, -, nil, [[s(0), []]]\ ]$
14. (MoveDotDown[13]:) $[LD_\alpha, LP_\alpha, 1, la, 1, -, -, nil, [[s(0), [[e(1), []]]]]\ ]$
15. (Predictor[14]:) $[LD_\beta, LP_\beta, 0, la, 1, -, -, nil, [[s(0), []]]\ ]$
16. (Predictor[15]:) $[LD_\beta, LP_\beta, 0, lb, 1, -, -, nil, [[s(0), []]]\ ]$
17. (MoveDotDown[16]:) $[LD_\beta, LP_\beta, 1, la, 1, -, -, nil, [[s(0), [[a(1), []]]]]\ ]$

State set S_2:

18. (Scanner[14]:) $[LD_\alpha, LP_\alpha, 1, ra, 1, -, -, nil, [[s(0), [[e(1), []]]]]\ ]$
19. (MoveDotUp[18]:) $[LD_\alpha, LP_\alpha, 0, rb, 1, -, -, nil, [[s(0), [[e(1), []]]]]\ ]$
20. (Completor[19+12]:) $[LD_\beta, LP_\beta, 22, rb, 1, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []]]]]]]\ ]$
21. (Completor[20+11]:) $[LD_\beta, LP_\beta, 22, ra, 1, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []]]]]]]\ ]$
22. (MoveDotUp[21]:) $[LD_\beta, LP_\beta, 21, la, 1, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []]]]]]]\ ]$
23. (MoveDotUp[21]:) $[LD_\beta, LP_\beta, 23, la, 1, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [c(23), []]]]]]]\ ]$

State set S_3:

24. (Scanner[23]:) $[LD_\beta, LP_\beta, 21, ra, 1, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []]]]]]]\ ]$
25. (MoveDotUp[24]:) $[LD_\beta, LP_\beta, 23, la, 1, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []], [c(23), []]]]]]]\ ]$

State set S_4:

26. (Scanner[25]:) $[LD_\beta, LP_\beta, 23, ra, 1, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []], [c(23), []]]]]]]\ ]$
27. (MoveDotUp[26]:) $[LD_\beta, LP_\beta, 2, rb, 1, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []], [c(23), []]]]]]]\ ]$
28. (Completor[12+27]:) $[LD_\beta, LP_\beta, 22, rb, 1, 1, 4, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []]]]]]]\ ]$
29. (Completor[8+28]:) $[LD_\beta, LP_\beta, 22, ra, 1, 1, 4, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []]]]]]]\ ]$

30. (MoveDotUp[29]:) $[LD_{\beta}, LP_{\beta}, 21, la, 1, 1, 4, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []]]]]]]\ ]$
31. (MoveDotUp[29]:) $[LD_{\beta}, LP_{\beta}, 23, la, 1, 1, 4, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [c(23), []]]]]]]\ ]$
32. (Completor[8+27]:) $[LD_{\beta}, LP_{\beta}, 2, ra, 0, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []], [c(23), []]]]]]\ ]$
33. (MoveDotUp[32]:) $[LD_{\beta}, LP_{\beta}, 3, la, 0, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []], [c(23), []]]]], [d(3), []]]]]\ ]$

State set S_5:

34. (Scanner[33]:) $[LD_{\beta}, LP_{\beta}, 3, ra, 0, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []], [c(23), []]]]], [d(3), []]]]]\ ]$
35. (MoveDotUp[34]:) $[LD_{\beta}, LP_{\beta}, 0, rb, 0, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []], [c(23), []]]]], [d(3), []]]]]\ ]$
36. (Completor[35+4]:) $[LD_{\beta}, LP_{\beta}, 0, ra, 0, 1, 2, nil, [[s(0), [[a(1), []], [s(2), [[s(22), []], [b(21), []], [c(23), []]]]], [d(3), []]]]]\ ]$
37. (Adjunctor[36+19]:) $[LD_{\alpha}, LP_{\alpha}, 0, rb, 0, -, -, true, [[s(0), [[e(1), []]]]]\ ]$
38. (Completor[37+1]:) $[LD_{\alpha}, LP_{\alpha}, 0, ra, 0, -, -, nil, [[s(0), [[e(1), []]]]]\ ]$

Figure 8. State sets constituting the chart for the TAG(LD/TLP) example. The input string is *aebcd*.

3.5 Move-dot Operations

Refraining from an extensive description of the various operations of the recognizer, the focus of attention in this section is on the move-dot operations *MoveDotDown* and *MoveDotUp* which are not used explicitly in the 91-parser and are further modified to deal with the permutation of non-sister nodes in the next section.

The move-dot-operations are adaptions of the two operations of the same name used in the 88-parser to move the dot passed dot-positions which do not correspond to a particular operation of the parser. In the 91-parser, the movement of the dot from such a redundant dot-position to a relevant dot-postion is taken care of by defining dot-position-equivalences. With respect to the generalization of the 91-parser to deal with factorized LP-information the move-dot-operations are indispensible as they accomodate all necessary LP-tests. Furthermore, they accomplish the gradual linearization of an LD-structure by adding all its nodes to a corresponding dotted tree.

MoveDotDown is applicable to a state in which the dot is situated left-below an internal node of which no daughter nodes have yet been added to the dotted tree. A new state is added to the current state set for each daughter node of the internal node that can be the leftmost daughter node. In these new states the daughter node has been attached to the dotted tree and the dot is situated left-above of this new node.

A structure, i.e. an LD-structure as well as a dotted tree, is considered to be a function from addresses to the symbols of the grammar: if *address* is a valid address in a structure *S*, then *S(address)* is the label of the node at address *address* in the structure *S*.

MoveDotDown applies to a state of the form
$[LD_{\delta}, LP_{\delta}, dot, lb, i, j, k, sat?, d]$ in state set S_x.
It adds for every $LD_{\delta}(l)$ that can be the leftmost daughter of *d(dot)* according to
LP_{δ} and has not been added to d already a state of the form
$[LD_{\delta}, LP_{\delta}, l, la, i, j, k, sat?, d']$ to S_x,
where *d'* is the tree resulting from adding $LD_{\delta}(d)$ as the leftmost daughter of

$d(dot)$ `to` d.

MoveDotUp applies in two situations. First, in case the dot is right-above an internal node of which sister nodes remain in the LD-structure to be attached to the dotted tree. Then a new state is created and added to the current state set for each of the possible sister nodes in the LD-structure not attached to the dotted tree yet that can be the right sister of the dotted node. In these new states a new right sister nodes are added to the dotted tree and the dot is moved to the position left-above of this newly added node. Second, in case the dot is right-above a node of which all sister nodes are attached to the dotted tree already, a new state is created and added to the state set, where the dot is moved to the position right-below the mother of the dotted node.

`MoveDotUp applies to a state of the form`
$[LD_\delta, LP_\delta, dot, ra, i, j, k, sat?, d]$ `in state set` S_x
`such that` $d(dot)$ `has a mother node.`

- `Case 1: If` $d(dot)$ `is not the rightmost daughter of its mother node` $d(m)$`,`
  `then it adds for every` $LD_\delta(r)$ `that can be the right sister of` $d(dot)$`,`
  `according to` LP_δ`, and has not been added to` d `already a state`
  `of the form` $[LD_\delta, LP_\delta, r, la, i, j, k, sat?, d']$ `to` S_x`,`
  `where` d' `is the tree resulting from adding` $LD_\delta(r)$ `as the right sister of`
  $d(dot)$ `to` d.
- `Case 2: If` $d(dot)$ `is the rightmost daughter of its mother node` $d(m)$`, then it`
  `adds a state of the form`
  $[LD_\delta, LP_\delta, m, rb, i, j, k, sat?, d]$ `to` S_x.

The described recognizer can be extended to produce the parse trees corresponding to the recognition of an input string in much the same way as in case of the 91-parser by including pointers indicating on the basis of which states a new state was added to the chart.

4 ENTANGLEMENT WITHIN ELEMENTARY STRUCTURES

4.1 Refinement of the Notion of Entanglement

The default inheritance accomplished by the CC, as discussed in §2.2, can be viewed as a kind of meta-LP-constraint: if two nodes are explicitly constrained to appear in a particular order, i.e. one of the LP-constraints in the set of LP-constraints of an elementary structure refers to both node labels, the substructures rooted at these nodes are not allowed to entangle. As a result of obeying the CC in case of entanglement of constituents in a structural description no particular linear precedence is enforced between these constituents. In figure 9 this is illustrated by means of an example. As a result of the absence of an LP-constraint stipulating an order between the nodes S and x the structures rooted at these nodes are allowed to entangle. However, the absence of such an LP-constraint is problematic, because S can be viewed as preceding x (1), but x can also be viewed as preceding S (2). The absence of a precedence relation between intruded and intruding constituents leads to a kind of spurious ambiguity for every entanglement of constituents, because it can be analyzed as if the intruding constituent precedes the intruded constituent and vice versa.

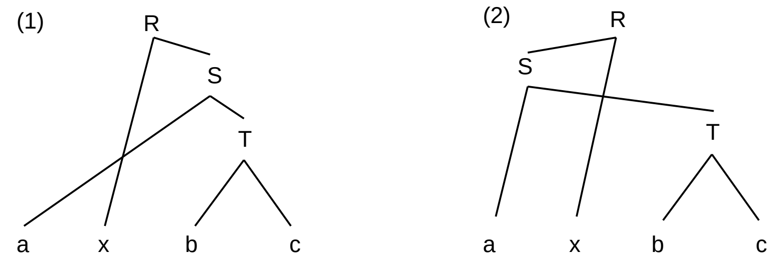

Figure 9. Entanglement and linear precedence.

In order to solve this problem, structural descriptions like 1 are excluded in favour of structural descriptions like 2. In case structural descriptions like 1 are permitted by the formalism, it is possible that before the structure rooted at node S is accessed one of its daughter nodes is encountered in the input string and therefore needs to be scanned. This leads to a complicated administration, because it is necessary to anticipate on the tree traversal. By excluding structural descriptions like 1 in favour of 2 these problems are avoided

In order to establish the exclusion of structural descriptions like 1 the following refinement of the notion of entanglement is adopted[8]:

> Refined Notion of Entanglement: If a constituent x is intruded by a constituent y, the intruding constituent y is always rooted at (a daughter of) a right sister node of one of the nodes dominating x.

Adopting this refined notion of entanglement amounts to stipulating a precedence between constituents involved in an entanglement which does not interfere with the independently motivated CC: in case of entanglement within elementary structures the intruded constituent always precedes the intruding constituent.

4.2 MoveDotUp Revisited

Given the refinement of the notion of entanglement, the algorithm presented in section 3 can be further generalized to cope with entanglement within elementary structures, entanglement which involves only intruding constituents from the domain of locality. The only

[8] A similar restriction on entanglement of constituents is proposed in [4].

operation that is in need of additional modification is *MoveDotUp*. More specifically, case 2 of *MoveDotUp* needs to be changed such that it adds not only right sister nodes of the dotted node as a right sister node to their common mother node, but also nodes corresponding to possibly intruding constituents from within the same elementary LD-structure. This is, of course, constrained by the applicable set of LP-constraints and the CC , such that the linearized instance of the elementary structure is LP-acceptable[9]. The revised version of *MoveDotUp* looks as follows:

`MoveDotUp' applies to a state of the form`
$[LD_\delta, LP_\delta, dot, ra, i, j, k, sat?, d]$ `in state set` S_x
`such that` *d(dot)* `has a mother node.`

- `Case 1: If` *d(dot)* `is not the rightmost daughter of its mother node` *d(m)*`,`
  `then it adds for every` $LP_\delta(r)$ `that can be the right sister of` *d(dot)*`,`
  `or that corresponds to an intruding constituent, according to` LP_δ`, and has`
  `not been added to` d `already, a state of the form`
  $[LD_\delta, LP_\delta, r, la, i, j, k, sat?, d']$ `to` S_x`,`
  `where` *d'* `is the tree resulting from adding` $LD_\delta(r)$ `as the right sister of`
  d(dot) `to` d.

- `Case 2: If` *d(dot)* `is the rightmost daughter of its mother node` *d(m)*`, then it`
  `adds a state of the form`
  $[LD_\delta, LP_\delta, m, rb, i, j, k, sat?, d]$ `to` S_x`,`

It is important to note that though the structural descriptions that are eventually extracted from the state sets in the chart are no longer tree structures, the representation of the linearized and unlinearized structural representation do preserve the tree-property, because intruding constituents are attached as a subconstituent to the intruded constituent and can be represented in a state without any additional changes.

In case of entanglement within elementary structures the extraction of a structural description is more complicated. However, due to the fact that the nodes of the dotted trees in a state representation are labeled by a (non)terminal symbol and the original address in the LD-structure of which the dotted tree is a linearized instance structural descriptions which exhibit entanglement of constituents within an elementary structure can easily be retrieved.

4.3 Example of Recognition of Entanglement within Elementary Structures

By means of example, the relevant state sets constructed upon recognition of the input string *abxc* given the sample grammar in figure 10 are given in figure 11.

Upon the recogntition of the first two elements of the input string state 17 in state set S_2 is put on the chart as a result of the previous extension of the operation *MoveDotUp*: the intruding constituent x is attached as a subconstituent to the intruded constituent S (2). The intruding constituent is subsequently recognized (state 18) and the dot is moved to the

[9]Notice that due to the fact that only entanglement within elementary structures is allowed, the CC influences entanglement only locally, i.e. within the domain of locality.

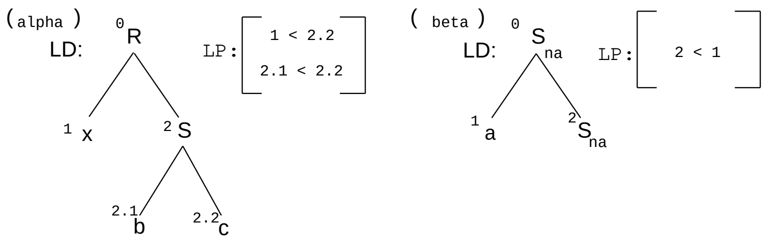

Figure 10. Sample TAG(LD/LP).

part of constituent S (2) that remains to be recognized.

State set S_2:
15.(Scanner[14]:) $[LD_\alpha, LP_\alpha, 21, ra, 1, -, -, nil, [[r(0), [[s(2), [[b(21), []]]]]]]\ ]$
16.(MoveDotUp'[15]:) $[LD_\alpha, LP_\alpha, 22, la, 1, -, -, nil, [[r(0), [[s(2), [[b(21), []], [c(22), []]]]]]]\ ]$
17.(MoveDotUp'[15]:) $[LD_\alpha, LP_\alpha, 1, la, 1, -, -, nil, [[r(0), [[s(2), [[b(21), []], [x(1), []]]]]]]\ ]$
State set S_3:
18.(Scanner[17]:) $[LD_\alpha, LP_\alpha, 1, ra, 1, -, -, nil, [[r(0), [[s(2), [[b(21), []], [x(1), []]]]]]]\ ]$
19.(MoveDotUp'[18]:) $[LD_\alpha, LP_\alpha, 22, la, 1, -, -, nil, [[r(0), [[s(2), [[b(21), []], [x(1), []], [c(22), []]]]]]]\ ]$
State set S_4:
20.(Scanner[19]:) $[LD_\alpha, LP_\alpha, 22, ra, 1, -, -, nil, [[r(0), [[s(2), [[b(21), []], [x(1), []], [c(22), []]]]]]]\ ]$
21(MoveDotUp'[20]:) $[LD_\alpha, LP_\alpha, 2, rb, 1, -, -, nil, [[r(0), [[s(2), [[b(21), []], [x(1), []], [c(22), []]]]]]]\ ]$
22.(Completor[21+12]:) $[LD_\beta, LP_\beta, 2, rb, 1, 1, 4, nil, [[s(0), [[a(1), []], [s(2), []]]]]\ ]$
23.(Completor[22+11]:) $[LD_\beta, LP_\beta, 2, ra, 0, 1, 4, nil, [[s(0), [[a(1), []], [s(2), []]]]]\ ]$
24.(MoveDotUp'[23]:) $[LD_\beta, LP_\beta, 0, rb, 0, 1, 4, nil, [[s(0), [[a(1), []], [s(2), []]]]]\ ]$
25.(Completor[24+7]:) $[LD_\beta, LP_\beta, 0, ra, 0, 1, 4, nil, [[s(0), [[a(1), []], [s(2), []]]]]\ ]$
26.(Adjunctor[21+25]:) $[LD_\alpha, LP_\alpha, 2, rb, 0, -, -, true, [[r(0), [[s(2), [[b(21), []], [x(1), []], [c(22), []]]]]]]\ ]$
27.(Completor[26+4]:) $[LD_\alpha, LP_\alpha, 2, ra, 0, -, -, nil, [[r(0), [[s(2), [[b(21), []], [x(1), []], [c(22), []]]]]]]\ ]$
28.(MoveDotUp'[27]:) $[LD_\alpha, LP_\alpha, 0, rb, 0, -, -, nil, [[r(0), [[s(2), [[b(21), []], [x(1), []], [c(22), []]]]]]]\ ]$
29.(Completor[28+1]:) $[LD_\alpha, LP_\alpha, 0, ra, 0, -, -, nil, [[r(0), [[s(2), [[b(21), []], [x(1), []], [c(22), []]]]]]]\ ]$

Figure 11. State sets constituting the chart for the example of entanglement within elementary structures. The input string is *abxc*.

5 EXTRACTION OF STRUCTURAL DESCRIPTIONS

The algorithm presented in §3 is a recognizer. However, it can be extended to produce the parse trees corresponding to the recognition of an input string in much the same way as in case of the 91parser by including pointers indicating on the basis of which states a new state was added to the chart.

In case of entanglement within elementary trees the extraction of a structural description is more complicated. However, due to the fact that the nodes of the dotted trees in a state representation are labeled by a (non)terminal symbol and the original address in the LD-structure of which the dotted tree is a linearized instance structural descriptions which exhibit entanglement of constituents within an elementary tree can easily be retrieved.

6 TIME COMPLEXITY OF THE ALGORITHM

No rigorous demonstration of time complexity is presented. The conjecture is that with respect to time complexity, the presented algorithm relates to the time complexity of the 91parser, similar to the way direct parsing of ID/LP CFGs relates to Earley's algorithm ([17]). Even though the strong generative capacity of TAG is increased by allowing the entanglement of constituents within elementary trees the time complexity due to the length of the input string of the presented algorithm is identical to that of the 91parser, due to the fact that LP-tests and finding and attaching daughter nodes to a dotted tree take constant time.

However the time complexity of both algorithms is also dependent on grammar size. The worst case time complexity the 91parser is: $\mid G \mid^2 .N.n^6$, where $\mid G \mid$ is the size of the grammar, N is the number of nodes in an LD-structure and n the length of the input string. As shown in [1], in case ID/LP parsing (and also in case of parsing of the restricted variant of TAG(LD/LP) discussed here,) the influence of grammar size in the worst case completely swamps the complexity due to input length: the problem is exponential in grammar size. Which leads to the NP-completeness of the general problem.

Nevertheless, in case of a TAG(LD/LP) the presented algorithm performs considerably better than the 91parser on a corresponding TAG, especially for linguistically motivated grammars for languages with relative free word-order. In addition, the possibility of entanglement within elementary trees does not have a corresponding TAG, because TAG allows only elementary *trees* which necessitates an algorithm as the one presented.

7 CONCLUDING REMARKS

In this paper a generalization of the 91parser for parsing of a restricted variant of TAG(LD/LP) was described. The restricted variant of TAG(LD/LP) allows for the permutation of sister nodes and non-sister nodes within elementary trees. The time complexity due to the length of the input string of the presented algorithm is claimed to be identical to that of the 91parser.

Future research will focus on the extension of the presented algorithm to cope with a unification-based version of the restricted variant of TAG(LD/LP). As noted by Seiffert ([16]), a non-local flow of information in unification grammars which factor LP-information can in connection with the use of restriction to ensure termination of an Earley-type parser in some cases lead to undetected LP-violations. In order to detect these LP-violations it is necessary to double-check the satisfaction of LP-constraints for every single structural description assigned to the input string. This problem is certainly less predominant in case of unification-based TAG, because non-local flow of information is severely reduced as a result of the extended domain of locality.

Another interesting theoretical issue that will be addressed in future research is whether further generalization of the presented algorithm is possible to cope with a less restricted variant of TAG(LD/LP) without an increase of the computational complexity of the algorithm. An interesting less restricted variant of TAG(LD/LP) to investigate would be the variant that establishes strong equivalence to HGs (TAG and HG are proven weakly equivalent in [18].) The restricted variant of TAG(LD/LP) dealt with in this paper is able to assign structural descriptions to input strings that correspond to derivation structures in HGs as proposed by [12] that TAG can not assign ([18]). However, structural descriptions corresponding to derivation structures in HG can also exhibit a specific kind of entanglement between elementary structures. The treatment of entanglement of constituents between elementary structures brings about more severe problems, because intruding constituents come from beyond the domain of locality of the intruded constituent. The conjecture is that the further extension of the presented algorithm along these lines is facilitated by the refined notion of entanglement as adopted in this paper for entanglement within elementary structures.

References

[1] Barton, E., R. Berwick & E. Ristad [1987] *Computational Complexity and Natural Language.* MIT press.

[2] Becker, T., A. Joshi & O. Rambow [1991] Long-distance Scrambling and Tree Adjoining Grammars. In *Proceedings of the 5th conference of the European Chapter of the Association for Computational Linguistics.*

[3] Becker, T. & O. Rambow [1991] *Long-distance Scrambling, TAGs, and HPSG.* Paper presented at the workshop German Grammar in HPSG, Saarbrücken.

[4] Bunt, H. [1991] DPSG and its use in Parsing. In M. Tomita (ed.) *Current Issues in Parsing Technology.* Kluwer Academic Press.

[5] Earley, J. [1970] An Efficient Context-Free Parsing Algorithm. *Communications of the ACM* 13(2).

[6] Gazdar, G., E. Klein, G. Pullum & I. Sag [1985] *Generalized Phrase Structure Grammar.* Basil Blackwell.

[7] Joshi, A., K. Vijay-Shanker & D. Weir [1991] The Convergence of Mildly Context-Sensitive Grammar Formalisms. In P. Sells, S. Shieber & T. Wasow (eds.) *Foundational Issues in Natural Language Processing.* MIT press.

[8] Joshi, A., L. Levy & M. Takahashi [1975] Tree Adjunct Grammars. *Journal of Comput. Syst. Scie.* 10(1).

[9] Joshi, A. [1985] How much Context-Sensitivity is Necessary for Characterizing Structural Descriptions - Tree Adjoining Grammars. In D. Dowty, L. Karttunen & A. Zwicky (eds.) *Natural Language Processing - Theoretical, Computational and Psychological Perspectives.* Cambridge University Press, New York.

[10] Joshi, A. [1987] An Introduction to Tree Adjoining Grammars. In Manaster-Ramer, A. (ed.) *Mathematics of Language*, John Benjamin, Amsterdam.

[11] Joshi, A. [1987] The Relevance of Tree Adjoining Grammar to Generation. Chapter 16 in Kempen, G. (ed.) *Natural Language Generation.* Martinus Nijhoff, Dordrecht.

[12] Pollard, C. [1984] *Generalized Phrase Structure Grammars, Head Grammars and Natural Language.* PhD dissertation, Stanford University.

[13] Poller, P. [1992] Incremental Parsing with LD/TLP-TAGs. In *Proceedings of TAG+ workshop.* Philadelphia.

[14] Schabes, Y. & A. Joshi [1988] An Earley-type Parsing Algorithm for Tree Adjoining Grammars. In *Proceedings 26th Annual Meeting of the Association for Computational Linguistics.* Buffalo.

[15] Schabes, Y. [1991] The Valid Prefix Property and Left to Right Parsing of Tree Adjoining Grammar. In *Proceedings of the International Workshop on Parsing Technologies*. Cancun.

[16] Seiffert, R. [1987] Chart-Parsing of Unification-Based Grammars with ID/LP-Rules. In E. Klein & J. van Benthem (eds.) *Categories, Polymorphism & Unification*. Centre for Cognitive Science, University of Edinburgh/ Institute for Language, Logic & Information, University of Amsterdam.

[17] Shieber, S. [1984], Direct Parsing of ID/LP Grammars. In *Linguistics and Philosophy* 7(2).

[18] Weir, D., K. Vijay-Shanker & A. Joshi [1986] The relationship between Tree Adjoining Grammars and Head Grammars. In *Proceedings 24th Annual Meeting of the Association for Computational Linguistics*.