Parsing Non-Immediate Dominance Relations[*]

Tilman Becker
DFKI GmbH
D-66123 Saarbruecken
becker@dfki.uni-sb.de

Owen Rambow
CoGenTex, Inc.
Ithaca, NY 14850
owen@cogentex.com

Abstract

We present a new technique for parsing grammar formalisms that express non-immediate dominance relations by 'dominance-links'. Dominance links have been introduced in various formalisms such as extensions to CFG and TAG in order to capture long-distance dependencies in free-word order languages (Becker et al., 1991; Rambow, 1994).

We show how the addition of 'link counters' to standard parsing algorithms such as CKY- and Earley-based methods for TAG results in a polynomial time complexity algorithm for parsing lexicalized V-TAG, a multi-component version of TAGs defined in (Rambow, 1994). A variant of this method has previously been applied to context-free grammar based formalisms such as UVG-DL.

1 Linguistic Data

Scrambling is the permutation of arguments of a verb in languages such as Finnish, German, Korean, Japanese, Hindi, and Russian. If there are embedded clauses, certain matrix verbs allow scrambling of elements out of the embedded clauses ("long-distance" scrambling). In German, scrambling is quite free.[1] There is no bound on the number of clause boundaries over which an element can scramble. Furthermore, more than one constituent can scramble in the same sentence ("iterability of scrambling"), and an element scrambled (long-distance or not) from one clause does not preclude an element from another clause from being scrambled.

(1) ...daß [dem Kunden]$_i$ [den Kühlschrank]$_j$ bisher noch niemand t_i
...that [the client]$_{\text{DAT}}$ [the refrigerator]$_{\text{ACC}}$ so far as yet no-one$_{\text{NOM}}$
[[t_j zu reparieren] zu versuchen] versprochen hat
to repair to try promised has
...that so far, no-one yet has promised the client to repair the refrigerator

We conclude that scrambling in German (and other languages) is "doubly unbounded" in the sense that there is no bound on the distance over which each element can scramble (unboundedness of scrambling), and there is no bound on the number of elements that can scramble in one sentence (iterability of scrambling). This generalization should not be taken to mean that all sentences in which "doubly unbounded" scrambling has occurred will be judged equally acceptable. Clearly, scrambling is constrained by pragmatic and processing factors, and perhaps also by semantic factors. For example, processing load appears to increase with an increasing number of scrambled elements. However, we do not have any reason to define a particular "cut-off point" beyond which all orders are ungrammatical. (This argument is similar to the argument in favor of allowing unlimited recursion in the competence grammar.)

Finally, we observe that scrambling does not preclude long-distance topicalization (a separate linguistic phenomenon also found in English, in which a single element moves into sentence-initial position):

[*]We would like to thank Aravind Joshi, Ron Kaplan, Martin Kay, John Maxwell, Giorgio Satta, K. Vijay-Shanker, and David Weir for helpful comments and discussions. This work was partially performed while the authors were at the University of Pennsylvania under grants ARO DAAL 03-89-C-0031; DARPA N00014-90-J-1863; NSF IRI 90-16592; and Ben Franklin 91S.3078C-1.

[1]In German, scrambling can never proceed out of tensed clauses. It is widely assumed that embedded infinitival clauses undergo "clause union". If clause union takes place, then in fact there is no long-distance scrambling in German because no clause boundary is crossed. However, throughout this paper we will use the term "long-distance scrambling" in a more descriptive way. We will take the term to mean that an argument of a verb appears to the left of an argument of a less deeply embedded verb.

(2) [Dieses Buch]$_i$	hat [den Kindern]$_j$	bisher noch niemand	[PRO t$_j$ t$_i$ zu geben]	versucht.
[this book]$_{\mathrm{ACC}}$	has [the children]$_{\mathrm{DAT}}$	so far yet [no-one]$_{\mathrm{NOM}}$	to give	tried

So far, no-one has tried to give this book to the children.

We conclude that doubly unbounded non-local dependencies occur in natural language, that they co-occur with other unbounded dependency constructions (such as topicalization) and that there is a need to account for such constructions, both formally and computationally.

2 V-TAG

Tree Adjoining Grammar (TAG, see (Joshi, 1987) for an introduction) is a tree rewriting system. A TAG consists of a set of *elementary trees* which are combined by the *adjoining* operation, which may insert one tree into the center of another. TAGs are more powerful than context-free grammars but they are only mildly context-sensitive since they generate only semi-linear languages and are polynomially parsable. Their extended domain of locality allows the factoring of recursion and the statement of linguistic dependencies within the elementary structures of the grammar.

However, in (Becker et al., 1991), we argue that scrambling is beyond the formal power of TAG by assuming that elementary trees express a complete predicate-argument structure. (Becker et al., 1991) proposes two variants of the TAG formalism which can derive scrambling while still preserving most of the desirable properties of TAGs (in particular, an extended domain of locality and the factoring of recursion). One of them, FO-TAG, is based on relaxing LP constraints, and we do not discuss it here. The other approach is based on relaxing immediate dominance. This has the effect of creating several "chunks" of the tree which are related by (not necessarily immediate) dominance edges or *dominance links.* Dominance links are essential for encoding structural relations (c-command) between related linguistic elements, such as a head and its arguments.[2] The resulting formalism is called Multi-Component TAG with Dominance Links (MC-TAG-DL).

In defining MC-TAG-DL, several options are available for the definition of the derivation relation. In V-TAG (Rambow, 1994), there are no restrictions on adjunction sites. Trees from one tree set can be adjoined anywhere in the derived tree, and they need not be adjoined simultaneously or in a fixed order. Furthermore, trees in the tree sets are equipped with dominance links. A dominance link can relate the foot node of a tree to any node in any tree of the same set. The dominance links provide a constraint on possible derivations: after a derivation is completed, each dominance link must hold in the derived tree.

We give a V-TAG derivation for sentence (2). Scrambling will be modeled by multi-component adjunction, while Topicalization is derived by standard adjunction. The grammar is a set of tree sets. Each tree set contains a head (e.g., a verb) and its projections, and slots for its arguments. Two examples are shown in Figure 1. We use IP (= S) and CP (= S′) as projections of the verb. In the set for the *geben* 'to give' embedded clause, one nominal argument is in a separate auxiliary tree, reflecting the fact that it may be scrambled, and the other nominal argument is included in the verbal projection tree, reflecting the fact that it is (long-distance) topicalized. The dotted line represents the dominance link. In the set for the *versuchen* 'to try' matrix clause, the only nominal argument is in a separate auxiliary tree. Its clausal subcategorization requirement is indicated by the fact that the verb is in an auxiliary tree (rooted in C′), forcing adjunction into an embedded clause.

The derivation now proceeds by first adjoining the matrix clause into the embedded clause at the C′ node, yielding the structure on the left in Figure 2. This adjunction implements the long-distance topicalization of the embedded direct argument. We are left with two auxiliary trees that still need to be adjoined, representing the scrambled arguments. We first adjoin the

[2]Without being formally defined, dominance links have been used previously in linguistic work, for example (Kroch, 1989).

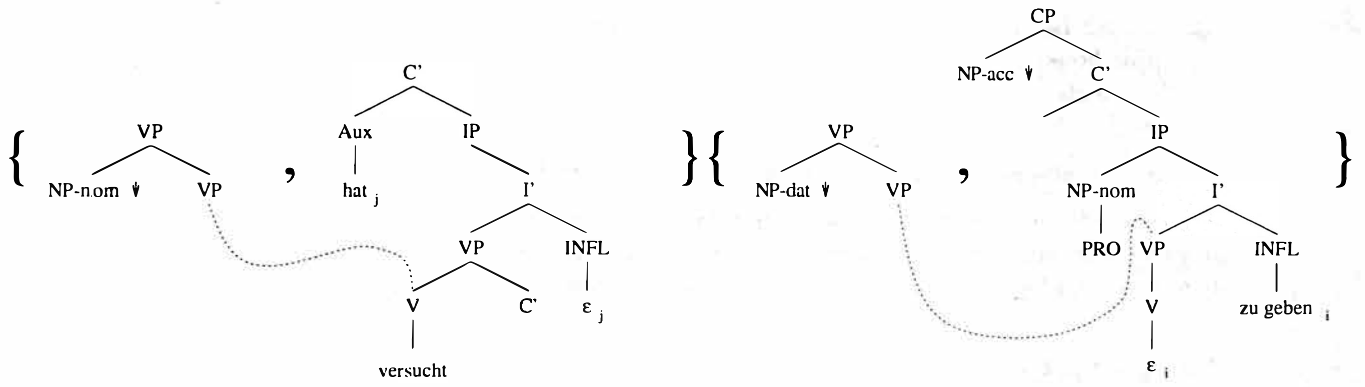

Figure 1: Initial tree set for *versuchen* matrix clause and *geben* embedded clause

matrix subject into its own clause, and then adjoin the embedded indirect object just above the matrix subject. The result is shown in Figure 2 on the right.

Observe that the tree sets given in Figure 1 have the property that they each represent a verb. In linguistic applications of TAG and related formalisms such as V-TAG, it is useful to associate each elementary structure (tree set in the case of V-TAG) with at least one lexical item (i.e., terminal symbol). Such a grammar is called "lexicalized". This has an important consequence, namely that derivations in a lexicalized grammar are always bounded in length by a linear function of the length of the derived sentence. In the following discussion of a parser for V-TAG, we will make crucial use of this property.

3 Parsing V-TAG

Both parsing algorithms presented in this paper deal only with adjunction, the core operation in TAG and in V-TAG. We ignore the substitution operation as well as constraints on adjunction which can easily be added and do not contribute to the complexity of the parsing algorithms.

In this section, we use an extension of the CYK-type parser for TAG defined by Vijay-Shanker (1987, p.110) to give a polynomial time parser for a large subset of the V-TAG languages. We first describe Vijay-Shanker's parser for simple TAG, and then describe the extensions necessary for V-TAG.

The main idea of Vijay-Shanker's parser is the introduction of a 4-dimensional matrix T, in which an entry of a node η from an elementary tree τ at $T[i,j,k,l]$ represents the fact that either

(i) there is some derived tree τ' such that η is its root node and η dominates the substring $a_{i+1}\cdots a_j\eta_1 a_k\cdots a_l$ where η_1 is the (label of the) foot node of τ or

(ii) there is some derived tree τ' such that η is its root node and η dominates the substring $a_{i+1}\cdots a_l$ and $j = k$.

The parser fills the matrix T bottom-up, starting from entries for the leaves. (We assume that the grammar is in extended two form, i.e., in every tree every node has at most two children.) In the presentation here, we split every node into a top and a bottom version, similar to the definition of "top" and "bottom" features in a feature-based TAG (Vijay-Shanker, 1987). If η is a node in some tree of some set of a V-TAG, then η^{T} denotes the top version of that node, and η^{B} the bottom version. There are six cases which fall into two basic categories:

(i) Cases 1 to 4 and correspond to context-free expansions within one elementary tree. In Cases 1 to 3, two top versions of sibling nodes η_1 and η_2 are combined and the bottom version of their parent η is added to T if we know that η_1 and η_2 cover a contiguous string (possibly interrupted only by the foot node). In Case 1 the footnote is dominated by η_1, in Case 2 it is dominated by η_2 and in Case 3 it is not dominated by either. Case 4 deals with nodes without siblings. Figure 3 shows Case 1.

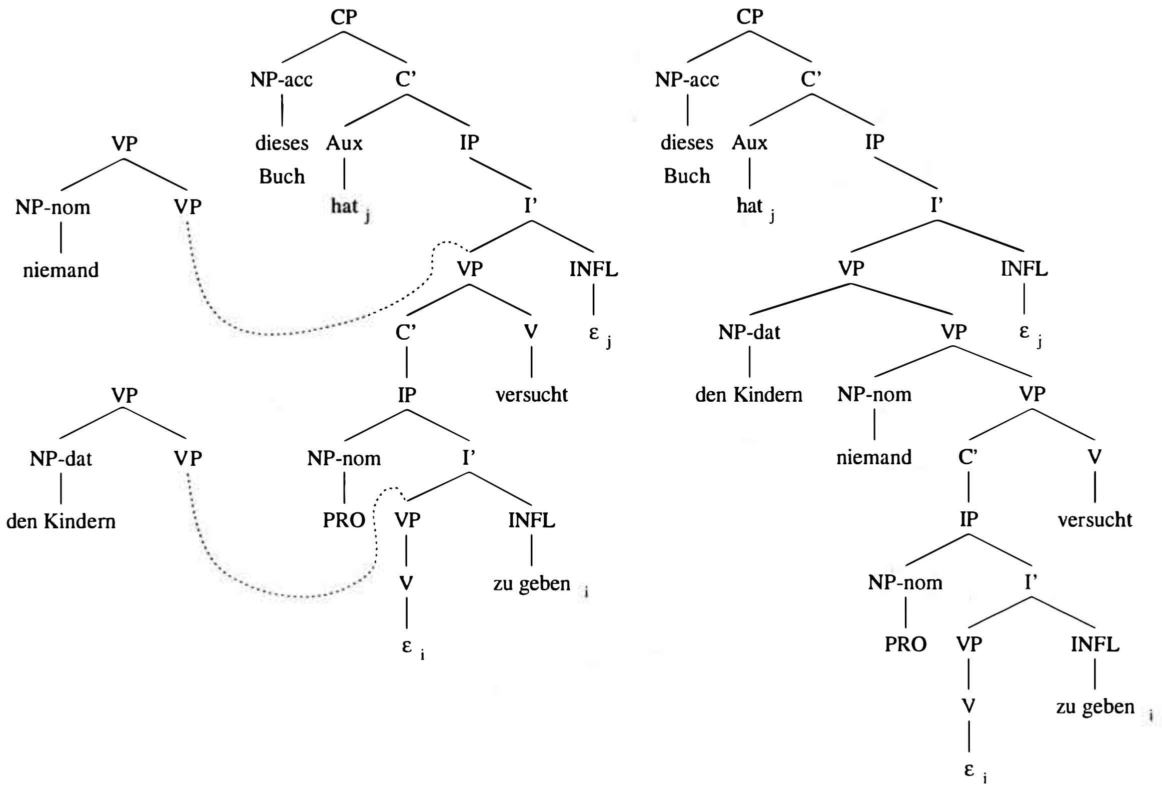

Figure 2: After adjoining matrix clause into subordinate clause (left) and final derived tree (right)

(ii) Cases 5a, 5b, and 6 deal with adjunction. Cases 5a and 5b correspond to adjunction (either at a node which dominates the foot node (5a) or not (5b)). The top version of the node is added to the matrix to reflect the string covered after adjunction at that node has taken place, as illustrated in Figure 3 for Case 5a. Case 6 corresponds to no adjunction: the top version of a node is added if the bottom version is already present in the same cell of the matrix.

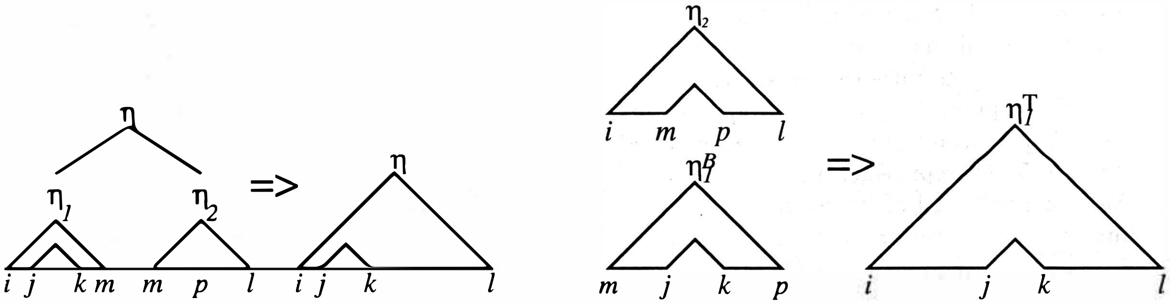

Figure 3: Cases 1 and 5a.

We now turn to the extensions necessary to handle V-TAG. We first introduce some additional terminology. If two nodes η_1 and η_2 are linked by a dominance link such that η_1 dominates η_2, then we will say that η_1 has a *required bottom* and that η_2 has a *required top*. If the tree of which η_1 (η_2) is a node has been adjoined during a derivation, but the tree of which η_2 (η_1) is a node has not, the requirement (top or bottom) will be called *unfulfilled*. The multiset of unfulfilled required tops of a node η will be denoted by $\top(\eta)$, and the multiset of all required bottoms will be denoted by $\bot(\eta)$. We extend this notation to derived trees in the obvious way. Observe that a (partial) derived initial tree (i.e., a tree without a foot node on its frontier) cannot have any unfulfilled required bottoms if it is to be part of a successful derivation.

Clearly, in a lexicalized V-TAG, in every derivation $|\top(\tau)|$ and $|\bot(\tau)|$ are always linear with respect to the length of the input string. Note that a V-TAG such that $|\top(\tau)|$ and $|\bot(\tau)|$ are always bounded by a constant c in every derivation is equivalent to a TAG.

In order to keep track of unfulfilled dominance requirements, we add to each entry in the matrix two link-counters which record the number and type of required tops and bottoms, respectively, which still need to be fulfilled. A link-counter γ is an array whose elements are indexed on the dominance links of G, and whose values are integers.[3] The sum of two counters is defined component-wise, the norm $|\gamma|$ is defined as the sum of the values of all components. We will denote by $\gamma^\top$ the required tops counter, by $\gamma^\bot$ the required bottoms counter, and by 0 the counter all of whose values are 0.

The need for maintaining two link counters – one for required tops and one for required bottoms – arises from the fact that entries in the four-dimensional parse matrix do not span a contiguous substring of the input string. Thus a (partial) derivation of a (derived) auxiliary tree may have required bottoms as well as required tops which will only be fulfilled once this structure is adjoined into another structure.

We now spell out what happens to the link counters in the cases 1 and 5a. The other cases are analogous. In the following, $a \dot{-} b$ is defined to be $a - b$ if $a \geq b$, and 0 otherwise.

Case 1: η_1 dominates the foot node (see Figure 3). If there is $(\eta_1^\top, \gamma_1^\bot, \gamma_1^\top) \in T[i, j, k, m]$ and $(\eta_2^\top, 0, \gamma_2^\top) \in T[m, p, p, l]$, $i \leq j \leq k \leq m \leq p \leq l$, then add $(\eta^B, \gamma_1^\bot, \gamma_1^\top + \gamma_2^\top + \top(\eta))$ to $T[i, j, k, l]$.

Case 5a: η_1 dominates the foot node. If there is $(\eta_1^B, \gamma_1^\bot, \gamma_1^\top) \in T[m, j, k, p]$ and $(\eta_2^\top, \gamma_2^\bot, \gamma_2^\top) \in T[i, m, p, l]$ where η_2 is the root node of an auxiliary tree with the same symbol as η_1, then add $(\eta_1^\top, (\gamma_2^\bot \dot{-} \gamma_1^\top) + \gamma_1^\bot, (\gamma_1^\top \dot{-} \gamma_2^\bot) + \gamma_2^\top)$ to $T[i, j, k, l]$.

In all six cases, after calculating the new $\gamma^\bot$ and $\gamma^\top$, the entry is discarded if $|\gamma^\bot + \gamma^\top| \geq c \cdot n$, where c is the maximal number of links in a tree set of the grammar. The recognition of a string $a_1 \cdots a_n$ is successful if for some j, $0 \leq j \leq n$, and some η, a root node of an initial tree, we have $(\eta^\top, 0, 0) \in T[0, j, j, n]$.

In case 1 (see figure 3) $\eta_1^\top$ represents a partial derivation of a (derived) auxiliary tree where η_1 dominates the foot node. The required bottoms link-counter $\gamma_1^\bot$ represents links that go down from some nodes in the partially derived tree. These nodes must be on the spine of the tree and they can only be fulfilled after the tree is adjoined into another tree. Therefore, these links must 'go through' the foot node. For the node η dominating η_1 the required bottom counter is simply copied. (Recall that only foot nodes can be at the top end of a link.) η_2 cannot contribute any required bottoms since the substring underneath η_2 is already completely derived and no nodes will added underneath η_2. The required tops however are simply added from $\gamma_1^\top$, $\gamma_2^\top$ and $\top(\eta)$ since they can be fulfilled by nodes which are added later above η.

In case 5a η_1^B represents a partial derivation of a (derived) auxiliary tree where η_1 dominates the foot node. All of its required bottoms ($\gamma_1^\bot$) must be added to the new required bottoms link-counter. However, some of the required bottoms of the adjoined auxiliary tree ($\gamma_2^\bot$) might have been fulfilled by required tops at node η_1, so only $\gamma_2^\bot \dot{-} \gamma_1^\top$ many are added to the new required bottoms link-counter. For the same reason, not all required tops of η_1 are added to the new required tops link-counter, but only $\gamma_1^\top \dot{-} \gamma_2^\bot$ many. However, all required tops of η_2, i.e. $\gamma_2^\top$ are added to the new required tops link-counter.

The core of the algorithm is a loop through indices $i, j, k, l \in \{0..n\}$, $i \leq j \leq k \leq l$, applying Cases 1 through 6 until the matrix is unchanged.

Using back pointers (e.g., for every $(\eta, \gamma^\bot, \gamma^\top)$ which is added to T, pointers to the contributing nodes η_1 (and η_2) in their respective positions are added), the matrix T can be augmented to

[3]Link-counters are used for an extension to CFG (called UVG-DL) in (Rambow, 1994), and for a different CFG-based system with dominance links (called D-Tree Grammar) in (Rambow et al., 1995b). The present paper is similar to (Rambow et al., 1995b) in that it it is concerned with a formal system that includes dominance links. In both papers, these are parsed using multisets, and the parsers are polynomial for the same reason. However, the fact that V-TAG is a tree rewriting system which includes adjunction provides much of the conceptual complexity of the algorithms presented here.

represent a parse forest from which all derivations of an accepted string can be constructed.
Theorem: A lexicalized V-TAG is parsable in deterministic polynomial time.

The correctness of the recognition algorithm for TAG is proven by Vijay-Shanker (1987). It can easily be seen by induction on the number of dominance links that the link-counters correctly impose the dominance constraints.

The time complexity of the algorithm is that of Vijay-Shanker's algorithm, $O(n^6)$, multiplied by a factor representing the maximal number of elements of each cell of matrix T. Since $|\gamma^\perp|, |\gamma^\top|$ are in $O(n)$, we have that the number of possible pairs of link-counters is bounded by $O(n^{2|L|})$ (where $|L|$ is the total number of links in G), and the the time complexity of the algorithm is in $O(|G|n^{4|L|+6})$.

While for a linguistic grammar with significant coverage $|L|$ may be quite large, in practice the time complexity of the parser will be much lower. This is because in general, the number of different link-counters associated with the same node in any square of the parse matrix will be low. Several factors contribute to this fact.

First, recall that we are using multicomponent adjunction only for scrambling (and perhaps certain forms of "long" topicalization out of picture NPs), and not for raising and standard topicalization. These syntactic phenomena are still derived by simple adjunction, which does not contribute to the link counters.

Second, since CKY is a pure bottom-up parser, we need not actually distinguish between required top links which have the same tree at their top end. In fact, in scrambling languages such as German, the set of trees at the top end of dominance links is quite limited: it consists of substitution structures for nominal and clausal arguments, perhaps one for each case or verbal form. Thus $|L|$ in fact is a small number (say, four to six) rather than the large number one obtains if one counts the dominance links in the elementary structures for every single verb in the language.

Finally, observe that if we choose a constant c and fix the number of open dominance links at any point in the derivation to be less than or equal to c, then we obtain a parser that simply runs in time $O(n^6)$. This is because the numbers of possible entries in the square of the parse matrix is again bounded by a grammar constant, part of which is c. In this case, we can view the formalism and the parser as a dynamic implementation of "slash" categories. Since in fact multiple scrambling is quite rare in real text, we can choose such a constant – say, $c = 3$ – and obtain a $O(n^6)$ parser for all but a tiny percentage of the sentences of the language in question. Determining the proper number may require empirical investigations.

The data structures which are built up in our parsing algorithms also yield themselves to an iterative algorithm if we gradually increase the maximum link constant c. The steps of such an iterative algorithm all have time complexity $O(n^6)$. Careful bookkeeping minimizes overhead with respect to a non-iterative approach. An iterative strategy will find "unscrambled" parses earlier than "scrambled" parses. In particular, we can continue the iteration process only if the "unscrambled" parses do not meet semantic or pragmatic conditions. This approach avoids many false ambiguities that arise when the parser postulates multiple scrambling, when in fact none has occurred.

4 An Earley-Type Parsing Algorithm

We now briefly describe an extension of the Earley-based TAG parsing algorithm of (Schabes, 1990) which results in a practical parsing algorithm for V-TAG. Again, we use link counters to keep track of unfulfilled dominance requirements. We will only give an informal review of the original parsing algorithm. For full details, we refer the reader to (Schabes, 1990). However, some detail is necessary to explain our extensions.

The basic data structure is a 'dotted tree'. A dotted tree is an elementary tree, usually an auxiliary tree, with a dot marking a node in this tree, together with two intervals (i.e. four indices) which represent the recognized strings underneath the marked node; one to the left and one to the right of the foot node. For a dot on a node, there are four possible positions: it is either to the left (prediction phase) or the right (completion phase) and independent of this it is either above or below the node.

The parser proceeds in a mixed fashion with top-down prediction steps and a bottom-up completion steps. Beginning with an initial tree and the dot in the left-above position on the root node, the dot is propagated in a depth-first left-to-right fashion through the entire tree until it returns to the root node (to the right-above position). Dotted trees are collected in states sets S_i which contain all dotted trees that represent a partial derivation that covers the input string up to position i.

In order to extend the algorithm to V-TAG, we again use 'link counters' to store the information about unfulfilled dominance constraints. Every dotted tree is extended by a link counter that for each link in the grammar counts the number of required bottoms in the partial derivation which is represented by the dotted tree. Since every state in the parsing algorithm is reached through a sequence of top-down and bottom-up steps which begins with the root node of the initial tree, there is never a need to keep track of required tops. As long as the dot is on the left, it moves downward, collecting required bottoms. If it hits required tops on a node, it either subtracts from the list of required bottoms or, if there are none, it immediately signals failure, since all nodes above a given node in a partial derivation have been visited already. When the dot is on the right, the bottom-up steps always refer to already visited structures (see especially the tests in the 'move dot up' and 'right-prediction' steps).

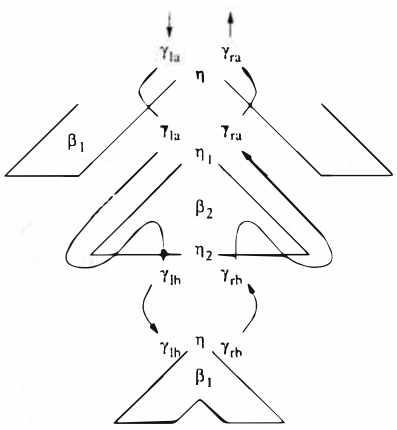

Figure 4: Percolation of the link counter.

Figure 4 shows the movements of the dot and the corresponding link counters on one node η for all relevant steps of the parsing algorithm. The node η is shown twice, with the dots in the above positions on the top of the figure, an auxiliary tree that might adjoin is shown in the middle, and on the bottom again the node η is shown with the dots in the below positions.

In the first step with the dot in the left-above position of node η of a dotted tree β_1, the corresponding link counter is γ_{1-la}. It contains all required bottoms in this partial derivation. Note that for this partial derivation, the dot has been moved through all positions above and to the left of η. The left-prediction step creates an entry for every auxiliary tree β' that can be adjoined, with the dot in the left-above position on the root node. The link counter γ_{1-la} is copied and if the root node of β' fulfills some required bottoms, the link counter is decremented accordingly. When moving the dot has proceeded through β' to the left-below position of the foot node, a left-completion step for β_1 can take place. The link-counter γ'_{lb} is copied as γ_{lb}. An alternative left-completion step assumes that no adjunction has taken place on node η and γ_{la} is copied as γ_{lb}. If node η of β_1 is itself a foot node and introduces a required bottom, then the link-counter γ_{lb} is incremented accordingly.

The next move of the dot is the 'move dot down' to the left-above position of the leftmost daughter of node η. If this daughter node fulfills some required bottom, the link-counter is decremented accordingly.

Now the dot moves until it reaches the right-below position of η. If the node η itself is a foot node and has a required bottom, the relevant component of the link-counter γ_{lb} is decremented. If this is impossible (because the value is zero), then the derivation fails. Then, the new link-counter γ_{rb} must be compared with the old link-counter γ_{lb}. If any component of the link counter γ_{rb} is greater than γ_{lb}, something has gone wrong and the dot is not moved. This can only happen if new required bottoms are added, but not fulfilled below η. Since for this partial derivation, the dot has moved through the entire derived tree below η, these required bottoms can never be fulfilled. The old link-counter γ_{lb} must be retrieved from the appropriate set S_i.

The next step is the (right-)prediction of an adjunction at node η. If there is a dotted (auxiliary) tree β' covering an immediate preceding substring with the dot in the left-below position of the foot node, this dot is moved to the right-below position. The counter γ_{rb} is copied as γ_{2-rb}.

Again, there are two alternative right-completion steps. Assuming that no adjunction has taken place, the dot is moved up to the right-above position. The new link-counter γ_{ra} is copied from γ_{rb}.

If there is a dotted (auxiliary) tree β' covering an immediate surrounding substring in which the dot has moved all the way up to the right-above position of the root node, the adjunction of this tree is assumed and the new link-counter γ_{ra} is copied from γ_{2-ra}.

From there, the dot is moved up, either to the left-above position of the right sister node, or, if there is no right sister, to the right-below position of the mother node. In the first case the link-counter is copied without changes, in the second case the above mentioned test is performed,

Also, in the scanner steps, the link counter is copied without changes.

Note that the only point in which the comparison of new and old link-counters is necessary is the arrival of the dot in a right-below position. In all moves, after calculating the new link-counter γ, the entry is discarded if $|\gamma| \geq c \cdot n$, where c is the maximal number of links in a tree set of the grammar. The recognition of a string $a_1 \cdots a_n$ is successful if $[\alpha, 0, ra, 0, -, -, n]$ is in S_n with α an initial tree.

The core of the algorithm is a loop from $i := 0$ to n through the sets S_i where the modified steps of the Earley-style parser are applied until no more states can be added.

The correctness of the recognition algorithm for TAG is proven by Schabes (1990). It can easily be seen by induction on the number of dominance links that the link-counters correctly impose the dominance constraints.

The time complexity of the algorithm is that of Schabes' algorithm, $O(n^6)$, multiplied by a factor representing the maximal number of entries in the states sets S_i which differ only in the link counters γ. Since $|\gamma| \leq cn$, we have that the number of possible link-counters is bounded by $O(n^{2\times|L|})$ (where $|L|$ is the total number of links in G), and the the time complexity of the algorithm is in $O(|G|n^{2\times|L|}n^6)$.

Again, using back pointers, the entries in the states sets can be augmented to represent a parse forest from which all derivations of an accepted string can be constructed. The discussion about the relevance of the exponents and an iterative algorithm from section 3 also applies to this Earley-style parsing algorithm.

Bibliography

Becker, Tilman; Joshi, Aravind; and Rambow, Owen (1991). Long distance scrambling and tree adjoining grammars. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL'91)*, pages 21–26. ACL.

Joshi, Aravind K. (1987). An introduction to Tree Adjoining Grammars. In Manaster-Ramer, A., editor, *Mathematics of Language*, pages 87–115. John Benjamins, Amsterdam.

Kroch, Anthony (1989). Asymmetries in long distance extraction in a Tree Adjoining Grammar. In Baltin, Mark and Kroch, Anthony, editors, *Alternative Conceptions of Phrase Structure*, pages 66–98. University of Chicago Press.

Rambow, Owen (1994). *Formal and Computational Aspects of Natural Language Syntax*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia. Available as Technical Report 94-08 from the Institute for Research in Cognitive Science (IRCS).

Rambow, Owen; Vijay-Shanker, K.; and Weir, David (1995a). D-tree grammars. In *33rd Meeting of the Association for Computational Linguistics (ACL'95)*. ACL.

Rambow, Owen; Vijay-Shanker, K.; and Weir, David (1995b). Parsing D-Tree Grammars. Fourth International Workshop on Parsing Technologies.

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

Vijay-Shanker, K. (1987). *A study of Tree Adjoining Grammars*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.