PARSING D-TREE GRAMMARS

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

K. Vijay-Shanker
Department of Computer &
Information Science
University of Delaware
Newark, DE 19716
vijay@udel.edu

David Weir
School of Cognitive &
Computing Sciences
University of Sussex
Brighton, BN1 9HQ, UK.
david.weir@cogs.susx.ac.uk

Word Count: 6213 words

Abstract

In this paper we will describe a polynomial time Earley-style predictive parser for D-Tree Grammars (DTG). DTG were designed to share some of the advantages of TAG while overcoming some of its limitations. In developing parsers for TAG it has turned out to be useful to make use of a direct mapping from TAG to Linear Indexed Grammars (LIG) and to develop algorithms that work with the resulting LIG. In the case of DTG we use a similarly direct mapping into what we call Linear Prioritized Multiset Grammar (LPMG). This makes it possible to give a straightforward statement of the parsing algorithm.

1. Motivation

Rambow, Vijay-Shanker and Weir (1995), we define a new grammar formalism, called D-Tree Grammars (DTG), which arises from work on Tree-Adjoining Grammars (TAG) (Joshi et al., 1975). A salient feature of TAG is the extended domain of locality it provides. Each elementary structure can be associated with a lexical item (as in Lexicalized TAG (LTAG) (Joshi and Schabes, 1991)). Properties related to the lexical item (such as subcategorization, agreement, certain types of word order variation) can be expressed within the elementary structure (Kroch, 1987; Frank, 1992). In addition, TAG remain tractable, yet their generative capacity is sufficient to account for certain syntactic phenomena that, it has been argued, lie beyond Context-Free Grammars (CFG) (Shieber, 1985). TAG, however, has two limitations. The first problem is that the TAG operations of substitution and adjunction do not map cleanly onto the relations of complementation and modification. A second problem has to do with the inability of TAG to provide analyses for certain syntactic phenomena. We will discuss the first issue in some detail here, and refer to Rambow et al. (1995) for a broader discussion of linguistic data that motivates the definition of DTG.

In LTAG, the operations of substitution and adjunction relate two lexical items. It is therefore natural to interpret these operations as establishing a direct linguistic relation between the two lexical items, namely a relation of complementation (predicate-argument relation) or of modification. In purely CFG-based approaches, these relations are only implicit. However, they represent important linguistic intuition, they provide a uniform interface to semantics, and they are, as Schabes and Shieber (1994) argue, important in order to support statistical parameters in stochastic frameworks and appropriate adjunction constraints in TAG. In many frameworks, complementation and modification are in fact made explicit: LFG (Bresnan and Kaplan, 1982) provides a separate functional (f-) structure, and dependency grammars (see e.g. Mel'čuk (1988)) use these notions as the principal basis for syntactic representation. We will follow the dependency literature in referring to complementation and modification as syntactic dependency. As observed by Rambow and Joshi (1992), for TAG, the importance of the dependency structure means that not only the derived phrase-structure tree is of interest, but also the operations by which we obtained it from elementary structures. This information is encoded in the derivation tree (Vijay-Shanker, 1987).

However, as Vijay-Shanker (1992) observes, the TAG composition operations are not used uniformly: while substitution is used only to add a (nominal) complement, adjunction is used both for modification and (clausal) complementation. Clausal complementation could not be handled uniformly by substitution because of the existence of syntactic phenomena such as long-distance *wh*-movement in English. Furthermore, there is an inconsistency in the directionality of the operations used for complementation in TAG: nominal complements are substituted into their governing verb's tree, while the governing verb's tree is adjoined into its own clausal complement. The fact that adjunction and substitution are used in a linguistically heterogeneous manner means that (standard) TAG derivation trees do not provide a good representation of the dependencies between the words of the sentence, i.e., of the predicate-argument and modification structure.

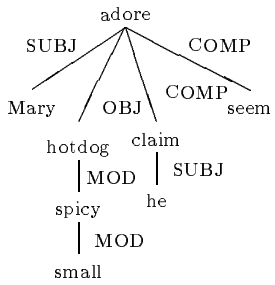

Figure 1: TAG Derivation trees for (1)

For instance, English sentence (1) gets the derivation structure shown on the left in Figure 1[1].

(1) Small spicy hotdogs he claims Mary seems to adore

When comparing this derivation structure to the dependency structure in Figure 2, the following problems become apparent. First, both adjectives depend on *hotdog*, while in the derivation structure *small* is a daughter of *spicy*. In addition, *seem* depends on *claim* (as does its nominal argument, *he*), and *adore* depends on *seem*. In the derivation structure, *seem* is a daughter of *adore* (the direction does not express the actual dependency), and *claim* is also a daughter of *adore* (though neither is an argument of the other).

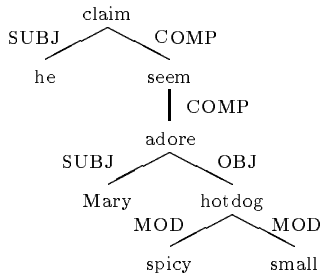

Figure 2: Dependency tree for (1)

Schabes and Shieber (1994) solve the first problem by distinguishing between the adjunction of modifiers and of clausal complements. While this might provide a satisfactory treatment of modification at the derivation level, there are now three types of operations (two adjunctions and substitution) for two types of dependencies (arguments and modifiers), and the directionality problem for embedded clauses remains unsolved.

[1]For clarity, we depart from standard TAG notational practice and annotate nodes with lexemes and arcs with grammatical function.

In defining DTG we have attempted to resolve these problems with the use of a single operation (that we call subsertion) for handling all complementation and a second operation (called sister-adjunction) for modification. However, the definition remains faithful to what we see as the key advantages of TAG (in particular, its enlarged domain of locality).

We mention in passing that our definition also handles certain syntactic phenomena that are beyond the scope of simple TAG, such as extraction from picture-NPs in English, *wh*-movement in Kashmiri, and long-distance scrambling in Hindi, Japanese, Korean, Turkish, and other languages. These syntactic phenomena have inspired several other extensions to formalisms, see Becker, Joshi and Rambow (1991), the V-TAG formalism of Rambow (Rambow, 1994a)[2], and an extension of categorial grammars developed by Hoffman (1995).

2. Definition of DTG

We begin with the definition of the structures manipulated by a DTG. A **d-tree** is a tree with two types of edges: domination edges (**d-edges**) and immediate domination edges (**i-edges**). In defining d-trees, we specify additional constraints on the distribution of d-edges and i-edges. For each internal node, either all of its daughters are linked by i-edges or it has a single daughter that is linked to it by a d-edge. Each node is labelled with a terminal symbol, a nonterminal symbol or the empty string. A d-tree containing n d-edges can be decomposed into $n+1$ **components** containing only (0 or more) i-edges, with the root of all components other than topmost one connected to a node in a component above by a d-edge. In the d-trees α and β shown in Figure 3, these components are shown as triangles.

We introduce *subsertion* as an operation on d-trees. When a d-tree α is subserted into another d-tree β, a component of α is substituted at a frontier nonterminal node (a **substitution node**) of β and all components of α that are above the substituted component are inserted into d-edges in β above the substituted node or placed above the root node of β. For example, Figure 3 illustrates a possible subsertion of α in β. Here we have considered the component $\alpha(5)$ as being substituted at a substitution node in β. The components, $\alpha(1)$, $\alpha(2)$, and $\alpha(4)$ of α above $\alpha(5)$ drift up the path in β which runs from the substitution node. While in the composed d-tree shown, γ, all these components have been inserted in the d-edges of β, in general, some of these components could have been placed above the root of β. When a component $\alpha(i)$ of some d-tree α is inserted into a d-edge between nodes η_1 and η_2 two new d-edges are created, the first of which relates η_1 and the root node of $\alpha(i)$, and the second of which relates the frontier node of $\alpha(i)$ that dominates the substituted component to η_2. It is possible for components above the substituted node to drift arbitrarily far up in any way that is compatible with the domination relationships present in the substituted d-tree. DTG provide a mechanism called **subsertion-insertion constraints** to control what can appear within d-edges (see below).

The second composition operation involving d-trees is called sister-adjunction. We insist that the target node of sister-adjunction must be a node at the top of an i-edge. When a d-tree α is sister-adjoined at a node η in a d-tree β the composed d-tree γ results from the addition to β of α as a new leftmost or rightmost sub-d-tree below η. Note that sister-adjunction involves the addition of exactly one new immediate domination edge and that several sister-adjunctions can occur at the same node. **Sister-adjoining constraints** specify where d-trees can be sister-adjoined and whether they will be right- or left-sister-adjoined (see below).

D-edges and i-edges are intended to express domination and immediate domination relations between nodes. Note by using these two operations, these relations are never rescinded when d-trees are composed. Nodes separated by an i-edge will remain in a mother-daughter relationship through-

[2]Becker and Rambow (1994; 1995) describe a parser for V-TAG a related formal system that includes dominance links and uses multisets in parsing. However, unlike V-TAG, DTG give rise to linguistically meaningful derivation structures; DTG do not include the TAG tree adjunction operation, which simplifies certain aspects of the algorithm; and the parser is defined in terms of a related string-rewriting formalism.

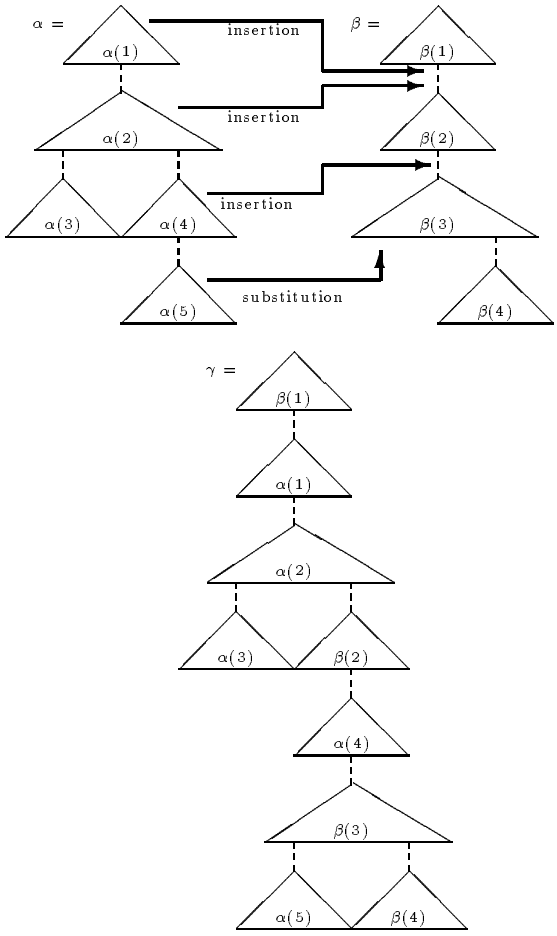

Figure 3: Subsertion

out the derivation, whereas nodes separated by an d-edge can have a path of any length inserted between them during a derivation.

A DTG is a four tuple $G = (V_N, V_T, S, D)$ where V_N and V_T are the usual nonterminal and terminal alphabets, $S \in V_N$ is a distinguished nonterminal and D is a finite set of **elementary** d-trees. A DTG is said to be **lexicalized** if each d-tree in the grammar has at least one terminal node. The elementary d-trees of a grammar G have two additional annotations: subsertion-insertion constraints and sister-adjoining constraints. These will be described below, but first we define simultaneously DTG derivations and subsertion-adjoining trees (SA-trees), which are partial derivation structures that can be interpreted as representing dependency information, the importance of which was stressed in the introduction[3]. SA-trees contain information regarding the substitutions and sister-adjunctions that take place in a derivation, capturing the linguistic dependency relations of complementation and modification. Thus the SA-trees are exactly the dependency trees discussed earlier as motivations for DTG.

Consider a DTG $G = (V_N, V_T, S, D)$. In defining SA-trees, we assume some naming convention for the elementary d-trees in D and some consistent ordering on the components and nodes of elementary d-trees in D. Similar to the derivation trees in TAG, the nodes in SA-trees are labeled by the names of the elementary d-trees. An SA-tree consisting of a single node labelled by an elementary d-tree α will correspond to the derived d-tree that is just this elementary d-tree. We

[3]Due to space limitations, in the following definitions we are forced to be somewhat imprecise when we identify a node in a derived d-tree with the node in the elementary d-trees (elementary nodes) from which it was derived. This is often done in TAG literature, and hopefully it will be clear what is intended.

say that $\alpha \in T(\alpha)$. The components of the elementary d-tree are considered the **substitutable**[4] components of this tree considered as a derived d-tree.

Let α be an elementary d-tree with k substitution nodes. Subsertion (substitution) must take place at each of these nodes. Further, when we consider sister-adjunction into α, these substitution nodes cannot be the target of sister-adjunction, and vice-versa. The combination by (k) subsertions and $(j \geq 0)$ sister-adjunctions into an instance of α is represented by an SA-tree, say τ, with root labelled α which has $(k+j)$ children. k of these children represent the k d-trees being subserted and the remaining j, the d-trees being used in the sister-adjunction. As more than one d-tree can be sister-adjoined at the same node in α, and as different sequences of sister-adjunctions at a node could lead to different d-trees being derived, the children of nodes in an SA-tree corresponding to those used in sister-adjunction at the same node have to be ordered with respect to each other. Without loss of generality, we assume the children of a node in an SA-tree are ordered, and in particular, we assume that initial sequences of children of a node in an SA-tree correspond to subsertion[5]. Let the children of root of τ be labelled $\alpha_1, \ldots \alpha_{k+j}$ respectively, with $\tau_1, \ldots, \tau_{k+j}$ being the sub-SA-trees rooted at these nodes. Inductively, we assume τ_i correspond to the derivation of d-trees $\gamma_i \in T(\alpha_i)$ for $i \leq (k+j)$. As α has k substitution nodes k of these derived d-trees are subserted at the k substitution nodes and the rest are sister-adjoined. Thus, $\gamma_1 \in T(\alpha_1), \ldots, \gamma_k \in T(\alpha_k)$ are assumed to be subserted. The k-subsertions are indicated in τ by labelling the corresponding edges by pairs (l_i, n_i) for $1 \leq i \leq k$. This label indicates the substitution of the l_i^{th} substitutable component of γ_i at the substitution node in α with address n_i. $\gamma_{k+1}, \ldots \gamma_{k+j}$ are sister-adjoined. The corresponding edges in τ are labelled by pairs (d_i, m_i), for $k < i \leq k+j$, with m_i giving the address of the target of sister-adjunction and $d_i \in \{\mathsf{left}, \mathsf{right}\}$ used to indicate whether left- or right-sister-adjunction took place. τ is the SA-tree for the d-tree, $\gamma \in T(\alpha)$, obtained by this sequence of substitutions and sister-adjunctions into α. The new components of γ that came from α are now the substitutable components of γ and are ordered (or numbered) in the same manner as in α. Note that τ may be the SA-tree for several d-trees that can be obtained by this sequence of operations that differ because of differing insertions of components above the substituted components in $\gamma_1, \ldots, \gamma_k$.

The **tree set** $T(G)$ generated by G is defined as the set of trees γ such that there is a $\gamma' \in T(\alpha)$ for some elementary d-tree α of G; γ' is rooted with the nonterminal S; the frontier of γ' is a string in V_T^*; and γ results from the removal of all d-edges from γ'. A d-edge is removed by merging the nodes at either end of the edge as long as they are labelled by the same symbol. The **string language** $L(G)$ associated with G is the set of terminal strings appearing on the frontier of trees in $T(G)$.

As indicated earlier, an SA-tree only partially captures the derivation of a d-tree, and does not specify the placement of the inserted components during subsertions. We now describe informally a structure that can be used to encode a DTG derivation. A derivation graph for $\gamma \in T(G)$ results from the addition of insertion edges to a SA-tree τ for γ. The location in γ of an inserted elementary component $\alpha(i)$ can be unambiguously determined by identifying the source of the node (say the node with address n in the elementary d-tree α') with which the root of this occurrence of $\alpha(i)$ is merged with when d-edges are removed. The insertion edge will relate the two (not necessarily distinct) nodes corresponding to appropriate occurrences of α and α' and will be labelled by the pair (i, n).

Earlier, we had indicated that DTG provides a mechanism to control the components that can be appear within d-edges. Each d-edge in elementary d-trees has an associated subsertion-insertion constraint (SIC). A SIC is a finite set of elementary node addresses (ENAs). An ENA η specifies some elementary d-tree $\alpha \in D$, a component of α and the address of a node within that component of α. If an ENA η is in the SIC associated with a d-edge between η_1 and η_2 in an elementary d-tree

[4]The notion of substitutability is used to ensure the SA-tree is a tree. That is, an elementary structure cannot be subserted into more than one structure since this would be counter to our motivations for using subsertion for complementation.

[5]The children corresponding to subsertion can be ordered in any arbitrary but fixed manner.

α then η cannot appear properly within the path that appears from η_1 to η_2 in the derived tree $\gamma \in T(G)$.

Each node at the top of an i-edge of elementary d-trees has an associated sister-adjunction constraint (SAC). A SAC is a finite set of pairs, each pair identifying a direction, $d \in \{\mathsf{left}, \mathsf{right}\}$, and an elementary d-tree, α. A SAC gives a complete specification of what can be sister-adjoined at a node. If a node η is associated with a SAC containing a pair (d, α) then the d-tree α can be d-sister-adjoined at η.

3. Parsing Framework

Our approach to DTG parsing is inspired by a methodology introduced by Lang (Billot and Lang, 1989; Lang, 1991) for CFG. This involves breaking the parsing process into two phases. For example, given a CFG, an equivalent PDA is constructed. The transition moves of this (non-deterministic) PDA represent the primitive steps of the recognition process. Dynamic programming techniques then serve to determinize the recognition process, with different dynamic programming styles yielding different recognition algorithms. This approach is particularly well-suited to formalisms such as TAG and DTG that have an enlarged domain of locality, where individual structures can span noncontiguous substrings. For example, in the case of TAG, the adjoining operation combines two *compound* structures in a single derivation step. Adjunction of two structures does not correspond to a single step in the recognition process but has to be decomposed into several more primitive steps (in recognizers that only consider contiguous substrings). This decomposition corresponds to the conversion from grammar to automaton in Lang's framework.

We adopt a variant of this approach in which the primitive recognition steps are captured with a second grammar formalism rather than an automaton model. In the case of TAG, the TAG-equivalent formalisms of LIG or HG are suitable candidates, since for both formalisms each rewriting step corresponds directly to the basic steps of recognition. Indeed, parsing algorithms for TAG have been developed by adapting the algorithms for LIG or HG (Vijay-Shanker and Weir, 1993a)[6].

The elementary structures of DTG, like TAG, have an enlarged domain of locality. A single subsertion involves one substitution and possibly several insertions. The recognition process corresponding to a single subsertion is comprised of several more primitive steps. As with TAG, it is valuable to use a formalism with a smaller domain of locality. For this reason, we introduce the Linear Prioritized Multiset Grammar (LPMG) formalism. LPMG is similar to LIG differing in its use of multisets in place of stacks.

We provide a translation from DTG to LPMG that, we believe, captures the primitive steps of DTG recognition. Different dynamic programming techniques can then be applied to obtain different recognition algorithms. Specialization of the recognition grammar for a specific input serves to encode the set of all parses for that input. Clearly, one could develop an algorithm for DTG directly and have the translation to LPMG be implicit. However, for this preliminary work on DTG parsing, we believe we will be better served by making the translation explicit so that we can systematically consider different dynamic programming methods to obtain different DTG recognition algorithms. For space reasons, we consider only one Earley-style method in this paper. It would be equally straightforward to develop a CKY-style algorithm from the LPMG grammar.

4. Definition of LPMG

We now define LPMG, which is a variant of {}-LIG (Rambow, 1994b) obtained by providing an additional mechanism for restricting the manipulation the members of the multiset.

[6]The use of grammars rather than automata ties in with Lang's later work (for instance, see Lang (1994)) where grammars can be specialized for a specific input to represent the shared forest of derivation trees for that input. In particular, Vijay-Shanker and Weir (1993b), show that for TAG an equivalent HG or LIG can be used in the grammar specialization process, rather than the object (TAG) grammar.

A multiset over the finite alphabet V is a function $m : V \to \mathsf{N}$ where N is the natural numbers. The set of all multisets over V is denoted N^V. For two multisets m_1 and m_2 over V, multiset addition and subtraction are defined as follows. For all $v \in V$:

$$(m_1 \oplus m_2)(v) = m_1(v) + m_2(v) \qquad (m_1 \ominus m_2)(v) = max\,(m_1(v) - m_2(v), 0)$$

When context permits we will abuse this notation and not distinguish between a symbol and the multiset that contains only that symbol. Thus if m is a multiset over V and $v \in V$ then

$$(m \oplus v)(v') = \begin{cases} m(v') & \text{if } v \neq v' \\ m(v) + 1 & \text{otherwise} \end{cases}$$

$$(m \ominus v)(v') = \begin{cases} m(v') & \text{if } v \neq v' \\ max\,(m(v) - 1, 0) & \text{otherwise} \end{cases}$$

An ordering on multisets is defined as follows.

$$m_1 \sqsubseteq m_2 \text{ iff } m_1(v) \leq m_2(v) \text{ for all } v \in V$$

The empty multiset is denoted ϕ. The number of elements in the multiset m is denoted $size\,(m)$.

A LPMG is a six tuple $G = (V_N, V_T, V_M, S, \sigma_0, P, \rho)$ where V_N, V_T and V_M are the nonterminal, terminal and multiset alphabets, respectively; $S \in V_N$ and σ_0 are the initial nonterminal and multiset symbols, respectively; P is a finite set of productions (see below); and $\rho \subseteq V_M \times V_M$ is the priority relation.

The objects appearing on the left of productions come from the set $V_T \cup \{\,\epsilon\,\} \cup \mathsf{P}(V_N, V_M) \cup \mathsf{P}^{\cdot\cdot}(V_N, V_M)$ where

$$\mathsf{P}(V_N, V_M) = \{\, A[\sigma_1, \ldots, \sigma_k] \mid A \in V_N,\, k \geq 1 \text{ and } \sigma \in V_M \,\}$$

$$\mathsf{P}^{\cdot\cdot}(V_N, V_M) = \{\, A[\cdot\cdot\, \sigma_1, \ldots, \sigma_k\, \cdot\cdot] \mid A \in V_N,\, k \geq 0 \text{ and } \sigma_1, \ldots, \sigma_k \in V_M \,\}$$

Productions in LPMG have one of the following forms.

1. $A[\cdot\cdot\, \sigma\, \cdot\cdot] \to X_1 \ldots X_k$ where $k \geq 1$ and $X_i \in \mathsf{P}(V_N, V_M) \cup \mathsf{P}^{\cdot\cdot}(V_N, V_M)$ for i $(1 \leq i \leq k)$
2. $A[\sigma] \to x$ where $A \in V_N$ and $x \in V_T \cup \{\,\epsilon\,\}$.

Given a priority relation ρ we define ρ-selectable as follows: σ is ρ-selectable from a multiset m iff $m(\sigma) \geq 1$ and for all σ' such that $\langle \sigma', \sigma \rangle \in \rho$ we have $m(\sigma') = 0$.

In derivations sentential forms are strings of objects taken from $V_T \cup \mathsf{M}(V_N, V_M)$ where

$$\mathsf{M}(V_N, V_M) = \{\, A[m] \mid A \in V_N \text{ and } m \in \mathsf{N}^{V_M} \,\}$$

For all $\Upsilon_1, \Upsilon_2 \in (\mathsf{M}(V_N, V_M) \cup V_T)^*$:

if $\quad A[\sigma] \to x \in P$

then $\quad \Upsilon_1 A[\sigma] \Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 x \Upsilon_2$;

if $\quad A[\cdot\cdot\, \sigma\, \cdot\cdot] \to X_1 \ldots X_k \in P$,
σ is ρ-selectable from m,
$m \ominus \sigma = m'_1 \oplus \ldots \oplus m'_k$,
$X_i = A_i[\sigma_{i,1}, \ldots, \sigma_{i,n_i}]$ or $X_i = A_i[\cdot\cdot\, \sigma_{i,1}, \ldots, \sigma_{i,n_i}\, \cdot\cdot]$ for each i $(1 \leq i \leq k)$,
$m_i = m'_i \oplus \sigma_{i,1} \oplus \ldots \oplus \sigma_{i,n_i}$ for each i $(1 \leq i \leq k)$ and
$m'_i = \phi$ if $X_i = A_i[\sigma_{i,1}, \ldots, \sigma_{i,n_i}]$ for each i $(1 \leq i \leq k)$

then $\quad \Upsilon_1 A[m] \Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 A_1[m_1] \ldots A_k[m_k] \Upsilon_2$.

In this latter derivation step, we are attempting to rewrite $A[m]$ by using a production with $A[\cdot\cdot\ \sigma\ \cdot\cdot]$ in its left-hand-side. Of course, this is possible only when $\sigma \in m$ and the contents of m and the priority relation allows the rewriting. Note both of these constraints are verified by the definition of ρ-selectivity. The application of the rule will remove the σ from m and the remaining multiset elements are distributed amongst the k elements in the right-hand-side of the rule. The multisets inherited by the i^{th} element is indicated as m'_i. Thus, $m'_1 \oplus \cdots \oplus m'_k$ must be equal to $m \ominus \sigma$. If the i^{th} element is $A_i[\sigma_{i_1}, \ldots, \sigma_{i,k_i}]$ then derivation from it should only have the multiset $\phi \oplus \sigma_{i_1} \oplus \ldots \oplus \sigma_{i,k_i}$. That is, it does not inherit any multiset elements. This is indicated above by stating that $m'_i = \phi$ in this case. On the other hand, if the i^{th} element is $A_i[\cdot\cdot\ \sigma_{i_1}, \ldots, \sigma_{i,k_i}\ \cdot\cdot]$ then derivation from it should not only have the multiset elements $\sigma_{i_1}, \ldots, \sigma_{i,k_i}$ associated with it, but also m'_i, that part of the multiset m that is inherited by the X_i.

The language generated by $G = (V_N, V_T, V_M, S, \sigma_0, P, \rho)$ is defined as

$$L(G) = \left\{ w \in V_T^* \;\middle|\; S[\sigma_0] \underset{G}{\overset{*}{\Longrightarrow}} w \right\}$$

where $\underset{G}{\overset{*}{\Longrightarrow}}$ is the reflexive, transitive closure of $\underset{G}{\Longrightarrow}$.

5. DTG to LPMG Conversion

We now show how a DTG can be converted into an equivalent LPMG. The construction described here is similar to the TAG to LIG conversion described by Vijay-Shanker and Weir (1994) that has been used in the development of TAG parsing algorithms (Vijay-Shanker and Weir, 1993a). Adjunction has the effect of embedding one elementary tree within another and the LIG's stack is used to control the potentially unbounded nesting of elementary trees that occur in TAG derivations. Following the UVG-DL to {}-LIG conversion described by Rambow (1994a), the DTG to LPMG conversion described below is similar to the TAG to LIG conversion except that multisets rather than stacks are used to control the embedding of d-trees. This is because there is a certain degree of freedom associated with the positioning of components inserted during a subsertion. In particular, there is only limited control over the relative positioning of the inserted components of two subserted d-trees. As a result, a multiset (whose elements are unordered) rather than a stack is used to encode the embedding contexts in DTG derivations.

Embedding context are multisets of ENAs (elementary node addresses) and at the start of a derivation this multiset contains only the root of some elementary d-tree that is labelled by the initial nonterminal. Embedding of d-trees occurs only at nodes at the top of d-edges and when this happens the multiset stores the ENA of the node at the bottom of the d-edge. When the node at the bottom of the d-edge is reached in the derivation, this ENA will be removed from the multiset and the remainder of its elementary d-tree can be traversed. Hence, we say that *open* d-edges are represented by elements of the multiset (corresponding to nodes at the bottom of the d-edge) and that the removal of these elements from the multiset corresponds to *closing* the corresponding d-edge.

LPMG nonterminals are used to encode the current ENA and the productions for each ENA are determined by the context of ENA in its elementary d-tree. That is, the productions depend on whether the node is at the top of a d-edge, top of an i-edge, or a substitution node. Productions correspond to inserting or not inserting within a d-edge, substituting a component, or sister-adjoining a d-tree. When we apply a production corresponding to the insertion of some ENA we must check that the ENA does not appear in the SIC that is associated with some open d-edge. As every open d-edge is represented in the multiset by the ENA of the node at the bottom of the d-edge, SICs can be checked as follows. First we define the priority relation so that whenever the multiset contains an ENA at the bottom of some open edge it is not possible to select from that multiset an ENA that is in that d-edge's SIC. Second, not only is the current ENA encoded by the nonterminal symbol but we also store it in the multiset. Whenever a production for some ENA is applied we also specify that the corresponding ENA must be removed from the multiset. Thus, it is not possible to use

a production for an ENA whose positioning at that point in the derivation violates a SIC. This explains the apparent redundancy in productions where an ENA is encoded both in the multiset and in the nonterminal.

One way to understand the LPMG productions obtained in the conversion is to view derivations top-down. The multiset will record the next node in an elementary d-tree to be visited in a top-down traversal of the derived d-tree. While LPMG nonterminals encode the nodes being visited, their productions insist that their ENAs are present in the multiset to ensure that they can be visited in the derivation.

Before giving details of the conversion, we note that for explanatory purposes we have slightly simplified the initial presentation of the construction. Two complications are discussed at the end of the section. Given a DTG $G = (V_N, V_T, S, D)$ we construct an equivalent LPMG $G' = (V'_N, V_T, V_M, S, \sigma_0, P, \prec)$. Let V_E be the set of ENAs of trees in D whose members will be denoted η with or without subscripts and primes. Let $V'_N = V_E \cup \{ S' \}$ and $V_M = V_E \cup \{ \sigma_0 \}$. ρ is defined such that if η_1 is the d-edge daughter of some elementary node and η_2 is in the SIC associated with this d-edge then the pair $\langle \eta_1, \eta_2 \rangle$ is included in ρ. P is defined as follows.

Case 1: As the root of a derived tree can correspond to the root of any elementary d-tree that is labelled by S, for each η_r labelled by S that is the root of some d-tree in D include the following in P.

$$S[\cdot\cdot\, \sigma_0\, \cdot\cdot] \rightarrow \eta_r[\cdot\cdot\, \eta_r\, \cdot\cdot]$$

Case 2: For each terminal node η that is labelled x include the following in P.

$$\eta[\eta] \rightarrow x$$

This production ensures that when a terminal node is visited the only ENA in the multiset that can be present at this point must be that of the terminal node.

Case 3: For each η and η_r such that η_r is the root of an elementary d-tree that (according to the SAC at η) can be left-sister-adjoined at η include the following in P.

$$\eta[\cdot\cdot\, \eta\, \cdot\cdot] \rightarrow \eta_r[\eta_r]\eta[\cdot\cdot\, \eta\, \cdot\cdot]$$

For each η and η_r such that η_r is the root of an elementary d-tree that (according to the SAC at η) can be right-sister-adjoined at η include the following in P.

$$\eta[\cdot\cdot\, \eta\, \cdot\cdot] \rightarrow \eta[\cdot\cdot\, \eta\, \cdot\cdot]\eta_r[\eta_r]$$

As sister-adjunction is being considered, we ensure that the multiset associated with η is not distributed to the element in the right-hand-side of the rule that corresponds to the d-tree being sister-adjoined.

Case 4: Suppose that η is a node in some d-tree in D with i-edge daughters $\eta_1, \ldots, \eta_k$ where $k \geq 1$.

Include the following productions in P.

$$\eta[\cdot\cdot\, \eta\, \cdot\cdot] \rightarrow \eta_1[\cdot\cdot\, \eta_1\, \cdot\cdot] \ldots \eta_k[\cdot\cdot\, \eta_k\, \cdot\cdot]$$

Here the multiset is to be distributed (in any manner) amongst the children, indicating that the open edges can be closed in the subtree below any of them. Viewing the derivations bottom-up this indicates that the components that are to appear above the individual children nodes must appear above the parent (connected by i-edges).

Case 5: Suppose that η_t is a node in some d-tree in D with d-edge daughter η_b.

If η_t and η_b are labelled by the same symbol then include the following production in P.

$$\eta_t[\cdot\cdot\ \eta_t\ \cdot\cdot] \rightarrow \eta_b[\cdot\cdot\ \eta_b\ \cdot\cdot]$$

For each η that is labelled with the same symbol as η_t and is the root of some elementary d-tree in D include the following in P.

$$\eta_t[\cdot\cdot\ \eta_t\ \cdot\cdot] \rightarrow \eta[\cdot\cdot\ \eta, \eta_b\ \cdot\cdot]$$

For each η that is labelled with the same symbol as η_t, is the root of some elementary component but is not the root of a d-tree include the following in P.

$$\eta_t[\cdot\cdot\ \eta_t\ \cdot\cdot] \rightarrow \eta[\cdot\cdot\ \eta_b\ \cdot\cdot]$$

The first production corresponds to the case where a component is not inserted within this d-edge. The latter two productions consider insertions at this d-edge. Note η_b is added to the multiset at this point indicating that it will be the next node in its elementary d-tree that is to be visited. When the component (with root η) that is to be inserted at this point is not the topmost component of its elementary d-tree (see the third production) then η must be found in the multiset and should not be added. this constitutes a guess that η is in the multiset, but note that it will only be possible to apply a production for the nonterminal η if this is the case. On the other hand, when the component is the topmost component in its d-tree (see the second production) the multiset at this point in the LPMG derivation will not record this instance of this tree (as this is where we are considering the embedding of the d-tree for the first time), hence η is added to the multiset.

Case 6: Suppose that η is a substitution node in some d-tree in D.

For each η_r that is labelled with the same symbol as η and is the root of an elementary d-tree in D include the following in P.

$$\eta[\cdot\cdot\ \eta\ \cdot\cdot] \rightarrow \eta_r[\cdot\cdot\ \eta_r\ \cdot\cdot]$$

For each η_r that is labelled with the same symbol as η, is the root of an elementary component but not the root of a d-tree in D include the following in P.

$$\eta[\cdot\cdot\ \eta\ \cdot\cdot] \rightarrow \eta_r[\cdot\cdot]$$

Any component (whether the topmost of an elementary d-tree or not) can be substituted at a substitution node provided their labels match. As in Case 5, we need to consider whether the component is the topmost component of its elementary d-tree. Thus, in second production the multiset is assumed to include an instance of η_r.

As mentioned, the above construction has been oversimplified slightly. In order to incorporate the substitutability conditions described in the definition of DTG derivations we must check that each elementary d-tree is involved in only one subsertion. This can be done by using two forms of multiset symbols for each ENA: one that is used for nodes in d-trees above the node that will be substituted; and the other for nodes below the substituted node. The substitutability constraint can be enforced by allowing substitution only with nodes encoded by the first form of ENA and by changing from the first to the second form of ENA at this point.

A second complication concerns the enforcement of SICs. The definition states that an ENA that is in a SIC at a d-edge between η_t and η_b cannot be place *properly* within the path from η_t to η_b. Since the root node η of a component that is inserted immediately below η_t will be merged with η_t, it should not be possible for the SIC at the d-edge from η_t to η_b to block the insertion of η

and, in addition, it should not be possible for the SIC at the d-edge immediately above η (when η is not at the root of a d-tree) to block the positioning of η_t. However, in the above construction, η_b will be in the multiset at the point that the production for η is applied and η could be in the multiset when the production for η_t is applied. The construction must be altered so that we do not put η_b into the multiset until after we have applied the production for η. This can be done by temporarily recording η_b in the nonterminal. Similarly, in the above construction, it is possible that the selection of η_t may be blocked by the presence of η in the multiset. Thus, we must remove η from the multiset before attempting to applying the production for η_t. Again, this can be done with extra nonterminals that temporarily record the identity of η.

6. Earley-style DTG Parsing

Let the components of the LPMG G, be $G = (V_N, V_T, V_M, S, \sigma_0, P, \rho)$ and the input string be $a_1...a_n$. The algorithm completes the $n+1$ item sets I_0, ..., I_n. We assume that the DTG from which G has been constructed is lexicalized.

Items in item sets have the form $((\omega_0 \rightarrow \omega_1 \cdot \omega_2, m_1), (m_2, i))$ where $\omega_0 \rightarrow \omega_1\omega_2$ is a LPMG production, i is the index of the item set that introduced the production; and m_1 and m_2 are multisets. In a LPMG production such as $A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow A_1[\cdot\cdot\ \sigma_1\ \cdot\cdot] \ldots A_k[\cdot\cdot\ \sigma_k\ \cdot\cdot]$ there is no restriction on how the multiset associated with the nonterminal on the left is distributed among the nonterminals on the right. Rather than considering all possible distributions of the multiset in the predictor step, we pass the the entire multiset to the first subtree and propagate the multiset through the remaining subtrees in a top-down, left-to-right traversal of the derived tree. The underlying idea is that in an item $((\omega_0 \rightarrow \omega_1 \cdot \omega_2, m_1), (m_2, i)) \in I_j$ the multiset m_2 is the multiset at the time that we introduce the dotted rule $\omega_0 \rightarrow \cdot\omega_1\omega_2$ into item list I_i, and m_1 is the current value of the multiset that is the remainder to be passed onto subtrees not yet considered (corresponding to parts to the right of the dot in this dotted rule).

Initialization:

if $\quad S[\cdot\cdot\ \sigma_0\ \cdot\cdot] \rightarrow \omega \in P$

then $\quad ((S[\cdot\cdot\ \sigma_0\ \cdot\cdot] \rightarrow \cdot\omega, \phi), (\sigma_0, 0)) \in I_0$

Note, initially the the multiset contains just σ_0. As the use of this rule will cause it to be removed, the empty multiset will be the current multiset that is passed to the descendant.

Prediction:

1. if $\quad ((\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma_1, \ldots, \sigma_k\ \cdot\cdot]\omega_2, m_1), (m_2, j)) \in I_i,$
   $A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \omega \in P,$
   σ is ρ-selectable from $m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ and
   $size\,(m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma) \leq n+1$

   then $\quad ((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \cdot\omega, m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma), (m_1, i)) \in I_i.$

   As indicated above, m_1 is the current value of the multiset, whereas the multiset to be passed onto the first descendant is obtained by adding $\sigma_1, \ldots, \sigma_k$ to m_1 and removing (one occurrence of) σ from the resulting set. Note that the condition that σ is ρ-selectable from $m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ ensures that σ is in $m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$. Due to the assumption that the underlying DTG is lexicalized we limit the application of this predictive step to situations where the number of elements in $m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma$ does not exceed $n+1$ (for an explanation of this see Section 7).

2. if $((\omega_0 \rightarrow \omega_1 \cdot A[\sigma_1, \ldots, \sigma_k]\omega_2, m_1), (m_2, j)) \in I_i$,
$A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \omega \in P$,
σ is ρ-selectable from $\phi \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ and
$k \leq n+1$

then $((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \cdot\omega, \phi \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma), (m_1, i)) \in I_i$.

In this case we do not feed the entire multiset m_1 into the new production since the new production is "called" with a multiset containing just $\sigma_1, \ldots, \sigma_k$.

Scanning:

1. if $((\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma\ \cdot\cdot]\omega_2, m_1), (m_2, j)) \in I_i$,
$A[\sigma] \rightarrow a \in P$ and
$a = a_{i+1}$

then $((\omega_0 \rightarrow \omega_1 A[\cdot\cdot\ \sigma\ \cdot\cdot] \cdot \omega_2, m_1), (m_2, j)) \in I_{i+1}$.

2. if $((\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma\ \cdot\cdot]\omega_2, m_1), (m_2, j)) \in I_i$ and
$A[\sigma] \rightarrow \epsilon \in P$

then $((\omega_0 \rightarrow \omega_1 A[\cdot\cdot\ \sigma\ \cdot\cdot] \cdot \omega_2, m_1), (m_2, j)) \in I_i$.

3. if $((\omega_0 \rightarrow \omega_1 \cdot A[\sigma]\omega_2, m_1), (m_2, j)) \in I_i$,
$A[\sigma] \rightarrow a \in P$ and
$a = a_{i+1}$

then $((\omega_0 \rightarrow \omega_1 A[\sigma] \cdot \omega_2, m_1), (m_2, j)) \in I_{i+1}$.

4. if $((\omega_0 \rightarrow \omega_1 \cdot A[\sigma]\omega_2, m_1), (m_2, j)) \in I_i$ and
$A[\sigma] \rightarrow \epsilon \in P$

then $((\omega_0 \rightarrow \omega_1 A[\sigma] \cdot \omega_2, m_1), (m_2, j)) \in I_i$.

Completer:

1. if $((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \omega\cdot, m_1), (m_2, j)) \in I_i$,
$m_1 \sqsubseteq m_2$,
$((\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma_1, \ldots, \sigma_k\ \cdot\cdot]\omega_2, m_2), (m_3, l)) \in I_j$,
σ is ρ-selectable from $m_2 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ and
$size\,(m_2 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma) \leq n+1$

then $((\omega_0 \rightarrow \omega_1 A[\cdot\cdot\ \sigma_1, \ldots, \sigma_k\ \cdot\cdot] \cdot \omega_2, m_1), (m_3, l)) \in I_i$.

Note that $((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \omega\cdot, m_1), (m_2, j)) \in I_i$ means that before we considered the use of the rule the multiset was m_2. Part of this has been used up in the derivation from A and the remainder is m_1. This means we need to verify that $m_1 \sqsubseteq m_2$. Furthermore, because the use of this rule must have been predicted earlier (when we were considering I_j) then we must expect the presence of an item in I_j containing a dotted rule of the form $\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma_1, \ldots, \sigma_k\ \cdot\cdot]\omega_2$. In particular, the completed item expects that the current multiset in that item must be m_2.

2. if $((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \omega\cdot, \phi), (m_1, j)) \in I_i$,
$((\omega_0 \rightarrow \omega_1 \cdot A[\sigma_1, \ldots, \sigma_k]\omega_2, m_1), (m_2, l)) \in I_j$,
σ is ρ-selectable from $\phi \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ and
$k \leq n+1$

then $((\omega_0 \rightarrow \omega_1 A[\sigma_1, \ldots, \sigma_k] \cdot \omega_2, m_1), (m_2, l)) \in I_i$.

In this case we are moving the dot over a nonterminal that is associated with a fixed multiset. Thus, the multiset associated with the completed production must be empty and the multiset

m_1 remains intact since it was not fed into the completed production (see case 2 of the predictor step).

The input is accepted if $((S[\cdot\cdot\ \sigma_0\ \cdot\cdot] \rightarrow \omega\cdot, \phi), (\sigma_0, 0)) \in I_n$.

A LPMG parse forest can be extracted from the completed item sets in the usual way. Since it is possible to recover the derivation graphs, and therefore SA-trees, of the underlying DTG from the corresponding derivation tree of the constructed LPMG grammar the LPMG parse forest provides a reasonable encoding of the set of DTG derivations for the input string. In one respect, the LPMG parse forest is particularly compact since there is a one-to-many mapping from LPMG derivation trees to DTG derivation graphs. When reconstructing a DTG derivation graph from a LPMG derivation tree it is necessary to establish which occurrences of ENA's (occurring in the multisets at nodes of the LPMG derivation tree) should be associated with the same occurrence of a d-tree in the DTG derivation. Because no distinction is made between different occurrences of the same multiset symbol in multiset there may be several ways of associating occurrences of a multiset symbols at different nodes in the derivation tree. Thus, it is possible that for a given LPMG derivation tree there will be several ways of making the correspondence of occurrences of multiset symbols to occurrences of elementary d-trees. This is attractive because for every possible way of making the correspondence there will be a legal DTG derivation. Thus, a single LPMG derivation tree is compactly encoding a set of DTG derivations. The issue of parse forests will be discussed at greater length in the full paper.

7. Complexity Analysis

We now discuss the time complexity of the algorithm. We assume that the DTG from which the LPMG was constructed was lexicalized. We will only be interested in the dependence on the length of the input string. The item sets contain tuples of the form $((\omega_0 \rightarrow \omega_1 \cdot \omega_2, m_1), (m_2, i))$ where $\omega_0 \rightarrow \omega_1\omega_2$ is a production, m_1 and m_2 are multisets, and $0 \leq i \leq n$ where n is the length of the input. Clearly, the number of items in a item set depends crucially on the number of possible multisets. Since we assume that the underlying DTG grammar is lexicalized the number of open d-edges at any node in the derived tree is bounded by the length of the input string. Although the derived LPMG is not lexicalized, the construction presented in Section 5 is such that the size of any multiset used in a derivation by a LPMG thus constructed is bounded by $n + 1$. The number of such multisets is bounded by $O(n^k)$, where is k is the total number of d-edges in the elementary d-trees of the grammar. Thus, the number of items in a item set is bounded by $O(n^{2k+1})$.

The completer step dominates the running time of the algorithm since for each of the $O(n^{2k+1})$ items in I_i we consider $O(n^{2k+1})$ items in I_j. Thus the running time of this step is $O(n^{4k+2})$. Since there are $n + 1$ item sets the total worst-case running time of the algorithm is $O(n^{4k+3})$. Note that the running time of the recognizer is sensitive to the number of open d-edges that arise in derivations and that in some applications (such as with grammars of English) this number may be very small (perhaps as low as 1 or 2).

8. Conclusion

We have presented an Earley-style parser for DTG. In future work, we intend to make more explicit the derivation of both the SA-tree and the full DTG derivation structure from the LPMG derivation. We also intend to follow the work of Lang (1994) and investigate how the LPMG parse forest can be obtained from the original grammar by grammar specialization.

References

Tilman Becker and Owen Rambow. 1994. Parsing free word order languages. In *Proceedings of the Third International Workshop on Tree Adjoining Grammars*, pages 12–15.

Tilman Becker and Owen Rambow. 1995. Parsing non-immediate dominance relations. submitted to IWPT-95.

Tilman Becker, Aravind Joshi, and Owen Rambow. 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.

Sylvie Billot and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *27th Meeting of the Association for Computational Linguistics (ACL'89)*.

Joan Bresnan and Ronald Kaplan. 1982. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press.

Robert Frank. 1992. *Syntactic Locality and Tree Adjoining Grammar: Grammatical, Acquisition and Processing Perspectives*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

Beryl Hoffman. 1995. Integrating "free" word order syntax and information structure. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics (EACL'95)*, pages 245–252.

Aravind Joshi and Yves Schabes. 1991. Tree-adjoining grammars and lexicalized grammars. In Maurice Nivat and Andreas Podelski, editors, *Definability and Recognizability of Sets of Trees*. Elsevier.

Aravind Joshi, Leon Levy, and M. Takahashi. 1975. Tree adjunct grammars. *J. Comput. Syst. Sci.*, 10(1):136–163.

Anthony Kroch. 1987. Subjacency in a tree adjoining grammar. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 143–172. John Benjamins, Amsterdam.

Bernard Lang. 1991. A uniform framework for parsing. In M. Tomita, editor, *Current Issues in Parsing Technology*, pages 153–171. Klumer Academic Publisher.

Bernard Lang. 1994. Recognition can be harder than parsing. *Computational Intelligence*, 10(4):486–494.

Owen Rambow and Aravind Joshi. 1992. A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. In *International Workshop on The Meaning-Text Theory*, Darmstadt. Arbeitspapiere der GMD 671.

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

Owen Rambow. 1994a. *Formal and Computational Aspects of Natural Language Syntax*. Ph.D. 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).

Owen Rambow. 1994b. Multiset-valued linear index grammars. In *32nd Meeting of the Association for Computational Linguistics (ACL'94)*, pages 263–270.

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

Stuart Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

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

K. Vijay-Shanker and David Weir. 1993b. The use of shared forests in TAG parsing. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics (EACL'93)*, pages 384–393, Utrecht.

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

K. Vijay-Shanker. 1987. *A Study of Tree Adjoining Grammars*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.

K. Vijay-Shanker. 1992. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18(4):481–517, December.