PARSING D-TREE GRAMMARS

K. Vijay-Shanker
Computer & Information Science
University of Delaware
vijay@udel.edu

David Weir
Cognitive & Computing Sciences
University of Sussex
david.weir@cogs.susx.ac.uk

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

1 Motivation

D-Tree Grammars (DTG) (Rambow et al., 1995) arise from work on Lexicalized Tree-Adjoining Grammars (LTAG) (Joshi & Schabes, 1991). A salient feature of LTAG is the extended domain of locality it provides. Each elementary structure can be associated with a lexical item and properties related to the lexical item (such as subcategorization) can be expressed within the elementary structure. In addition, LTAG 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). LTAG, however, has two limitations: the LTAG operations of substitution and adjunction do not map cleanly onto the relations of complementation and modification; and they cannot provide analyses for certain syntactic phenomena. We briefly discuss the first issue here, and refer to Rambow et al. (1995) for a broader discussion of both issues.

In LTAG, the operations of substitution and adjunction relate two lexical items, establishing either the linguistic relation of complementation (predicate-argument relation) or modification between them. These relations represent important linguistic intuition, they provide a uniform interface to semantics, and they are, as Schabes & Shieber (1994) argue, important in order to support statistical parameters in stochastic frameworks and appropriate adjunction constraints in LTAG.

However, the LTAG 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 LTAG: 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) LTAG 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.

In defining DTG we have attempted to resolve this problem 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 LTAG (in particular, its enlarged domain of locality)[1].

2 Definition of DTG

A **d-tree** is a tree with two types of edges: domination edges (**d-edges**) and immediate domination edges (**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

[1] Related formalisms can be found in Becker et al. (1991) and Rambow (1994a).

all components other than the topmost one connected to a node in a component above by a d-edge. In the d-trees α and β shown in Figure 1, these components are shown as triangles.

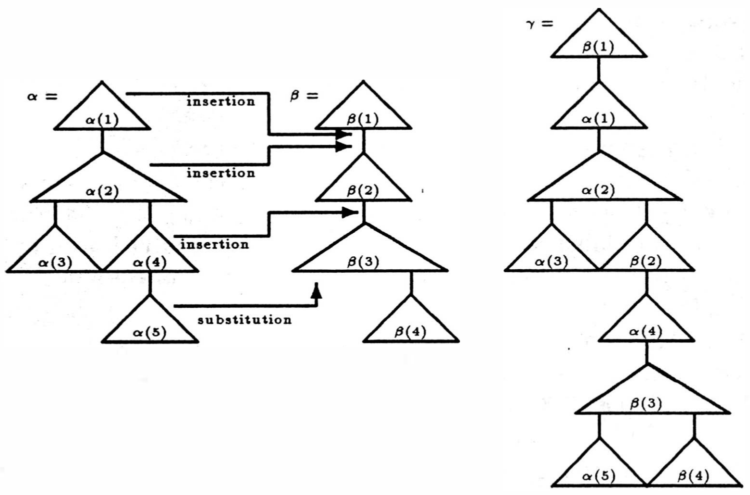

Figure 1: Subsertion

Subsertion is an operation on d-trees that is a generalization of tree substitution. 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 1 illustrates a possible subsertion of α in β with the component $\alpha(5)$ being substituted at a substitution node in β. In the composed d-tree, γ, the components, $\alpha(1)$, $\alpha(2)$, and $\alpha(4)$ of α above $\alpha(5)$ are inserted in the path in β between the root and the substitution node. In general, some of these components could have been placed above the root of β as well. When a component $\alpha(i)$ of some d-tree α is inserted into a d-edge between nodes η_1 and η_2 in β 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. 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).

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. We first discuss DTG derivations and subsertion-adjoining trees (SA-trees), which are partial derivation structures that represent dependency information, the importance of which was stressed in the introduction. 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.

Consider a DTG $G = (V_N, V_T, S, D)$. 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. Any elementary d-tree is also considered a derived d-tree. The components of the elementary d-tree are considered the **substitutable**[2] components of this tree considered as a derived d-tree. Derivations involve sister-adjoining or subsertions. If α is subserted in β, only substitutable components of α may be substituted (at a substitution node in β). In the resulting d-tree, say γ, the components of γ that came from β are now the substitutable components of γ and are ordered (or numbered) in the same manner as in β.

The **tree set** $T(G)$ generated by G is defined as the set of trees γ such that there is a derived d-tree, γ' rooted with the nonterminal S, whose frontier 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 in derivation trees of a LTAG, the nodes in SA-trees are labeled by the names of the elementary d-trees. The edges depict the composition operations used. In the case of sister-adjoining, an edge labelled (d, m) between nodes labelled by two elementary d-trees, say from α to β, represents sister adjoining of β at node in α addressed m. $d \in \{\text{left}, \text{right}\}$ indicates whether left- or right-sister-adjunction took place. For subsertion, an edge labelled (l, m) encodes the substitution of the l^{th} component of β at the substitution node addressed m in α.

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. 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).

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 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, $d \in \{\text{left}, \text{right}\}$, 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. Further details of the motivation and definition of DTG, example d-trees and derivations, and usage of SIC and SAC may be found in Rambow et al. (1995).

3 Parsing Framework

Our approach to DTG parsing is inspired by a methodology introduced by Lang (Billot & Lang, 1989; Lang, 1991) for CFG. This involves dividing 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. This approach is particularly well-suited to formalisms such as LTAG and DTG that have an enlarged domain of locality, where individual structures can span noncontiguous substrings. 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

[2] 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.

substrings. This decomposition corresponds to the conversion from grammar to automaton in Lang's framework.

We adopt a variant of this approach using a second grammar formalism rather than an automaton model. In the case of LTAG, the LTAG-equivalent formalisms of LIG or HG are suitable candidates, since unlike LTAG, in both formalisms, each rewriting step corresponds directly to the basic steps of recognition. Indeed, parsing algorithms for LTAG have been developed by adapting the algorithms for LIG or HG (Vijay-Shanker & Weir, 1993a)[3].

The elementary structures of DTG, like LTAG, 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 LTAG, 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 captures the primitive steps of DTG recognition. Specialization of the recognition grammar (the LPMG obtained) for an 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, we make 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 of the members of the multiset. A multiset over a finite alphabet V is a function $m : V \rightarrow \mathsf{N}$ 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 such that for all $v \in V$: $(m_1 \oplus m_2)(v) = m_1(v) + m_2(v)$ and $(m_1 \ominus m_2)(v) = max\,(m_1(v) - m_2(v), 0)$. When context permits we will abuse this notation and not distinguish between a symbol and the multiset that contains only that symbol. An ordering on multisets is defined as: $m_1 \sqsubseteq m_2$ iff $m_1(v) \leq m_2(v)$ 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 $\sigma_0 \in V_M$ are the initial nonterminal and multiset symbols, respectively; $\rho \subseteq V_M \times V_M$ is the priority relation and P is a finite set of productions having the form $A[\sigma] \rightarrow x$ where $A \in V_N$ and $x \in V_T \cup \{ \epsilon \}$; or $A[\cdot\cdot\, \sigma \,\cdot\cdot] \rightarrow X_1 \ldots X_k$ where $k \geq 1$ and for $1 \leq i \leq k$, $X_i \in \{ A[\sigma_1, \ldots, \sigma_k] \mid A \in V_N,\ k \geq 1 \text{ and } \sigma_1, \ldots, \sigma_k \in V_M \} \cup \{ A[\cdot\cdot\, \sigma_1, \ldots, \sigma_k \,\cdot\cdot] \mid A \in V_N,\ k \geq 0 \text{ and } \sigma_1, \ldots, \sigma_k \in V_M \}$

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 $\Upsilon_1, \Upsilon_2 \in (\mathsf{M}(V_N, V_M) \cup V_T)^*$: (1) if $A[\sigma] \rightarrow x \in P$ then $\Upsilon_1 A[\sigma] \Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 x \Upsilon_2$; and (2) if $A[\cdot\cdot\, \sigma \,\cdot\cdot] \rightarrow X_1 \ldots X_k \in P$, 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 $\Upsilon_1 A[m] \Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 A_1[m_1] \ldots A_k[m_k] \Upsilon_2$.

In this latter derivation step, we rewrite $A[m]$ by using a production with $A[\cdot\cdot\, \sigma \,\cdot\cdot]$ in its left-hand-side. This is possible only when $\sigma \in m$ and the contents of m and the priority relation

[3]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 & Weir (1993b), show that for LTAG an equivalent HG or LIG can be used in the grammar specialization process, rather than the object (LTAG) grammar.

allows the rewriting which is 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 the derivation from it should only have the multiset $m_i = \phi \oplus \sigma_{i_1} \oplus \ldots \oplus \sigma_{i,k_i}$. 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 the 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 $L(G)$ generated by G is the set of terminal strings derived from $S[\sigma_0]$.

5 DTG to LPMG Conversion

The construction described here is similar to the LTAG to LIG conversion described by Vijay-Shanker & Weir (1994) that has been used in the development of LTAG parsing algorithms (Vijay-Shanker & Weir, 1993a). Adjunction has the effect of embedding one elementary tree within another and the LIGs stack is used to control the unbounded nesting of elementary trees that occur in LTAG derivations. Following the UVG-DL to {}-LIG conversion described by Rambow (1994a), the DTG to LPMG conversion described below is similar to the LTAG to LIG conversion except that multisets rather than stacks are used to control the embedding of d-trees. This is because there is limited control over the relative positioning of the inserted components of two subserted d-trees.

Embedding context are multisets of ENAs (elementary node addresses). We consider embedding of d-trees only at nodes at the top of d-edges. When this happens the multiset stores the ENA at the bottom of the d-edge and when this node is reached in the derivation, this ENA is removed from the multiset. Thus, *open* d-edges are represented by elements of the multiset (corresponding to nodes at the bottom of the d-edge) and removal of these elements from the multiset corresponds to *closing* the corresponding d-edge. The multiset can be seen as a record of the elementary nodes that are still to be visited in a top-down traversal of the derived d-tree.

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. 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.

From a DTG $G = (V_N, V_T, S, D)$ we construct a 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 let $S[\cdot\cdot\, \sigma_0\, \cdot\cdot] \rightarrow \eta_r[\cdot\cdot\, \eta_r\, \cdot\cdot] \in P$.

Case 2: For each terminal node η that is labelled x let $\eta[\eta] \to x \in P$. 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 let $\eta[\cdot\cdot\ \eta\ \cdot\cdot] \to \eta_r[\eta_r]\eta[\cdot\cdot\ \eta\ \cdot\cdot] \in P$. 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 let $\eta[\cdot\cdot\ \eta\ \cdot\cdot] \to \eta[\cdot\cdot\ \eta\ \cdot\cdot]\eta_r[\eta_r] \in P$. 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: For each node η in some d-tree in D with i-edge daughters $\eta_1, \ldots, \eta_k$, where $k \geq 1$, let $\eta[\cdot\cdot\ \eta\ \cdot\cdot] \to \eta_1[\cdot\cdot\ \eta_1\ \cdot\cdot] \ldots \eta_k[\cdot\cdot\ \eta_k\ \cdot\cdot] \in P$. 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.

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 let $\eta_t[\cdot\cdot\ \eta_t\ \cdot\cdot] \to \eta_b[\cdot\cdot\ \eta_b\ \cdot\cdot] \in P$.

For each η that is labelled with the same symbol as η_t and is the root of some elementary d-tree in D let $\eta_t[\cdot\cdot\ \eta_t\ \cdot\cdot] \to \eta[\cdot\cdot\ \eta, \eta_b\ \cdot\cdot] \in P$.

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 let $\eta_t[\cdot\cdot\ \eta_t\ \cdot\cdot] \to \eta[\cdot\cdot\ \eta_b\ \cdot\cdot] \in P$.

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) being inserted is not the topmost component of its elementary d-tree (third production) then η must be found in the multiset and should not be added. Note that it will only be possible to apply a production for the nonterminal η if η is indeed in the multiset. On the other hand, when the component is the topmost component in its d-tree (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 also 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 let $\eta[\cdot\cdot\ \eta\ \cdot\cdot] \to \eta_r[\cdot\cdot\ \eta_r\ \cdot\cdot] \in P$.

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 let $\eta[\cdot\cdot\ \eta\ \cdot\cdot] \to \eta_r[\cdot] \in P$. 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.

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 fact that the definition of SICs 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. With the use of extra nonterminals it is straightforward to capture this definition, but due to space restrictions we are unable to give details.

6 Earley-style DTG Parsing

We assume that the DTG from which LPMG $G = (V_N, V_T, V_M, S, \sigma_0, P, \rho)$ is constructed is lexicalized. Let the input string be $a_1 ... a_n$. The algorithm completes the $n+1$ item sets I_0, ..., I_n. 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 predicting use of a LPMG production such as $A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow A_1[\cdot\cdot\ \sigma_1\ \cdot\cdot] \ldots A_k[\cdot\cdot\ \sigma_k\ \cdot\cdot]$, rather than considering all possible distributions of the multiset among the nonterminals on the right, we pass 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). The input is accepted if $((S[\cdot\cdot\ \sigma_0\ \cdot\cdot] \rightarrow \omega\cdot, \phi), (\sigma_0, 0)) \in I_n$.

Initialization: $((S[\cdot\cdot\ \sigma_0\ \cdot\cdot] \rightarrow \cdot\omega, \phi), (\sigma_0, 0)) \in I_0$
if $S[\cdot\cdot\ \sigma_0\ \cdot\cdot] \rightarrow \omega \in P$. Note, initially 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 (a): $((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \cdot\omega, m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma), (m_1, i)) \in I_i$
if $((\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma_1, \ldots, \sigma_k\ \cdot\cdot]\omega_2, m_1), (m_2, j)) \in I_i$, $A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \omega \in P$, σ is ρ-selectable from $m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ and $size(m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma) \leq n+1$. 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 $size(m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma) \leq n+1$.

Prediction (b): $((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \cdot\omega, \phi \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma), (m_1, i)) \in I_i$
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 $k - 1 \leq n+1$. 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: $((\omega_0 \rightarrow \omega_1 A[\cdot\cdot\ \sigma\ \cdot\cdot] \cdot \omega_2, m_1), (m_2, j)) \in I_{i'}$ (resp. $((\omega_0 \rightarrow \omega_1 A[\sigma] \cdot \omega_2, m_1), (m_2, j)) \in I_{i'}$)
if $((\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma\ \cdot\cdot]\omega_2, m_1), (m_2, j)) \in I_i$ (resp. $((\omega_0 \rightarrow \omega_1 \cdot A[\sigma]\omega_2, m_1), (m_2, j)) \in I_i$), $A[\sigma] \rightarrow x \in P$ and $i = i'$ when $x = \epsilon$ and $i' = i+1$ when $x = a_{i+1}$.

Completer (a): $((\omega_0 \rightarrow \omega_1 A[\cdot\cdot\ \sigma_1, \ldots, \sigma_k\ \cdot\cdot] \cdot \omega_2, m_1), (m_3, l)) \in I_i$
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 $size(m_2 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma) \leq n+1$. Before we considered the use of the production 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.

Completer (b): $((\omega_0 \rightarrow \omega_1 A[\sigma_1, \ldots, \sigma_k] \cdot \omega_2, m_1), (m_2, l)) \in I_i$
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 $k \leq n+1$. The dot is moved 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 the second case of the predictor step).

A LPMG parse forest can be extracted from the completed item sets in the usual way. Since there is a direct correspondence between a group of LPMG productions and DTG composition operations, 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. Thus, 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 ENAs (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 a 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 item sets contain tuples of the form $((\omega_0 \rightarrow \omega_1 \cdot \omega_2, m_1), (m_2, i))$ where $0 \leq i \leq n$ and n is input length. The number of items in a item set depends 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. 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).

References

T. Becker & O. Rambow. 1994. Parsing free word order languages. In *3rd TAG Workshop*.

T. Becker & O. Rambow. 1995. Parsing non-immediate dominance relations. IWPT'95.

T. Becker, A. Joshi, & O. Rambow. 1991. Long distance scrambling and Tree Adjoining Grammars. In EACL'91.

S. Billot & B. Lang. 1989. The structure of shared forests in ambiguous parsing. In ACL'89.

A. Joshi & Y. Schabes. 1991. Tree-Adjoining Grammars and lexicalized grammars. In M. Nivat & A. Podelski, eds., *Definability and Recognizability of Sets of Trees.*

B. Lang. 1991. A uniform framework for parsing. In M. Tomita, ed., *Current Issues in Parsing Technology.*

B. Lang. 1994. Recognition can be harder than parsing. *Computat. Intelligence.*, 10(4).

O. Rambow. 1994a. *Formal and Computational Aspects of Natural Language Syntax.* Ph.D. thesis, Dept. CIS, Univ. Penn.

O. Rambow, K. Vijay-Shanker, & D. Weir. 1995. D-Tree Grammars. In ACL'95.

O. Rambow. 1994b. Multiset-valued Linear Index Grammars. In ACL'94.

Y. Schabes & S. Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computat. Ling.*, 20(1).

S. Shieber. 1985. Evidence against the context-freeness of natural language. *Ling. & Philos.*, 8.

K. Vijay-Shanker & D. Weir. 1993a. Parsing some constrained grammar formalisms. *Computat. Ling.*, 19(4).

K. Vijay-Shanker & D. Weir. 1993b. The use of shared forests in TAG parsing. In EACL'93.

K. Vijay-Shanker & D. Weir. 1994. The equivalence of four extensions of Context-Free Grammars. *Math. Syst. Theory*, 27.