Appears in: Proc. 32nd Annual Meeting of the ACL, 1994

Capturing CFLs with Tree Adjoining Grammars

James Rogers*
Dept. of Computer and Information Sciences
University of Delaware
Newark, DE 19716, USA
jrogers@cis.udel.edu

Abstract

We define a decidable class of TAGs that is strongly equivalent to CFGs and is cubic-time parsable. This class serves to lexicalize CFGs in the same manner as the LCFGs of Schabes and Waters but with considerably less restriction on the form of the grammars. The class provides a normal form for TAGs that generate local sets in much the same way that regular grammars provide a normal form for CFGs that generate regular sets.

Introduction

We introduce the notion of *Regular Form* for Tree Adjoining Grammars (TAGs). The class of TAGs that are in regular from is equivalent in strong generative capacity[1] to the Context-Free Grammars, that is, the sets of trees generated by TAGs in this class are the *local sets*—the sets of derivation trees generated by CFGs.[2] Our investigations were initially motivated by the work of Schabes, Joshi, and Waters in lexicalization of CFGs via TAGs (Schabes and Joshi, 1991; Joshi and Schabes, 1992; Schabes and Waters, 1993a; Schabes and Waters, 1993b; Schabes, 1990). The class we describe not only serves to lexicalize CFGs in a way that is more faithful and more flexible in its encoding than earlier work, but provides a basis for using the more expressive TAG formalism to define Context-Free Languages (CFLs.)

In Schabes et al. (1988) and Schabes (1990) a general notion of *lexicalized grammars* is introduced. A grammar is lexicalized in this sense if each of the basic structures it manipulates is associated with a lexical item, its *anchor*. The set of structures relevant to a particular input string, then, is selected by the lexical items that occur in that string. There are a number of reasons for exploring lexicalized grammars. Chief among these are linguistic considerations—lexicalized grammars reflect the tendency in many current syntactic theories to have the details of the syntactic structure be projected from the lexicon. There are also practical advantages. All lexicalized grammars are finitely ambiguous and, consequently, recognition for them is decidable. Further, lexicalization supports strategies that can, in practice, improve the speed of recognition algorithms (Schabes et al., 1988).

One grammar formalism is said to *lexicalize* another (Joshi and Schabes, 1992) if for every grammar in the second formalism there is a lexicalized grammar in the first that generates exactly the same set of structures. While CFGs are attractive for efficiency of recognition, Joshi and Schabes (1992) have shown that an arbitrary CFG cannot, in general, be converted into a strongly equivalent lexicalized CFG. Instead, they show how CFGs can be lexicalized by LTAGS (Lexicalized TAGs). While the LTAG that lexicalizes a given CFG must be strongly equivalent to that CFG, both the languages and sets of trees generated by LTAGs as a class are strict supersets of the CFLs and local sets. Thus, while this gives a means of constructing a lexicalized grammar from an existing CFG, it does not provide a direct method for constructing lexicalized grammars that are known to be equivalent to (unspecified) CFGs. Furthermore, the best known recognition algorithm for LTAGs runs in $O(n^6)$ time.

Schabes and Waters (1993a; 1993b) define Lexicalized Context-Free Grammars (LCFGs), a class of lexicalized TAGs (with restricted adjunction) that not only lexicalizes CFGs, but is cubic-time parsable and is *weakly* equivalent to CFGs. These LCFGs have a couple of shortcomings. First, they are not strongly equivalent to CFGs. Since they are cubic-time parsable this is primarily a theoretical rather than practical concern. More importantly, they employ structures of a highly restricted form. Thus the restrictions of the formalism, in some cases, may override linguistic considerations in constructing the grammar. Clearly any class of TAGs that are cubic-time parsable, or that are equivalent in

*The work reported here owes a great deal to extensive discussions with K. Vijay-Shanker.

[1]We will refer to equivalence of the sets of trees generated by two grammars or classes of grammars as *strong equivalence*. Equivalence of their string languages will be referred to as *weak equivalence*.

[2]Technically, the sets of trees generated by TAGs in the class are *recognizable sets*. The local and recognizable sets are equivalent modulo projection. We discuss the distinction in the next section.

Appears in: Proc. 32nd Annual Meeting of the ACL, 1994

any sense to CFGs, must be restricted in some way. The question is what restrictions are necessary.

In this paper we directly address the issue of identifying a class of TAGs that are strongly equivalent to CFGs. In doing so we define such a class—TAGs in *regular form*—that is decidable, cubic-time parsable, and lexicalizes CFGs. Further, regular form is essentially a closure condition on the elementary trees of the TAG. Rather than restricting the form of the trees that can be employed, or the mechanisms by which they are combined, it requires that whenever a tree with a particular form can be derived then certain other related trees must be derivable as well. The algorithm for deciding whether a given grammar is in regular form can produce a set of elementary trees that will extend a grammar that does not meet the condition to one that does.[3] Thus the grammar can be written largely on the basis of the linguistic structures that it is intended to capture. We show that, while the LCFGs that are built by Schabes and Waters's algorithm for lexicalization of CFGs are in regular form, the restrictions they employ are unnecessarily strong.

Regular form provides a partial answer to the more general issue of characterizing the TAGs that generate local sets. It serves as a normal form for these TAGs in the same way that regular grammars serve as a normal form for CFGs that generate regular languages. While for every TAG that generates a local set there is a TAG in regular form that generates the same set, and every TAG in regular form generates a local set (modulo projection), there are TAGs that are not in regular form that generate local sets, just as there are CFGs that generate regular languages that are not regular grammars.

The next section of this paper briefly introduces notation for TAGs and the concept of recognizable sets. Our results on regular form are developed in the subsequent section. We first define a restricted use of the adjunction operation—derivation by *regular adjunction*—which we show derives only recognizable sets. We then define the class of TAGs in regular form and show that the set of trees derivable in a TAG of this form is derivable by regular adjunction in that TAG and is therefore recognizable. We next show that every local set can be generated by a TAG in regular form and that Schabes and Waters's construction for LCFGs in fact produces TAGs in regular form. Finally, we provide an algorithm for deciding if a given TAG is in regular form. We close with a discussion of the implications of this work with respect to the lexicalization of CFGs and the use of TAGs to define languages that are strictly context-free, and raise the question of whether our results can be strengthened for some classes of TAGs.

[3] Although the result of this process is not, in general, equivalent to the original grammar.

Preliminaries

Tree Adjoining Grammars

Formally, a TAG is a five-tuple $\langle \Sigma, \mathrm{NT}, I, A, S\rangle$ where:

Σ	is a finite set of *terminal symbols*,
NT	is a finite set of *non-terminal symbols*,
I	is a finite set of *elementary initial trees*,
A	is a finite set of *elementary auxiliary trees*,
S	is a distinguished non-terminal, the *start symbol*.

Every non-frontier node of a tree in $I \cup A$ is labeled with a non-terminal. Frontier nodes may be labeled with either a terminal or a non-terminal. Every tree in A has exactly one frontier node that is designated as its *foot*. This must be labeled with the same non-terminal as the root. The auxiliary and initial trees are distinguished by the presence (or absence, respectively) of a foot node. Every other frontier node that is labeled with a non-terminal is considered to be *marked for substitution.* In a lexicalized TAG (LTAG) every tree in $I \cup A$ must have some frontier node designated the *anchor*, which must be labeled with a terminal.

Unless otherwise stated, we include both elementary and derived trees when referring to initial trees and auxiliary trees. A TAG derives trees by a sequence of substitutions and adjunctions in the elementary trees. In *substitution* an instance of an *initial tree* in which the root is labeled $X \in \mathrm{NT}$ is substituted for a frontier node (other than the foot) in an instance of either an initial or auxiliary tree that is also labeled X. Both trees may be either an elementary tree or a derived tree.

In *adjunction* an instance of an *auxiliary tree* in which the root and foot are labeled X is inserted at a node, also labeled X, in an instance of either an initial or auxiliary tree as follows: the subtree at that node is excised, the auxiliary tree is substituted at that node, and the excised subtree is substituted at the foot of the auxiliary tree. Again, the trees may be either elementary or derived.

The set of objects ultimately derived by a TAG G is $T(G)$, the set of *completed* initial trees derivable in G. These are the initial trees derivable in G in which the root is labeled S and every frontier node is labeled with a terminal (thus no nodes are marked for substitution.) We refer to the set of all trees, both initial and auxiliary, with or without nodes marked for substitution, that are derivable in G as $T'(G)$. The *language* derived by G is $L(G)$ the set of strings in Σ^* that are the yields of trees in $T(G)$.

In this paper, all TAGs are *pure* TAGs, i.e., without adjoining constraints. Most of our results go through for TAGs with adjoining constraints as well, but there is much more to say about these TAGs and the implications of this work in distinguishing the pure TAGs from TAGs in general. This is a part of our ongoing research.

The path between the root and foot (inclusive) of an auxiliary tree is referred to as its *spine*. Auxiliary trees

Appears in: Proc. 32nd Annual Meeting of the ACL, 1994

in which no node on the spine other than the foot is labeled with the same non-terminal as the root we call a *proper* auxiliary tree.

Lemma 1 *For any TAG G there is a TAG G' that includes no improper elementary trees such that $T(G)$ is a projection of $T(G')$.*

Proof (Sketch)**:** The grammar G can be relabeled with symbols in $\{\langle x, i\rangle \mid x \in \Sigma \cup \mathrm{NT}, i \in \{0,1\}\}$ to form G'. Every auxiliary tree is duplicated, with the root and foot labeled $\langle X, 0\rangle$ in one copy and $\langle X, 1\rangle$ in the other. Improper elementary auxiliary trees can be avoided by appropriate choice of labels along the spine. □

The labels in the trees generated by G' are a refinement of the labels of the trees generated by G. Thus G' partitions the categories assigned by G into subcategories on the basis of (a fixed amount of) context. While the use here is technical rather than natural, the approach is familiar, as in the use of slashed categories to handle movement.

Recognizable Sets

The local sets are formally very closely related to the recognizable sets, which are somewhat more convenient to work with. These are sets of trees that are accepted by *finite-state tree automata* (Gécseg and Steinby, 1984). If Σ is a finite alphabet, a *Σ-valued tree* is a finite, rooted, left-to-right ordered tree, the nodes of which are labeled with symbols in Σ. We will denote such a tree in which the root is labeled σ and in which the subtrees at the children of the root are $t_1, \ldots, t_n$ as $\sigma(t_1, \ldots, t_n)$. The set of all Σ-valued trees is denoted T_Σ.

A *(non-deterministic) bottom-up finite state tree automaton* over Σ-valued trees is a tuple $\langle \Sigma, Q, M, F\rangle$ where:

Σ	is a finite *alphabet*,
Q	is a finite set of *states*,
F	is a subset of Q, the set of *final states*, and
M	is a partial function from $\Sigma \times Q^*$ to $\mathcal{P}(Q)$ (the powerset of Q) with finite domain, the *transition function*.

The transition function M associates sets of states with alphabet symbols. It induces a function that associates sets of states with trees, $\overline{M} : T_\Sigma \to \mathcal{P}(Q)$, such that:

$$
\begin{aligned}
& q \in \overline{M}(t) \overset{\text{def}}{\iff} \\
& \quad t \text{ is a leaf labeled } \sigma \text{ and } q \in M(\sigma, \varepsilon), \text{ or} \\
& \quad t = \sigma(t_0, \ldots, t_n) \text{ and there is a sequence} \\
& \qquad \text{of states } q_0, \ldots, q_n \text{ such that } q_i \in \overline{M}(t_i), \\
& \qquad \text{for } 0 \le i \le n, \text{ and } q \in M(\sigma, q_0 \cdot \cdots \cdot q_n).
\end{aligned}
$$

An automaton $\mathcal{A} = \langle \Sigma, Q, M, F\rangle$ accepts a tree $t \in T_\Sigma$ iff, by definition, $F \cap \overline{M}(t)$ is not empty. The set of trees accepted by an automaton $\mathcal{A}$ is denoted $T(\mathcal{A})$.

A set of trees is *recognizable* iff, by definition, it is $T(\mathcal{A})$ for some automaton $\mathcal{A}$.

Lemma 2 *(Thatcher, 1967) Every local set is recognizable. Every recognizable set is the projection of some local set.*

The projection is necessary because the automaton can distinguish between nodes labeled with the same symbol while the CFG cannot. The set of trees (with bounded branching) in which exactly one node is labeled A, for instance, is recognizable but not local. It is, however, the projection of a local set in which the labels of the nodes that dominate the node labeled A are distinguished from the labels of those that don't.

As a corollary of this lemma, the *path set* of a recognizable (or local) set, i.e., the set of strings that label paths in the trees in that set, is regular.

TAGs in Regular Form

Regular Adjunction

The fact that the path sets of recognizable sets must be regular provides our basic approach to defining a class of TAGs that generate only recognizable sets. We start with a restricted form of adjunction that can generate only regular path sets and then look for a class of TAGs that do not generate any trees that cannot be generated with this restricted form of adjunction.

Definition 1 Regular adjunction *is ordinary adjunction restricted to the following cases:*

- *any auxiliary tree may be adjoined into any initial tree or at any node that is not on the spine of an auxiliary tree,*
- *any* proper *auxiliary tree may be adjoined into any auxiliary tree at the root or foot of that tree,*
- *any auxiliary tree γ_1 may be adjoined at any node along the spine of any auxiliary tree γ_2 provided that no instance of γ_2 can be adjoined at any node along the spine of γ_1.*

In figure 1, for example, this rules out adjunction of β_1 into the spine of β_3, or *vice versa*, either directly or indirectly (by adjunction of β_3, say, into β_2 and then adjunction of the resulting auxiliary tree into β_1.) Note that, in the case of TAGs with no improper elementary auxiliary trees, the requirement that only proper auxiliary trees may be adjoined at the root or foot is not actually a restriction. This is because the only way to derive an improper auxiliary tree in such a TAG without violating the other restrictions on regular adjunction is by adjunction at the root or foot. Any sequence of such adjunctions can always be re-ordered in a way which meets the requirement.

We denote the set of completed initial trees derivable by regular adjunction in G as $T_R(G)$. Similarly, we denote the set of all trees that are derivable by regular adjunction in G as $T'_R(G)$. As intended, we can show that $T_R(G)$ is always a recognizable set. We are looking, then, for a class of TAGs for which $T(G) = T_R(G)$ for every G in the class. Clearly, this will be the case if $T'(G) = T'_R(G)$ for every such G.

Appears in: Proc. 32nd Annual Meeting of the ACL, 1994

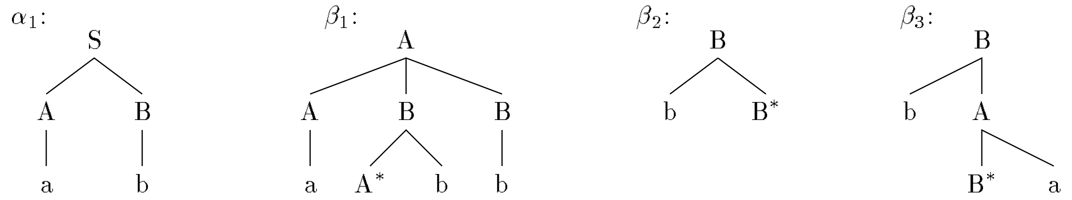

Figure 1: Regular Adjunction

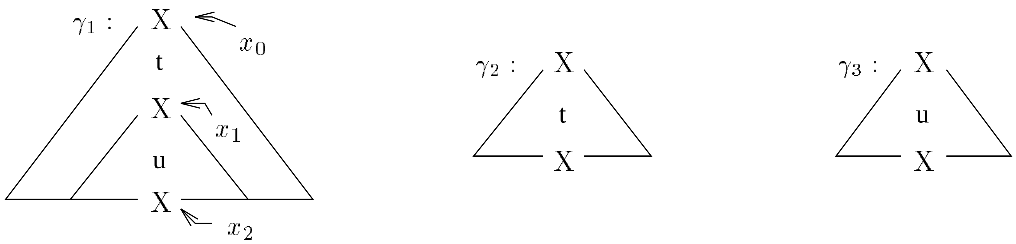

Figure 2: Regular Form

Proposition 1 *If G is a TAG and $T'(G) = T'_R(G)$. Then $T(G)$ is a recognizable set.*

Proof (Sketch)**:** This follows from the fact that in regular adjunction, if one treats adjunction at the root or foot as substitution, there is a fixed bound, dependent only on G, on the depth to which auxiliary trees can be nested. Thus the nesting of the auxiliary trees can be tracked by a fixed depth stack. Such a stack can be encoded in a finite set of states. It's reasonably easy to see, then, how G can be compiled into a bottom-up finite state tree automaton. □

Since regular adjunction generates only recognizable sets, and thus (modulo projection) local sets, and since CFGs can be parsed in cubic time, one would hope that TAGs that employ only regular adjunction can be parsed in cubic time as well. In fact, such is the case.

Proposition 2 *If G is a TAG for which $T(G) = T_R(G)$ then there is a algorithm that recognizes strings in $L(G)$ in time proportional to the cube of the length of the string.*[4]

Proof (Sketch)**:** This, again, follows from the fact that the depth of nesting of auxiliary trees is bounded in regular adjunction. A CKY-style style parsing algorithm for TAGs (the one given in Vijay-Shanker and Weir (1993), for example) can be modified to work with a two-dimensional array, storing in each slot $[i, j]$ a set of structures that encode a node in an elementary tree that can occur at the root of a subtree spanning the input from position i through j in some tree derivable in G, along with a stack recording the nesting of elementary auxiliary trees around that node in the derivation of that tree. Since the stacks are bounded the amount of data stored in each node is independent of the input length and the algorithm executes in time proportional to the cube of the length of the input. □

Regular Form

We are interested in classes of TAGs for which $T'(G) = T'_R(G)$. One such class is the TAGs in *regular form.*

Definition 2 *A TAG is in* **regular form** *iff whenever a* completed *auxiliary tree of the form γ_1 in Figure 2 is derivable, where $x_0 \neq x_1 \neq x_2$ and no node labeled* X *occurs properly between x_0 and x_1, then trees of the form γ_2 and γ_3 are derivable as well.*

Effectively, this is a closure condition on the elementary trees of the grammar. Note that it immediately implies that every improper elementary auxiliary tree in a regular form TAG is redundant. It is also easy to see, by induction on the number of occurrences of X along the spine, that any auxiliary tree γ for X that is derivable in G can be decomposed into the concatenation of a sequence of proper auxiliary trees for X each of which is derivable in G. We will refer to the proper auxiliary trees in this sequence as the *proper segments* of γ.

Lemma 3 *Suppose G is a TAG in regular form. Then $T'(G) = T'_R(G)$*

Proof: Suppose γ is any non-elementary auxiliary tree derivable by unrestricted adjunction in G and that any smaller tree derivable in G is derivable by regular adjunction in G. If γ is proper, then it is clearly derivable from two strictly smaller trees by regular adjunction, each of which, by the induction hypothesis, is in $T'_R(G)$. If γ is improper, then it has the form of γ_1 in Figure 2 and it is derivable by regular adjunction of γ_2 at the root of γ_3. Since both of these are derivable and strictly

[4] This result was suggested by K. Vijay-Shanker.

Appears in: Proc. 32nd Annual Meeting of the ACL, 1994

smaller than γ they are in $T'_R(G)$. It follows that γ is in $T'_R(G)$ as well. $\square$

Lemma 4 *Suppose G is a TAG with no improper elementary trees and $T'(G) = T'_R(G)$. Then G is in regular form.*

Proof: Suppose some γ with the form of γ_1 in Figure 2 is derivable in G and that for all trees γ' that are smaller than γ every proper segment of γ' is derivable in G. By assumption γ is not elementary since it is improper. Thus, by hypothesis, γ is derivable by regular adjunction of some γ'' into some γ' both of which are derivable in G.

Suppose γ'' adjoins into the spine of γ' and that a node labeled X occurs along the spine of γ''. Then, by the definition of regular adjunction, γ'' must be adjoined at either the root or foot of γ'. Thus both γ' and γ'' consist of sequences of consecutive proper segments of γ with γ'' including t and the initial (possibly empty) portion of u and γ' including the remainder of u or vice versa. In either case, by the induction hypothesis, every proper segment of both γ' and γ'', and thus every proper segment of γ is derivable in G. Then trees of the form γ_2 and γ_3 are derivable from these proper segments.

Suppose, on the other hand, that γ'' does not adjoin along the spine of γ' or that no node labeled X occurs along the spine of γ''. Note that γ'' must occur entirely within a proper segment of γ. Then γ' is a tree with the form of γ_1 that is smaller than γ. From the induction hypothesis every proper segment of γ' is derivable in G. It follows then that every proper segment of γ is derivable in G, either because it is a proper segment of γ' or because it is derivable by adjunction of γ'' into a proper segment of γ'. Again, trees of the form γ_2 and γ_3 are derivable from these proper segments. $\square$

Regular Form and Local Sets

The class of TAGs in regular form is related to the local sets in much the same way that the class of regular grammars is related to regular languages. Every TAG in regular form generates a recognizable set. This follows from Lemma 3 and Proposition 1. Thus, modulo projection, every TAG in regular form generates a local set. Conversely, the next proposition establishes that every local set can be generated by a TAG in regular form. Thus regular form provides a normal form for TAGs that generate local sets. It is not the case, however, that all TAGs that generate local sets are in regular form.

Proposition 3 *For every CFG G there is a TAG G' in regular form such that the set of derivation trees for G is exactly $T(G')$.*

Proof: This is nearly immediate, since every CFG is equivalent to a Tree Substitution Grammar (in which all trees are of depth one) and every Tree Substitution Grammar is, in the definition we use here, a TAG with no elementary auxiliary trees. It follows that this TAG can derive no auxiliary trees at all, and is thus vacuously in regular form. $\square$

This proof is hardly satisfying, depending as it does on the fact that TAGs, as we define them, can employ substitution. The next proposition yields, as a corollary, the more substantial result that every CFG is strongly equivalent to a TAG in regular form in which substitution plays no role.

Proposition 4 *The class of TAGs in regular form can lexicalize CFGs.*

Proof: This follows directly from the equivalent lemma in Schabes and Waters (1993a). The construction given there builds a *left-corner derivation graph* (LCG). Vertices in this graph are the terminals and nonterminals of G. Edges correspond to the productions of G in the following way: there is an edge from X to Y labeled $X \rightarrow Y\alpha$ iff $X \rightarrow Y\alpha$ is a production in G. Paths through this graph that end on a terminal characterize the left-corner derivations in G. The construction proceeds by building a set of elementary initial trees corresponding to the simple (acyclic) paths through the LCG that end on terminals. These capture the non-recursive left-corner derivations in G. The set of auxiliary trees is built in two steps. First, an auxiliary tree is constructed for every simple cycle in the graph. This gives a set of auxiliary trees that is sufficient, with the initial trees, to derive every tree generated by the CFG. This set of auxiliary trees, however, may include some which are not lexicalized, that is, in which every frontier node other than the foot is marked for substitution. These can be lexicalized by substituting every corresponding elementary initial tree at one of those frontier nodes. Call the LCFG constructed for G by this method G'. For our purposes, the important point of the construction is that every simple cycle in the LCG is represented by an elementary auxiliary tree. Since the spines of auxiliary trees derivable in G' correspond to cycles in the LCG, every proper segment of an auxiliary tree derivable in G' is a simple cycle in the LCG. Thus every such proper segment is derivable in G' and G' is in regular form. $\square$

The use of a graph which captures left-corner derivations as the foundation of this construction guarantees that the auxiliary trees it builds will be left-recursive (will have the foot as the left-most leaf.) It is a requirement of LCFGs that all auxiliary trees be either left- or right-recursive. Thus, while other derivation strategies may be employed in constructing the graph, these must always expand either the left- or right-most child at each step. All that is required for the construction to produce a TAG in regular form, though, is that every simple cycle in the graph be realized in an elementary tree. The resulting grammar will be in regular form no

Appears in: Proc. 32nd Annual Meeting of the ACL, 1994

matter what (complete) derivation strategy is captured in the graph. In particular, this admits the possibility of generating an LTAG in which the anchor of each elementary tree is some linguistically motivated "head".

Corollary 1 *For every CFG G there is a TAG G' in regular form* in which no node is marked for substitution*, such that the set of derivation trees for G is exactly $T(G')$.*

This follows from the fact that the step used to lexicalize the elementary auxiliary trees in Schabes and Waters's construction can be applied to every node (in both initial and auxiliary trees) which is marked for substitution. Paradoxically, to establish the corollary it is not necessary for every elementary tree to be lexicalized. In Schabes and Waters's lemma G is required to be finitely ambiguous and to not generate the empty string. These restrictions are only necessary if G' is to be lexicalized. Here we can accept TAGs which include elementary trees in which the only leaf is the foot node or which yield only the empty string. Thus the corollary applies to all CFGs without restriction.

Regular Form is Decidable

We have established that regular form gives a class of TAGs that is strongly equivalent to CFGs (modulo projection), and that LTAGs in this class lexicalize CFGs. In this section we provide an effective procedure for deciding if a given TAG is in regular form. The procedure is based on a graph that is not unlike the LCG of the construction of Schabes and Waters.

If G is a TAG, the *Spine Graph* of G is a directed multi-graph on a set of vertices, one for each non-terminal in G. If β_i is an elementary auxiliary tree in G and the spine of β_i is labeled with the sequence of non-terminals $\langle X_0, X_1, \ldots, X_n\rangle$ (where $X_0 = X_n$ and the remaining X_j are not necessarily distinct), then there is an edge in the graph from each X_j to X_{j+1} labeled $\langle \beta_i, j, t_{i,j}\rangle$, where $t_{i,j}$ is that portion of β_i that is dominated by X_j but not properly dominated by X_{j+1}. There are no other edges in the graph except those corresponding to the elementary auxiliary trees of G in this way.

The intent is for the spine graph of G to characterize the set of auxiliary trees derivable in G by adjunction along the spine. Clearly, any vertex that is labeled with a non-terminal for which there is no corresponding auxiliary tree plays no active role in these derivations and can be replaced, along with the pairs of edges incident on it, by single edges. Without loss of generality, then, we assume spine graphs of this reduced form. Thus every vertex has at least one edge labeled with a 0 in its second component incident from it.

A *well-formed-cycle* (wfc) in this graph is a (non-empty) path traced by the following non-deterministic automaton:

- The automaton consists of a single push-down stack. Stack contents are labels of edges in the graph.
- The automaton starts on any vertex of the graph with an empty stack.
- At each step, the automaton can move as follows:
  - If there is an edge incident from the current vertex labeled $\langle \beta_i, 0, t_{i,0}\rangle$ the automaton can push that label onto the stack and move to the vertex at the far end of that edge.
  - If the top of stack contains $\langle \beta_i, j, t_{i,j}\rangle$ and there is an edge incident from the current vertex labeled $\langle \beta_i, j+1, t_{i,j+1}\rangle$ the automaton may pop the top of stack, push $\langle \beta_i, j+1, t_{i,j+1}\rangle$ and move to the vertex at the end of that edge.
  - If the top of stack contains $\langle \beta_i, j, t_{i,j}\rangle$ but there is no edge incident from the current vertex labeled $\langle \beta_i, j+1, t_{i,j+1}\rangle$ then the automaton may pop the top of stack and remain at the same vertex.
- The automaton may halt if its stack is empty.
- A path through the graph is traced by the automaton if it starts at the first vertex in the path and halts at the last vertex in the path visiting each of the vertices in the path in order.

Each wfc in a spine graph corresponds to the auxiliary tree built by concatenating the third components of the labels on the edges in the cycle in order. Then every wfc in the spine graph of G corresponds to an auxiliary tree that is derivable in G by adjunction along the spine only. Conversely, every such auxiliary tree corresponds to some wfc in the spine graph.

A *simple cycle* in the spine graph, by definition, is any minimal cycle in the graph that ignores the labels of the edges but not their direction. Simple cycles correspond to auxiliary trees in the same way that wfcs do. Say that two cycles in the graph are equivalent iff they correspond to the same auxiliary tree. The simple cycles in the spine graph for G correspond to the minimal set of elementary auxiliary trees in any presentation of G that is closed under the regular form condition in the following way.

Lemma 5 *A TAG G is in regular form iff every simple cycle in its spine graph is equivalent to a wfc in that graph.*

Proof:
(If every simple cycle is equivalent to a wfc then G is in regular form.)

Suppose every simple cycle in the spine graph of G is equivalent to a wfc and some tree of the form γ_1 in Figure 2 is derivable in G. Wlog, assume the tree is derivable by adjunction along the spine only. Then there is a wfc in the spine graph of G corresponding to that tree that is of the form $\langle X_0, \ldots, X_k, \ldots, X_n\rangle$ where $X_0 = X_k = X_n$, $0 \neq k \neq n$, and $X_i \neq X_0$ for all $0 < i < k$. Thus $\langle X_0, \ldots, X_k\rangle$ is a simple cycle in the spine graph. Further, $\langle X_k, \ldots, X_n\rangle$ is a sequence of one or more such simple cycles. It follows that both $\langle X_0, \ldots, X_k\rangle$ and $\langle X_k, \ldots, X_n\rangle$ are wfc in the

Appears in: Proc. 32nd Annual Meeting of the ACL, 1994

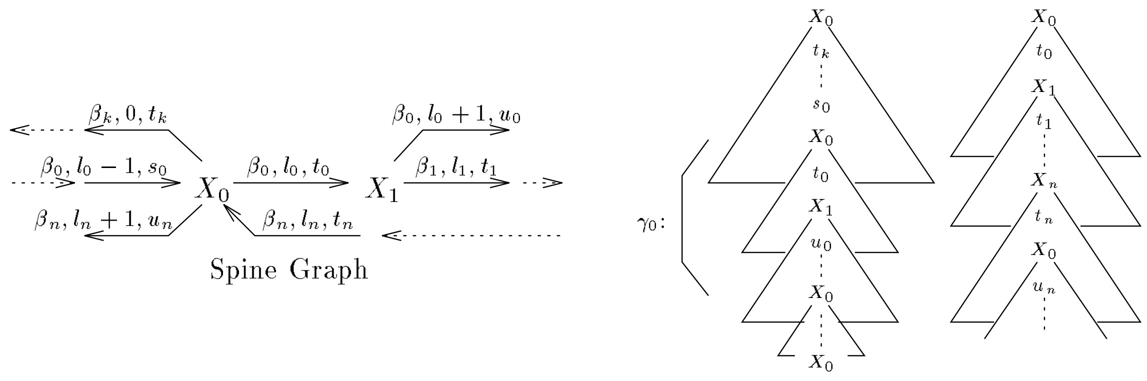

Figure 3: Regular Form is Decidable

spine graph and thus both γ_2 and γ_3 are derivable in G.
(If G is in regular form then every simple cycle corresponds to a wfc.)

Assume, wlog, the spine graph of G is connected. (If it is not we can treat G as a union of grammars.) Since the spine graph is a union of wfcs it has an Eulerian wfc (in the usual sense of Eulerian). Further, since every vertex is the initial vertex of some wfc, every vertex is the initial vertex of some Eulerian wfc.

Suppose there is some simple cycle

$$X_0 \langle \beta_0, l_0, t_0 \rangle \; X_1 \langle \beta_1, l_1, t_1 \rangle \cdots \\ \cdots X_n \langle \beta_n, l_n, t_n \rangle \; X_0$$

where the X_j are the vertices and the tuples are the labels on the edges of the cycle. Then there is a wfc starting at X_0 that includes the edge $\langle \beta_0, l_0, t_0 \rangle$, although not necessarily initially. In particular the Eulerian wfc starting at X_0 is such a wfc. This corresponds to a derivable auxiliary tree that includes a proper segment beginning with t_0. Since G is in regular form, that proper segment is a derivable auxiliary tree. Call this γ_0 (see Figure 3.) The spine of that tree is labeled $X_0, X_1, \ldots, X_0$, where anything (other than X_0) can occur in the ellipses.

The same cycle can be rotated to get a simple cycle starting at each of the X_j. Thus for each X_j there is a derivable auxiliary tree starting with t_j. Call it γ_j. By a sequence of adjunctions of each γ_j at the second node on the spine of γ_{j-1} an auxiliary tree for X_0 is derivable in which the first proper segment is the concatenation of

$$t_0, t_1, \ldots, t_n.$$

Again, by the fact that G is in regular form, this proper segment is derivable in G. Hence there is a wfc in the spine graph corresponding to this tree. □

Proposition 5 *For any TAG G the question of whether G is in regular form is decidable. Further, there is an effective procedure that, given any TAG, will extend it to a TAG that is in regular form.*

Proof: Given a TAG G we construct its spine graph. Since the TAG is finite, the graph is as well. The TAG is in regular form iff every simple cycle is equivalent to a wfc. This is clearly decidable. Further, the set of elementary trees corresponding to simple cycles that are not equivalent to wfcs is effectively constructible. Adding that set to the original TAG extends it to regular form. □
Of course the set of trees generated by the extended TAG may well be a proper superset of the set generated by the original TAG.

Discussion

The LCFGs of Schabes and Waters employ a restricted form of adjunction and a highly restricted form of elementary auxiliary tree. The auxiliary trees of LCFGs can only occur in left- or right-recursive form, that is, with the foot as either the left- or right-most node on the frontier of the tree. Thus the structures that can be captured in these trees are restricted by the mechanism itself, and Schabes and Waters (in (1993a)) cite two situations where an existing LTAG grammar for English (Abeillé et al., 1990) fails to meet this restriction. But while it is sufficient to assure that the language generated is context-free and cubic-time parsable, this restriction is stronger than necessary.

TAGs in regular form, in contrast, are ordinary TAGs utilizing ordinary adjunction. While it is developed from the notion of regular adjunction, regular form is just a closure condition on the elementary trees of the grammar. Although that closure condition assures that all improper elementary auxiliary trees are redundant, the form of the elementary trees themselves is unrestricted. Thus the structures they capture can be driven primarily by linguistic considerations. As we noted earlier, the restrictions on the form of the trees in an LCFG significantly constrain the way in which CFGs can be lexicalized using Schabes and Waters's construction. These constraints are eliminated if we require only that the result be in regular form and the lexicalization can then be structured largely on linguistic principles.

Appears in: Proc. 32nd Annual Meeting of the ACL, 1994

On the other hand, regular form is a property of the grammar as a whole, while the restrictions of LCFG are restrictions on individual trees (and the manner in which they are combined.) Consequently, it is immediately obvious if a grammar meets the requirements of LCFG, while it is less apparent if it is in regular form. In the case of the LTAG grammar for English, neither of the situations noted by Schabes and Waters violate regular form themselves. As regular form is decidable, it is reasonable to ask whether the grammar as a whole is in regular form. A positive result would identify the large fragment of English covered by this grammar as strongly context-free and cubic-time parsable. A negative result is likely to give insight into those structures covered by the grammar that require context-sensitivity.

One might approach defining a context-free language within the TAG formalism by developing a grammar with the intent that all trees derivable in the grammar be derivable by regular adjunction. This condition can then be verified by the algorithm of previous section. In the case that the grammar is not in regular form, the algorithm proposes a set of additional auxiliary trees that will establish that form. In essence, this is a prediction about the strings that would occur in a context-free language extending the language encoded by the original grammar. It is then a linguistic issue whether these additional strings are consistent with the intent of the grammar.

If a grammar is not in regular form, it is not necessarily the case that it does not generate a recognizable set. The main unresolved issue in this work is whether it is possible to characterize the class of TAGs that generate local sets more completely. It is easy to show, for TAGs that employ adjoining constraints, that this is not possible. This is a consequence of the fact that one can construct, for any CFG, a TAG in which the path language is the image, under a bijective homomorphism, of the string language generated by that CFG. Since it is undecidable if an arbitrary CFG generates a regular string language, and since the path language of every recognizable set is regular, it is undecidable if an arbitrary TAG (employing adjoining constraints) generates a recognizable set. This ability to capture CFLs in the string language, however, seems to depend crucially on the nature of the adjoining constraints. It does not appear to extend to pure TAGs, or even TAGs in which the adjoining constraints are implemented as monotonically growing sets of simple features. In the case of TAGs with these limited adjoining constraints, then, the questions of whether there is a class of TAGs which includes all and only those which generate recognizable sets, or if there is an effective procedure for reducing any such TAG which generates a recognizable set to one in regular form, are open.

References

Anne Abeillé, Kathleen M. Bishop, Sharon Cote, and Yves Schabes. 1990. A lexicalized tree adjoining grammar for English. Technical Report MS-CIS-90-24, Department of Computer and Information Science, University of Pennsylvania.

Ferenc Gécseg and Magnus Steinby. 1984. *Tree Automata*. Akadémiai Kiadó, Budapest.

Aravind K. Joshi and Yves Schabes. 1992. Tree-adjoining grammars and lexicalized grammars. In M. Nivat and A. Podelski, editors, *Tree Automata and Languages*, pages 409–431. Elsevier Science Publishers B.V.

Yves Schabes and Aravind K. Joshi. 1991. Parsing with lexicalized tree adjoining grammar. In Masaru Tomita, editor, *Current Issues in Parsing Technology*, chapter 3, pages 25–47. Kluwer Academic Publishers.

Yves Schabes and Richard C. Waters. 1993a. Lexicalized context-free grammars. In *31st Annual Meeting of the Association for Computational Linguistics (ACL'93)*, pages 121–129, Columbus, OH. Association for Computational Linguistics.

Yves Schabes and Richard C. Waters. 1993b. Lexicalized context-free grammar: A cubic-time parsable, lexicalized normal form for context-free grammar that preserves tree structure. Technical Report 93-04, Mitsubishi Electric Research Laboratories Cambridge Research Center, Cambridge, MA, June.

Yves Schabes, Anne Abeillé, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING'88)*, Budapest, Hungary. Association for Computational Linguistics.

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

J. W. Thatcher. 1967. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, 1:317–322.

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