Handling Coordination in a Tree Adjoining Grammar

Anoop Sarkar and Aravind Joshi
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
{anoop,joshi}@linc.cis.upenn.edu

Draft of August 19, 1997
Longer version of (Sarkar and Joshi, 1996)

Abstract

In this paper we show that an account for coordination can be constructed using the derivation structures in a lexicalized Tree Adjoining Grammar (LTAG). We present a notion of derivation in LTAGs that preserves the notion of fixed constituency in the LTAG lexicon while providing the flexibility needed for coordination phenomena. We also discuss the construction of a practical parser for LTAGs that can handle coordination including cases of non-constituent coordination.

1 Introduction

Lexicalized Tree Adjoining Grammars (LTAG) and Combinatory Categorial Grammar (CCG) (Steedman, 1997) are known to be weakly equivalent but not strongly equivalent. Coordination schema have a natural description in CCG, while these schema have no natural equivalent in a standard LTAG.

In (Joshi and Schabes, 1991) it was shown that in principle it is possible to construct a CCG-like account for coordination in the framework of LTAGs, but there was no clear notion of what the derivation structure would look like. In this paper, continuing the work of (Joshi and Schabes, 1991), we show that an account for coordination can be constructed using the derivation structures in an LTAG.

Combinatory Categorial Grammar (CCG) (Steedman, 1985; Steedman, 1997) violates traditional notions of constituency in assigning multiple structures to unambiguous strings. For instance, CCG assigns multiple bracketings to the sentence *Keats steals apples.*

(1) (Keats (steals apples))
(2) ((Keats steals) apples)

Although "spuriously" ambiguous structures are generated by CCG they produce appropriate and fully compositional semantics. The justification for such "spurious" ambiguous structures is that they provide an explanation for a variety of coordination constructions and that they correspond directly to intonation. Coordination in CCG has a natural definition due to this flexible notion of phrase structure. The combinators added to the context-free categorial grammar in CCG can account for a number of otherwise "non-constituent" coordination phenomena. For instance, the bracketing in (2) is necessary for the CCG account for (3)

(3) (((Keats grows) and (Shelley steals)) apples)

In this paper we show how a CCG-like account for coordination can be defined over the elementary trees of a LTAG and present a notion of derivation in LTAGs that preserves the notion of fixed constituency in the LTAG lexicon while providing for the flexibility needed for the various coordination phenomena. In CCG, being a constituent is the same as being a function or semantic type and vice versa. In the (Joshi and Schabes, 1991) treatment and in our present treatment a constituent is always a function or semantic type but the converse is not necessarily true. In our account, the standard notion of constituency in a LTAG is retained; phrase structure is preserved at the level of an elementary tree.

Using the notions given in this paper we also discuss the construction of practical parser for LTAGs that can handle coordination including cases of non-constituent coordination. This approach has been implemented in the XTAG system (XTAG Research Group, 1995) thus extending it to handle coordination. This is the first full implementation of coordination in the LTAG framework.

2 LTAG

An LTAG is a set of trees which have at least one terminal symbol on its frontier called the *anchor* of that tree. For example, Figure 1 shows an example of a tree for a transitive verb *cooked.* Each node in the tree has a unique address obtained by applying a Gorn tree addressing scheme. For instance, the object *NP* has address 2.2. In the LTAG formalism, trees can be composed using the two operations of *substitution* (corresponds to string concatenation) and *adjunction* (corresponds to string wrapping). A history of these operations on elementary trees in the form of a derivation tree can be used to reconstruct the derivation of a string recognized by a LTAG. Figure 2 shows an example of a derivation tree and the corresponding parse tree for the derived structure obtained when $\alpha(John)$ and $\alpha(beans)$ substitute into $\alpha(cooked)$ and $\beta(dried)$ adjoins into $\alpha(beans)$ giving us a derivation tree for *John cooked dried beans.* Trees that adjoin are termed as *auxiliary trees*, trees that are not auxiliary are called *initial.* Each node in the derivation tree is the name of an elementary tree. The labels on the edges denote the address in the parent node where a substitution or adjunction has occured.

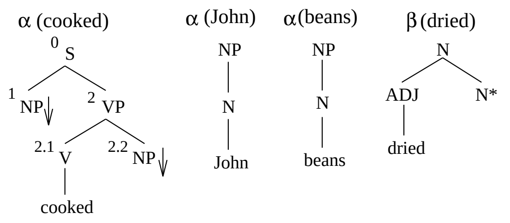

Figure 1: Example of a lexicalized TAG

3 Trees as Structured Categories

In (Joshi and Schabes, 1991) elementary trees as well as derived trees in a LTAG (a lexicalized TAG with both substitution and adjunction) were considered as structured categories. A structured category was a 3-tuple of an elementary or derived tree, the string it spanned and the functional type of the tree, e.g the 3-tuple $\langle\sigma_1, l_1, \tau_1\rangle$ in Figure 3. The functional types for trees could be thought of as defining un-Curried functions corresponding

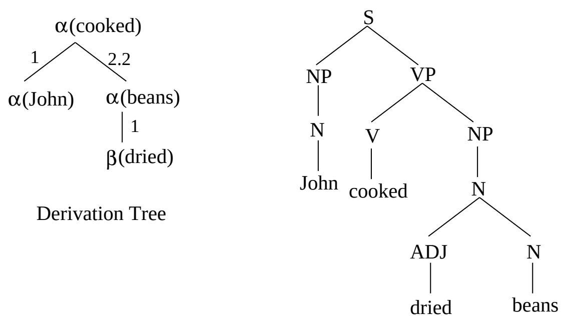

Figure 2: Example of a derivation tree and corresponding parse tree

to the Curried CCG counterpart. A functional interpretation was given to sequences of lexical items in trees even when they were not contiguous; hence discontinuous constituents were also assigned types. They were, however, barred from coordinating.

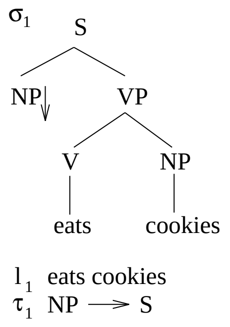

Figure 3: Structured Category for *eats cookies*

Coordination of two structured categories σ_1, σ_2 succeeded if the lexical strings of both categories were contiguous, the functional types were identical, and the least nodes dominating the strings spanned by the component tree have the same label. However, in (Joshi and Schabes, 1991) coordination was not the simple conjunction of equivalent functional types. It was a multi-step operation that picked the appropriate node at which coordination should take place, equated the shared arguments in the two structures being conjoined, and produced the appropriate derived structure (by collapsing parts of the original tree). For example, in Figure 4 the tree corresponding to *eats cookies and drinks beer* would be obtained by:

1. equating the *NP* nodes[1] in σ_1 and σ_2, preserving the linear precedence of the arguments.

2. coordinating the *VP* nodes, which are the least nodes dominating the two contiguous strings.

3. collapsing the supertrees above the *VP* node.

4. selecting the leftmost *NP* as the lexical site for the argument, since precedence with the verb is maintained by this choice.

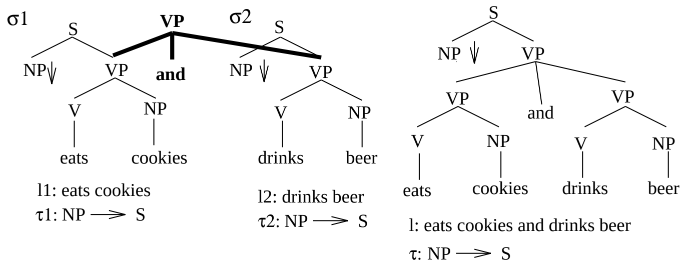

Figure 4: Coordination of *eats cookies and drinks beer*

This process handles cases of nonconstituent coordination as in (4) by coordinating at the label *S*, at the root of the two trees for *likes* and *hates*.

(4) John likes and Bill hates bananas.

In such an approach the process of coordination builds a new derived structure given previously built pieces of derived structure (or perhaps elementary structures). However, there was no clear notion of what the derivation tree for this process of coordination should be in this approach to coordination. Given these insights from (Joshi and Schabes, 1991), we present a

[1]This notion of sharing should not be confused with a deletion type analysis of coordination. The scheme presented in (Joshi and Schabes, 1991) as well as the analysis presented in this paper are not deletion analyses.

notion of coordination in LTAG that makes the same linguistic predictions as the earlier model but which operates on the elementary trees of a standard LTAG and which has a well defined notion of derivation for coordinate structures.

4 Coordination in TAG

An account for coordination in a standard LTAG cannot be given without introducing a notion of sharing of arguments in the two lexically anchored trees because of the notion of *locality* of arguments in LTAG. Consider (5) for instance, the NP *the beans that I bought from Alice* in the Right-Node Raising (RNR) construction has to be shared by the two elementary trees (which are anchored by *cooked* and *ate* respectively). Notice that in CCG this notion of "sharing" is vestigial due to type raising and function composition.

(5) (((Harry cooked) and (Mary ate)) the beans that I bought from Alice)

The notation $\downarrow$ in Figure 1 is used to denote that a node is a nonterminal and hence expects a substitution operation to occur. The notation $*$ is used to mark the foot node of an auxiliary tree. This denotes, for example, that when $\beta(\textit{dried})$ adjoins into $\alpha(\textit{beans})$ the subtree under Gorn address 1 in $\alpha(\textit{beans})$ is placed under the foot node of $\beta(\textit{dried})$. Making this notation explicit we can view an elementary tree as an ordered pair of the tree structure and an ordered set[2] of such nodes from its frontier[3], e.g. the tree for *cooked* will be represented as $\langle \alpha(\textit{cooked}), \{1, 2.1, 2.2\} \rangle$[4]. Note that this representation is not required by the LTAG formalism. The second projection of this ordered pair is used here for ease of explication. We will occasionally use the first projection of the elementary tree to refer to the ordered pair.

4.1 Setting up Contractions

We introduce an operation called *build-contraction* that takes an elementary tree, places a subset from its second projection into a *contraction set* and assigns the difference of the set in the second projection of the original

[2]The ordering is given by the fact that the elements of the set are Gorn addresses.

[3]In this paper, we shall assume there are no adjunction constraints.

[4]The reason why node address 2.1 is included in the second projection is discussed in Section 6

elementary tree and the contraction set to the second projection of the new elementary tree. The contents of the contraction set of a tree can be inferred from the contents of the set in the second projection of the elementary tree. Hence, while we refer to the contraction set of an elementary tree, it does not have to be stored along with its representation.

Figure 5 gives some examples of this operation; each node in the contraction set is circled in the figure. In the tree $\langle \alpha(\mathit{cooked}), \{1, 2.1, 2.2\}\rangle$ application of this operation on the *NP* node at address 2.2 gives us a tree with the contraction set $\{2.2\}$. The new tree is denoted by $\langle \alpha(\mathit{cooked})_{\{2.2\}}, \{1\}\rangle$, or $\alpha(\mathit{cooked})_{\{2.2\}}$ for short. Targeting the *NP* nodes at addresses 1 and 2.2 of the tree $\alpha(\mathit{cooked})$ gives us $\alpha(\mathit{cooked})_{\{1,2.2\}}$.

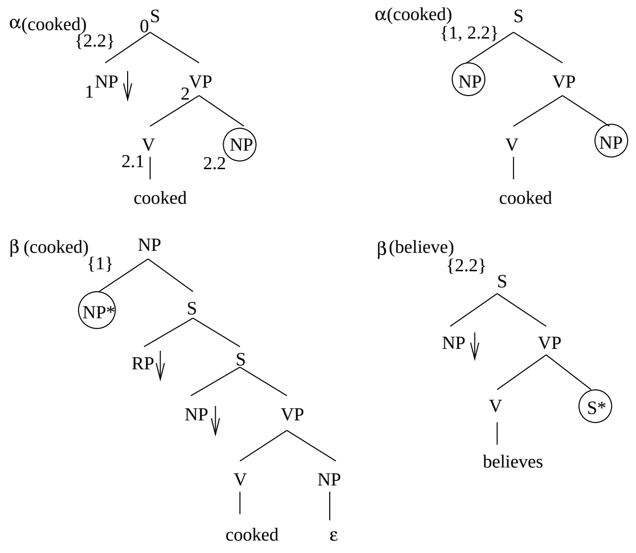

Figure 5: Building contraction sets

We assume that the anchor (the terminal that lexicalizes an elementary tree) cannot be the target of *build-contraction*. This assumption needs to be revised in the case of verbs when gapping is considered in this framework (See Section 6).

4.2 The Coordination Schema

We use the standard notion of coordination which is shown in Figure 6 which maps two constituents of *like type*, but with different interpretations, into a constituent of the same type[5].

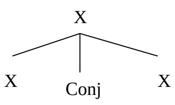

Figure 6: Coordination schema

We add a new operation to the LTAG formalism (in addition to substitution and adjunction) called *conjoin*[6]. While substitution and adjunction take two trees to give a derived tree, *conjoin* takes three trees and composes them to give a derived tree. One of the trees is always the tree obtained by specializing the schema in Figure 6 for a particular category[7].

Informally, the conjoin operation works as follows: The two trees being coordinated are substituted into the conjunction tree. This notion of substitution differs from the traditional LTAG substitution operation in the following way: In LTAG substitution, always the root node of the tree being substituted is identified with the substitution site. In the conjoin operation however, the node substituting into the conjunction tree is given by an algorithm, which we shall call *FindRoot* that takes into account the contraction sets of the two trees. *FindRoot* returns the lowest node that dominates all nodes in the second projection of the elementary tree[8].

For example, $FindRoot(\alpha(cooked)_{\{2.2\}})$ will return the root node, i.e.

[5]In this paper, we do not consider coordination of unlike categories, e.g. *Pat is a Republican and proud of it.* (Jorgensen and Abeillé, 1992) contains an analysis of these types of coordination in a TAG framework.

[6]Later we will discuss an alternative which replaces this operation by the traditional operations of substitution and adjunction.

[7]The tree obtained will be a lexicalized tree, with the lexical anchor as the conjunction: *and*, *but*, etc.

[8]This will allow a node being coordinated to dominate a pair of foot nodes. Such a case occurs, for instance, when two auxiliary trees with substitution nodes at the same tree address are coordinated with only the substitution nodes in the contraction set. This is resolved by stating the restriction on not having a discontinuous constituent in the definition of the conjoin operation. This particular problem was captured (in a similar way) by the string contiguity condition in (Joshi and Schabes, 1991).

corresponding to the *S conj S* instantiation of the coordination schema. $FindRoot(\alpha(cooked)_{\{1,2.2\}})$ will return node address 2.1, corresponding to the *V conj V* instantiation, and $FindRoot(\alpha(cooked)_{\{1\}})$ will return address 2, corresponding to the *VP conj VP* instantiation.

The conjoin operation then creates a *contraction* between nodes in the contraction sets of the trees being coordinated. The term *contraction* is taken from the graph-theoretic notion of edge contraction. In a graph, when an edge joining two vertices is contracted, the nodes are merged and the new vertex retains edges to the union of the neighbors of the merged vertices[9]. The conjoin operation supplies a new edge between each corresponding node in the contraction set and then contracts that edge. For the purposes of this paper, the contraction sets are taken to be identical[10].

Another way of viewing the conjoin operation is as the construction of an auxiliary structure from an elementary tree. For example, from the elementary tree $\langle\alpha(drinks), \{1, 2.1, 2.2\}\rangle$, the conjoin operation would create the auxiliary structure $\langle\beta(drinks)_{\{1\}}, \{2.2\}\rangle$ shown in Figure 7. The adjunction operation would now be responsible for creating contractions between nodes in the contraction sets of the two trees supplied to it. Such an approach is attractive for two reasons. First, it uses only the traditional operations of substitution and adjunction. Secondly, it treats *conj X* as a kind of "modifier" on the left conjunct X. A similar view is taken in the CCG approach. We do not choose between the two representations but for this paper, we will continue to view the conjoin operation as a part of our formalism.

In summary, the conjoin operation works as follows. Let $\mathcal{C}$ be some instance of the coordination schema and T_1 and T_2 be two elementary trees:

- substitute T_1 and T_2 into $\mathcal{C}$ using *FindRoot* if nodes in T_1 and T_2 where substitution occurs do not dominate footnodes.

- create edges between identical nodes in the contraction sets of T_1 and T_2 and contract each edge.

For example, applying *conjoin* to the trees *Conj(and)*, $\alpha(eats)_{\{1\}}$ and $\alpha(drinks)_{\{1\}}$ gives us the derivation tree and derived structure for the constituent in (6) shown in Figure 8.

[9]Merging in the graph-theoretic definition of contraction involves the identification of two previously distinct nodes. In the process of contraction over nodes in elementary trees it is the operation on that node (either substitution or adjunction) that is identified.

[10]This is a constraint on the application of the conjoin operation similar to the notion of adjoining constraints.

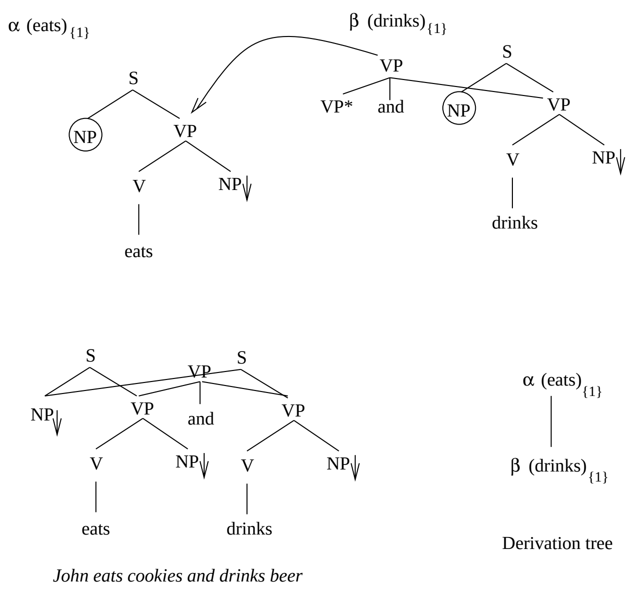

Figure 7: Coordination as adjunction.

(6) . . . eats cookies and drinks beer.

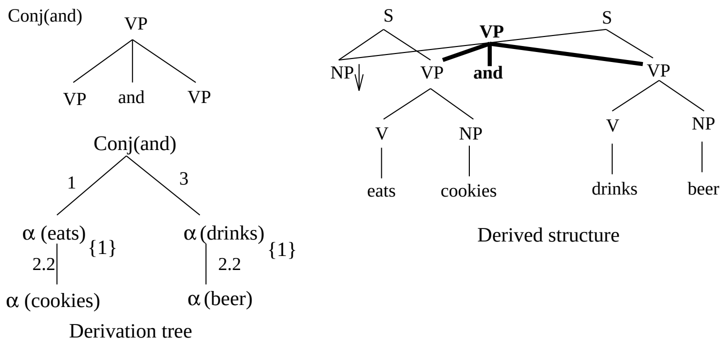

Figure 8: An example of the *conjoin* operation.

The contraction set corresponds to a set of arguments that remain to be supplied to a functor. A node in a derivation tree with a non-empty contraction set indicates that the derivation is incomplete. So, for instance, in Figure 8 the nodes $\alpha(eats)_{\{1\}}$ and $\alpha(drinks)_{\{1\}}$ signify an incomplete derivation.

4.3 The Effects of Contraction

One of the effects of contraction is that the notion of a derivation tree for the LTAG formalism has to be extended to an acyclic *derivation graph*. Simultaneous substitution or adjunction modifies a derivation tree into a graph as can be seen in Figure 9[11]. We shall use the general notation *derivation structure* to refer to both derivation trees and derivation graphs.

If a contracted node in a tree (after the conjoin operation) is a substitution node, then the argument is recorded as a substitution into the two elementary trees as for example in the sentences (7) and (8).

(7) Chapman eats cookies and drinks beer.
(8) Keats steals and Chapman eats apples.

[11] The notion of simultaneous modification at the same node address was also explored in (Schabes and Shieber, 1994) for independent reasons. However, in their formalism distinct trees modify a single node address. Hence they do not have to contend with derivation graphs.

Figure 9 contains the derivation and derived structures for (7) and Figure 10 for (8). Notice that in Figure 10 the derivation graph for sentence (8) accounts for the coordinations of the traditional nonconstituent "Keats steals" by carrying out the coordination at the root, i.e. *S conj S*. No constituent corresponding to "Keats steals" is created in the process of coordination. An example of coordination at the *V* category is given in Figure 11.

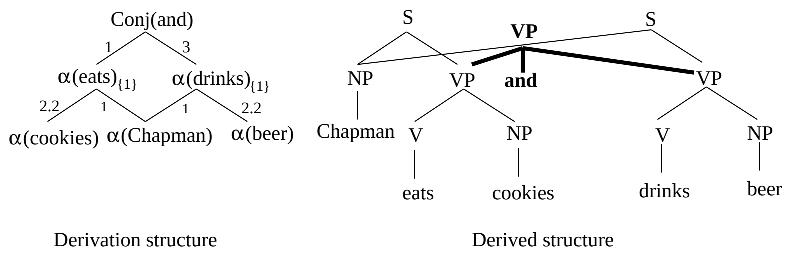

Figure 9: Derivation for *Chapman eats cookies and drinks beer.*

On the other hand if a foot node is contracted in an auxiliary tree then the effect of contraction is that both conjuncts adjoin into the same structure simultaneously, as in the sentence (9). Figure 12 contains the derivation graph and the various elementary trees for the sentence in (9).

(9) I liked the beans that Harry cooked and which Mary ate.

Considerations of the locality of movement phenomena and its representation in the LTAG formalism (Kroch and Joshi, 1986) can also now explain constraints on coordinate structure, such as across-the-board exceptions to the well known coordinate structure constraint, see Fig. 13. Also in cases of unbounded right node raising such as *Keats likes and Chapman thinks Mary likes beans* simply adjoins into the right conjunct of the coordinate structure as shown in Figure 4.3.

If we consider how the derivation obtained in Figure 13 works within the context of a larger derivation, like for instance, *I know who laughed and seemed to be happy*, then comparing the approach of using the *conjoin* operation as opposed to using the modified adjunction approach to coordination (shown for the same example in Figure 4.3), we find that in Figure 13 there is an ambiguity as the clause *I know* can attach to either *S* node, while in

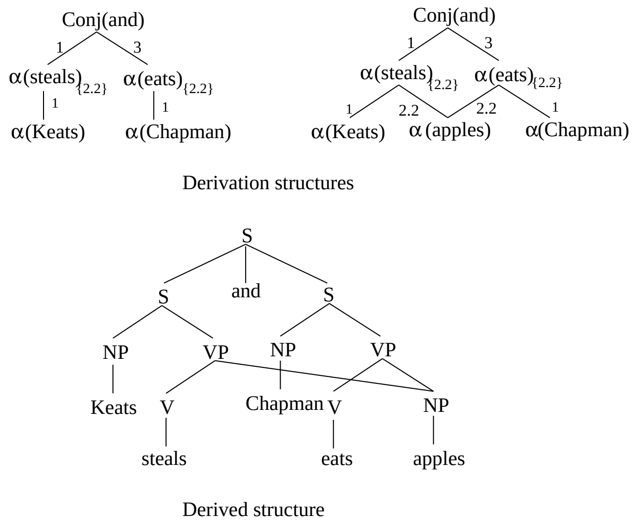

Figure 10: Derivation for *Keats steals and Chapman eats apples.*

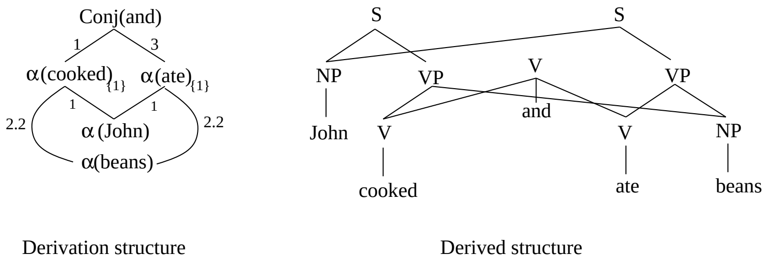

Figure 11: Derivation and derived structures for *John cooked and ate the beans.*

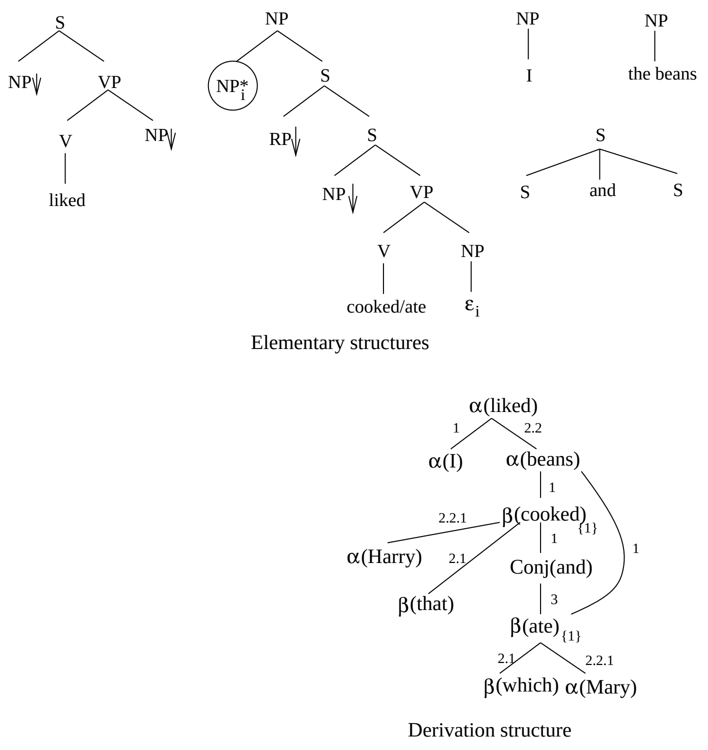

Figure 12: Derivation graph for *I liked the beans that Harry cooked and which Mary ate*

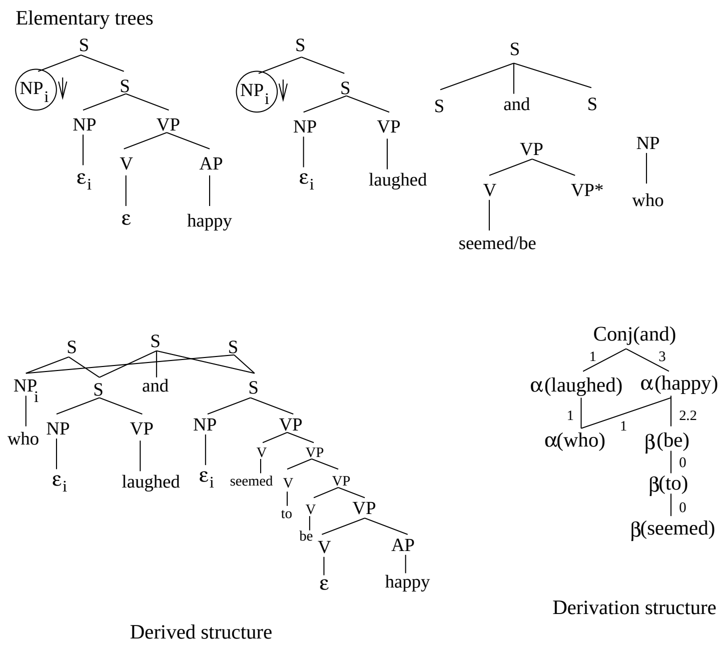

Figure 13: Derivation for *Who laughed and seemed to be happy?*

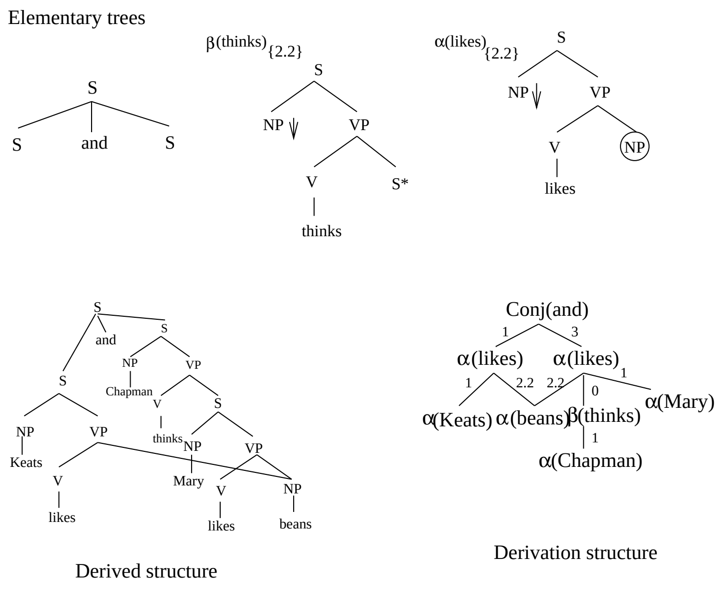

Figure 14: Derivation for *Keats likes and Chapman thinks Mary likes beans.*

Figure 4.3 since the right conjunct is a modifier, there is no such ambiguity as the derivation tree in Figure 4.3 shows.

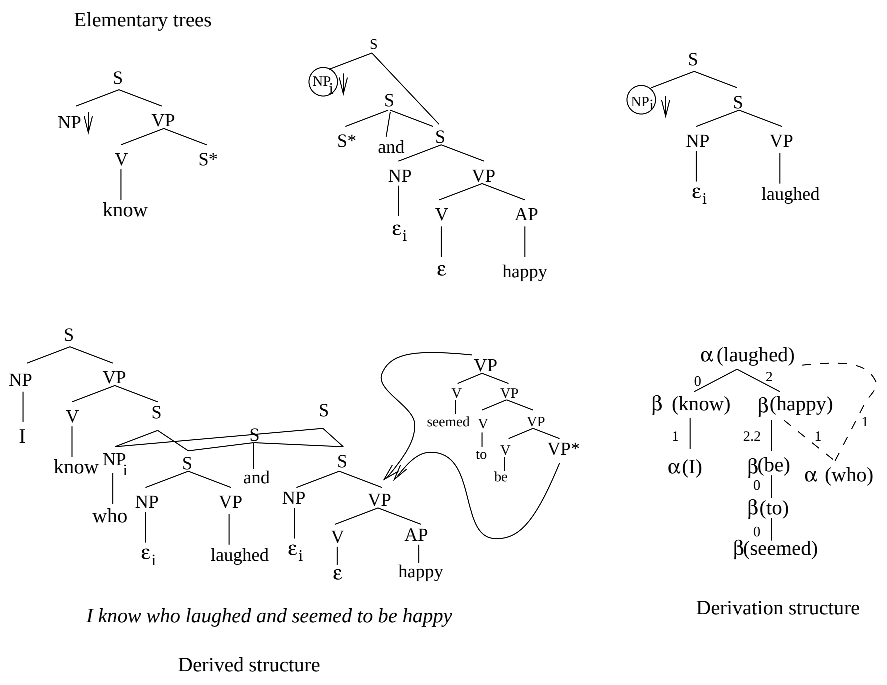

Figure 15: Derivation for *I know who laughed and seemed to be happy.*

4.4 Creating Tree Structures

The derived structures created in the above examples are difficult to reconcile with traditional notions of phrase structure. However, the derivation structure encodes the history of a derivation, i.e. exactly how the derived structure is built from particular elementary structures. Hence the derived structure is much less significant in an LTAG. Also, the derivation tree gives us all the information about dependency that we need about the constituent.

Figure 16 shows one way of reconciling the derived structure given by the derivation graph in Figure 9 to a tree structure. The root of the conjunction tree assumes the position of its conjuncts in the derived tree. The parents of each conjunct are also merged. In general, for any coordinated node the supertree upto the root of its elementary tree has to be merged.[12].

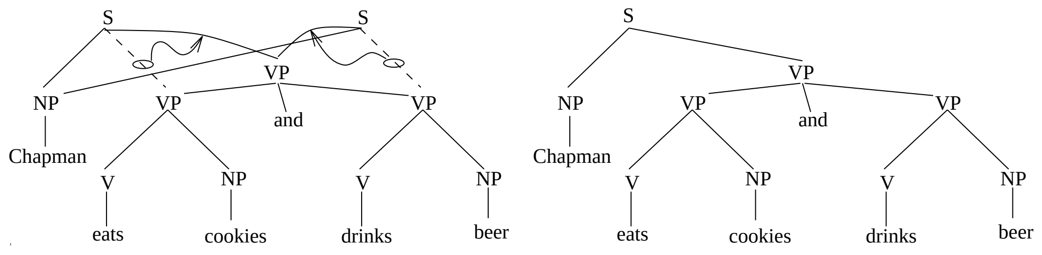

Figure 16: Producing a tree by collapsing supertrees in a derived structure.

5 Comparisions

Using the analogy of elementary trees as structured categories, we can view substitution either as function application or in cases where the substituted element itself brings in its argument structure as function composition. Adjunction is always like function composition. Contraction can be viewed as distributing the arguments or functors for the simultaneous use of functional application or composition

Although the approach presented using TAGs is CCG-like, it builds derivations over larger structures, namely the elementary trees in a TAG. This encodes locally in a tree, a derivation history which is non-local in a CCG. Sometimes, such a history can be useful. For instance, in the sentence *John thinks Mary and Harry won* using type raising and composition rules, the string *John thinks Mary* can be associated with the type *S/(S\NP)* (see Figure 17. This is the type associated with a type raised NP such as *Harry*, thus the coordination rule can apply on these constituents to give us *((John thinks Mary) and (Harry)) won.* (This was also noted in (Henderson, 1992).) One way to rule out this derivation is by adding some kind of feature to the type *S/(S\NP)*. Such a feature would encode (in some appropriate way) the fact that the type *S/(S\NP)* for the left conjunct, *John thinks Mary*, has been derived from the type *S/S* while the type *S/(S\NP)* for the right

[12]The tree addresses in the tree created by the derivation structure also have to be properly updated.

conjunct, *Harry*, has been type raised from an *NP*. In the approach to coordination presented in this paper, such a problem does not arise since entire elementary trees are coordinated. This provides enough context to rule out an LTAG derivation analogous to the CCG derivation.

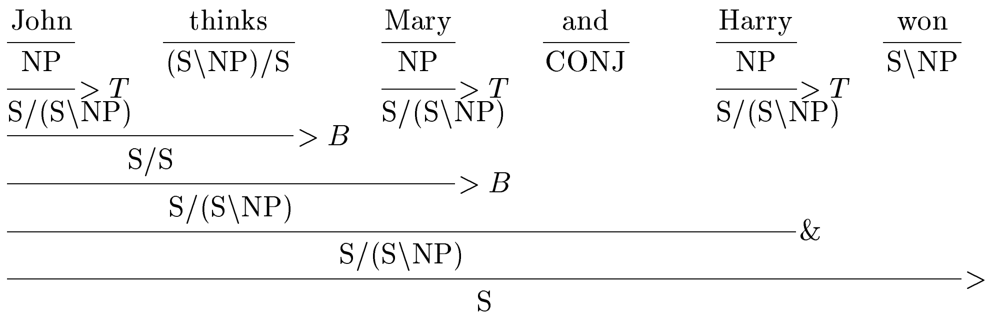

Figure 17: A CCG derivation for *John thinks Mary and Harry won*.

6 Contractions on Anchors

We now address the earlier assumption that the anchor (the terminal that lexicalizes a tree) cannot be the target of *build-contraction*. An LTAG along with the operations of substitution and adjunction also has the implicit operation of lexical lookup or lexical insertion (represented as the diamond mark in Figure 18). Under this view, the LTAG trees are taken to be templates. For example, the tree in Figure 18 is represented as $\langle \alpha(eat), \{1, 2.1, 2.2\} \rangle$.

If we extend the notion of contraction in the conjoin operation together with the operation of lexical insertion we have the following observations:

- The two trees to be used by the conjoin operation are no longer strictly lexicalized as the label associated with the diamond mark is a preterminal.
- Previous uses of conjoin applied to two distinct trees. If the lexicalization operation is to apply simultaneously the same anchor projects two elementary trees from the lexicon.
- Since two distinct copies of the anchor are not selected from the lexicon, the terminal string at the anchor position in one of the two ele-

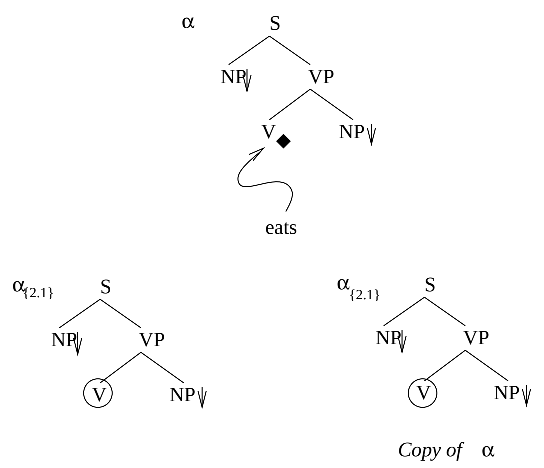

Figure 18: Lexicalization in a LTAG.

mentary trees is realized as the null string. Its interpretation however is determined by the common anchor.

Earlier in Section 4.2 we had considered an alternative to the *conjoin* operation which involves only the usual operations of substitution and adjunction. However, when contractions are performed on anchors it is not appropriate to treat *conj X* as a "modifier" on some category *X*. Here we have to use the conjoin operation, as combining three elementary structures, instead. This is perhaps consistent with the observation that the constructions discussed in Sections 6.1 and 6.2 are difficult to describe as *conj X* "modifying" the left conjunct *X*. For these reasons, although treating conjuncts as introduced by adjunction has several appealing advantages, we have presented most of the discussion in this paper in terms of the *conjoin* operation.

6.1 Gapping

Using this extension to *conjoin*, we can handle sentences that have the "gapping" construction like sentence (10).

(10) John ate bananas and Bill strawberries.

The conjoin operation applies to copies of the same elementary tree when the lexical anchor is in the contraction set. For example, let $\alpha(eats)$ be the tree selected by *eats*. The coordination of $\alpha(eats)_{\{2.1\}}$ with a copy of itself and the subsequent derivation tree is depicted in Fig. 19[13].

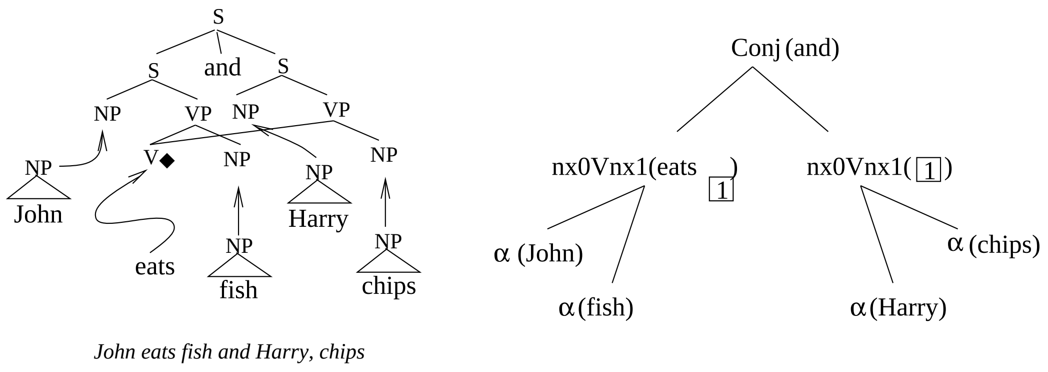

Figure 19: Handling the gapping construction using contractions.

From a parsing perspective, for these simple cases of gapping, the structure can be built before the input is handed to the parser.

(11) John wants Penn to win and Bill, Princeton.
(12) John wants to try to see Mary and Bill, Susan.

However, to handle cases such as sentences (11) and (12), lexical insertion has to be handled by the parser while building a derivation. The identity of copies of trees as opposed to originals is relevant here as allowing adjunctions of copies onto originals and vice versa would create incorrect derivations. Hence appropriate constraints on adjunction have to be imposed on copies made while targeting an anchor for contraction.

6.2 Coordinating Ditransitive verbs.

In sentence (13) if we take the position that the string *Mary a book* is not a constituent (i.e. *give* has a structure as in Fig. 20), then we can use the notion of contraction over the anchor of a tree to derive the sentence in (13). The structure we derive is shown in Fig. 21.

[13]In English, following (Ross, 1970), the anchor goes to the left conjunct.

(13) John gave Mary a book and Susan a flower.

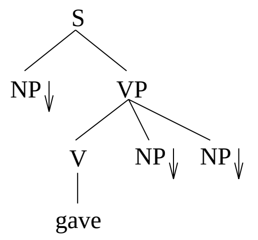

Figure 20: Tree for a ditransitive verb in LTAG.

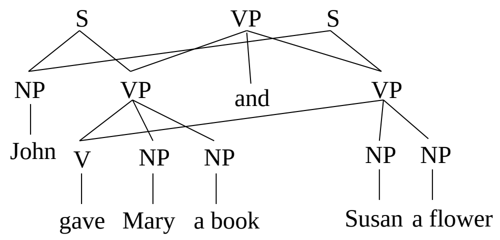

Figure 21: Derived tree for *John gave Mary a book and Susan a flower.*

6.3 Interactions.

Permitting contractions on multiple substitution and adjunction sites along with contractions on the anchor allow the derivation of *stripping* structures such as (14) (where the conjunct *Bill too* can be interpreted as *[John loves] Bill too* or as *Bill [loves Mary] too.*

(14) John loves Mary and Bill too.

7 Parsing Issues

This section discusses the parsing issues that arise in the modified TAG formalism that we have presented in this paper. We do not discuss the general issues in parsing TAGs, rather we give the appropriate modifications that are needed for the formalism in this paper to the existing algorithm for TAGs due to (Schabes and Joshi, 1988). Modifications to the parser are given as inference rules in the deductive parsing framework described in (Shieber, Schabes, and Pereira, 1995), following (Schabes, 1994).

7.1 Notation

Let $G = (\Sigma, NT, I, A, S, \mathcal{C})$ be a TAG with the operations of substitution, adjunction and the conjoin operation. Σ is the set of terminal symbols, NT is the set of non-terminals distinct from Σ, I and A are the sets of initial and auxiliary trees, $I \cup A$ is the set of elementary trees, S is the distinguished start symbol and $\mathcal{C}$ is the set of trees that are instantiations of the coordination schema (see Figure 6). All the sets are finite. Suppose $a_1 \ldots a_n$ is an input string. The Greek letters μ, ρ and ν are used to denote nodes in elementary trees. Greek letters ψ and χ are used to denote nodes in trees from the set $\mathcal{C}$. We assume that multiple adjunctions on a single node are allowed (Schabes and Shieber, 1994). This is a modification from the standard TAG derivation (Vijay-Shanker, 1987) where it was disallowed.

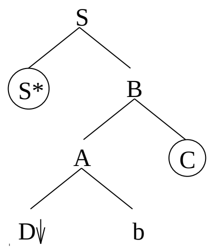

Figure 22: An auxiliary tree with a contraction set

A layer of an elementary tree is represented textually in a style similar to a production rule, e.g. $\mu_X \rightarrow \nu_Y \rho_Z$. For instance, the tree in Figure 22 is represented by the production rules in (1)[14].

The predicate *Init*(μ_X) is true if and only if μ_X is the root of an initial tree. For each $\psi \in \mathcal{C}$, *Init*(ψ) is defined to be true. The predicate $Aux(\rho_X)$

[14]This representation and the notations for presenting the deductive parsing algorithm have been inspired by (Schabes and Waters, 1995).

is true if and only if ρ_X is the root of an auxiliary tree. The predicate *Subst*(μ_X) is true if and only if μ_X is marked for substitution. The predicate *Foot*(μ_X) is true if and only if μ_X is the foot node of an auxiliary tree. The predicate *Share*(μ_X) is true if and only if μ_X is in the contraction set of an elementary tree μ. The predicate *Root*(μ_X) is true if and only if μ_X is the highest node (smallest Gorn address) that dominates all nodes μ_Y such that $\neg$*Share*(μ_X) and *Subst*(μ_Y) or *Foot*(μ_Y) is true. The predicate *Conj*(ψ_X) is true if and only if ψ_X is the category being coordinated in some tree $\psi \in \mathcal{C}$. The predicate *Conjoin*(ψ_X, μ_X, ν_X) is true if and only if μ and ν can be conjoined at node X using $\psi \in \mathcal{C}$. This predicate can check certain constraints such as identical contraction sets in the trees being coordinated.

$$\begin{aligned}
&\mu_S^0 \rightarrow \mu_S^1 \mu_B^2 \\
&\mu_B^2 \rightarrow \mu_A^{2.1} \mu_C^{2.2} \\
&\mu_A^{2.1} \rightarrow \mu_D^{2.2.1} \mu_b^{2.2.2} \\
&Aux(\mu_S^0) \\
&Root(\mu_A^{2.1}) \\
&Foot(\mu_S^1) \wedge Share(\mu_S^1) \\
&Subst(\mu_C^{2.2}) \wedge Share(\mu_C^{2.2}) \\
&Subst(\mu_D^{2.2.1})
\end{aligned} \tag{1}$$

7.2 Left to Right Parsing

The algorithm relies on a tree traversal that scans the input string from left to right while recognizing the application of the conjoin operation on the elementary trees selected by the terminals in the input string. The nodes in the elementary trees are visited top-down left to right as defined in (Schabes, 1994) (see Figure 23).

In a manner analogous to dotted rules for CFGs as defined in (Earley, 1970) the dot in Figure 23 divides a subtree into a left context and a right context, enabling the algorithm to scan the elementary tree in a top-down left to right manner while trying to recognize possible applications of the conjoin operation.

The derived structure corresponding to a succesful conjoin operation is a composite structure built by conjoining two elementary trees into an instantiation of the coordination schema. The algorithm never builds derived

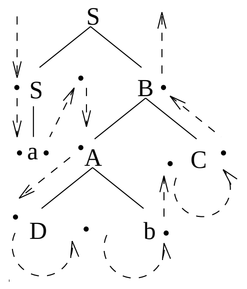

Figure 23: Example of a tree traversal

structures. It builds the derivation by visiting the appropriate nodes during its tree traversal in the following order (see Figure 24).

$$1\ 2 \cdots 3\ 4 \cdots 5\ 6 \cdots 2'\ 7' \cdots 3'\ 4' \cdots 5'\ 6' \cdots 7\ 8$$

The other task of the algorithm is to compute the correct span of the input string for the nodes that have been identified with each other via a contraction. Figure 24 gives the possible scenarios to be considered for the position of shared nodes in the derivation (i.e. nodes that have been linked by a contraction). All of the cases in Figure 24 can occur while building a derivation structure for the conjoin operation.

Specifically, when foot nodes undergo contraction, the algorithm has to ensure that both the foot nodes share the subtree pushed under them by the *predict completion* move, e.g. $\cdots 9\ 10 \cdots$ and $\cdots 9'\ 10' \cdots$ in Figure 24(a). Similarly, when substitution nodes undergo contraction, the algorithm has to ensure that the tree recognized due to the *predict substitution* move is shared by the nodes, e.g. $\cdots 11\ 12 \cdots$ and $\cdots 11'\ 12' \cdots$ in Figures 24(b) and 24(c) . The various positions in a top-down, left-to-right traversal at which these nodes can occur is also shown in Figure 24.

In order to achieve this traversal across and within trees, some data structures need to be defined.

7.3 Chart states

Dot positions in an elementary tree are represented by placing a dot in the production for the corresponding layer in the tree. For example, the first

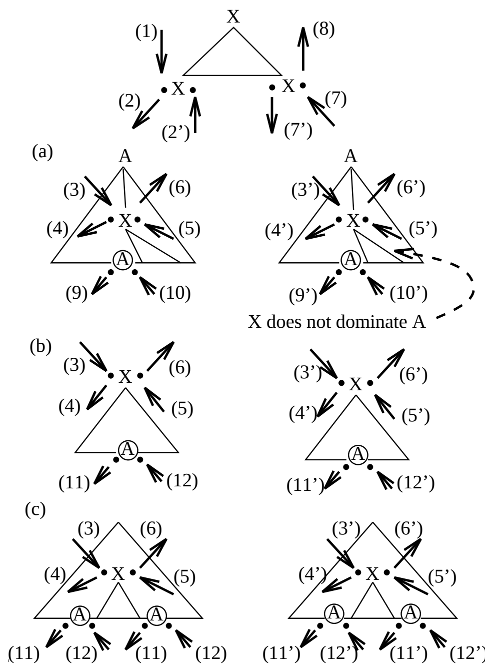

Figure 24: Moving the dot while recognizing a conjoin operation

dot position in Figure 23 is $\mu^0_S \to \bullet\mu^1_S\mu^2_B$. In dotted layer productions, the Greek letters γ, δ, and ζ are used to represent sequences of zero or more nodes.

The algorithm collects states into a set called a *chart*. States are placed and maintained in the chart by a suitable agenda mechanism. A *state* is a 6-tuple $[p, i, j, k, l, rec?]$ where: p is a position in an elementary tree, i, j, k, l are indices of positions in the input string recognized so far, i and l are always bound, j and k can be unbound written as $-$, $rec?$ is a boolean flag used at some state $[\mu_X \to \gamma\mu_Y \bullet \delta, i, j, k, l, rec?]$ and *rec?* is *true* if and only if *Share*(μ_Y) and either the subtree under footnode μ_Y or a substitution at μ_Y has been recognized.

Recognition of the subtree under the foot-node which is in a contraction set or completion of substitution on a node in a contraction set can occur in a variety of environments. It can occur before the conjoin operation is predicted, as in Figure 24(a) (when X does not dominate A) and Figure 24(c). It can also occur after prediction of the conjoin operation but before completion, as in Figure 24(a) (X dominates A) and Figure 24(b). Even after completion of the conjoin operation as in Figure 24(a) (X does not dominate A) and Figure 24(c). To handle all of these cases it is useful to define a notation to handle unifying the indices of shared nodes in the chart, say μ_X and ρ_X, such that *Share*(μ_X) is true and *Share*(ρ_X) is true. Assuming $[\mu_X \to \bullet\gamma, i, j, k, l, rec?_{\mu_X}]$ and $[\rho_X \to \bullet\delta, m, n, o, p, rec?_{\rho_X}]$ are states in the chart, let $\mu_X \sqcup \rho_X$ be shorthand for $[i, j, k, l] \sqcup [m, n, o, p]$ and when the indices have values spanning the input string, i.e. they are not unbound, $rec?_{\mu_X} = rec?_{\rho_X} = true$.

7.4 Parsing Algorithm

The algorithm is given as a set of inference rules. It is only concerned with the conjoin operation. In a full parser, handling substitution and adjunction would proceed exactly as defined in (Schabes, 1994).

We initialize the chart by adding all dot positions at the root of an initial tree. This is also done for initial trees with non-empty contraction sets and for instantiations of the coordination schema.

$$\mathit{Init}(\mu_S) \vdash [\mu_S \to \bullet\gamma, 0, -, -, 0, rec?] \tag{2}$$

The algorithm handles two tasks, one is the recognition of the conjoin operation and the other is the handling of contractions. Traversal of all trees, including trees with contractions, is handled by the traversal mechanisms of the standard parser (Schabes, 1994).

We predict all possible elementary trees with the dot at some node ψ_X in an instantiation of the coordination schema where a conjoin operation is predicted, i.e. *Conj(*ψ_X*)* is true.

$$\begin{aligned}&[\psi_X^m \rightarrow \bullet \psi_X^n \gamma, i, j, k, l, rec?] \wedge \mathit{Conj}(\psi_X^n) \wedge \mathit{Root}(\mu_X) \\ &\qquad\qquad \vdash [\mu_X \rightarrow \bullet\gamma, l, -, -, l, rec?]\end{aligned} \tag{3}$$

Also, we predict completion of all nodes recognized under ψ_X.

$$\begin{aligned}&[\mu_X \rightarrow \gamma\bullet, l, -, -, m, rec?] \wedge \mathit{Root}(\mu_X) \\ &\qquad \wedge[\psi_X^m \rightarrow \bullet\psi_X^n \gamma, i, j, k, l, rec?] \\ &\qquad \vdash [\psi_X^m \rightarrow \psi_X^n \bullet \gamma, i, j, k, m, rec?]\end{aligned} \tag{4}$$

When the chart contains a state of the form $[\psi_X \rightarrow \gamma\bullet, i, j, k, l, rec?]$, where ψ_X is the root of some instantiation of the coordination schema, we can complete recognition of the conjoin operation. This rule searches the chart for two conjuncts and completes the conjoin operation. This state also triggers the rule that ensures that positions over the input string for nodes that are shared will be identical. The actual positions may be computed by the inference rules that handle contractions.

$$\begin{aligned}&[\psi_X \rightarrow \gamma\bullet, i, j, k, l, rec?] \wedge \mathit{Conjoin}(\psi_X, \mu_X, \rho_X) \\ &\wedge[\mu_X \rightarrow \delta\bullet, p, j, k, q, rec?] \wedge [\rho_X \rightarrow \zeta\bullet, j, x, y, k, rec?] \\ &\qquad \wedge\forall\mu_Y^i(\mathit{Share}(\mu_Y^i))(\mu_Y^i \sqcup \rho_Y^i) \\ &\qquad \vdash [\psi_X \rightarrow \gamma\bullet, i, p, q, l, rec?]\end{aligned} \tag{5}$$

The complexity of the algorithm stems from step (5). Since to recognize completion of the conjoin operation the algorithm has to loop over the chart to match values of eight distinct indices, the time complexity is $O(n^8)$. This step accounts for the complexity of the entire algorithm since it is the most expensive step. It is not clear if one can do better at this task, however by adopting the conjoin operation as a modified adjunction operation mentioned earlier, the complexity can be brought back to the $O(n^6)$ of a standard Earley-style parser (Schabes, 1994).

The following inference rules handle substitution and the recognition of the subtree under a foot node for the nodes that are shared via contractions between two trees. Rather than repeat the inference rules in the parser for a standard TAG, we will assume that those rules do not apply to nodes μ_X when *Share(μ_X)* is true.

The first such inference rule skips over a contracted node without accepting any part of the input string.

$$\begin{aligned}[\mu_X \rightarrow \bullet\delta, i, j, k, l, rec?] \wedge \mathit{Share}(\mu_X) \\ \vdash [\mu_X \rightarrow \delta\bullet, i, j, k, l, false]\end{aligned} \tag{6}$$

If some contracted foot node can complete recognition in the subtree under the foot node we record this fact on the state for future reference.

$$\begin{aligned}[\mu_X \rightarrow \mu_A\bullet, i, j, k, l, rec?] \\ \wedge[\rho_Y \rightarrow \bullet\rho_A, i, -, -, i, rec?] \wedge \mathit{Foot}(\rho_A) \wedge \mathit{Share}(\rho_A) \\ \vdash [\mu \rightarrow \gamma\bullet, i, i, l, l, true]\end{aligned} \tag{7}$$

Similarly, if some contracted node can complete substitution then we mark this fact on the state for future reference.

$$\begin{aligned}[\rho_X \rightarrow \rho_A\bullet, l, -, -, m, rec?] \\ \wedge[\mu_Y \rightarrow \bullet\mu_A, i, j, k, l, rec?] \wedge \mathit{Subst}(\mu_A) \wedge \mathit{Share}(\mu_A) \\ \vdash [\mu_Y \rightarrow \mu_A\bullet, i, j, k, m, true]\end{aligned} \tag{8}$$

If there is a state of the form $[\mu^0_S \rightarrow \gamma\bullet, 0, -, -, n, rec?]$ in the chart with $\mu \in I$ and for all states that contribute to the derivation (this can be done in linear time if along with placing states in the chart we annotate them with the reason for adding them to the chart), the value for *rec*? in those states is *true*.

8 Conclusion

In summary, we have shown that a CCG-like account for coordination can be given in a LTAG while maintaining the notion of a derivation tree which is central to the LTAG approach. We showed that fixed constituency can be maintained at the level of the elementary tree while accounting for cases of non-constituent coordination. In the discussion of coordination the central operation of *contraction* was disallowed on items that anchor an elementary tree. We showed that the "gapping" construction as well as cases of non-constituent coordination can be satisfactorily handled by allowing such an operation to work on anchors. We have also briefly presented a parser which was used in extending the XTAG system (XTAG Research Group, 1995) thus extending it to handle coordination. This is the first full implementation of coordination in the LTAG framework.

9 Acknowledgements

We would like to thank Daniel Hardt, Nobo Komagata, Seth Kulick, David Milward, Jong Park, James Rogers, Yves Schabes, B. Srinivas, and Mark Steedman for their valuable comments.

References

Earley, J. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Henderson, James. 1992. A structural interpretation of combinatory categorial grammar. Technical Report MS-CIS-92-49, University of Pennsylvania, Philadelphia, PA.

Jorgensen, H. and A. Abeillé. 1992. Coordination of "Unlike" Categories in TAG. In *Proceedings of the 2nd TAG Workshop*, Philadelphia, PA.

Joshi, Aravind and Yves Schabes. 1991. Fixed and flexible phrase structure: Coordination in Tree Adjoining Grammar. In *Presented at the DARPA Workshop on Spoken Language Systems*, Asilomar, CA.

Kroch, A. and A. K. Joshi. 1986. Analyzing extraposition in a tree adjoining grammar. In G. Huck and A. Ojeda, editors, *Syntax and Semantics: Discontinuous Constituents.* Academic Press, New York.

McCawley, James. 1982. Parentheticals and discontinuous constituent structure. *Linguistic Inquiry*, 13(1):91–106.

Ross, John. 1970. Gapping and the order of constituents. In M. Bierwisch and K. Heidolph, editors, *Progress in Linguistics.* Mouton, The Hague.

Sarkar, Anoop and Aravind Joshi. 1996. Coordination in TAG: Formalization and implementation. In *Proceeding of COLING'96*, Copenhagen.

Schabes, Yves. 1990. The valid prefix property and left to right parsing of tree adjoining grammars. In *Proceedings of the Second International*

Workshop on Parsing Technologies, Cancun, Mexico, August. Association for Computational Linguistics.

Schabes, Yves. 1994. Left to right parsing of lexicalized tree adjoining grammars. *Computational Intelligence*, 10(4):506–524.

Schabes, Yves and Aravind K. Joshi. 1988. An Earley-type parsing algorithm for tree adjoining grammars. In *26th Meeting of the Association for Compuatational Linguistics (ACL '88)*, Buffalo, NY. Association for Computational Linguistics.

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

Schabes, Yves and Richard Waters. 1995. Tree insertion grammar: A cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–513.

Shieber, Stuart, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic and Computation*, 24(1-2):3–36.

Steedman, Mark. 1985. Dependency and coordination in the grammar of Dutch and English. *Language*, 61:523–568.

Steedman, Mark. 1990. Gapping as constituent coordination. *Linguistics and philosophy*, 13:207–264.

Steedman, Mark. 1997. *Surface Structure and Interpretation: Unbounded and Bounded Dependency in Combinatory Grammar.* Linguistic Inquiry monograph (to appear). MIT Press.

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

Vijay-Shanker, K., Aravind Joshi, and David Weir. 1990. The convergence of mildly context-sensitive grammatical formalisms. In Peter Sells, Stuart Shieber, and Tom Wasow, editors, *Foundational Issues in Natural Language Processing.* MIT Press, Cambridge MA.

Weir, David J. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms.* Ph.D. thesis, Dept. of Computer and Info. Sc., University of Pennsylvania, Philadelphia, PA.

XTAG Research Group. 1995. A lexicalized tree adjoining grammar for english. Technical report, IRCS, University of Pennsylvania, Philadelphia, PA.