Multiple Adjunction in Feature-Based Tree-Adjoining Grammar

Claire Gardent*
CNRS, LORIA

Shashi Narayan*
Université de Lorraine, LORIA

In parsing with Tree Adjoining Grammar (TAG), independent derivations have been shown by Schabes and Shieber (1994) to be essential for correctly supporting syntactic analysis, semantic interpretation, and statistical language modeling. However, the parsing algorithm they propose is not directly applicable to Feature-Based TAGs (FB-TAG). We provide a recognition algorithm for FB-TAG that supports both dependent and independent derivations. The resulting algorithm combines the benefits of independent derivations with those of Feature-Based grammars. In particular, we show that it accounts for a range of interactions between dependent vs. independent derivation on the one hand, and syntactic constraints, linear ordering, and scopal vs. nonscopal semantic dependencies on the other hand.

1. Introduction

A Tree Adjoining Grammar (TAG; Joshi and Schabes 1997) consists of a set of elementary trees and two combining operations, substitution and adjunction. Consequently, a TAG derivation can be described by a tree (called a **derivation tree**) specifying which elementary TAG trees were combined using which operations to yield that derivation. In this tree, each vertex is labeled with a tree name and each edge with a description of the operation (node address and operation type) used to combine the trees labeling its end vertices. As we shall see in Section 3.2, in TAG, each derivation tree specifies a unique parse tree, also called **derived tree**.

In previous work, it has been argued that TAG derivation trees provide a good approximation of semantic dependencies between the words of a sentence (Kroch 1989; Rambow, Vijay-Shanker, and Weir 1995; Candito and Kahane 1998; Kallmeyer and Kuhlmann 2012). As shown by Schabes and Shieber (1994), however, there are several possible ways of defining TAG derivation trees, depending on how multiple adjunction of several auxiliary trees at the same tree node is handled. The *standard notion of derivation* proposed by Vijay-Shanker (1987) forbids multiple adjunction, thus enforcing *dependent* derivations. In contrast, the *extended notion of derivation* proposed by Schabes

* UMR 7503, Campus Scientifique, BP 239, F-54506 Vandoeuvre-lès-Nancy Cedex, France.
E-mail:{claire.gardent,shashi.narayan}@loria.fr.

Submission received: 2 August 2013; revised version received: 16 June 2014; accepted for publication: 21 September 2014.

doi:10.1162/COLI_a_00217

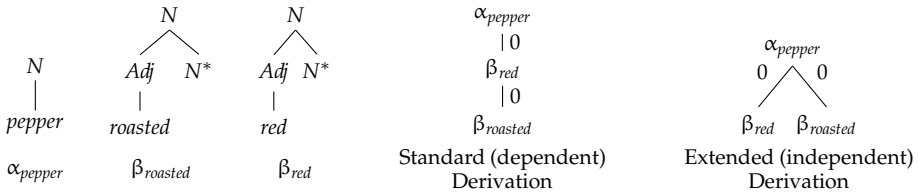


Figure 1
An example TAG with the alternative TAG derivations for the phrase *roasted red pepper*. α_{pepper}, β_{red}, and $\beta_{roasted}$ are the elementary trees for *pepper* (initial tree), *red* (auxiliary tree), and *roasted* (auxiliary tree), respectively.

and Shieber (1992, 1994) allows multiple adjunction at a single node, thereby yielding so-called **independent** derivations (i.e., derivations where the relation between the adjoining trees is left unspecified). The difference between the two types of derivations is illustrated in Figure 1. While in the standard (dependent) derivation, one adjective tree is adjoined to the other adjective tree, which itself is adjoined to the noun tree for *pepper*; in the extended (independent) derivation, both adjective trees adjoin to the noun tree.

Schabes and Shieber (1994) argue that allowing both for dependent and independent derivations better reflects linguistic dependencies. Making use of the distinction introduced in TAG between predicative and modifier auxiliary trees (Schabes and Shieber (1994), Section 3.1), they define a parsing algorithm that assigns dependent derivations to predicative auxiliary trees but independent derivations to multiple modifier auxiliary trees adjoining to the same node. In case both predicative and modifier auxiliary trees adjoin to the same node, their parsing algorithm ensures that predicative trees appear above the modifier trees in the derived tree.

This parsing algorithm is defined for featureless variants of TAG. In contrast, in implemented TAGs (e.g., XTAG [The XTAG Research Group 2001], SemXTAG [Gardent 2008], or XXTAG[1] [Alahverdzhieva 2008]) feature structures and feature unification are central. They are used to minimize the size of the grammar; to model linguistic phenomena such as verb/subject agreement; and to encode a unification-based syntax/semantics interface (e.g., Gardent and Kallmeyer 2003).

In this article, we extend Schabes and Shieber's proposal to Feature-Based TAG (FB-TAG); and we show that the resulting parsing algorithm naturally accounts for the interplay of dependent vs. independent derivation structures with syntactic constraints, linear ordering, and scopal vs. nonscopal semantic dependencies.

The article is organized as follows. In Section 2, we recap the motivations for independent derivations put forward by Schabes and Shieber (1994) and we briefly discuss the interactions that may arise between dependent and independent derivations. Section 3 summarizes their approach. In Section 4, we present the intuitions and motivations underlying our proposal and we highlight the differences with Schabes and Shieber's approach. Section 5 presents our proposal. Section 6 concludes.

2. Why Are Independent Derivations Desirable?

We start by summarizing Schabes and Shieber's motivations for independent derivations. We then discuss the interactions between dependent and independent derivations.

1 XXTAG stands for XMG (Crabbé et al. 2013)-based XTAG.

2.1 Motivations for Independent Derivations

Schabes and Shieber (1994) give three main motivations for independent derivations. The first motivation concerns the interaction of verbs with multiple modifiers. Consider sentences[2] in Examples (1) and (2).

(1) a. Richard Parker and Pi wandered the Algae Island yesterday through the meerkats.
b. Richard Parker and Pi wandered the Algae Island yesterday.
c. Richard Parker and Pi wandered the Algae Island through the meerkats.

(2) a. ⊗ The Orangutan reminded Pi of his mother yesterday through the meerkats.
b. The Orangutan reminded Pi of his mother yesterday.
c. ⊗ The Orangutan reminded Pi of his mother through the meerkats.

Movement verbs such as *to wander* allow for directional modifiers such as *through the meerkats*, whereas verbs such as *to remind* do not. In TAG, such restrictions can be modeled using selective adjoining constraints to specify which modifier tree may or may not be adjoined at a particular node in a given tree. Therefore, it is possible to license (1) and to rule out (2c). In Example (2a), however, under the dependent notion of adjunction, the tree for the directional adverbial *through the meerkats* will adjoin to the modifier tree for *yesterday*, which itself will adjoin to the tree selected by *reminded*. Thus, constraints placed by the verb on its modifiers must be passed through by modifier trees (here, the tree for *yesterday*) to also rule out sentences such as Example (2a). Propagating selective adjunction constraints in TAG would lead to a formalism for which derivation trees are no longer context-free (Schabes and Shieber 1994).

The second motivation for independent adjunction stems from probabilistic approaches. Stochastic lexicalized TAG specifies the probability of an adjunction of a given auxiliary tree at a given node in another elementary tree (Resnik 1992; Schabes 1992). Thus, under the standard notion of derivation, the overall probability of the string *roasted red pepper* would be determined by the probability of *red* adjoining to *pepper* and the probability of *roasted* adjoining to *red*. In contrast, independent adjunction would result in a derivation such that the overall probability of the string *roasted red pepper* would be determined by the probability of both *red* and *roasted* adjoining to *pepper*. Schabes and Shieber (1994, page 97) argue that it is plausible that "the most important relationships to characterize statistically are those between modifier and modified, rather than between two modifiers."

A third motivation comes from semantics and, more particularly, from scope ambiguities involving modifiers. Given a sentence such as Example (3), where the relative scope of the modifiers *twice* and *intentionally* is ambiguous,[3] Shieber (1994) shows that, under the extended definition of adjunction, a synchronous TAG modeling the relation between syntactic trees and logical formulae can account for both readings.

(3) John blinked twice intentionally.

2 The characters in these sentences are borrowed from Yann Martel's book *Life of Pi*.
3 The sentence can describe either *a single intentional act of blinking twice* or *two intentional acts each of single blinking* (Shieber 1994).

The account crucially relies on multiple independent adjunction of the two modifier trees to the tree for *blink*: Depending on which order the auxiliary trees for *twice* and *intentionally* adjoins to *blink*, the logical formula built will be either *intentionally(twice(blink))* or *twice(intentionally(blink))*, thus capturing the ambiguity.

2.2 Dependent, Independent, and Mixed Derivations

To capture the different types of semantic dependencies and morpho-syntactic constraints that may hold between multiple auxiliary trees adjoining to the same entity, both dependent and independent derivations are needed.

As argued earlier, because there are no constraints or semantic relation holding between each of them, multiple intersective modifiers applying to the same entity (Example (4)) are best modeled using an independent derivation.

(4) The tall black meerkat slept. (Independent derivation)

In contrast, because they may involve strong scopal and morpho-syntactic constraints, stacked predicative verbs (i.e., verbs taking a sentential complement, Example (5a)) and non-intersective modifiers (Example (5c)) require dependent derivations. Consider sentences (5a–b), for instance. If predicative trees were assigned an independent derivation, sentence (5a) would be judged ungrammatical (because *want* requires an infinitival complement but would adjoin to the finite verb *slept*) and conversely, sentence (5b) would incorrectly be judged grammatical (because both *want* and *try* require an infinitival complement). Similarly, in Example (5c), the church is Syrian Orthodox, not Syrian and Orthodox. Assigning a dependent rather than an independent derivation to such cases straightforwardly captures the distinction between intersective and non-intersective modifiers.

(5) a. ✓John wanted to assume that Peter slept. (Dependent derivation)
b. ⊗John wanted Peter tries to walk.
c. The meerkat admired the Syrian Orthodox church. (Dependent derivation)

Finally, some multiple adjunctions may involve both dependent and independent derivations, for example, when multiple modifiers and predicative verbs adjoin to the same verb (Example (6a)) or in the case of a derivation (Example (6b)) involving both intersective (old) and non-intersective (i.e., Syrian in Syrian Orthodox) modifiers.

(6) a. Yann said that John knows that Richard Parker and Pi wandered the Algae Island yesterday through the meerkats. (Mixed derivation)
b. The meerkat admired the old Syrian Orthodox church. (Mixed derivation)

As we shall see in Section 5.3, the parsing algorithm we propose licenses dependent, independent, and mixed derivations but is restricted to appropriately distinguish between various types of modifiers. Moreover, the feature information encoded in the grammar further restricts the derivation structures produced, thereby accounting for the interactions between adjunction, linear ordering, and morpho-syntactic constraints.

3. Multiple Adjunction in Tree Adjoining Grammars

Vijay-Shanker and Weir (1991) introduce a compilation of TAG to Linear Indexed Grammars (LIGs; Gazdar 1988) that makes the derivation process explicit. Schabes and

Shieber (1994) modify this compilation to allow for both dependent and independent derivations. The resulting LIG is further exploited to specify a parsing algorithm that recovers those derivations.

In this section, we summarize Schabes and Shieber's proposal. We start (Section 3.1) with an informal description of their approach. In Section 3.2, we introduce ordered derivation trees. Section 3.3 provides a brief introduction to LIG. Section 3.4 summarizes the TAG-to-LIG compilation proposed by Vijay-Shanker and Weir (1991). Finally, Section 3.5 describes the modifications introduced by Schabes and Shieber (1994) to allow both for dependent and for independent derivations.

3.1 Schabes and Shieber's Proposal: Motivations and Intuitions

Tree Adjoining Grammar distinguishes between two types of auxiliary trees, namely, modifier vs. predicative auxiliary trees (Joshi and Vijay-Shanker 2001). Whereas predicative trees are assigned to verbs taking a sentential argument, modifier trees are assigned to all other auxiliary trees (e.g., verbal auxiliaries, adjectives, adverbs, prepositions, and determiners). More generally, the difference between a predicative and a modifier tree is that in a predicative tree, the foot node, like the substitution nodes, corresponds to an argument node selected by its lexical anchor (i.e., the word that selects that tree) whereas in a modifier auxiliary tree, the foot node is an open slot corresponding to the phrase being modified. When associating semantic entities with tree nodes (as proposed, for example, by Joshi and Vijay-Shanker [2001] and Gardent and Kallmeyer [2003]), this difference can be seen by noting the entities associated with root and foot nodes: These are distinct in a predicative tree but identical in modifier trees.

In their approach, Schabes and Shieber specify a TAG-to-LIG conversion that systematically associates dependent derivations with predicative auxiliary trees and independent derivations with modifier auxiliary trees. In addition, they introduce two mechanisms to ensure that each derivation tree unambiguously specifies a linguistically plausible derived tree.

First, they enforce ordering constraints between modifier trees adjoining at the same node (which are thus ambiguous with respect to the derived tree they describe) by assuming that derivation trees are ordered and that linear precedence (LP) statements can be used to constrain the order of siblings in a derivation tree. For instance, given the independent derivation shown in Figure 1, an LP statement stating that β_{red} must occur before $\beta_{roasted}$ in the derivation tree will ensure that $\beta_{roasted}$ appears above β_{red} in the derived tree and therefore that the resulting derived tree yields the phrase *roasted red pepper* rather than *red roasted pepper*.

Second, when both predicative and modifier trees adjoin at the same address, predicative trees always occur above all modifier trees in the derived tree ("outermost predication"). This ensures, for instance, that under the reading where *yesterday* refers to the *arriving* rather than the *saying* (i.e., when both *say* and *yesterday* adjoin to *arrive*), Example (7a) is derived but not Example (7b).

(7) a. ✓Peter says that yesterday John arrived late.

b. ⊗Yesterday Peter says that John arrived late.

3.2 Ordered Derivation Trees

In the standard version of TAG, each derivation tree describes a unique derived tree. In the case of a dependent derivation, unicity follows from the fact that dependent

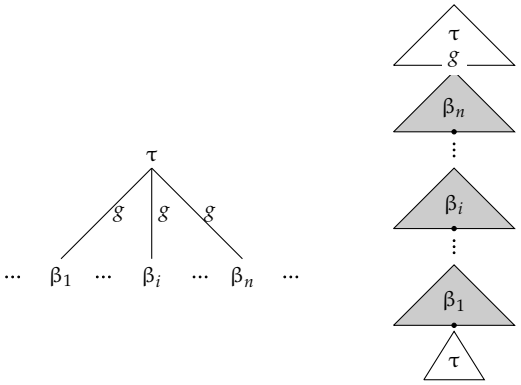


Figure 2
Ordered derivation tree and corresponding derived tree. τ, β_1, β_i, and β_n are elementary trees. β_1, β_i, and β_n are auxiliary trees that all adjoin at the address g in the elementary tree τ.

derivations specify the order in which adjunction takes place (e.g., β_2 adjoins to β_1 and the result to α). As a result, if β_2 adjoins to β_1, there is only one possible derived tree—namely, a tree where β_2 appears above β_1.

When allowing for independent derivations, however, several derived trees are possible, depending on the order in which the auxiliary trees are adjoined. To ensure a unique mapping from derivation to derived tree, Schabes and Shieber (1994) therefore introduce the notion of **ordered derivation trees**. Ordered derivation trees differ from standard TAG derivation trees in that (i) they may contain sibling edges labeled with the same address, and (ii) they specify a total order on such siblings.

Figure 2 shows an example ordered derivation tree and associated derived tree. As indicated by the shared g address on their parent edge, auxiliary trees $\beta_1, \ldots, \beta_n$ adjoin to the same node—namely, the node with address g in the elementary tree τ. Because the derivation tree is ordered, β_1 will appear below β_2 in the derived tree, which in turn will be below β_3, and so on. In short, given a set of auxiliary trees all adjoining to the same tree node, the derived tree produced from an ordered derivation tree following an independent derivation will be identical to the derived tree produced with the corresponding dependent derivation—that is, the dependent derivation where $\beta_1, \ldots, \beta_n$ appear in increasing index order from top to bottom.

3.3 Linear Indexed Grammar

Like context-free grammars, LIGs (Gazdar 1988) are string rewriting systems where strings are composed of terminals and nonterminals. In a LIG, however, nonterminal symbols may be associated with a stack of symbols, called **indices**. A LIG rule can thus be represented as follows:

$$N[..\mu] \rightarrow N_1[\mu_1] \ldots N_{i-1}[\mu_{i-1}] N_i[..\mu_i] N_{i+1}[\mu_{i+1}] \ldots N_n[\mu_n] \tag{8}$$

N and N_i are nonterminals whereas μ and μ_i are strings of stack symbols. The symbol .. stands for the remainder of the stack symbols. Note that the remainder of the stack symbols associated with the left-hand side is associated with only one of the nonterminal (namely, N_i) on the right-hand side.

LIGs have been used in the literature (Weir and Joshi 1988; Vijay-Shanker and Weir 1991) to provide a common framework for the extensions of context-free grammars. In particular, Vijay-Shanker and Weir (1991, 1993) showed a weak equivalence between LIGs, TAGs, and combinatory categorial grammars (Steedman 2000) and proposed a LIG-based polynomial-time CYK recognition algorithm for TAGs and combinatory categorical grammars. In what follows, we show how Schabes and Shieber (1994) use a LIG variant of TAGs to license both dependent and independent derivations.

3.4 TAG to LIG Compilation

The TAG-to-LIG compilation proposed by Vijay-Shanker and Weir (1991) produces LIG rules that simulate a traversal of the derived tree produced by the original TAG grammar. In these LIG rules, each node η of a TAG elementary tree is viewed as having both a top t[..η] and a bottom b[..η] component to account for the possibility of an adjunction. Figure 3 illustrates the traversal of the TAG-derived trees specified by the LIG resulting fromVijay-Shanker and Weir (1991) TAG-to-LIG compilation.

Figure 4 lists the LIG rules resulting from the TAG to LIG compilation process. Each nonterminal (t[..η] or b[..η]) with the top of the stack symbol in a LIG rule corresponds to a unique node in some elementary tree of the grammar. The inner stack symbols are used to keep track of the nodes higher in the derived tree where an auxiliary tree has been adjoined.

Rules of Types 1 and 2 capture immediate dominance between the bottom of a node η and the top of its immediate daughters in two configurations, depending on whether η dominates the foot node (Type 1) or not (Type 2). Rules of Type 3 handle nodes that require neither substitution nor adjunction. This rule handles cases where no adjunction occurs at a node by rewriting the top of this node to its bottom. Rules of Type 6 model substitution. Finally, rules of Types 4 and 5 handle adjunction. They specify that, for any given node η and any auxiliary tree β that may adjoin to η, the top of η rewrites to the top of the root node of β; and the bottom of the foot of β to the bottom of η. It follows that there can be no multiple adjunction in this LIG version of TAG.

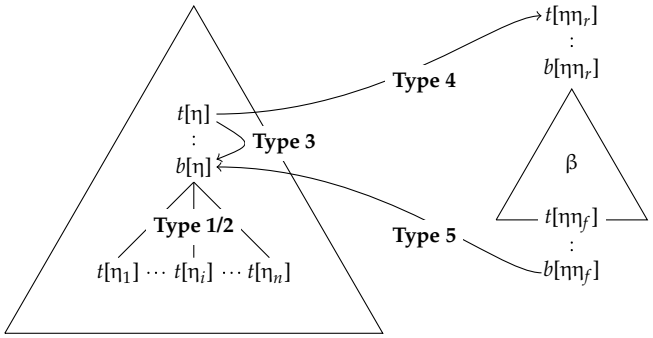


Figure 3
LIG variant of TAG for the Standard derivation. Each of the tree nodes in the grammar is assigned a unique address. For example, here η, η_1, η_i, and η_n point to the distinct nodes in the left elementary tree and η_r and η_f point to the root and the foot nodes of the shown auxiliary tree β in the grammar. t[..η] and b[..η] are the top and bottom components of the tree node η in the grammar.

Type 1: *Immediate domination dominating foot.* For each node η in the auxiliary trees that dominates the foot node and with children $\eta_1, \ldots, \eta_i, \ldots, \eta_n$, where the child η_i also dominates the foot node, the following LIG production rule is generated.

$$b[..\eta] \rightarrow t[\eta_1] \ldots t[\eta_{i-1}] t[..\eta_i] t[\eta_{i+1}] \ldots t[\eta_n]$$

Type 2: *Immediate domination not dominating foot.* For each elementary tree node η that does not dominate the foot node and with children $\eta_1, \ldots, \eta_n$, the following LIG production rule is generated.

$$b[\eta] \rightarrow t[\eta_1] \ldots t[\eta_n]$$

Type 3: *No adjunction.* For each elementary tree node η that is not marked for substitution or obligatory adjunction, the following LIG production rule is generated.

$$t[..\eta] \rightarrow b[..\eta]$$

Type 4: *Start root of adjunction.* For each elementary tree node η that allows the adjunction of the auxiliary tree with the root node η_r, the following LIG production rule is generated.

$$t[..\eta] \rightarrow t[..\eta\eta_r]$$

Type 5: *Start foot of adjunction.* For each elementary tree node η that allows the adjunction of the auxiliary tree with the foot node η_f, the following LIG production rule is generated.

$$b[..\eta\eta_f] \rightarrow b[..\eta]$$

Type 6: *Start substitution.* For each elementary tree node η that allows the substitution of the initial tree with the root node η_r, the following LIG production rule is generated (not shown in Figure 3).

$$t[\eta] \rightarrow t[\eta_r]$$

Figure 4
LIG production rules for the Standard derivation.

3.5 Modifying the TAG to LIG Compilation to Allow for Multiple Adjunctions

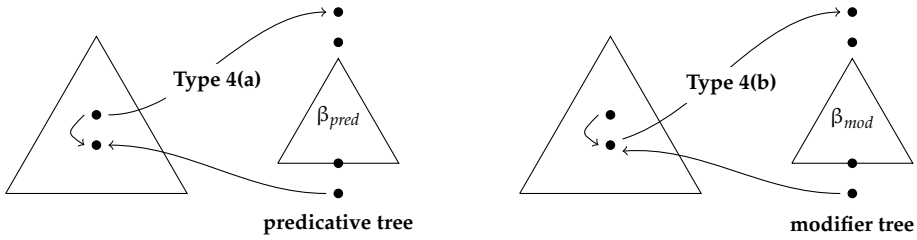
To associate predicative tree adjunctions with dependent derivations and multiple modifier adjunctions with independent derivations, Schabes and Shieber (1994) modify the compilation of TAG to LIG, proposed by Vijay-Shanker and Weir (1991) as sketched in Figure 5. Type 4(a) rules apply to adjunctions involving predicative trees. They are identical to Type 4 rules in the Vijay-Shanker and Weir's approach and therefore enforce a standard (dependent) derivation for predicative trees. In contrast, Type 4(b) rules apply to adjunctions involving modifiers and result in an independent derivation.

Note also that the Outermost Predication constraint (i.e., predicative trees always occur above modifier trees adjoined at the same node) alluded to in Section 3.1 follows from the interactions between the Type 4(a) and Type 4(b) LIG rules.

Schabes and Shieber prove the weak-generative equivalence of TAGs under both Standard and Extended derivation using the LIG compilation. They also propose a recognition and a parsing algorithm with complexity of $O(n^6)$ in the length of the string.

4. Multiple Adjunction in Feature-Based TAG

In this section, we explain why a straightforward extension of Schabes and Shieber's proposal to FB-TAG would not work and we outline the intuitions and motivations underlying our approach. Section 5 will then introduce the details of our proposal.



Type 4(a): *Start root of adjunction for predicative trees.* For each elementary tree node η that allows the adjunction of the predicative auxiliary tree with the root node η_r, the following LIG production rule is generated.

$$t[..\eta] \rightarrow t[..\eta\eta_r]$$

Type 4(b): *Start root of adjunction for modifier trees.* For each elementary tree node η that allows the adjunction of the modifier auxiliary tree with the root node η_r, the following LIG production rule is generated.

$$b[..\eta] \rightarrow t[..\eta\eta_r]$$

Figure 5
LIG variant of TAG for Schabes and Shieber's Extended derivation and associated production rules. The top and bottom components of the nodes are presented by •. Type 4(a) transitions support dependent derivations, and Type 4(b) transitions support independent derivations.

4.1 Feature-Based Tree Adjoining Grammar

We start by a brief description of FB-TAG and of the unifications performed during derivation. FB-TAG was introduced by Vijay-Shanker (1987) and Vijay-Shanker and Joshi (1988, 1991) to support the use of feature structures in TAG. Figure 6 shows a toy FB-TAG for illustration.

An FB-TAG differs from a TAG in that tree nodes are decorated with feature structures. Nonterminal and foot nodes are decorated with two feature structures called top (T) and bottom (B), and substitution nodes are decorated with a single top feature structure. During derivation, feature structure unification constrains tree combination, as illustrated in Figure 7. Substitution unifies the top feature structure of a substitution node with the top feature structure of the root node of the tree being substituted. The adjunction of an auxiliary tree β to a tree node η_o unifies the top and bottom feature structures of η_o with the top feature structure of the root node of β and the bottom feature structure of its foot node, respectively. Finally, at the end of the derivation, the top and bottom feature structures of all nodes in the derived tree are unified.

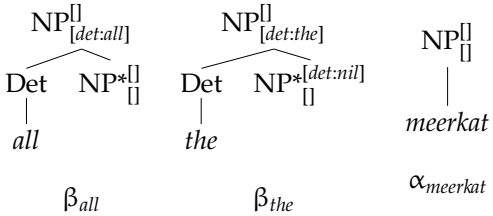


Figure 6
A toy FB-TAG. For the sake of clarity, feature structures are abbreviated.

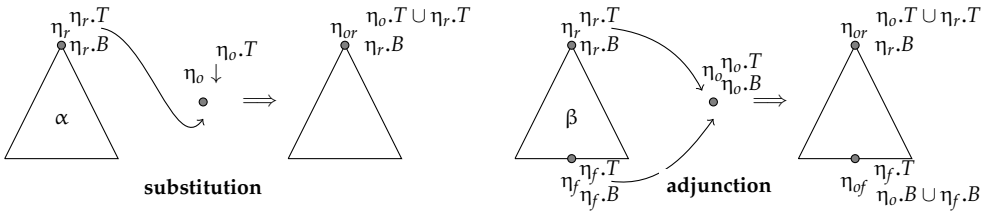


Figure 7
Feature unifications along substitution and adjunction in FB-TAG. The node η_o in some elementary tree τ is the operation site for a substitution or an adjunction. For the sake of clarity, we only show the operation node η_o.

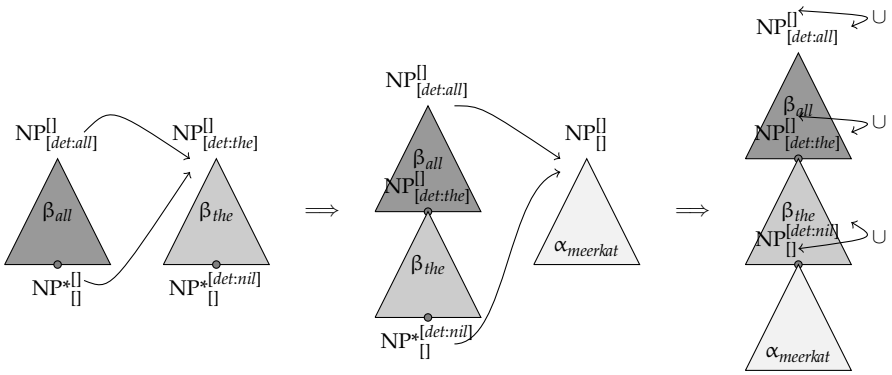


Figure 8
Standard derivation for *all the meerkats*.

Figure 8 shows the standard derivation for the phrase *all the meerkats*, using the grammar shown in Figure 6, and Figure 9 shows the corresponding derived and derivation trees. As can be seen, the feature constraints encoded in the grammar correctly ensure that *all the meerkats* can be derived (leftmost derivation tree in Figure 9) but not *the all meerkats* (rightmost in Figure 9). The incorrect derivation is blocked by the feature structure [*det* : *nil*] on the foot of the auxiliary tree β_{the}, which leads to a unification failure if β_{the} is adjoined at the root of β_{all} with bottom feature structure [*det* : *the*].

4.2 Why a Simple Extension of the LIG Framework to FB-TAG Will Not Work

To motivate our approach, we start by considering a simple extension of Schabes and Shieber's LIG framework to FB-TAG, where each LIG rule enforces unifications mimicking those applied in FB-TAG. In particular, let us assume that Type 3 rules

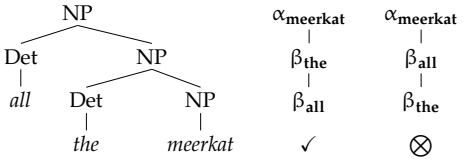


Figure 9
The derived tree (left), the successful standard derivation tree (middle), and the failed dependent derivation tree (right) for *all the meerkats*.

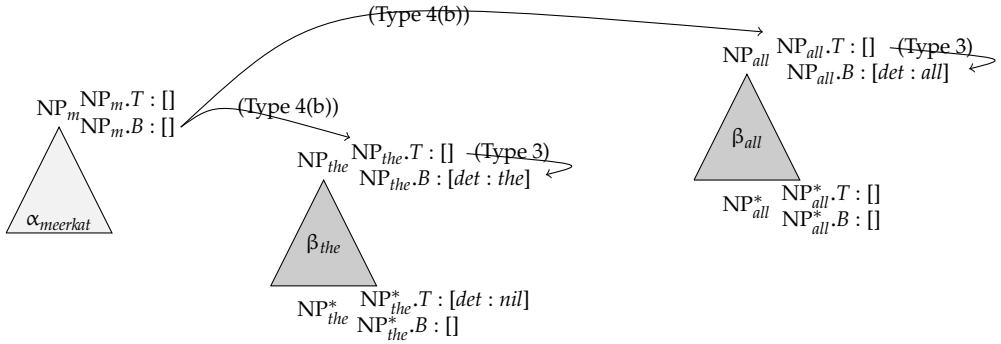


Figure 10
Failed derivation under a simple extension of the Schabes and Shieber LIG framework to FB-TAG: Type 4(b) rules unify NP$_m$.T with both NP$_{the}$.T and NP$_{all}$.T while Type 3 rules unify NP$_{the}$.T with NP$_{the}$.B and NP$_{all}$.T with NP$_{all}$.B. Hence NP$_{the}$.B and NP$_{all}$.B should unify. However, because their *det* values differ, derivation fails.

("No adjunction") unify the top and the bottom feature structures of nodes where no adjunction occurs, and Type 4(b) rules ("Start root of adjunction") unify the top feature ($\eta.T$) of the node (η) being adjoined to with the top feature structure ($\eta_r.T$) of the root node (η_r) of the auxiliary tree being adjoined:[4]

$$b[..\eta] \rightarrow t[..\eta\eta_r] \qquad \eta.T \cup \eta_r.T \qquad \text{(Type 4b)} \qquad (9)$$

$$t[..\eta] \rightarrow b[..\eta] \qquad \eta.T \cup \eta.B \qquad \text{(Type 3)} \qquad (10)$$

As shown in Figure 10, this approach can incorrectly lead to derivation failures in the case of an independent multiple adjunction. Intuitively, the reason for this is that, in Schabes and Shieber's approach, multiple adjunction starts and ends from the bottom component of the node being adjoined to. This is fine when no features are involved because the category of the node being adjoined to is always identical to the root and foot node of the auxiliary trees being adjoined. When nodes carry feature structures, however, a unification clash can occur that makes derivation fail. Thus, in our example, derivation incorrectly fails because the bottom feature structures of the root node of the auxiliary tree for *all* and the bottom feature structure of the root node of the auxiliary tree for *the* should unify but have conflicting value. As shown by the dependent derivation for *all the meerkats* depicted in Figure 8, this is incorrect.

4.3 Proposal: Intuition and Motivations

As we just saw, in the case of multiple independent adjunctions, a straightforward extension of Schabes and Shieber's LIG framework to FB-TAG fails to correctly capture the unification constraints encoded in the grammar. More generally, when extending multiple independent adjunction to FB-TAG, it is crucial that the feature constraints encoded by the linguist describe the same set of derived trees no matter

4 We associate $\eta.T \cup \eta_r.T$ with the Type 4(b) rules to mimic the adjunction in FB-TAG, as shown in Figure 7.

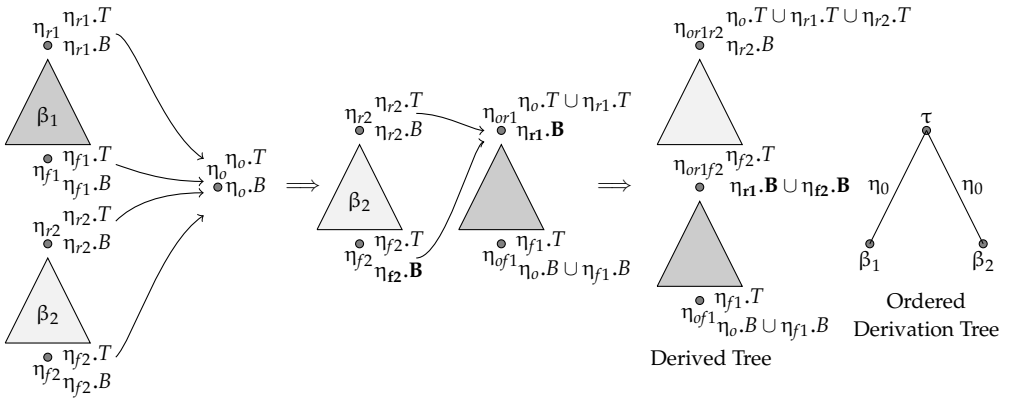


Figure 11
Independent derivation and feature unification. The unifications performed by the independent adjunction of β_1 and β_2 to η_o are the same as those that would be performed by a dependent adjunction of β_2 to β_1 and of the resulting derived tree to η_o. Crucially in the independent derivation, although both β_1 and β_2 adjoin to η_o, the adjunction of β_2 requires access to the feature structure of the root of β_1 ($\eta_{\mathbf{f2}}.\mathbf{B} \cup \eta_{\mathbf{r1}}.\mathbf{B}$).

which derivation tree is produced. We therefore propose a parsing algorithm that, given several auxiliary trees $\beta_1, \ldots, \beta_n$ adjoining at the same node η_o, performs the same unifications independently of whether the derivation is dependent, independent, or mixed dependent/independent.

Figure 11 shows the unifications resulting from the multiple adjunction of β_1 and β_2 to a single node η_o. While it depicts the unifications enforced by our parsing algorithm for the derivation tree shown on the right hand side (i.e., for the independent adjunction of β_1 and β_2 to η_o), these unifications are in fact exactly the same as those that would be enforced by a dependent adjunction of β_2 into β_1 into η_o.

One key point illustrated by Figure 11 is that whereas multiple adjunction operates on a single node (here η_o), the unification constraints of FB-TAG require that the bottom feature structure of the foot of an auxiliary tree which appears higher in the derived tree (here, β_2) unifies with the bottom feature structure of the root of the auxiliary tree appearing immediately below it in the derived tree (here β_1)—not with that of the root of the node to which it adjoins (here η_o). In other words, while a multiple adjunction on η_o operates on η_o only, a correct implementation of FB-TAG unification constraints requires keeping track of the feature structures associated with the auxiliary trees successively adjoining to η_o.

In our proposal, we capture this bookkeeping requirement by associating tree nodes not with feature structures but with reference variables pointing to feature structures. The parsing algorithm is then specified so as to support dependent, independent, and mixed derivations while enforcing the same unifications as would be performed under a dependent adjunction.

4.4 Comparison With Schabes and Shieber's Approach

Before giving the technical details of our parsing algorithm (Section 5), we first highlight some differences between our and Schabes and Shieber's approach. In particular, we show that (i) whereas Schabes and Shieber resort to three distinct mechanisms to account for word order constraints (i.e., selective adjoining constraints, linear

precedence statements on derivation trees, and a constraint on parsing), the FB-TAG approach supports a uniform treatment of word order and (ii) our approach straightforwardly accounts for mixed dependent/independent derivations that would require some additional stipulation in Schabes and Shieber's approach.

4.4.1 Ordering Constraints among Modifier Auxiliary Trees. In TAG, determiners and verbal auxiliaries are modifier rather than predicative auxiliary trees (cf. Section 3.1). Because Schabes and Shieber's definitions systematically associate modifiers with independent derivations, all examples in (11a–d) undergo an independent derivation and constraints must therefore be provided to determine the order of the sibling nodes in the resulting derivation tree.

(11) a. ✓The sonatas should have been being played by Sarah.
b. ⊗The sonatas have should been being played by Sarah.
c. ✓All the meerkats
d. ⊗The all meerkats

To specify these constraints on similar cases (soft ordering constraints on adjectives and strict ordering constraints on temporal and spatial adverbial phrases in German), Schabes and Shieber (1994) suggest the use of LP constraints on derivation tree siblings. As illustrated by the derivation of Example (11c–d) in Figures 8 and 9, in the FB-TAG approach, such additional constraints are unnecessary: They simply fall out of the feature constraints encoded in the grammar.

Note that even if determiners and auxiliary verbs were to be handled using dependent adjunction, the word ordering constraints used by Schabes and Shieber would fail to account for cases such as Example (12), where auxiliary verbs are interleaved with adverbs.

(12) John has often been selected for nomination.

In this case, if the auxiliary verbs *has* and *been* were treated as predicative trees, Schabes and Shieber's constraint that predicative trees adjoin above modifier trees would preclude the derivation of Example (12) and incorrectly predict the derived sentence to be *John has been often selected for nomination*.

4.4.2 Ordering Constraints among Predicative Trees. As discussed by Schabes and Shieber (1994), auxiliary predicative trees may impose different constraints on the type of sentential complement they accept. Thus Example (13a) is correct but not Example (13b) because *want* expects an infinitival complement (previously shown in Example (5)).

(13) a. ✓John wanted to assume that Peter slept.
b. ⊗John wanted Peter tries to walk.

Although in the Schabes and Shieber's approach, selective adjoining constraints are used to license Example (13a) and rule out Example (13b), in the FB-TAG approach, this can be achieved using feature constraints.

4.4.3 Ordering Constraints between Predicative and Modifier Auxiliary Trees. In sentences such as Example (14a) where both modifier and predicative auxiliary trees adjoin to the same address, the predicative trees should generally adjoin above any modifier trees so that the predicative verb precedes the modifier in the derived string.

(14) a. ✓John promised that Peter will leave tomorrow.
b. ⊗Tomorrow John promised that Peter will leave.

To ensure the appropriate linearization, the Schabes and Shieber's approach introduces the **outermost-predication rule**, which stipulates that predicative trees adjoin above modifier auxiliary trees. In contrast, the FB-TAG approach allows both orders and lets feature constraints rule out ungrammatical sentences such as Example (14b). This allows the approach to directly extend to a counter-example discussed by Schabes and Shieber (1994), where a modifier (here, *At what time*) must in fact adjoin above a predicative tree.

(15) At what time did Brockway say Harrison arrived?

Figure 12 shows two possible derivation trees for the sentence (15) under the interpretation where it is the time of arriving (rather than the time of saying) which is questioned. These derivation trees show the two possible relative orderings of the (predicative) auxiliary tree for *say* and the (modifier) auxiliary tree *at what time*. Because the Outermost-Predication rule requires that predicative trees adjoin above modifier trees (and thus occur outermost in the derivation tree), in Schabes and Shieber's approach, only the right-hand side derivation is possible, thus failing to derive sentence (15). In contrast, because our approach does not explicitly constrain the relative ordering of predicative and modifier auxiliary trees adjoining to the same node, both derivations are possible, thereby licensing both Example (15) and the sentence *Did Brockway say at what time Harrison arrived?*

4.4.4 Mixed Dependent and Independent Multiple Adjunctions. In Schabes and Shieber's approach, all modifier auxiliary trees undergo independent derivation. As shown in Section 2.2, however, non-intersective modifiers arguably license a dependent derivation while some cases of multiple adjunction may involve both a dependent and an independent derivation. As we shall see in Section 5, our FB-TAG approach accounts for such cases by allowing both for independent and dependent derivations, by ruling out dependent derivations for intersective modifiers and by using feature constraints to regulate the interactions between multiply adjoining auxiliary trees.

5. Extending Schabes and Shieber's LIG Framework for FB-TAGs

We now propose a compilation of FB-TAG-to-LIG that makes both dependent and independent derivations in FB-TAG explicit. We use this resulting LIG to specify an

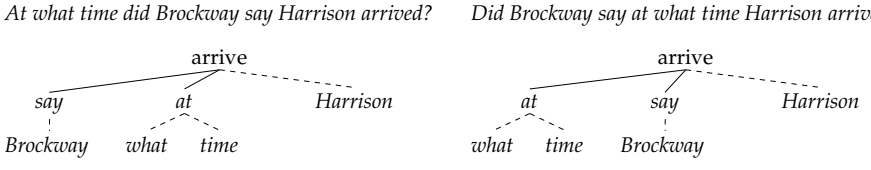


Figure 12
Ordered derivation trees for the sentence of Example (15). Dotted lines indicate substitutions and plain lines indicate adjunctions. Schabes and Shieber's Outermost Predication principle rules out a derivation tree on the left-hand side.

Earley algorithm for recovering multiple adjunctions in FB-TAG. This compilation differs in two main ways from that proposed by Schabes and Shieber (1994). First, tree nodes are associated with reference variables pointing to feature structures. Second, the LIG rules are modified and extended with unification operations.

5.1 Feature Structures and Reference Variables

To account for FB-TAG unifications while allowing for independent derivations, we replace the feature structures of FB-TAG with reference variables pointing to those feature structures. Each node in the elementary trees is decorated with two reference variables: The top reference variable P_T contains the reference to the top feature structure T and the bottom reference variable P_B contains the reference to the bottom feature structure B. The top and the bottom feature structures of a node η can be traced by $val(\eta.P_T)$ and $val(\eta.P_B)$, respectively, where P_T and P_B are the top and the bottom reference variables decorating the node η, and the function $val(P)$ returns the feature structures referred to by the reference variable P.

When specifying the parsing algorithm, we use reference variables to ensure the appropriate unifications, as follows. In an independent derivation where the node η_o is adjoined to, first by β_1 and second by β_2, the bottom feature structure $\eta_o.B$ of η_o (i) unifies with the bottom feature structure $\eta_{f1}.B$ of the foot of β_1 and (ii) is reassigned ($:=$) to the bottom reference variable $\eta_{r1}.P_B$ of the root of β_1. When β_2 is adjoined, its foot node will therefore correctly be unified, not with the bottom feature structure of η_o but with that of η_{r1}.

5.2 LIG Rules with Unification Operations

To support both dependent and independent derivations while enforcing the correct unifications, we modify the TAG-to-LIG compilation in such a way that the resulting LIG rules capture the tree traversal depicted in Figure 13. Independent derivations are accounted for by the fact that adjunction starts and ends at the bottom component of the node being adjoined to (Type 4 and 5 rules). Our LIG compilation automatically supports dependent derivations by allowing sequential adjunctions at the roots of auxiliary trees.

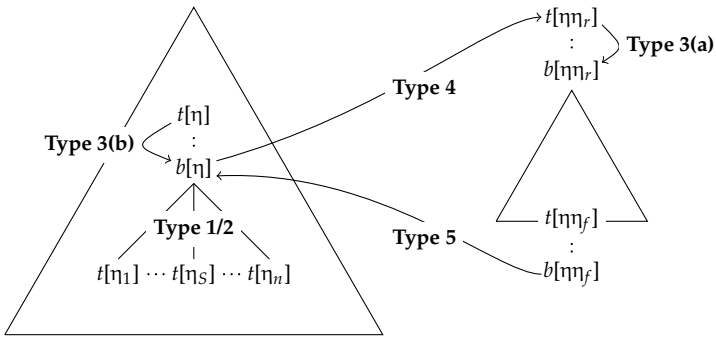


Figure 13
LIG variant of TAG for the extended derivation in FB-TAG.

Type 4: *Start root of adjunction.* For each elementary tree node η that allows the adjunction of the auxiliary tree with the root node η_r, the following LIG production rule is generated.

$$b[..\eta] \rightarrow t[..\eta\eta_r] \qquad val(\eta.P_T) \cup val(\eta_r.P_T), \eta.P_B := \eta_r.P_B$$

Type 5: *Start foot of adjunction.* For each elementary tree node η that allows the adjunction of the auxiliary tree with the foot node η_f, the following LIG production rule is generated.

$$b[..\eta\eta_f] \rightarrow b[..\eta] \qquad val(\eta.P_B) \cup val(\eta_f.P_B)$$

Type 6: *Start substitution.* For each elementary tree node η that allows the substitution of the initial tree with the root node η_r, the following LIG production rule is generated (not shown in Figure 13).

$$t[\eta] \rightarrow t[\eta_r] \qquad val(\eta.P_T) \cup val(\eta_r.P_T)$$

To perform multiple adjunction while enforcing the appropriate feature unifications (as depicted in Figure 11), we split Type 3 rules into two subtypes. Type 3(a) rules apply to the root of auxiliary trees and perform no unification. By no unification, they ensure that feature structures are not blocked for the possibility of the adjunction of the following auxiliary tree and allow for the correct unifications to be carried out for independent derivations. Type 3(b) rules function as termination of multiple adjunction by unifying the top and bottom feature structures of the node. It is applicable to all tree nodes except roots of auxiliary trees.

Type 3(a): *Terminating adjunction at the root of the auxiliary tree.* For each root node η of the auxiliary trees, the following LIG production rule is generated.

$$t[..\eta] \rightarrow b[..\eta] \qquad \emptyset$$

Type 3(b): *Terminating adjunction at any other node.* For each node η that is not a root node of some auxiliary tree and is not marked for substitution, the following LIG production rule is generated.

$$t[..\eta] \rightarrow b[..\eta] \qquad val(\eta.P_T) \cup val(\eta.P_B)$$

Given this set of rules, both dependent and independent derivations are possible. For example, given two auxiliary trees β_1 and β_2 adjoining at the node η in an elementary tree τ, a dependent derivation will occur whenever the Type 4 rule applies to predict the adjunction of, for example, β_2 at the root of β_1. Conversely, if the Type 3(a) rule applies at the root of β_1, recognition will move from the top of the root of β_1 to its bottom, allowing for Type 5 rule to complete the adjunction of β_1 at the node η; and the Type 4 rule applies to predict the adjunction of β_2 at the node η of τ, registering an independent derivation.

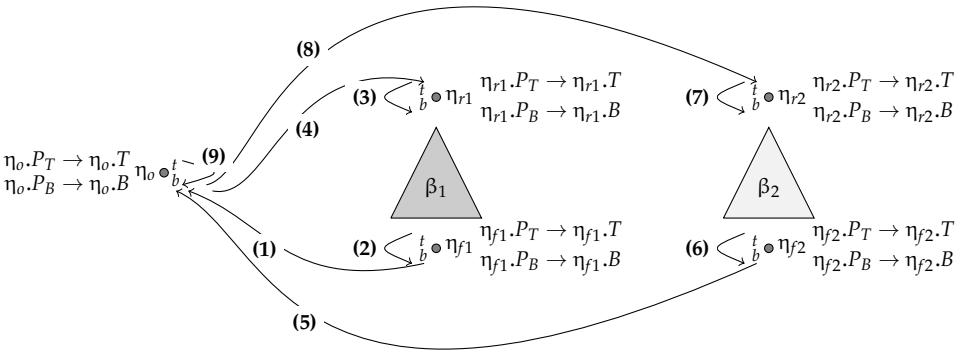
5.3 Parsing Algorithm

In this section, we present our parsing algorithm for FB-TAGs with dependent and independent derivations. We start with an informal description of how the algorithm handles the interactions between unification and independent derivations. We then go on to specify the inference rules making up the algorithm. We do this in two steps. First, we present a basic set of rules allowing for both dependent and independent derivations. Second, we show how to constrain this algorithm to minimize spurious ambiguity.

5.3.1 Independent Derivations and Feature-Structure Unification. Before specifying the parsing algorithm, we illustrate by means of an example the interplay between multiple independent adjunction and feature structure unifications.

Figure 14 displays the feature unifications and reassignment performed during the recognition process of a multiple independent adjunction. The linear ordering of the equations reflects the order of the parsing completion operations.

Given the auxiliary tree β_1 and the adjunction site η_o, the picture shows that unifying the bottom feature structure of the foot node of β_1 with the bottom feature structure of η_o (Step 1: Type 5, $\eta_o.B \cup \eta_{f1}.B$) occurs before the bottom reference variable of η_o is reassigned to the bottom feature structure of the root of β_1 (Step 4: Type 4, $\eta_o.P_B \rightarrow \eta_{r1}.B$). Also, the reassignment ensures that the follow-up adjunction of β_2 at the node η_o has access to the bottom feature of the root of the previous auxiliary tree β_1 (Step 5: Type 5, $\eta_{r1}.B \cup \eta_{f2}.B$). At the end of the adjunction (Step 9), the Type 3(b) rule ensures that the top and the bottom features of the root of the last auxiliary tree (here, β_2) adjoined are unified ($\eta_{r2}.T \cup \eta_{r2}.B$).



(1). Type 5, $\eta_o.P_B \rightarrow F_1, \eta_{f1}.P_B \rightarrow F_1, F_1 = \eta_o.\mathbf{B} \cup \eta_{f1}.\mathbf{B}$
(2). Type 3(b), $\eta_{f1}.P_T \rightarrow F_2, \eta_{f1}.P_B \rightarrow F_2, F_2 = \eta_o.B \cup \eta_{f1}.T \cup \eta_{f1}.B$
(3). Type 3(a), $\eta_{r1}.P_T \rightarrow \eta_{r1}.T, \eta_{r1}.P_B \rightarrow \eta_{r1}.B$
(4). Type 4, $\eta_o.P_T \rightarrow F_3, \eta_{r1}.P_T \rightarrow F_3, F_3 = \eta_o.T \cup \eta_{r1}.T, \eta_o.\mathbf{P_B} \rightarrow \eta_{r1}.\mathbf{B}$
(5). Type 5, $\eta_o.P_B \rightarrow F_4, \eta_{f2}.P_B \rightarrow F_4, F_4 = \eta_{r1}.B \cup \eta_{f2}.B$
(6). Type 3(b), $\eta_{f2}.P_T \rightarrow F_5, \eta_{f2}.P_B \rightarrow F_5, F_5 = \eta_{r1}.B \cup \eta_{f2}.T \cup \eta_{f2}.B$
(7). Type 3(a), $\eta_{r2}.P_T \rightarrow \eta_{r2}.T, \eta_{r2}.P_B \rightarrow \eta_{r2}.B$
(8). Type 4, $\eta_o.P_T \rightarrow F_6, \eta_{r2}.P_T \rightarrow F_6, F_6 = \eta_o.T \cup \eta_{r1}.T \cup \eta_{r2}.T, \eta_o.P_B \rightarrow \eta_{r2}.B$
(9). Type 3(b), $\eta_o.P_T \rightarrow F_7, \eta_o.P_B \rightarrow F_7, F_7 = \eta_o.T \cup \eta_{r1}.T \cup \eta_{r2}.T \cup \eta_{r2}.B$

Figure 14
Multiple independent adjunction of β_1 and β_2 to η_o. The unifications and reassignments are listed in the order in which they are performed during the recognition process.

As we shall see subsequently, this correct ordering between unification and reassignment follows from the proposed Earley algorithm. Type 4 completor rules complete the prediction triggered at the root of an auxiliary tree ("Start root of adjunction") and Type 5 completor rules complete the prediction triggered at the foot node of an auxiliary tree ("Start foot of adjunction"). Because completion operates bottom–up, it follows that Type 5 rules apply before Type 4 rules. Thus, when adjoining an auxiliary tree β_1 to a node η_o, the Type 5 completor rules, unifying the bottom feature structure of the foot node of β_1 with the bottom feature structure of the node η_o, occurs before the Type 4 completor rules, which reassign the bottom reference variable of η_o to the bottom feature structure of the root of β_1.

5.3.2 Inference Rules. The parsing algorithm for FB-TAG is a modification of the algorithm presented by Schabes and Shieber (1994). It is a chart-based parsing method based on the Earley type deduction system. Each item in the chart is of the format $\langle N[..\eta] \rightarrow \Gamma \bullet \Delta, i, j, k, l\rangle$, where N is some LIG nonterminal (i.e., t or b), and Γ and Δ present the sequences of LIG nonterminals associated with stacks of node indices. The indices i, j, k, and l are markers in the input string, showing the recognized portion:[5] The recognized item starts in position i, ends in position l, and if η dominates a foot node, the tree dominated by the foot node starts in j and ends in k. If the foot node is not dominated by the recognized nonterminal sequence Γ, the values for j and k are taken to be the dummy value '$-$'. As in Earley algorithms, the $\bullet$ separates the nonterminal sequence Γ which was parsed from the nonterminal sequence Δ yet to be parsed.

The first three types of rules (Scanner, Predictor, and Type 1/2 Completor) are identical to those introduced by Schabes and Shieber (1994) and do not involve any unification operations.

The Type 3(b) completor rule enforces top and bottom unification on all nodes that are not the root of an auxiliary tree, and the Type 3(a) completor rule prevents top and bottom unification at the root of auxiliary trees.

The Type 4 completor rule unifies the top feature structure of the root of the auxiliary tree with the top feature structure of the adjunction site. In addition, it ensures that on completion of an adjunction at node η, the bottom feature structure of η is reassigned to the bottom feature structure labeling the root of the auxiliary tree. In this way, the unifications occurring in an independent derivation will mirror those occurring in a dependent one in that any following adjunction will induce unifications as if it were happening at the root node η_r of the preceding auxiliary tree (not at η).

On completion of a foot node prediction (the tree dominated by the foot of the auxiliary tree has been recognized), the Type 5 completor rule unifies the bottom feature structure of the foot of the auxiliary tree with the bottom feature structure of the adjunction site.

Finally, the Type 6 completor unifies the top feature structure of a substitution node with the top feature structure of the root of the tree being substituted.

- *Scanner:*

$$\frac{\langle b[..\eta] \rightarrow \Gamma \bullet w\Delta, i, j, k, l\rangle}{\langle b[..\eta] \rightarrow \Gamma w \bullet \Delta, i, j, k, l+1\rangle}, \qquad w = w_{l+1}, \qquad \emptyset$$

5 The indices $\langle i, j, k, l\rangle$ have been used in previous parsing algorithms for tree-adjoining grammars (Vijay-Shankar and Joshi 1985; Schabes and Joshi 1988; Schabes 1991). They deliver the same functionality here.

If w (a terminal symbol) occurs at position l+1, the scanner rule creates a new item whose span extends to l+1.

- *Predictor:*

$$\frac{\langle N[..\eta] \rightarrow \Gamma \bullet N'[\mu]\Delta, i, j, k, l\rangle}{\langle N'[\mu] \rightarrow \bullet\Theta, l, -, -, l\rangle}, \qquad \emptyset$$

Predictor rules are produced for all types of production rules. N and N' are LIG variables taking the value t or b. Γ, Δ, and Θ are the sequences of LIG nonterminals associated with stacks of node indices. μ is a sequence of node indices.

- *Type 1 and 2 Completor:*

$$\frac{\langle b[..\eta] \rightarrow \Gamma \bullet t[\eta_1]\Delta, m, j', k', i\rangle \quad \langle t[\eta_1] \rightarrow \Theta\bullet, i, j, k, l\rangle}{\langle b[..\eta] \rightarrow \Gamma t[\eta_1] \bullet \Delta, m, j \oplus j', k \oplus k', l\rangle}, \qquad \begin{array}{l}\emptyset,\\ \eta_1 \text{ not a root node}\end{array}$$

Types 1 and 2 Completor rules permit completing Rules 1 and 2 whenever the top of a child node is fully recognized. Here, $t[\eta_1]$ has been fully recognized as the substring between i and l (i.e., $w_{i+1} \ldots w_l$). Therefore, $t[\eta_1]$ can be completed in $b[..\eta]$. If one of $t[\eta_1]$ or $b[..\eta]$ dominates the foot node of the tree, the final $b[..\eta]$ will have indices associated with the substring recognized by the foot subtree. The operation $\oplus$ is defined as follows:

$$x \oplus y = \begin{cases} x, & \text{if } y = - \\ y, & \text{if } x = - \\ x, & \text{if } x = y \\ \text{undefined}, & \text{otherwise} \end{cases}$$

- *Type 3(a) Completor:*

$$\frac{\langle t[..\eta] \rightarrow \bullet b[..\eta], i, -, -, i\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l\rangle}{\langle t[..\eta] \rightarrow b[..\eta]\bullet, i, j, k, l\rangle}, \qquad \begin{array}{l}\emptyset,\\ \eta \text{ an auxiliary tree root node}\end{array}$$

The Type 3(a) Completor rule is used to complete the prediction of an auxiliary tree rooted in η. Once the auxiliary tree dominated by $b[..\eta]$ has been recognized, the auxiliary tree itself is completely recognized. As explained earlier, there is in this case no feature unification between the top and the bottom of the root of the auxiliary tree.

- *Type 3(b) Completor:*

$$\frac{\langle t[..\eta] \rightarrow \bullet b[..\eta], i, -, -, i\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l\rangle}{\langle t[..\eta] \rightarrow b[..\eta]\bullet, i, j, k, l\rangle}, \qquad \begin{array}{l}val(\eta.P_T) \cup val(\eta.P_B),\\ \eta \text{ not an auxiliary tree root node}\end{array}$$

The Type 3(b) Completor rule ensures the unification of the top and bottom feature structures for all nodes that are not the root node of an auxiliary tree.

- *Type 4 Completor:*

$$\frac{\begin{array}{c}\langle b[..\eta] \rightarrow \bullet t[..\eta\eta_r], i, -, -, i\rangle \\ \langle t[..\eta\eta_r] \rightarrow \Theta\bullet, i, j, k, l\rangle \\ \langle b[..\eta] \rightarrow \Delta\bullet, j, p, q, k\rangle\end{array}}{\langle b[..\eta] \rightarrow t[..\eta\eta_r]\bullet, i, p, q, l\rangle}, \qquad \begin{array}{c} val(\eta.P_T) \cup val(\eta_r.P_T) \\ \eta.P_B := \eta_r.P_B \end{array}$$

The auxiliary tree associated with the predicted adjunction ($t[..\eta\eta_r]$) at the node η and the subtree dominated by the node η (below $b[..\eta]$) are completed, hence $b[..\eta]$ can be completely recognized with this adjunction. The associated feature unification unifies the content of the top reference variable of the adjoining node site η with the content of the top reference variable of the root node η_r of the adjoined auxiliary tree. After the successful adjunction of this adjoining tree, the bottom reference variable of the adjoining node site η is reassigned to the content of the bottom reference variable of the root node η_r of the adjoined auxiliary tree.

- *Type 5 Completor:*

$$\frac{\langle b[..\eta\eta_f] \rightarrow \bullet b[..\eta], i, -, -, i\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l\rangle}{\langle b[..\eta\eta_f] \rightarrow b[..\eta]\bullet, i, i, l, l\rangle}, \qquad val(\eta.P_B) \cup val(\eta_f.P_B)$$

The foot node prediction can be completed when the adjunction has been performed and the bottom part of the adjoining node site η has been recognized. The associated feature unification unifies the content of the bottom reference variable of the adjoining node site η with the content of the bottom reference variable of the foot node η_f of the auxiliary tree being adjoined.

- *Type 6 Completor:*

$$\frac{\langle t[\eta] \rightarrow \bullet t[\eta_r], i, -, -, i\rangle \quad \langle t[\eta_r] \rightarrow \Theta\bullet, i, -, -, l\rangle}{\langle t[\eta] \rightarrow t[\eta_r]\bullet, i, -, -, l\rangle}, \qquad val(\eta.P_T) \cup val(\eta_r.P_T)$$

The Type 6 Completor rule completes the substitution at the node η. The associated feature unification unifies the content of the top reference variable of the node η with the content of the top reference variable of the root node η_r of the initial tree.

Given these inference rules, the recognition process is initialized using axioms of the form $\langle t[\eta_s] \rightarrow \bullet\Gamma, 0, -, -, 0\rangle$ for each rule $t[\eta_s] \rightarrow \Gamma$ where η_s is the root node of an initial tree labeled with the start symbol. Given an input string $w_1 \ldots w_n$ to be recognized, the goal items in the chart are of the form $\langle S \rightarrow t[\eta_s]\bullet, 0, -, -, n\rangle$. Once at least one goal item is found in the chart, the recognition process succeeds and the string is successfully accepted by the grammar; otherwise it is rejected. We refer the reader to Appendix A (cf. Figure A.2) for a detailed example of the recognition of the sentence *all the meerkats* using the proposed inference system.

Note also that although the recognition algorithm we described uses unreduced rules (i.e., generated grammar rules maintaining the full information of nonterminals and the associated index stacks), it is possible to define a more efficient algorithm by having reduced LIG rules and chart items listing only the single top stack element for each constituent (Vijay-Shanker and Weir 1991, 1993). The resulting recognition

algorithm is still complete because the proposed TAG-to-LIG compilation maintains a one-to-one correspondence between the generated rules and their reduced forms (Schabes and Shieber 1994).

As mentioned by Schabes and Shieber (1994), this recognition algorithm can be turned into a parsing algorithm by associating a set of operations with each chart item to build up associated derived trees.

Note also that the derivation trees built as a side effect of the parsing process are the (dependent and/or independent) derivation trees of an FB-LTAG and are therefore context-free.

5.3.3 Handling Spurious Parses. As explained at the end of Section 5.2, the parsing algorithm presented in the previous section systematically allows for dependent and independent adjunction. For example, the recognition of the sentence *all the meerkats* (Figure A.2) produces both dependent and independent derivations that are not rejected by the unification constraints. In Section 2.2, however, we argued that different types of auxiliary trees license different types of derivations. To capture these distinctions, we modify the recognition algorithm so that it associates scopal auxiliary trees (Example (16a–b)) with dependent derivations only and multiple intersective modifier auxiliary trees (Example (16c)) with only an independent derivation.

(16) a. John thinks that Peter said that the meerkat left.
b. The meerkat admired the Syrian orthodox church.
c. The tall black meerkat slept.

To block dependent adjunctions between intersective modifiers, we modify the TAG-to-LIG transformation so that, given two intersective modifier trees β_1 and β_2, no Type 4 or Type 5 rule is produced.

Type 4: *Start root of adjunction.* For each elementary tree node η in tree β_1 that allows the adjunction of the auxiliary tree β_2 with root node η_r, the following LIG production rule is generated *if and only if* β_1 *and* β_2 *are not intersective modifier auxiliary trees*.

$$b[..\eta] \rightarrow t[..\eta\eta_r] \qquad val(\eta.P_T) \cup val(\eta_r.P_T)$$

Type 5: *Start foot of adjunction.* For each elementary tree node η that allows the adjunction of the auxiliary tree β_2 with the foot node η_f, the following LIG production rule is generated *if and only if* β_1 *and* β_2 *are not intersective modifier auxiliary trees*.

$$b[..\eta\eta_f] \rightarrow b[..\eta] \qquad val(\eta.P_B) \cup val(\eta_f.P_B)$$

Thus, for instance, in the derivation of *all the meerkats* depicted in Appendix Figure A.2, the following rules will not be produced, thereby blocking the production of the dependent derivation.

$$\left.\begin{array}{l} b[..\,\mathrm{NP}_r^{the}] \rightarrow t[..\,\mathrm{NP}_r^{the}\,\mathrm{NP}_r^{all}] \\ b[..\,\mathrm{NP}_r^{all}] \rightarrow t[..\,\mathrm{NP}_r^{all}\,\mathrm{NP}_r^{the}] \end{array}\right\} Type4 \qquad \left.\begin{array}{l} b[..\,\mathrm{NP}_r^{all}\,\mathrm{NP}_f^{the}] \rightarrow b[..\,\mathrm{NP}_r^{all}] \\ b[..\,\mathrm{NP}_r^{the}\,\mathrm{NP}_f^{all}] \rightarrow b[..\,\mathrm{NP}_r^{the}] \end{array}\right\} Type5$$

Similarly, to block independent adjunctions between scopal auxiliary trees, we add a flag *scopal?* to states in the parsing algorithm. The Type 4 Completor rules associated

with scopal modifiers are modified to mark the progress of a scopal adjunction and to block the independent adjunction of another scopal modifier at the same node.

Type 4 Completor:

$$\frac{\begin{array}{c}\langle b[..\eta] \rightarrow \bullet t[..\eta\eta_r], i, -, -, i, scopal?\rangle \\ \langle t[..\eta\eta_r] \rightarrow \Theta\bullet, i, j, k, l, scopal?\rangle \\ \langle b[..\eta] \rightarrow \Delta\bullet, j, p, q, k, scopal?\rangle\end{array}}{\langle b[..\eta] \rightarrow t[..\eta\eta_r]\bullet, i, p, q, l, True\rangle}, \quad \begin{array}{c} val(\eta.P_T) \cup val(\eta_r.P_T) \\ \eta.P_B := \eta_r.P_B \end{array}$$

In the Type 4 Completor rule, once a scopal auxiliary tree β with root node η_r adjoins at some node η, the bottom component of the node η is marked with *True*, recording that a scopal adjunction has occurred at node η and that it therefore should not accept any further scopal adjunction.

Thus, for instance, the derivation of *Syrian orthodox churches* will proceed in a similar manner as the derivation of *all the meerkats* depicted in Appendix Figure A.2, but it will fail to produce the chart items $(40, 42, \ldots, 52)$ associated with the independent adjunction. Therefore, only the dependent derivation will be produced.

Note that this modification does not block modifier adjunction above a predicative adjunction. Therefore, it successfully recognizes the sentence *At what time did Brockway say Harrison arrived?*, shown in Example (15), where a *wh*-modifier needs to be adjoined above a predicative adjunction. Figure 15 shows the complete recognition algorithm modified to rule out spurious parses in the case of multiple scopal auxiliary trees and intersective modifier auxiliary trees.

5.3.4 Weak Generative Equivalence. The weak-generative equivalence refers to the set of strings characterized by the formal system. In contrast, the strong-generative equivalence relates to the set of structural descriptions (such as derivation trees, dags, proof trees, etc.) assigned by a formal system to the strings that it specifies (Vijay-Shankar and Joshi 1985; Joshi 2000).

Using an argument similar to that put forward by Schabes and Shieber (1994), we can prove the weak-generative equivalence of TAGs under the dependent and our independent derivations. We call the set of languages generated by the standard derivation in TAG, TAL_{std}; the set of languages generated by Schabes and Shieber's extended derivation in TAG, $TAL_{ext_{ss}}$; the set of languages generated with our modifications for FB-TAG, TAL_{ext}; and the set of languages generated by the LIG, *LIL*. Our derivation allows both dependent and independent derivations; therefore, our derivation will recognize all the strings recognized by the standard (dependent) derivation. More precisely, our derivation can mimic the standard derivation by not allowing more than one adjunction on a tree node by treating all auxiliary trees as scopal auxiliary trees (cf. Section 5.3.3), henceforth, $TAL_{std} \subseteq TAL_{ext}$. The proposed compilation from TAGs to LIGs for the independent derivation concluded $TAL_{ext} \subseteq LIL$. Finally, $LIL \subseteq TAL_{std}$ has been proven by Vijay-Shanker (1987). Combining these three inclusions, we can conclude that $TAL_{std} = TAL_{ext}$. In addition, Schabes and Shieber (1994) have shown that $TAL_{std} = TAL_{ext_{ss}}$. Hence, we can conclude the weak-generative equivalence of all three derivations in TAGs, $TAL_{std} = TAL_{ext_{ss}} = TAL_{ext}$. Feature structures enhance TAGs' descriptive ability without affecting their generative capacity (Vijay-Shanker and Joshi 1988). The proposed algorithm simulates the established unification mechanism in FB-TAG without affecting the representation and the stipulations (e.g., null adjunction at

- *Scanner:*

$$\frac{\langle b[..\eta] \rightarrow \Gamma \bullet w\Delta, i, j, k, l, S?\rangle}{\langle b[..\eta] \rightarrow \Gamma w \bullet \Delta, i, j, k, l+1, S?\rangle}, \qquad w = w_{l+1}, \qquad \emptyset$$

- *Predictor:*

$$\frac{\langle P[..\eta] \rightarrow \Gamma \bullet P'[\mu]\Delta, i, j, k, l, -\rangle}{\langle P'[\mu] \rightarrow \bullet\Theta, l, -, -, l, S?\rangle}, \qquad \emptyset$$

- *Type 1 and 2 Completor:*

$$\frac{\langle b[..\eta] \rightarrow \Gamma \bullet t[\eta_1]\Delta, m, j', k', i, S?\rangle \quad \langle t[\eta_1] \rightarrow \Theta\bullet, i, j, k, l, -\rangle}{\langle b[..\eta] \rightarrow \Gamma t[\eta_1] \bullet \Delta, m, j \oplus j', k \oplus k', l, S?\rangle}, \qquad \begin{array}{l}\emptyset,\\ \eta_1 \text{ not a root node}\end{array}$$

- *Type 3(a) Completor:*

$$\frac{\langle t[..\eta] \rightarrow \bullet b[..\eta], i, -, -, i, -\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l, S?\rangle}{\langle t[..\eta] \rightarrow b[..\eta]\bullet, i, j, k, l, S?\rangle}, \qquad \begin{array}{l}\emptyset,\\ \eta \text{ an auxiliary tree root node}\end{array}$$

- *Type 3(b) Completor:*

$$\frac{\langle t[..\eta] \rightarrow \bullet b[..\eta], i, -, -, i, -\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l, S?\rangle}{\langle t[..\eta] \rightarrow b[..\eta]\bullet, i, j, k, l, S?\rangle}, \qquad \begin{array}{l}val(\eta.P_T) \cup val(\eta.P_B),\\ \eta \text{ not an auxiliary tree root node}\end{array}$$

- *Type 4 Completor:*

$$\frac{\begin{array}{r}\langle b[..\eta] \rightarrow \bullet t[..\eta\eta_r^{scopal}], i, -, -, i, -\rangle\\ \langle t[..\eta\eta_r^{scopal}] \rightarrow \Theta\bullet, i, j, k, l, -\rangle\\ \langle b[..\eta] \rightarrow \Delta\bullet, j, p, q, k, S?\rangle\end{array}}{\langle b[..\eta] \rightarrow t[..\eta\eta_r^{scopal}]\bullet, i, p, q, l, True\rangle}, \qquad \begin{array}{l}val(\eta.P_T) \cup val(\eta_r^{scopal}.P_T)\\ \eta.P_B := \eta_r^{scopal}.P_{B_r}\end{array}$$

$$\frac{\begin{array}{r}\langle b[..\eta] \rightarrow \bullet t[..\eta\eta_r^{others}], i, -, -, i, -\rangle\\ \langle t[..\eta\eta_r^{others}] \rightarrow \Theta\bullet, i, j, k, l, -\rangle\\ \langle b[..\eta] \rightarrow \Delta\bullet, j, p, q, k, S?\rangle\end{array}}{\langle b[..\eta] \rightarrow t[..\eta\eta_r^{others}]\bullet, i, p, q, l, S?\rangle}, \qquad \begin{array}{l}val(\eta.P_T) \cup val(\eta_r^{others}.P_T)\\ \eta.P_B := \eta_r^{others}.P_{B_r}\end{array}$$

- *Type 5 Completor:*

$$\frac{\langle b[..\eta\eta_f^{scopal}] \rightarrow \bullet b[..\eta], i, -, -, i, S?\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l, S1?\rangle}{\langle b[..\eta\eta_f^{scopal}] \rightarrow b[..\eta]\bullet, i, i, l, l, S?\rangle}, \qquad \begin{array}{l}val(\eta.P_B) \cup val(\eta_f^{scopal}.P_{B_f})\\ S1? \neq True\end{array}$$

$$\frac{\langle b[..\eta\eta_f^{others}] \rightarrow \bullet b[..\eta], i, -, -, i, S?\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l, -\rangle}{\langle b[..\eta\eta_f^{others}] \rightarrow b[..\eta]\bullet, i, i, l, l, S?\rangle}, \qquad val(\eta.P_B) \cup val(\eta_f^{others}.P_{B_f})$$

- *Type 6 Completor:*

$$\frac{\langle t[\eta] \rightarrow \bullet t[\eta_r], i, -, -, i, S?\rangle \quad \langle t[\eta_r] \rightarrow \Theta\bullet, i, -, -, l, -\rangle}{\langle t[\eta] \rightarrow t[\eta_r]\bullet, i, -, -, l, S?\rangle}, \qquad val(\eta.P_T) \cup val(\eta_r.P_{T_r})$$

Figure 15
Recognition algorithm taking into account spurious parses.

the foot node and the bounded feature structures) of the grammar itself. Therefore, the association with feature structures will not affect this equivalence.

6. Conclusion

Although independent derivations have been shown by Schabes and Shieber (1994) to be essential for correctly supporting syntactic analysis, semantic interpretation, and statistical language modeling, the parsing algorithm they propose is restricted to TAG and is therefore not directly applicable to large scale implemented Feature-Based TAGs. We have provided a recognition algorithm for FB-TAGs that supports both dependent and independent derivations under certain restrictions enforced jointly by feature constraints and by side conditions in the parsing algorithm. The resulting algorithm combines the benefits of independent derivations with those of Feature-Based Grammars. In particular, we showed that it accounts for a range of interactions between dependent vs. independent derivation on the one hand, and syntactic constraints, linear ordering, and scopal vs. nonscopal semantic dependencies on the other hand.

Appendix A: Recognition of the String *all the meerkats*

We show the recognition of the string *all the meerkats* as per the inference rules described in Section 5.3.2.

Figure A.1 shows the grammar used for the derivation. To support multiple adjunction in FB-TAG, it implements two main modifications. First, to facilitate the TAG to LIG compilation, each tree node in the grammar is marked with a unique identifier. For example, in Figure A.1, NP^{mk}, NP_r^{the}, Det^{the}, $\mathrm{NP}_f^{the}*$, NP_r^{all}, Det^{all}, and $\mathrm{NP}_f^{the}*$ are unique node identifiers in the grammar. Second, to implement the reassignment mechanism in

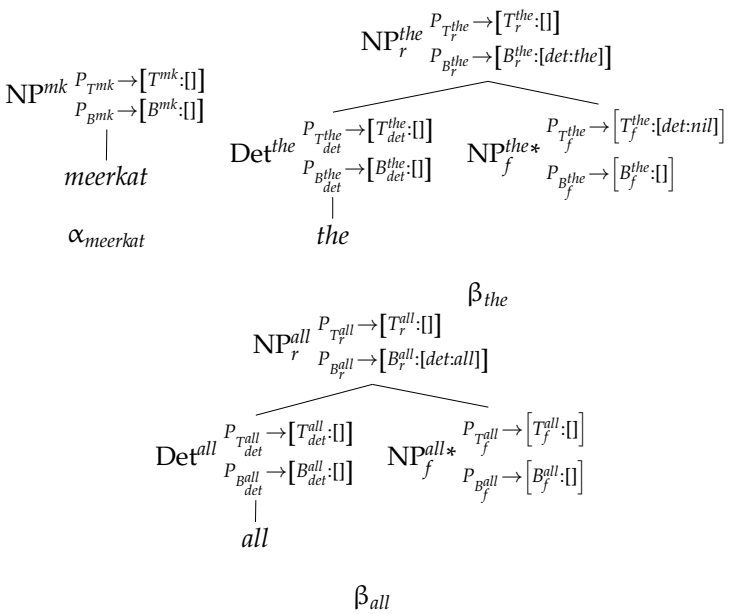


Figure A.1
A feature-based Tree-Adjoining Grammar.

Type 1 and Type 2 production rules

$$b[\text{NP}^{mk}] \rightarrow \text{meerkat}$$

$$b[..\ \text{NP}_r^{the}] \rightarrow t[\text{Det}^{the}] \quad t[..\ \text{NP}_f^{the}]$$

$$b[\text{Det}^{the}] \rightarrow \text{the}$$

$$b[..\ \text{NP}_r^{all}] \rightarrow t[\text{Det}^{all}] \quad t[..\ \text{NP}_f^{all}]$$

$$b[\text{Det}^{all}] \rightarrow \text{all}$$

Type 3(a) production rules

$$t[..\ \text{NP}_r^{the}] \rightarrow b[..\ \text{NP}_r^{the}]$$

$$t[..\ \text{NP}_r^{all}] \rightarrow b[..\ \text{NP}_r^{all}]$$

Type 3(b) production rules

$$t[..\ \text{NP}^{mk}] \rightarrow b[..\ \text{NP}^{mk}]$$

$$t[..\ \text{Det}^{the}] \rightarrow b[..\ \text{Det}^{the}]$$

$$t[..\ \text{NP}_f^{the}] \rightarrow b[..\ \text{NP}_f^{the}]$$

Type 3(b) production rules (continued)

$$t[..\ \text{Det}^{all}] \rightarrow b[..\ \text{Det}^{all}]$$

$$t[..\ \text{NP}_f^{all}] \rightarrow b[..\ \text{NP}_f^{all}]$$

Type 4 production rules

$$b[..\ \text{NP}^{mk}] \rightarrow t[..\ \text{NP}^{mk}\ \text{NP}_r^{the}]$$

$$b[..\ \text{NP}^{mk}] \rightarrow t[..\ \text{NP}^{mk}\ \text{NP}_r^{all}]$$

$$b[..\ \text{NP}_r^{the}] \rightarrow t[..\ \text{NP}_r^{the}\ \text{NP}_r^{all}]$$

$$b[..\ \text{NP}_r^{all}] \rightarrow t[..\ \text{NP}_r^{all}\ \text{NP}_r^{the}]$$

Type 5 production rules

$$b[..\ \text{NP}^{mk}\ \text{NP}_f^{the}] \rightarrow b[..\ \text{NP}^{mk}]$$

$$b[..\ \text{NP}_r^{all}\ \text{NP}_f^{the}] \rightarrow b[..\ \text{NP}_r^{all}]$$

$$b[..\ \text{NP}^{mk}\ \text{NP}_f^{all}] \rightarrow b[..\ \text{NP}^{mk}]$$

$$b[..\ \text{NP}_r^{the}\ \text{NP}_f^{all}] \rightarrow b[..\ \text{NP}_r^{the}]$$

Figure A.2
LIG production rules for the TAG shown in Figure A.1.

the parsing algorithm, the top (T) and the bottom (B) feature structures of each node are assigned reference variables P_T and P_B, respectively.

Figure A.2 shows the LIG production rules for the FB-TAG shown in Figure A.1.

Table A.1 shows the step-wise recognition of the string *all the meerkats*. Recall Section 5.3.2: The LIG rules shown in Figure A.1 does not deal with spurious parses and produces all valid derivations, dependent or independent, that are not blocked by feature unification constraints. Hence, for the string *all the meerkats*, it generates both independent (Step 52) and dependent (Step 53) derivations. As explained in Figures A.3 and A.4, both derivations undergo the identical set of feature unifications. In both figures, prediction rules are abbreviated because they do not enforce any feature unification.

Table A.1
Recognition of the string $_0$ *all* $_1$ *the* $_2$ *meerkats* $_3$ in FB-TAG.

#	Chart Items	Description
1.	$\langle S \rightarrow \bullet t[\text{NP}^{mk}], 0, -, -, 0\rangle$	axiom
2.	$\langle t[\text{NP}^{mk}] \rightarrow \bullet b[\text{NP}^{mk}], 0, -, -, 0\rangle$	3(b)-pred, 1
3.	$\langle b[\text{NP}^{mk}] \rightarrow \bullet t[\text{NP}^{mk}\ \text{NP}_r^{the}], 0, -, -, 0\rangle$	4-pred, 2
4.	$\langle b[\text{NP}^{mk}] \rightarrow \bullet t[\text{NP}^{mk}\ \text{NP}_r^{all}], 0, -, -, 0\rangle$	4-pred, 2
5.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_r^{the}], 0, -, -, 0\rangle$	3(a)-pred, 3
6.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{all}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_r^{all}], 0, -, -, 0\rangle$	3(a)-pred, 4
7.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow \bullet t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}], 0, -, -, 0\rangle$	4-pred, 5
8.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{all}] \rightarrow \bullet t[\text{Det}^{all}]\ t[\text{NP}^{mk}\ \text{NP}_f^{all}], 0, -, -, 0\rangle$	1/2-pred, 6
9.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_r^{the'}\ \text{NP}_r^{all}], 0, -, -, 0\rangle$	3(a)-pred, 7
10.	$\langle t[\text{Det}^{all}] \rightarrow \bullet b[\text{Det}^{all}], 0, -, -, 0\rangle$	3(b)-pred, 8

Table A.1
(*continued*)

#	Chart Items	Description
11.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}] \rightarrow \bullet t[\text{Det}^{all}]\ t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}], 0, -, -, 0\rangle$	1/2-pred, 9
12.	$\langle b[\text{Det}^{all}] \rightarrow \bullet\text{all}, 0, -, -, 0\rangle$	1/2-pred, 10
13.	$\langle b[\text{Det}^{all}] \rightarrow \text{all}\bullet, 0, -, -, 1\rangle$	scan, 12
14.	$\langle t[\text{Det}^{all}] \rightarrow b[\text{Det}^{all}]\bullet, 0, -, -, 1\rangle$	3(b)-comp, (10, 13)
15.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{all}] \rightarrow t[\text{Det}^{all}]\ \bullet\ t[\text{NP}^{mk}\ \text{NP}_f^{all}], 0, -, -, 1\rangle$	1/2-comp, (8, 14)
16.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}] \rightarrow t[\text{Det}^{all}]\ \bullet\ t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}], 0, -, -, 1\rangle$	1/2-comp, (11, 14)
17.	$\langle t[\text{NP}^{mk}\ \text{NP}_f^{all}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_f^{all}], 1, -, -, 1\rangle$	3(b)-pred, 15
18.	$\langle t[\text{NP}^{mk}\ \text{NP}_f^{the}\ \text{NP}_f^{all}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}], 1, -, -, 1\rangle$	3(b)-pred, 16
19.	$\langle b[\text{NP}^{mk}\ \text{NP}_f^{all}] \rightarrow \bullet b[\text{NP}^{mk}], 1, -, -, 1\rangle$	5-pred, 17
20.	$\langle b[\text{NP}^{mk}\ \text{NP}_f^{the}\ \text{NP}_f^{all}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_r^{the}], 1, -, -, 1\rangle$	5-pred, 18
21.	$\langle b[\text{NP}^{mk}] \rightarrow \bullet t[\text{NP}^{mk}\ \text{NP}_r^{the}], 1, -, -, 1\rangle$	4-pred, 19
22.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow \bullet t[\text{Det}^{the}]\ t[\text{NP}^{mk}\ \text{NP}_f^{the}], 1, -, -, 1\rangle$	1/2-pred, 20
23.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_r^{the}], 1, -, -, 1\rangle$	3(a)-pred, 21
24.	$\langle t[\text{Det}^{the}] \rightarrow \bullet b[\text{Det}^{the}], 1, -, -, 1\rangle$	3(b)-pred, 22
25.	$\langle b[\text{Det}^{all}] \rightarrow \bullet\text{the}, 1, -, -, 1\rangle$	1/2-pred, 24
26.	$\langle b[\text{Det}^{the}] \rightarrow \text{the}\bullet, 1, -, -, 2\rangle$	scan, 25
27.	$\langle t[\text{Det}^{the}] \rightarrow b[\text{Det}^{the}]\bullet, 1, -, -, 2\rangle$	3(b)-comp, (24, 26)
28.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow t[\text{Det}^{the}]\ \bullet\ t[\text{NP}^{mk}\ \text{NP}_f^{the}], 1, -, -, 2\rangle$	1/2-comp, (22, 27)
29.	$\langle t[\text{NP}^{mk}\ \text{NP}_f^{the}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_f^{the}], 2, -, -, 2\rangle$	3(b)-pred, 28
30.	$\langle b[\text{NP}^{mk}\ \text{NP}_f^{the}] \rightarrow \bullet b[\text{NP}^{mk}], 2, -, -, 2\rangle$	5-pred, 29
31.	$\langle b[\text{NP}^{mk}] \rightarrow \bullet\text{meerkat}, 2, -, -, 2\rangle$	1/2-pred, 30
32.	$\langle b[\text{NP}^{mk}] \rightarrow \text{meerkat}\bullet, 2, -, -, 3\rangle$	scan, 31
33.	$\langle b[\text{NP}^{mk}\ \text{NP}_f^{the}] \rightarrow b[\text{NP}^{mk}]\bullet, 2, 2, 3, 3\rangle$	5-comp, (30, 32)
34.	$\langle t[\text{NP}^{mk}\ \text{NP}_f^{the}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_f^{the}]\bullet, 2, 2, 3, 3\rangle$	3(b)-comp, (29, 33)
35.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow t[\text{Det}^{the}]\ t[\text{NP}^{mk}\ \text{NP}_f^{the}]\bullet, 1, 2, 3, 3\rangle$	1/2-comp, (28, 34)
36.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{the}]\bullet, 1, 2, 3, 3\rangle$	3(a)-comp, (23, 35)
37.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{the}]\bullet, 1, 1, 3, 3\rangle$	5-comp, (20, 35)
38.	$\langle b[\text{NP}^{mk}] \rightarrow t[\text{NP}^{mk}\ \text{NP}_r^{the}]\bullet, 1, -, -, 3\rangle$	4-comp, (21, 36, 32)
39.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}]\bullet, 1, 1, 3, 3\rangle$	3(b)-comp, (18, 37)
40.	$\langle b[\text{NP}^{mk}\ \text{NP}_f^{all}] \rightarrow b[\text{NP}^{mk}]\bullet, 1, 1, 3, 3\rangle$	5-comp, (19, 38)
41.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}] \rightarrow t[\text{Det}^{all}]\ t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}]\bullet, 0, 1, 3, 3\rangle$	1/2-comp, (16, 39)
42.	$\langle t[\text{NP}^{mk}\ \text{NP}_f^{all}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_f^{all}]\bullet, 1, 1, 3, 3\rangle$	3(b)-comp, (17, 40)
43.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}]\bullet, 0, 1, 3, 3\rangle$	3(a)-comp, (9, 41)
44.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{all}] \rightarrow t[\text{Det}^{all}]\ t[\text{NP}^{mk}\ \text{NP}_f^{all}]\bullet, 0, 1, 3, 3\rangle$	1/2-comp, (15, 42)
45.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}]\bullet, 0, 2, 3, 3\rangle$	4-comp, (7, 43, 35)
46.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{all}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{all}]\bullet, 0, 1, 3, 3\rangle$	3(a)-comp, (6, 44)
47.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{the}]\bullet, 0, 2, 3, 3\rangle$	3(a)-comp, (5, 45)
48.	$\langle b[\text{NP}^{mk}] \rightarrow t[\text{NP}^{mk}\ \text{NP}_r^{all}]\bullet, 0, -, -, 3\rangle$	4-comp, (4, 46, 38)
49.	$\langle b[\text{NP}^{mk}] \rightarrow t[\text{NP}^{mk}\ \text{NP}_r^{the}]\bullet, 0, -, -, 3\rangle$	4-comp, (3, 47, 32)
50.	$\langle t[\text{NP}^{mk}] \rightarrow b[\text{NP}^{mk}]\bullet, 0, -, -, 3\rangle$	3(b)-comp, (2, 48)
51.	$\langle t[\text{NP}^{mk}] \rightarrow b[\text{NP}^{mk}]\bullet, 0, -, -, 3\rangle$	3(b)-comp, (2, 49)
52.	$\langle S \rightarrow t[\text{NP}^{mk}]\bullet, 0, -, -, 3\rangle$	axiom-comp, (1, 50)
53.	$\langle S \rightarrow t[\text{NP}^{mk}]\bullet, 0, -, -, 3\rangle$	axiom-comp, (1, 51)

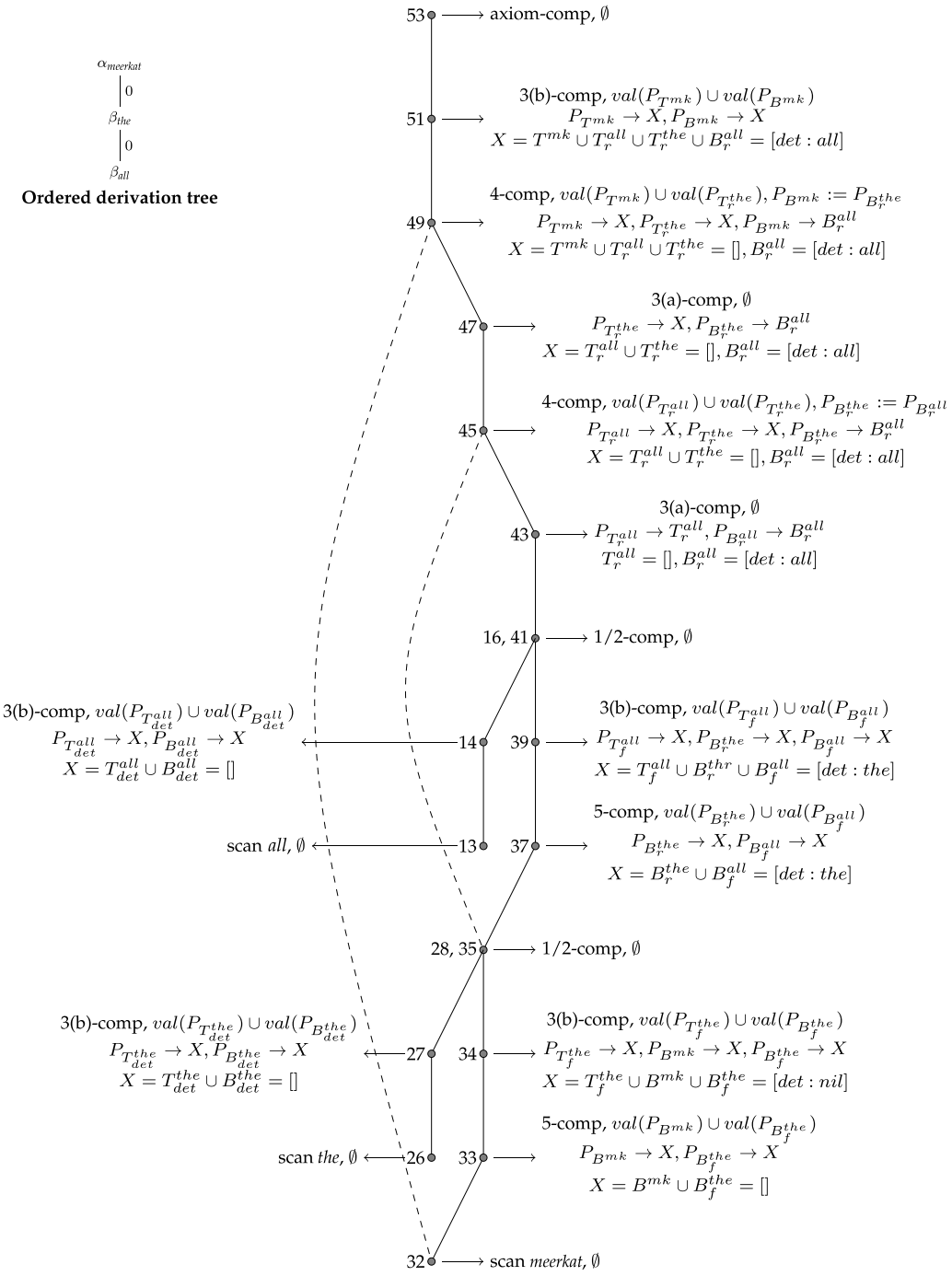


Figure A.3
Feature unifications in dependent derivation of *all the meerkats* (prediction rules are abbreviated).

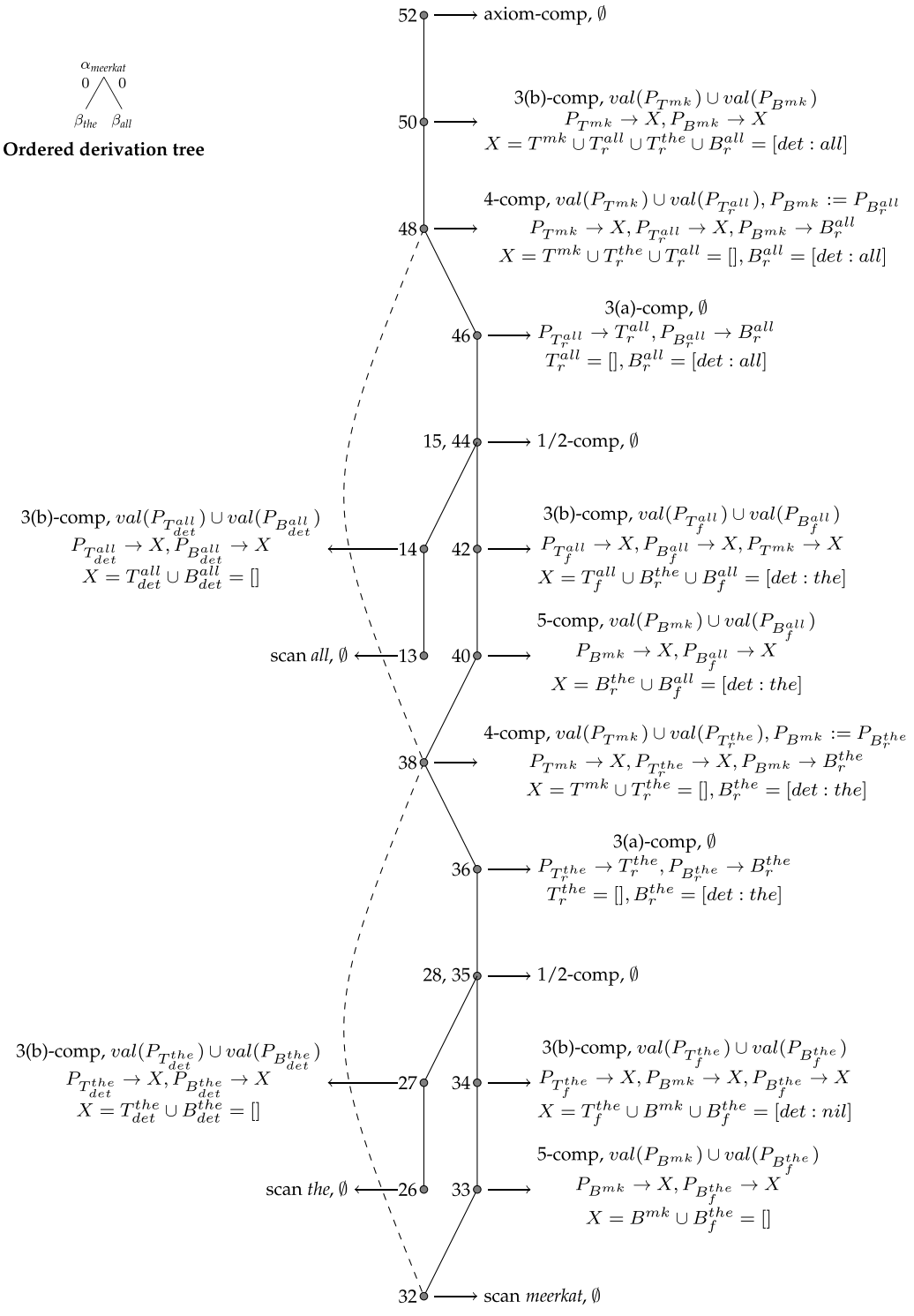


Figure A.4
Feature unifications in independent derivation of *all the meerkats* (prediction rules are abbreviated).

Acknowledgments

We would like to thank Laura Perez-Beltrachini for useful discussion on the related topic. We are grateful to our anonymous reviewers for their insightful comments, which helped us improve the quality of the article in terms of both presentation and content.

References

Alahverdzhieva, Katya. 2008. XTAG using XMG. Master's thesis, Université de Nancy.

Candito, Marie-Hélène and Sylvain Kahane. 1998. Can the TAG derivation tree represent a semantic graph? An answer in the light of Meaning-Text Theory. In *Proceedings of the Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks (TAG+4)*, pages 21–24, Philadelphia, PA.

Crabbé, Benoît, Denys Duchier, Claire Gardent, Joseph Le Roux, and Yannick Parmentier. 2013. XMG: eXtensible MetaGrammar. *Computational Linguistics*, 39(3):1–66.

Gardent, Claire. 2008. Integrating a unification-based semantics in a large scale Lexicalised Tree Adjoining Grammar for French. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING)*, pages 249–256, Manchester.

Gardent, Claire and Laura Kallmeyer. 2003. Semantic construction in Feature-Based TAG. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 123–130, Budapest.

Gazdar, Gerald. 1988. Applicability of indexed grammars to natural languages. In Uwe Reyle and Christian Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, volume 35 of *Studies in Linguistics and Philosophy*. Springer, Netherlands, pages 69–94.

Joshi, Aravind K. 2000. Relationship between strong and weak generative power of formal systems. In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 107–114, Paris.

Joshi, Aravind K. and Yves Schabes. 1997. Tree-Adjoining Grammars. *Handbook of Formal Languages and Automata*, 3:69–124.

Joshi, Aravind K. and K. Vijay-Shanker. 2001. Compositional semantics with lexicalized tree-adjoining grammar (LTAG): How much underspecification is necessary? In Harry Bunt, Reinhard Muskens, and Elias Thijsse, editors, *Computing Meaning*, volume 77 of *Studies in Linguistics and Philosophy*. Springer, Netherlands, pages 147–163.

Kallmeyer, Laura and Marco Kuhlmann. 2012. A formal model for plausible dependencies in lexicalized tree adjoining grammar. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, pages 108–116, Paris.

Kroch, Anthony. 1989. Asymmetries in long distance extraction in a tree adjoining grammar. In M. Baltin and A. Kroch, editors, *Alternative conceptions of phrase structure*, University of Chicago Press, pages 66–98.

Rambow, Owen, K. Vijay-Shanker, and David Weir. 1995. D-Tree Grammars. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 151–158, Cambridge, MA.

Resnik, Philip. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the 14th Conference on Computational linguistics (COLING)*, volume 2, pages 418–424, Nantes.

Schabes, Yves. 1991. The valid prefix property and left to right parsing of tree-adjoining grammar. In *Proceedings of the 2nd International Workshop on Parsing Technologies (IWPT)*, pages 21–30, Cancun.

Schabes, Yves. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th Conference on Computational Linguistics (COLING)*, volume 2, pages 425–432, Nantes.

Schabes, Yves and Aravind K. Joshi. 1988. An Earley-type parsing algorithm for tree adjoining grammars. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 258–269, Buffalo, NY.

Schabes, Yves and Stuart M. Shieber. 1992. An alternative conception of tree-adjoining derivation. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 167–176, Newark, DE.

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

Shieber, Stuart M. 1994. Restricting the weak-generative capacity of synchronous

tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385.
Steedman, Mark. 2000. *The syntactic process*. MIT Press, Cambridge, MA.
The XTAG Research Group. 2001. A lexicalized tree adjoining grammar for English. Technical report ICRS-01-03, Institute for Research in Cognitive Science, University of Pennsylvania.
Vijay-Shankar, K. and Aravind K. Joshi. 1985. Some computational properties of tree adjoining grammars. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 82–93, Chicago, IL.
Vijay-Shanker, K. 1987. *A Study of Tree Adjoining Grammars*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia.
Vijay-Shanker, K. and Arvind K. Joshi. 1988. Feature structures based tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, pages 714–719, Budapest.
Vijay-Shanker, K. and Arvind K. Joshi. 1991. Unification-based tree adjoining grammar. Technical report MS-CIS-91-25, School of Engineering and Applied Science, Department of Computer and Information Science, University of Pennsylvania.
Vijay-Shanker, K. and David J. Weir. 1991. Polynomial parsing of extensions of Context-Free Grammars. In Masaru Tomita, editor, *Current Issues in Parsing Technology*, volume 126 of *The Springer International Series in Engineering and Computer Science*. Springer US, pages 191–206.
Vijay-Shanker, K. and David J. Weir. 1993. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.
Weir, David J. and Aravind K. Joshi. 1988. Combinatory Categorial Grammars: Generative power and relationship to linear context-free rewriting systems. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 278–285, Buffalo, NY.