Representing and Accessing [Textual] Digital Information (COMS/INFO 630), Spring 2006

Lecture 22: TAG Adjunction Trees and Feature Based TAGs

4/20/06

Lecturer: Lillian Lee

Scribes: Nicolas Hamatake (nh39), Min Park (mp257)

---

1 Overview and Review

In today's lecture we continue our discussion of *Tree Adjoining Grammars* (TAG's) in attempting to analyze a phrase that has the same syntactic structure for different semantic interpretations. Recall that toward the end of last lecture, we took note that for TAG's, parse trees are based on individual lexical items that compose associated information. Is it possible then, to use TAG's to also model idioms that have non-compositional meaning?

Idioms are 'fixed' phrases with non-compositional meanings, and few modifications (if any) are allowed. We introduced the following idiom as a concrete example to guide our discussion today:

Bob kicked the bucket.

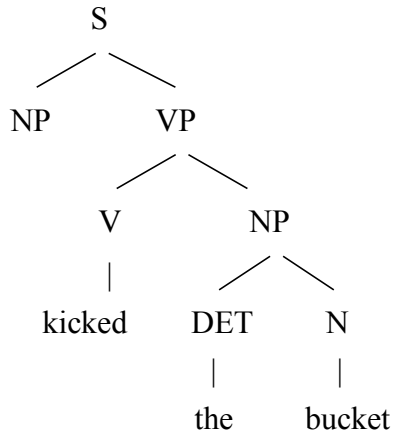
Interpretation of this phrase is not unique. There is at least the literal interpretation, but also an idiomatic one. Either Bob kicked—with his foot—a pail (literal); or, Bob died (idiomatic).

Notice that only a few idiomatic modifications of this phrase are imaginable. For example, *Bob kicked the red bucket, etc.,* is not idiomatic (within reason). Still, *Bob kicked the proverbial bucket* would be an idiomatic modification of the phrase. Today we continue exploring the difficulty that arises from idiomatic phrases.

2 Ambiguity in Analyses of Idiomatic Phrases

2.1 Idiomatic and Non-Idiomatic Syntactic Analyses

Consider the following simplified *non-idiomatic* syntactic analysis of (1.1):



where *Det* denotes determiners—words such as *the*, *that*, and so forth. What are the compositional semantics here?

We could imagine that a compositional semantic analysis would be easily derived via writing a TAG with an initial tree for *kicked*, α_k [1]. This initial tree for "kicked" will require two arguments: a subject and an object. Thus for a compositional semantic analysis, we have the logical form:

Kicked(*Bob,bucket)*

which is just a function (*Kicked*()), which has two arguments for the subject and object analogous to the two non-terminal leaf nodes in α_{kicked} .

For the case of the *idiomatic* syntactic analysis of (1.1), the appropriate logical form would be:

Died(Bob)

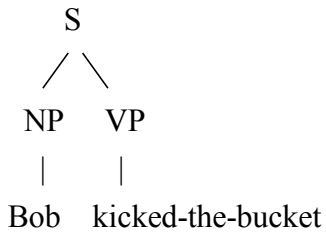
which is a function of one argument instead of two. We now have two different logical forms, (1.2) and (1.3), for our phrase (1.1).

What is each of their corresponding *syntactic* analyses? Should they be different from each other? If they are different, the implication would be that grammar is able to reflect their difference in meaning.

2.2 Syntactic Analyses

2.2.1 *Idiom as One Lexical Item: No.*

First, one might attempt a syntactic analysis where the idiom is treated as a single lexical item:



Initially, this seems to make sense, given the non-compositional semantics of the phrase. However, treating 'kicked-the-bucket' as one word would not allow for *any* modification such as 'kicked-the-*proverbial*-bucket' which is still idiomatic. Thus, we need a place in our analysis for a modifier as in the case with the non-idiomatic analysis (unless we decided to have a separate tree for the modified case).

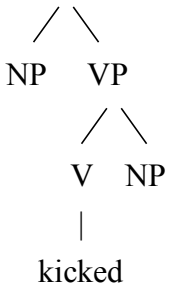
In this sense, we need the 'same' syntactic analysis for both the idiomatic and non-idiomatic case. Clearly we are faced with ambiguities in having the 'same' syntax for divergent semantics. Is there some way to capture these ambiguities? Where exactly does the vagueness arise?

[1] Given in the following page.

2.2.2 Same Syntax for Two Meanings: Where does the Ambiguity Arise?

First consider using TAG's to represent the literal interpretation of (1.1).

α_k :

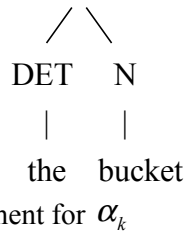


α_k is the initial tree for *kicked*, which takes two arguments.

α_B :

α_B is the tree for *Bob*, one of the arguments for α_k

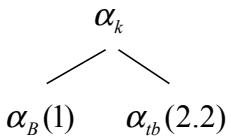
α_{tb} :



argument for α_k

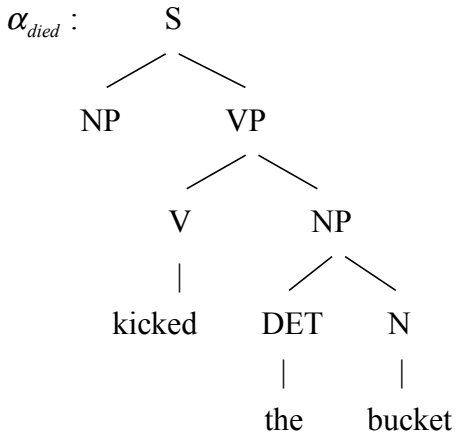
α_{tb} is the simplified tree for *the bucket*, the other

Starting with the initial tree α_k we can write the derivation tree for (1.1) using TAG's as:



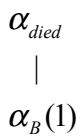
We will claim that it is possible to immediately observe the logical form of (1.1) from this derivation tree. Recall (1.2), *Kicked*(*Bob,bucket*), the logical form for the non-idiomatic syntactic analysis of (1.1). The nodes $\alpha_B(1)$ and $\alpha_{tb}(2.2)$ correspond to the arguments, while α_k corresponds to the function *Kicked*.

Secondly, consider using TAG's to represent the idiomatic interpretation of (1.1).



α_{died} is another initial tree, which takes one argument.

Starting with the initial tree α_{died}, we can also write the derivation tree for (1.1) using TAG's as:



Again, we can 'read off' the logical form for this syntactic analysis, *Died*(*Bob*), immediately from this derivation tree.

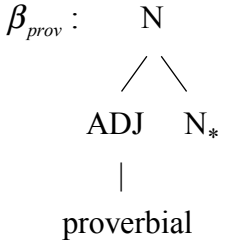
By using TAG's, we have shown that the derived tree itself is not the source of ambiguity since both derivations lead to the same tree; however, the semantics has become clear in the derivation process. That is, the derivation process is the source of vagueness as represented by the fact that we have two plausible derivation trees.

3 Modification for Idioms: Adjunction

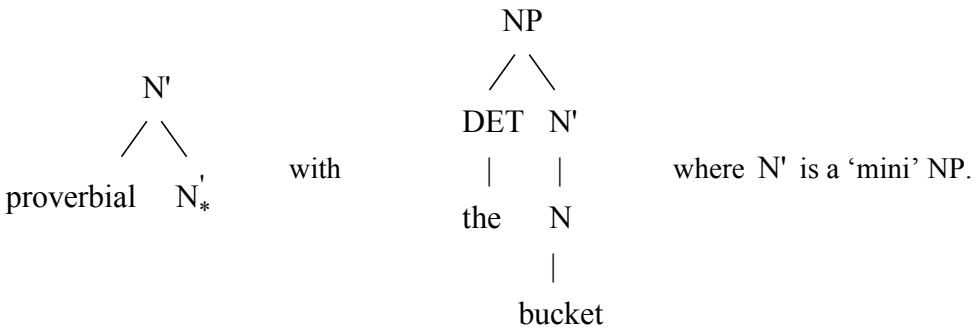
3.1 Adjunction Trees to Allow for Modification

We return to our problem presented in section *2.2.1* where we attempted a syntactic analysis of (1.1) treating *kicked-the-bucket* as one lexical item. The problem was that this analysis would not allow for any (idiomatic) modification of the phrase.

To allow for modification of an idiom, consider having the following adjunction tree. Since it will modify *bucket* , its root (under simplified analysis) must be N and so should its foot:

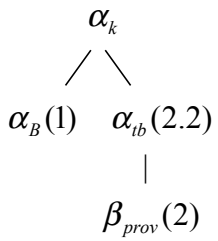


Note that we are slightly oversimplifying here, and that β_{prov} should really be:



Adhering to our oversimplified notation (which is acceptable for our purposes), the logical form for β_{prov} is
$proverbial(bucket)$.

But notice that we have a problem. While we introduced β_{prov} to modify idioms, it also applies in the non-idiomatic case. The derivation tree



implies the logical form *Kicked*(*Bob*,*proverbial*(*Bucket*)), where the second object is a metaphorical object (note that we have *not* analyzed this as *Kicked*(*Bob*, *bucket*(*proverbial*), see questions). This is not possible (it is an 'incorrect' logical form). We could try fixing this problem with some constraints on adjunction.

3.2 Adjunction Contraints

3.2.1. Selective Adjunction Constraints

A selective adjunction constraint would be that only auxiliary trees in a node-specific set may adjoin. For example the set $\{\beta_{prov}, \beta_{so-called}\}$ for note 2.2.2 of α_{died}.

3.2.2 Obligatory Adjunction Constraints

Do we need any other types of constraints? Consider the following sentences:

1. Bob saw Joe.

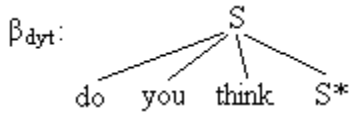
2. Whom did Bob <u>see</u>?

Note that in order to form the interrogative of the first sentence, we have to change the form of the verb phrase (or include a trace) and indicate the requirement for an appropriate filler, as we have described before. This would imply that we need multiple initial trees for the verb "see." This is probably fine, since there are only three basic types of English sentences.

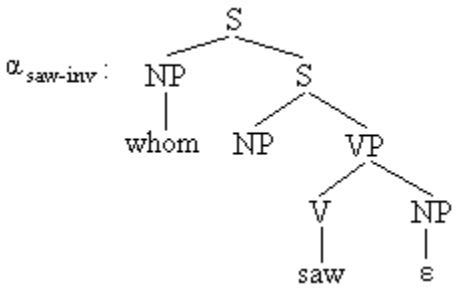
A more important problem arises when the sentence is qualified:

Whom <u>do you think</u> Bob saw?

Many qualifiers are possible here: "do you claim," "do you suppose," "do you assert," etc. The number of initial trees required, if they must include such qualifiers, is beginning to explode multiplicatively. To avoid this, we might treat "do you think" as a modifier and represent it using a (simplified) auxiliary tree:



The initial tree that this would be adjoined into would look like:



("inv" stands for "inverted")

If we adjoin β_{dyt} into $\alpha_{saw\text{-}inv}$ at node (2) and substitute an appropriate α_{Bob} into node (2.1), we get "Whom do you think Bob saw?" However, if we just substitute α_{Bob} into $\alpha_{saw\text{-}inv}$ without adjoining β_{dyt}, we get, "Whom Bob saw?" which is clearly not a valid sentence. The problem in this case is not the insertion of α_{Bob} at node (2.1); rather, the problem is that the sentence is not complete without some kind of qualifying phrase, or at least an auxiliary verb like "will" or "did," being adjoined into node (2).

This problem can be solved by introducing an *obligatory adjunction constraint*. This type of constraint, like a selection constraint, associates a set of auxiliary trees with a specific node of an elementary tree. The constraint indicates that some auxiliary tree in the set must be adjoined into the elementary tree at the given node in order for the derived tree to be considered "complete." To solve the problem with the example above, we would introduce an obligatory constraint like "node (2) of $\alpha_{saw\text{-}inv}$: $\{\beta_{dyt}, \beta_{do\text{-}you\text{-}claim}, \ldots\}$."

3.2.3 Null adjunction constraints

We now note one final problem with the above examples to introduce the last type of constraint. Since β_{dyt} has a node of type "S" as its root and foot, we could legally adjoin it into the root of any initial tree. We then get sentences like "Do you think Bob saw," which are not desirable. To remedy this, we introduce the *null adjoining constraint*, which simply specifies that no adjunctions are allowed at a

particular node of a particular elementary tree (e.g., the root in this case). Observe that this is equivalent to specifying a selection constraint in which the set of legal auxiliary trees is empty. For this reason, the null constraint is sometimes considered a special case of the selection constraint, rather than its own distinct type.

3.2.4 General observations about adjunction constraints

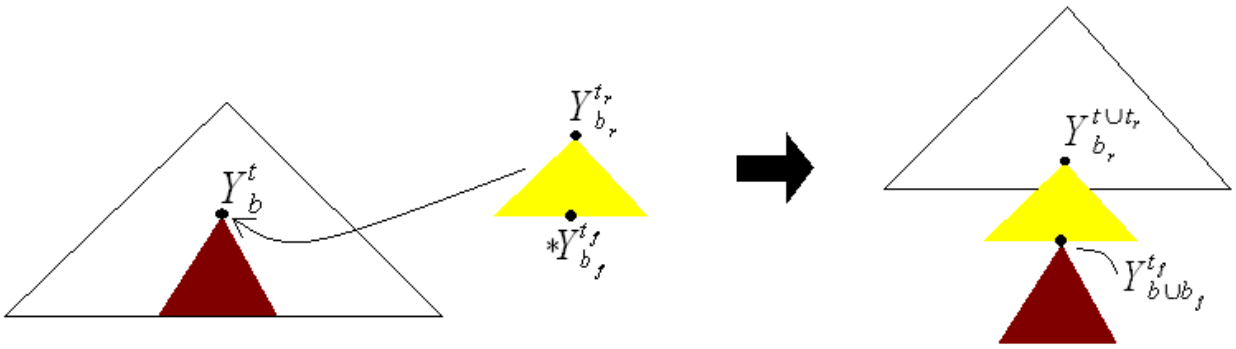
Somewhat surprisingly, adjunction constraints actually add power to TAG's in a formal, mathematical sense (although this claim is not proven here). They also increase the size of the grammar combinatorially, since almost every node of every tree will need constraints, many of which involve large sets of auxiliary trees.

4 Feature-based TAG's

This combinatorial explosion of grammar size was encountered in our consideration of CFG's, under very similar conditions. In CFG's, the problem was ameliorated with features, so it is natural to ask whether features can somehow be incorporated into TAG's with similar results.

The "feature-based TAG" was presented by Vijay-Shankar and Joshi in 1988. The overall idea is to add features to the root and foot of an auxiliary tree and require that these features match those of any node that it is adjoined into.

Specifically, each node (of both elementary and auxiliary trees) has a set of "top" and "bottom" features. We use Y^t_b to denote the node Y, with top features t and bottom features b. In order for an auxiliary tree to be adjoined into Y, the top features of its root must be compatible with the top features of Y and, similarly, the bottom features of its foot must be compatible with the bottom features of Y. The process can be illustrated as follows:



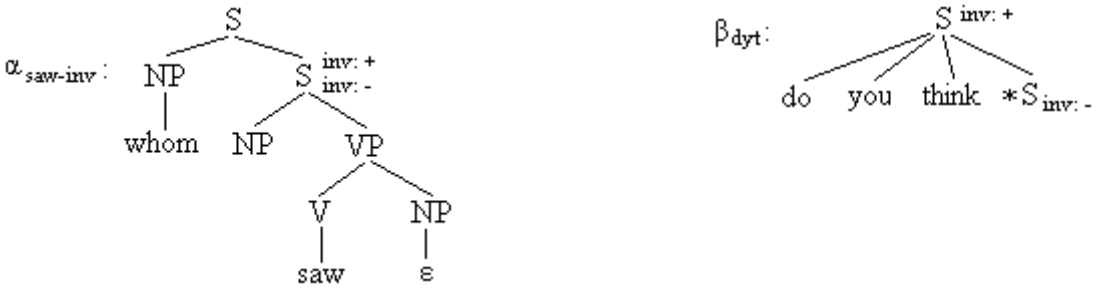
In this figure, the auxiliary tree with root $Y^{t_r}_{b_r}$ and foot $Y^{t_f}_{b_f}$ is being adjoined into the initial tree at node Y^t_b. The features t_r and t must be compatible with each other, and their "unification" becomes the top features of the top "copy" of Y in the final tree. Similarly, b and b_f must be compatible, and their unification forms the bottom features of the bottom copy of Y in the final tree. The union symbol is used in the figure to represent the unification of features. If t_r and t are not compatible, the adjunction is not allowed (and similarly for b and b_f).

4.1 Constraint enforcement in feature-based TAG's

This setup enables straightforward enforcing of selection constraints. If we only want to allow certain adjunctions at Y_b^t, we add a feature to *t*, then add this same feature to the top feature sets (t_r, in the above figure) of the roots of exactly the auxiliary trees that are allowed to adjoin.

In order to enforce null constraints, we need a special non-unifying feature that does not unify with any other features (including itself).

To enforce obligatory constraints, we require that the top and bottom features of every node in a derived tree be unifiable in order for the derivation to be considered complete. If we wish to force an adjunction at node Y_b^t, we make sure that *t* and *b* are not compatible and we engineer an auxiliary tree that will resolve the incompatibility. This is understood more easily with an example. Suppose we wish to enforce an obligatory constraint to prevent sentences like "Whom Bob saw" from being legal. Our basic trees would look like:



(Only the minimum number of features needed to enforce the constraint are made explicit).

Since $\alpha_{saw\text{-}inv}$ now contains a node whose top and bottom features are incompatible, the tree would not be considered complete after just substituting α_{Bob} into the appropriate node. When β_{dyt} is adjoined, the features match up to yield an S node with inv:+ for both top and bottom features, and an S* node with inv:- for both top and bottom features.

Roughly speaking, a node's top features represent what that node "wants" to have beneath it, while a node's bottom features represent what is currently beneath it. The S node in $\alpha_{saw\text{-}inv}$ wants something inverted beneath it, but currently only sees the non-inverted "NP VP" structure below. The root of β_{dyt} provides the structure that the S node in $\alpha_{saw\text{-}inv}$ is seeking.

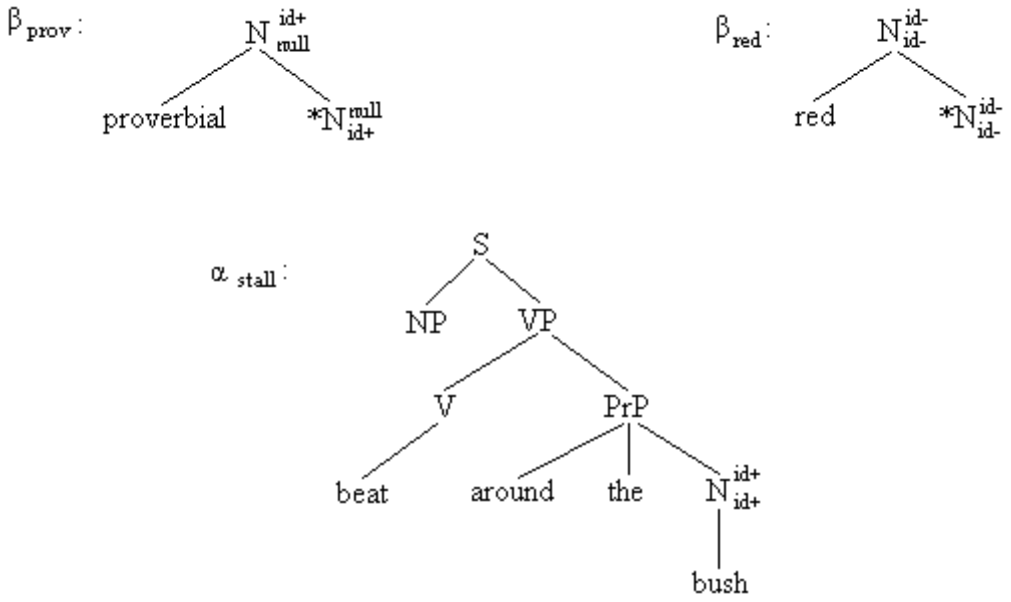
5 Questions

- Consider the idiom "beat around the bush." We want to construct a tree to represent this phrase as-is, but we also want to allow "beat around the proverbial bush" to share the same construction. We do not, however, want to allow "beat around the red bush." Construct trees α_{stall}, β_{prov}, and β_{red}, providing feature-based constraints that will enforce the structure described above. (Note: We do not want to allow "beat around the proverbial proverbial bush.")

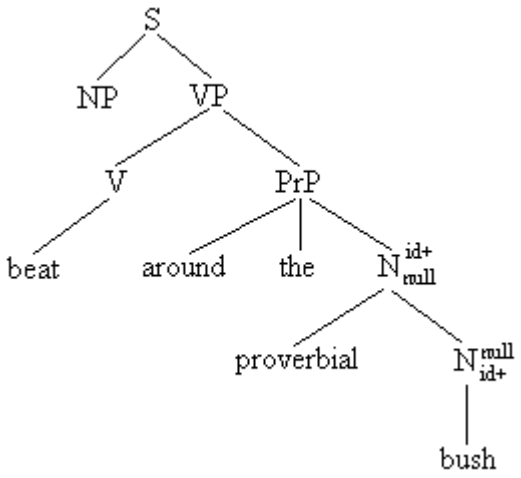
- Given a "reasonable" grammar, the logical form of "Bob beat around the bush" would look like: stall(Bob). What logical form arises when β_{prov} is used to modify this idiomatic phrase? In what way(s) could this be problematic?

Answers

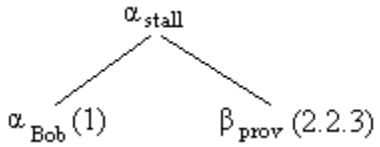
1) Sample trees could look like:



"null" is used to indicate a feature that does not unify with anything. Note that as soon as an appropriate noun phrase is substituted at node 1, α_{stall} is considered complete, since the features of "N" (node 2.2.3) already unify. The modifier "red" cannot be spliced in, since its features don't match. The modifier "proverbial" can be spliced into the tree, but it will leave null features at the "N" node above "bush," preventing any further modifiers from being spliced in:

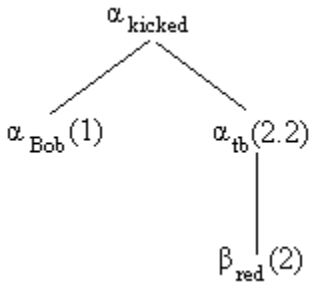


2) If we assume the obvious structure for α_{Bob} and substitute it into the above tree, the resulting derivation tree would look like:



We now seem to have a problem, in that it looks like stall(a) is now required to take <u>two</u> arguments. (We saw a similar problem for "bucket" in the derivation tree given in section 3.1.) So, we need to specify that auxiliary-tree nodes in a derivation tree do not represent arguments of the function represented by their parent, but rather are functions that take whatever their parent represents as an argument: metaphorical(stall(Bob)).

Does this work for more typical modifiers? Consider the "Bob kicked the bucket" example from section 3.1 and assume the obvious structure for β_{red}. The derivation tree for "Bob kicked the red bucket" would look like:



This suggests the logical form: kicked(Bob, red(bucket)), which seems fine.

Open question: Is this a robust solution to the representation of the "semantics" of auxiliary trees?