CS630 Lecture Notes

Lecturer: Lillian Lee
Scribes: Chris Danis (cgd3) & Brian Rogan (bcr6)

Lecture 20: 13 April 2006

1 Introduction

Today, we are continuing our coverage of models of syntax. We will focus on feature-based context-free grammars (FBCFGs), and will also introduce tree-adjoining grammars (TAGs).

2 Motivations

Last time, we covered context-free grammars, or CFGs. We encountered two issues in attempting to use CFGs to model natural language.

The first issue we encountered was that of category/feature proliferation. In order to get tenses, plurality, etc., to agree, we were forced to create a non-terminal and a production rule for every possible combination (e.g. Noun-phrase-3rd-person-singular), resulting in a combinatorial explosion. While this isn't a showstopper, it certainly is inelegant.

Even more troubling was the issue of maintaining long-distance dependencies. The constituents of an English sentence must agree in several different ways, and this agreement must be consistent throughout the sentence. The most telling example is given by "filler-gap constructions". Consider the sentence:
$[[\text{police}]_{\text{NP}}\ [[\text{informed}]_{\text{V}}\ [\text{Bob}]_{\text{NP}}\ [\text{that...}]_{\text{S}'}]_{\text{VP}}]_{\text{S}}$
To convert this question to an interrogative, we would write:
$[[\text{whom}]_{\text{NP}}\ \text{did police}\ [[\text{inform}]_{\text{V}}\ [\text{that...}]_{\text{S}'}]_{\text{VP}}]_{\text{S}}$

To model the first sentence in our CFG, we would need the rule

$$\text{S} \rightarrow \text{NP}\ \ \text{VP} \tag{1}$$

and also the rule

$$\text{VP} \rightarrow \text{V}\ \ \text{NP}\ \ \text{S}' \tag{2}$$

And to model the interrogative form in our CFG, we would also need the rule

$$\text{VP} \rightarrow \text{V}\ \ \text{S}' \tag{3}$$

However, note that the combination of 3 and 1 permits a sentence NP V S′. While such a construction is permissible for some verbs, it is clearly bad in this case.

Two solutions have been proposed to address this issue. The first is to introduce a non-terminal VP-missing-object. This is clumsy, and only exacerbates the category proliferation issue. The standard proposal is to use "traces". We use ε to represent an empty string – essentially, a silent, unwritten "word". We can rewrite rule 3:

$$\text{NP} \rightarrow \varepsilon$$

and combined with rule 2, we have the "missing direct object" case. We now write: $[[\text{whom}]_{\text{NP}}$ did police $[[\text{inform}]_{\text{V}}$ $[\varepsilon]_{\text{NP}}$ $[\text{that...}]_{\text{S}'}]_{\text{VP}}]_{\text{S}}$. Note that the "filler" *whom* coordinates with the "gap" ε.

However, we still have agreement problems. We'd *like* to write something like

$$\text{S} \rightarrow \text{NP VP} \quad \text{w/agr(NP)} = \text{agr(VP)}$$

that is, a sentence is a noun phrase and a verb phrase, such that the agreement information for the two phrases match, and

$$\text{S} \rightarrow \text{NP InvS} \quad \text{w/NP} \neq \varepsilon, \ \varepsilon \in \text{InvS},$$

$$\text{f(NP)} = \text{f(InvS)}$$

that is, another type of sentence is a noun phrase followed by an "inverted sentence" structure, with NP not empty, an ε in the inverted sentence constituent, and with feature agreement between the two.

3 Feature-based CFGs

Feature-based CFGs describe the exact same set of sentences and of parse trees as CFGs[1]. However, by adding notation, we can reduce some of the inelegancies we encountered. In FBCFGs, we describe constraints using features, which are simply pairs of variables and values in a "feature structure".

Lexical entries in FBCFGs are simply production rules that map from feature structures to the actual words. For instance, a lexical entry for inform[2]:

$$\begin{bmatrix} CAT : V \\ ROOT : inform \\ SUBCAT : \begin{bmatrix} 1 : \begin{bmatrix} CAT : & NP \\ RES : & animate \\ CASE : & \{acc, -\} \end{bmatrix} \\ 2 : \begin{bmatrix} CAT : S' \end{bmatrix} \end{bmatrix} \end{bmatrix} \rightarrow \textit{inform}$$

Or, add VFORM: past to the feature structure and then we have a lexical entry for "informed".

[1] This is assuming that the variables take on only a finite set of values, and also assuming that we ignore the differences in the names of labels.

[2] Where "RES" is short for "further restrictions".

We'd like to have only one VP rule for both the gapped and the ungapped case (considering only the first argument). Here it is for FBCFGs:

$$\begin{bmatrix} CAT : VP \\ GAP : \ ?g \end{bmatrix} \rightarrow \begin{bmatrix} CAT : & V \\ SUBCAT : & \begin{bmatrix} 1 : ?a1 \\ 2 : ?a2 \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} ?a1 \\ GAP : ?g \end{bmatrix} \quad \begin{bmatrix} ?a2 \\ GAP : - \end{bmatrix}$$

Note that the variables ($?g$, $?a1$, $?a2$) handle the equality constraints, and pass around the information involved in maintaining the long-distance dependencies. (Our notation is not quite right here, but we will overlook this for simplicity's sake.) Specifically, the V will impose constraints on its arguments via its $SUBCAT$ feature, which forces these constraints to hold for the 2nd & 3rd nonterminals on the right-hand side of the rule; and because the gap features must match any relevant features of the 2nd argument, the constraints will be passed, via the $?g$ variable, to the parent VP.

We can also write a rule to generate a trace (ε) in the appropriate places:

$$\begin{bmatrix} CAT : & ?c \\ CASE : & ?case \\ RES : & ?r \\ GAP : & \begin{bmatrix} CAT : & ?c \\ CASE : & ?case \\ RES : & ?r \end{bmatrix} \end{bmatrix} \rightarrow \varepsilon$$

FBCFGs work well for handling natural language in practice. However, there are still inelegant elements. For instance, there is redundancy in the specification of feature structures and production rules:

$$[\,] \rightarrow \begin{bmatrix} 1 : ?a1 \\ 2 : ?a2 \end{bmatrix} \ [\,] \ [\,]$$

Here we specify *twice* that we need two arguments: first, in the feature structure specifying ?a1 and ?a2, and then, in the actual production rule. This is, again, inelegant.

4 Tree-adjoining grammars

Tree-adjoining grammars (TAGs) are a rather different approach that address the issues we saw with both plain CFGs and feature-based CFGs. TAGs were originally developed by in 1975 by Joshi, Levi, and Takahashi ([1]).

The basic unit in a tree-adjoining grammar is (as one might expect) a tree. All internal nodes are labeled by non-terminals, and the leaves of a tree can be terminals or non-terminals. There are two elementary tree types; however, the only type we discuss today is the initial tree. Several examples of initial trees are shown in Figure 1. Note that all the argument requirements for "inform" are specified by a single initial tree and nowhere else (in these examples).

The operation used to manipulate and combine initial trees is substitution. A substitution operation is performed by replacing a leaf non-terminal of an elementary tree with another initial or derived tree whose root has the same label as the leaf non-terminal being replaced. We use Gorn numbering to denote the nodes involved in the substitution operation (see Figure 2).

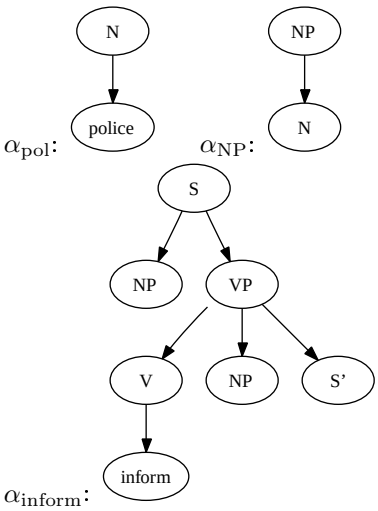

Figure 1: Several example initial trees

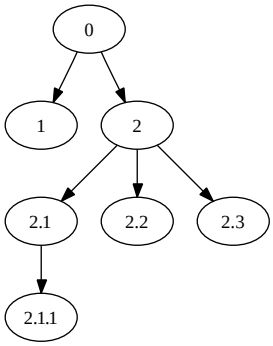

Figure 2: An example Gorn numbering

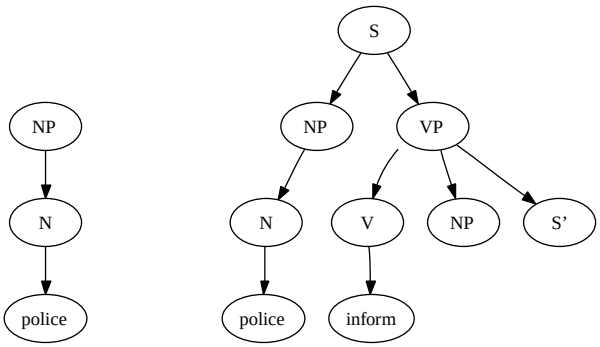

Figure 3: Substitution operations

Substituting α_{pol} into node 1 of α_{NP} results in the tree shown on the left in Figure 3 and substituting that resultant tree into node 1 of α_{inform} results in the tree shown on the right in Figure 3.

We will continue our coverage of TAGs next lecture.

References

[1] Aravind K. Joshi, L. Levy, and M. Takahashi. Tree-adjunct grammars. *Journal of Computer and System Science*, 21(2), 1975.

CS630 Lecture Problems

Lecturer: Lillian Lee
Scribes: Chris Danis (cgd3) & Brian Rogan (bcr6)

Lecture 20: 13 April 2006

1. Recall in class that we discussed filler-gap constructions.

   (a) Describe in your own words what a filler-gap construction is.
   ANSWER:
   As we have specified this be provided in the reader's own words, we find it unlikely that any solution offered here can be correct for any student other than the authors. If the reader wished, however, to adapt our understanding of the concept, she might start by describing a filler-gap construction as a relic of restating a specific sentence where various parts of the sentence are moved in the restatement. This provides a filler (the new location of that part of the sentence) and a gap (the place where the displaced piece formerly resided). For example, when converting a statement to an interrogative, it is possible for a noun-phrase to move from being part of a verb phrase to being part of that sentence at the top level. (Ed: Another example is topicalization: "Burgers, I like. Sushi, I don't", or relative clauses. Another example is: "The officer that Joe, Bob and I complained to ... ")

   (b) Describe why this poses a problem when using (non-feature based) CFG's as a model of natural language.
   ANSWER:
   Filler-gap constructions pose a problem to using CFG's because they appear to force the CFG's to capture long-distance dependencies. Observe the example that we discussed in class:

   $[[\text{whom}]_{NP}$ did police $[[$ inform $]_V$ $[\epsilon]_{NP}$ [that ...$]_{S'}$ $]_{VP}$ $]_S$

   Notice that the "whom" plays the role of the ϵ, however, the ϵ is part of the verb phrase, while the "whom" is actually part of the sentence that contains the verb phrase, not the verb phrase itself. This means that the CFG gives no clean way to specify that the filler has been provided by the sentence, and does not need to be provided by the verb phrase. Essentially, the production/constituent

that includes the noun-phrase has changed from the verb phrase to the sentence. This means though that information must be passed from the sentence production into the verb phrase production to ensure the sentence stays valid. In lecture we called this problem a "long-distance dependency," and posed it as evidence suggesting that innately CFG's may not be the best tool for the job.

(c) Would posing NL as the intersection of two CFL's help to solve this problem? What is the downside of this approach?

ANSWER:

Posing NL as the intersection of two CFL's would certainly seem to help with this problem: one CFL could provide basic structure just as we did above, while simply allowing fillers and gaps to be included in any place where they are relevant, while the second CFL could enforce one long-distance constraint (for example that the filler and the gap match). One potential advantage of using a CFG over a simpler grammar is far greater context sensitivity: a CFG could specify that the filler was part of the first noun phrase while the gap was part of a verb phrase that followed later on in the sentence. In this way the second CFG could mirror the first, except the terminals would only match a filler to a gap and allow context-specific wildcards everywhere else. This would allow substantial control over where in the sentence filler-gap constructions were allowed. In some ways this is a redundancy, but it could help to separate the filler-gap rules from the general language rules. (Ed: It may be possible to get much the same power simply by intersecting a CFG with a regular set, which is effectively a CFL. In this case we've effectively gained nothing.)

There are other significant drawbacks to this approach though. First, it is not clear that being able to enforce one long-distance constraint is sufficiently expressive for all scenarios. In a complex English sentence, it may be the case that there are multiple filler-gap constructions which all need to be enforced. Second, this does seem like a substantial added complexity for a small benefit. Finally, this also feels like somewhat dangerous territory. Though deciding membership in a set formed by the intersection of two CFLs is simple enough (we simply run our membership algorithm for the input string on both CFLs and only accept when both accept) determining the emptiness of this intersected set is undecidable. Thus if we devised some arbitrary formulation of valid sentences based upon this intersection rule, it might not be possible to determine if the formulation actually accepted any sentences at all.

2. In class we also spoke about feature-based CFG approaches to describing sentences.

   (a) What was the primary attraction of feature-based approaches over

just plain CFGs as we used in the previous question?

ANSWER:

Recall that when using "straight" CFGs, as in the previous part, we need to rewrite the same rules over and over to ensure agreement of various forms (gender, tense, gap, etc.). This meant that we needed many many productions to state even the simplest ideas, many of which were restatements of one another except providing for agreement for each value that the relevant variables could take (e.g. past tense/present tense).

(b) In order to implement the notion of variables in feature-based CFGs, we essentially need rewrite rules that convert variables to all possible values (the CFG formalism itself makes no allowance for variables, simply terminals and non-terminals). How is this different from what we discussed regarding non feature-based CFGs in the last question?

ANSWER:

In one sense this is very similar to what we discussed with non-feature based CFGs because ultimately, if we were to look at this as the context free language that results from the application of rewrite rules, it would contain a large number of productions which were highly repetitive, and differed only in terms required to agree (where there were variables before). On the other hand, by extending the language to have semantics which can be used to specify agreement, we place a clear delineation between the underlying rules of our language, and how these rules can be converted to the context-free grammar. This means that for individuals specifying the rules, they need to write far fewer rules which have much more meaning in terms of specifying important features of the language. As a side-note, it may be possible to modify CFG membership algorithms to explicitly do this variable based matching, rather than having to actually apply rewriting rules to an underlying grammar to get a CFG.

(c) How is it that feature based CFGs handle long-distance dependencies better than straight CFGs when ultimately they rely on the same class of grammars and algorithms?

ANSWER:

They handle long-distance dependencies better by explicitly modeling them as features, rather than some more ad-hoc method like creating multiple versions of the same productions (which is admittedly what we're doing under the hood). For example, in class we discussed how we could include a GAP feature, which ensured gap-agreement. Because this is a feature of the productions at the top-level, long-distance information is propagated from leaves up to the root, and ultimately to other leaves where it is relevant.

3. Recall that in class we briefly discussed tree-adjoining grammars. How are the initial trees like the lexical entries in a feature-based CFG model?

How are they different?

ANSWER:
Like a lexical entry in a feature-based model, the elementary trees describe the required context of a particular word, and where it can be used. This feels very feature-based, because the features in the lexical entry place constraints on the valid uses of a word. Unlike feature-based models though, they are far more expressive, because basic rules of the language (i.e. how sentences are composed) are stored at the level of elementary trees, so certain sentences which are valid in the general case can be disallowed for specific words and vice-versa. Superficially at least, this seems like a more descriptive language. We don't need to make general rules for how nouns are used, but rather can produce rules for specific kinds of nouns, in a way that seems more elegant than a feature-based approach.