Syntax Models

Scribed by Gilly Leshed, Blazej Kot

Outline

1. Context-Free Grammar (CFG) issues
2. Feature-based CFGs
3. Tree-Adjoining Grammars (TAGs)

This lecture presents some complications one might encounter when trying to model natural language using CFGs. It then proposes possible approaches to mend these problems, in the form of Feature-Based CFGs and Tree-Adjoining Grammars. It should be noted that although TAGs are arguably more interesting, in practice CFGs are more often used.

1 Context-Free Grammars (CFGs) Issues

CFGs were introduced in the previous lecture as rules that express the decomposition of constituents (parts of a sentence) to sub-constituents. The elementary items of a CFG consist of terminals (lexical items, i.e., words) and non-terminals (constituent labels). The rules are considered 'context-free' as they can be applied 'out of context': given a rule of the form '*constituent X* → *sub-constituents Y*', one can replace *X* with *Y* without considering the context in which *X* occurs.

Important and omitted last time: the definition of CFG requires that the sets of terminals, non-terminals, and productions (rewrite rules) are *finite*.

The following issues arise when attempting to apply CFGs to natural language: category proliferation and long distance dependencies. These issues were presented in the previous lecture, and are briefly reviewed in the next two sections.

1.1 Category/feature proliferation

On the one hand, a rich representation of natural language is valuable, as one of the purposes of natural language is to symbolize and communicate about a complex and rich world. On the other hand, this causes computational problems from an engineering point of view when attempting to apply CFGs in processing natural language.

Note: *Selectional restriction violations* (presented in the previous lecture) are treated by CFGs even though they are semantic problems and not syntactic. For instance, the sentence

police [informed]$_V$ [rocks]$_{NP}$ that students had hired lawyers (i)

is semantically incorrect because the verb *informed* requires an animate subject, and *rocks* is inanimate. However, since this kind of lexically-based problem can be treated by CFGs, practically it is convenient to do so.

1.2 Long-distance dependencies

For example: filler-gap constructions of questions

$[[\text{police}]_{NP}\,[[\text{informed}]_{V}\,[\text{bob}]_{NP}\,[\text{that students had hired lawyers}]_{S'}\,]_{VP}\,]_{S}$ (ii)

Turned into question form:

$[[\text{whom}]_{NP}\,\text{did}\,[\text{police}]_{NP}\,[[\text{inform}]_{V}\,[\text{that students had hired lawyers}]_{S'}\,]_{VP}\,]_{S}$ (iii)

Notice that turning (ii) into (iii) involves 'movement' of the *NP* in the sentence: *bob* disappears and instead we have the *NP whom* in a different position.

To model this movement, we can write a CFG with the following rules:

(1) $S \rightarrow NP\ \ VP$

(2) $VP \rightarrow V\ \ NP\ \ S'$

(3?) $VP \rightarrow V\ \ S'$

Once (1) and (3?) are applied jointly, the following derivation is allowed:

(4) $S \Rightarrow NP\ \ V \Rightarrow NP\ \ V\ \ S'$

This allows the CFG to generate incorrect sentences, for example:

* police informed that students had hired lawyers (iv)

The error here results from the semantics of the verb *informed*, which requires a direct object to complete its meaning.

There may be two possible solutions within CFG to this problem:

1. Include the option to include *VP* followed by a 'missing object'. This rule is general enough to be applied in CFG, however it suffers from being 'aesthetically clumsy', because we have to add such a rule for every type of gapping construction for every constituent that allows a gapped argument.
2. Traces: a standard linguistic solution.
   In this solution, the trace of an object is represented by ε, an 'empty string' object which represents an invisible silent word. Now, the following rule can be introduced:

   (3') $NP \rightarrow \varepsilon$

   With rule (3') at hand, rule (2) does not need to be modified to obtain the case where the direct object has been moved out of the *VP* to form the question. Instead, the *NP* direct object remains as a 'silent ghost'. Applying (2) and (3') jointly allows constructing the following sentence:

   $[[\text{whom}]_{NP}\,\text{did police}\,[[\text{inform}]_{V}\,[\varepsilon]_{NP}\,[\text{that students had hired lawyers}]_{S'}\,]_{VP}\,]_{S}$ (v)

   filler gap containing a trace

   However, this generates a new problem in which a sentence is constructed of *VP* without an *NP*:

   $* \ S \Rightarrow NP\ \ VP \Rightarrow \varepsilon\ \ VP$

   (1) (3')

We do not want the filler *NP* to 'disappear' into an empty string ε as well. In other words, the filler and the gap should be somehow coordinated. Furthermore, the type of filler (e.g. whether we use the term *whom* or *what* in the question form) needs to match the type of noun object in the original sentence that 'turned' into the gap in the question form of the sentence. These issues can be resolved (at least in part) using feature-based CFGs.

2 Feature-Based CFGs

The problems described above can be fixed by adding some flexibility to CFGs. For example, to solve the agreement problems we discussed in the previous lecture, we would like to write the rule $S \rightarrow NP\ \ VP$ with an equality constraint $agr(VP) = agr(NP)$. As another example, in the creation of a question form of a sentence using the rule $S \rightarrow NP\ \ Inv\text{-}S$, we would like to add the constraints: $NP \neq \varepsilon$ and $Inv\text{-}S$ has ε with same features as NP. (Notation: $Inv\text{-}S$ = Inverted Sentence)

In Feature-Based (FB) CFGs these constraints are expressed using features: variables that obtain values. Feature structures are recursive in nature, meaning that one feature structure can be embedded in another feature structure.

Example: the lexical entry for the verb *informed* in a FBCFG:

$$\begin{bmatrix} \text{CAT}: & V \\ \text{ROOT}: & \textit{inform} \\ \text{VFORM}: & \textit{past} \\ \text{SUBCAT}: & \begin{bmatrix} 1. & \begin{bmatrix} \text{CAT}: & NP \\ \text{RES}: & \textit{animate} \\ \text{CASE}: & \{ACC, -\} \end{bmatrix} \\ 2. & \begin{bmatrix} \text{CAT}: & S' \end{bmatrix} \end{bmatrix} \end{bmatrix} \rightarrow \textit{informed}$$

Notations:

CAT = category

RES = restrictions

ACC = accusative

Compare this FB CFG with the standard CFG:

$V \rightarrow \textit{informed}$

Notice that each lexical entry, i.e. terminal, is to be represented using this notation (i.e. a production wherein a preterminal, given as a feature structure containing all the relevant information for the lexical item w in question, expands into w). Furthermore, there could be many additional features that could be expressed in this formalism. For instance, the terminal *eat* requires a direct object that is edible (which we would indicate using the RES feature), but this feature of course does not apply to the terminal *informed*.

In order to solve the filler-gap problem, we would like to use one *VP* expansion rule that would be applied for both the gapped and the non-gapped cases:

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

Notation:
?x = variable

(Note: we are abusing notation somewhat with ?a1 and ?a2 in the second and third nonterminals on the right-hand side.)

Compare this FB CFG with the CFG in section 1.2:

$$VP \rightarrow V\ NP\ S'$$

The right-hand side of the rule describes a verb that accepts two arguments, *A1* and *A2*. These arguments must fulfill equality constraints imposed via the use of variable equality constraints, as expressed through the two appearances of the variables ?a1 and ?a2. The variable *?g* expresses the case where the *VP* potentially involves a gap. Using the same variable in two different places in the rule denotes that whatever is plugged into one place should also be plugged into the other place. The variable *?g* appears both in the left-hand side, the *VP*, and in the first argument of the verb. This means that whenever the *VP* involves a gap, it needs to be equally treated in the filler as in the first argument *A1*. However, the GAP part of the second argument does not contain the variable *?g*, meaning that the filler does not need to agree with the second argument *A2* in the special case of a gapped *VP* and indeed it should *not* exhibit a gap, since a sentence should (ordinarily) not exhibit two gaps in its main VP. This expression makes sure that the filler obtains its features from the appropriate argument. For example, when turning sentence (ii) into its question form (iii) above, the rule ensures that the filler that is used will be *whom* and not *what* because of the restriction of the direct object of *inform* to be animate. Note that this way of handling constraints is more elegant than the other solutions presented in section 1.2 above.

Trace generation: The following rule can be used to generate a trace:

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

Notice that the same variables appear in the GAP feature as well as in the 'parent' features of the non-terminal. This ensures that the gap will have the same features as the category that generated it.

Filler generation: The following rule is required to generate the filler *whom* in sentence (iii) above:

$$S \rightarrow \begin{bmatrix} \text{CAT}: & NP \\ \text{NFORM}: & pronoun \\ \text{GAP}: & ?g \end{bmatrix} did\ police \begin{bmatrix} \text{CAT}: & VP \\ \text{GAP}: & ?g \end{bmatrix}$$

Notice that the verb *inform* tells the *VP* which gap features to use, which then tells the *NP* which gap features to use.

The GAP feature here cannot have '–'

Note: in the current discussion we will not deal with the parsing problem to solve equality (or unification) constraints.

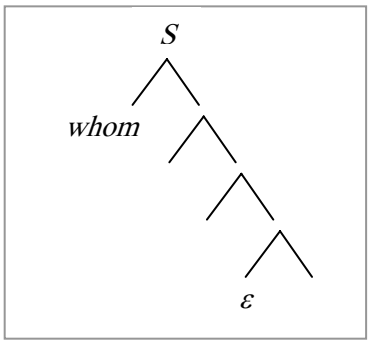
Some comments on FB CFGs:

- Features provide a practical way to apply CFGs to representing natural language, although FB CFGs are still a fairly complicated way to model constraints. This goes back to the question we discussed in the previous lecture of whether English is a context-free language that can be represented using CFGs.
- As an approach, FB CFGs are usable, but from an aesthetic point of view they involve redundancies. This can be observed in the following example, which depicts the redundant expression of the arguments *?a1* and *?a2* in two separate places in the CFG:

$$\begin{bmatrix} \quad \end{bmatrix} \rightarrow \begin{bmatrix} \quad \begin{bmatrix} 1. & ?a1 \\ 2. & ?a2 \end{bmatrix} \end{bmatrix} \begin{bmatrix} ?a1 & \\ & \end{bmatrix} \begin{bmatrix} ?a2 & \\ & \end{bmatrix}$$

- The purpose of using FB CFG for the filler-gap construction is to pass information around in the syntactic tree structure. The information should be passed between the filler constituent and the gap constituent, so that the features of the gap constituent will be sustained by the filler constituent. The phenomenon of *moving constituents within a sentence* is expressed using a mechanism of *preserving constituents' features* in CFGs. Therefore, this approach suffers from a mismatch between the intuitive phenomenon (constituents' movement) and the mechanism (feature conservation). This weakness can be alleviated using Tree-Adjoining Grammars.

3 Tree-Adjoining Grammars (TAGs)[1]



Section 1.2 above discussed long-distance dependencies between constituents. This 'long-distance' can be counted in terms of the number of words in the sentence that separate the two constituents that need to 'communicate'. In terms of the syntactic tree structure, the 'long-distance' can be counted as the number of branches, i.e. rewrite rules that separate the sub-trees of the sentence tree.

In TAGs, sub-trees are treated as *tree-units*. This allows collapsing the distance between the constituents that need to communicate, also described as 'extending the domain of locality'. Notice that CFGs are a special case of TAGs, in which tree units are restricted to be of depth one.

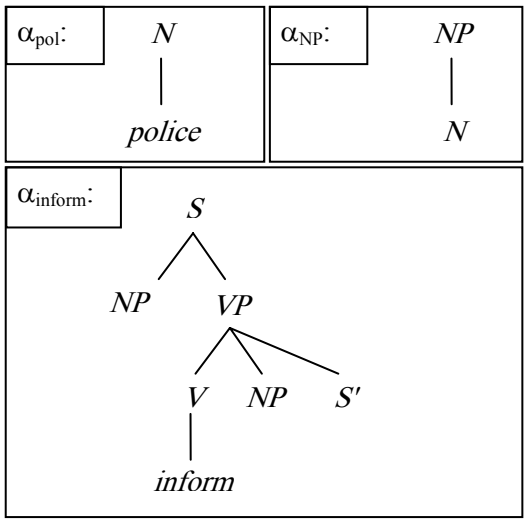
The basic units that are treated are trees, which must satisfy the following restrictions:

- Internal nodes all have non-terminal labels
- Leaves can be terminals or non-terminals

3.1 Initial Trees

There are two elementary types of trees. This lecture covers only one type: initial trees.

Examples:



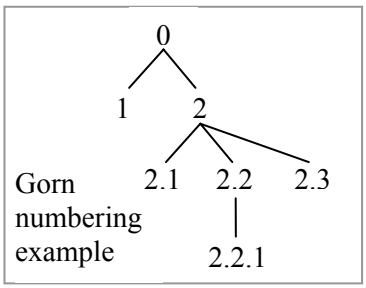
Notation: α = initial tree

Notice that all these trees satisfy the tree requirements: their internal nodes are non-terminals and their leaves are either terminals or non-terminals

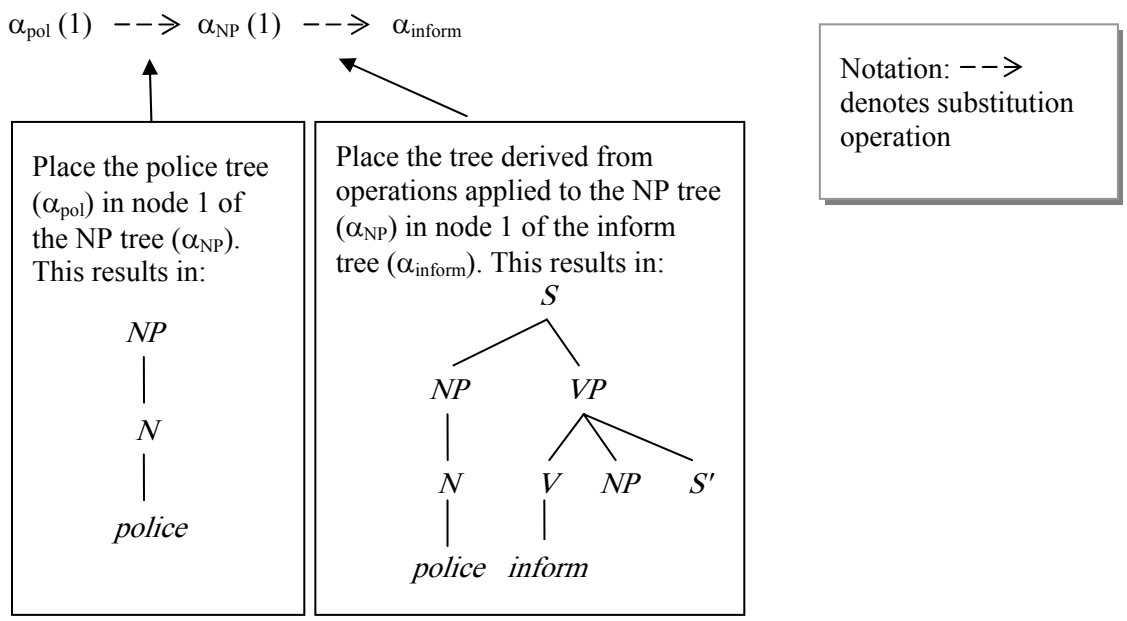
[1] Joshi. A. K., Levy, L. S., and Takahashi, M. Tree adjunct grammar, *Journal of Computer and System Science*, vol. 21, no. 2, 1975.

<u>Substitution operation:</u> The initial tree type can be used to perform operations to obtain other derived trees. The substitution operation involves replacing one non-terminal leaf of an elementary tree by another initial or derived tree whose root has the same label.

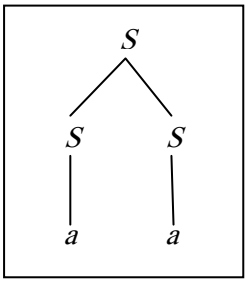
Notation: Gorn numbering of tree nodes will be used to help express operations in certain nodes.



Example:



The first impression is that this is a very similar derivation process to that exhibited by CFGs. However, TAGs generate different tree sets than CFGs do, even though the same sets of sentences are represented. For example the following TAG:



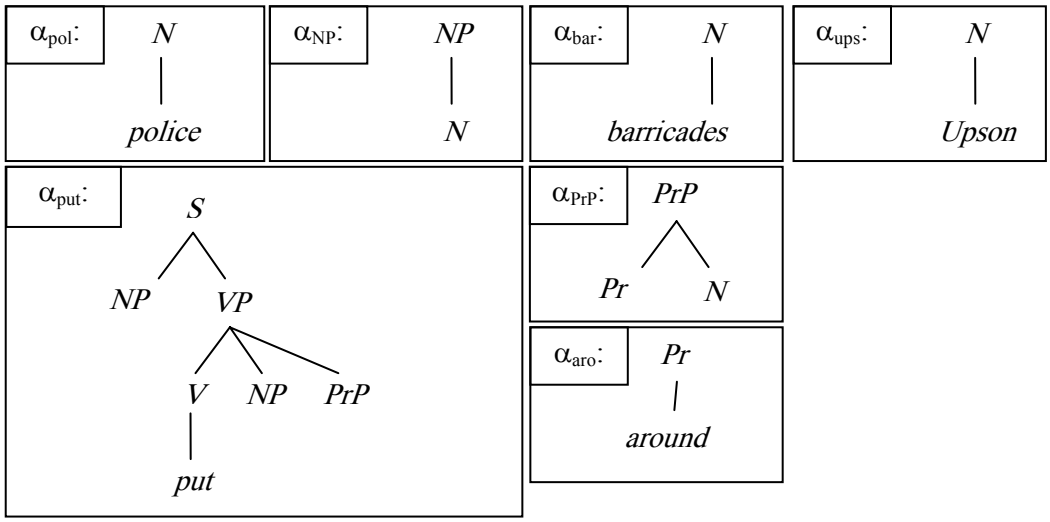
generates only one tree; any CFG generating this tree must generate an infinite number of other trees, too.

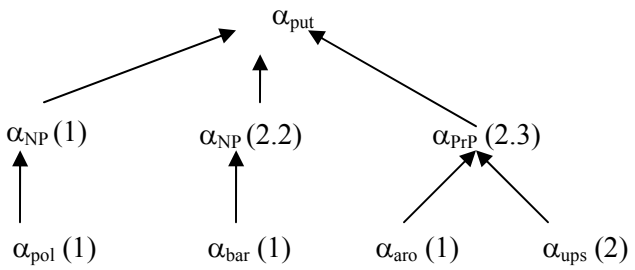
Questions

Question 1

Draw a set of TAG initial trees for the sentence 'police put barricades around Upson'. Do this by first drawing an initial tree for the verb 'put'. Then give the derivation tree describing the derivation of the sentence.

Answer





Question 2

In one sentence, describe the key difference between TAGs (as defined so far in this lecture) and CFGs. How does this difference make TAGs (arguably) more suited to representing the grammar of natural language? Where do Feature-Based CFGs fit in this picture?

Answer

CFGs consist of a series of rules for replacing a single constituent with a sequence of constituents, while TAGs consist of a series of rules (initial trees) for replacing a single node with a subtree. In natural language, constituents of a sentence often have to 'know' what the other constituents in far-away parts of the sentence are; an example is the filler-gap problem we discussed. TAGs help resolve this by providing a way of reducing this distance - after all, if both constituents are part of the same subtree, than the distance between them is zero. FB-CFGs are a mechanism built on top of CFGs which also allow this sort of communication among constituents.

Question 3

Why do we require that for CFGs the sets of terminals and non-terminals are finite? What 'silly' thing could we do if they were allowed to be infinite? Why is this thing silly and what does it tell you about what we want 'non-silly' grammars to be?

Answer

We could write one rule for each allowed natural language sentence. We don't want to do this since it is useless, as the grammar will be infinite hence not something we can run inferences and analyses on a computer in reasonable time. We want grammars to be a representation of the rules of language which are simpler than writing out all the allowed sentences, so we can use them to make computers understand what a piece of natural language is about. An analogy: memorizing the times table is not the same thing as understanding the multiplication operation.