Last Class: Part-of-Speech Tagging
   1. HMM Tagger

Today: Parsing
   1. Grammars and parsing
   2. Top-down and bottom-up parsing
   3. Bottom-up chart parsing

Syntax
syntax: from the Greek *syntaxis*, meaning “setting out together or arrangement.”
Refers to the way words are arranged together.
Why worry about syntax?
   • The boy ate the frog.
   • The frog was eaten by the boy.
   • The frog that the boy ate died.
   • The boy whom the frog was eaten by died.

Syntactic Analysis
Key ideas:
   • constituency: groups of words may behave as a single unit or phrase
   • grammatical relations: refer to the subject, object, indirect object, etc.
   • subcategorization and dependencies: refer to certain kinds of relations between words and phrases, e.g. want can be followed by an infinitive, but find cannot.
All can be modeled by various kinds of grammars that are based on context-free grammars.

Grammars and Parsing
Need a grammar: a formal specification of the structures allowable in the language.
Need a parser: algorithm for assigning syntactic structure to an input sentence.
Sentence
   Beavis ate the cat.

Parse Tree
   \[
   \begin{array}{c}
   S \\
   \downarrow \\
   V \\
   \downarrow \\
   \text{ate} \\
   \downarrow \\
   \text{the} \\
   \downarrow \\
   \text{cat}
   \end{array}
   \]
**CFG example**

CFG’s are also called phrase-structure grammars. Equivalent to Backus-Naur Form (BNF).

1. \( S \rightarrow NP \ VP \)
2. \( VP \rightarrow V \ NP \)
3. \( NP \rightarrow NAME \)
4. \( NP \rightarrow ART \ N \)
5. \( NAME \rightarrow Beavis \)
6. \( V \rightarrow ate \)
7. \( ART \rightarrow the \)
8. \( N \rightarrow cat \)

- CFG’s are powerful enough to describe most of the structure in natural languages.
- CFG’s are restricted enough so that efficient parsers can be built.

---

### Derivations

- If the rule \( A \rightarrow \beta \in P \), and \( \alpha \) and \( \gamma \) are strings in the set \((\Sigma \cup N)^*\), then we say that \( \alpha A \gamma \) directly derives \( \alpha \beta \gamma \), or \( \alpha A \gamma \Rightarrow \alpha \beta \gamma \)
- Let \( \alpha_1, \alpha_2, \ldots, \alpha_m \) be strings in \((\Sigma \cup N)^*\), \( m > 1 \), such that
  \[
  \alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{m-1} \Rightarrow \alpha_m
  \]
  then we say that \( \alpha_1 \) derives \( \alpha_m \) or \( \alpha_1 \Rightarrow^* \alpha_m \)

---

### Language \( L_G \)

The language \( L_G \) generated by a grammar \( G \) is the set of strings composed of terminal symbols that can be derived from the designated start symbol \( S \).

\[
L_G = \{ w | w \in \Sigma^*, S \Rightarrow^* w \}
\]

Parsing: the problem of mapping from a string of words to its parse tree according to a grammar \( G \).
General Parsing Strategies

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Top-Down</th>
<th>Bottom-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $S \rightarrow NP \ VP$</td>
<td>$S \rightarrow NP \ VP$</td>
<td>$\rightarrow NAME \ ate \ the \ cat$</td>
</tr>
<tr>
<td>2. $VP \rightarrow V NP$</td>
<td>$\rightarrow NAME \ VP$</td>
<td>$\rightarrow NAME \ V \ the \ cat$</td>
</tr>
<tr>
<td>3. $NP \rightarrow NAME$</td>
<td>$\rightarrow Beav \ VP$</td>
<td>$\rightarrow NAME \ V \ ART \ cat$</td>
</tr>
<tr>
<td>4. $NP \rightarrow ART \ N$</td>
<td>$\rightarrow Beav \ V \ NP$</td>
<td>$\rightarrow NAME \ V \ ART \ N$</td>
</tr>
<tr>
<td>5. $NAME \rightarrow Beavis$</td>
<td>$\rightarrow Beav \ ate \ NP$</td>
<td>$\rightarrow NP \ V \ ART \ N$</td>
</tr>
<tr>
<td>6. $V \rightarrow ate$</td>
<td>$\rightarrow Beav \ ate \ ART \ N$</td>
<td>$\rightarrow NP \ V \ NP$</td>
</tr>
<tr>
<td>7. $ART \rightarrow the$</td>
<td>$\rightarrow Beav \ ate \ the \ N$</td>
<td>$\rightarrow NP \ VP$</td>
</tr>
<tr>
<td>8. $N \rightarrow cat$</td>
<td>$\rightarrow Beav \ ate \ the \ cat$</td>
<td>$\rightarrow S$</td>
</tr>
</tbody>
</table>

Efficient Parsing

Have the first year PhD students in the computer science department take the Q-exam.

Have the first year PhD students in the computer science department taken the Q-exam?

Chart Parsers

**chart:** data structure that stores partial results of the parsing process in such a way that they can be reused. The chart for an $n$-word sentence consists of:

- $n + 1$ vertices
- a number of edges that connect vertices

Chart Parsing: The General Idea

The process of parsing an $n$-word sentence consists of forming a chart with $n + 1$ vertices and adding edges to the chart one at a time.

- Goal: To produce a complete edge that spans from vertex 0 to $n$ and is of category $S$.
- There is no backtracking.
- Everything that is put in the chart stays there.
- Chart contains all information needed to create parse tree.
**Bottom-UP Chart Parsing Algorithm**

Do until there is no input left:

1. If the agenda is empty, get next word from the input, look up word categories, add to agenda (as constituent spanning two positions).
2. Select a constituent from the agenda: constituent $C$ from $p_1$ to $p_2$.
3. Insert $C$ into the chart from position $p_1$ to $p_2$.
4. For each rule in the grammar of form $X \rightarrow C X_1 \ldots X_n$, add an active edge of form $X \rightarrow C \circ X_1 \ldots X_n$ from $p_1$ to $p_2$.
5. Extend existing edges that are looking for a $C$.
   (a) For any active edge of form $X \rightarrow X_1 \ldots \circ CX_n$ from $p_0$ to $p_1$, add a new active edge $X \rightarrow X_1 \ldots \circ CX_n$ from $p_0$ to $p_2$.
   (b) For any active edge of form $X \rightarrow X_1 \ldots X_n \circ C$ from $p_0$ to $p_1$, add a new (completed) constituent of type $X$ from $p_0$ to $p_2$ to the agenda.

---

**Grammar and Lexicon**

**Grammar:**
1. $S \rightarrow NP \ VP$
2. $NP \rightarrow ART \ N$
3. $NP \rightarrow ART \ ADJ \ N$
4. $VP \rightarrow VN \ P$

**Lexicon:**
- the: ART
- man: N, V
- old: ADJ, N
- boat: N

**Sentence:** 1 The 2 old 3 man 4 the 5 boat 6

---

**Example**

[See .ppt slides]
The old man the boat.

Efficient Parsing

\[ n = \text{sentence length} \]

Time complexity for bottom-up chart parser: \( \bigO(n^3) \)

1. Don’t do twice what you can do once.