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* and *work* 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
CFG example

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

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

- 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.

Slide CS474–5

Slide CS474–6

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 \( \alpha_1 \Rightarrow \alpha_m \)

Slide CS474–7

Slide CS474–8

\[ L_G \]

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>$\rightarrow$ VP</td>
</tr>
<tr>
<td>7. ART $\rightarrow$ the</td>
<td>$\rightarrow$ Beav ate the N</td>
<td>$\rightarrow$ VP</td>
</tr>
<tr>
<td>8. N $\rightarrow$ cat</td>
<td>$\rightarrow$ Beav ate the cat</td>
<td>$\rightarrow$ S</td>
</tr>
</tbody>
</table>

#### A Top-Down Parser

**Input:** CFG grammar, lexicon, sentence to parse  
**Output:** yes/no  
**State of the parse:** (symbol list, position)  

start state: ((S) 1)

### Grammar and Lexicon

**Grammar:**
1. $S \rightarrow NP \ VP$  
2. $NP \rightarrow art \ n$  
3. $NP \rightarrow art \ adj \ n$  
4. $VP \rightarrow v$  
5. $VP \rightarrow v \ NP$  

**Lexicon:**
the: art  
old: adj, n  
man: n, v  
cried: v

1 The 2 old 3 man 4 cried 5
Example

Current state | Backup states
--- | ---
1. ((S) 1) | 8. ((v NP) 3) ((art adj n VP) 1) leads to backtracking...
2. ((NP VP) 1) | 9. ((art adj n VP) 1)
3. ((art n VP) 1) | 10. ((adj n VP) 2)
4. ((n VP) 2) | 11. ((n VP) 3)
5. ((VP) 3) | 12. ((VP) 4)
6. ((v) 3) | 13. ((v) 4) ((v NP) 4)
7. (() 4) | 14. (() 5) ((v NP) 4)
| ((v NP) 3) ((art adj n VP) 1) Backtrack | YES

DONE!

Problems with the Top-Down Parser

1. Only judges grammaticality.
2. Stops when it finds a single derivation.
3. No semantic knowledge employed.
4. No way to rank the derivations.
5. Problems with left-recursive rules.
6. Problems with ungrammatical sentences.

Efficient Parsing

The top-down parser is terribly inefficient.

*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

```
S -> NP . VP
VP -> V NP .
S -> NP VP .
```

Judge Ito scolded the defense.

---

**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.