<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>$\rightarrow \ NAME \ space \ ate \ space \ the \ space \ cat$</td>
</tr>
<tr>
<td>2. $VP \rightarrow V \ NP$</td>
<td>$\rightarrow \ NAME \ VP$</td>
<td>$\rightarrow \ NAME \ space \ V \ space \ the \ space \ cat$</td>
</tr>
<tr>
<td>3. $NP \rightarrow NAME$</td>
<td>$\rightarrow \ Beav \ VP$</td>
<td>$\rightarrow \ NAME \ space \ V \ space \ ART \ space \ cat$</td>
</tr>
<tr>
<td>4. $NP \rightarrow ART \ N$</td>
<td>$\rightarrow \ Beav \ V \ NP$</td>
<td>$\rightarrow \ NAME \ space \ V \ space \ ART \ space \ N$</td>
</tr>
<tr>
<td>5. $NAME \rightarrow Beavis$</td>
<td>$\rightarrow \ Beav \ space \ ate \ space \ NP$</td>
<td>$\rightarrow \ NP \ space \ V \ space \ ART \ space \ N$</td>
</tr>
<tr>
<td>6. $V \rightarrow ate$</td>
<td>$\rightarrow \ Beav \ space \ ate \ space \ ART \ space \ N$</td>
<td>$\rightarrow \ NP \ space \ V \ space \ NP$</td>
</tr>
<tr>
<td>7. $ART \rightarrow the$</td>
<td>$\rightarrow \ Beav \ space \ ate \ space \ the \ space \ N$</td>
<td>$\rightarrow \ NP \ space \ VP$</td>
</tr>
<tr>
<td>8. $N \rightarrow cat$</td>
<td>$\rightarrow \ Beav \ space \ ate \ space \ the \ space \ cat$</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:** \((\text{symbol list}, \text{position})\)

\[
1 \text{ The } 2 \text{ old } 3 \text{ man } 4 \text{ cried } 5
\]

start state: \(((\text{S}) 1)\)
Grammar and Lexicon

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

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

1 The 2 old 3 man 4 cried 5
Algorithm for a Top-Down Parser

\[ PSL \leftarrow (((S) 1)) \]

1. Check for failure. If PSL is empty, return NO.
2. Select the current state, \( C \). \( C \leftarrow \text{pop} \ (PSL) \).
3. Check for success. If \( C = ((()) \ <\text{final-position}>)) \), YES.
4. Otherwise, generate the next possible states.
   (a) \( s_1 \leftarrow \text{first-symbol}(C) \)
   (b) If \( s_1 \) is a lexical symbol and next word can be in that class, create a new state by removing \( s_1 \), updating the word position, and adding it to \( PSL \). (I’ll add to front.)
   (c) If \( s_1 \) is a non-terminal, generate a new state for each rule in the grammar that can rewrite \( s_1 \). Add all to \( PSL \). (Add to front.)
<table>
<thead>
<tr>
<th>Current state</th>
<th>Backup states</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ((S) 1)</td>
<td></td>
</tr>
<tr>
<td>2. ((NP VP) 1)</td>
<td></td>
</tr>
<tr>
<td>3. ((art n VP) 1)</td>
<td>((art adj n VP) 1)</td>
</tr>
<tr>
<td>4. ((n VP) 2)</td>
<td>((art adj n VP) 1)</td>
</tr>
<tr>
<td>5. ((VP) 3)</td>
<td>((art adj n VP) 1)</td>
</tr>
<tr>
<td>6. ((v) 3)</td>
<td>((v NP) 3) ((art adj n VP) 1)</td>
</tr>
<tr>
<td>7. (() 4)</td>
<td>((v NP) 3) ((art adj n VP) 1) Backtrack</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>8.</td>
<td></td>
</tr>
<tr>
<td>( ((v \ NP) \ 3) )</td>
<td>( ((\text{art} \ \text{adj} \ n \ \text{VP}) \ 1) ) leads to backtracking</td>
</tr>
</tbody>
</table>
|   | ...
| 9. | \( ((\text{art} \ \text{adj} \ n \ \text{VP}) \ 1) \)
| 10. | \( ((\text{adj} \ n \ \text{VP}) \ 2) \)
| 11. | \( ((n \ \text{VP}) \ 3) \)
| 12. | \( ((\text{VP}) \ 4) \)
| 13. | \( ((v) \ 4) \)
| 14. | \( ((v \ NP) \ 4) \)
| YES | \( ((v \ NP) \ 4) \)

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

S→ NP VP .
```

```
Judge Ito scolded the defense.
```

```
0 Judge 1 Ito 2 scolded 3 the 4 defense. 5
```

```
S→ NP VP

VP→V NP .
```
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 $X \rightarrow X_1 \ldots C \circ X_n$ 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. $VP \rightarrow V \ NP$

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.
Bottom-up Chart Parser

Is it any less naive than the top-down parser?

1. Only judges grammaticality. [fixed]
2. Stops when it finds a single derivation. [fixed]
3. No semantic knowledge employed.
4. No way to rank the derivations.
5. Problems with ungrammatical sentences. [better]
6. Terribly inefficient.