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 Parse Tree

Beavis ate the cat.

CFG example

CFG’s are also called phase-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.
CFG’s

A context free grammar consists of:
1. a set of non-terminal symbols \( N \)
2. a set of terminal symbols \( \Sigma \) (disjoint from \( N \))
3. a set of productions, \( P \), each of the form \( A \rightarrow \alpha \), where \( A \) is a non-terminal and \( \alpha \) is a string of symbols from the infinite set of strings \( (\Sigma \cup N)^* \)
4. a designated start symbol \( S \)

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 \( \alpha_1 \Rightarrow \alpha_m \)

\( L_G \)

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 \( S \).

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

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 → NP VP</td>
<td>S → NP VP</td>
<td>→ NAME ate the cat</td>
</tr>
<tr>
<td>2. VP → V NP</td>
<td>→ NAME VP</td>
<td>→ NAME V the cat</td>
</tr>
<tr>
<td>3. NP → NAME</td>
<td>→ Beav VP</td>
<td>→ NAME V ART cat</td>
</tr>
<tr>
<td>4. NP → ART N</td>
<td>→ Beav V NP</td>
<td>→ NAME V ART N</td>
</tr>
<tr>
<td>5. NAME → Beavis</td>
<td>→ Beav ate NP</td>
<td>→ NP V ART N</td>
</tr>
<tr>
<td>6. V → ate</td>
<td>→ Beav ate ART N</td>
<td>→ NP V NP</td>
</tr>
<tr>
<td>7. ART → the</td>
<td>→ Beav ate the N</td>
<td>→ NP VP</td>
</tr>
<tr>
<td>8. N → cat</td>
<td>→ Beav ate the cat</td>
<td>→ 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)

1. The 2 old 3 man 4 cried 5

start state: ((S) 1)

---

**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 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.)

---

**Grammar and Lexicon**

**Grammar:**

1. \( S \rightarrow \text{NP VP} \)
2. \( \text{NP} \rightarrow \text{art n} \)
3. \( \text{NP} \rightarrow \text{art adj n} \)
4. \( \text{VP} \rightarrow \text{v} \)
5. \( \text{VP} \rightarrow \text{v NP} \)

**Lexicon:**

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

1. The 2 old 3 man 4 cried 5

---

**Example**

<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>
</tbody>
</table>
8. ((v NP) 3) leads to backtracking
...
9. ((art adj n VP) 1)
10. ((adj n VP) 2)
11. ((n VP) 3)
12. ((VP) 4)
13. ((v) 4)
14. () 5)
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.

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

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