Parsing
1. Grammars and parsing
2. Top-down and bottom-up parsing
3. Chart parsers
4. Bottom-up chart parsing
5. The Earley Algorithm

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 phrase-structure grammars. Equivalent to Backus-Naur Form (BNF).

1. S → NP VP
2. VP → V NP
3. NP → NAME
4. NP → ART N
5. NAME → Beavis
6. V → ate
7. ART → the
8. N → 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
  \[
  \alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \ldots \Rightarrow \alpha_{m-1} \Rightarrow \alpha_m,
  \]
  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 \( S \).

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

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)

start state: ((S) 1)

Algorithm for a Top-Down Parser

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

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.)
### 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

![Diagram of a chart for an n-word sentence](image)

Chart Parsers: 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 $X \rightarrow X_1 \ldots X_n$, 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 $X \rightarrow X_1 \ldots \circ C X_n$ 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]

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.
Efficient Parsing

\( n = \text{sentence length} \)

Time complexity for naive algorithm: exponential in \( n \)

Time complexity for bottom-up chart parser: \( \mathcal{O}(n^3) \)

Options for improving efficiency:
1. Don’t do twice what you can do once.
2. Don’t represent distinctions that you don’t need.
   
   Fall leaves fall and spring leaves spring.
3. Don’t do once what you can avoid altogether.
   
   The can holds the water. (“can”: AUX, V, N)

--

Earley Algorithm: Top-Down Chart Parser

For all S rules of the form \( S \rightarrow X_1 \ldots X_k \), add a (top-down) edge from 1 to 1 labeled: \( S \rightarrow \circ X_1 \ldots X_k \).

Do until there is no input left:
1. If the agenda is empty, look up word categories for next word, add to agenda.
2. Select a constituent from the agenda: constituent \( C \) from \( p_1 \) to \( p_2 \).
3. Using the (bottom-up) edge extension algorithm, combine \( C \) with every active edge on the chart (adding \( C \) to chart as well). Add any new constituents to the agenda.
4. For any active edges created in Step 3, add them to the chart using the top-down edge introduction algorithm.

--

Grammar and Lexicon

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. S → NP VP</td>
<td>the: ART</td>
</tr>
<tr>
<td>2. NP → ART ADJ N</td>
<td>large: ADJ</td>
</tr>
<tr>
<td>3. NP → ART N</td>
<td>can: N, AUX, V</td>
</tr>
<tr>
<td>4. NP → ADJ N</td>
<td>hold: N, V</td>
</tr>
<tr>
<td>5. VP → AUX VP</td>
<td>water: N, V</td>
</tr>
<tr>
<td>6. VP → V NP</td>
<td></td>
</tr>
</tbody>
</table>

Sentence: 1. The 2 large 3 can 4 can 5 hold 6 water 7