Last Class: Parsing

1. Grammars and parsing
2. Top-down and bottom-up parsing
3. A top-down parser

Today: Parsing

1. Bottom-up chart parsing
2. Earley algorithm

---

**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>

---

**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 ← 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 \( PSL \).
   (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 \( PSL \).
Judge Ito scolded the defense.

---

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 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.

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 constituent of type X from $p_0$ to $p_2$ to the agenda.

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.

2. Select a constituent from the agenda: constituent C from $p_1$ to $p_2$.

3. 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 $p_2$.

4. Insert C into the chart from position $p_1$ to $p_2$.

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 
old: ADJ, N 
man: N, V 
boat: N

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

- \( n = \) 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)

---

**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. 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.
**Top-down edge introduction.**

To add an edge \( S \rightarrow C_1 \ldots C_j \ldots C_n \) ending at position \( j \):

For each rule in the grammar of form \( C_i \rightarrow X_1 \ldots X_k \),

recursively add the new edge \( C_i \rightarrow oX_1 \ldots X_k \) from \( j \) to \( j \).

---

**Grammar and Lexicon**

**Grammar**

1. \( S \rightarrow NP \ VP \)
2. \( NP \rightarrow ART \ ADJ \ N \)
3. \( NP \rightarrow ART \ N \)
4. \( NP \rightarrow ADJ \ N \)
5. \( VP \rightarrow AUX \ VP \)
6. \( VP \rightarrow V \ NP \)

**Lexicon**

- the: ART
- large: ADJ
- can: N, AUX, V
- hold: N, V
- water: N, V

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