Natural Language Processing
CMSC 723 (spring, 2001)
April 04, 2001

- Review CFG's
- Top-Down Parsing, Bottom-Up Parsing
- Top-Down with Bottom-up Filtering
- Ambiguity, Recursion
- Repeated parsing of substructures
- Dynamic Programming
- Dotted Rule Notation

Example Context-Free Grammar and
Example sentence

[Figure 10.1]

Parsing as search

- **Top down**: parser searches for a parse tree by trying to build from the root node S down to the leaves.

- **Bottom up**: parser starts with words of input and tries to build trees from the words up, applying rules from grammar one at a time.
<table>
<thead>
<tr>
<th>Top-Down Parsing</th>
<th>Basic Top-Down Parser</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the goal?</td>
<td>[Figure 10.6]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expanding Top-Down Search Space</th>
<th>Bottom-Up Parsing</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Figure 10.3]</td>
<td>What is the primary consideration?</td>
</tr>
</tbody>
</table>
Bottom-Up parsing

[Figure 10.4]

Top-Down and Bottom-Up Combined

Many ways to combine top-down expectations with bottom-up data.

Most popular: use one method as the basic search control strategy and then other method to filter out "bad" structures.

Comparing Top-Down and Bottom-Up parsing

What are the advantages and disadvantages of TD vs. BU parsing?

• TOP DOWN
  Advantages:
  Disadvantages:

• BOTTOM UP
  Advantages:
  Disadvantages:

Combining Top-Down with Bottom-Up Filtering

Digression: Search Strategies

1. Parallel

2. Depth-First
Search Control Issues

Additional Digression:

- Choosing which node in the tree to expand next

- Choosing which of the applicable grammar rule to try

Top-Down Parsing Example

[Figure 10.7a]

Top-Down Parsing Example

[Figure 10.7b]

Top-Down Parsing Example

[Figure 10.7c]
Top-Down Parsing Example

[Figure 10.7d]

Ambiguity

Two types of ambiguity:

- Local ambiguity: locally reasonable, but eventually leads nowhere. Example: “Book that flight”

- Global ambiguity: multiple parses for the same input. Example: [Figure 10.13]

Why would it be beneficial to add in Bottom-up filtering?

- Ambiguity
- Left recursion
- Repeated parsing of subtrees

Left Recursion: Immediate

NP → NP PP
VP → VP PP
S → S and S
NP → NP and NP
Left Recursion: Indirect

Abstractly...

\[ A \rightarrow BC \]
\[ B \rightarrow DE \]
\[ D \rightarrow AF \]

What's an example?

Rule Ordering

Basic idea...

Bad:
\[ \text{NP} \rightarrow \text{NP PP} \]
\[ \text{NP} \rightarrow \text{Det Nominal} \]

Better alternative?

Solutions

- Rule ordering
- Don't use recursive rules
- Limit the depth of recursion
- Don't use top-down parsing

Grammar Rewriting

Rewrite left-recursive grammar as weakly equivalent non-recursive one.
Grammar Rewriting Example

NP → NP PP
NP → Det Nominal

[\[NP [NP the book]
  [PP on [NP [NP the table]]]
  [PP in [NP [NP the yard]]]
  [PP of [NP the house]]]]

[\[NP the book
  [PP on [NP the table]
  [Det [PP the yard]]
  [Det of [NP the house]]]
  [Det]]]

[Det]]

Ambiguity

- Rely on semantics
- Rely on probabilities
- Both

Depth Bound

To use a depth-bound, there are many different approaches, e.g., setting an arbitrary or analyti-
cally derived bound.

Adding Bottom-Up Filtering

Improvement: Parser should eliminate any grammar rule if the current input cannot serve as the first word along the left edge of some derivation from this rule.

<table>
<thead>
<tr>
<th>Category</th>
<th>Left-Corners</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>Det, Proper-Noun, Aux, Verb</td>
</tr>
<tr>
<td>Nominal</td>
<td>Det, Proper-Noun</td>
</tr>
<tr>
<td>VP</td>
<td>Noun, Verb</td>
</tr>
</tbody>
</table>

Filtering with left corners: Don’t consider any expansion where the current input cannot serve as the left-corner of that expansion.
*Invariants*

Sentence: “a flight from Indianapolis to Houston on TWA”

NP $\rightarrow$ Det Nominal
NP $\rightarrow$ NP PP
NP $\rightarrow$ Proper-Noun

*Invariants cont.*

[Figure 10.14b]

[Figure 10.14a]

[Figure 10.14c]
**Dynamic Programming**

We want an algorithm that fills a table with solutions to subproblems that:

- Does not do repeated work
- Does top-down search with bottom-up filtering (sort of)
- Solves the left-recursion problem
- Solves an exponential problem in \(O(n^3)\) time.

**States**

\[
S \rightarrow \bullet \text{ VP} \\
\text{NP} \rightarrow \text{Det} \bullet \text{ Nominal} \\
\text{VP} \rightarrow \text{ V NP} \bullet
\]

**Dynamic Programming and Parsing**

Use a table of size \(n+1\). The table entries sit in the gaps between the words:

- Completed constituents
- In-progress constituents
- Predicted constituents

**States cont.**

Keep track of:

- What word it is currently processing.
- Where it is in the processing of the current rule.
- Where it should return to when done w/ current rule.