Speech and Language Processing

Chapter 12
Syntactic Parsing

Today

- Parsing with CFGs
  - Bottom-up, top-down
  - Ambiguity
  - CKY parsing
  - (Earley)
  - Shallow
Parsing

- Parsing with CFGs refers to the task of assigning proper trees to input strings.
- Proper here means a tree that covers all and only the elements of the input and has an S at the top.
- It doesn’t actually mean that the system can select the correct tree from among all the possible trees.

Parsing

- As with everything of interest, parsing involves a search which involves the making of choices.
- We’ll start with some basic (meaning bad) methods before moving on to the one that you need to know.
For Now

- Assume...
  - You have all the words already in some buffer
  - The input isn’t POS tagged
  - We won’t worry about morphological analysis
  - All the words are known

- These are all problematic in various ways, and would have to be addressed in real applications.

Top-Down Search

- Since we’re trying to find trees rooted with an S (Sentences), why not start with the rules that give us an S.

- Then we can work our way down from there to the words.

- “Book that flight”
Bottom-Up Parsing

- Of course, we also want trees that cover the input words. So we might also start with trees that link up with the words in the right way.
- Then work your way up from there to larger and larger trees.

"The old dog the footsteps of the young."

<table>
<thead>
<tr>
<th>S</th>
<th>NP VP</th>
<th>VP</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Aux NP VP</td>
<td>VP</td>
<td>V PP</td>
</tr>
<tr>
<td>S</td>
<td>-&gt; VP</td>
<td>PP</td>
<td>-&gt; Prep NP</td>
</tr>
<tr>
<td>NP</td>
<td>-&gt; Det Nom</td>
<td>N</td>
<td>old</td>
</tr>
<tr>
<td>NP</td>
<td>-&gt; PropN</td>
<td>V</td>
<td>dog</td>
</tr>
<tr>
<td>Nom</td>
<td>-&gt; Adj N</td>
<td>Aux</td>
<td>does</td>
</tr>
<tr>
<td>Nom</td>
<td>-&gt; N</td>
<td>Prep</td>
<td>from</td>
</tr>
<tr>
<td>Nom</td>
<td>-&gt; N Nom</td>
<td>PropN</td>
<td>Fido</td>
</tr>
<tr>
<td>Nom</td>
<td>-&gt; Nom PP</td>
<td>Det</td>
<td>that</td>
</tr>
<tr>
<td>VP</td>
<td>-&gt; V NP</td>
<td>Adj</td>
<td>-&gt; old</td>
</tr>
</tbody>
</table>
Top-Down and Bottom-Up

- Top-down
  - Only searches for trees that can be answers (i.e. S’s)
  - But also suggests trees that are not consistent with any of the words

- Bottom-up
  - Only forms trees consistent with the words
  - But suggests trees that make no sense globally

Control

- Of course, in both cases we left out how to keep track of the search space and how to make choices
  - Which node to try to expand next
  - Which grammar rule to use to expand a node

- One approach is called backtracking.
  - Make a choice, if it works out then fine
  - If not then back up and make a different choice
Problems

- Even with the best filtering, backtracking methods are doomed because of two inter-related problems
  - Ambiguity
  - Shared subproblems

Ambiguity
Example types of ambiguity

- POS
- Attachment
  - PP
  - Coordination (*old dogs and cats*)

Shared Sub- Problems

- No matter what kind of search (top-down or bottom-up or mixed) that we choose.
  - We don’t want to redo work we’ve already done.
  - Unfortunately, naïve backtracking will lead to duplicated work.
Shared Sub-Problems

- Consider
  - A flight from Indianapolis to Houston on TWA

- Assume a top-down parse making choices among the various Nominal rules.
- In particular, between these two
  - Nominal -> Noun
  - Nominal -> Nominal PP
- Statically choosing the rules in this order leads to the following bad results...
Shared Sub-Problems

NP
  /\  \
Det Nominal
  |   |
a Noun
    |
flight...

Shared Sub-Problems

NP
  /\  \
Det Nominal
  |   |
a   |
Nominal PP
  |   |
Noun flight
    |
from Indianapolis...
Shared Sub-Problem

NP
  └── Det
    └── a

Nominal
  └── PP
    └── to Houston...

Nominal
  └── PP
    └── from Indianapolis

Nominal
  └── PP
    └── on TWA

Nominal
  └── PP
    └── to Houston

Nominal
  └── PP
    └── from Indianapolis

Nominal
  └── PP
    └── flight

Nominal
  └── PP
    └── flight
Dynamic Programming

- DP search methods fill tables with partial results and thereby
  - Avoid doing avoidable repeated work
  - Solve exponential problems in polynomial time
  - Efficiently store ambiguous structures with shared sub-parts.
- Two approaches roughly correspond to bottom-up and top-down approaches.
  - CKY
  - Earley

CKY Parsing

- First we’ll limit our grammar to epsilon-free, binary rules (more later)
- Consider the rule $A \rightarrow BC$
  - If there is an $A$ somewhere in the input then there must be a $B$ followed by a $C$ in the input.
  - If the $A$ spans from $i$ to $j$ in the input then there must be some $k$ st. $i < k < j$
    - I.e. The $B$ splits from the $C$ someplace.
Problem

- What if your grammar isn’t binary?
  - As in the case of the TreeBank grammar?
- Convert it to binary... any arbitrary CFG can be rewritten into Chomsky-Normal Form automatically.
- What does this mean?
  - The resulting grammar accepts (and rejects) the same set of strings as the original grammar.
  - But the resulting derivations (trees) are different.

Problem

- More specifically, we want our rules to be of the form
  
  \[ A \rightarrow BC \]

  Or

  \[ A \rightarrow w \]

That is, rules can expand to either 2 non-terminals or to a single terminal.
### Binarization Intuition

- Eliminate chains of unit productions.
- Introduce new intermediate non-terminals into the grammar that distribute rules with length > 2 over several rules.

- So... \( S \rightarrow A \; B \; C \) turns into

\[
S \rightarrow X \; C \quad \text{and} \quad X \rightarrow A \; B
\]

Where \( X \) is a symbol that doesn’t occur anywhere else in the the grammar.

---

### Sample L1 Grammar

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow NP ; VP )</td>
<td>Det ( \rightarrow ) that</td>
</tr>
<tr>
<td>( S \rightarrow Aux ; NP ; VP )</td>
<td>Noun ( \rightarrow ) book</td>
</tr>
<tr>
<td>( S \rightarrow VP )</td>
<td>Verb ( \rightarrow ) book</td>
</tr>
<tr>
<td>( NP \rightarrow Pronoun )</td>
<td>Pronoun ( \rightarrow ) I</td>
</tr>
<tr>
<td>( NP \rightarrow Proper-Noun )</td>
<td>Proper-Noun ( \rightarrow ) Houston</td>
</tr>
<tr>
<td>( NP \rightarrow Det ; Nominal )</td>
<td>Aux ( \rightarrow ) does</td>
</tr>
<tr>
<td>Nominal ( \rightarrow ) Noun</td>
<td>Preposition ( \rightarrow ) from</td>
</tr>
<tr>
<td>Nominal ( \rightarrow ) Nominal Noun</td>
<td></td>
</tr>
<tr>
<td>Nominal ( \rightarrow ) Nominal PP</td>
<td></td>
</tr>
<tr>
<td>VP ( \rightarrow ) Verb</td>
<td></td>
</tr>
<tr>
<td>VP ( \rightarrow ) Verb NP</td>
<td></td>
</tr>
<tr>
<td>VP ( \rightarrow ) Verb NP PP</td>
<td></td>
</tr>
<tr>
<td>VP ( \rightarrow ) Verb PP</td>
<td></td>
</tr>
<tr>
<td>VP ( \rightarrow ) VP PP</td>
<td></td>
</tr>
<tr>
<td>PP ( \rightarrow ) Preposition NP</td>
<td></td>
</tr>
</tbody>
</table>
### CNF Conversion

<table>
<thead>
<tr>
<th>$L_1$ Grammar</th>
<th>$L_1$ in CNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>$S \rightarrow NP \ VP$</td>
</tr>
<tr>
<td>$S \rightarrow Aux \ NP \ VP$</td>
<td>$S \rightarrow X1 \ VP$</td>
</tr>
<tr>
<td>$X1 \rightarrow Aux \ NP$</td>
<td>$S \rightarrow book \mid include \mid prefer$</td>
</tr>
<tr>
<td>$S \rightarrow VP$</td>
<td>$S \rightarrow Verb \ NP$</td>
</tr>
<tr>
<td>$S \rightarrow X2 \ PP$</td>
<td>$S \rightarrow Verb \ PP$</td>
</tr>
<tr>
<td>$S \rightarrow VP \ PP$</td>
<td>$S \rightarrow VP \ PP$</td>
</tr>
<tr>
<td>$NP \rightarrow Pronoun$</td>
<td>$NP \rightarrow I \mid she \mid me$</td>
</tr>
<tr>
<td>$NP \rightarrow Proper \ Noun$</td>
<td>$NP \rightarrow TWA \mid Houston$</td>
</tr>
<tr>
<td>$NP \rightarrow Det \ Nominal$</td>
<td>$NP \rightarrow Det \ Nominal$</td>
</tr>
<tr>
<td>$Nominal \rightarrow Noun$</td>
<td>$Nominal \rightarrow book \mid flight \mid meal \mid money$</td>
</tr>
<tr>
<td>$Nominal \rightarrow Nominal \ Noun$</td>
<td>$Nominal \rightarrow Nominal \ Noun$</td>
</tr>
<tr>
<td>$Nominal \rightarrow Nominal \ PP$</td>
<td>$Nominal \rightarrow Nominal \ PP$</td>
</tr>
<tr>
<td>$VP \rightarrow Verb$</td>
<td>$VP \rightarrow book \mid include \mid prefer$</td>
</tr>
<tr>
<td>$VP \rightarrow Verb \ NP$</td>
<td>$VP \rightarrow Verb \ NP$</td>
</tr>
<tr>
<td>$VP \rightarrow Verb \ NP \ PP$</td>
<td>$VP \rightarrow X2 \ PP$</td>
</tr>
<tr>
<td>$X2 \rightarrow Verb \ NP$</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow Verb \ PP$</td>
<td>$VP \rightarrow Verb \ PP$</td>
</tr>
<tr>
<td>$VP \rightarrow VP \ PP$</td>
<td>$VP \rightarrow VP \ PP$</td>
</tr>
<tr>
<td>$PP \rightarrow Preposition \ NP$</td>
<td>$PP \rightarrow Preposition \ NP$</td>
</tr>
</tbody>
</table>

### CKY

- So let’s build a table so that an A spanning from $i$ to $j$ in the input is placed in cell $[i,j]$ in the table.

- So a non-terminal spanning an entire string will sit in cell $[0, n]$
  - Hopefully an $S$

- If we build the table bottom-up, we’ll know that the parts of the A must go from $i$ to $k$ and from $k$ to $j$, for some $k$. 