Parsing

Budditha Hettige
Department of Computer Engineering
Introduction

• To parse a string according to a grammar means to reconstruct the production tree (or trees) that indicate how the given string can be produced from the given grammar.
A miniature English grammar and lexicon

<table>
<thead>
<tr>
<th>Production Rule</th>
<th>Production Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>Det → that</td>
</tr>
<tr>
<td>S → Aux NP VP</td>
<td>Noun → book</td>
</tr>
<tr>
<td>S → VP</td>
<td>Verb → book</td>
</tr>
<tr>
<td>NP → Det Nominal</td>
<td>Aux → does</td>
</tr>
<tr>
<td>Nominal → Noun</td>
<td>Prep → from</td>
</tr>
<tr>
<td>Nominal → Noun Nominal</td>
<td>Proper-Noun → Houston</td>
</tr>
<tr>
<td>NP → Proper-Noun</td>
<td></td>
</tr>
<tr>
<td>VP → Verb</td>
<td>Nominal → Nominal PP</td>
</tr>
<tr>
<td>VP → Verb NP</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 10.2** A miniature English grammar and lexicon.
Problem: Ambiguity

1. \( \text{Sum} \rightarrow \text{Digit} \)  \( \{ A_0 := A_1 \} \)
2. \( \text{Sum} \rightarrow \text{Sum} + \text{Sum} \)  \( \{ A_0 := A_1 + A_3 \} \)
3a. \( \text{Digit} \rightarrow 0 \)  \( \{ A_0 := 0 \} \)
...
3j. \( \text{Digit} \rightarrow 9 \)  \( \{ A_0 := 9 \} \)
Ambiguity

1. $\text{Sum}_s \rightarrow \text{Digit}$  \{ $A_0 := A_1$ \}
2. $\text{Sum} \rightarrow \text{Sum} + \text{Sum}$  \{ $A_0 := A_1 + A_3$ \}
3a. $\text{Digit} \rightarrow 0$  \{ $A_0 := 0$ \}
   ...
3j. $\text{Digit} \rightarrow 9$  \{ $A_0 := 9$ \}
Top-Down Parsing

• A top-down parser searches for a parse tree by trying to build from the root node $S$ down to the leaves.
Linearization of the Parse Tree

• a parser can produce a list of rule numbers instead, which means that it linearizes the parse tree

• Three main ways to linearize a tree.
  – prefix, postfix and infix.

leftmost: 2 2 1 3c 1 3e 1 3a

rightmost: 3c 1 3e 1 2 3a 1 2

left-corner: (((3c)1) 2 ((3e)1)) 2 ((3a)1)
Two Ways to Parse a Sentence

- Suppose we have the monotonic grammar for the language \(a^n b^n c^n\) from sentence is \(aabbcc\).

\[
S_s \rightarrow aSQ \\
S \rightarrow abc \\
bQc \rightarrow bbcc \\
cQ \rightarrow Qc
\]
Parse a Sentence
Bottom-Up Parsing
Top-down Parsers

- Recursive descent parser
- LL parser
- Early Parser,
- X-SAIGA parser
Bottom-up Parsing

• **Bottom-up** parsing is the earliest known parsing algorithm

• In bottom-up parsing, the parser starts with the words of the input, and tries to build trees from the words up, again by applying rules from the grammar one at a time.
Example
Problem with Top-down Parsing

- left-recursion,
- Ambiguity
- Inefficient reparsing of sub trees
Prolog based Parsing

Prolog grammar parser generator

\[ S \rightarrow a\ S \\
S \rightarrow B \\
B \rightarrow b\ C \\
C \rightarrow b \]

Sample Code

\[ s \rightarrow np, \ vp. \quad /* \text{sentence} */ \]
\[ np \rightarrow pn. \quad /* \text{noun phrase} */ \]
\[ np \rightarrow d, n, \text{rel.} \]
\[ vp \rightarrow tv, np. \quad /* \text{verb phrase} */ \]
\[ vp \rightarrow iv. \]
Prolog Sample

rel --> []. /* relative clause */
rel --> rpn, vp.

pn --> [PN], {pn(PN)}. /* proper noun */
  pn(mary).
  pn(henry).

rpn --> [RPN], {rpn(RPN)}. /* relative pronoun */
  rpn(that).
  rpn(which).
  rpn(who).

iv --> [IV], {iv(IV)}. /* intransitive verb */
  iv(runs).
  iv(sits).
d --> [DET], {d(DET)}. /* determiner */
d(a).
d(the).

n --> [N], {n(N)}. /* noun */
n(book).
n(girl).
n(boy).

tv --> [TV], {tv(TV)}. /* transitive verb */
tv(gives).
tv(reads).
• DCG rules can contain arguments, using auxiliary variables
• is to force number agreement for subject and verb

\[
\begin{align*}
\text{s(s(NP,VP))} & \rightarrow \text{np(Num,NP), vp(Num,VP)}. \\
\text{np(Num,np(PN))} & \rightarrow \text{pn(Num,PN)}. \\
\text{np(Num,NP)} & \rightarrow \\
& \quad \text{d(Det),} \\
& \quad \text{n(Num,N),} \\
& \quad \text{rel(Num,Rel),} \\
& \quad \text{\{build_np(Det,N,Rel,NP)\}.} /\* \text{embedded Prolog goal */}
\end{align*}
\]
Prolog Code

?- s([the,boy,who,sits,reads,a,book],[]).
yes
?- s([henry,reads],[]).
No
Example

?- s(X, [the, boy, who, sits, reads, the, book], []).  
X = s(np(d(the), n(boy), rel(rpn(who), vp(iv(sits)))), vp(tv(reads), np(d(the), n(book )))).
Sample

?- np([the, boy, who, sits], [])