Context-Free Grammar

CSCI-GA.2590 – Lecture 3

Ralph Grishman
A Grammar Formalism

• We have informally described the basic constructs of English grammar

• Now we want to introduce a formalism for representing these constructs
  – a formalism that we can use as input to a parsing procedure
Context-Free Grammar

• A context-free grammar consists of
  – a set of non-terminal symbols $A, B, C, \ldots \in N$
  – a set of terminal symbols $a, b, c, \ldots \in T$
  – a start symbol $S \in N$
  – a set of productions $P$ of the form $N \rightarrow (N \cup T)^*$
A Simple Context-Free Grammar

A simple CFG:

S \rightarrow NP \text{ VP}

NP \rightarrow \text{ cats}

NP \rightarrow \text{ the cats}

NP \rightarrow \text{ the old cats}

NP \rightarrow \text{ mice}

VP \rightarrow \text{ sleep}

VP \rightarrow \text{ chase NP}
Derivation and Language

If $A \rightarrow \beta$ is a production of the grammar, we can rewrite

$$\alpha A \gamma \rightarrow \alpha \beta \gamma$$

A derivation is a sequence of rewrite operations

$$\rightarrow \ldots \rightarrow \ldots \rightarrow$$

NP VP $\rightarrow$ cats VP $\rightarrow$ cats chase NP

The language generated by a CFG is the set of strings (sequences of terminals) which can be derived from the start symbol

$S \rightarrow \ldots \rightarrow \ldots \rightarrow T^*$

$S \rightarrow$ NP VP $\rightarrow$ cats VP $\rightarrow$ cats chase NP $\rightarrow$ cats chase mice
Preterminals

It is convenient to include a set of symbols called *preterminals* (corresponding to the parts of speech) which can be directly rewritten as terminals (words). This allows us to separate the productions into a set which generates sequences of preterminals (the “grammar”) and those which rewrite the preterminals as terminals (the “dictionary”).
A Grammar with Preterminals

grammar:
\[
  S \rightarrow NP \ VP \\
  NP \rightarrow N \\
  NP \rightarrow ART \ N \\
  NP \rightarrow ART \ ADJ \ N \\
  VP \rightarrow V \\
  VP \rightarrow V \ NP
\]
dictionary:
\[
  N \rightarrow \text{cats} \\
  N \rightarrow \text{mice} \\
  ADJ \rightarrow \text{old} \\
  DET \rightarrow \text{the} \\
  V \rightarrow \text{sleep} \\
  V \rightarrow \text{chase}
\]
Grouping Alternates

- To make the grammar more compact, we group productions with the same left-hand side:
  \[
  S \rightarrow NP \ VP \\
  NP \rightarrow N \mid ART \ N \mid ART \ ADJ \ N \\
  VP \rightarrow V \mid V \ NP
  \]
• A grammar can be used to
  – generate
  – recognize
  – parse

• Why parse?
  – parsing assigns the sentence a structure that may be helpful in determining its meaning
vs Finite State Language

• CFGs are more powerful than finite-state grammars (regular expressions)
  – FSG cannot generate center embeddings
    \[ S \rightarrow (S) \mid x \]
  – even if FSG can capture the language, it may be unable to assign the nested structures we want
A slightly bigger CFG

sentence → np vp
np → ngroup | ngroup pp
ngroup → n | art n | art adj n
vp → v | v np | v vp | v np pp (auxilliary)
pp → p np (pp = prepositional phrase)
Ambiguity

- Most sentences will have more than one parse
- Generally different parses will reflect different meanings ...
  “I saw the man with a telescope.”

Can attach pp (“with a telescope”) under np or vp
A CFG with just 2 nonterminals

S \rightarrow \text{NP} \ V \mid \text{NP} \ V \ \text{NP}

\text{NP} \rightarrow \text{N} \mid \text{ART} \ \text{NOUN} \mid \text{ART} \ \text{ADJ} \ N

use this for tracing our parsers
Top-down parser

repeat
• expand leftmost non-terminal using first production (save any alternative productions on backtrack stack)
• if we have matched entire sentence, quit (success)
• if we have generated a terminal which doesn't match sentence, pop choice point from stack (if stack is empty, quit (failure))
Top-down parser

0: S

the cat chases mice
Top-down parser

0: S
  1: NP
  2: V

backtrack table
0: S → NP  V  NP

cat chases mice

the
Top-down parser

0: S
1: NP
2: V
3: N

backtrack table
0: S → NP  V  NP
1: NP → ART  ADJ  N
1: NP → ART  N

the cat chases mice
Top-down parser

backtrack table
0: S → NP V NP
1: NP → ART ADJ N

the cat chases mice
Top-down parser

0: S
  1: NP
    3: ART
    4: ADJ
    5: N
  2: V

backtrack table
0: S → NP V NP

the cat chases mice
Top-down parser

0: S
  1: NP
  2: V
  3: NP

the cat chases mice

backtrack table
Top-down parser

```
the cat chases mice
```

backtrack table
1: NP → ART ADJ N
1: NP → ART N
Top-down parser

```
0: S
  1: NP
    4: ART
      5: N
  2: V
  3: NP
```

backtrack table
1:NP → ART ADJ N

the cat chases mice
Top-down parser

backtrack table
1: NP → ART ADJ N
3: NP → ART ADJ N
3: NP → ART N

parse!

the cat chases mice
Bottom-up parser

- Builds a table

<table>
<thead>
<tr>
<th>symbol</th>
<th>start</th>
<th>end</th>
<th>constituents</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

where each row represents a parse tree node spanning the words from \textit{start} up to \textit{end}
Bottom-up parser

• We initialize the table with the parts-of-speech of each word ...

<table>
<thead>
<tr>
<th>symbol</th>
<th>start</th>
<th>end</th>
<th>constituents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ART</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>V</td>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>N</td>
<td>3</td>
<td>4</td>
<td>-</td>
</tr>
</tbody>
</table>
Bottom-up parser

• We initialize the table with the parts-of-speech of each word ...

<table>
<thead>
<tr>
<th>symbol</th>
<th>start</th>
<th>end</th>
<th>constituents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ART</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>V</td>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>N</td>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>N</td>
<td>3</td>
<td>4</td>
<td>-</td>
</tr>
</tbody>
</table>

• remembering that many English words have several parts of speech
Bottom-up parser

- Then if there is a production $A \rightarrow B \, C$ and we have entries for $B$ and $C$ with $\text{end}_B = \text{start}_C$, we add an entry for $A$ with $\text{start} = \text{start}_B$ and $\text{end} = \text{end}_C$

<table>
<thead>
<tr>
<th>node #</th>
<th>symbol</th>
<th>start</th>
<th>end</th>
<th>constituents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ART</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>N</td>
<td>1</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>V</td>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>N</td>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>3</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>NP</td>
<td>0</td>
<td>2</td>
<td>[0, 1]</td>
</tr>
</tbody>
</table>

[see lecture notes for handling general productions]
# Bottom-up parser

<table>
<thead>
<tr>
<th>node #</th>
<th>symbol</th>
<th>start</th>
<th>end</th>
<th>constituents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ART</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>N</td>
<td>1</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>V</td>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>N</td>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>3</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>NP</td>
<td>0</td>
<td>2</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>6</td>
<td>NP</td>
<td>1</td>
<td>2</td>
<td>[1]</td>
</tr>
<tr>
<td>7</td>
<td>NP</td>
<td>2</td>
<td>3</td>
<td>[3]</td>
</tr>
<tr>
<td>8</td>
<td>NP</td>
<td>3</td>
<td>4</td>
<td>[4]</td>
</tr>
<tr>
<td>9</td>
<td>S</td>
<td>0</td>
<td>4</td>
<td>[5, 2, 8]</td>
</tr>
<tr>
<td>10</td>
<td>S</td>
<td>1</td>
<td>4</td>
<td>[6, 2, 8]</td>
</tr>
</tbody>
</table>

Several more S’s

Parse!