Basic Parsing with Context-Free Grammars

Some slides adapted from Julia Hirschberg and Dan Jurafsky
Announcements

- To view past videos:
  - [http://globe.cvn.columbia.edu:8080/oncampus.php?c=133ae14752e27fde909fddb64c06b337](http://globe.cvn.columbia.edu:8080/oncampus.php?c=133ae14752e27fde909fddb64c06b337)

- Usually available only for 1 week. Right now, available for all previous lectures
Earley Parsing

- Allows arbitrary CFGs
- Fills a table in a single sweep over the input words
  - Table is length $N+1$; $N$ is number of words
  - Table entries represent
    - Completed constituents and their locations
    - In–progress constituents
    - Predicted constituents
It would be nice to know where these things are in the input so...

- **S → · VP [0,0]**
  - A VP is predicted at the start of the sentence

- **NP → Det · Nominal [1,2]**
  - An NP is in progress; the Det goes from 1 to 2

- **VP → V NP · [0,3]**
  - A VP has been found starting at 0 and ending at 3
Graphically

S → .VP

NP → Det Nominal

VP → V NP .

0 Book 1 that 2 flight 3
March through chart left-to-right.
At each step, apply 1 of 3 operators
  ◦ Predictor
    • Create new states representing top-down expectations
  ◦ Scanner
    • Match word predictions (rule with word after dot) to words
  ◦ Completer
    • When a state is complete, see what rules were looking for that completed constituent

Done when an S spans from 0 to n
Given a state

- With a non-terminal to right of dot (not a part-of-speech category)
- Create a new state for each expansion of the non-terminal
- Place these new states into same chart entry as generated state, beginning and ending where generating state ends.

So predictor looking at

- \( S \rightarrow . \text{VP} [0,0] \)

results in

- \( \text{VP} \rightarrow . \text{Verb} [0,0] \)
- \( \text{VP} \rightarrow . \text{Verb NP} [0,0] \)
Scanner

- Given a state
  - With a non-terminal to right of dot that is a part-of-speech category
  - If the next word in the input matches this POS
  - Create a new state with dot moved over the non-terminal
  - So scanner looking at \( VP \rightarrow .\) Verb NP [0,0]
  - If the next word, “book”, can be a verb, add new state:
    - \( VP \rightarrow \) Verb . NP [0,1]
  - Add this state to chart entry following current one
  - Note: Earley algorithm uses top-down input to disambiguate POS! Only POS predicted by some state can get added to chart!
Completierung

- Applied to a state when its dot has reached right end of role.
- Parser has discovered a category over some span of input.
- Find and advance all previous states that were looking for this category
  - copy state, move dot, insert in current chart entry

Given:
- NP -> Det Nominal . [1,3]
- VP -> Verb. NP [0,1]

Add
- VP -> Verb NP . [0,3]
How do we know we are done?

- Find an S state in the final column that spans from 0 to n and is complete.

- If that’s the case you’re done.
  - $S \rightarrow \alpha \cdot [0,n]$
More specifically…

1. Predict all the states you can upfront

2. Read a word
   1. Extend states based on matches
   2. Add new predictions
   3. Go to 2

3. Look at N to see if you have a winner
Example

- Book that flight
- We should find... an S from 0 to 3 that is a completed state...
## CFG for Fragment of English

<table>
<thead>
<tr>
<th>Production</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>VP → V</td>
</tr>
<tr>
<td>S → Aux NP VP</td>
<td>PP -&gt; Prep NP</td>
</tr>
<tr>
<td>NP → Det Nom</td>
<td>N → old</td>
</tr>
<tr>
<td>NP → PropN</td>
<td>V → dog</td>
</tr>
<tr>
<td>Nom -&gt; Adj Nom</td>
<td>Aux → does</td>
</tr>
<tr>
<td>Nom → N</td>
<td>Prep → from</td>
</tr>
<tr>
<td>Nom → N Nom</td>
<td>PropN → Bush</td>
</tr>
<tr>
<td>Nom → Nom PP</td>
<td>Det → that</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>Adj -&gt; old</td>
</tr>
<tr>
<td>Chart[0]</td>
<td>S0</td>
</tr>
<tr>
<td>---------</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td>S1</td>
</tr>
<tr>
<td></td>
<td>S2</td>
</tr>
<tr>
<td></td>
<td>S3</td>
</tr>
<tr>
<td></td>
<td>S4</td>
</tr>
<tr>
<td></td>
<td>S5</td>
</tr>
<tr>
<td></td>
<td>S6</td>
</tr>
<tr>
<td></td>
<td>S7</td>
</tr>
<tr>
<td></td>
<td>S8</td>
</tr>
<tr>
<td></td>
<td>S9</td>
</tr>
<tr>
<td></td>
<td>S10</td>
</tr>
<tr>
<td></td>
<td>S11</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Chart[1]</th>
<th>Production</th>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>S12</td>
<td>$\text{Verb} \rightarrow \text{book}$</td>
<td>[0,1]</td>
<td>Scanner</td>
</tr>
<tr>
<td>S13</td>
<td>$\text{VP} \rightarrow \text{Verb}$</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S14</td>
<td>$\text{VP} \rightarrow \text{Verb} \cdot \text{NP}$</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S15</td>
<td>$\text{VP} \rightarrow \text{Verb} \cdot \text{NP PP}$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S16</td>
<td>$\text{VP} \rightarrow \text{Verb} \cdot \text{PP}$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S17</td>
<td>$\text{S} \rightarrow \text{VP}$</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S18</td>
<td>$\text{VP} \rightarrow \text{VP} \cdot \text{PP}$</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S19</td>
<td>$\text{NP} \rightarrow \cdot \text{Pronoun}$</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S20</td>
<td>$\text{NP} \rightarrow \cdot \text{Proper-Noun}$</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S21</td>
<td>$\text{NP} \rightarrow \cdot \text{Det Nominal}$</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S22</td>
<td>$\text{PP} \rightarrow \cdot \text{Prep NP}$</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Chart[2]</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S23</td>
<td>$\text{Det} \rightarrow \text{that} \bullet$</td>
<td>[1,2]</td>
</tr>
<tr>
<td>S24</td>
<td>$\text{NP} \rightarrow \text{Det} \bullet \text{Nominal}$</td>
<td>[1,2]</td>
</tr>
<tr>
<td>S25</td>
<td>$\text{Nominal} \rightarrow \bullet \text{Noun}$</td>
<td>[2,2]</td>
</tr>
<tr>
<td>S26</td>
<td>$\text{Nominal} \rightarrow \bullet \text{Nominal Noun}$</td>
<td>[2,2]</td>
</tr>
<tr>
<td>S27</td>
<td>$\text{Nominal} \rightarrow \bullet \text{Nominal PP}$</td>
<td>[2,2]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chart[3]</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S28</td>
<td>$\text{Noun} \rightarrow \text{flight} \bullet$</td>
<td>[2,3]</td>
</tr>
<tr>
<td>S29</td>
<td>$\text{Nominal} \rightarrow \text{Noun} \bullet$</td>
<td>[2,3]</td>
</tr>
<tr>
<td>S30</td>
<td>$\text{NP} \rightarrow \text{Det Nominal} \bullet$</td>
<td>[1,3]</td>
</tr>
<tr>
<td>S31</td>
<td>$\text{Nominal} \rightarrow \text{Nominal} \bullet \text{Noun}$</td>
<td>[2,3]</td>
</tr>
<tr>
<td>S32</td>
<td>$\text{Nominal} \rightarrow \text{Nominal} \bullet \text{PP}$</td>
<td>[2,3]</td>
</tr>
<tr>
<td>S33</td>
<td>$\text{VP} \rightarrow \text{Verb NP} \bullet$</td>
<td>[0,3]</td>
</tr>
<tr>
<td>S34</td>
<td>$\text{VP} \rightarrow \text{Verb NP} \bullet \text{PP}$</td>
<td>[0,3]</td>
</tr>
<tr>
<td>S35</td>
<td>$\text{PP} \rightarrow \bullet \text{Prep NP}$</td>
<td>[3,3]</td>
</tr>
<tr>
<td>S36</td>
<td>$\text{S} \rightarrow \text{VP} \bullet$</td>
<td>[0,3]</td>
</tr>
</tbody>
</table>
What kind of algorithms did we just describe

- Not parsers – recognizers
  - The presence of an S state with the right attributes in the right place indicates a successful recognition.
  - But no parse tree... no parser
  - That’s how we solve (not) an exponential problem in polynomial time
Converting Earley from Recognizer to Parser

- With the addition of a few pointers we have a parser
- Augment the “Completer” to point to where we came from.
Augmenting the chart with structural information

**Chart[1]**

<table>
<thead>
<tr>
<th>S8</th>
<th>Verb</th>
<th>book</th>
<th>[0,1]</th>
<th>Scanner</th>
</tr>
</thead>
<tbody>
<tr>
<td>S9</td>
<td>VP</td>
<td>Verb</td>
<td>[0,1]</td>
<td>Completer S8</td>
</tr>
<tr>
<td>S10</td>
<td>S</td>
<td>VP</td>
<td>[0,1]</td>
<td>Completer S9</td>
</tr>
<tr>
<td>S11</td>
<td>VP</td>
<td>Verb</td>
<td>NP</td>
<td>[0,1]</td>
</tr>
<tr>
<td>S12</td>
<td>NP</td>
<td>Det</td>
<td>NOMINAL</td>
<td>[1,1]</td>
</tr>
<tr>
<td>S13</td>
<td>NP</td>
<td>Proper-Noun</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
</tbody>
</table>

**Chart[2]**

<table>
<thead>
<tr>
<th>Det</th>
<th>that</th>
<th>[1,2]</th>
<th>Scanner</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>Det</td>
<td>NOMINAL</td>
<td>[1,2]</td>
</tr>
<tr>
<td>NOMINAL</td>
<td>Noun</td>
<td>[2,2]</td>
<td>Predictor</td>
</tr>
<tr>
<td>NOMINAL</td>
<td>Noun</td>
<td>NOMINAL</td>
<td>[2,2]</td>
</tr>
</tbody>
</table>
Retrieving Parse Trees from Chart

- All the possible parses for an input are in the table.
- We just need to read off all the backpointers from every complete S in the last column of the table.
- Find all the $S \rightarrow X \cdot [0,N+1]$.
- Follow the structural traces from the Completer.
- Of course, this won’t be polynomial time, since there could be an exponential number of trees.
- We can at least represent ambiguity efficiently.
Depth-first search will never terminate if grammar is *left recursive* (e.g. NP $\rightarrow$ NP PP)

\[(A \rightarrow^* \alpha AB, \alpha \rightarrow^* \varepsilon)\]
Solutions:
- Rewrite the grammar (automatically?) to a weakly equivalent one which is not left-recursive
e.g. The man {on the hill with the telescope...}
NP → NP PP (wanted: Nom plus a sequence of PPs)
NP → Nom PP
NP → Nom
Nom → Det N
...becomes...
NP → Nom NP’
Nom → Det N
NP’ → PP NP’ (wanted: a sequence of PPs)
NP’ → e
- Not so obvious what these rules mean...
• Harder to detect and eliminate *non-immediate left recursion*
  • NP --> Nom PP
  • Nom --> NP

• Fix depth of search explicitly

• Rule ordering: non–recursive rules first
  • NP --> Det Nom
  • NP --> NP PP
Another Problem: Structural ambiguity

- Multiple legal structures
  - Attachment (e.g. I saw a man on a hill with a telescope)
  - Coordination (e.g. younger cats and dogs)
  - NP bracketing (e.g. Spanish language teachers)
NP vs. VP Attachment
Solution?
- Return all possible parses and disambiguate using “other methods”
Summing Up

- Parsing is a search problem which may be implemented with many control strategies
  - Top-Down or Bottom-Up approaches each have problems
    - Combining the two solves some but not all issues
  - Left recursion
  - Syntactic ambiguity

- Rest of today (and next time): Making use of statistical information about syntactic constituents
  - Read Ch 14
Probabilistic Parsing
How to do parse disambiguation

- Probabilistic methods
- Augment the grammar with probabilities
- Then modify the parser to keep only most probable parses
- And at the end, return the most probable parse
Probabilistic CFGs

- The probabilistic model
  - Assigning probabilities to parse trees
- Getting the probabilities for the model
- Parsing with probabilities
  - Slight modification to dynamic programming approach
  - Task is to find the max probability tree for an input
Probability Model

- Attach probabilities to grammar rules
- The expansions for a given non-terminal sum to 1

\[
\begin{align*}
\text{VP} \rightarrow \text{Verb} & \quad .55 \\
\text{VP} \rightarrow \text{Verb NP} & \quad .40 \\
\text{VP} \rightarrow \text{Verb NP NP} & \quad .05
\end{align*}
\]

- Read this as P(Specific rule | LHS)
### PCFG

<table>
<thead>
<tr>
<th>Production</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow NP ; VP )</td>
<td>.80</td>
</tr>
<tr>
<td>( S \rightarrow \text{Aux} ; NP ; VP )</td>
<td>.15</td>
</tr>
<tr>
<td>( S \rightarrow VP )</td>
<td>.05</td>
</tr>
<tr>
<td>( NP \rightarrow \text{Det} ; \text{Nom} )</td>
<td>.20</td>
</tr>
<tr>
<td>( NP \rightarrow \text{Proper-Noun} )</td>
<td>.35</td>
</tr>
<tr>
<td>( NP \rightarrow \text{Nom} )</td>
<td>.05</td>
</tr>
<tr>
<td>( NP \rightarrow \text{Pronoun} )</td>
<td>.40</td>
</tr>
<tr>
<td>( \text{Nom} \rightarrow \text{Noun} )</td>
<td>.75</td>
</tr>
<tr>
<td>( \text{Nom} \rightarrow \text{Noun Nom} )</td>
<td>.20</td>
</tr>
<tr>
<td>( \text{Nom} \rightarrow \text{Proper-Noun Nom} )</td>
<td>.05</td>
</tr>
<tr>
<td>( VP \rightarrow \text{Verb} )</td>
<td>.55</td>
</tr>
<tr>
<td>( VP \rightarrow \text{Verb} ; NP )</td>
<td>.40</td>
</tr>
<tr>
<td>( VP \rightarrow \text{Verb} ; NP ; NP )</td>
<td>.05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Production</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Det} \rightarrow \text{that} )</td>
<td>.05</td>
</tr>
<tr>
<td>( \text{Det} \rightarrow \text{the} )</td>
<td>.80</td>
</tr>
<tr>
<td>( \text{Det} \rightarrow \text{a} )</td>
<td>.15</td>
</tr>
<tr>
<td>( \text{Noun} \rightarrow \text{book} )</td>
<td>.10</td>
</tr>
<tr>
<td>( \text{Noun} \rightarrow \text{flights} )</td>
<td>.50</td>
</tr>
<tr>
<td>( \text{Noun} \rightarrow \text{meal} )</td>
<td>.40</td>
</tr>
<tr>
<td>( \text{Verb} \rightarrow \text{book} )</td>
<td>.30</td>
</tr>
<tr>
<td>( \text{Verb} \rightarrow \text{include} )</td>
<td>.30</td>
</tr>
<tr>
<td>( \text{Verb} \rightarrow \text{want} )</td>
<td>.40</td>
</tr>
<tr>
<td>( \text{Aux} \rightarrow \text{can} )</td>
<td>.40</td>
</tr>
<tr>
<td>( \text{Aux} \rightarrow \text{does} )</td>
<td>.30</td>
</tr>
<tr>
<td>( \text{Aux} \rightarrow \text{do} )</td>
<td>.30</td>
</tr>
<tr>
<td>( \text{Proper-Noun} \rightarrow \text{TWA} )</td>
<td>.40</td>
</tr>
<tr>
<td>( \text{Proper-Noun} \rightarrow \text{Denver} )</td>
<td>.40</td>
</tr>
<tr>
<td>( \text{Pronoun} \rightarrow \text{you} )</td>
<td>.40</td>
</tr>
<tr>
<td>( \text{Pronoun} \rightarrow \text{I} )</td>
<td>.60</td>
</tr>
</tbody>
</table>
### PCFG

#### (a)
- **S** → **Aux** **NP** **VP**
  - **Aux** → Pro
  - **NP** → **PNoun** **Noun**
    - **PNoun** → TWA
    - **Noun** → flights

#### (b)
- **S** → **Aux** **NP** **VP**
  - **Aux** → Pro
  - **NP** → **PNoun** **Noun**
    - **PNoun** → TWA
    - **Noun** → flights

#### Rules and Probabilities

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → Aux NP VP</td>
<td>.15</td>
</tr>
<tr>
<td>NP → Pro</td>
<td>.40</td>
</tr>
<tr>
<td>VP → V NP NP</td>
<td>.05</td>
</tr>
<tr>
<td>NP → Nom</td>
<td>.05</td>
</tr>
<tr>
<td>Nom → PNoun</td>
<td>.35</td>
</tr>
<tr>
<td>Nom → Noun</td>
<td>.75</td>
</tr>
<tr>
<td>Aux → Can</td>
<td>.40</td>
</tr>
<tr>
<td>NP → Pro</td>
<td>.40</td>
</tr>
<tr>
<td>Pro → you</td>
<td>.40</td>
</tr>
<tr>
<td>Verb → book</td>
<td>.30</td>
</tr>
<tr>
<td>PNoun → TWA</td>
<td>.40</td>
</tr>
<tr>
<td>Noun → flights</td>
<td>.50</td>
</tr>
<tr>
<td>S → Aux NP VP</td>
<td>.15</td>
</tr>
<tr>
<td>NP → Pro</td>
<td>.40</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>.40</td>
</tr>
<tr>
<td>NP → Nom</td>
<td>.05</td>
</tr>
<tr>
<td>Nom → PNoun Nom</td>
<td>.05</td>
</tr>
<tr>
<td>Nom → Noun</td>
<td>.75</td>
</tr>
<tr>
<td>Aux → Can</td>
<td>.40</td>
</tr>
<tr>
<td>NP → Pro</td>
<td>.40</td>
</tr>
<tr>
<td>Pro → you</td>
<td>.40</td>
</tr>
<tr>
<td>Verb → book</td>
<td>.30</td>
</tr>
<tr>
<td>PNoun → TWA</td>
<td>.40</td>
</tr>
<tr>
<td>Noun → flights</td>
<td>.50</td>
</tr>
</tbody>
</table>
Probability Model (1)

- A derivation (tree) consists of the set of grammar rules that are in the tree.
- The probability of a tree is just the product of the probabilities of the rules in the derivation.
**Probability model**

\[ P(T, S) = \prod_{n \in T} p(r_n) \]

\[ P(T, S) = P(T)P(S|T) = P(T); \text{ since } P(S|T)=1 \]

\[ P(T_l) = .15 \times .40 \times .05 \times .05 \times .35 \times .75 \times .40 \times .40 \times .40 \times .30 \times .40 \times .50 \]
\[ = 1.5 \times 10^{-6} \]

\[ P(T_r) = .15 \times .40 \times .40 \times .05 \times .05 \times .75 \times .40 \times .40 \times .40 \times .30 \times .40 \times .50 \]
\[ = 1.7 \times 10^{-6} \]
The probability of a word sequence $P(S)$ is the probability of its tree in the unambiguous case.

It’s the sum of the probabilities of the trees in the ambiguous case.
From an annotated database (a treebank)
  ◦ So for example, to get the probability for a particular VP rule just count all the times the rule is used and divide by the number of VPs overall.
((S
  (NP-SBJ (DT That))
  (JJ cold) (, ,)
  (JJ empty) (NN sky))
(VP (VBD was))
  (ADJP-PRD (JJ full))
    (PP (IN of))
      (NP (NN fire))
        (CC and)
        (NN light))))
  (., .))
  (a)
)

((S
  (NP-SBJ The/DT flight/NN)
  (VP should/MD)
    (VP arrive/VB)
      (PP-TMP at/IN)
        (NP eleven/CD a.m/RB))
        (NP-TMP tomorrow/NN))))
  (b)
That cold, empty sky was full of fire and light.
( (S (" " " ))
   (S-TPC-2
     (NP-SBJ-1 (PRP We) )
     (VP (MD would)
      (VP (VB have)
       (S
        (NP-SBJ (-NONE- *-1) )
        (VP (TO to)
        (VP (VB wait)
         (SBAR-TMP (IN until)
          (S
           (NP-SBJ (PRP we) )
           (VP (VBP have)
            (VP (VBN collected)
             (PP-CLR (IN on)
              (NP (DT those) (NNS assets) ))))))))))))
   (S (-NONE- *T*-2 ))
   ( . . ) )))

Treebanks
Treebank Grammars

<table>
<thead>
<tr>
<th>$S$</th>
<th>$\rightarrow$ $NP$ $VP$ .</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NP$</td>
<td>$\rightarrow$ $NP$ $VP$</td>
</tr>
<tr>
<td></td>
<td>$&quot;S&quot;$, $NP$ $VP$ .</td>
</tr>
<tr>
<td></td>
<td>$\text{-NONE-}$</td>
</tr>
<tr>
<td></td>
<td>$DT$ $NN$</td>
</tr>
<tr>
<td></td>
<td>$DT$ $NN$ $NNS$</td>
</tr>
<tr>
<td></td>
<td>$NN$ $CC$ $NN$</td>
</tr>
<tr>
<td></td>
<td>$CD$ $RB$</td>
</tr>
<tr>
<td>$NP$</td>
<td>$\rightarrow$ $DT$ $JJ$, $JJ$ $NN$</td>
</tr>
<tr>
<td></td>
<td>$PRP$</td>
</tr>
<tr>
<td></td>
<td>$\text{-NONE-}$</td>
</tr>
<tr>
<td>$VP$</td>
<td>$\rightarrow$ $MD$ $VP$</td>
</tr>
<tr>
<td></td>
<td>$VBD$ $ADJP$</td>
</tr>
<tr>
<td></td>
<td>$VBD$ $S$</td>
</tr>
<tr>
<td></td>
<td>$VB$ $PP$</td>
</tr>
<tr>
<td></td>
<td>$VB$ $S$</td>
</tr>
<tr>
<td></td>
<td>$VB$ $SBAR$</td>
</tr>
<tr>
<td></td>
<td>$VBP$ $VP$</td>
</tr>
<tr>
<td></td>
<td>$VBN$ $VP$</td>
</tr>
<tr>
<td></td>
<td>$TO$ $VP$</td>
</tr>
<tr>
<td>$SBAR$</td>
<td>$\rightarrow$ $IN$ $S$</td>
</tr>
<tr>
<td>$ADJP$</td>
<td>$\rightarrow$ $JJ$ $PP$</td>
</tr>
<tr>
<td>$PP$</td>
<td>$\rightarrow$ $IN$ $NP$</td>
</tr>
<tr>
<td>$PRP$</td>
<td>$\rightarrow$ $we$, $he$</td>
</tr>
<tr>
<td>$DT$</td>
<td>$\rightarrow$ $the$, $that$, $those$</td>
</tr>
<tr>
<td>$JJ$</td>
<td>$\rightarrow$ $cold$, $empty$, $full$</td>
</tr>
<tr>
<td>$NN$</td>
<td>$\rightarrow$ $sky$, $fire$, $light$, $flight$</td>
</tr>
<tr>
<td>$NNS$</td>
<td>$\rightarrow$ $assets$</td>
</tr>
<tr>
<td>$CC$</td>
<td>$\rightarrow$ $and$</td>
</tr>
<tr>
<td>$IN$</td>
<td>$\rightarrow$ $of$, $at$, $until$, $on$</td>
</tr>
<tr>
<td>$CD$</td>
<td>$\rightarrow$ $eleven$</td>
</tr>
<tr>
<td>$RB$</td>
<td>$\rightarrow$ $a.m.$</td>
</tr>
<tr>
<td>$VB$</td>
<td>$\rightarrow$ $arrive$, $have$, $wait$</td>
</tr>
<tr>
<td>$VBD$</td>
<td>$\rightarrow$ $said$</td>
</tr>
<tr>
<td>$VBP$</td>
<td>$\rightarrow$ $have$</td>
</tr>
<tr>
<td>$VBN$</td>
<td>$\rightarrow$ $collected$</td>
</tr>
<tr>
<td>$MD$</td>
<td>$\rightarrow$ $should$, $would$</td>
</tr>
<tr>
<td>$TO$</td>
<td>$\rightarrow$ $to$</td>
</tr>
</tbody>
</table>
Lots of flat rules

\[
\begin{align*}
\text{NP} & \rightarrow \text{DT JJ NN} \\
\text{NP} & \rightarrow \text{DT JJ NNS} \\
\text{NP} & \rightarrow \text{DT JJ NN NN} \\
\text{NP} & \rightarrow \text{DT JJ NN} \\
\text{NP} & \rightarrow \text{DT JJ CD NNS} \\
\text{NP} & \rightarrow \text{RB DT JJ NN NN} \\
\text{NP} & \rightarrow \text{RB DT JJ JJ NNS} \\
\text{NP} & \rightarrow \text{DT JJ JJ NNP NNS} \\
\text{NP} & \rightarrow \text{DT NNP NNP NNP NNP NNP JJ NN} \\
\text{NP} & \rightarrow \text{DT JJ NNP CC JJ JJ NN NNS} \\
\text{NP} & \rightarrow \text{RB DT JJS NN NN SBAR} \\
\text{NP} & \rightarrow \text{DT VBG JJ NNP NNP CC NNP} \\
\text{NP} & \rightarrow \text{DT JJ NNS , NNS CC NN NNS NN} \\
\text{NP} & \rightarrow \text{DT JJ JJ VBG NN NNP NNP FW NNP} \\
\text{NP} & \rightarrow \text{NP JJ , JJ ‘‘ SBAR ‘‘ NNS}
\end{align*}
\]
Example sentences from those rules

- Total: over 17,000 different grammar rules in the 1–million word Treebank corpus

(9.19) [DT The] [JJ state-owned] [JJ industrial] [VBG holding] [NN company] [NNP Instituto] [NNP Nacional] [FW de] [NNP Industria]

(9.20) [NP Shearson’s] [JJ easy-to-film], [JJ black-and-white] “[SBAR Where We Stand]” [NNS commercials]
We’re assuming that there is a **grammar** to be used to parse with.

We’re assuming the existence of a large robust **dictionary** with parts of speech.

We’re assuming the ability to parse (i.e. a **parser**).

Given all that... we can parse probabilistically.
Typical Approach

- Bottom-up (CKY) dynamic programming approach
- Assign probabilities to constituents as they are completed and placed in the table
- Use the max probability for each constituent going up
Say we’re talking about a final part of a parse
$S \rightarrow_0 NP_i VP_j$

The probability of the S is...
$P(S \rightarrow NP \ VP) \cdot P(NP) \cdot P(VP)$

The green stuff is already known. We’re doing bottom-up parsing
Max

- I said the P(NP) is known.
- What if there are multiple NPs for the span of text in question (0 to i)?
- Take the max (where?)
Problems with PCFGs

- The probability model we’re using is just based on the rules in the derivation…
  - Doesn’t use the words in any real way
  - Doesn’t take into account where in the derivation a rule is used
Solution

- Add lexical dependencies to the scheme...
  - Infiltrate the predilections of particular words into the probabilities in the derivation
  - I.e. Condition the rule probabilities on the actual words
Heads

- To do that we’re going to make use of the notion of the head of a phrase
  - The head of an NP is its noun
  - The head of a VP is its verb
  - The head of a PP is its preposition
  (It’s really more complicated than that but this will do.)
Example (right)

Attribute grammar

S(dumped)

NP(workers)
  NNS(workers)
    workers
  VBD(dumped)
    dumped

NP(sacks)
  NNS(sacks)
    sacks
  P(into)
    into

PP(into)
  NP(bin)
    DT(a)
      a
    NN(bin)
      bin
Example (wrong)
How?

- We used to have
  - $\text{VP} \rightarrow \text{V NP PP} \quad P(\text{rule} | \text{VP})$
    - That’s the count of this rule divided by the number of VPs in a treebank

- Now we have
  - $\text{VP(dumped)} \rightarrow \text{V(dumped) NP(sacks)PP(in)}$
    - $P(r | \text{VP ^ dumped is the verb ^ sacks is the head of the NP ^ in is the head of the PP})$
    - Not likely to have significant counts in any treebank
When stuck, exploit independence and collect the statistics you can…

We’ll focus on capturing two things

- Verb subcategorization
  - Particular verbs have affinities for particular VPs
- Objects affinities for their predicates (mostly their mothers and grandmothers)
  - Some objects fit better with some predicates than others
Subcategorization

- Condition particular VP rules on their head...
  so
  \[ r: \text{VP} \rightarrow V \text{NP PP} \quad P(r|\text{VP}) \]
  Becomes
  \[ P(r \mid \text{VP} \wedge \text{dumped}) \]

What’s the count?
How many times was this rule used with (head) dump, divided by the number of VPs that dump appears (as head) in total

Think of left and right modifiers to the head
Example (right)

Attribute grammar

S(dumped)
  └── NP(workers)
      ├── NNS(workers)
      └── VBD(dumped)
  └── VP(dumped)
      ├── NP(sacks)
      │   ├── NNS(sacks)
      │   └── P(into)
      │       ├── DT(a)
      │           └── NN(bin)
      └── PP(into)
            └── NP(bin)
                  └── a
            └── bin
Probability model

\[ P(T, S) = \prod_{n \in T} p(r_n) \]

- \( P(T, S) = S \rightarrow NP \ VP \ (.5)^* \)
- \( VP(\text{dumped}) \rightarrow V \ NP \ PP \ (.5) \ (T1) \)
- \( VP(\text{ate}) \rightarrow V \ NP \ PP \ (.03) \)
- \( VP(\text{dumped}) \rightarrow V \ NP \ (.2) \ (T2) \)
Preferences

- Subcategorization captures the affinity between VP heads (verbs) and the VP rules they go with.
- What about the affinity between VP heads and the heads of the other daughters of the VP?
- Back to our examples...
Example (right)

```
S(dumped)
  NP(workers)  VP(dumped)
    NNS(workers)  VBD(dumped)
         NP(sacks)  PP(into)
           NNS(sacks)  P(into)
                 DT(a)  NN(bin)
                   bin
```
Example (wrong)
The issue here is the attachment of the PP. So the affinities we care about are the ones between dumped and into vs. sacks and into.

So count the places where dumped is the head of a constituent that has a PP daughter with into as its head and normalize.

Vs. the situation where sacks is a constituent with into as the head of a PP daughter.
Probability model

\[ P(T,S) = \prod_{n \in T} p(r_n) \]

- \( P(T,S) = S \rightarrow \text{NP VP (.5)*} \)
- \( \text{VP(dumped)} \rightarrow \text{V NP PP(into) (.7) (T1)} \)
- \( \text{NOM(sacks)} \rightarrow \text{NOM PP(into) (.01) (T2)} \)
Consider the VPs
  ◦ Ate spaghetti with gusto
  ◦ Ate spaghetti with marinara

The affinity of **gusto** for **eat** is much larger than its affinity for **spaghetti**

On the other hand, the affinity of **marinara** for **spaghetti** is much higher than its affinity for **ate**
Note the relationship here is more distant and doesn’t involve a headword since gusto and marinara aren’t the heads of the PPs.

- Vp (ate) → Vp(ate) → Pp(with) → Ate spaghetti with gusto
- Vp(ate) → Np(spag) → np → Pp(with) → Ate spaghetti with marinara
Summary

- Context-Free Grammars
- Parsing
  - Top Down, Bottom Up Metaphors
  - Dynamic Programming Parsers: CKY, Earley
- Disambiguation:
  - PCFG
  - Probabilistic Augmentations to Parsers
  - Tradeoffs: accuracy vs. data sparcity
  - Treebanks