Lecture 7
CFGs: Lord of the Loops; Take a chance

Professor Robert C. Berwick
berwick@csail.mit.edu
<table>
<thead>
<tr>
<th>S</th>
<th>→</th>
<th>NP</th>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>→</td>
<td>NP</td>
<td>VB</td>
</tr>
<tr>
<td>VP</td>
<td>→</td>
<td>V&lt;sub&gt;i&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td>→</td>
<td>V&lt;sub&gt;t&lt;/sub&gt;</td>
<td>NP</td>
</tr>
<tr>
<td>VP</td>
<td>→</td>
<td>VP</td>
<td>PP</td>
</tr>
<tr>
<td>NP</td>
<td>→</td>
<td>DT</td>
<td>NN</td>
</tr>
<tr>
<td>NP</td>
<td>→</td>
<td>NP</td>
<td>PP</td>
</tr>
<tr>
<td>PP</td>
<td>→</td>
<td>P</td>
<td>NP</td>
</tr>
</tbody>
</table>

| V<sub>i</sub> | →  | sleeps |
| V<sub>t</sub> | →  | saw    |
| NN            | →  | guy    |
| NN            | →  | dog    |
| NN            | →  | person |
| NN            | →  | telescope |
| DT            | →  | the    |
| IN            | →  | with   |
| IN            | →  | in     |
Given a CFG $G = N, V, S, R$ and input sentence $s$.

function CKY-Parse($words, G$) returns table

1 for $j \leftarrow 1$ to length($words$)
2 do $table[j - 1, j] \leftarrow \{ A | A \rightarrow w_{j-1}, j \in R \}$
3 for $i \leftarrow j - 2$ downto 0
4 do for $k \leftarrow i + 1$ to $j - 1$
5 $table[i, j] \leftarrow table[i, j] \cup \{ A | A \rightarrow B \ C \in R \land$
6 $B \in table[i, k] \land$
7 $C \in table[k, j] \}$

Figure 2: The CKY algorithm works with the upper-triangular portion of a matrix table. Here we have indicated the indices associated with each cell in the table as used by the algorithm, and arrows to indicate the general flow of operations. There is one column in the table for each word in the sentence. The upper right corner of the table holds an entry that is meant to derive the entire sentence.

<table>
<thead>
<tr>
<th></th>
<th>row 0</th>
<th>row 1</th>
<th>row 2</th>
<th>row 3</th>
<th>row 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0,1</td>
<td>0,2</td>
<td>0,3</td>
<td>0,4</td>
<td>0,5</td>
</tr>
<tr>
<td>row 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table columns $j = 1 \ldots n$ and rows $i = j - 2 \ldots 0$.
Filling in a column and building a new tree out of two subtrees: With \( j = 2 \) then, CKY scans up column 2 to work on each of the cells in the column in turn, starting with the word *dog*. It first fills in the cell for \( \text{table}[j,1] \) as before, so \( \text{table}[1,2] \) is filled with **DT** because there is a rule that expands **DT** as the **DT**. There is nothing more to do, so CKY will move on to the next column, with \( j = 2 \), for word *dog*.

The algorithm then scans up the column by decrementing the counter \( i \). Note that since the parts of speech are always filled at row \( j = 1 \) (here, \( j = 2 \) so \( j_1 = 1 \)), the beginning of this scan upwards always starts with the row above this, so initially, \( i = j = 2 \) for this loop running up the columns, which will then decrement down to 0 (the topmost row in a column). In our case, since \( j_2 = 0 \), CKY moves up to row 0. At this cell, \( \text{table}[0,2] \), it then starts in the last, innermost third loop, which attempts to find all possible split points \( k \) such that there are 2 subtrees, one starting at \( i \) and running through \( k \), and the second, starting at \( k \), and running through to \( j \), that can be combined to span the words \( w_{i,k} \) and \( w_{k,j} \) (as per our earlier fact and picture about how trees are 'glued together' in normal form CFGs). Thus, the inner most loop must range over \( k \) running from \( i + 1 \) (the first possible split point after the 'edge' of any possible subtree on the left, all the way to \( j_1 \) (the last possible split point before any subtree on the right). For each possible split point \( k \), CKY tries to find a nonterminal \( A \) such that \( A \rightarrow BC \), and the subtrees \( B \) and \( C \) can be found in the cells in the table already filled in, that span the proper range. In our case, since \( j_2 = 0 \), \( i = 0 \), and we have only a single possible value for \( k \) between \( i \) and \( j \) exclusively, that is, \( k = 1 \). So, CKY looks at the entries in \( \text{table}[0,1] \) (one cell to the left of the current cell being analyzed) and \( \text{table}[1,2] \) (one cell down from the current cell). We see that **DT** is found in \( \text{table}[1,1] \) and **NN** in \( \text{table}[1,2] \), and there is in fact a rule **NP** \( \rightarrow \text{DT NN} \) in the grammar, so CKY adds the symbol **NP** to the cell \( \text{table}[0,2] \). This is what the code in lines 5-7 do. Figure 4 displays this graphically. At this point, CKY is done with column 2, because there are no more rows above this one, with \( i = 0 \).
Note that in line 5 we take the union of any previous values in table \([i, j]\) with what was there before. That is, CKY can potentially accumulate a set of nonterminals entered into a single cell, just as with ambiguous parts of speech. In this case, multiple nonterminals in a single cell always imply some parsing ambiguity.

\[
\begin{array}{c|c|c|c|c|}
   & 0,1 & 0,2 & 0,3 & 0,4 & 0,5 \\
\hline
row 0 & DT & NP &   &   &   \\
\hline
row 1 & the &   & NN &   &   \\
\hline
row 2 &   & dog & 2,3 & 2,4 & 2,5 \\
\hline
row 3 &   &   &   & 3,4 & 3,5 \\
\hline
row 4 &   &   &   &   & 4,5 \\
\end{array}
\]

... dog
The diagram illustrates the process of parsing a sentence using the Stanford parser. The sentence is "the dog saw no rules." The columns are labeled with indices, and the rows are labeled with tokens. The diagram shows the dependency relationships between the tokens, with arrows indicating the grammatical relationships such as subject (NP), object (NP), and verb (V).

- **Row 0**: DT (Determiner) is associated with column 0, and NP (Noun Phrase) is associated with column 3.
- **Row 1**: "the" (DT) is associated with column 1, and "no rules" (NP) is associated with column 3.
- **Row 2**: "dog" (NN) is associated with column 2, and $V_t$ (Verb) is associated with column 3.
- **Row 3**: "saw" (V) is associated with column 3.

The diagonal arrows indicate the head of each phrase, and the dotted lines show the dependency relationships. The sentence "the dog saw no rules." is parsed with these relationships.
<table>
<thead>
<tr>
<th>row 0</th>
<th>0,1</th>
<th>DT</th>
<th>0,2</th>
<th>NP</th>
<th>0,3</th>
<th>0,4</th>
<th>0,5</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 1</td>
<td>0,2</td>
<td>the</td>
<td>1,2</td>
<td>NN</td>
<td>1,3</td>
<td>1,4</td>
<td>1,5</td>
</tr>
<tr>
<td>row 2</td>
<td>1,3</td>
<td>dog</td>
<td>2,3</td>
<td>Vt</td>
<td>2,4</td>
<td>∅</td>
<td>2,5</td>
</tr>
<tr>
<td>row 3</td>
<td>2,4</td>
<td>no</td>
<td>3,4</td>
<td>rules</td>
<td>saw</td>
<td>DT</td>
<td>3,5</td>
</tr>
<tr>
<td>row 4</td>
<td>3,5</td>
<td>the</td>
<td>4,5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The rules and dependencies are illustrated in the table above. The column $j = 4$ indicates the position in the sentence where the action occurs. The table shows how each word is linked to the previous words, demonstrating the syntactic structure of the sentence.
The diagram shows the input sentence "the dog saw the man" in a tabular format with columns labeled as column $j = 5$. Rows are numbered from 0 to 4, and columns are labeled with DT (determiner), NP (noun phrase), and V (verb). The sentence structure is represented by arrows indicating the dependency relations between words. 

- **Row 0**: 
  - Column 0,1: DT
  - Column 0,2: NP
  - Column 0,3:  
  - Column 0,4:  
  - Column 0,5:  

- **Row 1**: 
  - Column 1,2: the
  - Column 1,3: NN
  - Column 1,4:  
  - Column 1,5:  

- **Row 2**: 
  - Column 2,3: dog
  - Column 2,4: V$_t$
  - Column 2,5: VP

- **Row 3**: 
  - Column 3,4: saw
  - Column 3,5: DT NP

- **Row 4**: 
  - Column 4,5: the
  - Column 4,6: NN

The bottom line of the diagram indicates the word "man".
The diagram illustrates a sentence with a dependency tree. The sentence is:

"the dog saw the man"
Time complexity: 3 loops to rule them all

\begin{align*}
\text{function} & \quad \text{CKY-PARSE}(\text{words, grammar}) \quad \text{returns} \quad \text{table} \\
\text{for } j & \leftarrow \text{from } 1 \text{ to } \text{LENGTH}(\text{words}) \text{ do} \\
\quad \text{table}[j - 1, j] & \leftarrow \{ A \mid A \rightarrow \text{words}[j] \in \text{grammar} \} \\
\text{for } i & \leftarrow \text{from } j - 2 \text{ downto } 0 \text{ do} \\
\quad \text{for } k & \leftarrow i + 1 \text{ to } j - 1 \text{ do} \\
\quad \quad \text{table}[i,j] & \leftarrow \text{table}[i,j] \cup \\
\quad & \{ A \mid A \rightarrow BC \in \text{grammar}, \\
\quad & \quad B \in \text{table}[i,k], \\
\quad & \quad C \in \text{table}[k,j] \} \\
\end{align*}

Loop 1 \textit{across} cols, \( j \)

Loop 2 \textit{up} rows, \( i \)

Loop 3 over split points \( k \)
\[ NP \rightarrow NP \; NP \]
\[ NP \rightarrow \text{natural} \mid \text{language} \mid \text{processing} \mid \text{book} \]
Probabilistic Context Free Grammars (PCFGs)

In part to deal with this problem of ambiguity and too many parses, one can turn to making the rules in a CFG probabilistic. In this section, we will defined PCFGs, and see how to estimate their probabilities from training data. We will then see how to alter CKY so it can use such rules. Finally, we shall see why PCFGs themselves are not enough to deal with all the problems of ambiguity in natural language.

2.1 Definitions

The key idea in probabilistic context-free grammars is to extend our definition to give a probability distribution over possible derivations. That is, we will find a way to define a distribution over parse trees, such that for any \( t \in T_G \),

\[
p(t) > 0
\]

and as usual to get a valid probability distribution,

\[
\sum_{t \in T_G} p(t) = 1
\]

This seems tricky because each parse tree \( t \) is a complex structure, and the set \( T_G \) will most likely be infinite. However, we will see that there is a very simple extension to context-free grammars that allows us to define a function \( p(t) \). Once we have this distribution, then we have a ranking over possible parses for any sentence in order of probability. In particular, given a sentence \( s \), we can return as the output from our parser the most likely parse tree for \( s \) given the grammar and the probability model:

\[
\text{argmax}_{t \in T_G} (s) p(t)
\]

(This doesn’t include the fifth parse – which one is missing?)
Probabilistic Context-free Grammars (PCFGs)

- Why: capture notion that while all derivations are equal, some derivations are more equal than others…
- Idea: add *probabilities* to each grammar rule
- *Multiply* these probabilities as we go through a derivation to find the probability of a generated sentence, *wrt* that derivation
How to acquire all these rules? How to sift through the possibilities?

• Read linguistic theory, supervised *learning*
• Read the newspaper, supervised learning

• Sifting:
  • Use dynamic programming so that you never have to explicitly produce all parse trees
  • Use probabilistic rules, so you can prune away unlikely candidates
Example grammar with probabilities

<table>
<thead>
<tr>
<th>S</th>
<th>NP</th>
<th>VP</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP</td>
<td>( V_i )</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td>( V_t ) NP</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td>VP PP</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>DT NN</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>NP PP</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>PP</td>
<td>IN NP</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( V_i )</th>
<th>( V_t )</th>
<th>sleeps</th>
<th>saw</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN</td>
<td>( V_t )</td>
<td>guy</td>
<td>0.3</td>
</tr>
<tr>
<td>NN</td>
<td>person</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>telescope</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>DT</td>
<td>the</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>with</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>in</td>
<td>0.5</td>
<td></td>
</tr>
</tbody>
</table>
Applying rules in a derivation

- Probability of a tree $t$ with rules

$$\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_n \beta_n$$

is

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$$

where $q(\alpha \rightarrow \beta)$ is the probability for rule $\alpha \rightarrow \beta$.

What statistical assumption does this make about rules?
Example derivation

\[ \text{the guy sleeps} \]

\[
\begin{array}{c|c|c}
\text{S} & \rightarrow & \text{NP VP} \\
\hline
\text{VP} & \rightarrow & V_i \\
\hline
\text{VP} & \rightarrow & V_t \text{ NP} \\
\hline
\text{VP} & \rightarrow & \text{VP PP} \\
\hline
\text{NP} & \rightarrow & \text{DT NN} \\
\hline
\text{NP} & \rightarrow & \text{NP PP} \\
\hline
\text{PP} & \rightarrow & \text{IN NP} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
V_i & \rightarrow & \text{sleeps} \\
V_t & \rightarrow & \text{saw} \\
\hline
\text{NN} & \rightarrow & \text{guy} \\
\hline
\text{NN} & \rightarrow & \text{person} \\
\hline
\text{NN} & \rightarrow & \text{telescope} \\
\hline
\text{DT} & \rightarrow & \text{the} \\
\hline
\text{IN} & \rightarrow & \text{with} \\
\hline
\text{IN} & \rightarrow & \text{in} \\
\end{array}
\]

So total probability = \(1.0 \times 0.3 \times 1 \times 0.3 \times 0.4 \times 1 = 0.036\ldots\)
Properties of PCFGs

• Assigns a probability to each left-most derivation, or parse tree, allowed by the underlying CFG

• Say we have a sentence $s$, set of derivations for that sentence is $T(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $T(s)$. i.e., we now have a ranking in order of probability.

• (Q: Any problems with this that you can think of in terms of probability theory?)

• The most likely parse tree for a sentence $s$ is:

$$\arg \max_{t \in T} p(t)$$
How do we derive a PCFG from a corpus of parsed sentences?

- Given a set of example parse trees, take the underlying CFG as all the rules seen in the corpus.
- You should be able to figure it out: what is the MLE estimate for a rule probability, e.g.,
  
  \[ q_{ML}(S \rightarrow \text{NP VP}) \]

- Maximum likelihood estimates:
  
  \[ q_{ML} = \frac{\text{count}(S \rightarrow \text{NP VP})}{\text{count}(S)} \]

- If the training data is generated by a PCFG, then as the training data size goes to infinity, the maximum-likelihood PCFG will converge to the same distribution as the “true” PCFG.
Given a CFG $G = N, V, S, R$ and input sentence $s$.

function CKY-Parse(words, G) returns table
1   for $j \leftarrow 1$ to length(words)
2       do $table[j - 1, j] \leftarrow \{ A | A \rightarrow w_{j-1, j} \in R \}$
3       for $i \leftarrow j - 2$ downto 0
4           do for $k \leftarrow i + 1$ to $j - 1$
5               $table[i, j] \leftarrow table[i, j] \cup \{ A | A \rightarrow B \ C \in R \land B \in table[i, k] \land C \in table[k, j] \}$
function CKY-Parse(words, G) returns most probable parse and its probability

    for j ← 1 to length(words)
        do table[j − 1, j, A] ← \{A | A → w_{j−1,j} \in R\}
        for i ← j − 2 downto 0
            do
                for k ← i + 1 to j − 1
                    do
                        forall \{A | A → BC \in R, \}
                            and table[i, k, B] > 0 and table[k, j, C] > 0
                                if (table[i, j, A] < P(A → BC) \times table[i, k, B] \times table[k, j, C])
                                    then
                                        table[i, j, A] ← P(A → BC) \times table[i, k, B] \times table[k, j, C]
                                        back[i, j, A] ← \{k, B, C\}
                                    end
                                end
            end
        end
    end

return Build-Tree(back[1, length(words), S]), table[1, length(words), S]
Given a CFG $G = N, V, S, R$ and input sentence $s$.

**function** CKY-Parse($words, G$) **returns** table

1. for $j \leftarrow 1$ to length($words$) do
   2. $table[j - 1, j] \leftarrow \{ A | A \rightarrow w_{j-1,j} \in R \}$
   3. for $i \leftarrow j - 2$ downto 0 do
     4. for $k \leftarrow i + 1$ to $j - 1$ do
       5. $table[i, j] \leftarrow table[i, j] \cup \{ A | A \rightarrow BC \in R, B \in table[i, k], C \in table[k, j] \}$

**function** CKY-Parse($words, G$) **returns** most probable parse and its probability

1. for $j \leftarrow 1$ to length($words$) do
   2. $table[j - 1, j, A] \leftarrow \{ A | A \rightarrow w_{j-1,j} \in R \}$
   3. for $i \leftarrow j - 2$ downto 0 do
     4. for $k \leftarrow i + 1$ to $j - 1$ do
       5. forall \{ $A | A \rightarrow BC \in R, and table[i, k, B] > 0$ and table[k, j, C] > 0 \} if ($table[i, j, A] < P(A \rightarrow BC) \times table[i, k, B] \times table[k, j, C]$)
         6. then
             7. $table[i, j, A] \leftarrow P(A \rightarrow BC) \times table[i, k, B] \times table[k, j, C]$
             8. $back[i, j, A] \leftarrow \{ k, B, C \}$
         9. return Build-Tree($back[1, \text{length}(words), S]), table[1, \text{length}(words), S]$
Can PCFGs help with the ambiguity problem?

- Eg: the guy saw the person with the telescope

2 rules compete:

NP → NP PP and VP → VP PP

Rank one higher, so we get ‘guy on hill’) – how?

But will this always work?
The guy saw the person on the hill with the telescope.

(a) Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → DT NN</td>
<td>0.3</td>
</tr>
<tr>
<td>DT → the</td>
<td>1.0</td>
</tr>
<tr>
<td>NN → guy</td>
<td>0.3</td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>VP → Vt NP</td>
<td>0.4</td>
</tr>
<tr>
<td>Vt → saw</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → DT NN</td>
<td>0.3</td>
</tr>
<tr>
<td>DT → the</td>
<td>1.0</td>
</tr>
<tr>
<td>NN → person 0.6</td>
<td>1.0</td>
</tr>
<tr>
<td>PP → IN NP</td>
<td>1.0</td>
</tr>
<tr>
<td>IN → on</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → DT NN</td>
<td>0.3</td>
</tr>
<tr>
<td>DT → the</td>
<td>1.0</td>
</tr>
<tr>
<td>NN → telescope 0.1</td>
<td>1.0</td>
</tr>
</tbody>
</table>

(a) Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → DT NN</td>
<td>0.3</td>
</tr>
<tr>
<td>DT → the</td>
<td>1.0</td>
</tr>
<tr>
<td>NN → guy</td>
<td>0.3</td>
</tr>
<tr>
<td>VP → Vt NP</td>
<td>0.4</td>
</tr>
<tr>
<td>Vt → saw</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → DT NN</td>
<td>0.3</td>
</tr>
<tr>
<td>DT → the</td>
<td>1.0</td>
</tr>
<tr>
<td>NN → person 0.6</td>
<td>1.0</td>
</tr>
<tr>
<td>PP → IN NP</td>
<td>1.0</td>
</tr>
<tr>
<td>IN → on</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → DT NN</td>
<td>0.3</td>
</tr>
<tr>
<td>DT → the</td>
<td>1.0</td>
</tr>
<tr>
<td>NN → telescope 0.1</td>
<td>1.0</td>
</tr>
</tbody>
</table>

the guy saw the person on the hill with the telescope
workers dumped sacks into a bin

dumped workers sacks into a bin
(a) Rules

\[
\begin{align*}
S &\rightarrow \text{NP } \text{VP} \\
\text{NP} &\rightarrow \text{NNS} \\
\text{VP} &\rightarrow \text{VP } \text{PP} \\
\text{VP} &\rightarrow \text{VBD } \text{NP} \\
\text{NP} &\rightarrow \text{NNS} \\
\text{PP} &\rightarrow \text{IN } \text{NP} \\
\text{NP} &\rightarrow \text{DT } \text{NN} \\
\text{NNS} &\rightarrow \text{workers} \\
\text{VBD} &\rightarrow \text{dumped} \\
\text{NNS} &\rightarrow \text{sacks} \\
\text{INS} &\rightarrow \text{into} \\
\text{DT} &\rightarrow \text{a} \\
\text{NN} &\rightarrow \text{bin}
\end{align*}
\]

(b) Rules

\[
\begin{align*}
S &\rightarrow \text{NP } \text{VP} \\
\text{NP} &\rightarrow \text{NNS} \\
\text{VP} &\rightarrow \text{VP } \text{PP} \\
\text{VP} &\rightarrow \text{VBD } \text{NP} \\
\text{NP} &\rightarrow \text{NNS} \\
\text{PP} &\rightarrow \text{IN } \text{NP} \\
\text{NP} &\rightarrow \text{DT } \text{NN} \\
\text{NNS} &\rightarrow \text{workers} \\
\text{VBD} &\rightarrow \text{dumped} \\
\text{NNS} &\rightarrow \text{sacks} \\
\text{INS} &\rightarrow \text{into} \\
\text{DT} &\rightarrow \text{a} \\
\text{NN} &\rightarrow \text{bin}
\end{align*}
\]

if \(q(\text{NP} \rightarrow \text{NP } \text{PP}) > q(\text{NP} \rightarrow \text{VP } \text{PP})\) then (b) is more probable, else (a) is more probable.
However, in the PTB, the form (a) is twice as common. How can we capture this kind of information?