CFGs and PCFGs

(Probabilistic) Context-Free Grammars
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with

people fish tanks
people fish with rods
Phrase structure grammars  
= context-free grammars (CFGs)

- $G = (T, N, S, R)$
  - $T$ is a set of terminal symbols
  - $N$ is a set of nonterminal symbols
  - $S$ is the start symbol ($S \in N$)
  - $R$ is a set of rules/productions of the form $X \rightarrow \gamma$
    - $X \in N$ and $\gamma \in (N \cup T)^*$

- A grammar $G$ generates a language $L$.
Phrase structure grammars in NLP

- $G = (T, C, N, S, L, R)$
  - $T$ is a set of terminal symbols
  - $C$ is a set of preterminal symbols
  - $N$ is a set of nonterminal symbols
  - $S$ is the start symbol ($S \in N$)
  - $L$ is the lexicon, a set of items of the form $X \rightarrow x$
    - $X \in P$ and $x \in T$
    - $R$ is the grammar, a set of items of the form $X \rightarrow \gamma$
      - $X \in N$ and $\gamma \in (N \cup C)^*$

- By usual convention, $S$ is the start symbol, but in statistical NLP, we usually have an extra node at the top (ROOT, TOP)
- We usually write $e$ for an empty sequence, rather than nothing
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with

people fish tanks
people fish with rods
Probabilistic – or stochastic – context-free grammars (PCFGs)

- $G = (T, N, S, R, P)$
  - $T$ is a set of terminal symbols
  - $N$ is a set of nonterminal symbols
  - $S$ is the start symbol ($S \in N$)
  - $R$ is a set of rules/productions of the form $X \rightarrow \gamma$
  - $P$ is a probability function
    - $P: R \rightarrow [0,1]$
    - $\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$

- A grammar $G$ generates a language model $L$.
  $$\sum_{\gamma \in T^*} P(\gamma) = 1$$
### A PCFG

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.6</td>
</tr>
<tr>
<td>VP → V NP PP</td>
<td>0.4</td>
</tr>
<tr>
<td>NP → NP NP</td>
<td>0.1</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → N</td>
<td>0.7</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
</tr>
<tr>
<td>N → <em>people</em></td>
<td>0.5</td>
</tr>
<tr>
<td>N → <em>fish</em></td>
<td>0.2</td>
</tr>
<tr>
<td>N → <em>tanks</em></td>
<td>0.2</td>
</tr>
<tr>
<td>N → <em>rods</em></td>
<td>0.1</td>
</tr>
<tr>
<td>V → <em>people</em></td>
<td>0.1</td>
</tr>
<tr>
<td>V → <em>fish</em></td>
<td>0.6</td>
</tr>
<tr>
<td>V → <em>tanks</em></td>
<td>0.3</td>
</tr>
<tr>
<td>P → <em>with</em></td>
<td>1.0</td>
</tr>
</tbody>
</table>

[With empty NP removed so less ambiguous]
The probability of trees and strings

- $P(t)$ – The probability of a tree $t$ is the product of the probabilities of the rules used to generate it.
- $P(s)$ – The probability of the string $s$ is the sum of the probabilities of the trees which have that string as their yield.

\[
P(s) = \sum_j P(s, t) \quad \text{where } t \text{ is a parse of } s
\]

\[= \sum_j P(t)\]
t₁:

```
S₁.0
  NP₀.7
    N₀.5
    people
  VP₀.4
    V₀.6
    fish
    NP₀.7
    N₀.2
    tanks
    P₁.0
    with
    NP₀.7
    N₀.1
    rods
```
t2:

S1.0

NP0.7

N0.5

people

VP0.6

V0.6

fish

NP0.2

NP0.7

N0.2

tanks

PP1.0

P1.0

with

NP0.7

N0.1

rods
Tree and String Probabilities

- $s = \textit{people fish tanks with rods}$
- $P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 = 0.0008232$
- $P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.2 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 = 0.00024696$
- $P(s) = P(t_1) + P(t_2) = 0.0008232 + 0.00024696 = 0.00107016$
t_1: S_{1.0}
  /    
NP_{0.7} VP_{0.4}
   /     
N_{0.5} V_{0.6} NP_{0.7} PP_{1.0}
  /     /     
people fish N_{0.2} P_{1.0} NP_{0.7}
       /     /     
tanks with N_{0.1} 
       /     
rods
t_2:
S_1.0
  NP_0.7
    N_0.5
    people
  VP_0.6
    V_0.6
    fish
  NP_0.2
    PP_1.0
      N_0.2
      tanks
      P_1.0
      with
      N_0.1
      rods
CFGs and PCFGs

(Probabilistic) Context-Free Grammars
Grammar Transforms

Restricting the grammar form for efficient parsing
Chomsky Normal Form

- All rules are of the form $X \rightarrow Y \ Z$ or $X \rightarrow w$
  - $X, Y, Z \in N$ and $w \in T$
- A transformation to this form doesn’t change the weak generative capacity of a CFG
  - That is, it recognizes the same language
    - But maybe with different trees
- Empties and unaries are removed recursively
- n-ary rules are divided by introducing new nonterminals ($n > 2$)
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

S → NP VP
S → VP
VP → V NP
VP → V
VP → V NP PP
VP → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V
S → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
NP → N
PP → P NP
PP → P

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
S → *people*
V → *fish*
S → *fish*
V → *tanks*
S → *tanks*
P → *with*
Chomsky Normal Form steps

\[
\begin{align*}
S & \rightarrow NP \ VP \\
V P & \rightarrow V \ NP \\
S & \rightarrow V \ NP \\
V P & \rightarrow V \ NP \ PP \\
S & \rightarrow V \ NP \ PP \\
V P & \rightarrow V \ PP \\
S & \rightarrow V \ PP \\
N P & \rightarrow NP \ NP \\
N P & \rightarrow NP \\
N P & \rightarrow NP \ PP \\
N P & \rightarrow PP \\
N P & \rightarrow N \\
P P & \rightarrow P \ NP \\
P P & \rightarrow P \\
N & \rightarrow people \\
N & \rightarrow fish \\
N & \rightarrow tanks \\
N & \rightarrow rods \\
V & \rightarrow people \\
S & \rightarrow people \\
V P & \rightarrow people \\
V & \rightarrow fish \\
S & \rightarrow fish \\
V P & \rightarrow fish \\
V & \rightarrow tanks \\
S & \rightarrow tanks \\
V P & \rightarrow tanks \\
P & \rightarrow with
\end{align*}
\]
Chomsky Normal Form steps

\[ S \rightarrow NP \ VP \]
\[ VP \rightarrow V \ NP \]
\[ S \rightarrow V \ NP \]
\[ VP \rightarrow V \ NP \ PP \]
\[ S \rightarrow V \ NP \ PP \]
\[ VP \rightarrow V \ PP \]
\[ S \rightarrow V \ PP \]
\[ NP \rightarrow NP \ NP \]
\[ NP \rightarrow NP \ PP \]
\[ NP \rightarrow P \ NP \]
\[ PP \rightarrow P \ NP \]

\[ NP \rightarrow people \]
\[ NP \rightarrow fish \]
\[ NP \rightarrow tanks \]
\[ NP \rightarrow rods \]
\[ V \rightarrow people \]
\[ S \rightarrow people \]
\[ VP \rightarrow people \]
\[ V \rightarrow fish \]
\[ S \rightarrow fish \]
\[ VP \rightarrow fish \]
\[ V \rightarrow tanks \]
\[ S \rightarrow tanks \]
\[ VP \rightarrow tanks \]
\[ P \rightarrow with \]
\[ PP \rightarrow with \]
## Chomsky Normal Form steps

\[
\begin{align*}
S & \rightarrow \text{NP VP} & \text{NP} & \rightarrow \text{people} \\
\text{VP} & \rightarrow \text{V NP} & \text{NP} & \rightarrow \text{fish} \\
S & \rightarrow \text{V NP} & \text{NP} & \rightarrow \text{tanks} \\
\text{VP} & \rightarrow \text{V }@\text{VP_V} & \text{NP} & \rightarrow \text{rods} \\
@\text{VP_V} & \rightarrow \text{NP PP} & \text{V} & \rightarrow \text{people} \\
S & \rightarrow \text{V }@\text{S_V} & \text{S} & \rightarrow \text{people} \\
@\text{S_V} & \rightarrow \text{NP PP} & \text{VP} & \rightarrow \text{people} \\
\text{VP} & \rightarrow \text{V PP} & \text{V} & \rightarrow \text{fish} \\
S & \rightarrow \text{V PP} & \text{S} & \rightarrow \text{fish} \\
\text{NP} & \rightarrow \text{NP NP} & \text{VP} & \rightarrow \text{fish} \\
\text{NP} & \rightarrow \text{NP PP} & \text{V} & \rightarrow \text{tanks} \\
\text{NP} & \rightarrow \text{P NP} & \text{S} & \rightarrow \text{tanks} \\
\text{PP} & \rightarrow \text{P NP} & \text{VP} & \rightarrow \text{tanks} \\
\text{PP} & \rightarrow \text{P NP} & \text{P} & \rightarrow \text{with} \\
\end{align*}
\]
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V @VP_V
@VP_V → NP PP
S → V @S_V
@S_V → NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → people
NP → fish
NP → tanks
NP → rods
V → people
S → people
VP → people
V → fish
S → fish
VP → fish
V → tanks
S → tanks
VP → tanks
P → with
PP → with
Chomsky Normal Form

- You should think of this as a transformation for efficient parsing
- With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
- In practice full Chomsky Normal Form is a pain
  - Reconstructing n-aries is easy
  - Reconstructing unaries/empties is trickier

- **Binarization** is crucial for cubic time CFG parsing

- The rest isn’t necessary; it just makes the algorithms cleaner and a bit quicker
An example: before binarization...

```
S
  NP
    N  V  NP  PP
    people fish tanks with rods
```

ROOT
After binarization...
Treebank: empties and unaries

PTB Tree  NoFuncTags  NoEmpties  High  Low
NoUnaries
Unary rules: alchemy in the land of treebanks
Same-Span Reachability

- ADJP
- ADVP
- FRAG
- INTJ
- NP
- PP
- PRN
- QP
- S
- SBAR
- UCP
- VP
- WHNP
- SINV
- X
- RRC
- SQ
- LST
- CONJP
- NAC
- WHADJP
- SBARQ
- WHPP
- WHADVP
- NoEmpties
Grammar Transforms

Restricting the grammar form for efficient parsing
CKY Parsing

Exact polynomial time parsing of (P)CFGs
Constituency Parsing

PCFG

Rule Prob $\theta_i$

$S \rightarrow NP \ VP \ \ \ \theta_0$

$NP \rightarrow NP \ NP \ \ \ \ \theta_1$

$...$

$N \rightarrow fish \ \ \ \ \theta_{42}$

$N \rightarrow people \ \ \ \ \theta_{43}$

$V \rightarrow fish \ \ \ \ \theta_{44}$

$...$
Cocke-Kasami-Younger (CKY)
Constituency Parsing

fish  people  fish  tanks
Viterbi (Max) Scores

NP → NN NNS 0.13
\[ i_{NP} = (0.13)(0.0023) \]
\[ = 1.87 \times 10^{-7} \]

NP → NNP NNS 0.056
\[ i_{NP} = (0.056)(0.001) \]
\[ = 7.84 \times 10^{-8} \]
**Viterbi (Max) Scores**

```
S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
```
Extended CKY parsing

- Unaries can be incorporated into the algorithm
  - Messy, but doesn’t increase algorithmic complexity
- Empties can be incorporated
  - Use fenceposts
  - Doesn’t increase complexity; essentially like unaries

- Binarization is *vital*
  - Without binarization, you don’t get parsing cubic in the length of the sentence and in the number of nonterminals in the grammar
    - Binarization may be an explicit transformation or implicit in how the parser works (Early-style dotted rules), but it’s always there.
The CKY algorithm (1960/1965)  
... extended to unaries

function CKY(words, grammar) returns [most_probable_parse,prob]
  score = new double[#(words)+1][#(words)+1][#(nonterms)]
  back = new Pair[#(words)+1][#(words)+1][#nonterms]
  for i=0; i<#(words); i++
      for A in nonterms
          if A -> words[i] in grammar
              score[i][i+1][A] = P(A -> words[i])
  //handle unaries
  boolean added = true
  while added
      added = false
      for A, B in nonterms
          if score[i][i+1][B] > 0 && A->B in grammar
              prob = P(A->B)*score[i][i+1][B]
              if prob > score[i][i+1][A]
                  score[i][i+1][A] = prob
                  back[i][i+1][A] = B
                  added = true
The CKY algorithm (1960/1965)  
... extended to unaries

for span = 2 to #(words)
  for begin = 0 to #(words)- span
      end = begin + span
      for split = begin+1 to end-1
          for A,B,C in nonterms
              prob = score[begin][split][B]*score[split][end][C]*P(A→BC)
              if prob > score[begin][end][A]
                  score[begin][end][A] = prob
                  back[begin][end][A] = new Triple(split,B,C)
          //handle unaries
          boolean added = true
          while added
              added = false
              for A, B in nonterms
                  prob = P(A→B)*score[begin][end][B];
                  if prob > score[begin][end][A]
                      score[begin][end][A] = prob
                      back[begin][end][A] = B
                      added = true
          return buildTree(score, back)
Quiz Question!

What constituents (with what probability can you make?)

- PP → IN: 0.002
- NP → NNS NNS: 0.01
- NP → NNS NP: 0.005
- NP → NNS PP: 0.01
- VP → VB PP: 0.045
- VP → VB NP: 0.015

NNS 0.0023
VB 0.001

PP 0.2
IN 0.0014
NNS 0.0001

runs
down
CKY Parsing

Exact polynomial time parsing of (P)CFGs
CKY Parsing

A worked example
The grammar: Binary, no ε transitions,

\[
\begin{align*}
S & \rightarrow NP \ VP \quad 0.9 \\
S & \rightarrow VP \quad \quad 0.1 \\
VP & \rightarrow V \ NP \quad 0.5 \\
VP & \rightarrow V \quad \quad 0.1 \\
VP & \rightarrow V @VP_V \quad 0.3 \\
VP & \rightarrow V PP \quad 0.1 \\
@VP_V & \rightarrow NP PP \quad 1.0 \\
NP & \rightarrow NP \ NP \quad 0.1 \\
NP & \rightarrow NP PP \quad 0.2 \\
NP & \rightarrow N \quad \quad 0.7 \\
PP & \rightarrow P NP \quad 1.0 \\
N & \rightarrow people \quad 0.5 \\
N & \rightarrow fish \quad \quad 0.2 \\
N & \rightarrow tanks \quad 0.2 \\
N & \rightarrow rods \quad 0.1 \\
V & \rightarrow people \quad 0.1 \\
V & \rightarrow fish \quad \quad 0.6 \\
V & \rightarrow tanks \quad 0.3 \\
P & \rightarrow with \quad \quad 1.0
\end{align*}
\]
S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0

N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0

for i=0; i<#(words); i++
for A in nonterms
    if A -> words[i] in grammar
        score[i][i+1][A] = P(A -> words[i]);
### Grammar Rules

- **S**: NP VP (0.9)
- **S**: VP (0.1)
- **VP**: V NP (0.5)
- **VP**: V (0.1)
- **VP**: V @VP_V (0.3)
- **VP**: V PP (0.1)
- **@VP_V**: NP PP (1.0)
- **NP**: NP NP (0.1)
- **NP**: NP PP (0.2)
- **NP**: N (0.7)
- **PP**: P NP (1.0)

### Non-terminal Symbols

- **N**: people (0.5)
- **N**: fish (0.2)
- **N**: tanks (0.2)
- **N**: rods (0.1)
- **V**: people (0.1)
- **V**: fish (0.6)
- **V**: tanks (0.3)
- **P**: with (1.0)

### Score Matrix

<table>
<thead>
<tr>
<th></th>
<th>fish</th>
<th>people</th>
<th>fish</th>
<th>tanks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N → fish 0.2</td>
<td>V → fish 0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>N → people 0.5</td>
<td>V → people 0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>N → fish 0.2</td>
<td>V → fish 0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>N → tanks 0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### // Handle Unaries

```java
boolean added = true
while added
    added = false
    for A, B in nonterms
        if score[i][i+1][B] > 0 & A->B in grammar
            prob = P(A->B)*score[i][i+1][B]
            if(prob > score[i][i+1][A])
                score[i][i+1][A] = prob
                back[i][i+1][A] = B
                added = true
```

```
```
<table>
<thead>
<tr>
<th></th>
<th>fish</th>
<th>people</th>
<th>fish</th>
<th>tanks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N → fish 0.2</td>
<td>NP → NP NP 0.0049</td>
<td>NP → NP NP 0.0049</td>
<td>NP → NP NP 0.0049</td>
</tr>
<tr>
<td></td>
<td>V → fish 0.6</td>
<td>VP → V NP 0.105</td>
<td>VP → V NP 0.105</td>
<td>VP → V NP 0.105</td>
</tr>
<tr>
<td></td>
<td>NP → N 0.14</td>
<td>S → NP VP 0.00126</td>
<td>S → NP VP 0.0189</td>
<td>S → VP 0.0189</td>
</tr>
<tr>
<td>1</td>
<td>N → people 0.5</td>
<td>NP → N 0.35</td>
<td>NP → N 0.35</td>
<td>NP → N 0.35</td>
</tr>
<tr>
<td></td>
<td>V → people 0.1</td>
<td>VP → V 0.01</td>
<td>VP → V 0.01</td>
<td>VP → V 0.01</td>
</tr>
<tr>
<td></td>
<td>NP → N 0.35</td>
<td>S → NP VP 0.0189</td>
<td>S → NP VP 0.0189</td>
<td>S → NP VP 0.0189</td>
</tr>
<tr>
<td>2</td>
<td>N → fish 0.2</td>
<td>NP → NP NP 0.00196</td>
<td>NP → NP NP 0.00196</td>
<td>NP → NP NP 0.00196</td>
</tr>
<tr>
<td></td>
<td>V → fish 0.6</td>
<td>VP → V NP 0.042</td>
<td>VP → V NP 0.042</td>
<td>VP → V NP 0.042</td>
</tr>
<tr>
<td></td>
<td>NP → N 0.14</td>
<td>S → VP 0.006</td>
<td>S → VP 0.006</td>
<td>S → VP 0.006</td>
</tr>
<tr>
<td></td>
<td>VP → V 0.06</td>
<td>S → VP 0.00126</td>
<td>S → VP 0.00126</td>
<td>S → VP 0.00126</td>
</tr>
<tr>
<td>3</td>
<td>N → tanks 0.2</td>
<td>N → tanks 0.2</td>
<td>N → tanks 0.2</td>
<td>N → tanks 0.2</td>
</tr>
<tr>
<td></td>
<td>V → tanks 0.1</td>
<td>NP → N 0.14</td>
<td>NP → N 0.14</td>
<td>NP → N 0.14</td>
</tr>
<tr>
<td></td>
<td>VP → V 0.03</td>
<td>S → VP 0.003</td>
<td>S → VP 0.003</td>
<td>S → VP 0.003</td>
</tr>
<tr>
<td></td>
<td>S → VP 0.001</td>
<td>P → with 1.0</td>
<td>P → with 1.0</td>
<td>P → with 1.0</td>
</tr>
<tr>
<td>Rule</td>
<td>Probability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S → NP VP</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S → VP</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP → V</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP → V @VP_V</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP → V PP</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@VP_V → NP PP</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP → NP NP</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP → N</td>
<td>0.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N → people</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N → fish</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N → tanks</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N → rods</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V → people</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V → fish</td>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V → tanks</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P → with</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

for split = begin+1 to end-1
for A,B,C in nonterms
prob = score[begin][split][B] * score[split][end][C] * P(A→BC)
if prob > score[begin][end][A]
    score[begin][end][A] = prob
back[begin][end][A] = new Triple(split,B,C)
<table>
<thead>
<tr>
<th>Nonterm</th>
<th>Production</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>NP VP</td>
<td>0.9</td>
</tr>
<tr>
<td>S</td>
<td>VP</td>
<td>0.1</td>
</tr>
<tr>
<td>VP</td>
<td>V NP</td>
<td>0.5</td>
</tr>
<tr>
<td>VP</td>
<td>V</td>
<td>0.1</td>
</tr>
<tr>
<td>VP</td>
<td>V @VP_V</td>
<td>0.3</td>
</tr>
<tr>
<td>VP</td>
<td>V PP</td>
<td>0.1</td>
</tr>
<tr>
<td>@VP_V</td>
<td>NP PP</td>
<td>1.0</td>
</tr>
<tr>
<td>NP</td>
<td>NP NP</td>
<td>0.1</td>
</tr>
<tr>
<td>NP</td>
<td>NP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP</td>
<td>N</td>
<td>0.7</td>
</tr>
<tr>
<td>PP</td>
<td>P NP</td>
<td>1.0</td>
</tr>
<tr>
<td>N</td>
<td>people</td>
<td>0.5</td>
</tr>
<tr>
<td>N</td>
<td>fish</td>
<td>0.2</td>
</tr>
<tr>
<td>N</td>
<td>tanks</td>
<td>0.2</td>
</tr>
<tr>
<td>N</td>
<td>rods</td>
<td>0.1</td>
</tr>
<tr>
<td>V</td>
<td>people</td>
<td>0.1</td>
</tr>
<tr>
<td>V</td>
<td>fish</td>
<td>0.6</td>
</tr>
<tr>
<td>V</td>
<td>tanks</td>
<td>0.3</td>
</tr>
<tr>
<td>P</td>
<td>with</td>
<td>1.0</td>
</tr>
</tbody>
</table>

```
for split = begin+1 to end-1
    for A,B,C in nonterms
        prob = score[begin][split][B] * score[split][end][C] * P(A->BC)
        if prob > score[begin][end][A]
            score[begin][end][A] = prob
            back[begin][end][A] = new Triple(split,B,C)
```
### Probabilistic Parsing

#### Grammar Rules

- **NP**: Non-terminal symbols representing phrases.
- **VP**: Non-terminal symbols representing verb phrases.
- **S**: Non-terminal symbol representing the sentence.
- **N**: Non-terminal symbol representing nouns.
- **V**: Non-terminal symbol representing verbs.
- **P**: Non-terminal symbol representing prepositions.

#### Probabilities

- **NP → NP**: 0.0049
- **NP → N**: 0.14
- **VP → V**: 0.06
- **S → VP**: 0.006
- **N → fish**: 0.2
- **V → fish**: 0.6
- **VP → V**: 0.06
- **S → VP**: 0.006
- **N → people**: 0.5
- **V → people**: 0.1
- **NP → N**: 0.35
- **VP → V**: 0.01
- **S → VP**: 0.001
- **N → tanks**: 0.2
- **V → tanks**: 0.3
- **VP → V**: 0.03
- **S → VP**: 0.006

#### Parsing Algorithm

1. **for split = begin+1 to end-1**
2. **for A,B,C in nonterms**
3. **prob = score[begin][split][B] * score[split][end][C] * P(A → BC)**
4. **if prob > score[begin][end][A]**
5. **score[begin][end][A] = prob**
6. **back[begin][end][A] = new Triple(split,B,C)**
```
S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
V → with 1.0
P → with 1.0

Call buildTree(score, back) to get the best parse
```
CKY Parsing

A worked example
Constituency Parser Evaluation
Evaluating constituency parsing

Gold standard brackets: S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6-9), NP-(7,9), NP-(9:10)

Candidate brackets: S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:6), PP-(6-10), NP-(7,10)
Evaluating constituency parsing

Gold standard brackets:
S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6-9), NP-(7,9), NP-(9:10)

Candidate brackets:
S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:6), PP-(6-10), NP-(7,10)

Labeled Precision 3/7 = 42.9%
Labeled Recall 3/8 = 37.5%
LP/LR F1 40.0%
Tagging Accuracy 11/11 = 100.0%
How good are PCFGs?

- Penn WSJ parsing accuracy: about 73% LP/LR F1
- Robust
  - Usually admit everything, but with low probability
- Partial solution for grammar ambiguity
  - A PCFG gives some idea of the plausibility of a parse
  - But not so good because the independence assumptions are too strong
- Give a probabilistic language model
  - But in the simple case it performs worse than a trigram model
- The problem seems to be that PCFGs lack the lexicalization of a trigram model
Constituency Parser Evaluation