

Probabilistic Context Free Grammars

Many slides from Michael Collins and Chris Manning

Overview

- ▶ Probabilistic Context-Free Grammars (PCFGs)
- ▶ The CKY Algorithm for parsing with PCFGs

A Probabilistic Context-Free Grammar (PCFG)

S	⇒	NP	VP	1.0
VP	⇒	Vi		0.4
VP	⇒	Vt	NP	0.4
VP	⇒	VP	PP	0.2
NP	⇒	DT	NN	0.3
NP	⇒	NP	PP	0.7
PP	⇒	P	NP	1.0

Vi	⇒	sleeps	1.0
Vt	⇒	saw	1.0
NN	⇒	man	0.7
NN	⇒	woman	0.2
NN	⇒	telescope	0.1
DT	⇒	the	1.0
IN	⇒	with	0.5
IN	⇒	in	0.5

- ▶ Probability of a tree t with rules

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

is $p(t) = \prod_{i=1}^n q(\alpha_i \rightarrow \beta_i)$ where $q(\alpha \rightarrow \beta)$ is the probability for rule $\alpha \rightarrow \beta$.

DERIVATION

S

NP VP

DT NN VP

the NN VP

the dog VP

the dog Vi

the dog laughs

RULES USED

$S \rightarrow NP VP$

$NP \rightarrow DT NN$

$DT \rightarrow \text{the}$

$NN \rightarrow \text{dog}$

$VP \rightarrow V_i$

$V_i \rightarrow \text{laughs}$

PROBABILITY

1.0

0.3

1.0

0.1

0.4

0.5

Properties of PCFGs

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

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 $\mathcal{T}(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $\mathcal{T}(s)$. i.e., *we now have a ranking in order of probability.*

Properties of PCFGs

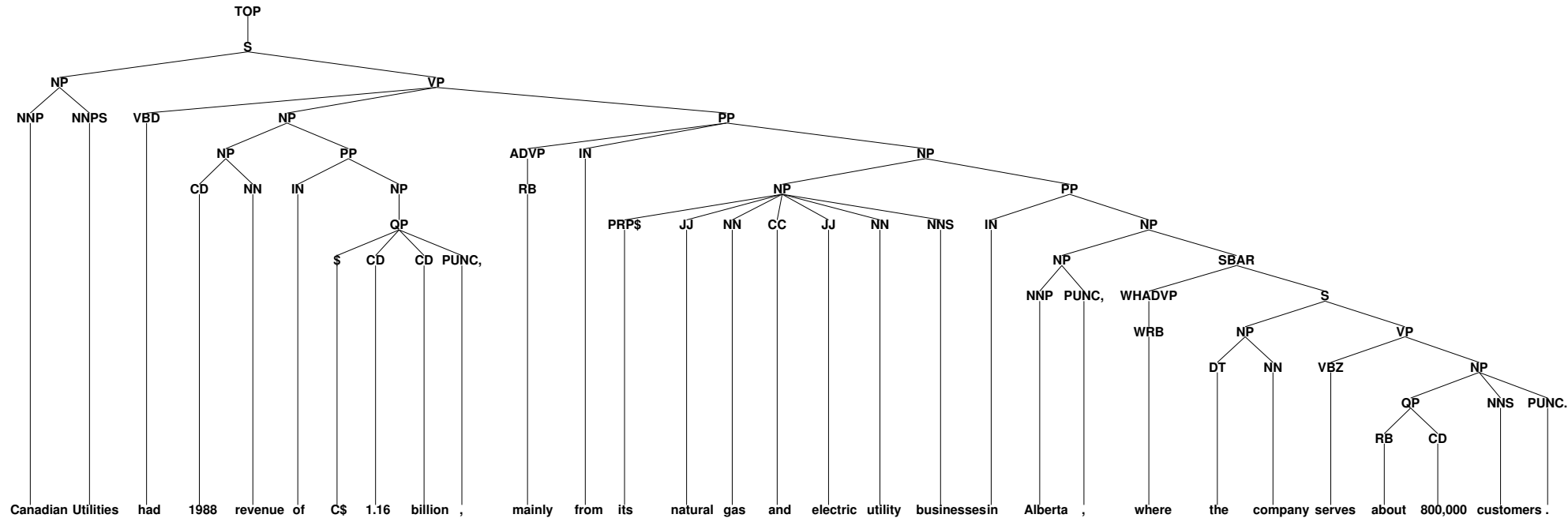
- ▶ Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG
- ▶ Say we have a sentence s , set of derivations for that sentence is $\mathcal{T}(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $\mathcal{T}(s)$. i.e., *we now have a ranking in order of probability.*
- ▶ The most likely parse tree for a sentence s is

$$\arg \max_{t \in \mathcal{T}(s)} p(t)$$

Data for Parsing Experiments: Treebanks

- ▶ Penn WSJ Treebank = 50,000 sentences with associated trees
- ▶ Usual set-up: 40,000 training sentences, 2400 test sentences

An example tree:



Deriving a PCFG from a Treebank

- ▶ Given a set of example trees (a treebank), the underlying CFG can simply be **all rules seen in the corpus**
- ▶ Maximum Likelihood estimates:

$$q_{ML}(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

where the counts are taken from a training set of example trees.

- ▶ **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.

Parsing with a PCFG

- ▶ Given a PCFG and a sentence s , define $\mathcal{T}(s)$ to be the set of trees with s as the yield.
- ▶ Given a PCFG and a sentence s , how do we find

$$\arg \max_{t \in \mathcal{T}(s)} p(t)$$

Chomsky Normal Form

A context free grammar $G = (N, \Sigma, R, S)$ in Chomsky Normal Form is as follows

- ▶ N is a set of non-terminal symbols
- ▶ Σ is a set of terminal symbols
- ▶ R is a set of rules which take one of two forms:
 - ▶ $X \rightarrow Y_1Y_2$ for $X \in N$, and $Y_1, Y_2 \in N$
 - ▶ $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$
- ▶ $S \in N$ is a distinguished start symbol

A Dynamic Programming Algorithm

- ▶ Given a PCFG and a sentence s , how do we find

$$\max_{t \in \mathcal{T}(s)} p(t)$$

- ▶ Notation:

n = number of words in the sentence

w_i = i 'th word in the sentence

N = the set of non-terminals in the grammar

S = the start symbol in the grammar

- ▶ Define a dynamic programming table

$\pi[i, j, X]$ = maximum probability of a constituent with non-terminal X
spanning words $i \dots j$ inclusive

- ▶ Our goal is to calculate $\max_{t \in \mathcal{T}(s)} p(t) = \pi[1, n, S]$

A Dynamic Programming Algorithm

- ▶ Base case definition: for all $i = 1 \dots n$, for $X \in N$

$$\pi[i, i, X] = q(X \rightarrow w_i)$$

(note: define $q(X \rightarrow w_i) = 0$ if $X \rightarrow w_i$ is not in the grammar)

- ▶ Recursive definition: for all $i = 1 \dots n$, $j = (i + 1) \dots n$,
 $X \in N$,

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

The Full Dynamic Programming Algorithm

Input: a sentence $s = x_1 \dots x_n$, a PCFG $G = (N, \Sigma, S, R, q)$.

Initialization:

For all $i \in \{1 \dots n\}$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Algorithm:

- ▶ For $l = 1 \dots (n - 1)$
 - ▶ For $i = 1 \dots (n - l)$
 - ▶ Set $j = i + l$
 - ▶ For all $X \in N$, calculate

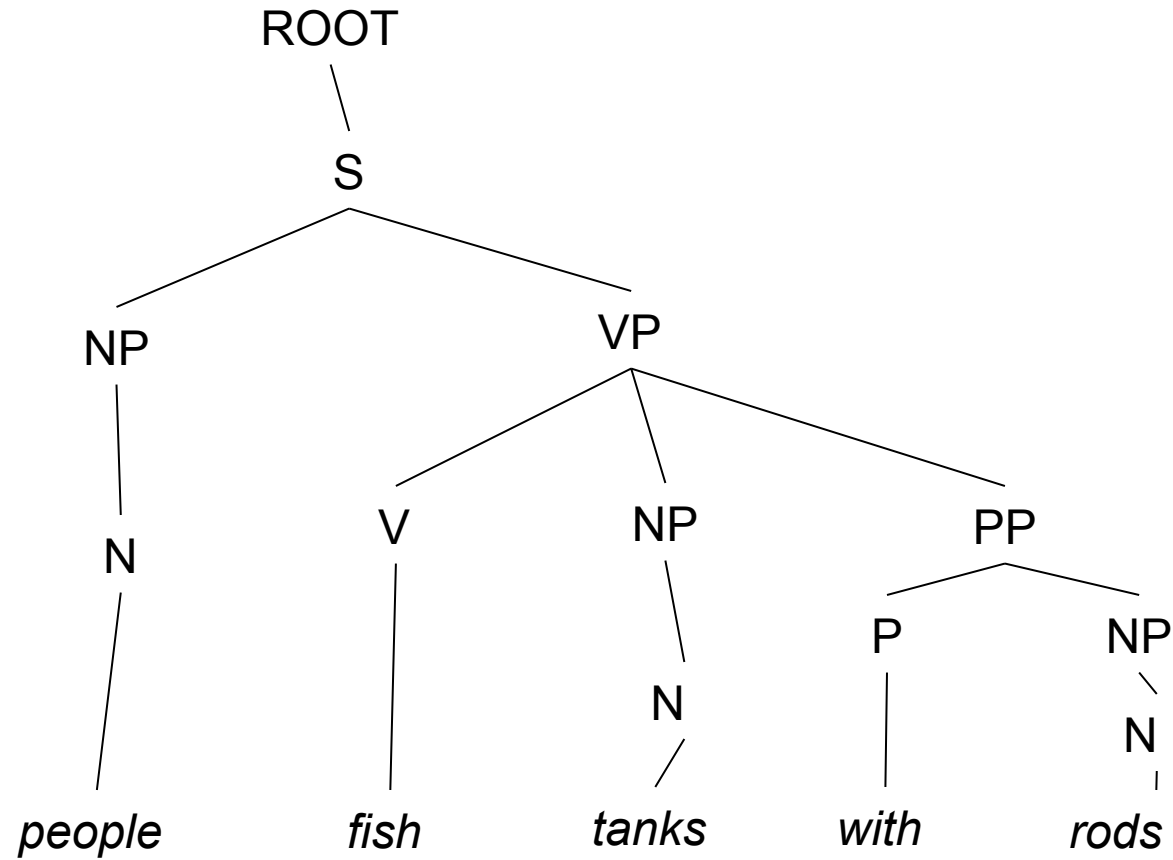
What's the run time Complexity?

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

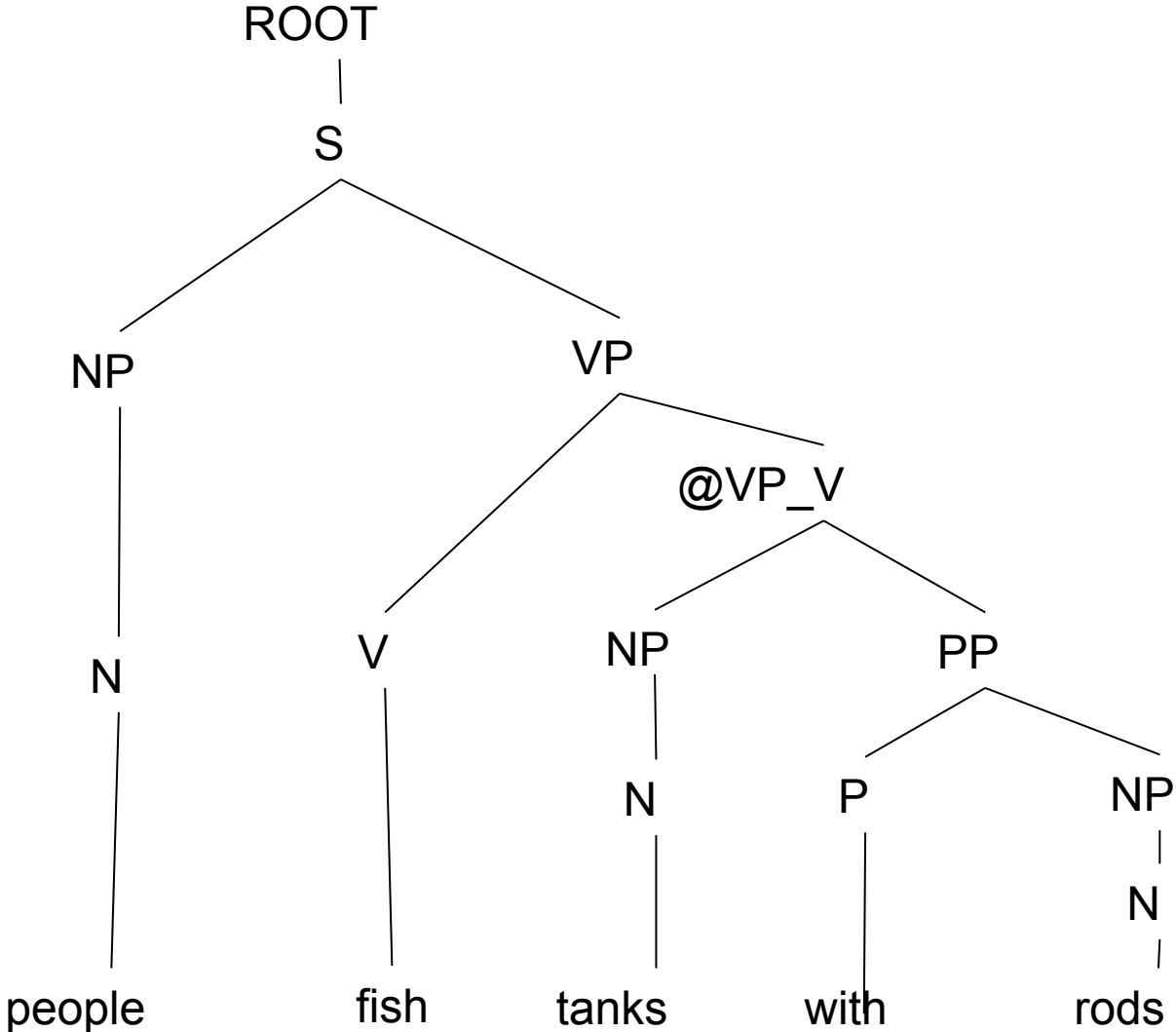
and

$$bp(i, j, X) = \arg \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

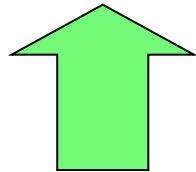
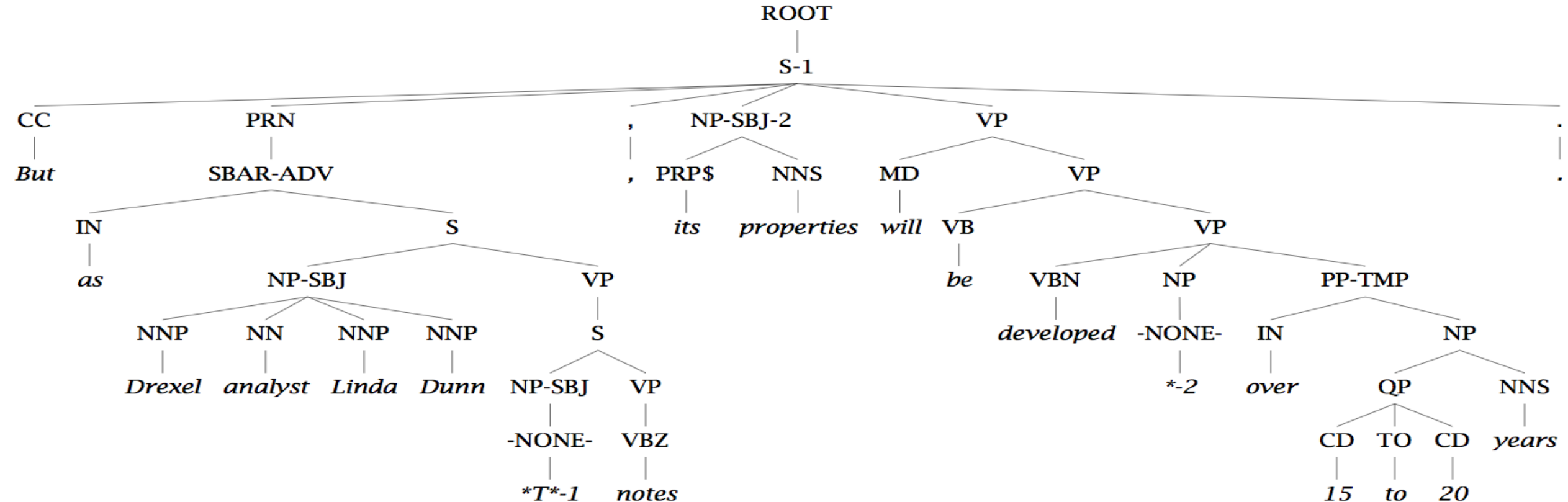
An example: before binarization...



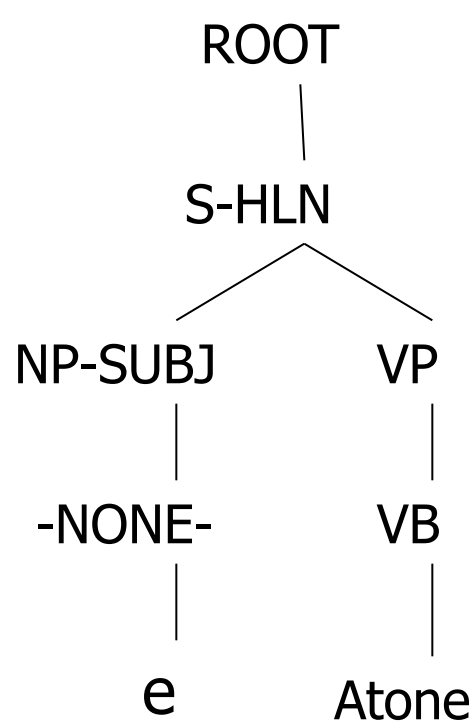
After binarization...



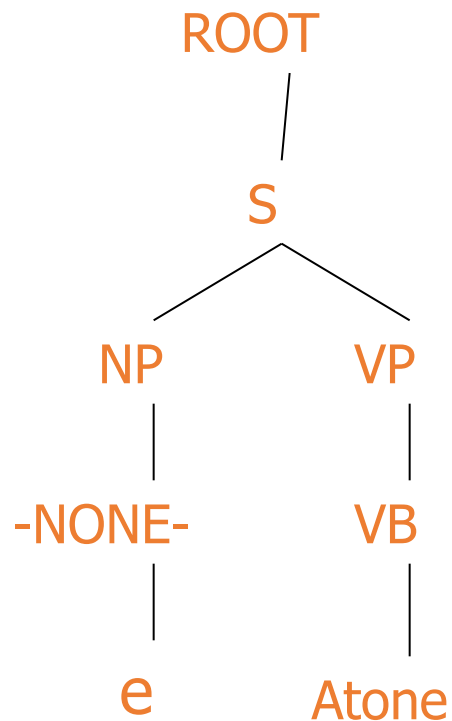
Unary rules: alchemy in the land of treebanks



Treebank: empties and unaries



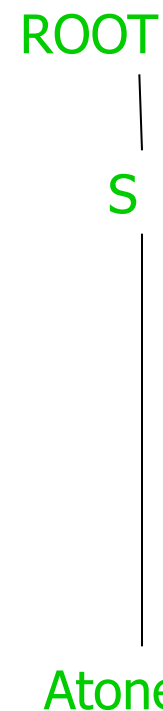
PTB Tree



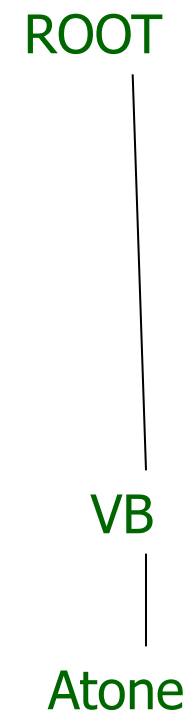
NoFuncTags



NoEmpties



High



Low

NoUnaries

Extended CKY parsing

- Unaries can be incorporated into the algorithm
 - Messy, but doesn't increase algorithmic complexity
- Empties can be incorporated
 - 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

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)
```

CKY Parsing

A worked example

The grammar: Binary, Unaries, no epsilons,

$S \rightarrow NP VP$ 0.9

$S \rightarrow VP$ 0.1

$VP \rightarrow V NP$ 0.5

$VP \rightarrow V$ 0.1

$VP \rightarrow V @VP_V$ 0.3

$VP \rightarrow V PP$ 0.1

$@VP_V \rightarrow NP PP$ 1.0

$NP \rightarrow NP NP$ 0.1

$NP \rightarrow NP PP$ 0.2

$NP \rightarrow N$ 0.7

$PP \rightarrow P NP$ 1.0

$N \rightarrow people$ 0.5

$N \rightarrow fish$ 0.2

$N \rightarrow tanks$ 0.2

$N \rightarrow rods$ 0.1

$V \rightarrow people$ 0.1

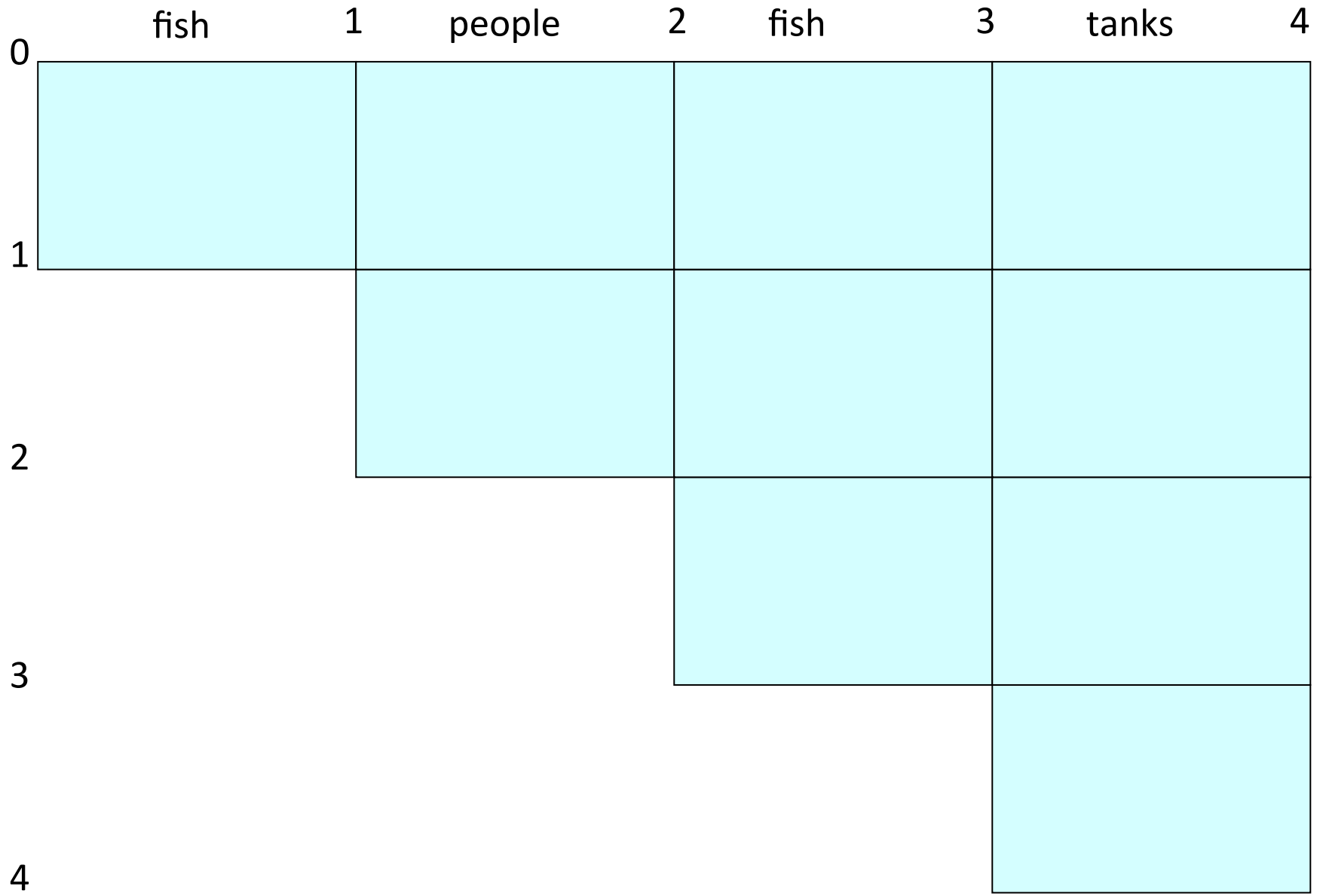
$V \rightarrow fish$ 0.6

$V \rightarrow tanks$ 0.3

$P \rightarrow with$ 1.0

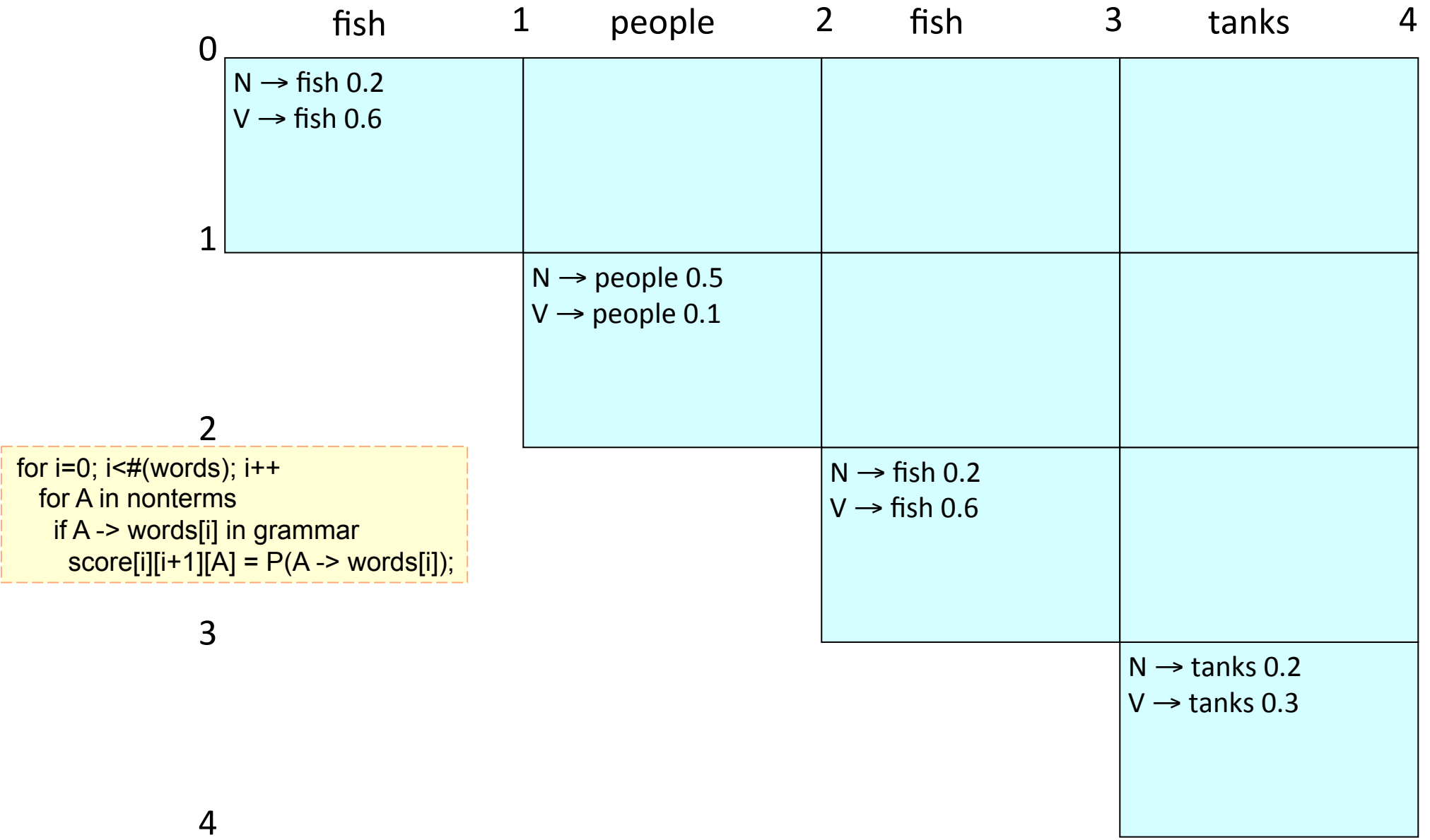
- S → NP VP 0.9
- S → VP 0.1
- VP → V NP 0.5
- VP → V 0.1
- VP → V @VP_V 0.3
- VP → V PP 0.1
- @VP_V → NP PP 1.0
- NP → NP NP 0.1
- NP → NP PP 0.2
- NP → N 0.7
- PP → P NP 1.0

- 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



- 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



- 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

	fish	1	people	2	fish	3	tanks	4					
0	<div style="border: 1px solid black; padding: 5px; background-color: #e0f7fa;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </div>												
1													
2													
			<div style="border: 1px solid black; padding: 5px; background-color: #e0f7fa;"> N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001 </div>										
				<div style="border: 1px solid black; padding: 5px; background-color: #e0f7fa;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </div>									
						<div style="border: 1px solid black; padding: 5px; background-color: #e0f7fa;"> N → tanks 0.2 V → tanks 0.3 NP → N 0.14 VP → V 0.03 S → VP 0.003 </div>							

```

// 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

```

- 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

	fish	1	people	2	fish	3	tanks	4
0								
1								
1	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006		NP → NP NP 0.0049 VP → V NP 0.105 S → NP VP 0.00126					
2			N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001		NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189			
3					N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006		NP → NP NP 0.00196 VP → V NP 0.042 S → NP VP 0.00378	
4							N → tanks 0.2 V → tanks 0.3 NP → N 0.14 VP → V 0.03 S → VP 0.003	

```

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)

```

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 → <i>people</i>	0.5
N → <i>fish</i>	0.2
N → <i>tanks</i>	0.2
N → <i>rods</i>	0.1
V → <i>people</i>	0.1
V → <i>fish</i>	0.6
V → <i>tanks</i>	0.3
P → <i>with</i>	1.0

	fish	1	people	2	fish	3	tanks	4
0								
1	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006		NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105					
2			N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001		NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189			
3					N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006		NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042	
4							N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003	

```
//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
```

- 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

	fish	1	people	2	fish	3	tanks	4
0	<div style="display: flex; justify-content: space-between;"> <div style="width: 25%;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </div> <div style="width: 25%;"> NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105 </div> <div style="width: 25%;"> NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882 </div> </div>							
1								
2								
3								
4	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)							

- 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

	fish	1	people	2	fish	3	tanks	4
0	<div style="display: flex; justify-content: space-between;"> <div style="width: 25%;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </div> <div style="width: 25%;"> NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105 </div> <div style="width: 25%;"> NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882 </div> <div style="width: 25%;"></div> </div>							
1								
2								
3								
4	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)							

- 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

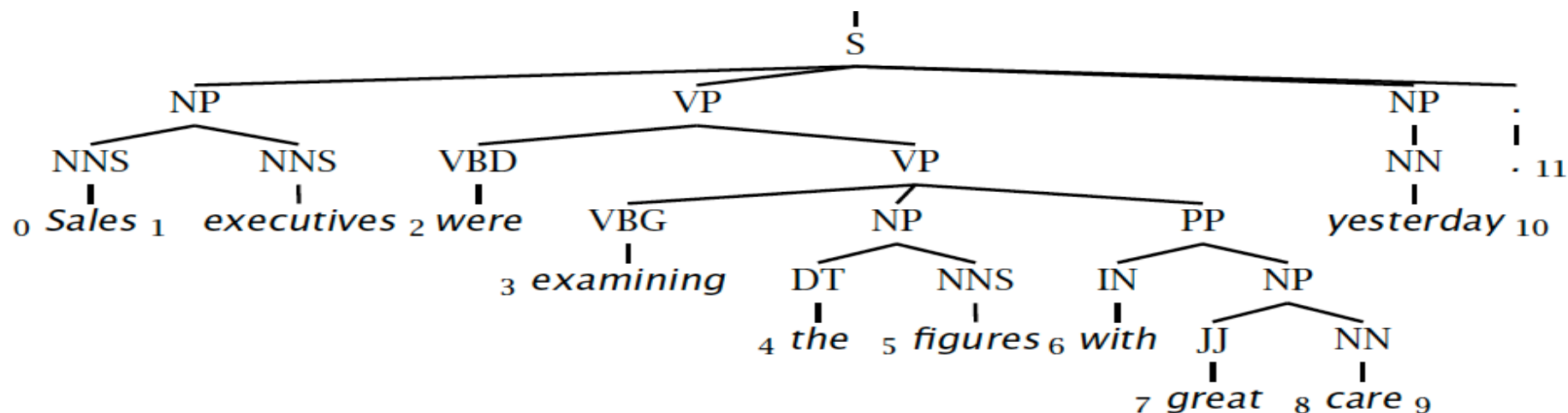
	fish	1	people	2	fish	3	tanks	4																																														
0	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.0000009604 VP → V NP 0.00002058 S → NP VP 0.00018522 </td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">1</td> <td colspan="8" rowspan="2" style="background-color: #e0ffff;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323 </td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">2</td> <td colspan="8" rowspan="2" style="background-color: #e0ffff;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042 </td> </tr> </table> </td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">3</td> <td colspan="8" rowspan="2" style="background-color: #ffff00; border: 1px dashed orange;"> <pre> 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) </pre> </td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">4</td> <td colspan="8" rowspan="2" style="background-color: #e0ffff;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003 </td> </tr> </table> </td> </tr> </table></td></tr></table>								N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	NP → NP NP 0.0000009604 VP → V NP 0.00002058 S → NP VP 0.00018522	1	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323 </td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">2</td> <td colspan="8" rowspan="2" style="background-color: #e0ffff;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042 </td> </tr> </table> </td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">3</td> <td colspan="8" rowspan="2" style="background-color: #ffff00; border: 1px dashed orange;"> <pre> 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) </pre> </td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">4</td> <td colspan="8" rowspan="2" style="background-color: #e0ffff;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003 </td> </tr> </table> </td> </tr> </table>								N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323	2	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042 </td> </tr> </table>								N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042	3	<pre> 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) </pre>								4	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003 </td> </tr> </table>								N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006									NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	NP → NP NP 0.0000009604 VP → V NP 0.00002058 S → NP VP 0.00018522																																											
1									<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323 </td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">2</td> <td colspan="8" rowspan="2" style="background-color: #e0ffff;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042 </td> </tr> </table> </td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">3</td> <td colspan="8" rowspan="2" style="background-color: #ffff00; border: 1px dashed orange;"> <pre> 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) </pre> </td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">4</td> <td colspan="8" rowspan="2" style="background-color: #e0ffff;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003 </td> </tr> </table> </td> </tr> </table>								N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323	2	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042 </td> </tr> </table>								N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042	3	<pre> 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) </pre>								4	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003 </td> </tr> </table>								N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003					
N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001																	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323																																				
2	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006 </td> <td style="width: 25%; padding: 5px;"> NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042 </td> </tr> </table>								N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042																																												
N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006									NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042																																													
3	<pre> 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) </pre>																																																					
4									<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;"> N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003 </td> </tr> </table>								N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003																																					
N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003																																																						

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

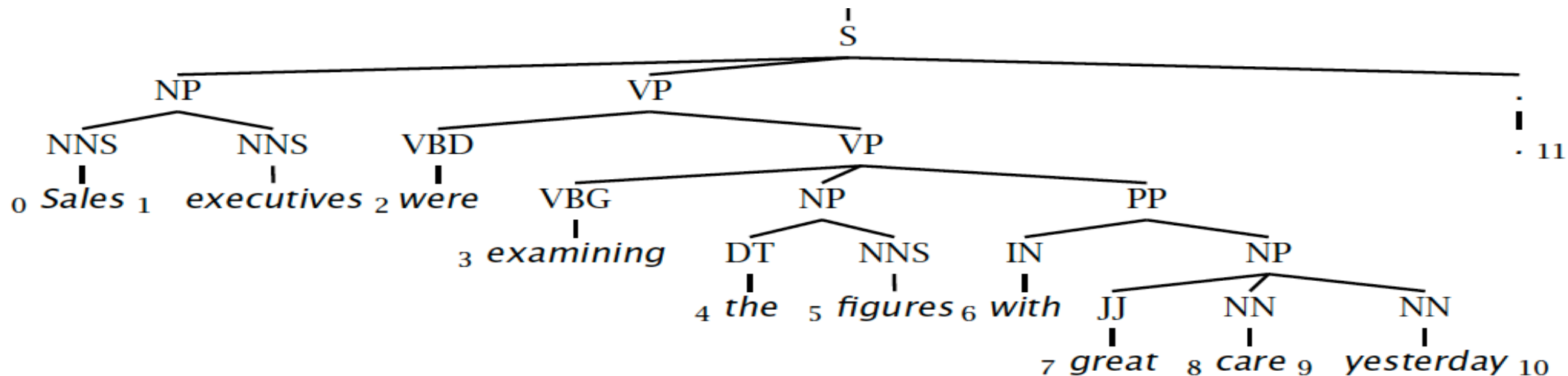
Constituency Parser Evaluation

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 $11/11 = 100.0\%$

Summary

- ▶ PCFGs augments CFGs by including a probability for each rule in the grammar.
- ▶ The probability for a parse tree is the product of probabilities for the rules in the tree
- ▶ To build a PCFG-parsed parser:
 1. Learn a PCFG from a treebank
 2. Given a test data sentence, use the CKY algorithm to compute the highest probability tree for the sentence under the PCFG

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