(Probabilistic) Context-Free Grammars

A phrase structure grammar

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

people fish tanks
people fish with rods

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

Ambiguous: People people people, fish fish fish

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 a start symbol (S ∈ 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 $X \rightarrow x$
    - $X \in C$ 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 (empty, unary, binary)

Grammar Rules

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → *e*

PP → P NP

EMPTY fish tanks

people fish EMPTY

Lexicon

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

V → *fish*

V → *tanks*

P → *with*

Probabilistic/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 ∈ 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 PCFG

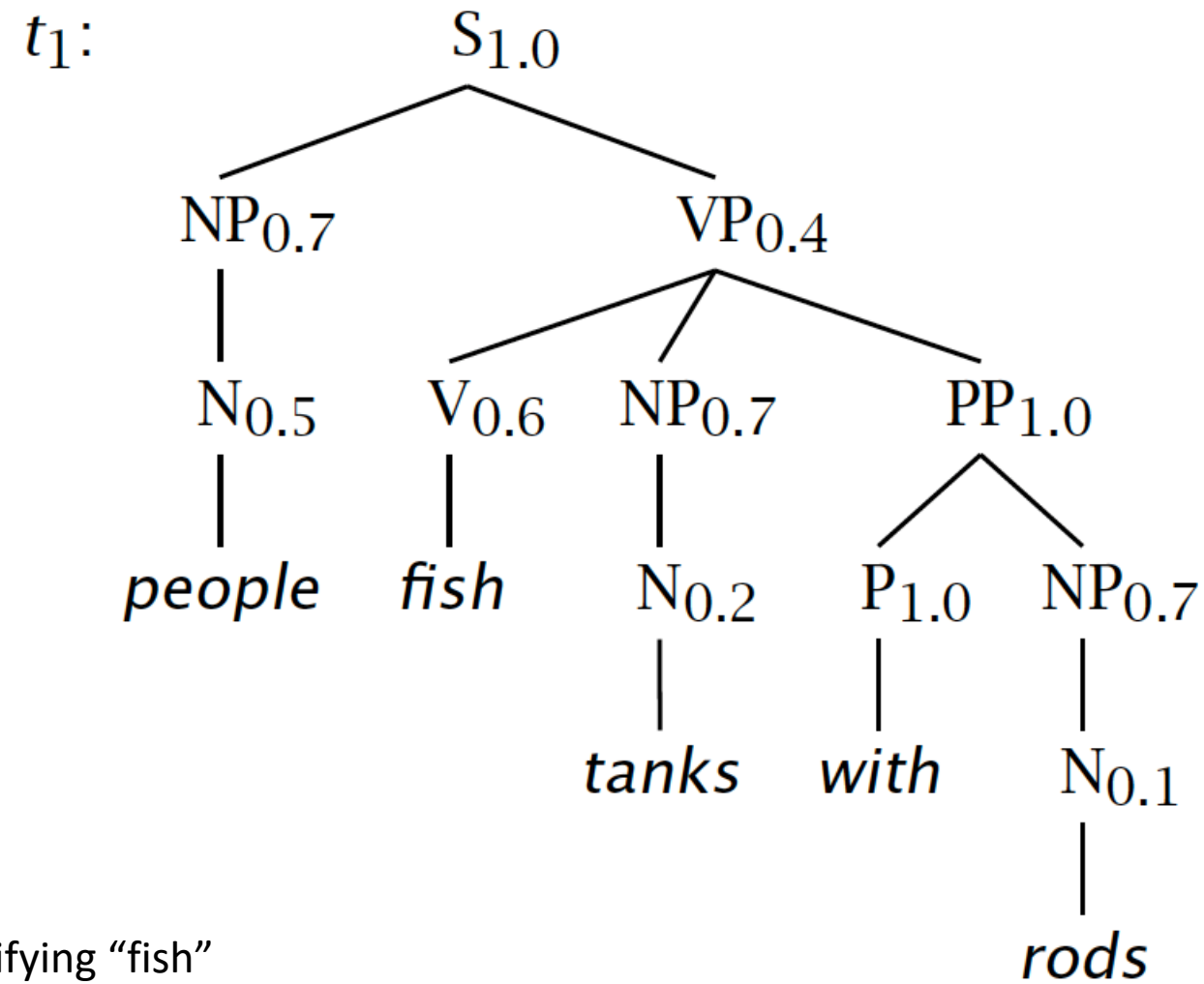
S → NP VP	1.0	
VP → V NP	0.6	
VP → V NP PP		0.4
NP → NP NP		0.1
NP → NP PP	0.2	
NP → N	0.7	
PP → P NP	1.0	

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

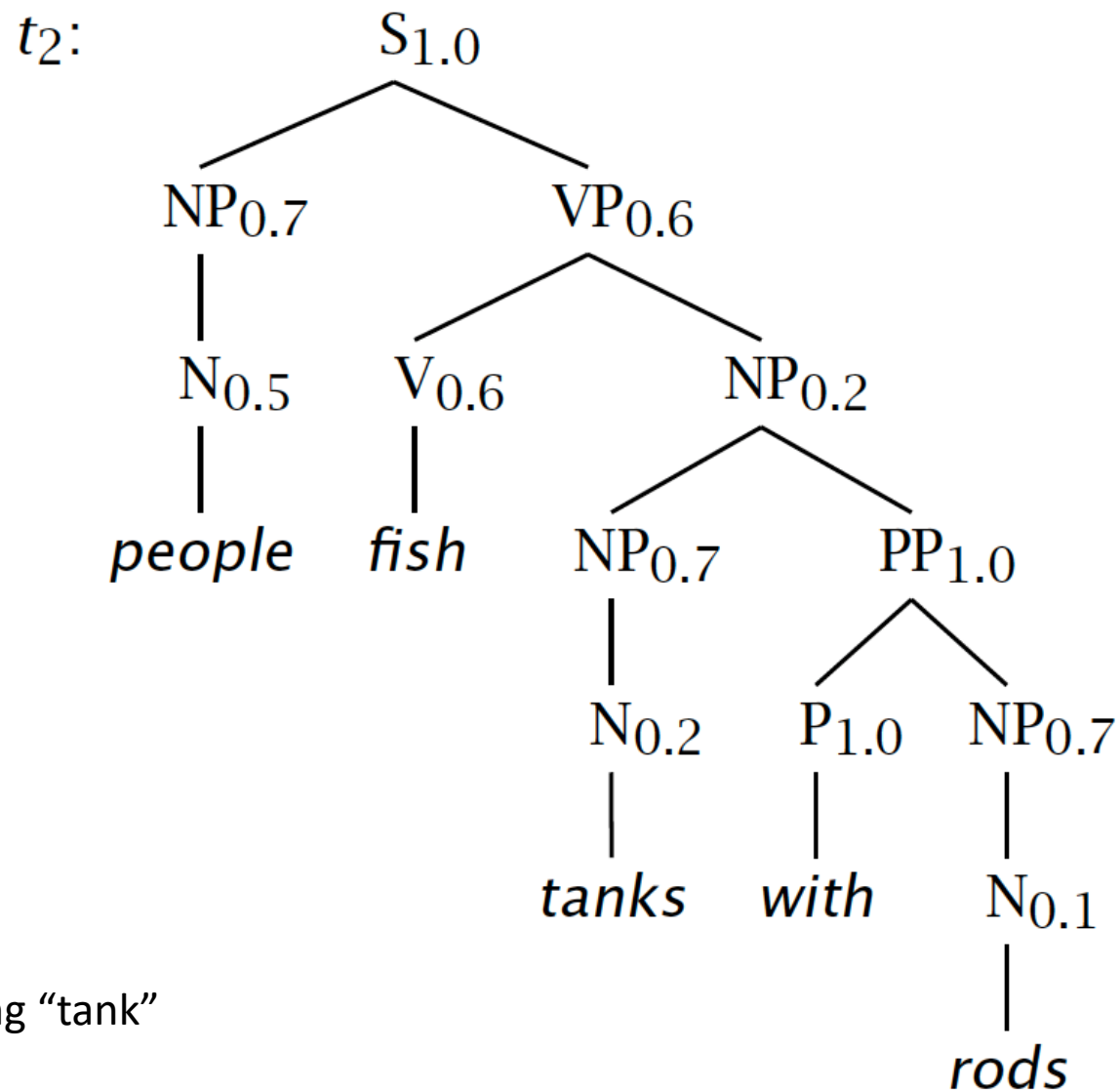
The probability of trees and strings

- 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

$$P(s) = \Sigma_t P(s, t) \text{ where } t \text{ is a parse of } s$$
$$= \Sigma_t P(t)$$



Preposition “with” modifying “fish”



Preposition “with” modifying “tank”

Tree and String Probabilities

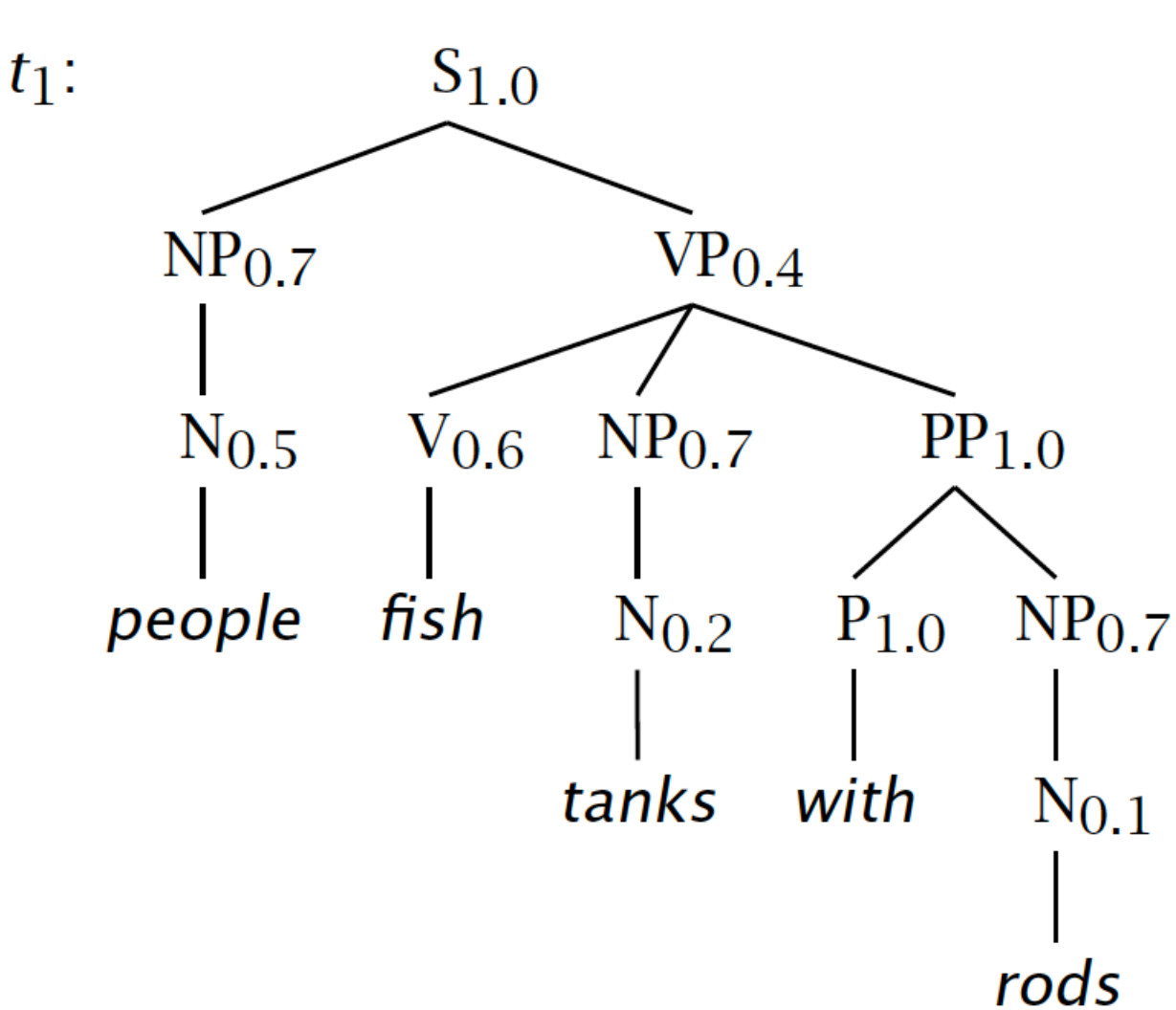
- *s* = *people fish tanks with rods*
- $P(t_1)$ = 1.0 × 0.7 × 0.4 × 0.5 × 0.6 × 0.7 × 1.0 × 0.2 × 1.0 × 0.7 × 0.1
  = 0.0008232

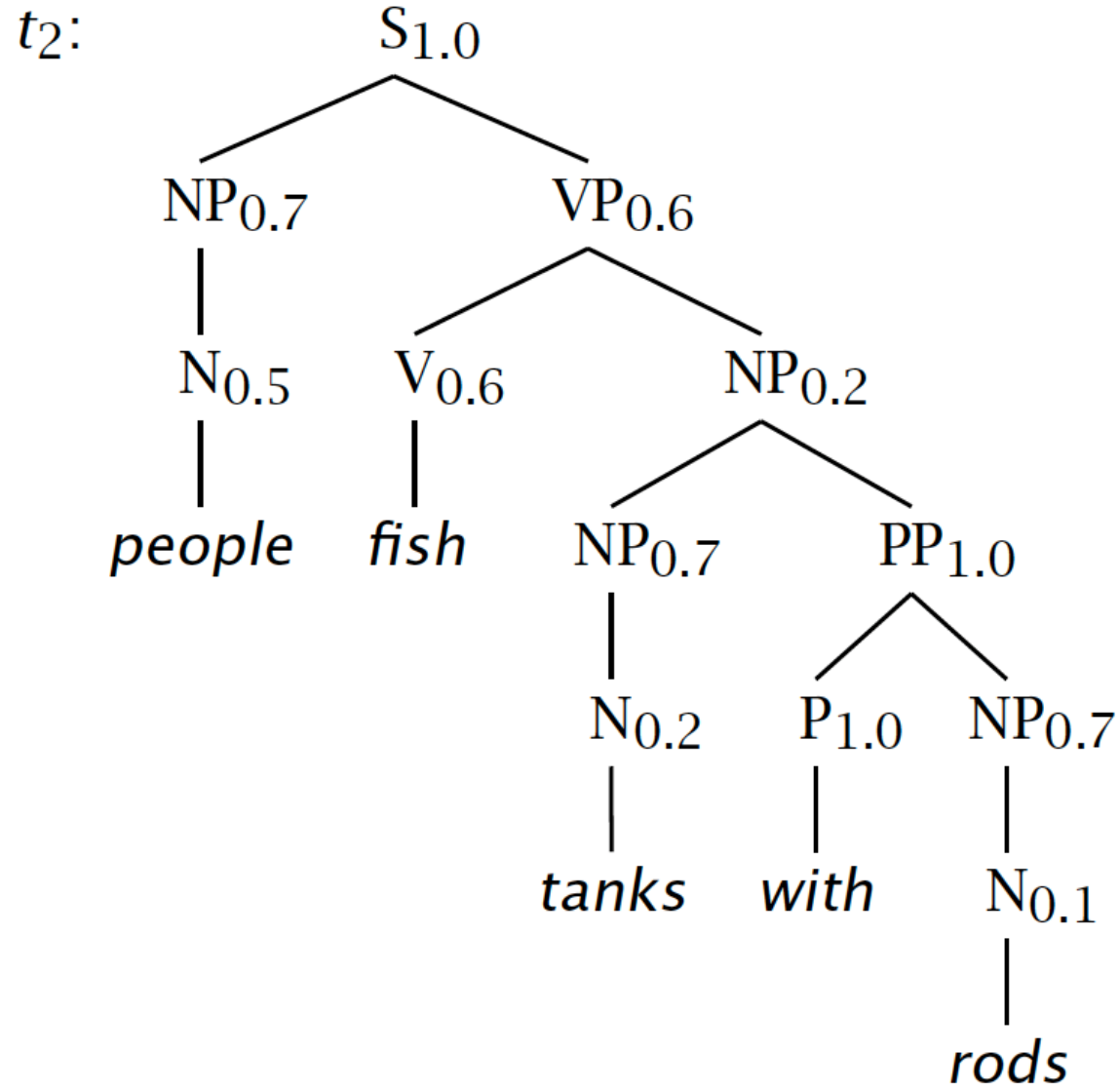
Verb attach

- $P(t_2)$ = 1.0 × 0.7 × 0.6 × 0.5 × 0.6 × 0.2 × 0.7 × 1.0 × 0.2 × 1.0 × 0.7 × 0.1
  = 0.00024696

Noun attach

- $P(s)$ = $P(t_1)$ + $P(t_2)$
  = 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:
$S_{1.0}$
$NP_{0.7}$
$VP_{0.6}$
$N_{0.5}$
$V_{0.6}$
$NP_{0.2}$
people
fish
$NP_{0.7}$
$PP_{1.0}$
$N_{0.2}$
$P_{1.0}$
$NP_{0.7}$
tanks
with
$N_{0.1}$
rods

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*

Start discussing epsilon removal

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*

Start discussing unary removal downwards: remove S → VP

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

PP → P NP

PP → P

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

V → *fish*

V → *tanks*

P → *with*

Remove more unaries, next S → V

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*

After remove S → V get this, and then do VP → V

Chomsky Normal Form steps

S → NP VP

VP → V NP

S → V NP

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*

VP → *people*

V → *fish*

S → *fish*

VP → *fish*

V → *tanks*

S → *tanks*

VP → *tanks*

P → *with*

Chomsky Normal Form steps

S → NP VP

VP → V NP

S → V NP

VP → V NP PP

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*

And then binarize

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*

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*

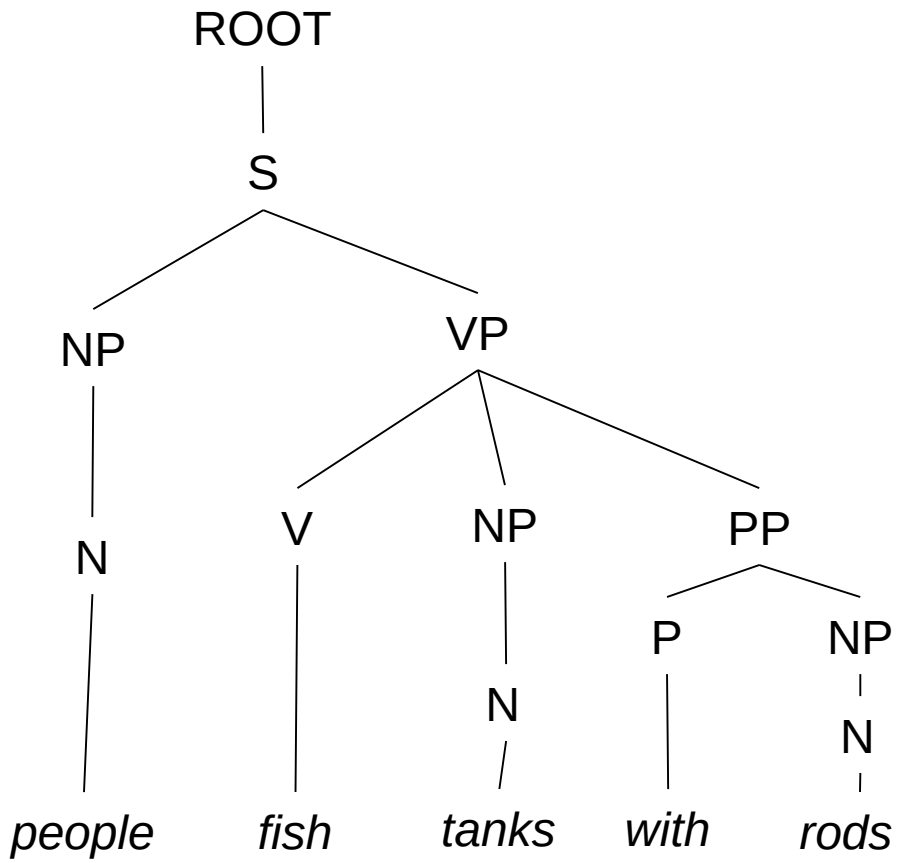
Chomsky Normal Form

- 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

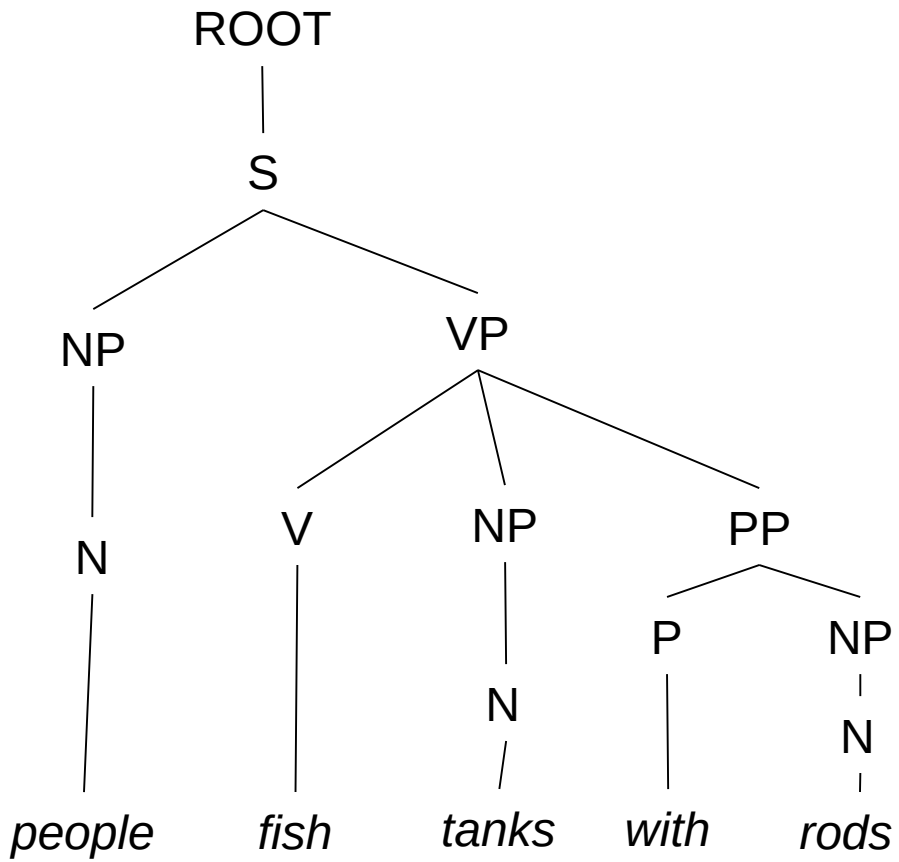
- **Binarization** is crucial for cubic time CFG parsing

- The rest isn't necessary; it just makes the algorithms cleaner and a bit quicker

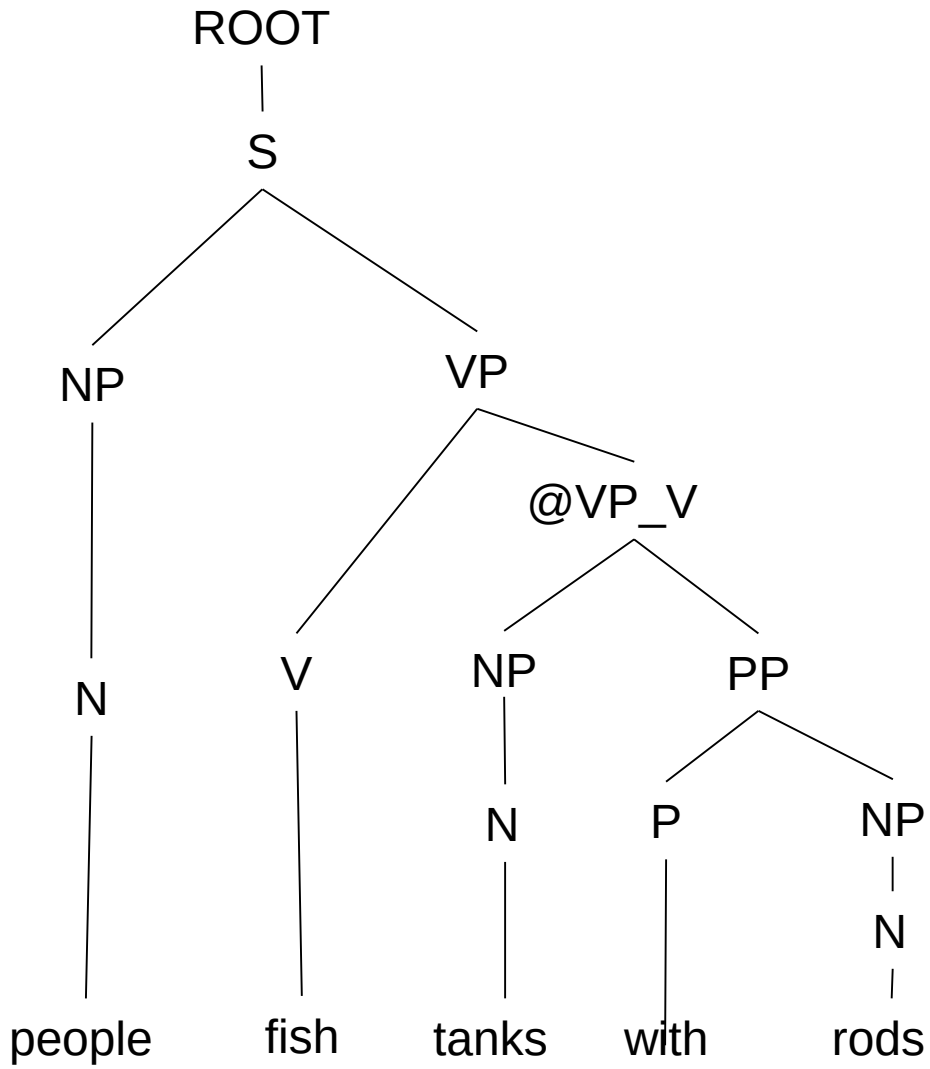
An example: before binarization…



An example: before binarization…



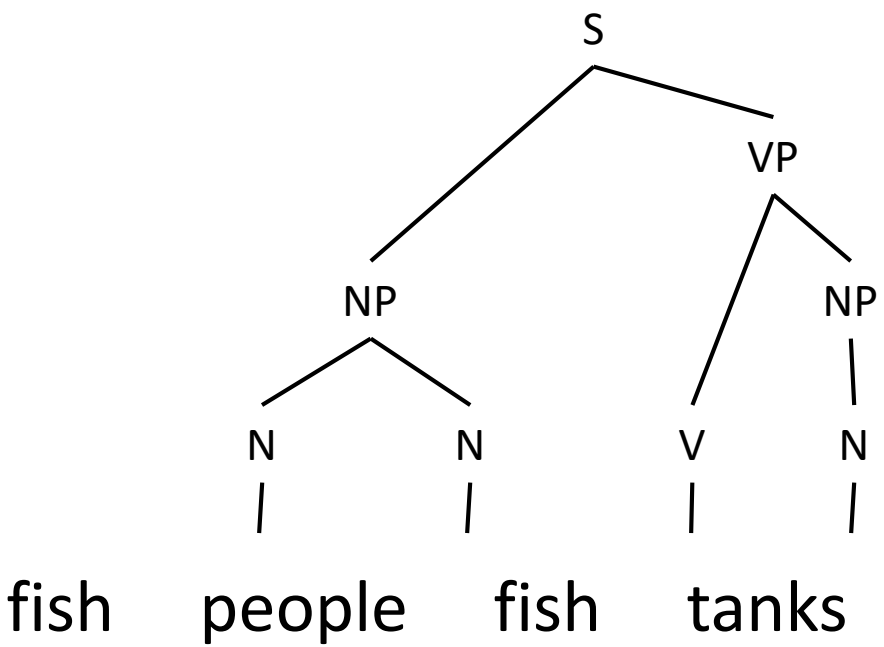
After binarization…



CKY Parsing

Exact polynomial time parsing of (P)CFGs

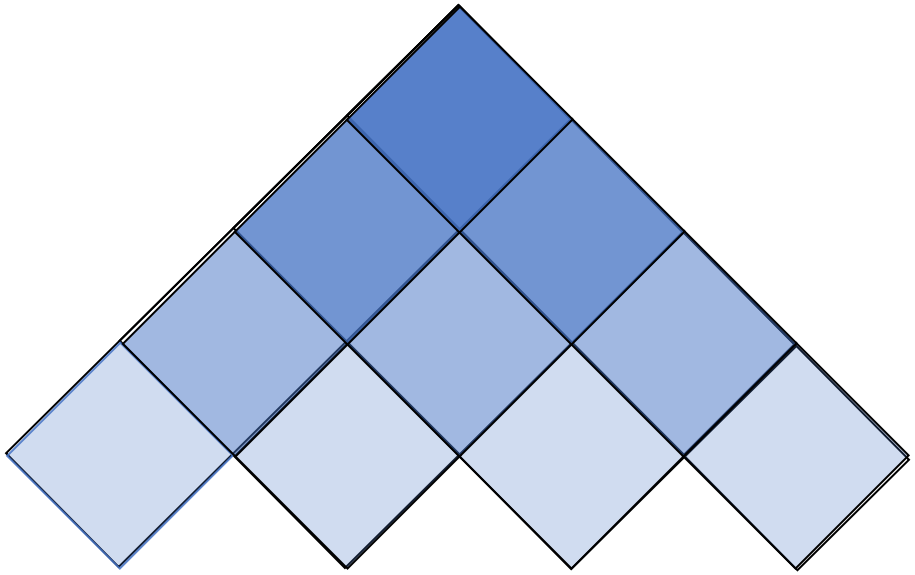
Constituency Parsing



PCFG

Rule	**Prob θ_i**
$S \rightarrow NP\ VP$	θ_0
$NP \rightarrow NP\ NP$	θ_1
…	
$N \rightarrow$ fish	θ_{42}
$N \rightarrow$ people	θ_{43}
$V \rightarrow$ fish	θ_{44}
…	

Cocke-Kasami-Younger (CKY) Constituency Parsing



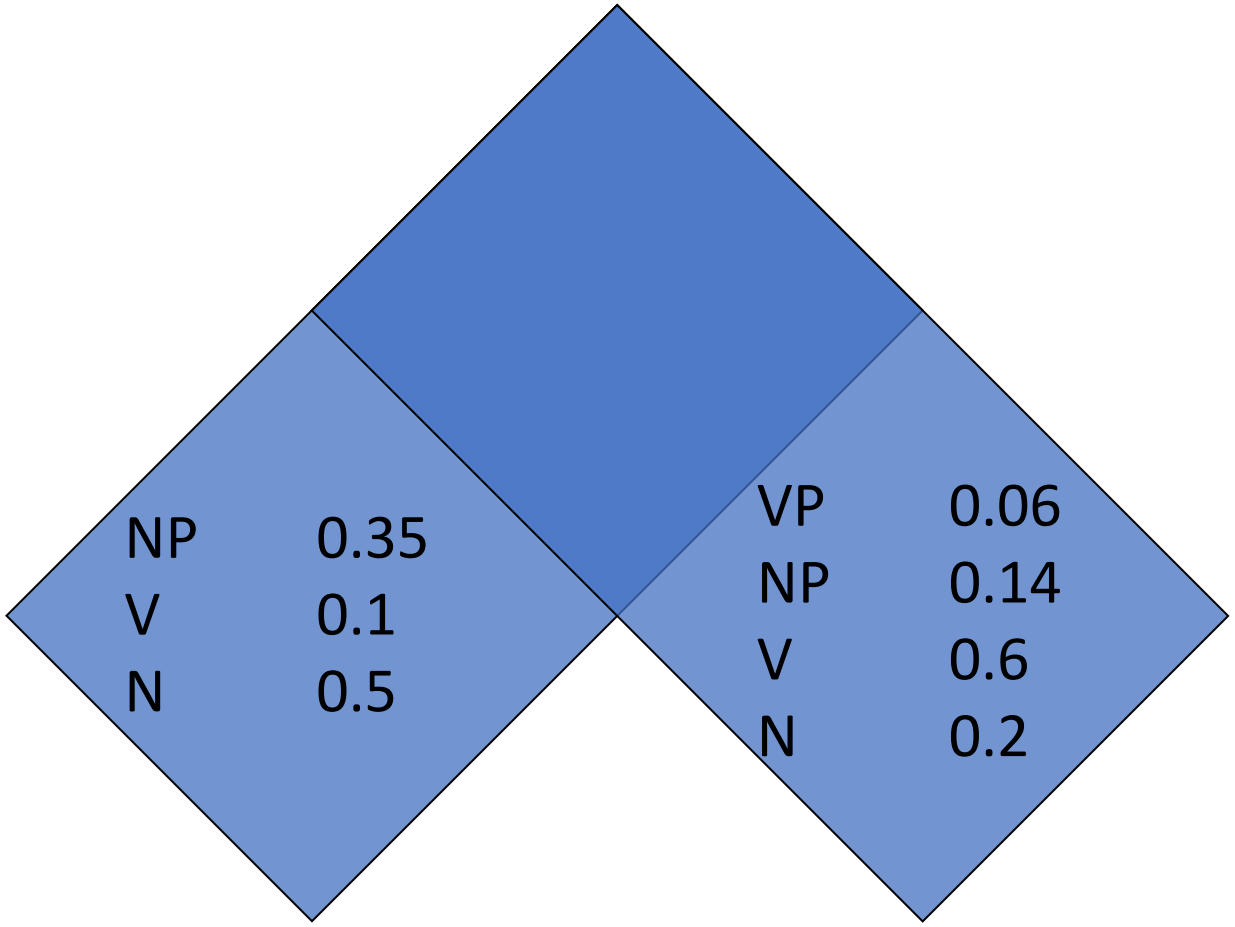
fish people fish tanks

Parse triangle/ chart

Viterbi (Max) Scores

NP -> NP NP = 0.35 * 0.14 * 0.1 = 0.0049
VP -> V NP = 0.1 * 0.14 * 0.5 = 0.007
S -> VP = 0.007 * 0.1 = 0.0007

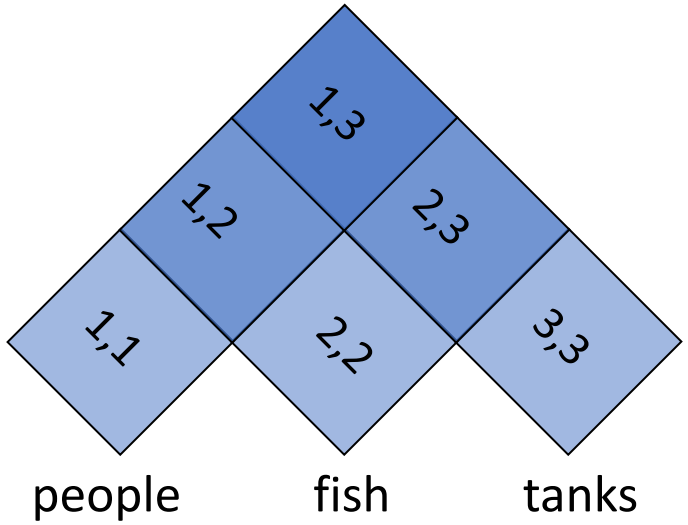
S -> NP VP = 0.35 * 0.06 * 0.9 = 0.0189

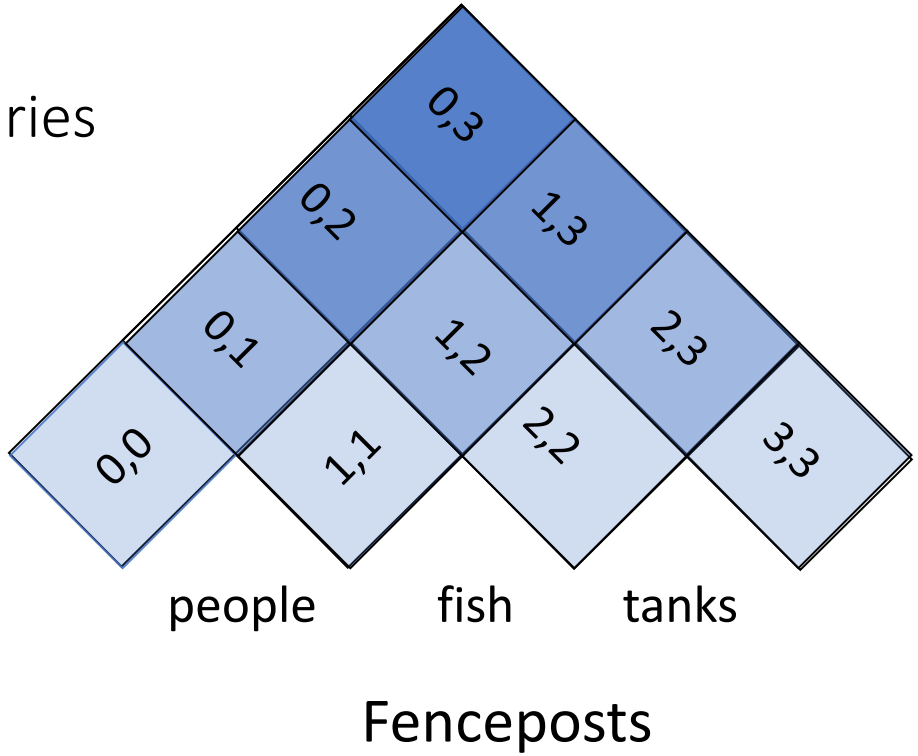


S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

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





Fenceposts

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 (Earley-style dotted rules), but it's always there.

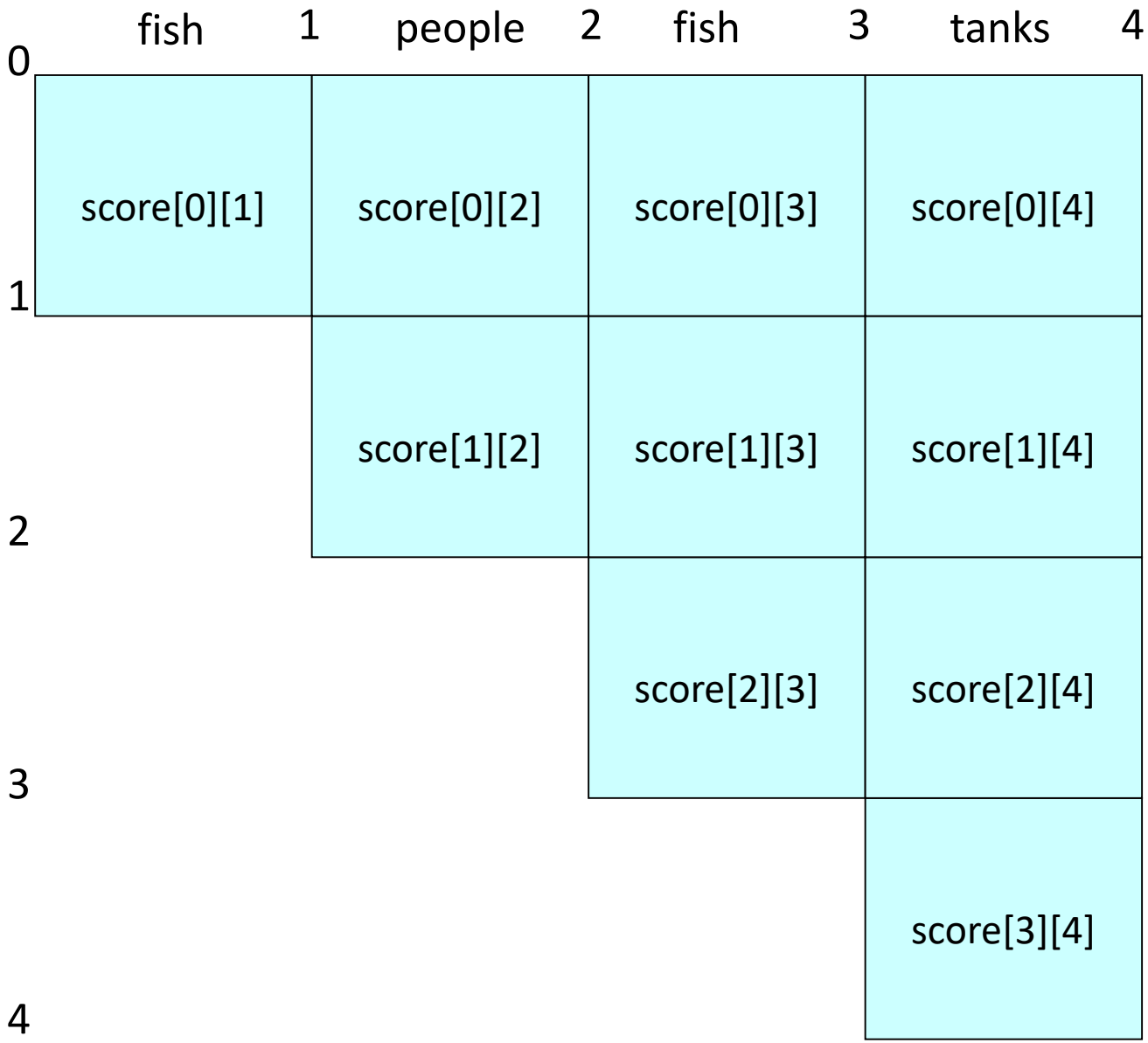
CKY Parsing

A worked example

The grammar:
Binary, no epsilons,

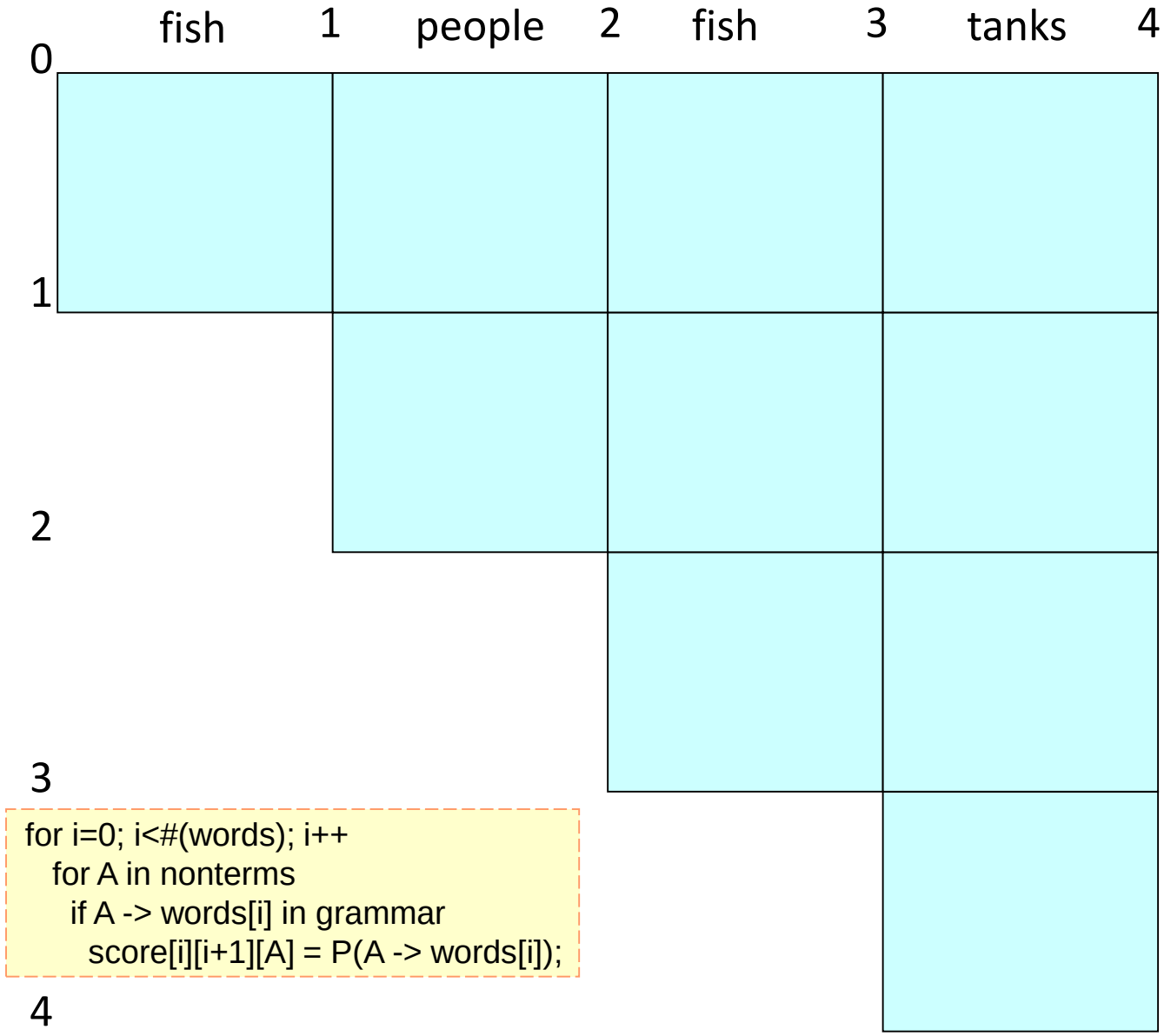
S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

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
2
3
4
score[0][1]
score[0][2]
score[0][3]
score[0][4]
score[1][2]
score[1][3]
score[1][4]
score[2][3]
score[2][4]
score[3][4]

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

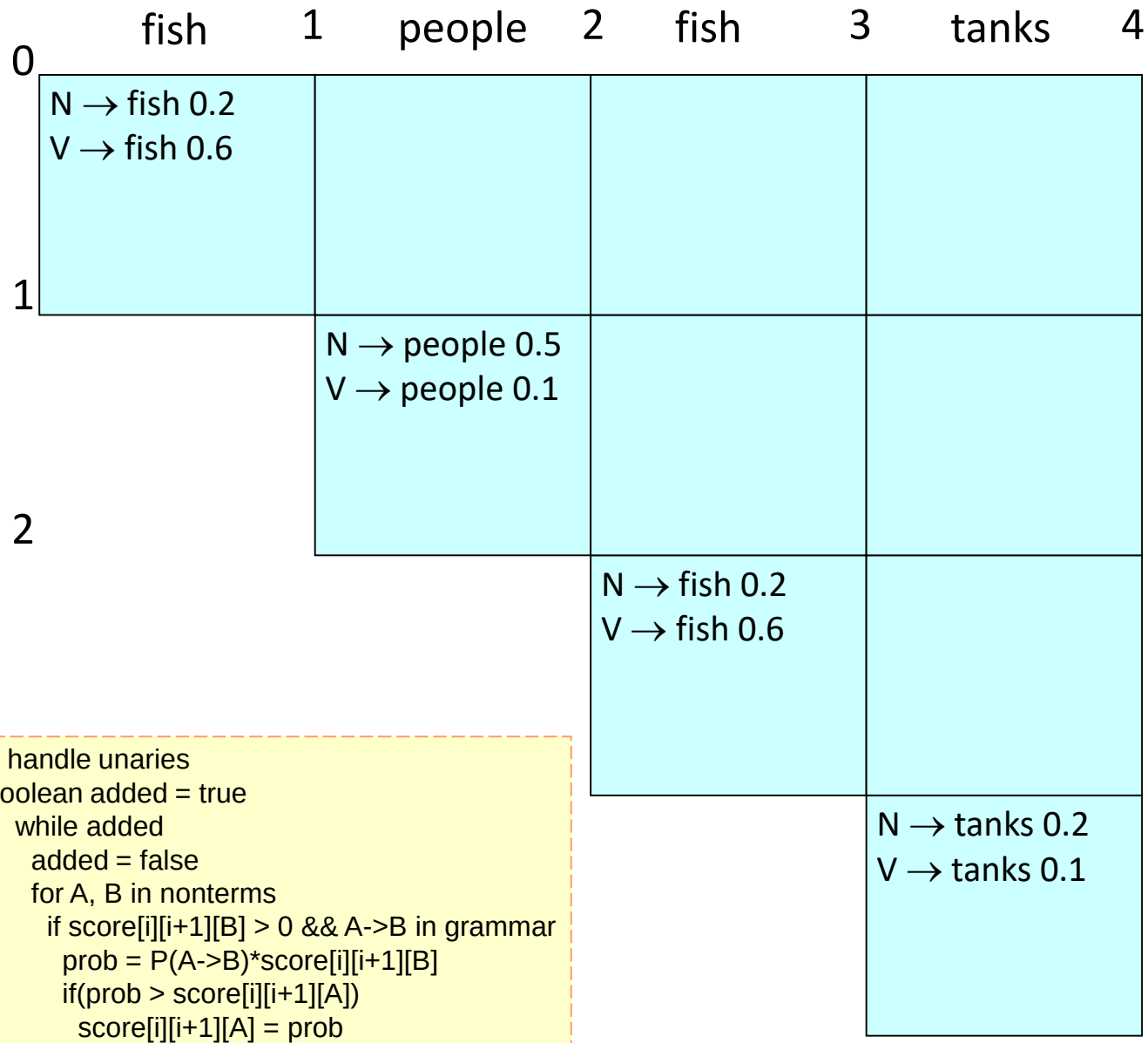
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0




```
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]);
```

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

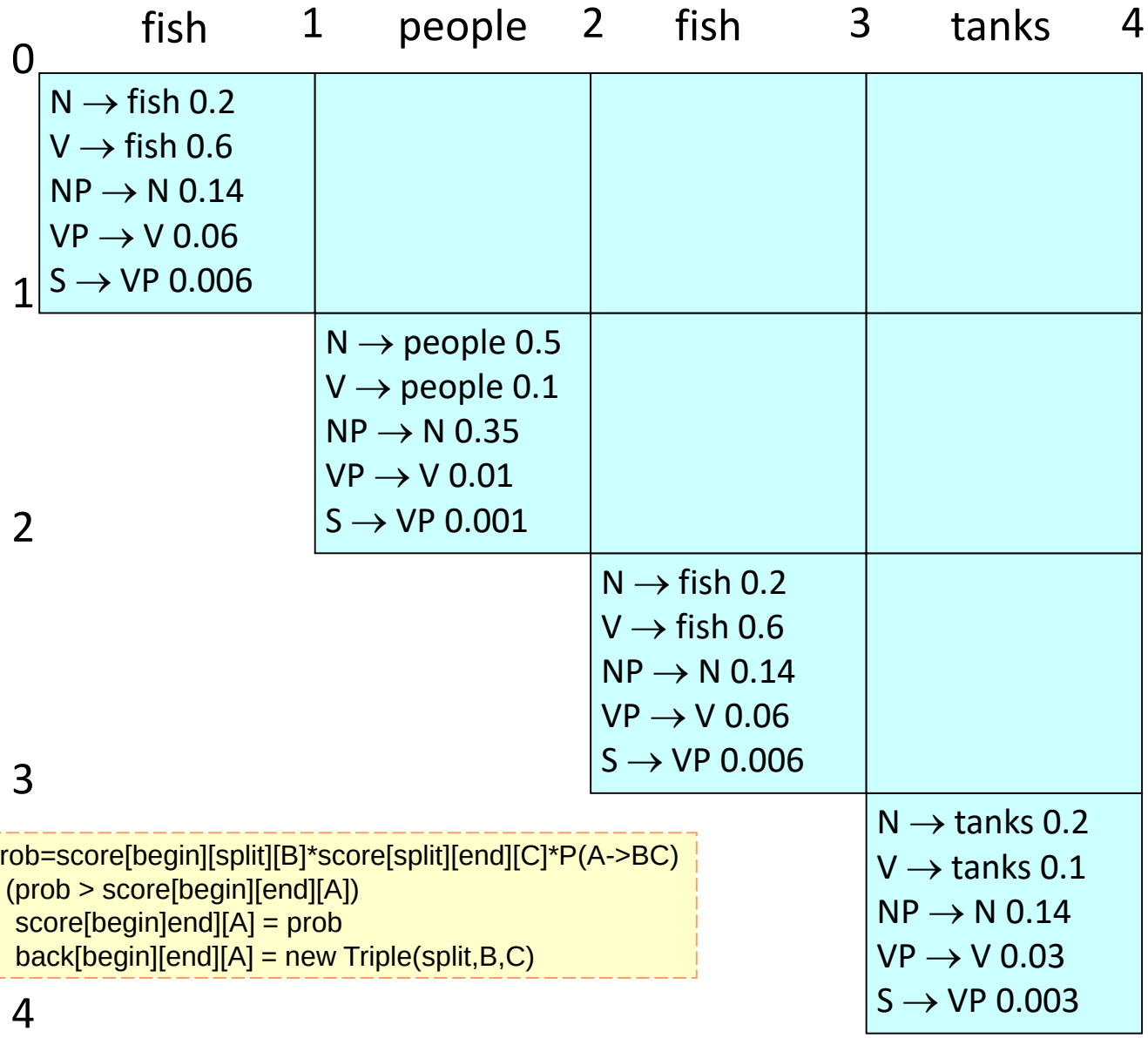
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0




```
// 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



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

	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0	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		
1		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	
2			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
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

```
//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	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		
1		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	
2			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				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
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	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	
1		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	
2			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				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
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	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			
1			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					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							N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003	
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

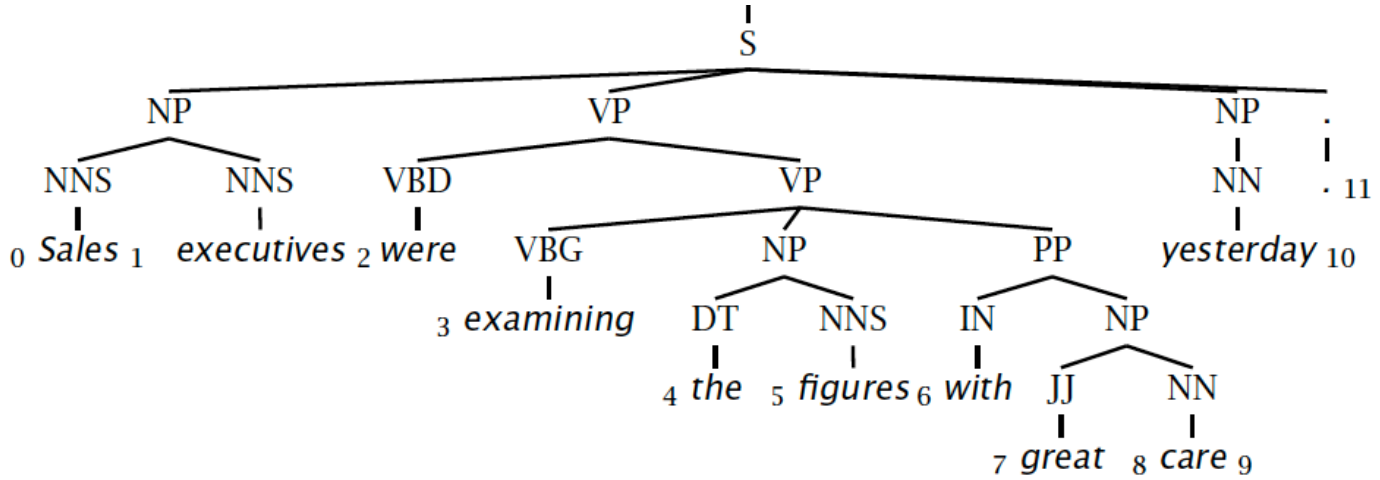
	0 fish	1 people	2 fish	3 tanks	4
1	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	NP → NP NP 0.0000009604 VP → V NP 0.00002058 S → NP VP 0.00018522	
2		N → people 0.5 V → people 0.1 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	NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323	
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	

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

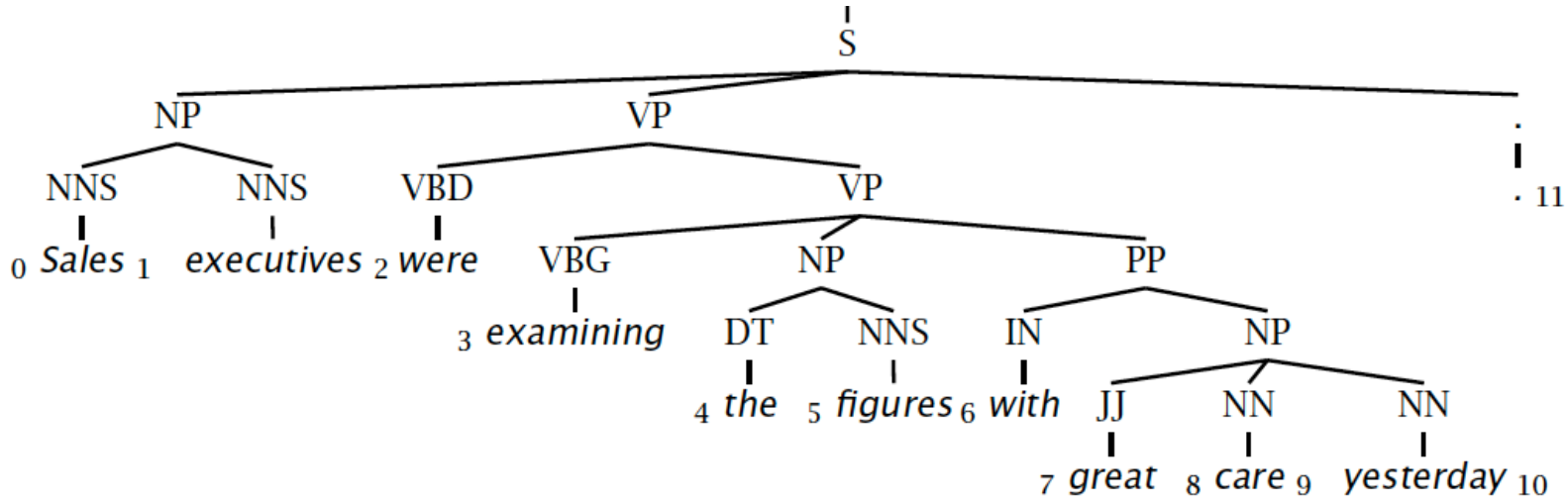
Constituency Parser Evaluation

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%