

EECS 487: Introduction to Natural Language Processing

Instructor: Prof. Lu Wang

Computer Science and Engineering

University of Michigan

<https://web.eecs.umich.edu/~wangluxy/>

Two views of linguistic structure:

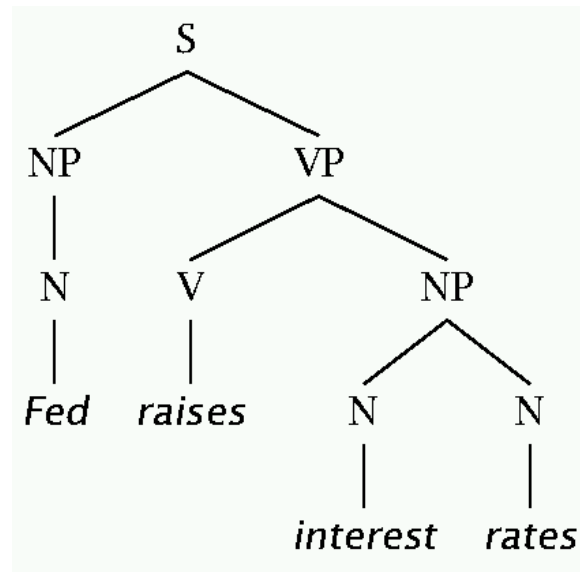
1. Constituency (phrase structure)

- Phrase structure organizes words into nested constituents.
 - Fed raises interest rates

Two views of linguistic structure:

1. Constituency (phrase structure)

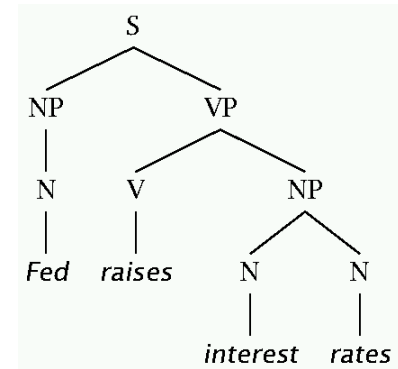
- Phrase structure organizes words into nested constituents.

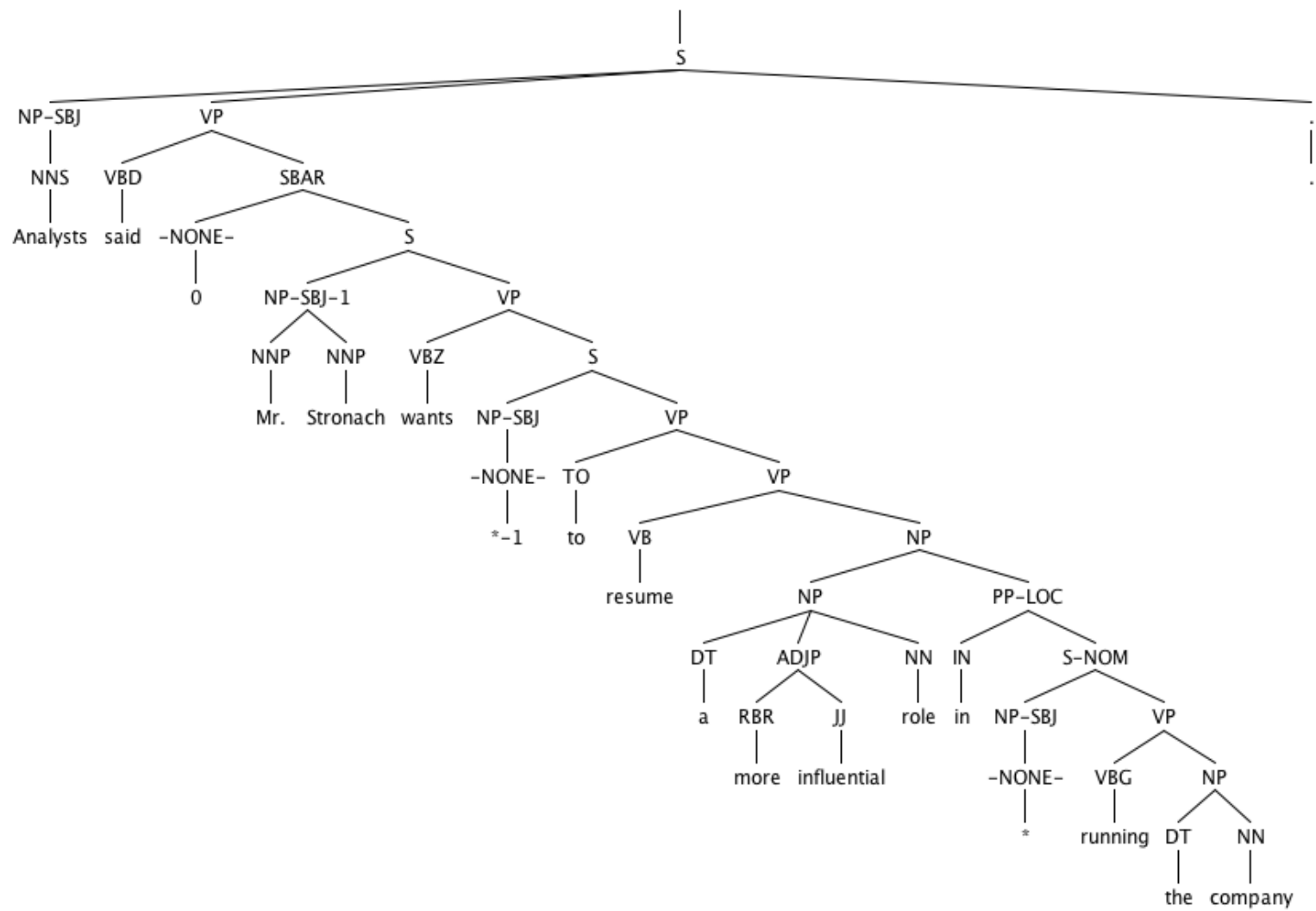


Two views of linguistic structure:

1. Constituency (phrase structure)

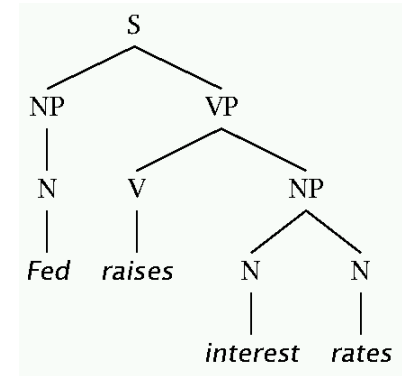
- Phrase structure organizes words into nested constituents.
- How do we know what is a **constituent**? (Not that linguists don't argue about some cases.)
 - Distribution: a constituent behaves as a unit that can appear in different places:
 - John talked [to the children] [about drugs].
 - John talked [about drugs] [to the children].
 - *John talked drugs to the children about
 - Substitution/expansion/pronoun:
 - I sat [on the box/right on top of the box/there].





Headed phrase structure

- Context-free grammar
- $VP \rightarrow \dots VB^* \dots$
- $NP \rightarrow \dots NN^* \dots$
- $ADJP \rightarrow \dots JJ^* \dots$
- $ADVP \rightarrow \dots RB^* \dots$
- $S \rightarrow \dots NP VP \dots$
- Plus minor phrase types:
 - QP (quantifier phrase in NP: *some people*), CONJP (multi word constructions: *as well as*), INTJ (interjections: *aha*), etc.



Two views of linguistic structure:

2. Dependency structure

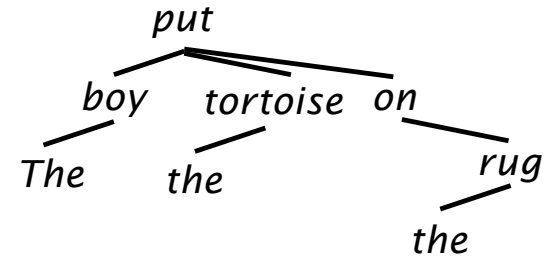
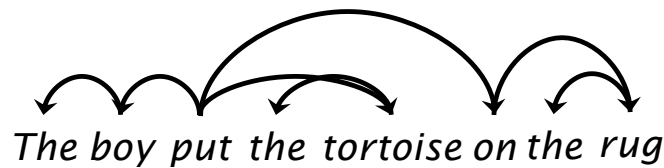
- Dependency structure shows which words depend on (modify or are arguments of) which other words.

The boy put the tortoise on the rug

Two views of linguistic structure:

2. Dependency structure

- Dependency structure shows which words depend on (modify or are arguments of) which other words.



Outline

- ➡ • Phrase Chunking
 - (Probabilistic) Context-Free Grammars
 - Chomsky Normal Form
 - CKY Parsing

Phrase Chunking

- Find all non-recursive noun phrases (NPs) and verb phrases (VPs) in a sentence.
 - [NP I] [VP ate] [NP the spaghetti] [PP with] [NP meatballs].
 - [NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only 1.8 billion] [PP in] [NP September] .

Phrase Chunking as Sequence Labeling

- Tag individual words with one of 3 tags
 - B (Begin) word starts new target phrase
 - I (Inside) word is part of target phrase but not the first word
 - O (Other) word is not part of target phrase
- Sample for NP chunking
 - He reckons the current account deficit will narrow to only 1.8 billion in September.

Begin

Inside

Other

Evaluating Chunking

Per token accuracy does not evaluate finding correct full chunks.
Instead use:

$$\text{Precision} = \frac{\text{Number of correct chunks found}}{\text{Total number of chunks found}}$$

$$\text{Recall} = \frac{\text{Number of correct chunks found}}{\text{Total number of actual chunks}}$$

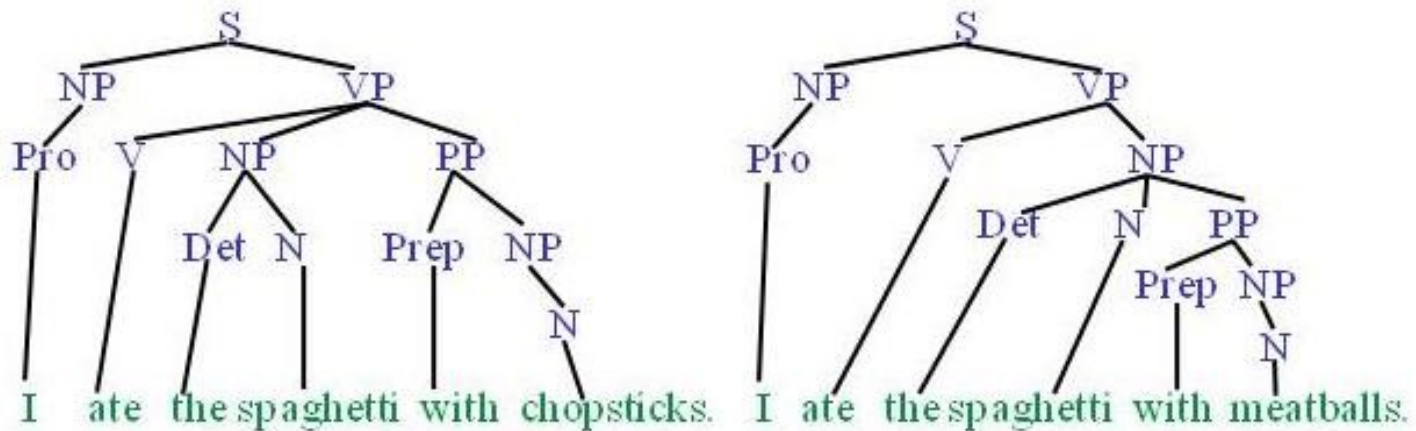
$$\text{F measure: } F_1 = \frac{1}{(\frac{1}{P} + \frac{1}{R})/2} = \frac{2PR}{P + R}$$

Outline

- Phrase Chunking
- ➔ • (Probabilistic) Context-Free Grammars
- Chomsky Normal Form
- CKY Parsing

Syntactic Parsing

- Produce the correct syntactic parse tree for a sentence.



Annotated data: The Penn Treebank

[Marcus et al. 1993, *Computational Linguistics*]

```
( (S
  (NP-SBJ (DT The) (NN move))
  (VP (VBD followed)
    (NP
      (NP (DT a) (NN round))
      (PP (IN of)
        (NP
          (NP (JJ similar) (NNS increases))
          (PP (IN by)
            (NP (JJ other) (NNS lenders)))
          (PP (IN against)
            (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans))))))
    (, ,)
    (S-ADV
      (NP-SBJ (-NONE- *))
      (VP (VBG reflecting)
        (NP
          (NP (DT a) (VBG continuing) (NN decline))
          (PP-LOC (IN in)
            (NP (DT that) (NN market))))))
      (. .)))
```

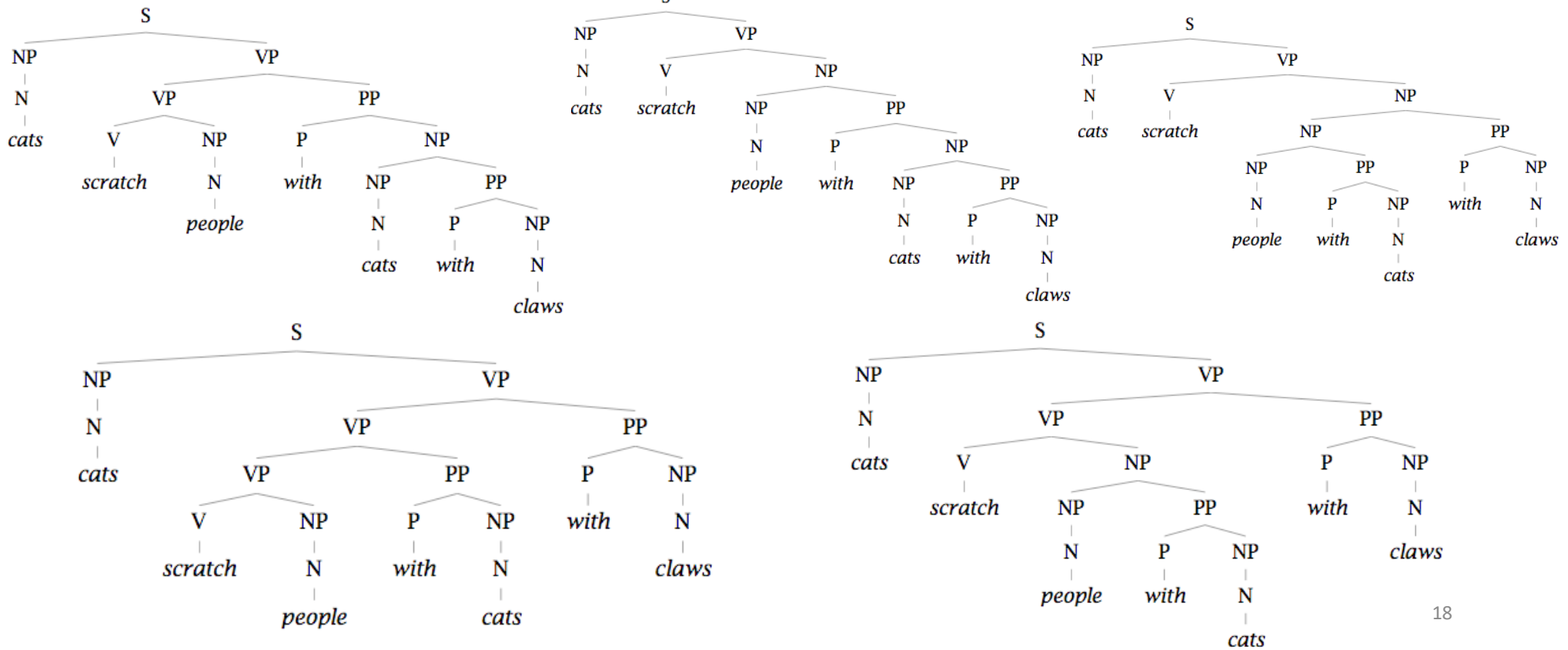
The rise of annotated data

- Starting off, building a treebank seems a lot slower and less useful than building a grammar
- But a treebank gives us many things
 - Reusability of the labor
 - Many parsers, POS taggers, etc.
 - Valuable resource for linguistics
 - Broad coverage
 - Frequencies and distributional information
 - A way to evaluate systems

Two problems to solve for parsing:

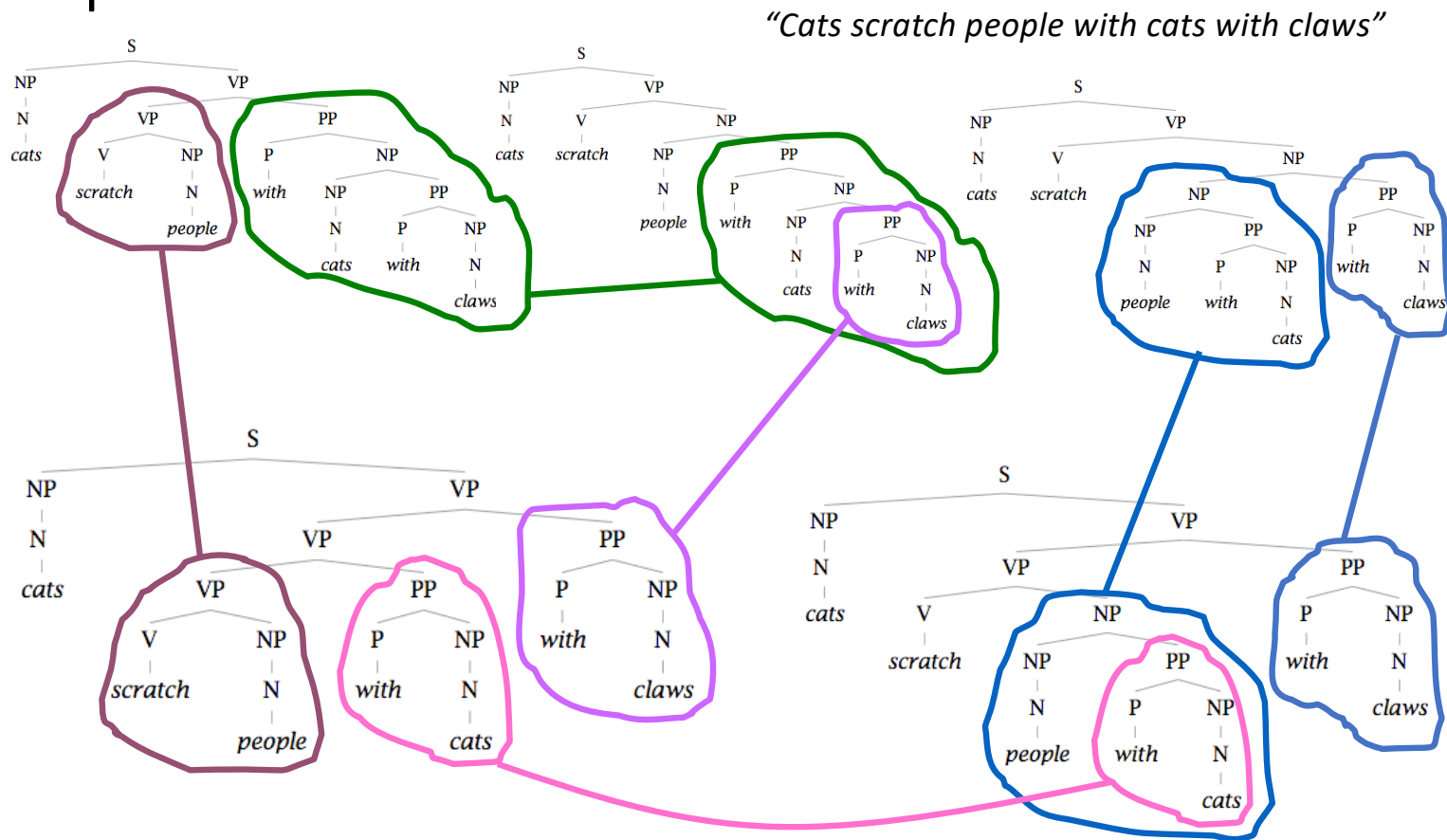
1. Repeated work...

"Cats scratch people with cats with claws"



Two problems to solve for parsing:

1. Repeated work...



Two problems to solve for parsing:

2. Choosing the correct parse

- How do we work out the correct attachment:
 - She saw the man with a telescope
- Words are good predictors of attachment, even absent full understanding
 - Moscow **sent** more than 100,000 soldiers **into** Afghanistan ...
 - Sydney Water breached an **agreement with** NSW Health ...
- Our statistical parsers will try to exploit such statistics.

Statistical parsing applications

Statistical parsers are now robust and widely used in larger NLP applications:

- High precision question answering [Pasca and Harabagiu SIGIR 2001]
- Improving biological named entity finding [Finkel et al. JNLPBA 2004]
- Syntactically based sentence compression [Lin and Wilbur 2007]
- Extracting opinions about products [Bloom et al. NAACL 2007]
- Improved interaction in computer games [Gorniak and Roy 2005]
- Helping linguists find data [Resnik et al. BLS 2005]
- Source sentence analysis for machine translation [Xu et al. 2009]
- Relation extraction systems [Fundel et al. Bioinformatics 2006]

(Probabilistic) Context-Free Grammars

- CFG
- PCFG

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 phrase structure grammar

$S \rightarrow NP VP$

$VP \rightarrow V NP$

$VP \rightarrow V NP PP$

$NP \rightarrow NP NP$

$NP \rightarrow NP PP$

$NP \rightarrow N$

$NP \rightarrow e$

$PP \rightarrow P NP$

$N \rightarrow \textit{people}$

$N \rightarrow \textit{fish}$

$N \rightarrow \textit{tanks}$

$N \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

people fish tanks

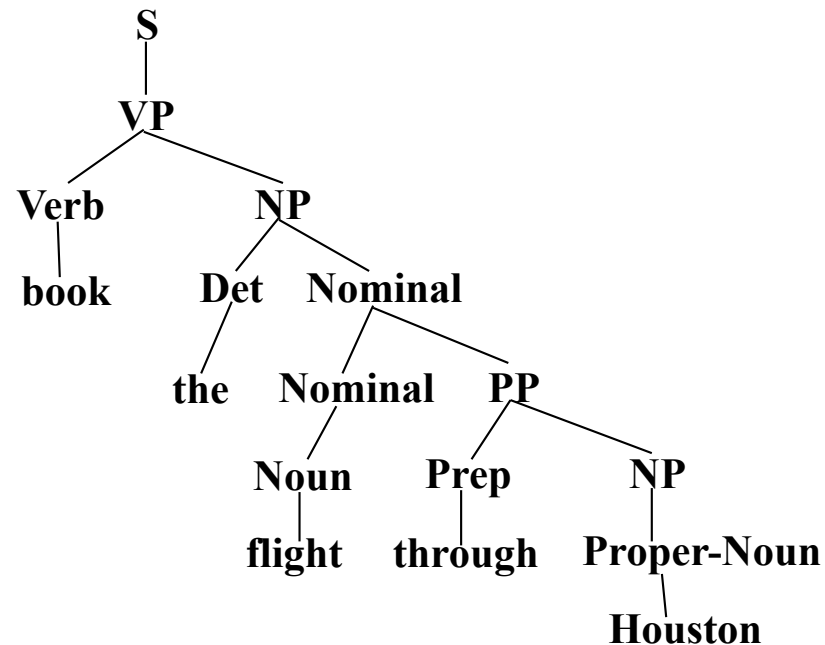
people fish with rods

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 .

Sentence Generation

- Sentences are generated by recursively rewriting the start symbol using the productions until only terminal symbols remain.



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

$S \rightarrow NP VP$

$VP \rightarrow V NP$

$VP \rightarrow V NP PP$

$NP \rightarrow NP NP$

$NP \rightarrow NP PP$

$NP \rightarrow N$

$NP \rightarrow e$

$PP \rightarrow P NP$

$N \rightarrow \textit{people}$

$N \rightarrow \textit{fish}$

$N \rightarrow \textit{tanks}$

$N \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

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 .

A PCFG

$S \rightarrow NP VP$ 1.0

$VP \rightarrow V NP$ 0.6

$VP \rightarrow V NP PP$ 0.4

$NP \rightarrow NP NP$ 0.1

$NP \rightarrow NP PP$ 0.2

$NP \rightarrow N$ 0.7

$PP \rightarrow P NP$ 1.0

$N \rightarrow \textit{people}$ 0.5

$N \rightarrow \textit{fish}$ 0.2

$N \rightarrow \textit{tanks}$ 0.2

$N \rightarrow \textit{rods}$ 0.1

$V \rightarrow \textit{people}$ 0.1

$V \rightarrow \textit{fish}$ 0.6

$V \rightarrow \textit{tanks}$ 0.3

$P \rightarrow \textit{with}$ 1.0

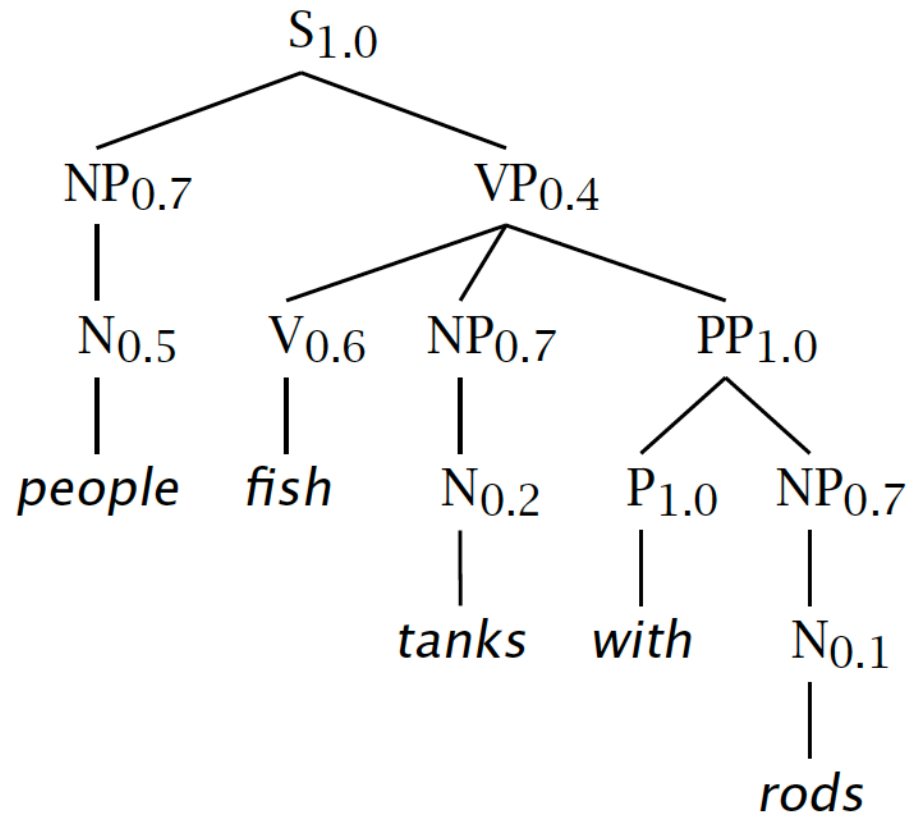
[With empty NP removed so
less ambiguous]

The probability of trees and strings

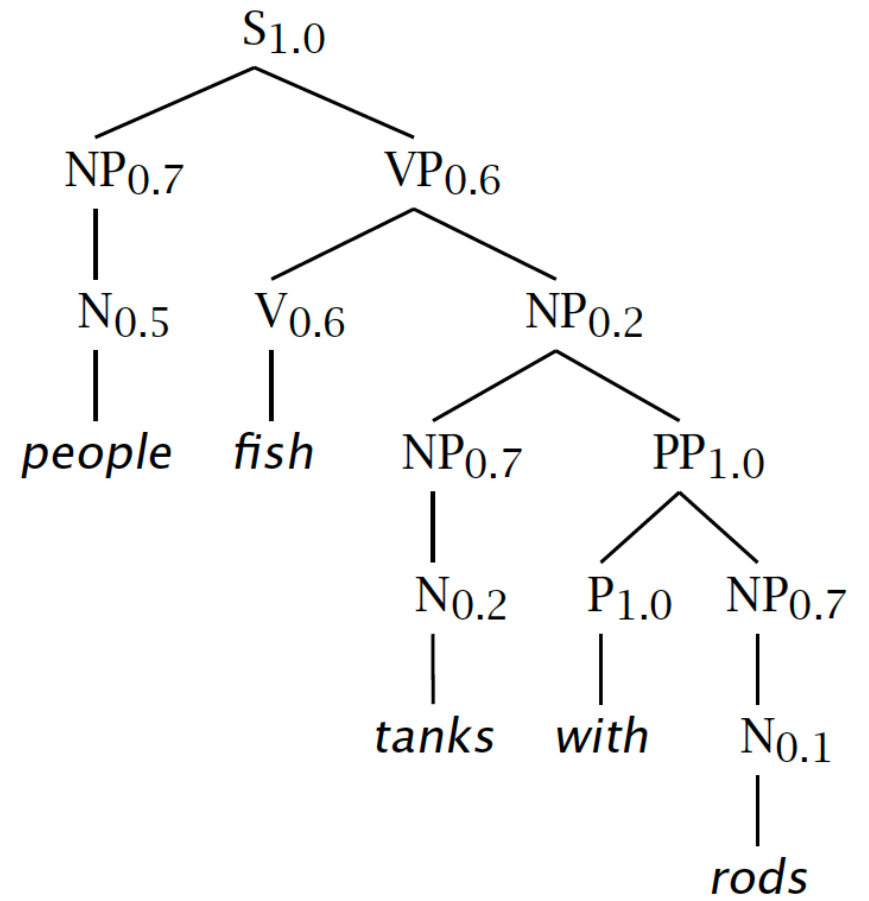
- $P(t)$ – The probability of a tree t is the product of the probabilities of the rules used to generate it.
- $P(s)$ – The probability of the string s is the sum of the probabilities of the trees which have that string as their yield

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

t_1 :



t_2 :



Tree and String Probabilities

• $s = \text{people fish tanks with rods}$

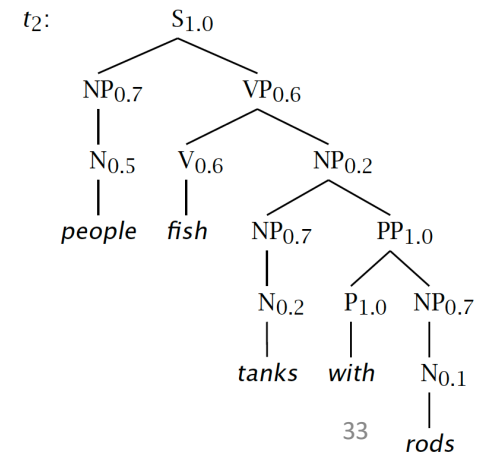
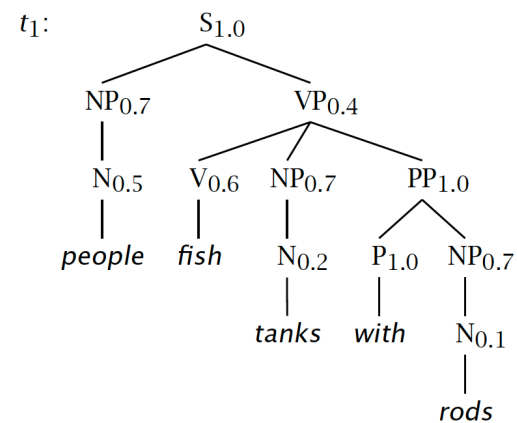
• $P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 0.7$
 $\times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$
 $= 0.0008232$

Verb attach

• $P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.2$
 $\times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$
 $= 0.00024696$

Noun attach

• $P(s) = P(t_1) + P(t_2)$
 $= 0.0008232 + 0.00024696$
 $= 0.00107016$



Outline

- Phrase Chunking
- (Probabilistic) Context-Free Grammars
- ➔ • Chomsky Normal Form
- CKY Parsing

Chomsky Normal Form

- All rules are of the form $X \rightarrow YZ$ or $X \rightarrow w$
 - $X, Y, Z \in N$ and $w \in T$
- A transformation to this form doesn't change the 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

$S \rightarrow NP VP$

$VP \rightarrow V NP$

$VP \rightarrow V NP PP$

$NP \rightarrow NP NP$

$NP \rightarrow NP PP$

$NP \rightarrow N$

$NP \rightarrow e$

$PP \rightarrow P NP$

$N \rightarrow \textit{people}$

$N \rightarrow \textit{fish}$

$N \rightarrow \textit{tanks}$

$N \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

Chomsky Normal Form steps

$S \rightarrow NP VP$

$S \rightarrow VP$

$VP \rightarrow V NP$

$VP \rightarrow V$

$VP \rightarrow V NP PP$

$VP \rightarrow V PP$

$NP \rightarrow NP NP$

$NP \rightarrow NP$

$NP \rightarrow NP PP$

$NP \rightarrow PP$

$NP \rightarrow N$

$PP \rightarrow P NP$

$PP \rightarrow P$

$N \rightarrow \textit{people}$

$N \rightarrow \textit{fish}$

$N \rightarrow \textit{tanks}$

$N \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

Chomsky Normal Form steps

$S \rightarrow NP VP$

$VP \rightarrow V NP$

$S \rightarrow V NP$

$VP \rightarrow V$

$S \rightarrow V$

$VP \rightarrow V NP PP$

$S \rightarrow V NP PP$

$VP \rightarrow V PP$

$S \rightarrow V PP$

$NP \rightarrow NP NP$

$NP \rightarrow NP$

$NP \rightarrow NP PP$

$NP \rightarrow PP$

$NP \rightarrow N$

$PP \rightarrow P NP$

$PP \rightarrow P$

$N \rightarrow \textit{people}$

$N \rightarrow \textit{fish}$

$N \rightarrow \textit{tanks}$

$N \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

Chomsky Normal Form steps

$S \rightarrow NP VP$

$VP \rightarrow V NP$

$S \rightarrow V NP$

$VP \rightarrow V$

$VP \rightarrow V NP PP$

$S \rightarrow V NP PP$

$VP \rightarrow V PP$

$S \rightarrow V PP$

$NP \rightarrow NP NP$

$NP \rightarrow NP$

$NP \rightarrow NP PP$

$NP \rightarrow PP$

$NP \rightarrow N$

$PP \rightarrow P NP$

$PP \rightarrow P$

$N \rightarrow \textit{people}$

$N \rightarrow \textit{fish}$

$N \rightarrow \textit{tanks}$

$N \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$S \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$S \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$S \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

Chomsky Normal Form steps

$S \rightarrow NP VP$

$VP \rightarrow V NP$

$S \rightarrow V NP$

$VP \rightarrow V NP PP$

$S \rightarrow V NP PP$

$VP \rightarrow V PP$

$S \rightarrow V PP$

$NP \rightarrow NP NP$

$NP \rightarrow NP$

$NP \rightarrow NP PP$

$NP \rightarrow PP$

$NP \rightarrow N$

$PP \rightarrow P NP$

$PP \rightarrow P$

$N \rightarrow \textit{people}$

$N \rightarrow \textit{fish}$

$N \rightarrow \textit{tanks}$

$N \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$S \rightarrow \textit{people}$

$VP \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$S \rightarrow \textit{fish}$

$VP \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$S \rightarrow \textit{tanks}$

$VP \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

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

$S \rightarrow NP VP$

$VP \rightarrow V NP$

$S \rightarrow V NP$

$VP \rightarrow V @VP_V$

$@VP_V \rightarrow NP PP$

$S \rightarrow V @S_V$

$@S_V \rightarrow NP PP$

$VP \rightarrow V PP$

$S \rightarrow V PP$

$NP \rightarrow NP NP$

$NP \rightarrow NP PP$

$NP \rightarrow P NP$

$PP \rightarrow P NP$

$NP \rightarrow \textit{people}$

$NP \rightarrow \textit{fish}$

$NP \rightarrow \textit{tanks}$

$NP \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$S \rightarrow \textit{people}$

$VP \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$S \rightarrow \textit{fish}$

$VP \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$S \rightarrow \textit{tanks}$

$VP \rightarrow \textit{tanks}$

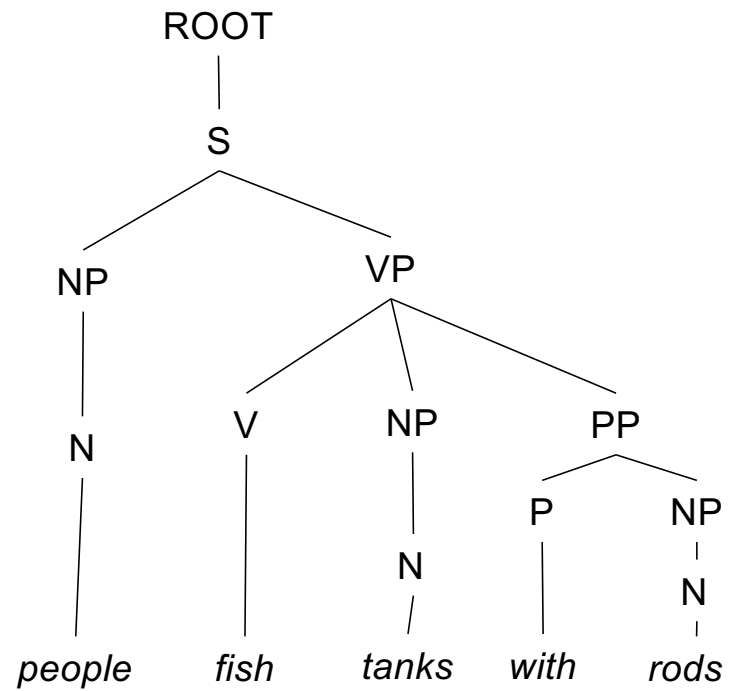
$P \rightarrow \textit{with}$

$PP \rightarrow \textit{with}$

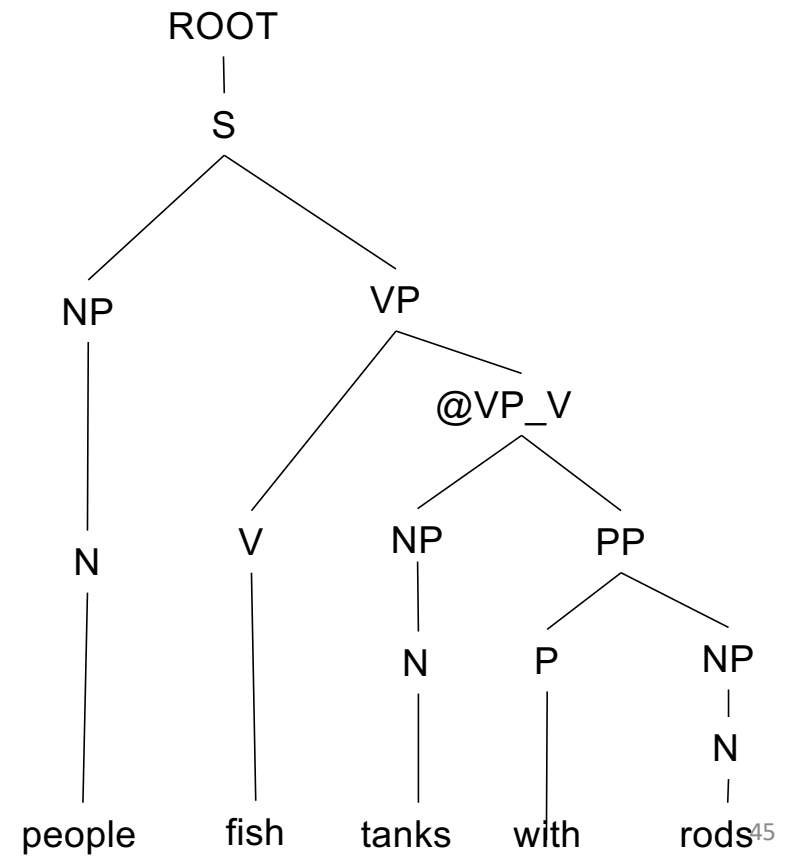
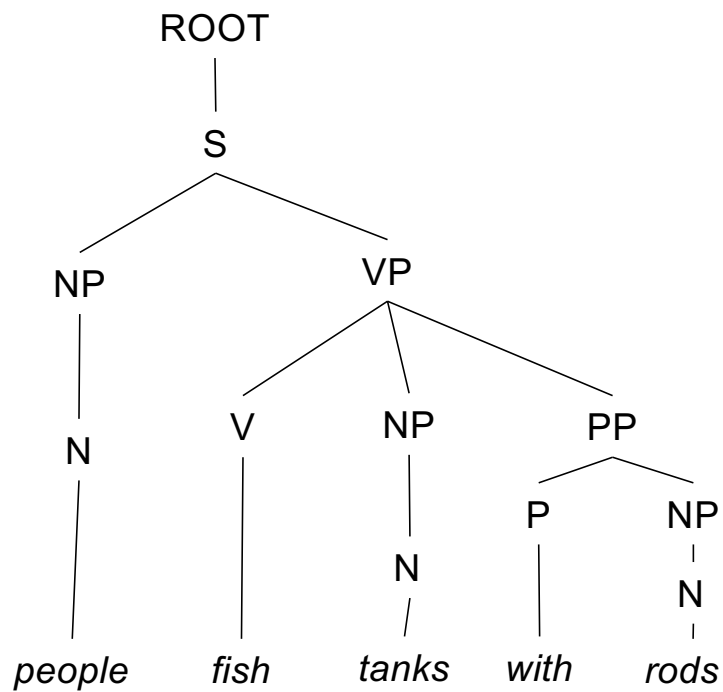
Chomsky Normal Form

- You should think of this as a transformation for efficient parsing
- **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...



Before and After binarization on VP



Outline

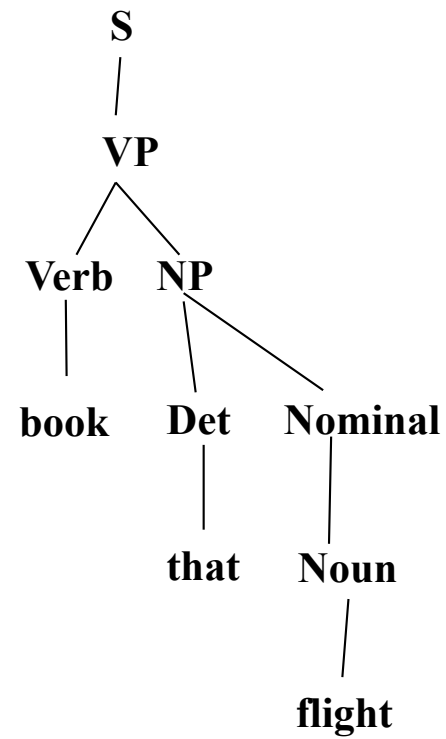
- Phrase Chunking
- (Probabilistic) Context-Free Grammars
- Chomsky Normal Form
-  • CKY Parsing

Parsing

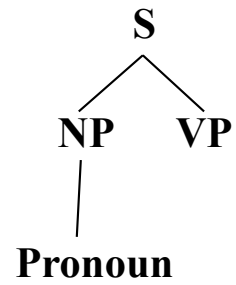
- Given a string of terminals (e.g. sentences) and a CFG, determine if the string can be generated by the CFG.
 - Also return a parse tree for the string
 - Also return all possible parse trees for the string
- Must search space of derivations for one that derives the given string.
 - **Top-Down Parsing**: Start searching space of derivations for the start symbol.
 - **Bottom-up Parsing**: Start search space of reverse derivations from the terminal symbols in the string.

Parsing Example

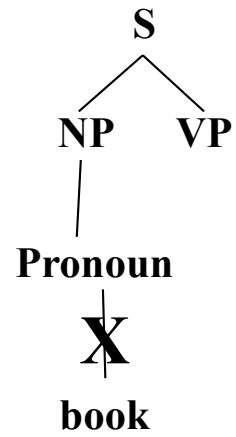
book that flight



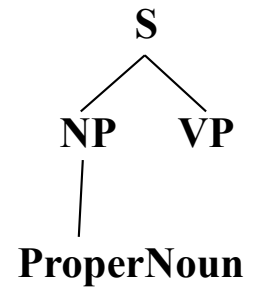
Top Down Parsing



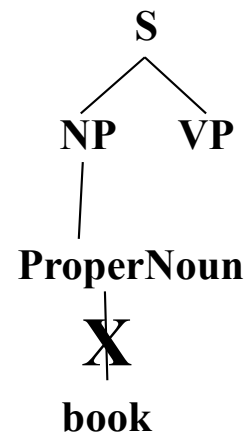
Top Down Parsing



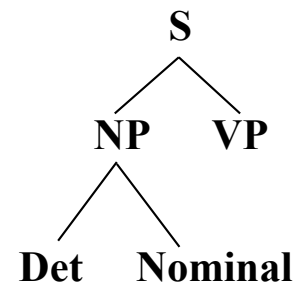
Top Down Parsing



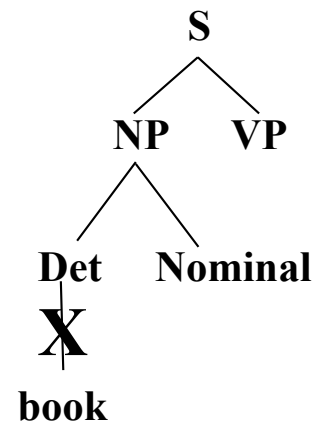
Top Down Parsing



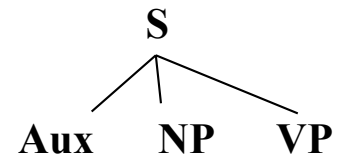
Top Down Parsing



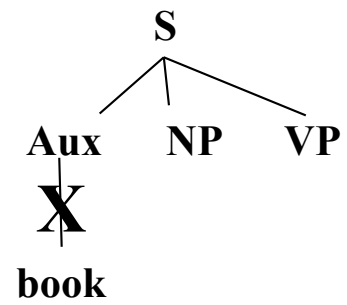
Top Down Parsing



Top Down Parsing



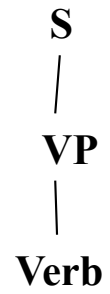
Top Down Parsing



Top Down Parsing

S
|
VP

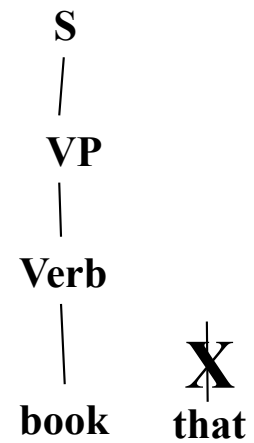
Top Down Parsing



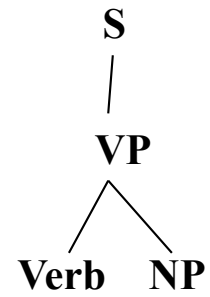
Top Down Parsing



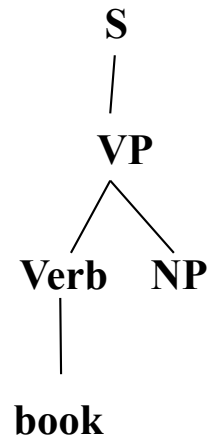
Top Down Parsing



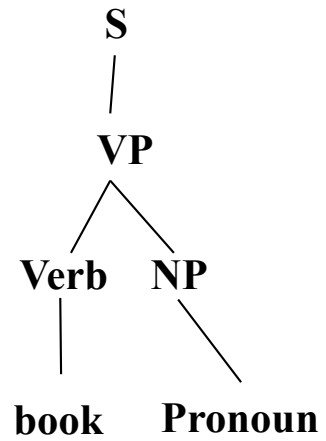
Top Down Parsing



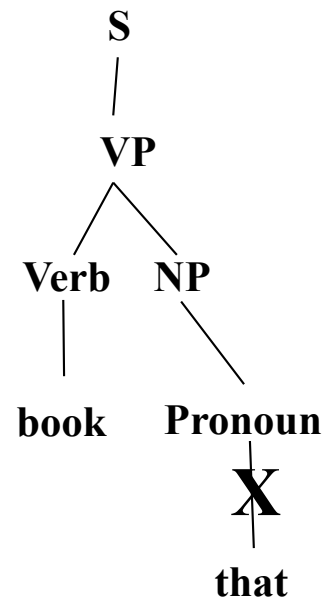
Top Down Parsing



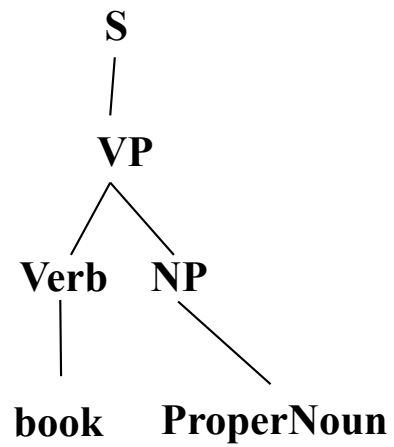
Top Down Parsing



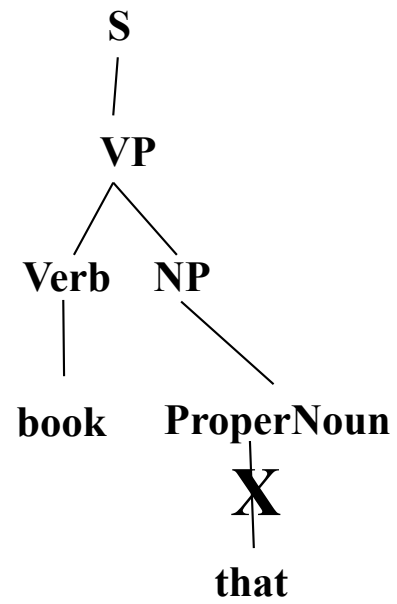
Top Down Parsing



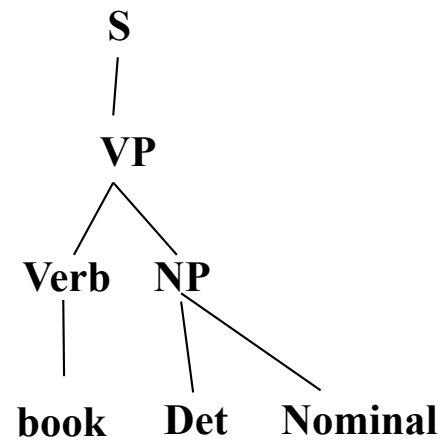
Top Down Parsing



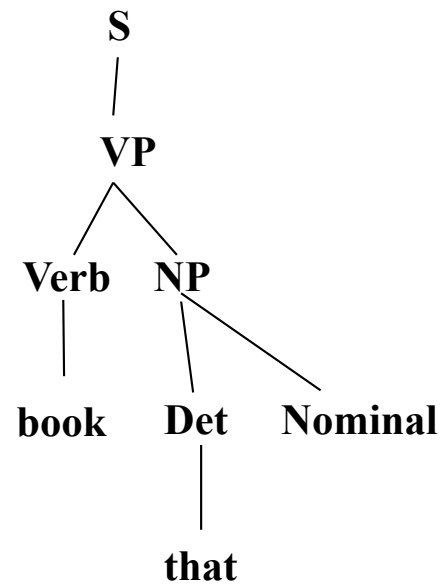
Top Down Parsing



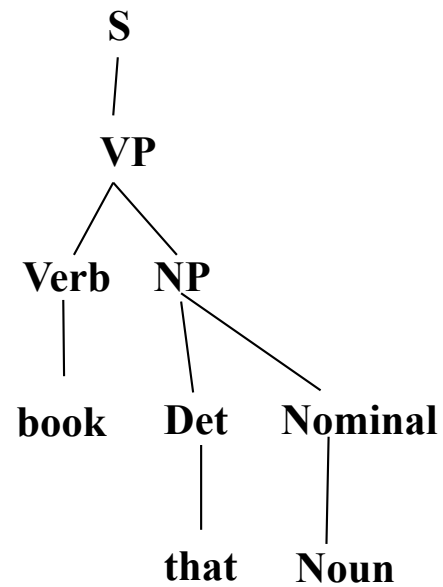
Top Down Parsing



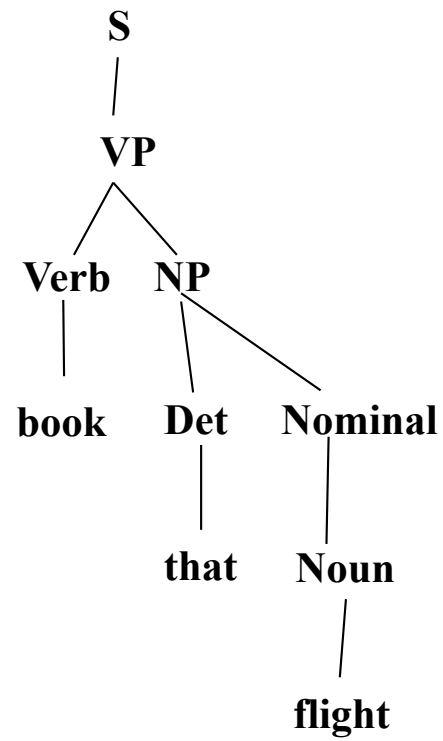
Top Down Parsing



Top Down Parsing



Top Down Parsing



Bottom Up Parsing

book that flight

Bottom Up Parsing

Noun
|
book **that** **flight**

Bottom Up Parsing

Nominal



Noun

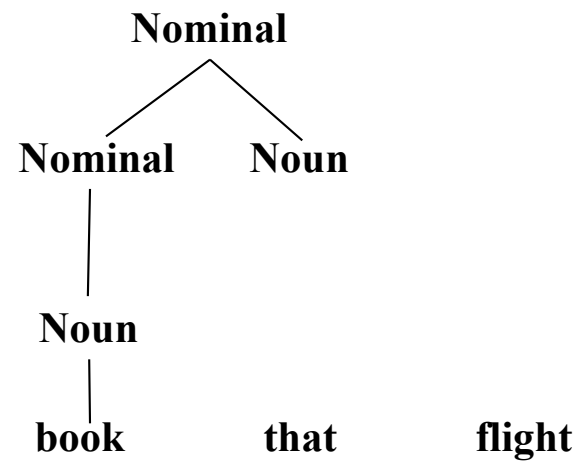


book

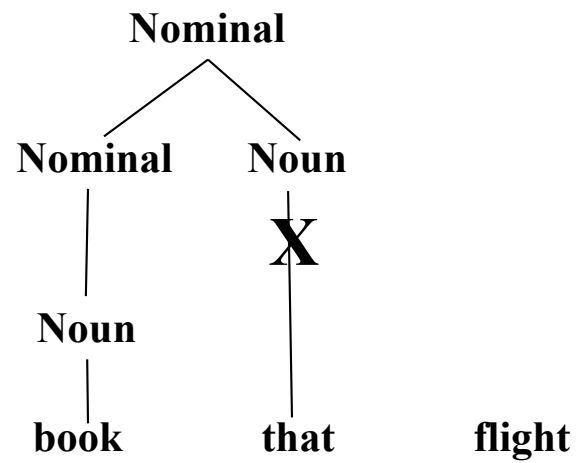
that

flight

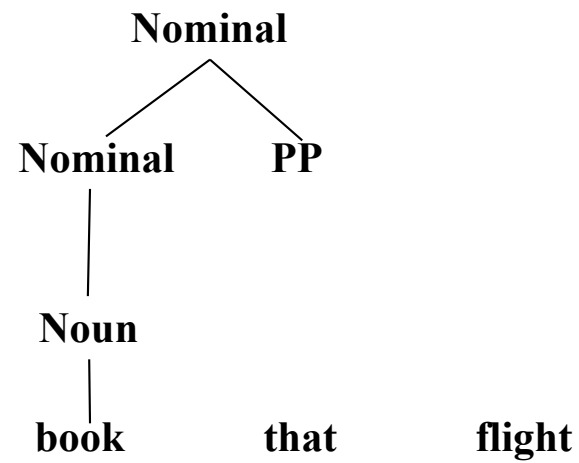
Bottom Up Parsing



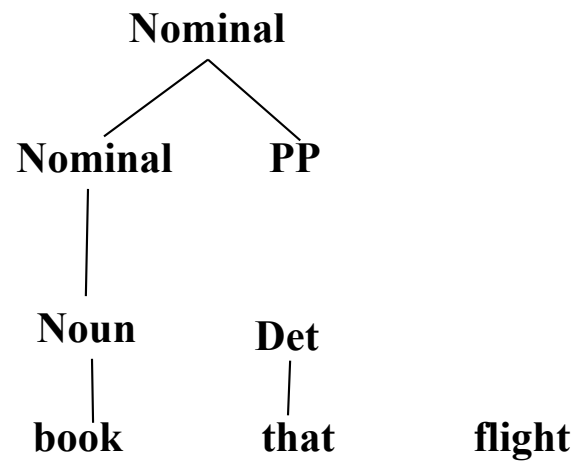
Bottom Up Parsing



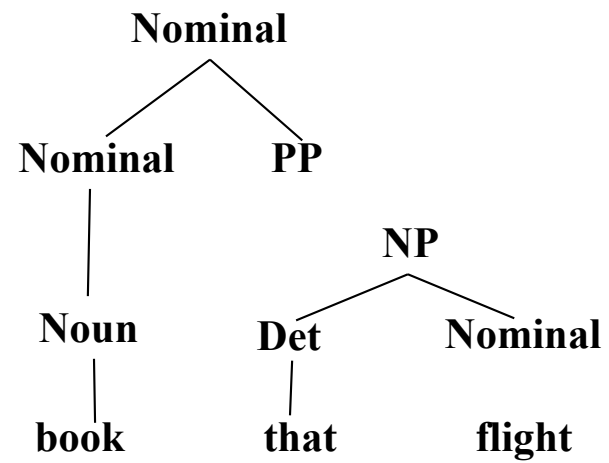
Bottom Up Parsing



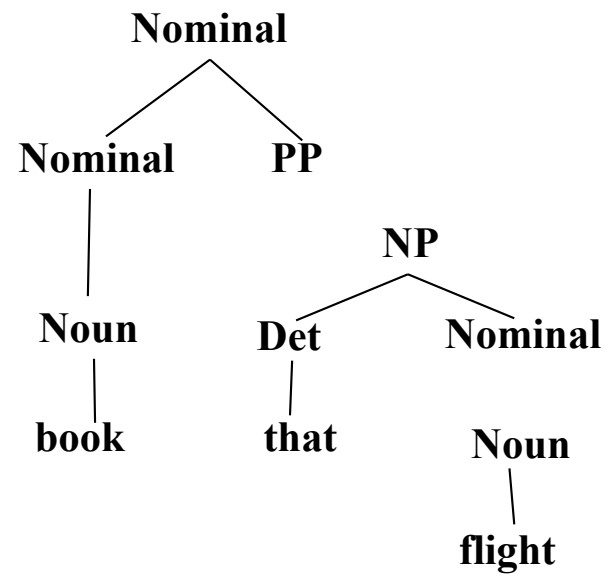
Bottom Up Parsing



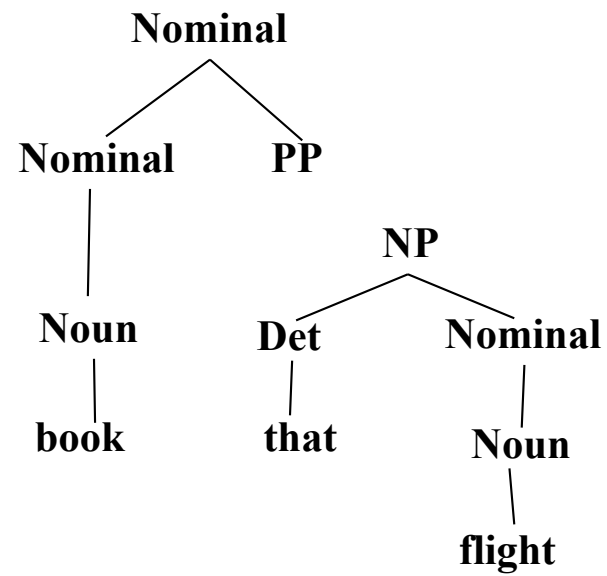
Bottom Up Parsing



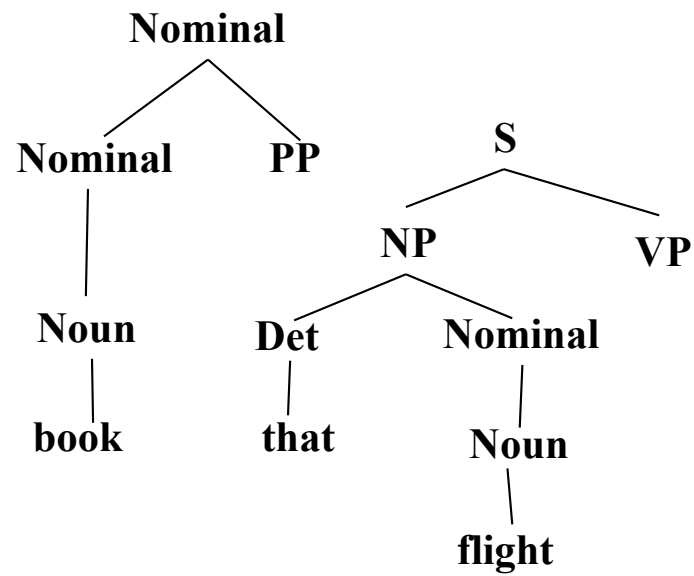
Bottom Up Parsing



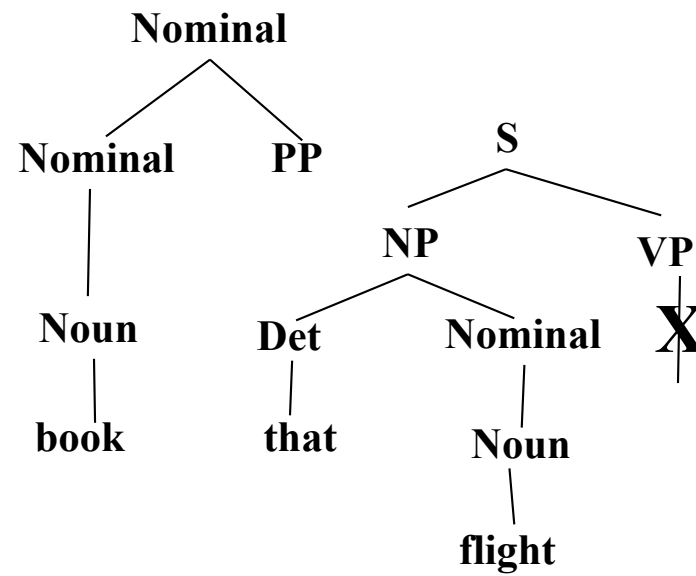
Bottom Up Parsing



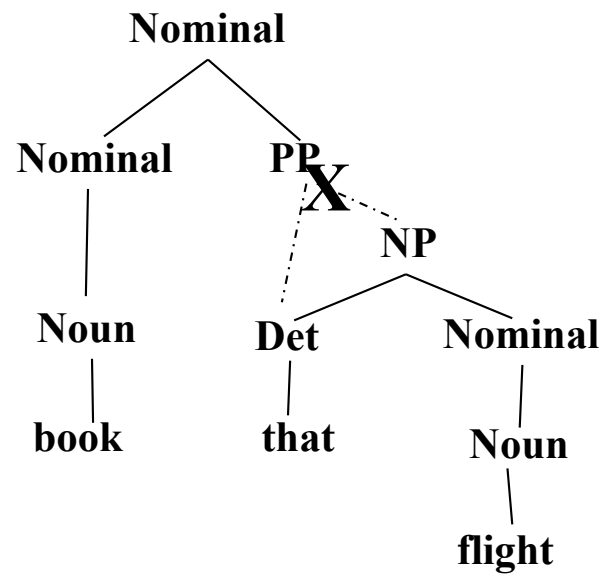
Bottom Up Parsing



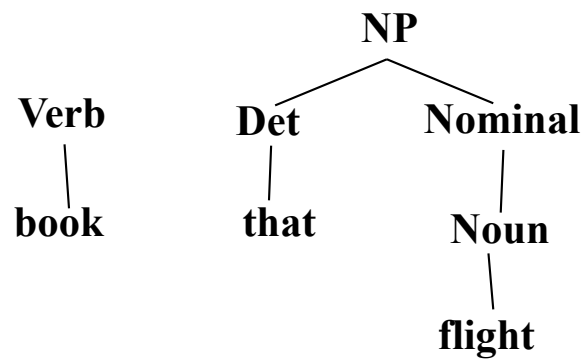
Bottom Up Parsing



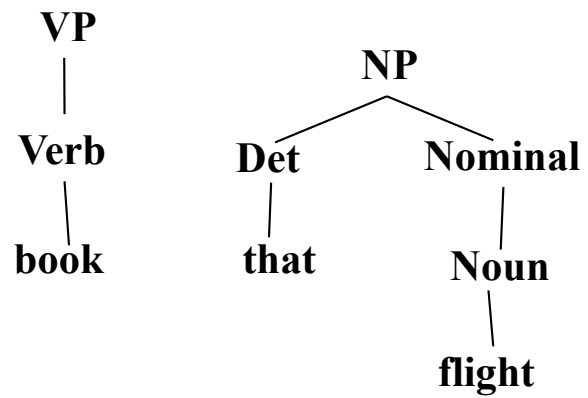
Bottom Up Parsing



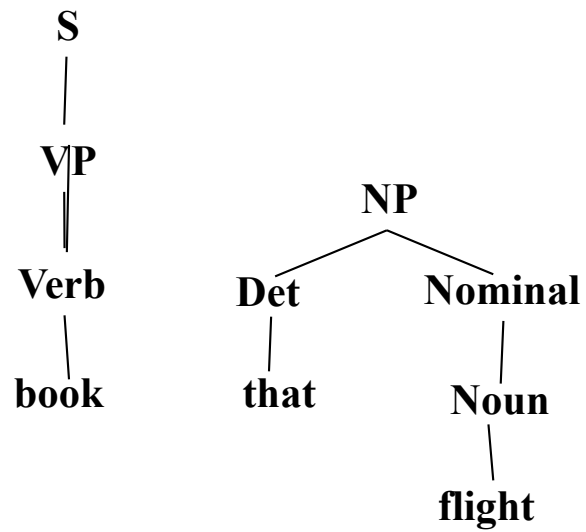
Bottom Up Parsing



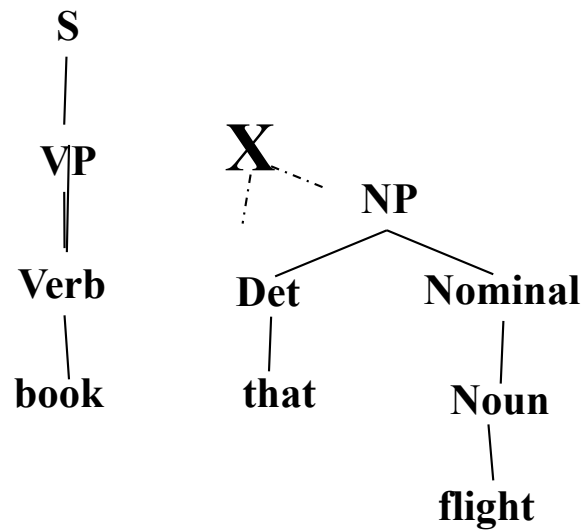
Bottom Up Parsing



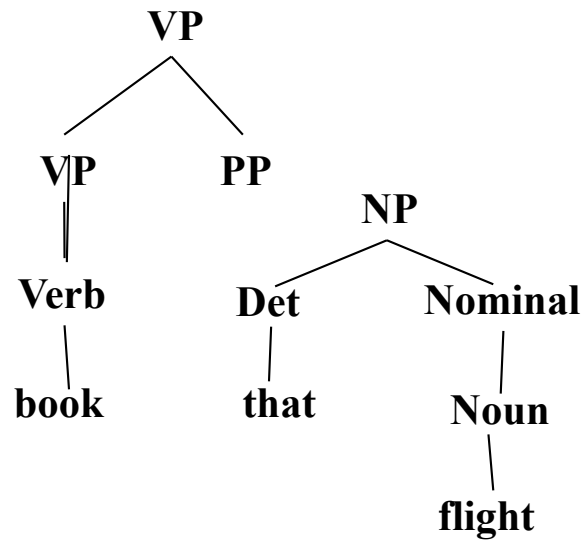
Bottom Up Parsing



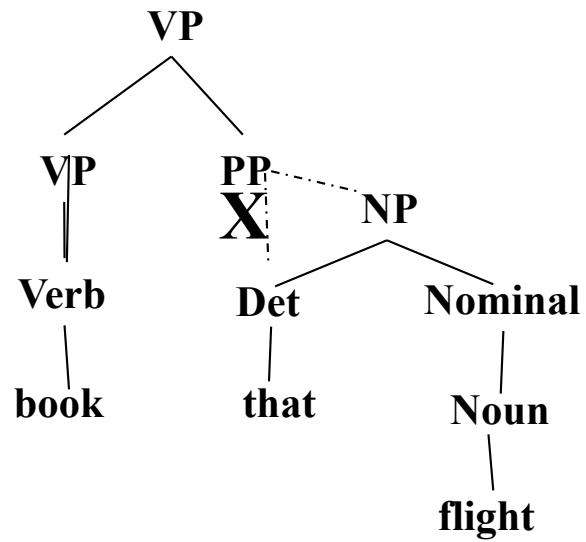
Bottom Up Parsing



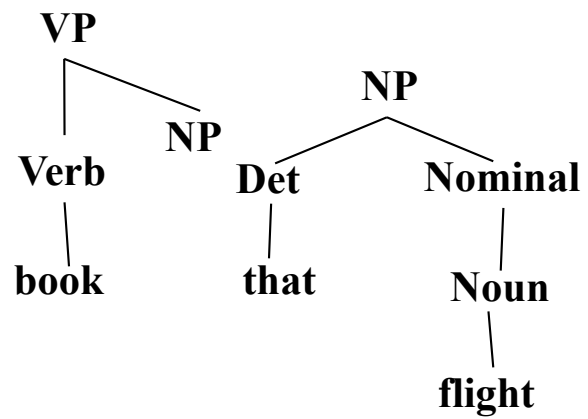
Bottom Up Parsing



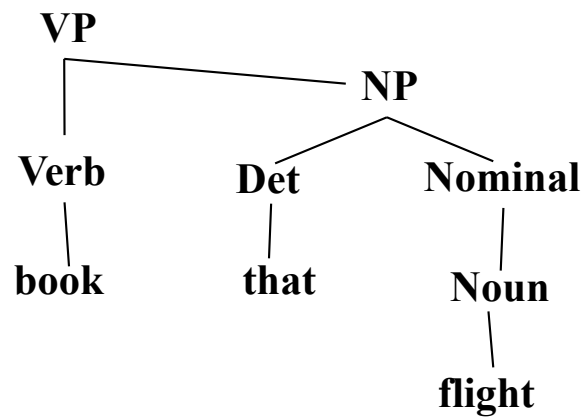
Bottom Up Parsing



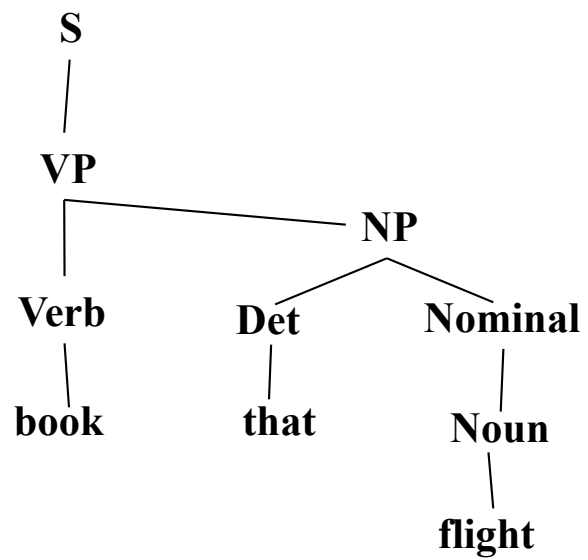
Bottom Up Parsing



Bottom Up Parsing



Bottom Up Parsing

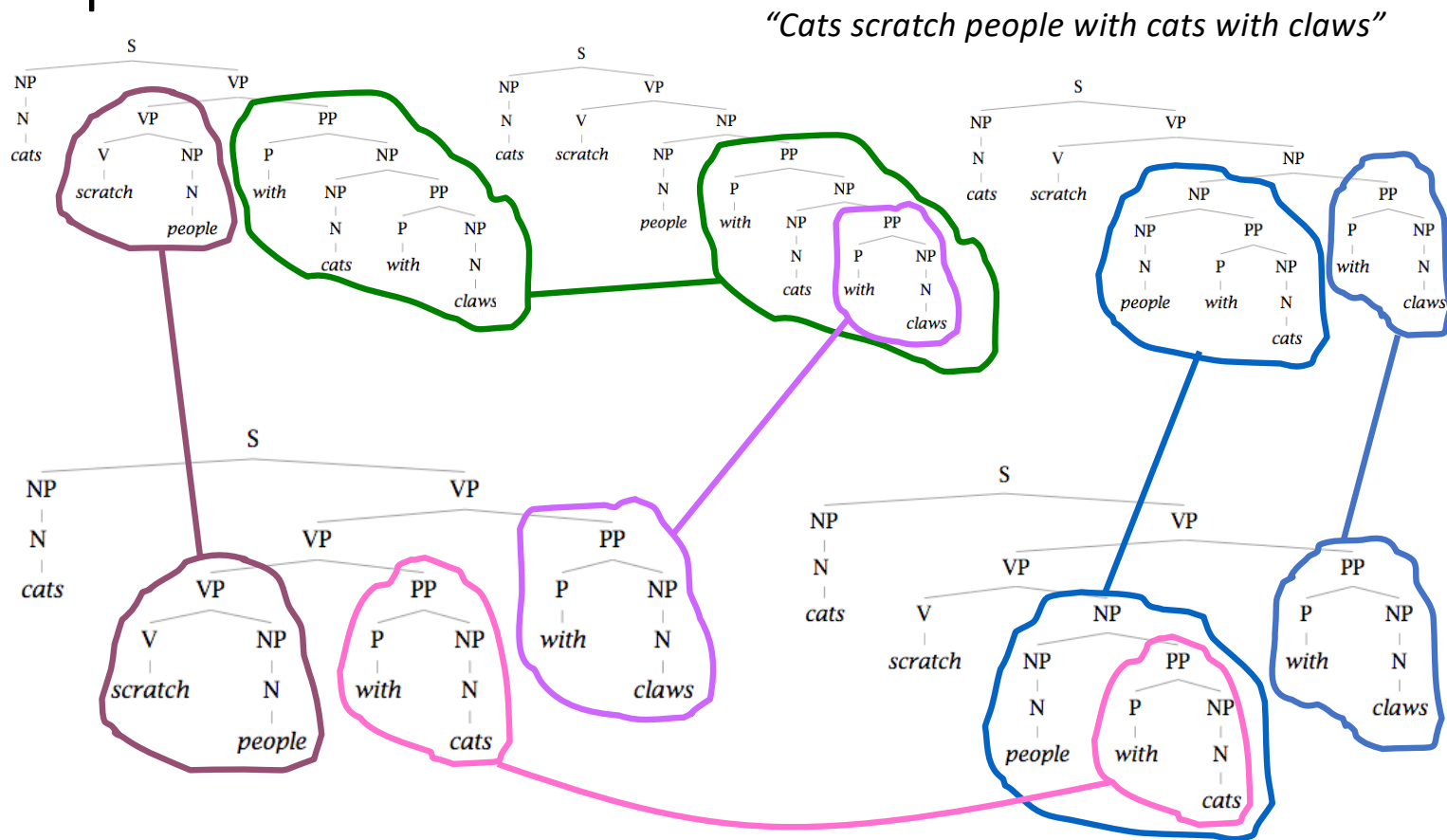


Top Down vs. Bottom Up

- **Top down** never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- **Bottom up** never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.

Two problems to solve for parsing:

1. Repeated work



Dynamic Programming Parsing

- To avoid extensive repeated work, must cache intermediate results, i.e. completed phrases.
- Caching (memorizing) is critical to obtaining a polynomial time parsing (recognition) algorithm for CFGs.

(Probabilistic) CKY Parsing

Constituency Parsing

Input: a PCFG, and a sentence

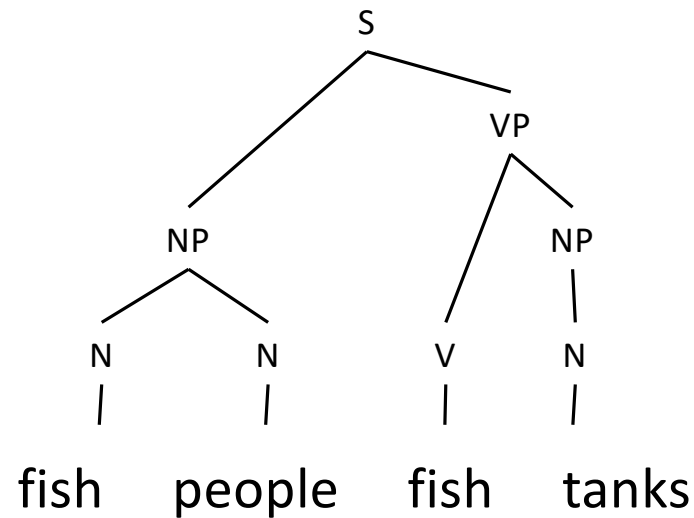
fish people fish tanks

PCFG

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

Constituency Parsing

Output: a parsing tree



PCFG

Rule Prob θ_i

$S \rightarrow NP VP$ θ_0

$NP \rightarrow NP NP$ θ_1

...

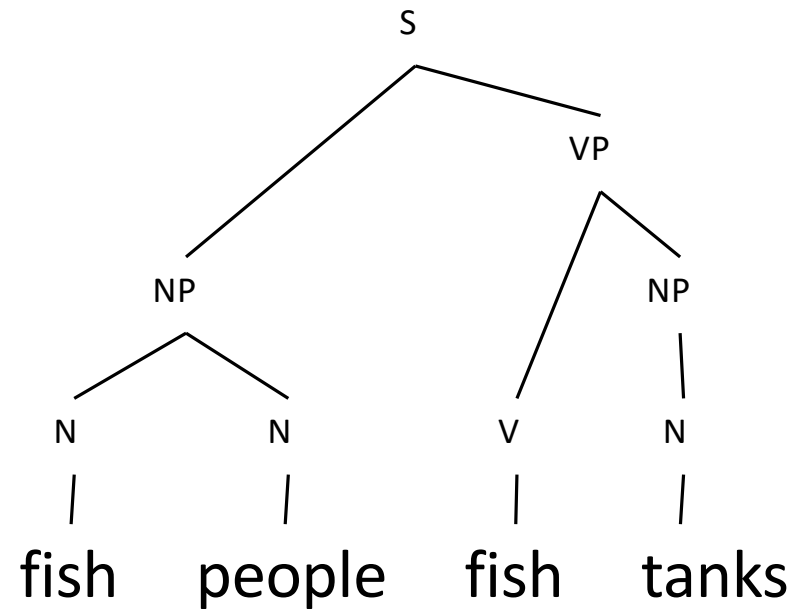
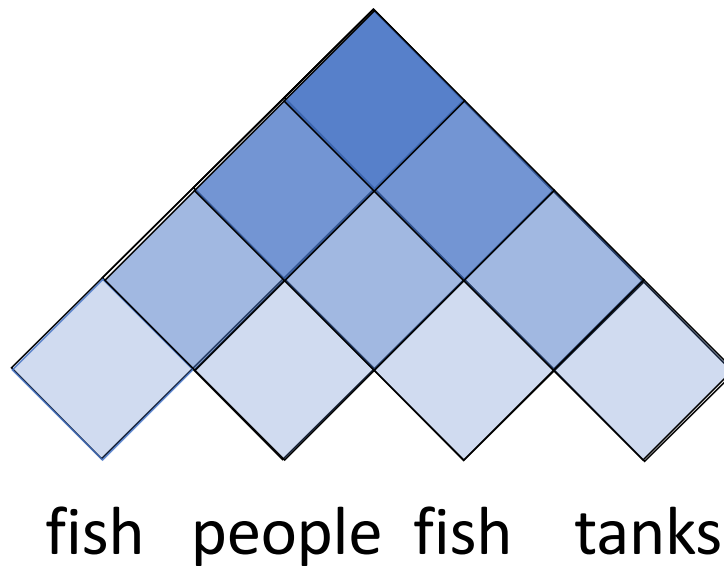
$N \rightarrow \text{fish}$ θ_{42}

$N \rightarrow \text{people}$ θ_{43}

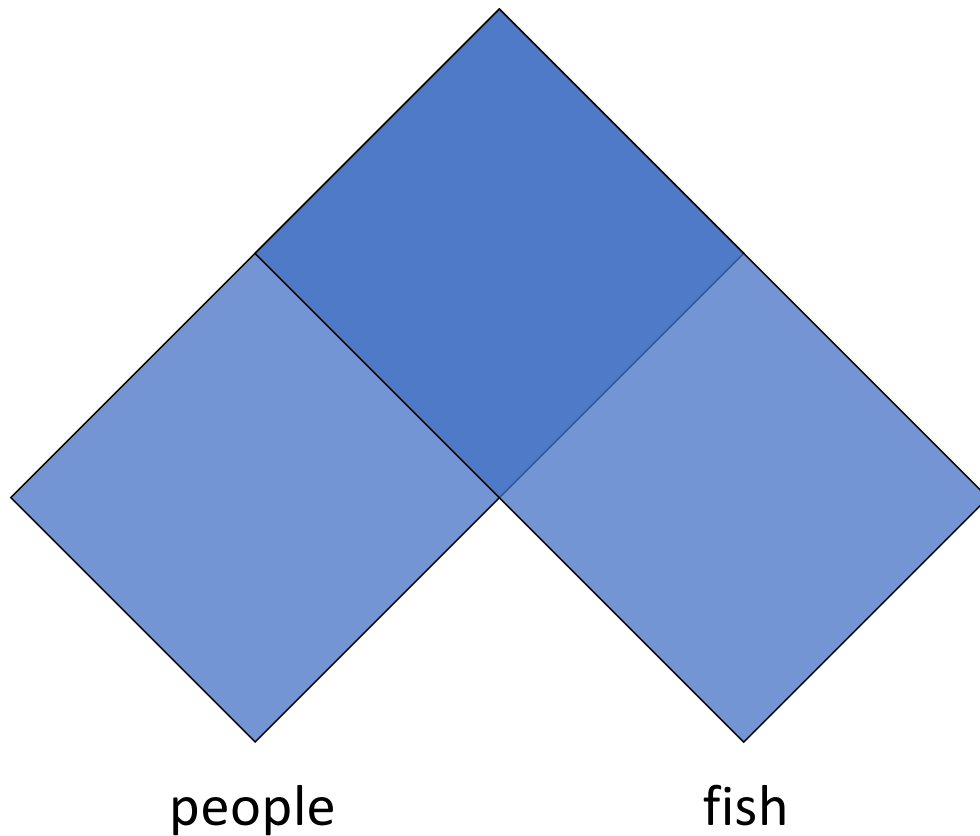
$V \rightarrow \text{fish}$ θ_{44}

...

Cocke-Kasami-Younger (CKY) Constituency Parsing



Reusing local decisions



NP \rightarrow people 0.35

V \rightarrow people 0.1

N \rightarrow people 0.5

VP \rightarrow fish 0.06

V \rightarrow fish 0.6

N \rightarrow fish 0.2

S \rightarrow NP VP 0.9

S \rightarrow VP 0.1

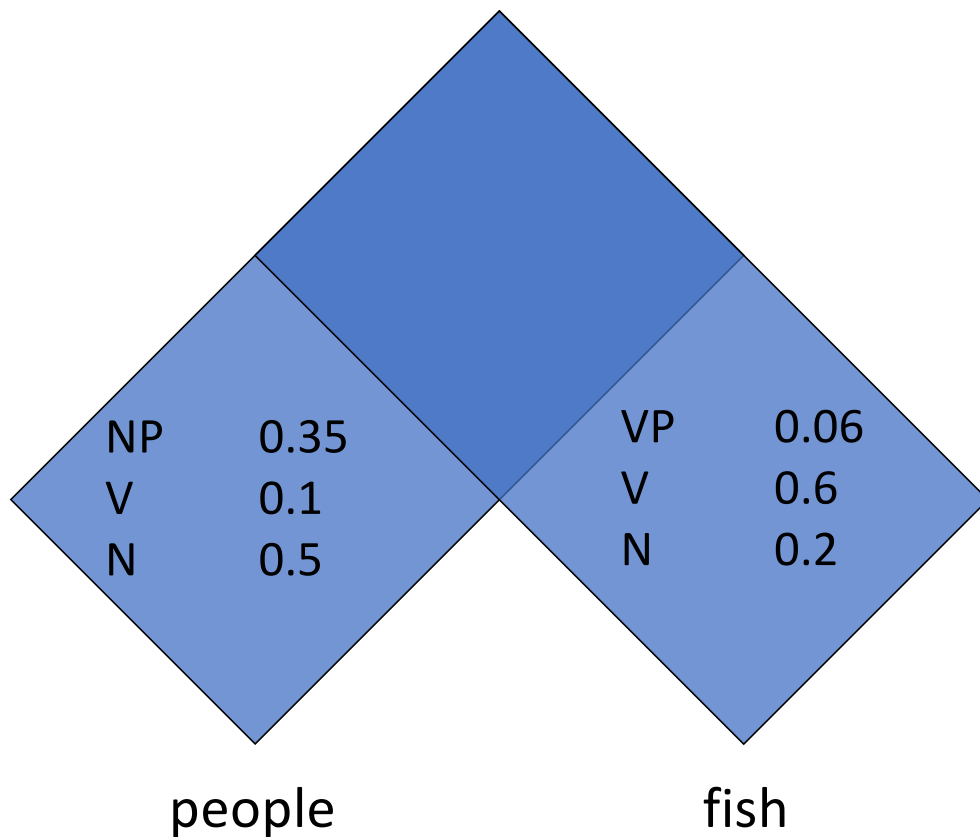
VP \rightarrow V NP 0.5

NP \rightarrow NP NP 0.1

NP \rightarrow NP PP 0.2

PP \rightarrow P NP 1.0

Reusing local decisions



NP → people 0.35

V → people 0.1

N → people 0.5

VP → fish 0.06

V → fish 0.6

N → fish 0.2

S → NP VP 0.9

S → VP 0.1

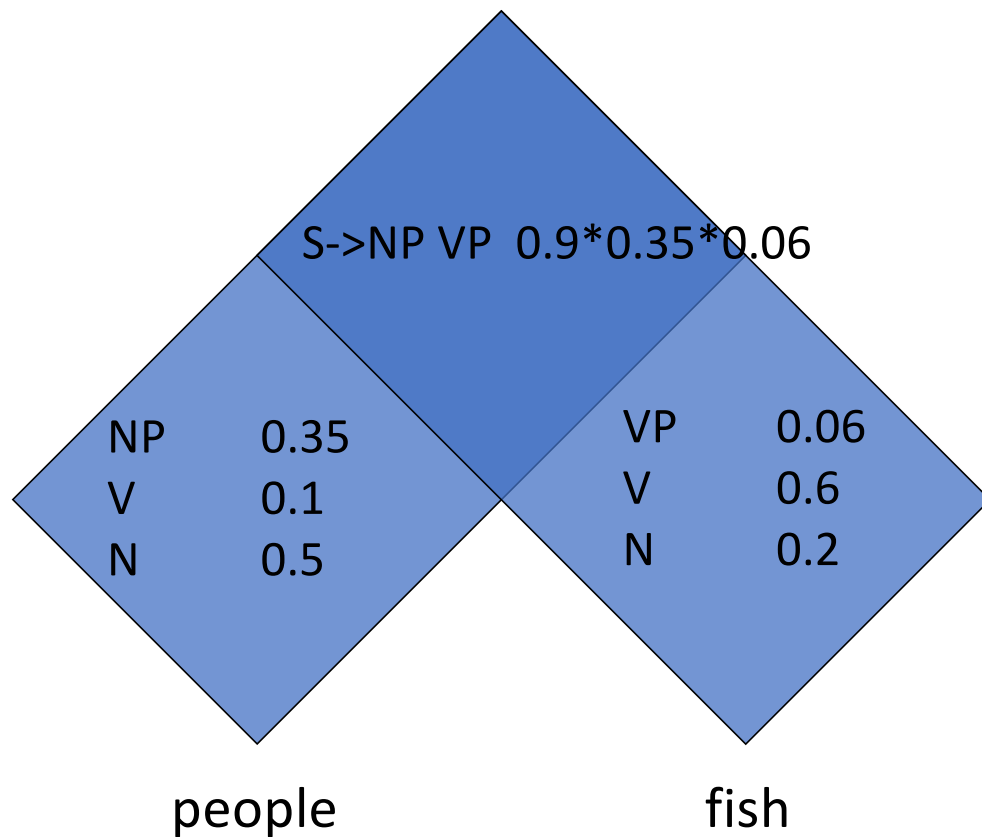
VP → V NP 0.5

NP → NP NP 0.1

NP → NP PP 0.2

PP → P NP 1.0

Reusing local decisions



NP → people	0.35
V → people	0.1
N → people	0.5
VP → fish	0.06
V → fish	0.6
N → fish	0.2

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
NP → NP NP	0.1
NP → NP PP	0.2
PP → P NP	1.0

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)
```