Probabilistic Syntactic Parsing

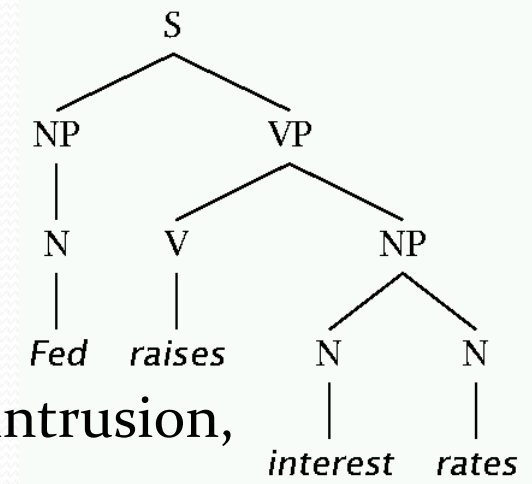
Le Anh Cuong

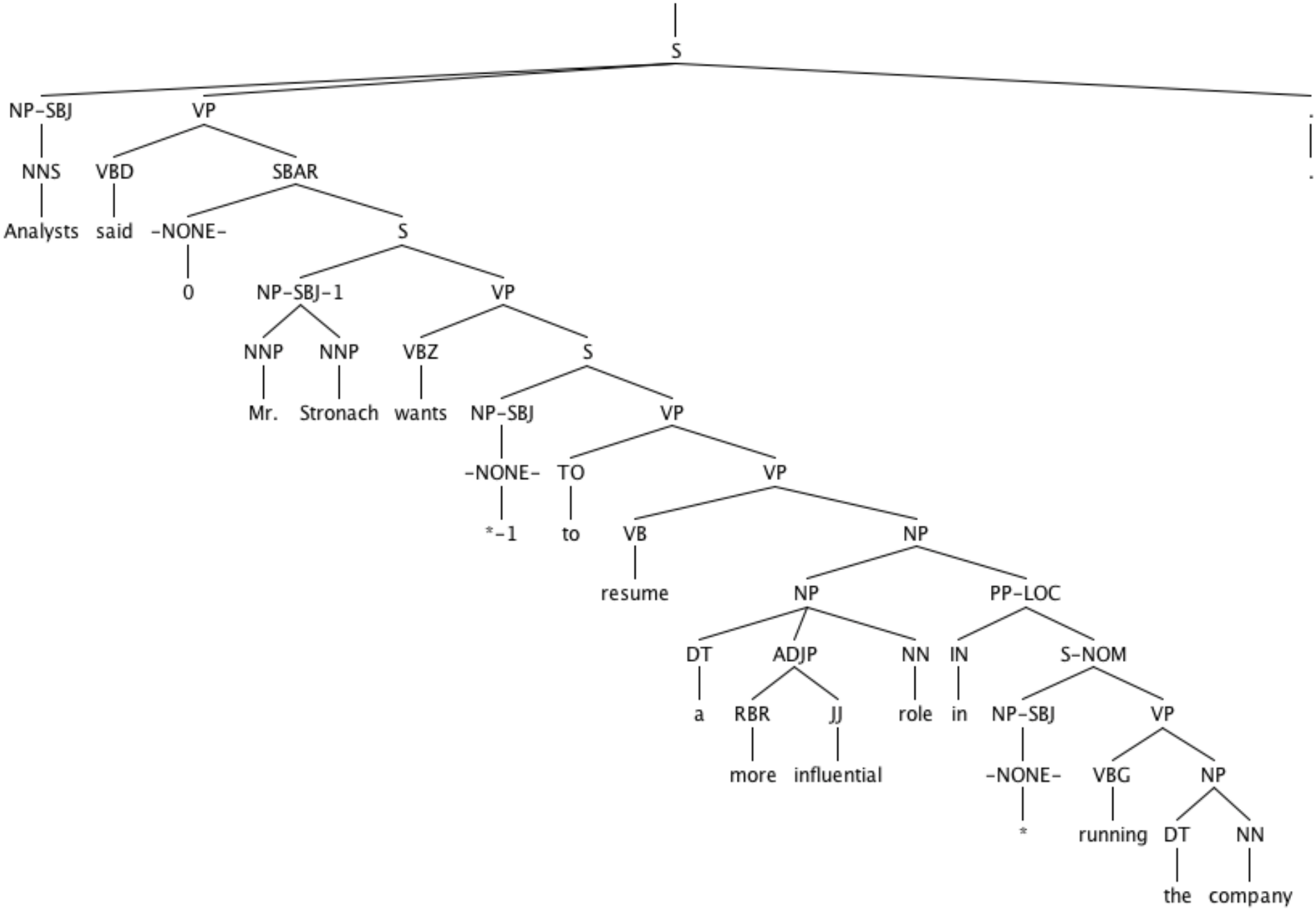
Reading

- Chapter 14 [1]

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/pro-forms:
    - I sat [on the box/right on top of the box/there].
  - Coordination, regular internal structure, no intrusion, fragments, semantics, …




S
NP-SBJ
VP
.
NNS
VBD
SBAR
.
Analysts
said
-NONE-
S
0
NP-SBJ-1
VP
NNP
NNP
VBZ
S
Mr.
Stronach
wants
NP-SBJ
VP
-NONE-
TO
VP
*-1
to
VB
NP
resume
NP
PP-LOC
DT
ADJP
NN
IN
S-NOM
a
RBR
JJ
role
in
NP-SBJ
VP
more
influential
-NONE-
VBG
NP
*
running
DT
NN
the
company

Headed phrase structure

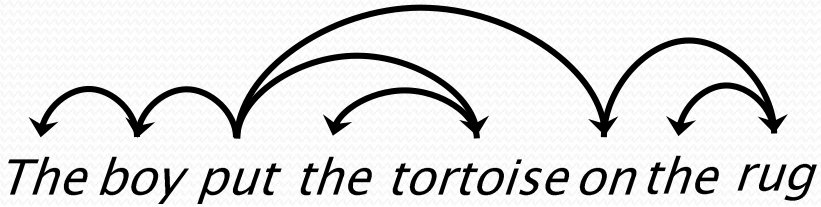
- VP → … VB* …
- NP → … NN* …
- ADJP → … JJ* …
- ADVP → … RB* …

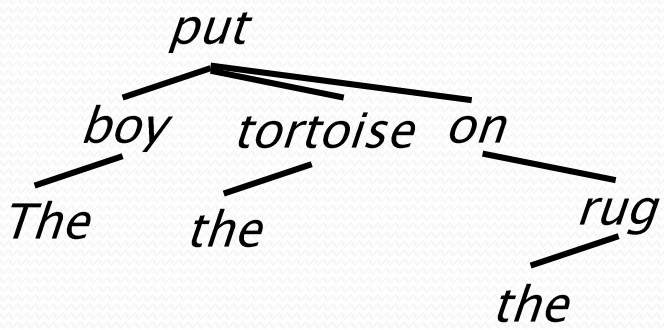
- SBAR(Q) → S|SINV|SQ → … NP VP …

- Plus minor phrase types:
  - QP (quantifier phrase in NP), CONJP (multi word constructions: *as well as*), INTJ (interjections), etc.

Two views of linguistic structure: 2. Dependency structure

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





Pre 1990 ("Classical") NLP Parsing

- Wrote symbolic grammar (CFG or often richer) and lexicon

S → NP VP	NN → *interest*
NP → (DT) NN	NNS → *rates*
NP → NN NNS	NNS → *raises*
NP → NNP	VBP → *interest*
VP → V NP	VBZ → *rates*

- Used grammar/proof systems to prove parses from words
- This scaled very badly and didn't give coverage. For sentence:

  Fed raises interest rates 0.5% in effort to control inflation

  - Minimal grammar: 36 parses
  - Simple 10 rule grammar: 592 parses
  - Real-size broad-coverage grammar: millions of parses

Classical NLP Parsing: The problem and its solution

- Categorical constraints can be added to grammars to limit unlikely/weird parses for sentences
  - But the attempt make the grammars not robust
    - In traditional systems, commonly 30% of sentences in even an edited text would have *no* parse.
- A less constrained grammar can parse more sentences
  - But simple sentences end up with ever more parses with no way to choose between them
- We need mechanisms that allow us to find the most likely parse(s) for a sentence
  - Statistical parsing lets us work with very loose grammars that admit millions of parses for sentences but still quickly find the best parse(s)

The rise of 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

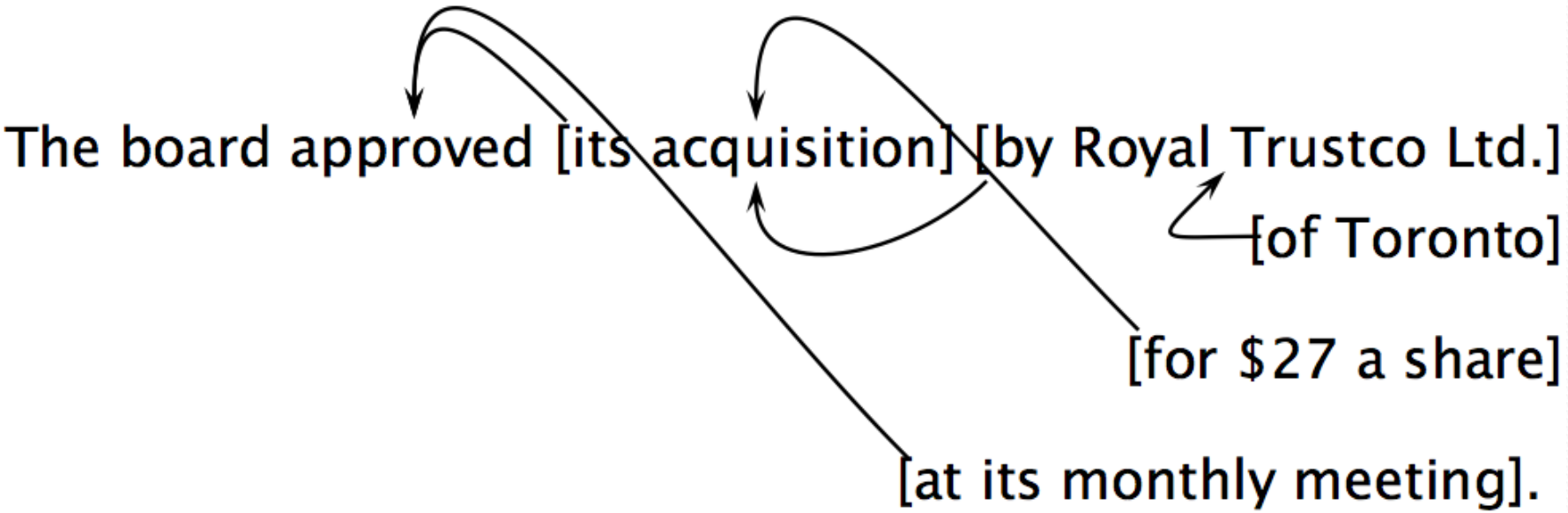
Statistical parsing applications

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]

Attachment ambiguities

- A key parsing decision is how we 'attach' various constituents
  - PPs, adverbial or participial phrases, infinitives, coordinations, etc.



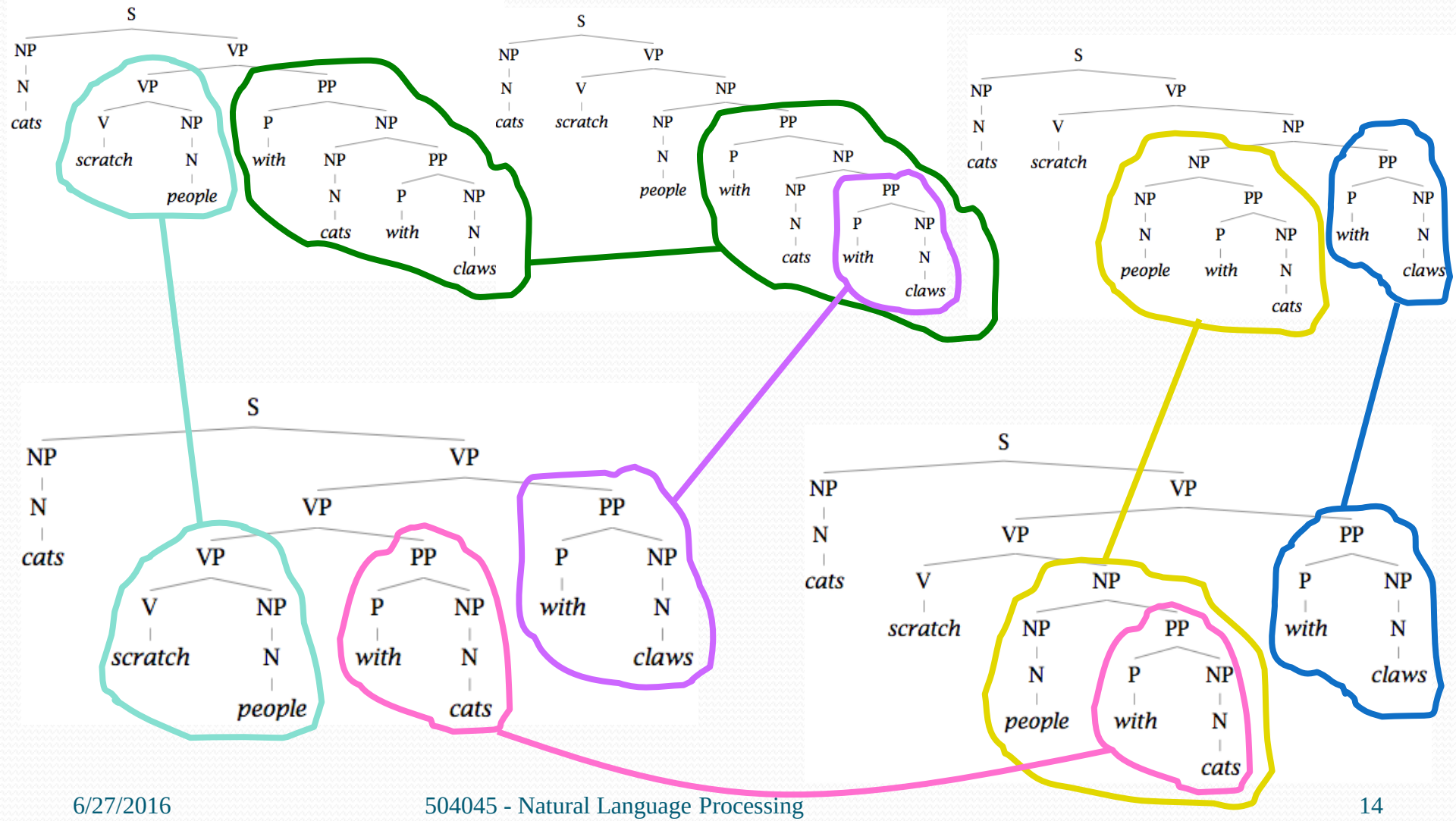
- Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$
- An exponentially growing series, which arises in many tree-like contexts:
  - E.g., the number of possible triangulations of a polygon with $n+2$ sides
    - Turns up in triangulation of probabilistic graphical models….

Quiz Question!

- How many distinct parses does the following sentence have due to PP attachment ambiguities?
  - A PP can attach to any preceding V or N within the verb phrase, subject only to the parse still being a tree.
    - (This is equivalent to there being no crossing dependencies, where if d_2 is a dependent of d_1 and d_3 is a dependent of d_2, then the line d_2–d_3 begins at d_2 under the line from d_1 to d_2.)

John wrote the book with a pen in the room.

Two problems to solve:
1. Repeated work…



Two problems to solve:
2. Choosing the correct parse

- How do we work out the correct attachment:
  - She saw the man with a telescope
- Is the problem 'AI complete'? Yes, but …
- 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.

CFGs and PCFGs

(Probabilistic) Context-Free Grammars

A phrase structure grammar

$S \rightarrow NP\ VP$
$VP \rightarrow V\ NP$
$VP \rightarrow V\ NP\ PP$
$NP \rightarrow NP\ NP$
$NP \rightarrow NP\ PP$
$NP \rightarrow N$
$NP \rightarrow e$
$PP \rightarrow P\ NP$

people fish tanks
people fish with rods

$N \rightarrow$ people
$N \rightarrow$ fish
$N \rightarrow$ tanks
$N \rightarrow$ rods
$V \rightarrow$ people
$V \rightarrow$ fish
$V \rightarrow$ tanks
$P \rightarrow$ with

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

$S \rightarrow NP\ VP$
$VP \rightarrow V\ NP$
$VP \rightarrow V\ NP\ PP$
$NP \rightarrow NP\ NP$
$NP \rightarrow NP\ PP$
$NP \rightarrow N$
$NP \rightarrow e$
$PP \rightarrow P\ NP$

people fish tanks
people fish with rods

$N \rightarrow$ *people*
$N \rightarrow$ *fish*
$N \rightarrow$ *tanks*
$N \rightarrow$ *rods*
$V \rightarrow$ *people*
$V \rightarrow$ *fish*
$V \rightarrow$ *tanks*
$P \rightarrow$ *with*

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 ∈ N)
  - R is a set of rules/productions of the form $X \rightarrow \gamma$
  - P is a probability function
    - P: R → [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.

$$\sum_{\omega \in T^*} P(\omega) = 1$$

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

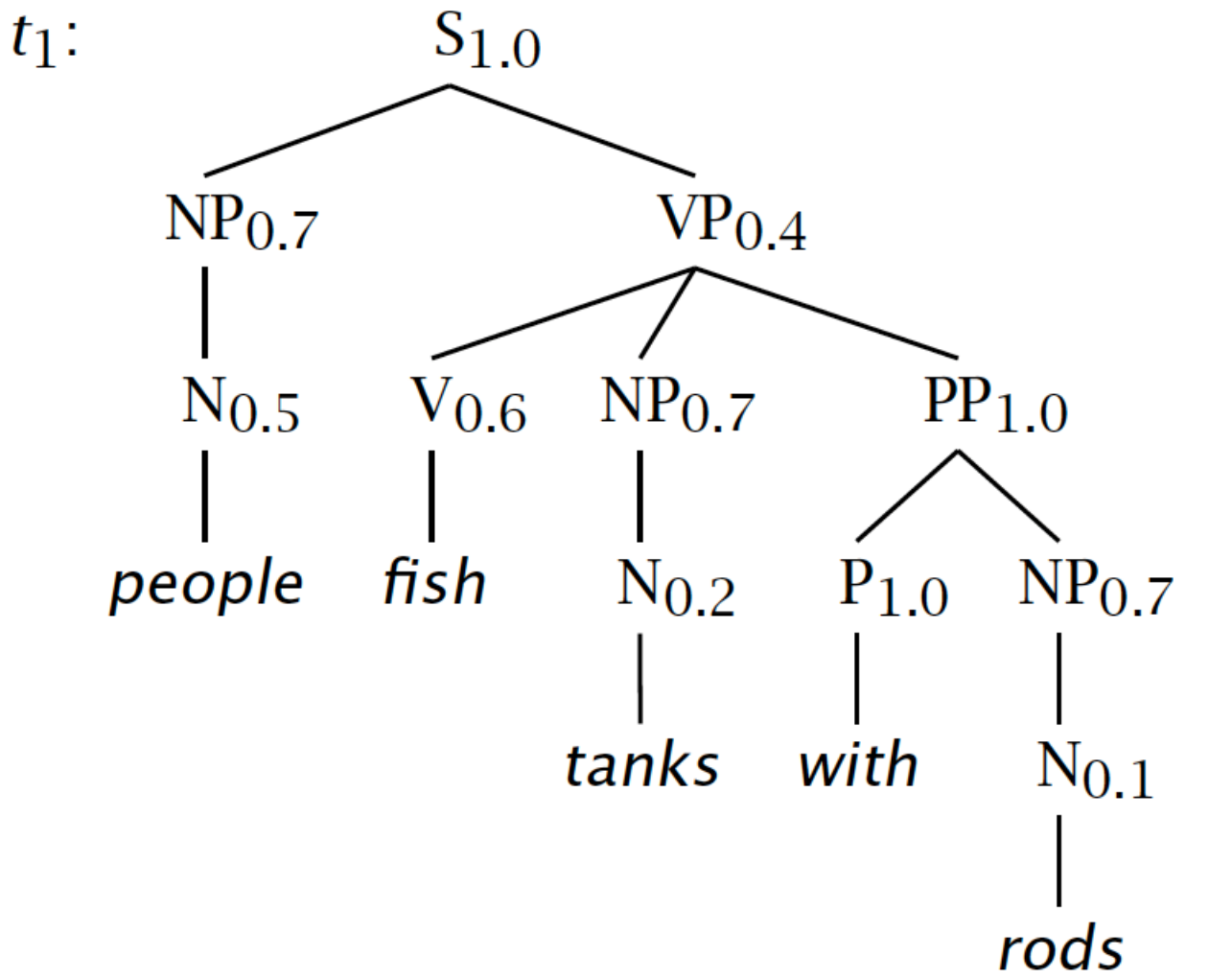
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

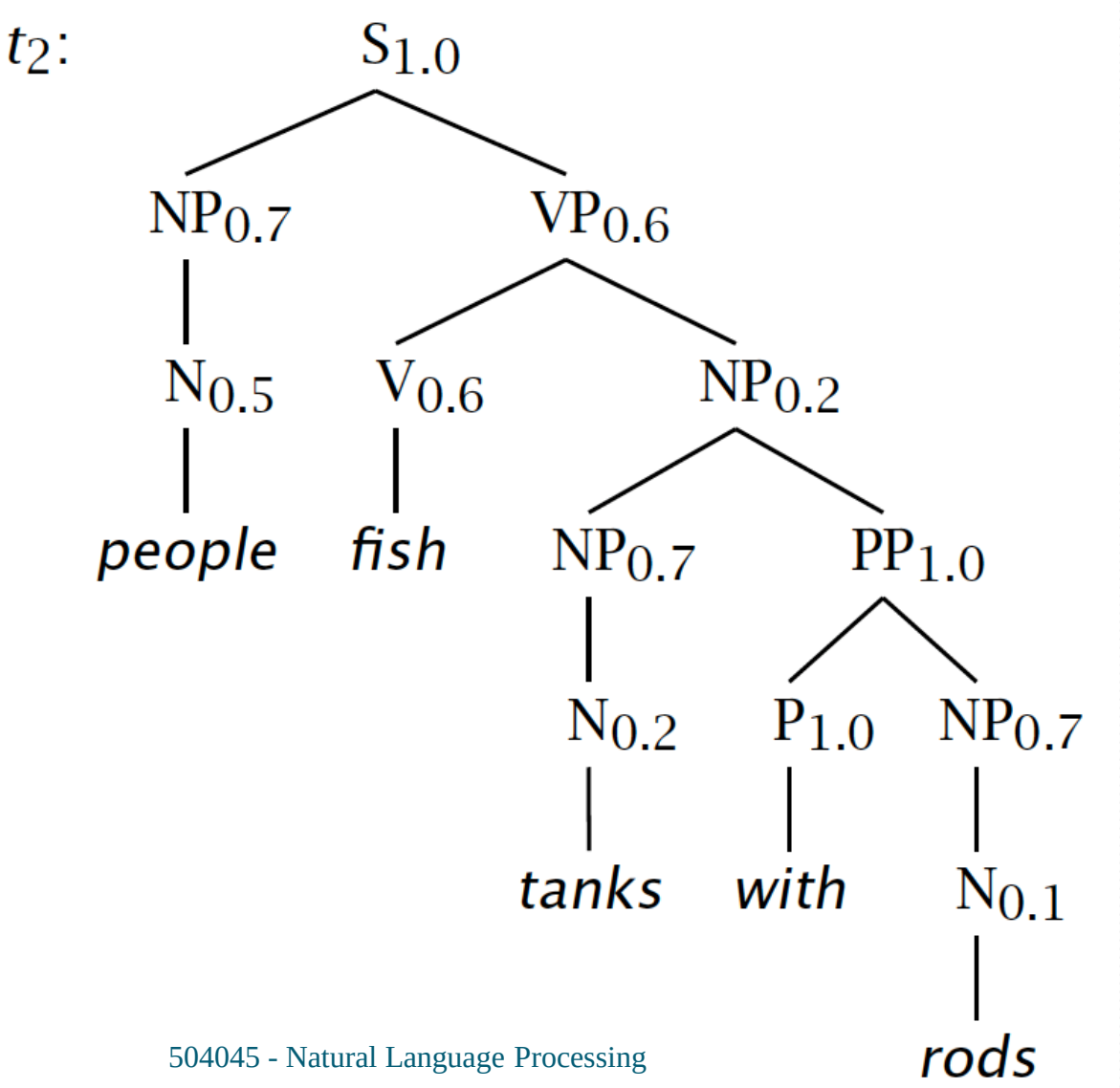
[With empty NP removed so less ambiguous]

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

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


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

Tree and String Probabilities

- s = *people fish tanks with rods*
- $P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$ Verb attach

  $= 0.0008232$
- $P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$ Noun attach

  $= 0.00024696$ [more depth → small number]
- $P(s) = P(t_1) + P(t_2)$

  $= 0.0008232 + 0.00024696$

  $= 0.00107016$

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

$S \rightarrow NP\ VP$

$VP \rightarrow V\ NP$

$VP \rightarrow V\ NP\ PP$

$NP \rightarrow NP\ NP$

$NP \rightarrow NP\ PP$

$NP \rightarrow N$

$\mathbf{NP \rightarrow e}$

$PP \rightarrow P\ NP$

$N \rightarrow people$

$N \rightarrow fish$

$N \rightarrow tanks$

$N \rightarrow rods$

$V \rightarrow people$

$V \rightarrow fish$

$V \rightarrow tanks$

$P \rightarrow with$

Chomsky Normal Form steps

$S \rightarrow NP\ VP$

$\mathbf{S \rightarrow VP}$

$VP \rightarrow V\ NP$

$VP \rightarrow V$

$VP \rightarrow V\ NP\ PP$

$VP \rightarrow V\ PP$

$NP \rightarrow NP\ NP$

$NP \rightarrow NP$

$NP \rightarrow NP\ PP$

$NP \rightarrow PP$

$NP \rightarrow N$

$PP \rightarrow P\ NP$

$PP \rightarrow P$

$N \rightarrow \textit{people}$

$N \rightarrow \textit{fish}$

$N \rightarrow \textit{tanks}$

$N \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

Chomsky Normal Form steps

$S \rightarrow NP\ VP$
$VP \rightarrow V\ NP$
$S \rightarrow V\ NP$
$VP \rightarrow V$
$\mathbf{S \rightarrow V}$
$VP \rightarrow V\ NP\ PP$
$S \rightarrow V\ NP\ PP$
$VP \rightarrow V\ PP$
$S \rightarrow V\ PP$
$NP \rightarrow NP\ NP$
$NP \rightarrow NP$
$NP \rightarrow NP\ PP$
$NP \rightarrow PP$
$NP \rightarrow N$
$PP \rightarrow P\ NP$
$PP \rightarrow P$

$N \rightarrow people$
$N \rightarrow fish$
$N \rightarrow tanks$
$N \rightarrow rods$
$V \rightarrow people$
$V \rightarrow fish$
$V \rightarrow tanks$
$P \rightarrow with$

Chomsky Normal Form steps

$S \rightarrow NP\ VP$
$VP \rightarrow V\ NP$
$S \rightarrow V\ NP$
$\mathbf{VP \rightarrow V}$
$VP \rightarrow V\ NP\ PP$
$S \rightarrow V\ NP\ PP$
$VP \rightarrow V\ PP$
$S \rightarrow V\ PP$
$NP \rightarrow NP\ NP$
$NP \rightarrow NP$
$NP \rightarrow NP\ PP$
$NP \rightarrow PP$
$NP \rightarrow N$
$PP \rightarrow P\ NP$
$PP \rightarrow P$

$N \rightarrow \textit{people}$
$N \rightarrow \textit{fish}$
$N \rightarrow \textit{tanks}$
$N \rightarrow \textit{rods}$
$V \rightarrow \textit{people}$
$S \rightarrow \textit{people}$
$V \rightarrow \textit{fish}$
$S \rightarrow \textit{fish}$
$V \rightarrow \textit{tanks}$
$S \rightarrow \textit{tanks}$
$P \rightarrow \textit{with}$

Chomsky Normal Form steps

$S \rightarrow NP\ VP$

$VP \rightarrow V\ NP$

$S \rightarrow V\ NP$

$VP \rightarrow V\ NP\ PP$

$S \rightarrow V\ NP\ PP$

$VP \rightarrow V\ PP$

$S \rightarrow V\ PP$

$NP \rightarrow NP\ NP$

$\mathbf{NP \rightarrow NP}$

$NP \rightarrow NP\ PP$

$NP \rightarrow PP$

$\mathbf{NP \rightarrow N}$

$PP \rightarrow P\ NP$

$PP \rightarrow P$

$N \rightarrow \textit{people}$

$N \rightarrow \textit{fish}$

$N \rightarrow \textit{tanks}$

$N \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$S \rightarrow \textit{people}$

$VP \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$S \rightarrow \textit{fish}$

$VP \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$S \rightarrow \textit{tanks}$

$VP \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

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

$S \rightarrow NP\ VP$

$VP \rightarrow V\ NP$

$S \rightarrow V\ NP$

$VP \rightarrow V\ @VP\_V$

$@VP\_V \rightarrow NP\ PP$

$S \rightarrow V\ @S\_V$

$@S\_V \rightarrow NP\ PP$

$VP \rightarrow V\ PP$

$S \rightarrow V\ PP$

$NP \rightarrow NP\ NP$

$NP \rightarrow NP\ PP$

$NP \rightarrow P\ NP$

$PP \rightarrow P\ NP$

$NP \rightarrow \textit{people}$

$NP \rightarrow \textit{fish}$

$NP \rightarrow \textit{tanks}$

$NP \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$S \rightarrow \textit{people}$

$VP \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$S \rightarrow \textit{fish}$

$VP \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$S \rightarrow \textit{tanks}$

$VP \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

$PP \rightarrow \textit{with}$

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

$S \rightarrow NP\ VP$

$VP \rightarrow V\ NP$

$S \rightarrow V\ NP$

$VP \rightarrow V\ @VP\_V$

$@VP\_V \rightarrow NP\ PP$

$S \rightarrow V\ @S\_V$

$@S\_V \rightarrow NP\ PP$

$VP \rightarrow V\ PP$

$S \rightarrow V\ PP$

$NP \rightarrow NP\ NP$

$NP \rightarrow NP\ PP$

$NP \rightarrow P\ NP$

$PP \rightarrow P\ NP$

$NP \rightarrow \textit{people}$

$NP \rightarrow \textit{fish}$

$NP \rightarrow \textit{tanks}$

$NP \rightarrow \textit{rods}$

$V \rightarrow \textit{people}$

$S \rightarrow \textit{people}$

$VP \rightarrow \textit{people}$

$V \rightarrow \textit{fish}$

$S \rightarrow \textit{fish}$

$VP \rightarrow \textit{fish}$

$V \rightarrow \textit{tanks}$

$S \rightarrow \textit{tanks}$

$VP \rightarrow \textit{tanks}$

$P \rightarrow \textit{with}$

$PP \rightarrow \textit{with}$

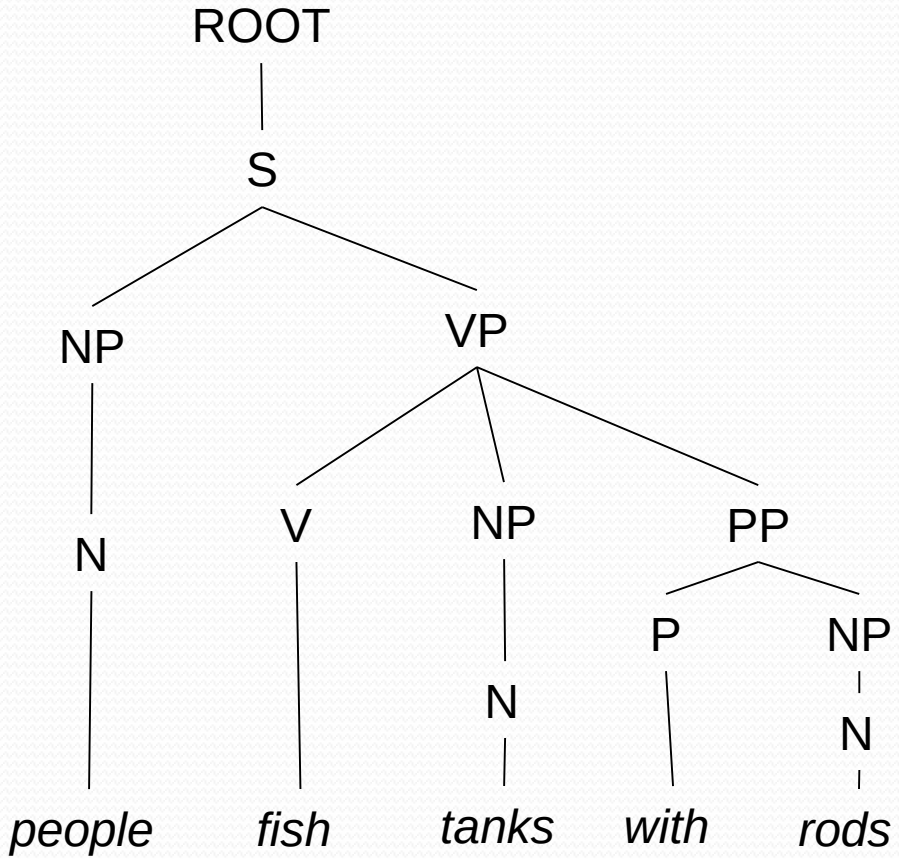
Chomsky Normal Form

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

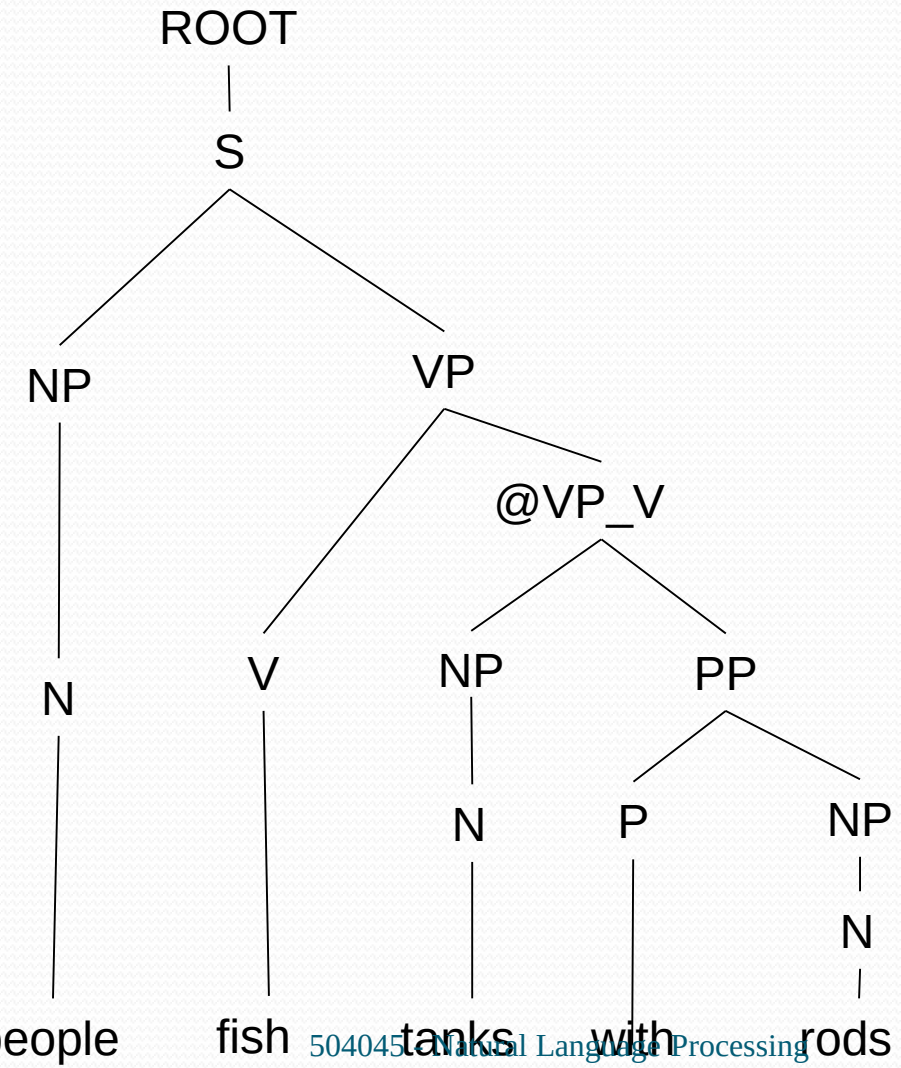
- **Binarization** is crucial for cubic time CFG parsing

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

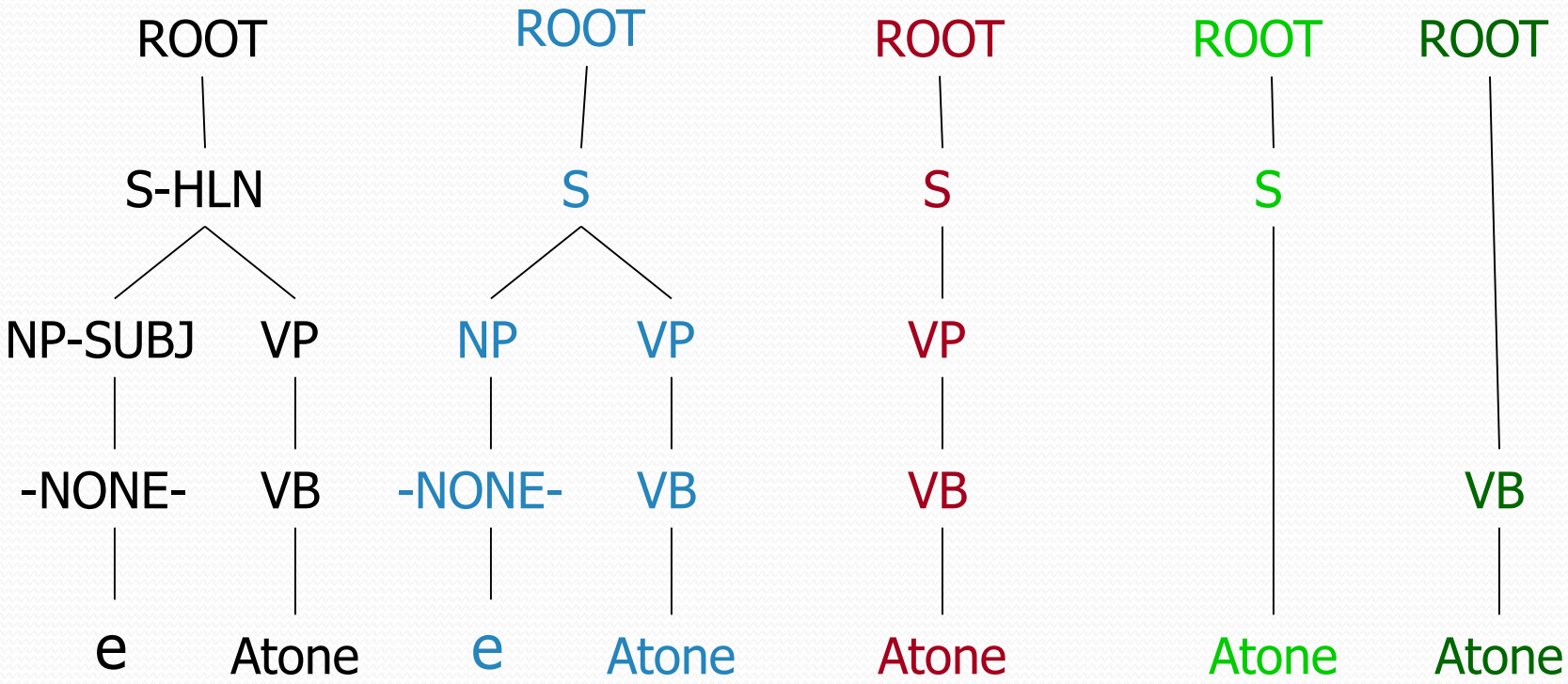
An example: before binarization…



After binarization…



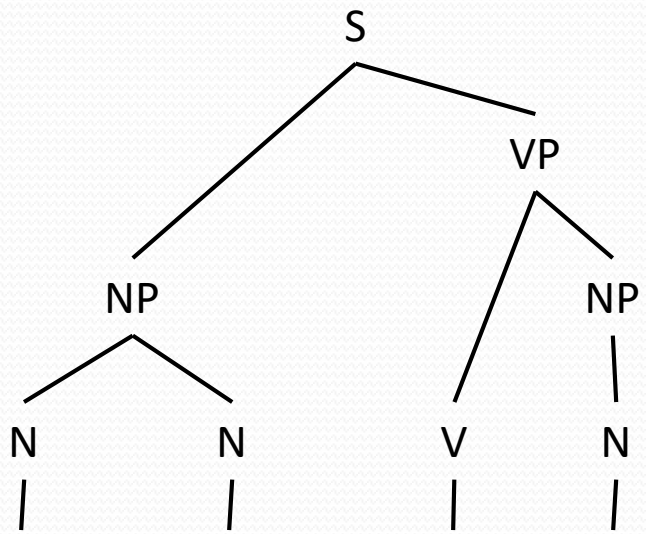
Treebank: empties and unaries



CKY Parsing

Exact polynomial time parsing of (P)CFGs

Constituency Parsing

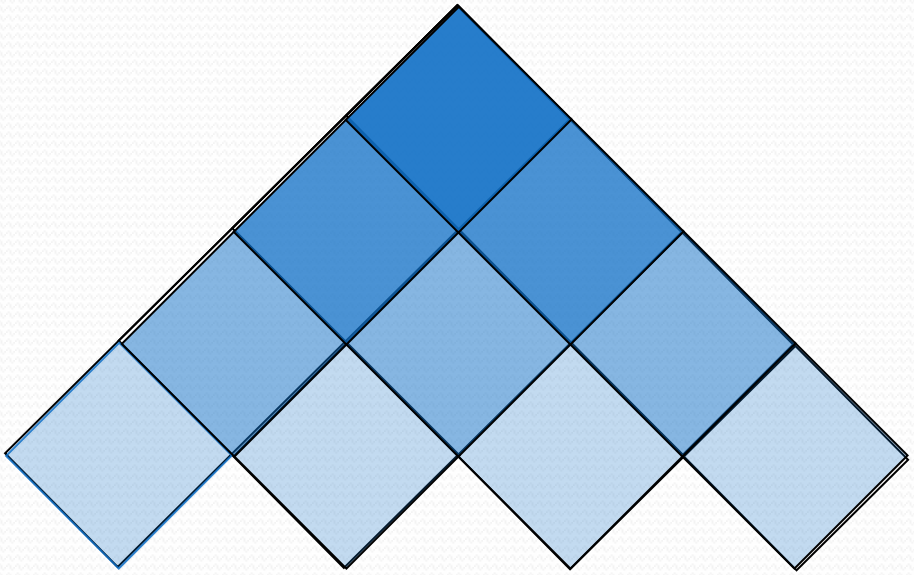


fish people fish tanks

PCFG

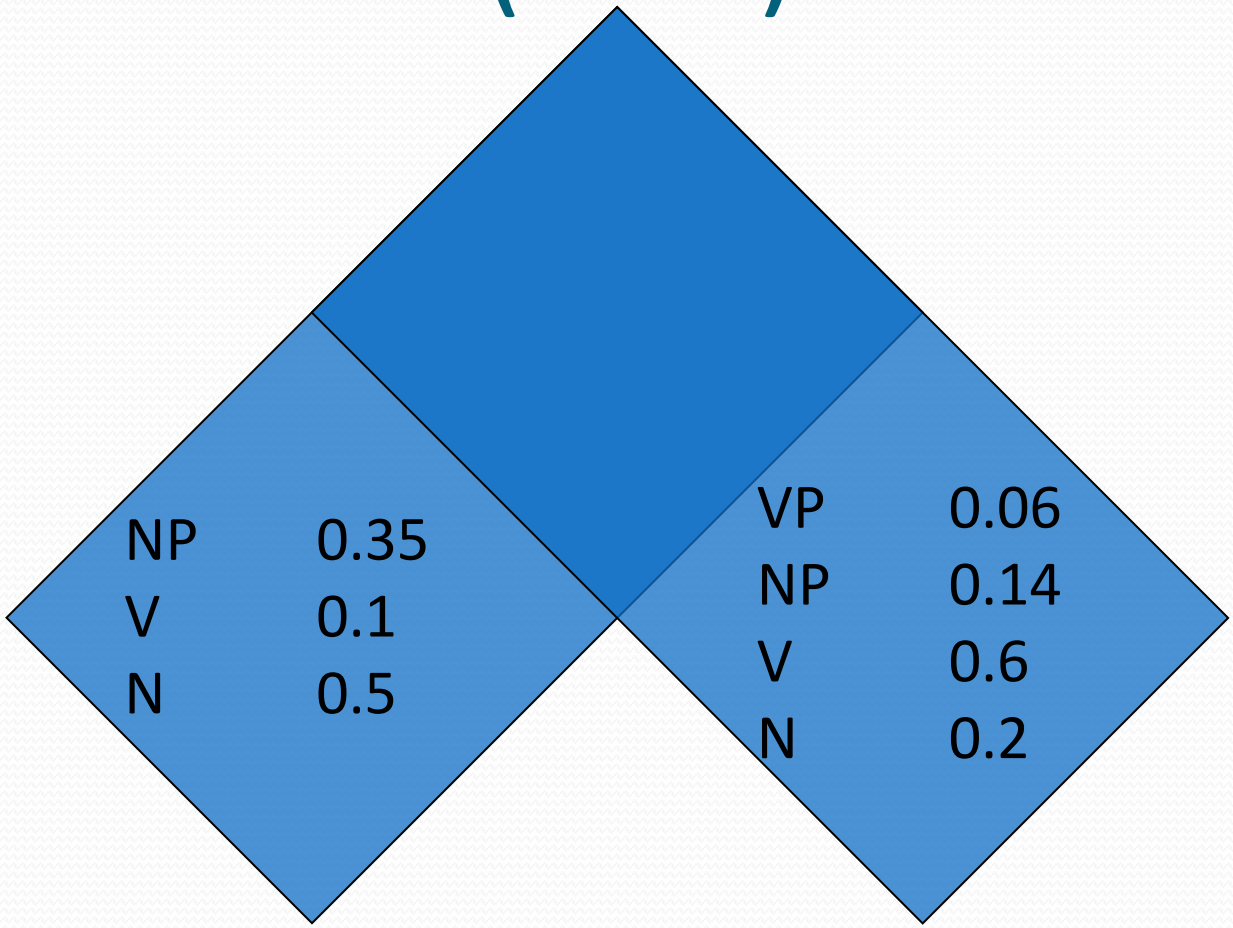
Rule	**Prob θ_i**
$S \rightarrow NP\ VP$	θ_0
$NP \rightarrow NP\ NP$	θ_1
…	
$N \rightarrow$ fish	θ_{42}
$N \rightarrow$ people	θ_{43}
$V \rightarrow$ fish	θ_{44}
…	

Cocke-Kasami-Younger (CKY) Constituency Parsing



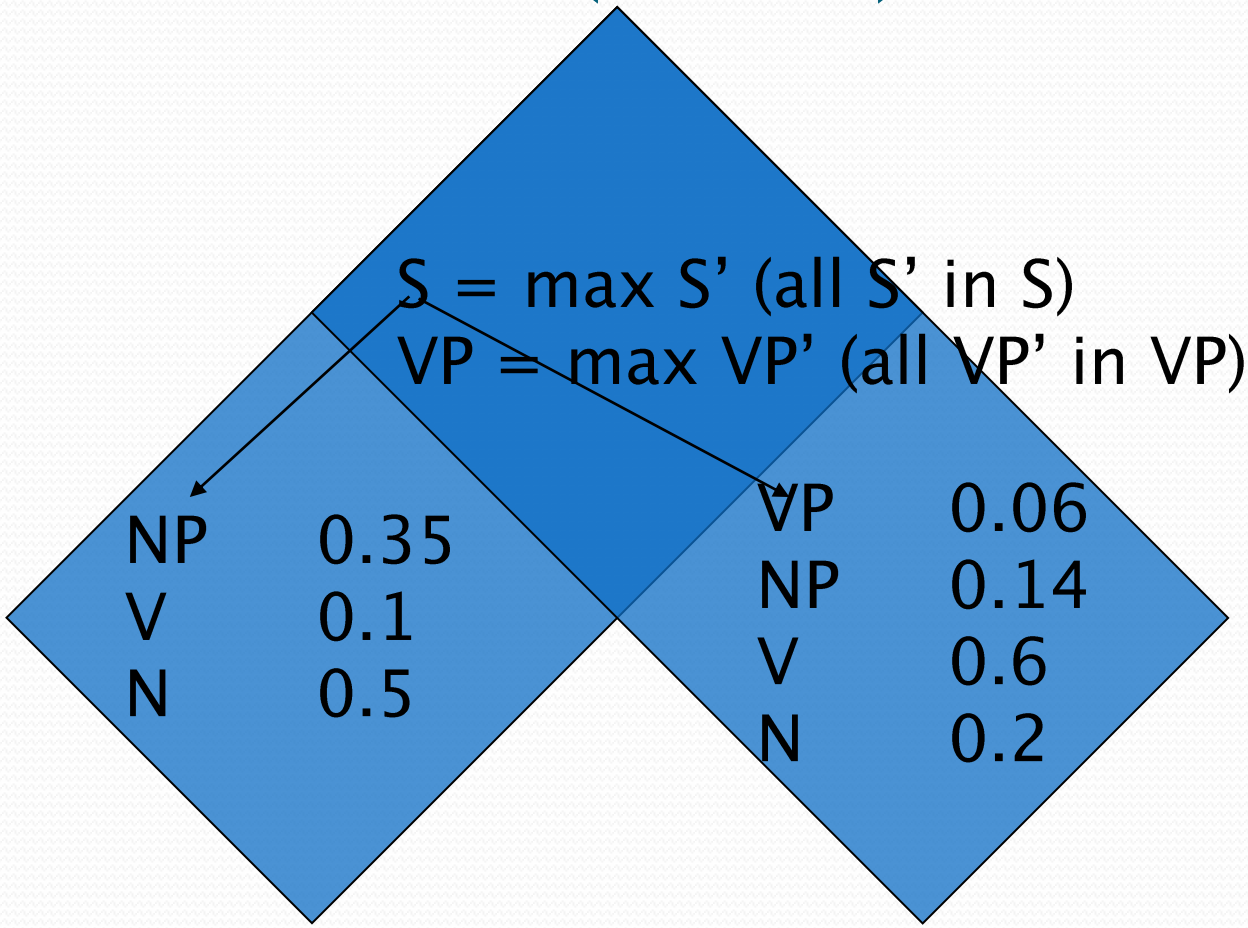
fish people fish tanks

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

Viterbi (Max) Scores



Extended CKY parsing

- Unaries can be incorporated into the algorithm
  - Messy, but doesn't increase algorithmic complexity
- Empties can be incorporated
  - Use fenceposts – [0 people 1 fish 2 tank 3 fish 4]
  - 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.

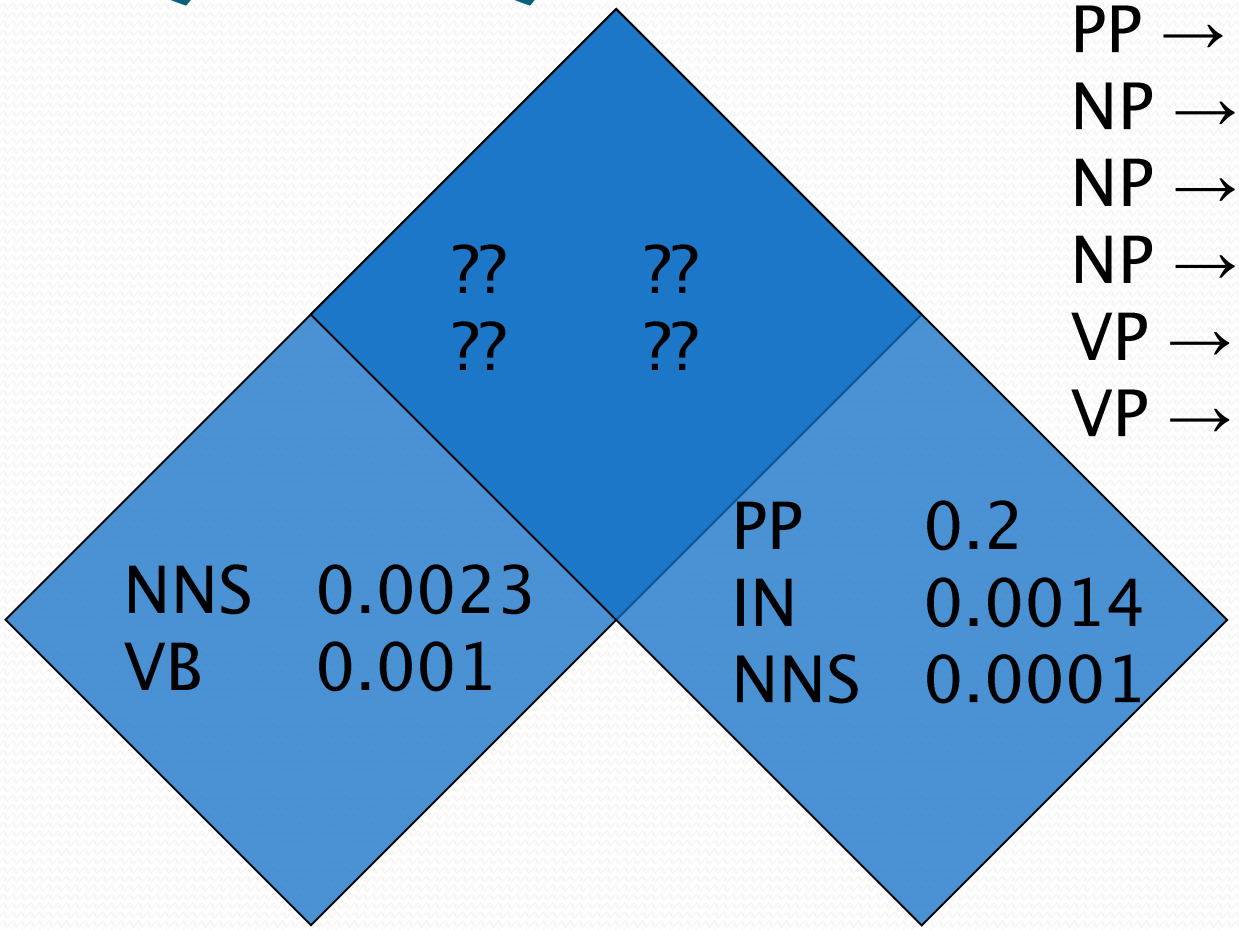
The CKY algorithm (1960/1965) … extended to unaries

```
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!



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

What constituents (with what probability can you make?

CKY Parsing

A worked example

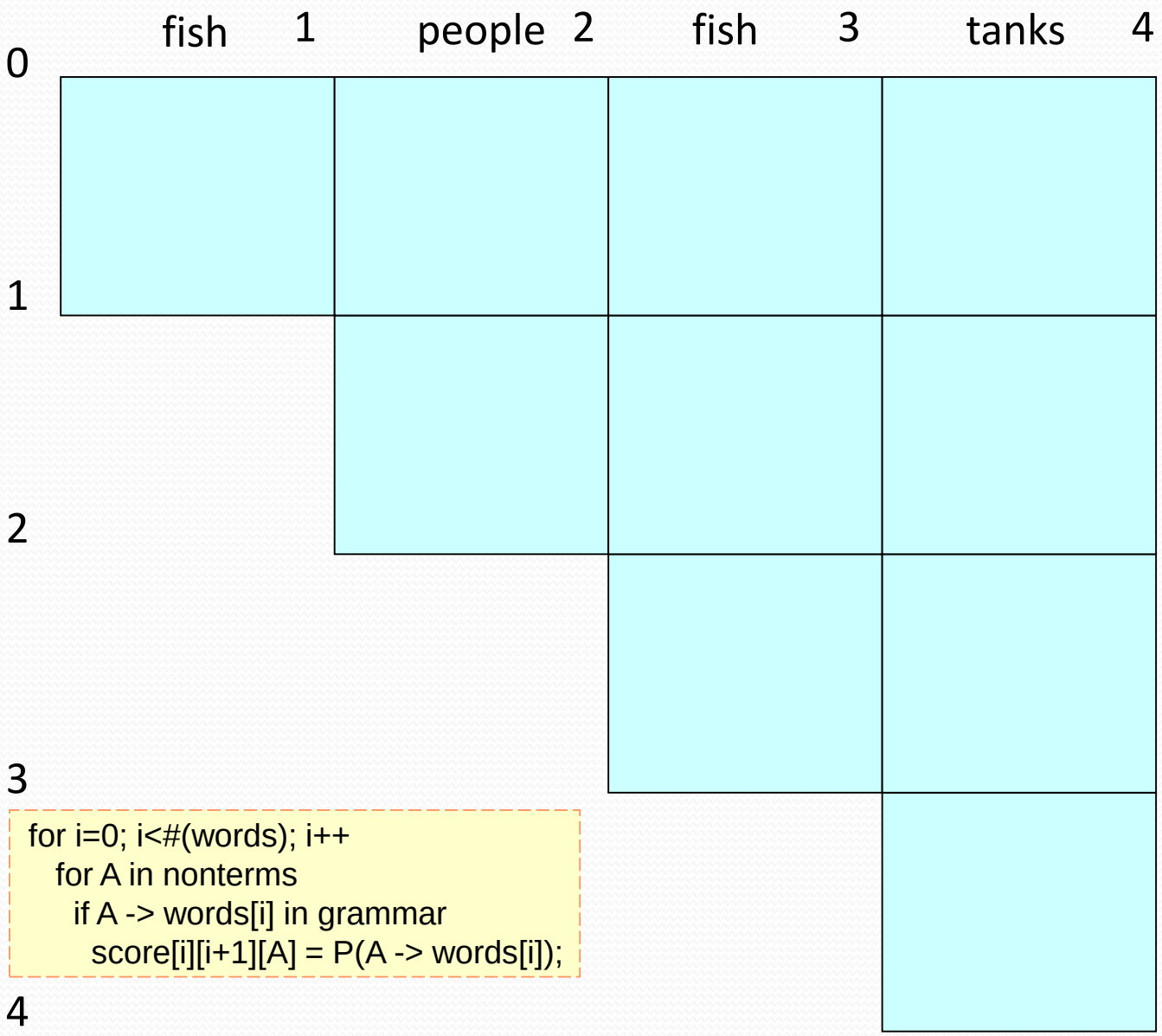
The grammar: Binary, no epsilons,

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]

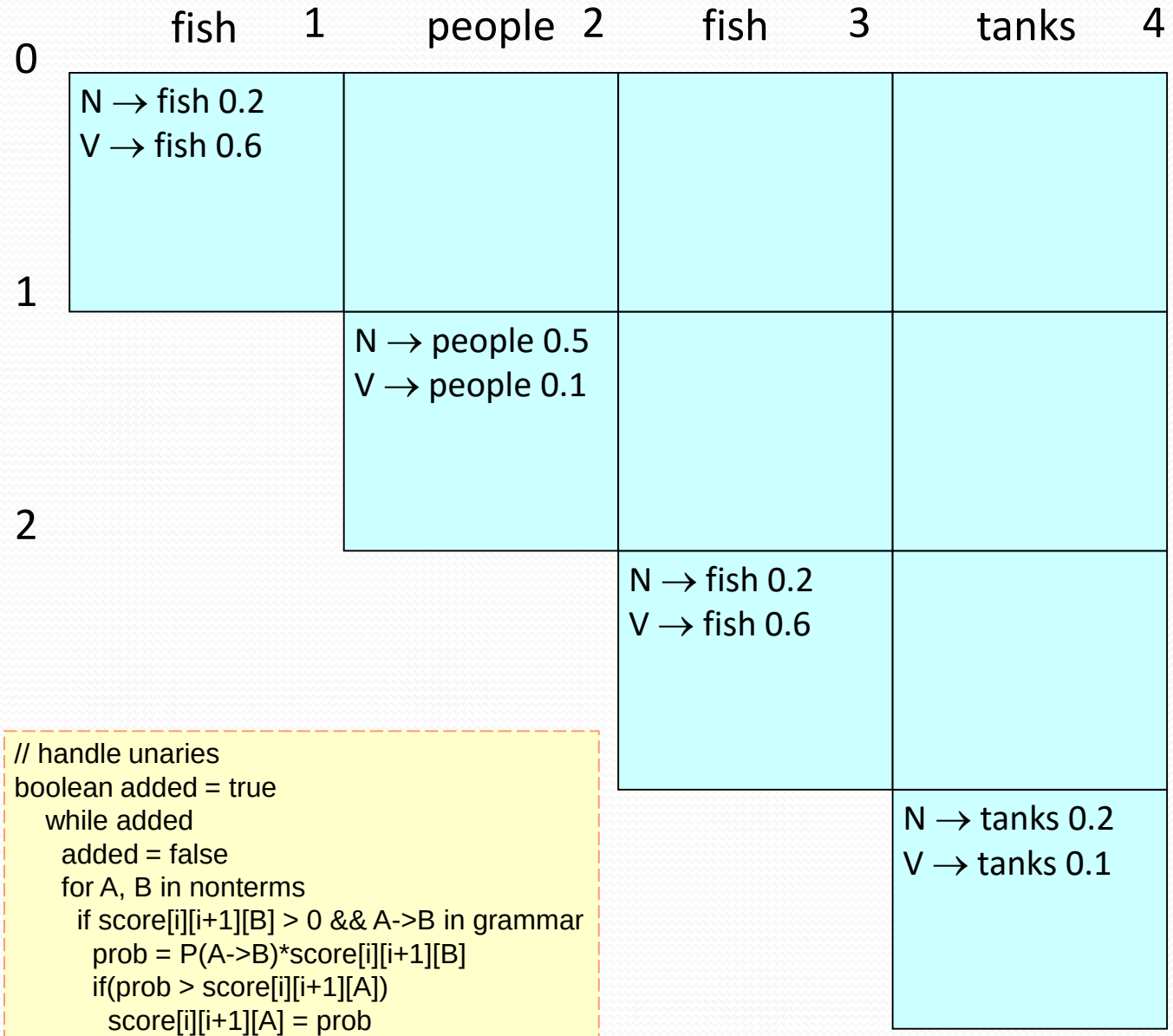
S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0




```
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

	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			
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001		
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

```
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 → 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	1
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	2
PP → P NP	1.0	
N → *people*	0.5	
N → *fish*	0.2	3
N → *tanks*	0.2	
N → *rods*	0.1	
V → *people*	0.1	
V → *fish*	0.6	4
V → *tanks*	0.3	
P → *with*	1.0	

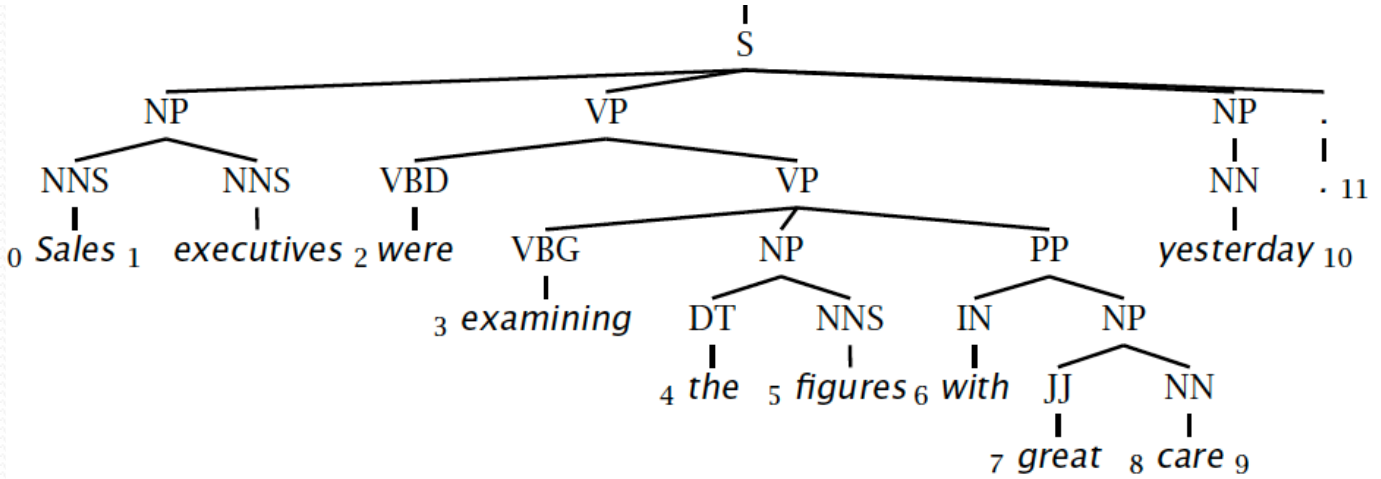
0 fish 1	people 2	fish 3	tanks 4
N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	NP → NP NP 0.0000009604 VP → V NP 0.00002058 S → NP VP 0.00018522
	N → people 0.5 V → people 0.1 NP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

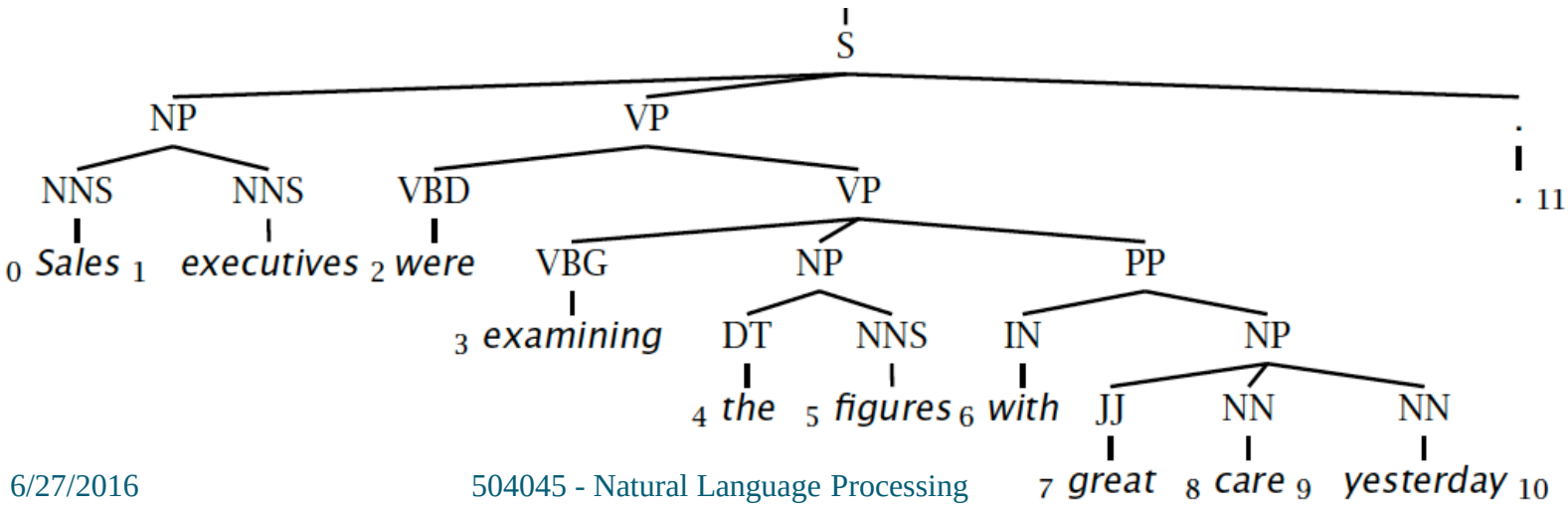
Constituency Parser Evaluation

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%

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

Lexicalization of PCFGs

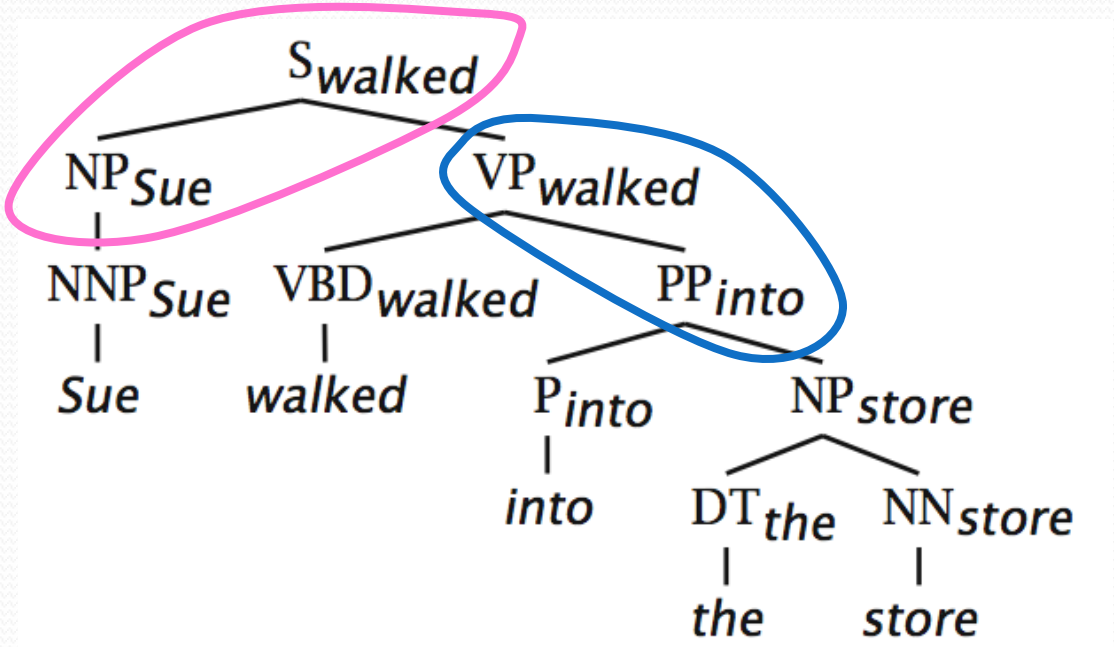
Introduction

Christopher Manning

(Head) Lexicalization of PCFGs

[Magerman 1995, Collins 1997; Charniak 1997]

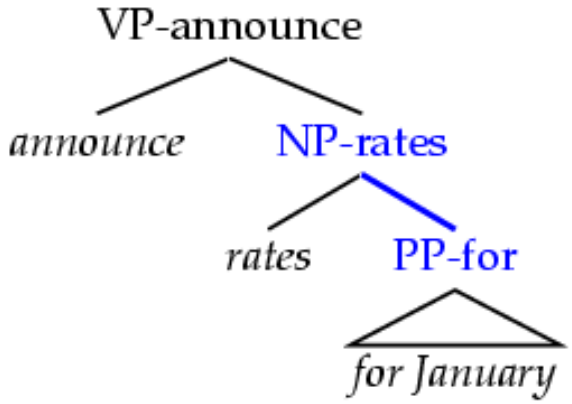
- The head word of a phrase gives a good representation of the phrase's structure and meaning
- Puts the properties of words back into a PCFG

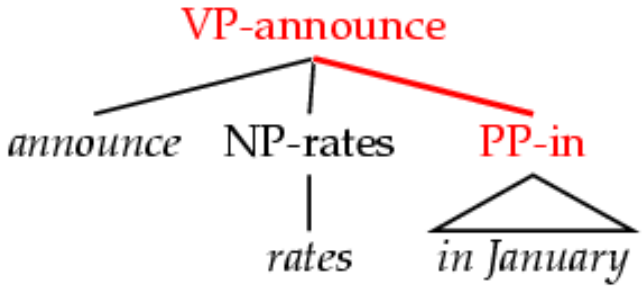


(Head) Lexicalization of PCFGs

[Magerman 1995, Collins 1997; Charniak 1997]

- Word-to-word affinities are useful for certain ambiguities
  - PP attachment is now (partly) captured in a local PCFG rule.
    - Think about: What useful information isn't captured?





  - Also useful for: coordination scope, verb complement patterns

Lexicalized parsing was seen as *the* parsing breakthrough of the late 1990s

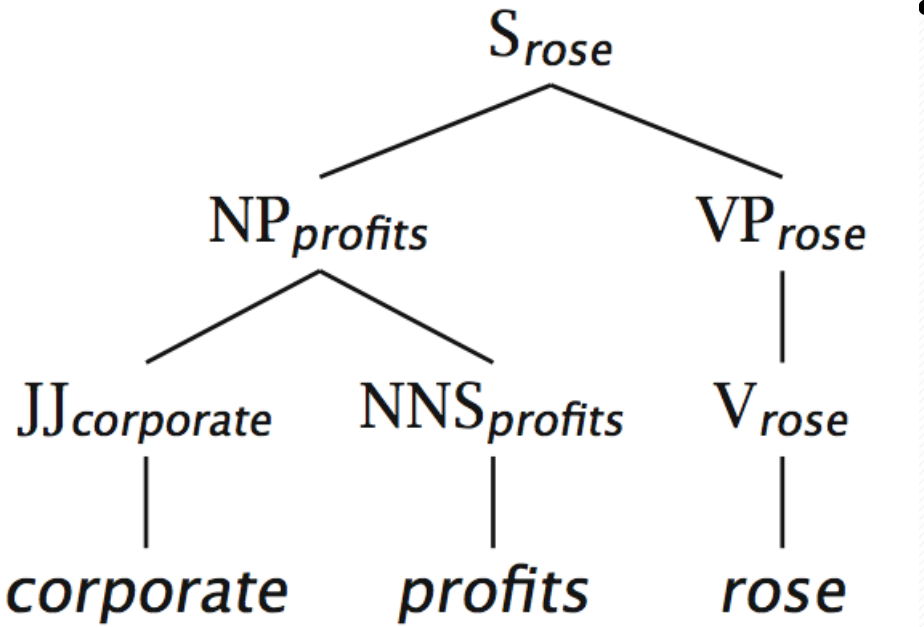
- Eugene Charniak, 2000 JHU workshop: "To do better, it is necessary to condition probabilities on the actual words of the sentence. This makes the probabilities much tighter:
  - $p(\text{VP} \rightarrow \text{V NP NP}) = 0.00151$
  - $p(\text{VP} \rightarrow \text{V NP NP} \mid \text{said}) = 0.00001$
  - $p(\text{VP} \rightarrow \text{V NP NP} \mid \text{gave}) = 0.01980$ "
- Michael Collins, 2003 COLT tutorial: "Lexicalized Probabilistic Context-Free Grammars … perform vastly better than PCFGs (88% vs. 73% accuracy)"

Lexicalization of PCFGs

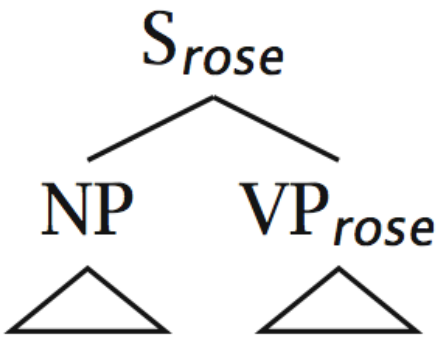
The model of Charniak (1997)

Charniak (1997)

- A very straightforward model of a lexicalized PCFG
- Probabilistic conditioning is "top-down" like a regular PCFG
  - But act[illegible] at like the CKY algorit[illegible]



Charniak (1997) example



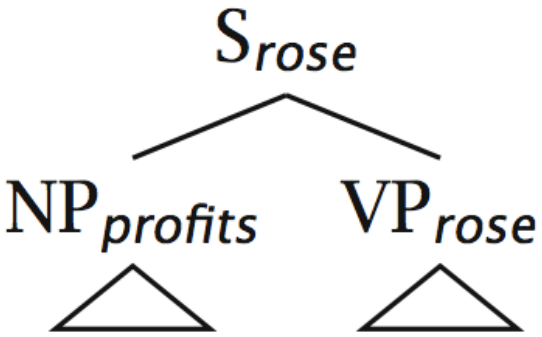
a. $h = profits$; $c = \text{NP}$

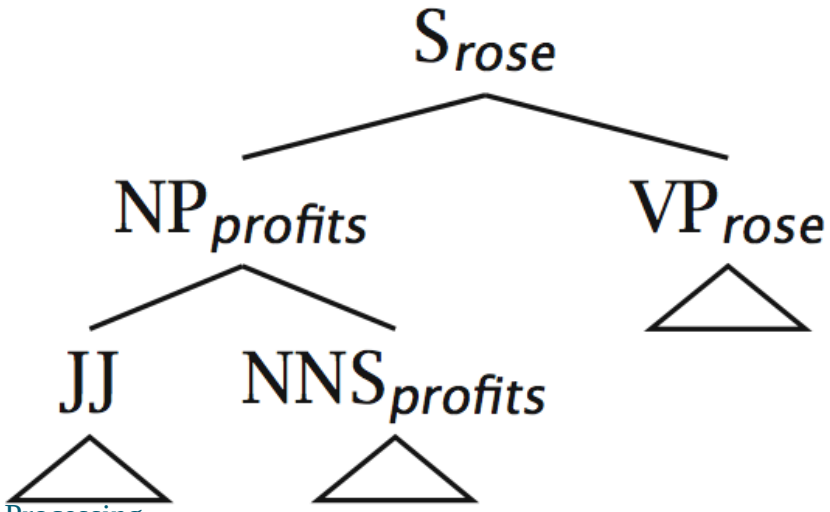
b. $ph = rose$; $pc = \text{S}$

c. $P(h|ph, c, pc)$

d. $P(r|h, c, pc)$

h: head word
r:rule





Lexicalization models argument selection by sharpening rule expansion probabilities

- The probability of different verbal complement frames (i.e., "subcategorizations") depends on the verb:

Local Tree	*come*	*take*	*think*	*want*
VP → V	9.5%	2.6%	4.6%	5.7%
VP → V NP	1.1%	32.1%	0.2%	13.9%
VP → V PP	34.5%	3.1%	7.1%	0.3%
VP → V SBAR	6.6%	0.3%	73.0%	0.2%
VP → V S	2.2%	1.3%	4.8%	70.8%
VP → V NP S	0.1%	5.7%	0.0%	0.3%
VP → V PRT NP	0.3%	5.8%	0.0%	0.0%
VP → V PRT PP	6.1%	1.5%	0.2%	0.0%

"monolexical" probabilities

Lexicalization sharpens probabilities: Predicting heads

"Bilexical probabilities"

- P(prices | n-plural) = .013
- P(prices | n-plural, NP) = .013
- P(prices | n-plural, NP, S) = .025
- P(prices | n-plural, NP, S, v-past) = .052
- P(**prices** | n-plural, NP, S, v-past, **fell**) = .146

Charniak (1997) linear interpolation/shrinkage

$$\begin{aligned}\hat{P}(h|ph,c,pc) &= \lambda_1(e)P_{\text{MLE}}(h|ph,c,pc) \\ &+\lambda_2(e)P_{\text{MLE}}(h|C(ph),c,pc) \\ &+\lambda_3(e)P_{\text{MLE}}(h|c,pc) + \lambda_4(e)P_{\text{MLE}}(h|c)\end{aligned}$$

- $\lambda_i(e)$ is here a function of how much one would expect to see a certain occurrence, given the amount of training data, word counts, etc.
- $C(ph)$ is semantic class of parent headword
- Techniques like these for dealing with data sparseness are vital to successful model construction

Charniak (1997) shrinkage example

	$P(\text{prft}\mid\text{rose}, \text{NP}, \text{S})$	$P(\text{corp}\mid\text{prft}, \text{JJ}, \text{NP})$
$P(h\mid ph, c, pc)$	0	0.245
$P(h\mid C(ph), c, pc)$	0.00352	0.0150
$P(h\mid c, pc)$	0.000627	0.00533
$P(h\mid c)$	0.000557	0.00418

- Allows utilization of rich highly conditioned estimates, but smoothes when sufficient data is unavailable
- One can't just use MLEs: one commonly sees previously unseen events, which would have probability 0.

Sparseness & the Penn Treebank

- The Penn Treebank – 1 million words of parsed English WSJ – has been a key resource (because of the widespread reliance on supervised learning)
- But 1 million words is like nothing:
  - 965,000 constituents, but only 66 WHADJP, of which only 6 aren't *how much* or *how many*, but there is an infinite space of these
    - *How clever/original/incompetent (at risk assessment and evaluation) …*
- Most of the probabilities that you would like to compute, you can't compute

Quiz question!

- Classify each of the italic red phrases as a:

  WHNP WHADJP WHADVP WHPP

1. That explains *why* she is succeeding.
2. *Which student* scored highest on the assignment?
3. Nobody knows *how deep* the recession will be.
4. *During which class* did the slide projection not work?
5. *Whose iPhone* was stolen?

Sparseness & the Penn Treebank (2)

- Many parse preferences depend on bilexical statistics: likelihoods of relationships between pairs of words (compound nouns, PP attachments, …)
- Extremely sparse, even on topics central to the WSJ:
  - *stocks plummeted* 2 occurrences
  - *stocks stabilized* 1 occurrence
  - *stocks skyrocketed* 0 occurrences
  - #*stocks discussed* 0 occurrences
- There has been only modest success in augmenting the Penn Treebank with extra unannotated materials or using semantic classes – given a reasonable amount of annotated training data.
  - Cf. Charniak 1997, Charniak 2000
  - But McClosky et al. 2006 doing self-training and Koo and Collins2008 semantic classes are rather more successful!

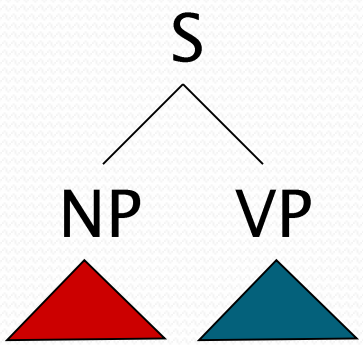
PCFG Independence Assumptions

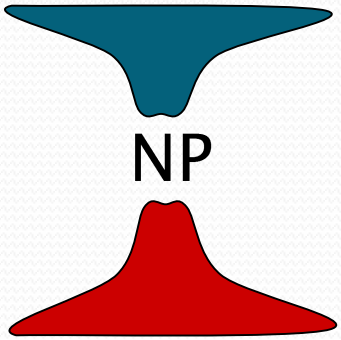
PCFGs and Independence

- The symbols in a PCFG define independence assumptions:

$S \rightarrow NP\ VP$

$NP \rightarrow DT\ NN$

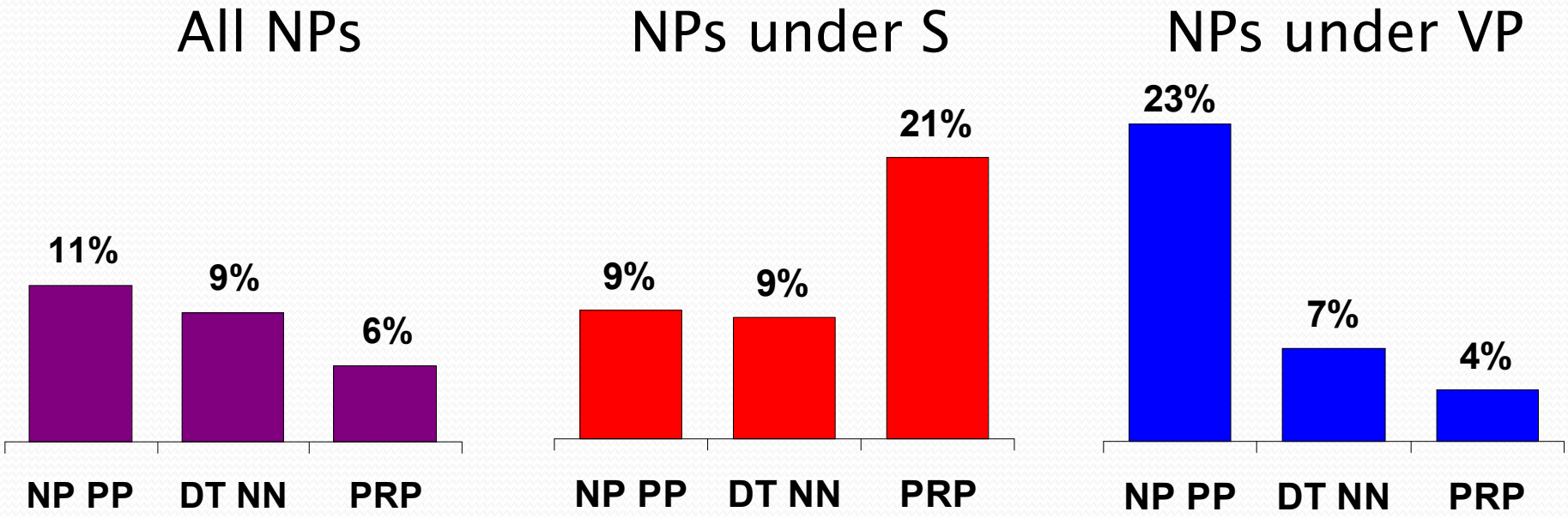




- At any node, the material inside that node is independent of the material outside that node, given the label of that node
- Any information that statistically connects behavior inside and outside a node must flow through that node's label

Non-Independence I

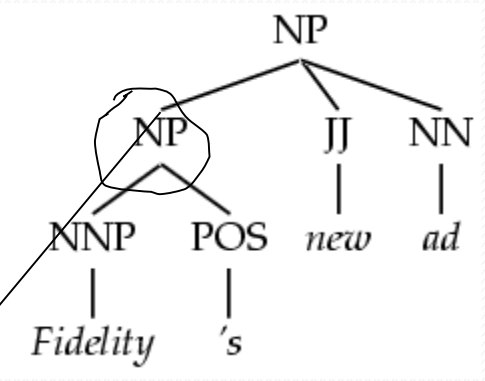
- The independence assumptions of a PCFG are often too strong

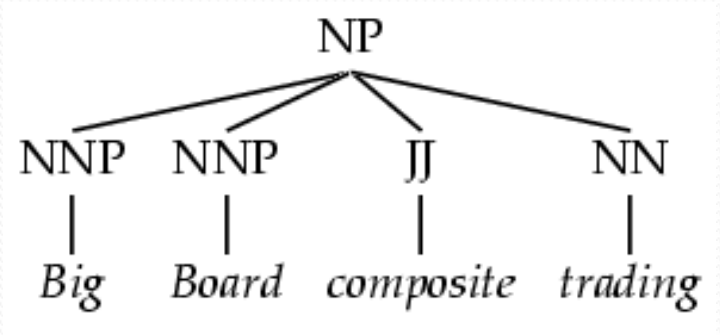


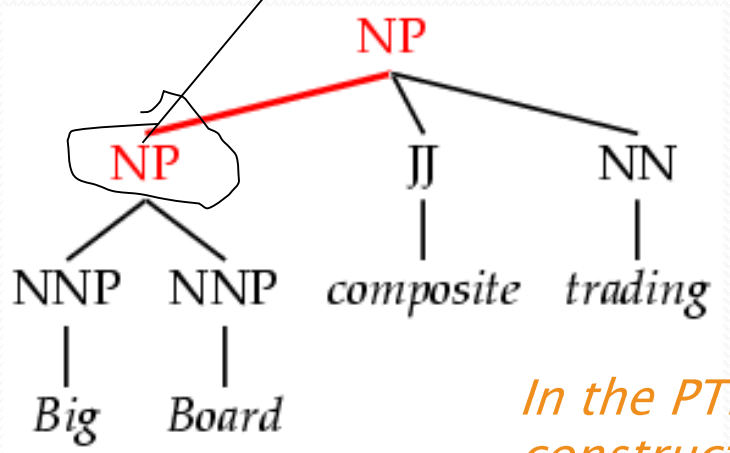
- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects)

Non-Independence II

- Symptoms of overly strong assumptions
  - Rewrites get used where they don't belo





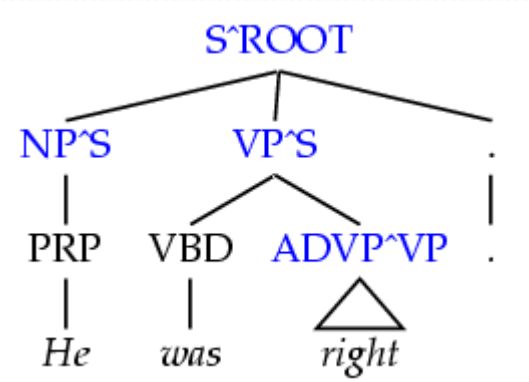


In the PTB, this construction is for possessives

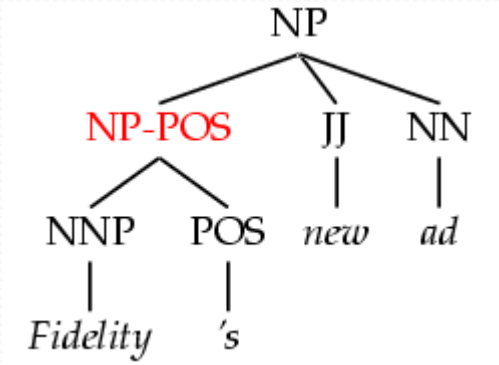
Refining the Grammar Symbols

- We can relax independence assumptions by encoding dependencies into the PCFG symbols, by state splitting:

Parent annotation
[Johnson 98]



Marking
possessive NPs



- Too much state-splitting ➔ sparseness (no smoothing used!)
- What are the most useful features to encode?

Annotations

- Annotations split the grammar categories into sub-categories.
- Conditioning on history vs. annotating
  - P(NP^S → PRP) is a lot like P(NP → PRP | S)
  - P(NP-POS → NNP POS) isn't history conditioning.
- Feature grammars vs. annotation
  - Can think of a symbol like NP^NP-POS as NP [parent:NP, +POS]
- After parsing with an annotated grammar, the annotations are then stripped for evaluation.

The Return of Unlexicalized PCFGs

Accurate Unlexicalized Parsing

[Klein and Manning 1993]

- What do we mean by an "unlexicalized" PCFG?
  - Grammar rules are not systematically specified down to the level of lexical items
    - NP-stocks is not allowed
    - NP^S-CC is fine
  - Closed vs. open class words
    - Long tradition in linguistics of using function words as features or markers for selection (VB-have, SBAR-if/whether)
    - Different to the bilexical idea of semantic heads
    - Open-class selection is really a proxy for semantics
- Thesis
  - Most of what you need for accurate parsing, and much of what lexicalized PCFGs actually capture *isn't* lexical selection between content words but just basic grammatical features, like verb form, finiteness, presence of a verbal auxiliary, etc.

Experimental Approach

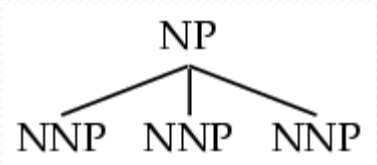
- Corpus: Penn Treebank, WSJ; iterate on small dev set

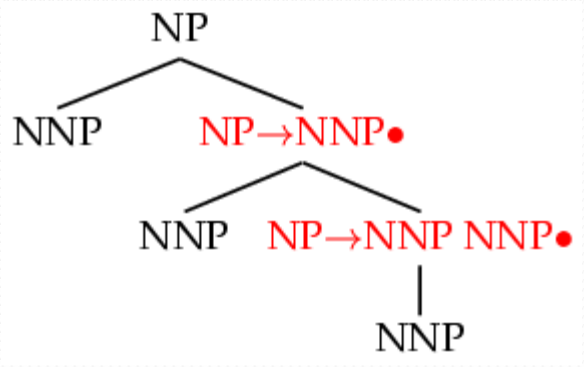
Training:	sections	02–21
Development:	section	22 (first 20 files) ←
Test:	section	23

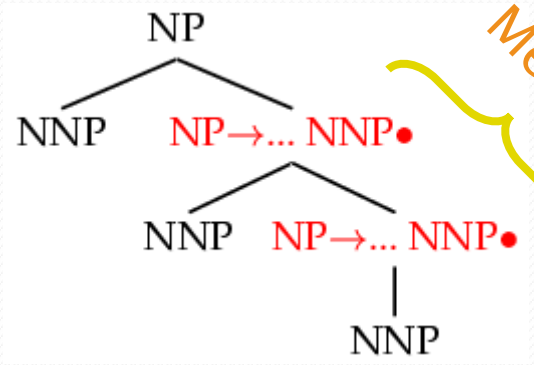
- Size – number of symbols in grammar.
  - Passive / complete symbols: NP, NP^S
  - Active / incomplete symbols: @NP_NP_CC [from binarization]
- We state-split as sparingly as possible
  - Highest accuracy with fewest symbols
  - Error-driven, manual hill-climb, one annotation at a time

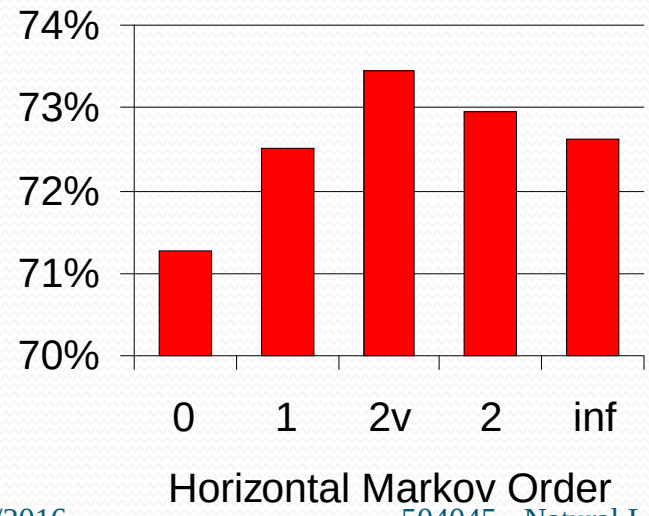
Horizontal Markovization

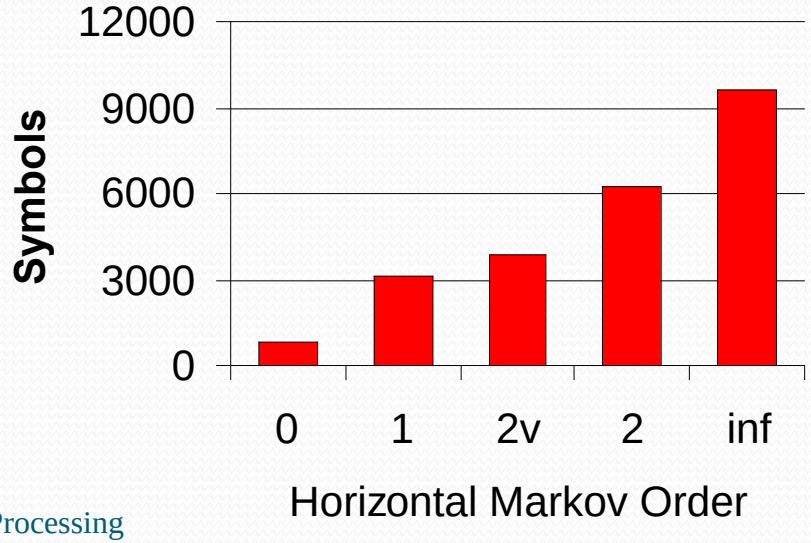
- Horizontal Markovization: Merges States







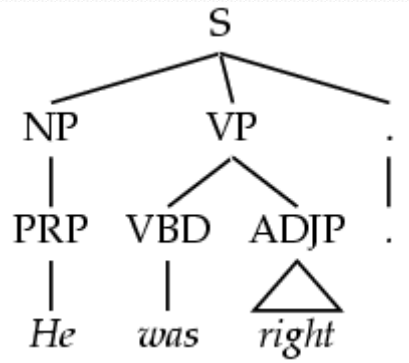




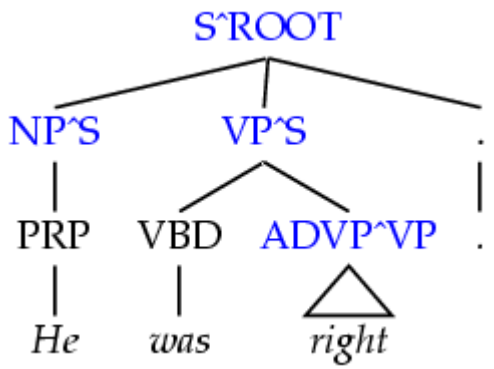
Vertical Markovization

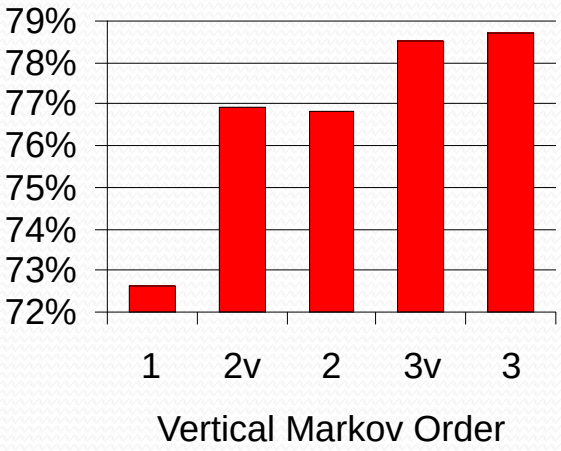
- Vertical Markov order: rewrites depend on past *k* ancestor nodes. (i.e., parent annotation)

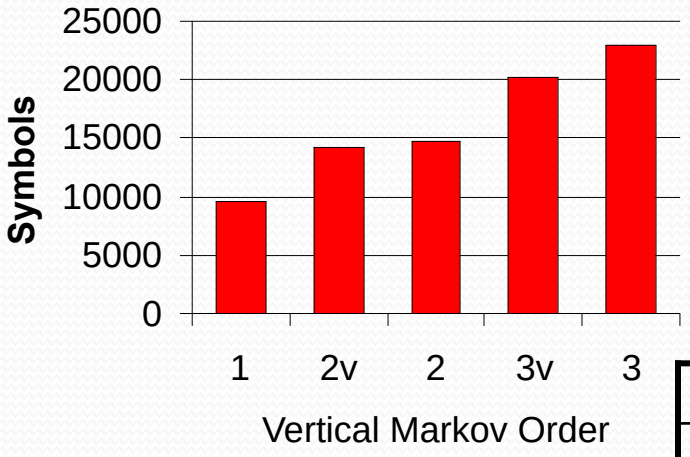
Order 1



Order 2



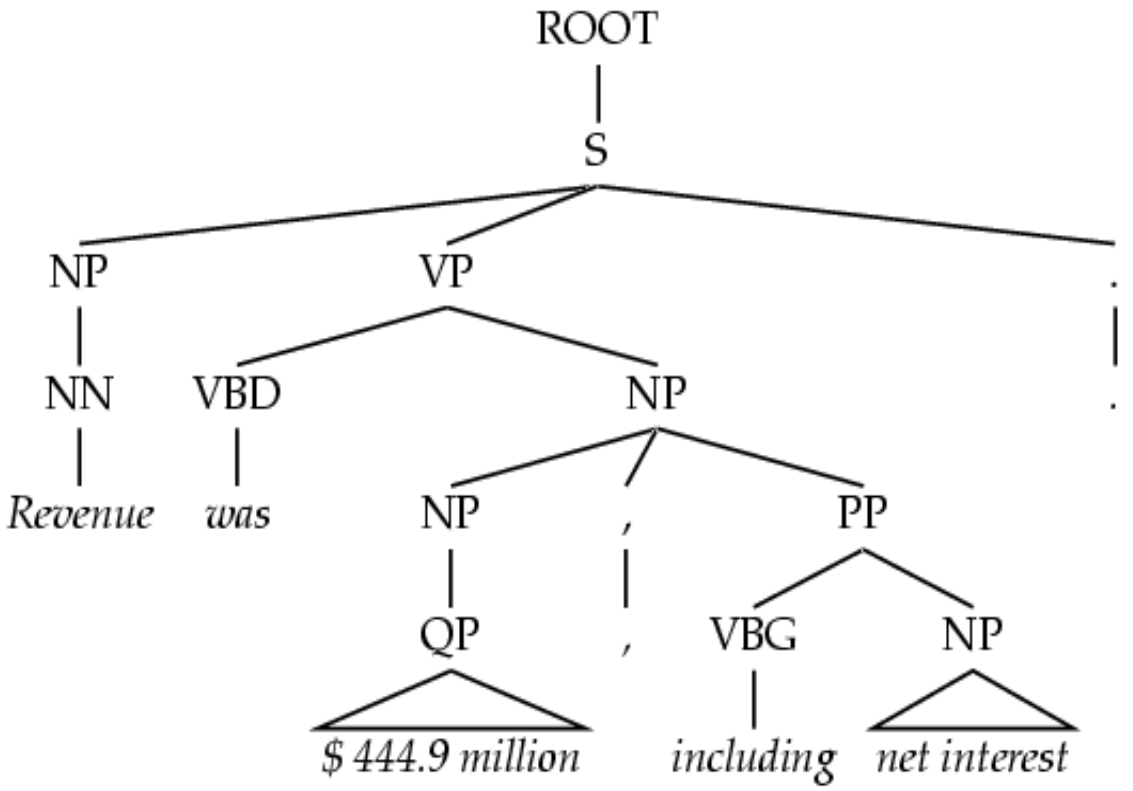




Model	F1	Size
v=h=2v	77.8	7.5K

Unary Splits

- Problem: unary rewrites are used to transmute categories so a high-probability rule can be used.

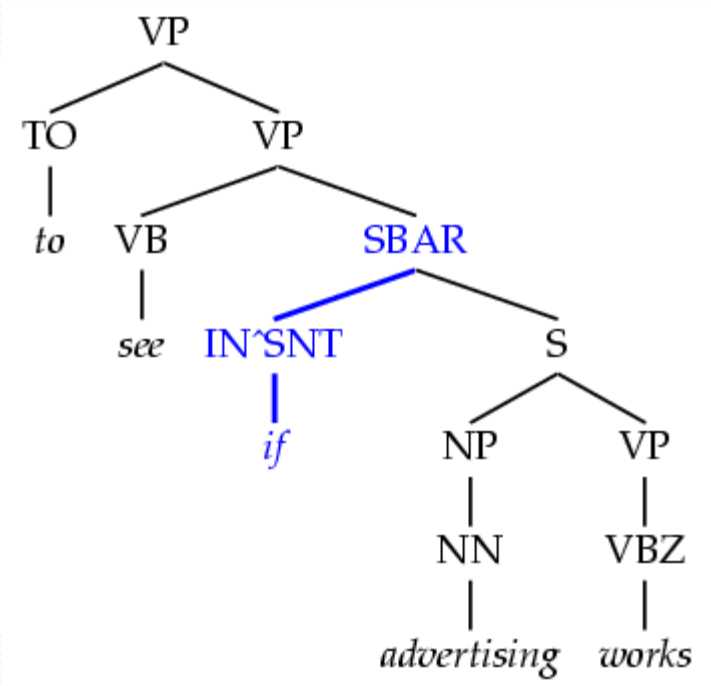


- Solution: Mark unary rewrite sites with -U

Annotation	F1	Size
Base	77.8	7.5K
UNARY	78.3	8.0K

Tag Splits

- Problem: Treebank tags are too coarse.
- Example: SBAR sentential complementizers (*that, whether, if)*, subordinating conjunctions (*while, after)*, and true prepositions (*in, of, to)* are all tagged IN.
- Partial Solution:
  - Subdivide the IN tag.



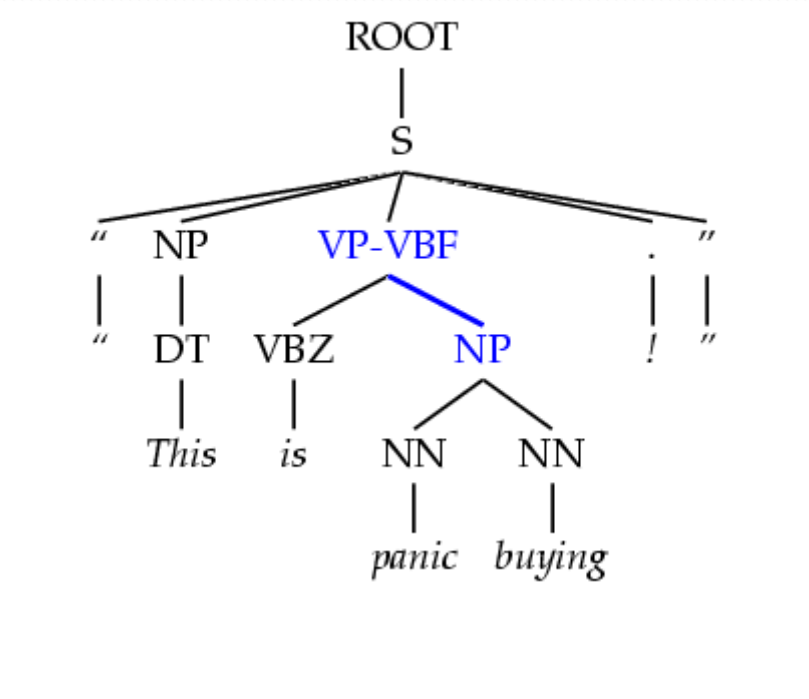
Annotation	F1	Size
Previous	78.3	8.0K
SPLIT–IN	80.3	8.1K

Yield Splits

- Problem: sometimes the behavior of a category depends on something inside its future yield.

- Examples:
  - Possessive NPs
  - Finite vs. infinite VPs
  - Lexical heads!

- Solution: annotate future elements into nodes.

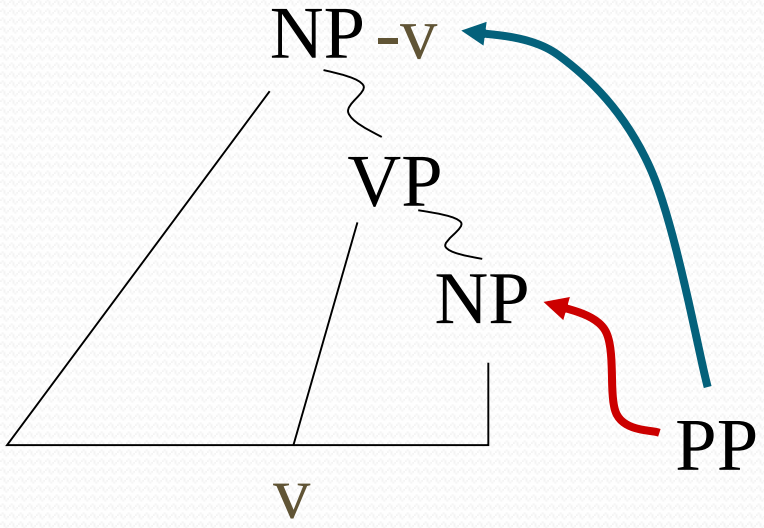


Annotation	F1	Size
tag splits	82.3	9.7K
POSS-NP	83.1	9.8K
SPLIT-VP	85.7	10.5K

Distance / Recursion Splits

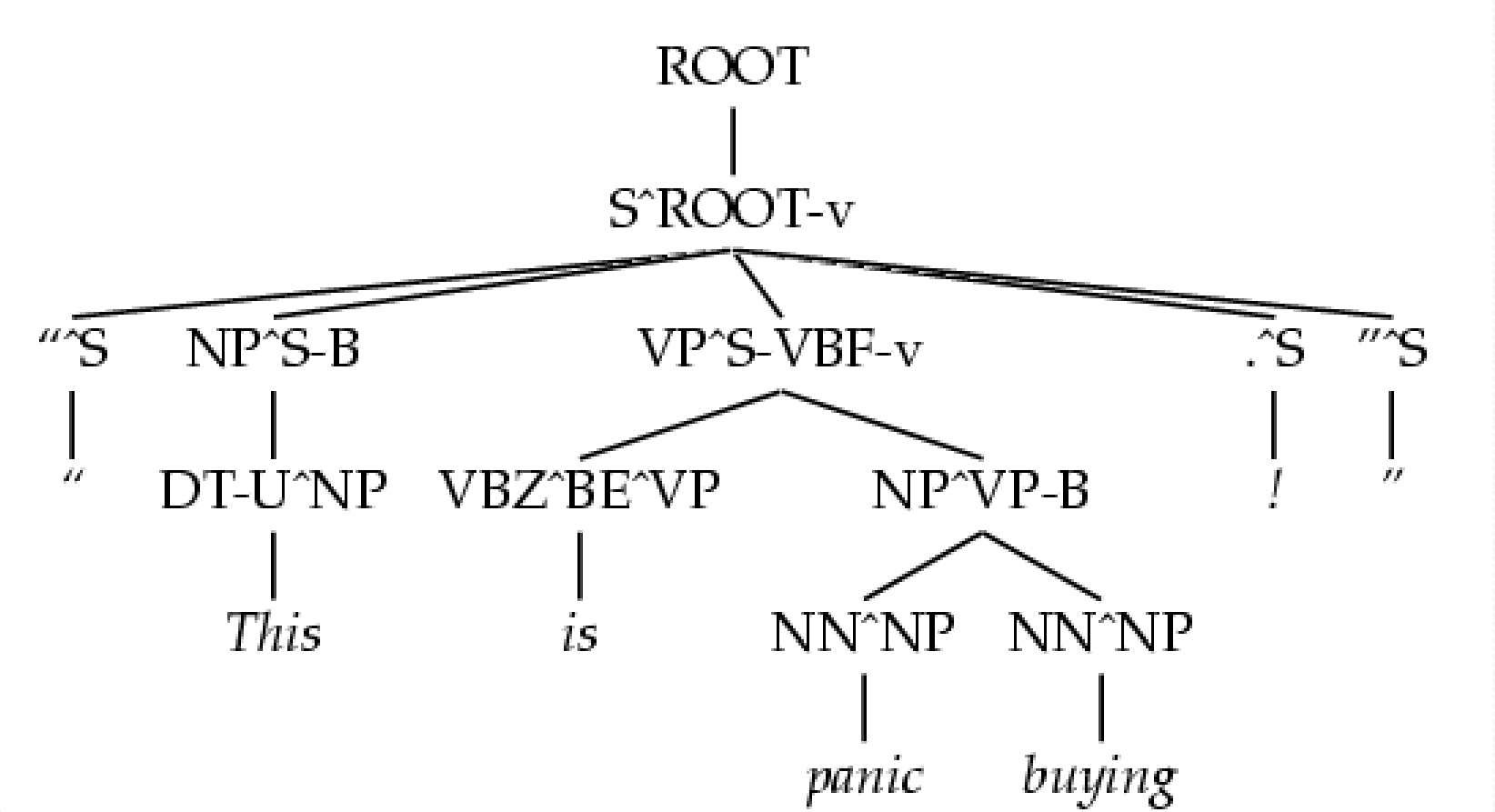
- Problem: vanilla PCFGs cannot distinguish attachment heights.

- Solution: mark a property of higher or lower sites:
  - Contains a verb.
  - Is (non)-recursive.
    - Base NPs [cf. Collins 99]
    - Right-recursive NPs



Annotation	F1	Size
Previous	85.7	10.5K
BASE-NP	86.0	11.7K
DOMINATES-V	86.9	14.1K
RIGHT-REC-NP	87.0	15.2K

A Fully Annotated Tree



Final Test Set Results

Parser	LP	LR	**F1**
Magerman 95	84.9	84.6	**84.7**
Collins 96	86.3	85.8	**86.0**
Klein & Manning 03	86.9	85.7	**86.3**
Charniak 97	87.4	87.5	**87.4**
Collins 99	88.7	88.6	**88.6**

- Beats “first generation” lexicalized parsers

Latent Variable PCFGs

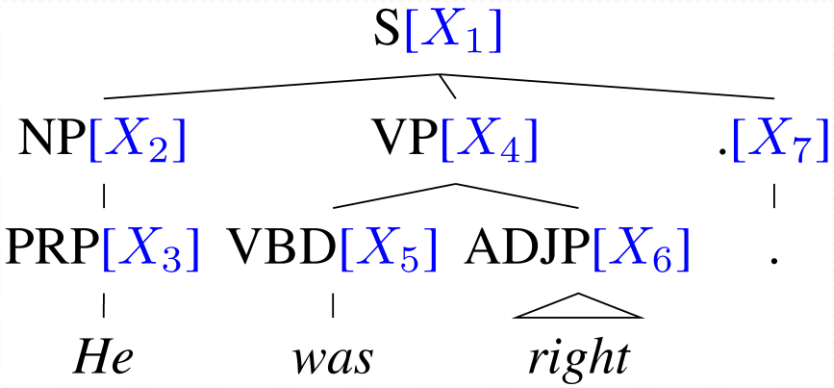
Extending the idea to induced syntactico-semantic classes

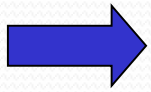
Learning Latent Annotations

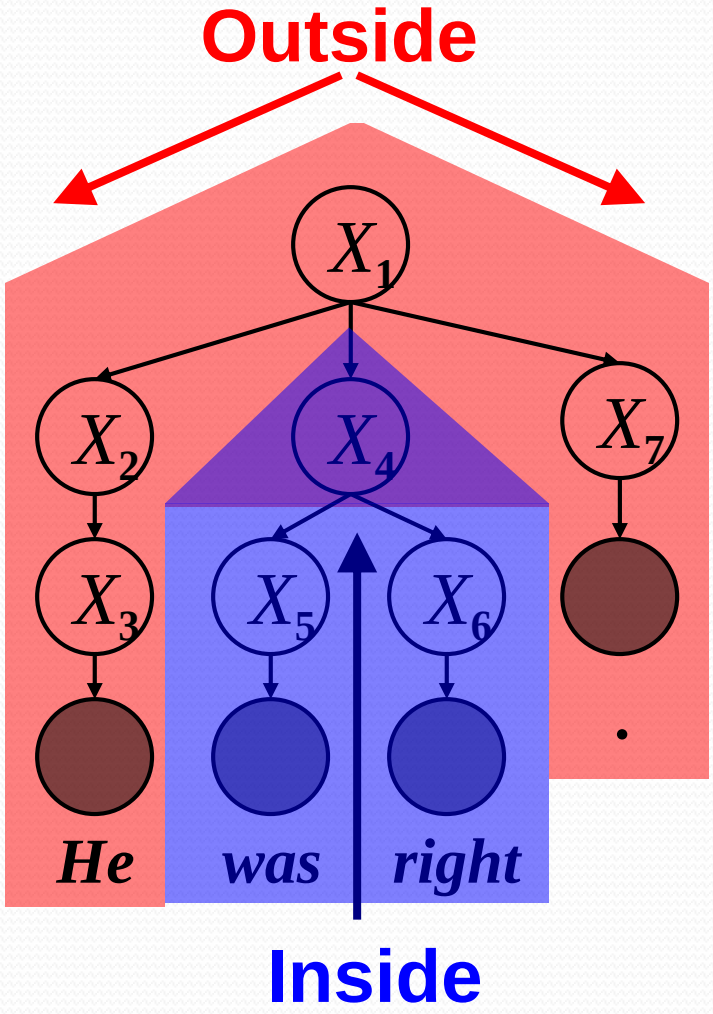
[Petrov and Klein 2006, 2007]

Can you automatically find good symbols?

- Brackets are known
- Base categories are known
- Induce subcategories
- Clever split/merge category refinement







EM algorithm, like Forward-Backward for HMMs, but constrained by tree

POS tag splits' commonest words: effectively a semantic class-based model

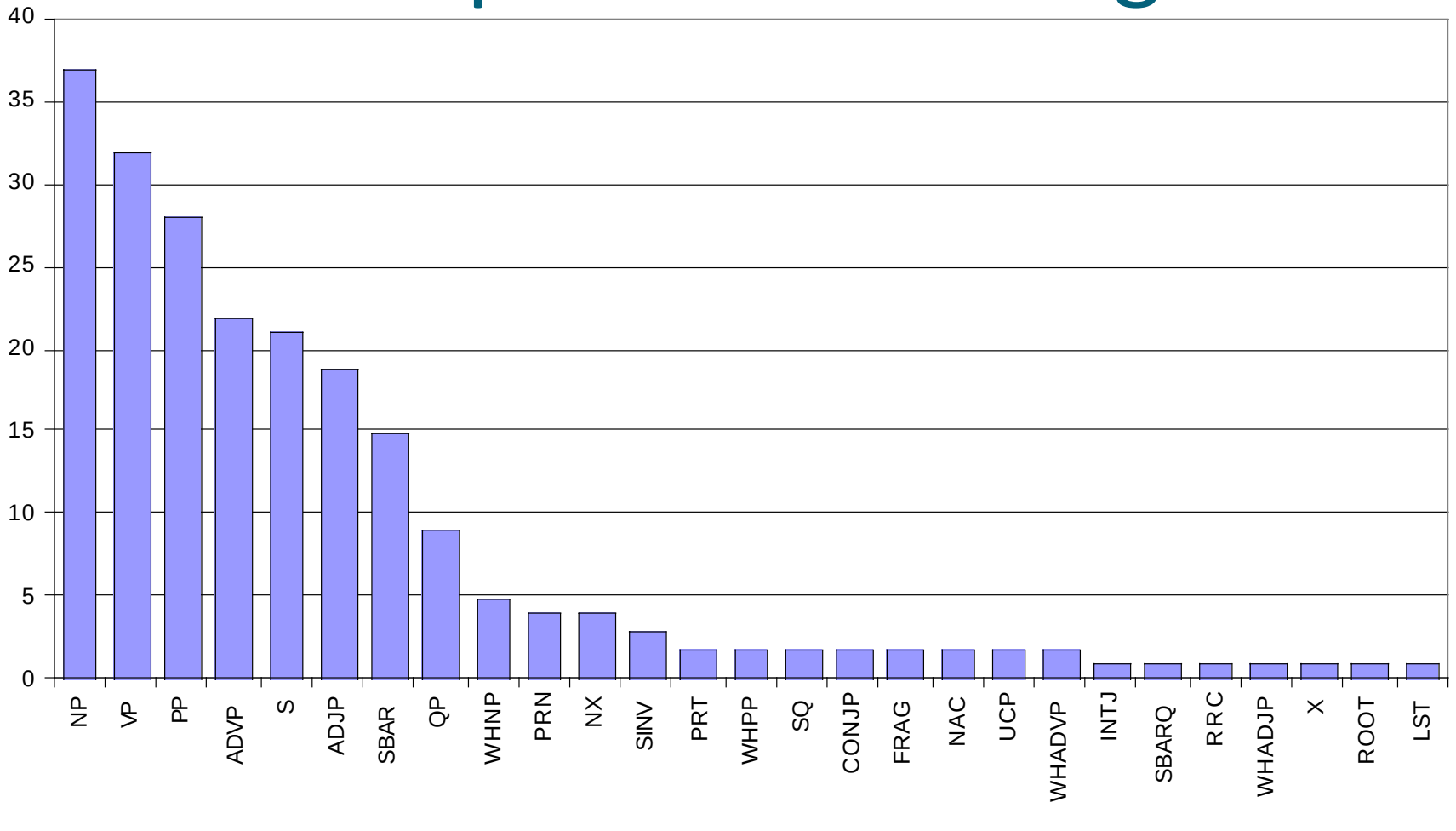
- Proper Nouns (NNP):

NNP-14	Oct.	Nov.	Sept.
NNP-12	John	Robert	James
NNP-2	J.	E.	L.
NNP-1	Bush	Noriega	Peters
NNP-15	New	San	Wall
NNP-3	York	Francisco	Street

- Personal pronouns (PRP):

PRP-0	It	He	I
PRP-1	it	he	they
PRP-2	it	them	him

Number of phrasal subcategories



The Latest Parsing Results...

(English PTB3 WSJ train 2-21, test 23)

Parser	*F1* ≤ *40 words*	*F1* *all words*
Klein & Manning unlexicalized 2003	86.3	85.7
Matsuzaki et al. simple EM latent states 2005	86.7	86.1
Charniak generative, lexicalized ("maxent inspired") 2000	90.1	89.5
Petrov and Klein NAACL 2007	90.6	90.1
Charniak & Johnson discriminative reranker 2005	92.0	91.4
Fossum & Knight 2009 combining constituent parsers		**92.4**

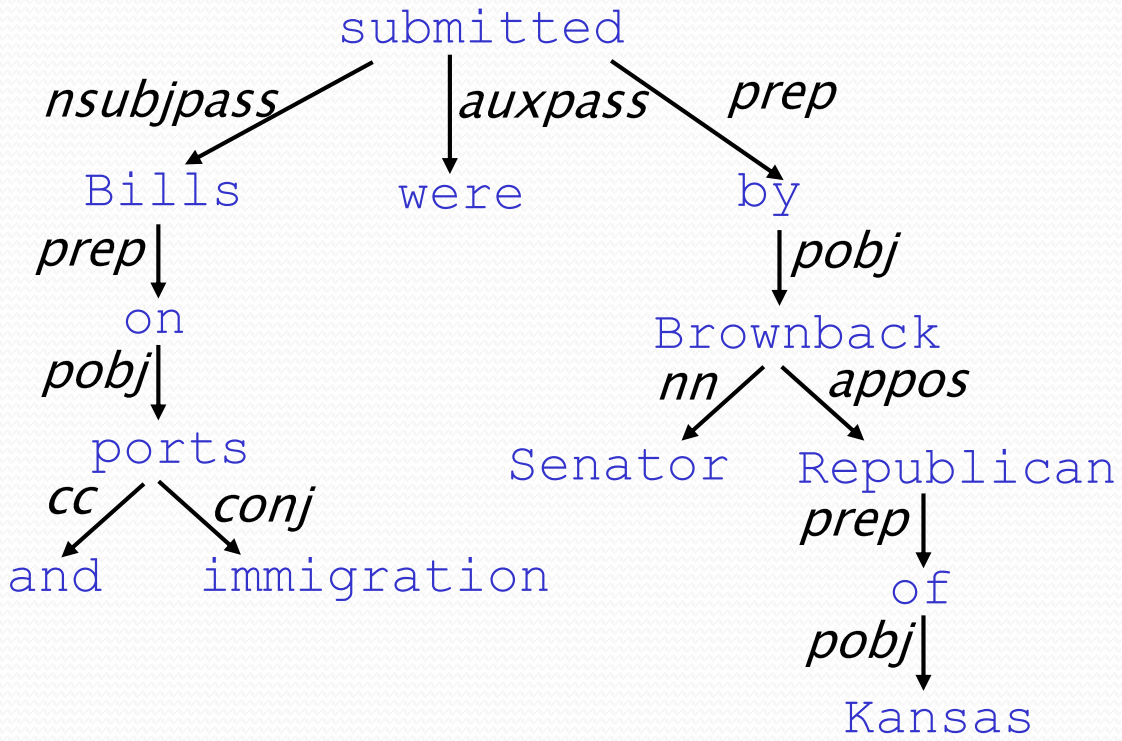
Dependency Parsing

Introduction

Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc.)

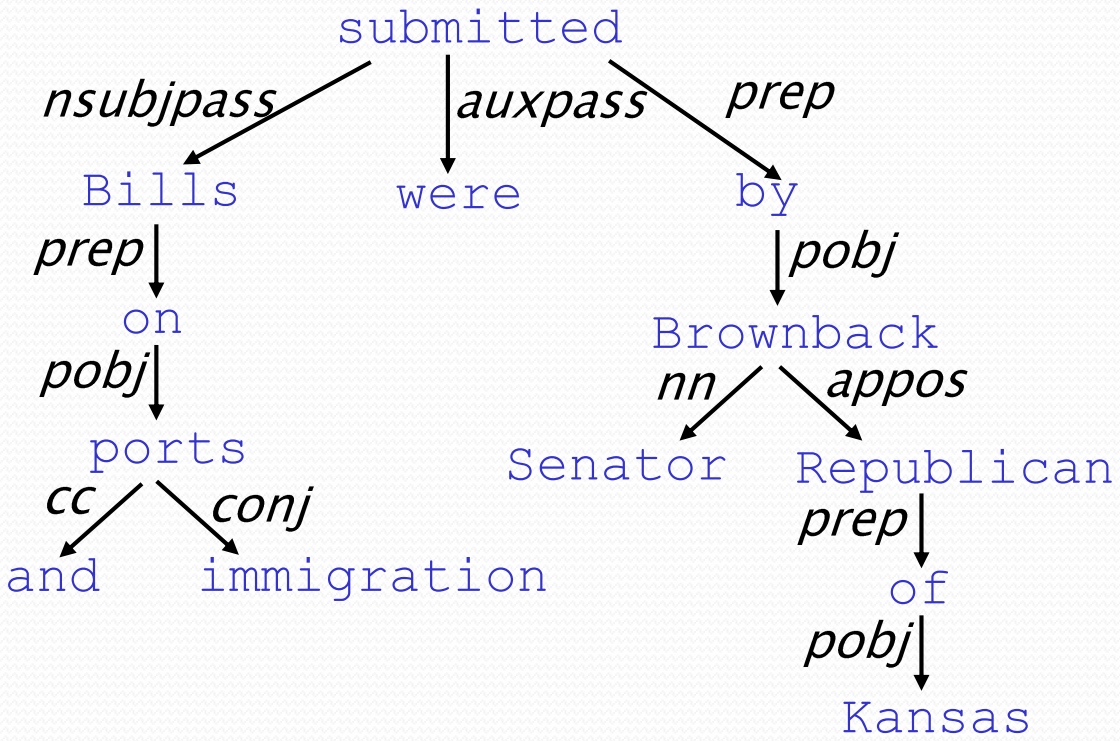


Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)

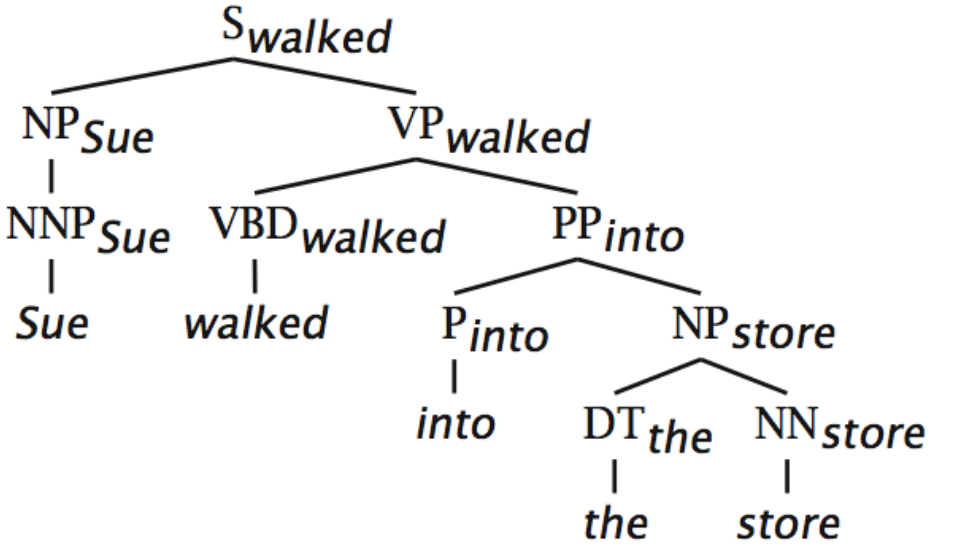
Usually, dependencies form a tree (connected, acyclic, single-head)



structure and dependency structure

- A dependency grammar has a notion of a head. Officially, CFGs don't.
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, …) do, via hand-written phrasal "head rules":
  - The head of a Noun Phrase is a noun/number/adj/…
  - The head of a Verb Phrase is a verb/modal/….
- The head rules can be used to extract a dependency parse from a CFG parse

- The closure of dependencies give constituency from a dependency tree
- But the dependents of a word must be at the same level (i.e., "flat") – there can be no VP!



Methods of Dependency Parsing

1. Dynamic programming (like in the CKY algorithm)

   You can do it similarly to lexicalized PCFG parsing: an $O(n^5)$ algorithm

   Eisner (1996) gives a clever algorithm that reduces the complexity to $O(n^3)$, by producing parse items with heads at the ends rather than in the middle

2. Graph algorithms

   You create a Maximum Spanning Tree for a sentence

   McDonald et al.'s (2005) MSTParser scores dependencies independently using a ML classifier (he uses MIRA, for online learning, but it could be MaxEnt)

3. Constraint Satisfaction

   Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.

4. "Deterministic parsing"

   Greedy choice of attachments guided by machine learning classifiers

   MaltParser (Nivre et al. 2008) – discussed in the next segment

Dependency Conditioning Preferences

What are the sources of information for dependency parsing?

1. Bilexical affinities [issues → the] is plausible
2. Dependency distance mostly with nearby words
3. Intervening material
   Dependencies rarely span intervening verbs or punctuation
4. Valency of heads
   How many dependents on which side are usual for a head?

Quiz question!

- Consider this sentence:

Retail sales drop in April cools afternoon market trading.

- Which word are these words a dependent of?
  1. sales
  2. April
  3. afternoon
  4. trading

Greedy Transition-Based Parsing

MaltParser

MaltParser

[Nivre et al. 2008]

- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom up actions
  - Roughly like "shift" or "reduce" in a shift-reduce parser, but the "reduce" actions are specialized to create dependencies with head on left or right
- The parser has:
  - a stack σ, written with top to the right
    - which starts with the ROOT symbol
  - a buffer β, written with top to the left
    - which starts with the input sentence
  - a set of dependency arcs A
    - which starts off empty
  - a set of actions

Basic transition-based dependency parser

Start: $\sigma = [\text{ROOT}], \beta = w_1, \ldots, w_n, A = \emptyset$

1. Shift $\quad \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\quad \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\}$
3. Right-Arc$_r$ $\quad \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_i|\beta, A \cup \{r(w_i, w_j)\}$

Finish: $\beta = \emptyset$

Notes:

- Unlike the regular presentation of the CFG reduce step, dependencies combine one thing from each of stack and buffer

Actions ("arc-eager" dependency parser)

Start: $\sigma = [\text{ROOT}], \beta = w_1, \ldots, w_n, A = \emptyset$

1. Left-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\}$
   Precondition: $r'(w_k, w_i) \notin A, w_i \neq \text{ROOT}$
2. Right-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma|w_i|w_j, \beta, A \cup \{r(w_i, w_j)\}$
3. Reduce $\sigma|w_i, \beta, A \rightarrow \sigma, \beta, A$
   Precondition: $r'(w_k, w_i) \in A$
4. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$

Finish: $\beta = \emptyset$

This is the common "arc-eager" variant: a head can immediately take a right dependent, before *its* dependents are found

Example

1. Left-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\}$
   Precondition: $(w_k, r', w_i) \notin A$, $w_i \neq$ ROOT
2. Right-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma|w_i|w_j, \beta, A \cup \{r(w_i, w_j)\}$
3. Reduce $\sigma|w_i, \beta, A \rightarrow \sigma, \beta, A$
   Precondition: $(w_k, r', w_i) \in A$
4. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$

Happy children like to play with their friends .

	[ROOT]	[Happy, children, ...]	$\emptyset$
Shift	[ROOT, Happy]	[children, like, ...]	$\emptyset$
LA_{amod}	[ROOT]	[children, like, ...]	{amod(children, happy)} = A_1
Shift	[ROOT, children]	[like, to, ...]	A_1
LA_{nsubj}	[ROOT]	[like, to, ...]	$A_1 \cup$ {nsubj(like, children)} = A_2
RA_{root}	[ROOT, like]	[to, play, ...]	$A_2 \cup$ {root(ROOT, like) = A_3
Shift	[ROOT, like, to]	[play, with, ...]	A_3
LA_{aux}	[ROOT, like]	[play, with, ...]	$A_3 \cup$ {aux(play, to) = A_4
RA_{xcomp}	[ROOT, like, play]	[with their, ...]	$A_4 \cup$ {xcomp(like, play) = A_5

1. Left-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\}$
   Precondition: $(w_k, r', w_i) \notin A$, $w_i \neq$ ROOT
2. Right-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma|w_i|w_j, \beta, A \cup \{r(w_i, w_j)\}$
3. Reduce $\sigma|w_i, \beta, A \rightarrow \sigma, \beta, A$
   Precondition: $(w_k, r', w_i) \in A$
4. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$

Example

Happy children like to play with their friends .

RA_{xcomp}	[ROOT, like, play]	[with their, ...]	$A_4 \cup \{xcomp(like, play) = A_5$
RA_{prep}	[ROOT, like, play, with]	[their, friends, ...]	$A_5 \cup \{prep(play, with) = A_6$
Shift	[ROOT, like, play, with, their]	[friends, .]	A_6
LA_{poss}	[ROOT, like, play, with]	[friends, .]	$A_6 \cup \{poss(friends, their) = A_7$
RA_{pobj}	[ROOT, like, play, with, friends]	[.]	$A_7 \cup \{pobj(with, friends) = A_8$
Reduce	[ROOT, like, play, with]	[.]	A_8
Reduce	[ROOT, like, play]	[.]	A_8
Reduce	[ROOT, like]	[.]	A_8
RA_{punc}	[ROOT, like, .]	[]	$A_8 \cup \{punc(like, .) = A_9$

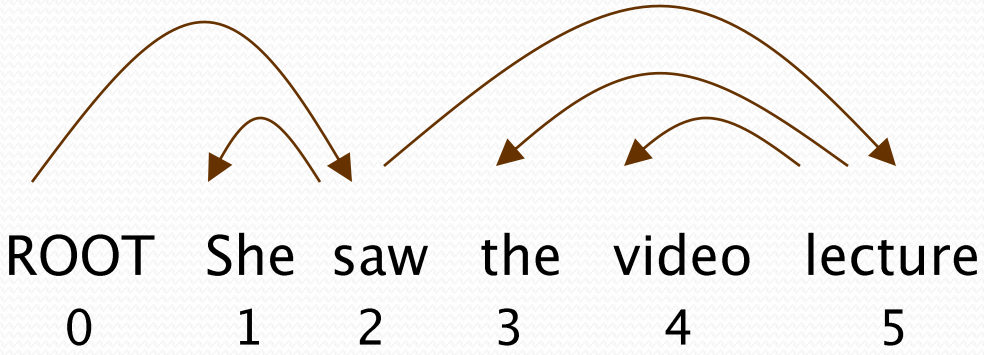
You terminate as soon as the buffer is empty. Dependencies = A_9

MaltParser

[Nivre et al. 2008]

- We have left to explain how we choose the next action
- Each action is predicted by a discriminative classifier (often SVM, could be maxent classifier) over each legal move
  - Max of 4 untyped choices, max of |R| × 2 + 2 when typed
  - Features: top of stack word, POS; first in buffer word, POS; etc.
- There is NO search (in the simplest and usual form)
  - But you could do some kind of beam search if you wish
- The model's accuracy is *slightly* below the best LPCFGs (evaluated on dependencies), but
- It provides close to state of the art parsing performance
- It provides VERY fast linear time parsing

Evaluation of Dependency Parsing: (labeled) dependency accuracy



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

UAS = 4 / 5 = 80%

LAS = 2 / 5 = 40%

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	dobj

Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

Representative performance numbers

- The CoNLL-X (2006) shared task provides evaluation numbers for various dependency parsing approaches over 13 languages
  - MALT: LAS scores from 65–92%, depending greatly on language/treebank
-

Parser	UAS%
Sagae and Lavie (2006) ensemble of dependency parsers	92.7
Charniak (2000) generative, constituency	92.2
Collins (1999) generative, constituency	91.7
McDonald and Pereira (2005) – MST graph-based dependency	91.5
Yamada and Matsumoto (2003) – transition-based dependency	90.4

Projectivity

- Dependencies from a CFG tree using heads, must be projective
  - There must not be any crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words.
- But dependency theory normally does allow non-projective structures to account for displaced constituents
  - You can't easily get the semantics of certain constructions right without these nonprojective dependencies

Handling non-projectivity

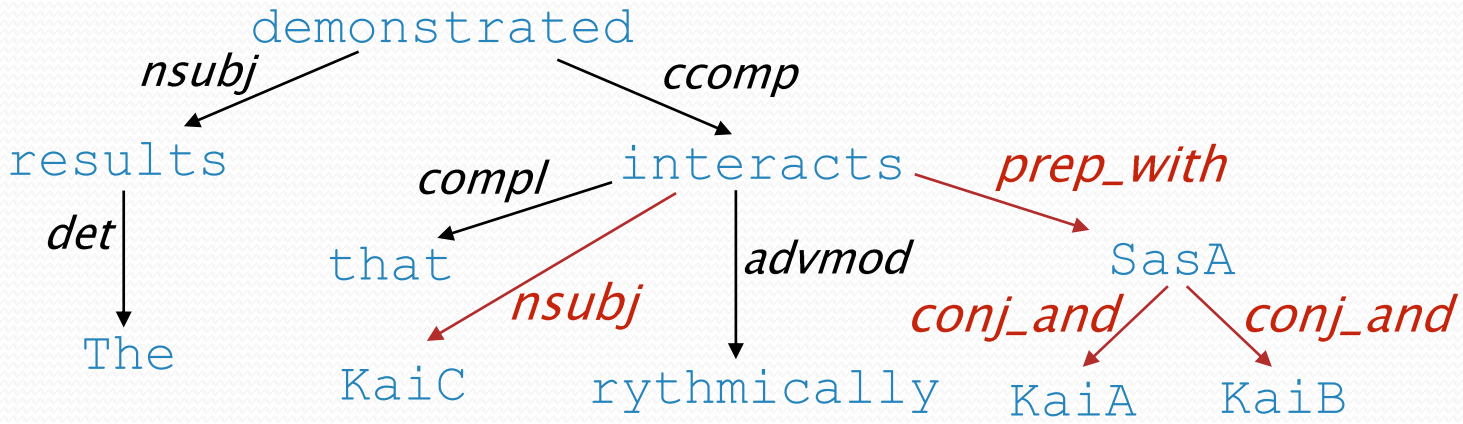
- The arc-eager algorithm we presented only builds projective dependency trees
- Possible directions to head:
  1. Just declare defeat on nonprojective arcs
  2. Use a dependency formalism which only admits projective representations (a CFG doesn't represent such structures…)
  3. Use a postprocessor to a projective dependency parsing algorithm to identify and resolve nonprojective links
  4. Add extra types of transitions that can model at least most non-projective structures
  5. Move to a parsing mechanism that does not use or require any constraints on projectivity (e.g., the graph-based MSTParser)

Dependencies encode relational structure

Relation Extraction with Stanford Dependencies

Dependency paths identify relations like protein interaction

[Erkan et al. EMNLP 07, Fundel et al. 2007]

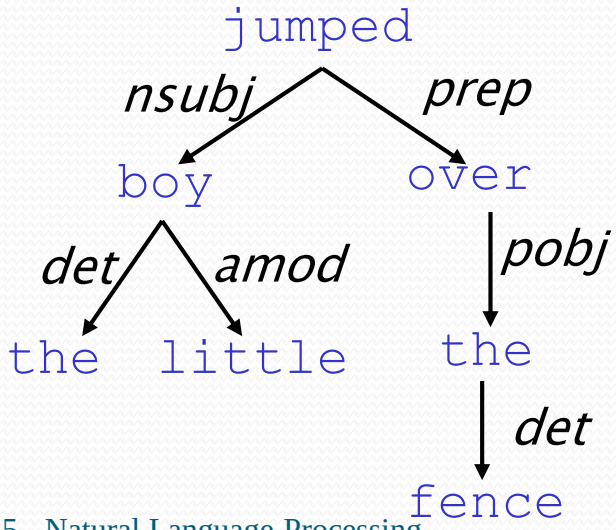


KaiC ←nsubj interacts prep_with→ SasA
KaiC ←nsubj interacts prep_with→ SasA conj_and→ KaiA
KaiC ←nsubj interacts prep_with→ SasA conj_and→ KaiB

Stanford Dependencies

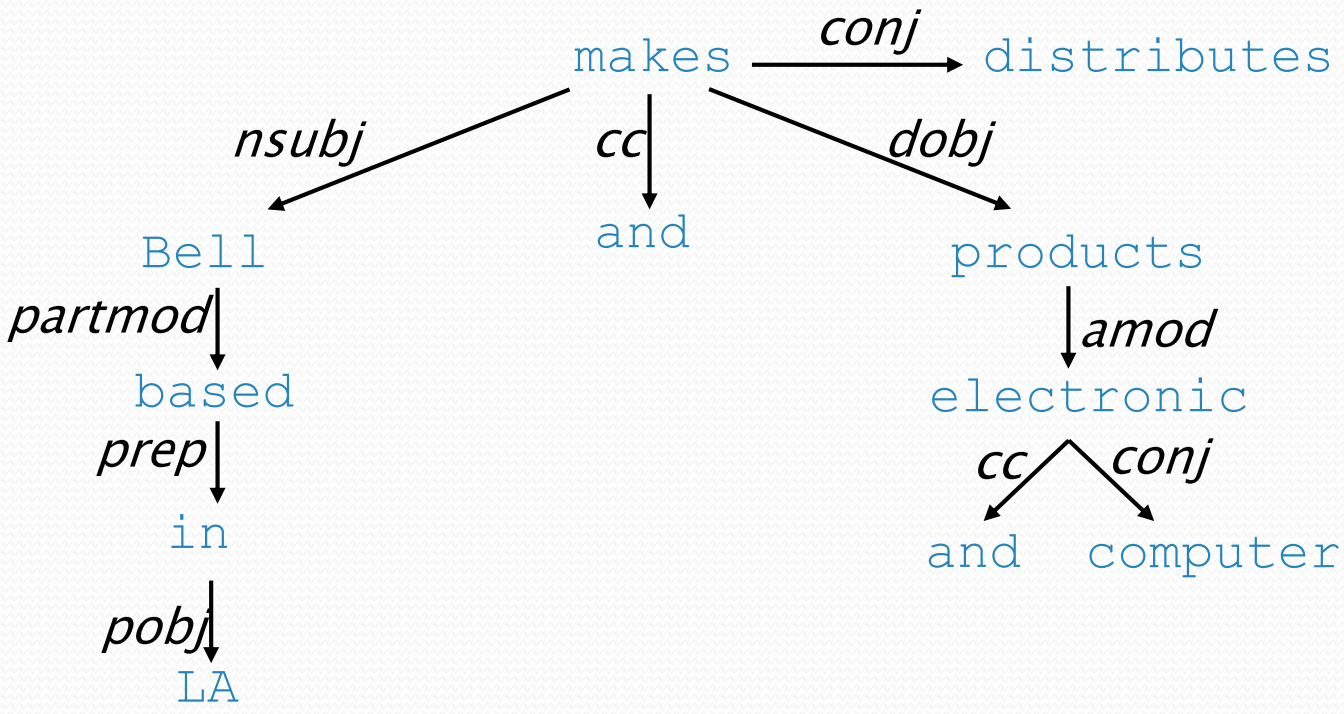
[de Marneffe et al. LREC 2006]

- The basic dependency representation is projective
- It can be generated by postprocessing headed phrase structure parses (Penn Treebank syntax)
- It can also be generated directly by dependency parsers, such as MaltParser, or the Easy-First Parser



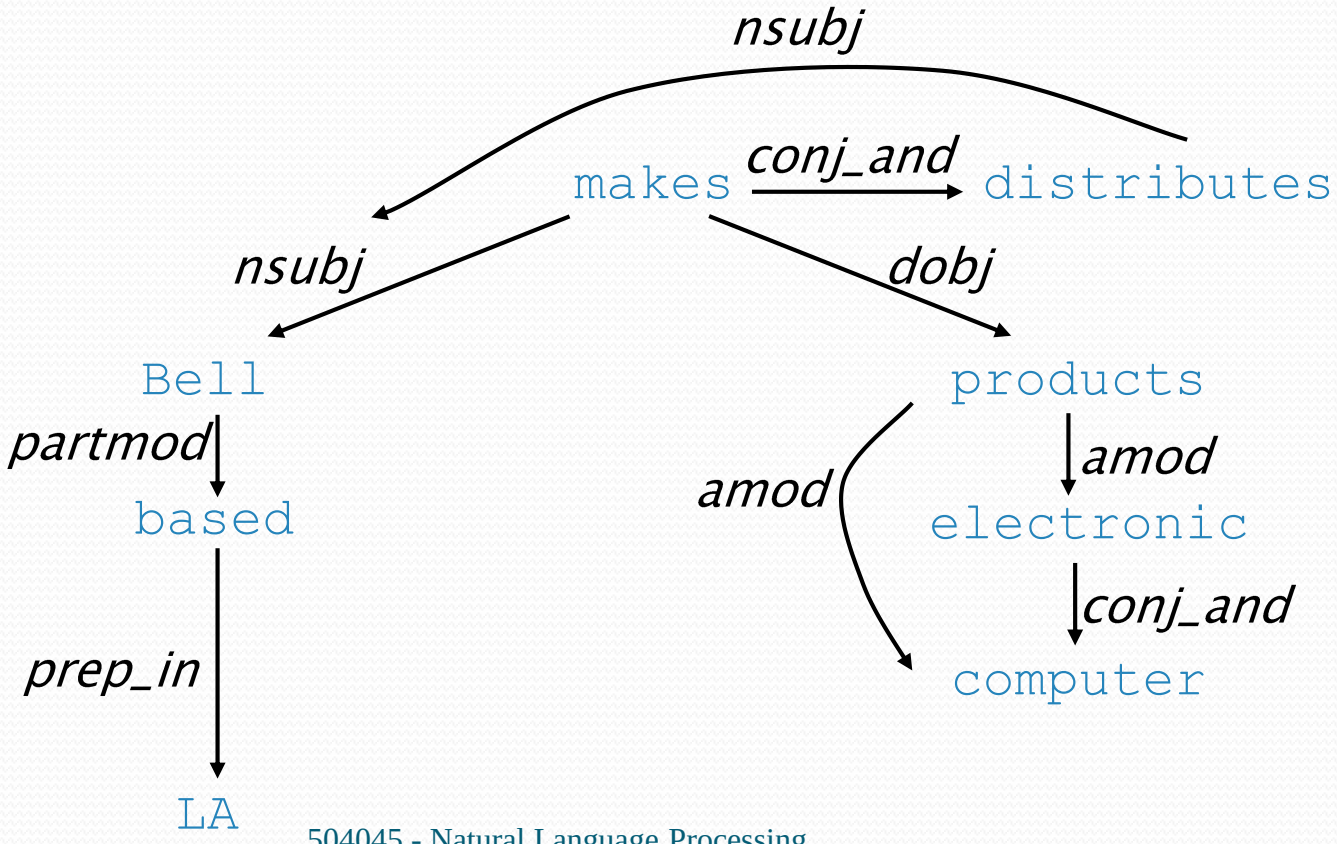
Graph modification to facilitate semantic analysis

Bell, based in LA, makes and distributes electronic and computer products.



Graph modification to facilitate semantic analysis

Bell, based in LA, makes and distributes electronic and computer products.



BioNLP 2009/2011 relation extraction shared tasks [Björne et al. 2009]