

---

# Syntax and Parsing

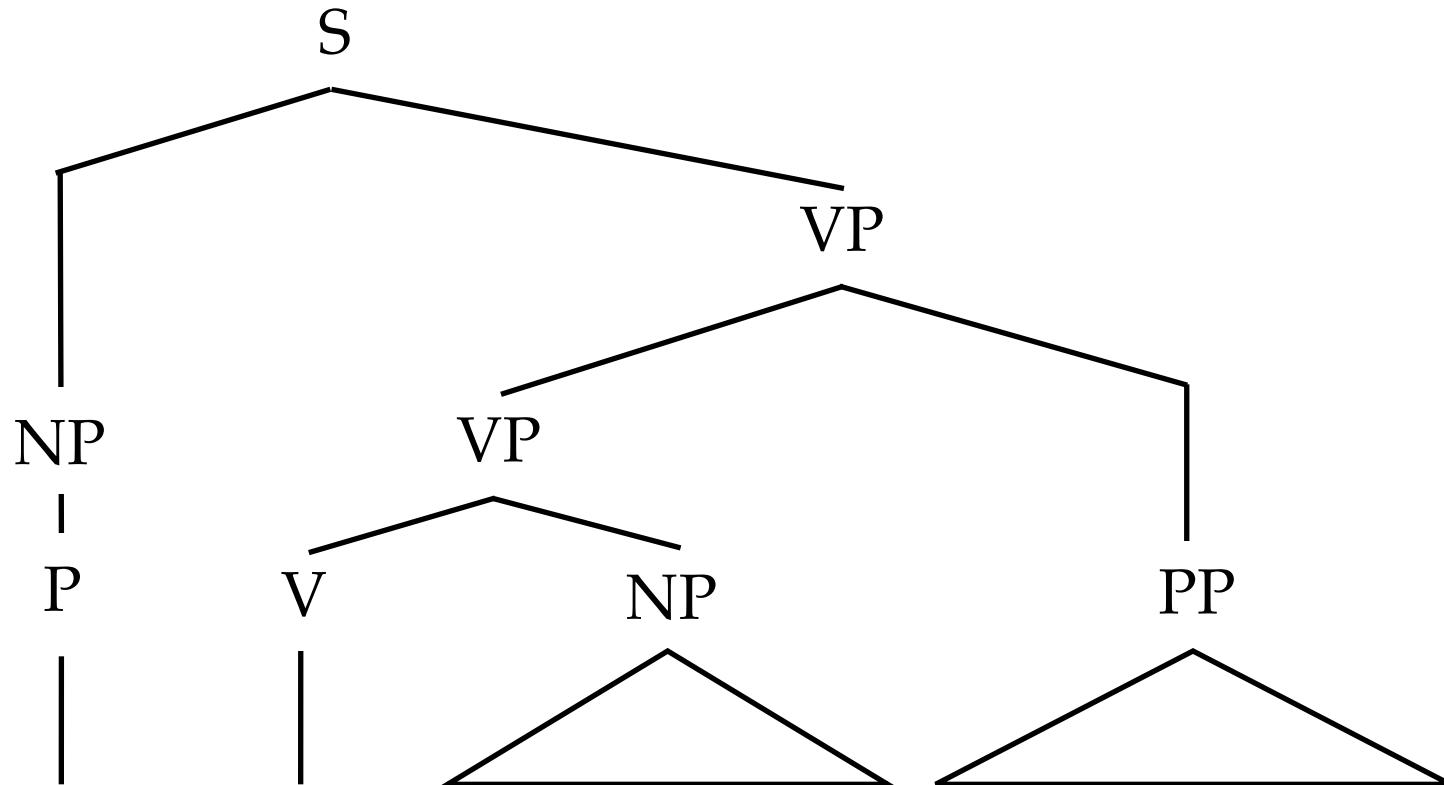
Slav Petrov – Google Research

Thanks to: Dan Klein, Ryan McDonald

Lisbon Machine Learning School

# Analyzing Natural Language

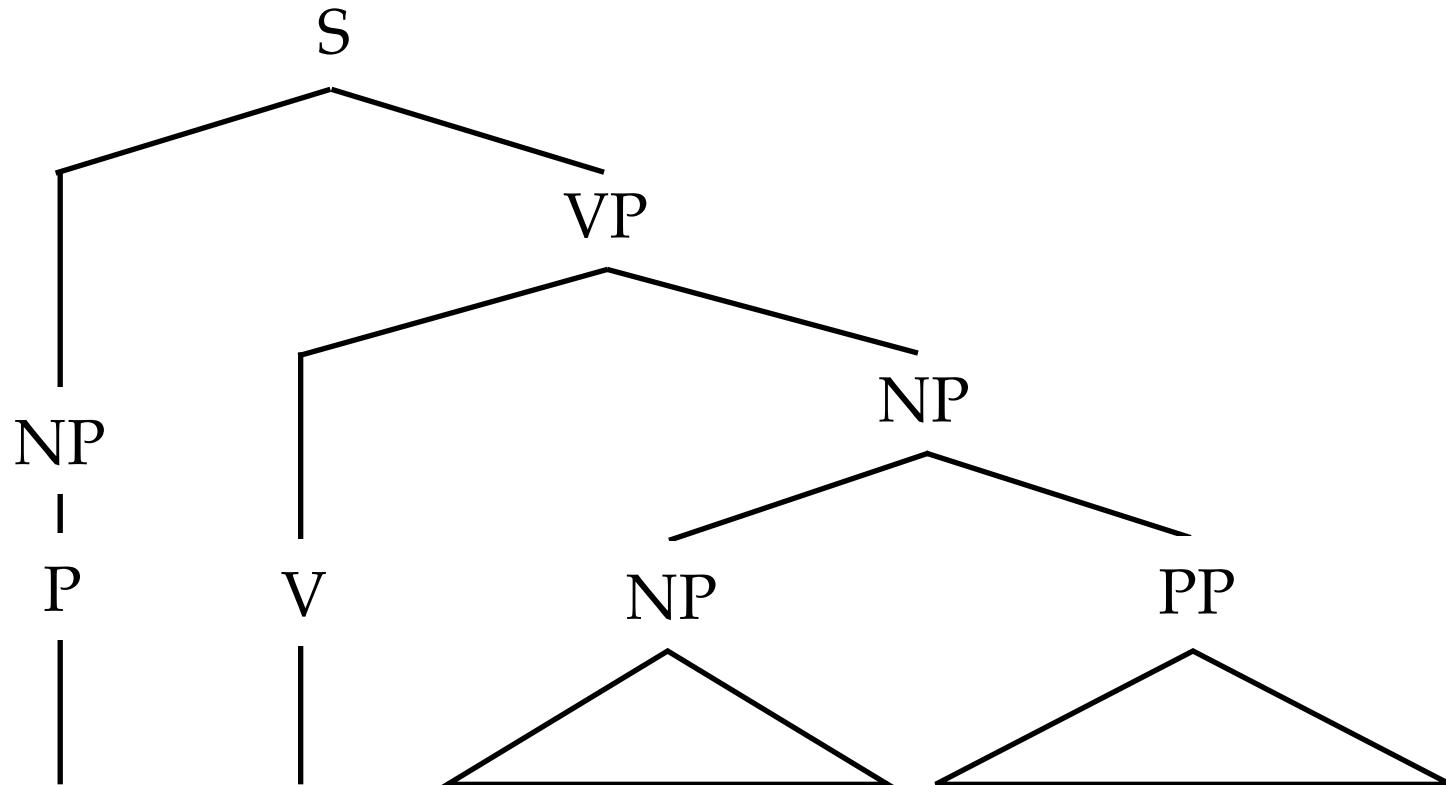
---



*They [solved [the problem] with statistics].*

# Syntax and Semantics

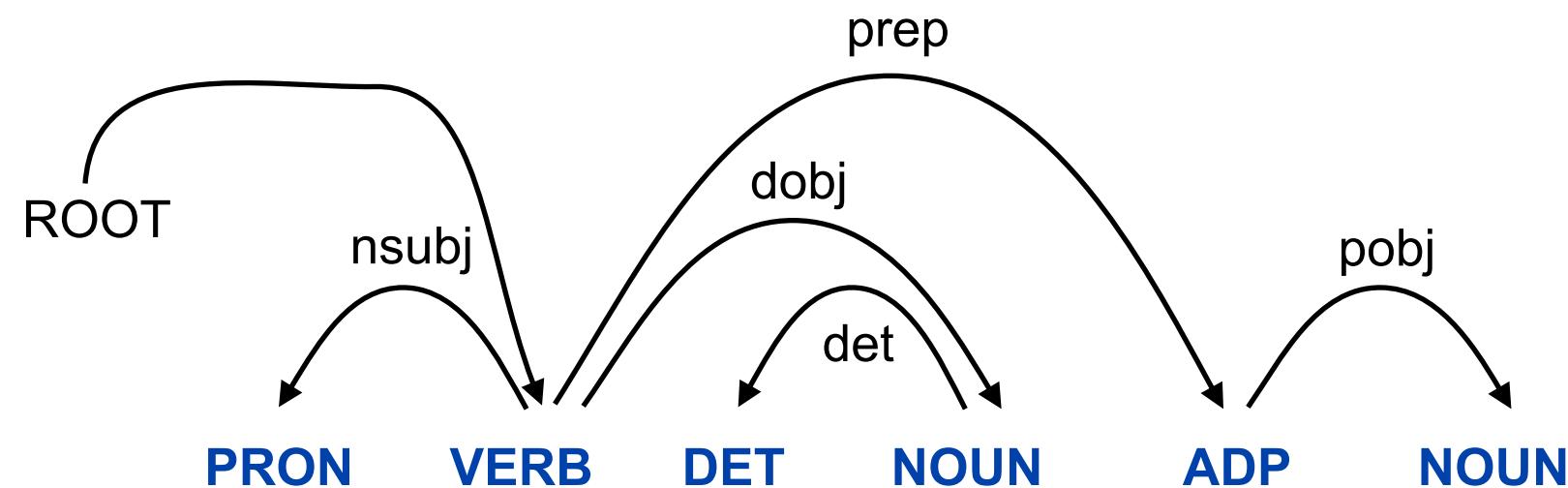
---



*They solved the problem with statistics .*

# Constituency and Dependency

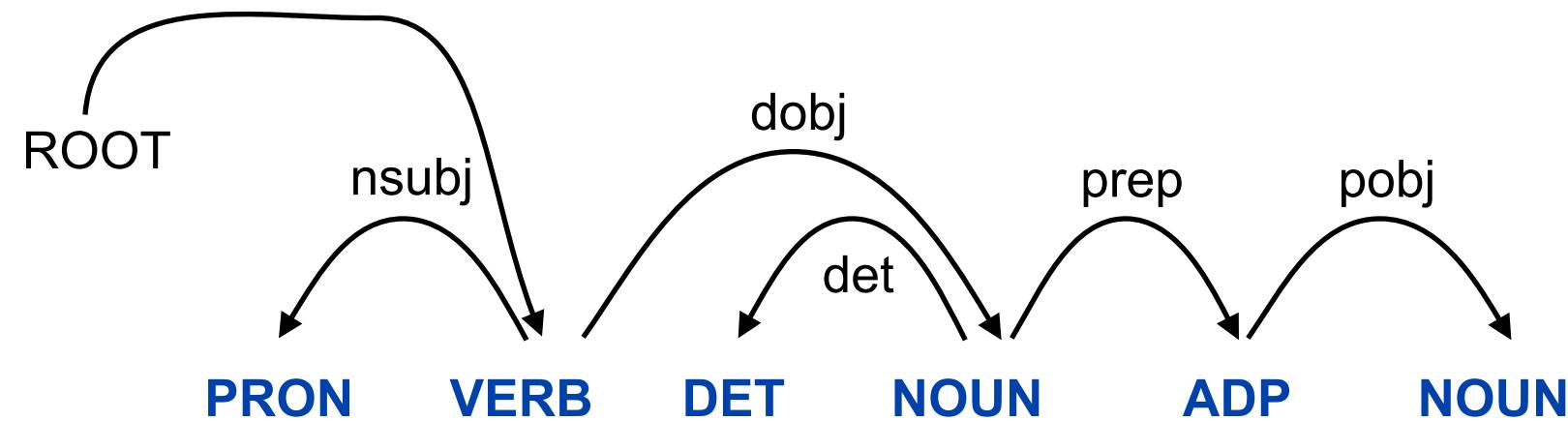
---



*They solved the problem with statistics .*

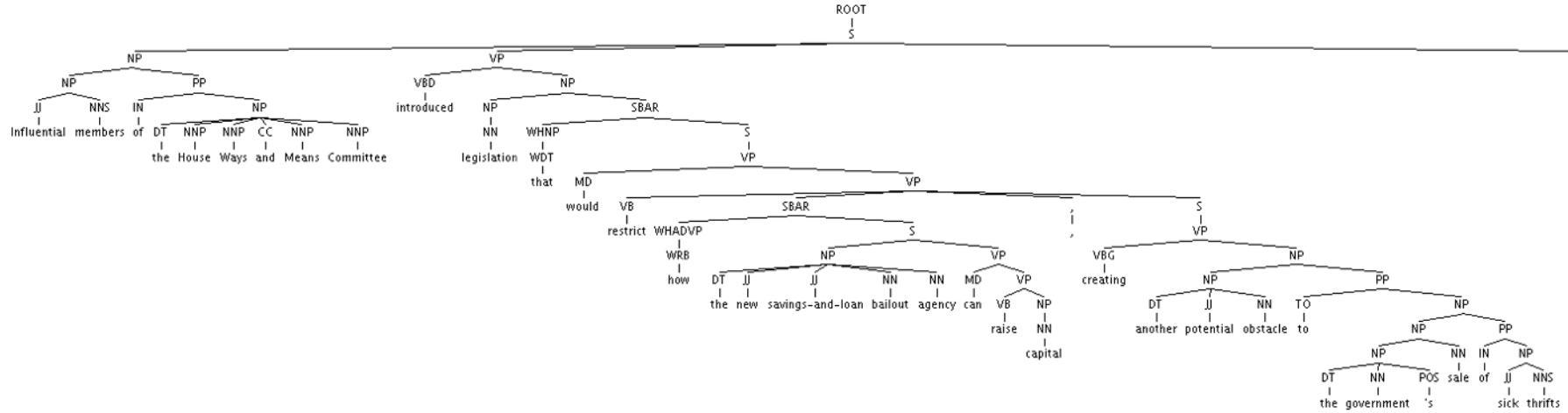
# Constituency and Dependency

---



*They solved the problem with statistics .*

# A “real” Sentence

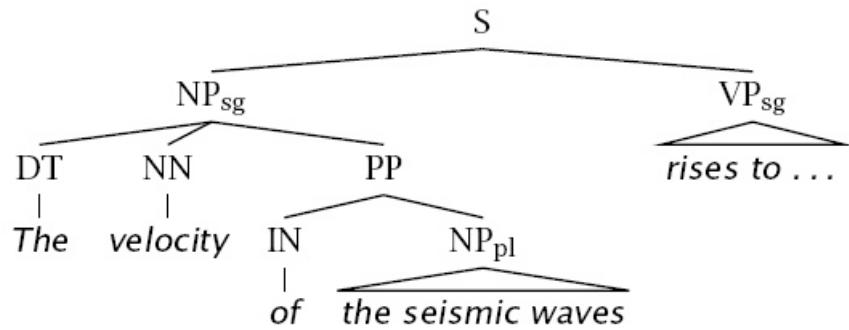


Influential members of the House Ways and Means Committee introduced legislation that would restrict how the new savings-and-loan bailout agency can raise capital, creating another potential obstacle to the government's sale of sick thrif.

# Phrase Structure Parsing

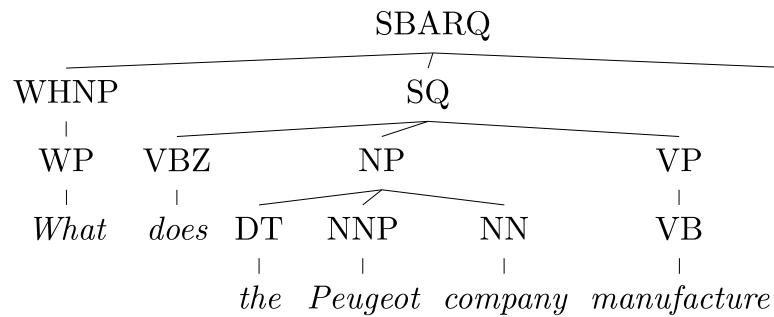
---

- Phrase structure parsing organizes syntax into *constituents* or *brackets*
- In general, this involves nested trees
- Linguists can, and do, argue about details
- Lots of ambiguity
- Not the only kind of syntax...
- First part of today's lecture

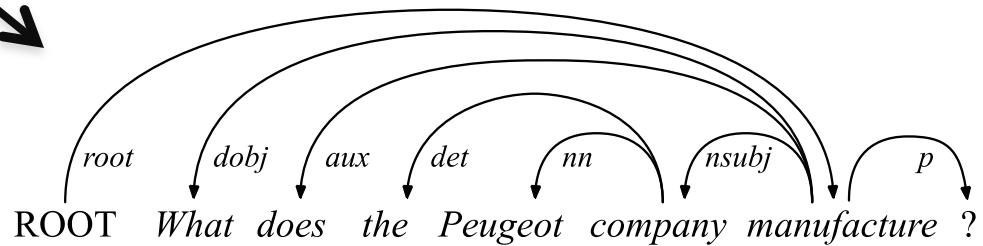


*new art critics write reviews with computers*

# Dependency Parsing



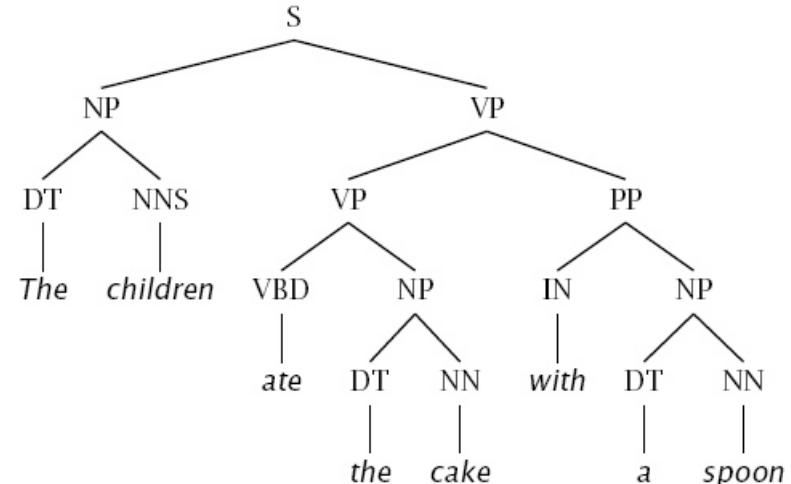
- Directed edges between pairs of word (head, dependent)
- Can handle free word-order languages
- Very efficient decoding algorithms exist
- Second part of today's lecture



# Constituency Tests

---

- How do we know what nodes go in the tree?
- Classic constituency tests:
  - Substitution by *proform*
  - Question answers
  - Semantic grounds
    - Coherence
    - Reference
    - Idioms
  - Dislocation
  - Conjunction
- Cross-linguistic arguments, too



# Conflicting Tests

---

- Constituency isn't always clear

- Units of transfer:

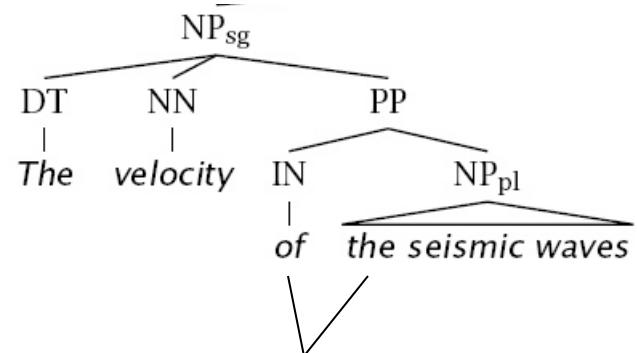
- think about ~ penser à
    - talk about ~ hablar de

- Phonological reduction:

- I will go → I'll go
    - I want to go → I wanna go
    - a le centre → au centre

- Coordination

- He went to and came from the store.



La vélocité des ondes sismiques

# Classical NLP: Parsing

---

- Write symbolic or logical rules:

VBD		VB			
VBN	VBZ	VBP	VBZ		
NNP	NNS	NN	NNS	CD	NN

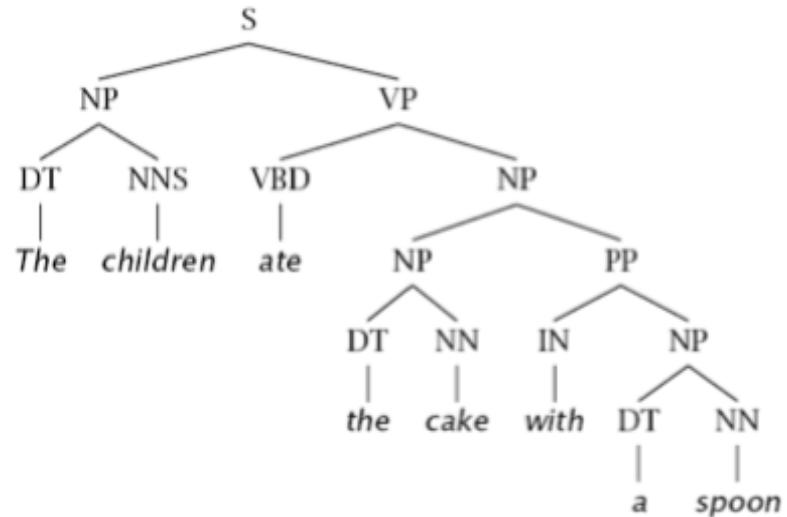
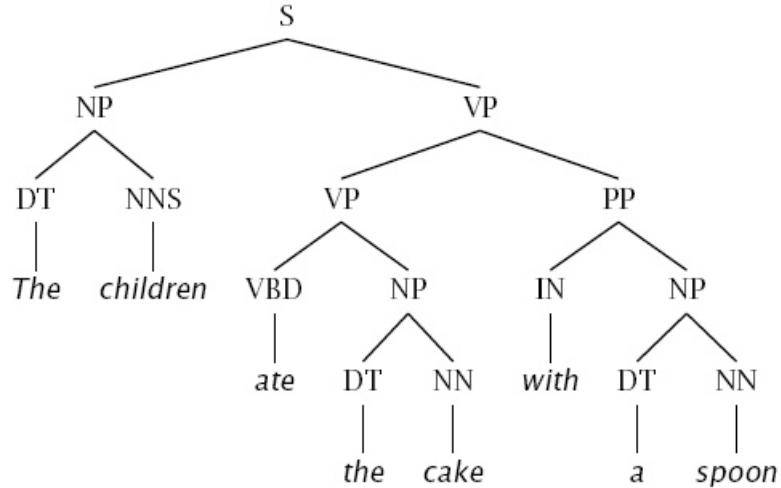
Fed raises interest rates 0.5 percent

---

Grammar (CFG)		Lexicon
$\text{ROOT} \rightarrow S$	$NP \rightarrow NP\ PP$	$NN \rightarrow \text{interest}$
$S \rightarrow NP\ VP$	$VP \rightarrow VBP\ NP$	$NNS \rightarrow \text{raises}$
$NP \rightarrow DT\ NN$	$VP \rightarrow VBP\ NP\ PP$	$VBP \rightarrow \text{interest}$
$NP \rightarrow NN\ NNS$	$PP \rightarrow IN\ NP$	$VBZ \rightarrow \text{raises}$

- Use deduction systems to prove parses from words
  - Minimal grammar on “Fed raises” sentence: 36 parses
  - Real-size grammar: many millions of parses
- This scaled very badly, didn’t yield broad-coverage tools

# Ambiguities: PP Attachment



The board approved [its acquisition] [by Royal Trustco Ltd.]  
[of Toronto]  
[for \$27 a share]  
[at its monthly meeting].

# Attachments

---

- I cleaned the dishes from dinner
- I cleaned the dishes with detergent
- I cleaned the dishes in my pajamas
- I cleaned the dishes in the sink

# Syntactic Ambiguities I

---

- Prepositional phrases:

*They cooked the beans in the pot on the stove with handles.*

- Particle vs. preposition:

*The puppy tore up the staircase.*

- Complement structures

*The tourists objected to the guide that they couldn't hear.*

*She knows you like the back of her hand.*

- Gerund vs. participial adjective

*Visiting relatives can be boring.*

*Changing schedules frequently confused passengers.*

# Syntactic Ambiguities II

---

- Modifier scope within NPs  
*impractical design requirements*  
*plastic cup holder*
- Multiple gap constructions  
*The chicken is ready to eat.*  
*The contractors are rich enough to sue.*
- Coordination scope:  
*Small rats and mice can squeeze into holes or cracks in the wall.*



# Probabilistic Context-Free Grammars

---

- A context-free grammar is a tuple  $\langle N, T, S, R \rangle$ 
  - $N$  : the set of non-terminals
    - Phrasal categories: S, NP, VP, ADJP, etc.
    - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
  - $T$  : the set of terminals (the words)
  - $S$  : the start symbol
    - Often written as ROOT or TOP
    - Not usually the sentence non-terminal S
  - $R$  : the set of rules
    - Of the form  $X \rightarrow Y_1 Y_2 \dots Y_k$ , with  $X, Y_i \in N$
    - Examples:  $S \rightarrow NP\ VP$ ,  $VP \rightarrow VP\ CC\ VP$
    - Also called rewrites, productions, or local trees
- A PCFG adds:
  - A top-down production probability per rule  $P(Y_1 Y_2 \dots Y_k | X)$

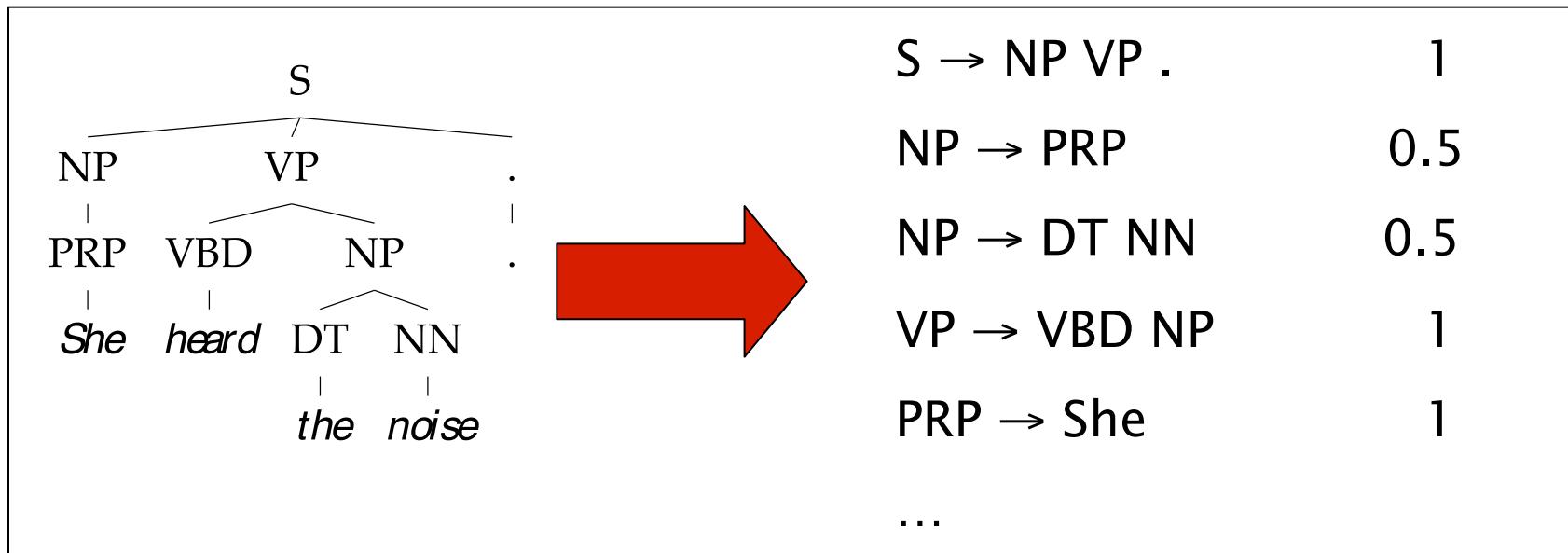
# Treebank Sentences

---

```
( (S (NP-SBJ The move)
      (VP followed
        (NP (NP a round)
            (PP of
              (NP (NP similar increases)
                  (PP by
                    (NP other tenders)))
              (PP against
                (NP Arizona real estate loans))))))
  ,
  (S-ADV (NP-SBJ *)
    (VP reflecting
      (NP (NP a continuing decline)
          (PP-LOC in
            (NP that market))))))
.)
```

# Treebank Grammars

- Need a PCFG for broad coverage parsing.
- Can take a grammar right off the trees (doesn't work well):

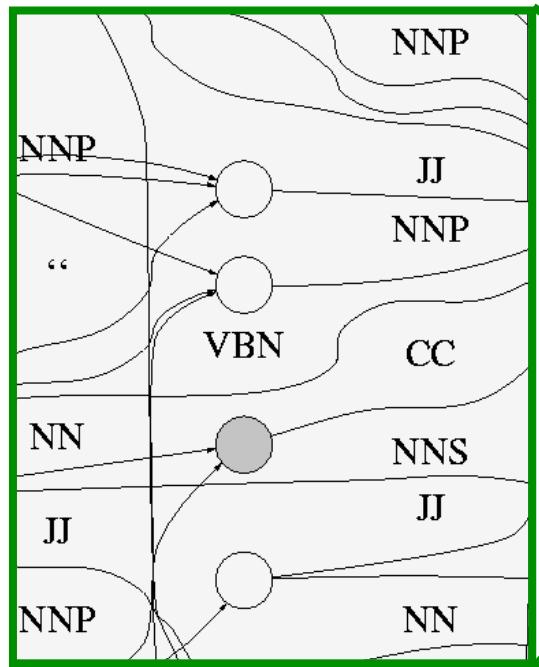


- Better results by enriching the grammar (e.g., lexicalization).
- Can also get reasonable parsers without lexicalization.

# Treebank Grammar Scale

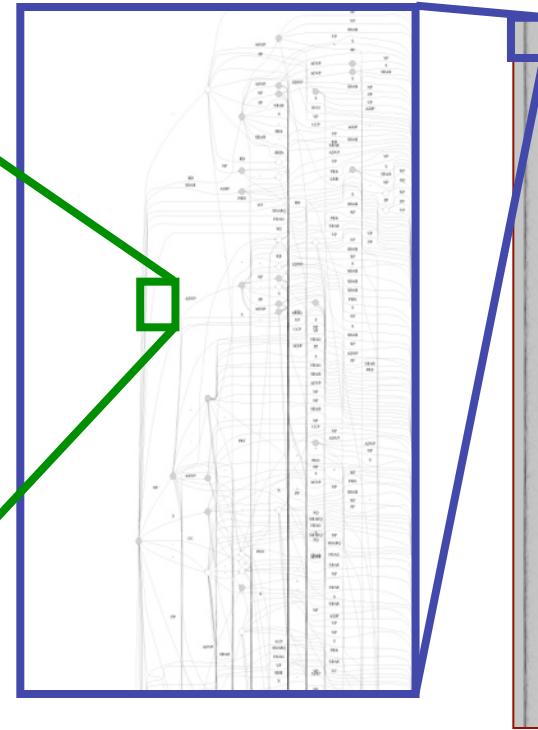
- Treebank grammars can be enormous
  - As FSAs, the raw grammar has ~10K states, excluding the lexicon
  - Better parsers usually make the grammars larger, not smaller

NP



)UN

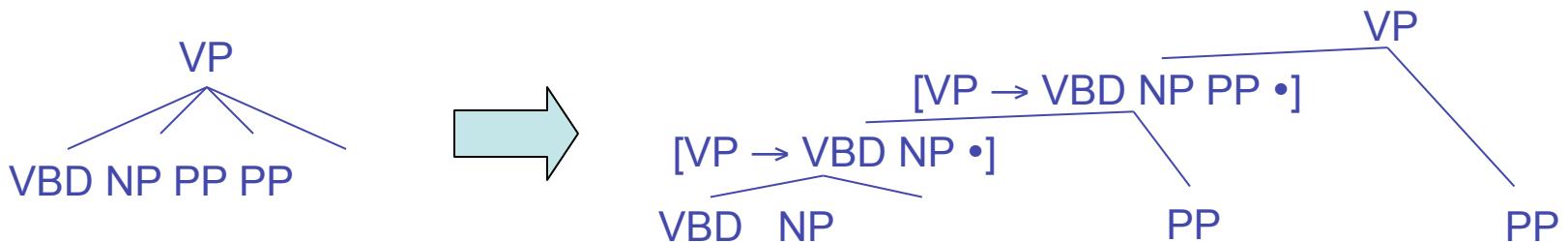
,



# Chomsky Normal Form

---

- Chomsky normal form:
  - All rules of the form  $X \rightarrow Y Z$  or  $X \rightarrow w$
  - In principle, this is no limitation on the space of (P)CFGs
    - N-ary rules introduce new non-terminals



- Unaries / empties are “promoted”
- In practice it’s kind of a pain:
  - Reconstructing n-aries is easy
  - Reconstructing unaries is trickier
  - The straightforward transformations don’t preserve tree scores
- Makes parsing algorithms simpler!

# A Recursive Parser

---

```
bestScore(X,i,j,s)
    if (j = i+1)
        return tagScore(X,s[i])
    else
        return max score(X->YZ) *
                    bestScore(Y,i,k) *
                    bestScore(Z,k,j)
```

- Will this parser work?
- Why or why not?
- Memory requirements?

# A Memoized Parser

---

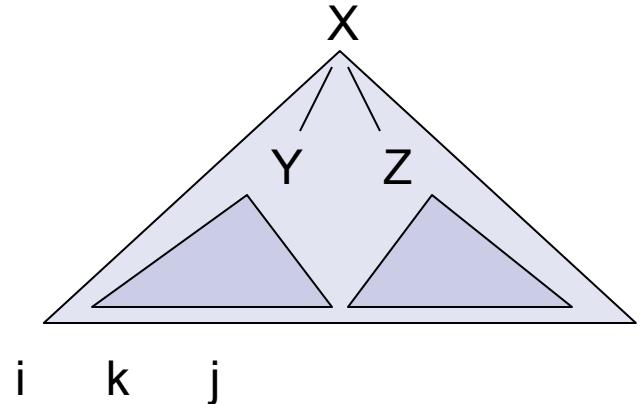
- One small change:

```
bestScore(X,i,j,s)
    if (scores[X][i][j] == null)
        if (j = i+1)
            score = tagScore(X,s[i])
        else
            score = max  score(X->YZ) *
                          bestScore(Y,i,k) *
                          bestScore(Z,k,j)
            scores[X][i][j] = score
    return scores[X][i][j]
```

# A Bottom-Up Parser (CKY)

- Can also organize things bottom-up

```
bestScore(s)
    for (i : [0,n-1])
        for (X : tags[s[i]])
            score[X][i][i+1] =
                tagScore(X,s[i])
    for (diff : [2,n])
        for (i : [0,n-diff])
            j = i + diff
            for (X->YZ : rule)
                for (k : [i+1, j-1])
                    score[X][i][j] = max score[X][i][j],
                        score(X->YZ) *
                        score[Y][i][k] *
                        score[Z][k][j]
```



# Unary Rules

---

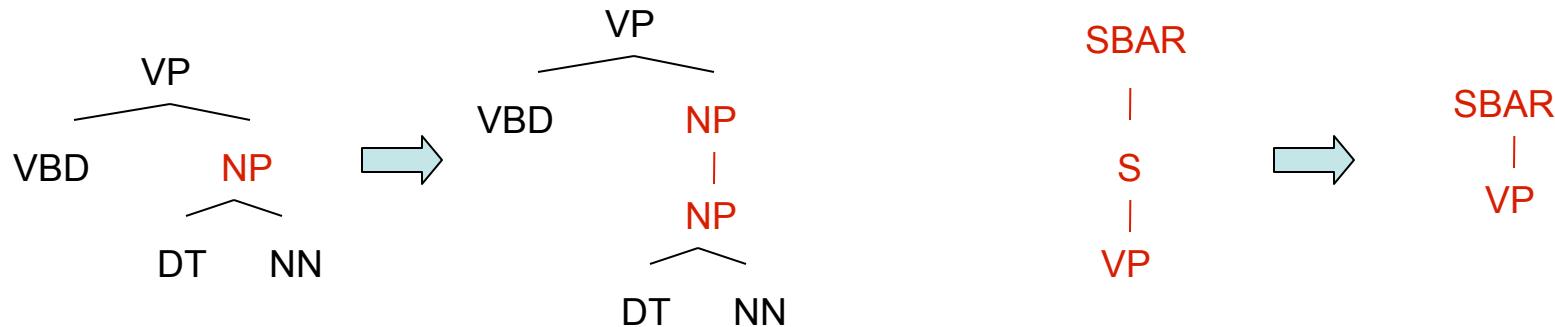
- Unary rules?

```
bestScore(X,i,j,s)
    if (j = i+1)
        return tagScore(X,s[i])
    else
        return max max score(X->YZ) *
        bestScore(Y,i,k) *
        bestScore(Z,k,j)
                max score(X->Y) *
                bestScore(Y,i,j)
```

# CNF + Unary Closure

---

- We need unaries to be non-cyclic
  - Can address by pre-calculating the *unary closure*
  - Rather than having zero or more unaries, always have exactly one



- Alternate unary and binary layers
- Reconstruct unary chains afterwards

# Alternating Layers

---

```
bestScoreB(X,i,j,s)
    return max max score(X->YZ) *
bestScoreU(Y,i,k) *
bestScoreU(Z,k,j)
```

```
bestScoreU(X,i,j,s)
if (j = i+1)
    return tagScore(X,s[i])
else
    return max max score(X->Y) *
                    bestScoreB(Y,i,j)
```

# Memory

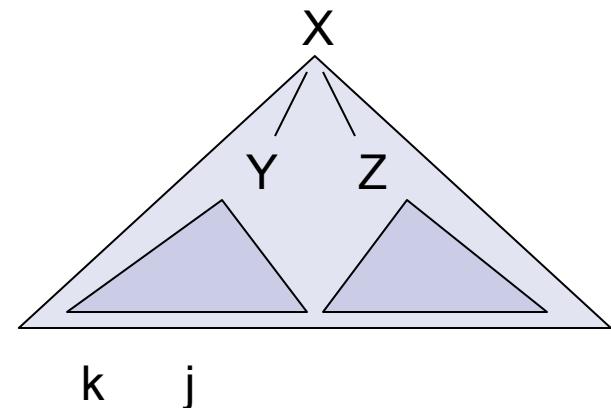
---

- How much memory does this require?
  - Have to store the score cache
  - Cache size:  $|\text{symbols}| * n^2$  doubles
  - For the plain treebank grammar:
    - $X \sim 20K$ ,  $n = 40$ , double  $\sim 8$  bytes  $= \sim 256\text{MB}$
    - Big, but workable.
- Pruning: Beams
  - $\text{score}[X][i][j]$  can get too large (when?)
  - Can keep beams (truncated maps  $\text{score}[i][j]$ ) which only store the best few scores for the span  $[i,j]$
- Pruning: Coarse-to-Fine
  - Use a smaller grammar to rule out most  $X[i,j]$
  - Much more on this later...

# Time: Theory

---

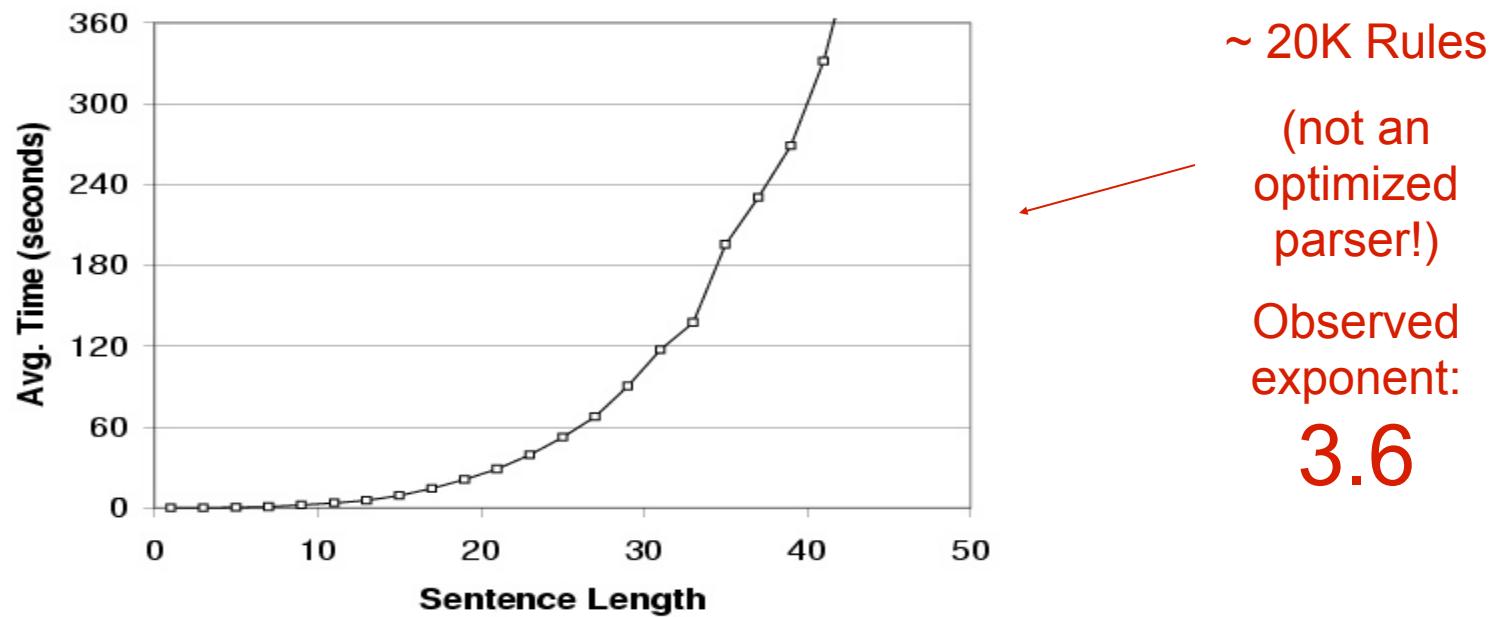
- How much time will it take to parse?
  - For each diff ( $\leq n$ )
    - For each  $i$  ( $\leq n$ )
      - For each rule  $X \rightarrow Y Z$ 
        - For each split point  $k$   
Do constant work
  - Total time:  $|\text{rules}| * n^3$
  - Something like 5 sec for an unoptimized parse of a 20-word sentences, or 0.2sec for an optimized parser



# Time: Practice

---

- Parsing with the vanilla treebank grammar:



- Why's it worse in practice?
  - Longer sentences “unlock” more of the grammar
  - All kinds of systems issues don’t scale

# Efficient CKY

---

- Lots of tricks to make CKY efficient
  - Most of them are little engineering details:
    - E.g., first choose  $k$ , then enumerate through the  $Y:[i,k]$  which are non-zero, then loop through rules by left child.
    - Optimal layout of the dynamic program depends on grammar, input, even system details.
  - Another kind is more critical:
    - Many  $X:[i,j]$  can be suppressed on the basis of the input string
    - We'll see this next class as figures-of-merit or A\* heuristics

# Human Processing

---

- Garden pathing:

The cotton clothing is made of grows in Mississippi.

The author wrote the novel was likely to be a best-seller.

The complex houses married and single soldiers and their families.

- Ambiguity maintenance

Have the police . . . eaten their supper?

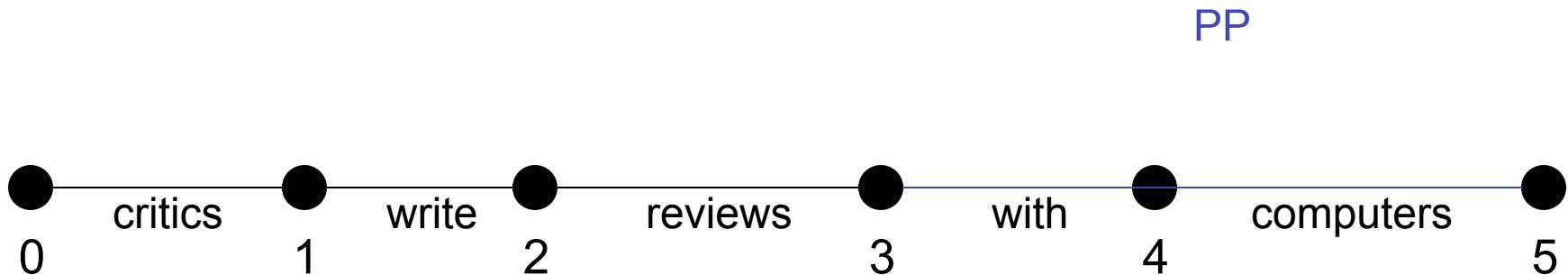
                          come in and look around.

                          taken out and shot.

# Agenda-Based Parsing

---

- Agenda-based parsing is like graph search (but over a hypergraph)
- Concepts:
  - Numbering: we number fenceposts between words
  - “Edges” or items: spans with labels, e.g. PP[3,5], represent the sets of trees over those words rooted at that label (cf. search states)
  - A chart: records edges we’ve expanded (cf. closed set)
  - An agenda: a queue which holds edges (cf. a fringe or open set)



# Word Items

---

- Building an item for the first time is called discovery.  
Items go into the agenda on discovery.
- To initialize, we discover all word items (with score 1.0).

## AGENDA

---

critics[0,1], write[1,2], reviews[2,3], with[3,4], computers[4,5]

## CHART [EMPTY]

---

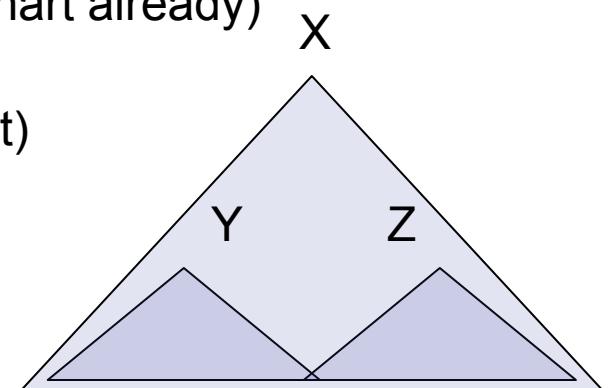


critics   write   reviews   with   computers

# Item Successors

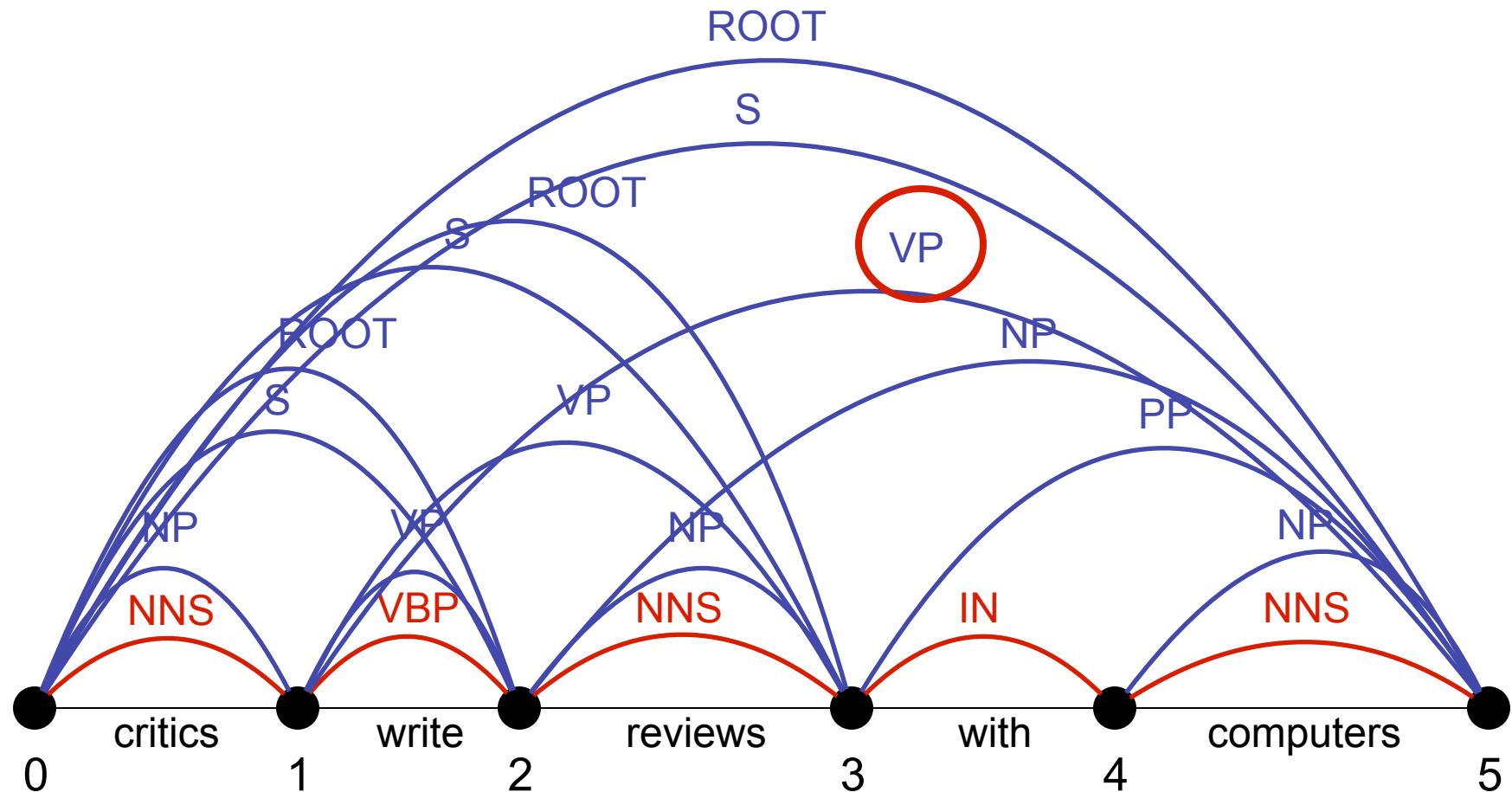
---

- When we pop items off of the agenda:
  - Graph successors: unary projections ( $\text{NNS} \rightarrow \text{critics}$ ,  $\text{NP} \rightarrow \text{NNS}$ )  
 $Y[i,j]$  with  $X \rightarrow Y$  forms  $X[i,j]$
  - Hypergraph successors: combine with items already in our chart  
 $Y[i,j]$  and  $Z[j,k]$  with  $X \rightarrow Y Z$  form  $X[i,k]$
- Enqueue / promote resulting items (if not in chart already)
- Record backtraces as appropriate
- Stick the popped edge in the chart (closed set)
- Queries a chart must support:
  - Is edge  $X:[i,j]$  in the chart? (What score?)
  - What edges with label  $Y$  end at position  $j$ ?
  - What edges with label  $Z$  start at position  $i$ ?



# An Example

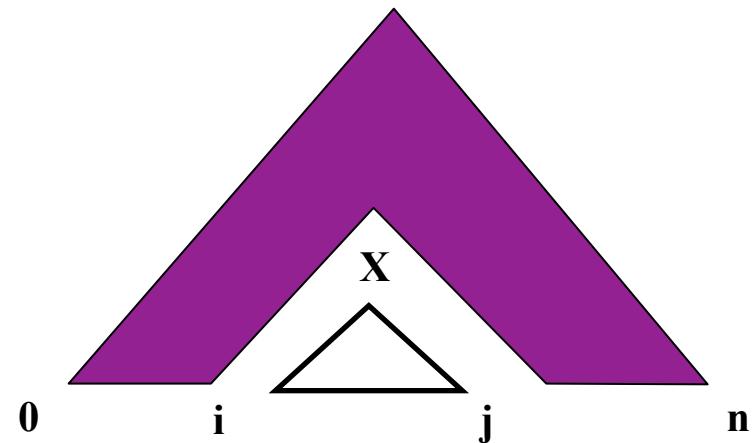
NNS[0,1] VBP[1,2] NNS[2,3] IN[3,4] NNS[3,4] NP[0,1] VP[1,2] NP[2,3] NP[4,5] S[0,2]  
VP[1,3] PP[3,5] ROOT[0,2] S[0,3] VP[1,5] NP[2,5] ROOT[0,3] S[0,5] ROOT[0,5]



# UCS / A\*

---

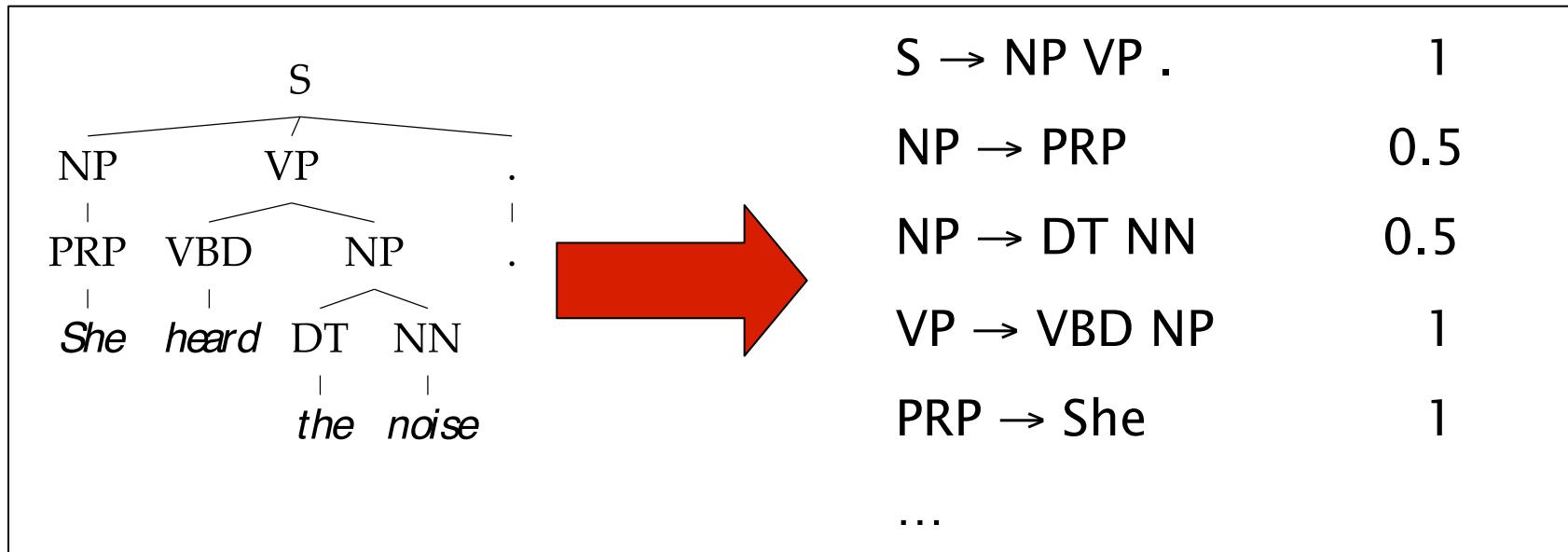
- With weighted edges, order matters
  - Must expand optimal parse from bottom up (subparses first)
  - CKY does this by processing smaller spans before larger ones
  - UCS pops items off the agenda in order of decreasing Viterbi score
  - A\* search also well defined
- You can also speed up the search without sacrificing optimality
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A\* heuristic, no loss of optimality [Klein and Manning 03]



# Treebank Grammars

[Charniak '96]

- Need a PCFG for broad coverage parsing.
- Can take a grammar right off the trees (doesn't work well):



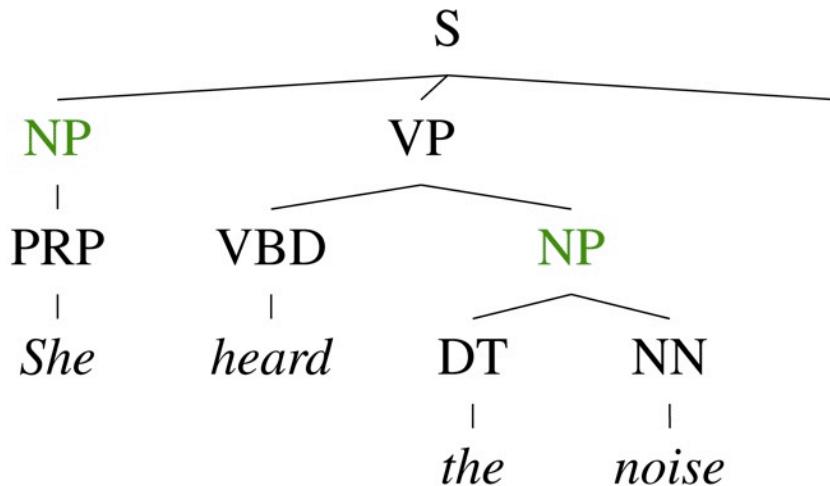
- Better results by enriching the grammar (context-free, etc.)
- Can also get reasonable parsers without

Model	F1
Charniak '96	72.0

# Conditional Independence?

---

- Not every NP expansion can fill every NP slot

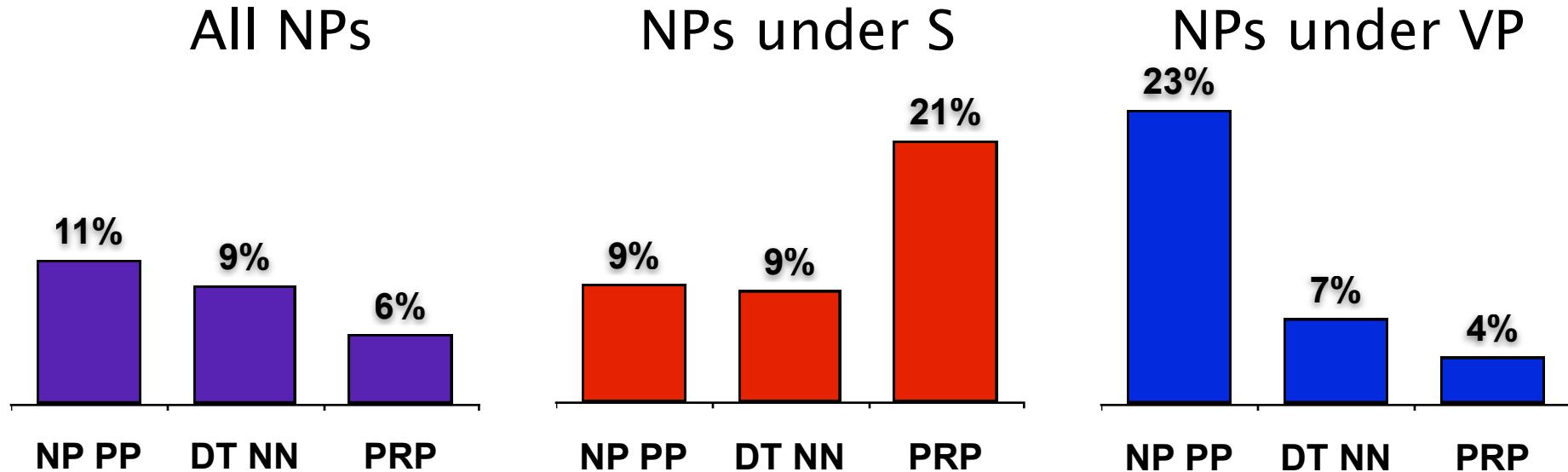


- A grammar with symbols like “NP” won’t be context-free
- Statistically, conditional independence too strong

# Non-Independence

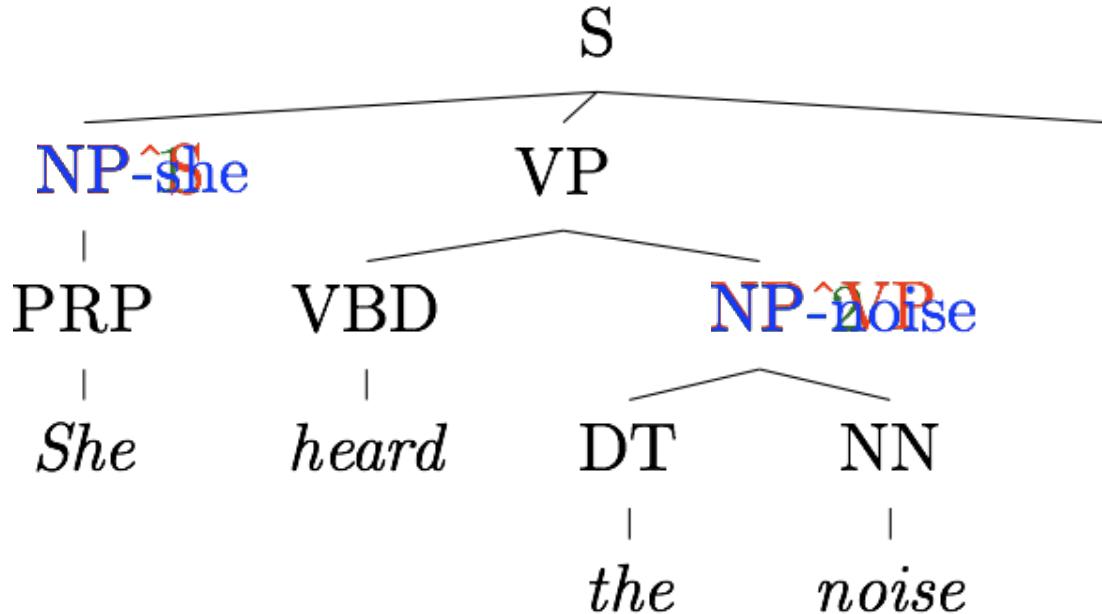
---

- Independence assumptions are often too strong.



- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).
- Also: the subject and object expansions are correlated!

# The Game of Designing a Grammar



- Structure Annotation [Johnson '98, Klein&Manning '03]
- Lexicalization [Collins '99, Charniak '00]
- Latent Variables [Matsuzaki et al. 05, Petrov et al. '06]

# Typical Experimental Setup

---

- Corpus: Penn Treebank, WSJ



Training: sections 02-21

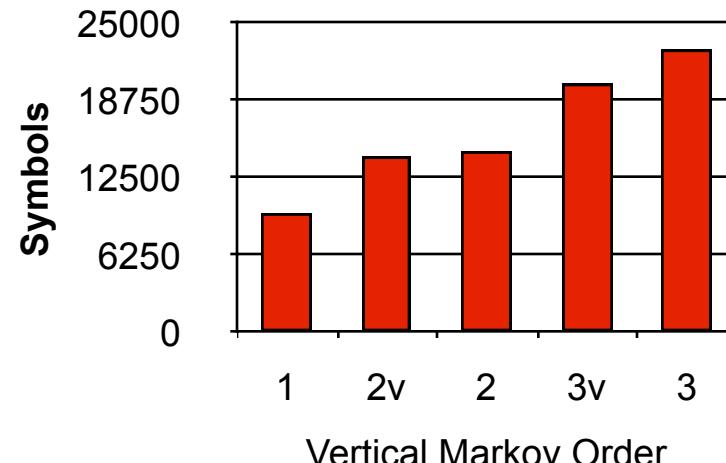
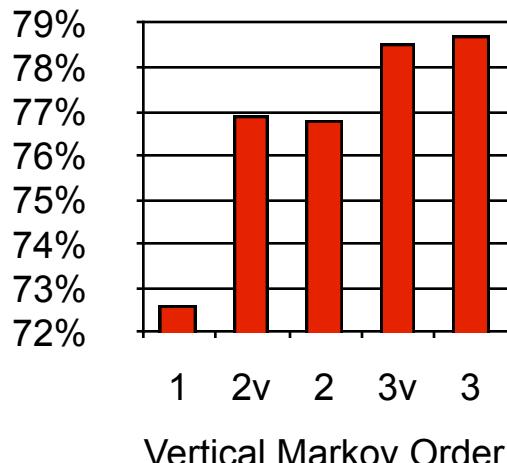
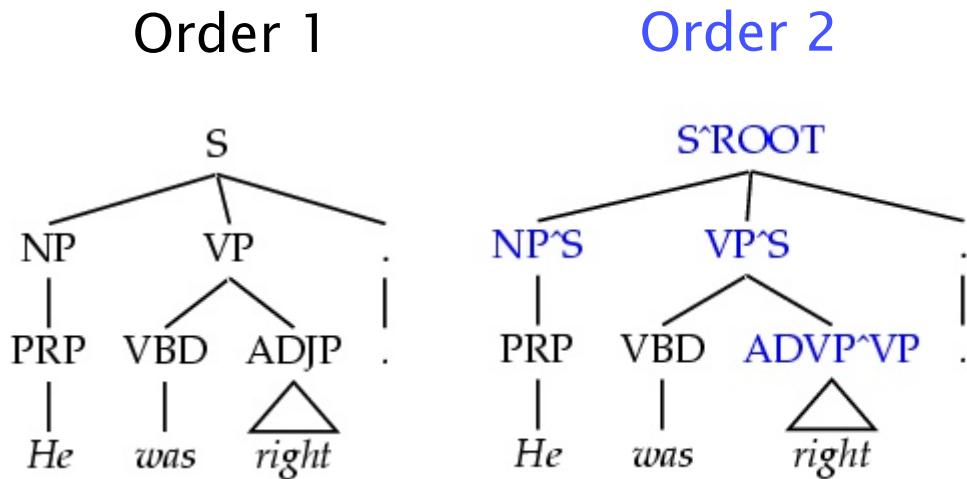
Development: section 22 (here, first 20 files)

Test: section 23

- Accuracy – F1: harmonic mean of per-node labeled precision and recall.
- Here: also size – number of symbols in grammar.
  - Passive / complete symbols: NP, NP<sup>A</sup>S
  - Active / incomplete symbols: NP → NP CC •

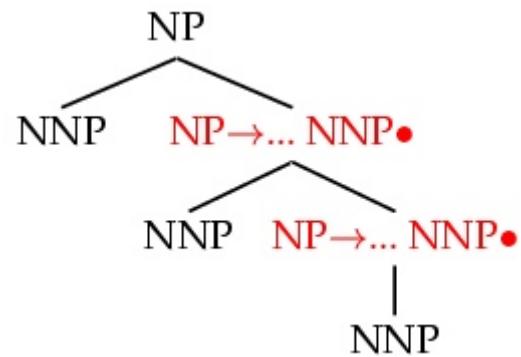
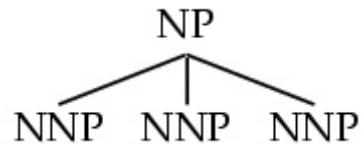
# Vertical Markovization

- Vertical Markov order: rewrites depend on past  $k$  ancestor nodes.  
(cf. parent annotation)

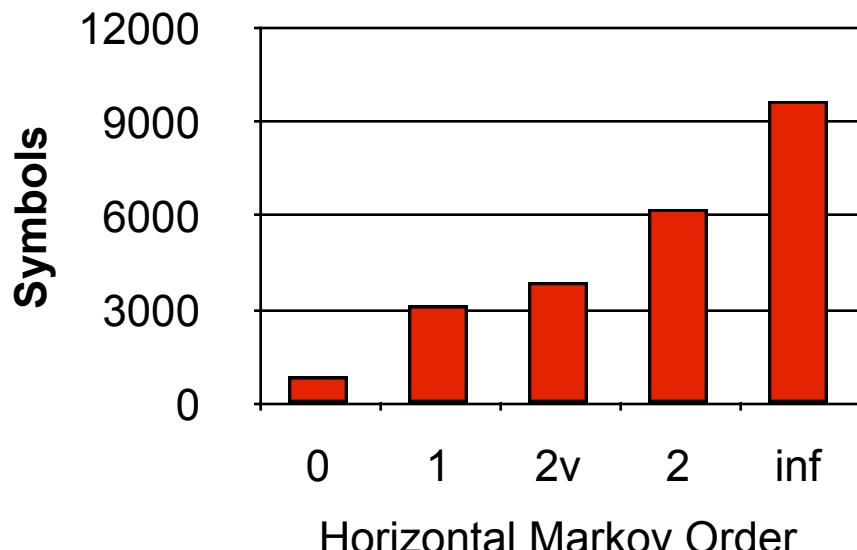
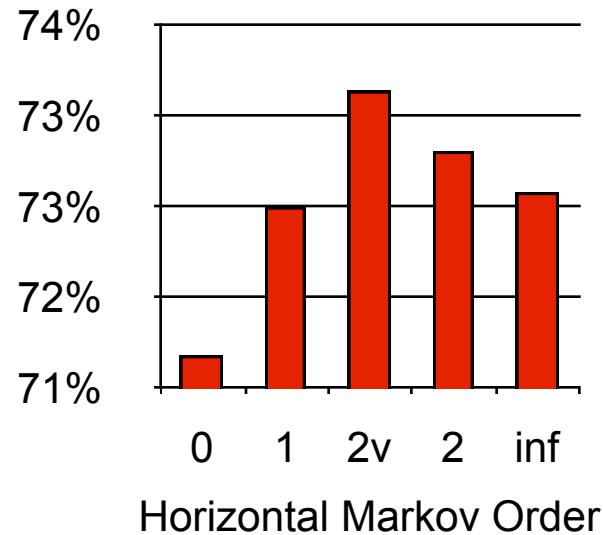
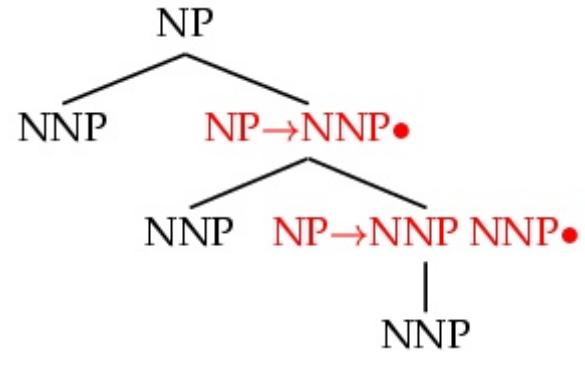


# Horizontal Markovization

Order 1

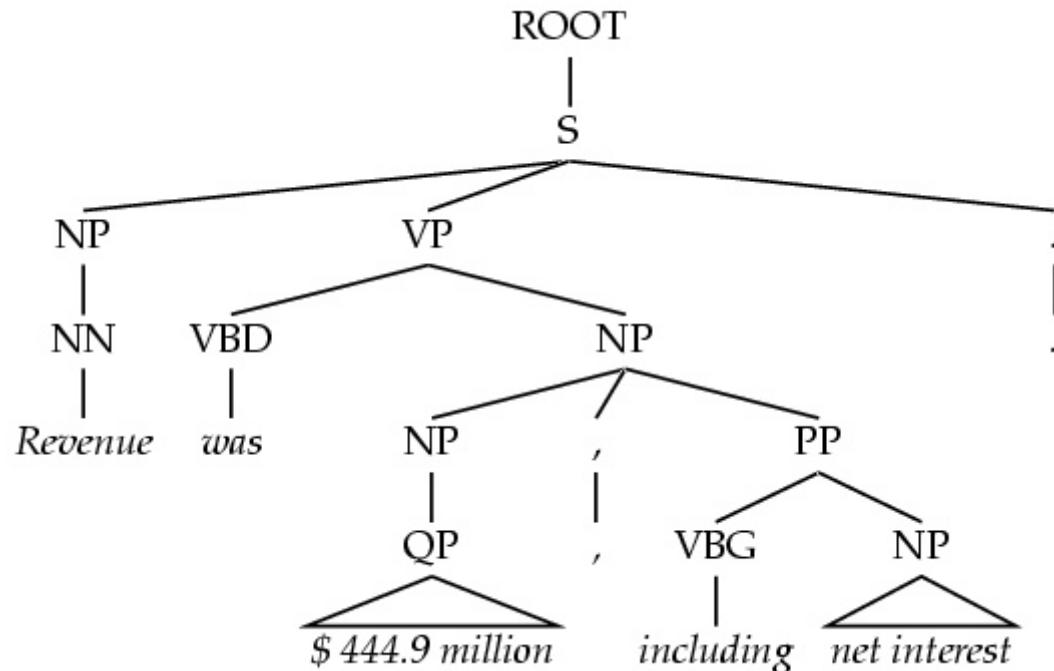


Order  $\infty$



# Unary Splits [Klein&Manning '03]

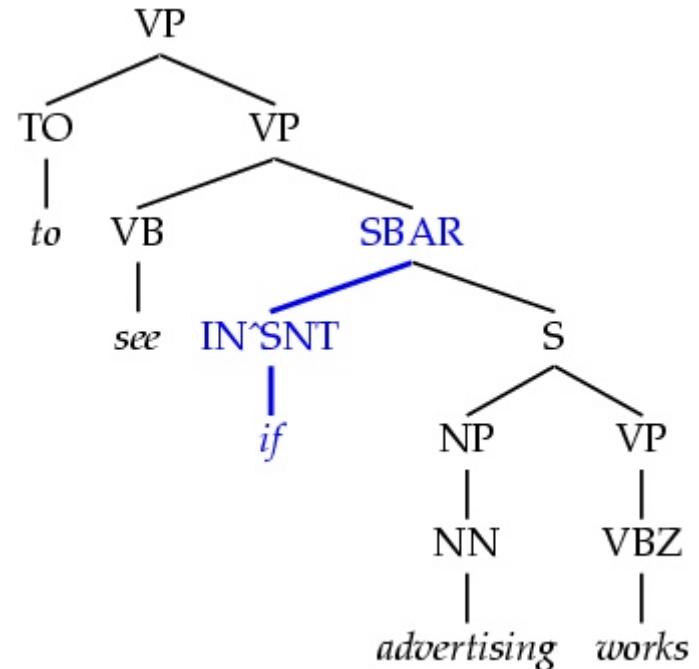
- Problem: unary rewrites used to transmute categories so a high-probability rule can be used.
- Solution: Mark unary rewrite sites with -U



Annotation	F1	Size
Base	77.8	7.5K
UNARY	78.3	8.0K

# Tag Splits

- Problem: Treebank tags are too coarse.
- Example: Sentential, PP, and other prepositions are all marked IN.
- Partial Solution:
  - Subdivide the IN tag.



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

# Other Tag Splits

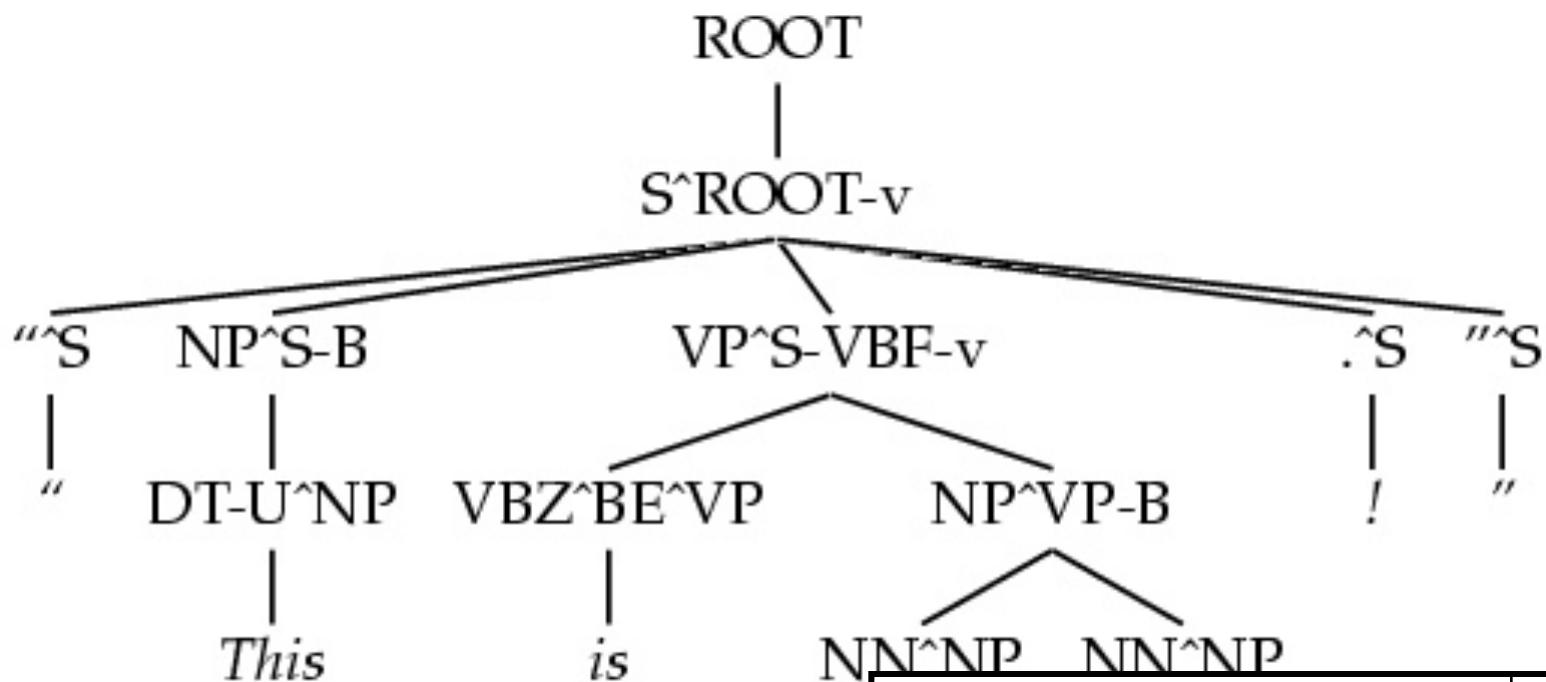
---

- **UNARY-DT**: mark demonstratives as  $DT^U$  (“the X” vs. “those”)
- **UNARY-RB**: mark phrasal adverbs as  $RB^U$  (“quickly” vs. “very”)
- **TAG-PA**: mark tags with non-canonical parents (“not” is an  $RB^VP$ )
- **SPLIT-AUX**: mark auxiliary verbs with  $-AUX$  [cf. Charniak 97]
- **SPLIT-CC**: separate “but” and “&” from other conjunctions
- **SPLIT-%**: “%” gets its own tag.

F1	Size
80.4	8.1K
80.5	8.1K
81.2	8.5K
81.6	9.0K
81.7	9.1K
81.8	9.3K

# A Fully Annotated (Unlexicalized) Tree

[Klein & Manning '03]



Model	F1
Charniak '96	72.0
Klein&Manning '03	86.3

# Some Test Set Results

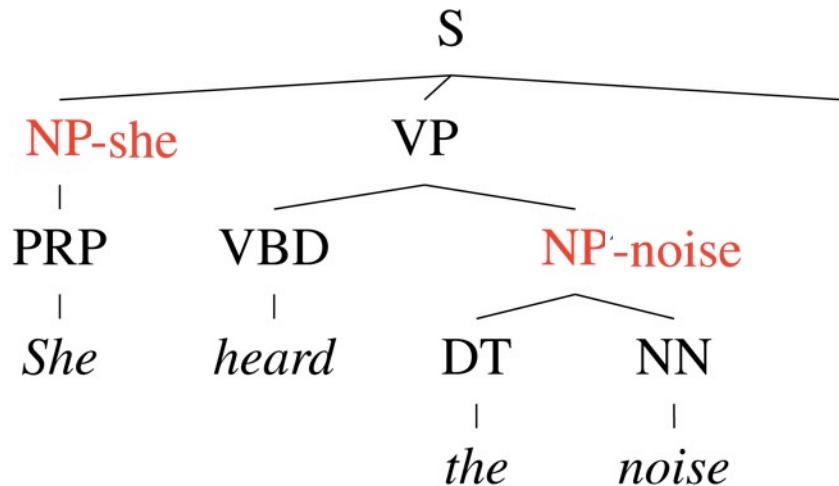
---

Parser	LP	LR	F1	CB	0 CB
Magerman	84.9	84.6	<b>84.7</b>	1.26	56.6
Collins 96	86.3	85.8	<b>86.0</b>	1.14	59.9
Unlexicalize	86.9	85.7	<b>86.3</b>	1.10	60.3
Charniak 97	87.4	87.5	<b>87.4</b>	1.00	62.1
Collins 99	88.7	88.6	<b>88.6</b>	0.90	67.1

- Beats “first generation” lexicalized parsers.
- Lots of room to improve – more complex models next.

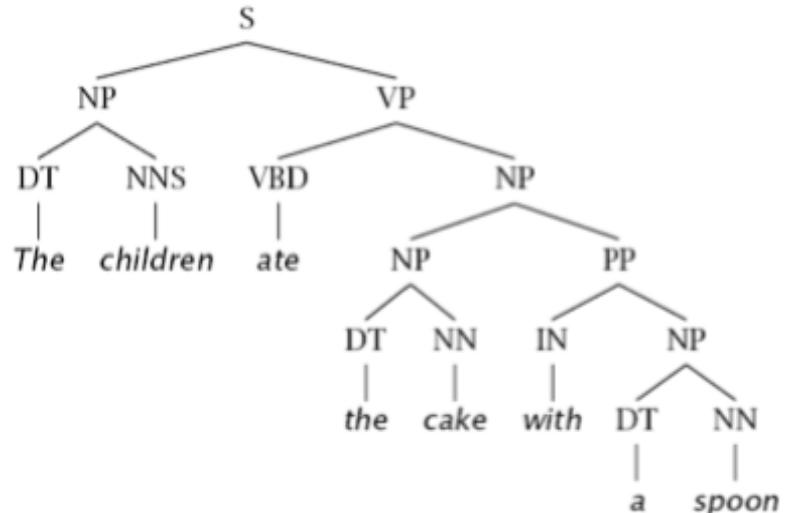
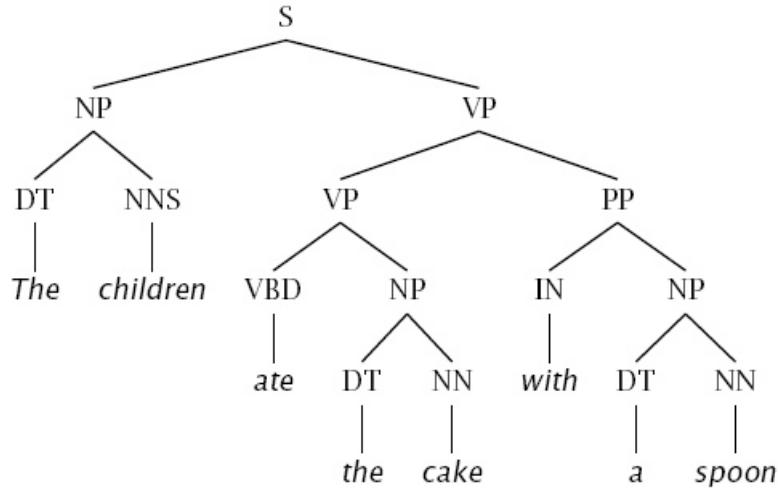
# The Game of Designing a Grammar

---



- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Head lexicalization [Collins '99, Charniak '00]

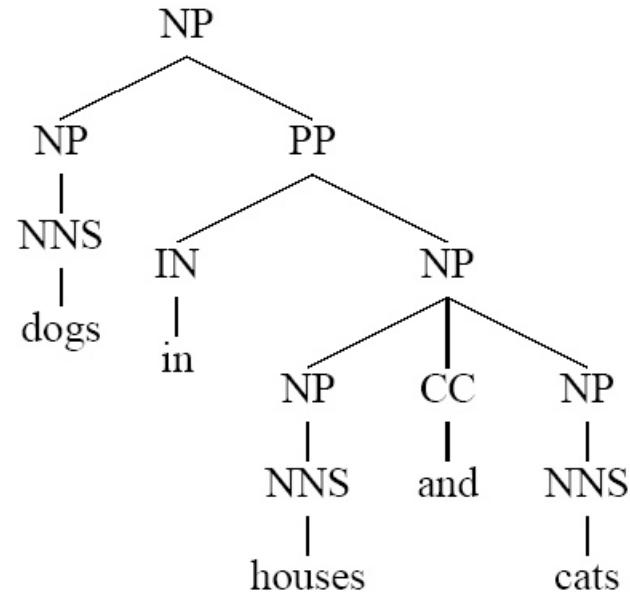
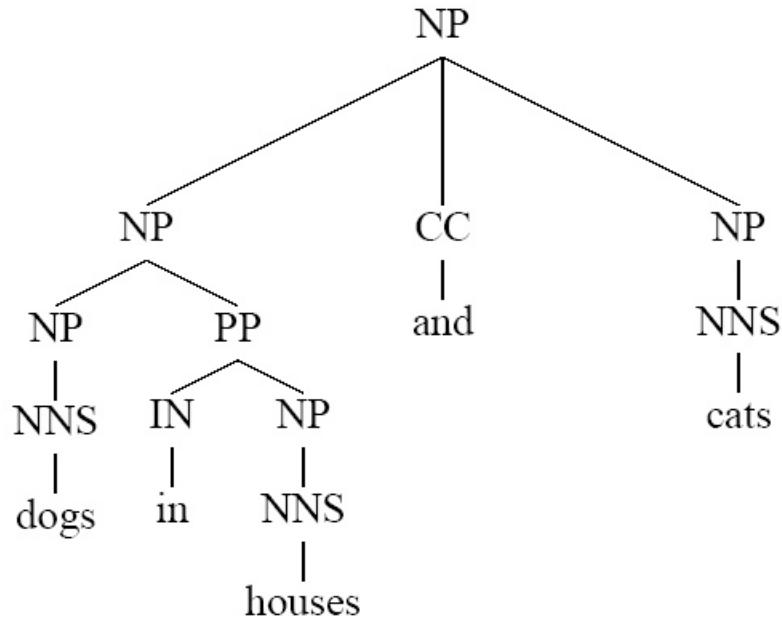
# Problems with PCFGs



- If we do no annotation, these trees differ only in one rule:
  - $VP \rightarrow VP\ PP$
  - $NP \rightarrow NP\ PP$
- Parse will go one way or the other, regardless of words
- We addressed this in one way with unlexicalized grammars (how?)
- Lexicalization allows us to be sensitive to specific words

# Problems with PCFGs

---

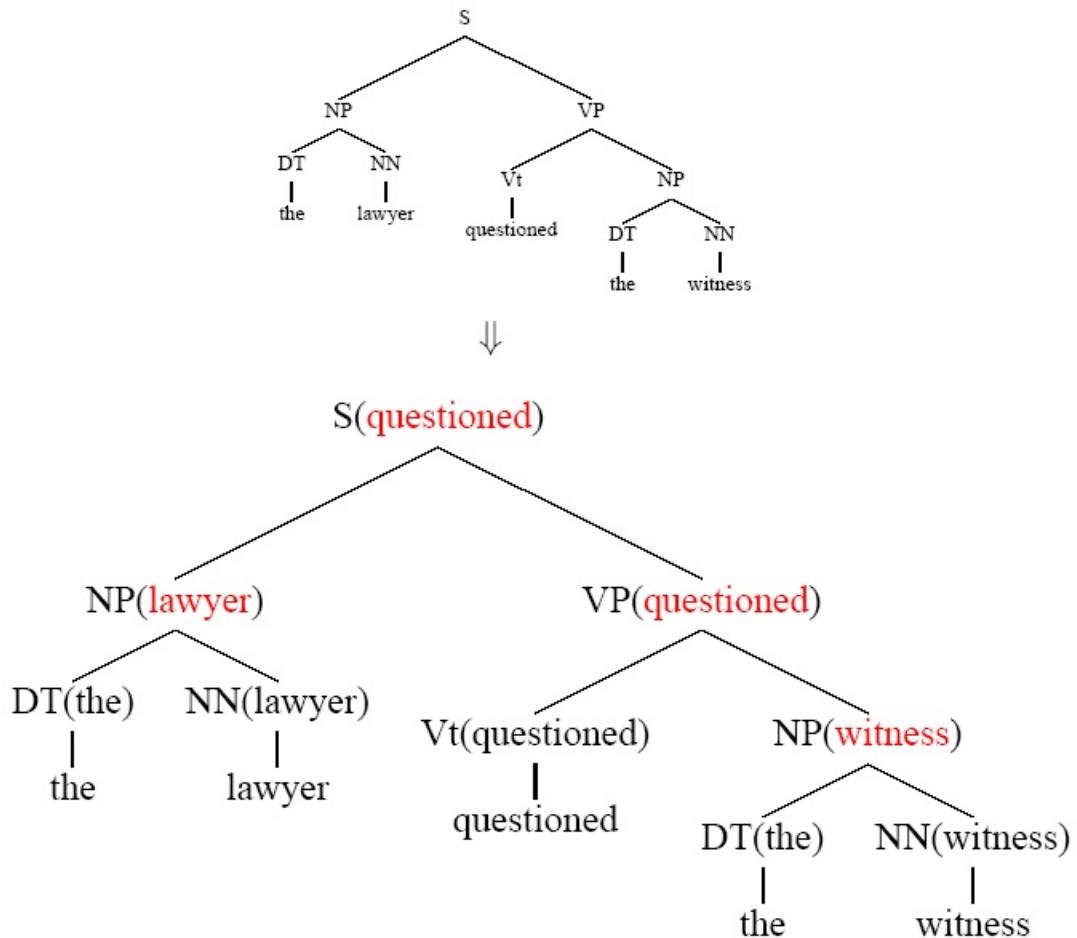


- What's different between basic PCFG scores here?
- What (lexical) correlations need to be scored?

# Lexicalized Trees

[Charniak '97,  
Collins '97]

- Add “headwords” to each phrasal node
  - Syntactic vs. semantic heads
  - Headship not in (most) treebanks
  - Usually *use head rules*, e.g.:
    - NP:
      - Take leftmost NP
      - Take rightmost N\*
      - Take rightmost JJ
      - Take right child
    - VP:
      - Take leftmost VB\*
      - Take leftmost VP
      - Take left child

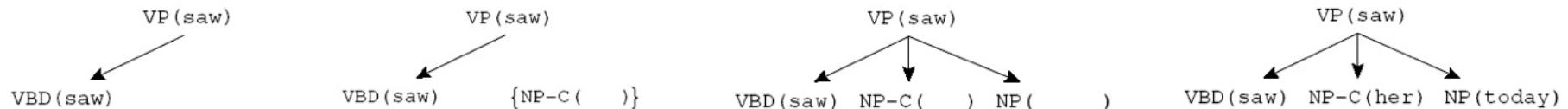


# Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like

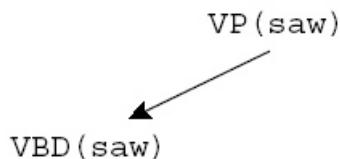
$\text{VP}(\text{saw}) \rightarrow \text{VBD}(\text{saw}) \text{ NP-C(her)} \text{ NP(today)}$

- Never going to get these atomically off of a treebank
- Solution: break up derivation into smaller steps

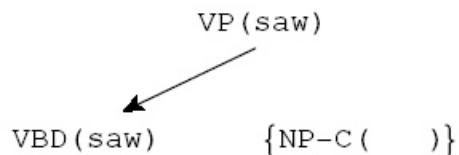


# Lexical Derivation Steps

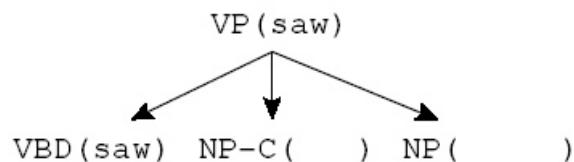
- A derivation of a local tree [Collins '99]



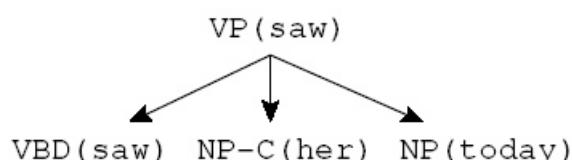
Choose a head tag and word



Choose a complement bag



Generate children (incl. adjuncts)

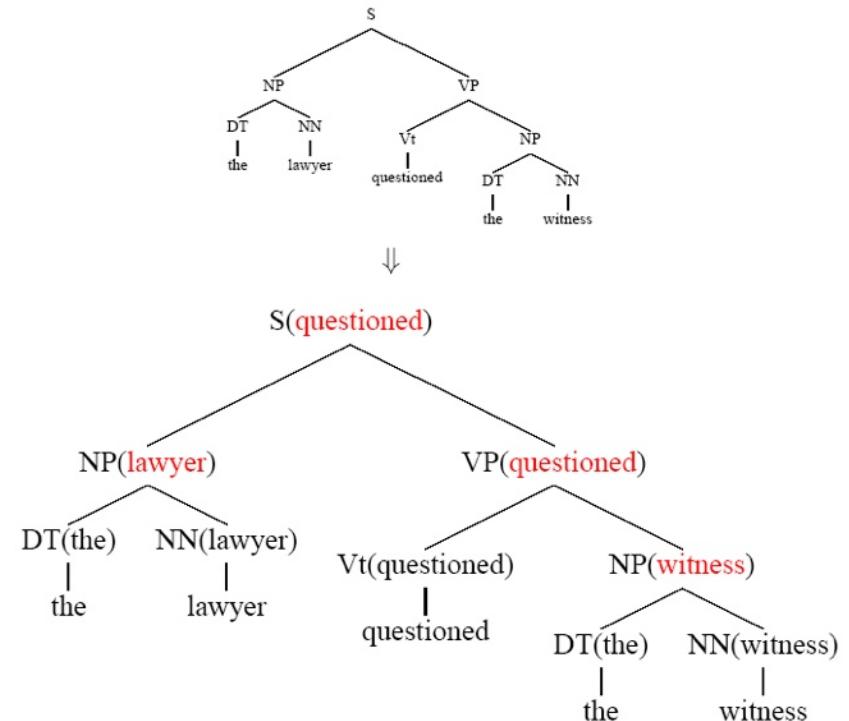


Recursively derive children

# Lexicalized Grammars

- Challenges:

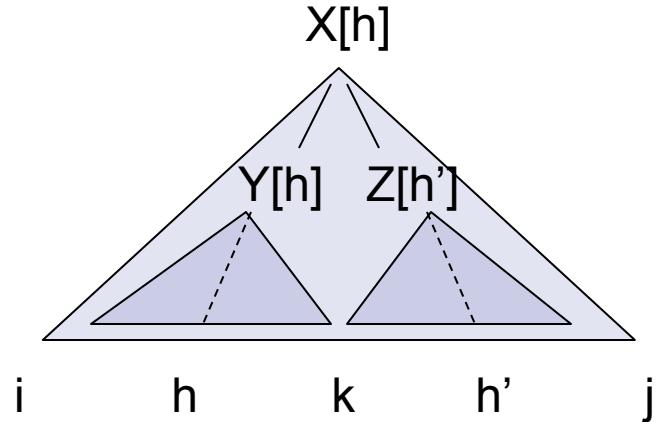
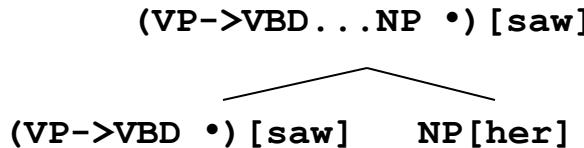
- Many parameters to estimate: requires sophisticated smoothing techniques
- Exact inference is too slow: requires pruning heuristics
- Difficult to adapt to new languages: At least head rules need to be specified, typically more changes needed



Model	F1
Klein&Manning '03	86.3
Charniak '00	90.1

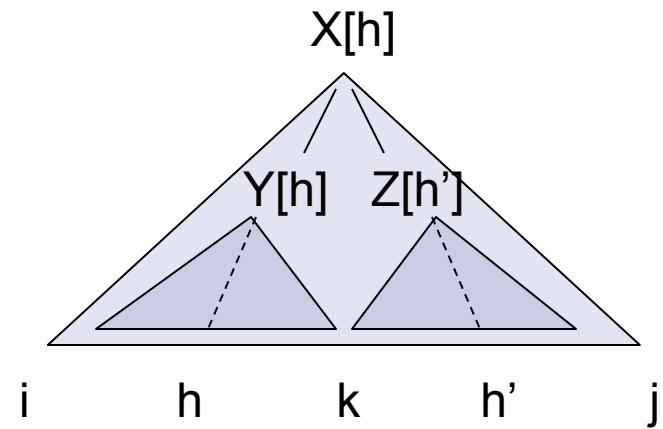
# Lexicalized CKY

```
bestScore(X,i,j,h)
if (j = i+1)
    return tagScore(X,s[i])
else
    return
        max max score(X[h]->Y[h] Z[h']) *
            k,h',X->YZ bestScore(Y,i,k,h) *
                bestScore(Z,k,j,h')
        max score(X[h]->Y[h'] Z[h]) *
            k,h',X->YZ bestScore(Y,i,k,h') *
                bestScore(Z,k,j,h)
```



# Pruning with Beams

- The Collins parser prunes with per-cell beams [Collins 99]
  - Essentially, run the  $O(n^5)$  CKY
  - Remember only a few hypotheses for each span  $\langle i, j \rangle$ .
  - If we keep  $K$  hypotheses at each span, then we do at most  $O(nK^2)$  work per span (why?)
  - Keeps things more or less cubic
- Also: certain spans are forbidden entirely on the basis of punctuation (crucial for speed)



# Pruning with a PCFG

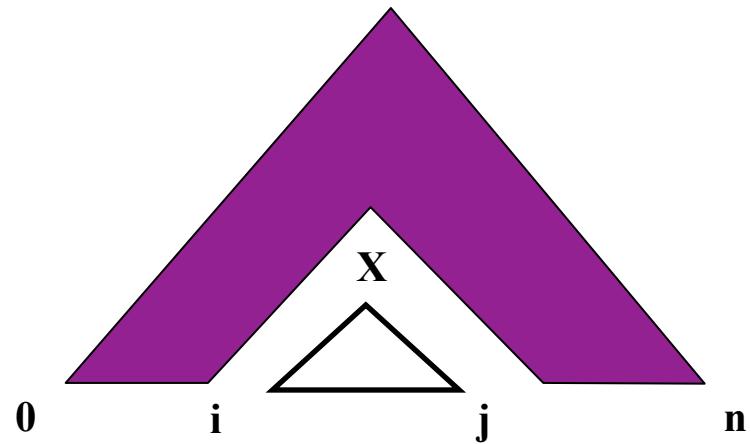
---

- The Charniak parser prunes using a two-pass approach [Charniak 97+]
  - First, parse with the base grammar
  - For each  $X:[i,j]$  calculate  $P(X|i,j,s)$ 
    - This isn't trivial, and there are clever speed ups
  - Second, do the full  $O(n^5)$  CKY
    - Skip any  $X :[i,j]$  which had low (say,  $< 0.0001$ ) posterior
  - Avoids almost all work in the second phase!
- Charniak et al 06: can use more passes
- Petrov et al 07: can use many more passes

# Pruning with A\*

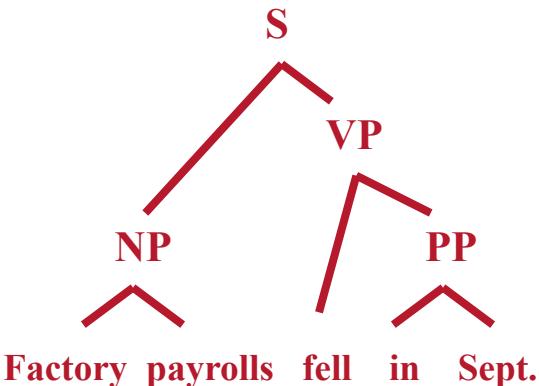
---

- You can also speed up the search without sacrificing optimality
- For agenda-based parsers:
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A\* heuristic, no loss of optimality [Klein and Manning 03]



# Projection-Based A\*

$\pi$  SYNTACTIC



S:fell

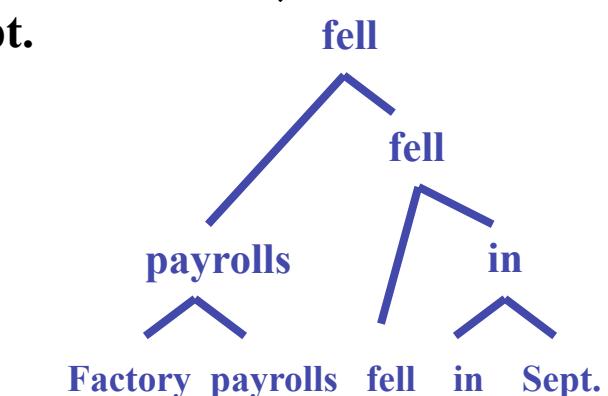
NP:payrolls

VP:fell

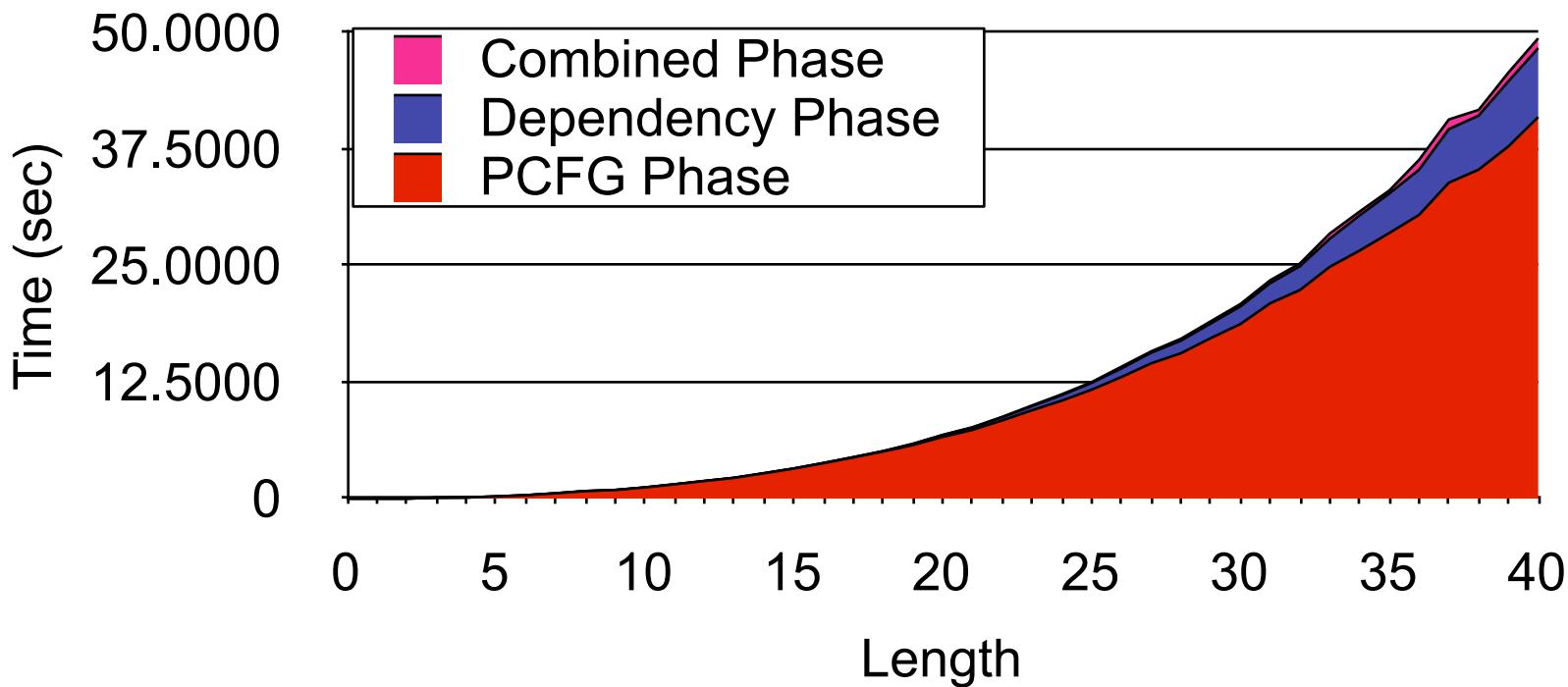
PP:in

Factory payrolls fell in Sept.

$\pi$  SEMANTIC



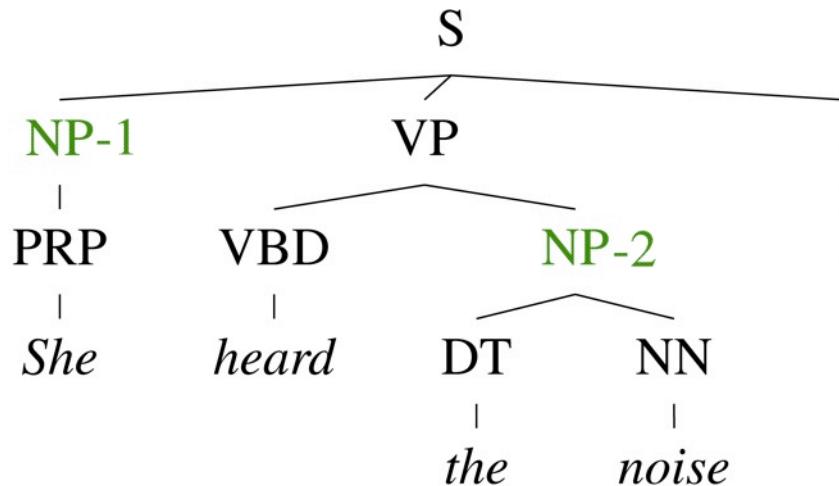
# A\* Speedup



- Total time dominated by calculation of A\* tables in each projection...  $O(n^3)$

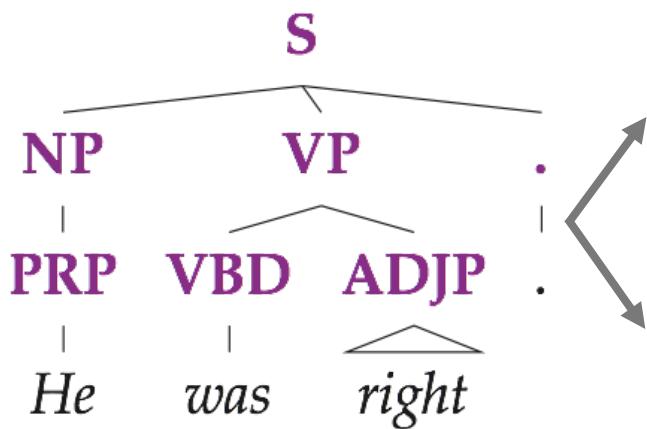
# The Game of Designing a Grammar

---

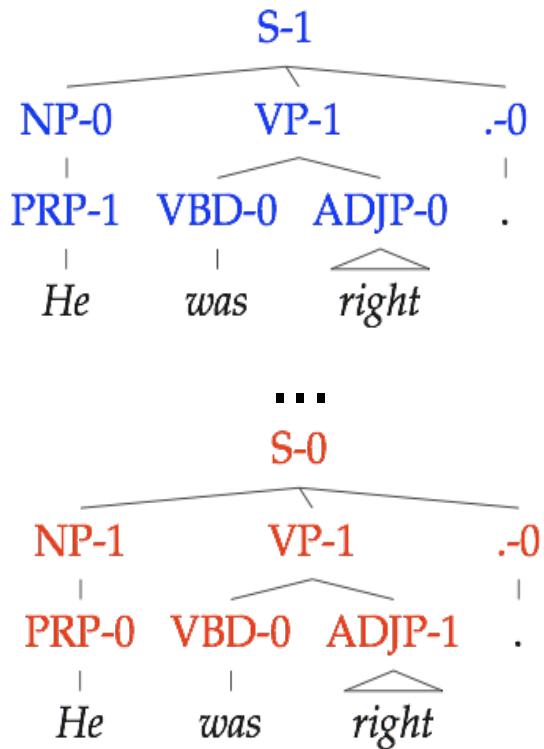


- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Automatic clustering

# Latent Variable Grammars



Parse Tree  $T$   
Sentence  $w$



Derivations  $t : T$

Grammar $G$	
$S_0 \rightarrow NP_0 VP_0$	?
$S_0 \rightarrow NP_1 VP_0$	?
$S_0 \rightarrow NP_0 VP_1$	?
$S_0 \rightarrow NP_1 VP_1$	?
$S_1 \rightarrow NP_0 VP_0$	?
...	
$S_1 \rightarrow NP_1 VP_1$	?
...	
$NP_0 \rightarrow PRP_0$	?
$NP_0 \rightarrow PRP_1$	?
...	

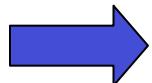
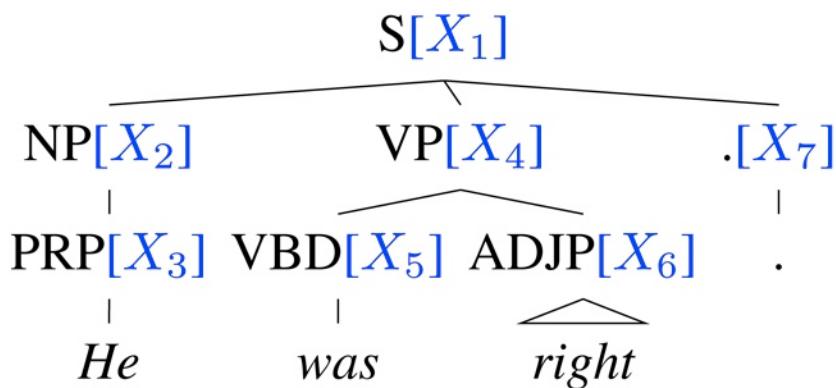
Lexicon	
$PRP_0 \rightarrow She$	?
$PRP_1 \rightarrow She$	?
...	
$VBD_0 \rightarrow was$	?
$VBD_1 \rightarrow was$	?
$VBD_2 \rightarrow was$	?
...	

Parameters  $\theta$

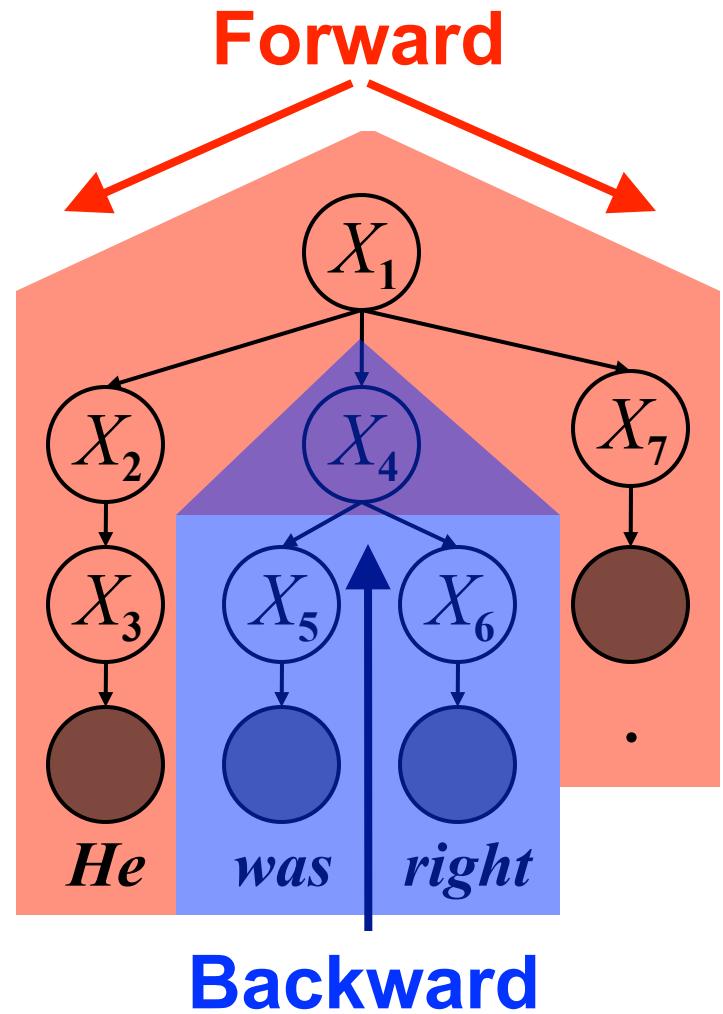
# Learning Latent Annotations

EM algorithm:

- Brackets are known
- Base categories are known
- Only induce subcategories

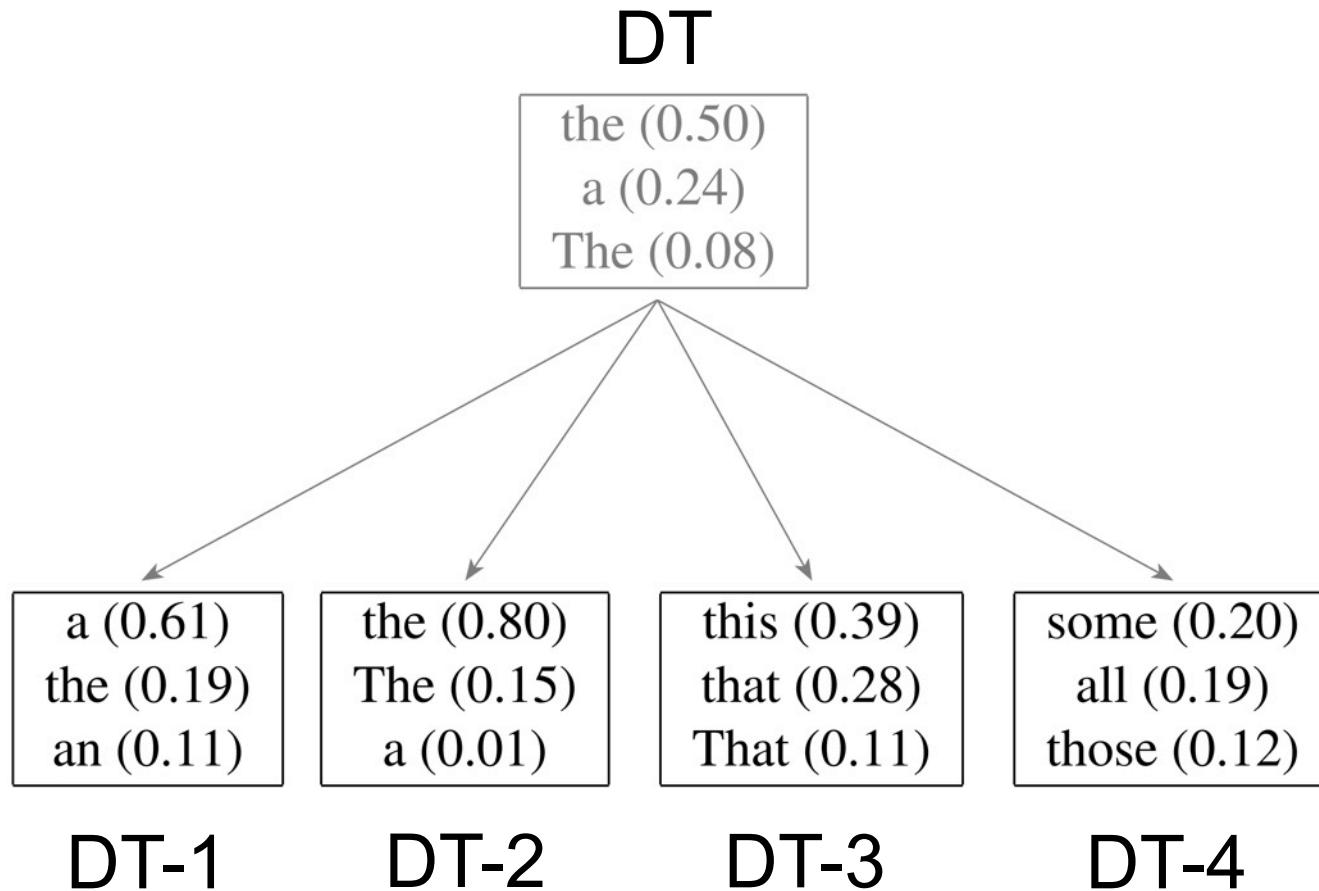


Just like Forward-Backward  
for HMMs.



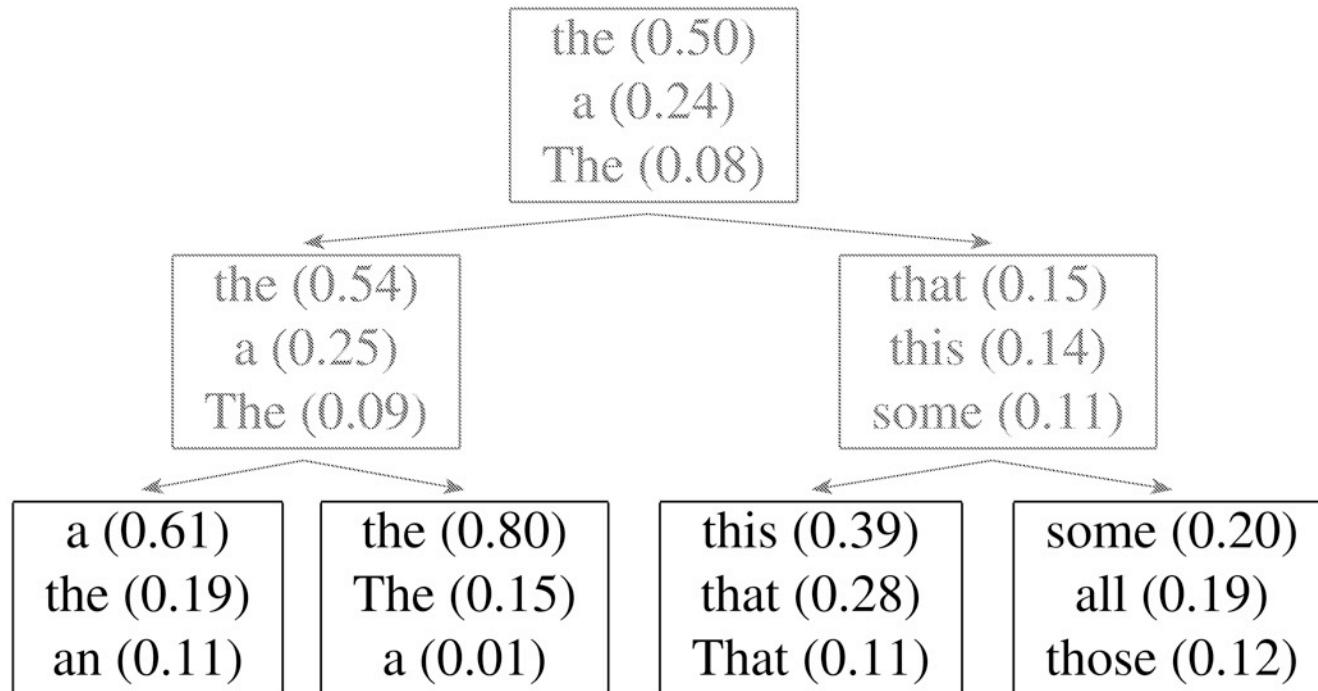
# Refinement of the DT tag

---

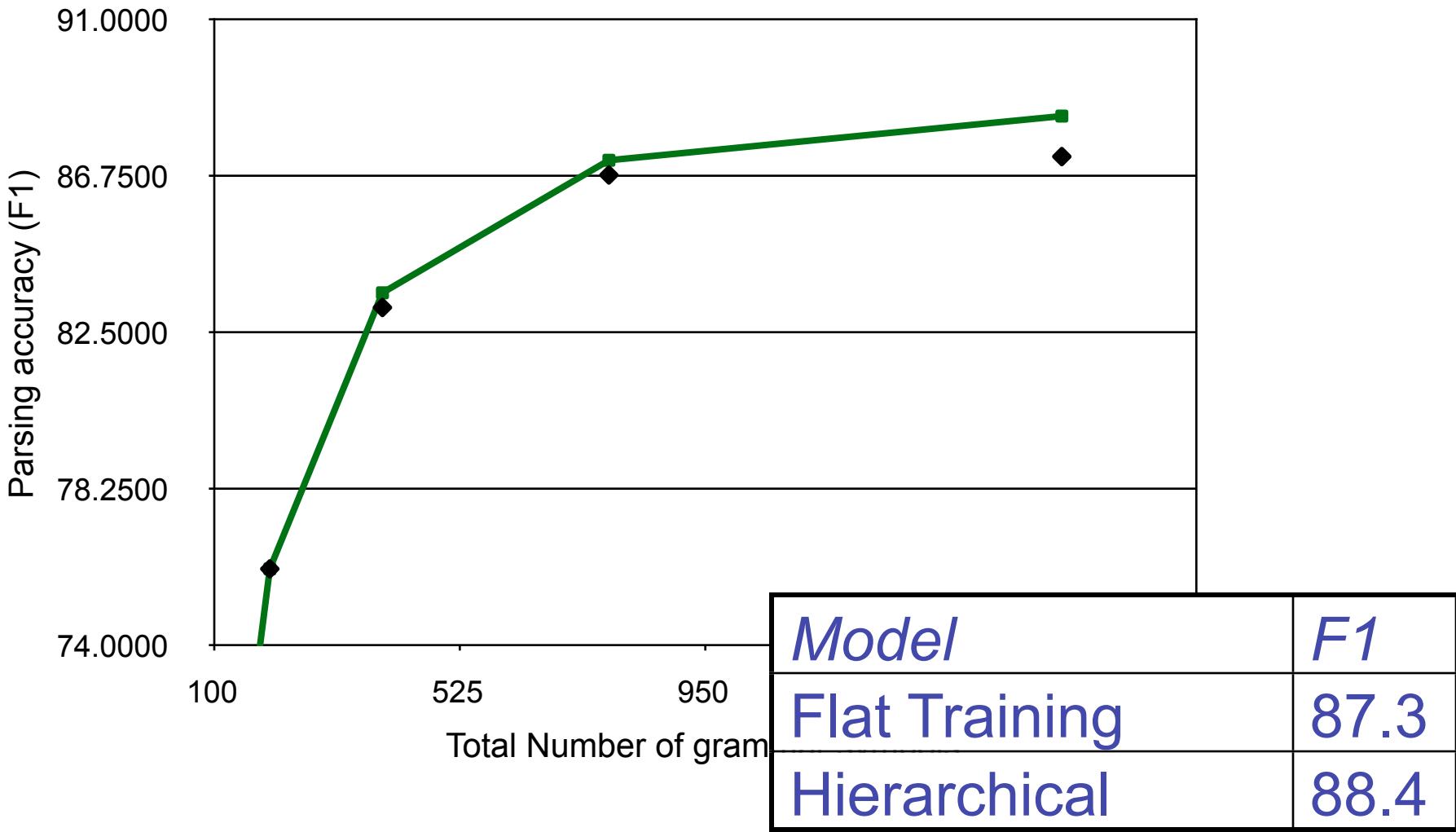


# Hierarchical Refinement

---



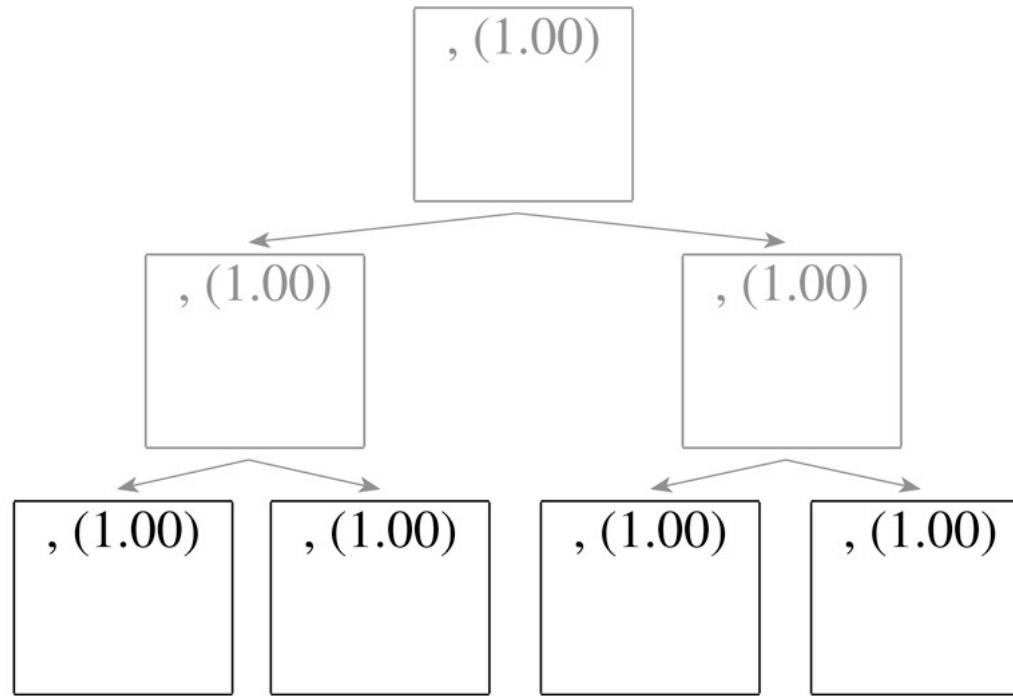
# Hierarchical Estimation Results



# Refinement of the , tag

---

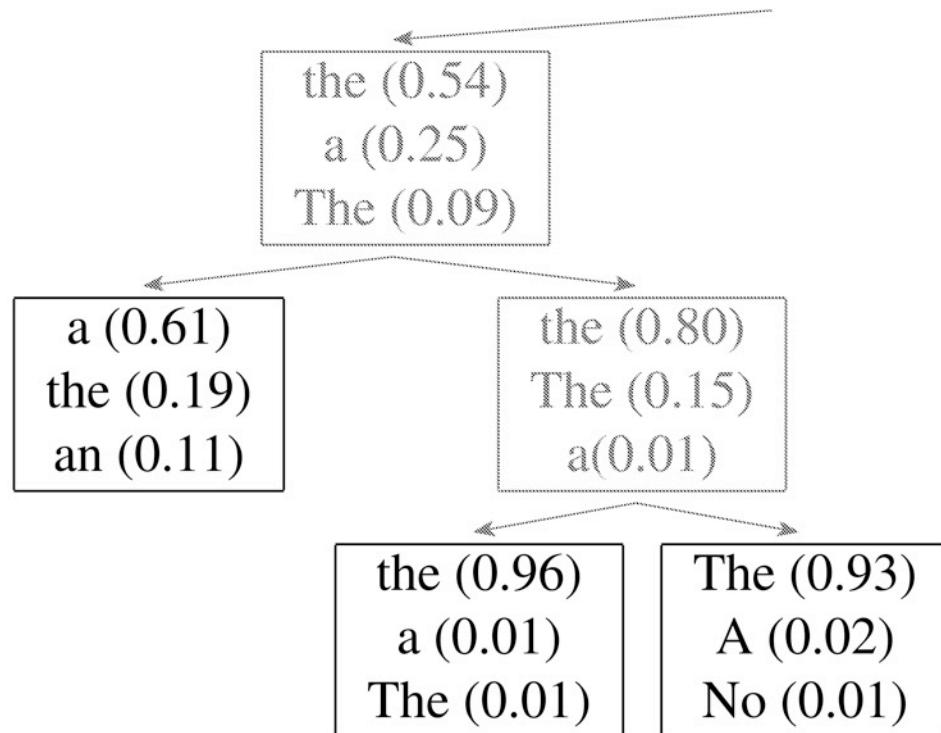
- Splitting all categories equally is wasteful:



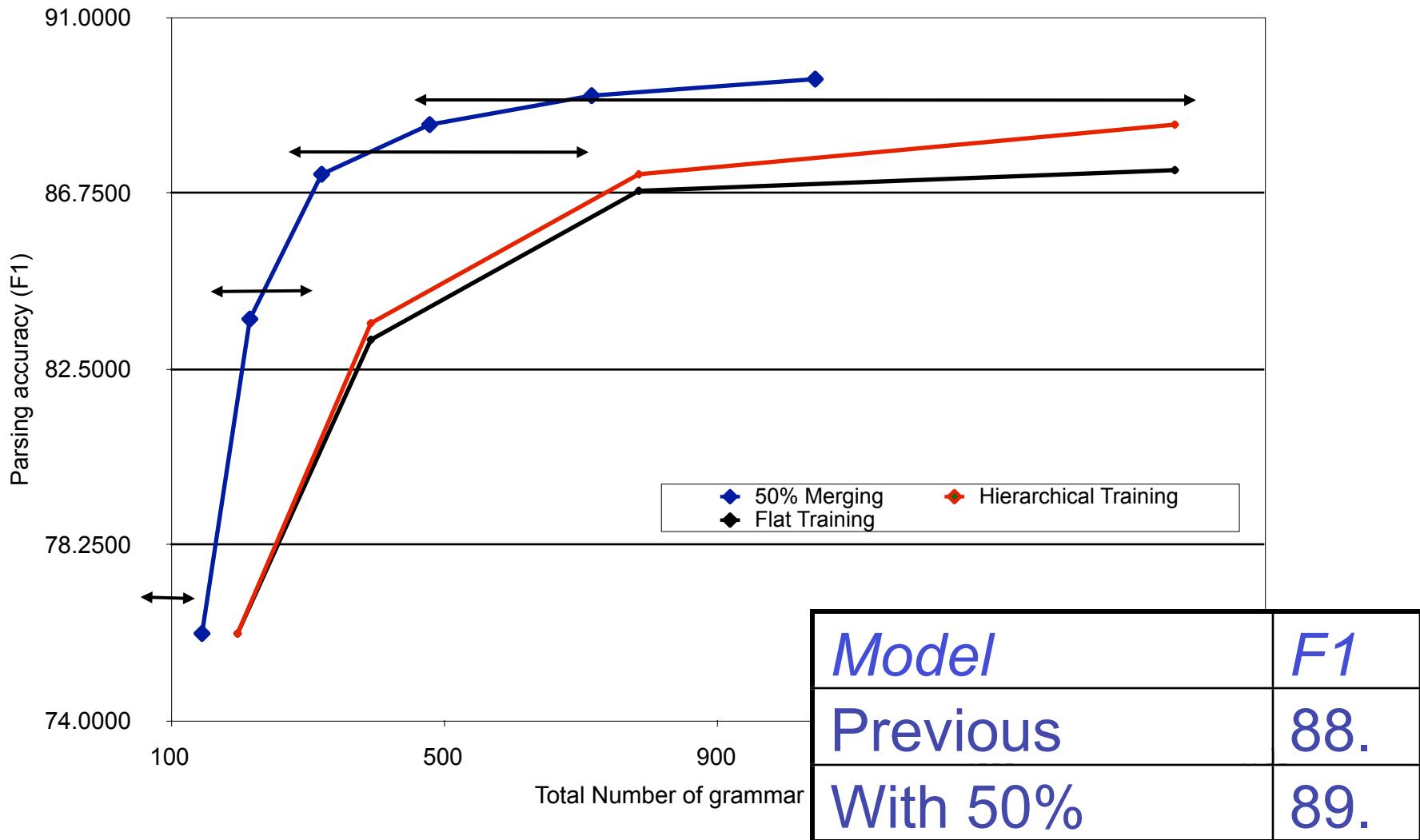
# Adaptive Splitting

---

- Want to split complex categories more
- Idea: split everything, roll back splits which were least useful

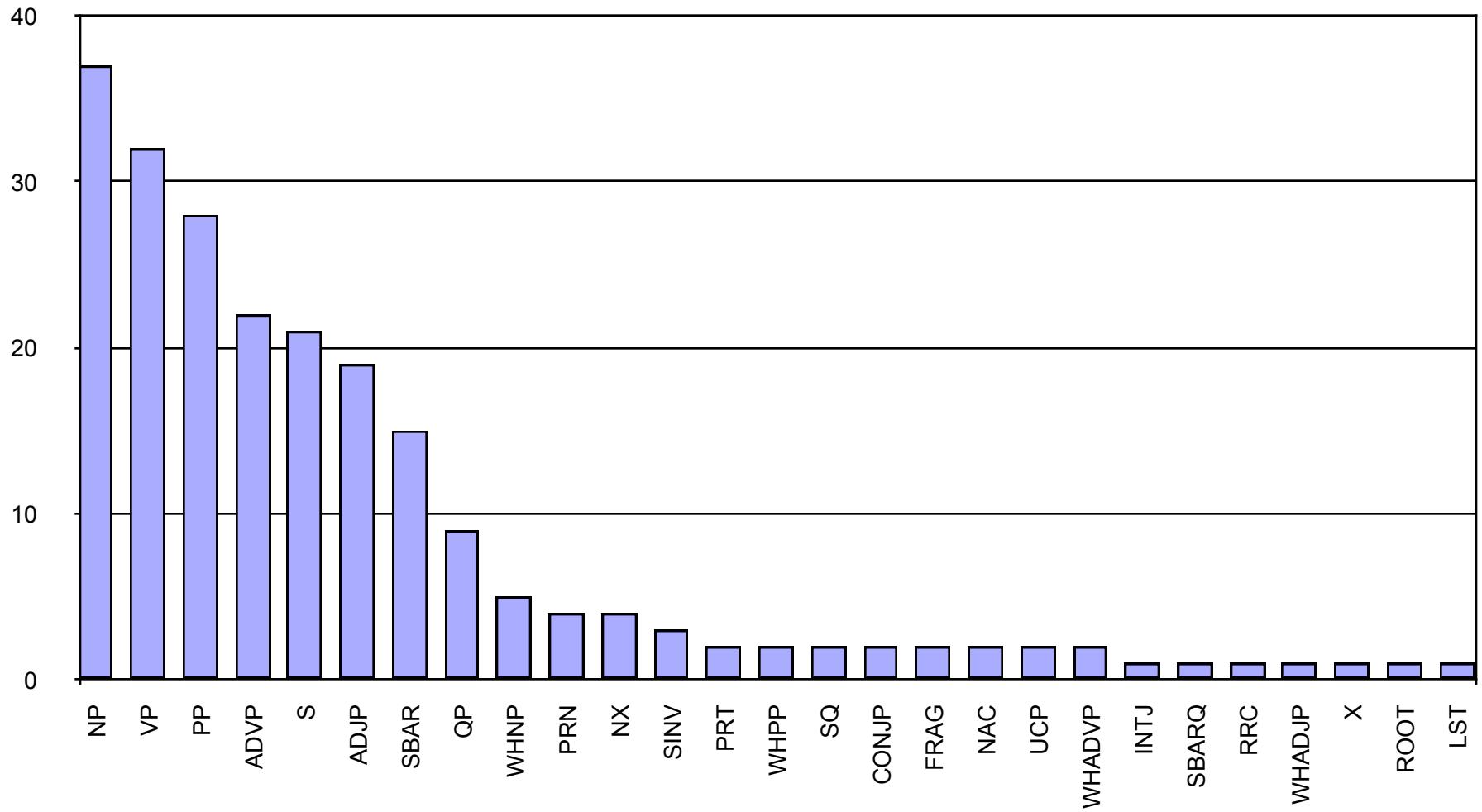


# Adaptive Splitting Results



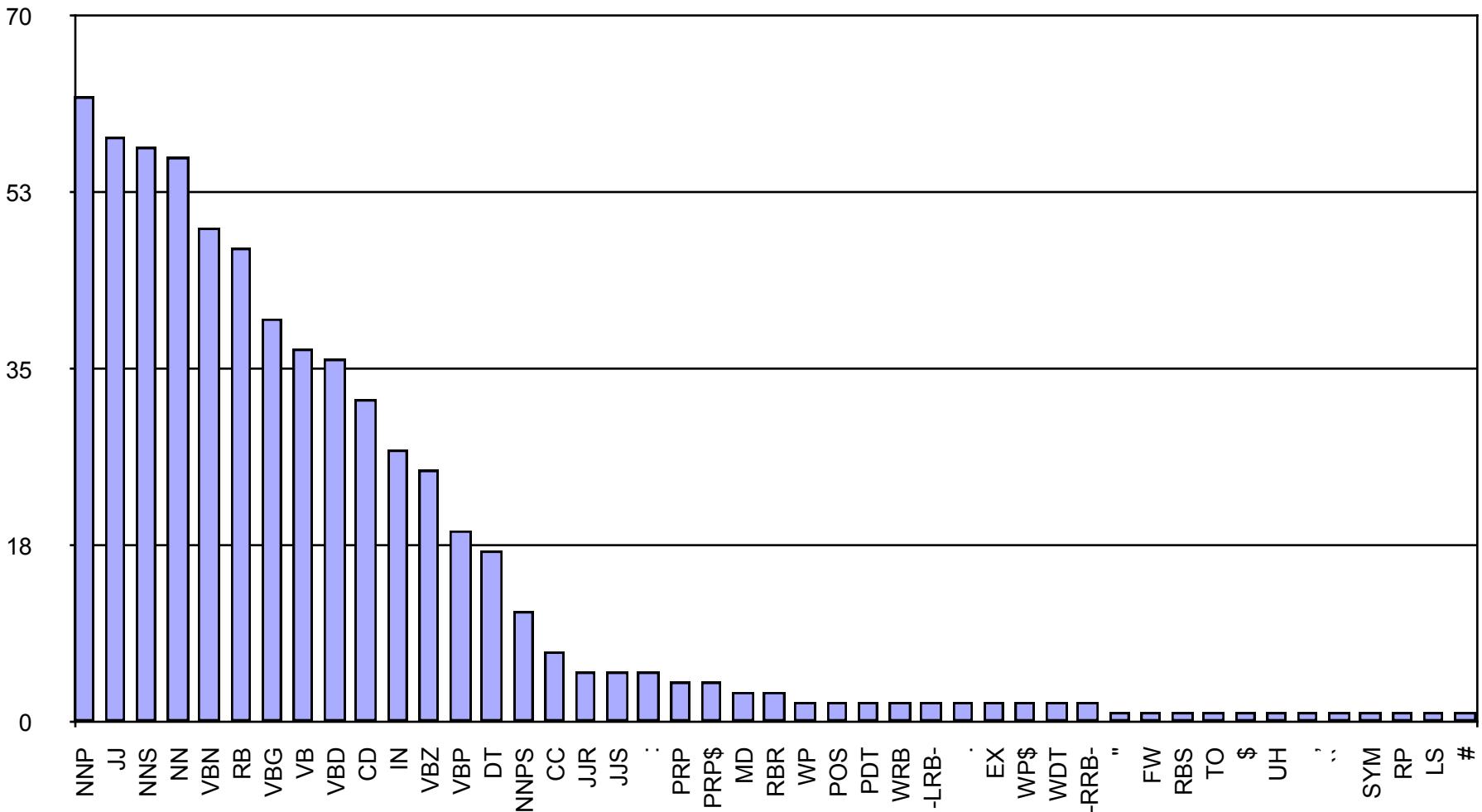
# Number of Phrasal Subcategories

---



# Number of Lexical Subcategories

---



# Learned Splits

---

- 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

# Learned Splits

---

- Relative adverbs (RBR):

RBR-0	Further	lower	higher
RBR-1	more	less	More
RBR-2	earlier	Earlier	later

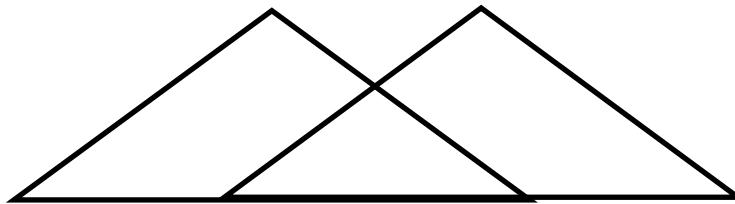
- Cardinal Numbers (CD):

CD-7	one	two	Three
CD-4	1989	1990	1988
CD-11	million	billion	trillion
CD-0	1	50	100
CD-3	1	30	31
CD-9	78	58	34

# Coarse-to-Fine Inference

---

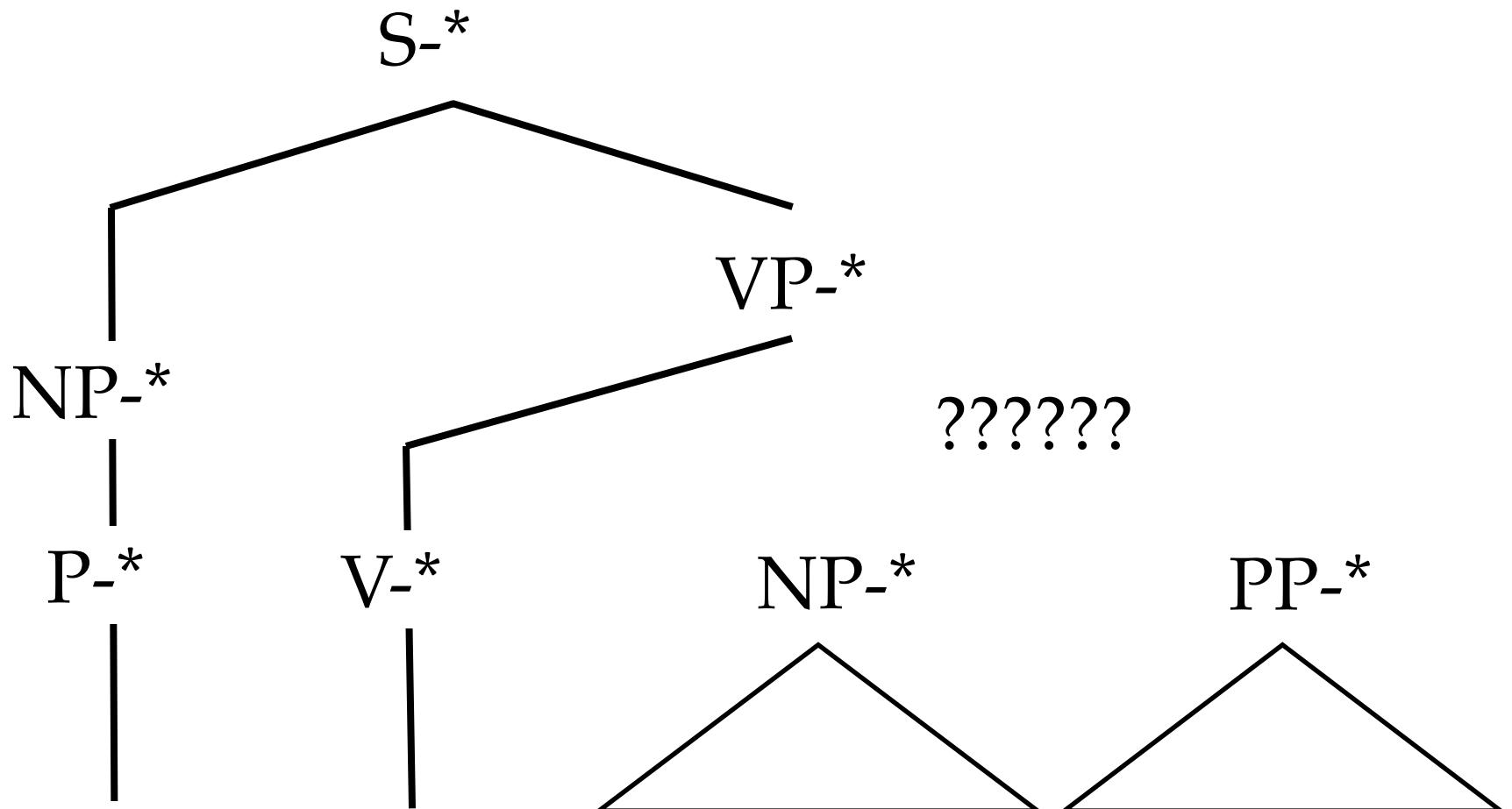
NP-~~31~~ RP-~~31~~



*They solved the problem with statistics .*

# Coarse-to-Fine Inference

---



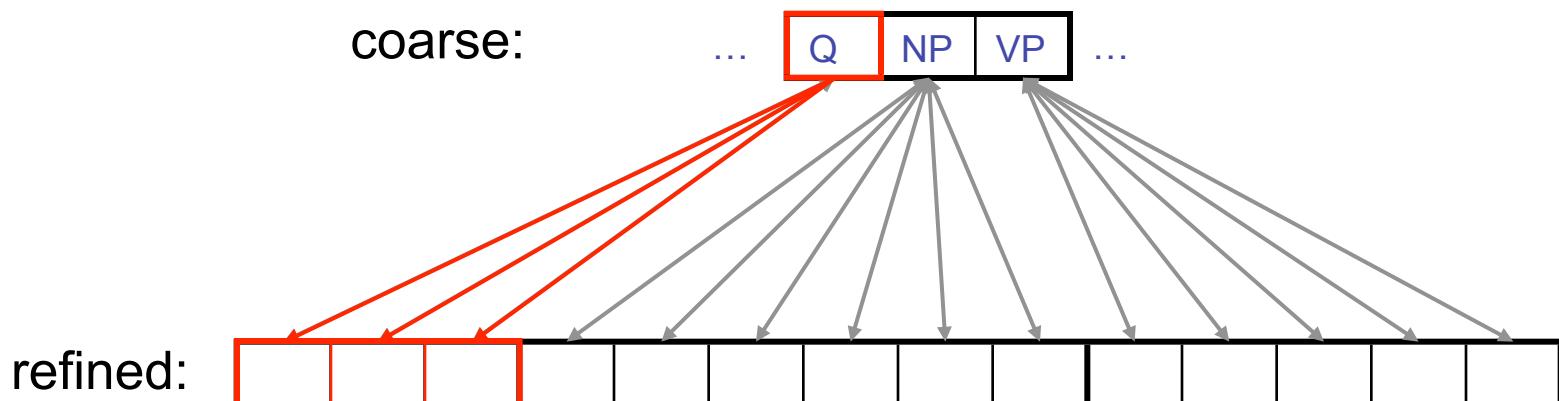
*They solved the problem with statistics .*

# Prune?

For each chart item  $X[i,j]$ , compute posterior probability:

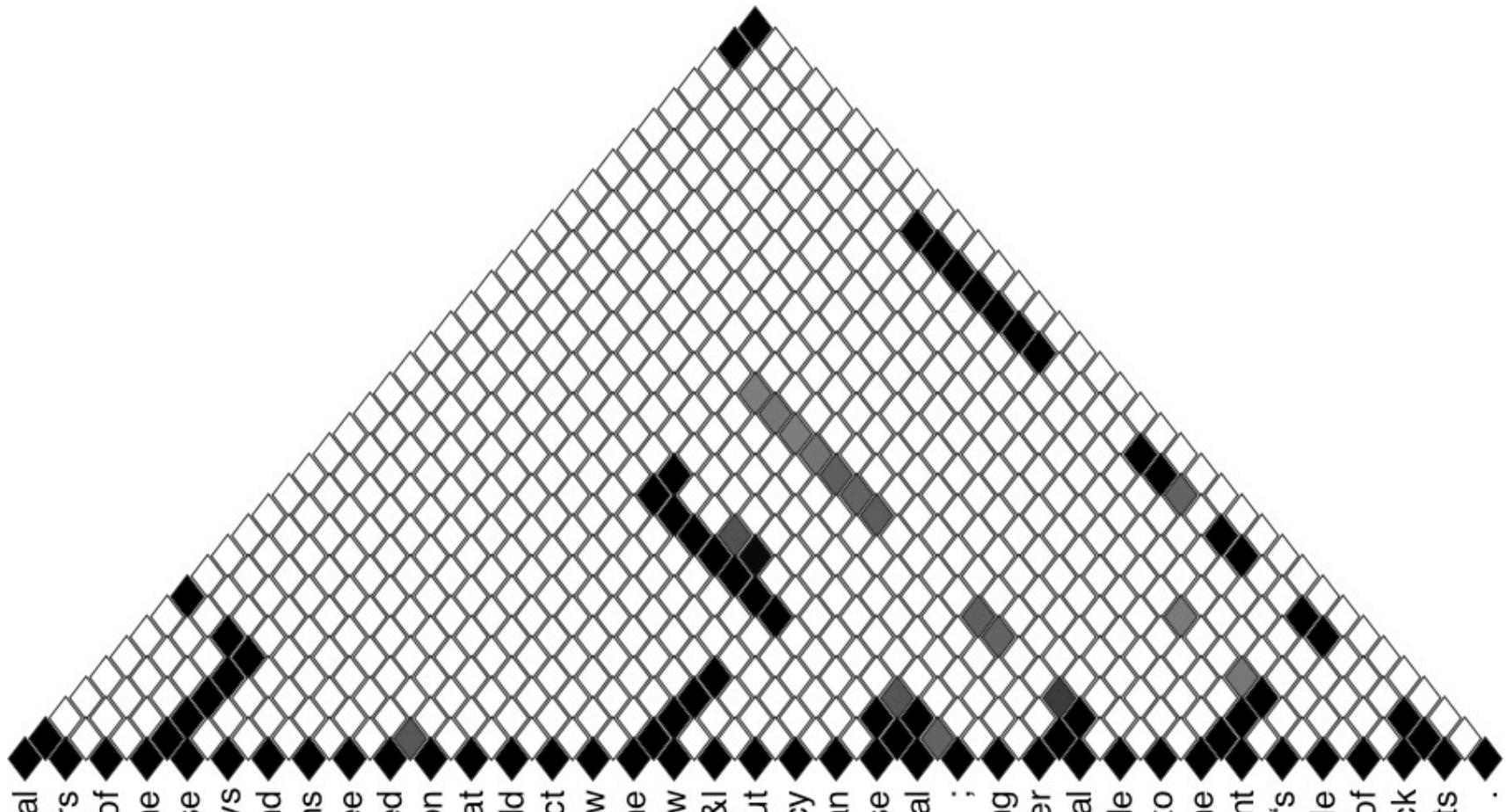
$$\frac{P_{\text{IN}}(X, i, j) \cdot P_{\text{OUT}}(X, i, j)}{P_{\text{IN}}(\text{root}, 0, n)} < \text{threshold}$$

E.g. consider the span 5 to 12:



# Bracket Posteriors

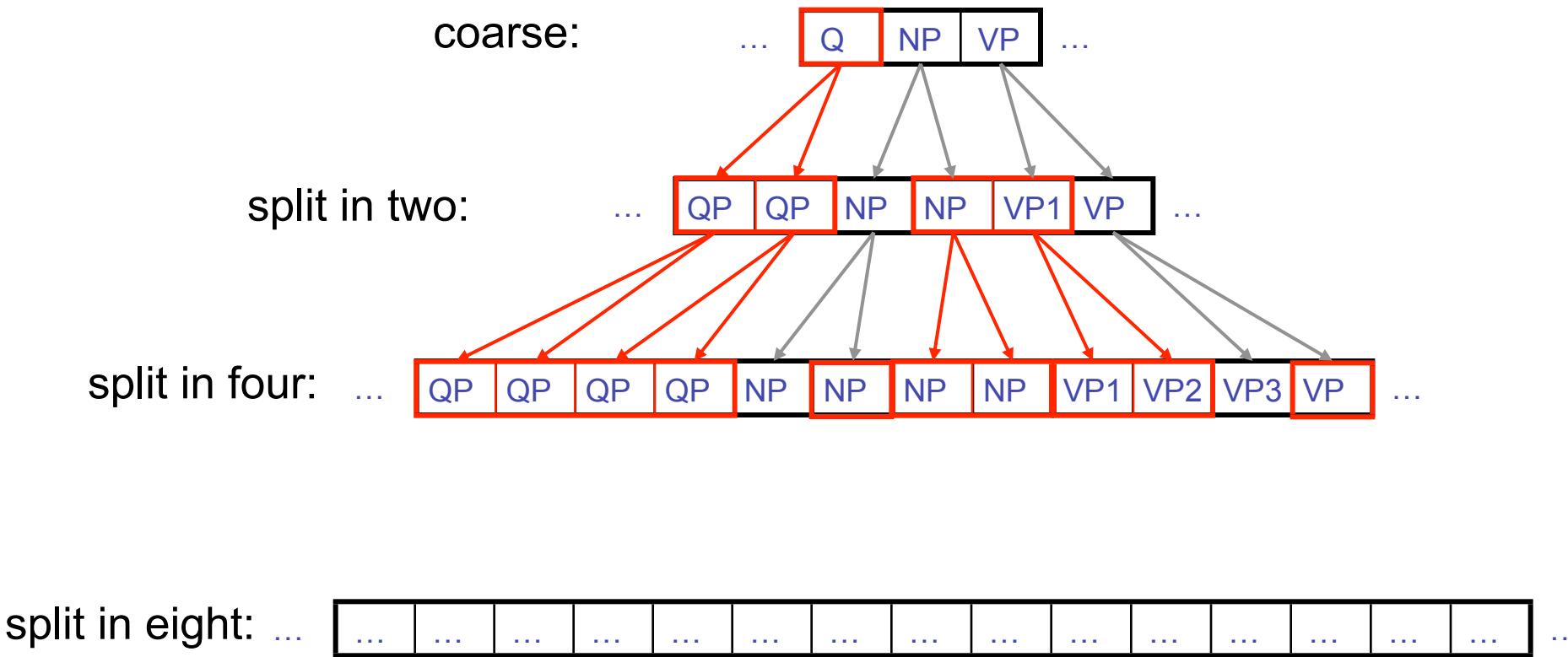
---



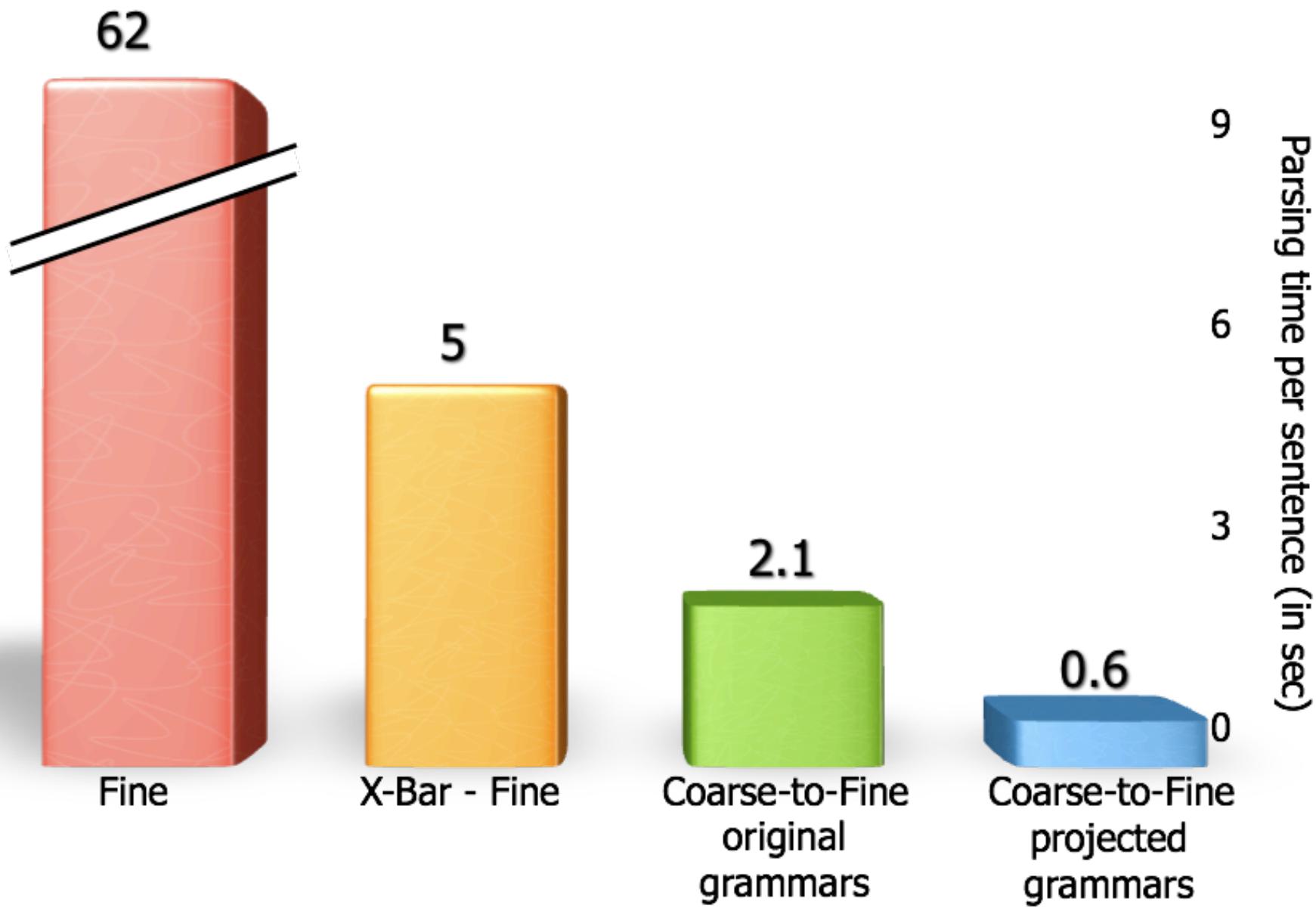
Influential members of the House Ways and Means Committee introduced legislation that would restrict how the new S&L bailout agency can raise capital ; creating another potential obstacle to the government 's sale of sick thrifts

# Hierarchical Pruning

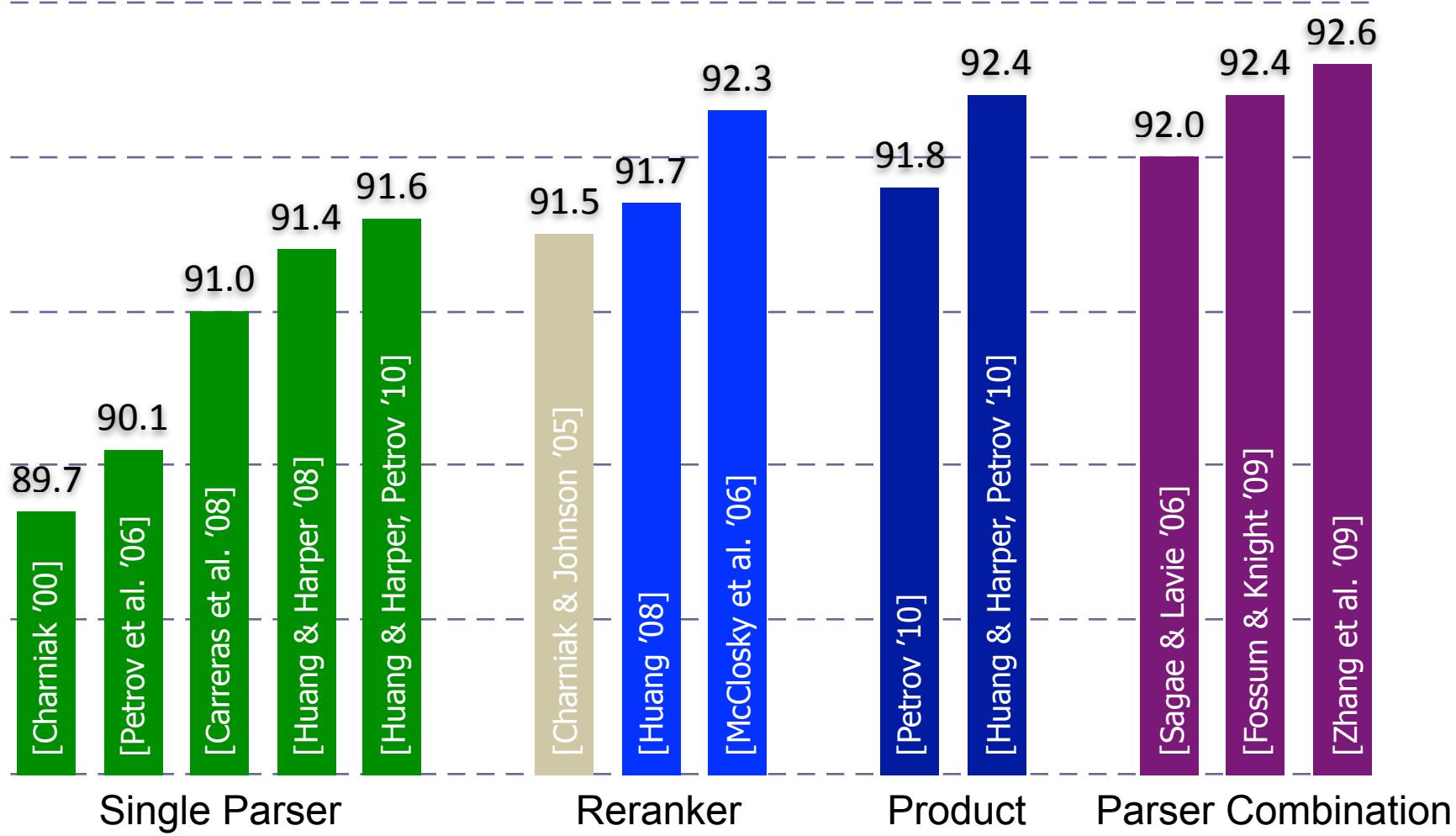
---



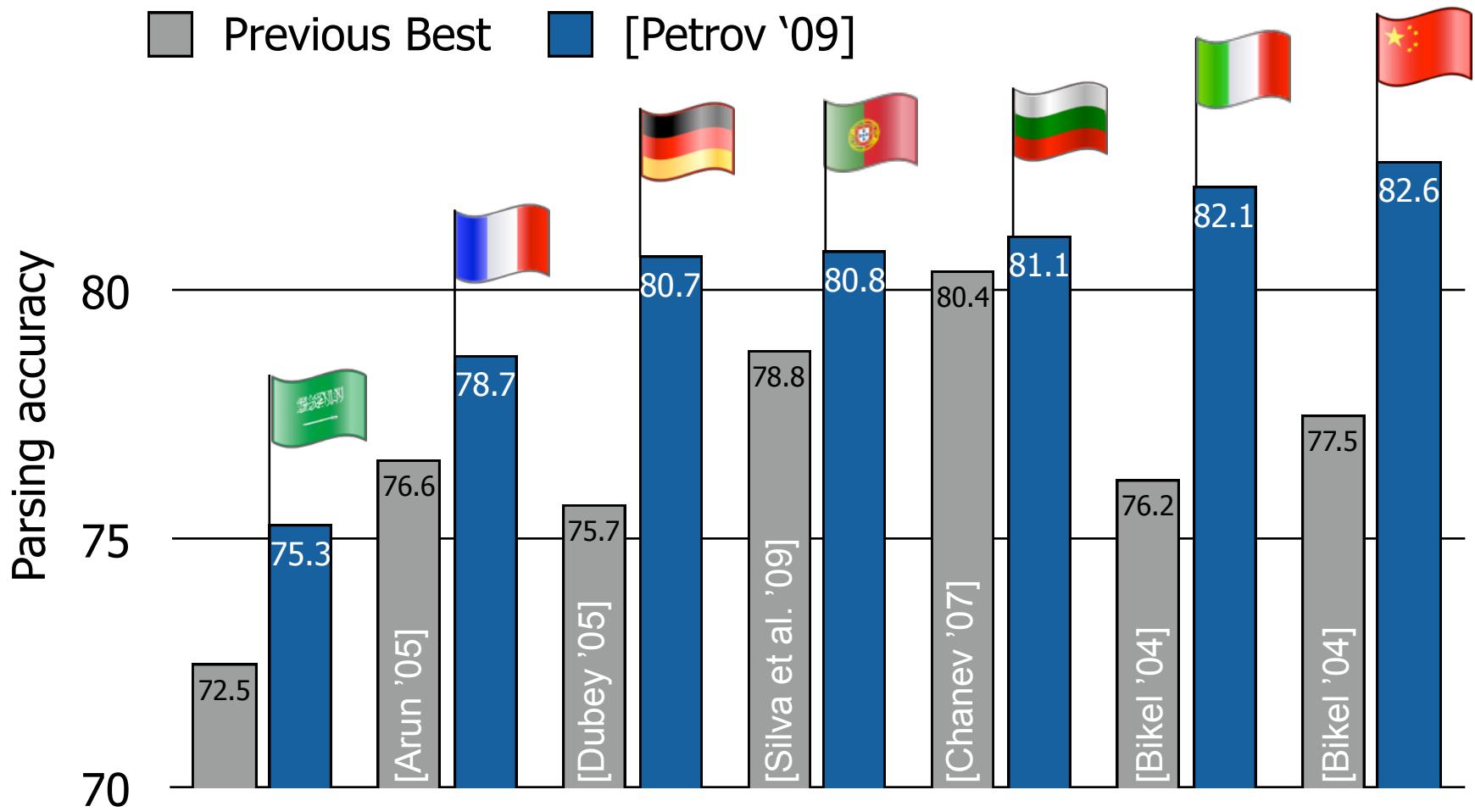
# Parsing Times (per sentence)



# Detailed English Results

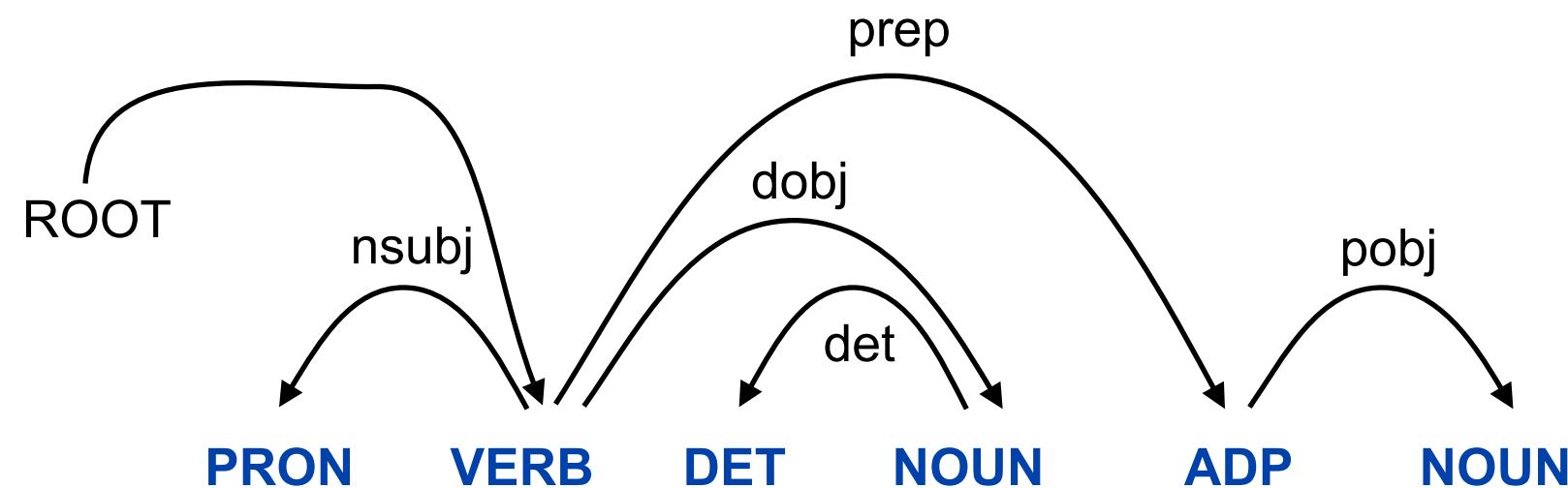


# Multi-Lingual Results



# Dependency Parsing

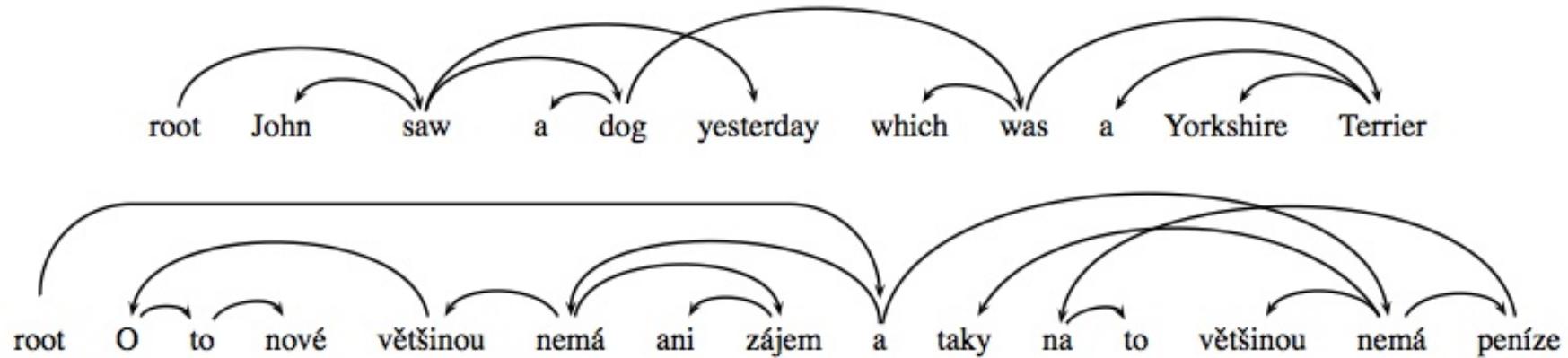
---



*They solved the problem with statistics .*

# (Non-)Projectivity

- Crossing Arcs needed to account for non-projective constructions
- Fairly rare in English but can be common in other languages (e.g. Czech):



*He is mostly not even interested in the new things and in most cases, he has no money for it either.*

# Formal Conditions

---

- ▶ For a dependency graph  $G = (V, A)$
- ▶ With label set  $L = \{l_1, \dots, l_{|L|}\}$
- ▶  $G$  is (weakly) **connected**:
  - ▶ If  $i, j \in V$ ,  $i \leftrightarrow^* j$ .
- ▶  $G$  is **acyclic**:
  - ▶ If  $i \rightarrow j$ , then not  $j \rightarrow^* i$ .
- ▶  $G$  obeys the **single-head** constraint:
  - ▶ If  $i \rightarrow j$ , then not  $i' \rightarrow j$ , for any  $i' \neq i$ .
- ▶  $G$  is **projective**:
  - ▶ If  $i \rightarrow j$ , then  $i \rightarrow^* i'$ , for any  $i'$  such that  $i < i' < j$  or  $j < i' < i$ .

# Arc-Factored Models

---

- ▶ Assumes that the score / probability / **weight** of a dependency graph factors by its arcs

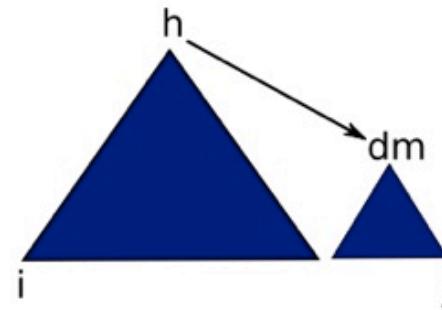
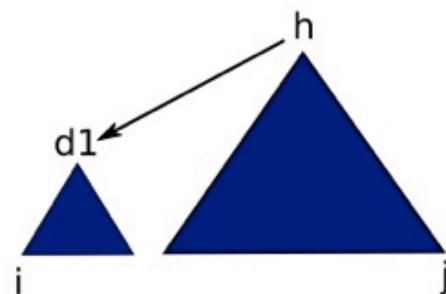
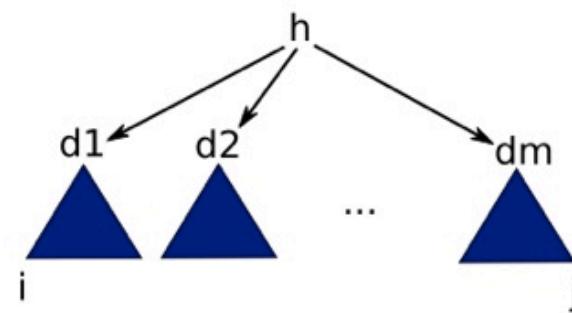
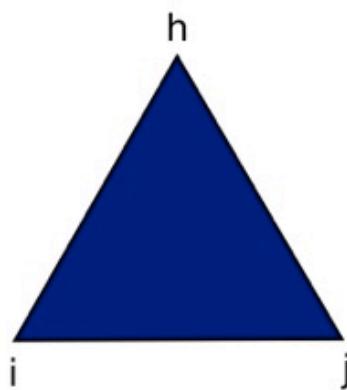
$$w(G) = \prod_{(i,j,k) \in G} w_{ij}^k \quad \text{look familiar?}$$

- ▶  $w_{ij}^k$  is the weight of creating a dependency from word  $w_i$  to  $w_j$  with label  $l_k$
- ▶ Thus there is an assumption that each dependency decision is independent
  - ▶ Strong assumption! Will address this later.

# Arc-factored Projective Parsing

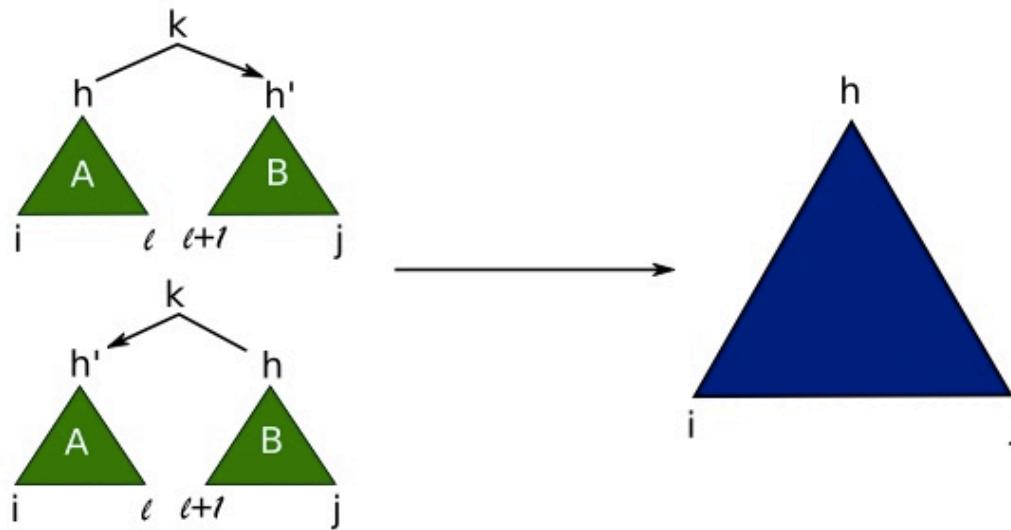
---

- All projective graphs can be written as the combination of two smaller **adjacent** graphs



# Arc-factored Projective Parsing

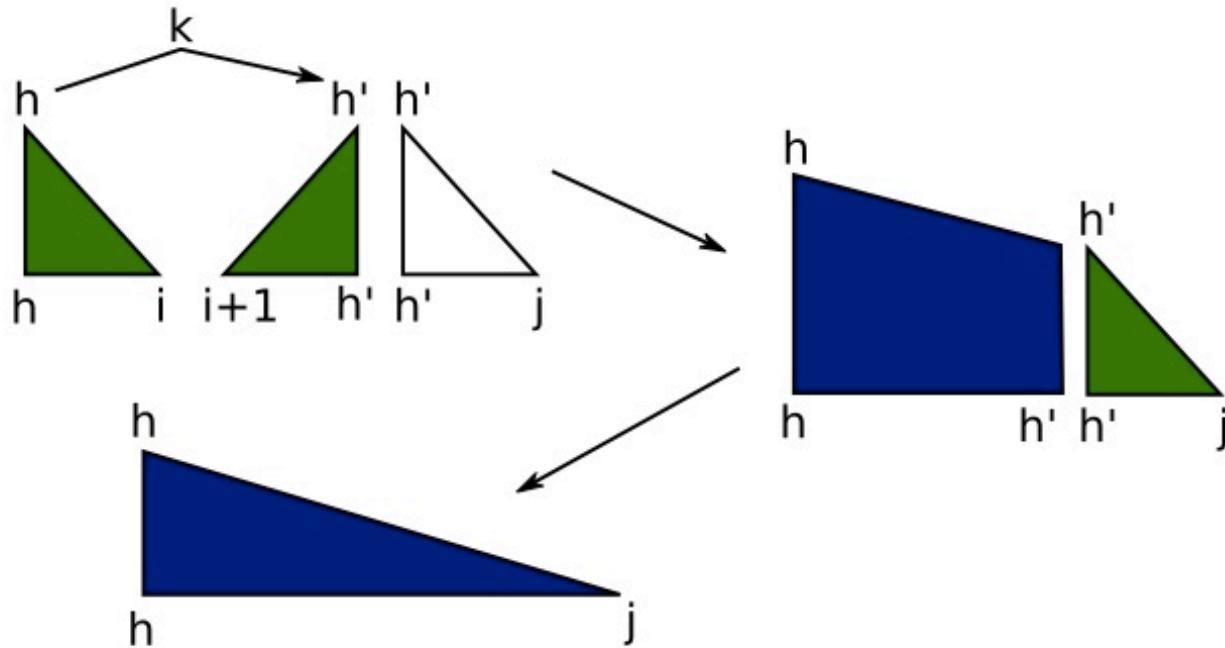
- ▶ Chart item filled in a bottom-up manner
  - ▶ First do all strings of length 1, then 2, etc. just like CKY



- ▶ Weight of new item:  $\max_{I,j,k} w(A) \times w(B) \times w_{hh'}^k$
- ▶ Algorithm runs in  $O(|L|n^5)$
- ▶ Use back-pointers to extract best parse (like CKY)

# Eisner Algorithm

- ▶  $O(|L|n^5)$  is not that good
- ▶ [Eisner 1996] showed how this can be reduced to  $O(|L|n^3)$ 
  - ▶ Key: split items so that sub-roots are always on periphery



# Eisner Algorithm PseudoCode

---

Initialization:  $C[s][s][d][c] = 0.0 \quad \forall s, d, c$

for  $k : 1..n$

    for  $s : 1..n$

$t = s + k$

        if  $t > n$  then break

        % First: create incomplete items

$C[s][t][\leftarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(t, s))$

$C[s][t][\rightarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(s, t))$

        % Second: create complete items

$C[s][t][\leftarrow][1] = \max_{s \leq r < t} (C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0])$

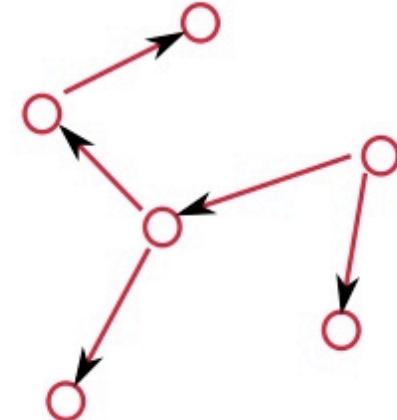
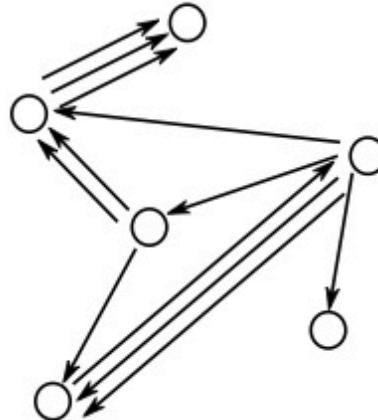
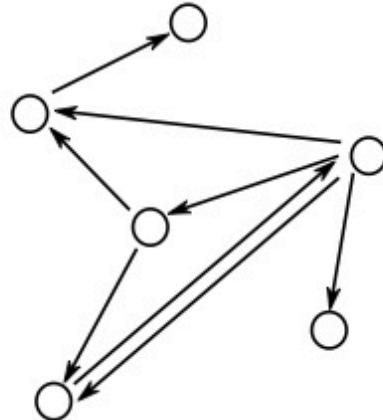
$C[s][t][\rightarrow][1] = \max_{s < r \leq t} (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1])$

    end for

end for

# Maximum Spanning Trees (MSTs)

- ▶ A directed spanning tree of a (multi-)digraph  $G = (V, A)$ , is a subgraph  $G' = (V', A')$  such that:
  - ▶  $V' = V$
  - ▶  $A' \subseteq A$ , and  $|A'| = |V'| - 1$
  - ▶  $G'$  is a tree (acyclic)
- ▶ A spanning tree of the following (multi-)digraphs

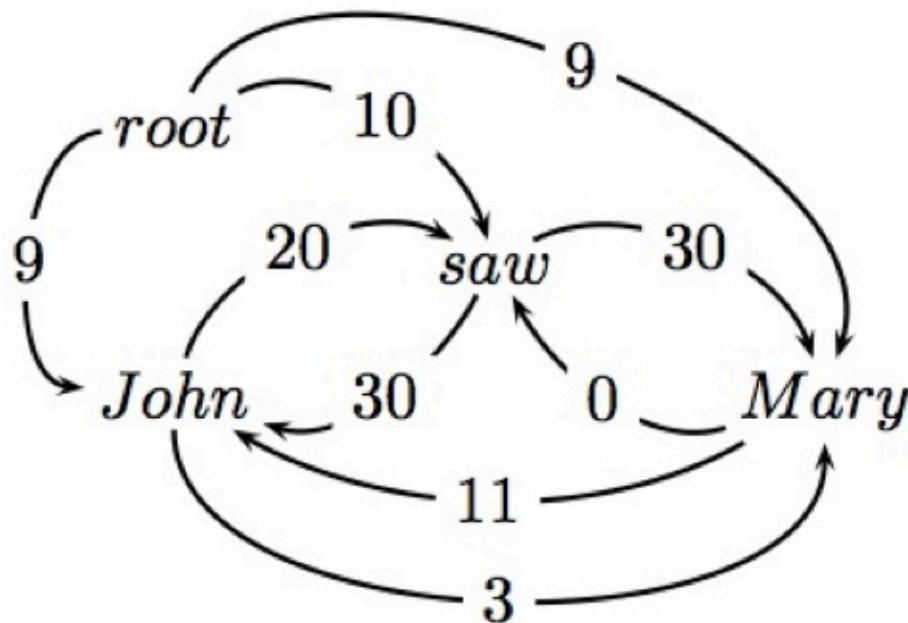


**Can use MST algorithms for nonprojective parsing!**

# Chu-Liu-Edmonds

---

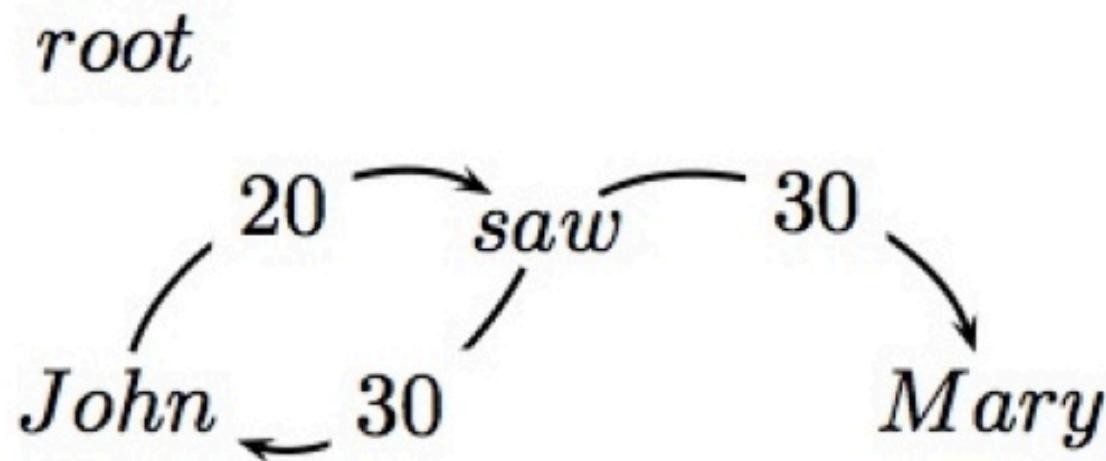
- ▶  $x = \text{root John saw Mary}$



# Chu-Liu-Edmonds

---

- ▶ Find highest scoring incoming arc for each vertex

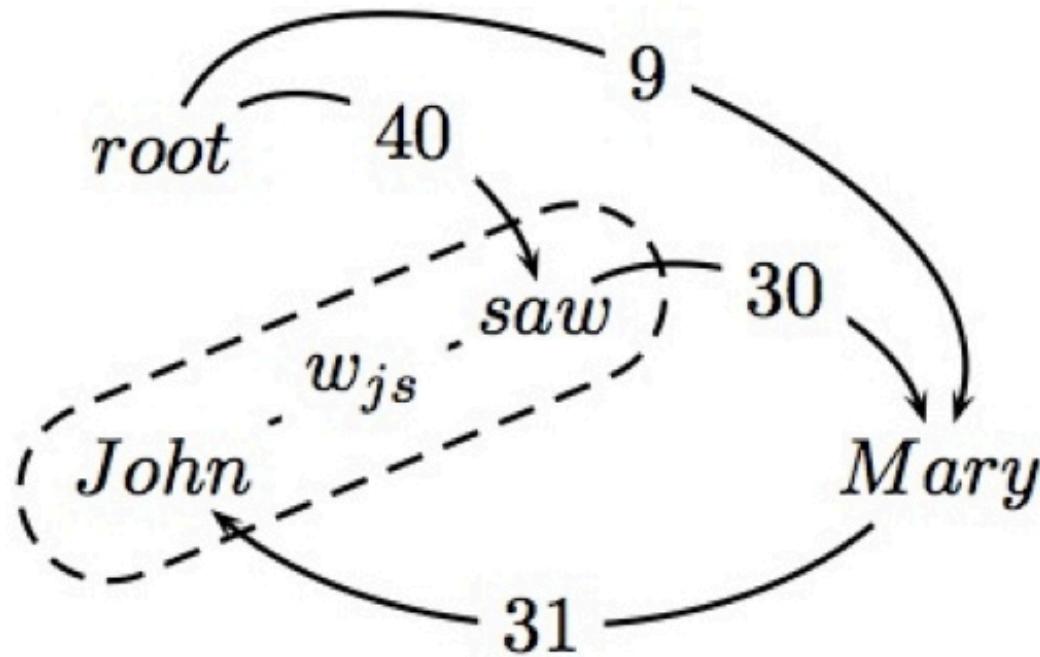


- ▶ If this is a tree, then we have found MST!!

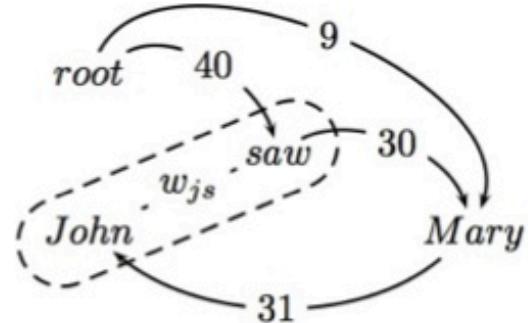
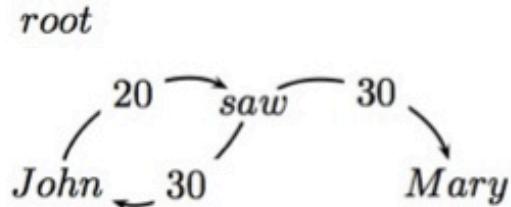
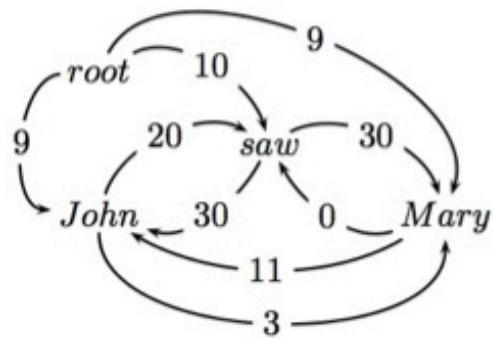
# Find Cycle and Contract

---

- ▶ If not a tree, identify cycle and contract
- ▶ Recalculate arc weights into and out-of cycle



# Recalculate Edge Weights



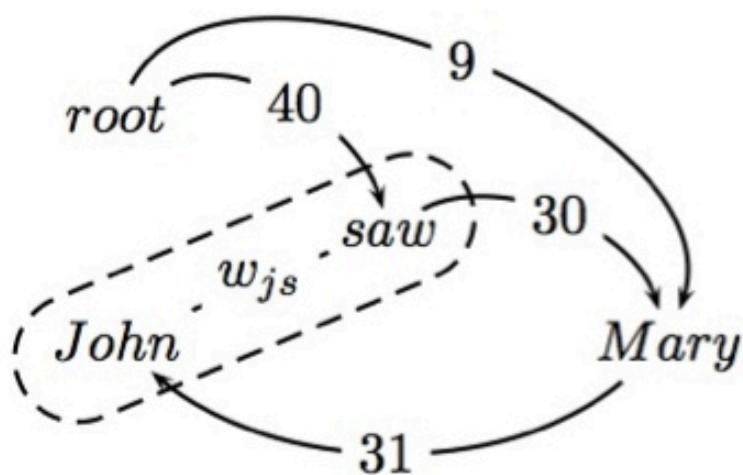
## ► Incoming arc weights

- ▶ Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
- ▶  $\text{root} \rightarrow \text{saw} \rightarrow \text{John}$  is 40 (\*\*)
- ▶  $\text{root} \rightarrow \text{John} \rightarrow \text{saw}$  is 29

# Theorem

---

The weight of the MST of this contracted graph is equal to the weight of the MST for the original graph

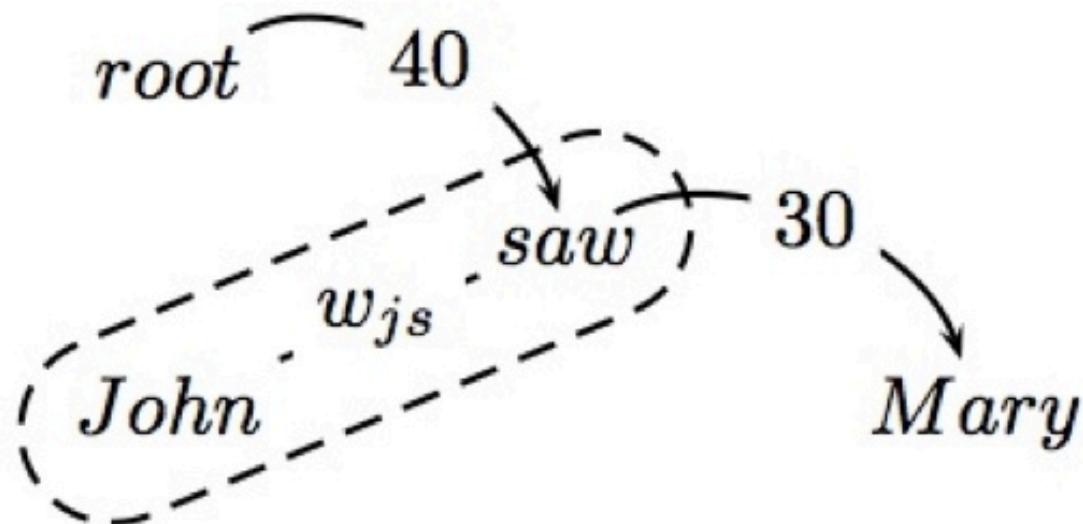


- ▶ Therefore, recursively call algorithm on new graph

# Final MST

---

- ▶ This is a tree and the MST for the contracted graph!!



- ▶ Go back up recursive call and reconstruct final graph

# Chu-Liu-Edmonds PseudoCode

---

**Chu-Liu-Edmonds( $G_x, w$ )**

1. Let  $M = \{(i^*, j) : j \in V_x, i^* = \arg \max_{i'} w_{ij}\}$
2. Let  $G_M = (V_x, M)$
3. If  $G_M$  has no cycles, then it is an MST: return  $G_M$
4. Otherwise, find a cycle  $C$  in  $G_M$
5. Let  $\langle G_C, c, ma \rangle = \text{contract}(G, C, w)$
6. Let  $G = \text{Chu-Liu-Edmonds}(G_C, w)$
7. Find vertex  $i \in C$  such that  $(i', c) \in G$  and  $ma(i', c) = i$
8. Find arc  $(i'', i) \in C$
9. Find all arc  $(c, i''') \in G$
10.  $G = G \cup \{(ma(c, i'''), i''')\}_{\forall (c, i''') \in G} \cup C \cup \{(i', i)\} - \{(i'', i)\}$
11. Remove all vertices and arcs in  $G$  containing  $c$
12. return  $G$

- ▶ Reminder:  $w_{ij} = \arg \max_k w_{ij}^k$

# Chu-Liu-Edmonds PseudoCode

---

```
contract( $G = (V, A)$ ,  $C$ ,  $w$ )
1. Let  $G_C$  be the subgraph of  $G$  excluding nodes in  $C$ 
2. Add a node  $c$  to  $G_C$  representing cycle  $C$ 
3. For  $i \in V - C : \exists_{i' \in C} (i', i) \in A$ 
   Add arc  $(c, i)$  to  $G_C$  with
    $ma(c, i) = \arg \max_{i' \in C} score(i', i)$ 
    $i' = ma(c, i)$ 
    $score(c, i) = score(i', i)$ 
4. For  $i \in V - C : \exists_{i' \in C} (i, i') \in A$ 
   Add edge  $(i, c)$  to  $G_C$  with
    $ma(i, c) = \arg \max_{i' \in C} [score(i, i') - score(a(i'), i')]$ 
    $i' = ma(i, c)$ 
    $score(i, c) = [score(i, i') - score(a(i'), i') + score(C)]$ 
      where  $a(v)$  is the predecessor of  $v$  in  $C$ 
      and  $score(C) = \sum_{v \in C} score(a(v), v)$ 
5. return  $< G_C, c, ma >$ 
```

# Arc Weights

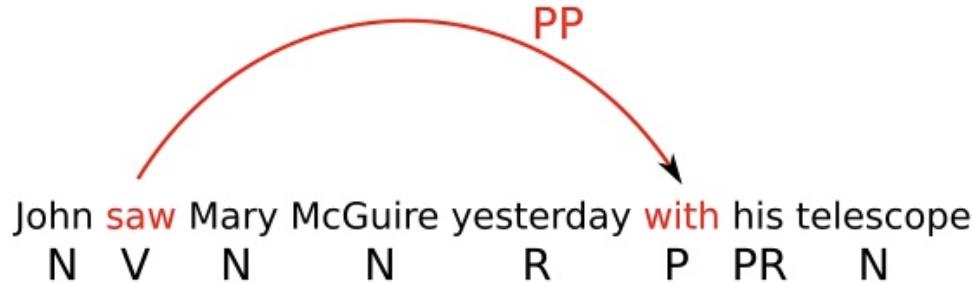
---

$$w_{ij}^k = e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)}$$

- ▶ Arc weights are a linear combination of features of the arc,  $\mathbf{f}$ , and a corresponding weight vector  $\mathbf{w}$
- ▶ Raised to an exponent (simplifies some math ...)
- ▶ What arc features?
- ▶ [McDonald et al. 2005] discuss a number of binary features

# Arc Feature Ideas for $f(i,j,k)$

---



- Identities of the words  $w_i$  and  $w_j$  and the label  $l_k$
- Part-of-speech tags of the words  $w_i$  and  $w_j$  and the label  $l_k$
- Part-of-speech of words surrounding and between  $w_i$  and  $w_j$
- Number of words between  $w_i$  and  $w_j$ , and their orientation
- Combinations of the above

# (Structured) Perceptron

---

Training data:  $\mathcal{T} = \{(x_t, G_t)\}_{t=1}^{|\mathcal{T}|}$

1.  $\mathbf{w}^{(0)} = 0; i = 0$
2. for  $n : 1..N$
3.     for  $t : 1..T$
4.         Let  $G' = \arg \max_{G'} \mathbf{w}^{(i)} \cdot \mathbf{f}(G')$
5.         if  $G' \neq G_t$
6.              $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \mathbf{f}(G_t) - \mathbf{f}(G')$
7.              $i = i + 1$
8. return  $\mathbf{w}^i$

# Partition Function

---

**Partition Function:**  $Z_x = \sum_{G \in T(G_x)} w(G)$

- ▶ Lapacian Matrix  $Q$  for graph  $G_x = (V_x, A_x)$

$$Q_{jj} = \sum_{i \neq j, (i,j,k) \in A_x} w_{ij}^k \quad \text{and} \quad Q_{ij} = \sum_{i \neq j, (i,j,k) \in A_x} -w_{ij}^k$$

- ▶ Cofactor  $Q^i$  is the matrix  $Q$  with the  $i^{th}$  row and column removed

**The Matrix Tree Theorem** [Tutte 1984]

The determinant of the cofactor  $Q^0$  is equal to  $Z_x$

- ▶ Thus  $Z_x = |Q^0|$  – determinants can be calculated in  $O(n^3)$
- ▶ Constructing  $Q$  takes  $O(|L|n^2)$
- ▶ Therefore the whole process takes  $O(n^3 + |L|n^2)$

# Arc Expectations

---

$$\langle i, j, k \rangle_x = \sum_{G \in T(G_x)} w(G) \times \mathbb{1}[(i, j, k) \in A]$$

- ▶ Can easily be calculated, first reset some weights

$$w_{i'j}^{k'} = 0 \quad \forall i' \neq i \text{ and } k' \neq k$$

- ▶ Now,  $\langle i, j, k \rangle_x = Z_x$
- ▶ Why? All competing arc weights to zero, therefore every non-zero weighted graph must contain  $(i, j, k)$
- ▶ Naively takes  $O(n^5 + |L|n^2)$  to compute all expectations
- ▶ But can be calculated in  $O(n^3 + |L|n^2)$  (see [McDonald and Satta 2007, Smith and Smith 2007, Koo et al. 2007])

# Summary

---

- Constituency Parsing
  - CKY Algorithm
  - Lexicalized Grammars
  - Latent Variable Grammars
- Dependency Parsing
  - Eisner Algorithm
  - Maximum Spanning Tree Algorithm

**There is lots more and these models are being actively used in practice!**