Algorithms for NLP

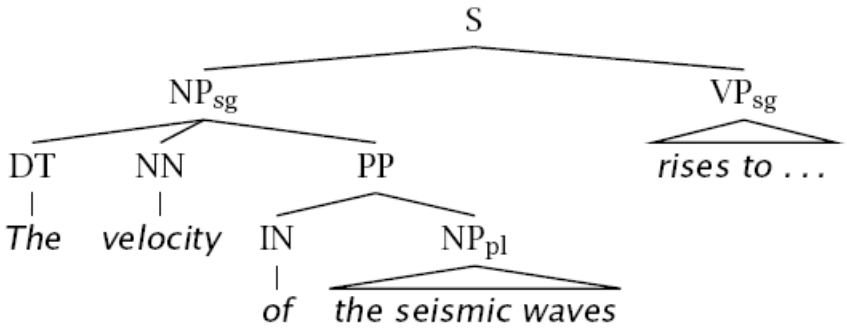
Parsing II

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley

Phrase Structure Parsing

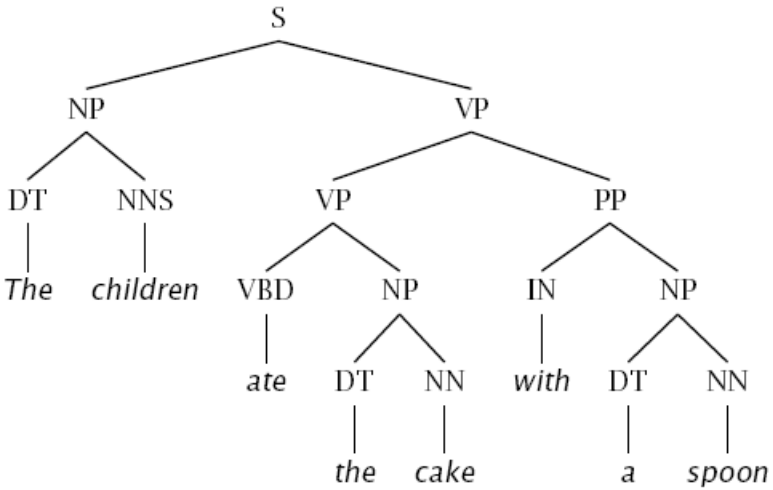
- Phrase structure parsing organizes syntax into *constituents* or *brackets*
- In general, this involves nested trees
- Linguists can, and do, argue about details
- Lots of ambiguity
- Not the only kind of syntax…



new art critics write reviews with computers

Constituency Tests

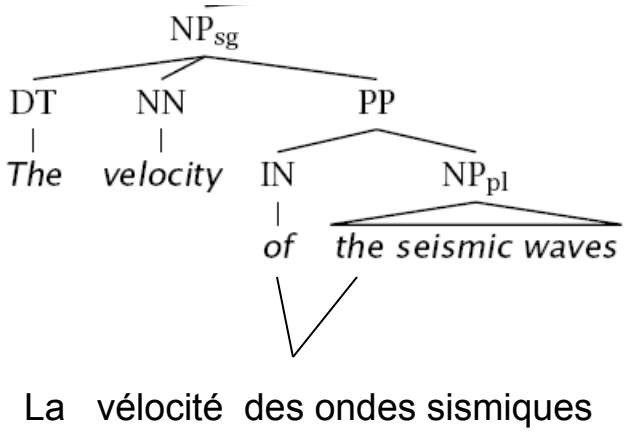
- How do we know what nodes go in the tree?
- Classic constituency tests:
  - Substitution by *proform*
  - Question answers
  - Semantic gounds
    - Coherence
    - Reference
    - Idioms
  - Dislocation
  - Conjunction



- 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.



Classical NLP: Parsing

- Write symbolic or logical rules:

Grammar (CFG)

ROOT → S	NP → NP PP
S → NP VP	VP → VBP NP
NP → DT NN	VP → VBP NP PP
NP → NN NNS	PP → IN NP

Lexicon

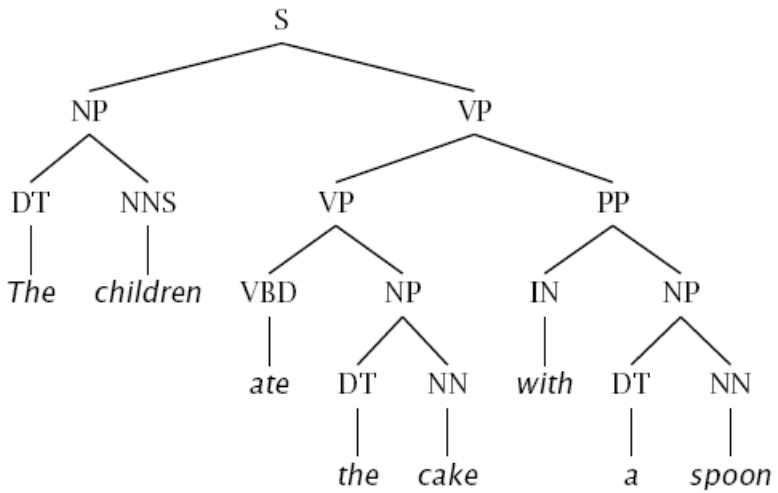
NN → interest
NNS → raises
VBP → interest
VBZ → raises
…

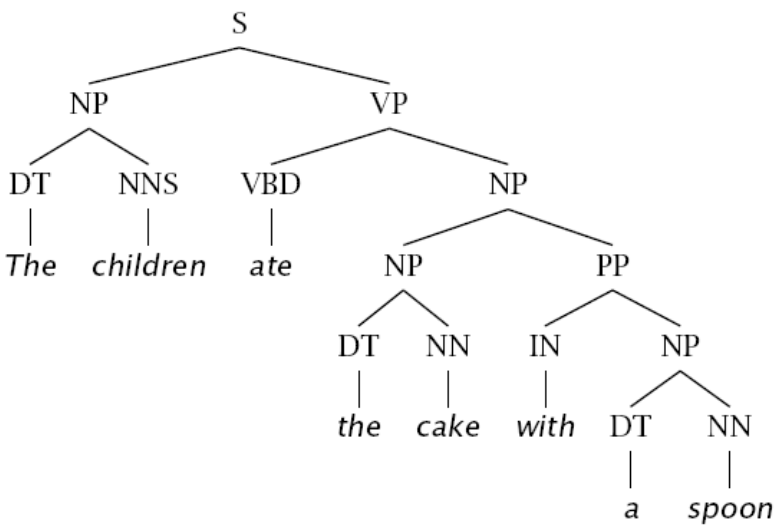
- Use deduction systems to prove parses from words
  - Minimal grammar on "Fed raises" sentence: 36 parses
  - Simple 10-rule grammar: 592 parses
  - Real-size grammar: many millions of parses

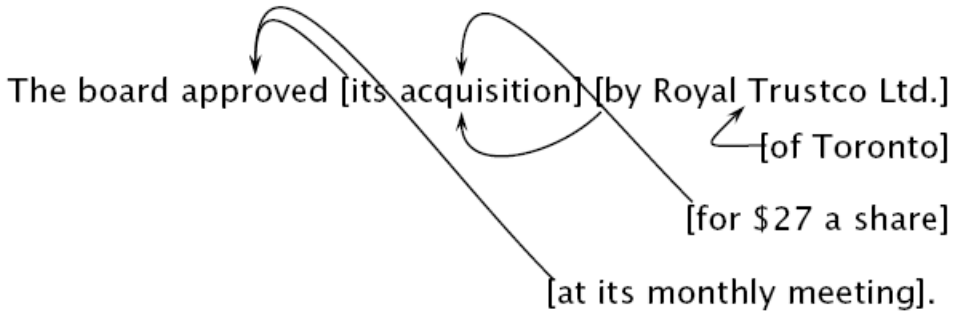
- This scaled very badly, didn't yield broad-coverage tools

Ambiguities

Ambiguities: PP Attachment







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.

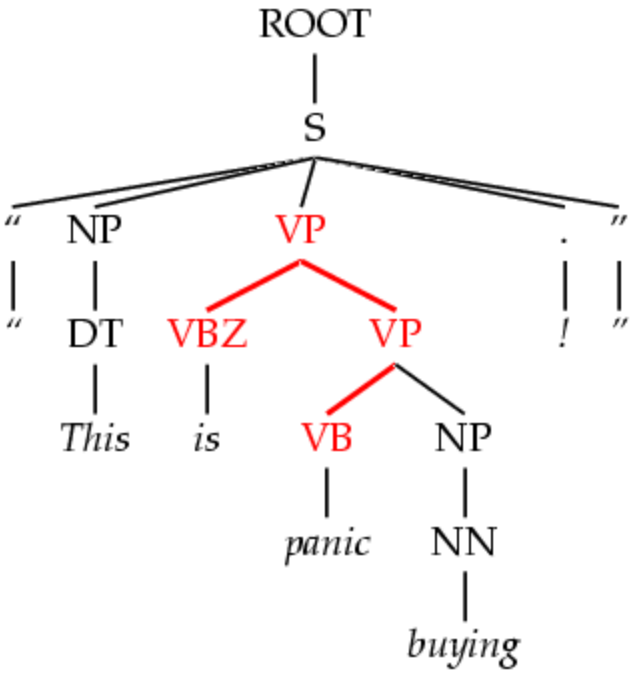
- Coordination scope:
  Small rats and mice can squeeze into holes or cracks in the wall.

Dark Ambiguities

- *Dark ambiguities*: most analyses are shockingly bad (meaning, they don't have an interpretation you can get your mind around)

This analysis corresponds to the correct parse of

"This will panic buyers ! "



- Unknown words and new usages
- Solution: We need mechanisms to focus attention on the best ones, probabilistic techniques do this

PCFGs

Probabilistic Context-Free Grammars

- A context-free grammar is a tuple <*N, T, S, R*>
  - *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 $X \rightarrow Y_1\ Y_2 \ldots Y_k$, with $X, Y_i \in N$
    - Examples: S → NP VP, VP → VP CC VP
    - Also called rewrites, productions, or local trees

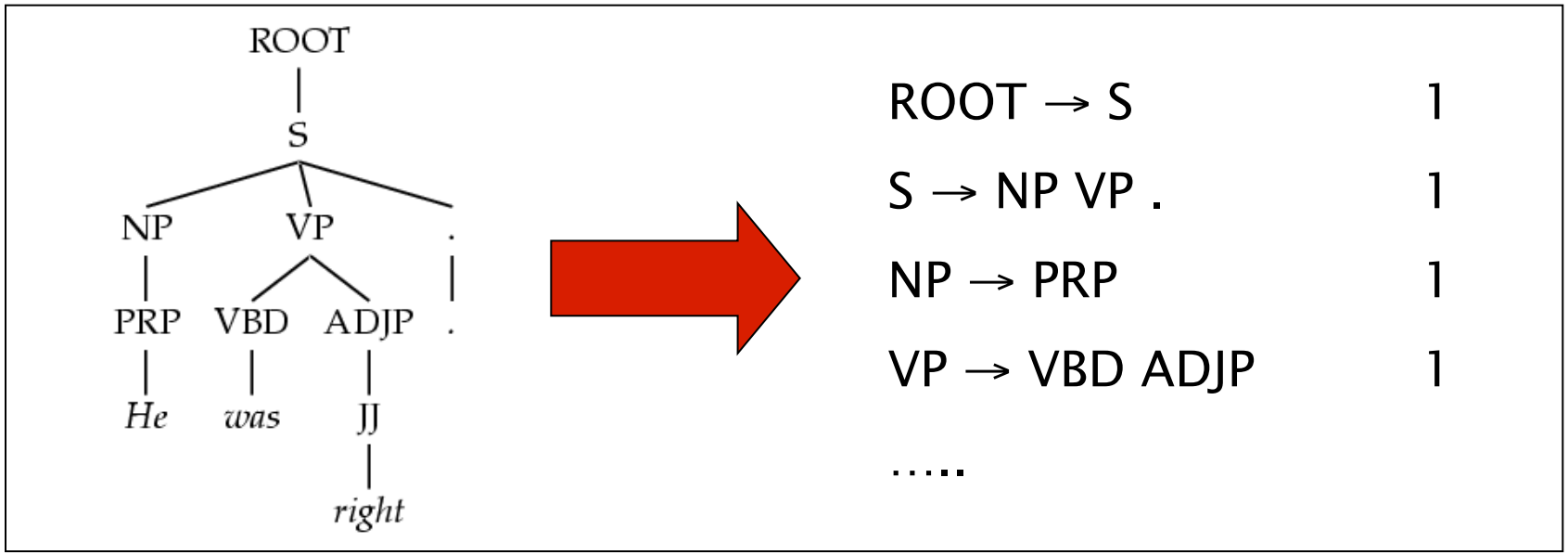
- A PCFG adds:
  - A top-down production probability per rule $P(Y_1\ Y_2 \ldots Y_k \mid X)$

Treebank Sentences

```
( (S (NP-SBJ The move)
     (VP followed
         (NP (NP a round)
             (PP of
                 (NP (NP similar increases)
                     (PP by
                         (NP other lenders))
                     (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

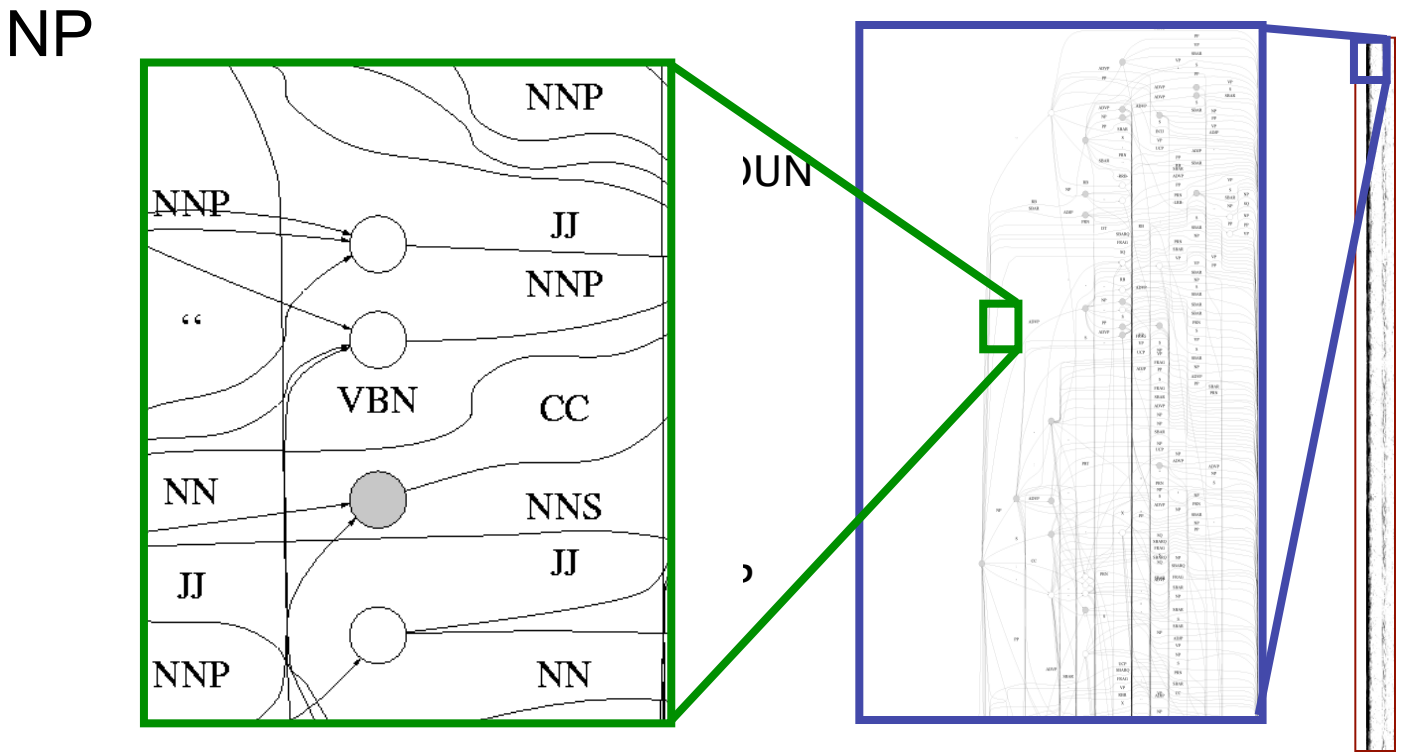
- Need a PCFG for broad coverage parsing.
- Can take a grammar right off the trees (doesn't work well):



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

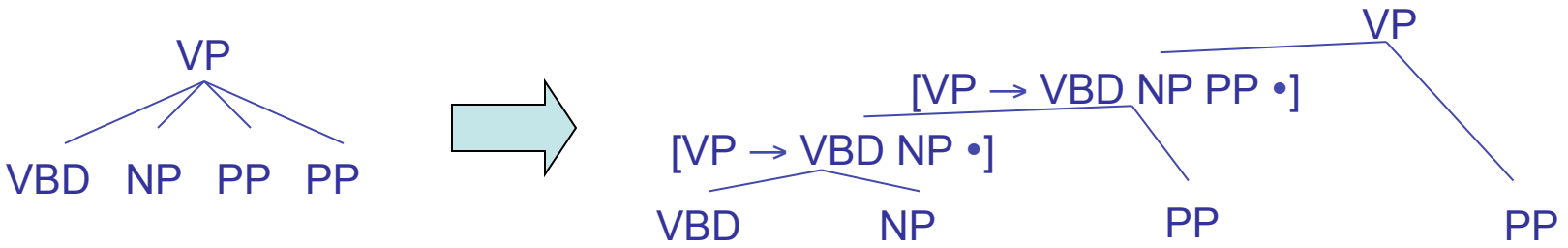
Treebank Grammar Scale

- 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



Chomsky Normal Form

- Chomsky normal form:
  - All rules of the form X → Y Z or X → 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!

CKY Parsing

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

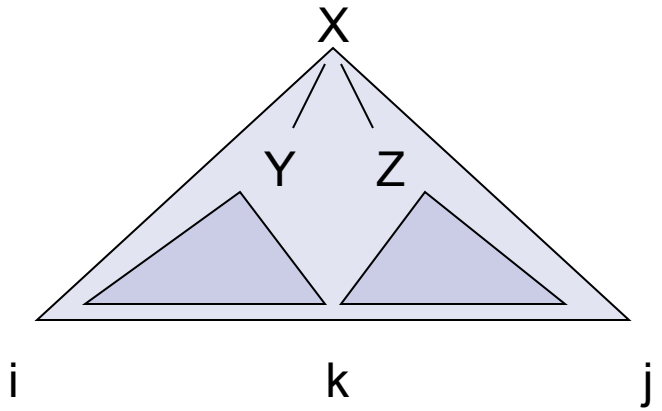
- One small change:

```
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]
```



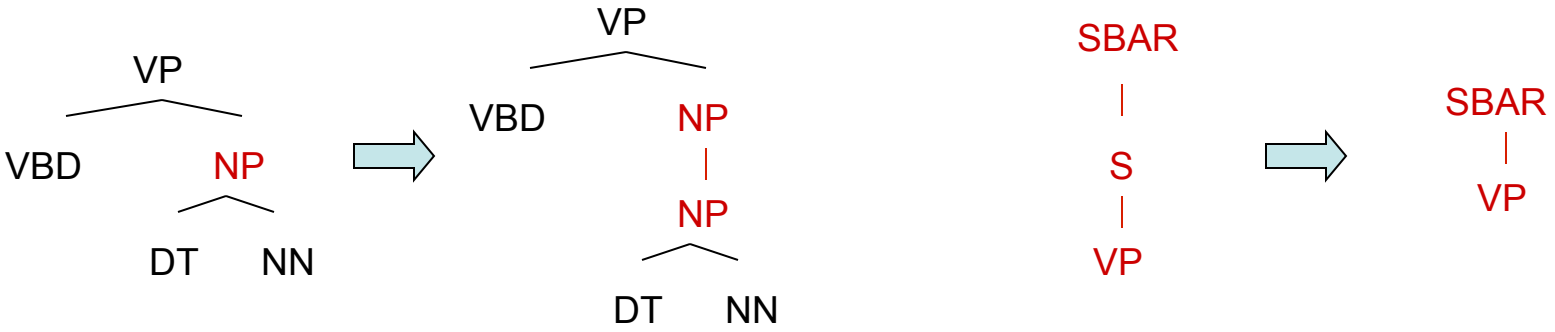
Unary Rules

- 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)
```

Analysis

Memory

- How much memory does this require?
  - Have to store the score cache
  - Cache size: |symbols|*n^2 doubles
  - For the plain treebank grammar:
    - X ~ 20K, n = 40, double ~ 8 bytes = ~ 256MB
    - Big, but workable.

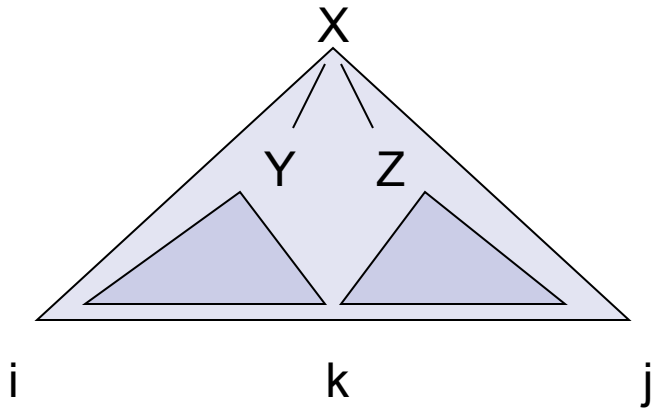
- Pruning: Beams
  - score[X][i][j] can get too large (when?)
  - Can keep beams (truncated maps score[i][j]) which only store the best few scores for the span [i,j]

- 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 (<= n)
    - For each i (<= n)
      - For each rule X → Y Z
        - For each split point k

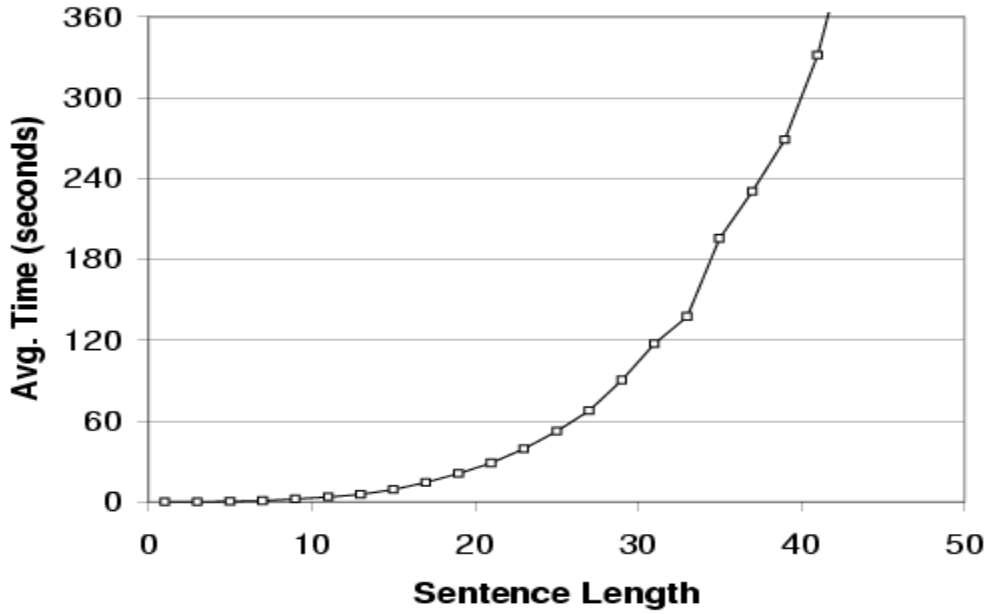
          Do constant work



  - Total time: |rules|*n^3
  - Something like 5 sec for an unoptimized parse of a 20-word sentence

Time: Practice

- Parsing with the vanilla treebank grammar:



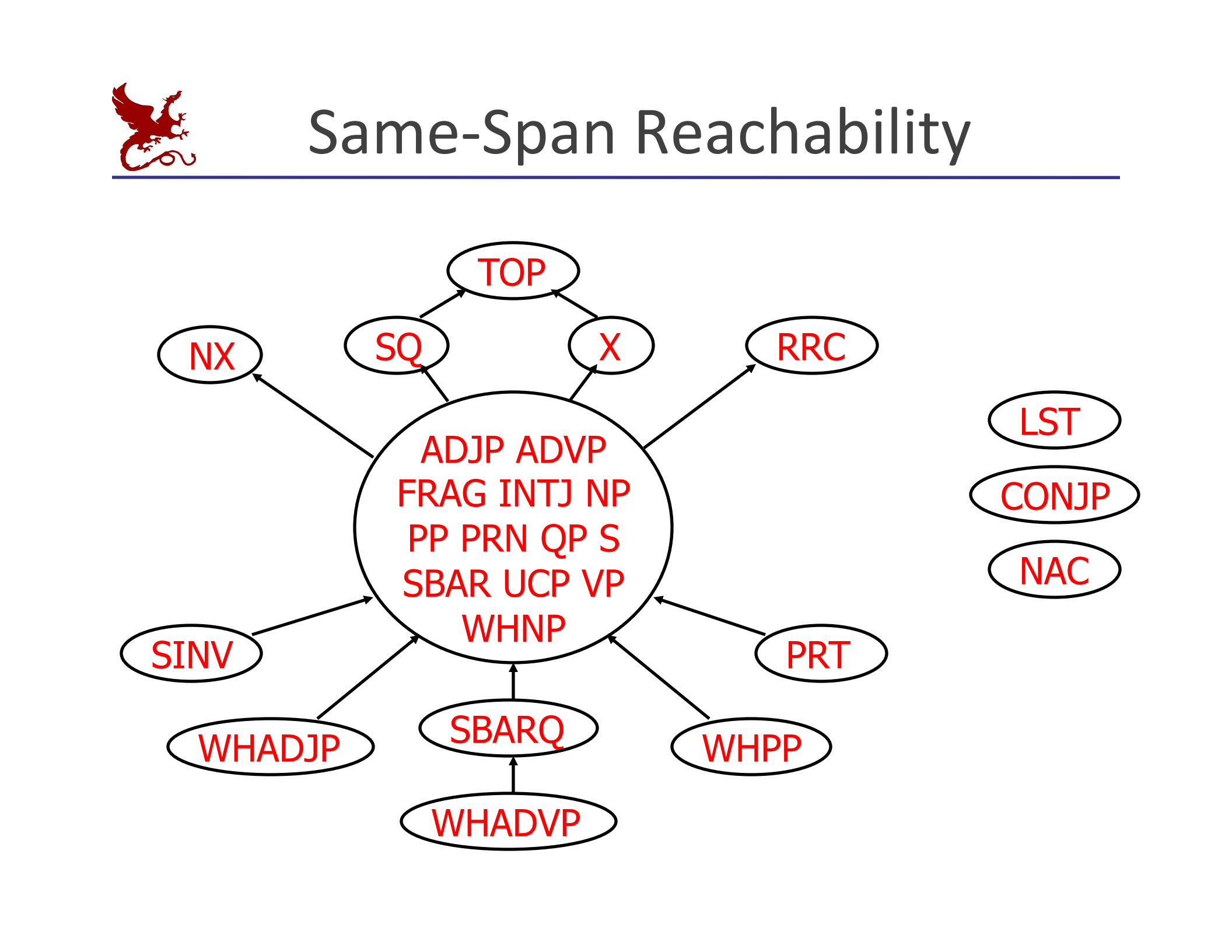
~ 20K Rules

(not an optimized parser!)

Observed exponent:

3.6

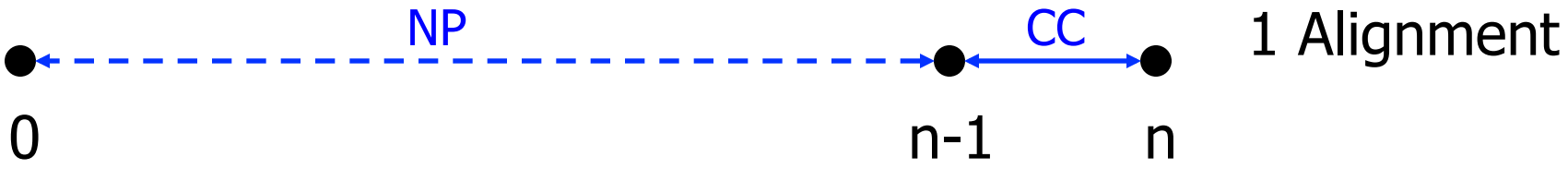
- Why's it worse in practice?
  - Longer sentences "unlock" more of the grammar
  - All kinds of systems issues don't scale



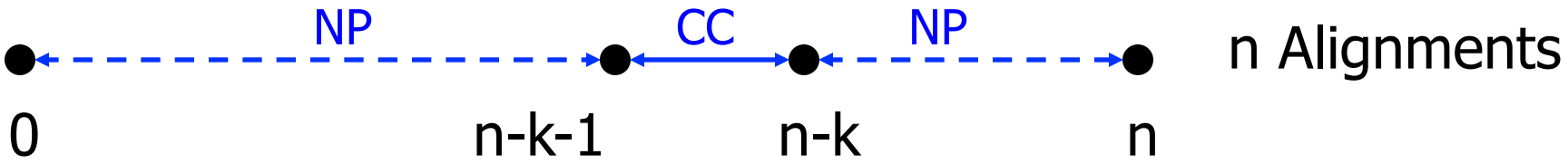
Same-Span Reachability

Rule State Reachability

Example: NP CC •



Example: NP CC NP •



- Many states are more likely to match larger spans!

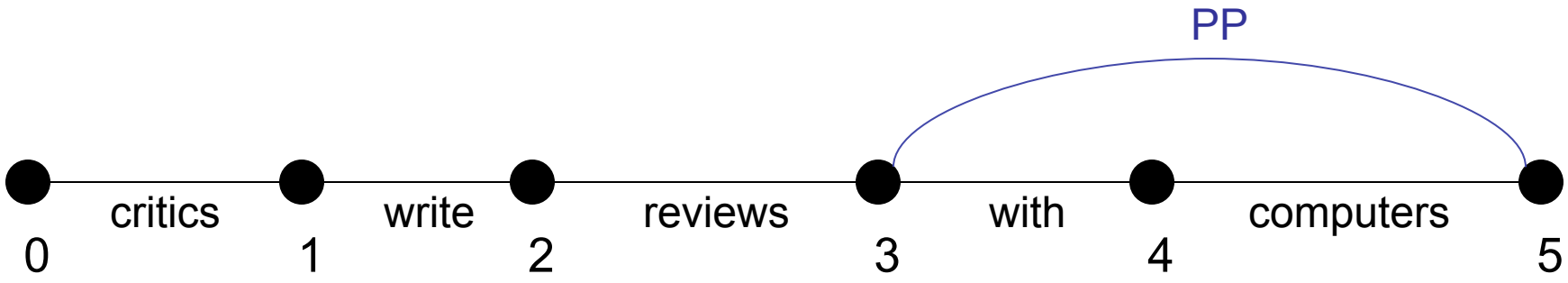
Efficient CKY

- Lots of tricks to make CKY efficient
  - Some 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 important (and interesting):
    - Many X[i,j] can be suppressed on the basis of the input string
    - We'll see this next class as figures-of-merit, A* heuristics, coarse-to-fine, etc

Agenda-Based Parsing

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]

Unary Projection

- When we pop a word item, the lexicon tells us the tag item successors (and scores) which go on the agenda

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

critics write reviews with computers

Item Successors

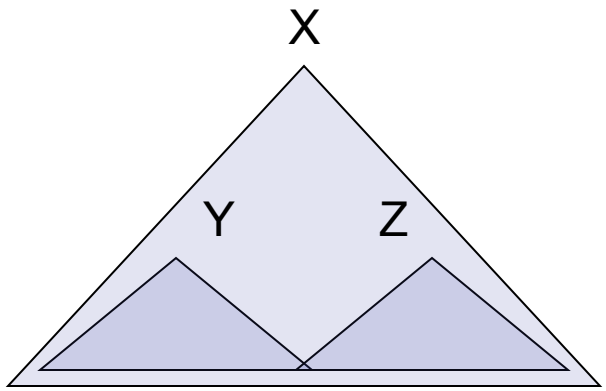
- When we pop items off of the agenda:
  - Graph successors: unary projections (NNS → critics, NP → NNS)

$$Y[i,j] \text{ with } X \rightarrow Y \text{ forms } X[i,j]$$

  - Hypergraph successors: combine with items already in our chart

$$Y[i,j] \text{ and } Z[j,k] \text{ with } X \rightarrow Y\ Z \text{ form } X[i,k]$$

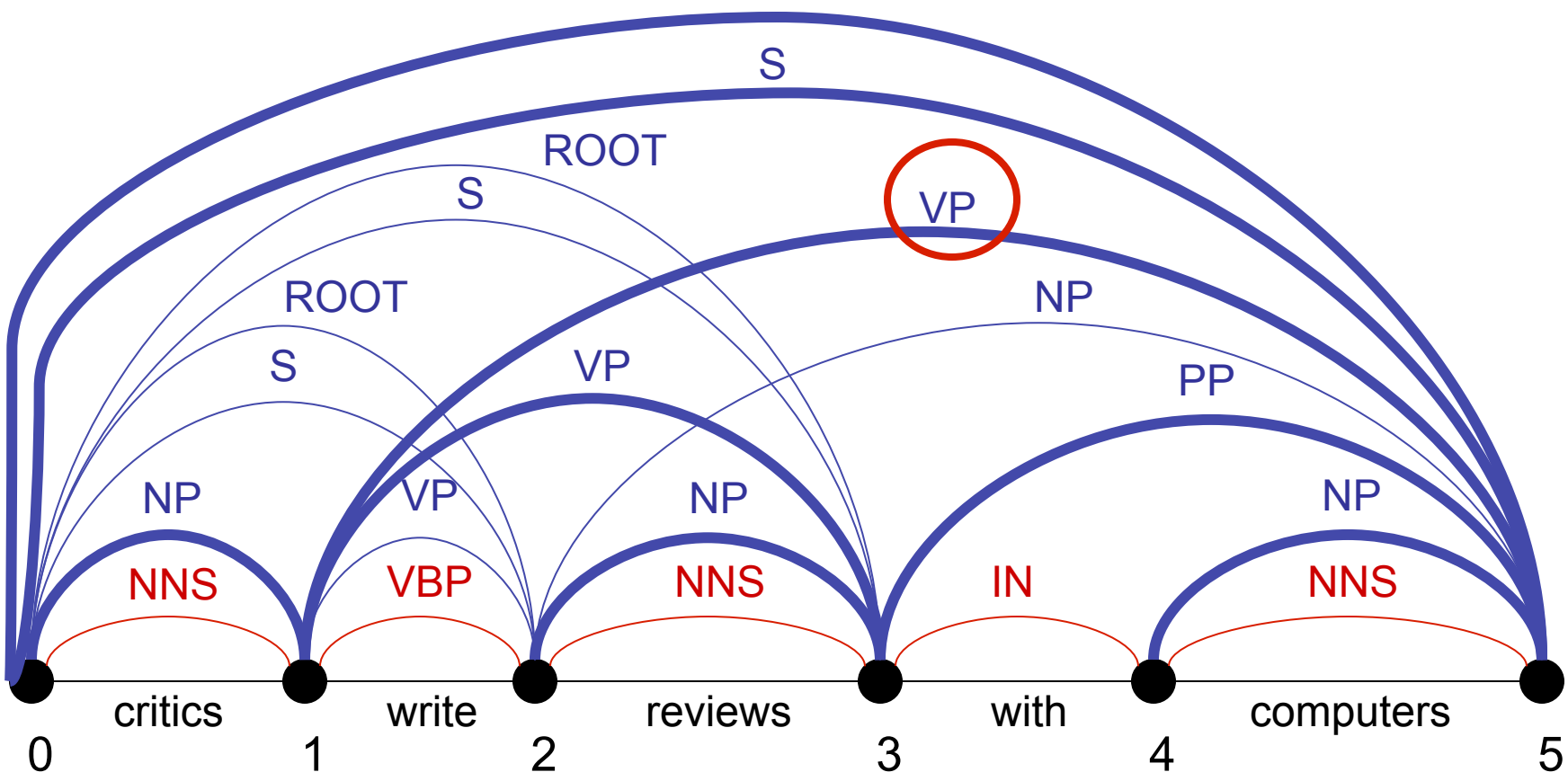
  - Enqueue / promote resulting items (if not in chart already)
  - Record backtraces as appropriate
  - Stick the popped edge in the chart (closed set)



- 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]
ROOT



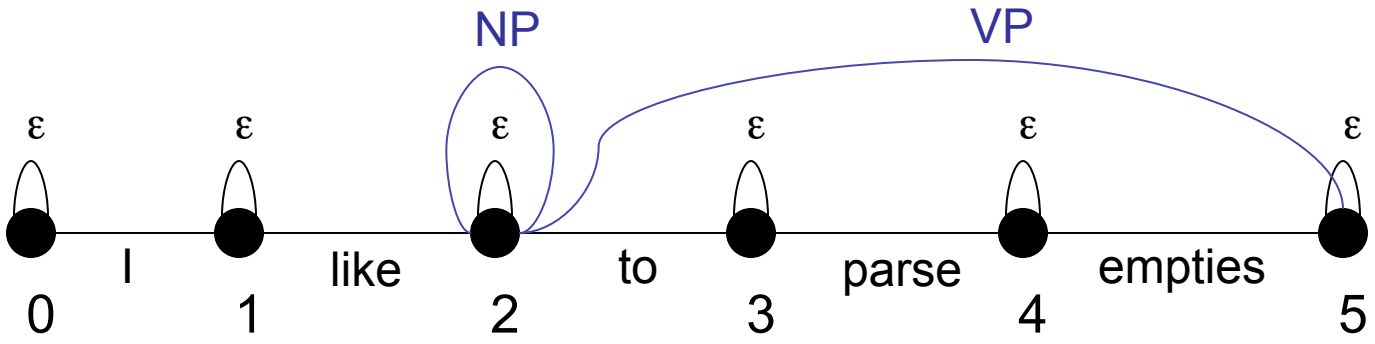
Empty Elements

- Sometimes we want to posit nodes in a parse tree that don't contain any pronounced words:

I want you to parse this sentence

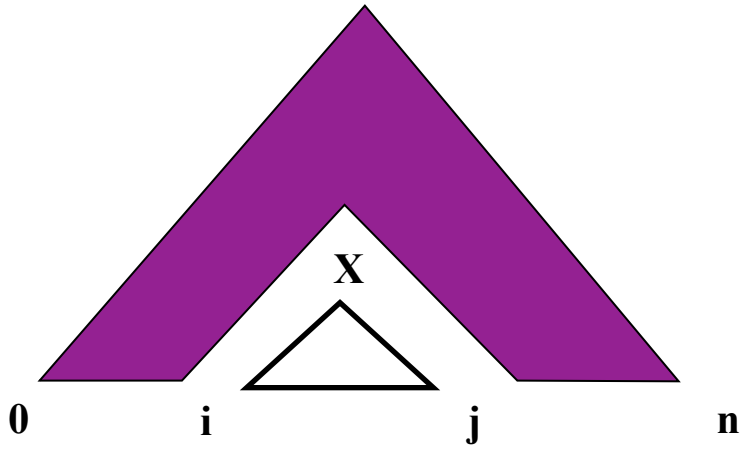
I want [ ] to parse this sentence

- These are easy to add to a agenda-based parser!
  - For each position i, add the "word" edge ε[i,i]
  - Add rules like NP → ε to the grammar
  - That's it!



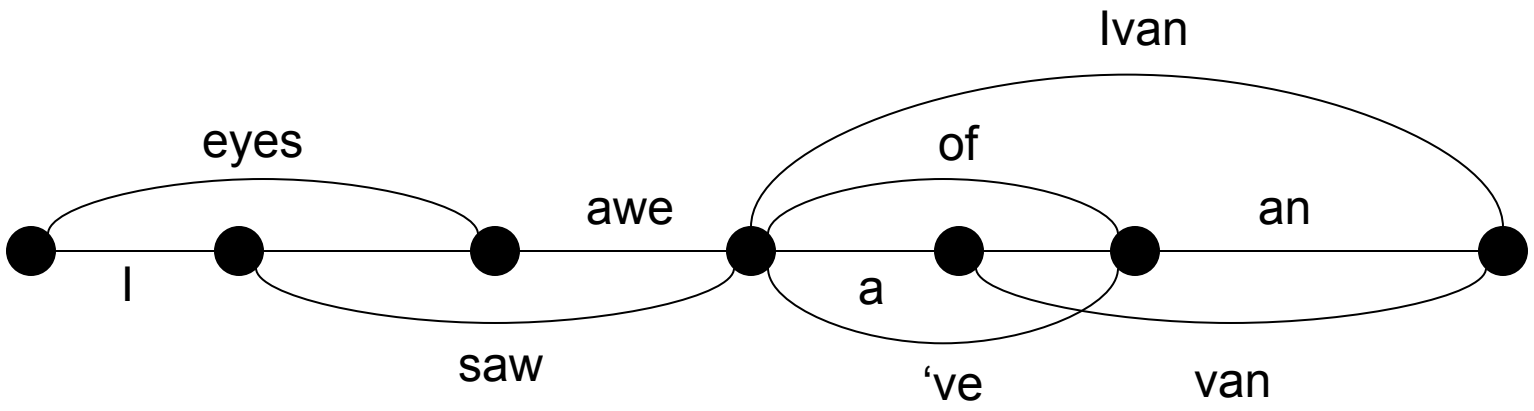
UCS / A*

- 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 optimiality [Klein and Manning 03]



(Speech) Lattices

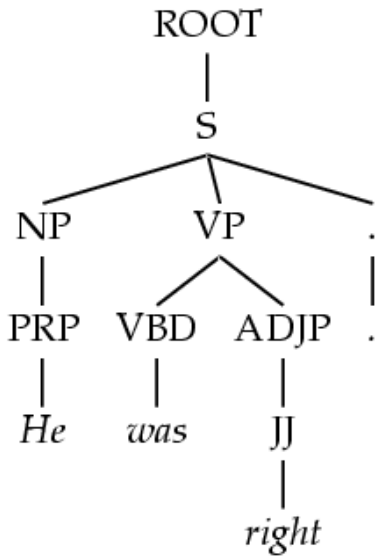
- There was nothing magical about words spanning exactly one position.
- When working with speech, we generally don't know how many words there are, or where they break.
- We can represent the possibilities as a lattice and parse these just as easily.

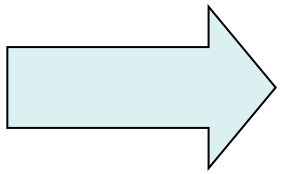


Learning PCFGs

Treebank PCFGs [Charniak 96]

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

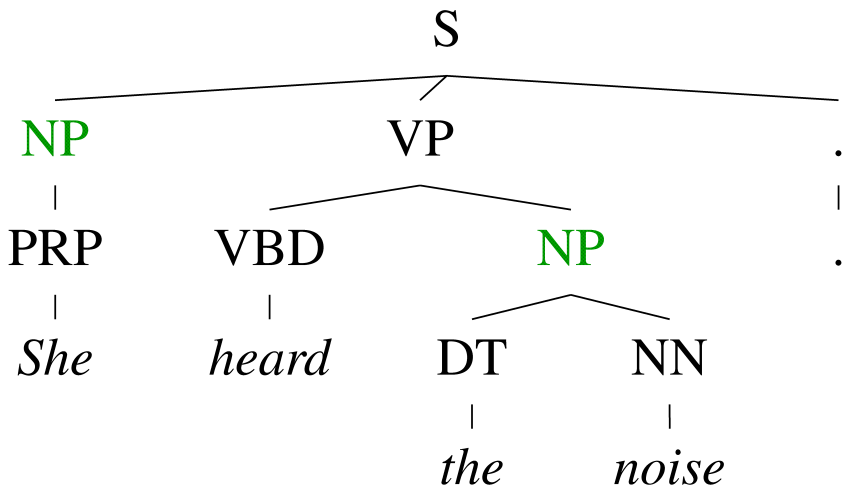




ROOT → S	1
S → NP VP .	1
NP → PRP	1
VP → VBD ADJP	1
…..	

Model	*F1*
Baseline	72.0

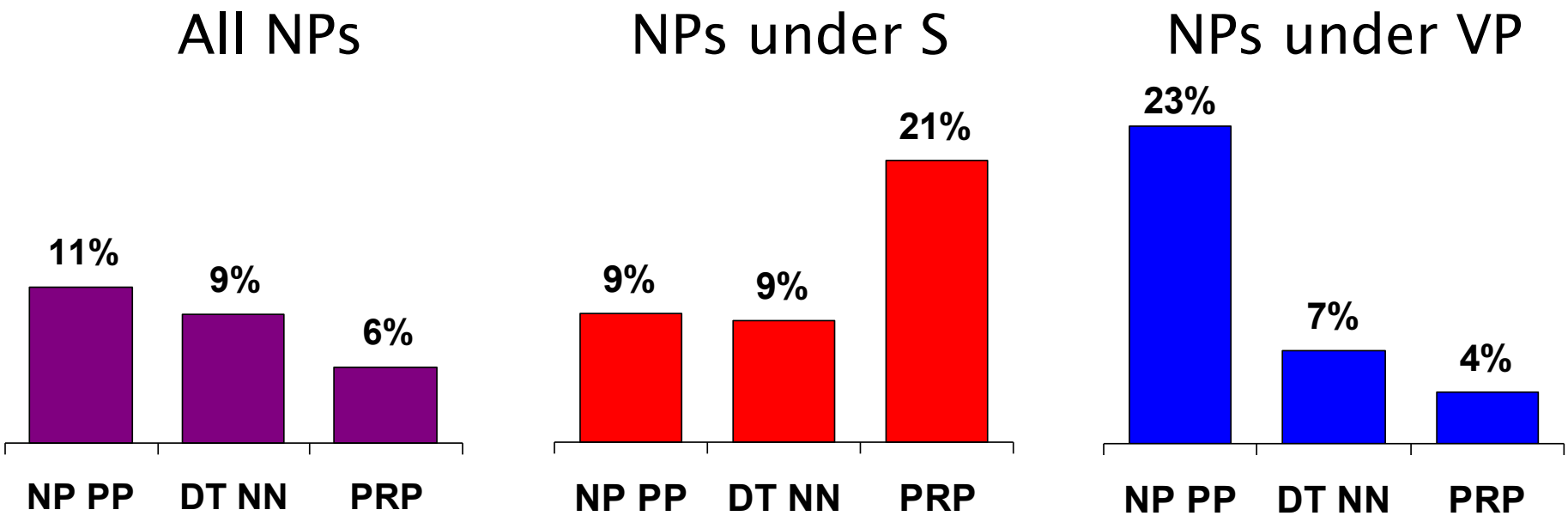
Conditional Independence?



- 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

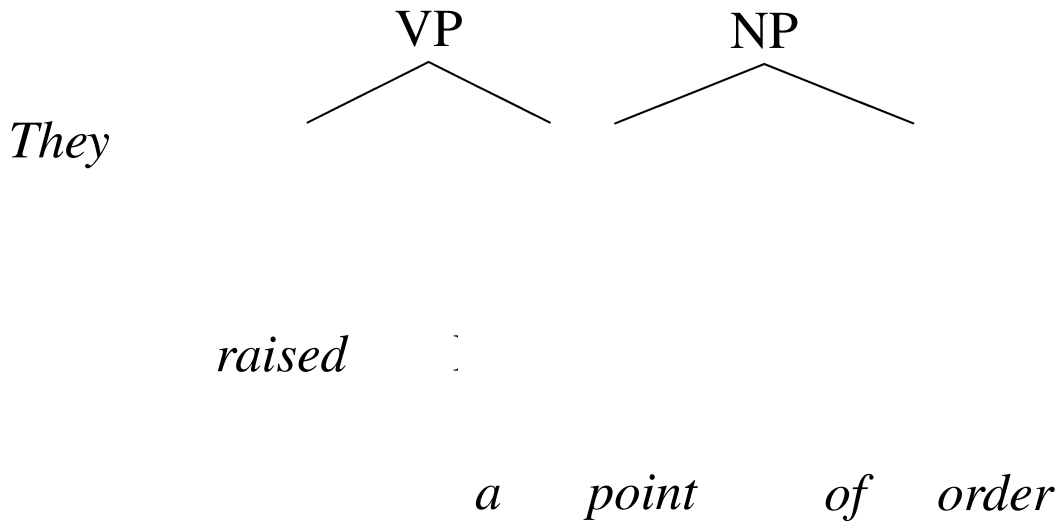
- Independence assumptions are often too strong.



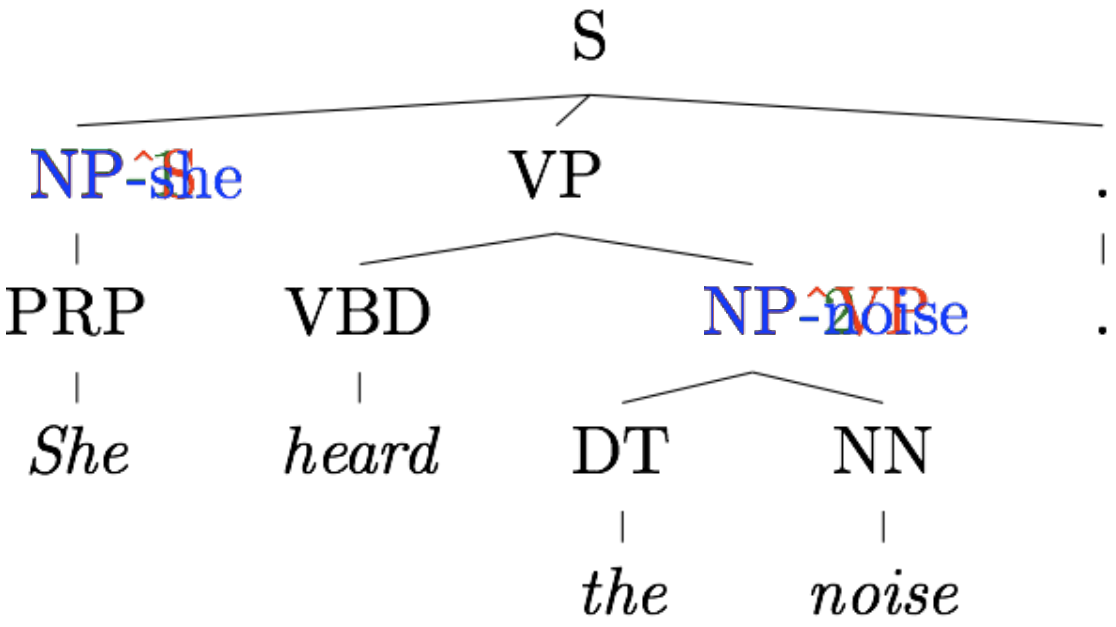
- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).
- Also: the subject and object expansions are correlated!

Grammar Refinement

- Example: PP attachment



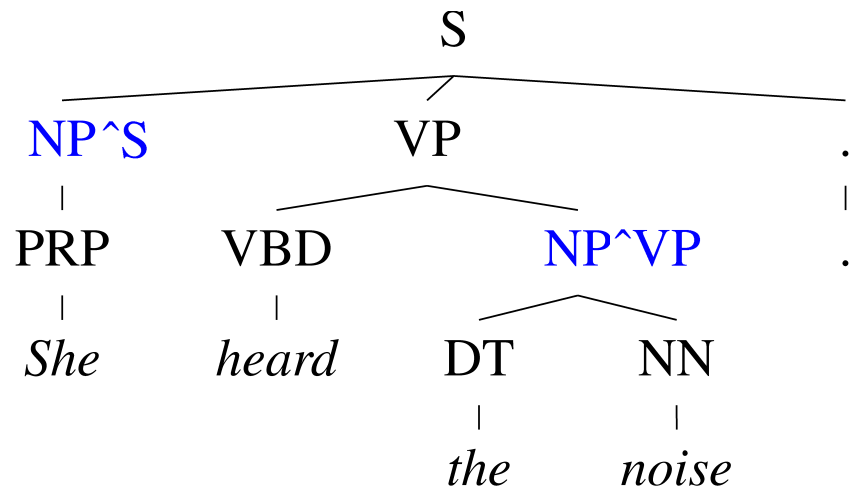
Grammar Refinement



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

Structural Annotation

The Game of Designing a Grammar



- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Structural annotation

Typical Experimental Setup

- Corpus: Penn Treebank, WSJ

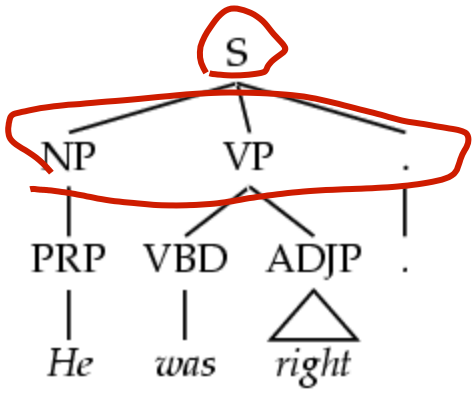
Training:	sections	02-21
Development:	section	22 (here, first 20 files)
Test:	section	23

- Accuracy – F1: harmonic mean of per-node labeled precision and recall.
- Here: also size – number of symbols in grammar.

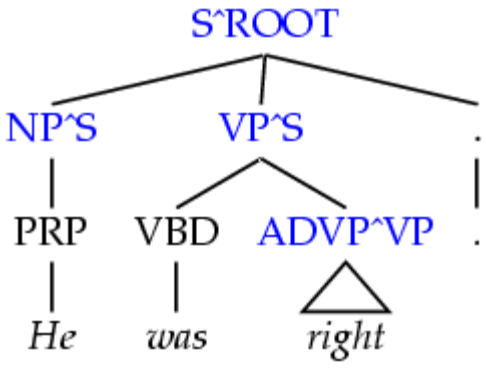
Vertical Markovization

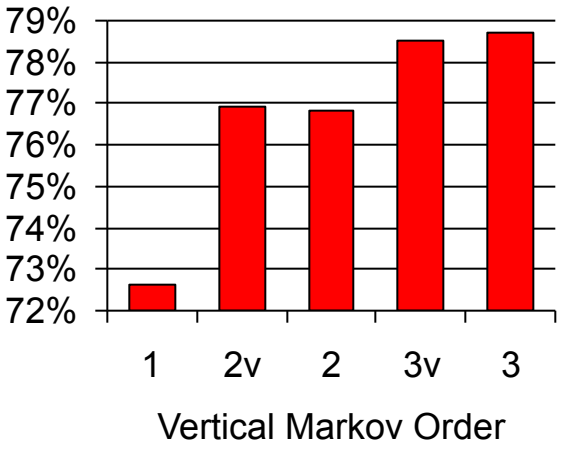
- Vertical Markov order: rewrites depend on past k ancestor nodes.
  (cf. parent annotation)

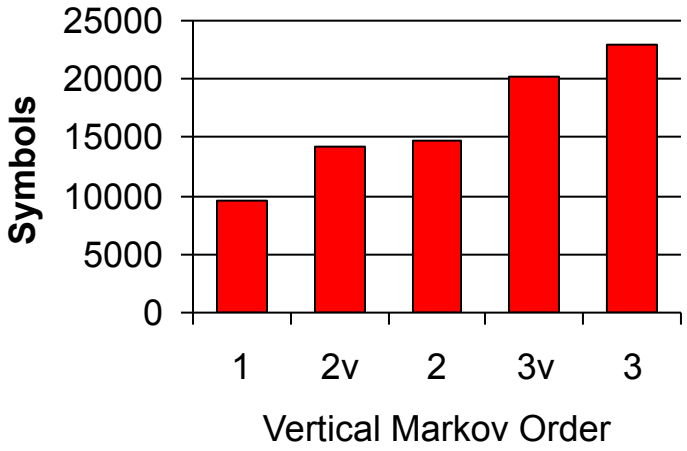
Order 1



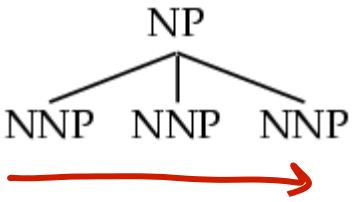
Order 2



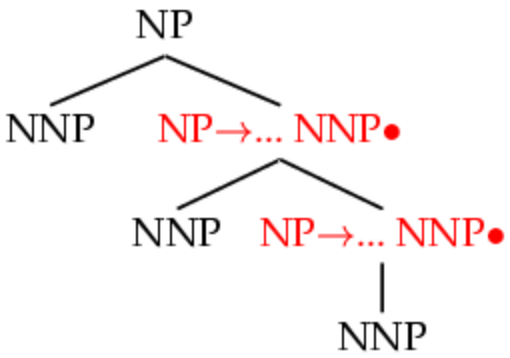




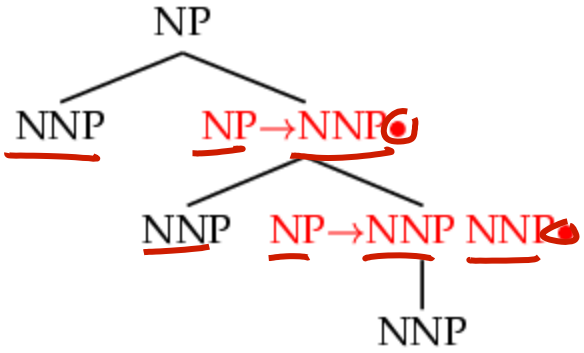
Horizontal Markovization

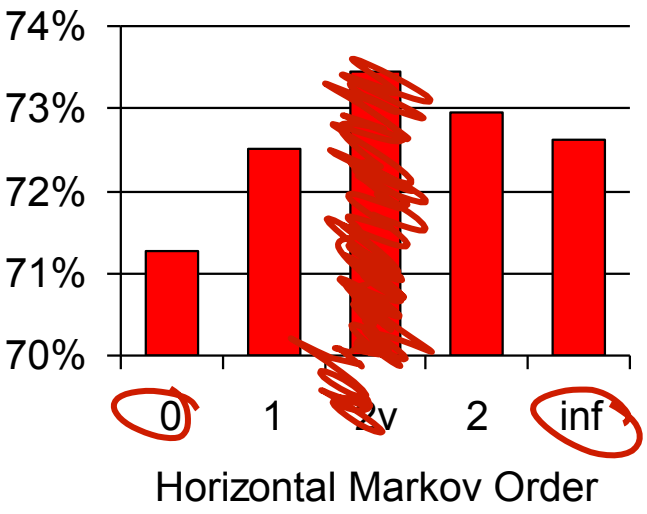


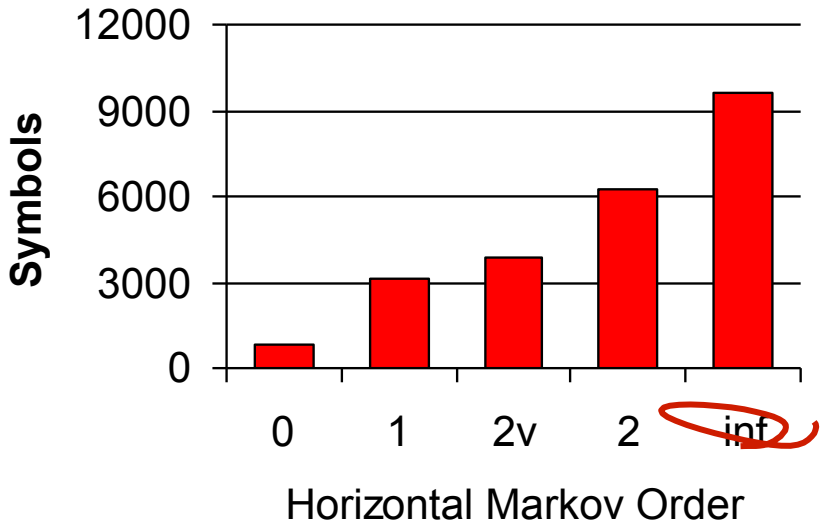
Order 1



Order ∞

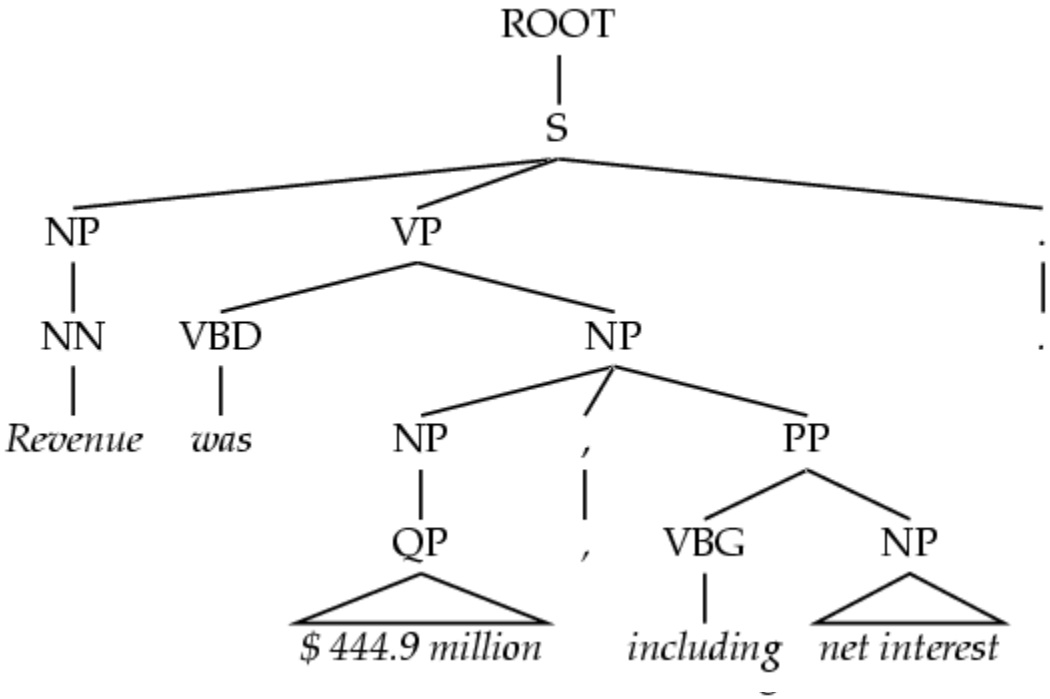






Unary Splits

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

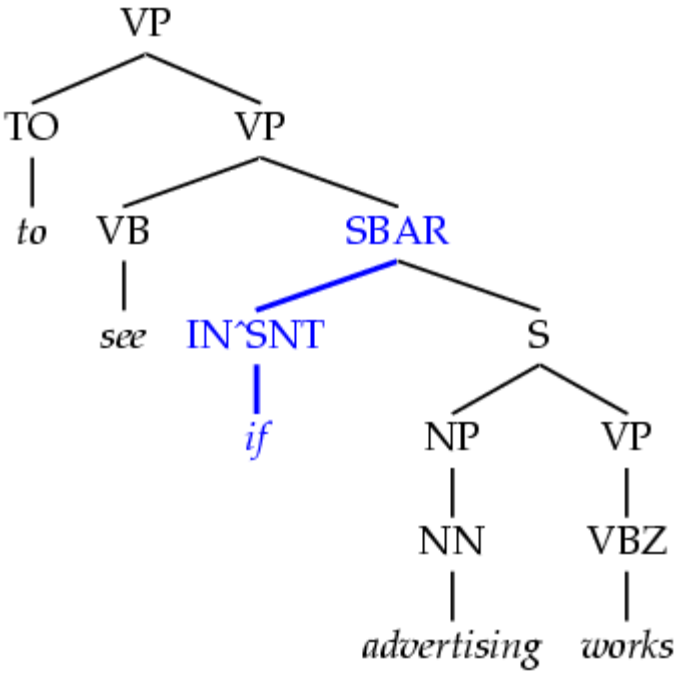


- Solution: Mark unary rewrite sites with -U

Annotation	F1	Size
Base	77.8	7.5K
UNARY	78.3	8.0K

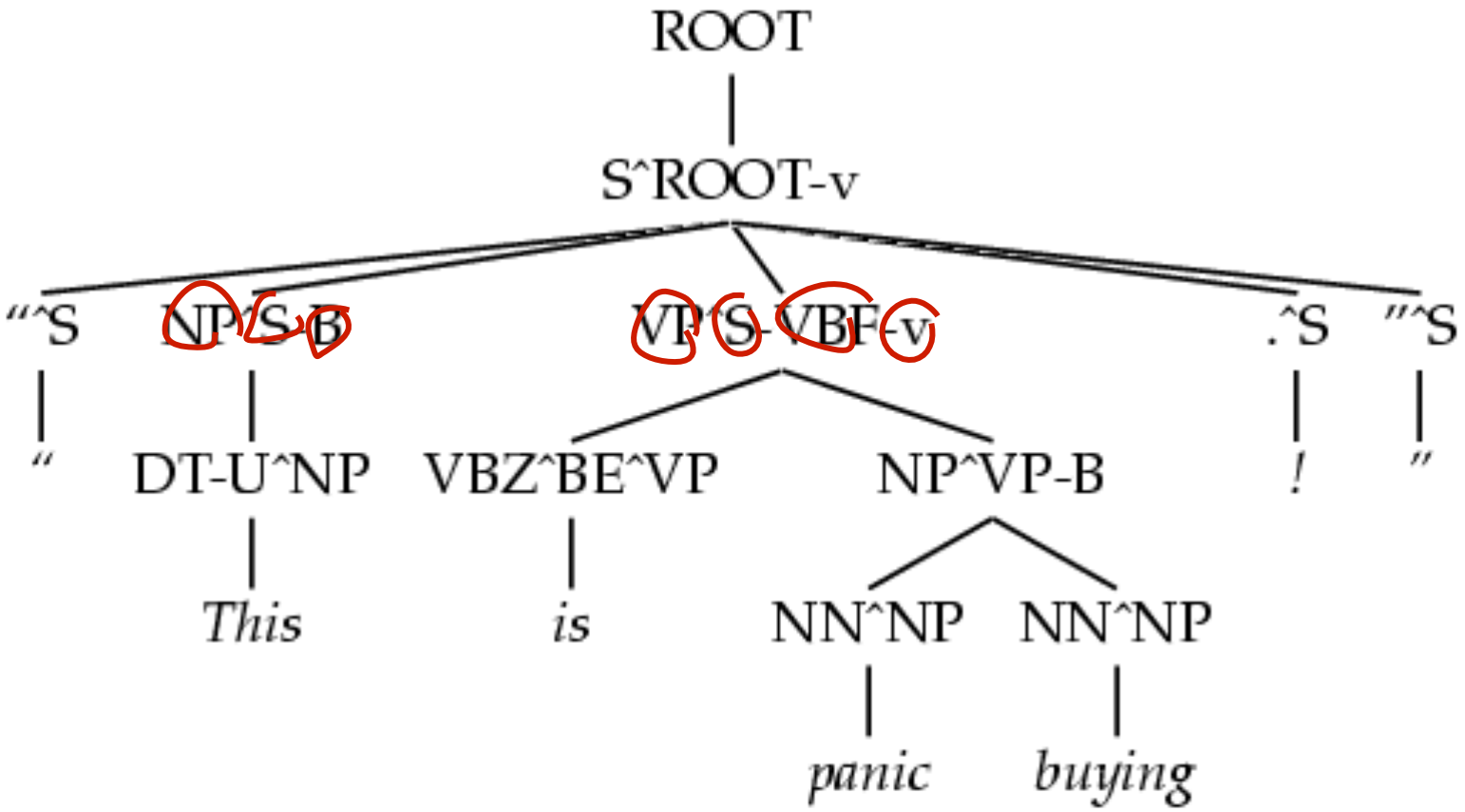
Tag Splits

- 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

A Fully Annotated (Unlex) Tree



Some Test Set Results

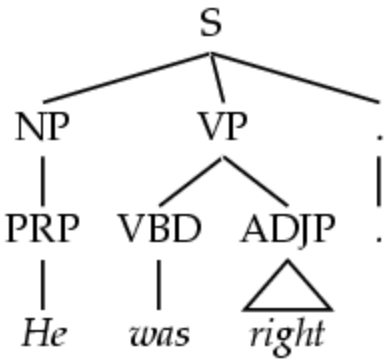
Parser	LP	LR	**F1**	CB	0 CB
Magerman 95	84.9	84.6	**84.7**	1.26	56.6
Collins 96	86.3	85.8	**86.0**	1.14	59.9
Unlexicalized	86.9	85.7	**86.3**	1.10	60.3
Charniak 97	87.4	87.5	**87.4**	1.00	62.1
Collins 99	88.7	88.6	**88.6**	0.90	67.1

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

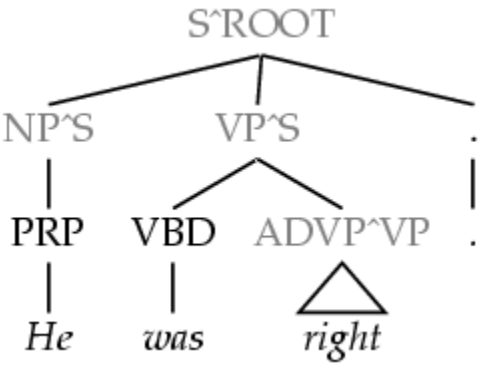
Efficient Parsing for Structural Annotation

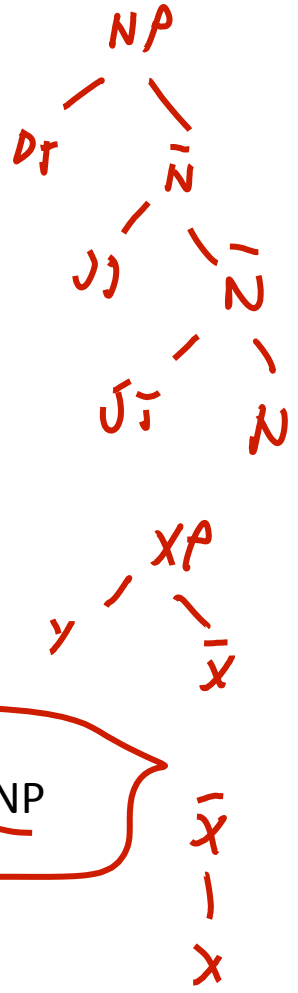
Grammar Projections

Coarse Grammar



Fine Grammar





NP → DT N'

NP^S → DT^NP N'[...DT]^NP

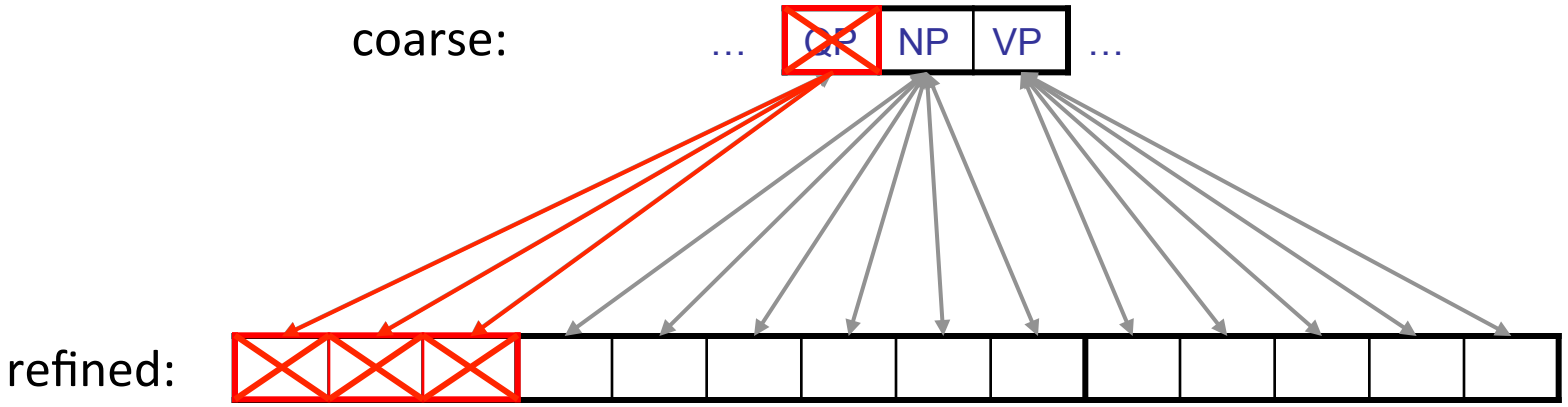
Note: X-Bar Grammars are projections with rules like XP → Y X' or XP → X' Y or X' → X

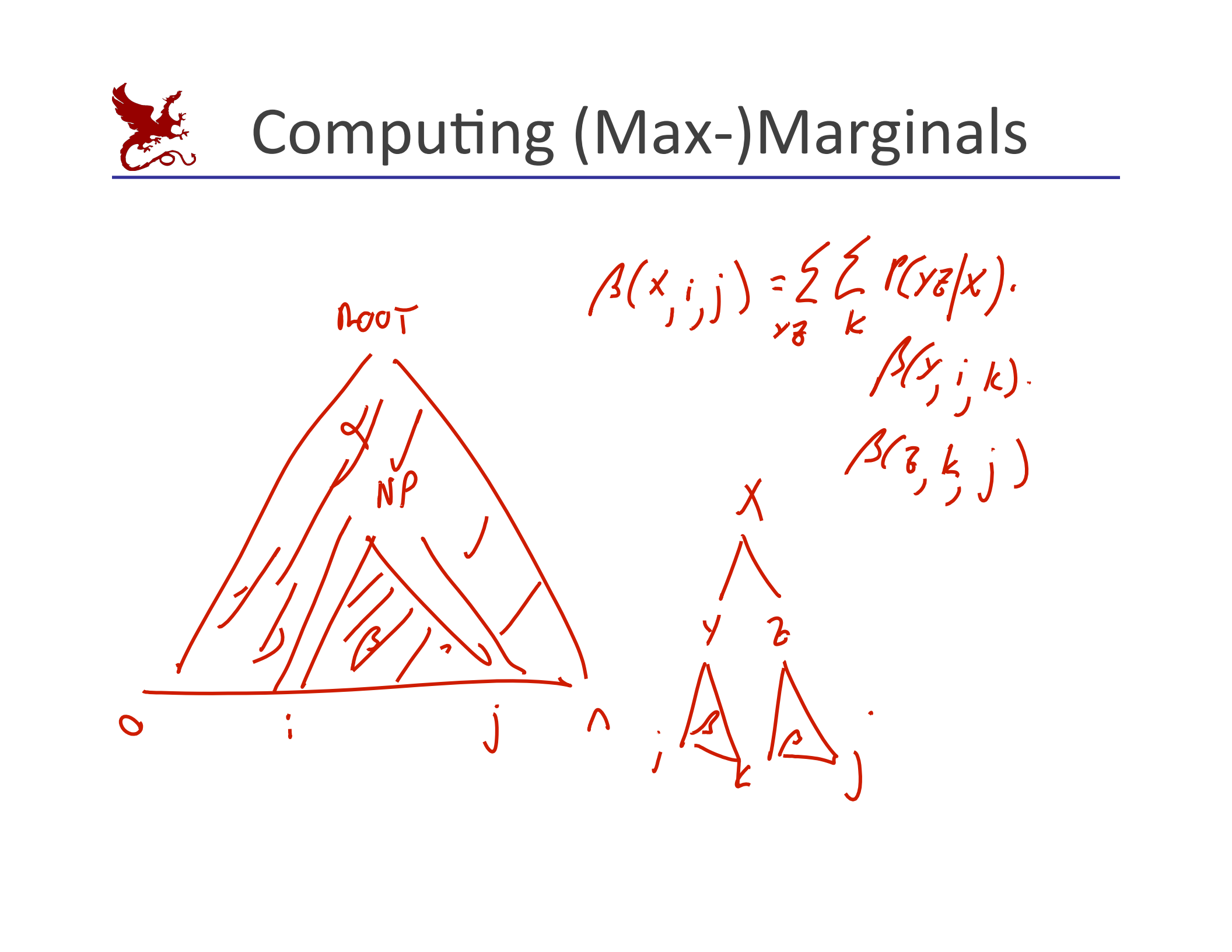
Coarse-to-Fine Pruning

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

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

E.g. consider the span 5 to 12:

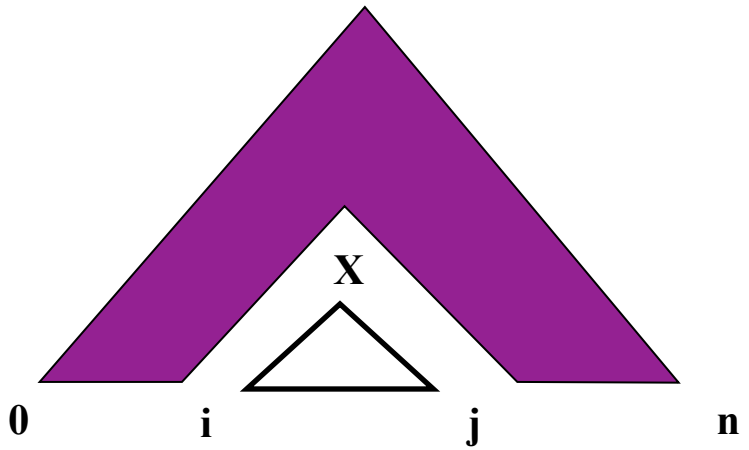



Computing (Max-)Marginals
ROOT
NP
0
i
j
n
X
Y
Z
i
k
j

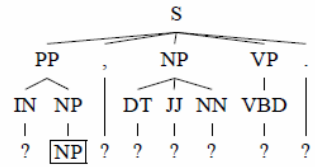
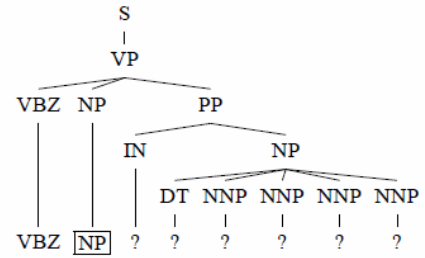
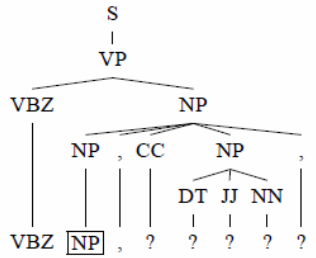
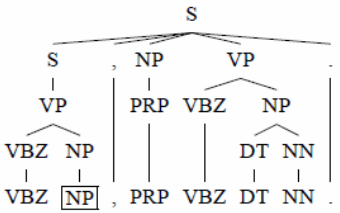
Inside and Outside Scores

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 optimiality [Klein and Manning 03]

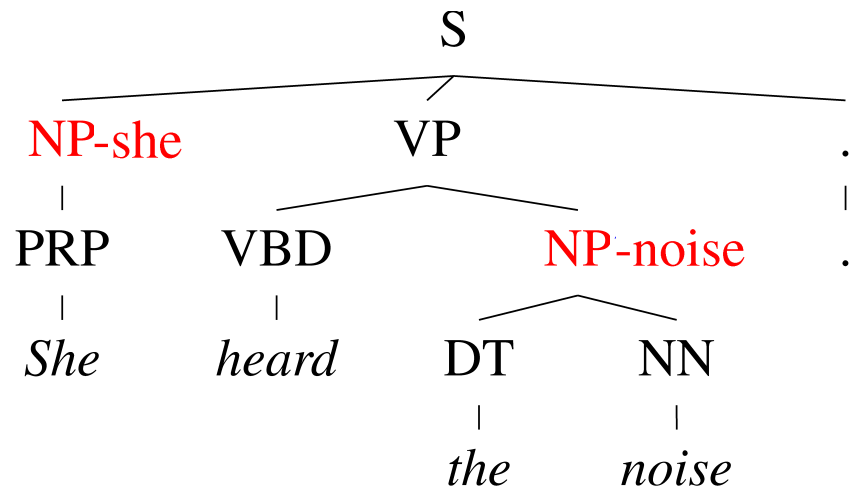


A* Parsing

Estimate	SX	SXL	SXLR	TRUE
Summary	(1,6,NP)	(1,6,NP,VBZ)	(1,6,NP,VBZ,“,”)	(entire context)
Best Tree				
Score	−11.3	−13.9	−15.1	−18.1

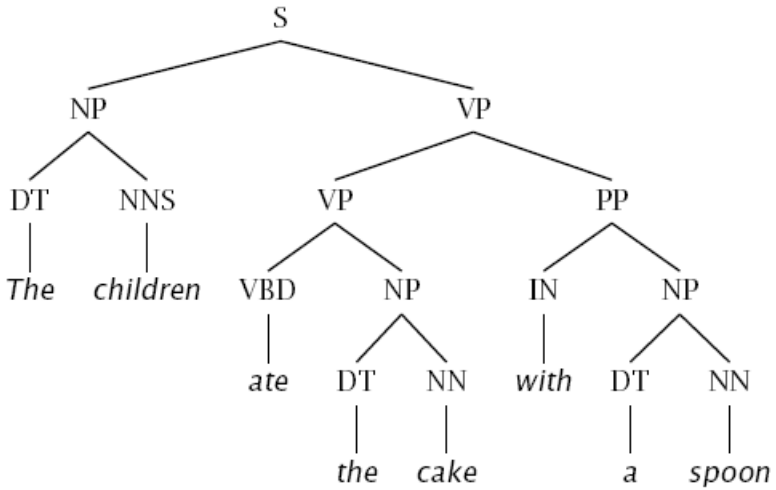
Lexicalization

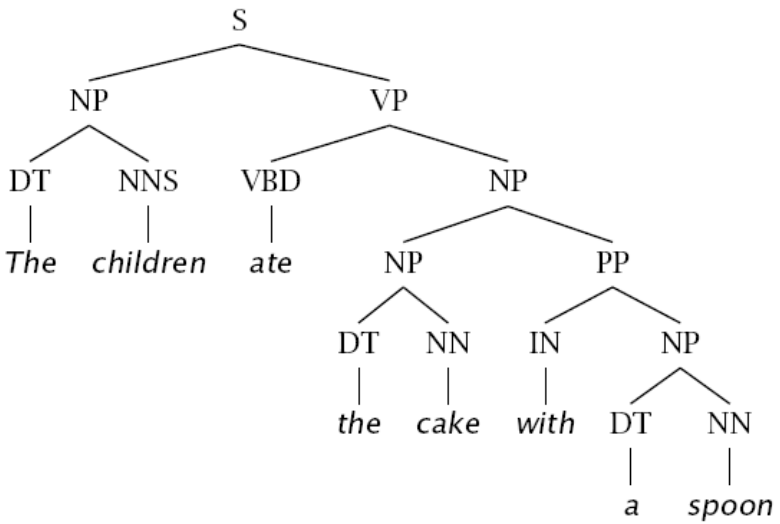
The Game of Designing a Grammar



- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Structural annotation [Johnson '98, Klein and Manning 03]
  - Head lexicalization [Collins '99, Charniak '00]

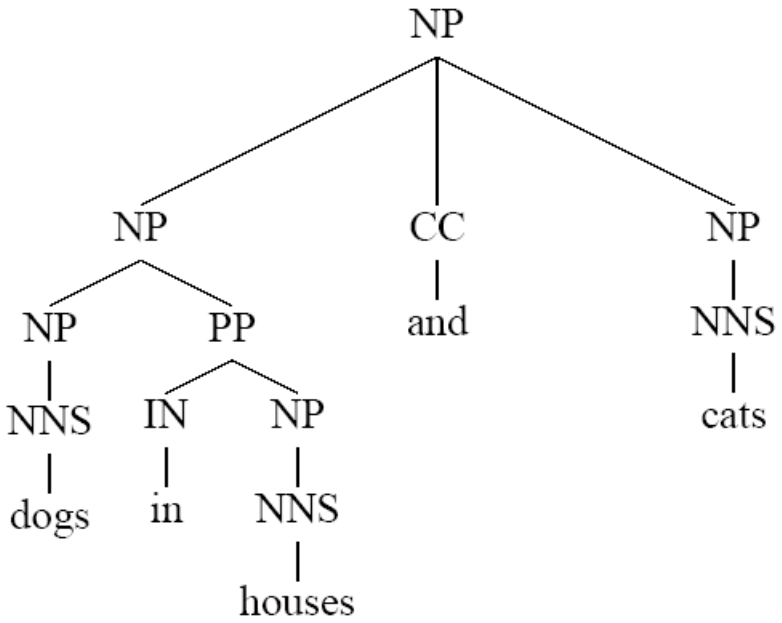
Problems with PCFGs

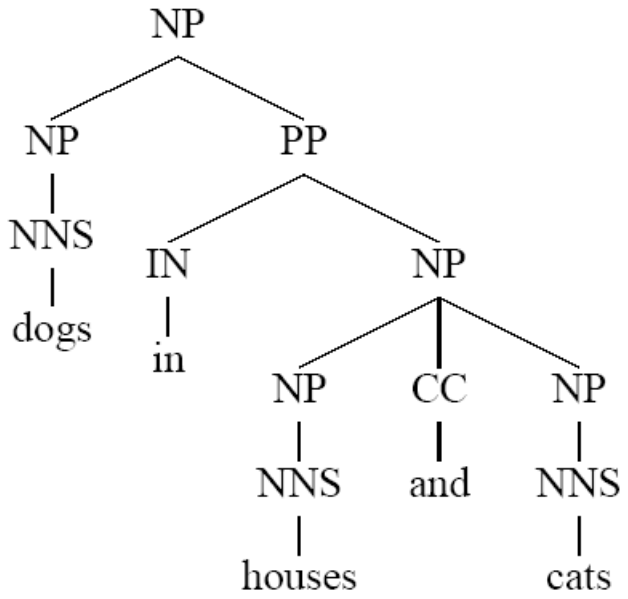




- If we do no annotation, these trees differ only in one rule:
  - VP → VP PP
  - NP → NP PP
- Parse will go one way or the other, regardless of words
- We addressed this in one way with unlexicalized grammars (how?)
- Lexicalization allows us to be sensitive to specific words

Problems with PCFGs

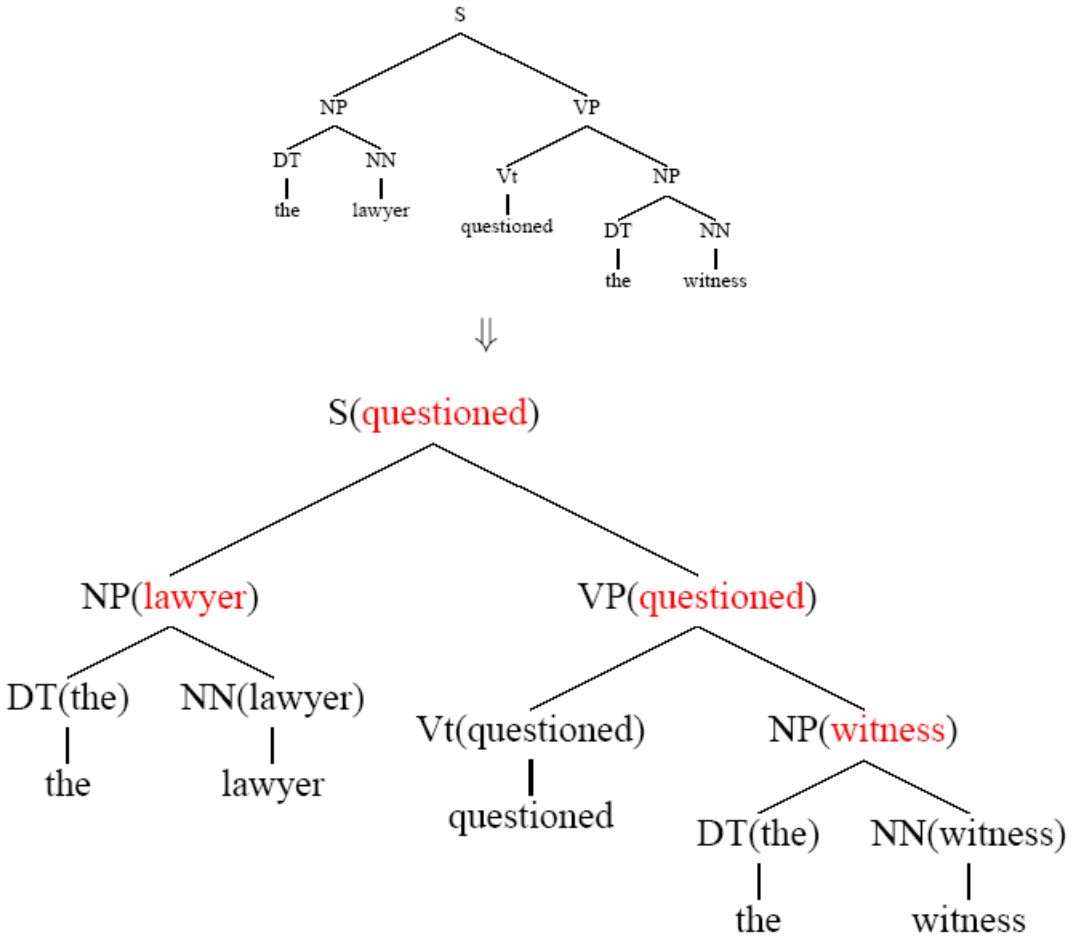




- What's different between basic PCFG scores here?
- What (lexical) correlations need to be scored?

Lexicalized Trees

- Add "head words" 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

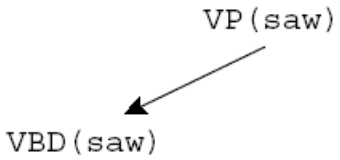
```
VP(saw) -> VBD(saw) NP-C(her) NP(today)
```

- Never going to get these atomically off of a treebank

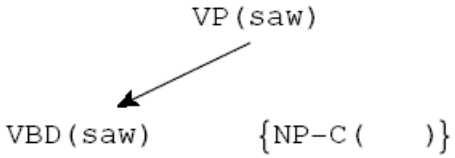
- Solution: break up derivation into smaller steps

Lexical Derivation Steps

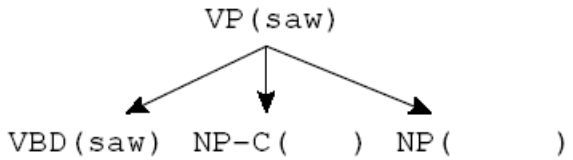
- A derivation of a local tree [Collins 99]



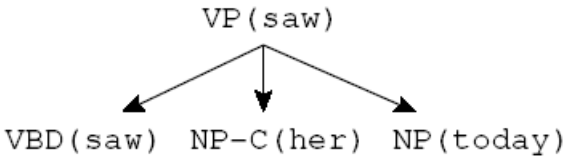
Choose a head tag and word



Choose a complement bag

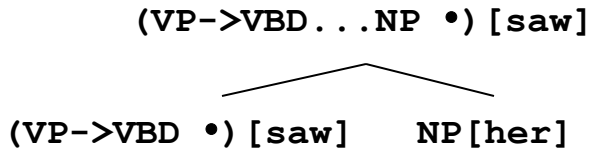


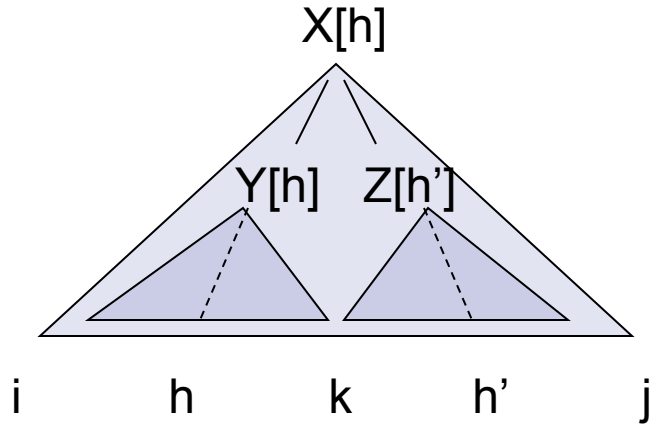
Generate children (incl. adjuncts)



Recursively derive children

Lexicalized CKY



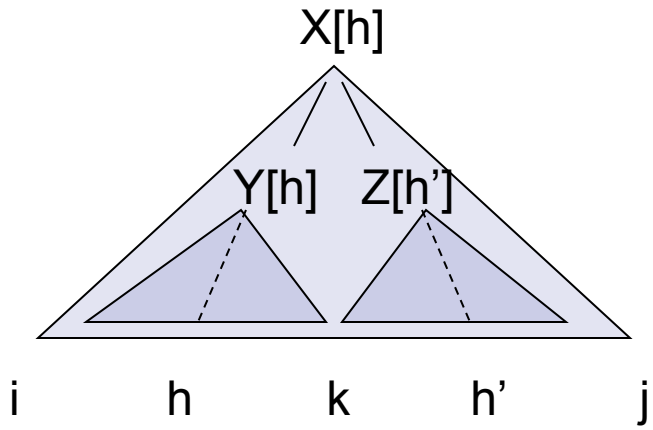


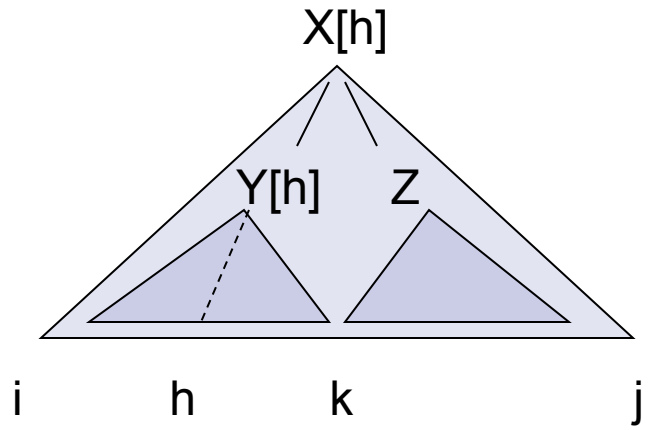

```
bestScore(X,i,j,h)
  if (j = i+1)
    return tagScore(X,s[i])
  else
    return
      max max score(X[h]->Y[h] Z[h']) *
    k,h',X->YZ
            bestScore(Y,i,k,h) *
            bestScore(Z,k,j,h')
        max score(X[h]->Y[h'] Z[h]) *
    k,h',X->YZ
            bestScore(Y,i,k,h') *
         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

Efficient Parsing for Lexical Grammars

Quartic Parsing

- Turns out, you can do (a little) better [Eisner 99]

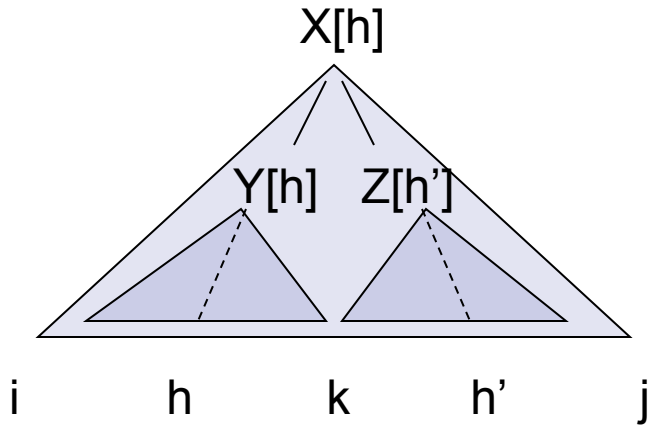




- Gives an $O(n^4)$ algorithm
- Still prohibitive in practice if not pruned

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 <i,j>.
  - 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 (and in practice is more like linear!)



- Also: certain spans are forbidden entirely on the basis of punctuation (crucial for speed)

Pruning with a PCFG

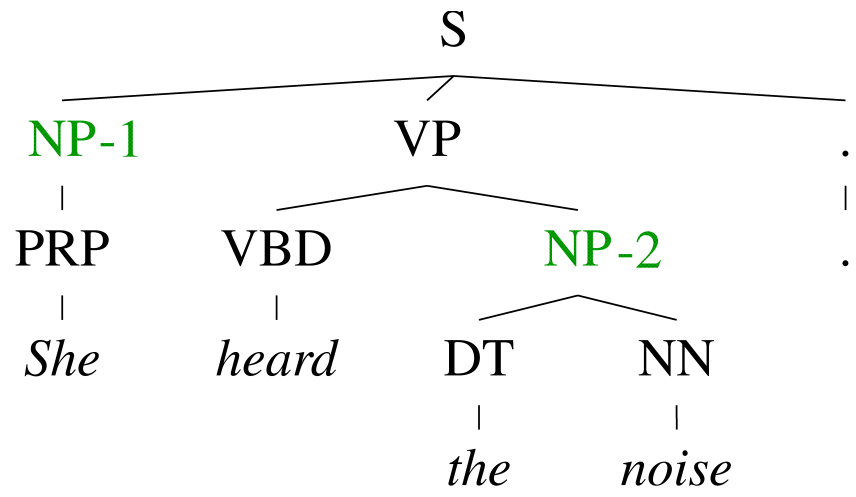
- The Charniak parser prunes using a two-pass, coarse-to-fine approach [Charniak 97+]
  - First, parse with the base grammar
  - For each X:[i,j] calculate P(X|i,j,s)
    - This isn't trivial, and there are clever speed ups
  - Second, do the full $O(n^5)$ CKY
    - Skip any X :[i,j] which had low (say, < 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

Results

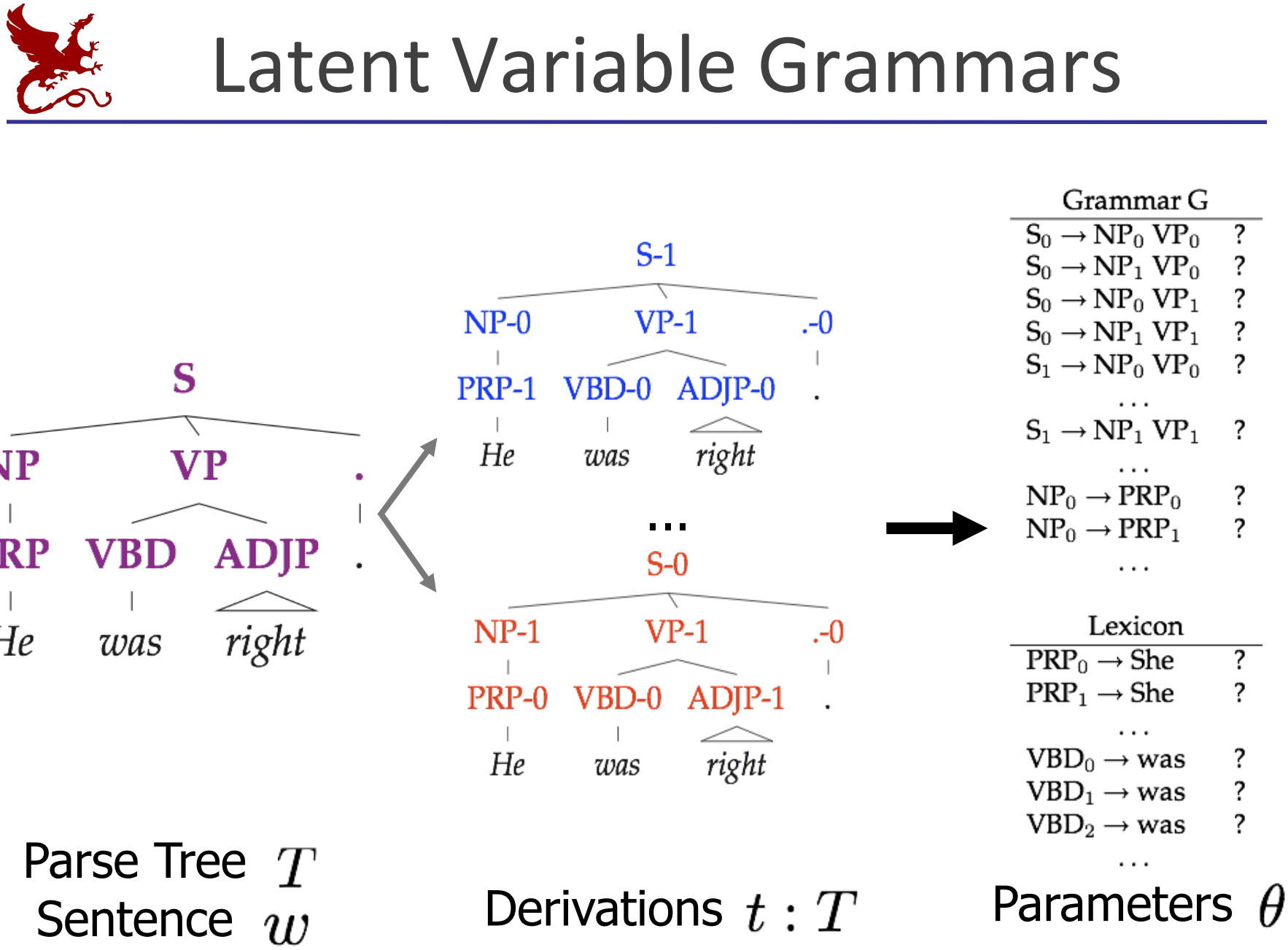
- Some results
  - Collins 99 – 88.6 F1 (generative lexical)
  - Charniak and Johnson 05 – 89.7 / 91.3 F1 (generative lexical / reranked)
  - Petrov et al 06 – 90.7 F1 (generative unlexical)
  - McClosky et al 06 – 92.1 F1 (gen + rerank + self-train)
- However
  - Bilexical counts rarely make a difference (why?)
  - Gildea 01 – Removing bilexical counts costs < 0.5 F1

Latent Variable PCFGs

The Game of Designing a Grammar



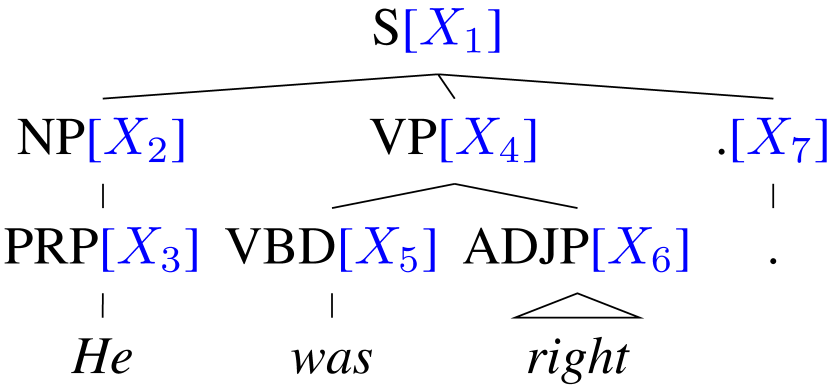
- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson '98]
  - Head lexicalization [Collins '99, Charniak '00]
  - Automatic clustering?

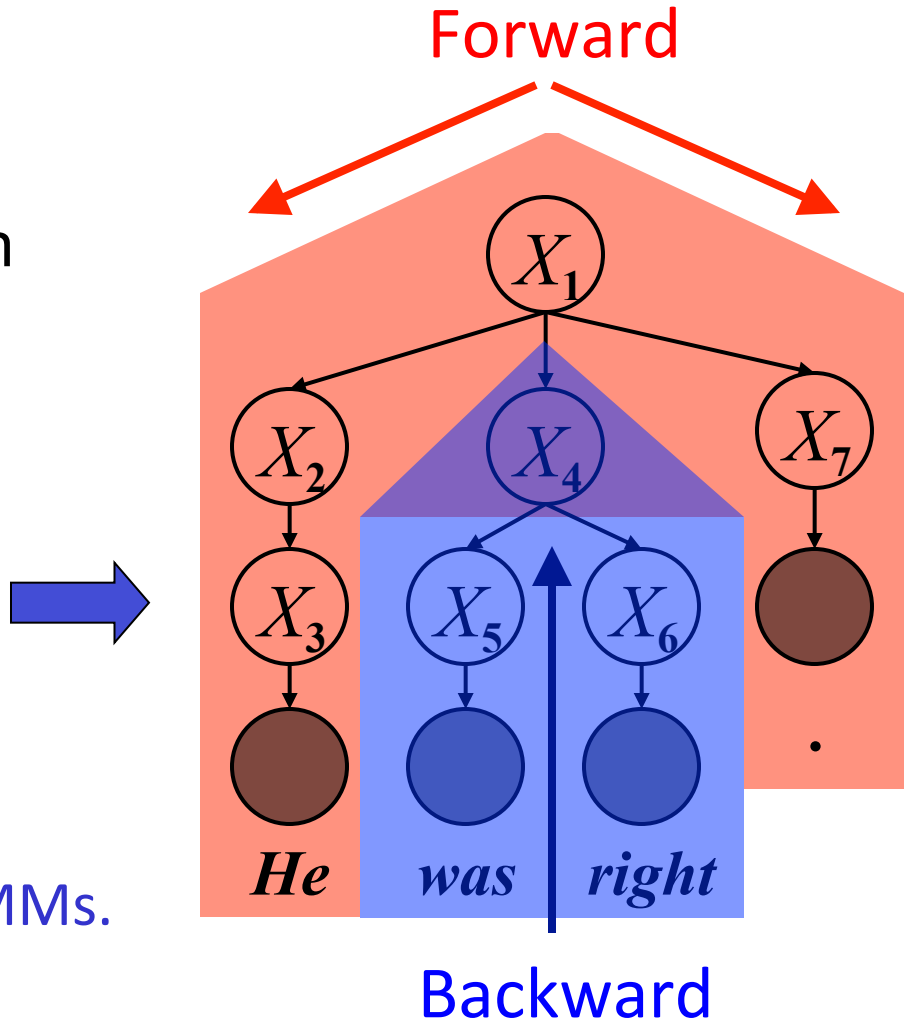

Latent Variable Grammars
S
NP
VP
.
PRP
VBD
ADJP
.
He
was
right
S-1
NP-0
VP-1
.-0
PRP-1
VBD-0
ADJP-0
.
He
was
right
...
S-0
NP-1
VP-1
.-0
PRP-0
VBD-0
ADJP-1
.
He
was
right
Grammar G
$S_0 \rightarrow NP_0\ VP_0$?
$S_0 \rightarrow NP_1\ VP_0$?
$S_0 \rightarrow NP_0\ VP_1$?
$S_0 \rightarrow NP_1\ VP_1$?
$S_1 \rightarrow NP_0\ VP_0$?
...
$S_1 \rightarrow NP_1\ VP_1$?
...
$NP_0 \rightarrow PRP_0$?
$NP_0 \rightarrow PRP_1$?
...
Lexicon
$PRP_0 \rightarrow$ She ?
$PRP_1 \rightarrow$ She ?
...
$VBD_0 \rightarrow$ was ?
$VBD_1 \rightarrow$ was ?
$VBD_2 \rightarrow$ was ?
...
Parse Tree T
Sentence w
Derivations $t : T$
Parameters θ

Learning Latent Annotations

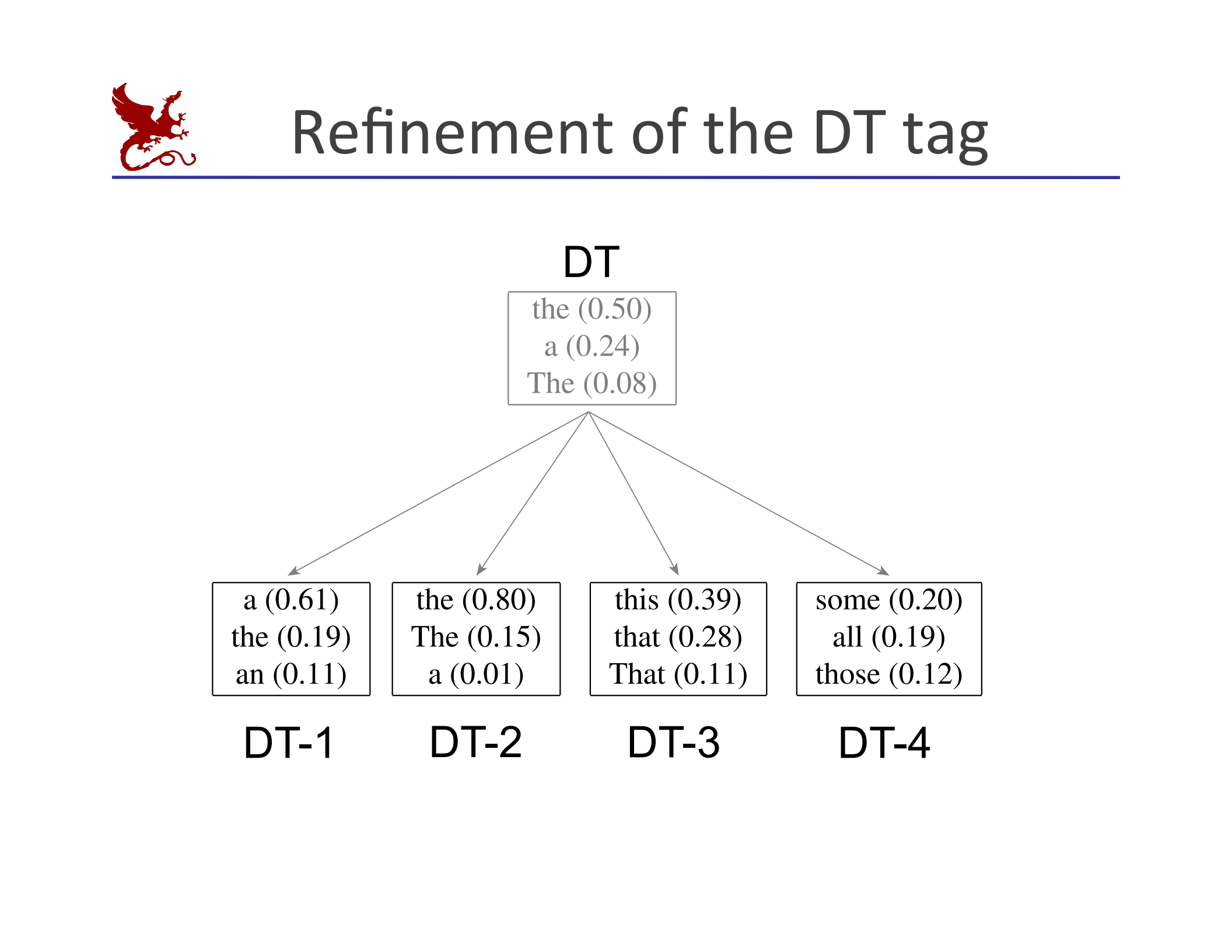
EM algorithm:

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

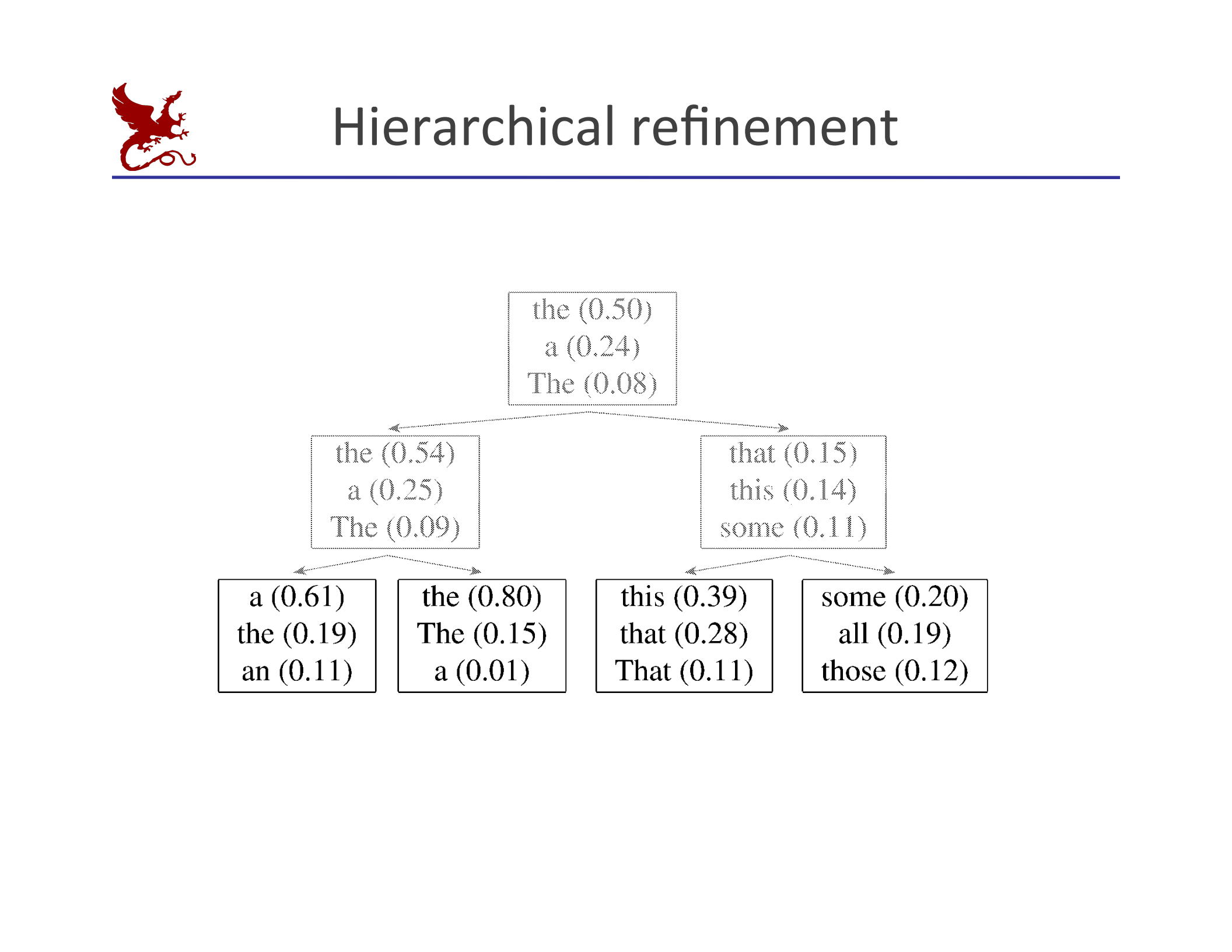




Just like Forward-Backward for HMMs.

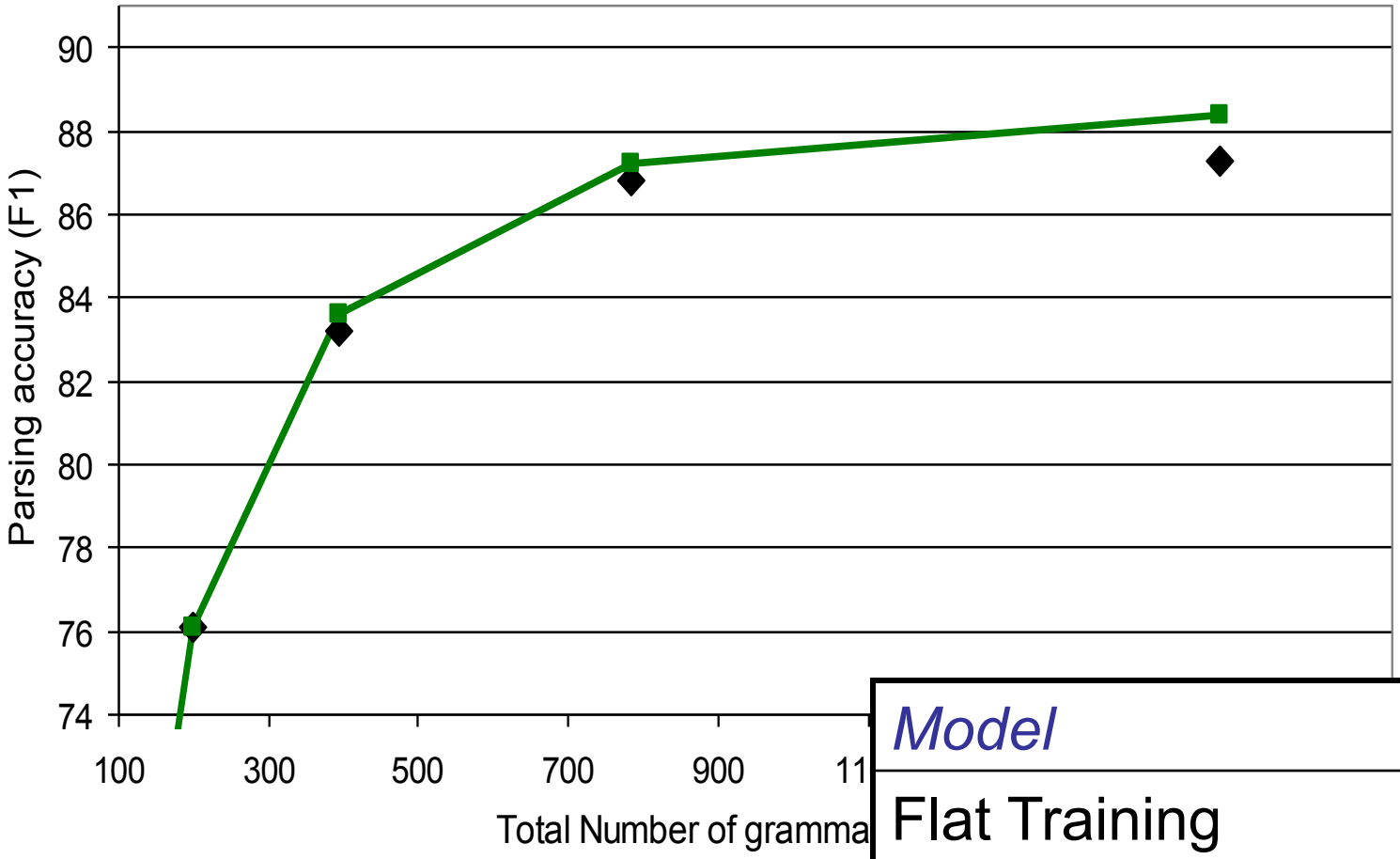


Refinement of the DT tag



Hierarchical refinement

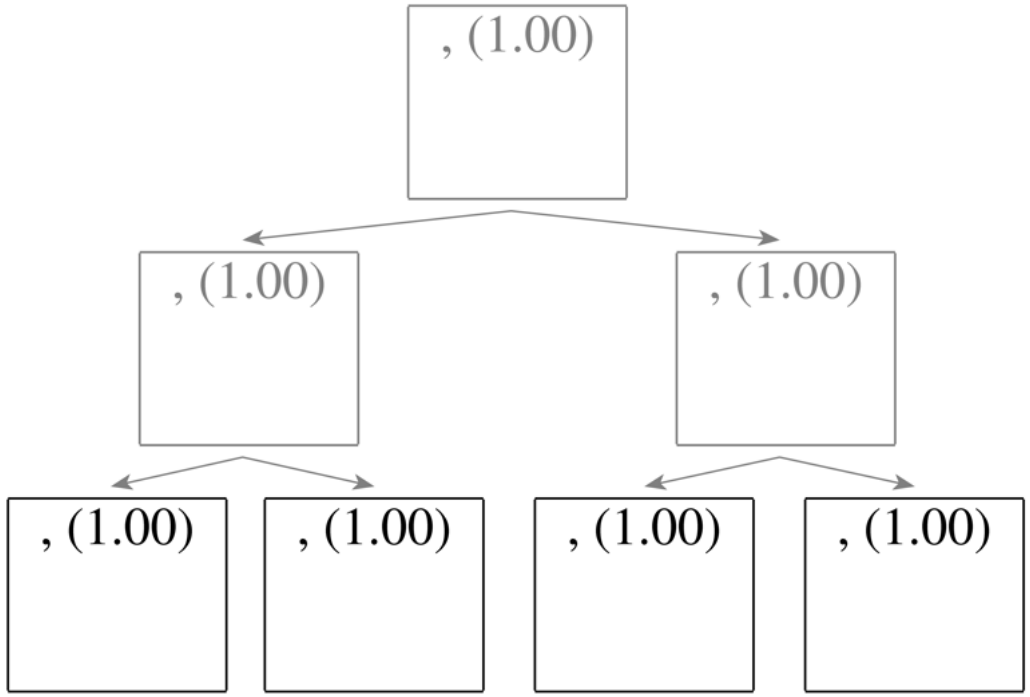
Hierarchical Estimation Results



Model	*F1*
Flat Training	87.3
Hierarchical Training	88.4

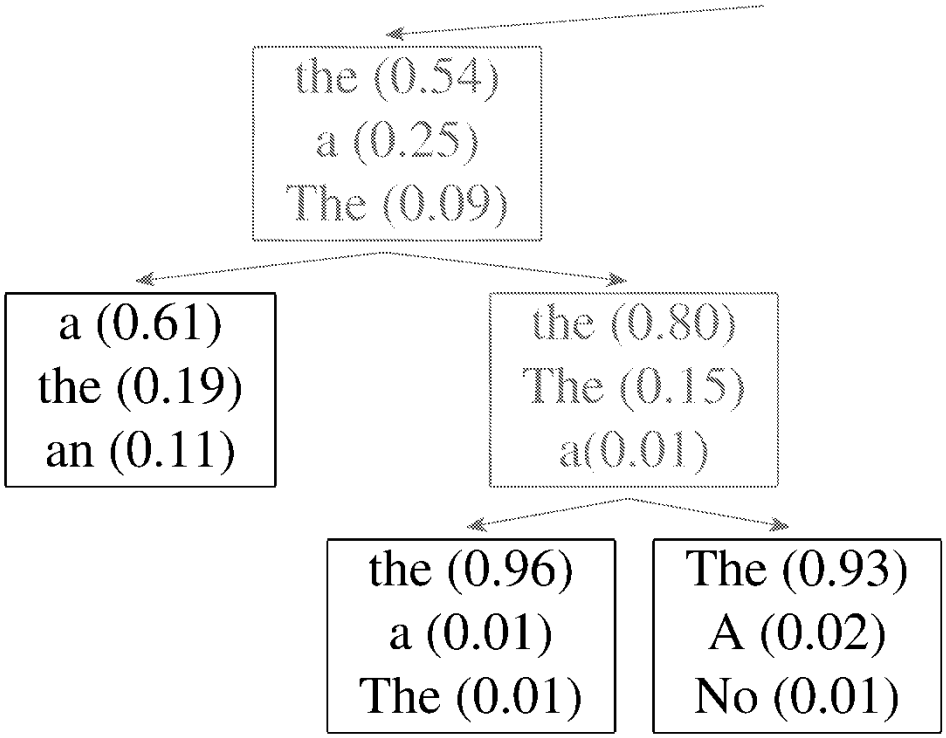
Refinement of the , tag

- Splitting all categories equally is wasteful:

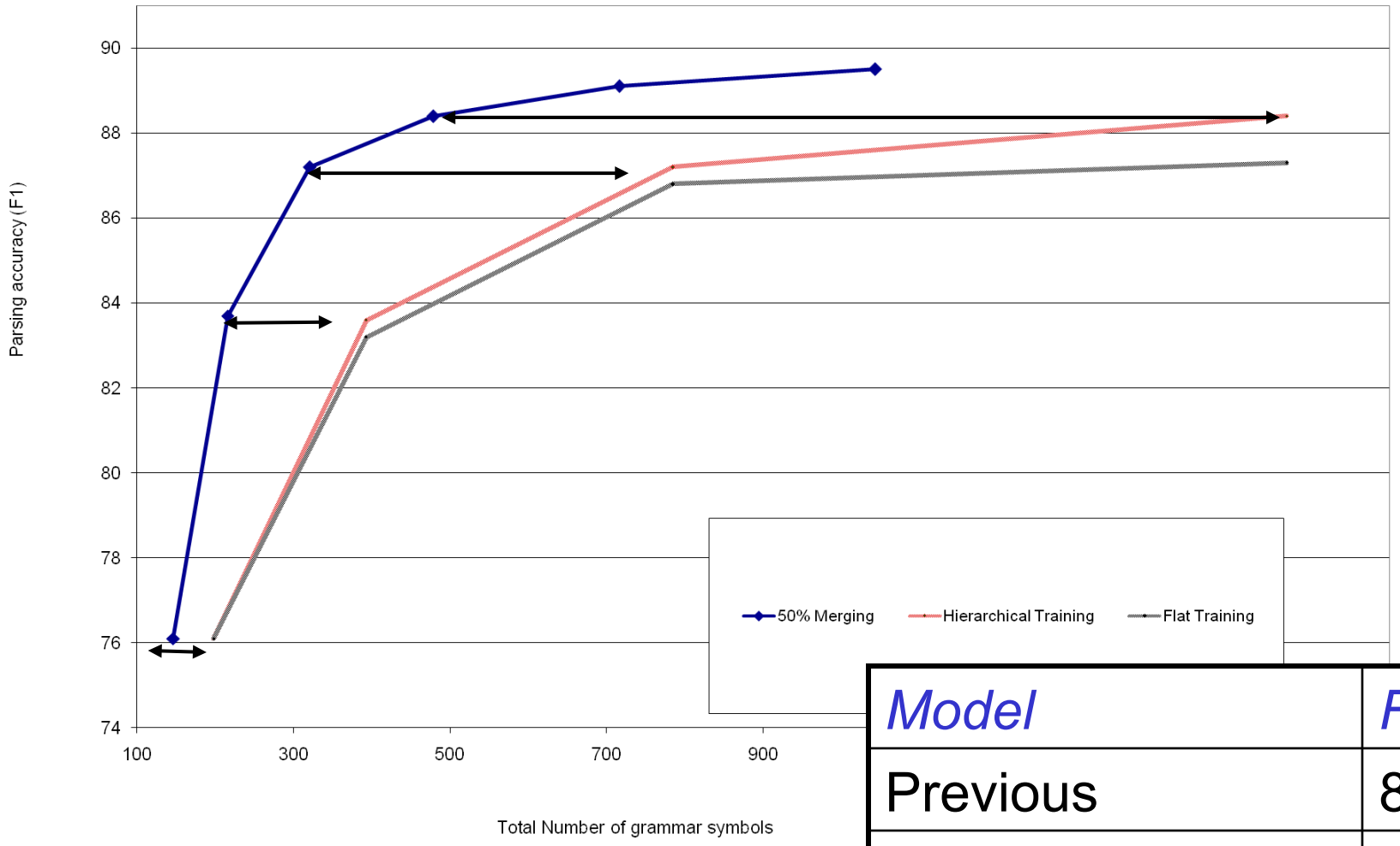


Adaptive Splitting

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

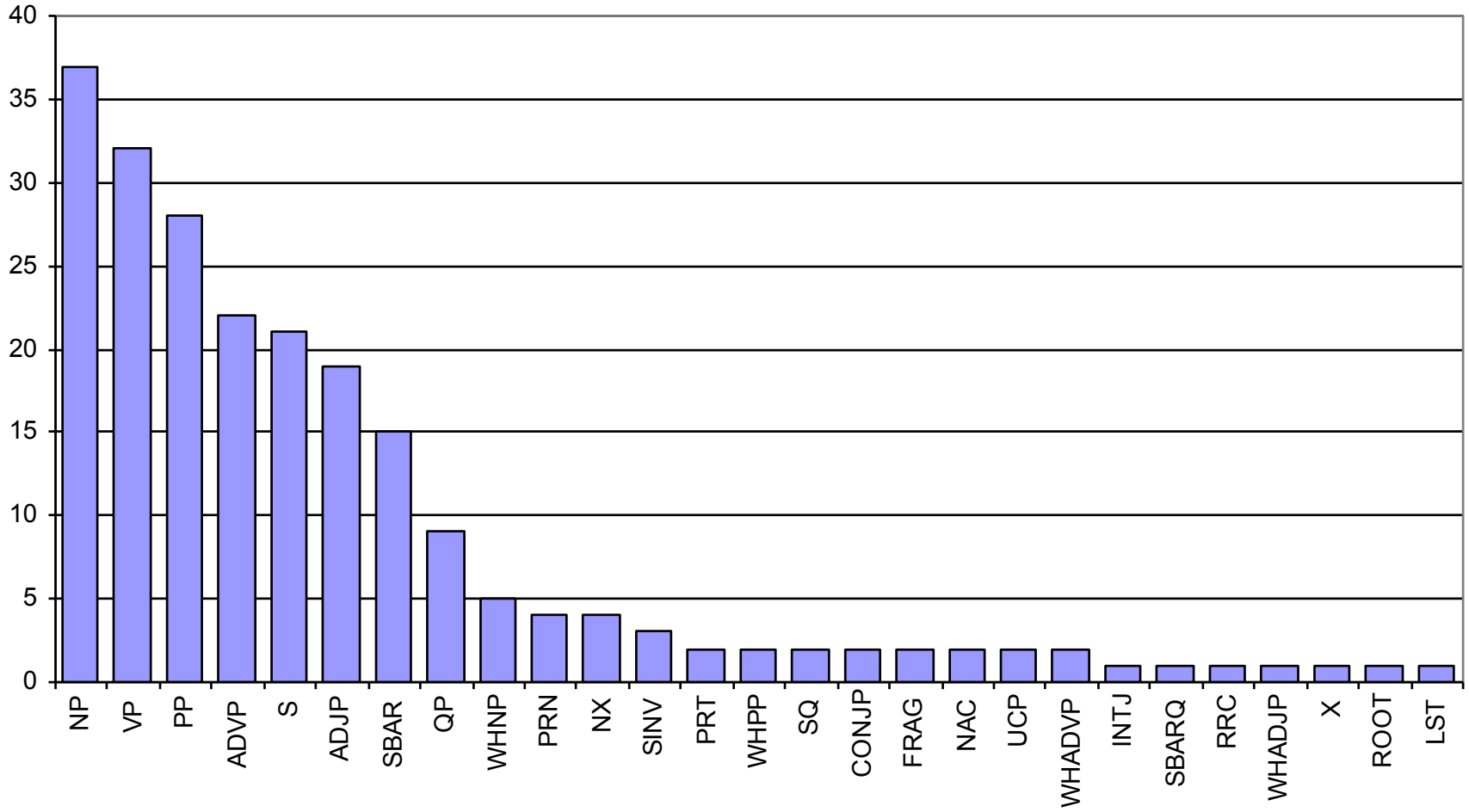


Adaptive Splitting Results

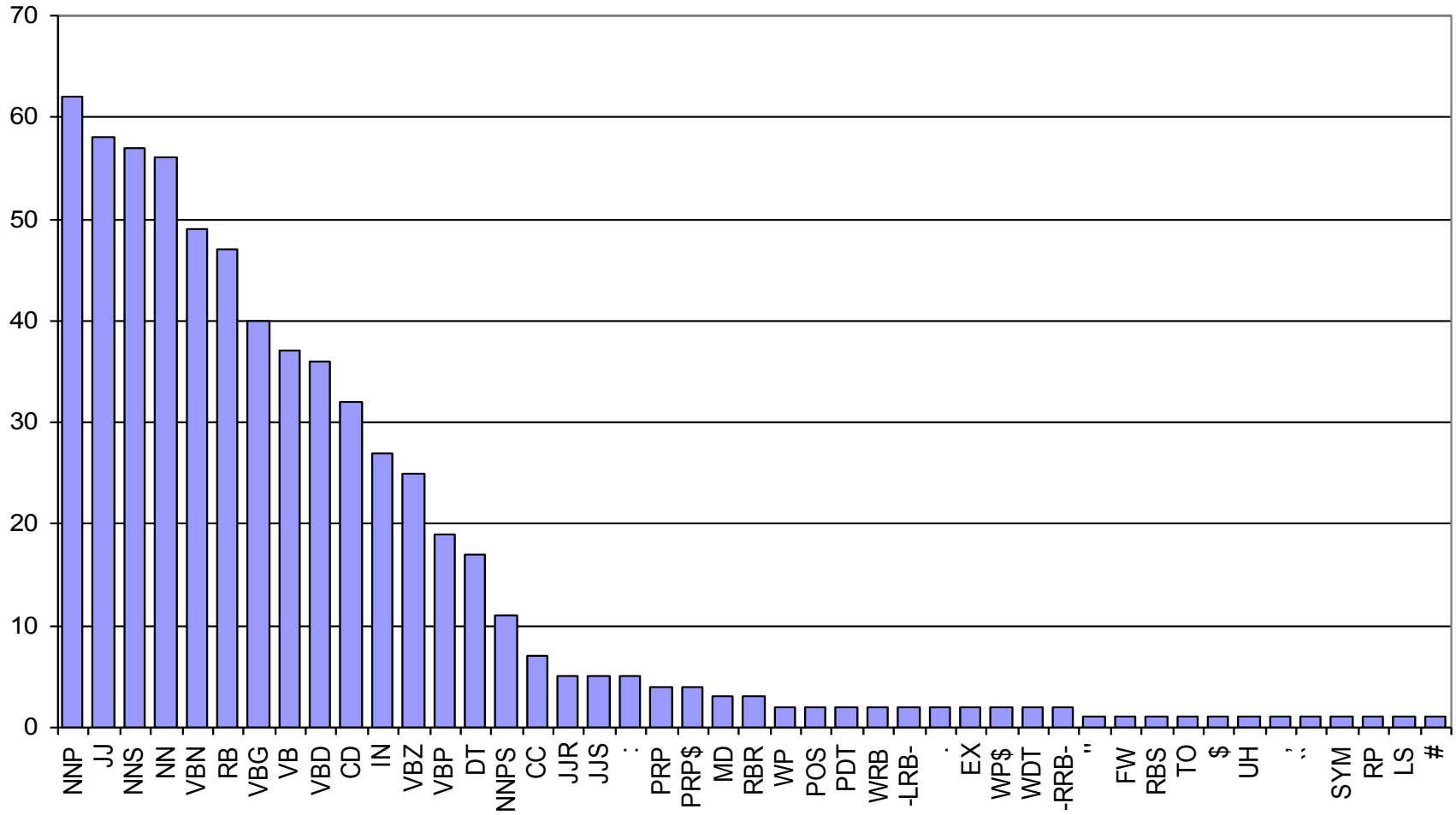


Model	*F1*
Previous	88.4
With 50% Merging	89.5

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

Final Results (Accuracy)

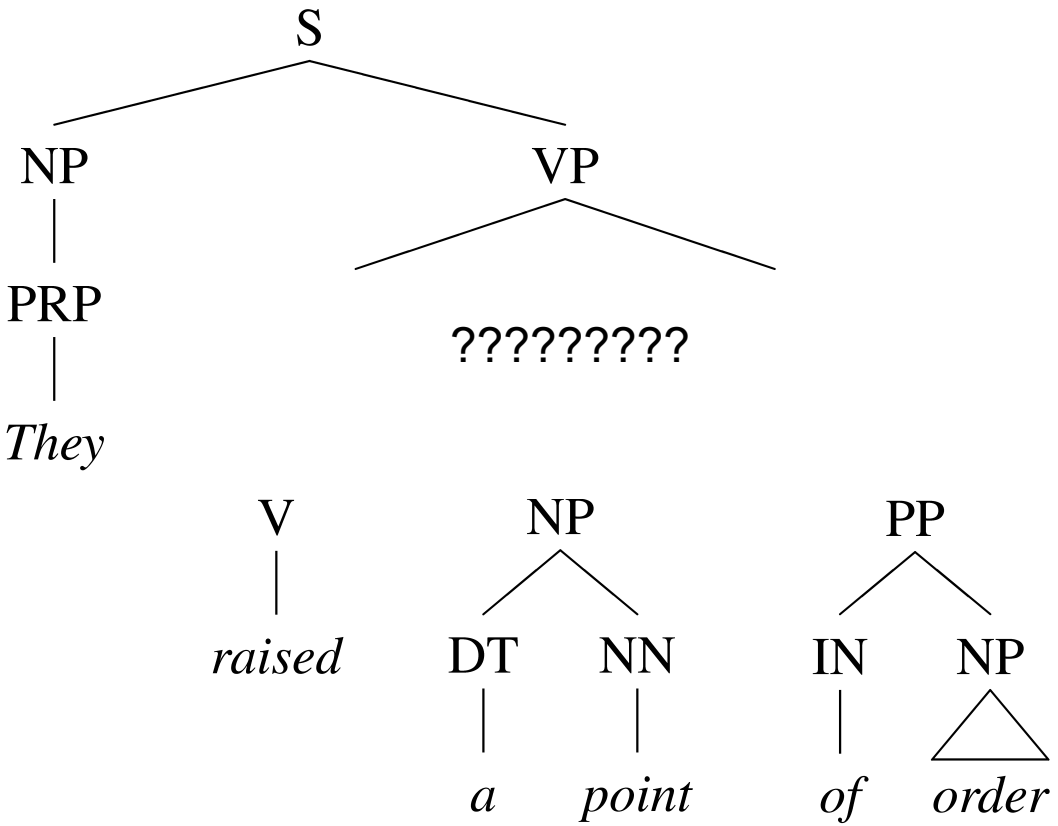
		≤ 40 words F1	all F1
ENG	Charniak&Johnson ‘05 (generative)	90.1	89.6
	Split / Merge	**90.6**	**90.1**
GER	Dubey ‘05	76.3	-
	Split / Merge	**80.8**	**80.1**
CHN	Chiang et al. ‘02	80.0	76.6
	Split / Merge	**86.3**	**83.4**

Still higher numbers from reranking / self-training methods

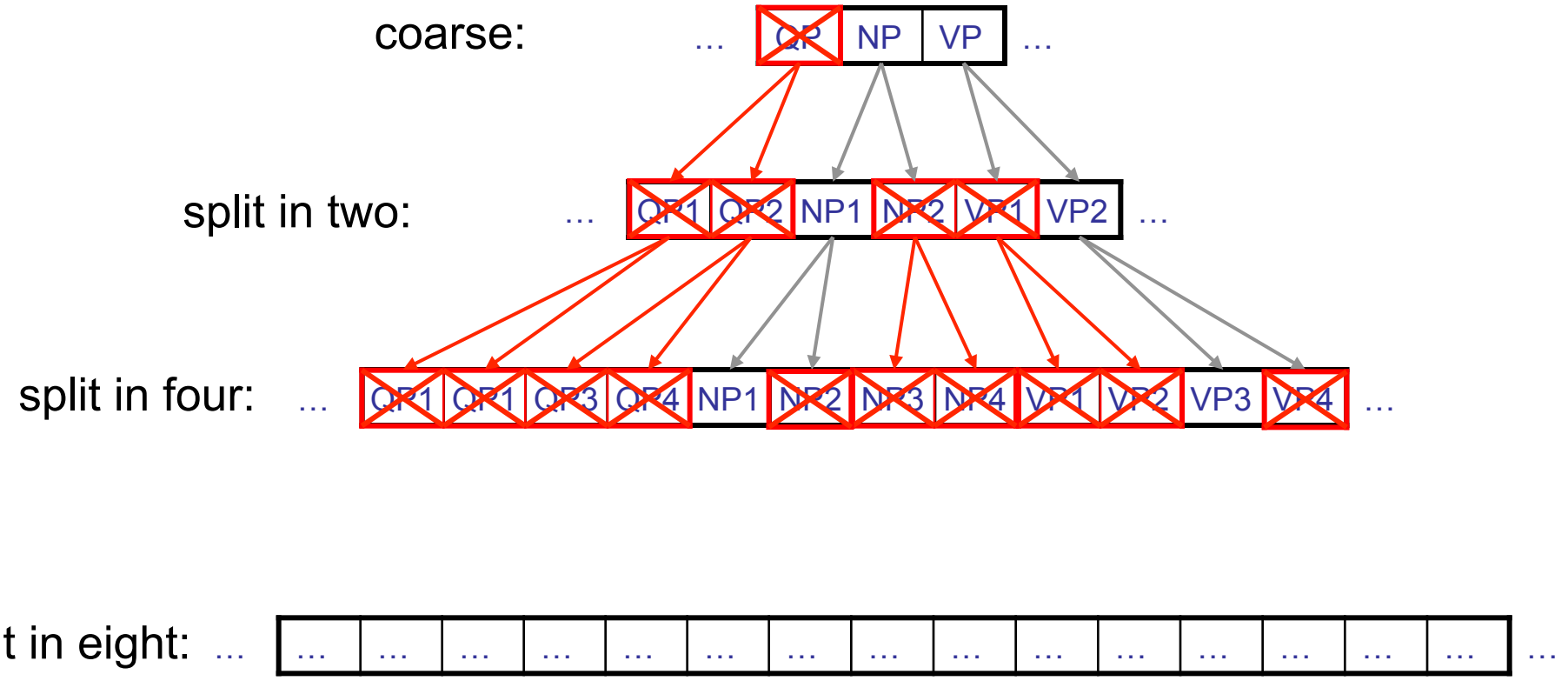
Efficient Parsing for Hierarchical Grammars

Coarse-to-Fine Inference

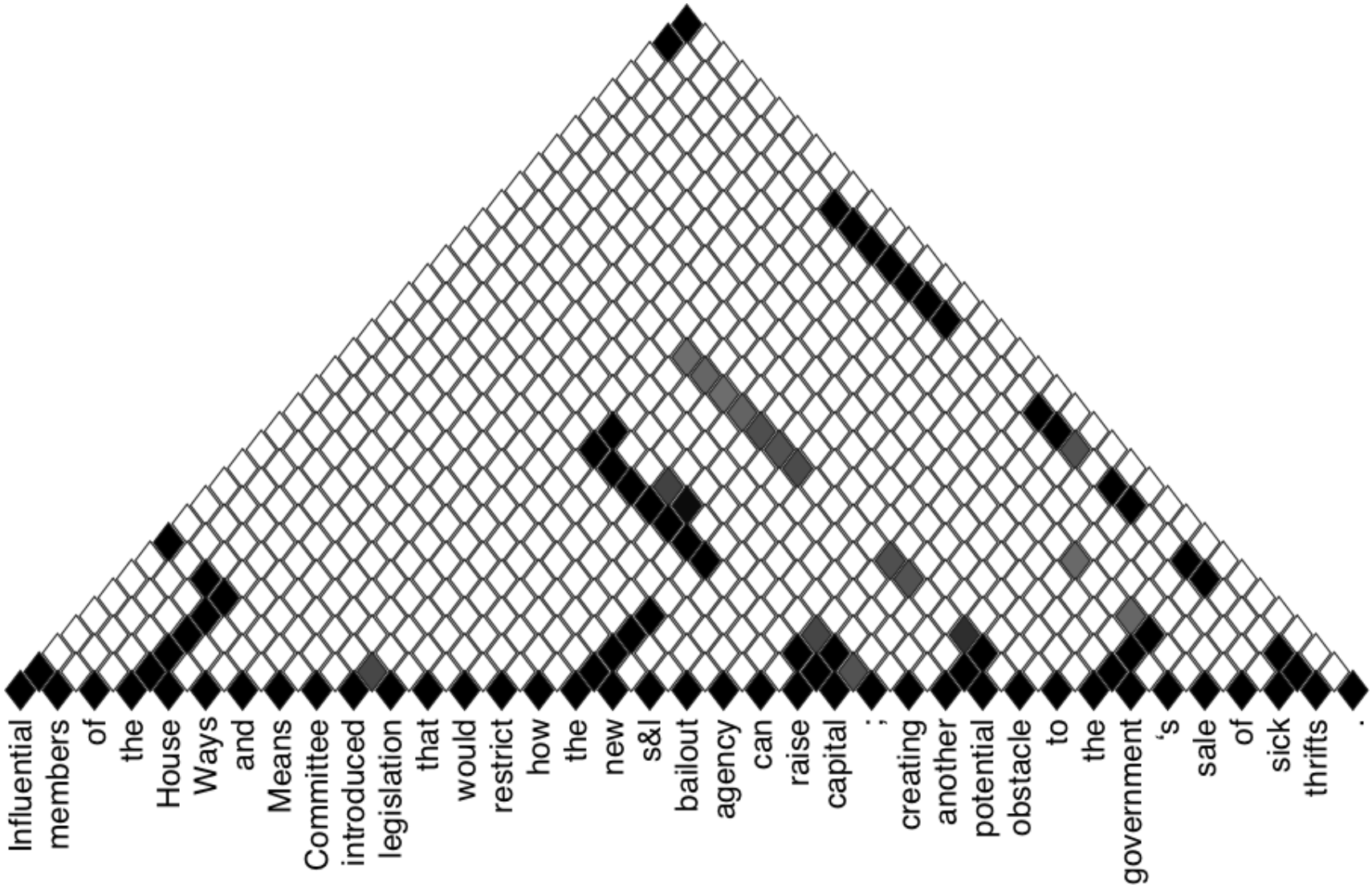
- Example: PP attachment



Hierarchical Pruning



Bracket Posteriors



1621 min

111 min

35 min

15 min

(no search error)

Unsupervised Tagging

Unsupervised Tagging?

- AKA part-of-speech induction
- Task:
  - Raw sentences in
  - Tagged sentences out
- Obvious thing to do:
  - Start with a (mostly) uniform HMM
  - Run EM
  - Inspect results

EM for HMMs: Process

- Alternate between recomputing distributions over hidden variables (the tags) and reestimating parameters
- Crucial step: we want to tally up how many (fractional) counts of each kind of transition and emission we have under current params:

$$\text{count}(w, s) = \sum_{i:w_i=w} P(t_i = s|\mathbf{w})$$

$$\text{count}(s \rightarrow s') = \sum_{i} P(t_{i-1} = s, t_i = s'|\mathbf{w})$$

- Same quantities we needed to train a CRF!

Merialdo: Setup

- Some (discouraging) experiments [Merialdo 94]

- Setup:
  - You know the set of allowable tags for each word
  - Fix k training examples to their true labels
    - Learn $P(w|t)$ on these examples
    - Learn $P(t|t_{-1},t_{-2})$ on these examples
  - On n examples, re-estimate with EM

- Note: we know allowed tags but not frequencies

Merialdo: Results

Number of tagged sentences used for the initial model							
	0	100	2000	5000	10000	20000	all
Iter	Correct tags (% words) after ML on 1M words						
0	77.0	90.0	95.4	96.2	96.6	96.9	97.0
1	80.5	92.6	95.8	96.3	96.6	96.7	96.8
2	81.8	93.0	95.7	96.1	96.3	96.4	96.4
3	83.0	93.1	95.4	95.8	96.1	96.2	96.2
4	84.0	93.0	95.2	95.5	95.8	96.0	96.0
5	84.8	92.9	95.1	95.4	95.6	95.8	95.8
6	85.3	92.8	94.9	95.2	95.5	95.6	95.7
7	85.8	92.8	94.7	95.1	95.3	95.5	95.5
8	86.1	92.7	94.6	95.0	95.2	95.4	95.4
9	86.3	92.6	94.5	94.9	95.1	95.3	95.3
10	86.6	92.6	94.4	94.8	95.0	95.2	95.2