Algorithms for NLP

Parsing V

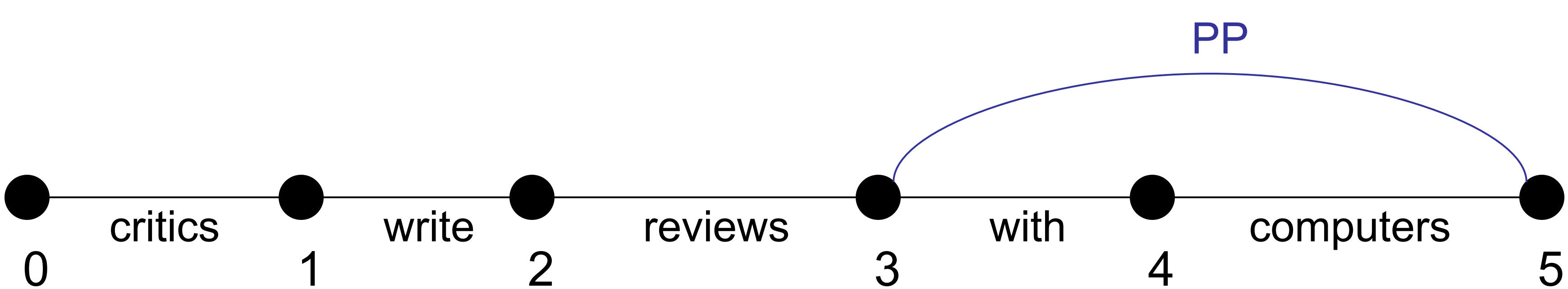
Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley

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]

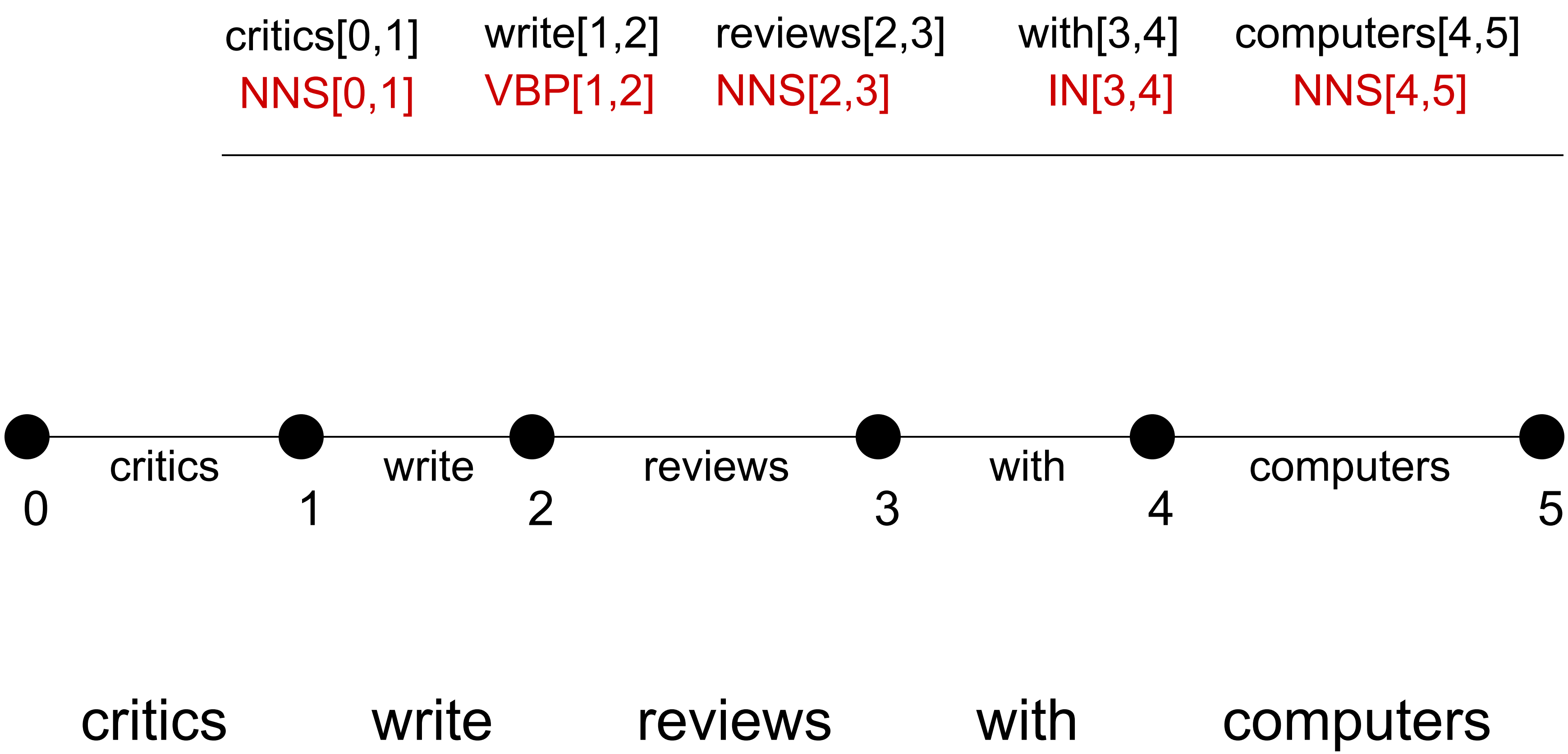
CHART [EMPTY]

0 1 2 3 4 5

critics write reviews with computers

Unary Projection

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



Item Successors

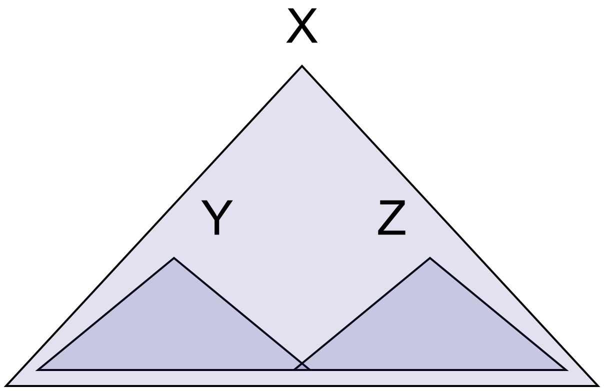
- When we pop items off of the agenda:
  - Graph successors: unary projections (NNS → critics, NP → NNS)

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

  - 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)

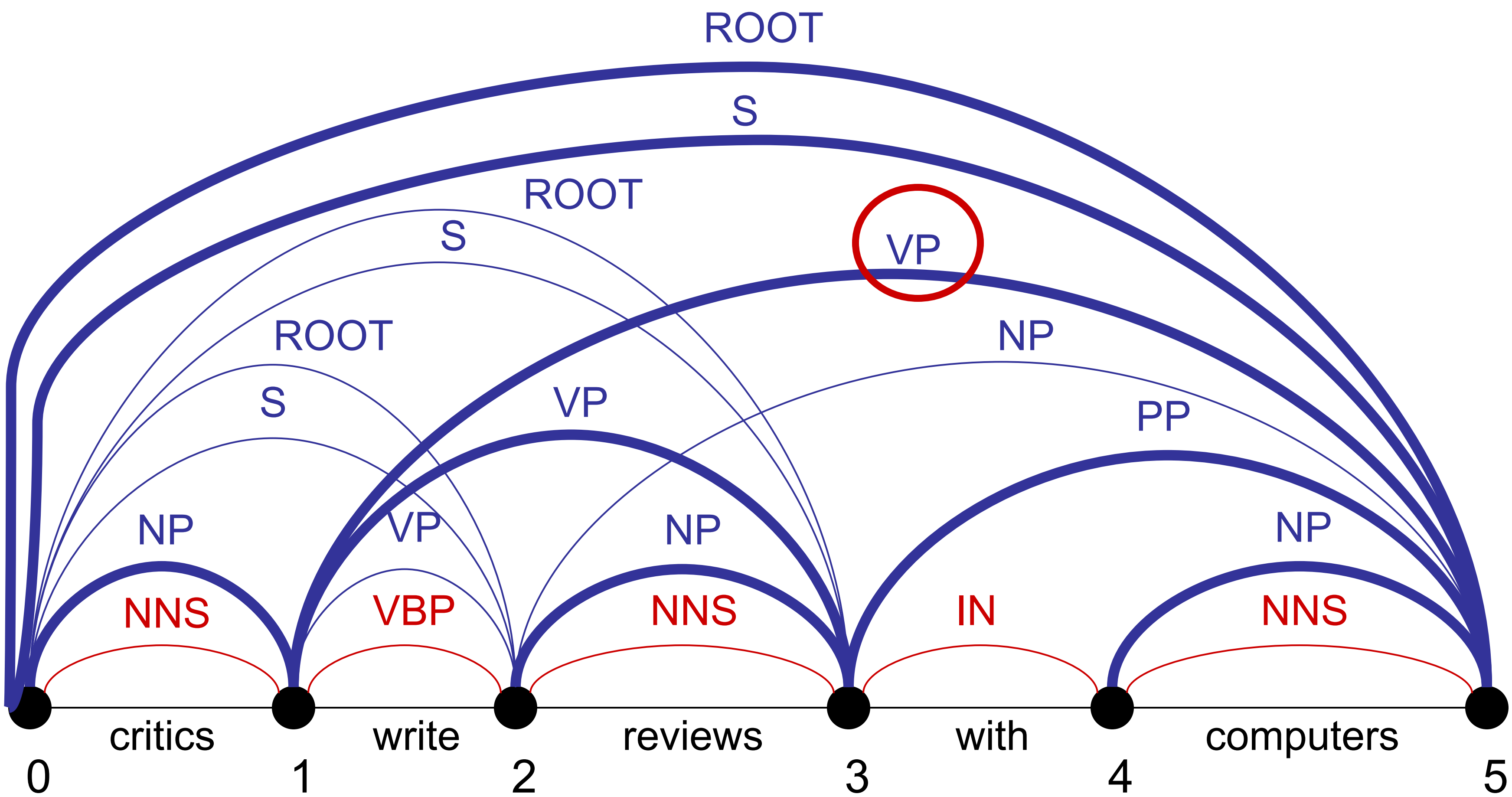


- Queries a chart must support:
  - Is edge X[i,j] in the chart? (What score?)
  - What edges with label Y end at position j?
  - What edges with label Z start at position i?

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]



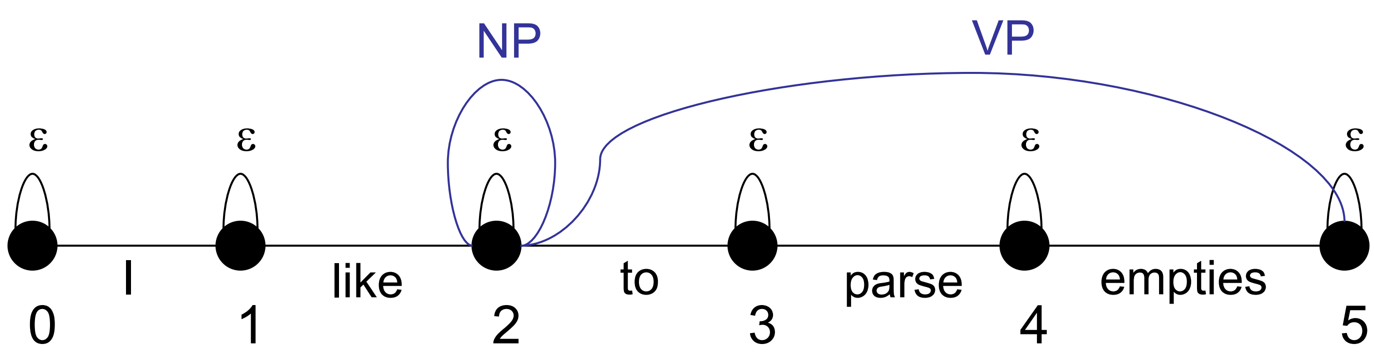
Empty Elements

- 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

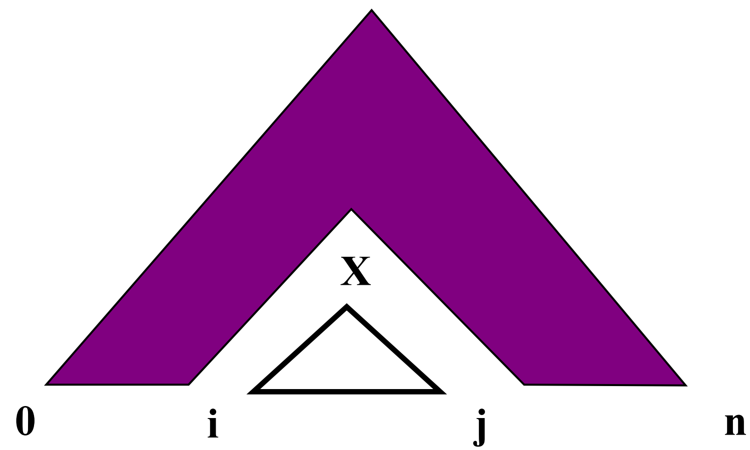
- These are easy to add to a agenda-based parser!
  - For each position i, add the "word" edge ε[i,i]
  - Add rules like NP → ε to the grammar
  - That's it!



UCS / A*

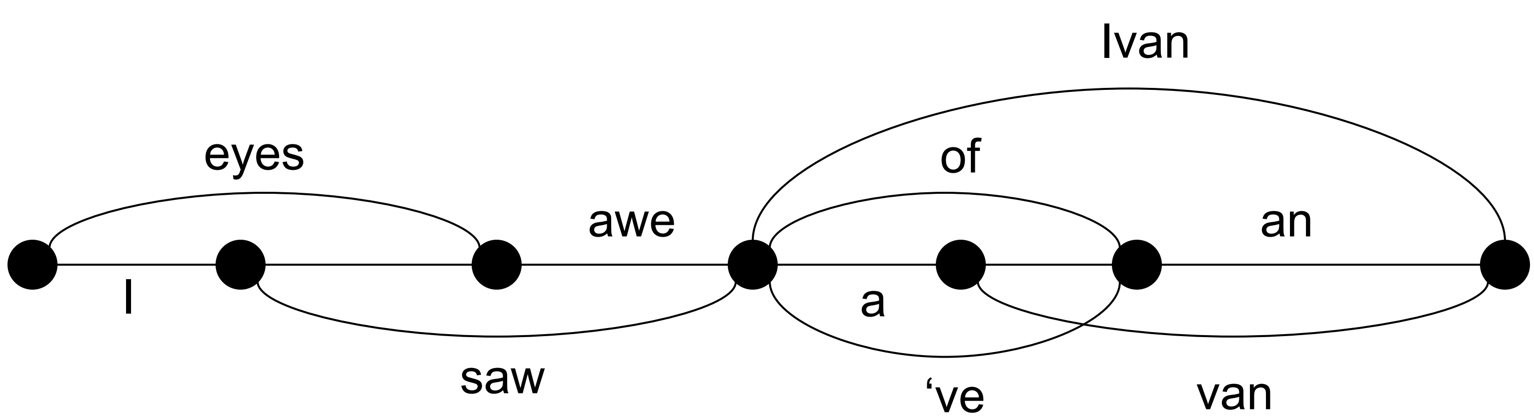
- With weighted edges, order matters
  - Must expand optimal parse from bottom up (subparses first)
  - CKY does this by processing smaller spans before larger ones
  - UCS pops items off the agenda in order of decreasing Viterbi score
  - A* search also well defined

- 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

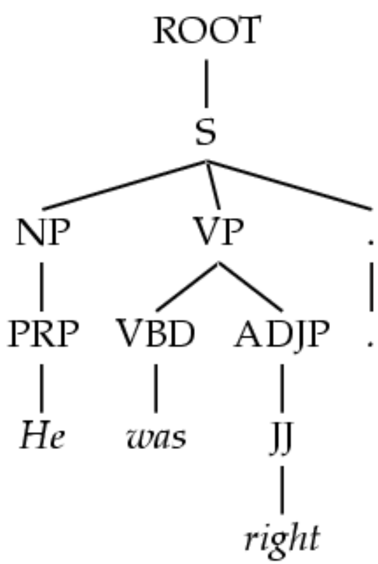
- There was nothing magical about words spanning exactly one position.
- When working with speech, we generally don't know how many words there are, or where they break.
- We can represent the possibilities as a lattice and parse these just as easily.



Learning PCFGs

Treebank PCFGs [Charniak 96]

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

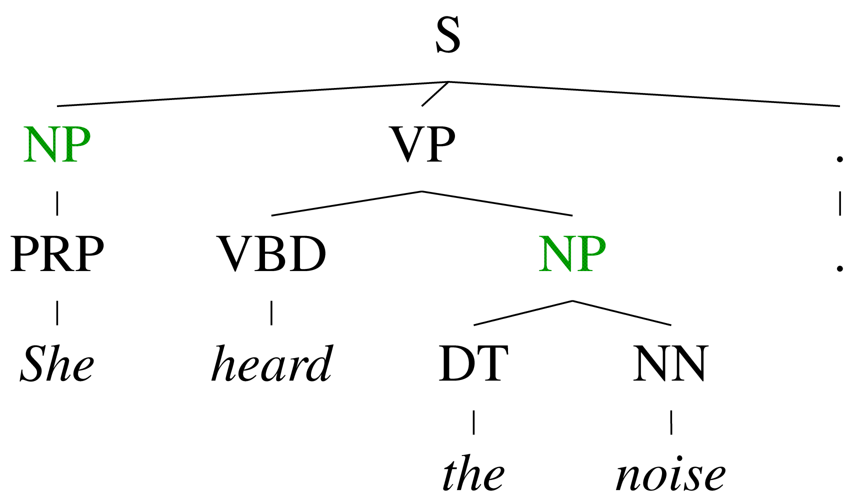


ROOT → S	1
S → NP VP .	1
NP → PRP	1
VP → VBD ADJP	1

.....

Model	*F1*
Baseline	72.0

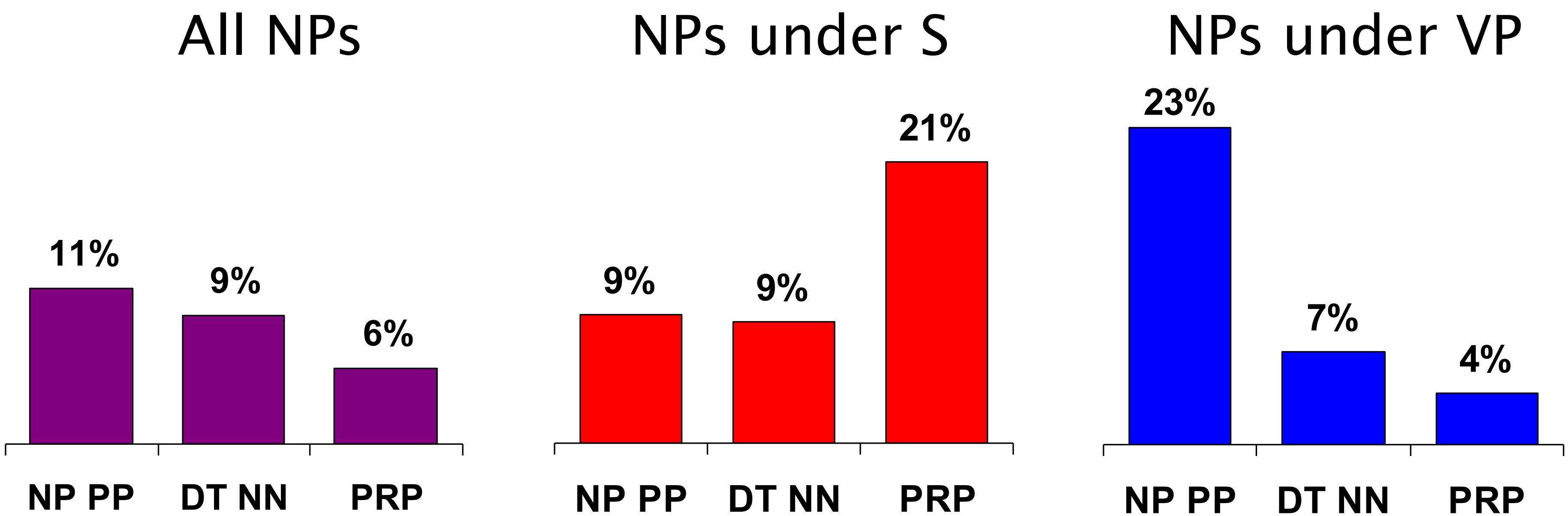
Conditional Independence?



- 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

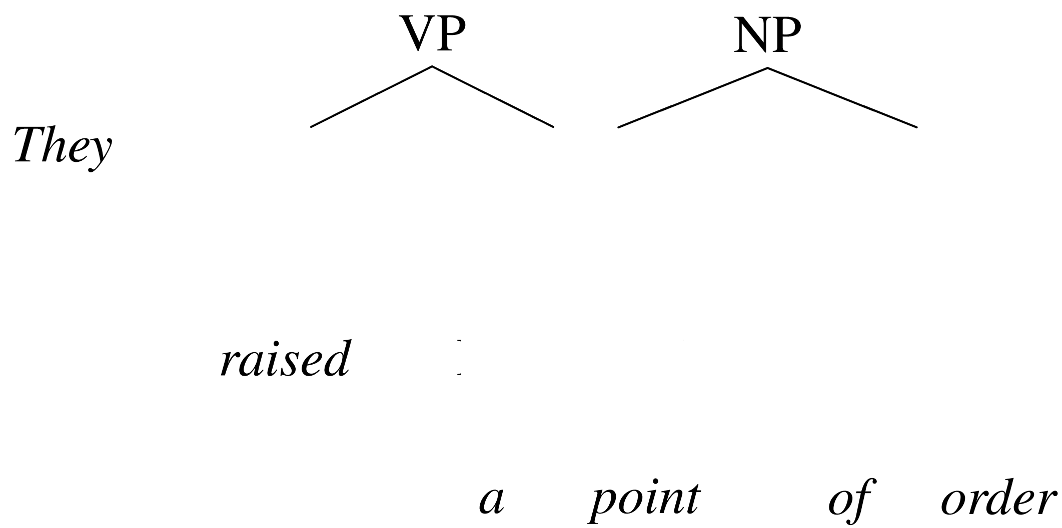
- Independence assumptions are often too strong.



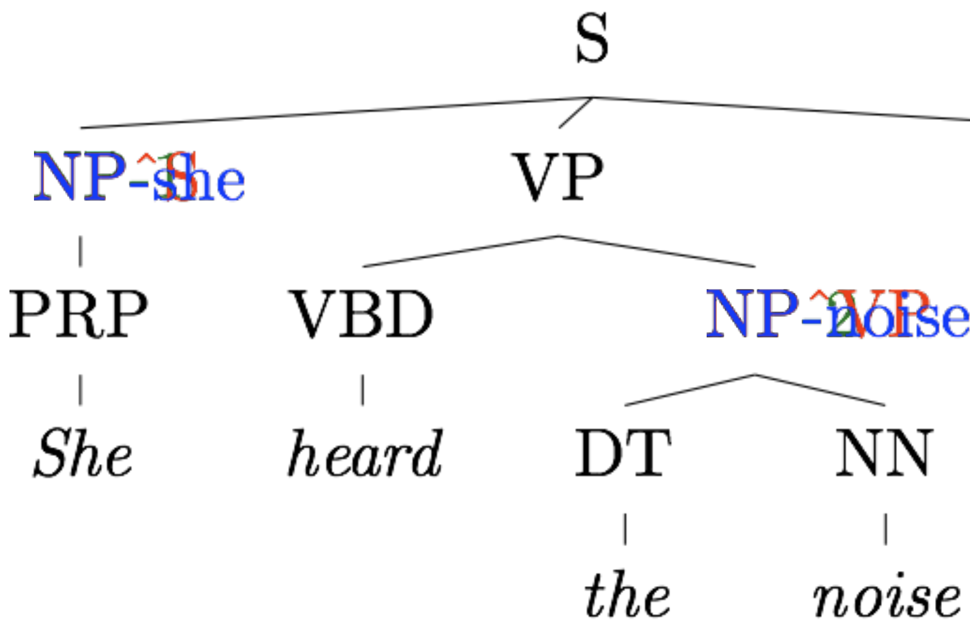
- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).
- Also: the subject and object expansions are correlated!

Grammar Refinement

- Example: PP attachment



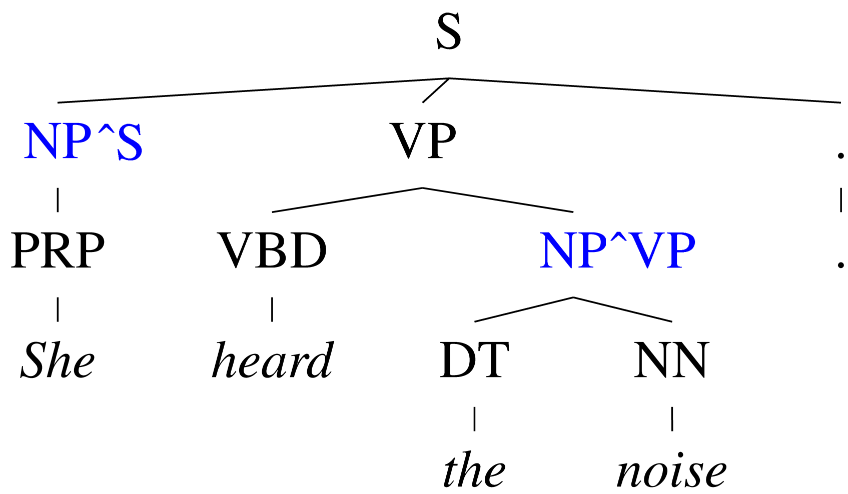
Grammar Refinement



- 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

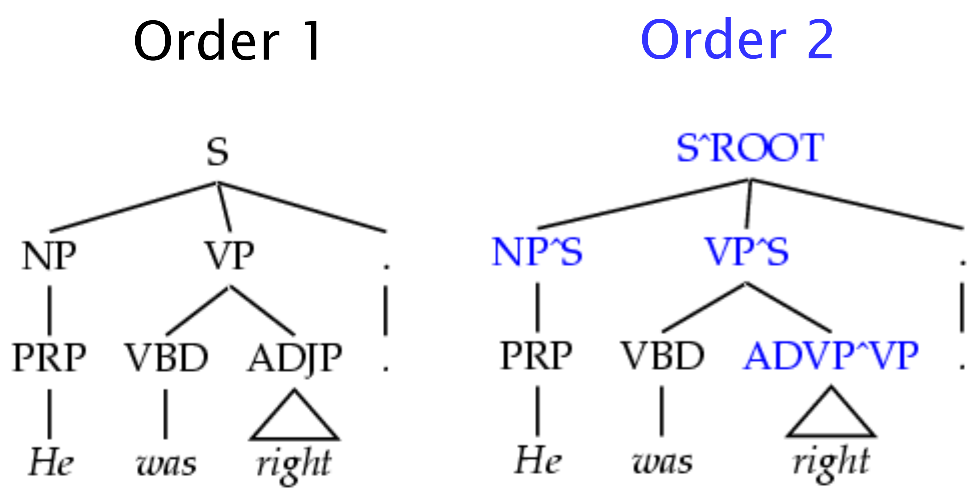
- Corpus: Penn Treebank, WSJ

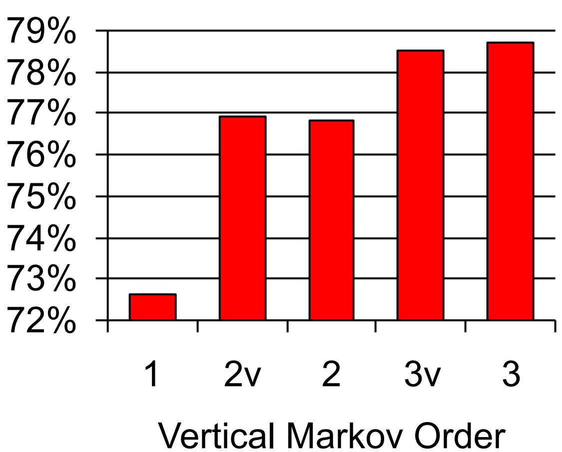
Training:	sections	02-21
Development:	section	22 (here, first 20 files)
Test:	section	23

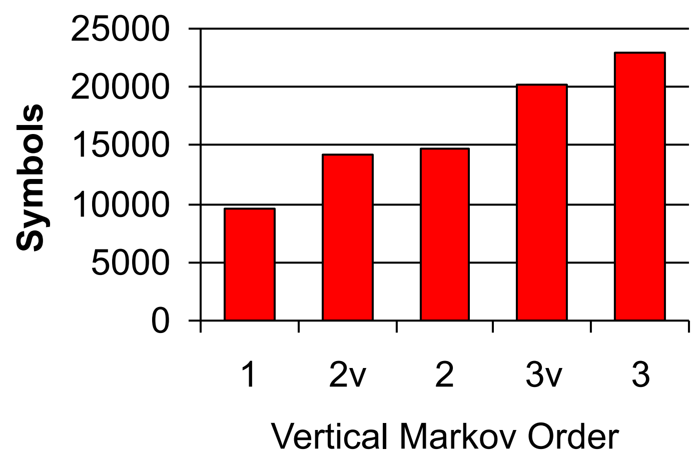
- Accuracy – F1: harmonic mean of per-node labeled precision and recall.
- Here: also size – number of symbols in grammar.

Vertical Markovization

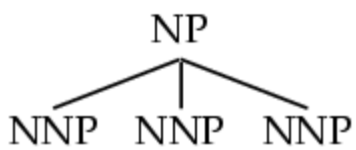
- Vertical Markov order: rewrites depend on past k ancestor nodes.
  (cf. parent annotation)



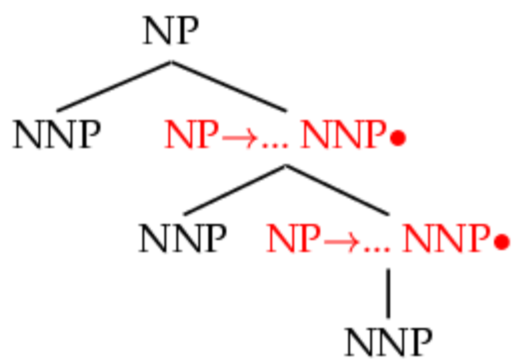




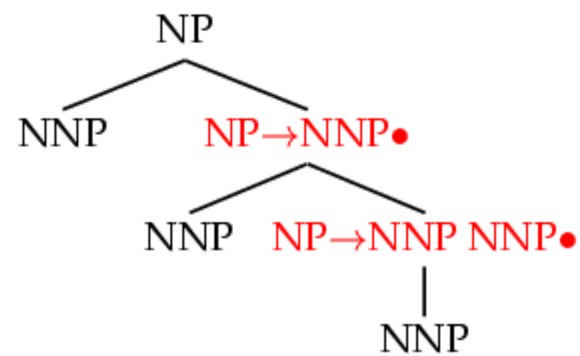
Horizontal Markovization

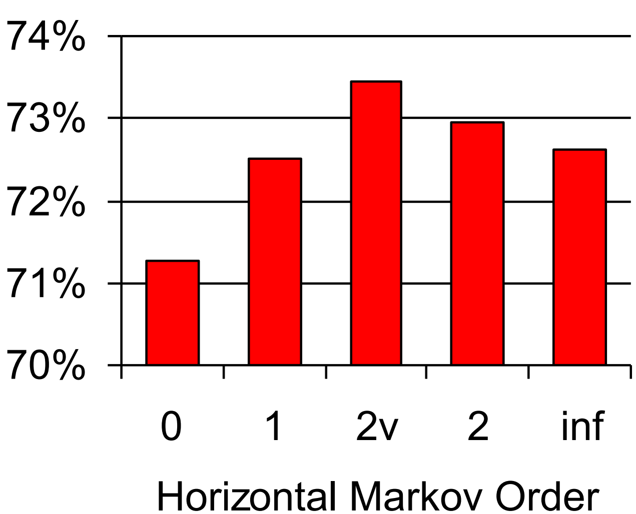


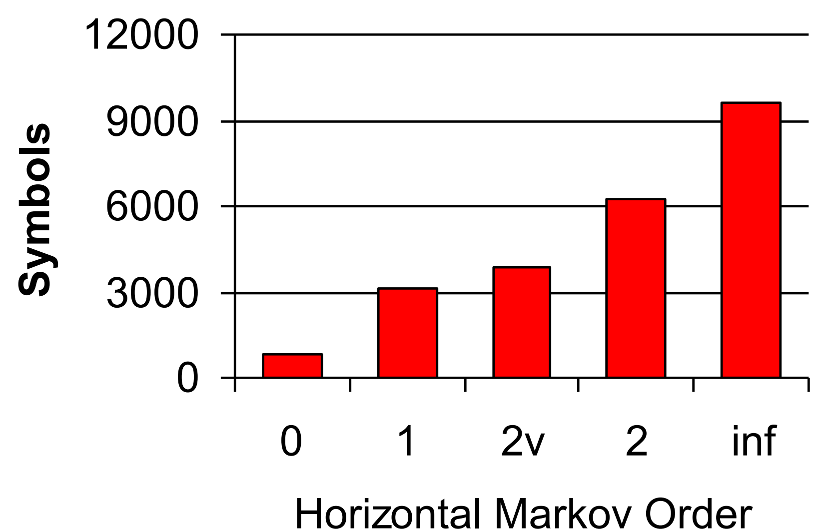
Order 1



Order ∞

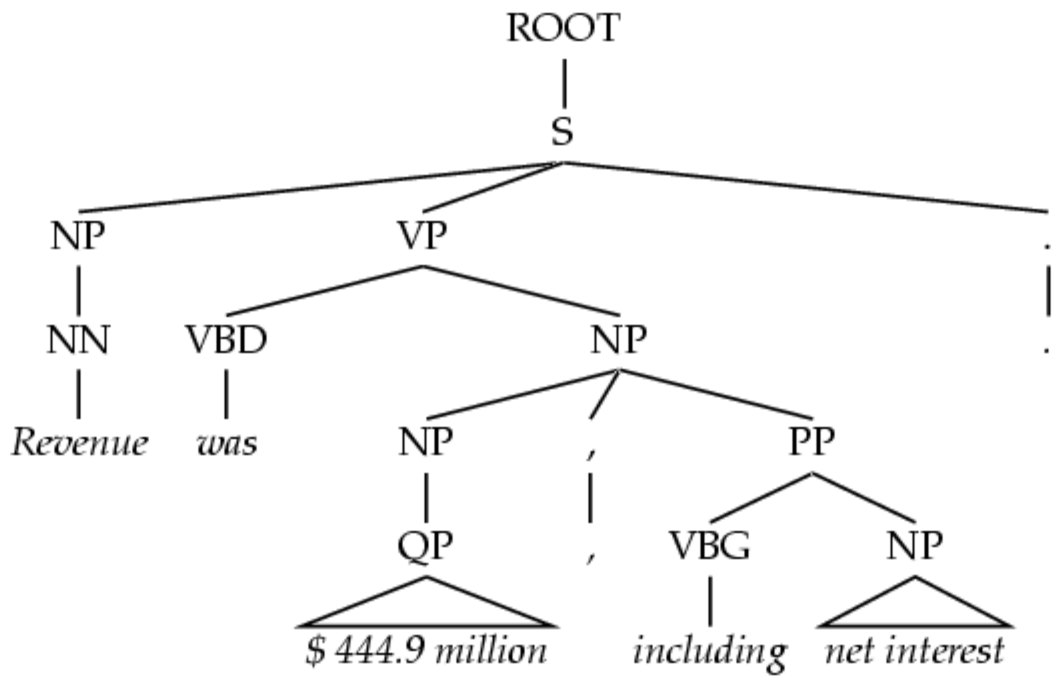






Unary Splits

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

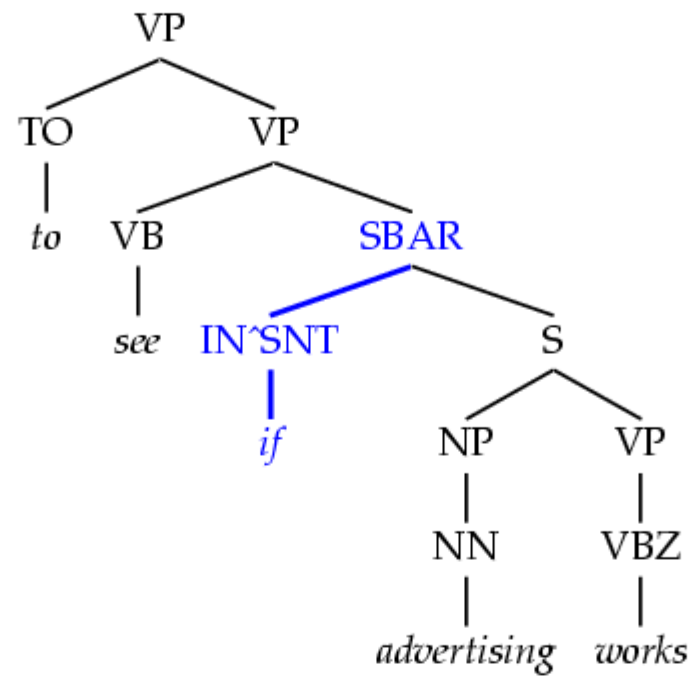


- Solution: Mark unary rewrite sites with -U

Annotation	F1	Size
Base	77.8	7.5K
UNARY	78.3	8.0K

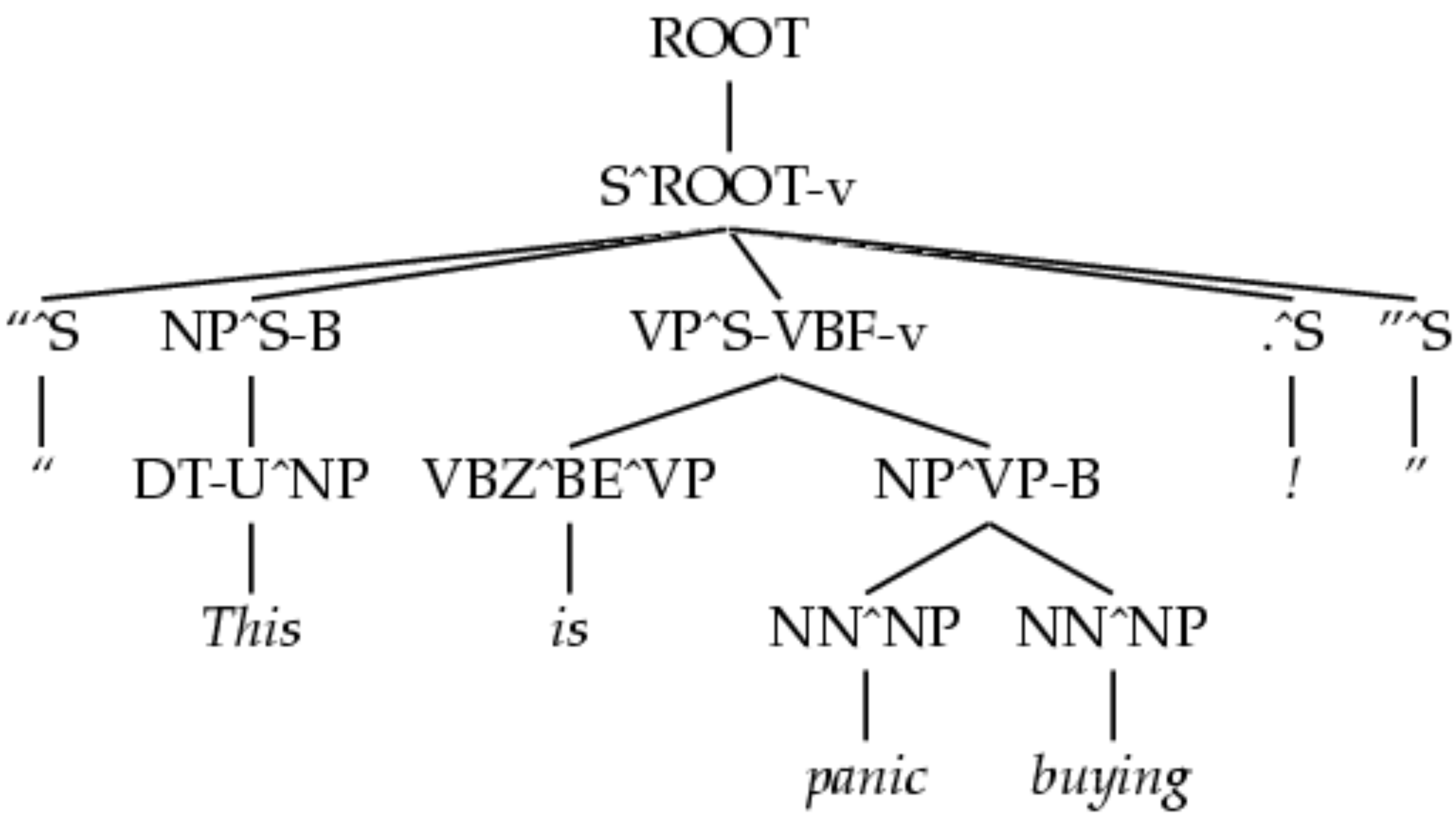
Tag Splits

- 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

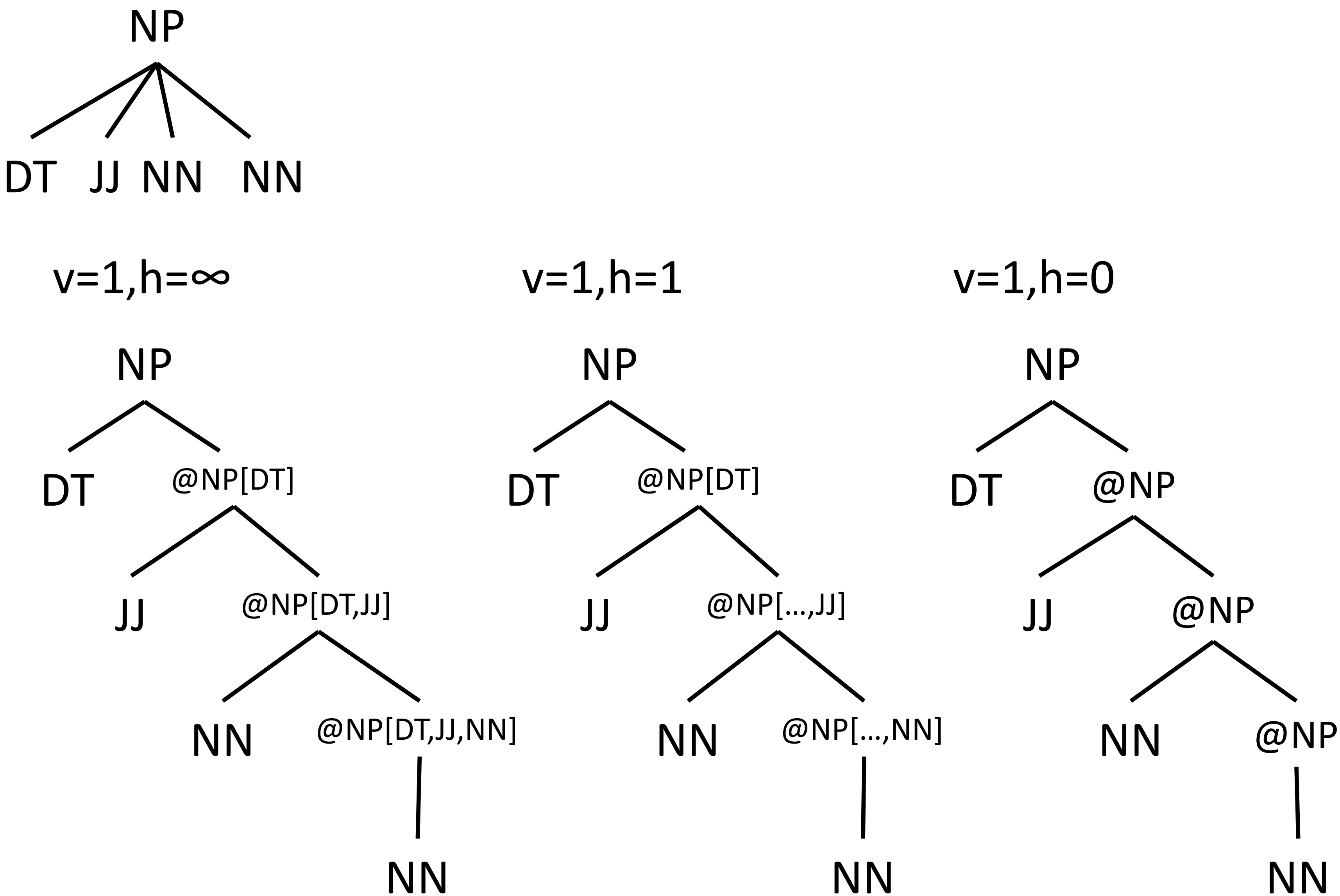


Some Test Set Results

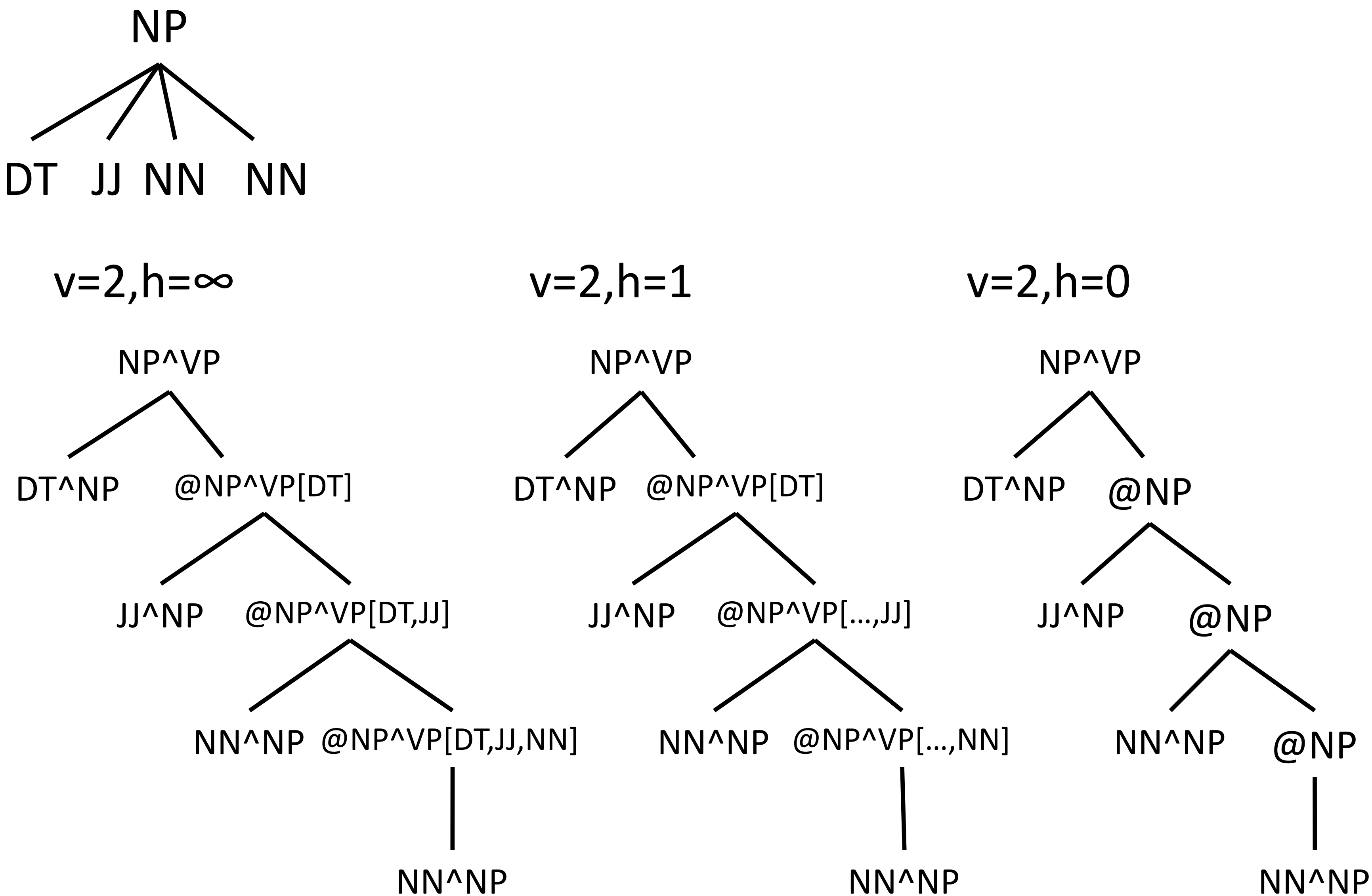
Parser	LP	LR	**F1**	CB	0 CB
Magerman 95	84.9	84.6	**84.7**	1.26	56.6
Collins 96	86.3	85.8	**86.0**	1.14	59.9
Unlexicalized	86.9	85.7	**86.3**	1.10	60.3
Charniak 97	87.4	87.5	**87.4**	1.00	62.1
Collins 99	88.7	88.6	**88.6**	0.90	67.1

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

Binarization / Markovization

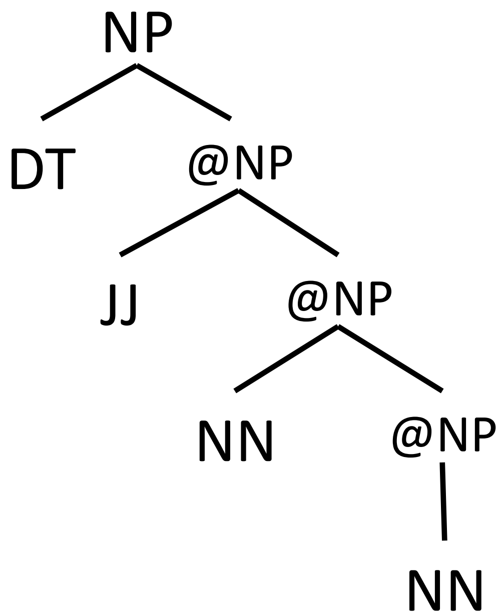


Binarization / Markovization

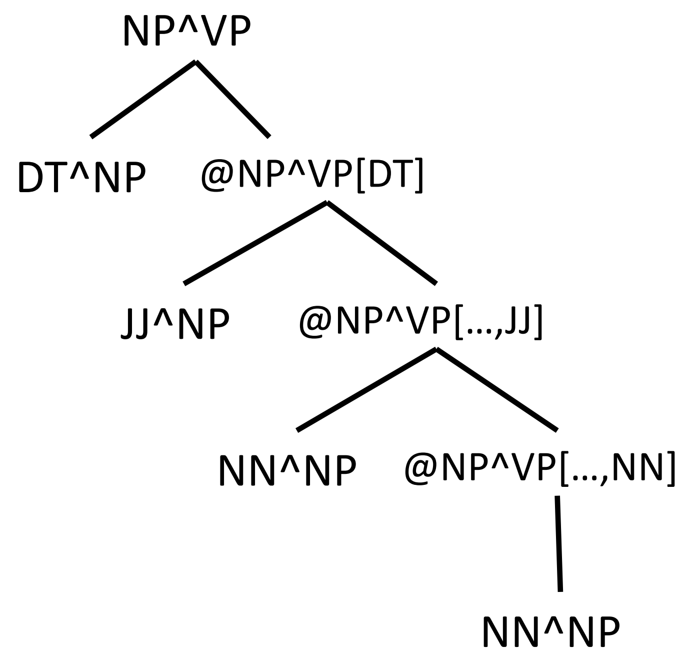


Grammar Projections

Coarse Grammar



Fine Grammar



NP → DT @NP

NP^VP → DT^NP @NP^VP[DT]

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

Grammar Projections

Coarse Symbols	Fine Symbols
NP	NP^VP NP^S
@NP	@NP^VP[DT] @NP^S[DT] @NP^VP[…,JJ] @NP^S[…,JJ]
DT	DT^NP

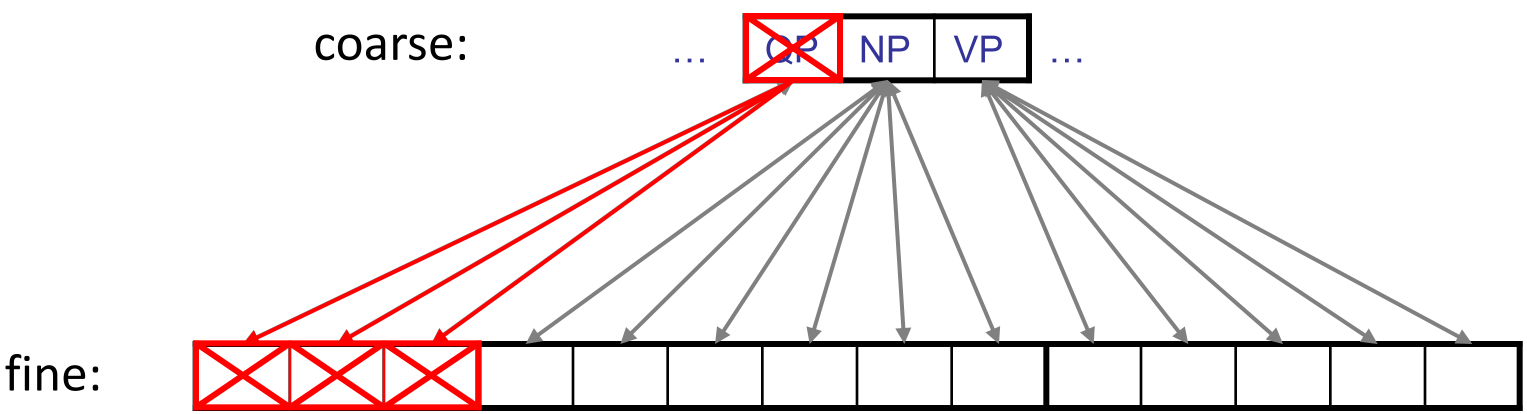
Efficient Parsing for Structural Annotation

Coarse-to-Fine Pruning

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

$$P(X|i,j,S) \quad < \quad \textit{threshold}$$

E.g. consider the span 5 to 12:

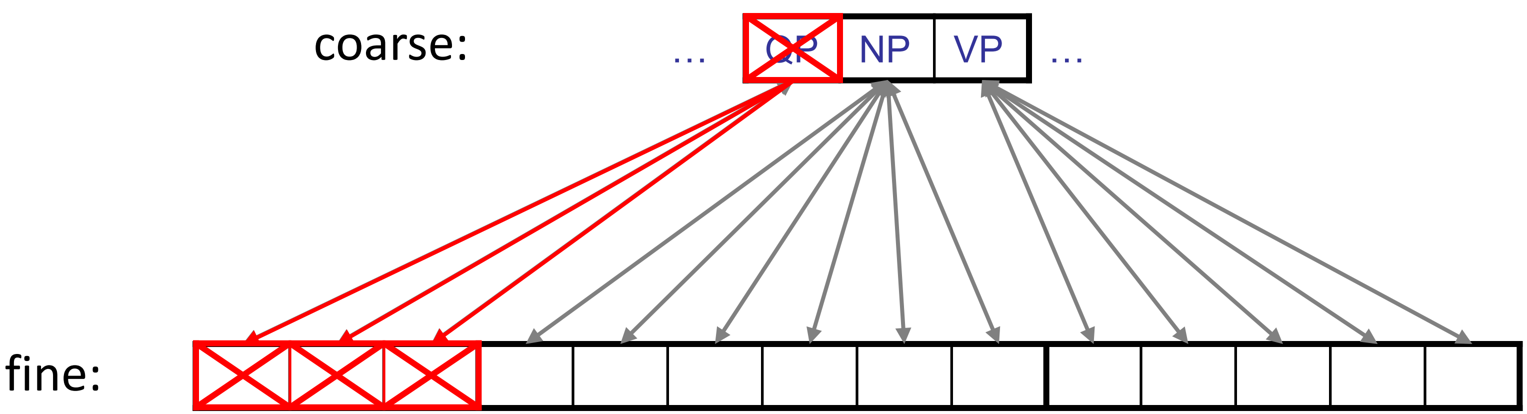


Coarse-to-Fine Pruning

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

$$\frac{\alpha(X,i,j)\cdot\beta(X,i,j)}{\alpha(\mathrm{root},0,n)} < \textit{threshold}$$

E.g. consider the span 5 to 12:



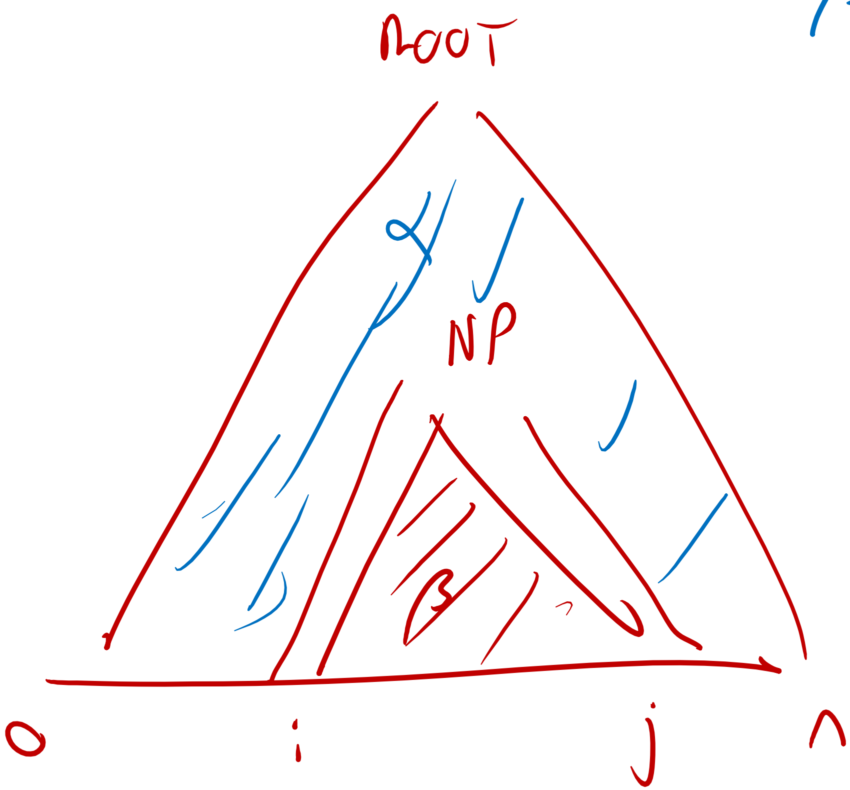
Computing Marginals

$$\alpha(X,i,j) = \sum_{X\to YZ} \sum_{k\in(i,j)} P(X \to YZ)\alpha(Y,i,k)\alpha(Z,k,j)$$

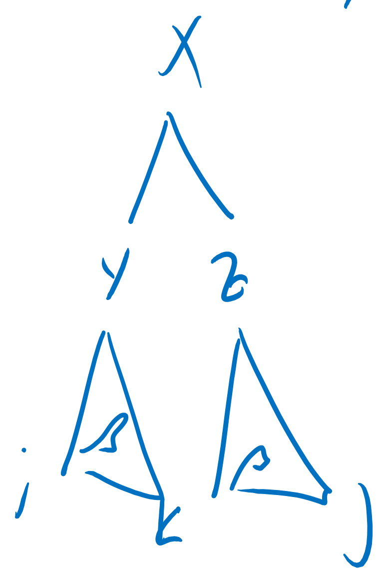
Computing Marginals

$$\beta(X,i,j) = \sum_{Y \to ZX} \sum_{k \in [0,i)} P(Y \to ZX)\beta(Y,k,j)\alpha(B,k,i)$$
$$+ \sum_{Y \to XZ} \sum_{k \in (j,n]} P(Y \to XZ)\beta(Y,i,k)\alpha(Z,j,k)$$

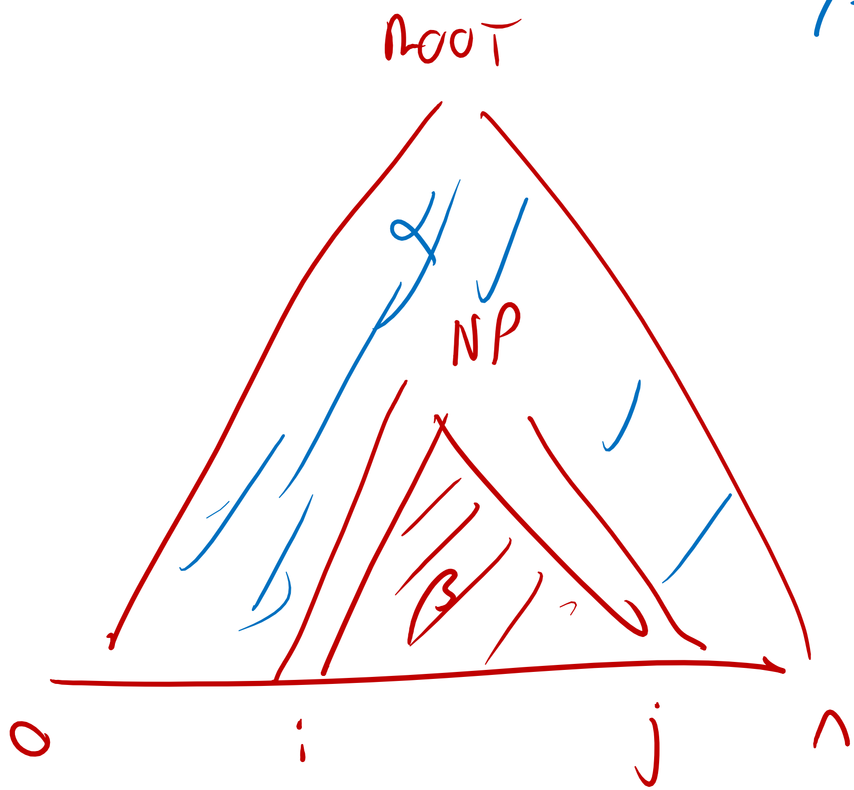
Computing (Max-)Marginals



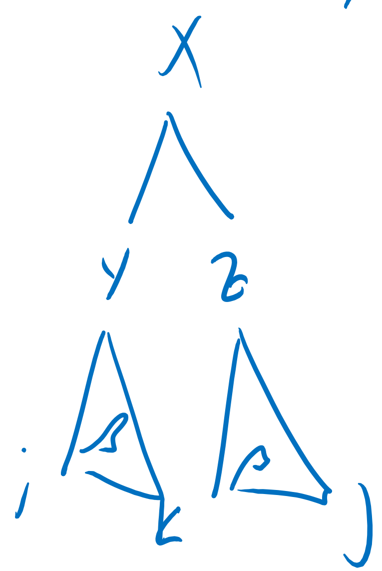
$$\beta(x, i, j) = \sum_{yz} \sum_{k} P(yz|x) \cdot \beta(y, i, k) \cdot \beta(z, k, j)$$



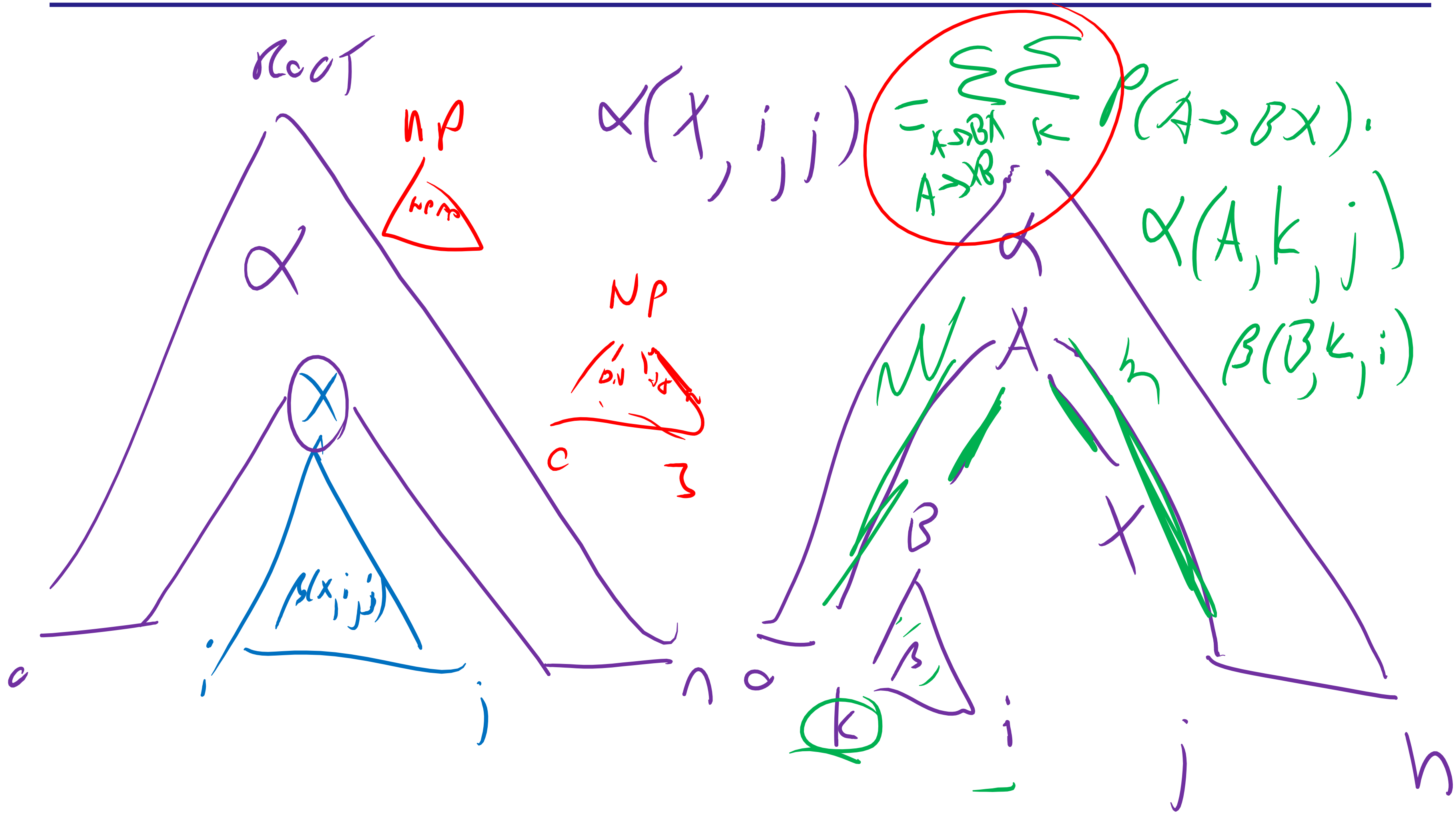
Computing (Max-)Marginals



$$\beta(X,i,j) = \sum_{YZ} \sum_{k} P(YZ|X) \cdot \beta(Y,i,k) \cdot \beta(Z,k,j)$$

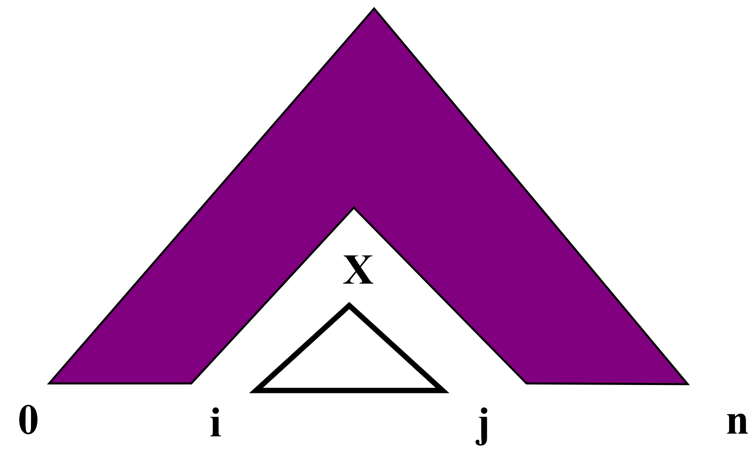


Inside and Outside Scores



Pruning with A*

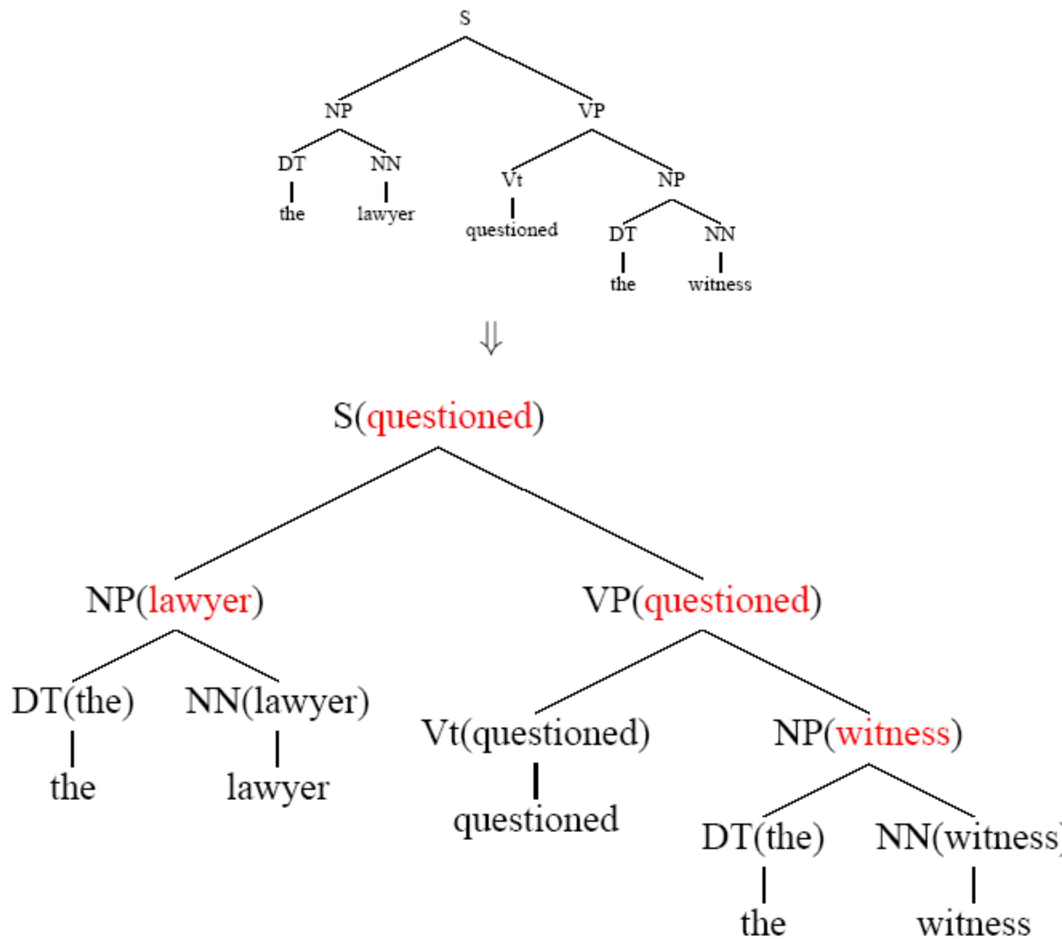
- You can also speed up the search without sacrificing optimality
- For agenda-based parsers:
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A* heuristic, no loss of optimiality [Klein and Manning 03]



Efficient Parsing for Lexical Grammars

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



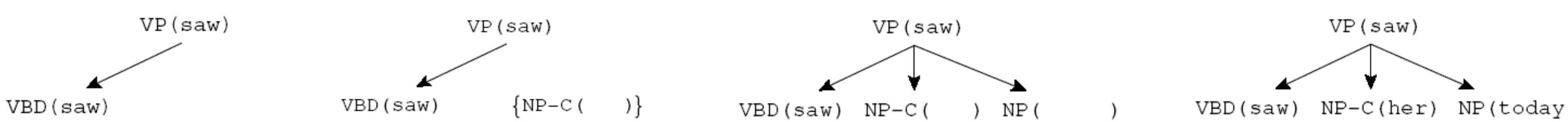
Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like

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

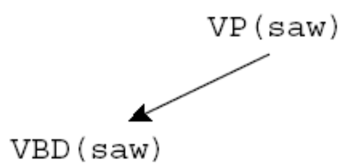
- Never going to get these atomically off of a treebank

- Solution: break up derivation into smaller steps

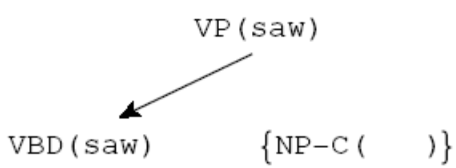


Lexical Derivation Steps

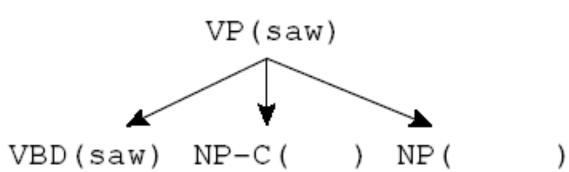
- A derivation of a local tree [Collins 99]



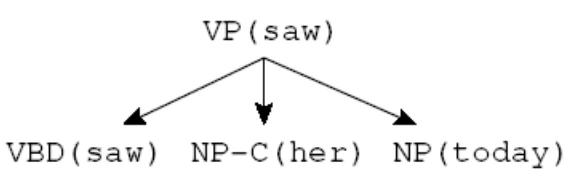
Choose a head tag and word



Choose a complement bag



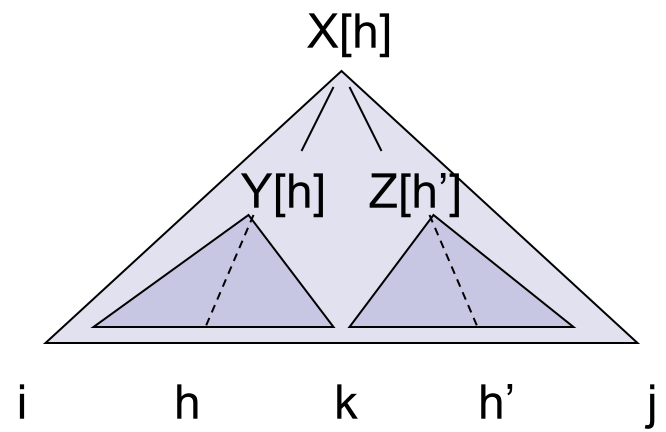
Generate children (incl. adjuncts)



Recursively derive children

Lexicalized CKY

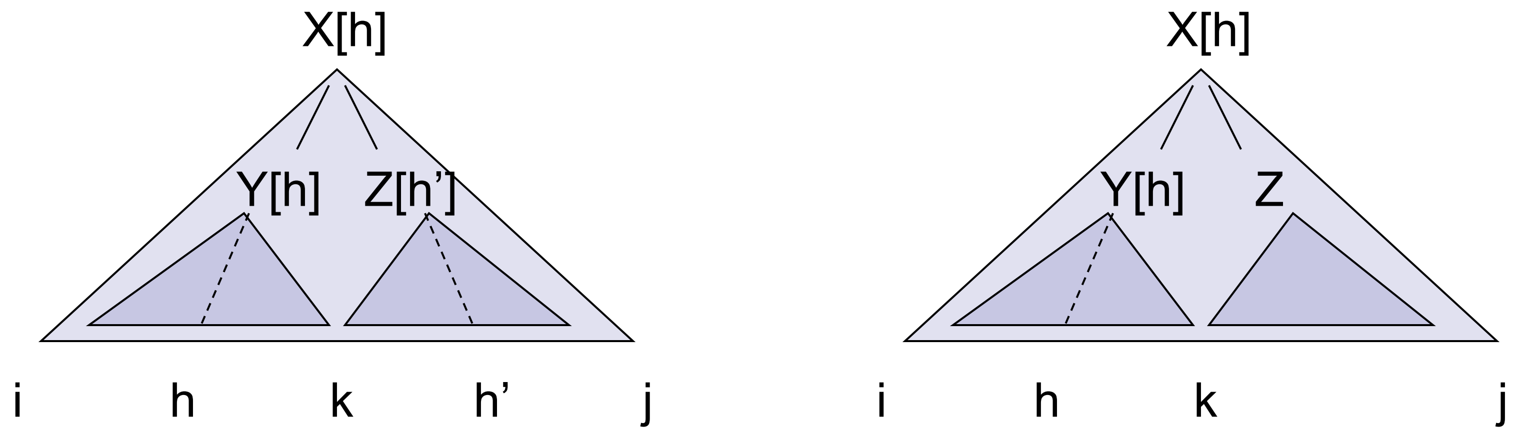
```
                (VP->VBD...NP •)[saw]
               /                     \
(VP->VBD •)[saw]                      NP[her]
```




```
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)
```

Quartic Parsing

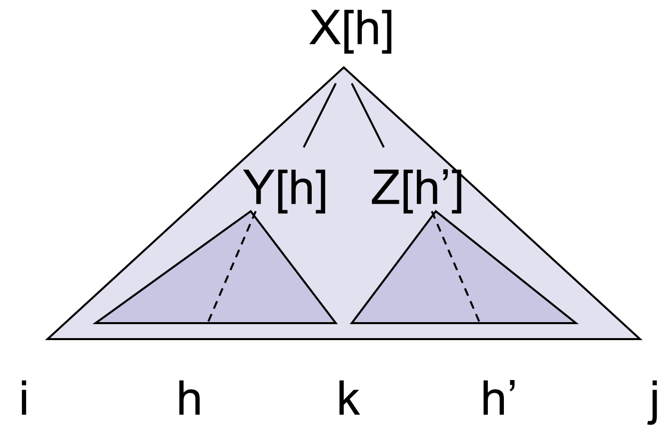
- Turns out, you can do (a little) better [Eisner 99]



- 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!

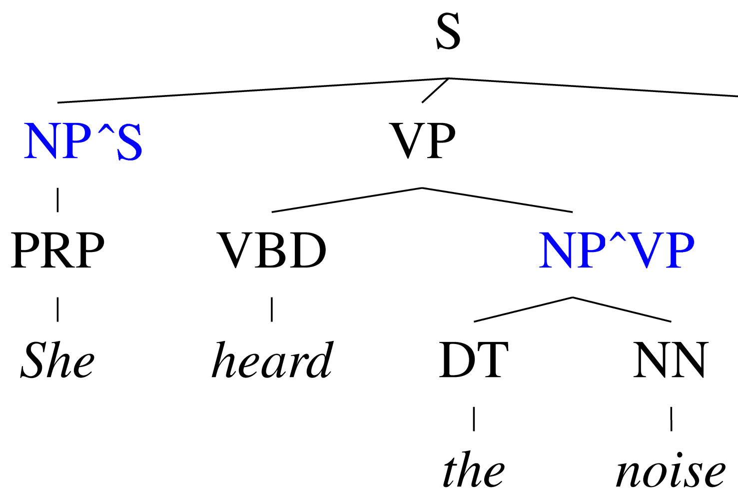
- Charniak et al 06: can use more passes
- Petrov et al 07: can use many more passes

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)

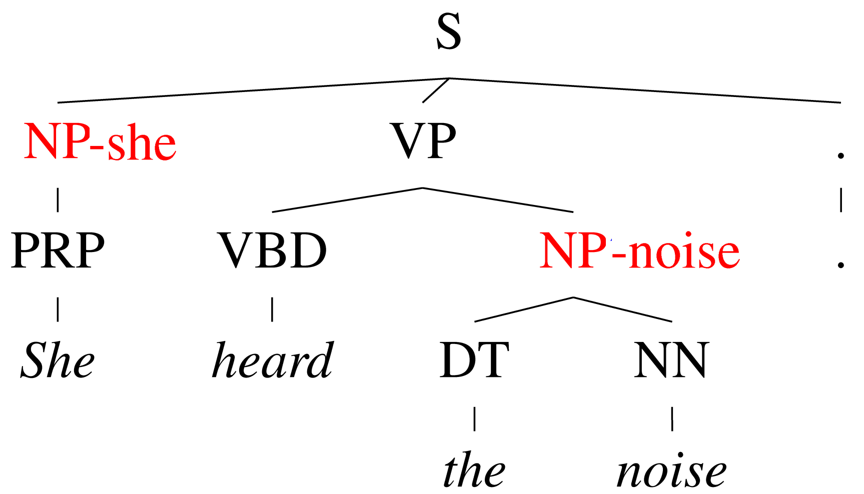
Latent Variable PCFGs

The Game of Designing a Grammar



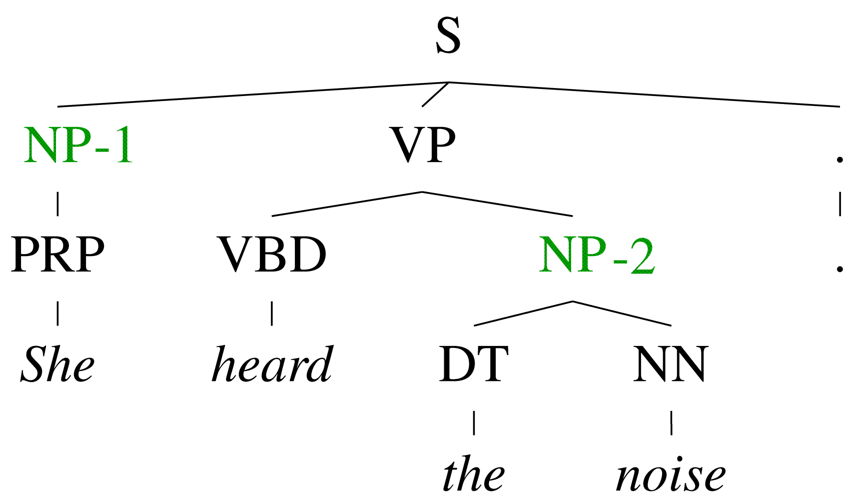
- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson '98]

The Game of Designing a Grammar



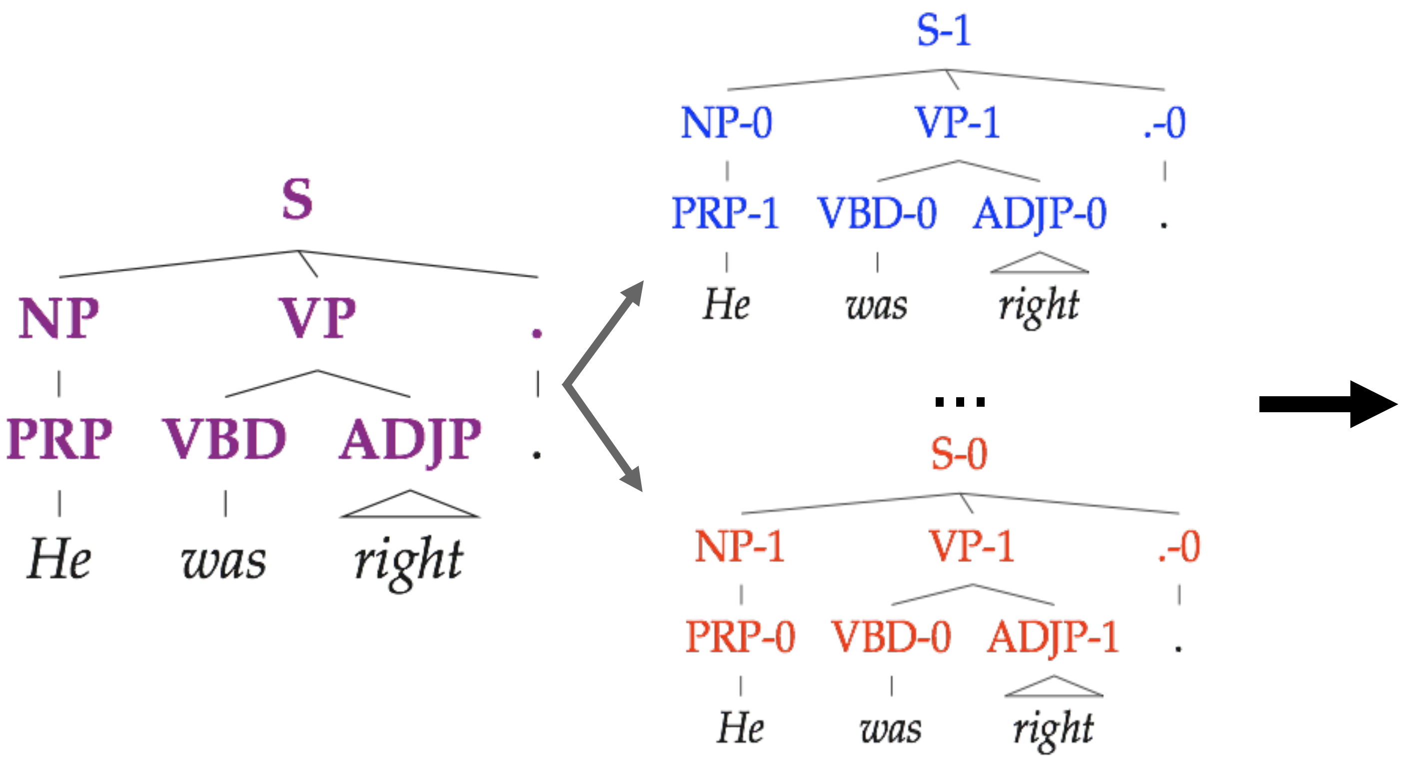
- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson ’98]
  - Head lexicalization [Collins ’99, Charniak ’00]

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



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

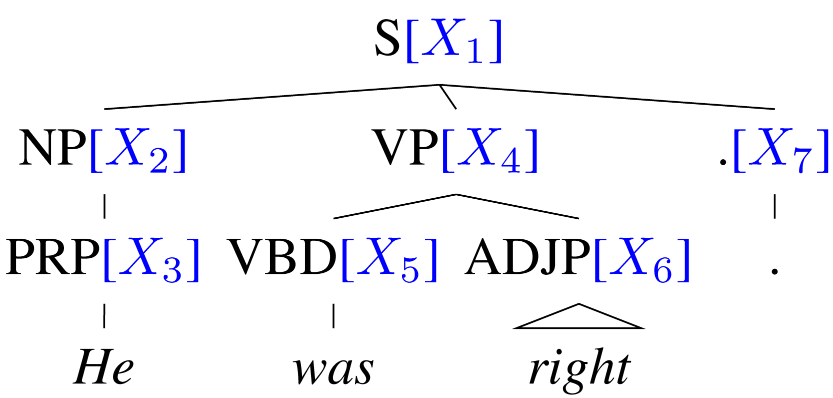
Derivations $t : T$

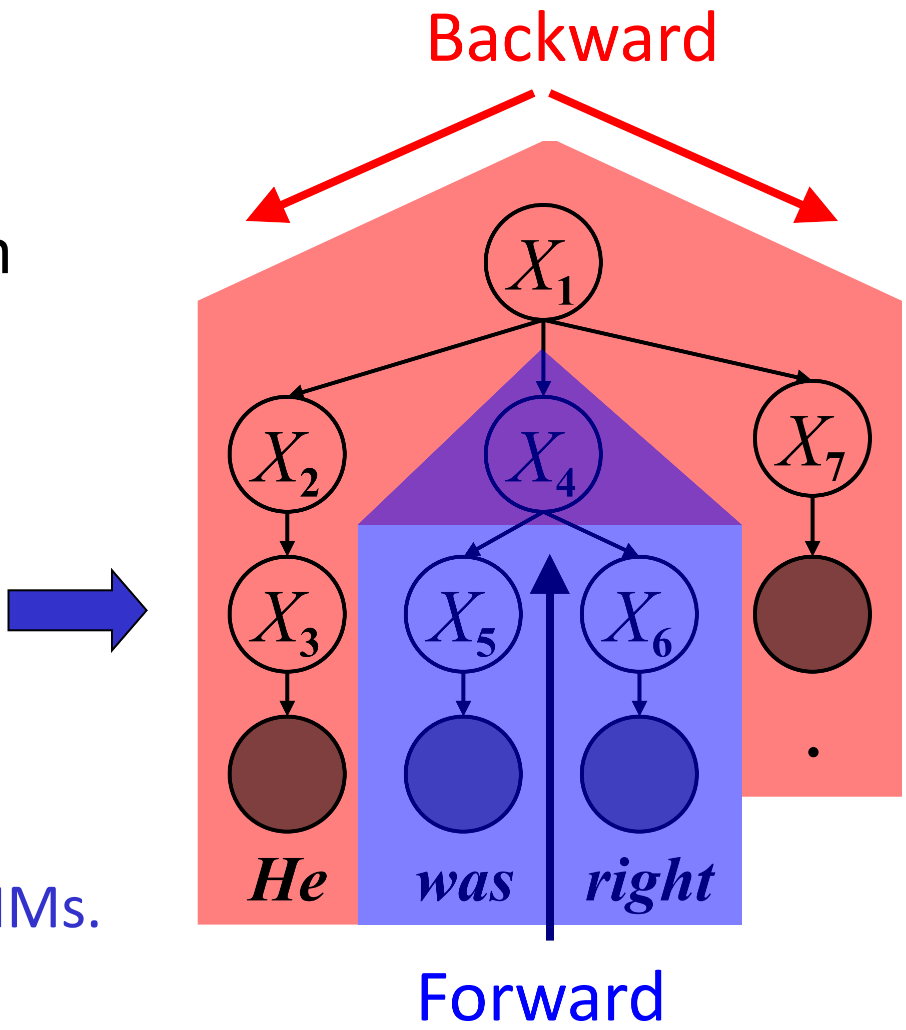
Parameters θ

Learning Latent Annotations

EM algorithm:

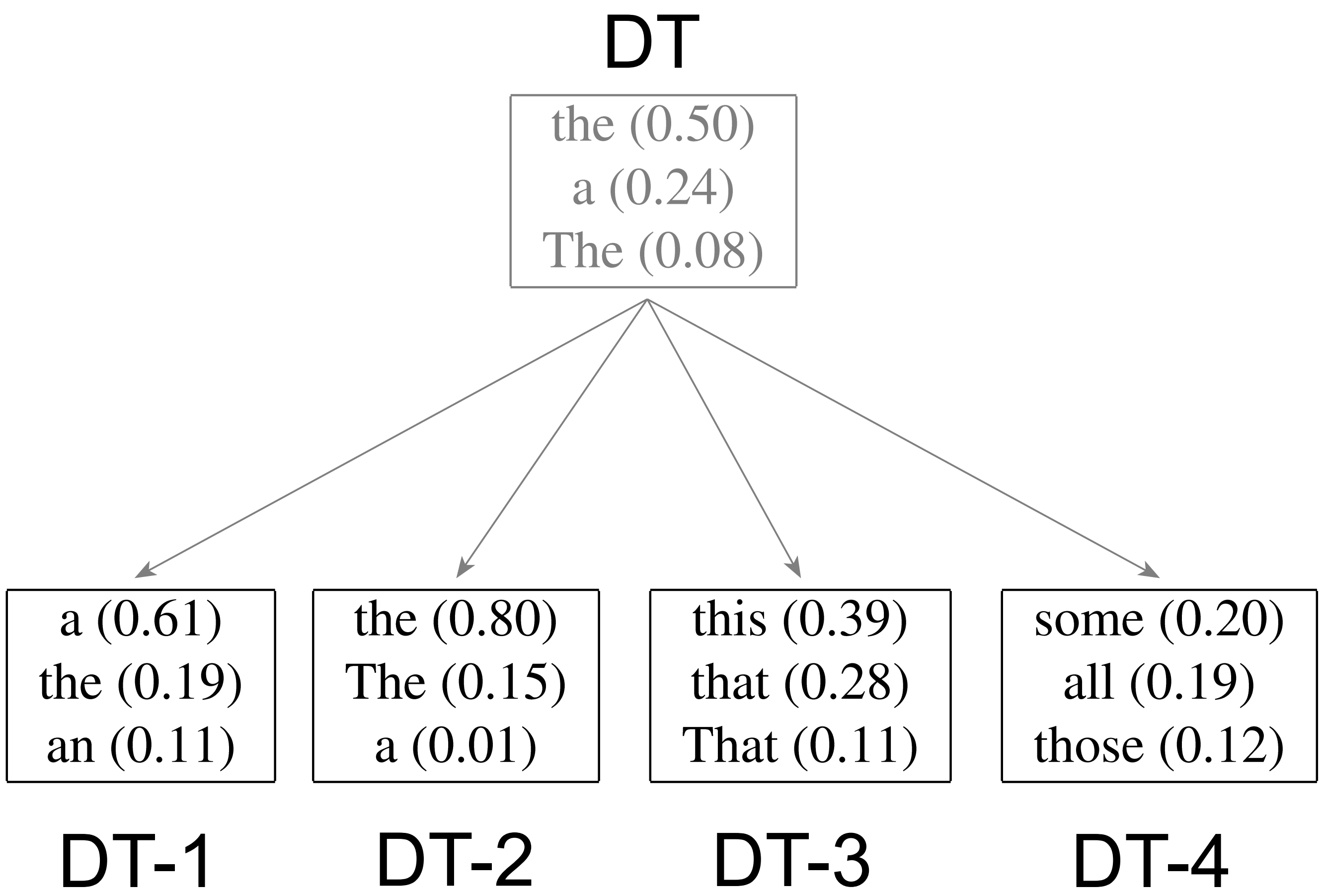
- Brackets are known
- Base categories are known
- Only induce subcategories



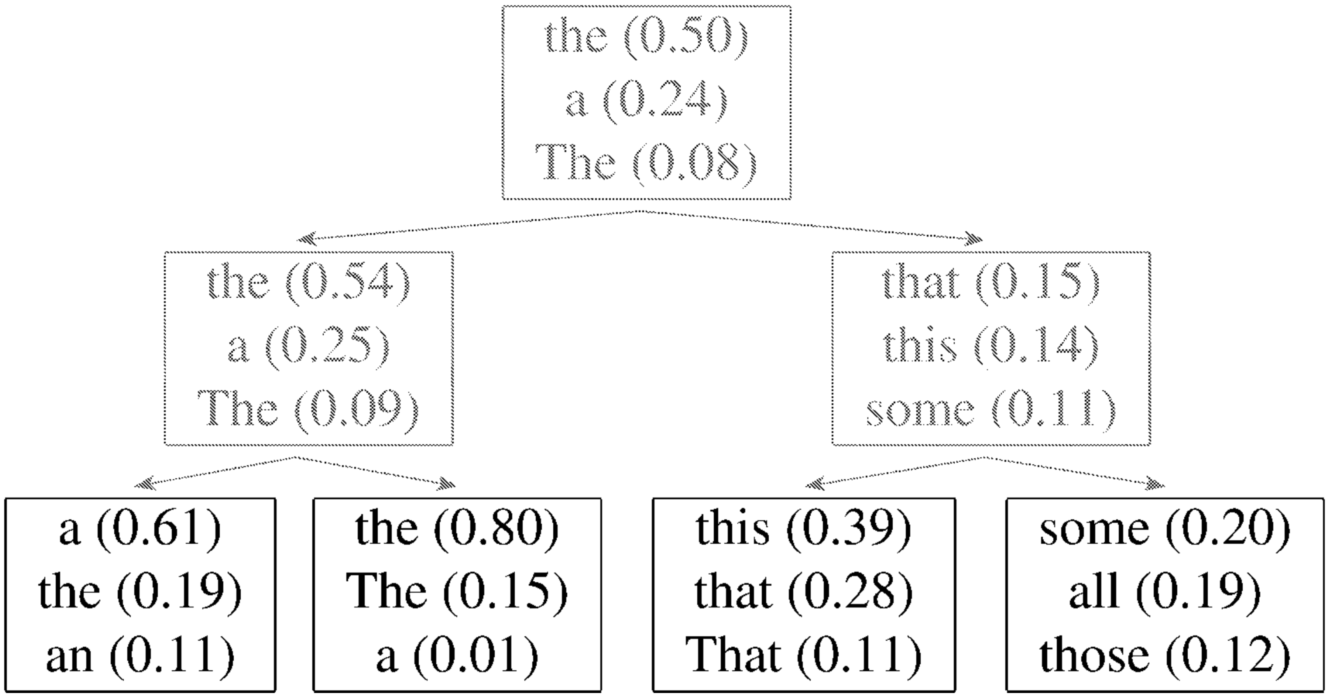


Just like Forward-Backward for HMMs.

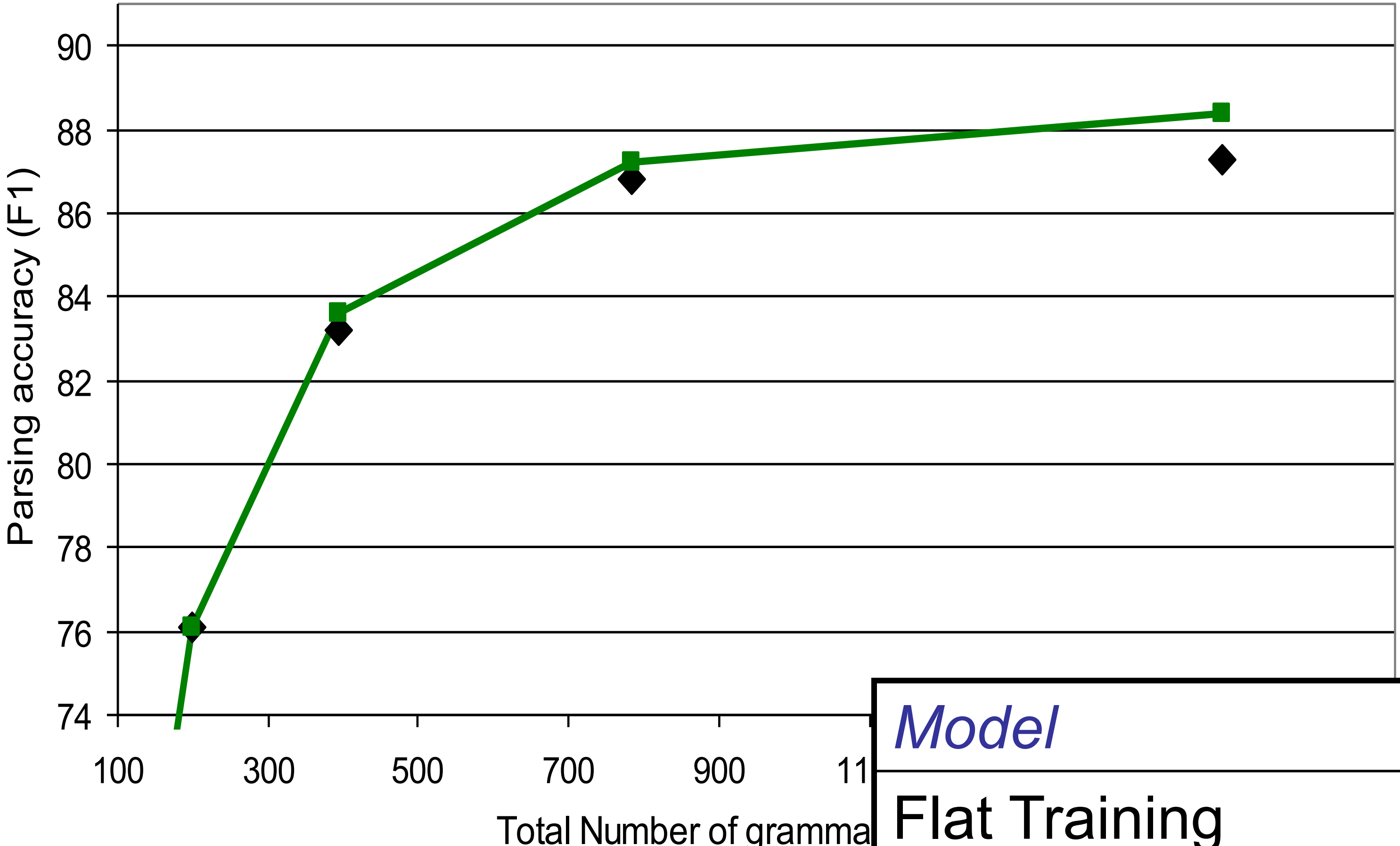
Refinement of the DT tag



Hierarchical refinement



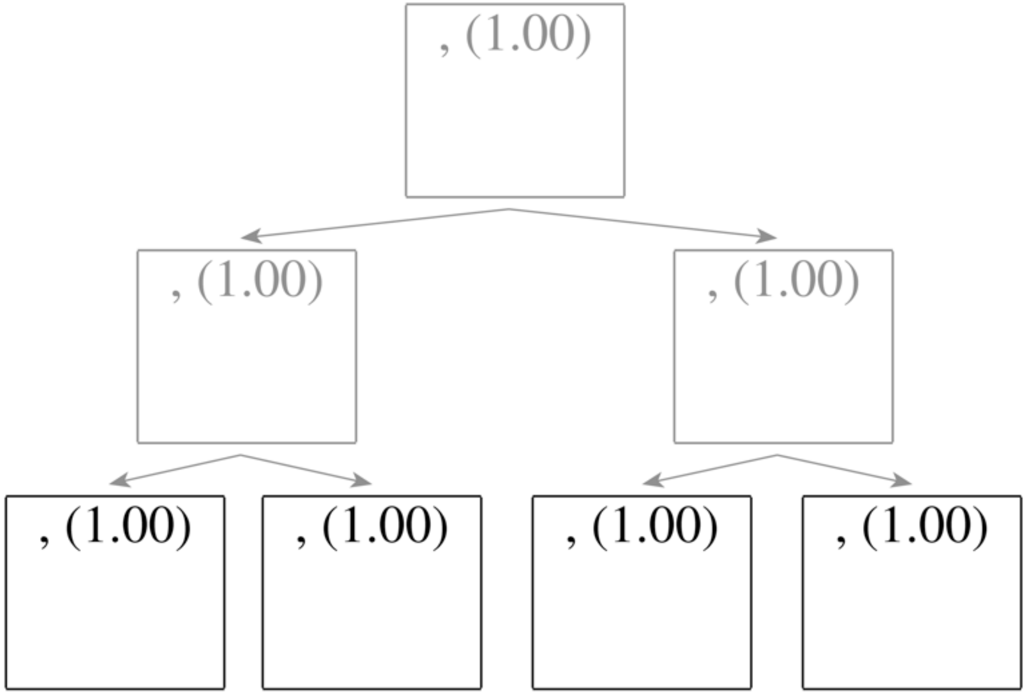
Hierarchical Estimation Results



Model	*F1*
Flat Training	87.3
Hierarchical Training	88.4

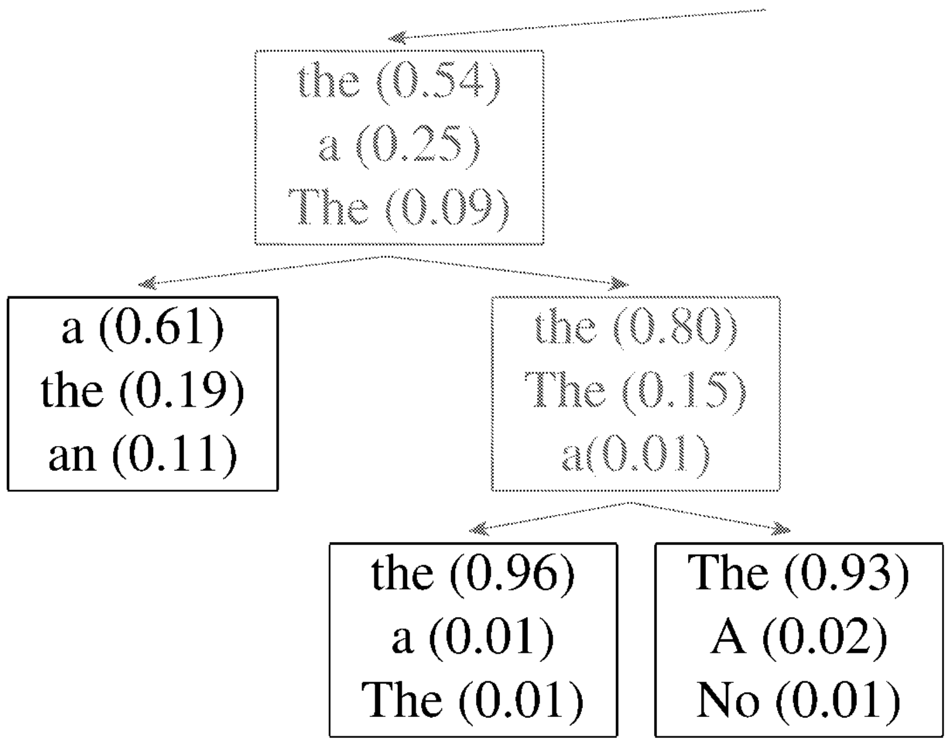
Refinement of the , tag

- Splitting all categories equally is wasteful:

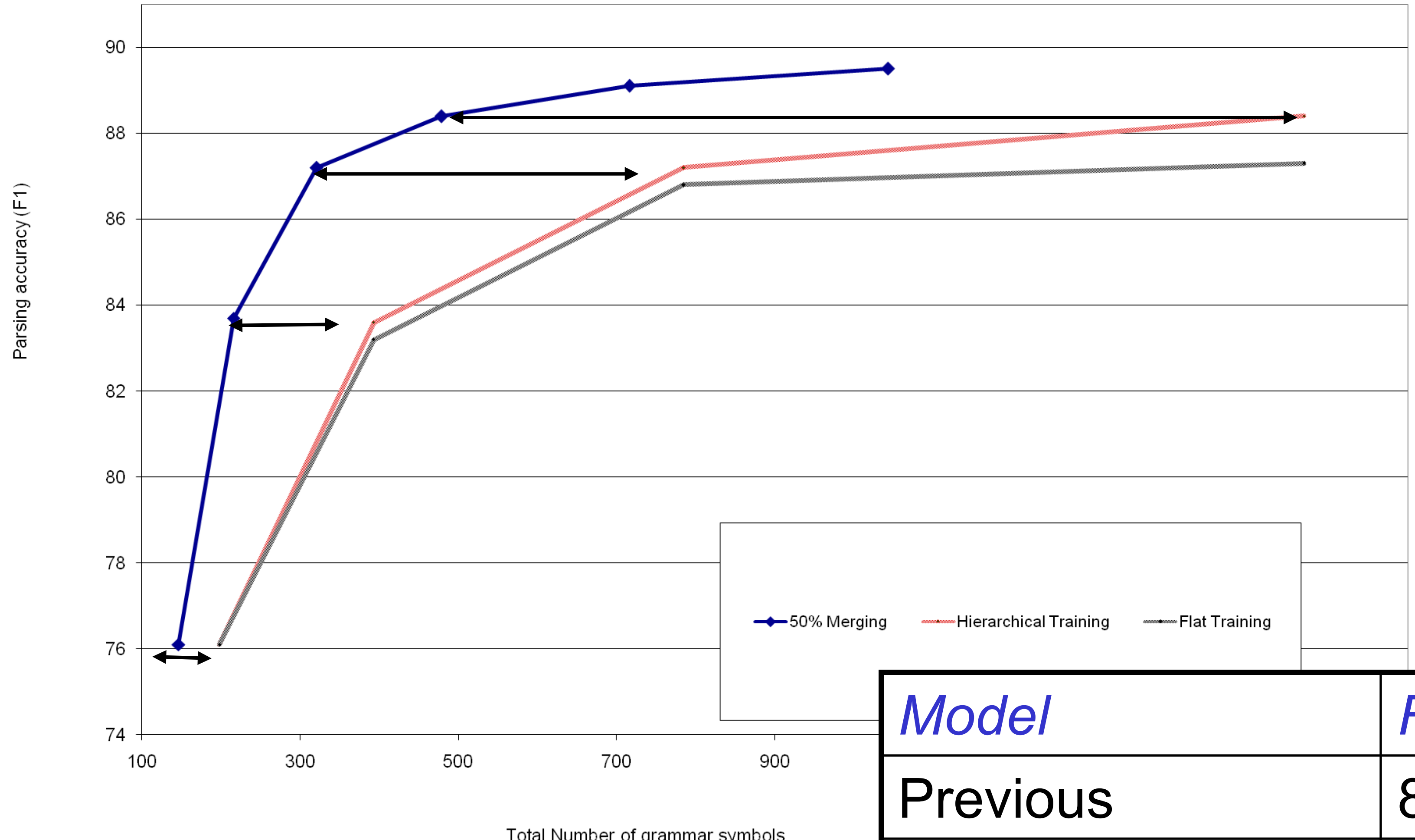


Adaptive Splitting

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

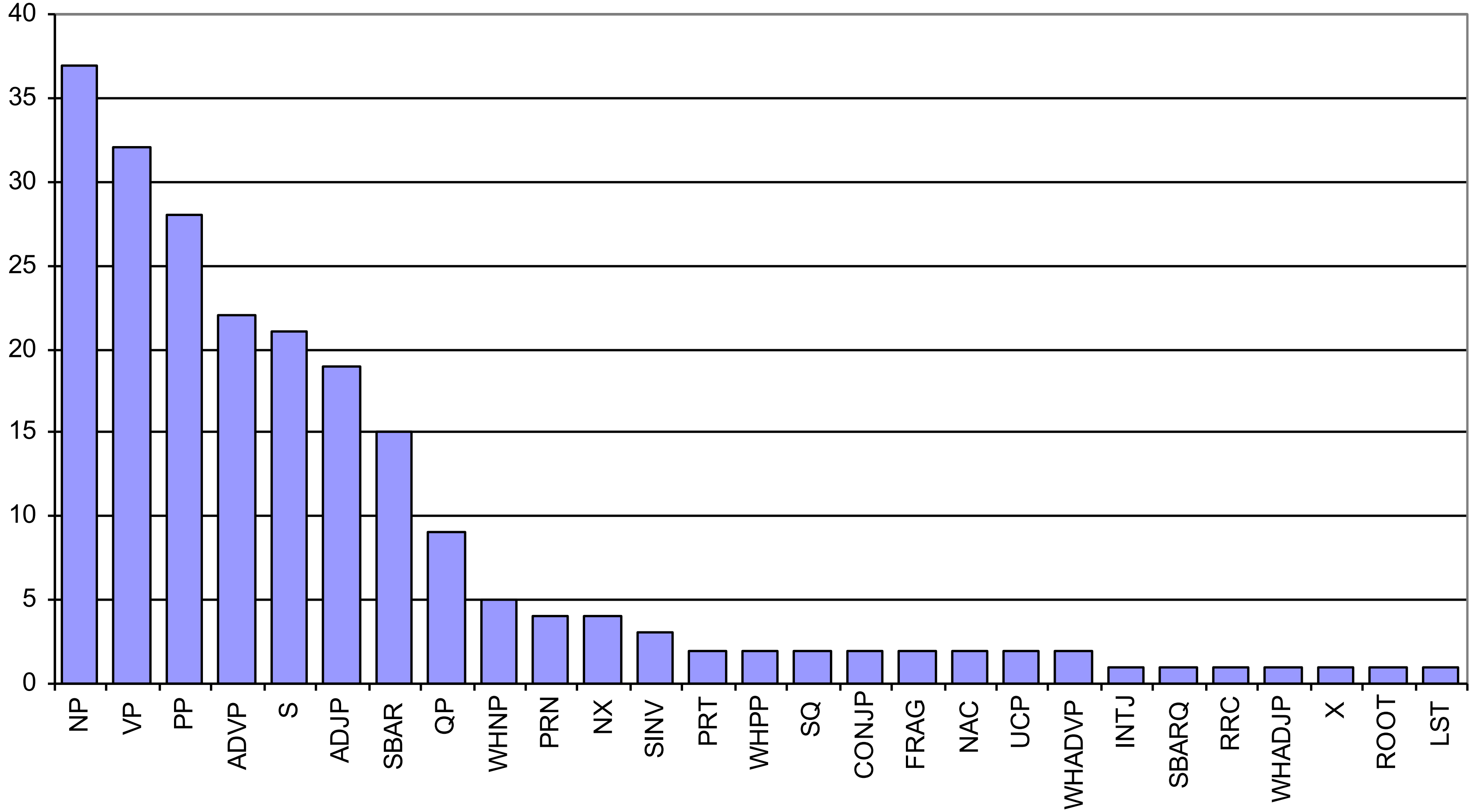


Adaptive Splitting Results

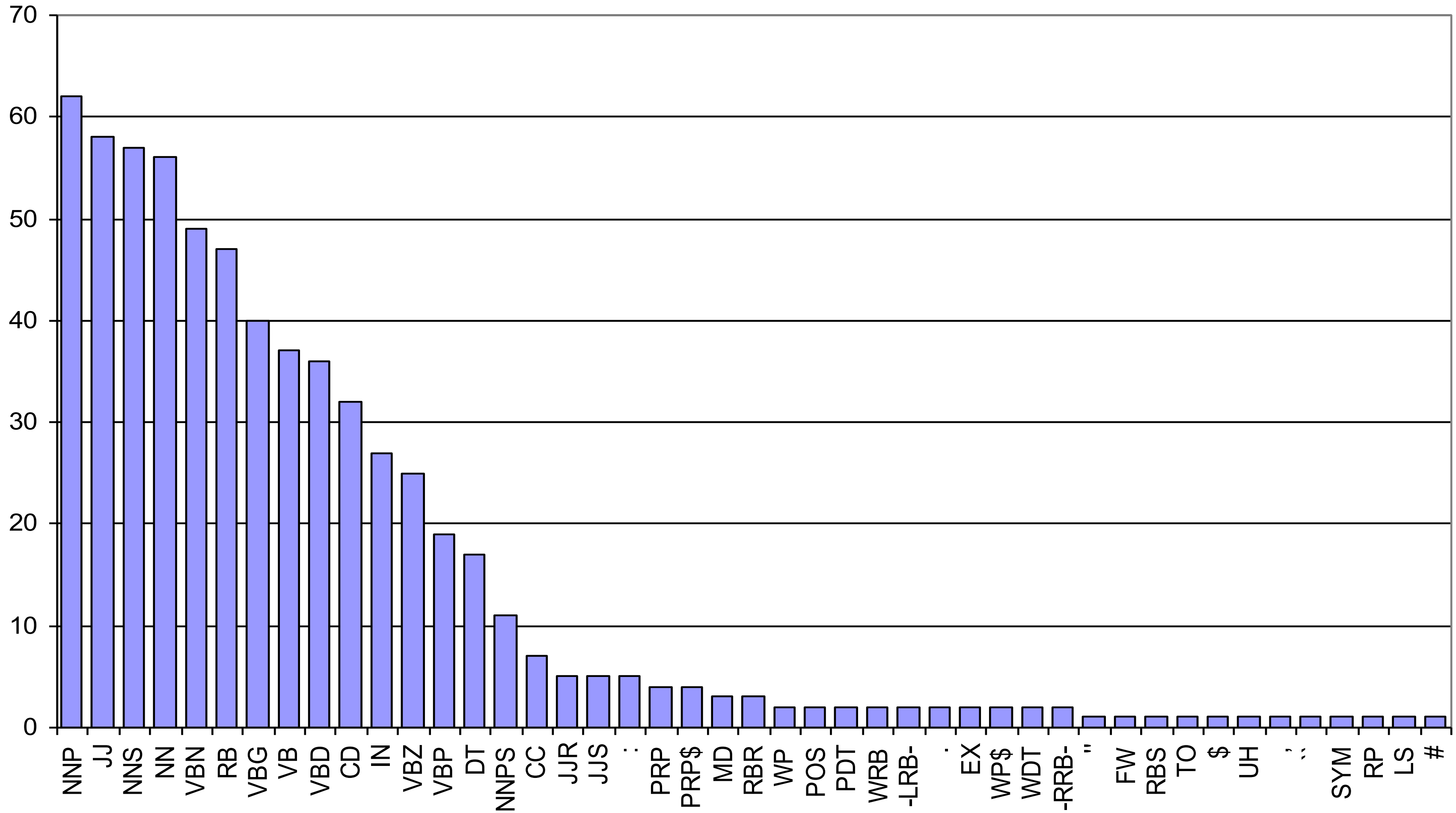


Model	*F1*
Previous	88.4
With 50% Merging	89.5

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)

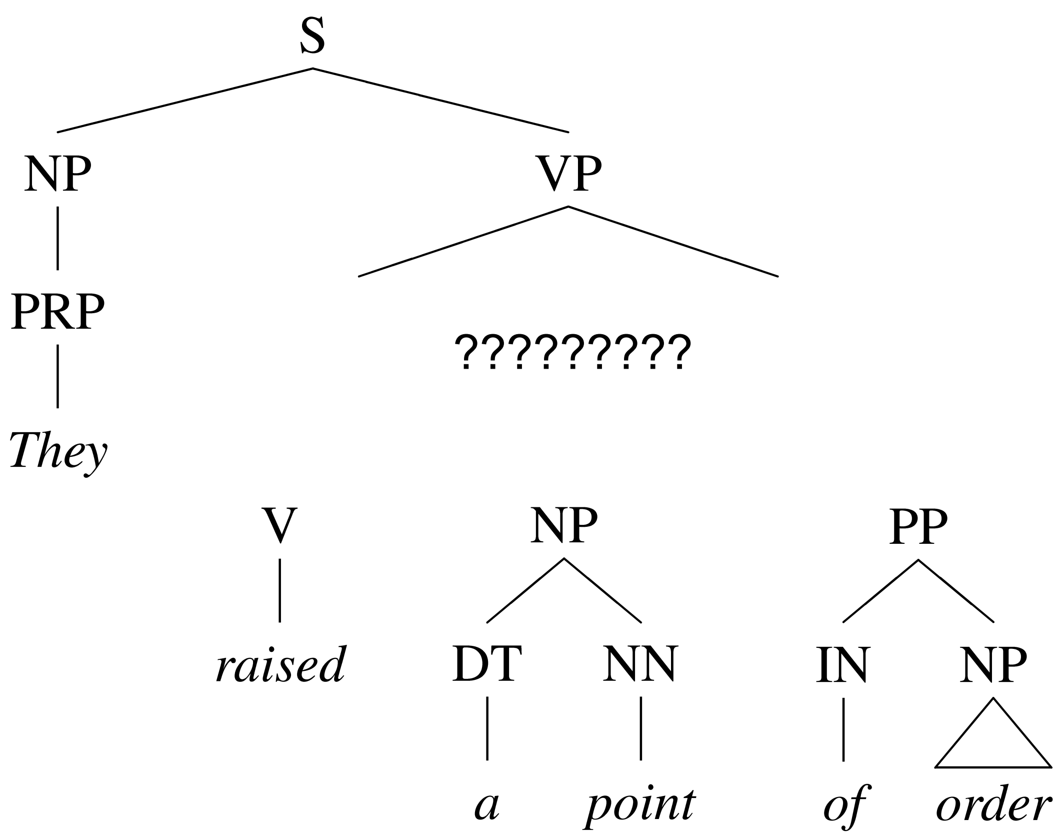
		≤ 40 words F1	all F1
ENG	Charniak&Johnson ‘05 (generative)	90.1	89.6
	Split / Merge	**90.6**	**90.1**
GER	Dubey ‘05	76.3	-
	Split / Merge	**80.8**	**80.1**
CHN	Chiang et al. ‘02	80.0	76.6
	Split / Merge	**86.3**	**83.4**

Still higher numbers from reranking / self-training methods

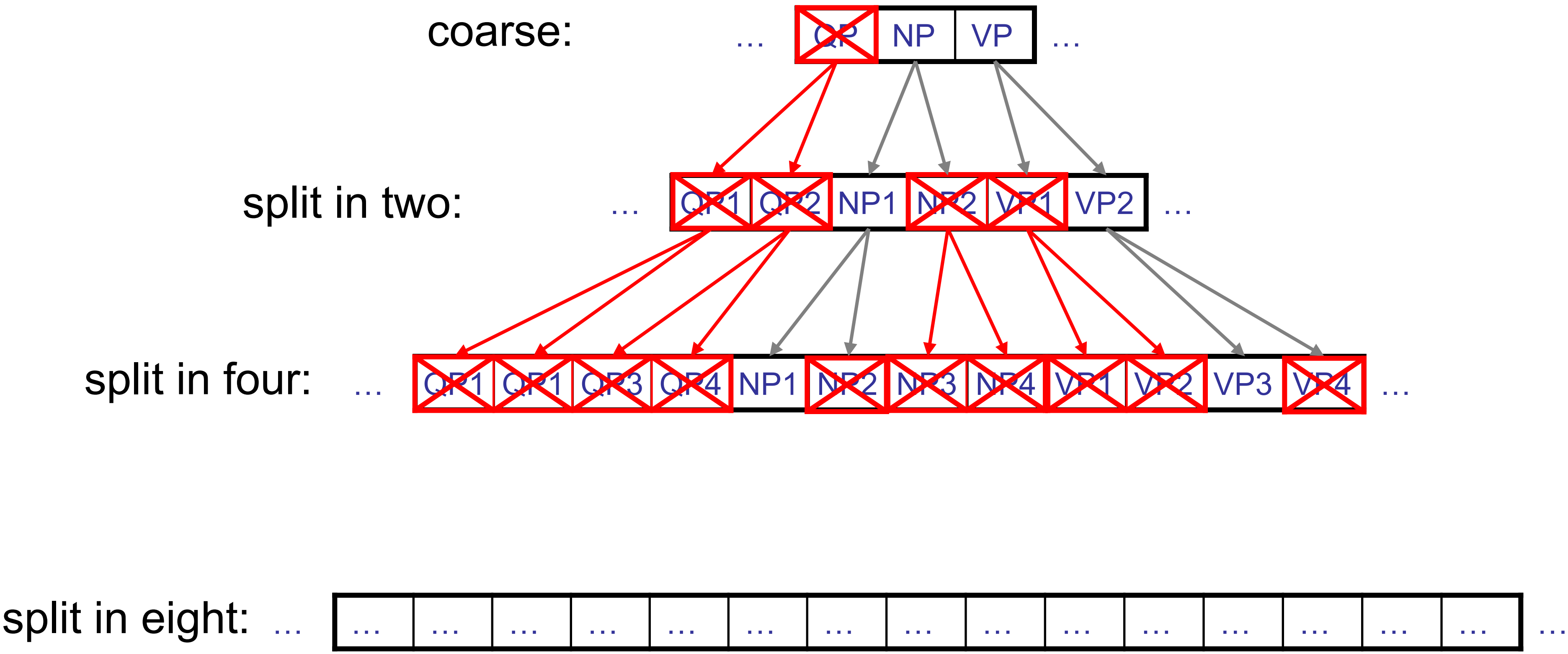
Efficient Parsing for Hierarchical Grammars

Coarse-to-Fine Inference

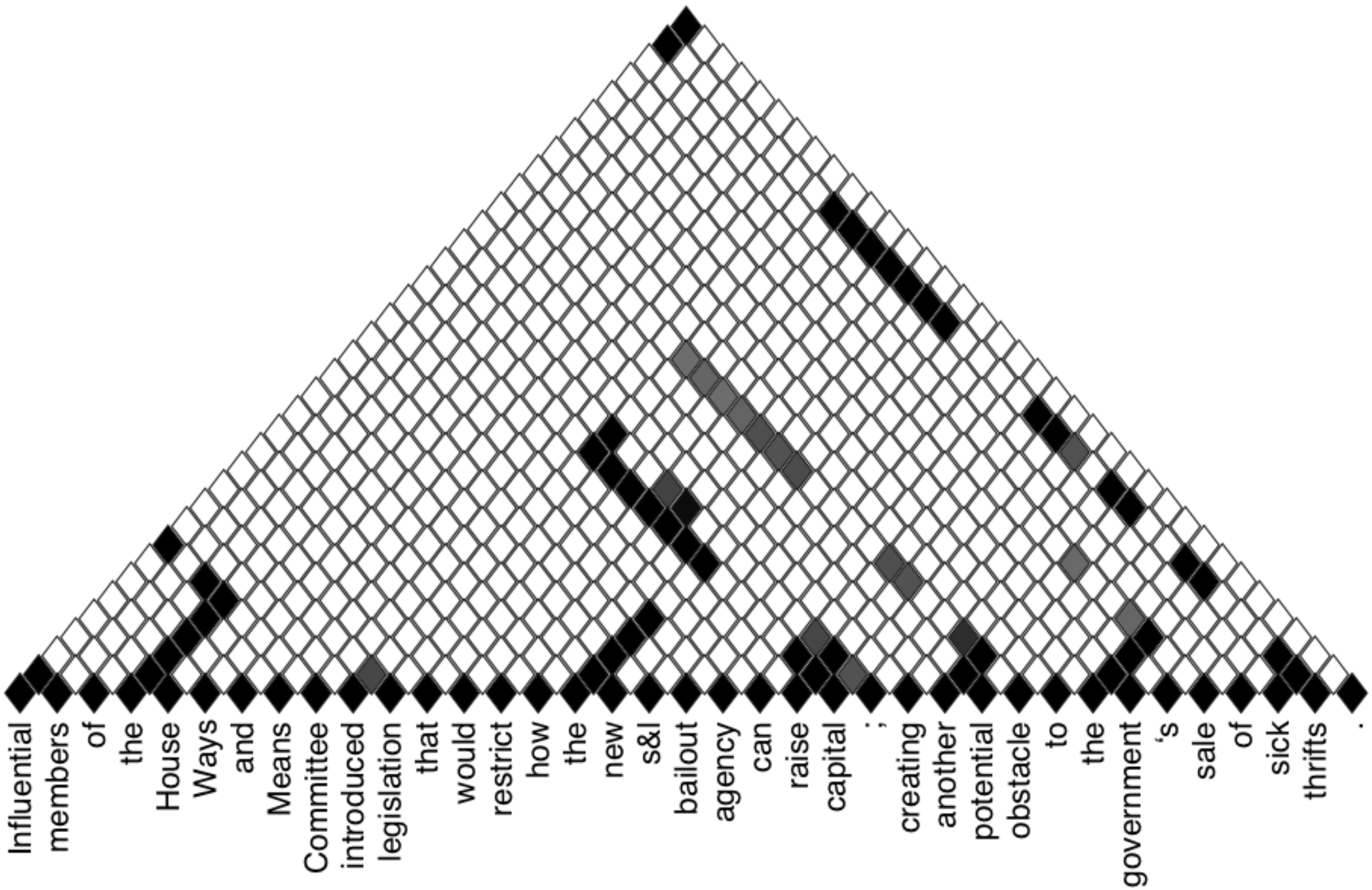
- Example: PP attachment



Hierarchical Pruning



Bracket Posteriors



1621 min

111 min

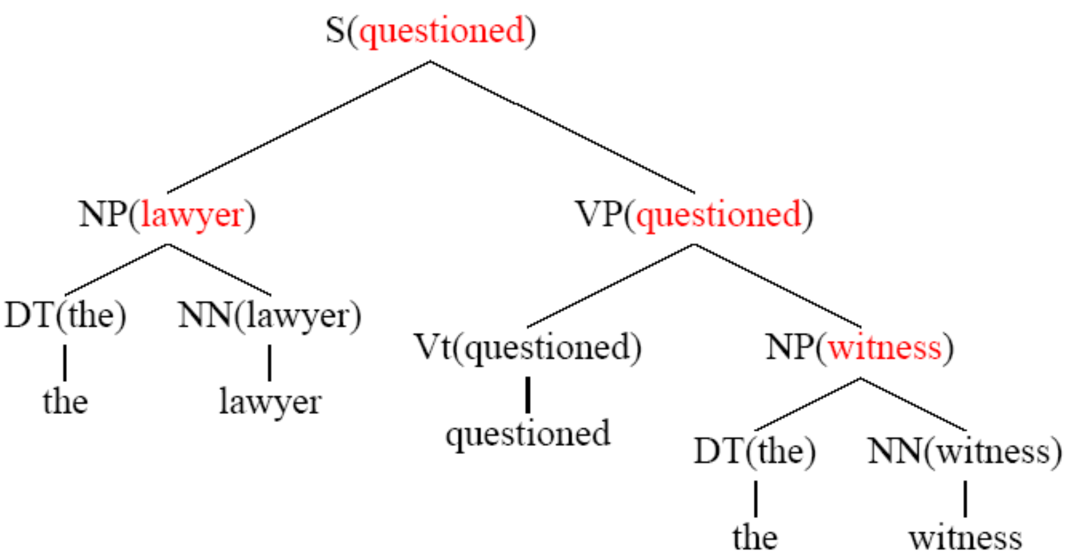
35 min

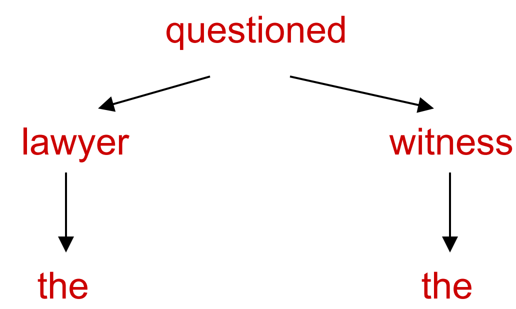
15 min
(no search error)

Other Syntactic Models

Dependency Parsing

- Lexicalized parsers can be seen as producing *dependency trees*

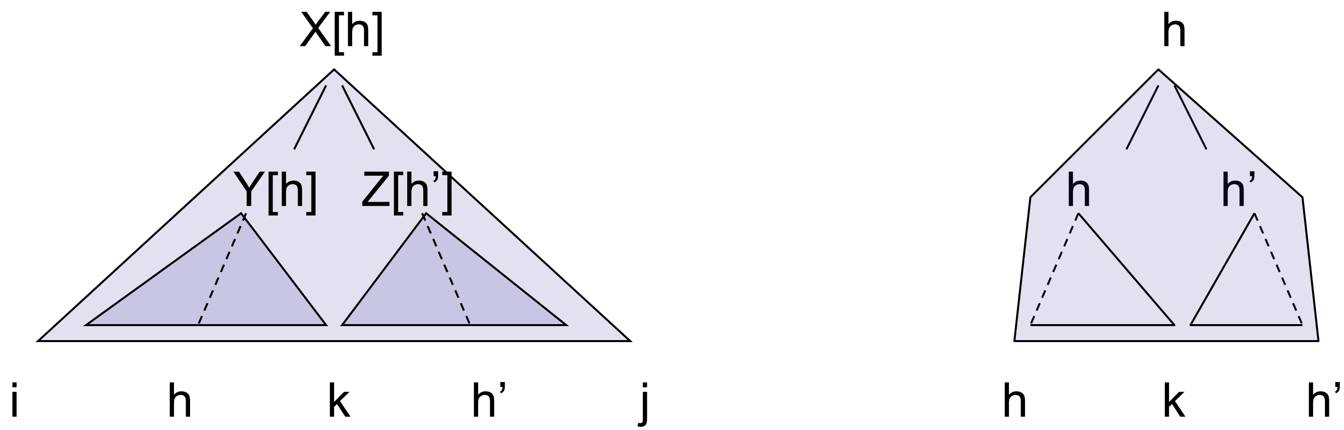




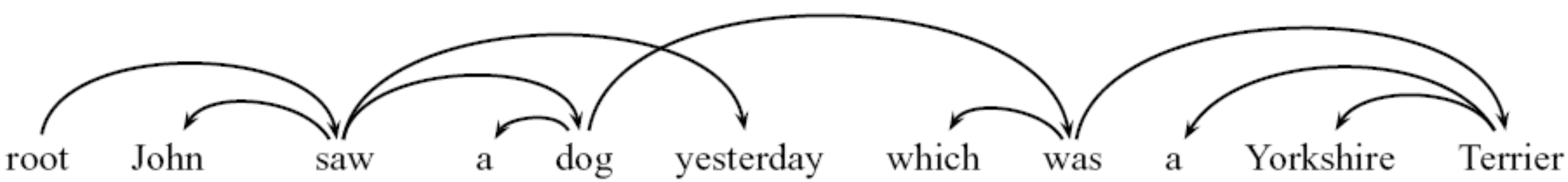
- Each local binary tree corresponds to an attachment in the dependency graph

Dependency Parsing

- Pure dependency parsing is only cubic [Eisner 99]

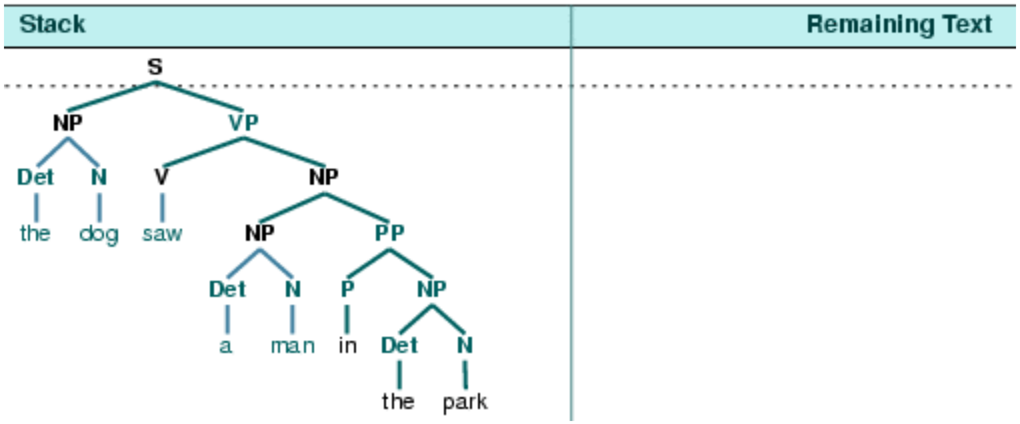


- Some work on *non-projective* dependencies
  - Common in, e.g. Czech parsing
  - Can do with MST algorithms [McDonald and Pereira 05]



Shift-Reduce Parsers

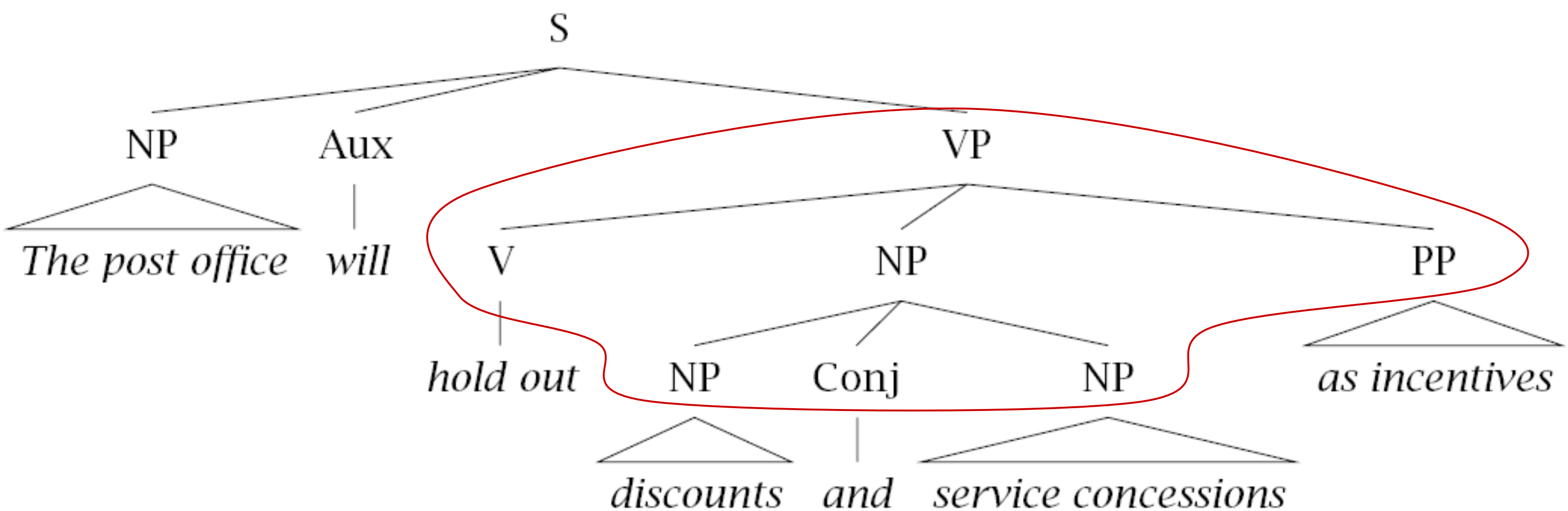
- Another way to derive a tree:



- Parsing
  - No useful dynamic programming search
  - Can still use beam search [Ratnaparkhi 97]

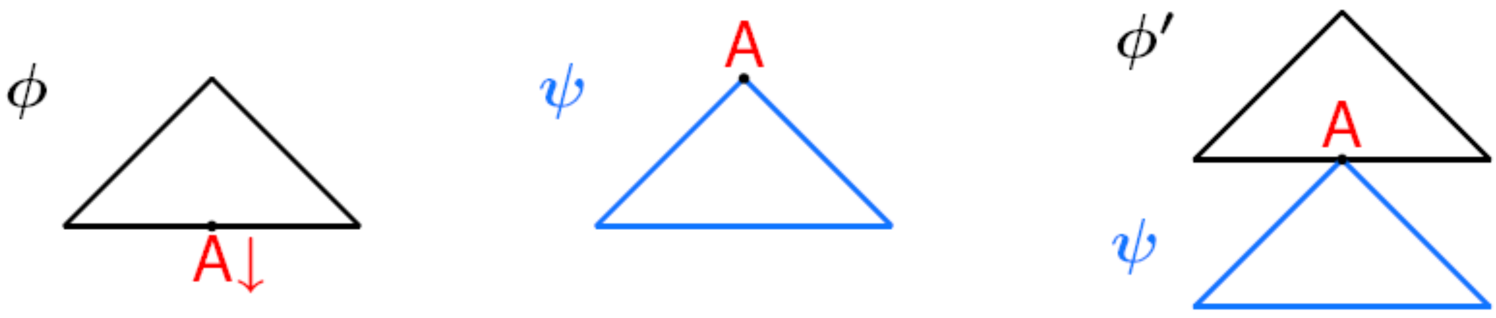
Tree Insertion Grammars

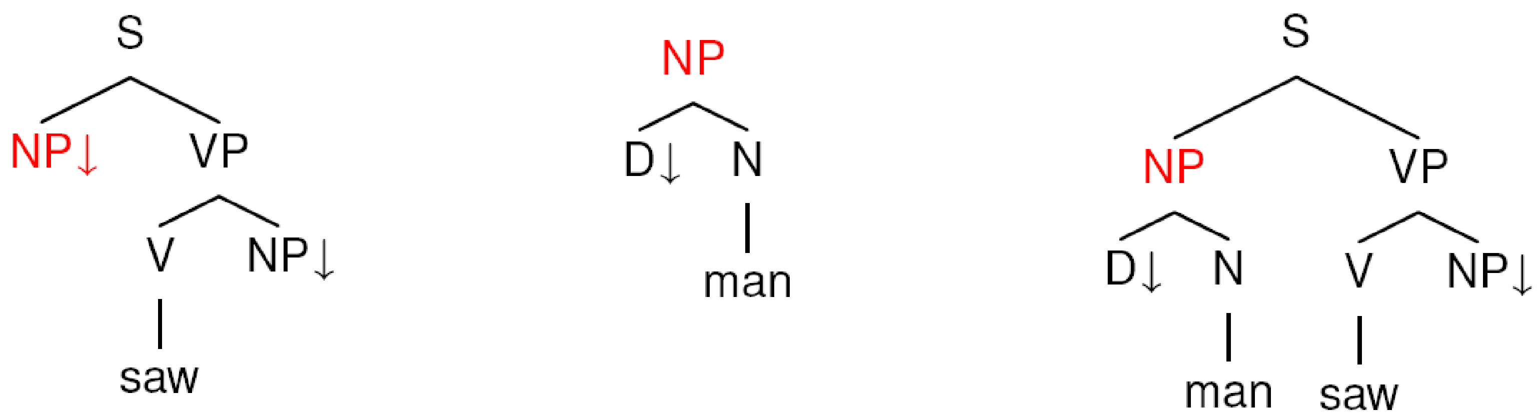
- Rewrite large (possibly lexicalized) subtrees in a single step



- Formally, a *tree-insertion grammar*
- Derivational ambiguity whether subtrees were generated atomically or compositionally
- Most probable *parse* is NP-complete

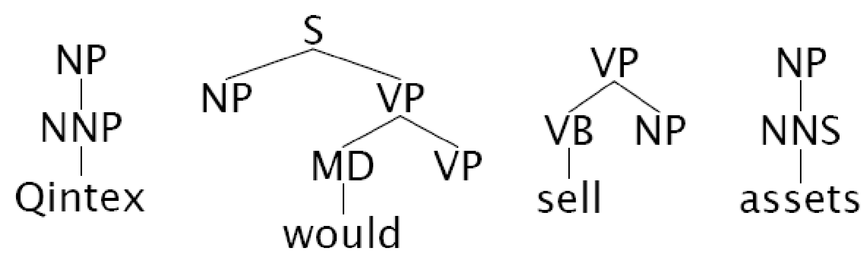
TIG: Insertion

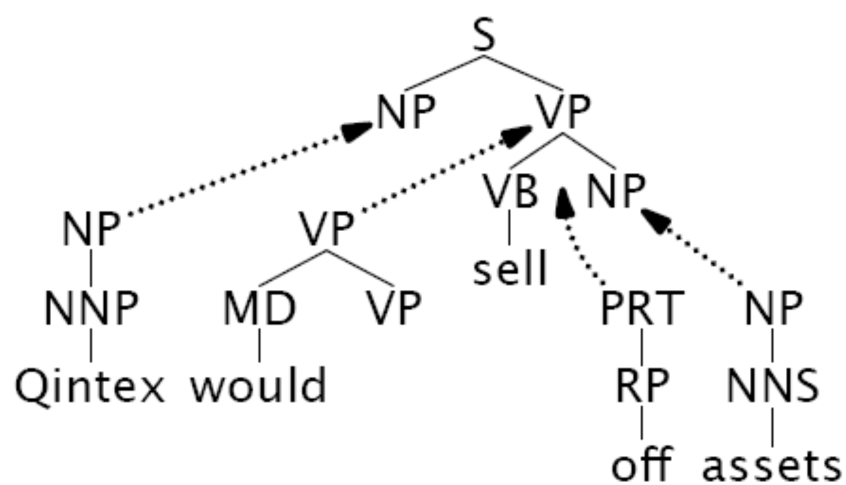




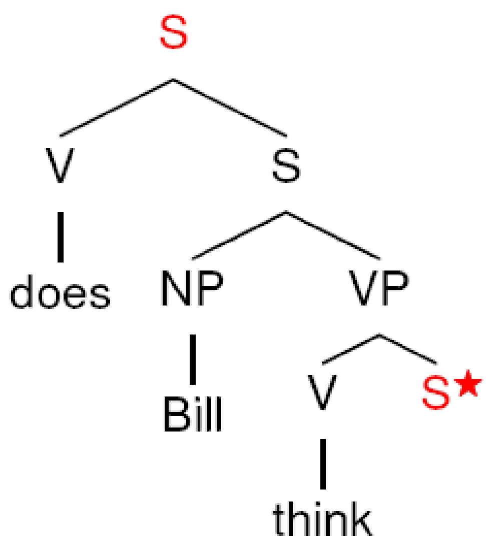
Tree-adjoining grammars

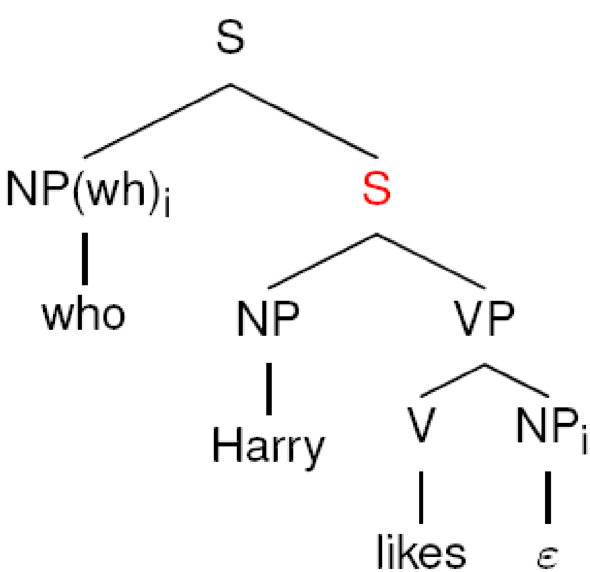
- Start with *local trees*
- Can insert structure with *adjunction* operators
- Mildly context-sensitive
- Models long-distance dependencies naturally
- … as well as other weird stuff that CFGs don’t capture well (e.g. cross-serial dependencies)

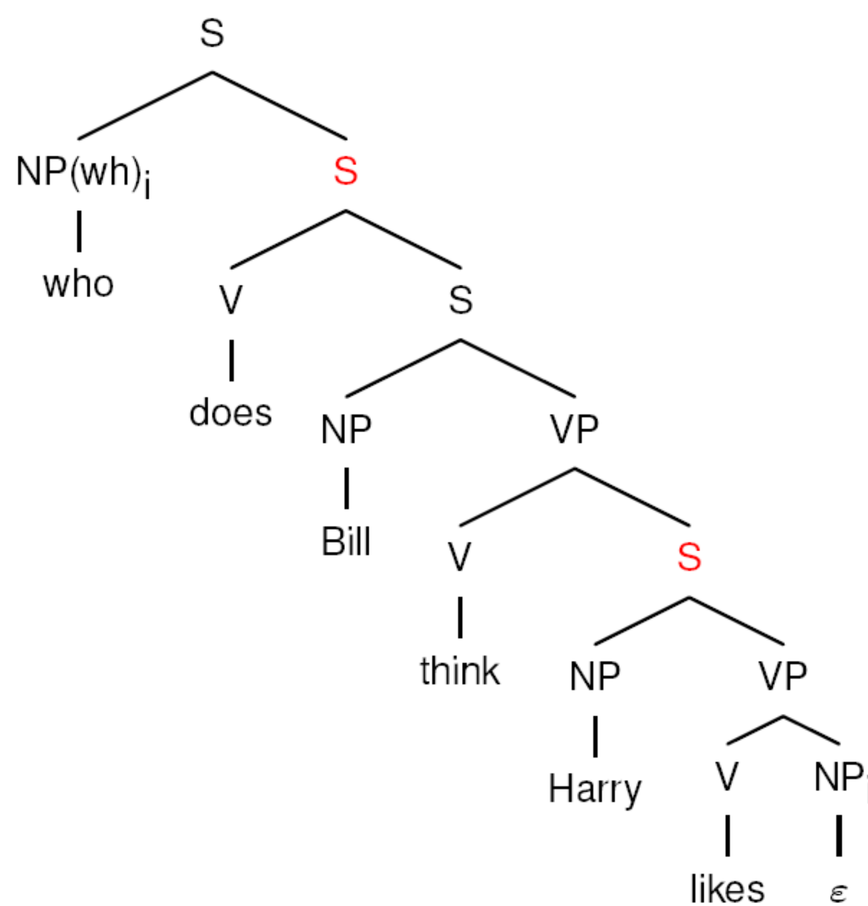




TAG: Long Distance







CCG Parsing

- Combinatory Categorial Grammar
  - Fully (mono-) lexicalized grammar
  - Categories encode argument sequences
  - Very closely related to the lambda calculus (more later)
  - Can have spurious ambiguities (why?)

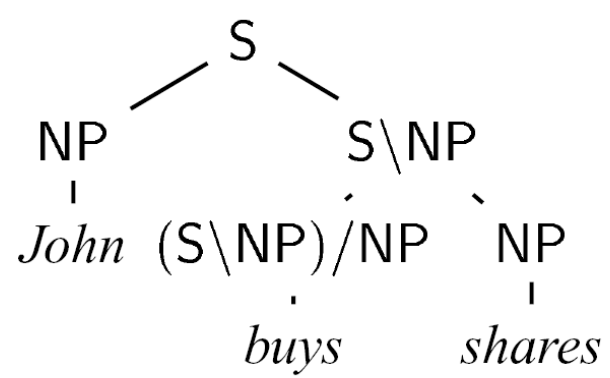
$$\textit{John} \vdash \mathsf{NP}$$

$$\textit{shares} \vdash \mathsf{NP}$$

$$\textit{buys} \vdash (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP}$$

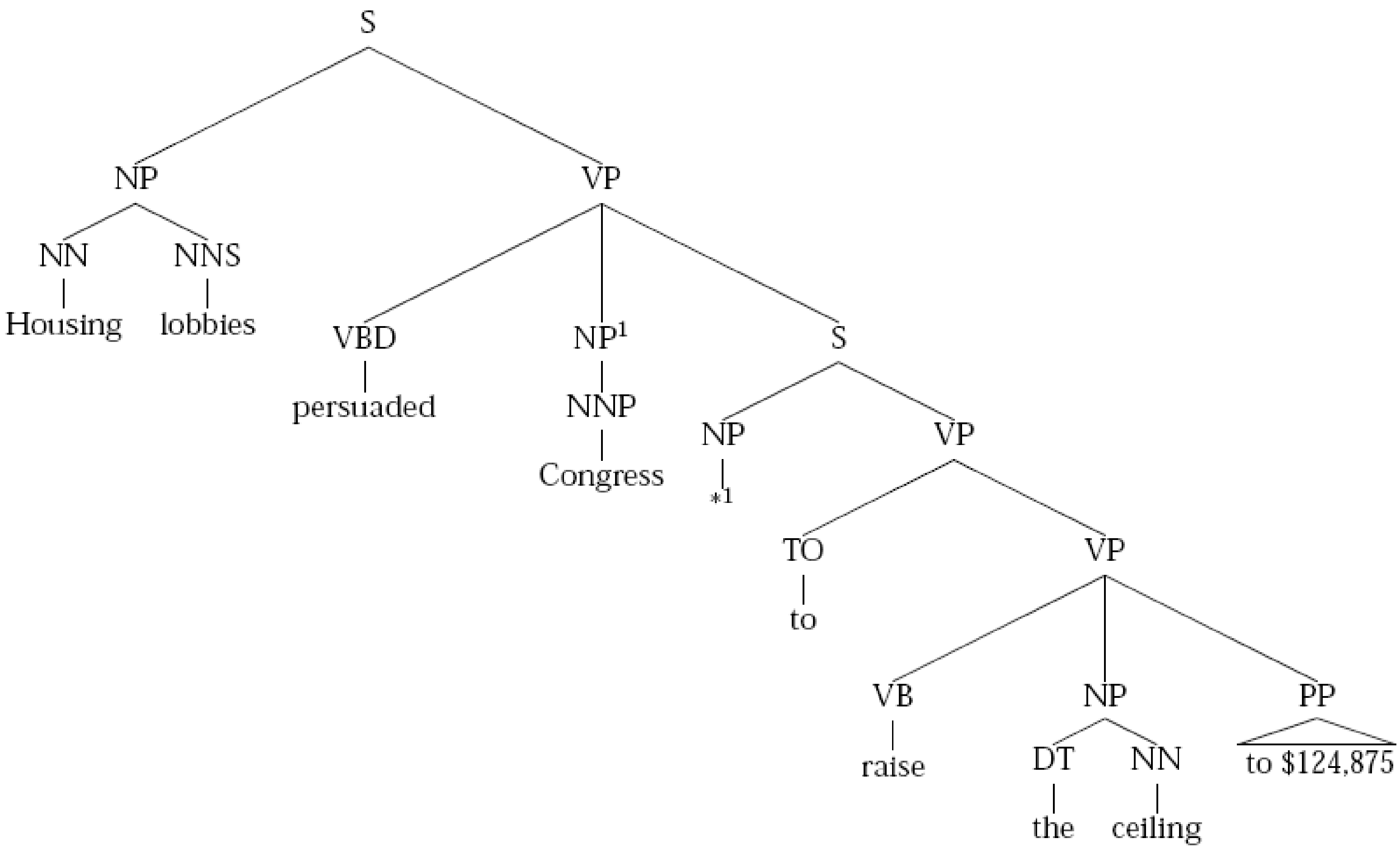
$$\textit{sleeps} \vdash \mathsf{S}\backslash\mathsf{NP}$$

$$\textit{well} \vdash (\mathsf{S}\backslash\mathsf{NP})\backslash(\mathsf{S}\backslash\mathsf{NP})$$



Empty Elements

Empty Elements

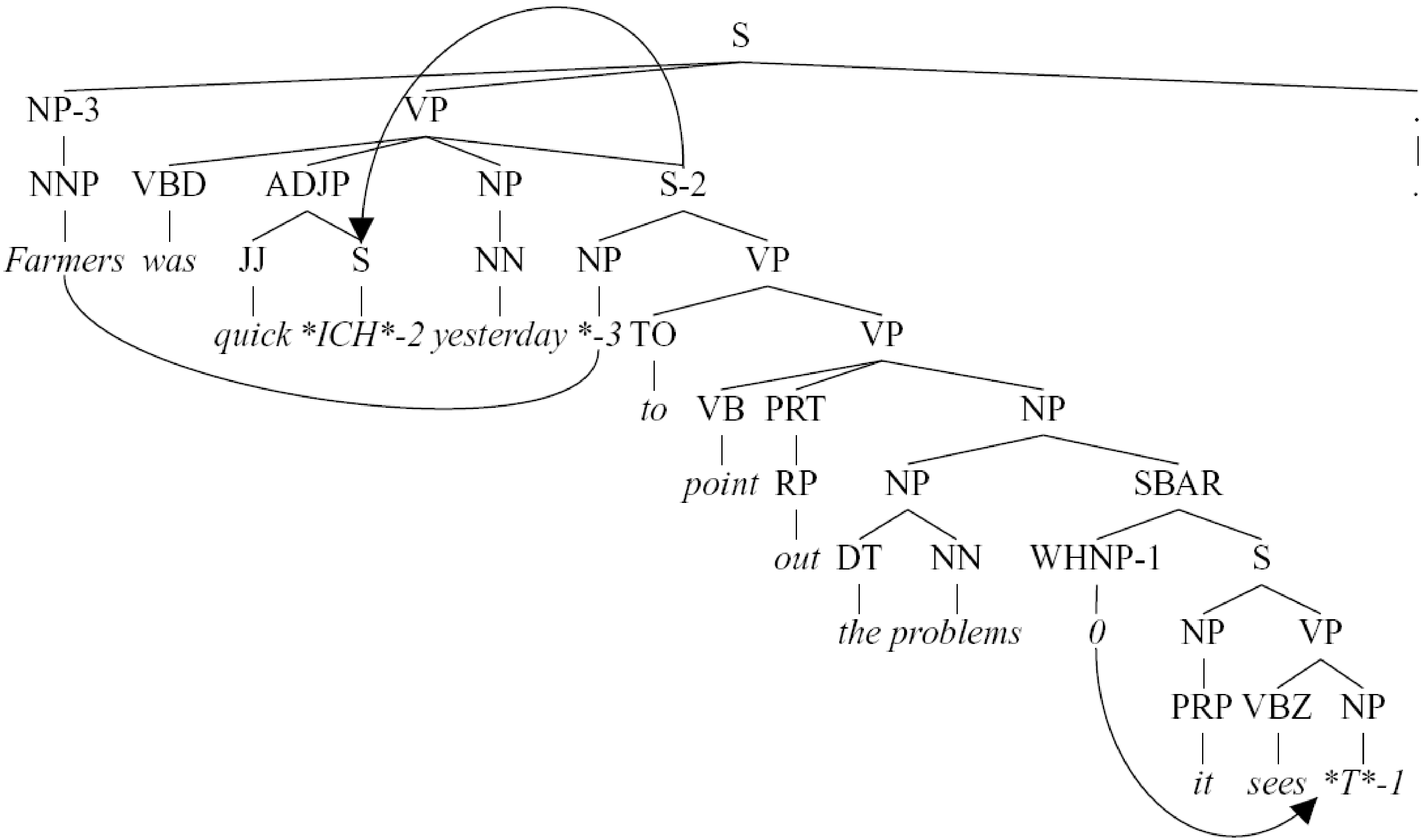


Empty Elements

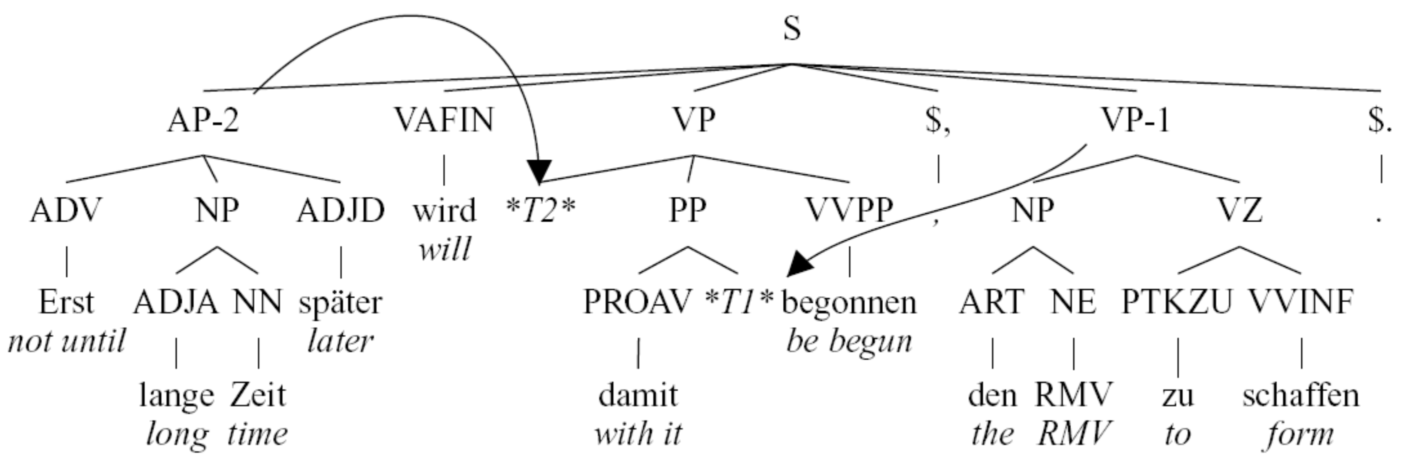
- In the PTB, three kinds of empty elements:
  - Null items (usually complementizers)
  - Dislocation (WH-traces, topicalization, relative clause and heavy NP extraposition)
  - Control (raising, passives, control, shared argumentation)

- Need to reconstruct these (and resolve any indexation)

Example: English

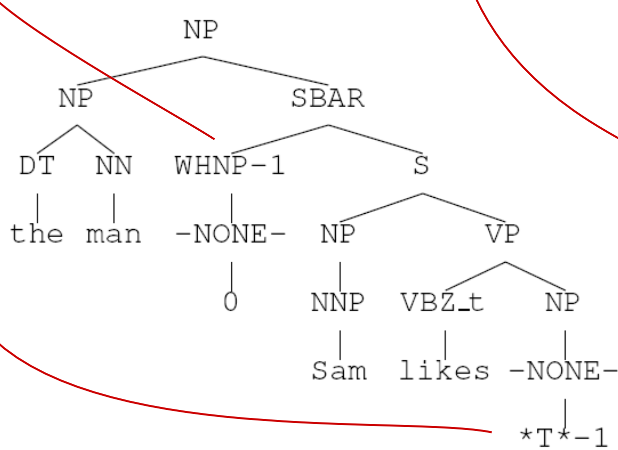


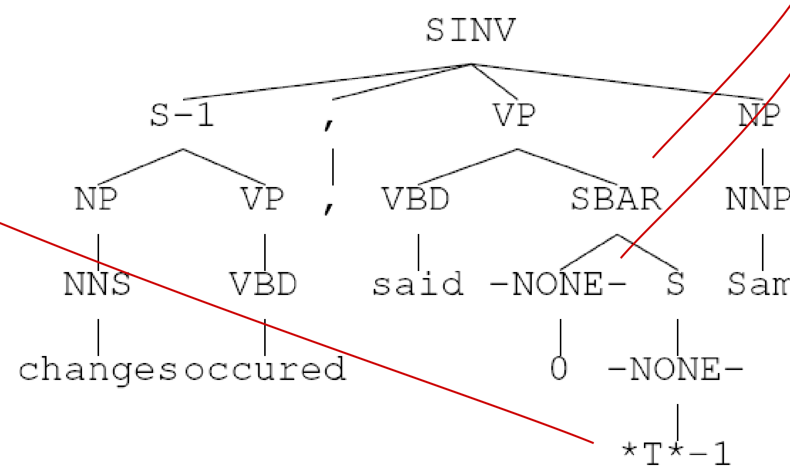
Example: German



Types of Empties

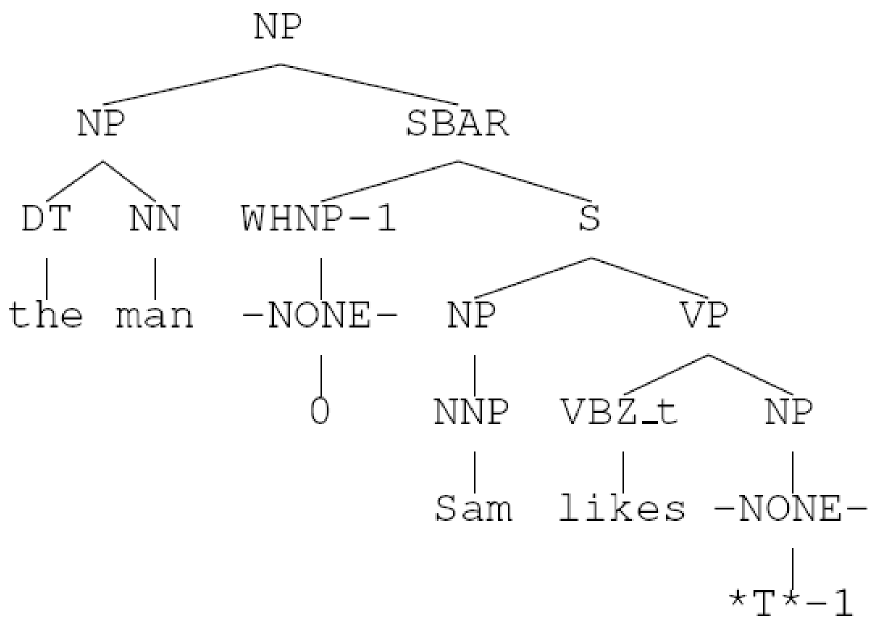
Antecedent	POS	Label	Count	Description
NP	NP	*	18,334	NP trace (e.g., *Sam was seen **)
	NP	*	9,812	NP PRO (e.g., ** to sleep is nice*)
WHNP	NP	*T*	8,620	WH trace (e.g., *the woman who you saw *T**)
		U	7,478	Empty units (e.g., *$ 25 *U**)
		0	5,635	Empty complementizers (e.g., *Sam said 0 Sasha snores*)
S	S	*T*	4,063	Moved clauses (e.g., *Sam had to go, Sasha explained *T**)
WHADVP	ADVP	*T*	2,492	WH-trace (e.g., *Sam explained how to leave *T**)
	SBAR		2,033	Empty clauses (e.g., *Sam had to go, Sasha explained (SBAR)*)
	WHNP	0	1,759	Empty relative pronouns (e.g., *the woman 0 we saw*)
	WHADVP	0	575	Empty relative pronouns (e.g., *no reason 0 to leave*)

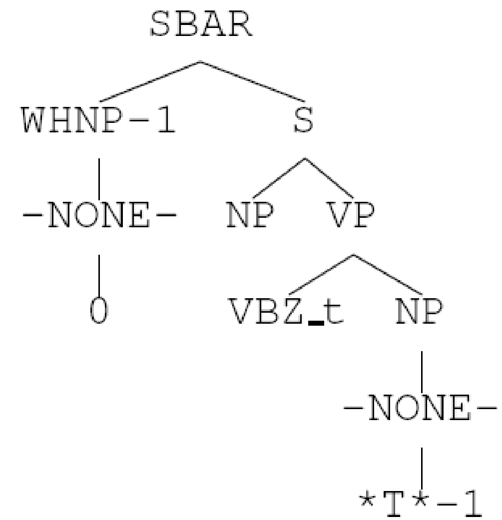


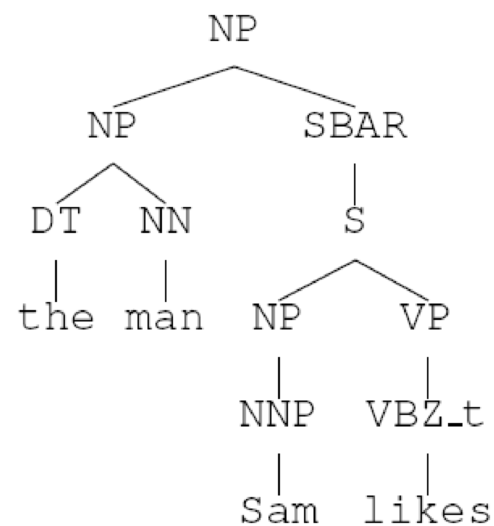


A Pattern-Matching Approach

- [Johnson 02]







Pattern-Matching Details

- Something like transformation-based learning
- Extract patterns
  - Details: transitive verb marking, auxiliaries
  - Details: legal subtrees
- Rank patterns
  - Pruning ranking: by correct / match rate
  - Application priority: by depth
- Pre-order traversal
- Greedy match

Top Patterns Extracted

Count	Match	Pattern
5816	6223	(S (NP (-NONE- *)) VP)
5605	7895	(SBAR (-NONE- 0) S)
5312	5338	(SBAR WHNP-1 (S (NP (-NONE- *T*-1)) VP))
4434	5217	(NP QP (-NONE- *U*))
1682	1682	(NP $ CD (-NONE- *U*))
1327	1593	(VP VBN_t (NP (-NONE- *)) PP)
700	700	(ADJP QP (-NONE- *U*))
662	1219	(SBAR (WHNP-1 (-NONE- 0)) (S (NP (-NONE- *T*-1)) VP))
618	635	(S S-1 , NP (VP VBD (SBAR (-NONE- 0) (S (-NONE- *T*-1)))) .)
499	512	(SINV `` S-1 , '' (VP VBZ (S (-NONE- *T*-1))) NP .)
361	369	(SINV `` S-1 , '' (VP VBD (S (-NONE- *T*-1))) NP .)
352	320	(S NP-1 (VP VBZ (S (NP (-NONE- *-1)) VP)))
346	273	(S NP-1 (VP AUX (VP VBN_t (NP (-NONE- *-1)) PP)))
322	467	(VP VBD_t (NP (-NONE- *)) PP)
269	275	(S `` S-1 , '' NP (VP VBD (S (-NONE- *T*-1))) .)

Results

Empty node		Section 23			Parser output		
POS	**Label**	P	R	f	P	R	f
(Overall)		0.93	0.83	0.88	0.85	0.74	0.79
NP	*	0.95	0.87	0.91	0.86	0.79	0.82
NP	*T*	0.93	0.88	0.91	0.85	0.77	0.81
	0	0.94	0.99	0.96	0.86	0.89	0.88
	U	0.92	0.98	0.95	0.87	0.96	0.92
S	*T*	0.98	0.83	0.90	0.97	0.81	0.88
ADVP	*T*	0.91	0.52	0.66	0.84	0.42	0.56
SBAR		0.90	0.63	0.74	0.88	0.58	0.70
WHNP	0	0.75	0.79	0.77	0.48	0.46	0.47

Semantic Roles

Semantic Role Labeling (SRL)

- Characterize clauses as *relations* with *roles*:

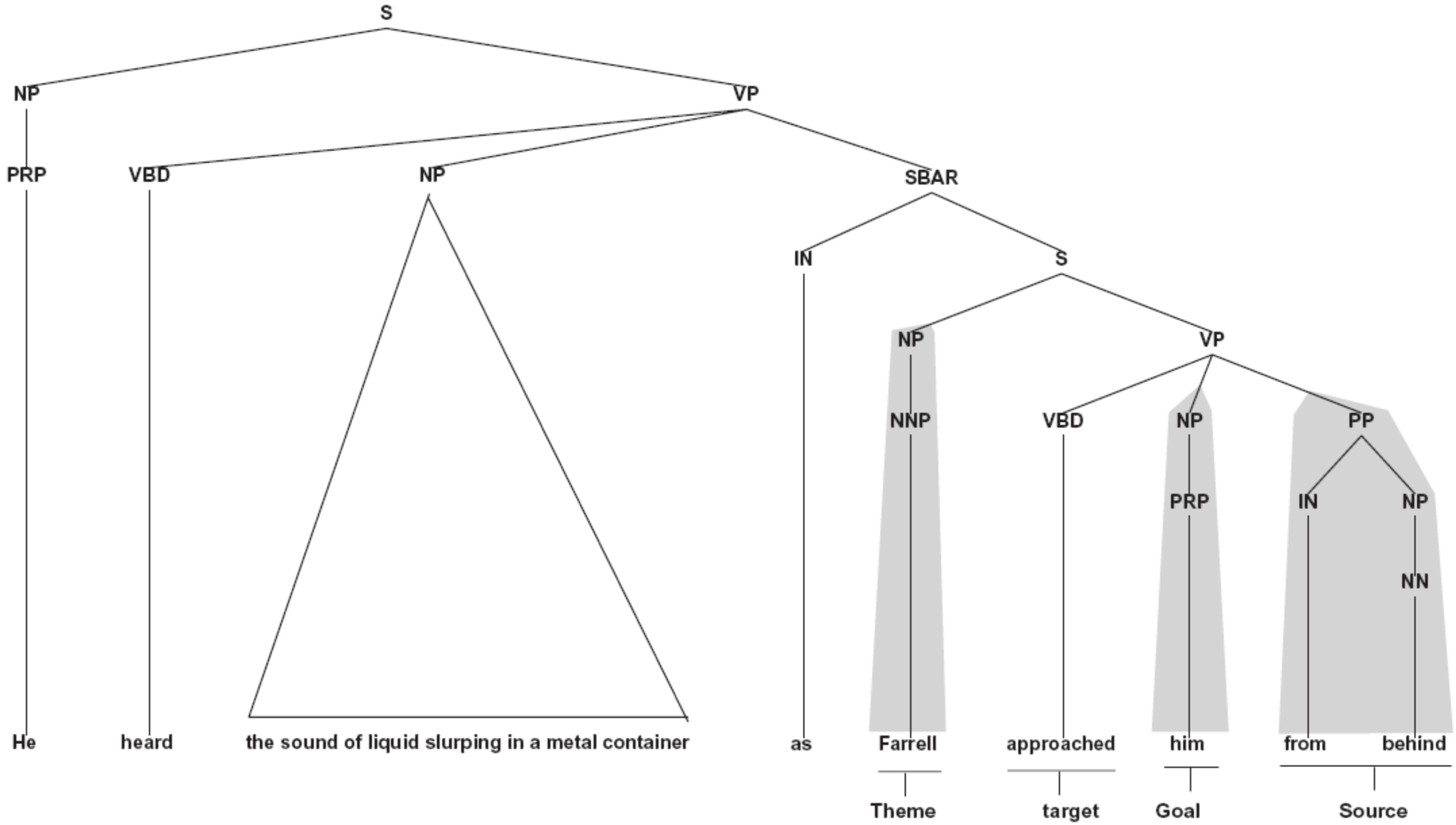
[$_{Judge}$ She] **blames** [$_{Evaluee}$ the Government] [$_{Reason}$ for failing to do enough to help] .

Holman would characterise this as **blaming** [$_{Evaluee}$ the poor] .

The letter quotes Black as saying that [$_{Judge}$ white and Navajo ranchers] misrepresent their livestock losses and **blame** [$_{Reason}$ everything] [$_{Evaluee}$ on coyotes] .

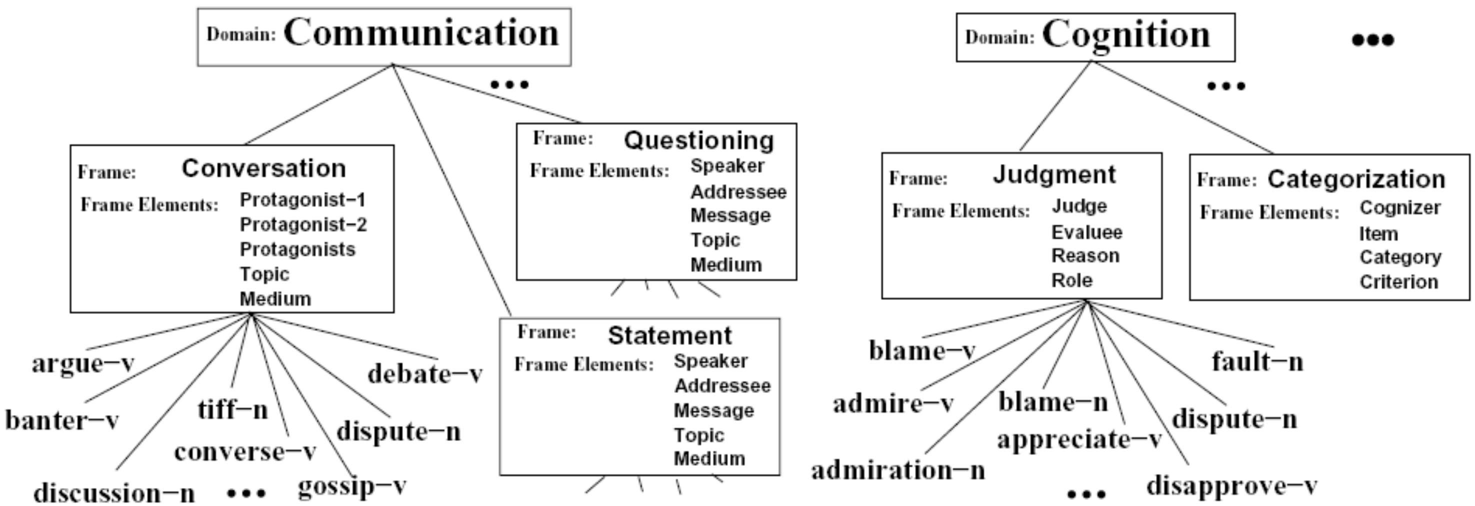
- Says more than which NP is the subject (but not much more):
- Relations like *subject* are syntactic, relations like *agent* or *message* are semantic
- Typical pipeline:
  - Parse, then label roles
  - Almost all errors locked in by parser
  - Really, SRL is quite a lot easier than parsing

SRL Example



S
NP
VP
PRP
VBD
NP
SBAR
IN
S
NP
VP
NNP
VBD
NP
PP
PRP
IN
NP
NN
He
heard
the sound of liquid slurping in a metal container
as
Farrell
approached
him
from
behind
Theme
target
Goal
Source

PropBank / FrameNet



- FrameNet: roles shared between verbs
- PropBank: each verb has its own roles
- PropBank more used, because it's layered over the treebank (and so has greater coverage, plus parses)
- Note: some linguistic theories postulate fewer roles than FrameNet (e.g. 5-20 total: agent, patient, instrument, etc.)

PropBank Example

fall.01		sense: move downward	
	roles:	Arg1:	thing falling
		Arg2:	extent, distance fallen
		Arg3:	start point
		Arg4:	end point

Sales fell to $251.2 million from $278.7 million.

arg1: Sales
rel: fell
arg4: to $251.2 million
arg3: from $278.7 million

PropBank Example

rotate.02		sense: shift from one thing to another	
	roles:	Arg0:	causer of shift
		Arg1:	thing being changed
		Arg2:	old thing
		Arg3:	new thing

Many of Wednesday's winners were losers yesterday as investors quickly took profits and rotated their buying to other issues, traders said. (wsj_1723)

arg0:	investors
rel:	rotated
arg1:	their buying
arg3:	to other issues

PropBank Example

aim.01 sense: intend, plan
roles: Arg0: aimer, planner
Arg1: plan, intent

The Central Council of Church Bell Ringers aims *trace* to improve relations with vicars. (wsj_0089)

arg0: The Central Council of Church Bell Ringers
rel: aims
arg1: *trace* to improve relations with vicars

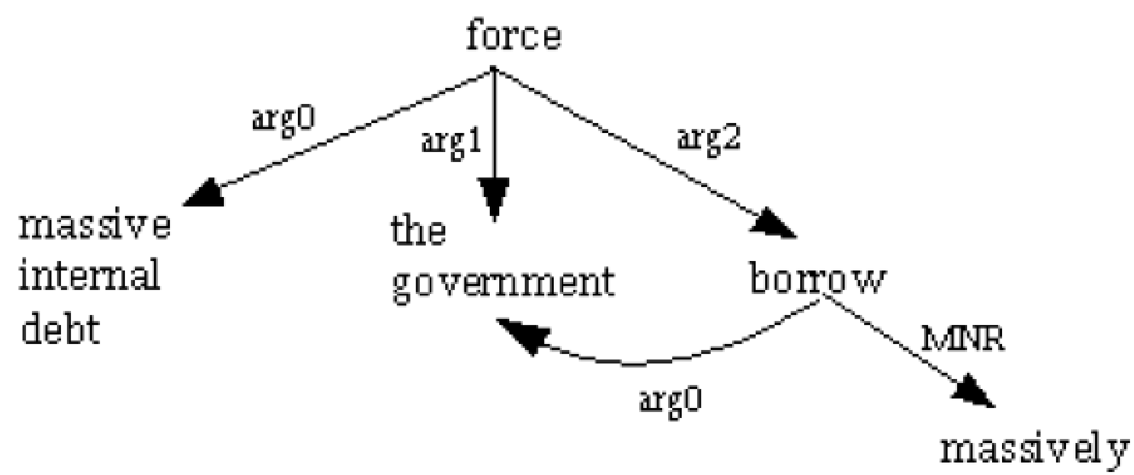
aim.02 sense: point (weapon) at
roles: Arg0: aimer
Arg1: weapon, etc.
Arg2: target

Banks have been aiming packages at the elderly.

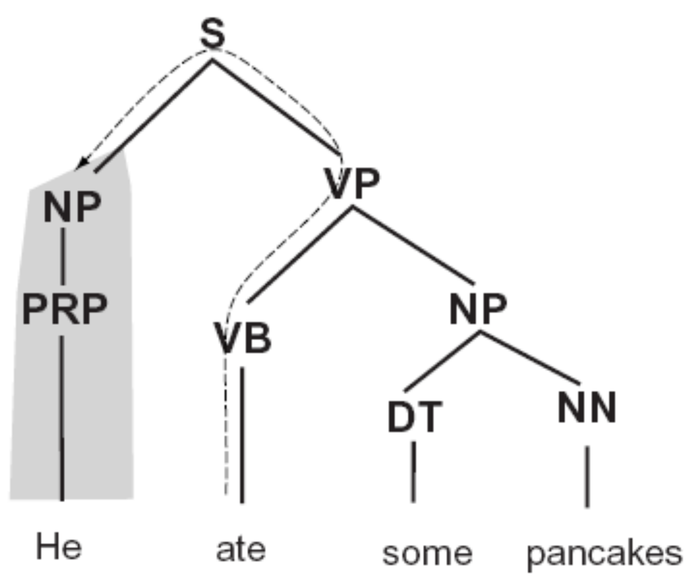
arg0: Banks
rel: aiming
arg1: packages
arg2: at the elderly

Shared Arguments

```
(NP-SBJ (JJ massive) (JJ internal) (NN debt) )
  (VP (VBZ has)
   (VP (VBN forced)
    (S
     (NP-SBJ-1 (DT the) (NN government) )
     (VP
      (VP (TO to)
       (VP (VB borrow)
        (ADVP-MNR (RB massively) )...
```



Path Features



Path	*Description*
VB↑VP↓PP	PP argument/adjunct
VB↑VP↑S↓NP	subject
VB↑VP↓NP	object
VB↑VP↑VP↑S↓NP	subject (embedded VP)
VB↑VP↓ADVP	adverbial adjunct
NN↑NP↑NP↓PP	prepositional complement of noun

Results

- Features:
  - Path from target to filler
  - Filler's syntactic type, headword, case
  - Target's identity
  - Sentence voice, etc.
  - Lots of other second-order features

- Gold vs parsed source trees
  - SRL is fairly easy on gold trees

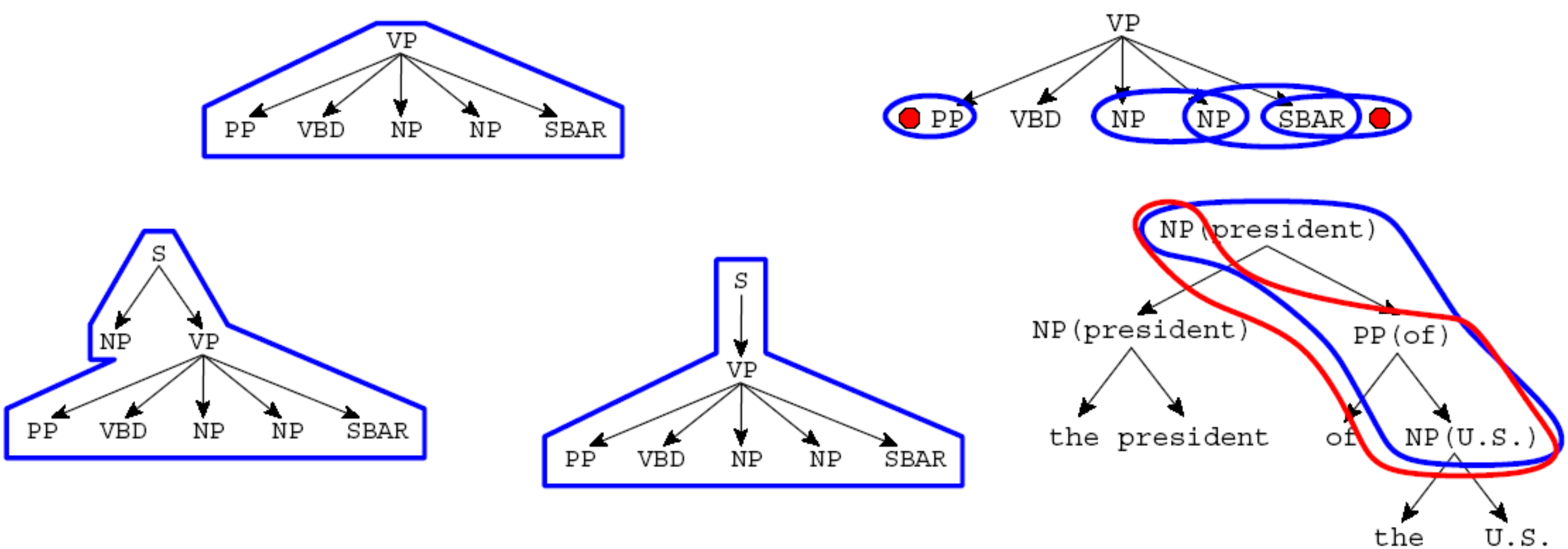
CORE		ARGM	
F1	Acc.	F1	Acc.
92.2	80.7	89.9	71.8

  - Harder on automatic parses

CORE		ARGM	
F1	Acc.	F1	Acc.
84.1	66.5	81.4	55.6

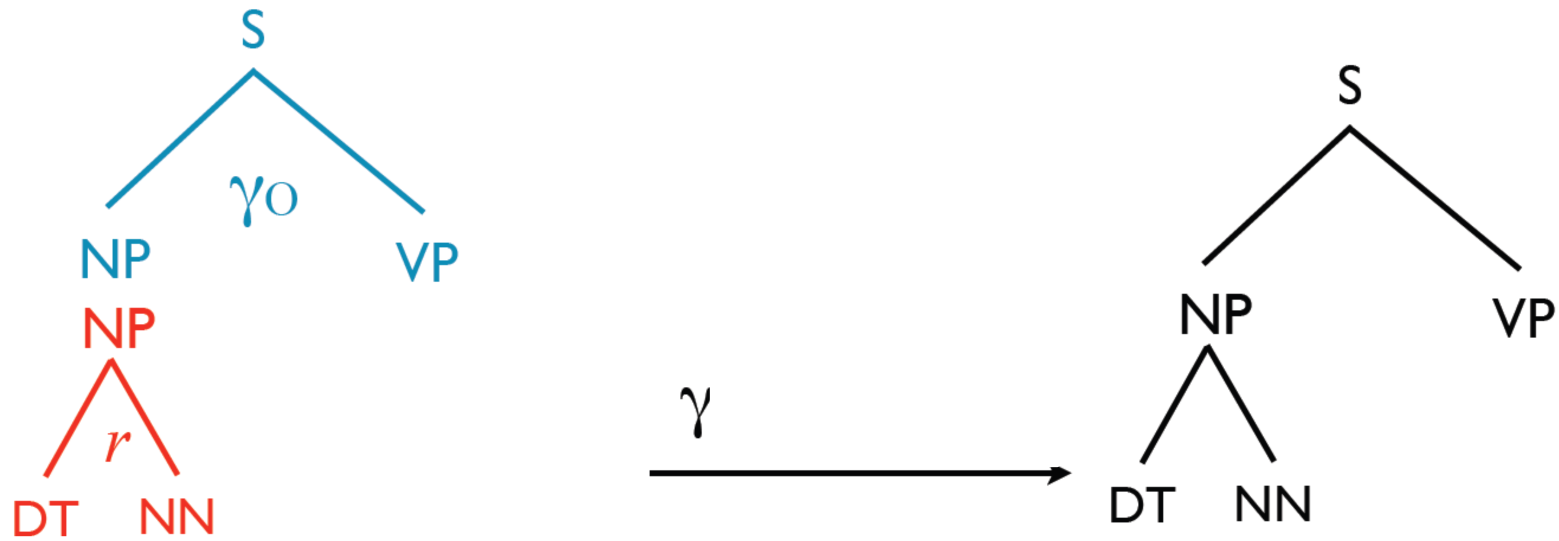
Parse Reranking

- Assume the number of parses is very small
- We can represent each parse T as a feature vector $\varphi(T)$
  - Typically, all local rules are features
  - Also non-local features, like how right-branching the overall tree is
  - [Charniak and Johnson 05] gives a rich set of features



K-Best Parsing

[Huang and Chiang 05, Pauls, Klein, Quirk 10]



$\gamma = \gamma_O + r$