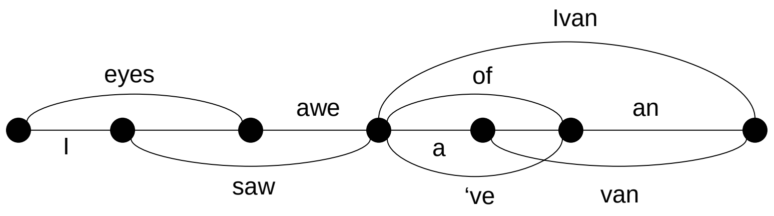
Statistical NLP
Spring 2007

Lecture 17: Lexicalized Parsing

Dan Klein – UC Berkeley

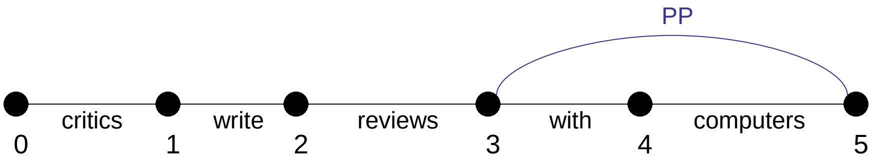
(Speech) Lattices

- 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.



A Simple Chart Parser

- Chart parsers are sparse dynamic programs
- Ingredients:
  - Nodes: positions between words
  - Edges: spans of words with labels, represent the set of trees over those words rooted at x
  - A chart: records which edges we've built
  - An agenda: a holding pen for edges (a queue)
- We're going to figure out:
  - What edges can we build?
  - All the ways we built them.



PP

critics write reviews with computers

0 1 2 3 4 5

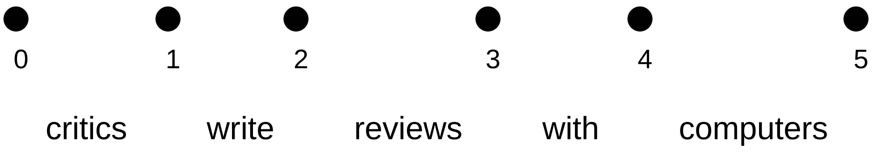
Word Edges

- An edge found for the first time is called discovered. Edges go into the agenda on discovery.
- To initialize, we discover all word edges.

AGENDA

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

CHART [EMPTY]



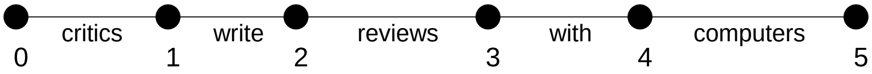
0 1 2 3 4 5

critics write reviews with computers

Unary Projection

- When we pop an word edge off the agenda, we check the lexicon to see what tag edges we can build from it

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[3,4]



critics write reviews with computers

The “Fundamental Rule”

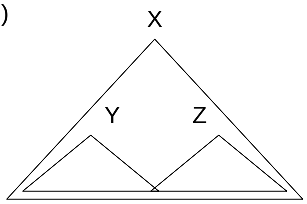
- When we pop edges off of the agenda:
  - Check for unary projections (NNS → critics, NP → NNS)

Y[i,j] with X → Y forms X[i,j]

  - Combine with edges already in our chart (this is sometimes called the fundamental rule)

Y[i,j] and Z[j,k] with X → Y Z form X[i,k]

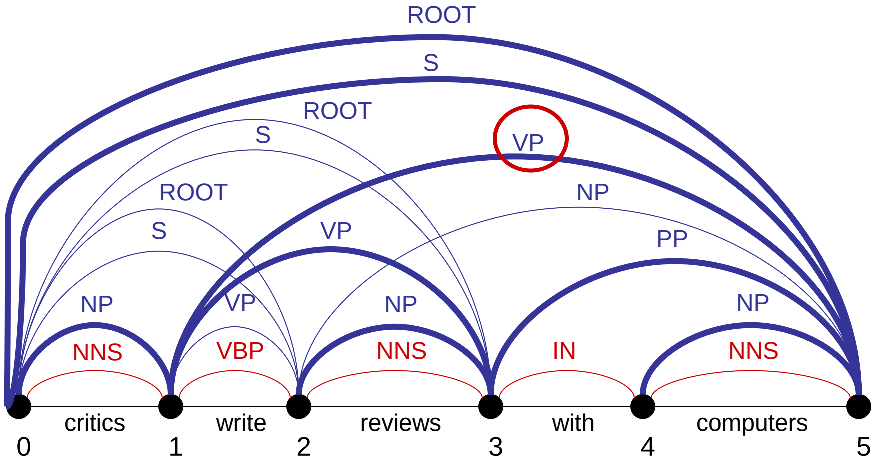
  - Enqueue resulting edges (if newly discovered)
  - Record backtraces (called traversals)
  - Stick the popped edge in the chart



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

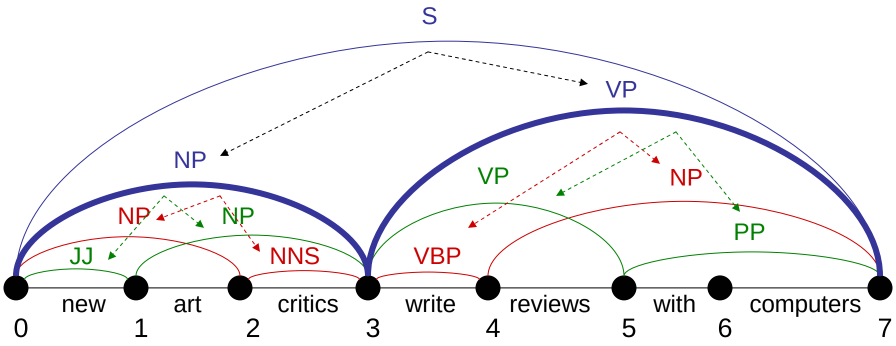
An Example

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]



Exploiting Substructure

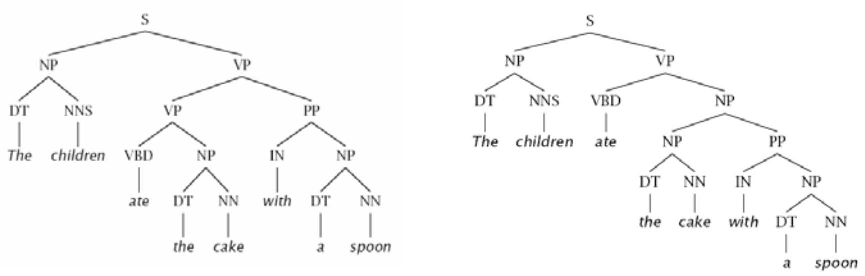
- Each edge records all the ways it was built (locally)
  - Can recursively extract trees
  - A chart may represent too many parses to enumerate (how many?)



Order Independence

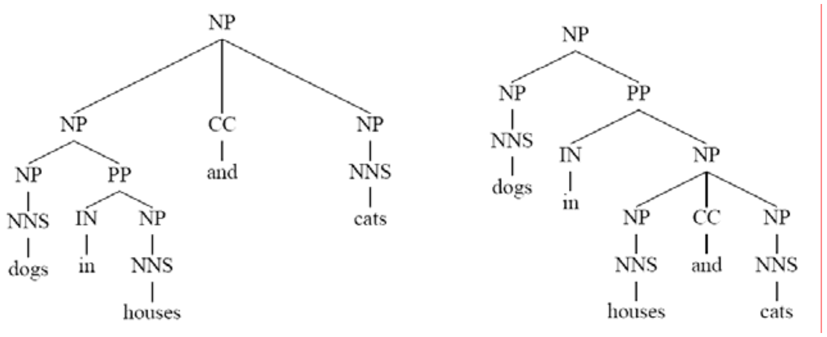
- A nice property:
  - It doesn't matter what policy we use to order the agenda (FIFO, LIFO, random).
  - Why? Invariant: before popping an edge:
    - Any edge X[i,j] that can be directly built from chart edges and a single grammar rule is either in the chart or in the agenda.
    - Convince yourselves this invariant holds!
  - This will not be true weighted parsers:
    - Instead must also insure that an edge has best score when added to the chart
    - Sufficient (but not necessary) to order agenda items by current best score

Problems with PCFGs?



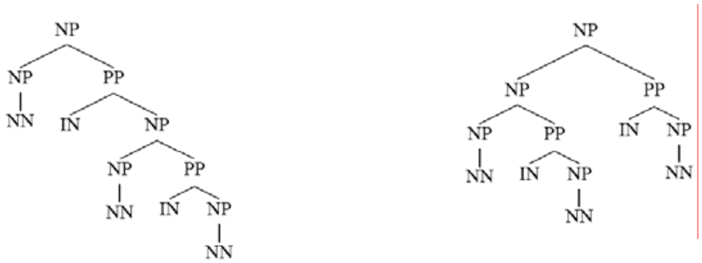
- If we do no annotation, these trees differ only in one rule:
  - VP → VP PP
  - NP → NP PP
- Parse will go one way or the other, regardless of words
- We addressed this in one way with unlexicalized grammars (how?)
- Lexicalization allows us to be sensitive to specific words

Problems with PCFGs



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

Problems with PCFGs

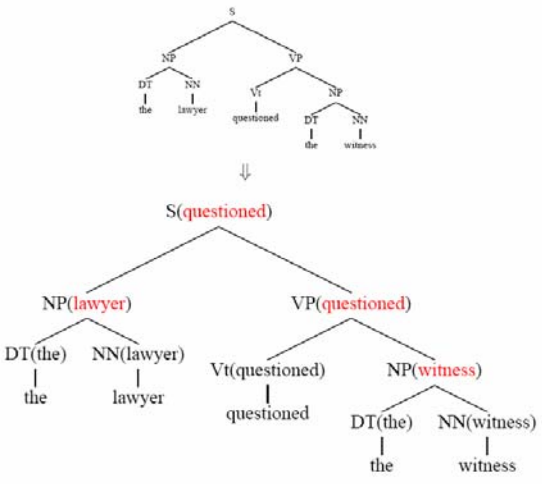


president of a company in Africa

- Another example of PCFG indifference
  - Left structure far more common
  - How to model this?
  - Really structural: “chicken with potatoes with gravy”
  - Lexical parsers model this effect, but not by virtue of being lexical

Lexicalized Trees

- 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

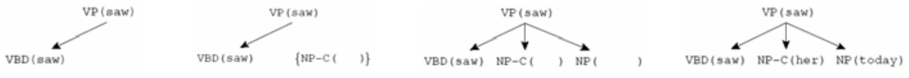


Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like

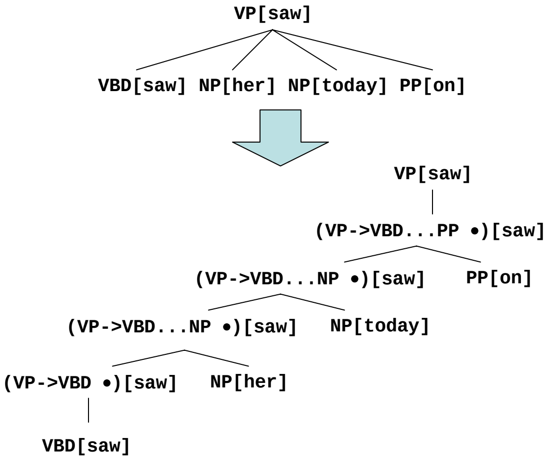
```
VP(saw) -> VBD(saw) NP-C(her) NP(today)
```

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



Lexical Derivation Steps

- Simple derivation of a local tree [simplified Charniak 97]



VP[saw]

VBD[saw] NP[her] NP[today] PP[on]

VP[saw]

(VP->VBD...PP •)[saw]

(VP->VBD...NP •)[saw] PP[on]

(VP->VBD...NP •)[saw] NP[today]

(VP->VBD •)[saw] NP[her]

VBD[saw]

Still have to smooth with mono- and non-lexical backoffs

$$P(\texttt{STOP}|\texttt{VBD}[saw], \texttt{VP}, \texttt{PP})$$

$$P(\texttt{PP}[on]|\texttt{VBD}[saw], \texttt{VP}, \texttt{NP})$$

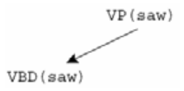
$$P(\texttt{NP}[today]|\texttt{VBD}[saw], \texttt{VP}, \texttt{NP})$$

$$P(\texttt{NP}[her]|\texttt{VBD}[saw], \texttt{VP}, \texttt{START})$$

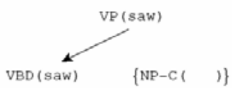
$$P(\texttt{VBD}[saw]|\texttt{VP}[saw])$$

Lexical Derivation Steps

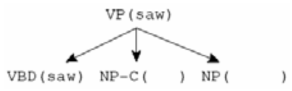
- Another derivation of a local tree [Collins 99]



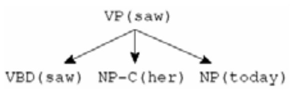
Choose a head tag and word



Choose a complement bag



Generate children (incl. adjuncts)

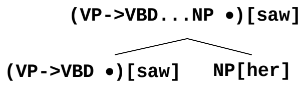


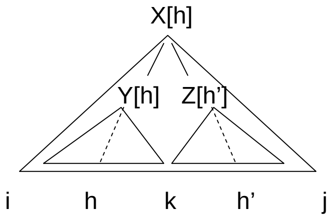
Recursively derive children

Naïve Lexicalized Parsing

- Can, in principle, use CKY on lexicalized PCFGs
  - $O(Rn^3)$ time and $O(Sn^2)$ memory
  - But $R = rV^2$ and $S = sV$
  - Result is completely impractical (why?)
  - Memory: 10K rules * 50K words * (40 words)^2 * 8 bytes ≈ 6TB
- Can modify CKY to exploit lexical sparsity
  - Lexicalized symbols are a base grammar symbol and a pointer into the input sentence, not any arbitrary word
  - Result: $O(rn^5)$ time, $O(sn^3)$
  - Memory: 10K rules * (40 words)^3 * 8 bytes ≈ 5GB

Lexicalized CKY

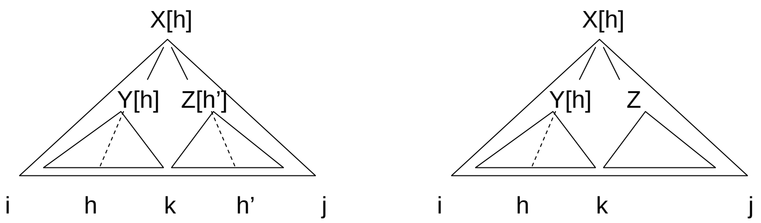




```
bestScore(X,i,j,h)
  if (j = i+1)
    return tagScore(X,s[i])
  else
    return
      max $\max_{k,X\to YZ}$ score(X[h]->Y[h] Z[h’]) *
                      bestScore(Y,i,k,h) *
                      bestScore(Z,k,j,h’)
          $\max_{k,X\to YZ}$ score(X[h]->Y[h’] Z[h]) *
                      bestScore(Y,i,k,h’) *
                      bestScore(Z,k,j,h)
```

Quartic Parsing

- Turns out, you can do better [Eisner 99]



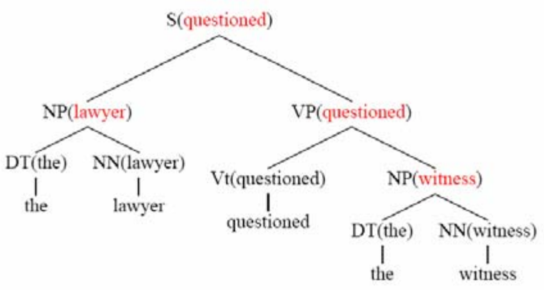
X[h]
Y[h] Z[h']
i h k h' j

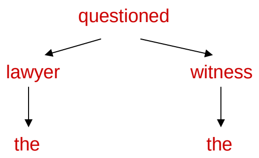
X[h]
Y[h] Z
i h k j

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

Dependency Parsing

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



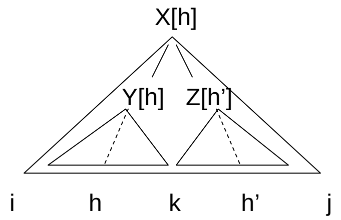


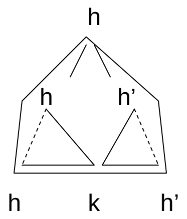
questioned
lawyer
witness
the
the

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

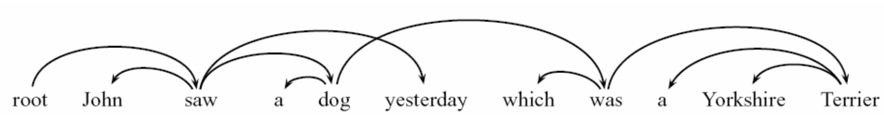
Dependency Parsing

- Pure dependency parsing is only cubic [Eisner 99]



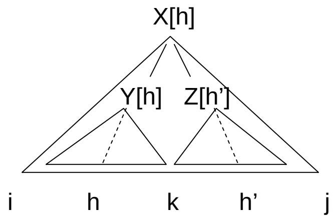


- Some work on *non-projective* dependencies
  - Common in, e.g. Czech parsing
  - Can do with MST algorithms [McDonald and Pereira 05]



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

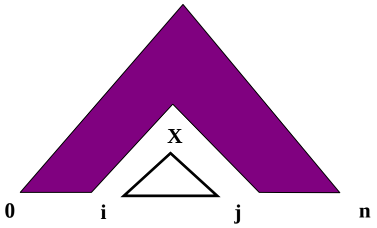


Pruning with a PCFG

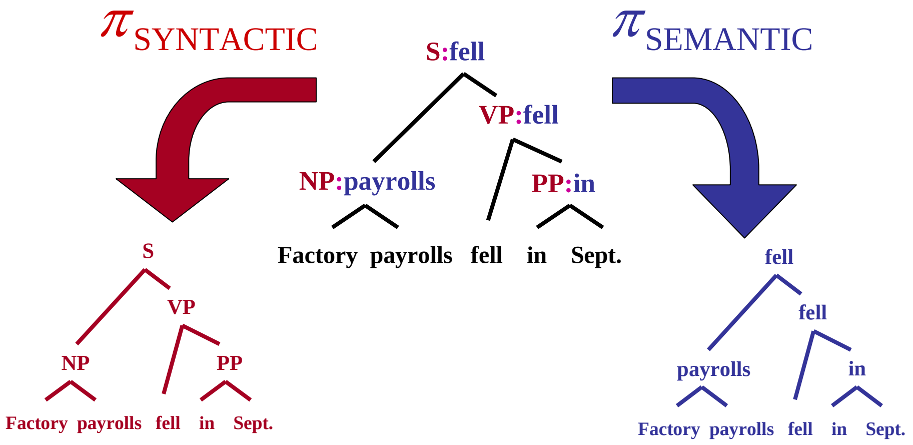
- The Charniak parser prunes using a two-pass approach [Charniak 97+]
  - First, parse with the base grammar
  - For each X:[i,j] calculate P(X|i,j,s)
    - This isn’t trivial, and there are clever speed ups
  - Second, do the full $O(n^5)$ CKY
    - Skip any X :[i,j] which had low (say, < 0.0001) posterior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

Pruning with A*

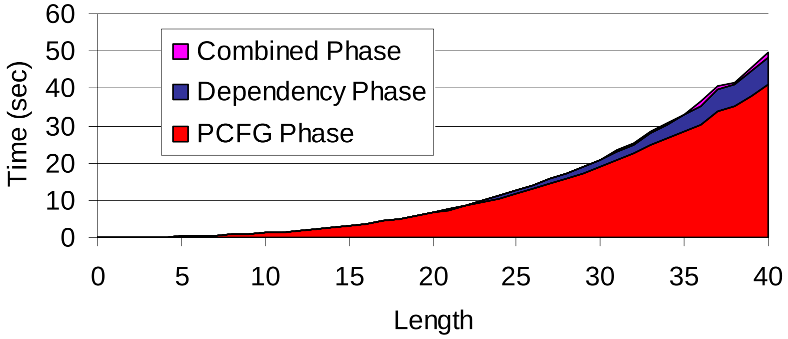
- You can also speed up the search without sacrificing optimality
- For agenda-based parsers:
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A* heuristic, no loss of optimiality [Klein and Manning 03]



Projection-Based A*



A* Speedup



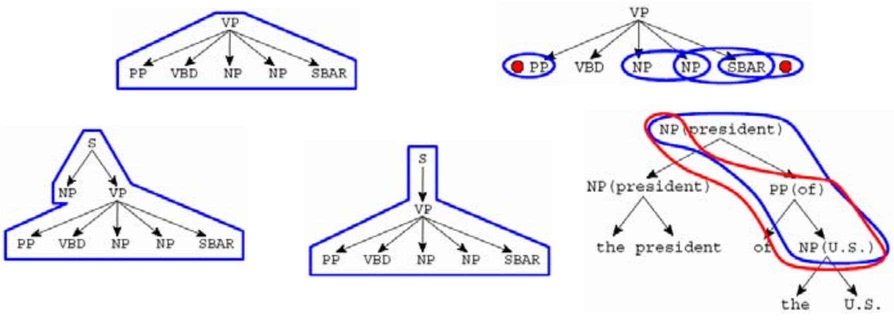
- Total time dominated by calculation of A* tables in each projection… O(n^3)

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
- Bilexical vs. monolexical vs. smart smoothing

Parse Reranking

- Assume the number of parses is very small
- We can represent each parse T as an arbitrary feature vector φ(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

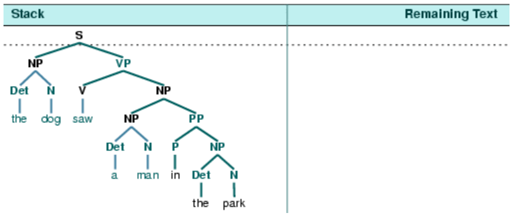


Parse Reranking

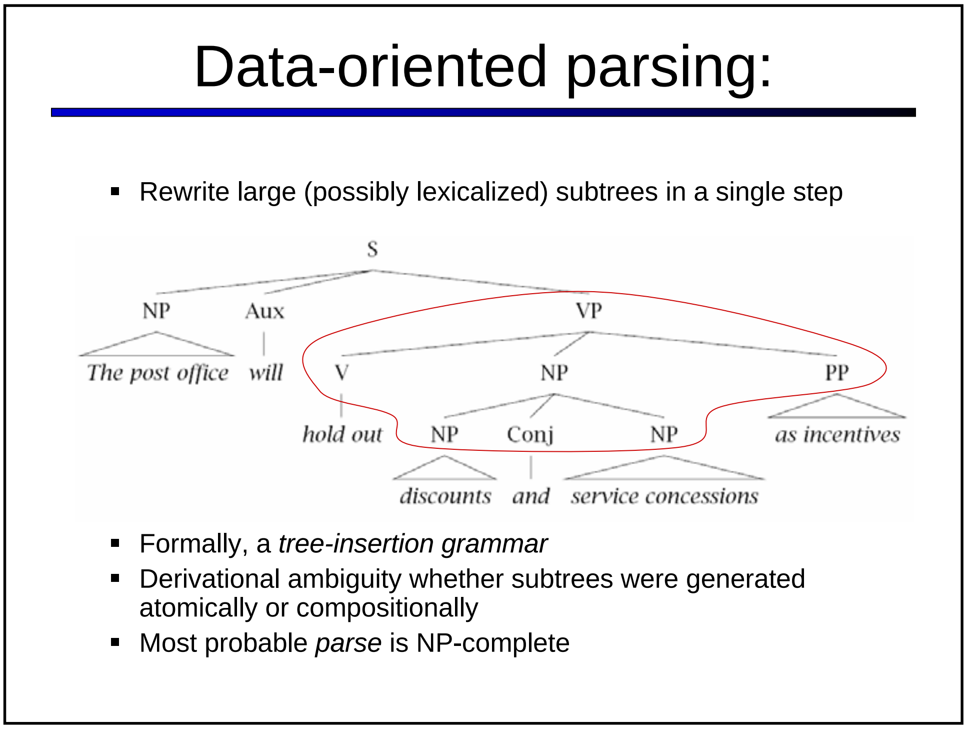
- Since the number of parses is no longer huge
  - Can enumerate all parses efficiently
  - Can use simple machine learning methods to score trees
  - E.g. maxent reranking: learn a binary classifier over trees where:
    - The top candidates are positive
    - All others are negative
    - Rank trees by P(+|T)
- The best parsing numbers are from reranking systems

Shift-Reduce Parsers

- Another way to derive a tree:



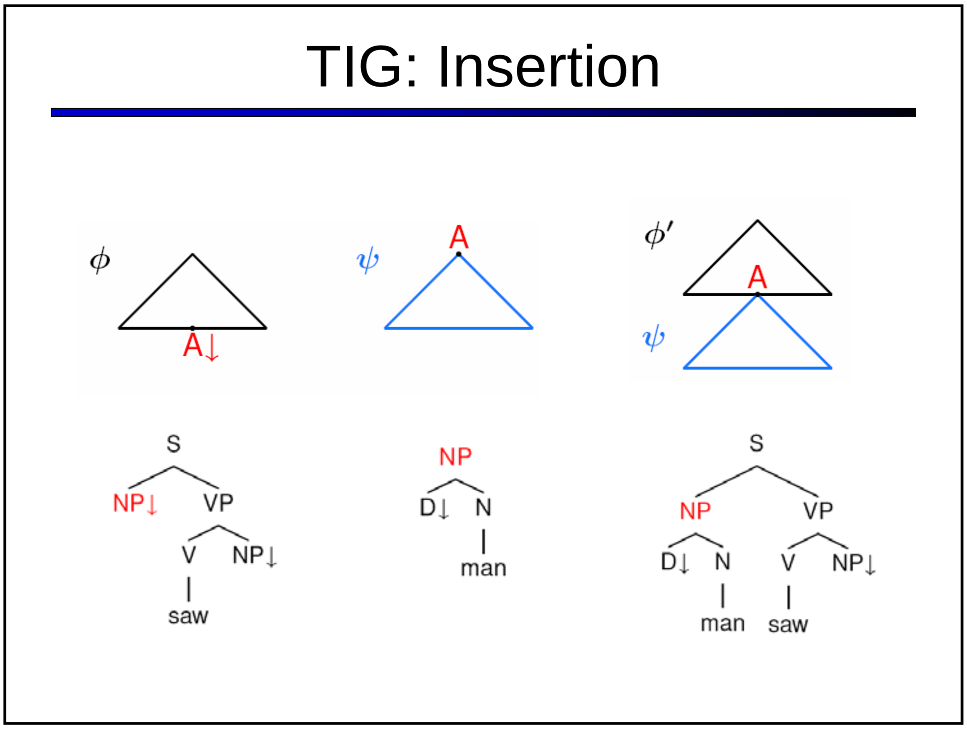
- Parsing
  - No useful dynamic programming search
  - Can still use beam search [Ratnaparkhi 97]



Data-oriented parsing:

- 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

Derivational Representations

- Generative derivational models:

$$P(D) = \prod_{d_i \in D} P(d_i|d_0 \ldots d_{i-1})$$

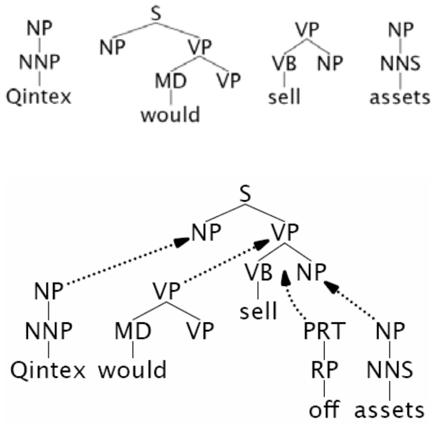
- How is a PCFG a generative derivational model?
- Distinction between *parses* and *parse derivations*.

$$P(T) = \sum_{D:D\to T} P(D)$$

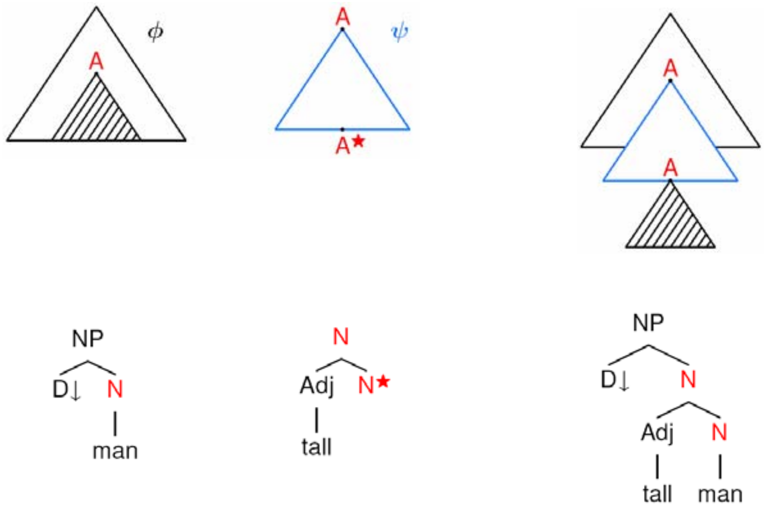
- How could there be multiple derivations?

Tree-adjoining grammars

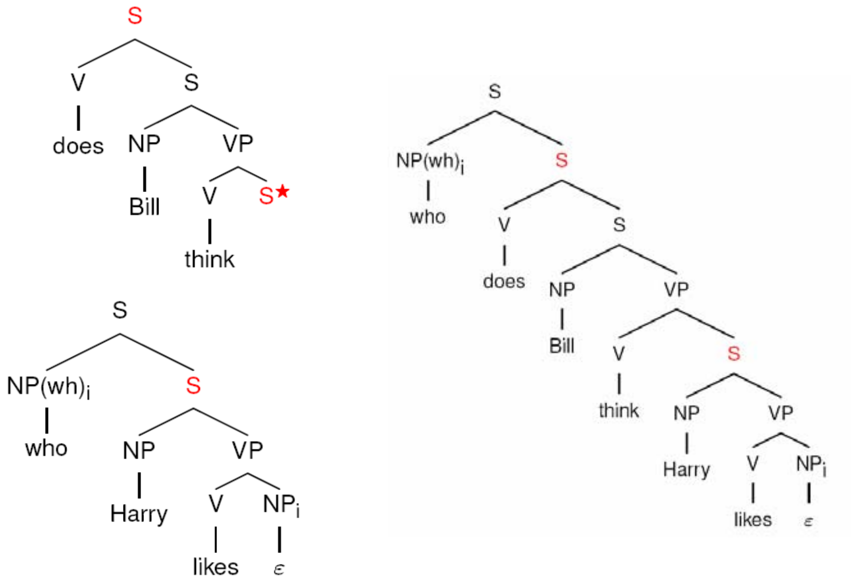
- Start with *local trees*
- Can insert structure with *adjunction* operators
- Mildly context-sensitive
- Models long-distance dependencies naturally
- … as well as other weird stuff that CFGs don't capture well (e.g. cross-serial dependencies)

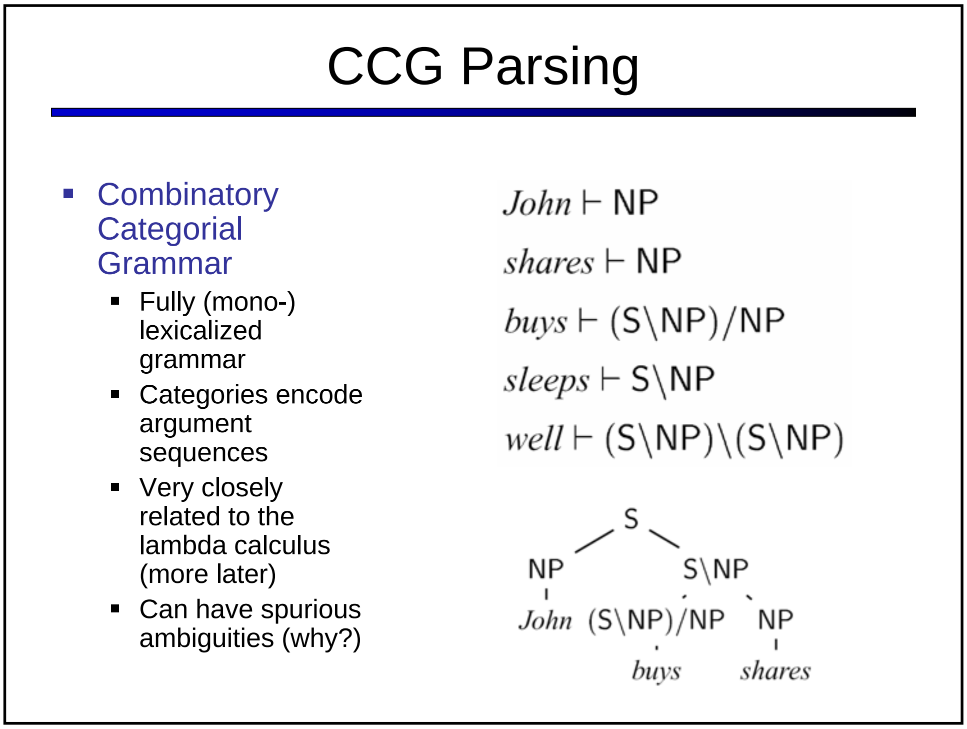


TAG: Adjunction



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?)

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

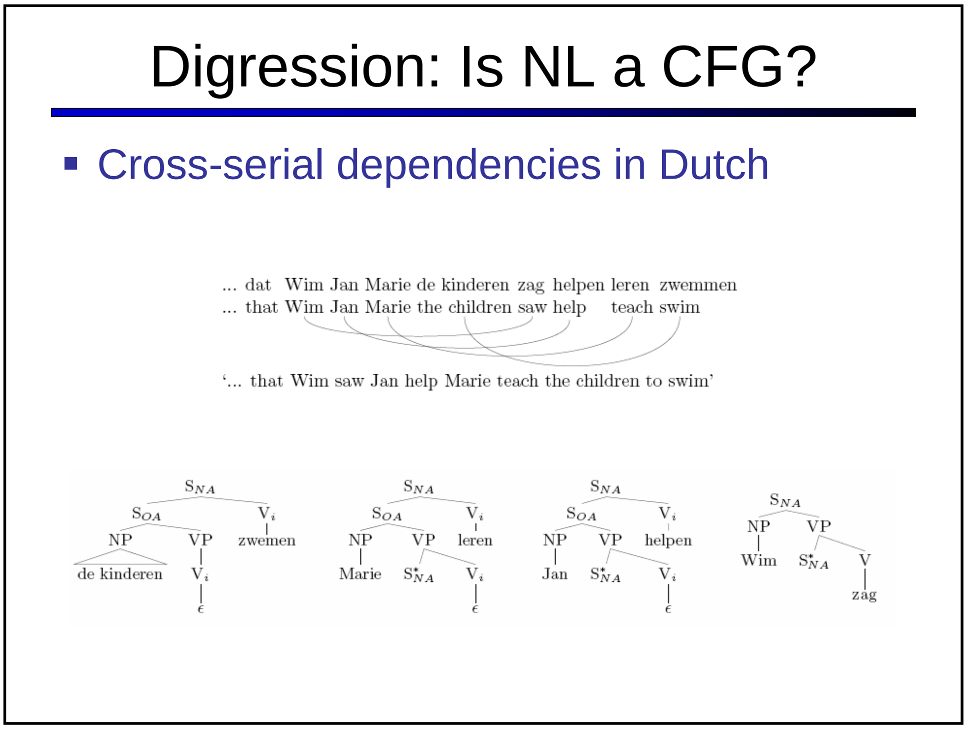
$\mathit{shares} \vdash \mathsf{NP}$

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

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

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

S
NP
S\NP
John
(S\NP)/NP
NP
buys
shares



Digression: Is NL a CFG?

- Cross-serial dependencies in Dutch

... dat Wim Jan Marie de kinderen zag helpen leren zwemmen
... that Wim Jan Marie the children saw help teach swim

'... that Wim saw Jan help Marie teach the children to swim'

S_{NA}
S_{OA}
V_i
NP
VP
zwemen
de kinderen
V_i
ϵ
S_{NA}
S_{OA}
V_i
NP
VP
leren
Marie
S^{*}_{NA}
V_i
ϵ
S_{NA}
S_{OA}
V_i
NP
VP
helpen
Jan
S^{*}_{NA}
V_i
ϵ
S_{NA}
NP
VP
Wim
S^{*}_{NA}
V
zag