Natural Language Dependency Parsing

SPFLODD

October 13, 2011

Quick Review from Tuesday

- What are CFGs? PCFGs? Weighted CFGs?
- Statistical parsing using the Penn Treebank
  - What it looks like
  - How to build a simple treebank (P)CFG
  - How to evaluate your treebank parser
- Weighted CFG parsing algorithms
  - CKY
  - Earley's algorithm
- Decorating the treebank: parent annotation, heads, lexicalization
- The Collins parser

Some Famous Parsers

(Review from Tuesday)

Collins Model 1 (1997)

- Trees are headed and lexicalized
  - What's the difference?

- Huge number of rules!

$VP_{saw} \rightarrow \underline{V}_{saw}\ NP_{man}\ PP_{through}$

$VP_{saw} \rightarrow \underline{V}_{saw}\ NP_{man}\ PP_{with}$

$VP_{saw} \rightarrow \underline{V}_{saw}\ NP_{woman}\ PP_{through}$

$VP_{saw} \rightarrow \underline{V}_{saw}\ NP_{man}$

- Key: factor probabilities within rule.

(REVIEW FROM TUESDAY)

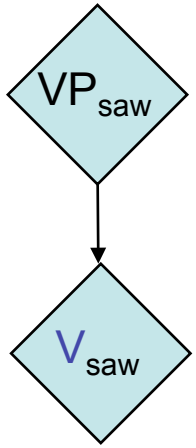
Collins Model 1 (1997)

- Everything factors down to rules, then further. We're given the parent nonterminal and head word.

(REVIEW FROM TUESDAY)

Collins Model 1 (1997)

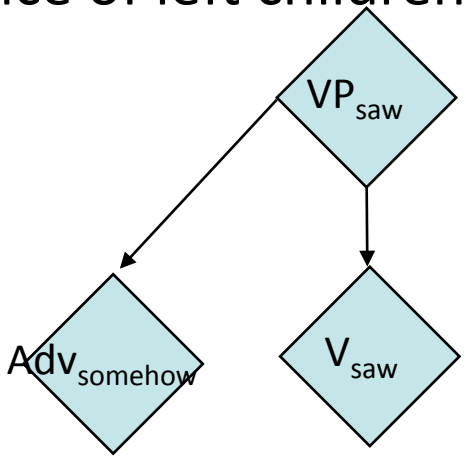
- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.



(REVIEW FROM TUESDAY)

Collins Model 1 (1997)

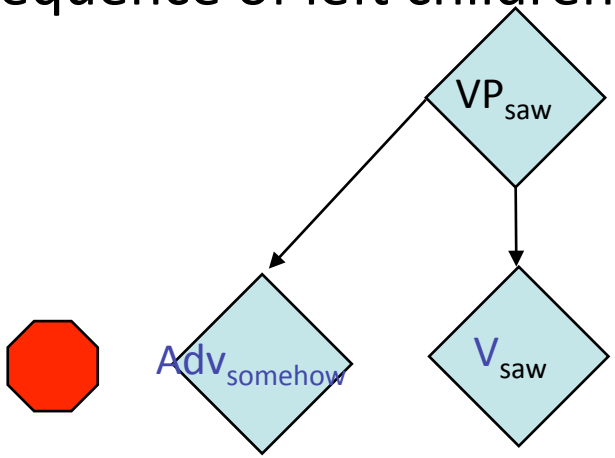
- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.
- Generate a sequence of left children.



(REVIEW FROM TUESDAY)

Collins Model 1 (1997)

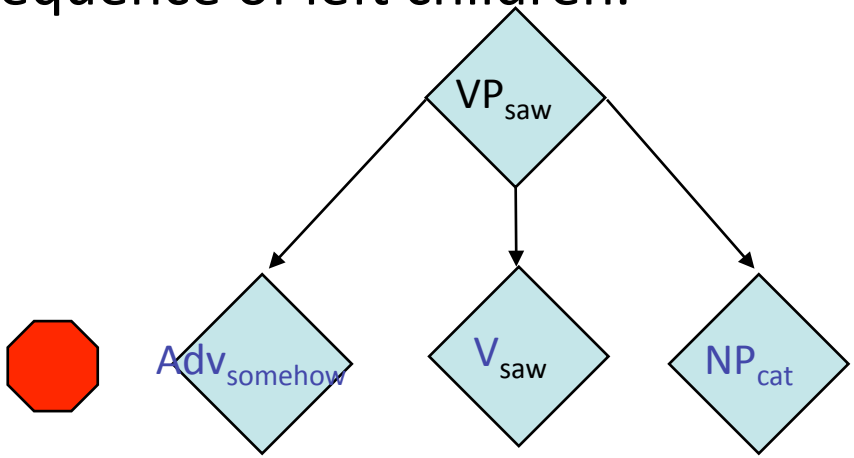
- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.
- Generate a sequence of left children.



(REVIEW FROM TUESDAY)

Collins Model 1 (1997)

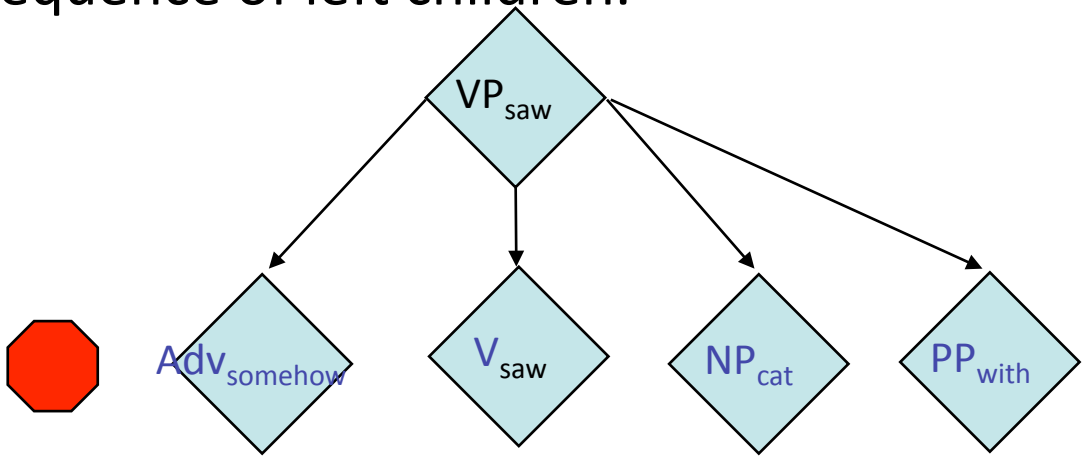
- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.
- Generate a sequence of left children.
- Then right.



(REVIEW FROM TUESDAY)

Collins Model 1 (1997)

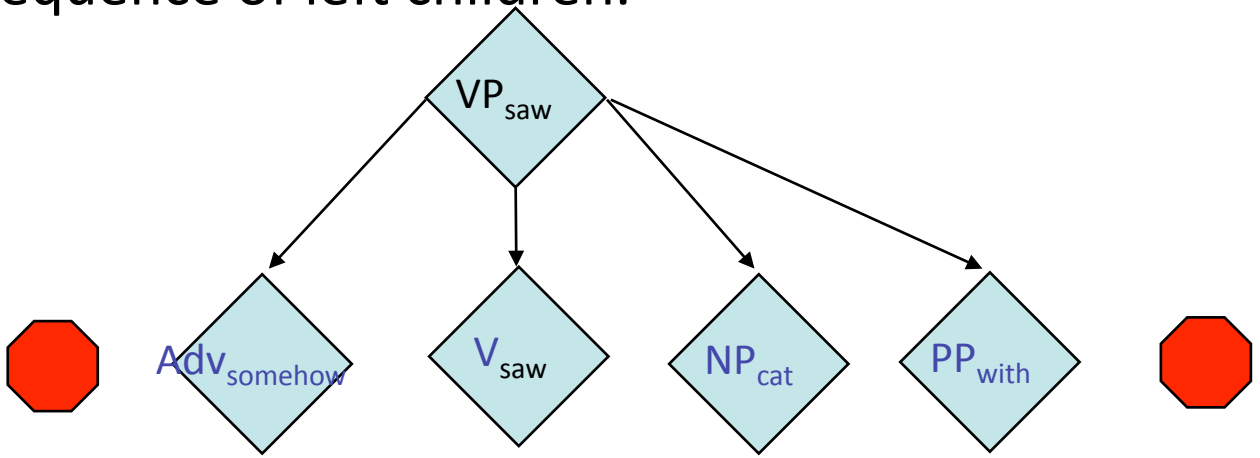
- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.
- Generate a sequence of left children.
- Then right.



(REVIEW FROM TUESDAY)

Collins Model 1 (1997)

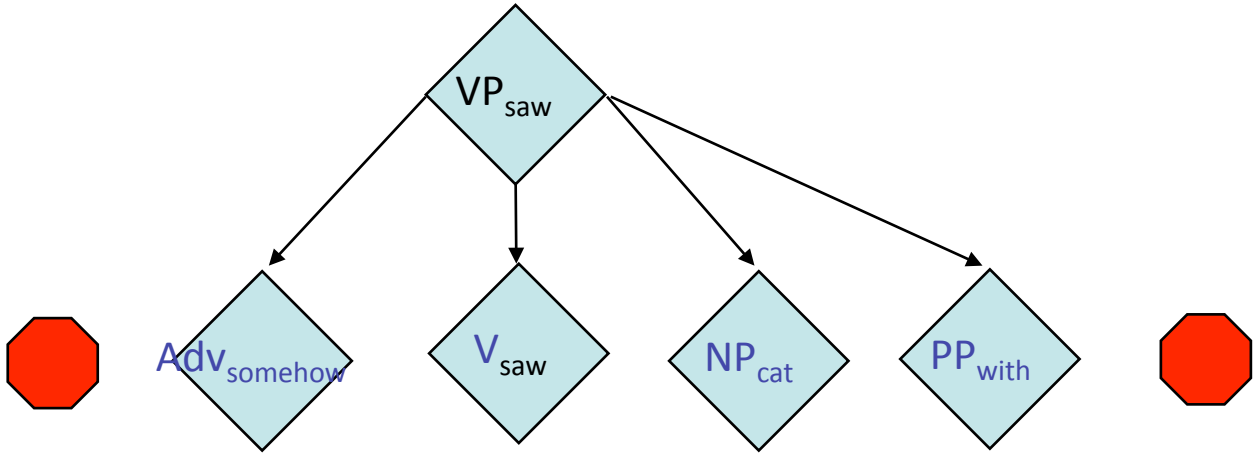
- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.
- Generate a sequence of left children.
- Then right.



(REVIEW FROM TUESDAY)

Collins Model 1 (1997)

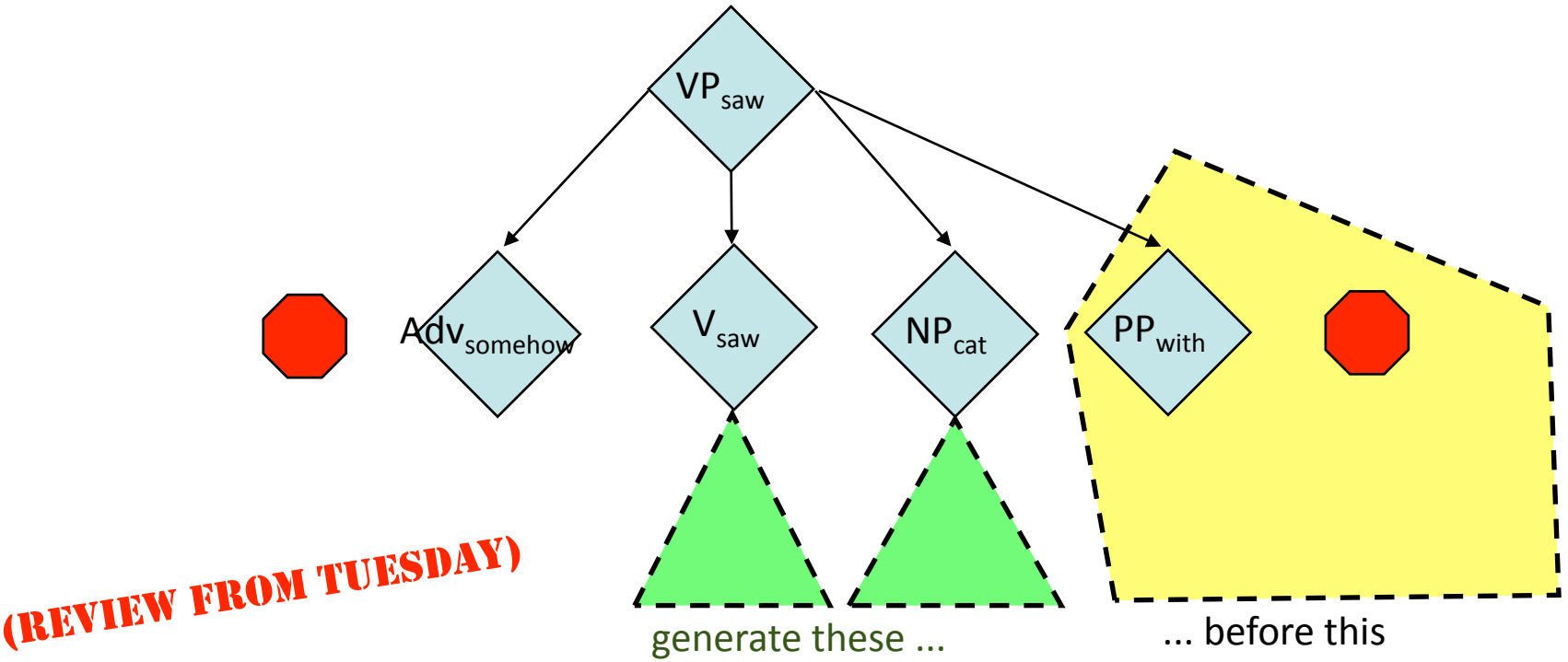
- Interesting twist: want to model the **distance** between head constituent and child constituent. How?



(REVIEW FROM TUESDAY)

Collins Model 1 (1997)

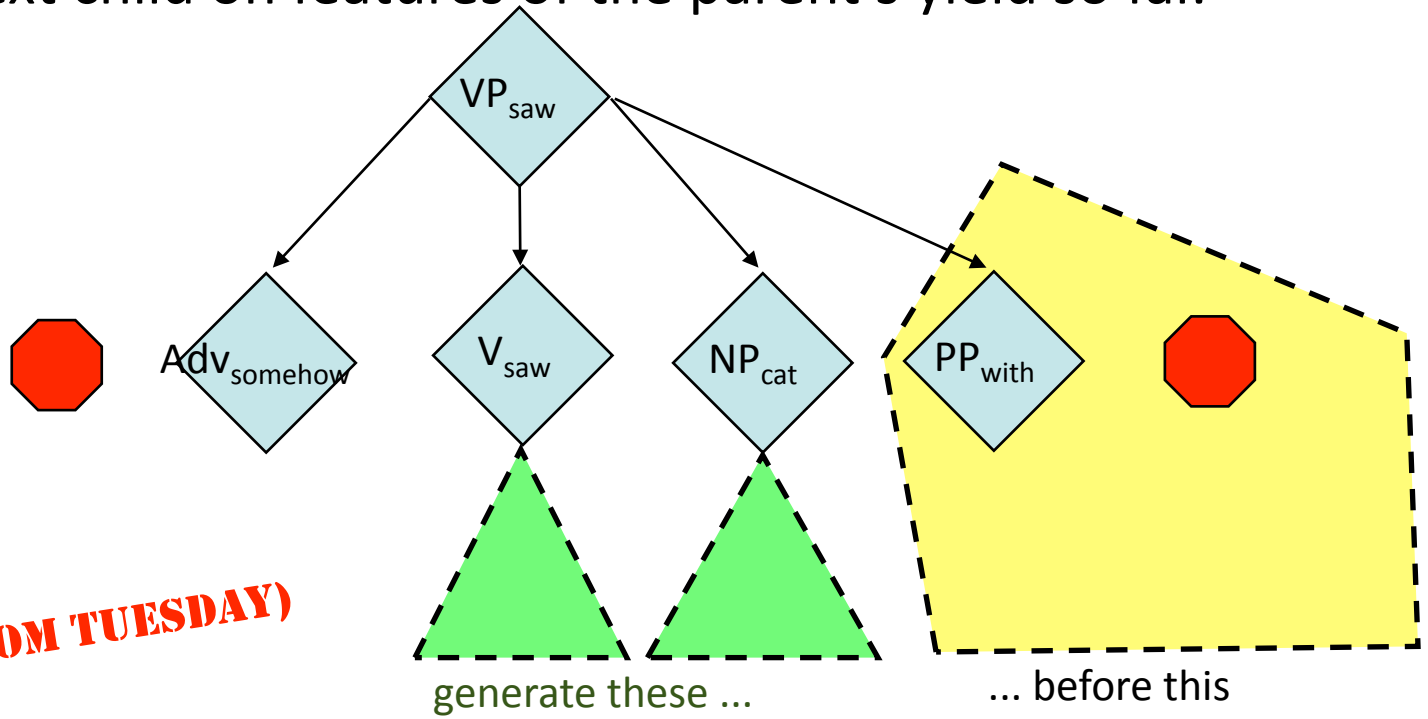
- Interesting twist: want to model the **distance** between head constituent and child constituent. How?
- Depth-first recursion.



(REVIEW FROM TUESDAY)

Collins Model 1 (1997)

- Interesting twist: want to model the **distance** between head constituent and child constituent. How?
- Depth-first recursion.
- Condition next child on features of the parent's yield so far.



(REVIEW FROM TUESDAY)

Collins Model 1 (1997)

- Interesting twist: want to model the **distance** between head constituent and child constituent. How?
- Depth-first recursion.
- Condition next child on features of the parent's yield so far.

$$p(\text{PP}_{\text{with}} \mid \text{VP}_{\text{saw}}, \text{right}, \text{“the cat who liked milk”}) \approx p(\text{PP}_{\text{with}} \mid \text{VP}_{\text{saw}}, \text{right}, \text{length} > 0, +\text{verb})$$

$$\begin{aligned} & p(L_n, u_n, L_{n-1}, u_{n-1}, ..., L_1, u_1, H, w, R_1, v_1, R_2, v_2, ..., R_m, v_m \mid P, w) \\ = \; & p(H \mid P, w) \\ & \cdot \prod_{i=1}^{n} p(L_i, u_i \mid P, w, H, \text{left}, \Delta_i) \\ & \cdot p(\text{stop} \mid P, w, H, \text{left}, \Delta_{n+1}) \\ & \cdot \prod_{i=1}^{m} p(R_i, v_i \mid P, w, H, \text{right}, \Delta'_i) \\ & \cdot p(\text{stop} \mid P, w, H, \text{right}, \Delta'_{n+1}) \end{aligned}$$

(REVIEW FROM TUESDAY)

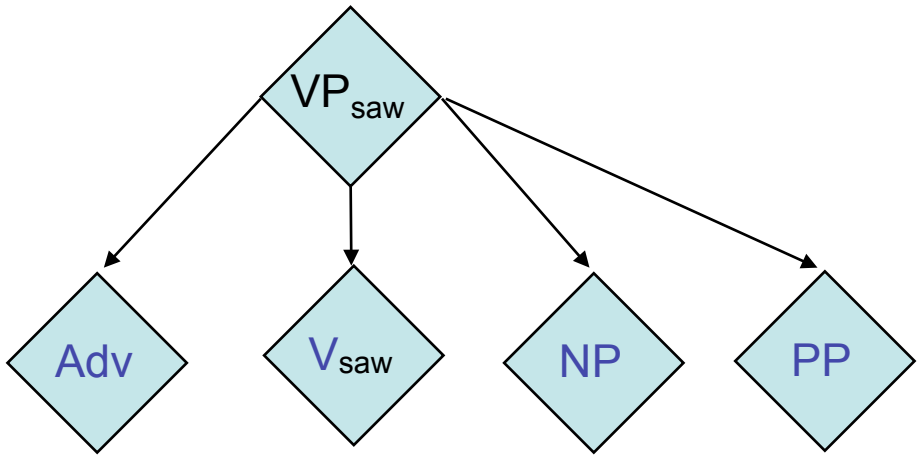
Charniak (1997) - in brief

- Generally similar to Collins
- Key differences:
  - Used an additional 30 million words of unparsed text in training
  - Rules not fully markovized: pick full nonterminal sequence, then lexicalize each child independently

Charniak (1997) - in brief

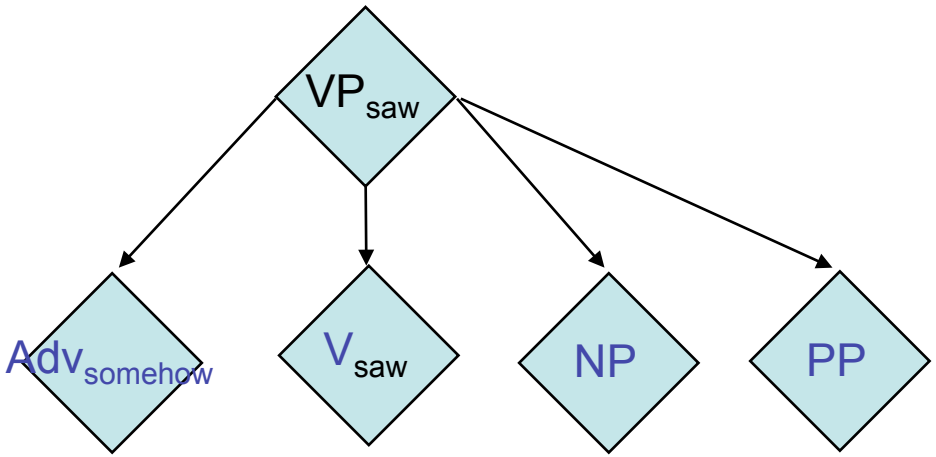
Charniak (1997) - in brief

$VP_{saw} \rightarrow Adv \underline{V} NP PP$



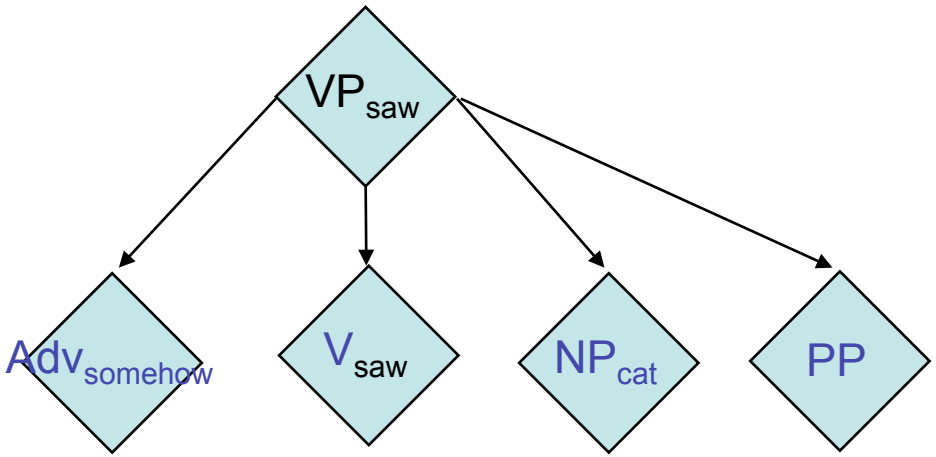
Charniak (1997) - in brief

p(somehow | VP_{saw}, Adv)



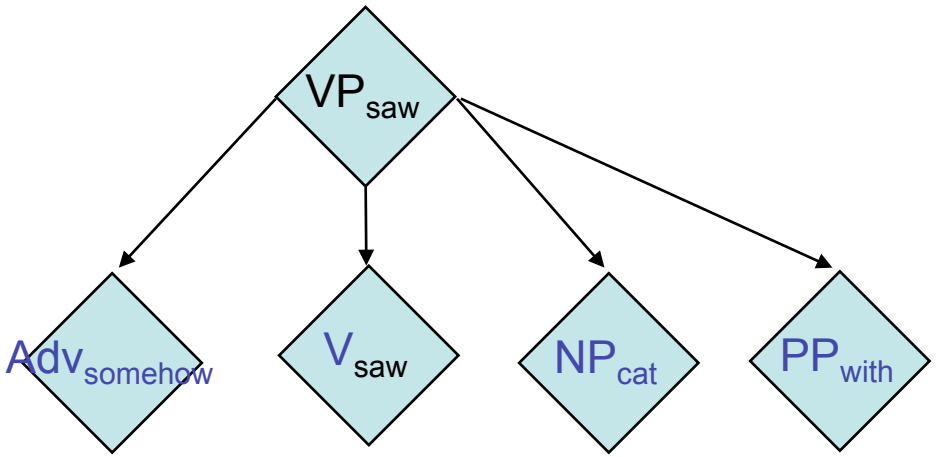
Charniak (1997) - in brief

$p(\text{cat} \mid VP_{saw}, NP)$



Charniak (1997) - in brief

p(with | VP_{saw}, PP)



Charniak (2000)

- Uses grandparents (Johnson ’98 transformation)
- Markovized children (like Collins)
- Bizarre probability model:
  - Smoothed estimates at many backoff levels
  - Multiply them together
  - “Maximum entropy inspired”
  - Kind of a product of experts (untrained)

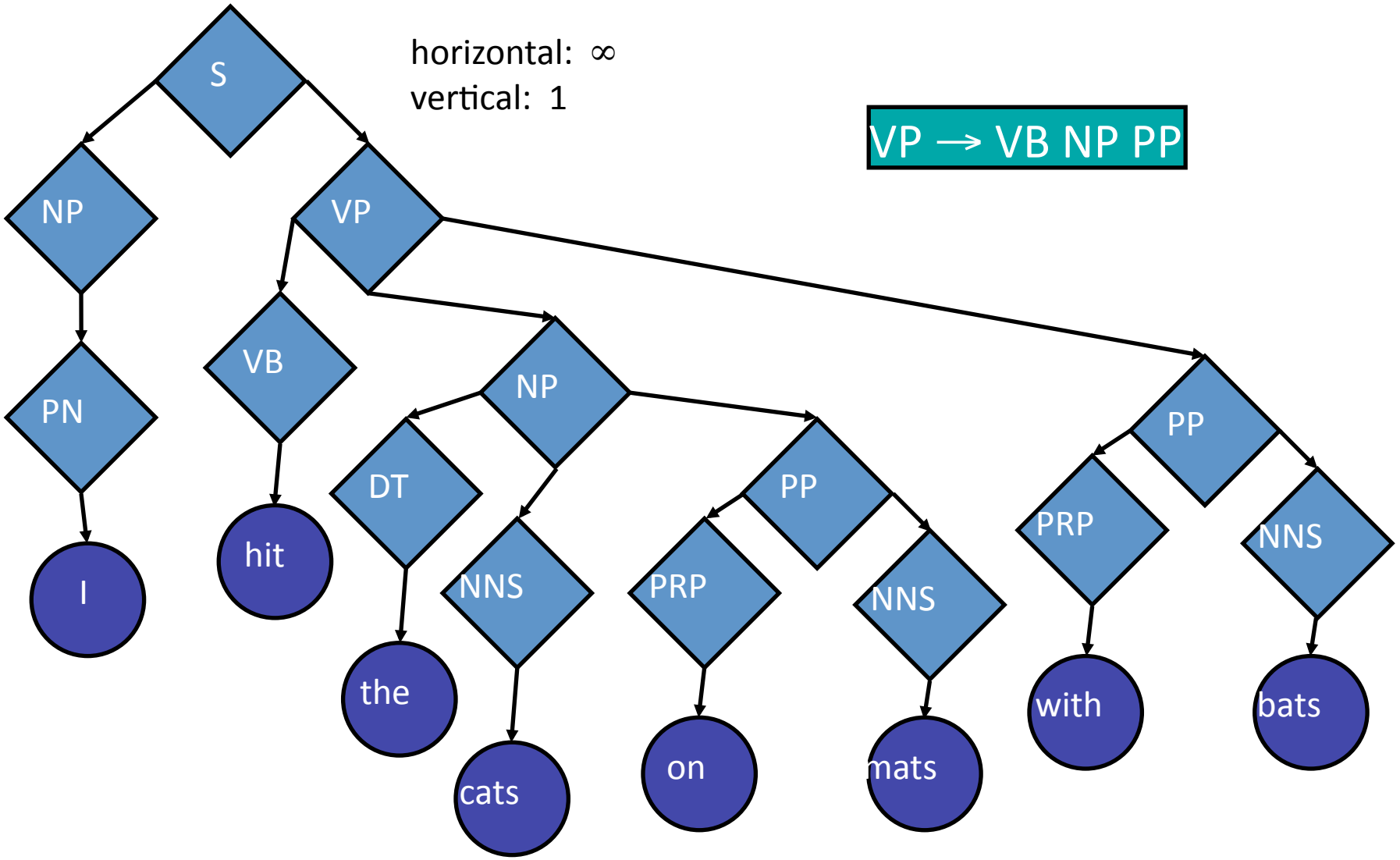
Comparison

		labeled recall	labeled precision	average crossing brackets
Collins	Model 1	87.5	87.7	1.09
	Model 2	88.1	88.3	1.06
	Model 3	88.0	88.3	1.05
Charniak	1997	86.7	86.6	1.20
	2000	89.6	89.5	0.88

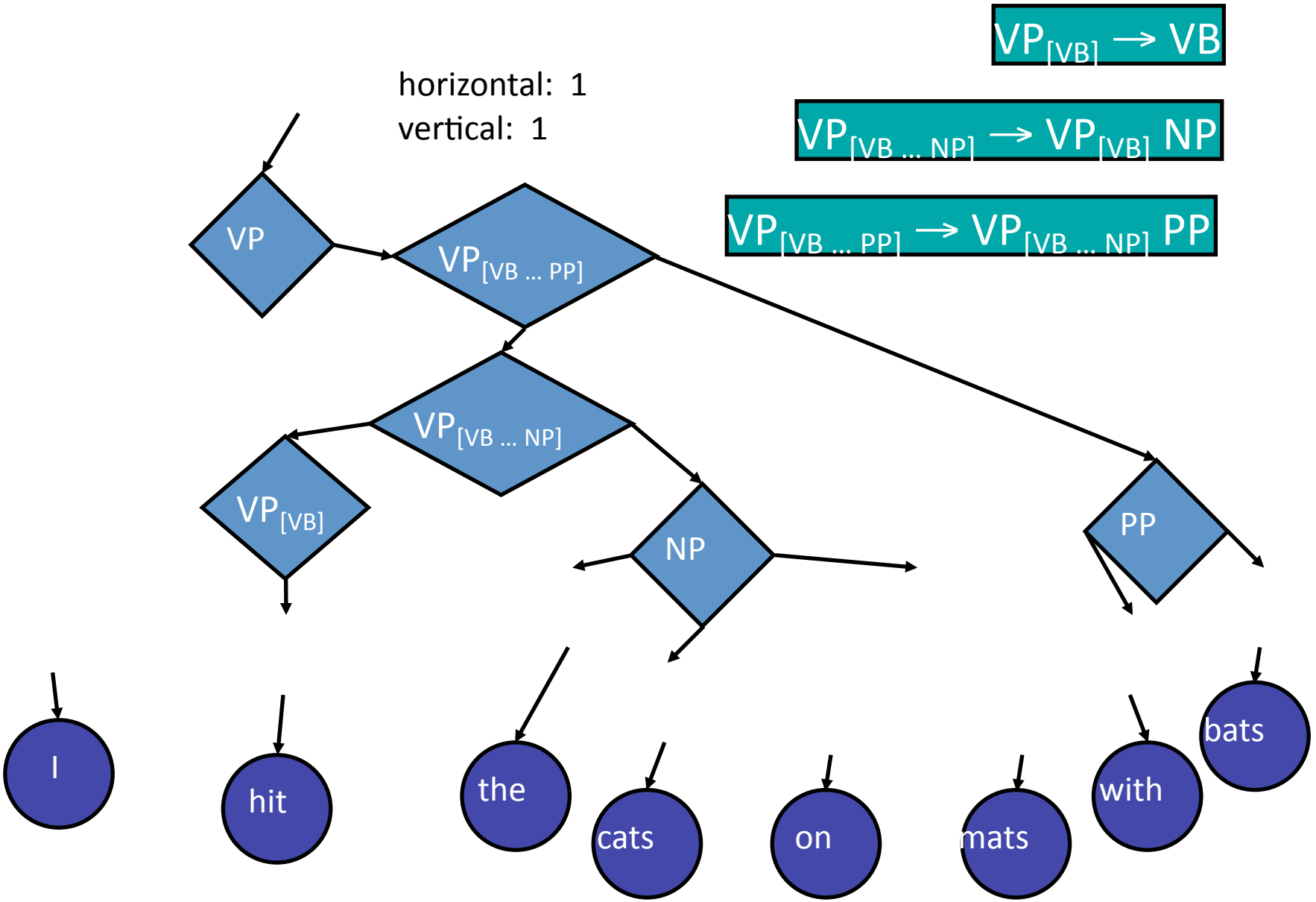
Klein and Manning (2003)

- By now, lexicalization was kind of controversial
  - So many probabilities, such expensive parsing: is it necessary?
- Goal: reasonable unlexicalized baseline
  - What tree transformations make sense?
  - Markovization (what order?)
  - Add all kinds of information to each node in the treebank
- Performance close to Collins model, much better than earlier unlexicalized models

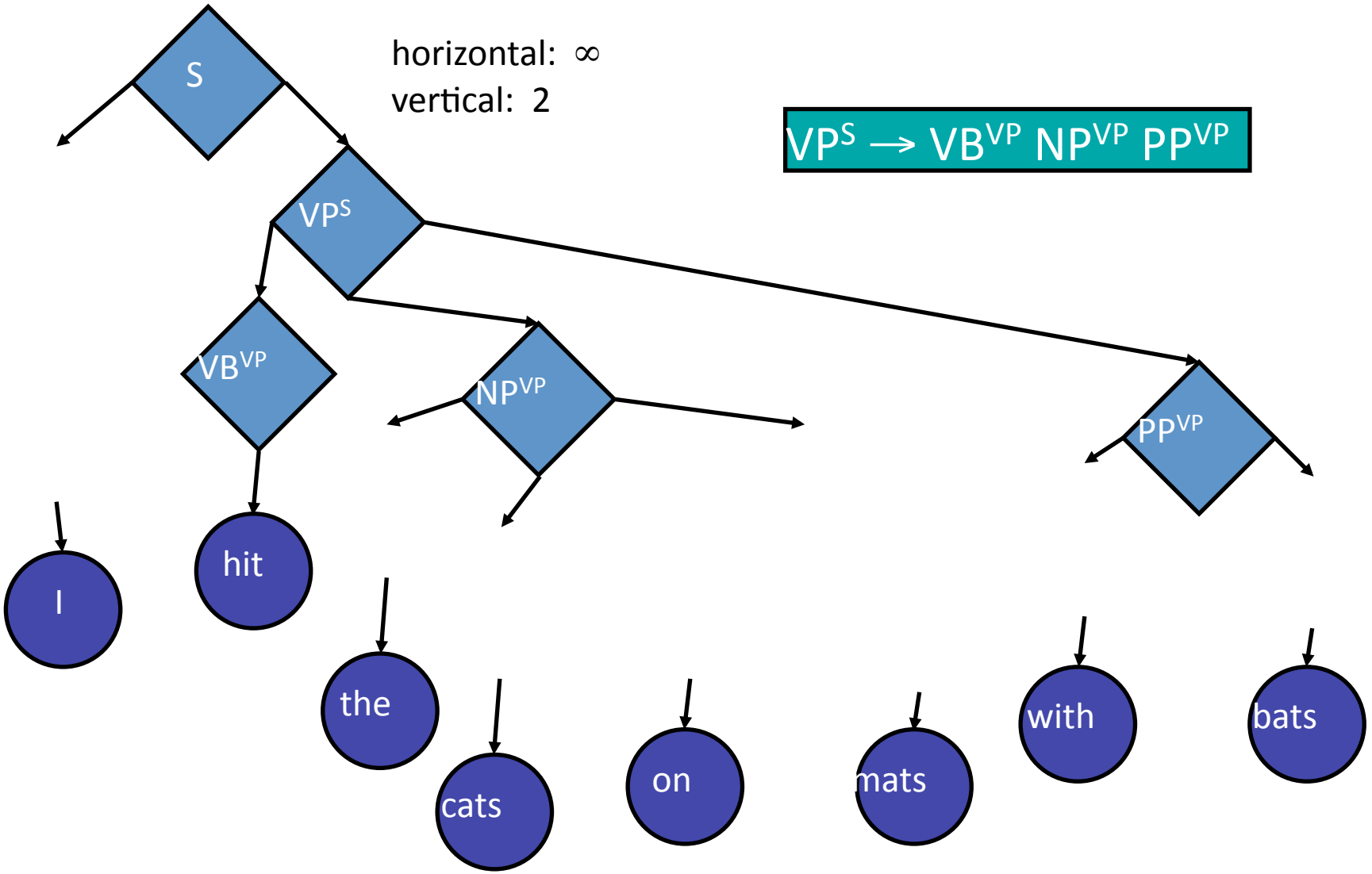
Markovization



Markovization



Markovization



Markovization

- More vertical Markovization is better
  - Consistent with Johnson (1998)
- Horizontal 1 or 2 beats 0 or ∞
- Used (2, 2), but if sparse "back off" to 1

Other Tree Decorations

- Mark nodes with only 1 child as UNARY
- Mark DTs (determiners), RBs (adverbs) when they are only children
- Annotate POS tags with their parents
- Split IN (prepositions; 6 ways), AUX, CC, %
- NPs: temporal, possessive, base
- VPs annotated with head tag (finite vs. others)
- DOMINATES-V
- RIGHT-RECURSIVE NP

Comparison

		labeled recall	labeled precision	average crossing brackets
Collins	Model 1	87.5	87.7	1.09
	Model 2	88.1	88.3	1.06
	Model 3	88.0	88.3	1.05
Charniak	1997	86.7	86.6	1.20
	2000	89.6	89.5	0.88
K&M	2003	86.3	85.1	1.31

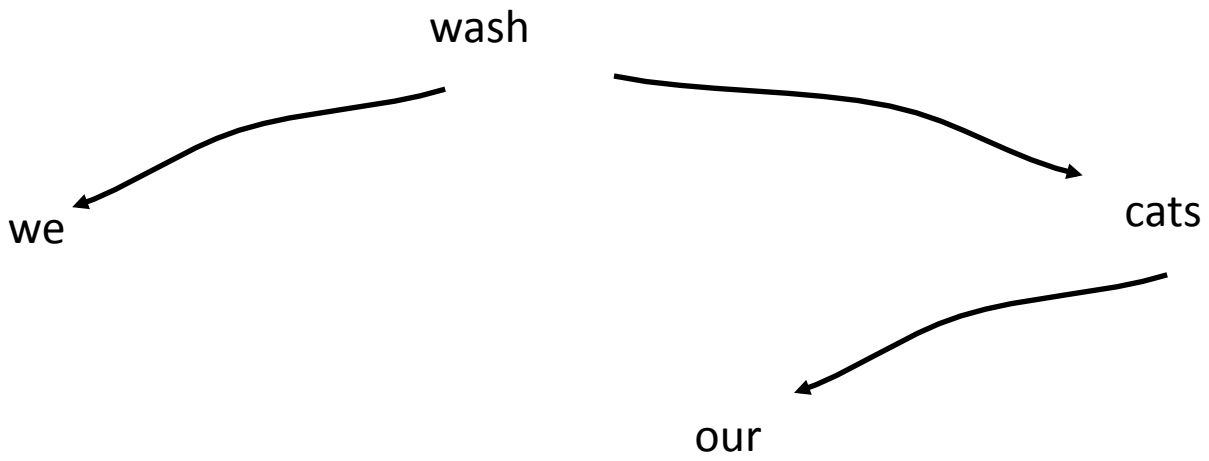
Dependency Parsing

Dependency Grammar

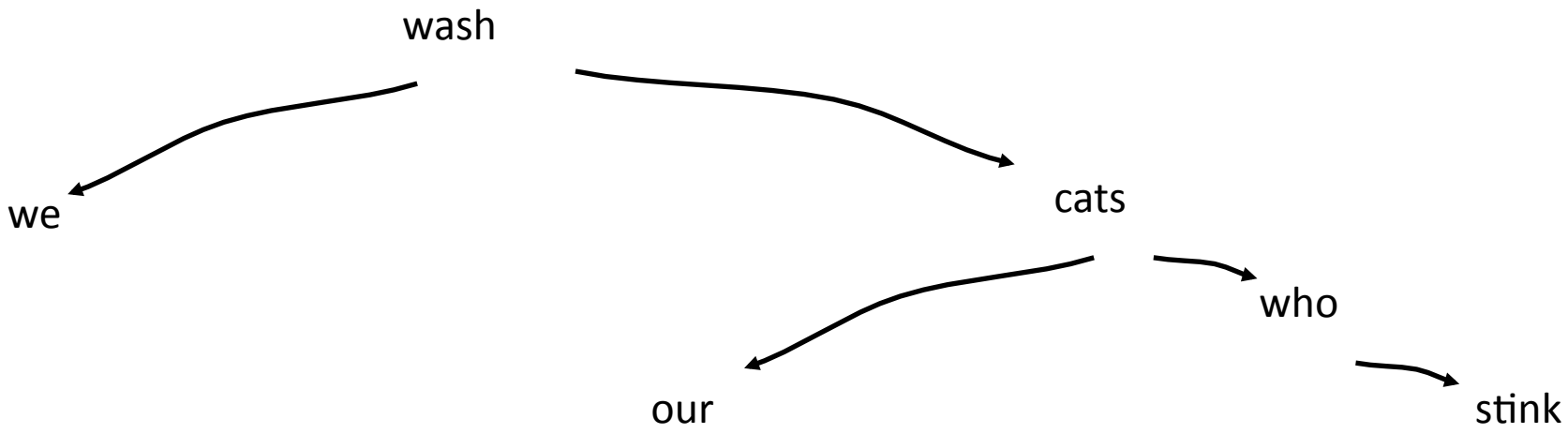
- A variety of theories and formalisms
- Focus on relationship between **words** and their syntactic relationships
- Correlates with study of languages that have free(r) word order (e.g., Czech)
- Lexicalization is central, phrases secondary

- We will talk about **bare bones** dependency trees (Eisner, 1996), then consider adding dependency labels

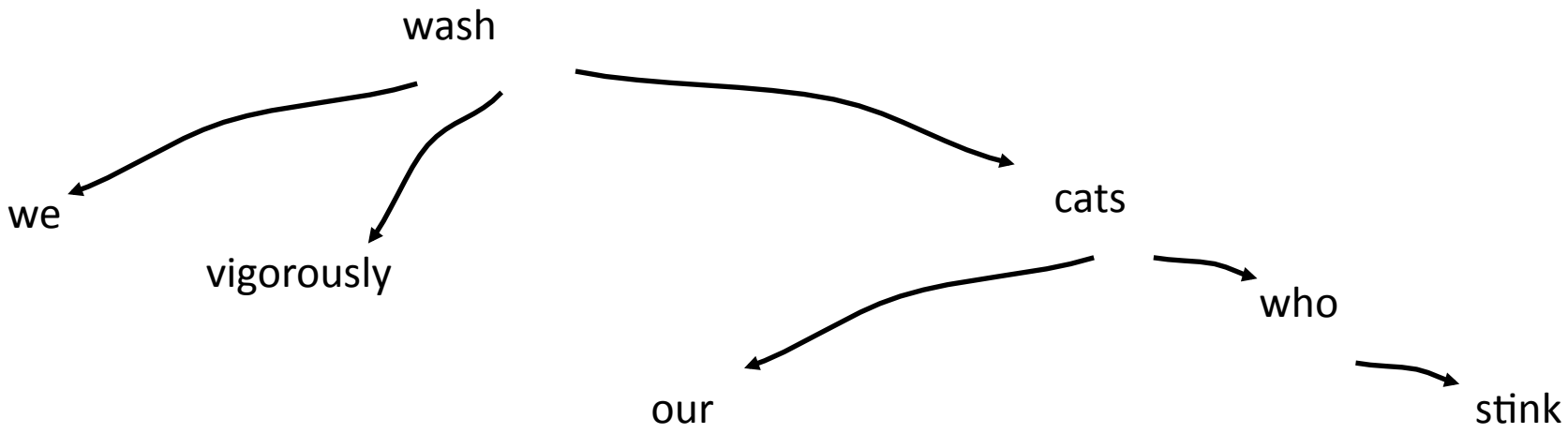
Bare Bones Dependency Parse



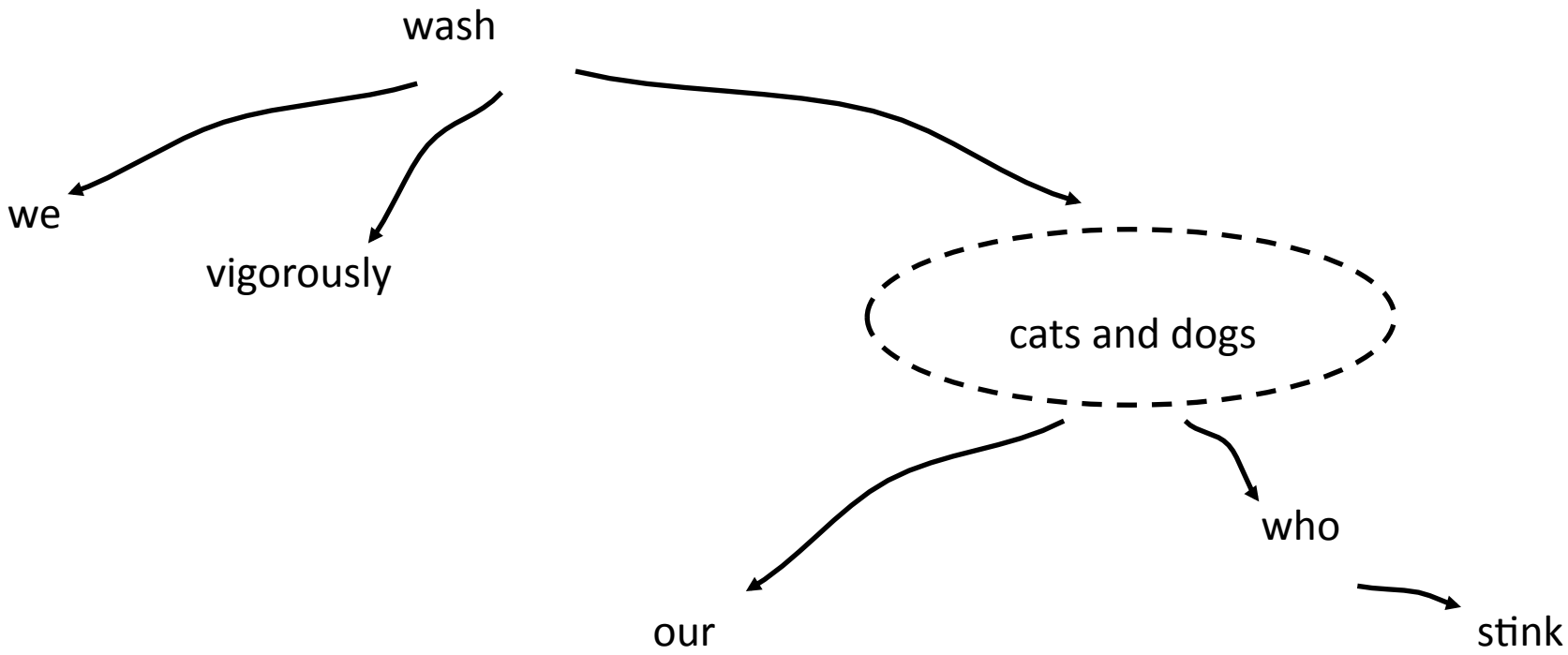
Bare Bones Dependency Parse



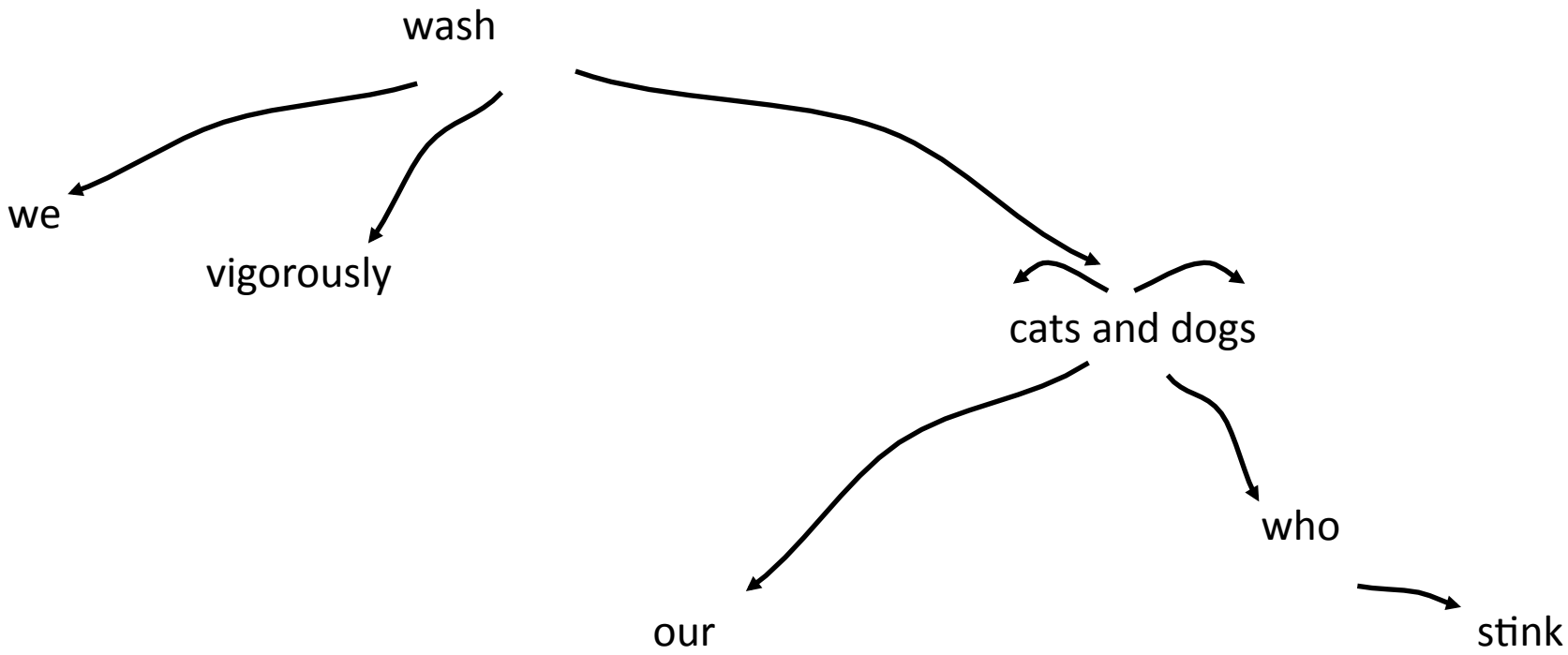
Bare Bones Dependency Parse



Bare Bones Dependency Parse



Bare Bones Dependency Parse



Bare Bones Dependencies and Labels

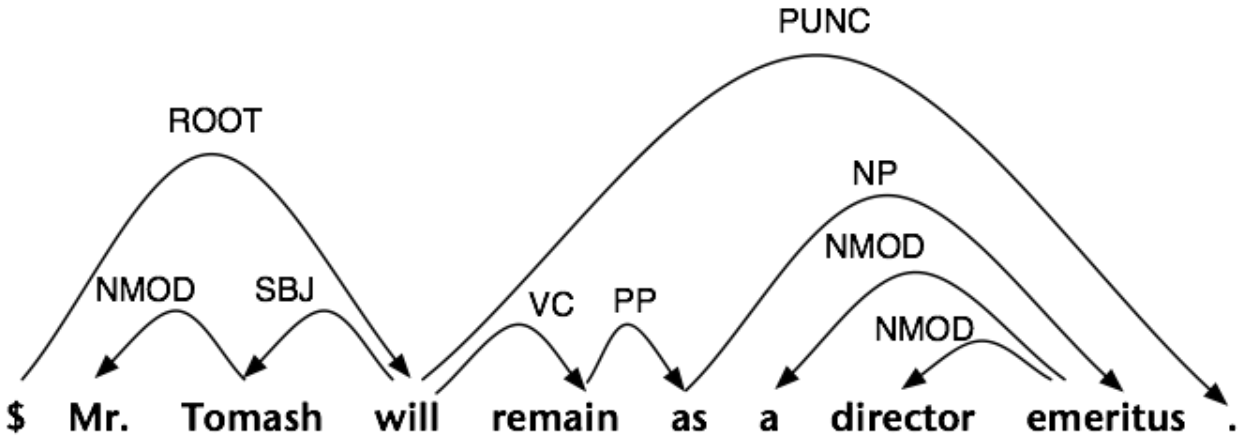
- The way to represent a lot of phenomena is clear (predicate-argument and modifier relationships)
- Conjunctions pose a problem
- Sometimes words that "should" be connected are not, because of the single-parent rule
- From bare bones to **labels**:
  - consider labeled edges
  - most algorithms can be easily extended for labeled dependency parsing
- Linguistically imperfect, but computationally attractive

Evaluation

- Attachment accuracy
  - Labeled
  - Unlabeled

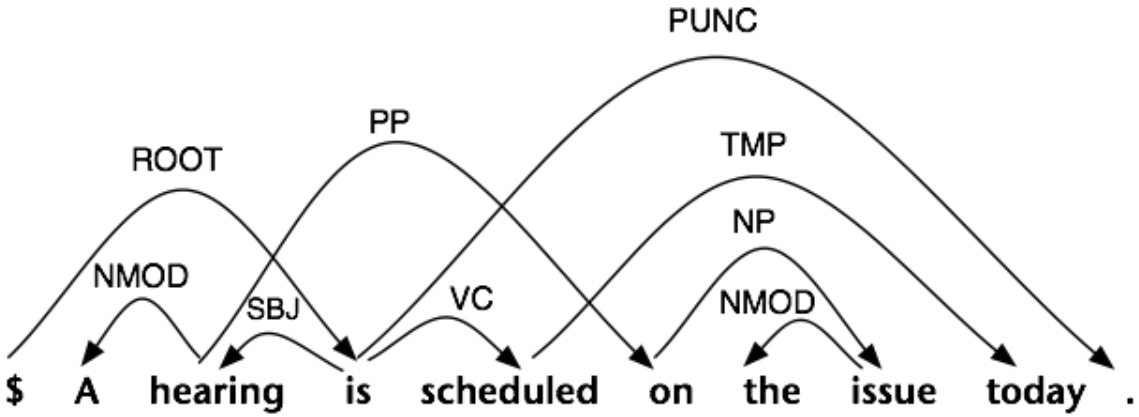
Dependencies and Context-Freeness

- **Projective** dependency trees are ones where edges don't cross
- Projective dependency parsing means searching only for projective trees
- English is mostly projective...



Dependencies and Context-Freeness

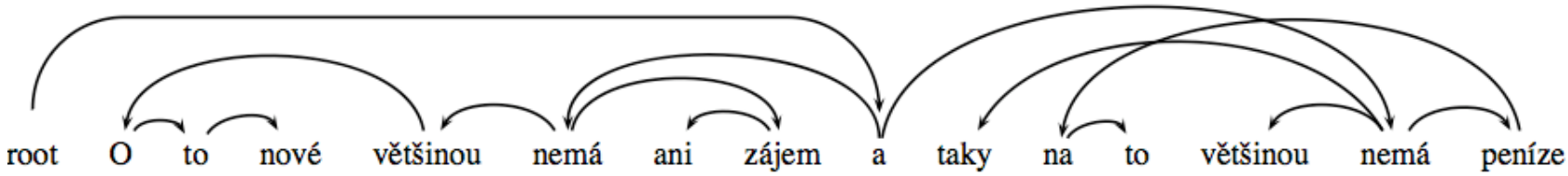
- But not entirely!



- Dependencies constructed through simple means from the Penn treebank will be projective. Why?

Dependencies and Context-Freeness

- Other languages are arguably less projective



- **Projective** dependency grammars generate **context-free** languages
- **Non-projective** dependency grammars can generate **context-sensitive** languages

Projective Dependency Parsing

- Major assumption: edge-factored model

$$p(\tau, w_1^n) \propto \prod_{i=1}^{n} \phi(w_1^n, \tau(w_i))$$

- Carroll and Charniak (1992) described a PCFG that has this property
- Eisner (1996) described several stochastic models for generating projective trees like this
- You should see that this is a linear model with a certain kind of feature locality
- We're not going to go into the details of the features that have been proposed!

Graph-based vs. Transition-based

- All models above optimize a global score and resort to local features
  - These are sometimes known as **graph-based models.**
- Just like in the phrase-structure/constituent world, there are also approaches that use shift-reduce algorithms.
- With good statistical learning methods, you can get very high performance using **greedy** search without back-tracking!
- "Local decisions, global features"
  - These are known as **transition-based models**; they reduce a structured problem to a lot of classification decisions, kind of like MEMMs.
  - See work by J. Nivre.

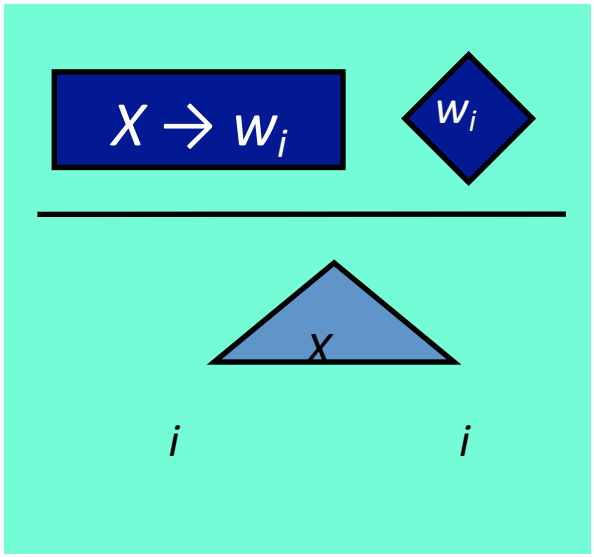
Algorithms != Models

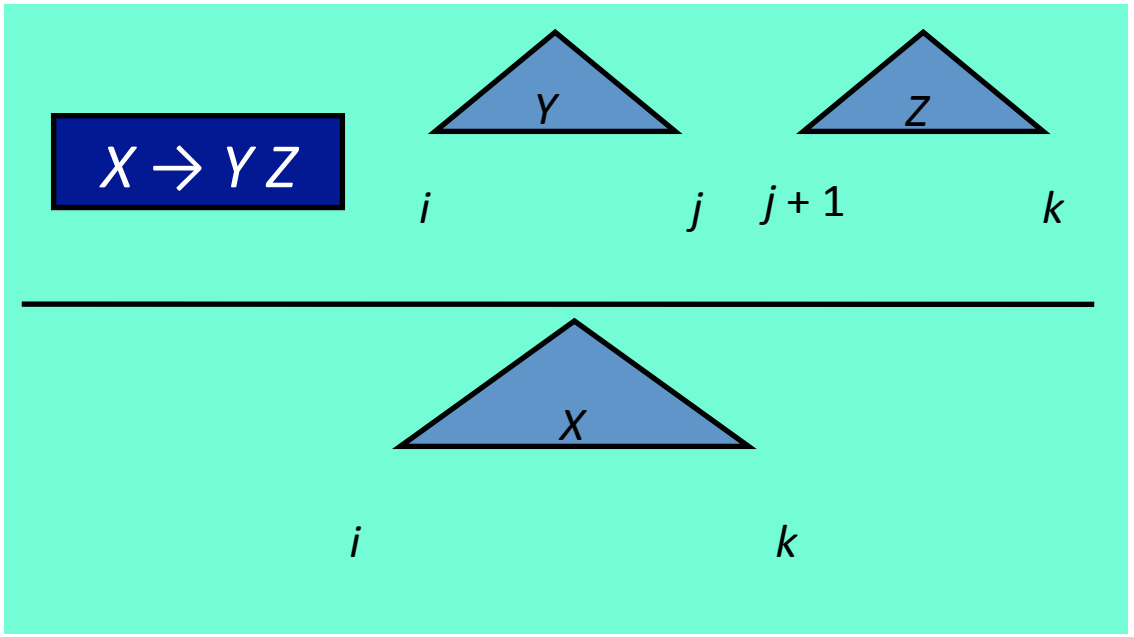
- As in HMMs, PCFGs, etc., the algorithms we need depend on the independence assumptions, **not** on the specific formulation of the statistical scores.
- We assume, from here on, that the scores are factored by dependency tree edges.

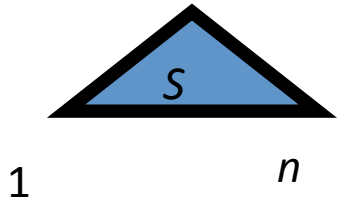
$$s(\tau) = \sum_{i=1}^{n} s(w_1^n, \tau(w_i))$$

- Projective algorithm (Eisner, 1996)
- Non-projective algorithm (McDonald et al., 2005)

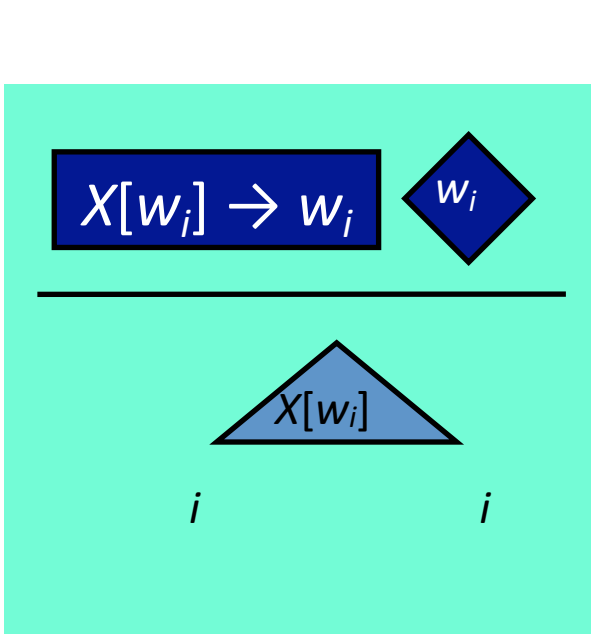
CKY

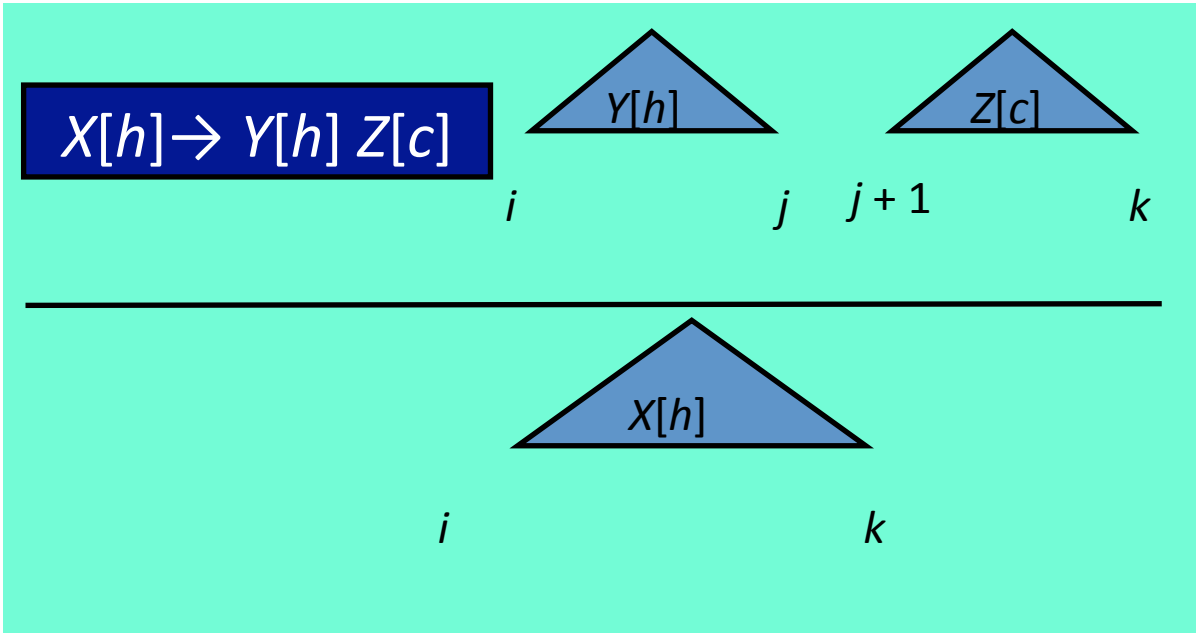


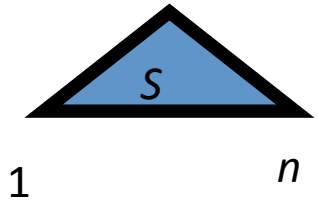




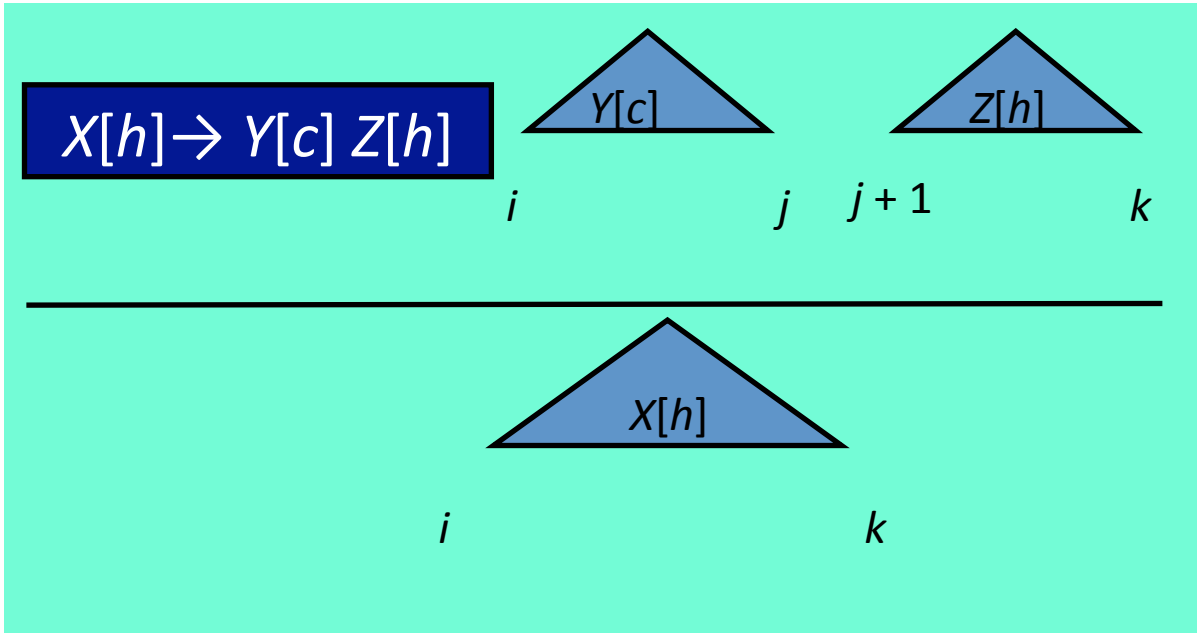
CKY with Heads



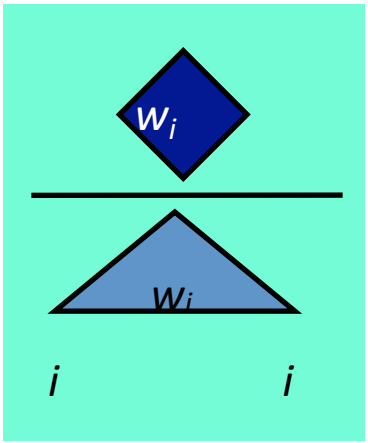


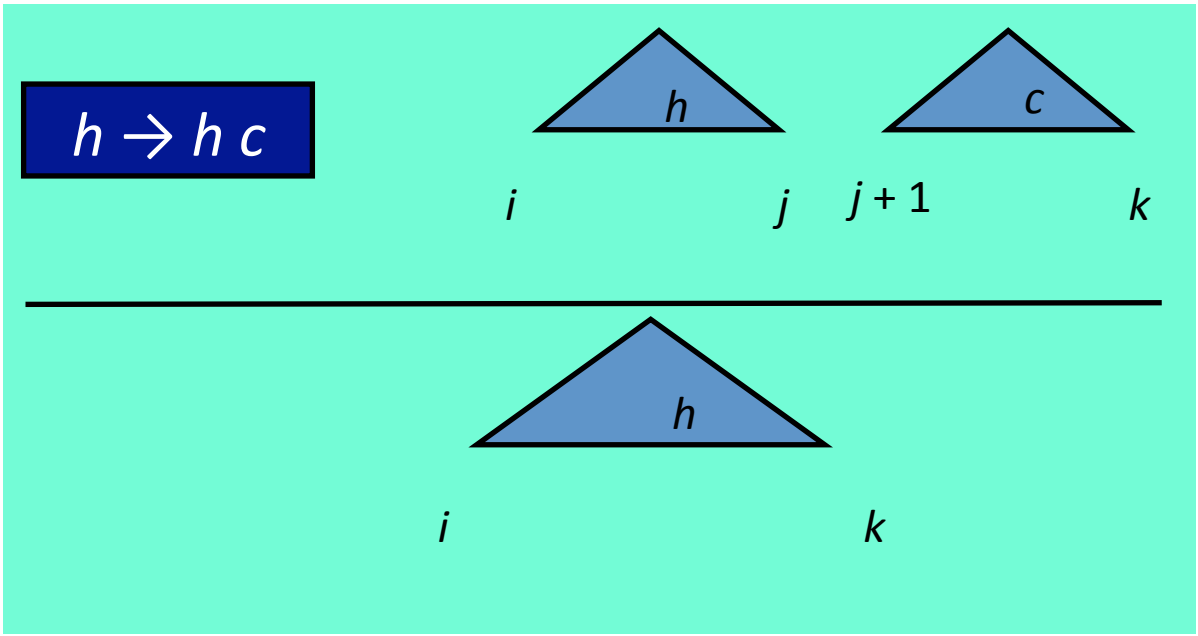


CKY with Heads (one more rule)



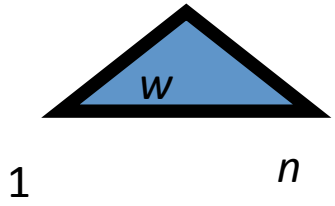
CKY with Heads, without Nonterminals





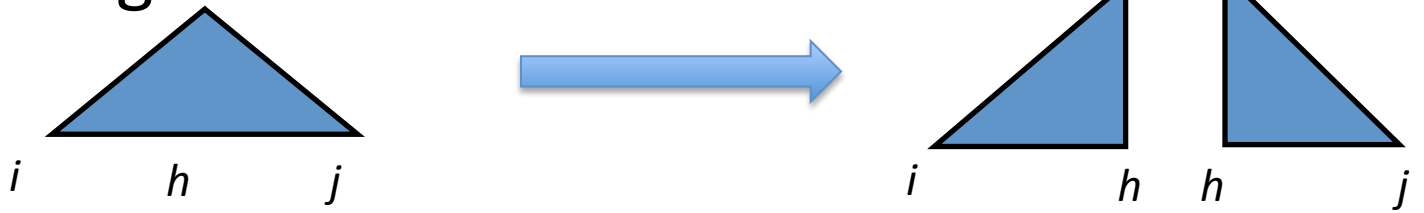
*Plus the rule for $h \rightarrow c\ h$.

What's the runtime?

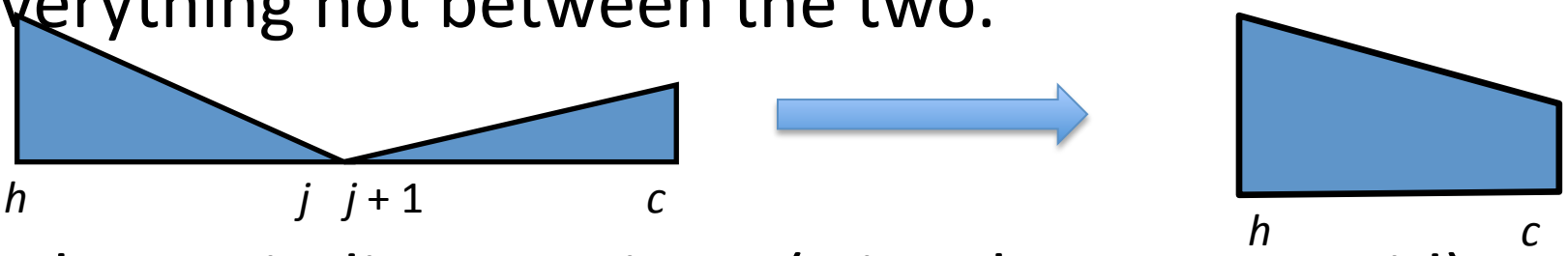


Eisner's (1996) Algorithm

- Restructure the computation so that "triangles" are organized around heads.

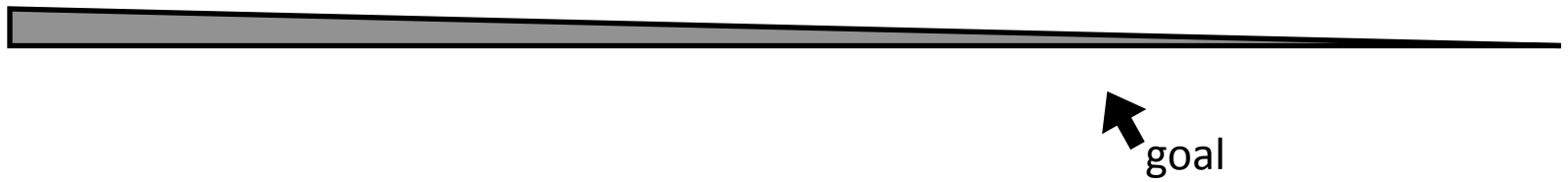


- "Half" constituents get put together from head outward; attaching a child to a parent ignores everything not between the two.



- Only two indices per item (triangle or trapezoid).

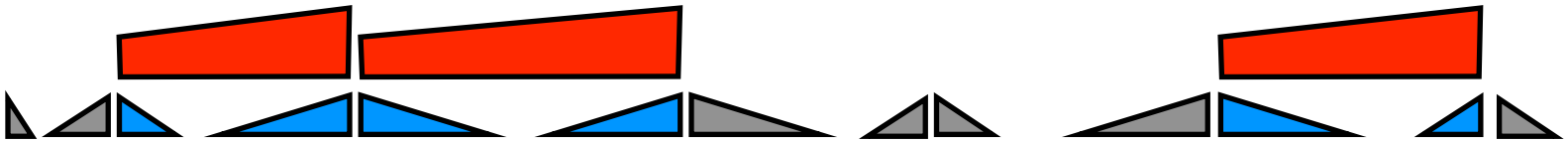
Example of the Eisner Algorithm



$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

Attach:

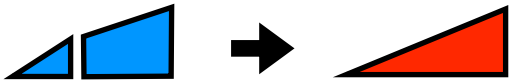


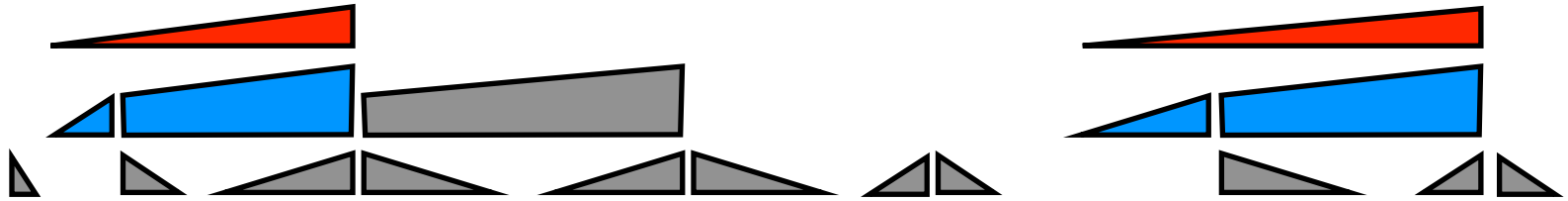
$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

Example of the Eisner Algorithm

Complete:

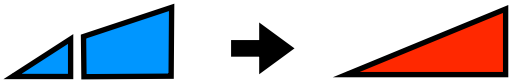


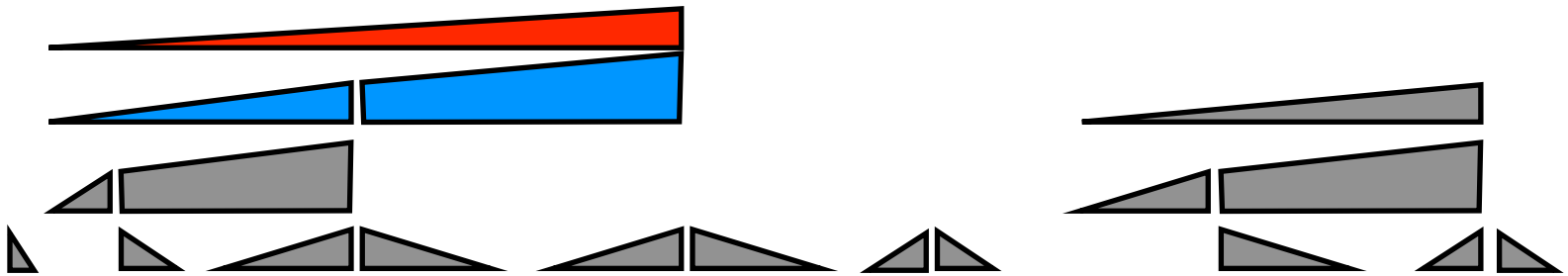


$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

Complete:

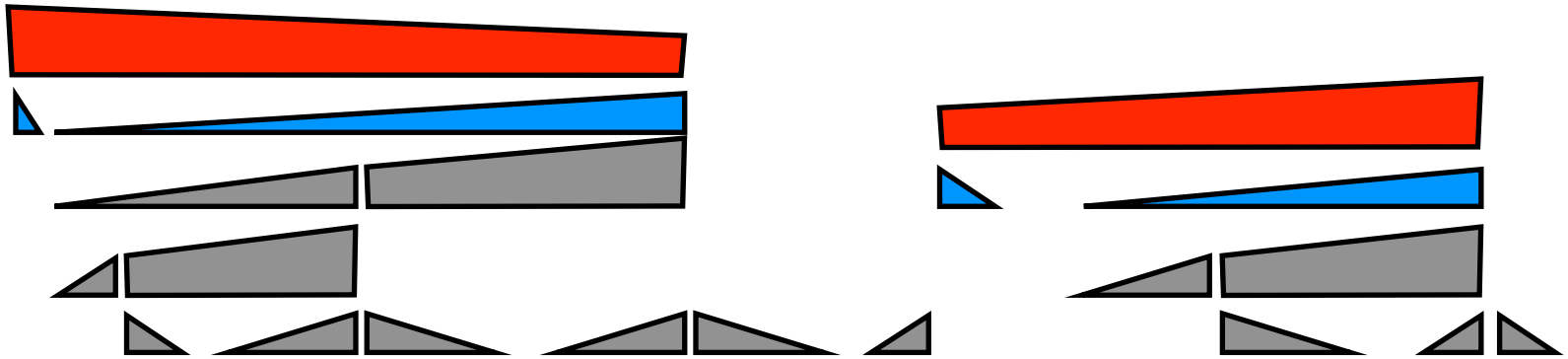




$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

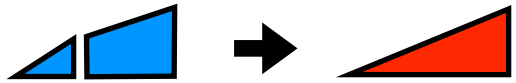
Attach:

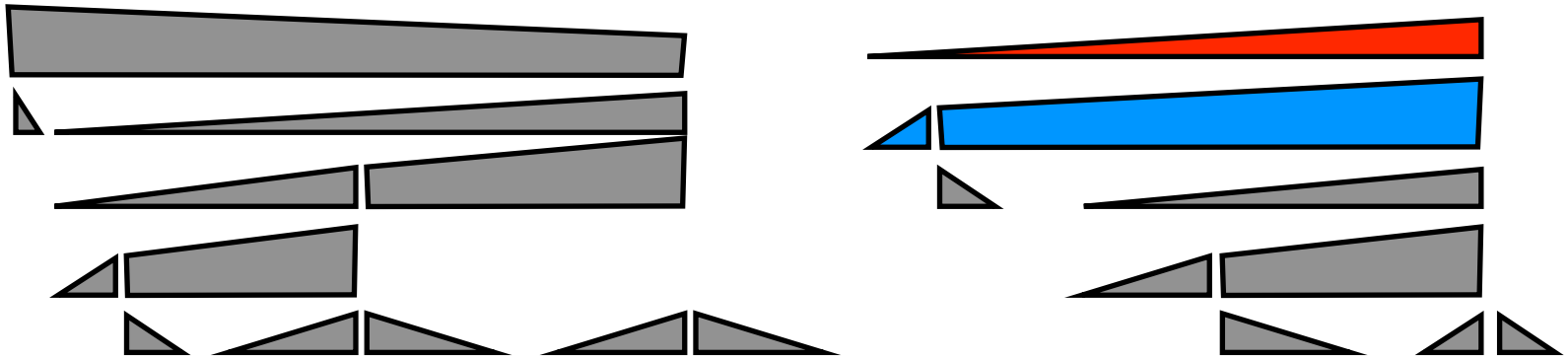


$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

Complete:

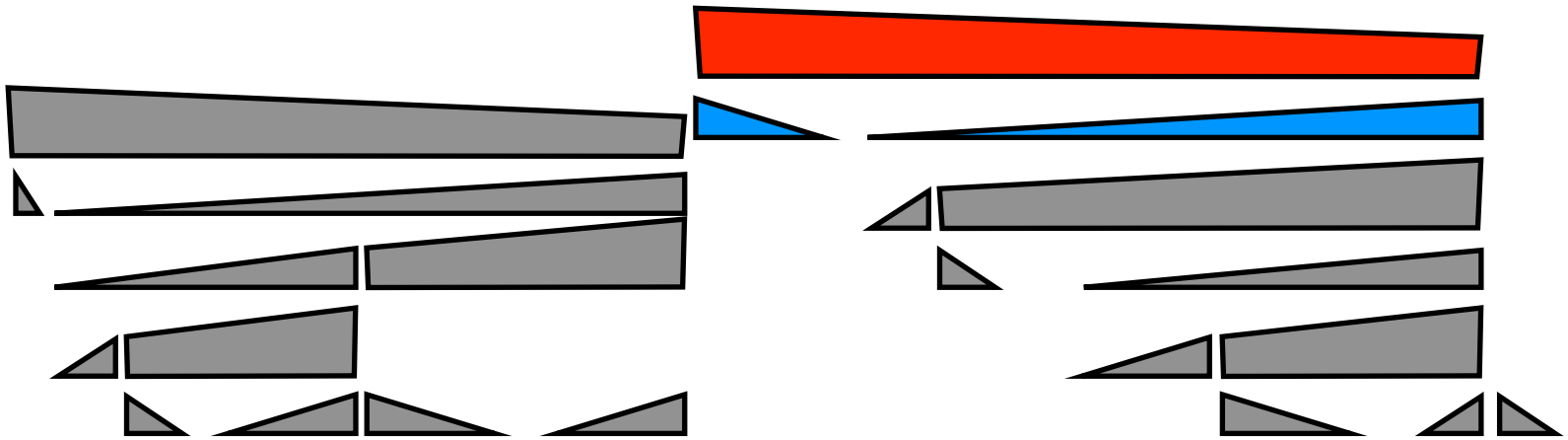




$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

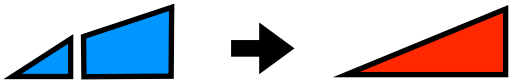
Attach:

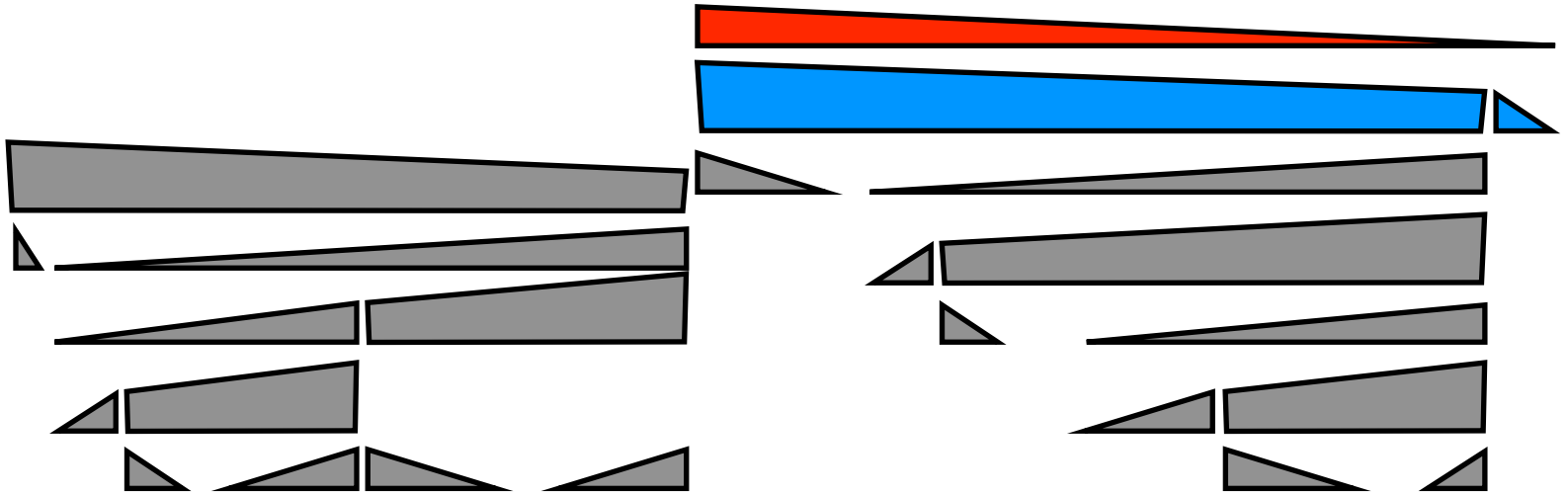


$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

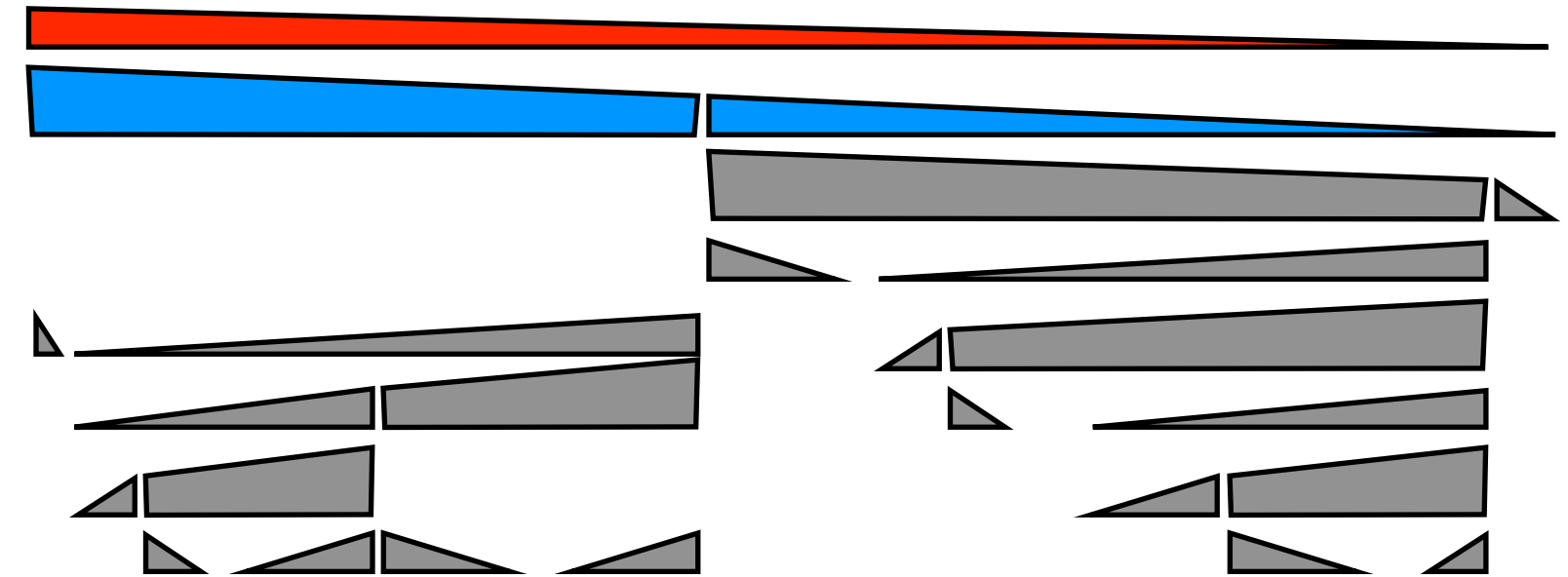
Complete:





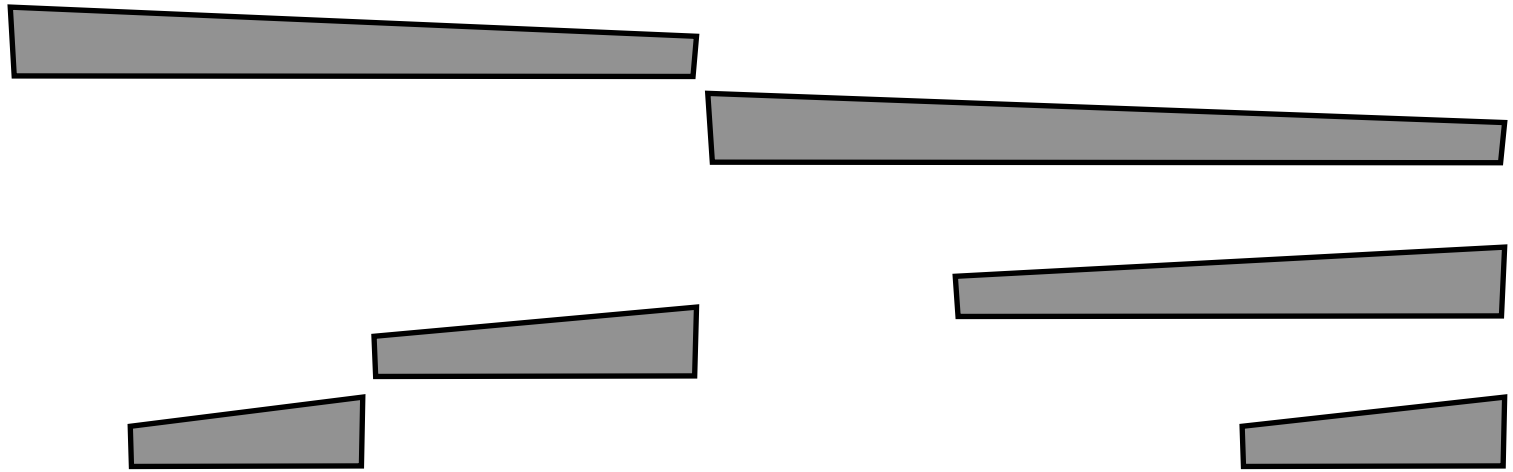
$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm



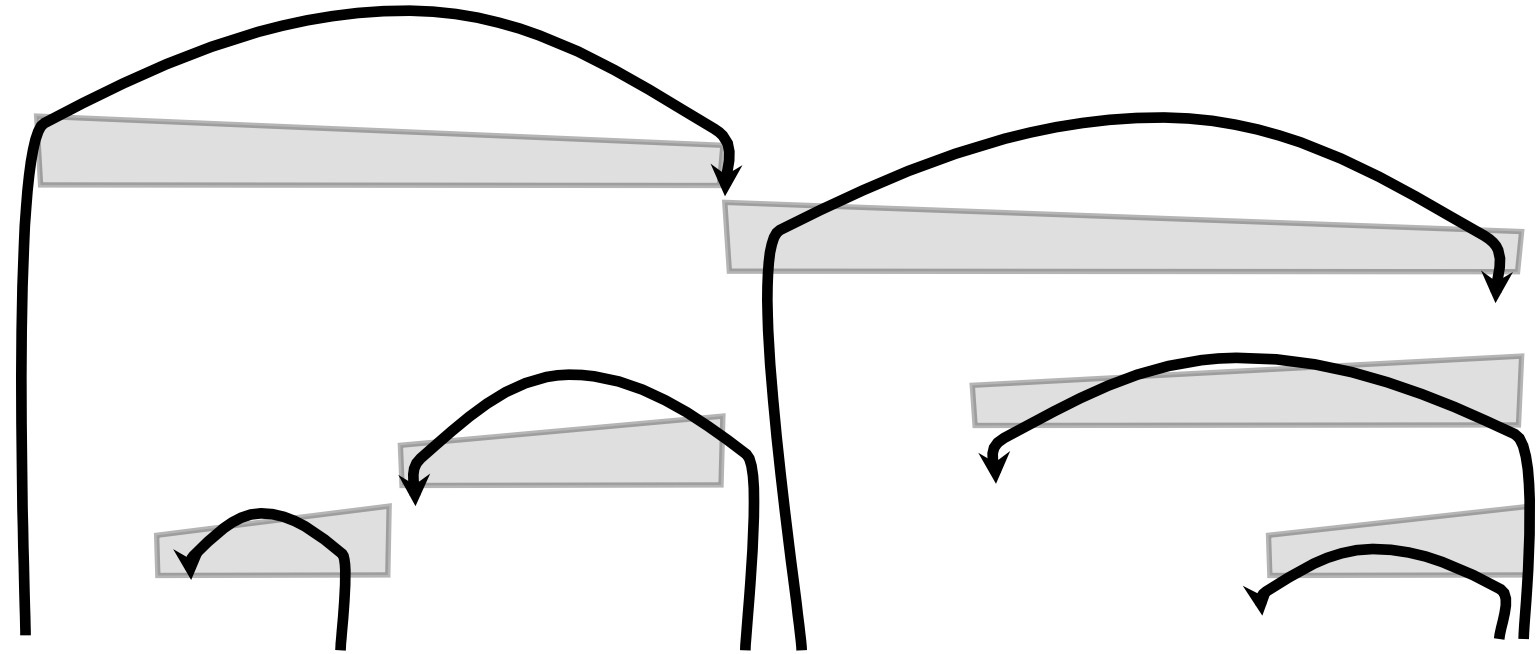
$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm



$ The professor chuckled with unabashed glee

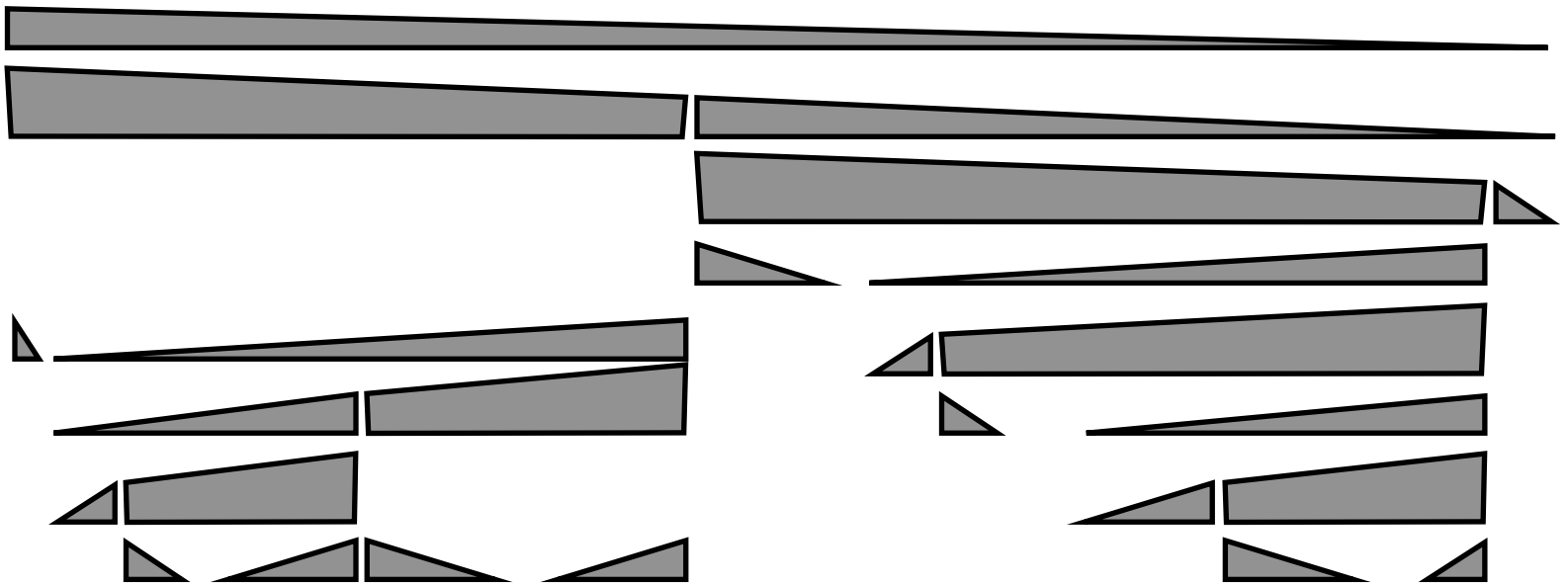
Example of the Eisner Algorithm



$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

Of course, we have to build a lot of other triangles and trapezoids if we want to be sure we have the *best* parse.



$ The professor chuckled with unabashed glee

Punchline

- Rethinking the algorithm in terms of attachments rather than constituents gives us an asymptotic savings!
- Bare bones, projective dependency parsing is $O(n^3)$

- What about non-projectivity?

Non-projective Dependency Parsing (McDonald et al., 2005)

- Key idea: a non-projective dependency parse is a **directed spanning tree** where
  - vertices = words
  - directed edges = parent-to-child relations
- Well-known problem: minimum-cost spanning tree
- Solution: Chu-Liu-Edmonds algorithm (cubic)
  - Tarjan: quadratic for dense graphs (like ours)
- Good news: fast! can now recover non-projective trees!
- Bad news: much larger search space, potential for error

Breaking Independence Assumptions

- Adding labels doesn't fundamentally change Eisner or MST
- What about edge-factoring?
- Projective case: local statistical dependence among same-side children of a given head - still cubic (Eisner and Satta, 1999).
- Non-projective parsing with any kind of second-order features (e.g., on adjacent edges) is NP-hard.
  - McDonald explored approximations in his thesis
  - Find the best projective parse and then rearrange the edges as long as the score improves - $O(n^3)$
  - ILP (Martins et al., 2009)

CoNLL 2006 & 2007

- 2006 and 2007: dependency parsing on a variety of languages was the shared task at CoNLL - a few dozen systems.
- Many of the datasets are freely available.
- Parsing papers now typically evaluate on most or all of these datasets.

Parsing in 2011

Current Research Directions

- Better learning for parsing (e.g., max margin as in Taskar et al., 2004; CRFs as in Finkel et al., 2008; many other "parsing" papers that are really about learning for structured prediction)
- Integrating learning and search (Petrov, 2009)
- Synchronous grammars in machine translation; bilingual parsing
- Richer formalisms (CCG, TAG, unification-based grammars)
- Integrating parsing with morphological or semantic analysis