

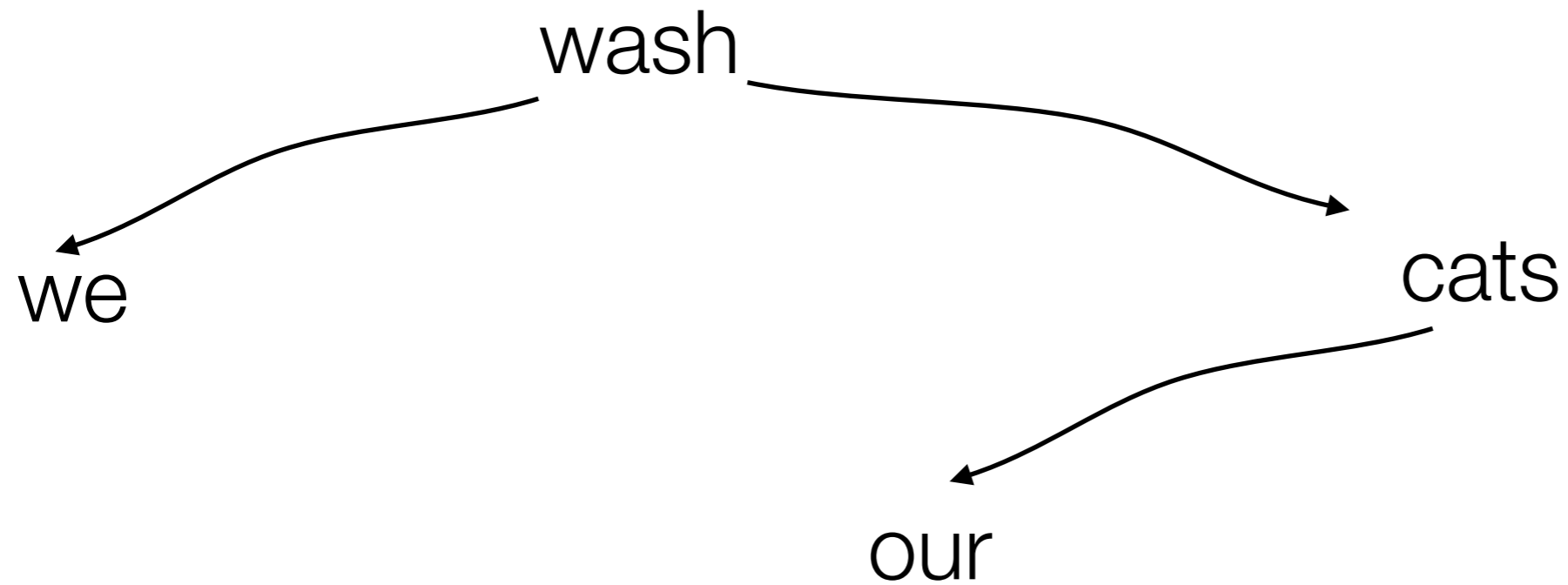
Language and Statistics II

Lecture 17: Dependency Parsing

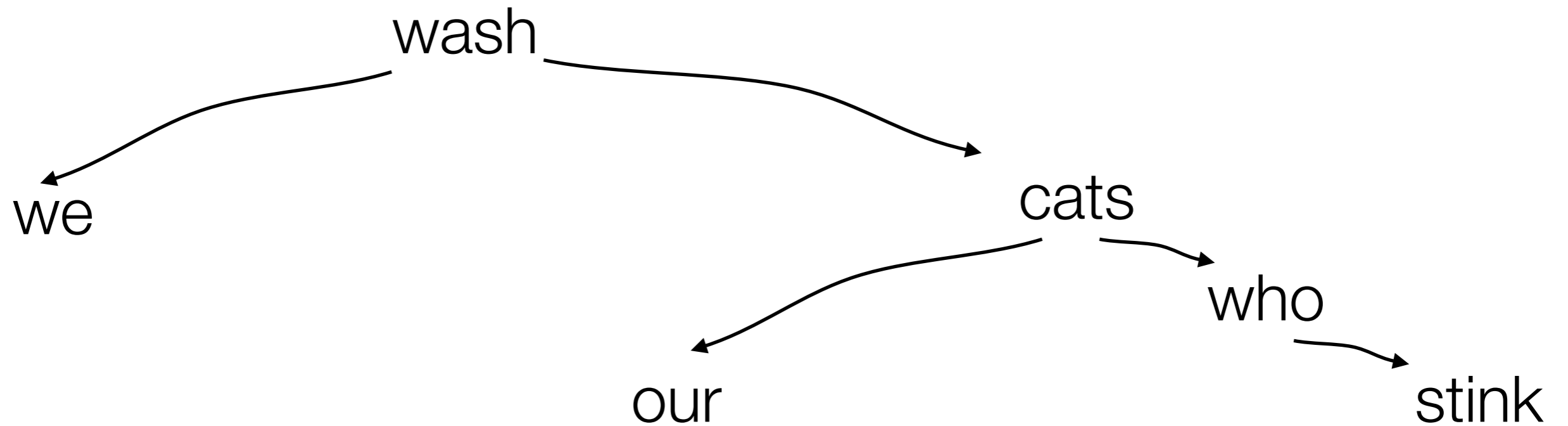
Dependency Grammar

- 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

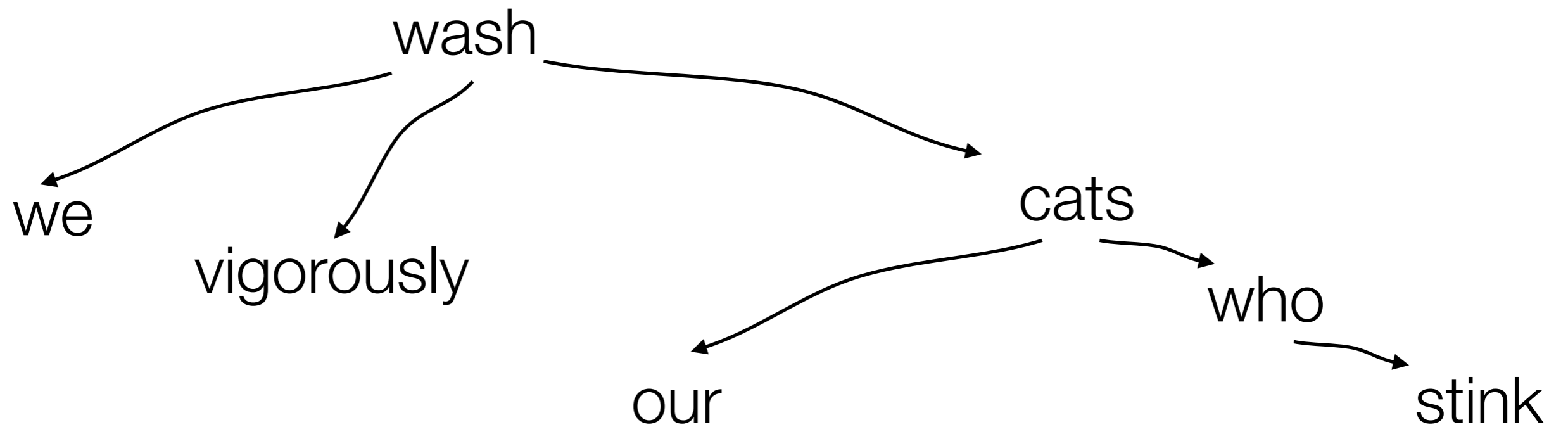
Bare Bones Dependency Parse



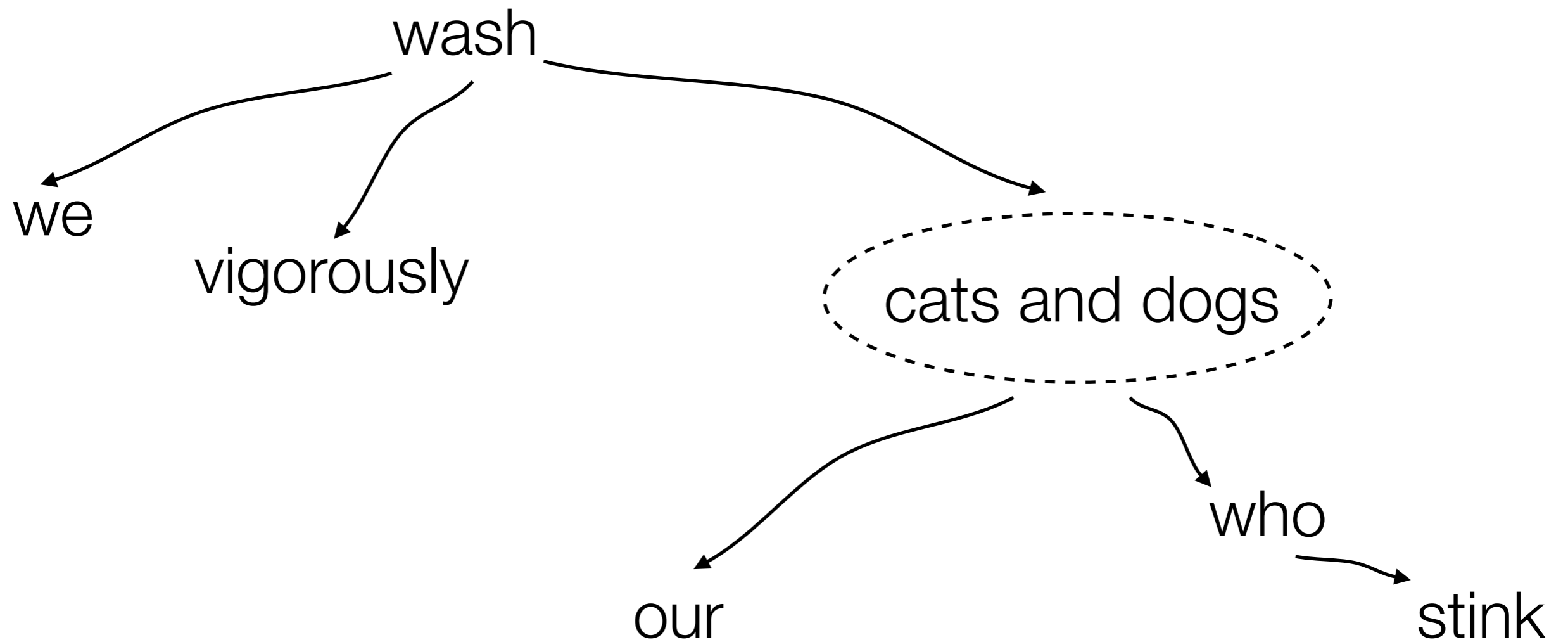
Bare Bones Dependency Parse



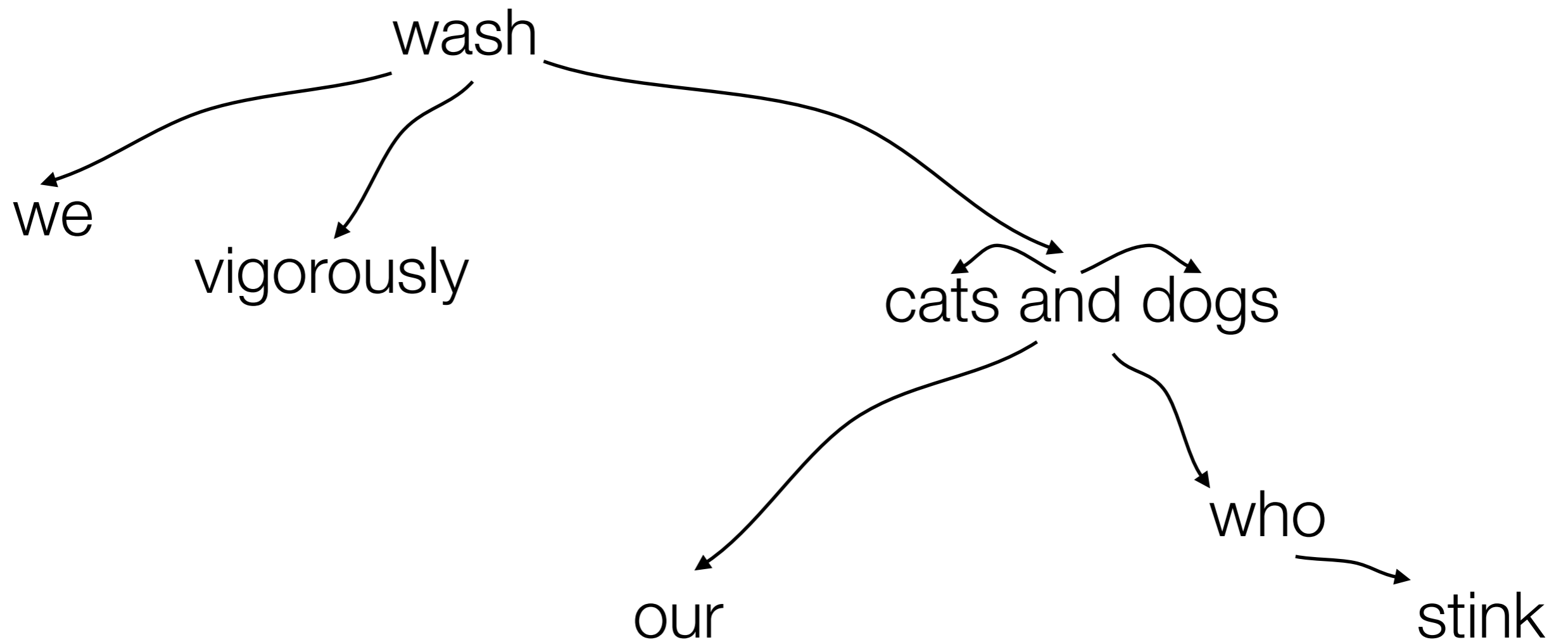
Bare Bones Dependency Parse



Bare Bones Dependency Parse



Bare Bones Dependency Parse

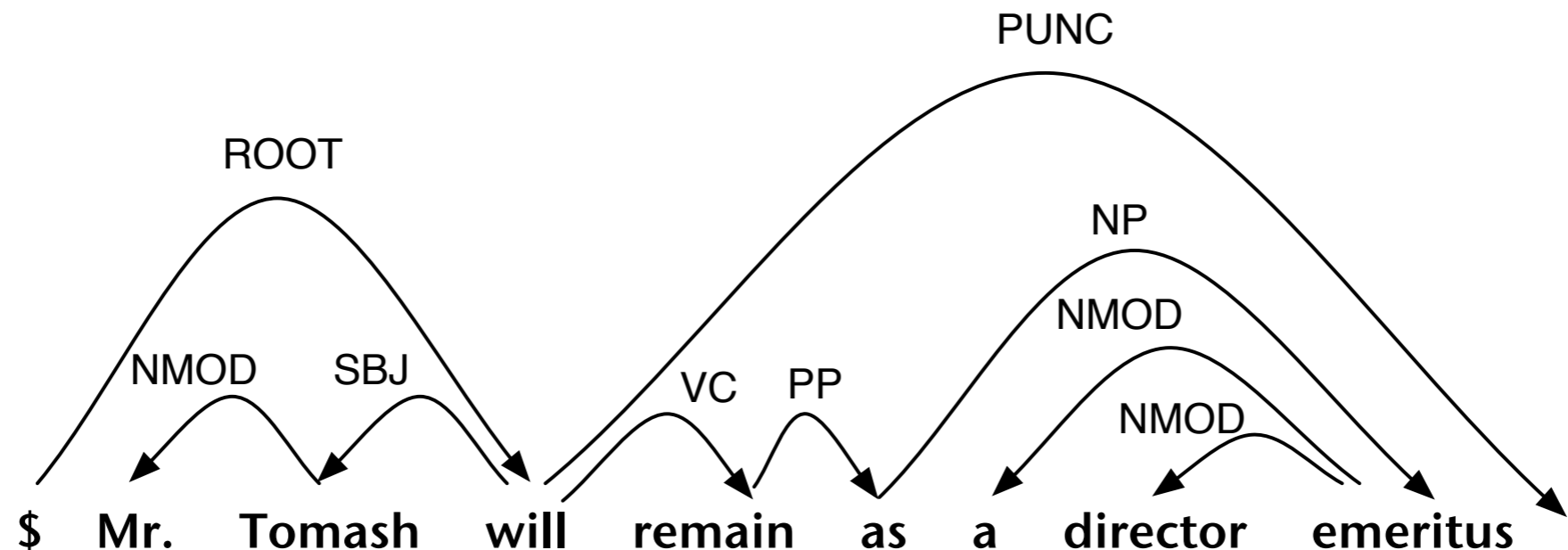


Bare Bones Dependencies and Labels

- The way to represent a lot of phenomena is clear (predicate-argument 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

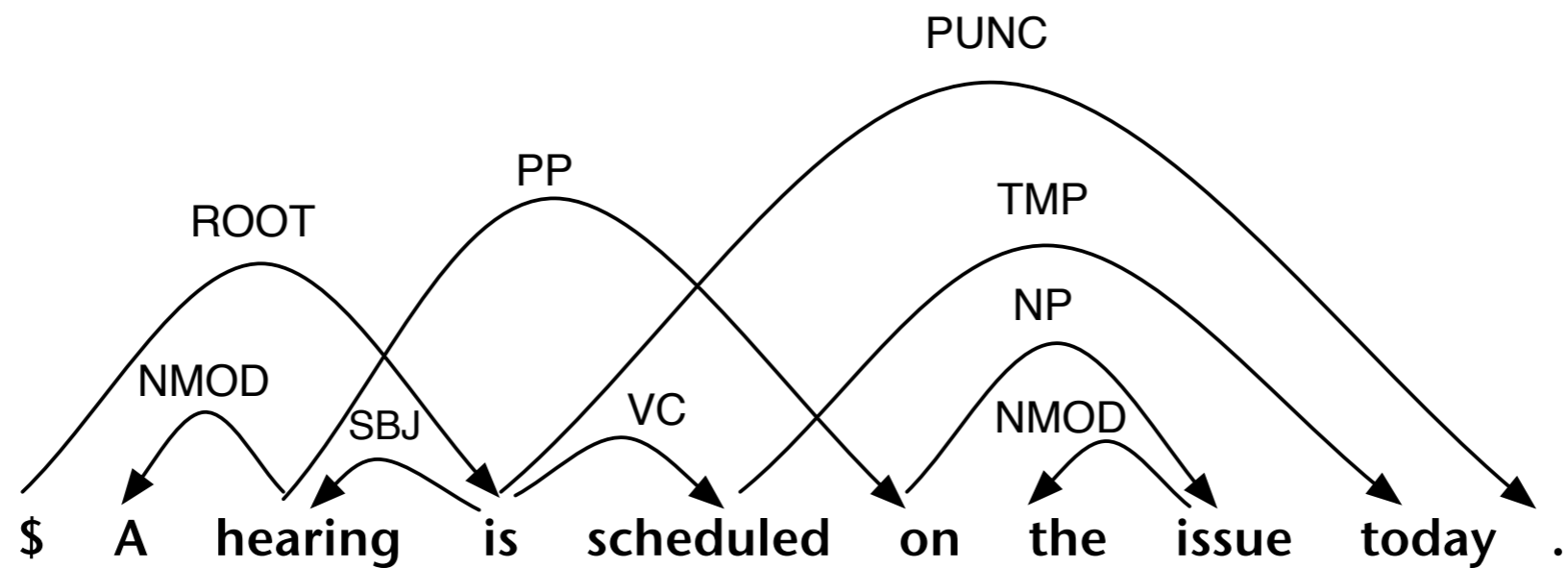
Dependencies and Context-Freeness

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



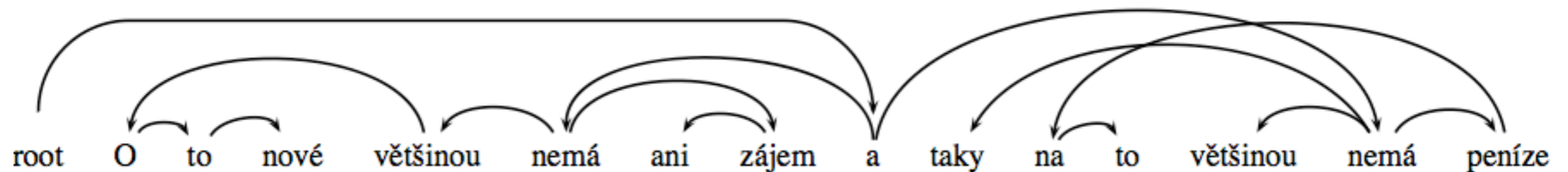
Dependencies and Context-Freeness

- But not entirely!



Dependencies 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

$$p_{\vec{\theta}}(\mathbf{y} \mid \mathbf{x}) \propto \prod_{i=1}^n e^{\vec{\theta} \cdot \vec{f}(\mathbf{x}, x_i, x_{\mathbf{y}(i)})}$$

- 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 log-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 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**
 - See work by J. Nivre.

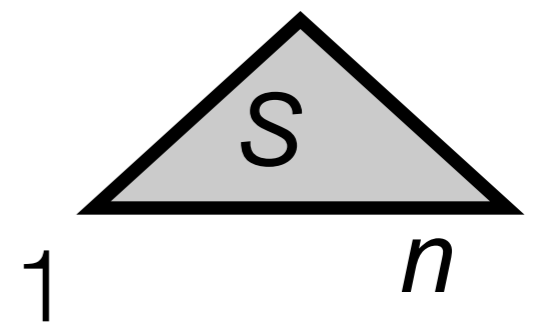
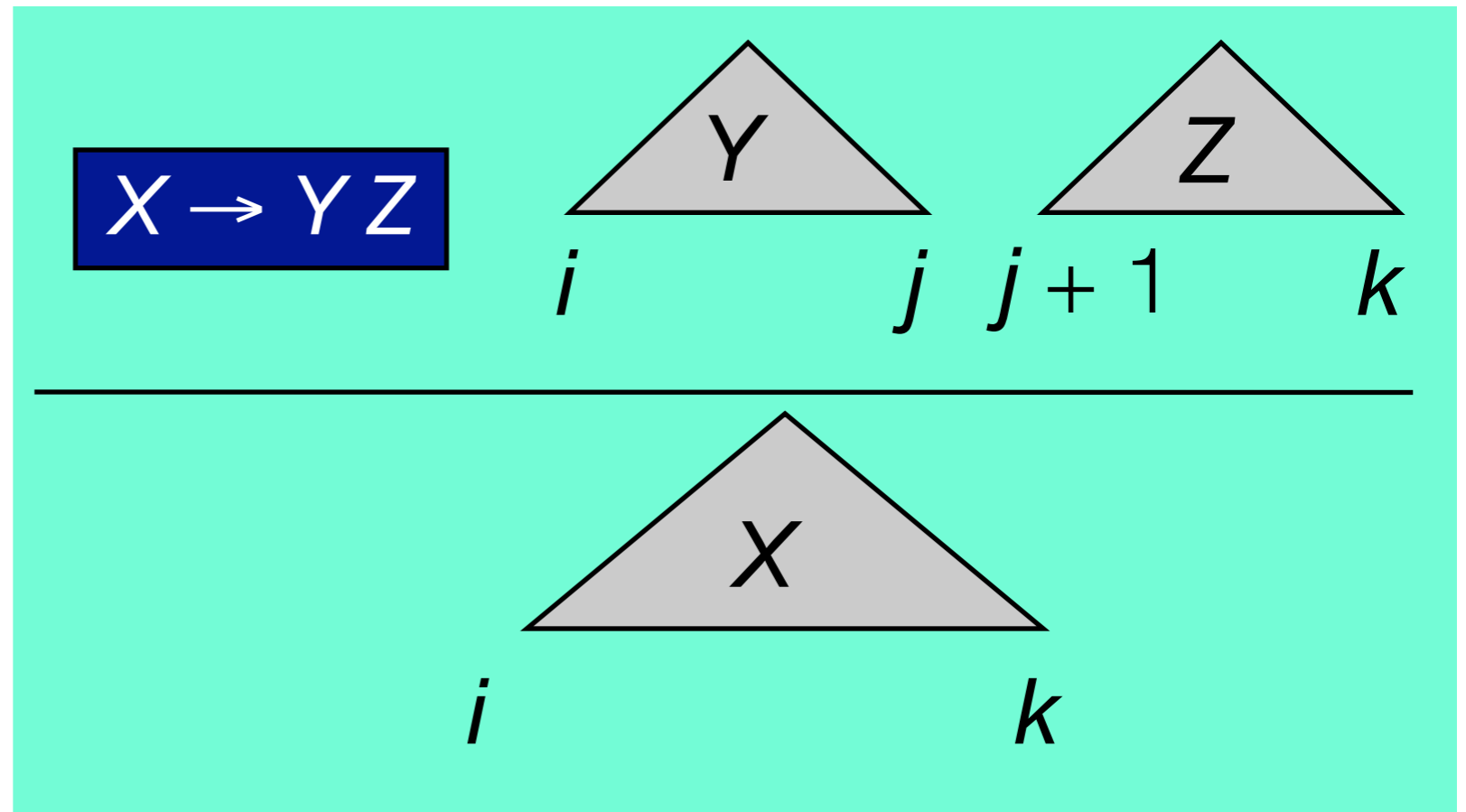
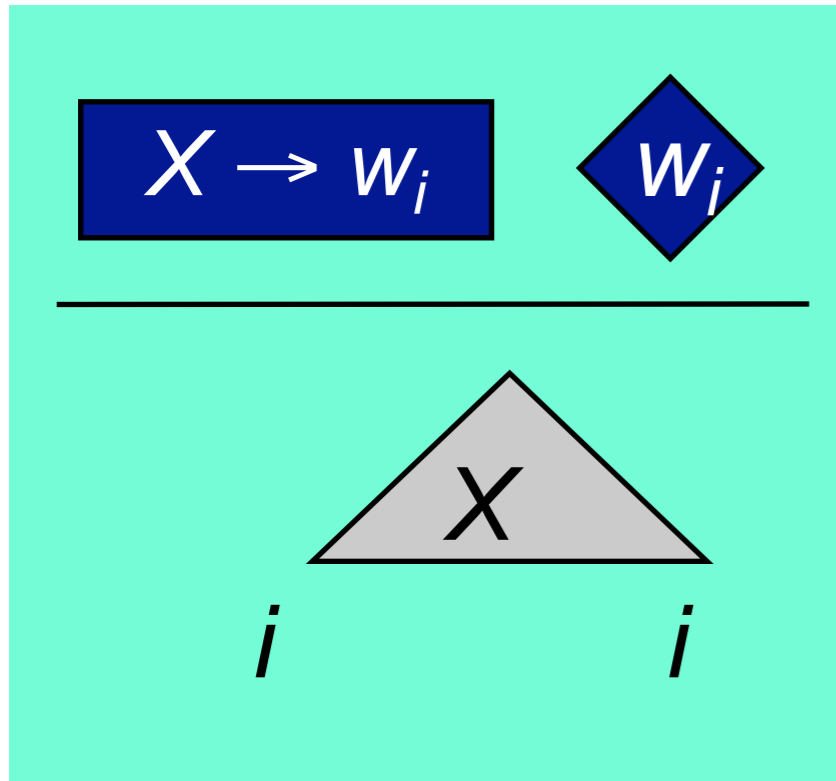
Algorithms != Models

- 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 features are factored by dependency tree edges.

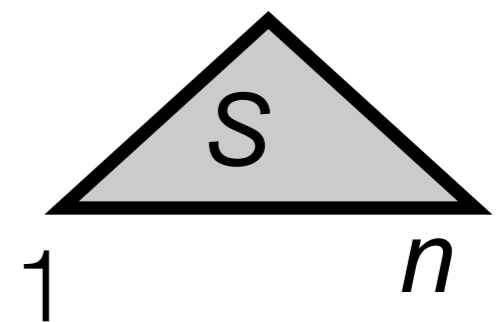
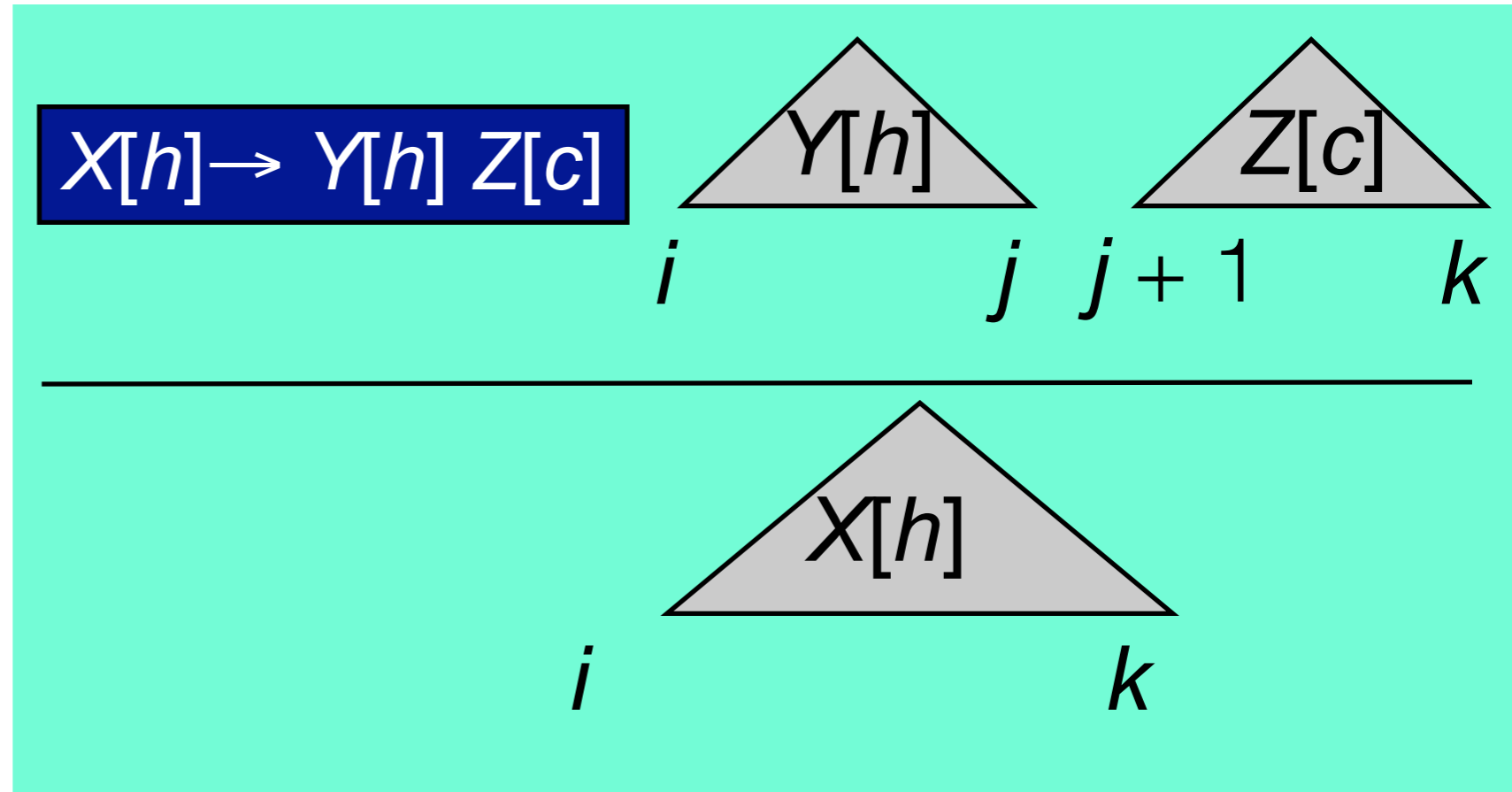
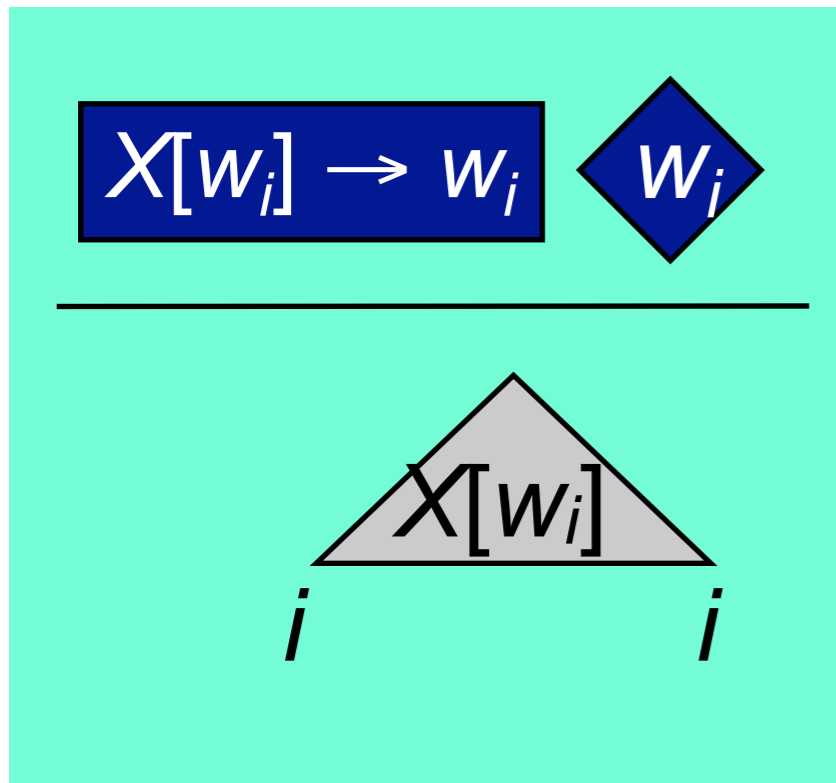
$$\vec{f}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n \vec{f}(\mathbf{x}, x_i, x_{\mathbf{y}(i)})$$

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

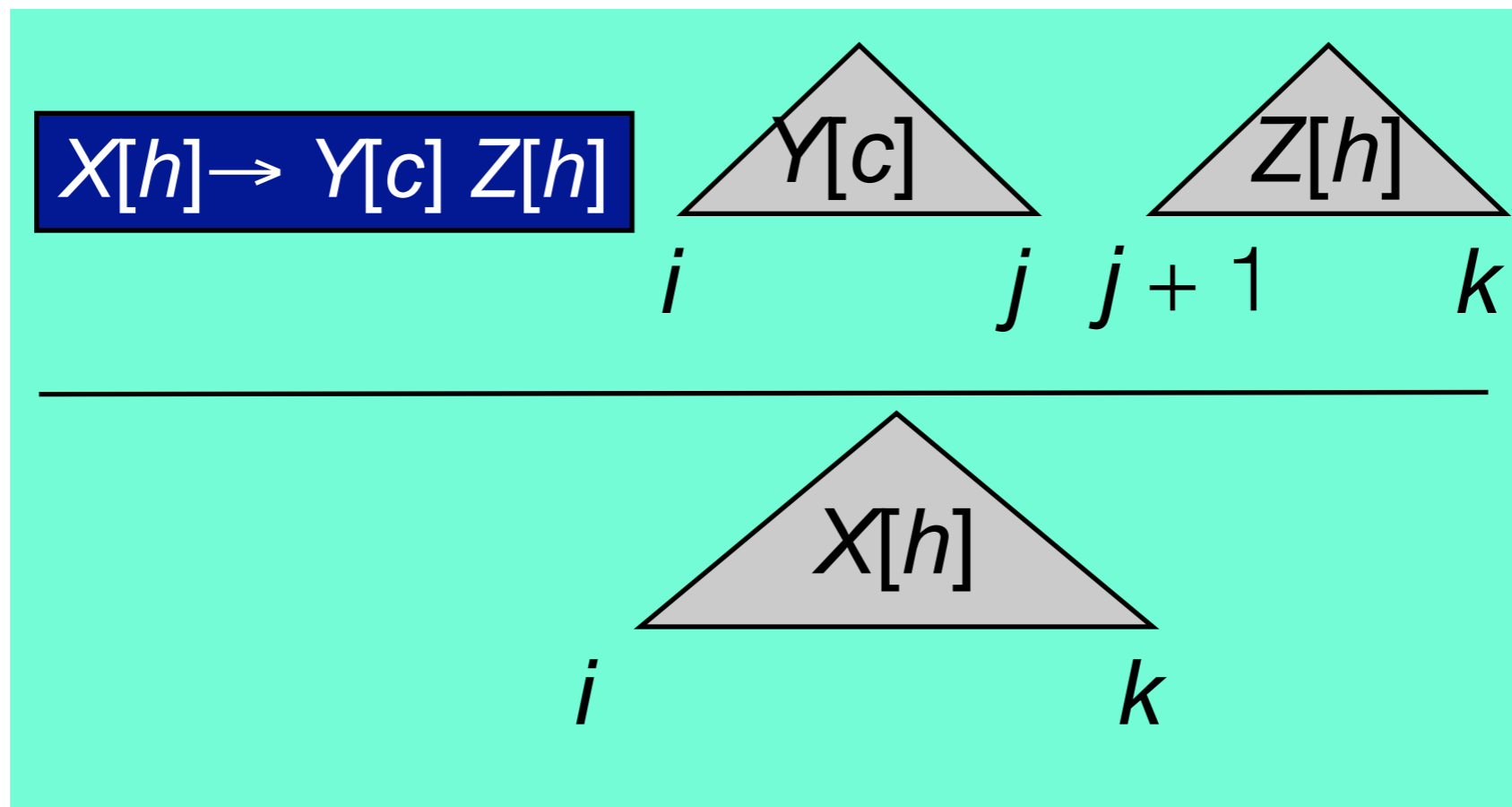
CKY



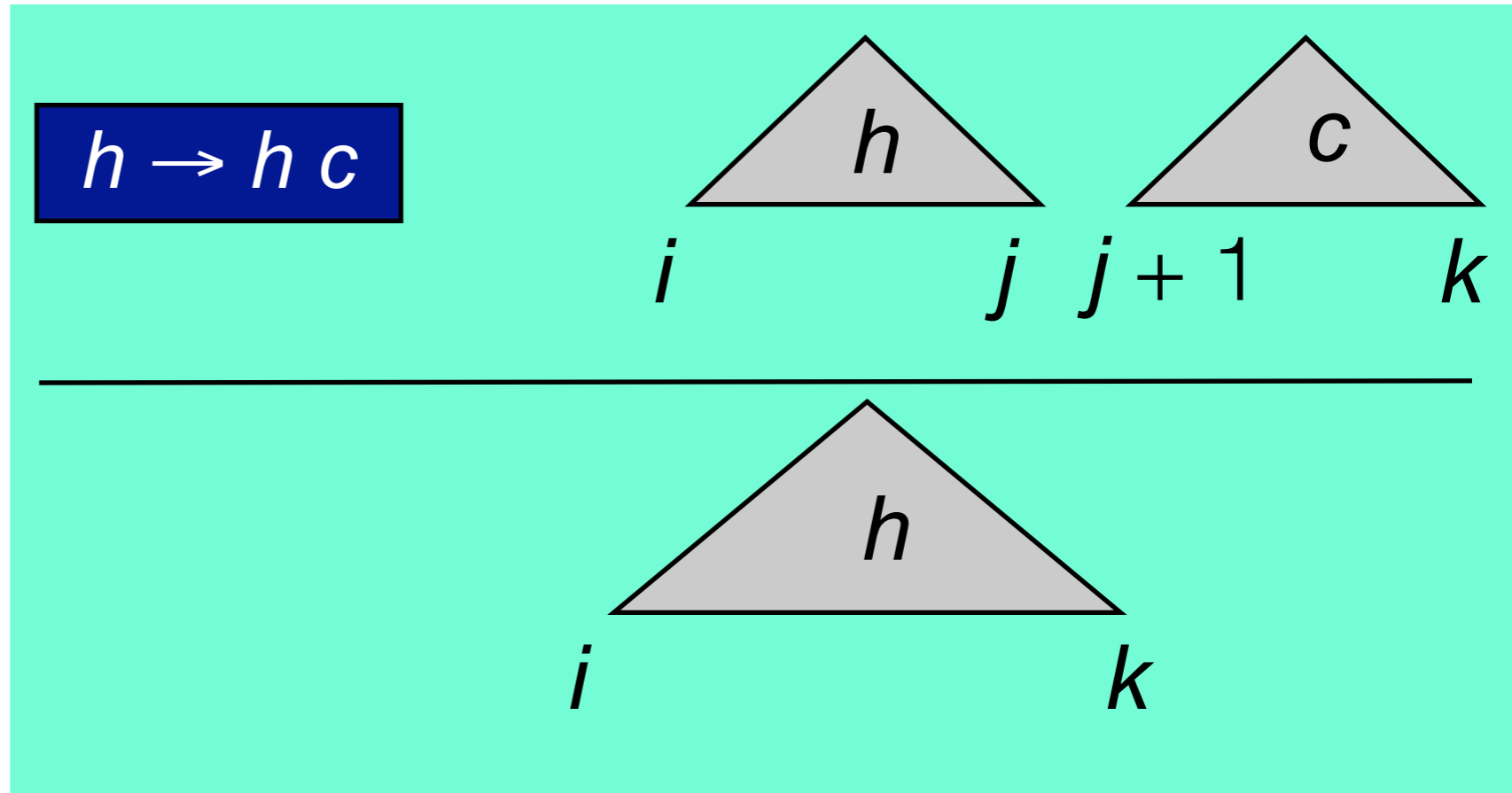
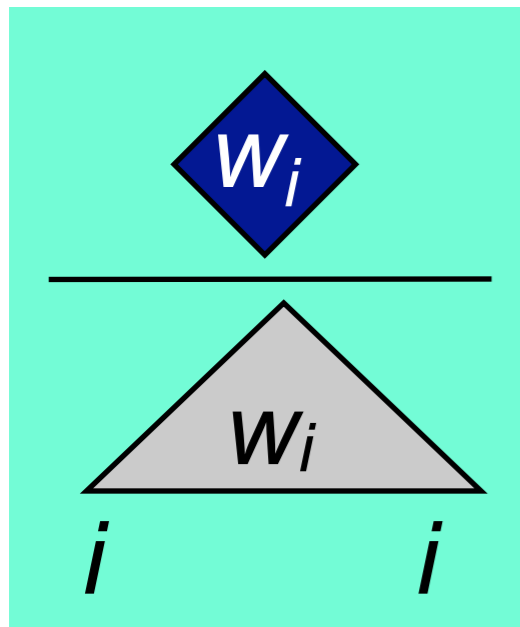
CKY with Heads



CKY with Heads (one more rule)

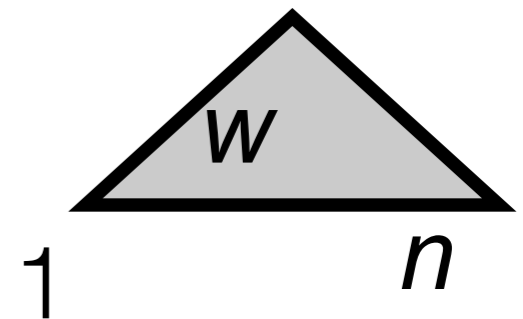


CKY with Heads, without Nonterminals

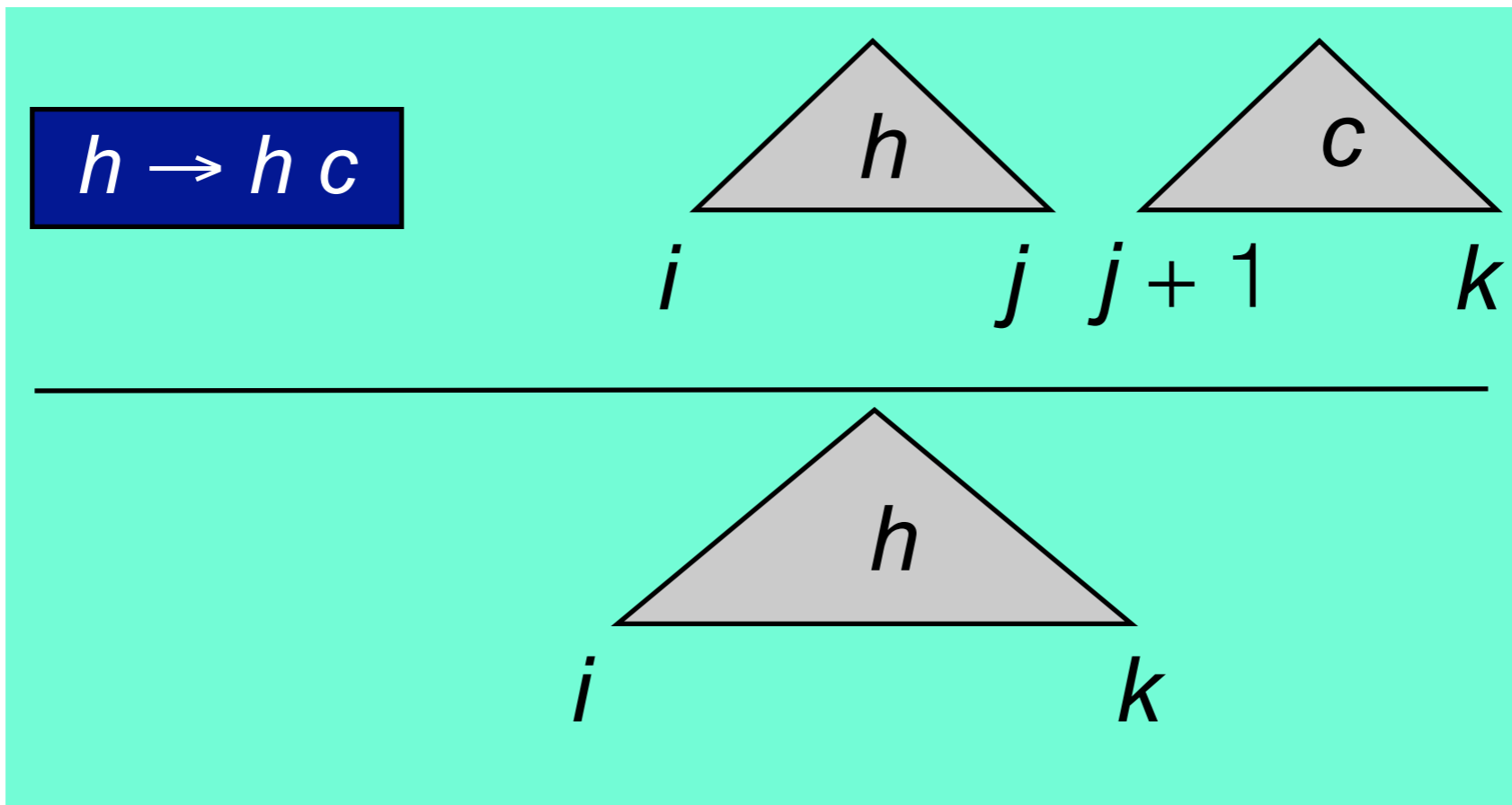


*Plus the rule for $h \rightarrow c h$.

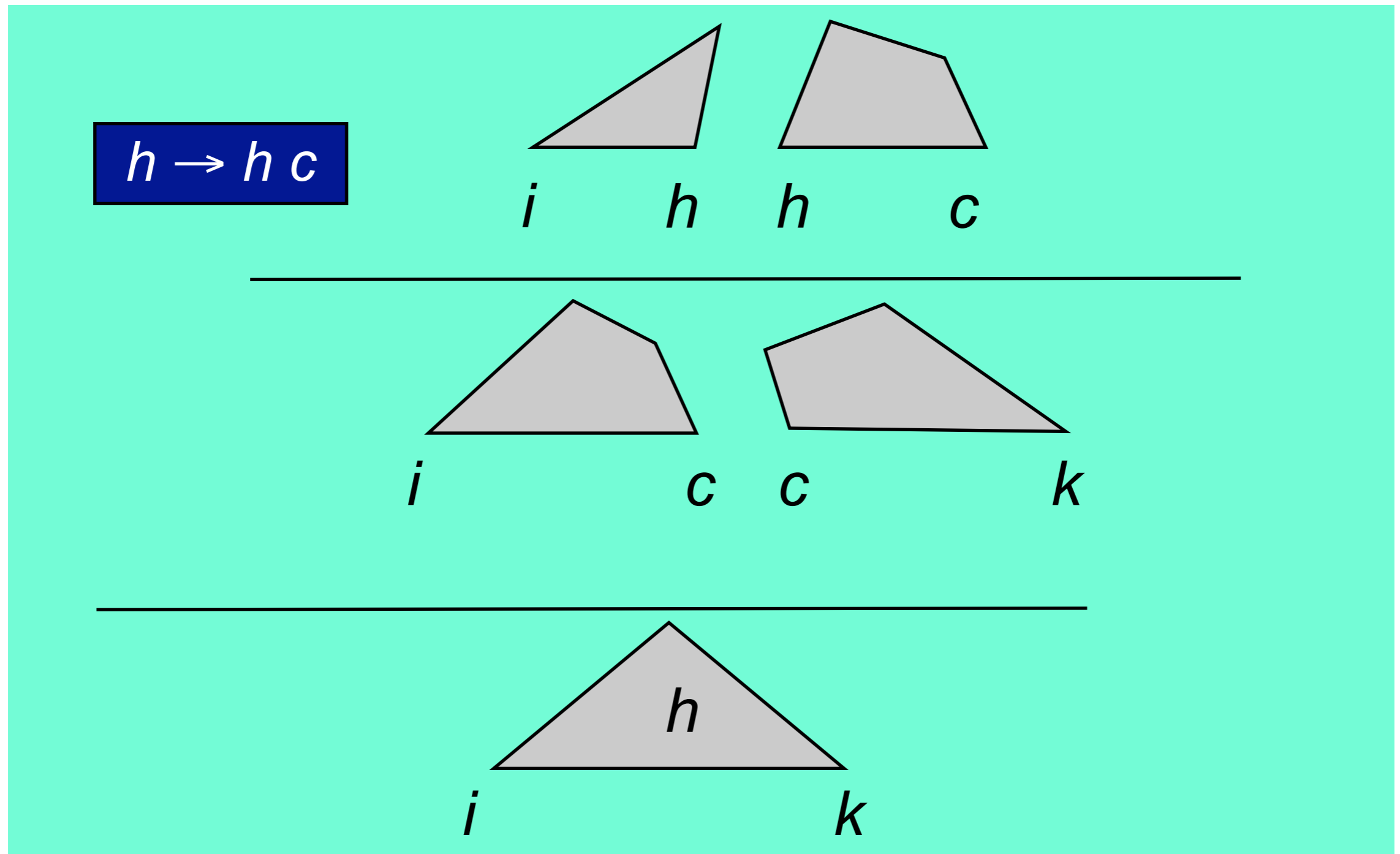
What's the runtime?



From CKY to Eisner's Algorithm



From CKY to Eisner's Algorithm



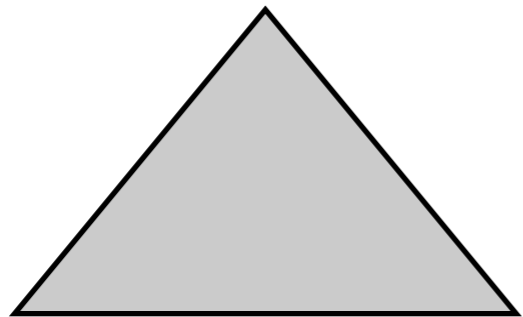
What's the runtime?

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?

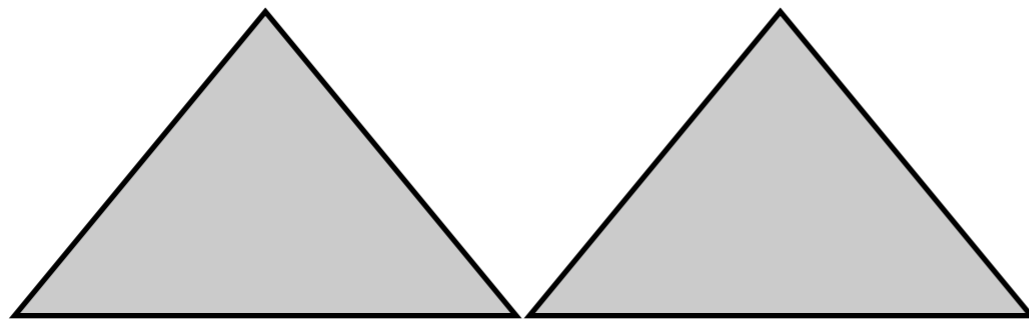
Punchline

- 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?



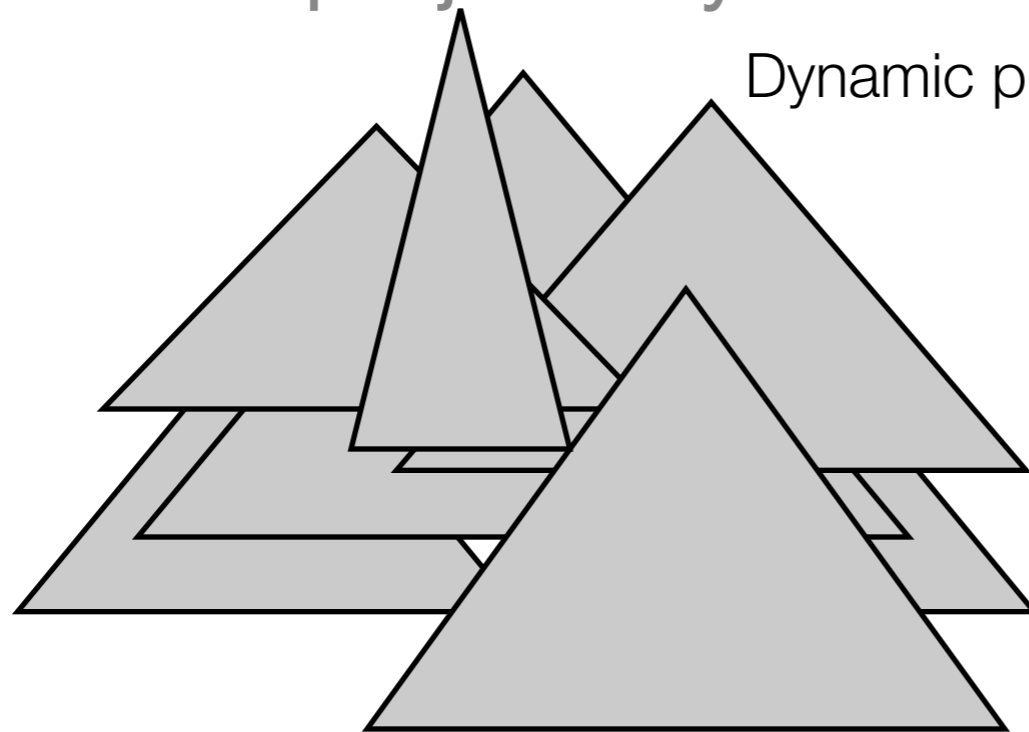
Punchline

- 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?



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?



Dynamic programming doesn't seem to work here!

Non-projective Dependency Parsing

- 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

Training graph-based parsers

- Online large margin learning (McDonald et al, 2005)

Training data: $\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^T$

1. $\vec{\theta}_0 = \mathbf{0}; \vec{\phi} = \mathbf{0}; i = 0$

2. for $n : 1..N$

3. for $t : 1..T$

4. $\min \|\vec{\theta}^{(i+1)} - \vec{\theta}^{(i)}\|$

 s.t. $s(\mathbf{x}_t, \mathbf{y}_t) - s(\mathbf{x}_t, \mathbf{y}') \geq L(\mathbf{y}_t, \mathbf{y}'), \forall \mathbf{y}' \in \text{dt}(\mathbf{x}_t)$

5. $\vec{\phi} = \vec{\phi} + \vec{\theta}^{(i+1)}$

6. $i = i + 1$

7. $\vec{\theta} = \vec{\phi} / (N * T)$

Training graph-based parsers

- Online large margin learning (McDonald et al, 2005)

Training data: $\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^T$

1. $\vec{\theta}_0 = 0; \vec{\phi} = 0; i = 0$

2. for $n : 1..N$

3. for $t : 1..T$

4. $\min \|\vec{\theta}^{(i+1)} - \vec{\theta}^{(i)}\|$

s.t. $s(\mathbf{x}_t, \mathbf{y}_t) - s(\mathbf{x}_t, \mathbf{y}') \geq L(\mathbf{y}_t, \mathbf{y}'), \forall \mathbf{y}' \in \text{dt}(\mathbf{x}_t)$

5. $\vec{\phi} = \vec{\phi} + \vec{\theta}^{(i+1)}$

6. $i = i + 1$

7. $\vec{\theta} = \vec{\phi} / (N * T)$

smallest update on the weight vector...

possible dependency trees of \mathbf{x}_t

... that ensures a separation margin between the correct parse and all the wrong parses

Training graph-based parsers

- Online large margin learning (McDonald et al, 2005)

Training data: $\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^T$

1. $\vec{\theta}_0 = 0; \vec{\phi} = 0; i = 0$

2. for $n : 1..N$

3. for $t : 1..T$

4. $\min \|\vec{\theta}^{(i+1)} - \vec{\theta}^{(i)}\|$

s.t. $s(\mathbf{x}_t, \mathbf{y}_t) - s(\mathbf{x}_t, \mathbf{y}') \geq L(\mathbf{y}_t, \mathbf{y}'), \forall \mathbf{y}' \in \text{dt}(\mathbf{x}_t)$

5. $\vec{\phi} = \vec{\phi} + \vec{\theta}^{(i+1)}$

6. $i = i + 1$

7. $\vec{\theta} = \vec{\phi} / (N * T)$

$s(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n \vec{\theta} \cdot \vec{f}(\mathbf{x}, x_i, x_{\mathbf{y}(i)})$

smallest update on the weight vector...

possible dependency trees of \mathbf{x}_t

... that ensures a separation margin between the correct parse and all the wrong parses

Training graph-based parsers

- Probabilistic Discriminative Models (Smith and Smith, 2007)

$$p_{\vec{\theta}}(\mathbf{y} \mid \mathbf{x}) = \frac{e^{\vec{\theta} \cdot \vec{f}(\mathbf{x}, \mathbf{y})}}{\sum_{\mathbf{y}' \in \mathcal{Y}_{\mathbf{x}}} e^{\vec{\theta} \cdot \vec{f}(\mathbf{x}, \mathbf{y}')}} = \frac{e^{\vec{\theta} \cdot \vec{f}(\mathbf{x}, \mathbf{y})}}{Z_{\vec{\theta}}(\mathbf{x})}$$

Training graph-based parsers

- Probabilistic Discriminative Models (Smith and Smith, 2007)

$$p_{\vec{\theta}}(\mathbf{y} \mid \mathbf{x}) = \frac{e^{\vec{\theta} \cdot \vec{f}(\mathbf{x}, \mathbf{y})}}{\sum_{\mathbf{y}' \in \mathcal{Y}_{\mathbf{x}}} e^{\vec{\theta} \cdot \vec{f}(\mathbf{x}, \mathbf{y}')}} = \frac{e^{\vec{\theta} \cdot \vec{f}(\mathbf{x}, \mathbf{y})}}{Z_{\vec{\theta}}(\mathbf{x})}$$

- Training the model:

$$\max_{\vec{\theta}} \sum_{\mathbf{x}, \mathbf{y}} \tilde{p}(\mathbf{x}, \mathbf{y}) \left(\vec{\theta} \cdot \vec{f}(\mathbf{x}, \mathbf{y}) \right) - \sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \log Z_{\vec{\theta}}(\mathbf{x})$$

Training graph-based parsers

- Training the model:

$$\max_{\vec{\theta}} \sum_{\mathbf{x}, \mathbf{y}} \tilde{p}(\mathbf{x}, \mathbf{y}) \left(\vec{\theta} \cdot \vec{f}(\mathbf{x}, \mathbf{y}) \right) - \sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \log Z_{\vec{\theta}}(\mathbf{x})$$

summing up over all possible trees: **expensive**

Training graph-based parsers

- Training the model:

$$\max_{\vec{\theta}} \sum_{\mathbf{x}, \mathbf{y}} \tilde{p}(\mathbf{x}, \mathbf{y}) \left(\vec{\theta} \cdot \vec{f}(\mathbf{x}, \mathbf{y}) \right) - \sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \log Z_{\vec{\theta}}(\mathbf{x})$$

- Smith and Smith showed that this can be done efficiently in $O(n^3)$ time, using the Matrix Tree Theorem.

CoNLL 2006 & 2007

- 2006 and 2007: dependency parsing on a variety of languages was the shared task at CoNLL - a few dozen systems.
- Evaluation: attachment accuracy (usually punctuation is ignored).
 - Labeled version (edges have labels - how are algorithms affected?)
 - Unlabeled version (“bare bones”)
- Many of the datasets are freely available.
- State-of-the-art: a combination of graph-based and transition-based parsers!

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 - $O(n^3)$

More Approximate Methods

- Nivre and McDonald (2008) combined graph-based and transition-based approaches
- Martins et. al. (2008) showed that this is an instance of stacked learning, and reported impressive results for several languages

More Approximate Methods

- Integer Linear Programming

$$\begin{array}{ll} \min_{\mathbf{x} \in \mathbb{R}^d} & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^d \end{array}$$

More Approximate Methods

- Integer Linear Programming

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^d} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^d \end{aligned}$$

- Dependency Parsing as ILP

$$\begin{aligned} \max_{y \in \mathcal{Y}(x)} \mathbf{w}^\top \mathbf{f}(x, y) &= \max_{\mathbf{z} \in \mathcal{Z}(x)} \mathbf{w}^\top \mathbf{F}(x) \mathbf{z} \\ &= \max_{\mathbf{z}, \phi} \mathbf{s}^\top \mathbf{z} \\ \text{s.t.} \quad & \mathbf{A} \begin{bmatrix} \mathbf{z} \\ \phi \end{bmatrix} \leq \mathbf{b} \\ & \mathbf{z} \in \mathbb{B} \end{aligned}$$

More Approximate Methods

- Integer Linear Programming

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^d} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^d \end{aligned}$$

- Dependency Parsing as ILP

$$\begin{aligned} \max_{y \in \mathcal{Y}(x)} \mathbf{w}^\top \mathbf{f}(x, y) &= \max_{\mathbf{z} \in \mathcal{Z}(x)} \mathbf{w}^\top \mathbf{F}(x) \mathbf{z} \\ &= \max_{\mathbf{z}, \phi} \mathbf{s}^\top \mathbf{z} \\ \text{s.t.} \quad & \mathbf{A} \begin{bmatrix} \mathbf{z} \\ \phi \end{bmatrix} \leq \mathbf{b} \\ & \mathbf{z} \in \mathbb{B} \end{aligned}$$

- For more details, refer to the ACL paper by Martins et. al. (2009)