Natural Language Dependency Parsing

SPFLODD

19 February 2017

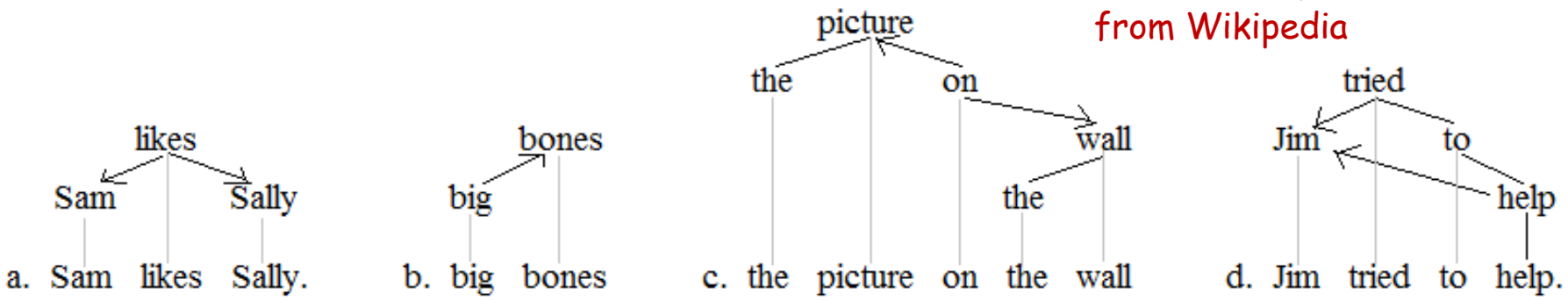
Grammar Structure

- The grammars we've seen so far are instances of *phrase structure* grammars
  - Deal with arrangement of *contiguous phrases* within the grammar
  - Not specific to natural language
    - Works for many other languages, e.g. computer programs, the language of irrationals
  - Natural language:
    - Assumption: Meaning is held in contiguous phrases
    - Phrase structure does not always reveal the entire story
      - Particularly in languages with free word order

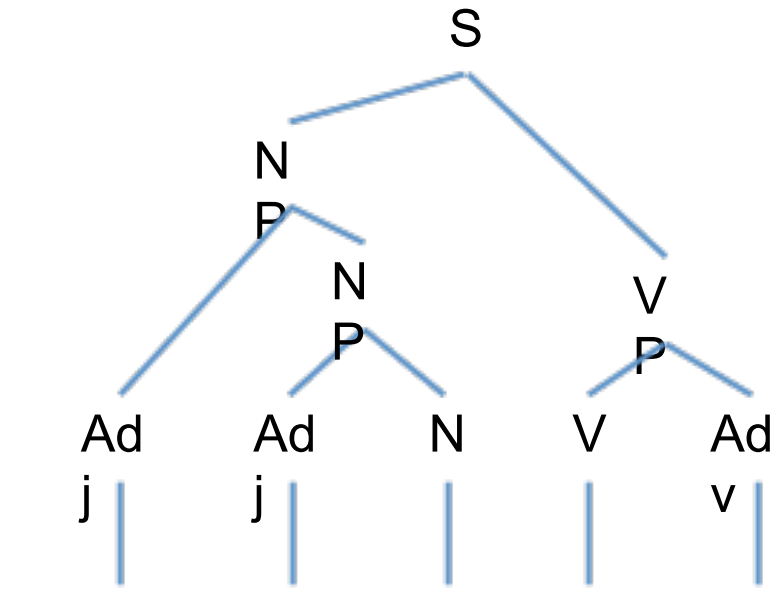
Dependency

- Dependency theory is built on the hypothesis that words are governed by one another directly, rather than through intermediate non-terminal entities
- Identifying these connections helps us interpret a sentence
- "Governance" can be identified based on

Semantic dependencies, from Wikipedia

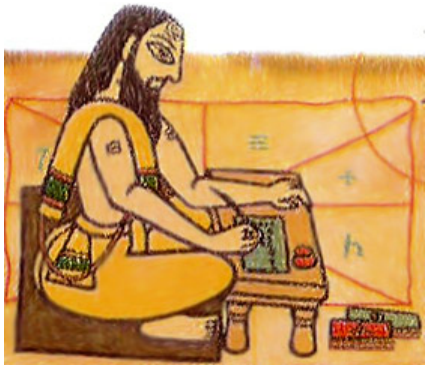


Dependency vs Phrase Structure



- But how do we decide which word connects to which?
  - Coming up..

A brief history



- The existence of dependencies between grammatical units was possibly first described by Pāṇini ca 7th century BC
  - I.e. at least as old as constituency theory

- The first grammarian to use the (equivalent of the) word "dependency" was by Ibn Maḍāʾ, in

Dependency

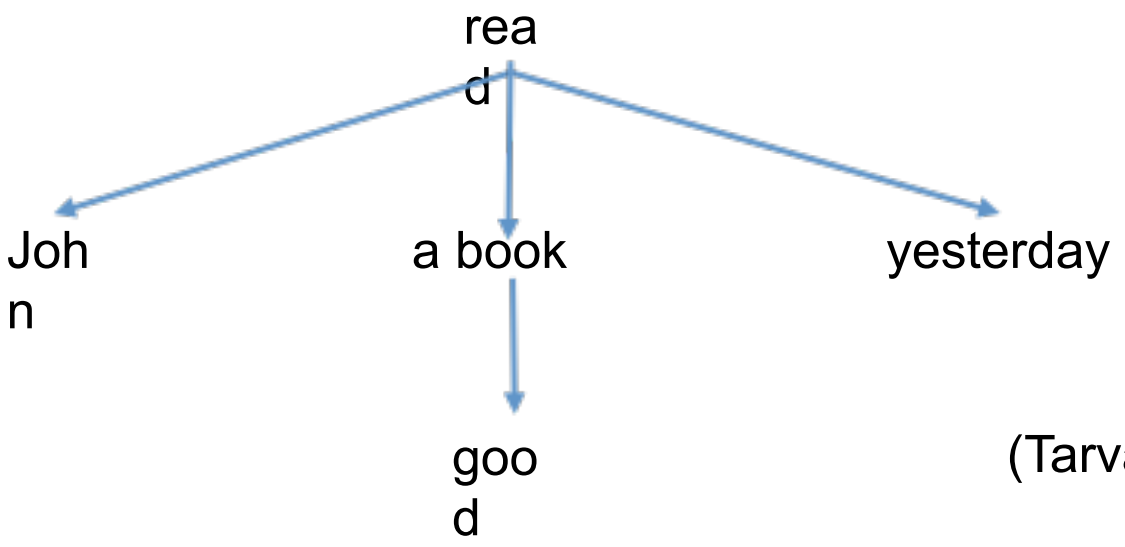
- Modern dependency theory traces back to the work of Tesnière
  - Lucien Tesnière. Éléments de syntaxe structural. Klincksieck, Paris 1959 (now available in English for only USD 175)
- Dependency is the notion that linguistic units (words) are connected to one another by directed links.
- The verb is taken to be the structural center of

Dependency structure

- Specifically, every word is assumed to be governed by *one* other word in the sentence
  - Though each word may govern multiple words
  - "Parent-child" relationship
  - I.e. *tree* structure

Dependency: hierarchy

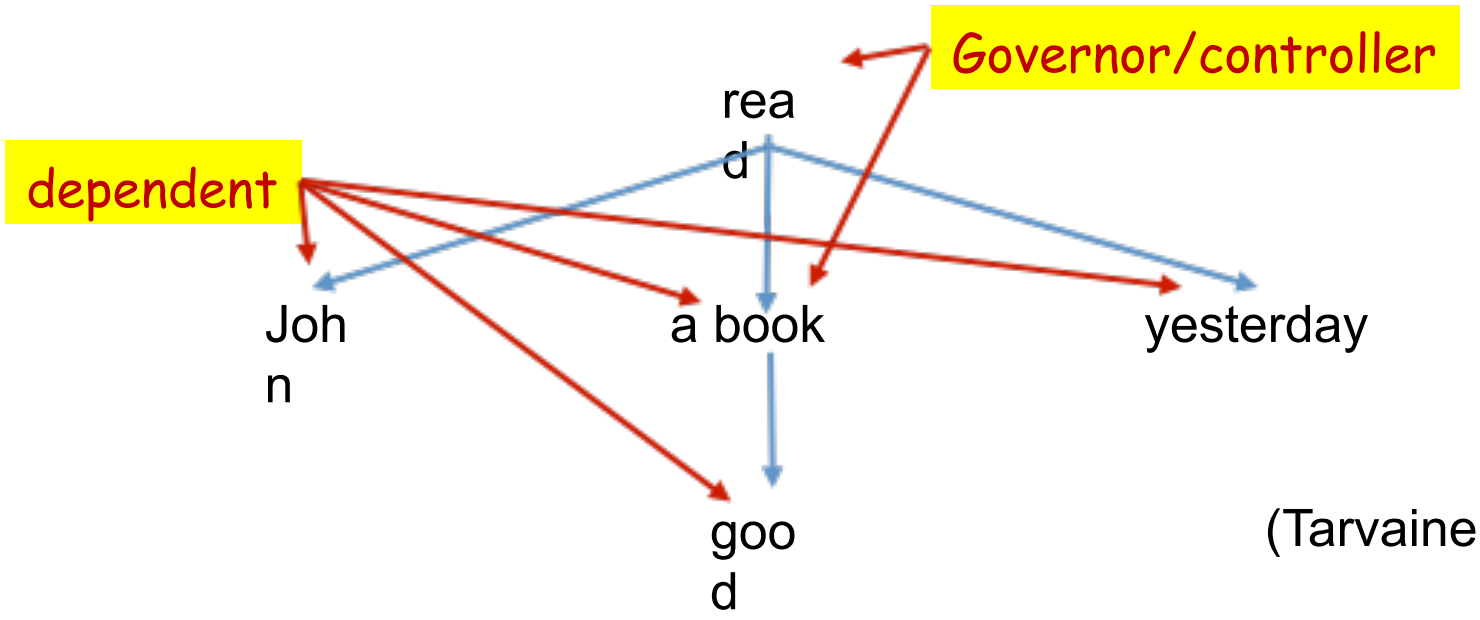
- A sentence is described as a hierarchical structure, where some parts of the sentence have a higher place than others
  - The verb has the highest place



(Tarvainen, ‘87)

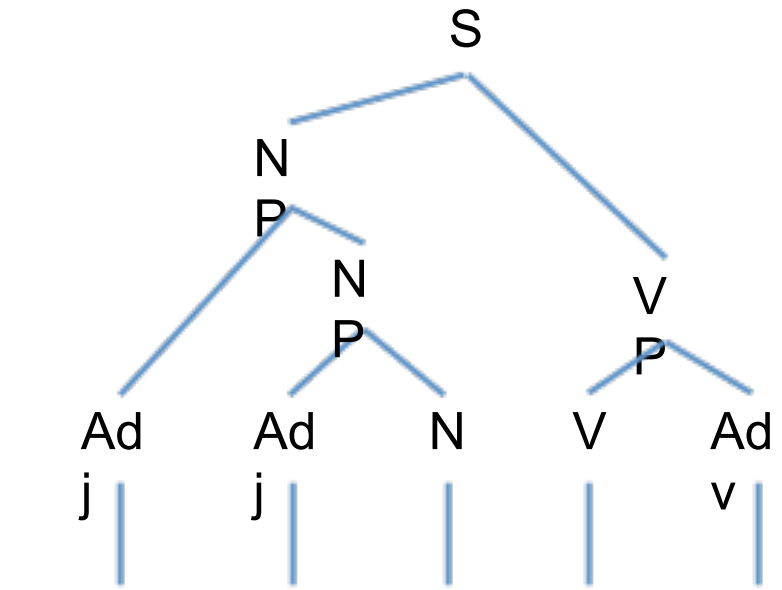
Dependency: hierarchy

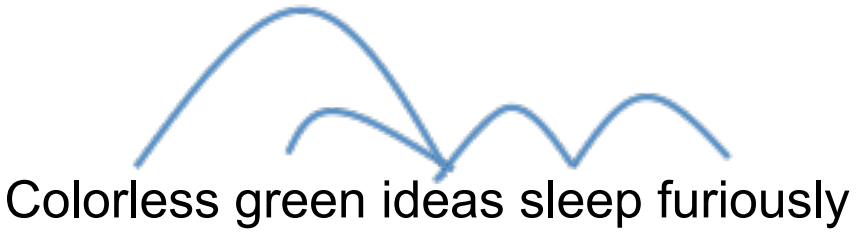
- A sentence is described as a hierarchical structure, where some parts of the sentence have a higher place than others
  - The verb has the highest place



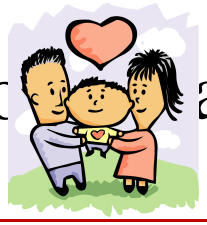
(Tarvainen, '87)

The hidden story

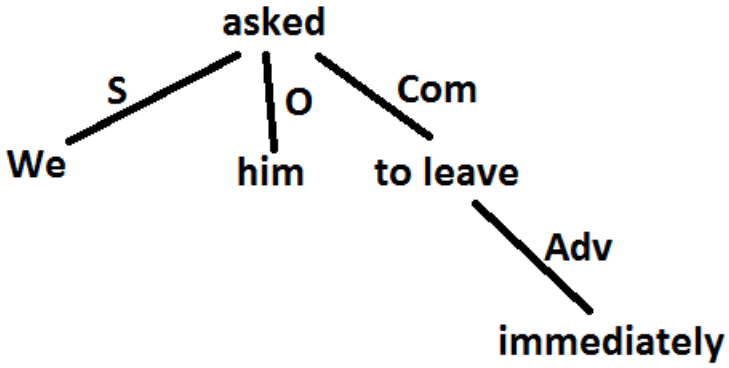


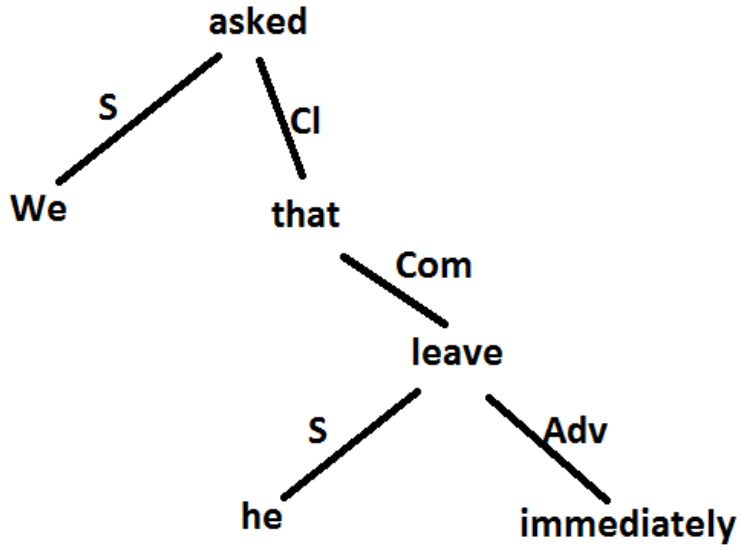


- Modern linguistic theories (both Chomsky and Tesnière) assume parent-child relationships
  - Single parent, (possibly) multiple children
  - It is this assumption that makes co ational modelling tractable



Dependency

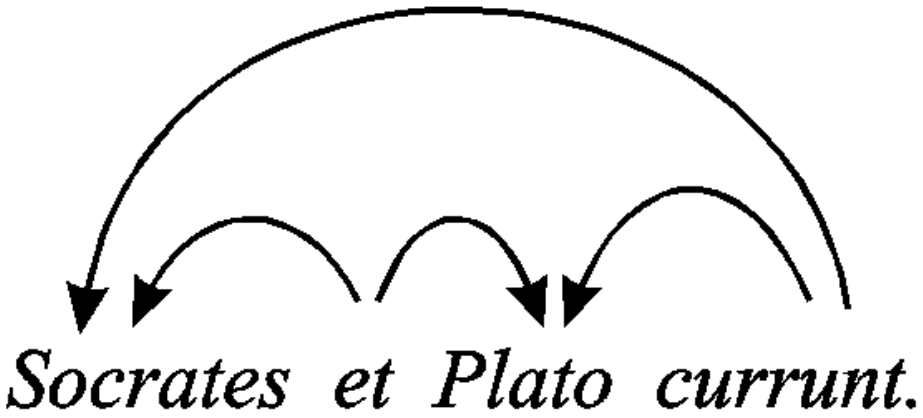




https://www.linkedin.com/pulse/parent-child-principle-grammar-wei-li

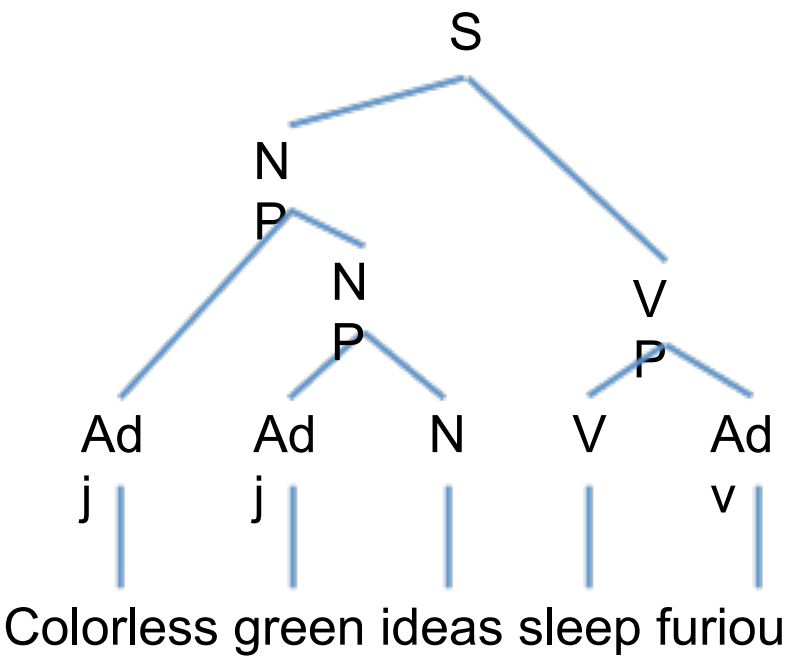
- Can change without changing meaning

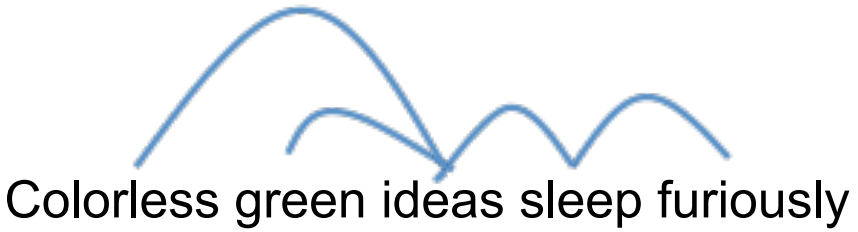
Even conjugation is a problem



- Even simple things like conjugation can cause problems
  - Example from Radulphus Brito, ca 1300
    - Brito was one of the "modistae", a group of grammarians in the late 1200s who also tried to formalize language

So: why dependency?





- Dependency trees are much smaller than phrase-structure trees
  - Easier to represent
- If appropriately plotted, can capture information that most (tractable) phrase

Equivalence

- "In general, it can be shown that for any *dependency grammar* there is a phrase structure grammar which will generate an identical set of sentences; likewise for any phrase structure grammar (or any PSF limited to the form of rule which we have illustrated) the same set of sentences can be generated by a dependency grammar. In that sense the two are said to be weakly equivalent"

The problem

- Determining a meaningful dependency structure in a sentence helps us interpret it
  - Assumption: structure is somehow related to meaning
- Problem: How do we determine the dependency structure?

The problem

Colorless green ideas sleep furiously

Colorless green ideas sleep furiously

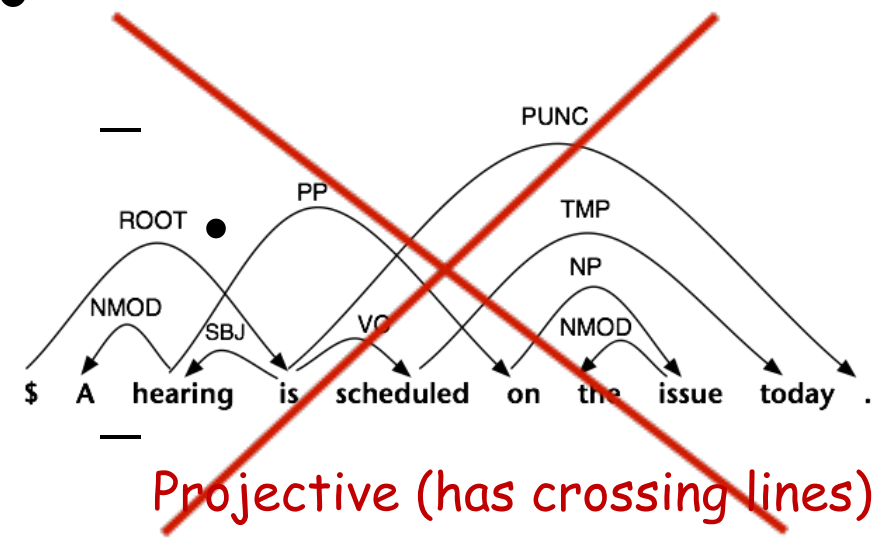
Colorless green ideas sleep furiously

- An exponentially large number of ways of drawing the tree

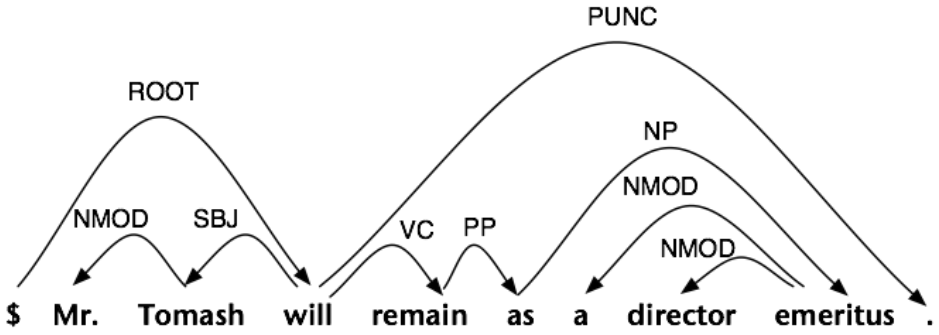
How do we choose the right one

Finding the dependency tree..

- AKA dependency parsing
- Define a structure to the tree
  - E.g. Exclude some options
-



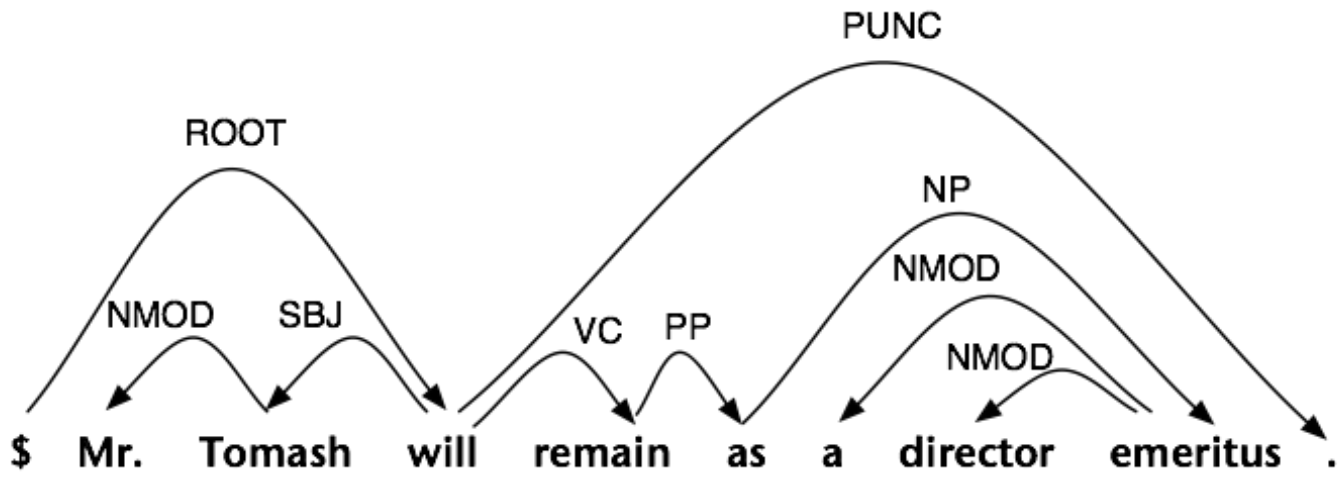
Projective (has crossing lines)



Projective (no crossing lines)

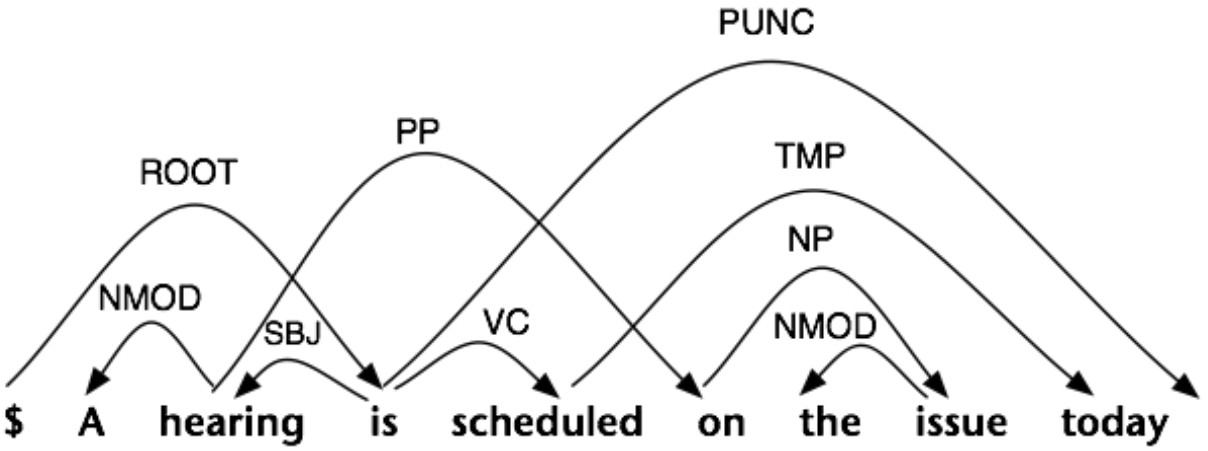
Dependencies and Context-Freeness

- **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

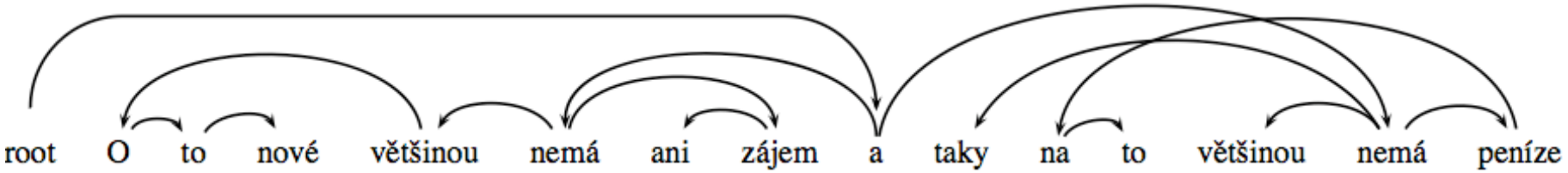
- But not entirely!



- Dependencies constructed through simple means from the Penn treebank will be projective.
  - Parses follow a CFG

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

Finding the dependency tree..

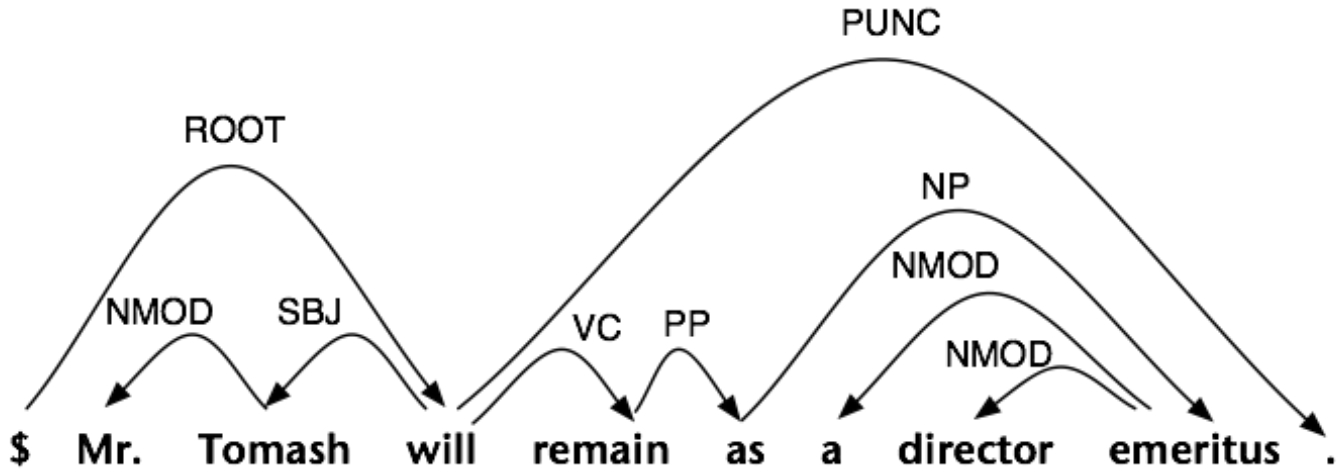
- AKA dependency parsing
- Define a structure to the tree
  - Exclude some options
- Define rules of dependency
  - What kinds of dependencies are possible
    - Or what makes some dependencies more likely than others
  - AKA a dependency grammar!
  - Problem: Need to define potential dependency between every word and every word!

Typical criteria for dependency

- W depends on head H in a stru
  - H determines syntactic catego
    - And could even replace S
  - H is mandatory, W is optional
  - H determines W and decides
  - The form of W depends on H
  - The position of W is with refe

Some arbitrary rules

Milk and Cookies



- Conjugation is linear
- Periods attach to the verb

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
- This is a linear model with a certain kind of feature locality
  - Will not go into actual features that have been proposed

Projective Dependency Parsing

- Major assumption: edge

$$S\left(\tau\left(w_1^n\right)\right)$$

- A common form of edge

$$S(e.\tau) =$$

Projective Dependency Parsing

- $S\left(\tau(w_1^n)\right) = S(e$

  - Removing an edge *splits* the tr
    - Property of a tree
  - For our dependency graph, eve be written as: $e = (w_i, w_j)$

Projective Dependency Parsing

- $S\left(\tau(w_1^n)\right) = S(\epsilon$

  - Removing an edge *splits* the tr
    - Property of a tree
  - For our dependency graph, eve
    be written as: $e =$

This is sufficient to build a DP algorithm!

Basic idea

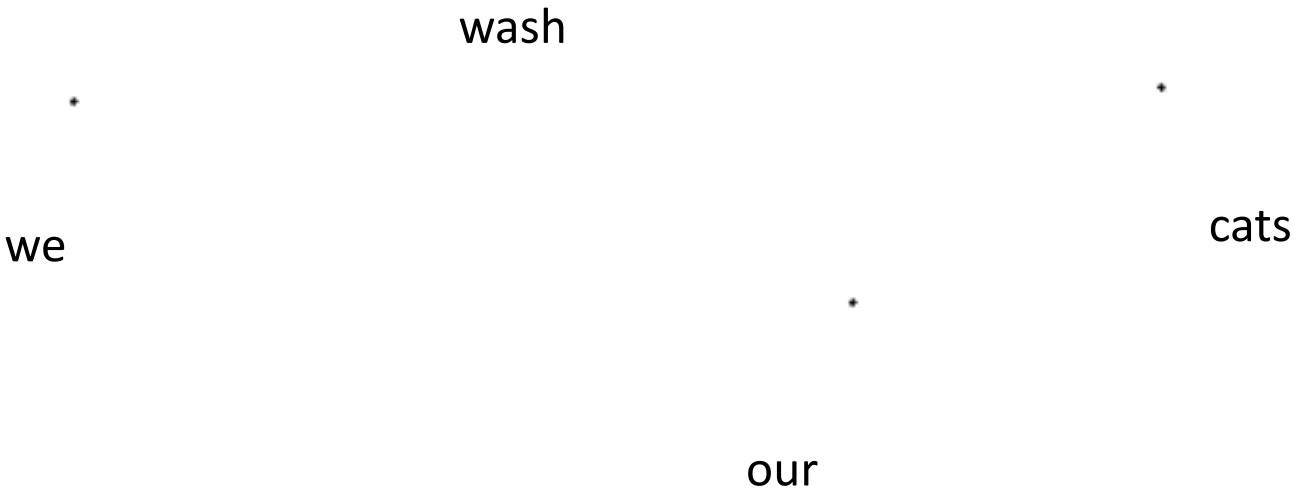
- Given many ways of partitioning a string into two subtrees, the merged tree that gives you the best combined score is the best overall tree
  - We build the DP on this idea

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

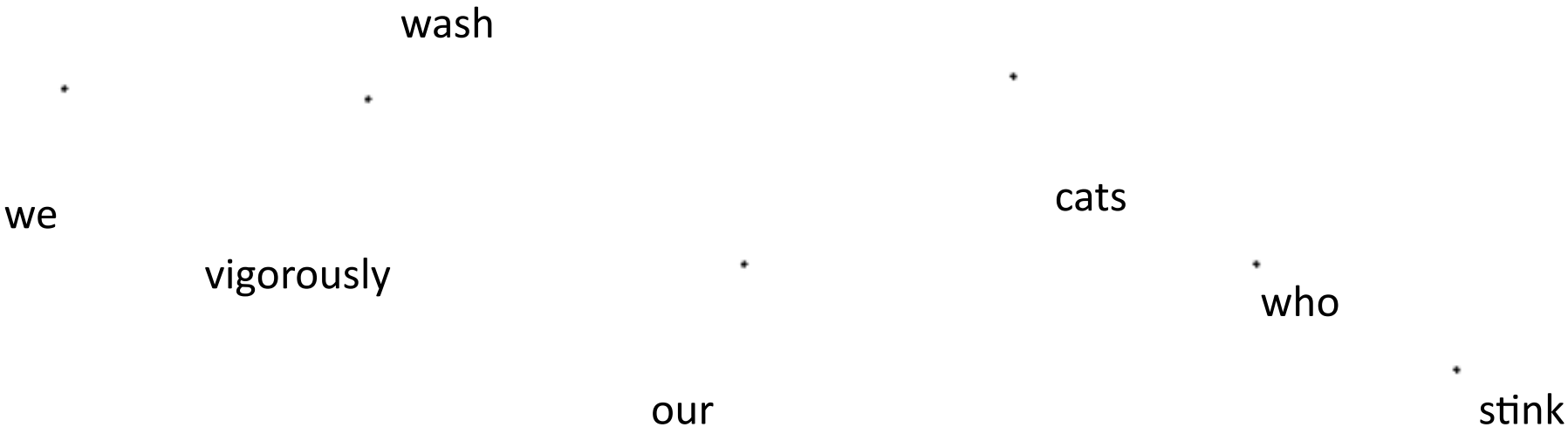
- We will talk about **bare bones** dependency trees (Eisner, 1996), then consider adding dependency labels

Bare Bones Dependency Parse

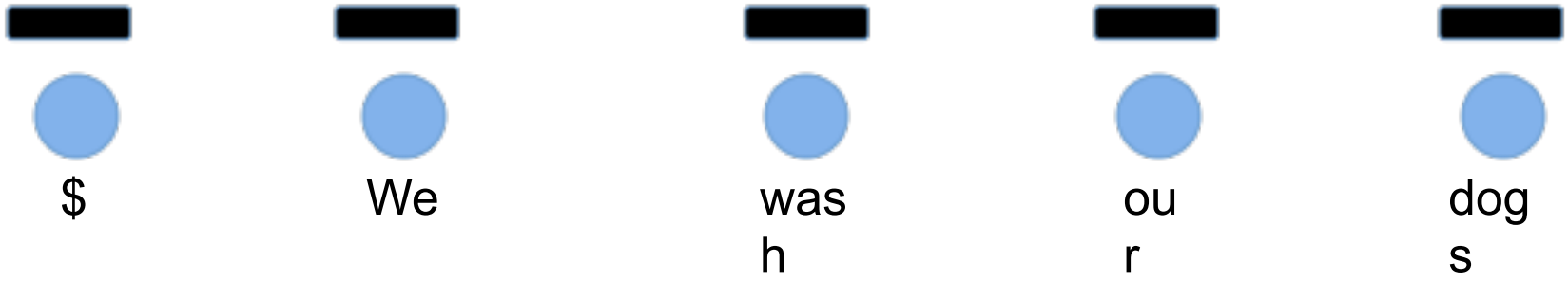


Bare Bones Dependency Parse

Bare Bones Dependency Parse

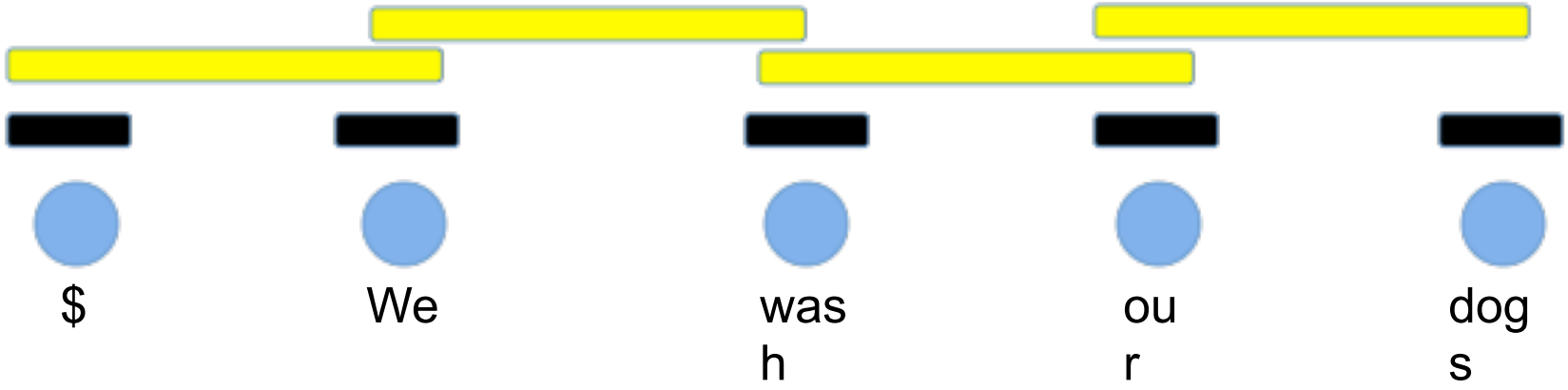


The naive algorithm



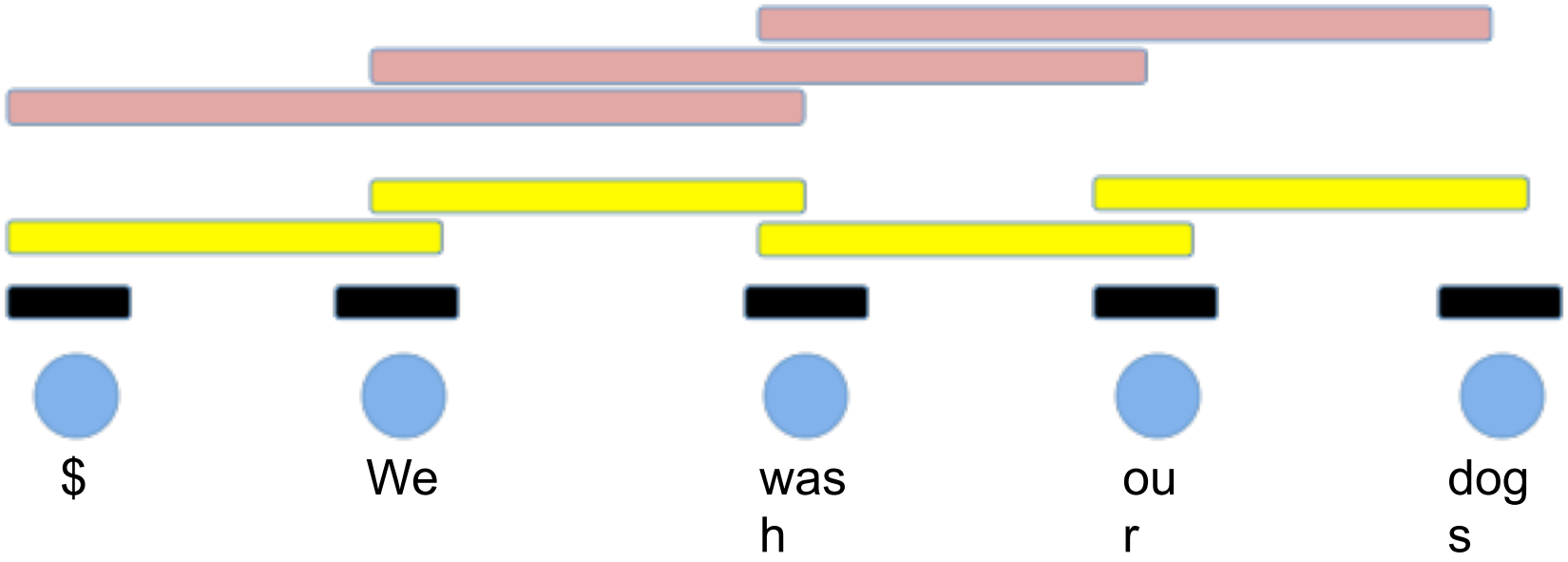
- For each possible "span" (length of string)
  - Starting from each position
    - Compute and save all possible trees (with different headwords) from lower-span trees

The naive algorithm



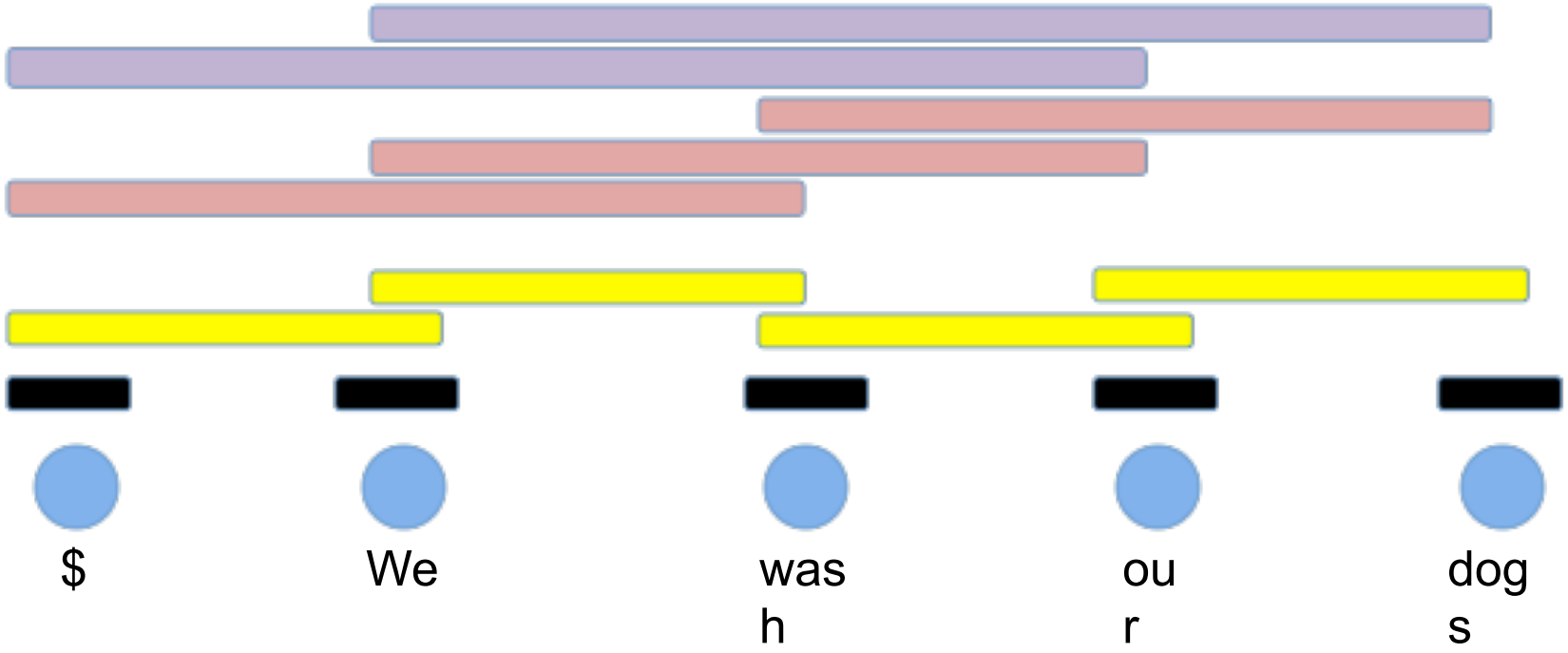
- For each possible "span" (length of string)
  - Starting from each position
    - Compute and save all possible trees (with different headwords) from lower-span trees

The naive algorithm



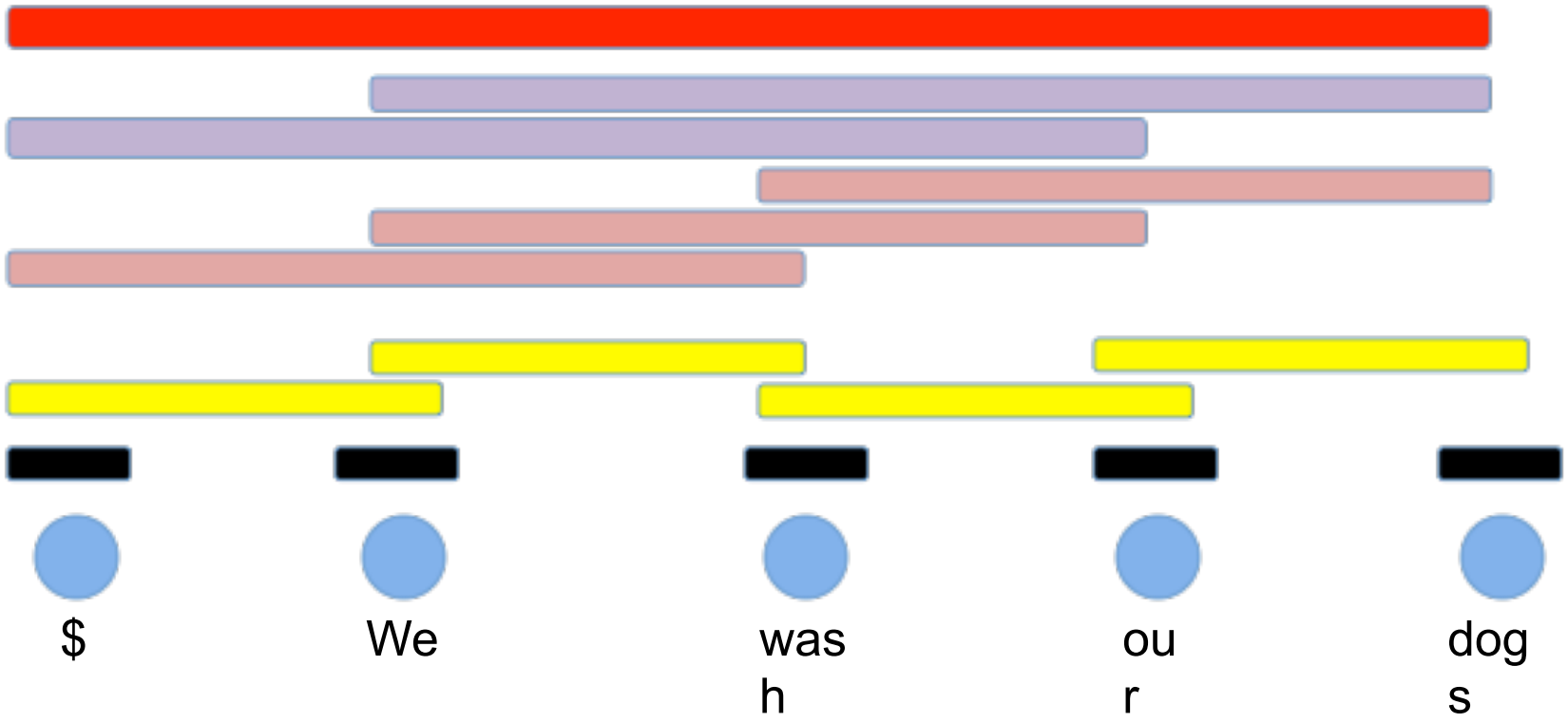
- For each possible "span" (length of string)
  - Starting from each position
    - Compute and save all possible trees (with different headwords) from lower-span trees

The naive algorithm



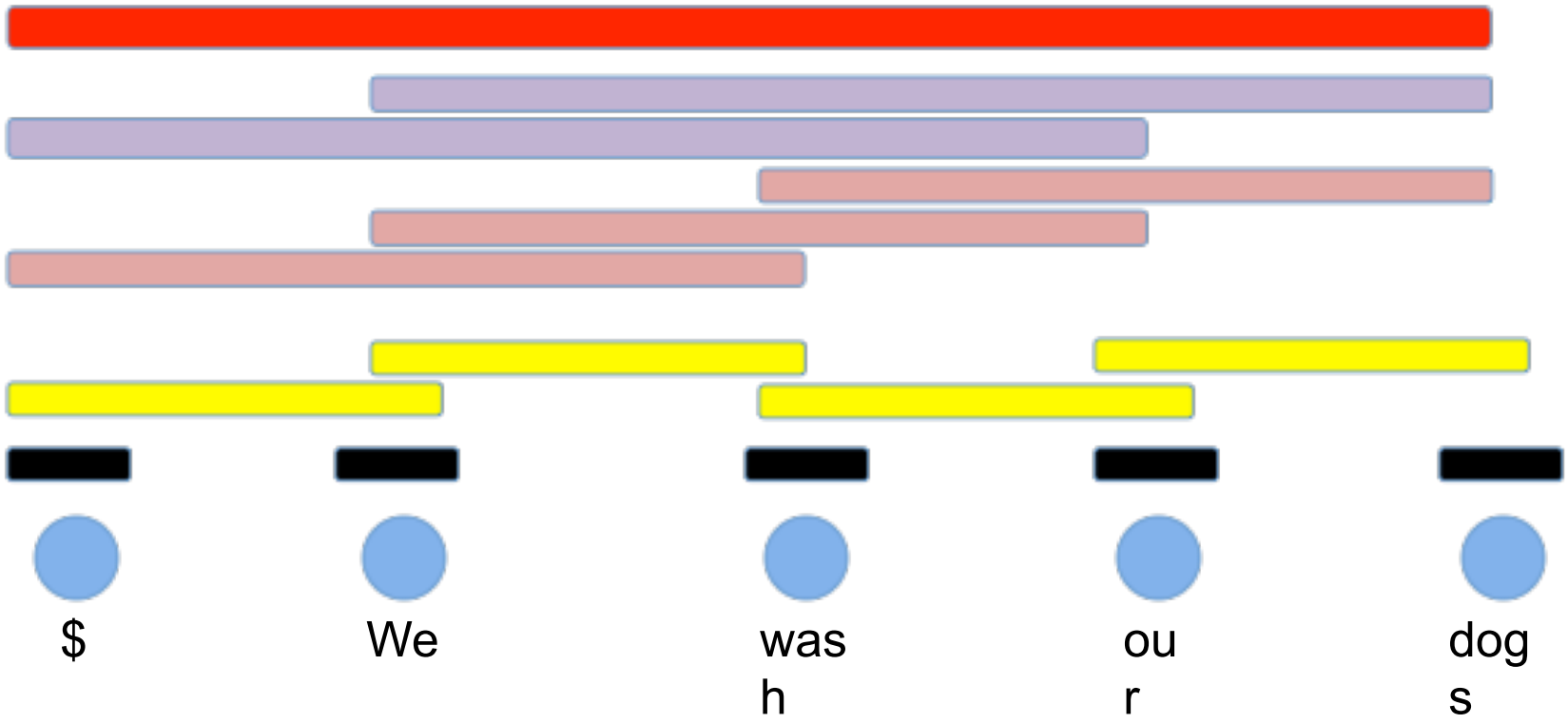
- For each possible "span" (length of string)
  - Starting from each position
    - Compute and save all possible trees (with different headwords) from lower-span trees

The naive algorithm



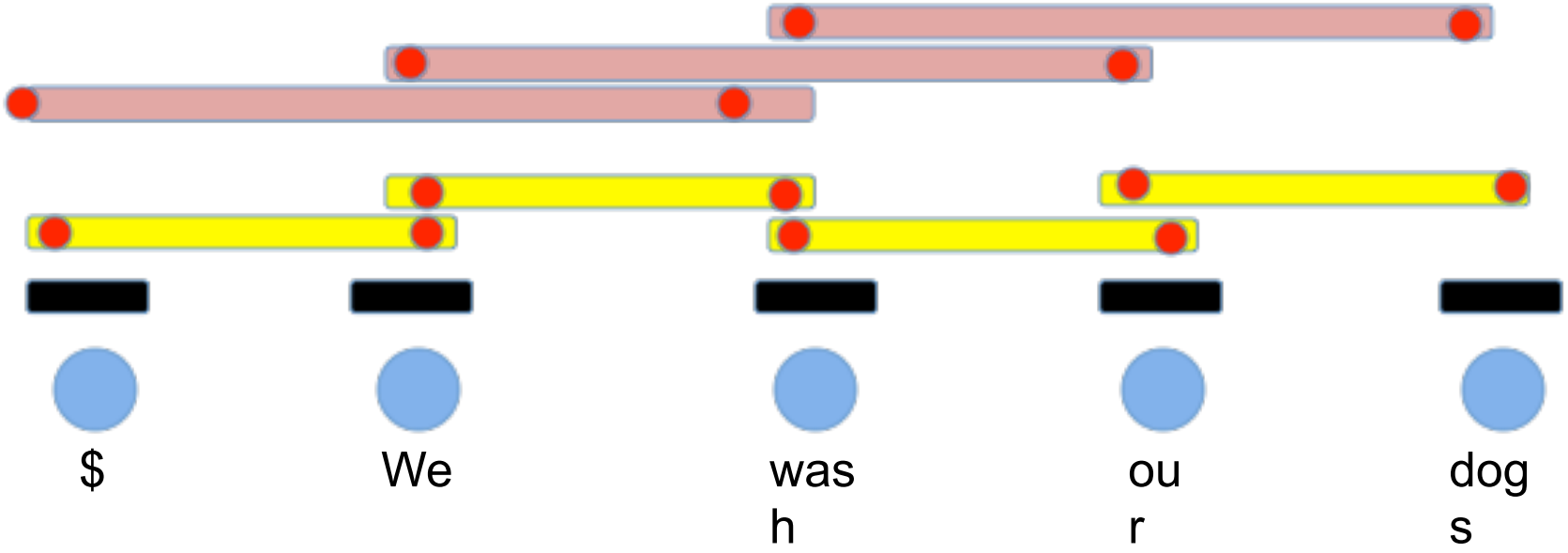
- For each possible "span" (length of string)
  - Starting from each position
    - Compute and save all possible trees (with different headwords) from lower-span trees

The naive algorithm



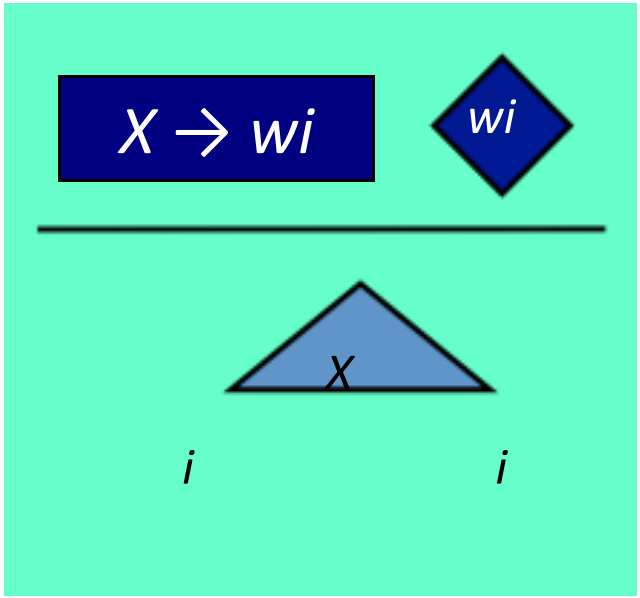
- Each span of K words ha
  - Representing K trees

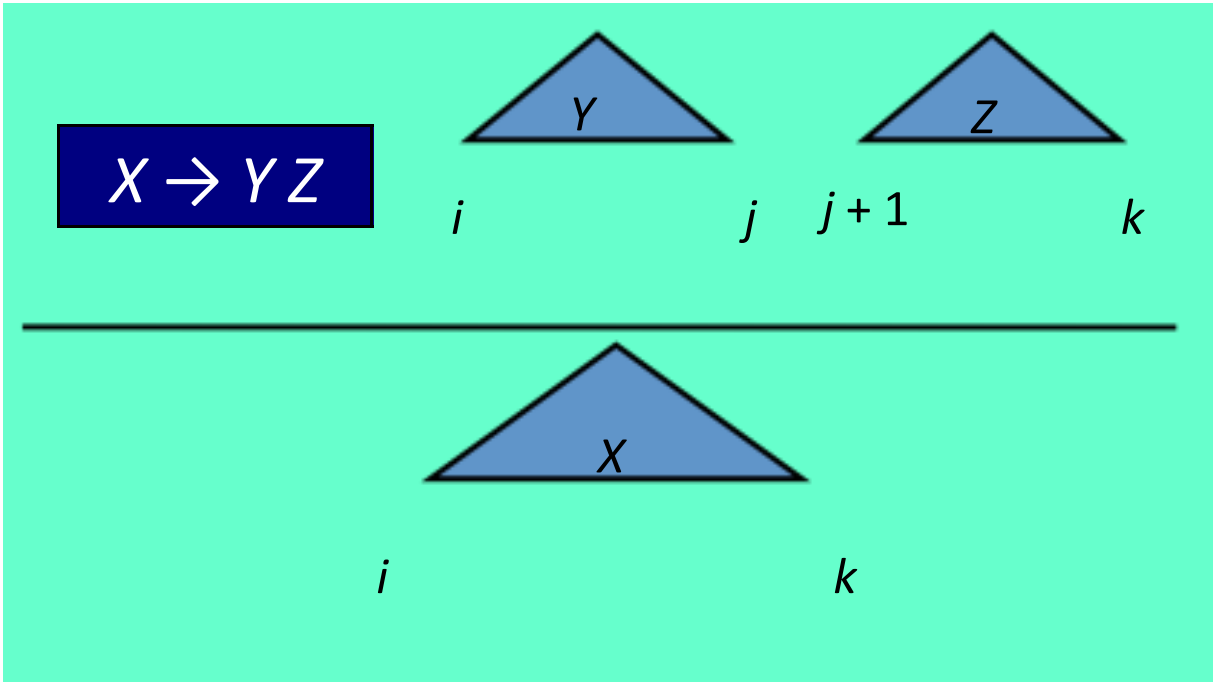
Eisner ‘96

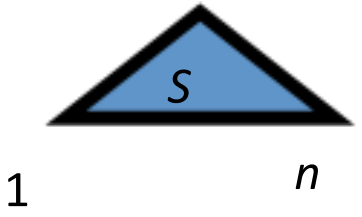


-
- Realization: You don’
evaluate all K possib

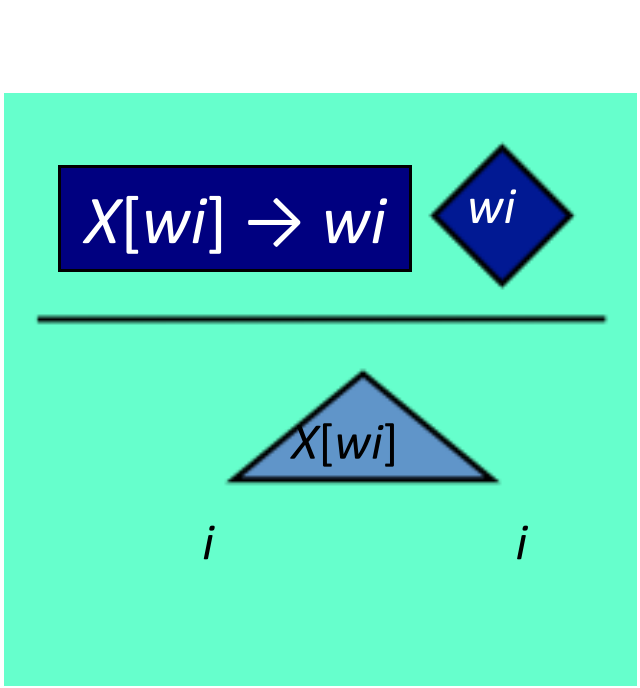
Remember CKY

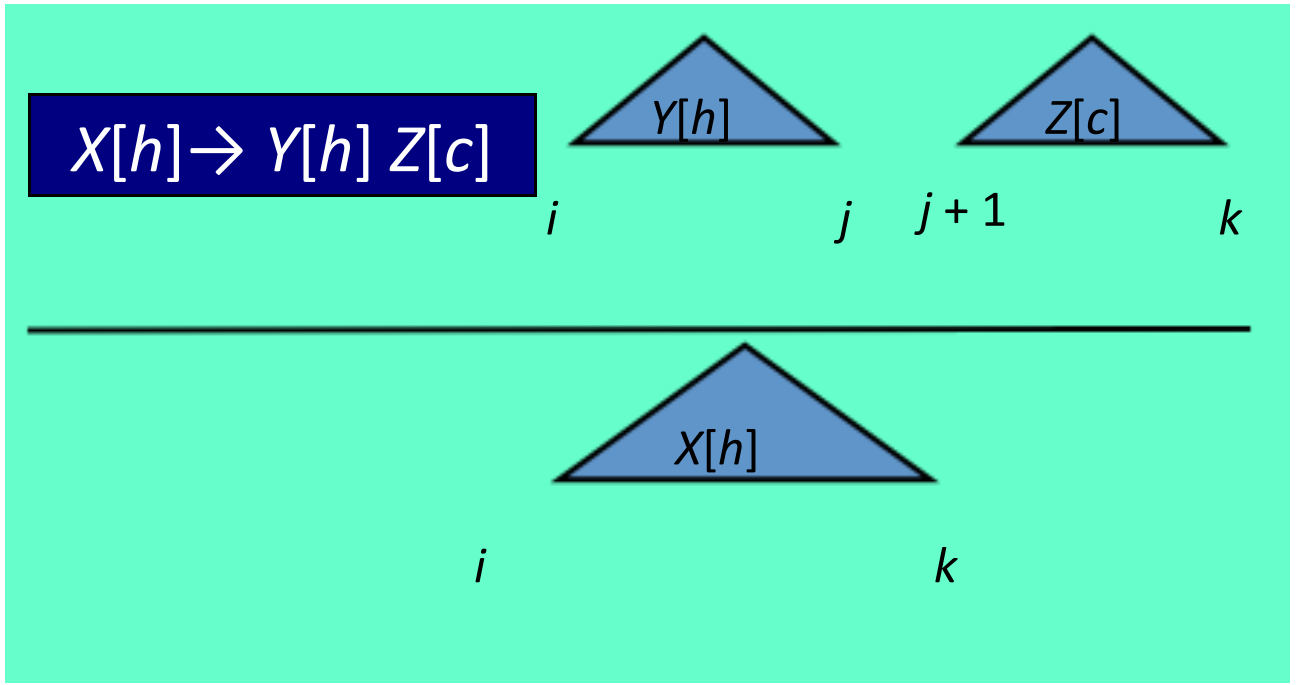


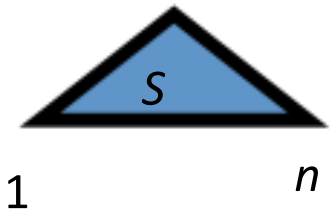




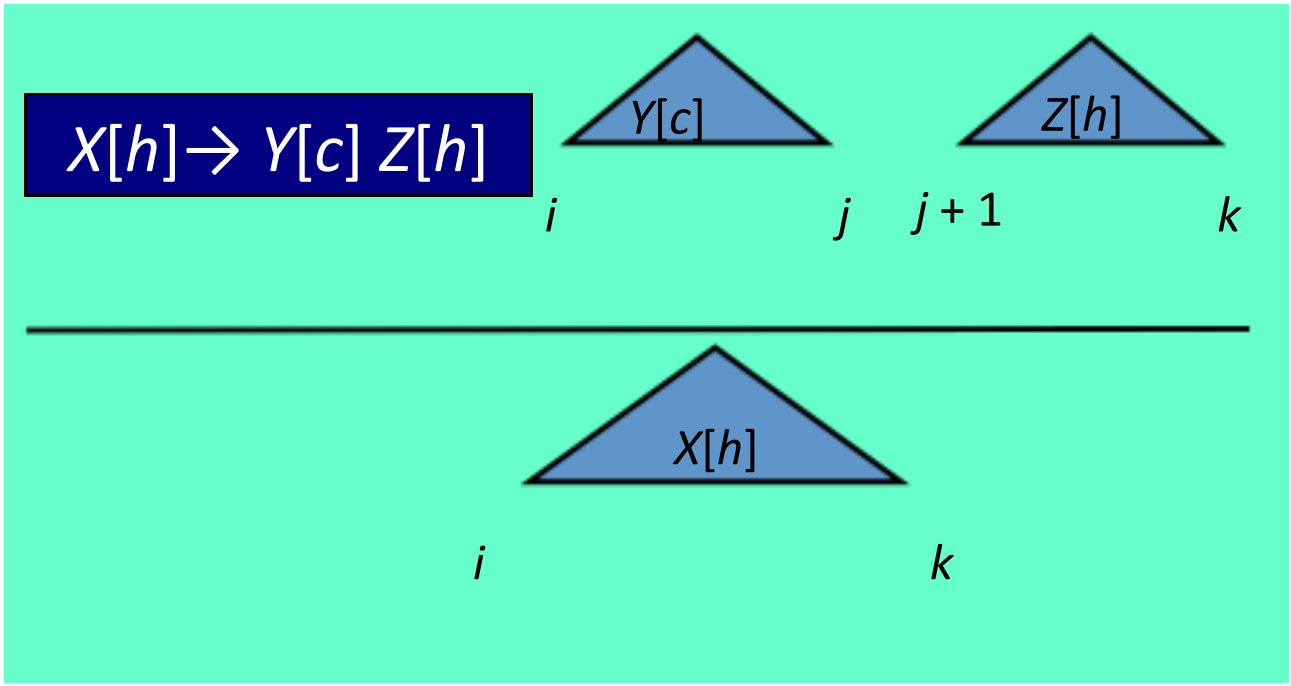
CKY with Heads



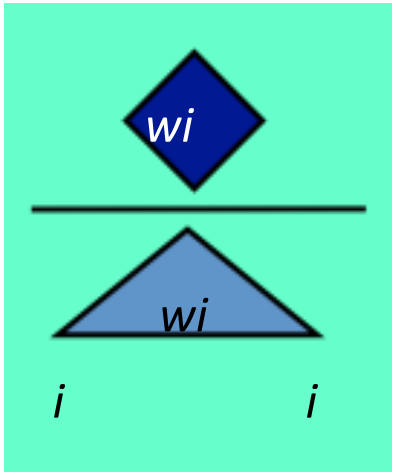


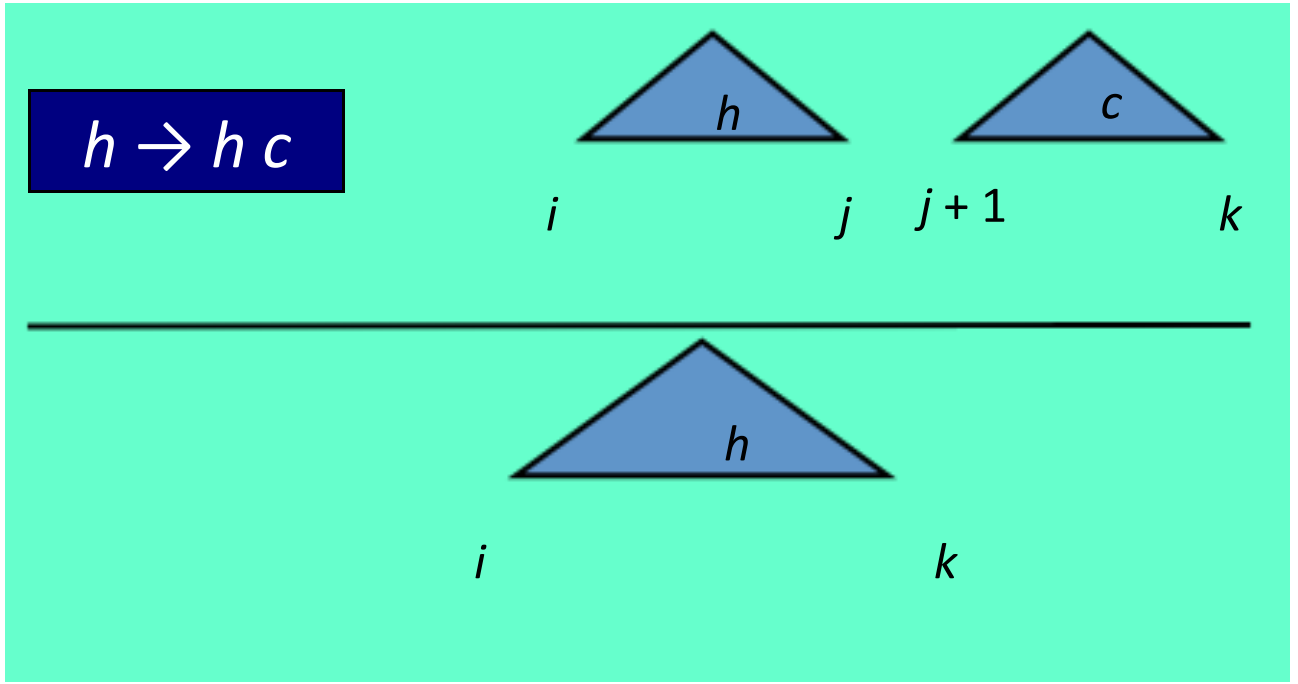


CKY with Heads (one more rule)



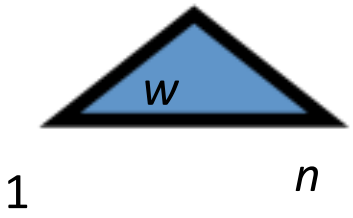
CKY with Heads, without Nonterminals





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

What's the runtime?



Eisner's Algorithm

- An instance of a *maximum spanning tree* algorithm
  - Or a minimum spanning tree, depending on whether we consider a score or a cost
- Note: The "root" symbol at the beginning
- Finds a spanning tree from the root that has the highest score (or probability)

Eisner's Algorithm

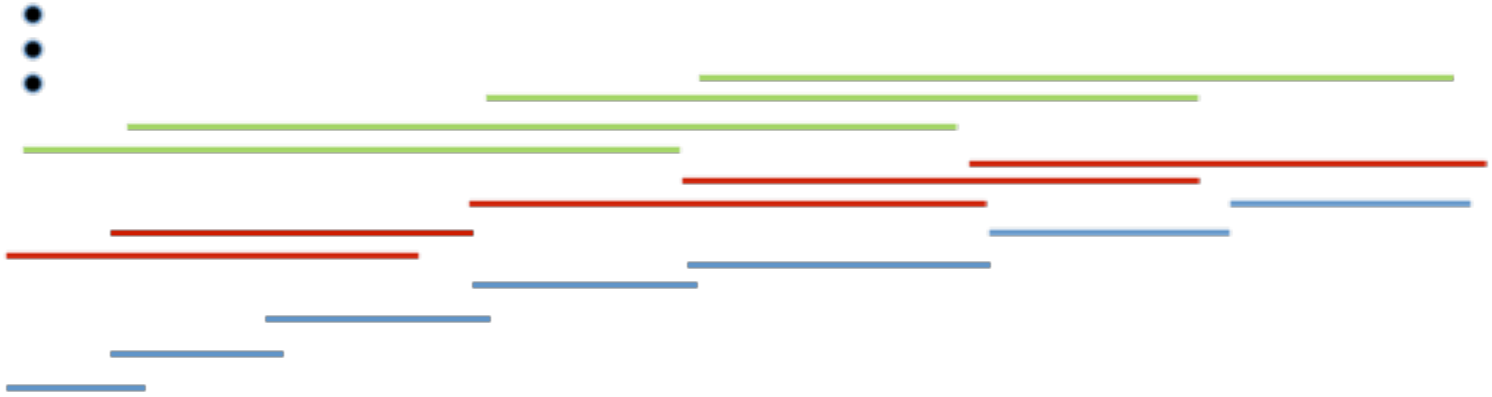
-
  - Notation:

    $C[i]$

  - Best cost of tree spanning

    $\xrightarrow{d}$: Root at i

Eisner's Algorithm



$ I SAW THE GIRL WITH A BOOK

- Recursive, for every span, find
  - Best left-to-right incomplete tree
  - Best right-to-left incomplete tree
  - Best left-to-right complete tree
  - Best right-to-left complete tree

Eisner's Algorithm

for $i : 0 \ldots n$ **and all** d, c:

$C[i][i][d][c] = 0$

for $l: 1 \ldots n$:

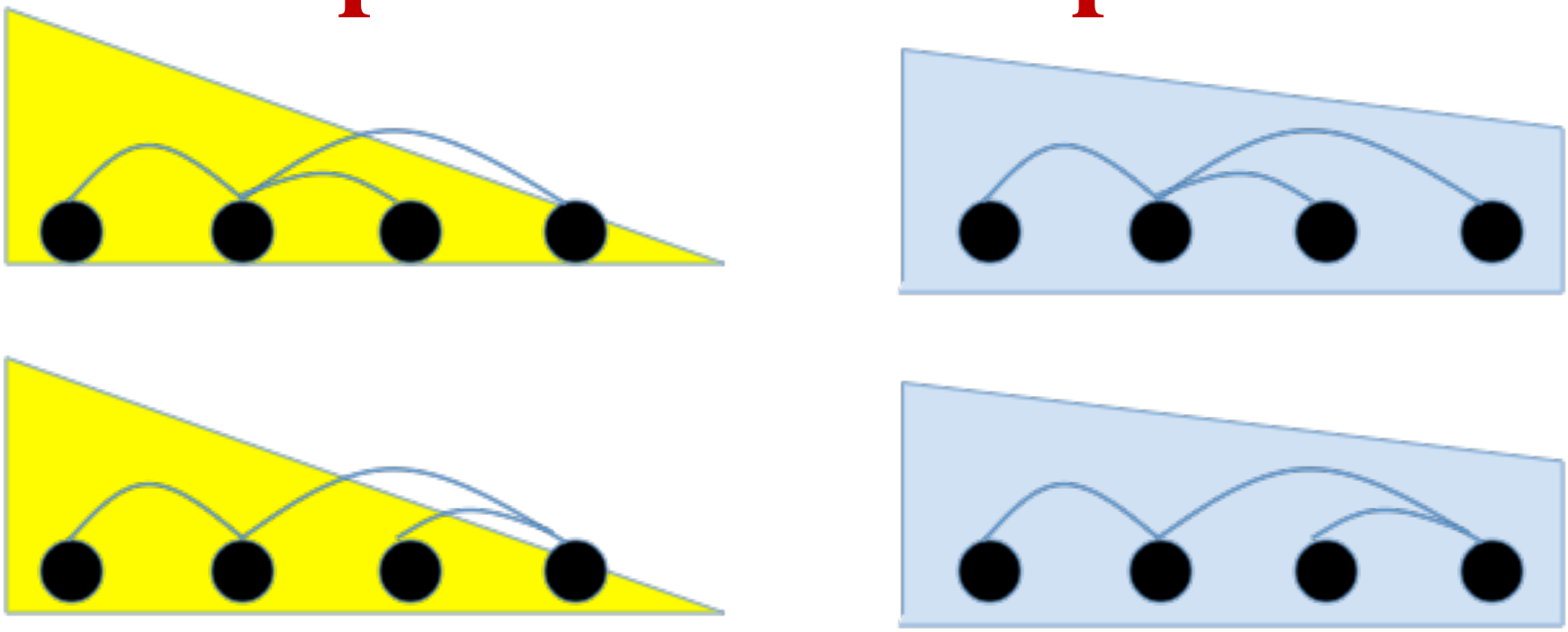
for $s: 0 \ldots n - l$:

$t = s + l$

$C[s][t][\rightarrow][0] = \max C[s][i]$

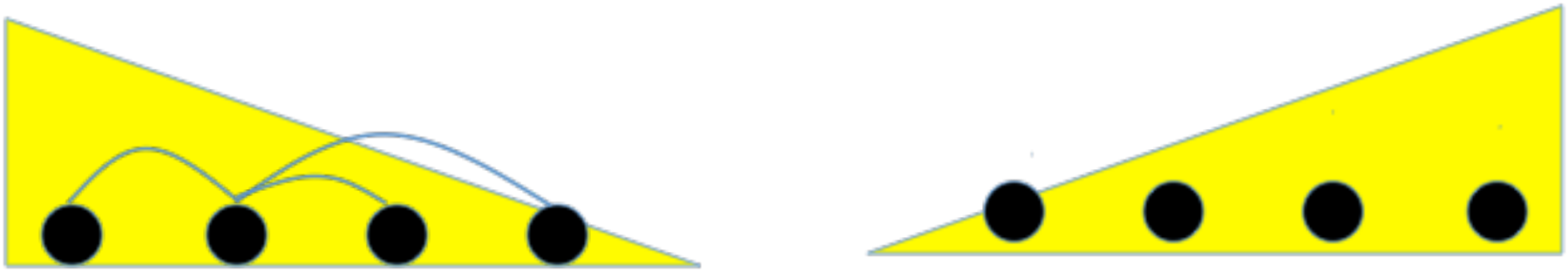
- Note dynamic programming structure
- Uses best length-N trees to find best length N+1 trees

Explanation with pictures



- Right triangle: Complete tree, with root at left
- Right trapezium: Incomplete tree with root at left
- Left triangle and left trapezium are corresponding structures with the root to the

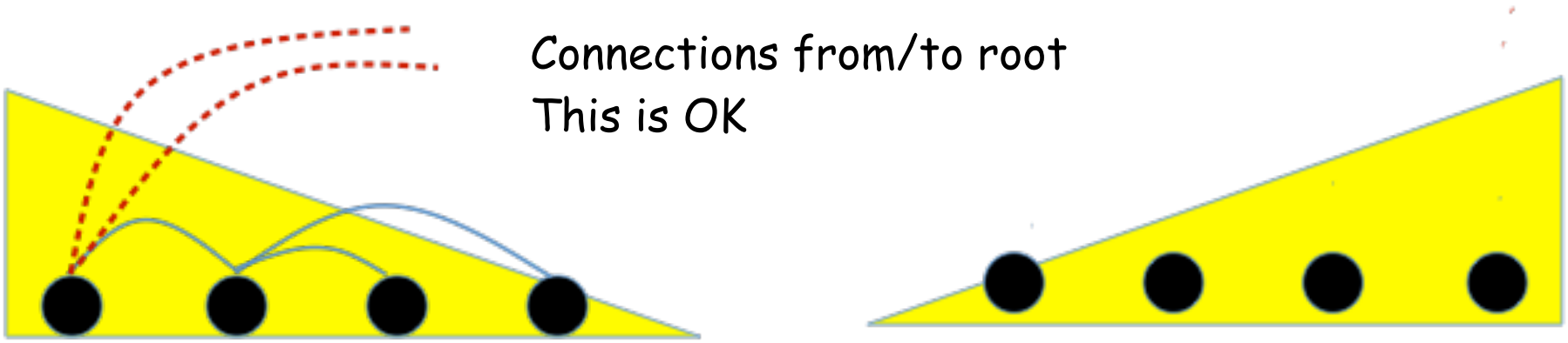
Explanation with pictures



- Complete right subtree: Cannot take additional right children
  - Can have arbitrary substructure within subtree
  - **Root to extreme left**

- Complete left subtree: Cannot take additional

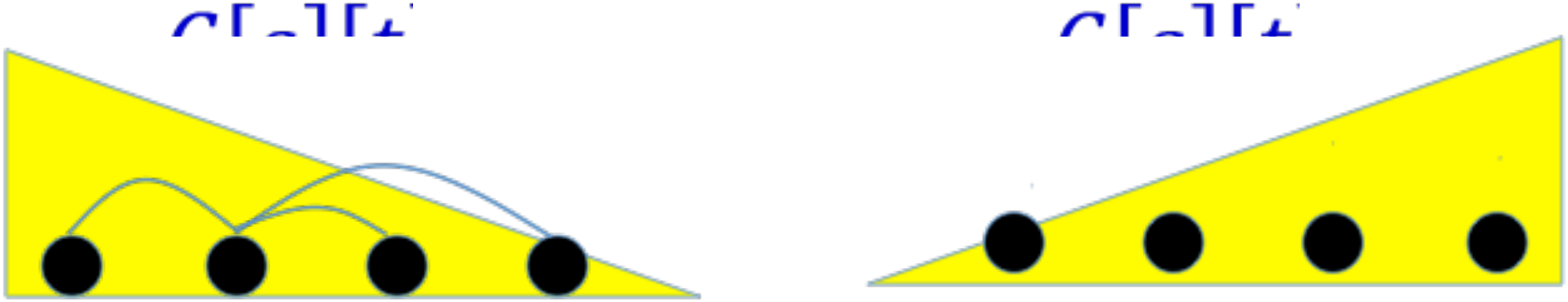
Explanation with pictures



- Complete right subtree: Cannot take additional right children
  - Can have arbitrary substructure within subtree
  - **Root to extreme left**

- Complete left subtree: Cannot take additional

Explanation with pictures



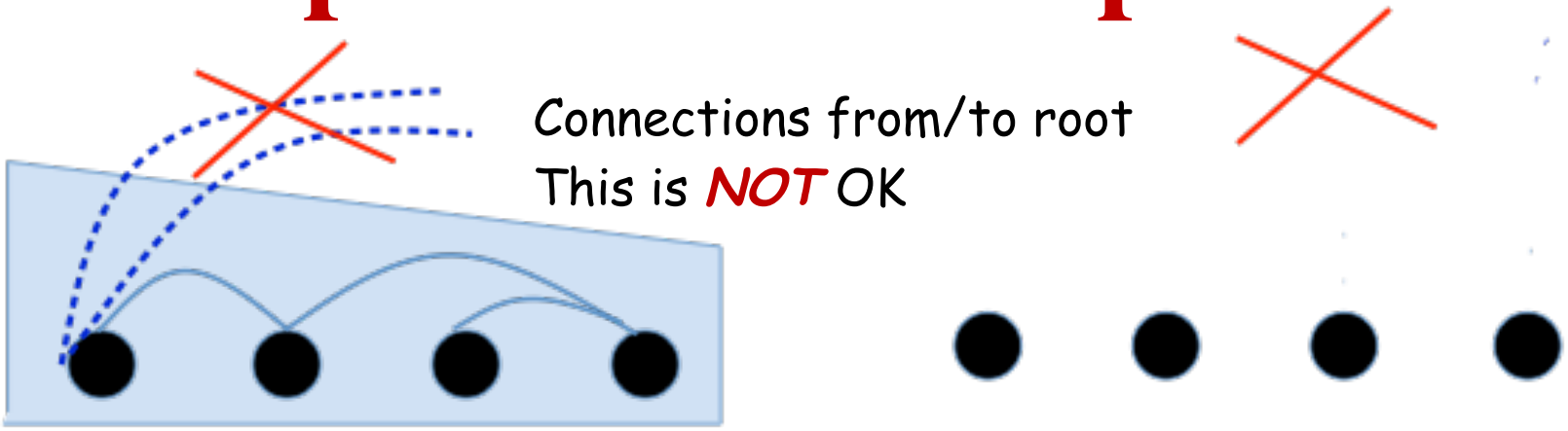
- Note representation of best score

Explanation with pictures

- Incomplete right subtree: Can extend to the right from rightmost node
  - But not from any other node
  - Root at extreme left

- Incomplete left subtree: Can extend to the left

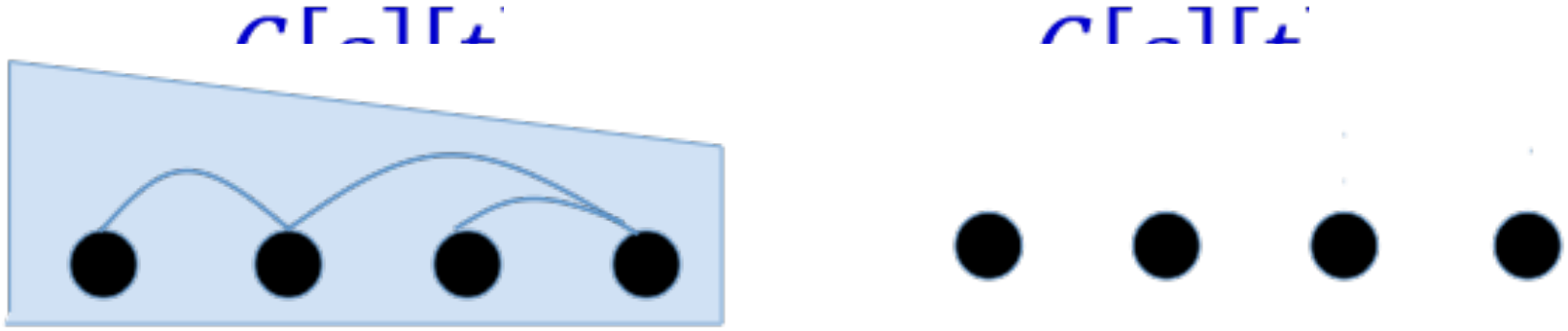
Explanation with pictures



- Incomplete right subtree: Can extend to the right from rightmost node
  - But not from any other node
  - Root at extreme left

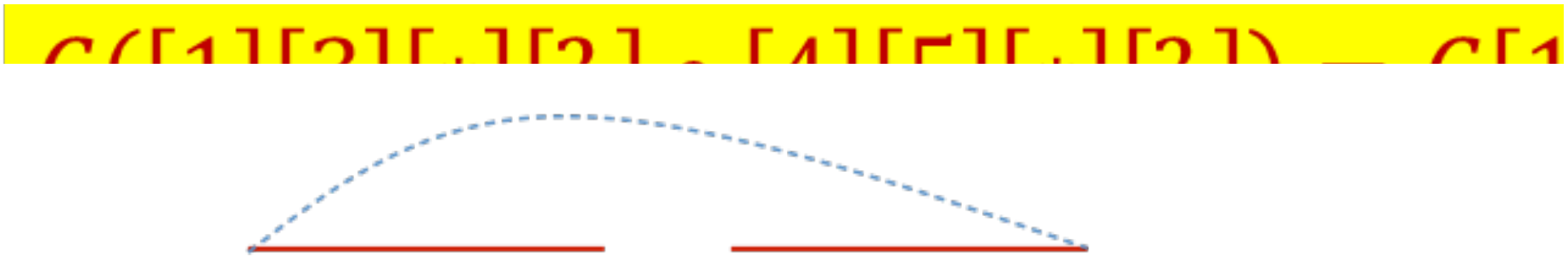
- Incomplete left subtree: Can extend to the left

Explanation with pictures



- Note representation of best score

Logic behind recursion



$ I SAW THE GIRL WITH A BOOK

Case 1: Connecting the heads of two spans one word apart

- By our model: The score of connecting two spans is the sum of the score of the individual spans plus the score of the connection

Logic behind recursion

$C[1][5][*][?] = \max C[1][i][$

$ I SAW THE GIRL WITH A BOOK

Connecting the heads of two spans one word apart

- The *best* score for co

Logic behind recursion

$ I SAW THE GIRL WITH A BOOK

Case 2: Connecting two spans at a boundary

- By our model: The score of connecting two spans is the sum of the scores of the individual spans

Logic behind recursion

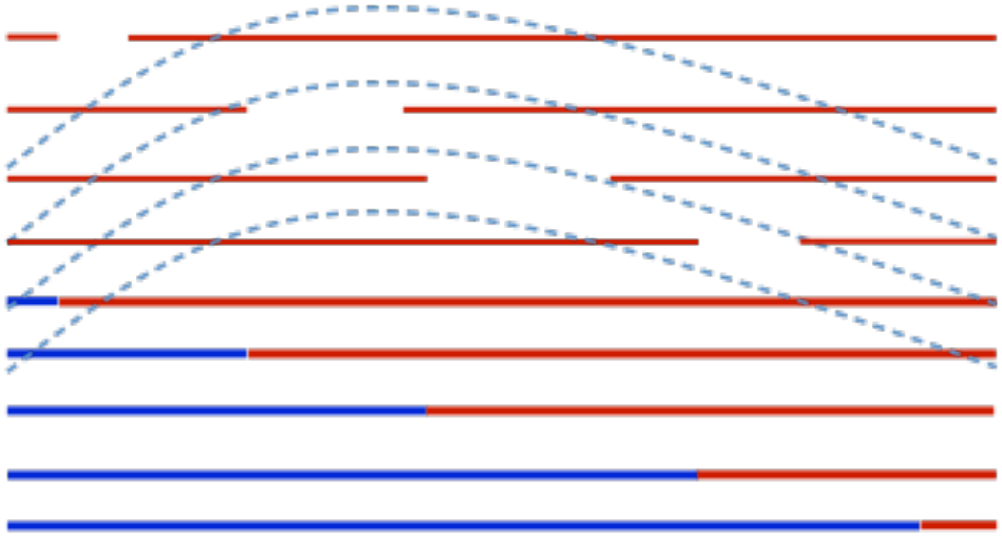
$C[1][5][*][?] = \max C$

$ I SAW THE GIRL WITH A BOOK

Connecting the heads of two spans one word apart

-
  - The *best* score for cor

Logic behind recursion

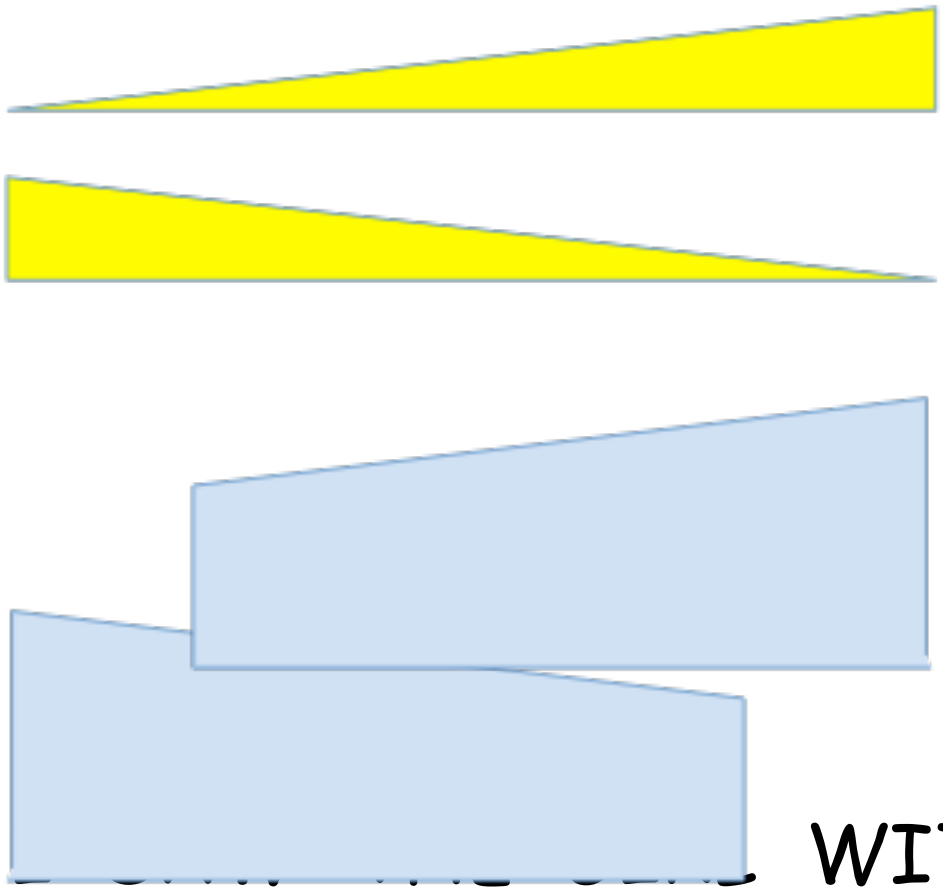


$ I SAW THE GIRL WITH A BOOK

Connecting the heads of two spans one word apart

- All possible ways of forming the span

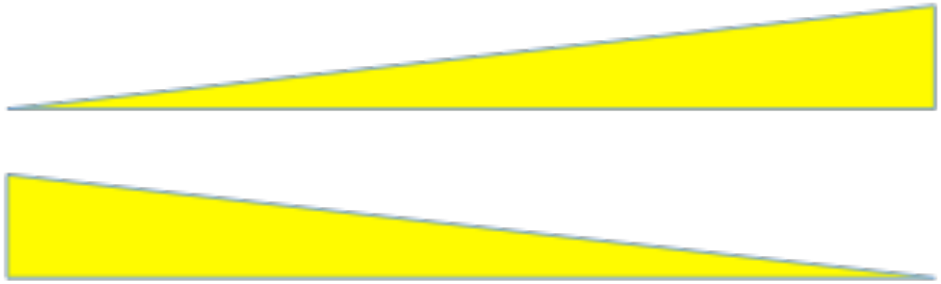
In figures



$ WITH A BOOK

- There are four kinds of subtrees we can compose within any span with the root at a boundary
- Lets see how we can compose these

In figures



If we do this right, we never have to worry about composing any subtree with a root in the middle

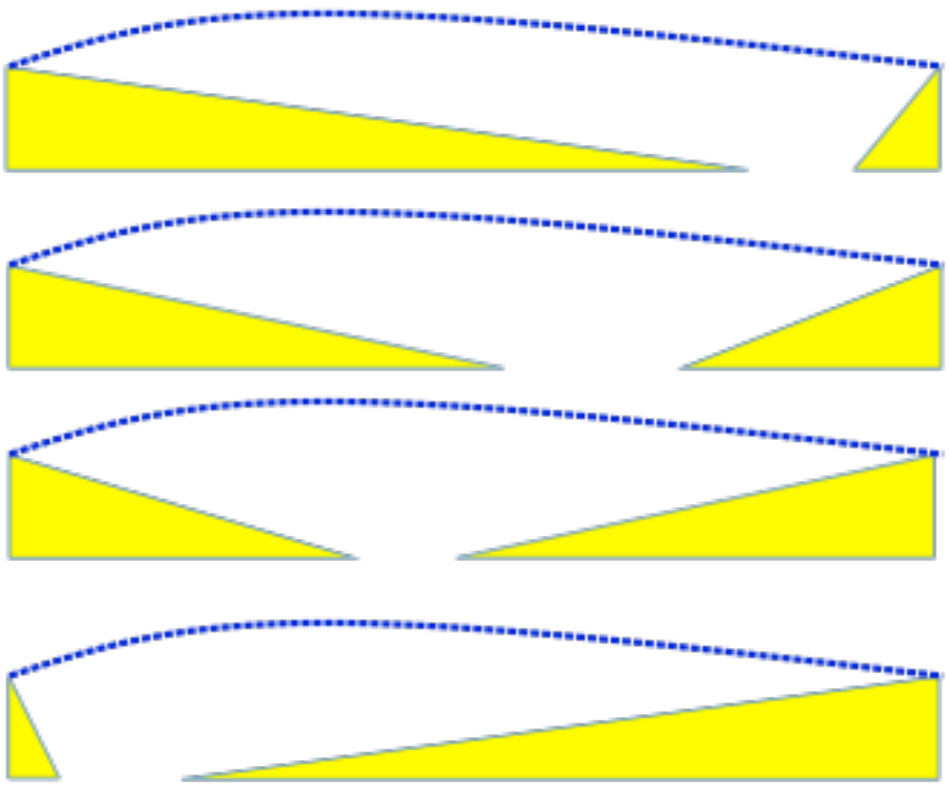
It will naturally fall out of the algorithm

$ I SAW THE GIRL WITH A BOOK

- There are four kinds of subtrees we can compose within any span with the root at a boundary
- Lets see how we can compose these

$$C[s][t][\rightarrow][0] = \max C[s][i][\rightarrow]$$

Left to right



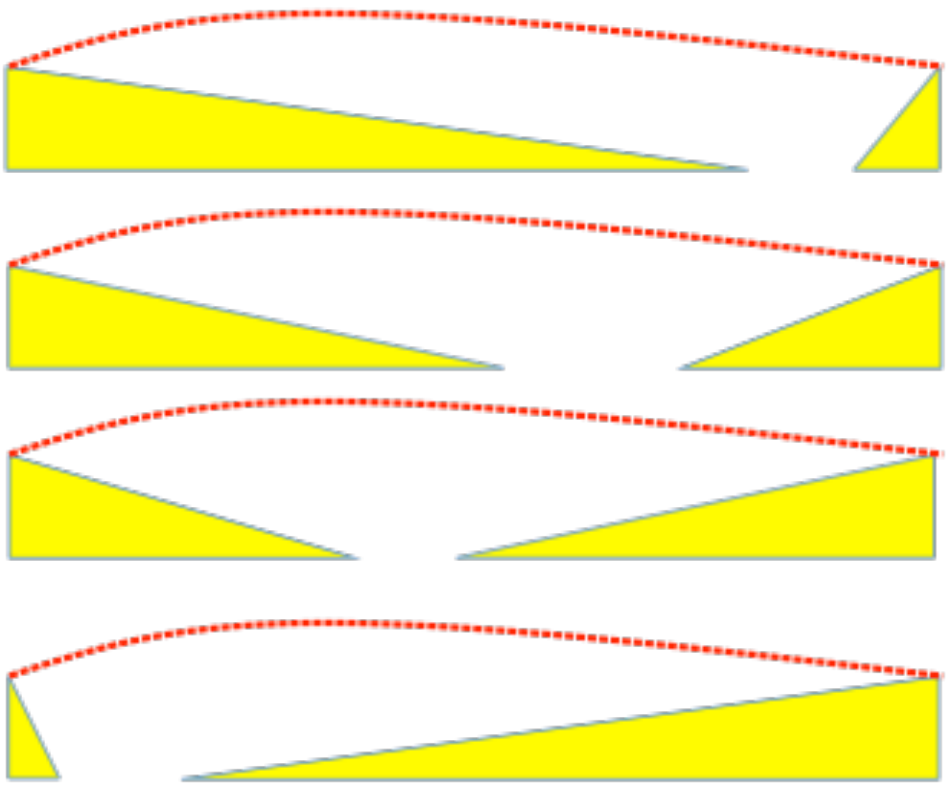
Select the best one

$ I SAW THE GIRL WITH A BOOK

Outcome

- Connecting two completed trees to get an incomplete tree

$C[s][t][\leftarrow][0] = \max C[s][i][\rightarrow]$



Right to left

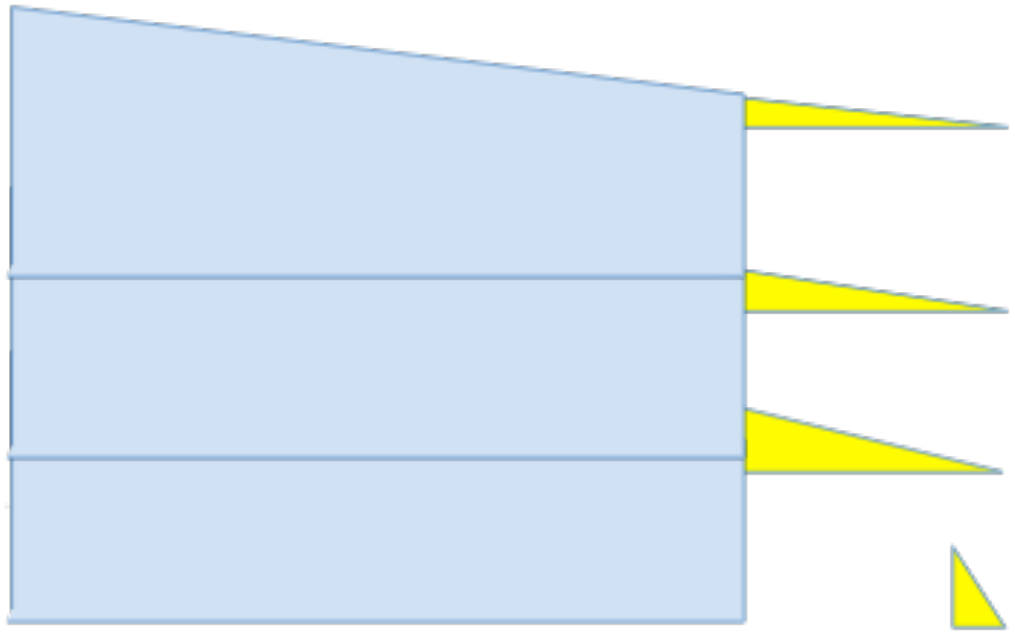
Select the best one

$ I SAW THE GIRL WITH A BOOK

Outcome

- Connecting two completed trees to get an incomplete tree

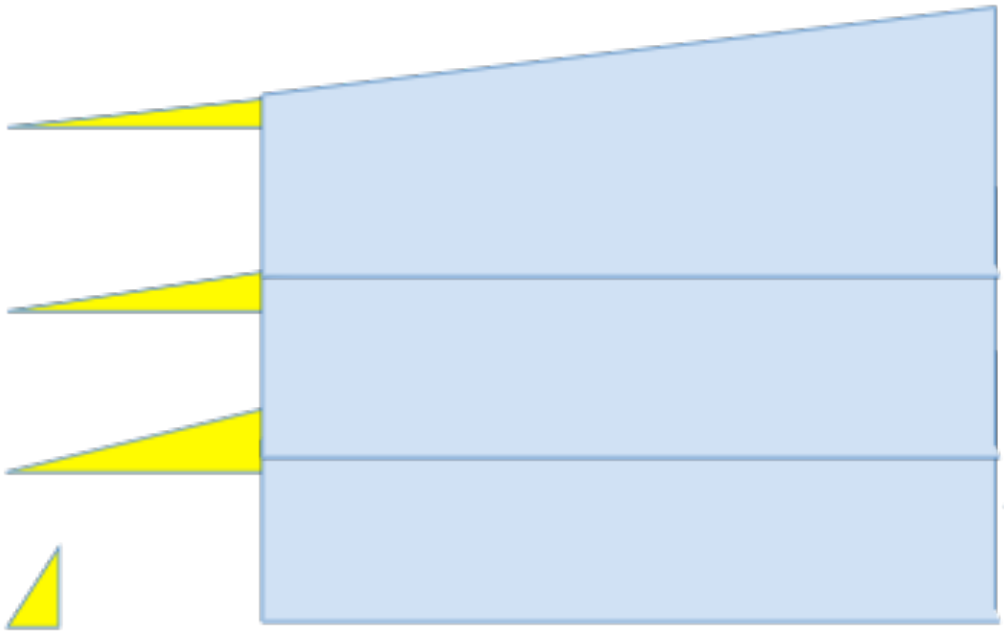
$C[s][t][\rightarrow][1] = \max C[$



$ I SAW THE GIRL WITH A BOOK

- Connecting an incomplete tree and a complete tree to get a complete tree

$C[s][t][\leftarrow][1] = \max \; C[$



$ I SAW THE GIRL WITH A BOOK

- Connecting an incomplete tree and a complete tree to get a complete tree

An Example

```
$ I SAW THE GIRL WITH A BOOK
```

- The sequence of operations

Eisner's Algorithm

for $i : 0 \ldots n$ **and all** d, c:

$C[i][i][d][c] = 0$

for $l: 1 \ldots n$:

for $s: 0 \ldots n - l$:

$t = s + l$

$C[s][t][\rightarrow][0] = \max C[s][i]$

- Note dynamic programming structure
- Uses best length-N trees to find best length N+1 trees

Eisner's Algorithm

for $i : 0 \ldots n$ **and all** d, c:

$C[i][i][d][c] = 0$

for $l: 1 \ldots n$:

for $s: 0 \ldots n - l$:

$t = s + l$

$C[s][t][\rightarrow][0] = \max C[s][i]$

- Note dynamic programming structure
- Uses best length-N trees to find best length N+1 trees

An Example

$ I SAW THE GIRL WITH A BOOK

- The sequence of operations
- Level 0: every word is both a left and right directed tree with score 0

Eisner's Algorithm

for $i : 0 \ldots n$ **and all** d, c:

$C[i][i][d][c] = 0$

for $l : 1 \ldots n$:

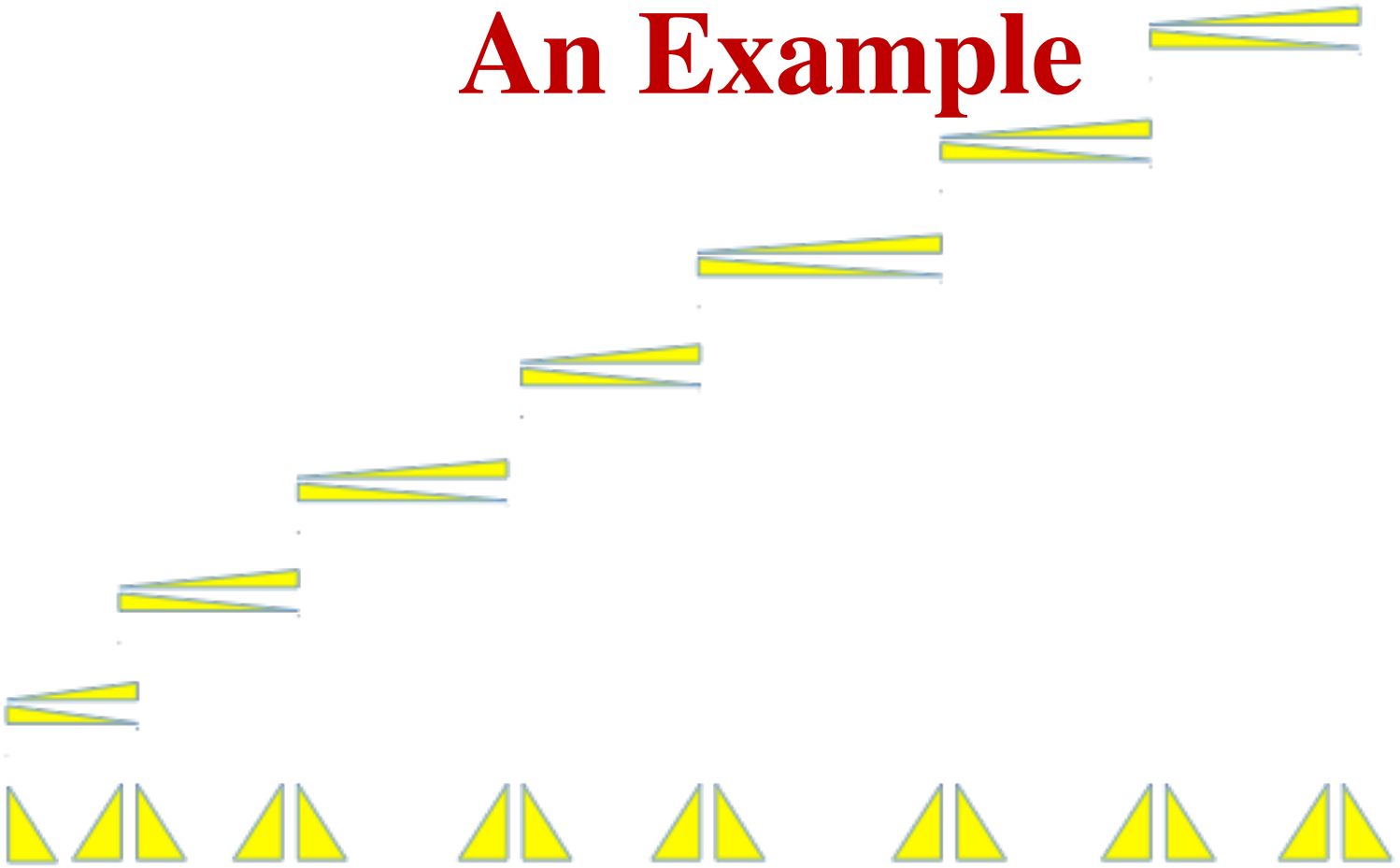
for $s : 0 \ldots n - l$:

$t = s + l$

$C[s][t][\rightarrow][0] = \max C[s][i]$

- Note dynamic programming structure
- Uses best length-N trees to find best length N+1 trees

An Example



- Level 1: Using level 0 trees, compose two-word span trees from every word
  - Left incomplete tree

Eisner's Algorithm

for $i : 0 \ldots n$ **and all** d, c:

$C[i][i][d][c] = 0$

for $l : 1 \ldots n$:

for $s : 0 \ldots n - l$:

$t = s + l$

$C[s][t][\rightarrow][0] = \max C[s][i]$

- Note dynamic programming structure
- Uses best length-N trees to find best length N+1 trees

An Example

- Level 1: Using level 0 and level 1 trees, compose all four types of three-word span trees, starting at every word

Eisner's Algorithm

for $i : 0 \ldots n$ **and all** d, c:

$C[i][i][d][c] = 0$

for $l : 1 \ldots n$:

for $s : 0 \ldots n - l$:

$t = s + l$

$C[s][t][\rightarrow][0] = \max C[s][i]$

- Note dynamic programming structure
- Uses best length-N trees to find best length N+1 trees

An Example

$ I SAW THE GIRL WITH A BOOK

- Level 1: Using level 0, level 1 and level 2 trees, compose all four types of four-word span trees, starting at every word

An Example

$ I SAW THE GIRL WITH A BOOK

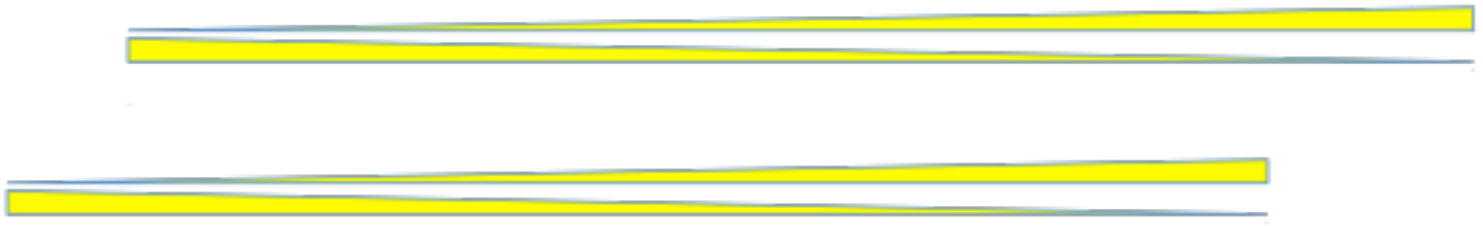
- Construct all five-word trees from lower-level trees, starting at each position
  - As the length of the tree increases, the number of

An Example

$ I SAW THE GIRL WITH A BOOK

- Recursively increase the span
  - Increasing span decreases the number of trees

An Example



$ I SAW THE GIRL WITH A BOOK

- Recursively increase the span
  - Increasing span decreases the number of trees

An Example

$ I SAW THE GIRL WITH A BOOK

- Recursively increase the span
  - Increasing span decreases the number of trees

An Example

$ I SAW THE GIRL WITH A BOOK

- Eventually we should have a single complete LR tree, starting from the initial $ (the root)
  - If we don't, the parse has failed

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

$ The professor chuckled with unabashed glee

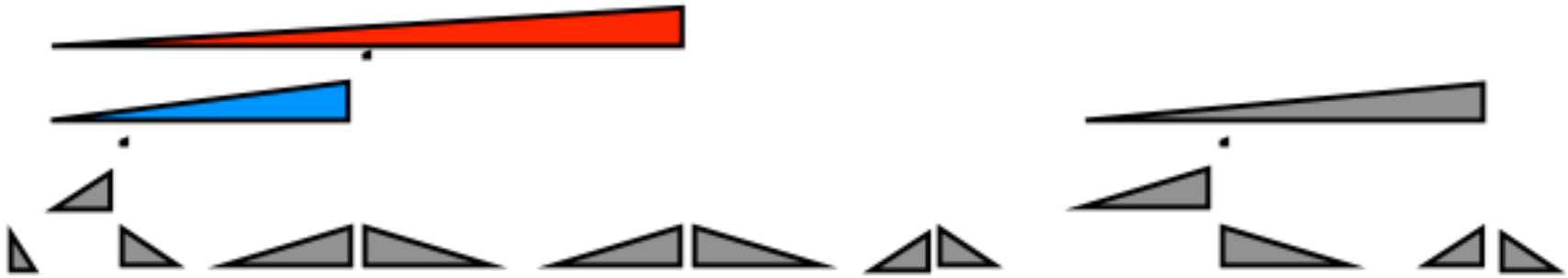
Example of the Eisner Algorithm

Complete:

$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

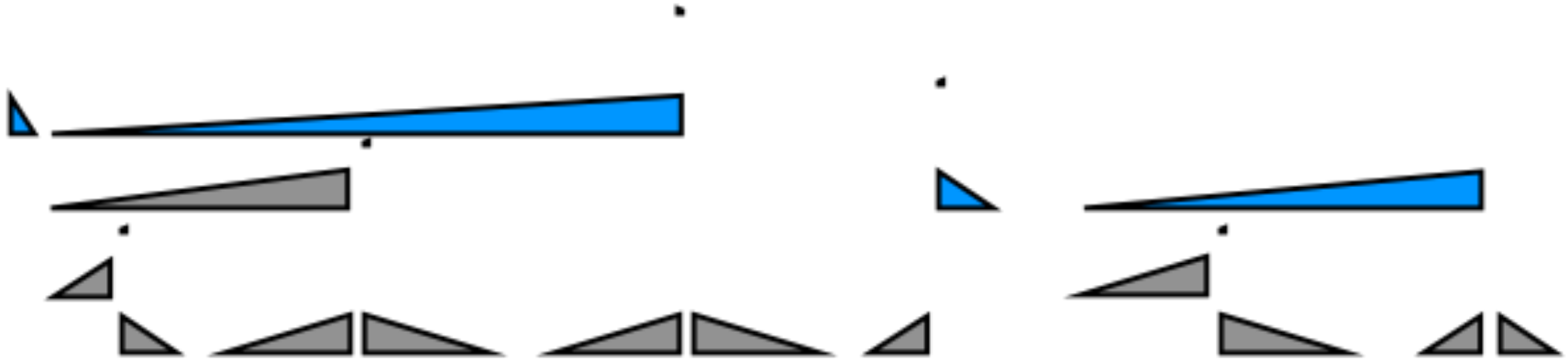
Complete:



$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

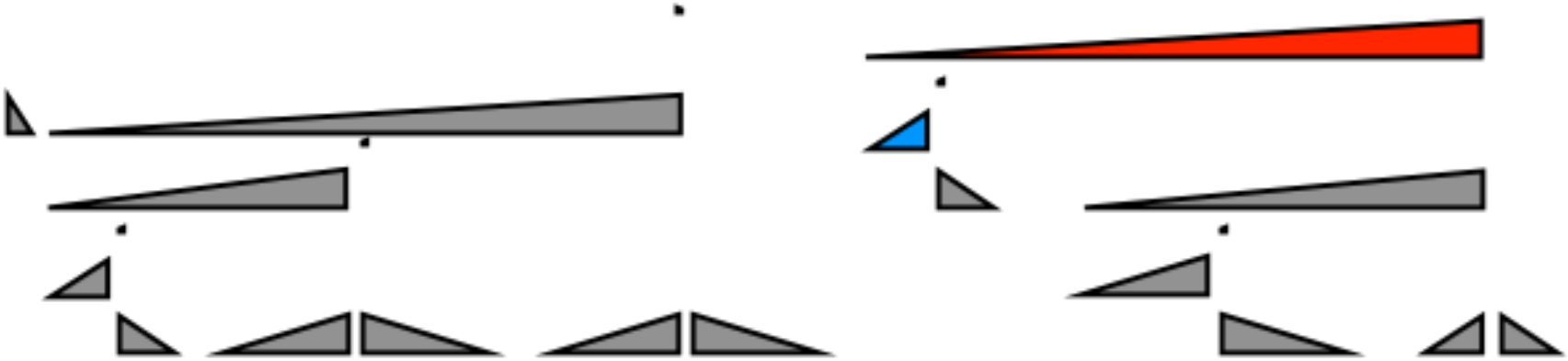
Attach:



$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

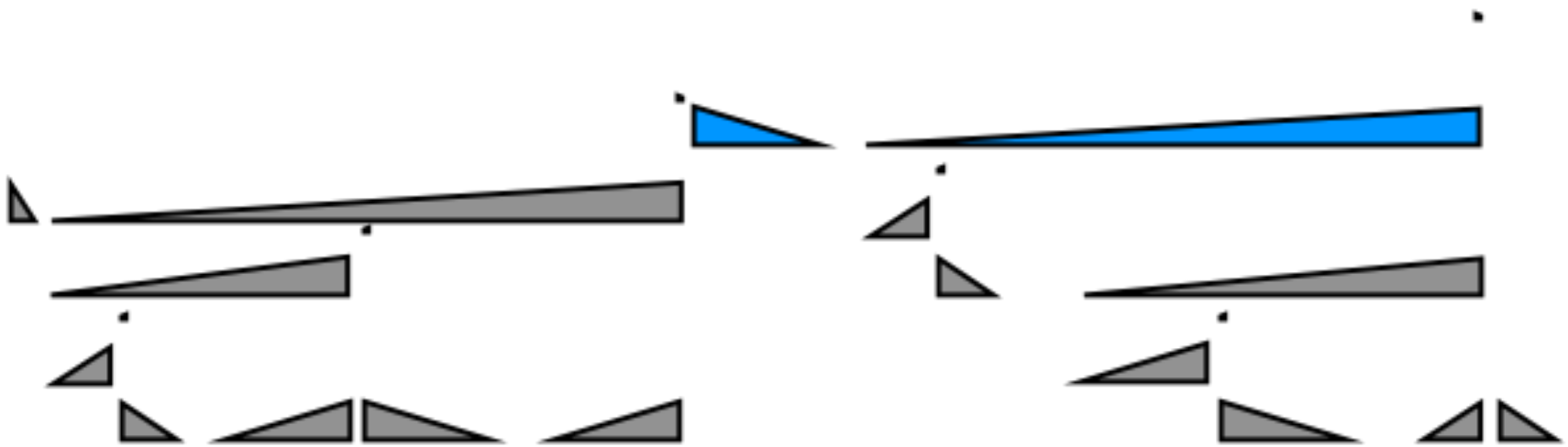
Complete:



$ The professor chuckled with unabashed glee

Example of the Eisner Algorithm

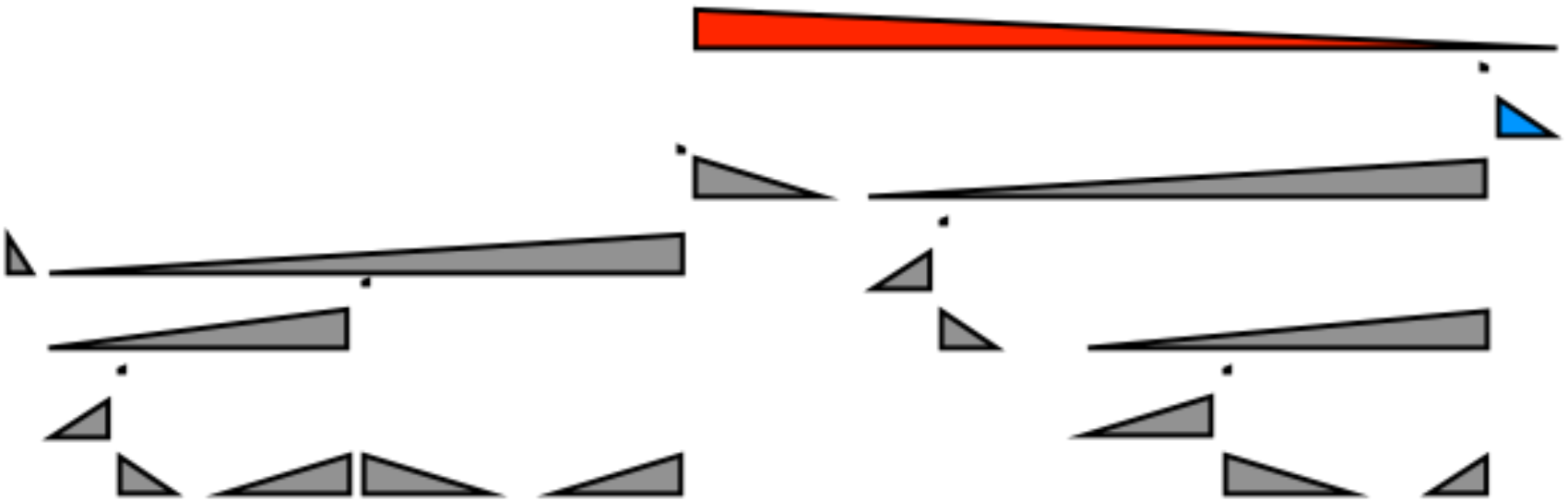
Attach:



$ The professor chuckled with unabashed glee

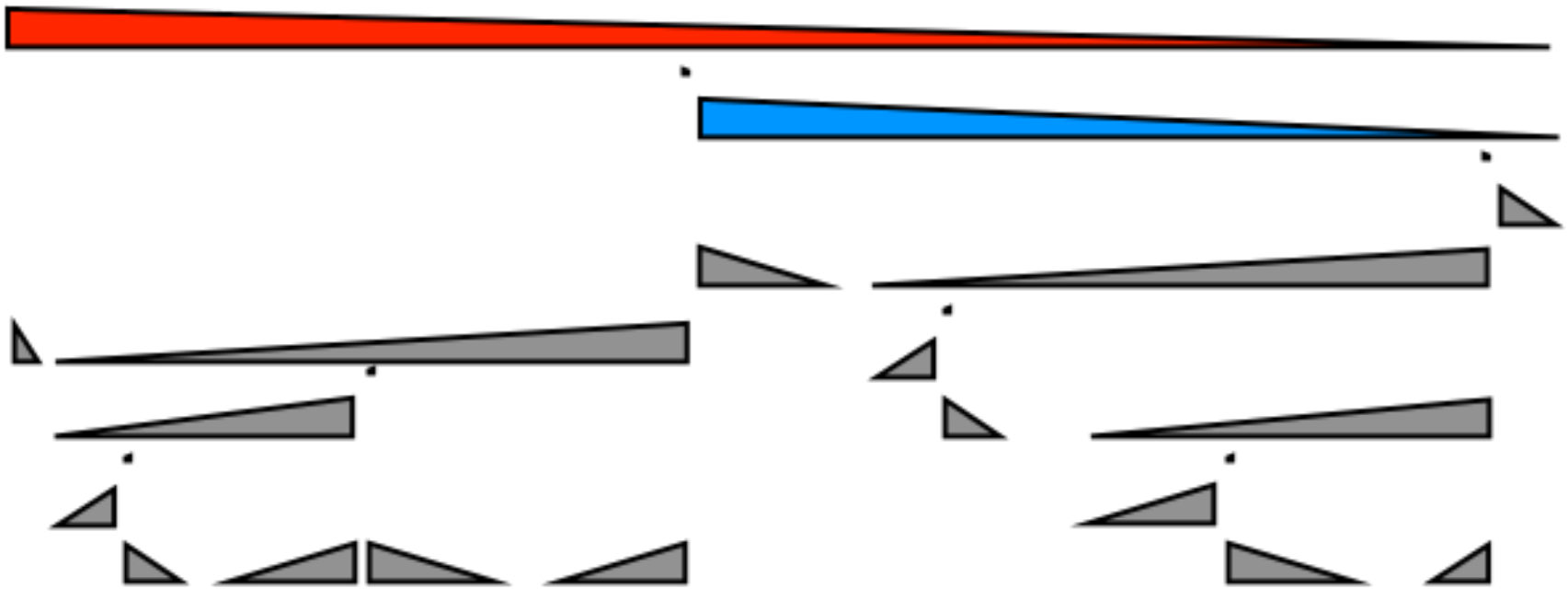
Example of the Eisner Algorithm

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

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

Evaluation

- Attachment accuracy
  - Labeled
  - Unlabeled

Bottomline

- Rethinking the algorithm in terms of attachments rather than constituents gives us an asymptotic savings!
- Bare bones, projective dependency parsing is O(n3)

- 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)
  - 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

CoNLL Tasks

- Dependency parsing is now more popular than constituency parsing
  - Just showed a very basic framework
  - But much current work builds on similar frameworks

- 2006 and 2007: dependency parsing on a variety of languages was the shared task at CoNLL - a few dozen systems.