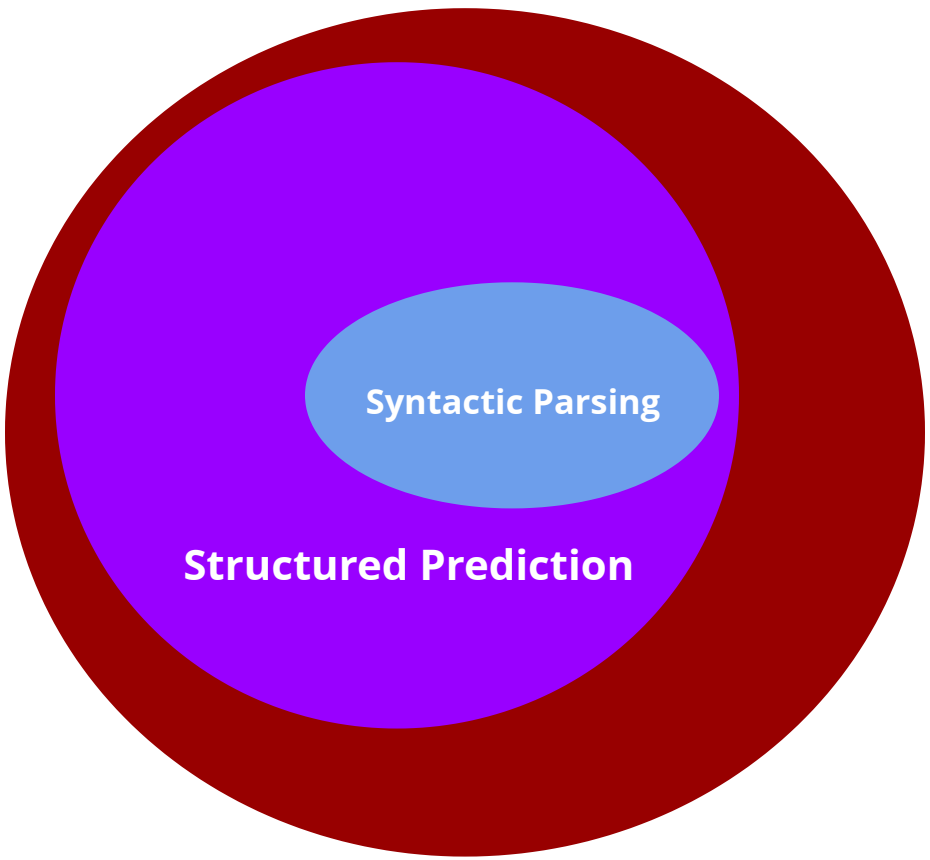
50.040
Natural Language Processing

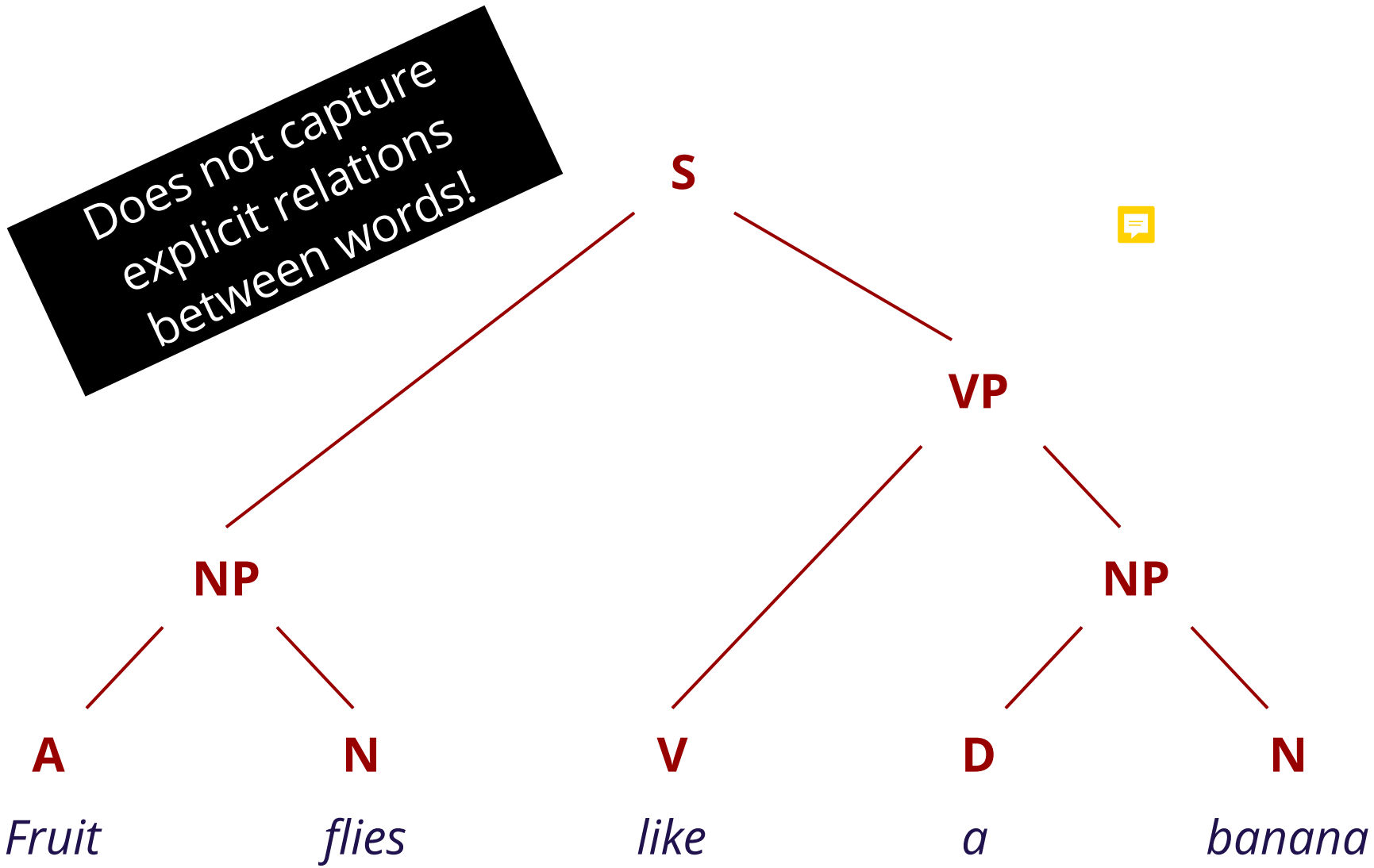
Lu, Wei

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Tasks in NLP

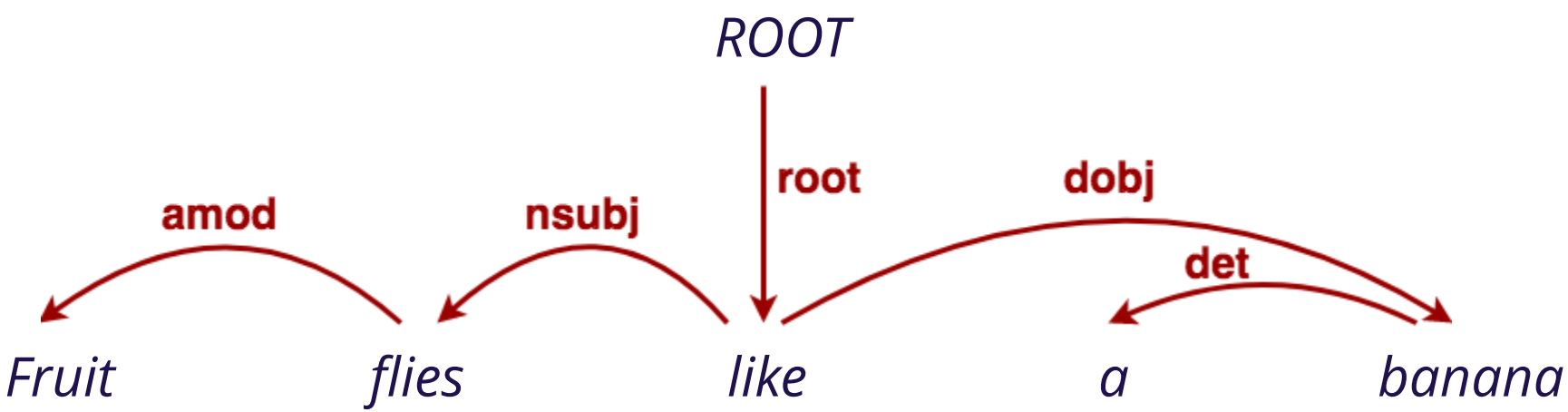


Constituency Parsing



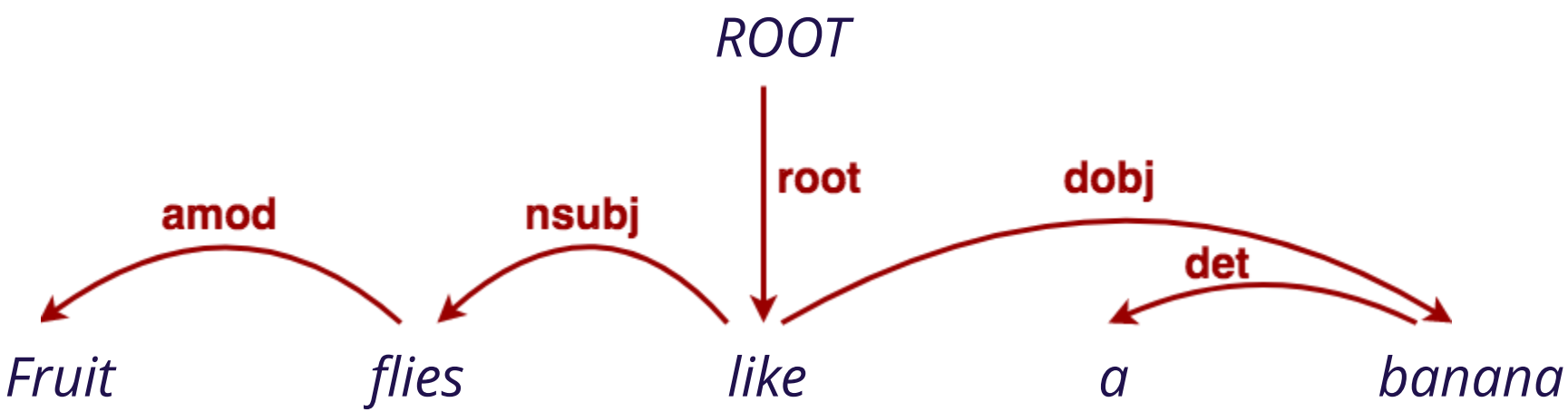
Dependency Parsing

It also analyzes the grammar structure of a sentence, but it explicitly captures the relations between "head" words and the words which modify those head words.



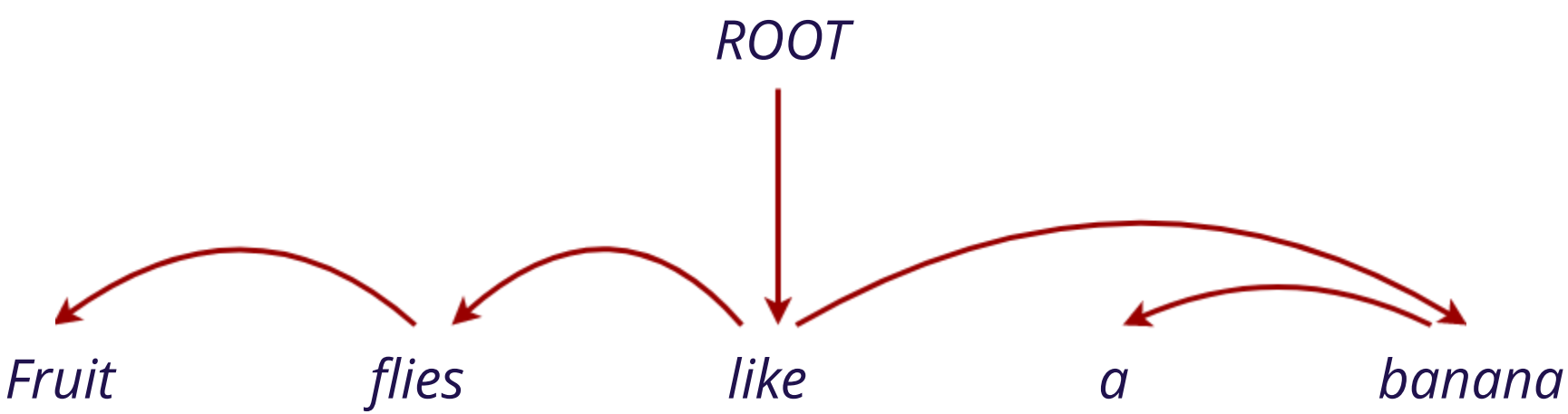
Dependency Parsing

1. It is a tree structure where each word strictly depends on one parent
2. The dependency tree is one single connected tree where each word serves as one node in the tree



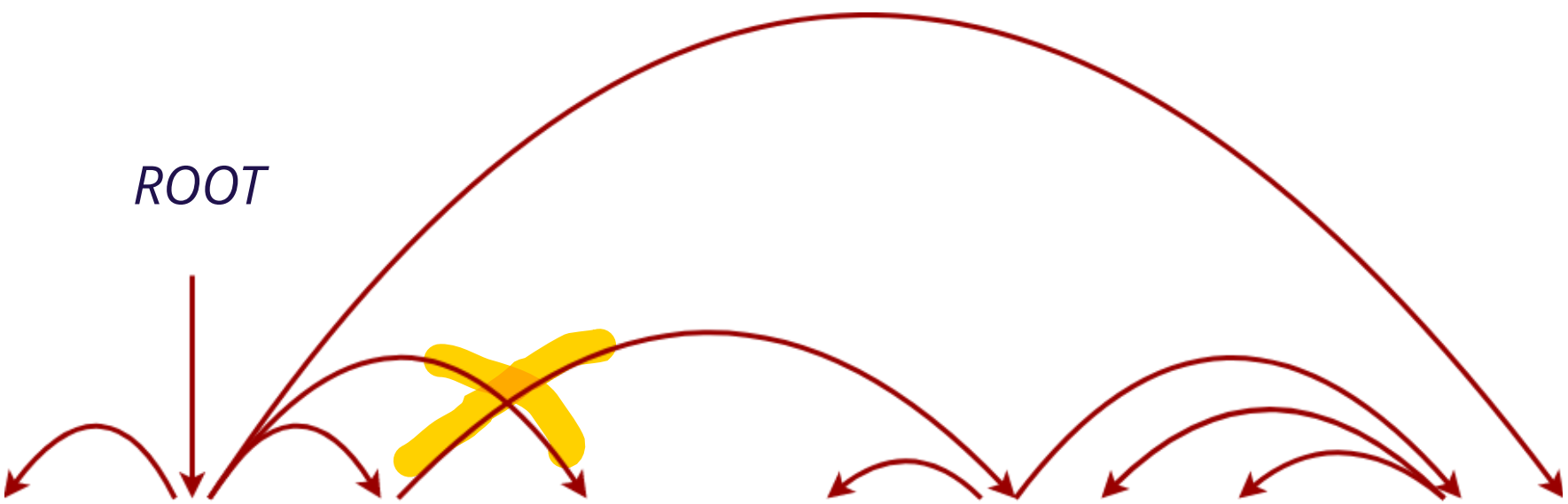
Projective Tree

Edges above each word do not cross

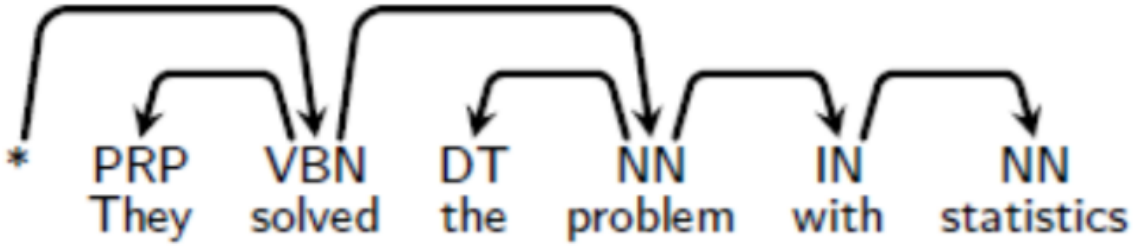


Non-Projective Tree

A tree that is not projective



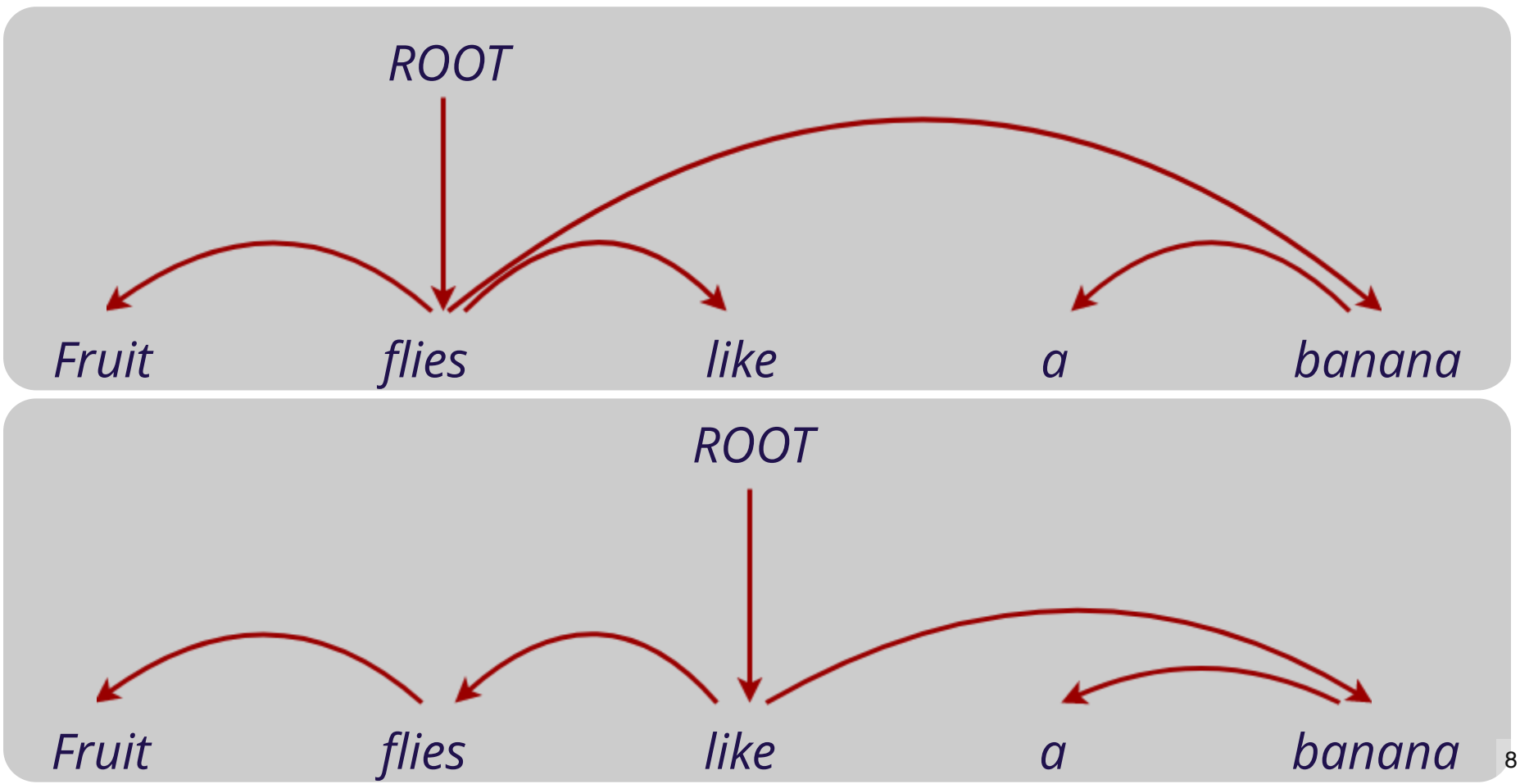
Conditions on Dependency Structures



- Dependency tree
  a) Each non-root token has exactly one incoming arc
  b) All tokens are weakly connected
  c) There are no cycles
  - That is, dependency arcs form a directed tree rooted at *
- Projective dependency trees
  - There are no crossing dependencies
- Non-projective dependency trees, no further constraints
  - Hence a projective tree is also a non-projective set

Dependency Parsing

There are many projective trees for the same input sentence

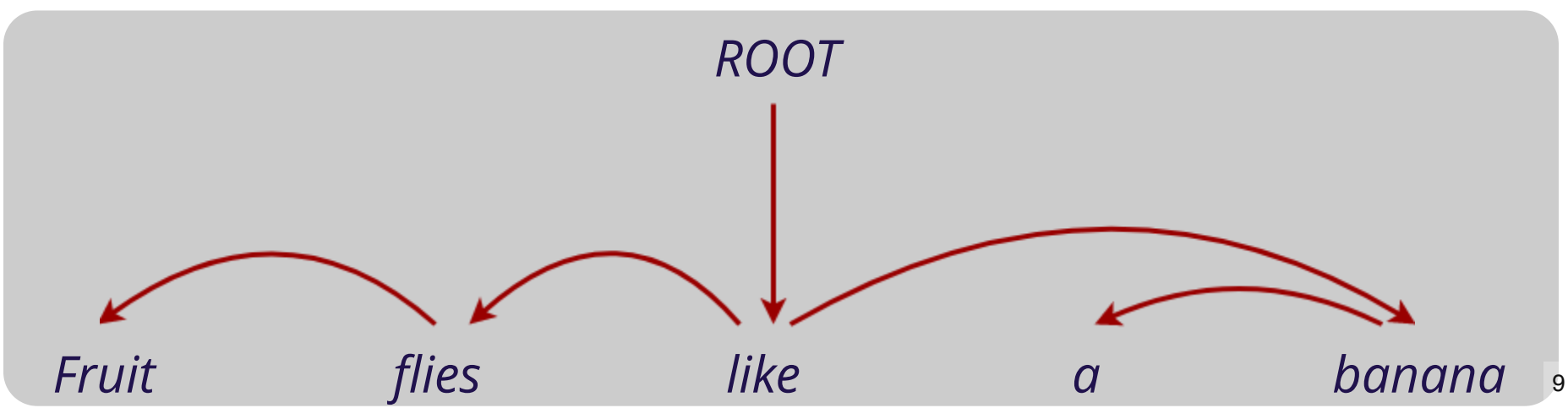


Dependency Parsing

How do we score a tree?

$$y^* = \arg\max_{y \in \mathbf{GEN}(x)} score(y)$$

The set of trees that are compatible with the input sentence.

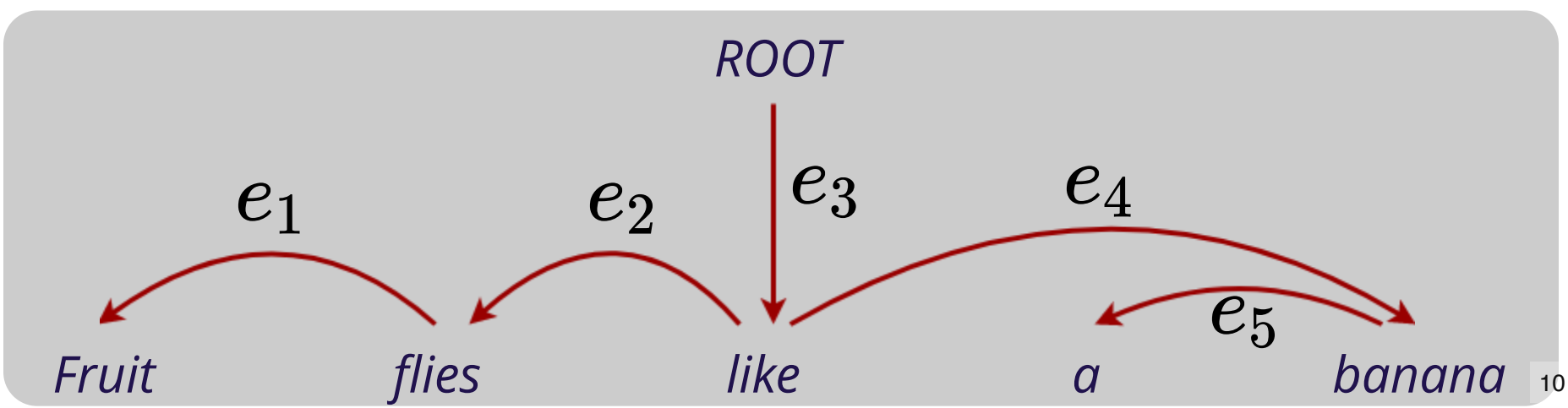


Dependency Parsing

Arc-factored Model

$$\begin{aligned} \boldsymbol{y}^* &= \arg\max_{\boldsymbol{y} \in \mathbf{GEN}(\boldsymbol{x})} score(\boldsymbol{y}) \\ &= \arg\max_{\boldsymbol{y} \in \mathbf{GEN}(\boldsymbol{x})} \sum_{e \in \boldsymbol{y}} score(e) \end{aligned}$$

The score can be either log-probabilities or weights



Question

Can we perform efficient *projective* dependency parsing?

Projective Parsing (Eisner, 1996)

Three New Probabilistic Models for Dependency Parsing: An Exploration*

Jason M. Eisner
CIS Department, University of Pennsylvania
200 S. 33rd St., Philadelphia, PA 19104-6389, USA
jeisner@linc.cis.upenn.edu

Abstract

After presenting a novel $O(n^3)$ parsing algorithm for dependency grammar, we develop three contrasting ways to stochasticize it. We propose (a) a lexical affinity model where words struggle to modify each other, (b) a sense tagging model where words fluctuate randomly in their selectional preferences, and (c) a generative model where the speaker fleshes out each word's syntactic and conceptual structure without regard to the implications for the hearer. We also give preliminary empirical results from evaluating the three models' parsing performance on annotated *Wall Street Journal* training text (derived from the Penn Treebank). In these results, the generative model performs significantly better than the others, and does about equally well at assigning part-of-speech tags.

1 Introduction

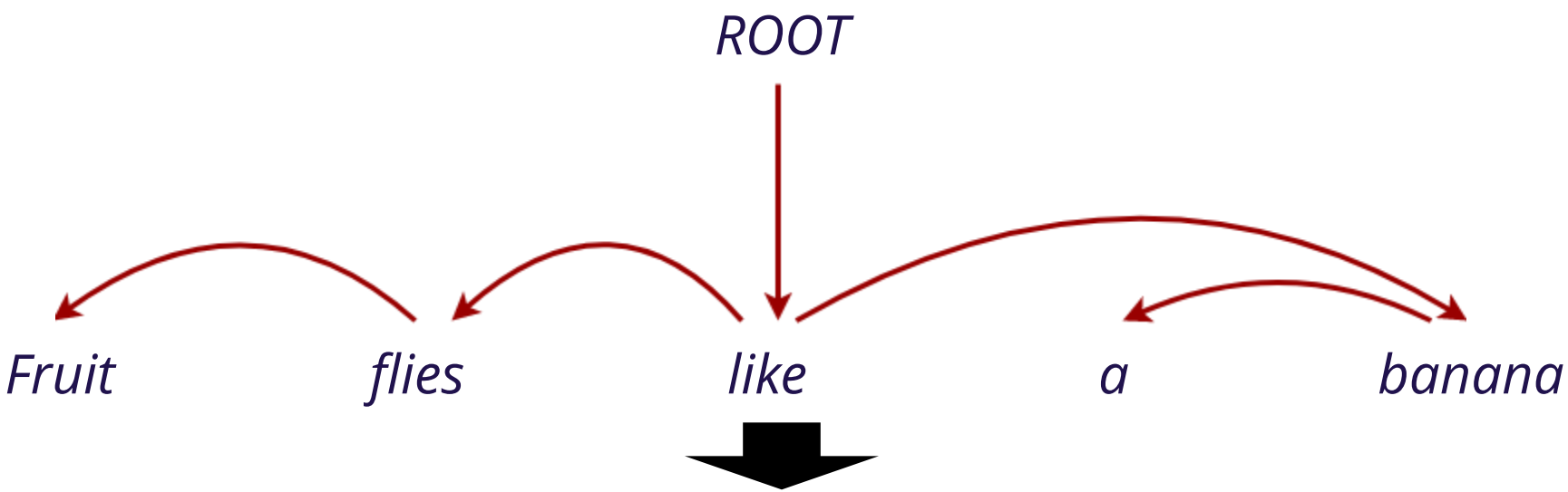
In recent years, the statistical parsing community has begun to reach out for syntactic formalisms that recognize the individuality of words. Link grammars (Sleator and Temperley, 1991) and lexicalized tree-adjoining grammars (Schabes, 1992) have now received stochastic treatments. Other researchers, not wishing to abandon context-free grammar (CFG) but disillusioned with its lexical blind spot, have tried to re-parameterize stochastic CFG in context-sensitive ways (Black et al., 1992) or have augmented the formalism with lexical

Figure 1: (a) A bare-bones dependency parse. Each word points to a single **parent**, the word it modifies; the head of the sentence points to the **EOS** (end-of-sentence) mark. Crossing links and cycles are not allowed. (b) Constituent structure and subcategorization may be highlighted by displaying the same dependencies as a lexical tree.

them. It is useful to look into these basic questions before trying to fine-tune the performance of systems whose behavior is harder to understand.[1]

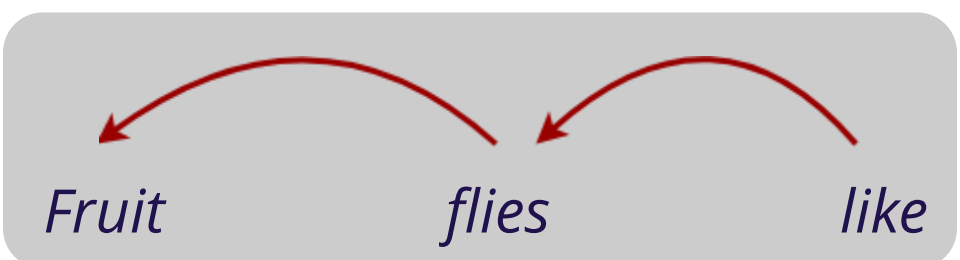
The main contribution of the work is to propose three distinct, lexicalist hypotheses about the probability space underlying sentence structure. We illustrate how each hypothesis is expressed in a dependency framework, and how each can be used to guide our parser toward its favored solution. Finally, we point to experimental results that compare the three hypotheses' parsing performance on sentences from the *Wall Street Journal*. The parser is trained on an annotated corpus; no hand-written grammar is required.

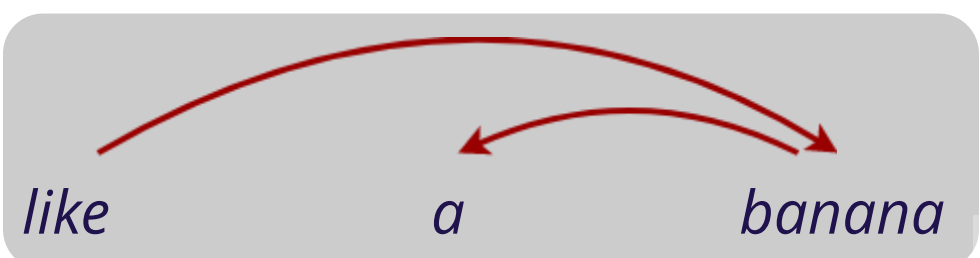
Eisner's Algorithm



Both sides are subtrees with root "like"

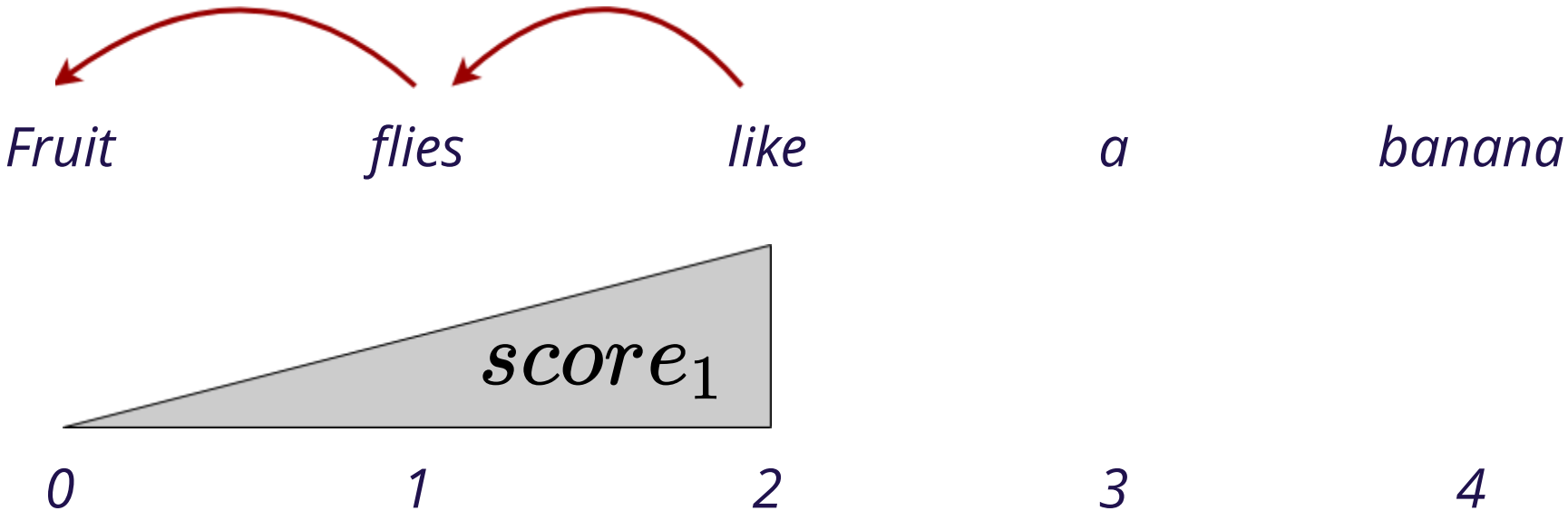
Dynamic programming!





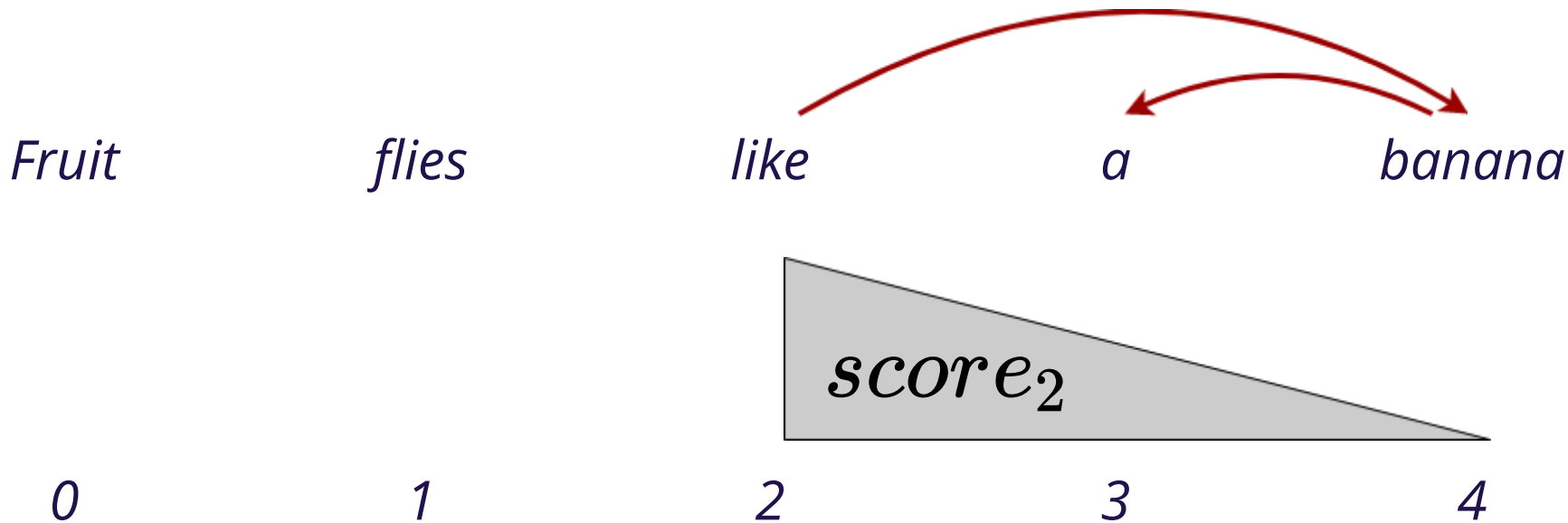
Fruit flies a banana

Eisner's Algorithm



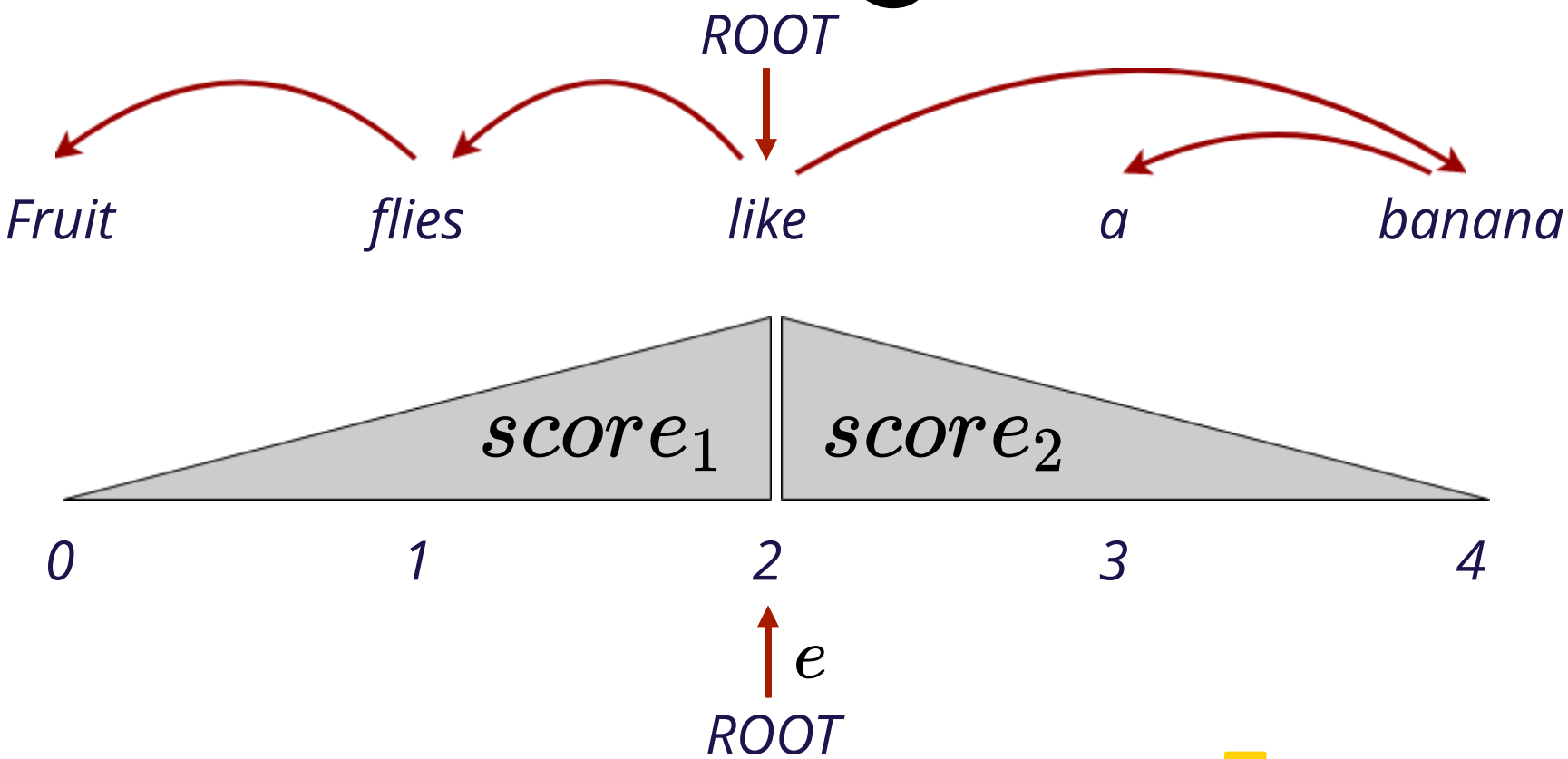
The score of the most probable tree with root at 2 that covers the span [0, 2]

Eisner's Algorithm



The score of the most probable tree with root at 2 that covers the span [2, 4]

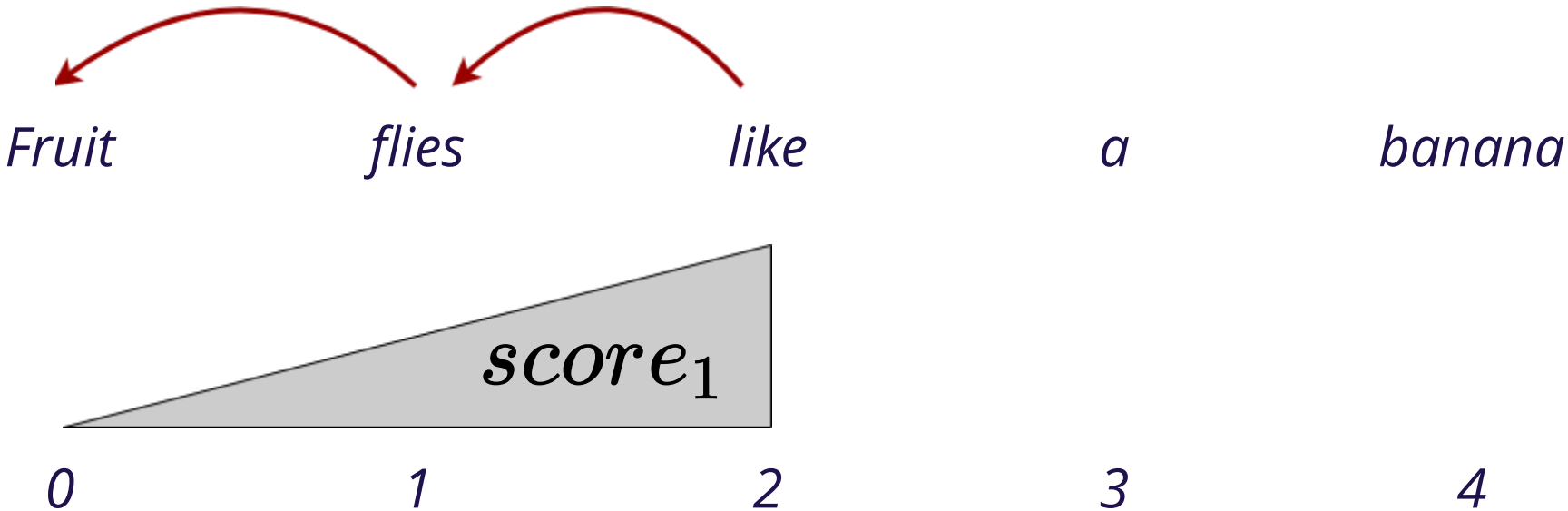
Eisner's Algorithm



The score of the most probable tree with root at 2 and covers the span [0, 4]

$$score_1 + score_2 + score(e)$$

Eisner's Algorithm

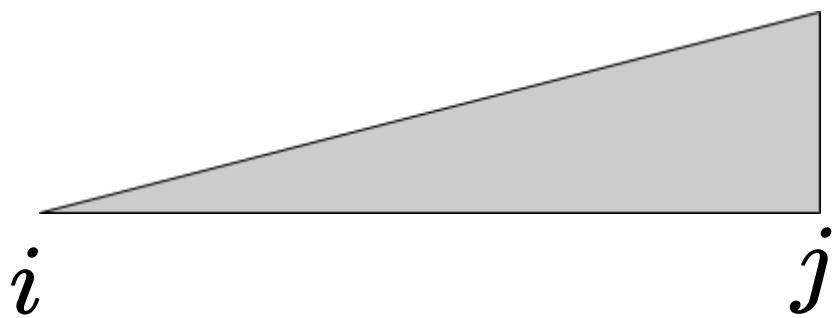


The score of the most probable tree with root at 2 that covers the span [0, 2]

How to find this score (or the tree)?

Eisner's Algorithm

A General Case

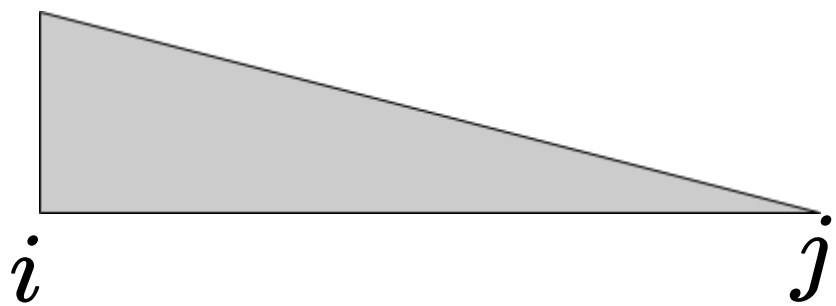


The score of the most probable tree with root at j that covers the span $[i, j]$

$$\overleftarrow{\pi}[i, j]$$

Eisner's Algorithm

Similarly

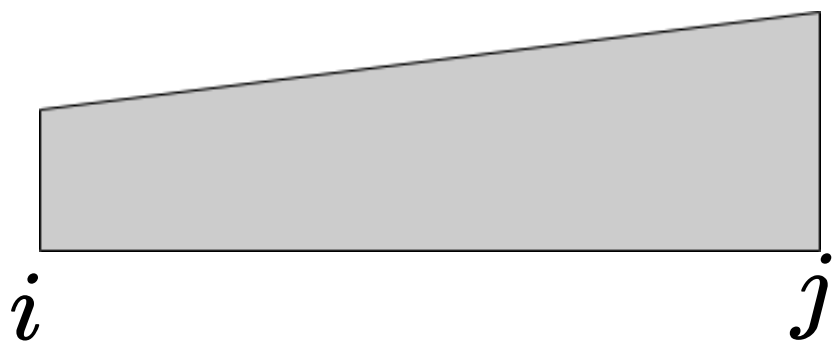


The score of the most probable tree with root at i that covers the span $[i, j]$

$$\overrightarrow{\pi}[i, j]$$

Eisner's Algorithm

An auxiliary term

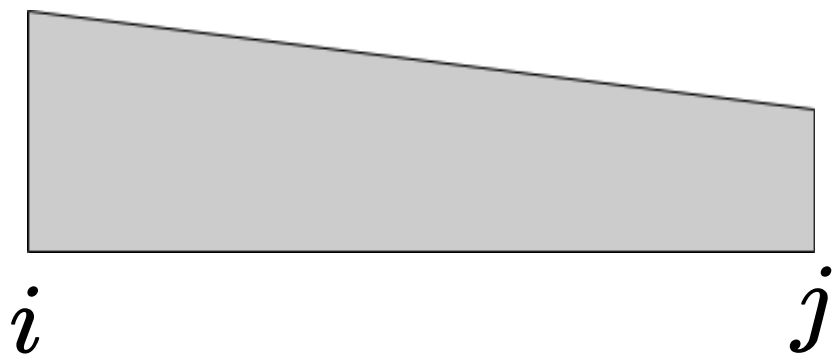


The score of the most probable tree with root at j that covers the span $[i, j]$ and there is a direct arc from j to i

$$\overleftarrow{s}\,[i, j]$$

Eisner's Algorithm

Similarly



The score of the most probable tree with root at i that covers the span $[i, j]$ and there is a direct arc from i to j

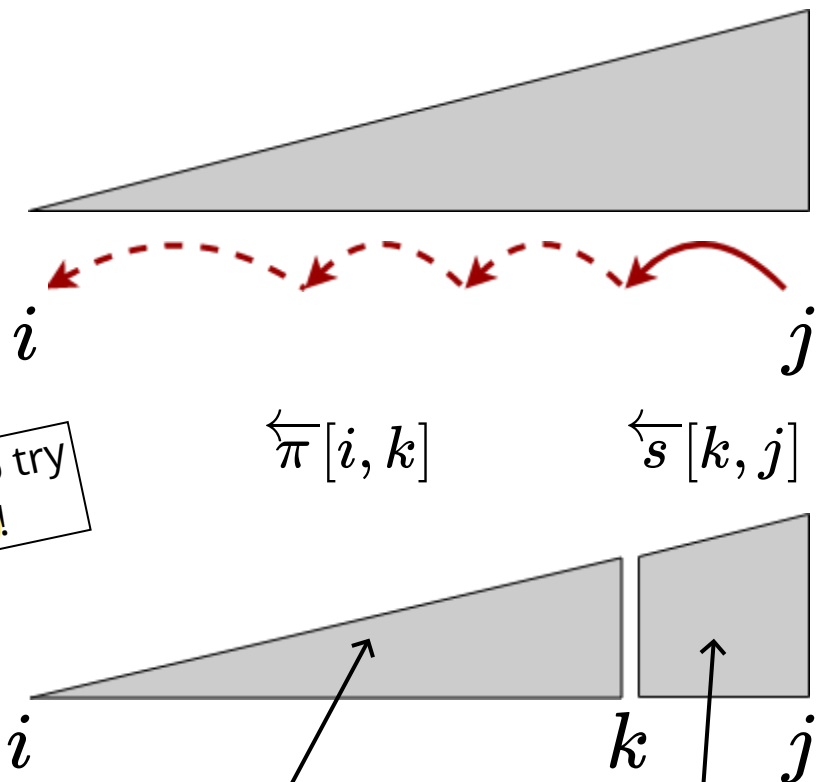
$$\overrightarrow{s}[i, j]$$

Eisner's Algorithm

1

The score of the most probable tree with root at j that covers the span $[i, j]$.

$$\overleftarrow{\pi}[i,j] = \max_{i \le k \le j} (\overleftarrow{\pi}[i,k] + \overleftarrow{s}[k,j])$$



You need a 'for' loop to try all the values for k!

The score of the most probable tree with root at k that covers the span $[i, k]$.

The score of the most probable tree with root at j that covers the span $[k, j]$, and there is a direct edge from j to k.

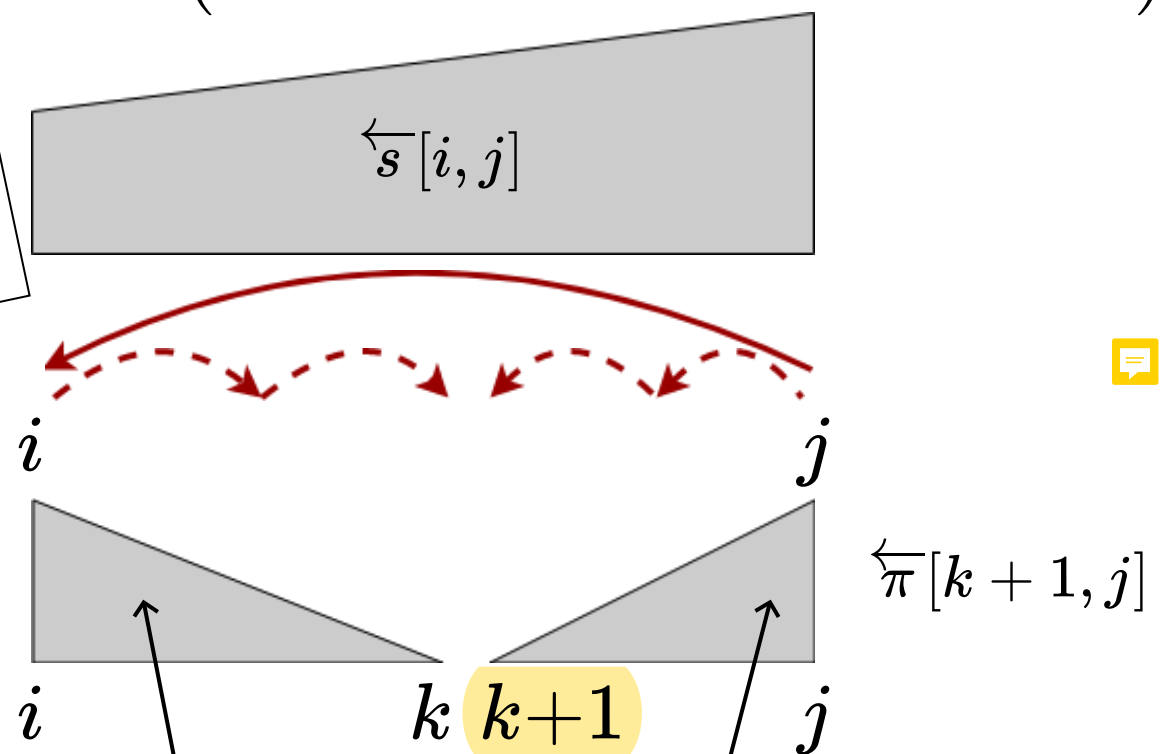
Eisner's Algorithm

2

The score of the most probable tree with root at j that covers the span $[i, j]$ and there is a direct edge from j to i.

$$\overleftarrow{s}[i,j] = \max_{i \le k < j} \left(\overrightarrow{\pi}[i,k] + \overleftarrow{\pi}[k+1,j] + score(i \leftarrow j) \right)$$

Again, you need a 'for' loop to try all the values for k !



The score of the most probable tree with root at i that covers the span $[i, k]$.

The score of the most probable tree with root at j that covers the span $[k+1, j]$.

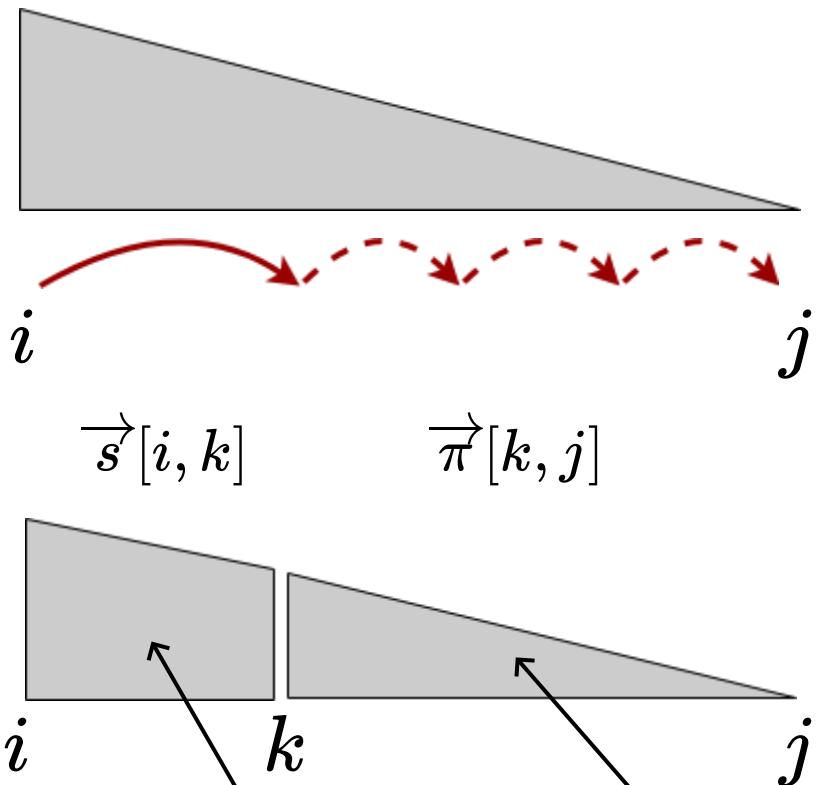
These are the two sub-problems. Use bottom-up dynamic programming!

Eisner's Algorithm

3

The score of the most probable tree with root at i that covers the span $[i, j]$.

$$\overrightarrow{\pi}[i,j] = \max_{i \le k \le j} (\overrightarrow{s}[i,k] + \overrightarrow{\pi}[k,j])$$



The score of the most probable tree with root at i that covers the span $[i, k]$, and there is a direct edge from i to k.

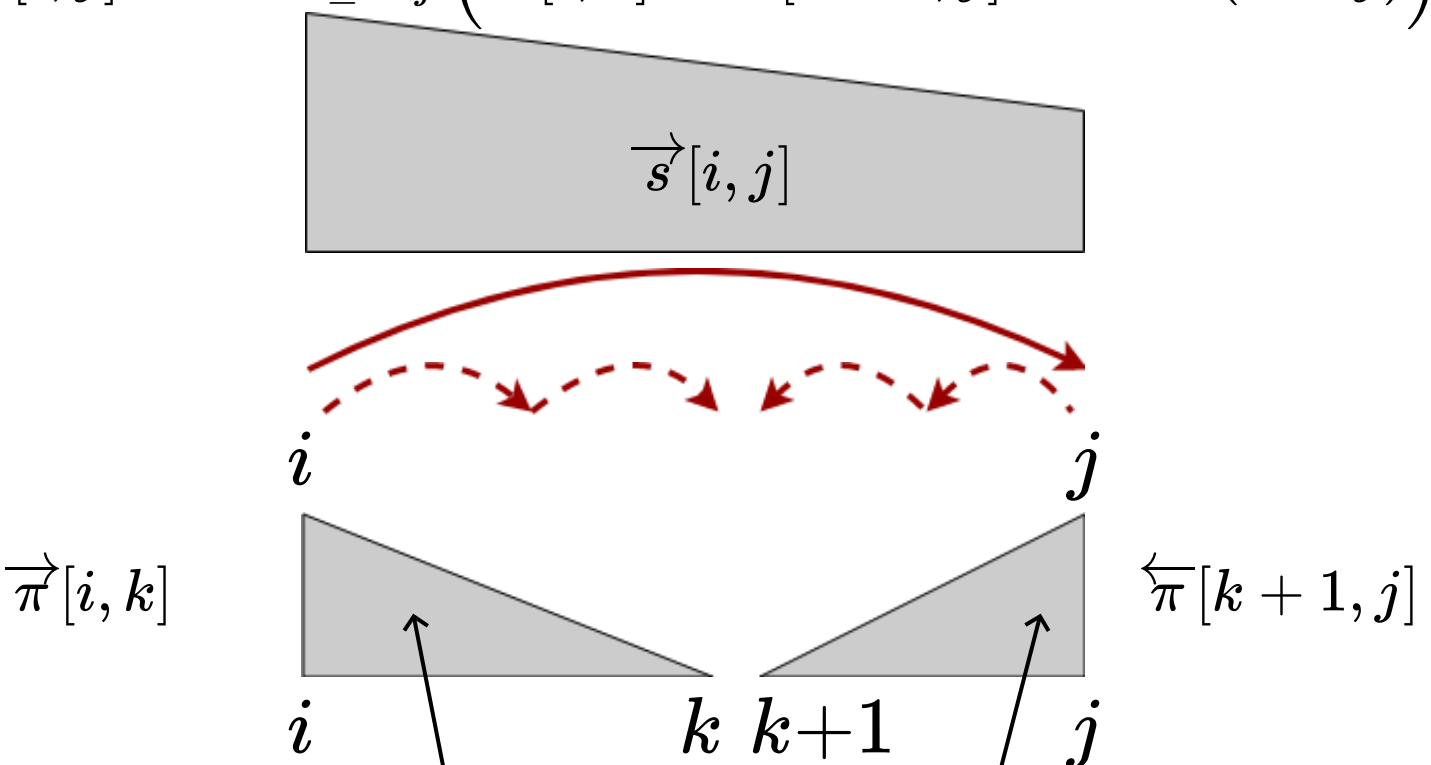
The score of the most probable tree with root at k that covers the span $[k, j]$.

Eisner's Algorithm

4

The score of the most probable tree with root at i that covers the span $[i, j]$ and there is a direct edge from j to i.

$$\overrightarrow{s}[i,j] = \max_{i \le k < j} \left(\overrightarrow{\pi}[i,k] + \overleftarrow{\pi}[k+1,j] + score(i \rightarrow j) \right)$$



The score of the most probable tree with root at i that covers the span $[i, k]$.

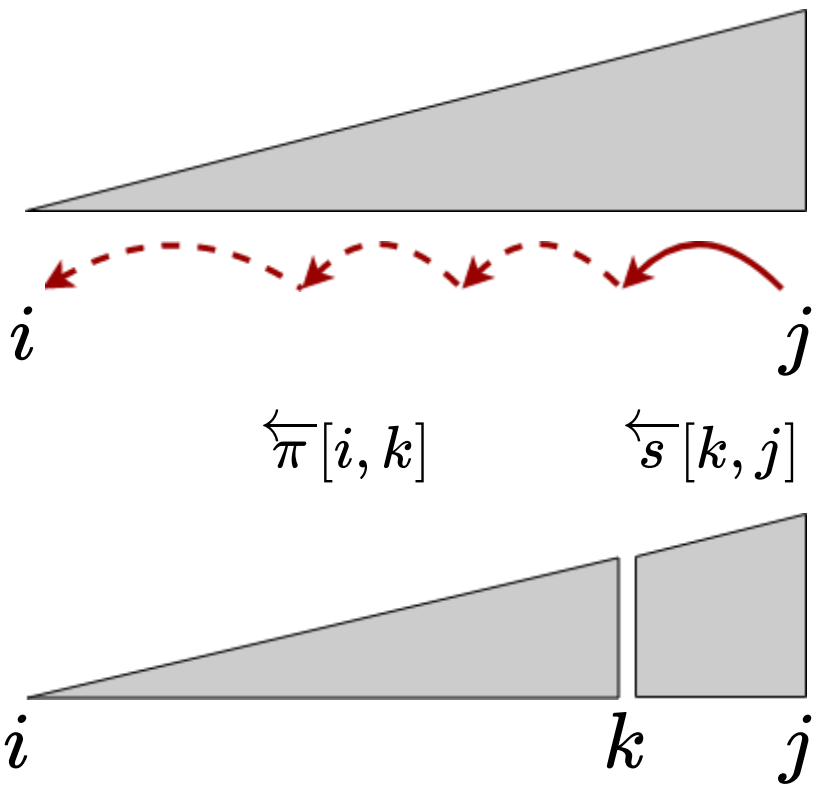
The score of the most probable tree with root at j that covers the span $[k+1, j]$.

Question

What if we have tags along the edges?

Eisner's Algorithm

$$\overleftarrow{\pi}[i,j] = \max_{i \le k \le j}(\overleftarrow{\pi}[i,k] + \overleftarrow{s}[k,j])$$



How to modify this algorithm to support arcs of different labels?

Question

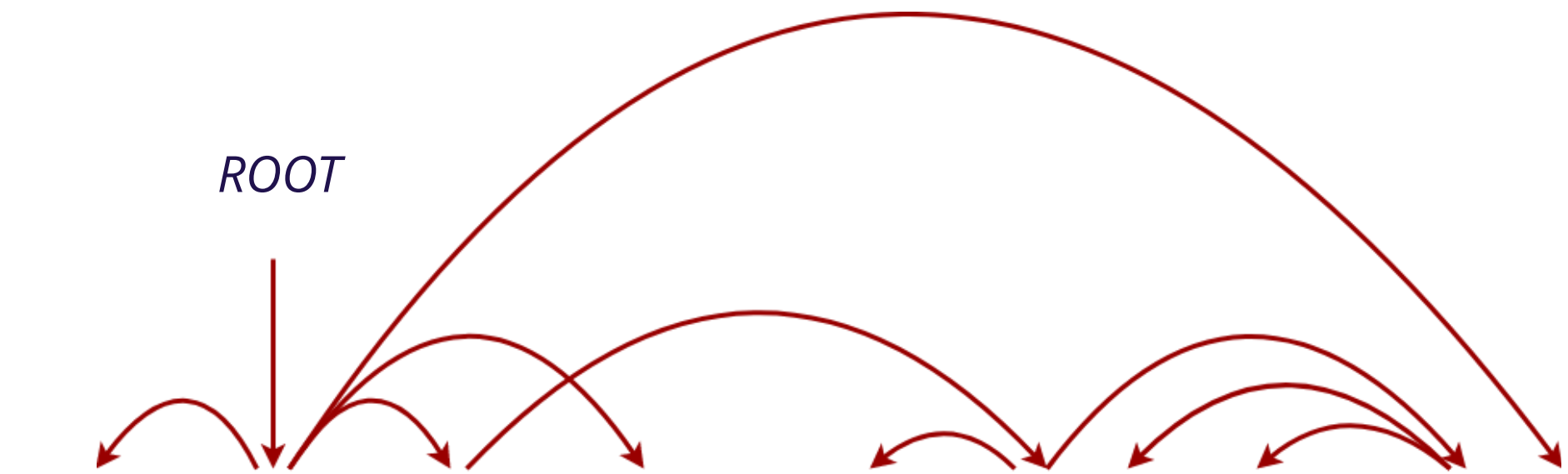
What is the time complexity of Eisner's Algorithm?

$O(n^3)$
Possible (i, j) combinations, n^2
For each (i, j) interval, we need to find out the split point (k)
For each sentence, find i, then j, then k
nC3 --> cube complexity.

Or look at possible spans = n^2. To get the score, you need to visit all possible scores.

Non-Projective Trees

Can we perform efficient
non-projective dependency parsing?



Non-Projective Parsing (McDonald et al, 2005)

Non-projective Dependency Parsing using Spanning Tree Algorithms

Ryan McDonald **Fernando Pereira**
Department of Computer and Information Science
University of Pennsylvania
{ryantm,pereira}@cis.upenn.edu

Kiril Ribarov **Jan Hajič**
Institute of Formal and Applied Linguistics
Charles University
{ribarov,hajic}@ufal.ms.mff.cuni.cz

Abstract

We formalize weighted dependency parsing as searching for maximum spanning trees (MSTs) in directed graphs. Using this representation, the parsing algorithm of Eisner (1996) is sufficient for searching over all projective trees in $O(n^3)$ time. More surprisingly, the representation is extended naturally to non-projective parsing using Chu-Liu-Edmonds (Chu and Liu, 1965; Edmonds, 1967) MST algorithm, yielding an $O(n^2)$ parsing algorithm. We evaluate these methods on the Prague Dependency Treebank using online large-margin learning techniques (Crammer et al., 2003; McDonald et al., 2005) and show that MST parsing increases efficiency and accuracy for languages with non-projective dependencies.

1 Introduction

Dependency parsing has seen a surge of interest lately for applications such as relation extraction (Culotta and Sorensen, 2004), machine translation (Ding and Palmer, 2005), synonym generation (Shinyama et al., 2002), and lexical resource

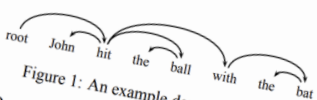


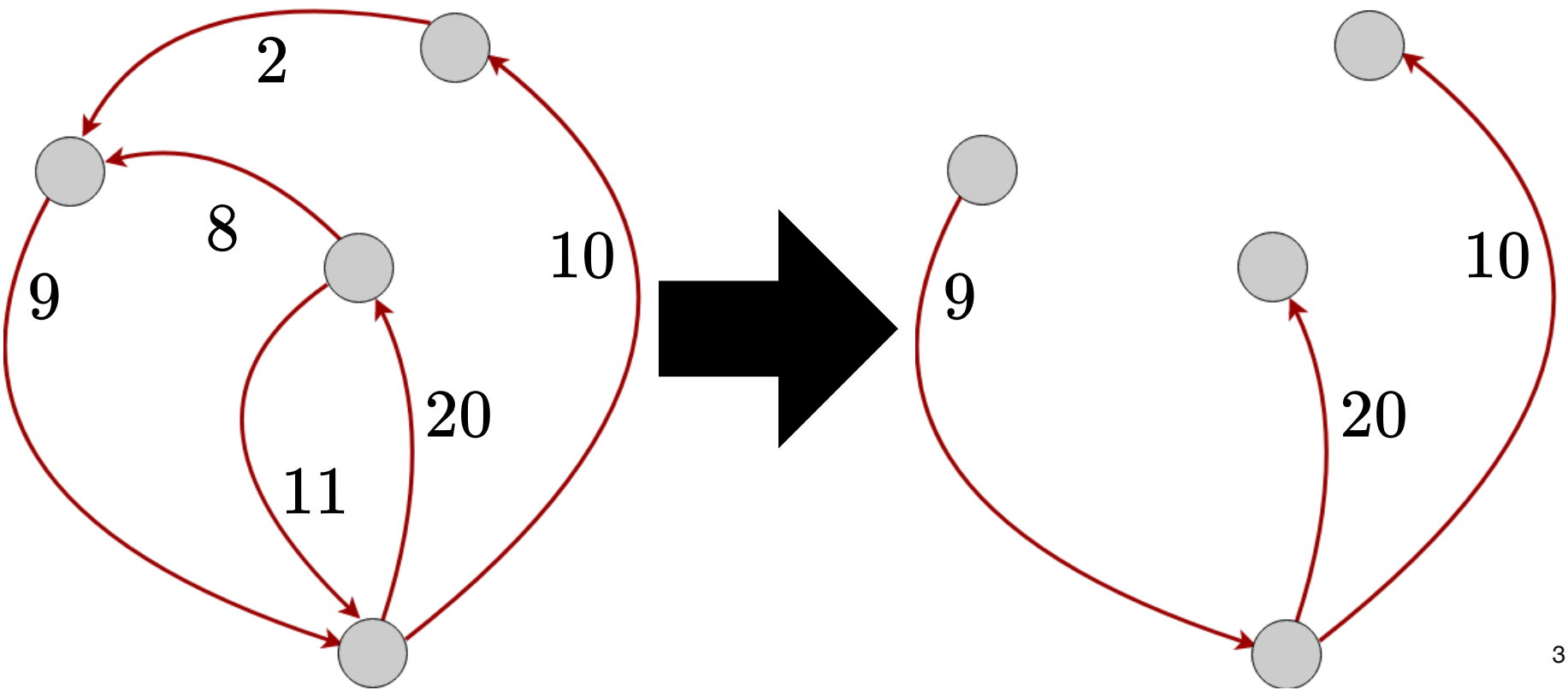
Figure 1: An example dependency tree.

Dependency representations, which link words to their arguments, have a long history (Hudson, 1984). Figure 1 shows a dependency tree for the sentence *John hit the ball with the bat*. We restrict ourselves to *dependency tree* analyses, in which each word depends on exactly one parent, either another word or a dummy root symbol as shown in the figure. The tree in Figure 1 is *projective*, meaning that if we put the words in their linear order, preceded by the root, the edges can be drawn above the words without crossings, or, equivalently, a word and its descendants form a contiguous substring of the sentence.

In English, projective trees are sufficient to analyze most sentence types. In fact, the largest source of English dependency trees is automatically generated from the Penn Treebank (Marcus et al., 1993) and is by convention exclusively projective. However, there are certain examples in which a non-projective tree is preferable. Consider the sentence *John saw a dog yesterday which was a Yorkshire Terrier*. Here the relative clause *which was a Yorkshire Terrier* and the object it modifies (the *dog*) are separated by an adverb. There is no way to draw the

Maximum Spanning Tree

Finding the optimal non-projective dependency tree is essentially the task of finding the maximum spanning tree over the nodes, where each node is a word.



Optional

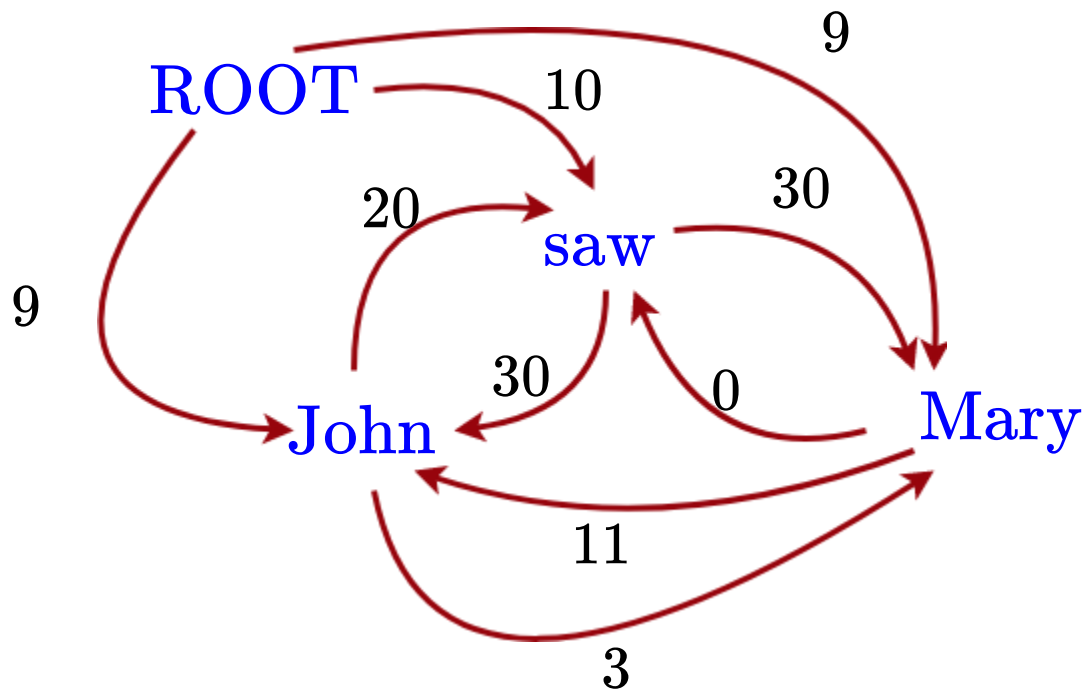
MST Parsing
Chu-Liu-Edmonds Algorithm

Chu-Liu-Edmonds(G, s)
Graph $G = (V, E)$
Edge weight function $s : E \rightarrow \mathbb{R}$
1. Let $M = \{(x^*, x) : x \in V, x^* = \arg\max_{x'} s(x', x)\}$
2. Let $G_M = (V, M)$
3. If G_M has no cycles, then it is an MST: return G_M
4. Otherwise, find a cycle C in G_M
5. Let $G_C = \text{contract}(G, C, s)$
6. Let $\boldsymbol{y} = \text{Chu-Liu-Edmonds}(G_C, s)$
7. Find a vertex $x \in C$ s. t. $(x', x) \in \boldsymbol{y}, (x'', x) \in C$
8. return $\boldsymbol{y} \cup C - \{(x'', x)\}$

contract$(G = (V, E), C, s)$
1. Let G_C be the subgraph of G excluding nodes in C
2. Add a node c to G_C representing cycle C
3. For $x \in V - C : \exists_{x' \in C}(x', x) \in E$
   Add edge (c, x) to G_C with
   $s(c, x) = \max_{x' \in C} s(x', x)$
4. For $x \in V - C : \exists_{x' \in C}(x, x') \in E$
   Add edge (x, c) to G_C with
   $s(x, c) = \max_{x' \in C} [s(x, x') - s(a(x'), x') + s(C)]$
   where $a(v)$ is the predecessor of v in C
   and $s(C) = \sum_{v \in C} s(a(v), v)$
5. return G_C

Optional

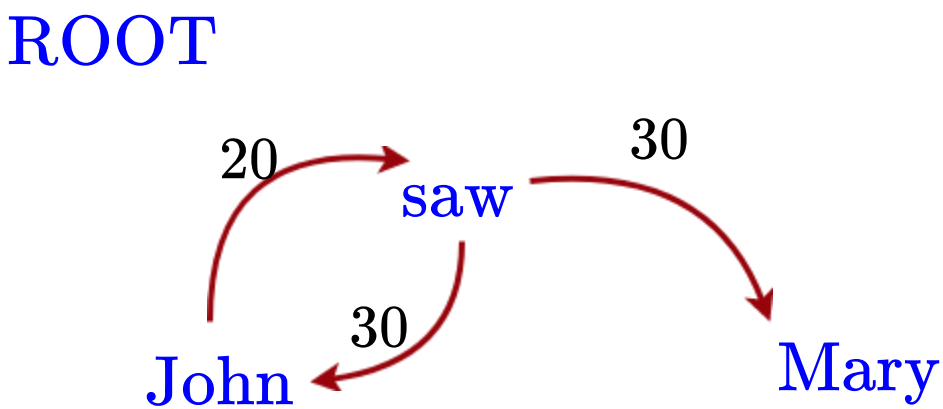
MST Parsing



ROOT John saw Mary

Optional

MST Parsing

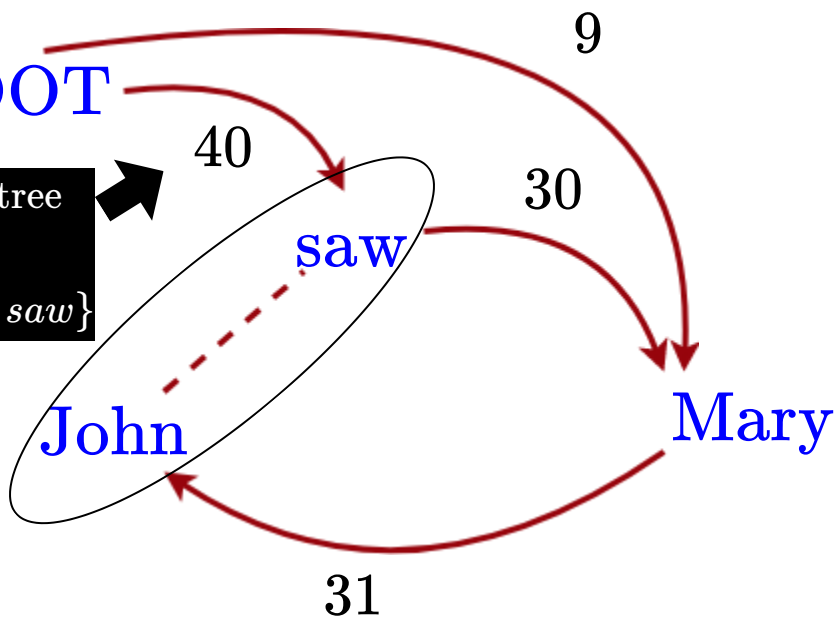


First, find for each word the highest scoring incoming edge.
If it is a tree, then we are done.
However, we have a cycle.

Optional

MST Parsing

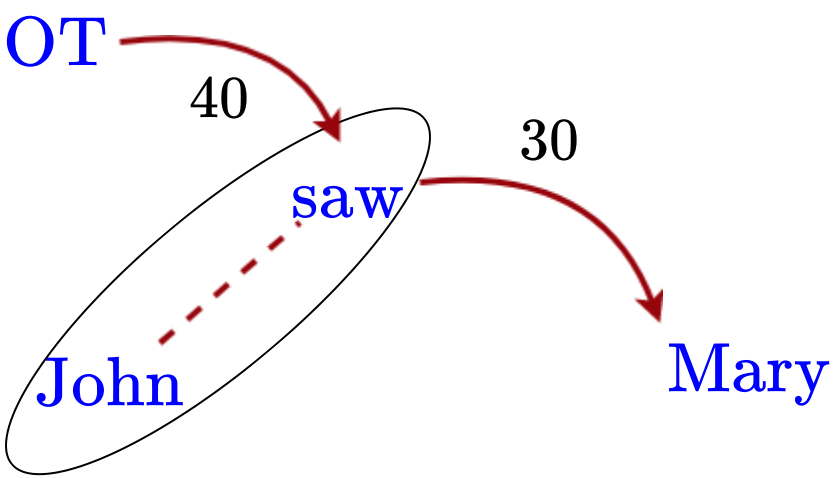
The score of the maximum spanning tree originating from *ROOT* and including only the nodes in $\{John, saw\}$



So we contract it into a "single node", and recalculate edge weights (see the algorithm).

Optional

MST Parsing



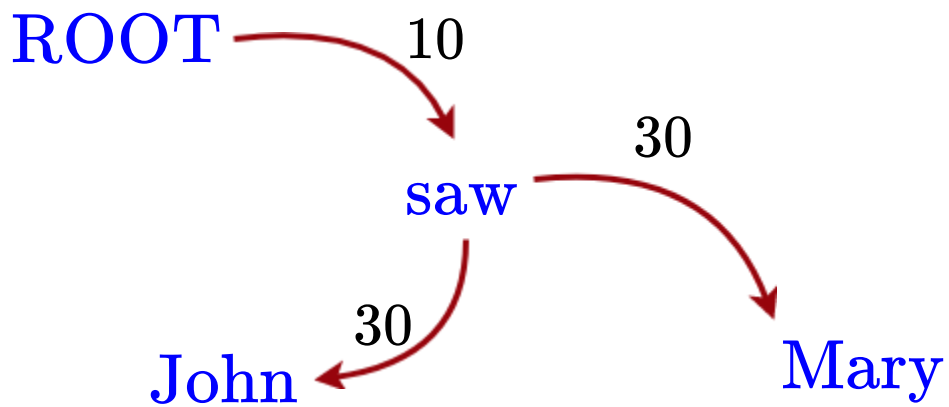
There are some technical details such as keeping track of the endpoints. Refer to the algorithm.

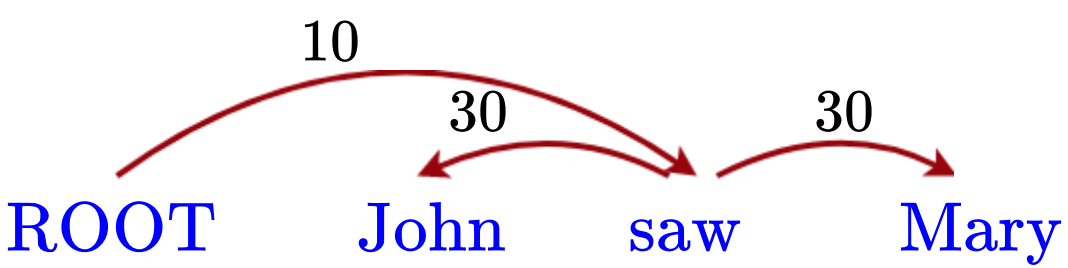
Following the algorithm, we arrive at a tree. Recover the original MST.

Optional

MST Parsing

Overall time complexity $O(n^2)$





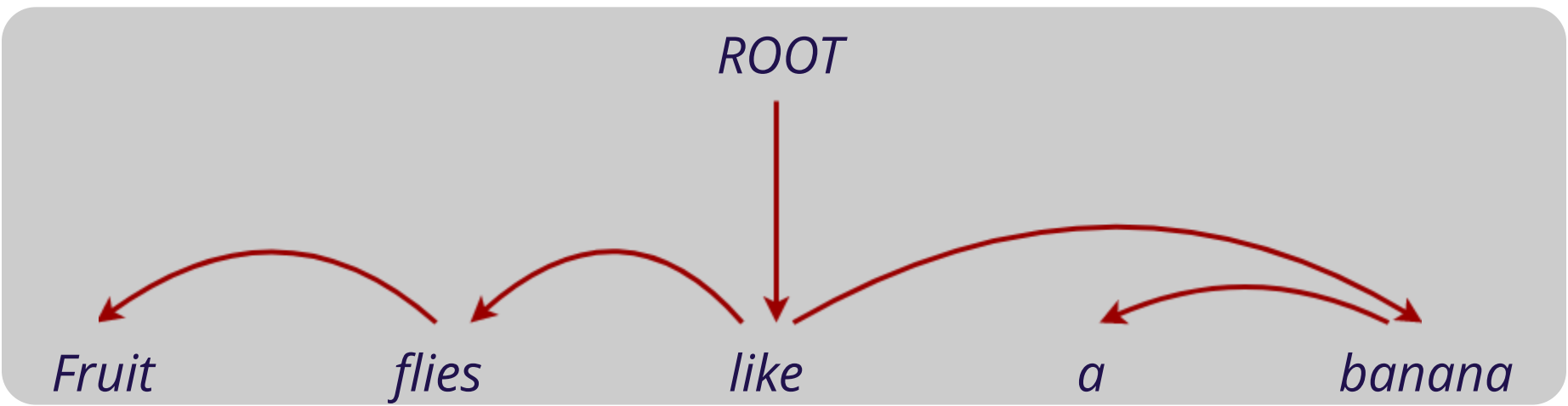
The detailed algorithm is not required in this course, but you need to know how to make use of the MST algorithm for solving the non-projective parsing problem!

Question

We know how to do decoding, but how to do learning?

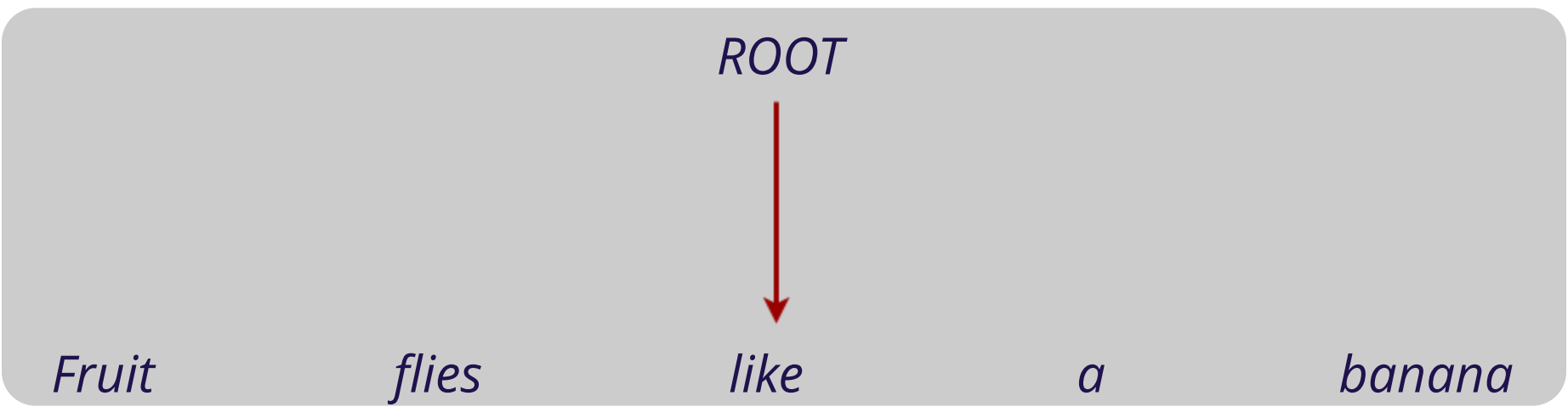
Learning the Weights
Generative Approach

We can assume the following tree is produced from an underlying generative process.



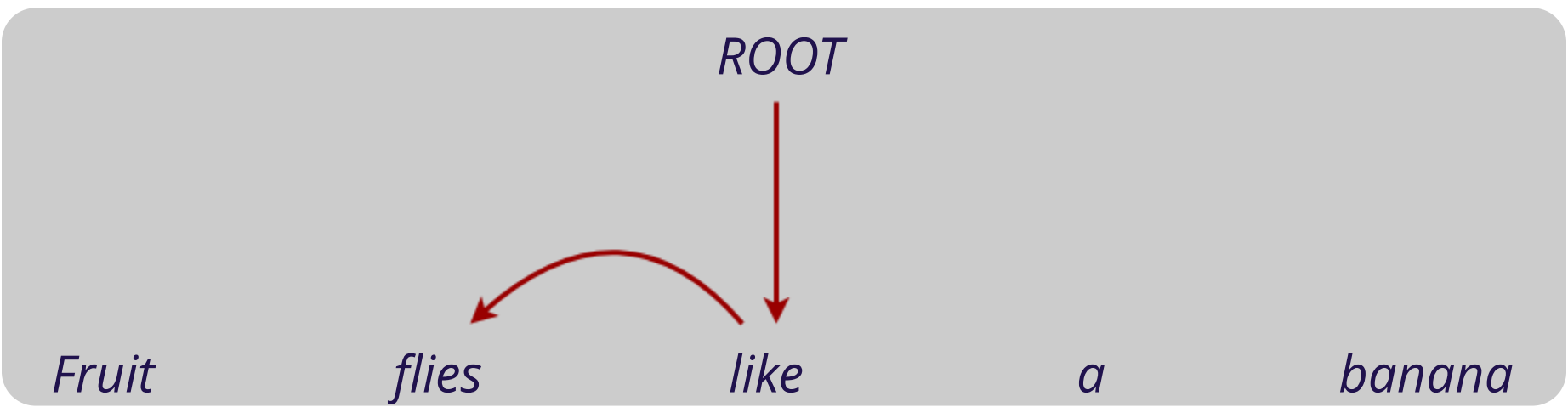
Learning the Weights
Generative Approach

We can assume the following tree is produced from an underlying generative process.



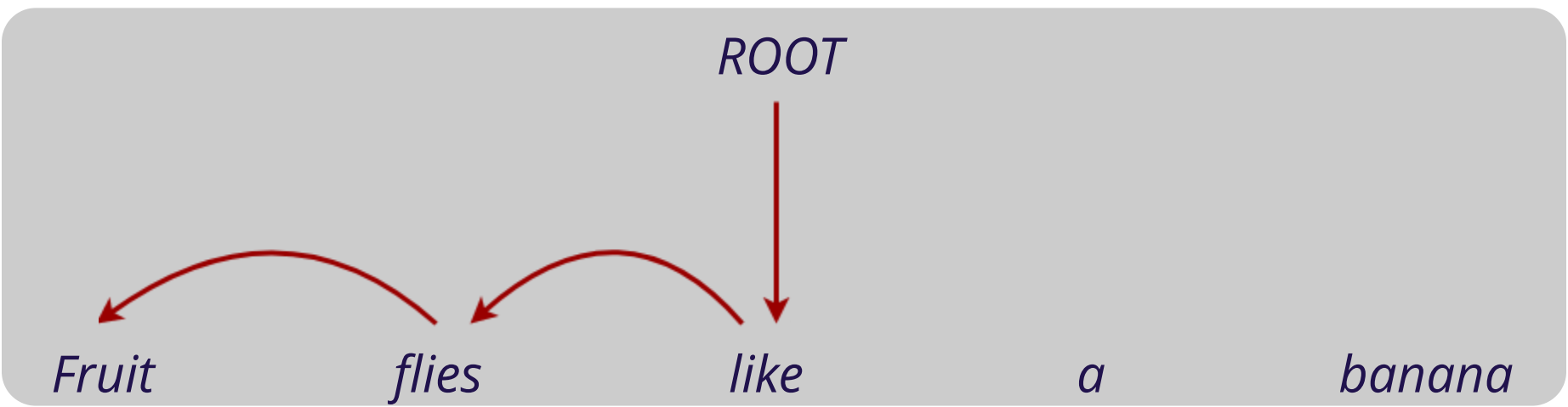
Learning the Weights
Generative Approach

We can assume the following tree is produced from an underlying generative process.



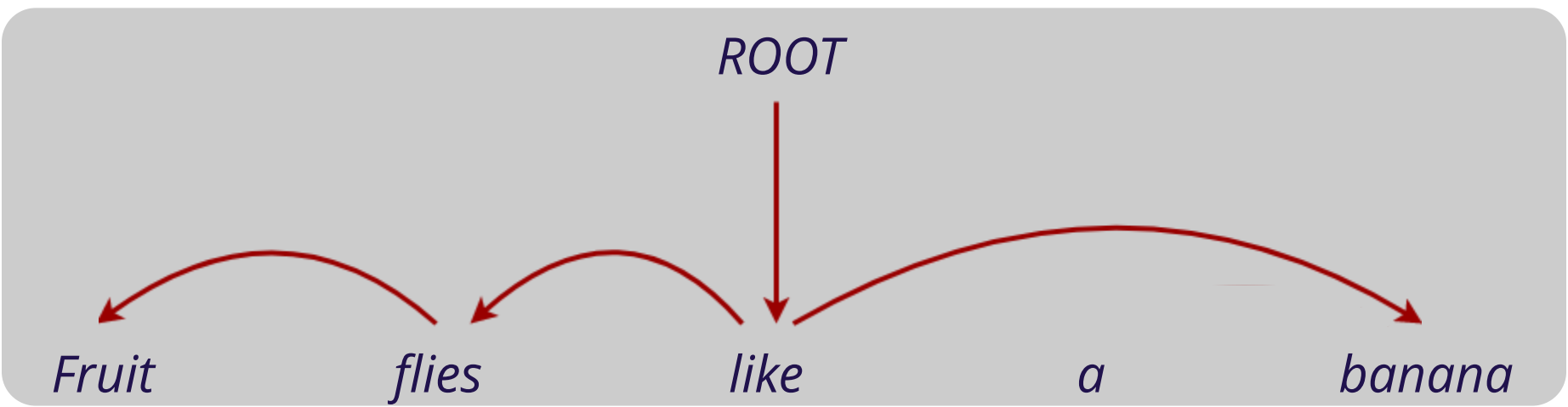
Learning the Weights
Generative Approach

We can assume the following tree is produced from an underlying generative process.



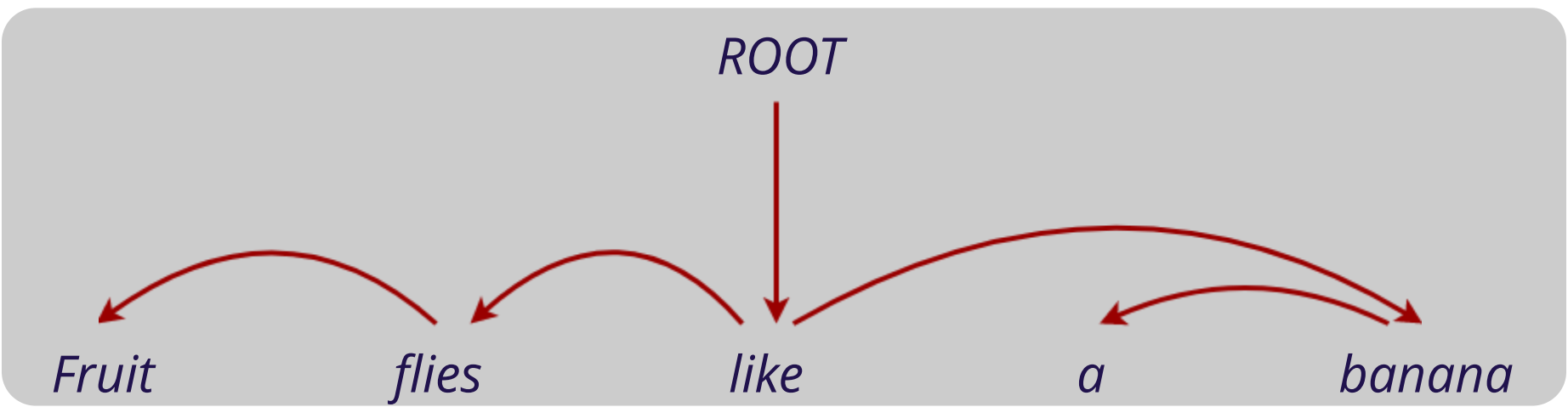
Learning the Weights
Generative Approach

We can assume the following tree is produced from an underlying generative process.



Learning the Weights
Generative Approach

We can assume the following tree is produced from an underlying generative process.



Now, with such a generative process, how to estimate the weights/parameters?

Learning the Weights

Discriminative Approach

We have already learned many standard discriminative approaches for learning structured prediction models. Such standard approaches can be applied here. For example structured perceptron

Depending on the decoding algorithm used, we will be searching from a different space of $\boldsymbol{y}$.

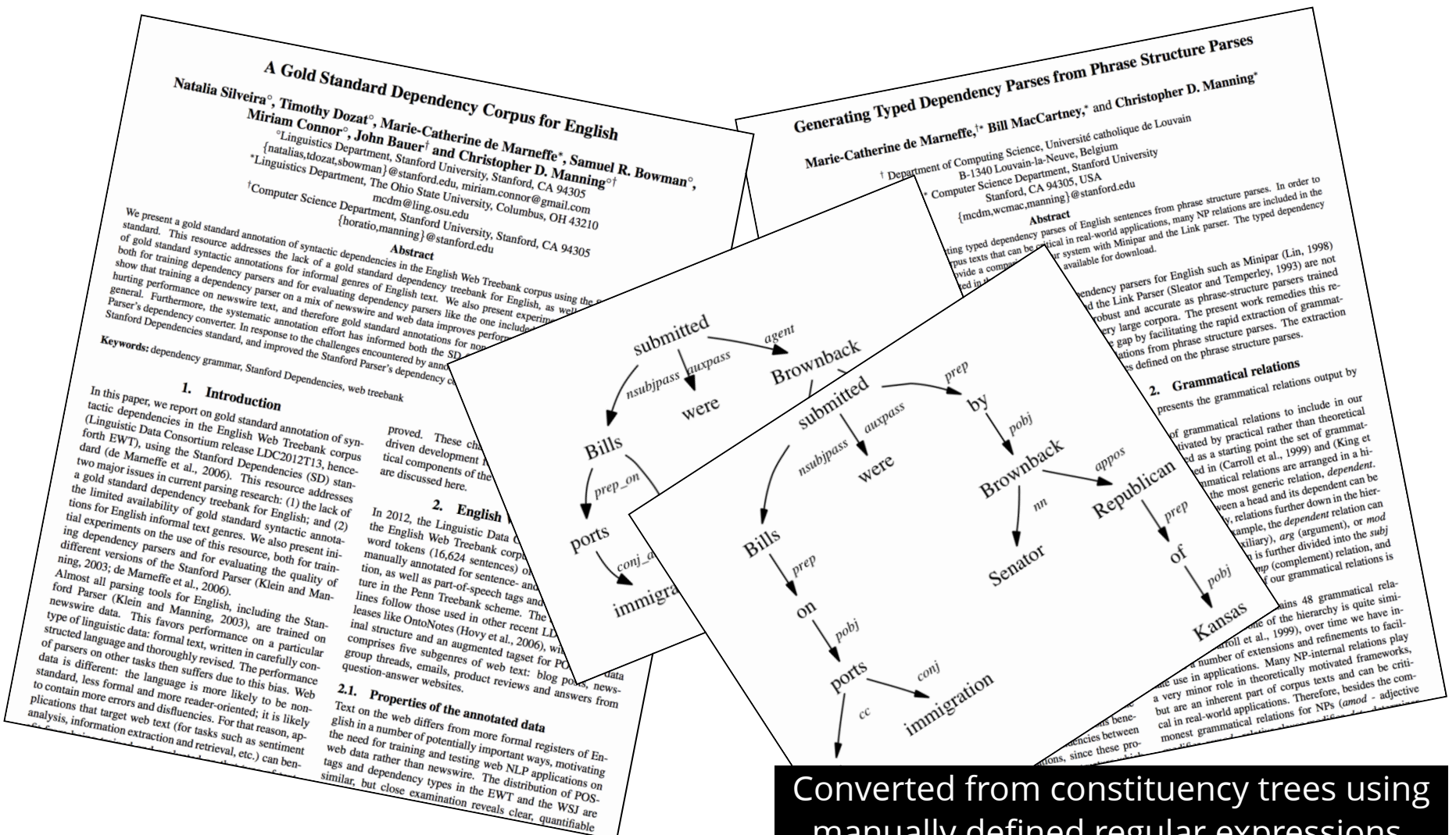
$$\hat{\boldsymbol{y}}_i \leftarrow \arg\max_{\boldsymbol{y} \in \mathbf{GEN}(\boldsymbol{x}_i)} \mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\theta}$$

$$\text{if } \hat{\boldsymbol{y}}_i \neq \boldsymbol{y}_i \text{ then}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \mathbf{f}(\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i)$$

Benchmark Data

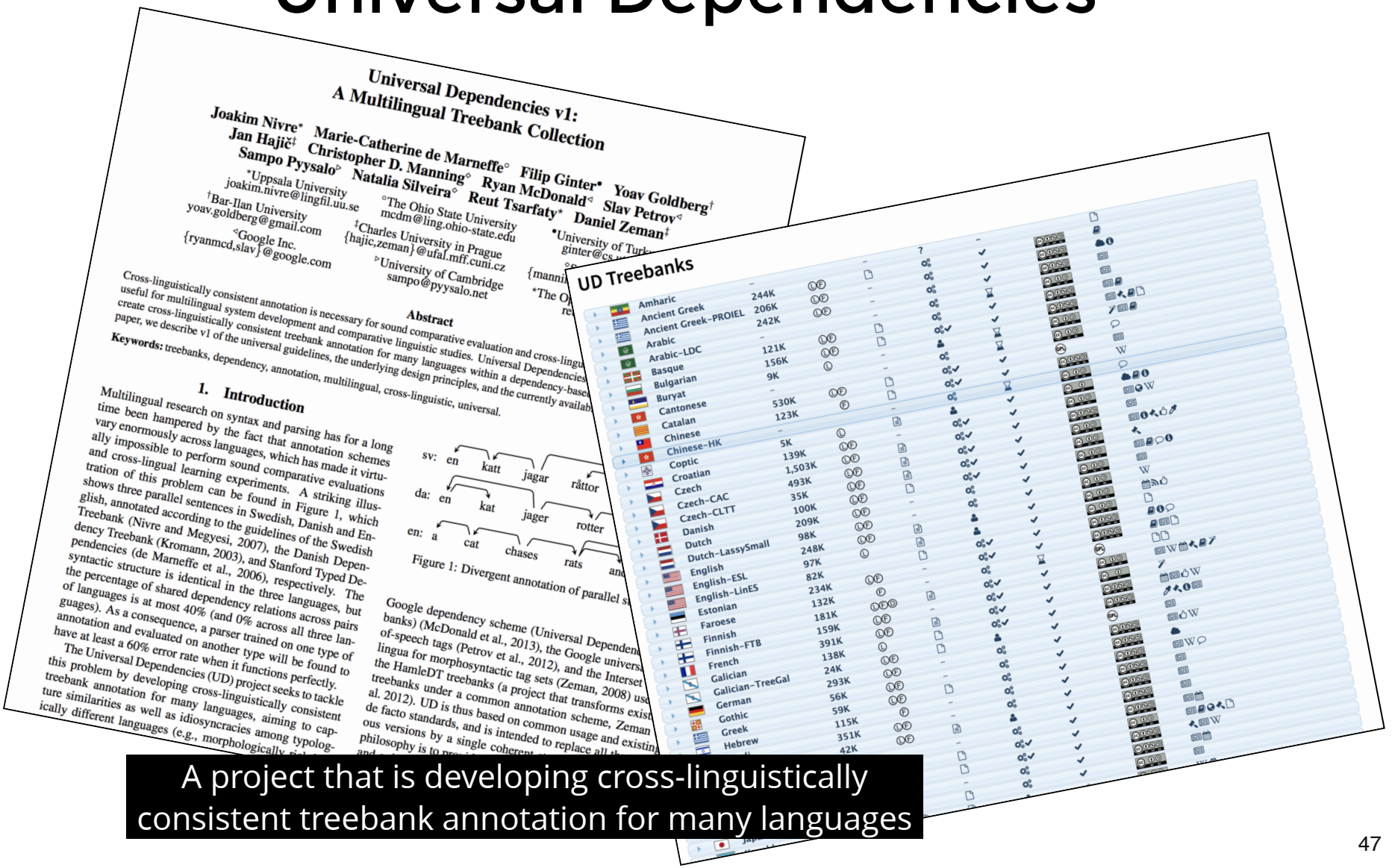
Stanford Dependencies



Converted from constituency trees using manually defined regular expressions.

Benchmark Data

Universal Dependencies



Universal Dependencies v1: A Multilingual Treebank Collection

Joakim Nivre* Marie-Catherine de Marneffe° Filip Ginter• Yoav Goldberg†
Jan Hajič‡ Christopher D. Manning◇ Ryan McDonald◁ Slav Petrov◁
Sampo Pyysalo▷ Natalia Silveira◇ Reut Tsarfaty* Daniel Zeman‡

*Uppsala University
joakim.nivre@lingfil.uu.se

†Bar-Ilan University
yoav.goldberg@gmail.com

◁Google Inc.
{ryanmcd,slav}@google.com

°The Ohio State University
mcdm@ling.ohio-state.edu

‡Charles University in Prague
{hajic,zeman}@ufal.mff.cuni.cz

▷University of Cambridge
sampo@pyysalo.net

Abstract

Cross-linguistically consistent annotation is necessary for sound comparative evaluation and cross-lingu
useful for multilingual system development and comparative linguistic studies. Universal Dependencies
create cross-linguistically consistent treebank annotation for many languages within a dependency-base
paper, we describe v1 of the universal guidelines, the underlying design principles, and the currently availab

Keywords: treebanks, dependency, annotation, multilingual, cross-linguistic, universal.

1. Introduction

Multilingual research on syntax and parsing has for a long time been hampered by the fact that annotation schemes vary enormously across languages, which has made it virtually impossible to perform sound comparative evaluations and cross-lingual learning experiments. A striking illustration of this problem can be found in Figure 1, which shows three parallel sentences in Swedish, Danish and English, annotated according to the guidelines of the Swedish Treebank (Nivre and Megyesi, 2007), the Danish Dependency Treebank (Kromann, 2003), and Stanford Typed Dependencies (de Marneffe et al., 2006), respectively. The syntactic structure is identical in the three languages, but the percentage of shared dependency relations across pairs of languages is at most 40% (and 0% across all three languages). As a consequence, a parser trained on one type of annotation and evaluated on another type will be found to have at least a 60% error rate when it functions perfectly.

The Universal Dependencies (UD) project seeks to tackle this problem by developing cross-linguistically consistent treebank annotation for many languages, aiming to capture similarities as well as idiosyncracies among typologically different languages (e.g., morphologically

sv: en katt jagar råttor
da: en kat jager rotter
en: a cat chases rats

Figure 1: Divergent annotation of parallel s

Google dependency scheme (Universal Dependen
banks) (McDonald et al., 2013), the Google univers
of-speech tags (Petrov et al., 2012), and the Interset
lingua for morphosyntactic tag sets (Zeman, 2008) us
the HamleDT treebanks (a project that transforms exist
treebanks under a common annotation scheme, Zeman
al. 2012). UD is thus based on common usage and existing
de facto standards, and is intended to replace all
ous versions by a single coherent
philosophy is to pr

UD Treebanks

Amharic	–			?	–		
Ancient Greek	244K	LFE			✓		
Ancient Greek-PROIEL	206K	LFE			✓		
Arabic	242K	LFE			✓		
Arabic-LDC	–						
Basque	121K	LFE			✓		
Bulgarian	156K	LFE			✓		
Buryat	9K	L			✓		
Cantonese	–						
Catalan	530K	LFE			✓		
Chinese	123K	F			✓		
Chinese-HK	–						
Coptic	5K	L			✓		
Croatian	139K	LFE			✓		
Czech	1,503K	LFE			✓		
Czech-CAC	493K	LFE			✓		
Czech-CLTT	35K	LFE			✓		
Danish	100K	LFE			✓		
Dutch	209K	LFE			✓		
Dutch-LassySmall	98K	LFE			✓		
English	248K	L			✓		
English-ESL	97K				✓		
English-LinES	82K	LFE			✓		
Estonian	234K	F			✓		
Faroese	132K	LFED			✓		
Finnish	181K	LFE			✓		
Finnish-FTB	159K	LFE			✓		
French	391K	L			✓		
Galician	138K	LFE			✓		
Galician-TreeGal	24K	LFE			✓		
German	293K	LFE			✓		
Gothic	56K	LFE			✓		
Greek	59K	F			✓		
Hebrew	115K	LFE			✓		
	351K	LFE			✓		
	42K				✓		

A project that is developing cross-linguistically consistent treebank annotation for many languages

Evaluation Metrics

Unlabeled Attachment Score	Labeled Attachment Score
The percentage of words that have the correct head.	The percentage of words that have the correct head and correct type along the arc from the head to the word.

Precision	Recall	F1-measure
The percentage of dependencies with a specific type in the parser output that were correct.	The percentage of dependencies with a specific type in the test set that were correctly parsed.	The harmonic mean of precision and recall.

Syntactic Parsing
Summary

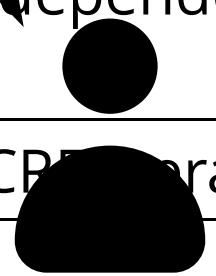
Constituency Parsing	Dependency Parsing
CKY Algorithm	Eisner's Algorithm (projective)
Earley's Algorithm	MST Algorithm (non-projective or projective)

Training	Decoding
Generative	Probabilistic CFG or probabilistic dependency rules
Discriminative	Perceptron, CRF, Reranking

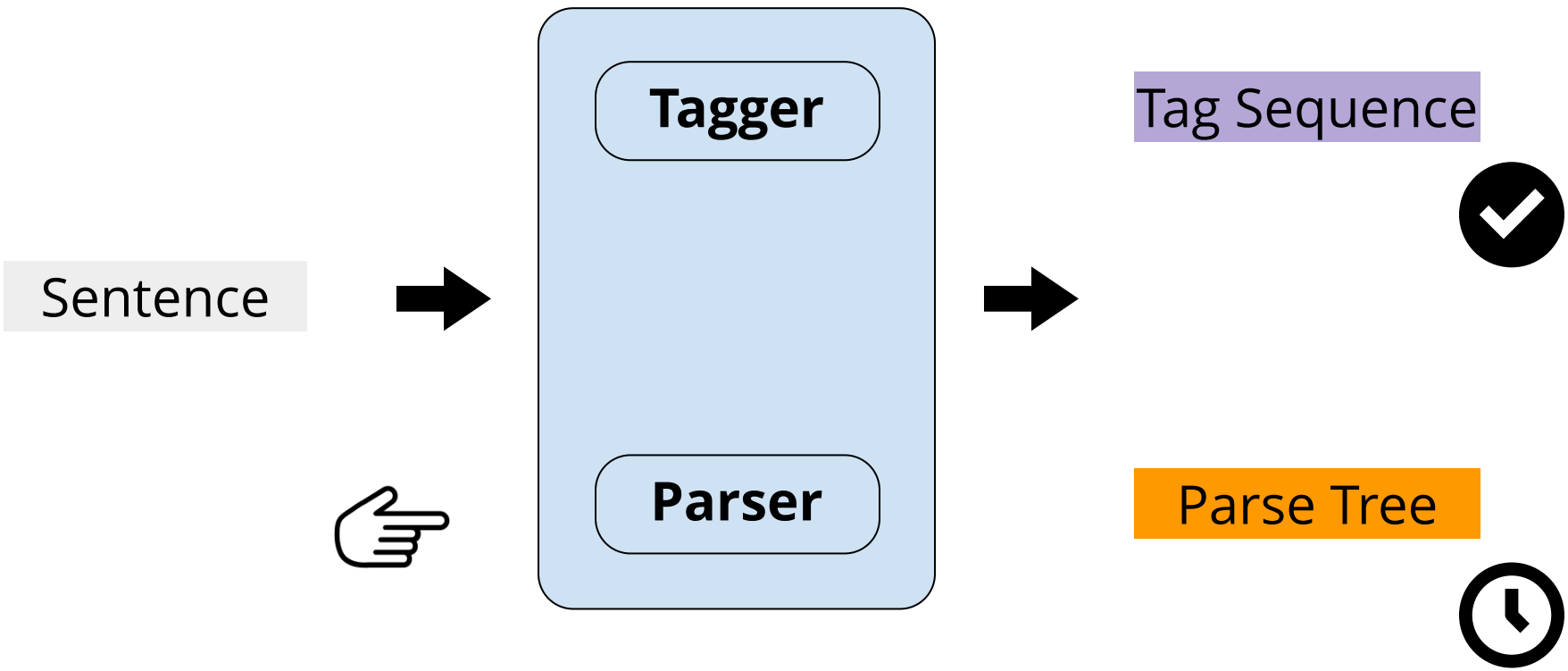
Syntactic Parsing
Summary

Constituency Parsing	Dependency Parsing
Training	Decoding
Generative	Probabilistic CFG or probabilistic dependency rules
Discriminative	Perceptron, CRF, reranking

Next time I'll tell you a new paradigm that builds the parse trees using an incremental approach through a sequence of *actions*!



Structured Prediction



Structured Space $\mathcal{S}_i$ | Structured Prediction Model M | Structured Space $\mathcal{S}_o$

$$\mathsf{M} : \mathcal{S}_i \rightarrow \mathcal{S}_o$$