dependency parsing

CS 585, Fall 2018

Introduction to Natural Language Processing

http://people.cs.umass.edu/~miyyer/cs585/

Mohit Iyyer

College of Information and Computer Sciences

University of Massachusetts Amherst

many slides from Marine Carpuat & Brendan O'Connor

questions from last time…

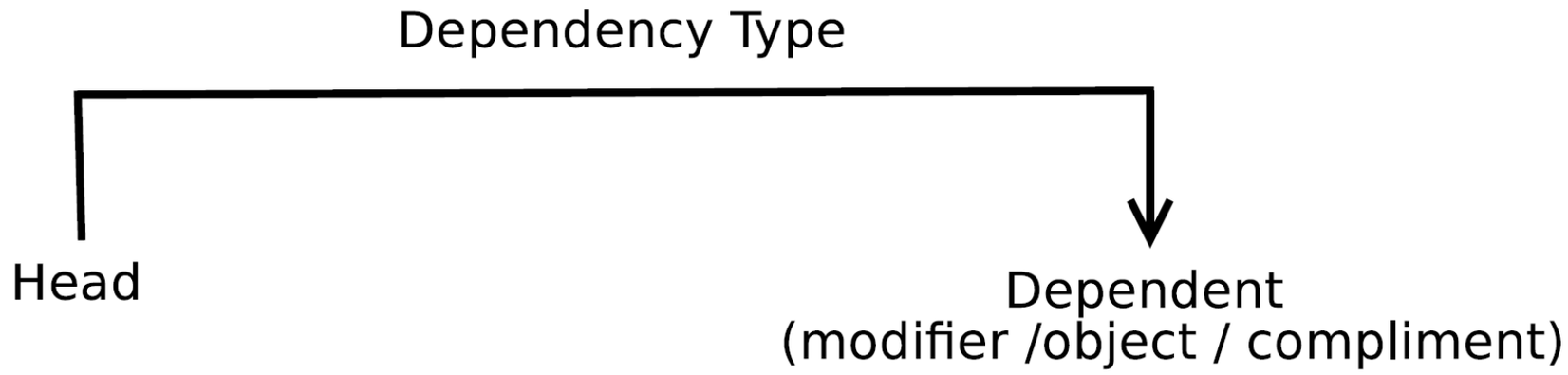
- can you “misplace” the midterms and give us a retest?
  - no
  - we’ll apply a curve when computing final grades, so don’t worry too much! unless you totally bombed it :(
- HW 3?
  - last HW (we’ll merge HWs 3 and 4)!
  - will be due *after* Thanksgiving and have an extra credit component
  - more time to work on your projects
    - **thus, i expect a significant amount of work to go into the progress reports (due nov 16)!**

more stuff

- Mohit out Thursday, guest lecture by Abe Handler (NLP PhD student)
  - will be helpful for your projects!
- No instructor office hours this Friday
- **what topics do you want to see covered towards the end of the class?**
  - **suggest things using the anonymous form or piazza or in person or whatever**

Dependency Grammars

- Syntactic structure = lexical items linked by binary asymmetrical relations called dependencies

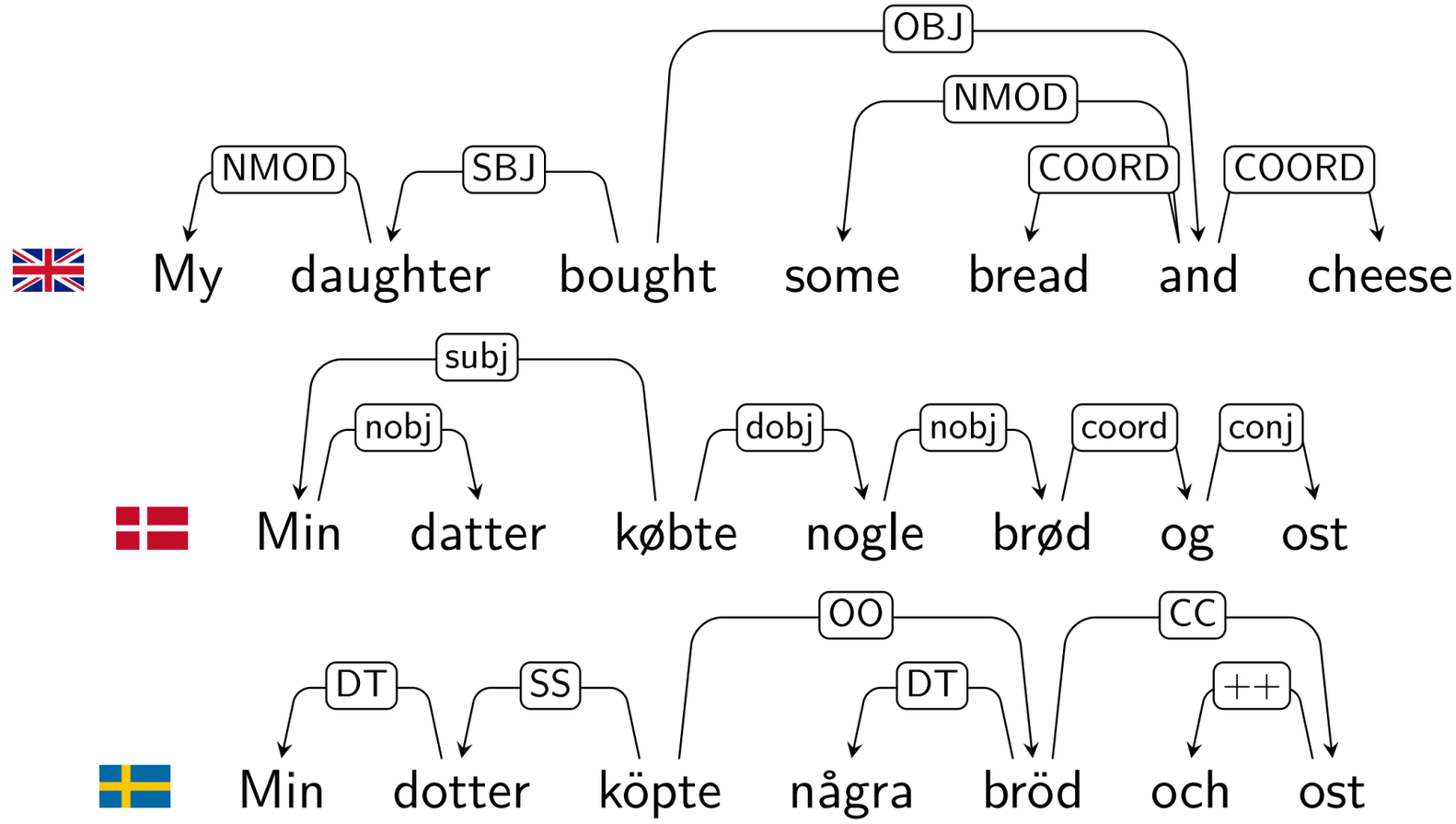


Dependency Relations

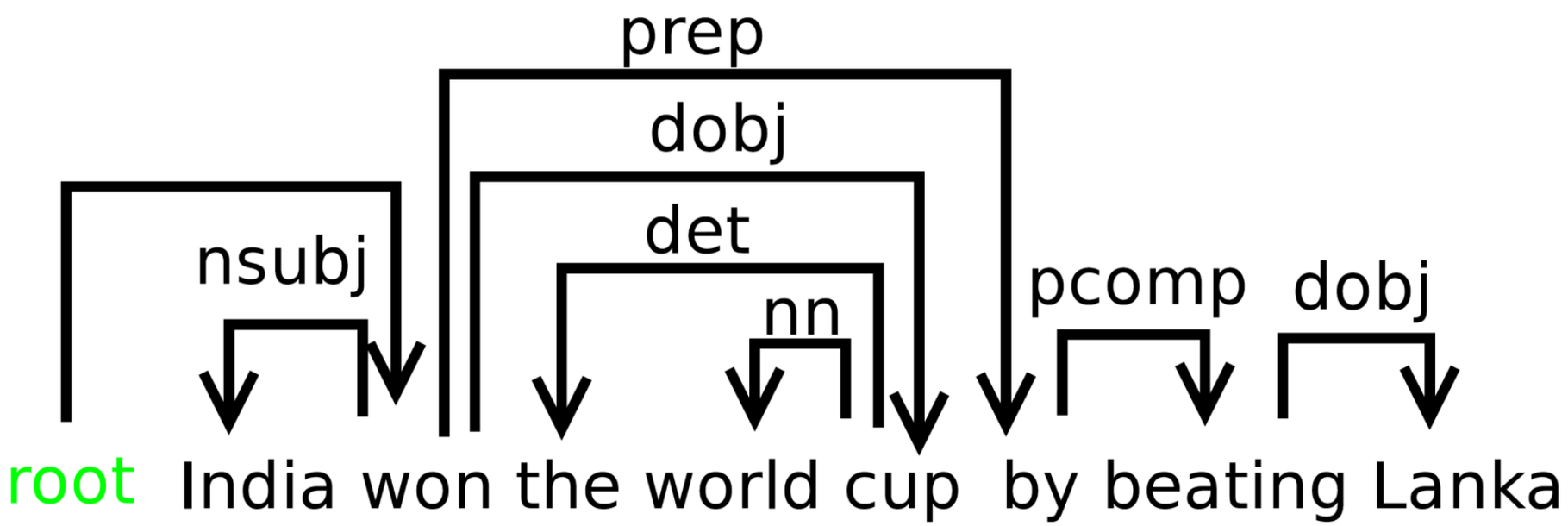
Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Figure 14.3 Examples of core Universal Dependency relations.



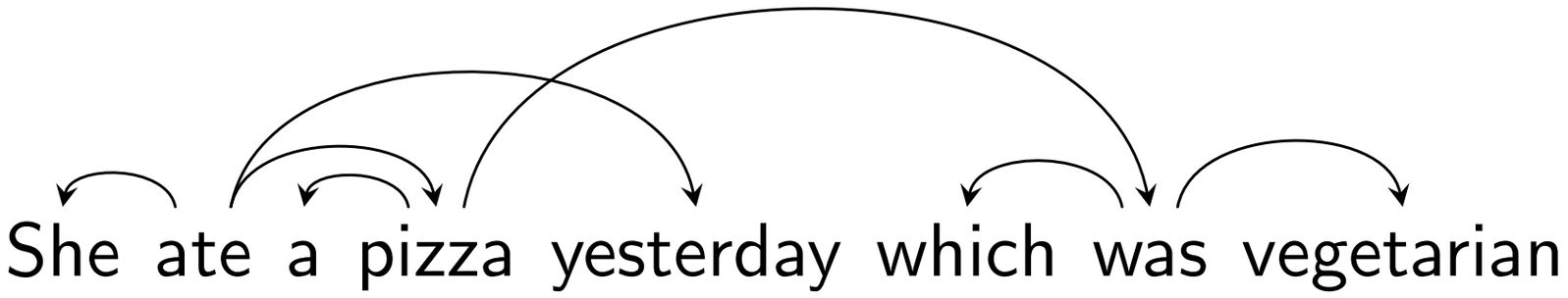
Example Dependency Parse



Projectivity

In **projective** dependency parsing, there are no crossing edges.

- Crossing edges are rare in English:



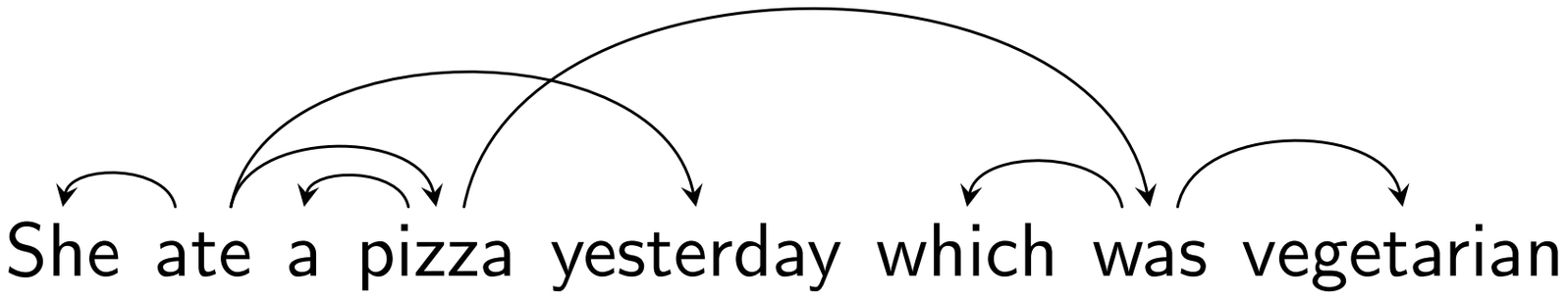
[2]figure from (Nivre 2007)

[Example: Jacob Eisenstein]

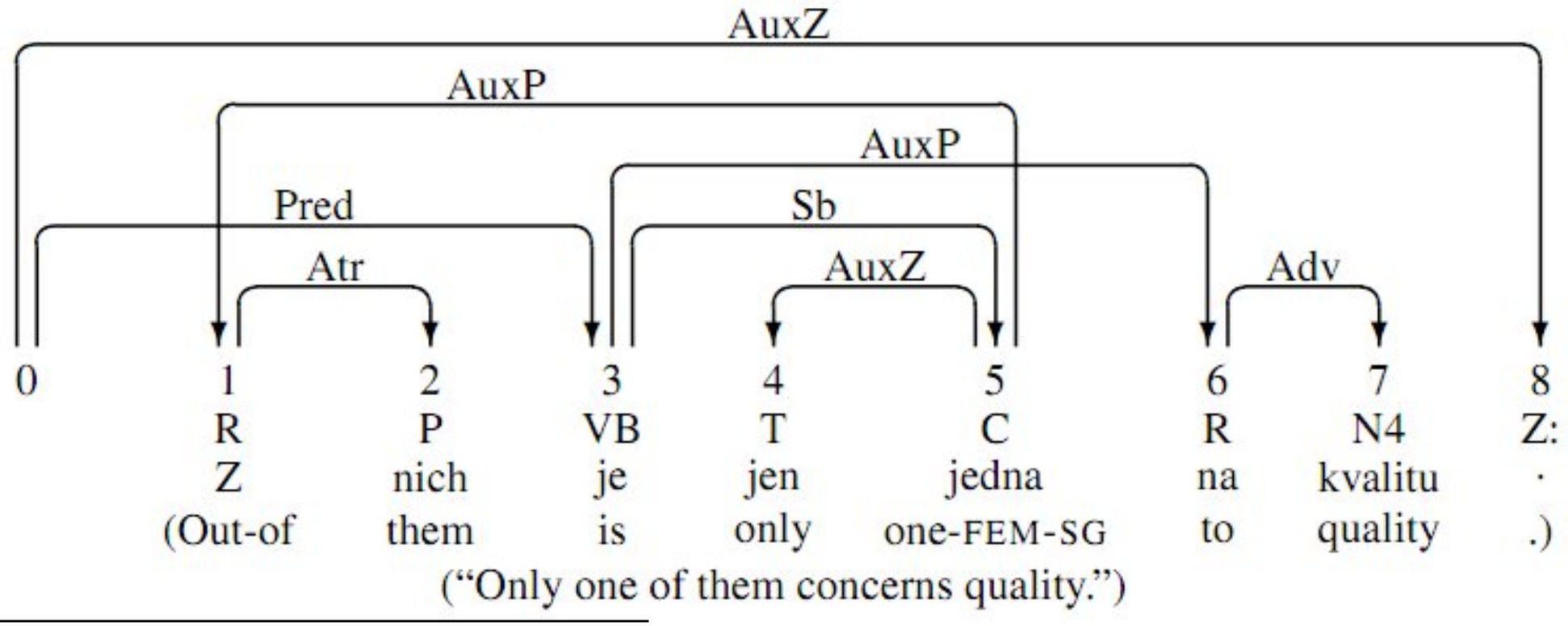
Projectivity

In **projective** dependency parsing, there are no crossing edges.

- Crossing edges are rare in English:



- They are more common in other languages, like Czech:[2]



[2] figure from (Nivre 2007)

[Example: Jacob Eisenstein]

Projectivity

	% non-projective edges	% non-projective sentences
Czech	1.86%	22.42%
English	0.39%	7.63%
German	2.33%	28.19%

Table 12.1: Frequency of non-projective dependencies in three languages (Kuhlmann and Nivre, 2010)

[Example: Jacob Eisenstein]

Dependency formalisms

- Most general form: a graph G = (V,A)
  - V vertices: usually one per word in sentence
  - A arcs (set of ordered pairs of vertices): head-dependent relations between elements in V
- Restricting to **trees** provide computational advantages
  - Single designated ROOT node that has no incoming arcs
  - Except for ROOT, each vertex has exactly one incoming arc
  - Unique path from ROOT to each vertex in V

- Each word has a single head
- Dependency structure is connected
- There is a single root node from which there is a unique path to each word

Data-driven dependency parsing

Goal: learn a good predictor of dependency graphs

Input: sentence

Output: dependency graph/tree G = (V,A)

Can be framed as a structured prediction task

- very large output space
- with interdependent labels

2 dominant approaches: transition-based parsing and graph-based parsing

Transition-based dependency parsing

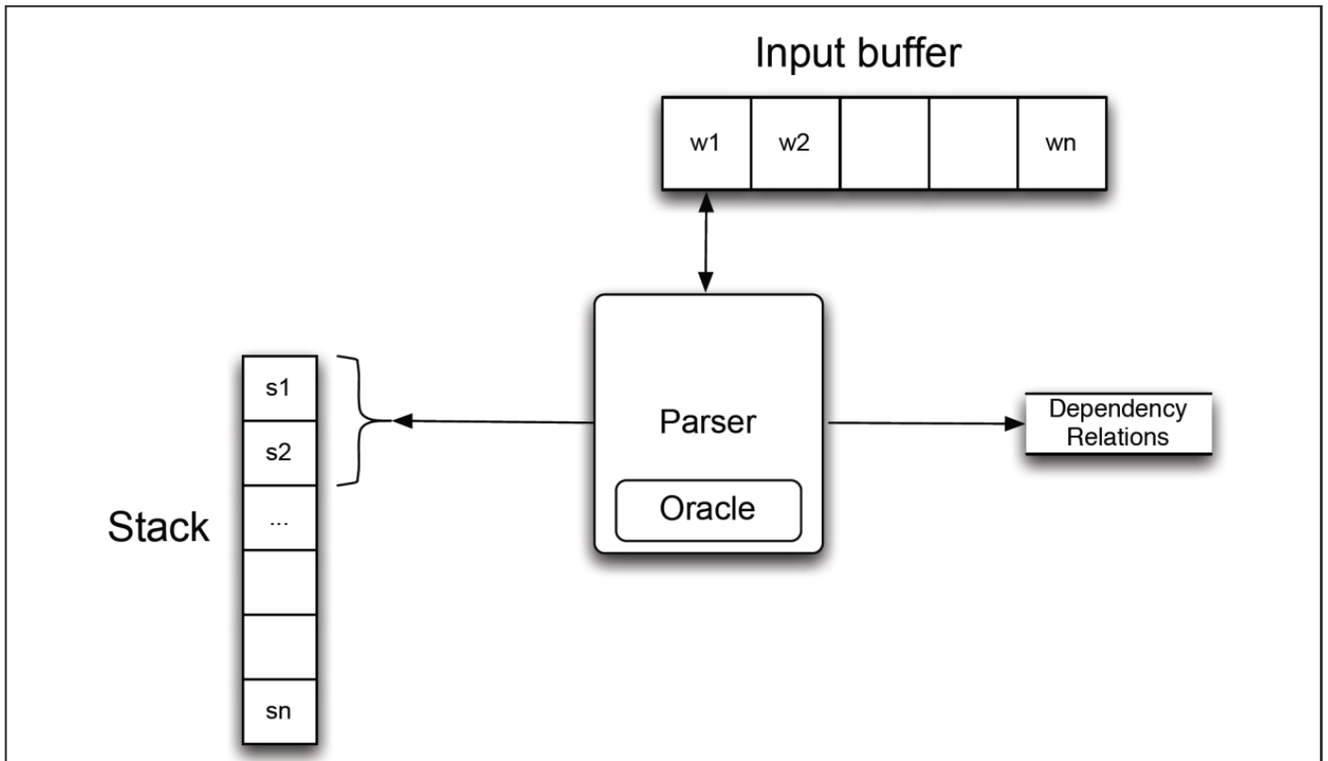


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

- Builds on shift-reduce parsing
  [Aho & Ullman, 1927]

- **Configuration**
  - **Stack**
  - **Input buffer** of words
  - Set of dependency relations

- Goal of parsing
  - find a final configuration where
  - all words accounted for
  - Relations form dependency tree

Transition operators

- Transitions: produce a new configuration given current configuration

- Parsing is the task of
  - Finding a sequence of transitions
  - That leads from start state to desired goal state

- Start state
  - Stack initialized with ROOT node
  - Input buffer initialized with words in sentence
  - Dependency relation set = empty

- End state
  - Stack and word lists are empty
  - Set of dependency relations = final parse

Arc Standard Transition System

- Defines 3 transition operators [Covington, 2001; Nivre 2003]
- LEFT-ARC:
  - create head-dependent rel. between word at top of stack and 2nd word (under top)
  - remove 2nd word from stack
- RIGHT-ARC:
  - Create head-dependent rel. between word on 2nd word on stack and word on top
  - Remove word at top of stack
- SHIFT
  - Remove word at head of input buffer
  - Push it on the stack

Arc standard transition systems

- Preconditions
  - ROOT cannot have incoming arcs
  - LEFT-ARC cannot be applied when ROOT is the 2nd element in stack
  - LEFT-ARC and RIGHT-ARC require 2 elements in stack to be applied

Transition-based Dependency Parser

```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← ORACLE(state)         ; choose a transition operator to apply
    state ← APPLY(t, state)   ; apply it, creating a new state
  return state
```

Figure 14.6 A generic transition-based dependency parser

- Assume an oracle
- Parsing complexity
  - Linear in sentence length!
- Greedy algorithm
  - Unlike Viterbi for POS tagging

example:

book me the morning flight

Where do we get an oracle???

Where do we get an oracle???

we have treebanks annotated with dependencies…

Where do we get an oracle???

we have treebanks annotated with dependencies…

We can treat finding the correct action as a multi-class classification problem!

input: current parser state (stack / buffer / prev actions)

output: ground-truth action from converted treebank

Where do we get an oracle???

we have treebanks annotated with dependencies…

We can treat finding the correct action as a multi-class classification problem!

input: current parser state (stack / buffer / prev actions)

output: ground-truth action from converted treebank

How many possible actions are there?

Where do we get an oracle???

we have treebanks annotated with dependencies…

We can treat finding the correct action as a multi-class classification problem!

input: current parser state (stack / buffer / prev actions)

output: ground-truth action from converted treebank

How many possible actions are there?

shift

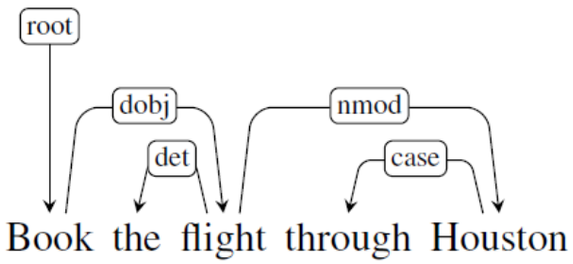
right-arc(X)

left-arc(X)

X is any dependency relation!!!

Generating Training Examples

- What we have in a treebank



- What we need to train an oracle
  - Pairs of configurations and predicted parsing action

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Figure 14.8 Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.

Features

- Configuration consist of stack, buffer, current set of relations

- Typical features
  - Features focus on top level of stack
  - Use word forms, POS, and their location in stack and buffer

Features example

- Given configuration

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United) (flights → morning) (flights → the)

what kind of model can take features like these as input?

can we use a neural network for this task? how?

- Example of useful features

$$\langle s_1.w = flights, op = shift \rangle$$
$$\langle s_2.w = canceled, op = shift \rangle$$
$$\langle s_1.t = NNS, op = shift \rangle$$
$$\langle s_2.t = VBD, op = shift \rangle$$
$$\langle b_1.w = to, op = shift \rangle$$
$$\langle b_1.t = TO, op = shift \rangle$$
$$\langle s_1.wt = flightsNNS, op = shift \rangle$$
$$\langle s_1t.s_2t = NNSVBD, op = shift \rangle$$

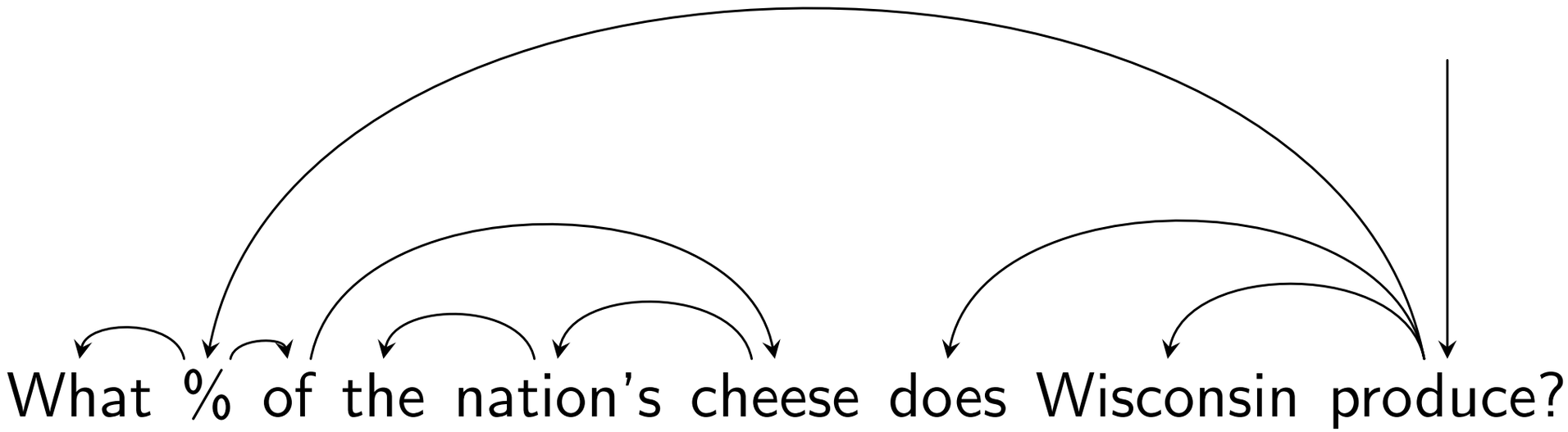
Dependency parsing in action

Dependency parsing is used in many real-world applications, like question answering (Cui et al, 2005):

[Example: Jacob Eisenstein]

Dependency parsing in action

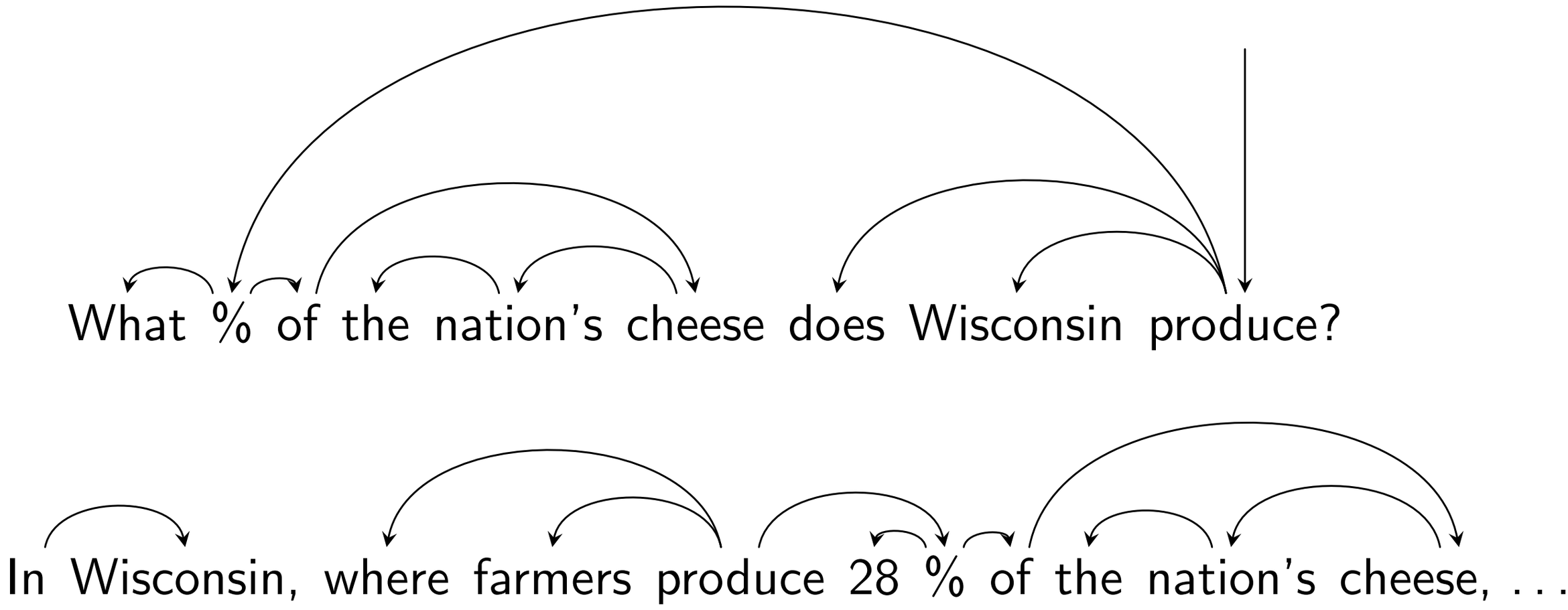
Dependency parsing is used in many real-world applications, like question answering (Cui et al, 2005):



[Example: Jacob Eisenstein]

Dependency parsing in action

Dependency parsing is used in many real-world applications, like question answering (Cui et al, 2005):

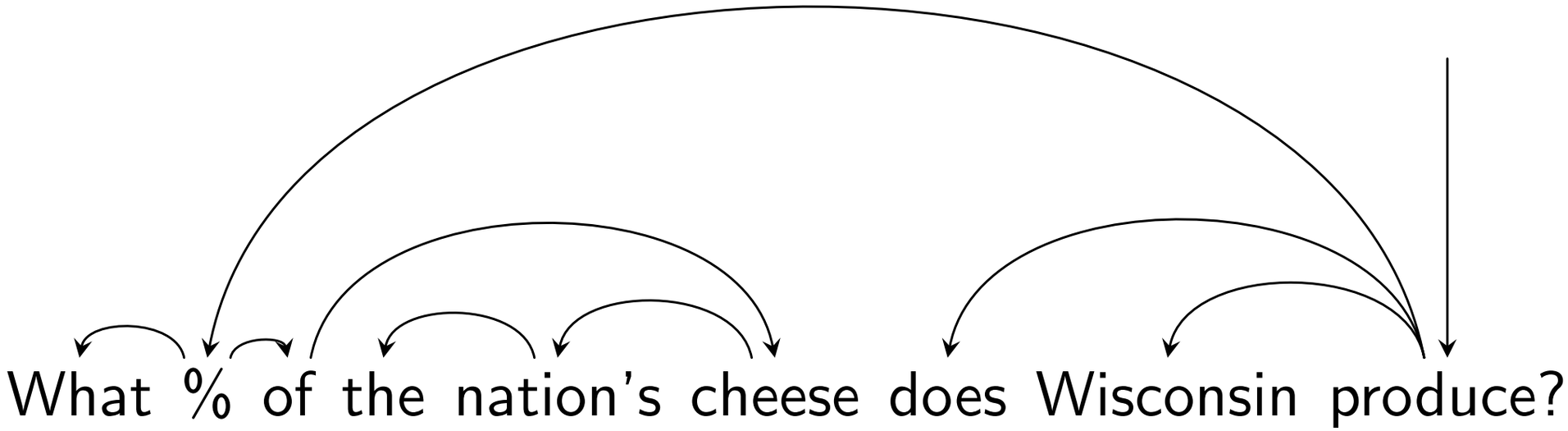


[Example: Jacob Eisenstein]

Dependency parsing in action

Question answering works by searching for statements which match well against the query.

- In the surface form of the question, *produce* and *%* are six words apart.
- But in the dependency parse, they're adjacent.

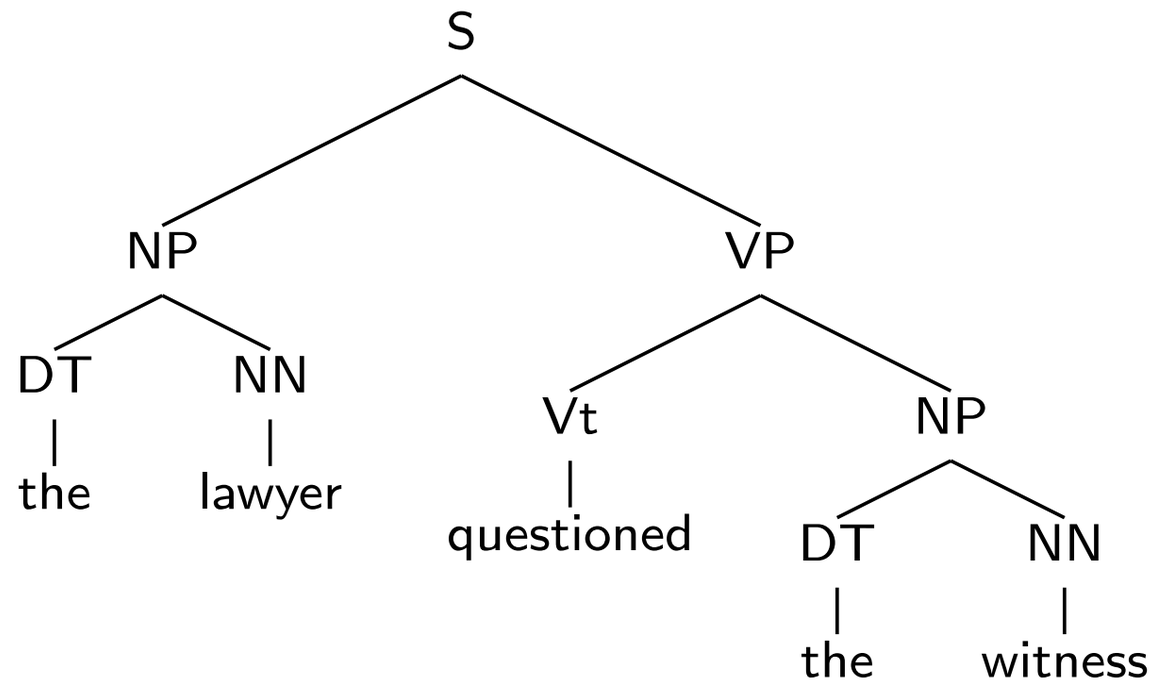


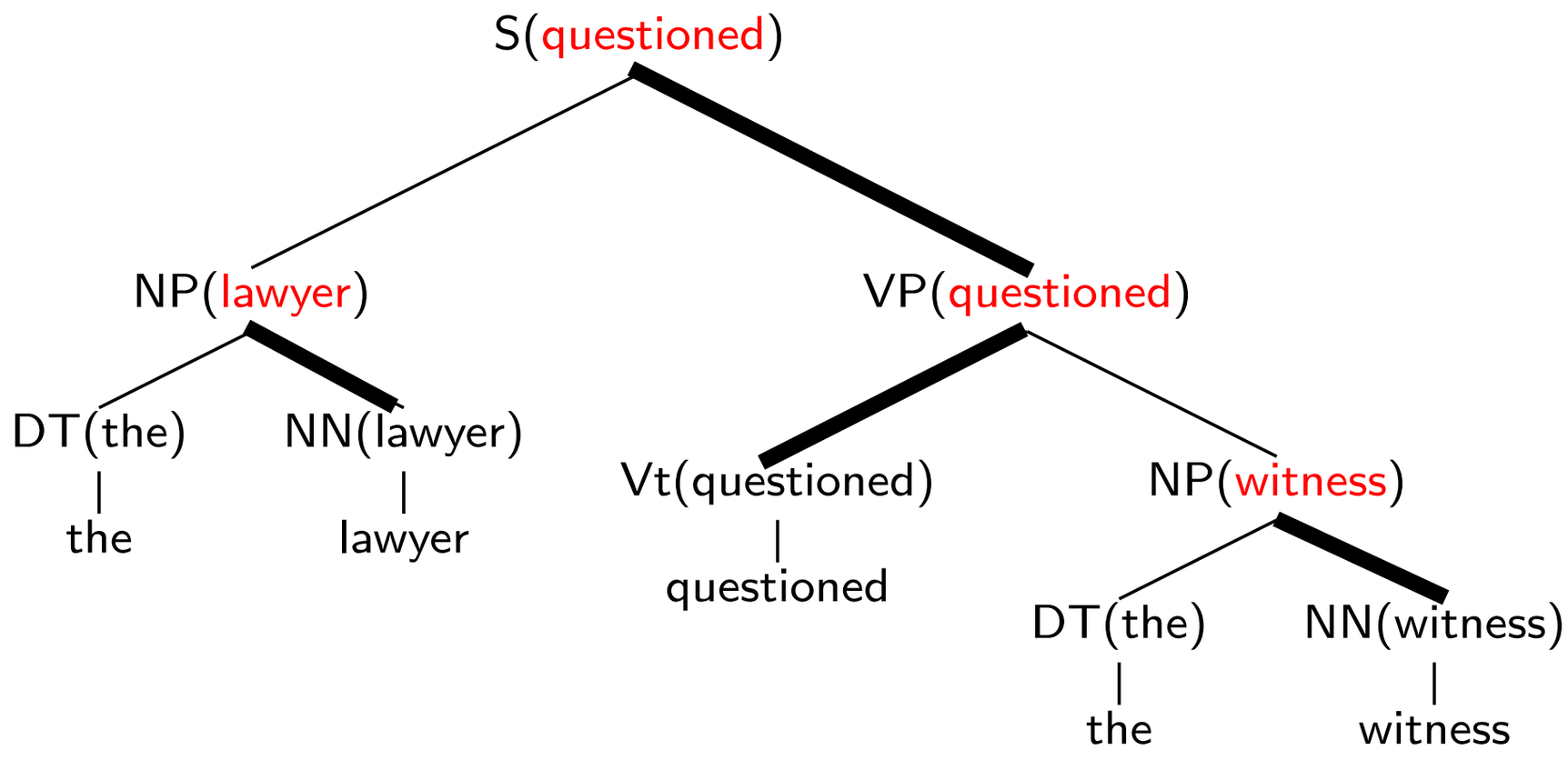
[Example: Jacob Eisenstein]

Constits -> Deps

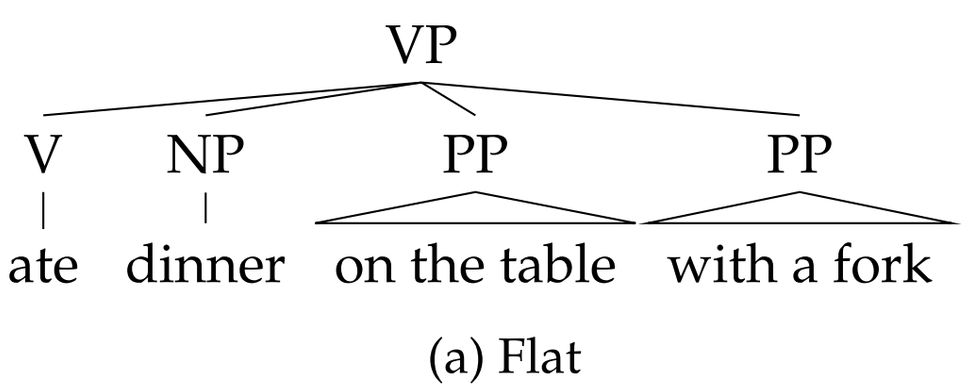
- Every phrase has a head word. It dominates all other words of that phrase in the dep. graph.
- Head rules: for every nonterminal in tree, choose one of its children to be its "head". This will define head words.
- Every nonterminal type has a different head rule; e.g. from Collins (1997):

- If parent is NP,
  - Search from right-to-left for first child that's NN, NNP, NNPS, NNS, NX, JJR
  - Else: search left-to-right for first child which is NP

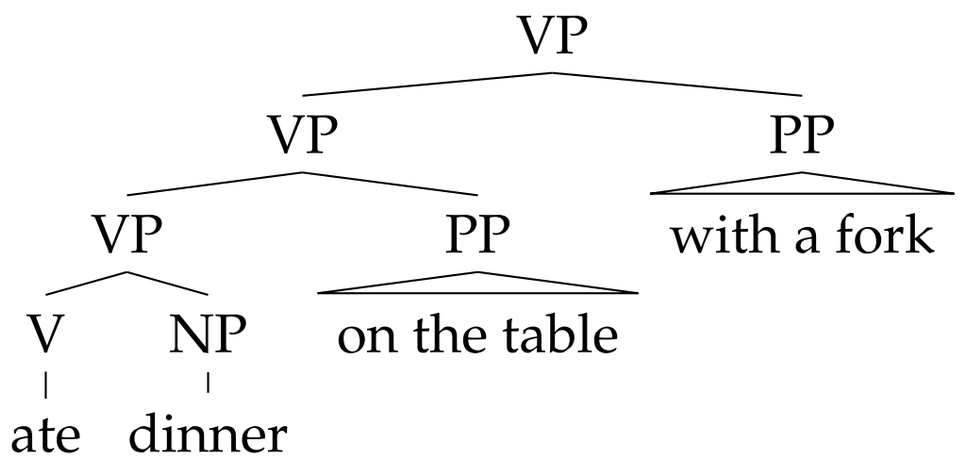

S
NP
VP
DT
NN
Vt
NP
the
lawyer
questioned
DT
NN
the
witness


S(questioned)
NP(lawyer)
VP(questioned)
DT(the)
NN(lawyer)
Vt(questioned)
NP(witness)
the
lawyer
questioned
DT(the)
NN(witness)
the
witness

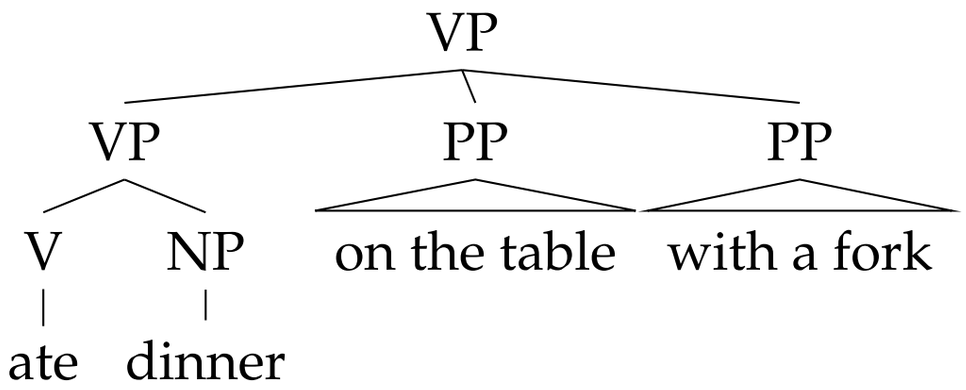
- Dependencies tend to be less specific than constituent structure



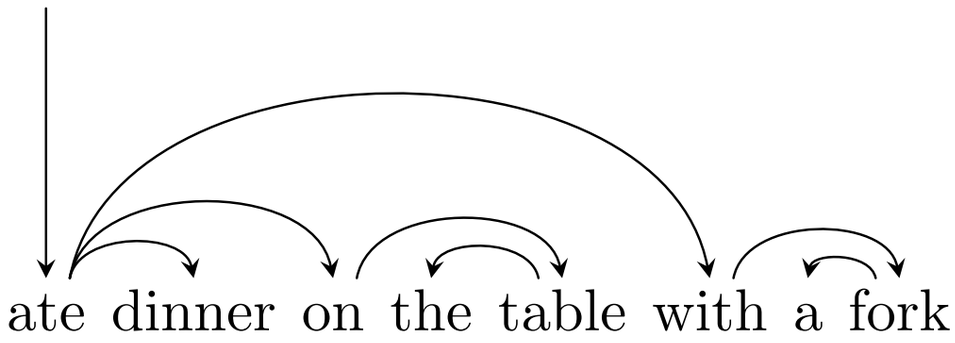
(a) Flat



(b) Two-level (PTB-style)



(c) Chomsky adjunction



(d) Dependency representation

[Example: Jacob Eisenstein]

Parsing to dependencies

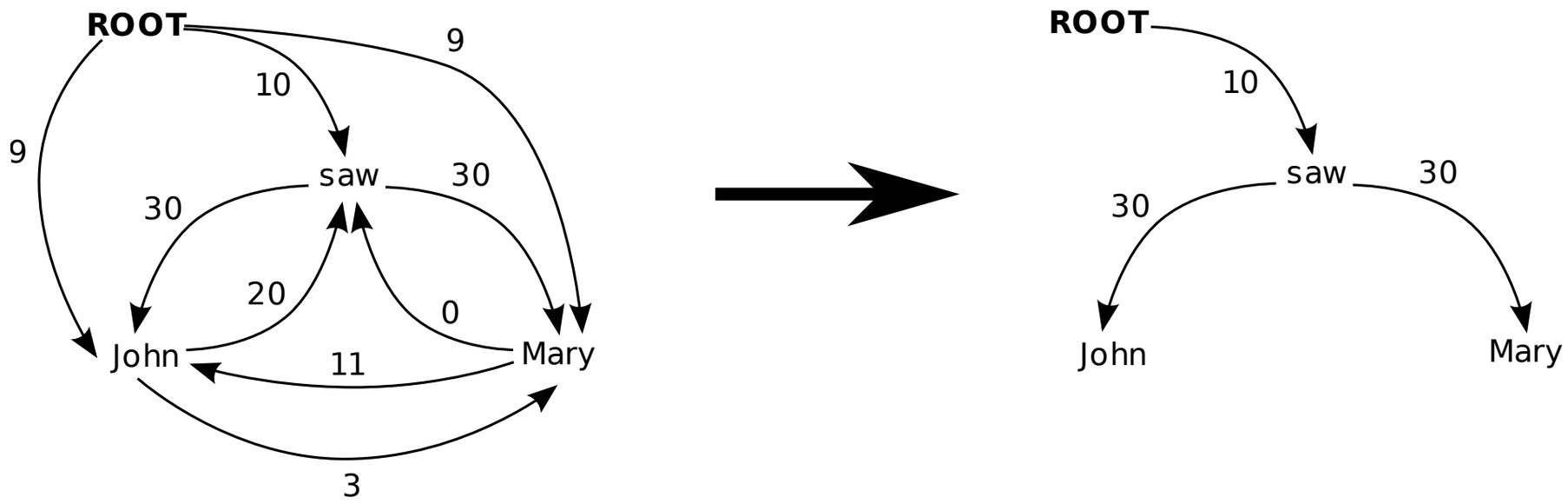
- Constituents -> Dependency conversion is one approach

- Direct dependency parsing more common
  - Annotating dependencies is easier
  - http://universaldependencies.org/

- Algorithmic approaches
  - Graph-based: global CRF-style models
  - History-based: shift-reduce (Nivre)

Graph-based parsing

Edge scoring models



Inference: minimum spanning tree algorithms
Learning: structured perceptron/svm

[Slides: McDonald and Nivre, EACL 2014 tutorial]

Linear vs neural features

- Non-stateful
  - Nivre (~2003 & others), "MALT": linear SVM to make shift-reduce decisions, trained on oracle decisions
  - Chen and Manning (2014): neural softmax, trained on oracle decisions
  - Andors et al. (2016), "SyntaxNet": similar but with CRF-style global normalization
- Stateful: recurrent neural networks over sentence or state transitions

Greedy, Local, Transition-Based Parsing

- Advantages:
  - Highly efficient parsing – linear time complexity with constant time oracles and transitions
  - Rich history-based feature representations – no rigid constraints from inference algorithm
- Drawback:
  - Sensitive to search errors and error propagation due to greedy inference and local learning
- The major question in transition-based parsing has been how to improve learning and inference, while maintaining high efficiency and rich feature models

Better search

- Greedy decoding: errors can propagate (e.g. garden paths!)
- Why not Viterbi?
- Beam search
  - Beam: contain K automaton states (partial parses)
  - Iterate until done:
    - For each item on beam: enumerate expansions
    - Take top-K scoring items from ALL expansions, as the new beam.
  - Take top item on final beam as solution
- Most common heuristic search strategy for left-to-right NLP models (incl. generation, machine translation)