Morphology and Syntax, III

Dependency Grammar

Aaron Mueller
CAS CS 505: Natural Language Processing
Boston University
Spring 2026

Admin

- There is a poll on Piazza about releasing HW3 solutions. Please vote by tonight!

- The exam will be this coming **Tuesday, Mar. 24**, in this classroom, at our usual class time (3:30 – 4:45).
    - You can bring an 8.5” x 11” note sheet. You can use both sides of the paper. You’ll turn this in alongside your exam.
    - Remember to bring a pencil!
    - No electronics allowed, like phones, laptops, or calculators.

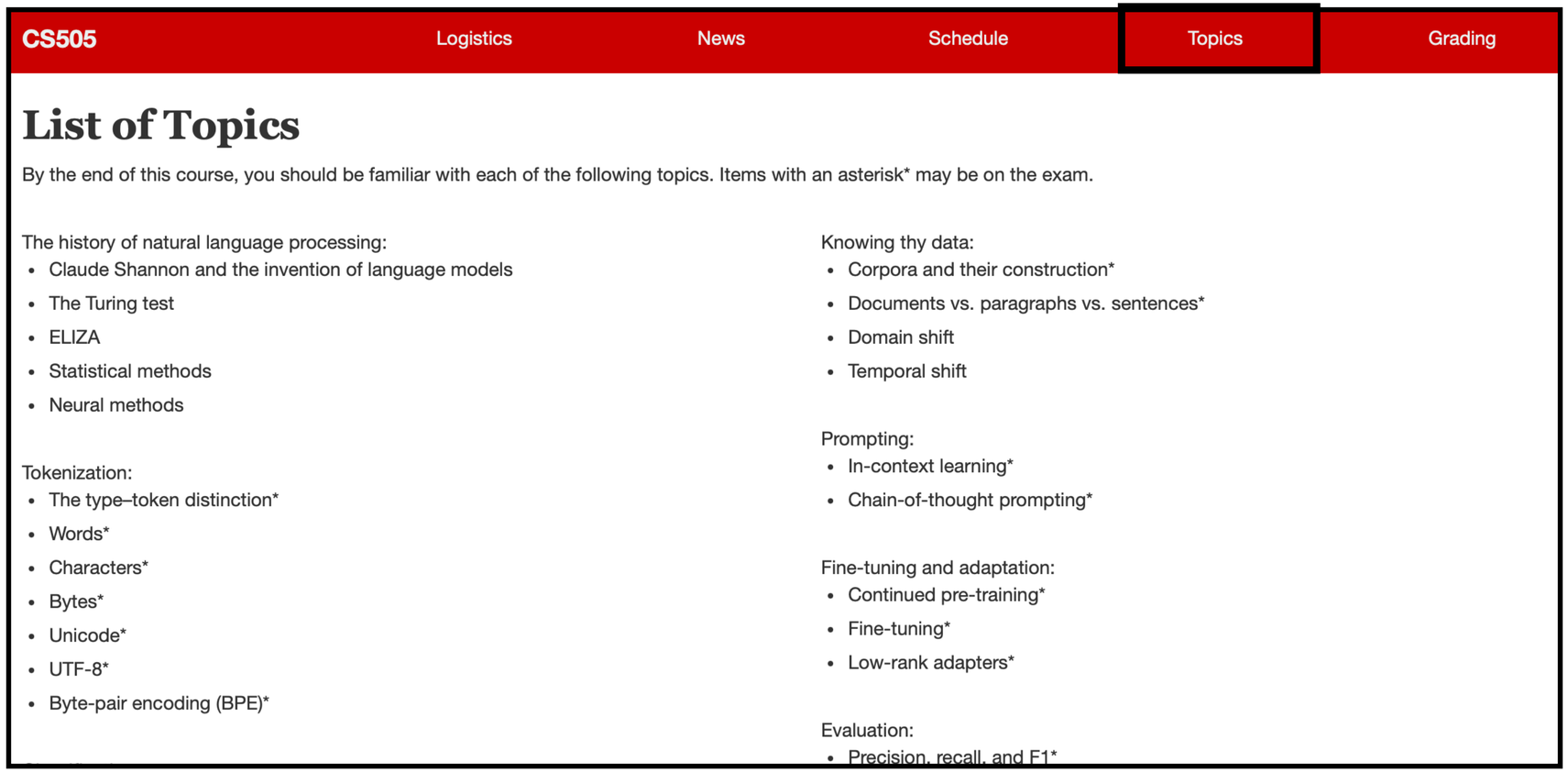
- When will exam grades be released? 1 – 1.5 weeks after the exam.

- When will HW2 grades be released? By the end of this week.

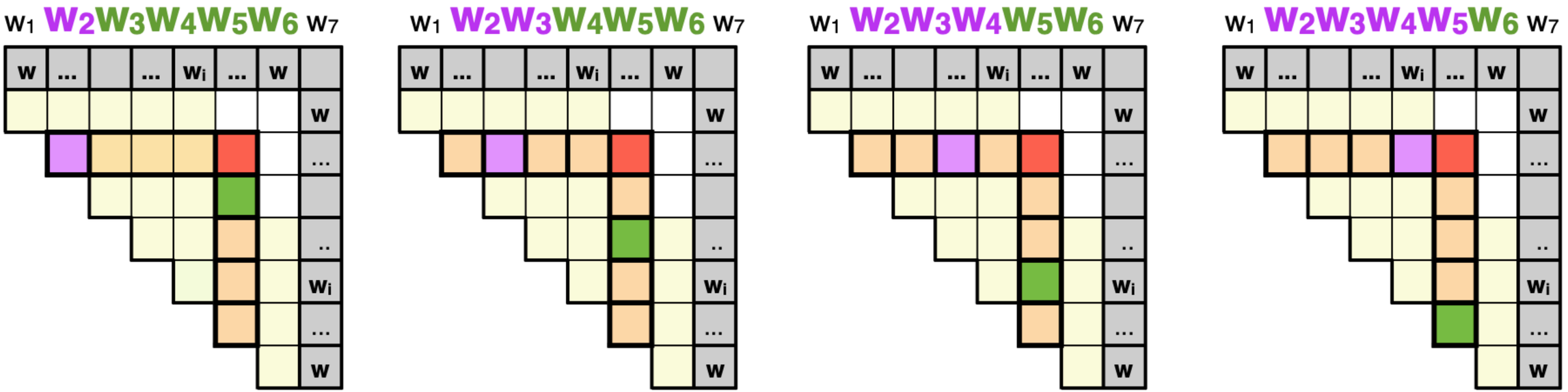
- Change to office hours next week: I’ll be available **Monday, Mar. 23, 5:00-7:00pm** at CDS 806. Stop by for last-minute exam questions and review or with final project Qs. No office hours next week on Tue. or Fri.
    - I will still have office hours tomorrow!

Exam Topics

The course site contains a list of topics. Make sure you're comfortable with the topics with an asterisk*.



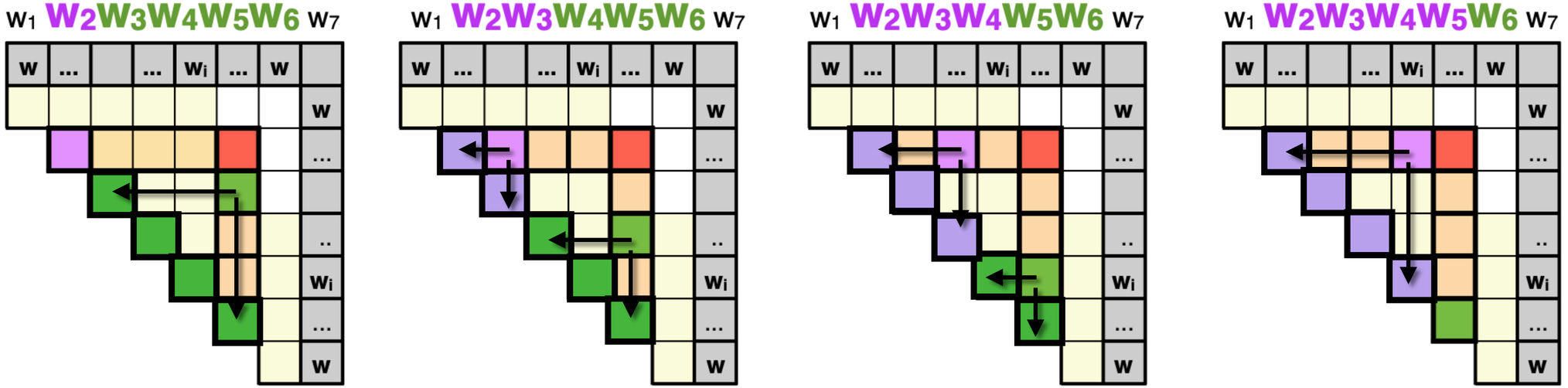
Quick follow-up on CKY



Why does CKY work like this? Why not just do all pairwise comparisons of cells to the left and below the red cell?

Quick follow-up on CKY

Each cell contains a partial parse of every word to its left and below it (and everything in between).



So if you try any other combination of cells, you might end up missing some words in your parse or with overlapping constituencies, which are not allowed.

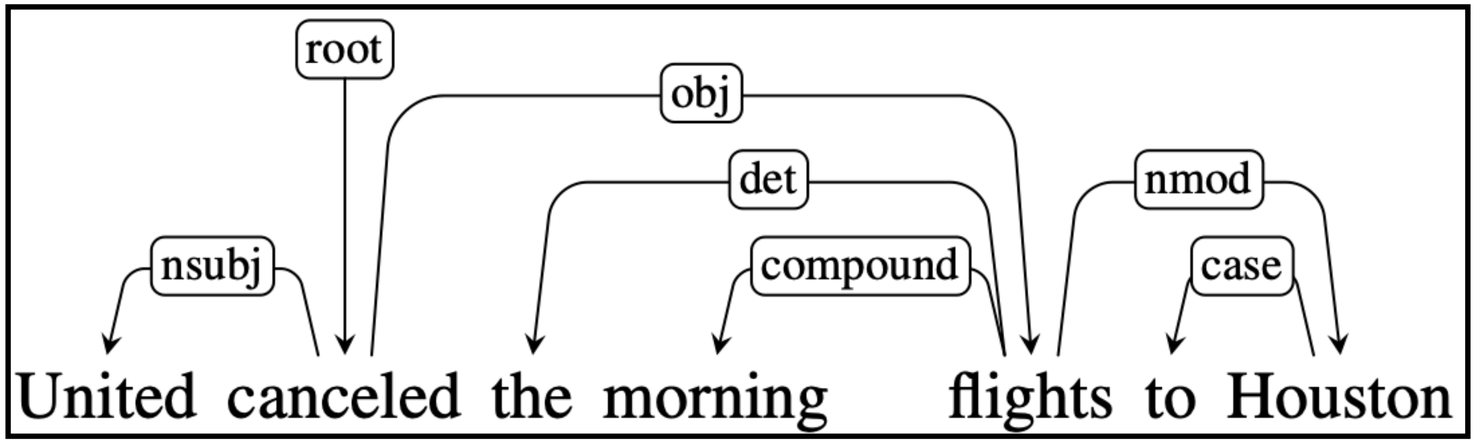
Overview of Concepts

Dependencies are syntactic relationships between pairs of words.

Projective parses are those with no crossing edges.

Shift-reduce parsing is an algorithm for mapping from a string to a dependency parse.

A **dependency relation** tells you about the type of syntactic relationship between two words.



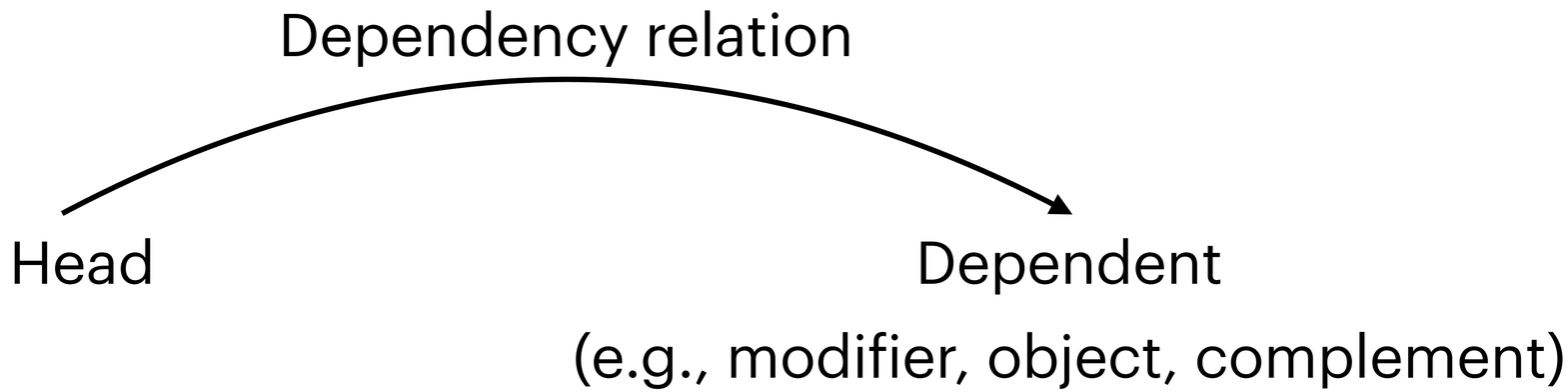
Last Time

- Context-free grammars (CFGs) and constituency grammars
- Constituencies can group words together into variable-length units:
    - [[The [elephant]] [eats [food]]]

- Today, we'll focus on a different grammar formalism: **dependency grammars**

Dependencies

Dependency grammars hold that syntactic structure consists of binary and asymmetric relations between words called **dependencies**.



Dependency Grammar

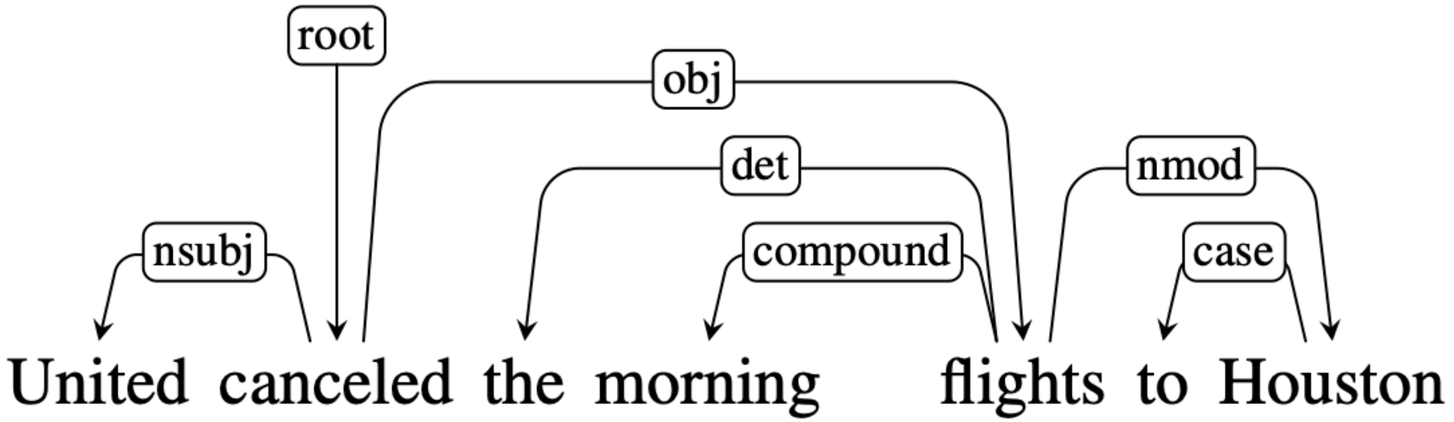
- Dependency grammars (DGs) assume that syntactic structure consists *only* of dependencies.

- DGs are descriptive (*not* generative).
    - Unlike CFGs, they cannot natively generate sentences. It's possible, but not straightforward.

- There exist some methods to convert between constituencies and dependencies that we'll discuss later.

Dependency Grammar

- Dependencies have **heads**, **dependents**, and **dependency relation types**.

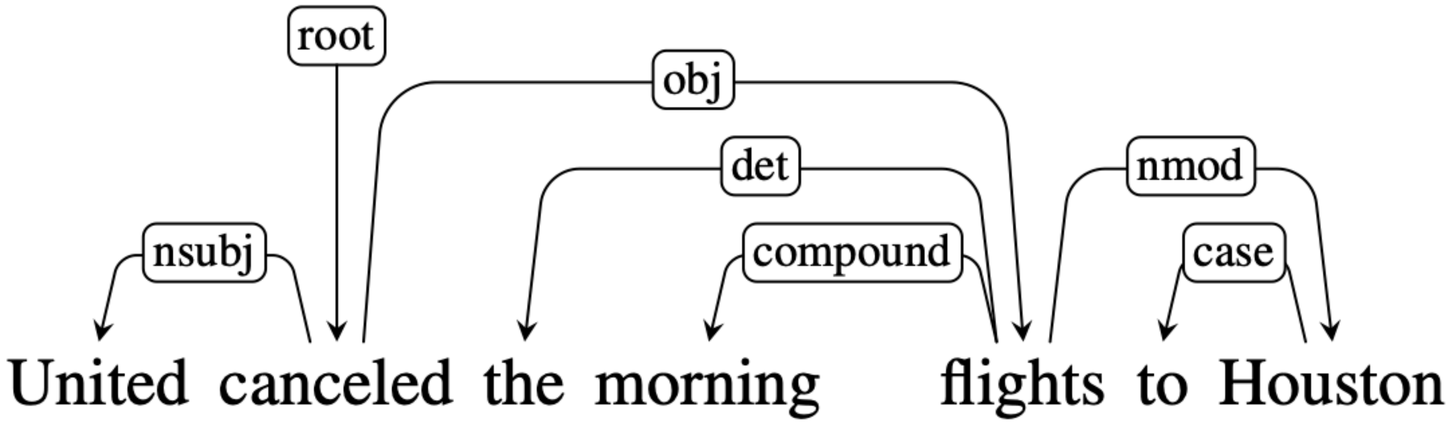


canceled → flights (“flights” is the object/dependent of “canceled”)

flights → morning (“morning” is an attribute/dependent of “flights”)

A dependency graph starts at the ROOT node.

Dependency Parses/Graphs



- Graph is **connected** (every word has one head) and **acyclic** (no loops)
- **Single-head constraint:** every vertex has <u>at most</u> one incoming arc
  - Equivalently: every dependent has <u>at most</u> one head
  - (But a head can have multiple dependents!)
- There is a single ROOT node that has no incoming arcs
- There is a <u>unique</u> path from ROOT to each vertex

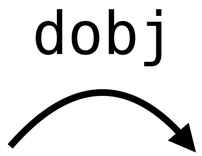
$G = (V, A)$

V: vertices (words)

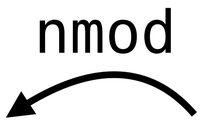
A: arcs (dependencies)

Dependency Relation Types

- Head-argument: eat food

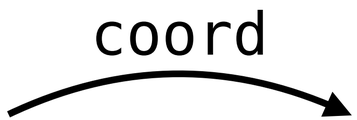


- Head-modifier: fresh food



- Head-specifier: this food

- Coordination: food and drinks



Dependency Relation Types

Clausal Argument Relations	Description
NSUBJ	Nominal subject
OBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Popularity Contest: Constituencies vs. Dependencies

- Constituencies are popular in linguistics, but dependencies are more popular with computer scientists.
    - Dependency parsing has a much faster algorithm.
- Dependencies are typically *easier to apply* for NLP tasks like question answering.
- In languages with free word order (like German or Latin), dependencies are more natural than constituency grammars (for reasons that will become clear today).
- Dependency treebanks exist for *many* languages
    - Universal Dependencies project: >150 languages!

CFGs vs. Dependency Grammars

Noam Chomsky

Lucien Tesnière

Constituencies

American structuralists and generativists

Part–whole relations

Trees and boxes

Dependencies

European tradition of “stemmas”

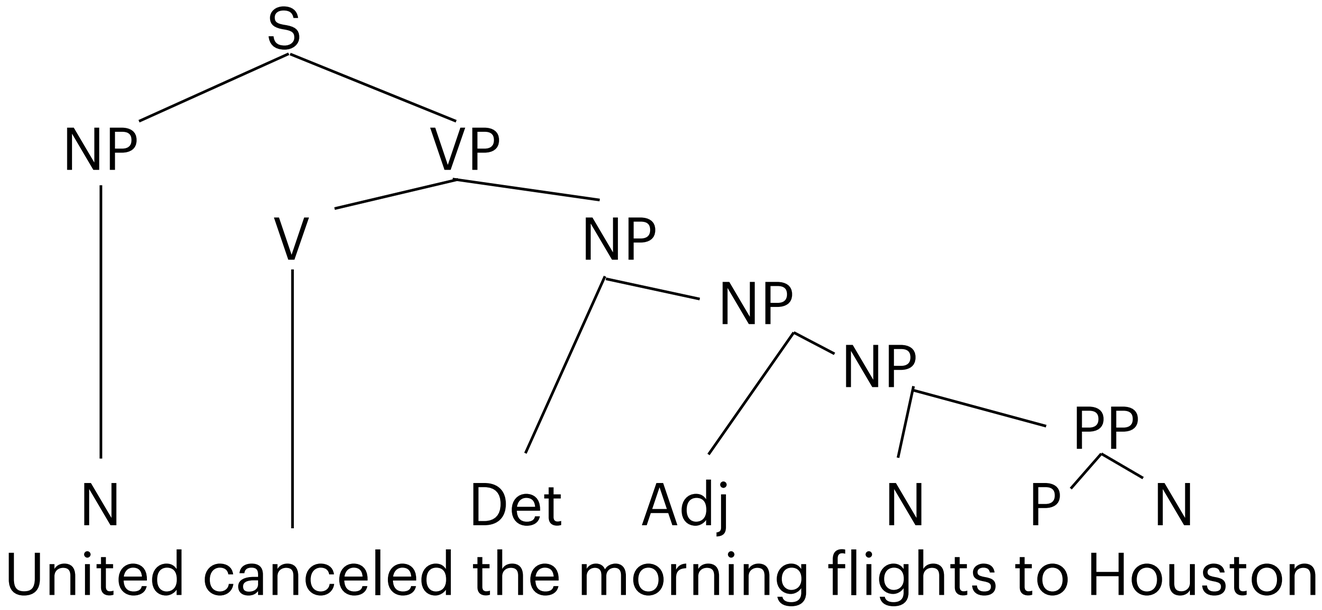
Part–part relations

Heads and modifiers

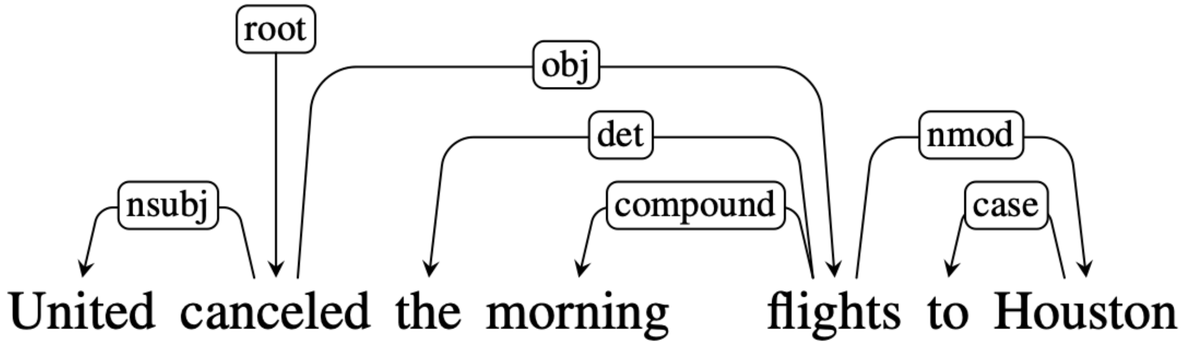
Some argue that dependencies are strictly necessary for syntax, while constituencies are not.

CFGs vs. Dependency Grammars

Constituencies assume that words can be grouped into higher-level structures:

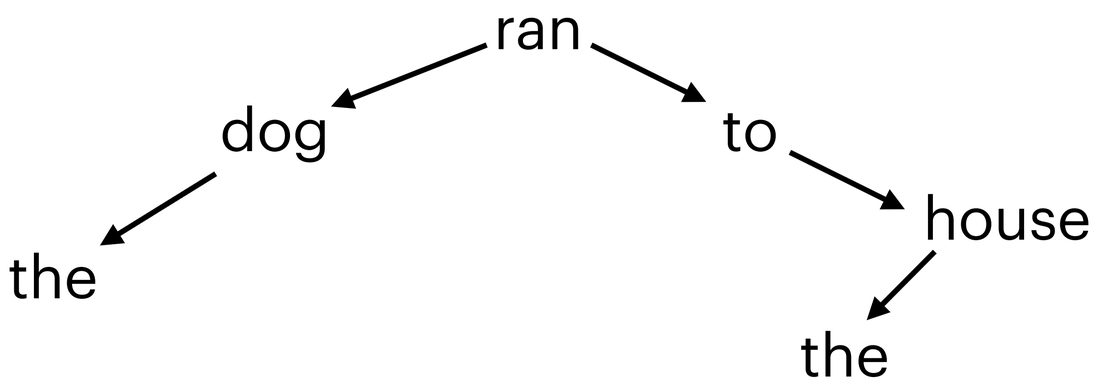


Dependency grammars only assume relationships between pairs of words:



Dependencies are still hierarchical and recursive.

With dependencies, we still have a notion of hierarchy! Subtrees align with constituents:

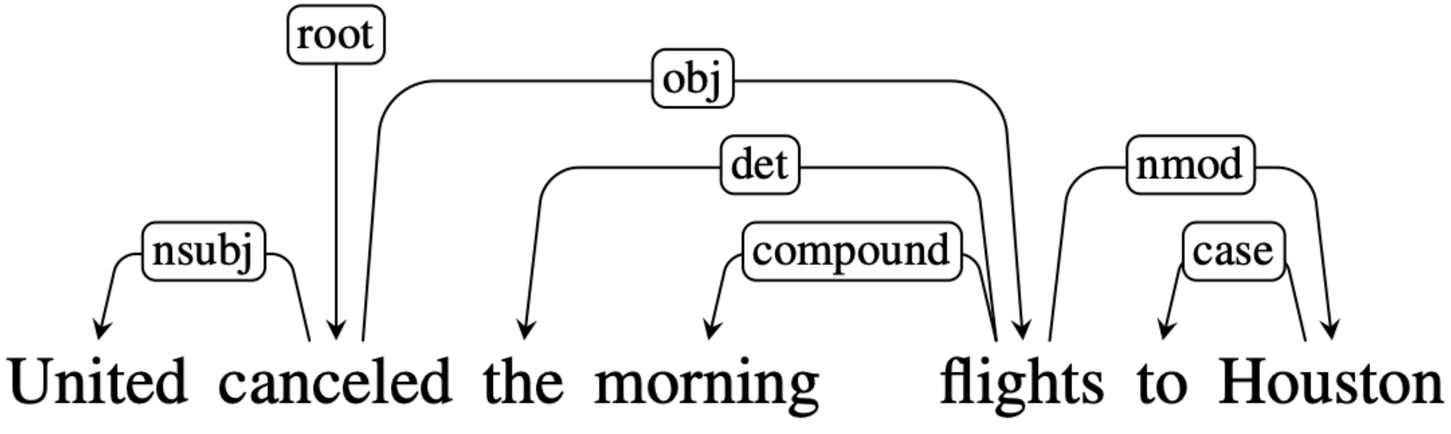


Dependencies are also still recursive, like CFGs. The head of a dependency can be a dependent of another head.

There’s no limit to how deep a dependency graph can get.

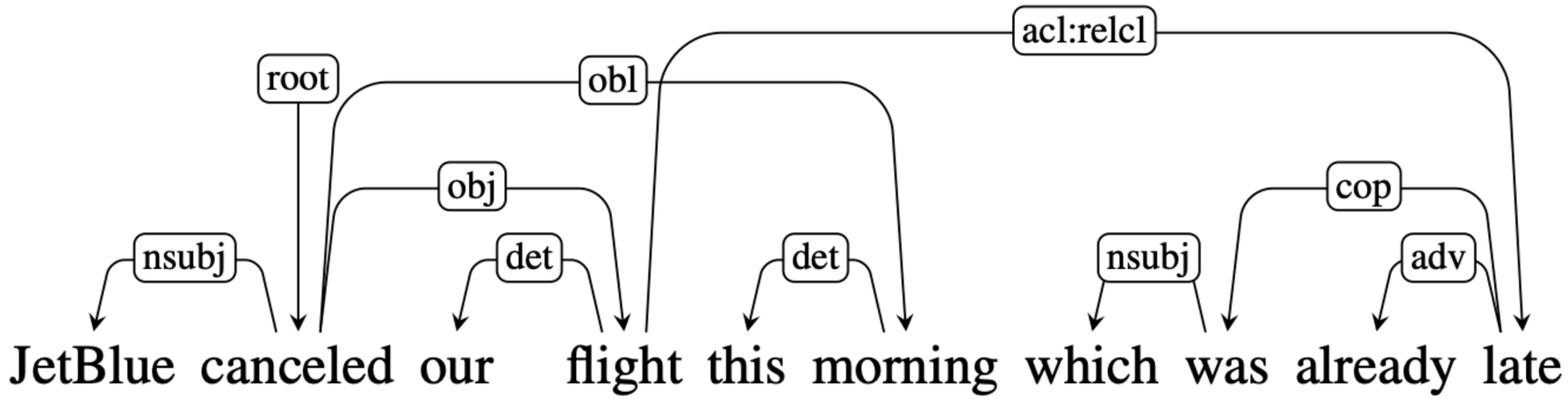
Projectivity

- CFGs only capture nested dependencies.
  - Their dependency graphs are **trees**
  - Their dependencies **do not cross**
- In other words, when you convert a constituency to a dependency parse, the dependency graph will always be **projective**.
  - (Actual definition: a word's full recursive set of dependents (and itself) are contiguous.)



Non-projectivity

- Crossing edges are rare, but possible in English. Crossing edges make a graph **non-projective.**
  - Example: *extraposition*



  - Example: *topicalization* (“Tuna, I thought the cat ate”)

Why care about projectivity?

We care about projectivity for two main reasons:

1. Most English dependency treebanks were derived from phrase-structure grammars like CFGs, so they are guaranteed to be projective
    - This means any systems trained on such data will fail when the actual dependency graph is non-projective

2. Most dependency parsing algorithms only work with projective trees

Projectivity Across Languages

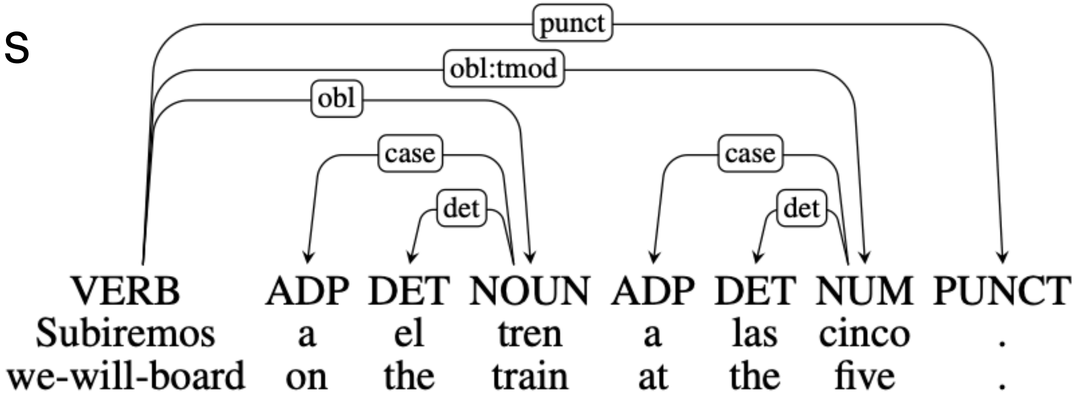
	% non-projective edges	% non-projective sentences
Czech	1.86%	22.42%
English	0.39%	7.63%
German	2.33%	28.19%

Non-projectivity is way more common in some languages than others.

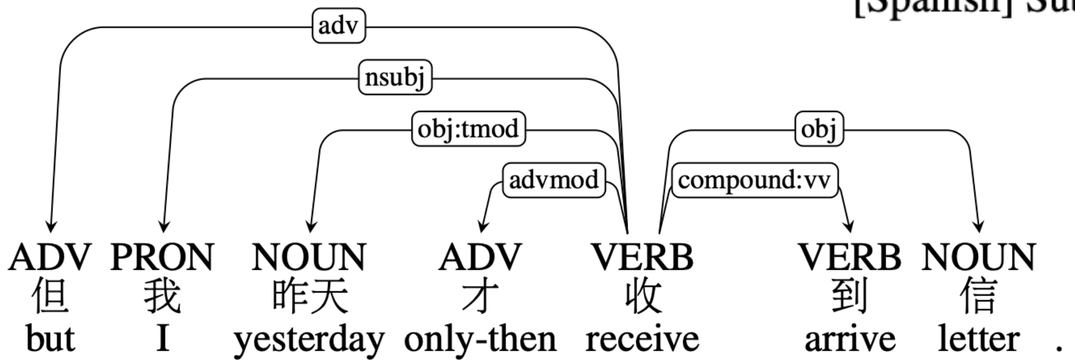
CFGs yield projective parses. So if a language naturally has a lot of non-projectivity, dependencies are often an easier formalism to work with.

Dependency Treebanks

- Treebanks are critical for developing and evaluating dependency parsers
- Universal Dependencies: >150 languages
  - 37 dependency relations, intended to be applicable to all languages



[Spanish] Subiremos al tren a las cinco. "We will be boarding the train at five." (19.4)



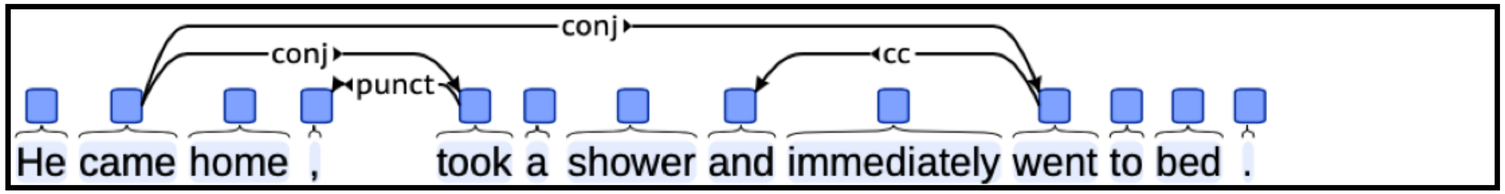
[Chinese] 但我昨天才收到信 "But I didn't receive the letter until yesterday" (19.6)

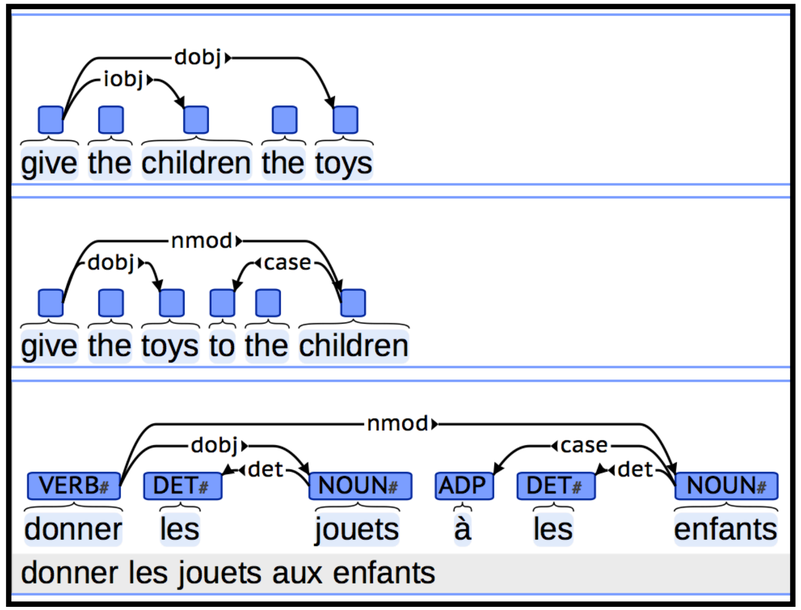
Universal Dependencies

- Dependencies primarily hold between **content words**
  - Function words vary a lot across languages

- **Function words** attach to the most closely related content word, and typically don't have their own dependents

- For coordination, we draw an edge from the first head to the second

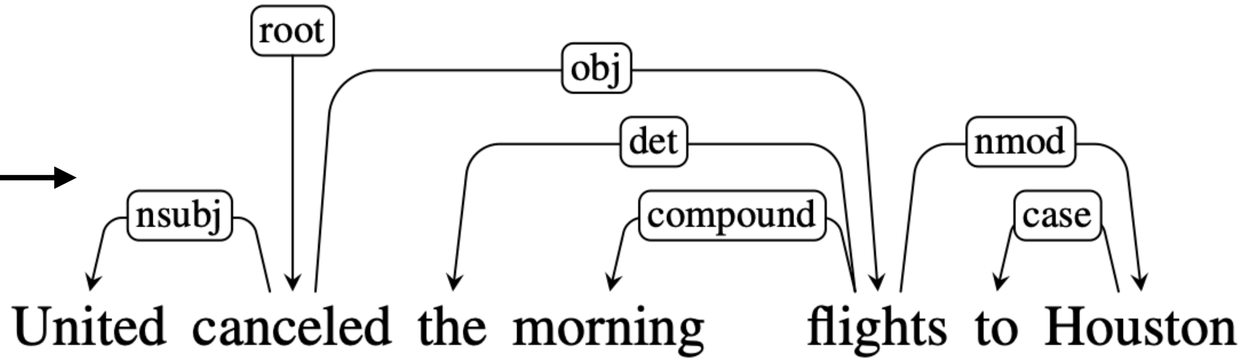




Dependency Parsing

- **Goal**: map from a string to a dependency parse

United canceled the morning flights to Houston →



- Two main approaches:
  - Transition-based parsing
  - Graph-based parsing

Dependency Parsing

- We will focus on transition-based parsers based on **shift-reduce parsing**:
  - Read the sentence left to right, word by word
  - For each word, we can take one of two types of action:
    - **Shift:** read the next word
    - **Reduce:** attach one word to another (i.e., add an arc)
- This algorithm can only produce projective dependency trees.

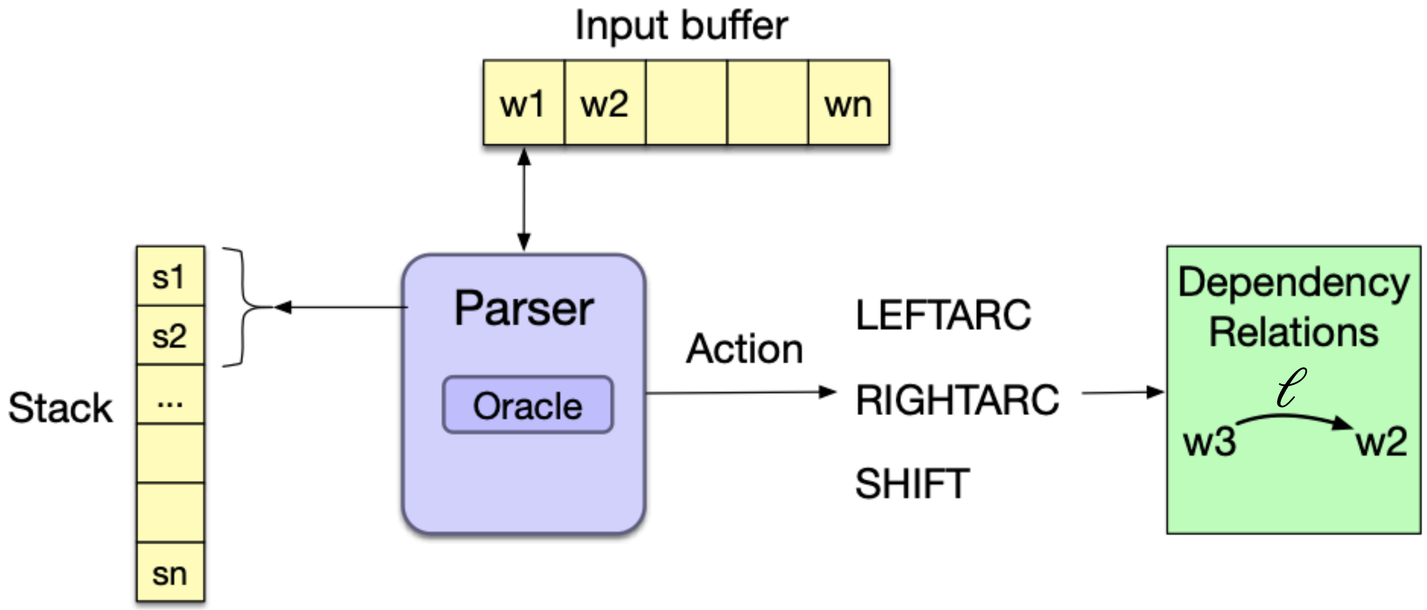
- Shift-reduce parsers used to be less popular because they didn't work very well. But these days, they're based on neural networks; they work well and are *very* fast!

Transition-based Parsing

A shift-reduce parser contains:

- A **stack**: a *stack* of partially processed words $w_i \in T_S$
- A **buffer:** a *list* of input words that haven't yet been processed $w_i \in T_S$
- A **dependency relation set**: $(w_i, \ell, w_j) \in T_S \times L \times T_S$

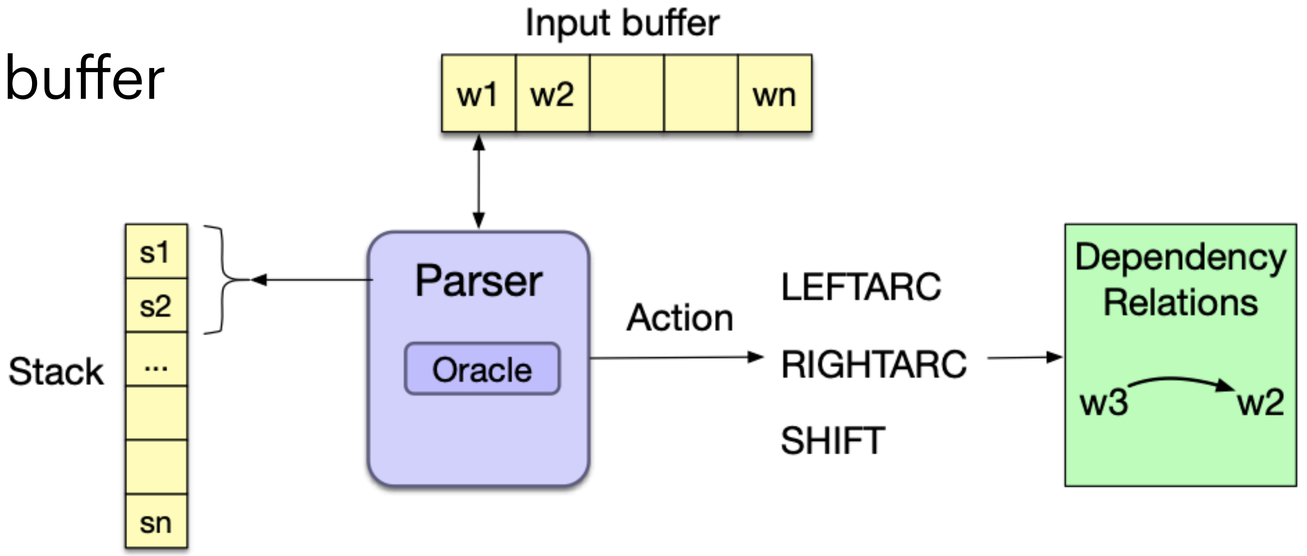
T_S = [the, cat, slept, soundly]



Parser Steps

Start state:

- All words in the sentence are in the buffer
- The stack only contains ROOT
- Dependency relation set is empty



End state:

- Stack and word list are both empty
- Dependency relation set is the final parse

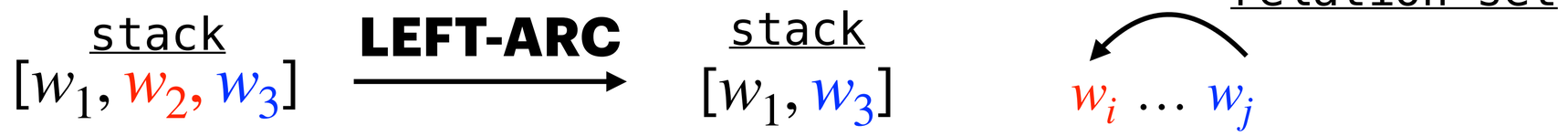
Arc Standard Transition System

The Arc Standard Transition System defines 3 possible transition operators.

1. **SHIFT:** move first word from buffer to the top of the stack:

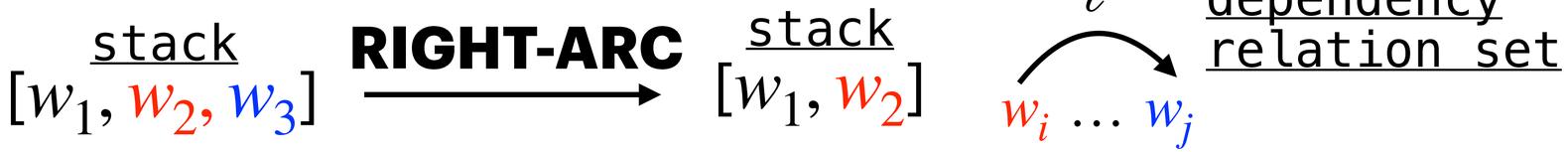
---

2. **LEFT-ARC:** add leftward dependency arc with label ℓ from w_j to w_i.
   Remove w_i (second from top word) from stack.



---

3. **RIGHT-ARC:** add rightward dependency arc with label ℓ from w_i to w_j.
   Remove w_j (top word) from stack.



The Shift-Reduce Algorithm

1. Start in **initial configuration:** $([w_0], [w_1, \ldots, w_n], \{\})$.
   - All we can do is push w_1 onto the stack.

2. We want to end in a **terminal configuration:** $([w_0], [], A)$
   - We have read all input words and have attached all input words, and ended up with dependency tree A.

```
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
```

$O(n)$ time!

(Recall: CKY was $O(n^3 |G|)$).

Parsing Models

- Given a configuration (stack, buffer, relation set), which action should the parser take?

- We need a **parsing model** that assigns a score to each of the possible actions given a configuration.
  - We can learn this model from a dependency treebank

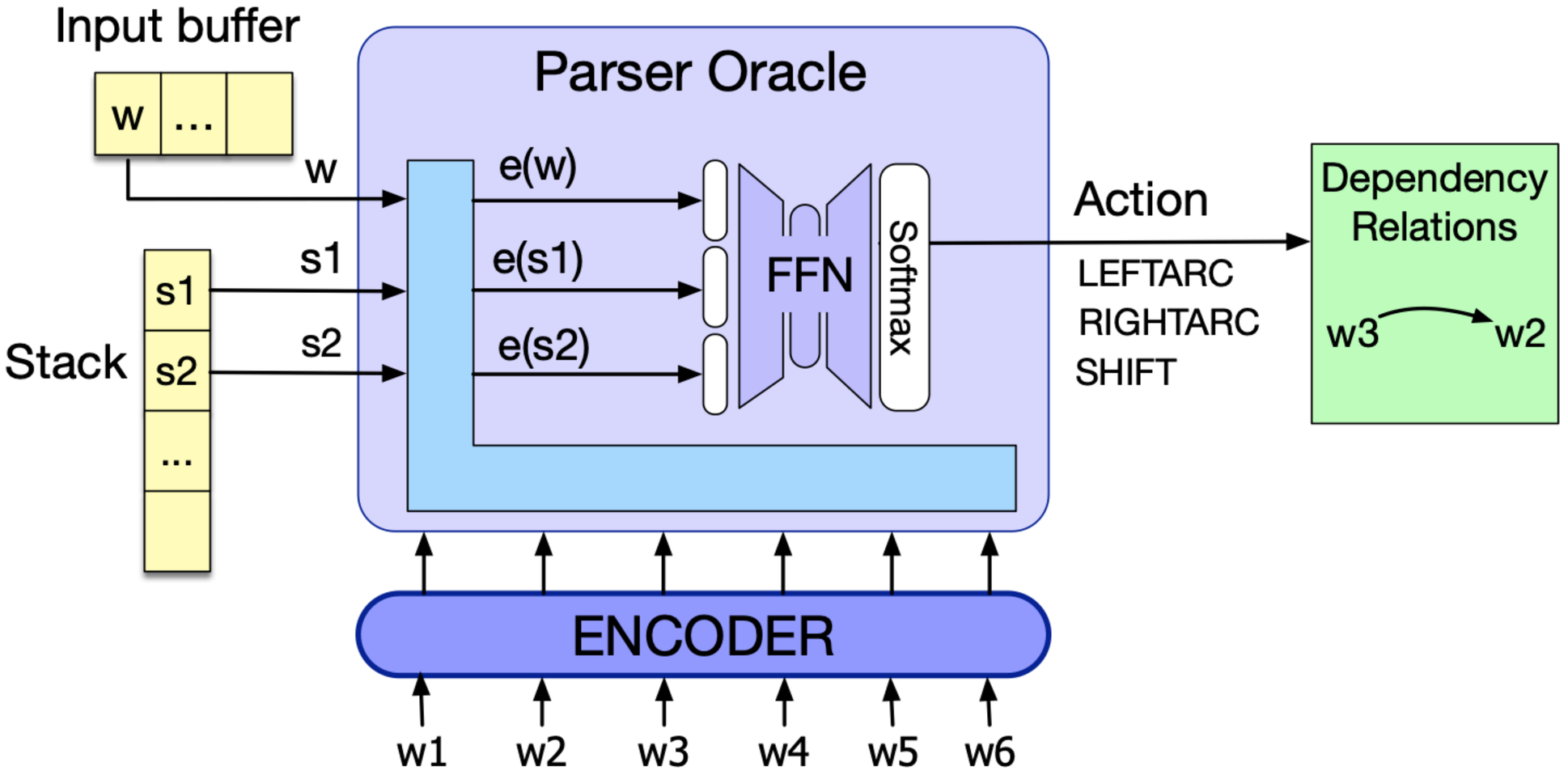
- **Chen & Manning [2014]**: predict the next action with a neural network with one hidden layer
  - Input: parser configurations, each represented as feature vectors
  - Output: softmax over (1+2L) actions (shift + 2 arc actions per label $\ell \in L$)

Training a Parsing Model

- We can train a parsing model using supervised machine learning! This is just a multinomial classification problem.
- To get training data, we'll supply a model with training sentences along with their reference parses from a treebank.

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

A Neural Classifier



Example

Book me the morning flight.

Step	Stack	Word List (buffer)	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Example

Book me the morning flight.

Step	Stack	Word List (buffer)	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2				
3				
4				
5				
6				
7				
8				
9				
10				

Example

Step	Stack	Word List (buffer)	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3				
4				
5				
6				
7				
8				
9				
10				

iobj

Example

Step	Stack	Word List (buffer)	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4				
5				
6				
7				
8				
9				
10				

iobj

Example

Step	Stack	Word List (buffer)	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5				
6				
7				
8				
9				
10				

iobj

Example

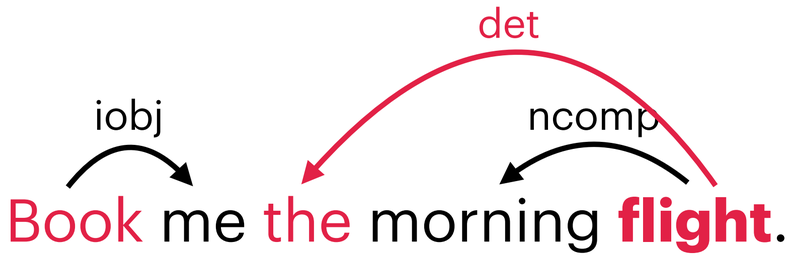
Step	Stack	Word List (buffer)	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6				
7				
8				
9				
10				

iobj

Example

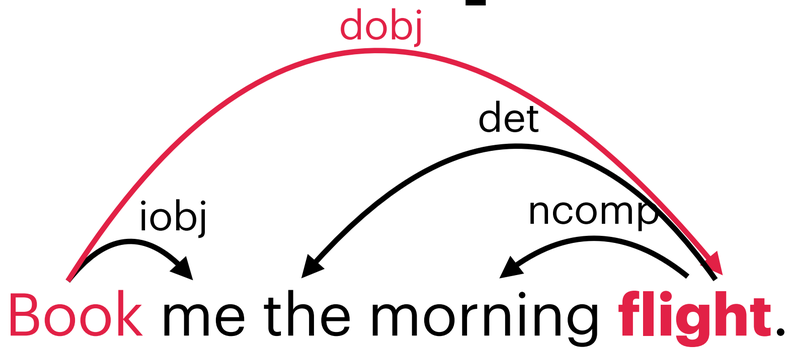
Step	Stack	Word List (buffer)	Action	Relation Added	
0	[root]	[book, me, the, morning, flight]	SHIFT		
1	[root, book]	[me, the, morning, flight]	SHIFT		
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)	iobj
3	[root, book]	[the, morning, flight]	SHIFT		
4	[root, book, the]	[morning, flight]	SHIFT		
5	[root, book, the, morning]	[flight]	SHIFT		
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)	ncomp
7					
8					
9					
10					

Example



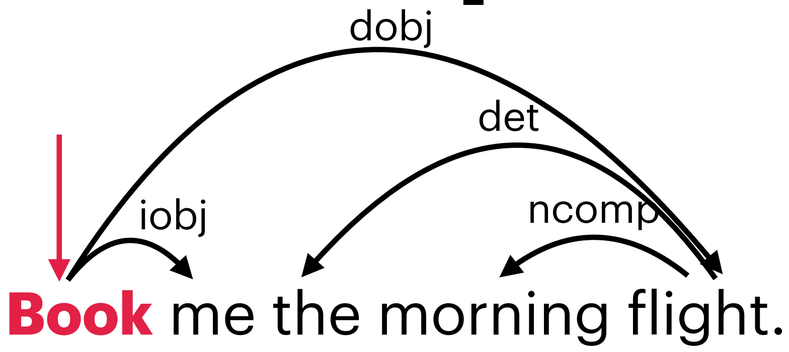
Step	Stack	Word List (buffer)	Action	Relation Added	
0	[root]	[book, me, the, morning, flight]	SHIFT		
1	[root, book]	[me, the, morning, flight]	SHIFT		
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)	iobj
3	[root, book]	[the, morning, flight]	SHIFT		
4	[root, book, the]	[morning, flight]	SHIFT		
5	[root, book, the, morning]	[flight]	SHIFT		
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)	ncomp
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)	det
8					
9					
10					

Example



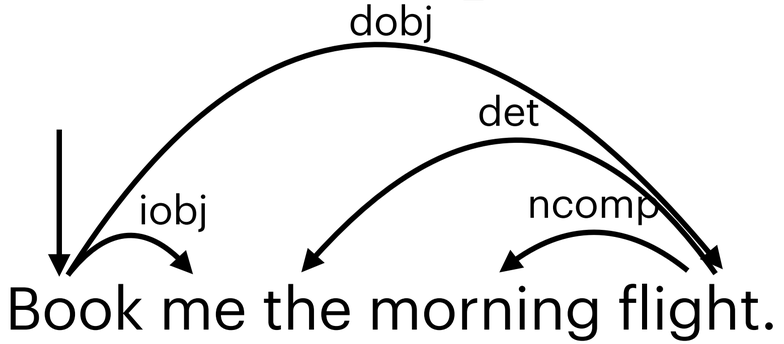
Step	Stack	Word List (buffer)	Action	Relation Added	
0	[root]	[book, me, the, morning, flight]	SHIFT		
1	[root, book]	[me, the, morning, flight]	SHIFT		
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)	iobj
3	[root, book]	[the, morning, flight]	SHIFT		
4	[root, book, the]	[morning, flight]	SHIFT		
5	[root, book, the, morning]	[flight]	SHIFT		
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)	ncomp
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)	det
8	[root, book, flight]	[]	RIGHTARC	(book → flight)	dobj
9					
10					

Example



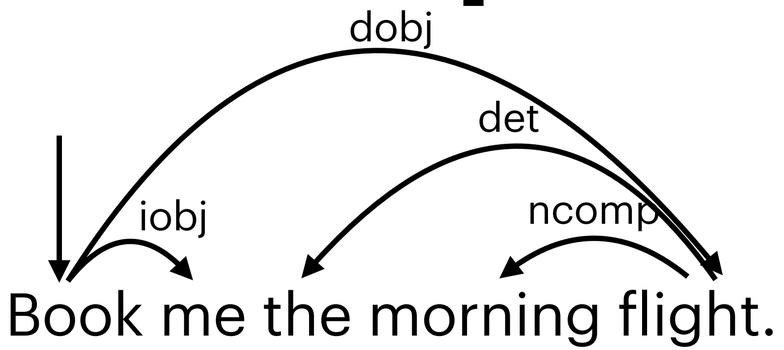
Step	Stack	Word List (buffer)	Action	Relation Added	
0	[root]	[book, me, the, morning, flight]	SHIFT		
1	[root, book]	[me, the, morning, flight]	SHIFT		
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)	iobj
3	[root, book]	[the, morning, flight]	SHIFT		
4	[root, book, the]	[morning, flight]	SHIFT		
5	[root, book, the, morning]	[flight]	SHIFT		
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)	ncomp
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)	det
8	[root, book, flight]	[]	RIGHTARC	(book → flight)	dobj
9	[root, book]	[]	RIGHTARC	(root → book)	
10					

Example



Step	Stack	Word List (buffer)	Action	Relation Added	
0	[root]	[book, me, the, morning, flight]	SHIFT		
1	[root, book]	[me, the, morning, flight]	SHIFT		
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)	iobj
3	[root, book]	[the, morning, flight]	SHIFT		
4	[root, book, the]	[morning, flight]	SHIFT		
5	[root, book, the, morning]	[flight]	SHIFT		
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)	ncomp
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)	det
8	[root, book, flight]	[]	RIGHTARC	(book → flight)	dobj
9	[root, book]	[]	RIGHTARC	(root → book)	
10	[root]	[]	Done		

Example



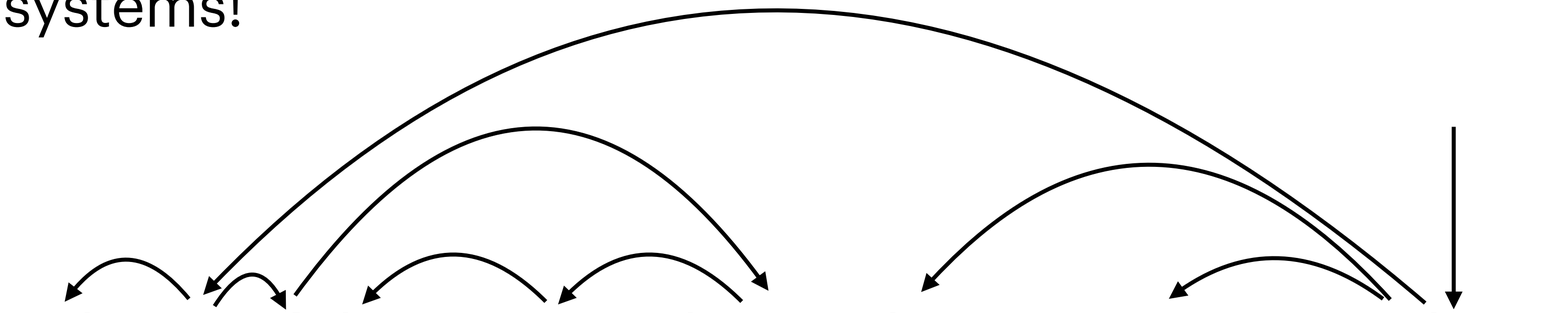
You should only attach a word to its head when that word has no further dependents.

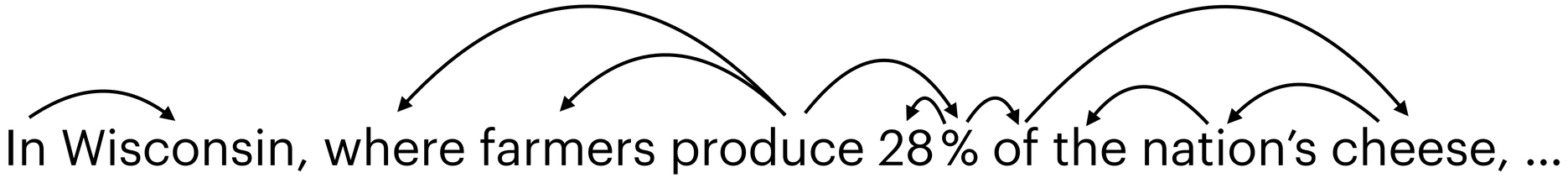
This is why this algorithm can't handle non-projective parses.

Step	Stack	Word List (buffer)	Action	Relation Added	
0	[root]	[book, me, the, morning, flight]	SHIFT		
1	[root, book]	[me, the, morning, flight]	SHIFT		
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)	iobj
3	[root, book]	[the, morning, flight]	SHIFT		
4	[root, book, the]	[morning, flight]	SHIFT		
5	[root, book, the, morning]	[flight]	SHIFT		
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)	ncomp
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)	det
8	[root, book, flight]	[]	RIGHTARC	(book → flight)	dobj
9	[root, book]	[]	RIGHTARC	(root → book)	
10	[root]	[]	Done		

Applying Dependencies for NLP

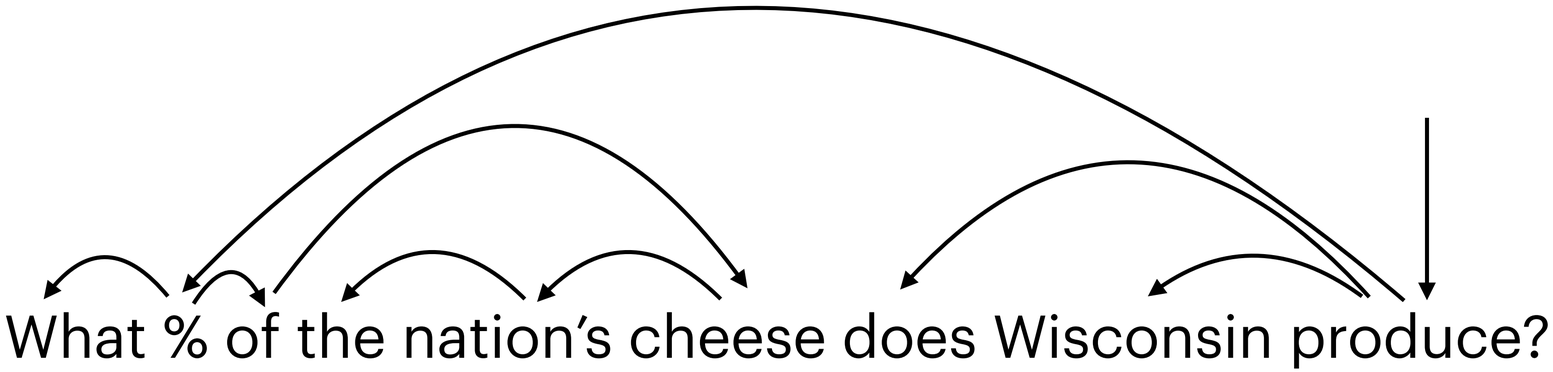
- Dependency parses are used in many real-world applications, like question answering systems!





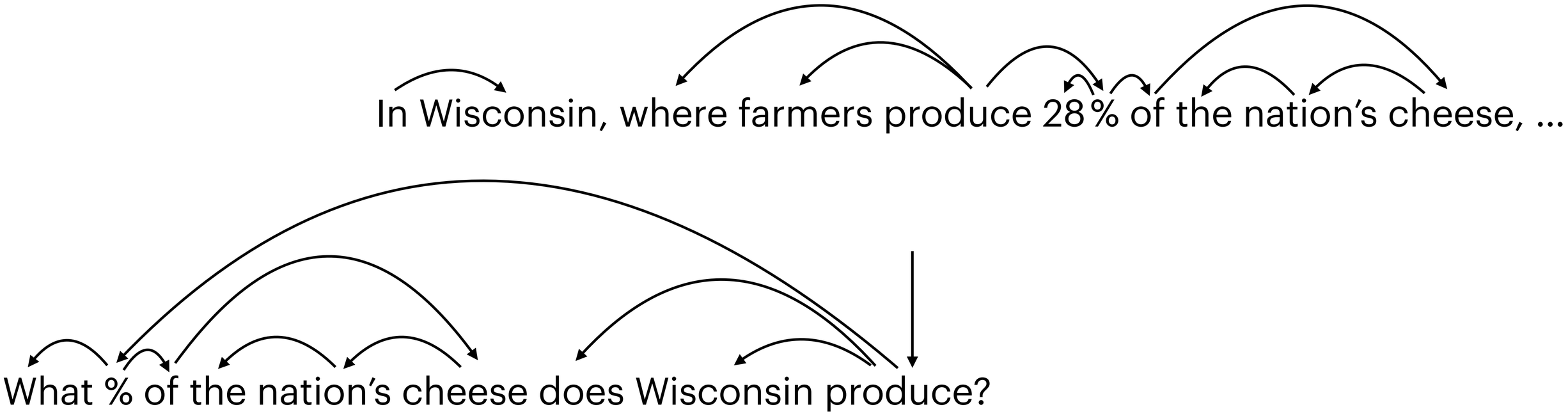
Applying Dependencies for NLP

- Dependency parses are used in many real-world applications, like question answering systems!
  - *“produce”* and *“%”* are six words apart.
  - But in the parse, they’re adjacent!



Applying Dependencies for NLP

- Dependency parses are used in many real-world applications, like question answering systems!
    - We can look for the dependent of “produce”, then the dependent of “%”. This will be our answer.



Converting Constituencies to Dependencies

Every phrase has a **head word**. It is the head of all other words of that phrase in the dependency graph.

VP → Verb NP	**Head**: Verb	**Dependent**: NP
NP → Det Noun	**Head**: Noun	**Dependent**: Det

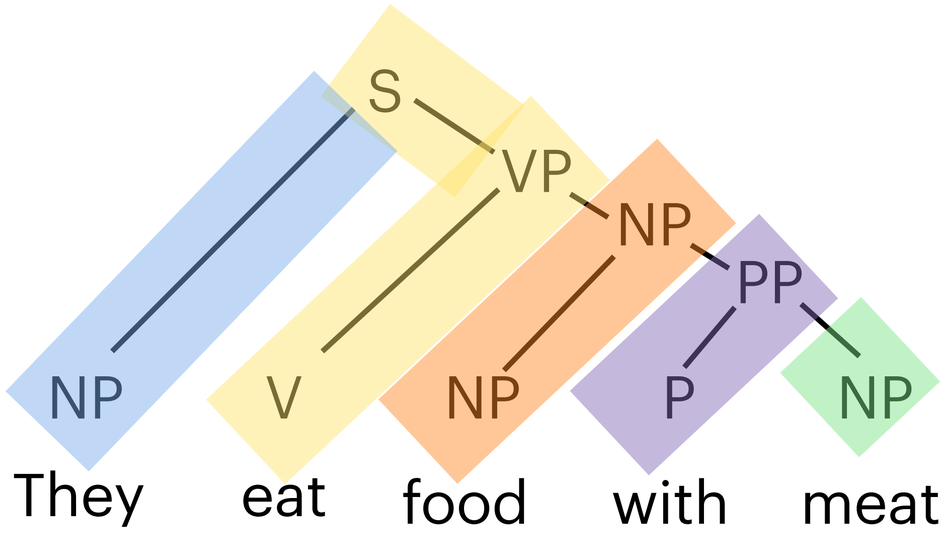
For every non-terminal in a constituency tree, you can choose one of its children to be its “head”.

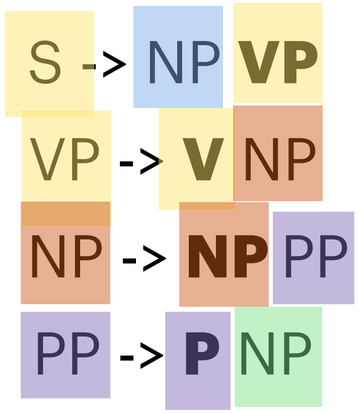
Every non-terminal type has a different “head rule”.

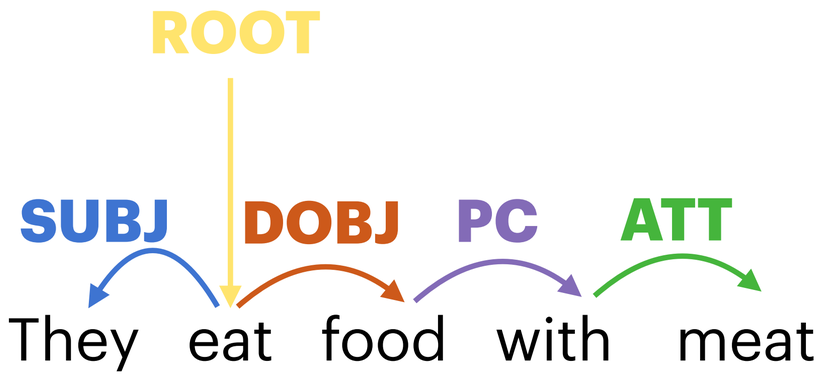
NP: If parent is NP, search right-to-left for first child that’s a noun.

Else, search left-to-right for first child that’s an NP.

Converting Constituencies to Dependencies







To convert CFG into dependency parse:

Start at the root of the tree (S).

Follow **head path** to the **head word** of the sentence (“eats”).

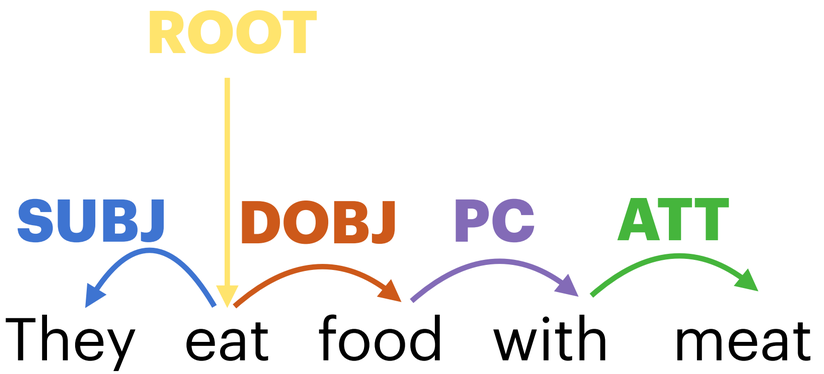
Add a ROOT dependency to the word.

For all other non-terminals, follow their head paths to get their head words; add the corresponding dependencies.

Converting Constituencies to Dependencies

Notice that we're using "head" in two ways:

1. The source of an arc (head/governor of a dependency)
2. The main (head) word of a constituent

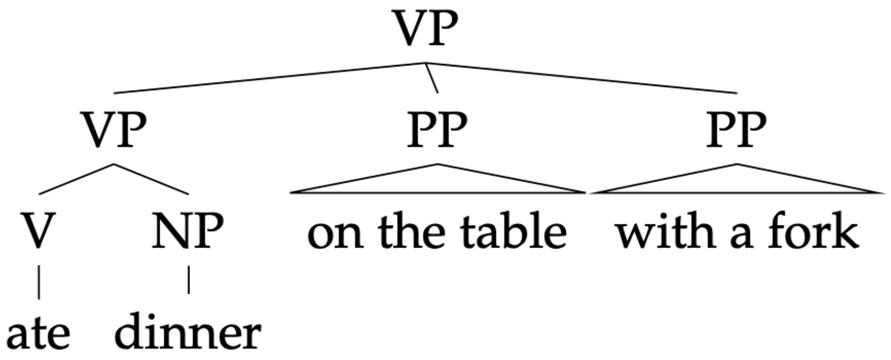


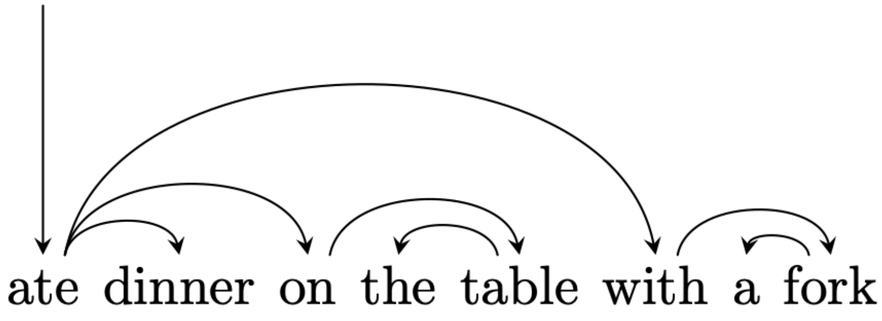
For example, "eat" is the *head* of the DOBJ *dependency*.

But the *head word* of the object NP is "food".

CFGs vs. Dependency Grammars

- As you may have noticed, dependencies tend to be less specific than constituencies.





Better Search Methods

- Shift-reduce parsing is a **greedy** algorithm.
  - It's fast, but easy to make a mistake. These errors will propagate to future steps.

- Can use **beam search** instead (still greedy, but better).
  - Hold on to top k states (partial parses), according to some scoring method
  - Take top item from final beam at the end as the parse