CS-AD 220 – Spring 2016

Natural Language Processing

Session 18: 5-Apr-16

Prof. Nizar Habash

Some slides are adapted from talks by Jurafsky and Martin; Nivre; and Mooney.

NYUAD Course CS-AD 220 – Spring 2016

Natural Language Processing

Assignment #3 : POS Tagging and Parsing

Assigned Mar 31, 2016 / Due Apr 17, 2016 (11:59pm)

I. Grading & Submission

This assignment is about the development of a dependency parser and a part-of-speech (POS) tagger for English. The assignment accounts for 15% of the full grade. It consists of five exercises. **There is also a bonus exercise that can count for up to 5% of the full grade**. The additional exercise consists of a parsing competition on an unseen test set. Participation earns 2%. The first, second and third ranked systems earn additional 3%, 2% and 1%, respectively.

Assignment #3 posted on NYU Classes

START EARLY!

DEADLINE PUSHED FORWARD TO APR 17

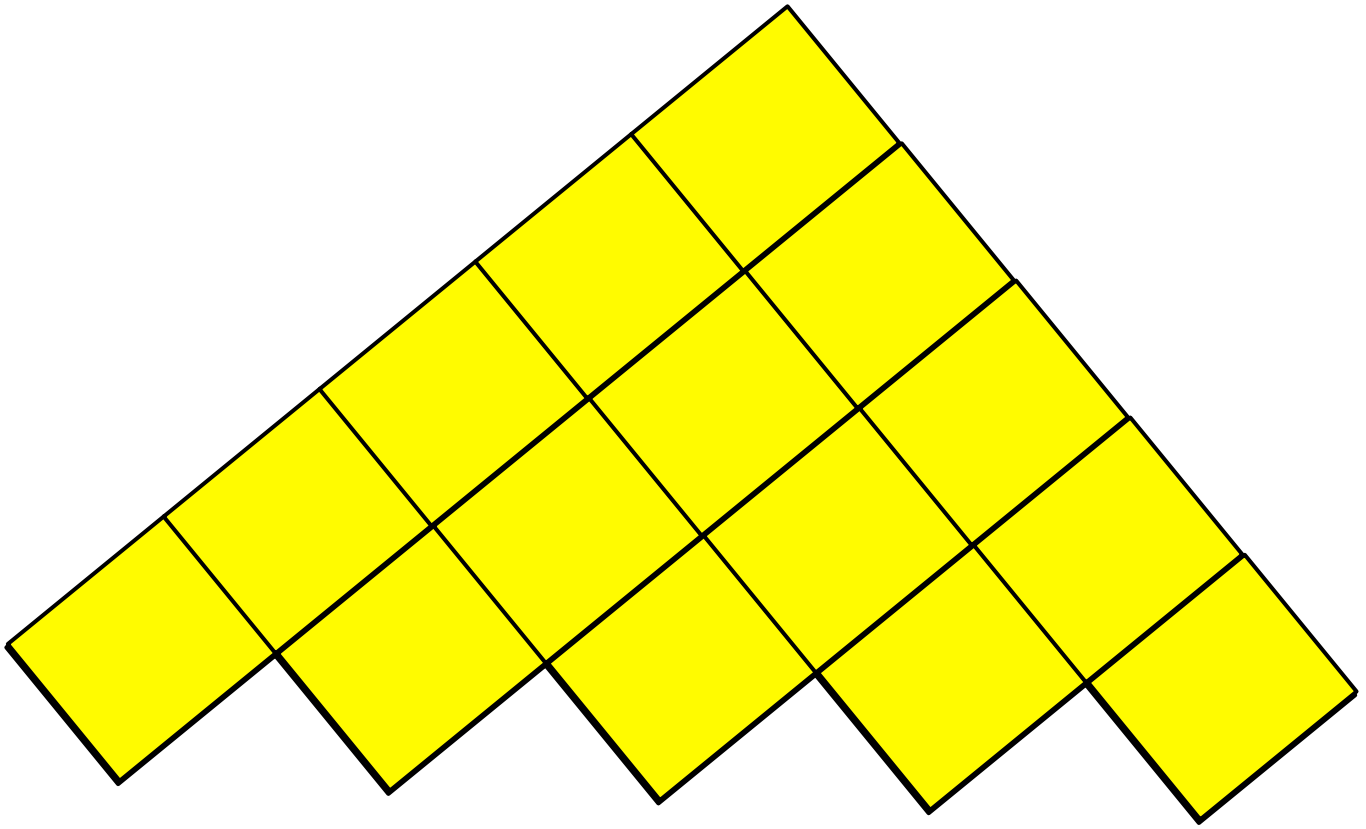
Grades in Class So Far

- Grades are posted on NYU Classes

Types of Syntactic Analyses

- **Phrase Structure Parsing**
  - **aka constituency parsing**
  - **CKY Parser**
- Dependency Parsing
  - MaltParser
- Chunking
  - Base-phrase chunking

Revisiting CKY Parser

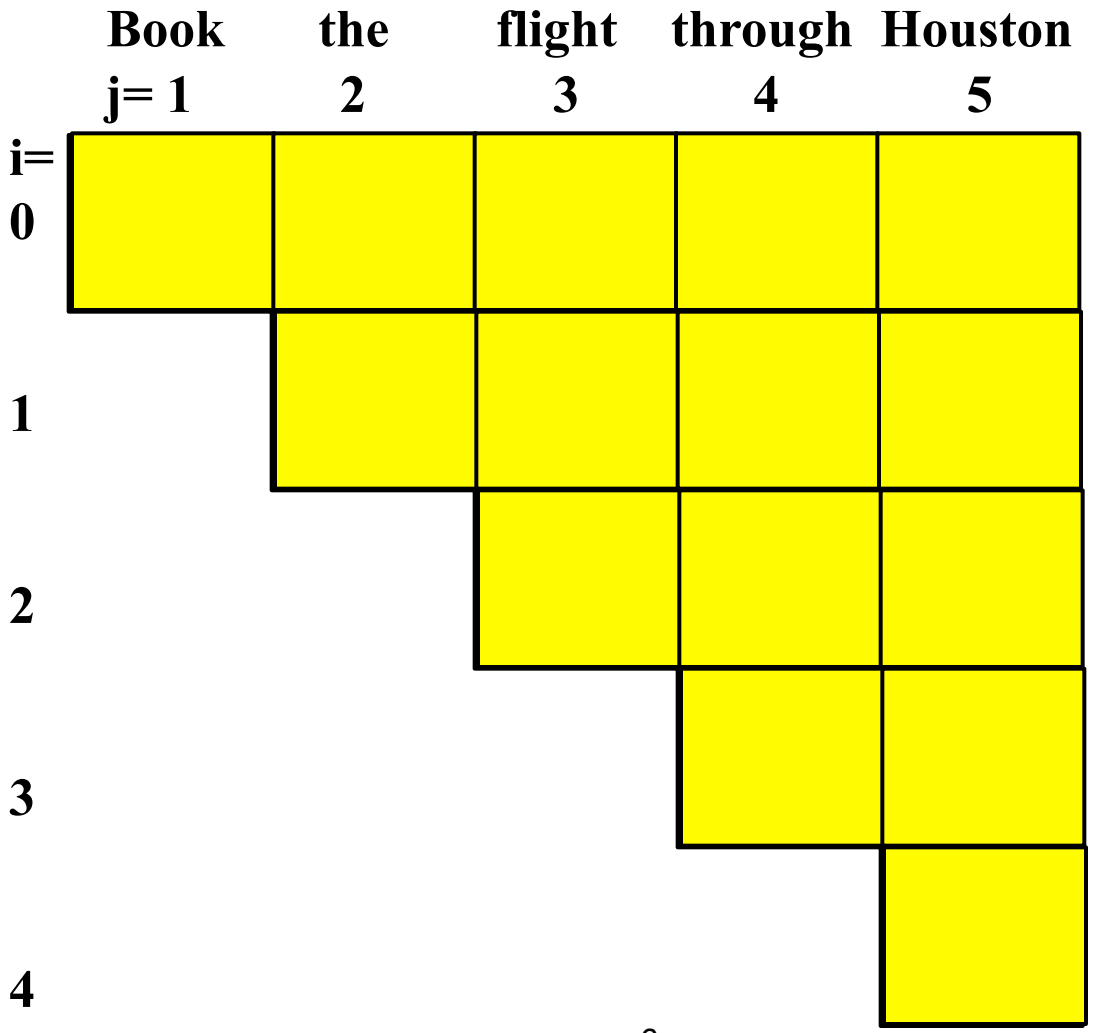


Book the flight through Houston

CKY Parser

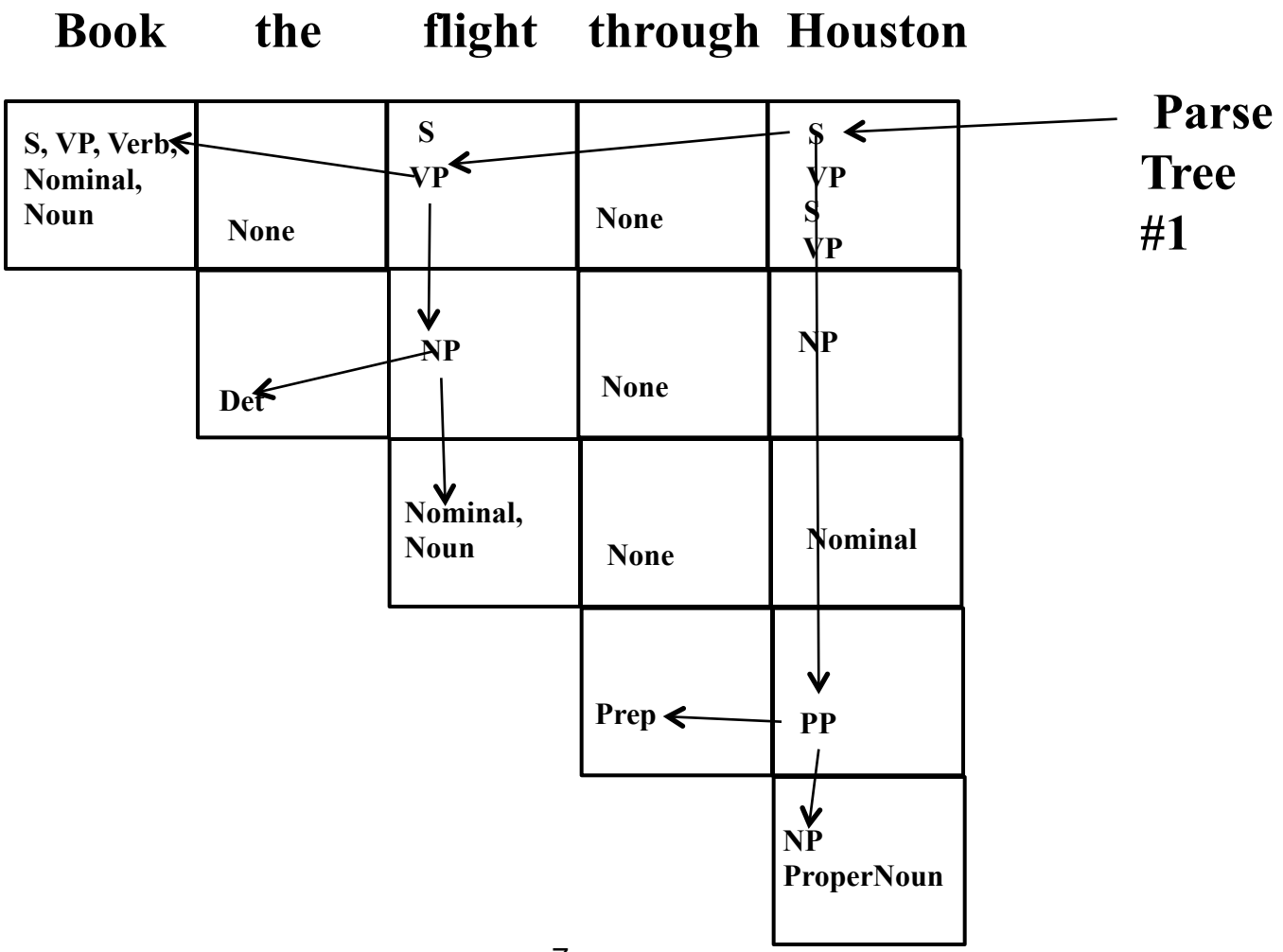
```
function CKY-PARSE(words, grammar) returns table

for j ← from 1 to LENGTH(words) do
    table[j − 1, j] ← {A | A → words[j] ∈ grammar}
    for i ← from j − 2 downto 0 do
        for k ← i + 1 to j − 1 do
            table[i,j] ← table[i,j] ∪
                {A | A → BC ∈ grammar,
                     B ∈ table[i,k],
                     C ∈ table[k, j]}
```

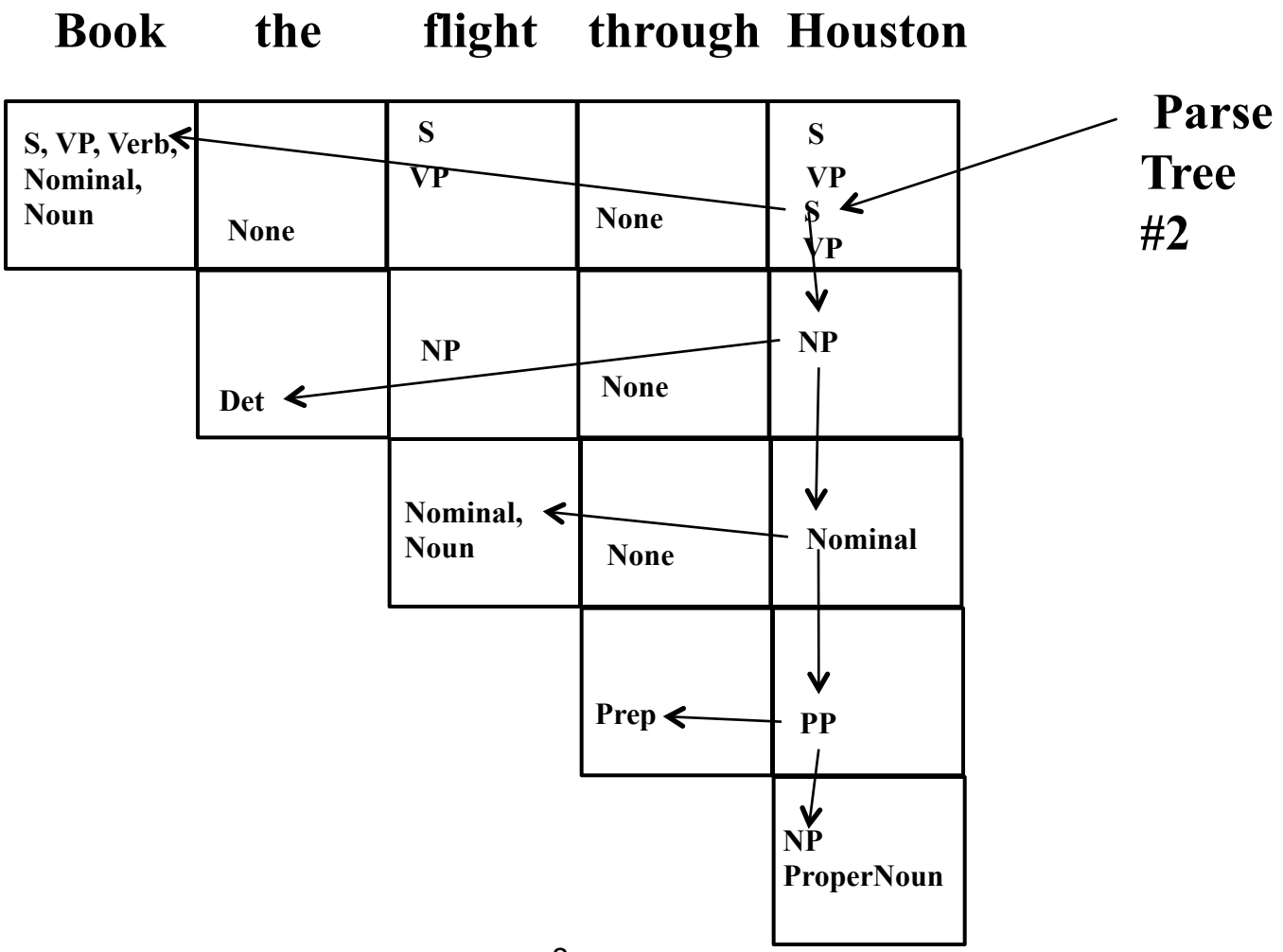


Cell[*i,j*] contains all constituents (non-terminals) covering words *i* +1 through *j*

CKY Parser


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
None
S
VP
None
S
VP
S
VP
Parse Tree #1
Det
NP
None
NP
Nominal, Noun
None
Nominal
Prep
PP
NP
ProperNoun

CKY Parser


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
None
S
VP
None
S
VP
S
VP
Parse Tree #2
NP
Det
NP
None
NP
Nominal, Noun
None
Nominal
Prep
PP
NP
ProperNoun

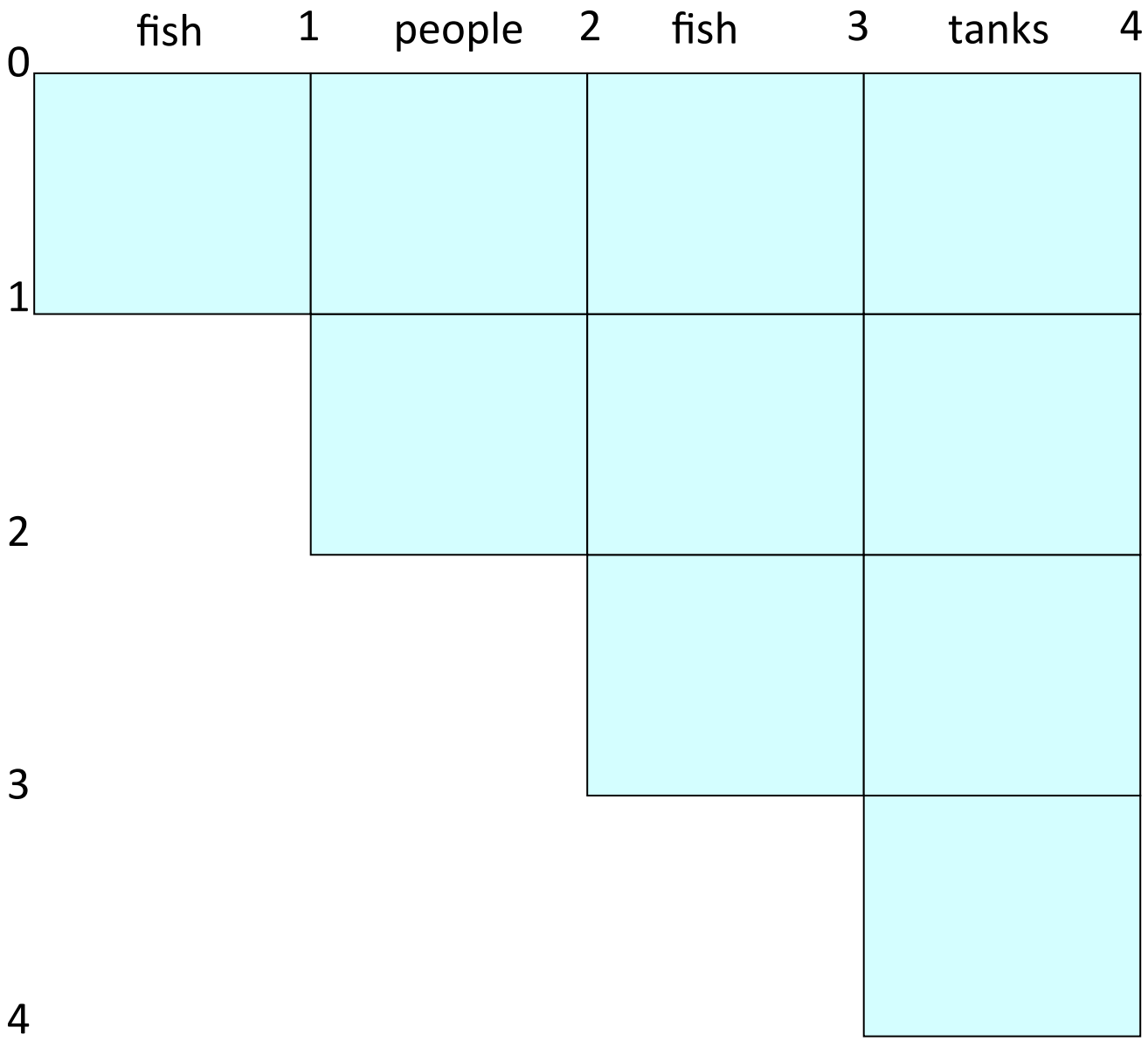
Syntactic Ambiguity

- The algorithm we discussed just produces all possible parse trees.
- Does not address the important issue of ambiguity resolution.

Probabilistic CFG (PCFG)

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0


fish
1
people
2
fish
3
tanks
4
0
1
2
3
4

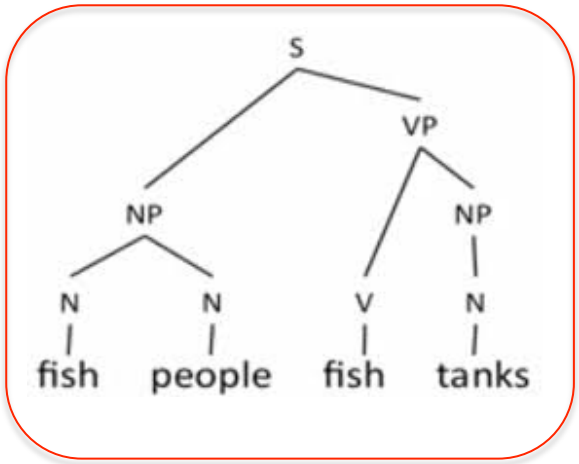
S → NP VP	0.9	
S → VP	0.1	
VP → V NP	0.5	
VP → V	0.1	
VP → V @VP_V	0.3	
VP → V PP	0.1	
@VP_V → NP PP		1.0
NP → NP NP	0.1	
NP → NP PP	0.2	
NP → N	0.7	
PP → P NP	1.0	
N → *people*	0.5	
N → *fish*	0.2	
N → *tanks*		0.2
N → *rods*	0.1	
V → *people*	0.1	
V → *fish*	0.6	
V → *tanks*	0.3	
P → *with*	1.0	

	fish	1	people	2	fish	3	tanks	4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	1	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	2	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	3	NP → NP NP 0.0000009604 VP → V NP 0.00002058 S → NP VP 0.00018522	4
1		2	N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	3	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	4	NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323	
2		3		4	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006		NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042	
3		4					N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003	
4								

S → NP VP	0.9	
S → VP	0.1	
VP → V NP	0.5	
VP → V	0.1	
VP → V @VP_V	0.3	
VP → V PP	0.1	
@VP_V → NP PP		1.0
NP → NP NP	0.1	
NP → NP PP	0.2	
NP → N	0.7	
PP → P NP	1.0	

N → *people*	0.5	
N → *fish*	0.2	
N → *tanks*		0.2
N → *rods*	0.1	
V → *people*	0.1	
V → *fish*	0.6	
V → *tanks*	0.3	
P → *with*	1.0	

	fish	1	people	2	fish	3	tanks	4
0								
1	N → fish 0.2 V → fish 0.6 **NP → N 0.14** VP → V 0.06 S → VP 0.006		**NP → NP NP 0.0049** VP → V NP 0.105 S → VP 0.0105		NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882		NP → NP NP 0.0000009604 VP → V NP 0.00002058 **S → NP VP 0.00018522**	
2			N → people 0.5 V → people 0.1 **NP → N 0.35** VP → V 0.01 S → VP 0.001		NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189		NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323	
3					N → fish 0.2 **V → fish 0.6** NP → N 0.14 VP → V 0.06 S → VP 0.006		NP → NP NP 0.00196 **VP → V NP 0.042** S → VP 0.0042	
4							N → tanks 0.2 V → tanks 0.1 **NP → N 0.14** VP → V 0.03 S → VP 0.003	



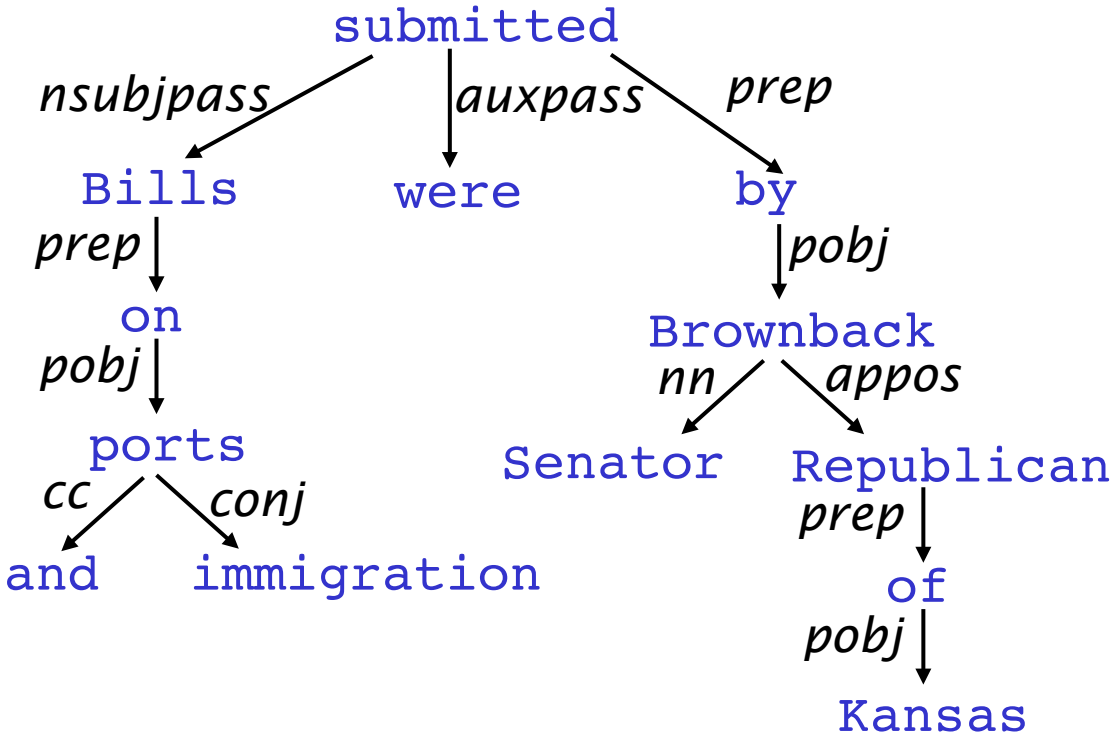
Types of Syntactic Analyses

- Phrase Structure Parsing
  - aka constituency parsing
  - CKY Parser
- **Dependency Parsing**
  - **MaltParser**
- Chunking
  - Base-phrase chunking

Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc.)



Dependency Relations

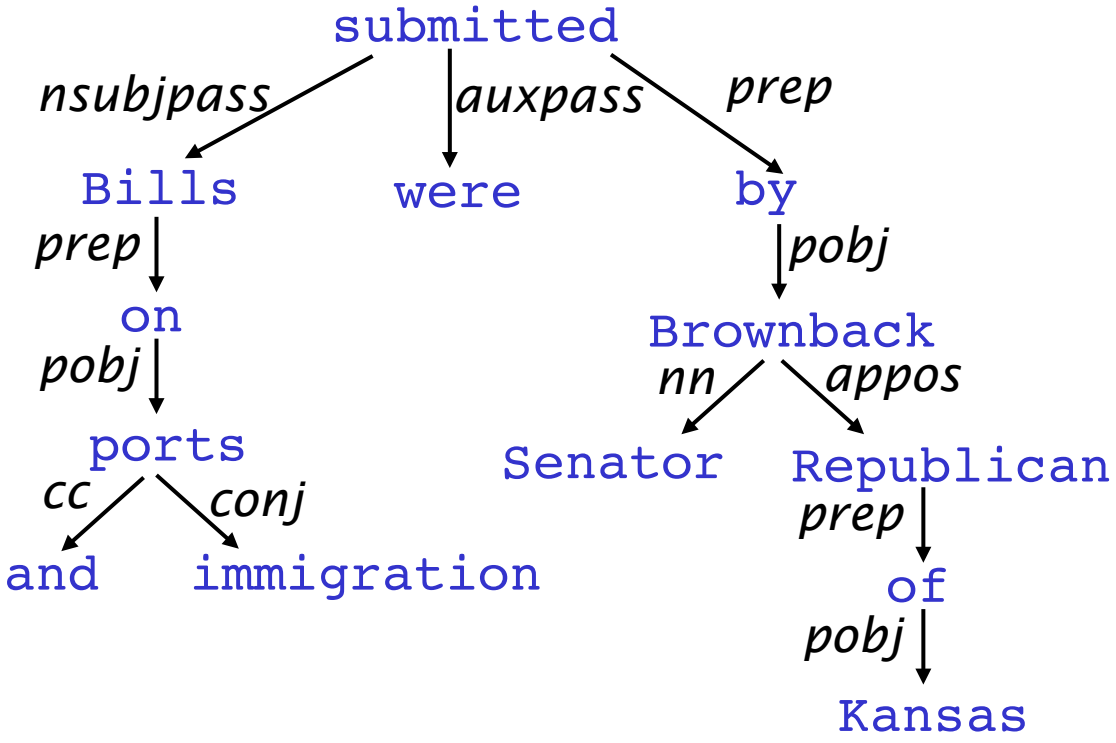
Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

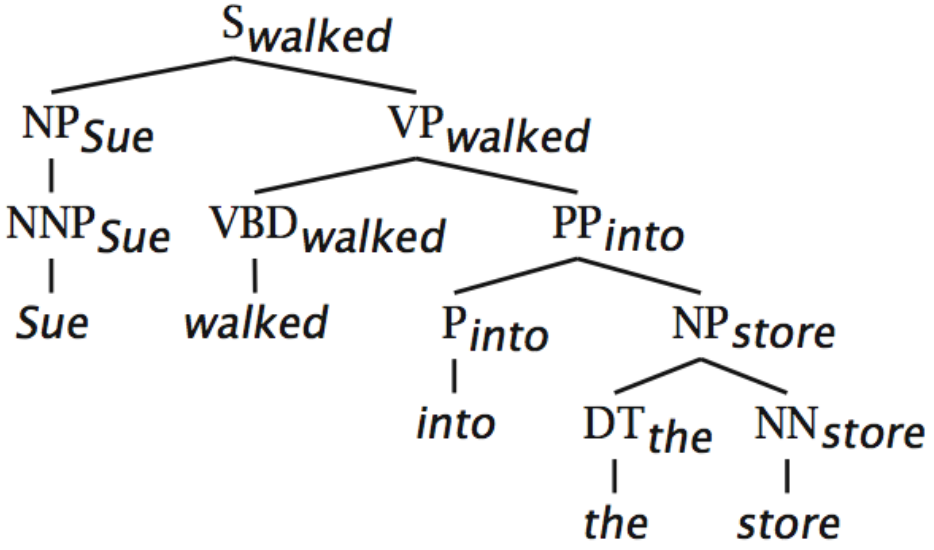
The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)



Relation between phrase structure and dependency structure

- A dependency grammar has a notion of a head. Officially, CFGs don't.
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, …) do, via hand-written phrasal "head rules":
  - The head of a Noun Phrase is a noun/number/adj/…
  - The head of a Verb Phrase is a verb/modal/….
- The head rules can be used to extract a dependency parse from a CFG parse



Methods of Dependency Parsing

1. Dynamic programming (like in the CKY algorithm)

   You can do it similarly to lexicalized PCFG parsing: an $O(n^5)$ algorithm

   Eisner (1996) gives a clever algorithm that reduces the complexity to $O(n^3)$

2. Graph algorithms

   You create a Maximum Spanning Tree for a sentence

   McDonald et al.'s (2005) MSTParser scores dependencies independently using a ML classifier (he uses MIRA, for online learning, but it could be MaxEnt)

3. Constraint Satisfaction

   Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.

4. "Deterministic parsing"

   Greedy choice of attachments guided by machine learning classifiers

   MaltParser (Nivre et al. 2008) – discussed next [Assignment #3 uses this approach]

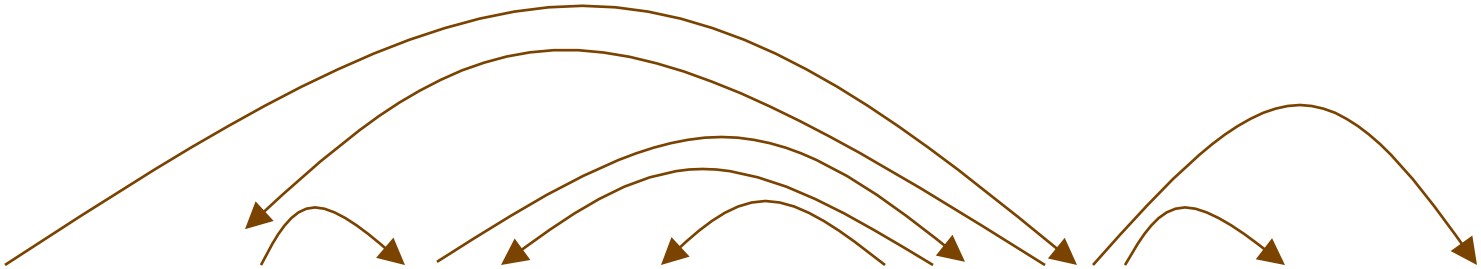
Dependency Conditioning Preferences

What are the sources of information for dependency parsing?

1. Bilexical affinities [issues → the] is plausible
2. Dependency distance mostly with nearby words
3. Intervening material

   Dependencies rarely span intervening verbs or punctuation
4. Valency of heads

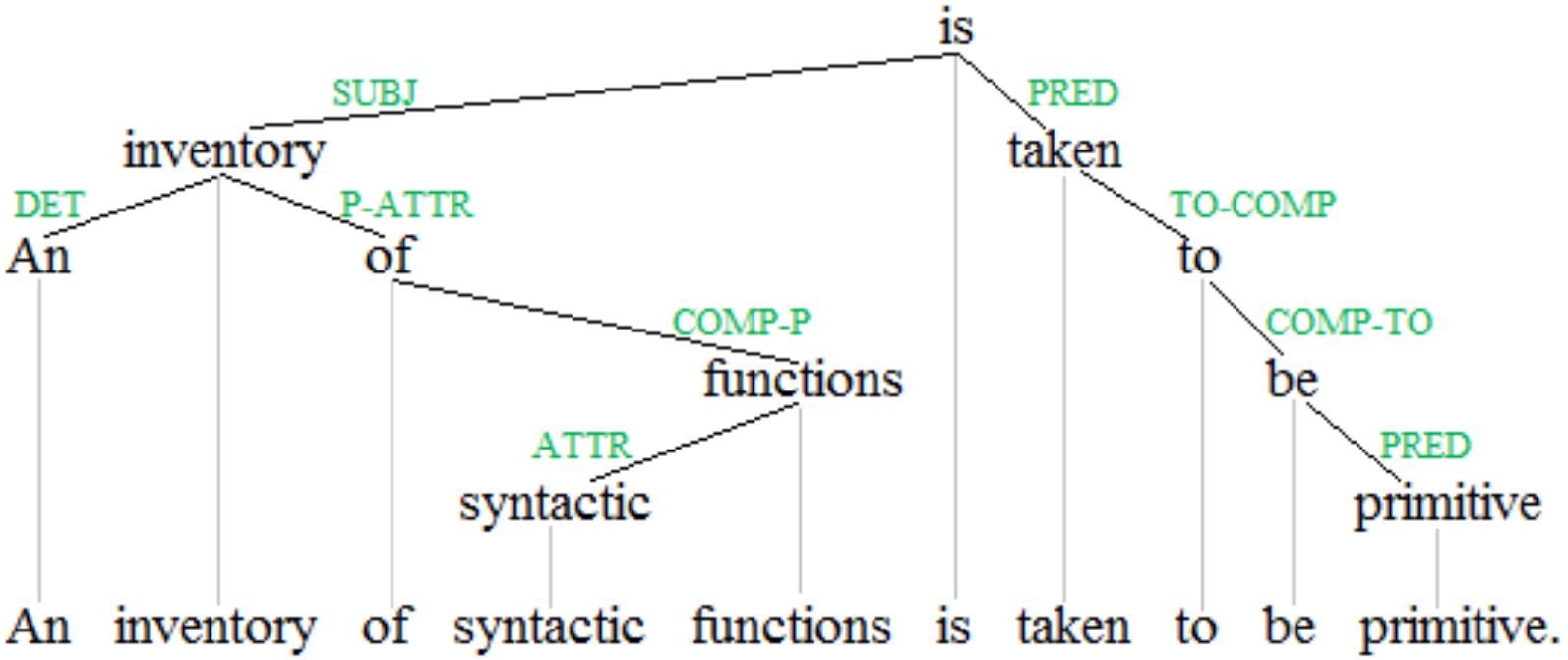
   How many dependents on which side are usual for a head?



Dependency Parse

An inventory of syntactic functions is taken to be primitive.

Dependency Parse



Dependency Parse

CONLL Format

ID	Word		CPOS	FPOS		ParentID	Relation		
1	The	_	DET	DT	_	4	det	_	_
2	luxury	_	NOUN	NN	_	4	compmod	_	_
3	auto	_	NOUN	NN	_	4	compmod	_	_
4	maker	_	NOUN	NN	_	7	nsubj	_	_
5	last	_	ADJ	JJ	_	6	amod	_	_
6	year	_	NOUN	NN	_	7	nmod	_	_
7	sold	_	VERB	VBD	_	0	ROOT	_	_
8	1,214	_	NUM	CD	_	9	num	_	_
9	cars	_	NOUN	NNS	_	7	dobj	_	_
10	in	_	ADP	IN	_	7	adpmod	_	_
11	the	_	DET	DT	_	12	det	_	_
12	U.S.	_	NOUN	NNP	_	10	adpobj	_	_

MaltParser

[Nivre et al. 2008]

- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom up actions
  - "shift", "reduce", "left-arc", "right-arc"
- The parser has:
  - a stack σ, written with top to the right
    - which starts with the ROOT symbol
  - a buffer β, written with top to the left
    - which starts with the input sentence
  - a set of dependency arcs A
    - which starts off empty
  - a set of actions

Transition-based Dependency Parser

Actions	**Stack σ**	**Buffer β**	**Arcs A** *r(head,dep)*	***Pre-conditions***
<start>	[ROOT]	$w_1, \ldots, w_n$	∅	none
Shift	σ	$w_i \mid \beta$	A	
➔	$\sigma \mid w_i$	β	A	
Reduce	$\sigma \mid w_i$	β	A	$r'(w_k, w_i) \in A$
➔	σ	β	A	
Left-Arc$_r$	$\sigma \mid w_i$	$w_j \mid \beta$	A	$r'(w_k, w_i) \notin A$, $w_i \neq$ ROOT
➔	σ	$w_j \mid \beta$	$A \cup \{r(w_j,w_i)\}$	e.g., children|-| want
Right-Arc$_r$	$\sigma \mid w_i$	$w_j \mid \beta$	A	
➔	$\sigma \mid w_i \mid w_j$	β	$A \cup \{r(w_i,w_j)\}$	e.g., want |-| toys
<finish>		∅		

This is the common "arc-eager" variant: a head can immediately take a right dependent, before *its* dependents are found

Example

Actions	Stack σ	Buffer β	Arcs A	*Pre-conditions*
Shift	σ	w_i|β	A	
➔	σ|w_i	β	A	
Reduce	σ|w_i	β	A	$r'(w_k, w_i) \in A$
➔	σ	β	A	
Left-Arc$_r$	σ | w_i	w_j|β	A	$r'(w_k, w_i) \notin A$, $w_i \neq$ ROOT
➔	σ	w_j|β	$A \cup \{r(w_j,w_i)\}$	e.g., children|-| want
Right-Arc$_r$	σ|w_i	w_j|β	A	
➔	σ|w_i|w_j	β	$A \cup \{r(w_i,w_j)\}$	e.g., want |-| toys

Happy children like to play with their friends .

	[ROOT]	[Happy, children, …]	∅
Shift	[ROOT, Happy]	[children, like, …]	∅
LA$_{amod}$	[ROOT]	[children, like, …]	{amod(children, happy)} = A_1
Shift	[ROOT, children]	[like, to, …]	A_1
LA$_{nsubj}$	[ROOT]	[like, to, …]	$A_1 \cup$ {nsubj(like, children)} = A_2
RA$_{root}$	[ROOT, like]	[to, play, …]	$A_2 \cup$ {root(ROOT, like) = A_3
Shift	[ROOT, like, to]	[play, with, …]	A_3
LA$_{aux}$	[ROOT, like]	[play, with, …]	$A_3 \cup$ {aux(play, to) = A_4
RA$_{xcomp}$	[ROOT, like, play]	[with their, …]	$A_4 \cup$ {xcomp(like, play) = A_5

Example

Actions	Stack σ	Buffer β	Arcs A	*Pre-conditions*
Shift	σ	$w_i \mid \beta$	A	
➔	$\sigma \mid w_i$	β	A	
Reduce	$\sigma \mid w_i$	β	A	$r'(w_k, w_i) \in A$
➔	σ	β	A	
Left-Arc$_r$	$\sigma \mid w_i$	$w_j \mid \beta$	A	$r'(w_k, w_i) \notin A$, $w_i \neq$ ROOT
➔	σ	$w_j \mid \beta$	$A \cup \{r(w_j, w_i)\}$	e.g., children|-| want
Right-Arc$_r$	$\sigma \mid w_i$	$w_j \mid \beta$	A	
➔	$\sigma \mid w_i \mid w_j$	β	$A \cup \{r(w_i, w_j)\}$	e.g., want |-| toys

Happy children like to play with their friends .

RA$_{xcomp}$	[ROOT, like, play]	[with their, …]	$A_4 \cup \{xcomp(like, play) = A_5$
RA$_{prep}$	[ROOT, like, play, with]	[their, friends, …]	$A_5 \cup \{prep(play, with) = A_6$
Shift	[ROOT, like, play, with, their]	[friends, .]	A_6
LA$_{poss}$	[ROOT, like, play, with]	[friends, .]	$A_6 \cup \{poss(friends, their) = A_7$
RA$_{pobj}$	[ROOT, like, play, with, friends]	[.]	$A_7 \cup \{pobj(with, friends) = A_8$
Reduce	[ROOT, like, play, with]	[.]	A_8
Reduce	[ROOT, like, play]	[.]	A_8
Reduce	[ROOT, like]	[.]	A_8
RA$_{punc}$	[ROOT, like, .]	[]	$A_8 \cup \{punc(like, .) = A_9$

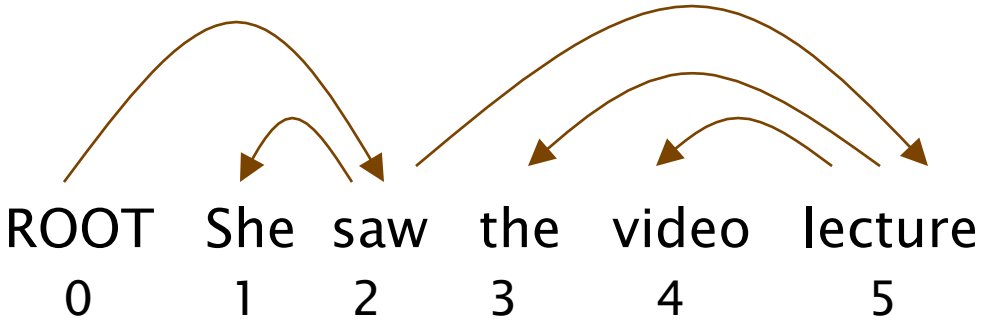
Terminate as soon as the buffer is empty. Dependencies = A_9

MaltParser

[Nivre et al. 2008]

- We have left to explain how we choose the next action
- Each action is predicted by a discriminative classifier (e.g. Support Vector Machine, maxent classifier, etc.) over each legal move
  - Max of 4 untyped choices, max of |R| × 2 + 2 when typed
  - Features: top of stack word, POS; first in buffer word, POS; etc.
- There is NO search (in the simplest and usual form)
- The model's accuracy is *slightly* below the best LPCFGs (Lexicalized Probabilistic CFGs) when evaluated on dependencies, but…
- It provides close to state-of-the-art parsing performance
- It provides VERY fast linear time parsing

Evaluation: Dependency Accuracy



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

UAS =

LAS =

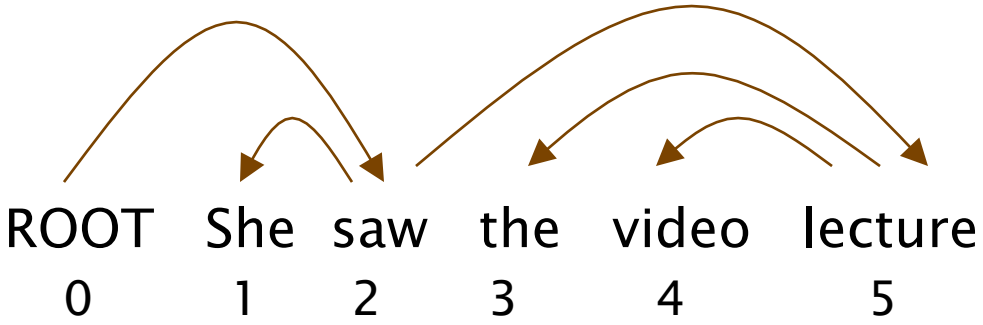
Gold			
1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	dobj

Parsed			
1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

UAS = Unlabeled Attachment Score

LAS = Labeled Attachment Score

Evaluation: Dependency Accuracy



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

UAS = 4 / 5 = 80%
LAS = 2 / 5 = 40%

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	dobj

Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

UAS = Unlabeled Attachment Score
LAS = Labeled Attachment Score

Next Time

- Read J+M Chap 25 (intro up to 25.5)
- Come with questions about Assignment #3