Natural Language Processing

Info 159/259

Lecture 14: Syntactic Parsing (March 6, 2024)

Many slides & instruction ideas borrowed from:

David Bamman, Greg Durrett & Dan Jurafsky

Logistics

- Homework 4 is due this Friday 3/8 (start now if you haven’t already)
  - Open AI API keys
- Quiz 6 will be out on Friday afternoon (due Monday)
- This week: Syntax & Parsing

Context-free grammar

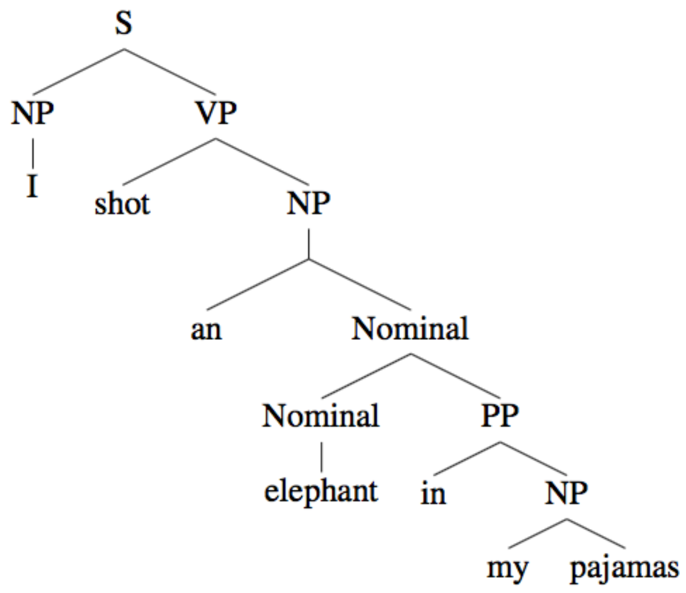
A context-free grammar defines how symbols in a language combine to form valid structures

NP	→	Det Nominal
NP	→	ProperNoun
Nominal	→	Noun | Nominal Noun
Det	→	a | the
Noun	→	flight

non-terminals

lexicon/
terminals

Constituents



Every internal node is a phrase

- my pajamas
- in my pajamas
- elephant in my pajamas
- an elephant in my pajamas
- shot an elephant in my pajamas
- I shot an elephant in my pajamas

Each phrase could be replaced by another of the same type of constituent

PCFG

- Probabilistic context-free grammar: each production is also associated with a probability.

- This lets us calculate the probability of a parse for a given sentence; for a given parse tree T for sentence S comprised of n rules from R (each A → β):

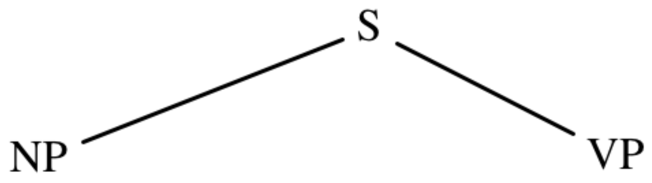
$$P(T, S) = \prod_{i}^{n} P(\beta \mid A)$$

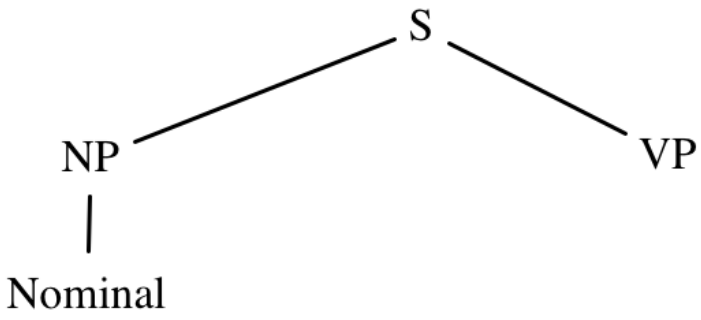
PCFG

N	Finite set of non-terminal symbols	NP, VP, S
Σ	Finite alphabet of terminal symbols	the, dog, a
R	Set of production rules, each A → β [p] p = P(β | A)	S → NP VP Noun → dog
S	Start symbol	

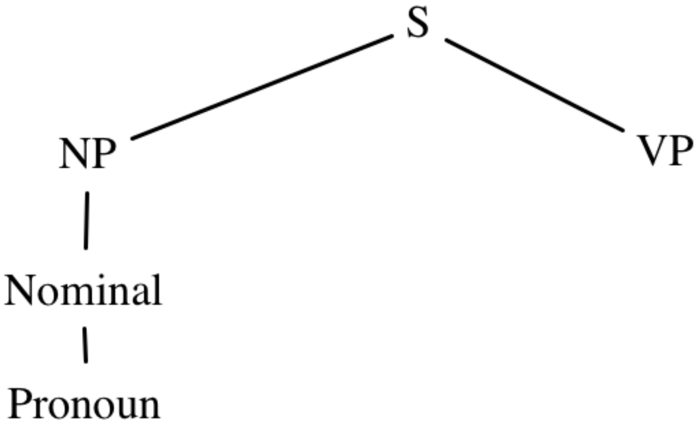
S

$P(\text{NP VP} \mid \text{S})$

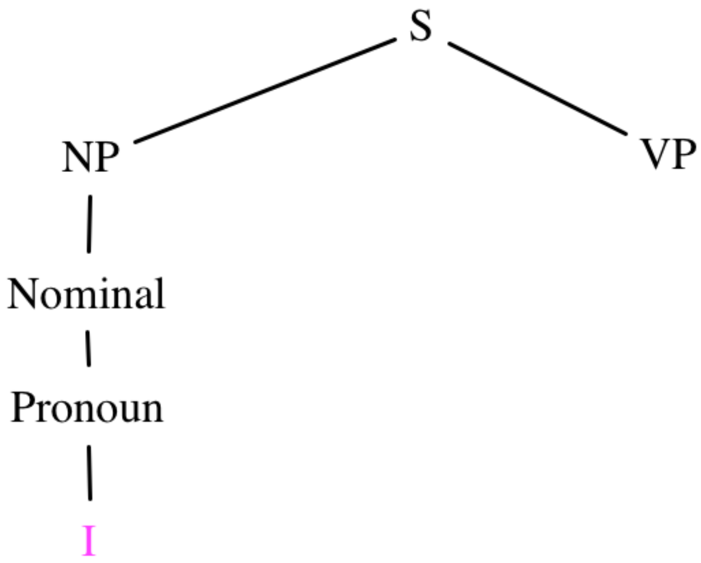




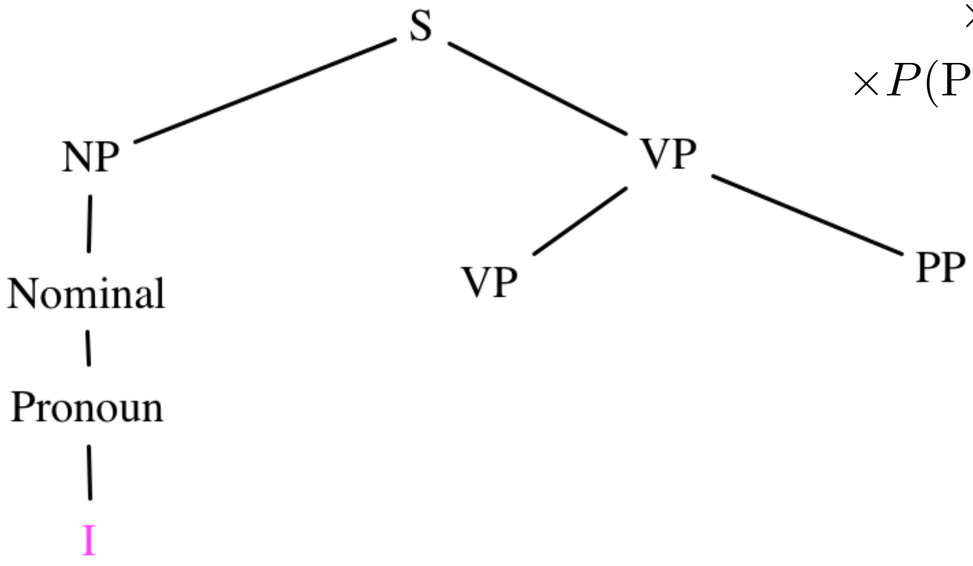
$$P(\text{NP VP} \mid \text{S})$$
$$\times P(\text{Nominal} \mid \text{NP})$$



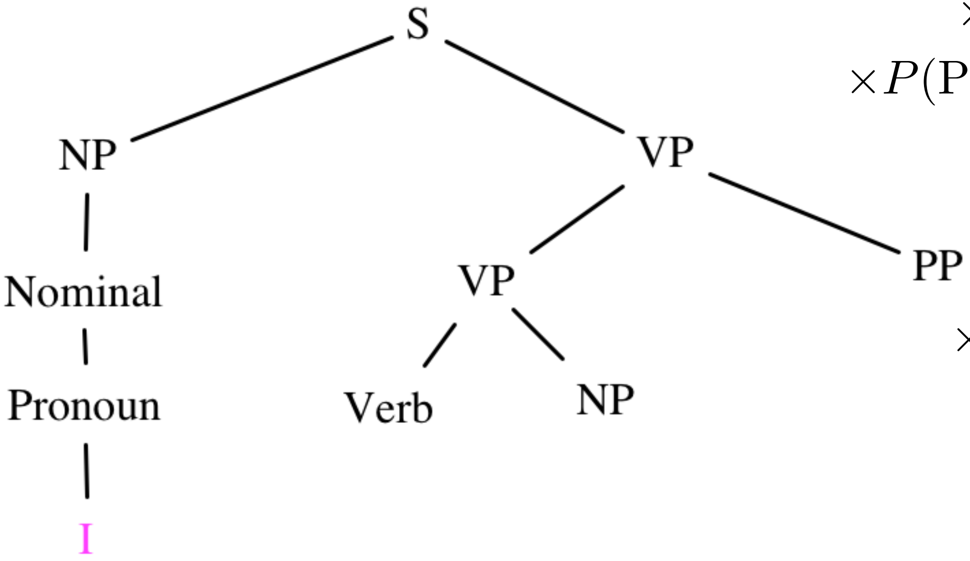
$$P(\text{NP VP} \mid \text{S})$$
$$\times P(\text{Nominal} \mid \text{NP})$$
$$\times P(\text{Pronoun} \mid \text{Nominal})$$



$$P(\text{NP VP} \mid \text{S})$$
$$\times P(\text{Nominal} \mid \text{NP})$$
$$\times P(\text{Pronoun} \mid \text{Nominal})$$
$$\times P(\text{I} \mid \text{Pronoun})$$

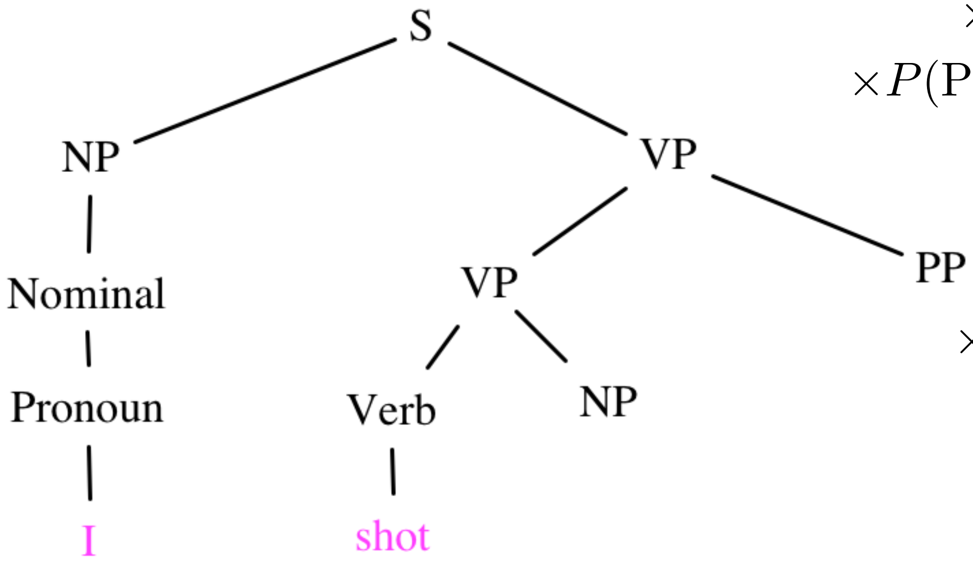


$$P(\text{NP VP} \mid \text{S})$$
$$\times P(\text{Nominal} \mid \text{NP})$$
$$\times P(\text{Pronoun} \mid \text{Nominal})$$
$$\times P(\text{I} \mid \text{Pronoun})$$
$$\times P(\text{VP PP} \mid \text{VP})$$

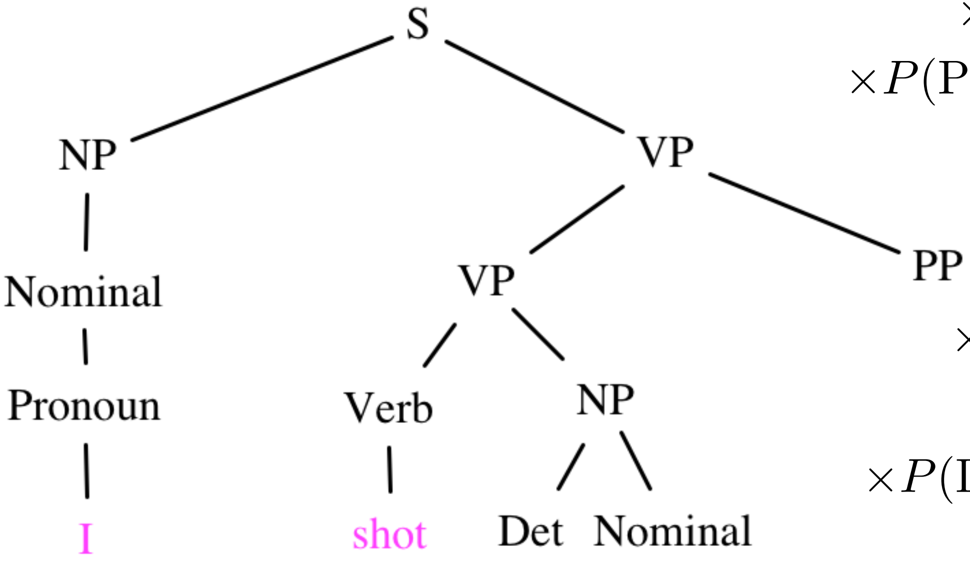


$$P(\text{NP VP} \mid \text{S})$$
$$\times P(\text{Nominal} \mid \text{NP})$$
$$\times P(\text{Pronoun} \mid \text{Nominal})$$
$$\times P(\text{I} \mid \text{Pronoun})$$
$$\times P(\text{VP PP} \mid \text{VP})$$

$$\times P(\text{Verb NP} \mid \text{VP})$$

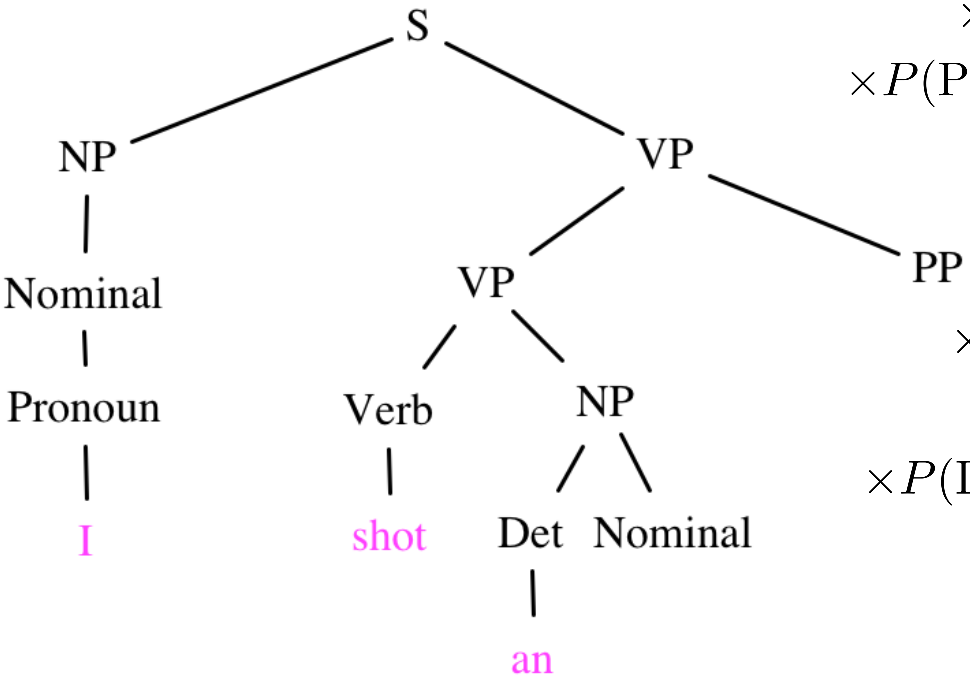


$$P(\text{NP VP} \mid \text{S})$$
$$\times P(\text{Nominal} \mid \text{NP})$$
$$\times P(\text{Pronoun} \mid \text{Nominal})$$
$$\times P(\text{I} \mid \text{Pronoun})$$
$$\times P(\text{VP PP} \mid \text{VP})$$
$$\times P(\text{Verb NP} \mid \text{VP})$$
$$\times P(\text{shot} \mid \text{Verb})$$

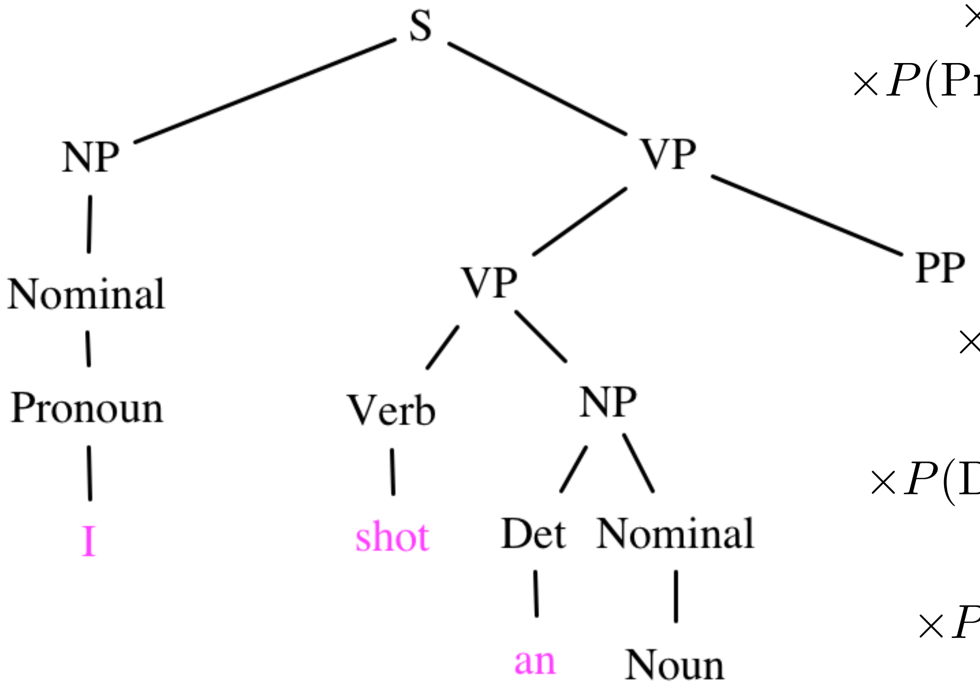


$$P(\text{NP VP} \mid \text{S})$$
$$\times P(\text{Nominal} \mid \text{NP})$$
$$\times P(\text{Pronoun} \mid \text{Nominal})$$
$$\times P(\text{I} \mid \text{Pronoun})$$
$$\times P(\text{VP PP} \mid \text{VP})$$

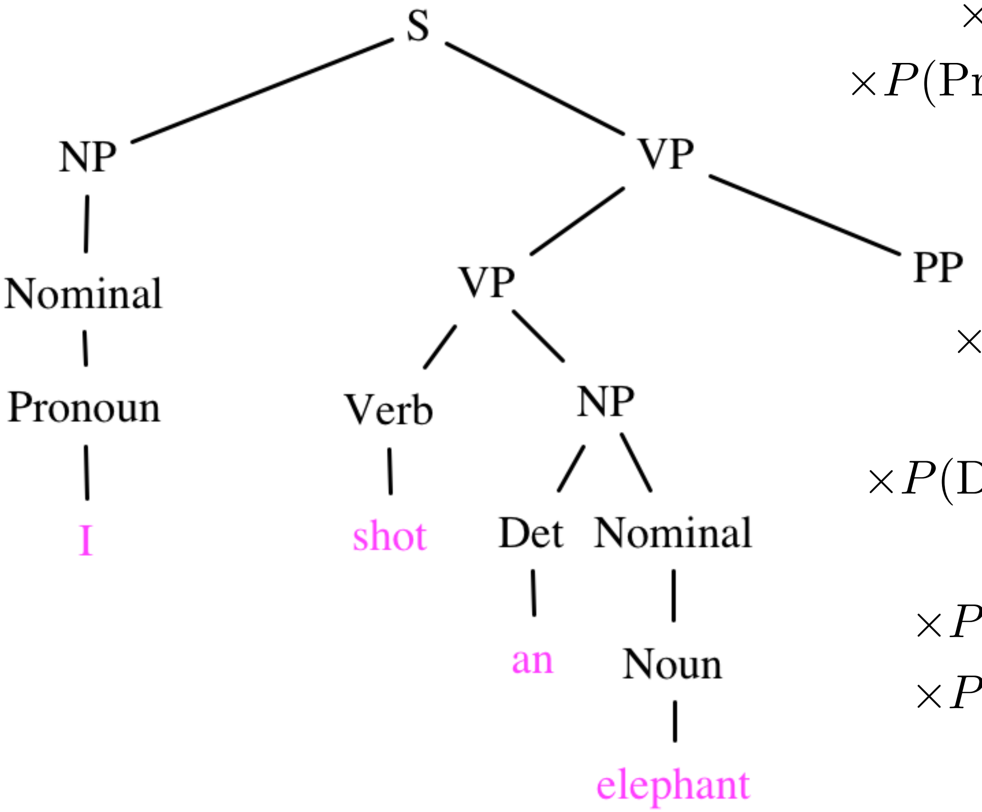
$$\times P(\text{Verb NP} \mid \text{VP})$$
$$\times P(\text{shot} \mid \text{Verb})$$
$$\times P(\text{Det Nominal} \mid \text{NP})$$



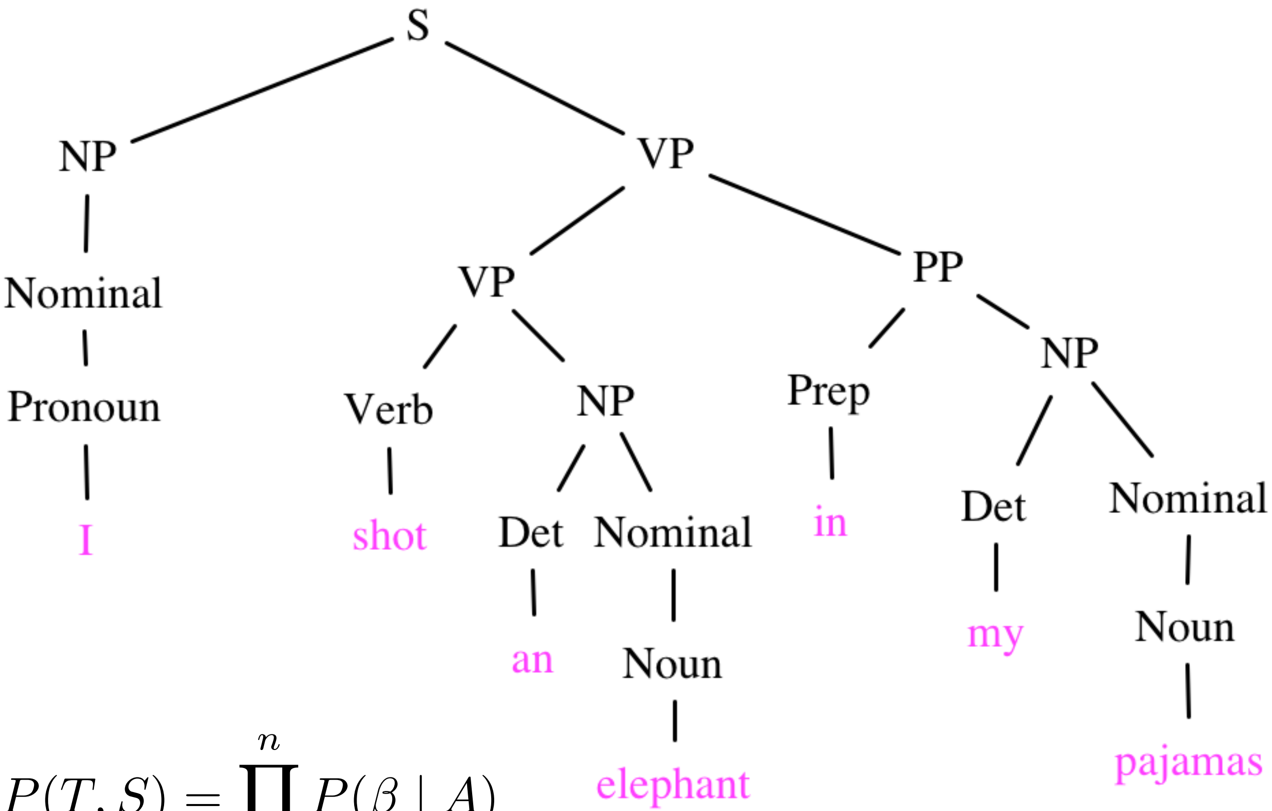
$$P(\text{NP VP} \mid \text{S})$$
$$\times P(\text{Nominal} \mid \text{NP})$$
$$\times P(\text{Pronoun} \mid \text{Nominal})$$
$$\times P(\text{I} \mid \text{Pronoun})$$
$$\times P(\text{VP PP} \mid \text{VP})$$
$$\times P(\text{Verb NP} \mid \text{VP})$$
$$\times P(\text{shot} \mid \text{Verb})$$
$$\times P(\text{Det Nominal} \mid \text{NP})$$
$$\times P(\text{an} \mid \text{Det})$$



$$P(\text{NP VP} \mid \text{S})$$
$$\times P(\text{Nominal} \mid \text{NP})$$
$$\times P(\text{Pronoun} \mid \text{Nominal})$$
$$\times P(\text{I} \mid \text{Pronoun})$$
$$\times P(\text{VP PP} \mid \text{VP})$$
$$\times P(\text{Verb NP} \mid \text{VP})$$
$$\times P(\text{shot} \mid \text{Verb})$$
$$\times P(\text{Det Nominal} \mid \text{NP})$$
$$\times P(\text{an} \mid \text{Det})$$
$$\times P(\text{Noun} \mid \text{Nominal})$$



$$
\begin{aligned}
& P(\text{NP VP} \mid \text{S}) \\
& \times P(\text{Nominal} \mid \text{NP}) \\
& \times P(\text{Pronoun} \mid \text{Nominal}) \\
& \times P(\text{I} \mid \text{Pronoun}) \\
& \times P(\text{VP PP} \mid \text{VP}) \\
& \times P(\text{Verb NP} \mid \text{VP}) \\
& \times P(\text{shot} \mid \text{Verb}) \\
& \times P(\text{Det Nominal} \mid \text{NP}) \\
& \times P(\text{an} \mid \text{Det}) \\
& \times P(\text{Noun} \mid \text{Nominal}) \\
& \times P(\text{elephant} \mid \text{Noun})
\end{aligned}
$$



$$P(T, S) = \prod_{i}^{n} P(\beta \mid A)$$

PCFGs

- A PCFG gives us a mechanism for assigning scores (here, probabilities) to different parses for the same sentence.
- But we often care about is finding the single best parse with the highest probability.

Context-free grammar

N	Finite set of non-terminal symbols	NP, VP, S
Σ	Finite alphabet of terminal symbols	the, dog, a
R	Set of production rules, each A → β β ∈ (Σ, *N*)	NP → DT JJ NN Noun → dog
S	Start symbol	

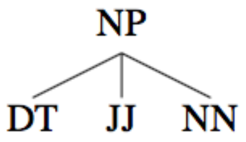
Chomsky Normal Form (CNF)

N	Finite set of non-terminal symbols	NP, VP, S
Σ	Finite alphabet of terminal symbols	the, dog, a
R	Set of production rules, each $A \rightarrow \beta$ β = single terminal (from Σ) or two non-terminals (from N)	S → NP VP Noun → dog
S	Start symbol	

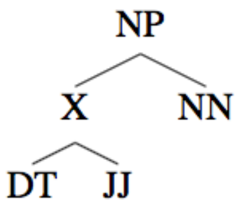
Chomsky Normal Form (CNF)

- Any CFG can be converted into weakly equivalent CNF (recognizing the same set of sentences as existing in the grammar but differing in their derivation).

NP → DT JJ NN



NP → X NN
X → DT JJ



S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nominal	→	Nominal PP
Nominal	→	NN
Nominal	→	NNS
Nominal	→	PRP
PP	→	IN NP
NP	→	DT NN
NP	→	Nominal
NP	→	PRP$ Nominal

VBD	→	shot
DT	→	an | my
NN	→	elephant
NNS	→	pajamas
PRP	→	I
PRP$	→	my
IN	→	in

I shot an elephant in my pajamas

S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nominal	→	Nominal PP
Nominal	→	pajamas | elephant | I
PP	→	IN NP
NP	→	DT NN
NP	→	pajamas | elephant | I
NP	→	PRP$ Nominal

VBD	→	shot
DT	→	an | my
PRP	→	I
PRP$	→	my
IN	→	in

I shot an elephant in my pajamas

CKY

- Cocke-Kasami-Younger algorithm (also CYK) for parsing from a grammar expressed in CNF.
    - Kasami (1965)
    - Younger (1967)
    - Cocke and Schwartz (1970)

- Bottom-up dynamic programming: once we discover a constituent, we can make it available for any rule that needs it.

0 I 1 shot 2 an 3 elephant 4 in 5 my 6 pajamas 7

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]						
	VBD [1,2]					
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nominal	→	Nominal PP
Nominal	→	pajamas | elephant | I
PP	→	IN NP
NP	→	DT NN
NP	→	pajamas | elephant | I
NP	→	PRP$ Nominal

VBD	→	shot
DT	→	an | my
PRP	→	I
PRP$	→	my
IN	→	in

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]						
	VBD [1,2]					
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

Each cell i,j keeps track of all phrase types that can be formed from *all* words from position i through position j

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]						
	VBD [1,2]					
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

What phrases can be formed from “shot an elephant in”

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]						
	VBD [1,2]					
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

What phrases can be formed from "I shot an elephant in my pajamas"

CNF

- In CNF, each non-terminal generates two non-terminals

S → NP VP

$[_S\ [_{NP}\ \text{I}]\ [_{VP}\ \text{shot an elephant in my pajamas}]\ ]$

- If the left-side non-terminal (S) spans tokens i-j, the right side (NP VP) must also span i-j, and there must be a single position k that separates them.

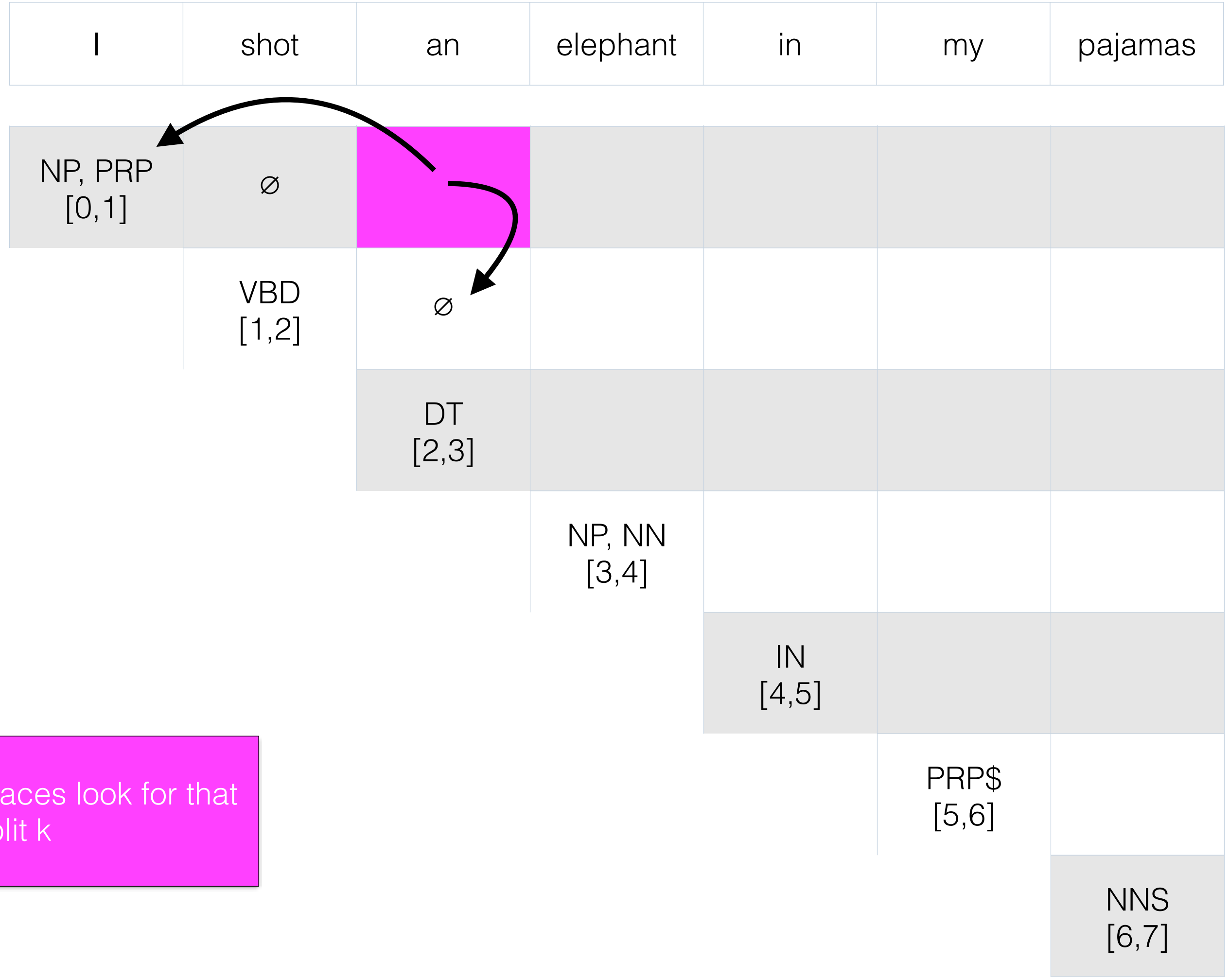
I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]						
	VBD [1,2]					
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

Does any rule generate PRP VBD?

i.e., is there any production in our CFG that generates:
? → PRP VBD

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅					
	VBD [1,2]					
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

Does any rule generate VBD DT?



Two possible places look for that split k

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅					
	VBD [1,2]	∅				
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

Two possible places look for that split k

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅					
	VBD [1,2]	∅				
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

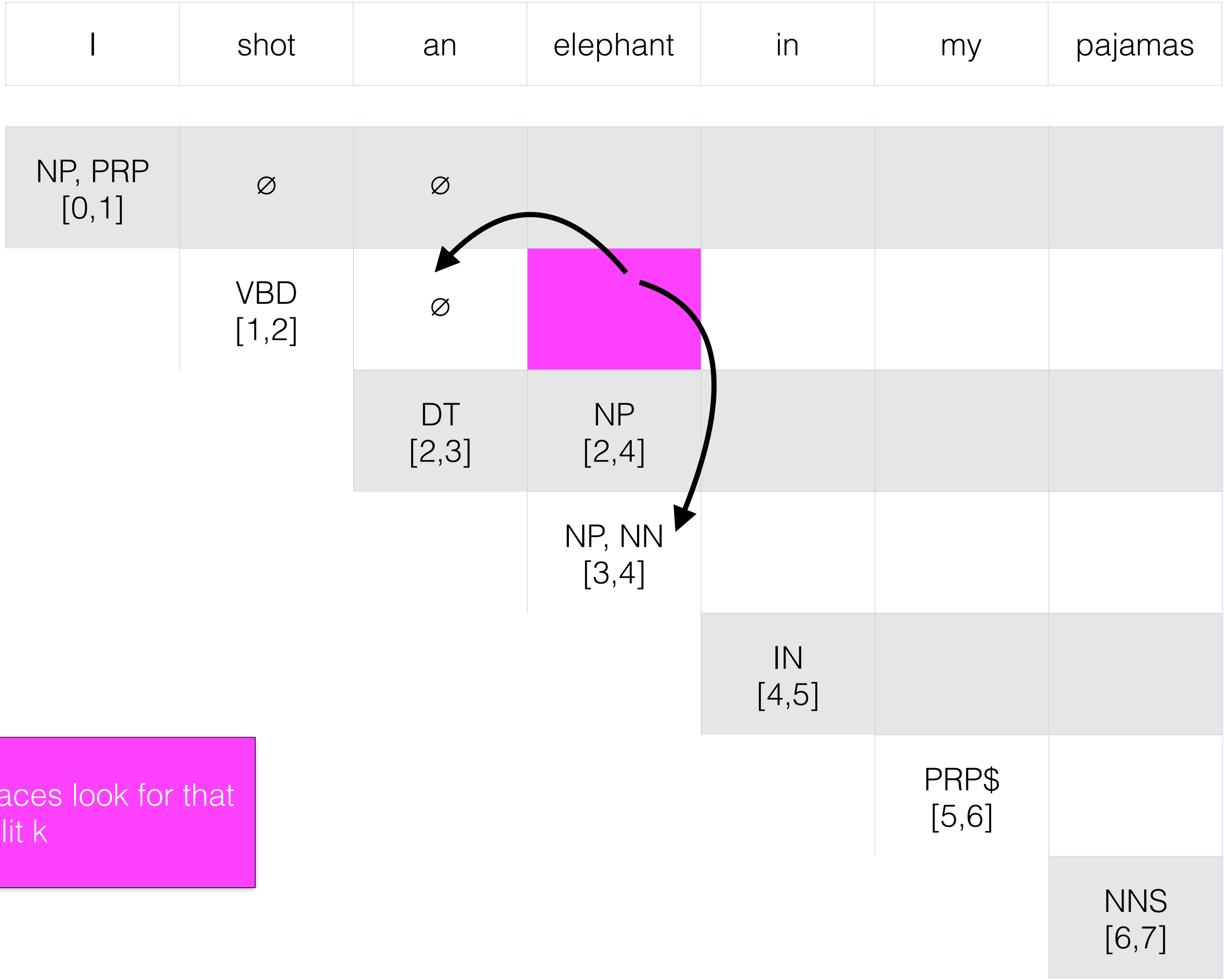
Two possible places look for that split k

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅				
	VBD [1,2]	∅				
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

Does any rule generate DT NN?

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅				
	VBD [1,2]	∅				
		DT [2,3]	NP [2,4]			
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

Two possible places look for that split k



Two possible places look for that split k

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅				
	VBD [1,2]	∅	VP [1,4]			
		DT [2,3]	NP [2,4]			
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

Three possible places look for that split k

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅				
	VBD [1,2]	∅	VP [1,4]			
		DT [2,3]	NP [2,4]			
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

Three possible places look for that split k

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅				
	VBD [1,2]	∅	VP [1,4]			
		DT [2,3]	NP [2,4]			
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

Three possible places look for that split k

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅				
	VBD [1,2]	∅	VP [1,4]			
		DT [2,3]	NP [2,4]			
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

Three possible places look for that split k

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]			
	VBD [1,2]	∅	VP [1,4]			
		DT [2,3]	NP [2,4]			
			NP, NN [3,4]			
				IN [4,5]		
					PRP$ [5,6]	
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	
			NP, NN [3,4]	∅	∅	
				IN [4,5]	∅	
					PRP$ [5,6]	
						NNS [6,7]

*elephant in
*an elephant in
*shot an elephant in
*I shot an elephant in

*in my
*elephant in my
*an elephant in my
*shot an elephant in my
*I shot an elephant in my

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	
			NP, NN [3,4]	∅	∅	
				IN [4,5]	∅	
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	
			NP, NN [3,4]	∅	∅	
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	NP [3,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	NP [3,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	NP [3,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	NP [3,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	NP [3,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	NP [3,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	
		DT [2,3]	NP [2,4]	∅	∅	NP [3,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	VP_1, VP_2 [1,7]
		DT [2,3]	NP [2,4]	∅	∅	NP [2,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	
	VBD [1,2]	∅	VP [1,4]	∅	∅	VP_1, VP_2 [1,7]
		DT [2,3]	NP [2,4]	∅	∅	NP [2,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

Possibilities:

$S_1 \rightarrow NP\ VP_1$
$S_2 \rightarrow NP\ VP_2$
? → S PP
? → $PRP\ VP_1$
? → $PRP\ VP_2$

I	shot	an	elephant	in	my	pajamas

NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	S_1, S_2 [0,7]
	VBD [1,2]	∅	VP [1,4]	∅	∅	VP_1, VP_2 [1,7]
		DT [2,3]	NP [2,4]	∅	∅	NP [2,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

Success! We've recognized a total of two valid parses

CKY algorithm

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

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  for all $\{A \mid A \rightarrow words[j] \in grammar\}$
    $table[j-1, j] \leftarrow table[j-1, j] \cup A$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      for all $\{A \mid A \rightarrow BC \in grammar$ **and** $B \in table[i,k]$ **and** $C \in table[k,j]\}$
        $table[i,j] \leftarrow table[i,j] \cup A$

Figure 12.5 The CKY algorithm.

I	shot	an	elephant	in	my	pajamas
NP, PRP [0,1]	∅	∅	S [0,4]	∅	∅	S_1, S_2 [0,7]
	VBD [1,2]	∅	VP [1,4]	∅	∅	VP_1, VP_2 [1,7]
		DT [2,3]	NP [2,4]	∅	∅	NP [2,7]
			NP, NN [3,4]	∅	∅	NP [3,7]
				IN [4,5]	∅	PP [4,7]
					PRP$ [5,6]	NP [5,7]
						NNS [6,7]

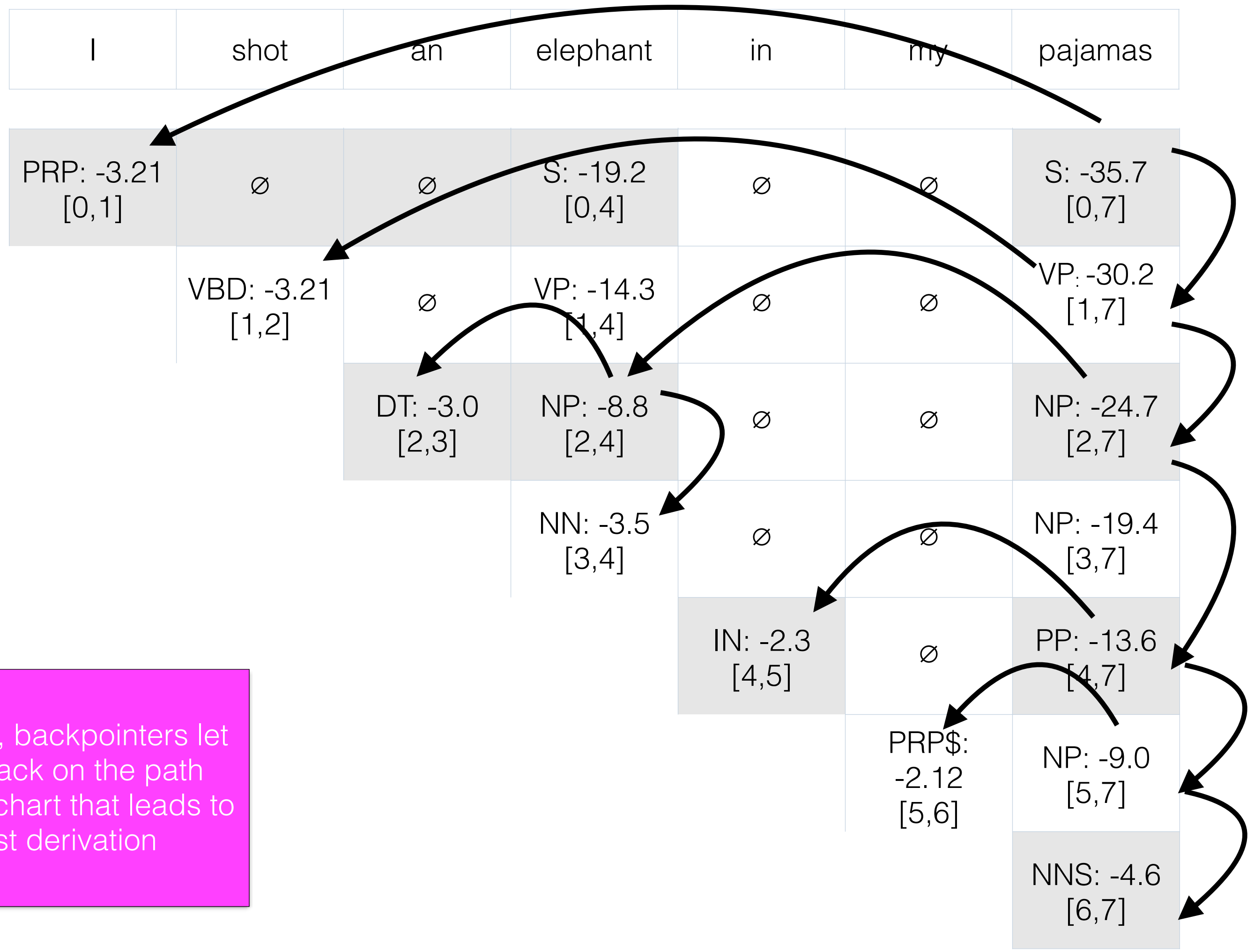
Runtime complexity?

CFG

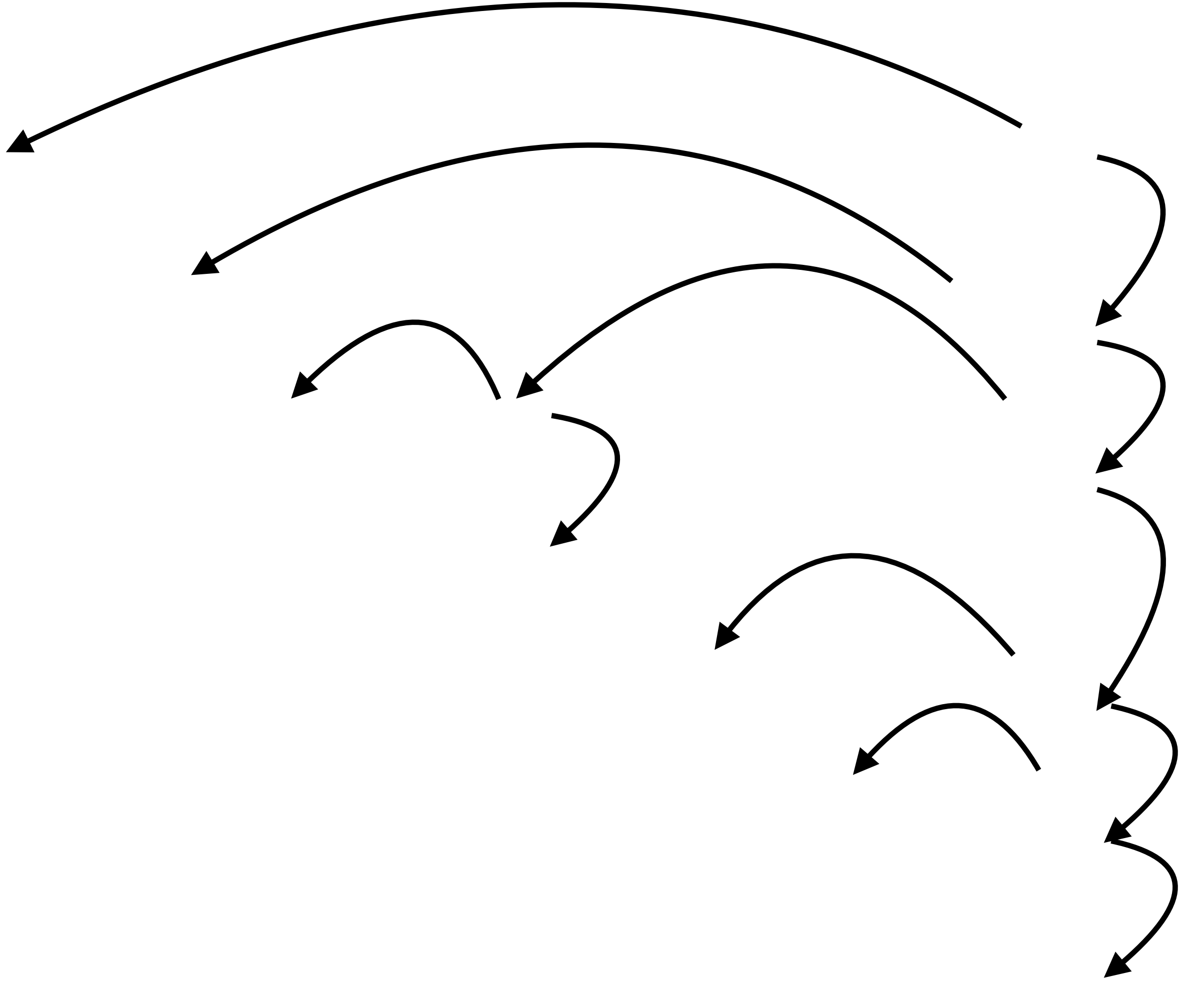
- This use of CKY allows us to:
    - check whether a sentence in grammatical in the language defined by the CFG
    - enumerate all possible parses for a sentence
- But it doesn’t tell us on its own which of those possible parses is most likely.

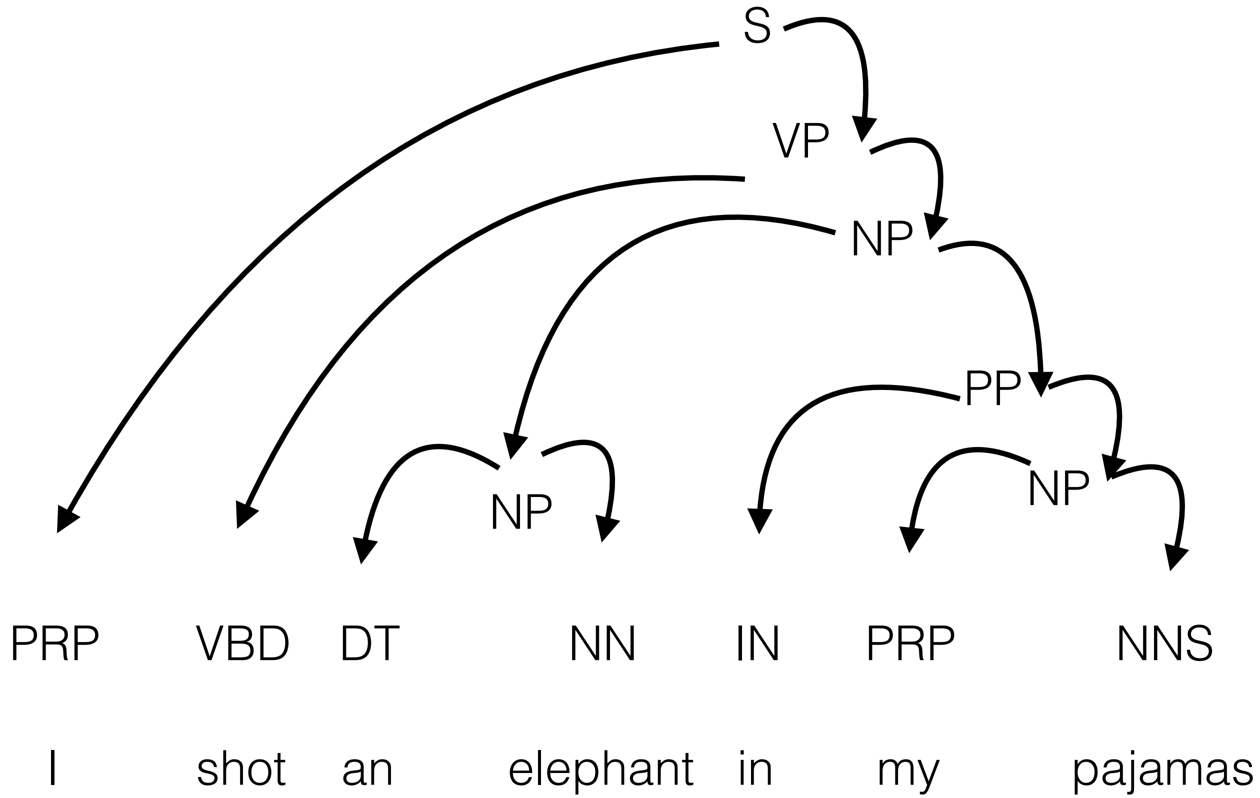
PCFGs

- A PCFG gives us a mechanism for assigning scores (here, probabilities) to different parses for the same sentence.
- We often care about is finding the single best parse with the highest probability.
- We calculate the max probability parse using CKY by storing the probability of each phrase within each cell as we build it up.



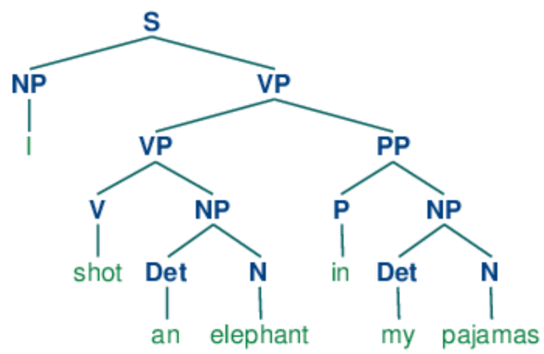
As in Viterbi, backpointers let us keep track on the path through the chart that leads to the best derivation



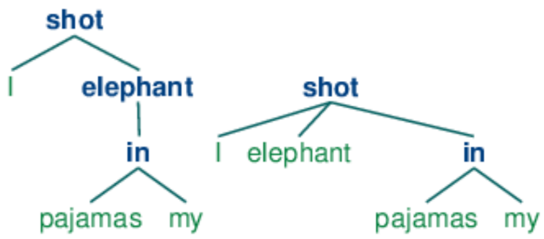

S
VP
NP
PP
NP
NP
PRP
VBD
DT
NN
IN
PRP
NNS
I
shot
an
elephant
in
my
pajamas

Formalisms

Phrase structure grammar
(Chomsky 1957)



Dependency grammar
(Mel'čuk 1988; Tesnière 1959; Pāṇini)



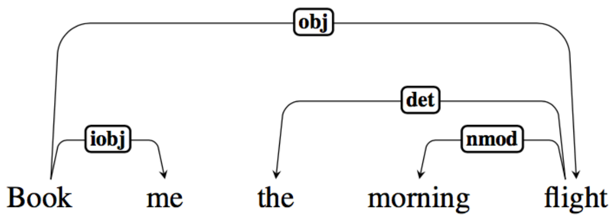
Dependency syntax

- Enables "Who Did What to Whom" kind of analysis for semantics.
- Syntactic structure = asymmetric, binary relations between words.

Tesnier 1959; Nivre 2005

Trees

- A dependency structure is a directed graph G = (V,A) consisting of a set of vertices *V* and arcs *A* between them. Typically constrained to form a tree:
    - Single root vertex with no incoming arcs
    - Every vertex has exactly one incoming arc except root (single head constraint)
    - There is a unique path from the root to each vertex in V (acyclic constraint)



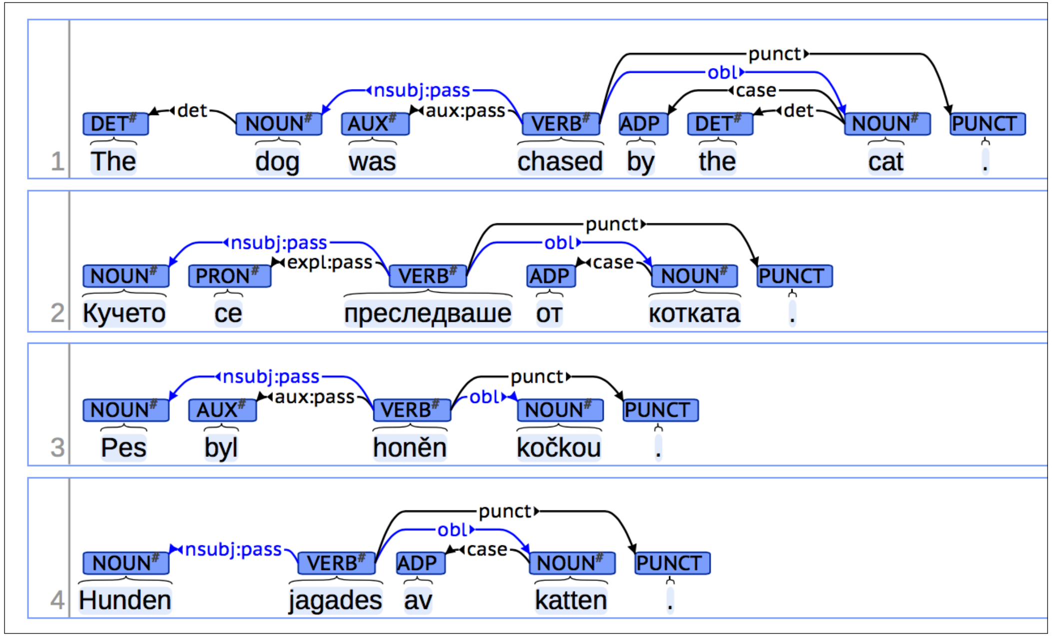
Universal Dependencies

English

Bulgarian

Czech

Swedish

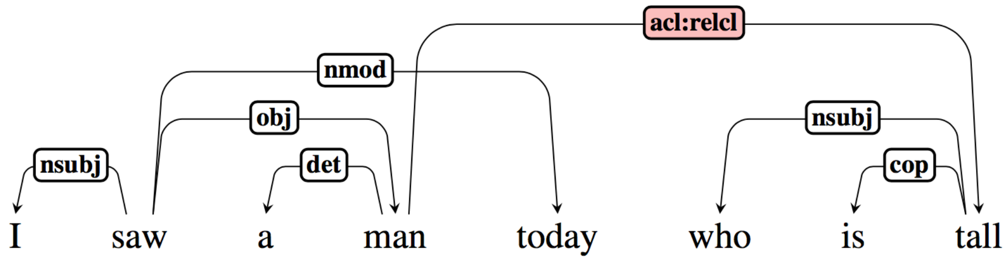


http://universaldependencies.org

Dependency parsing

- Transition-based parsing
  - O(n)
  - Only projective structures (pseudo-projective [Nivre and Nilsson 2005])
- Graph-based parsing
  - $O(n^2)$
  - Projective and non-projective trees

Projectivity



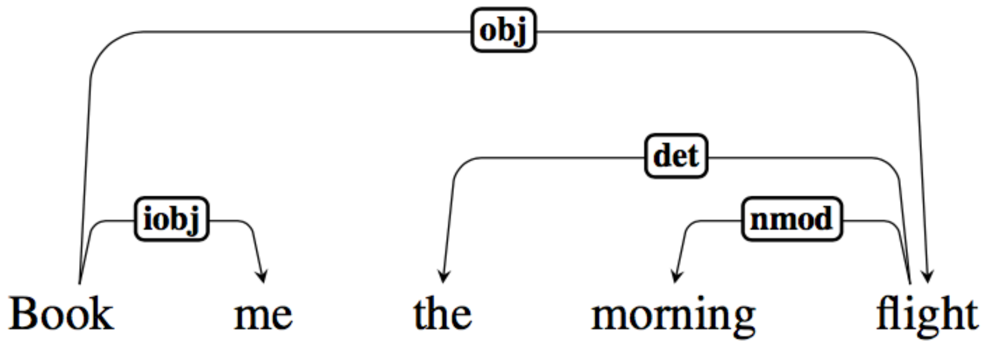
- An arc between a head and dependent is projective if there is a path from the head to every word between the head and dependent. Every word between head and dependent is a descendent of the head.

Transition-based parsing

- Basic idea: parse a sentence into a dependency tree by training a local classifier to predict a parser's next action from its current configuration.

Configuration

- Stack
- Input buffer of words
- Arcs in a parsed dependency tree
- Parsing = sequences of transitions through space of possible configurations


obj
det
iobj
nmod
Book
me
the
morning
flight

∅ book me the morning flight

stack	action	arc

∅ book me the morning flight

stack

action

arc

LeftArc(label): assert relation between head at $stack_1$ and dependent at $stack_{2:}$ remove $stack_2$

RightArc(label): assert relation between head at $stack_2$ and dependent at $stack_1$; remove $stack_1$

☞ Shift: Remove word from front of input buffer (∅) and push it onto stack book me the morning flight

stack		action	arc
		LeftArc(label): assert relation between head at $stack_1$ (∅) and dependent at $stack_2$: remove $stack_2$	
		RightArc(label): assert relation between head at $stack_2$ and dependent at $stack_1$ (∅); remove $stack_1$ (∅)	
∅	☞	Shift: Remove word from front of input buffer (book) and push it onto stack	

If we remove an element from the stack, it can't have any further dependents

me the morning flight

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	
book	RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	
∅	☞ Shift: Remove word from front of input buffer (me) and push it onto stack	

the morning flight

stack	action	arc
		iobj(book, me)
	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book); remove $stack_2$ (book)	
me	☞ RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
book		
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

the morning flight

stack	action	arc
		iobj(book, me)
	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	
book	RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

morning flight

stack	action	arc
		iobj(book, me)
	LeftArc(label): assert relation between head at $stack_1$ (the) and dependent at $stack_2$ (book): remove $stack_2$ (book)	
the	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (the); remove $stack_1$ (the)	
book		
∅	☞ Shift: Remove word from front of input buffer (morning) and push it onto stack	

flight

stack | action | arc

stack	action	arc
		iobj(book, me)
morning	LeftArc(label): assert relation between head at $stack_1$ (morning) and dependent at $stack_2$ (the): remove $stack_2$ (the)	
the	RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (morning); remove $stack_1$ (morning)	
book		
∅	☞ Shift: Remove word from front of input buffer (flight) and push it onto stack	

stack	action	arc
flight	☞ LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (morning); remove $stack_2$ (morning)	*iobj(book, me)*
morning	RightArc(label): assert relation between head at $stack_2$ (morning) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*nmod(flight, morning)*
the	~~Shift: Remove word from front of input buffer and push it onto stack~~	
book		
∅		

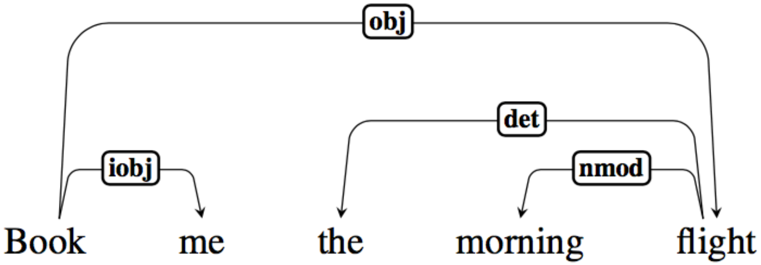
stack	action	arc
flight	☞ LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (the): remove $stack_2$ (the)	*iobj(book, me)*
the	RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*nmod(flight, morning)*
book	~~Shift: Remove word from front of input buffer and push it onto stack~~	*det(flight, the)*
∅		

stack	action	arc
flight	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (book): remove $stack_2$ (book)	*iobj(book, me)*
	☞ RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*nmod(flight, morning)*
book	~~Shift: Remove word from front of input buffer and push it onto stack~~	*det(flight, the)*
∅		*obj(book, flight)*

This is our parse

stack	action	arc
		iobj(book, me)
	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	*nmod(flight, morning)*
		det(flight, the)
	☞ RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	*obj(book, flight)*
book		*root(∅, book)*
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	

This is our parse



arc

iobj(book, me)

nmod(flight, morning)

det(flight, the)

obj(book, flight)

root(∅, book)

Let's go back to this earlier configuration

the morning flight

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book): remove $stack_2$ (book)	
me	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
book		
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

Output space $\mathcal{Y}$ =

Shift
LeftArc(nsubj)
RightArc(nsubj)
LeftArc(det)
RightArc(det)
LeftArc(obj)
RightArc(obj)
…

- This is a multiclass classification problem: given the current configuration — i.e., the elements in the stack, the words in the buffer, and the arcs created so far, what's the best transition?

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

buffer

the morning flight

arc

feature	example
$stack_1$ = me	1
$stack_2$ = book	1
$stack_1$ POS = PRP	1
$buffer_1$ = the	1
$buffer_2$ = morning	1
$buffer_1$ = today	0
$buffer_1$ POS = RB	0
$stack_1$ = me AND $stack_2$ = book	1
$stack_1$ = PRP AND $stack_2$ = VB	1
iobj(book,*) in arcs	0

Use any multiclass classification model

- Logistic regression
- SVM
- NB
- Neural network

feature	example	β
$stack_1$ = me	1	0.7
$stack_2$ = book	1	1.3
$stack_1$ POS = PRP	1	6.4
$buffer_1$ = the	1	-1.3
$buffer_2$ = morning	1	-0.07
$buffer_1$ = today	0	0.52
$buffer_1$ POS = RB	0	-2.1
$stack_1$ = me AND $stack_2$ =	1	0
$stack_1$ = PRP AND $stack_2$ =	1	-0.1
iobj(book,*) in arcs	0	3.2

Training

We're training to predict the parser action —Shift, RightArc(label), LeftArc(label)—given the featurized configuration

Configuration features	Label
<stack1 = me, 1>, <stack2 = book, 1>, <stack1 POS = PRP, 1>, <buffer1 = the, 1>,	Shift
<stack1 = me, 0>, <stack2 = book, 0>, <stack1 POS = PRP, 0>, <buffer1 = the, 0>,	RightArc(det)
<stack1 = me, 0>, <stack2 = book, 1>, <stack1 POS = PRP, 0>, <buffer1 = the, 0>,	RightArc(nsubj)

Neural Shift-Reduce Parsing

- We can train a neural shift-reduce parser by just changing how we:
  - represent the configuration
  - predict the label from that representation
- Otherwise training and prediction remains the same.

Chen and Manning (2014), “A Fast and Accurate Dependency Parser using Neural Networks”

Neural Shift-Reduce Parsing

Softmax layer:

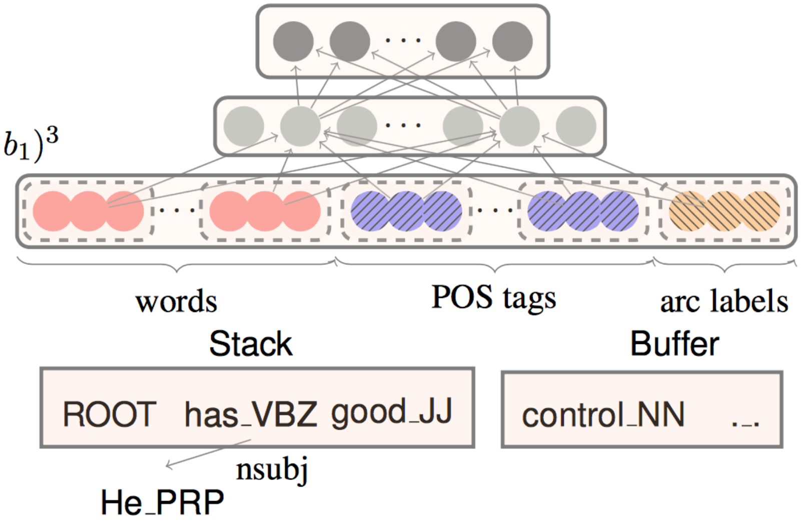
$$p = \mathtt{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

Input layer: $[x^w, x^t, x^l]$

Configuration



Chen and Manning (2014), "A Fast and Accurate Dependency Parser using Neural Networks"

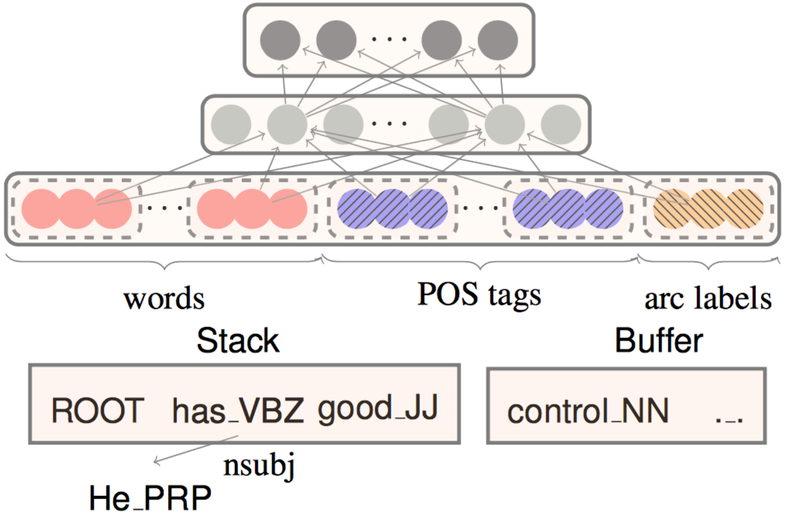
Neural Shift-Reduce Parsing

Representation for configuration:

- Embeddings for words/POS tags on top of stack
- Embeddings for words/POS tags at front of buffer
- Embeddings for existing arc labels

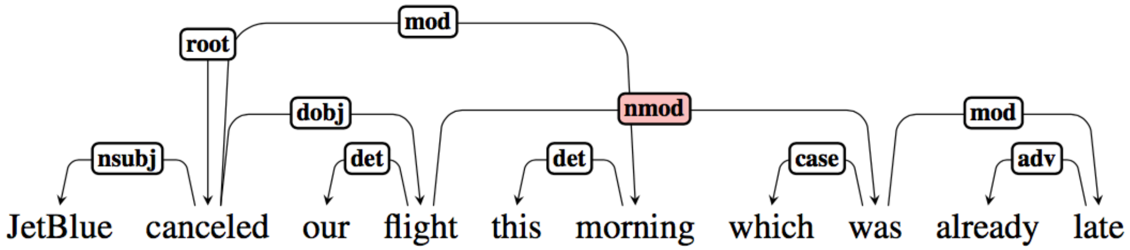
Classifier:

- Feed-forward neural network (input representation has a fixed dimensionality)



Chen and Manning (2014), “A Fast and Accurate Dependency Parser using Neural Networks”

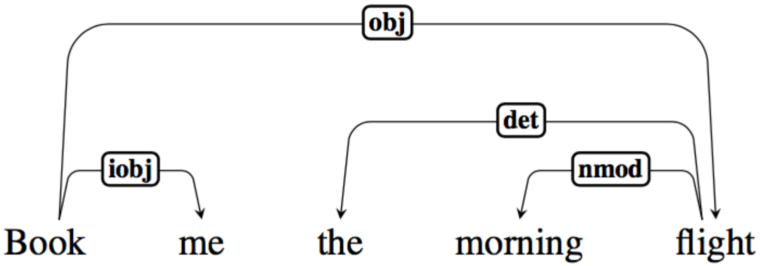
Training data



Our training data comes from treebanks (native dependency syntax or converted to dependency trees).

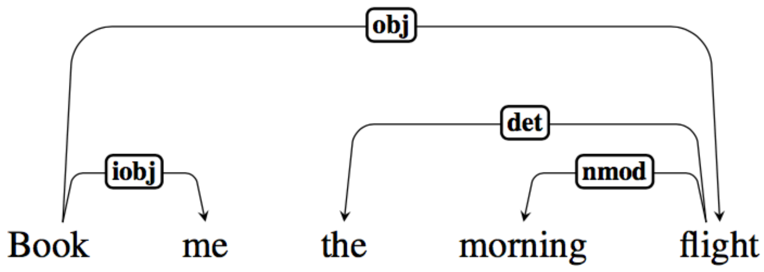
Oracle

- An algorithm for converting a gold-standard dependency tree into a series of actions a transition-based parser should follow to yield the tree.



→

Configuration features	Label
<stack1 = “”>, <stack2 = “”>, <stack1 POS = “”>, <buffer1 = ∅>,	Shift
<stack1 = ∅>, <stack2 = “”>, <stack1 POS = ∅>, <buffer1 = book>,	Shift
<stack1 = book>, <stack2 =∅>, <stack1 POS = VB>, <buffer1 = me>,	Shift



This is our parse

arc

iobj(book, me)

nmod(flight, morning)

det(flight, the)

obj(book, flight)

root(∅, book)

∅ book me the morning flight

stack	action	gold tree
		iobj(book, me)
		nmod(flight, morning)
		det(flight, the)
		obj(book, flight)
		root(∅, book)

∅ book me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	*iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

root(∅, book) exists but book has dependents in gold tree!

book me the morning flight

stack	action	gold tree
∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	*iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

iobj(book, me) exists and me has no dependents in gold tree

me the morning flight

stack	action	gold tree
book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	*iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

the morning flight

stack	action	gold tree
		✅ *iobj(book, me)*
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*nmod(flight, morning)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*det(flight, the)*
me		*obj(book, flight)*
book		*root(∅, book)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	

morning flight

stack	action	gold tree
the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$, $stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
morning		*nmod(flight, morning)*
the	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*det(flight, the)*
book		*obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

nmod(flight,morning)

stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
morning	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	✅ *nmod(flight, morning)*
the	Else shift: Remove word from front of input buffer and push it onto stack	*det(flight, the)*
book		*obj(book, flight)*
∅		*root(∅, book)*

det(flight,the)

stack	action	gold tree
flight the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* ✅ *det(flight, the)* *obj(book, flight)* *root(∅, book)*

obj(book,flight)

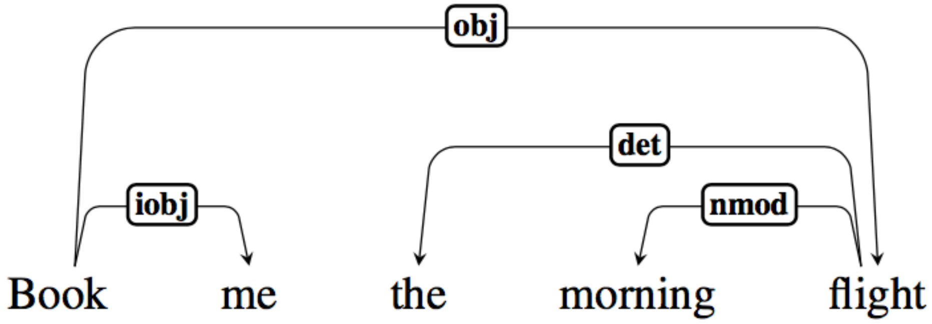
stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	✅ *nmod(flight, morning)*
		✅ *det(flight, the)*
book	Else shift: Remove word from front of input buffer and push it onto stack	✅ *obj(book, flight)*
∅		*root(∅, book)*

root(∅, book) *and* book has no more dependents we haven't seen

stack	action	gold tree
book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* ✅ *det(flight, the)* ✅ *obj(book, flight)* ✅ *root(∅, book)*

With only ∅ left on the stack and nothing in the buffer, we're done

stack	action	gold tree
∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	✅ *nmod(flight, morning)*
	Else shift: Remove word from front of input buffer and push it onto stack	✅ *det(flight, the)*
		✅ *obj(book, flight)*
		✅ *root(∅, book)*



Shift
Shift
Shift
RightArc(iobj)
Shift
Shift
Shift
LeftArc(nmod)
LeftArc(det)
RightArc(obj)
RightArc(root)

Logistics

- Homework 4 is due this Friday 3/8 (start now if you haven’t already)
    - Open AI API keys
- Quiz 6 will be out on Friday afternoon (due Monday)
- Next week: Semantics