



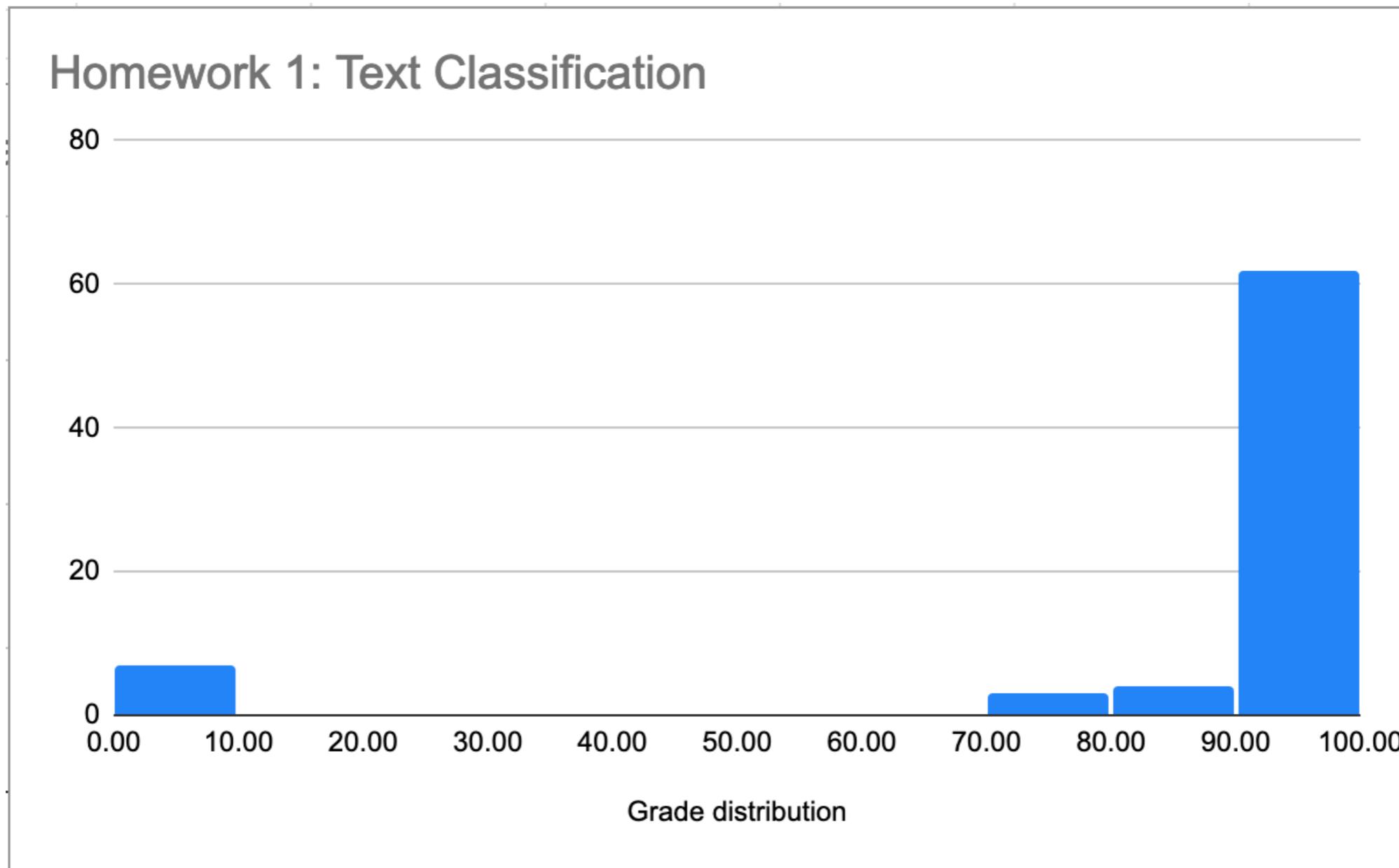
SI 630

Natural Language Processing: Algorithms and People

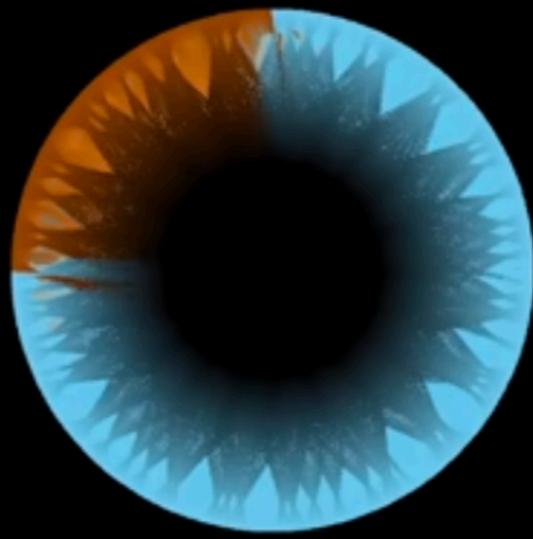
Lecture 6: Parsing
Feb 12, 2020

David Jurgens
jurgens@umich.edu

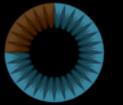
Homework 1 grades are out!



i



3Blue1Brown



◀ ▶ ▶| 🔍 0:02 / 19:13

CC HD 🎞 [] []

3BLUE1BROWN SERIES S3 • E1

But what is a Neural Network? | Deep learning, chapter 1

Today's journey

- What's the structure of language and how do we model it?
- Two algorithms for computationally determining structure



What is the subject?

I saw an elephant in the zoo

What is the verb?

I saw an elephant in the zoo

What is “in the zoo”?

I saw an elephant in the zoo

What is “in the zoo”?

I saw an elephant in the zoo

How this phrase relates to the rest
of the sentence is an example of
structure in language.

Most sentences have lots of implicit structure

While on vacation, my friend quickly saw a big elephant at the zoo

Most sentences have lots of implicit structure

While on vacation, my friend quickly saw a big elephant at the zoo

Most sentences have lots of implicit structure

While on vacation, my friend quickly saw a big elephant at the zoo

Most sentences have lots of implicit structure

While on vacation, my friend quickly saw a big elephant at the zoo

Most sentences have lots of implicit structure

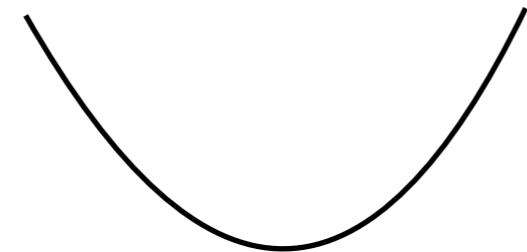
While on vacation, my friend quickly saw a big elephant at the zoo

Most sentences have lots of implicit structure

While on vacation, my friend quickly saw a big elephant at the zoo

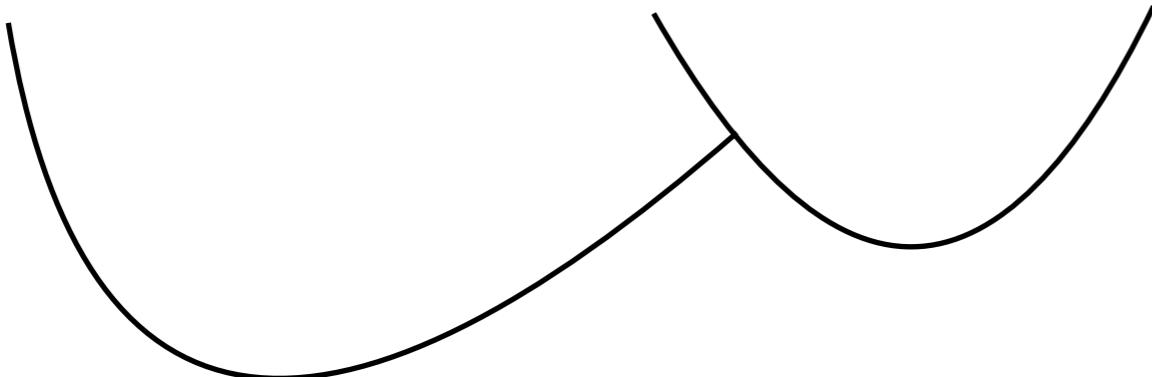
Most sentences have lots of implicit structure

While on vacation, my friend quickly saw a big elephant at the zoo



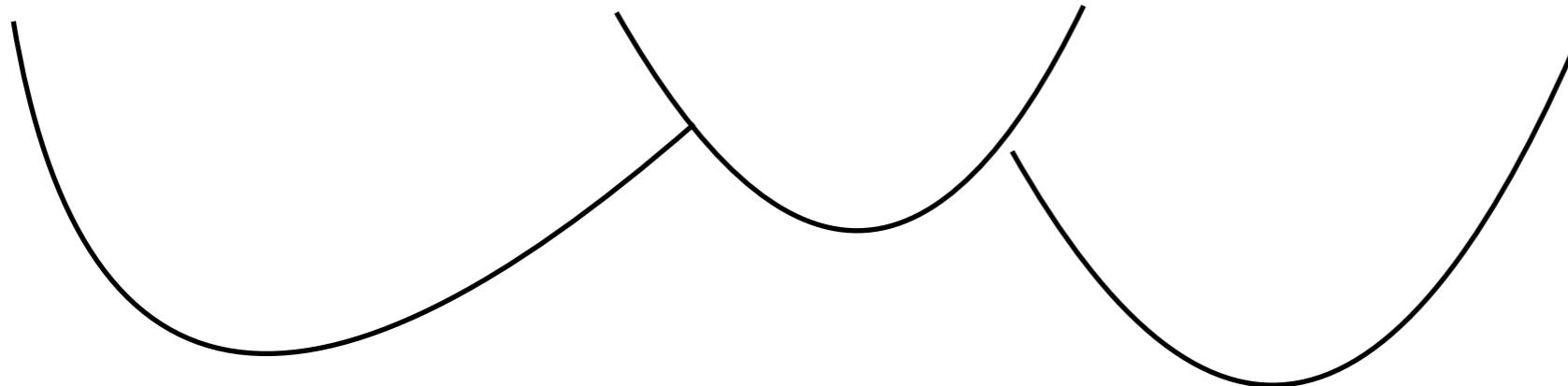
Most sentences have lots of implicit structure

While on vacation, my friend quickly saw a big elephant at the zoo



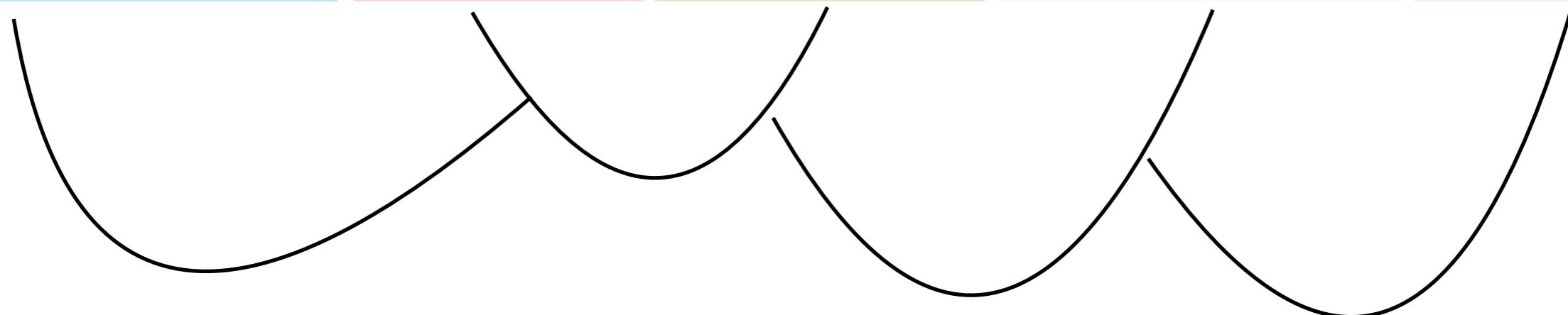
Most sentences have lots of implicit structure

While on vacation, my friend quickly saw a big elephant at the zoo



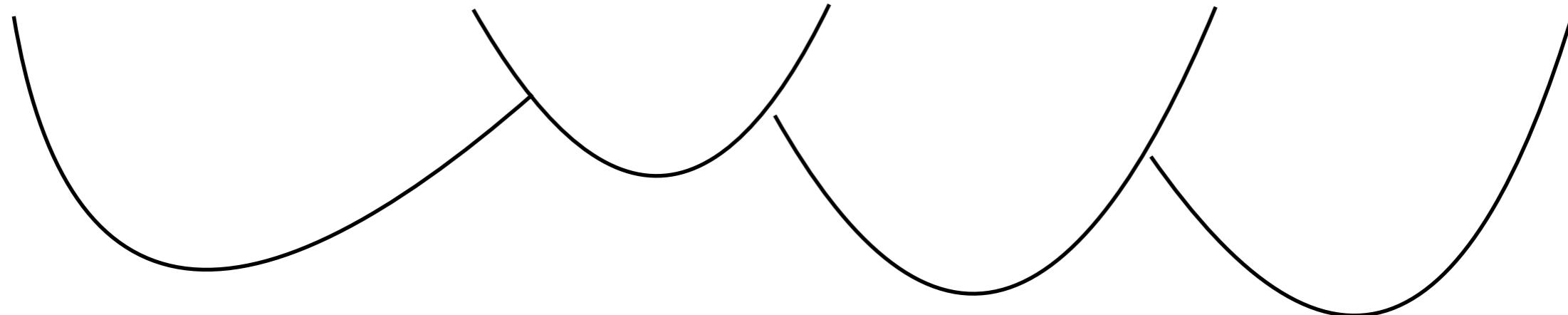
Most sentences have lots of implicit structure

While on vacation, my friend quickly saw a big elephant at the zoo



Most sentences have lots of implicit structure

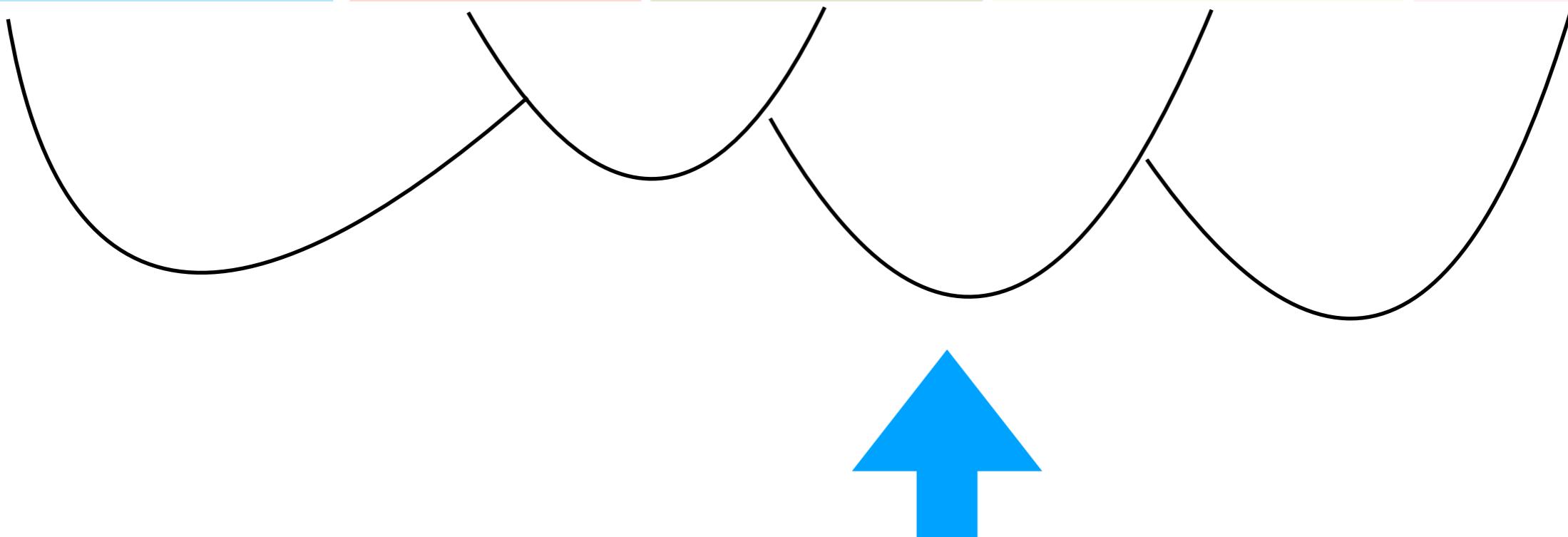
While on vacation, my friend quickly saw a big elephant at the zoo



We can think of a sentence's **syntax** as a **tree structure** describing how these units relate to each other

How do we describe this structure?

While on vacation, my friend quickly saw a big elephant at the zoo

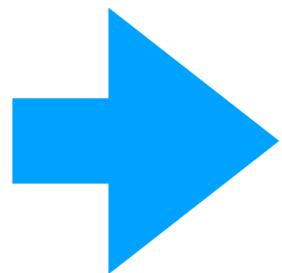


How do we describe this structure?

Cats eat food

How do we describe this structure?

Terminal (leaf) nodes

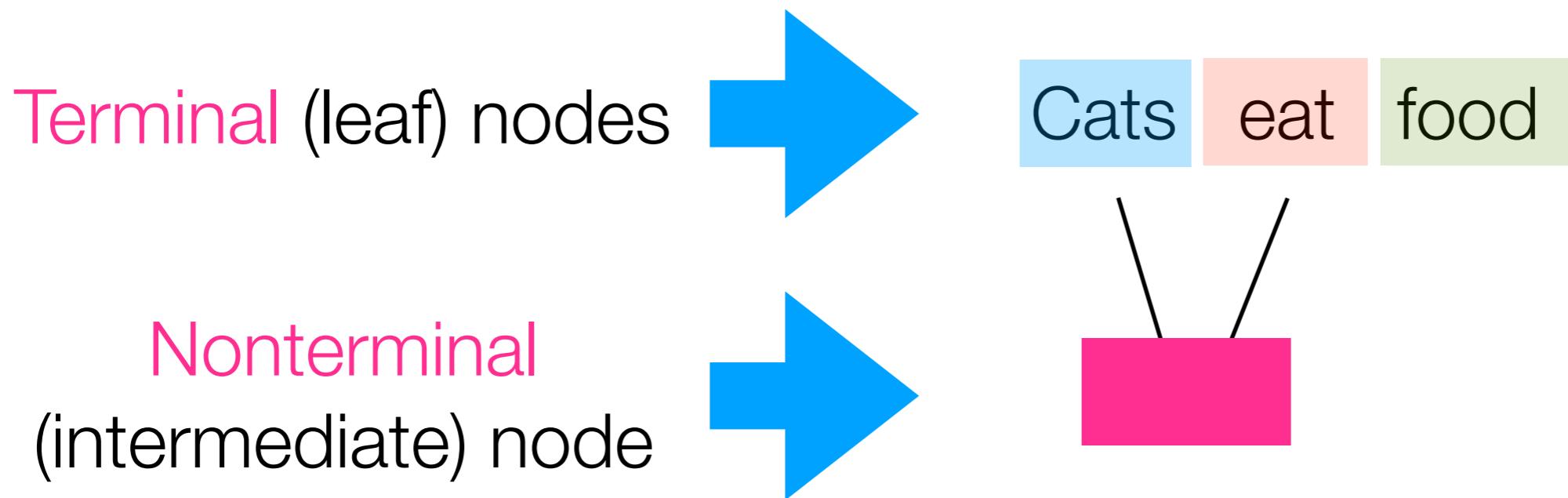


Cats eat food

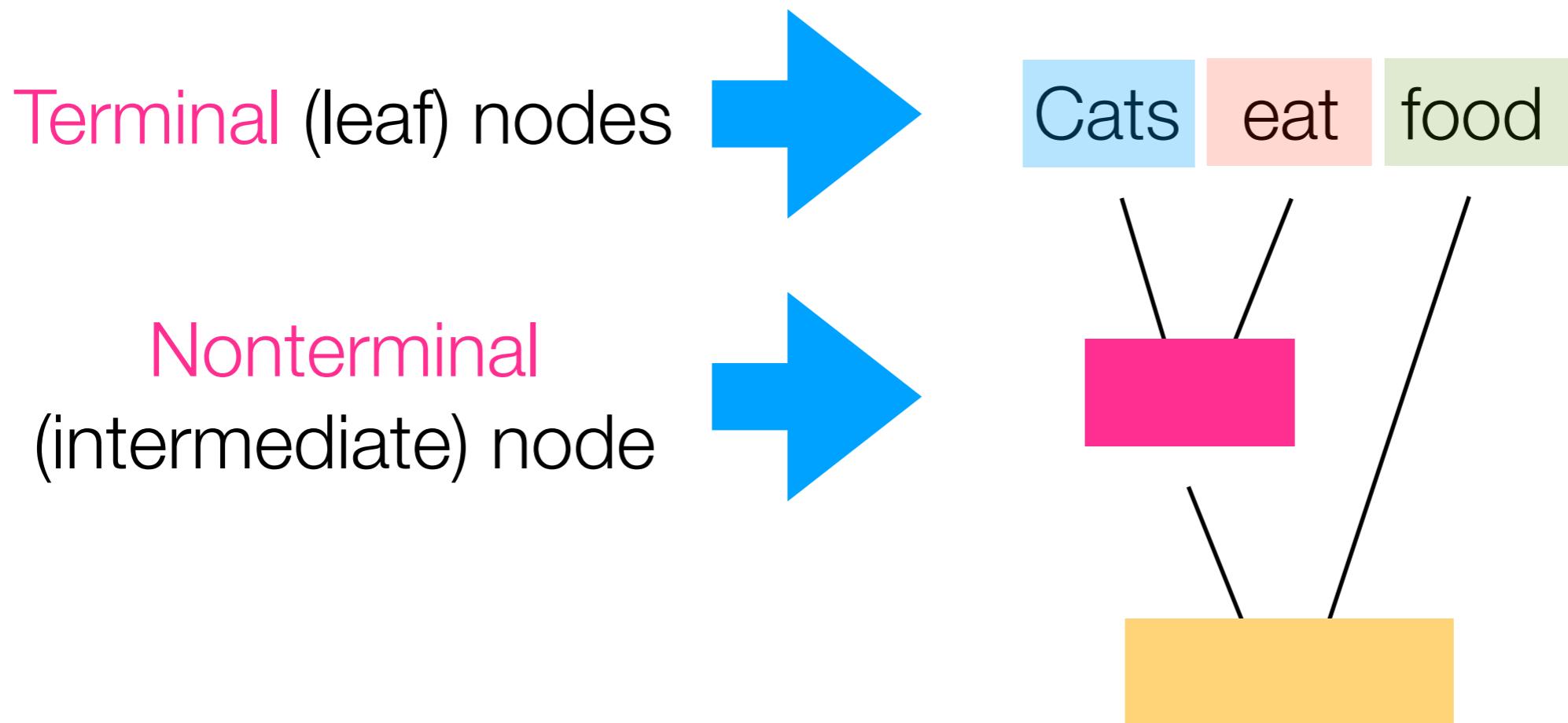
How do we describe this structure?



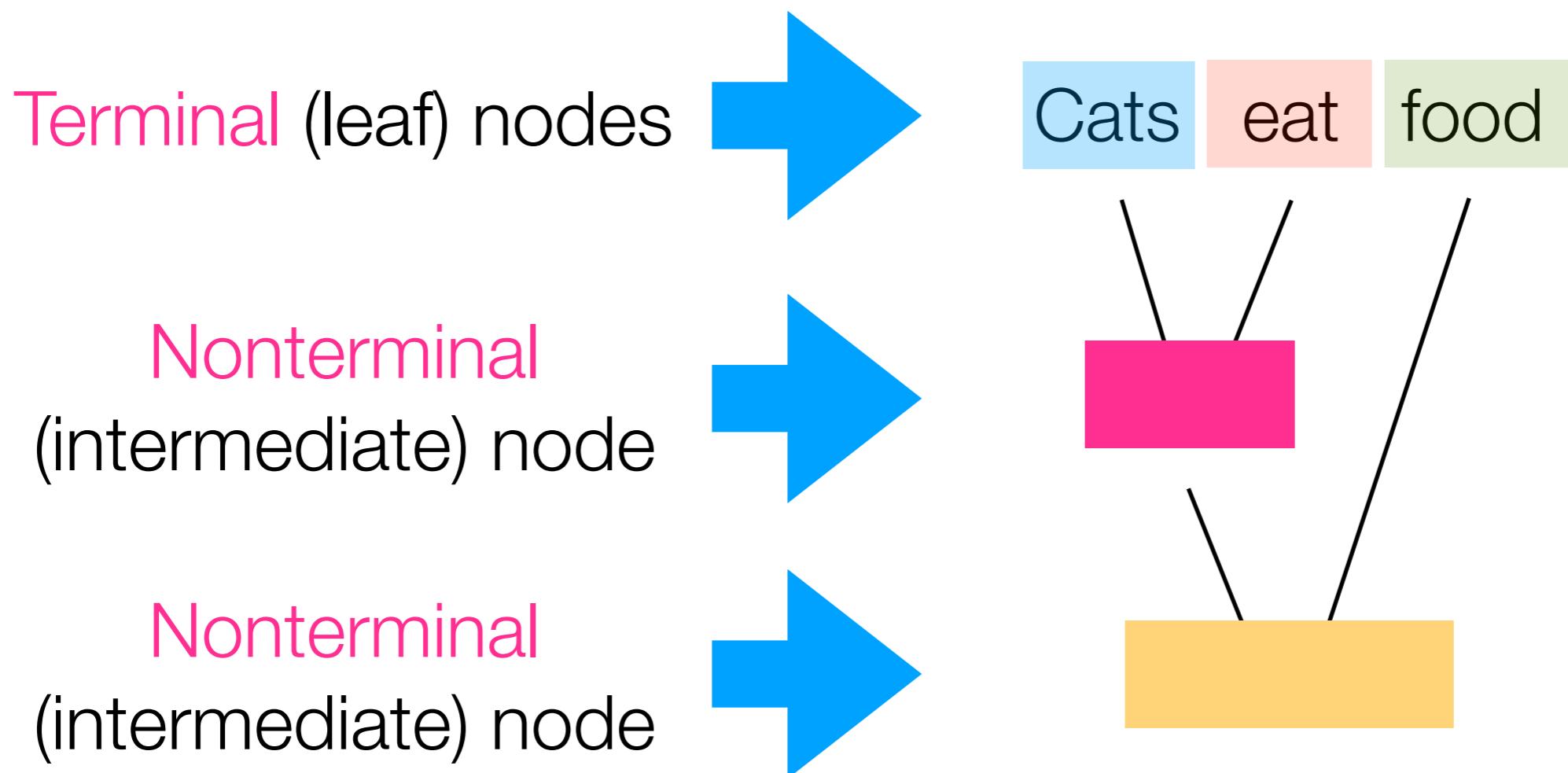
How do we describe this structure?



How do we describe this structure?



How do we describe this structure?



Context-free grammar

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

Context-free grammar

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

Noun	→	flight
------	---	--------

Context-free grammar

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

Each row defines a rule from one **non-terminal** to either a terminal or another non-terminal

Noun	→	flight
------	---	--------

Context-free grammar

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

Each row defines a rule from one **non-terminal** to either a terminal or another non-terminal

Noun

→

flight

| separates options

Context-free grammar

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

Each row defines a rule from one **non-terminal** to either a terminal or another non-terminal

Noun	→	flight
Det	→	a the

| separates options

Context-free grammar

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

Each row defines a rule from one **non-terminal** to either a terminal or another non-terminal

Noun	→	flight
Det	→	a the

lexicon/
terminals

Context-free grammar

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

Each row defines a rule from one **non-terminal** to either a terminal or another non-terminal

Noun	→	flight
Det	→	a the
NP	→	Det Nominal

lexicon/
terminals

Context-free grammar

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

Each row defines a rule from one **non-terminal** to either a terminal or another non-terminal

Noun	→	flight
Det	→	a the separates options
NP	→	Det Nominal
NP	→	Proper_Noun

lexicon/
terminals

Context-free grammar

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

Each row defines a rule from one **non-terminal** to either a terminal or another non-terminal

Noun	→	flight
Det	→	a the
NP	→	Det Nominal
NP	→	Proper_Noun
Nominal	→	Noun Nominal Noun

lexicon/
terminals

Context-free grammar

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

Each row defines a rule from one **non-terminal** to either a terminal or another non-terminal

Noun	→	flight
Det	→	a the
NP	→	Det Nominal
NP	→	Proper_Noun
Nominal	→	Noun Nominal Noun
S	→	NP

lexicon/
terminals

Context-free grammar

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

Each row defines a rule from one **non-terminal** to either a terminal or another non-terminal

Noun	→	flight
Det	→	a the
NP	→	Det Nominal
NP	→	Proper_Noun
Nominal	→	Noun Nominal Noun
S	→	NP

lexicon/
terminals

non-terminals
can produce
more non-
terminals

Context-free grammar

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

Each row defines a rule from one **non-terminal** to either a terminal or another non-terminal

Noun	→	flight
Det	→	a the
NP	→	Det Nominal
NP	→	Proper_Noun
Nominal	→	Noun Nominal Noun
S	→	NP

lexicon/
terminals

non-terminals
can produce
more non-
terminals

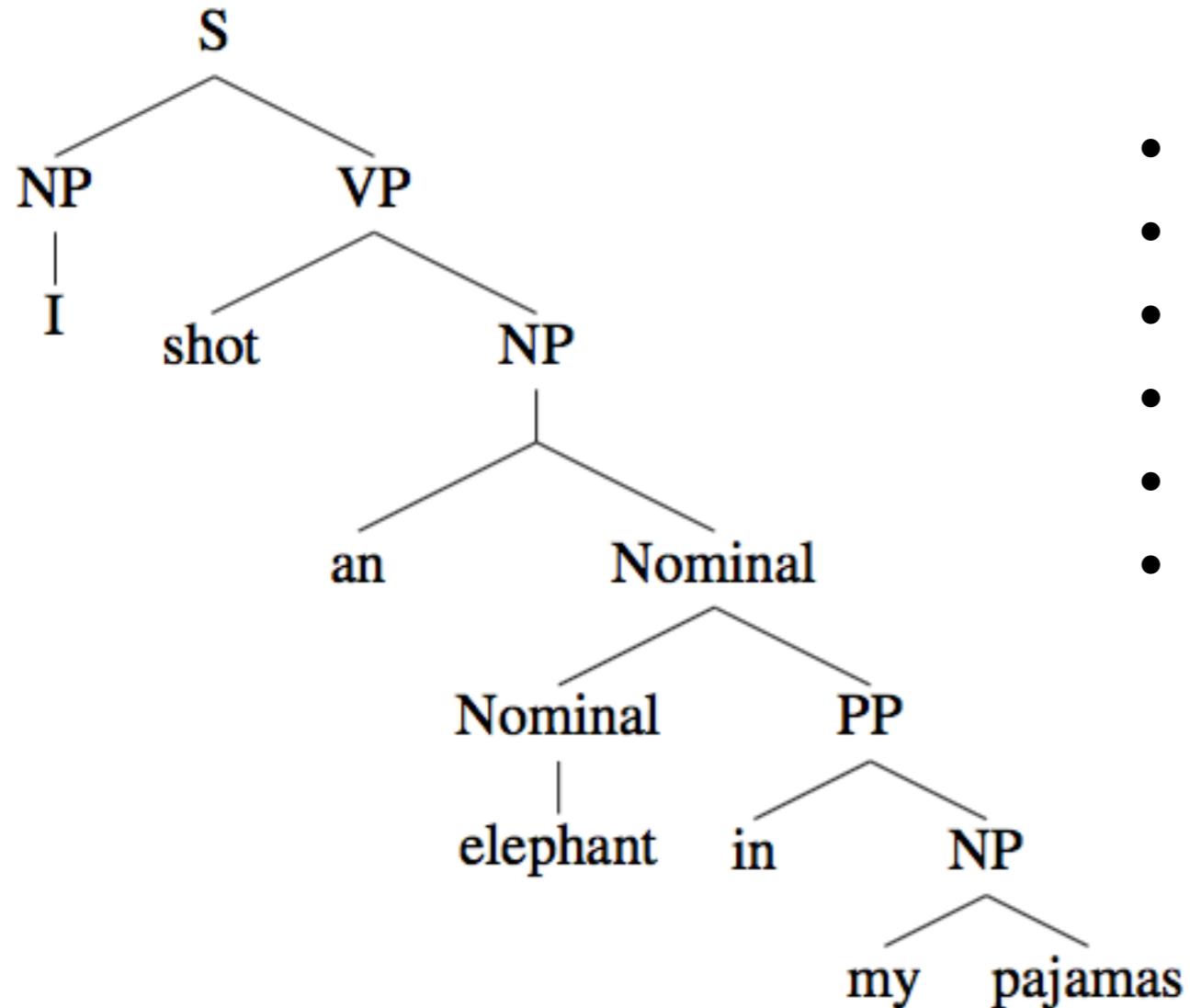
“S” is a special non-terminal that always forms
the root of the tree.

Generating sentences top-down using a grammar

Starting from “S”, what sentences can we make?

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

Constituents



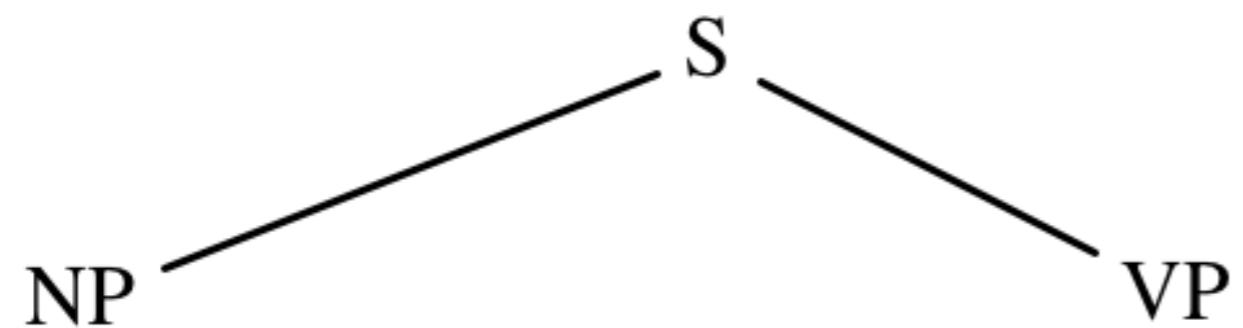
Every internal node is a phrase

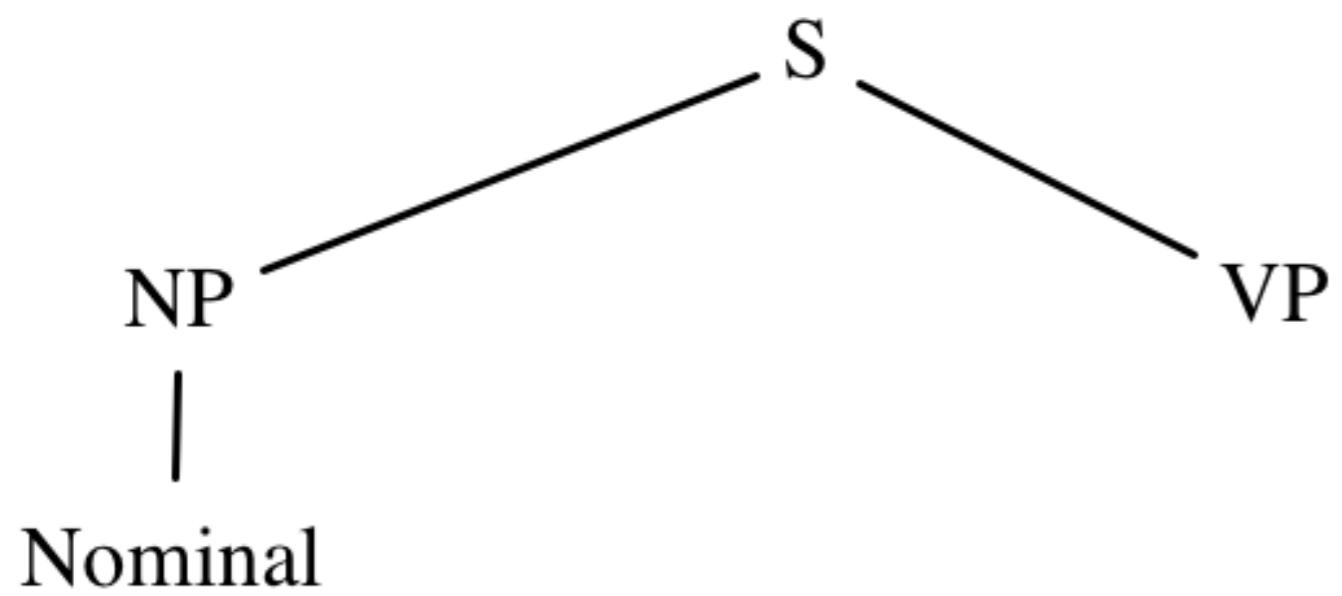
- my pajamas
- in my pajamas
- elephant in my pajamas
- an elephant in my pajamas
- shot an elephant in my pajamas
- I shot an elephant in my pajamas

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

How might we measure
how likely a parse tree is?

S

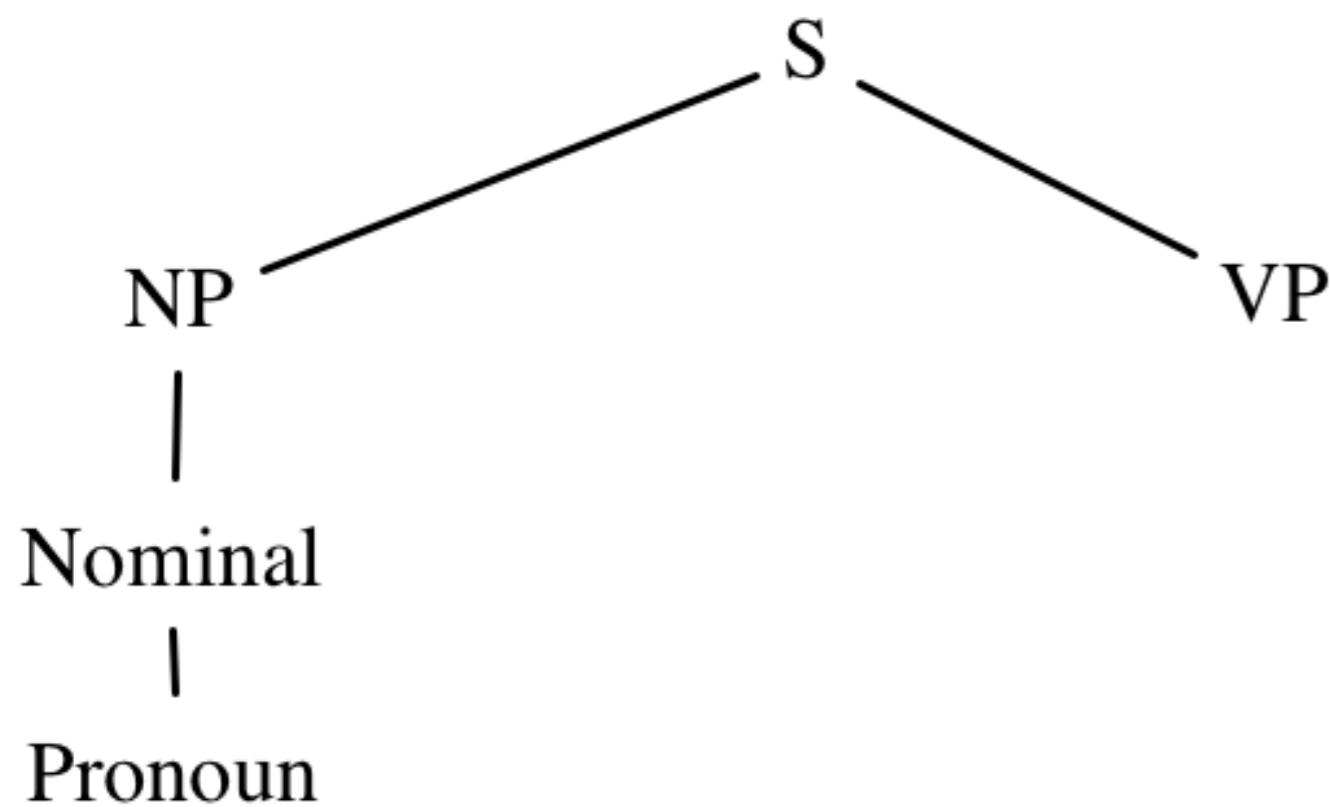
$P(\text{NP } \text{VP} \mid S)$ 

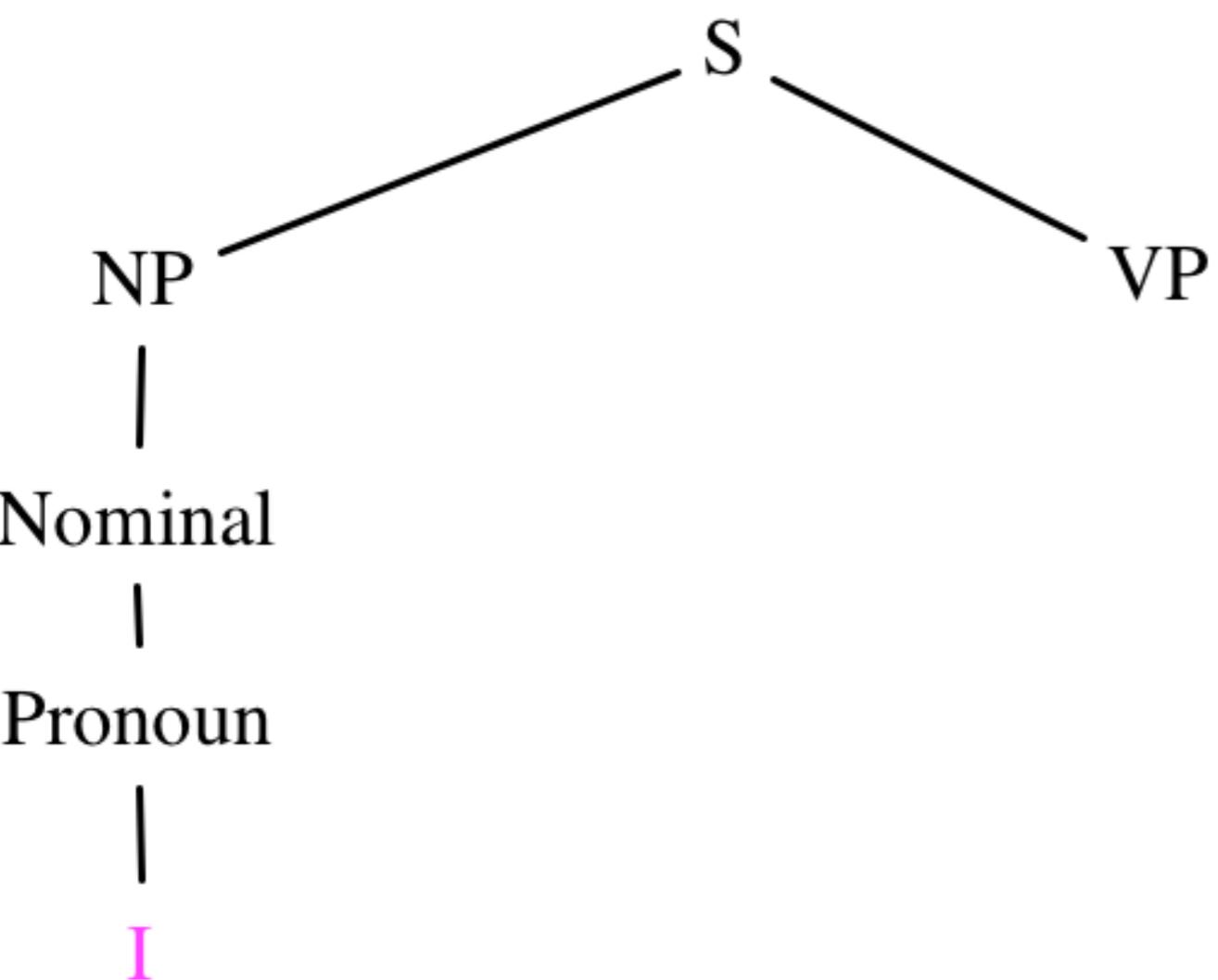
$P(\text{NP VP} \mid S)$ $\times P(\text{Nominal} \mid \text{NP})$ 

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

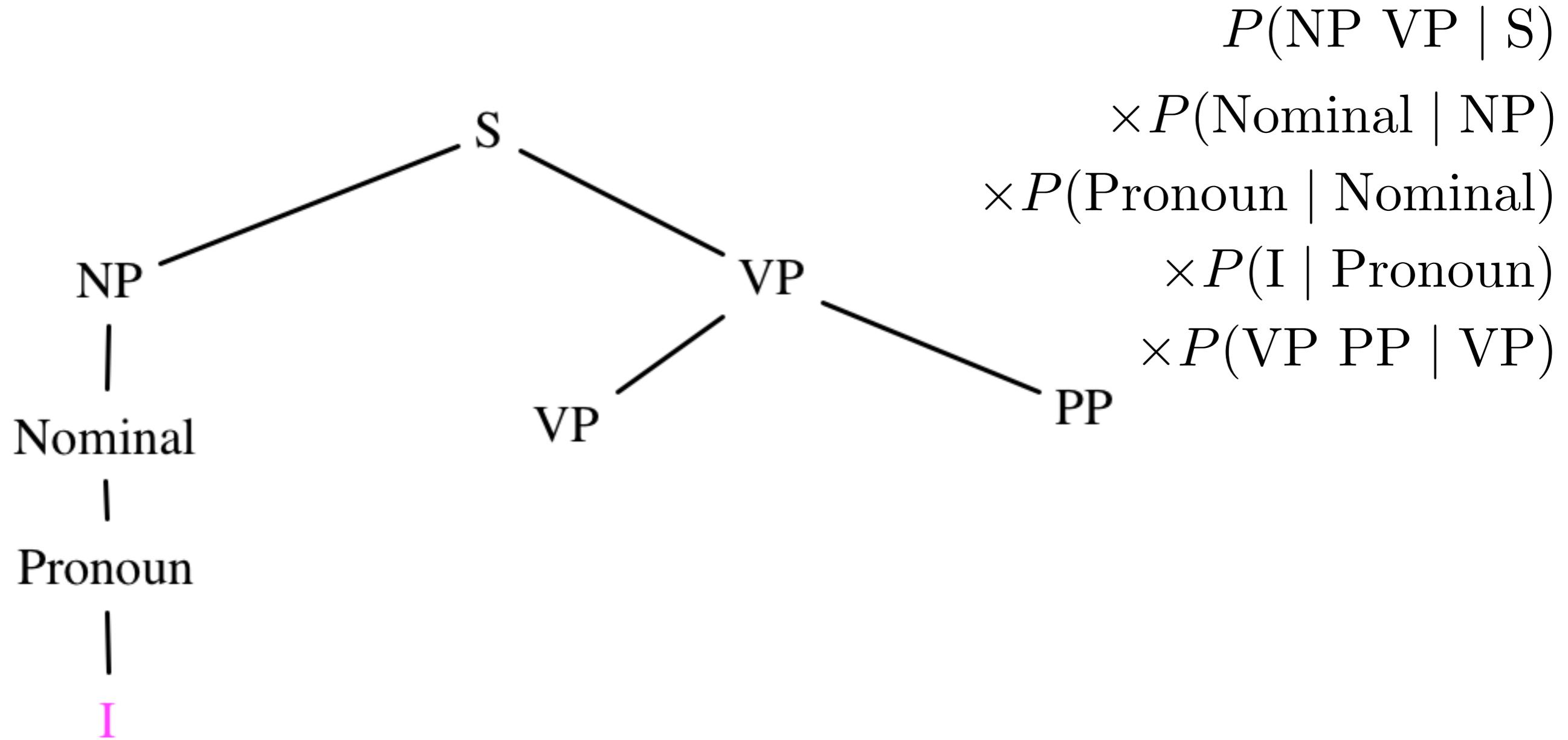
$$\times P(\text{Nominal} \mid \text{NP})$$

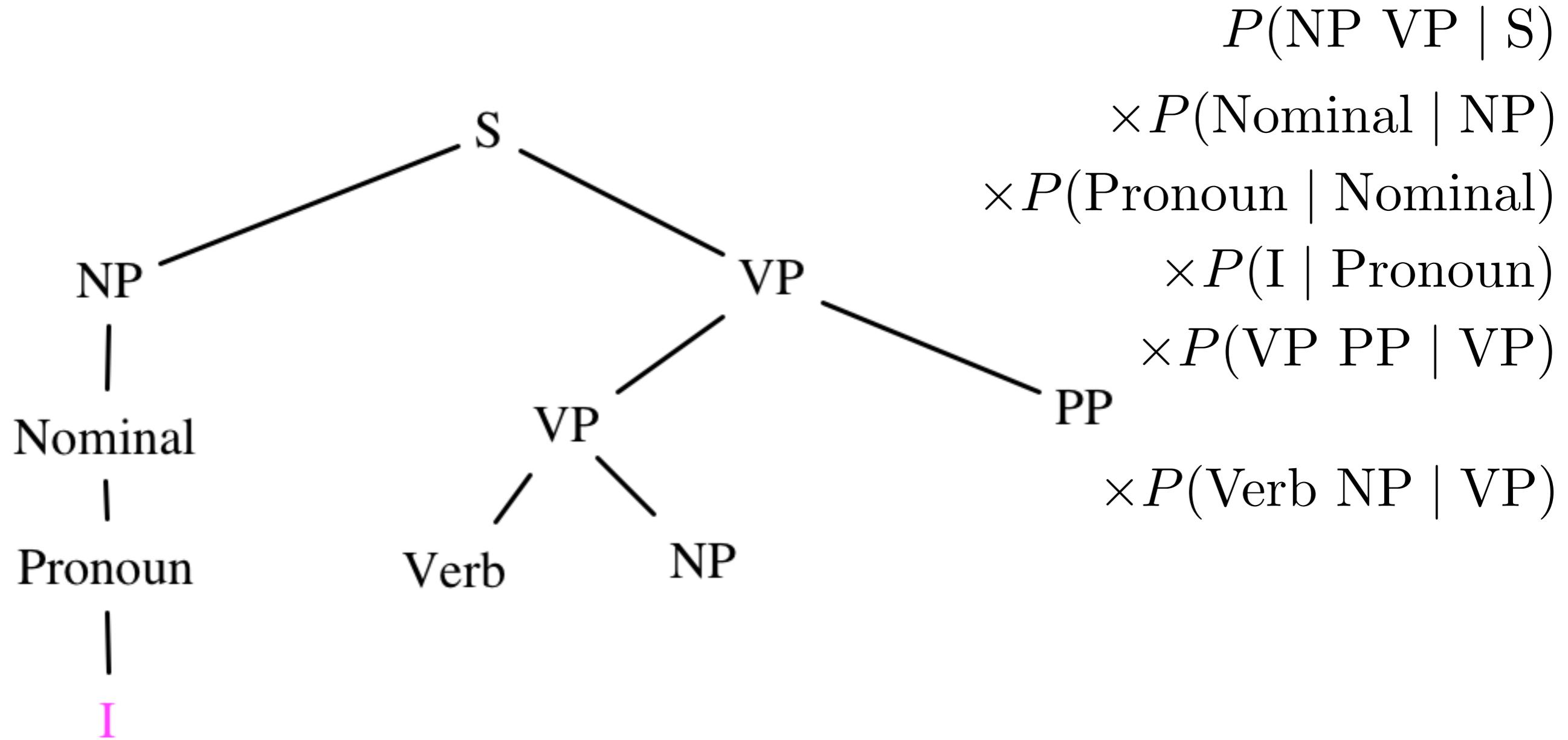
$$\times P(\text{Pronoun} \mid \text{Nominal})$$

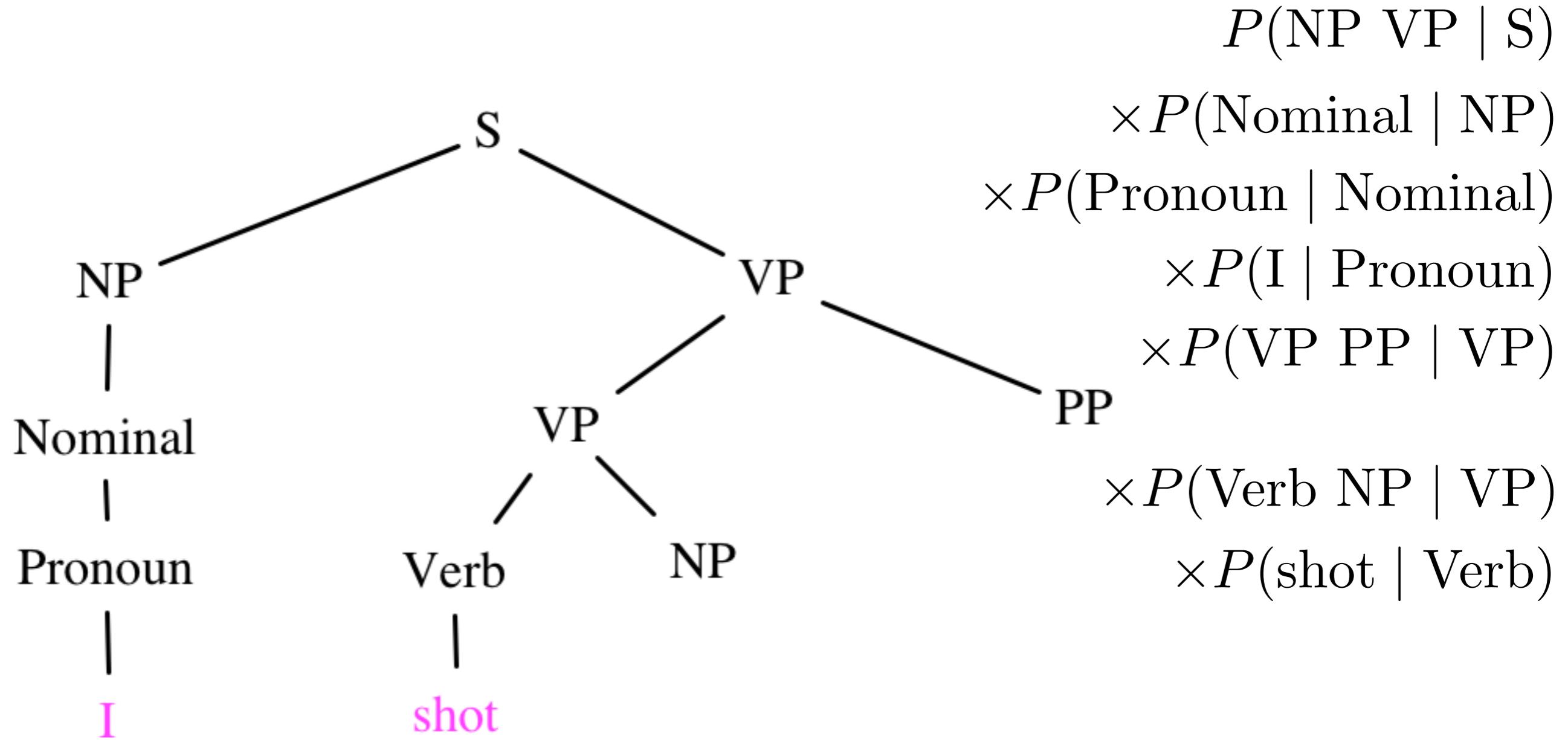


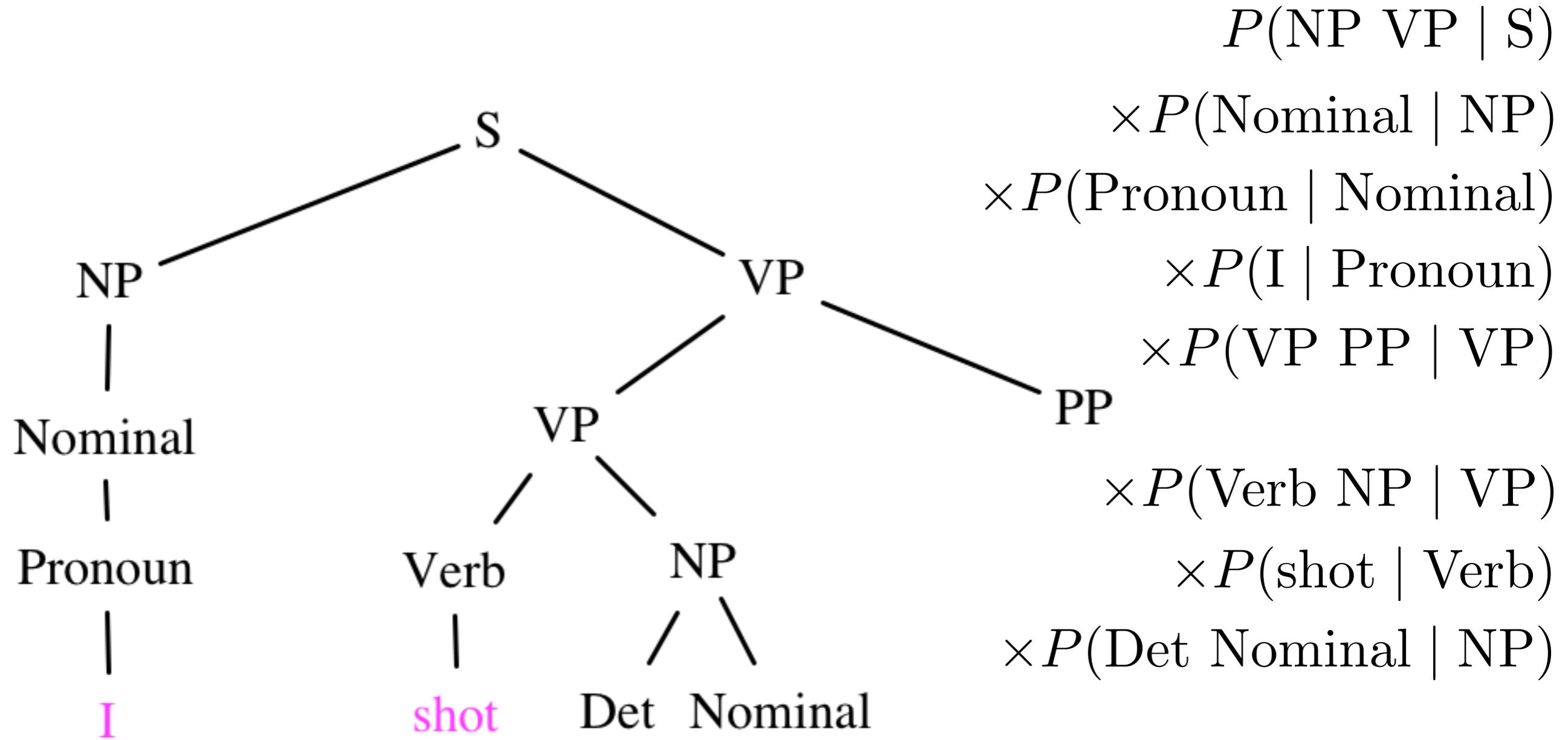


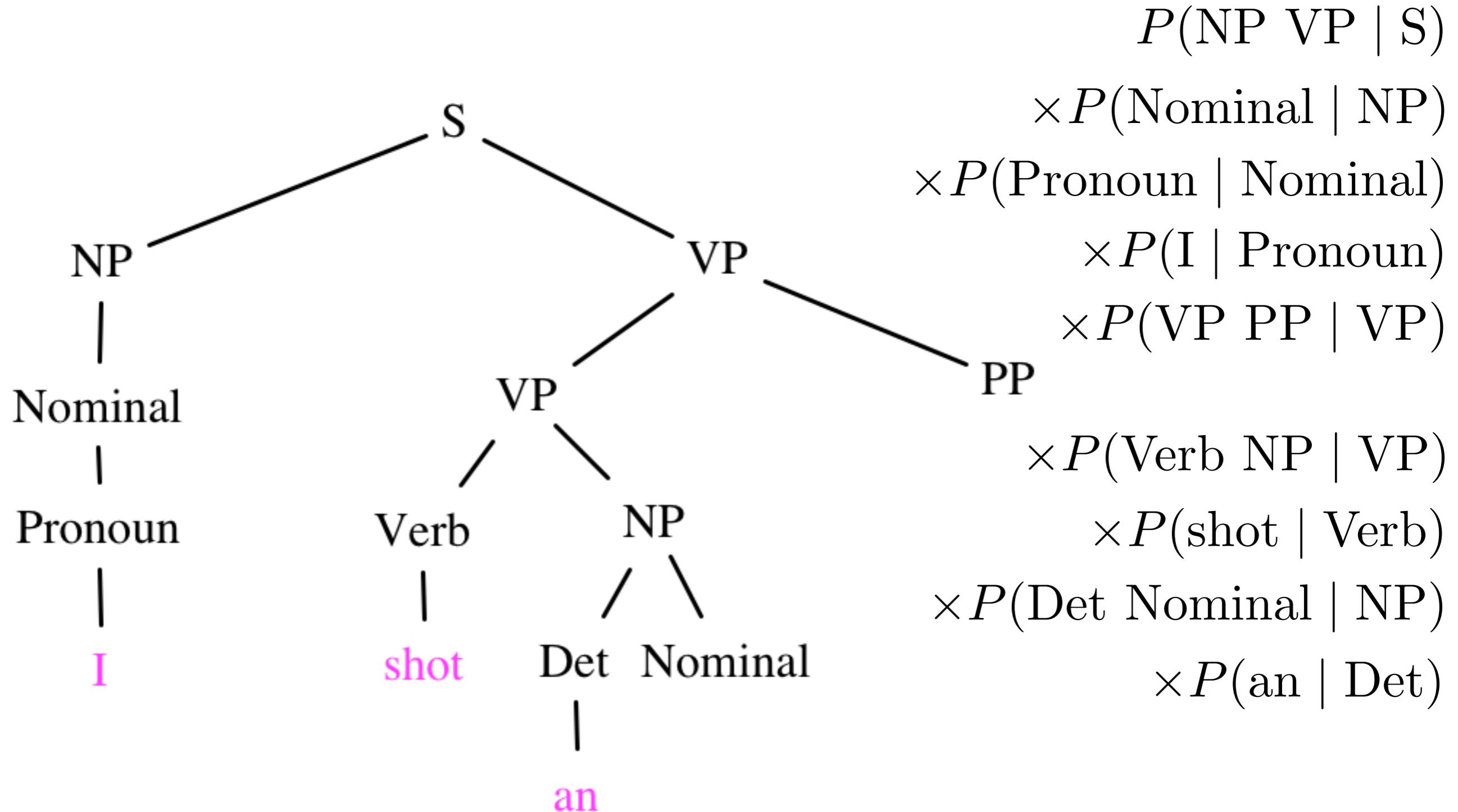
$$\begin{aligned} & P(\text{NP VP} \mid \text{S}) \\ & \times P(\text{Nominal} \mid \text{NP}) \\ & \times P(\text{Pronoun} \mid \text{Nominal}) \\ & \times P(\text{I} \mid \text{Pronoun}) \end{aligned}$$

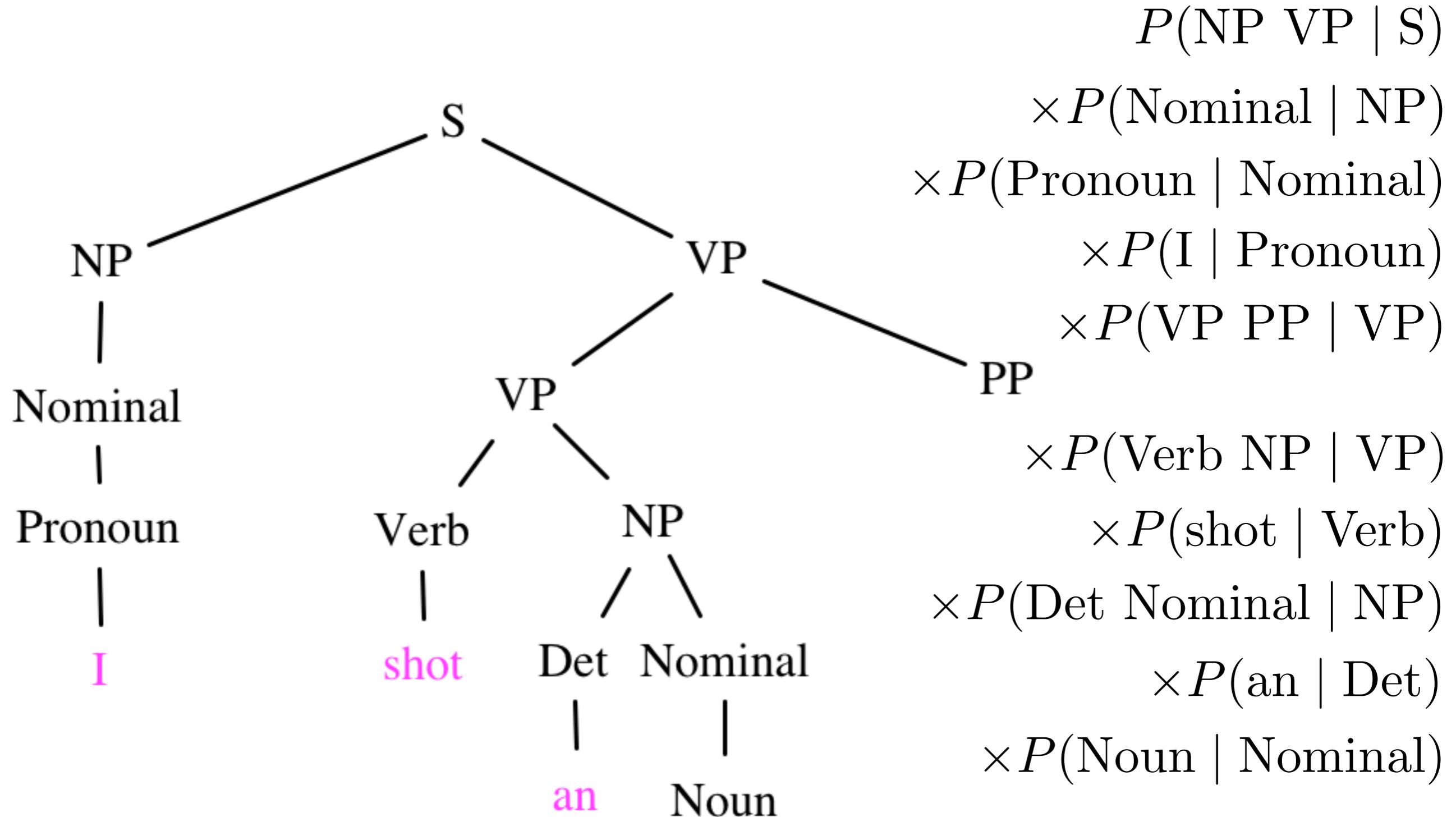


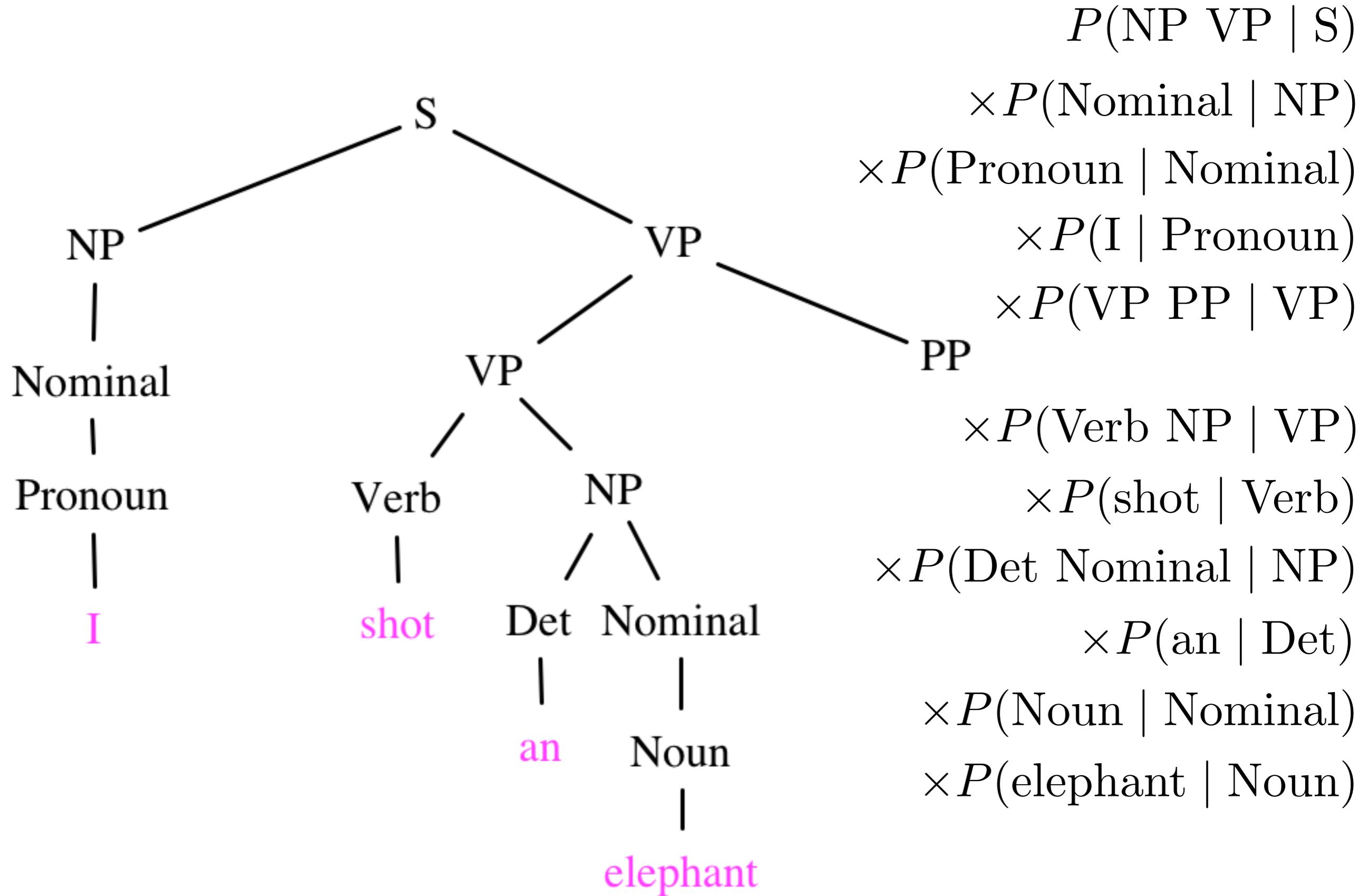


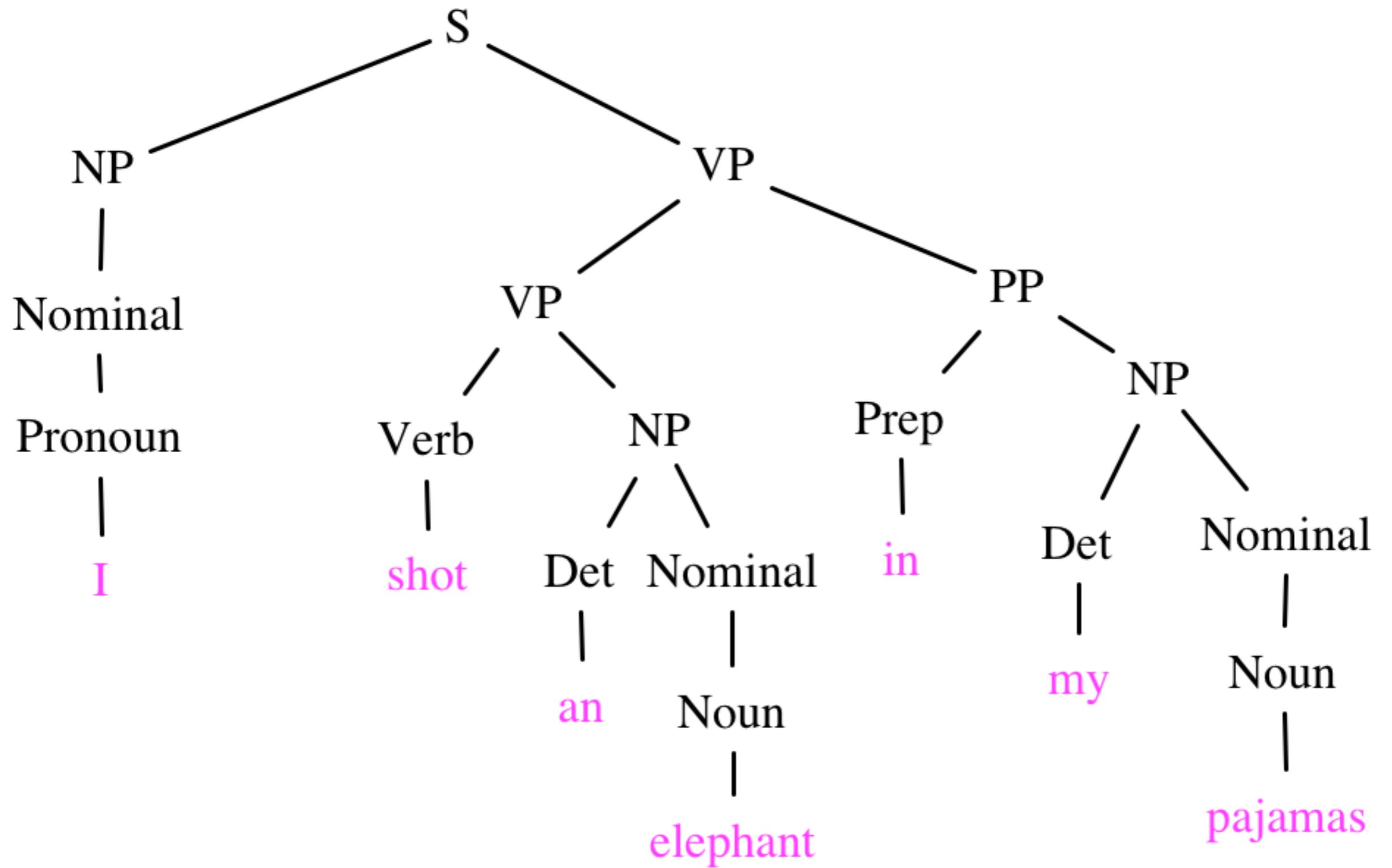












Probabilistic context-free grammar: PCFG

Probabilistic context-free grammar: PCFG

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

Probabilistic context-free grammar: 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 $A \rightarrow \beta$):

Probabilistic context-free grammar: 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 $A \rightarrow \beta$):

$$P(T, S) = \prod_i^n P(\beta | A)$$

Probabilistic context-free grammar: 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 $A \rightarrow \beta$):

$$P(T, S) = \prod_i^n P(\beta | A)$$

How probable is it that non-terminal A produced β ?

Probabilistic context-free grammar: 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 $A \rightarrow \beta$):

$$P(T, S) = \prod_i^n P(\beta | A)$$

How probable is it that non-terminal A produced β ?

How probable is the whole parse?

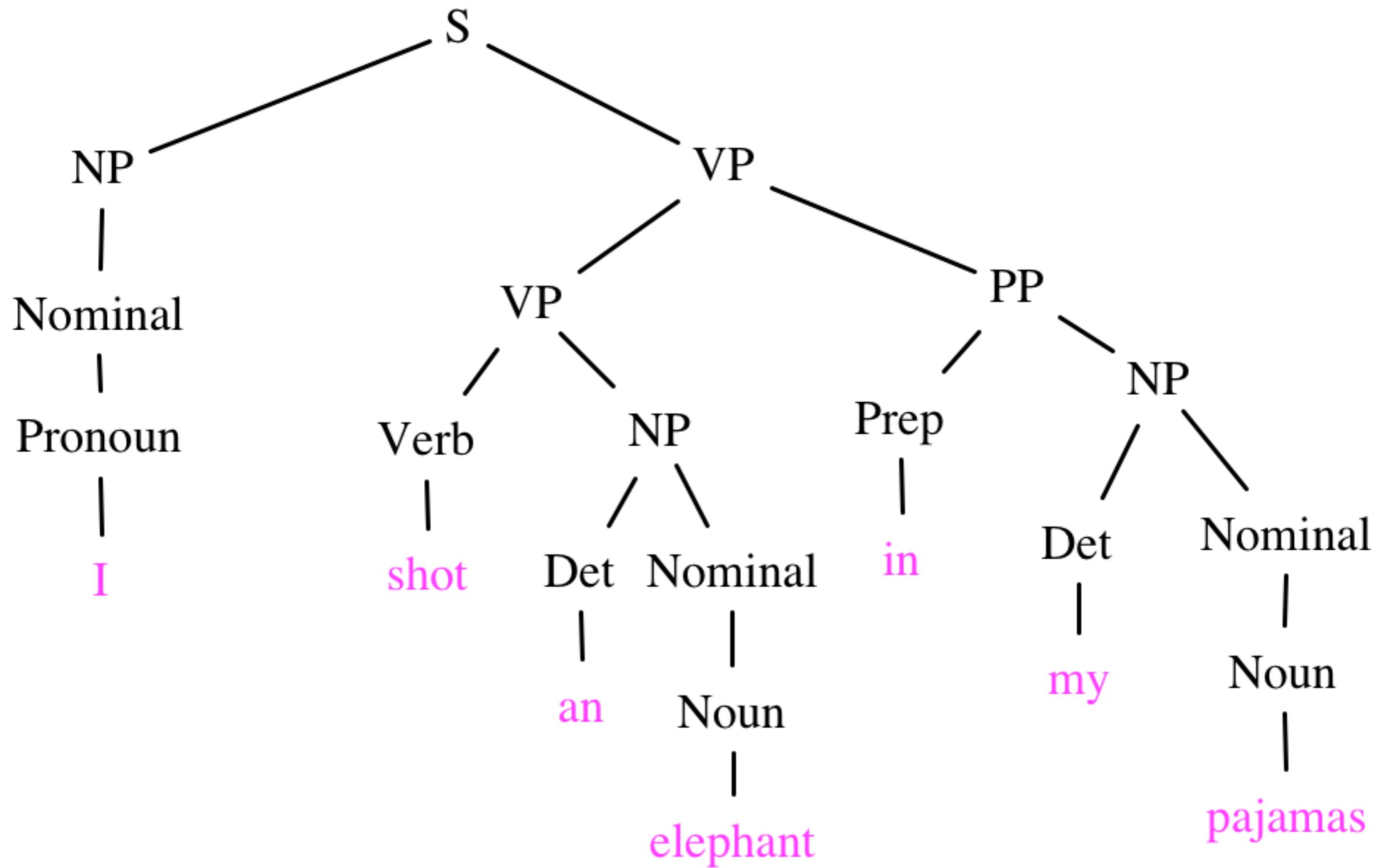
PCFG

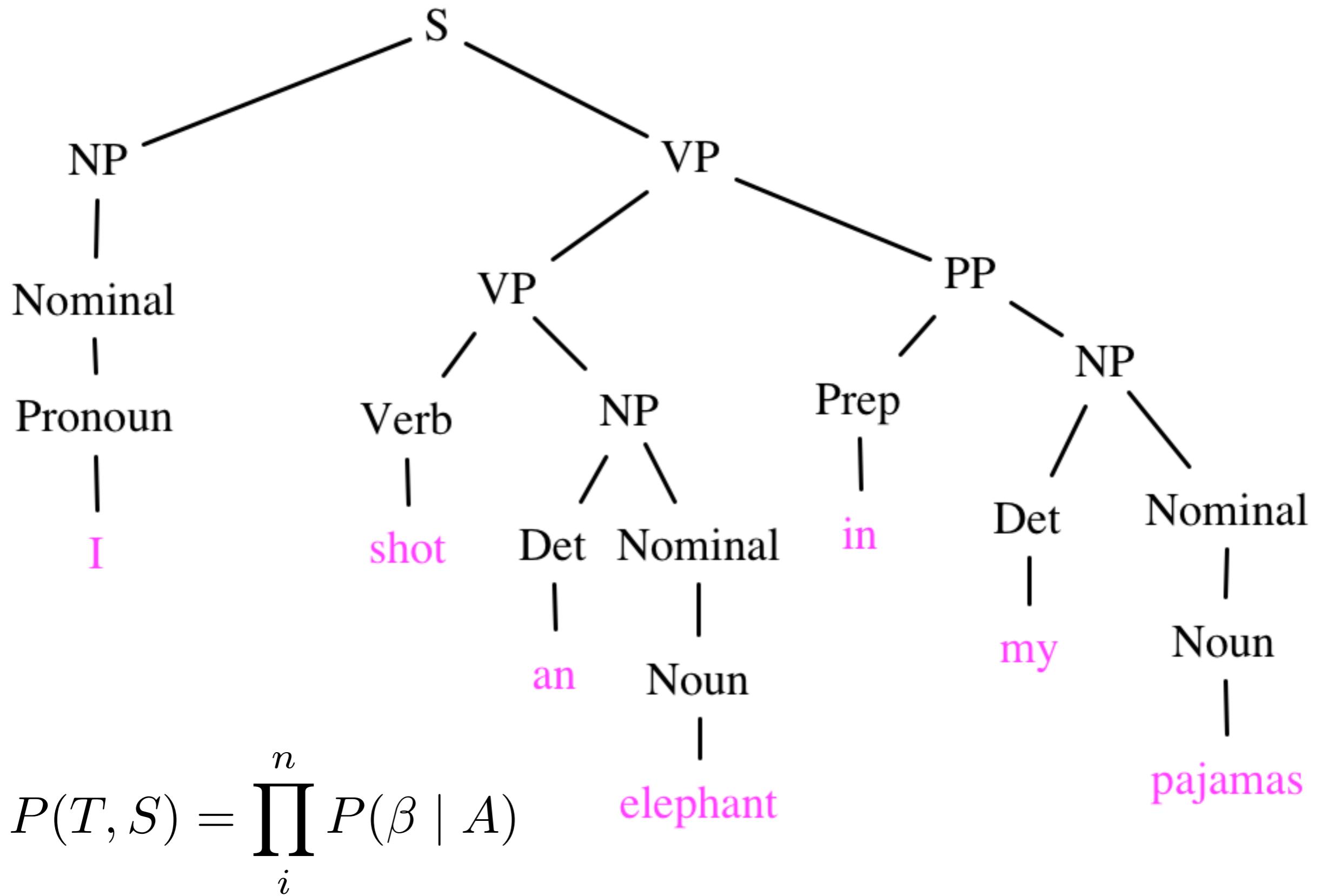
N Finite set of non-terminal symbols NP, VP, S

Σ Finite alphabet of terminal symbols the, dog, a

R Set of production rules, each
 $A \rightarrow \beta [p]$
 $p = P(\beta | A)$ $S \rightarrow NP\ VP$
 Noun \rightarrow dog

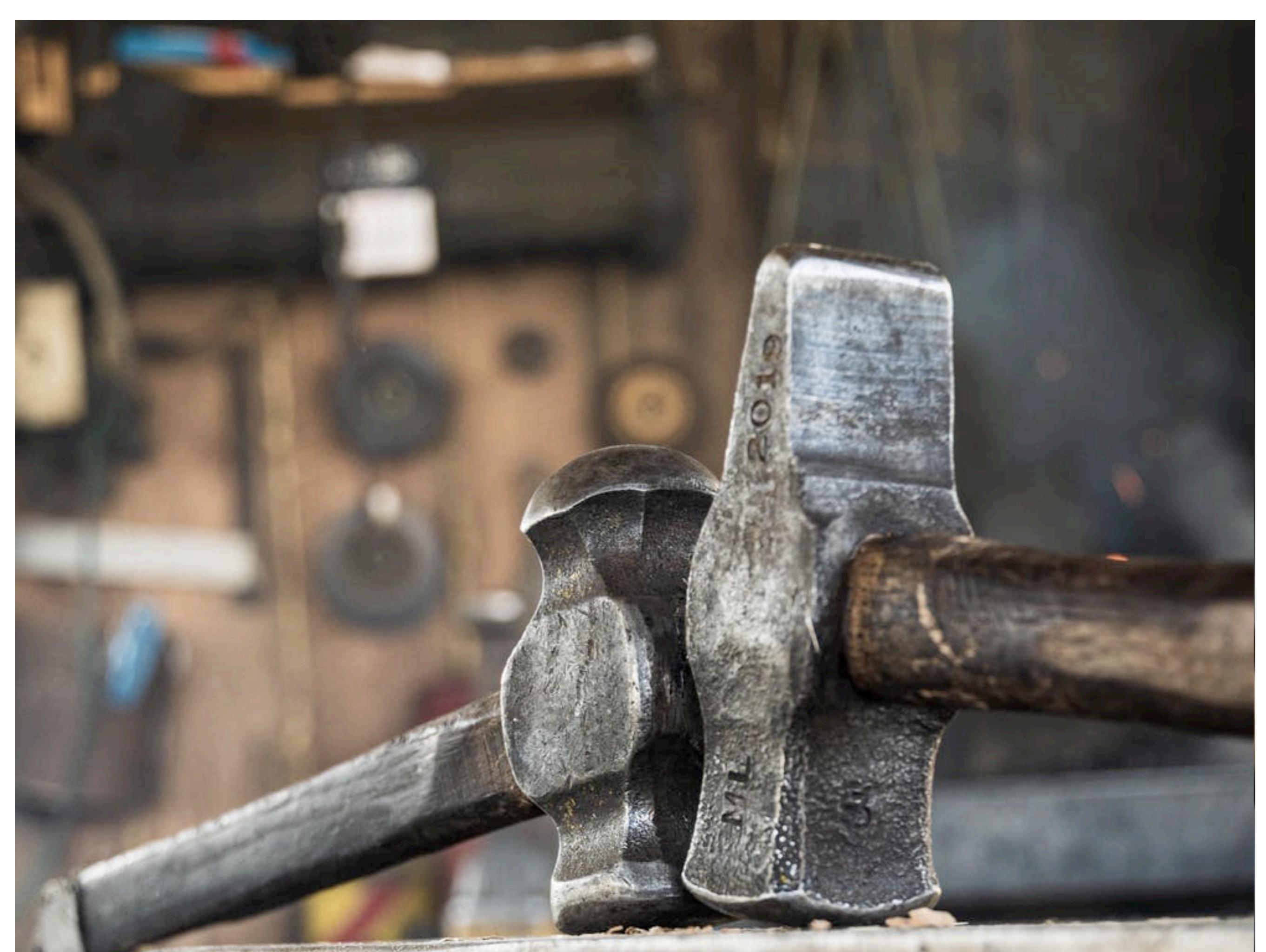
S Start symbol





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
 $A \rightarrow \beta$
 $\beta \in (\Sigma, N)$ NP \rightarrow DT JJ NN
 Noun \rightarrow dog

S Start symbol

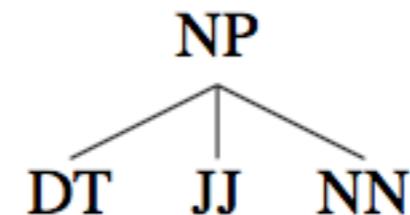
Chomsky Normal Form (CNF)

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 $A \rightarrow \beta$ $\beta = \text{single terminal (from } \Sigma \text{) or two non-terminals (from } N\text{)}$	$S \rightarrow NP VP$ Noun \rightarrow dog
S	Start symbol	

Chomsky Normal Form (CNF)

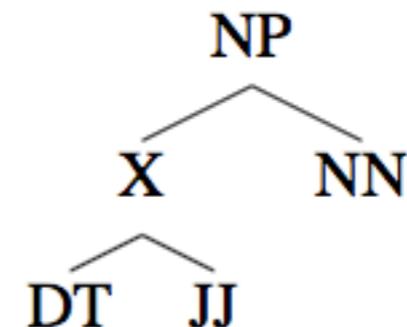
- Any CFG can be converted into weakly equivalent CNF (recognizing the same set of sentences as existing in the grammar but differing in their derivation).

$NP \rightarrow DT\ JJ\ NN$



$NP \rightarrow X\ NN$

$X \rightarrow DT\ JJ$



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

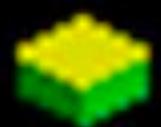
S → NP VP
VP → VBD NP
VP → VP PP
Nominal → Nominal PP
Nominal → pajamas elephant I
PP → IN NP
NP → DT NN
NP → pajamas elephant I
NP → PRP\$ Nominal

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

I shot an elephant in my pajamas

3400

CHANGE TO:



LEVEL: 1
ROUND: 2



BONUS 2600

CKY

CKY

- Cocke-Kasami-Younger algorithm (also CYK) for parsing from a grammar expressed in CNF.

CKY

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

CKY

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

CKY

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

CKY

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

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

I shot an elephant in my pajamas

0

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]

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]		
What phrases can be formed from "I shot"					PRP\$ [5,6]	
						NNS [6,7]

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

NP, PRP [0,1]						
	VBD [1,2]					
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
What phrases can be formed from “shot an elephant in”					PRP\$ [5,6]	
						NNS [6,7]

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]	
What phrases can be formed from “elephant in my”					NNS [6,7]	

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]
What phrases can be formed from “I shot an elephant in my pajamas”						

Chomsky Normal Form (CNF)

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

$$S \rightarrow NP\ VP$$

[s [NP I] [VP shot an elephant in my pajamas]]

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

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

S	\rightarrow	NP VP
VP	\rightarrow	VBD NP
VP	\rightarrow	VP PP
Nomin	\rightarrow	Nominal PP
Nomin	\rightarrow	pajamas elephant
PP	\rightarrow	IN NP
NP	\rightarrow	DT NN
NP	\rightarrow	pajamas elephant
NP	\rightarrow	PRP\$ Nominal
VBD	\rightarrow	shot
DT	\rightarrow	an my
PRP	\rightarrow	I
PRP\$	\rightarrow	my
IN	\rightarrow	in

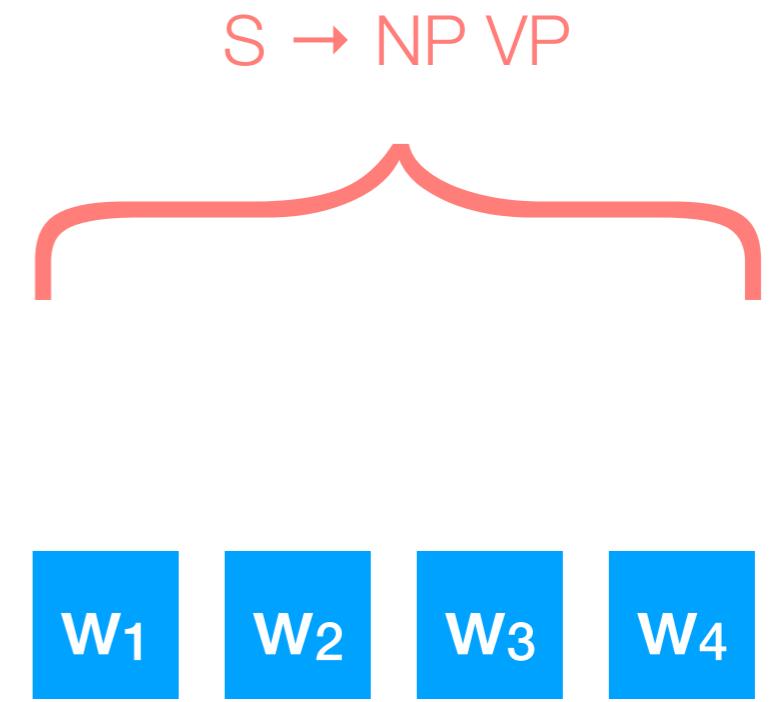
If the left-side non-terminal spans tokens i - j , the right side must also span i - j , and there must be a single position k that separates them.

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in



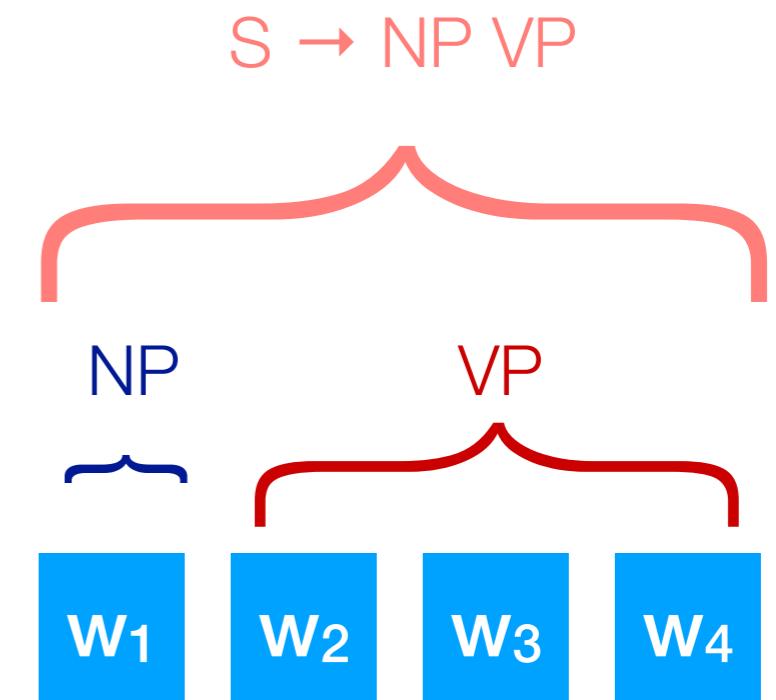
If the left-side non-terminal spans tokens $i-j$, the right side must also span $i-j$, and there must be a single position k that separates them.

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in



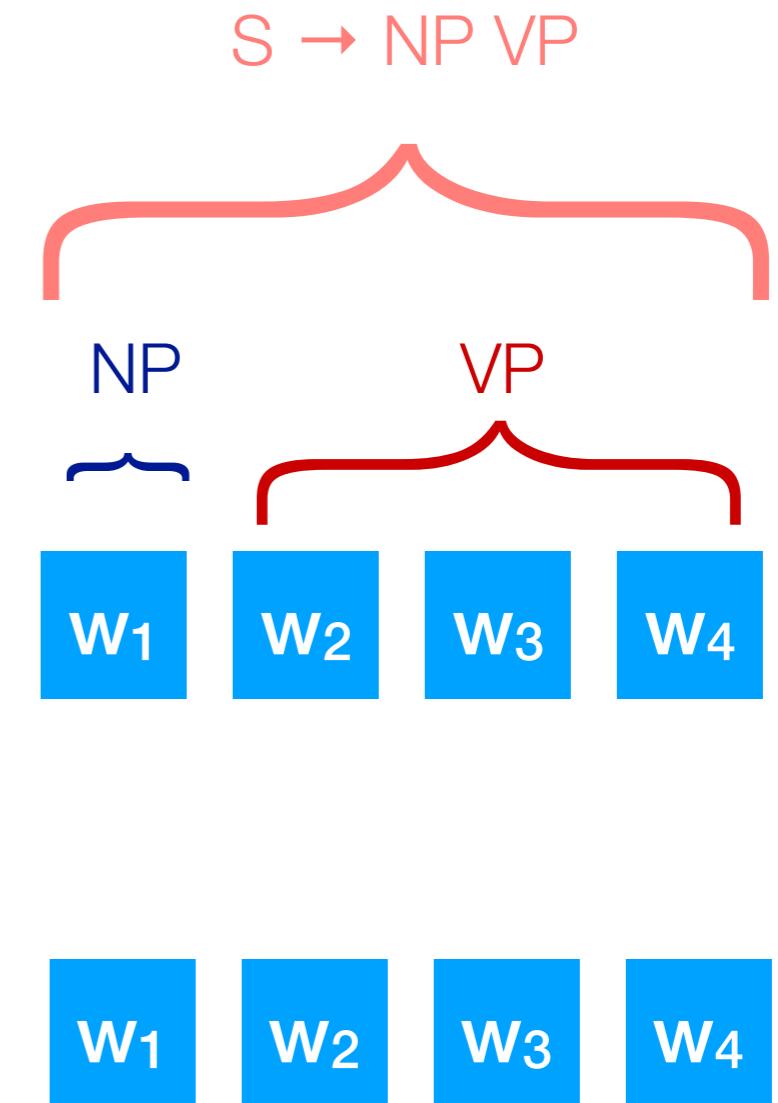
If the left-side non-terminal spans tokens i - j , the right side must also span i - j , and there must be a single position k that separates them.

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in



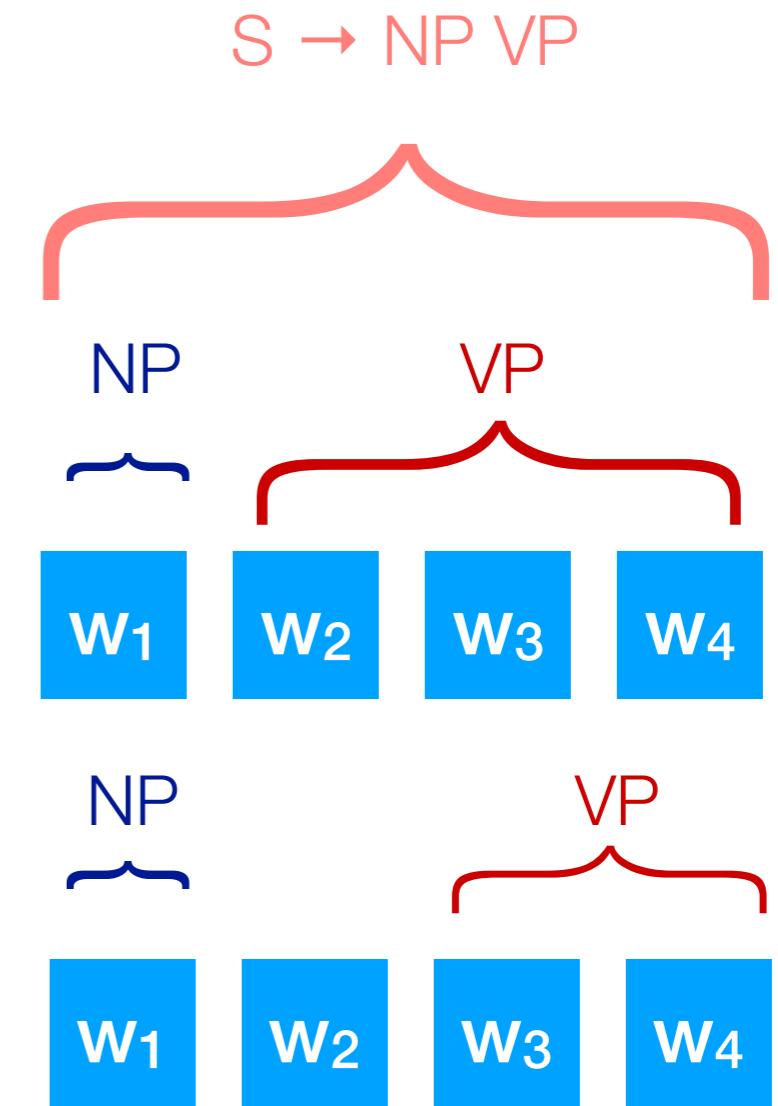
If the left-side non-terminal spans tokens i - j , the right side must also span i - j , and there must be a single position k that separates them.

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in



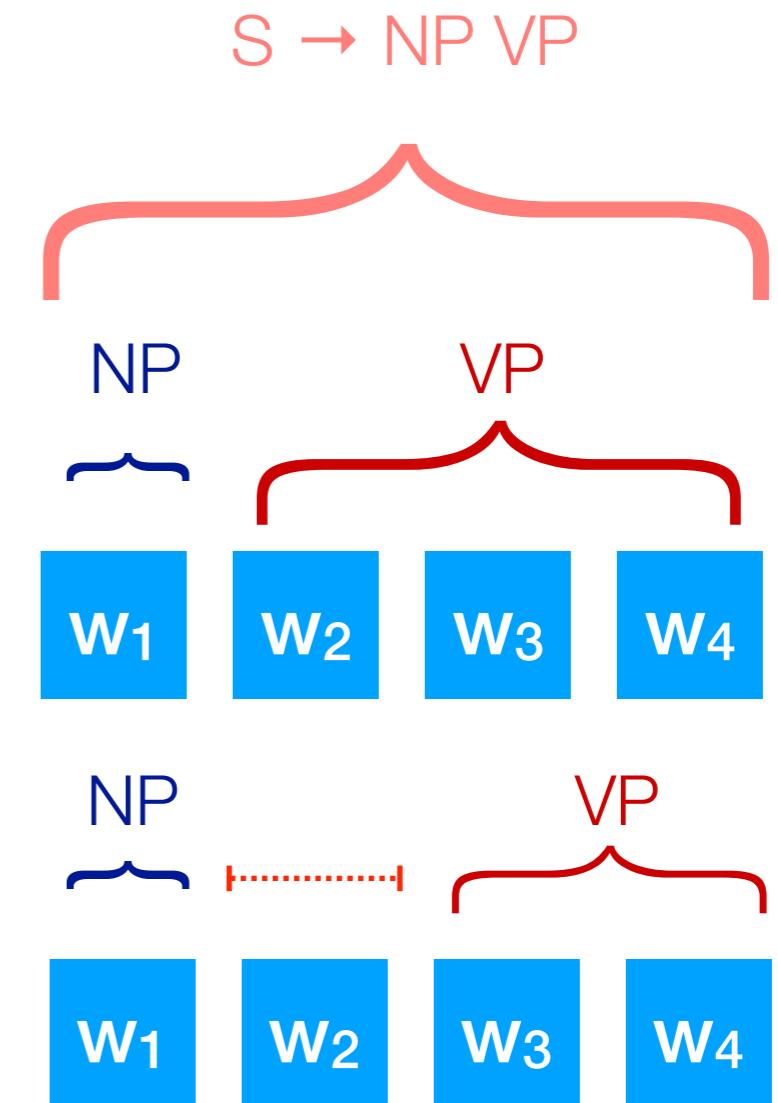
If the left-side non-terminal spans tokens i - j , the right side must also span i - j , and there must be a single position k that separates them.

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in



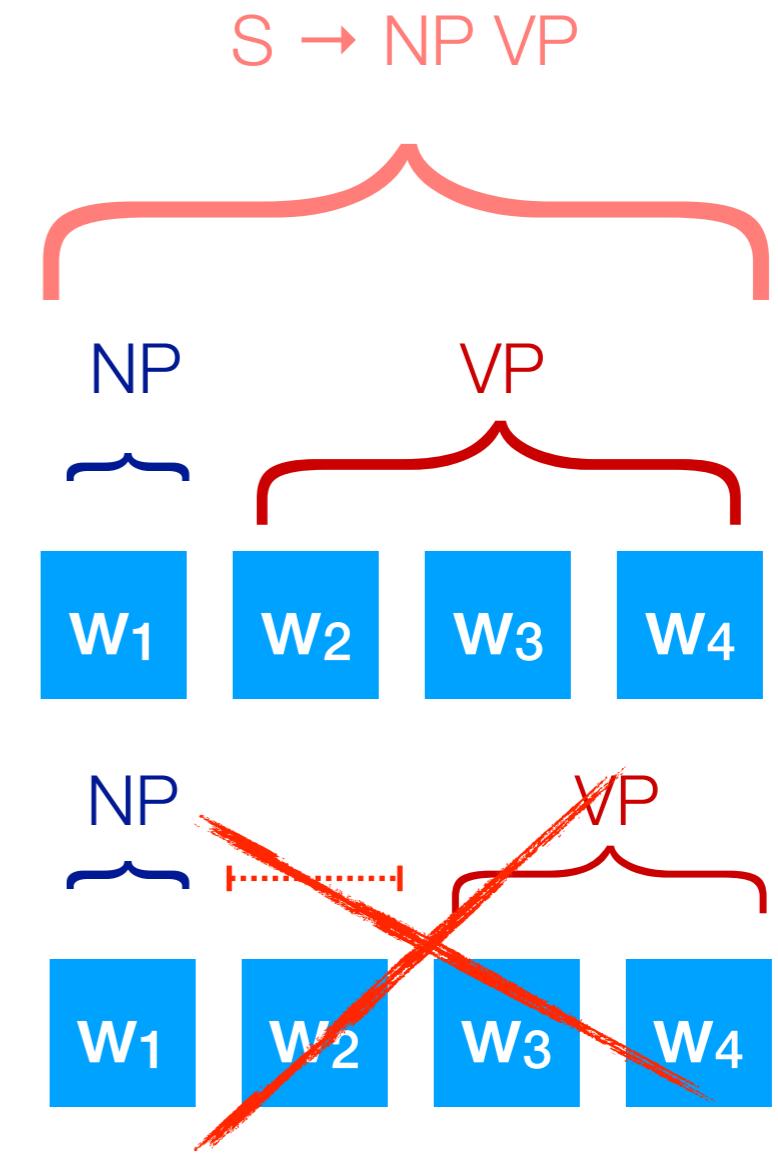
If the left-side non-terminal spans tokens i - j , the right side must also span i - j , and there must be a single position k that separates them.

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in



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

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in



Bottom-up Parsing

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in

Bottom-up Parsing

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in

W₁

W₂

W₃

W₄

Bottom-up Parsing

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in

w₁

|

w₂

shot

w₃

my

w₄

pajamas

Bottom-up Parsing

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in

W₁ W₂ W₃ W₄

| shot my pajamas

Can we map
any word
immediately
to a rule?

Bottom-up Parsing

S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nomin	→	Nominal PP
Nomin	→	pajamas elephant
PP	→	IN NP
NP	→	DT NN
NP	→	pajamas elephant
NP	→	PRP\$ Nominal
VBD	→	shot
DT	→	an my
PRP	→	I
PRP\$	→	my
IN	→	in

Nominal, NP,

PRP



W₁

W₂

W₃

W₄

| shot my pajamas

Can we map
any word
immediately
to a rule?

Bottom-up Parsing

S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nomin	→	Nominal PP
Nomin	→	pajamas elephant
PP	→	IN NP
NP	→	DT NN
NP	→	pajamas elephant
NP	→	PRP\$ Nominal
VBD	→	shot
DT	→	an my
PRP	→	I
PRP\$	→	my
IN	→	in



Can we map
any word
immediately
to a rule?

Bottom-up Parsing

S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nomin	→	Nominal PP
Nomin	→	pajamas elephant
PP	→	IN NP
NP	→	DT NN
NP	→	pajamas elephant
NP	→	PRP\$ Nominal
VBD	→	shot
DT	→	an my
PRP	→	I
PRP\$	→	my
IN	→	in



Can we map
any word
immediately
to a rule?

Bottom-up Parsing

S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nomin	→	Nominal PP
Nomin	→	pajamas elephant
PP	→	IN NP
NP	→	DT NN
NP	→	pajamas elephant
NP	→	PRP\$ Nominal
VBD	→	shot
DT	→	an my
PRP	→	I
PRP\$	→	my
IN	→	in



Bottom-up Parsing

S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nomin	→	Nominal PP
Nomin	→	pajamas elephant
PP	→	IN NP
NP	→	DT NN
NP	→	pajamas elephant
NP	→	PRP\$ Nominal
VBD	→	shot
DT	→	an my
PRP	→	I
PRP\$	→	my
IN	→	in

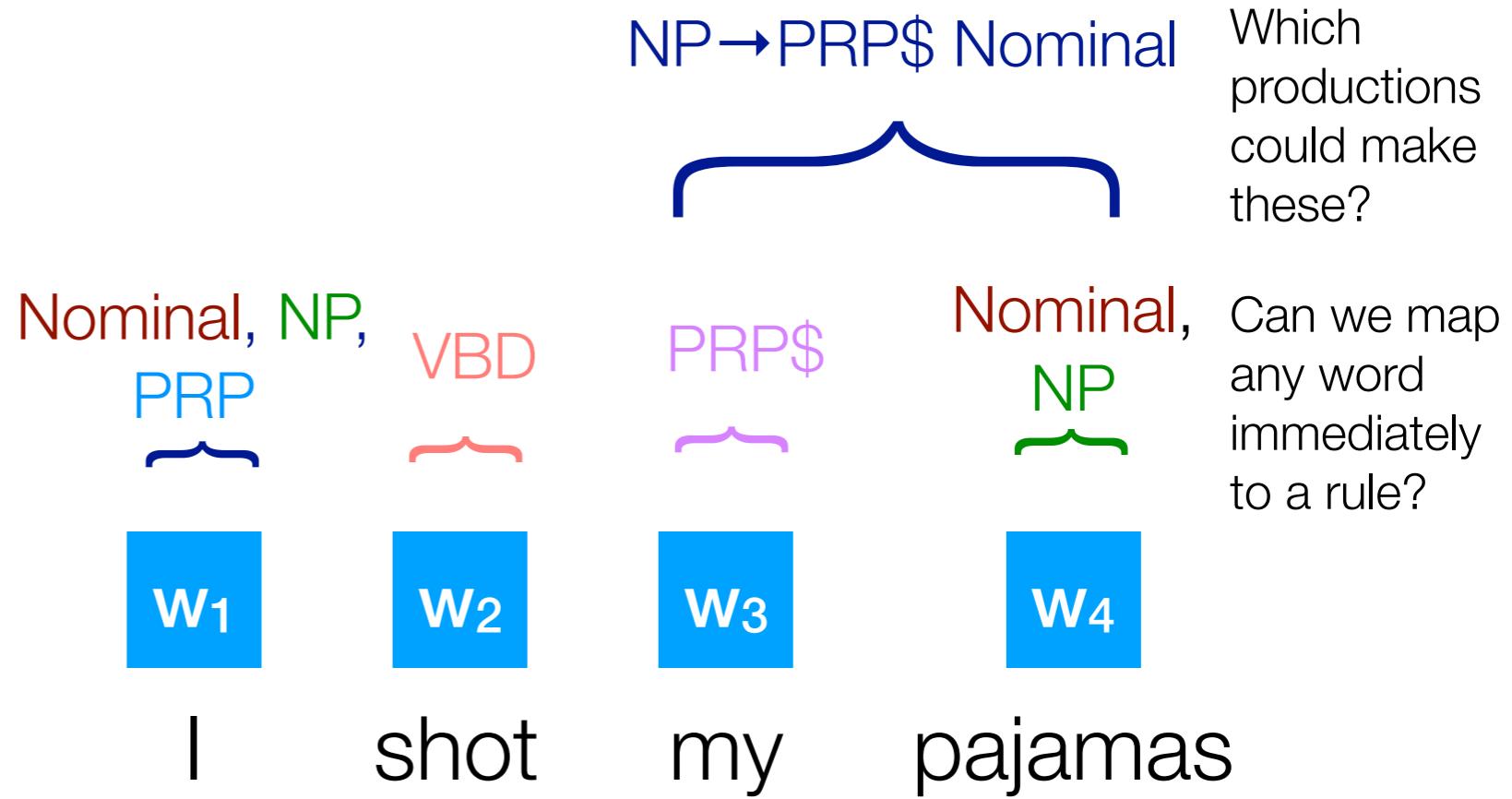


Which productions could make these?

Can we map any word immediately to a rule?

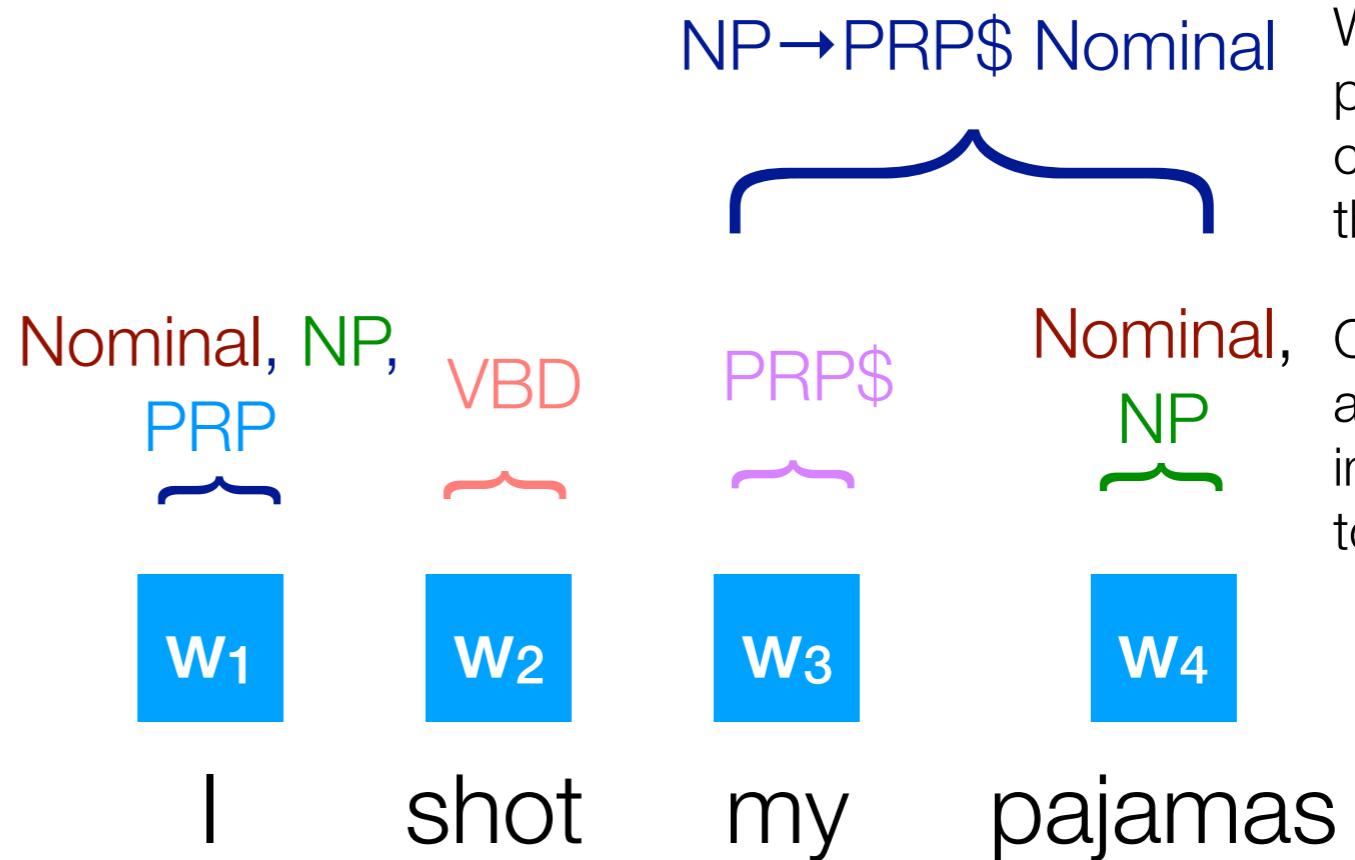
Bottom-up Parsing

S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nomin	→	Nominal PP
Nomin	→	pajamas elephant
PP	→	IN NP
NP	→	DT NN
NP	→	pajamas elephant
NP	→	PRP\$ Nominal
VBD	→	shot
DT	→	an my
PRP	→	I
PRP\$	→	my
IN	→	in



Bottom-up Parsing

S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nomin	→	Nominal PP
Nomin	→	pajamas elephant
PP	→	IN NP
NP	→	DT NN
NP	→	pajamas elephant
NP	→	PRP\$ Nominal
VBD	→	shot
DT	→	an my
PRP	→	I
PRP\$	→	my
IN	→	in



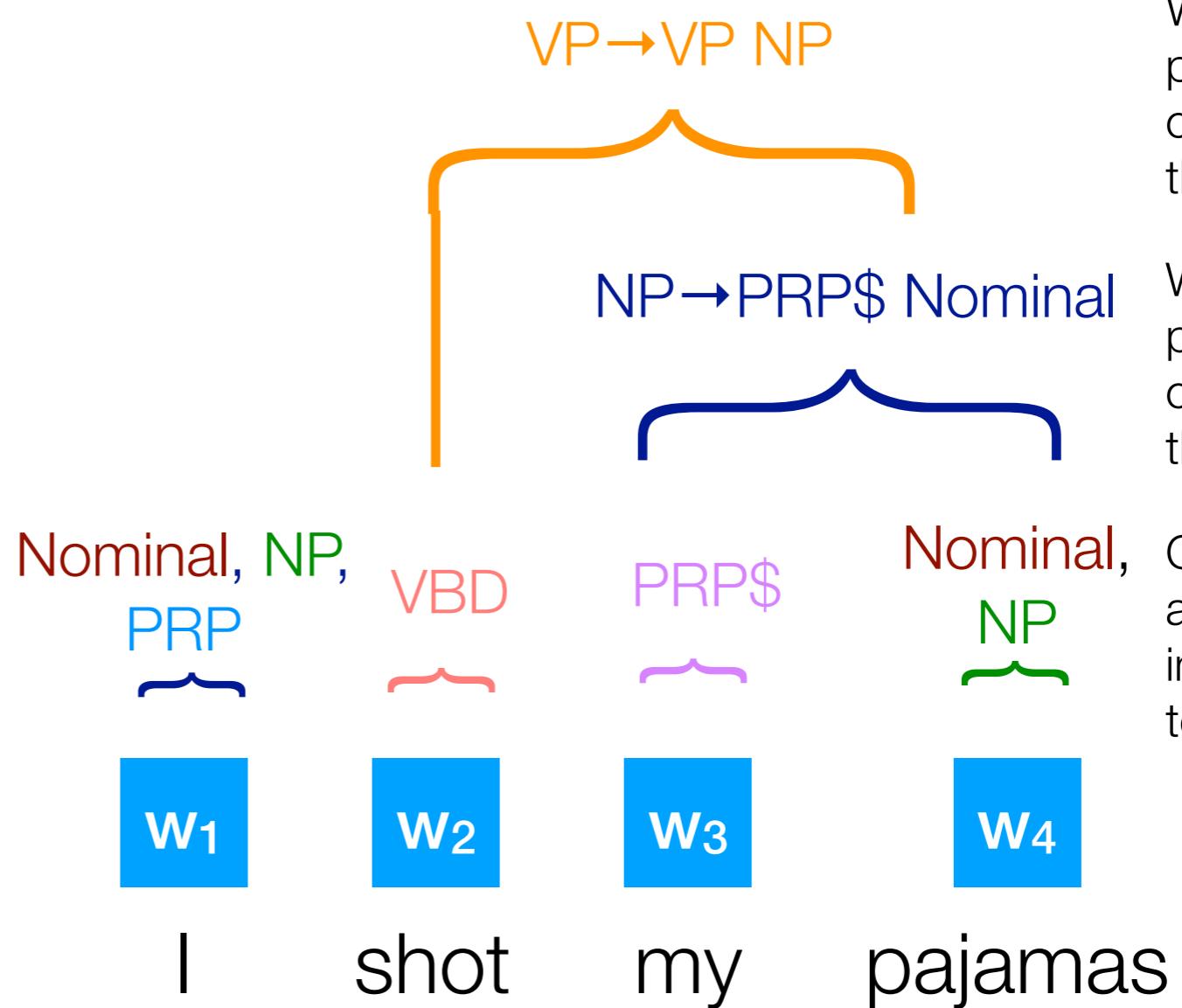
Which
productions
could make
these?

Which
productions
could make
these?

Can we map
any word
immediately
to a rule?

Bottom-up Parsing

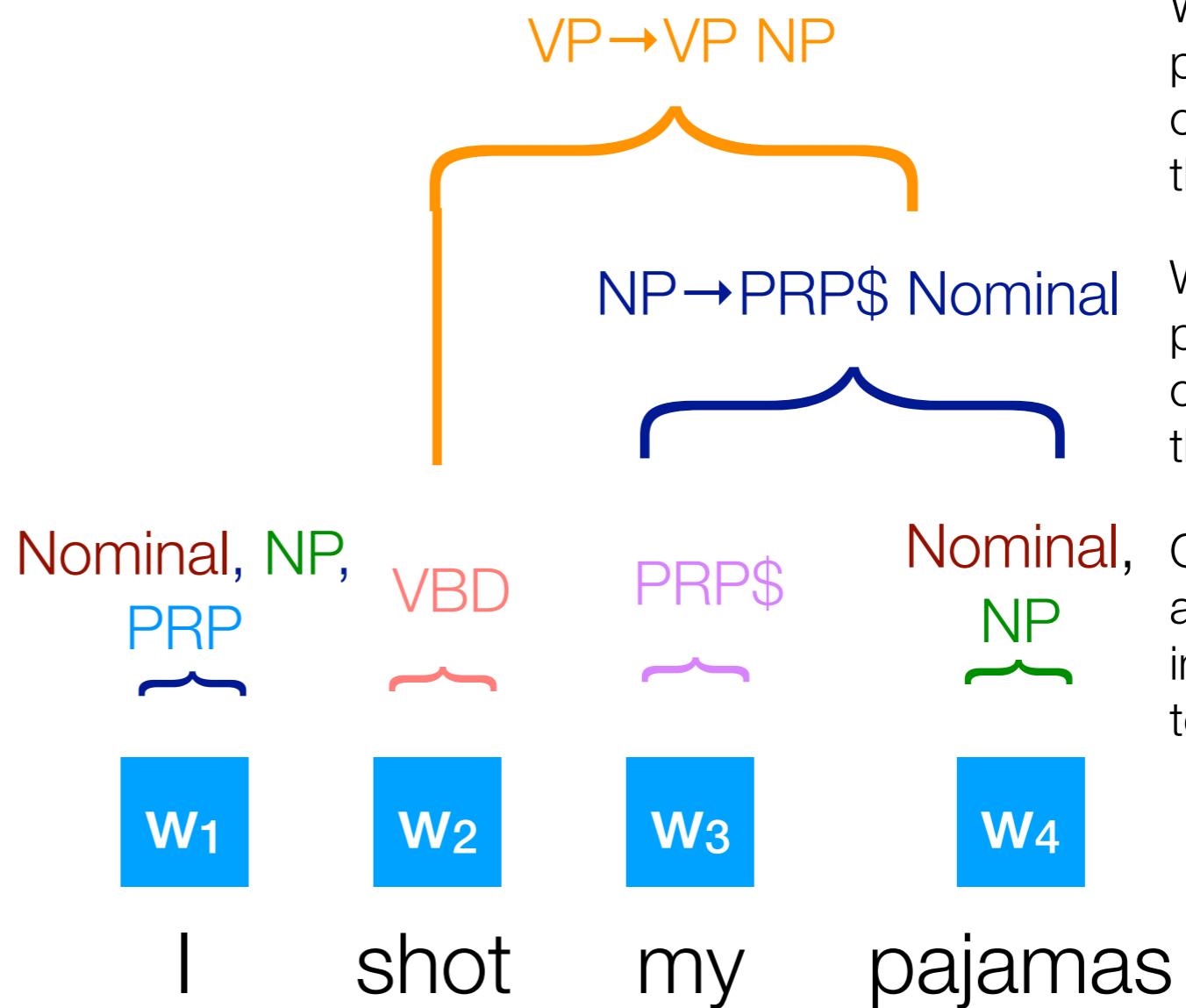
S	→	NP VP
VP	→	VBD NP
VP	→	VP PP
Nomin	→	Nominal PP
Nomin	→	pajamas elephant
PP	→	IN NP
NP	→	DT NN
NP	→	pajamas elephant
NP	→	PRP\$ Nominal
VBD	→	shot
DT	→	an my
PRP	→	I
PRP\$	→	my
IN	→	in



Bottom-up Parsing

Which productions could make these?

S	\rightarrow	NP VP
VP	\rightarrow	VBD NP
VP	\rightarrow	VP PP
Nomin	\rightarrow	Nominal PP
Nomin	\rightarrow	pajamas elephant
PP	\rightarrow	IN NP
NP	\rightarrow	DT NN
NP	\rightarrow	pajamas elephant
NP	\rightarrow	PRP\$ Nominal
VBD	\rightarrow	shot
DT	\rightarrow	an my
PRP	\rightarrow	I
PRP\$	\rightarrow	my
IN	\rightarrow	in



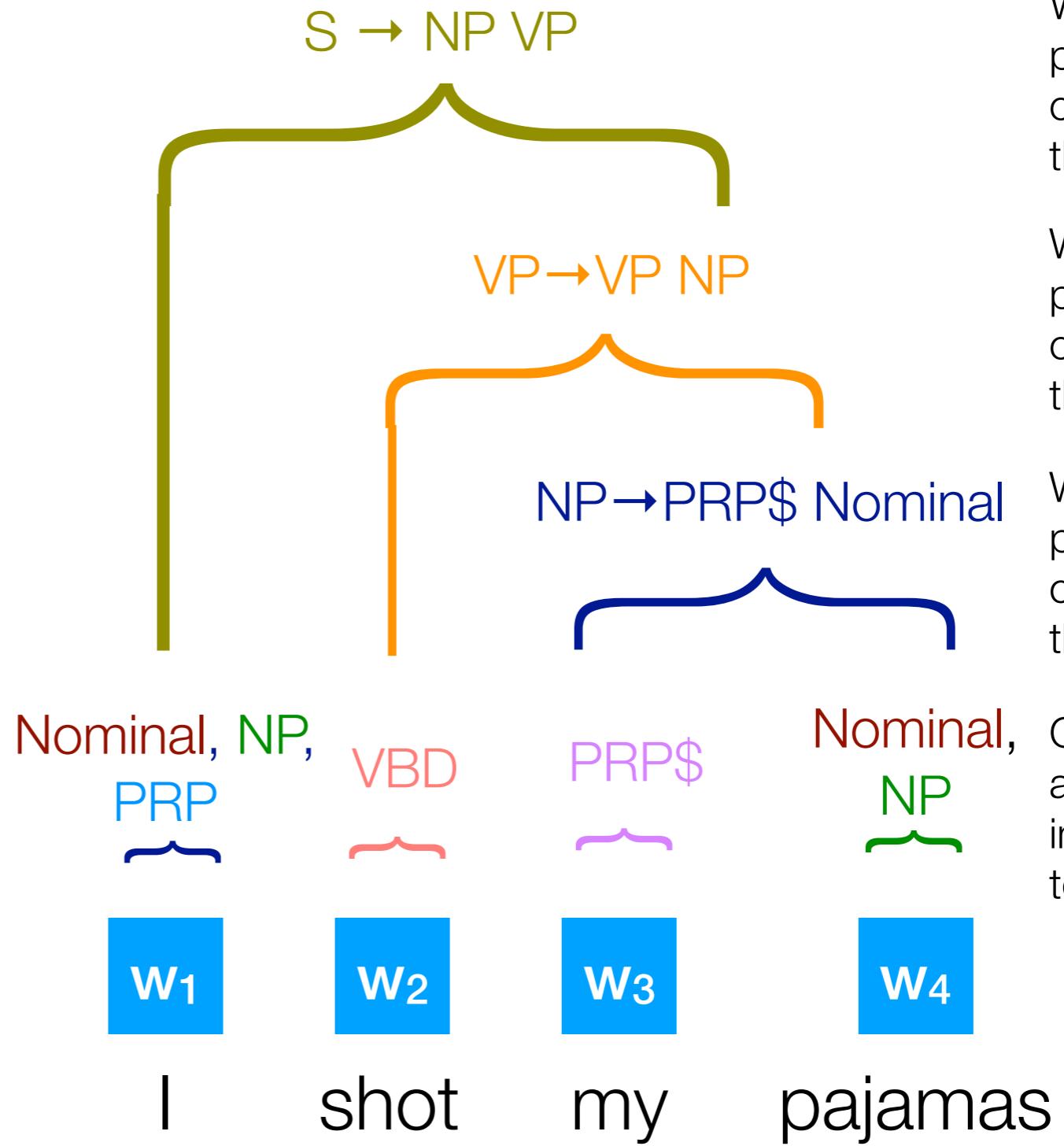
Which productions could make these?

Which productions could make these?

Can we map any word immediately to a rule?

Bottom-up Parsing

S → NP VP
VP → VBD NP
VP → VP PP
Nomin → Nominal PP
Nomin → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in



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

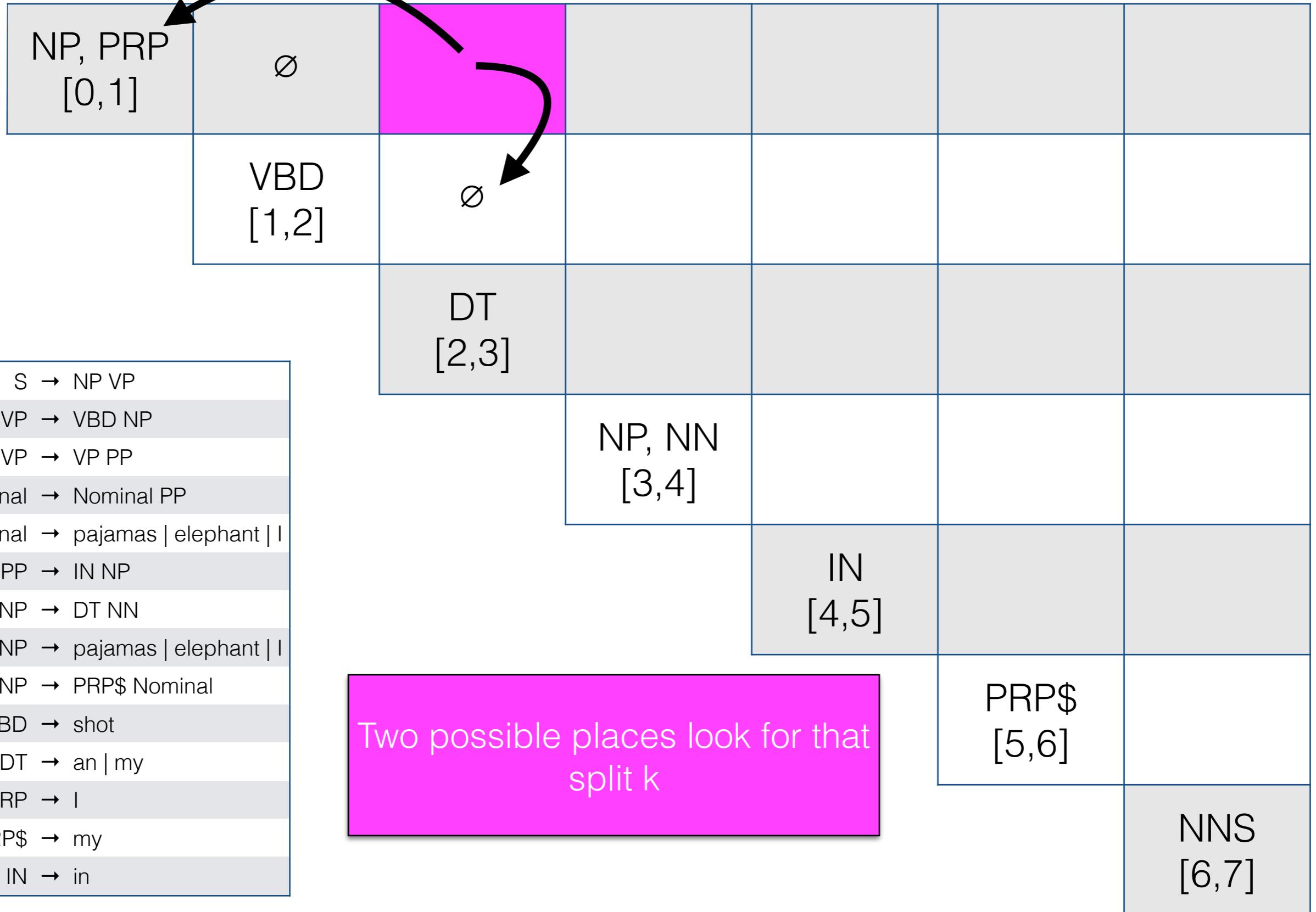
NP, PRP [0,1]						
	VBD [1,2]					
		DT [2,3]				
We initialize each of these cells with the production rules that can generate each terminal			NP, NN [3,4]			
				IN [4,5]		
					PRP\$ [5,6]	
						NNS [6,7]

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

NP, PRP [0,1]						
	VBD [1,2]					
		DT [2,3]				
			NP, NN [3,4]			
				IN [4,5]		
Does any rule generate PRP VBD?					PRP\$ [5,6]	
						NNS [6,7]

I shot an elephant in my pajamas

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



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

NP, PRP [0,1]	\emptyset					
	VBD [1,2]	\emptyset				
		DT [2,3]				
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						

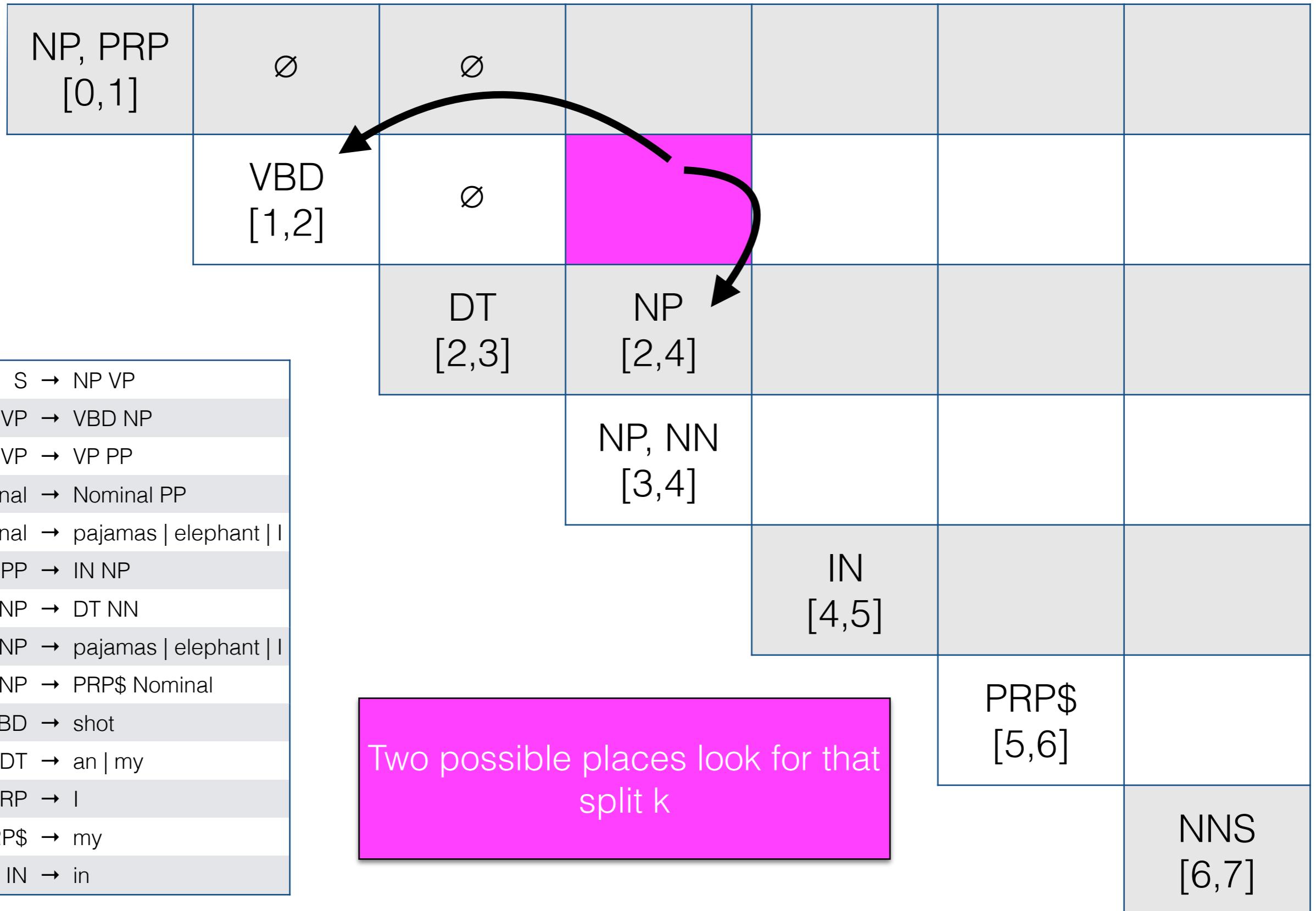
Two possible places look for that split k

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

NP, PRP [0,1]	\emptyset	\emptyset				
	VBD [1,2]	\emptyset				
		DT [2,3]				
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						

Does any rule generate
DT NN?

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



I shot an elephant in my pajamas

Two possible places look for that split k

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

NP, PRP [0,1]	\emptyset	\emptyset				
	VBD [1,2]	\emptyset	VP [1,4]			
		DT [2,3]	NP [2,4]			
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						

Three possible places look for that split k

I shot an elephant in my pajamas

NP, PRP [0,1]	\emptyset	\emptyset			
	VBD [1,2]	\emptyset		VP [1,4]	

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

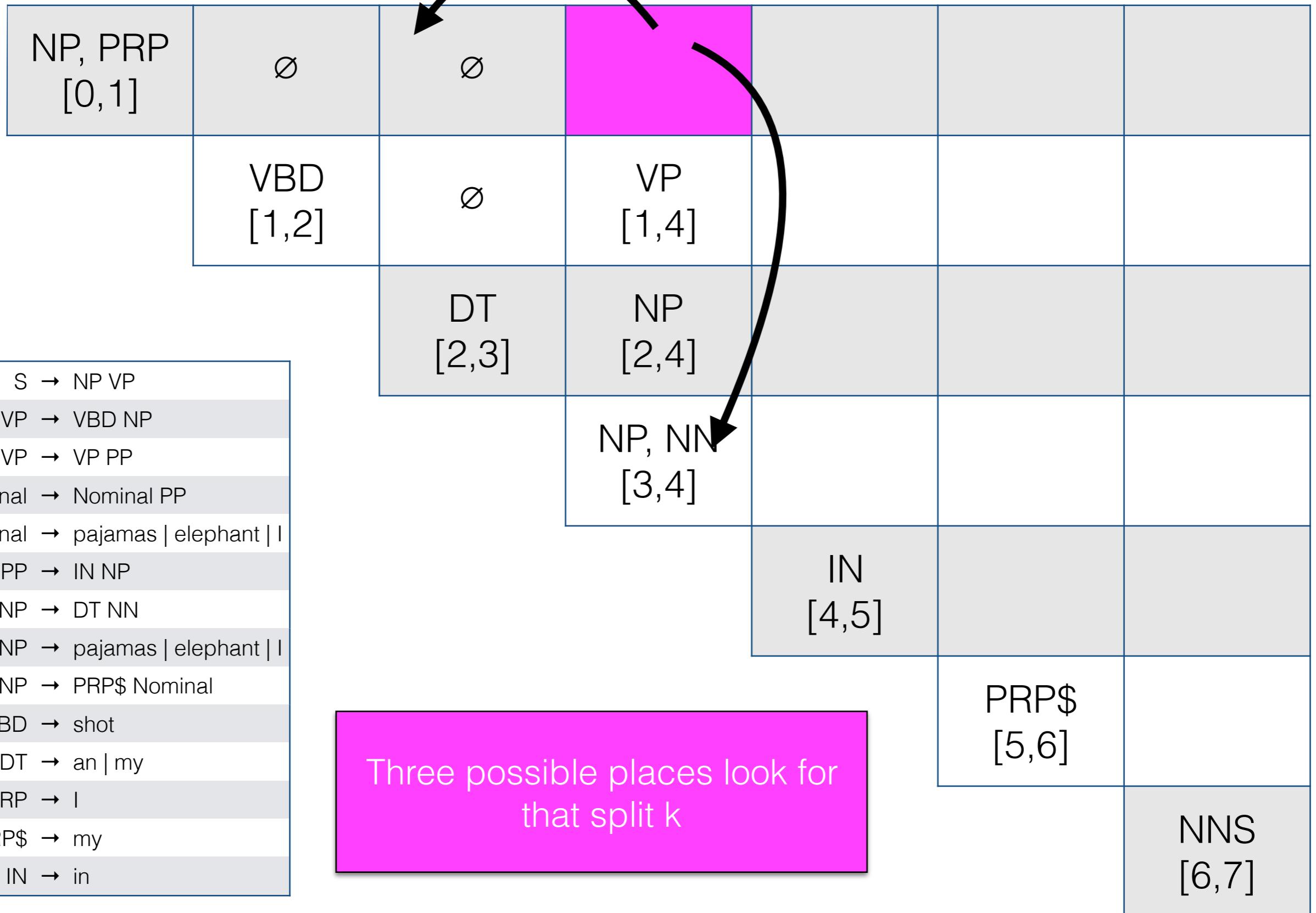
Three possible places look for
that split k

I shot an elephant in my pajamas

NP, PRP [0,1]	\emptyset	\emptyset				
	VBD [1,2]	\emptyset	VP [1,4]			
		DT [2,3]	NP [2,4]			
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						
RP → I						
P\$ → my						
IN → in						
Three possible places look for that split k						
					PRP\$ [5,6]	NNS [6,7]

Three possible places look for
that split k

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



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

NP, PRP [0,1]	Ø	Ø	S [0,4]			
	VBD [1,2]	Ø	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]	\emptyset	\emptyset	S [0,4]	\emptyset	\emptyset	
	VBD [1,2]	\emptyset	VP [1,4]	\emptyset	\emptyset	
		DT [2,3]	NP [2,4]	\emptyset	\emptyset	
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]	\emptyset	\emptyset	S [0,4]	\emptyset	\emptyset	
	VBD [1,2]	\emptyset	VP [1,4]	\emptyset	\emptyset	
		DT [2,3]	NP [2,4]	\emptyset	\emptyset	
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						

*elephant in

I shot an elephant in my pajamas

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

NP, PRP [0,1]	\emptyset	\emptyset	S [0,4]	\emptyset	\emptyset	
	VBD [1,2]	\emptyset	VP [1,4]	\emptyset	\emptyset	
		DT [2,3]	NP [2,4]	\emptyset	\emptyset	
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						
*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]	\emptyset	\emptyset	S [0,4]	\emptyset	\emptyset	
	VBD [1,2]	\emptyset	VP [1,4]	\emptyset	\emptyset	
		DT [2,3]	NP [2,4]	\emptyset	\emptyset	
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]	\emptyset	\emptyset	S [0,4]	\emptyset	\emptyset	
	VBD [1,2]	\emptyset	VP [1,4]	\emptyset	\emptyset	
		DT [2,3]	NP [2,4]	\emptyset	\emptyset	
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						

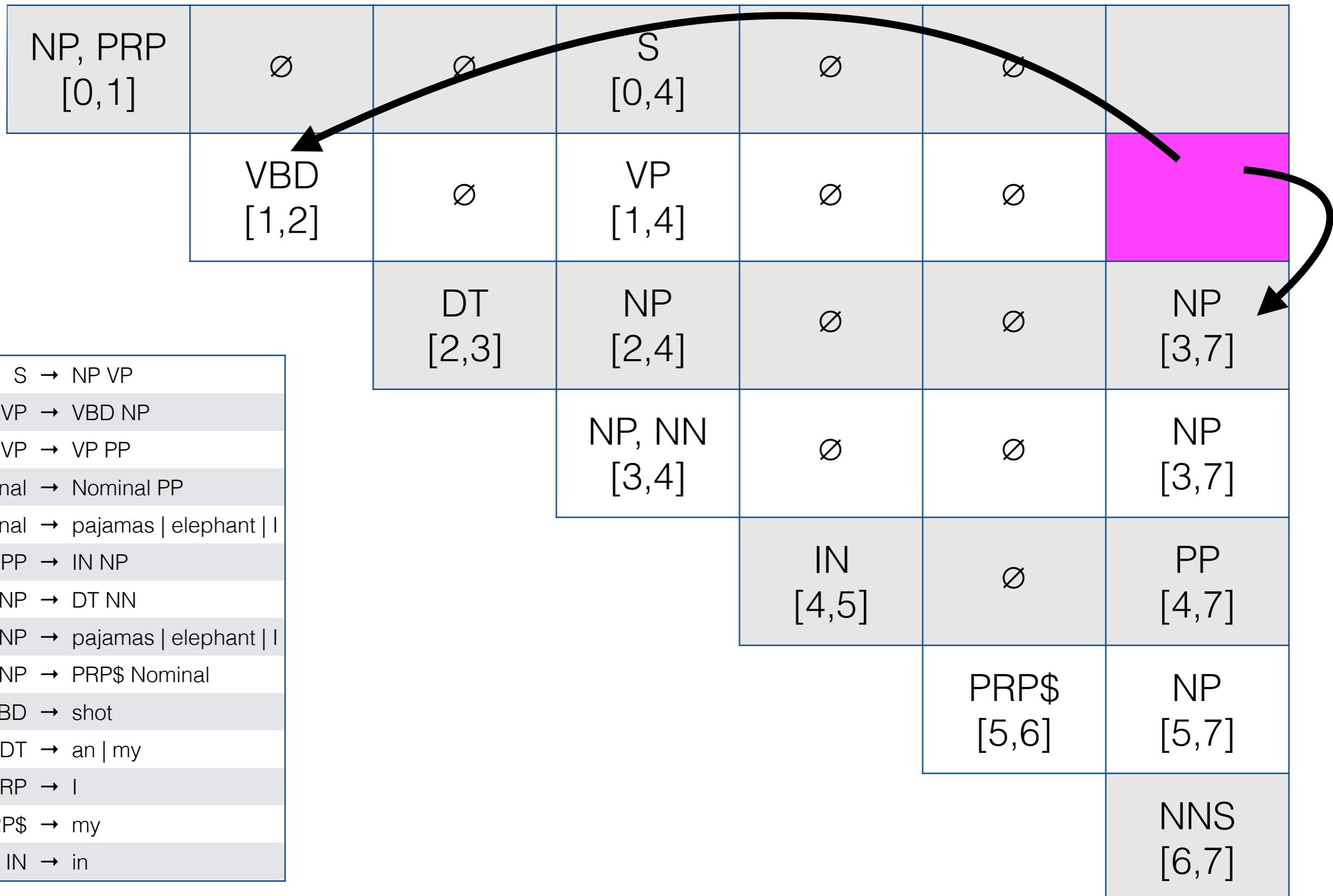
I	shot	an	elephant	in	my	pajamas
---	------	----	----------	----	----	---------

NP, PRP [0,1]	\emptyset	\emptyset	S [0,4]	\emptyset	\emptyset	
	VBD [1,2]	\emptyset	VP [1,4]	\emptyset	\emptyset	
		DT [2,3]	NP [2,4]	\emptyset	\emptyset	
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						

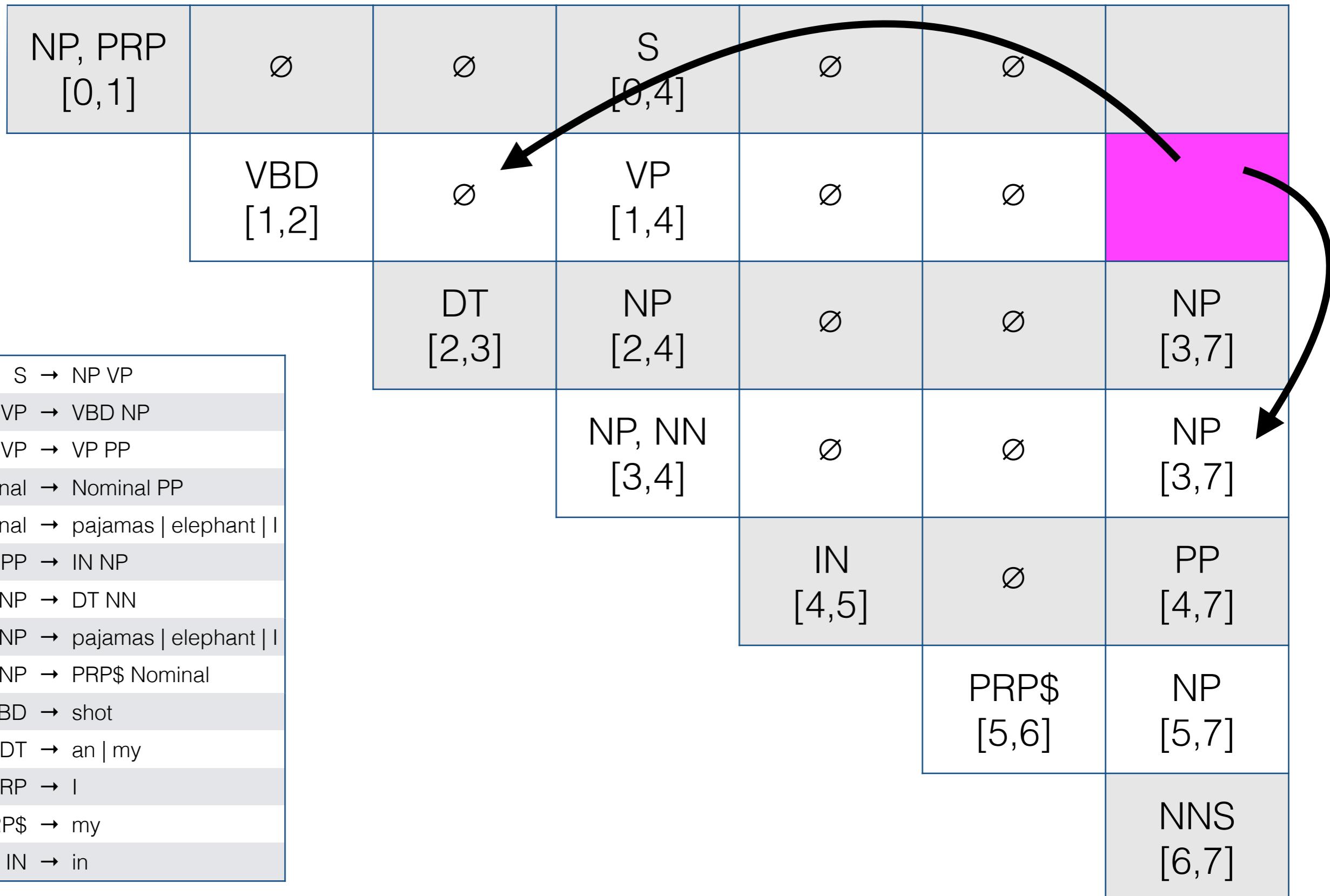
I	shot	an	elephant	in	my	pajamas
---	------	----	----------	----	----	---------

NP, PRP [0,1]	\emptyset	\emptyset	S [0,4]	\emptyset	\emptyset	
	VBD [1,2]	\emptyset	VP [1,4]	\emptyset	\emptyset	
		DT [2,3]	NP [2,4]	\emptyset	\emptyset	NP [3,7]
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						

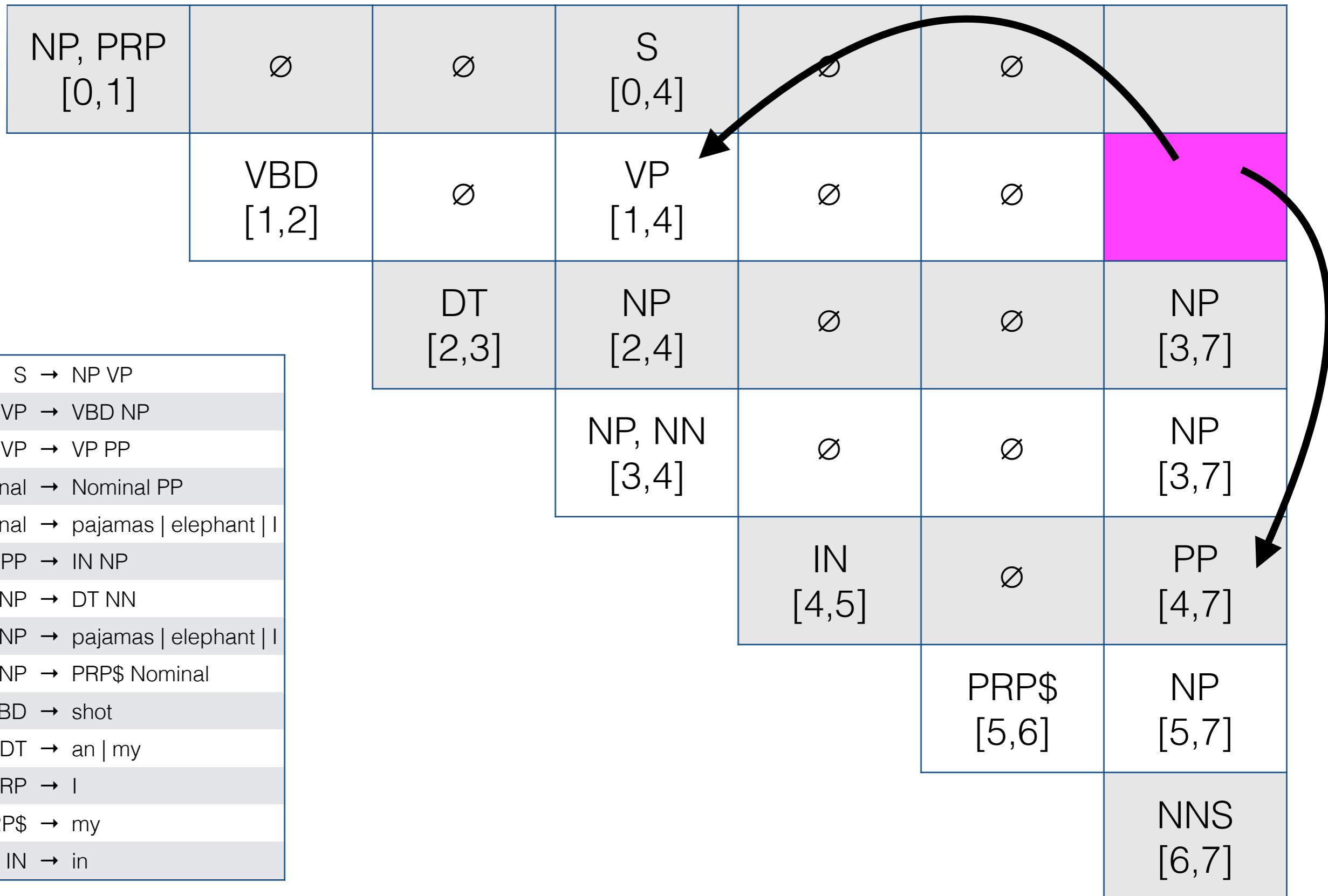
I	shot	an	elephant	in	my	pajamas
---	------	----	----------	----	----	---------



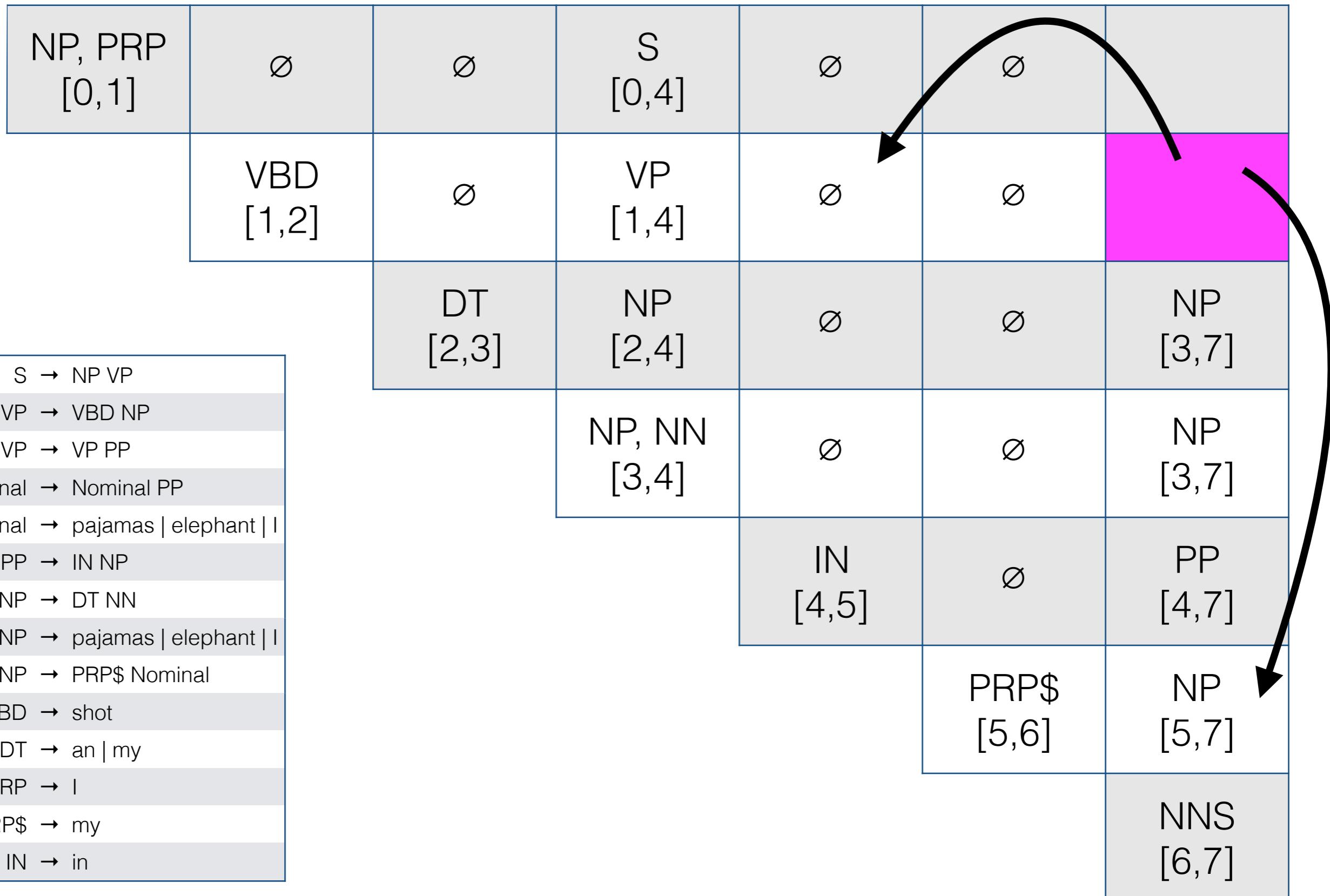
I	shot	an	elephant	in	my	pajamas
---	------	----	----------	----	----	---------



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



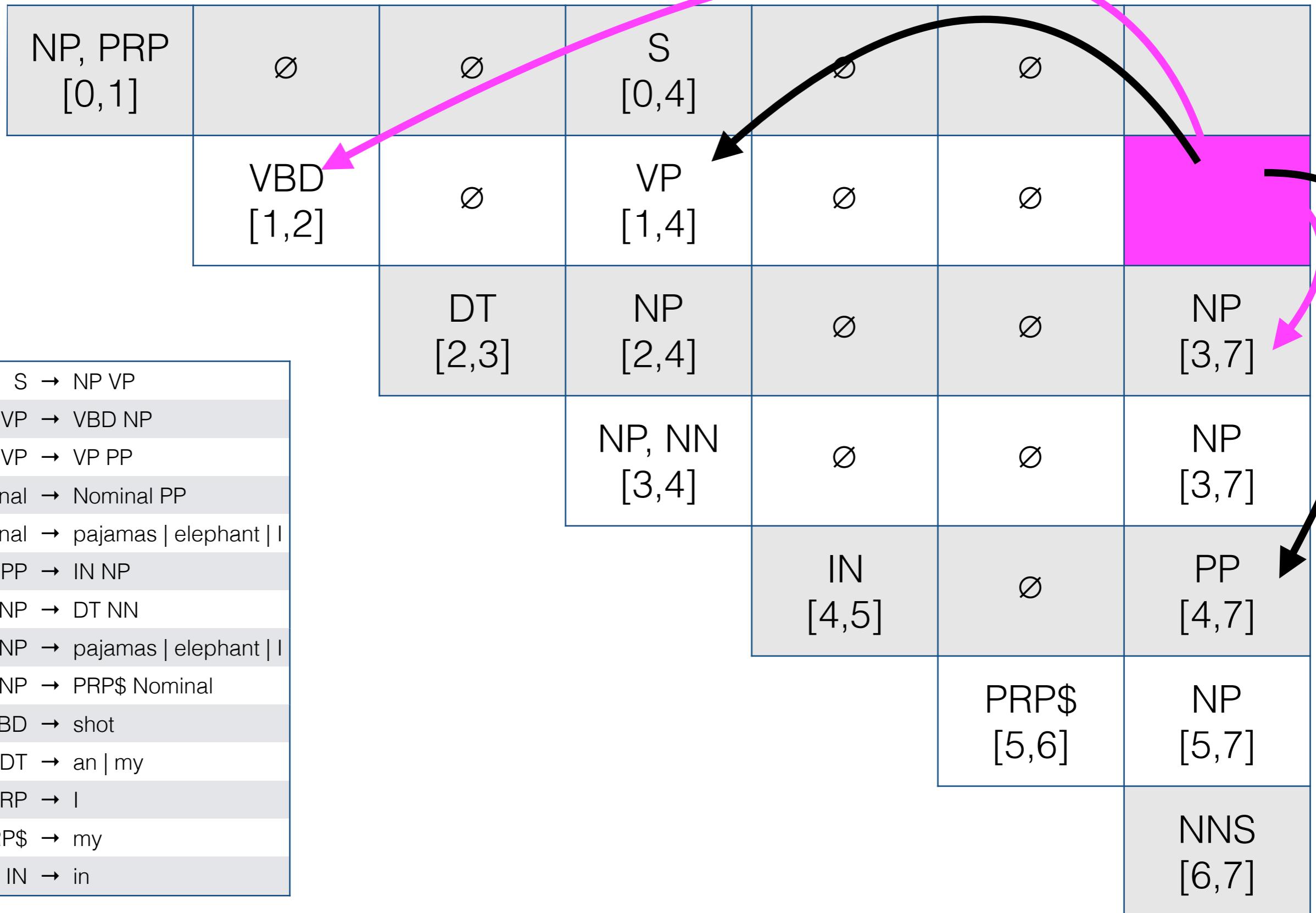
I	shot	an	elephant	in	my	pajamas
---	------	----	----------	----	----	---------



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

NP, PRP [0,1]	\emptyset	\emptyset	S [0,4]	\emptyset	\emptyset	
	VBD [1,2]	\emptyset	VP [1,4]	\emptyset	\emptyset	
		DT [2,3]	NP [2,4]	\emptyset	\emptyset	NP [3,7]
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
---	------	----	----------	----	----	---------



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

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

Possibilities:

- $S_1 \rightarrow NP VP_1$
- $S_2 \rightarrow NP VP_2$
- ? → S PP
- ? → PRP VP₁
- ? → PRP VP₂

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

NP, PRP [0,1]	\emptyset	\emptyset	S [0,4]	\emptyset	\emptyset	S ₁ , S ₂ [0,7]
	VBD [1,2]	\emptyset	VP [1,4]	\emptyset	\emptyset	VP ₁ , VP ₂ [1,7]
		DT [2,3]	NP [2,4]	\emptyset	\emptyset	NP [2,7]
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						

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

CKY algorithm (non-probabilistic)

```
function CKY-PARSE(words, grammar) returns table
    for j  $\leftarrow$  from 1 to LENGTH(words) do
        for all {A | A  $\rightarrow$  words[j]  $\in$  grammar} do
            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 | A  $\rightarrow$  BC  $\in$  grammar and B  $\in$  table[i, k] and C  $\in$  table[k, j]} do
                    table[i, j]  $\leftarrow$  table[i, j]  $\cup$  A
```

Figure 12.5 The CKY algorithm.

CKY algorithm (non-probabilistic)

```
function CKY-PARSE(words, grammar) returns table
  for j ← from 1 to LENGTH(words) do
    for all {A | A  $\rightarrow$  words[j]  $\in$  grammar} do
      table[j − 1, j]  $\leftarrow$  table[j − 1, j]  $\cup$  A
    for i ← from j − 2 downto 0 do
      for k ← i + 1 to j − 1 do
        for all {A | A  $\rightarrow$  BC  $\in$  grammar and B  $\in$  table[i, k] and C  $\in$  table[k, j]} do
          table[i, j]  $\leftarrow$  table[i, j]  $\cup$  A
```

Figure 12.5 The CKY algorithm.

CKY algorithm (non-probabilistic)

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

For each word, from left to right

Figure 12.5 The CKY algorithm.

CKY algorithm (non-probabilistic)

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

For each word, from left to right

Figure 12.5 The CKY algorithm.

CKY algorithm (non-probabilistic)

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

For each word, from left to right
Update the table with the
set of non-terminals that
produce this word (words[j])
```

Figure 12.5 The CKY algorithm.

CKY algorithm (non-probabilistic)

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

For each word, from left to right

Update the table with the set of non-terminals that produce this word (*words*[*j*])

Figure 12.5 The CKY algorithm.

CKY algorithm (non-probabilistic)

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

For each word, from left to right
Update the table with the set of non-terminals that produce this word (*words*[*j*])

Figure 12.5 The CKY algorithm.

If we see that words i-to-k are produced by non-terminal B

CKY algorithm (non-probabilistic)

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

For each word, from left to right
Update the table with the set of non-terminals that produce this word (*words*[*j*])

Figure 12.5 The CKY algorithm.

If we see that words i-to-k are produced by non-terminal B
and words k-to-j are produced by non-terminal C

CKY algorithm (non-probabilistic)

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

For each word, from left to right
Update the table with the set of non-terminals that produce this word (*words*[*j*])

Figure 12.5 The CKY algorithm.

If we see that words *i*-to-*k* are produced by non-terminal *B* and words *k*-to-*j* are produced by non-terminal *C* and there's a production rule *A* that produces *BC*,

CKY algorithm (non-probabilistic)

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

For each word, from left to right
Update the table with the set of non-terminals that produce this word (*words*[*j*])

Figure 12.5 The CKY algorithm.

If we see that words *i*-to-*k* are produced by non-terminal *B* and words *k*-to-*j* are produced by non-terminal *C* and there's a production rule *A* that produces *BC*, then update the table for [*i*,*j*] (i.e., the one for words *i*-to-*j*) with *A*.

CKY algorithm (non-probabilistic)

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

Figure 12.5 The CKY algorithm.

If we see that words i-to-k are produced by non-terminal B
and words k-to-j are produced by non-terminal C
and there's a production rule A that produces BC,
then update the table for [i,j] (i.e., the one for words i-to-j) with A.
Do this for all A's that match.

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

NP, PRP [0,1]	\emptyset	\emptyset	S [0,4]	\emptyset	\emptyset	S ₁ , S ₂ [0,7]
VBD [1,2]	\emptyset	\emptyset	VP [1,4]	\emptyset	\emptyset	VP ₁ , VP ₂ [1,7]
	DT [2,3]	NP [2,4]		\emptyset	\emptyset	NP [2,7]
		NP, NN [3,4]		\emptyset	\emptyset	NP [3,7]
			IN [4,5]		\emptyset	PP [4,7]
Runtime complexity?				PRP\$ [5,6]	NP [5,7]	NNS [6,7]

CFG

- This use of CKY allows us to:
 - check whether a sentence is 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.



Probabilistic CFGs

- 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.
- We calculate the max probability parse using CKY by storing the probability of each phrase within each cell as we build it up.

A Probabilistic Grammar

S → NP VP
VP → VBD NP
VP → VP PP
Nominal → Nominal PP
Nominal → pajamas elephant
PP → IN NP
NP → DT NN
NP → pajamas elephant
NP → PRP\$ Nominal
VBD → shot
DT → an my
PRP → I
PRP\$ → my
IN → in

A Probabilistic Grammar

1.0	S → NP VP
0.6	VP → VBD NP
0.4	VP → VP PP
0.2	Nominal → Nominal PP
0.8	Nominal → pajamas elephant
1.0	PP → IN NP
0.6	NP → DT NN
0.3	NP → pajamas elephant
0.1	NP → PRP\$ Nominal
1.0	VBD → shot
1.0	DT → an my
1.0	PRP → I
1.0	PRP\$ → my
1.0	IN → in

A Probabilistic Grammar

The probabilities for each nonterminal's production rules must sum to 1

1.0	S → NP VP
0.6	VP → VBD NP
0.4	VP → VP PP
0.2	Nominal → Nominal PP
0.8	Nominal → pajamas elephant
1.0	PP → IN NP
0.6	NP → DT NN
0.3	NP → pajamas elephant
0.1	NP → PRP\$ Nominal
1.0	VBD → shot
1.0	DT → an my
1.0	PRP → I
1.0	PRP\$ → my
1.0	IN → in

A Probabilistic Grammar

The probabilities for each nonterminal's production rules must sum to 1

1.0	S → NP VP
0.6	VP → VBD NP
0.4	VP → VP PP
0.2	Nominal → Nominal PP
0.8	Nominal → pajamas elephant
1.0	PP → IN NP
0.6	NP → DT NN
0.3	NP → pajamas elephant
0.1	NP → PRP\$ Nominal
1.0	VBD → shot
1.0	DT → an my
1.0	PRP → I
1.0	PRP\$ → my
1.0	IN → in

Where do we get these probabilities?

How to keep track of probabilities in Probabilistic CYK

$$table(i, j, A) = P(A \rightarrow BC) \times table(i, k, B) \times table(k, j, C)$$

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

PRP:0.04 [0,1]						
	VBD:0.04 [1,2]					
		DT:0.05 [2,3]				
			NN:0.03 [3,4]			
				IN:0.10 [4,5]		
Probability of a terminal (word) given its tag					PRP\$:0.1 2 [5 6]	
						NNS:0.01 [6,7]

$$P(A \rightarrow \beta)$$

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

PRP:0.04 [0,1]	∅	∅				
VBD:0.04 [1,2]		∅				
	DT:0.05 [2,3]	NP:0.0001 5 [2,4]				
		NN:0.03 [3,4]				
			IN:0.10 [4,5]			
				PRP\$:0.12 [5,6]		
					NNS:0.01 [6,7]	

$$table(2, 4, NP) = P(NP \rightarrow DT \ NN) \times table(2, 3, DT) \times table(3, 4, NN)$$

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

PRP:0.04 [0,1]	\emptyset	\emptyset				
	VBD:0.04 [1,2]	\emptyset	VP:0.0000 006 [1,4]			
		DT:0.05 [2,3]	NP:0.0001 5 [2,4]			
			NN:0.03 [3,4]			
				IN:0.10 [4,5]		
We just calculated this value and can use it now					PRP\$:0.12 [5,6]	
					NNS:0.01 [6,7]	

$$table(1, 4, VP) = P(VP \rightarrow VBD \ NP) \times table(1, 2, VBD) \times \text{table}(2, 4, NP)$$

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

PRP:0.04 [0,1]	\emptyset	\emptyset	S: 0.0000000 048 [0,4]			
VBD:0.04 [1,2]	\emptyset	\emptyset	VP:0.0000 006 [1,4]			
	DT:0.05 [2,3]	NP:0.0001 5 [2,4]				
		NN:0.03 [3,4]				
We just calculated this value and can use it now		IN:0.10 [4,5]			PRP\$:0.12 [5,6]	
					NNS:0.01 [6,7]	

$$table(0, 4, S) = P(S \rightarrow NP \ VP) \times table(0, 1, NP) \times \text{table}(1, 4, VP)$$

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

PRP:0.04 [0,1]	\emptyset	\emptyset	S: 0.0000000 048 [0,4]			
VBD:0.04 [1,2]	\emptyset	\emptyset	VP:0.0000 006 [1,4]			
	DT:0.05 [2,3]		NP:0.0001 5 [2,4]			
		NN:0.03 [3,4]				
			IN:0.10 [4,5]			
Note these values are getting very small! Better to add in log space			PRP\$:0.12 [5,6]			
			NNS:0.01 [6,7]			

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

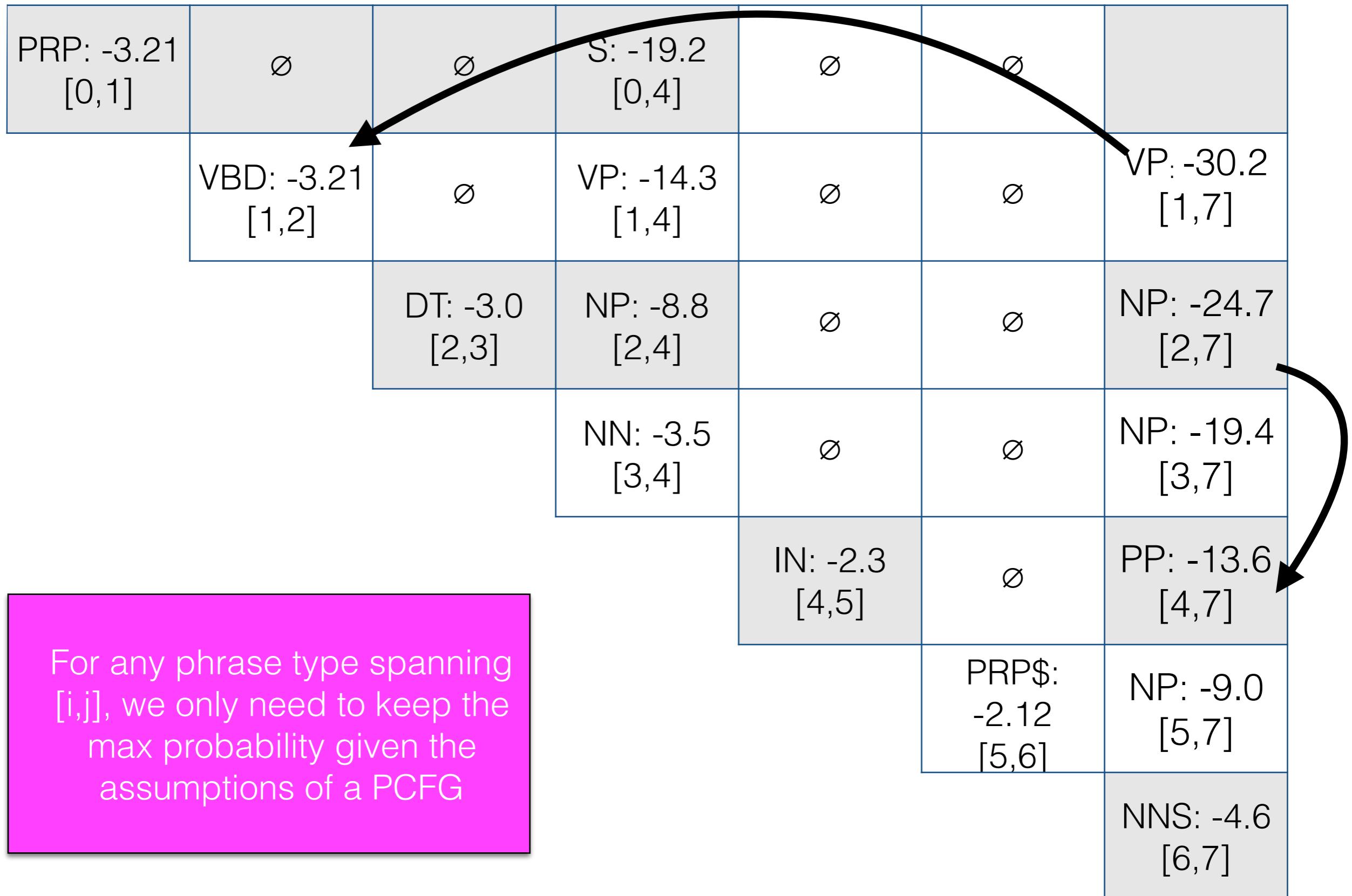
PRP: -3.21 [0,1]	∅	∅	S: -19.2 [0,4]			
VBD: -3.21 [1,2]	∅	VP: -14.3 [1,4]				
	DT: -3.0 [2,3]	NP: -8.8 [2,4]				
		NN: -3.5 [3,4]				
		IN: -2.3 [4,5]				
Note these values are getting very small! Better to add in log space		PRP\$: -2.12 [5,6]				
		NNS: -4.6 [6,7]				

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

PRP: -3.21 [0,1]	\emptyset	\emptyset	S: -19.2 [0,4]	\emptyset	\emptyset	
VBD: -3.21 [1,2]	\emptyset	\emptyset	VP: -14.3 [1,4]	\emptyset	\emptyset	VP ₁ , VP ₂ [1,7]
DT: -3.0 [2,3]	NP: -8.8 [2,4]	\emptyset	\emptyset	\emptyset	NP: -24.7 [2,7]	NP: -19.4 [3,7]
NN: -3.5 [3,4]	\emptyset	\emptyset	\emptyset	\emptyset	PP: -13.6 [4,7]	
IN: -2.3 [4,5]	\emptyset	\emptyset	\emptyset	\emptyset	PRP\$: -2.12 [5,6]	NP: -9.0 [5,7]
						NNS: -4.6 [6,7]

For any phrase type spanning [i,j], we only need to keep the max probability given the assumptions of a PCFG

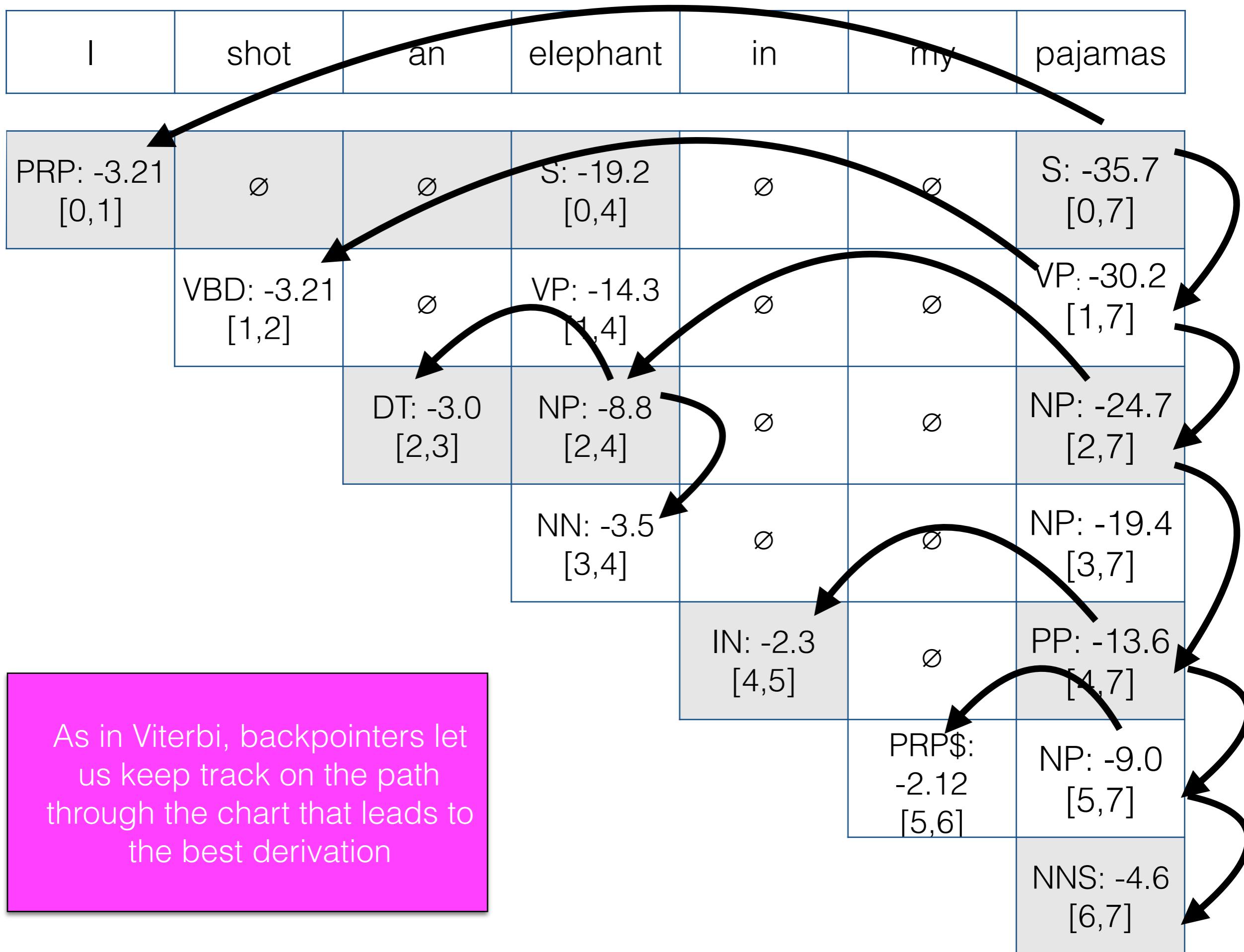
I	shot	an	elephant	in	my	pajamas
---	------	----	----------	----	----	---------

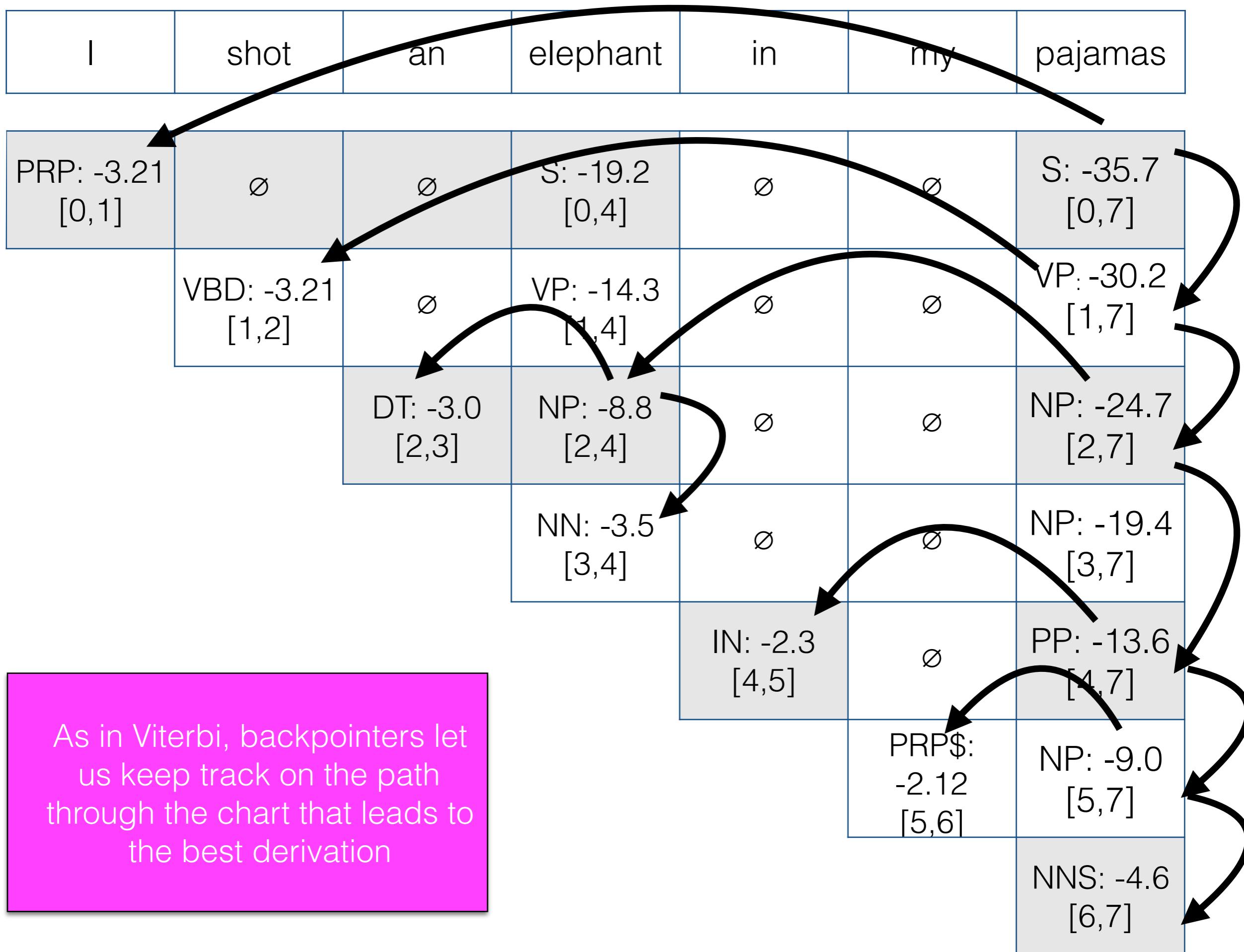


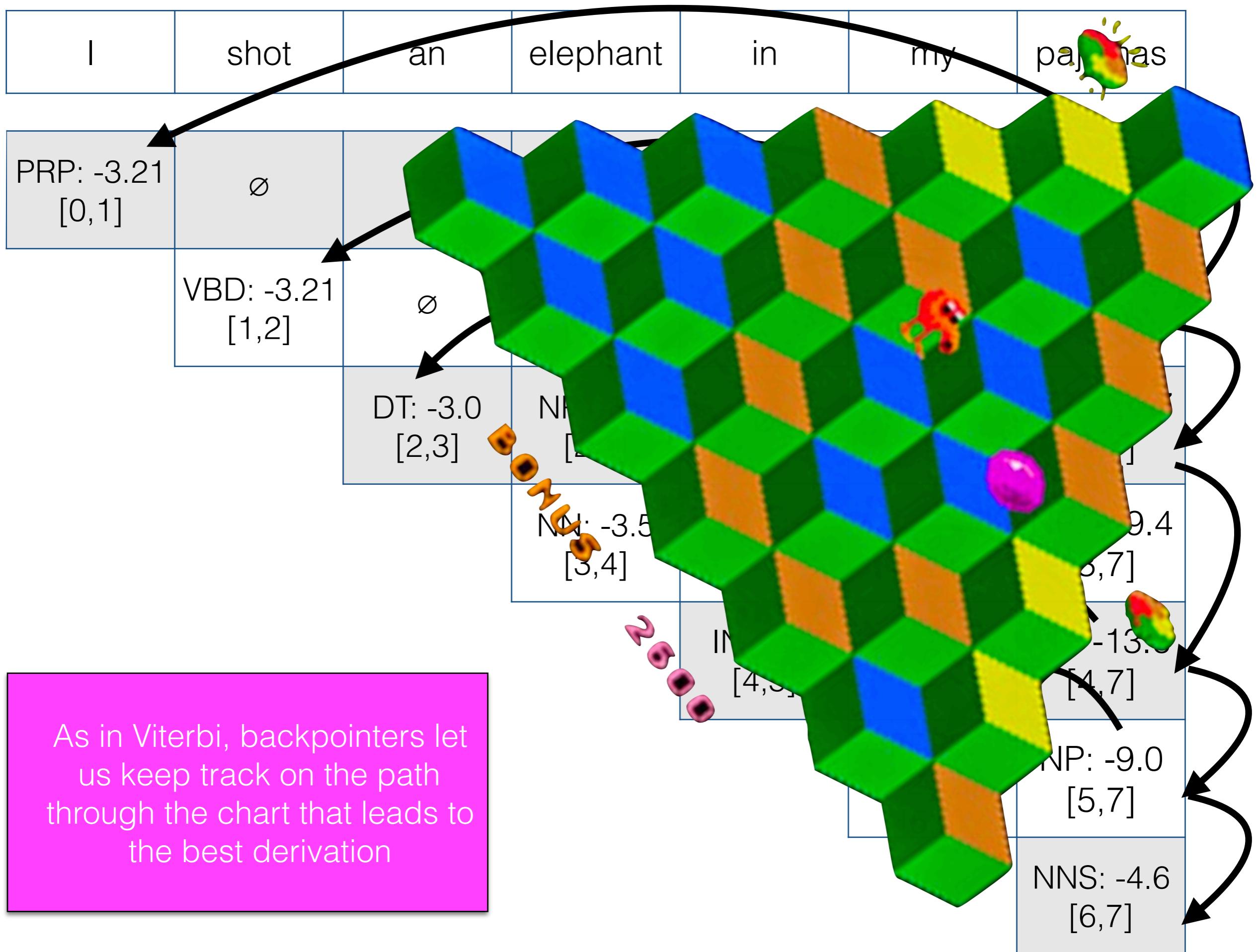
For any phrase type spanning $[i,j]$, we only need to keep the max probability given the assumptions of a PCFG

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

PRP: -3.21 [0,1]	\emptyset	\emptyset	S: -19.2 [0,4]	\emptyset	\emptyset	S: -35.7 [0,7]
VBD: -3.21 [1,2]	\emptyset	\emptyset	VP: -14.3 [1,4]	\emptyset	\emptyset	VP: -30.2 [1,7]
	DT: -3.0 [2,3]	NP: -8.8 [2,4]		\emptyset	\emptyset	NP: -24.7 [2,7]
		NN: -3.5 [3,4]		\emptyset	\emptyset	NP: -19.4 [3,7]
			IN: -2.3 [4,5]		\emptyset	PP: -13.6 [4,7]
For any phrase type spanning [i,j], we only need to keep the max probability given the assumptions of a PCFG				PRP\$: -2.12 [5,6]		NP: -9.0 [5,7]
						NNS: -4.6 [6,7]







PCFG

PCFG

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

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 $A \rightarrow \beta$):

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 $A \rightarrow \beta$):

$$P(T, S) = \prod_i^n P(\beta | A)$$

Probabilistic CYK

```
let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .
let  $P[n,n,r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.
let  $back[n,n,r]$  be an array of backpointing triples.
for each  $s = 1$  to  $n$ 
    for each unit production  $R_v \rightarrow w_s$ 
        set  $P[1,s,v] = \Pr(R_v \rightarrow w_s)$ 
for each  $l = 2$  to  $n$  -- Length of span
    for each  $s = 1$  to  $n-l+1$  -- Start of span
        for each  $p = 1$  to  $l-1$  -- Partition of span
            for each production  $R_a \rightarrow R_b R_c$ 
                prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p,s,b] * P[l-p,s+p,c]$ 
                if  $P[p,s,b] > 0$  and  $P[l-p,s+p,c] > 0$  and  $P[1,s,a] < \text{prob\_splitting}$  then
                    set  $P[1,s,a] = \text{prob\_splitting}$ 
                    set  $back[1,s,a] = \langle p, b, c \rangle$ 
```

Probabilistic CYK

We often use 'S' for start

```
let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .
let  $P[n,n,r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.
let  $back[n,n,r]$  be an array of backpointing triples.
for each  $s = 1$  to  $n$ 
    for each unit production  $R_v \rightarrow w_s$ 
        set  $P[1,s,v] = \Pr(R_v \rightarrow w_s)$ 
for each  $l = 2$  to  $n$  -- Length of span
    for each  $s = 1$  to  $n-l+1$  -- Start of span
        for each  $p = 1$  to  $l-1$  -- Partition of span
            for each production  $R_a \rightarrow R_b R_c$ 
                prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p,s,b] * P[l-p,s+p,c]$ 
                if  $P[p,s,b] > 0$  and  $P[l-p,s+p,c] > 0$  and  $P[1,s,a] < \text{prob\_splitting}$  then
                    set  $P[1,s,a] = \text{prob\_splitting}$ 
                    set  $back[1,s,a] = \langle p, b, c \rangle$ 
```

Probabilistic CYK

We often use 'S' for start

```
let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .  
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .  
let  $P[n, n, r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.  
let  $back[n, n, r]$  be an array of backpointing triples.  
for each  $s = 1$  to  $n$   
  for each unit production  $R_v \rightarrow w_s$   
    set  $P[1, s, v] = \Pr(R_v \rightarrow w_s)$   
for each  $l = 2$  to  $n$  -- Length of span  
  for each  $s = 1$  to  $n-l+1$  -- Start of span  
    for each  $p = 1$  to  $l-1$  -- Partition of span  
      for each production  $R_a \rightarrow R_b R_c$   
        prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p, s, b] * P[l-p, s+p, c]$   
        if  $P[p, s, b] > 0$  and  $P[l-p, s+p, c] > 0$  and  $P[1, s, a] < \text{prob\_splitting}$  then  
          set  $P[1, s, a] = \text{prob\_splitting}$   
          set  $back[1, s, a] = \langle p, b, c \rangle$ 
```

Update the table with the set of non-terminals that produce w_s (i.e., string of length 1)

Probabilistic CYK

We often use 'S' for start

```
let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .
let  $P[n, n, r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.
let  $back[n, n, r]$  be an array of backpointing triples.

for each  $s = 1$  to  $n$ 
    for each unit production  $R_v \rightarrow w_s$ 
        set  $P[1, s, v] = \Pr(R_v \rightarrow w_s)$ 

for each  $l = 2$  to  $n$  -- Length of span
    for each  $s = 1$  to  $n-l+1$  -- Start of span
        for each  $p = 1$  to  $l-1$  -- Partition of span
            for each production  $R_a \rightarrow R_b R_c$ 
                prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p, s, b] * P[l-p, s+p, c]$ 
                if  $P[p, s, b] > 0$  and  $P[l-p, s+p, c] > 0$  and  $P[1, s, a] < \text{prob\_splitting}$  then
                    set  $P[1, s, a] = \text{prob\_splitting}$ 
                    set  $back[1, s, a] = \langle p, b, c \rangle$ 
```

Update the table with the set of
non-terminals that produce w_s
(i.e., string of length 1)

How you choose to break
up the words in the span
according to different
production rules

Probabilistic CYK

We often use 'S' for start

```
let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .  
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .  
let  $P[n,n,r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.  
let  $back[n,n,r]$  be an array of backpointing triples.  
for each  $s = 1$  to  $n$   
    for each unit production  $R_v \rightarrow w_s$   
        set  $P[1,s,v] = \Pr(R_v \rightarrow w_s)$   
for each  $l = 2$  to  $n$  -- Length of span  
    for each  $s = 1$  to  $n-l+1$  -- Start of span  
        for each  $p = 1$  to  $l-1$  -- Partition of span  
            for each production  $R_a \rightarrow R_b R_c$   
                prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p,s,b] * P[l-p,s+p,c]$   
                if  $P[p,s,b] > 0$  and  $P[l-p,s+p,c] > 0$  and  $P[1,s,a] < \text{prob\_splitting}$  then  
                    set  $P[1,s,a] = \text{prob\_splitting}$   
                    set  $back[1,s,a] = \langle p, b, c \rangle$ 
```

Update the table with the set of non-terminals that produce w_s (i.e., string of length 1)

How you choose to break up the words in the span according to different production rules



Probabilistic CYK

We often use 'S' for start

```
let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .  
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .  
let  $P[n,n,r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.  
let  $back[n,n,r]$  be an array of backpointing triples.  
for each  $s = 1$  to  $n$   
    for each unit production  $R_v \rightarrow w_s$   
        set  $P[1,s,v] = \Pr(R_v \rightarrow w_s)$   
for each  $l = 2$  to  $n$  -- Length of span  
    for each  $s = 1$  to  $n-l+1$  -- Start of span  
        for each  $p = 1$  to  $l-1$  -- Partition of span  
            for each production  $R_a \rightarrow R_b R_c$   
                prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p,s,b] * P[l-p,s+p,c]$   
                if  $P[p,s,b] > 0$  and  $P[l-p,s+p,c] > 0$  and  $P[1,s,a] < \text{prob\_splitting}$  then  
                    set  $P[1,s,a] = \text{prob\_splitting}$   
                    set  $back[1,s,a] = \langle p, b, c \rangle$ 
```

Update the table with the set of non-terminals that produce w_s (i.e., string of length 1)

How you choose to break up the words in the span according to different production rules



Probabilistic CYK

We often use 'S' for start

```

let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .
let  $P[n, n, r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.
let  $back[n, n, r]$  be an array of backpointing triples.

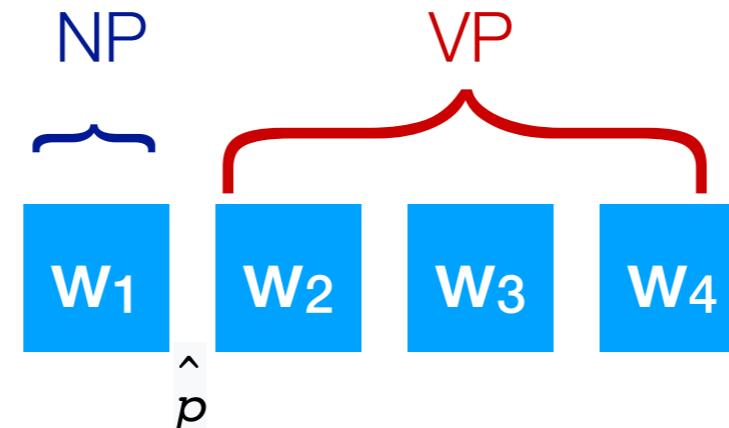
for each  $s = 1$  to  $n$ 
    for each unit production  $R_v \rightarrow w_s$ 
        set  $P[1, s, v] = \Pr(R_v \rightarrow w_s)$ 

for each  $l = 2$  to  $n$  -- Length of span
    for each  $s = 1$  to  $n-l+1$  -- Start of span
        for each  $p = 1$  to  $l-1$  -- Partition of span
            for each production  $R_a \rightarrow R_b R_c$ 
                prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p, s, b] * P[l-p, s+p, c]$ 
                if  $P[p, s, b] > 0$  and  $P[l-p, s+p, c] > 0$  and  $P[1, s, a] < \text{prob\_splitting}$  then
                    set  $P[1, s, a] = \text{prob\_splitting}$ 
                    set  $back[1, s, a] = \langle p, b, c \rangle$ 

```

Update the table with the set of non-terminals that produce w_s (i.e., string of length 1)

How you choose to break up the words in the span according to different production rules



Probabilistic CYK

We often use 'S' for start

```

let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .
let  $P[n, n, r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.
let  $back[n, n, r]$  be an array of backpointing triples.

for each  $s = 1$  to  $n$ 
    for each unit production  $R_v \rightarrow w_s$ 
        set  $P[1, s, v] = \Pr(R_v \rightarrow w_s)$ 

for each  $l = 2$  to  $n$  -- Length of span
    for each  $s = 1$  to  $n-l+1$  -- Start of span
        for each  $p = 1$  to  $l-1$  -- Partition of span
            for each production  $R_a \rightarrow R_b R_c$ 
                prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p, s, b] * P[l-p, s+p, c]$ 
                if  $P[p, s, b] > 0$  and  $P[l-p, s+p, c] > 0$  and  $P[1, s, a] < \text{prob\_splitting}$  then
                    set  $P[1, s, a] = \text{prob\_splitting}$ 
                    set  $back[1, s, a] = \langle p, b, c \rangle$ 

```

Update the table with the set of non-terminals that produce w_s (i.e., string of length 1)

How you choose to break up the words in the span according to different production rules



Probabilistic CYK

We often use 'S' for start

```

let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .
let  $P[n, n, r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.
let  $back[n, n, r]$  be an array of backpointing triples.

for each  $s = 1$  to  $n$ 
    for each unit production  $R_v \rightarrow w_s$ 
        set  $P[1, s, v] = \Pr(R_v \rightarrow w_s)$ 

for each  $l = 2$  to  $n$  -- Length of span
    for each  $s = 1$  to  $n-l+1$  -- Start of span
        for each  $p = 1$  to  $l-1$  -- Partition of span
            for each production  $R_a \rightarrow R_b R_c$ 
                prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p, s, b] * P[l-p, s+p, c]$ 
                if  $P[p, s, b] > 0$  and  $P[l-p, s+p, c] > 0$  and  $P[1, s, a] < \text{prob\_splitting}$  then
                    set  $P[1, s, a] = \text{prob\_splitting}$ 
                    set  $back[1, s, a] = \langle p, b, c \rangle$ 

```

Update the table with the set of non-terminals that produce w_s (i.e., string of length 1)

How you choose to break up the words in the span according to different production rules



Probabilistic CYK

We often use 'S' for start

```

let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .
let  $P[n, n, r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.
let  $back[n, n, r]$  be an array of backpointing triples.

for each  $s = 1$  to  $n$ 
    for each unit production  $R_v \rightarrow w_s$ 
        set  $P[1, s, v] = \Pr(R_v \rightarrow w_s)$ 

for each  $l = 2$  to  $n$  -- Length of span
    for each  $s = 1$  to  $n-l+1$  -- Start of span
        for each  $p = 1$  to  $l-1$  -- Partition of span
            for each production  $R_a \rightarrow R_b R_c$ 
                prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p, s, b] * P[l-p, s+p, c]$ 
                if  $P[p, s, b] > 0$  and  $P[l-p, s+p, c] > 0$  and  $P[1, s, a] < \text{prob\_splitting}$  then
                    set  $P[1, s, a] = \text{prob\_splitting}$ 
                    set  $back[1, s, a] = \langle p, b, c \rangle$ 

```

Update the table with the set of non-terminals that produce w_s (i.e., string of length 1)

How you choose to break up the words in the span according to different production rules



Probabilistic CYK

We often use 'S' for start

```

let the input be a string  $I$  consisting of  $n$  words:  $w_1 \dots w_n$ .
let the grammar contain  $r$  nonterminal symbols  $R_1 \dots R_r$ , with start symbol  $R_1$ .
let  $P[n, n, r]$  be an array of real numbers. Initialize all elements of  $P$  to zero.
let  $back[n, n, r]$  be an array of backpointing triples.

for each  $s = 1$  to  $n$ 
    for each unit production  $R_v \rightarrow w_s$ 
        set  $P[1, s, v] = \Pr(R_v \rightarrow w_s)$ 

for each  $l = 2$  to  $n$  -- Length of span
    for each  $s = 1$  to  $n-l+1$  -- Start of span
        for each  $p = 1$  to  $l-1$  -- Partition of span
            for each production  $R_a \rightarrow R_b R_c$ 
                prob_splitting =  $\Pr(R_a \rightarrow R_b R_c) * P[p, s, b] * P[l-p, s+p, c]$ 
                if  $P[p, s, b] > 0$  and  $P[l-p, s+p, c] > 0$  and  $P[1, s, a] < \text{prob\_splitting}$  then
                    set  $P[1, s, a] = \text{prob\_splitting}$ 
                    set  $back[1, s, a] = \langle p, b, c \rangle$ 

```

Update the table with the set of non-terminals that produce w_s (i.e., string of length 1)

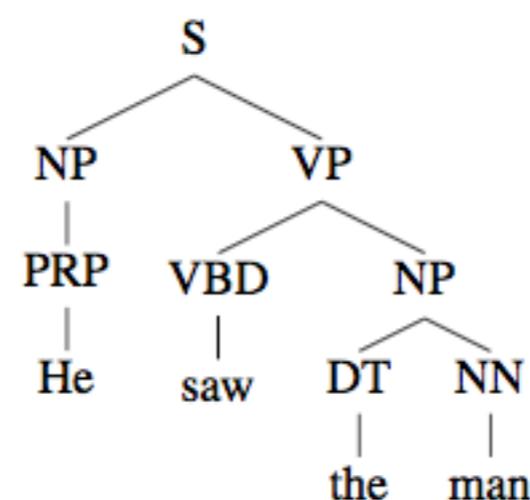
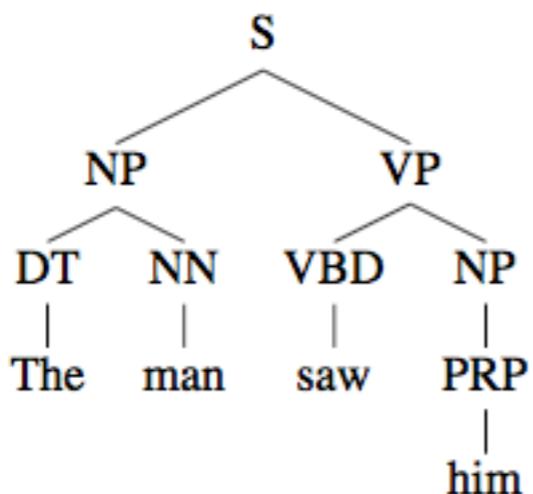
Only keep the highest probability parse

How you choose to break up the words in the span according to different production rules

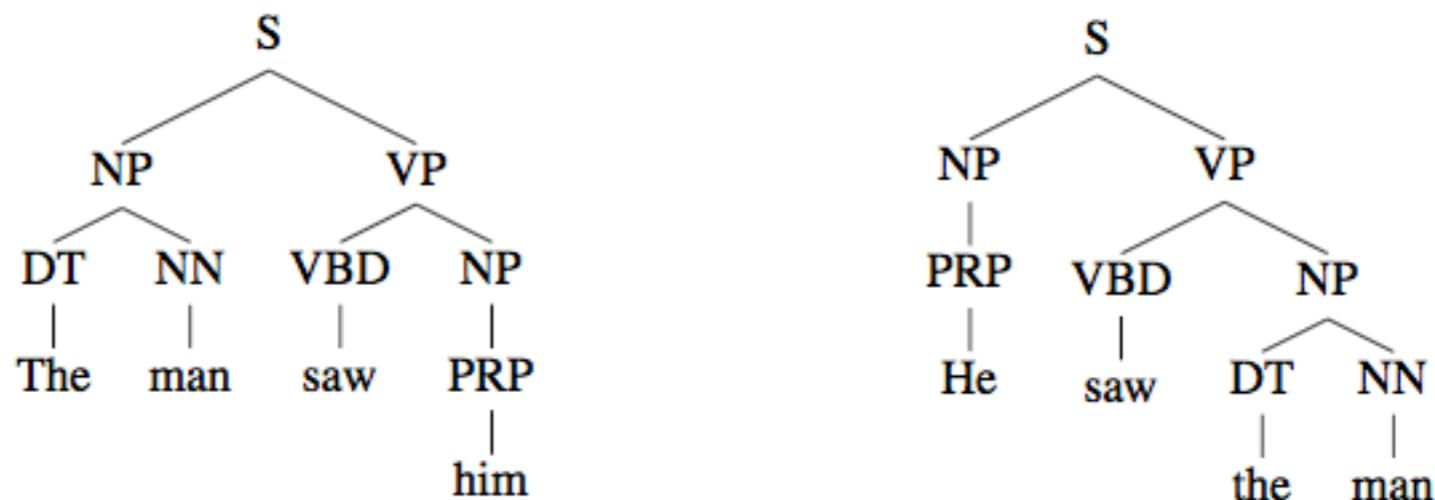


Problems with PCFGs

- Strong independence assumptions
 - Each production (e.g., $NP \rightarrow DT\ NN$) is **independent** of the rest of tree.
 - In real use, productions are strongly dependent on their place in the tree.



Problems with PCFGs



	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

Problems with PCFGs

	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

Maximum likelihood estimates
from Switchboard:

- $P(NP \rightarrow DT\ NN) = 0.28$
- $P(NP \rightarrow PRP) = 0.25$

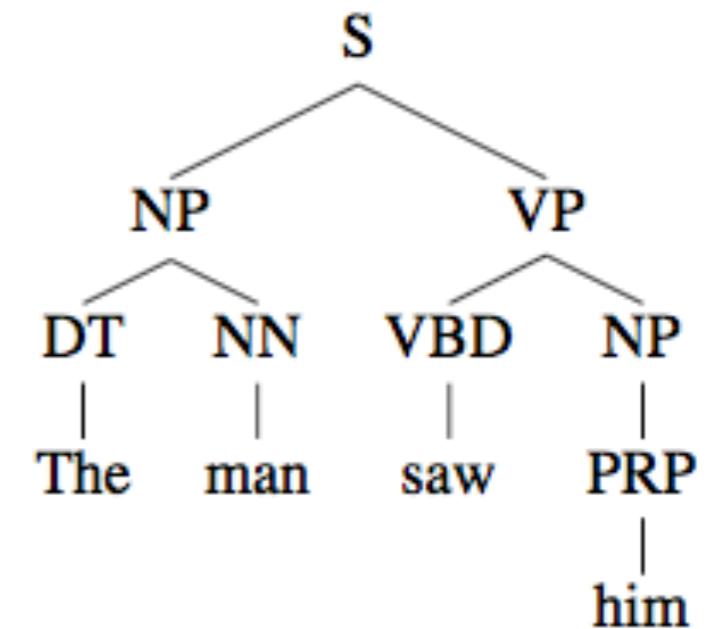
Splitting non-terminals

Rather than having a single rule for each non-terminal
 $P(NP \rightarrow DT\ NN)$, we can condition on some context
(Johnson 1998)

- $P_{\text{subject}}(NP \rightarrow DT\ NN)$
- $P_{\text{object}}(NP \rightarrow DT\ NN)$

Splitting non-terminals

- Subjects/objects are **structural relations** in phrase structure trees
- Subject = NP child of S
- Object = NP child of VP
child of S



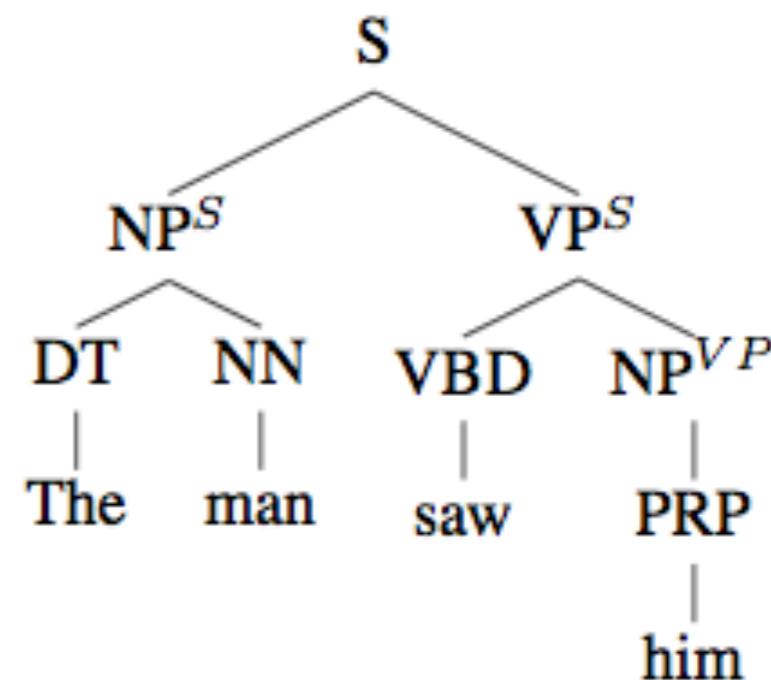
(For some subjects/objects; other rules
for embedded structures)

Parent annotation

We can encode context in a general way by annotating each node in a tree with its parent

This lets us learn different probabilities for:

- NPS (subjects)
- NPVP (objects)

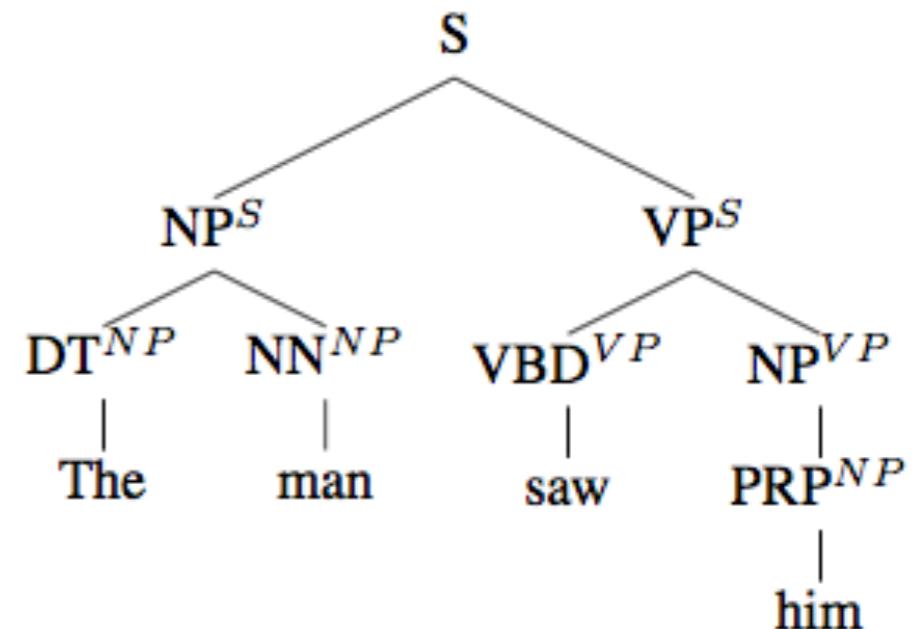


Parent annotation

We can also split the pre-terminal POS tags too
(Klein and Manning 2003)

This lets us learn different probabilities

- $P(RB^{VP} \rightarrow \text{not})$
- $P(RB^{NP} \rightarrow \text{not})$

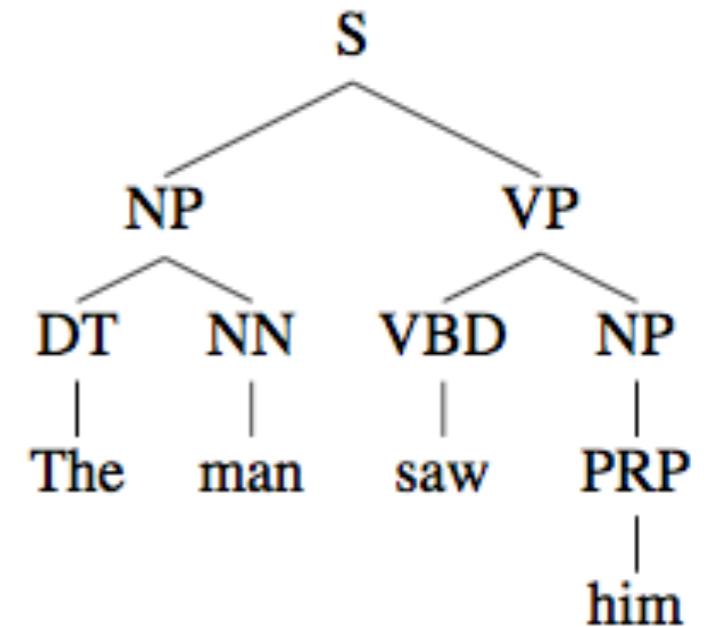


Splitting non-terminals

- Dramatically increases the size of the grammar → less training data for each production
- Modern approaches search for best splits that maximize the training data likelihood (Petrov et al 2006)

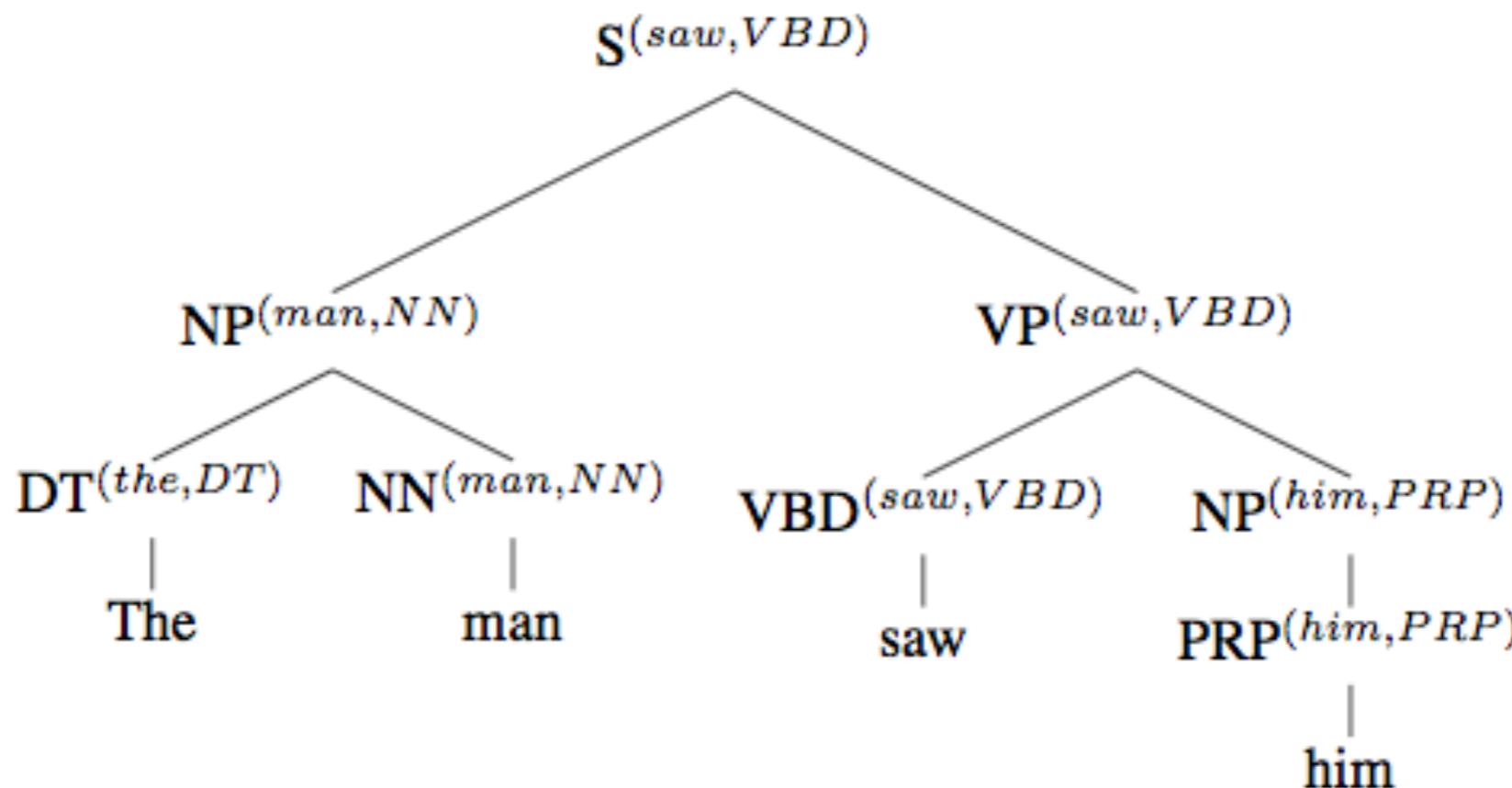
Problems with PCFGs

- Lexicon information in a PCFG has little influence on the overall parse structure
- $P(VBD \rightarrow \text{saw})$ — “saw” itself doesn’t influence the structure above it except through that pre-terminal.



Lexicalized PCFGs

- Annotate each node with its head + POS tag of head



Lexicalized PCFGs

- Annotate each node with its **head** + POS tag of head
- We can't have a rule for each fine-grained production — e.g. $P(S^{(\text{saw}, \text{VBD})} \rightarrow NP^{(\text{man}, \text{NN})} VP^{(\text{saw}, \text{VBD})})$
- Different models make different **independent assumptions** to make this quantity tractable (Collins 1999, Charniak 1997)

Phrase structure parsing

- Discriminative re-ranking (Charniak and Johnson 2005; McClosky et al. 2006)
- Parsing with compositional vector grammars (Socher et al. 2013)
- Parsing as sequence-to-sequence (Vinyals et al. 2015)
- Parsing with recurrent neural network grammars (Dyer et al. 2016)



Form W-4 (2014)

Purpose. Complete Form W-4 so that your employer can withhold the correct federal income tax from your pay. Consider completing a new Form W-4 each year and when your personal or financial situation changes.

Exemption from withholding. If you are exempt, complete **only** lines 1, 2, 3, 4, and 7 and sign the form to validate it. Your exemption for 2014 expires February 17, 2015. See Pub. 505, Tax Withholding and Estimated Tax.

Note. If another person can claim you as a dependent on his or her tax return, you cannot claim exemption from withholding if your income exceeds \$1,000 and includes more than \$350 of unearned income (for example, interest and dividends).

Exceptions. An employee may be able to claim exemption from withholding even if the employee is a dependent, if the employee:

- Is age 65 or older,
- Is blind, or
- Will claim adjustments to income; tax credits; or itemized deductions, on his or her tax return.

The exceptions do not apply to supplemental wages greater than \$1,000,000.

Basic instructions. If you are not exempt, complete the **Personal Allowances Worksheet** below. The worksheets on page 2 further adjust your withholding allowances based on itemized deductions, certain credits, adjustments to income, or two-earners/multiple jobs situations.

Complete all worksheets that apply. However, you may claim fewer (or zero) allowances. For regular wages, withholding must be based on allowances you claimed and may not be a flat amount or percentage of wages.

Head of household. Generally, you can claim head of household filing status on your tax return only if you are unmarried and pay more than 50% of the costs of keeping up a home for yourself and your dependent(s) or other qualifying individuals. See Pub. 501, Exemptions, Standard Deduction, and Filing Information, for information.

Tax credits. You can take projected tax credits into account in figuring your allowable number of withholding allowances. Credits for child or dependent care expenses and the child tax credit may be claimed using the **Personal Allowances Worksheet** below. See Pub. 505 for information on converting your other credits into withholding allowances.

Nonwage income. If you have a large amount of nonwage income, such as interest or dividends, consider making estimated tax payments using Form 1040-ES, Estimated Tax for Individuals. Otherwise, you may owe additional tax. If you have pension or annuity income, see Pub. 505 to find out if you should adjust your withholding on Form W-4 or W-4P.

Two earners or multiple jobs. If you have a working spouse or more than one job, figure the total number of allowances you are entitled to claim on all jobs using worksheets from only one Form W-4. Your withholding usually will be most accurate when all allowances are claimed on the Form W-4 for the highest paying job and zero allowances are claimed on the others. See Pub. 505 for details.

Nonresident alien. If you are a nonresident alien, see Notice 1392, Supplemental Form W-4 Instructions for Nonresident Aliens, before completing this form.

Check your withholding. After your Form W-4 takes effect, use Pub. 505 to see how the amount you are having withheld compares to your projected total tax for 2014. See Pub. 505, especially if your earnings exceed \$130,000 (Single) or \$180,000 (Married).

Future developments. Information about any future developments affecting Form W-4 (such as legislation enacted after we release it) will be posted at www.irs.gov/w4.

1

Personal Allowances Worksheet (Keep for your records.)

A	Enter "1" for yourself if no one else can claim you as a dependent	A
B	Enter "1" if: { • You are single and have only one job; or • You are married, have only one job, and your spouse does not work; or • Your wages from a second job or your spouse's wages (or the total of both) are \$1,500 or less. }	B
C	Enter "1" for your spouse . But, you may choose to enter "-0-" if you are married and have either a working spouse or more than one job. (Entering "-0-" may help you avoid having too little tax withheld.)	C
D	Enter number of dependents (other than your spouse or yourself) you will claim on your tax return	D
E	Enter "1" if you will file as head of household on your tax return (see conditions under Head of household above)	E
F	Enter "1" if you have at least \$2,000 of child or dependent care expenses for which you plan to claim a credit (Note. Do not include child support payments. See Pub. 503, Child and Dependent Care Expenses, for details.)	F
G	Child Tax Credit (including additional child tax credit). See Pub. 972, Child Tax Credit, for more information. <ul style="list-style-type: none">• If your total income will be less than \$65,000 (\$95,000 if married), enter "2" for each eligible child; then less "1" if you have three to six eligible children or less "2" if you have seven or more eligible children.• If your total income will be between \$65,000 and \$84,000 (\$95,000 and \$119,000 if married), enter "1" for each eligible child	G
H	Add lines A through G and enter total here. (Note. This may be different from the number of exemptions you claim on your tax return.) ► H	
For accuracy, complete all worksheets that apply.	{ • If you plan to itemize or claim adjustments to income and want to reduce your withholding, see the Deductions and Adjustments Worksheet on page 2. • If you are single and have more than one job or are married and you and your spouse both work and the combined earnings from all jobs exceed \$50,000 (\$20,000 if married), see the Two-Earners/Multiple Jobs Worksheet on page 2 to avoid having too little tax withheld.	

The other kind of parsing that
most people use in practice:
Dependency Parsing

Context-free grammar

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

Context-free grammar

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

non-terminals

Context-free grammar

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

NP → Det Nominal

non-terminals

Context-free grammar

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

NP → Det Nominal

NP → ProperNoun

non-terminals

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

non-terminals

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

non-terminals

lexicon/
terminals

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

non-terminals

lexicon/
terminals

Context-free grammar

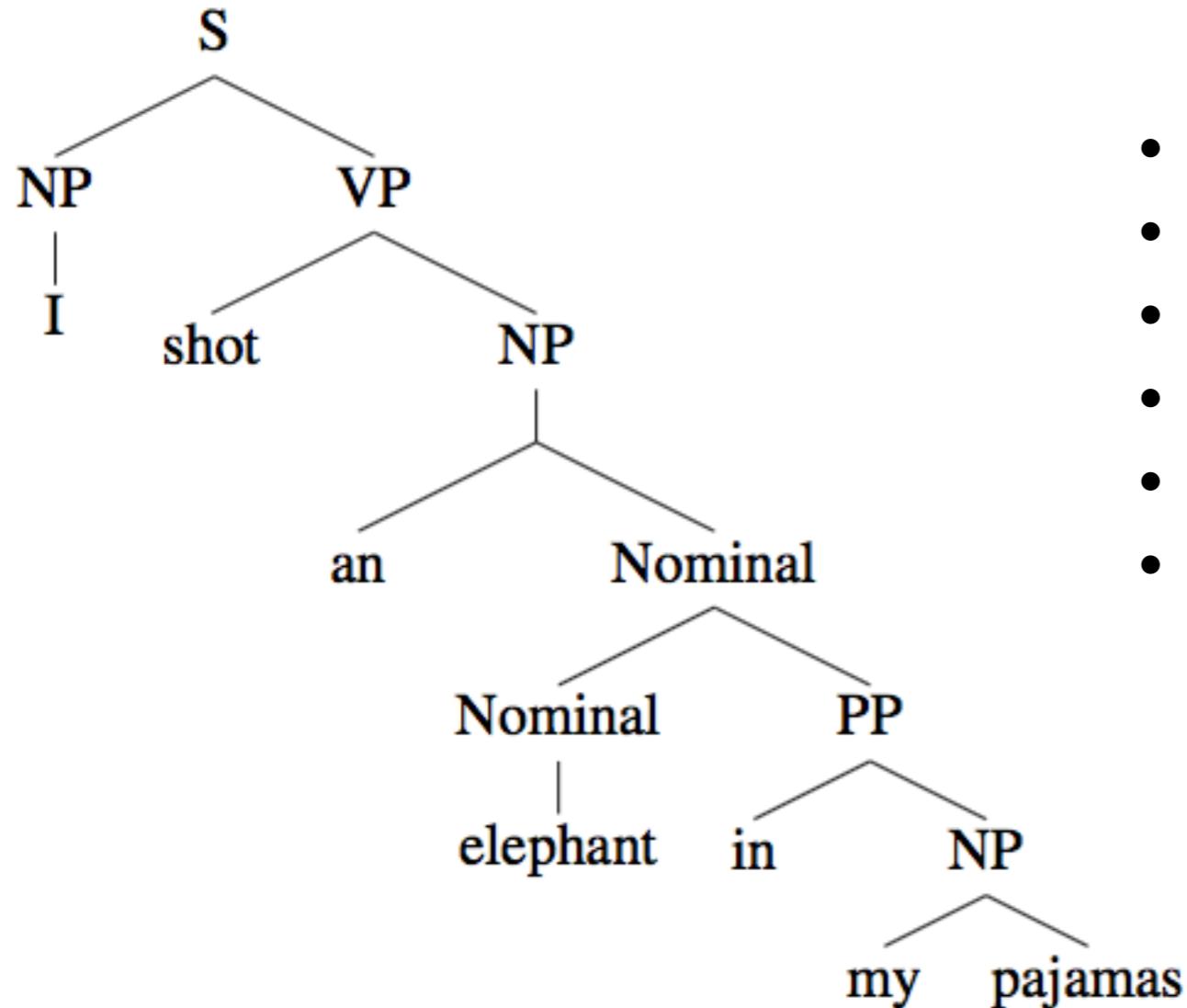
A context-free grammar defines how symbols in a language combine to form valid structures

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

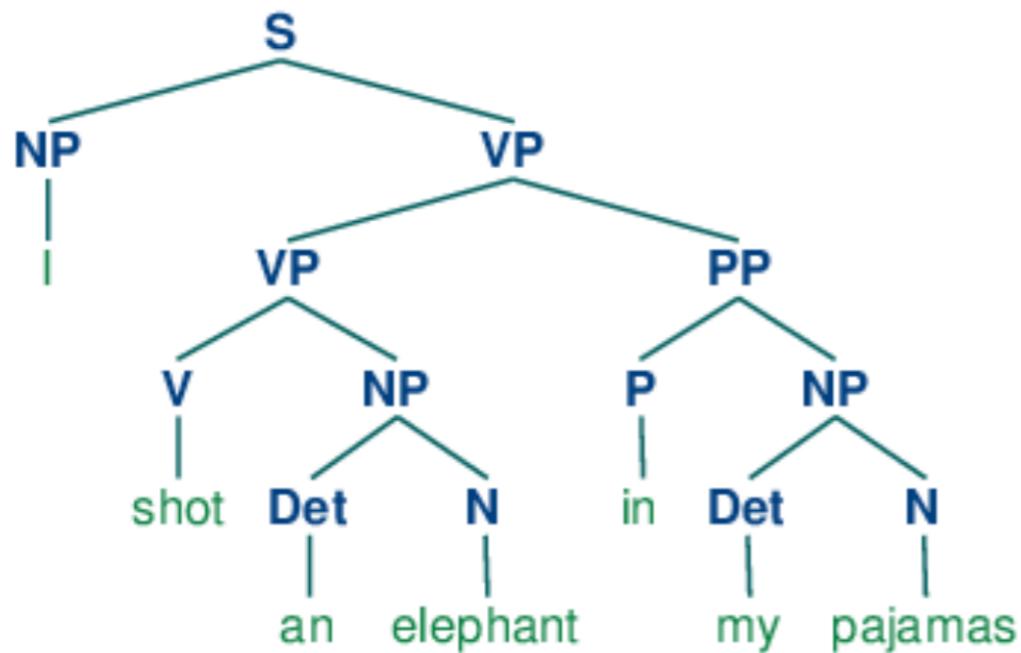
Each phrase could be replaced by another of the same type of constituent

Formalisms

Phrase structure grammar (Chomsky 1957)

Dependency grammar

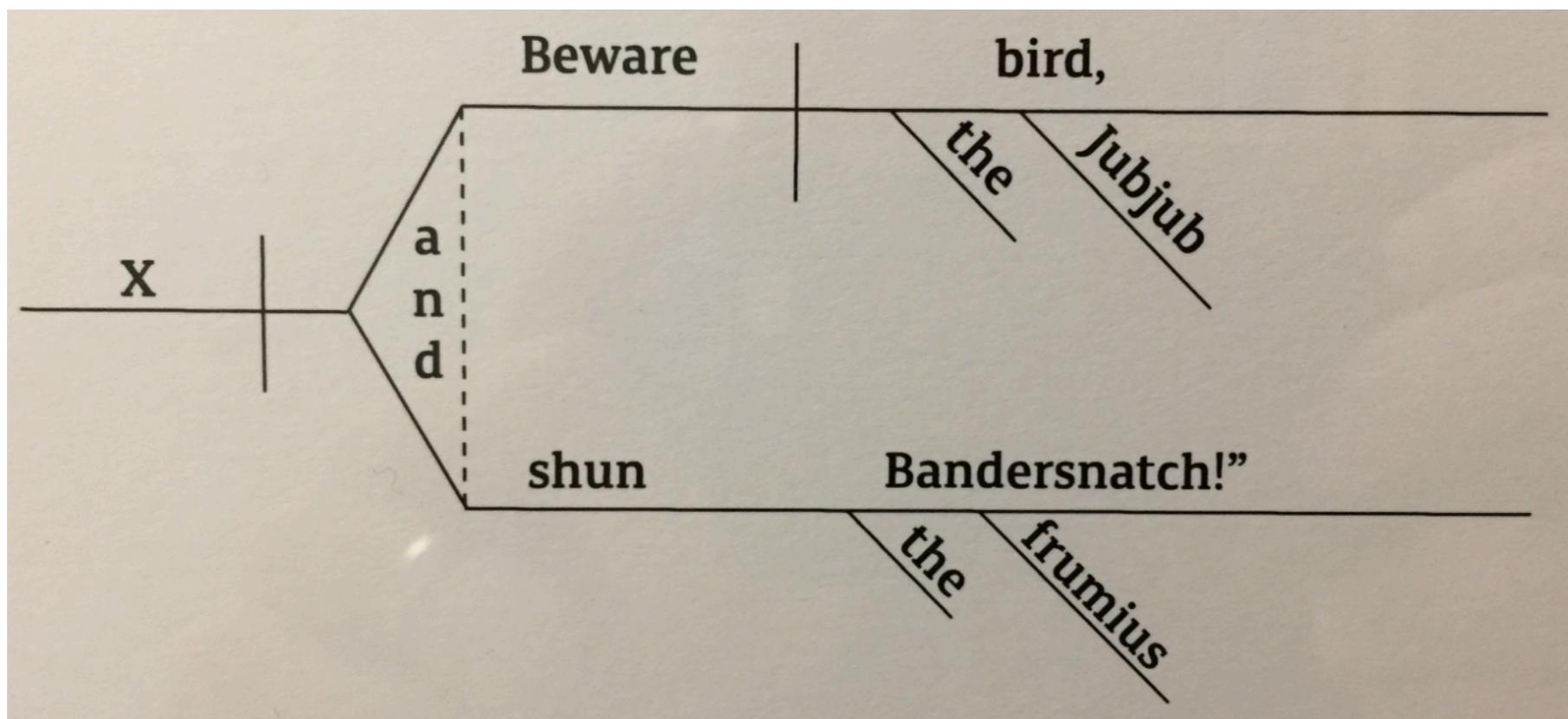
(Mel'čuk 1988; Tesnière 1959; Pāṇini)



Dependency syntax

- Pānini grammar of Sanskrit (ca. 5th-century BCE)
- Medieval theories of grammar (Covington 1984)
- Tesnière, *Éléments de syntaxe structurale* (1959)
- Mel'čuk, Dependency Syntax: Theory and Practice (1988)
- Sgall, Dependency-based formal description of language (1994)

Dependency syntax



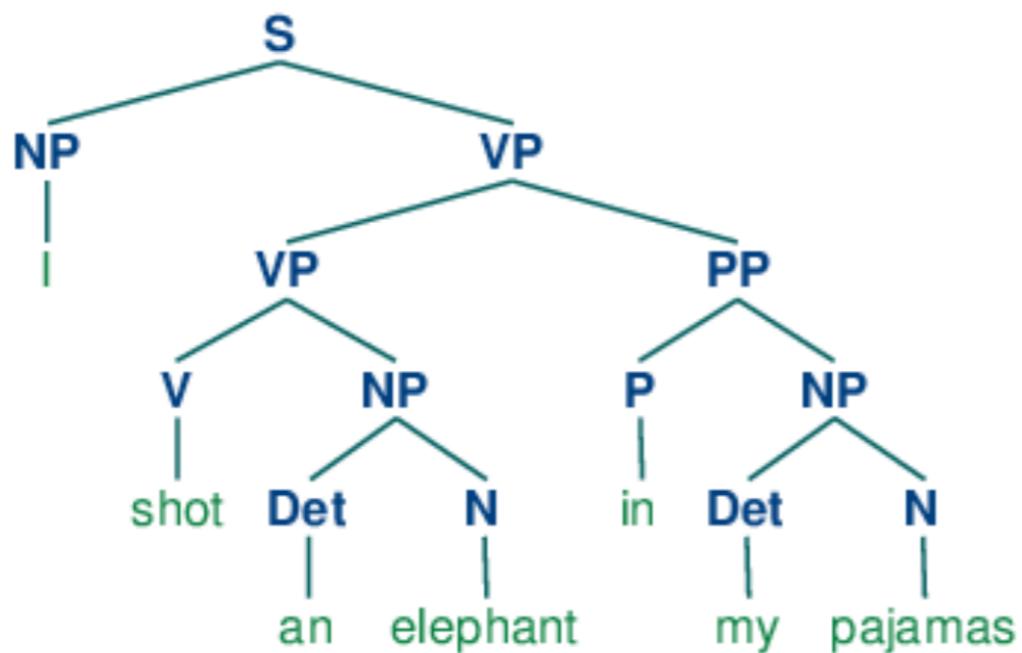
“Sentence diagramming”

Dependency syntax

- “Between the word and its neighbors, the mind perceives connections, the totality of which forms the structure of the sentence. The structural connections establish dependency relations between the words. Each connection in principle unites a superior and an inferior term.”

Tesnier 1959; Nivre 2005

Dependency syntax



- Dependency syntax doesn't have non-terminal structure like a CFG; words are directly linked to each other.

Dependency syntax

- Syntactic structure = **asymmetric, binary** relations between words.

Tesnier 1959; Nivre 2005

Dependency

- How do we decide which of a pair of words is the **head** and which is the **dependent**?

Dependency

Dependency

- Many (conflicting) frameworks:

Dependency

- Many (conflicting) frameworks:
 - Head determines the syntactic category of a construction

Dependency

- Many (conflicting) frameworks:
 - Head determines the syntactic category of a construction
 - Head is obligatory; dependents are optional

Dependency

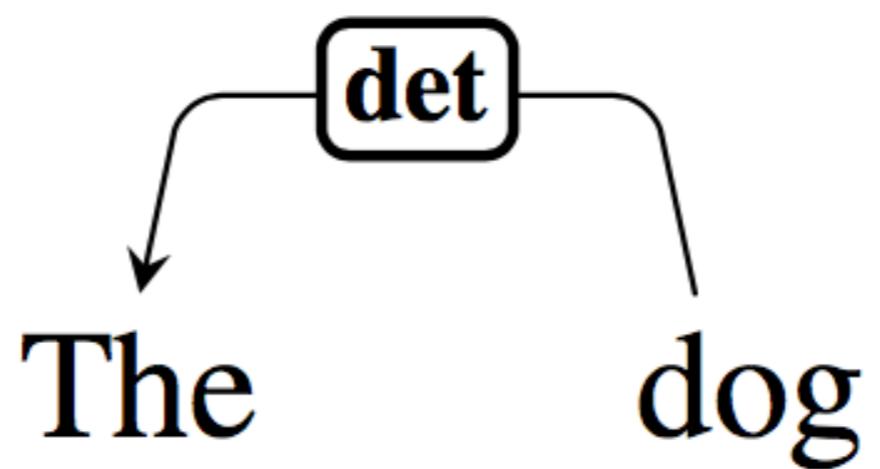
- Many (conflicting) frameworks:
 - Head determines the syntactic category of a construction
 - Head is obligatory; dependents are optional
 - Head selects dependents and determines whether the dependent is required

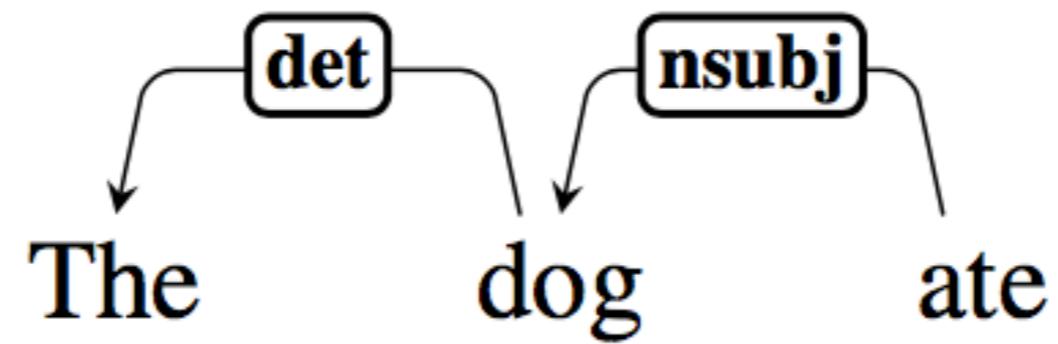
Dependency

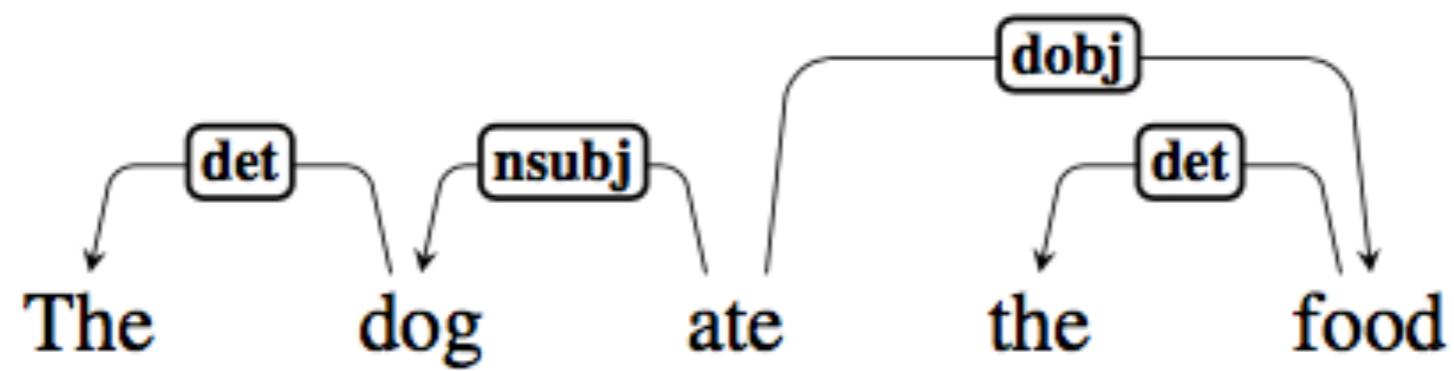
- Many (conflicting) frameworks:
 - Head determines the syntactic category of a construction
 - Head is obligatory; dependents are optional
 - Head selects dependents and determines whether the dependent is required
 - The form of the dependent depends on the head (e.g., agreement between nouns/verbs, adjectives/nouns)

Dependency

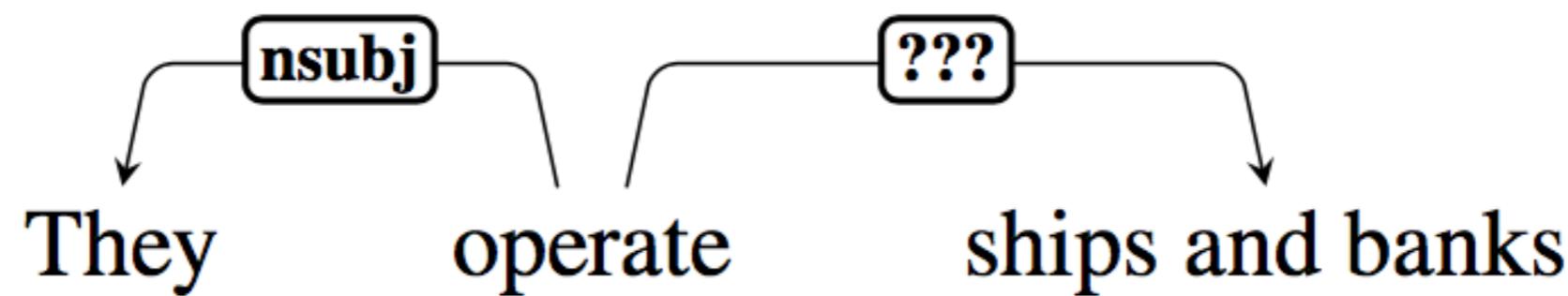
- Many (conflicting) frameworks:
 - Head determines the syntactic category of a construction
 - Head is obligatory; dependents are optional
 - Head selects dependents and determines whether the dependent is required
 - The form of the dependent depends on the head (e.g., agreement between nouns/verbs, adjectives/nouns)
 - The linear position of a dependent is specified with respect to the head.



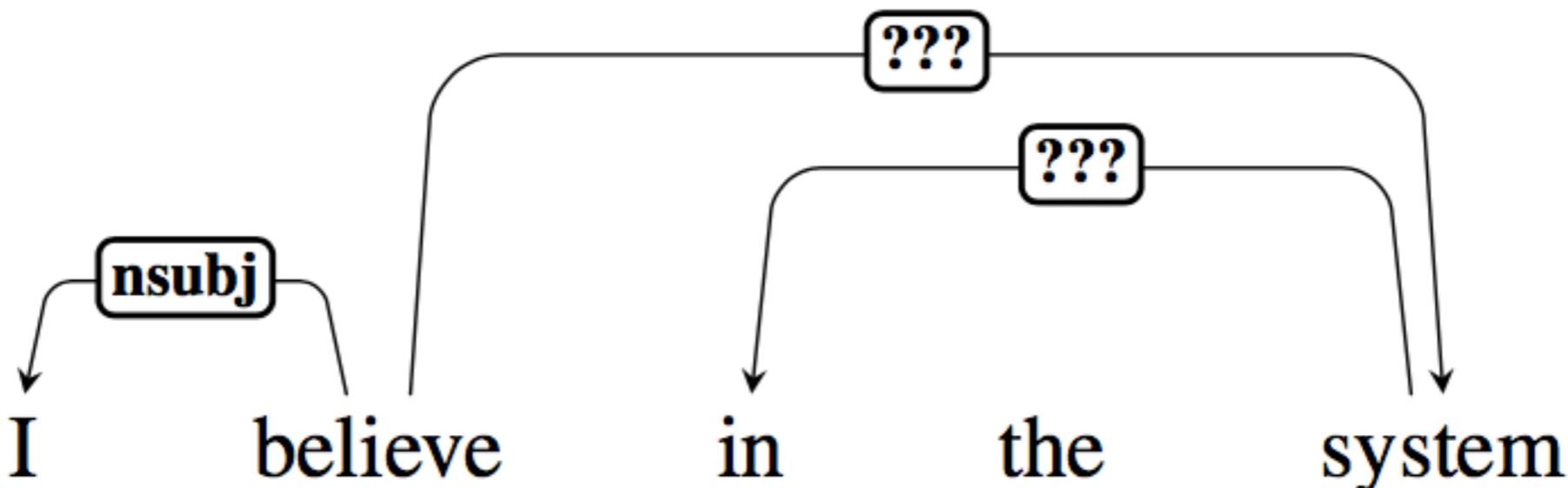
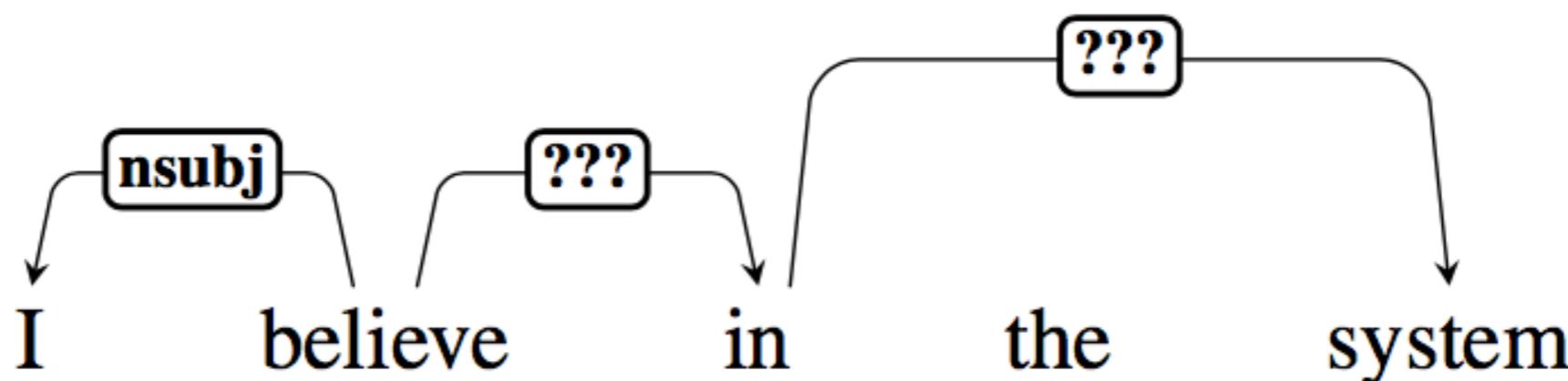




Coordination



Case marking prepositions



Trees

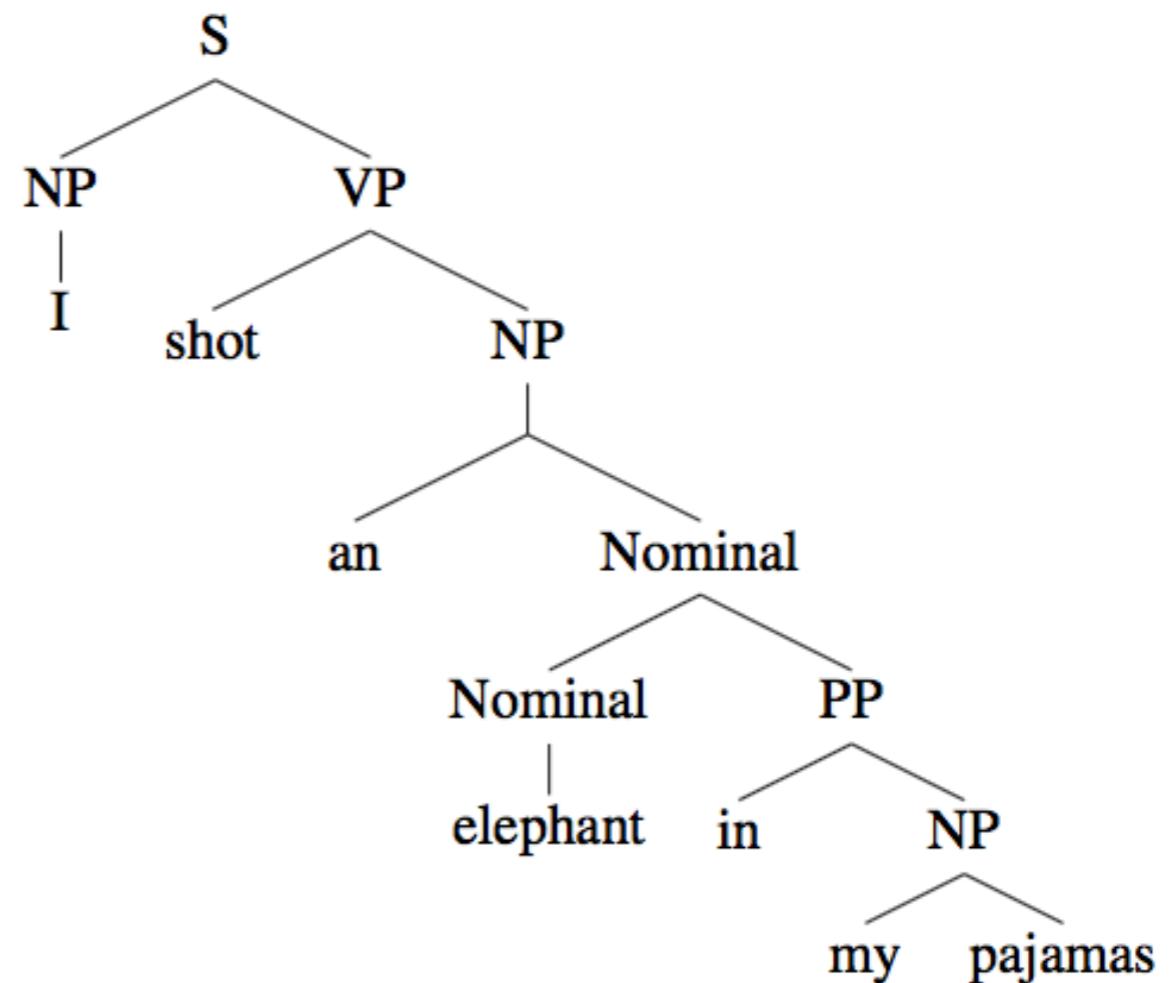
- A dependency structure is a directed graph $G = (V, A)$ consisting of a set of vertices V and arcs A between them. Typically constrained to form a **tree**:
 - Single root vertex with no incoming arcs
 - Every vertex has exactly one incoming arc except root (**single head constraint**)
 - There is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

Trees

- Unlike phrase-structure trees, dependency trees aren't tied to the linear order of the words in a sentence.
- Adding a constraint derived from the **linear order of words** in a sentence allows for more efficient parsing algorithms.

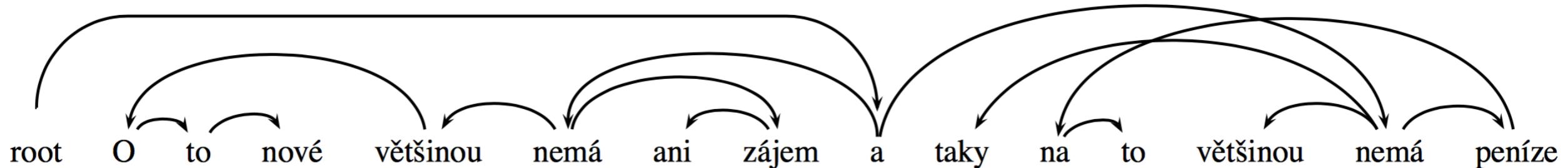
Word order

- Dependency relations belong to the structural order of a sentence, not the **linear order**.
- This is different from a phrase-structure tree, where the syntax is constrained by the linear order of the sentence (a different linear order yields a different parse tree).



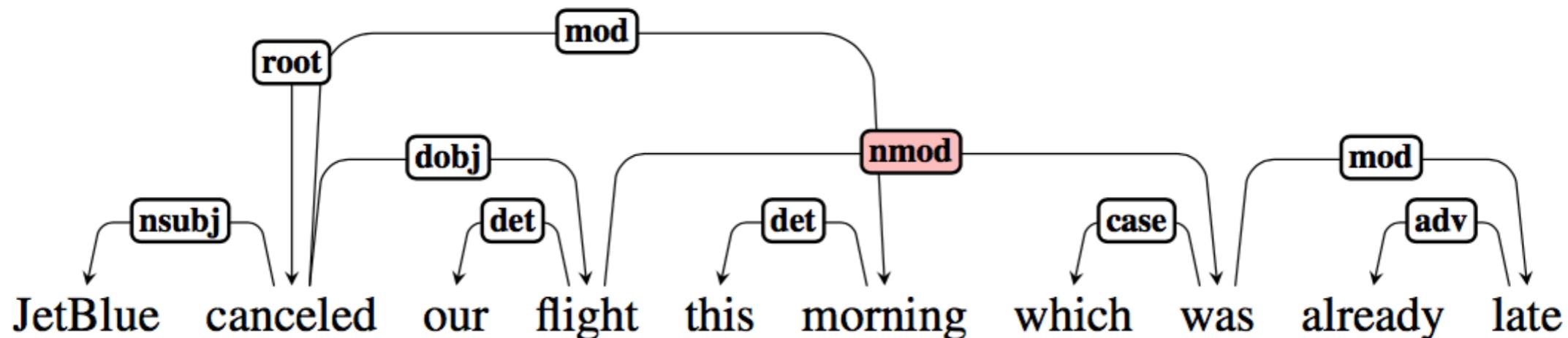
Tesnière 1959

Free word order

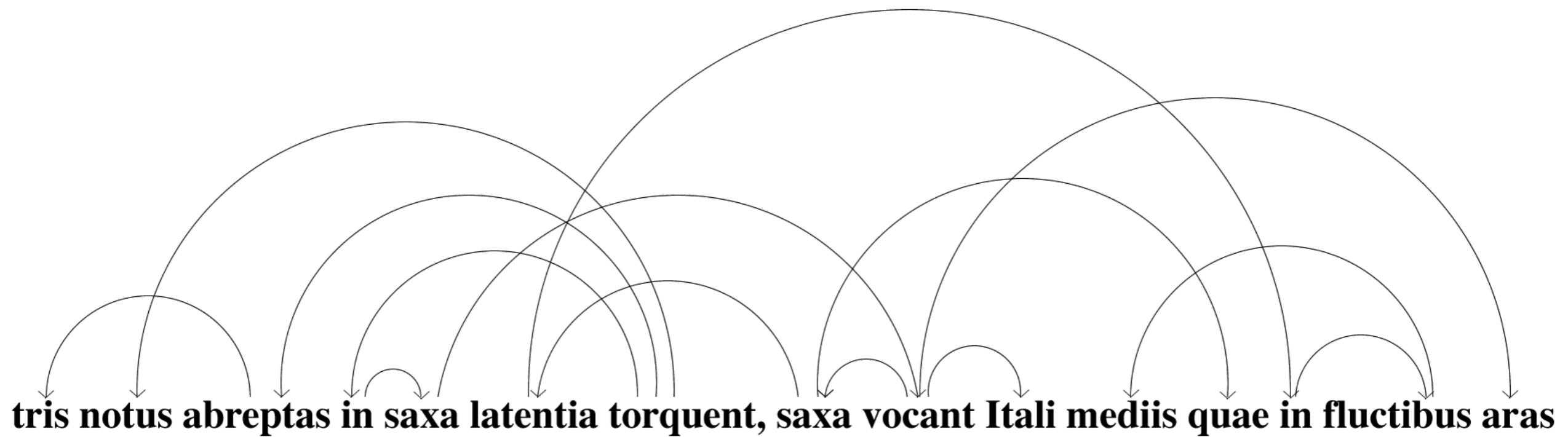


He is mostly not even interested in the new things and in most cases, he has no money for it either.

McDonald et al. 2006

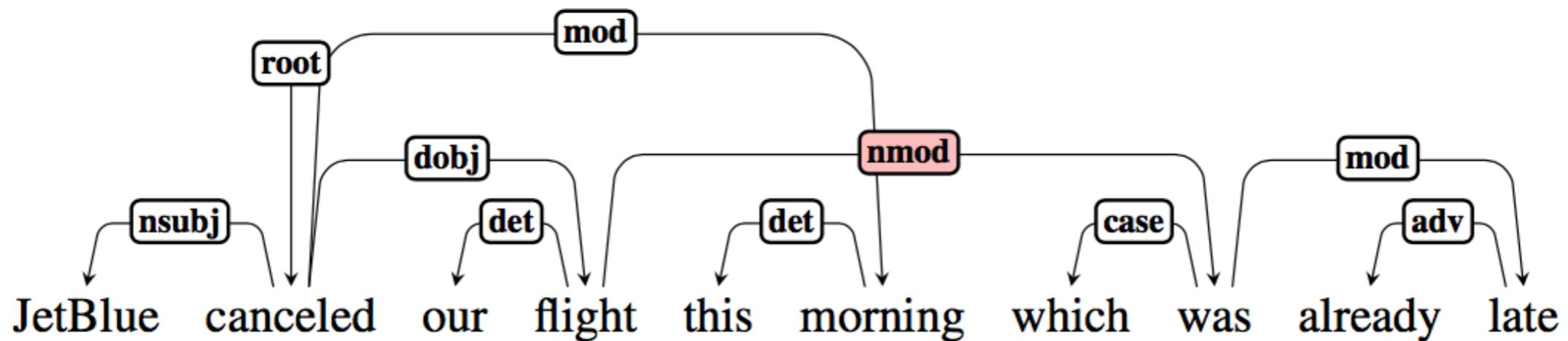


Free word order



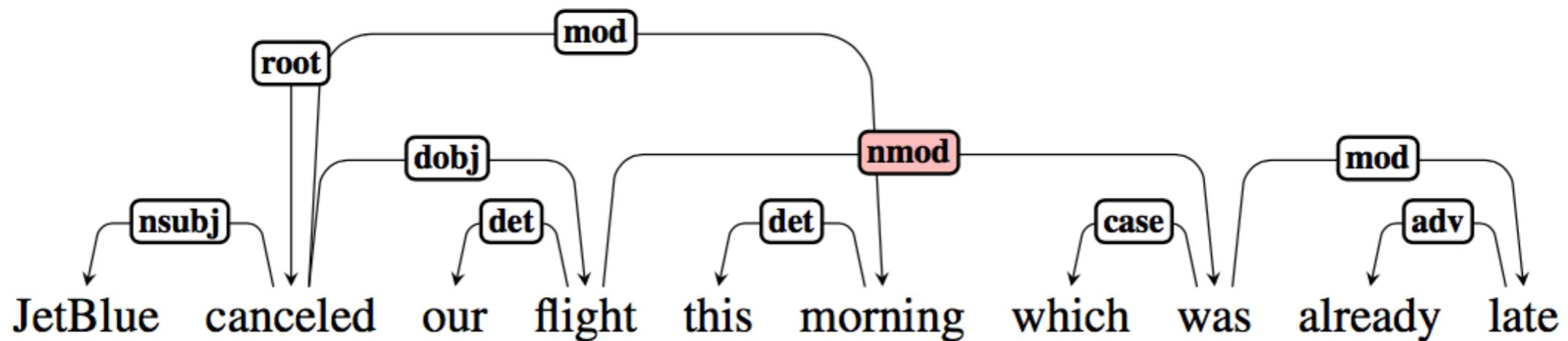
Projectivity

- 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.



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.



The nmod arc from “flight” to “was” passes over “this” and “morning” which are not linked to “flight”

Dependencies vs constituents

Covington 2001

Dependencies vs constituents

- Dependency links are closer to semantic relationships; no need to infer the relationships from the structure of a tree

Dependencies vs constituents

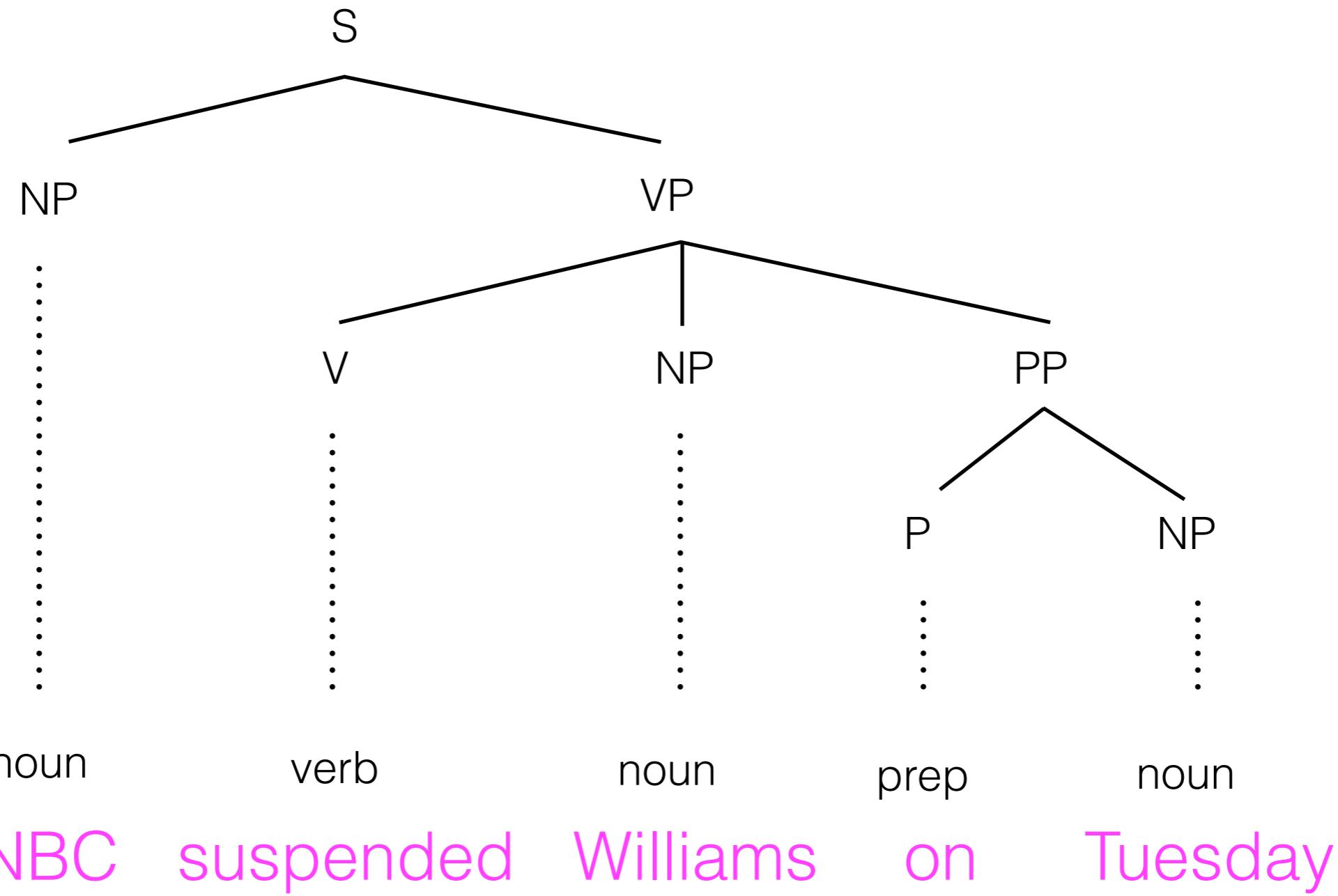
- Dependency links are closer to semantic relationships; no need to infer the relationships from the structure of a tree
- A dependency tree contains one edge for each word, no intermediate hidden structures that also need to be learned for parsing.

Dependencies vs constituents

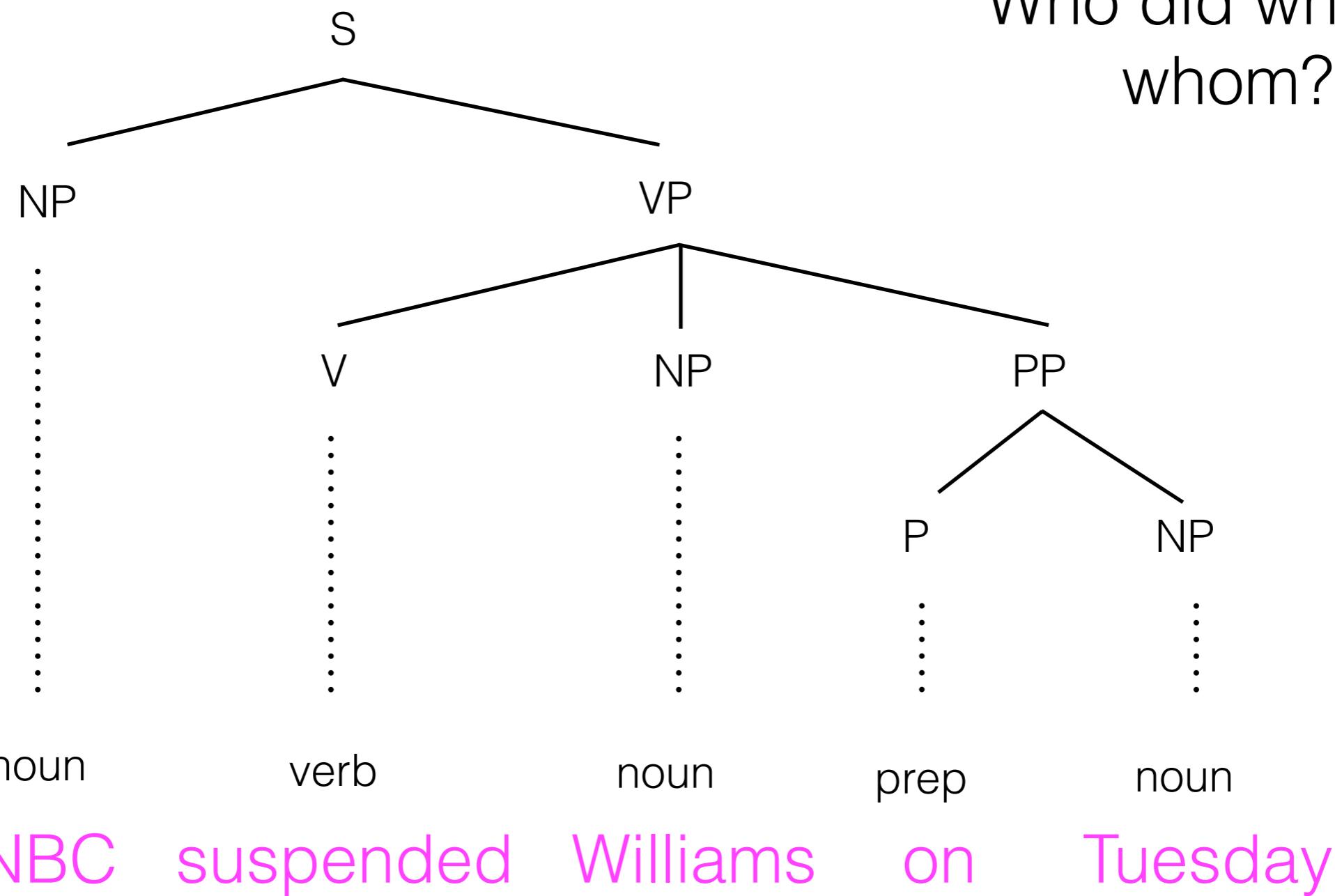
- Dependency links are closer to semantic relationships; no need to infer the relationships from the structure of a tree
- A dependency tree contains one edge for each word, no intermediate hidden structures that also need to be learned for parsing.
- Easier to represent languages with free word order.

NBC suspended Williams on Tuesday

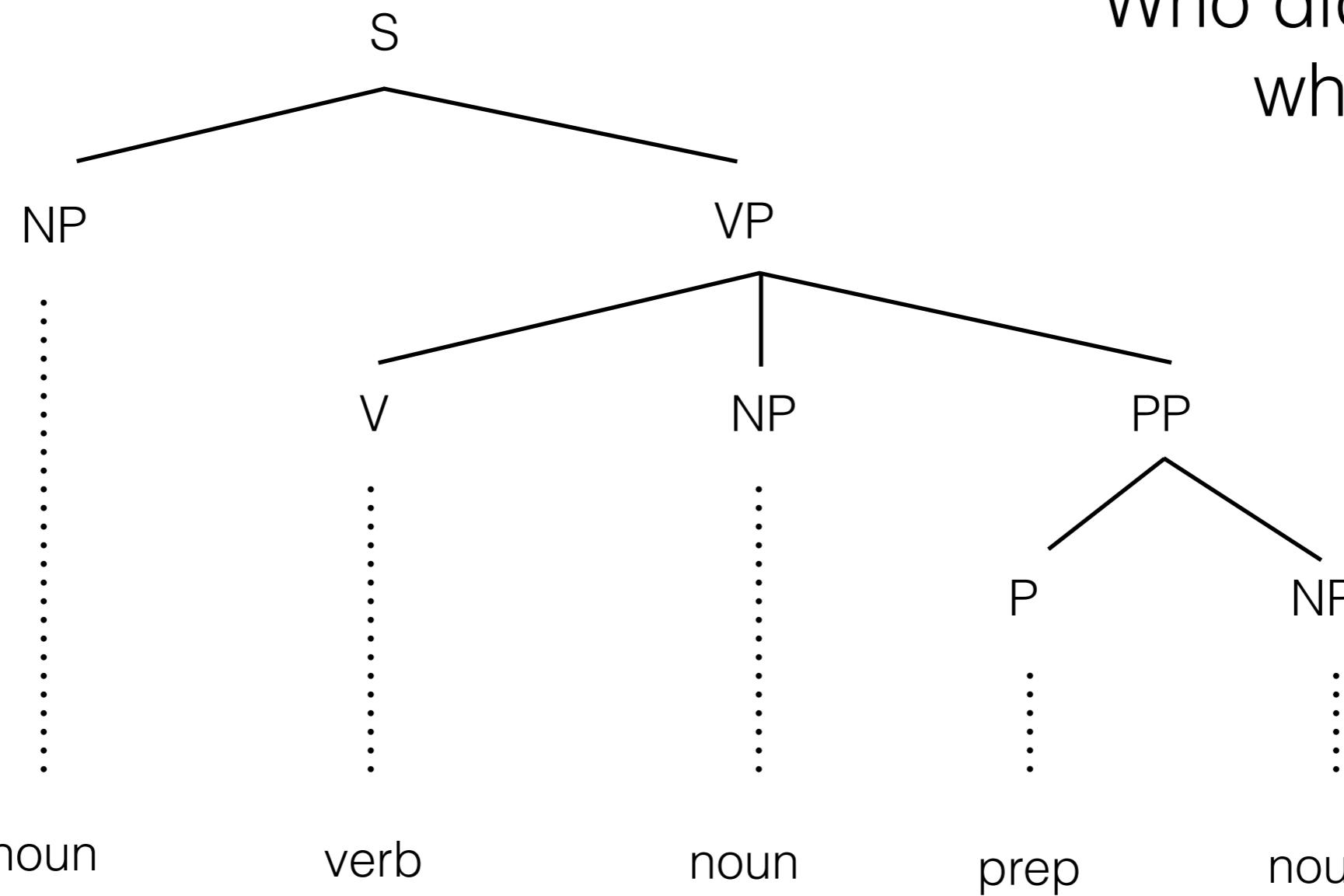
noun verb noun prep noun
NBC suspended Williams on Tuesday



Who did what to whom?

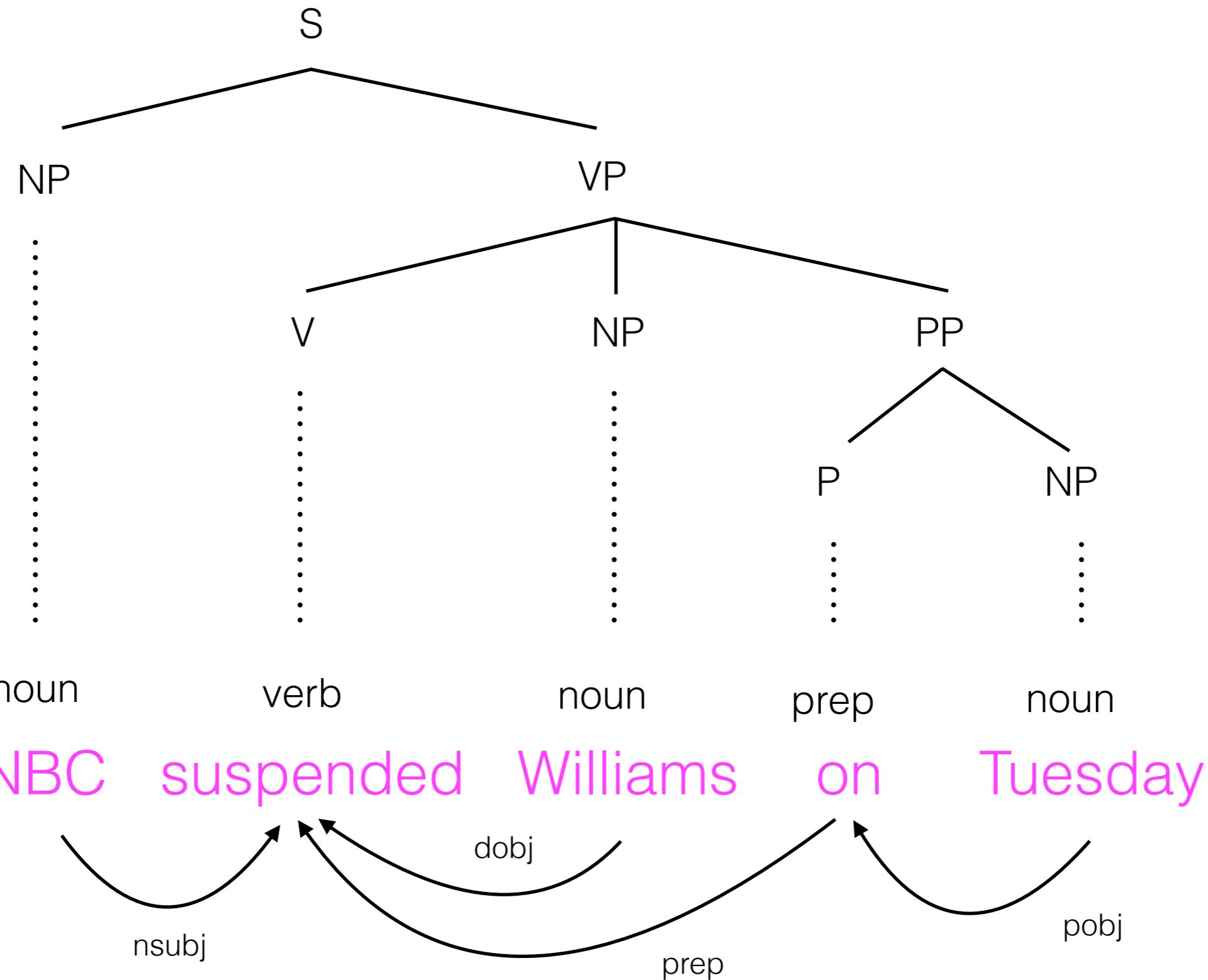


Who did what to whom?



subject: S → NP VP

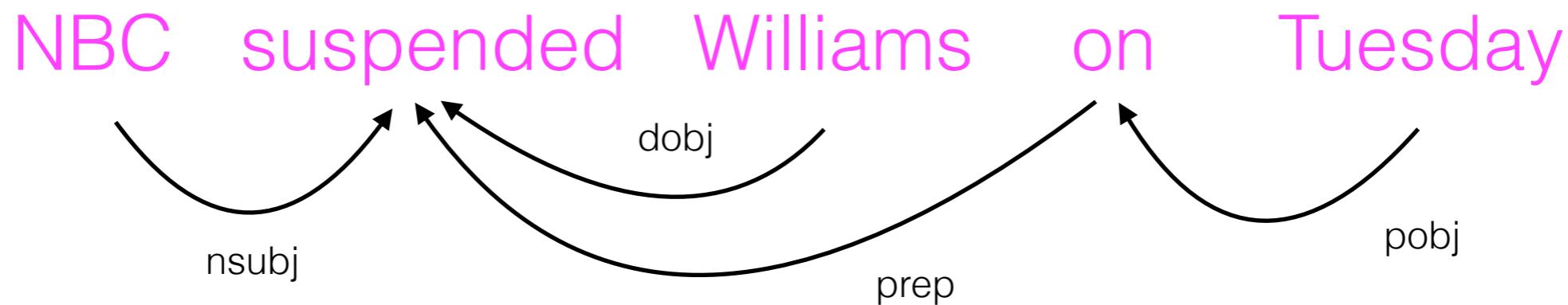
direct object: $S \rightarrow NP$ ($VP \rightarrow \dots NP \dots$)



Dependency grammar

Captures binary relations between words

- nsubj(NBC, suspended)
- dobj(Williams, suspended)



Data

- NELL SVO triples
(604 million
nsubj+dobj
relations from 230B
words on the web

police	found	five .030 bullets	1
police	found	seven dead rebels	3
police	found	two hidden cameras	2
police	found	wanders lover	1
police	found	211 pounds	4
police	found	Marcia	3
police	found	bank draft	1
police	found	diskette	2
police	found	five marijuana plants	3
police	found	items used	1
police	found	judge	5

Dependency-Based Word Embeddings

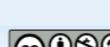
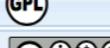
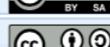
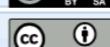
Levy & Goldberg,
ACL 2014

<http://irsrv2.cs.biu.ac.il:9998/>

Target Word	BoW5	BoW2	DEPS
batman	nightwing aquaman catwoman superman manhunter	superman superboy aquaman catwoman batgirl	superman superboy supergirl catwoman aquaman
hogwarts	dumbledore hallows half-blood malfoy snape	evernight sunnydale garderobe blandings collinwood	sunnydale collinwood calarts greendale millfield
turing	nondeterministic non-deterministic computability deterministic finite-state	non-deterministic finite-state nondeterministic buchi primality	pauling hotelling heting lessing hamming
florida	gainesville fla jacksonville tampa lauderdale	fla alabama gainesville tallahassee texas	texas louisiana georgia california carolina
object-oriented	aspect-oriented smalltalk event-driven prolog domain-specific	aspect-oriented event-driven objective-c dataflow 4gl	event-driven domain-specific rule-based data-driven human-centered
dancing	singing dance dances dancers tap-dancing	singing dance dances breakdancing clowning	singing rapping breakdancing miming busking

Universal Dependencies

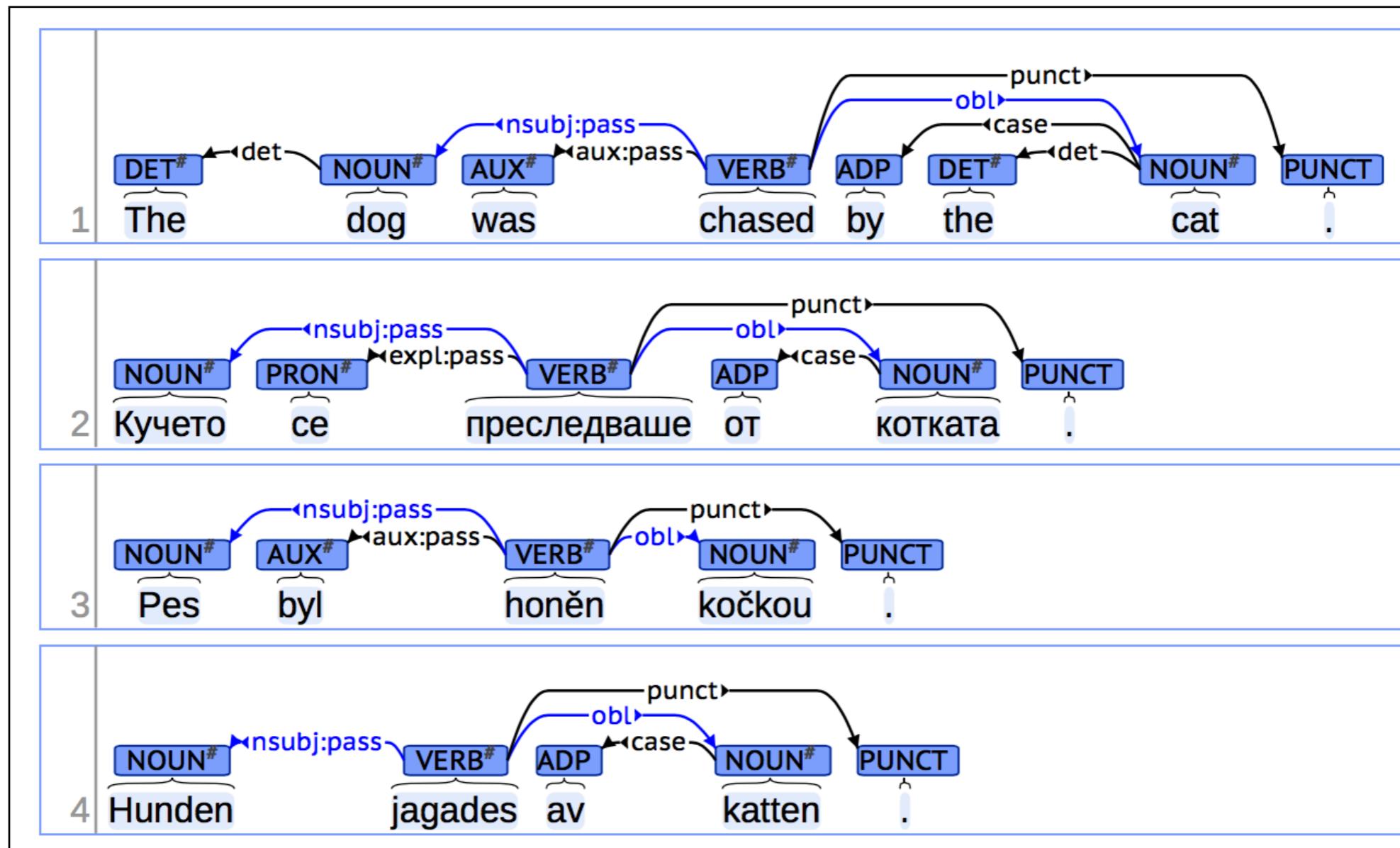
UD Treebanks

▶  Afrikaans	49K	L(F)	-				 
▶  Ancient Greek	202K	L(F)			✓		
▶  Ancient Greek-PROIEL	211K	L(F)	-		✓		 
▶  Arabic	242K	L(F)	-		✓		
▶  Arabic-NYUAD	629K	L(F)	-		✓		
▶  Arabic-PUD	20K	L(F)	-				 
▶  Basque	121K	L(F)			✓		 
▶  Belarusian	8K	L(F)	-		✓		
▶  Bulgarian	156K	L(F)			✓		  
▶  Buryat	10K	L(F)	-				  
▶  Catalan	530K	L(F)			✓		
▶  Chinese	123K	L(F)			✓		
▶  Chinese-CFL	7K	L					
▶  Chinese-PUD	21K	F	-				 
▶  Coptic	4K	L(F)			✓		  
▶  Croatian	197K	L(F)	-		✓		  
▶  Czech	1,503K	L(F)			✓		

Universal Dependencies

- Developing cross-linguistically consistent treebank annotation for many languages
- Goals:
 - Facilitating multilingual parser development
 - Cross-lingual learning
 - Parsing research from a language typology perspective.

Universal Dependencies





Transition-Based Parsing

(Also known as
Shift-Reduce Parsing)

Transition-based parsing

- Basic idea: parse a sentence into a dependency 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

Configuration

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

The Stack Data Structure

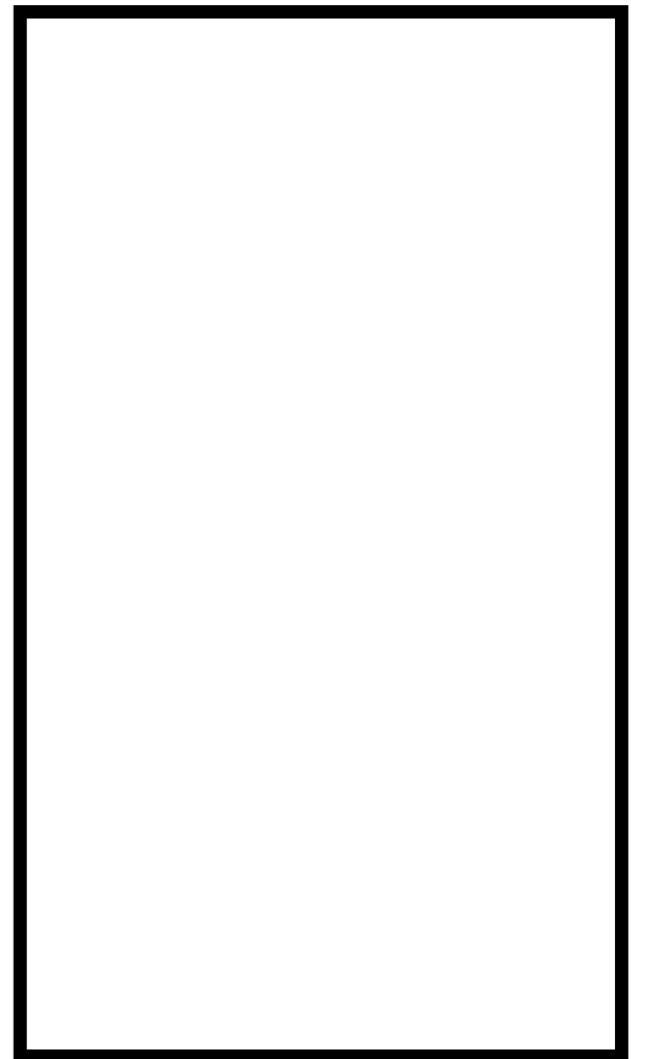
The Stack Data Structure

- Like a list but with only three operations:

The Stack Data Structure

- Like a list but with only three operations:

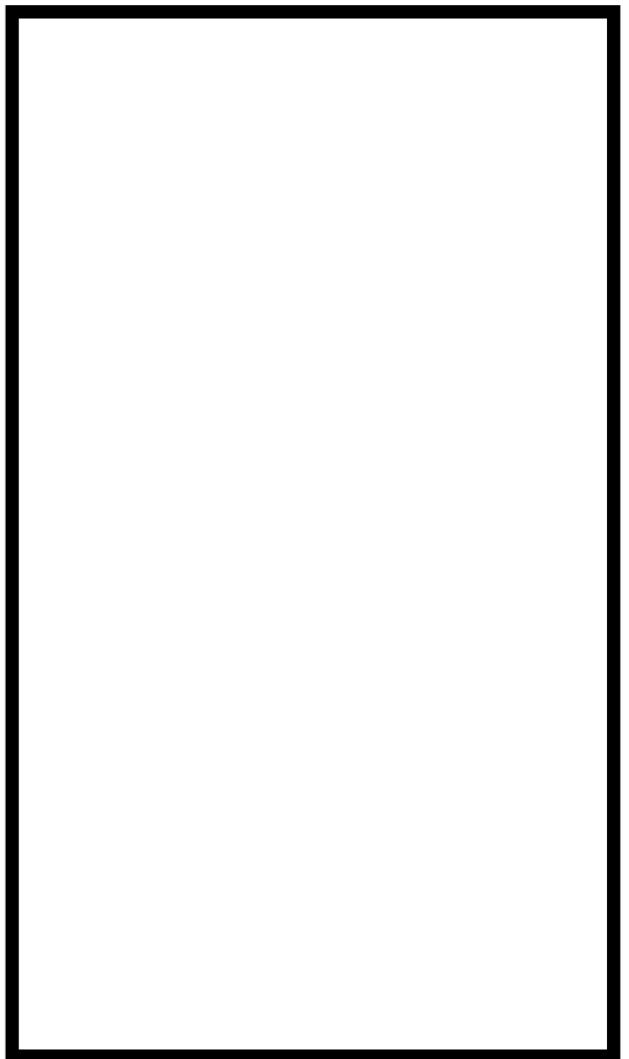
the stack



The Stack Data Structure

- Like a list but with only three operations:
 - Push — adds an item to the top of the stack

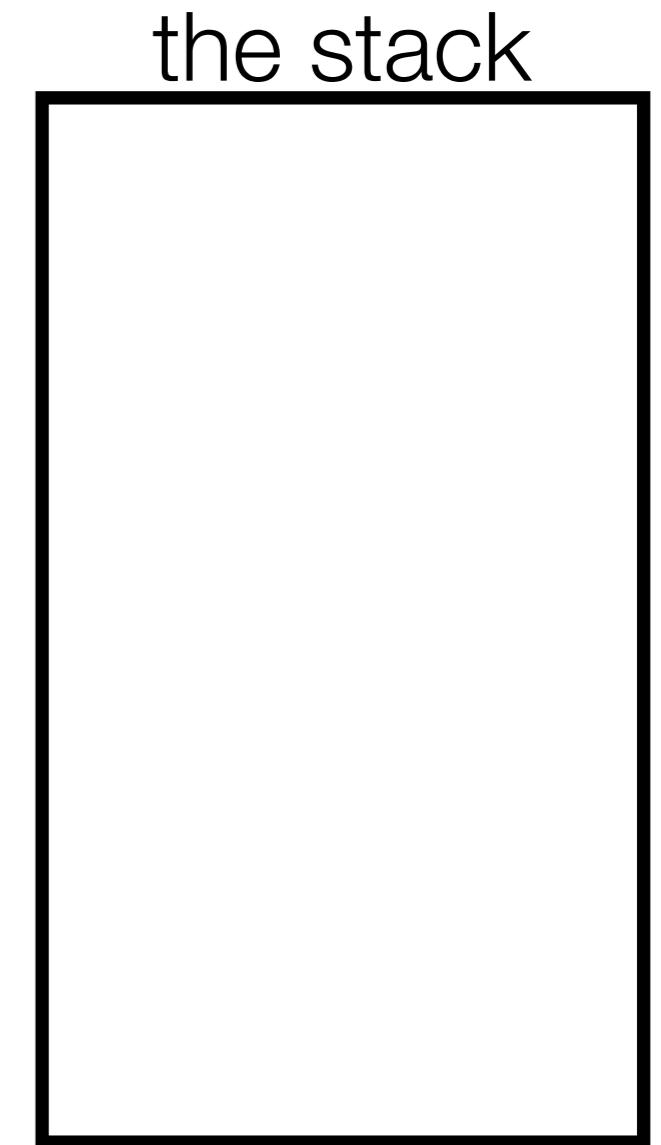
the stack



The Stack Data Structure

- Like a list but with only three operations:
 - Push — adds an item to the top of the stack

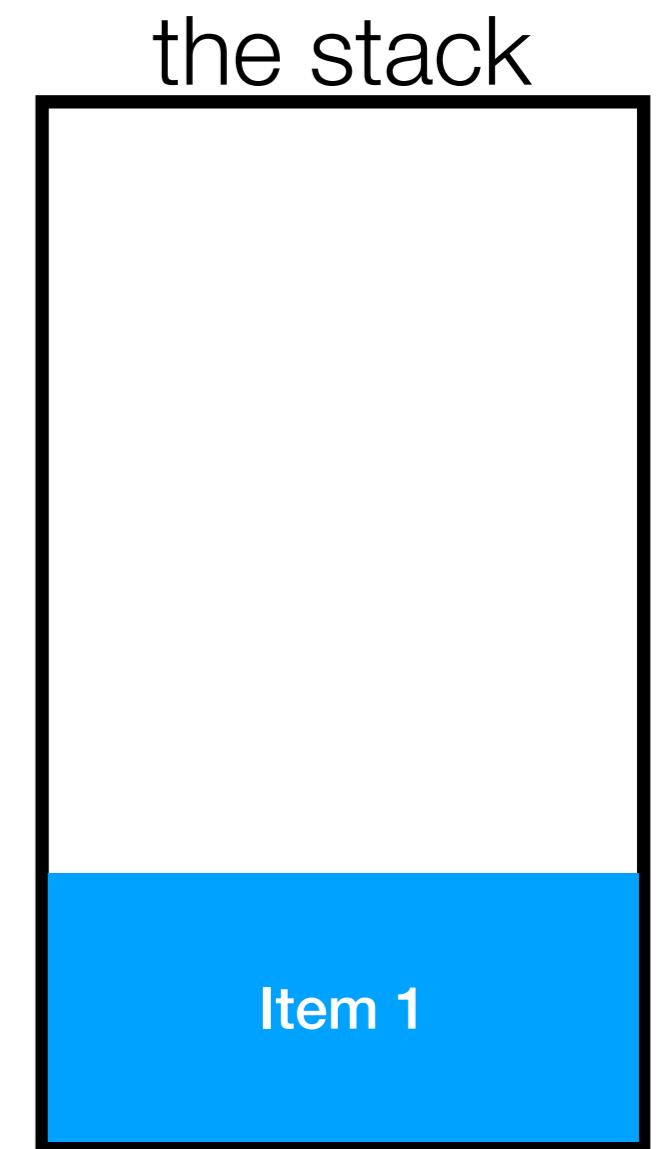
Push (Item 1)



The Stack Data Structure

- Like a list but with only three operations:
 - Push — adds an item to the top of the stack

Push (Item 1)

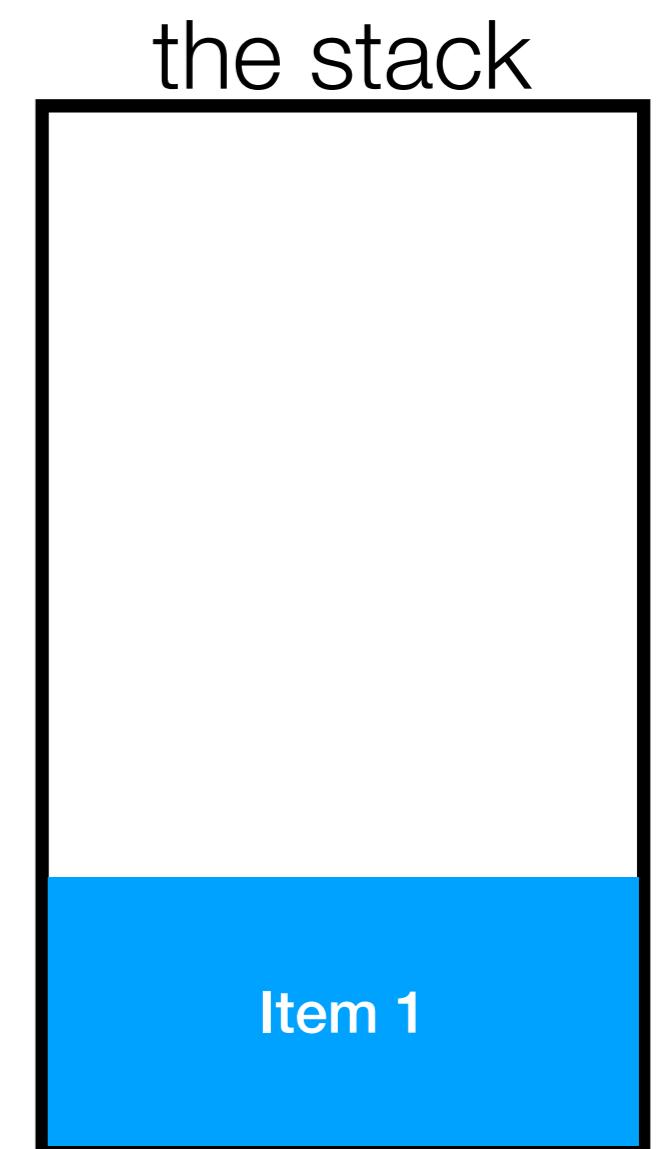


The Stack Data Structure

- Like a list but with only three operations:
 - Push — adds an item to the top of the stack

Push (Item 1)

Push (Item 2)

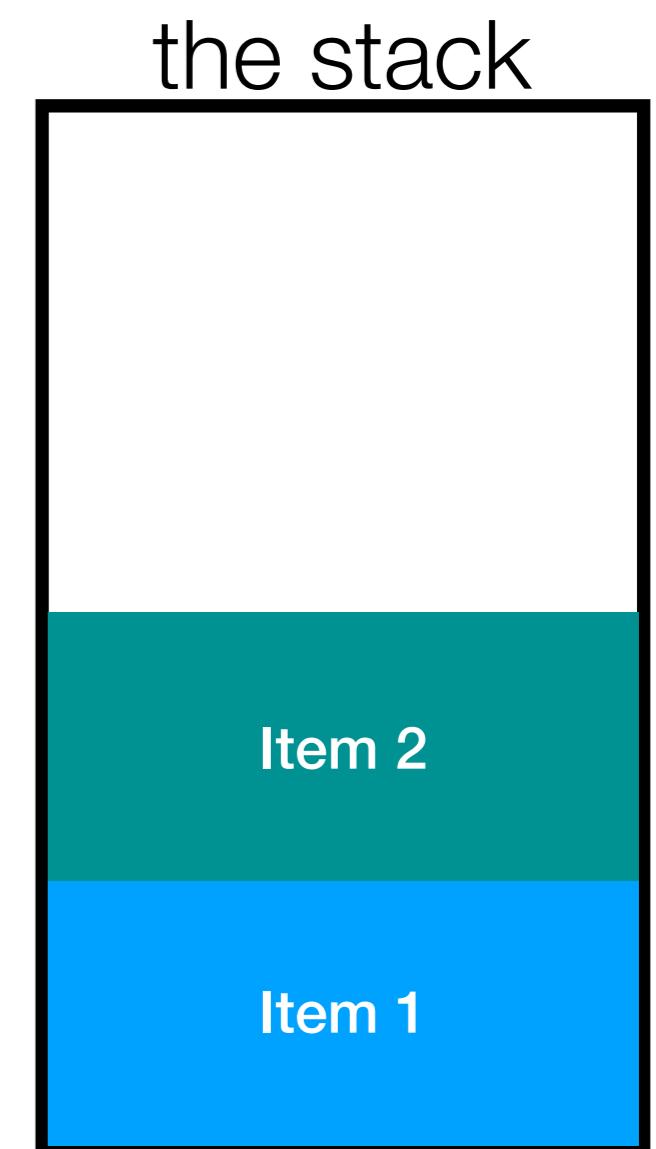


The Stack Data Structure

- Like a list but with only three operations:
 - Push — adds an item to the top of the stack

Push (Item 1)

Push (Item 2)

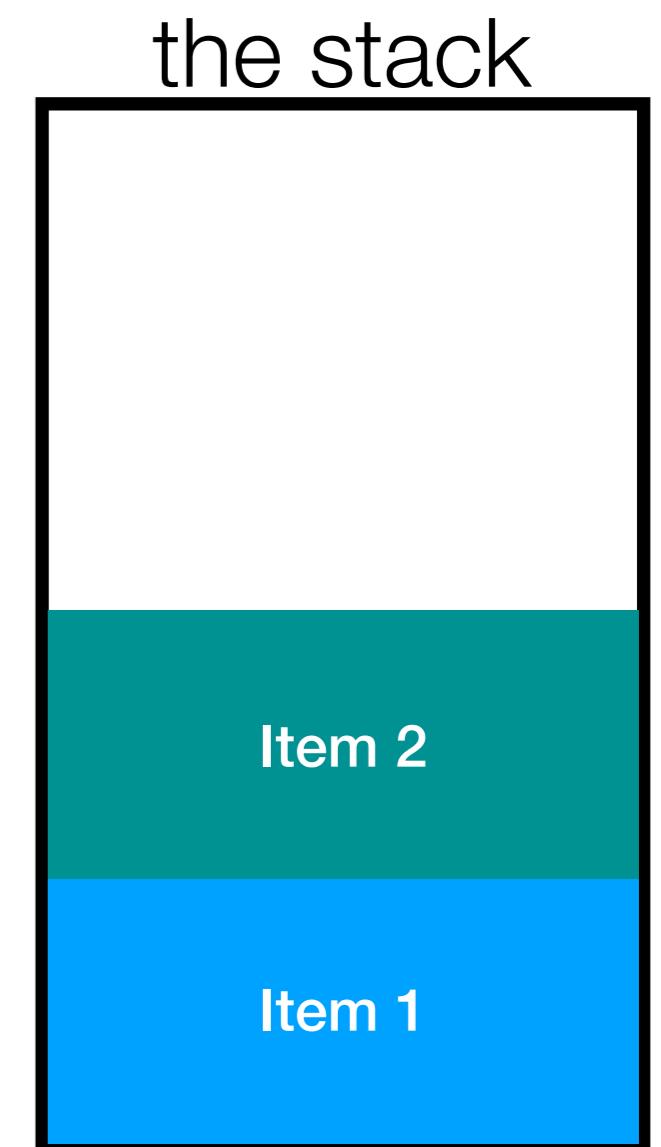


The Stack Data Structure

- Like a list but with only three operations:
 - Push — adds an item to the top of the stack
 - Pop — removes the item on the top of the stack

Push (Item 1)

Push (Item 2)



The Stack Data Structure

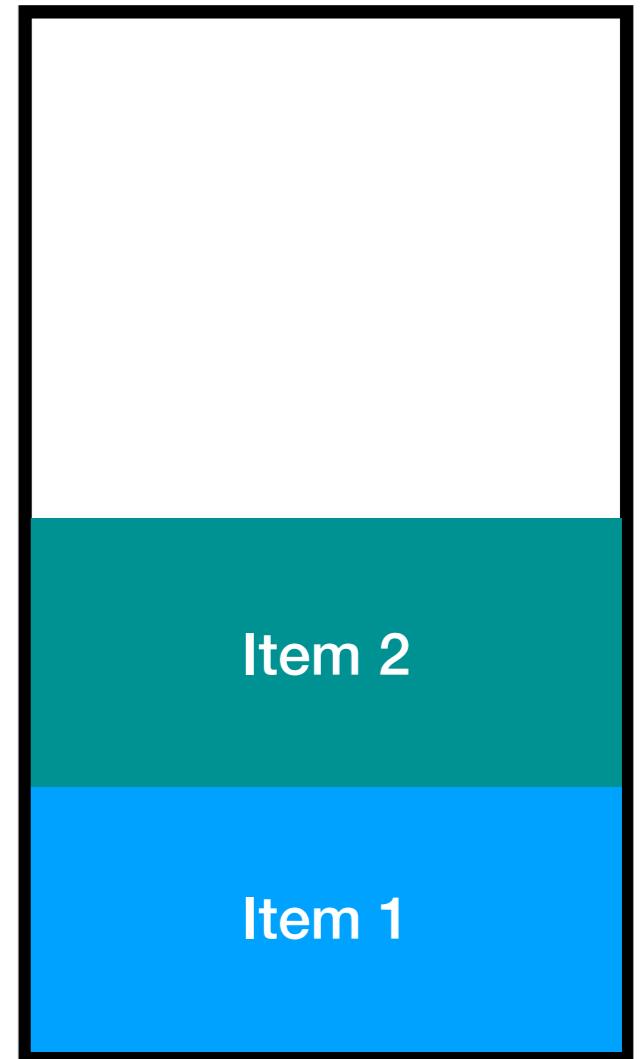
- Like a list but with only three operations:
 - Push — adds an item to the top of the stack
 - Pop — removes the item on the top of the stack

Push (Item 1)

Push (Item 2)

Pop ()

the stack



The Stack Data Structure

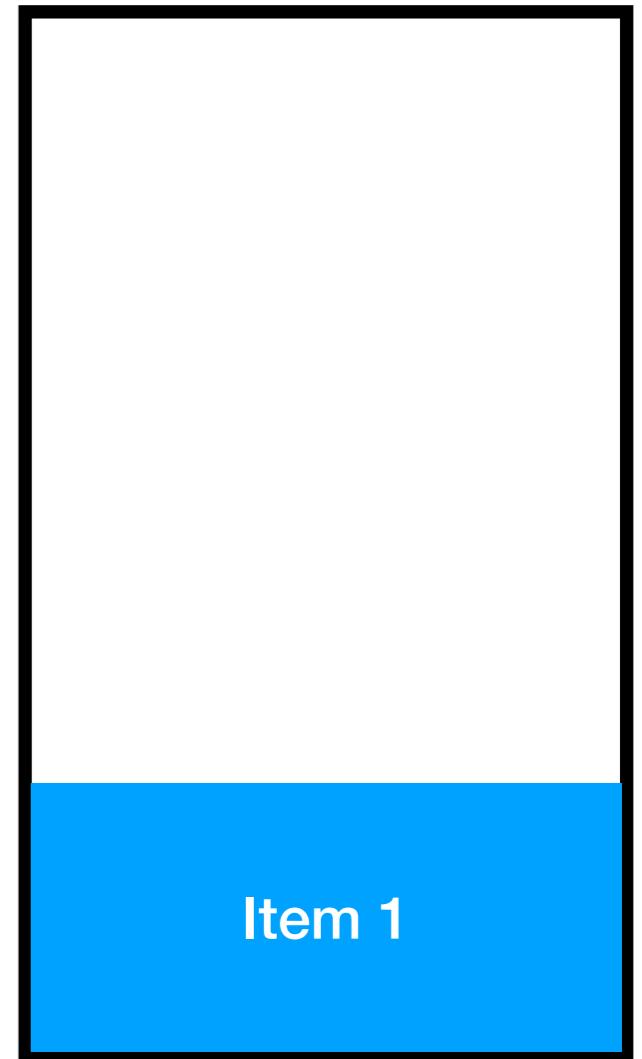
- Like a list but with only three operations:
 - Push — adds an item to the top of the stack
 - Pop — removes the item on the top of the stack

Push (Item 1)

Push (Item 2)

Pop ()

the stack



The Stack Data Structure

- Like a list but with only three operations:
 - Push — adds an item to the top of the stack
 - Pop — removes the item on the top of the stack

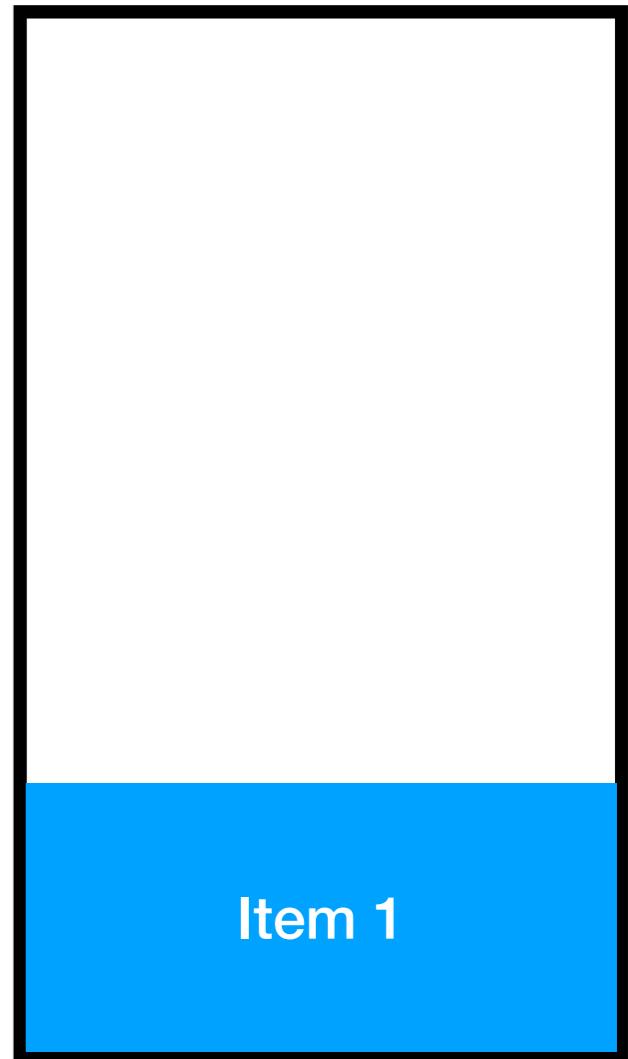
Push (Item 1)

Push (Item 2)

Pop ()

Push (Item 3)

the stack



The Stack Data Structure

- Like a list but with only three operations:
 - Push — adds an item to the top of the stack
 - Pop — removes the item on the top of the stack

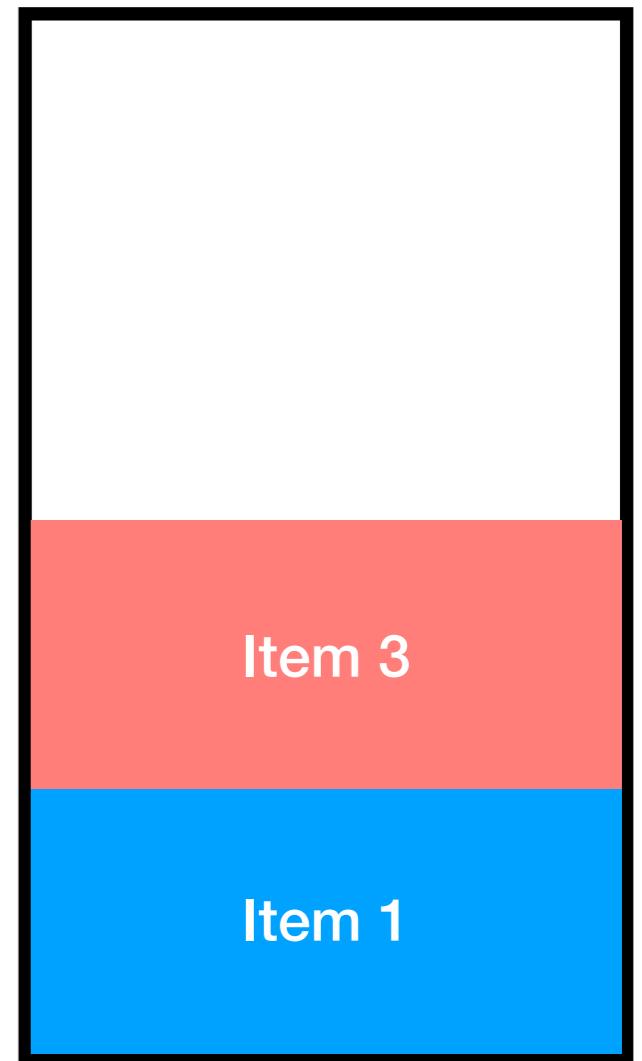
Push (Item 1)

Push (Item 2)

Pop ()

Push (Item 3)

the stack



The Stack Data Structure

- Like a list but with only three operations:
 - Push — adds an item to the top of the stack
 - Pop — removes the item on the top of the stack
 - Peek — looks at what's on top of the stack

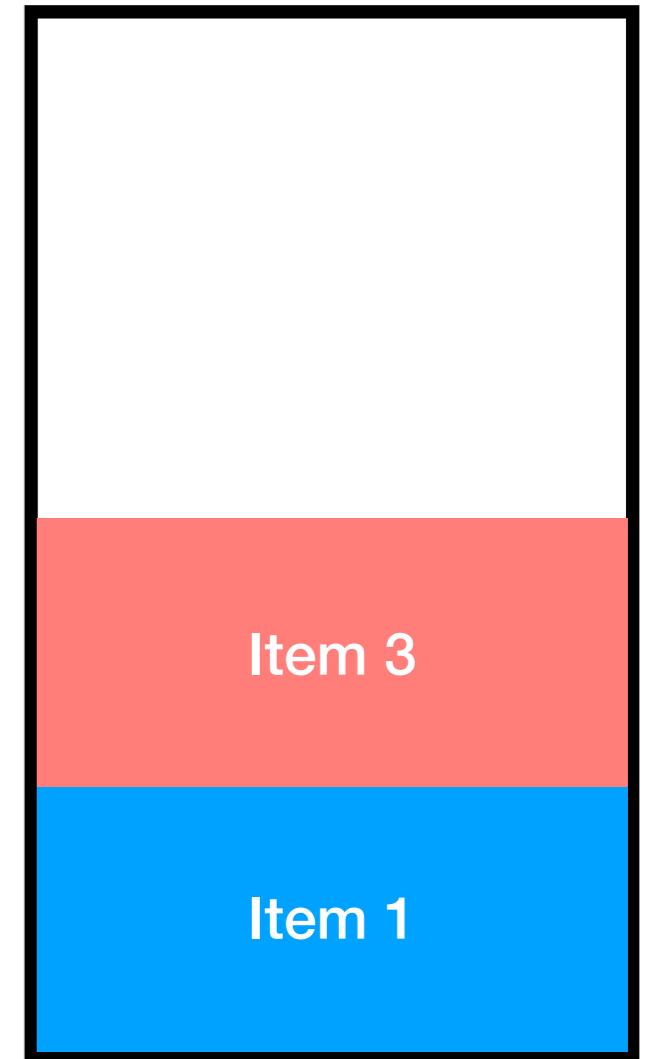
Push (Item 1)

Push (Item 2)

Pop ()

Push (Item 3)

the stack



The Stack Data Structure

- Like a list but with only three operations:
 - Push — adds an item to the top of the stack
 - Pop — removes the item on the top of the stack
 - Peek — looks at what's on top of the stack

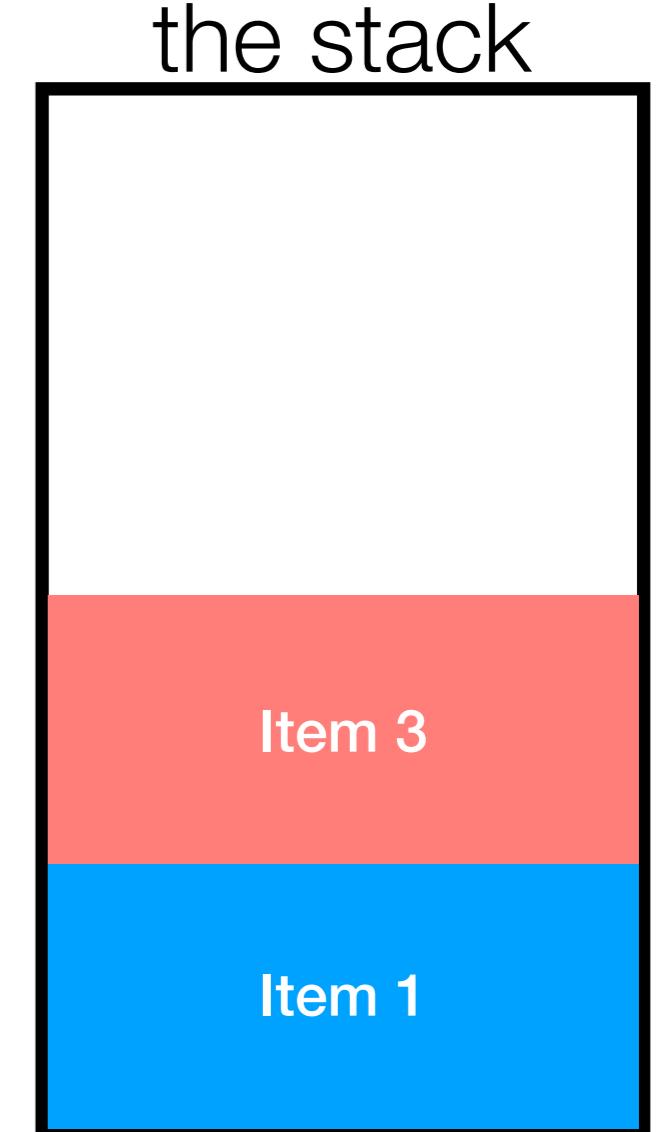
Push (Item 1)

Push (Item 2)

Pop ()

Push (Item 3)

Peek ()



Shift-reduce parsing operations

Shift-reduce parsing operations

- Shift — push something onto the stack

Shift-reduce parsing operations

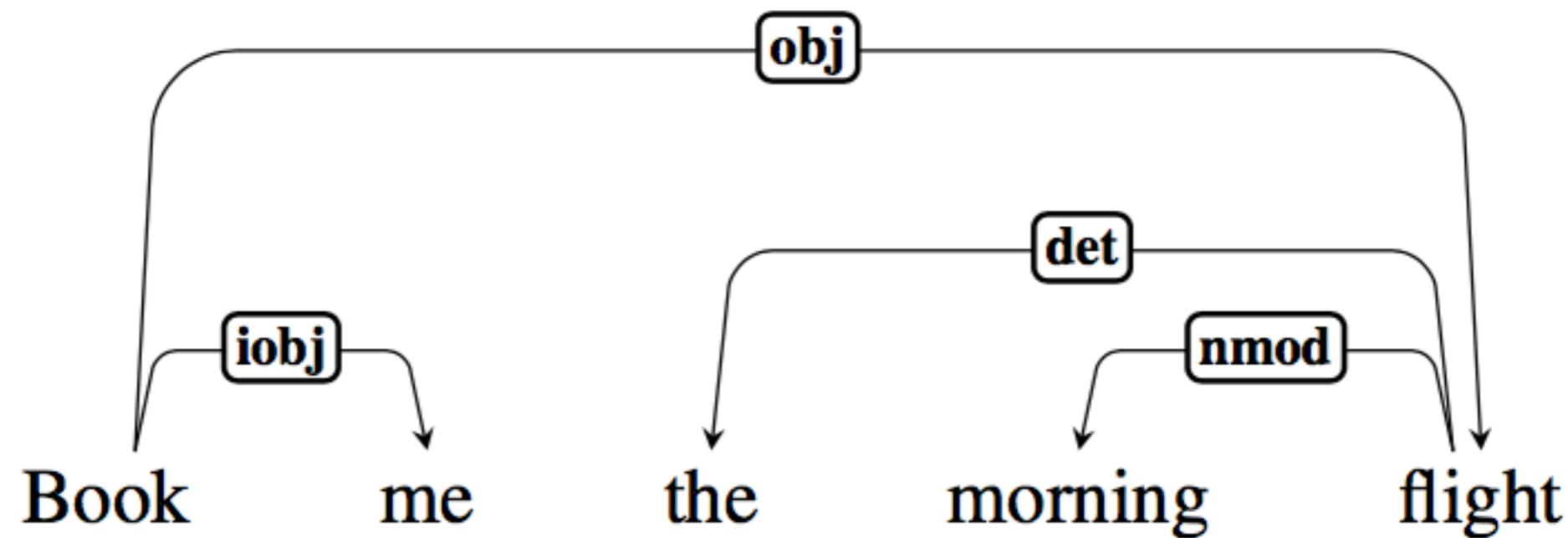
- Shift — push something onto the stack
- Reduce — pop two items off the top and replace them with something new

Shift-reduce parsing operations

- Shift — push something onto the stack
- Reduce — pop two items off the top and replace them with something new
 - Sometimes we have special “reduce” operations to decide what to put back on the top

Configuration

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



∅ book me the morning flight

stack

action

arc

Buffer of words

∅ book me the morning flight

stack

action

arc

LeftArc(label): assert relation between head at stack₁ and dependent at stack₂; remove stack₂

RightArc(label): assert relation between head at stack₂ and dependent at stack₁; remove stack₁

Shift: Remove word from front of input buffer (∅) and push it onto stack

Buffer of words

∅ book me the morning flight

stack

action

arc

LeftArc(label): assert relation between head at stack₁ and dependent at stack₂; remove stack₂

RightArc(label): assert relation between head at stack₂ and dependent at stack₁; remove stack₁

- 👉 Shift: Remove word from front of input buffer (∅) and push it onto stack

Buffer of words

book me the morning flight

stack

action

arc

LeftArc(label): assert relation between head at stack₁ and dependent at stack₂; remove stack₂

RightArc(label): assert relation between head at stack₂ and dependent at stack₁; remove stack₁

∅



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 stack₁ (\emptyset)
and dependent at stack₂:
remove stack₂

RightArc(label): assert
relation between head at
stack₂ and dependent at
stack₁ (\emptyset); remove stack₁ (\emptyset)

\emptyset

Shift: Remove word from
front of input buffer (book)
and push it onto stack

book me the morning flight

stack

action

arc

LeftArc(label): assert relation
between head at stack₁ (\emptyset)
and dependent at stack₂:
remove stack₂

RightArc(label): assert
relation between head at
stack₂ and dependent at
stack₁ (\emptyset); remove stack₁ (\emptyset)

\emptyset



Shift: Remove word from
front of input buffer (book)
and push it onto stack

me the morning flight

stack

action

arc

book

\emptyset

LeftArc(label): assert relation
between head at stack₁ (\emptyset)
and dependent at stack₂:
remove stack₂

RightArc(label): assert
relation between head at
stack₂ and dependent at
stack₁ (\emptyset); remove stack₁ (\emptyset)



Shift: Remove word from
front of input buffer (book)
and push it onto stack

me the morning flight

stack

action

arc

book

\emptyset

LeftArc(label): assert relation
between head at stack₁
(book) and dependent at
stack₂ (\emptyset); remove stack₂ (\emptyset)

RightArc(label): assert
relation between head at
stack₂ (\emptyset) and dependent at
stack₁ (book); remove stack₁
(book)

Shift: Remove word from
front of input buffer (me) 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 stack₁ (**book**) and dependent at stack₂ (\emptyset); remove stack₂ (\emptyset)

RightArc(label): assert relation between head at stack₂ (\emptyset) and dependent at stack₁ (**book**); remove stack₁ (**book**)

book

\emptyset

Shift: Remove word from front of input buffer (**me**) 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 stack₁ (**book**) and dependent at stack₂ (\emptyset); remove stack₂ (\emptyset)

RightArc(label): assert relation between head at stack₂ (\emptyset) and dependent at stack₁ (**book**); remove stack₁ (**book**)

book

\emptyset



Shift: Remove word from front of input buffer (**me**) and push it onto stack

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

the morning flight

stack

action

arc

me

book

\emptyset

LeftArc(label): assert relation between head at stack₁ (**book**) and dependent at stack₂ (\emptyset); remove stack₂ (\emptyset)

RightArc(label): assert relation between head at stack₂ (\emptyset) and dependent at stack₁ (**book**); remove stack₁ (**book**)



Shift: Remove word from front of input buffer (**me**) and push it onto stack

the morning flight

stack

action

arc

me

book

∅

LeftArc(label): assert relation
between head at stack₁ (**me**)
and dependent at stack₂
(**book**); remove stack₂ (**book**)

RightArc(label): assert
relation between head at
stack₂ (**book**) and
dependent at stack₁ (**me**);
remove stack₁ (**me**)

Shift: Remove word from
front of input buffer (**the**) and
push it onto stack

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	

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)	<i>iobj(book, me)</i>
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
book	LeftArc(label): assert relation between head at stack ₁ (me) and dependent at stack ₂ (book); remove stack ₂ (book)	<i>iobj(book, me)</i>
∅	RightArc(label): assert relation between head at stack ₂ (book) and dependent at stack ₁ (me); remove stack ₁ (me)	
	Shift: Remove word from front of input buffer (the) and push it onto stack	

the morning flight

stack	action	arc
book	LeftArc(label): assert relation between head at stack ₁ (me) and dependent at stack ₂ (book); remove stack ₂ (book)	<i>iobj(book, me)</i>
∅	RightArc(label): assert relation between head at stack ₂ (book) and dependent at stack ₁ (me); remove stack ₁ (me)	
	Shift: Remove word from front of input buffer (the) and push it onto stack	

morning flight

stack	action	arc
the	LeftArc(label): assert relation between head at stack ₁ (me) and dependent at stack ₂ (book); remove stack ₂ (book)	<i>iobj(book, me)</i>
book	RightArc(label): assert relation between head at stack ₂ (book) and dependent at stack ₁ (me); remove stack ₁ (me)	
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

morning flight

stack	action	arc
the	LeftArc(label): assert relation between head at stack ₁ (the) and dependent at stack ₂ (book); remove stack ₂ (book)	<i>iobj(book, me)</i>
book	RightArc(label): assert relation between head at stack ₂ (book) and dependent at stack ₁ (the); remove stack ₁ (the)	
∅	Shift: Remove word from front of input buffer (morning) and push it onto stack	

morning flight

stack	action	arc
the	LeftArc(label): assert relation between head at stack ₁ (the) and dependent at stack ₂ (book); remove stack ₂ (book)	<i>iobj(book, me)</i>
book	RightArc(label): assert relation between head at stack ₂ (book) and dependent at stack ₁ (the); remove stack ₁ (the)	
∅	Shift: Remove word from front of input buffer (morning) and push it onto stack	

flight

stack	action	arc
morning	LeftArc(label): assert relation between head at stack ₁ (the) and dependent at stack ₂ (book); remove stack ₂ (book)	<i>iobj(book, me)</i>
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

morning

the

book

∅

LeftArc(label): assert relation between head at stack₁ (*morning*) and dependent at stack₂ (*the*); remove stack₂ (*the*)

RightArc(label): assert relation between head at stack₂ (*the*) and dependent at stack₁ (*morning*); remove stack₁ (*morning*)

Shift: Remove word from front of input buffer (*flight*) and push it onto stack

iobj(book, me)

flight

stack

action

arc

morning

the

book

∅

LeftArc(label): assert relation between head at stack₁ (*morning*) and dependent at stack₂ (*the*); remove stack₂ (*the*)

RightArc(label): assert relation between head at stack₂ (*the*) and dependent at stack₁ (*morning*); remove stack₁ (*morning*)



Shift: Remove word from front of input buffer (*flight*) and push it onto stack

iobj(book, me)

stack	action	arc
flight	LeftArc(label): assert relation between head at stack ₁ (<i>morning</i>) and dependent at stack ₂ (<i>the</i>); remove stack ₂ (<i>the</i>)	<i>iobj(book, me)</i>
morning		
the	RightArc(label): assert relation between head at stack ₂ (<i>the</i>) and dependent at stack ₁ (<i>morning</i>); remove stack ₁ (<i>morning</i>)	
book		
∅	Shift: Remove word from front of input buffer (<i>flight</i>) 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)	<i>iobj(book, me)</i>
morning		<i>nmod(flight, morning)</i>
the	RightArc(label): assert relation between head at stack ₂ (morning) and dependent at stack ₁ (flight); remove stack ₁ (flight)	
book		
∅	Shift: Remove word from front of input buffer 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)	<i>iobj(book, me)</i>
morning		<i>nmod(flight, morning)</i>
the	RightArc(label): assert relation between head at stack ₂ (morning) and dependent at stack ₁ (flight); remove stack ₁ (flight)	
book		
∅	Shift: Remove word from front of input buffer 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)	<i>iobj(book, me)</i> <i>nmod(flight, morning)</i>
the book	RightArc(label): assert relation between head at stack ₂ (morning) and dependent at stack ₁ (flight); remove stack ₁ (flight)	
∅	Shift: Remove word from front of input buffer and push it onto stack	

stack	action	arc
flight	LeftArc(label): assert relation between head at stack ₁ (flight) and dependent at stack ₂ (the); remove stack ₂ (the)	<i>iobj(book, me)</i> <i>nmod(flight, morning)</i>
the	RightArc(label): assert relation between head at stack ₂ (the) and dependent at stack ₁ (flight); remove stack ₁ (flight)	
book		
∅	Shift: Remove word from front of input buffer and push it onto stack	

stack	action	arc
flight	 LeftArc(label): assert relation between head at stack ₁ (flight) and dependent at stack ₂ (the); remove stack ₂ (the)	<i>iobj(book, me)</i> <i>nmod(flight, morning)</i>
the	RightArc(label): assert relation between head at stack ₂ (the) and dependent at stack ₁ (flight); remove stack ₁ (flight)	
book		
∅	Shift: Remove word from front of input buffer and push it onto stack	

stack	action	arc
flight	 LeftArc(label): assert relation between head at stack ₁ (flight) and dependent at stack ₂ (the); remove stack ₂ (the)	<i>iobj(book, me)</i> <i>nmod(flight, morning)</i> <i>det(flight, the)</i>
the	RightArc(label): assert relation between head at stack ₂ (the) and dependent at stack ₁ (flight); remove stack ₁ (flight)	
book		
∅	Shift: Remove word from front of input buffer and push it onto stack	

stack	action	arc
flight	 LeftArc(label): assert relation between head at stack ₁ (flight) and dependent at stack ₂ (the); remove stack ₂ (the)	<i>iobj(book, me)</i> <i>nmod(flight, morning)</i> <i>det(flight, the)</i>
book	RightArc(label): assert relation between head at stack ₂ (the) and dependent at stack ₁ (flight); remove stack ₁ (flight)	
∅	Shift: Remove word from front of input buffer and push it onto stack	

stack	action	arc
flight	LeftArc(label): assert relation between head at stack ₁ (flight) and dependent at stack ₂ (book); remove stack ₂ (book)	<i>iobj(book, me)</i> <i>nmod(flight, morning)</i>
book	RightArc(label): assert relation between head at stack ₂ (book) and dependent at stack ₁ (flight); remove stack ₁ (flight)	<i>det(flight, the)</i>
∅	Shift: Remove word from front of input buffer and push it onto stack	

stack	action	arc
flight	LeftArc(label): assert relation between head at stack ₁ (flight) and dependent at stack ₂ (book); remove stack ₂ (book)	<i>iobj(book, me)</i> <i>nmod(flight, morning)</i>
book	RightArc(label): assert relation between head at stack ₂ (book) and dependent at stack ₁ (flight); remove stack ₁ (flight)	<i>det(flight, the)</i>
∅	Shift: Remove word from front of input buffer and push it onto stack	

stack	action	arc
flight	LeftArc(label): assert relation between head at stack ₁ (flight) and dependent at stack ₂ (book); remove stack ₂ (book)	<i>iobj(book, me)</i> <i>nmod(flight, morning)</i>
book	RightArc(label): assert relation between head at stack ₂ (book) and dependent at stack ₁ (flight); remove stack ₁ (flight)	<i>det(flight, the)</i> <i>obj(book, flight)</i>
∅	Shift: Remove word from front of input buffer and push it onto stack	

stack	action	arc
	LeftArc(label): assert relation between head at stack ₁ (<i>flight</i>) and dependent at stack ₂ (<i>book</i>); remove stack ₂ (<i>book</i>)	<i>iobj(book, me)</i>
		<i>nmod(flight, morning)</i>
		<i>det(flight, the)</i>
book	RightArc(label): assert relation between head at stack ₂ (<i>book</i>) and dependent at stack ₁ (<i>flight</i>); remove stack ₁ (<i>flight</i>)	<i>obj(book, flight)</i>
∅	Shift: Remove word from front of input buffer and push it onto stack	

stack	action	arc
book	LeftArc(label): assert relation between head at stack ₁ (book) and dependent at stack ₂ (\emptyset); remove stack ₂ (\emptyset)	<i>iobj(book, me)</i>
\emptyset	RightArc(label): assert relation between head at stack ₂ (\emptyset) and dependent at stack ₁ (book); remove stack ₁ (book)	<i>nmod(flight, morning)</i>
	Shift: Remove word from front of input buffer and push it onto stack	<i>det(flight, the)</i>
		<i>obj(book, flight)</i>

stack	action	arc
book	LeftArc(label): assert relation between head at stack ₁ (book) and dependent at stack ₂ (\emptyset); remove stack ₂ (\emptyset)	<i>iobj(book, me)</i>
\emptyset	RightArc(label): assert relation between head at stack ₂ (\emptyset) and dependent at stack ₁ (book); remove stack ₁ (book)	<i>nmod(flight, morning)</i>
	Shift: Remove word from front of input buffer and push it onto stack	<i>det(flight, the)</i>
		<i>obj(book, flight)</i>

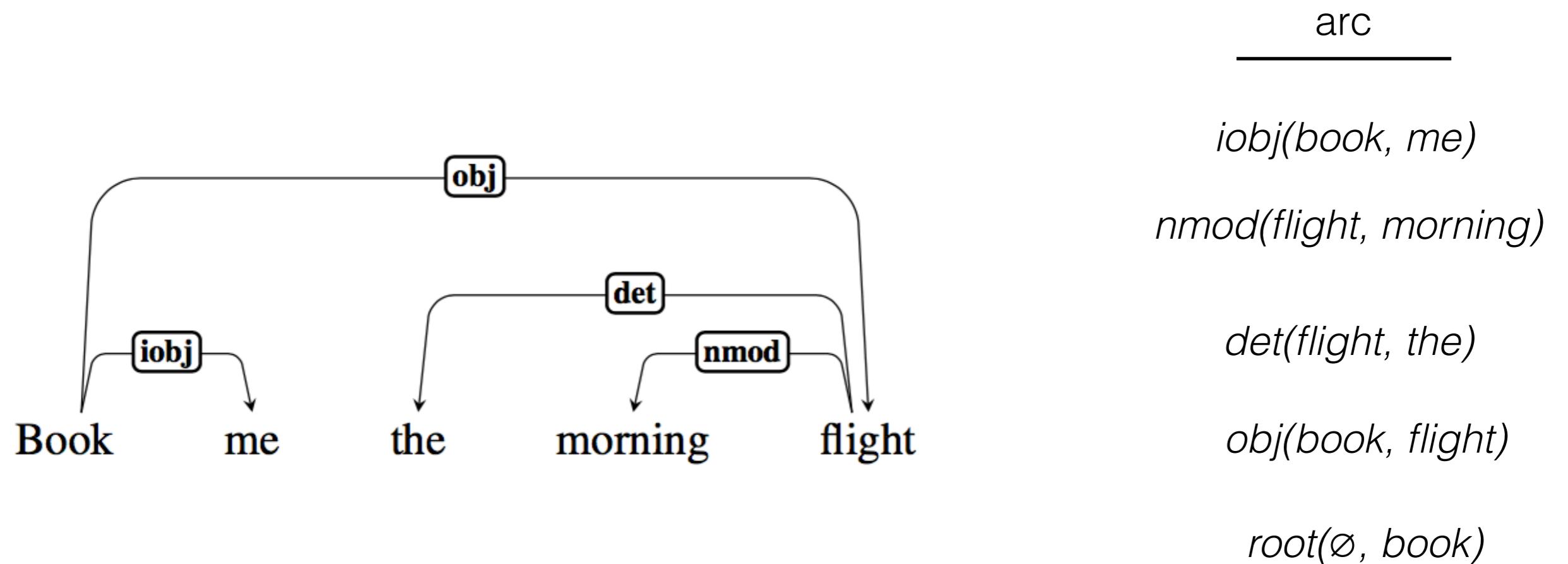
stack	action	arc
	LeftArc(label): assert relation between head at stack ₁ (book) and dependent at stack ₂ (\emptyset); remove stack ₂ (\emptyset)	<i>iobj(book, me)</i>
		<i>nmod(flight, morning)</i>
	RightArc(label): assert relation between head at stack ₂ (\emptyset) and dependent at stack ₁ (book); remove stack ₁ (book)	<i>det(flight, the)</i>
book		<i>obj(book, flight)</i>
\emptyset	Shift: Remove word from front of input buffer and push it onto stack	<i>root(\emptyset, book)</i>

stack	action	arc
	LeftArc(label): assert relation between head at stack ₁ (book) and dependent at stack ₂ (Ø); remove stack ₂ (Ø)	<i>iobj(book, me)</i>
		<i>nmod(flight, morning)</i>
👉	RightArc(label): assert relation between head at stack ₂ (Ø) and dependent at stack ₁ (book); remove stack ₁ (book)	<i>det(flight, the)</i>
		<i>obj(book, flight)</i>
Ø	Shift: Remove word from front of input buffer and push it onto stack	<i>root(Ø, book)</i>

This is our parse

stack	action	arc
	LeftArc(label): assert relation between head at stack ₁ (<i>book</i>) and dependent at stack ₂ (\emptyset); remove stack ₂ (\emptyset)	<i>iobj(book, me)</i>
		<i>nmod(flight, morning)</i>
		<i>det(flight, the)</i>
👉	RightArc(label): assert relation between head at stack ₂ (\emptyset) and dependent at stack ₁ (<i>book</i>); remove stack ₁ (<i>book</i>)	<i>obj(book, flight)</i>
		<i>root(\emptyset, book)</i>
∅	Shift: Remove word from front of input buffer and push it onto stack	

This is our parse



Let's go back to this earlier configuration

the morning flight

stack	action	arc
me	LeftArc(label): assert relation between head at stack ₁ (me) and dependent at stack ₂ (book); remove stack ₂ (book)	
book	RightArc(label): assert relation between head at stack ₂ (book) and dependent at stack ₁ (me); remove stack ₁ (me)	
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

Let's go back to this earlier configuration

the morning flight

stack

action

arc

me

LeftArc(label): assert relation between head at stack₁ (**me**) and dependent at stack₂ (**book**); remove stack₂ (**book**)

book

RightArc(label): assert relation between head at stack₂ (**book**) and dependent at stack₁ (**me**); remove stack₁ (**me**)

∅

Shift: Remove word from front of input buffer (**the**) and push it onto stack

How should we determine what operation to do next?

- This is a multi class 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?

Output space $\mathcal{Y} =$



stack

me

book

buffer

the morning flight

arc

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

stack

me

book

buffer

the morning flight

arc

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

feature

example

stack

me

book

buffer

the morning flight

arc

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

feature

example

stack₁ = me

1

stack

me

book

buffer

the morning flight

arc

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

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

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

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

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

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

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

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

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
- Naive Bayes
- SVM
- Neural network

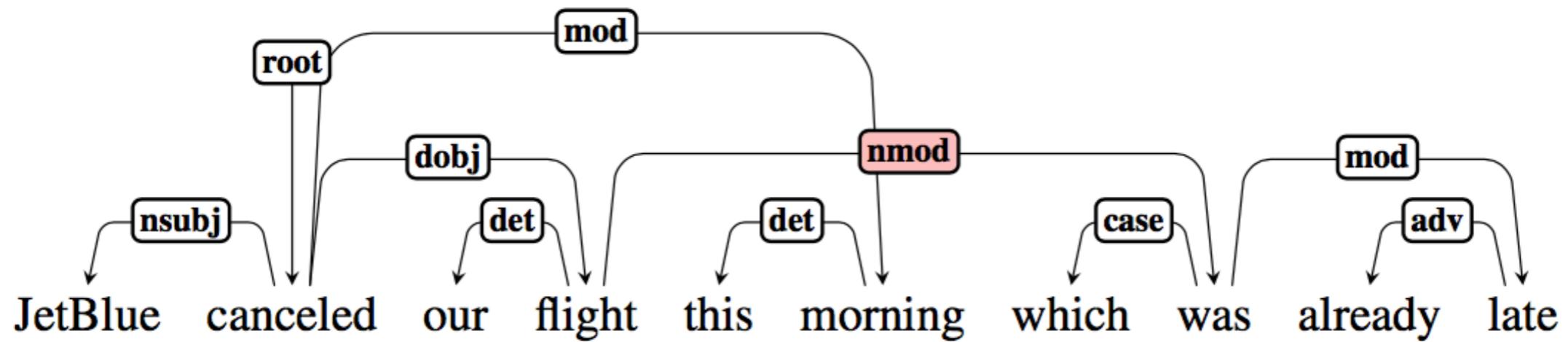
feature	example	β
stack ₁ = me	1	0.7
stack ₂ = book	1	1.3
stack ₁ POS = PRP	1	6.4
buffer ₁ = the	1	-1.3
buffer ₂ = morning	1	-0.07
buffer ₁ = today	0	0.52
buffer ₁ POS = RB	0	-2.1
stack ₁ = me AND stack ₂ =	1	0
stack ₁ = PRP AND stack ₂ =	1	-0.1
iobj(book,*) in arcs	0	3.2

Training

We're training to predict the parser action
(Shift, RightArc, LeftArc) 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)

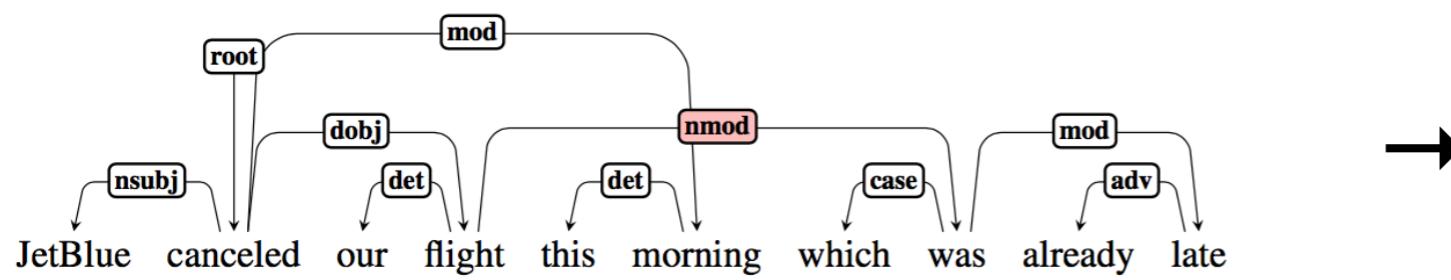
Training data



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

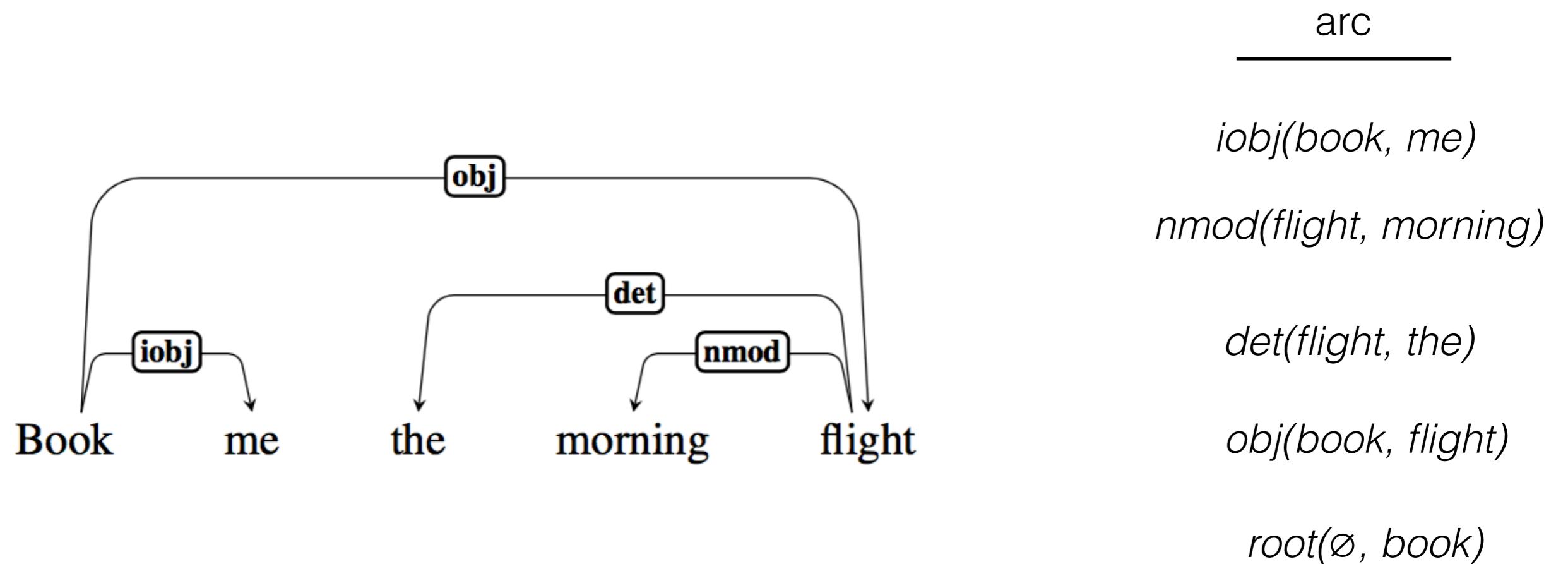
Oracle

- 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	Label
<stack1 = me, 1>,	Shift
<stack1 = me, 0>,	RightArc(det)
<stack1 = me, 0>,	RightArc(nsu

This is our parse



\emptyset book me the morning flight

stack

action

gold tree

iobj(book, me)

nmod(flight, morning)

det(flight, the)

obj(book, flight)

root(\emptyset , book)

\emptyset book me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	$iobj(book, me)$
	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$nmod(flight, morning)$
	Else shift: Remove word from front of input buffer and push it onto stack	$det(flight, the)$
		$obj(book, flight)$
		$root(\emptyset, book)$

book me the morning flight

stack	action	gold tree
\emptyset	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	$iobj(book, me)$
	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$nmod(flight, morning)$
	Else shift: Remove word from front of input buffer and push it onto stack	$det(flight, the)$
		$obj(book, flight)$
		$root(\emptyset, book)$

book me the morning flight

stack	action	gold tree
\emptyset	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	$iobj(book, me)$
	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$nmod(flight, morning)$
	Else shift: Remove word from front of input buffer and push it onto stack	$det(flight, the)$
		$obj(book, flight)$
		$root(\emptyset, book)$

me the morning flight

stack	action	gold tree
book	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	$iobj(book, me)$
\emptyset	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$nmod(flight, morning)$
	Else shift: Remove word from front of input buffer and push it onto stack	$det(flight, the)$
		$obj(book, flight)$
		$root(\emptyset, book)$

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

me the morning flight

stack

action

gold tree

Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack₂.

iobj(book, me)

Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack₁

det(flight, the)

obj(book, flight)

root(\emptyset , book)

book

\emptyset

Else shift: Remove word from front of input buffer and push it onto stack

me the morning flight

stack	action	gold tree
book	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	$iobj(book, me)$
\emptyset	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$nmod(flight, morning)$
	Else shift: Remove word from front of input buffer and push it onto stack	$det(flight, the)$
		$obj(book, flight)$
		$root(\emptyset, book)$

the morning flight

stack	action	gold tree
me	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<i>iobj(book, me)</i>
book	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	<i>nmod(flight, morning)</i>
\emptyset	Else shift: Remove word from front of input buffer and push it onto stack	<i>det(flight, the)</i> <i>obj(book, flight)</i> <i>root(\emptyset, book)</i>

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

the morning flight

stack

action

gold tree

me

book

\emptyset

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

$iobj(book, me)$

$nmod(flight, morning)$

$det(flight, the)$

$obj(book, flight)$

$root(\emptyset, book)$

the morning flight

stack	action	gold tree
me	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	$iobj(book, me)$ $nmod(flight, morning)$
book	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$det(flight, the)$ $obj(book, flight)$
\emptyset	Else shift: Remove word from front of input buffer and push it onto stack	$root(\emptyset, book)$

the morning flight

stack	action	gold tree
book	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	 $iobj(book, me)$ $nmod(flight, morning)$
\emptyset	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$det(flight, the)$ $obj(book, flight)$ $root(\emptyset, book)$
	Else shift: Remove word from front of input buffer and push it onto stack	

morning flight

stack	action	gold tree
the	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> $iobj(book, me)$ $nmod(flight, morning)$
book	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$det(flight, the)$ $obj(book, flight)$
\emptyset	Else shift: Remove word from front of input buffer and push it onto stack	$root(\emptyset, book)$

morning flight

stack	action	gold tree
the	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> $iobj(book, me)$
book	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$nmod(flight, morning)$
\emptyset	Else shift: Remove word from front of input buffer and push it onto stack	$det(flight, the)$ $obj(book, flight)$ $root(\emptyset, book)$

stack	action	gold tree
morning	Choose LeftArc(label) if label(stack ₁ , stack ₂) exists in gold tree. Remove stack ₂ .	✓ <i>iobj(book, me)</i> <i>nmod(flight, morning)</i>
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	<i>det(flight, the)</i> <i>obj(book, flight)</i>
book	Else shift: Remove word from front of input buffer and push it onto stack	<i>root(∅, book)</i>
∅		

stack	action	gold tree
morning	Choose LeftArc(label) if label(stack ₁ , stack ₂) exists in gold tree. Remove stack ₂ .	✓ <i>iobj(book, me)</i> <i>nmod(flight, morning)</i>
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	<i>det(flight, the)</i> <i>obj(book, flight)</i>
book	Else shift: Remove word from front of input buffer and push it onto stack	<i>root(∅, book)</i>
∅		

stack	action	gold tree
flight	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> $iobj(book, me)$
morning		$nmod(flight, morning)$
the	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$det(flight, the)$
book		$obj(book, flight)$
\emptyset	Else shift: Remove word from front of input buffer and push it onto stack	$root(\emptyset, book)$

stack	action	gold tree
flight	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> $iobj(book, me)$
morning		$nmod(flight, morning)$
the	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	$det(flight, the)$
book		$obj(book, flight)$
\emptyset	Else shift: Remove word from front of input buffer and push it onto stack	$root(\emptyset, book)$

nmod(flight, morning)

stack	action	gold tree
flight	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> <i>iobj(book, me)</i>
morning	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	<i>nmod(flight, morning)</i>
the		<i>det(flight, the)</i>
book		<i>obj(book, flight)</i>
∅	Else shift: Remove word from front of input buffer and push it onto stack	<i>root(∅, book)</i>

nmod(flight, morning)

stack	action	gold tree
flight	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> <i>iobj(book, me)</i>
morning	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	<input checked="" type="checkbox"/> <i>nmod(flight, morning)</i>
the		<i>det(flight, the)</i>
book		<i>obj(book, flight)</i>
∅	Else shift: Remove word from front of input buffer and push it onto stack	<i>root(∅, book)</i>

nmod(flight, morning)

stack

flight

the

book

\emptyset

action

Choose LeftArc(label) if
 $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in
gold tree. Remove stack₂.

Else choose RightArc(label)
if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists
in gold tree and all arcs
 $\text{label}(\text{stack}_1, *)$. have been
generated. Remove stack₁

Else shift: Remove word
from front of input buffer and
push it onto stack

gold tree

 *iobj(book, me)*

 *nmod(flight, morning)*

det(flight, the)

obj(book, flight)

root(\emptyset , book)

stack	action	gold tree
flight	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> $iobj(book, me)$
the	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	<input checked="" type="checkbox"/> $nmod(flight, morning)$
book	Else shift: Remove word from front of input buffer and push it onto stack	$det(flight, the)$
\emptyset		$obj(book, flight)$
		$root(\emptyset, book)$

det(flight,the)

stack	action	gold tree
flight	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> <i>iobj(book, me)</i>
the	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	<input checked="" type="checkbox"/> <i>nmod(flight, morning)</i>
book	Else shift: Remove word from front of input buffer and push it onto stack	<i>det(flight, the)</i> <i>obj(book, flight)</i> <i>root(∅, book)</i>
∅		

det(flight,the)

stack	action	gold tree
flight	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> <i>iobj(book, me)</i>
the	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	<input checked="" type="checkbox"/> <i>nmod(flight, morning)</i>
book	Else shift: Remove word from front of input buffer and push it onto stack	<input checked="" type="checkbox"/> <i>det(flight, the)</i> <i>obj(book, flight)</i> <i>root(∅, book)</i>
∅		

det(flight,the)

stack	action	gold tree
flight	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> <i>iobj(book, me)</i>
book	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	<input checked="" type="checkbox"/> <i>nmod(flight, morning)</i>
∅	Else shift: Remove word from front of input buffer and push it onto stack	<input checked="" type="checkbox"/> <i>det(flight, the)</i> <i>obj(book, flight)</i> <i>root(∅, book)</i>

stack	action	gold tree
flight	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ . Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁ Else shift: Remove word from front of input buffer and push it onto stack	<input checked="" type="checkbox"/> $iobj(book, me)$ <input checked="" type="checkbox"/> $nmod(flight, morning)$ <input checked="" type="checkbox"/> $det(flight, the)$ $obj(book, flight)$ $root(\emptyset, book)$
book		
\emptyset		

obj(book,flight)

stack

flight

book

\emptyset

action

Choose LeftArc(label) if
 $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in
gold tree. Remove stack₂.

Else choose RightArc(label)
if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists
in gold tree and all arcs
 $\text{label}(\text{stack}_1, *)$. have been
generated. Remove stack₁

Else shift: Remove word
from front of input buffer and
push it onto stack

gold tree

 *iobj(book, me)*

 *nmod(flight, morning)*

 *det(flight, the)*

obj(book, flight)

root(\emptyset , book)

obj(book,flight)

stack

flight

book

\emptyset

action

Choose LeftArc(label) if
 $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in
gold tree. Remove stack₂.

Else choose RightArc(label)
if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists
in gold tree and all arcs
 $\text{label}(\text{stack}_1, *)$. have been
generated. Remove stack₁

Else shift: Remove word
from front of input buffer and
push it onto stack

gold tree

 *iobj(book, me)*

 *nmod(flight, morning)*

 *det(flight, the)*

 *obj(book, flight)*

root(\emptyset , book)

obj(book,flight)

stack

action

gold tree

book

\emptyset

Choose LeftArc(label) if
 $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in
gold tree. Remove stack₂.

Else choose RightArc(label)
if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists
in gold tree and all arcs
 $\text{label}(\text{stack}_1, *)$. have been
generated. Remove stack₁

Else shift: Remove word
from front of input buffer and
push it onto stack

 $i\text{obj}(book, me)$

 $n\text{mod}(flight, morning)$

 $\text{det}(flight, the)$

 $\text{obj}(book, flight)$

$\text{root}(\emptyset, book)$

stack	action	gold tree
book	Choose LeftArc(label) if label(stack ₁ , stack ₂) exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> <i>iobj(book, me)</i>
∅	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	<input checked="" type="checkbox"/> <i>nmod(flight, morning)</i>
	Else shift: Remove word from front of input buffer and push it onto stack	<input checked="" type="checkbox"/> <i>det(flight, the)</i>
		<input checked="" type="checkbox"/> <i>obj(book, flight)</i>
		<i>root(∅, book)</i>

$\text{root}(\emptyset, \text{book})$ and book has no more dependents we haven't seen

stack	action	gold tree
book	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> $iobj(book, me)$
\emptyset	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	<input checked="" type="checkbox"/> $nmod(flight, morning)$
	Else shift: Remove word from front of input buffer and push it onto stack	<input checked="" type="checkbox"/> $det(flight, the)$
		<input checked="" type="checkbox"/> $obj(book, flight)$
		$root(\emptyset, book)$

$\text{root}(\emptyset, \text{book})$ and book has no more dependents we haven't seen

stack	action	gold tree
book	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> $iobj(book, me)$
\emptyset	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	<input checked="" type="checkbox"/> $nmod(flight, morning)$
	Else shift: Remove word from front of input buffer and push it onto stack	<input checked="" type="checkbox"/> $det(flight, the)$
		<input checked="" type="checkbox"/> $obj(book, flight)$
		<input checked="" type="checkbox"/> $root(\emptyset, book)$

stack	action	gold tree
\emptyset	Choose LeftArc(label) if $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in gold tree. Remove stack ₂ .	<input checked="" type="checkbox"/> $iobj(book, me)$
	Else choose RightArc(label) if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists in gold tree and all arcs $\text{label}(\text{stack}_1, *)$. have been generated. Remove stack ₁	<input checked="" type="checkbox"/> $nmod(flight, morning)$
	Else shift: Remove word from front of input buffer and push it onto stack	<input checked="" type="checkbox"/> $det(flight, the)$
		<input checked="" type="checkbox"/> $obj(book, flight)$
		<input checked="" type="checkbox"/> $root(\emptyset, book)$

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

stack

action

gold tree

Choose LeftArc(label) if
 $\text{label}(\text{stack}_1, \text{stack}_2)$ exists in
gold tree. Remove stack₂.

Else choose RightArc(label)
if $\text{label}(\text{stack}_2, \text{stack}_1)$ exists
in gold tree and all arcs
 $\text{label}(\text{stack}_1, *)$. have been
generated. Remove stack₁

Else shift: Remove word
from front of input buffer and
push it onto stack

\emptyset

- $iobj(book, me)$
- $nmod(flight, morning)$
- $det(flight, the)$
- $obj(book, flight)$
- $root(\emptyset, book)$



Neural Dependency Parsing

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

The feature vectors for these classifiers are usually very sparse and high-dimensional ($10^6 \sim 10^7$)

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

The feature vectors for these classifiers are usually very sparse and high-dimensional ($10^6 \sim 10^7$)

Use any multiclass classification model

- Logistic regression
- SVM
- NB
- Neural network

Word features

POS features

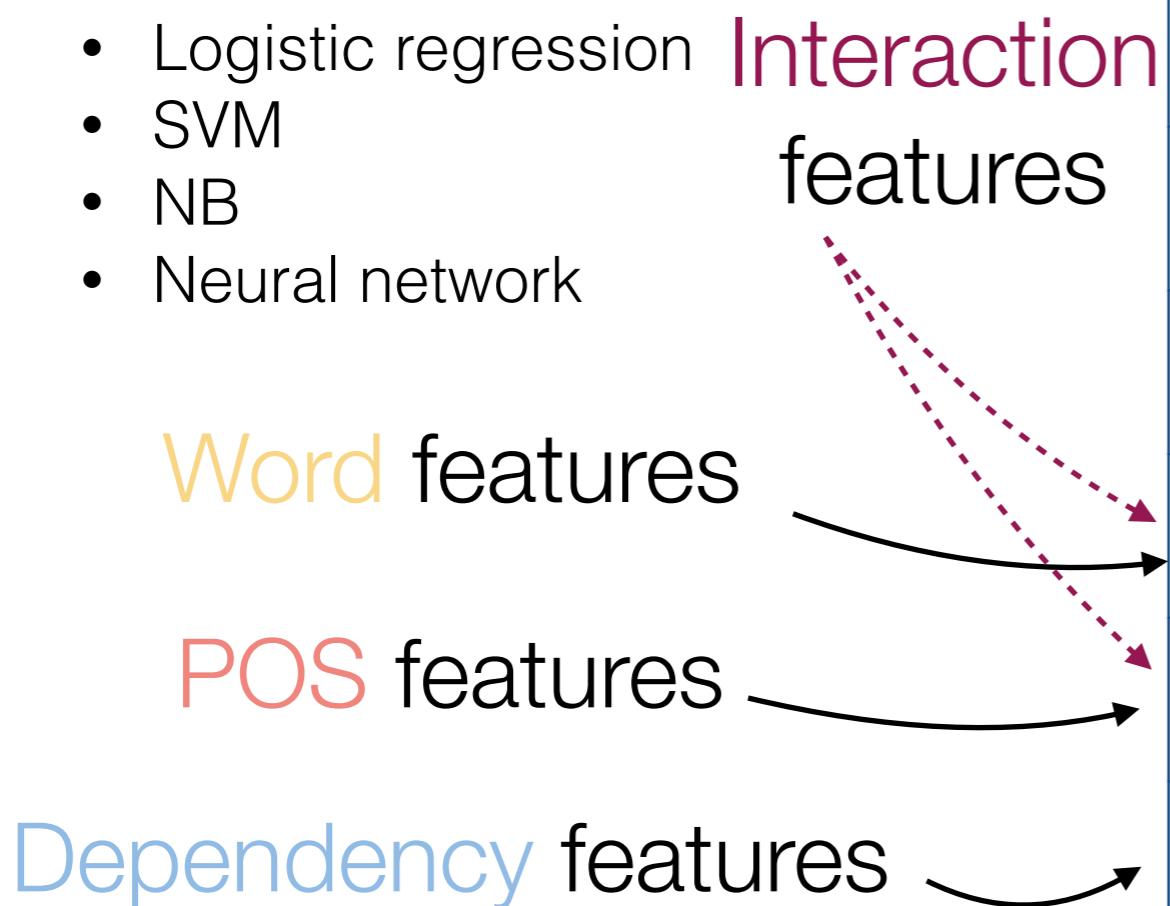
Dependency features

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

The feature vectors for these classifiers are usually very sparse and high-dimensional ($10^6 \sim 10^7$)

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

Problems with Indicator Features

Problems with Indicator Features

- Problem #1: sparse

Problems with Indicator Features

- Problem #1: sparse
- Problem #2: incomplete

Problems with Indicator Features

- Problem #1: sparse
- Problem #2: incomplete
- Problem #3: computationally expensive

Problems with Indicator Features

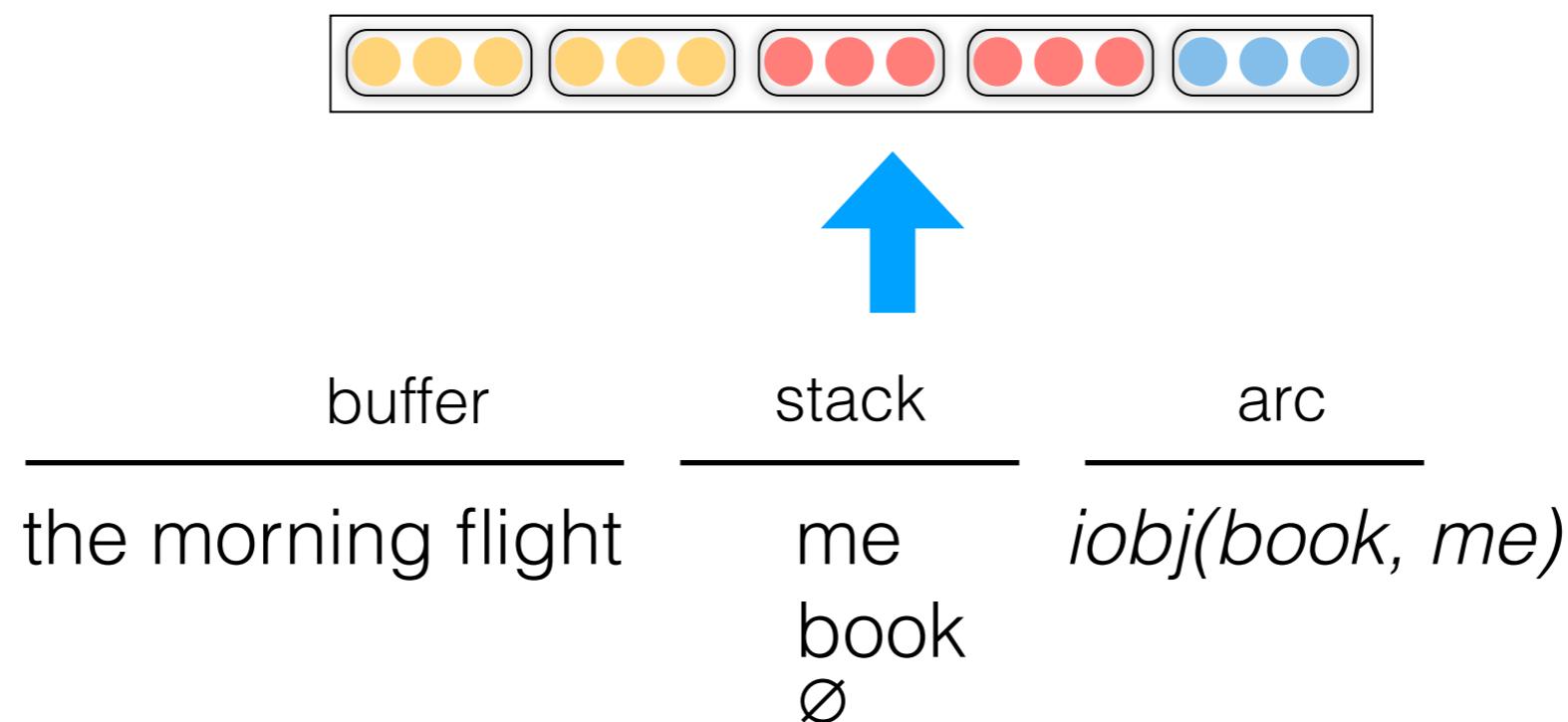
- Problem #1: sparse
- Problem #2: incomplete
- Problem #3: computationally expensive

More than 95% of parsing time is consumed by feature computation.

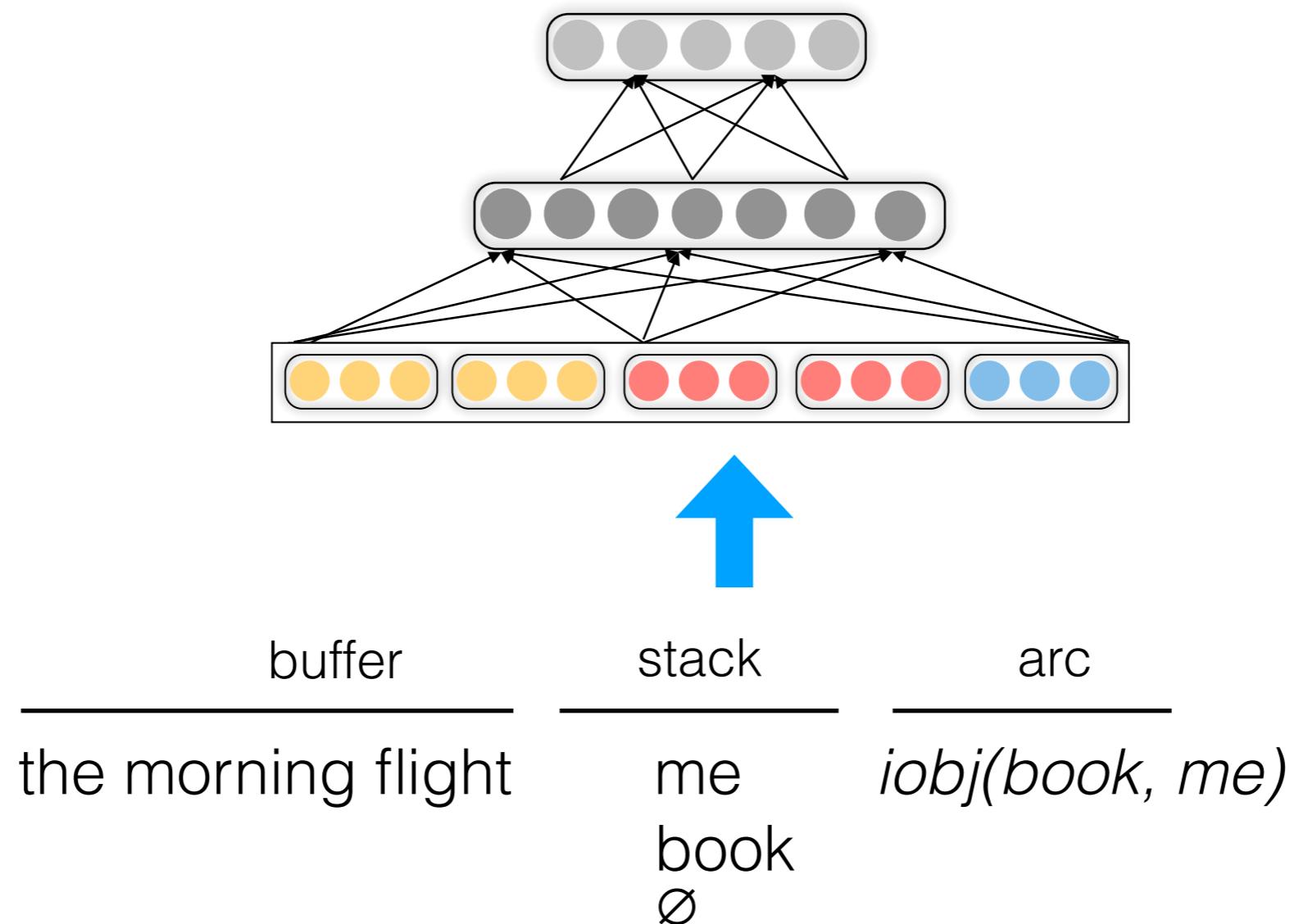
Potential Solution: Use dense neural representations!

buffer	stack	arc
the morning flight	me book \emptyset	$iobj(book, me)$

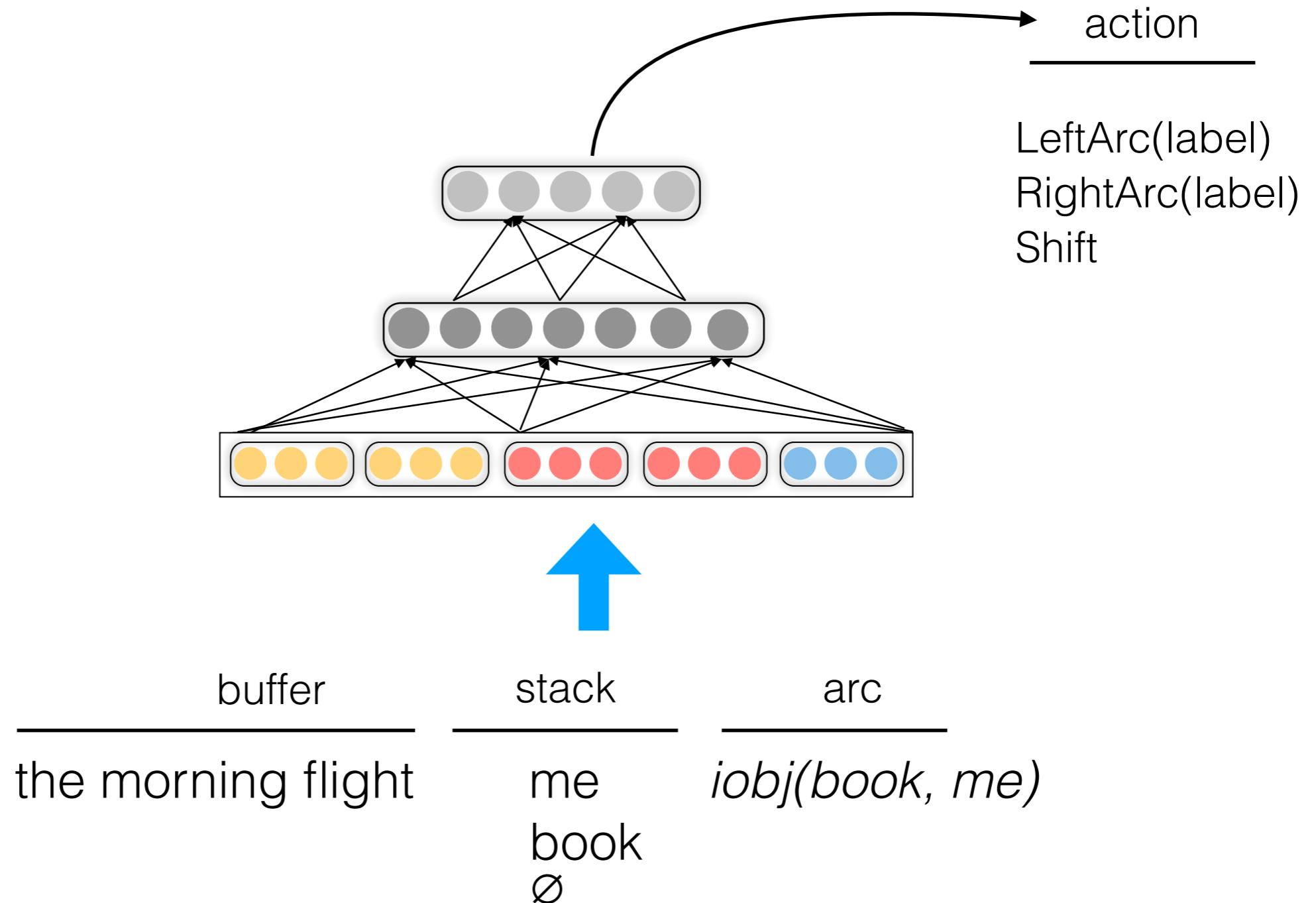
Potential Solution: Use dense neural representations!



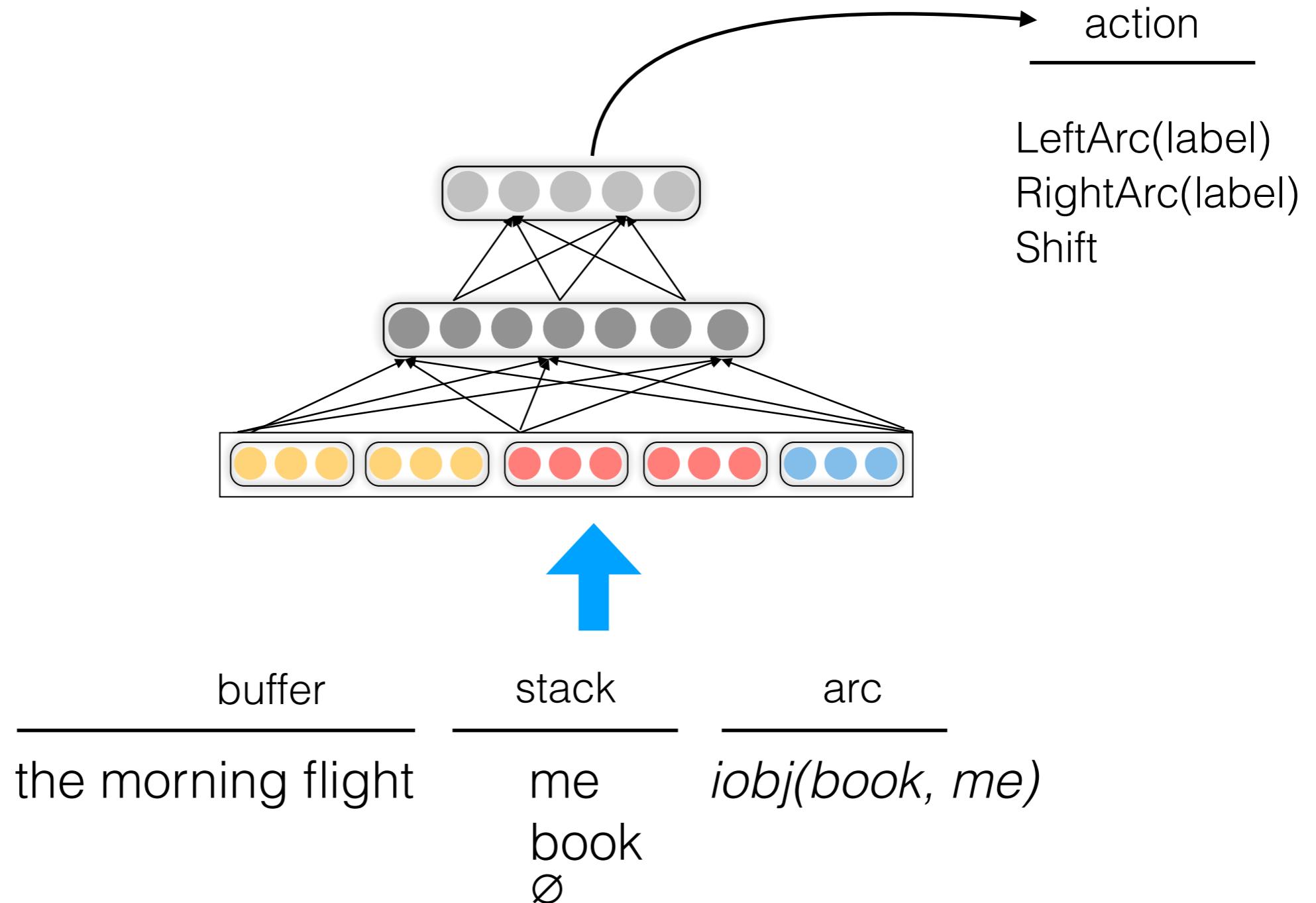
Potential Solution: Use dense neural representations!



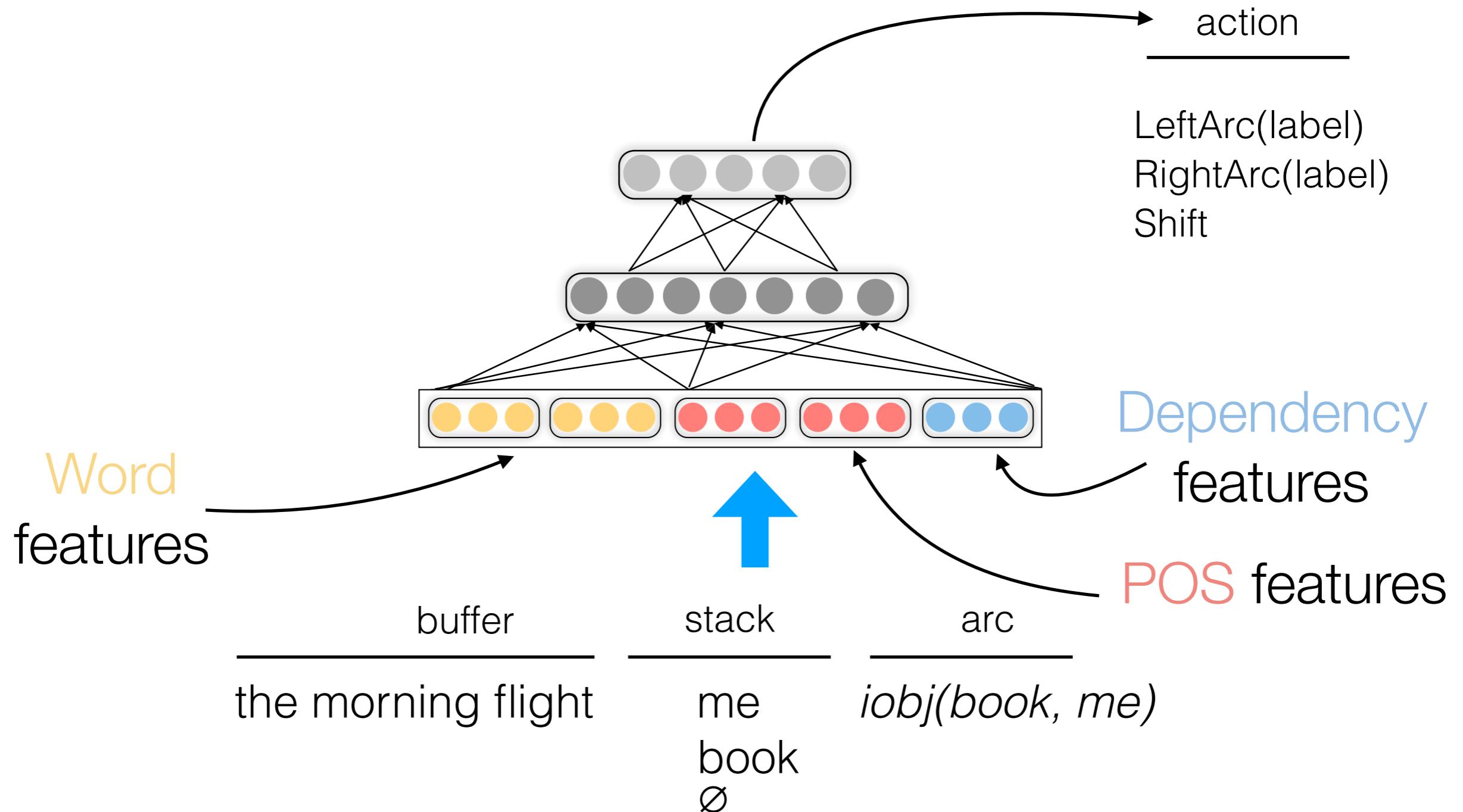
Potential Solution: Use dense neural representations!



Potential Solution: Use dense neural representations!

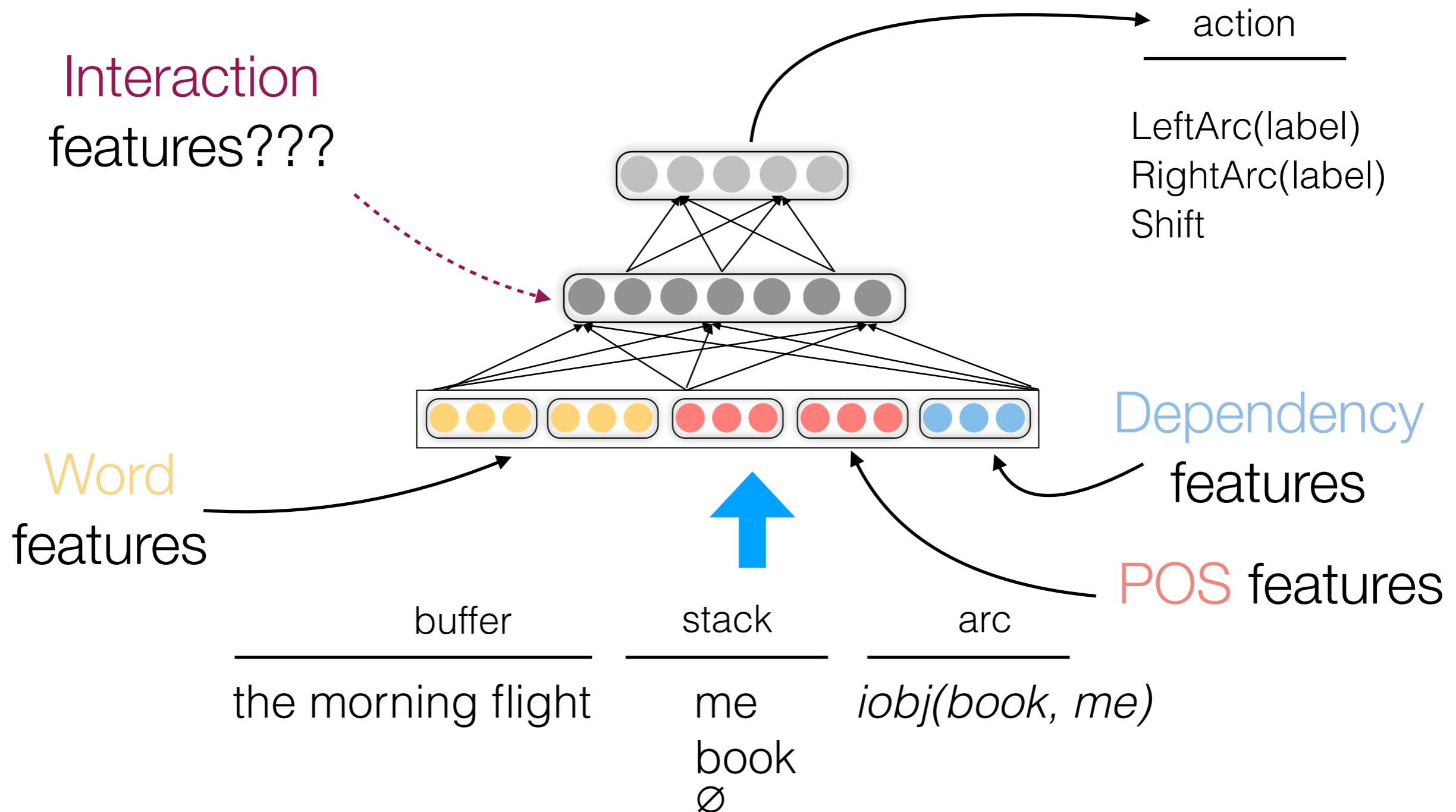


Potential Solution: Use dense neural representations!

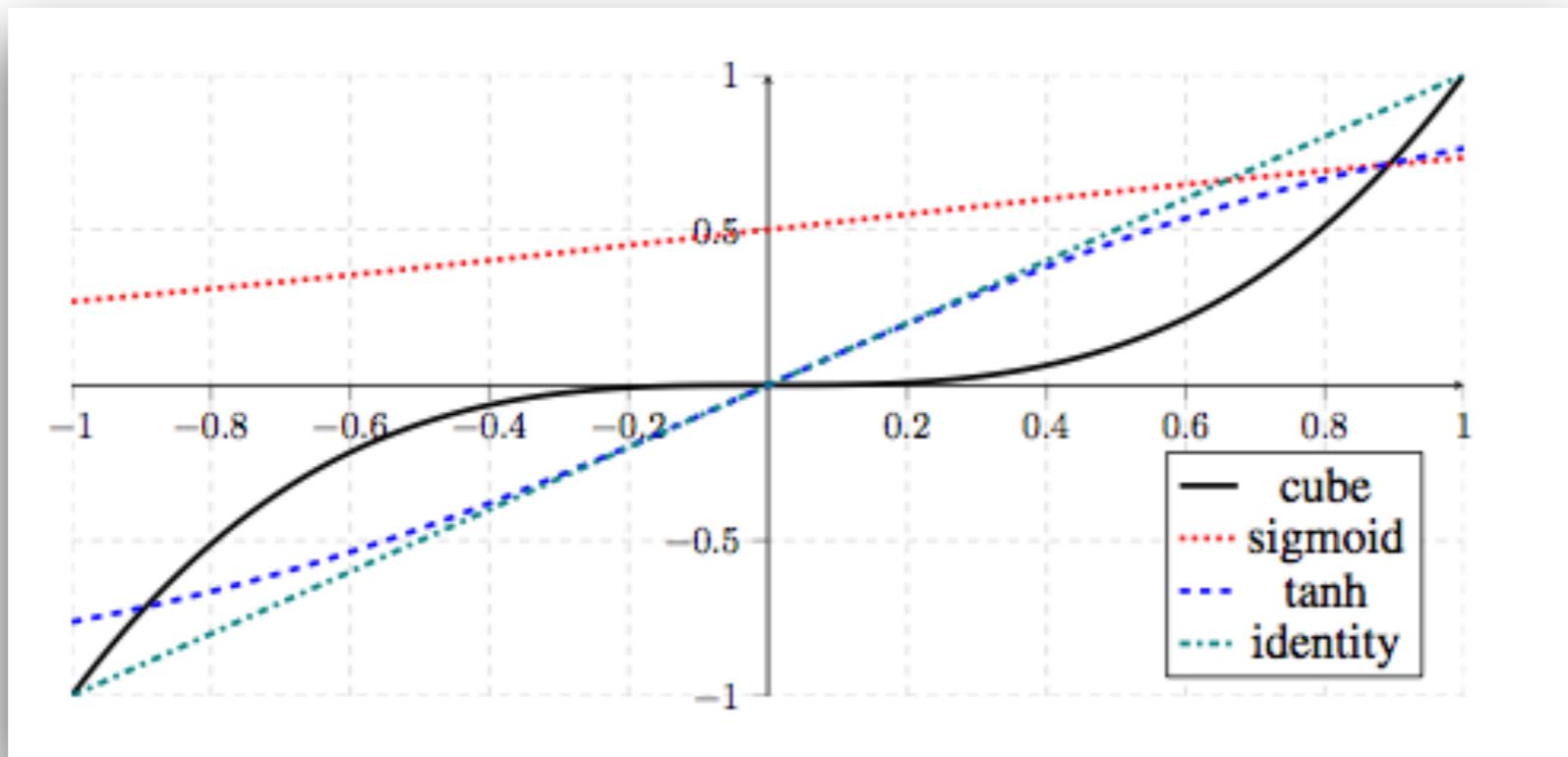


Potential Solution: Use dense neural representations!

Interaction
features???



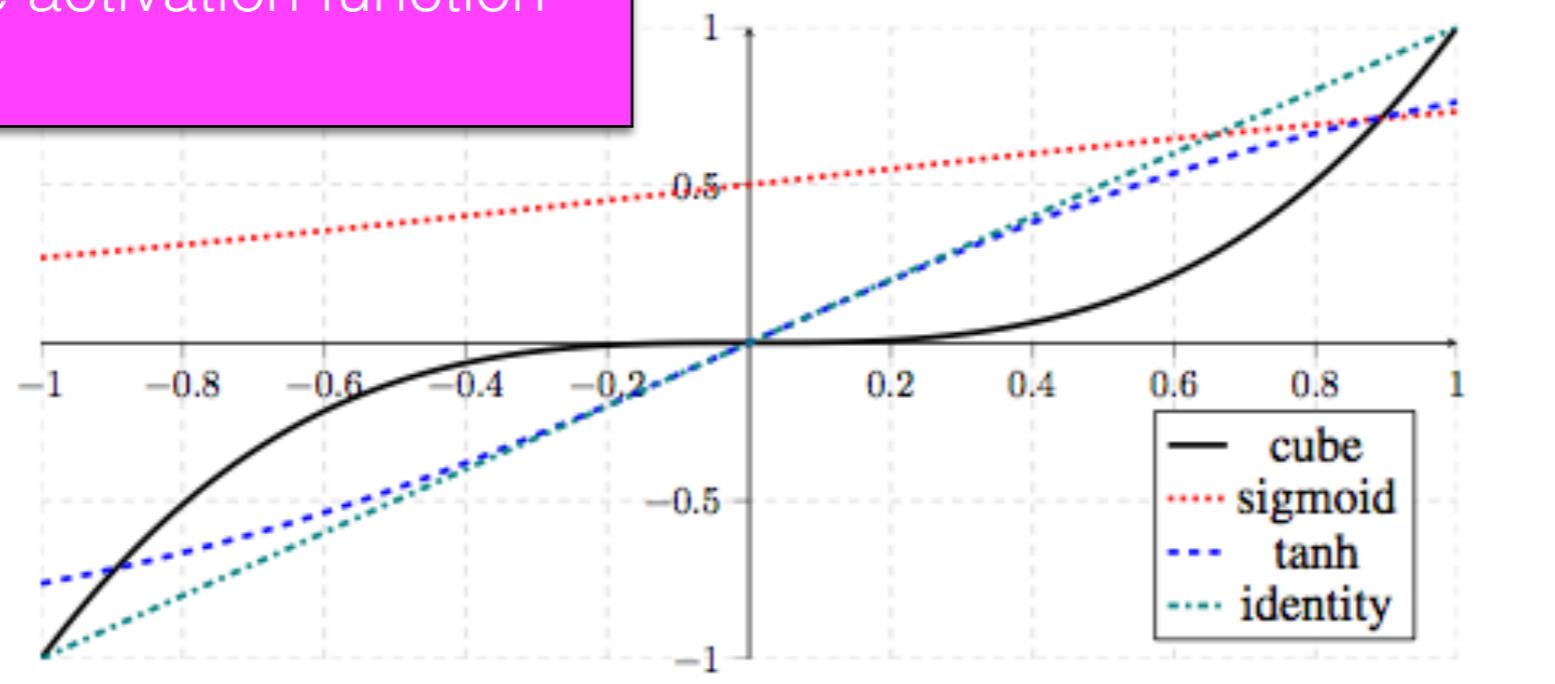
How to capture interactions between words?



$$\begin{aligned} g(w_1x_1 + \dots + w_mx_m + b) = \\ \sum_{i,j,k} (w_i w_j w_k) x_i x_j x_k + \sum_{i,j} b(w_i w_j) x_i x_j \dots \end{aligned}$$

How to capture interactions between words?

Use the cube activation function



$$\begin{aligned} g(w_1x_1 + \dots + w_mx_m + b) = \\ \sum_{i,j,k} (w_iw_jw_k)x_i x_j x_k + \sum_{i,j} b(w_iw_j)x_i x_j \dots \end{aligned}$$

Indicator features vs. (dense) Neural features

- Problem #1: sparse

Indicator features vs. (dense) Neural features

- Problem #1: sparse
 - Solution: Distribution representations capture similarities; also allow generalizing to new words

Indicator features vs. (dense) Neural features

- Problem #1: sparse
 - Solution: Distribution representations capture similarities; also allow generalizing to new words
- Problem #2: incomplete

Indicator features vs. (dense) Neural features

- Problem #1: sparse
 - Solution: Distribution representations capture similarities; also allow generalizing to new words
- Problem #2: incomplete
 - Solution: No need to see all combinations, neural interactions + cube activation provide completeness

Indicator features vs. (dense) Neural features

- Problem #1: sparse
 - Solution: Distribution representations capture similarities; also allow generalizing to new words
- Problem #2: incomplete
 - Solution: No need to see all combinations, neural interactions + cube activation provide completeness
- Problem #3: computationally expensive

Indicator features vs. (dense) Neural features

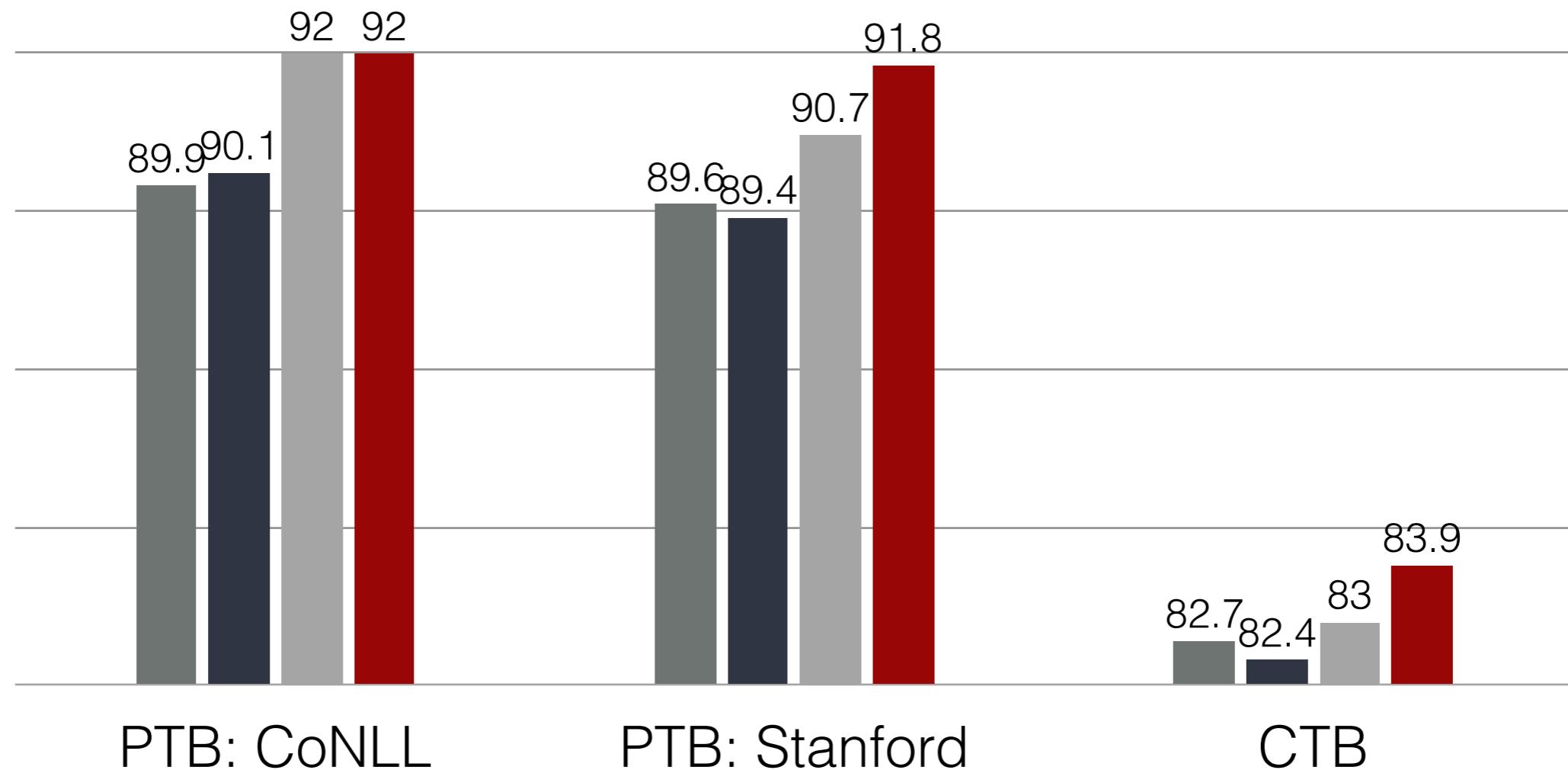
- Problem #1: sparse
 - Solution: Distribution representations capture similarities; also allow generalizing to new words
- Problem #2: incomplete
 - Solution: No need to see all combinations, neural interactions + cube activation provide completeness
- Problem #3: computationally expensive
 - Solution: Matrix representation are small and fast to compute

Experimental Setup

- Embedding size = 50
- Hidden size = 200
- Use mini-batched AdaGrad for optimization ($\alpha = 0.01$)
- Use 0.5 dropout on hidden layer.
- Pre-trained word vectors:
 - Collobert & Weston for English (a word2vec like approach)
 - Word2vec for Chinese
- 18 tokens from the configuration are encoded as state

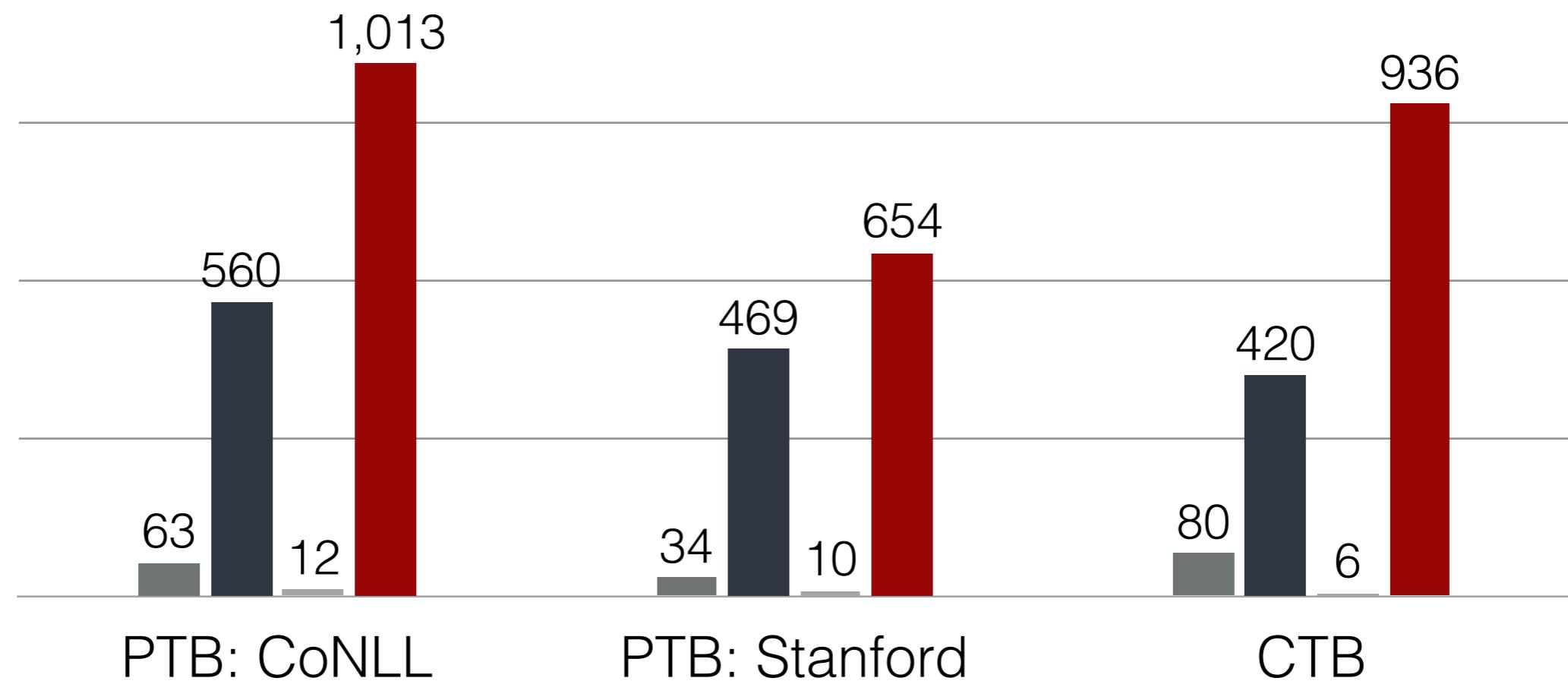
Unlabeled Attachment Score (UAS) Performance

- Standard / eager
- Malt (stackproj / nirveeager)
- MST
- Chen and Manning (2014)

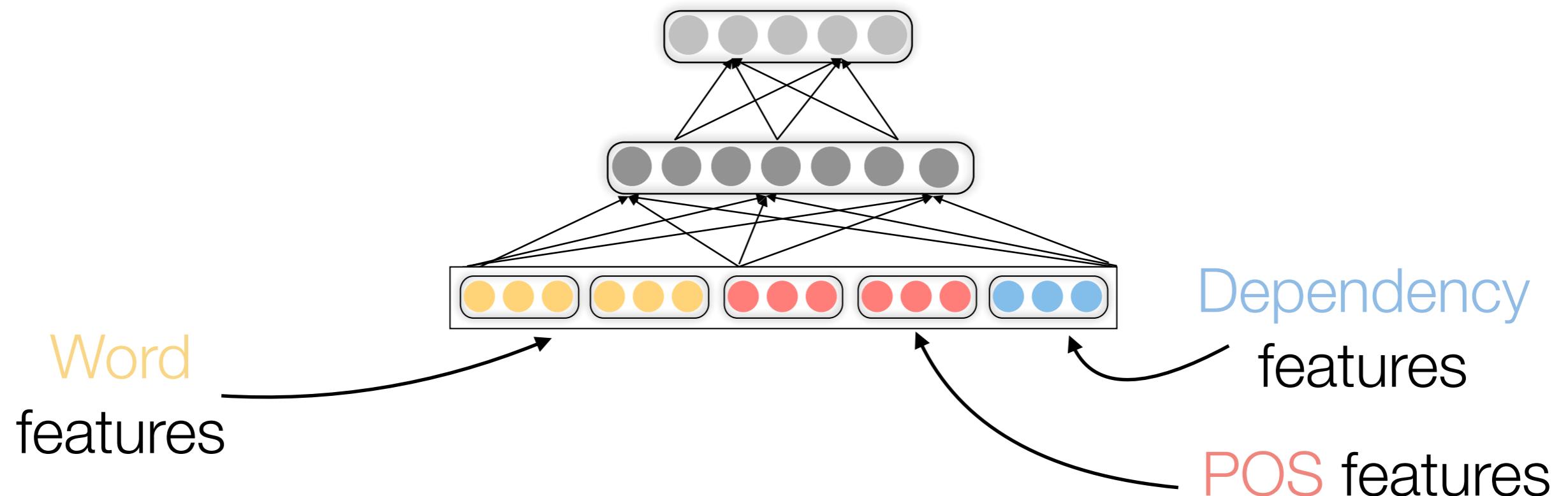


Neural methods can be quick!

- Standard / eager
 - Malt (stackproj / nirveeager)
 - MST
 - Chen and Manning (2014)
-



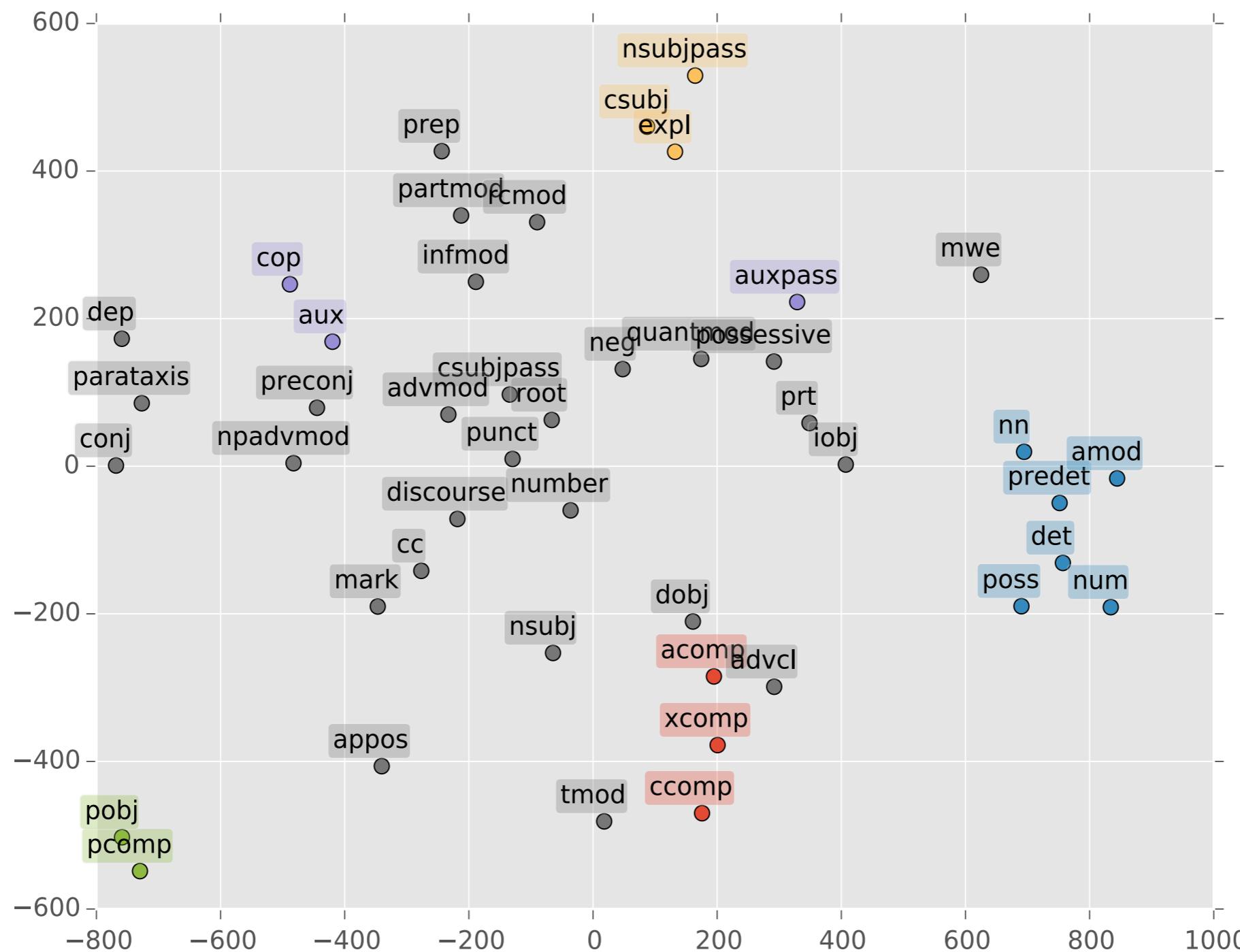
Thinking about the model's parameters a bit more....



What was learned in the POS Embeddings?



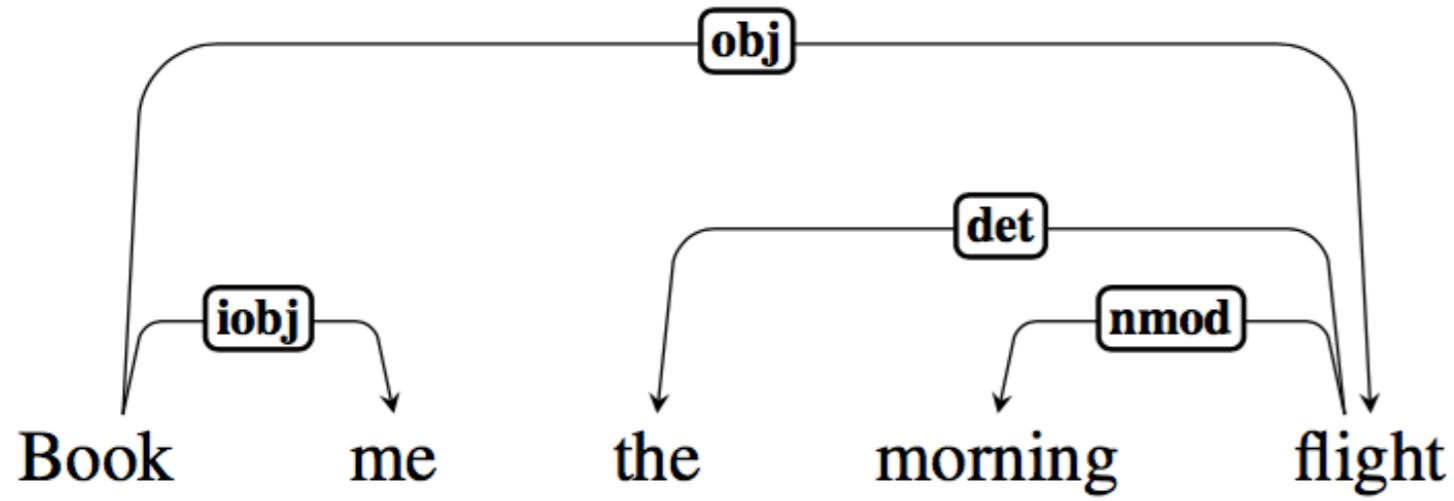
What was learned in the dependency embeddings?



SCIENCE
THEATER
3000

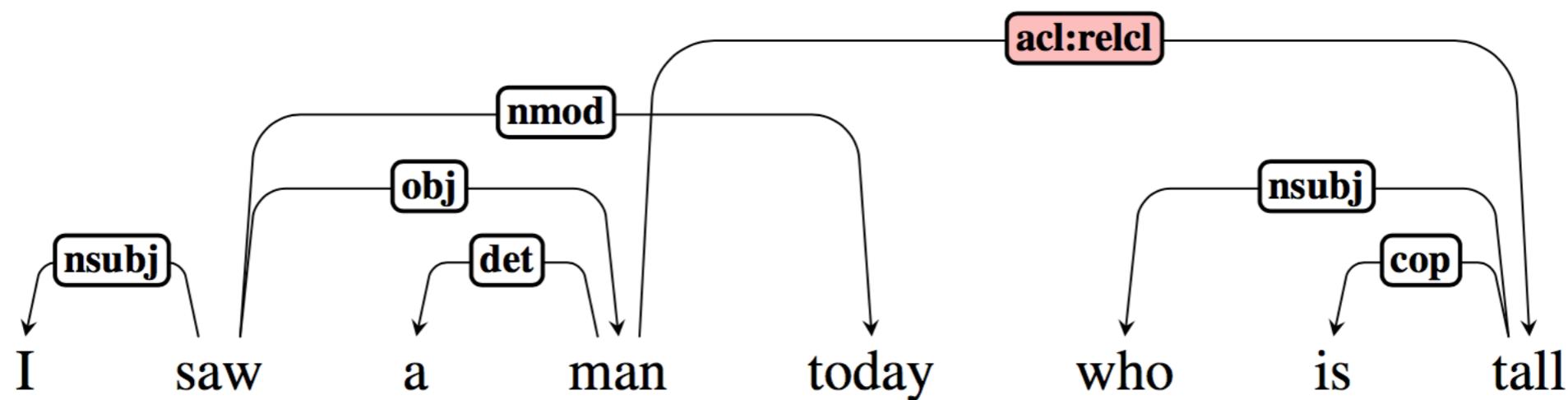


Graph-Based Parsing



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

Projectivity



- What happens if you run an oracle on a non-projective sentence?

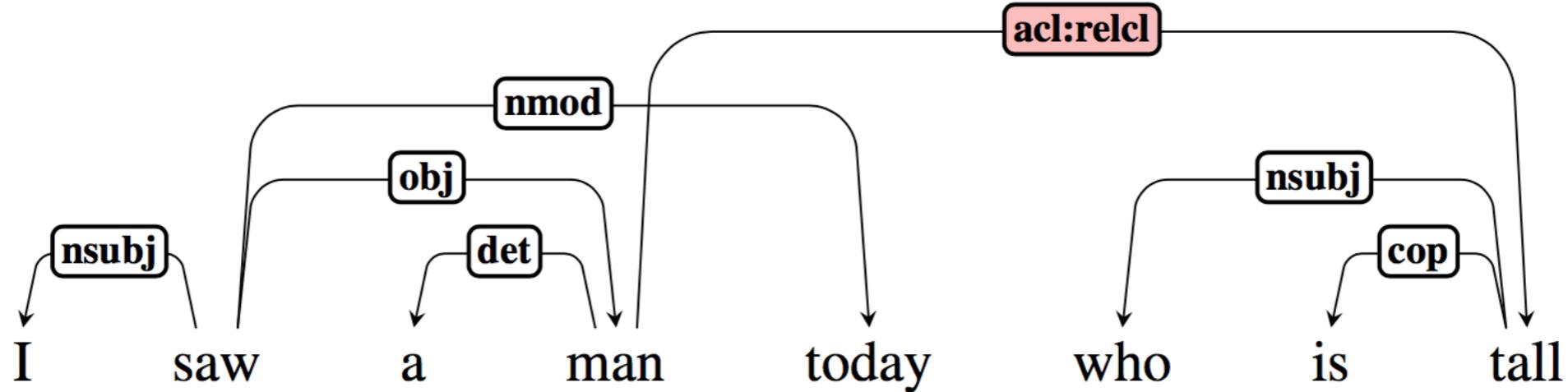
Graph-based parsing

- For a given sentence S , we want to find the highest-scoring tree among all possible trees for that sentence \mathcal{G}_S

$$\hat{T}(S) = \arg \max_{t \in \mathcal{G}_S} \text{score}(t, S)$$

- Edge-factored scoring: the total score of a tree is the sum of the scores for all of its edges (arcs):

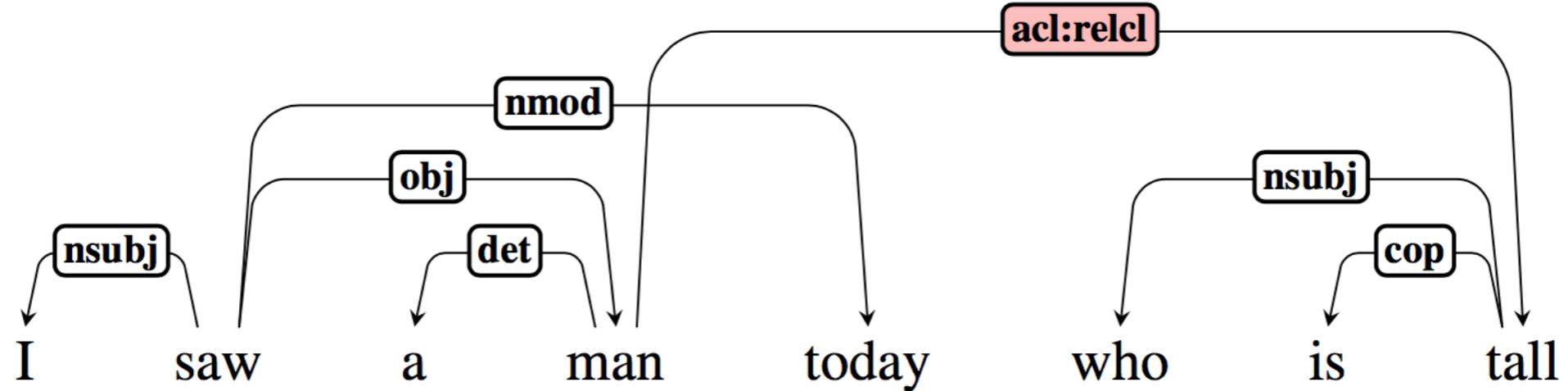
$$\text{score}(t, S) = \sum_{e \in t} \text{score}(e)$$



Edge-factored features

- Word form of head/dependent
- POS tag of head/dependent
- Distributed representation of h/d
- Distance between h/d
- POS tags between h/d
- Head to left of dependent?

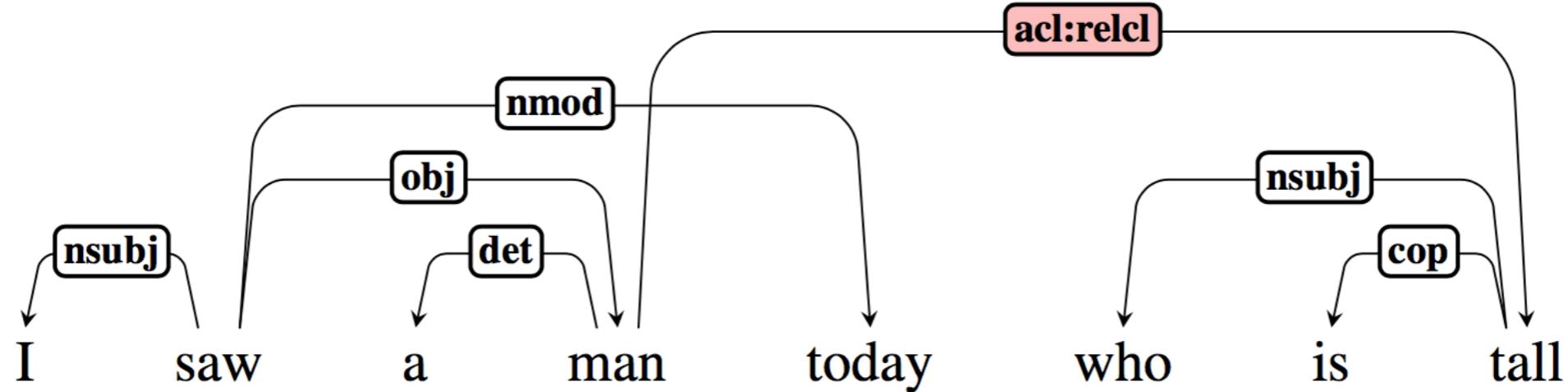
$\text{head}_t = \text{man}$	1
$\text{head}_{\text{pos}} = \text{NN}$	1
distance	4
$\text{child}_{\text{pos}} = \text{JJ}$ and $\text{head}_{\text{pos}} = \text{NN}$	1
$\text{child}_{\text{pos}} = \text{NN}$ and $\text{head}_{\text{pos}} = \text{JJ}$	0



$$\text{score}(e) = \sum_{i=1}^F x_i \beta_i$$

Feature value

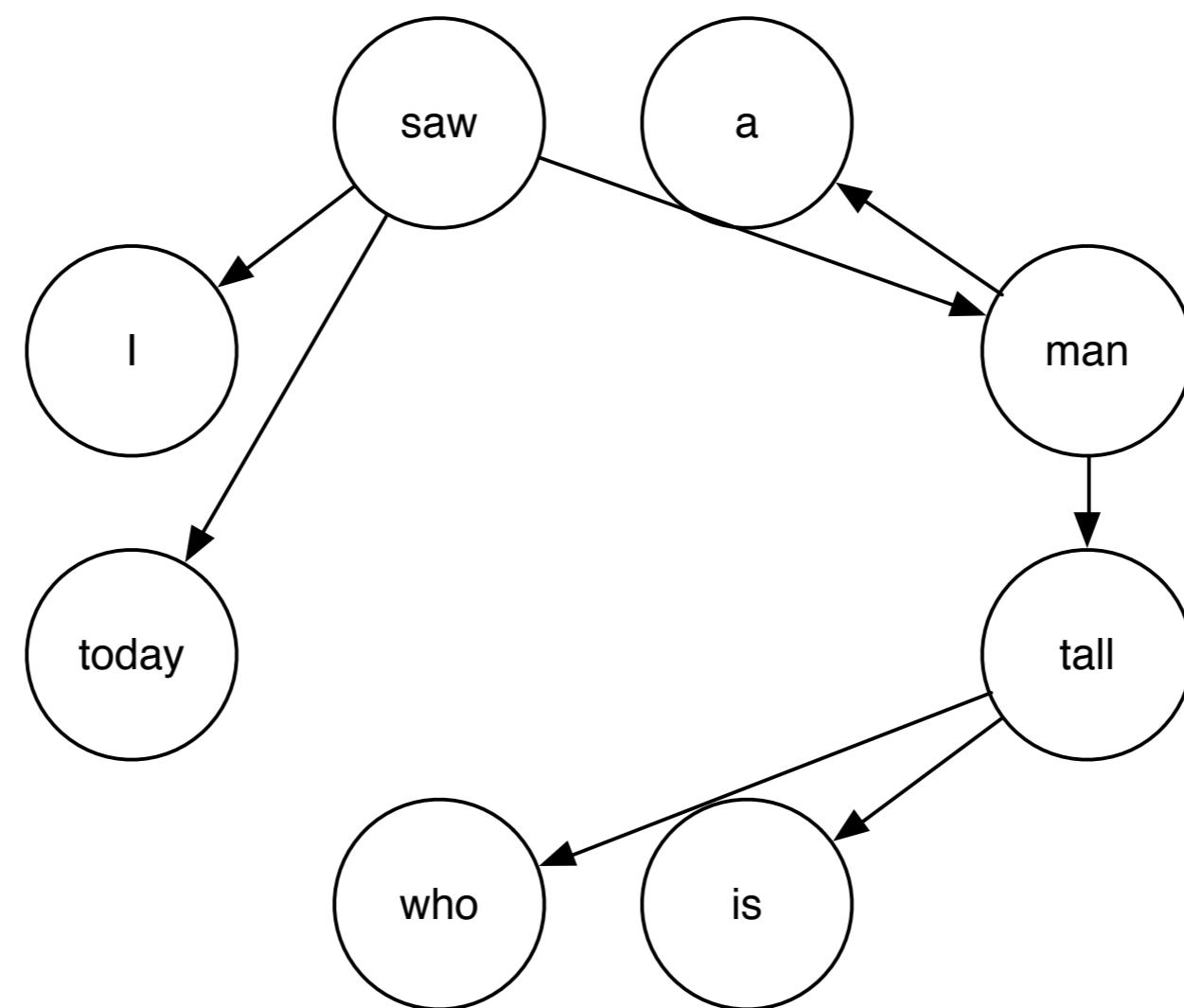
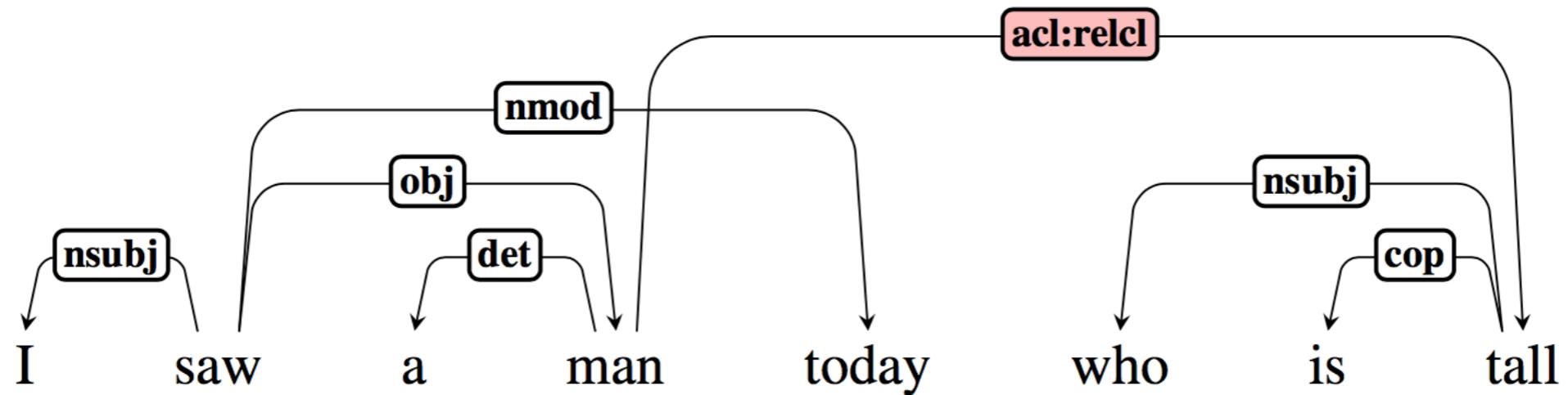
Learned coefficient for that feature



$$\text{score}(e) = \sum_{i=1}^F x_i \beta_i$$

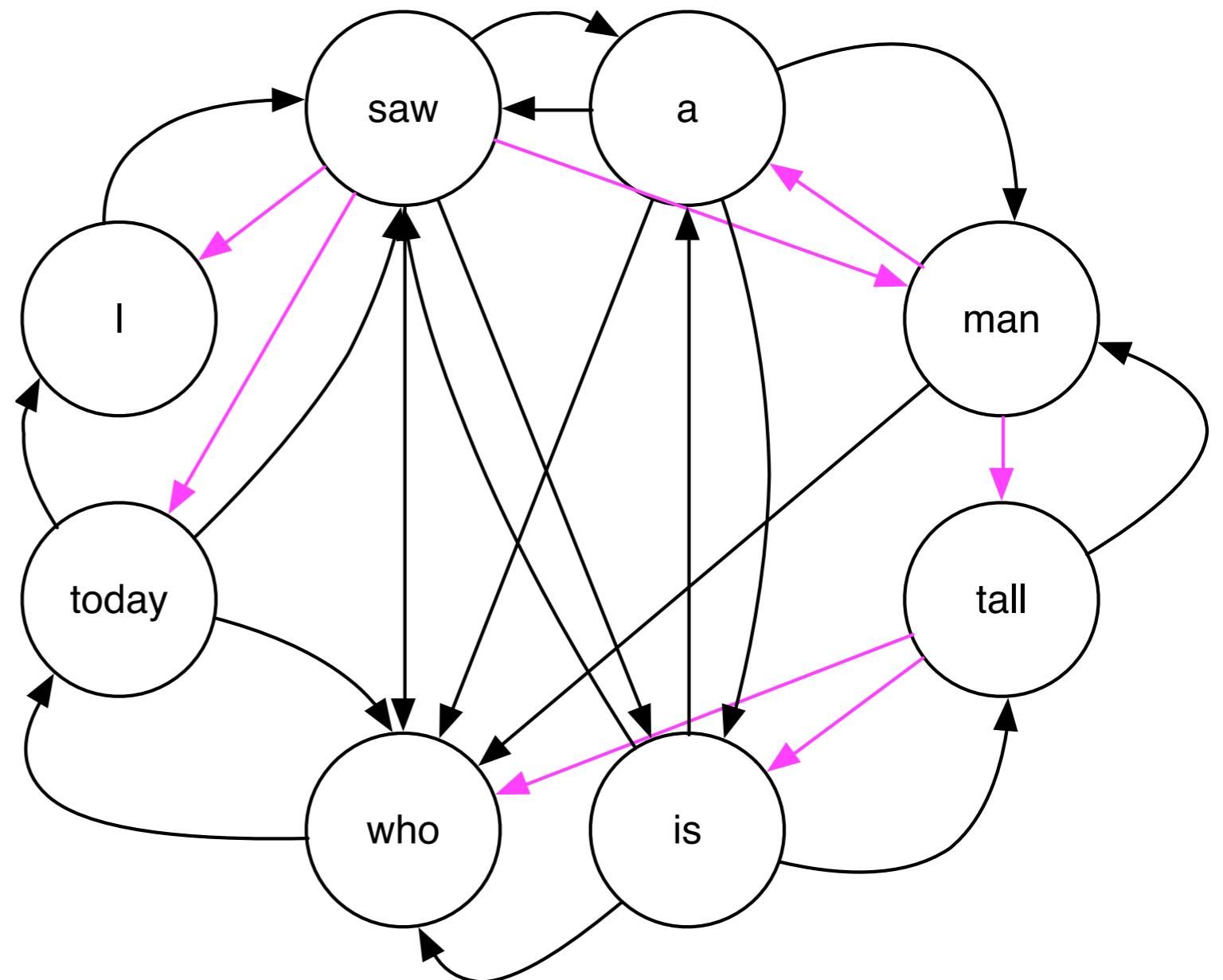
x	β
head _t = man	1
head _t = man	1
distance	4
child _{pos} = JJ and	1
child _{pos} = NN and	0

$$\text{score}(e) = 8.1$$



MST Parsing

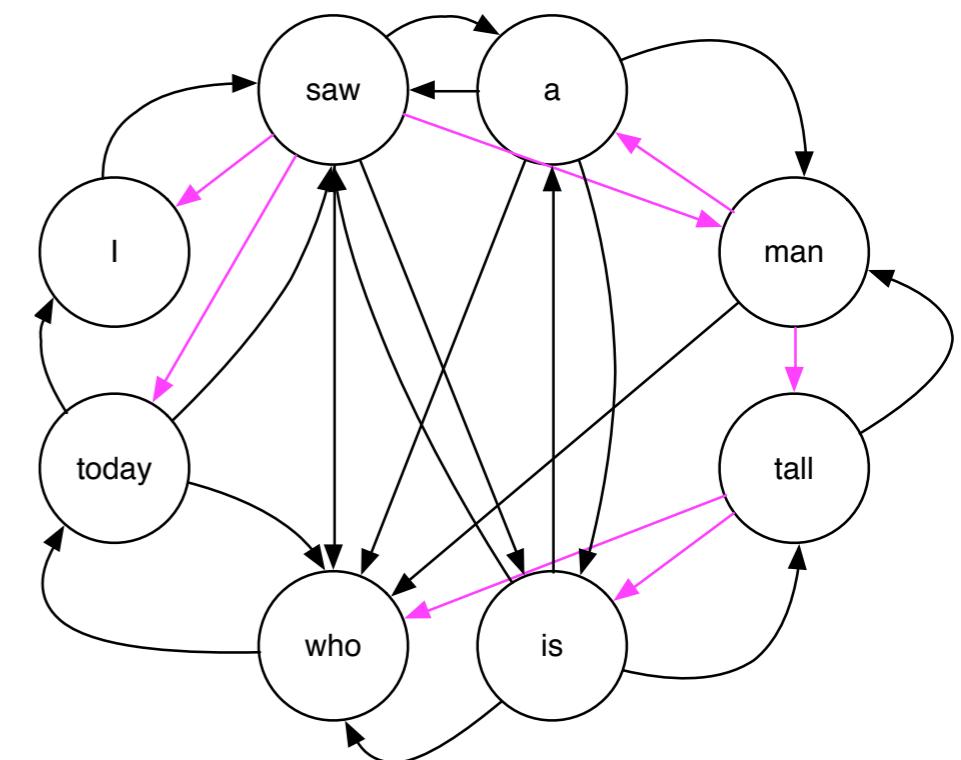
- We start out with a fully connected graph with a score for each edge
- N^2 edges total



(Assume one edge connects each node as dependent and node as head, N^2 total)

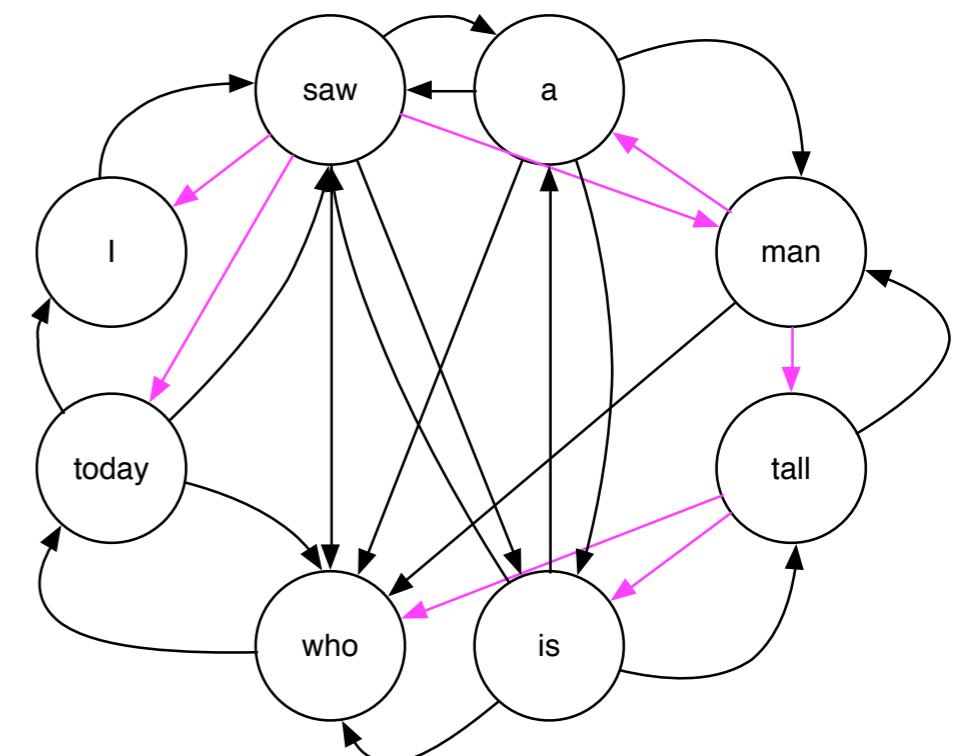
Maximal Spanning Tree (MST) Parsing

- From this graph G , we want to find a **spanning tree** (tree that spans G [includes all the vertices in G])
- If the edges have weights, the best parse is the **maximal spanning tree** (the spanning tree with the highest total weight).



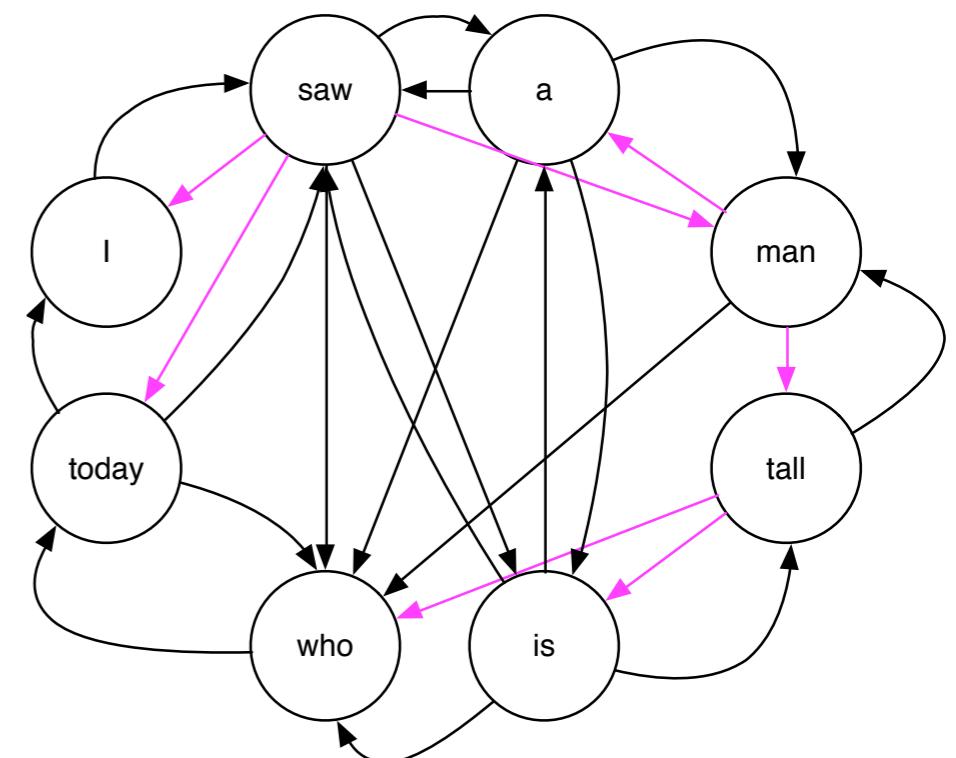
Maximal Spanning Tree (MST) Parsing

- To find the MST of any graph, we can use the Chu-Liu-Edmonds algorithm in $O(n^3)$ time.
- More efficient Gabow et al. find the MST in $O(n^2+n \log n)$



Maximal Spanning Tree (MST) Parsing

- To find the MST of any graph, we can use the Chu-Liu-Edmonds algorithm in $O(n^3)$ time.
- More efficient Gabow et al. find the MST in $O(n^2+n \log n)$



Lots of cool and difficult algorithms for how to learn how to score weights efficiently, but we're not going to discuss them

Preview of Homework 3!

Preview of Homework 3!



Implement* the Chen and
Manning (2014) neural parser!

Implement* the Chen and Manning (2014) neural parser!

- Lots of skeleton code provided

Implement* the Chen and Manning (2014) neural parser!

- Lots of skeleton code provided
- You get to implement
 - the neural network in PyTorch
 - the core training loop (also in PyTorch)

Implement* the Chen and Manning (2014) neural parser!

- Lots of skeleton code provided
- You get to implement
 - the neural network in PyTorch
 - the core training loop (also in PyTorch)
- Leverage the power of Deep Learning libraries 😍

Quick Example: Logistic Regression in PyTorch

Quick Example: Logistic Regression in PyTorch

```
# Model
class LogisticRegression(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.betas = nn.Linear(input_size, num_classes) #defines a layer

    def forward(self, x):
        out = self.betas(x)
        return out
```

Quick Example: Logistic Regression in PyTorch

```
# Model
class LogisticRegression(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.betas = nn.Linear(input_size, num_classes) #defines a layer

    def forward(self, x):
        out = self.betas(x)
        return out

criterion = nn.CrossEntropyLoss() #how to measure loss
model = LogisticRegression(input_size, num_classes)
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Quick Example: Logistic Regression in PyTorch

```
# Model
class LogisticRegression(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.betas = nn.Linear(input_size, num_classes) #defines a layer

    def forward(self, x):
        out = self.betas(x)
        return out

criterion = nn.CrossEntropyLoss() #how to measure loss
model = LogisticRegression(input_size, num_classes)
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    for i, (feature_vec, label) in enumerate(training_data):
        feature_vec = torch.tensor(feature_vec)
        label = torch.tensor(label)
```

Quick Example: Logistic Regression in PyTorch

```
# Model
class LogisticRegression(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.betas = nn.Linear(input_size, num_classes) #defines a layer

    def forward(self, x):
        out = self.betas(x)
        return out

criterion = nn.CrossEntropyLoss() #how to measure loss
model = LogisticRegression(input_size, num_classes)
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    for i, (feature_vec, label) in enumerate(training_data):
        feature_vec = torch.tensor(feature_vec)
        label = torch.tensor(label)

        # Forward + Backward + Optimize
        optimizer.zero_grad() # clears out any lingering gradient
        output = model(feature_vec) # == model.forward(feature_vec)
        loss = criterion(output, label)
        loss.backward() # Computes the gradient
        optimizer.step() # Updates the parameters
```

What you should know

- Language has lots of structure
- Constituency parsing builds syntax trees where words are terminal (leaf) nodes and internal non-terminal nodes describe how child nodes relate syntactically
- How the CKY algorithm works at a high level to produce constituency parse trees
- How shift reduce parsing works
- Dependency parsing creates trees with explicit syntactic (and almost semantic) relationships between words