

Natural Language Processing

Nuri Cingillioglu

<https://www.doc.ic.ac.uk/~nuric/>

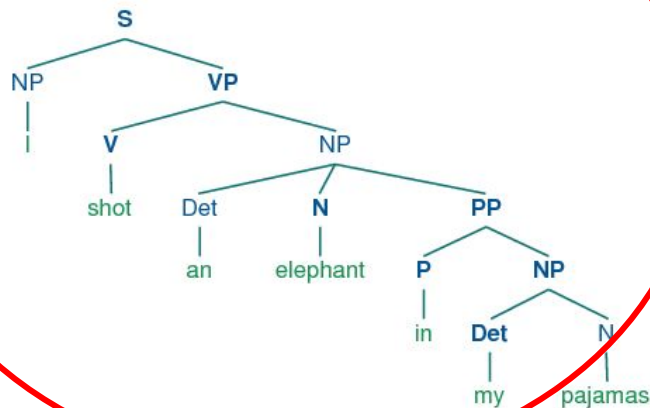
Many thanks to Lucia Specia

Constituency parsing

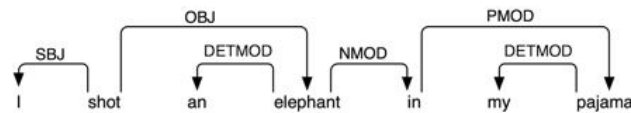
Definition

- Given a sequence of words (usually a sentence), generate its **syntactic structure**

Constituency Parsing



Dependency Parsing

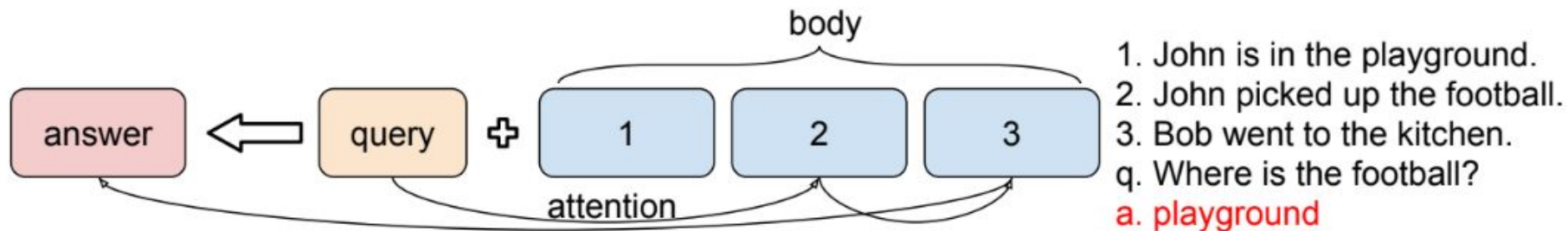


Applications

- Grammar checking
- Semantic analysis
 - Question answering
 - Named entity recognition
- Machine translation
- ...

Neuro-symbolic reasoning

Y:office ← Where is the X:apple? ,
Z:Mary picked up the X:apple. ,
Z:Mary went to the Y:office.



Challenges

- Harder than POS tagging:
 - One label per group of words (any length in principle)
 - Structural plus POS ambiguity:

Minister accused of having 8 wives in jail (PP attachment)

Ban on Nude Dancing on Governor's Desk

Juvenile Court to Try Shooting Defendant

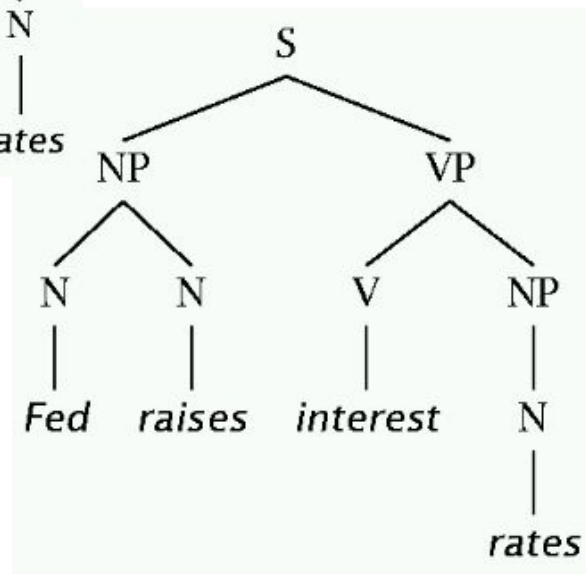
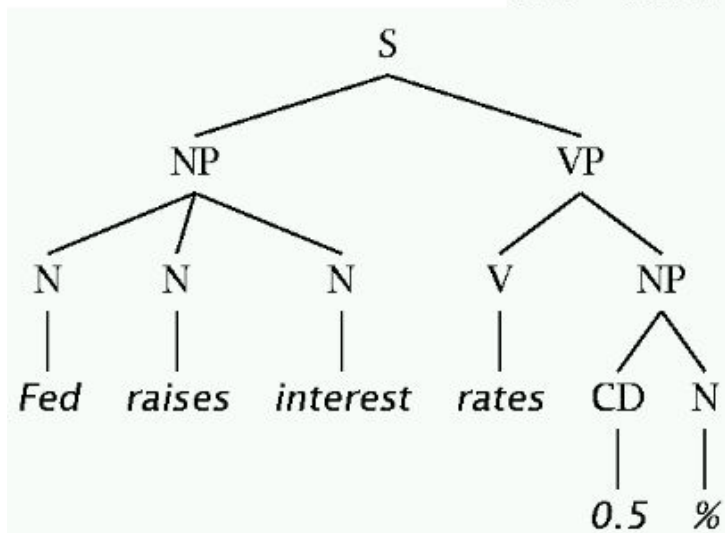
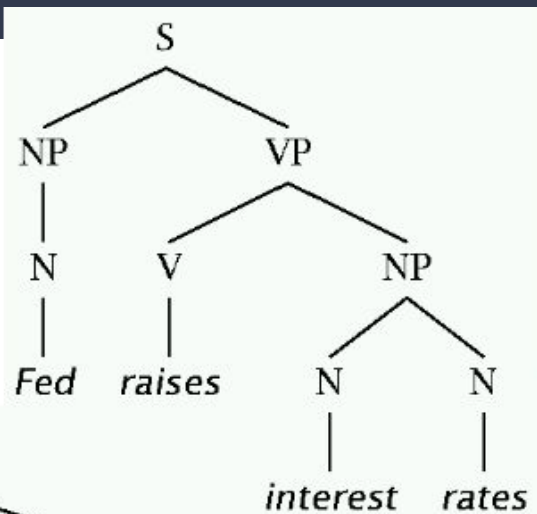
Stolen Painting Found by Tree

Fed raises interest rates 0.5% in effort to control inflation

Beautiful dogs and children (coordination)

Challenges

Example by Chris Manning



Classical parsing

- Given a
grammar
and a
lexicon

Grammar	Lexicon
$S \rightarrow NP VP$	$Det \rightarrow that \mid this \mid the \mid a$
$S \rightarrow Aux NP VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper-Noun$	$Proper-Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal Noun$	
$Nominal \rightarrow Nominal PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb NP$	
$VP \rightarrow Verb NP PP$	
$VP \rightarrow Verb PP$	
$VP \rightarrow VP PP$	
$PP \rightarrow Preposition NP$	

Classical parsing

- ... and a new sentence, e.g.

The flight includes a meal

- Goal: Generate the structure for the sentence

Constituency parsing

- Based on the idea of phrase structure
 - Words are grouped into constituents
- A constituent is a sequence of words that behaves as a **unit**, generally a phrase
 - There are tests for that, e.g. distribution: can I move phrases around?
 - John talked [to the children] [about drugs].
 - John talked [about drugs] [to the children].

Classical parsing – more formally

Phrase structure grammar is a **context-free grammar**

$G = (T, N, S, R)$, where:

- T is set of terminals
- N is set of nonterminals
- S is the start symbol (non-terminal)
- R is set of rules $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals & nonterminals

A grammar G generates a language L, or L is recognised by G.

Classical parsing

- To do this automatically
 - Use proof systems to prove parse trees from words
 - Search problem: all possible parse trees for string
 - Bottom-up
 - Words to grammar
 - Top-down
 - Grammar to words

The CKY algorithm

- Tests for possibilities to split the current sequence into **two** smaller sequences
- Grammar needs to be in Chomsky Normal Form (CNF)
 - Rules are of the form $X \rightarrow Y Z$ or $X \rightarrow w$
 - Deterministic process to convert any CFG into CNF

$VP \rightarrow V NP PP \quad \longrightarrow \quad VP \rightarrow V NEW$
 $NEW \rightarrow NP PP$

$INF-VP \rightarrow to VP \quad \longrightarrow \quad INF-VP \rightarrow TO VP$
 $TO \rightarrow to$

The CKY algorithm

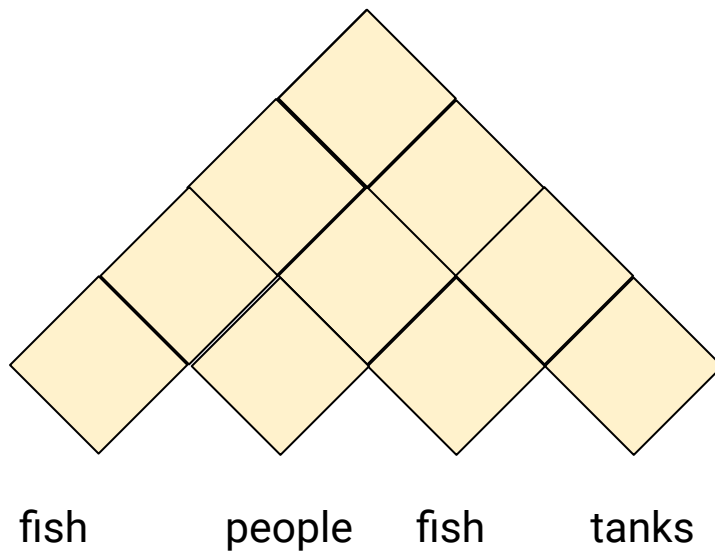
- Binarisation makes CKY very **efficient**
 - $O(n^3|G|)$: n is the length of the parsed string; $|G|$ is the size of the CNF grammar G
 - Otherwise it would be exponential
- **Dynamic programming** algorithm to efficiently generate all possible parse trees bottom-up

The CKY algorithm

- Use a two-dimensional matrix $n \times n$, where n is length of sentence, to encode the possible parses
- Use the upper-triangular portion of the matrix
- Each cell $[i, j]$ in matrix contains the set of **non-terminals** that represent all the constituents that span positions i - j of the input
- The cell that represents the entire input resides in position $[0, n]$ of the matrix (row x column)

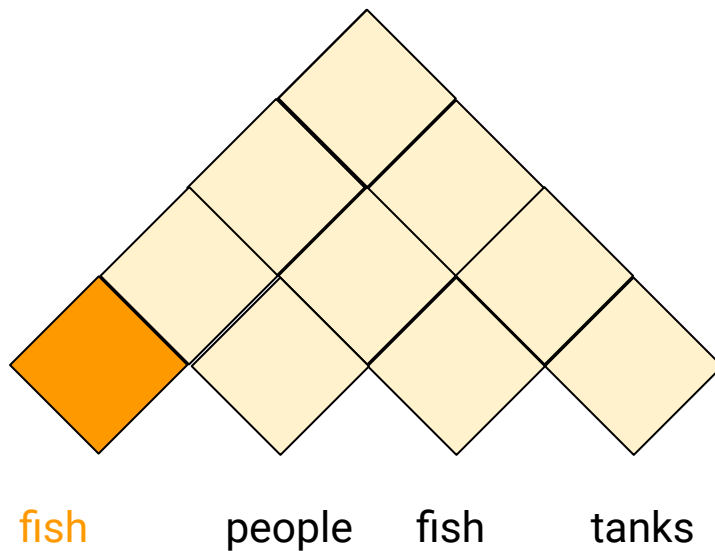
The CKY algorithm – intuition

Example by Chris Manning



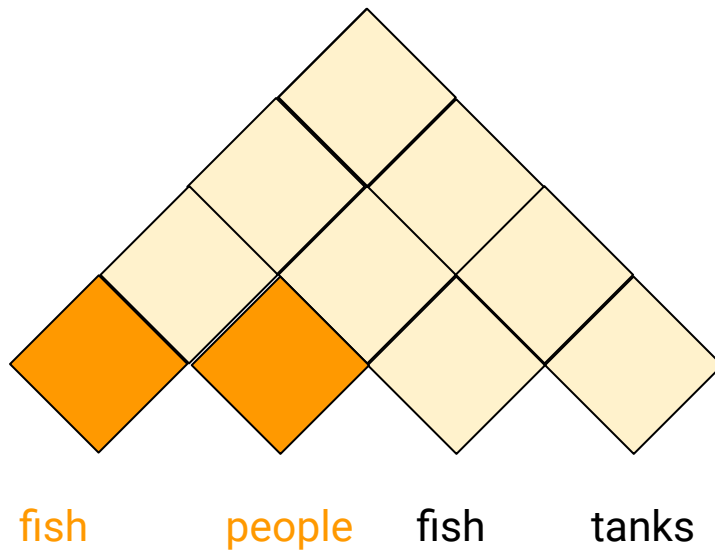
The CKY algorithm – intuition

Example by Chris Manning



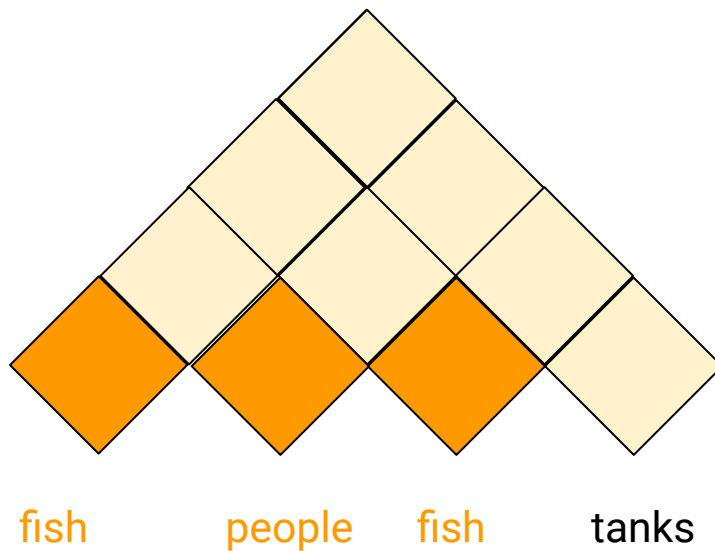
The CKY algorithm – intuition

Example by Chris Manning



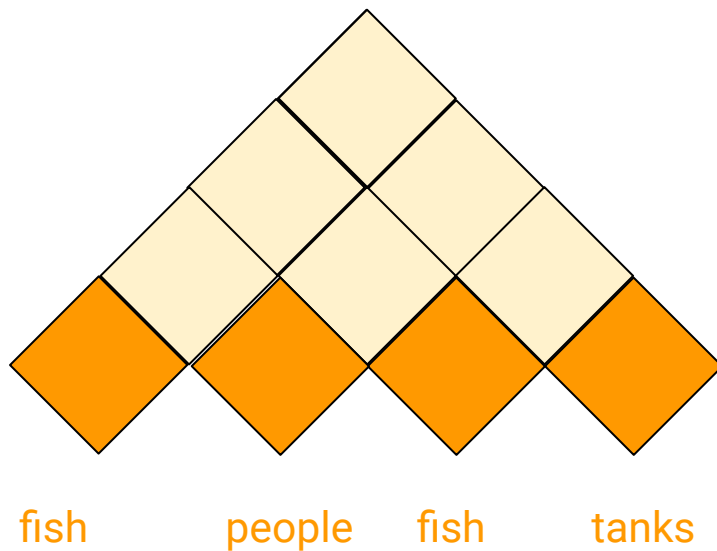
The CKY algorithm – intuition

Example by Chris Manning



The CKY algorithm – intuition

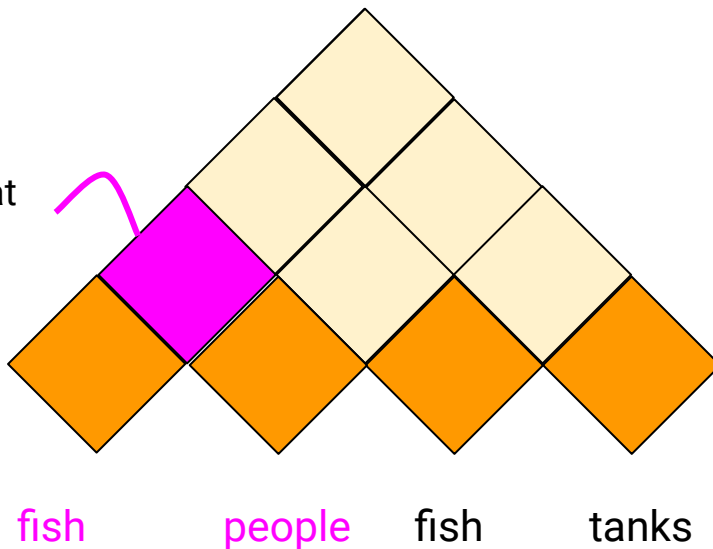
Example by Chris Manning



The CKY algorithm – intuition

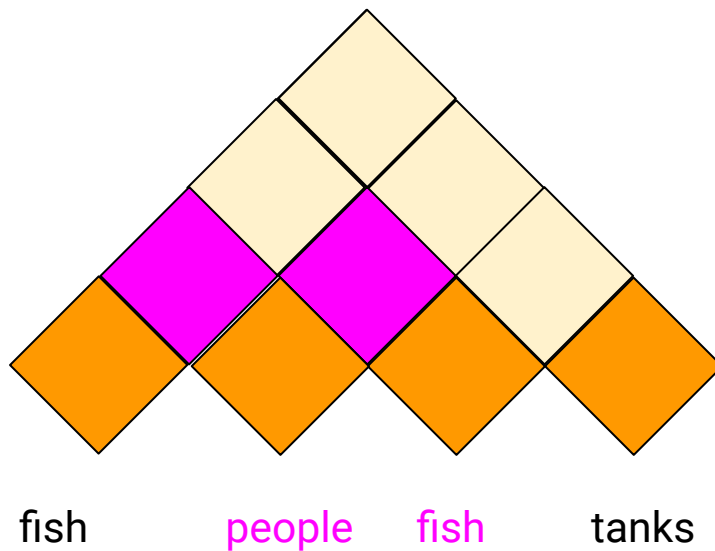
Example by Chris Manning

Is there a rule that
allows me to put
these two words
together in a
sequence?



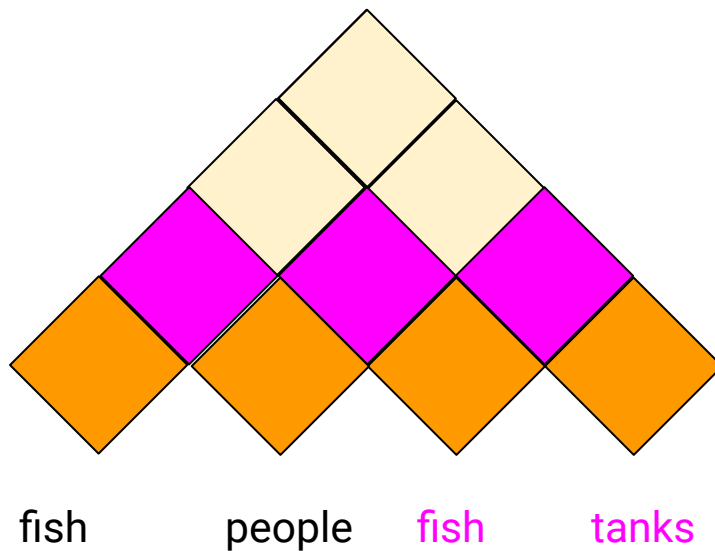
The CKY algorithm – intuition

Example by Chris Manning



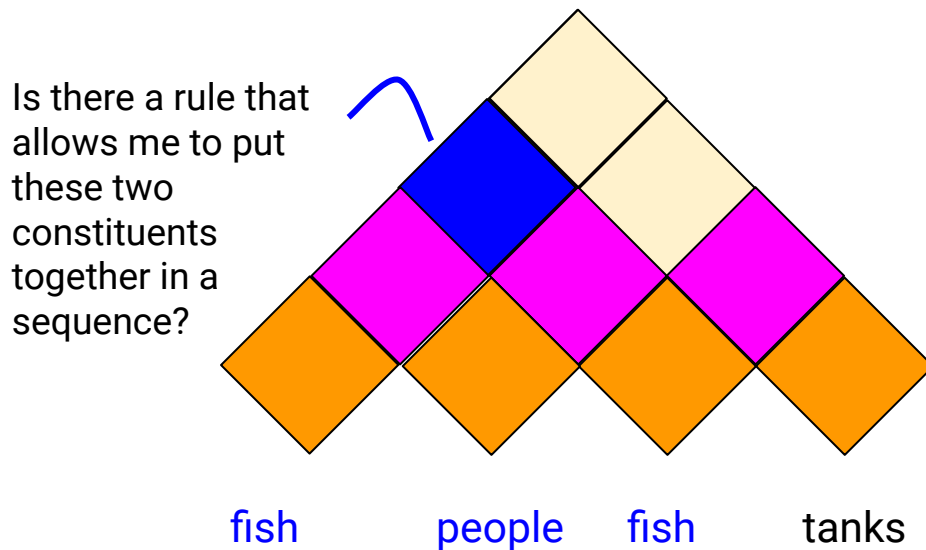
The CKY algorithm – intuition

Example by Chris Manning



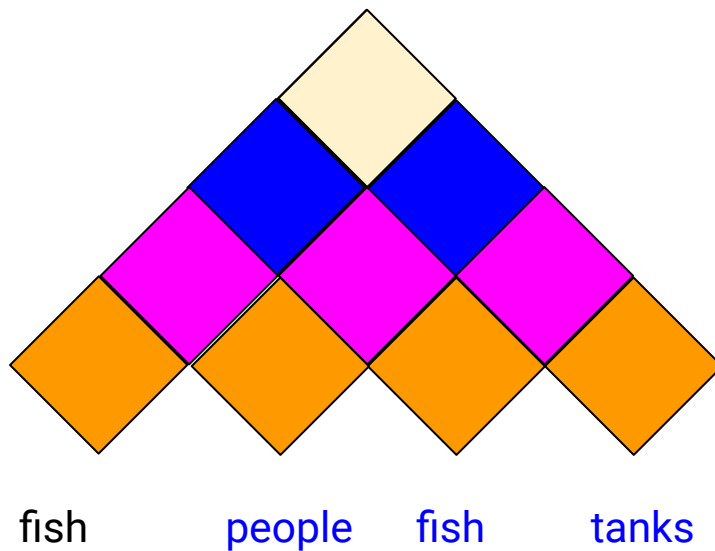
The CKY algorithm – intuition

Example by Chris Manning



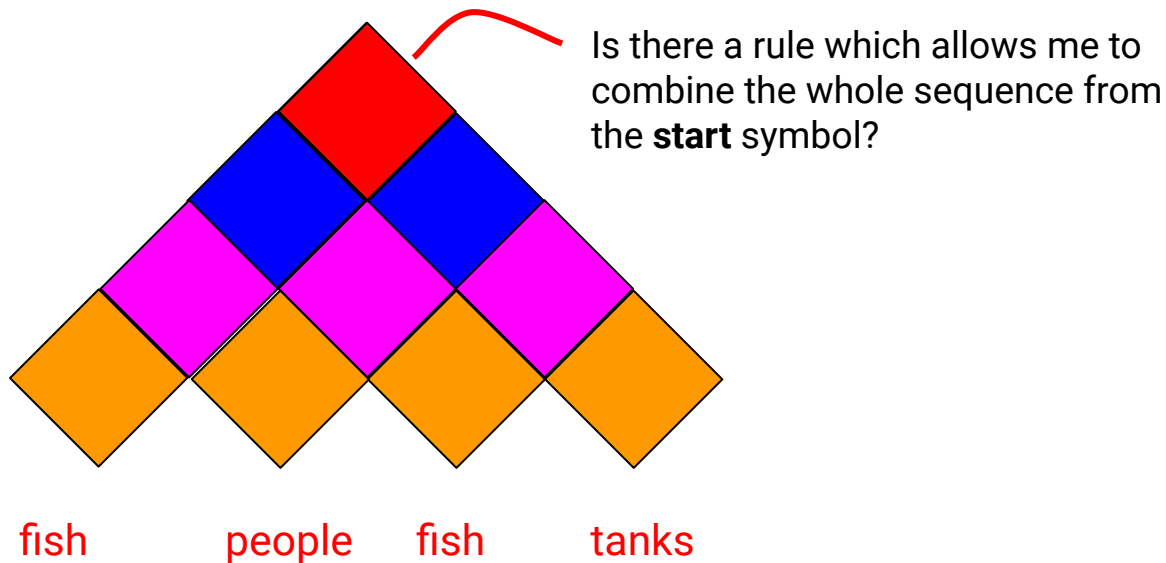
The CKY algorithm – intuition

Example by Chris Manning



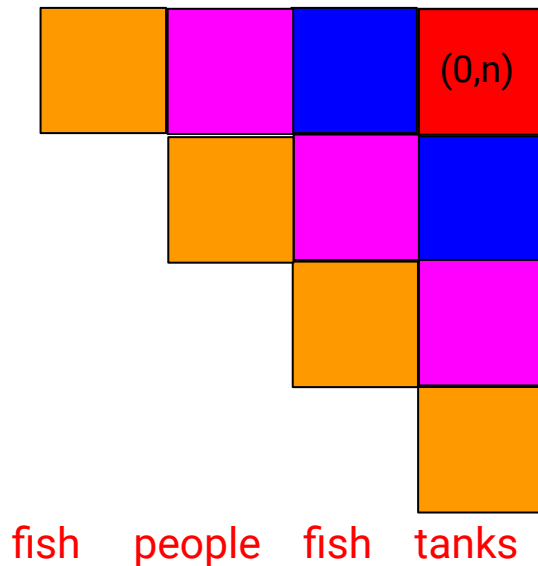
The CKY algorithm – intuition

Example by Chris Manning



The CKY algorithm – intuition

Example by Chris Manning



The CKY algorithm - example

$$S \rightarrow NP VP$$
$$VP \rightarrow VP PP$$
$$VP \rightarrow V NP$$

VP → eats

$$PP \rightarrow P \ NP$$

NP → Det N

NP → she

V → eats

P → with

N → fish

N → fork

 $\text{Det} \rightarrow a$ [illegible]

Adapted from:

https://upload.wikimedia.org/wikipedia/commons/f/f5/CYK_algorithm_animation_showing_every_step_of_a_sentence_parsing.gif

The CKY algorithm – example

S → NP VP
VP → VP PP
VP → V NP
VP → eats
PP → P NP
NP → Det N
NP → she
V → eats
P → with
N → fish
N → fork
Det → a

She	eats	a	fish	with	a	fork
NP						
	V, VP					
		Det				
			N			
				P		
					Det	
						N

The CKY algorithm – example

S → NP VP
VP → VP PP
VP → V NP
VP → eats
PP → P NP
NP → Det N
NP → she
V → eats
P → with
N → fish
N → fork
Det → a

She	eats	a	fish	with	a	fork
NP						
	V, VP					
		Det				
			N			
				P		
					Det	
						N

Are there rules of the type
 $? \rightarrow \text{NP V}$
or
 $? \rightarrow \text{NP VP}$

The CKY algorithm – example

S → NP VP
VP → VP PP
VP → V NP
VP → eats
PP → P NP
NP → Det N
NP → she
V → eats
P → with
N → fish
N → fork
Det → a

She	eats	a	fish	with	a	fork
NP	S					
	V, VP					
		Det				
			N			
				P		
					Det	
						N

Are there rules of the type
 $? \rightarrow NP V$
or
 $? \rightarrow NP VP$

The CKY algorithm – example

S → NP VP
VP → VP PP
VP → V NP
VP → eats
PP → P NP
NP → Det N
NP → she
V → eats
P → with
N → fish
N → fork
Det → a

She	eats	a	fish	with	a	fork
NP	S					
	V, VP					
		Det				
			N			
				P		
					Det	
						N

Are there rules of the type
 $? \rightarrow V \text{ Det}$
or
 $? \rightarrow VP \text{ Det}$

The CKY algorithm – example

S → NP VP
VP → VP PP
VP → V NP
VP → eats
PP → P NP
NP → Det N
NP → she
V → eats
P → with
N → fish
N → fork
Det → a

She	eats	a	fish	with	a	fork
NP	S					
	V, VP	-----				
		Det				
			N			
				P		
					Det	
						N

Are there rules of the type
? → V Det
or
? → VP Det

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

She	eats	a	fish	with	a	fork
NP	S					
	V, VP	-----				
		Det				
			N			
				P		
					Det	
						N

Are there rules of the type
 ? → Det N

The CKY algorithm – example

S → NP VP
VP → VP PP
VP → V NP
VP → eats
PP → P NP
NP → Det N
NP → she
V → eats
P → with
N → fish
N → fork
Det → a

She	eats	a	fish	with	a	fork
NP	S					
	V, VP	-----				
		Det	NP			
			N			
				P		
					Det	
						N

Are there rules of the type
? → Det N

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

She	eats	a	fish	with	a	fork
NP	S					
	V, VP	-----				
		Det	NP			
			N	-----		
				P	-----	
					Det	NP
						N

Keep going for each 2 words

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

She	eats	a	fish	with	a	fork
NP	S					
	V, VP	-----				
		Det	NP			
			N	-----		
				P	-----	
					Det	NP
						N

Sequences of 3 words, but
 our grammar is binary. So
 check for rules for
 combining words in 2
 substrings
 ? → <she> <eats a>
 ? → <she eats> <a>

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

? → NP -----

She	eats	a	fish	with	a	fork
NP	S					
	V, VP	-----				
		Det	NP			
			N	-----		
				P	-----	
					Det	NP
						N

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

? → S Det

She	eats	a	fish	with	a	fork
NP	S					
	V, VP	-----				
		Det	NP			
			N	-----		
				P	-----	
					Det	NP
						N

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

? → S Det

She	eats	a	fish	with	a	fork
NP	S	-----				
	V, VP	-----				
		Det	NP			
			N	-----		
				P	-----	
					Det	NP
						N

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

She	eats	a	fish	with	a	fork
NP	S	-----				
	V, VP	-----				
		Det	NP			
			N	-----		
				P	-----	
					Det	NP
						N

? → <eats> <a fish>

? → <eats a> <fish>

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

She	eats	a	fish	with	a	fork
NP	S	-----				
	V, VP	-----				
		Det	NP			
			N	-----		
				P	-----	
					Det	NP
						N

? → V NP

? → VP NP

(both V and VP → eats)

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

She	eats	a	fish	with	a	fork
NP	S	-----				
	V, VP	-----	VP			
		Det	NP			
			N	-----		
				P	-----	
					Det	NP
						N

? → V NP

? → VP NP

(both V and VP → eats)

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

? → ----- N

She	eats	a	fish	with	a	fork
NP	S	-----				
	V, VP	-----	VP			
		Det	NP			
			N	-----		
				P	-----	
					Det	NP
						N

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

Keep going for each 3 words.

Last PP:

? → <with> <a fork>

? → <with a> <fork>

i.e.:

? → P NP (yes, that's PP)

? → ----- N (no)

She	eats	a	fish	with	a	fork
NP	S	-----				
	V, VP	-----	VP			
		Det	NP	-----		
			N	-----	-----	
				P	-----	PP
					Det	NP
						N

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

She	eats	a	fish	with	a	fork
NP	S	-----				
	V, VP	-----	VP			
		Det	NP	-----		
			N	-----	-----	
				P	-----	PP
					Det	NP
						N

Sequences of 4 words, but
 our grammar is binary. So
 check for rules for
 combining words in 2
 substrings
 ? → <she> <eats a fish>
 ? → <she eats> <a fish>
 ? → <she eats a> <fish>

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

? → <she> <eats a fish>

? → NP VP

She	eats	a	fish	with	a	fork
NP	S	-----				
	V, VP	-----	VP			
		Det	NP	-----		
			N	-----	-----	
				P	-----	PP
					Det	NP
						N

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

? → <she> <eats a fish>

? → NP VP

She	eats	a	fish	with	a	fork
NP	S	-----	S			
	V, VP	-----	VP			
		Det	NP	-----		
			N	-----	-----	
				P	-----	PP
					Det	NP
						N

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

She	eats	a	fish	with	a	fork
NP	S	-----	S			
	V, VP	-----	VP	-----		
		Det	NP	-----	-----	
			N	-----	-----	-----
				P	-----	PP
					Det	NP
						N

Keep going for each 4 words

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

She	eats	a	fish	with	a	fork
NP	S	-----	S	-----		
	V, VP	-----	VP	-----	-----	
		Det	NP	-----	-----	-----
			N	-----	-----	-----
				P	-----	PP
					Det	NP
						N

Keep going for each 5 words

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

Sequences of 6 words

? → <she> <eats a fish with a>

? → <she eats> <a fish with a>

? → <she eats a> <fish with a>

....

? → <eats a fish> <with a fork>

...

i.e. (last one above)

? → VP PP (yes, VP!)

She	eats	a	fish	with	a	fork
NP	S	-----	S	-----	-----	
	V, VP	-----	VP	-----	-----	VP
		Det	NP	-----	-----	-----
			N	-----	-----	-----
				P	-----	PP
					Det	NP
						N

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

Sequences of 7 words

? → <she> <eats a fish with a fork>

? → <she eats> <a fish with a fork>

? → <she eats a> <fish with a fork>

....

I.e. (for the first one above)

? → NP VP (yes, S!)

	She	eats	a	fish	with	a	fork
NP	S	-----	S	-----	-----		
	V, VP	-----	VP	-----	-----	VP	
		Det	NP	-----	-----	-----	
			N	-----	-----	-----	
				P	-----	PP	
					Det	NP	
						N	

The CKY algorithm – example

S → NP VP
 VP → VP PP
 VP → V NP
 VP → eats
 PP → P NP
 NP → Det N
 NP → she
 V → eats
 P → with
 N → fish
 N → fork
 Det → a

Sequences of 7 words

? → <she> <eats a fish with a fork>

? → <she eats> <a fish with a fork>

? → <she eats a> <fish with a fork>

....

I.e. (for the first one above)

? → NP VP (yes, S!)

She	eats	a	fish	with	a	fork
NP	S	-----	S	-----	-----	S
	V, VP	-----	VP	-----	-----	VP
		Det	NP	-----	-----	-----
			N	-----	-----	-----
				P	-----	PP
					Det	NP
						N

Questions?

Exercise

Given the
grammar
and lexicon
defining the
language

\mathcal{L}_1

\mathcal{L}_1 in CNF

$S \rightarrow NP VP$
 $S \rightarrow X1 VP$
 $X1 \rightarrow Aux NP$
 $S \rightarrow book \mid include \mid prefer$
 $S \rightarrow Verb NP$
 $S \rightarrow X2 PP$
 $S \rightarrow Verb PP$
 $S \rightarrow VP PP$
 $NP \rightarrow I \mid she \mid me$
 $NP \rightarrow TWA \mid Houston$
 $NP \rightarrow Det Nominal$
 $Nominal \rightarrow book \mid flight \mid meal \mid money$
 $Nominal \rightarrow Nominal Noun$
 $Nominal \rightarrow Nominal PP$
 $VP \rightarrow book \mid include \mid prefer$
 $VP \rightarrow Verb NP$
 $VP \rightarrow X2 PP$
 $X2 \rightarrow Verb NP$
 $VP \rightarrow Verb PP$
 $VP \rightarrow VP PP$
 $PP \rightarrow Preposition NP$

$Det \rightarrow that \mid this \mid the \mid a$
 $Noun \rightarrow book \mid flight \mid meal \mid money$
 $Verb \rightarrow book \mid include \mid prefer$
 $Pronoun \rightarrow I \mid she \mid me$
 $Proper-Noun \rightarrow Houston \mid NWA$
 $Aux \rightarrow does$
 $Preposition \rightarrow from \mid to \mid on \mid near \mid through$

Exercise

Parse (recognise) the following sentence

[illegible]

Exercise

Answer

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S,VP,X2 [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

CKY for parsing

- So far: CKY for **recognition**
 - Fill out matrix such that $[0,n]$ cell has symbol S
- To use CKY for **parsing**, need to
 - 1) Add **backpointers**: augment entries in matrix s.t. each non-terminal is paired with **pointers** to the matrix entries from which it was derived
 - 2) Allow multiple copies of the same non-terminal to be entered into the matrix to track multiple paths
- Choose S from cell $[0, n]$ and recursively retrieve its component constituents from the matrix

CKY for parsing

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb Nominal, Noun [0,1]		S,VP,X2 [0,3]		
	Det [1,2]	NP [1,3]		
		Nominal, Noun [2,3]		Nominal [2,5]
			Prep [3,4]	
				NP, Proper- Noun [4,5]



<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb, Nominal, Noun [0,1]		S, VP, X2 [0,3]		S ₁ , VP, X2 [0,5]
	Det [1,2]	NP [1,3]		S ₂ , VP [1,5]
		Nominal, Noun [2,3]		S ₃ [2,5]
			Prep [3,4]	NP [3,5]
				NP, Proper- Noun [4,5]

Statistical parsing

- This does **not scale**: too many possible parse trees for comprehensive grammar
- Solution:
 - Find the most likely parse(s) via statistical parsing
 - Comprehensive grammars admit many parses for a sentence, but we can efficiently find the most likely parse

Statistical parsing

Statistical parsing – learning

- “Learn” probabilistic grammars from labelled data: **treebanks**

```
( (S
  (NP-SBJ (DT The) (NN move))
  (VP (VBD followed)
    (NP
      (NP (DT a) (NN round))
      (PP (IN of)
        (NP
          (NP (JJ similar) (NNS increases))
          (PP (IN by)
            (NP (JJ other) (NNS lenders)))
          (PP (IN against)
            (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans))))))
    (, ,)
    (S-ADV
      (NP-SBJ (-NONE- *))
      (VP (VBG reflecting)
        (NP
          (NP (DT a) (VBG continuing) (NN decline))
          (PP-LOC (IN in)
            (NP (DT that) (NN market))))))
    (. .)))
```

E.g. Penn Treebank

50K sentences,

1M words

Statistical parsing – learning

- Treebanks are **expensive** to build, but:
 - Frequencies and distributional information are important
 - Can be reused to build different parsing approaches
 - Provide a way to evaluate parsers

Statistical parsing – formally

Probabilistic/stochastic phrase structure grammar is a context-free grammar **PCFG** = (T, N, S, R, q) , where:

- T is set of terminals
- N is set of nonterminals
- S is the start symbol (non-terminal)
- R is set of rules $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals & nonterminals
- $q = P(R)$ gives the probability of each rule

$$X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$$

Statistical parsing – intuition

Example by Chris Manning

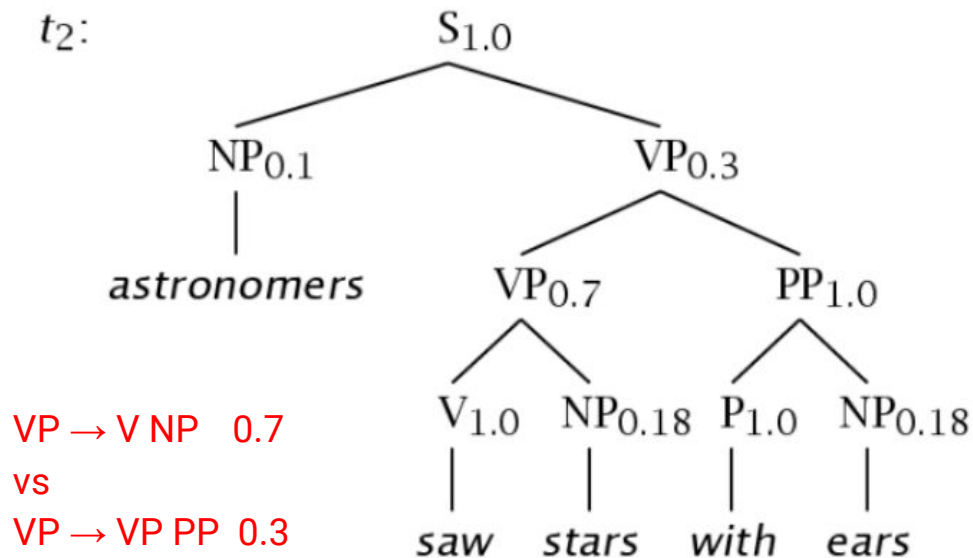
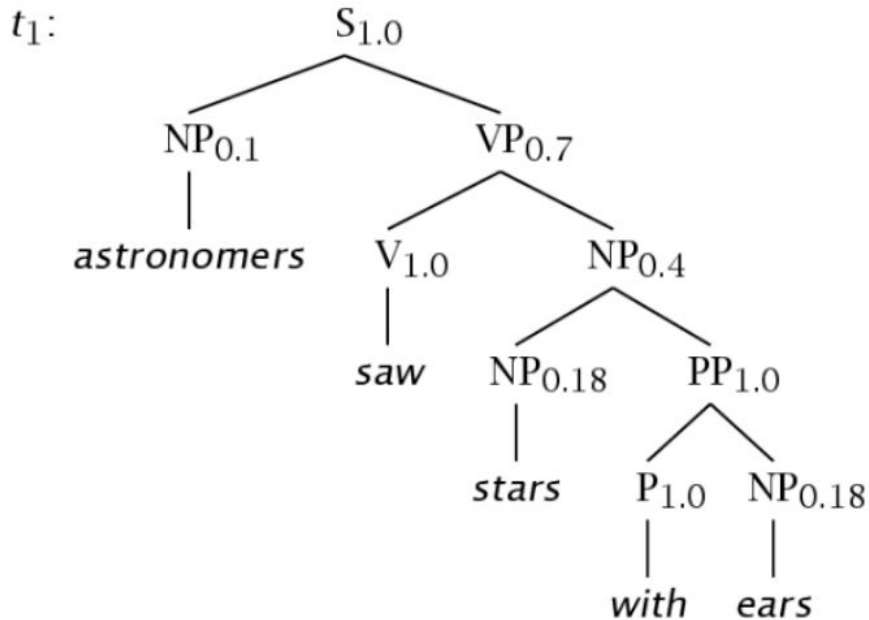
Given a probabilistic grammar and lexicon

$S \rightarrow NP VP$	1.0	$NP \rightarrow NP PP$	0.4
$VP \rightarrow V NP$	0.7	$NP \rightarrow \text{astronomers}$	0.1
$VP \rightarrow VP PP$	0.3	$NP \rightarrow \text{ears}$	0.18
$PP \rightarrow P NP$	1.0	$NP \rightarrow \text{saw}$	0.04
$P \rightarrow \text{with}$	1.0	$NP \rightarrow \text{stars}$	0.18
$V \rightarrow \text{saw}$	1.0	$NP \rightarrow \text{telescope}$	0.1

Statistical parsing – intuition

Example by Chris Manning

The following trees can be produced for the sentence
s = “astronomers saw stars with ears”



Statistical parsing – intuition

The probability of a given tree t is

$$P(t) = \prod_{i=1}^n q(X_i \rightarrow \gamma_i)$$

where $q(X \rightarrow \gamma)$ is the probability of rule $X \rightarrow \gamma$

Statistical parsing – intuition

Example by Chris Manning

s = “astronomers saw stars with ears”

- $P(t_1)$ = $1.0 * 0.1 * 0.7 * 1.0 * 0.4 * 0.18 * 1.0 * 1.0 * 0.18$
= 0.0009072
- $P(t_2)$ = $1.0 * 0.1 * 0.3 * 0.7 * 1.0 * 0.18 * 1.0 * 1.0 * 0.18$
= 0.0006804

So, $P(t_1)$ is more likely. Also, if want to find out probability of string s , sum $P(t_1) + P(t_2) = 0.0015876$ (works like a LM score)

Statistical parsing – intuition

Example by Chris Manning

s = “astronomers saw stars with ears”

- $P(t_1)$ $= 1.0 * 0.1 * 0.7 * 1.0 * 0.4 * 0.18 * 1.0 * 1.0 * 0.18$
 $= 0.0009072$
- $P(t_2)$ $= 1.0 * 0.1 * 0.3 * 0.7 * 1.0 * 0.18 * 1.0 * 1.0 * 0.18$
 $= 0.0006804$

The probability of each of the trees is obtained by multiplying the probabilities of each of the rules used in the derivation
This works as a disambiguation method!

Statistical parsing – intuition

Example by Chris Manning

s = “astronomers saw stars with ears”

- $P(t_1)$ $= 1.0 * 0.1 * 0.7 * 1.0 * 0.4 * 0.18 * 1.0 * 1.0 * 0.18$
 $= 0.0009072$
- $P(t_2)$ $= 1.0 * 0.1 * 0.3 * 0.7 * 1.0 * 0.18 * 1.0 * 1.0 * 0.18$
 $= 0.0006804$

The probability of each of the trees is obtained by multiplying the probabilities of each of the rules used in the derivation

This works as a disambiguation method!

This brute-force approach (enumerating all options) does not scale...

Questions?

Statistical parsing – more formally

- Given all the possible parse trees \mathcal{T} for a given sentence s . The **string of words** s is called the **yield** of any parse tree over s
- **Out of all parse trees that yield s** , the algorithm picks the parse tree t that is most probable given s ($t \in \tau(s)$):

$$\hat{t}(s) = \operatorname{argmax}_{t \in \tau(s)} P(t|s)$$

Statistical parsing - more formally

$$P(t, s) = \prod_{i=1}^n q(X_i \rightarrow \gamma_i)$$

- By definition $P(t, s) = P(t)P(s|t)$
- But $P(s|t) = 1$, since a parse tree includes all words in sentence, i.e.

$$P(t, s) = P(t)P(s|t) = P(t)$$

- Thus

$$\hat{t}(s) = \operatorname{argmax}_{t \in \tau(s)} P(t)$$

The CKY algorithm for PCFG

- Same dynamic programming algorithm, but now to find most likely parse tree $\hat{t}(s) = \operatorname{argmax}_{t \in \tau(s)} P(t)$
- Useful for
 - **Recognition**: does this sentence belong to the language?
 - **Parsing**: give me a possible derivation
 - **Disambiguation**: give me the best derivation
- All in polynomial time

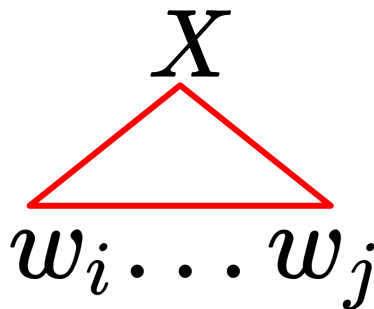
The CKY algorithm for PCFG

- Grammar needs to be in CNF, as before
 - Rules are of the form $X \rightarrow Y Z$ or $X \rightarrow w$
 - Modify probabilities s.t. the probability of each parse remains the same under the new CNF grammar. E.g.:
$$\begin{array}{lcl} \text{VP} \rightarrow \text{Vt NP PP} & 0.2 & \longrightarrow \text{VP} \rightarrow \text{New PP} \quad 0.2 \\ & & \text{New} \rightarrow \text{Vt NP} \quad 1.0 \end{array}$$
- Build matrix as with CKY before
- Other **unary rules** can be included, e.g. $\text{NP} \rightarrow \text{N}$, as long as they don't lead to loops

The CKY algorithm for PCFG

This is a hierarchical process:

- n is the number of words in sentence
- $w_{1 \dots n} = w_1 \dots w_n$ = the word sequence from 1 to n
- $w_{i \dots j}$ = the subsequence $w_i \dots w_j$
- $X_{i \dots j}$ = the nonterminal X dominating $w_i \dots w_j$



The CKY algorithm for PCFG

- Define the dynamic table

$\pi[i, j, X]$ = **maximum** probability of a constituent with non-terminal X spanning words $i \dots j$ inclusive

- With the goal to calculate

$$\max_{t \in \tau(s)} p(t) = \pi[1, n, S]$$

The CKY algorithm for PCFG

- Define the dynamic table

$\pi[i, j, X]$ = maximum probability of a constituent with non-terminal X spanning words $i \dots j$ inclusive

$$i = 1 \dots n \quad j = 1 \dots n \quad X \in N$$

$i \leq j$

- With the goal to calculate

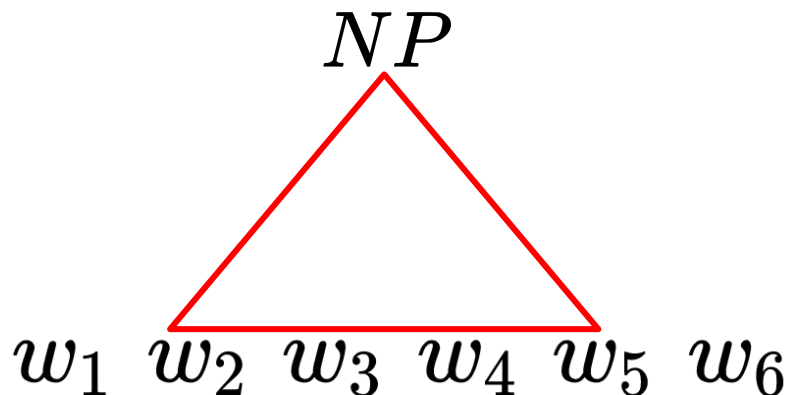
$$\max_{t \in \tau(s)} p(t) = \pi[1, n, S]$$

Highest scoring parse tree for entire sentence

The CKY algorithm for PCFG

- For example, say sentence is $w_1 w_2 w_3 w_4 w_5 w_6$
- And we take $\pi[2, 5, NP]$

↪ The parse tree with the highest probability for words 2-5, whose head is NP



- There may be multiple possible ways in which **NP** has a parse tree under it that spans **words 2-5**
- Every of those parse trees will have a probability (product of the rule probabilities in that parse tree)

The CKY algorithm for PCFG

- **Base case:** for all $i = 1 \dots n, X \in N$

$$\pi[i, i, X] = q(X \rightarrow w_i)$$

where $q(X \rightarrow w_i) = 0$ if $X \rightarrow w_i$ is not in grammar

- **Recursive case:** for all $i = 1 \dots n - 1, j = (i + 1) \dots n, X \in N$

$$\pi[i, j, X] = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi[i, s, Y] \times \pi[s + 1, j, Z])$$

The CKY algorithm for PCFG


- Base case:** for all $i = 1 \dots n, X \in N$

Terminals / unary rules

$$\pi[i, i, X] = q(X \rightarrow w_i)$$

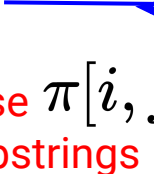
where $q(X \rightarrow w_i) = 0$ if $X \rightarrow w_i$ is not in grammar
- Recursive case:** for all $i = 1 \dots n - 1, j = (i + 1) \dots n, X \in N$

$$\pi[i, j, X] = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi[i, s, Y] \times \pi[s+1, j, Z])$$



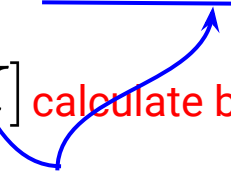
Split point

Recursive because $\pi[i, j, X]$ calculate based on the on π for substrings



i to s

and



s+1 to j

The CKY algorithm for PCFG – example

$$\pi[i, j, X] = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi[i, s, Y] \times \pi[s+1, j, Z])$$

the dog **saw the man with the telescope**

1 2 3 4 5 6 7 8

$\pi[3, 8, VP]$

$q(VP \rightarrow Vt NP) \times \pi[3, 3, Vt] \times \pi[4, 8, NP]$

$q(VP \rightarrow Vt NP) \times \pi[3, 4, Vt] \times \pi[5, 8, NP]$

....

$q(VP \rightarrow Vt NP) \times \pi[3, 7, Vt] \times \pi[8, 8, NP]$

$q(VP \rightarrow VP PP) \times \pi[3, 3, VP] \times \pi[4, 8, PP]$

$q(VP \rightarrow VP PP) \times \pi[3, 7, VP] \times \pi[8, 8, PP]$

$VP \rightarrow Vt NP$ 0.4

$VP \rightarrow VP PP$ 0.6

$s \in \{3, 4, 5, 6, 7\}$

The CKY algorithm for PCFG

Input: a sentence $s = x_1 \dots x_n$, a PCFG $G = (N, \Sigma, S, R, q)$.

Pseudocode by Michael Collins

Initialization:

For all $i \in \{1 \dots n\}$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Algorithm:

- ▶ For $l = 1 \dots (n - 1)$
 - ▶ For $i = 1 \dots (n - l)$
 - ▶ Set $j = i + l$
 - ▶ For all $X \in N$, calculate

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

and

$$bp(i, j, X) = \arg \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

The CKY algorithm for PCFG – exercise

Example by Michael Collins

Given the grammar:

S	⇒	NP	VP	1.0
VP	⇒	Vi		0.4
VP	⇒	Vt	NP	0.4
VP	⇒	VP	PP	0.2
NP	⇒	DT	NN	0.3
NP	⇒	NP	PP	0.7
PP	⇒	IN	NP	1.0

Vi	⇒	sleeps	1.0
Vt	⇒	saw	1.0
NN	⇒	man	0.7
NN	⇒	woman	0.2
NN	⇒	telescope	0.1
DT	⇒	the	1.0
IN	⇒	with	0.5
IN	⇒	in	0.5

Generate the best parse tree for the sentence:

The woman saw the man with the telescope

Evaluating parsers

- **Parseval** metrics: evaluate structure
 - How much of constituents in the **hypothesis parse tree** look like the constituents in a **gold-reference parse tree**
 - A constituent in hyp parse is labelled “correct” if there is a constituent in the ref parse with the same yield and LHS symbol
 - Only rules **from non-terminal to non-terminal**
 - Metrics are more **fine-grained** than full tree metrics, more robust to localised differences in hyp and ref parse trees

Evaluating parsers

$$\text{Precision} = \frac{\text{\# of correct constituents in hyp parse of } s}{\text{\# of total constituents in hyp parse of } s}$$

$$\text{Recall} = \frac{\text{\# of correct constituents in hyp parse of } s}{\text{\# of total constituents in ref parse of } s}$$

$$\text{F-measure} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Evaluating parsers – example

Example by Philipp Koehn

	X:a		X: a	
	Y:b	←	Z: b	
Hyp	Z: cd		V: cd	Ref
	W: abcd		Y: bcd	
			W: abcd	

These are flattened constituents.

Discarding LHS symbol:

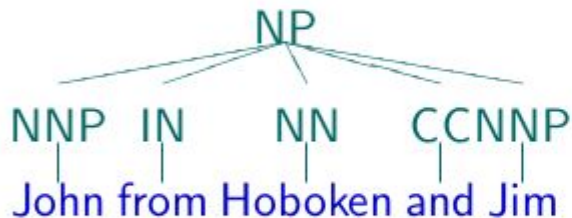
Precision = 4/4

Recall = $\frac{4}{5}$

F-measure = ...

Evaluating parsers - example

Hyp



NP_1 : John from Hoboken and Jim

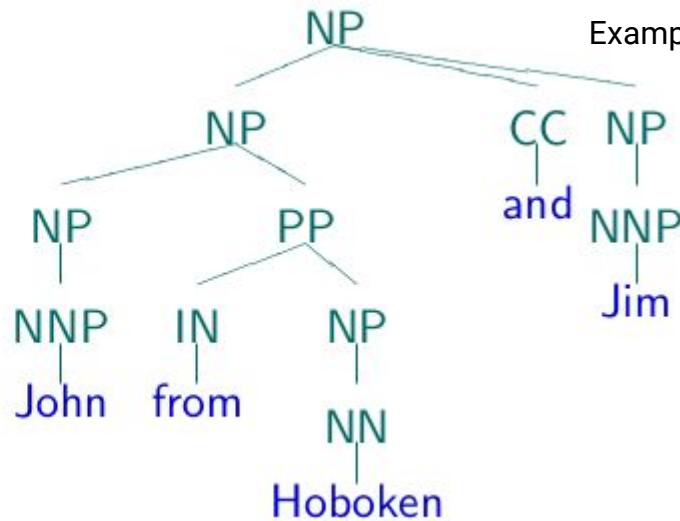
Not counting terminals or unary rules, but we can

Including LHS symbol:

Precision = 1/1 (NP_1 span)

Recall = 1/6

Example by Philipp Koehn



Ref

NP_1 : John from Hoboken and Jim

NP_2 : John

NP_3 : John from Hoboken

PP_1 : from Hoboken

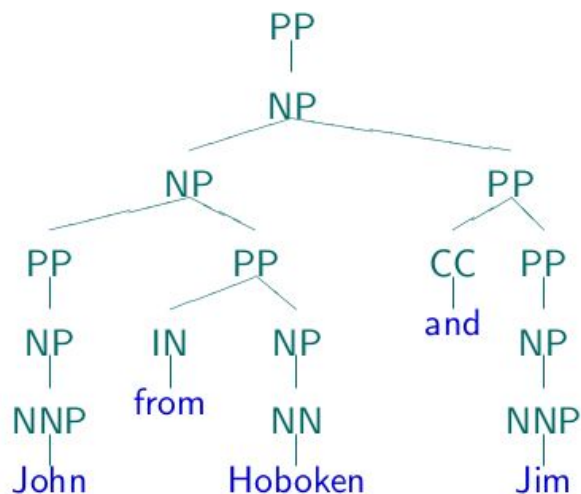
NP_4 : Hoboken

NP_5 : Jim

Evaluating parsers – exercise

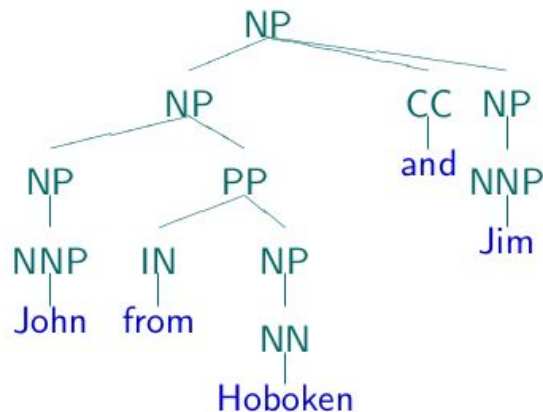
Example by Philipp Koehn

Hyp



PP: John from Hoboken and Jim
NP: John from Hoboken and Jim
NP: John from Hoboken
PP: John NP: John
PP: from Hoboken NP: Hoboken
PP: and Jim PP: Jim NP: Jim

Ref



NP: John from Hoboken and Jim
NP: John
NP: John from Hoboken
PP: from Hoboken
NP: Hoboken NP: Jim

Issues with PCFG

- **Poor independence assumption:** CFG rules impose an independence assumption on probabilities that leads to poor modelling of structural dependencies across the parse tree. Word is only dependent on its POS tag!
- **Lack of lexical conditioning:** CFG rules don't model syntactic facts about specific words, leading to problems with subcategorization ambiguities, preposition attachment, and coordinate structure ambiguities

Issues with PCFG

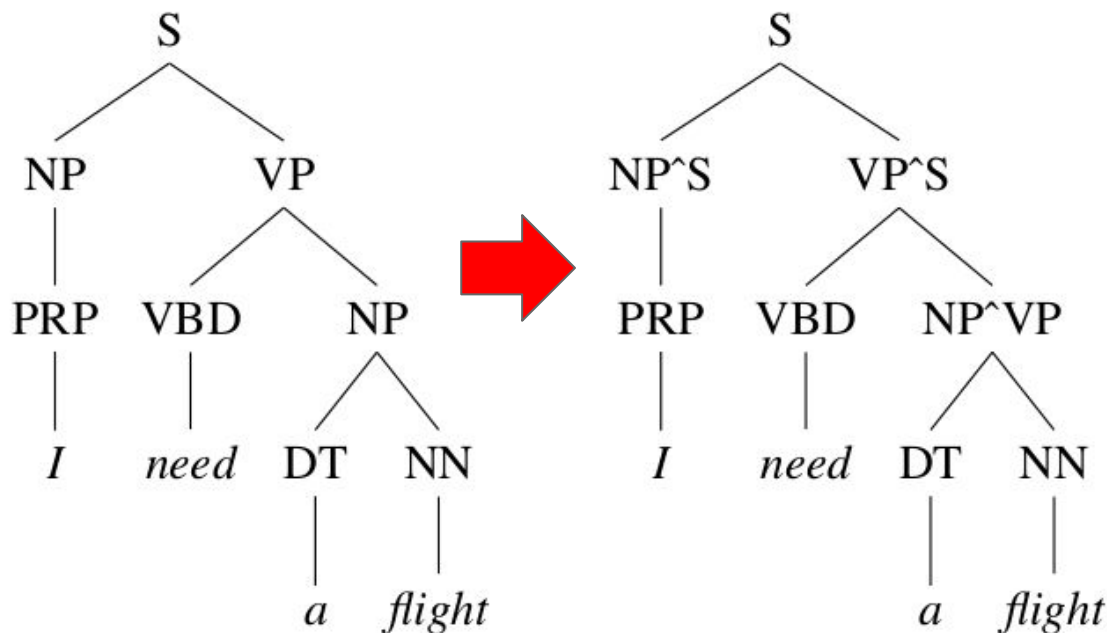
- **Poor independence assumption:**
 - Probability estimates of rules computed independently of surrounding context, e.g.:
 - In English, NPs in the **subject** function are more likely to be derived as pronominal (91%), while NPs in **object** function are more likely to be derived as non-pronominal (66%)
 - However, given overall probabilities
 - $NP \rightarrow DET\ N$ 0.28
 - $NP \rightarrow PRN$ 0.25
 - There is no way we can capture this prior information

Extensions PCFG

- **Poor independence assumption - Split non-terminals:**
 - Condition probabilities of $NP \rightarrow DET\ N$ and $NP \rightarrow PRP$ on whether the NP is subject or object \rightarrow parent annotation
$$NP_{\text{subject}} \rightarrow PRN \quad 0.91 \quad (\text{PRN: Personal Pronoun})$$
$$NP_{\text{object}} \rightarrow PRN \quad 0.34$$
 - To do so, annotate each node with its parent
 - In this case, $S \rightarrow NP^S\ VP^S$ could indicate that the first NP is the subject

Extensions PCFG

- **Poor independence assumption - Split non-terminals:**



Issues with PCFG

- **Lack of lexical conditioning:**

- Lexical information to resolve PP attachment ambiguity, e.g.:

Staff dumped masks **into** a bin (VP)

The boys caught kilos **of** fish (NP)

- VP attachment or NP attachment? NP attachment is more frequent in English, so often preferred by parsers
- The **affinity** between ‘dump’ and ‘into’ is greater than the affinity between ‘masks’ and ‘into’. Conversely, the affinity btw. ‘kilos’ and ‘of’ is greater than btw. ‘catch’ and ‘of’

Issues with PCFG

- **Lack of lexical conditioning:**
 - Lexical information to resolve coordination ambiguity, e.g.:
Dogs in houses **and** cats
 - [dogs in [NP houses and cats]] ???
 - The **affinity** between 'dogs' and 'cats' is greater than the affinity between 'houses' and 'cats'

Extensions PCFG

- **Lack of lexical conditioning - Probabilistic Lexicalised CFGs:**
 - Add annotations specifying the **head** of each rule
 - Each rule in the grammar identifies one of its children to be the head of the rule

S	⇒	NP	VP
VP	⇒	Vi	
VP	⇒	Vt	NP
VP	⇒	VP	PP
NP	⇒	DT	NN
NP	⇒	NP	PP
PP	⇒	IN	NP

Vi	⇒	sleeps
Vt	⇒	saw
NN	⇒	man
NN	⇒	woman
NN	⇒	telescope
DT	⇒	the
IN	⇒	with
IN	⇒	in

Probabilistic Lexicalised CFG

Grammar **PCFG** = **(T, N, S, R, q)**, where:

- T is set of terminals
- N is set of nonterminals
- S is the start symbol (non-terminal)
- R is set of rules in CNF which take 3 forms
 - $X(h) \rightarrow Y(h) Z(w)$
 - $X(h) \rightarrow Y(w) Z(h)$
 - $X(h) \rightarrow h$

Where X, Y, Z are in N , and h, w are in T

- $q = P(R)$ gives the probability of each rule

Probabilistic Lexicalised CFG

Example by Michael Collins

LPCFG grammar example:

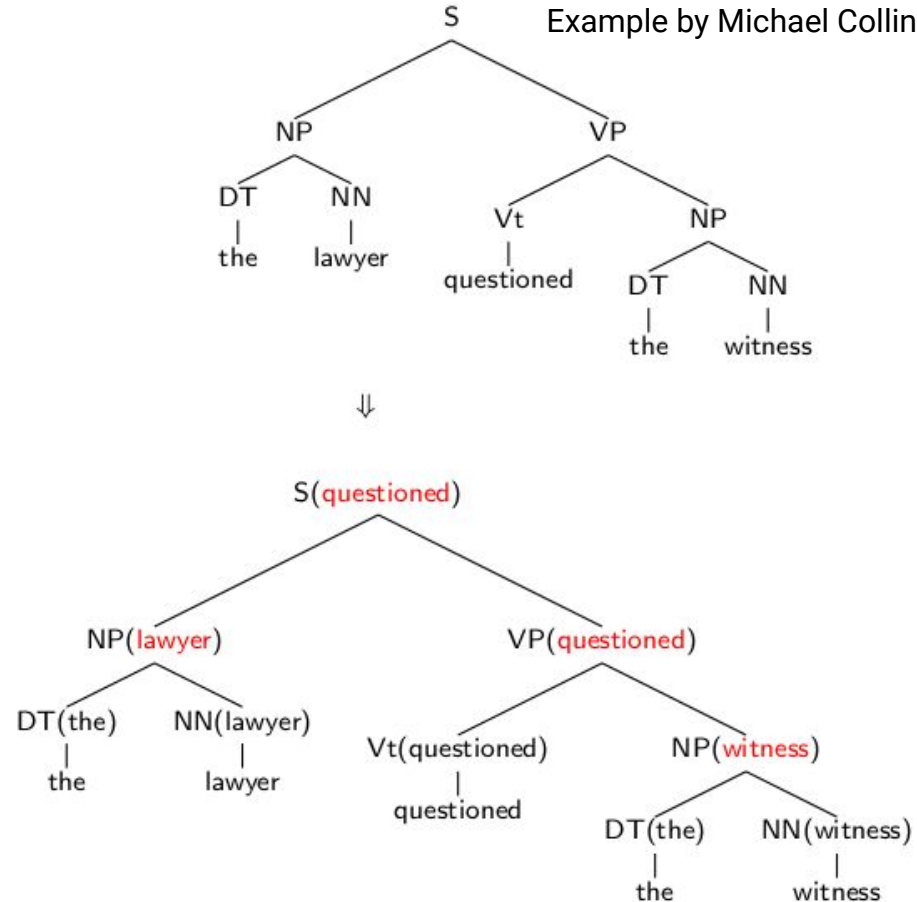
S(saw)	\rightarrow_2	NP(man)	VP(saw)
VP(saw)	\rightarrow_1	Vt(saw)	NP(dog)
NP(man)	\rightarrow_2	DT(the)	NN(man)
NP(dog)	\rightarrow_2	DT(the)	NN(dog)
Vt(saw)	\rightarrow	saw	
DT(the)	\rightarrow	the	
NN(man)	\rightarrow	man	
NN(dog)	\rightarrow	dog	

Probabilistic Lexicalised CFG

- A constituent receives its headword from its headchild

$S \rightarrow NP \text{ VP}$ (S receives from VP)
 $VP \rightarrow \text{Vt} \text{ VP}$ (VP receives from Vt)
 $NP \rightarrow DT \text{ NN}$ (NP receives from NN)

Example by Michael Collins



Probabilistic Lexicalised CFG

- **Head** is core linguistic concept, the central sub-constituent of each rule
- Treebanks not annotated for that, but can use rules to identify head

Probabilistic Lexicalised CFG

If the rule contains NN, NNS, or NNP:

Rule 1

Example by Michael Collins

Choose the rightmost NN, NNS, or NNP

Else If the rule contains an NP: Choose the leftmost NP Rule 2

Else If the rule contains a JJ: Choose the rightmost JJ Rule 3

Else If the rule contains a CD: Choose the rightmost CD Rule 4

Else Choose the rightmost child Rule 5

e.g.,

NP	⇒	DT	NNP	NN	Rule 1
NP	⇒	DT	NN	NNP	Rule 1
NP	⇒	NP	PP		Rule 2
NP	⇒	DT	JJ		Rule 3
NP	⇒	DT			Rule 5

Probabilistic Lexicalised CFG

Example by Michael Collins

If the rule contains Vi or Vt: Choose the leftmost Vi or Vt Rule 6

Else If the rule contains an VP: Choose the leftmost VP Rule 7

Else Choose the leftmost child

e.g.,

VP	⇒	Vt	NP	Rule 6
VP	⇒	VP	PP	Rule 7

Probabilistic Lexicalised CFG

Estimating the probabilities from treebank is similar:

- PCFG:

$$q(S \rightarrow NP VP)$$

- LPCFG

$$q(S(\text{saw}) \rightarrow NP(\text{man}) VP(\text{saw}))$$

Accuracy of PCFG on Penn Treebank = 72% → 88%