Natural Language Processing

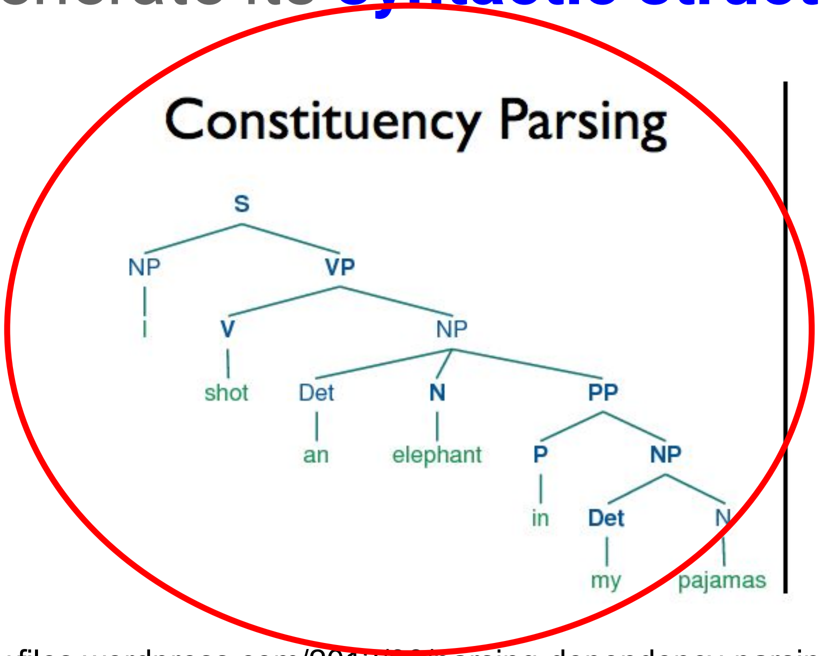
Nuri Cingillioglu
https://www.doc.ic.ac.uk/~nuric/

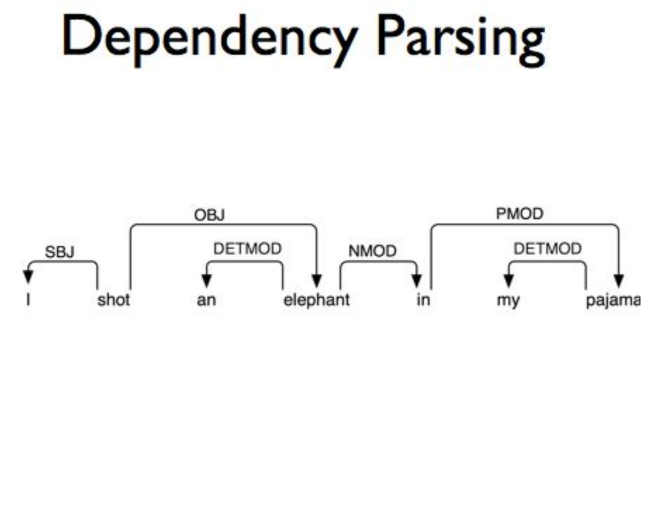
Many thanks to Lucia Specia

Constituency parsing

Definition

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





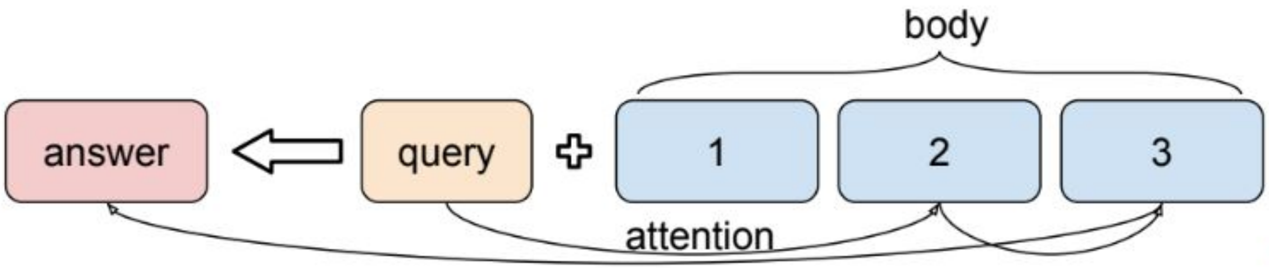
https://lee6boy.files.wordpress.com/2013/06/parsing-dependency-parsing-graph-based-parsingec9db4-ebad94eab080-002.png

Applications

- Grammar checking
- Semantic analysis
  - Question answering
  - Named entity recognition
- Parsing (e.g. chatbot commands like turn on the living room lights)
- As features for downstream task

Neuro-symbolic reasoning

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



1. John is in the playground.
2. John picked up the football.
3. Bob went to the kitchen.

q. Where is the football?

a. playground

Cingillioglu et. al (2019)

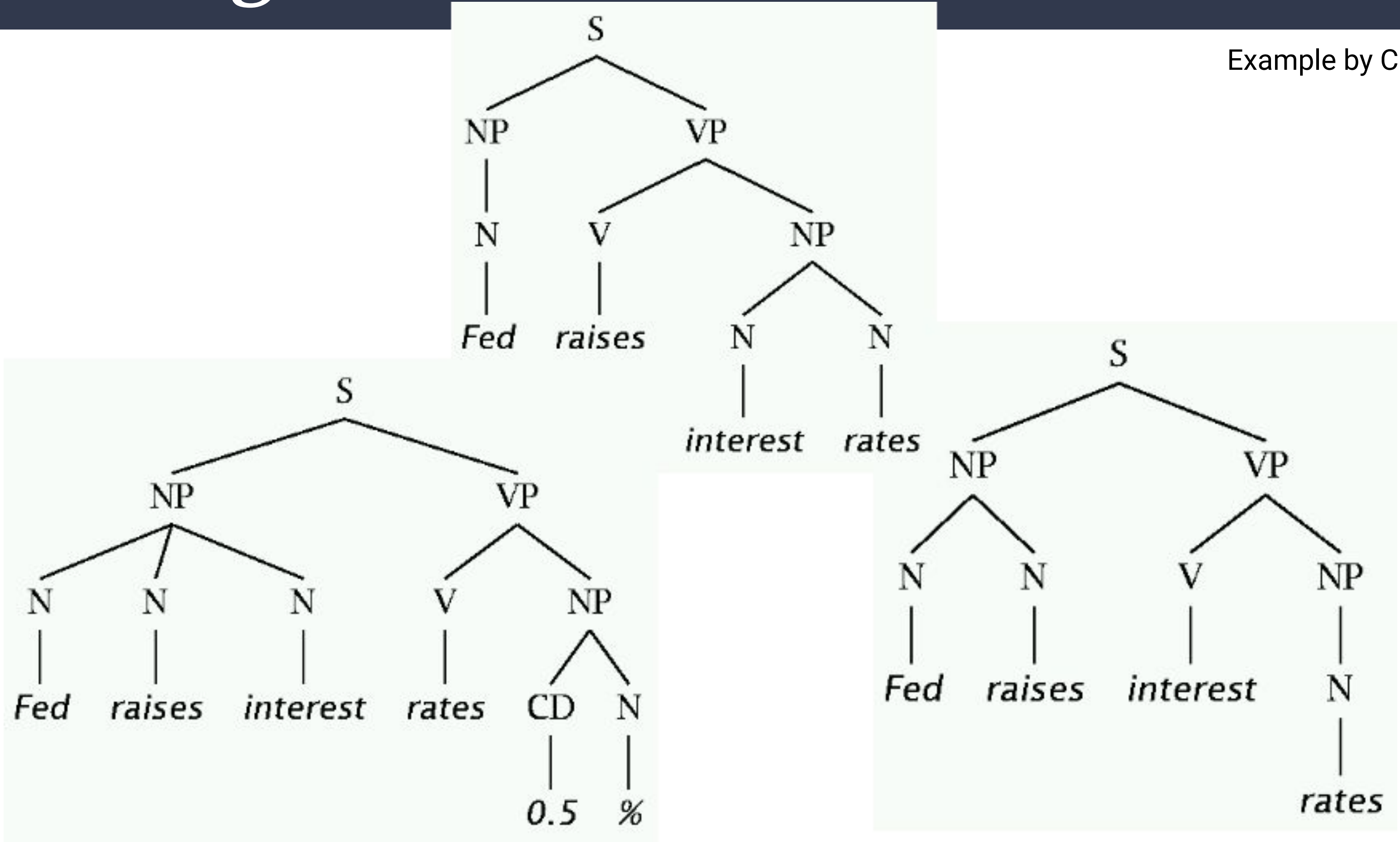
Challenges

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

Figure from Jurafsky, D and Martin, J, “Speech and Language Processing,” 2018, ch 13

Classical parsing

- … and a new sentence, e.g.

The flight includes a meal

- Goal: Generate the structure for the sentence

Classical parsing

ChatGPT

Sure, here's a constituency parse tree for the sentence "The flight includes a meal":

css   Copy code

```css
         [S]
          |
    .------'------.
  [NP]          [VP]
   |              |
 [Det]         .--'--.
   |          [V]   [NP]
  The        / |      |
            /  |  .--'---.
           /   | [Det]  [N]
          /    |   |      |
         /     |  The    meal
        /      |
    .---'      |
   [N]        [V]
    |          |
  flight    includes
```

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

$VP \rightarrow V\ NP\ PP \Rightarrow VP \rightarrow V\ NEW$, $NEW \rightarrow NP\ PP$

$INF\text{-}VP \rightarrow to\ VP \Rightarrow INF\text{-}VP \rightarrow TO\ VP$, $TO \rightarrow to$

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

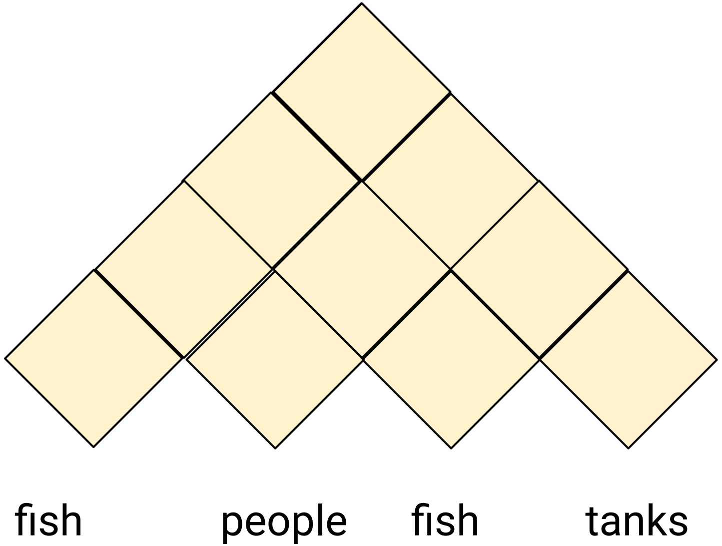
- **Dynamic programming** algorithm to efficiently generate all possible parse trees bottom-up

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)

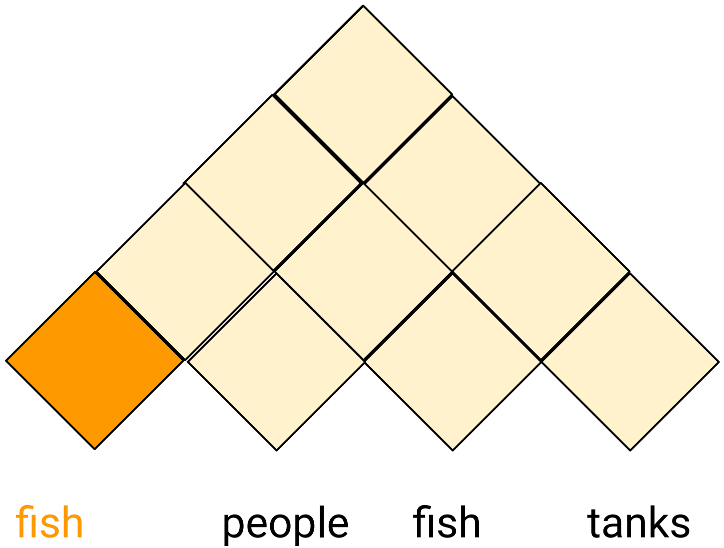
The CKY algorithm – intuition

Example by Chris Manning



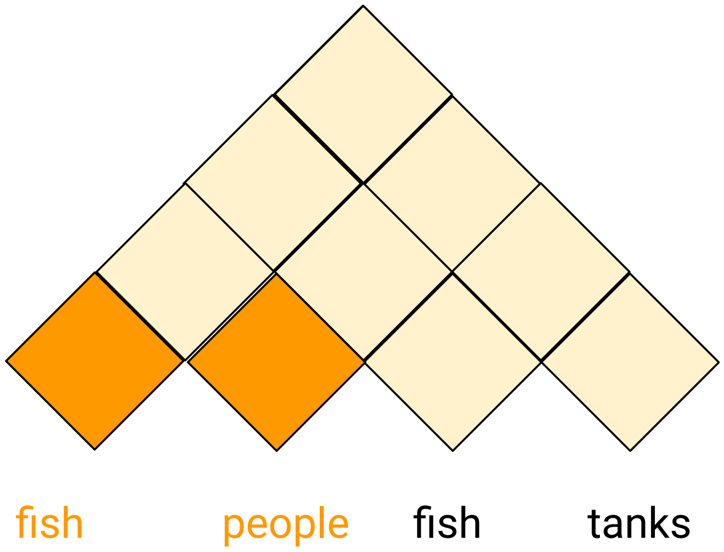
The CKY algorithm – intuition

Example by Chris Manning



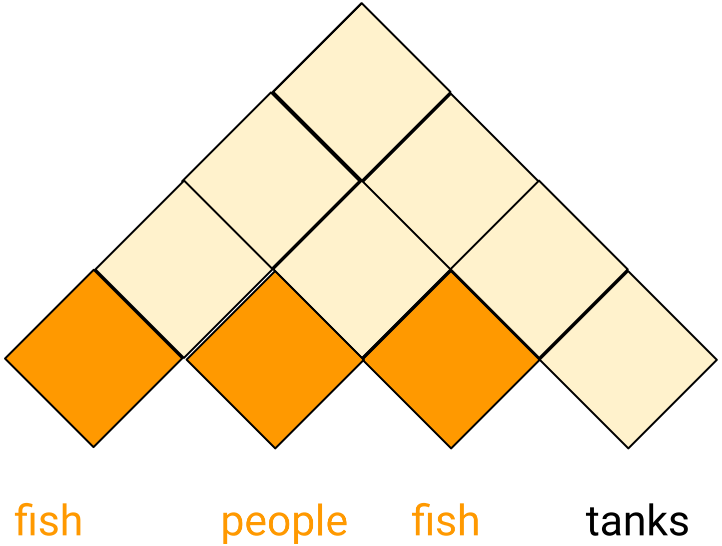
The CKY algorithm – intuition

Example by Chris Manning



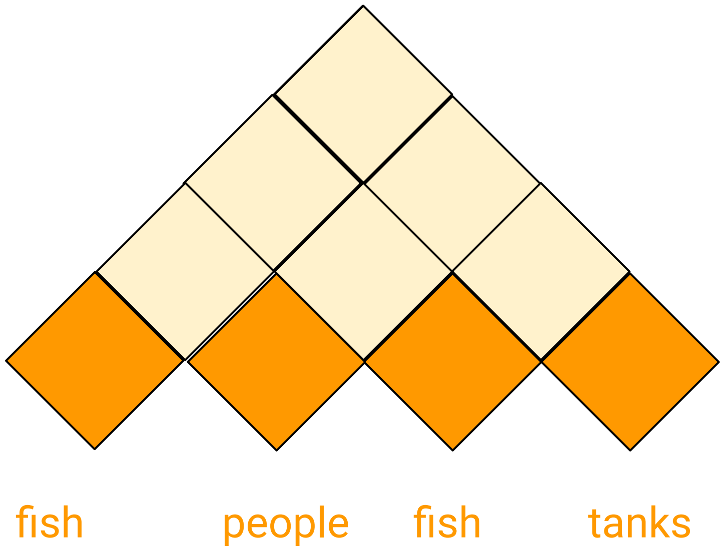
The CKY algorithm – intuition

Example by Chris Manning



The CKY algorithm – intuition

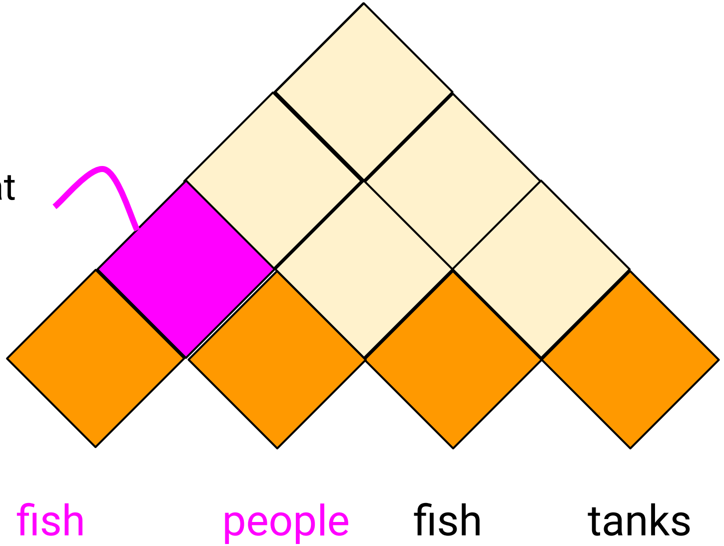
Example by Chris Manning



The CKY algorithm – intuition

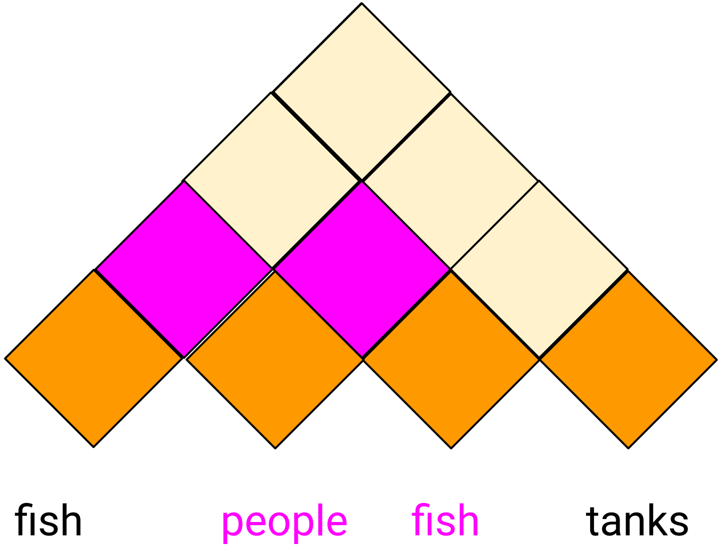
Example by Chris Manning

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



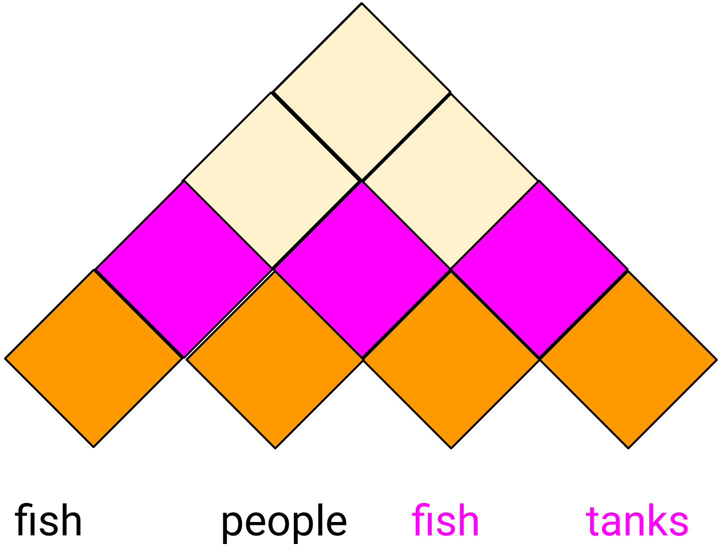
The CKY algorithm – intuition

Example by Chris Manning



The CKY algorithm – intuition

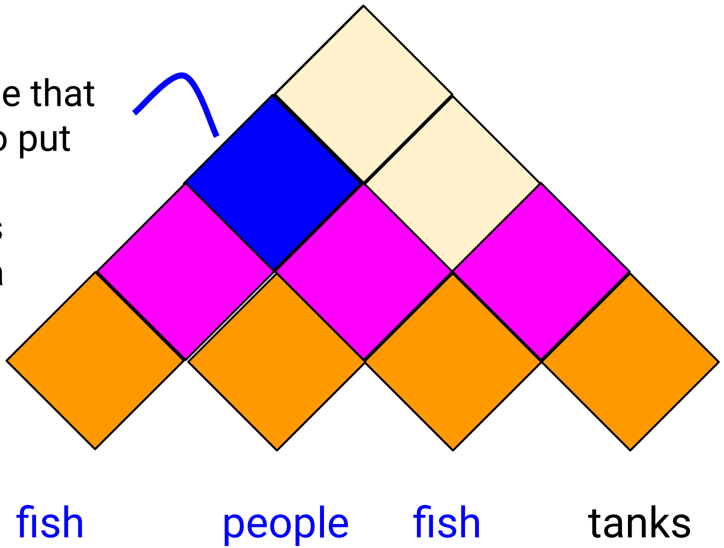
Example by Chris Manning



The CKY algorithm – intuition

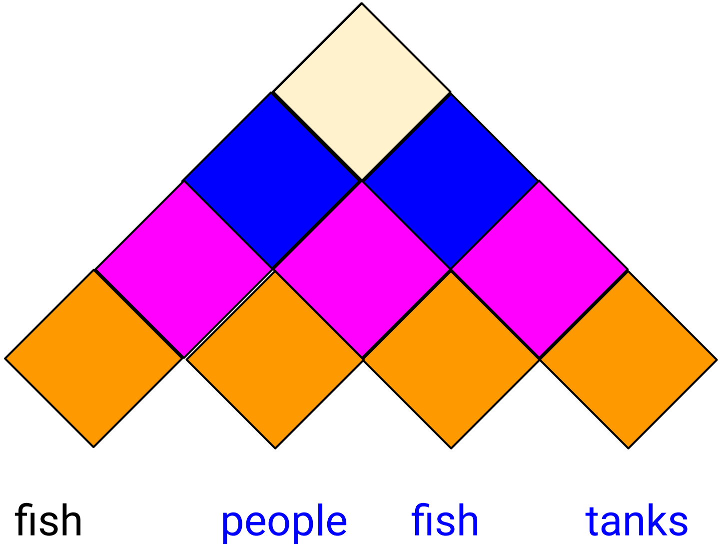
Example by Chris Manning

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



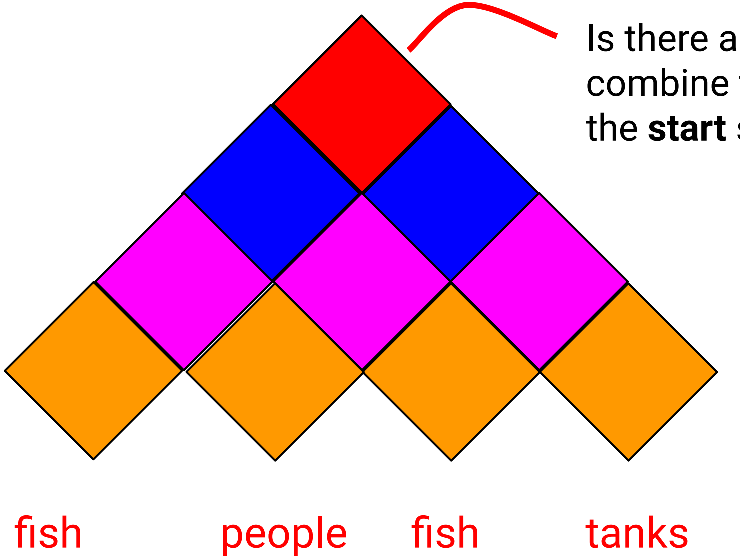
The CKY algorithm – intuition

Example by Chris Manning



The CKY algorithm – intuition

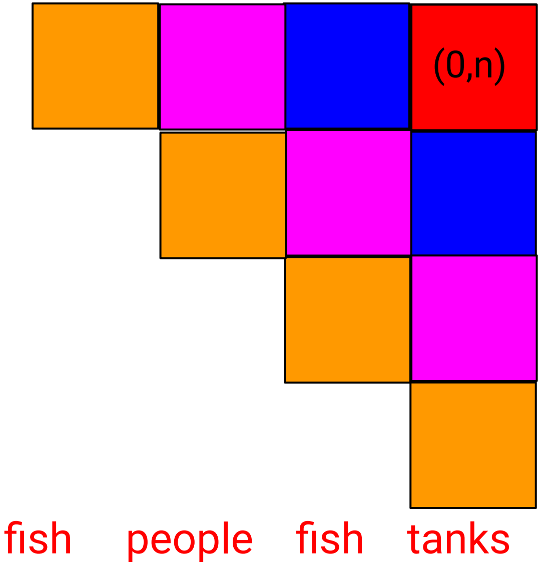
Example by Chris Manning



Is there a rule which allows me to combine the whole sequence from the **start** symbol?

The CKY algorithm – intuition

Example by Chris Manning



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

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
? → NP V
or
? → 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
? → NP V
or
? → 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
? → 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
? → 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

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

? → NP -------

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

? → S 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	NP			
			N	--------		
				P	--------	
					Det	NP
						N

? → S 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	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

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

? → -------- 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

Keep going for each 3 words.
Last PP:
? → <with> <a fork>
? → <with a> <fork>
i.e.:
? → P NP (yes, that's PP)
? → ------- N (no)

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

? → <she> <eats a fish>

? → 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	--------	S			
	V, VP	--------	VP			
		Det	NP	--------		
			N	--------	--------	
				P	--------	PP
					Det	NP
						N

? → <she> <eats a fish>

? → 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	--------	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

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

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!)

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	--------	--------	VP
		Det	NP	--------	--------	--------
			N	--------	--------	--------
				P	--------	PP
					Det	NP
						N

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!)

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	--------	--------	S
	V, VP	--------	VP	--------	--------	VP
		Det	NP	--------	--------	--------
			N	--------	--------	--------
				P	--------	PP
					Det	NP
						N

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!)

Questions?

Exercise

Given the grammar and lexicon defining the language $\mathscr{L}_1$

$\mathscr{L}_1$ in CNF
S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → I | she | me
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

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

From Jurafsky, D and Martin, J, “Speech and Language Processing,” 2018, ch 13

Exercise

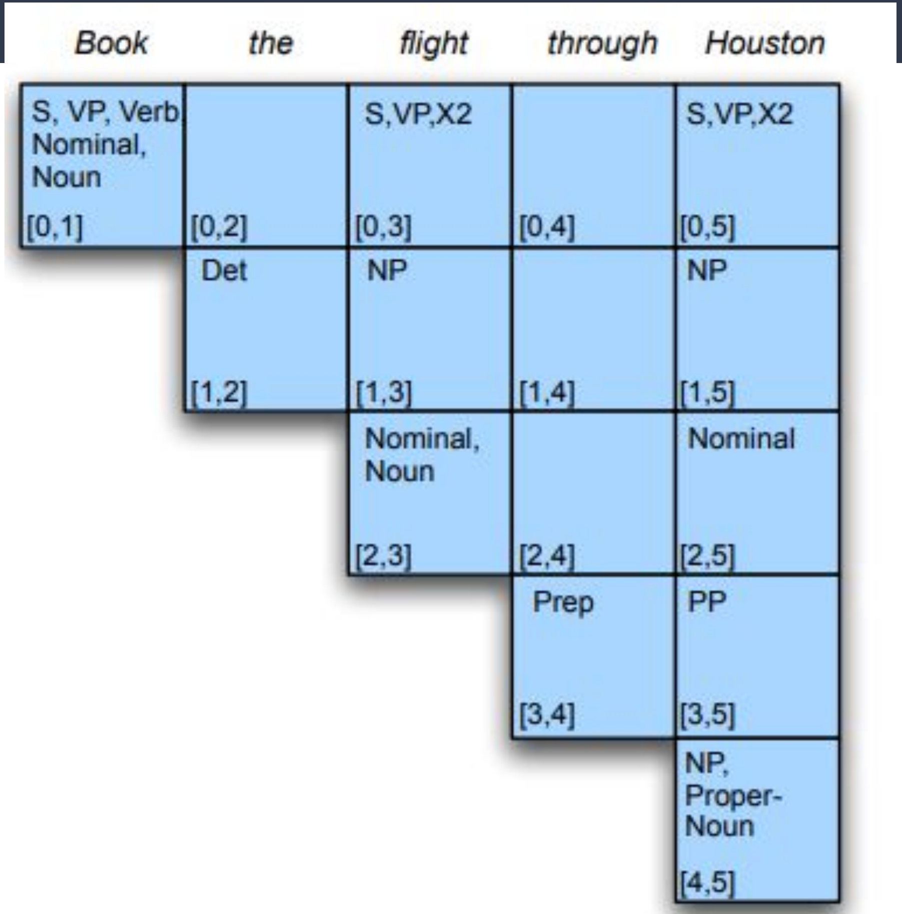
Parse (recognise) the following sentence

Book	the	flight	through	Houston

From Jurafsky, D and Martin, J, “Speech and Language Processing,” 2018, ch 13

Exercise

Answer



Book	the	flight	through	Houston
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]

From Jurafsky, D and Martin, J, “Speech and Language Processing,” 2018, ch 13

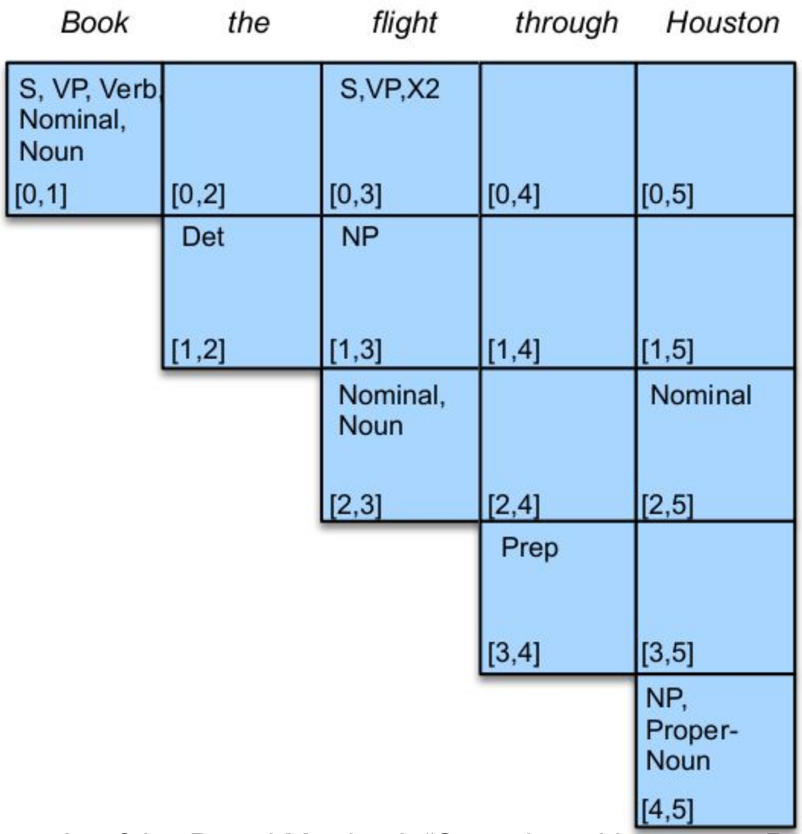
CKY for parsing

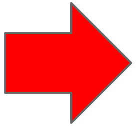
- So far: CKY for recognition
  - Fill out matrix such that [0,n] cell has symbol S
- To use CKY for parsing, need to

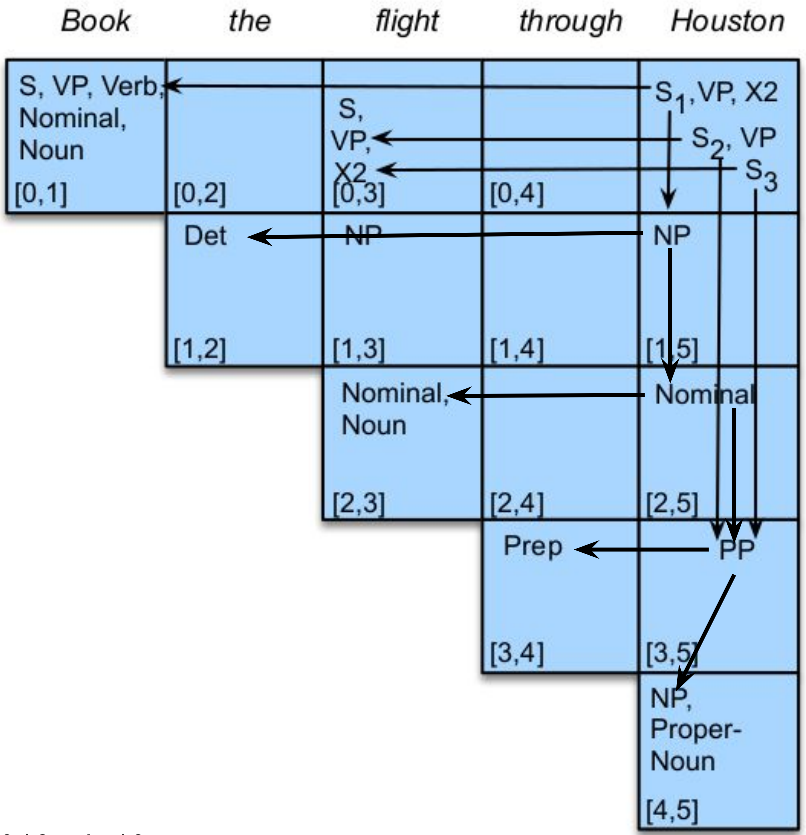
  1) Add **backpointers**: augment entries in matrix s.t. each non-terminal is paired with **pointers** to the matrix entries from which it was derived

  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







From Jurafsky, D and Martin, J, “Speech and Language Processing,” 2018, ch 13

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

https://catalog.ldc.upenn.edu/docs/LDC95T7/cl93.html

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 – learning

- What does it mean to “learn” a grammar?
  - Take annotated trees (e.g. Penn Treebank)
  - List all rules used, e.g. for NP:
    - NP $\rightarrow$ NP PP
    - NP $\rightarrow$ NP PP-LOC
    - NP $\rightarrow$ DT NN
    - NP $\rightarrow$ JJ NNS
    - NP $\rightarrow$ NNP JJ NN NNS
    - NP $\rightarrow$ DT VBG NN
    - NP $\rightarrow$ NP PP PP

```
( (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)))))))
    (. .)))
```

Statistical parsing – learning

- What does it mean to “learn” a grammar?
  - Assign probabilities to all rules by counting (c)

$$q(X \rightarrow \gamma) = \frac{\mathrm{c}(X \rightarrow \gamma)}{\mathrm{c}(X)}$$

E.g:

- NP → NP PP   q = 1/9
- NP → NP PP-LOC   q = 1/9
- NP → DT NN   q = 2/9
- NP → JJ NNS   q = 2/9
- NP → NNP JJ NN NNS   q = 1/9
- NP → DT VBG NN   q = 1/9
- NP → NP PP PP   q = 1/9

```
( (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)))))))
    (. .)))
```

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

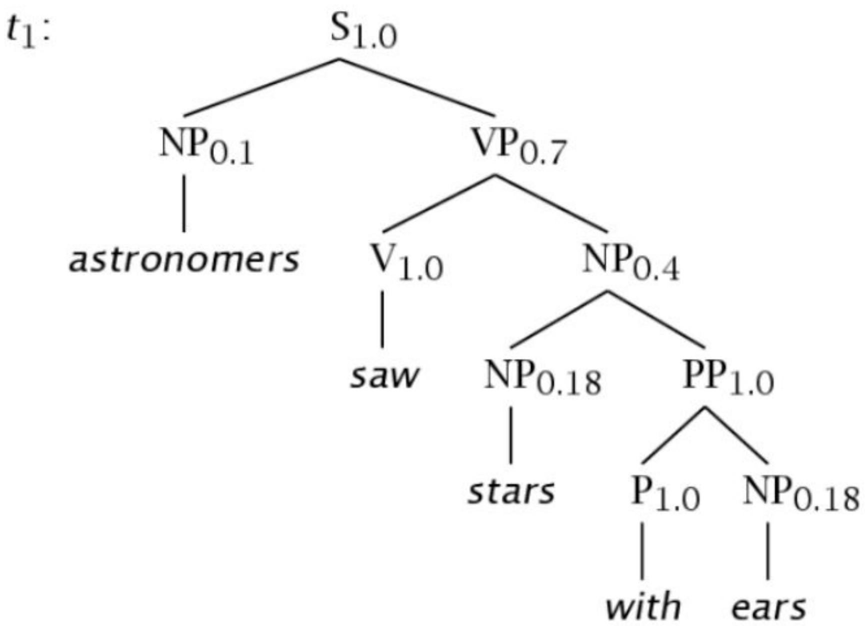
S → NP VP	1.0	NP → NP PP	0.4
VP → V NP	0.7	NP → astronomers	0.1
VP → VP PP	0.3	NP → ears	0.18
PP → P NP	1.0	NP → saw	0.04
P → with	1.0	NP → stars	0.18
V → saw	1.0	NP → telescope	0.1

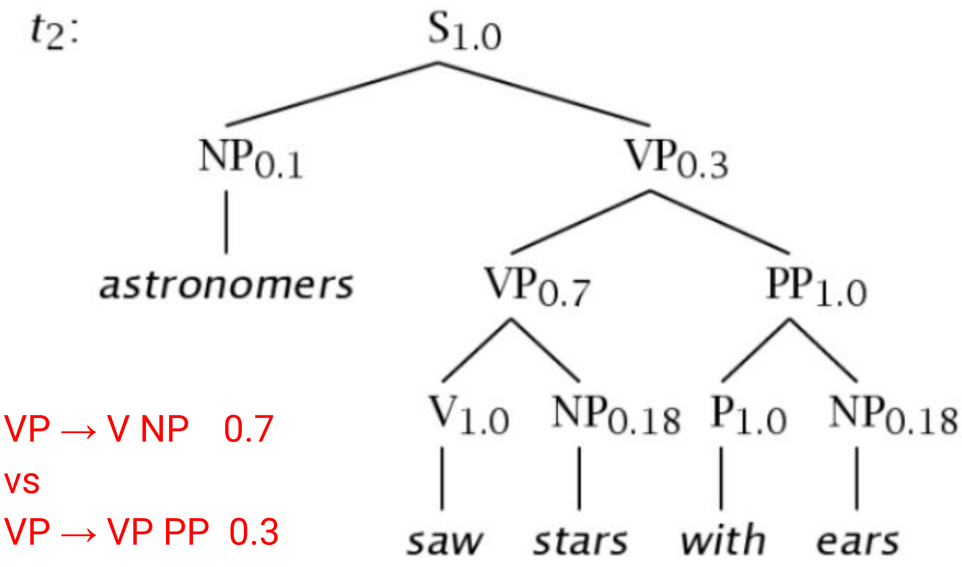
Statistical parsing – intuition

Example by Chris Manning

The following trees can be produced for the sentence

s = “astronomers saw stars with ears”





VP → V NP 0.7

vs

VP → VP PP 0.3

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 → Y Z or X → w
  - Modify probabilities s.t. the probability of each parse remains the same under the new CNF grammar. E.g.:

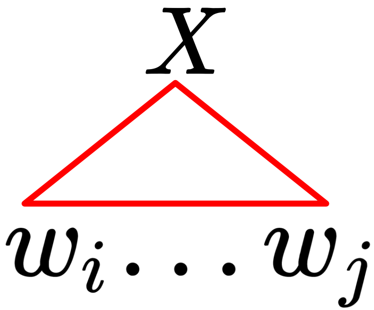
VP → Vt NP PP	0.2	⇒	VP → New PP	0.2
			New → Vt NP	1.0

- Build matrix as with CKY before
- Other **unary rules** can be included, e.g. NP → 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
- $w_{1\,n} = w_1 \ldots w_n$ = the word sequence from *1* to *n*
- $w_{i\,j}$ = the subsequence $w_i \ldots w_j$
- $X_{i\,j}$ = the nonterminal X dominating $w_i \ldots w_j$



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

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

The CKY algorithm for PCFG

- Define the dynamic table

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

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

$i <= j$

- With the goal to calculate

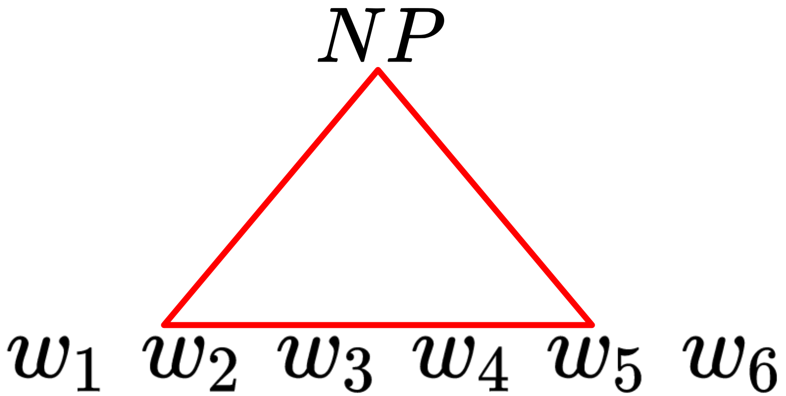
$$max_{t \in \mathcal{T}(s)} p(t) = \pi[1, n, S]$$

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 $i = 1...n, X \in N$

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

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

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

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

The CKY algorithm for PCFG

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

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

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

  Terminals / unary rules

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

  $$\pi[i, j, X] = \max_{\substack{X \to YZ \in R, \\ s \in \{i \ldots (j-1)\}}} \left(q(X \to YZ) \times \underline{\pi[i, s, Y]} \times \underline{\pi[s+1, j, Z]} \right)$$

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

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

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

Pseudocode by Michael Collins

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

Initialization:

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

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

Algorithm:

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

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

      and

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

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

Hyp

X:a
Y:b
Z: cd
W: abcd

Ref

X: a
Z: b
V: cd
Y: bcd
W: abcd

These are flattened constituents.

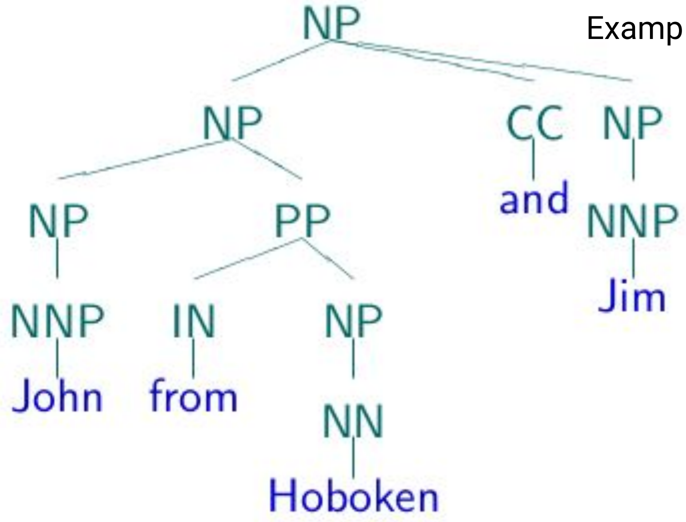
Discarding LHS symbol:
Precision = 4/4
Recall = ⅘

F-measure = …

Evaluating parsers – example

Example by Philipp Koehn

Hyp



Ref

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

NP_1: John from Hoboken and Jim

NP_2: John

NP_3: John from Hoboken

PP_1: from Hoboken

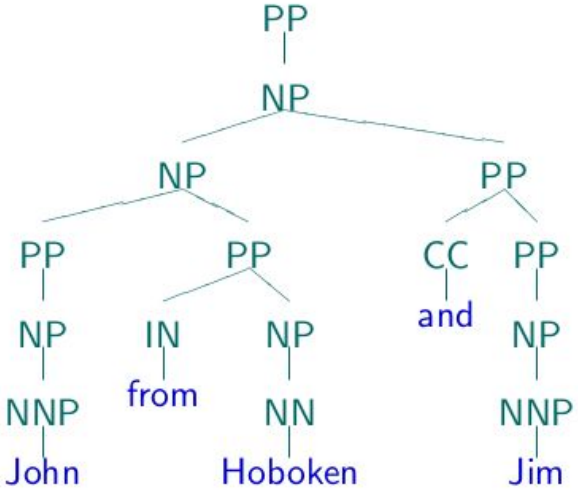
NP_4: Hoboken

NP_5: Jim

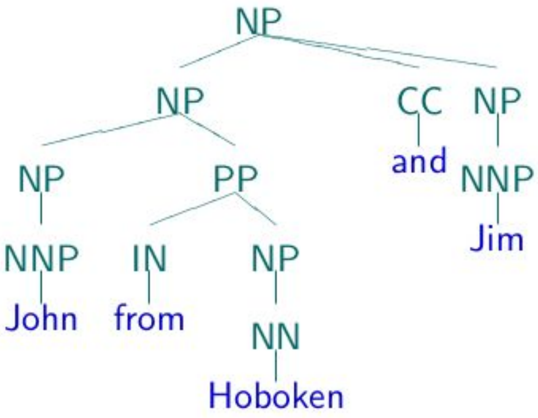
Evaluating parsers – exercise

Example by Philipp Koehn

Hyp



Ref



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

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

Questions?

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

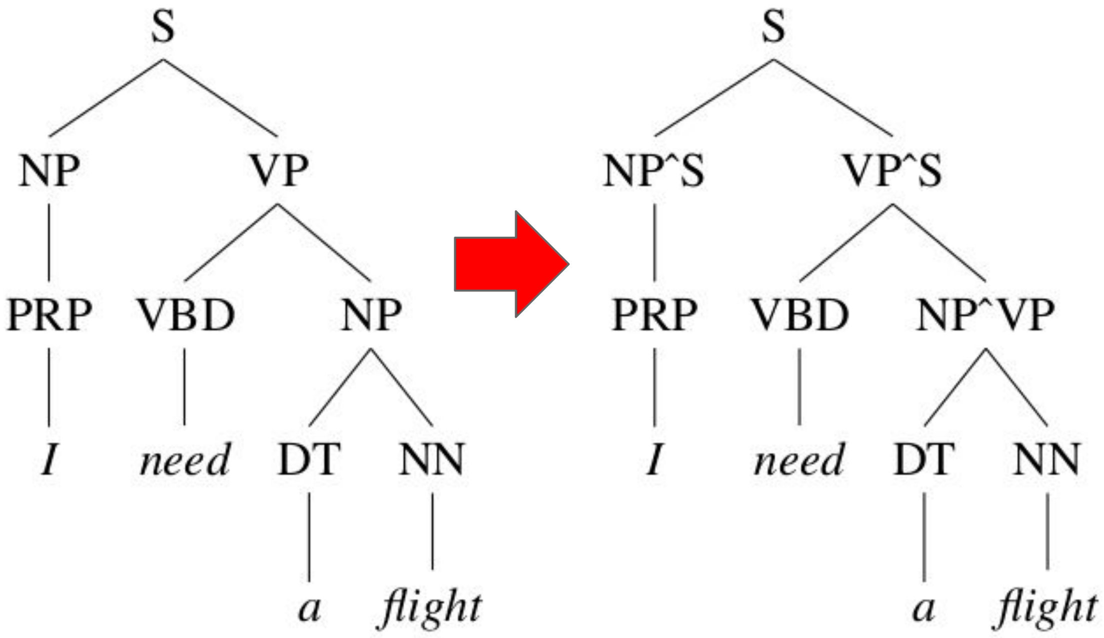
- **Poor independence assumption**:
    - Probability estimates of rules computed independently of surrounding context, e.g.:
    - In English, NPs in the **subject** function are more likely to be derived as pronominal (91%), while NPs in **object** function are more likely to be derived as non-pronominal (66%)
    - However, given overall probabilities
        - NP → DET N 0.28
        - NP → 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 → DET N and NP → PRP on whether the NP is subject or object → parent annotation
    $NP_{subject} \rightarrow PRN$ 0.91 (PRN: Personal Pronoun)
    $NP_{object} \rightarrow PRN$ 0.34
  - To do so, annotate each node with its parent
    - In this case, S → NP^S VP^S could indicate that the first NP is the subject

Extensions PCFG

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



From Jurafsky, D and Martin, J, “Speech and Language Processing,” 2018, ch 14

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	$\Rightarrow$	NP	VP
VP	$\Rightarrow$	Vi	
VP	$\Rightarrow$	Vt	NP
VP	$\Rightarrow$	VP	PP
NP	$\Rightarrow$	DT	NN
NP	$\Rightarrow$	NP	PP
PP	$\Rightarrow$	IN	NP

Vi	$\Rightarrow$	sleeps
Vt	$\Rightarrow$	saw
NN	$\Rightarrow$	man
NN	$\Rightarrow$	woman
NN	$\Rightarrow$	telescope
DT	$\Rightarrow$	the
IN	$\Rightarrow$	with
IN	$\Rightarrow$	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

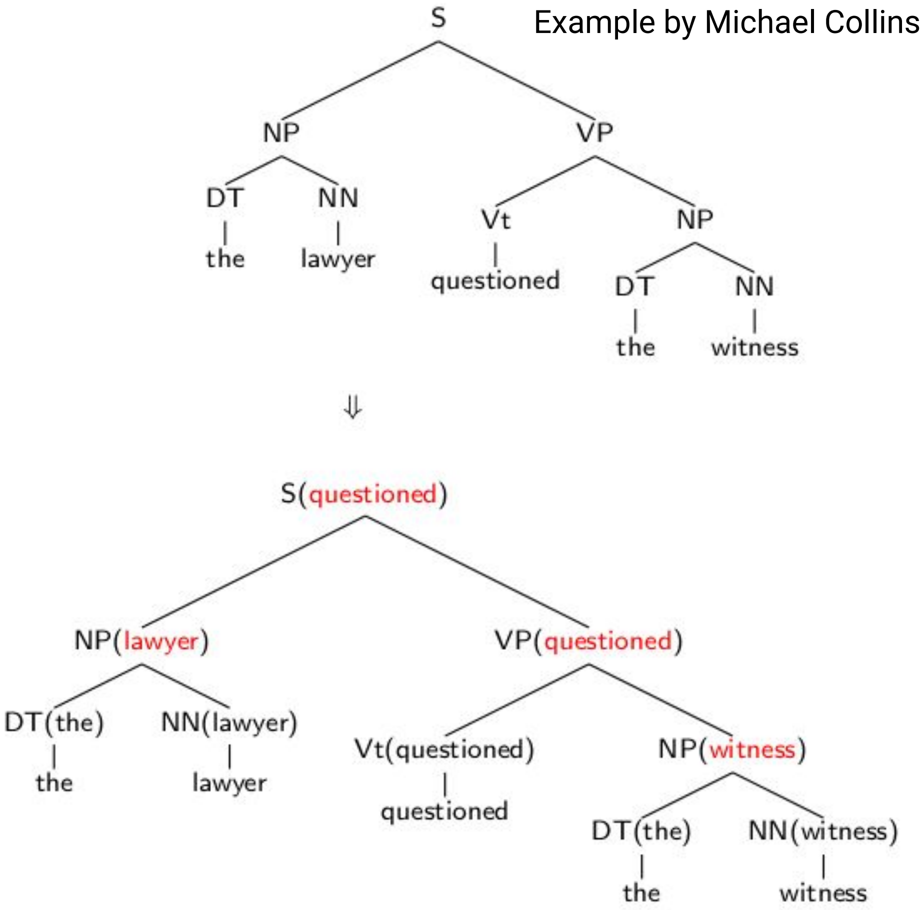
LPCFG grammar example:

$$
\begin{array}{lll}
\mathsf{S(saw)} & \rightarrow_2 & \mathsf{NP(man)} \quad \mathsf{VP(saw)} \\
\mathsf{VP(saw)} & \rightarrow_1 & \mathsf{Vt(saw)} \quad \mathsf{NP(dog)} \\
\mathsf{NP(man)} & \rightarrow_2 & \mathsf{DT(the)} \quad \mathsf{NN(man)} \\
\mathsf{NP(dog)} & \rightarrow_2 & \mathsf{DT(the)} \quad \mathsf{NN(dog)} \\
\mathsf{Vt(saw)} & \rightarrow & \mathsf{saw} \\
\mathsf{DT(the)} & \rightarrow & \mathsf{the} \\
\mathsf{NN(man)} & \rightarrow & \mathsf{man} \\
\mathsf{NN(dog)} & \rightarrow & \mathsf{dog}
\end{array}
$$

Probabilistic Lexicalised CFG

Example by Michael Collins

- A constituent receives its headword from its headchild



S ⟶ NP VP (S receives from VP)
VP ⟶ Vt VP (VP receives from Vt)
NP ⟶ DT NN (NP receives from NN)

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

Example by Michael Collins

If the rule contains NN, NNS, or NNP:
Choose the rightmost NN, NNS, or NNP — Rule 1

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	$\Rightarrow$	DT	NNP	NN	Rule 1
NP	$\Rightarrow$	DT	NN	NNP	Rule 1
NP	$\Rightarrow$	NP	PP		Rule 2
NP	$\Rightarrow$	DT	JJ		Rule 3
NP	$\Rightarrow$	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.,

$$\begin{array}{lllll} \mathsf{VP} & \Rightarrow & \mathsf{Vt} & \mathsf{NP} & \text{Rule 6} \\ \mathsf{VP} & \Rightarrow & \mathsf{VP} & \mathsf{PP} & \text{Rule 7} \end{array}$$

Probabilistic Lexicalised CFG

Estimating the probabilities from treebank is similar:

- PCFG:

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

- LPCFG

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

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