Constituency Parsing (cont'd)
+ PCFGs (if time)

CS 485, Fall 2023
Applications of Natural Language Processing
https://people.cs.umass.edu/~brenocon/cs485_f23/

Brendan O'Connor
College of Information and Computer Sciences
University of Massachusetts Amherst

UMass CS 485 In-class Exercise 10/19/23 Name: ____________________

Fill in the CYK dynamic programming table to parse the sentence below. In the bottom right corner, draw the two parse trees. Show the possible nonterminals in each cell. Optional: draw the backpointers too.

S → NP VP
NP → NP PP
VP → V NP
VP → VP PP
PP → P NP

NP → she
NP → fish
NP → fork
NP → chopsticks
V → eats
V → fish
P → with

Recursive

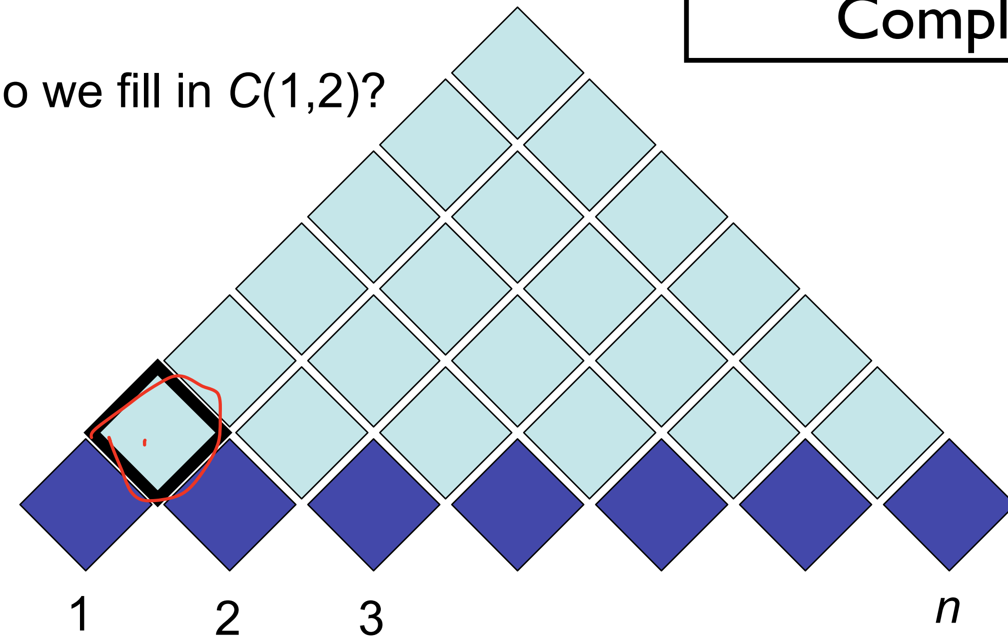
Dyn. Prog.

$O(N^2)$ cells

For cell [i,j] $O(n^2)$
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]

Computational Complexity ?

How do we fill in $C(1,2)$?



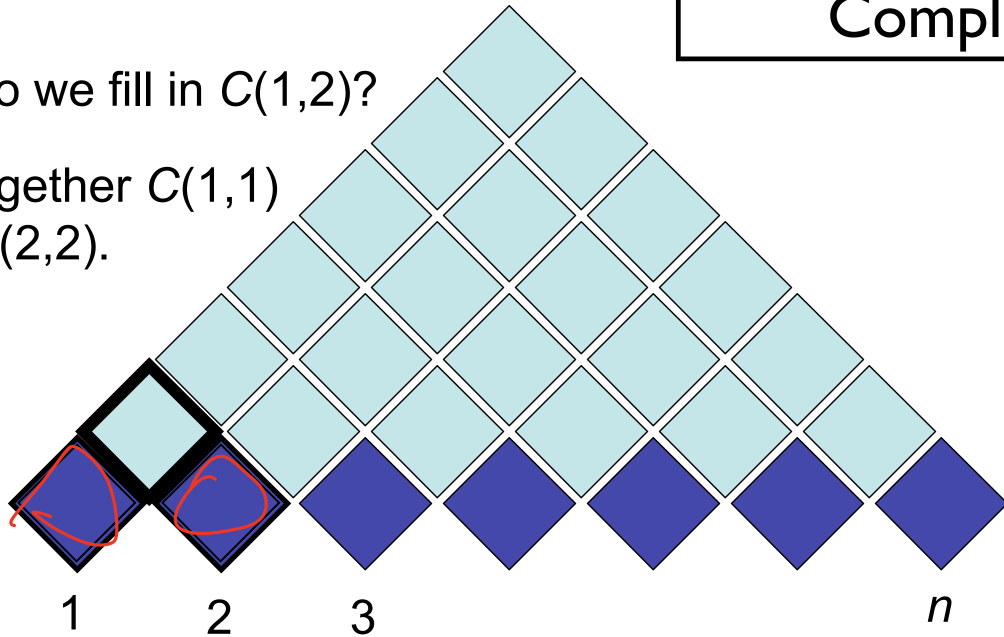
[Example from Noah Smith]

For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]

Computational Complexity ?

How do we fill in *C*(1,2)?

Put together *C*(1,1) and *C*(2,2).

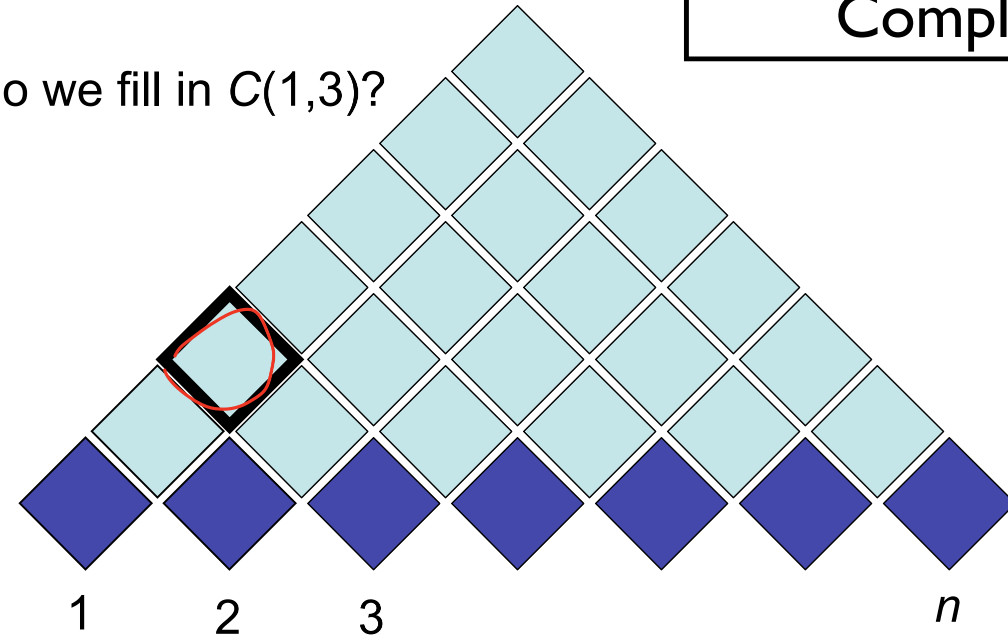


[Example from Noah Smith]

For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]

Computational Complexity ?

How do we fill in *C*(1,3)?



[Example from Noah Smith]

For cell [i,j]

For possible splitpoint k=(i+1)..(j-1):

For every B in [i,k] and C in [k,j],

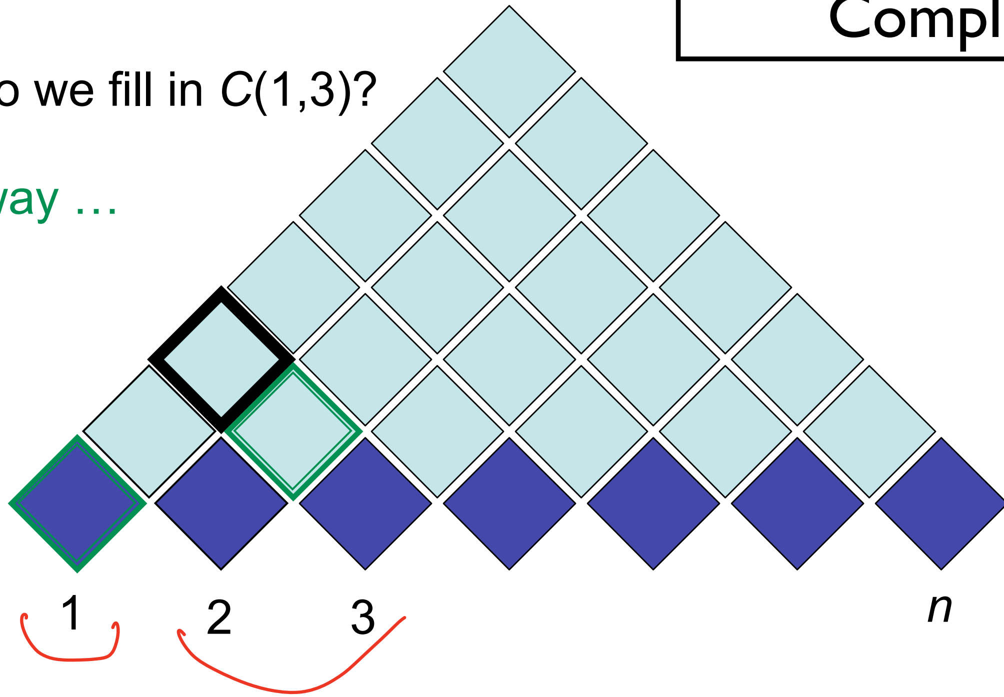
If exists rule A -> B C,

add A to cell [i,j]

Computational Complexity ?

How do we fill in *C*(1,3)?

One way …



[Example from Noah Smith]

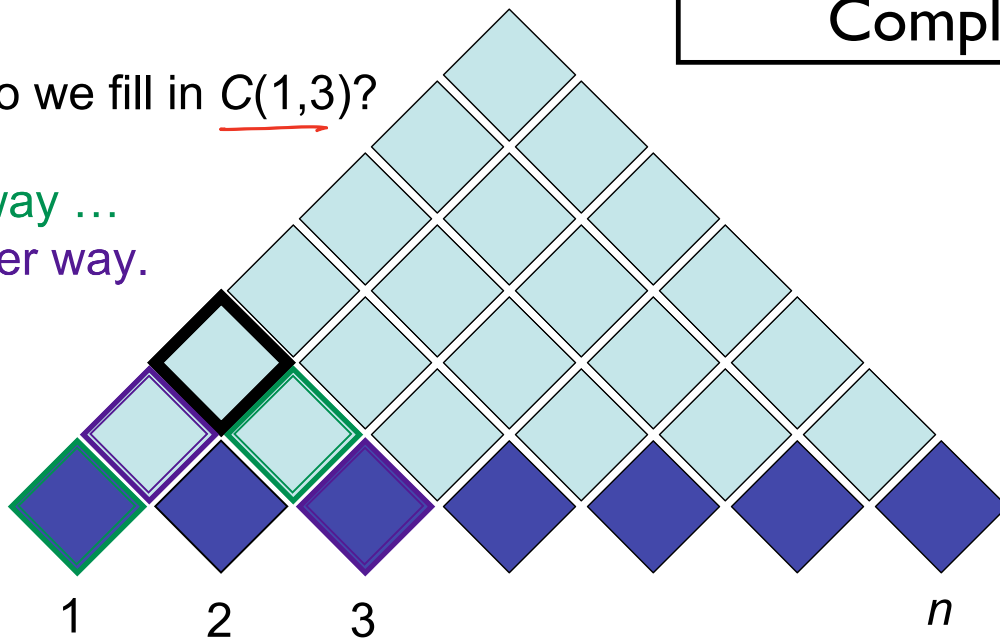
For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]

Computational Complexity ?

How do we fill in $C(1,3)$?

One way …

Another way.

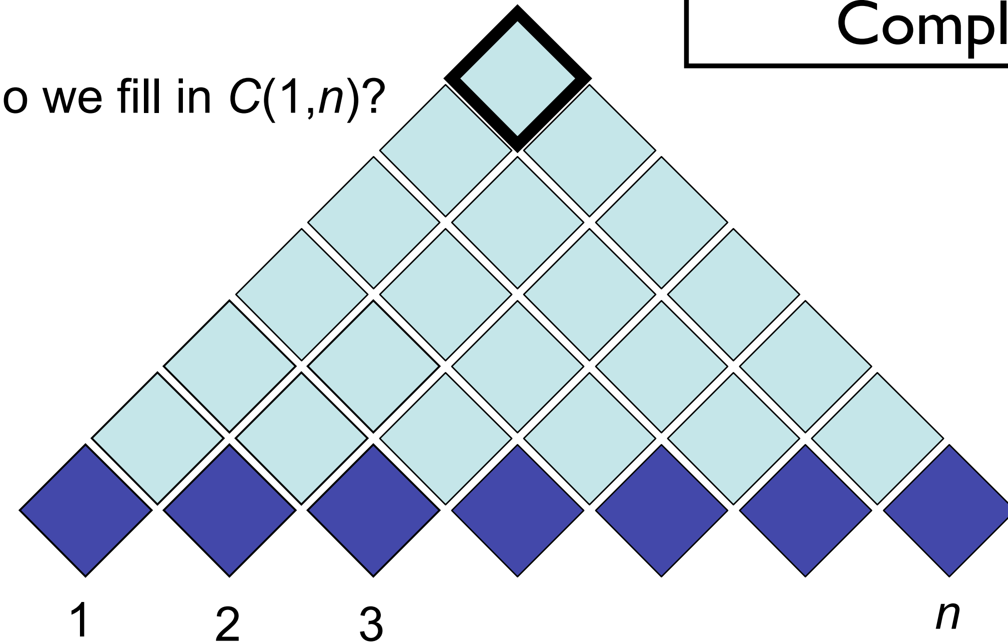


[Example from Noah Smith]

For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]

Computational Complexity ?

How do we fill in $C(1,n)$?



[Example from Noah Smith]

For cell [i,j]

  For possible splitpoint k=(i+1)..(j-1):

    For every B in [i,k] and C in [k,j],

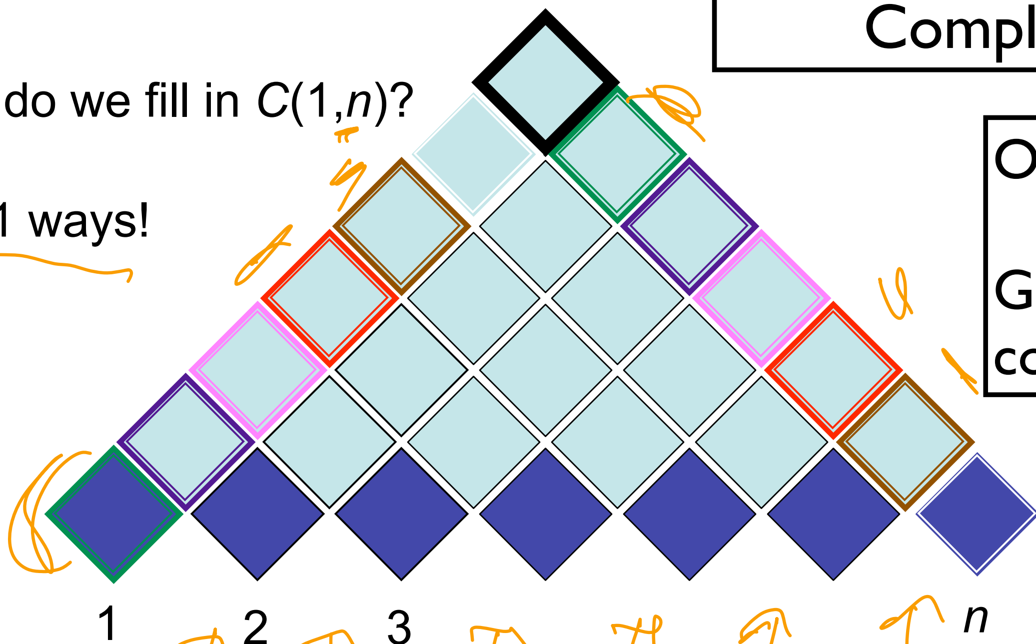
      If exists rule A -> B C,

        add A to cell [i,j]

Computational Complexity ?

How do we fill in $C(1,n)$?

n - 1 ways!



O(G n^3)

G = grammar constant

O(N) splits for 1 cell

O(N²) cells total ⇒ O(N³)

[Example from Noah Smith]

Probabilistic CFGs

P(S→NP VP | parent=S)
cur

$S \rightarrow NP\ VP$ [.80]	$Det \rightarrow that$ [.10] | a [.30] | the [.60]
$S \rightarrow Aux\ NP\ VP$ [.15]	$Noun \rightarrow book$ [.10] | $flight$ [.30]
$S \rightarrow VP$ [.05]	| $meal$ [.15] | $money$ [.05]
$NP \rightarrow Pronoun$ [.35]	| $flights$ [.40] | $dinner$ [.10]
$NP \rightarrow Proper\text{-}Noun$ [.30]	$Verb \rightarrow book$ [.30] | $include$ [.30]
$NP \rightarrow Det\ Nominal$ [.20]	| $prefer;$ [.40]
$NP \rightarrow Nominal$ [.15]	$Pronoun \rightarrow I$ [.40] | she [.05]
$Nominal \rightarrow Noun$ [.75]	| me [.15] | you [.40]
$Nominal \rightarrow Nominal\ Noun$ [.20]	$Proper\text{-}Noun \rightarrow Houston$ [.60]
$Nominal \rightarrow Nominal\ PP$ [.05]	| TWA [.40]
$VP \rightarrow Verb$ [.35]	$Aux \rightarrow does$ [.60] | can [40]
$VP \rightarrow Verb\ NP$ [.20]	$Preposition \rightarrow from$ [.30] | to [.30]
$VP \rightarrow Verb\ NP\ PP$ [.10]	| on [.20] | $near$ [.15]
$VP \rightarrow Verb\ PP$ [.15]	| $through$ [.05]
$VP \rightarrow Verb\ NP\ NP$ [.05]	
$VP \rightarrow VP\ PP$ [.15]	
$PP \rightarrow Preposition\ NP$ [1.0]	

- Defines a probabilistic generative process for words in a sentence
  - Can parse with a modified form of CKY
- How to learn? Fully supervised if you have a treebank

[J&M textbook]

PCFG as LM

w: sentence $\{w_1, w_2 \dots w_N\}$

y: tree

- sample p(w,y) = p(w|y) p(y)

$p(w, y) =$

$.8 \times .2 \times .6 \times \dots\dots$

ROOT

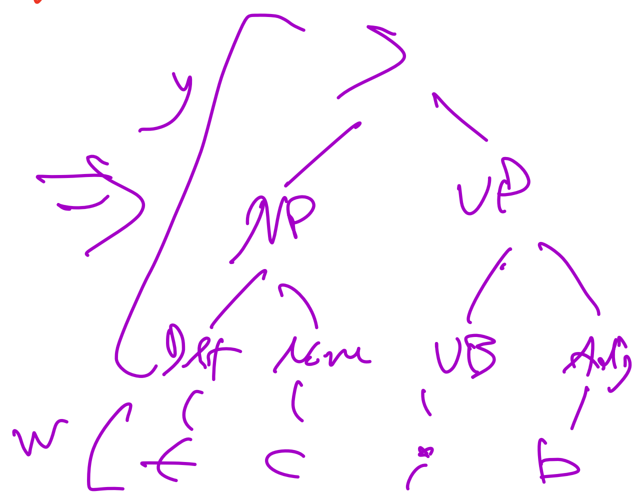
$\Rightarrow$ S　　　S → NP VP (.8)

$\Rightarrow$ NP VP　　　NP → Det Nom (.2)

$\Rightarrow$ Det Nom VP　　　Det → the (.6)

$\Rightarrow$ the Nom VP

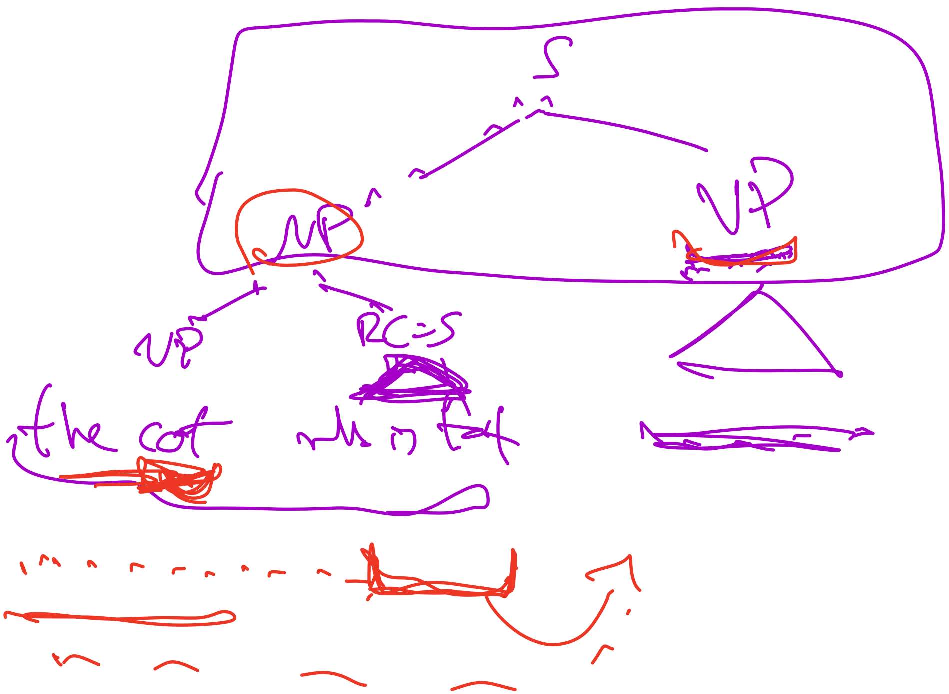
the cat is back



N-gram vs PCFG LM

$P(w_8 | w_6, w_7)$

- We also could sample from an n-gram (Markov) LM.... what differences should we expect?



PCFG as LM

- p(w,y) = multiply all the expansion probabilities

$\log p(w,y)$ = sum of expansion log probs!

(P)CFG model, (P)CKY algorithm

- CKY: given CFG and sentence w
  - Does there exist at least one parse?
  - Enumerate parses (backpointers)

y = tree (nonterms.)

- Probabilistic CKY: given PCFG and sentence w
  - Most probable parse ("Viterbi parse")

    $\hat{y} = \text{argmax}_y \underbrace{P(y \mid w)}_{P(\text{tree} \mid \text{sent})}$

    $P(y|w) \propto \overbrace{P(y)\,P(w|y)}^{\text{multiply exprs!}}$

  - Likelihood of sentence ("Inside algorithm")

    $P(w) = \Sigma_y P(w \mid y)\, P(y)$

- a PCFG with Penn Treebank's nonterminals encodes overly strong conditional independence assumptions - big problems for both generation and parsing
- a bunch of tricks improve treebank-trained PCFGs to get better parsing performance
  - ~80% F1: "Treebank grammar" (PCFG directly trained on PTB) span NT pred.
  - ~90% F1: PCFG with clever non-terminal splitting
  - ~95% F1: state of the art (not PCFG)

Very hard!!!

Better PCFG grammars

- Nonterminal splitting: because substitutability is too strong (e.g. “she” as subject vs object)

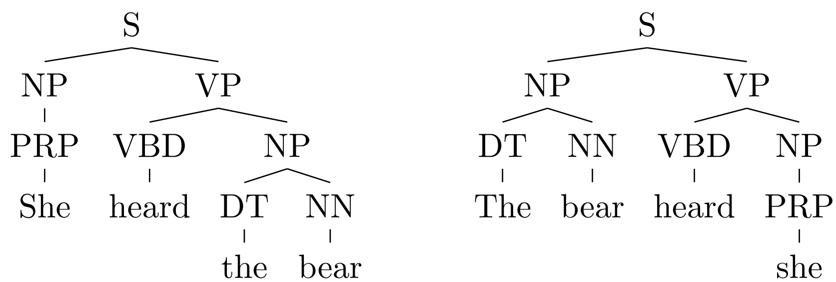


Figure 11.5: A grammar that allows *she* to take the object position wastes probability mass on ungrammatical sentences.

[From Eisenstein (2017)]

Better PCFG grammars

- Parent annotation

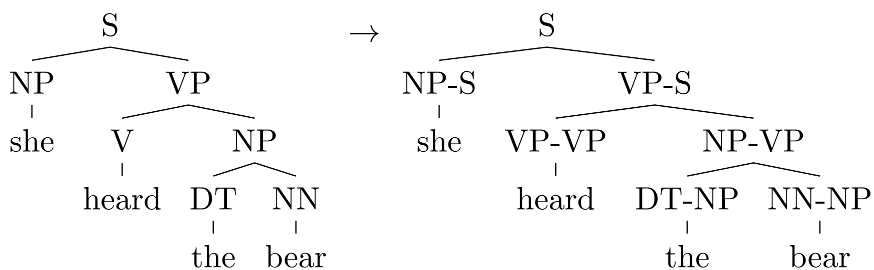


Figure 11.8: Parent annotation in a CFG derivation

[From Eisenstein (2017)]

Better PCFG grammars

- Linguistically designed state splits

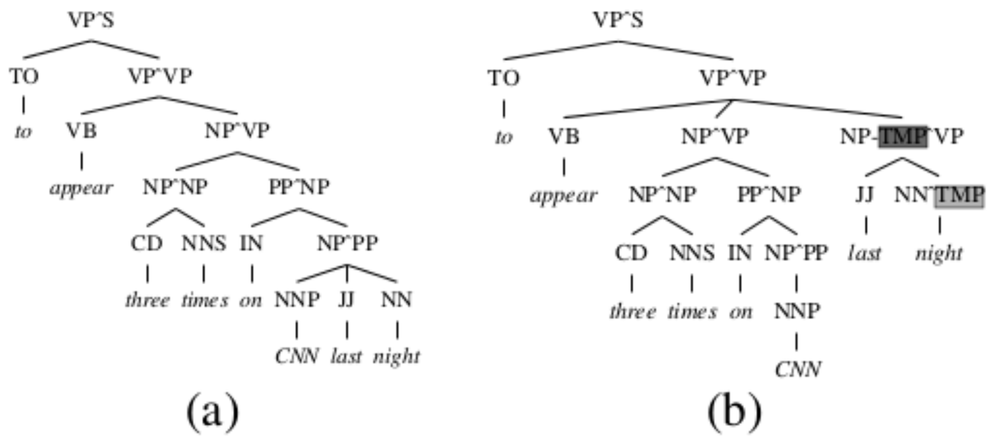


Figure 11.13: State-splitting creates a new non-terminal called NP-TMP, for temporal noun phrases. This corrects the PCFG parsing error in (a), resulting in the correct parse in (b).

[From Eisenstein (2017)]

Better PCFG grammars

- Lexicalization: encode semantic preferences

Non-terminal	Direction	Priority
S	right	VP SBAR ADJP UCP NP
VP	left	VBD VBN MD VBZ TO VB VP VBG VBP ADJP NP
NP	right	N* EX $ CD QP PRP ...
PP	left	IN TO FW

Table 11.3: A fragment of head percolation rules

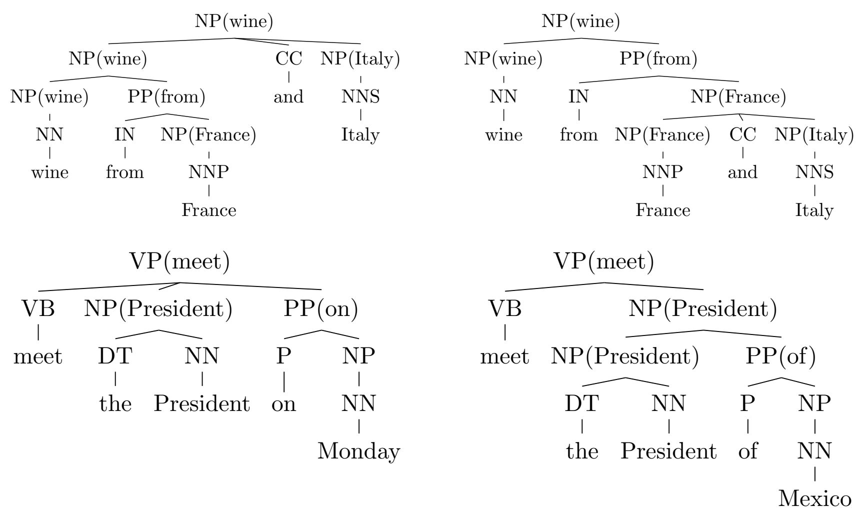


Figure 11.9: Lexicalization can address ambiguity on coordination scope (upper) and PP attachment (lower)

[From Eisenstein (2017)]

- a PCFG with Penn Treebank's nonterminals encodes overly strong conditional independence assumptions - big problems for both generation and parsing
- a bunch of tricks improve treebank-trained PCFGs to get better parsing performance
  - ~80% F1: "Treebank grammar" (PCFG directly trained on PTB)
  - ~90% F1: PCFG with clever non-terminal splitting
  - ~95% F1: state of the art (not PCFG)

- Parsing model accuracy: lots of ambiguity!!
  - PCFGs lack lexical information to resolve ambiguities (sneak in world knowledge?)
  - PCFGs that are successful parsers sneak in lexical information into the non-terminals ... but there are limits how much you can do
  - Next time: dependency parsing

- Practical guidance
  - O(N) left-to-right incremental algorithms are more practical than CKY
  - Look carefully at parser's errors — are they tolerable for your application?