Constituency Parsing: CKY

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

Context-Free Grammar

- CFG describes a generative process for an (infinite) set of strings
  - 1. Nonterminal symbols
    - "S": START symbol / "Sentence" symbol
  - 2. Terminal symbols: word vocabulary
  - 3. Rules (a.k.a. Productions). Practically, two types:

<u>"Grammar": *one* NT expands to >=1 NT</u>
always one NT on left side of rulep

S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

<u>Lexicon: NT expands to a terminal</u>

Noun	→	*flights* | *breeze* | *trip* | *morning* | ...
Verb	→	*is* | *prefer* | *like* | *need* | *want* | *fly*
Adjective	→	*cheapest* | *non − stop* | *first* | *latest* | *other* | *direct* | ...
Pronoun	→	*me* | *I* | *you* | *it* | ...
Proper-Noun	→	*Alaska* | *Baltimore* | *Los Angeles* | *Chicago* | *United* | *American* | ...
Determiner	→	*the* | *a* | *an* | *this* | *these* | *that* | ...
Preposition	→	*from* | *to* | *on* | *near* | ...
Conjunction	→	*and* | *or* | *but* | ...

Constituent Parse Trees

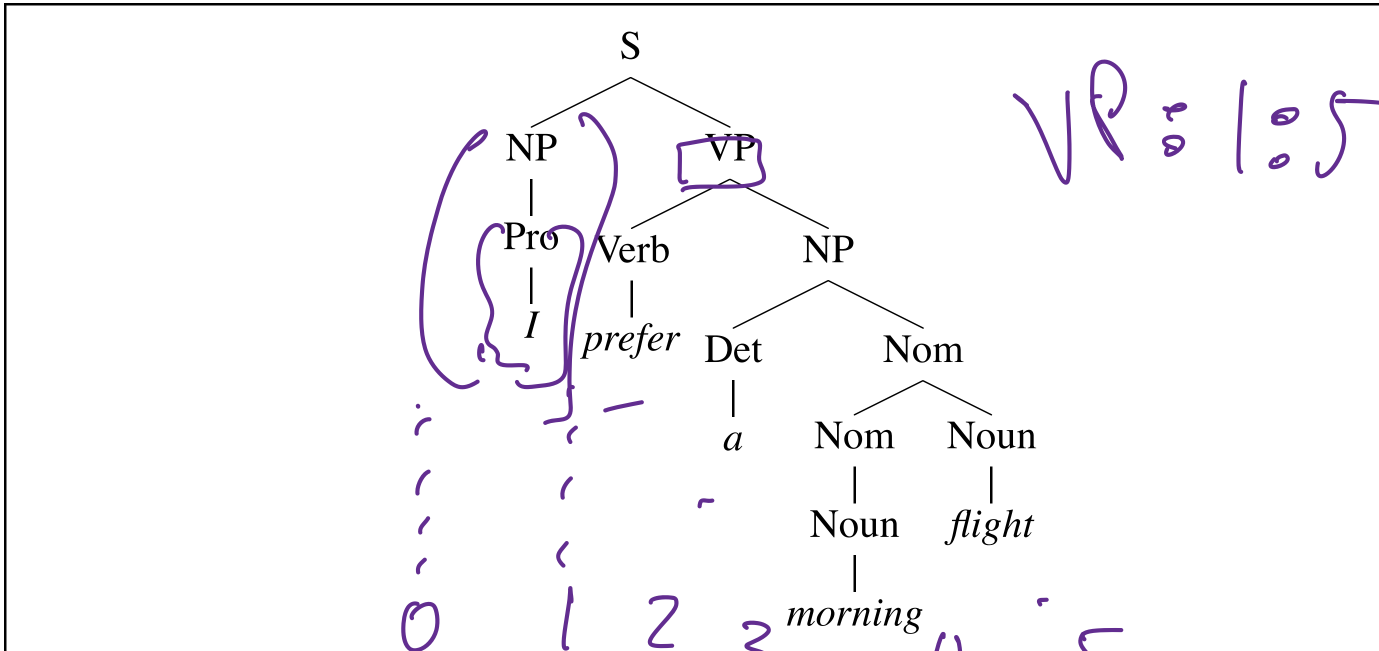


Figure 12.4 The parse tree for "I prefer a morning flight" according to grammar $\mathscr{L}_0$.

Bracket notation

(12.2) $[_S\ [_{NP}\ [_{Pro}$ I]] $[_{VP}\ [_V$ prefer] $[_{NP}\ [_{Det}$ a] $[_{Nom}\ [_N$ morning] $[_{Nom}\ [_N$ flight]]]]]]

<=> Set of non-terminal spans (start,end positions)

{(NP, 0, 1), (VP, 1, 5), (NP, 2, 5), ...}

Parsing with a CFG

- Task: given text and a CFG, answer:
  - Does there exist at least one parse?
  - Enumerate parses (backpointers)

⟺ is the string (text) grammatical (in this CFG)?

- Problem: extremely high number of possible trees for a sentence, and even a large number of *legal* trees (licensed by the grammar) for a sentence
  - Many parsing algorithms have been invented to tackle this

- Cocke-Kasami-Younger algorithm (CKY)
  - Bottom-up dynamic programming:
    Find possible nonterminals for short spans of sentence, then possible combinations for higher spans
  - *Maintains* local ambiguity, representing many subtrees for each span. ("Packed forest" representation)
  - Provably finds all possible parse trees (legal derivations), and correctly says when none exist.
  - Requires converting to Chomsky Normal Form (binarization)

Chomsky Normal Form

(binarized grammar)

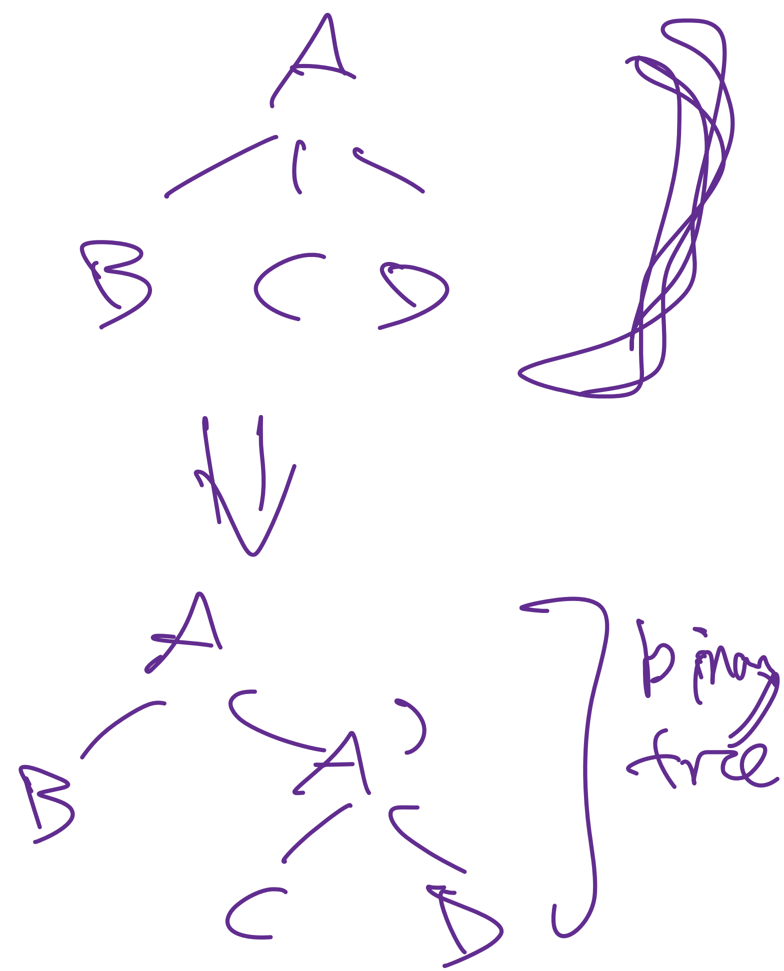
A → { } length 1 or 2

(eg) A → B C D (non CNF)

⇓

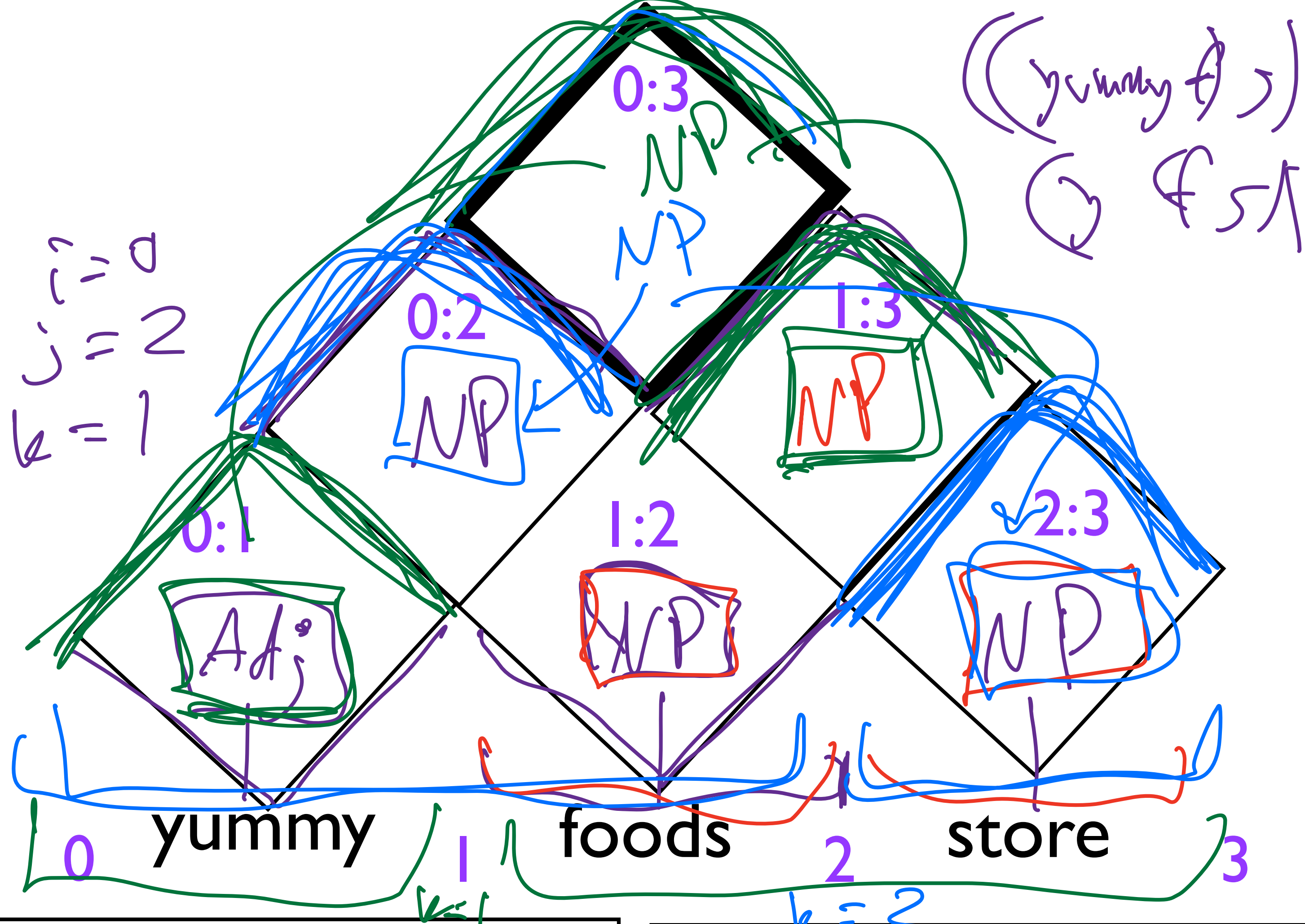
A → B A'

A' → C D



CKY

Grammar
Adj -> yummy
NP -> foods
NP -> store
NP -> NP NP
NP -> Adj NP



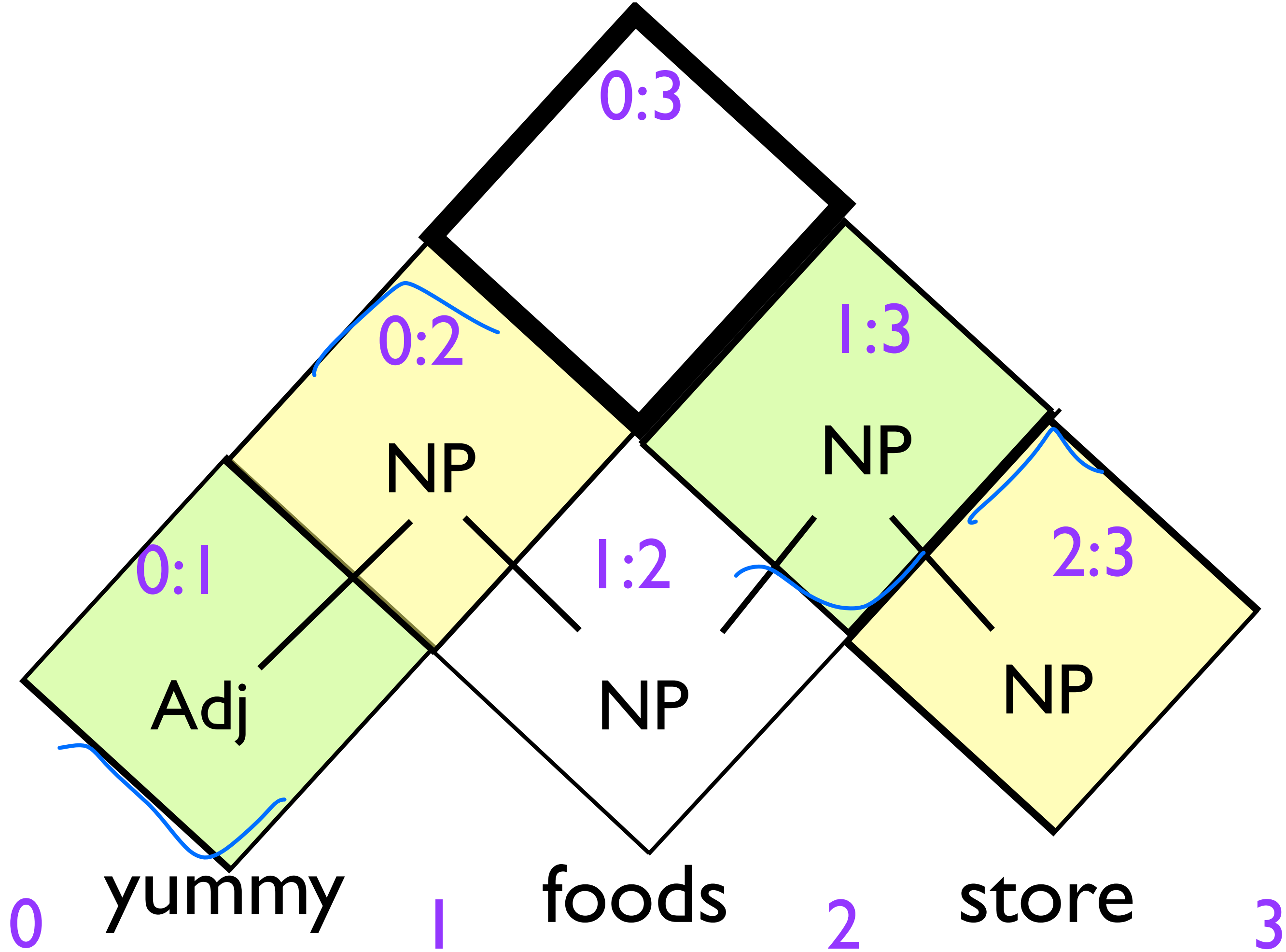
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]   (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

Recognizer: per span, record list of possible nonterminals

Parser: per span, record possible ways the nonterminal was constructed.

CKY

Grammar
Adj -> yummy
NP -> foods
NP -> store
NP -> NP NP
NP -> Adj NP



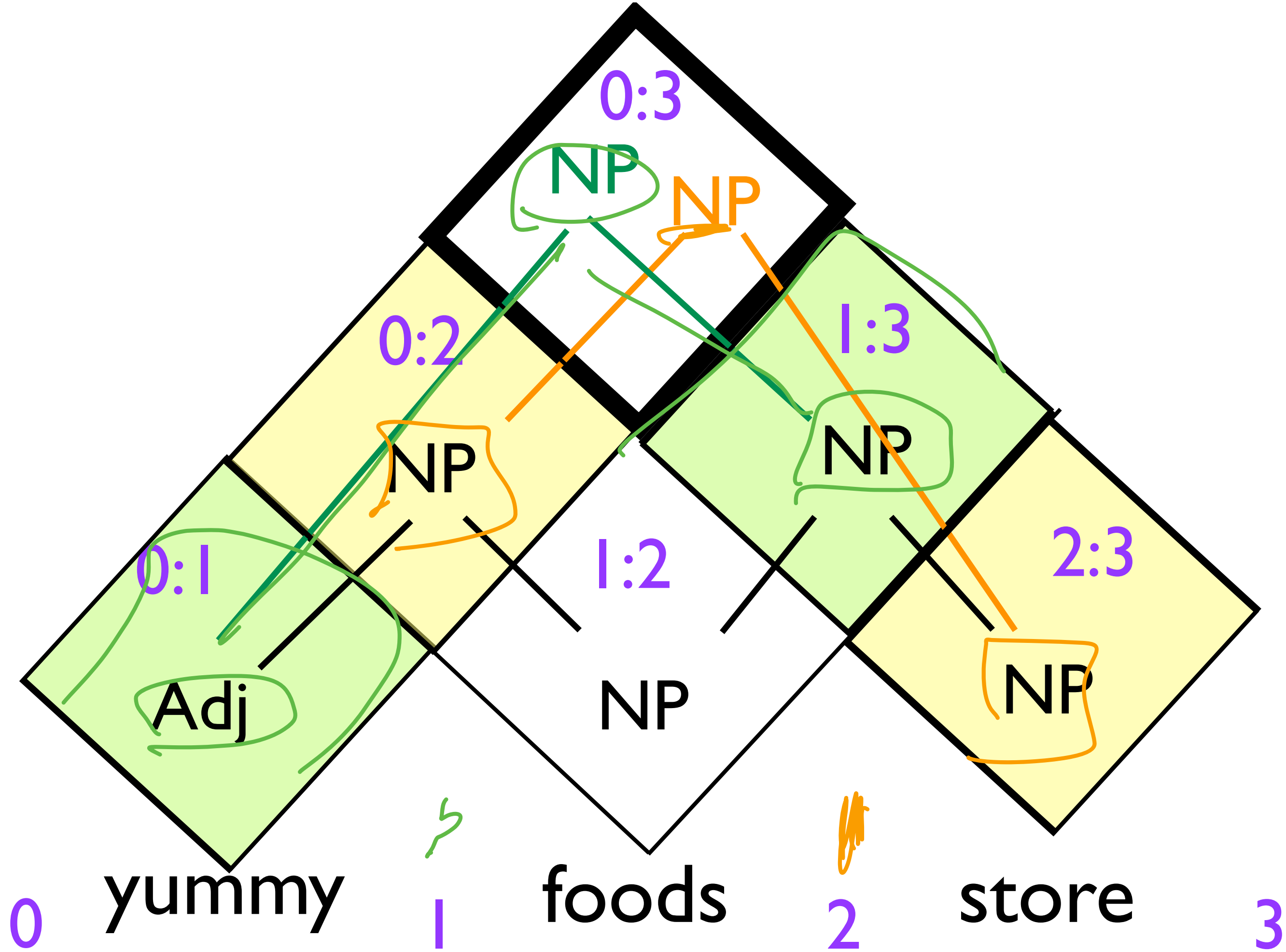
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

Recognizer: per span, record list of possible nonterminals

Parser: per span, record possible ways the nonterminal was constructed.

CKY

Grammar
Adj -> yummy
NP -> foods
NP -> store
NP -> NP NP
NP -> Adj NP

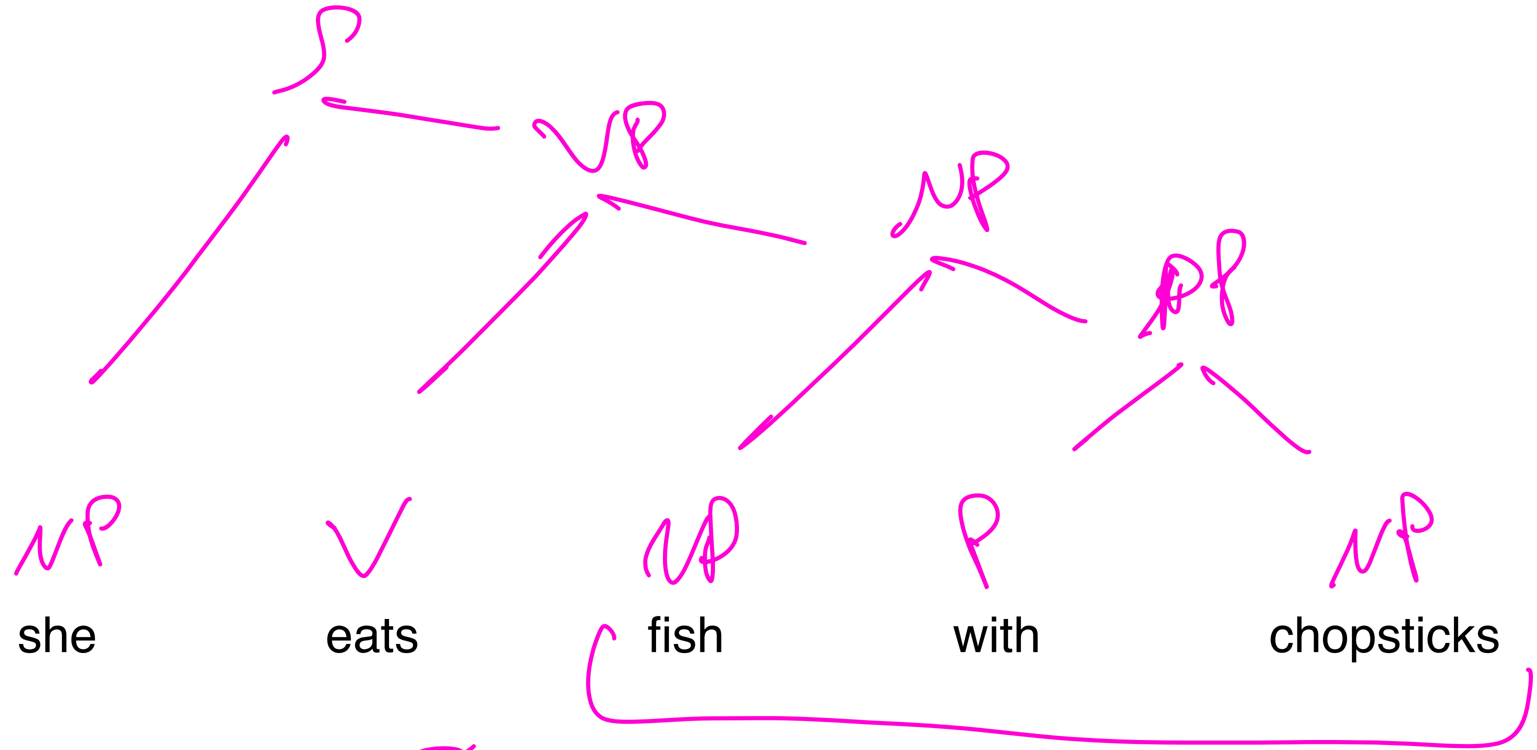


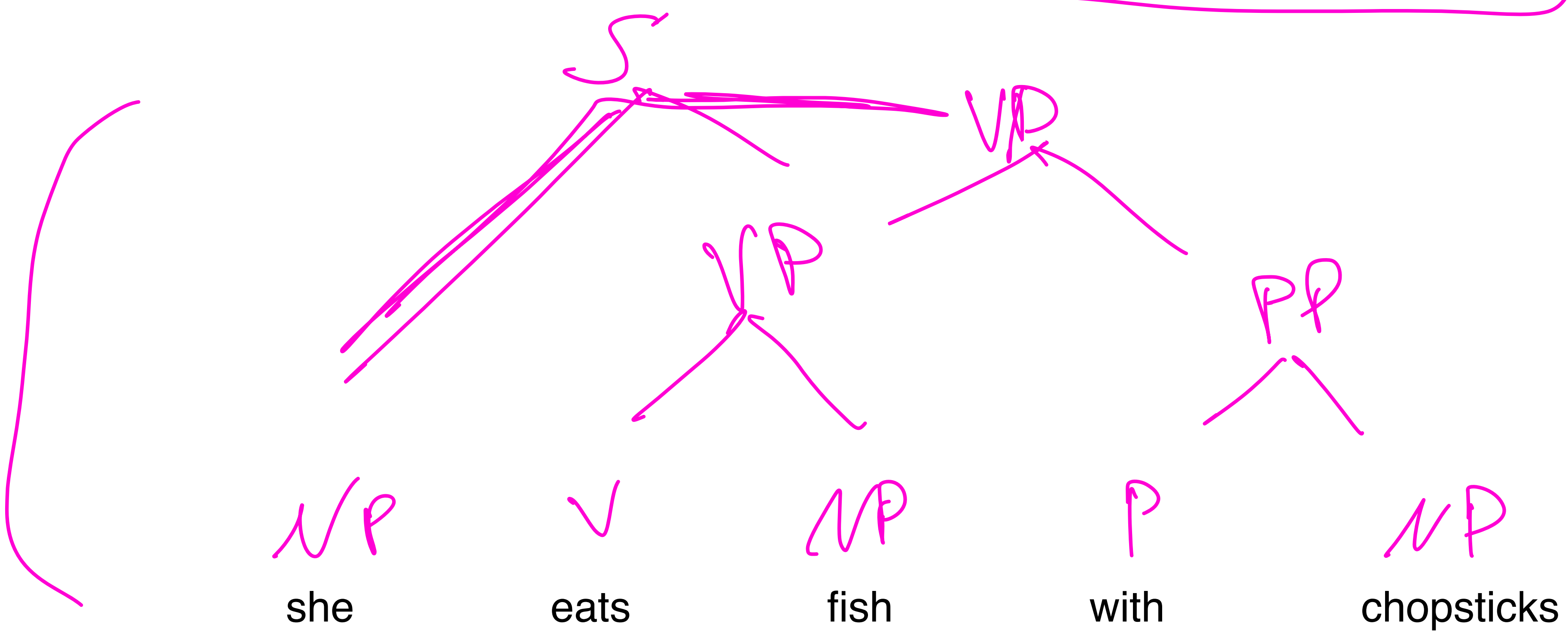
For cell [i,j] (loop through them bottom-up)
 For possible splitpoint k=(i+1)..(j-1):
  For every B in [i,k] and C in [k,j],
   If exists rule A -> B C,
    add A to cell [i,j] (*Recognizer*)
    ... or ...
    add (A,B,C, k) to cell [i,j] (*Parser*)

Recognizer: per span, record list of possible nonterminals

Parser: per span, record possible ways the nonterminal was constructed.

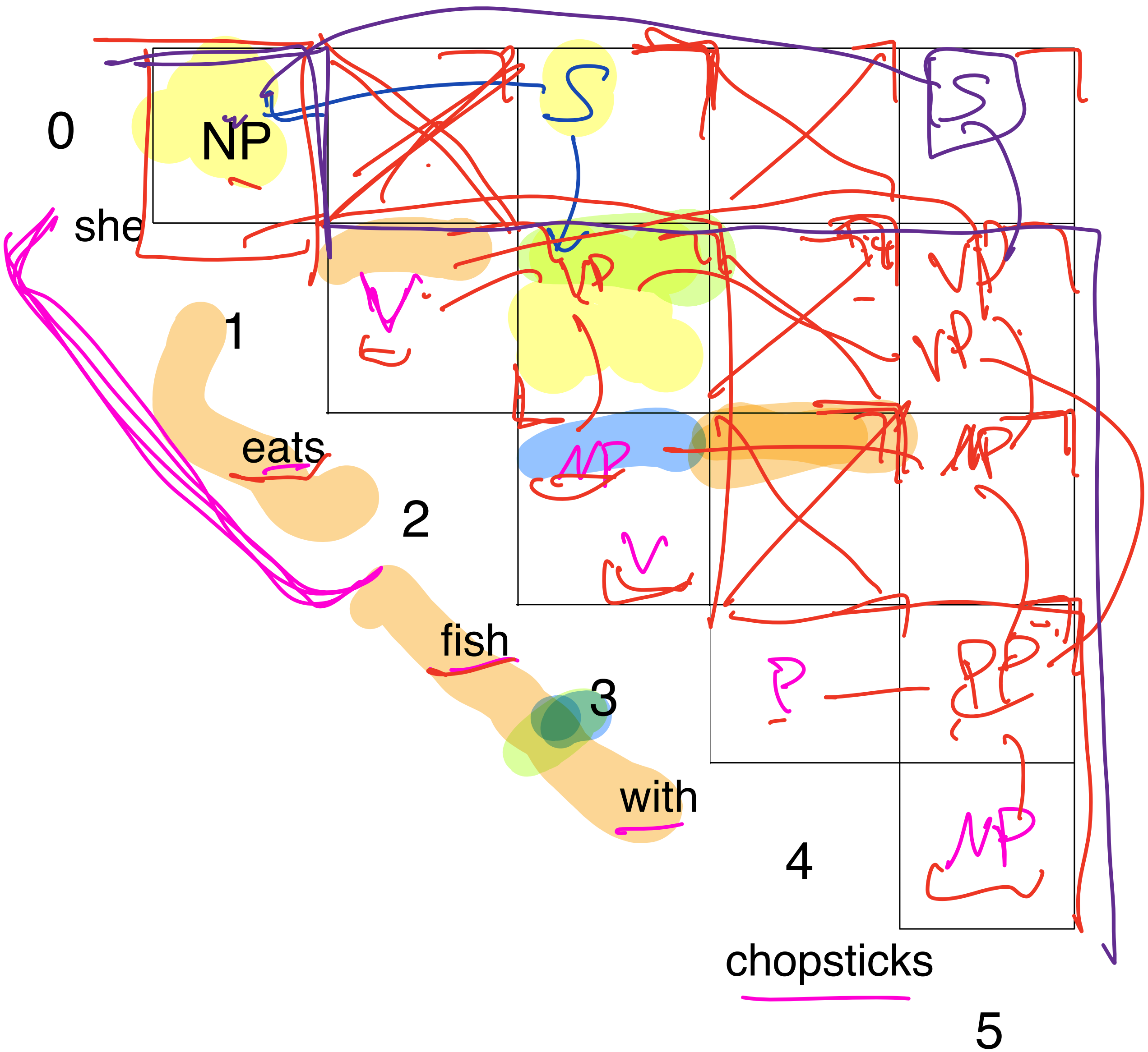
S → NP VP
NP → NP PP
VP → V NP
VP → VP PP
PP → P NP
NP → she
NP → fish
NP → fork
NP → chopsticks
V → eats
V → fish
P → with





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.



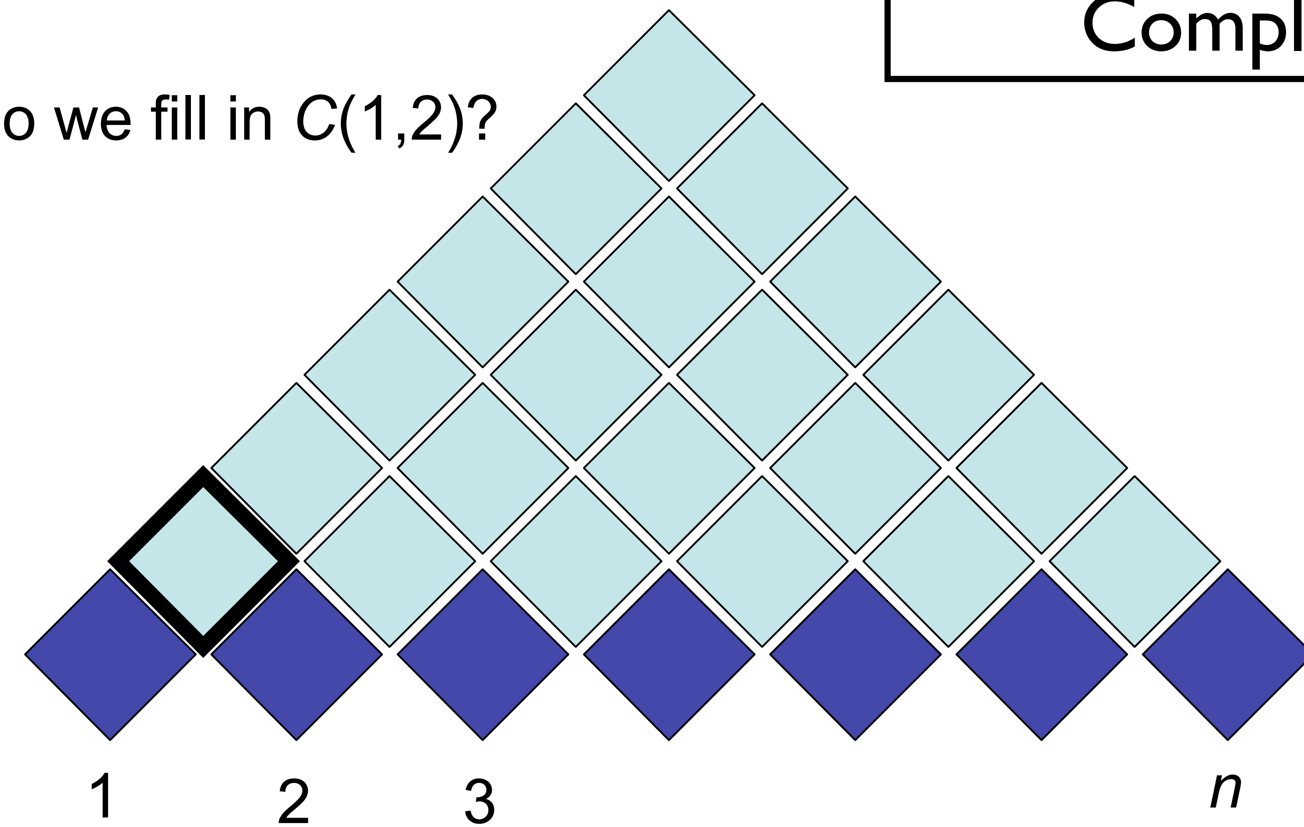
S → NP VP
NP → NP PP
VP → V NP
VP → VP PP
PP → P NP

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

```
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)?



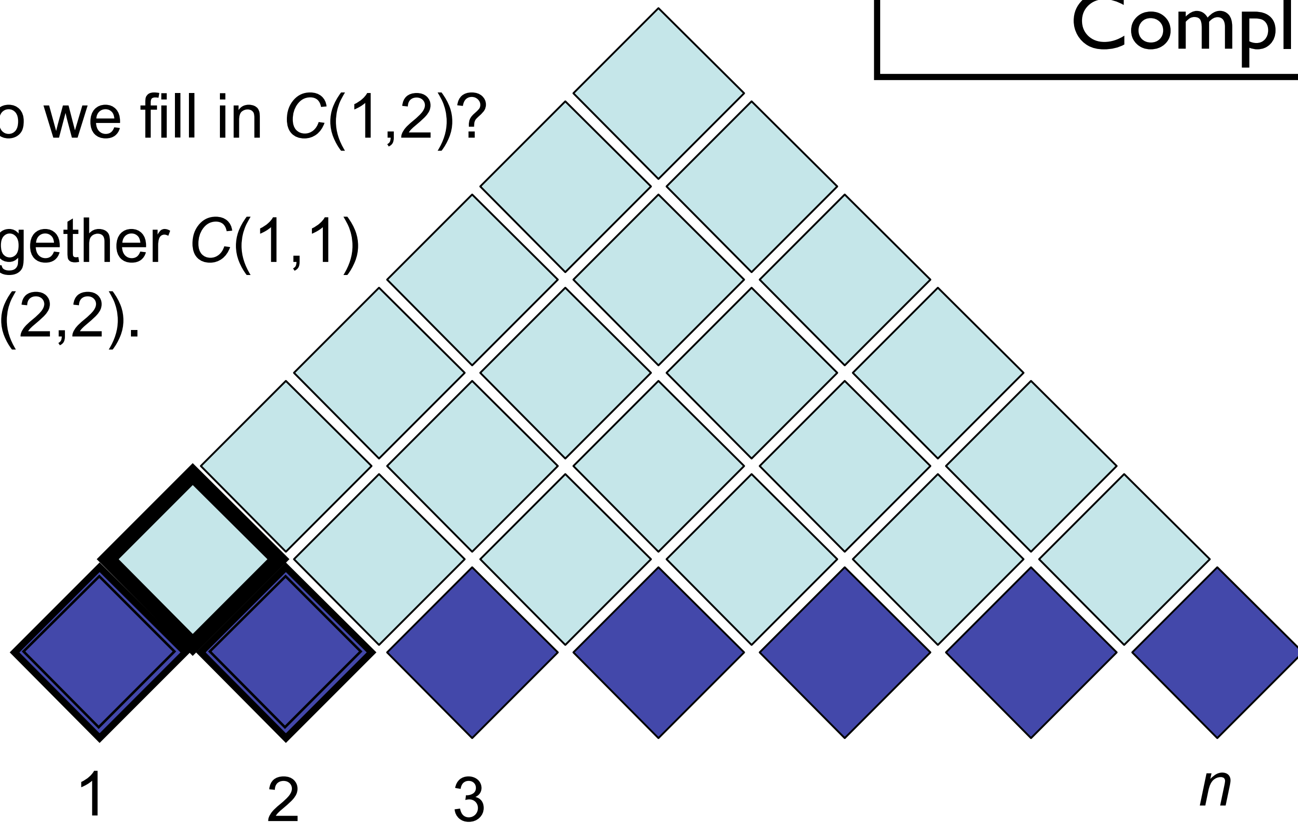
[Example from Noah Smith]

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).

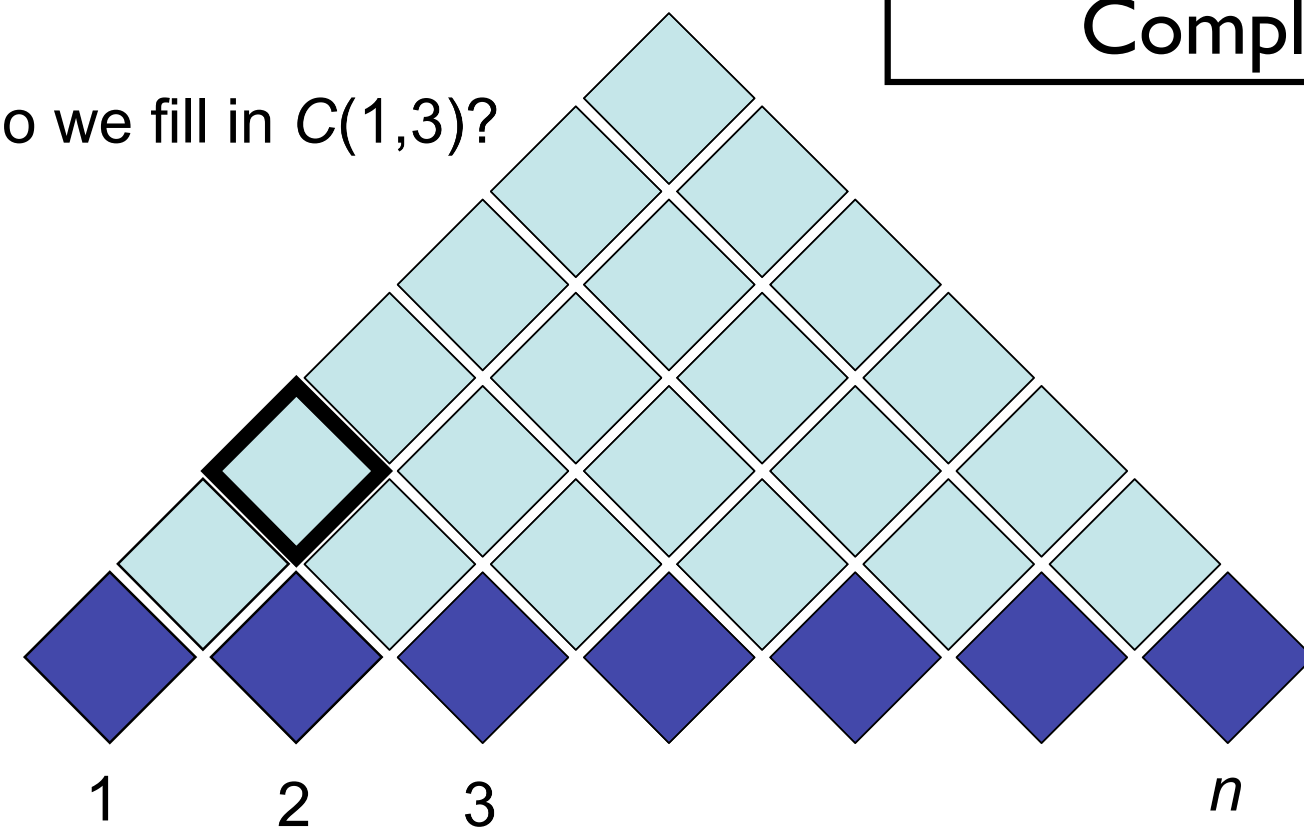


[Example from Noah Smith]

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)?



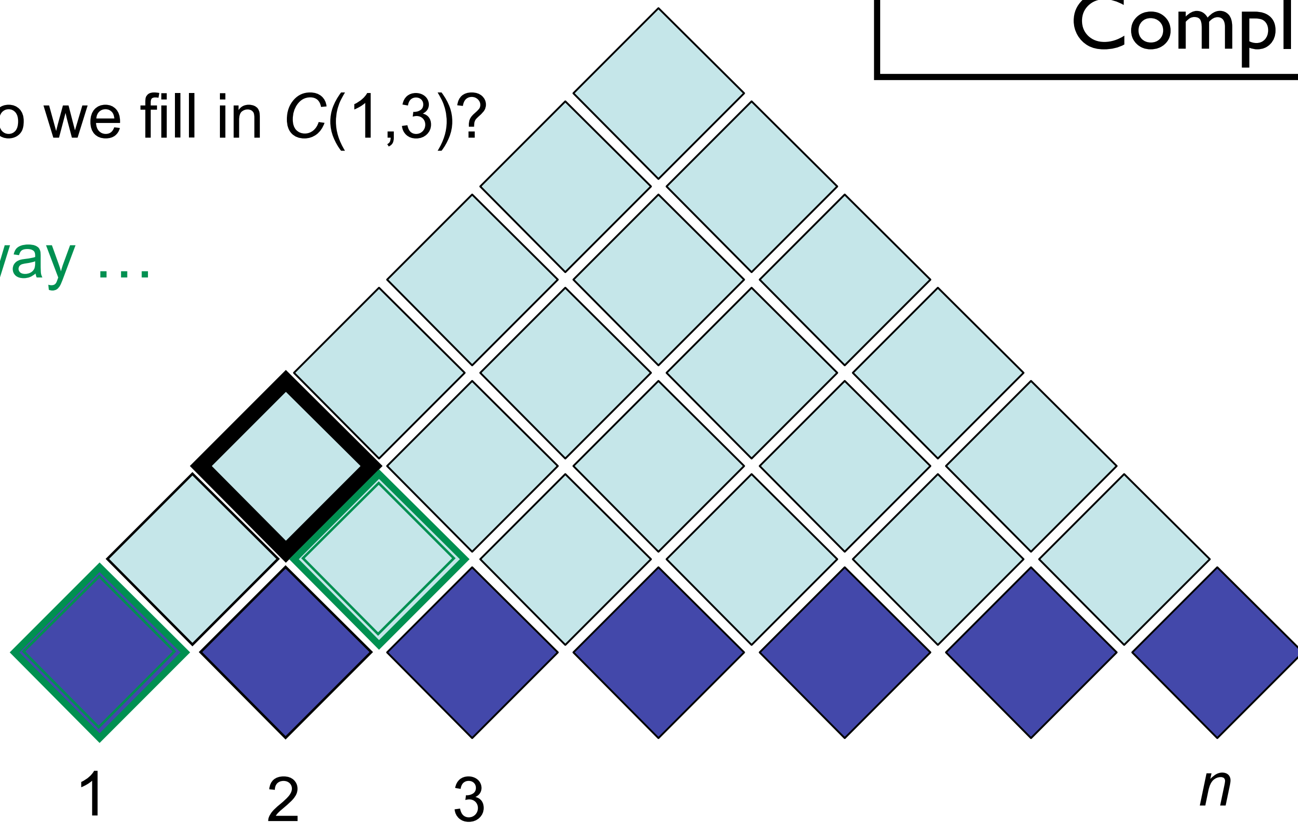
[Example from Noah Smith]

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 …



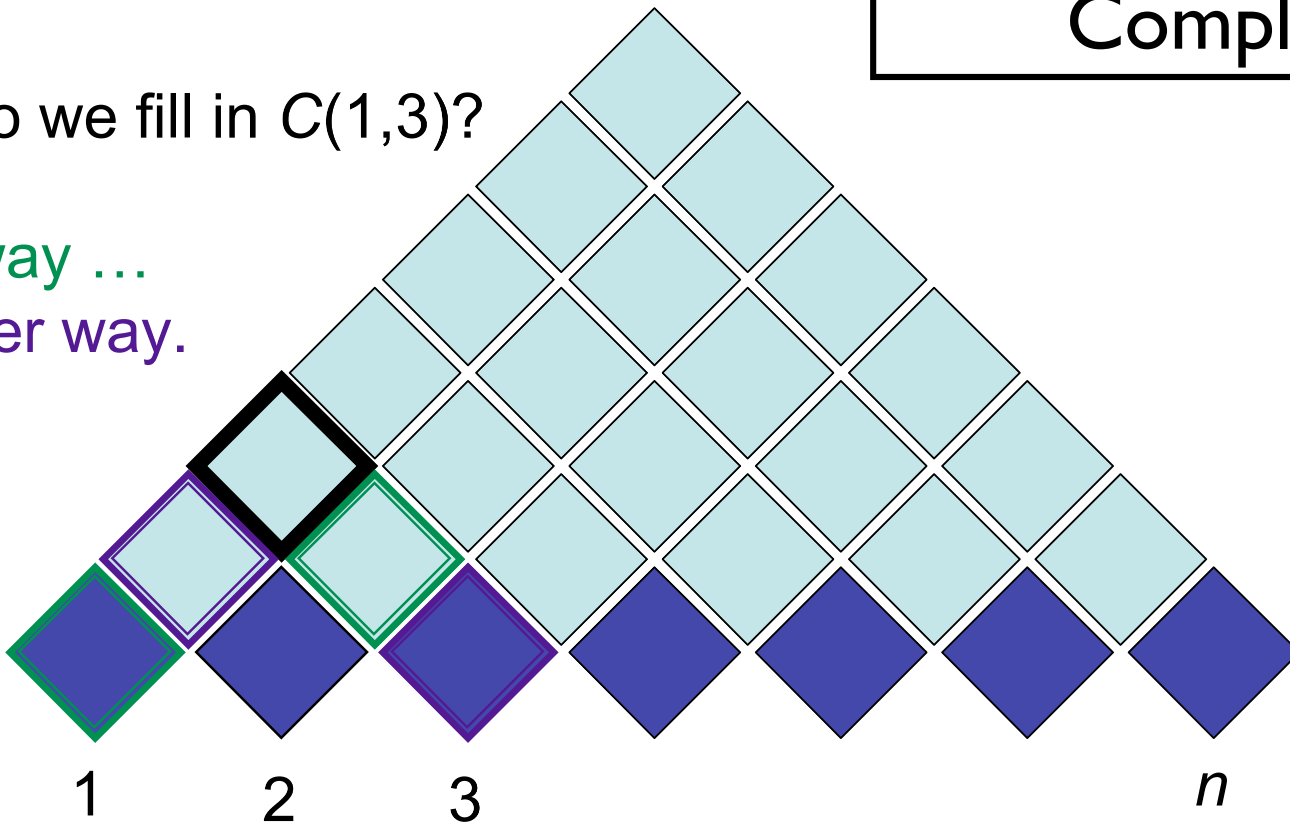
[Example from Noah Smith]

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.

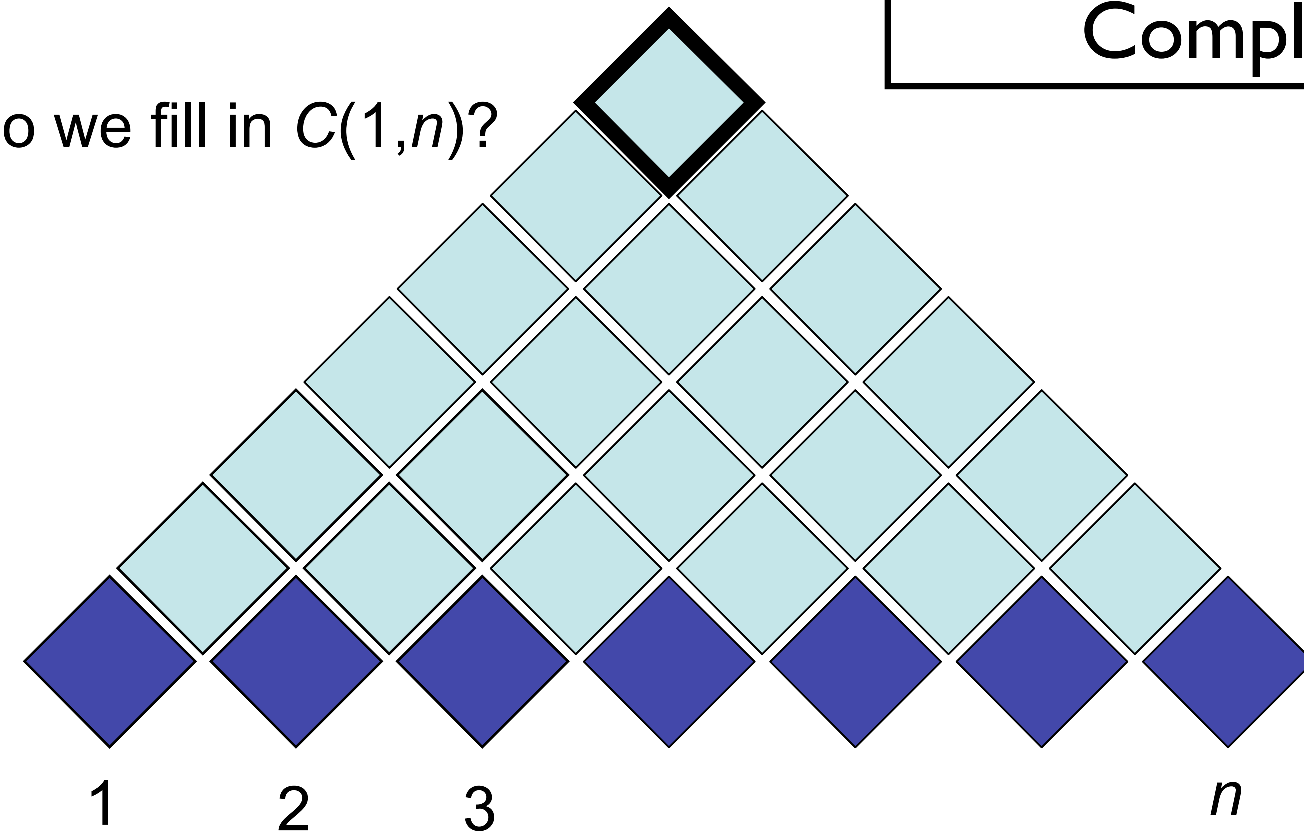


[Example from Noah Smith]

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 $C(1,n)$?



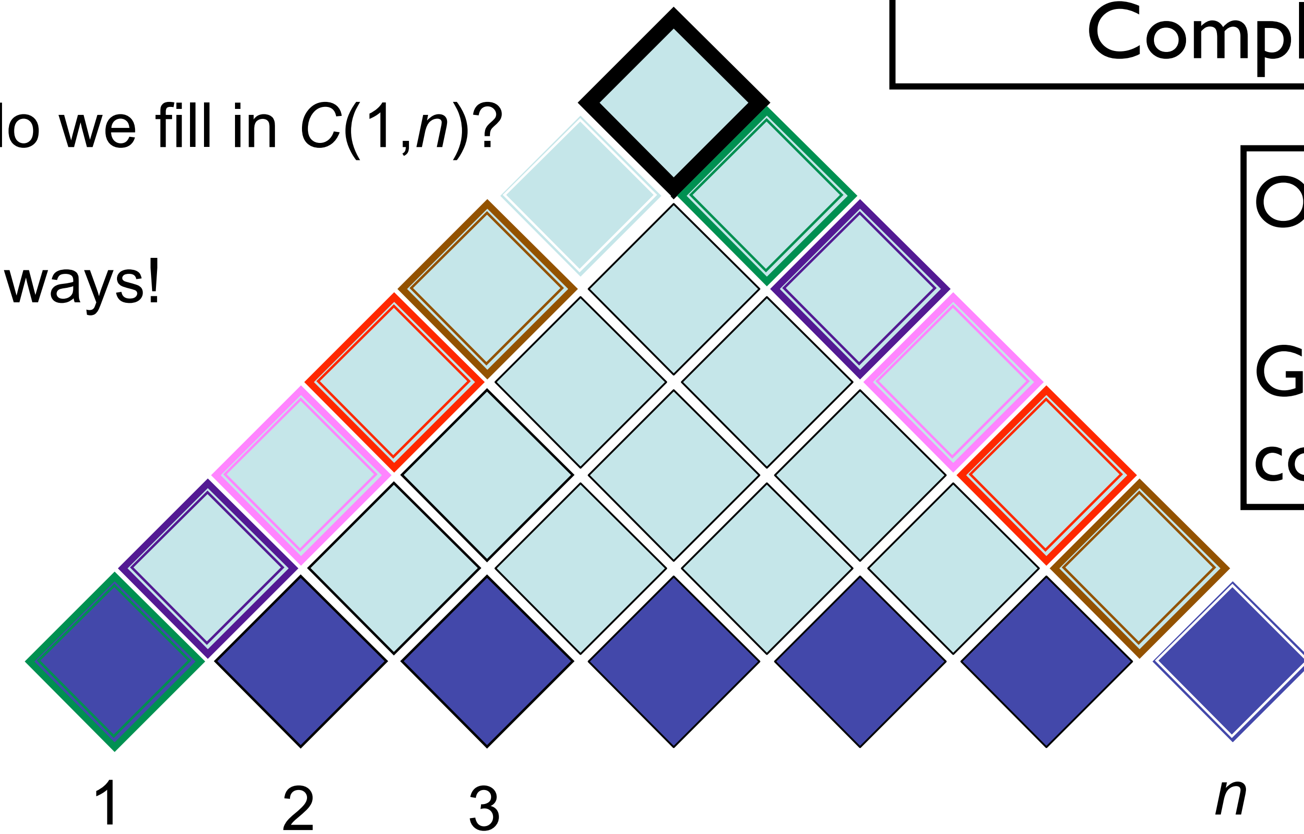
[Example from Noah Smith]

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

[Example from Noah Smith]

Probabilistic CFGs

$S \rightarrow NP\ VP$	[.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
- (How to learn? Fully supervised with a treebank...)

[J&M textbook]

Penn Treebank

```
( (S
    (NP-SBJ (NNP General) (NNP Electric) (NNP Co.) )
    (VP (VBD said)
      (SBAR (-NONE- 0)
        (S
          (NP-SBJ (PRP it) )
          (VP (VBD signed)
            (NP
              (NP (DT a) (NN contract) )
              (PP (-NONE- *ICH*-3) ))
            (PP (IN with)
              (NP
                (NP (DT the) (NNS developers) )
                (PP (IN of)
                  (NP (DT the) (NNP Ocean) (NNP State) (NNP Power) (NN project) ))))
            (PP-3 (IN for)
              (NP
                (NP (DT the) (JJ second) (NN phase) )
                (PP (IN of)
                  (NP
                    (NP (DT an) (JJ independent)
                      (ADJP
                        (QP ($ $) (CD 400) (CD million) )
                        (-NONE- *U*) )
                      (NN power) (NN plant) )
                    (, ,)
                    (SBAR
                      (WHNP-2 (WDT which) )
                      (S
                        (NP-SBJ-1 (-NONE- *T*-2) )
                        (VP (VBZ is)
                          (VP (VBG being)
                            (VP (VBN built)
                              (NP (-NONE- *-1) )
                              (PP-LOC (IN in)
                                (NP
                                  (NP (NNP Burrillville) )
                                  (, ,)
                                  (NP (NNP R.I) ))))))))))))))))
```

PCFG as LM

Is a PCFG a *good* LM? Yes...

Is a PCFG a *good* LM? No...

(P)CFG model, (P)CKY algorithm

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

- Probabilistic CKY: given PCFG and sentence w
  - Likelihood of sentence P(w)
  - Most probable parse ("Viterbi parse")
    $\text{argmax}_y P(y \mid w) = \text{argmax}_y P(y, w)$

- Parsing model accuracy: lots of ambiguity!!
  - PCFGs lack lexical information to resolve ambiguities (sneak in world knowledge?)
  - Modern constituent parsers: enrich PCFG with lexical information and fine-grained nonterminals
  - Modern dependency parsers: effectively the same trick
- Parsers' computational efficiency
  - Grammar constant; pruning & heuristic search
  - $O(N^3)$ for CKY (ok? depends...)
  - O(N) left-to-right incremental algorithms
- What was the syntactic training data?