Lecture: Syntax Part 2

CS 585, Fall 2016

Introduction to Natural Language Processing

http://people.cs.umass.edu/~brenocon/inlp2016

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:

"Grammar": *one* NT expands to >=1 NT
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

Lexicon: NT expands to a terminal

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* | ...

[only one token. ignore "L A"]

Constituent Parse Trees

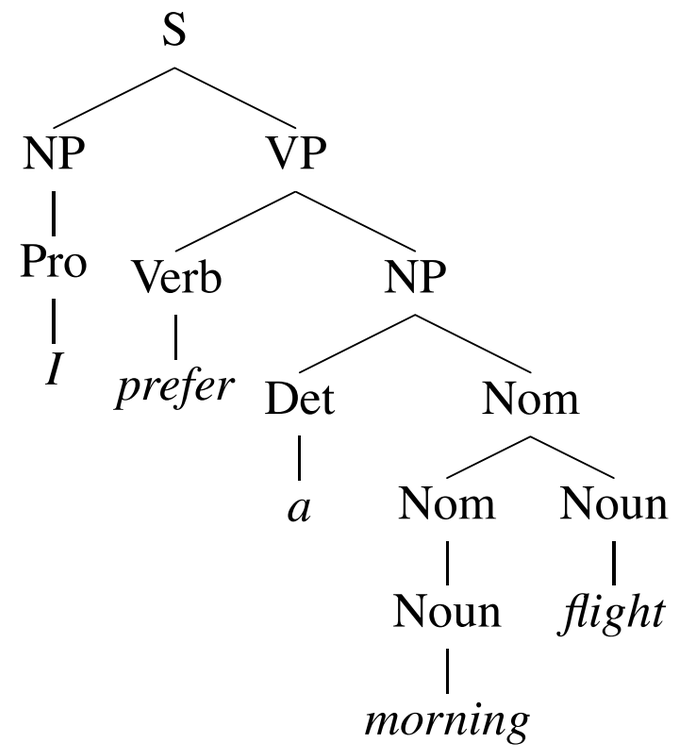


Figure 12.4 The parse tree for "I prefer a morning flight" according to grammar $\mathscr{L}_0$.

Representations:

Bracket notation

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

Non-terminal positional spans

e.g. (NP, 0, 1), (VP, 1, 5), (NP, 2, 5), etc.

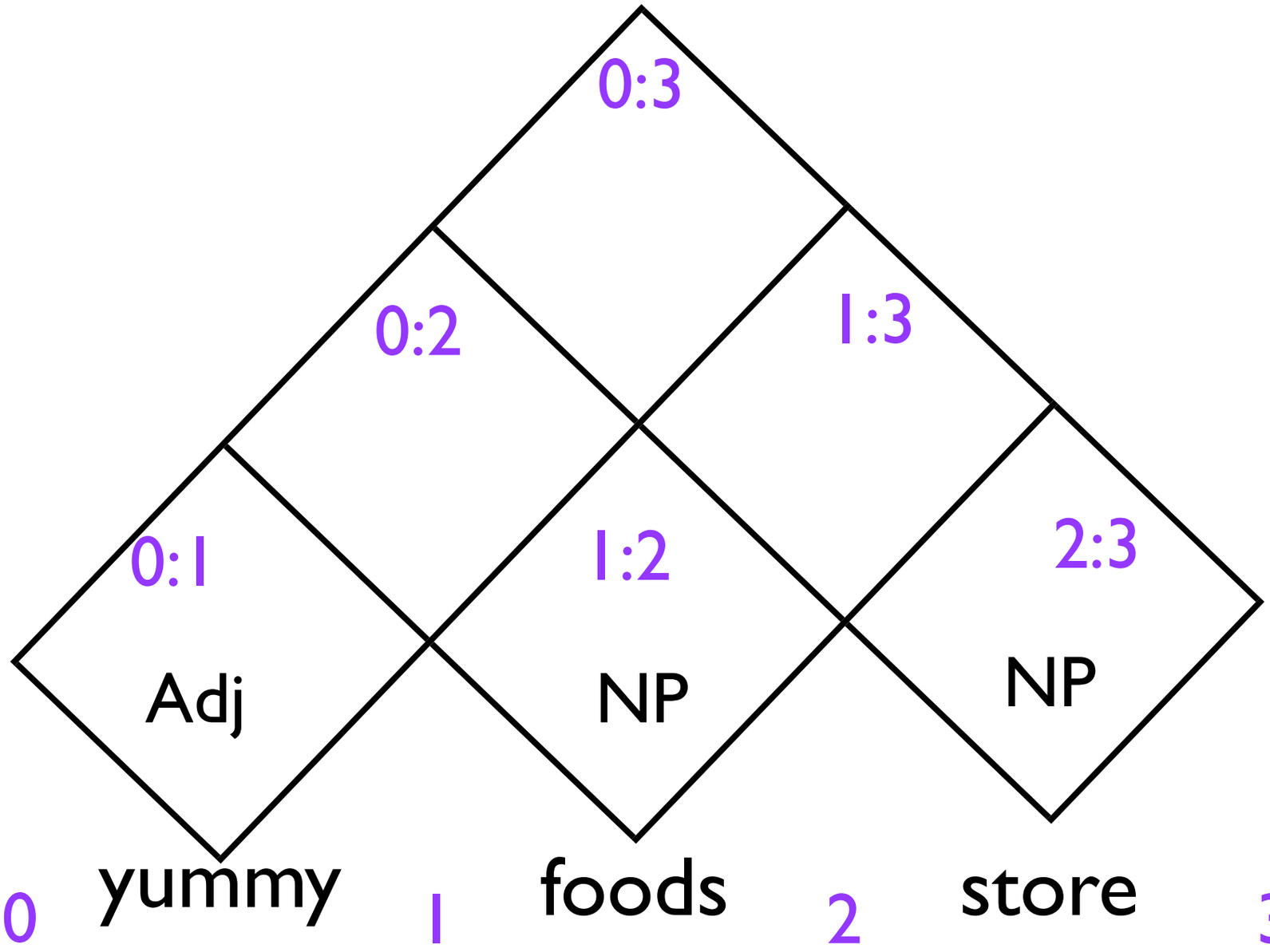
Parsing with a CFG

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

- Cocke-Kasami-Younger algorithm
  - Bottom-up dynamic programming:
    Find possible nonterminals for short spans of sentence, then possible combinations for higher spans
  - Requires converting CFG to Chomsky Normal Form (a.k.a. binarization)

CKY

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



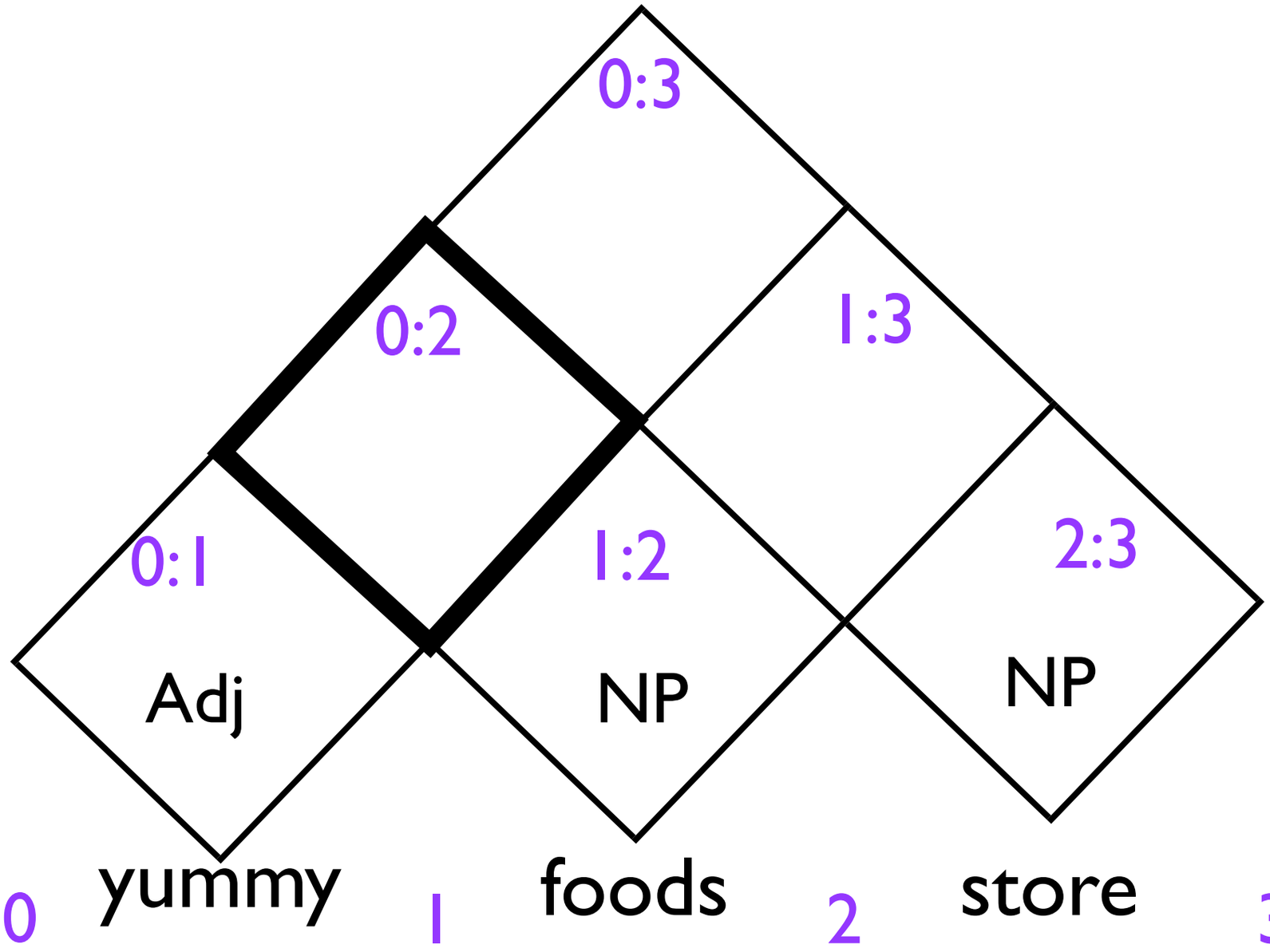
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



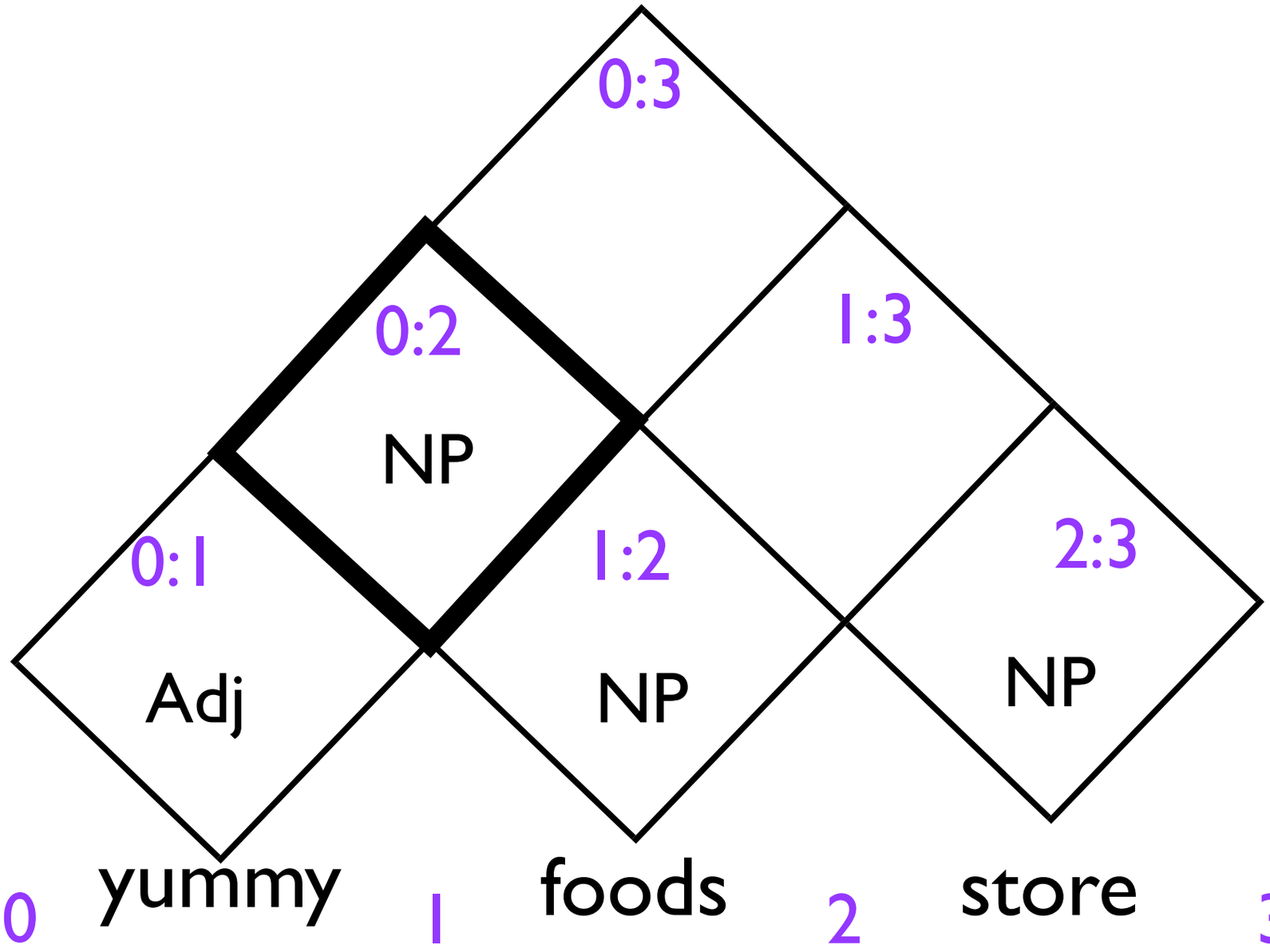
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



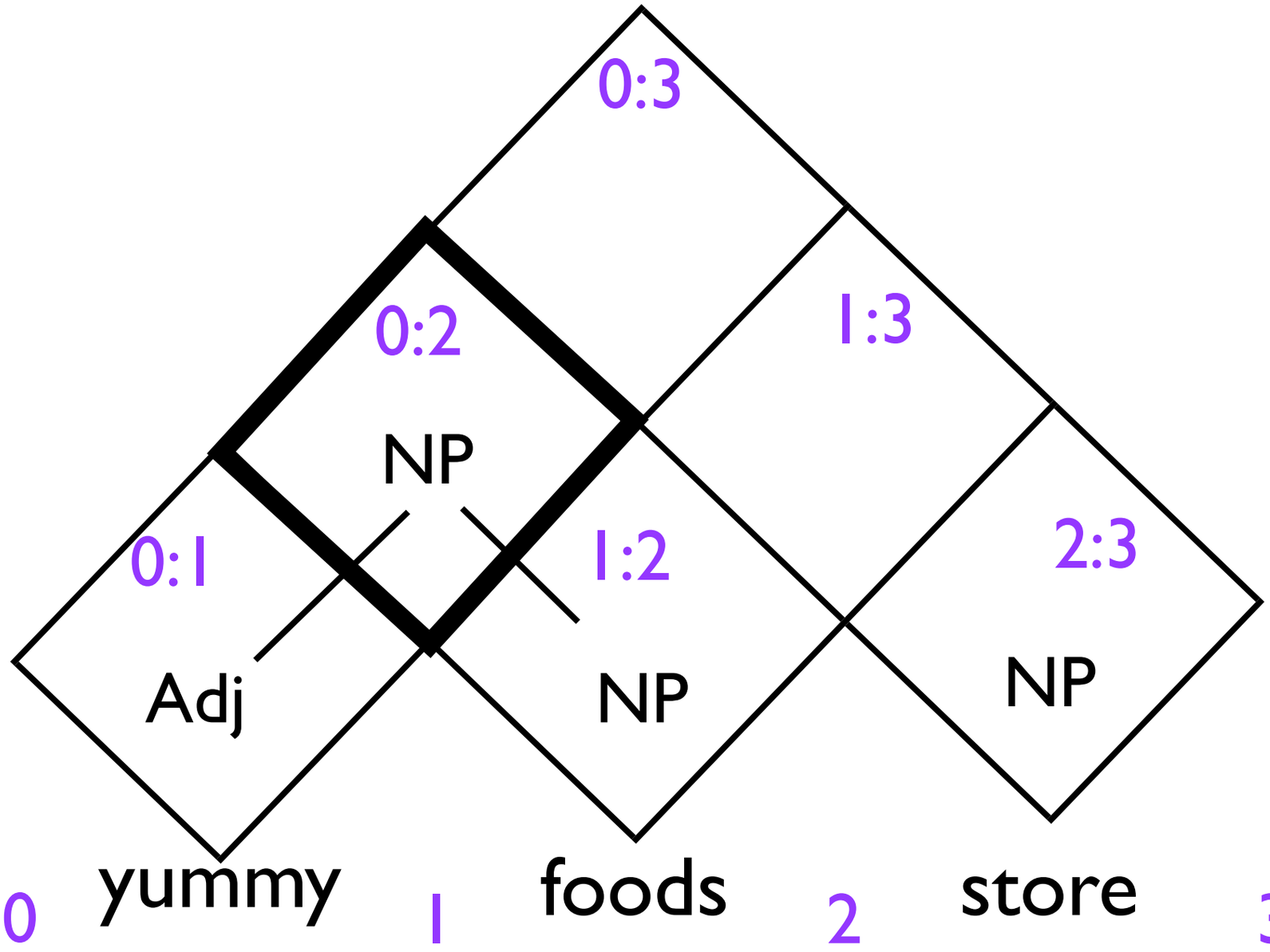
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

Recognizer: per span, record list of possible nonterminals

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

CKY

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



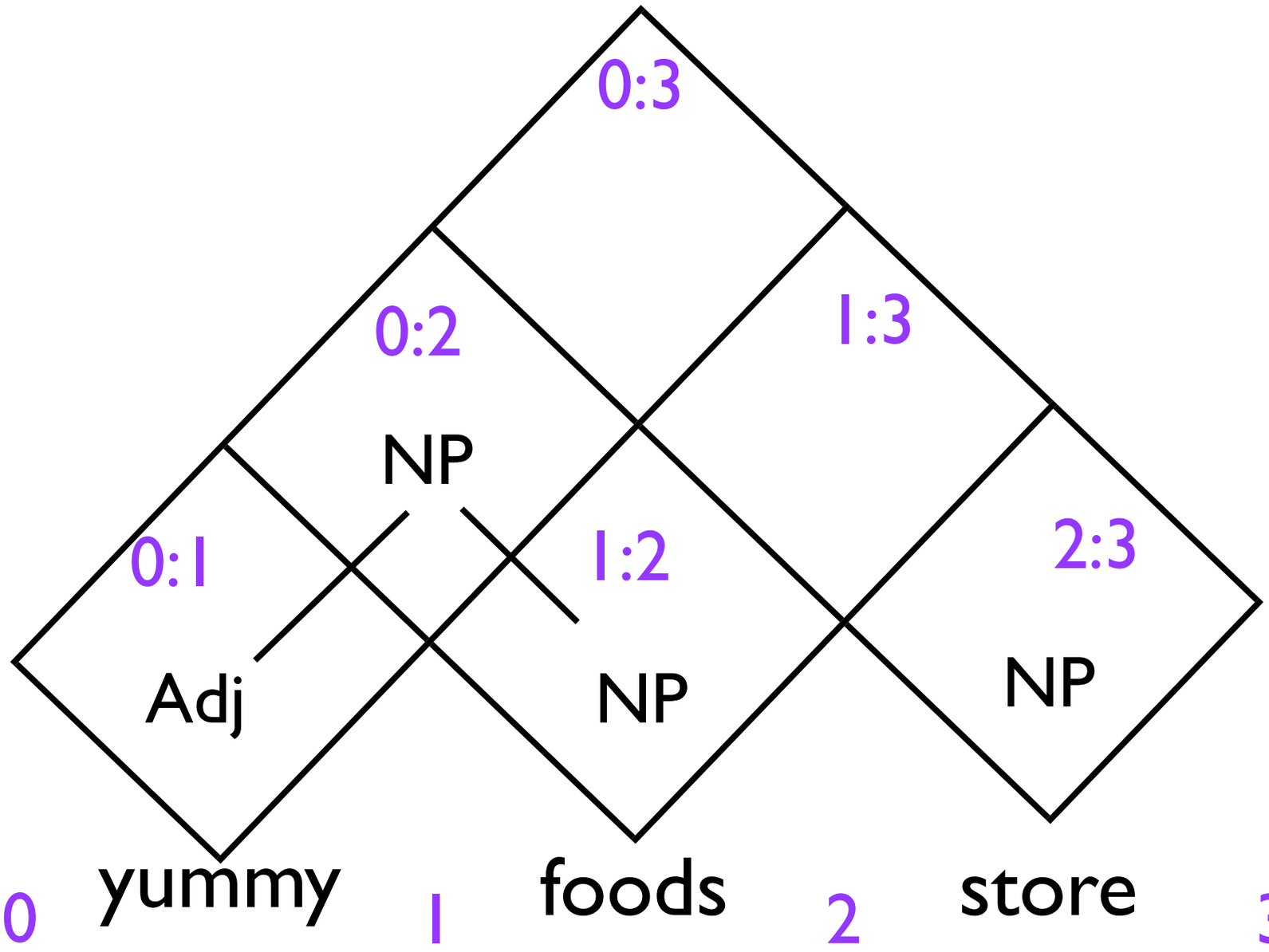
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



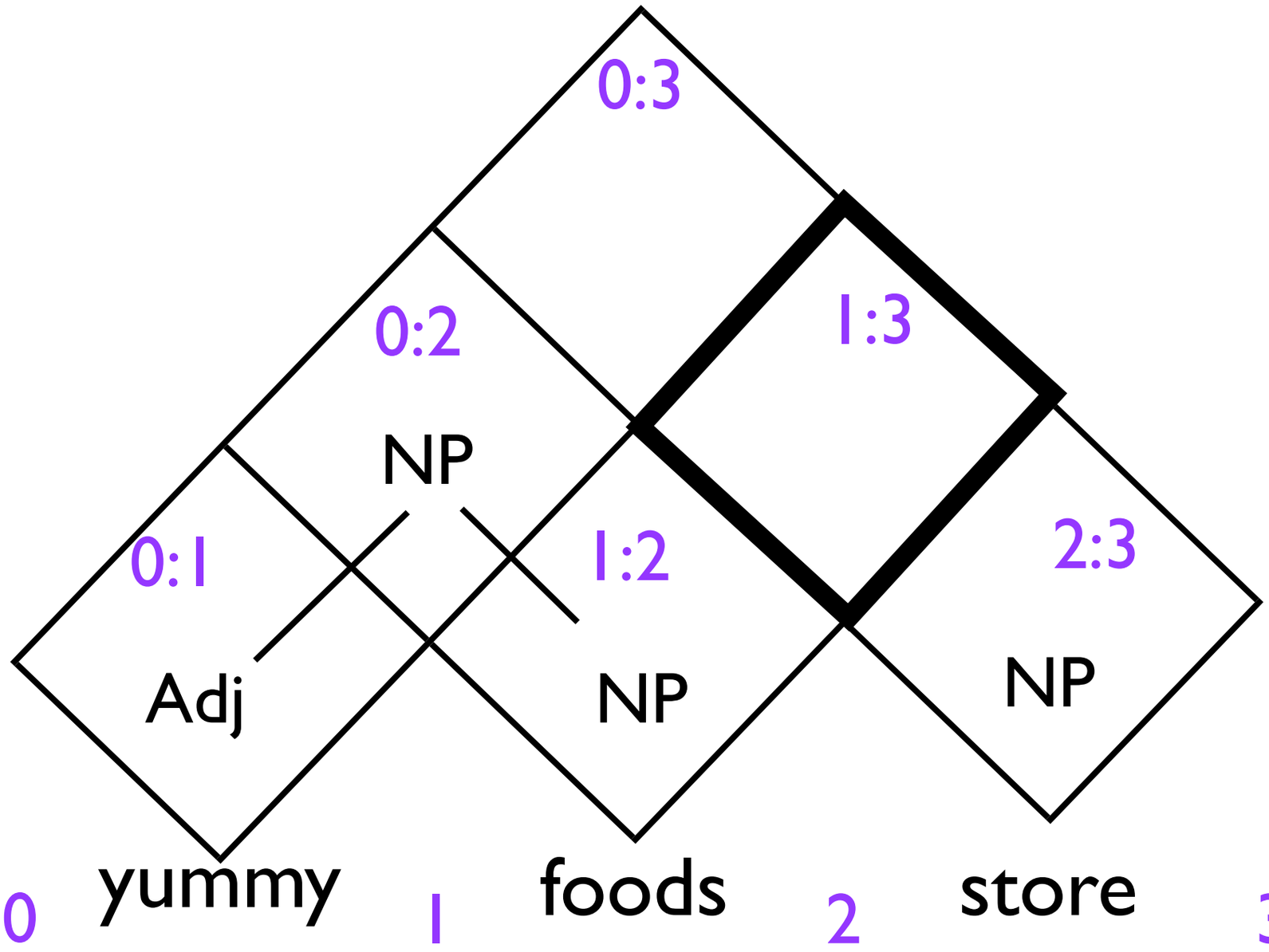
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



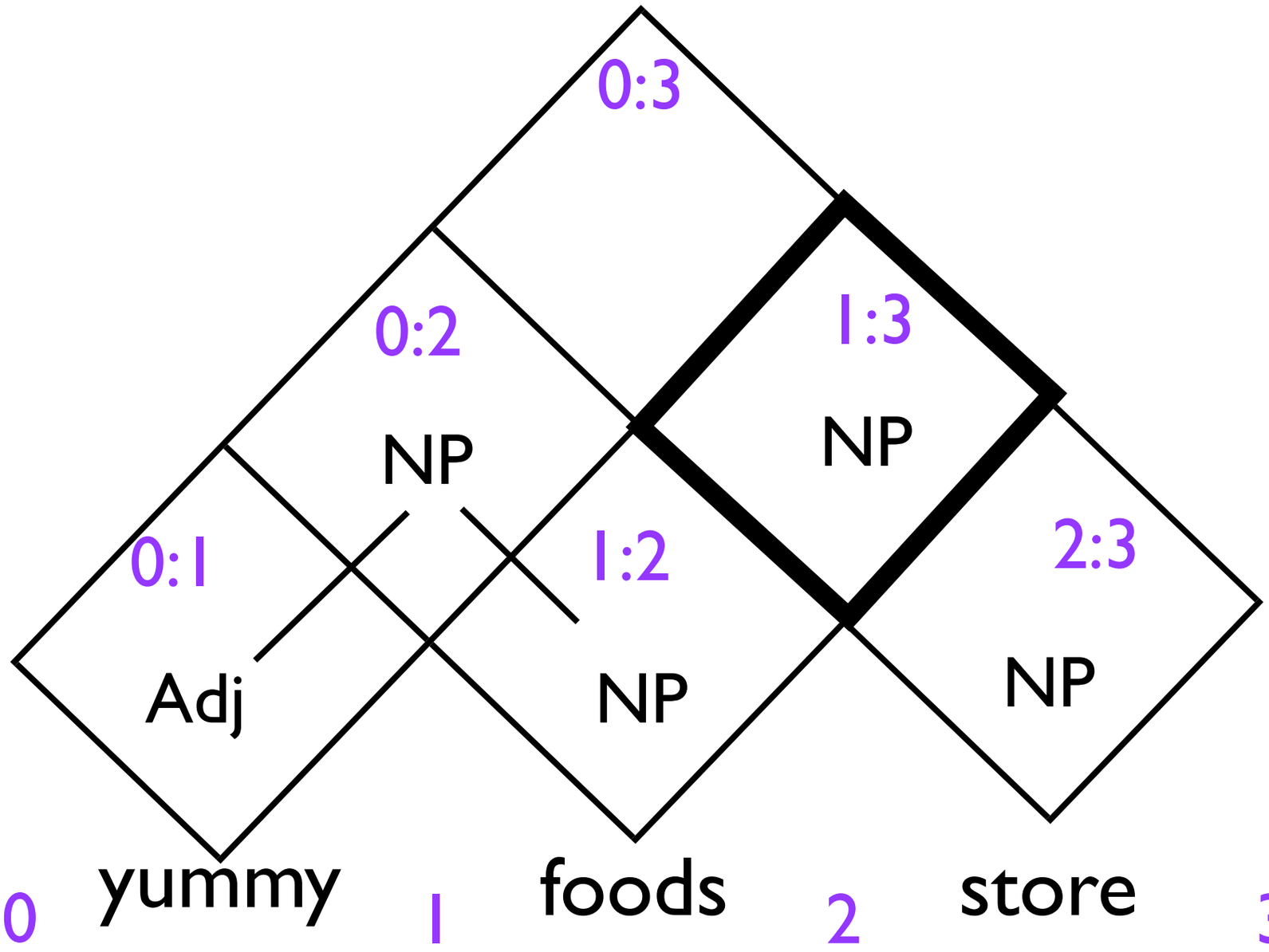
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

Recognizer: per span, record list of possible nonterminals

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

CKY

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



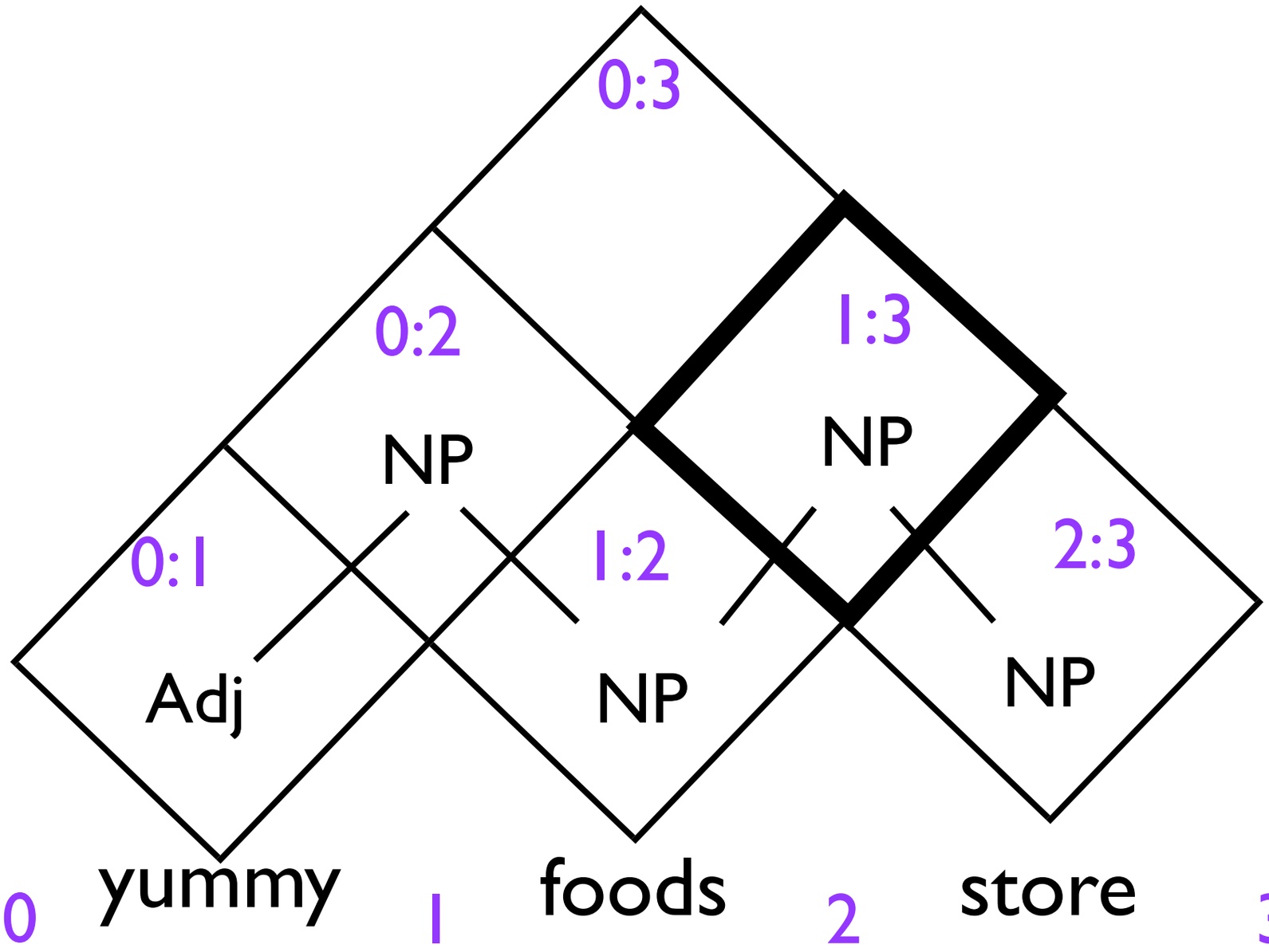
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



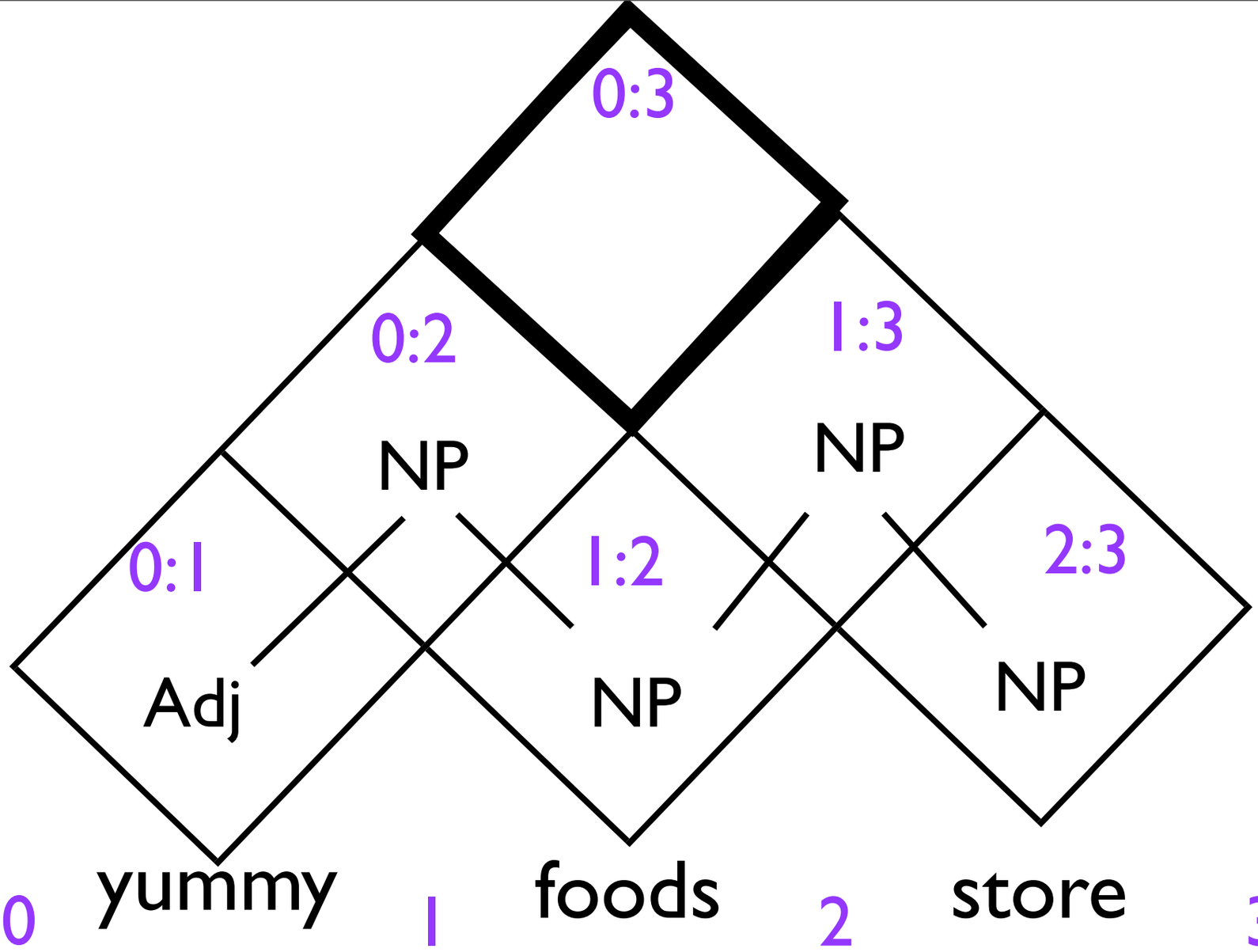
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



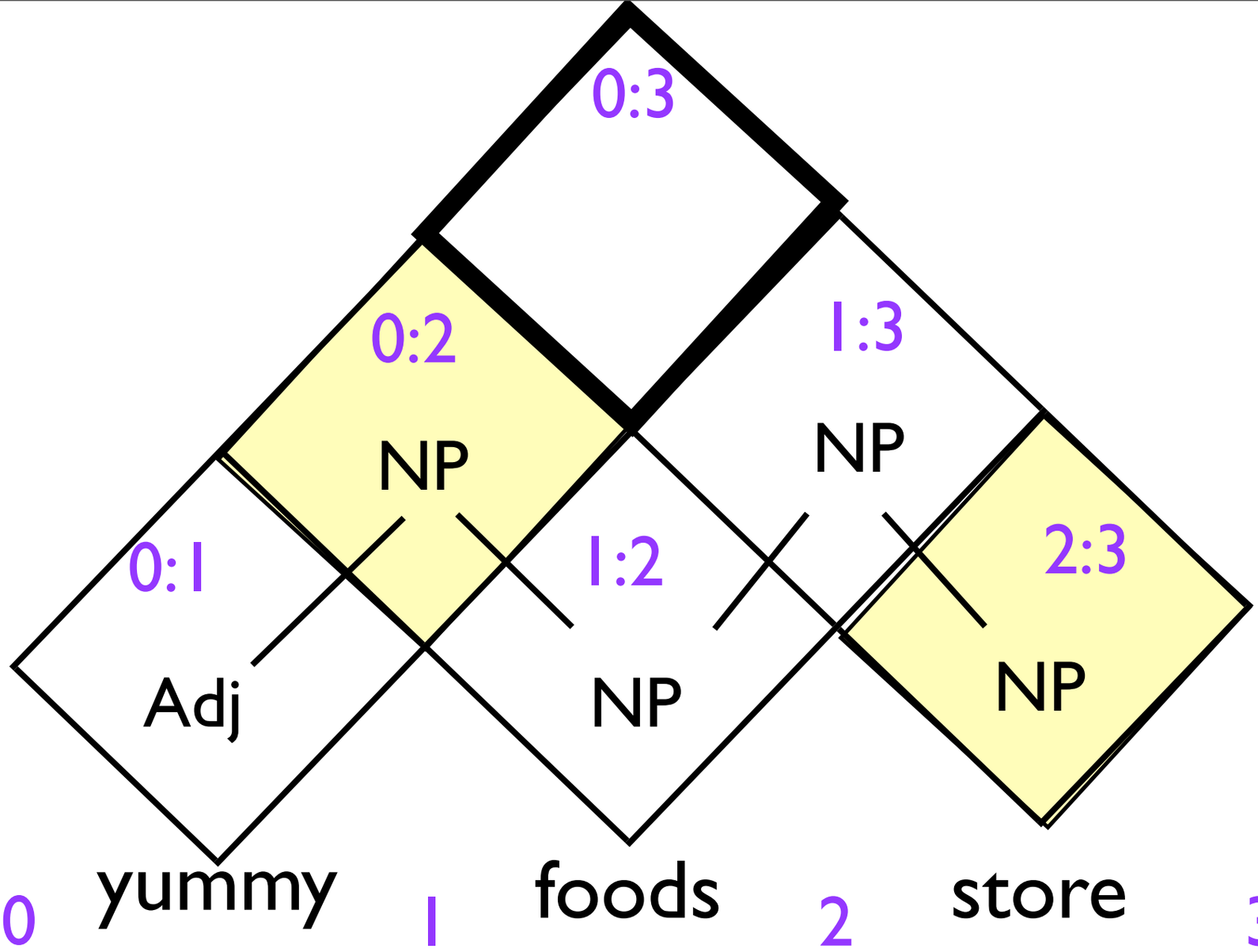
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

Recognizer: per span, record list of possible nonterminals

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

CKY

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



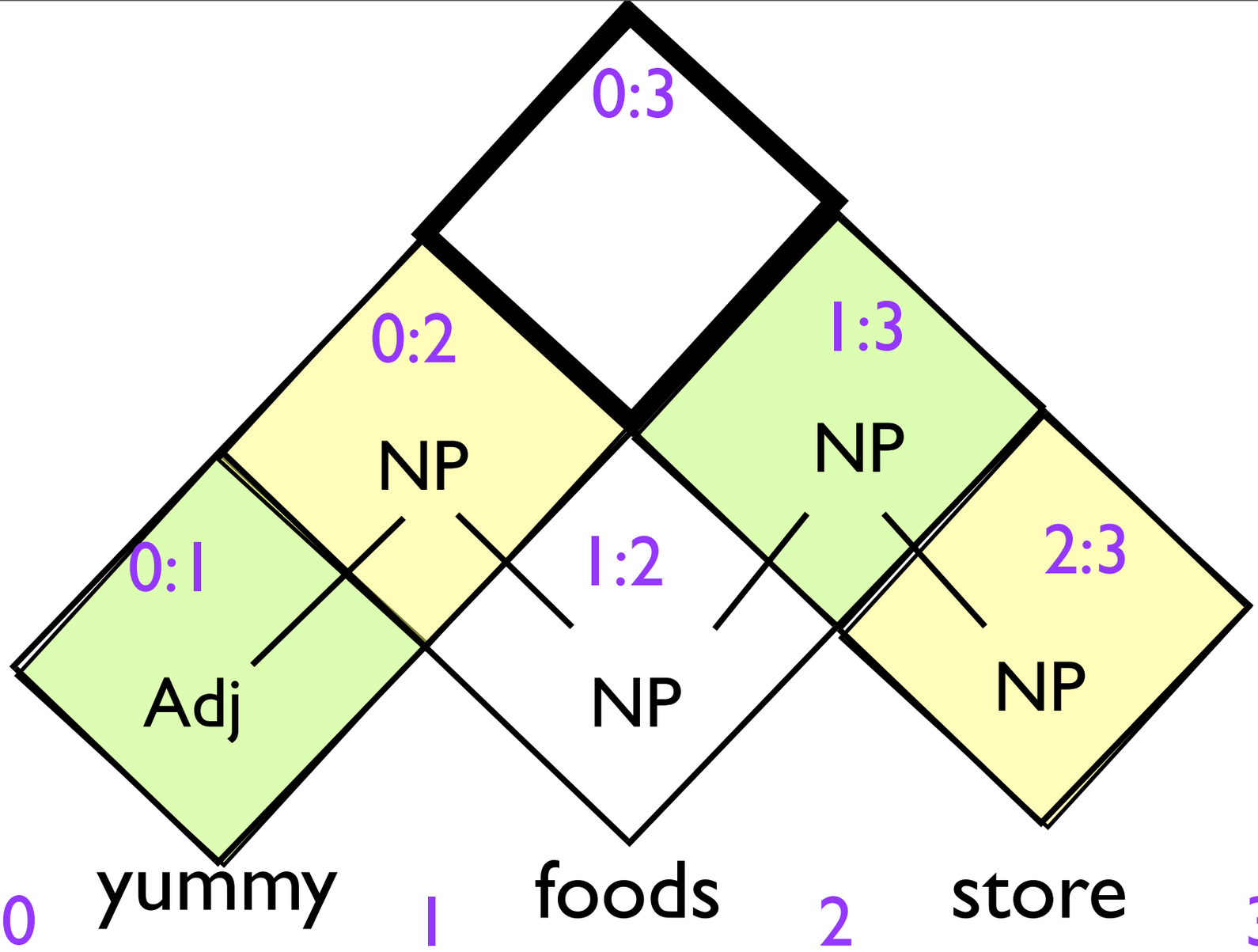
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



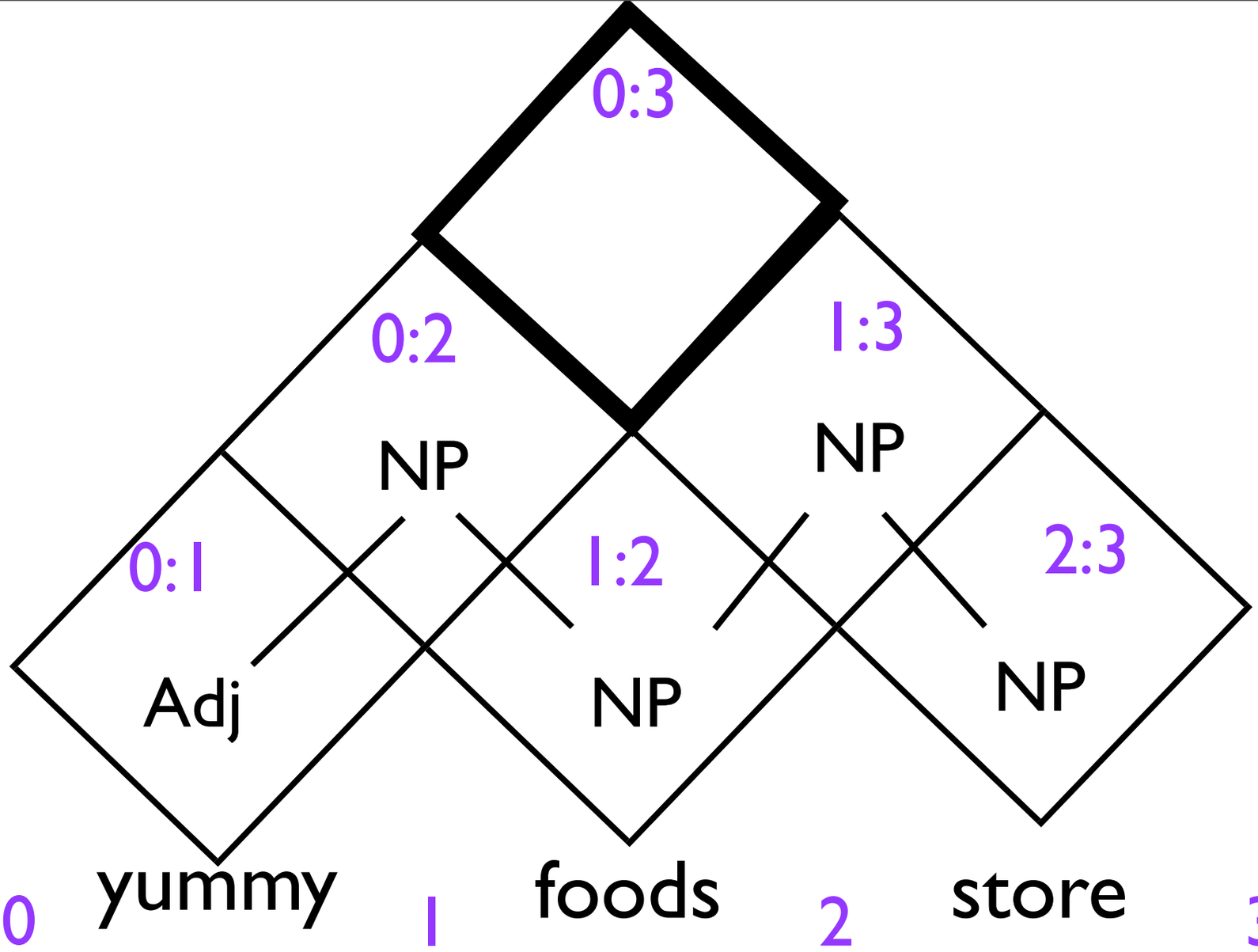
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



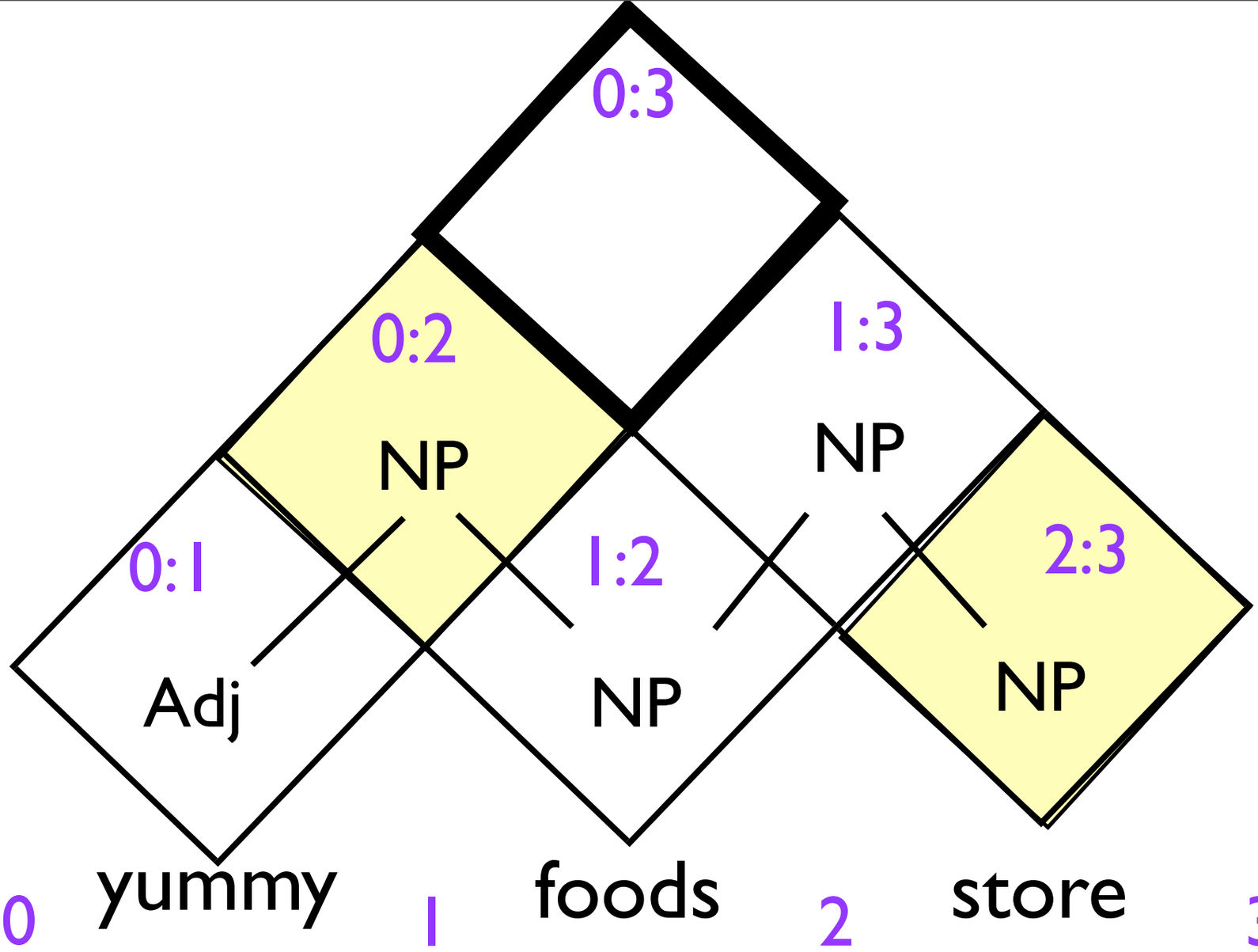
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

Recognizer: per span, record list of possible nonterminals

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

CKY

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



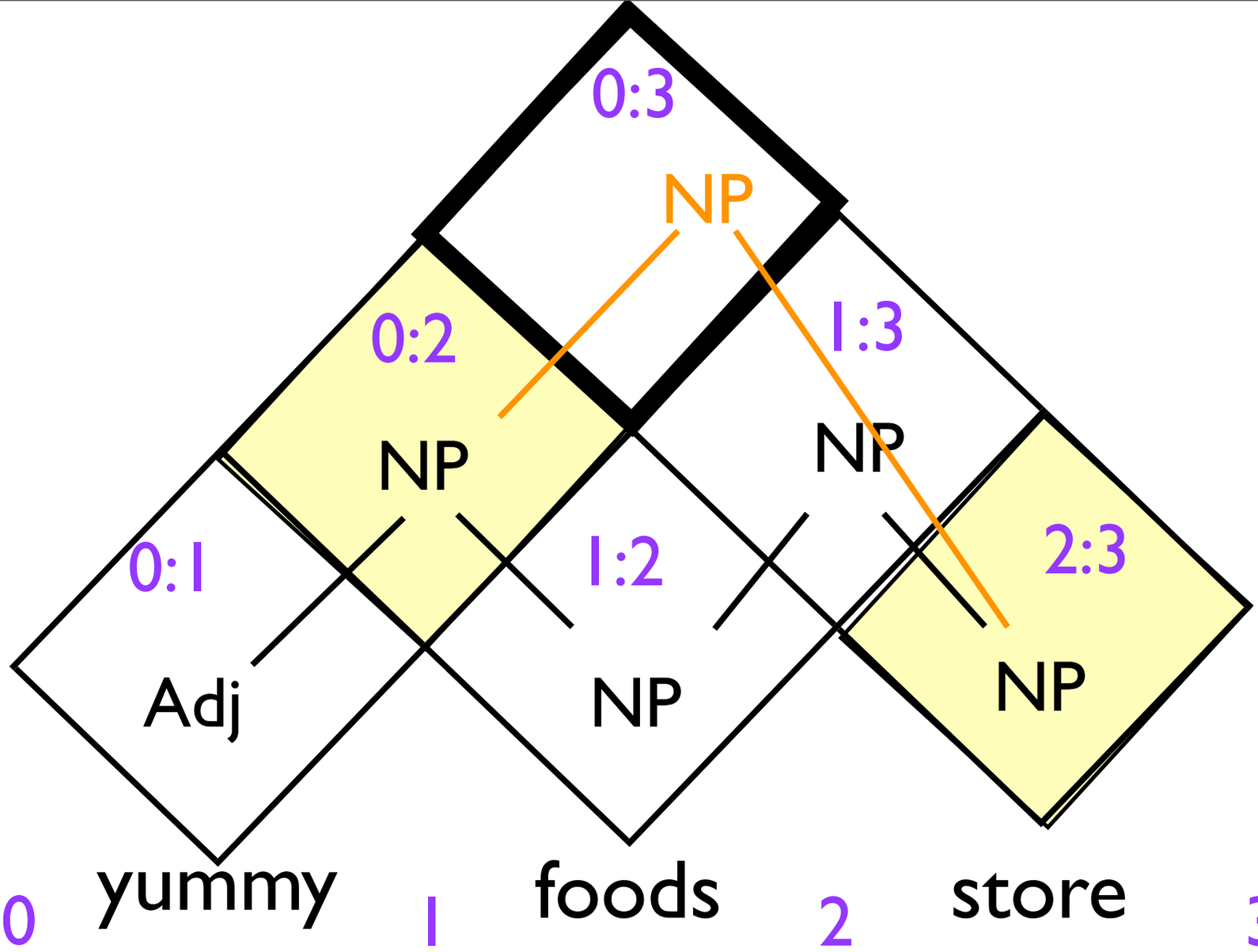
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



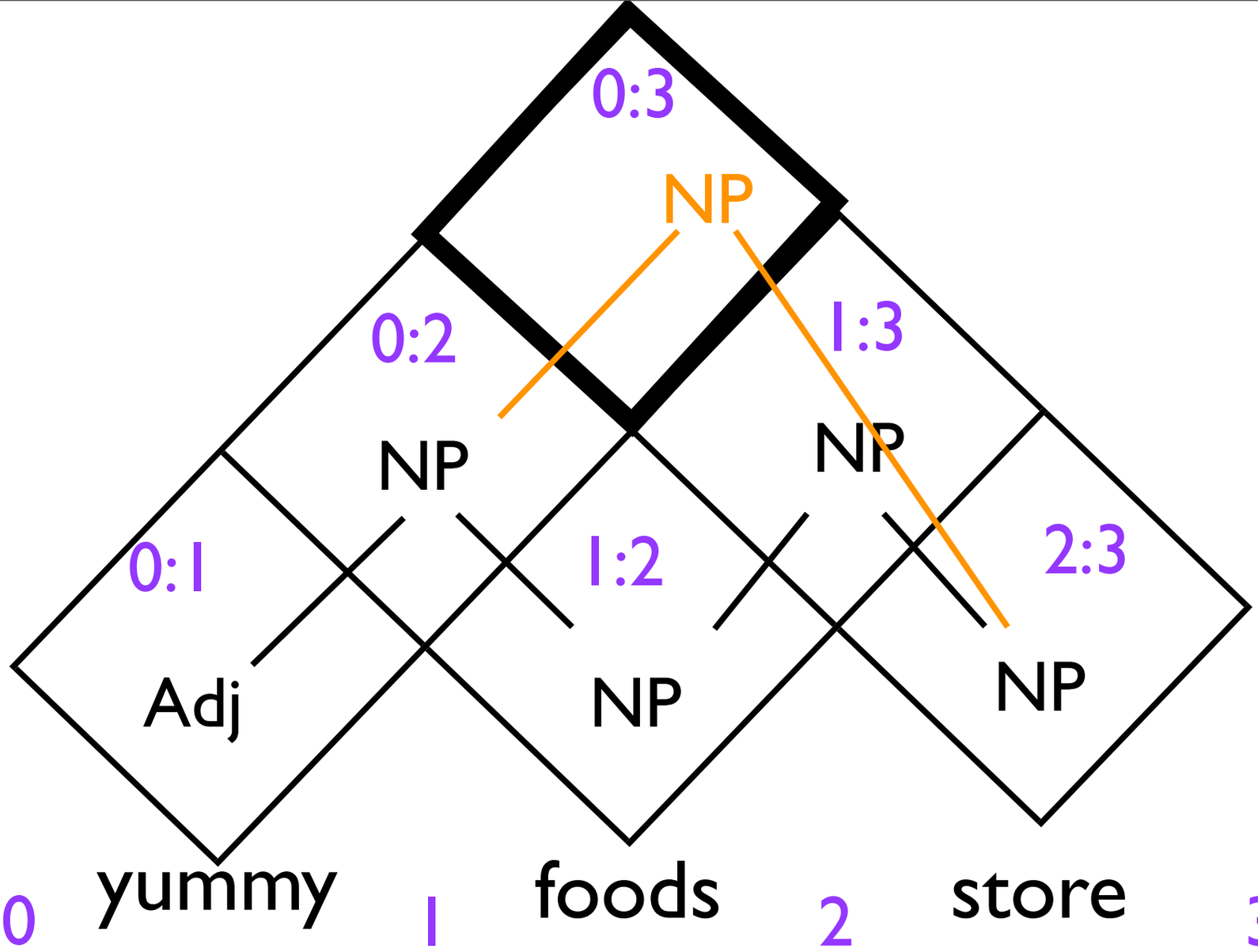
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



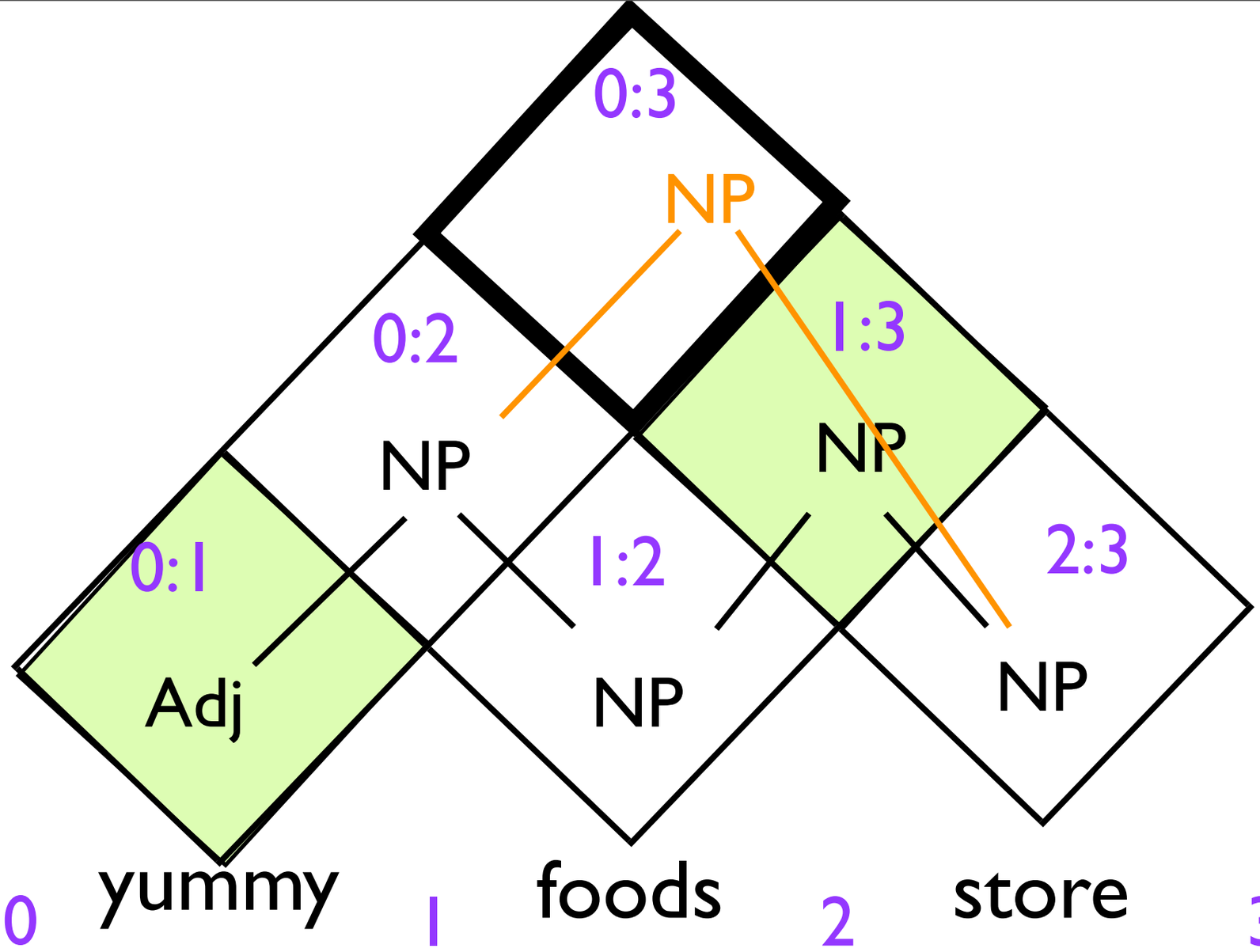
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

Recognizer: per span, record list of possible nonterminals

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

CKY

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



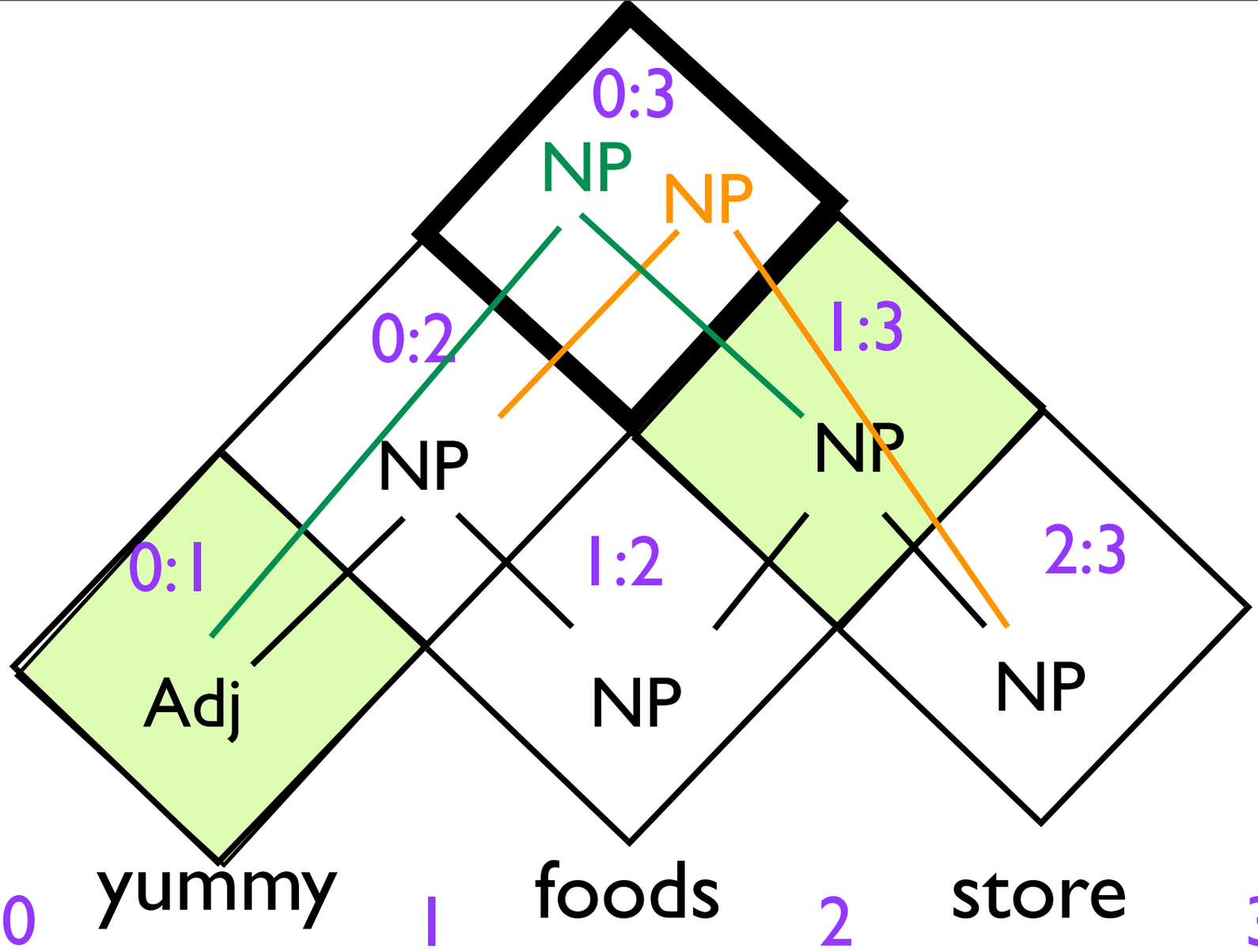
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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*)

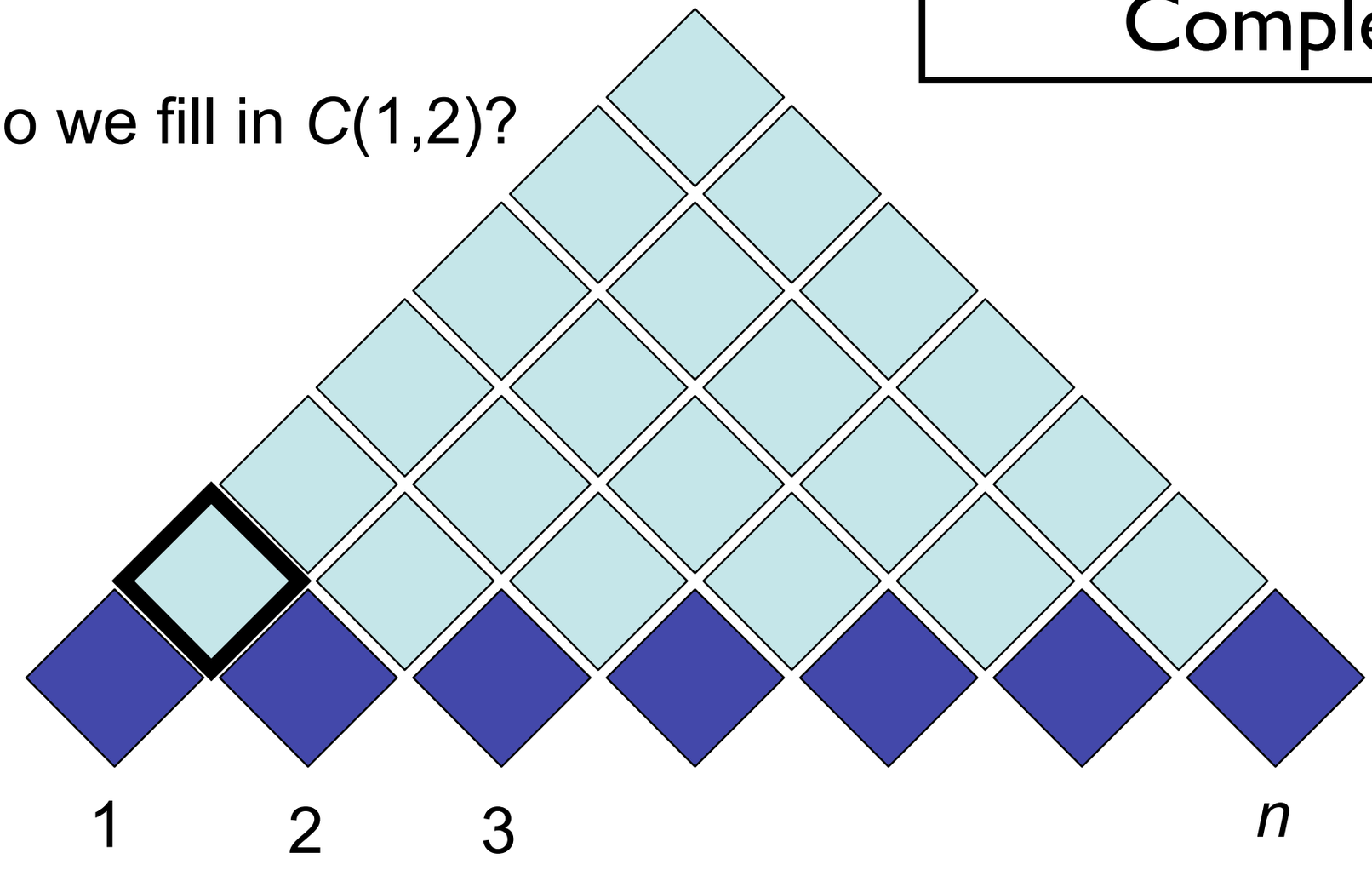
Recognizer: per span, record list of possible nonterminals

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

```
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)?

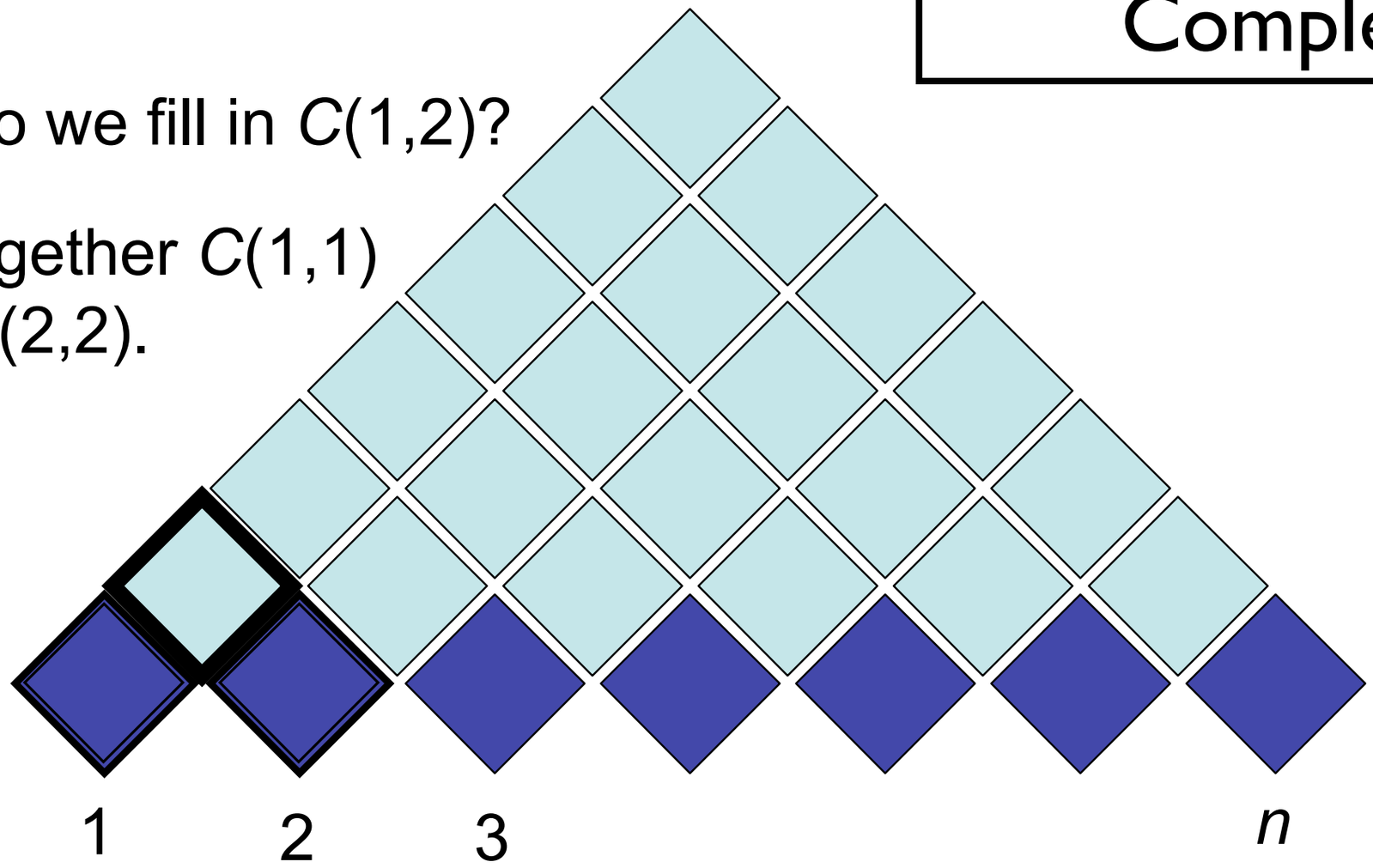


```
For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]
```

Computational Complexity ?

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

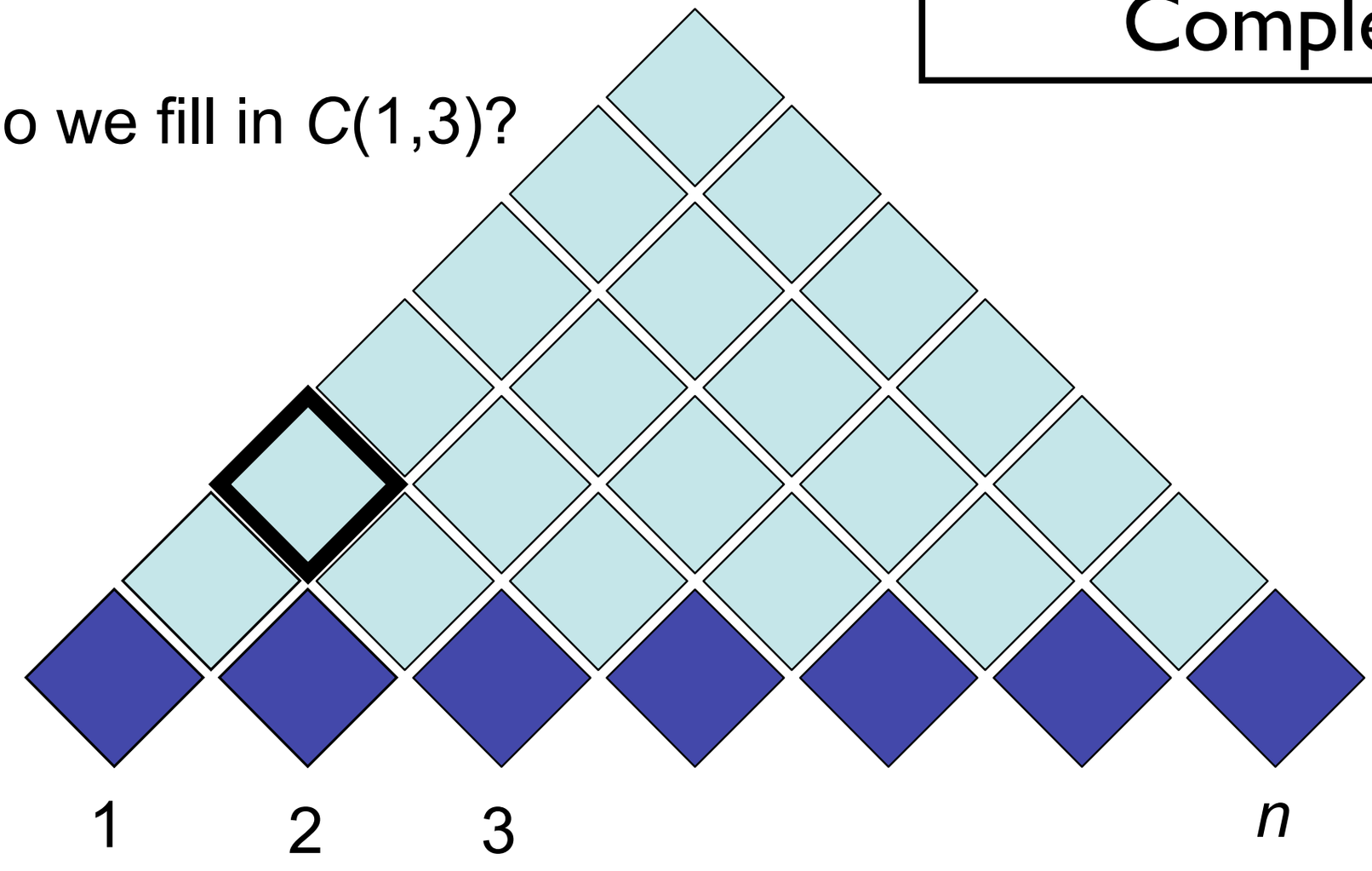
Put together *C*(1,1) and *C*(2,2).



```
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)?

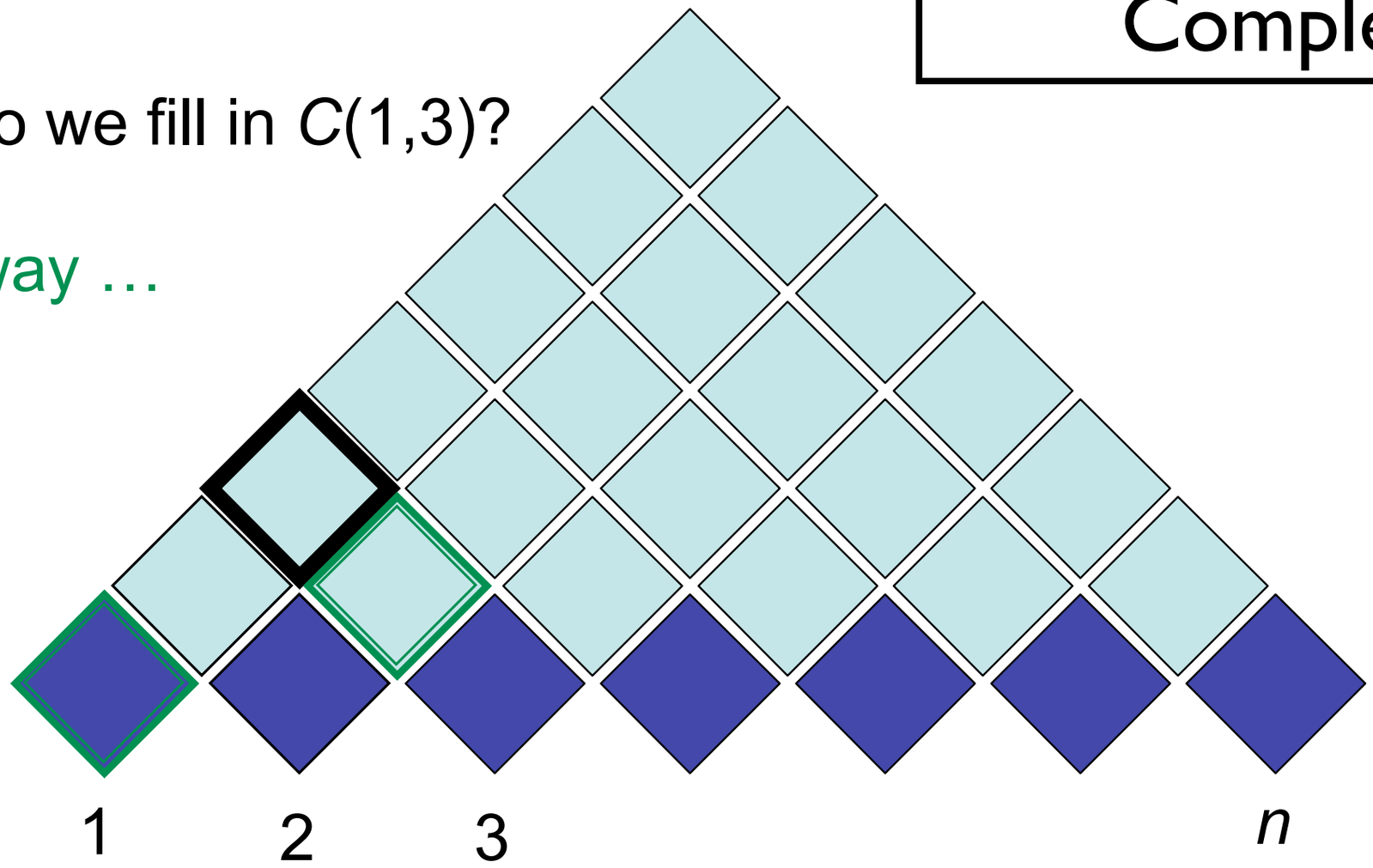


```
For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add  A to cell [i,j]
```

Computational Complexity ?

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

One way …



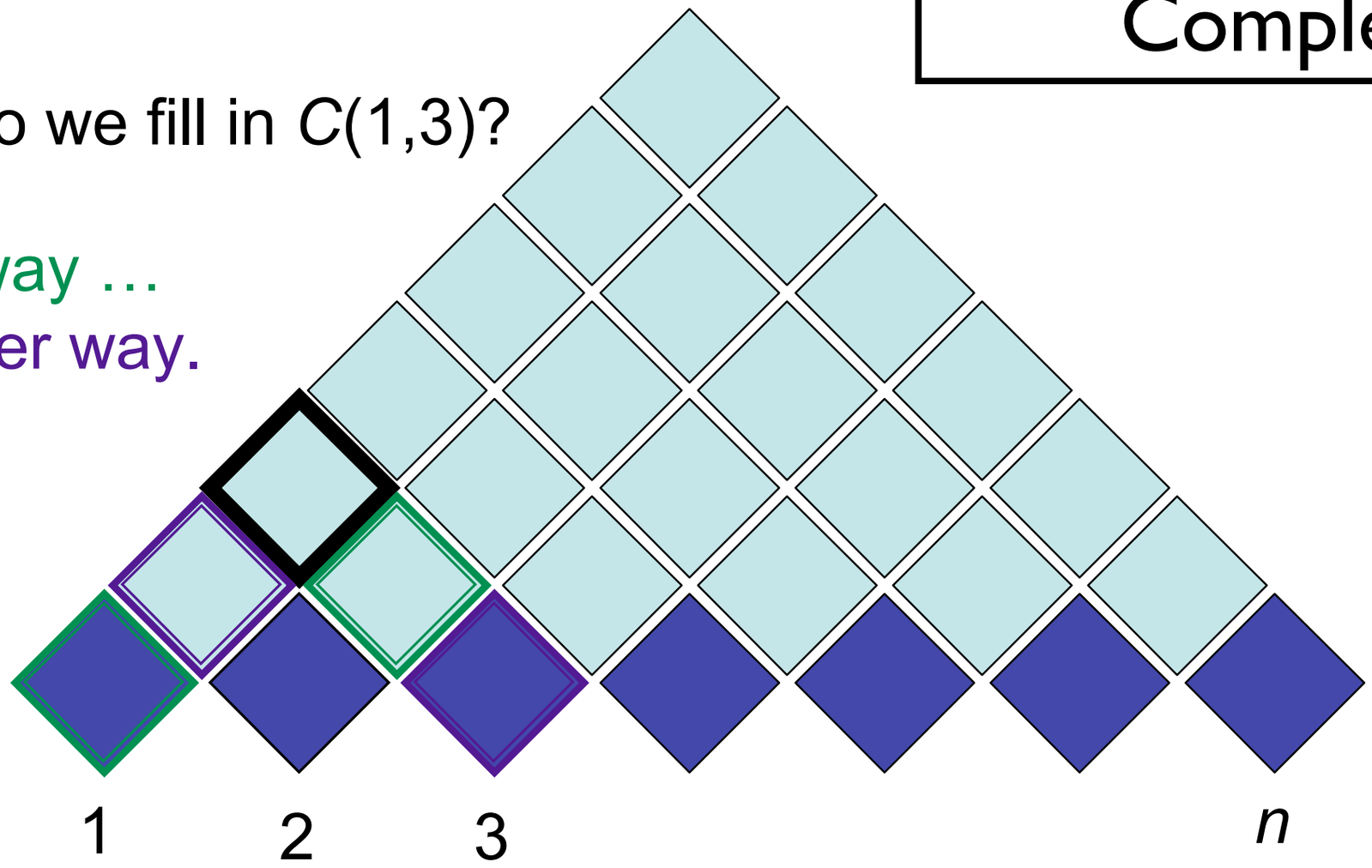
```
For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add  A to cell [i,j]
```

Computational Complexity ?

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

One way …

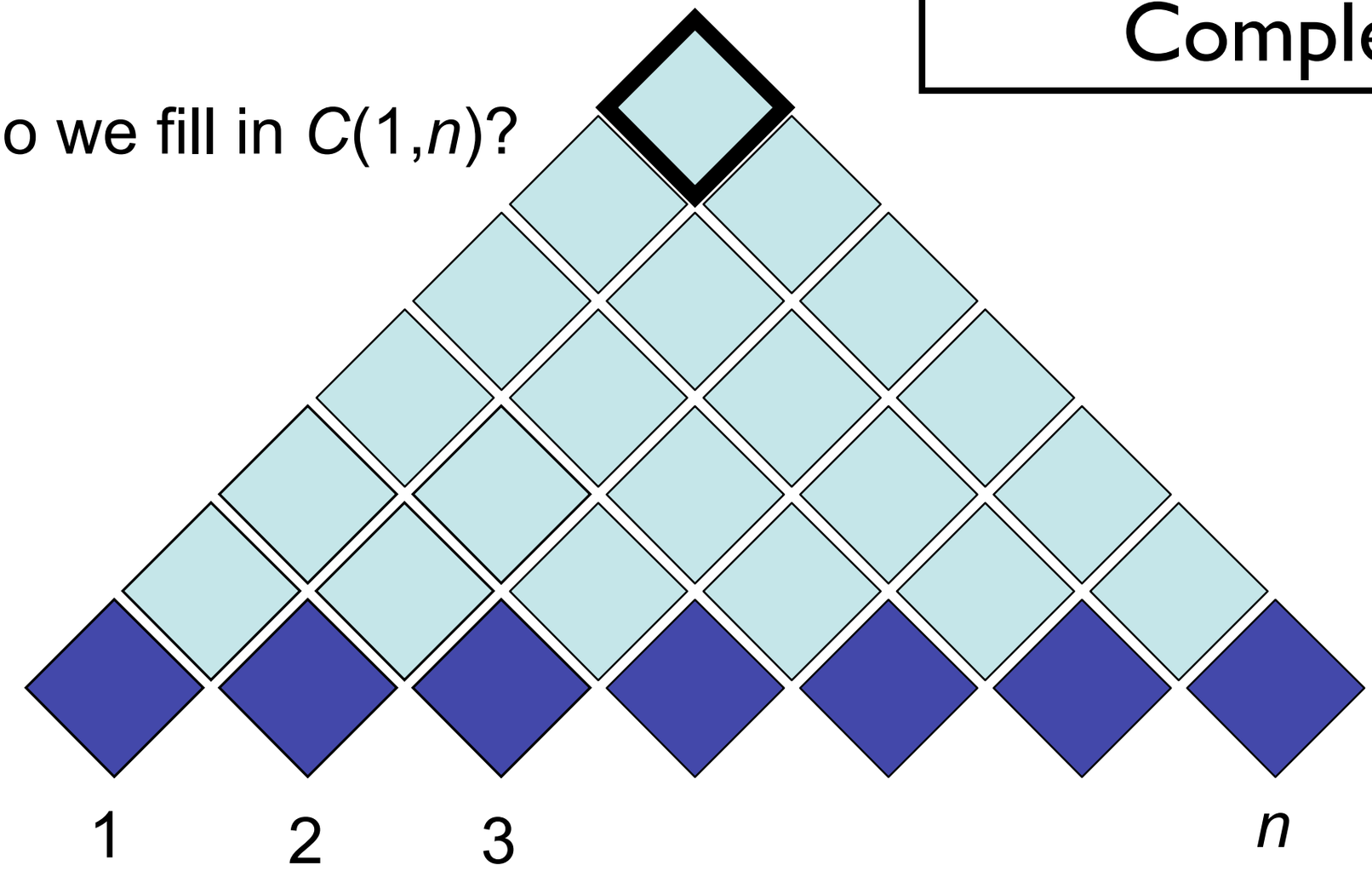
Another way.



```
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 $C(1,n)$?

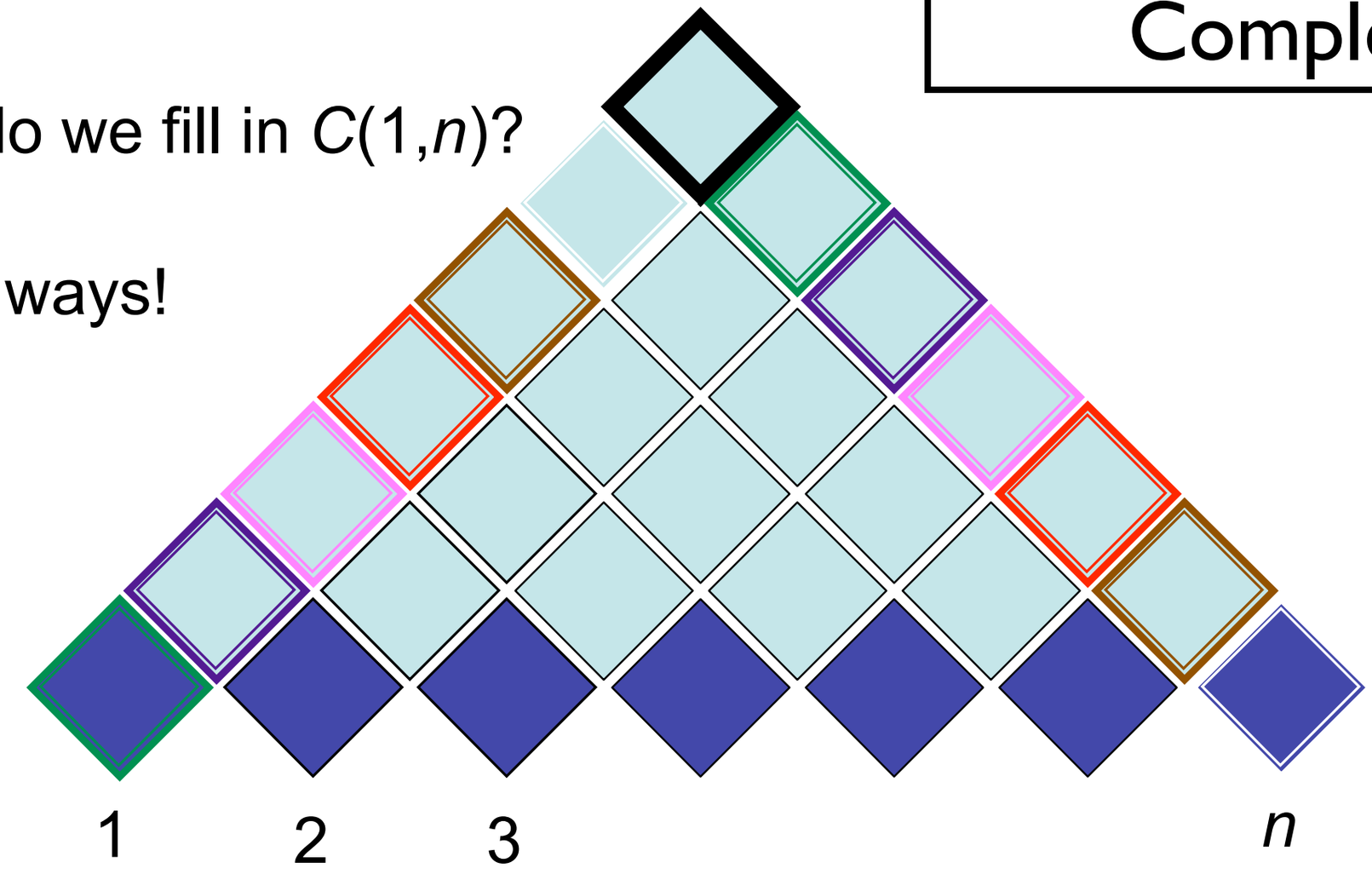


```
For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add  A to cell [i,j]
```

Computational Complexity ?

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

n - 1 ways!

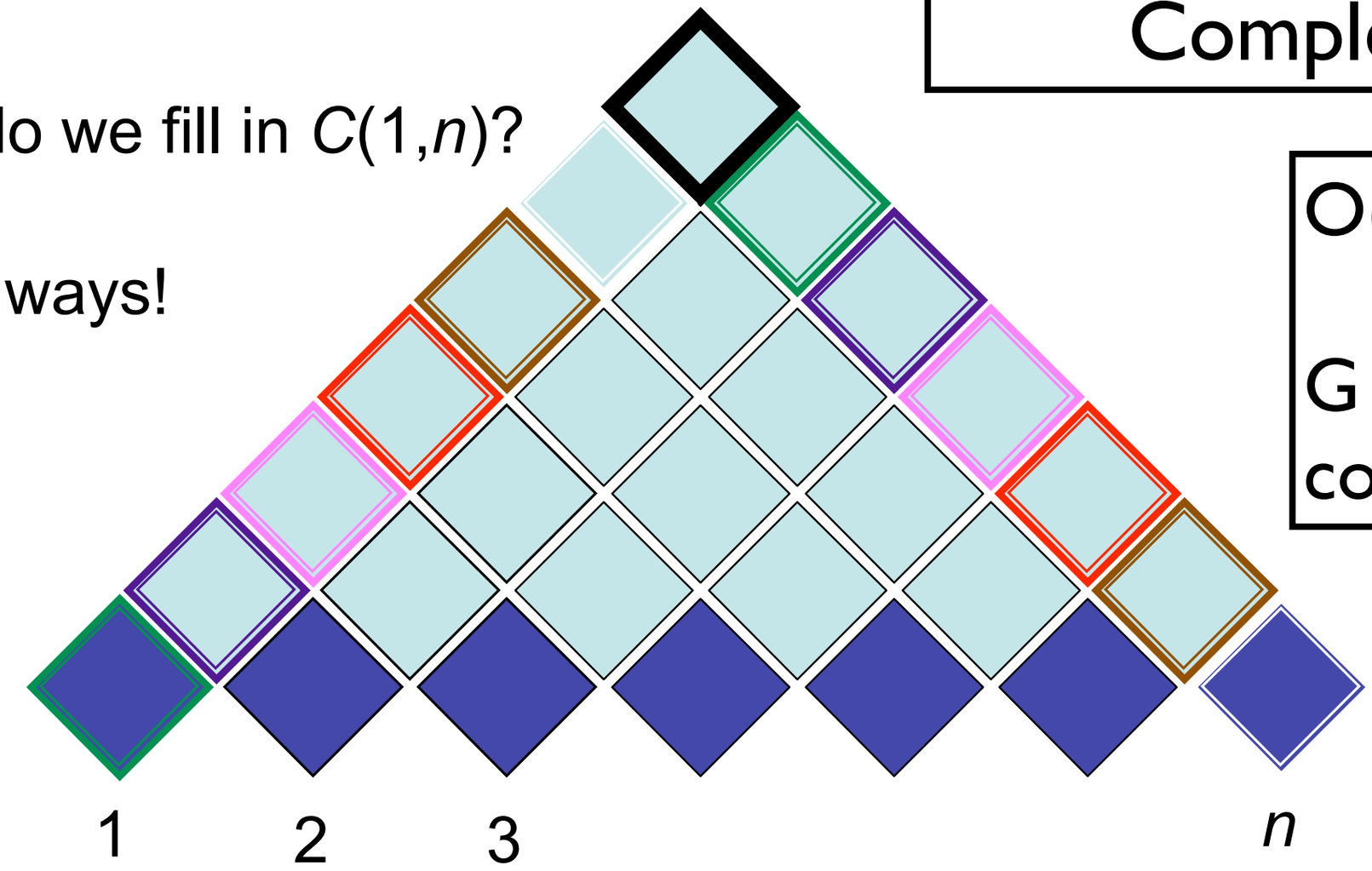


```
For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add  A to cell [i,j]
```

Computational Complexity ?

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

n - 1 ways!



O(G n^3)

G = grammar constant

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
- Extension of HMMs, strictly speaking
- (How to learn? Fully supervised with a treebank...)

(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")
    $\text{argmax}_y P(y \mid w) = \text{argmax}_y P(y, w)$

- (stopped here on 10/27/16 lecture)

- 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?

Treebanks

- Penn Treebank (constituents, English)
  - http://www.cis.upenn.edu/~treebank/home.html
  - Recent revisions in Ononotes
- Universal Dependencies
  - http://universaldependencies.org/

- Prague Treebank (syn+sem)
- many others...

- Know what you're getting!

Frequent noun phrases

"NP" from parser, vs. part-of-speech regex

Data Set	Method	Ranked List
Twitter	unigrams	snow, #tcot, al, dc, gore
	NPFST	
Old Bailey	unigrams	jacques, goodridge, rust, prisoner, sawtell
	ConsitParse	
	NPFST	
NYT	unigrams	will, united, one, government, new
	ConstitParse	
	NPFST	

Frequent noun phrases

"NP" from parser, vs. part-of-speech regex

Data Set	Method	Ranked List
Twitter	unigrams	snow, #tcot, al, dc, gore
	NPFST	
Old Bailey	unigrams	jacques, goodridge, rust, prisoner, sawtell
	ConsitParse	the prisoner, the warden, the draught, the fleet, the house
	NPFST	
NYT	unigrams	will, united, one, government, new
	ConstitParse	the united states, the government, the agreement, the president, the white house
	NPFST	

Frequent noun phrases

"NP" from parser, vs. part-of-speech regex

Data Set	Method	Ranked List
Twitter	unigrams	snow, #tcot, al, dc, gore
	NPFST	al gore's, snake oil science, 15 months, snow in dc, *bunch of snake oil science
Old Bailey	unigrams	jacques, goodridge, rust, prisoner, sawtell
	ConsitParse	the prisoner, the warden, the draught, the fleet, the house
	NPFST	middlesex jury, public house, warrant of attorney, baron perryn, *middlesex jury before lord loughborough
NYT	unigrams	will, united, one, government, new
	ConstitParse	the united states, the government, the agreement, the president, the white house
	NPFST	united states, united nations, white house, health care, *secretary of state warren christopher

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
(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) ))))))))))))))))
```