Parsing with Context-Free Grammars

CS 585, Fall 2017

Introduction to Natural Language Processing

http://people.cs.umass.edu/~brenocon/inlp2017

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	$\rightarrow$	$NP\ VP$	I + want a morning flight
NP	$\rightarrow$	$Pronoun$	I
	$\mid$	$Proper\text{-}Noun$	Los Angeles
	$\mid$	$Det\ Nominal$	a + flight
$Nominal$	$\rightarrow$	$Nominal\ Noun$	morning + flight
	$\mid$	$Noun$	flights
VP	$\rightarrow$	$Verb$	do
	$\mid$	$Verb\ NP$	want + a flight
	$\mid$	$Verb\ NP\ PP$	leave + Boston + in the morning
	$\mid$	$Verb\ PP$	leaving + on Thursday
PP	$\rightarrow$	$Preposition\ NP$	from + Los Angeles

Lexicon: NT expands to a terminal

$Noun$	$\rightarrow$	$flights \mid breeze \mid trip \mid morning \mid \ldots$
$Verb$	$\rightarrow$	$is \mid prefer \mid like \mid need \mid want \mid fly$
$Adjective$	$\rightarrow$	$cheapest \mid non-stop \mid first \mid latest$
		$\mid other \mid direct \mid \ldots$
$Pronoun$	$\rightarrow$	$me \mid I \mid you \mid it \mid \ldots$
$Proper\text{-}Noun$	$\rightarrow$	$Alaska \mid Baltimore \mid Los\ Angeles$
		$\mid Chicago \mid United \mid American \mid \ldots$
$Determiner$	$\rightarrow$	$the \mid a \mid an \mid this \mid these \mid that \mid \ldots$
$Preposition$	$\rightarrow$	$from \mid to \mid on \mid near \mid \ldots$
$Conjunction$	$\rightarrow$	$and \mid or \mid but \mid \ldots$

Constituent Parse Trees

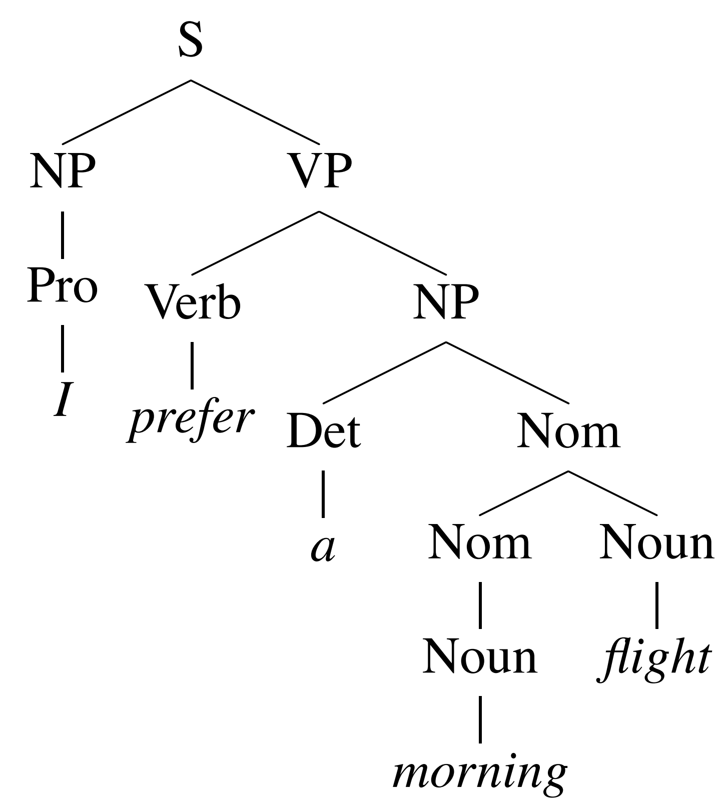


Figure 12.4 The parse tree for "I prefer a morning flight" according to grammar $\mathscr{L}_0$.

Representations:

Bracket notation

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

Non-terminal positional spans

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

Ambiguity in parsing

Syntactic ambiguity is endemic to natural language:[1]

- Attachment ambiguity: we eat sushi with chopsticks, I shot an elephant in my pajamas.

[1] Examples borrowed from Dan Klein

Ambiguity in parsing

Syntactic ambiguity is endemic to natural language:[1]

- Attachment ambiguity: we eat sushi with chopsticks, I shot an elephant in my pajamas.
- Modifier scope: southern food store

[1] Examples borrowed from Dan Klein

Ambiguity in parsing

Syntactic ambiguity is endemic to natural language:[1]

- Attachment ambiguity: we eat sushi with chopsticks, I shot an elephant in my pajamas.
- Modifier scope: southern food store
- Particle versus preposition: The puppy tore up the staircase.

[1] Examples borrowed from Dan Klein

Ambiguity in parsing

Syntactic ambiguity is endemic to natural language:[1]

- Attachment ambiguity: we eat sushi with chopsticks, I shot an elephant in my pajamas.
- Modifier scope: southern food store
- Particle versus preposition: The puppy tore up the staircase.
- Complement structure: The tourists objected to the guide that they couldn't hear.

[1] Examples borrowed from Dan Klein

Ambiguity in parsing

Syntactic ambiguity is endemic to natural language:[1]

- Attachment ambiguity: we eat sushi with chopsticks, I shot an elephant in my pajamas.
- Modifier scope: southern food store
- Particle versus preposition: The puppy tore up the staircase.
- Complement structure: The tourists objected to the guide that they couldn't hear.
- Coordination scope: "I see," said the blind man, as he picked up the hammer and saw.

[1] Examples borrowed from Dan Klein

Ambiguity in parsing

Syntactic ambiguity is endemic to natural language:[1]

- Attachment ambiguity: we eat sushi with chopsticks, I shot an elephant in my pajamas.
- Modifier scope: southern food store
- Particle versus preposition: The puppy tore up the staircase.
- Complement structure: The tourists objected to the guide that they couldn't hear.
- Coordination scope: "I see," said the blind man, as he picked up the hammer and saw.
- Multiple gap constructions: The chicken is ready to eat

[1] Examples borrowed from Dan Klein

Attachment ambiguity

Probability of attachment sites

- [imposed [a ban [on asbestos]]]
- [imposed [a ban][on asbestos]]

Attachment ambiguity

Probability of attachment sites

- [imposed [a ban [on asbestos]]]
- [imposed [a ban][on asbestos]]

Include head of embedded NP

- ...[it [would end [its venture [with Maserati]]]]
- ...[it [would end [its venture][with Maserati]]]

Attachment ambiguity

Probability of attachment sites

- [imposed [a ban [on asbestos]]]
- [imposed [a ban][on asbestos]]

Include head of embedded NP

- ...[it [would end [its venture [with Maserati]]]]
- ...[it [would end [its venture][with Maserati]]]

Resolve multiple ambiguities simultaneously

- Cats scratch people with claws with knives

Ambiguities make parsing hard

- **1. Computationally: how to reuse work across combinatorially many trees?**

- 2. How to make good attachment decisions?

Parsing with a CFG

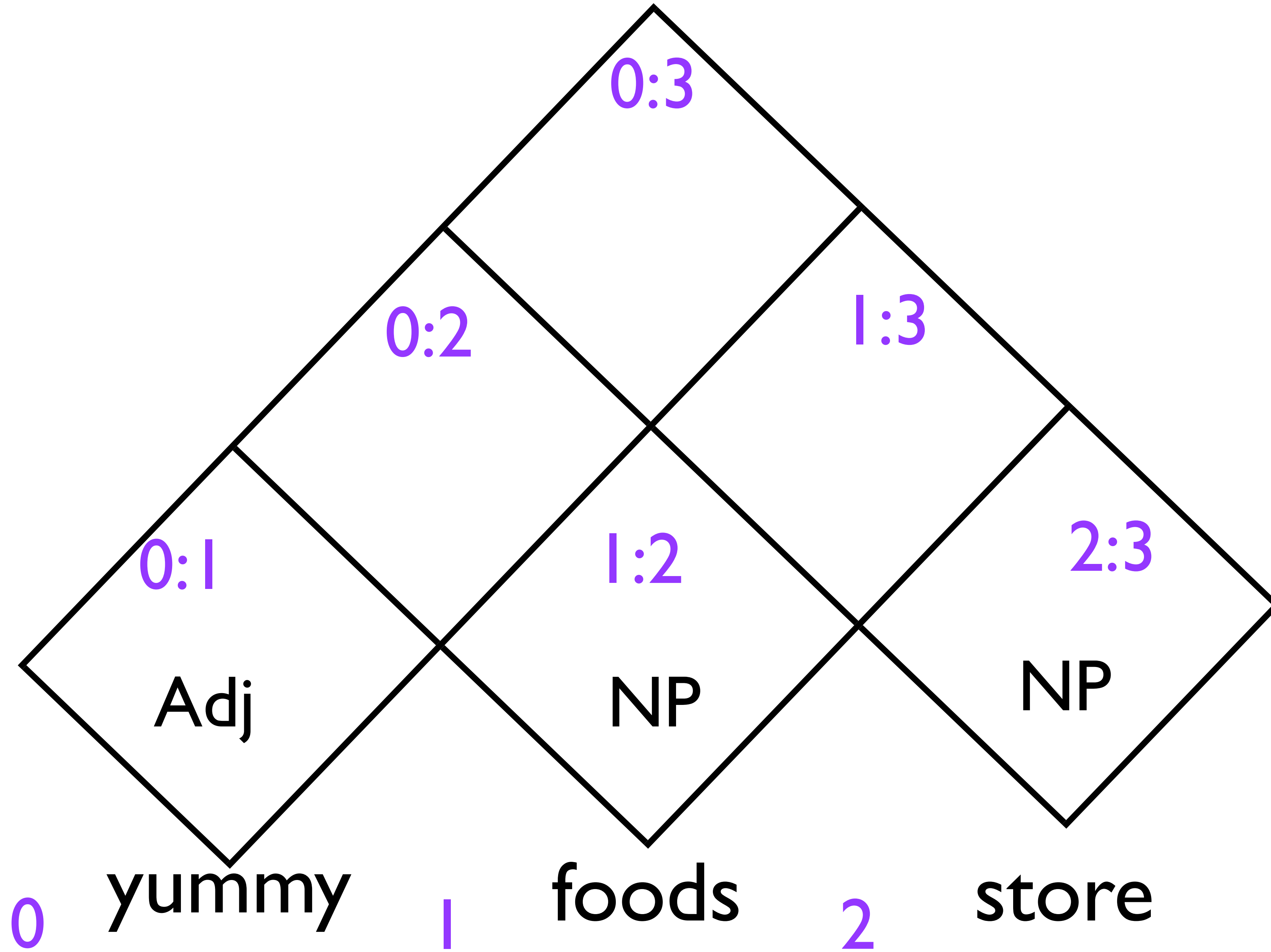
- Task: given text and a CFG, answer:
  - Does there exist at least one parse?
  - Enumerate parses (backpointers)

- Approaches: top-down, left-to-right, bottom-up

- CKY (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: <=2 nonterminals in expansion
    - instead of NP -> NP CC NP, could do:
      - NP -> NP_CC NP
      - NP_CC -> NP CC

CKY

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



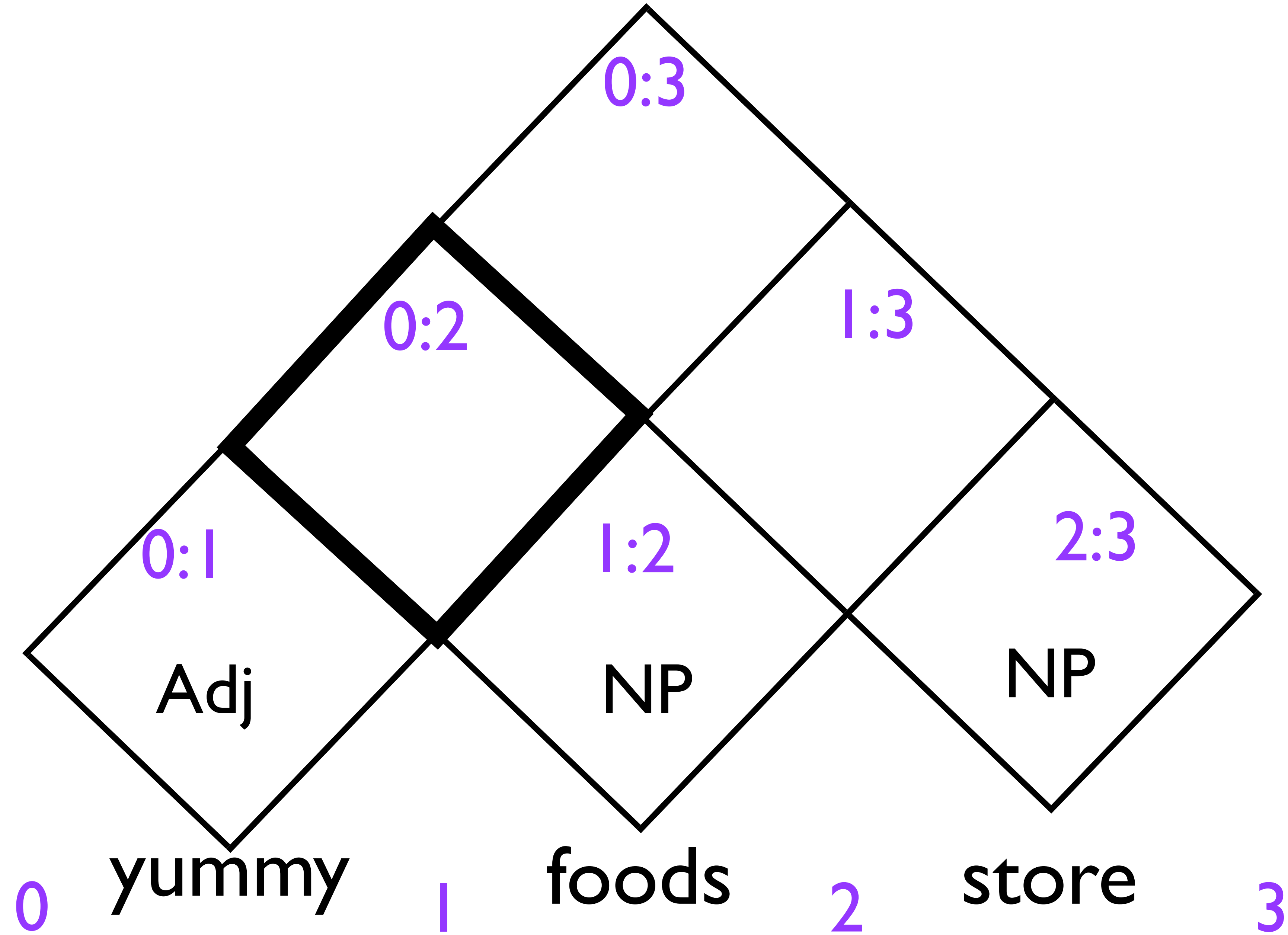
For cell [i,j] (loop through them bottom-up)
For possible splitpoint k=(i+1)..(j-1):
For every B in [i,k] and C in [k,j],
If exists rule A -> B C,
add A to cell [i,j] (*Recognizer*)
... or ...
add (A,B,C, k) to cell [i,j] (*Parser*)

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



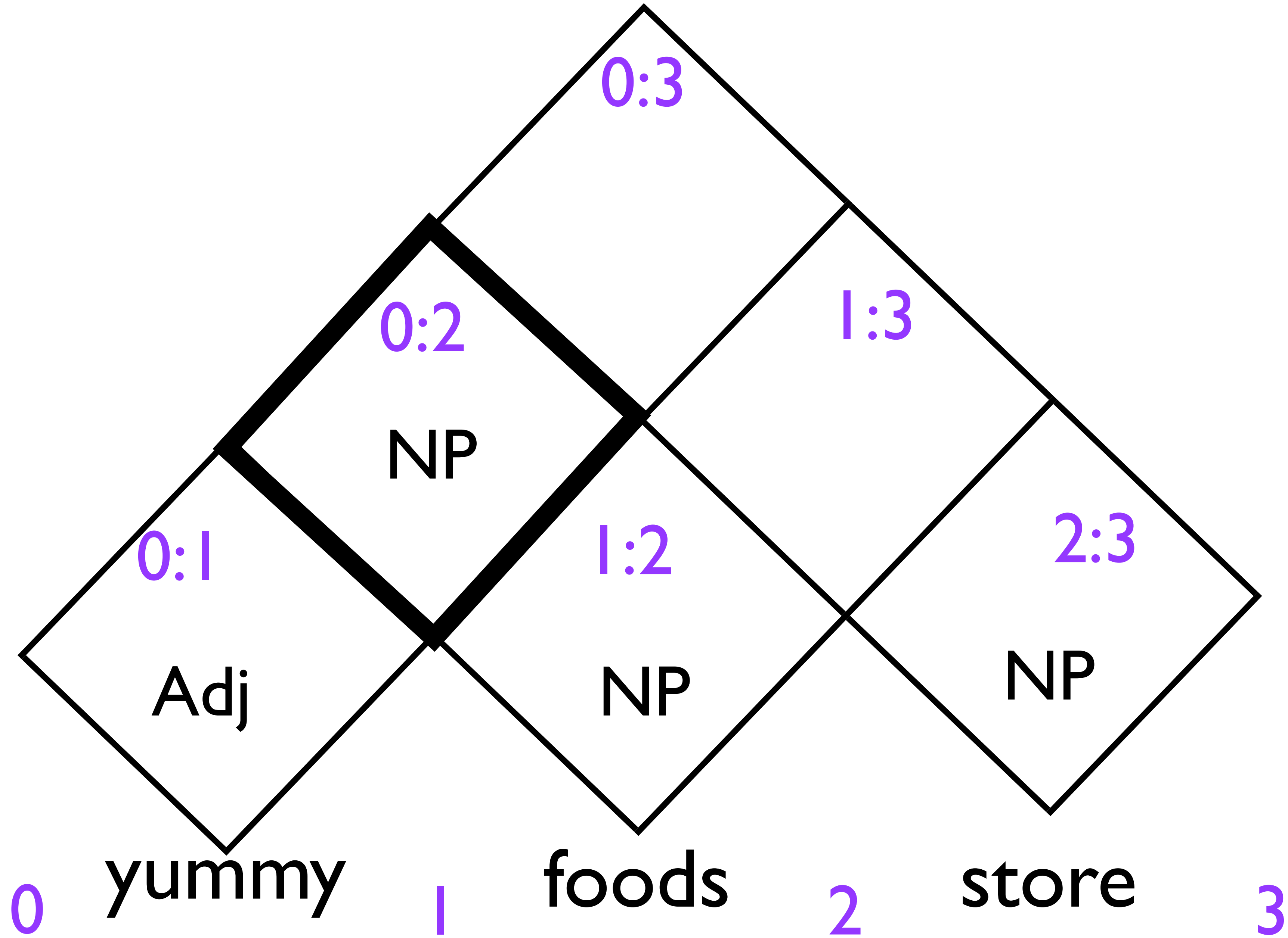
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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

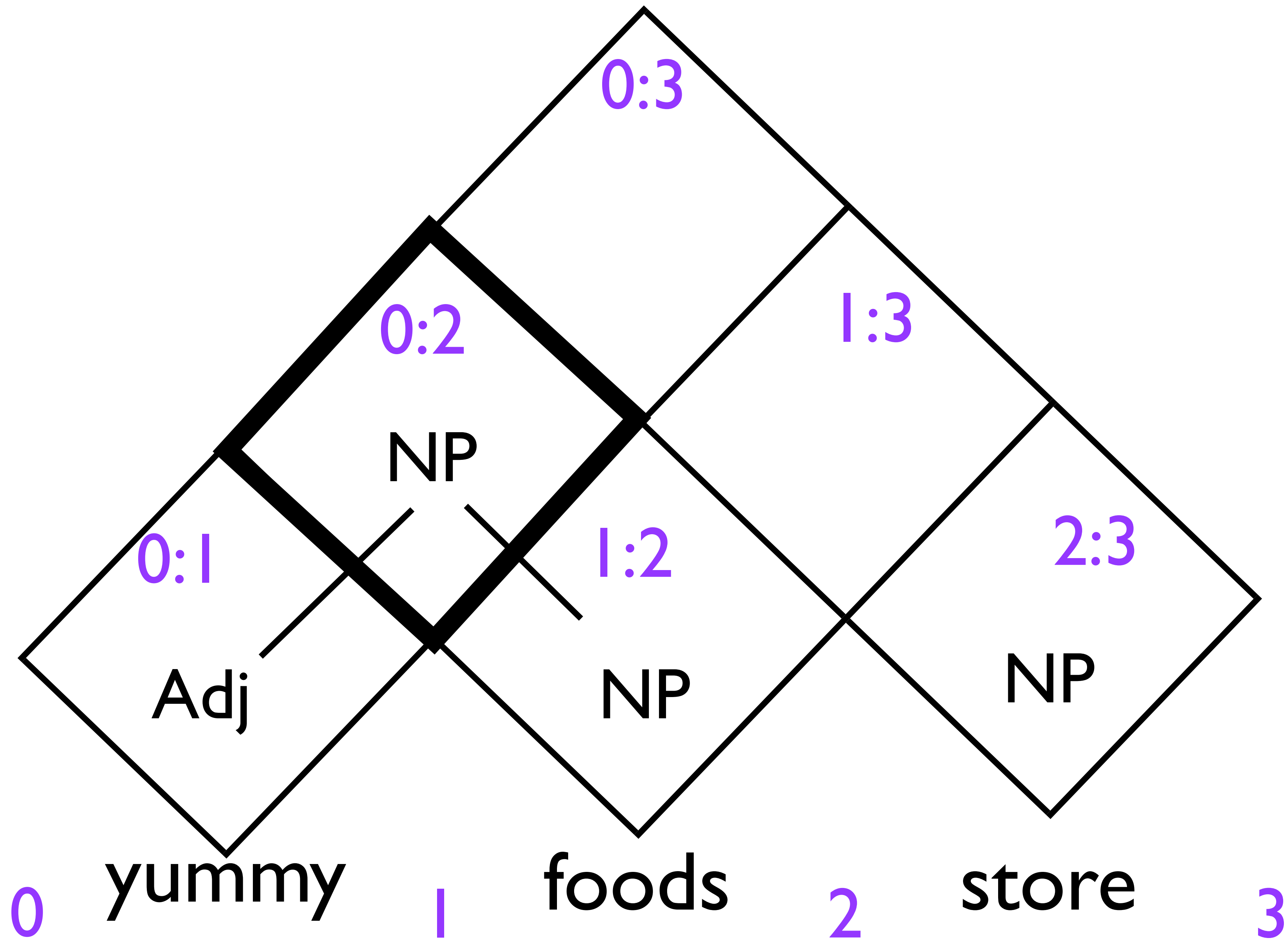


For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

Recognizer: per span, record list of possible nonterminals

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

CKY



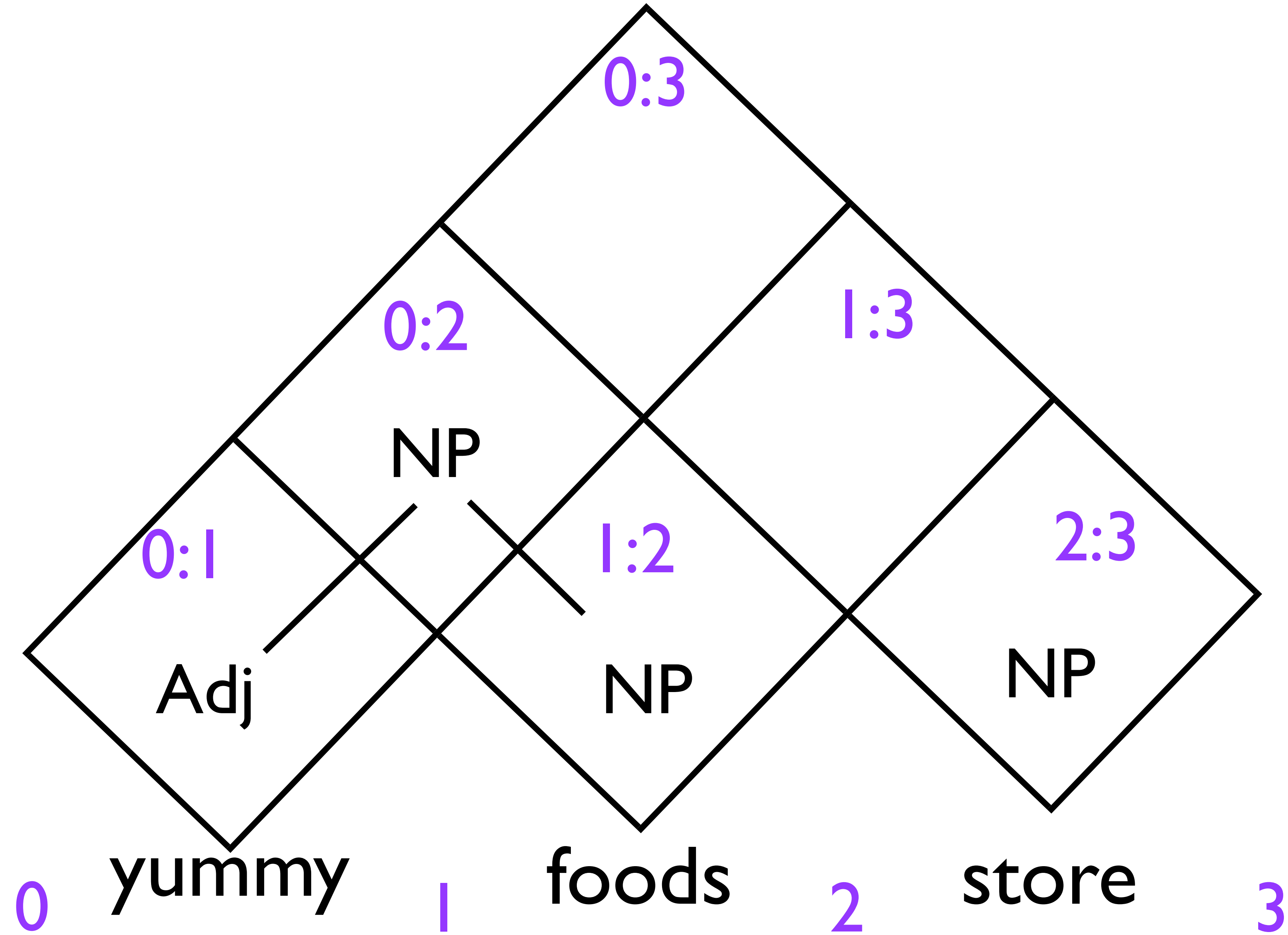
Grammar
Adj -> yummy
NP -> foods
NP -> store
NP -> NP NP
NP -> Adj NP

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

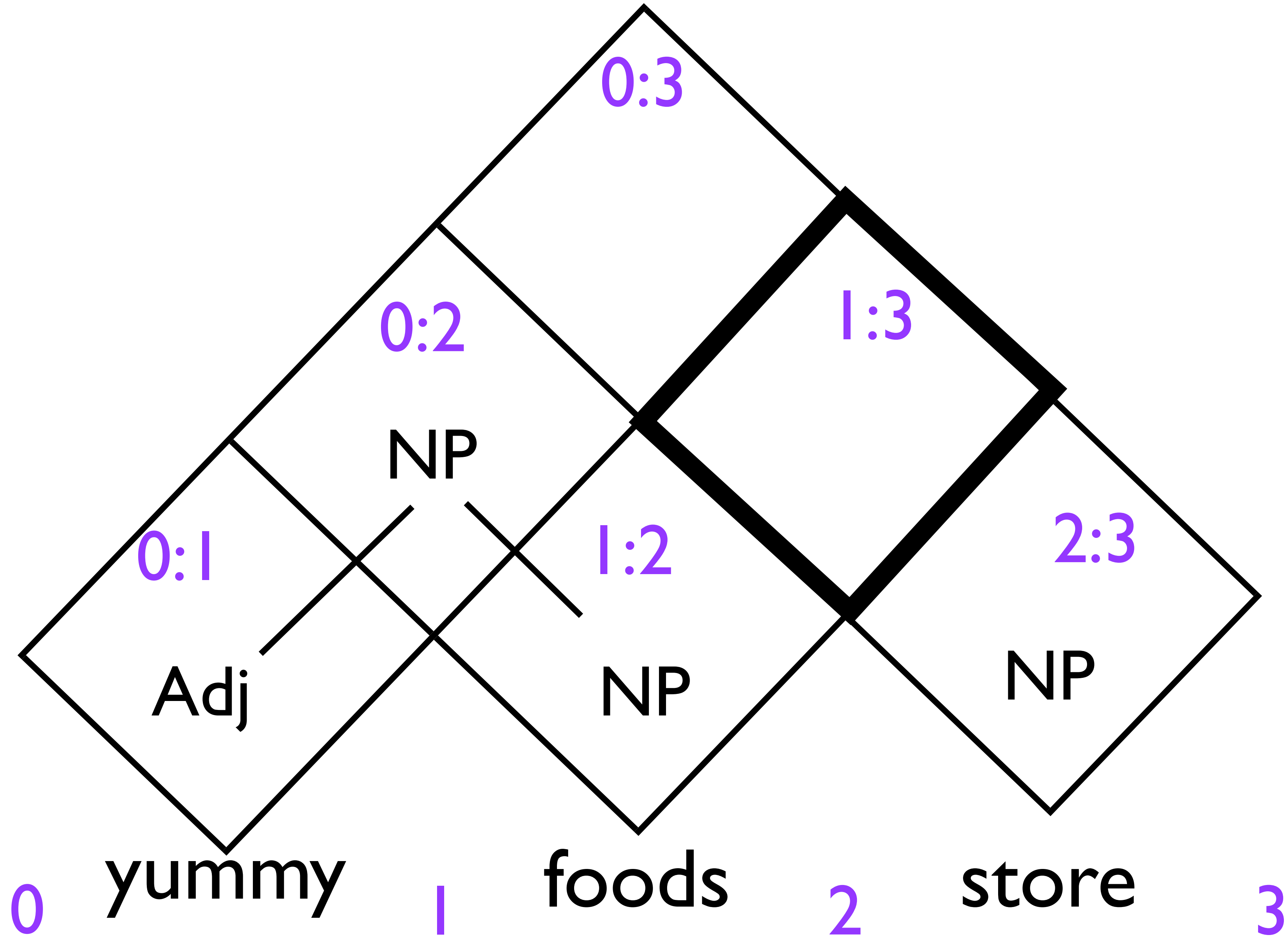
For cell [i,j] (loop through them bottom-up)
For possible splitpoint k=(i+1)..(j-1):
For every B in [i,k] and C in [k,j],
If exists rule A -> B C,
add A to cell [i,j] (*Recognizer*)
... or ...
add (A,B,C, k) to cell [i,j] (*Parser*)

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

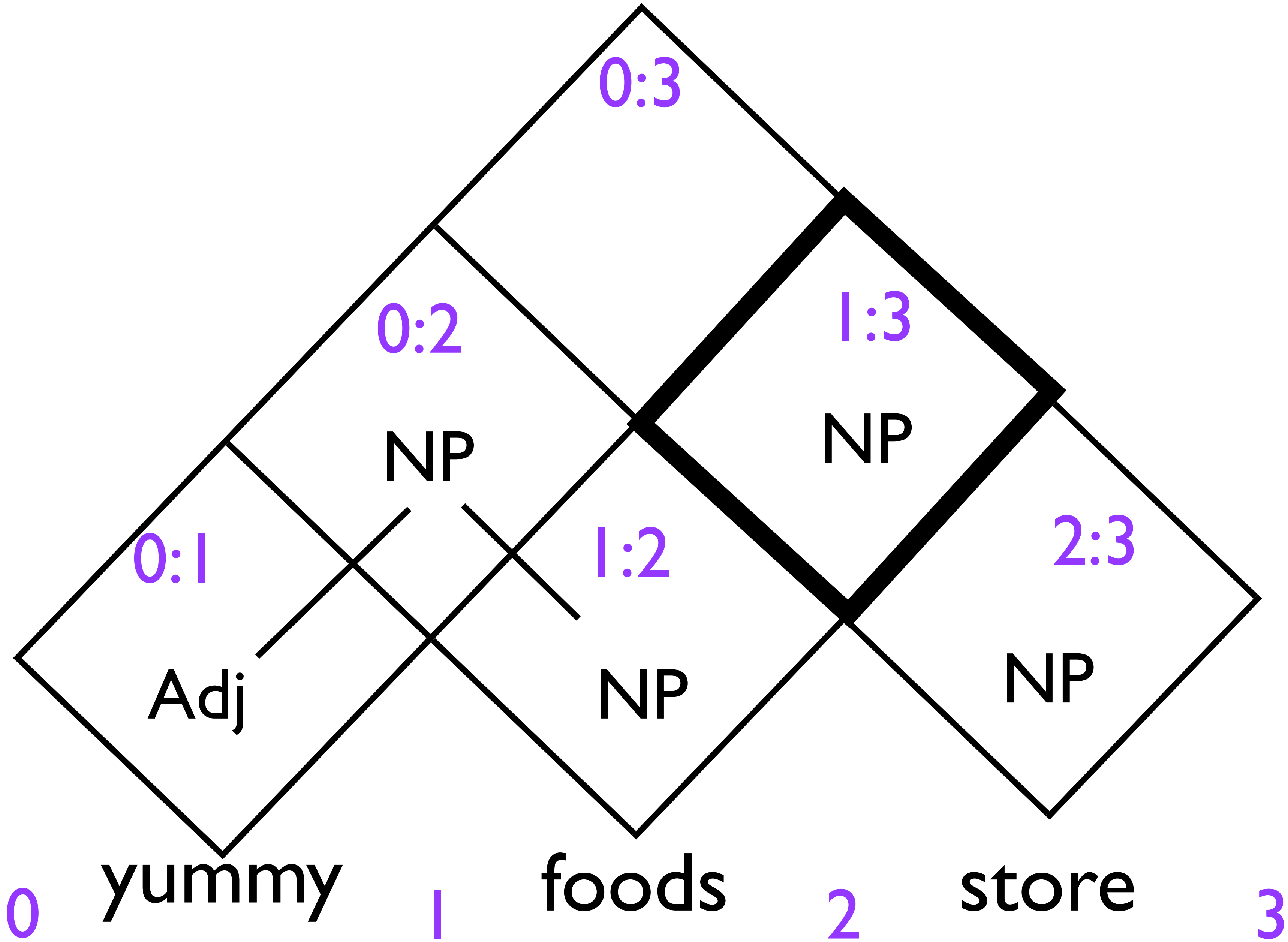


For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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

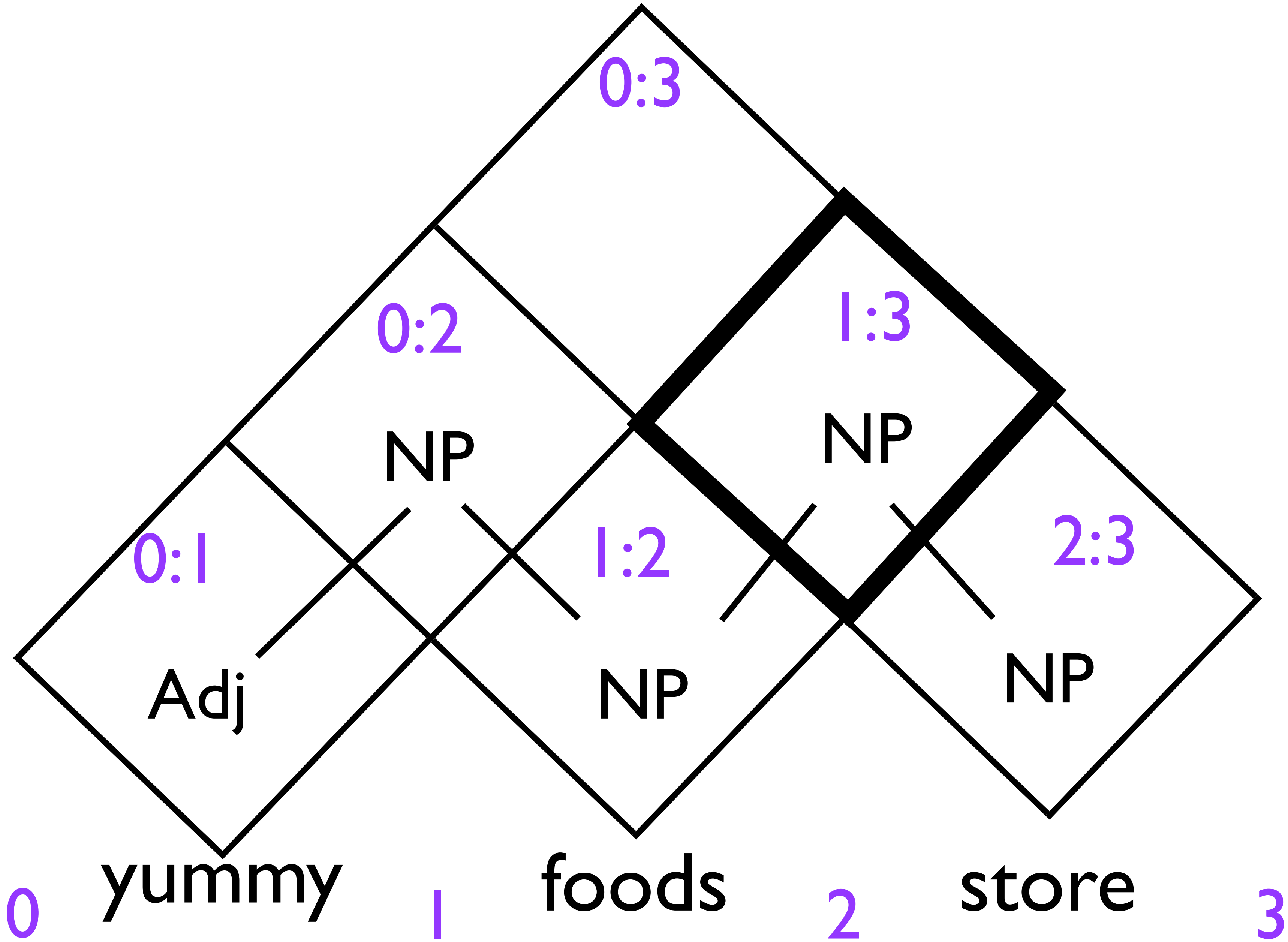
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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

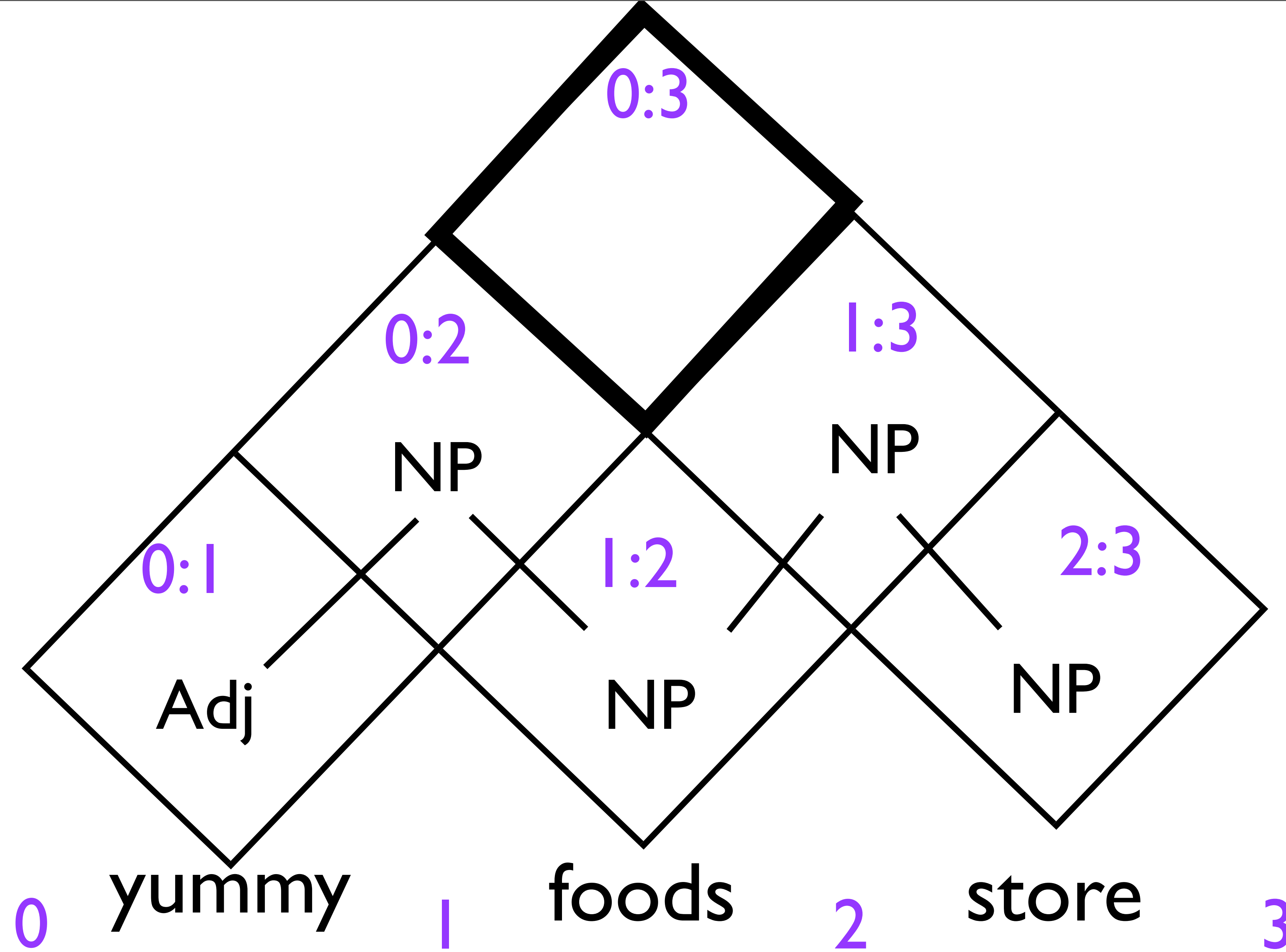


For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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

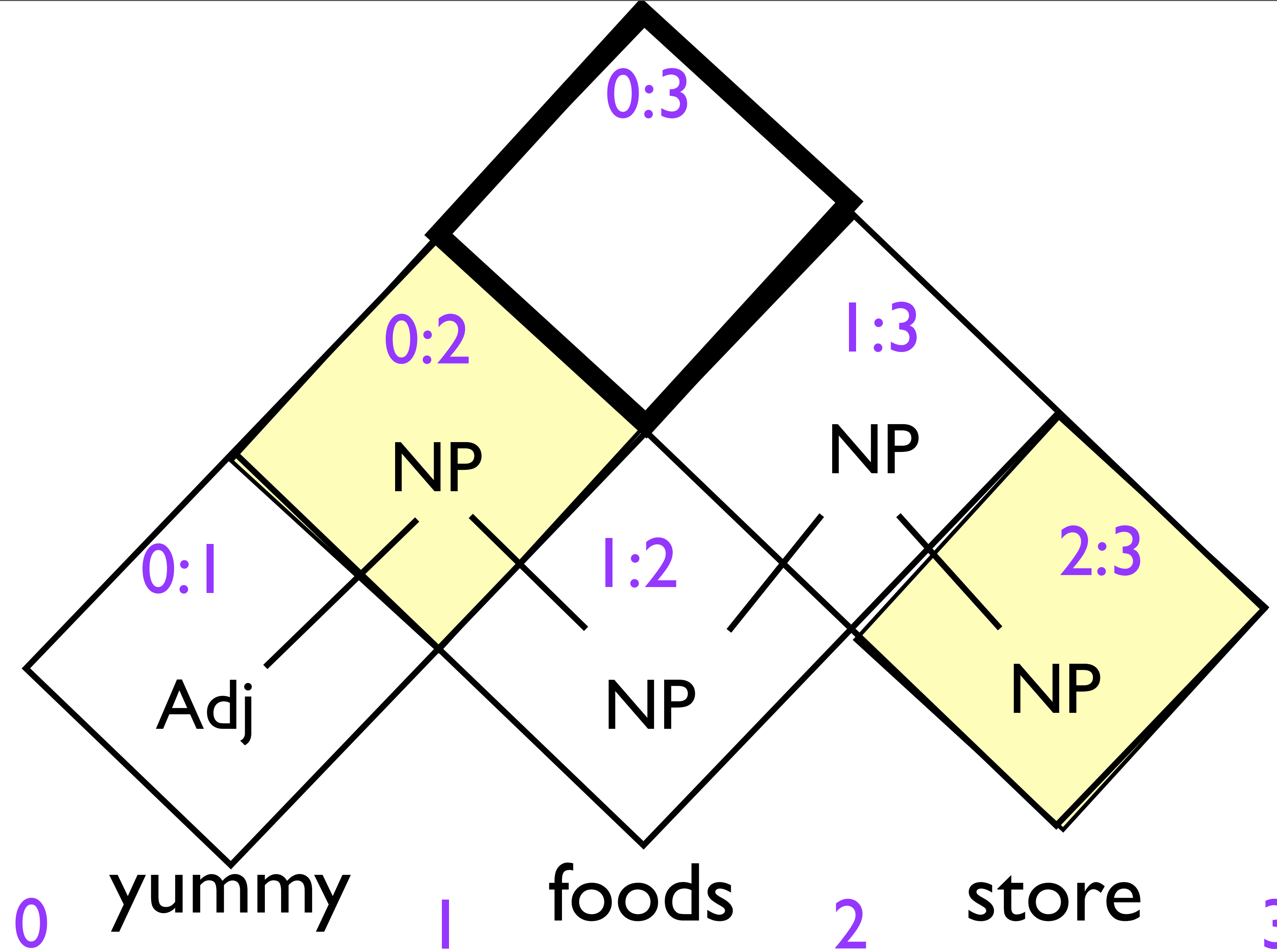
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



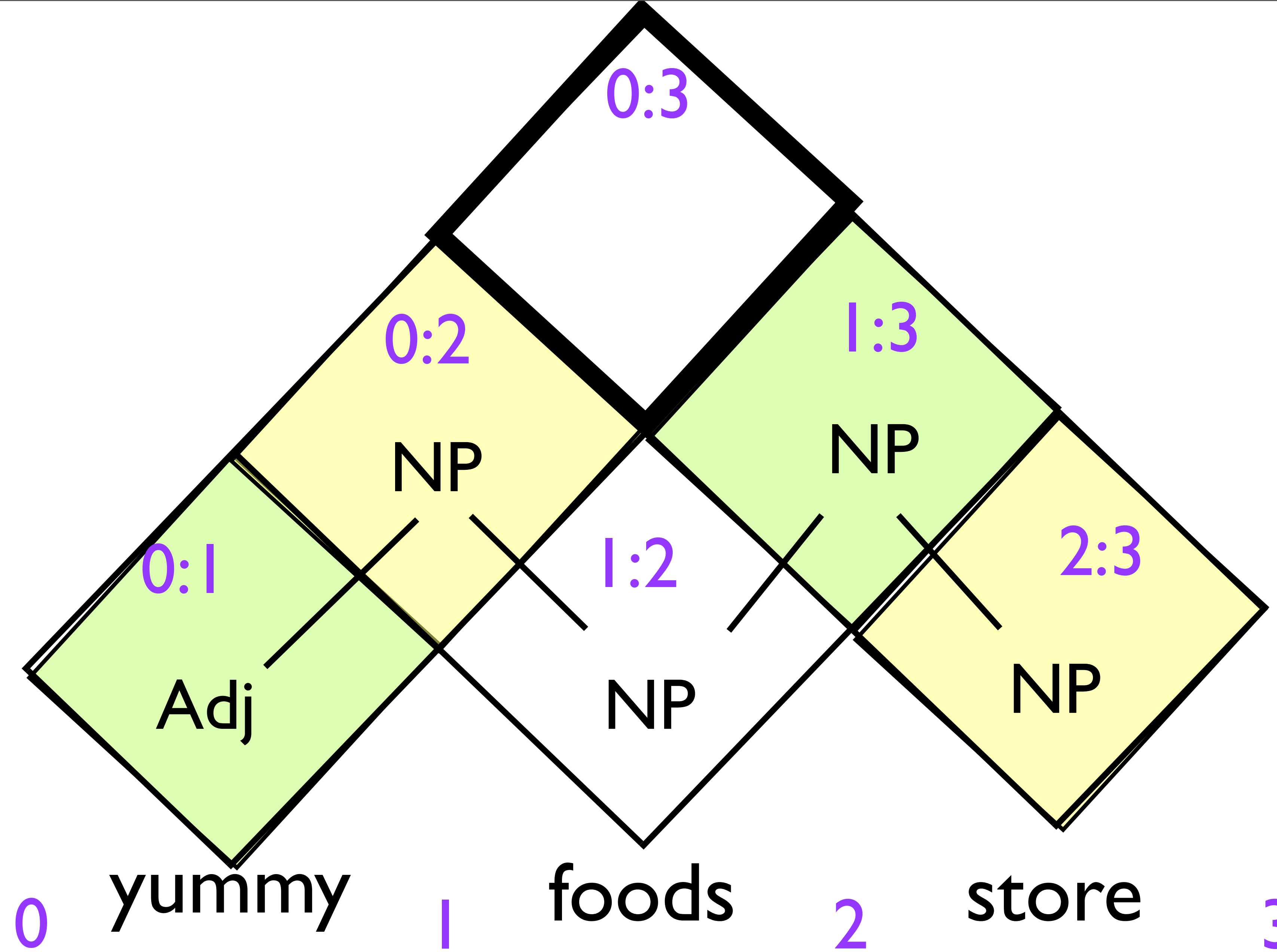
For cell [i,j] (loop through them bottom-up)
For possible splitpoint k=(i+1)..(j-1):
For every B in [i,k] and C in [k,j],
If exists rule A -> B C,
add A to cell [i,j] (*Recognizer*)
... or ...
add (A,B,C, k) to cell [i,j] (*Parser*)

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



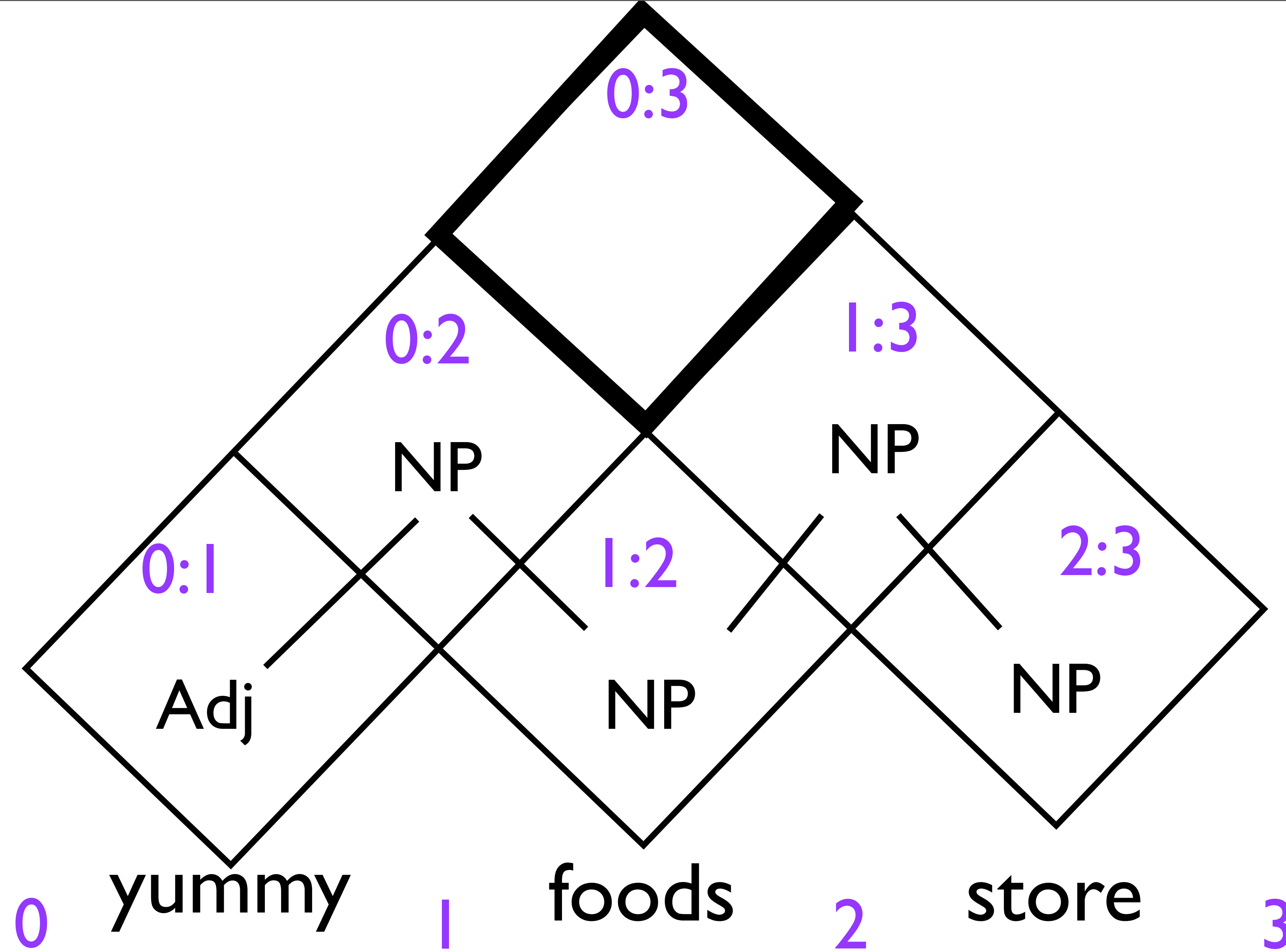
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]   (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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



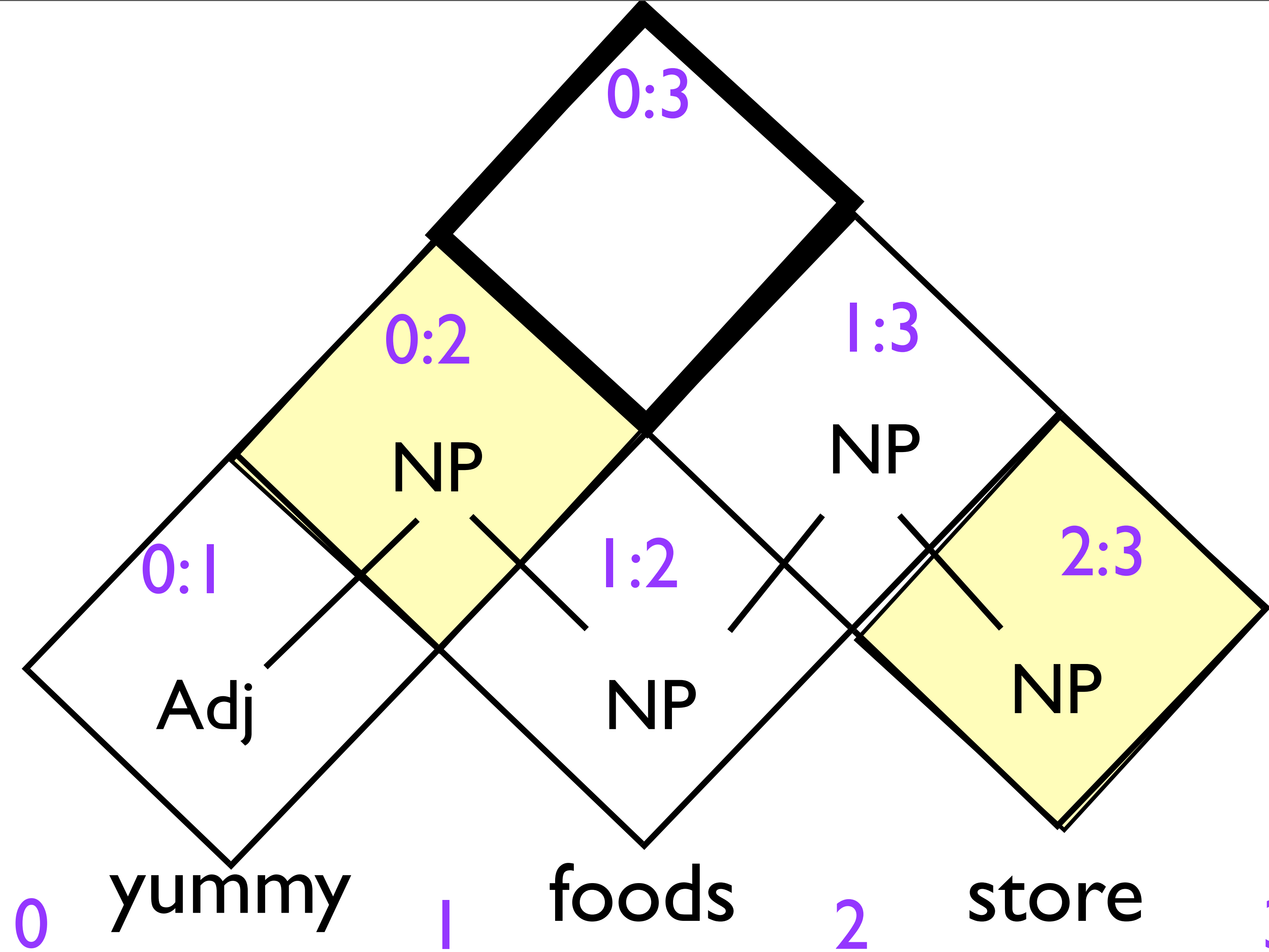
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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*)

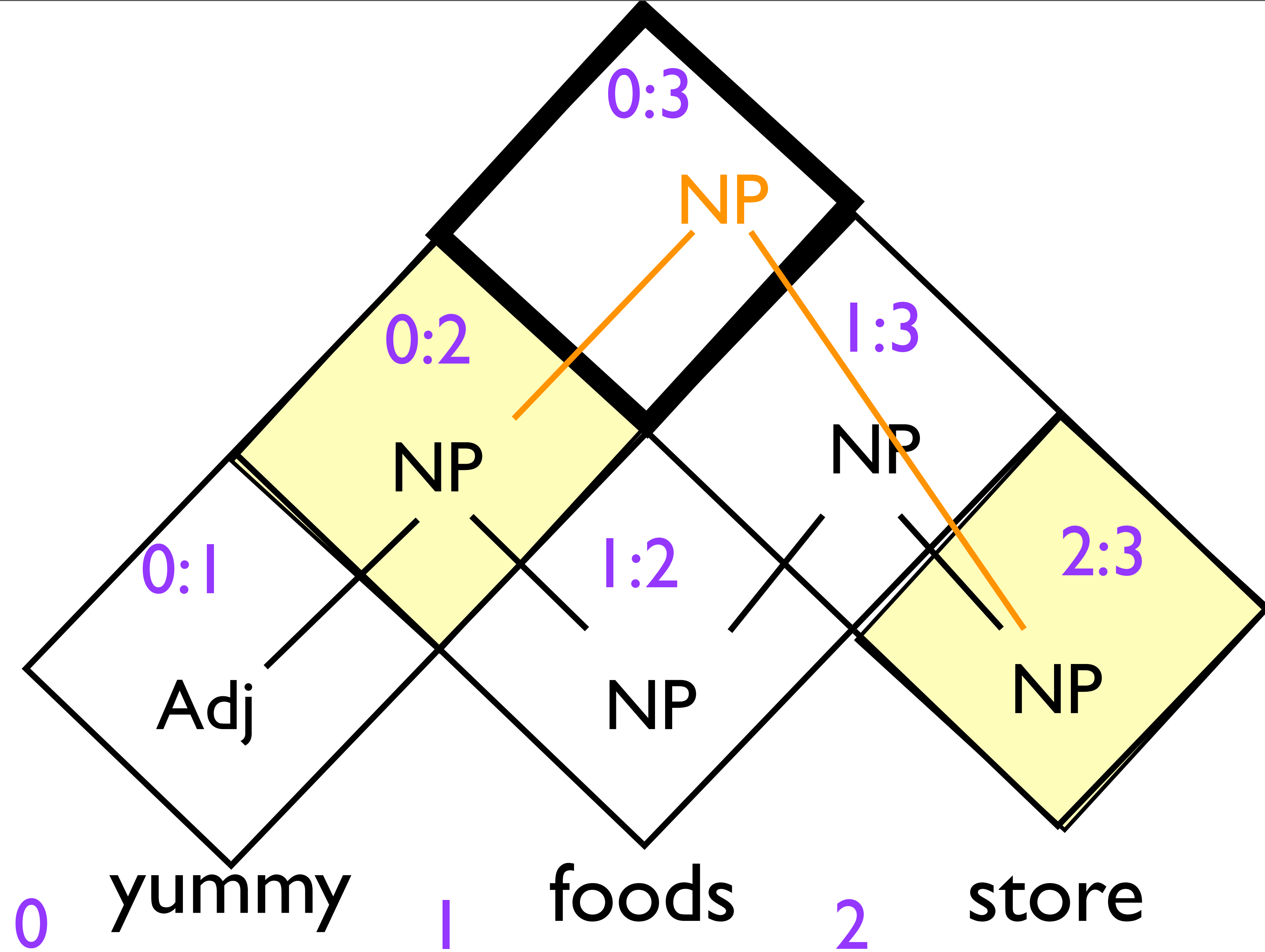
Recognizer: per span, record list of possible nonterminals

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

Thursday, November 16, 17

CKY

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



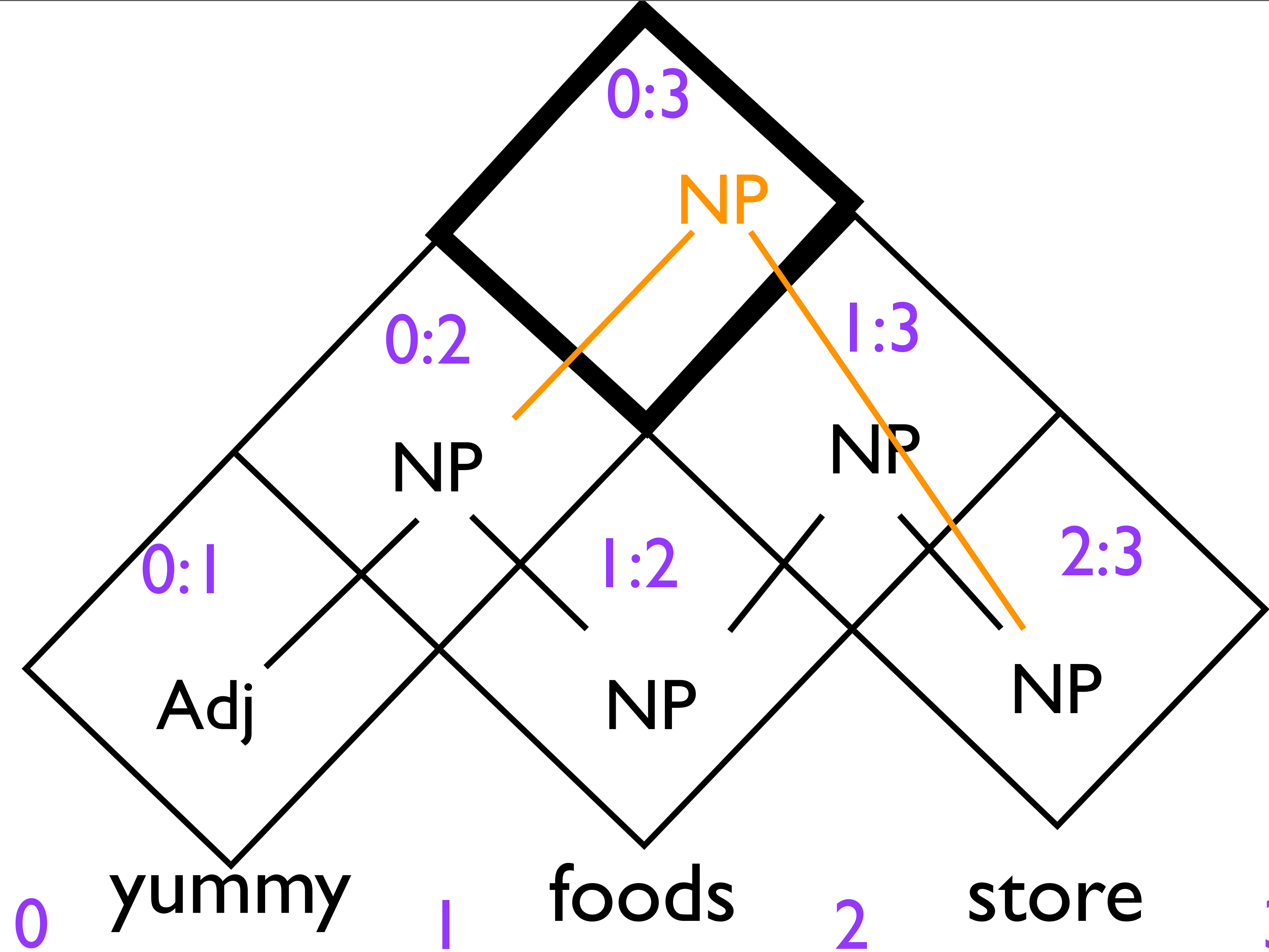
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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



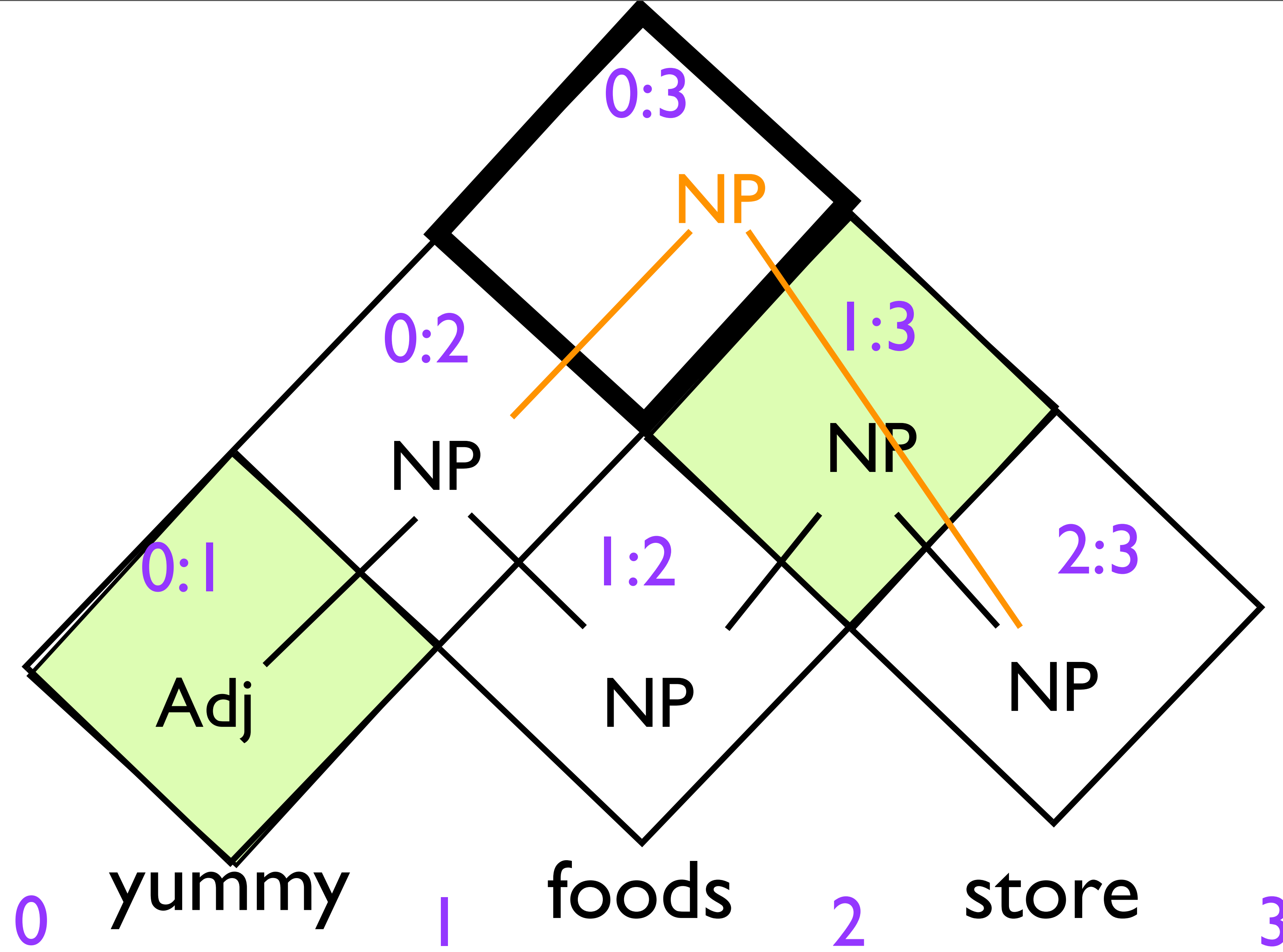
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



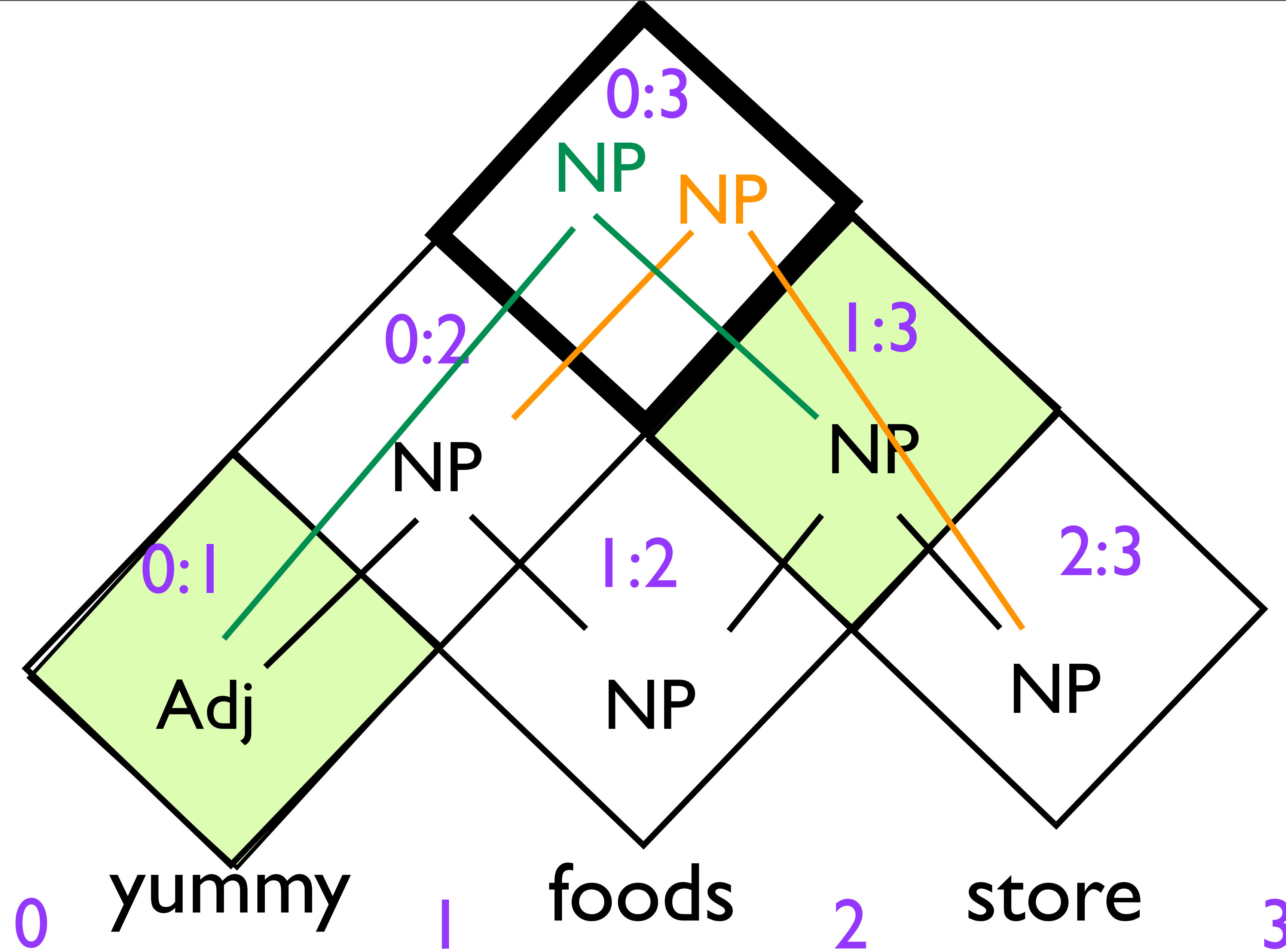
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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*)

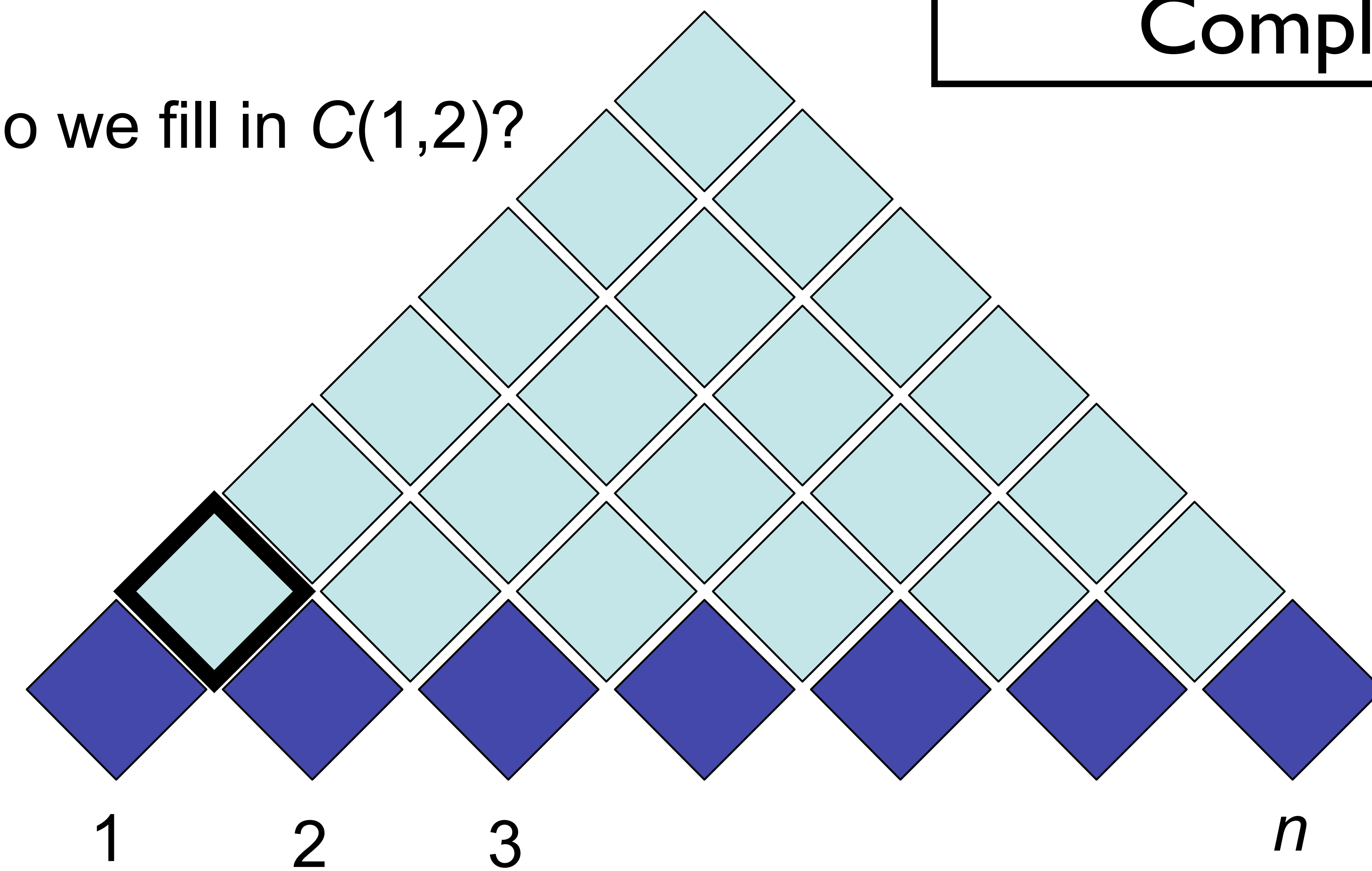
Recognizer: per span, record list of possible nonterminals

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)?

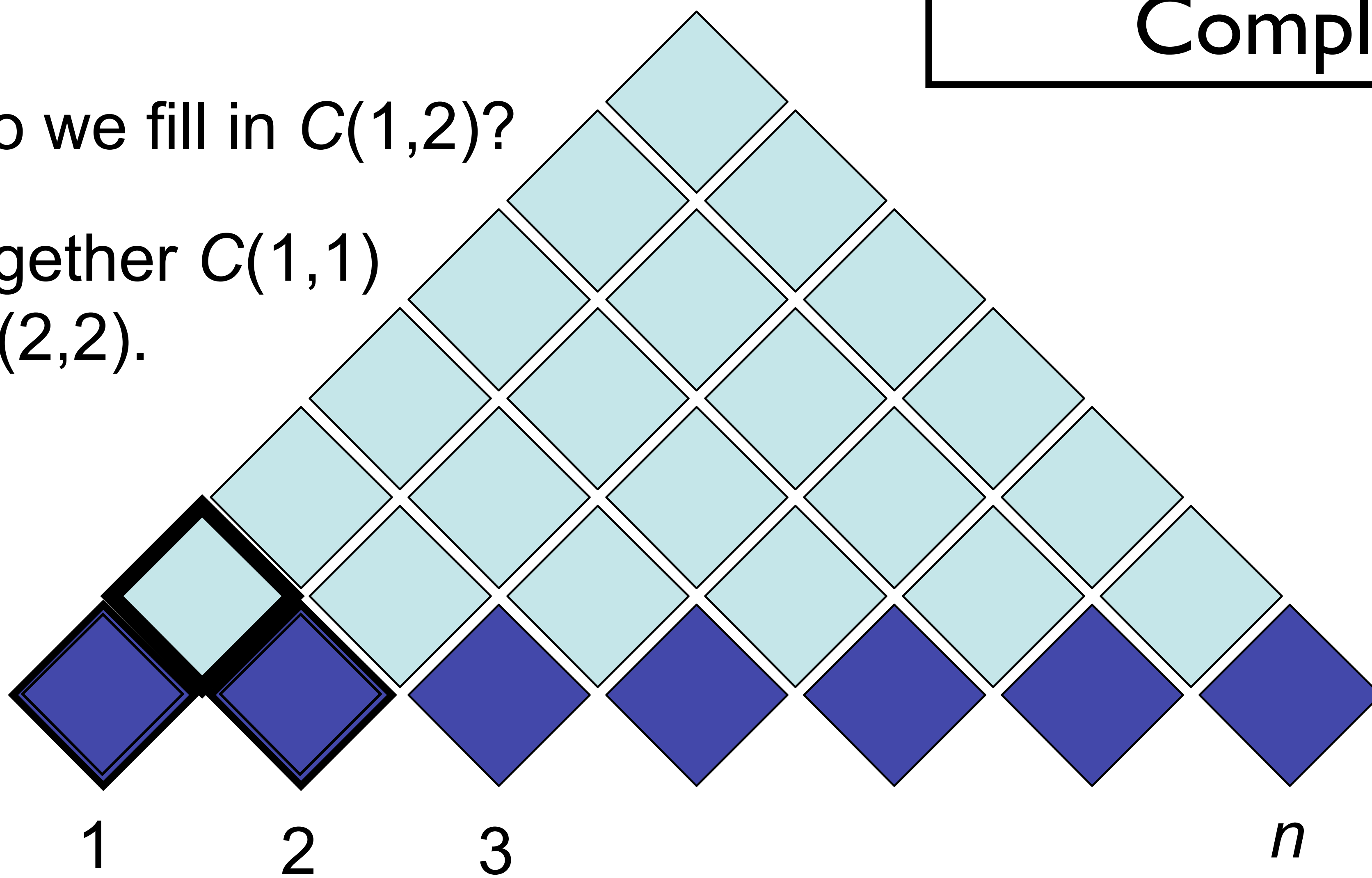


For cell [i,j]
For possible splitpoint k=(i+1)..(j-1):
For every B in [i,k] and C in [k,j],
If exists rule A -> B C,
add A to cell [i,j]

Computational Complexity ?

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

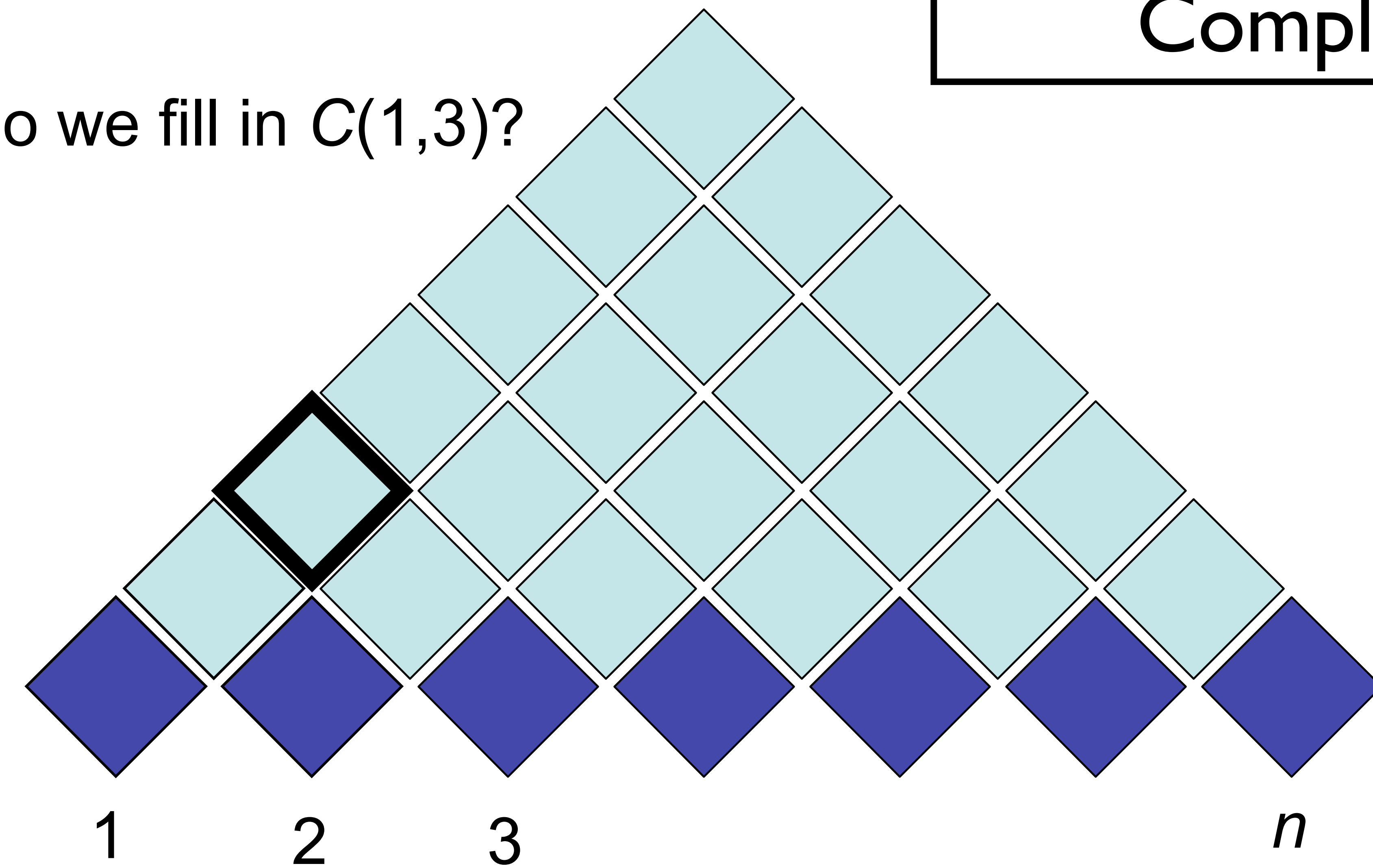
Put together *C*(1,1) and *C*(2,2).



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)?

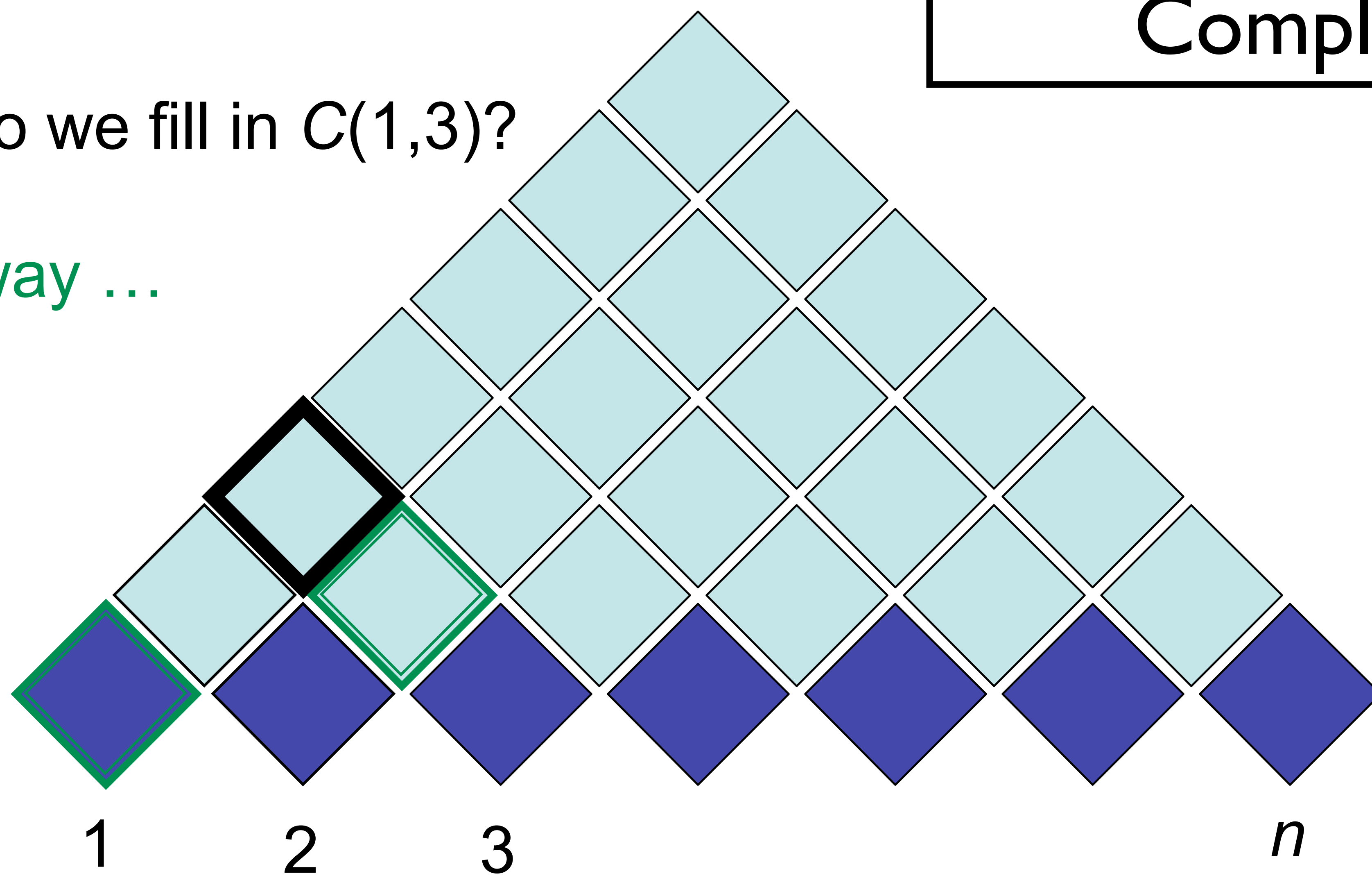


For cell [i,j]
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j]

Computational Complexity ?

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

One way …



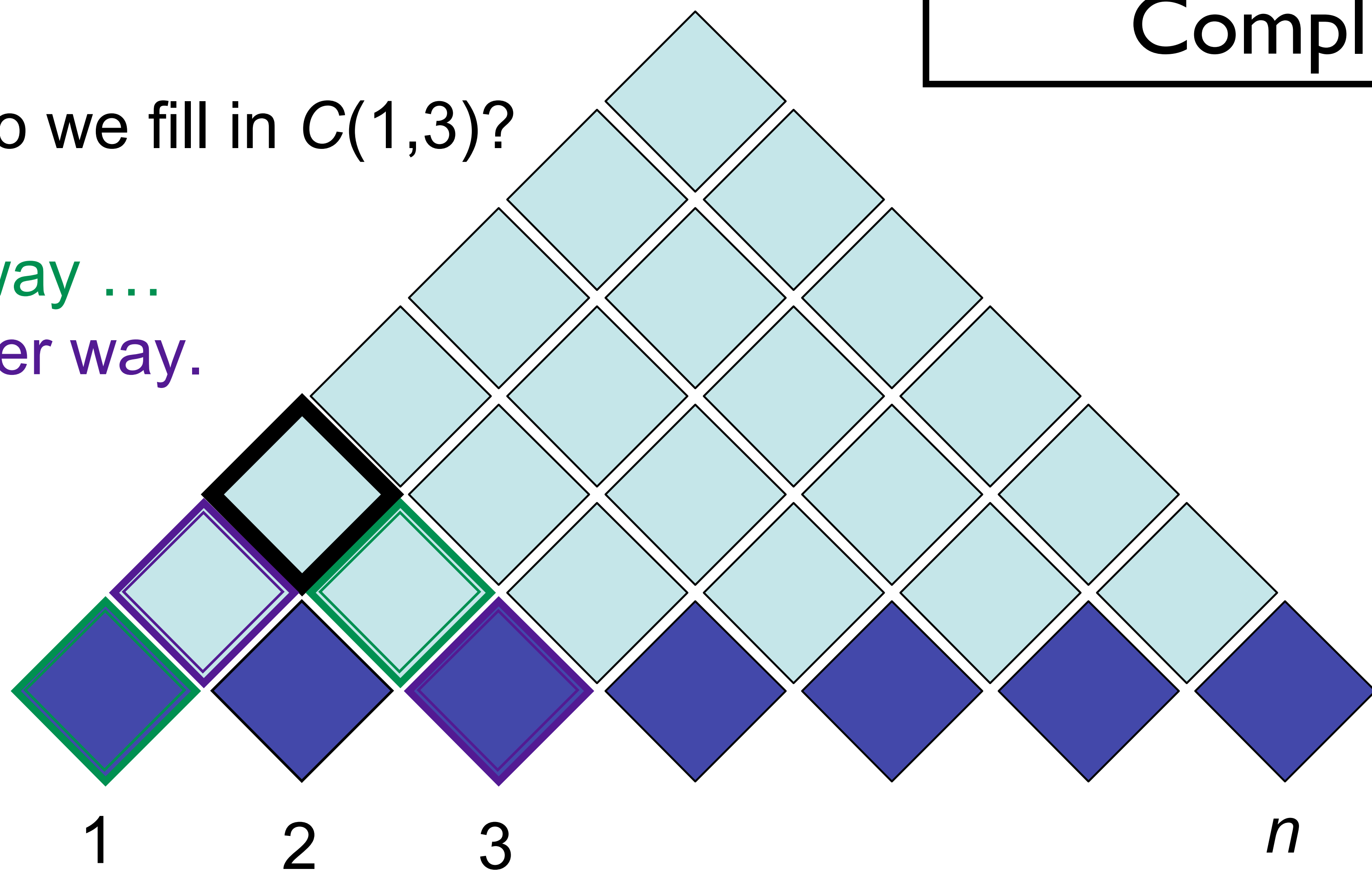
For cell [i,j]
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j]

Computational Complexity ?

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

One way …

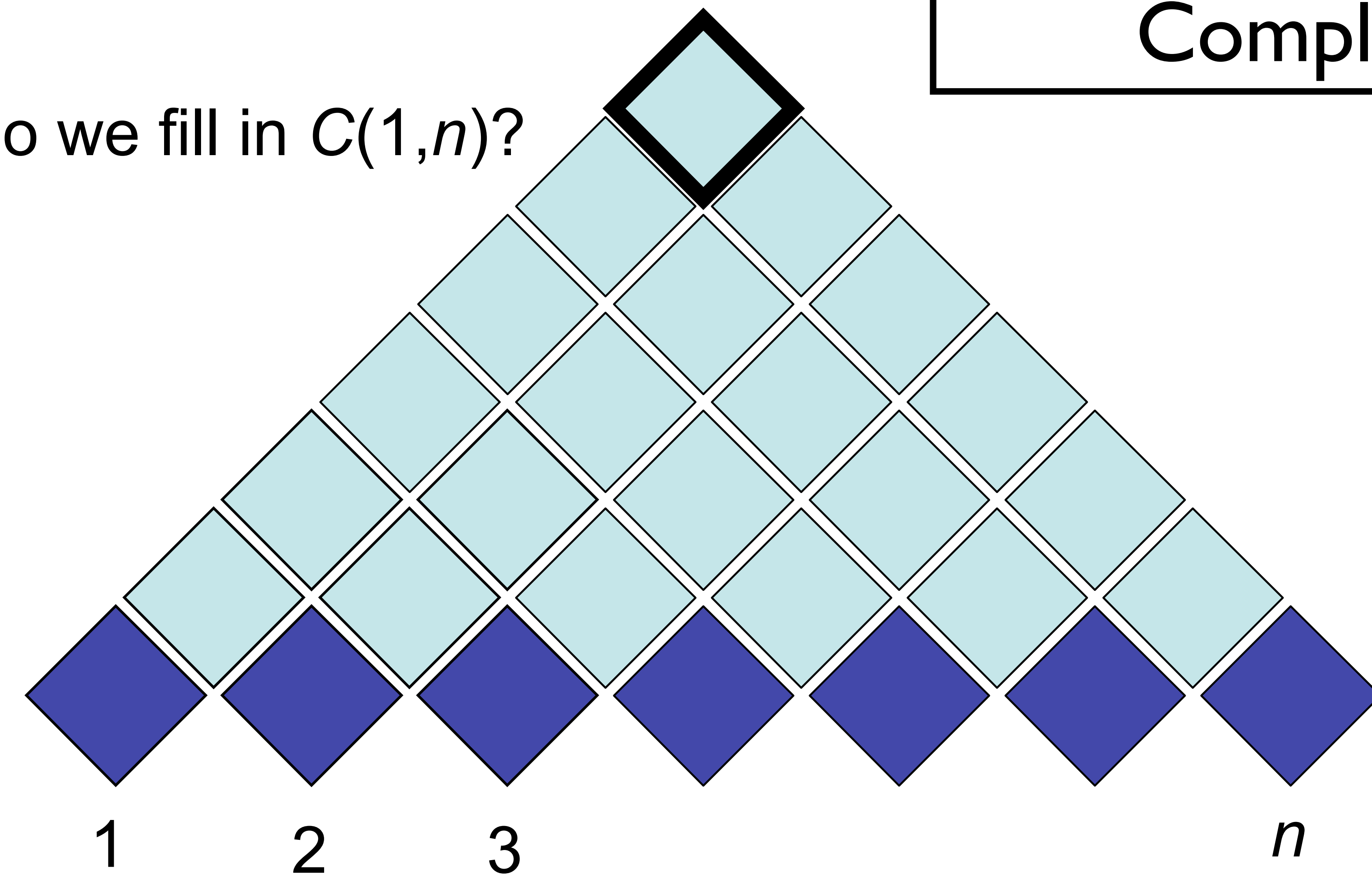
Another way.



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

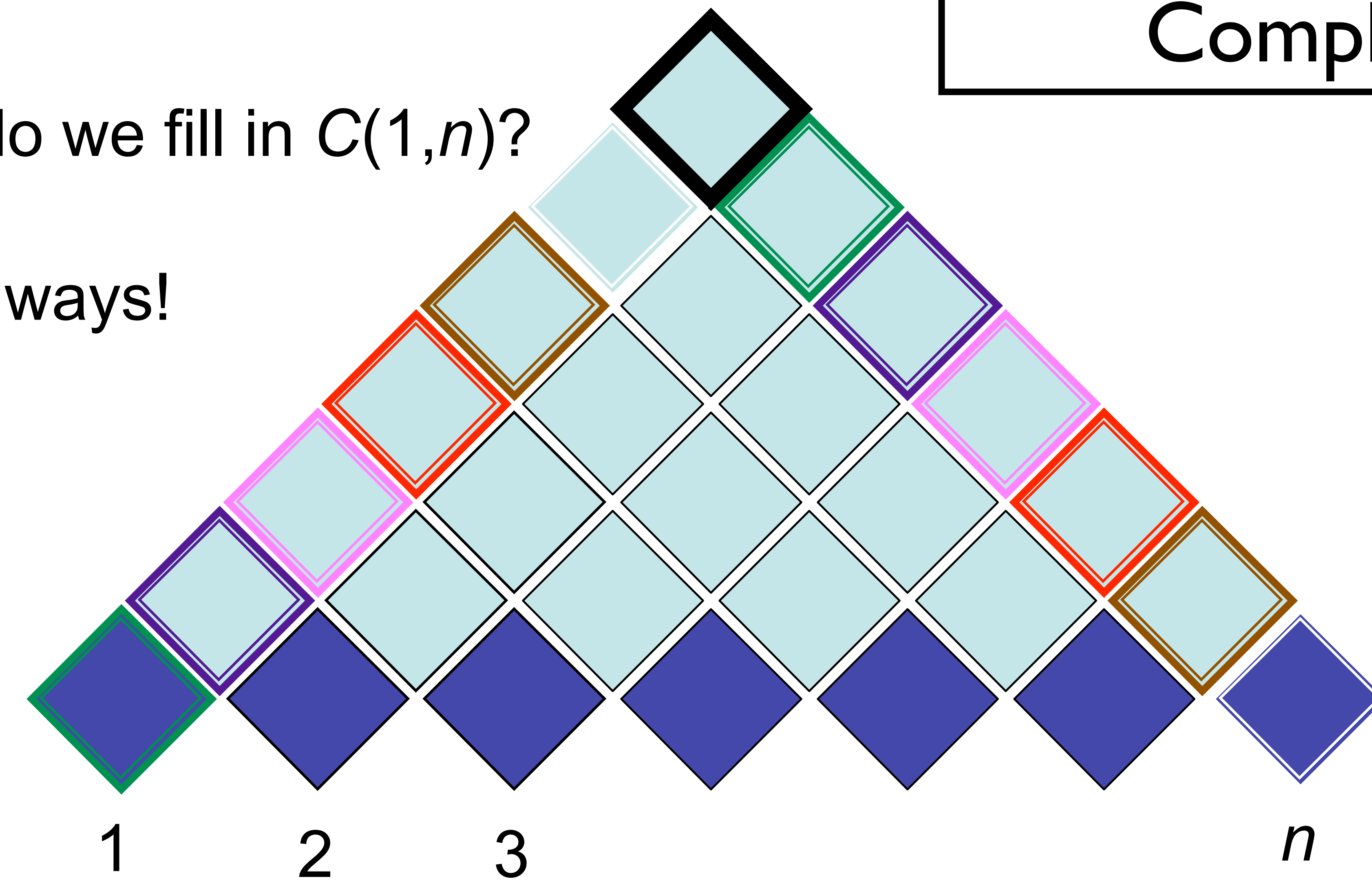


For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]

Computational Complexity ?

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

n - 1 ways!

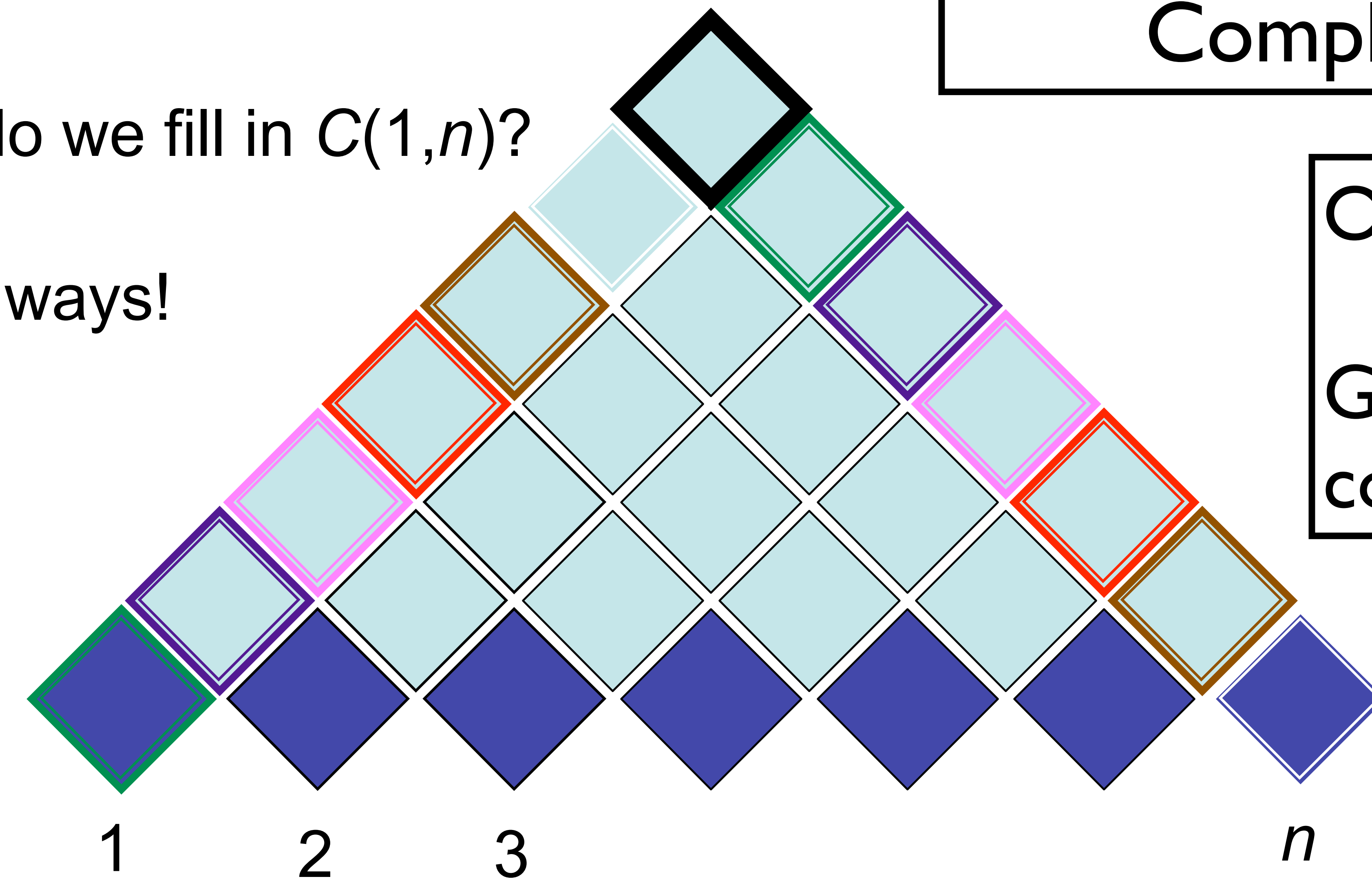


For cell [i,j]
For possible splitpoint k=(i+1)..(j-1):
For every B in [i,k] and C in [k,j],
If exists rule A -> B C,
add A to cell [i,j]

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
- Learning?
  - Fully supervised: need a treebank
  - Unsupervised: with EM

(P)CFG model, (P)CKY algorithm

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

- Weighted CKY: given PCFG and sentence w
  - => Viterbi parse
    $\arg\max_y P(y \mid w) = \arg\max_y P(y, w)$
- Inside-outside: Likelihood of sentence P(w)

- Parsers' computational efficiency
  - Grammar constant; pruning & heuristic search
  - $O(N^3)$ for CKY (ok? depends...)
  - O(N) [or so...]: left-to-right incremental algorithms

- Parsing model accuracy: still lots of ambiguity!!
  - PCFGs lack lexical information to resolve attachment decisions
  - Vanilla PCFG accuracy: 70-80%

Rules vs. Annotations

- In the old days: hand-built grammars. Difficult to scale.
- Annotation-driven sup. learning
  - ~1993: Penn Treebank
  - Construct PCFG (or whatever) with supervised learning
- (Cool open research: unsupervised learning?)

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

Treebanks

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

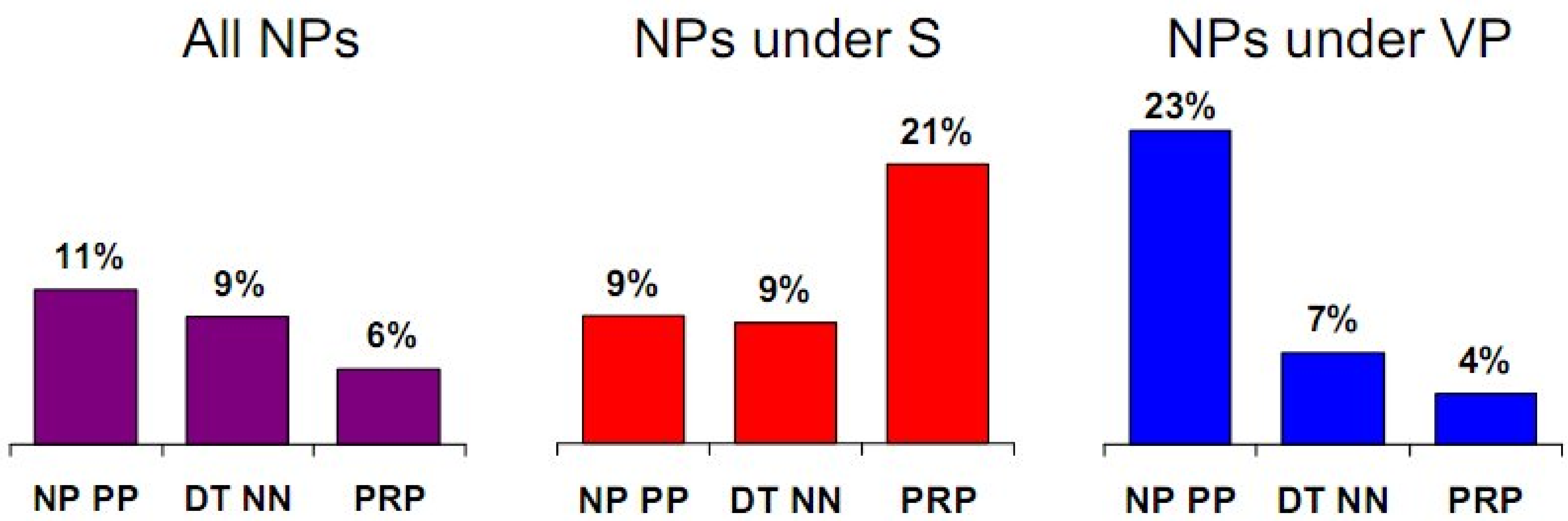
- Prague Treebank (syn+sem)
- many others...

- Know what you're getting!

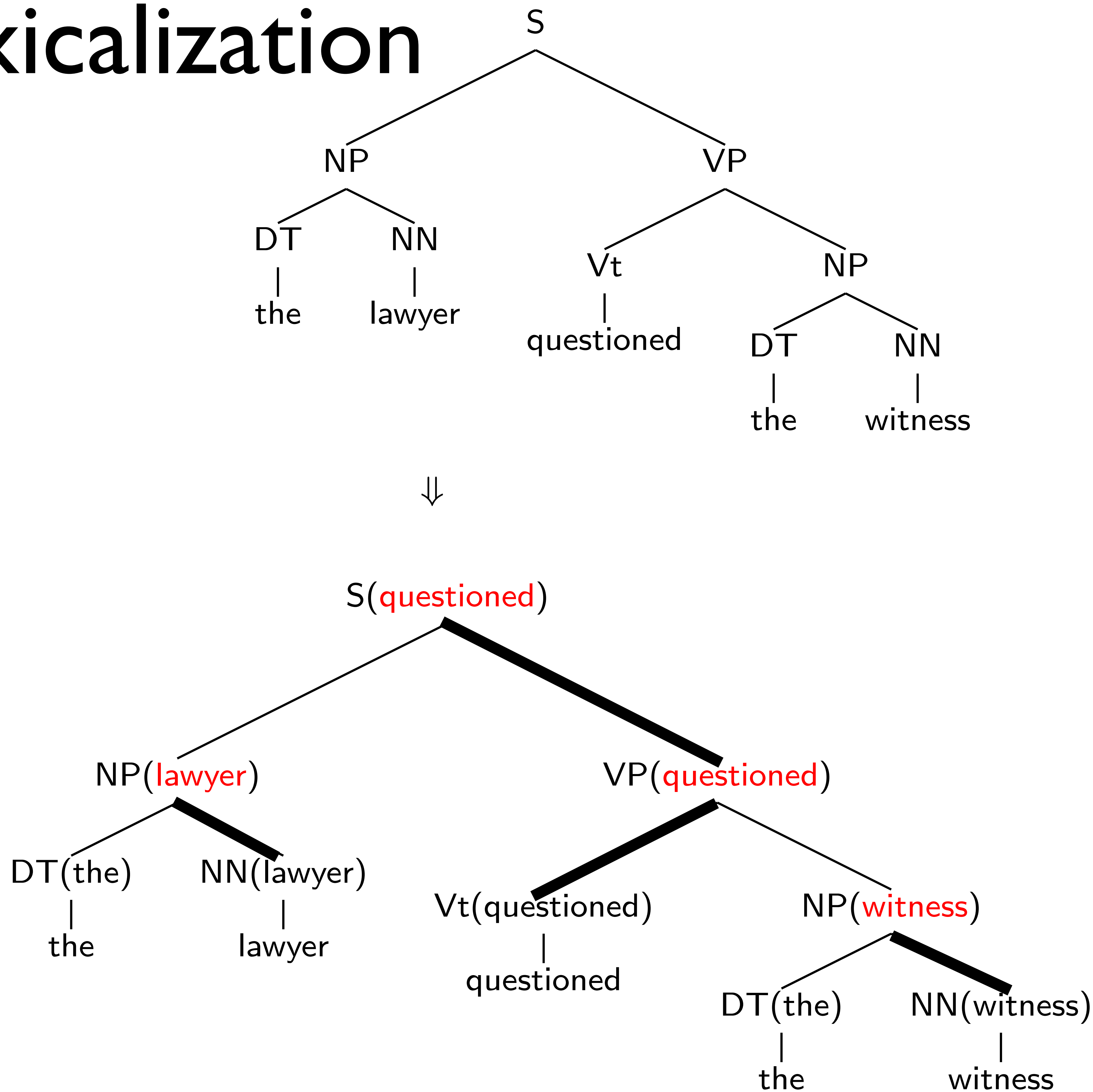
Ambiguities make parsing hard

- 1. Computationally: how to reuse work across combinatorially many trees?
- **2. How to make good attachment decisions?**
    - Enrich PCFG with
        - parent information: what's above me?
        - lexical information via head rules
            - VP[fight]: a VP headed by "fight"
    - (or better, word/phrase embedding-based generalizations: e.g. recurrent neural network grammars (RNNGs))

Parent annotation



Lexicalization



Head rules

- Idea: Every phrase has a head word
- Head rules: for every nonterminal in tree, choose one of its children to be its "head". This will define head words.
- Every nonterminal type has a different head rule; e.g. from Collins (1997):

- If parent is NP,
  - Search from right-to-left for first child that's NN, NNP, NNPS, NNS, NX, JJR
  - Else: search left-to-right for first child which is NP