Context-free Grammar

CS 690N, Spring 2017

Advanced Natural Language Processing

http://people.cs.umass.edu/~brenocon/anlp2017/

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

- Syntax: how do words structurally combine to form sentences and meaning?

- Representations
  - Constituents
    - [the big dogs] chase cats
    - [colorless green clouds] chase cats
  - Dependencies
    - The **dog chased** the cat.
    - My **dog**, a big old one, **chased** the cat.

- Idea of a *grammar (G)*: global template for how sentences / utterances / phrases *w* are formed, via latent syntactic structure *y*
  - Linguistics: what do G and P(w,y | G) look like?
  - Generation: score with, or sample from, P(w, y | G)
  - Parsing: predict P(y | w, G)

Is language context-free?

[Examples from Eisenstein (2017)]

Is language context-free?

- Regular language: repetition of repeated structures

[Examples from Eisenstein (2017)]

Is language context-free?

- Regular language: repetition of repeated structures
  - e.g. Justeson and Katz (1995)'s noun phrase pattern: (Noun | Adj)* Noun (Prep Det? (Noun | Adj)* Noun)*

[Examples from Eisenstein (2017)]

Is language context-free?

- Regular language: repetition of repeated structures
  - e.g. Justeson and Katz (1995)'s noun phrase pattern: (Noun | Adj)* Noun (Prep Det? (Noun | Adj)* Noun)*
- Context-free: hierarchical recursion

[Examples from Eisenstein (2017)]

Is language context-free?

- Regular language: repetition of repeated structures
  - e.g. Justeson and Katz (1995)'s noun phrase pattern: (Noun | Adj)* Noun (Prep Det? (Noun | Adj)* Noun)*
- Context-free: hierarchical recursion
- Center-embedding: classic theoretical argument for CFG vs. regular languages

[Examples from Eisenstein (2017)]

Is language context-free?

- Regular language: repetition of repeated structures
  - e.g. Justeson and Katz (1995)'s noun phrase pattern: (Noun | Adj)* Noun (Prep Det? (Noun | Adj)* Noun)*
- Context-free: hierarchical recursion
- Center-embedding: classic theoretical argument for CFG vs. regular languages
  - (10.1) The cat is fat.

[Examples from Eisenstein (2017)]

Is language context-free?

- Regular language: repetition of repeated structures
  - e.g. Justeson and Katz (1995)'s noun phrase pattern: (Noun | Adj)* Noun (Prep Det? (Noun | Adj)* Noun)*
- Context-free: hierarchical recursion
- Center-embedding: classic theoretical argument for CFG vs. regular languages
  - (10.1) The cat is fat.
  - (10.2) The cat that the dog chased is fat.

[Examples from Eisenstein (2017)]

Is language context-free?

- Regular language: repetition of repeated structures
  - e.g. Justeson and Katz (1995)'s noun phrase pattern: (Noun | Adj)* Noun (Prep Det? (Noun | Adj)* Noun)*
- Context-free: hierarchical recursion
- Center-embedding: classic theoretical argument for CFG vs. regular languages
  - (10.1) The cat is fat.
  - (10.2) The cat that the dog chased is fat.
  - (10.3) *The cat that the dog is fat.

[Examples from Eisenstein (2017)]

Is language context-free?

- Regular language: repetition of repeated structures
  - e.g. Justeson and Katz (1995)'s noun phrase pattern:
    (Noun | Adj)* Noun (Prep Det? (Noun | Adj)* Noun)*
- Context-free: hierarchical recursion
- Center-embedding: classic theoretical argument for CFG vs. regular languages
  - (10.1) The cat is fat.
  - (10.2) The cat that the dog chased is fat.
  - (10.3) *The cat that the dog is fat.
  - (10.4) The cat that the dog that the monkey kissed chased is fat.

[Examples from Eisenstein (2017)]

Is language context-free?

- Regular language: repetition of repeated structures
  - e.g. Justeson and Katz (1995)'s noun phrase pattern: (Noun | Adj)* Noun (Prep Det? (Noun | Adj)* Noun)*
- Context-free: hierarchical recursion
- Center-embedding: classic theoretical argument for CFG vs. regular languages
  - (10.1) The cat is fat.
  - (10.2) The cat that the dog chased is fat.
  - (10.3) *The cat that the dog is fat.
  - (10.4) The cat that the dog that the monkey kissed chased is fat.
  - (10.5) *The cat that the dog that the monkey chased is fat.

[Examples from Eisenstein (2017)]

Is language context-free?

- Regular language: repetition of repeated structures
  - e.g. Justeson and Katz (1995)'s noun phrase pattern: (Noun | Adj)* Noun (Prep Det? (Noun | Adj)* Noun)*
- Context-free: hierarchical recursion
- Center-embedding: classic theoretical argument for CFG vs. regular languages
  - (10.1) The cat is fat.
  - (10.2) The cat that the dog chased is fat.
  - (10.3) *The cat that the dog is fat.
  - (10.4) The cat that the dog that the monkey kissed chased is fat.
  - (10.5) *The cat that the dog that the monkey chased is fat.

- Competence vs. Performance?

[Examples from Eisenstein (2017)]

Hierarchical view of syntax

- "a Sentence made of Noun Phrase followed by a Verb Phrase"

a. S

NP

$$\left\{\begin{array}{l}\text{John}\\ \text{the man}\\ \text{the elderly janitor}\end{array}\right\}$$

VP

$$\left\{\begin{array}{l}\text{arrived}\\ \text{ate an apple}\\ \text{looked at his watch}\end{array}\right\}$$

b. S $\rightarrow$ NP VP (1)

[From Phillips (2003)]

Is language context-free?

- Practical examples where nesting seems like a useful explanation
  - *The **processor** **has** 10 million times fewer transistors on it than todays typical micro- processors, **runs** much more slowly, and **operates** at five times the voltage...*
  - $$\begin{aligned} S &\rightarrow NN\ VP \\ VP &\rightarrow VP3S \mid VPN3S \mid \ldots \\ VP3S &\rightarrow VP3S, VP3S, \text{ and } VP3S \mid VBZ \mid VBZ\ NP \mid \ldots \end{aligned}$$

[Examples from Eisenstein (2017)]

- Regular language <=> RegEx <=> paths in finite state machine
- Context-free language <=> CFG <=> derivations in pushdown automaton

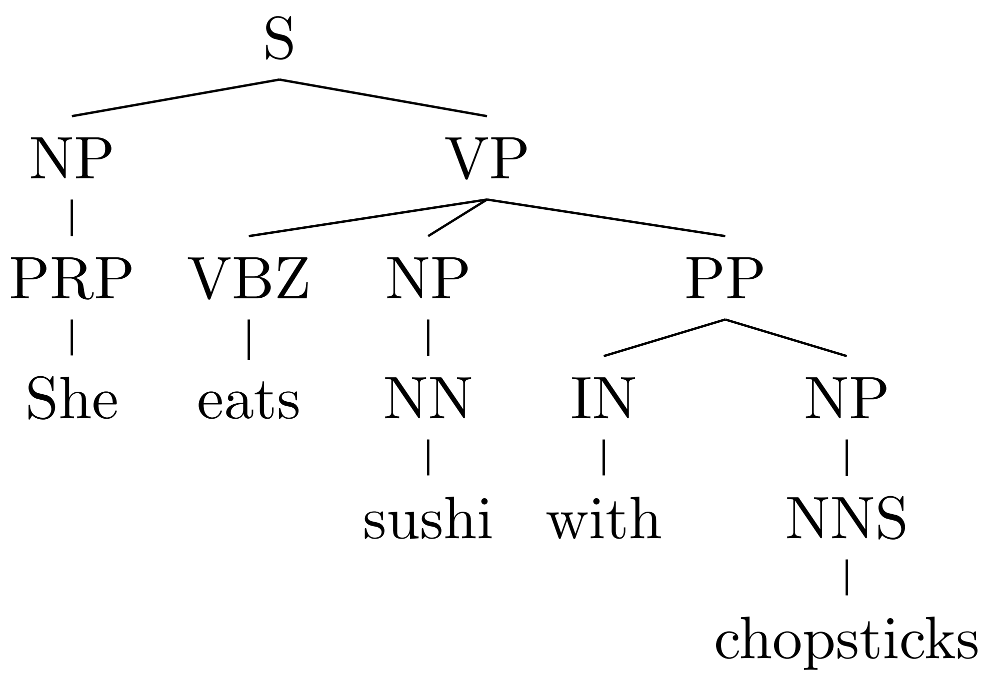
- A context-free grammar is a 4-tuple:

N	a set of non-terminals
Σ	a set of terminals (distinct from N)
R	a set of productions, each of the form $A \rightarrow \beta$, where $A \in N$ and $\beta \in (\Sigma \cup N)^*$
S	a designated start symbol

- Derivation: sequence of rewrite steps from S to a string (sequence of terminals, i.e. words)
- Yield: the final string

- A CFG is a "boolean language model"
- A probabilistic CFG is a probabilistic language model:
  - Every production rule has a probability; defines prob dist. over strings.

Example

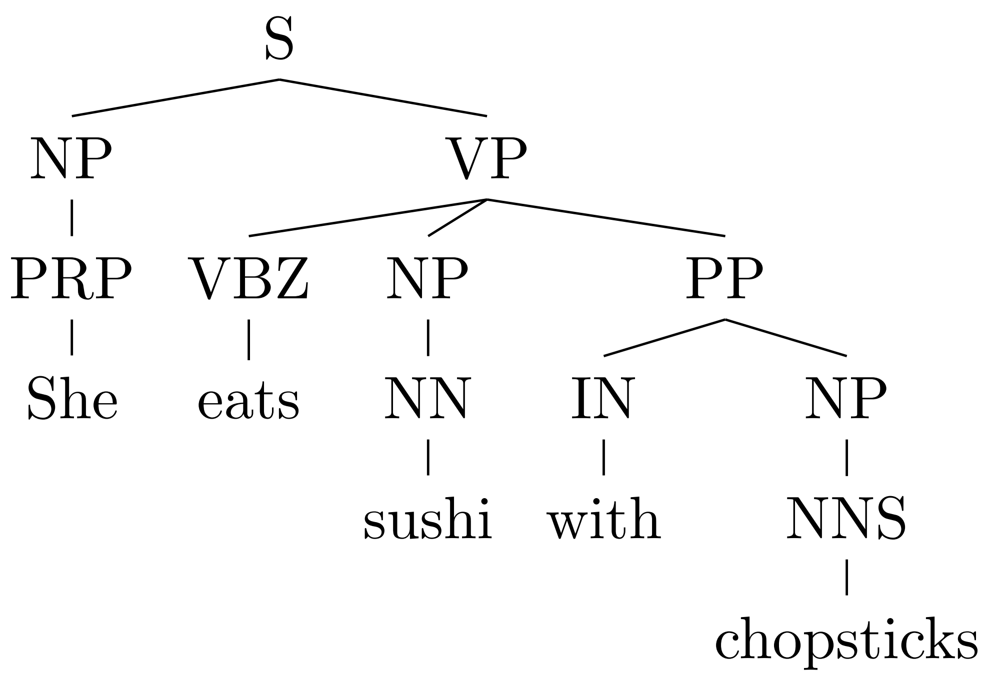


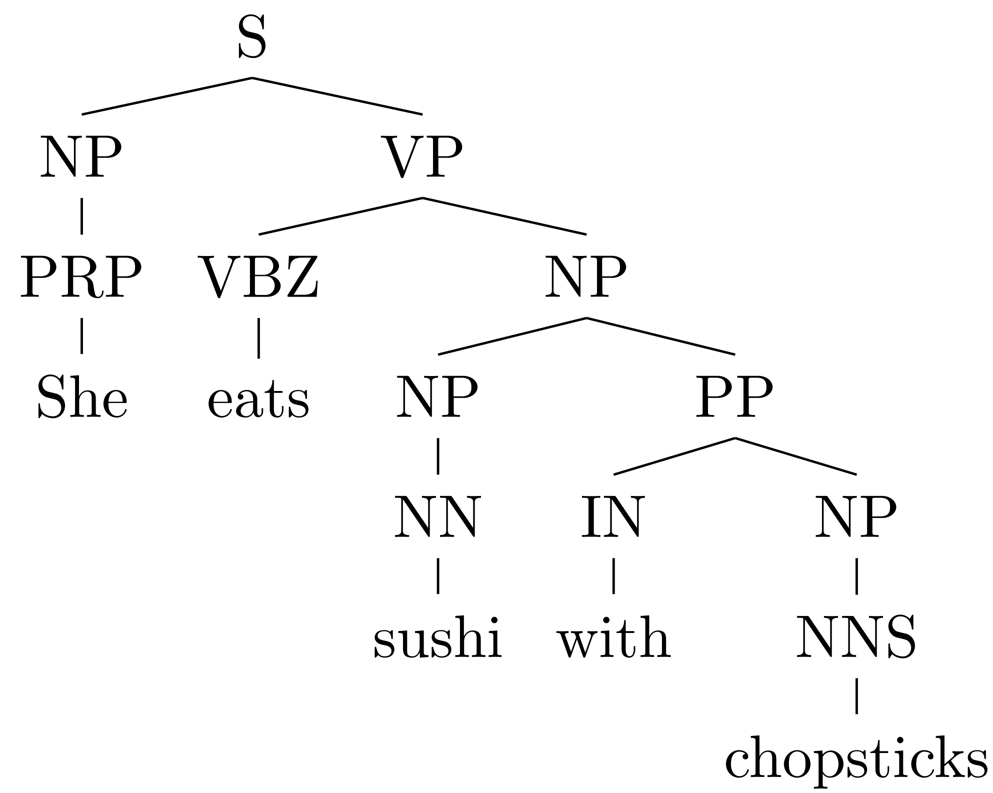
(S(NP(PRP *She*)(VP(VBZ *eats*)
(NP(NN *sushi*))
(PP(IN *with*)(NP(NNS *chopsticks*))))))

- All useful grammars are *ambiguous*: multiple derivations with same yield
- [Parse tree representations: Nested parens *or* non-terminal spans]

[Examples from Eisenstein (2017)]

Example





(S(NP(PRP *She*)(VP(VBZ *eats*)
(NP(NN *sushi*))
(PP(IN*with*)(NP(NNS *chopsticks*))))))

(S(NP(PRP *She*)(VP(VBZ *eats*)
(NP(NP(NN *sushi*))(PP (IN*with*)(NP(NNS *chopsticks*)))))))

- All useful grammars are *ambiguous*: multiple derivations with same yield
- [Parse tree representations: Nested parens *or* non-terminal spans]

[Examples from Eisenstein (2017)]

Constituents

- Constituent tree/parse is one representation of sentence's syntax. What should be considered a constituent, or constituents of the same category?
  - Substitution tests
  - Pronoun substitution
  - Coordination tests

- Simple grammar of English
  - Must balance *overgeneration* versus *undergeneration*
  - Noun phrases
  - NP modification: adjectives, PPs
  - Verb phrases
  - Coordination...

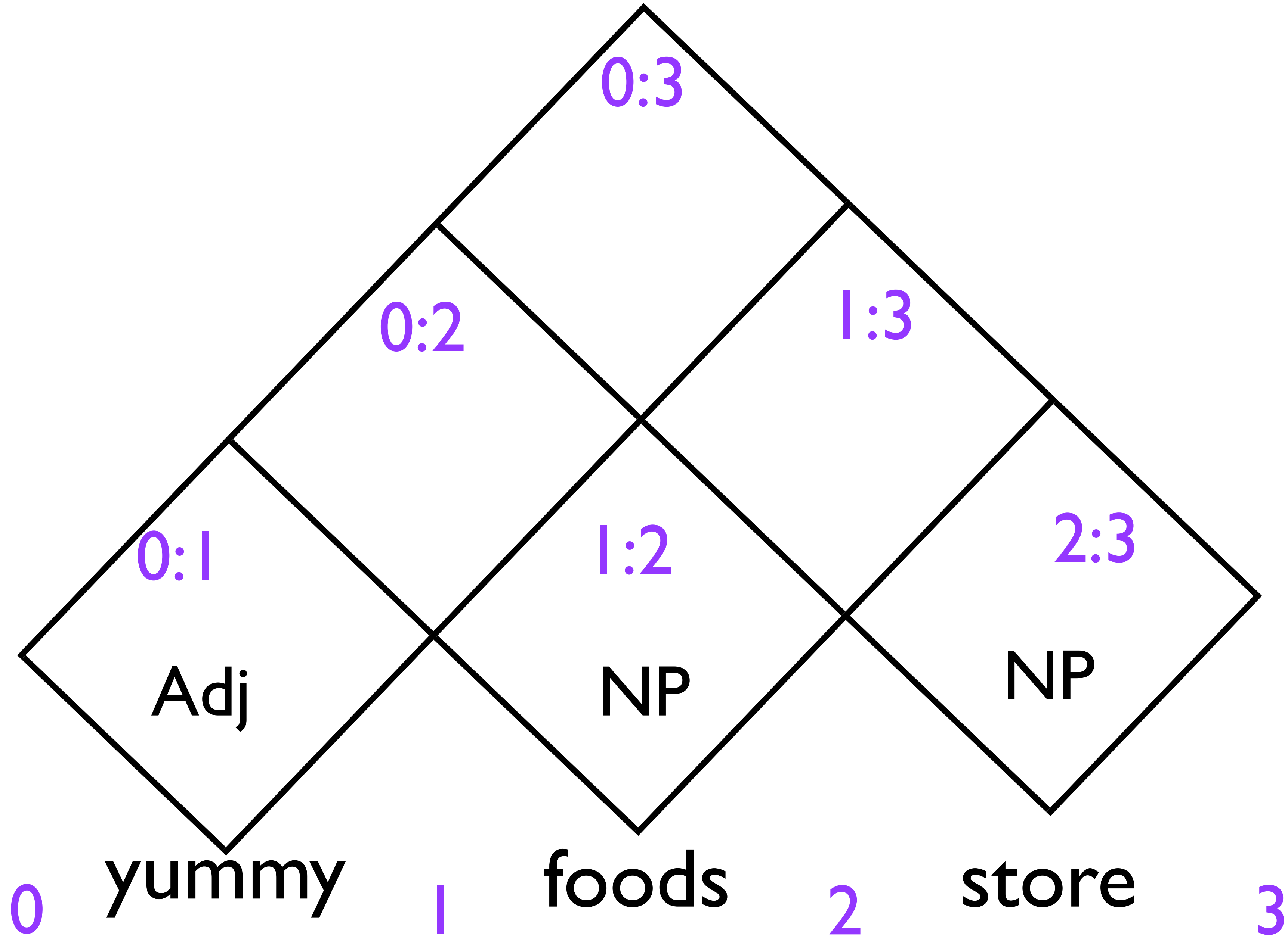
Parsing with a CFG

- 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



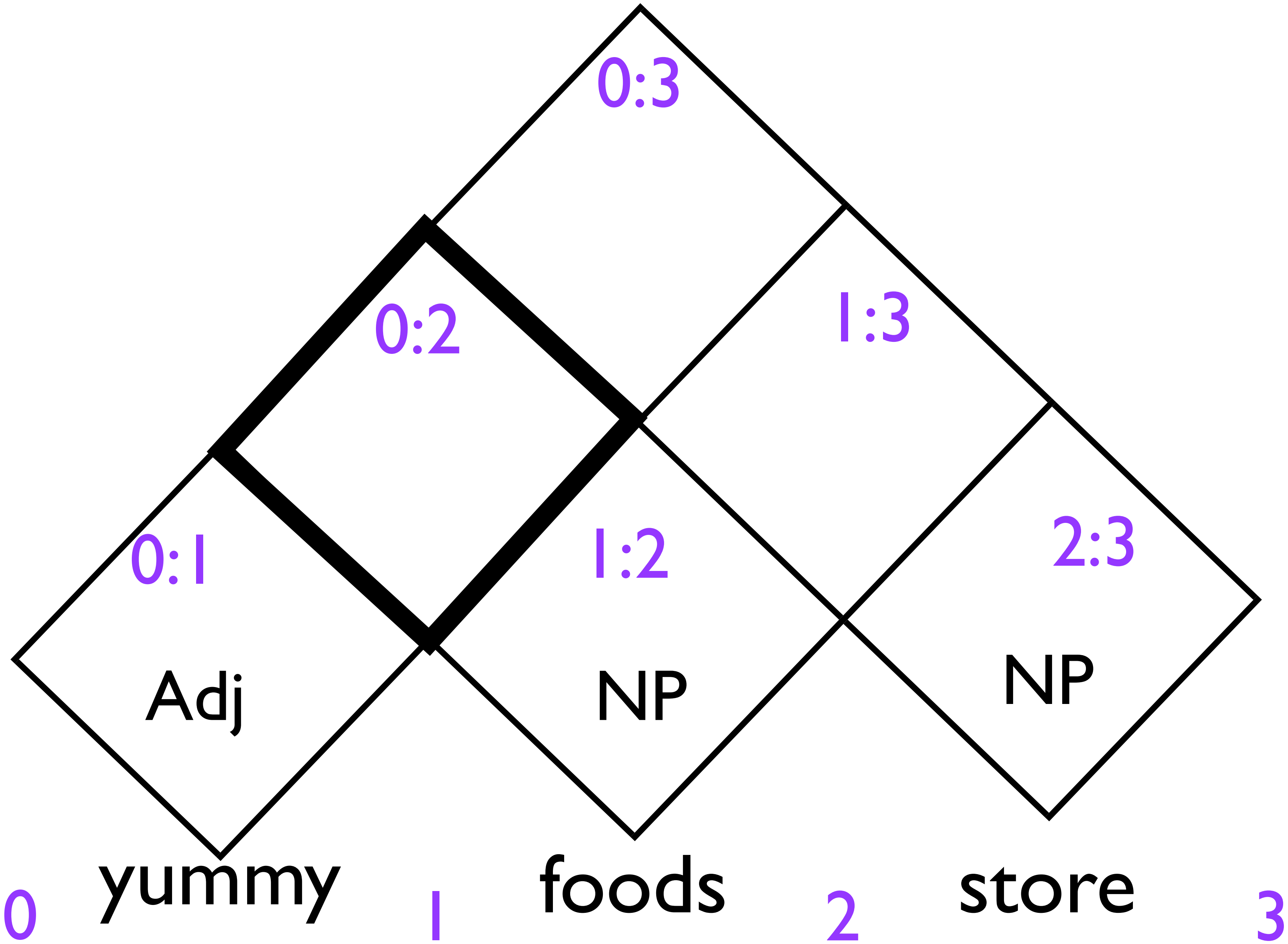
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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



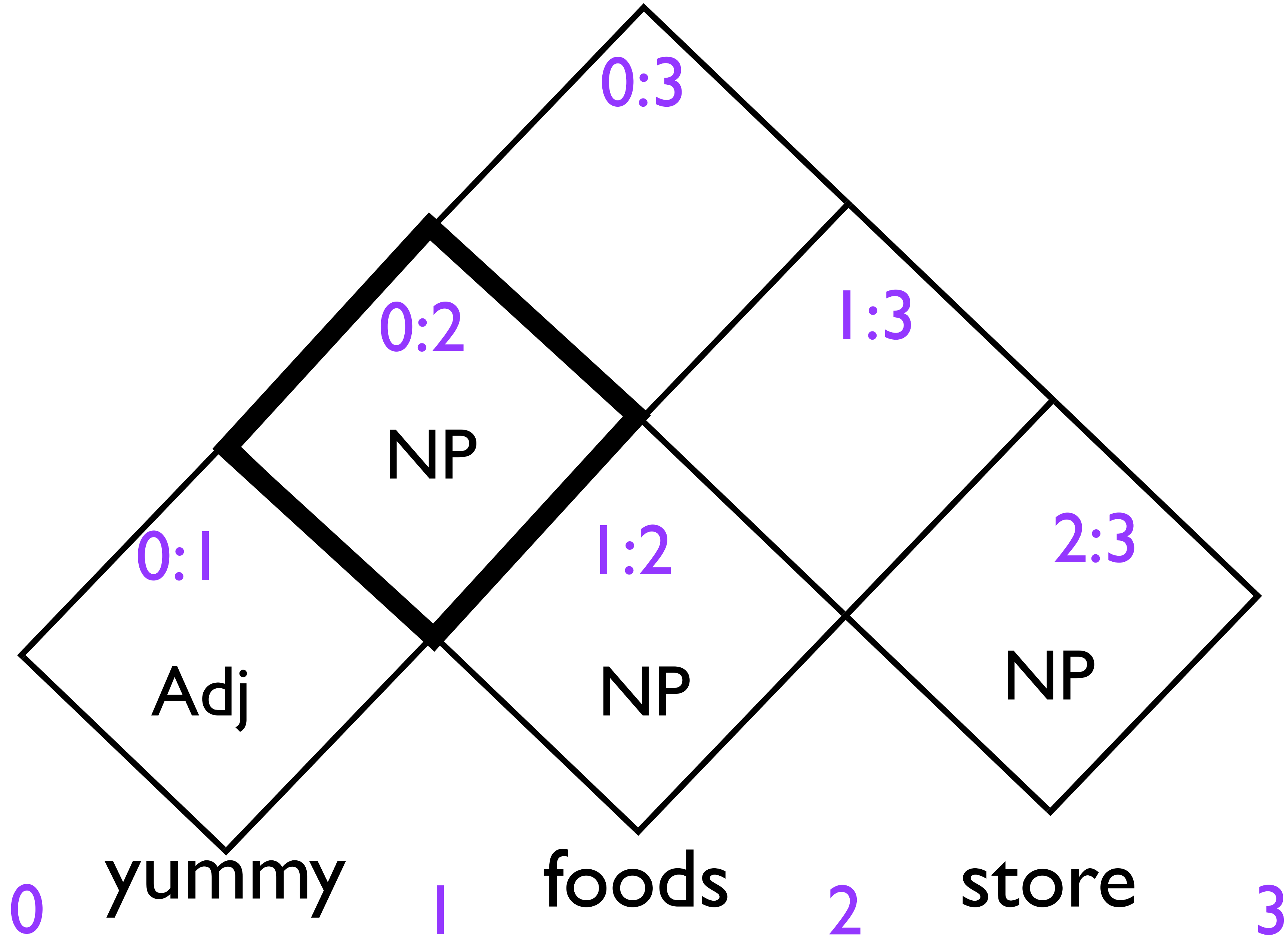
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



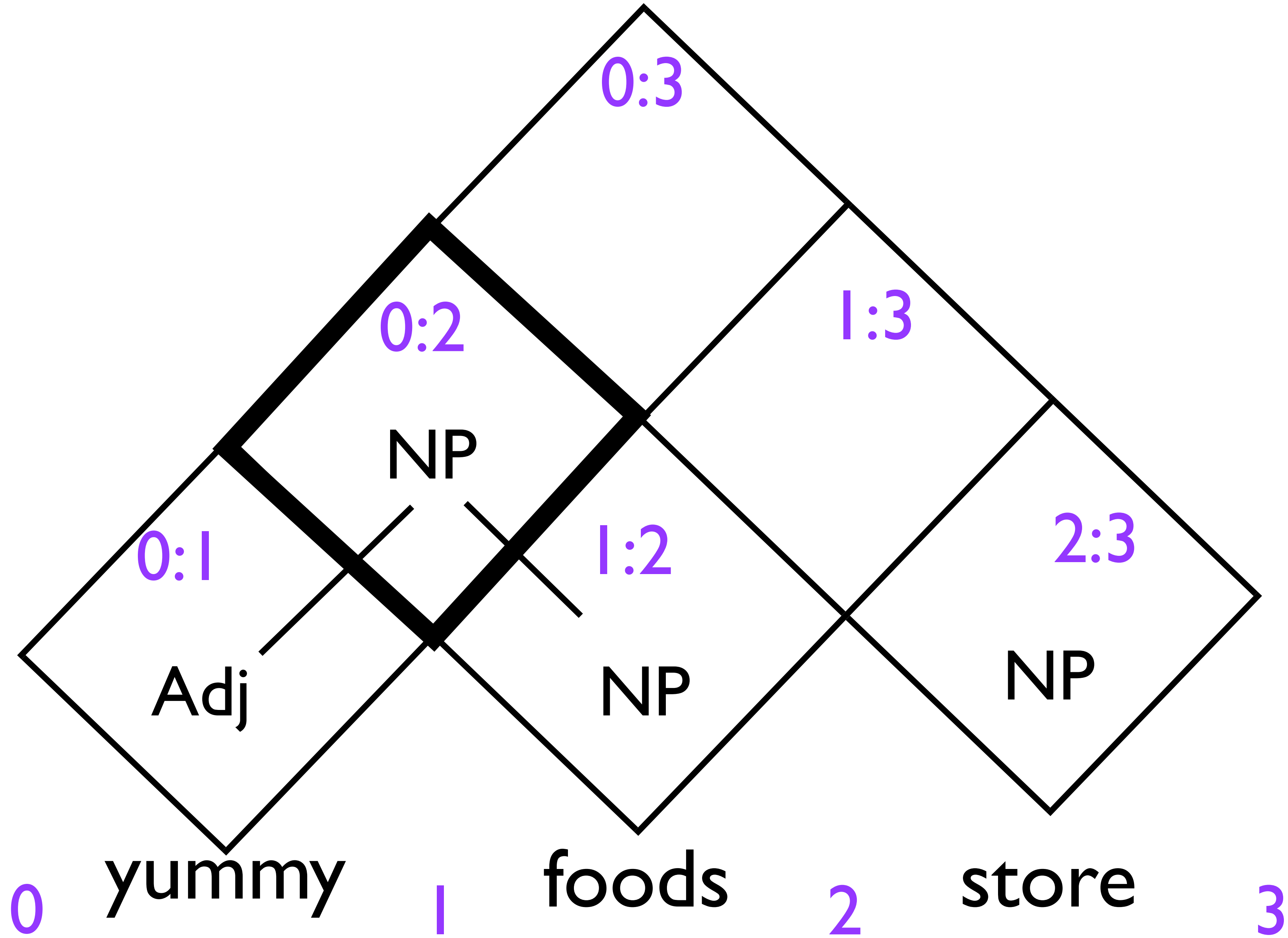
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



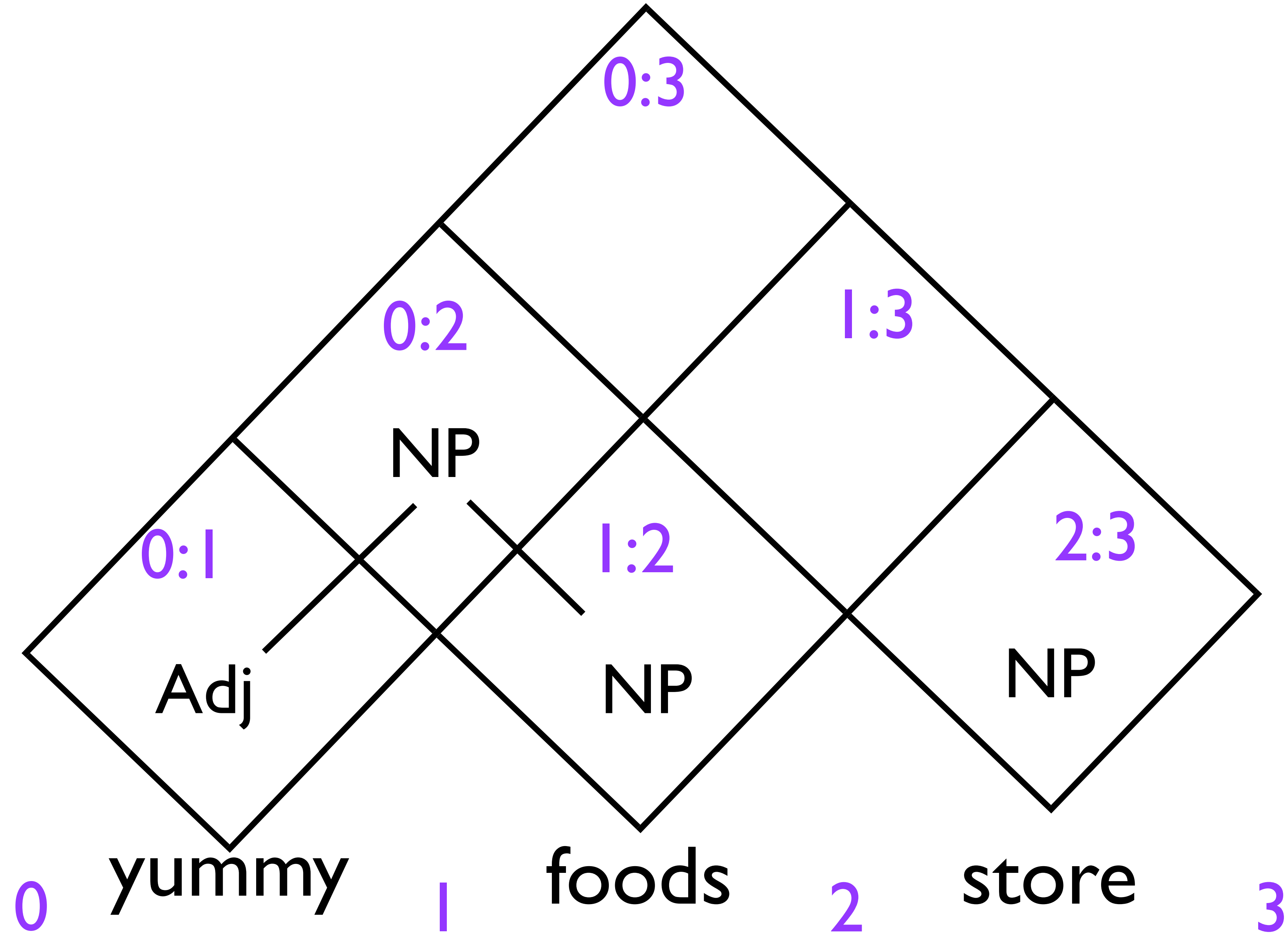
For cell [i,j] (loop through them bottom-up)
 For possible splitpoint k=(i+1)..(j-1):
  For every B in [i,k] and C in [k,j],
   If exists rule A -> B C,
    add A to cell [i,j] (*Recognizer*)
    ... or ...
    add (A,B,C, k) to cell [i,j] (*Parser*)

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



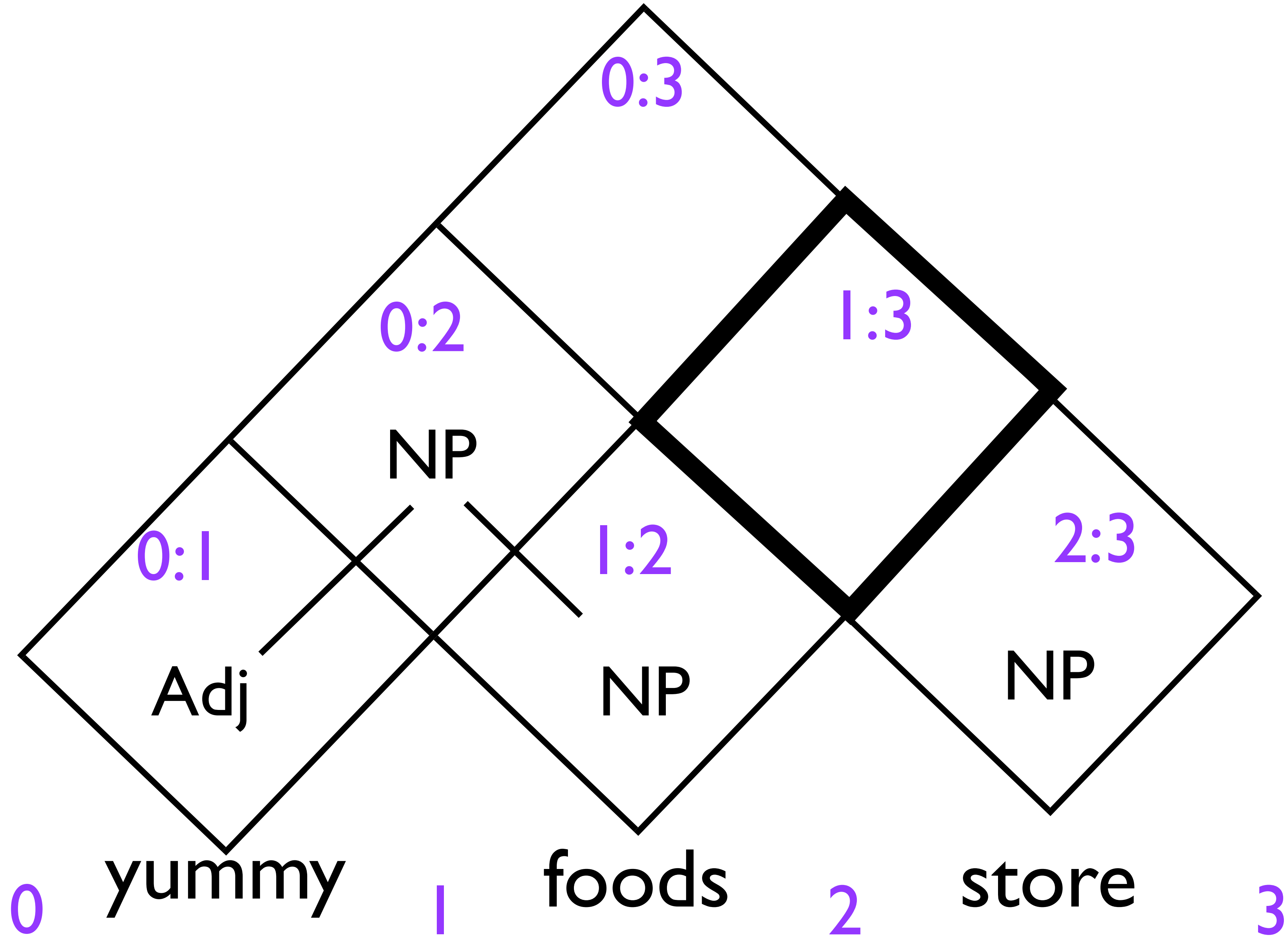
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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



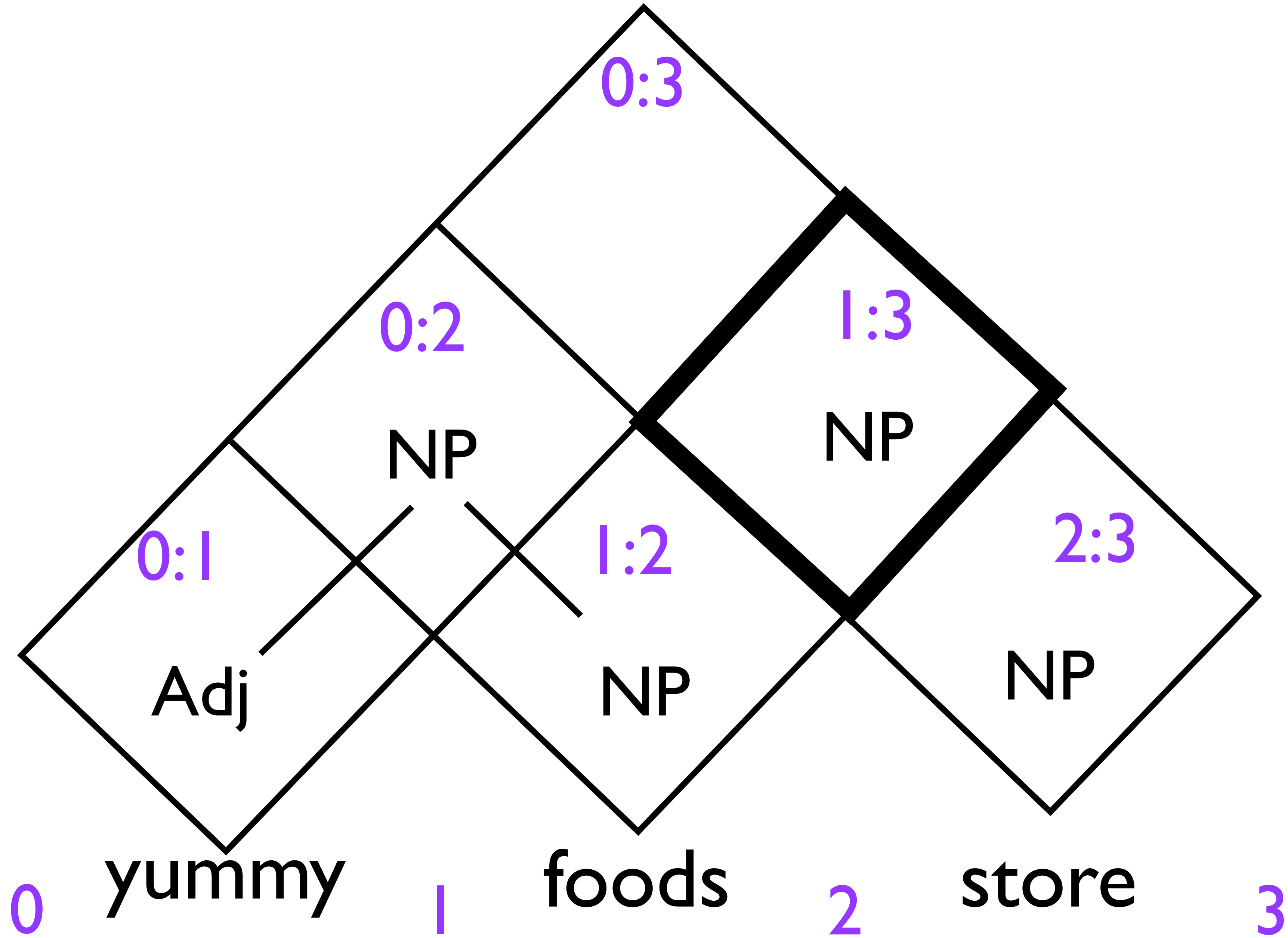
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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



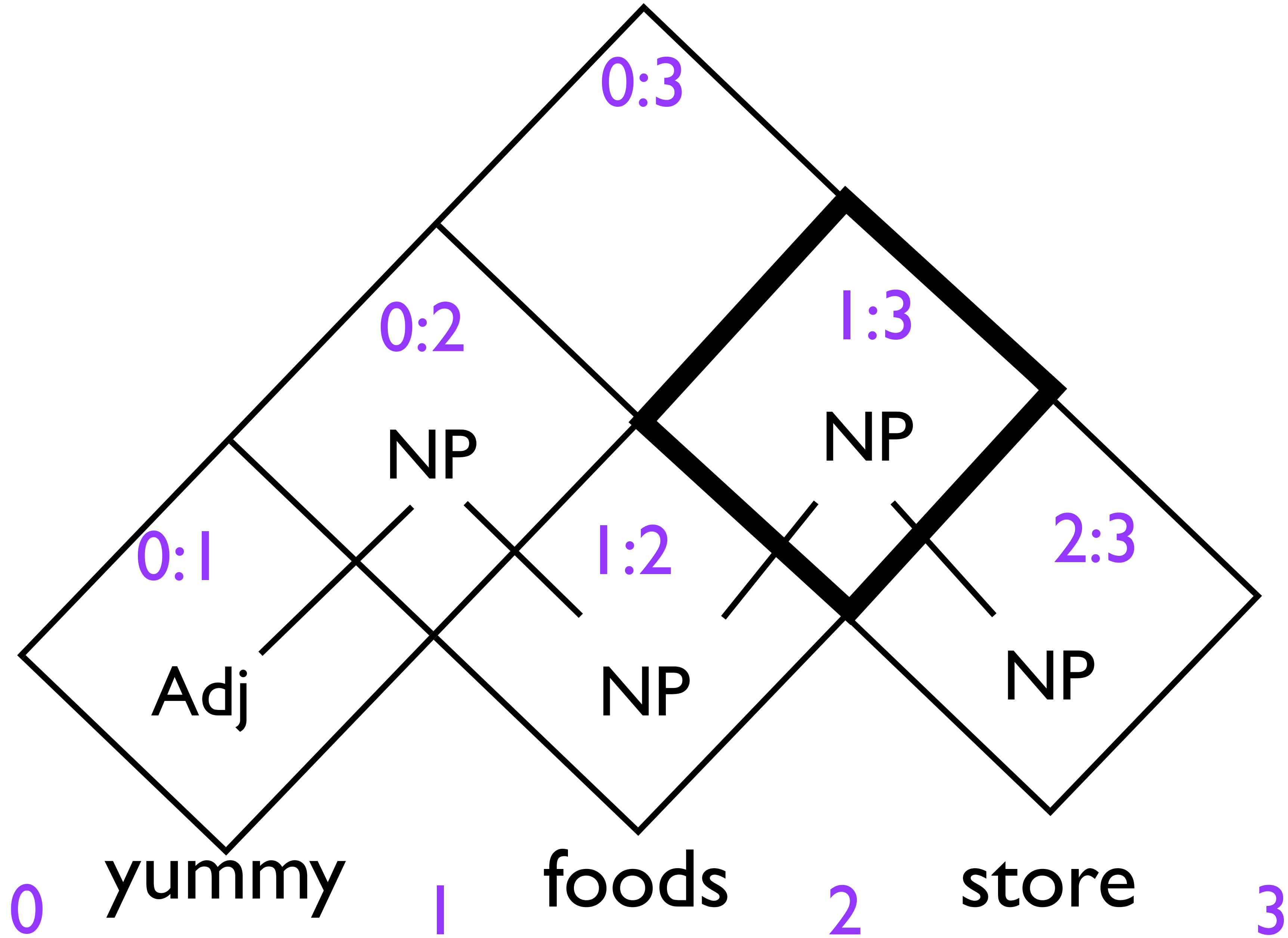
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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



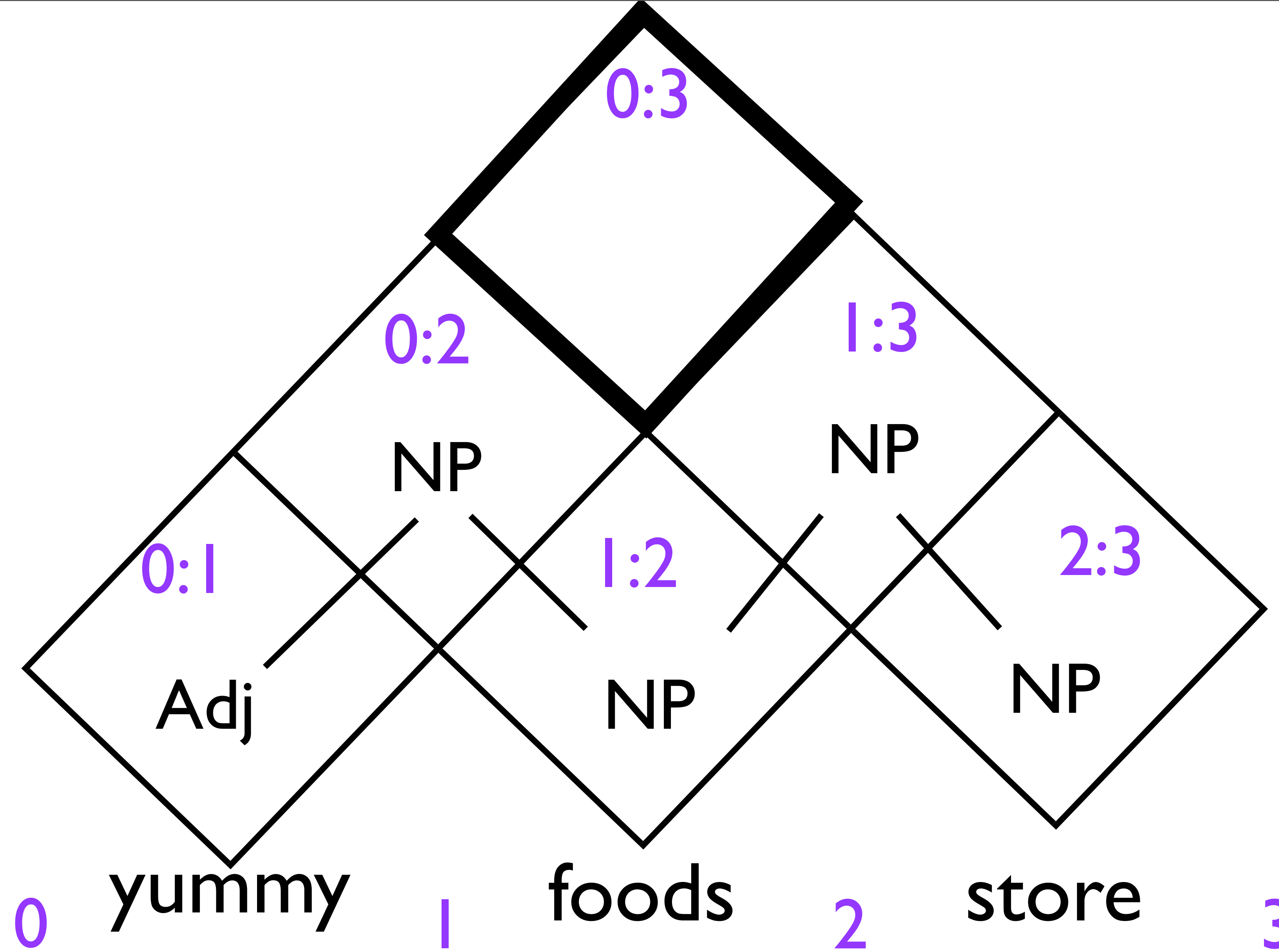
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



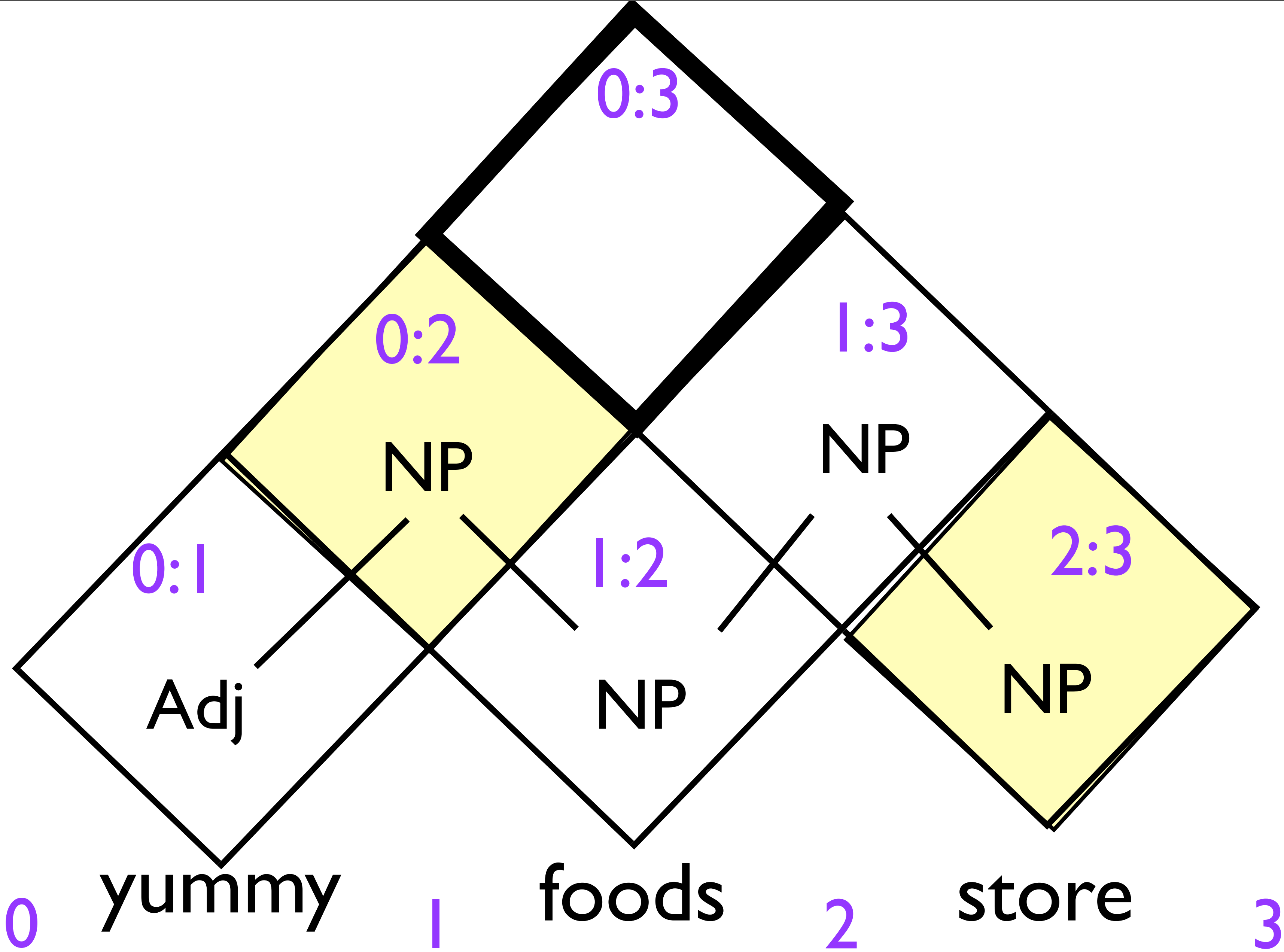
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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



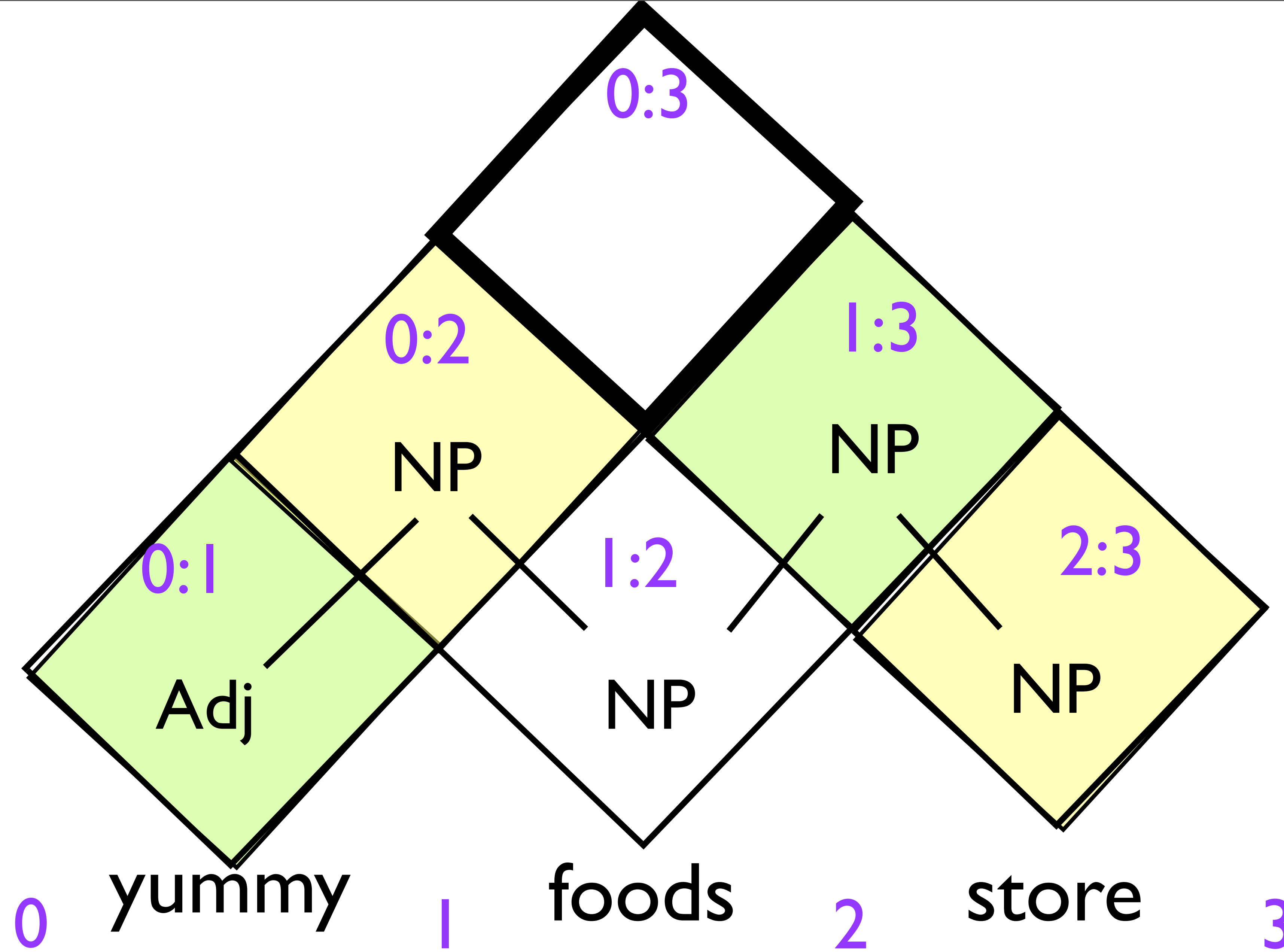
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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



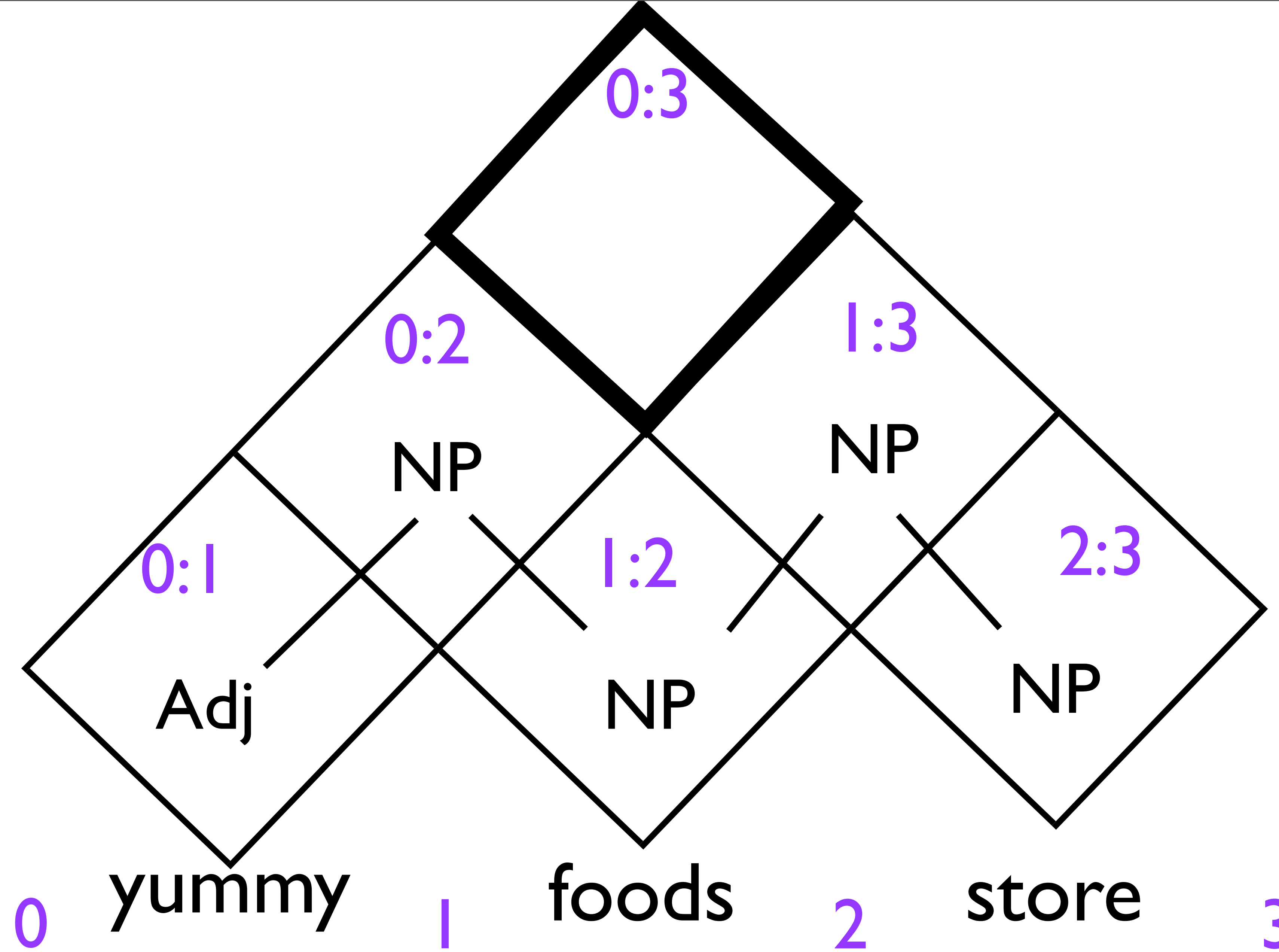
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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

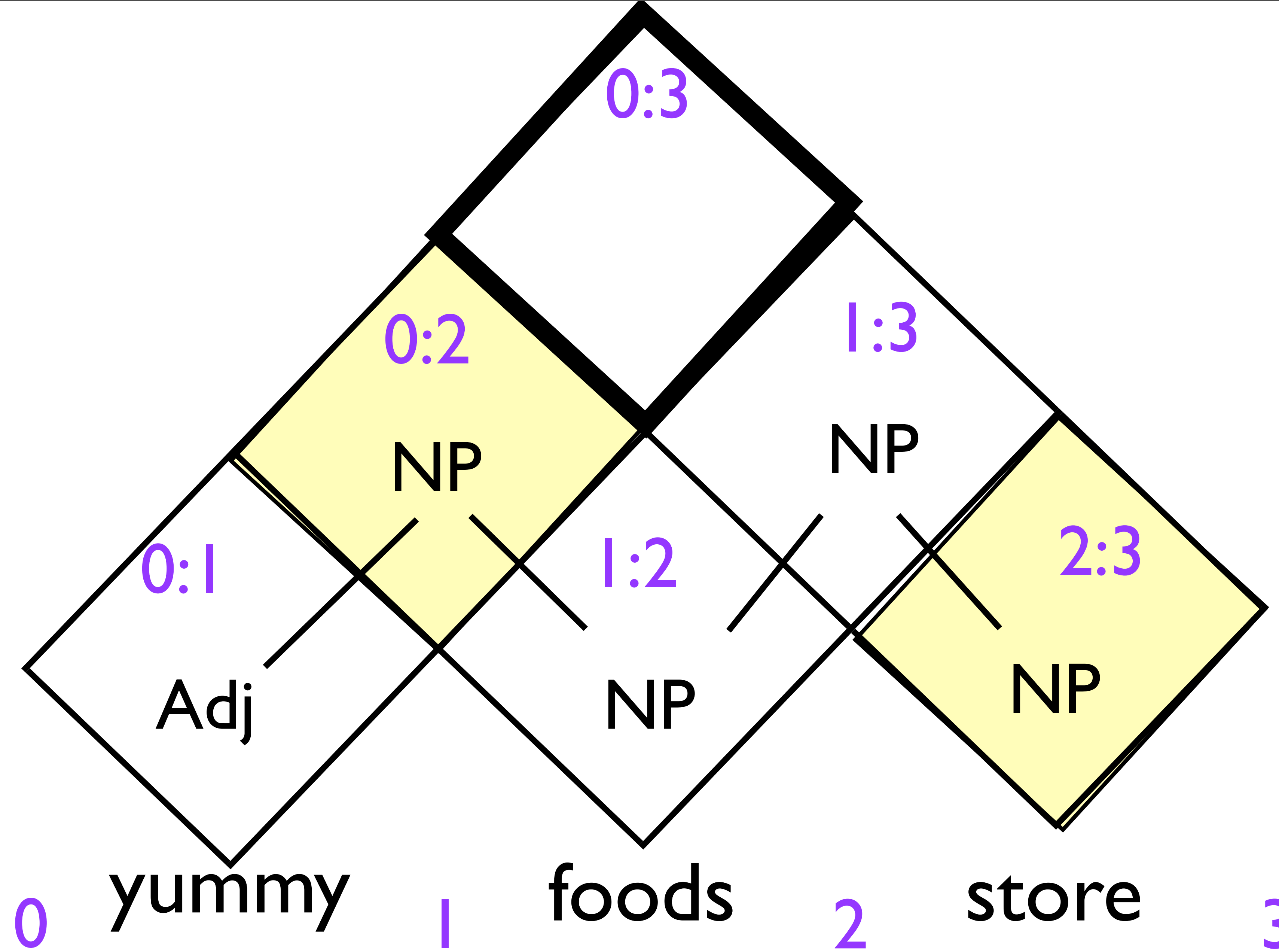


For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

Recognizer: per span, record list of possible nonterminals

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

CKY



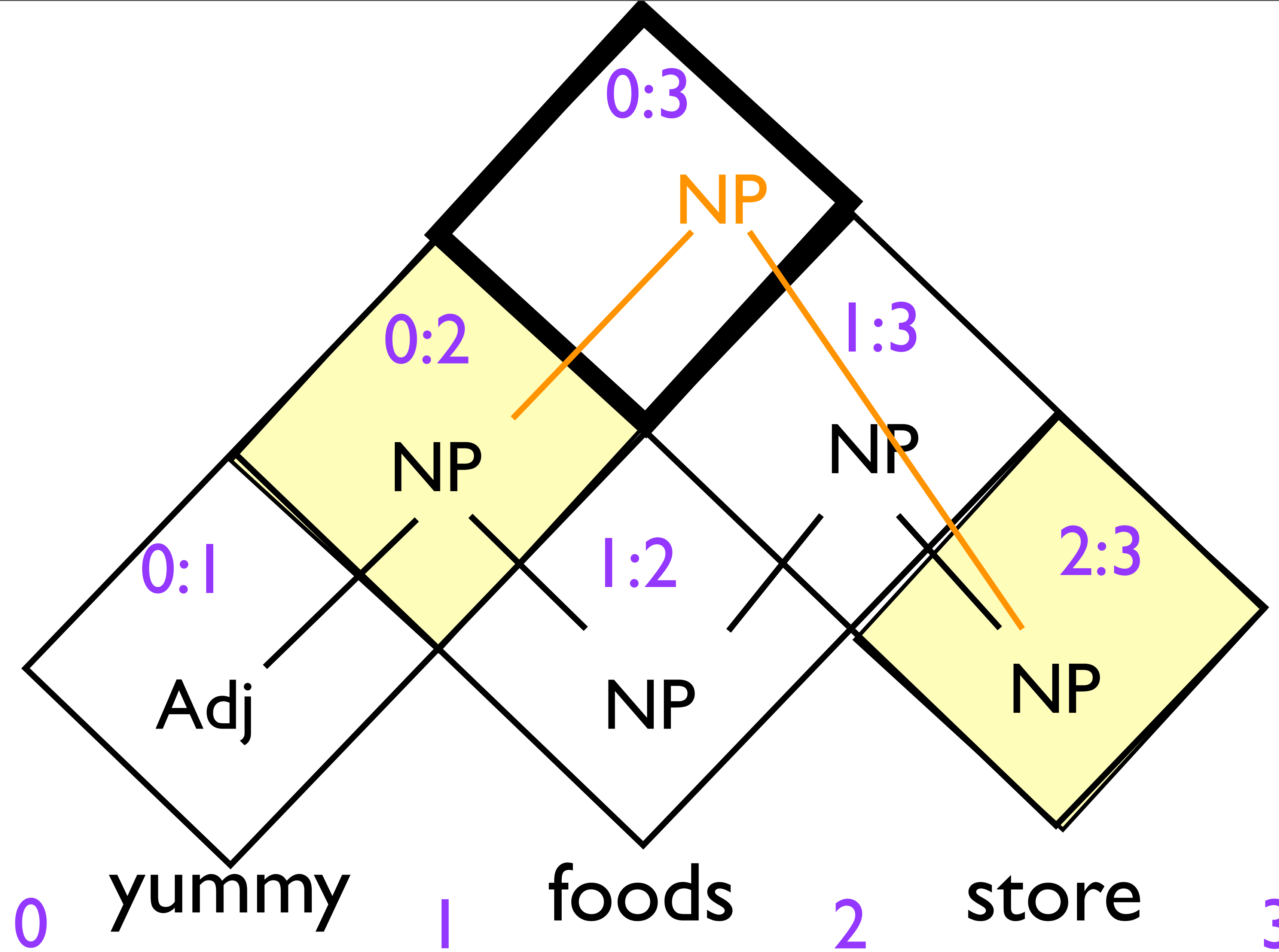
Grammar
Adj -> yummy
NP -> foods
NP -> store
NP -> NP NP
NP -> Adj NP

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

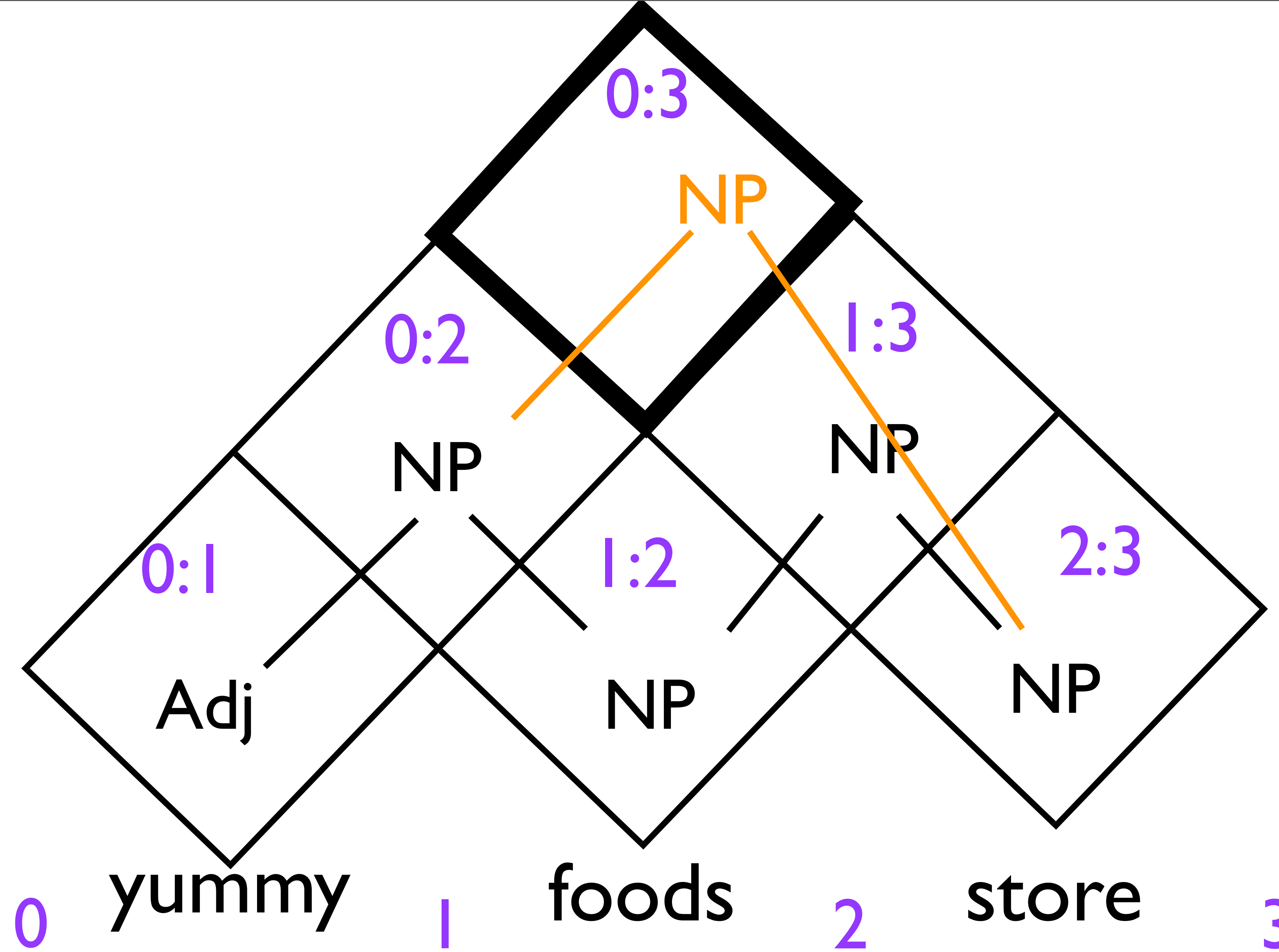
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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



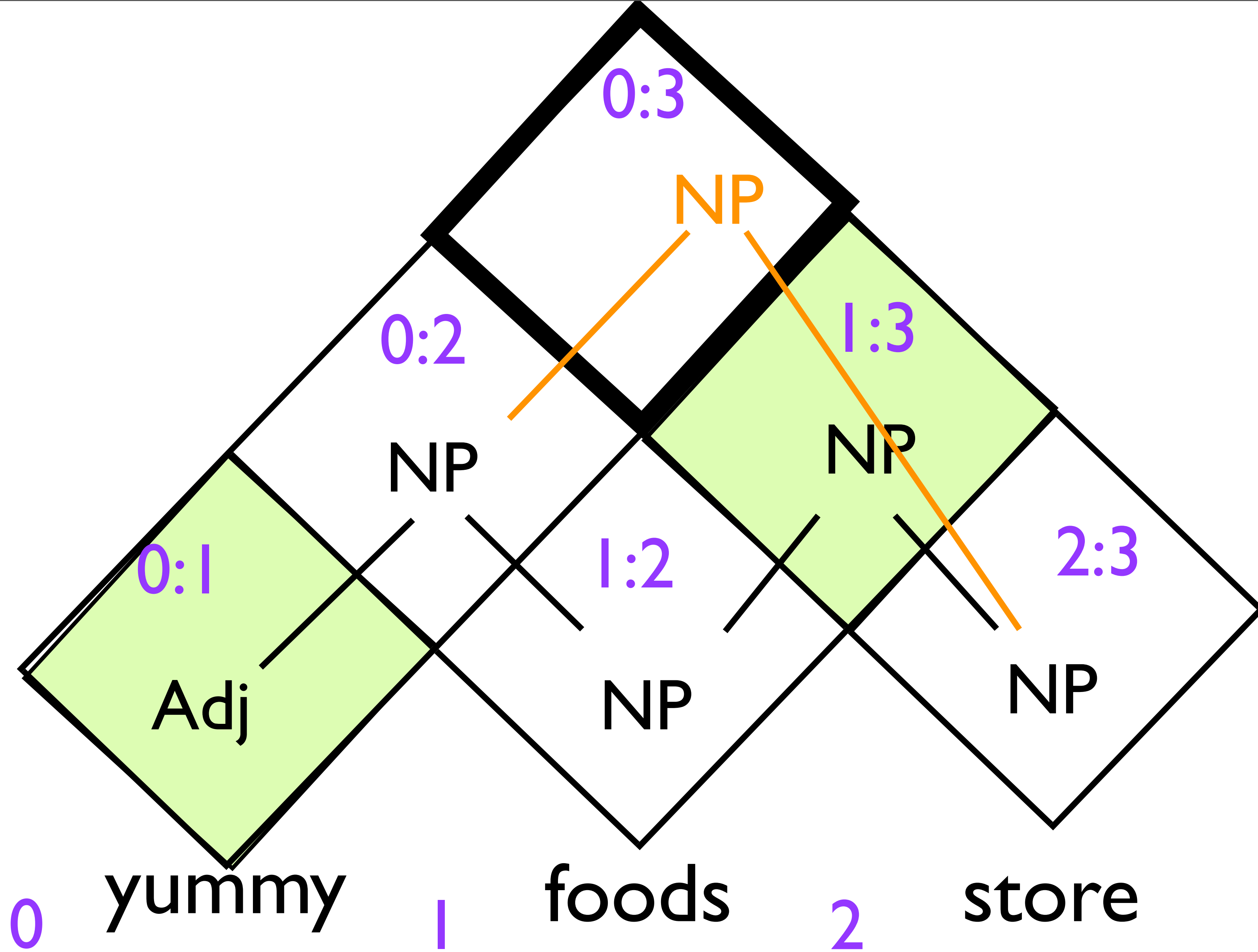
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

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



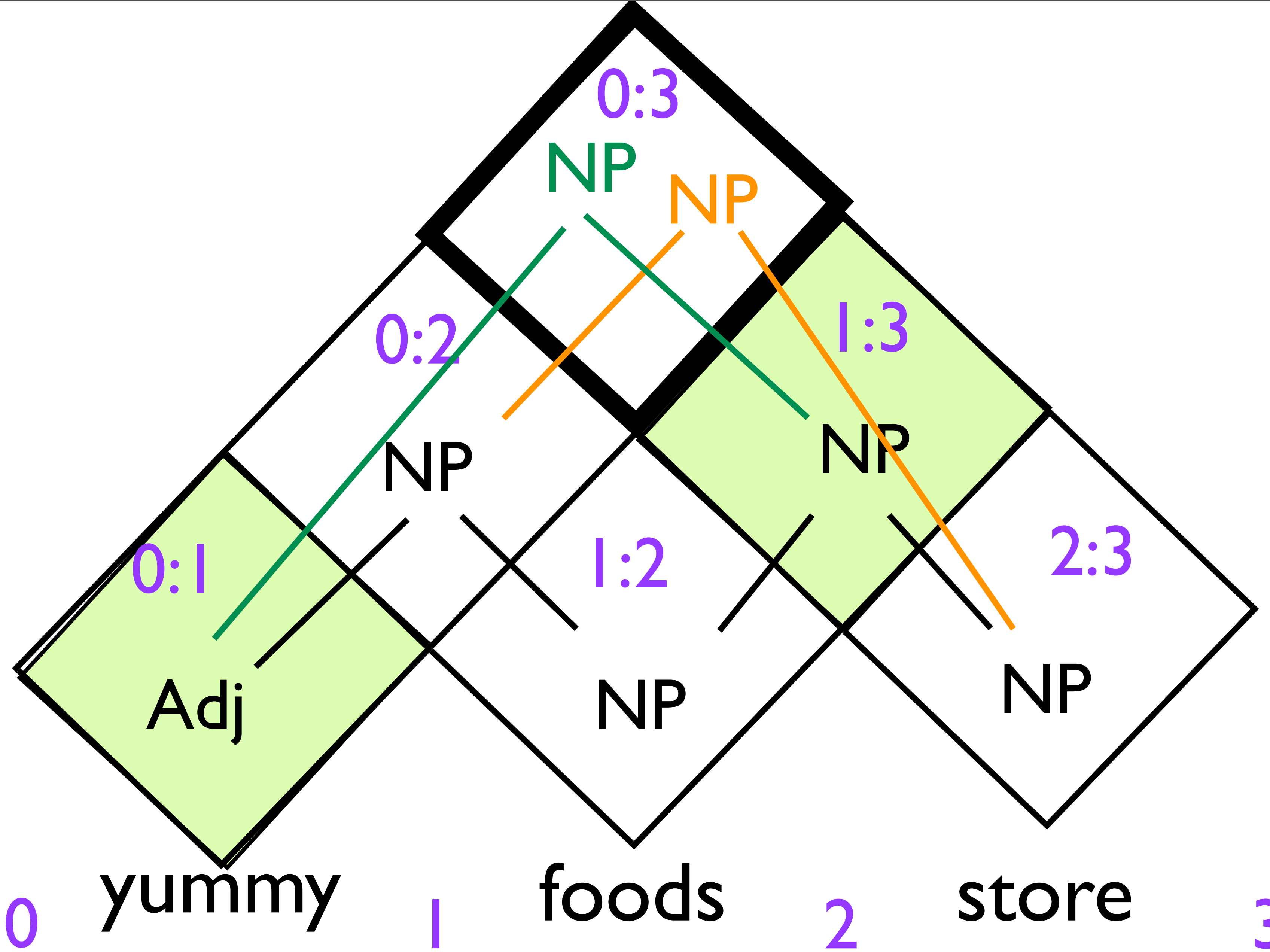
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j] (*Recognizer*)
                ... or ...
                add (A,B,C, k) to cell [i,j] (*Parser*)

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.