# Syntax (I)

CS 685, Spring 2021

Advanced Topics in Natural Language Processing

http://brenocon.com/cs685

https://people.cs.umass.edu/~brenocon/cs685_s21/

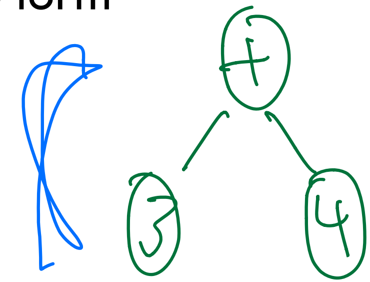

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst


3+4 3+4

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



- Representations → Tradeoffs!
  - 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.

amod

red cat

Blorps chase cats!

I saw [dogs and cats]

"word"

- 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 — VP

$$\left\{\begin{array}{l}\text{John}\\ \text{the man}\\ \text{the elderly janitor}\end{array}\right\} \left\{\begin{array}{l}\text{arrived}\\ \text{ate an apple}\\ \text{looked at his watch}\end{array}\right\}$$

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



[From Phillips (2003)]

# Is language context-free?

- Seems useful to explain e.g. nesting and agreement
  - *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

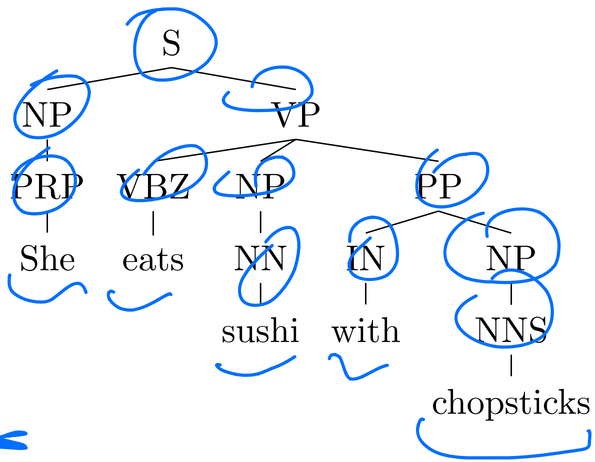
  - A context-free grammar is a 4-tuple:

| | |
|---|---|
| $N$ | a set of non-terminals |
| $\Sigma$ | a set of terminals (distinct from $N$) words |
| $R$ | a set of productions, each of the form $A \to \beta$, 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*))))))

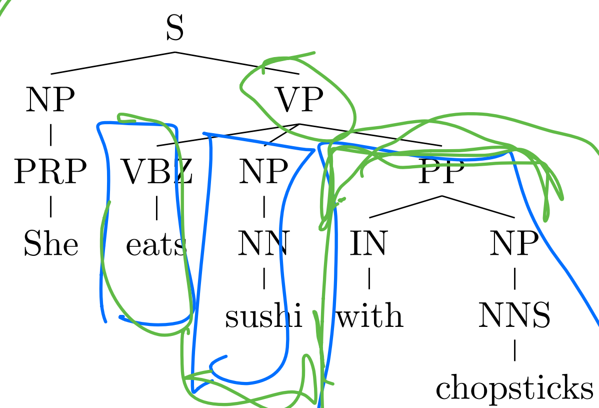
- All useful grammars are *ambiguous*: multiple derivations with same yield
- [Parse tree representations: Nested parens *or* non-terminal spans]



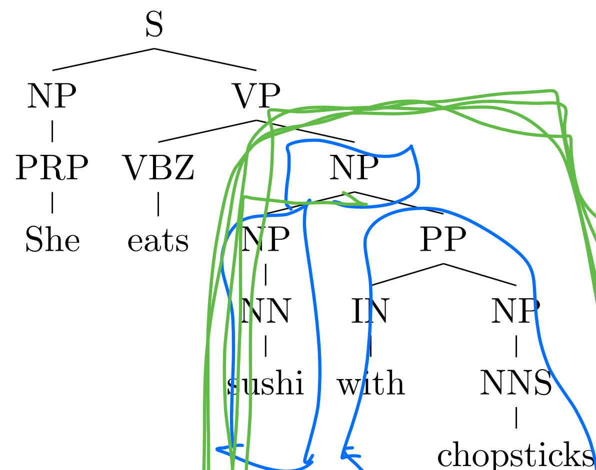
[Examples from Eisenstein (2017)]

# Example

PP Attachment Ambig.




(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
  - etc...
- Machine-learned grammars of English...

- Ambiguities in syntax

**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*



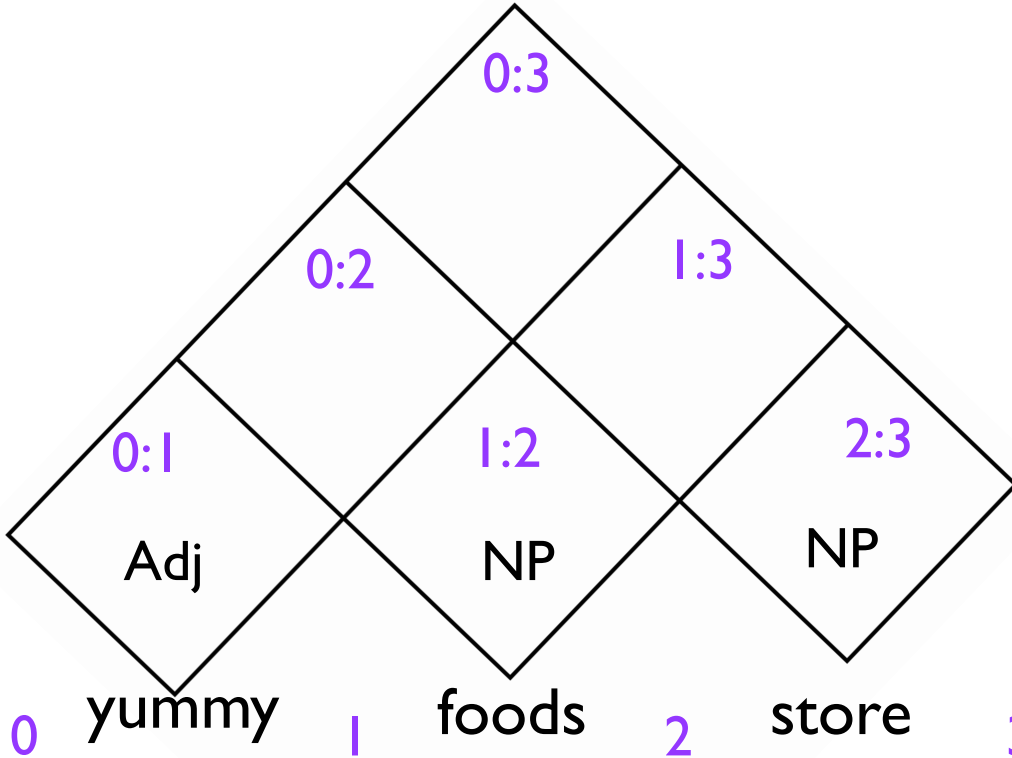
[Examples from Eisenstein (2017)]

# 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): always one or two RHS terms

# CKY




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

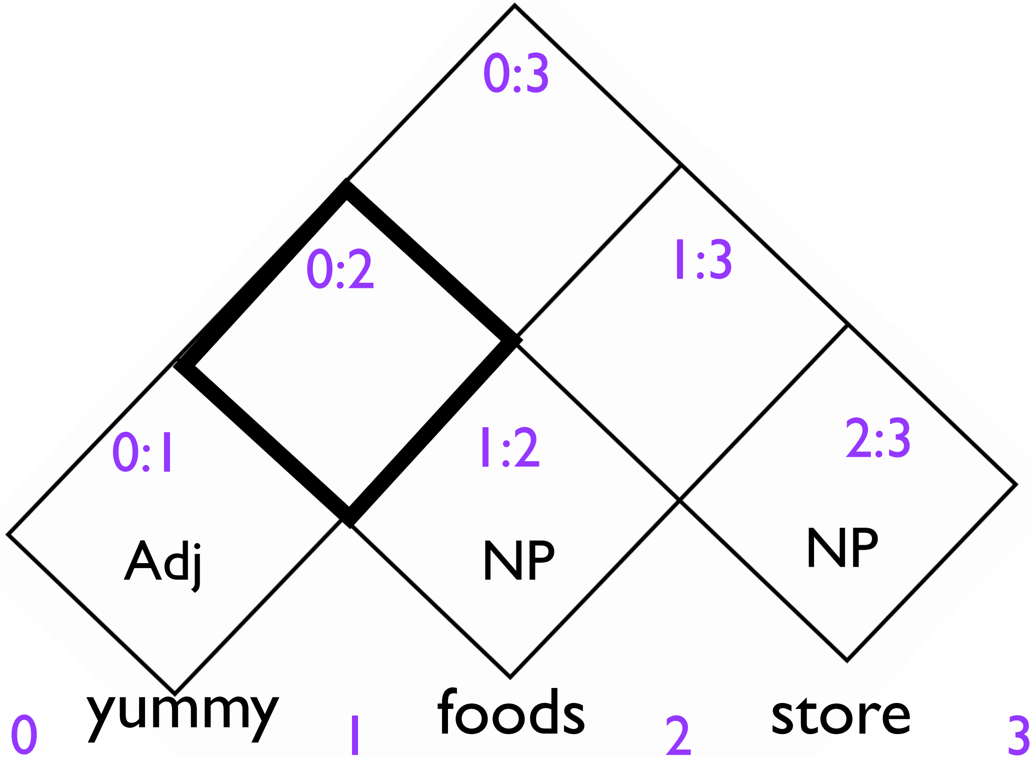
For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                *add* A to cell [i,j] (*Recognizer*)
                ... or ...
                *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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



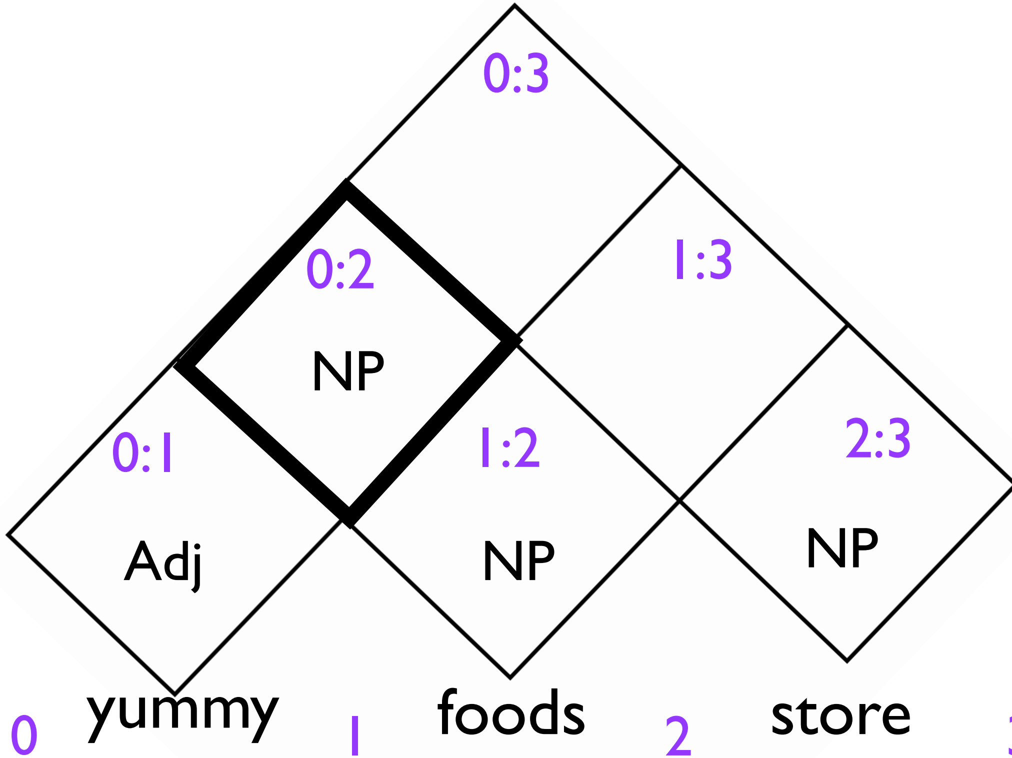

For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                _add_ A to cell [i,j] (_Recognizer_)
                ... or ...
                _add_ (A,B,C, k) to cell [i,j] (_Parser_)

_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



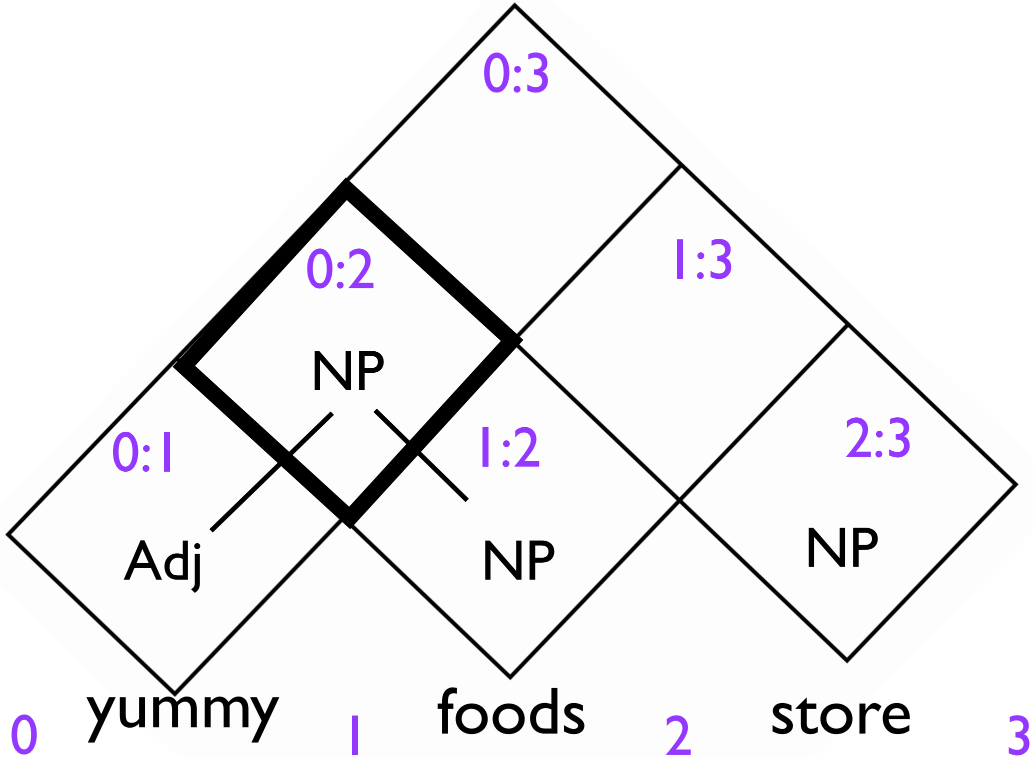

For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                *add* A to cell [i,j] (*Recognizer*)
                ... or ...
                *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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



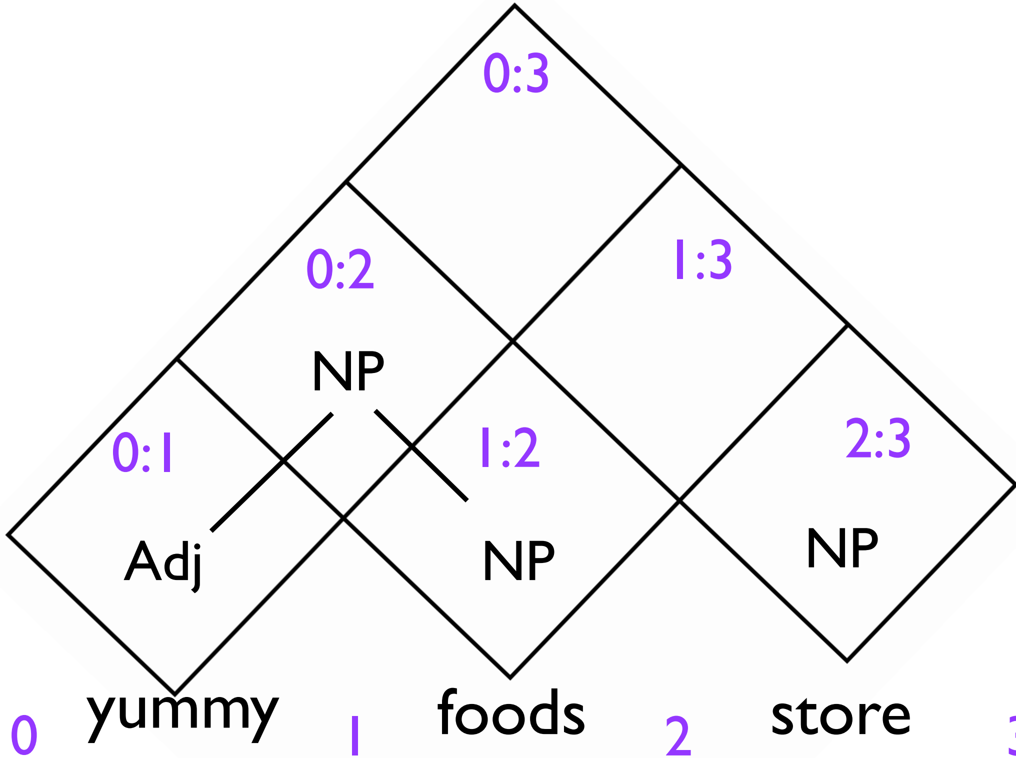

For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                *add* A to cell [i,j] (*Recognizer*)
                ... or ...
                *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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



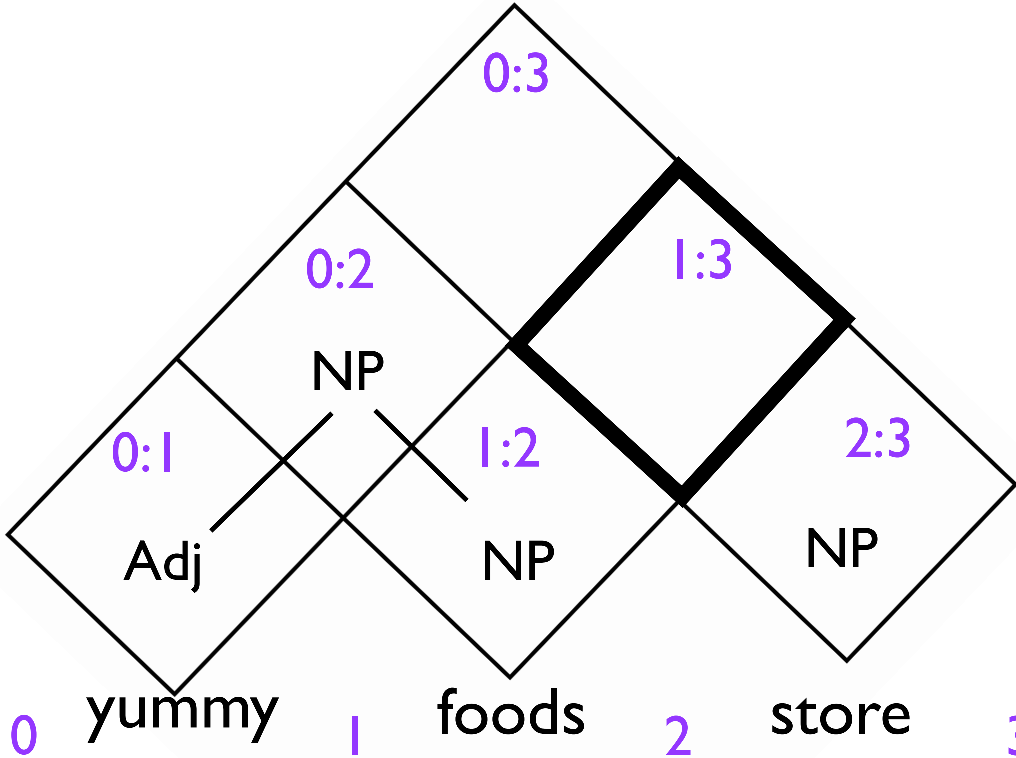

For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j] (*Recognizer*)
        ... or ...
        add (A,B,C, k) to cell [i,j] (*Parser*)

*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



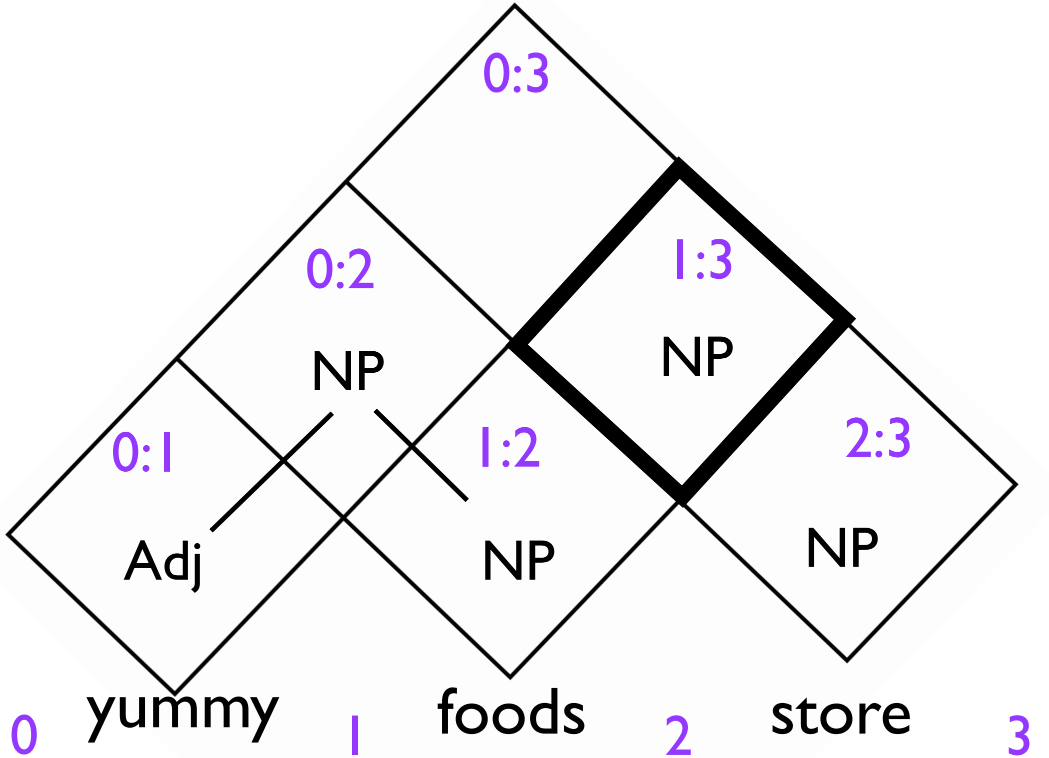

For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                *add* A to cell [i,j] (*Recognizer*)
                ... or ...
                *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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



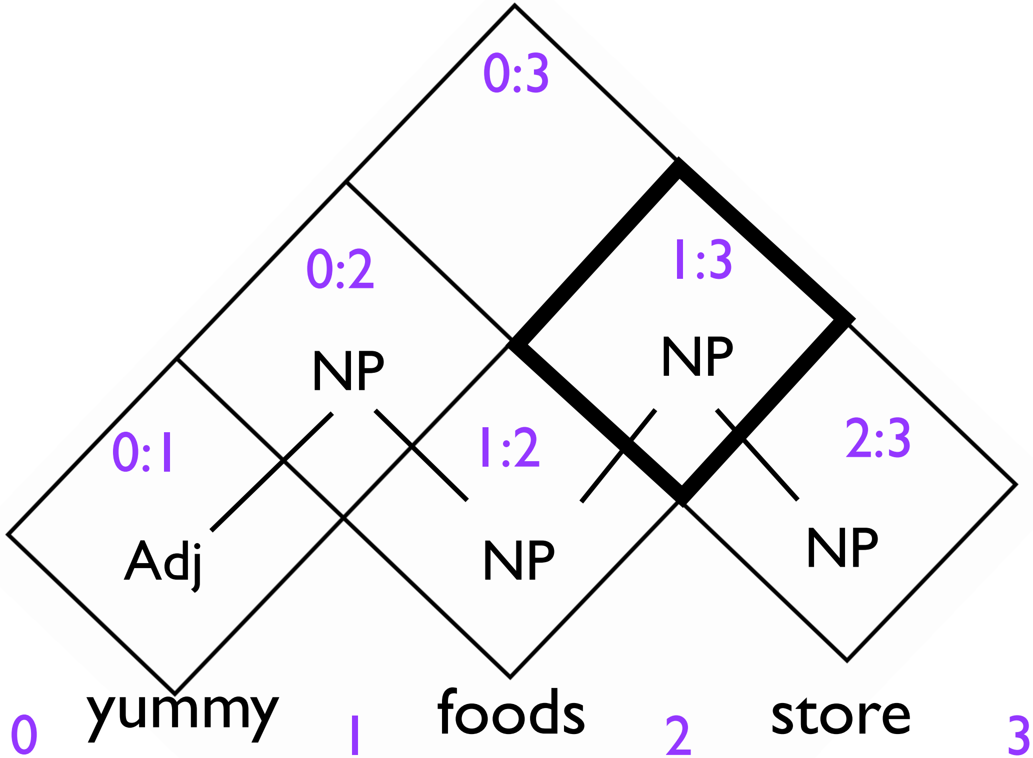

For cell [i,j] (loop through them bottom-up)
For possible splitpoint k=(i+1)..(j-1):
For every B in [i,k] and C in [k,j],
If exists rule A -> B C,
*add* A to cell [i,j] (*Recognizer*)
... or ...
*add* (A,B,C, k) to cell [i,j] (*Parser*)

*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



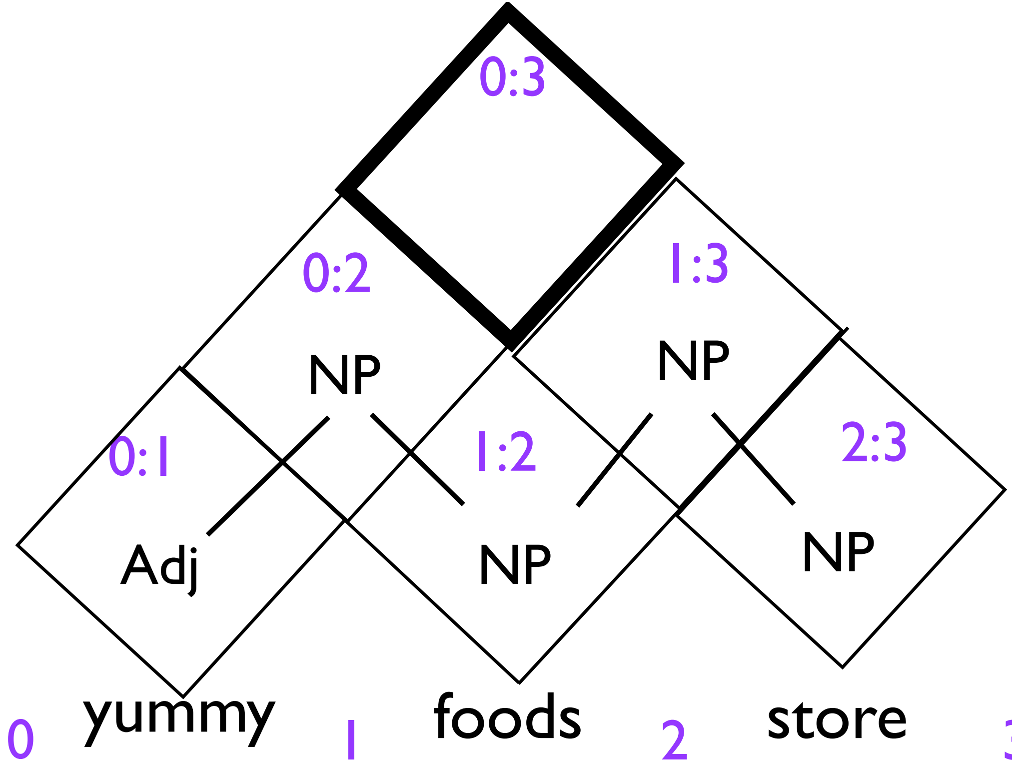

For cell [i,j] (loop through them bottom-up)
For possible splitpoint k=(i+1)..(j-1):
For every B in [i,k] and C in [k,j],
If exists rule A -> B C,
*add* A to cell [i,j] (*Recognizer*)
... or ...
*add* (A,B,C, k) to cell [i,j] (*Parser*)

*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



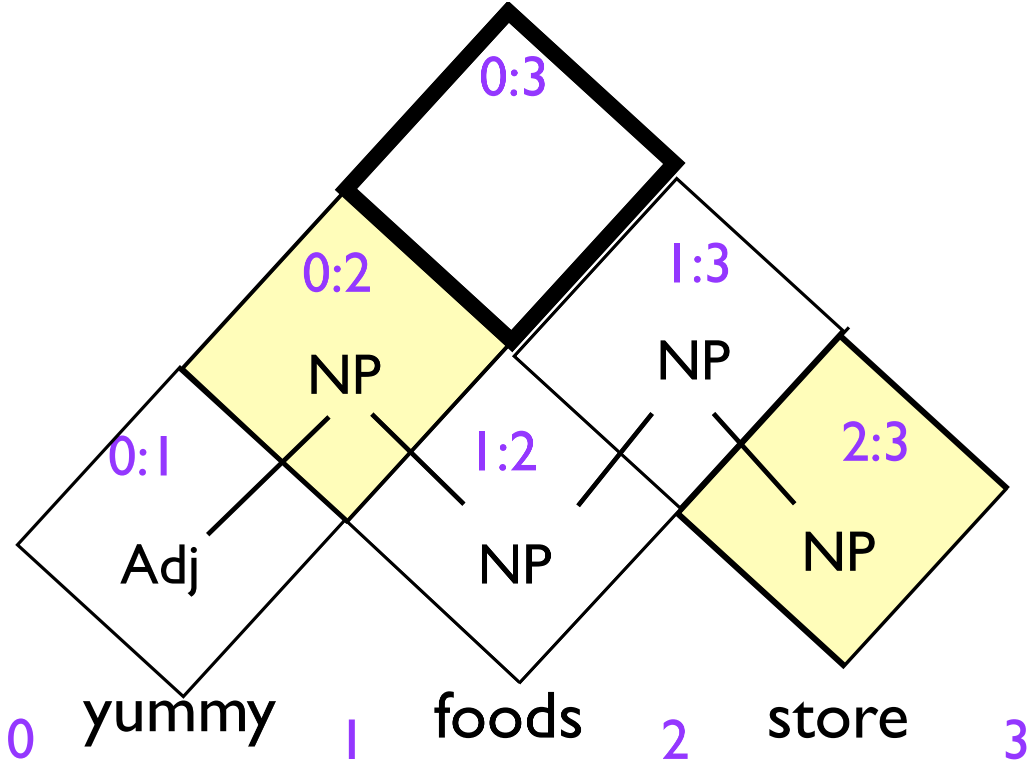

For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        *add* A to cell [i,j] (*Recognizer*)
        ... or ...
        *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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



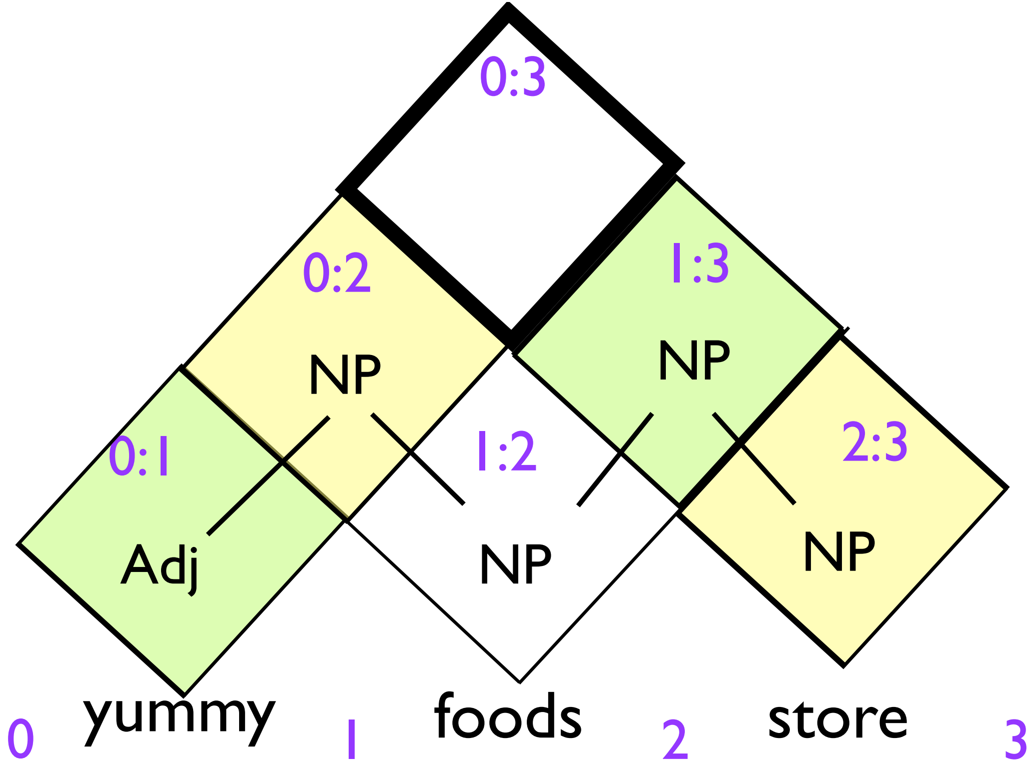

For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        *add* A to cell [i,j] (*Recognizer*)
        ... or ...
        *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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



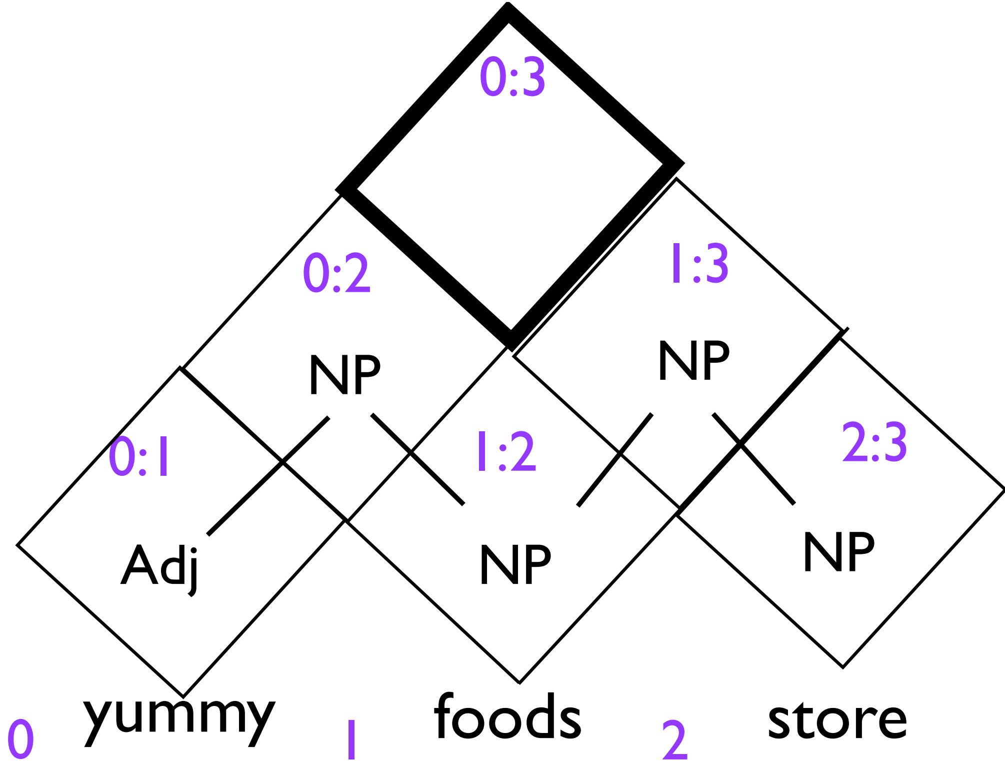

For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                *add* A to cell [i,j] (*Recognizer*)
                ... or ...
                *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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

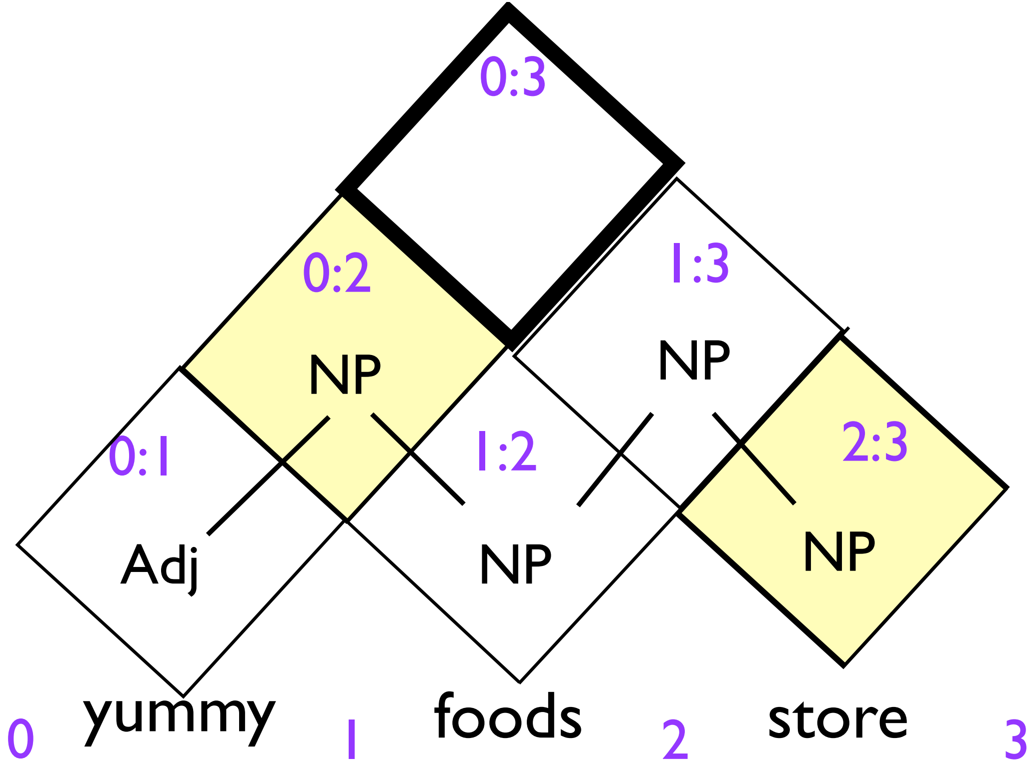



For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                *add* A to cell [i,j] (*Recognizer*)
                ... or ...
                *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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

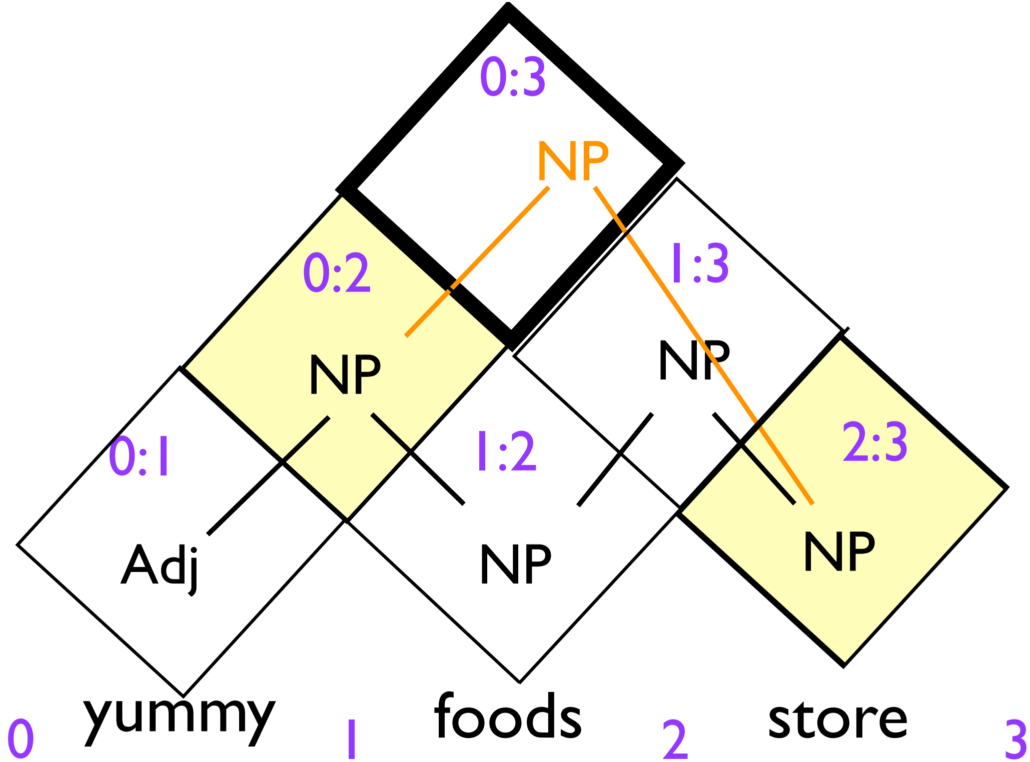
For cell [i,j] (loop through them bottom-up)
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        *add* A to cell [i,j] (*Recognizer*)
        ... or ...
        *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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



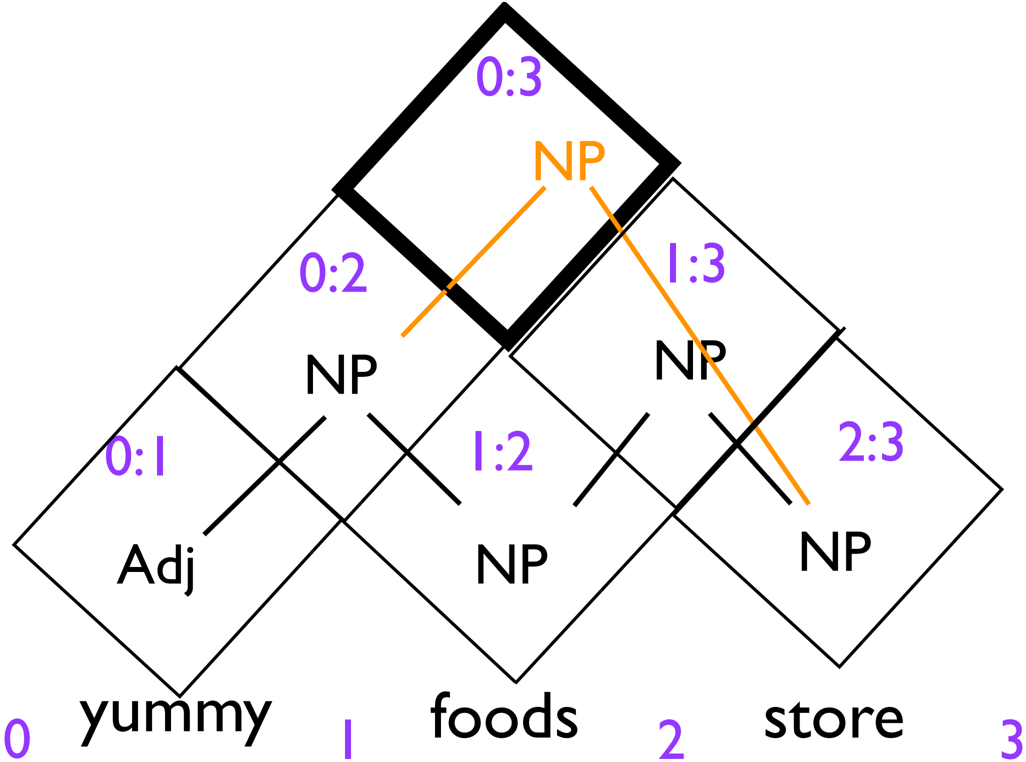

For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                *add* A to cell [i,j] (*Recognizer*)
                ... or ...
                *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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



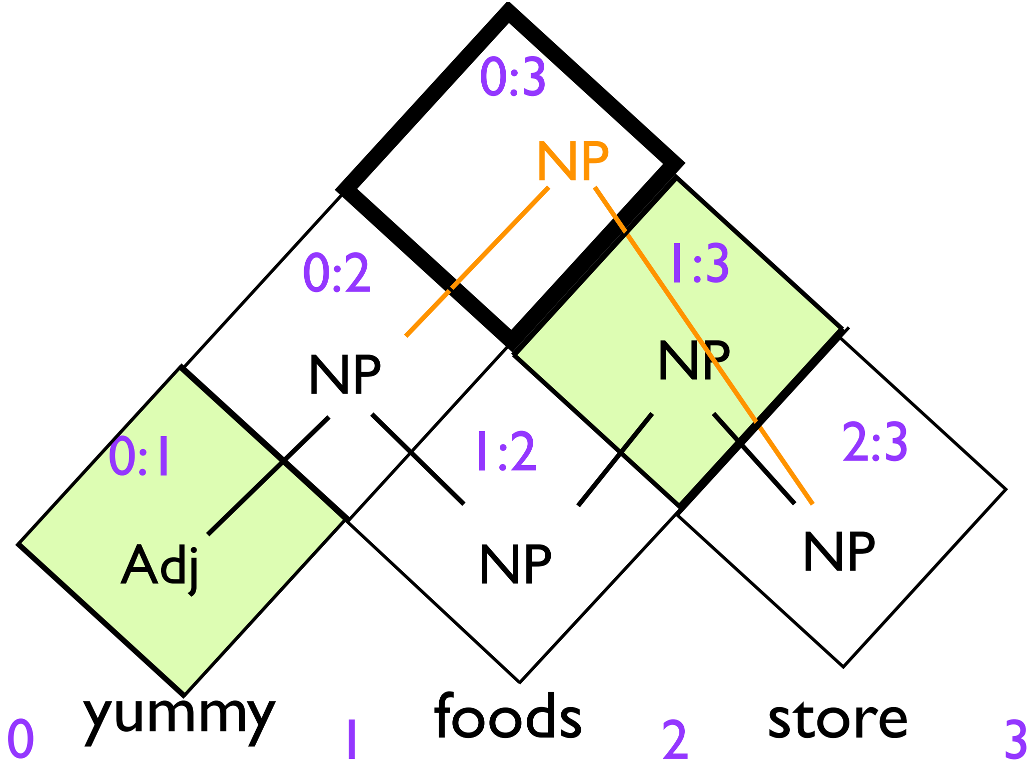

For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                *add* A to cell [i,j] (*Recognizer*)
                ... or ...
                *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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

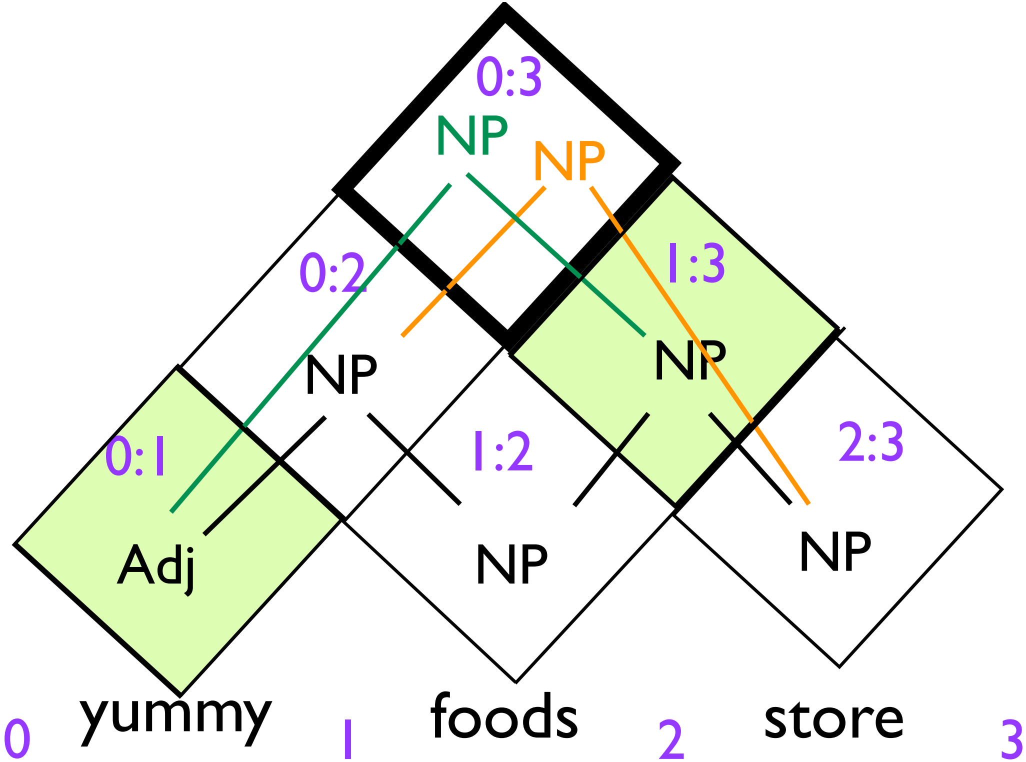



For cell [i,j] (loop through them bottom-up)
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                *add* A to cell [i,j] (*Recognizer*)
                ... or ...
                *add* (A,B,C, k) to cell [i,j] (*Parser*)

*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*)

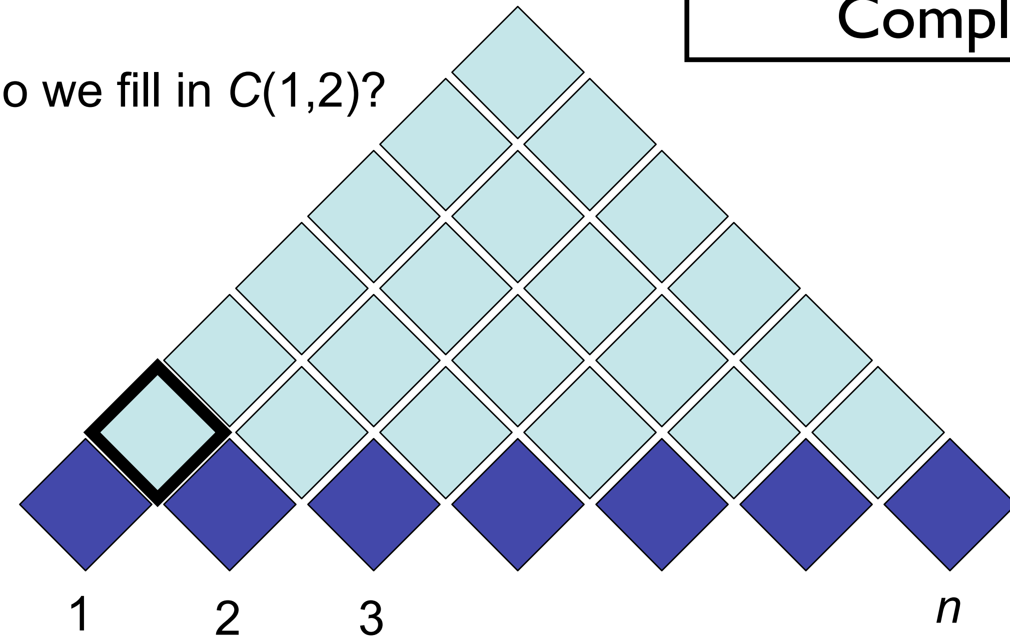
*Recognizer*: per span, record list of possible nonterminals

*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)?



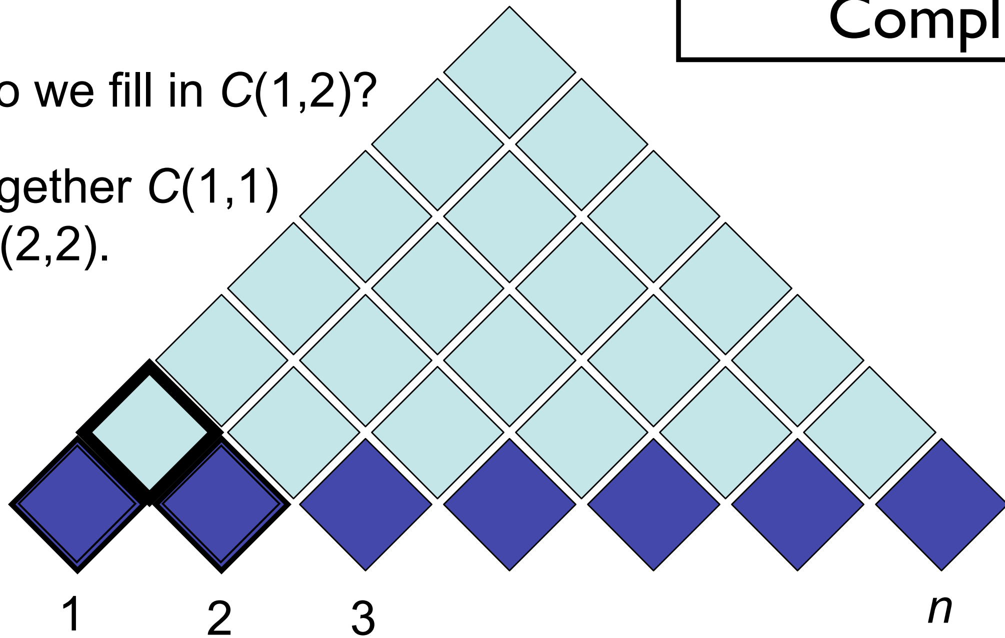

*[Example from Noah Smith]*

```
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).

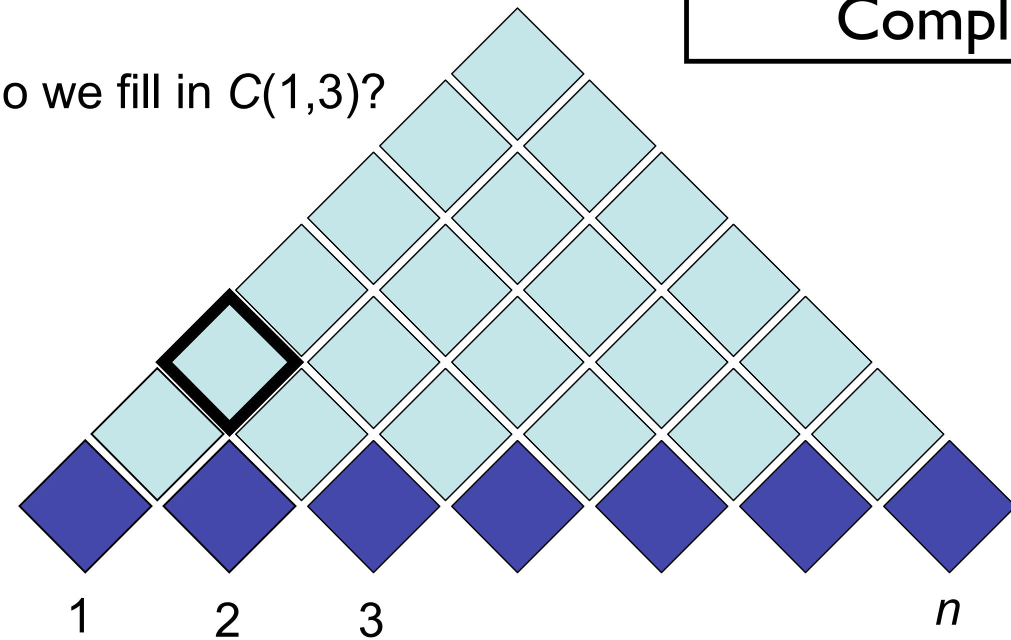



*[Example from Noah Smith]*

```
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)?



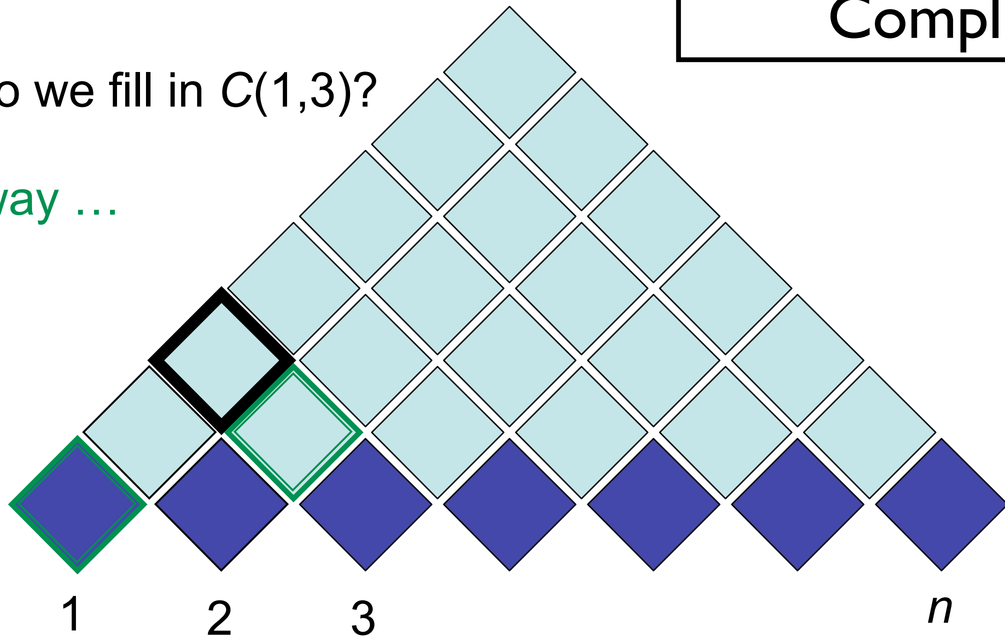

*[Example from Noah Smith]*

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 …



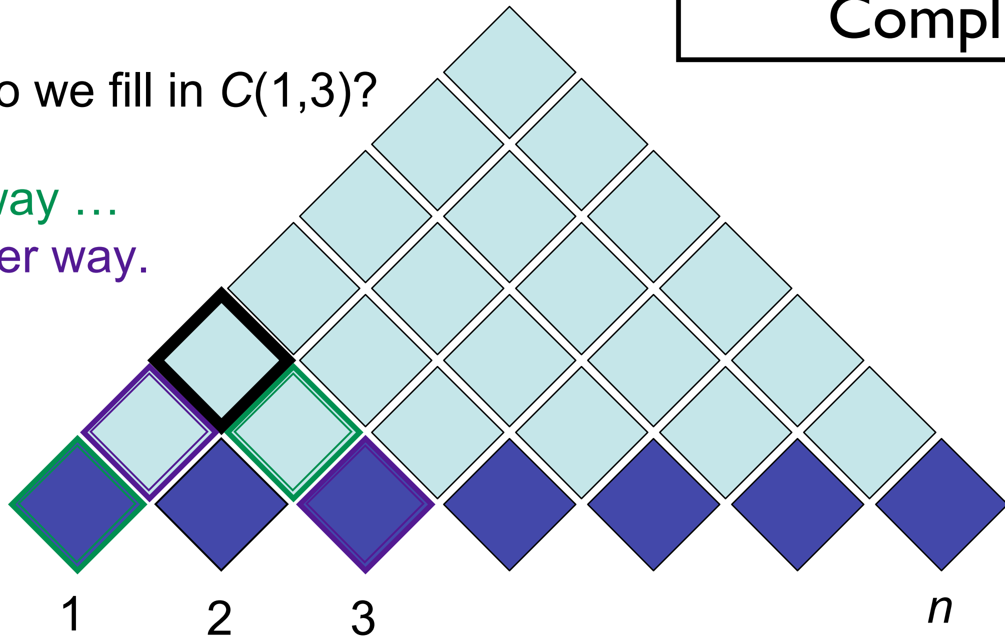

*[Example from Noah Smith]*

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.

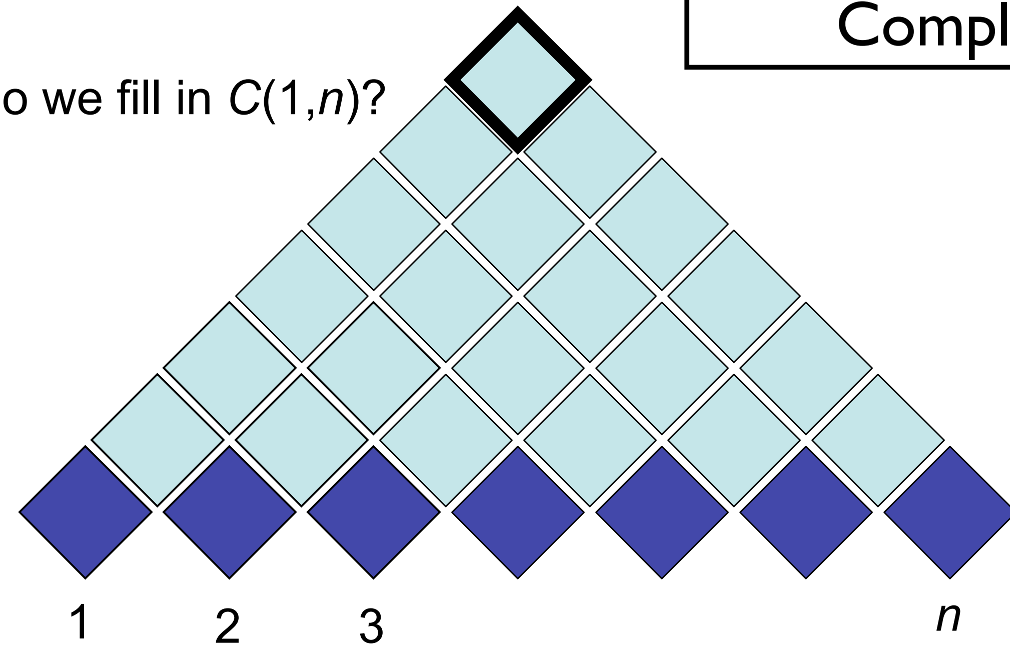



*[Example from Noah Smith]*

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 $C(1,n)$?



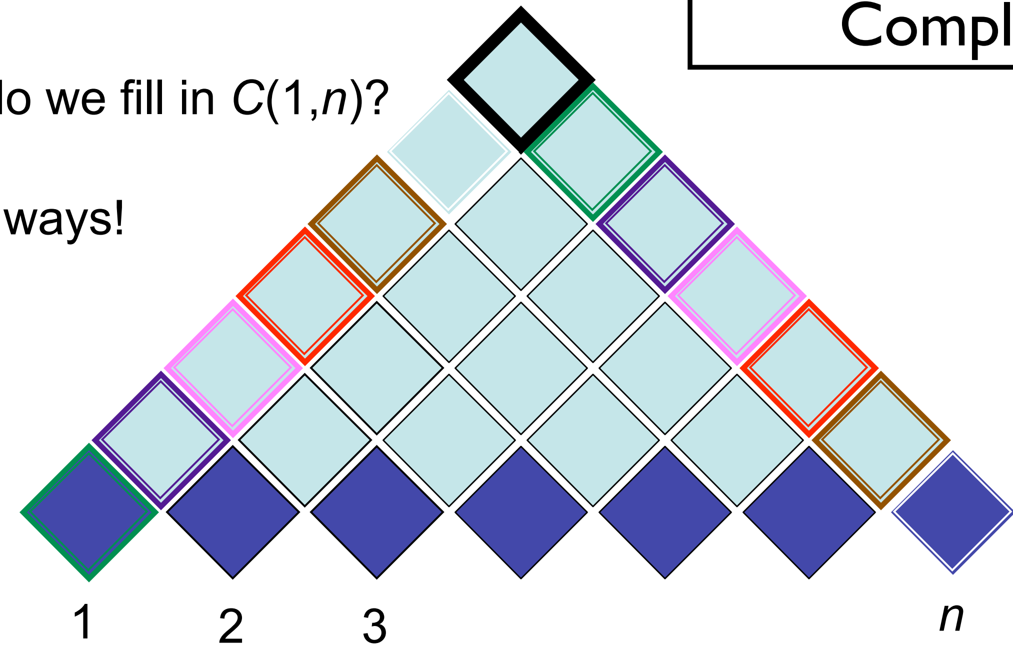

*[Example from Noah Smith]*

```
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!



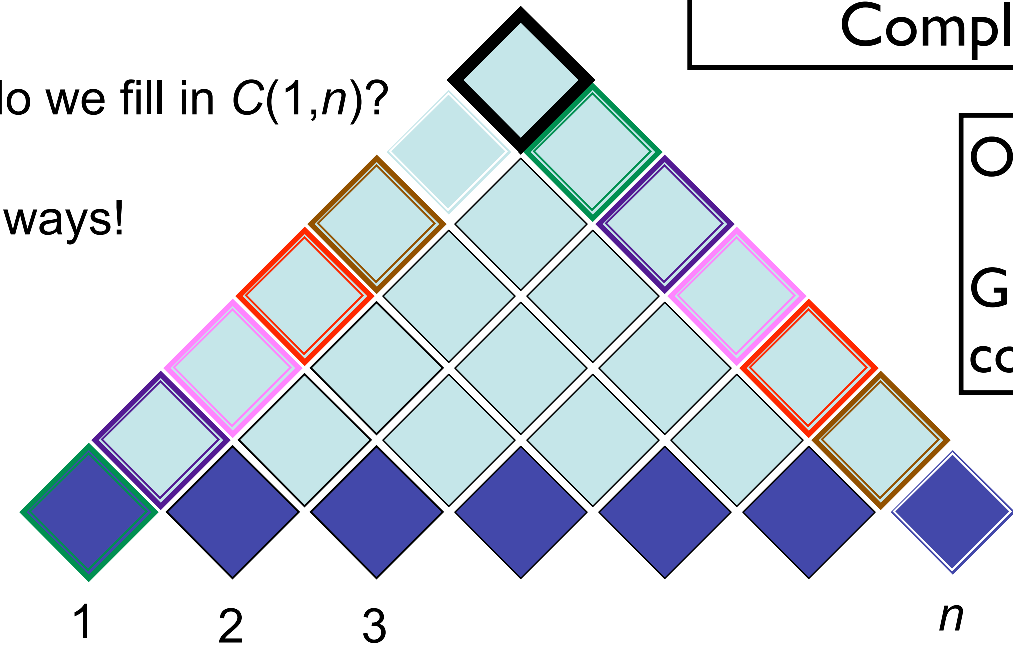

*[Example from Noah Smith]*

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



*[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) ))))))))))))))))
```

# (P)CFG model, (P)CKY algorithm

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

- Probabilistic/Weighted CKY: given PCFG and sentence w
  - Likelihood of sentence P(w)
  - Most probable parse ("Viterbi parse")
    $\mathrm{argmax}_y\, P(y \mid w) = \mathrm{argmax}_y\, P(y, w)$
  - Non-terminal span marginals (Inside-outside algorithm)

- Discriminative Tree-CRF parsing:
  $\mathrm{argmax}_y\, P(y \mid w)$

- Parsing model accuracy: lots of ambiguity!!
  - PCFGs lack lexical information to resolve ambiguities (sneak in world knowledge?)
  - Need to add word embeddings or other lexical information to enrich phrase representations

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

- Evaluate: precision and recall of labeled spans
- Treebank data

# 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

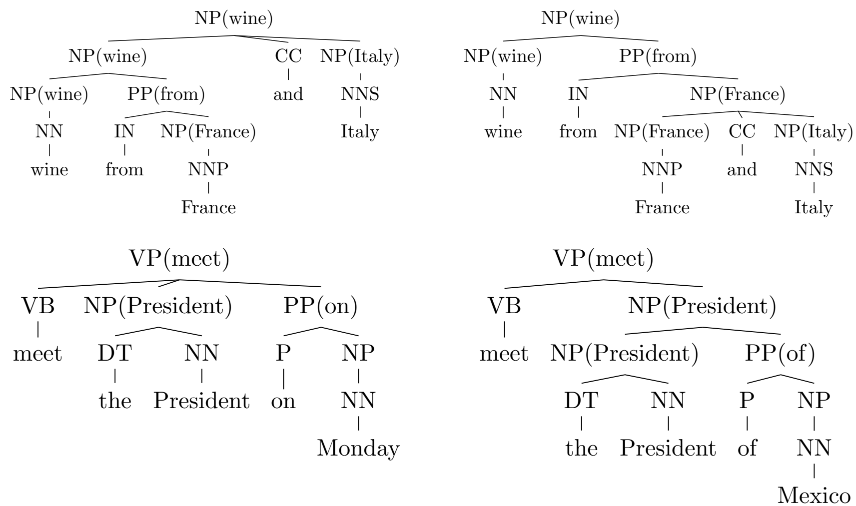



Figure 11.9: Lexicalization can address ambiguity on coordination scope (upper) and PP attachment (lower)



[From Eisenstein (2017)]

# Reranking

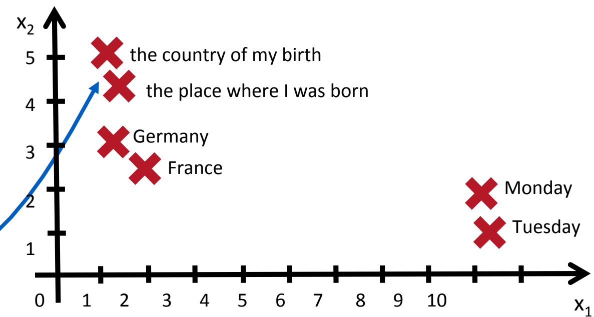
- (CRF/Neural/etc.) CFGs are fast, but only use local info
- Whole-structure scoring (features, tree RNNs, etc.) is slow, but can use global info
- Solution: **Reranking**
  - CKY/Viterbi to infer top-K parses from fast CFG model
  - Score each one with NN/features
    for K-way multiclass problem
    - or use a ranking loss, etc.

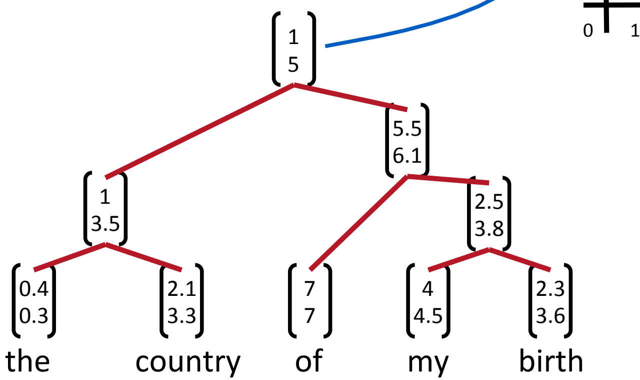
# Reranking: TreeRNN

[Socher et al. (2013)]

**Use principle of compositionality**

The meaning (vector) of a sentence is determined by
(1) the meanings of its words and
(2) the rules that combine them.




$$\boldsymbol{u}_{i,j} = f\left(\Theta_{X\to Y\ Z}\begin{bmatrix}\boldsymbol{u}_{i,k}\\ \boldsymbol{u}_{k,j}\end{bmatrix}\right)$$

(Can also be used for classification or other tasks, not just parsing itself)

- stopped here 3/9

# Model performance

| | |
|---|---|
| Vanilla PCFG | 72% |
| Parent-annotations (Johnson, 1998) | 80% |
| Lexicalized (Charniak, 1997) | 86% |
| Lexicalized (Collins, 2003) | 87% |
| Lexicalized, reranking, self-training (McClosky et al., 2006) | 92.1% |
| State splitting (Petrov and Klein, 2007) | 90.1% |
| CRF Parsing (Finkel et al., 2008) | 89% |
| TAG Perceptron Parsing (Carreras et al., 2008) | 91.1% |
| Compositional Vector Grammars (Socher et al., 2013a) | 90.4% |
| Neural CRF (Durrett and Klein, 2015) | 91.1% |

Table 11.7: Penn Treebank parsing scoreboard, circa 2015 (Durrett and Klein, 2015)



[From Eisenstein (2017)]

# Treebanks

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

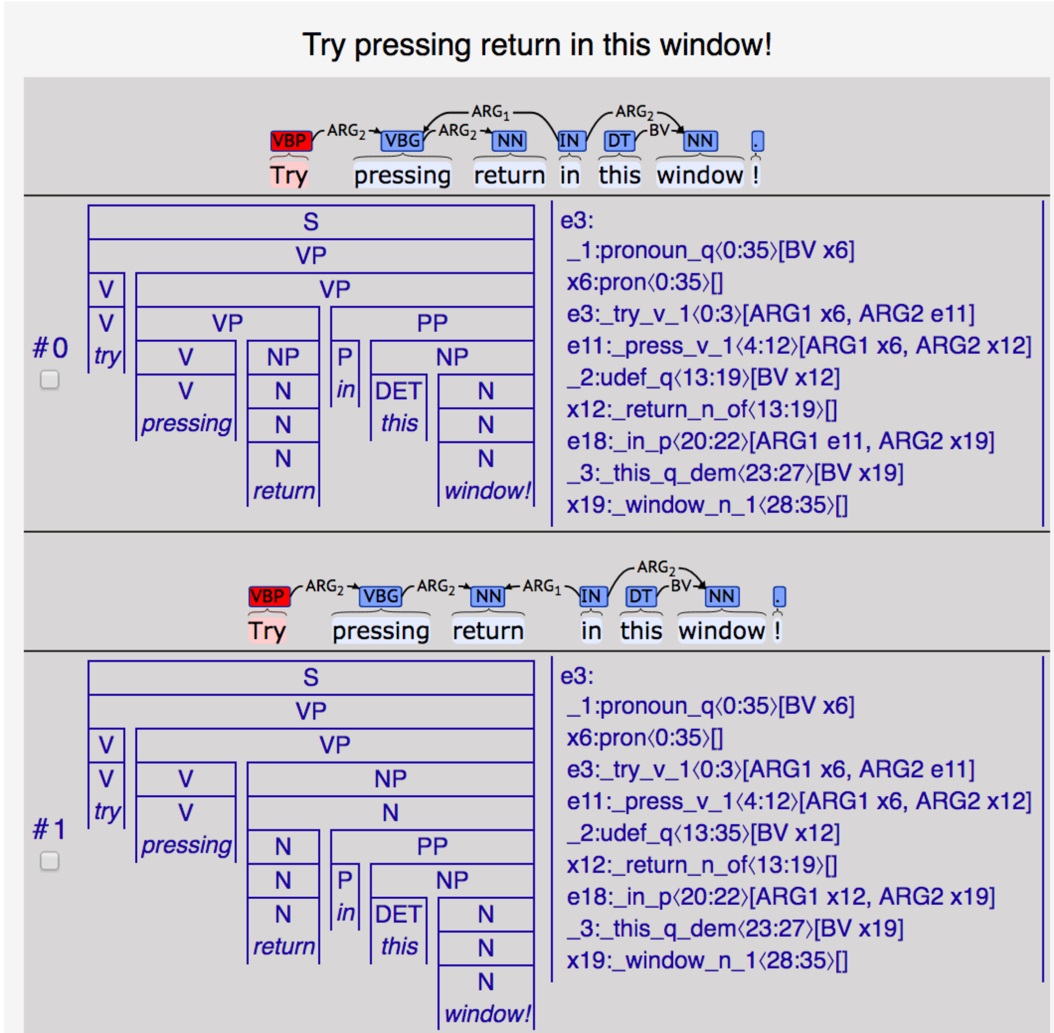
- Prague Treebank (syn+sem)
- many others...

- Know what you're getting!

# More sophisticated formalisms

- Beyond CFGs ("Mildly context-sensitive")
  - e.g. Combinatory Categorial Grammar, Tree Adjoining Grammar, unification grammars ...
  - Extend CFGs to incorporate features to enforce grammatical constraints, or lay the groundwork for meaning interpretation
- English Resource Grammar: a hand-engineered grammar+parser
  - http://erg.delph-in.net/logon
  - Head-driven Phrase Structure Grammar (HPSG)
  - Parse forest -- from a CKY-like chart
  - Dependencies integrated with constituents (Next time: dep parsing as its own task)
- ML-based parsers are SOTA: coverage and resolving ambiguities

# ERG


Try pressing return in this window!
#0
e3:
_1:pronoun_q⟨0:35⟩[BV x6]
x6:pron⟨0:35⟩[]
e3:_try_v_1⟨0:3⟩[ARG1 x6, ARG2 e11]
e11:_press_v_1⟨4:12⟩[ARG1 x6, ARG2 x12]
_2:udef_q⟨13:19⟩[BV x12]
x12:_return_n_of⟨13:19⟩[]
e18:_in_p⟨20:22⟩[ARG1 e11, ARG2 x19]
_3:_this_q_dem⟨23:27⟩[BV x19]
x19:_window_n_1⟨28:35⟩[]
#1
e3:
_1:pronoun_q⟨0:35⟩[BV x6]
x6:pron⟨0:35⟩[]
e3:_try_v_1⟨0:3⟩[ARG1 x6, ARG2 e11]
e11:_press_v_1⟨4:12⟩[ARG1 x6, ARG2 x12]
_2:udef_q⟨13:35⟩[BV x12]
x12:_return_n_of⟨13:19⟩[]
e18:_in_p⟨20:22⟩[ARG1 x12, ARG2 x19]
_3:_this_q_dem⟨23:27⟩[BV x19]
x19:_window_n_1⟨28:35⟩[]