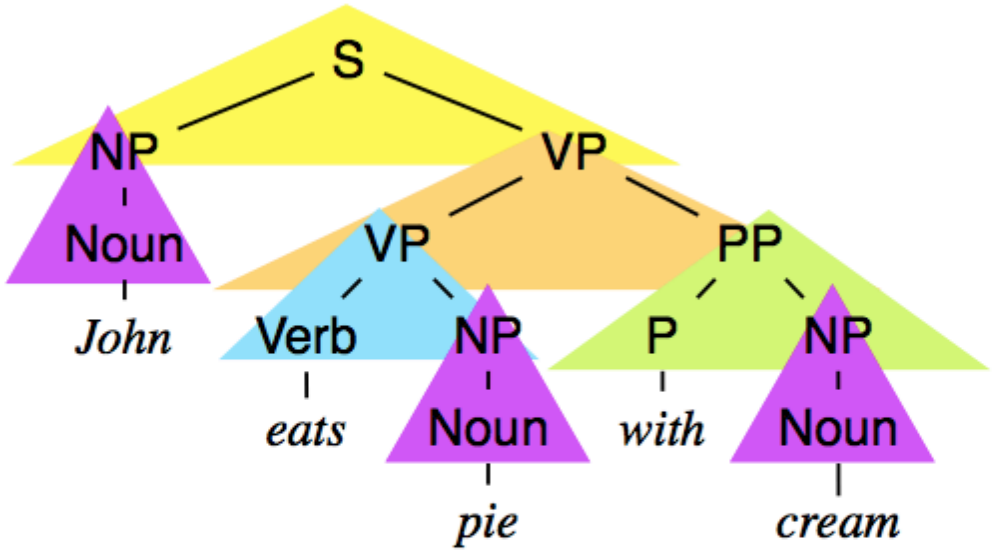
Probabilistic CFG

- normalization
  - $\text{sum}_\beta \; p(A \rightarrow \beta) = 1$
- where are these probs from?
  - "gold-stand" trees -- Treebank
  - $p(A \rightarrow \beta) = \#(A \rightarrow \beta) \; / \; \#(A)$
- what's the most likely tree? *easy*
- what's the most likely string? *hard*
- given string w, what's the most likely tree for w?
  - this is called "parsing" (like decoding)

S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

Probability of a tree

The probability of a tree τ is the product of the probabilities of all its rules:



S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

$$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^3$$

$$= \mathbf{0.00384}$$

Most Likely Tree - Knuth 77

S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

- highest-probability tree *out of* a PCFG
- idea 1: dynamic programming (DP)
  - optimize for each nonterminal X
  - problem: cyclic updates
    - if S -> NP VP and VP -> VB S
    - can't use Viterbi (which relies on topological ordering)
- idea 2: best-first DP: Dijkstra! (b/c prob <=1, cf. non-negative)
  - every time choose the best nonterminal in the queue to expand
  - but not always possible to combine w/ others to update LHS
  - update LHS only when all RHS nonterminals are ready (popped)

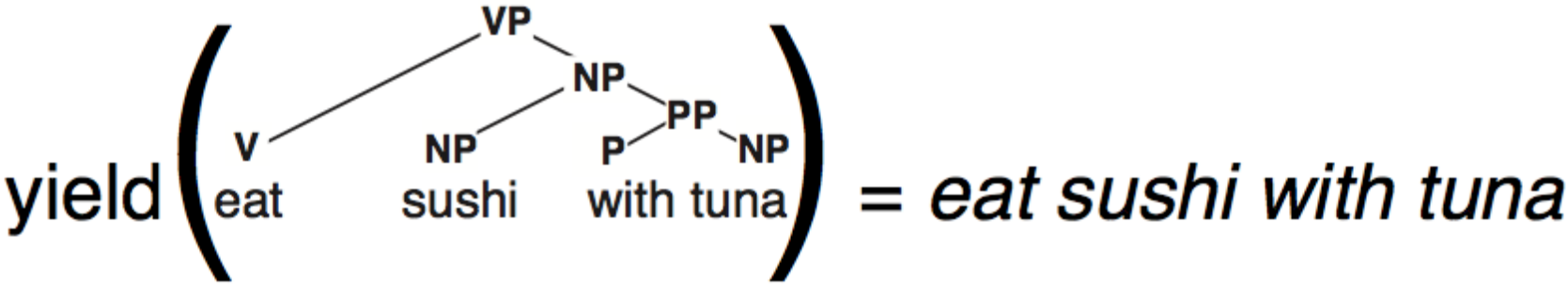
Knuth 77 Example

S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

- initial queue = (NP: 0.4, VP: 0.4)
- say pop VP: 0.4
  - which rules can be used for updates?
    - S -> NP VP 0.8; VP -> VP PP 0.2
    - can we use these rules now?
      - No, b/c NP and PP are not ready
- next, pop NP: 0.4
  - which rules can be used for updates?
    - S -> NP VP 0.8; PP ->P NP 1.0
    - update S to be 0.8x0.4x0.4=0.128 and PP to be 1.0x0.4=0.4
- next, pop PP: 0.4; but can't update anything (NP/VP already popped)
- next, pop S: 0.128, and finishes here since S is the start symbol

Most likely tree given string

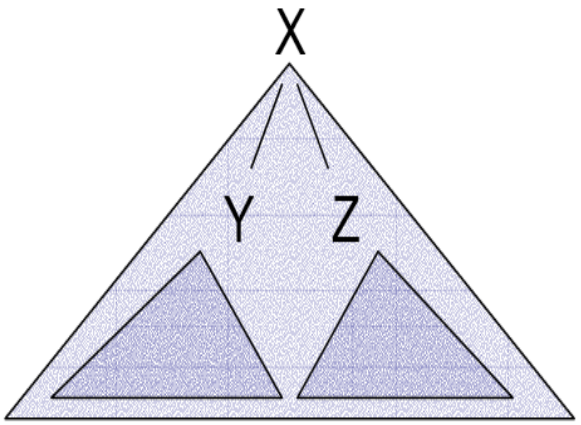
- parsing is to search for the best tree t* that:
  - $t^* = \text{argmax}_t \; p\,(t \mid w) = \text{argmax}_t \; p(t)\; p\,(w \mid t)$
  - $= \text{argmax}_{\{t:\, \text{yield}(t)=w\}} \; p(t)$
  - analogous to HMM decoding
- is it related to "intersection" or "composition" in FSTs?

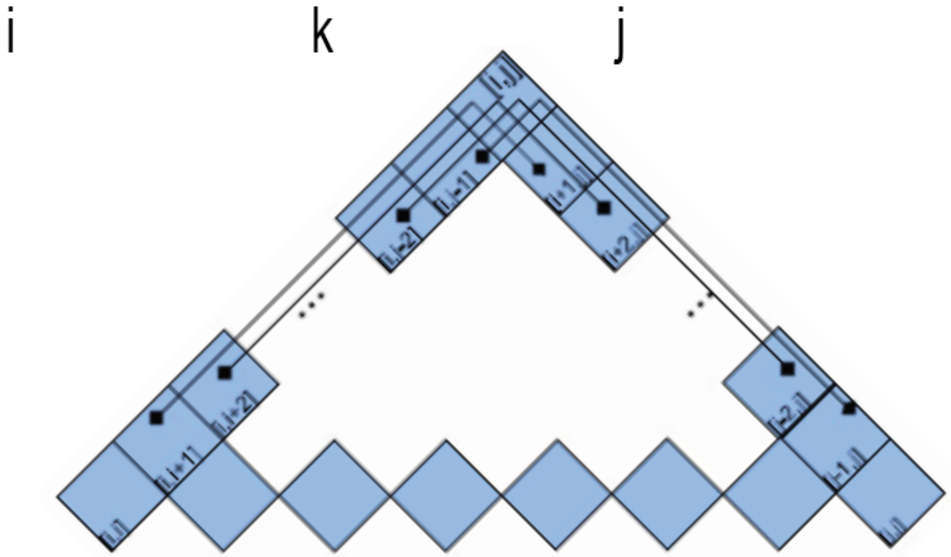


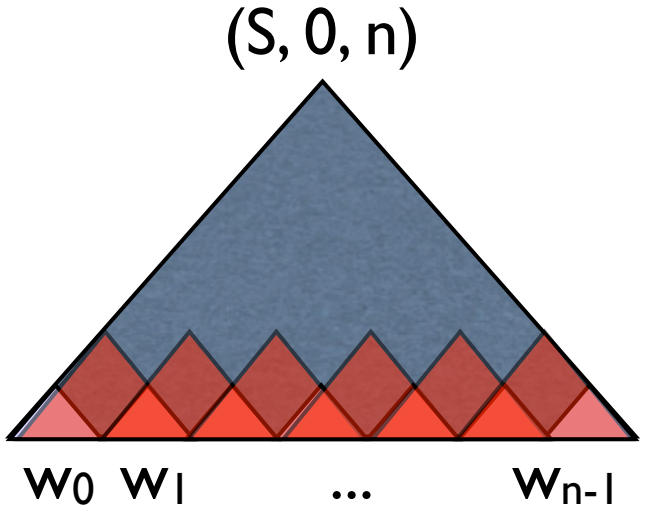
CKY Algorithm

- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max score[X][i][j],
                     score(X->YZ) *
                     score[Y][i][k] *
                     score[Z][k][j]
```



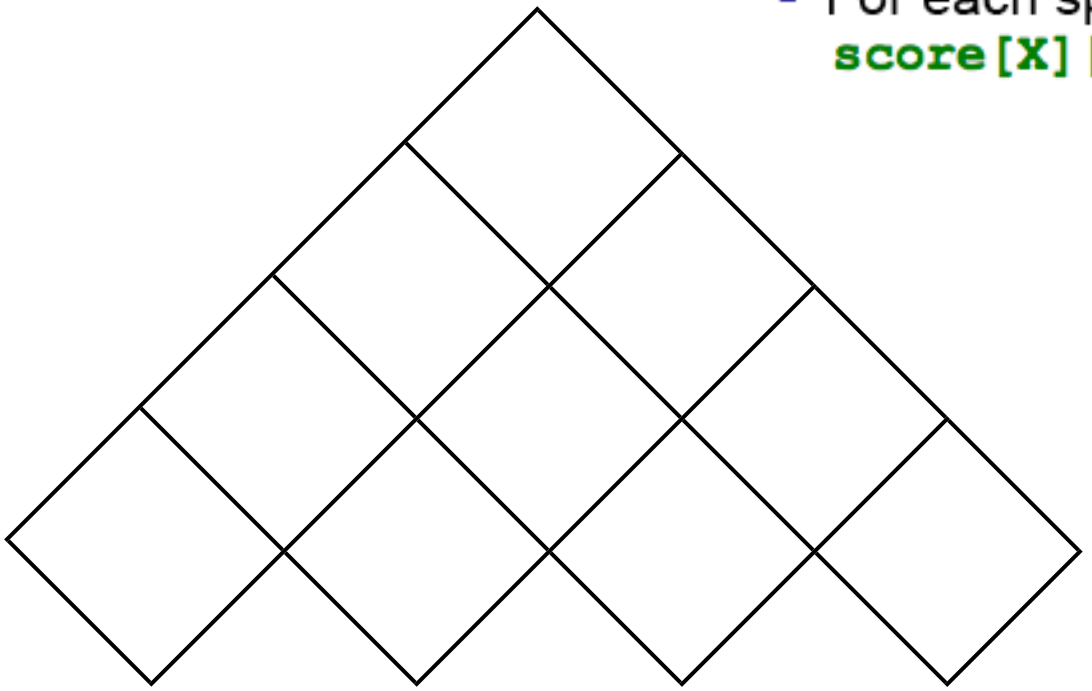




CKY Algorithm

- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max score[X][i][j],
                     score(X->YZ) *
                     score[Y][i][k] *
                     score[Z][k][j]
```



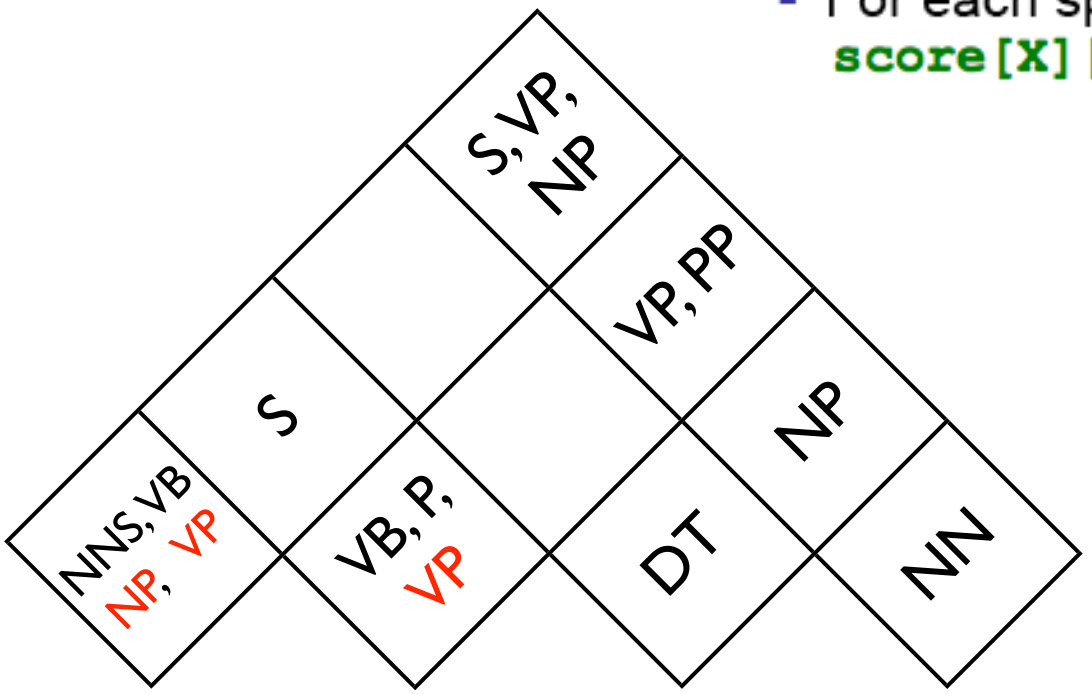
S → NP VP
NP → DT NN
NP → NNS
NP → NP PP
VP → VB NP
VP → VP PP
VP → VB
PP → P NP

VB → flies
NNS → flies
VB → like
P → like
DT → a
NN → flower

CKY Algorithm

- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max score[X][i][j],
                     score(X->YZ) *
                     score[Y][i][k] *
                     score[Z][k][j]
```



S → NP VP
NP → DT NN
NP → NNS
NP → NP PP
VP → VB NP
VP → VP PP
VP → VB
PP → P NP

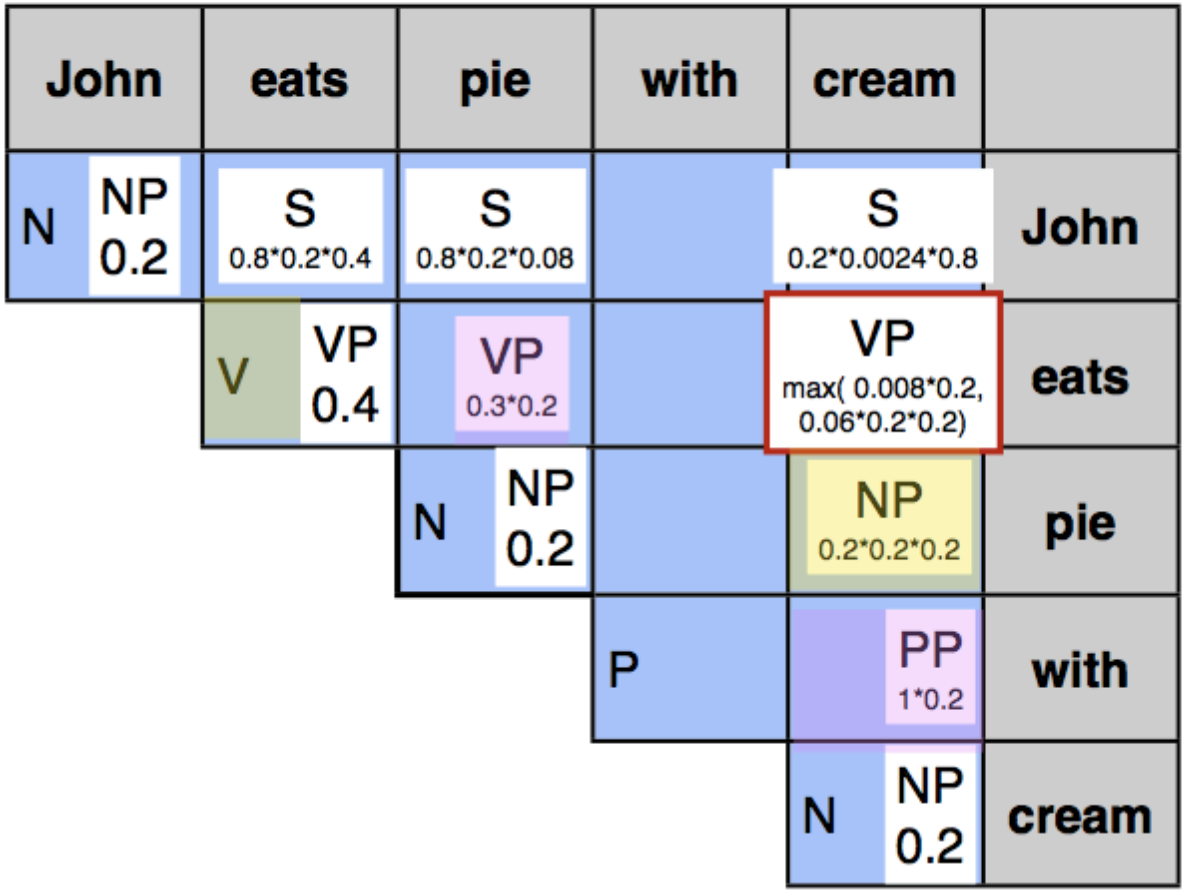
VB → flies
NNS → flies
VB → like
P → like
DT → a
NN → flower

note: *unary rules*

CKY Example

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`



S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

Chomsky Normal Form

- wait! how can you assume a CFG is binary-branching?
- well, we can always convert a CFG into Chomsky-Normal Form (CNF)
  - A → B C
  - A → a
- how to deal with epsilon-removal?
- how to do it with PCFG?