Language Technology

CUNY Graduate Center Spring 2013

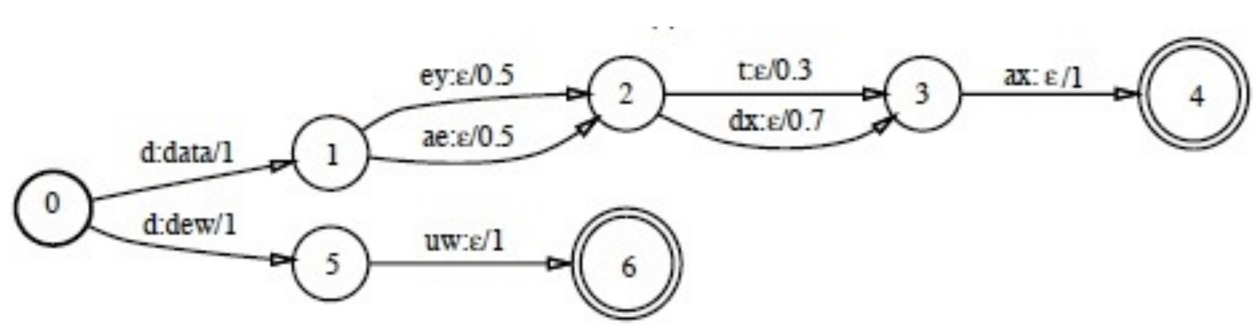
Unit 2: Tree Models

Lectures 9-11: Context-Free Grammars and Parsing

required

hard

optional



Professor Liang Huang

liang.huang.sh@gmail.com

Big Picture

- only 2 ideas in this course: Noisy-Channel and Viterbi (DP)
- we have already covered...
  - sequence models (WFSAs, WFSTs, HMMs)
  - decoding (Viterbi Algorithm)
  - supervised training (counting, smoothing)
- in this unit we'll look beyond sequences, and cover...
  - tree models (prob context-free grammars and extensions)
  - decoding ("parsing", CKY Algorithm)
  - supervised training (lexicalization, history-annotation, ...)

Course Project

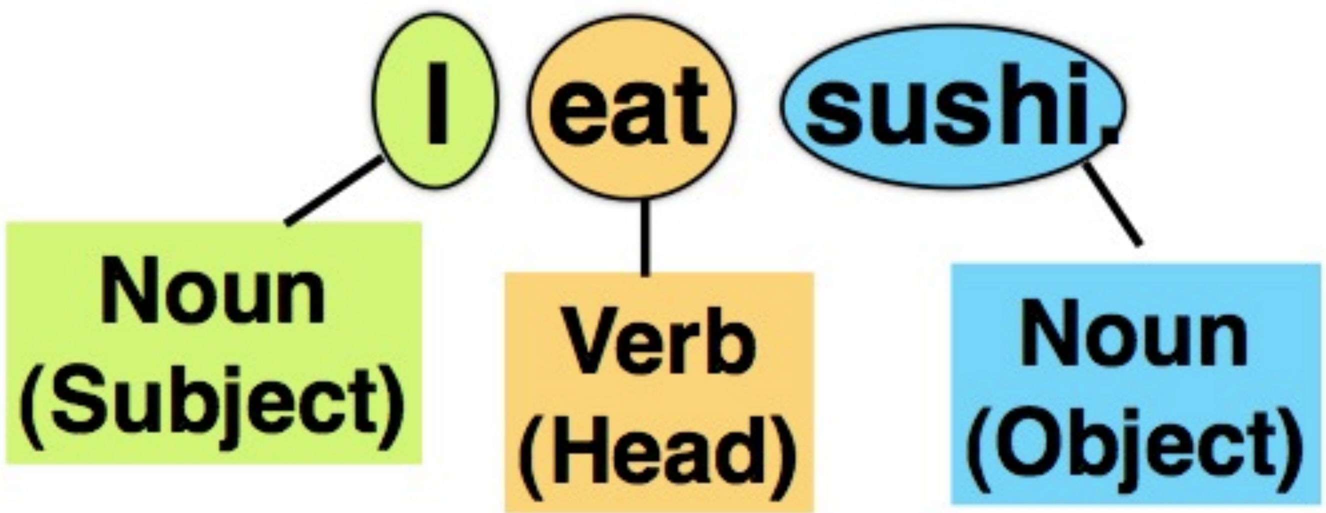
- Proposal
  - due next Tuesday 4/23 -- should also propose a simple baseline
  - please talk to us this Friday re: your topic
- Topic (see list of samples from previous years)
  - must involve *statistical* processing of linguistic *structures*
  - NO boring topics like text classification with bags of words
  - example 1: playing the Shannon game with higher-order LM
  - example 2: converting declarative sentences into questions
- Amount of Work: ~2 HWs for each student

Limitations of Sequence Models

- can you write an FSA/FST for the following?
  - $\{ (a^n, b^n) \}$ $\{ (a^{2n}, b^n) \}$
  - $\{ a^n\ b^n \}$
  - $\{ w\ w^R \}$
  - $\{ (w, w^R) \}$
- does it matter to human languages?
  - [The woman saw the boy [that heard the man [that left]]].
  - [The claim [that the house [he bought] is valuable] is wrong].
  - but humans can't really process infinite recursions... stack overflow!

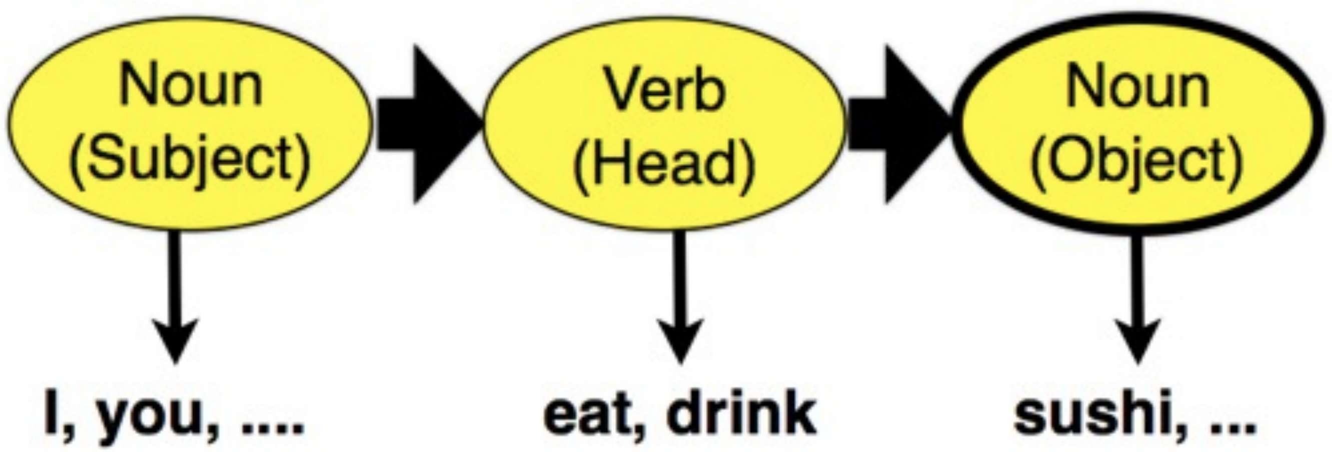
Let's try to write a grammar...

(courtesy of Julia Hockenmaier)



- let's take a closer look...
- we'll try our best to represent English in a FSA...
- basic sentence structure: N, V, N

Subject-Verb-Object



- compose it with a lexicon, and we get an HMM
- so far so good

(Recursive) Adjectives

(courtesy of Julia Hockenmaier)

the ball
the big ball
the big, red ball
the big, red, heavy ball ....

- then add Adjectives, which modify Nouns
- the number of modifiers/adjuncts can be unlimited.
- how about no determiner before noun? “play tennis”

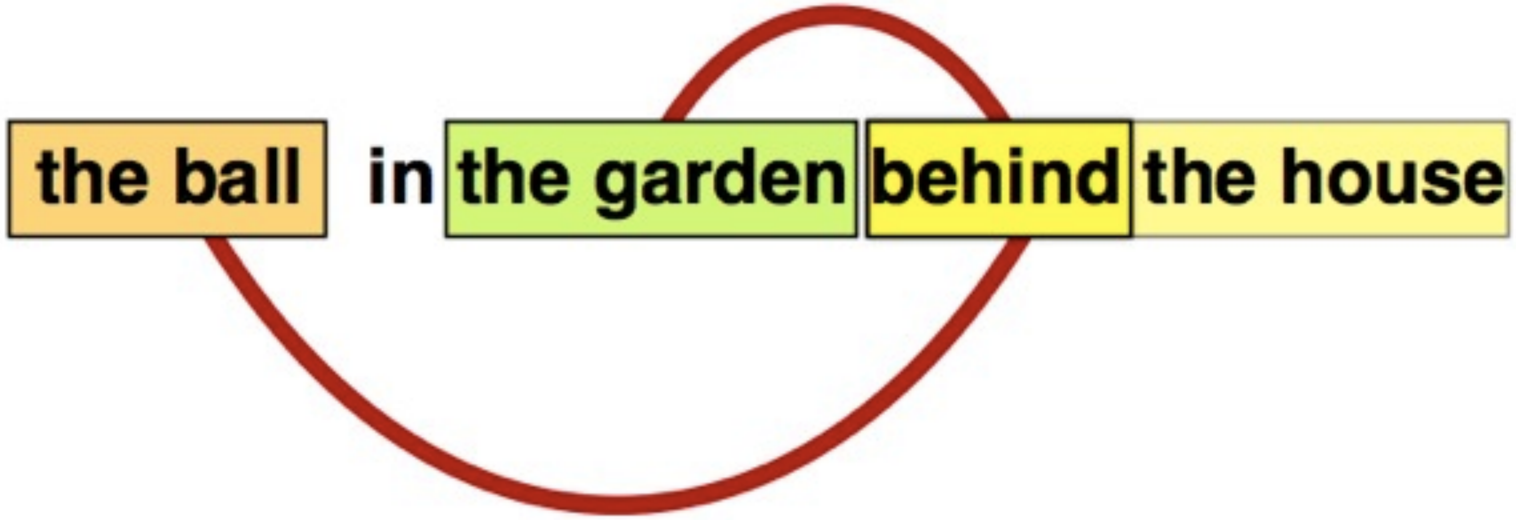
Recursive PPs

(courtesy of Julia Hockenmaier)

the ball
the ball in the garden
the ball in the garden behind the house
the ball in the garden behind the house near the school ....

- recursion can be more complex
- but we can still model it with FSAs!
- so why bother to go beyond finite-state?

FSAs can’t go hierarchical!



(courtesy of Julia Hockenmaier)

- but sentences have a hierarchical structure!
  - so that we can infer the *meaning*
  - we need not only strings, but also trees
- FSAs are flat, and can only do tail recursions (i.e., loops)
- but we need real (branching) recursions for languages

FSAs can't do Center Embedding

(courtesy of Julia Hockenmaier)

The mouse ate the corn.

The mouse that the snake ate ate the corn.

The mouse that the snake that the hawk ate ate ate the corn.

....

vs.

The claim that the house he bought was valuable was wrong.

vs.

I saw the ball in the garden behind the house near the school.

- in theory, these infinite recursions are still grammatical
  - competence (grammatical knowledge)
- in practice, studies show that English has a limit of 3
  - performance (processing and memory limitations)
- FSAs *can* model *finite* embeddings, but very *inconvenient.*

How about Recursive FSAs?

- problem of FSAs: only tail recursions, no branching recursions
  - can’t represent hierarchical structures (trees)
  - can’t generate center-embedded strings
- is there a simple way to improve it?
  - recursive transition networks (RTNs)

```
----------------------------------------     -----------------------------------
S                                        |   VP                                  |
               NP         VP             |             V         NP             |
        -> 0 ------> 1 ------> 2 ->      |        -> 0 ------> 1 ------> 2 ->  |
----------------------------------------     -----------------------------------
----------------------------------------
NP                                       |
               Det        N              |
        -> 0 ------> 1 ------> 2 ->      |
----------------------------------------
```

Context-Free Grammars

- S → **NP VP**
- **NP → Det N**
- **NP → NP PP**
- PP → P NP
- VP → V NP
- VP → VP PP
- …

- **N →** *{ball, garden, house, sushi }*
- **P →** *{in, behind, with}*
- **V → …**
- **Det → …**

Context-Free Grammars

A CFG is a 4-tuple $\langle N, \Sigma, R, S \rangle$

A set of nonterminals N
(e.g. N = {S, NP, VP, PP, Noun, Verb, ...})

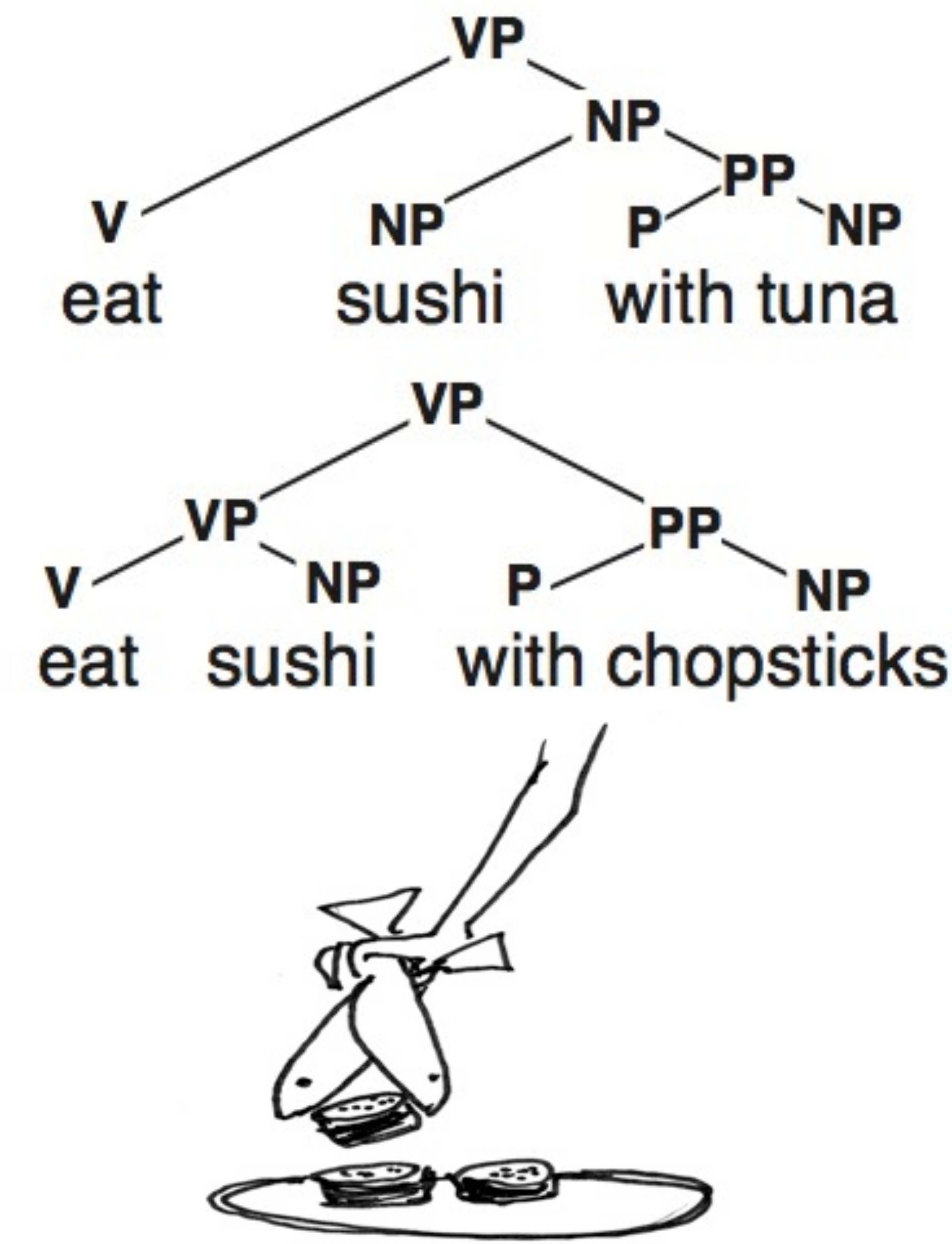
A set of terminals Σ
(e.g. Σ = {*I, you, he, eat, drink, sushi, ball,* })

A set of rules R
R $\subseteq$ **{**$A \rightarrow \beta$ **with left-hand-side (LHS)** $A \in N$
and right-hand-side (RHS) $\beta \in$ **(N** $\cup$ Σ**)* }**

A start symbol S (sentence)

Parse Trees

- **N** → *{sushi, tuna}*
- **P** → *{with}*
- **V** → *{eat}*
- **NP** → **N**
- NP → NP PP
- PP→P NP
- VP→V NP
- VP→VP PP



CFGs for Center-Embedding

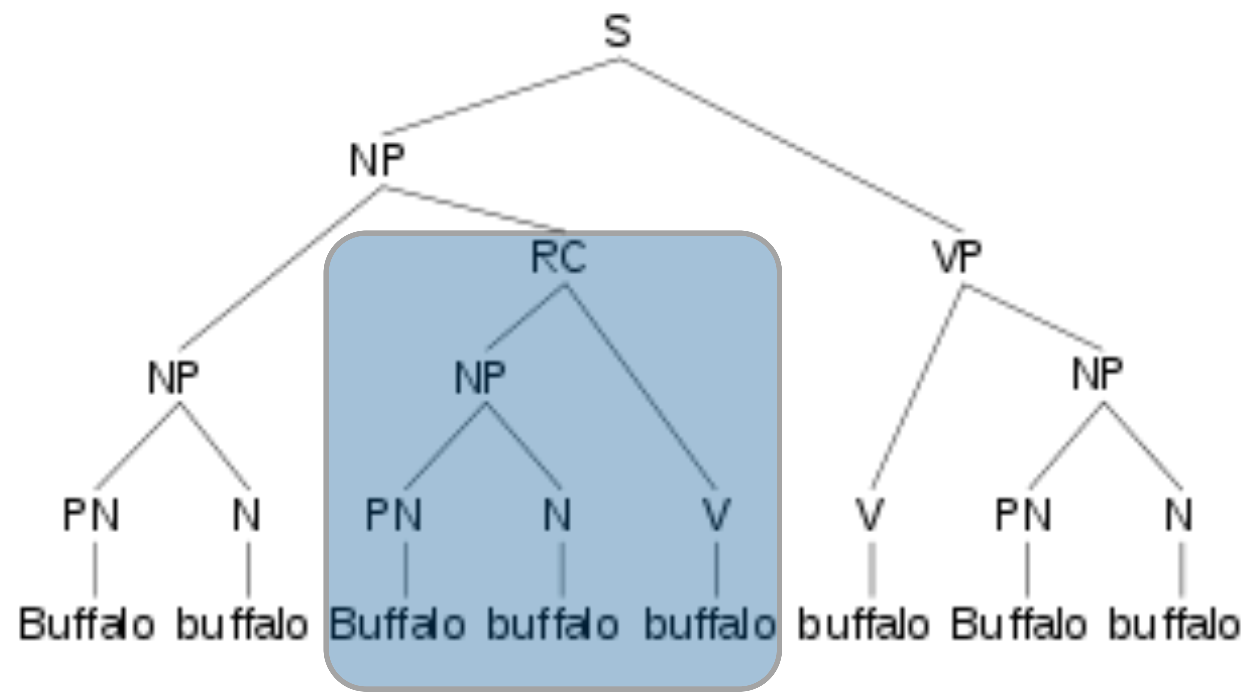
The mouse ate the corn.
The mouse that the snake ate ate the corn.
The mouse that the snake that the hawk ate ate ate the corn.

....

- $\{ a^n b^n \}$ $\{ w w^R \}$
- can you also do $\{ a^n b^n c^n \}$? or $\{ w w^R w \}$?
- $\{ a^n b^n c^m d^m \}$
- what's the limitation of CFGs?
- CFG for center-embedded clauses:
  - S → **NP ate NP; NP → NP RC; RC → that NP ate**

Review

- write a CFG for...
  - $\{ a^m b^n c^n d^m \}$
  - $\{ a^m b^n c^{3m+2n} \}$
  - $\{ a^m b^n c^m d^n \}$
  - buffalo buffalo buffalo ...



- write an FST or synchronous CFG for...
  - $\{ (w, w^R) \}$ $\{ (a^n, b^n) \}$
- HW3: including p(eprons) is wrong
- HW4: using carmel to test your own code

Chomsky Hierarchy

	Language	Automata	Parsing complexity	Dependencies
Type 3	Regular	Finite-state	linear	adjacent words
Type 2	Context-Free	Pushdown	cubic	nested
Type 1	Context-sensitive	Linear Bounded	exponential	
Type 0	Recursively Enumerable	Turing machine		

Constituents, Heads, Dependents

There are different kinds of constituents:

Noun phrases: *the man, a girl with glasses, Illinois*

Prepositional phrases: *with glasses, in the garden*

Verb phrases: *eat sushi, sleep, sleep soundly*

Every phrase has a head:

Noun phrases: *the man, a girl with glasses, Illinois*

Prepositional phrases: *with glasses, in the garden*

Verb phrases: *eat sushi, sleep, sleep soundly*

The other parts are its **dependents.**

Dependents are either **arguments** or **adjuncts**

Constituency Test

He talks *[in class]*.

Substitution test:
Can α be replaced by a single word?
He talks **[there]**.

Movement test:
Can α be moved around in the sentence?
[In class], he talks.

Answer test:
Can α be the answer to a question?
Where does he talk? - **[In class]**.

how about "there is" or "I do"?

Arguments and Adjuncts

- arguments are obligatory

Words subcategorize for specific sets of arguments:

Transitive verbs (sbj + obj): *[John] likes [Mary]*

All arguments have to be present:

**[John] likes.*   **likes [Mary].*

No argument can be occupied multiple times:

**[John] [Peter] likes [Ann] [Mary].*

Words can have multiple *subcat frames:*

Transitive *eat* (sbj + obj): *[John] eats [sushi].*

Intransitive *eat* (sbj): *[John] eats.*

Arguments and Adjuncts

- adjuncts are optional

Adverbs, PPs and adjectives can be adjuncts:

Adverbs: *John runs [fast].*
a [very] heavy book.
PPs: *John runs [in the gym].*
the book [on the table]
Adjectives: a [heavy] book

There can be an arbitrary number of adjuncts

John saw Mary.
John saw Mary [yesterday].
John saw Mary [yesterday] [in town]
John saw Mary [yesterday] [in town] [during lunch]
[Perhaps] John saw Mary [yesterday] [in town] [during lunch]

Noun Phrases (NPs)

Simple NPs:
[He] sleeps. **(pronoun)**
[John] sleeps. **(proper name)**
[A student] sleeps. **(determiner + noun)**

Complex NPs:
[A tall student] sleeps. **(det + adj + noun)**
[The student in the back] sleeps. **(NP + PP)**
[The student who likes MTV] sleeps. **(NP + Relative Clause)**

The NP Fragment

NP → Pronoun
NP → ProperName
NP → Det Noun

Det → *{a, the, every}*
Pronoun → *{he, she,...}*
ProperName → *{John, Mary,...}*
Noun → AdjP Noun
Noun → N
NP → NP PP
NP → NP RelClause

ADJPs and PPs

AdjP → Adj
AdjP → Adv AdjP
Adj → *{big, small, red,…}*
Adv → *{very, really,…}*

PP → P NP
P → *{with, in, above,…}*

Verb Phrase (VP)

He [eats].
He [eats sushi].
He [gives John sushi].
He [eats sushi with chopsticks].

VP → V
VP → V NP
VP → V NP PP
VP → VP PP

V → *{eats, sleeps gives,...}*

VPs redefined

He [eats].
He [eats sushi].
He [gives John sushi].
He [eats sushi with chopsticks].

VP → V_intrans
VP → V_trans NP
VP → V_ditrans NP NP
VP → VP PP
V_intrans→ *{eats, sleeps}*
V_trans→ *{eats}*
V_trans→ *{gives}*

Sentences

[He eats sushi].
[Sometimes, he eats sushi].
[In Japan, he eats sushi].

S → NP VP
S → AdvP S
S → PP S

He says [he eats sushi].
VP → V_comp S
V_comp → *{says, think, believes}*

Sentence Redefined

[He eats sushi]. ✔
[I eats sushi].* **???
[They eats sushi].* **???

S → NP.3sg VP.3sg
S → NP.1sg VP.1sg
S → NP.3pl VP.3pl

We need features to capture agreement:
(number, person, case,…)

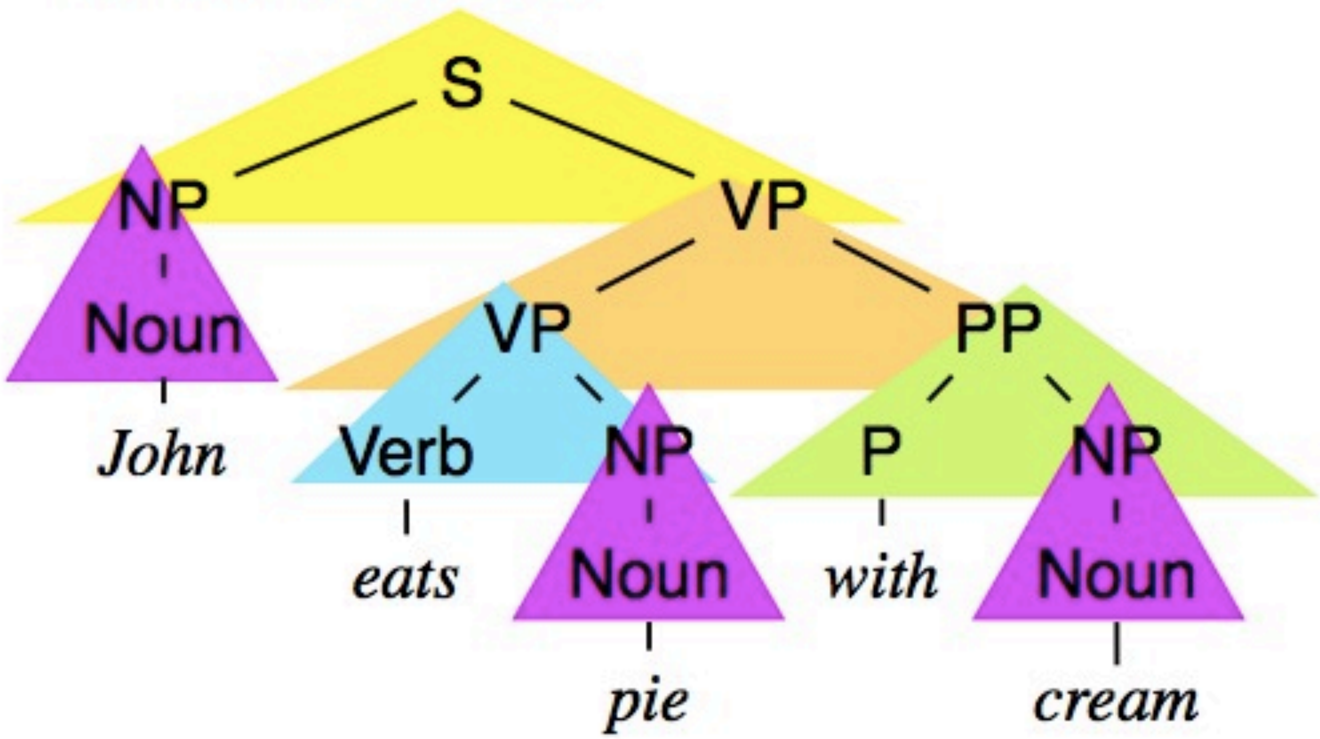
Probabilistic CFG

- normalization
  - $\text{sum}_\beta\ p(A \rightarrow \beta) = 1$
- what's the most likely tree?
  - in finite-state world,
- what's the most likely string
- given string w, what's the most likely tree for w
  - this is called "parsing" (like decoding)

S	$\rightarrow$	NP VP	0.8
S	$\rightarrow$	S conj S	0.2
NP	$\rightarrow$	Noun	0.2
NP	$\rightarrow$	Det Noun	0.4
NP	$\rightarrow$	NP PP	0.2
NP	$\rightarrow$	NP conj NP	0.2
VP	$\rightarrow$	Verb	0.4
VP	$\rightarrow$	Verb NP	0.3
VP	$\rightarrow$	Verb NP NP	0.1
VP	$\rightarrow$	VP PP	0.2
PP	$\rightarrow$	P NP	1.0

Probability of a tree

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



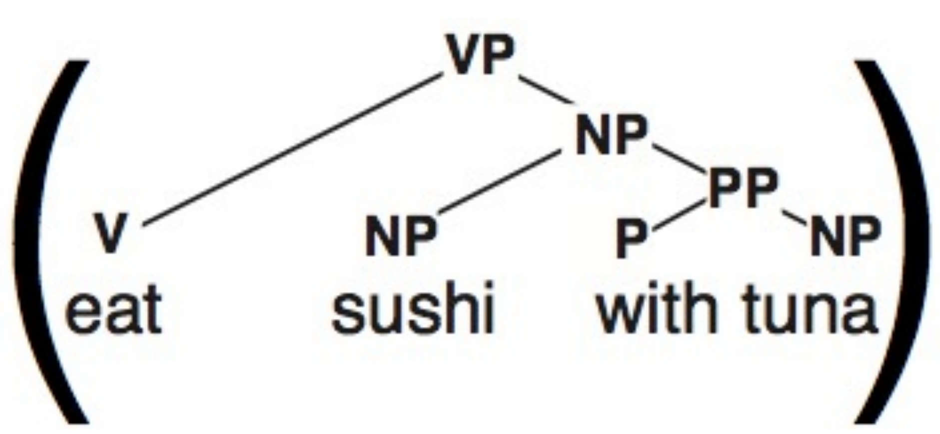
S	$\rightarrow$	NP VP	0.8
S	$\rightarrow$	S conj S	0.2
NP	$\rightarrow$	Noun	0.2
NP	$\rightarrow$	Det Noun	0.4
NP	$\rightarrow$	NP PP	0.2
NP	$\rightarrow$	NP conj NP	0.2
VP	$\rightarrow$	Verb	0.4
VP	$\rightarrow$	Verb NP	0.3
VP	$\rightarrow$	Verb NP NP	0.1
VP	$\rightarrow$	VP PP	0.2
PP	$\rightarrow$	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 given string

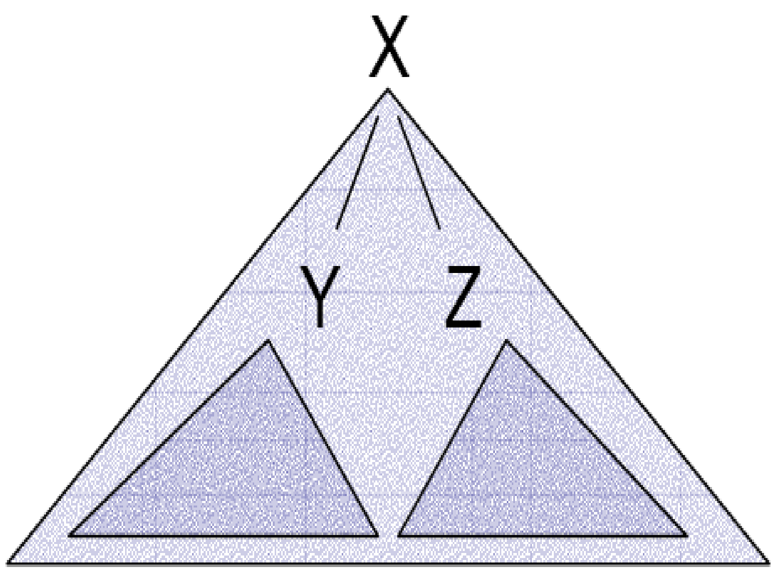
- parsing is to search for the best tree t* that:
  - $t^* = \text{argmax}_t \; p(t \mid w) = \text{argmax}_t \; p(t)\, p(w \mid t)$
  - $= \text{argmax}_{\{t:\, \text{yield}(t)=w\}} \; p(t)$
  - analogous to HMM decoding
- is it related to "intersection" or "composition" in FSTs?

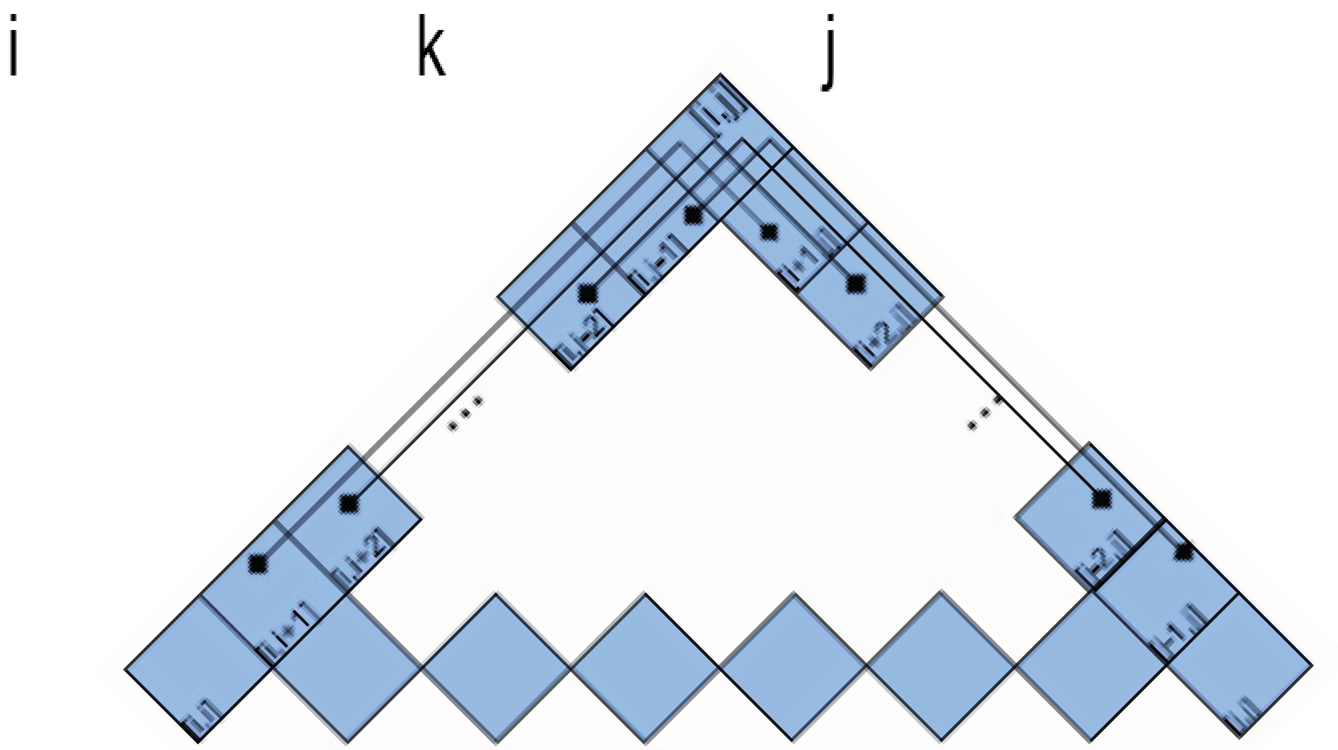
yield  = *eat sushi with tuna*

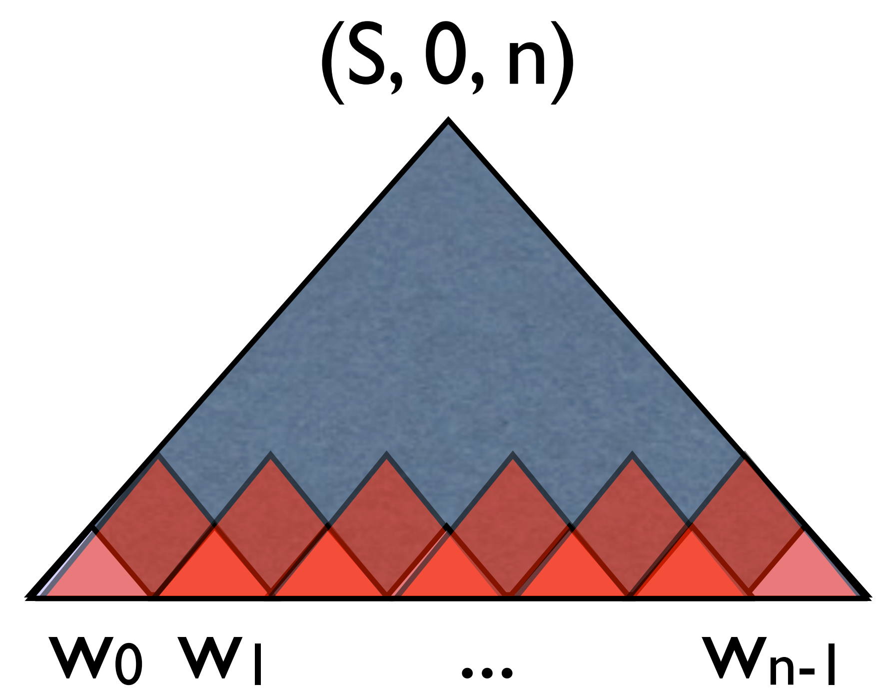
CKY Algorithm

- 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]
```



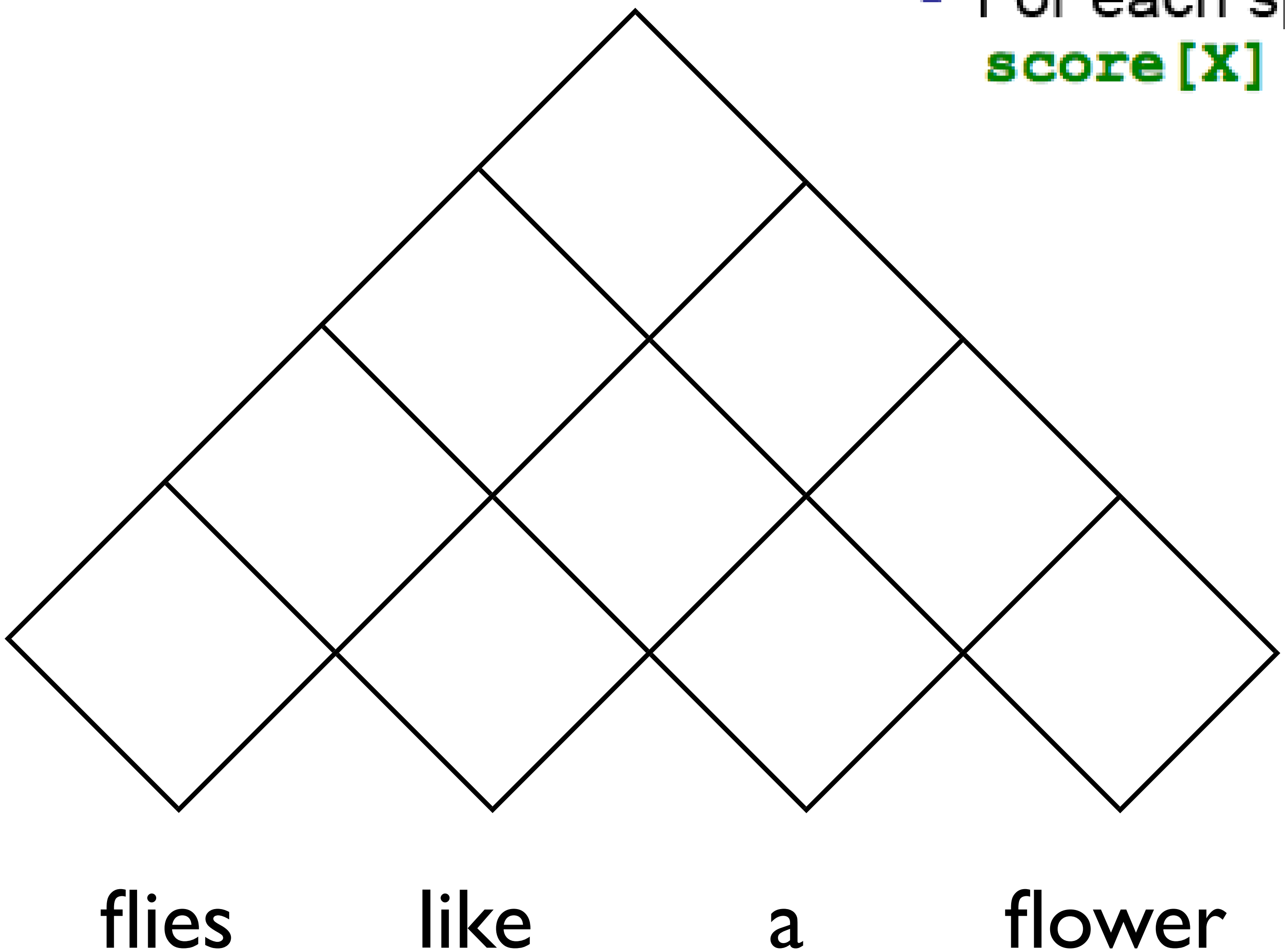




CKY Algorithm

- 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]
```



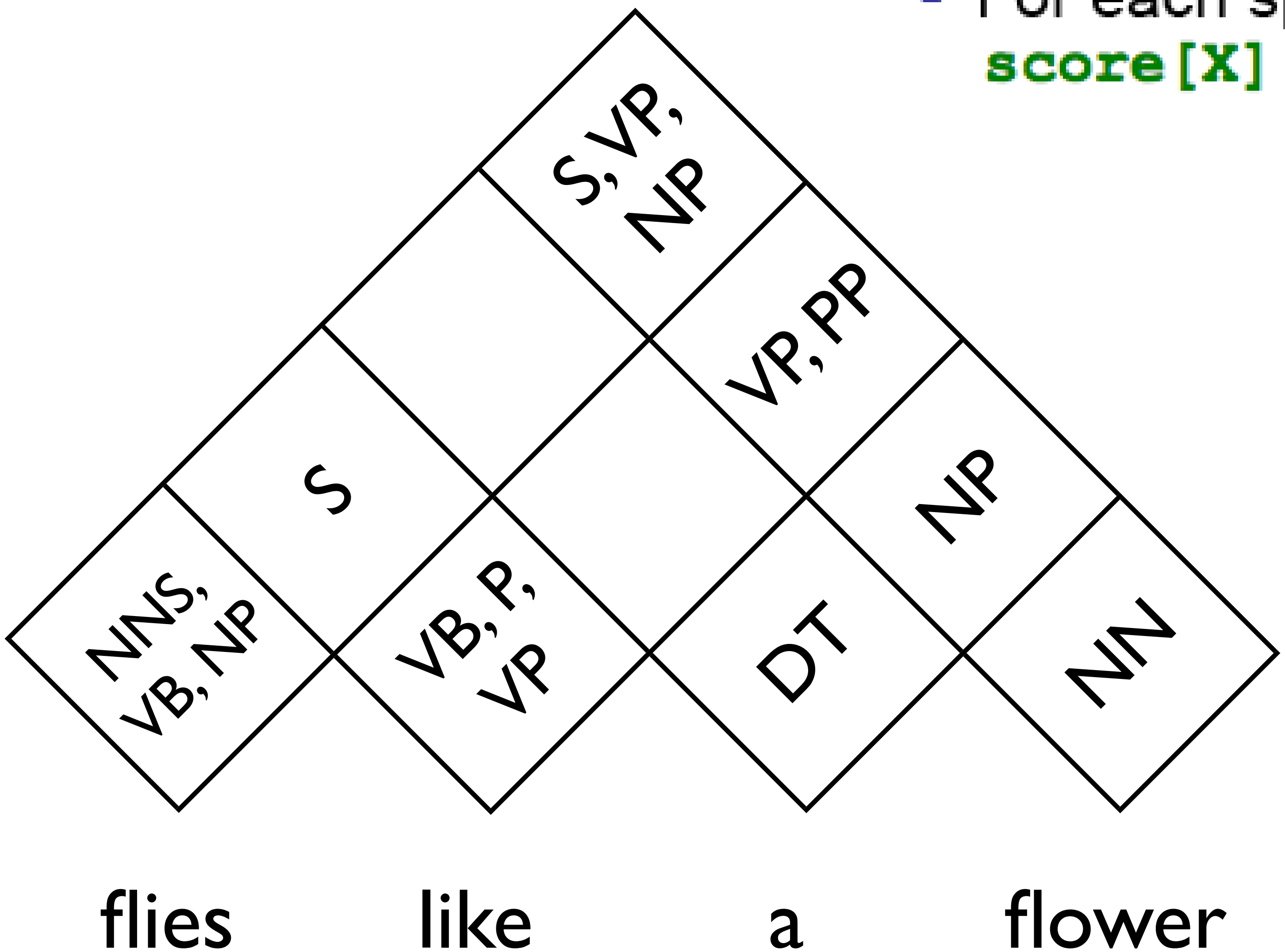
S → NP VP
NP → DT NN
NP → NNS
NP → NP PP
VP → VB NP
VP → VP PP
VP → VB
PP → P NP

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 $X \to Y\ Z$
      - 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]
```



$S \to NP\ VP$
$NP \to DT\ NN$
$NP \to NNS$
$NP \to NP\ PP$
$VP \to VB\ NP$
$VP \to VP\ PP$
$VP \to VB$
$PP \to P\ NP$

$VB \to$ flies
$NNS \to$ flies
$VB \to$ like
$P \to$ like
$DT \to$ a
$NN \to$ flower

$S \to VP$

CKY Example

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
N NP 0.2	S 0.8*0.2*0.4	S 0.8*0.2*0.08		S 0.2*0.0024*0.8	**John**
	V VP 0.4	VP 0.3*0.2		VP max(0.008*0.2, 0.06*0.2*0.2)	**eats**
		N NP 0.2		NP 0.2*0.2*0.2	**pie**
			P	PP 1*0.2	**with**
				N NP 0.2	**cream**

S	$\rightarrow$	NP VP	0.8
S	$\rightarrow$	S conj S	0.2
NP	$\rightarrow$	Noun	0.2
NP	$\rightarrow$	Det Noun	0.4
NP	$\rightarrow$	NP PP	0.2
NP	$\rightarrow$	NP conj NP	0.2
VP	$\rightarrow$	Verb	0.4
VP	$\rightarrow$	Verb NP	0.3
VP	$\rightarrow$	Verb NP NP	0.1
VP	$\rightarrow$	VP PP	0.2
PP	$\rightarrow$	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)
  - $A \rightarrow B\ C$
  - $A \rightarrow a$
- how to deal with epsilon-removal?
- how to do it with PCFG?

What if we don't do CNF...

- Earley's algorithm (dotted rules, internal binarization)

Item form: $[A, i, j]$

Axioms: $[A, i, i+1] \quad A \rightarrow w_{i+1}$

Goals: $[S, 0, n]$

Inference rules: $\dfrac{[B, i, j] \qquad [C, j, k]}{[A, i, k]} \quad A \rightarrow B\ C$

CKY deductive system

What if we don't do CNF...

- Earley's algorithm (dotted rules, internal binarization)

$[0, S' \to \bullet S, 0]$	initial
$[0, S' \to S \bullet, n]$	goal
$\dfrac{[i, A \to \alpha \bullet w_{j+1}\beta, j]}{[i, A \to \alpha w_{j+1} \bullet \beta, j+1]}$	scan
$\dfrac{[i, A \to \alpha \bullet B\beta, j]}{[j, B \to \bullet \gamma, j]} \quad B \to \gamma$	predict
$\dfrac{[i, A \to \alpha \bullet B\beta, k] \quad [k, B \to \gamma \bullet, j]}{[i, A \to \alpha B \bullet \beta, j]}$	complete

Earley (1970) deductive system

Earley Algorithm

- why *complete* must be first?
- how do you extend it for PCFG?

procedure EARLEYPARSER($w_{1..n}$)
  addToChart($\langle TOP \rightarrow \bullet S, [0,0] \rangle$)
  for all $\langle rule, [k,i] \rangle \in$ `chart` **do**
    if *rule* matches $X \rightarrow \gamma\bullet$ **then** ▷ COMPLETE X
      for all $\langle Y \rightarrow \alpha \bullet X \beta, [j,k] \rangle \in$ `chart` **do**
        addToChart($\langle Y \rightarrow \alpha X \bullet \beta, [j,i] \rangle$)
    else if *rule* matches $X \rightarrow \alpha \bullet Y \beta$ **then** ▷ PREDICT Y
      for all $Y \rightarrow \gamma \in$ RULES **do**
        addToChart($\langle Y \rightarrow \bullet \gamma, [i,i] \rangle$)
    else if *rule* matches $X \rightarrow \alpha \bullet t \beta$, and $\langle t, w_i \rangle \in$ LEX **then** ▷ SCAN w_i
      addToChart($\langle X \rightarrow \alpha\, t \bullet \beta, [k, i+1] \rangle$)
    if s=$\langle TOP \rightarrow S\bullet, [0,n] \rangle \in$ `chart` **then return** s
    else fail

Parsing as Deduction

$$\frac{(B, i, k) : a \quad (C, k, j) : b}{(A, i, j) : a \times b \times \Pr(A \rightarrow B\ C)}\ A \rightarrow B\ C$$

Parsing as Intersection

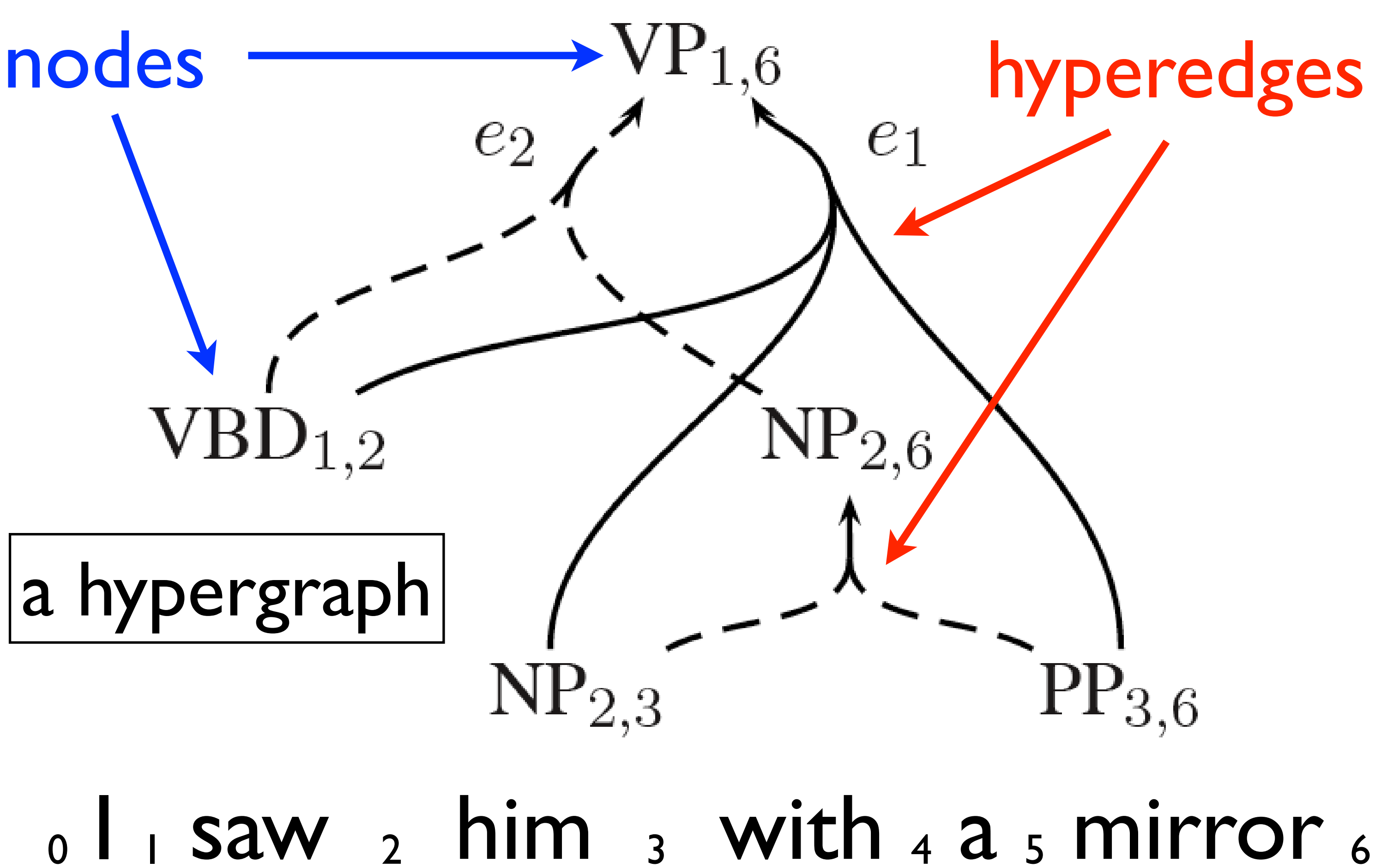
$$\frac{(B, i, k) : a \quad (C, k, j) : b}{(A, i, j) : a \times b \times \Pr(A \to B\ C)} \; A \to B\ C$$

- intersection between a CFG *G* and an FSA *D*:
  - define *L(G)* to be the set of *strings* (*i.e., yields*) *G* generates
  - define $L(G \cap D) = L(G) \cap L(D)$
  - what does this new language generate??
  - what does the new grammar *look like*?
- what about CFG ∩ CFG ?

Parsing as Composition

Packed Forests

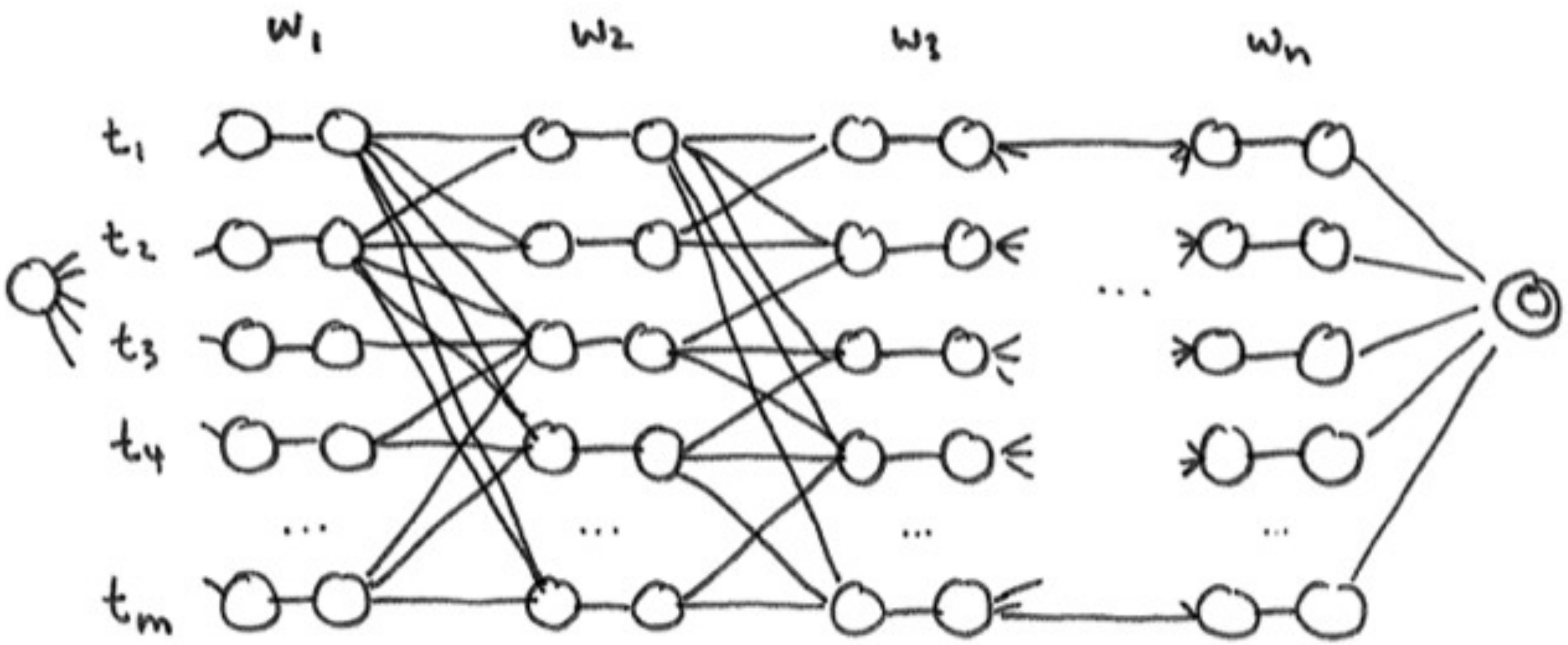
- a compact representation of many parses
  - by sharing common sub-derivations
  - polynomial-space encoding of exponentially large set

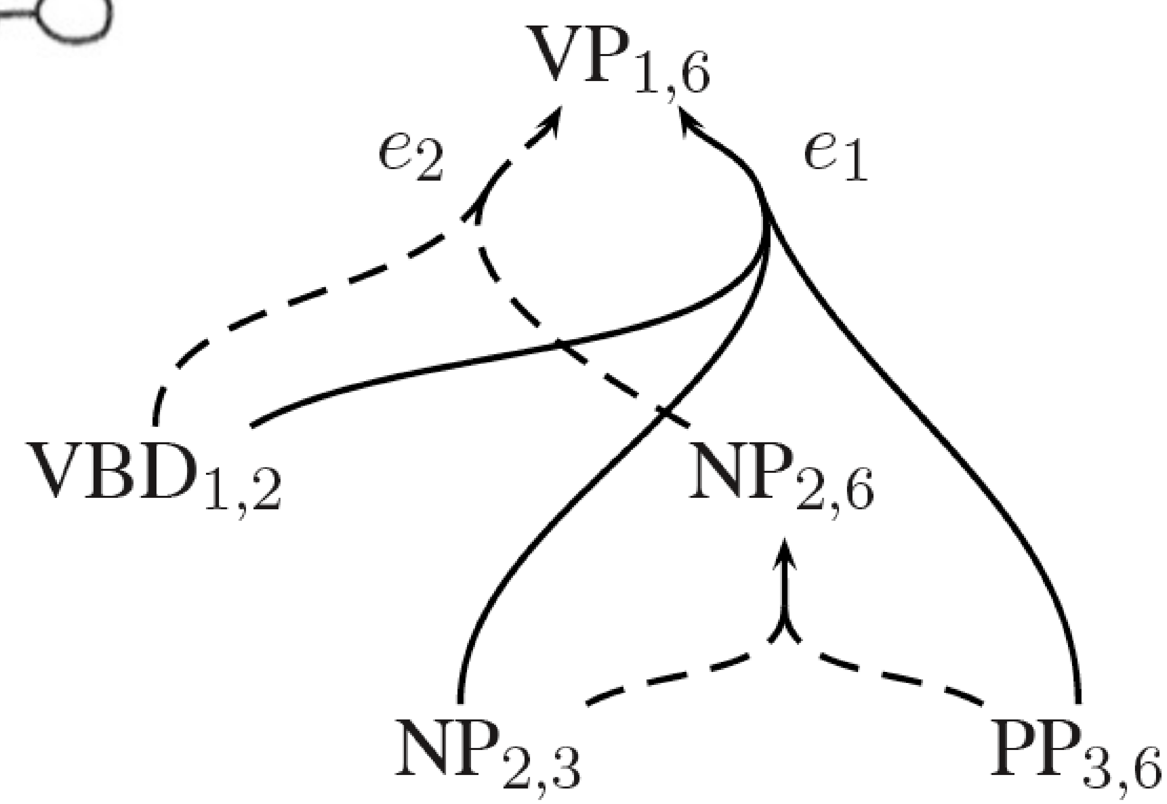


$$e_1 \quad \frac{\mathrm{VBD}_{1,2} \quad \mathrm{NP}_{2,3} \quad \mathrm{PP}_{3,6}}{\mathrm{VP}_{1,6}}$$

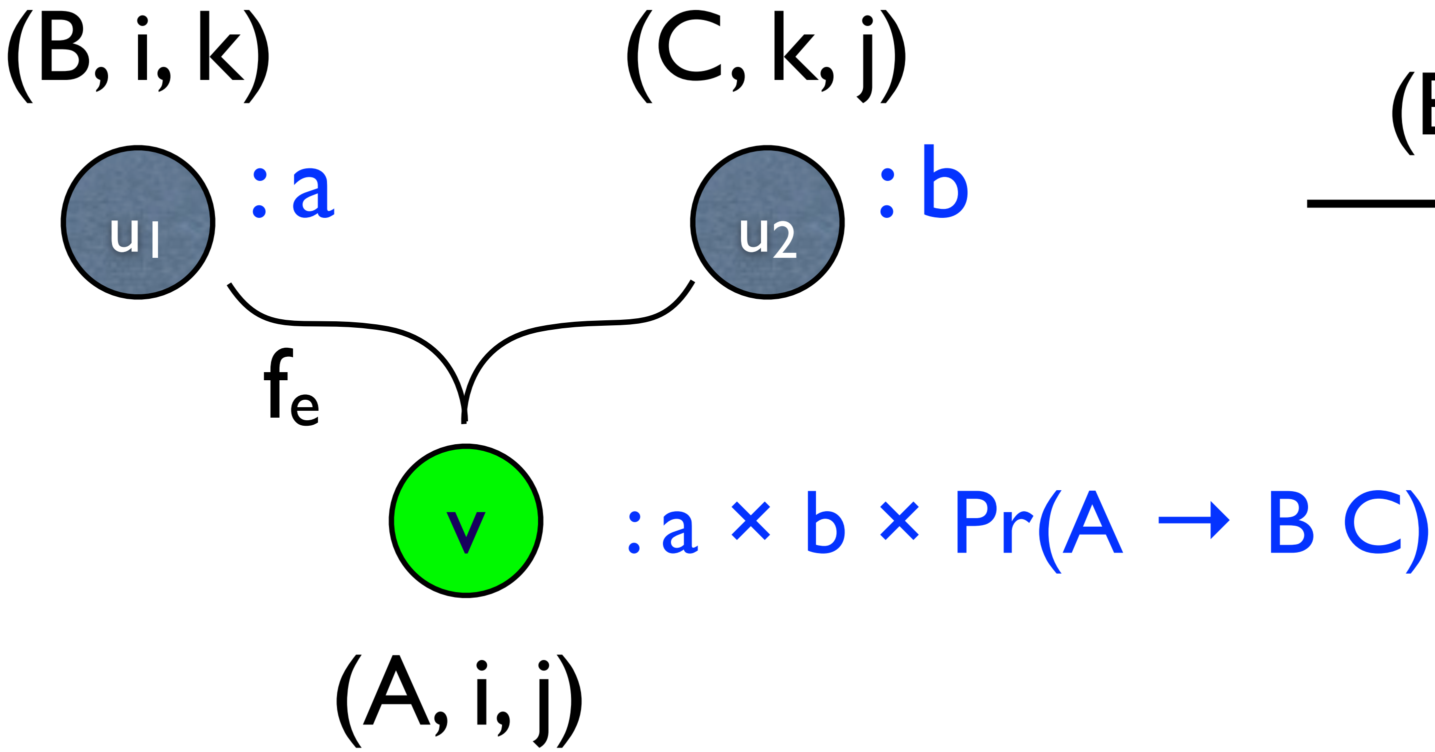
(Klein and Manning, 2001; Huang and Chiang, 2005)

Lattice vs. Forest



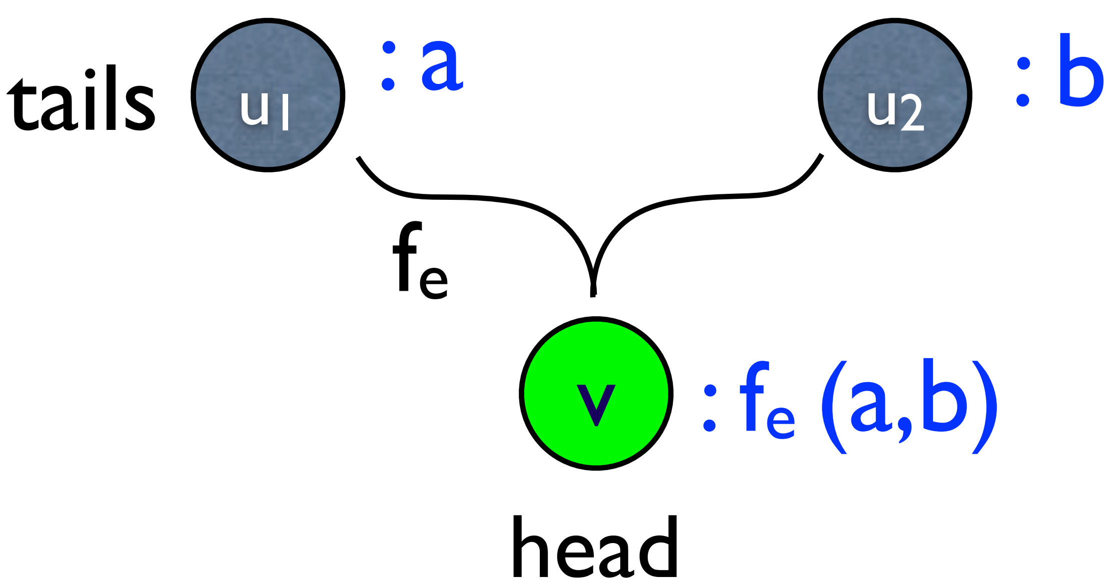


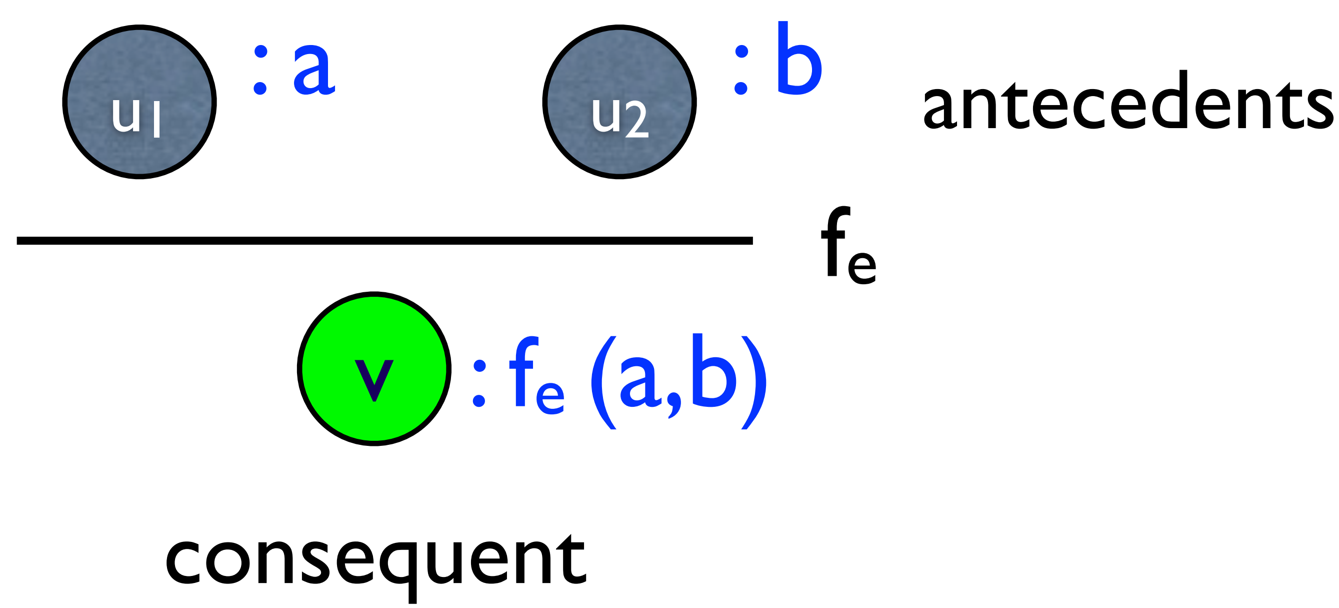
Forest and Deduction



$$\frac{(B, i, k) \qquad (C, k, j)}{(A, i, j)} \; A \to B\ C$$

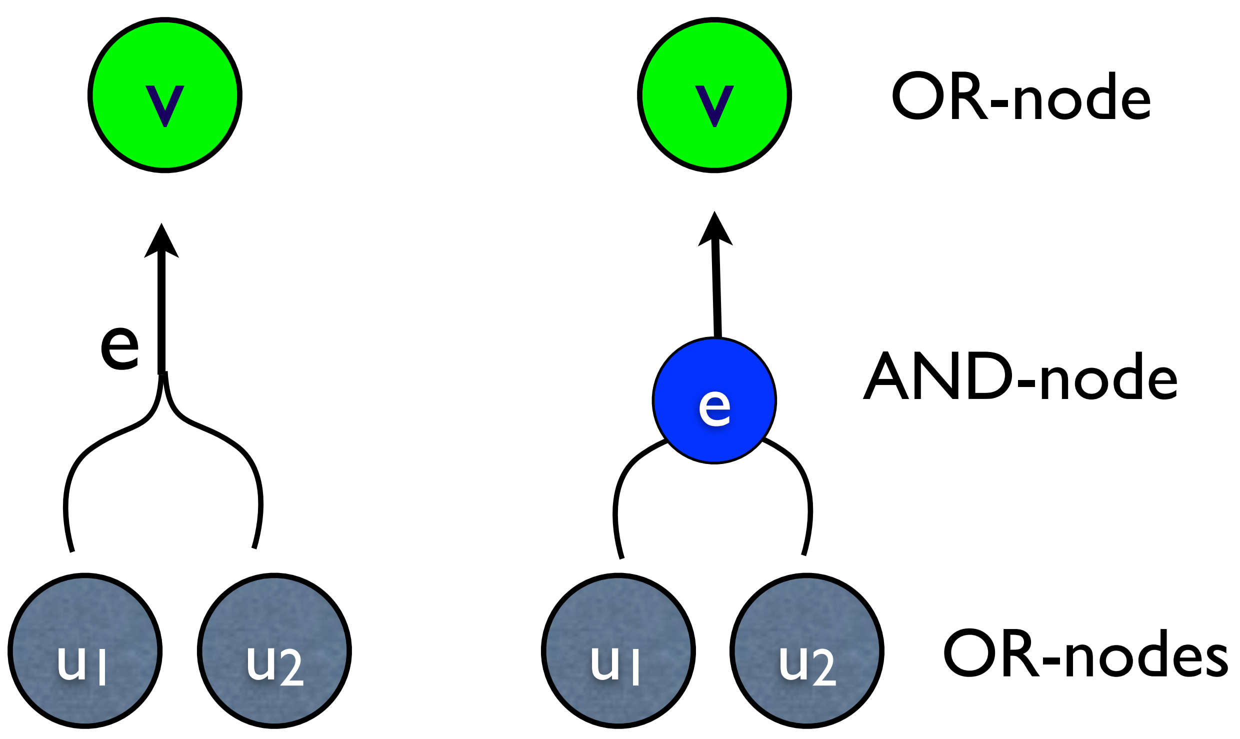
(Nederhof, 2003)





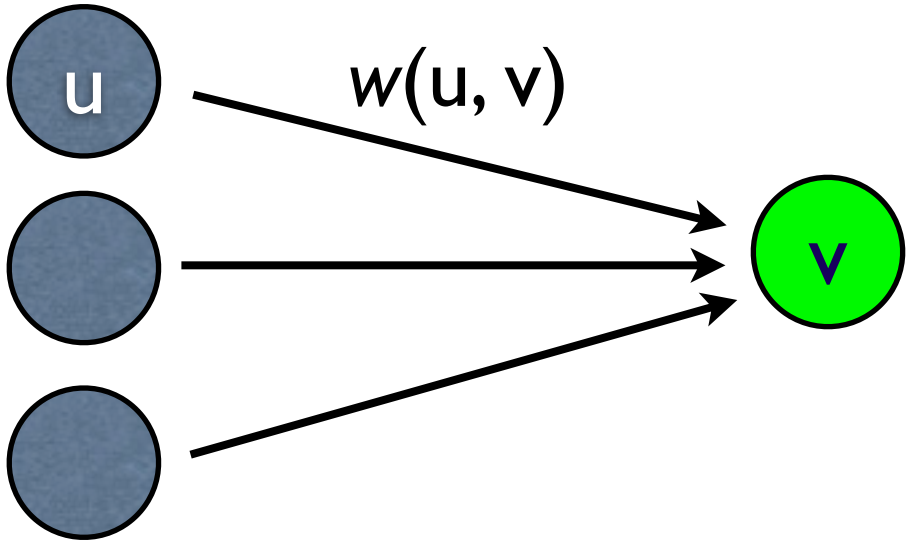
Related Formalisms

hypergraph	AND/OR graph	context-free grammar	deductive system
vertex	OR-node	symbol	item
source-vertex	leaf OR-node	terminal	axiom
target-vertex	root OR-node	start symbol	goal item
hyperedge	AND-node	production	instantiated deduction
$(\{u_1, u_2\}, v, f)$		$v \xrightarrow{f} u_1\ u_2$	$\dfrac{u_1 : a \quad u_2 : b}{v : f(a,b)}$



Viterbi Algorithm for DAGs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v):
   - key observation: d(u) is fixed to optimal at this time

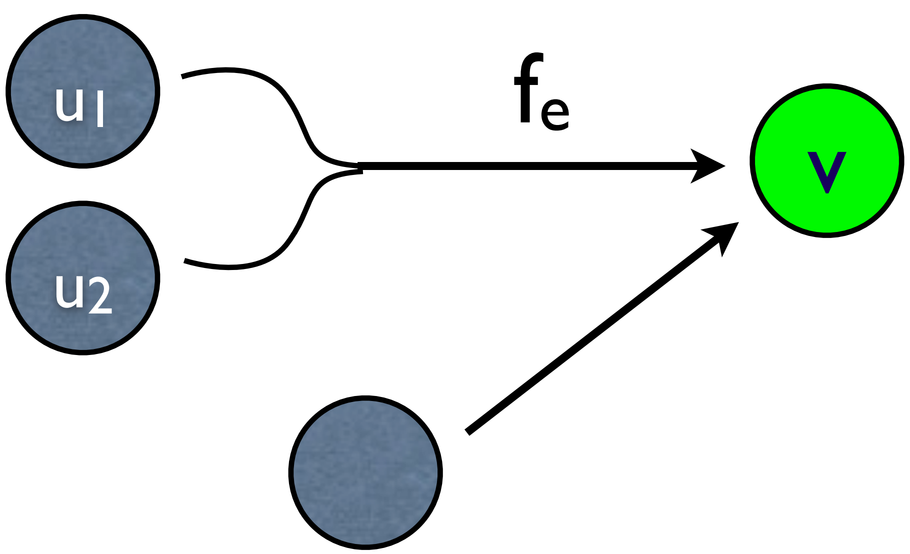


$$d(v) \oplus= d(u) \otimes w(u, v)$$

- time complexity: O(V + E)

Viterbi Algorithm for DAHs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming hyperedge $e = ((u_1, .., u_{|e|}), v, f_e)$
   - use $d(u_i)$'s to update $d(v)$
   - key observation: $d(u_i)$'s are fixed to optimal at this time



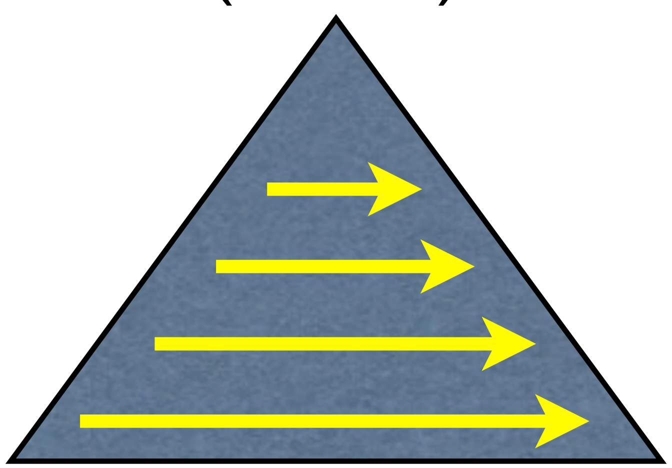
$$d(v) \oplus = f_e(d(u_1), \cdots, d(u_{|e|}))$$

- time complexity: O(V + E) (assuming constant arity)

Example: CKY Parsing

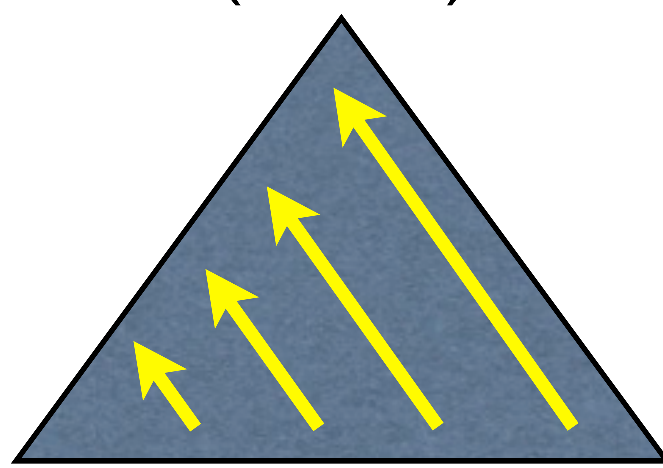
- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering

(S, 0, n)



bottom-up

(S, 0, n)



left-to-right

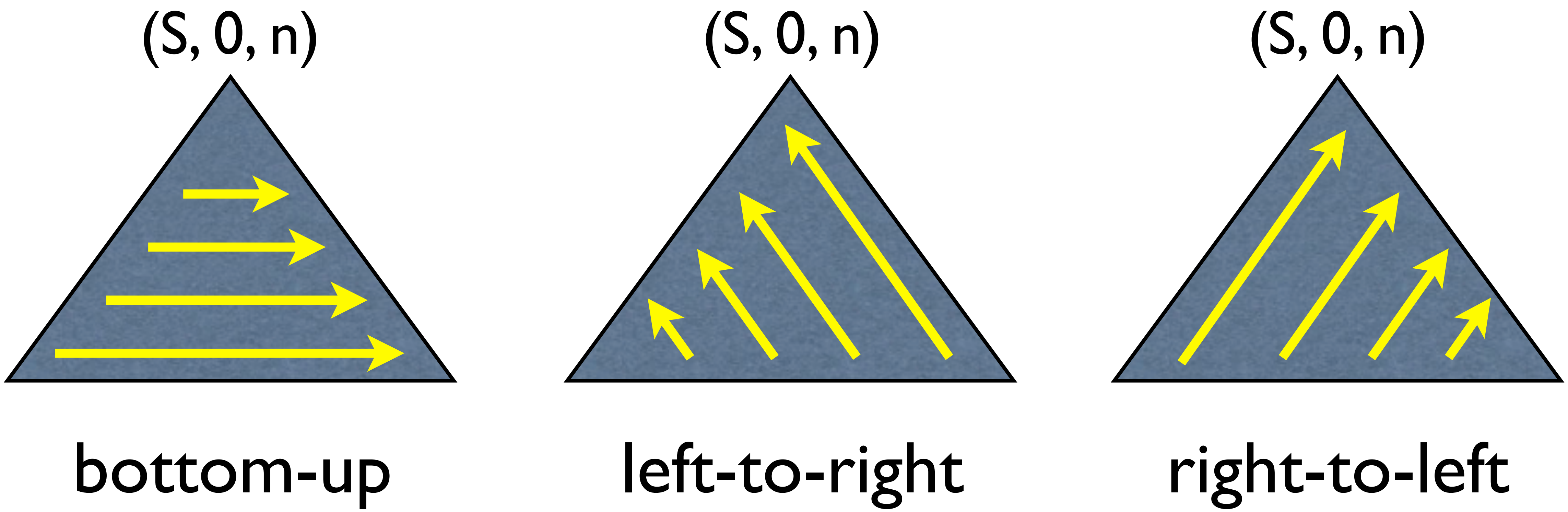
- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max
```

$O(n^3|P|)$

Example: CKY Parsing

- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering



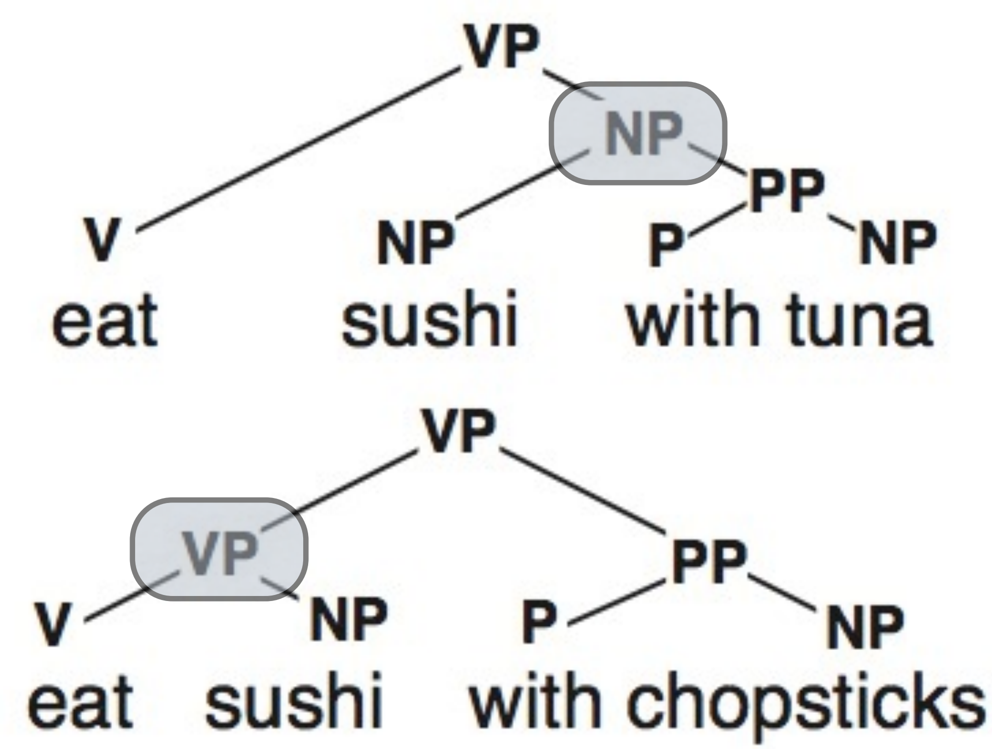
$O(n^3|P|)$

Parser/Tree Evaluation

- how would you evaluate the quality of output trees?
- need to define a "similarity measure" between trees
  - for sequences, we used
    - same length: hamming distance (e.g., POS tagging)
    - varying length: edit distance (e.g., Japanese transliteration)
    - varying length: precision/recall/F (e.g., word-segmentation)
    - varying length: crossing brackets (e.g., word-segmentation)
  - for trees, we use precision/recall/F and crossing brackets
    - standard "PARSEVAL" metrics (implemented as evalb.py)

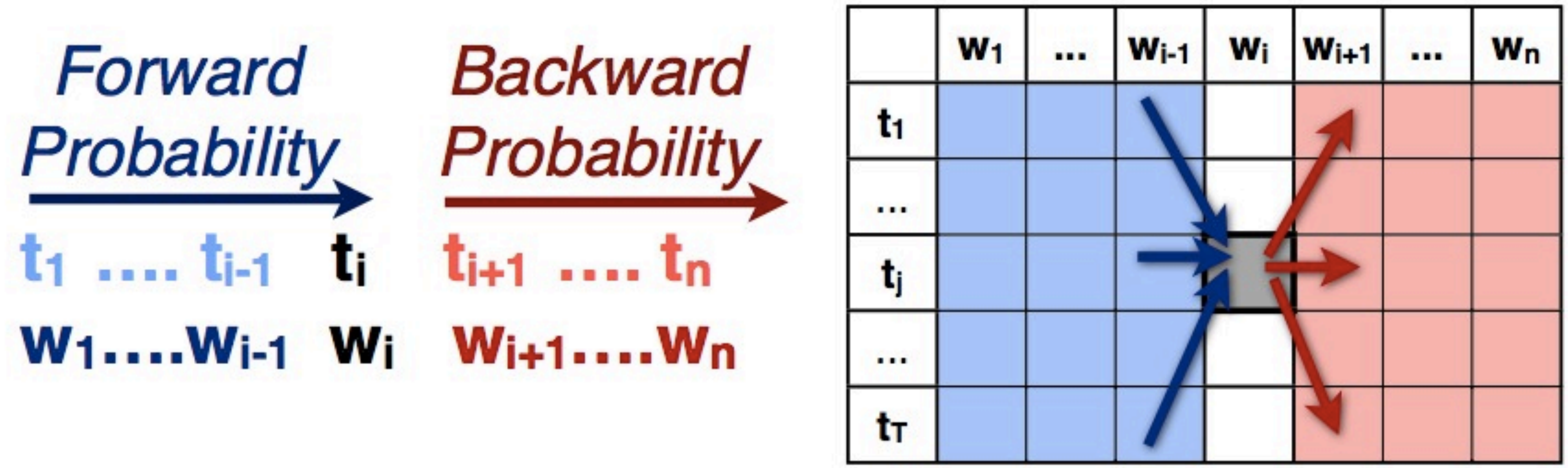
PARSEVAL

- comparing nodes ("brackets"):
  - labelled (by default): (NP, 2, 5); or unlabelled: (2, 5)
- precision: how many predicted nodes are correct?
- recall: how many correct nodes are predicted?
- how to *fake* precision or recall?
- F-score: F=2pr/(p+r)
- other metrics: crossing brackets



matched=6
predicted=7
gold=7
precision=6/7
recall=6/7
F=6/7

Inside-Outside Algorithm



Forward Probability of t_i: $\alpha_i(t)$

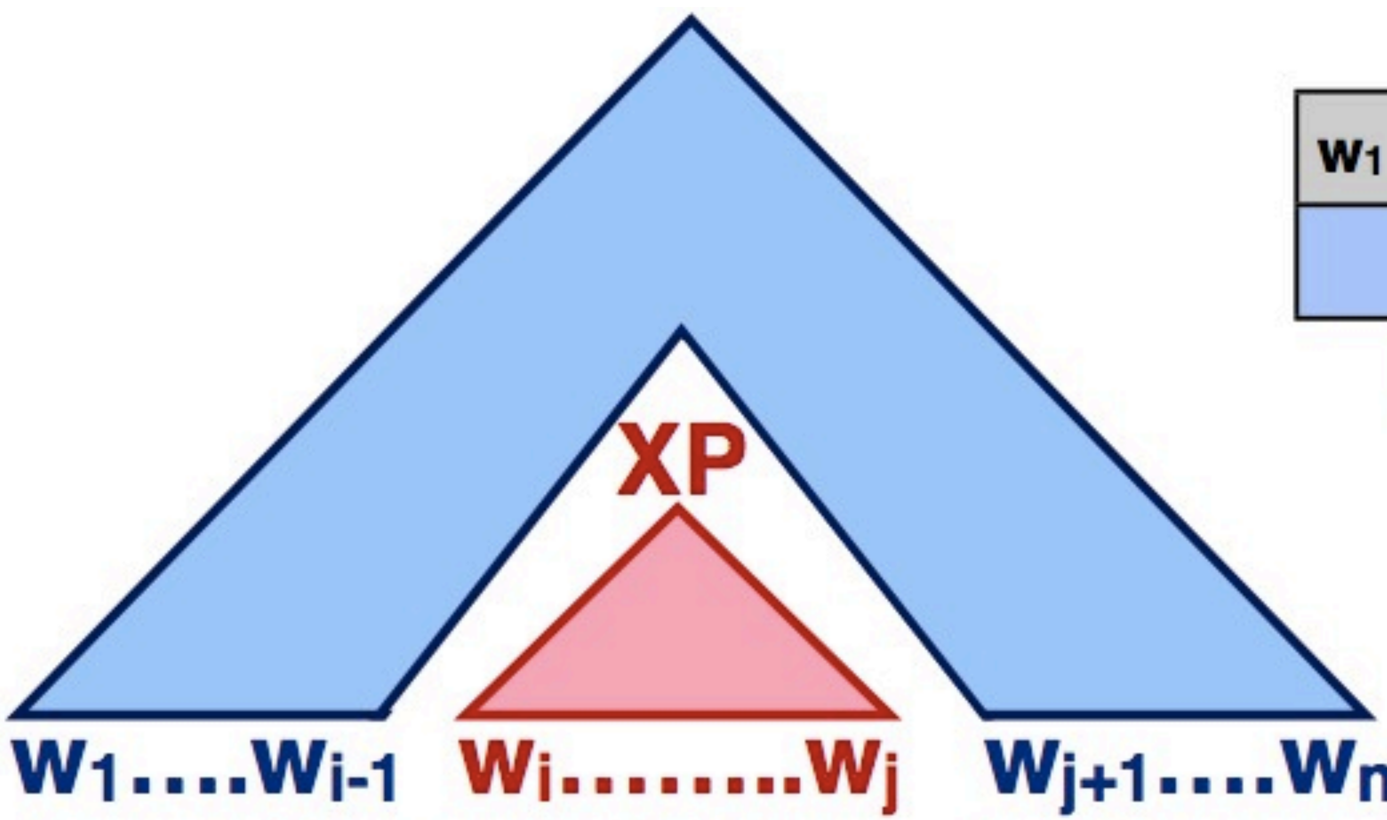
$\alpha_i(t) = P(w_1 ... w_i,\ tag_i = t_i)$

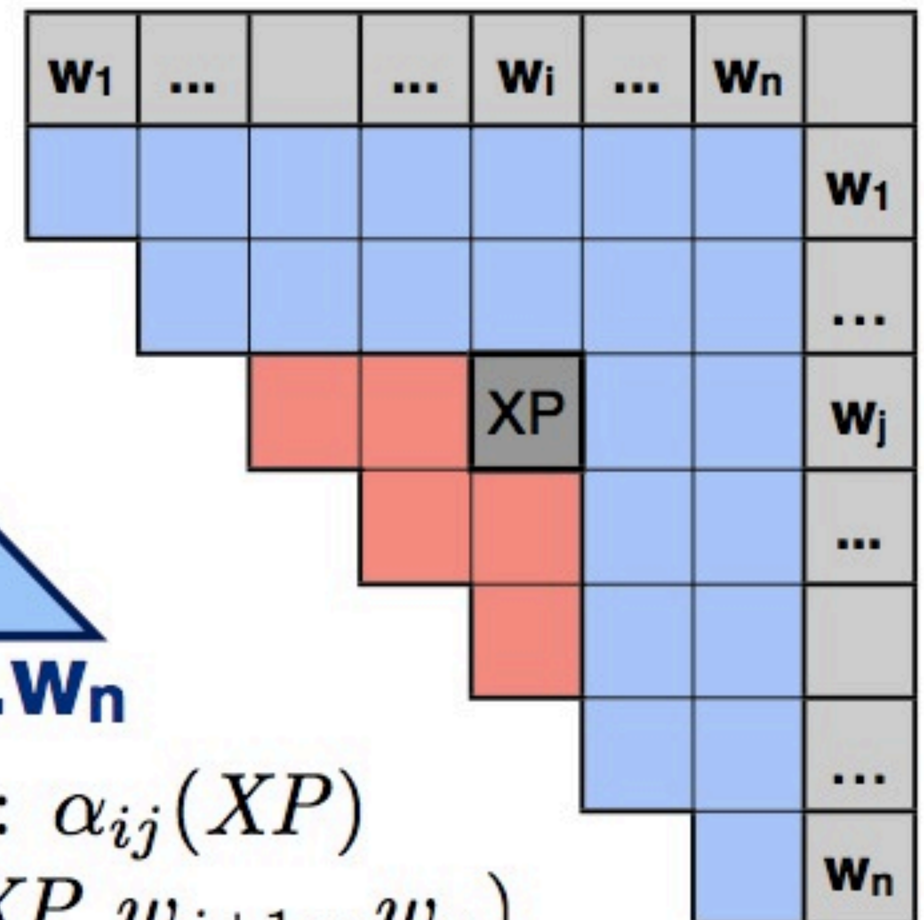
Backward Probability of t_i: $\beta_i(t)$

$\beta_i(t) = P(w_{i+1} ... w_n | tag_i = t)$

CS 498 JH: **Introduction to NLP** (Fall '08)

Inside-Outside Algorithm





Outside Probability of $XP_{i..j}$: $\alpha_{ij}(XP)$

$$\alpha_{ij}(XP) = P(S \Rightarrow^* \quad w_1..w_{i-1}\ XP\ w_{j+1}...w_n)$$

Inside Probability of $XP_{i..j}$: $\beta_{ij}(XP)$

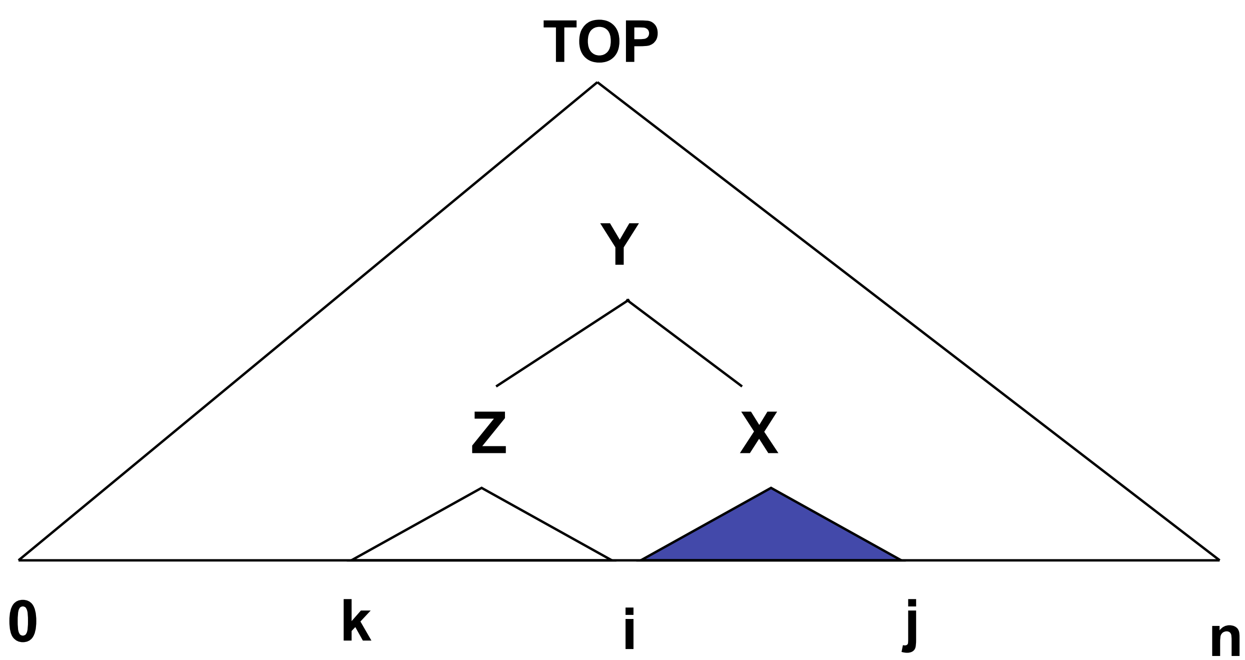
$$\beta_{ij}(XP) = P(XP \Rightarrow^* w_i...w_j)$$

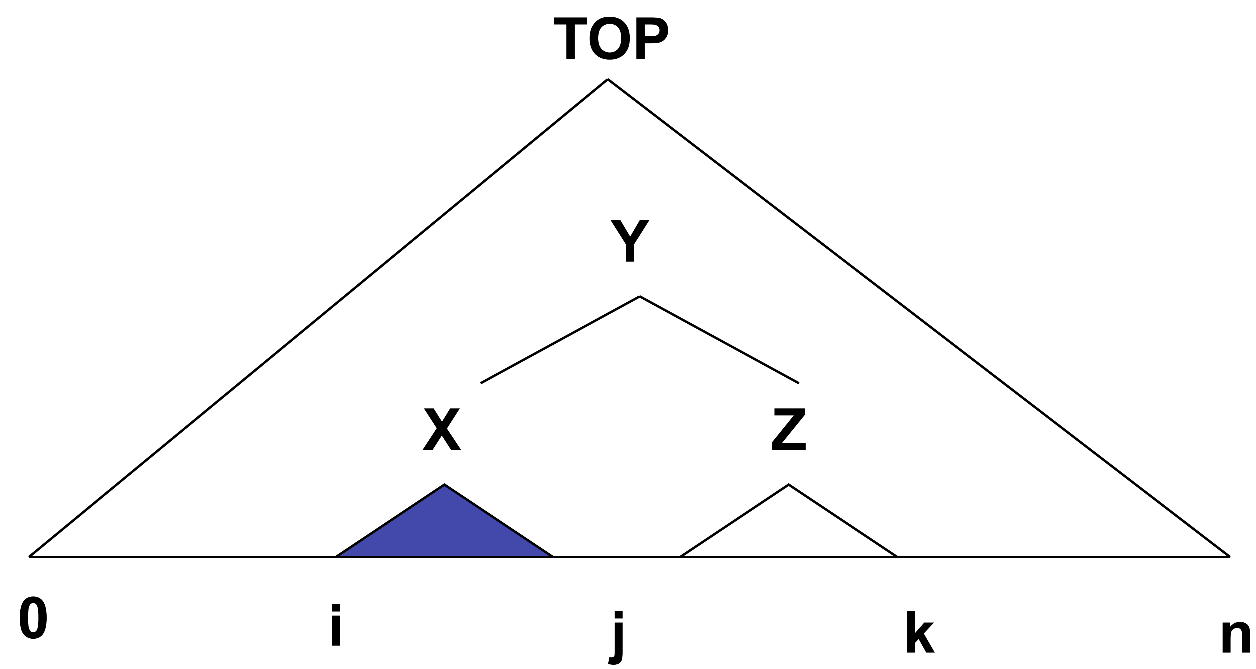
CS 498 JH: **Introduction to NLP** (Fall '08)

Inside-Outside Algorithm

- inside prob beta is easy to compute (CKY, max=>+)
- what is outside prob alpha(X,i,j)?
  - need to enumerate ways to go to TOP from X,i,j
  - X,i,j can be combined with other nodes on the left/right
  - L: sum_{Y->Z X, k} alpha(Y,k,j) Pr(Y->Z X) beta(Z,k,i)
  - R: sum_{Y->X Z, k} alpha(Y,i,k) Pr(Y->X Z) beta(Z,j,k)
  - why beta is used in alpha? very diff. from F-W algorithm
- what is the likelihood of the sentence?
  - beta(TOP, 0, n) or alpha(w_i, i, i+1) for *any* i

Inside-Outside Algorithm





- L: sum_{Y->Z X, k} alpha(Y,k,j) Pr(Y->Z X) beta(Z,k,i)
- R: sum_{Y->X Z, k} alpha(Y,i,k) Pr(Y->X Z) beta(Z,j,k)

Inside-Outside Algorithm

- how do you do EM with alphas and betas?
  - easy; M-step: divide by fractional counts
  - fractional count of rule (X,i,j -> Y,i,k Z,k,j) is
    - alpha(X,i,j) prob(Y Z|X) beta(Y,i,k) beta(Z,k,j)
- if we replace "+" by "max", what will alpha/beta mean?
  - beta': Viterbi inside: best way to derive X,i,j
  - alpha': Viterbi outside: best way to go to TOP from X,i,j
- now what is alpha'(X, i, j) beta'(X, i, j)?
  - best derivation that contains X,i,j (useful for pruning)

Viterbi => CKY

traversing order

search space

	topological (acyclic)	best-first (superior)
graphs with semirings (e.g., FSMs)	Viterbi	Dijkstra
hypergraphs with weight functions (e.g., CFGs)	Gen. Viterbi (e.g., CKY)	Knuth

How to generate from a CFG?

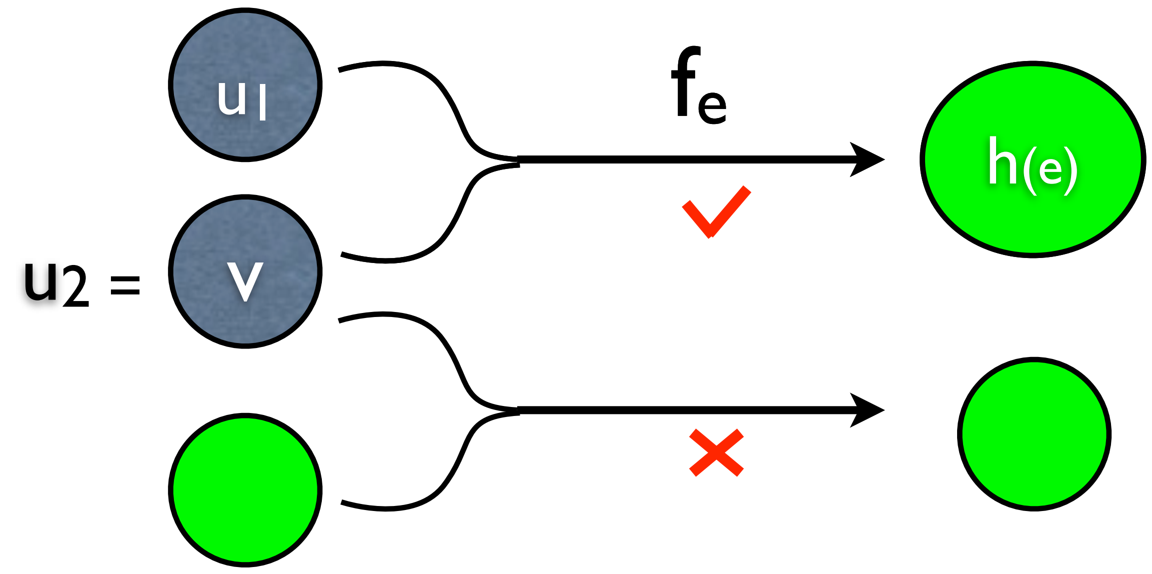
- analogy in finite-state world: given a WFSA, generate strings (either randomly or in order)
- Viterbi doesn't work (cycles)
- Dijkstra still works (as long as it's probabilities)
- What's the generalization of Dijkstra in the tree world?

Forward Variant for DAHs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each outgoing hyperedge e = $((u_1, .., u_{|e|}), h(e), f_e)$
   - if $d(u_i)$'s have all been fixed to optimal
     - use $d(u_i)$'s to update $d(h(e))$

$v = u_i$

Q: how to avoid repeated checking?
maintain a counter r[e] for each e:
how many tails yet to be fixed?
fire this hyperedge only if r[e]=0



- time complexity: O(V + E)

Example: Treebank Parsers

- State-of-the-art statistical parsers
  - (Collins, 1999; Charniak, 2000)
  - no fixed grammar (every production is possible)
  - can't do backward updates
    - don't know how to decompose a big item
  - forward update from vertex (X, i, j)
    - check all vertices like (Y, j, k) or (Y, k, i) in the chart (fixed)
    - try combine them to form bigger item (Z, i, k) or (Z, k, j)

Two Dimensional Survey

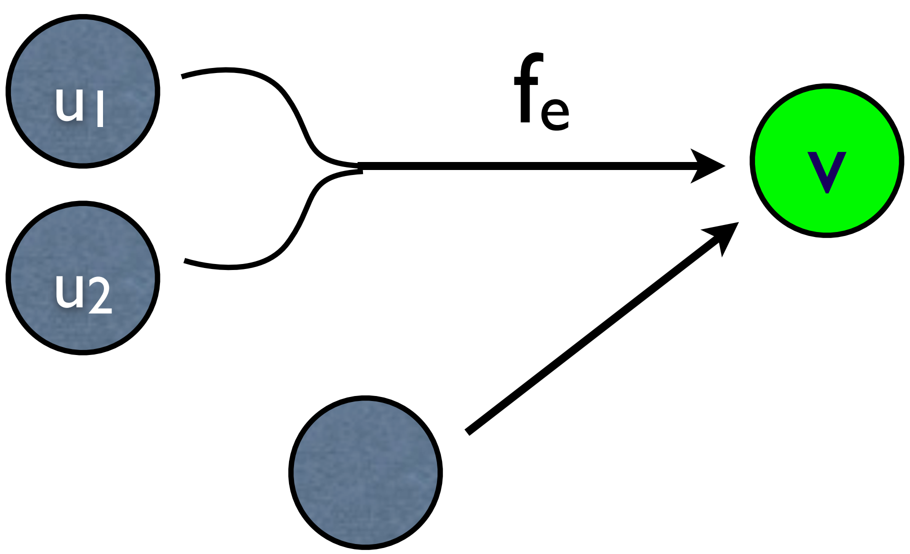
traversing order

search space

	topological (acyclic)	best-first (superior)
graphs with semirings (e.g., FSMs)	Viterbi	Dijkstra
hypergraphs with weight functions (e.g., CFGs)	Generalized Viterbi	Knuth

Viterbi Algorithm for DAHs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming hyperedge $e = ((u_1, .., u_{|e|}), v, f_e)$
   - use $d(u_i)$'s to update $d(v)$
   - key observation: $d(u_i)$'s are fixed to optimal at this time



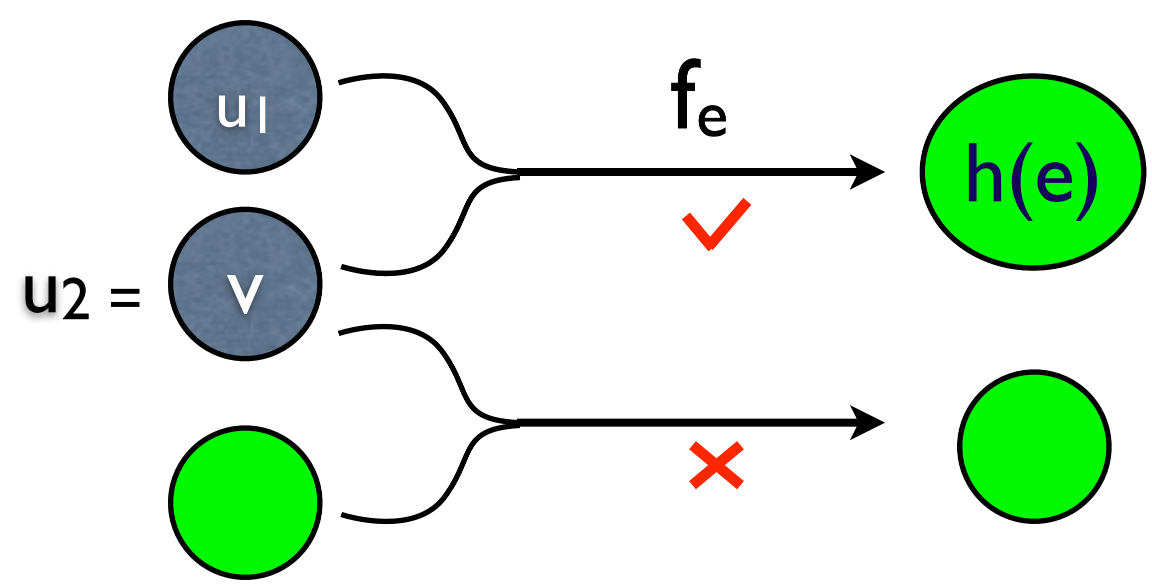
$$d(v) \oplus = f_e(d(u_1), \cdots, d(u_{|e|}))$$

- time complexity: O(V + E) (assuming constant arity)

Forward Variant for DAHs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each outgoing hyperedge $e = ((u_1, .., u_{|e|}), h(e), f_e)$
   - if $d(u_i)$'s have all been fixed to optimal
     - use $d(u_i)$'s to update $d(h(e))$

$v = u_i$

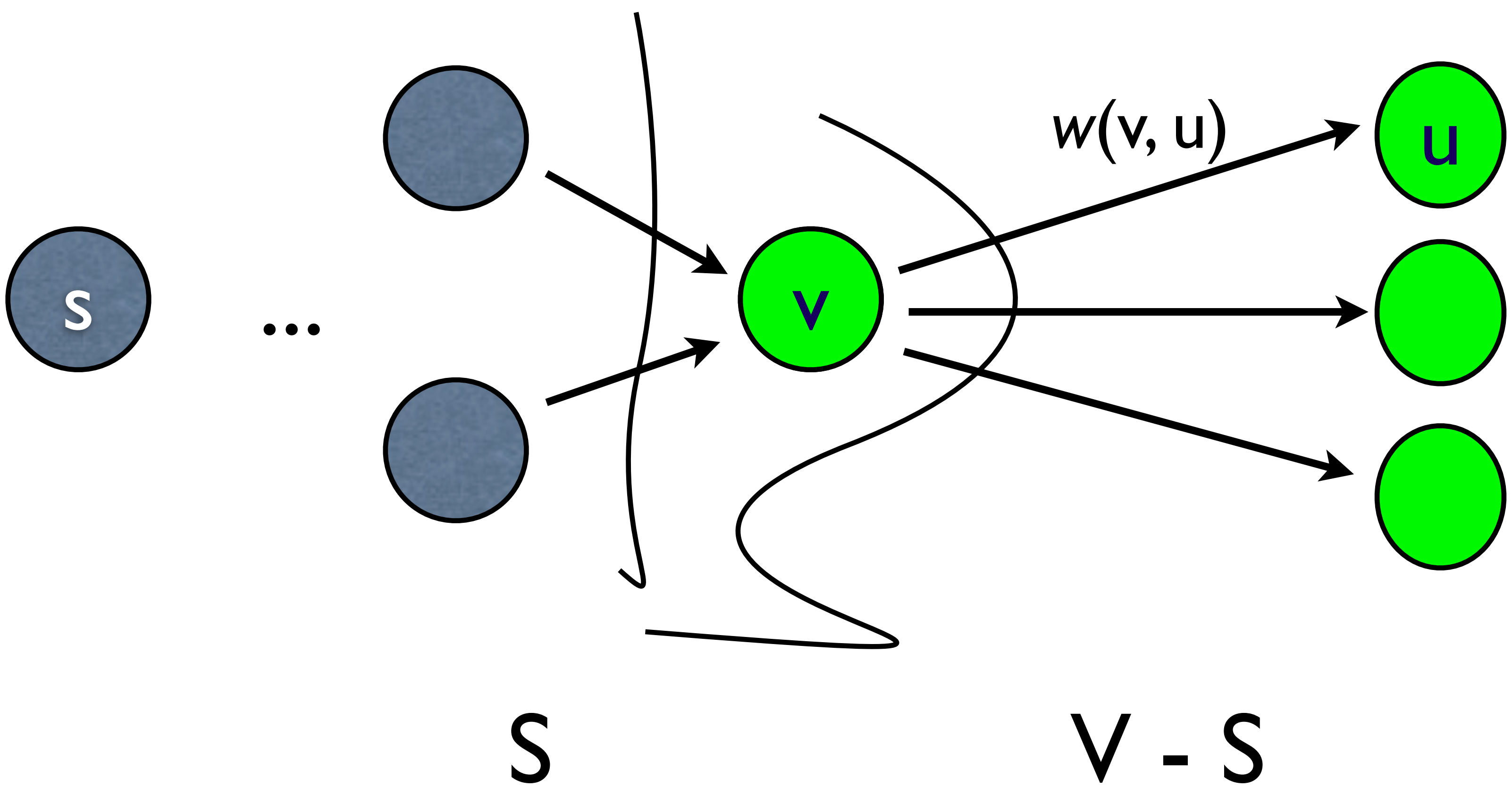


Q: how to avoid repeated checking?
maintain a counter r[e] for each e:
how many tails yet to be fixed?
fire this hyperedge only if r[e]=0

- time complexity: O(V + E)

Dijkstra Algorithm

- keep a cut (S :V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others

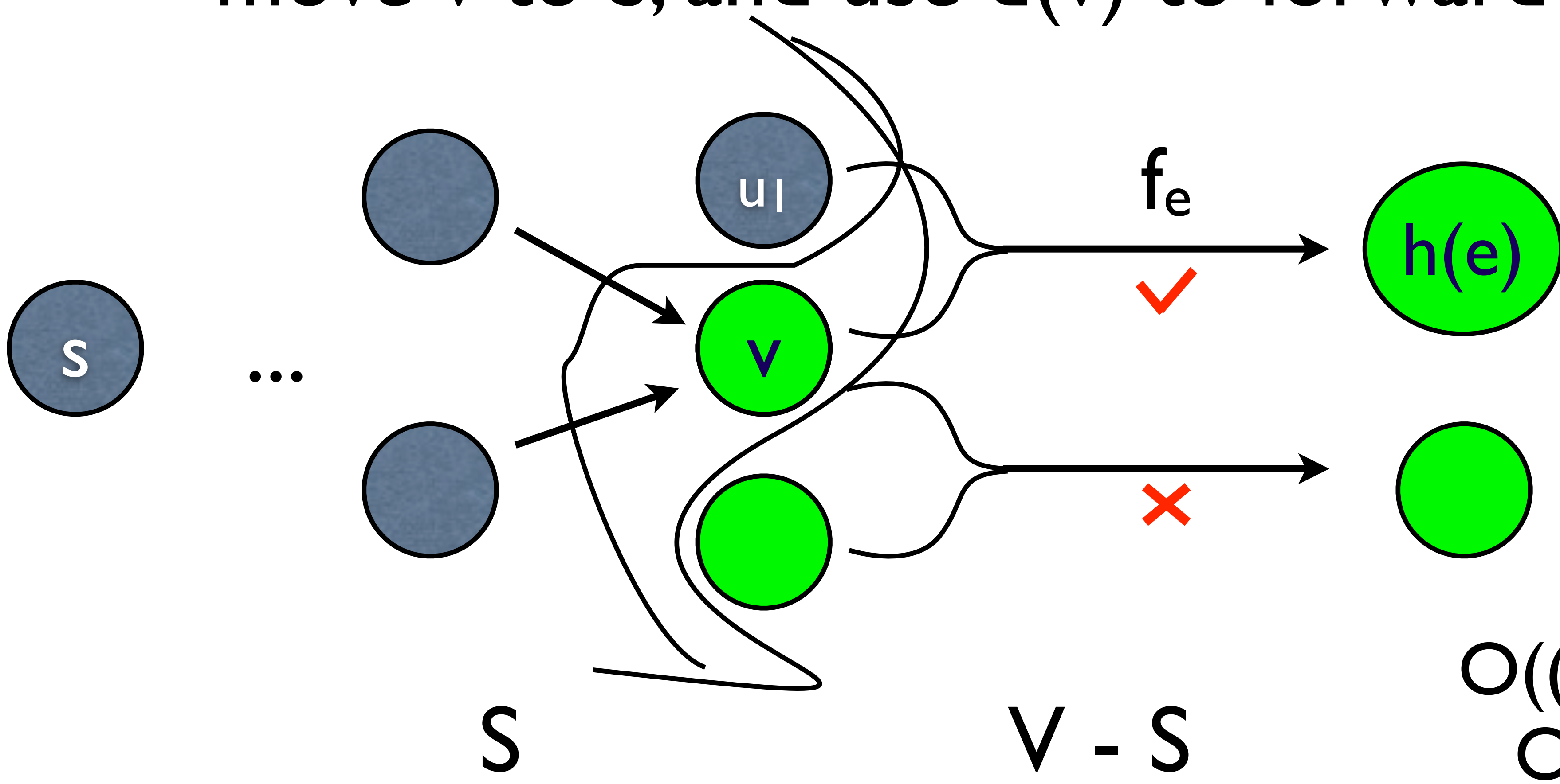


$$d(u) \oplus = d(v) \otimes w(v, u)$$

time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

Knuth (1977) Algorithm

- keep a cut (S : V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others



time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

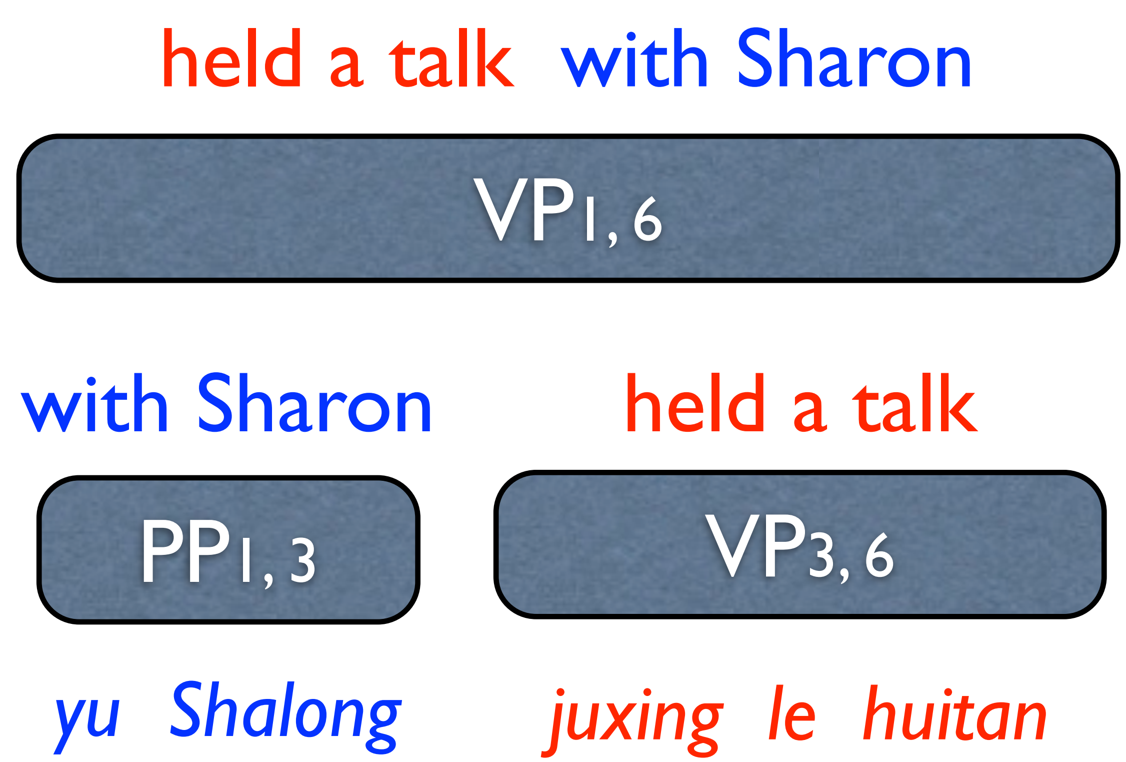
Summary of Perspectives on Parsing

- Parsing and can be viewed as:
  - search in the space of possible trees
  - (logical/probabilistic) deduction
  - intersection / composition
  - generation (from intersected grammar)
  - forest building
- Parsing algorithms introduced so far are DPs:
  - CKY: simplest, external binarization -- implement in hw5
  - intersection + Knuth 77: best-first search

Translation as Parsing

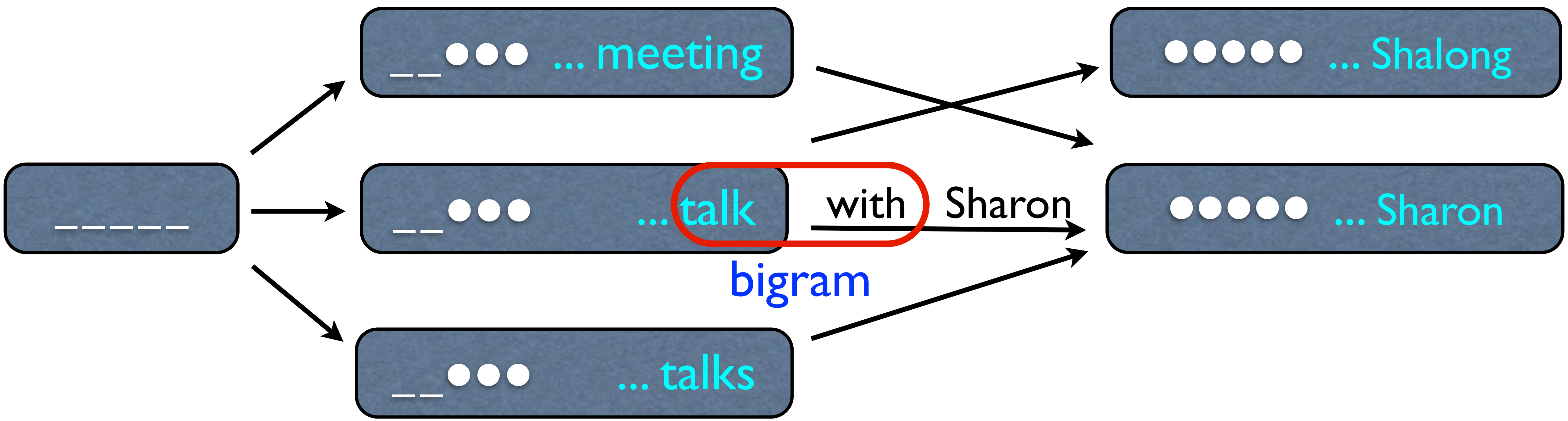
- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

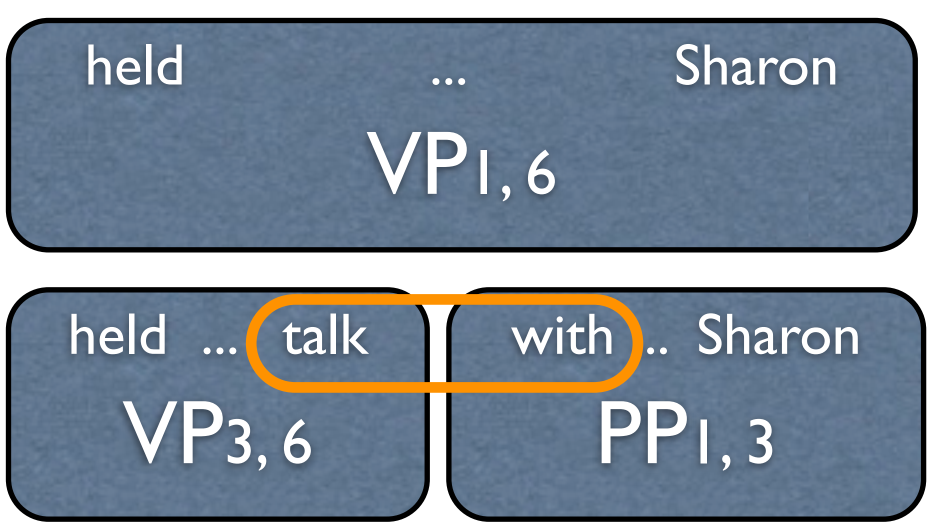
$$\begin{aligned}
\mathbf{VP} &\rightarrow \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, \\
\mathbf{VP} &\rightarrow \textit{juxing le huitan}, \\
\mathbf{PP} &\rightarrow \textit{yu Shalong},
\end{aligned}$$



complexity: same as CKY parsing -- $O(n^3)$

Adding a Bigram Model





complexity: $O(n^3 V^{4(m-1)})$