

Stochastic Context Free Grammars for RNA Structure Modeling

BMI/CS 776

www.biostat.wisc.edu/bmi776/

Spring 2017

Anthony Gitter

gitter@biostat.wisc.edu

Goals for Lecture

Key concepts

- transformational grammars
- the Chomsky hierarchy
- context free grammars
- stochastic context free grammars
- parsing ambiguity

Not covered, but here for your reference

- the Inside and Outside algorithms
- parameter learning via the Inside-Outside algorithm

Modeling RNA with Stochastic Context Free Grammars

- Consider tRNA genes
 - 274 in yeast genome, ~1500 in human genome
 - get transcribed, like protein-coding genes
 - don't get translated, therefore base statistics much different than protein-coding genes
 - but secondary structure is conserved
- To recognize new tRNA genes, model known ones using stochastic context free grammars [Eddy & Durbin, 1994; Sakakibara et al. 1994]
- But what is a grammar?

Transformational Grammars

- A transformational grammar characterizes a set of legal strings
- The grammar consists of
 - a set of abstract *nonterminal* symbols

$$\{s, c_1, c_2, c_3, c_4\}$$

- a set of *terminal* symbols (those that actually appear in strings)

$$\{A, C, G, U\}$$

- a set of *productions*

$$\begin{array}{ccccccccc} s \rightarrow c_1 & c_1 \rightarrow Uc_2 & c_2 \rightarrow Ac_3 & c_3 \rightarrow A & c_4 \rightarrow A & & & & \\ & & c_2 \rightarrow Gc_4 & c_3 \rightarrow G & & & & & \end{array}$$

A Grammar for Stop Codons

$$\begin{array}{l} s \rightarrow c_1 \quad c_1 \rightarrow Uc_2 \quad c_2 \rightarrow Ac_3 \quad c_3 \rightarrow A \quad c_4 \rightarrow A \\ c_2 \rightarrow Gc_4 \quad c_3 \rightarrow G \end{array}$$

- This grammar can generate the 3 stop codons:
UAA, UAG, UGA
- With a grammar we can ask questions like
 - what strings are derivable from the grammar?
 - can a particular string be derived from the grammar?
 - what sequence of productions can be used to derive a particular string from a given grammar?

The Derivation for UAG

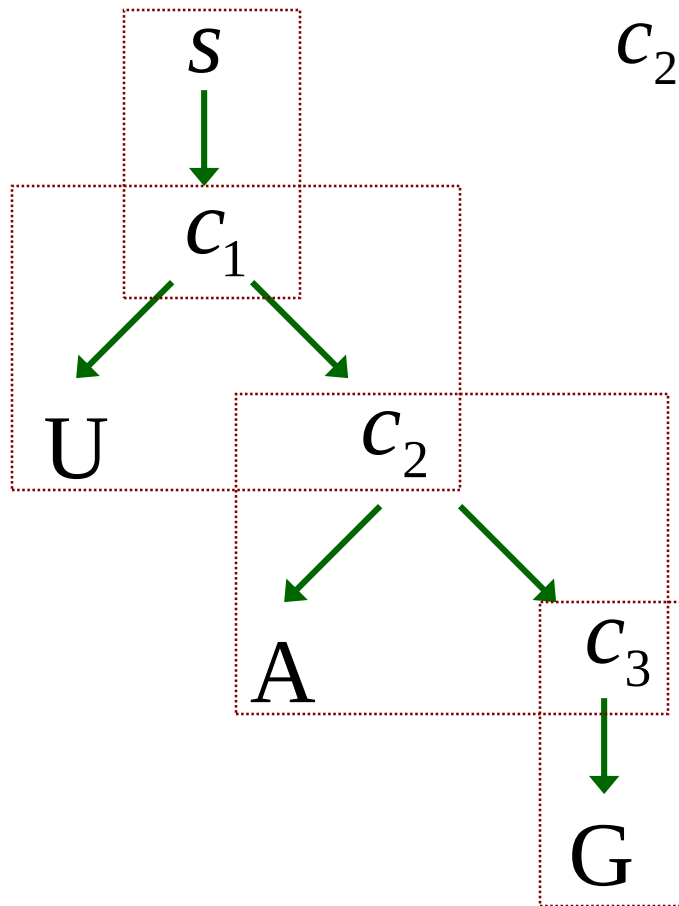
$$\begin{array}{ccccccccc} s \rightarrow c_1 & c_1 \rightarrow Uc_2 & c_2 \rightarrow Ac_3 & c_3 \rightarrow A & c_4 \rightarrow A \\ & & c_2 \rightarrow Gc_4 & c_3 \rightarrow G & & & & & \end{array}$$

$$s \Rightarrow c_1 \Rightarrow Uc_2 \Rightarrow UAc_3 \Rightarrow UAG$$

The Parse Tree for UAG

$s \rightarrow c_1$ $c_1 \rightarrow Uc_2$ $c_2 \rightarrow Ac_3$ $c_3 \rightarrow A$ $c_4 \rightarrow A$

$c_2 \rightarrow Gc_4$ $c_3 \rightarrow G$



Some Shorthand

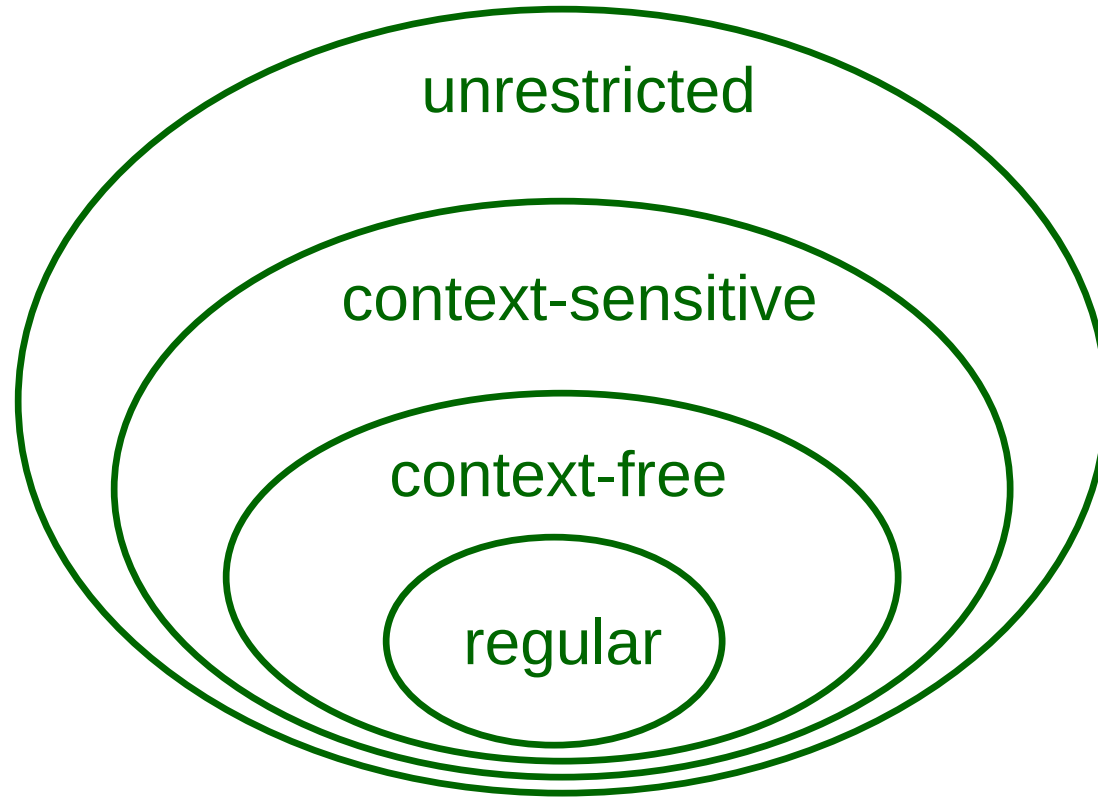
$$c_2 \rightarrow Ac_3$$

$$c_2 \rightarrow Gc_4$$



$$c_2 \rightarrow Ac_3 \mid Gc_4$$

The Chomsky Hierarchy



- A hierarchy of grammars defined by restrictions on productions

The Chomsky Hierarchy

- Regular grammars $u \rightarrow Xv$ $u \rightarrow X$
- Context-free grammars $u \rightarrow \beta$
- Context-sensitive grammars $\alpha_1 u \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$
- Unrestricted grammars $\alpha_1 u \alpha_2 \rightarrow \alpha_3$

u, v are nonterminals

X is a terminal

$\alpha_1, \alpha_2, \alpha_3$ are any sequence of terminals/nonterminals

β is any non-null sequence of terminals/nonterminals

CFGs and RNA

- Context free grammars are well suited to modeling RNA secondary structure because they can represent base pairing preferences
- A grammar for a 3-base stem with a loop of either **GAAA** or **GCAA**

$$s \rightarrow Aw_1U \mid Cw_1G \mid Gw_1C \mid Uw_1A$$

$$w_1 \rightarrow Aw_2U \mid Cw_2G \mid Gw_2C \mid Uw_2A$$

$$w_2 \rightarrow Aw_3U \mid Cw_3G \mid Gw_3C \mid Uw_3A$$

$$w_3 \rightarrow GAAA \mid GCAA$$

CFGs and RNA

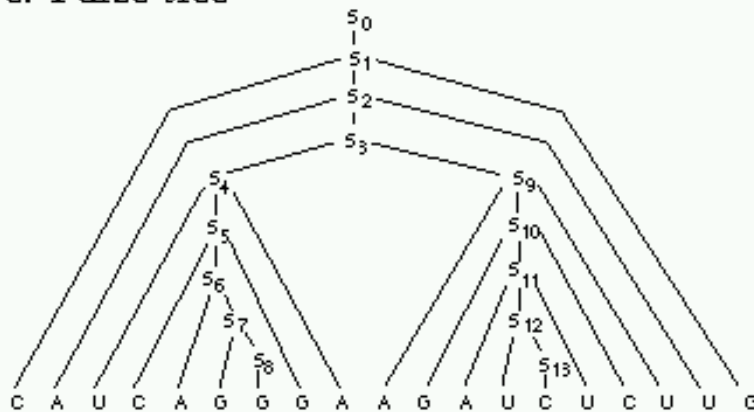
a. Productions

$$P = \left\{ \begin{array}{ll} S_0 \rightarrow S_1, & S_7 \rightarrow G S_8, \\ S_1 \rightarrow C S_2 G, & S_8 \rightarrow G, \\ S_1 \rightarrow A S_2 U, & S_8 \rightarrow U, \\ S_2 \rightarrow A S_3 U, & S_9 \rightarrow A S_{10} U, \\ S_3 \rightarrow S_4 S_9, & S_{10} \rightarrow C S_{10} G, \\ S_4 \rightarrow U S_5 A, & S_{10} \rightarrow G S_{11} C, \\ S_5 \rightarrow C S_6 G, & S_{11} \rightarrow A S_{12} U, \\ S_6 \rightarrow A S_7, & S_{12} \rightarrow U S_{13}, \\ S_7 \rightarrow U S_7, & S_{13} \rightarrow C \end{array} \right\}$$

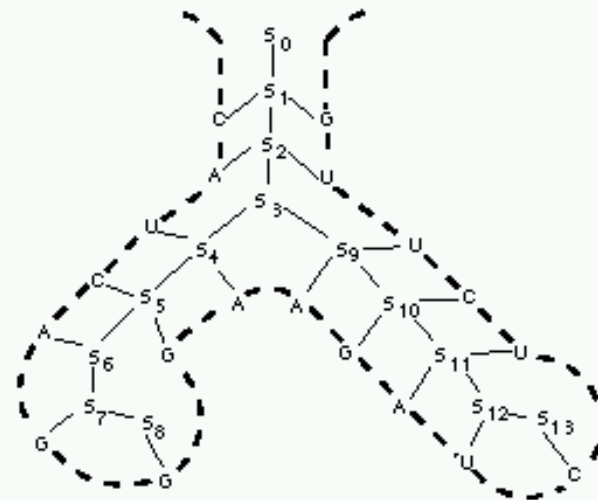
b. Derivation

$$\begin{aligned} S_0 &\Rightarrow S_1 \Rightarrow C S_2 G \Rightarrow C A S_3 U G \Rightarrow C A S_4 S_9 U G \\ &\Rightarrow C A U S_5 A S_9 U G \Rightarrow C A U C S_8 G A S_9 U G \\ &\Rightarrow C A U C A S_7 G A S_9 U G \Rightarrow C A U C A G S_8 G A S_9 U G \\ &\Rightarrow C A U C A G G G A S_9 U G \Rightarrow C A U C A G G G A A S_{10} U U G \\ &\Rightarrow C A U C A G G G A A G S_{11} C U U G \\ &\Rightarrow C A U C A G G G A A G A S_{12} U C U U G \\ &\Rightarrow C A U C A G G G A A G A U S_{13} U C U U G \\ &\Rightarrow C A U C A G G G A A G A U C U C U U G. \end{aligned}$$

c. Parse tree

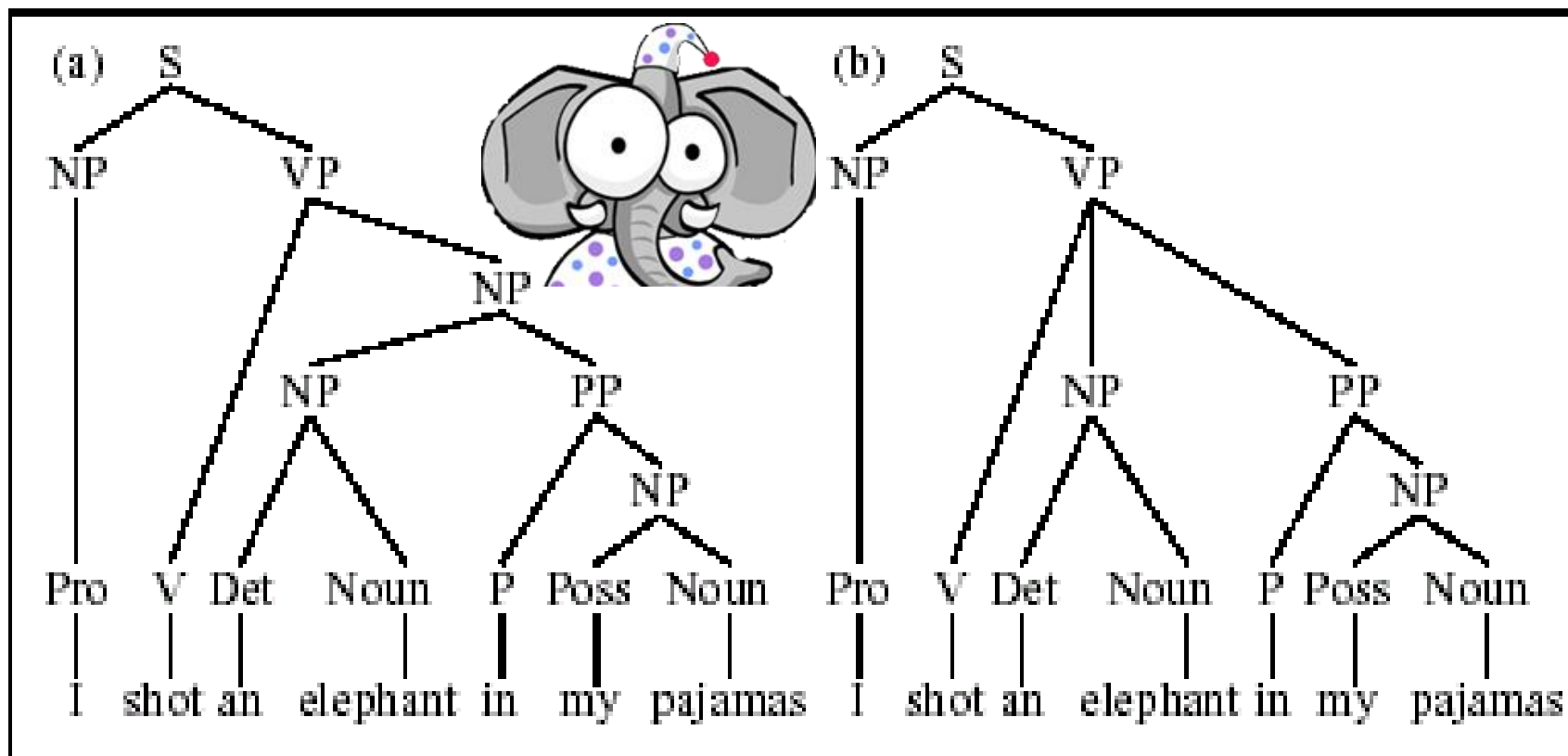


d. Secondary Structure



Ambiguity in Parsing

“I shot an elephant in my pajamas. How he got in my pajamas, I’ll never know.” – Groucho Marx



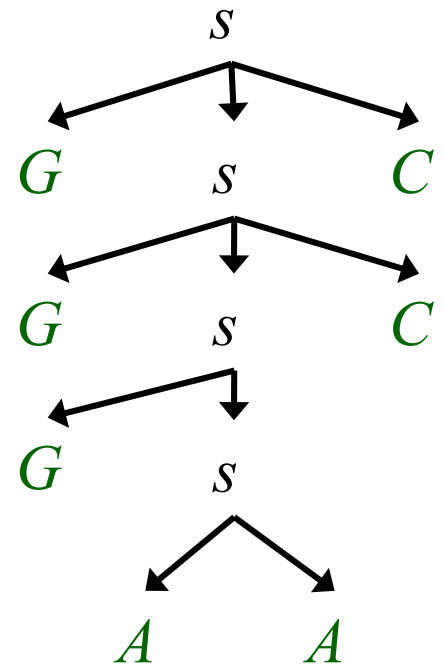
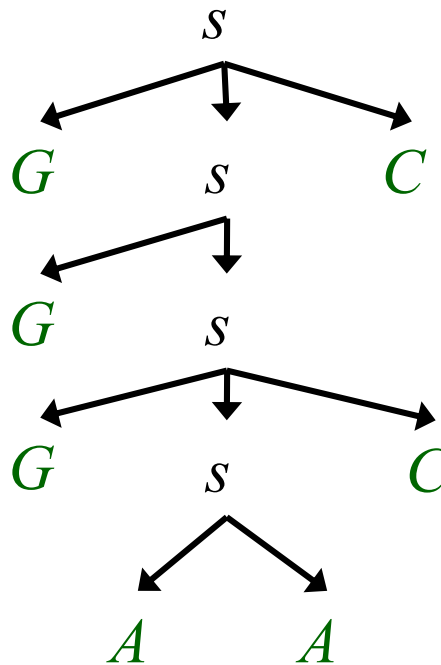
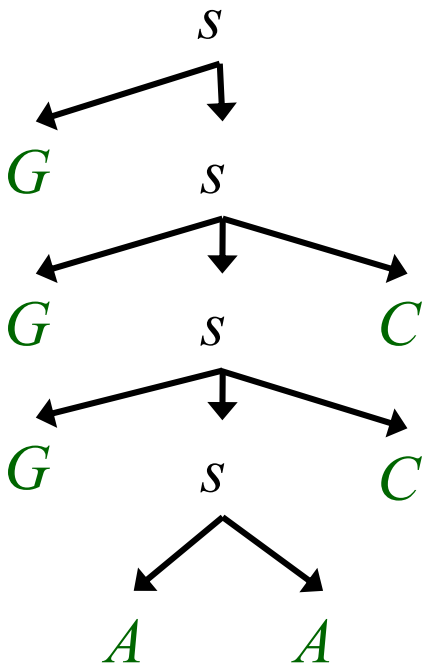
An Ambiguous RNA Grammar

$s \rightarrow G s C$

$s \rightarrow G s$

$s \rightarrow A A$

- With this grammar, there are 3 parses for the string *GGGAACC*



A Probabilistic Version of the Stop Codon Grammar

$$\begin{array}{ccccccccc} 1.0 & & 1.0 & & 0.7 & & 0.2 & & 1.0 \\ s \rightarrow c_1 & c_1 \rightarrow Uc_2 & c_2 \rightarrow Ac_3 & c_3 \rightarrow A & c_4 \rightarrow A \\ & & 0.3 & & 0.8 \\ & & c_2 \rightarrow Gc_4 & c_3 \rightarrow G \end{array}$$

- Each production has an associated probability
- Probabilities for productions with the same left-hand side sum to 1
- This *regular* grammar has a corresponding Markov chain model

Stochastic Context Free Grammars

(a.k.a. Probabilistic Context Free Grammars)

$$s \rightarrow \overset{0.25}{Aw_1U} \mid \overset{0.25}{Cw_1G} \mid \overset{0.25}{Gw_1C} \mid \overset{0.25}{Uw_1A}$$

$$w_1 \rightarrow \overset{0.1}{Aw_2U} \mid \overset{0.4}{Cw_2G} \mid \overset{0.4}{Gw_2C} \mid \overset{0.1}{Uw_2A}$$

$$w_2 \rightarrow \overset{0.25}{Aw_3U} \mid \overset{0.25}{Cw_3G} \mid \overset{0.25}{Gw_3C} \mid \overset{0.25}{Uw_3A}$$

$$w_3 \rightarrow \overset{0.8}{GAAA} \mid \overset{0.2}{GCAA}$$

Stochastic Grammars?

...the notion “probability of a sentence” is an entirely useless one, under any known interpretation of this term.

— Noam Chomsky
(famed linguist)

Every time I fire a linguist, the performance of the recognizer improves.

— Fred Jelinek
(former head of IBM speech recognition group)

Credit for pairing these quotes goes to Dan Jurafsky and James Martin,
Speech and Language Processing

Three Key Questions

- How likely is a given sequence?
the Inside algorithm
- What is the most probable parse for a given sequence?
the Cocke-Younger-Kasami (CYK) algorithm
- How can we learn the SCFG parameters given a grammar and a set of sequences?
the Inside-Outside algorithm

Chomsky Normal Form

- It is convenient to assume that our grammar is in *Chomsky Normal Form*; i.e. all productions are of the form:

$V \rightarrow yz$ right hand side consists of two nonterminals

$V \rightarrow A$ right hand side consists of a single terminal

- Any CFG can be put into Chomsky Normal Form

Converting a Grammar to CNF

$$s \rightarrow G s C$$

$$s \rightarrow G s$$

$$s \rightarrow A A$$



$$s \rightarrow b_G p$$

$$p \rightarrow s b_C$$

$$s \rightarrow b_G s$$

$$s \rightarrow b_A b_A$$

$$b_G \rightarrow G$$

$$b_C \rightarrow C$$

$$b_A \rightarrow A$$

Parameter Notation

- For productions of the form $v \rightarrow yz$, we'll denote the associated probability parameters

$$t_v(y, z) \quad \text{transition}$$

- For productions of the form $v \rightarrow A$, we'll denote the associated probability parameters

$$e_v(A) \quad \text{emission}$$

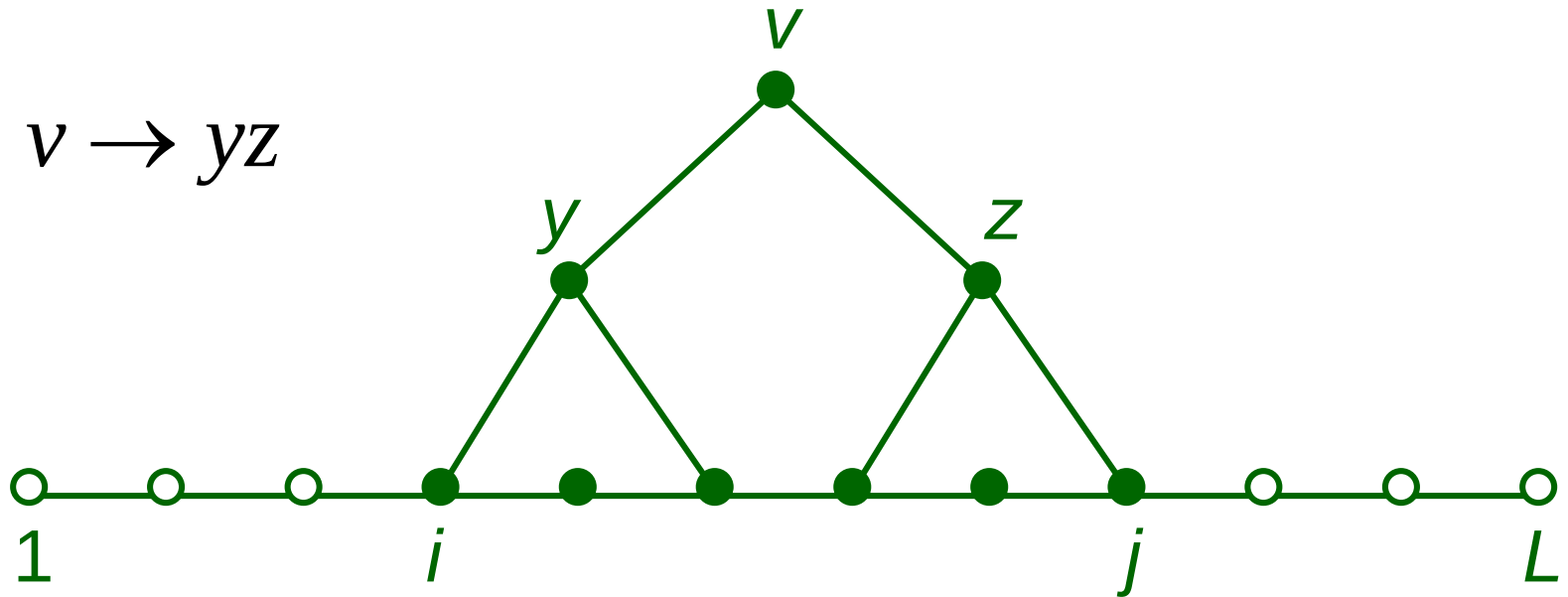
Determining the Likelihood of a Sequence: The Inside Algorithm

- Dynamic programming method, analogous to the Forward algorithm
- Involves filling in a 3D matrix

$$\alpha(i, j, v)$$

representing the probability of all parse subtrees rooted at nonterminal v for the subsequence from i to j

Determining the Likelihood of a Sequence: The Inside Algorithm



- $\alpha(i, j, v)$: the probability of all parse subtrees rooted at nonterminal v for the subsequence from i to j

Inside Calculation Example

$s \rightarrow b_G p$

$p \rightarrow s b_C$

$s \rightarrow b_G s$

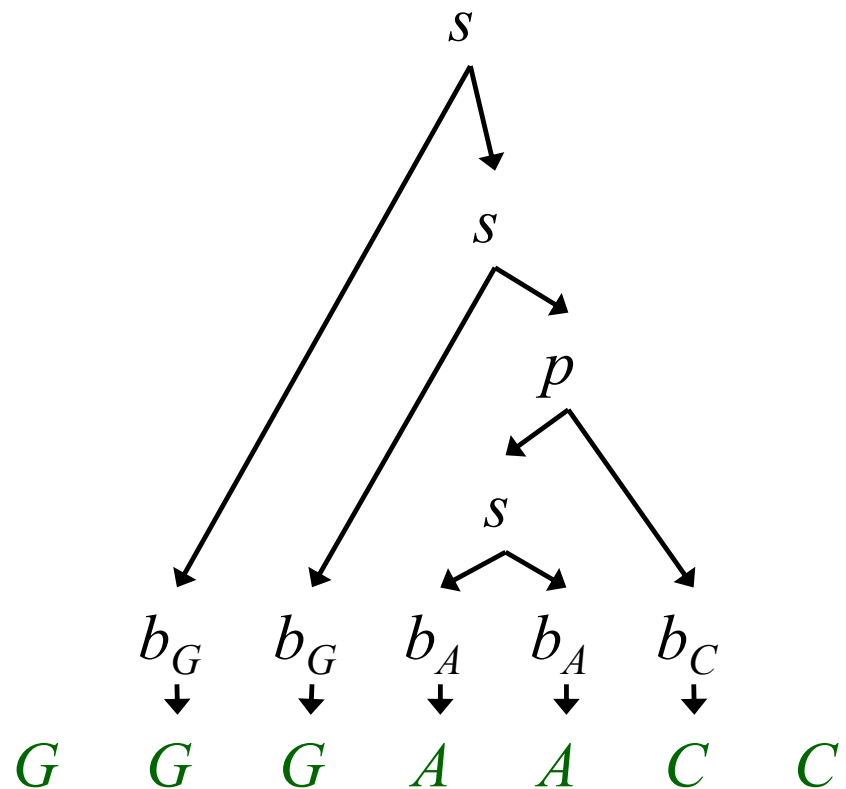
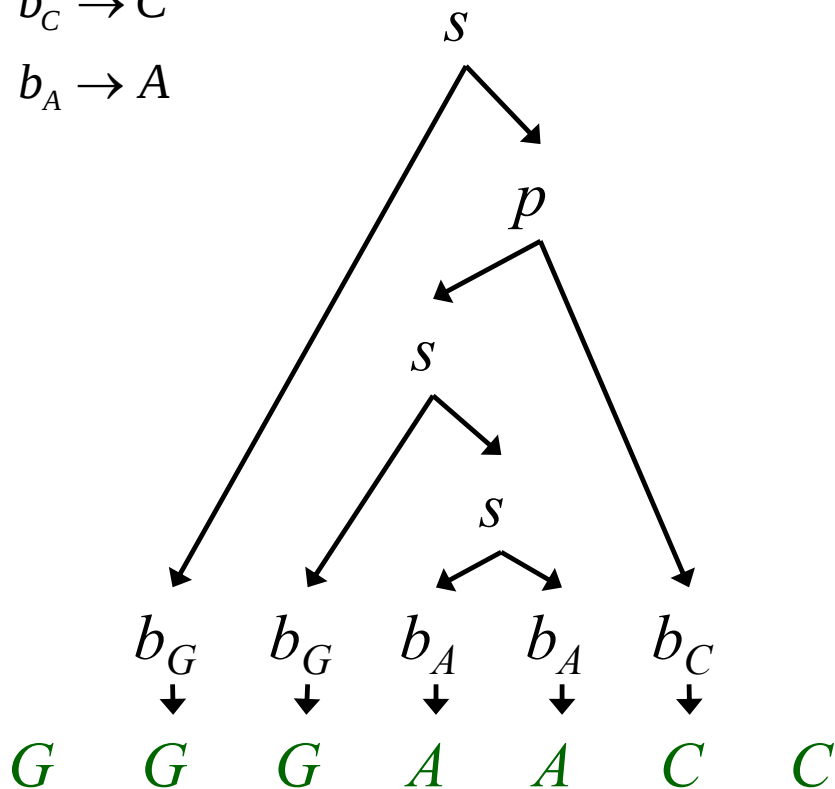
$s \rightarrow b_A b_A$

$b_G \rightarrow G$

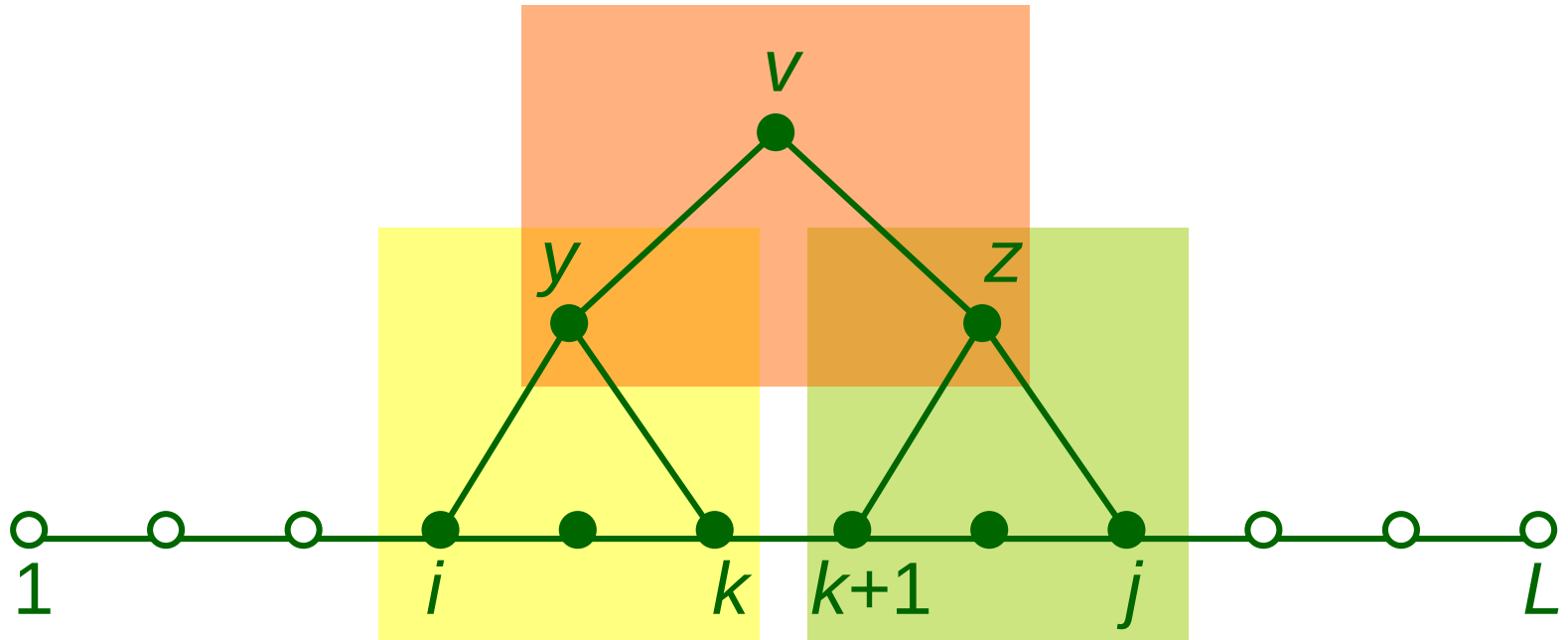
$b_C \rightarrow C$

$b_A \rightarrow A$

$$\alpha(2,6,s) = t_s(b_G, p) \alpha(2,2,b_G) \alpha(3,6,p) + t_s(b_G, s) \alpha(2,2,b_G) \alpha(3,6,s)$$



Determining the Likelihood of a Sequence: The Inside Algorithm



$$\alpha(i, j, v) = \sum_{y=1}^M \sum_{z=1}^M \sum_{k=i}^{j-1} t_v(y, z) \alpha(i, k, y) \alpha(k+1, j, z)$$

M is the number of nonterminals in the grammar

The Inside Algorithm

- Initialization (for $i = 1$ to L , $v = 1$ to M)

$$\alpha(i, i, v) = e_v(x_i)$$

- Iteration (for $i = L-1$ to 1 , $j = i+1$ to L , $v = 1$ to M)

$$\alpha(i, j, v) = \sum_{y=1}^M \sum_{z=1}^M \sum_{k=i}^{j-1} t_v(y, z) \alpha(i, k, y) \alpha(k+1, j, z)$$

- Termination

$$\Pr(x) = \alpha(1, L, 1)$$

↑
start nonterminal

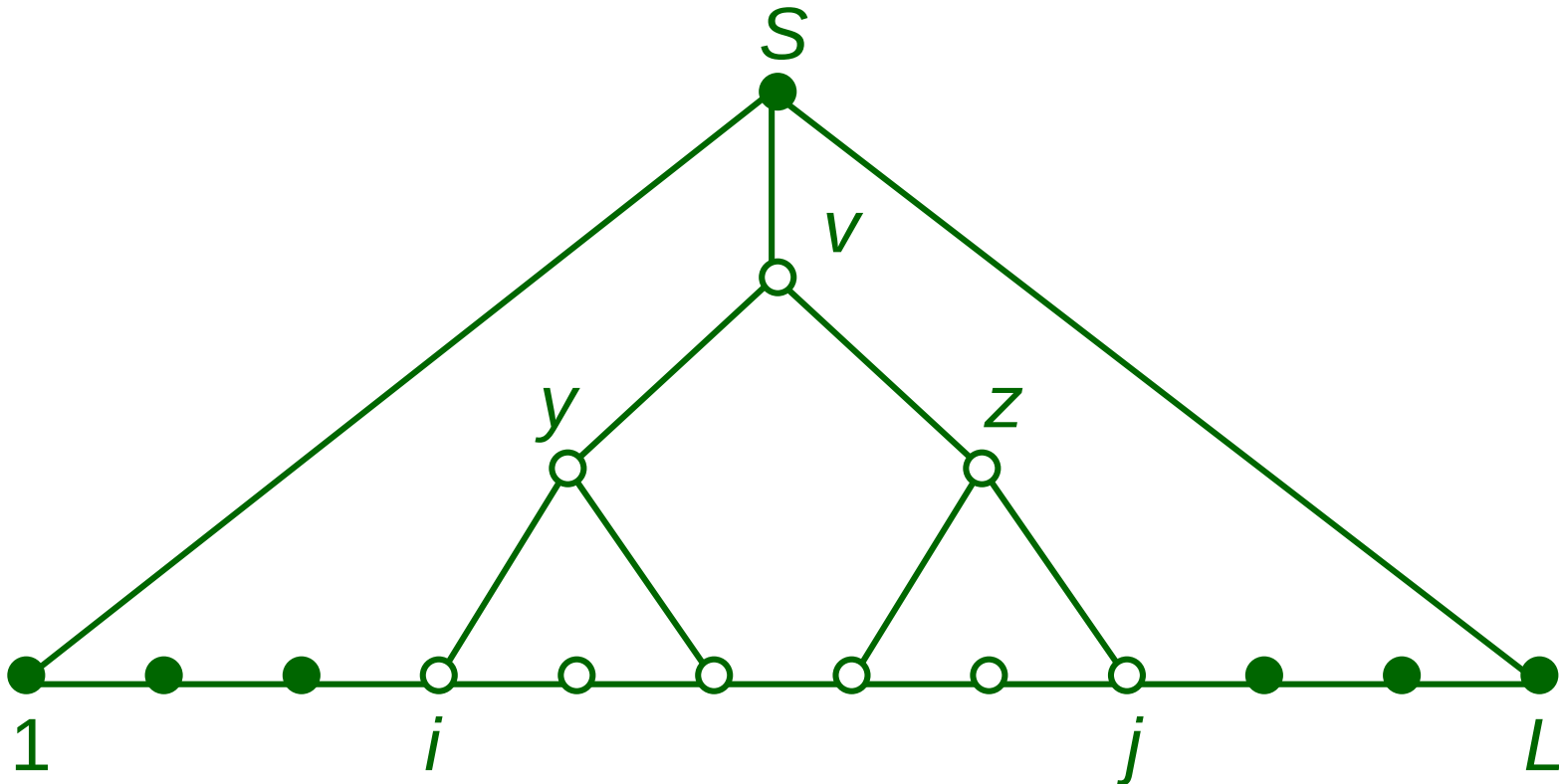
Learning SCFG Parameters

- If we know the parse tree for each training sequence, learning the SCFG parameters is simple
 - no hidden part of the problem during training
 - count how often each parameter (i.e. production) is used
 - normalize/smooth to get probabilities
- More commonly, there are many possible parse trees per sequence – we don't know which one is correct
 - thus, use an EM approach (Inside-Outside)
 - iteratively
 - determine *expected* # times each production is used
 - consider all parses
 - weight each by its probability
 - set parameters to maximize likelihood given these counts

The Inside-Outside Algorithm

- We can learn the parameters of an SCFG from training sequences using an EM approach called Inside-Outside
- In the E-step, we determine
 - the expected number of times each *nonterminal* is used in parses $c(v)$
 - the expected number of times each *production* is used in parses
$$c(v \rightarrow yz)$$
$$c(v \rightarrow A)$$
- In the M-step, we update our production probabilities

The Outside Algorithm



- $\beta(i, j, v)$: the probability of parse trees rooted at the start nonterminal, excluding the probability of all subtrees rooted at nonterminal v covering the subsequence from i to j

Outside Calculation Example

$s \rightarrow b_G p$

$p \rightarrow s b_C$

$s \rightarrow b_G s$

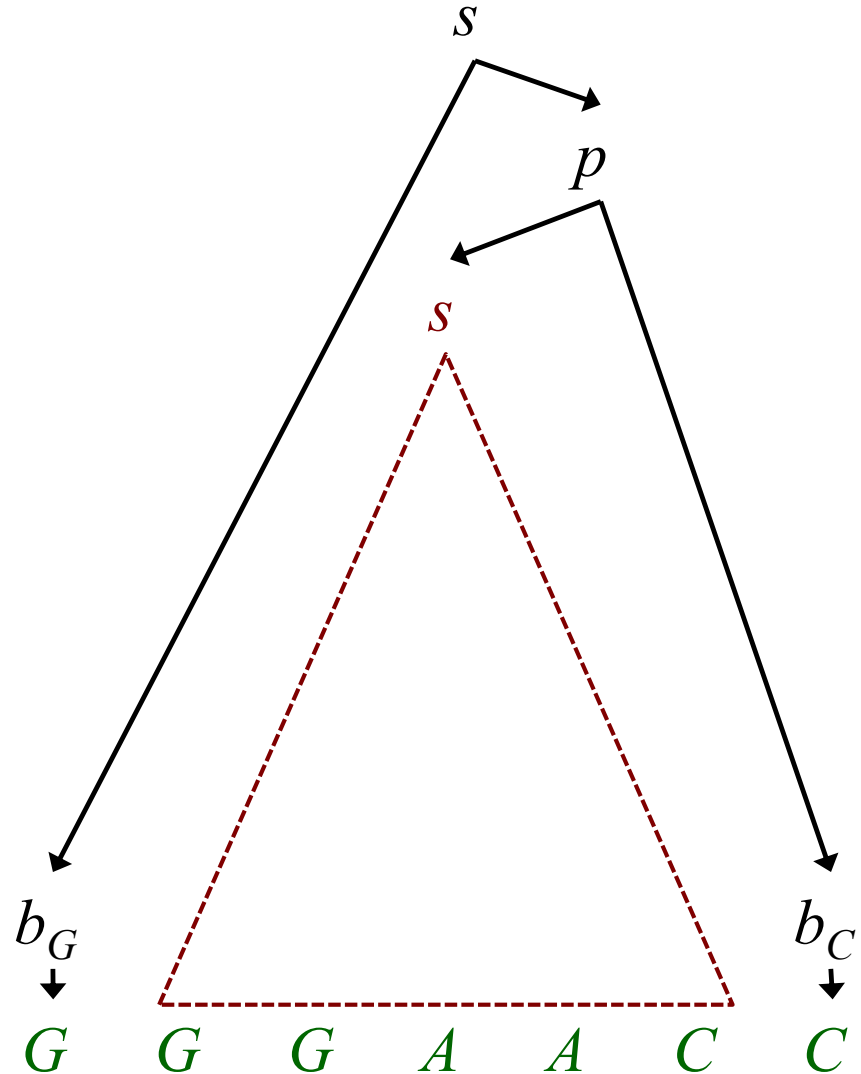
$s \rightarrow b_A b_A$

$b_G \rightarrow G$

$b_C \rightarrow C$

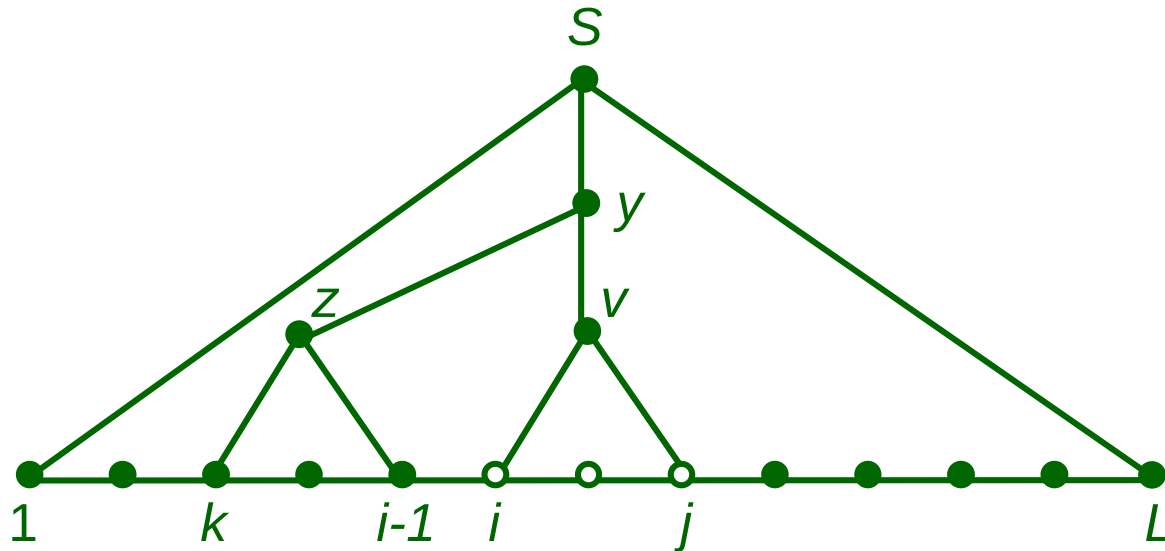
$b_A \rightarrow A$

$$\beta(2, 6, s) = t_p(s, b_C) \alpha(7, 7, b_C) \beta(2, 7, p)$$



The Outside Algorithm

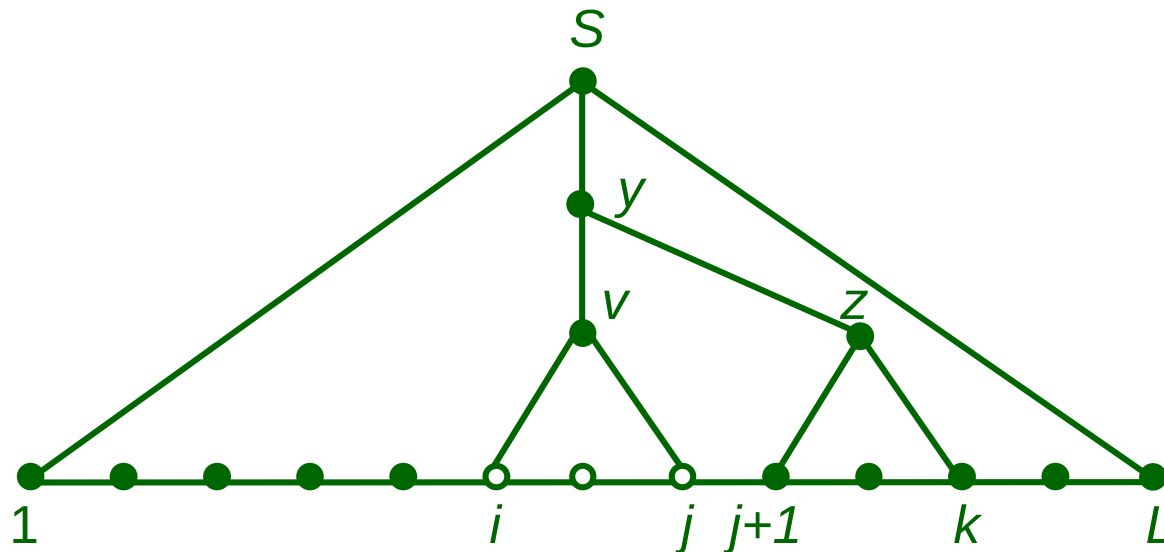
- We can recursively calculate $\beta(i, j, v)$ from β values we've calculated for y
- The first case we consider is where v is used in productions of the form: $y \rightarrow zv$



$$\sum_{y=1}^M \sum_{z=1}^M \sum_{k=1}^{i-1} t_y(z, v) \alpha(k, i-1, z) \beta(k, j, y)$$

The Outside Algorithm

- The second case we consider is where v is used in productions of the form: $y \rightarrow vZ$



$$\sum_{y=1}^M \sum_{z=1}^M \sum_{k=j+1}^L t_y(v, z) \alpha(j+1, k, z) \beta(i, k, y)$$

The Outside Algorithm

- Initialization

$$\beta(1, L, 1) = 1 \quad (\text{the } \textit{start} \text{ nonterminal})$$

$$\beta(1, L, v) = 0 \quad \text{for } v = 2 \text{ to } M$$

- Iteration (for $i = 1$ to L , $j = L$ to i , $v = 1$ to M)

$$\begin{aligned} \beta(i, j, v) = & \sum_{y=1}^M \sum_{z=1}^M \sum_{k=1}^{i-1} t_y(z, v) \alpha(k, i-1, z) \beta(k, j, y) + \\ & \sum_{y=1}^M \sum_{z=1}^M \sum_{k=j+1}^L t_y(v, z) \alpha(j+1, k, z) \beta(i, k, y) \end{aligned}$$

The Inside-Outside Algorithm

- We can learn the parameters of an SCFG from training sequences using an EM approach called Inside-Outside
- In the E-step, we determine
 - the expected number of times each *nonterminal* is used in parses $c(v)$
 - the expected number of times each *production* is used in parses
$$c(v \rightarrow yz)$$
$$c(v \rightarrow A)$$
- In the M-step, we update our production probabilities

The Inside-Outside Algorithm

- The EM re-estimation equations (for 1 sequence) are:

$$\hat{e}_v(A) = \frac{c(v \rightarrow A)}{c(v)} = \frac{\sum_{i | x_i = A} \beta(i, i, v) e_v(A)}{\sum_{i=1}^L \sum_{j=i}^L \beta(i, j, v) \alpha(i, j, v)}$$

← cases where v used to generate A

$$\hat{t}_v(y, z) = \frac{c(v \rightarrow yz)}{c(v)}$$

← cases where v used to generate any subsequence

$$= \frac{\sum_{i=1}^{L-1} \sum_{j=i+1}^L \sum_{k=i}^{j-1} t_v(y, z) \beta(i, j, v) \alpha(i, k, y) \alpha(k+1, j, z)}{\sum_{i=1}^L \sum_{j=i}^L \beta(i, j, v) \alpha(i, j, v)}$$

Finding the Most Likely Parse: The CYK Algorithm

- Involves filling in a 3D matrix

$$\gamma(i, j, v)$$

representing the most probable parse subtree rooted at nonterminal v for the subsequence from i to j

- and a matrix for the traceback

$$\tau(i, j, v)$$

storing information about the production at the top of this parse subtree

The CYK Algorithm

- Initialization (for $i = 1$ to L , $v = 1$ to M)

$$\gamma(i, i, v) = \log e_v(x_i)$$

$$\tau(i, i, v) = (0, 0, 0)$$

- Iteration (for $i = 1$ to $L - 1$, $j = i + 1$ to L , $v = 1$ to M)

$$\gamma(i, j, v) = \max_{\substack{y, z \\ k=i \dots j-1}} \{ \gamma(i, k, y) + \gamma(k + 1, j, z) + \log t_v(y, z) \}$$

$$\tau(i, j, v) = \arg \max_{\substack{y, z \\ k=i \dots j-1}} \{ \gamma(i, k, y) + \gamma(k + 1, j, z) + \log t_v(y, z) \}$$

- Termination

$$\log P(x, \hat{\pi} \mid \theta) = \gamma(1, L, 1)$$

↑
start nonterminal

The CYK Algorithm Traceback

- Initialization:

push $(1, L, 1)$ on the stack

- Iteration:

```
pop  $(i, j, v)$            // pop subsequence/nonterminal pair
 $(y, z, k) = \tau(i, j, v)$  // get best production identified by CYK
if  $(y, z, k) == (0, 0, 0)$  // indicating a leaf
    attach  $x_i$  as the child of  $v$ 
else
    attach  $y, z$  to parse tree as children of  $v$ 
    push( $i, k, y$ )
    push( $k+1, j, z$ )
```

Comparison of SCFG Algorithms to HMM Algorithms

	HMM algorithm	SCFG algorithm
optimal alignment	Viterbi	CYK
probability of sequence	forward	inside
EM parameter estimation	forward-backward	inside-outside
memory complexity	$O(LM)$	$O(L^2M)$
time complexity	$O(LM^2)$	$O(L^3M^3)$