

# Stochastic Context Free Grammars

for noncoding RNA gene prediction

CBB 261

*B. Majoros*  
Duke  
UNIVERSITY

# Formal Languages

A *formal language* is simply a set of strings (i.e., sequences). That set may be infinite.

Let  $M$  be a model denoting a language. If  $M$  is a *generative model* such as an HMM or a grammar, then  $L(M)$  denotes the language generated by  $M$ .

If  $M$  is an *acceptor model* such as a finite automaton, then  $L(M)$  denotes the language accepted by  $M$ .

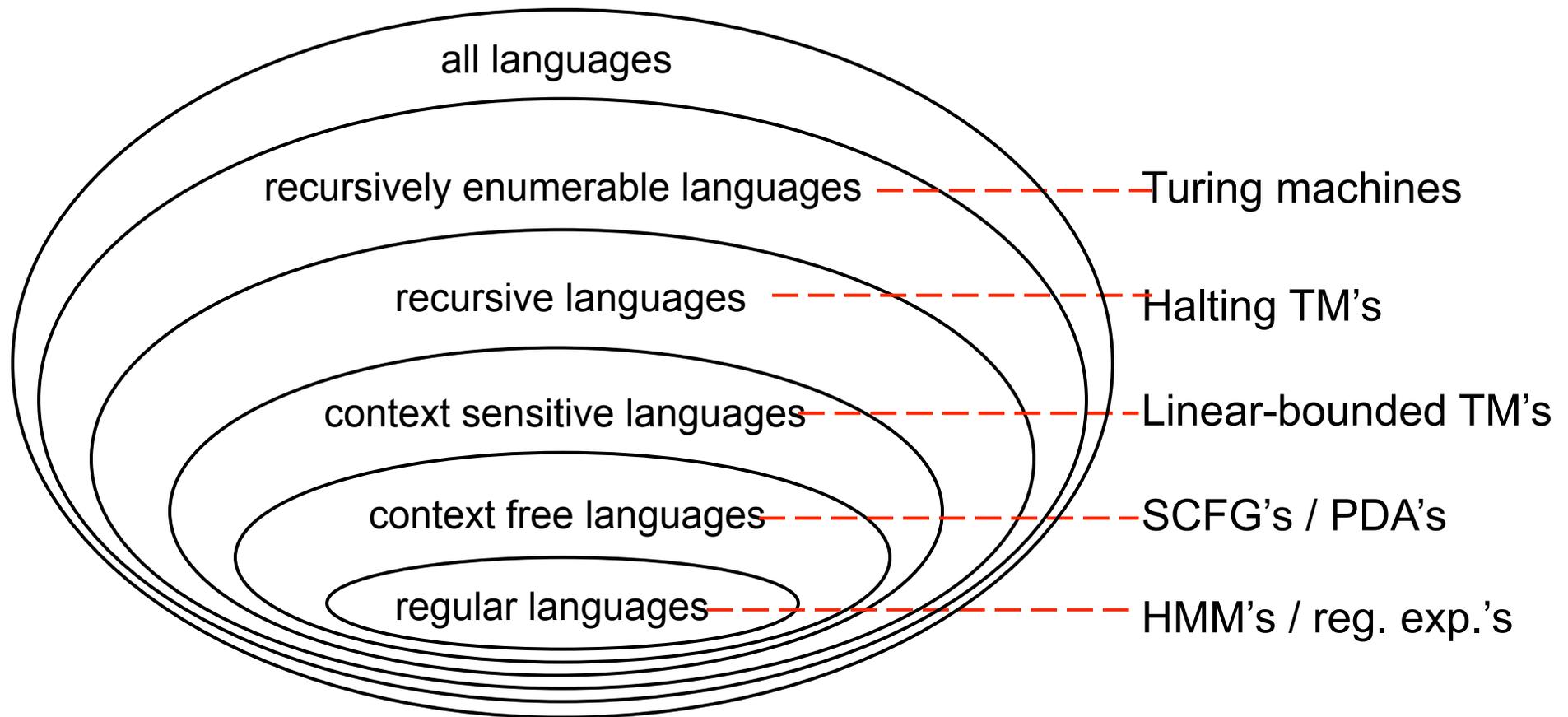
When all the parameters of a stochastic generative model are known, we can ask:

*“What is the probability that model  $M$  will generate string  $S$ ?”*

which we denote:

$$P(S | M)$$

# Recall: The Chomsky Hierarchy



*\* each class is a subset of the next higher class in the hierarchy*

## *Examples:*

- \* HMM-based gene-finders assume DNA is **regular***
- \* secondary structure prediction assumes RNA is **context-free***
- \* RNA pseudoknots are **context-sensitive***

# Context-free Grammars (CFG's)

A *context-free grammar* is a generative model denoted by a 4-tuple:

$$G = (V, \alpha, S, R)$$

where:

$\alpha$  is a *terminal alphabet*, (e.g.,  $\{a, c, g, t\}$  )

$V$  is a *nonterminal alphabet*, (e.g.,  $\{A, B, C, D, E, \dots\}$  )

$S \in V$  is a special *start symbol*, and

$R$  is a set of rewriting rules called *productions*.

Productions in  $R$  are rules of the form:

$$X \rightarrow \lambda$$

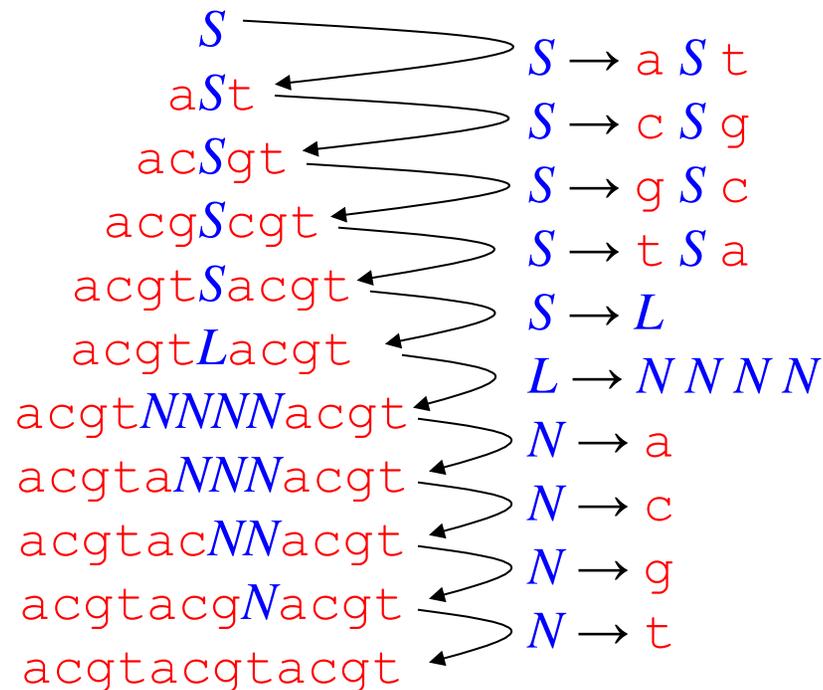
for  $X \in V$ ,  $\lambda \in (V \cup \alpha)^*$ ; such a production denotes that the nonterminal symbol  $X$  may be *rewritten* by the expression  $\lambda$ , which may consist of zero or more terminals and nonterminals.

# A Simple Example

As an example, consider  $\mathcal{G}=(V_{\mathcal{G}}, \alpha, S, R_{\mathcal{G}})$ , for  $V_{\mathcal{G}}=\{S, L, N\}$ ,  $\alpha=\{a, c, g, t\}$ , and  $R_{\mathcal{G}}$  the set consisting of:

$S \rightarrow a S t$	$S \rightarrow L$
$S \rightarrow t S a$	$L \rightarrow N N N N$
$S \rightarrow c S g$	$N \rightarrow a   c   g   t$
$S \rightarrow g S c$	

One possible *derivation* using this grammar is:



# Derivations

Suppose a CFG  $G$  has generated a *terminal string*  $x \in \alpha^*$ . A *derivation* denotes a single way by which  $G$  may have generated  $x$ . For a grammar  $G$  and a string  $x$ , there may exist multiple distinct derivations.

A *derivation* (or *parse*) consists of a series of applications of productions from  $R$ , beginning with the *start symbol*  $S$  and ending with a *terminal string*  $x$ :

$$S \Rightarrow s_1 \Rightarrow s_2 \Rightarrow s_3 \Rightarrow \dots \Rightarrow x$$

We can denote this more compactly as:  $S \Rightarrow^* x$ . Each string  $s_i$  in a derivation is called a *sentential form*, and may consist of both terminal and nonterminal symbols:  $s_i \in (V \cup \alpha)^*$ . Each step in a derivation must be of the form:

$$wXz \Rightarrow w\lambda z$$

for  $w, z \in (V \cup \alpha)^*$ , where  $X \rightarrow \lambda$  is a production in  $R$ ; note that  $w$  and  $z$  may be empty ( $\varepsilon$  denotes the empty string).

# Leftmost Derivations

A *leftmost derivation* is one in which at each step the leftmost nonterminal in the current sentential form is the one which is rewritten:

$$S \Rightarrow \dots \Rightarrow abXdYZ \Rightarrow abxxxdYZ \Rightarrow abxxdyyyZ \Rightarrow abxxxzydyyyzzz$$

For many applications, it is not necessary to restrict one's attention to only the leftmost derivations. In that case, there may exist multiple derivations which can produce the same exact string.

However, when we get to *stochastic CFG*'s, it will be convenient to assume that only leftmost derivations are valid. This will simplify probability computations, since we don't have to model the process of stochastically choosing a nonterminal to rewrite. Note that doing this does not reduce the representational power of the CFG in any way; it just makes it easier to work with.

# Context-freeness

The “*context-freeness*” of context-free grammars is imposed by the requirement that the l.h.s of each production rule may contain only a **single** symbol, and that symbol must be a **nonterminal**:

$$X \rightarrow \lambda$$

for  $X \in V$ ,  $\lambda \in (V \cup \alpha)^*$ . That is,  $X$  is a nonterminal and  $\lambda$  is any (possibly empty) string of terminals and/or nonterminals. Thus, a CFG cannot specify *context-sensitive* rules such as:

$$wXz \rightarrow w\lambda z$$

which states that nonterminal  $X$  can be rewritten by  $\lambda$  only when  $X$  occurs in the local context  $wXz$  in a sentential form. Such productions are possible in *context-sensitive grammars* (*CSG*'s).

# Context-free Versus Regular

The advantage of CFG's over HMM's lies in their ability to model arbitrary runs of matching pairs of “*palindromic*” elements, such as nested pairs of parentheses:

$$\dots(((((((\dots)))))))))\dots$$

where each opening parenthesis must have exactly one matching closing parenthesis on the right. When the number of nested pairs is unbounded (i.e., a matching close parenthesis can be arbitrarily far away from its open parenthesis), a finite-state model such as a DFA or an HMM is inadequate to enforce the constraint that all left elements must have a matching right element.

In contrast, the modeling of nested pairs of elements can be readily achieved in a CFG using rules such as  $X \rightarrow (X)$ . A sample derivation using such a rule is:

$$X \Rightarrow (X) \Rightarrow ((X)) \Rightarrow (((X))) \Rightarrow ((((X))))$$

An additional rule such as  $X \rightarrow \epsilon$  is necessary to terminate the recursion.

# Limitations of CFG's

One thing that CFG's can't model is the matching of arbitrary runs of matching elements in the *same direction* (i.e., *not* palindromic):

.....abcdefg.....abcdefg.....

In other words, languages of the form:

$wxw$

*Q: why isn't this very relevant to RNA structure prediction?*

*Hint: think of the directionality of paired strands.*

for strings  $w$  and  $x$  of arbitrary length, cannot be modeled using a CFG.

More relevant to ncRNA prediction is the case of *pseudoknots*, which also cannot be recognized using standard CFG's:

.....abcde.....rstuv.....edcba.....vutsr.....

The problem is that the matching palindromes (and the regions separating them) are of *arbitrary length*.

# Stochastic CFG's (SCFG's)

A *stochastic context-free grammar* (SCFG) is a CFG plus a probability distribution on productions:

$$G = (V, \alpha, S, R, P_p)$$

where  $P_p : R \mapsto \mathbb{R}$ , and probabilities are normalized at the level of each l.h.s. symbol  $X$ :

$$\forall [ \sum_{X \in V} P_p(X \rightarrow \lambda) = 1 ]$$

Thus, we can compute the probability of a single derivation  $S \Rightarrow^* x$  by multiplying the probabilities for all productions used in the derivation:

$$\prod_i P(X_i \rightarrow \lambda_i)$$

We can sum over all possible (leftmost) derivations of a given string  $x$  to get the probability that  $G$  will generate  $x$  at random:

$$P(x | G) = \sum_j P(S \Rightarrow_j^* x | G).$$

# A Simple Example

As an example, consider  $\mathcal{G}=(V_{\mathcal{G}}, \alpha, S, R_{\mathcal{G}}, P_{\mathcal{G}})$ , for  $V_{\mathcal{G}}=\{S, L, N\}$ ,  $\alpha=\{a, c, g, t\}$ , and  $R_{\mathcal{G}}$  the set consisting of:

$$S \rightarrow a S t \mid t S a \mid c S g \mid g S c \mid L \quad (P=0.2)$$

$$L \rightarrow N N N N \quad (P=1.0)$$

$$N \rightarrow a \mid c \mid g \mid t \quad (P=0.25)$$

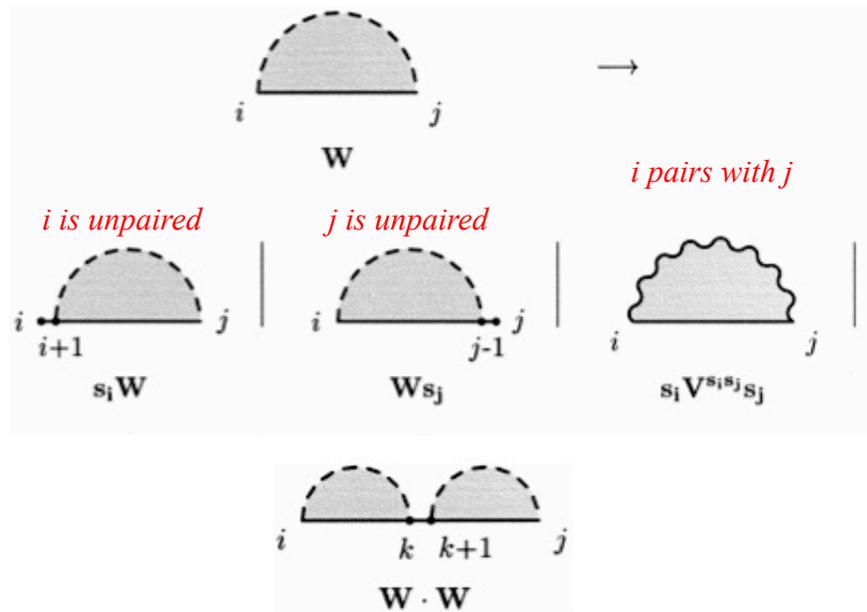
where  $\forall_{\lambda} P_{\mathcal{G}}(S \rightarrow \lambda)=0.2$ ,  $P_{\mathcal{G}}(L \rightarrow N N N N)=1$ , and  $\forall_{\lambda} P_{\mathcal{G}}(N \rightarrow \lambda)=0.25$ . Then the probability of the sequence  $acgtacgtacgt$  is given by:

$$\begin{aligned} P(acgtacgtacgt) = \\ P(S \Rightarrow aSt \Rightarrow acSgt \Rightarrow acgScgt \Rightarrow acgtSacgt \Rightarrow \\ acgtLacgt \Rightarrow acgtNNNNacgt \Rightarrow acgtaNNNacgt \Rightarrow \\ acgtacNNacgt \Rightarrow acgtacgNacgt \Rightarrow acgtacgtacgt) = \\ 0.2 \times 0.2 \times 0.2 \times 0.2 \times 0.2 \times 1 \times 0.25 \times 0.25 \times 0.25 \times 0.25 = 1.25 \times 10^{-6} \end{aligned}$$

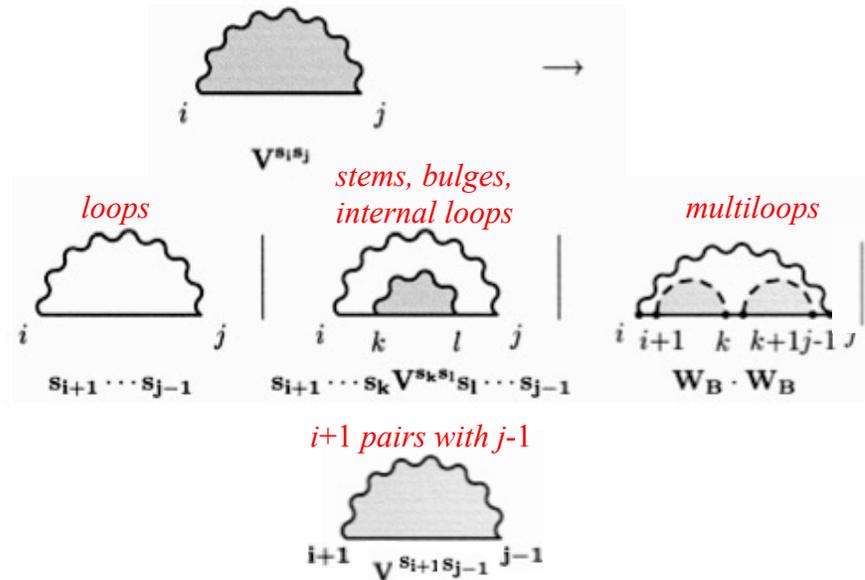
because this sequence has only one possible *leftmost* derivation under grammar  $\mathcal{G}$ .

*If multiple derivations were possible, we would use the Inside Algorithm.*

# Implementing Zuker in an SCFG



The  $V^{ab}$  are the paired states, that is, the states we are in after emitting a pair  $a, b \in \text{alphabet}$ . We therefore have 16 paired states, one for each pair of possibly emitted nucleotides. This allows us to retain information about a neighboring pair when another one is to be emitted, as in stacking correlations. The recursion for state  $V^{ab}$  is (without including hairpin mismatches, which are included in the program), . . .



Here the first transition corresponds to hairpin loops, and is equivalent to function  $FH(i, j)$  in Zuker and Stiegler (1981); the second transition corresponds to stems, bulges, and internal loops, and is equivalent to function  $FL(i, j, k, l)$  in Zuker and Stiegler (1981); the last transition corresponds to multiloops, that is, loops closed by more than two hydrogen bonds.

Rivas & Eddy 2000  
(Bioinformatics 16:583-605)

# Implementing Zuker in an SCFG

## *The thermodynamic model of RNA folding*

*Description of the model.* The model for the thermodynamic implementation is the same as the one presented for the probabilistic model, the only difference being that transition scores are not probabilities, but are taken instead from experimentally determined thermodynamic information provided by the Turner group (Freier *et al.*, 1986; Turner *et al.*, 1987).

$$V^{ab} \longrightarrow cV^{cd}d. \quad (1)$$

This production can be interpreted either thermodynamically or stochastically by

$$-\Delta G[FL(ab, cd)] + V^{cd}(i + 1, j - 1), \quad (2)$$

or

$$\log \frac{P[FL(ab, cd)]}{P^N(c)P^N(d)} + V^{cd}(i + 1, j - 1). \quad (3)$$

Where  $\Delta G[FL(ab, cd)]$  and  $P[FL(ab, cd)]$  stand for the free energy and probability respectively of the stem

Similarly, the partition function calculations introduced by McCaskill (1990) to be used in the thermodynamic implementation (in which all possible folding configurations are taken into account) have their counterpart in the Inside algorithm for a SCFG (Durbin *et al.*, 1998). The Inside algorithm calculates the probability of a RNA sequence given a SCFG by summing over all possible foldings (paths) that the model allows:

$$P(\text{sequence} \mid \text{SCFG}) = \sum_{\text{paths}} P(\text{sequence, path} \mid \text{SCFG}). \quad (4)$$

We have implemented the genefinder as an Inside algorithm. In this way we are taking into account suboptimal foldings that could contribute to the stability of the structure almost as much as the 'best path' or optimal folding calculated by the CYK algorithm.

Rivas & Eddy 2000  
(Bioinformatics 16:583-605)

# The Parsing Problem

## Two questions for a CFG:

- 1) Can a grammar  $G$  derive string  $x$ ?
- 2) If so, what series of productions would be used during the derivation? (*there may be multiple answers!*)

## Additional questions for an SCFG:

- 1) What is the *probability* that  $G$  derives string  $x$ ? (likelihood)
- 2) What is the *most probable* derivation of  $x$  via  $G$ ?

# Chomsky Normal Form (CNF)

Any CFG which does not derive the empty string (i.e.,  $\varepsilon \notin L(G)$ ) can be converted into an equivalent grammar in *Chomsky Normal Form (CNF)*. A CNF grammar is one in which all productions are of the form:

$$X \rightarrow YZ$$

or:

$$X \rightarrow a$$

for nonterminals  $X, Y, Z$ , and terminal  $a$ .

Transforming a CFG into CNF can be accomplished by appropriately-ordered application of the following operations:

- eliminating *useless symbols* (nonterminals that only derive  $\varepsilon$ )
- eliminating *null productions* ( $X \rightarrow \varepsilon$ )
- eliminating *unit productions* ( $X \rightarrow Y$ )
- factoring long rhs expressions ( $A \rightarrow abc$  factored into  $A \rightarrow aB, B \rightarrow bC, C \rightarrow c$ )
- factoring terminals ( $A \rightarrow cB$  is factored into  $A \rightarrow CB, C \rightarrow c$ )

(see, e.g., Hopcroft & Ullman, 1979).

# CNF - Example

Non-CNF:

$$S \rightarrow a S t \mid t S a \mid c S g \mid g S c \mid L$$
$$L \rightarrow N N N N$$
$$N \rightarrow a \mid c \mid g \mid t$$

CNF:

$$S \rightarrow A S_T \mid T S_A \mid C S_G \mid G S_C \mid N L_1$$
$$S_A \rightarrow S A$$
$$S_T \rightarrow S T$$
$$S_C \rightarrow S C$$
$$S_G \rightarrow S G$$
$$L_1 \rightarrow N L_2$$
$$L_2 \rightarrow N N$$
$$N \rightarrow a \mid c \mid g \mid t$$
$$A \rightarrow a$$
$$C \rightarrow c$$
$$G \rightarrow g$$
$$T \rightarrow t$$

Disadvantages of CNF: (1) more nonterminals & productions, (2) more convoluted relation to problem domain (can be important when implementing *posterior decoding*)

Advantages: (1) easy implementation of inference algorithms

# The CYK Parsing Algorithm

Cell  $(i, j)$  contains all the nonterminals  $X$  which can derive the entire subsequence:  
**actagctatctagcttacggtaatcgc**atcgcgc.

$(k+1, j)$  contains only those nonterminals which can derive the red substring.

$(i, k)$  contains only those nonterminals which can derive the green substring.

*initialization:*

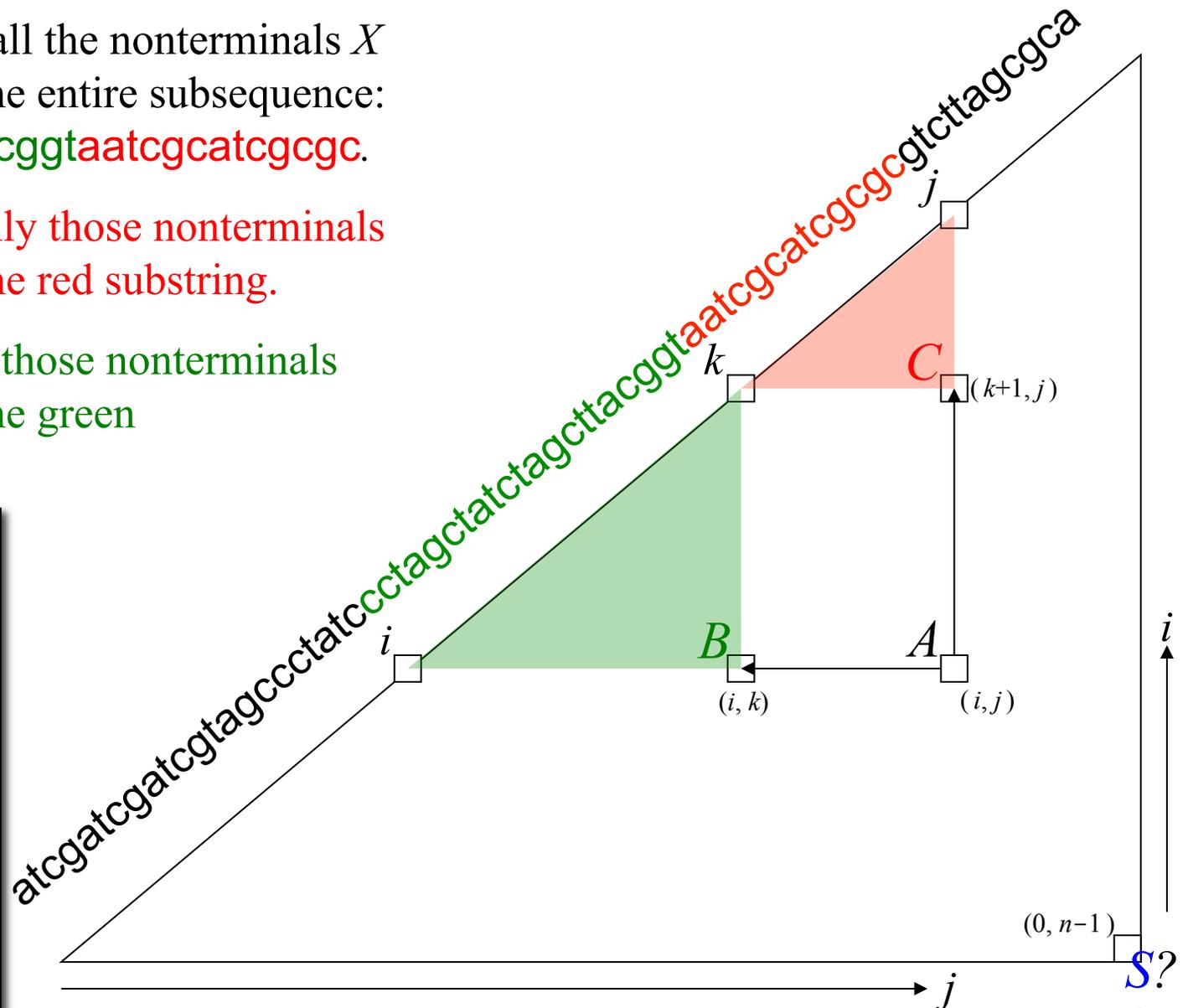
$X \rightarrow x$  (diagonal)

*inductive:*

$A \rightarrow BC$  (for all  $A$ ,  $BC$ , and  $k$ )

*termination:*

is  $S \in D_{0, n-1}$ ?



# The CYK Parsing Algorithm (CFG's)

Given a grammar  $G = (V, \alpha, S, R)$  in CNF, we initialize a DP matrix  $D$  such that:

$$\forall_{0 \leq i < n} D_{i,i} = \{A \mid A \rightarrow x_i \in R\}$$

for the input sequence  $I = x_0 x_1 \dots x_{n-1}$ . The remainder of the DP matrix is then computed row-by-row (left-to-right, top-to-bottom) so that:

$$D_{i,j} = \{A \mid A \rightarrow BC \in R, \text{ for some } B \in D_{i,k} \text{ and } C \in D_{k+1,j}, i \leq k < j\}.$$

for  $0 \leq i < j < n$ . By induction,  $X \in D_{i,j}$  iff  $X \Rightarrow^* x_i x_{i+1} \dots x_j$ . Thus,  $I \in L(G)$  iff  $S \in D_{0,n-1}$ .

We can obtain a derivation  $S \Rightarrow^* I$  from the DP matrix if we augment the above construction so as to include traceback pointers from each nonterminal  $A$  in a cell  $cell_A$  to the two cells  $cell_B$  and  $cell_C$  corresponding to  $B$  and  $C$  in the production  $A \rightarrow BC$  used in the above rule for computing  $D_{i,j}$ . Starting with the symbol  $S$  in cell  $(0, n-1)$ , we can recursively follow the traceback pointers to identify the series of productions for the reconstructed derivation.

# Modified CYK for SCFG's ("Inside Algorithm")

CYK can be easily modified to compute the probability of a string.

We associate a probability with each nonterminal in  $D_{i,j}$ , as follows:

- 1) For each nonterminal  $A$  we multiply the probabilities associated with  $B$  and  $C$  when applying the production  $A \rightarrow BC$  (and also multiply by the probability attached to the production itself)
- 2) We sum the probabilities associated with *different* productions for  $A$  and different values of the "split point"  $k$

The probability of the input string is then given by the probability associated with the start symbol  $S$  in cell  $(0, n-1)$ .

If we instead want the single highest-scoring parse, we can simply perform an *argmax* operation rather than the sums in step #2.

# The Inside Algorithm

Recall that for the *forward algorithm* we defined a *forward variable*  $f(i, j)$ . Similarly, for the *inside algorithm* we define an *inside variable*  $\alpha(i, j, X)$ :

$$\alpha(i, j, X) = P(X \Rightarrow^* x_i \dots x_j | X)$$

which denotes the probability that nonterminal  $X$  will derive subsequence  $x_i \dots x_j$ .

Computing this variable for all integers  $i$  and  $j$  and all nonterminals  $X$  constitutes the *inside algorithm*:

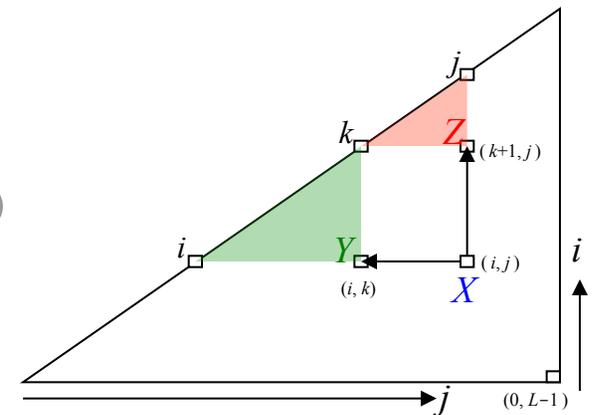
```

for i=0 up to L-1 do
  foreach nonterminal X do
     $\alpha(i, i, X) = P(X \rightarrow x_i)$ ;
for i=L-2 down to 0 do
  for j=i+1 up to L-1 do
    foreach nonterminal X do

```

$$\alpha(i, j, X) = \sum_Y \sum_Z \sum_{k=i \dots j-1} P(X \rightarrow YZ) \alpha(i, k, Y) \alpha(k+1, j, Z);$$

$time = O(L^3 N^3)$



Note that  $P(X \rightarrow YZ) = 0$  if  $X \rightarrow YZ$  is not a valid production in the grammar.

The probability  $P(x|G)$  of the full input sequence  $x$  of length  $L$  can then be found in the final cell of the matrix:  $\alpha(0, L-1, S)$  (the “**corner cell**”). Reconstructing the **most probable derivation** (“**parse**”) can be done by modifying this algorithm to (1) compute **max**’s instead of sums, and (2) to keep **traceback pointers** as in Viterbi.

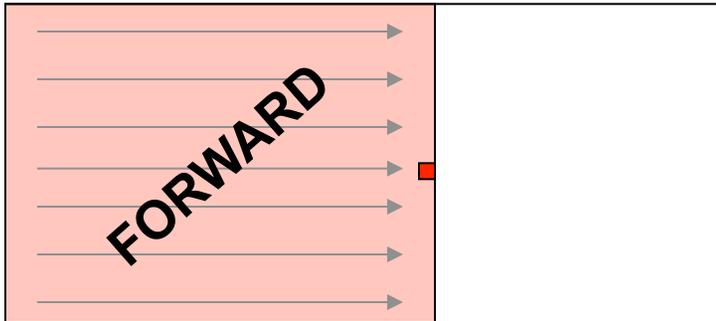
# Training an SCFG

Two common methods for training an SCFG:

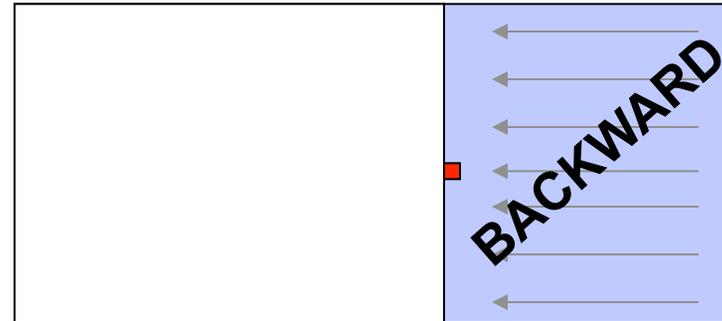
- 1) If parses are known for the training sequences, we can simply count the number of times each production occurs in the training parses and normalize these counts into probabilities. This is analogous to “*labeled sequence training*” of an HMM (i.e., when each symbol in a training sequence is labeled with an HMM state).
- 2) If parses are NOT known for the training sequences, we can use an EM algorithm similar to the Baum-Welch algorithm for HMMs. The EM algorithm for SCFGs is called *Inside-Outside*.

# Recall: Forward-Backward

CATCGTATCGCGGATATCTCGATCATCGCTCGACTATTATATCA



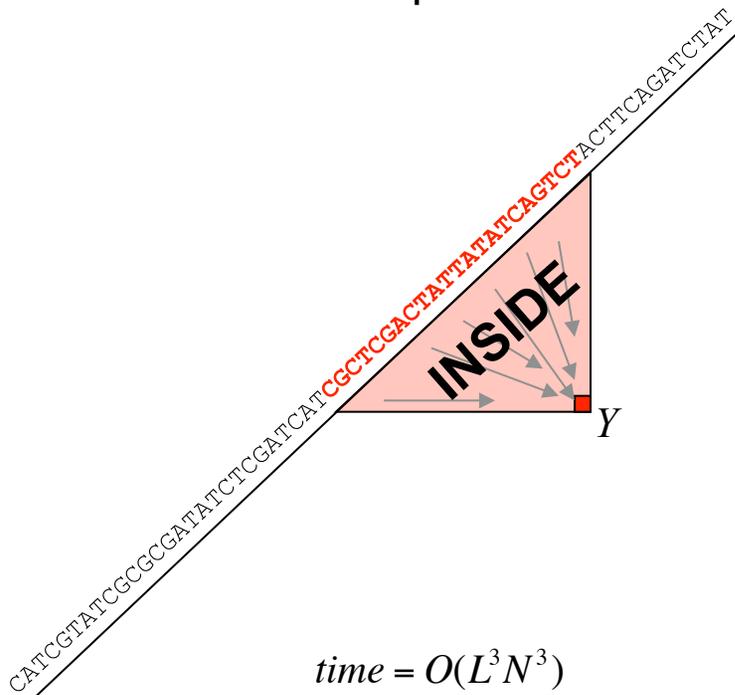
CATCGTATCGCGGATATCTCGATCATCGCTCGACTATTATATCA



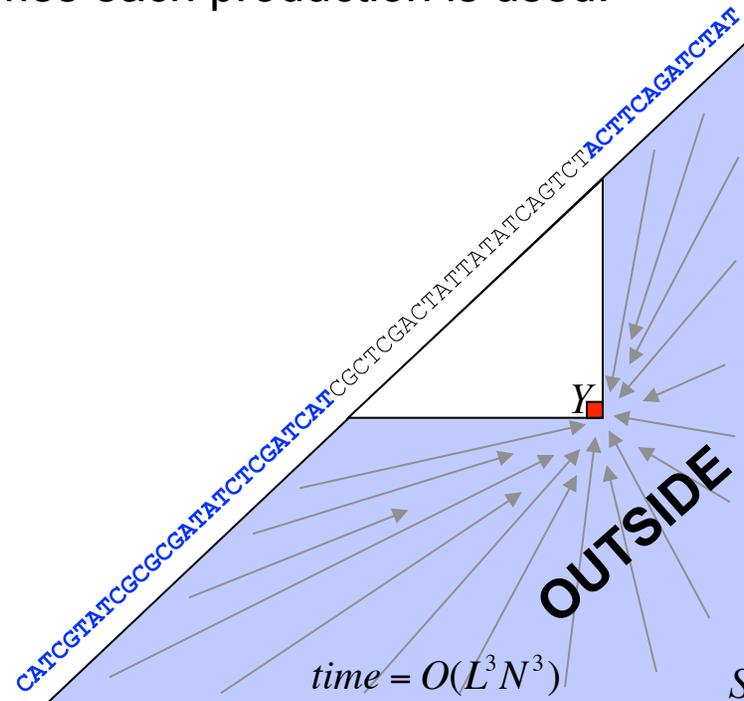
$L = \text{length}, N = \text{\#states}$       $\text{time} = O(LN^2)$

$\text{time} = O(LN^2)$

Inside-Outside uses a similar trick to estimate the expected number of times each production is used:



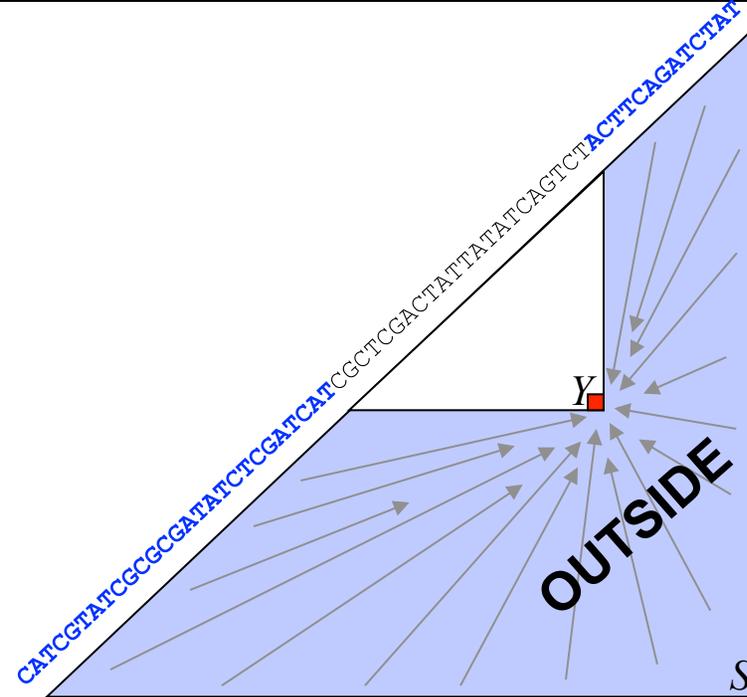
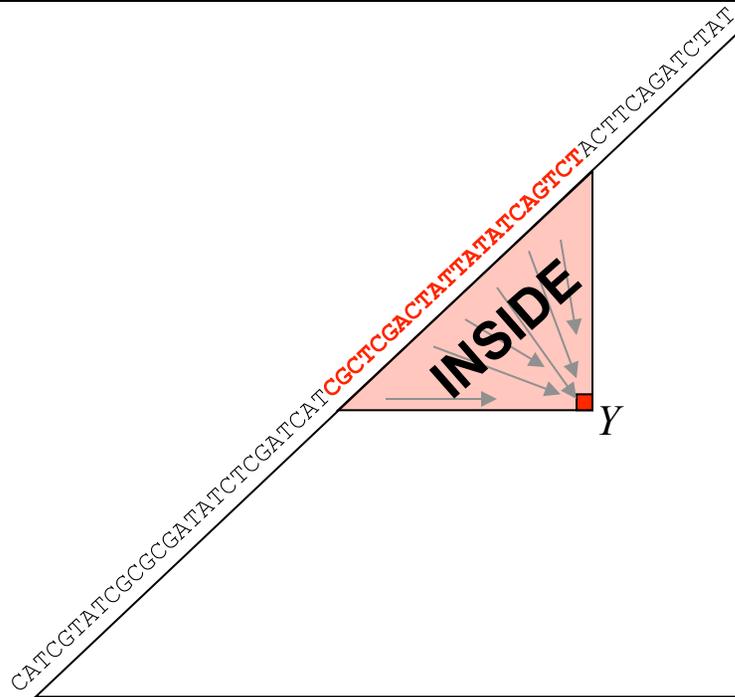
$\text{time} = O(L^3 N^3)$



$\text{time} = O(L^3 N^3)$

$L = \text{length}, N = \text{\#nonterminals}$

# Inside vs. Outside



$$\alpha(i, j, Y) = P( Y \Rightarrow^* \text{CGCTCGACTATTATATCAGTCT} \mid Y )$$

$$\beta(i, j, Y) = P( S \Rightarrow^* \text{CATCGTATCGCGGATATCTCGATCAT} Y \text{ACTTCAGATCTAT} )$$

$$\alpha(i, j, Y) \beta(i, j, Y) =$$

$$P( S \Rightarrow^* \text{CATCGTATCGCGGATATCTCGATCATCGCTCGACTATTATATCAGTCTACTTCAGATCTAT} ,$$

with the red subsequence being generated by  $Y$  )

$$\frac{\alpha(i, j, Y) \beta(i, j, Y)}{\alpha(0, L-1, S)} = \text{posterior probability } P(Y, i, j \mid \text{full sequence})$$

(def. of CFG: inside seq. is cond. indep. outside seq., given  $Y$ )

# The Outside Algorithm

For the *outside algorithm* we define an *outside variable*  $\beta(i, j, Y)$ :

$$\beta(i, k, Y) = P(S \Rightarrow^* x_0 \dots x_{i-1} Y x_{k+1} \dots x_{L-1})$$

which denotes the probability that the start symbol  $S$  will derive the sentential form  $x_0 \dots x_{i-1} Y x_{k+1} \dots x_{L-1}$  (i.e., that  $S$  will derive some string having prefix  $x_0 \dots x_{i-1}$  and suffix  $x_{k+1} \dots x_{L-1}$  and that the region between will be derived through nonterminal  $Y$ ).

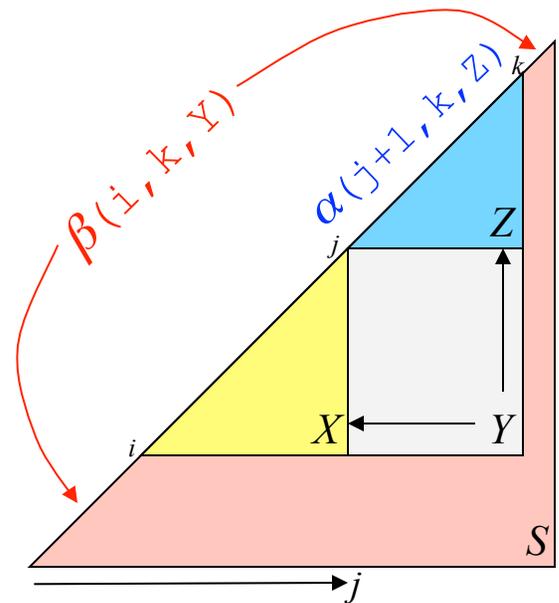
```

 $\beta(0, L-1, S) = 1;$ 
foreach  $X \neq S$  do  $\beta(0, L-1, X) = 0;$ 
for  $i=0$  up to  $L-1$  do
  for  $j=L-1$  down to  $i$  do
    foreach nonterminal  $X$  do
      if  $\beta(i, j, X)$  undefined then

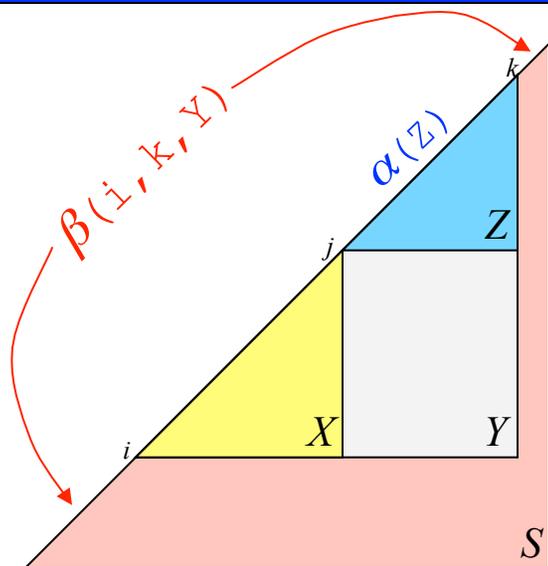
```

$$\beta(i, j, X) = \sum_Y \sum_Z \sum_{k=j+1..L-1} P(Y \rightarrow XZ) \alpha(j+1, k, Z) \beta(i, k, Y) + \sum_Y \sum_Z \sum_{k=0..i-1} P(Y \rightarrow ZX) \alpha(k, i-1, Z) \beta(k, j, Y);$$

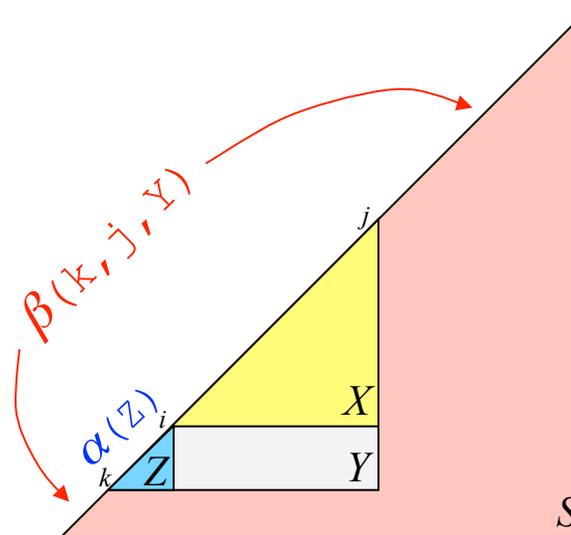
$time = O(L^3 N^3)$



# The Two Cases in the Outside Recursion

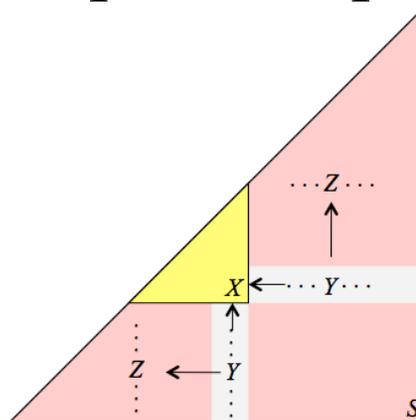


**$Y \rightarrow XZ$**



**$Y \rightarrow ZX$**

In both cases we compute  $\beta(X)$  in terms of  $\beta(Y)$  and  $\alpha(Z)$ , summing over all possible positions of  $Y$  and  $Z$ :



# Inside-Outside Parameter Estimation

EM-update equations:

$$P_{new}(X \rightarrow YZ) = \frac{E(X \rightarrow YZ|x)}{E(X|x)}$$

$$= \frac{\sum_{i=0}^{L-2} \sum_{j=i+1}^{L-1} \sum_{k=i}^{j-1} \beta(i, j, X) P(X \rightarrow YZ) \alpha(i, k, Y) \alpha(k+1, j, Z)}{\sum_{i=0}^{L-1} \sum_{j=i}^{L-1} \beta(i, j, X) \alpha(i, j, X)}$$

$$P_{new}(X \rightarrow a) = \frac{E(X \rightarrow a|x)}{E(X|x)}$$

$$= \frac{\sum_{i=0}^{L-1} \delta_{kron}(x_i, a) \beta(i, i, X) P(X \rightarrow a)}{\sum_{i=0}^{L-1} \sum_{j=i}^{L-1} \beta(i, j, X) \alpha(i, j, X)}$$

(see Durbin *et al.*, 1998, section 9.6)

# Posterior Decoding for SCFG's

“What is the probability that nonterminal  $X$  generates  $x_i$  (in some particular sequence)?”

$$P(X \Rightarrow x_i | x) = \frac{\beta(i, i, X) P(X \rightarrow x_i)}{\alpha(0, L-1, S)}$$

“What is the probability that nonterminal  $X$  generates the subsequence  $x_i \dots x_j$  via production  $X \rightarrow YZ$ , with  $Y$  generating  $x_i \dots x_k$  and  $Z$  generating  $x_{k+1} \dots x_j$ ?”

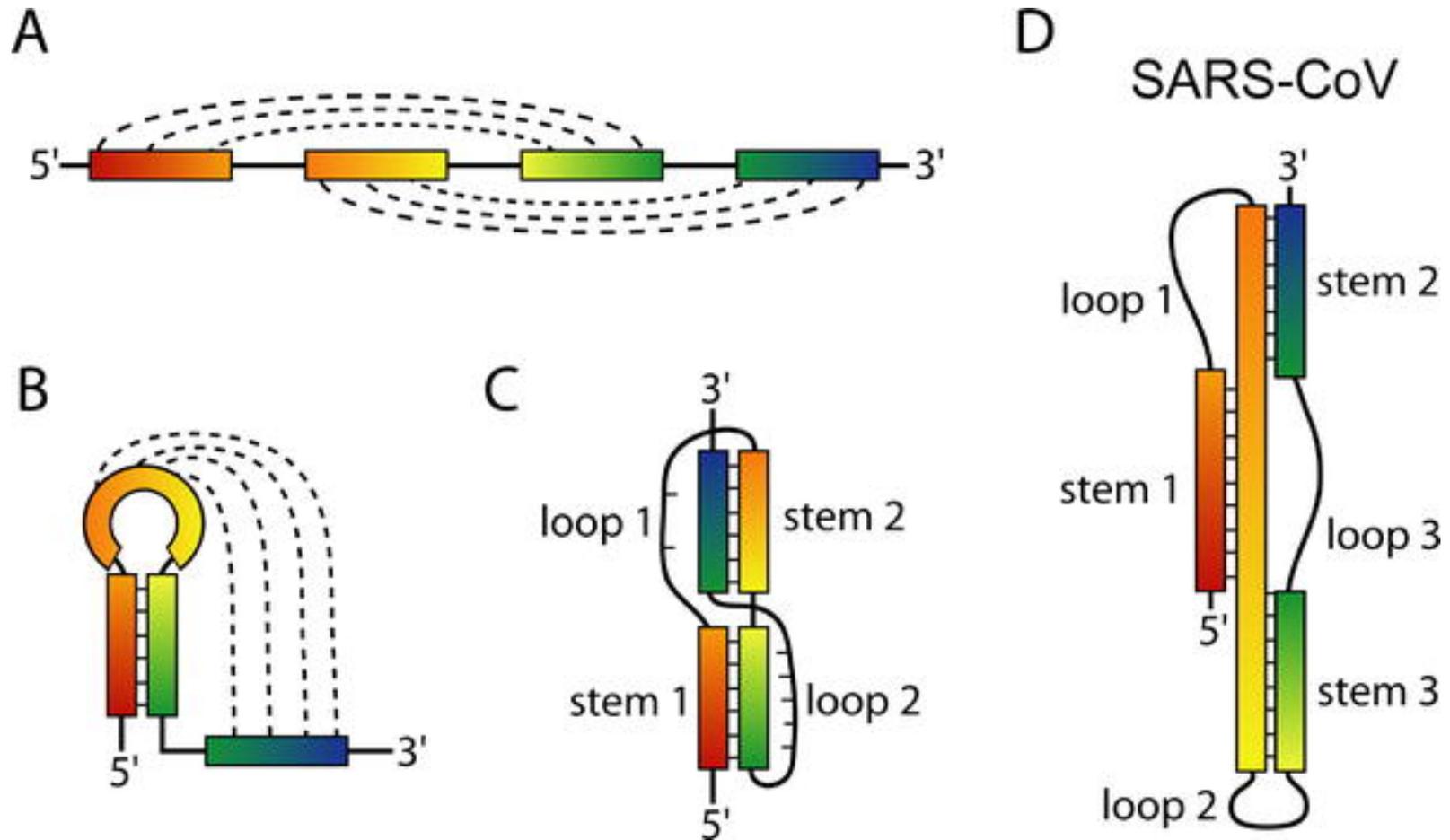
$$P(X_{i,j} \Rightarrow Y_{i,k} Z_{k+1,j} | x) = \frac{\beta(i, j, X) P(X \rightarrow YZ) \alpha(i, k, Y) \alpha(k+1, j, Z)}{\alpha(0, L-1, S)}$$

“What is the probability that a structural feature of type  $F$  will occupy sequence positions  $i$  through  $j$ ?”

$$P(F_{i,j} | x) = \frac{\dots \text{some product of } \alpha\text{'s and } \beta\text{'s (and other things)} \dots}{\alpha(0, L-1, S)}$$

# What about Pseudoknots?

"Among the most prevalent RNA structures is a motif known as the pseudoknot." (Staple & Butcher, 2005)



# Context-Sensitive Grammar for Pseudoknots

$$L(G) = \{ x y x^r y^r \mid x \in \alpha^*, y \in \alpha^* \}$$

$$S \rightarrow LXR$$

}place markers at left/right ends

$$X \rightarrow aXt \mid tXa \mid cXg \mid gXc \mid Y$$

}generate  $x$  and  $x^r$

$$Y \rightarrow aAY \mid cCY \mid gGY \mid tTY \mid \epsilon$$

}generate  $y$  and encoded 2<sup>nd</sup> copy

$$\begin{aligned} Aa &\rightarrow aA, & Ac &\rightarrow cA, & Ag &\rightarrow gA, & At &\rightarrow tA \\ Ca &\rightarrow aC, & Cc &\rightarrow cC, & Cg &\rightarrow gC, & Ct &\rightarrow tC \\ Ga &\rightarrow aG, & Gc &\rightarrow cG, & Gg &\rightarrow gG, & Gt &\rightarrow tG \\ Ta &\rightarrow aT, & Tc &\rightarrow cT, & Tg &\rightarrow gT, & Tt &\rightarrow tT \end{aligned}$$

} propagate encoded copy of  $y$  to end of sequence

$$AR \rightarrow tR, \quad CR \rightarrow gR, \quad GR \rightarrow cR, \quad TR \rightarrow aR$$

} reverse-complement second  $y$  at end of sequence

$$La \rightarrow aL, \quad Lc \rightarrow cL, \quad Lg \rightarrow gL, \quad Lt \rightarrow tL, \quad LR \rightarrow \epsilon$$

} erase extra “markers”

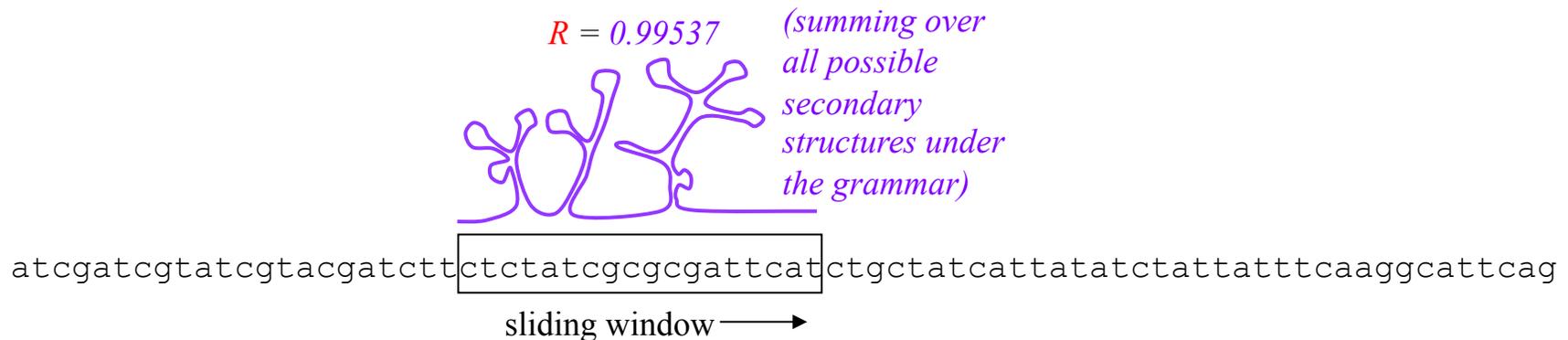
# Sliding Windows to Find ncRNA Genes

Given a grammar  $G$  describing ncRNA structures and an input sequence  $Z$ , we can slide a window of length  $L$  across the sequence, computing the probability  $P(Z_{i,i+L-1}|G)$  that the subsequence  $Z_{i,i+L-1}$  falling within the current window could have been generated by grammar  $G$ .

Using a likelihood ratio:

$$R = P(Z_{i,i+L-1}|G) / P(Z_{i,i+L-1}|\text{background}),$$

we can impose the rule that any subsequence having a score  $R \gg 1$  is likely to contain a ncRNA gene (where the *background model* is typically a *Markov chain*).



# Summary

- An *SCFG* is a generative model utilizing production rules to generate strings
- SCFG's are more powerful than HMM's because they can model arbitrary runs of *paired nested elements*, such as base-pairings in a stem-loop structure. They can't model *pseudoknots* (though *context-sensitive* grammars can)
- *Thermodynamic* folding algorithms can be simulated in an SCFG
- The probability of a string  $S$  being generated by an SCFG  $G$  can be computed using the *Inside Algorithm*
- Given a set of productions for a SCFG, the parameters can be estimated using the *Inside-Outside* (EM) algorithm