CSI5126. Algorithms in bioinformatics

RNA Secondary Structure **Search Problem**

Marcel Turcotte

School of Electrical Engineering and Computer Science (EECS)
University of Ottawa

Version November 20, 2018

Summary

We learnt that **RNA** evolves so as to preserve **bair pairs** patterns more than sequence. We discussed the impact on traditional bioinformatics approaches. Finally, we derived a dynamic programming algorithm to solve the **inference problem**. In this lecture, we will consider the **search problem**.

General objective

- **Implement** a pattern matching algorithm using context free grammars specifically to detect sequences who could fold into a specific structure.

Reading

- Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchinson (1998). Biological sequence analysis. Probabilistic models of proteins and nucleic acids. Cambridge University Press. Pages 277-297.

Project

- **Presentations**: 20 minutes
  - Tuesday, November 27, 2018
  - Thursday, November 29, 2018
  - Tuesday, December 4, 2018

---

https://docs.google.com/document/d/1gfcGDWWF4iLxpxLEAaBHDi-aY6Ome_p9D5RE2evLJE0

Summary

- RNA molecules play **important cellular roles**
- **Secondary structure** is more preserved than sequence
- **Nussinov-Jacobson** is an $\mathcal{O}(n^3)$ algorithm that maximizes the total number of base pairs
- MFOLD (by **Zuker**) is an $\mathcal{O}(n^3)$ algorithm that minimizes the free energy
- The **accessible pairs, cycles and order notation** are key to understand the recurrence equations of MFE methods
- **Consensus methods***, based on Sankoff 1985 algorithm, perform more consistently, but have a high time/space complexity

*Simultaneous alignment and folding

Summary

- RNA molecules play **important cellular roles**
- **Secondary structure** is more preserved than sequence
- **Nussinov-Jacobson** is an $\mathcal{O}(n^3)$ algorithm that maximizes the total number of base pairs
- MFOLD (by **Zuker**) is an $\mathcal{O}(n^3)$ algorithm that minimizes the free energy
- The **accessible pairs, cycles and order notation** are key to understand the recurrence equations of MFE methods
- **Consensus methods***, based on Sankoff 1985 algorithm, perform more consistently, but have a high time/space complexity

*Simultaneous alignment and folding

Summary

- RNA molecules play **important cellular roles**
- **Secondary structure** is more preserved than sequence
- **Nussinov-Jacobson** is an $\mathcal{O}(n^3)$ algorithm that maximizes the total number of base pairs
- MFOLD (by **Zuker**) is an $\mathcal{O}(n^3)$ algorithm that minimizes the free energy
- The **accessible pairs, cycles and order notation** are key to understand the recurrence equations of MFE methods
- **Consensus methods***, based on Sankoff 1985 algorithm, perform more consistently, but have a high time/space complexity

*Simultaneous alignment and folding

Summary

- RNA molecules play **important cellular roles**
- **Secondary structure** is more preserved than sequence
- **Nussinov-Jacobson** is an $\mathcal{O}(n^3)$ algorithm that maximizes the total number of base pairs
- MFOLD (by **Zuker**) is an $\mathcal{O}(n^3)$ algorithm that minimizes the free energy
- The **accessible pairs, cycles and order notation** are key to understand the recurrence equations of MFE methods
- **Consensus methods***, based on Sankoff 1985 algorithm, perform more consistently, but have a high time/space complexity

*Simultaneous alignment and folding

Summary

- RNA molecules play **important cellular roles**
- **Secondary structure** is more preserved than sequence
- **Nussinov-Jacobson** is an $\mathcal{O}(n^3)$ algorithm that maximizes the total number of base pairs
- MFOLD (by **Zuker**) is an $\mathcal{O}(n^3)$ algorithm that minimizes the free energy
- The **accessible pairs, cycles and order notation** are key to understand the recurrence equations of MFE methods
- **Consensus methods***, based on Sankoff 1985 algorithm, perform more consistently, but have a high time/space complexity

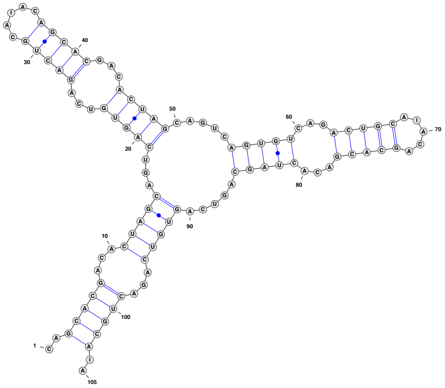
*Simultaneous alignment and folding

Summary

- RNA molecules play **important cellular roles**
- **Secondary structure** is more preserved than sequence
- **Nussinov-Jacobson** is an $\mathcal{O}(n^3)$ algorithm that maximizes the total number of base pairs
- MFOLD (by **Zuker**) is an $\mathcal{O}(n^3)$ algorithm that minimizes the free energy
- The **accessible pairs, cycles and order notation** are key to understand the recurrence equations of MFE methods
- **Consensus methods***, based on Sankoff 1985 algorithm, perform more consistently, but have a high time/space complexity

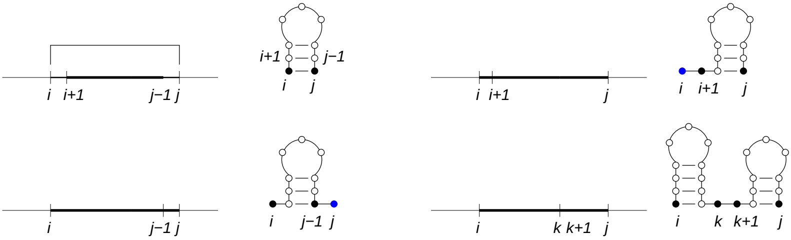
*Simultaneous alignment and folding

RNA **secondary structure**




```
GCACGACACUAGCAGUCAGUGUCAGACUGCAIACAGCACGACACUAGCAGUCAGUGUCAGACUGCAIACAGCACGACACUAGCAGUCAGUGUCA
(((((...(((((...(((((...(((((.....)))))...))))).....(((((...(((((.....)))))...))))).....)))))
```

Inference problem: Nussinov-Jacobson



Nussinov-Jacobson algorithm

Initialisation:

$$\gamma(i, i+k) \quad = \quad 0 \quad \text{for k} = 0 \text{ to } 1 \text{ and for i} = 1 \text{ to n} - \text{k}.$$

Recurrence:

$$\gamma(i,j) = \max \begin{cases} \gamma(i+1, j-1) + \delta(i,j); \\ \gamma(i+1, j); \\ \gamma(i, j-1); \\ \max_{i<k<(j-1)}[\gamma(i,k) + \gamma(k+1, j)]. \end{cases}$$

Matching score:

$$\delta(i,j) = \begin{cases} 1, \text{if } a_i : a_j \in \{A:U, U:A, G:C, C:G\} \bigcup \{G:U, U:G\}; \\ 0, \text{otherwise}. \end{cases}$$

	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	0	1	2	3
G	0	0	0	0	0	0	1	2	3
G		0	0	0	0	0	1	2	2
A			0	0	0	0	1	1	1
A				0	0	0	1	1	1
A					0	0	1	1	1
U						0	0	0	0
C							0	0	0
C								0	0

Other **paradigms**

- Reporting **sub-optimal structures** (MFOLD, SFOLD)
- **Partition function** and the McCaskill's calculation of P_{ij}'s
- **Folding kinetics**, identifying ribo-switches
- **MFE** for secondary structure for **interacting RNA molecules**
- **Partition function** for secondary structure for **interacting RNA molecules**
- Non-coding RNAs (**ncRNA genes**) identification (EvoFold, RNAz...)

Other **paradigms**

- Reporting **sub-optimal structures** (MFOLD, SFOLD)
- **Partition function** and the McCaskill's calculation of P_{ij}'s
- **Folding kinetics**, identifying ribo-switches
- **MFE** for secondary structure for **interacting RNA molecules**
- **Partition function** for secondary structure for **interacting RNA molecules**
- Non-coding RNAs (**ncRNA genes**) identification (EvoFold, RNAz...)

Other **paradigms**

- Reporting **sub-optimal structures** (MFOLD, SFOLD)
- **Partition function** and the McCaskill's calculation of P_{ij}'s
- **Folding kinetics**, identifying ribo-switches
- **MFE** for secondary structure for **interacting RNA molecules**
- **Partition function** for secondary structure for **interacting RNA molecules**
- Non-coding RNAs (**ncRNA genes**) identification (EvoFold, RNAz...)

Other **paradigms**

- Reporting **sub-optimal structures** (MFOLD, SFOLD)
- **Partition function** and the McCaskill's calculation of P_{ij}'s
- **Folding kinetics**, identifying ribo-switches
- **MFE** for secondary structure for **interacting RNA molecules**
- **Partition function** for secondary structure for **interacting RNA molecules**
- Non-coding RNAs (**ncRNA genes**) identification (EvoFold, RNAz...)

Other **paradigms**

- Reporting **sub-optimal structures** (MFOLD, SFOLD)
- **Partition function** and the McCaskill's calculation of P_{ij}'s
- **Folding kinetics**, identifying ribo-switches
- **MFE** for secondary structure for **interacting RNA molecules**
- **Partition function** for secondary structure for **interacting RNA molecules**
- Non-coding RNAs (**ncRNA genes**) identification (EvoFold, RNAz...)

Other **paradigms**

- Reporting **sub-optimal structures** (MFOLD, SFOLD)
- **Partition function** and the McCaskill's calculation of P_{ij}'s
- **Folding kinetics**, identifying ribo-switches
- **MFE** for secondary structure for **interacting RNA molecules**
- **Partition function** for secondary structure for **interacting RNA molecules**
- Non-coding RNAs (**ncRNA genes**) identification (EvoFold, RNAz...)

Now **what**?

A secondary structure was inferred!

- It can be analyzed in order to **propose new experiments**, to **propose a mechanism of action**, or to **develop novel therapeutic approaches** (a new drug for instance)
- It can be used for finding new members of its family (**homologues**) and this requires adapted database searching techniques
- It can serve as a starting point for predicting the **three-dimensional structure**

Now **what**?

A secondary structure was inferred!

- It can be analyzed in order to **propose new experiments**, to **propose a mechanism of action**, or to **develop novel therapeutic approaches** (a new drug for instance)
- It can be used for finding new members of its family (**homologues**) and this requires adapted database searching techniques
- It can serve as a starting point for predicting the **three-dimensional structure**

Now **what**?

A secondary structure was inferred!

- It can be analyzed in order to **propose new experiments**, to **propose a mechanism of action**, or to **develop novel therapeutic approaches** (a new drug for instance)
- It can be used for finding new members of its family (**homologues**) and this requires adapted database searching techniques
- It can serve as a starting point for predicting the **three-dimensional structure**

Now **what**?

A secondary structure was inferred!

- It can be analyzed in order to **propose new experiments**, to **propose a mechanism of action**, or to **develop novel therapeutic approaches** (a new drug for instance)
- It can be used for finding new members of its family (**homologues**) and this requires adapted database searching techniques
- It can serve as a starting point for predicting the **three-dimensional structure**

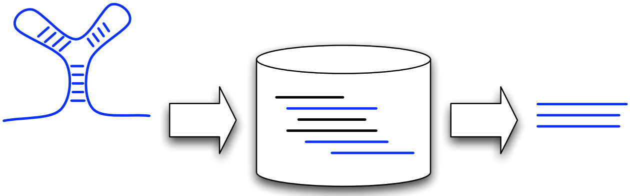
Now **what**?

A secondary structure was inferred!

- It can be analyzed in order to **propose new experiments**, to **propose a mechanism of action**, or to **develop novel therapeutic approaches** (a new drug for instance)
- It can be used for finding new members of its family (**homologues**) and this requires adapted database searching techniques
- It can serve as a starting point for predicting the **three-dimensional structure**

Database **search problem**

Find all sequences matching a user specified **secondary structure motif** or all the **sequences that can be folded into a user specified structure**



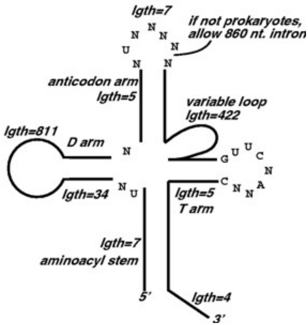
Non-probabilistic approaches

- The first practical approaches were **non-probabilistic**
- A **description language** allows the users to represent structural motifs, and search databases
- **RNAMOT**, **RNABOB**, **PatScan**, and **RNAMOTIF**

```
parms
    wc += gu;

descr
    h5(minlen=6,maxlen=7)
        ss(len=2)
        h5(minlen=3,maxlen=4)
            ss(minlen=4,maxlen=
        h3
        ss(len=1)
        h5(minlen=4,maxlen=5)
            ss(len=7)
        h3
        ss(minlen=4,maxlen=21)
        h5(minlen=4,maxlen=5)
            ss(len=7)
        h3
     h3
     ss(len=4)
```

A.



RNAMOT

- Gautheret D., Major F. & Cedergren R. (1990) Pattern searching/alignment with RNA primary and secondary structures: an effective descriptor for tRNA. *Comp. Appl. Biosc.* **6**, 325-331.
- Laferriere A., Gautheret D. & Cedergren R. (1994) An RNA pattern matching program with enhanced performances and portability. *Comp. Appl. Biosci.* **10**, 209-210.
- rna.igmors.u-psud.fr/gautheret/download

RNABOB

*RNABOB is an implementation of D. Gautheret's RNAMOT, but with a different underlying algorithm **using a non-deterministic finite state machine with node rewriting rules**. (Computer scientists would probably cringe in horror. It works, and it's fast, but is it street legal in a computer science department? Who knows.) If you're looking for an RNA motif that fits a hard consensus pattern — a la PROSITE patterns, but with base-pairing — you might check out RNABOB.*

- `http://eddylab.org/software.html`

RNAMOTIF

- Macke et al. (2001) *Nuc. Acids. Res.* **29**(22):4724-4735.
- **Sophisticated scripting language**
- Matches can be **ranked** using a user-defined scoring function
- **Minimum free energy** can be used in the definition of the scoring function
- casegroup.rutgers.edu/casegr-sh-2.5.html

Discussion

What are the main limitations?

Discussion

What are the main limitations?

- These computer programs are practical and can be applied to large data-sets

Discussion

What are the main limitations?

- These computer programs are practical and can be applied to large data-sets
- **Hard consensus pattern means hit-or-miss**

Discussion

What are the main limitations?

- These computer programs are practical and can be applied to large data-sets
- **Hard consensus pattern means hit-or-miss**
  - The major difficulties arises from the subjectivity in deriving the best descriptor for a family of sequences

Discussion

What are the main limitations?

- These computer programs are practical and can be applied to large data-sets
- **Hard consensus pattern means hit-or-miss**
  - The major difficulties arises from the subjectivity in deriving the best descriptor for a family of sequences
  - It can be quite difficult to design a pattern with both high sensitivity and high specificity

Discussion

How can one move away from “hard” patterns?

Discussion

How can one move away from "hard" patterns?

- **Edit-distance**

Discussion

How can one move away from "hard" patterns?

- **Edit-distance**
  - G. Myers. Approximately matching context-free languages. *Information Processing Letters* vol. **54** (2) pp. 85-92, 1995.
  - $\mathcal{O}(P^5 N^8 8^P)$, where P is the size of the grammar and N is length of the string.

Discussion

How can one move away from "hard" patterns?

- **Edit-distance**
  - G. Myers. Approximately matching context-free languages. *Information Processing Letters* vol. **54** (2) pp. 85-92, 1995.
  - $\mathcal{O}(P^5N^88^P)$, where P is the size of the grammar and N is length of the string.
- ***k*-mismatches**

Discussion

How can one move away from "hard" patterns?

- **Edit-distance**
  - G. Myers. Approximately matching context-free languages. *Information Processing Letters* vol. **54** (2) pp. 85-92, 1995.
  - $\mathcal{O}(P^5 N^8 8^P)$, where P is the size of the grammar and N is length of the string.
- ***k*-mismatches**
  - N. El-Mabrouk, M. Raffinot, J.E. Duchesne, M. Lajoie and N. Luc. Approximate Matching of Secondary Structures. *Journal of Bioinformatics and Computational Biology*, Vol. **3**, No. 2, pp. 317-342, 2005.
  - $\mathcal{O}(krpn)$, k is error threshold, n is string size, p is secondary structure size, r is number of "union" symbols

Discussion

How can one move away from “hard” patterns?

- **Edit-distance**
  - G. Myers. Approximately matching context-free languages. *Information Processing Letters* vol. **54** (2) pp. 85-92, 1995.
  - $\mathcal{O}(P^5 N^8 8^P)$, where P is the size of the grammar and N is length of the string.
- ***k*-mismatches**
  - N. El-Mabrouk, M. Raffinot, J.E. Duchesne, M. Lajoie and N. Luc. Approximate Matching of Secondary Structures. *Journal of Bioinformatics and Computational Biology*, Vol. **3**, No. 2, pp. 317-342, 2005.
  - $\mathcal{O}(krpn)$, k is error threshold, n is string size, p is secondary structure size, r is number of “union” symbols
- **Probabilistic**,

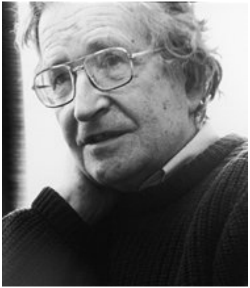
Discussion

How can one move away from "hard" patterns?

- **Edit-distance**
  - G. Myers. Approximately matching context-free languages. *Information Processing Letters* vol. **54** (2) pp. 85-92, 1995.
  - $\mathcal{O}(P^5 N^8 8^P)$, where P is the size of the grammar and N is length of the string.
- ***k*-mismatches**
  - N. El-Mabrouk, M. Raffinot, J.E. Duchesne, M. Lajoie and N. Luc. Approximate Matching of Secondary Structures. *Journal of Bioinformatics and Computational Biology*, Vol. **3**, No. 2, pp. 317-342, 2005.
  - $\mathcal{O}(krpn)$, k is error threshold, n is string size, p is secondary structure size, r is number of "union" symbols
- **Probabilistic**, a principled approach

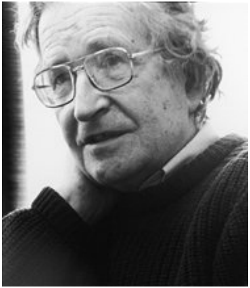
Transformational **grammars**

- Pioneered by **Noam Chomsky** in the ’50s to model natural languages
- **Formal grammars allow to determine what novel sentences are grammatical or not**
- Transformational grammars are sometimes called **generative grammars**
- **We look at non-probabilistic grammars first!**



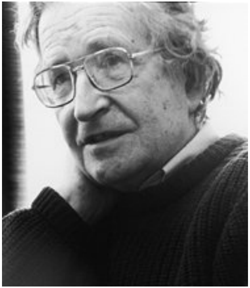
Transformational **grammars**

- Pioneered by **Noam Chomsky** in the '50s to model natural languages
- **Formal grammars allow to determine what novel sentences are grammatical or not**
- Transformational grammars are sometimes called **generative grammars**
- **We look at non-probabilistic grammars first!**



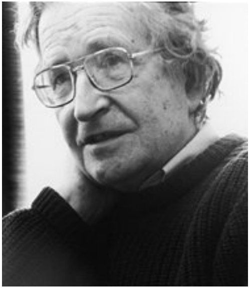
Transformational **grammars**

- Pioneered by **Noam Chomsky** in the ’50s to model natural languages
- **Formal grammars allow to determine what novel sentences are grammatical or not**
- Transformational grammars are sometimes called **generative grammars**
- **We look at non-probabilistic grammars first!**

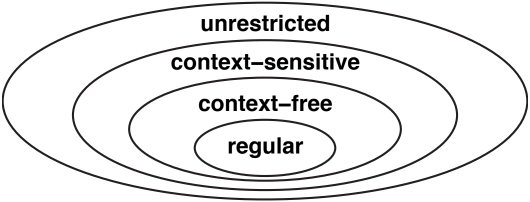


Transformational **grammars**

- Pioneered by **Noam Chomsky** in the ’50s to model natural languages
- **Formal grammars allow to determine what novel sentences are grammatical or not**
- Transformational grammars are sometimes called **generative grammars**
- **We look at non-probabilistic grammars first!**

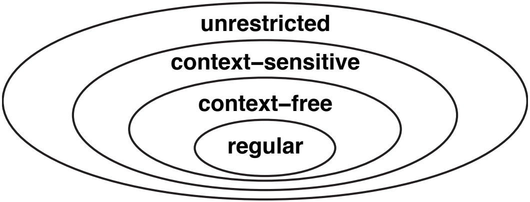


Chomsky hierarchy of transformational grammars



- Increasing order of **expressivity**, but also increasing order of computational **resources**.
- Each class of languages has its **associated machine** that serves for parsing (accepting, deciding, recognizing) sentences of this language.

Chomsky hierarchy of transformational grammars



- Increasing order of **expressivity**, but also increasing order of computational **resources**.
- Each class of languages has its **associated machine** that serves for parsing (accepting, deciding, recognizing) sentences of this language.

Transformational grammars: **definitions**

- Constituted of **symbols** and **rewriting rules** (also called **production rules**) having the following form,

$$\alpha \rightarrow \beta$$

- 2 types of symbols: **terminal** symbols and **non-terminal** symbols
- The **left-hand side** of a rule contains **at least one non-terminal symbol**, which is rewritten into the right hand-side of the rule
- Terminal symbols represents **instances of the language**, here nucleotides, and will be represented by lower-case letters

Transformational grammars: **definitions**

- Constituted of **symbols** and **rewriting rules** (also called **production rules**) having the following form,

$$\alpha \rightarrow \beta$$

- 2 types of symbols: **terminal** symbols and **non-terminal** symbols
- The **left-hand side** of a rule contains **at least one non-terminal symbol**, which is rewritten into the right hand-side of the rule
- Terminal symbols represents **instances of the language**, here nucleotides, and will be represented by lower-case letters

Transformational grammars: **definitions**

- Constituted of **symbols** and **rewriting rules** (also called **production rules**) having the following form,

$$\alpha \rightarrow \beta$$

- 2 types of symbols: **terminal** symbols and **non-terminal** symbols
- The **left-hand side** of a rule contains **at least one non-terminal symbol**, which is rewritten into the right hand-side of the rule
- Terminal symbols represents **instances of the language**, here nucleotides, and will be represented by lower-case letters

Transformational grammars: **definitions**

- Constituted of **symbols** and **rewriting rules** (also called **production rules**) having the following form,

$$\alpha \rightarrow \beta$$

- 2 types of symbols: **terminal** symbols and **non-terminal** symbols
- The **left-hand side** of a rule contains **at least one non-terminal symbol**, which is rewritten into the right hand-side of the rule
- Terminal symbols represents **instances of the language**, here nucleotides, and will be represented by lower-case letters

Transformational grammars: **definitions**

- A small example, a grammar denoted by G

$$
\begin{array}{lclcl}
S & \rightarrow & gS_1 & | & cS_2 \\
S_1 & \rightarrow & cS_2 & | & \epsilon \\
S_2 & \rightarrow & gS_1 & | & \epsilon
\end{array}
$$

- A **derivation** is the successive application of the rules starting with S (the start nonterminal).

$$S \Rightarrow cS_2 \Rightarrow cgS_1 \Rightarrow cgcS_2 \Rightarrow cgcgS_1 \Rightarrow cgcg$$

- The **language** generated by G, denoted $\mathcal{L}(G)$, is all the strings that can be **derived** from S, $\{w|S \stackrel{*}{\Rightarrow} w\}$.
- A string is **accepted** by the grammar if there exist a derivation of the string from S.

Transformational grammars: **definitions**

- A small example, a grammar denoted by G

$$
\begin{array}{lcl}
S & \rightarrow & gS_1 \mid cS_2 \\
S_1 & \rightarrow & cS_2 \mid \epsilon \\
S_2 & \rightarrow & gS_1 \mid \epsilon
\end{array}
$$

- A **derivation** is the successive application of the rules starting with S (the start nonterminal).

$$S \Rightarrow cS_2 \Rightarrow cgS_1 \Rightarrow cgcS_2 \Rightarrow cgcgS_1 \Rightarrow cgcg$$

- The **language** generated by G, denoted $\mathcal{L}(G)$, is all the strings that can be **derived** from S, $\{w | S \overset{*}{\Rightarrow} w\}$.
- A string is **accepted** by the grammar if there exist a derivation of the string from S.

Transformational grammars: **definitions**

- A small example, a grammar denoted by G

$$
\begin{array}{lcccc}
S & \rightarrow & gS_1 & | & cS_2 \\
S_1 & \rightarrow & cS_2 & | & \epsilon \\
S_2 & \rightarrow & gS_1 & | & \epsilon
\end{array}
$$

- A **derivation** is the successive application of the rules starting with S (the start nonterminal).

$$S \Rightarrow cS_2 \Rightarrow cgS_1 \Rightarrow cgcS_2 \Rightarrow cgcgS_1 \Rightarrow cgcg$$

- The **language** generated by G, denoted $\mathcal{L}(G)$, is all the strings that can be **derived** from S, $\{w|S \stackrel{*}{\Rightarrow} w\}$.
- A string is **accepted** by the grammar if there exist a derivation of the string from S.

Transformational grammars: **definitions**

- A small example, a grammar denoted by G

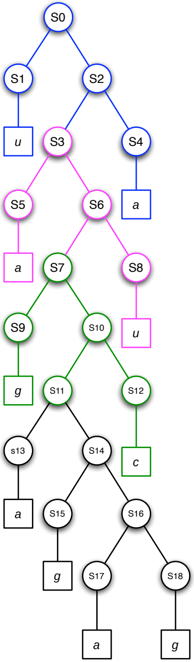
$$\begin{array}{lcccc} S & \rightarrow & gS_1 & \mid & cS_2 \\ S_1 & \rightarrow & cS_2 & \mid & \epsilon \\ S_2 & \rightarrow & gS_1 & \mid & \epsilon \end{array}$$

- A **derivation** is the successive application of the rules starting with S (the start nonterminal).

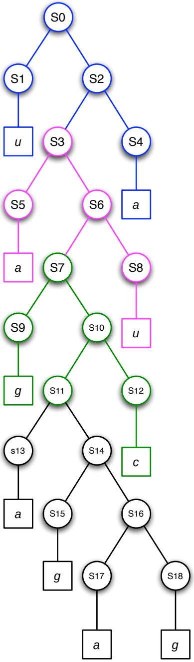
$$S \Rightarrow cS_2 \Rightarrow cgS_1 \Rightarrow cgcS_2 \Rightarrow cgcgS_1 \Rightarrow cgcg$$

- The **language** generated by G, denoted $\mathcal{L}(G)$, is all the strings that can be **derived** from S, $\{w | S \stackrel{\star}{\Rightarrow} w\}$.
- A string is **accepted** by the grammar if there exist a derivation of the string from S.

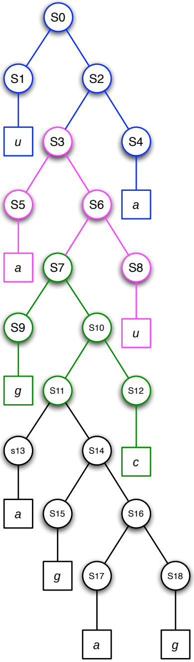
- A derivation can be visualized as a **parse** tree
- Terminals are **leaves** and non-terminals are **internal nodes**
- What was the **input string**?
- Can you **enumerate some of the productions** of the grammar?



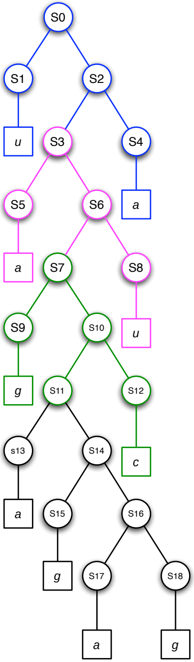
- A derivation can be visualized as a **parse** tree
- Terminals are **leaves** and non-terminals are **internal nodes**
- What was the **input string**?
- Can you **enumerate some of the productions** of the grammar?



- A derivation can be visualized as a **parse** tree
- Terminals are **leaves** and non-terminals are **internal nodes**
- What was the **input string**?
- Can you **enumerate some of the productions** of the grammar?



- A derivation can be visualized as a **parse** tree
- Terminals are **leaves** and non-terminals are **internal nodes**
- What was the **input string**?
- Can you **enumerate some of the productions** of the grammar?



Transformational **grammars**

- A small example

$$
\begin{array}{lclcl}
S & \rightarrow & gS_1 & | & cS_2 \\
S_1 & \rightarrow & cS_2 & | & \epsilon \\
S_2 & \rightarrow & gS_1 & | & \epsilon
\end{array}
$$

- Give examples of sentences accepted (generated) by the grammar.
- Which class of grammar is this?

Transformational **grammars**

- A small example

$$
\begin{array}{rcl|l}
S & \rightarrow & gS_1 & cS_2 \\
S_1 & \rightarrow & cS_2 & \epsilon \\
S_2 & \rightarrow & gS_1 & \epsilon
\end{array}
$$

- Give examples of sentences accepted (generated) by the grammar.
- Which class of grammar is this?

Transformational **grammars**

- A small example

$$\begin{array}{lcccl} S & \rightarrow & gS_1 & | & cS_2 \\ S_1 & \rightarrow & cS_2 & | & \epsilon \\ S_2 & \rightarrow & gS_1 & | & \epsilon \end{array}$$

- Give examples of sentences accepted (generated) by the grammar.
- Which class of grammar is this?

Chomsky hierarchy of transformational grammars

Grammar type	Decidability	Productions
Regular		
Context-free		
Context-sensitive		
Unrestricted		

Chomsky hierarchy of transformational grammars

Grammar type	Decidability	Productions
Regular	finite state automata	
Context-free		
Context-sensitive		
Unrestricted		

Chomsky hierarchy of transformational grammars

Grammar type	Decidability	Productions
Regular	finite state automata	$W \rightarrow aW, W \rightarrow a$
Context-free		
Context-sensitive		
Unrestricted		

Chomsky hierarchy of transformational grammars

Grammar type	Decidability	Productions
Regular	finite state automata	$W \rightarrow aW, W \rightarrow a$
Context-free	push-down automata	
Context-sensitive		
Unrestricted		

Chomsky hierarchy of transformational grammars

Grammar type	Decidability	Productions
Regular	finite state automata	$W \rightarrow aW, W \rightarrow a$
Context-free	push-down automata	$W \rightarrow \gamma$
Context-sensitive		
Unrestricted		

Chomsky hierarchy of transformational grammars

Grammar type	Decidability	Productions
Regular	finite state automata	$W \to aW, W \to a$
Context-free	push-down automata	$W \to \gamma$
Context-sensitive	linear bounded automata	
Unrestricted		

Chomsky hierarchy of transformational grammars

Grammar type	**Decidability**	**Productions**
Regular	finite state automata	$W \rightarrow aW, W \rightarrow a$
Context-free	push-down automata	$W \rightarrow \gamma$
Context-sensitive	linear bounded automata	$\alpha W \beta \rightarrow \alpha \gamma \beta$
Unrestricted		

Chomsky hierarchy of transformational grammars

Grammar type	Decidability	Productions
Regular	finite state automata	$W \rightarrow aW, W \rightarrow a$
Context-free	push-down automata	$W \rightarrow \gamma$
Context-sensitive	linear bounded automata	$\alpha W \beta \rightarrow \alpha \gamma \beta$
Unrestricted	Turing machines	

Chomsky hierarchy of transformational grammars

Grammar type	Decidability	Productions
Regular	finite state automata	$W \rightarrow aW, W \rightarrow a$
Context-free	push-down automata	$W \rightarrow \gamma$
Context-sensitive	linear bounded automata	$\alpha W \beta \rightarrow \alpha \gamma \beta$
Unrestricted	Turing machines	$\alpha \rightarrow \beta$

Prosite

- N-glycosylation site n-{p}-[st]-{p}

Prosite

- N-glycosylation site `n-{p}-[st]-{p}`

Prosite

- N-glycosylation site `n-{p}-[st]-{p}`
  $S_0 \rightarrow nS_1$
  $S_1 \rightarrow aS_2|cS_2|\ldots|yS_2$
  $S_2 \rightarrow sS_3|tS_3$
  $S_1 \rightarrow a|c|\ldots|y$

Prosite

- N-glycosylation site `n-{p}-[st]-{p}`
  $S_0 \rightarrow nS_1$
  $S_1 \rightarrow aS_2|cS_2|\ldots|yS_2$
  $S_2 \rightarrow sS_3|tS_3$
  $S_1 \rightarrow a|c|\ldots|y$
- What type of grammar is that?

Prosite

- N-glycosylation site `n-{p}-[st]-{p}`
  $S_0 \rightarrow nS_1$
  $S_1 \rightarrow aS_2|cS_2|\ldots|yS_2$
  $S_2 \rightarrow sS_3|tS_3$
  $S_1 \rightarrow a|c|\ldots|y$
- What type of grammar is that?
- www.expasy.ch/prosite

RNA **secondary structure**

- Write a grammar whose language consists of **all** the sequences folding into either of the following two stem-loop structures.

```
  G A           A G
 A   G         G   A
  N-N'          N-N'
  N-N'          N-N'
  N-N'          N-N'
```

RNA **secondary structure**

- Write a grammar whose language consists of **all** the sequences folding into either of the following two stem-loop structures.

```
   G A         A G
  A   G       G   A
   N-N'        N-N'
   N-N'        N-N'
   N-N'        N-N'
```

-

$$
\begin{array}{lclclclcl}
S & \rightarrow & aAu & | & cAg & | & gAc & | & uAa \\
A & \rightarrow & aBu & | & cBg & | & gBc & | & uBa \\
B & \rightarrow & aCu & | & cCg & | & gCc & | & uCa \\
C & \rightarrow & agag & | & gaga & & & &
\end{array}
$$

RNA **secondary structure**

- Write a grammar whose language consists of **all** the sequences folding into either of the following two stem-loop structures.

```
  G A          A G
 A   G        G   A
  N-N'         N-N'
  N-N'         N-N'
  N-N'         N-N'
```

- $$\begin{array}{lcllll} S & \rightarrow & aAu & \mid cAg & \mid gAc & \mid uAa \\ A & \rightarrow & aBu & \mid cBg & \mid gBc & \mid uBa \\ B & \rightarrow & aCu & \mid cCg & \mid gCc & \mid uCa \\ C & \rightarrow & agag & \mid gaga & & \end{array}$$

- What type of grammar is that?

Cocke-Younger-Kasami (CYK) algorithm

- CYK is a widely used algorithm for the parsing of **context-free grammars** (CFG)

Cocke-Younger-Kasami (CYK) algorithm

- CYK is a widely used algorithm for the parsing of **context-free grammars** (CFG)
- The CFG must be first transformed into its **Chomsky normal form** (CNF)

Cocke-Younger-Kasami (CYK) algorithm

- CYK is a widely used algorithm for the parsing of **context-free grammars** (CFG)
- The CFG must be first transformed into its **Chomsky normal form** (CNF)
- All the productions must be of the form:

Cocke-Younger-Kasami (CYK) algorithm

- CYK is a widely used algorithm for the parsing of **context-free grammars** (CFG)
- The CFG must be first transformed into its **Chomsky normal form** (CNF)
- All the productions must be of the form:
  - $A \rightarrow BC$ (**exactly two nonterminals**) or

Cocke-Younger-Kasami (CYK) algorithm

- CYK is a widely used algorithm for the parsing of **context-free grammars** (CFG)
- The CFG must be first transformed into its **Chomsky normal form** (CNF)
- All the productions must be of the form:
  - $A \rightarrow BC$ (**exactly two nonterminals**) or
  - $A \rightarrow a$ (**exactly one terminal**)

Cocke-Younger-Kasami (**CYK**) algorithm

$$S \rightarrow g\ T\ c$$

Cocke-Younger-Kasami (**CYK**) algorithm

$$S \rightarrow S_1 \; S_2$$

$$S \rightarrow g \, T \, c$$

Cocke-Younger-Kasami (**CYK**) algorithm

$$S \rightarrow g\,T\,c$$

$$\begin{aligned} S &\rightarrow S_1\ S_2 \\ S_1 &\rightarrow g \end{aligned}$$

Cocke-Younger-Kasami (**CYK**) algorithm

$$S \rightarrow g\, T\, c$$

$$\begin{aligned} S &\rightarrow S_1 \; S_2 \\ S_1 &\rightarrow g \\ S_2 &\rightarrow T \; S_4 \end{aligned}$$

Cocke-Younger-Kasami (**CYK**) algorithm

$$S \rightarrow g\,T\,c$$

$$\begin{aligned} S &\rightarrow S_1 \; S_2 \\ S_1 &\rightarrow g \\ S_2 &\rightarrow T \; S_4 \\ S_4 &\rightarrow c \end{aligned}$$

Cocke-Younger-Kasami (**CYK**) algorithm

$$S \rightarrow g\,T\,c$$

$$\begin{aligned} S &\rightarrow S_1 \; S_2 \\ S_1 &\rightarrow g \\ S_2 &\rightarrow T \; S_4 \\ S_4 &\rightarrow c \end{aligned}$$

Cocke-Younger-Kasami (**CYK**) algorithm

Write a **CFG** in **CNF** for the following stem-loop structure.

```
  G A
A     G
  G-C
  A-U
  U-A
```

Cocke-Younger-Kasami (**CYK**) algorithm

Write a **CFG** in **CNF** for the following stem-loop structure.

```
   G A
 A     G
   G-C
   A-U
   U-A
```

$$S \rightarrow S_1 S_2$$

Cocke-Younger-Kasami (**CYK**) algorithm

Write a **CFG** in **CNF** for the following stem-loop structure.

```
  G A
 A   G
  G-C
  A-U
  U-A
```

$$S \rightarrow S_1 S_2$$
$$S_1 \rightarrow u$$

Cocke-Younger-Kasami (**CYK**) algorithm

Write a **CFG** in **CNF** for the following stem-loop structure.

```
  G A
A     G
  G-C
  A-U
  U-A
```

$$S \rightarrow S_1 S_2$$
$$S_1 \rightarrow u$$
$$S_2 \rightarrow S_3 S_4$$

Cocke-Younger-Kasami (**CYK**) algorithm

Write a **CFG** in **CNF** for the following stem-loop structure.

```
  G A
 A   G
  G-C
  A-U
  U-A
```

$$
\begin{aligned}
S &\rightarrow S_1 S_2 \\
S_1 &\rightarrow u \\
S_2 &\rightarrow S_3 S_4 \\
S_4 &\rightarrow a
\end{aligned}
$$

Cocke-Younger-Kasami (**CYK**) algorithm

Write a **CFG** in **CNF** for the following stem-loop structure.

```
  G A
 A   G
  G-C
  A-U
  U-A
```

$$
\begin{aligned}
S &\rightarrow S_1 S_2 \\
S_1 &\rightarrow u \\
S_2 &\rightarrow S_3 S_4 \\
S_4 &\rightarrow a \\
S_3 &\rightarrow S_5 S_6
\end{aligned}
$$

Cocke-Younger-Kasami (**CYK**) algorithm

Write a **CFG** in **CNF** for the following stem-loop structure.

```
  G A
A     G
  G-C
  A-U
  U-A
```

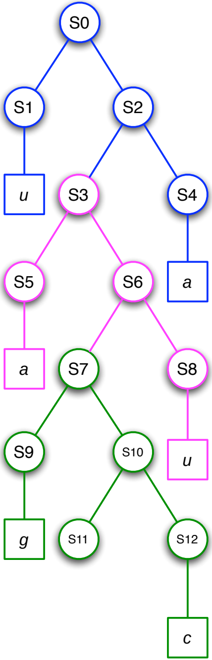
$$
\begin{aligned}
S &\rightarrow S_1 S_2 & \qquad S_5 &\rightarrow a \\
S_1 &\rightarrow u & S_6 &\rightarrow S_7 S_8 \\
S_2 &\rightarrow S_3 S_4 & S_8 &\rightarrow u \\
S_4 &\rightarrow a & S_7 &\rightarrow S_9 S_{10} \\
S_3 &\rightarrow S_5 S_6 & S_9 &\rightarrow g
\end{aligned}
$$

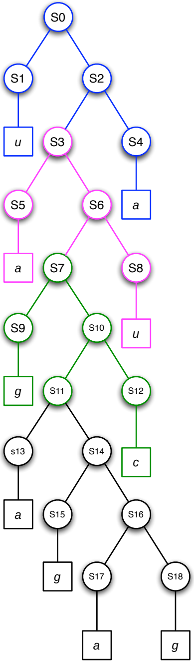
Cocke-Younger-Kasami (**CYK**) algorithm

Write a **CFG** in **CNF** for the following stem-loop structure.

```
 G A
A   G
 G-C
 A-U
 U-A
```

$$
\begin{array}{lll lll lll lll}
S & \rightarrow & S_1 S_2 & S_5 & \rightarrow & a & S_{10} & \rightarrow & S_{11} S_{12} & S_{15} & \rightarrow & g \\
S_1 & \rightarrow & u & S_6 & \rightarrow & S_7 S_8 & S_{12} & \rightarrow & c & S_{16} & \rightarrow & S_{17} S_{18} \\
S_2 & \rightarrow & S_3 S_4 & S_8 & \rightarrow & u & S_{11} & \rightarrow & S_{13} S_{14} & S_{17} & \rightarrow & a \\
S_4 & \rightarrow & a & S_7 & \rightarrow & S_9 S_{10} & S_{13} & \rightarrow & a & S_{18} & \rightarrow & g \\
S_3 & \rightarrow & S_5 S_6 & S_9 & \rightarrow & g & S_{14} & \rightarrow & S_{15} S_{16} & & &
\end{array}
$$


S0
S1
S2
u
S3
S4
S5
S6
a
a
S7
S8
S9
S10
u
g
S11
S12
c


S0
S1
S2
u
S3
S4
S5
S6
a
a
S7
S8
S9
S10
u
g
S11
S12
s13
S14
c
a
S15
S16
g
S17
S18
a
g

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar G, let $W \stackrel{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from W of G

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar *G*, let $W \overset{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from *W* of *G*
- Also, let **s** be an input string of length *n*

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar *G*, let $W \stackrel{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from *W* of *G*
- Also, let **s** be an input string of length *n*
- **Remember that G is in Chomsky Normal form!**

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar G, let $W \stackrel{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from W of G
- Also, let **s** be an input string of length n
- **Remember that G is in Chomsky Normal form!**
- Let $V(i, l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i + l - 1]\}$

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar G, let $W \stackrel{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from W of G
- Also, let **s** be an input string of length n
- **Remember that G is in Chomsky Normal form!**
- Let $V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$
- **For** $l = 1$

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar G, let $W \stackrel{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from W of G
- Also, let **s** be an input string of length n
- **Remember that G is in Chomsky Normal form!**
- Let $V(i, l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i + l - 1]\}$
- **For** $l = 1$

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar G, let $W \stackrel{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from W of G
- Also, let **s** be an input string of length n
- **Remember that G is in Chomsky Normal form!**
- Let $V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$
- **For** $l = 1$

$$V(i,1) =$$

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar G, let $W \stackrel{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from W of G
- Also, let $\mathbf{s}$ be an input string of length n
- **Remember that G is in Chomsky Normal form!**
- Let $V(i,l) = \{W|W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$
- **For** $l = 1$

$$V(i,1) = \{\ W|W \rightarrow\ \mathbf{s}[i,i]\ \}$$

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar G, let $W \stackrel{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from W of G
- Also, let **s** be an input string of length n
- **Remember that G is in Chomsky Normal form!**
- Let $V(i, l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i + l - 1]\}$
- **For** $l = 1$

$$V(i, 1) = \{\, W | W \rightarrow \ \mathbf{s}[i, i] \,\}$$

- **For** $l > 1$

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar G, let $W \stackrel{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from W of G
- Also, let **s** be an input string of length n
- **Remember that G is in Chomsky Normal form!**
- Let $V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$
- **For** $l = 1$

$$V(i,1) = \{\ W | W \rightarrow\ \mathbf{s}[i,i]\ \}$$

- **For** $l > 1$

Cocke-Younger-Kasami (CYK) algorithm: **idea**

- For a given grammar G, let $W \stackrel{\star}{\Rightarrow} \alpha$ indicate that the string α can be derived from W of G
- Also, let **s** be an input string of length n
- **Remember that G is in Chomsky Normal form!**
- Let $V(i, l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i + l - 1]\}$
- **For** $l = 1$

$$V(i, 1) = \{\, W | W \rightarrow \ \mathbf{s}[i, i] \,\}$$

- **For** $l > 1$

$$\begin{aligned} V(i, l) = \ \{ A \ | \ & A \rightarrow \ BC, \\ & B \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i + k - 1], \\ & C \stackrel{\star}{\Rightarrow} \mathbf{s}[i + k, i + l - 1], \\ & 1 \leq k < l \,\} \end{aligned}$$

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{rcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

$$V(i, l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i + l - 1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$\begin{array}{lcl} S & \rightarrow & AB \mid BC \\ A & \rightarrow & BA \mid a \\ B & \rightarrow & CC \mid b \\ C & \rightarrow & AB \mid a \end{array}$$

$$V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

$$V(i, l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i + l - 1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{llll}
S & \rightarrow & AB & | & BC \\
A & \rightarrow & BA & | & a \\
B & \rightarrow & CC & | & b \\
C & \rightarrow & AB & | & a
\end{array}
$$

$$V(i, l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i + l - 1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{lcccl}
S & \rightarrow & AB & | & BC \\
A & \rightarrow & BA & | & a \\
B & \rightarrow & CC & | & b \\
C & \rightarrow & AB & | & a
\end{array}
$$

$$V(i, l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

$$V(i,l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

$$V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l=1$	B	A,C	A,C	B	A,C
$l=2$	S,A	B	S,C	S,A	
$l=3$	$\emptyset$	B	B		
$l=4$	$\emptyset$	S,A,C			
$l=5$	S,A,C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{rcl|l}
S & \rightarrow & AB & BC \\
A & \rightarrow & BA & a \\
B & \rightarrow & CC & b \\
C & \rightarrow & AB & a
\end{array}
$$

$$V(i,l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

$$V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$S \rightarrow AB \mid BC$$
$$A \rightarrow BA \mid a$$
$$B \rightarrow CC \mid b$$
$$C \rightarrow AB \mid a$$

$$V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l=1$	B	A, C	A, C	B	A, C
$l=2$	S, A	B	S, C	S, A	
$l=3$	$\emptyset$	B	B		
$l=4$	$\emptyset$	S, A, C			
$l=5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{rcl|l}
S & \rightarrow & AB & BC \\
A & \rightarrow & BA & a \\
B & \rightarrow & CC & b \\
C & \rightarrow & AB & a
\end{array}
$$

$$V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l=1$	B	A, C	A, C	B	A, C
$l=2$	S, A	B	S, C	S, A	
$l=3$	$\emptyset$	B	B		
$l=4$	$\emptyset$	S, A, C			
$l=5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{llll}
S & \rightarrow & AB & | & BC \\
A & \rightarrow & BA & | & a \\
B & \rightarrow & CC & | & b \\
C & \rightarrow & AB & | & a
\end{array}
$$

$$V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$\begin{array}{rcl} S & \rightarrow & AB \mid BC \\ A & \rightarrow & BA \mid a \\ B & \rightarrow & CC \mid b \\ C & \rightarrow & AB \mid a \end{array}$$

$$V(i,l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l=1$	B	A, C	A, C	B	A, C
$l=2$	S, A	B	S, C	S, A	
$l=3$	$\emptyset$	B	B		
$l=4$	$\emptyset$	S, A, C			
$l=5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

$$V(i,l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l=1$	B	A, C	A, C	B	A, C
$l=2$	S, A	B	S, C	S, A	
$l=3$	$\emptyset$	B	B		
$l=4$	$\emptyset$	S, A, C			
$l=5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

$$V(i,l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l=1$	B	A, C	A, C	B	A, C
$l=2$	S, A	B	S, C	S, A	
$l=3$	$\emptyset$	B	B		
$l=4$	$\emptyset$	S, A, C			
$l=5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{lcl|l}
S & \rightarrow & AB & BC \\
A & \rightarrow & BA & a \\
B & \rightarrow & CC & b \\
C & \rightarrow & AB & a
\end{array}
$$

$$V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l=1$	B	A, C	A, C	B	A, C
$l=2$	S, A	B	S, C	S, A	
$l=3$	$\emptyset$	B	B		
$l=4$	$\emptyset$	S, A, C			
$l=5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{rcl|l}
S & \rightarrow & AB & BC \\
A & \rightarrow & BA & a \\
B & \rightarrow & CC & b \\
C & \rightarrow & AB & a
\end{array}
$$

$$V(i, l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i + l - 1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{lclcl}
S & \rightarrow & AB & | & BC \\
A & \rightarrow & BA & | & a \\
B & \rightarrow & CC & | & b \\
C & \rightarrow & AB & | & a
\end{array}
$$

$$V(i,l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

Cocke-Younger-Kasami (CYK) algorithm: **example**

$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

$$V(i,l) = \{W | W \stackrel{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

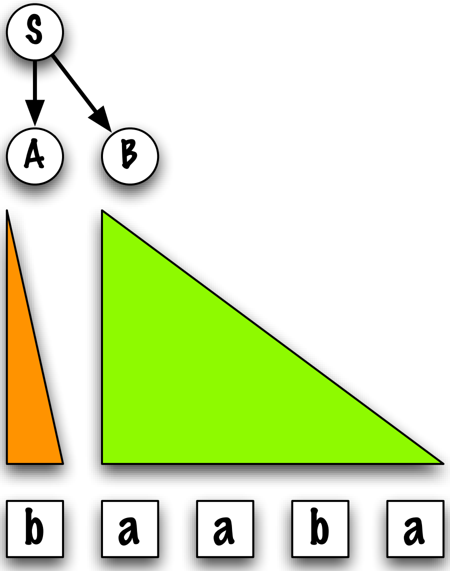
s	b	a	a	b	a
i	1	2	3	4	5
$l=1$	B	A,C	A,C	B	A,C
$l=2$	S,A	B	S,C	S,A	
$l=3$	$\emptyset$	B	B		
$l=4$	$\emptyset$	S,A,C			
$l=5$	S,A,C				

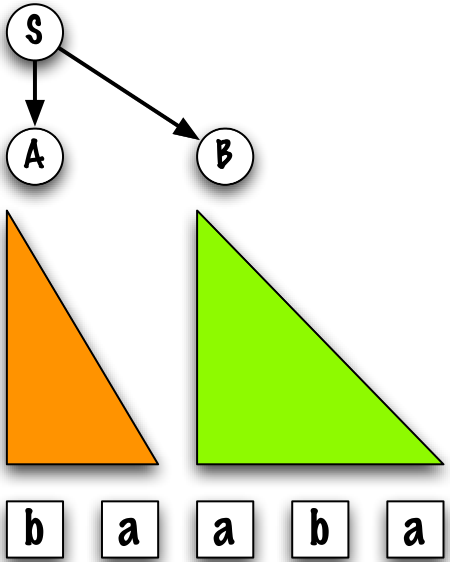
Cocke-Younger-Kasami (CYK) algorithm: **example**

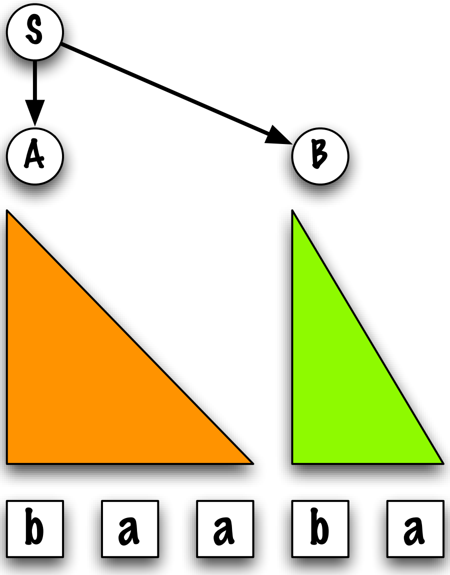
$$
\begin{array}{lcl|l}
S & \rightarrow & AB & BC \\
A & \rightarrow & BA & a \\
B & \rightarrow & CC & b \\
C & \rightarrow & AB & a
\end{array}
$$

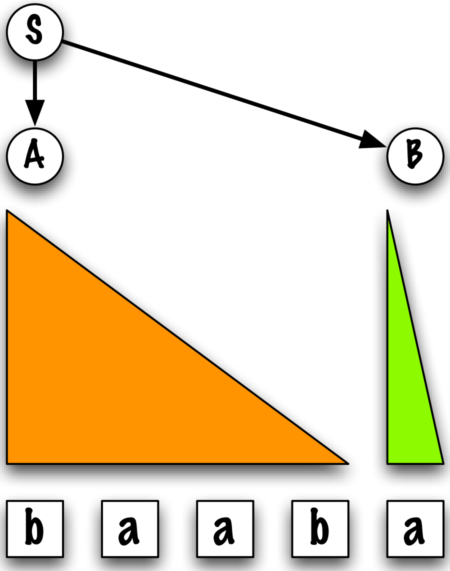
$$V(i,l) = \{W | W \overset{\star}{\Rightarrow} \mathbf{s}[i, i+l-1]\}$$

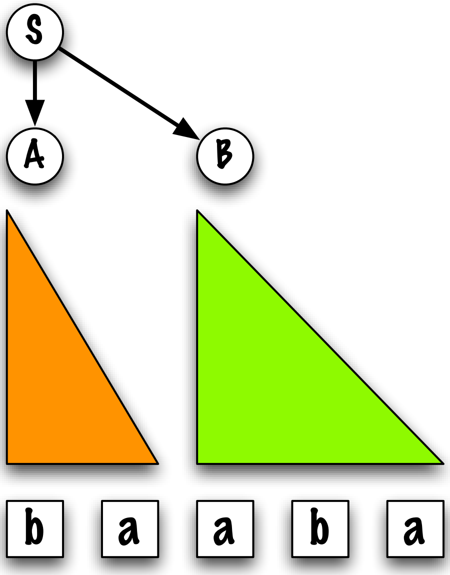
s	b	a	a	b	a
i	1	2	3	4	5
$l = 1$	B	A, C	A, C	B	A, C
$l = 2$	S, A	B	S, C	S, A	
$l = 3$	$\emptyset$	B	B		
$l = 4$	$\emptyset$	S, A, C			
$l = 5$	S, A, C				

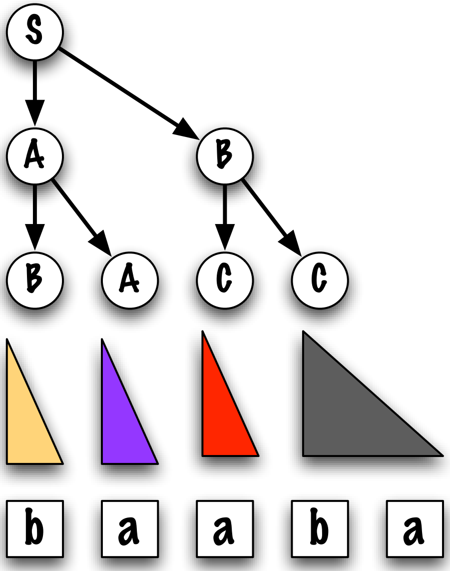

S
A
B
b
a
a
b
a

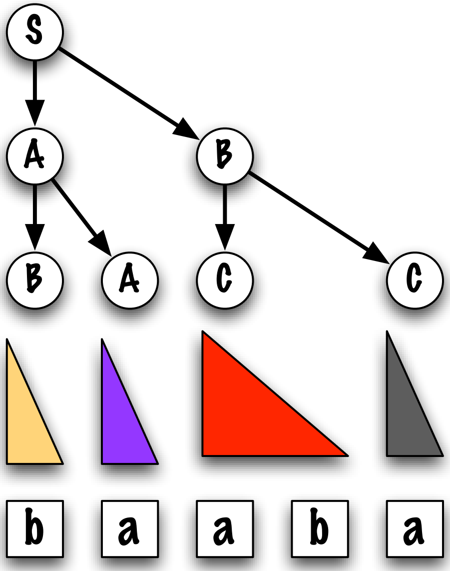

S
A
B
b
a
a
b
a

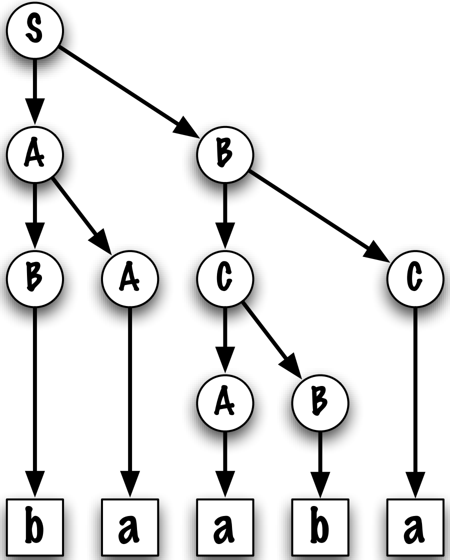

S
A
B
b
a
a
b
a


S
A
B
b
a
a
b
a


S
A
B
b
a
a
b
a


S
A
B
B
A
C
C
b
a
a
b
a


S
A
B
B
A
C
C
b
a
a
b
a


S
A
B
B
A
C
C
A
B
b
a
a
b
a

Cocke-Younger-Kasami (CYK) algorithm: **algorithm**

```
{ Initialization }

for i = 1 to n do
    V(i,1) = {A | A → a is a production and s[i] = a}

{ Iteration }

for l = 2 to n do
  for i = 1 to n - l + 1 do
    V(i,l) = ∅
    for k = 1 to l - 1 do
        V(i,l) = V(i,l)  ∪
                 {A | A → BC,
                      B ∈ V(i,k) and
                      C ∈ V(i+k,l-k)}
```

Given an input of size n and grammar having m nonterminal symbols, CYK runs in $\mathcal{O}(mn^2)$ space and $\mathcal{O}(m^2n^3)$ time.

Cocke-Younger-Kasami (CYK) algorithm: **algorithm**

```
{ Initialization }

for i = 1 to n do
    V(i,1) = {A | A → a is a production and s[i] = a}

{ Iteration }

for l = 2 to n do
  for i = 1 to n - l + 1 do
    V(i,l) = ∅
    for k = 1 to l - 1 do
        V(i,l) = V(i,l)  ∪
                      {A | A → BC,
                           B ∈ V(i,k) and
                           C ∈ V(i+k,l-k)}
```

Given an input of size n and grammar having m nonterminal symbols, CYK runs in $\mathcal{O}(mn^2)$ space and $\mathcal{O}(m^2n^3)$ time.

Cocke-Younger-Kasami (CYK) algorithm: **remarks**

- An **RNA secondary structure** (motif) can be represented as a CFG (in CNF)
- CYK can be used for **finding all** its occurrences in a database
- CYK finds an **exact match**
- **Still hit-or-miss algorithm**
- **Gene Myers adapted the algorithm for finding approximate matches**

Cocke-Younger-Kasami (CYK) algorithm: **remarks**

- An **RNA secondary structure** (motif) can be represented as a CFG (in CNF)
- CYK can be used for **finding all** its occurrences in a database
- CYK finds an **exact match**
- **Still hit-or-miss algorithm**
- **Gene Myers adapted the algorithm for finding approximate matches**

Cocke-Younger-Kasami (CYK) algorithm: **remarks**

- An **RNA secondary structure** (motif) can be represented as a CFG (in CNF)
- CYK can be used for **finding all** its occurrences in a database
- CYK finds an **exact match**
- **Still hit-or-miss algorithm**
- **Gene Myers adapted the algorithm for finding approximate matches**

Cocke-Younger-Kasami (CYK) algorithm: **remarks**

- An **RNA secondary structure** (motif) can be represented as a CFG (in CNF)
- CYK can be used for **finding all** its occurrences in a database
- CYK finds an **exact match**
- **Still hit-or-miss algorithm**
- **Gene Myers adapted the algorithm for finding approximate matches**

Cocke-Younger-Kasami (CYK) algorithm: **remarks**

- An **RNA secondary structure** (motif) can be represented as a CFG (in CNF)
- CYK can be used for **finding all** its occurrences in a database
- CYK finds an **exact match**
- **Still hit-or-miss algorithm**
- **Gene Myers adapted the algorithm for finding approximate matches**

Discussion

A	C	C	U
A	C	U	U
A	C	C	U
G	C	C	C
(	.	.	)
A	U	U	U
A	C	C	U
G	C	U	C

- *AUUU* is not accepted
- *ACCU* and *GCUC* are both accepted, but one is the **consensus** and the other the **exception**

Discussion

A	C	C	U
A	C	U	U
A	C	C	U
G	C	C	C
(	.	.	)
A	U	U	U
A	C	C	U
G	C	U	C

- *AUUU* is not accepted
- *ACCU* and *GCUC* are both accepted, but one is the **consensus** and the other the **exception**

Discussion

A	C	C	U
A	C	U	U
A	C	C	U
G	C	C	C
(	.	.	)
A	U	U	U
A	C	C	U
G	C	U	C

- *AUUU* is not accepted
- *ACCU* and *GCUC* are both accepted, but one is the **consensus** and the other the **exception**

Stochastic (Context-Free) grammars

- Because of their discrete nature, it's difficult to design patterns that **1)** are **specific enough 2)** and yet will be **general enough** to match unseen cases
- Any grammar in the Chomsky hierarchy can be transformed into a **probabilistic model**
- In practice, because the cost of parsing a string (sequence or database) using context-sensitive and unrestricted grammars is prohibitive, applications are restricted to **regular** and **context-free** grammars

Stochastic (Context-Free) grammars

- Because of their discrete nature, it's difficult to design patterns that **1)** are **specific enough 2)** and yet will be **general enough** to match unseen cases
- Any grammar in the Chomsky hierarchy can be transformed into a **probabilistic model**
- In practice, because the cost of parsing a string (sequence or database) using context-sensitive and unrestricted grammars is prohibitive, applications are restricted to **regular** and **context-free** grammars

Stochastic (Context-Free) grammars

- Because of their discrete nature, it's difficult to design patterns that **1)** are **specific enough 2)** and yet will be **general enough** to match unseen cases
- Any grammar in the Chomsky hierarchy can be transformed into a **probabilistic model**
- In practice, because the cost of parsing a string (sequence or database) using context-sensitive and unrestricted grammars is prohibitive, applications are restricted to **regular** and **context-free** grammars

Stochastic grammars

A **stochastic context-free grammar** (SCFG) for an RNA will have production rules of the following forms:

$$S_0 \rightarrow (.25) : g\, S_1\, c \mid (.25) : c\, S_1\, g \mid (.25) : a\, S_1\, u \mid (.25) : u\, S_1\, a$$

to represent base-pairs, and

$$S_i \rightarrow (.50) : u\, S_j \mid (.50) : g\, S_j$$

to represent single stranded regions.

Stochastic grammars: **problems**

- Given a sequence finding the most likely parse (**alignment**)
- Probability that this SCFG produces that sequence (**scoring**)
- Estimating the probabilities of the model (**training**)

Stochastic grammars: **problems**

- Given a sequence finding the most likely parse (**alignment**)
- Probability that this SCFG produces that sequence (**scoring**)
- Estimating the probabilities of the model (**training**)

Stochastic grammars: **problems**

- Given a sequence finding the most likely parse (**alignment**)
- Probability that this SCFG produces that sequence (**scoring**)
- Estimating the probabilities of the model (**training**)

Notation

- Given an SCFG in Chomsky normal form with M nonterminal symbols, $W = W_1, .., W_m$ and W_1 the start symbol
- Let v, w and z denote the indices for the nonterminal symbols, W_v, W_y and W_z
- Production rules are of the form:

$$W_v \rightarrow W_y W_z \text{ and } W_v \rightarrow a$$

- Let the probability parameters be called,

$$t_v(y, z)$$

  for *transitions* and

$$e_v(a)$$

  for *emissions*
- Finally, let i, j and k be the indices for the symbols x_i, x_j and x_k in the sequence x of length n

Notation

- Given an SCFG in Chomsky normal form with M nonterminal symbols, $W = W_1, .., W_m$ and W_1 the start symbol
- Let v, w and z denote the indices for the nonterminal symbols, W_v, W_y and W_z
- Production rules are of the form:

$$W_v \rightarrow W_y W_z \text{ and } W_v \rightarrow a$$

- Let the probability parameters be called,

$$t_v(y, z)$$

  for *transitions* and

$$e_v(a)$$

  for *emissions*
- Finally, let i, j and k be the indices for the symbols x_i, x_j and x_k in the sequence x of length n

Notation

- Given an SCFG in Chomsky normal form with M nonterminal symbols, $W = W_1, .., W_m$ and W_1 the start symbol
- Let v, w and z denote the indices for the nonterminal symbols, W_v, W_y and W_z
- Production rules are of the form:

$$W_v \rightarrow W_y W_z \text{ and } W_v \rightarrow a$$

- Let the probability parameters be called,

$$t_v(y, z)$$

for *transitions* and

$$e_v(a)$$

for *emissions*
- Finally, let i, j and k be the indices for the symbols x_i, x_j and x_k in the sequence x of length n

Notation

- Given an SCFG in Chomsky normal form with M nonterminal symbols, $W = W_1, .., W_m$ and W_1 the start symbol
- Let v, w and z denote the indices for the nonterminal symbols, W_v, W_y and W_z
- Production rules are of the form:

$$W_v \rightarrow W_y W_z \text{ and } W_v \rightarrow a$$

- Let the probability parameters be called,

$$t_v(y, z)$$

  for *transitions* and

$$e_v(a)$$

  for *emissions*
- Finally, let i, j and k be the indices for the symbols x_i, x_j and x_k in the sequence x of length n

Notation

- Given an SCFG in Chomsky normal form with M nonterminal symbols, $W = W_1, .., W_m$ and W_1 the start symbol
- Let v, w and z denote the indices for the nonterminal symbols, W_v, W_y and W_z
- Production rules are of the form:

$$W_v \rightarrow W_y W_z \text{ and } W_v \rightarrow a$$

- Let the probability parameters be called,

$$t_v(y, z)$$

for *transitions* and

$$e_v(a)$$

for *emissions*
- Finally, let i, j and k be the indices for the symbols x_i, x_j and x_k in the sequence x of length n

CYK algorithm (**alignment**)

```
{ Initialization }
```

for $i = 1$ to n, $v = 1$ to M

$$\gamma(i, 1, v) = e_v(x_i)$$

```
{ Iteration }
```

for $l = 2$ to n, $i = 1$ to $n - l + 1$, $v = 1$ to M

$$\gamma(i, l, v) = \max_{y,z} \max_{k=1,\ldots,l-1} \{\gamma(i, k, y)\gamma(i + k, l - k, z)t_v(y, z)\}$$

```
{ Termination }
```

$$\log P(x, \hat{\pi}|\theta) = \gamma(1, n, 1).$$

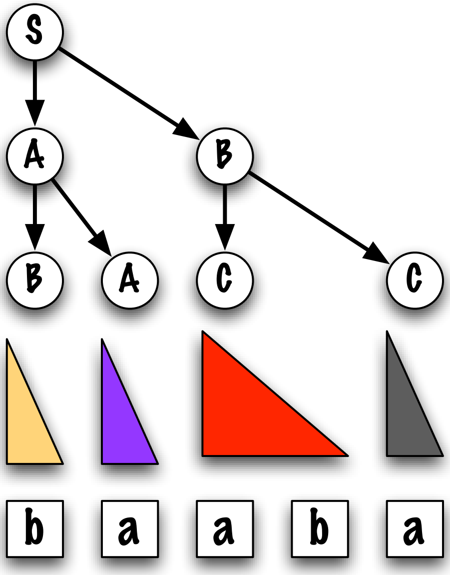
Cocke-Younger-Kasami (CYK) algorithm: **non-probabilistic**

```
{ Initialization }

for i = 1 to n do
    V(i,1) = {A | A → a is a production and s[i] = a}

{ Iteration }

for l = 2 to n do
    for i = 1 to n - l + 1 do
        V(i,l) = ∅
        for k = 1 to l - 1 do
            V(i,l) = V(i,l)  ∪
                      {A | A → BC, B ∈ V(i,k) and C ∈ V(i+k
```


S
A
B
B
A
C
C
b
a
a
b
a

CYK algorithm: **probabilistic**

```
{ Initialization }
```

for $i = 1$ to n, $v = 1$ to M

$$\gamma(i, 1, v) \quad = \quad log \;\; e_v(x_i)$$

```
{ Iteration }
```

for $l = 2$ to n, $i = 1$ to $n - l + 1$, $v = 1$ to M

$$\gamma(i, j, v) \quad = \quad \max_{y,z} \;\; \max_{k=1,\ldots,l-1} \;\; \{\gamma(i, k, y) + \gamma(i+k, l-k, z) + \log \;\; t_v(y, z)\}$$

```
{ Termination }
```

$$\log \;\; P(x, \hat{\pi} | \theta) \quad = \quad \gamma(1, n, 1).$$

Complexity

Memory $O(L^2M)$
Time $O(L^3M^3)$

CYK algorithm: inside (scoring)

{ Initialization }

for $i = 1$ to n, $v = 1$ to M

$$\alpha(i, 1, v) = e_v(x_i)$$

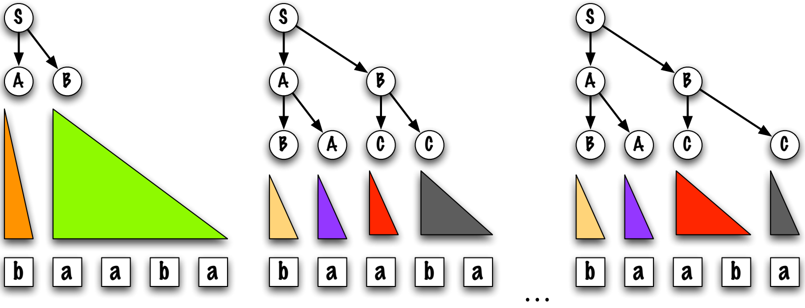
{ Iteration }

for $l = 2$ to n, $i = 1$ to $n - l + 1$, $v = 1$ to M

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

{ Termination }

$$\log P(x|\theta) = \alpha(1, n, 1).$$


S
A
B
b
a
a
b
a
S
A
B
B
A
C
C
b
a
a
b
a
...
S
A
B
B
A
C
C
b
a
a
b
a

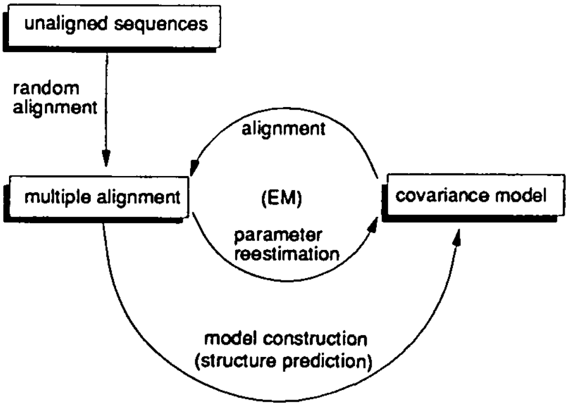
Estimating the probabilities

The transition and emission probabilities are estimated from the user input data (alignment and structure).

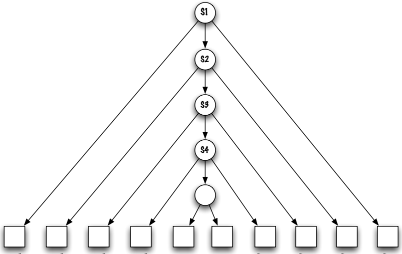
- **In theory:**
  - The **inside-outside**, an iterative **expectation-maximization** (EM), algorithm can be used for parameter re-estimation
- **In practice:**
  - Parameters are extracted from a user input alignment

Expectation-Maximization (EM)

Iterative algorithm for finding the **maximum-likelihood** estimates of the parameters.



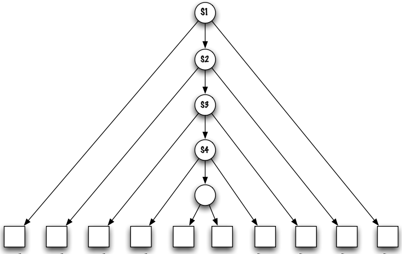
Estimating the parameters




```
( ( ( ( . . ) ) ) )
G G A G A U C U C C
G G G G A - C C C C
U G G G A A C C C A
G G G G A U C C C U
G G G G A A C C C C
```

$$S \xrightarrow{0.8} g\, S_1\, c$$
$$S \xrightarrow{0.2} u\, S_1\, a$$
$$S_1 \xrightarrow{1.0} g\, S_2\, c$$
$$S_2 \xrightarrow{0.8} g\, S_3\, c$$
$$S_2 \xrightarrow{0.2} a\, S_3\, u$$
$$S_3 \xrightarrow{1.0} g\, S_4\, c$$
$$S_4 \xrightarrow{1.0} S_5 S_6$$
$$S_5 \xrightarrow{1.0} a$$
$$S_6 \xrightarrow{0.4} a$$
$$S_6 \xrightarrow{0.4} u$$
$$S_6 \xrightarrow{0.2} \epsilon$$

Estimating the parameters




```
( ( ( ( . . ) ) ) )
G G A G A U C U C C
G G G G A - C C C C
U G G G A A C C C A
G G G G A U C C C U
G G G G A A C C C C
```

$$S \xrightarrow{0.8} g\, S_1\, c$$
$$S \xrightarrow{0.2} u\, S_1\, a$$
$$S_1 \xrightarrow{1.0} g\, S_2\, c$$
$$S_2 \xrightarrow{0.8} g\, S_3\, c$$
$$S_2 \xrightarrow{0.2} a\, S_3\, u$$
$$S_3 \xrightarrow{1.0} g\, S_4\, c$$
$$S_4 \xrightarrow{1.0} S_5 S_6$$
$$S_5 \xrightarrow{1.0} a$$
$$S_6 \xrightarrow{0.4} a\, S_6$$
$$S_6 \xrightarrow{0.4} u\, S_6$$
$$S_6 \xrightarrow{0.2} \epsilon$$

tRNA: a more **realistic input**

```
# STOCKHOLM 1.0
#=GF AU    Koala

DA0260           GGGCGAAUAGUGUCAGC.GGGAGCACACCAGACUUGCAUCUGGUAG.GGAGGGUUCGAGUCCCUCUUUGUCCAC
#=GR DA0260 SS   (((((((..((((.........)))).(((((......))))).....(((((.......))))))))))))..
DA0261           GGGCGAAUAGUGUCAGC.GGGAGCACACCAGACUUGCAUCUGGUAG.GGAGGGUUCGAGUCCCUCUUUGUCCAC
#=GR DA0261 SS   (((((((..((((.........)))).(((((......))))).....(((((.......))))))))))))..
DA0340           GGGCUCGUAGCUCAGC..GGGAGAGCGCCGCCUUUGCAGGCGGAGGCCGCGGGUUCAAAUCCCGCCGAGUCCA.
#=GR DA0340 SS   (((((((..((((.........)))).(((((......))))).....(((((.......))))))))))))..
DA0380           GGGCCCAUAGCUCAGU..GGUAGAGUGCCUCCUUUGCAGGAGGAUGCCCUGGGUUCGAAUCCCAGUGGGUCCA.
#=GR DA0380 SS   (((((((..((((.........)))).(((((......))))).....(((((.......))))))))))))..
DA0420           GGGCCCAUAGCUCAGU..GGUAGAGUGCCUCCUUUGCAGGAGGAUGCCCUGGGUUGGAAUCCCAGUGGGUCCA.
#=GR DA0420 SS   (((((((..((((.........)))).(((((......))))).....(((((.......))))))))))))..
DA0580           GGGCCCGUAGCUCAGACUGGGAGAGCGCCGCCCUUGCAGGCGGAGGCCCCGGGUUCAAAUCCCGGUGGGUCCA.
#=GR DA0580 SS   (((((((..((((.........)))).(((((......))))).....(((((.......))))))))))))..
DA0620           GGGCCCGUAGCUCAGACUGGGAGAGCGCCGCCCUUGCAGGCGGAGGCCCCGGGUUCAAAUCCCGGUGGGUCCA.
#=GR DA0620 SS   (((((((..((((.........)))).(((((......))))).....(((((.......))))))))))))..
```

Stochastic Context-Free Grammars (**SCFG**)

Sean Eddy, one of the pioneers of the use of **SCFG**s in bioinformatics, has developed several tools: `http://eddylab.org/software.html`

- **RSEARCH** aligns an RNA query to target sequences, using SCFG algorithms to score both secondary structure and primary sequence alignment simultaneously;
- **Infernal**. RNA structure analysis using covariance models (new);
- **COVE**. RNA structure analysis using covariance models (old).

RSearch

- **Input**: an RNA sequence and its secondary structure
- **Output**: similar RNAs on the basis of both primary sequence and secondary structure
- R.J. Klein and S.R. Eddy (2003) RSEARCH: Finding homologs of single structured RNA sequences. *BMC Bioinformatics*, **4**:44, 2003 (doi:10.1186/1471-2105-4-44)

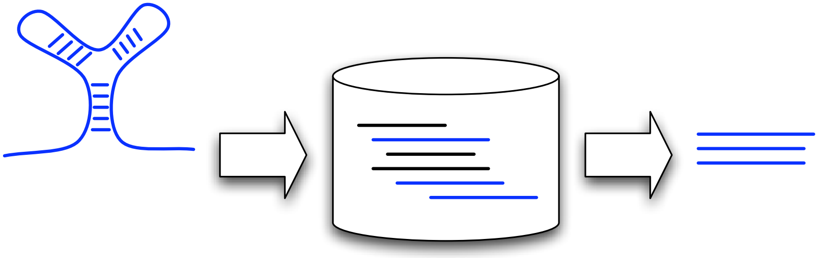
RSearch

- **Input**: an RNA sequence and its secondary structure
- **Output**: similar RNAs on the basis of both primary sequence and secondary structure
- R.J. Klein and S.R. Eddy (2003) RSEARCH: Finding homologs of single structured RNA sequences. *BMC Bioinformatics*, **4**:44, 2003 (doi:10.1186/1471-2105-4-44)

RSearch

- **Input**: an RNA sequence and its secondary structure
- **Output**: similar RNAs on the basis of both primary sequence and secondary structure
- R.J. Klein and S.R. Eddy (2003) RSEARCH: Finding homologs of single structured RNA sequences. *BMC Bioinformatics*, **4**:44, 2003 (doi:10.1186/1471-2105-4-44)

RSearch



```
# STOCKHOLM 1.0
#=GS Holley DE tRNA-Ala that Holley sequenced from Yeast genome
Holley
   GGGCGTGTGGCGTAGTCGGTAGCGCGCTCCCTTAGCATGGGAGAGGtCTCCGGTTCGATTCCGGACTCGTCCA
#=GR Holley SS
   (((((.(..((((........)))).(((((.......))))).....(((((.......)))))).))))).
//
```

Remarks

- **RIBOSUM** substitution matrices (analogous to residue substitution scores such as PAM and BLOSUM but for base pairs)
- Reports the statistical significance of all the matches
- Execution time is $\mathcal{O}(NM^3)$ where N is the size of the database and M is the length of the input sequence
- "(…) a typical single search of a metazoan genome may take a few thousand CPU hours."

Remarks

- **RIBOSUM** substitution matrices (analogous to residue substitution scores such as PAM and BLOSUM but for base pairs)
- Reports the statistical significance of all the matches
- Execution time is $\mathcal{O}(NM^3)$ where N is the size of the database and M is the length of the input sequence
- "(…) a typical single search of a metazoan genome may take a few thousand CPU hours."

Remarks

- **RIBOSUM** substitution matrices (analogous to residue substitution scores such as PAM and BLOSUM but for base pairs)
- Reports the statistical significance of all the matches
- Execution time is $\mathcal{O}(NM^3)$ where N is the size of the database and M is the length of the input sequence
- "(…) a typical single search of a metazoan genome may take a few thousand CPU hours."

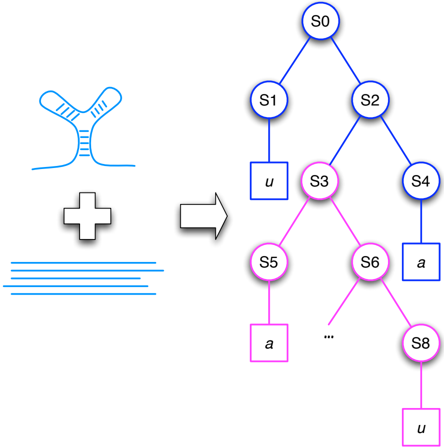
Remarks

- **RIBOSUM** substitution matrices (analogous to residue substitution scores such as PAM and BLOSUM but for base pairs)
- Reports the statistical significance of all the matches
- Execution time is $\mathcal{O}(NM^3)$ where N is the size of the database and M is the length of the input sequence
- “(…) a typical single search of a metazoan genome may take a few thousand CPU hours.”

INFERNAL

- **INFERNAL 1.1**
  Nawrocki, E. P. & Eddy, S. R. Infernal 1.1: 100-fold faster RNA homology searches. *Bioinformatics* **29**, 2933–2935 (2013).
- **Rfam 14** (August 2018, 2791 families, hand curated)
- Kalvari, I. et al. Rfam 13.0: shifting to a genome-centric resource for non-coding RNA families. *Nucleic Acids Res* **46**, D335–D342 (2018).

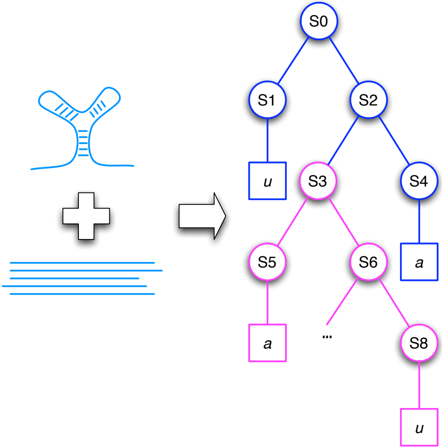
INFERNAL/Rfam covariance models



INFERNAL/Rfam covariance models

```
# STOCKHOLM 1.0

#=GC SS_cons   <<<<..>>>>
seq1           GGAGAUCUCC
seq2           GGGGAUCCCC
seq3           UGGGAACCCA
seq4           GGGGAUCCCU
seq5           GGGGAACCCC
//
```



Summary

- Hard consensus patterns are difficult to design
- SCFGs are powerful but slow
(thousands of hours for scanning a bacterial genome)
- Specialised programs have been developed, each recognising a specific structure; these programs are generally sensitive, specific and (relatively) fast:
  - tRNAscan-SE (by Sean Eddy)
  - detects 99% of the known tRNAs
  - with an error rate of 1 false positive per 15 billion nucleotides

Summary

- Hard consensus patterns are difficult to design
- SCFGs are powerful but slow
(thousands of hours for scanning a bacterial genome)
- Specialised programs have been developed, each recognising a specific structure; these programs are generally sensitive, specific and (relatively) fast:
  - tRNAscan-SE (by Sean Eddy)
  - detects 99% of the known tRNAs
  - with an error rate of 1 false positive per 15 billion nucleotides

Summary

- Hard consensus patterns are difficult to design
- SCFGs are powerful but slow
(thousands of hours for scanning a bacterial genome)
- Specialised programs have been developed, each recognising a specific structure; these programs are generally sensitive, specific and (relatively) fast:
    - tRNAscan-SE (by Sean Eddy)
    - detects 99% of the known tRNAs
    - with an error rate of 1 false positive per 15 billion nucleotides

Summary

- Hard consensus patterns are difficult to design
- SCFGs are powerful but slow
  (thousands of hours for scanning a bacterial genome)
- Specialised programs have been developed, each recognising a specific structure; these programs are generally sensitive, specific and (relatively) fast:
  - tRNAscan-SE (by Sean Eddy)
  - detects 99% of the known tRNAs
  - with an error rate of 1 false positive per 15 billion nucleotides

Summary

- Hard consensus patterns are difficult to design
- SCFGs are powerful but slow
(thousands of hours for scanning a bacterial genome)
- Specialised programs have been developed, each recognising a specific structure; these programs are generally sensitive, specific and (relatively) fast:
  - tRNAscan-SE (by Sean Eddy)
  - detects 99% of the known tRNAs
  - with an error rate of 1 false positive per 15 billion nucleotides

Summary

- Hard consensus patterns are difficult to design
- SCFGs are powerful but slow
  (thousands of hours for scanning a bacterial genome)
- Specialised programs have been developed, each recognising a specific structure; these programs are generally sensitive, specific and (relatively) fast:
  - tRNAscan-SE (by Sean Eddy)
  - detects 99% of the known tRNAs
  - with an error rate of 1 false positive per 15 billion nucleotides

Summary

- Hard consensus patterns are difficult to design
- SCFGs are powerful but slow
  (thousands of hours for scanning a bacterial genome)
- Specialised programs have been developed, each recognising a specific structure; these programs are generally sensitive, specific and (relatively) fast:
  - tRNAscan-SE (by Sean Eddy)
  - detects 99% of the known tRNAs
  - with an error rate of 1 false positive per 15 billion nucleotides

References

- M. Zuker.
  On finding all suboptimal foldings of an RNA molecule.
  244:48–52, 1989.

- Y. Ding and C. E. Lawrence.
  A bayesian statistical algorithm for rna secondary structure prediction.
  Computers & Chemistry, pages 387–400, 1999.

- J S McCaskill.
  The equilibrium partition function and base pair binding probabilities for RNA secondary structure.
  Biopolymers, 29(6-7):1105–19, Jan 1990.

References (cont.)

- Mirela Andronescu, Zhi Chuan Zhang, and Anne Condon.
  Secondary structure prediction of interacting RNA molecules.
  J Mol Biol, 345(5):987–1001, Feb 2005.

- Can Alkan, Emre Karakoc, Joseph H Nadeau, S Cenk Sahinalp, and Kaizhong Zhang.
  RNA-RNA interaction prediction and antisense RNA target search.
  J Comput Biol, 13(2):267–82, Mar 2006.

- Ho-Lin Chen, Anne Condon, and Hosna Jabbari.
  An O(n(5)) algorithm for MFE prediction of kissing hairpins and 4-chains in nucleic acids.
  J Comput Biol, 16(6):803–15, Jun 2009.

References (cont.)

- Hamidreza Chitsaz, Raheleh Salari, S Cenk Sahinalp, and Rolf Backofen.
  A partition function algorithm for interacting nucleic acid strands.
  Bioinformatics, 25(12):i365–73, Jun 2009.

- Jakob Skou Pedersen, Gill Bejerano, Adam Siepel, Kate Rosenbloom, Kerstin Lindblad-Toh, Eric S Lander, W James Kent, Webb Miller, and David Haussler.
  Identification and classification of conserved rna secondary structures in the human genome.
  PLoS Comput Biol, 2(4):e33, Apr 2006.

Pensez-y!

L'impression de ces notes n'est probablement pas nécessaire!