# Sampling RNA Secondary Structures with Pseudoknots using Analytic Combinatorics

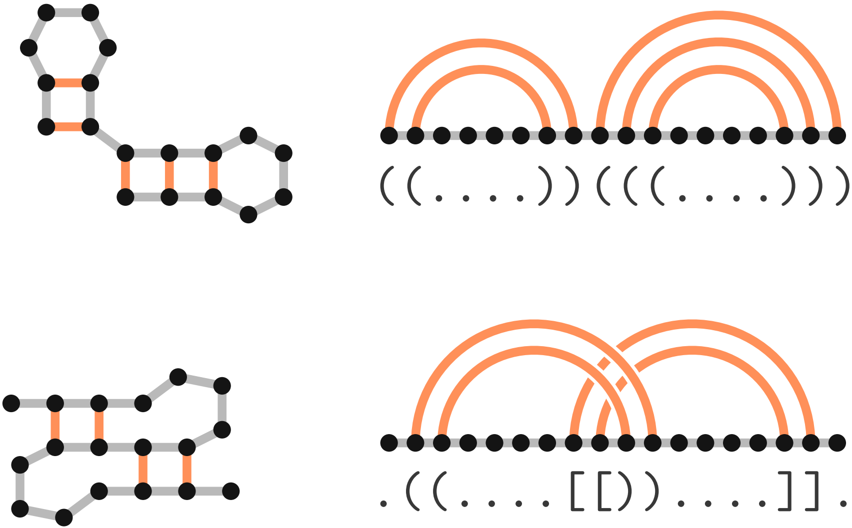
Casper Asbjørn Eriksen[1] Daniel Merkle[1,2]
Markus Nebel[2] Jonas Vistrup[1]

[1]Algorithmic Cheminformatics Group, University of Southern Denmark
[2]Faculty of Technology, Bielefeld University

39th TBI Winterseminar

# RNA pseudoknots





# Grammars for RNA Secondary Structures



The following CFG generates well-formed dot-bracket strings (Motzkin words):

$$S \to \bullet S \mid (S)\,S \mid \epsilon$$

However, we might want a grammar with biologically meaningful rules.

- Energy estimations
- Parametrised generation
- *Uniform / non-uniform Boltzmann sampling*

| Rules | |
|---|---|
| $f_1 = S \to TAC,$ $f_2 = T \to TAC,$ $f_3 = T \to C,$ $f_4 = C \to C\bullet,$ $f_5 = C \to \epsilon,$ | exterior loop |
| $f_6 = A \to (L),$ $f_7 = L \to (L),$ | initiate and extend stem |
| $f_8 = L \to M,$ | initiate multiloop |
| $f_9 = L \to P,$ $f_{10} = L \to Q,$ $f_{11} = L \to R,$ | initiate interior loop |
| $f_{12} = L \to F,$ | initiate hairpin loop |
| $f_{13} = L \to G,$ | initiate bulge loop |
| $f_{14} = G \to (L)\bullet,$ $f_{15} = G \to (L)B\bullet\bullet,$ $f_{16} = G \to \bullet(L),$ $f_{17} = G \to \bullet\bullet B(L),$ $f_{18} = B \to B\bullet,$ $f_{19} = B \to \epsilon,$ | bulge loops |
| $f_{20} = F \to \bullet\bullet\bullet,$ $f_{21} = F \to \bullet\bullet\bullet\bullet,$ $f_{22} = F \to \bullet\bullet\bullet\bullet\bullet H,$ $f_{23} = H \to H\bullet,$ $f_{24} = H \to \epsilon,$ | hairpin loop |
| $f_{25} = P \to \bullet(L)\bullet,$ $f_{26} = P \to \bullet(L)\bullet\bullet,$ $f_{27} = P \to \bullet\bullet(L)\bullet,$ $f_{28} = P \to \bullet\bullet(L)\bullet\bullet,$ | small interior loops |
| $f_{29} = Q \to \bullet\bullet(L)K\bullet\bullet\bullet,$ $f_{30} = Q \to \bullet\bullet\bullet J(L)K\bullet\bullet,$ $f_{31} = R \to \bullet(L)K\bullet\bullet\bullet,$ $f_{32} = R \to \bullet\bullet\bullet J(L)\bullet,$ $f_{33} = J \to J\bullet,$ $f_{34} = J \to \epsilon,$ $f_{35} = K \to K\bullet,$ $f_{36} = K \to \epsilon,$ | other interior loops |
| $f_{37} = M \to U(L)U(L)N,$ $f_{38} = N \to U(L)N,$ $f_{39} = N \to U,$ $f_{40} = U \to U\bullet,$ $f_{41} = U \to \epsilon.$ | multiloop |

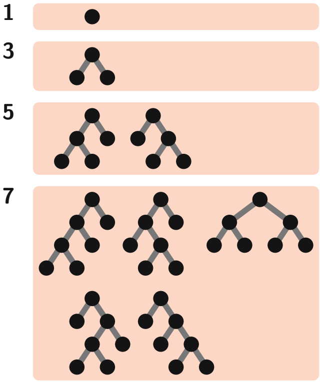
(Nebel et. al 2011)

# Analytic Combinatorics



(Ordered) Binary trees:

$$B = \bullet + [\bullet \times B \times B]$$




*"If you can specify it, you can analyse it."*
*- Philippe Flajolet*

# Analytic Combinatorics

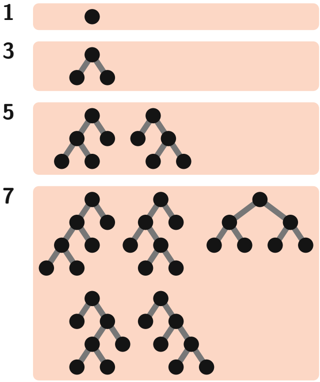


(Ordered) Binary trees:

$$B = \bullet + [\bullet \times B \times B]$$

Number of binary trees of size $n$:

1, 0, 1, 0, 2, 0, 5, 0, 14... (seq. **A000108**)




*"If you can specify it, you can analyse it."*
*- Philippe Flajolet*

# Analytic Combinatorics

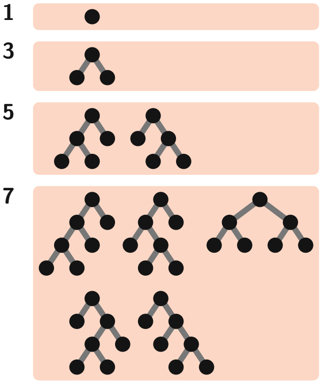


(Ordered) Binary trees:

$$B = \bullet + [\bullet \times B \times B]$$

Number of binary trees of size $n$:

1, 0, 1, 0, 2, 0, 5, 0, 14... (seq. **A000108**)




**Generating function:**

$$GF_B(z) = z + zGF_B(z)^2$$ Symbolic transfer thm.

$$= \frac{1 - \sqrt{1 - 4z}}{2}$$ Solve + expansion

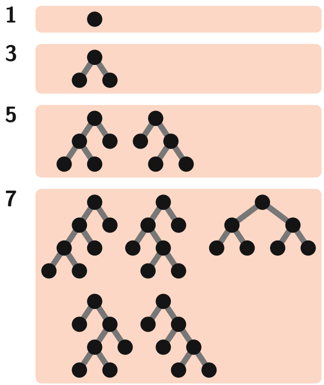
$$= \mathbf{1}z + \mathbf{1}z^3 + \mathbf{2}z^5 + \mathbf{5}z^7 + \mathbf{14}z^9 + \cdots$$

*"If you can specify it, you can analyse it."*
*- Philippe Flajolet*

# Analytic Combinatorics

Description of combinatorial structures:

- Binary trees: $B = \bullet + [\bullet \times B \times B]$
- General trees: $T = \bullet + [\bullet \times \mathrm{SEQ}(T)]$
- Derangements: $D = \mathrm{SET}(\mathrm{CYC}_{\geq 1}(\bullet))$




*"If you can specify it, you can analyse it."*
*- Philippe Flajolet*

# Analytic Combinatorics

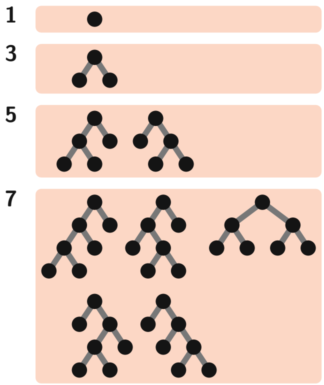


Description of combinatorial structures:

- Binary trees: $B = \bullet + [\bullet \times B \times B]$
- General trees: $T = \bullet + [\bullet \times \mathrm{SEQ}(T)]$
- Derangements: $D = \mathrm{SET}(\mathrm{CYC}_{\geq 1}(\bullet))$

A context-free language is a combinatorial class:

- Grammar: $S \to \bullet S \mid (S)\, S \mid \epsilon$
- Specification: $S = [\bullet \times S] + [\,(\, \times S \times\, )\, \times S]$



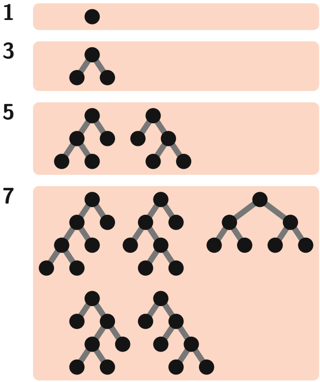

*"If you can specify it, you can analyse it."*
*- Philippe Flajolet*

# Analytic Combinatorics



A **Boltzmann Sampler** samples class a uniformly

- Recursive algorithm using GF




*"If you can specify it, you can analyse it."*
*- Philippe Flajolet*

# Analytic Combinatorics

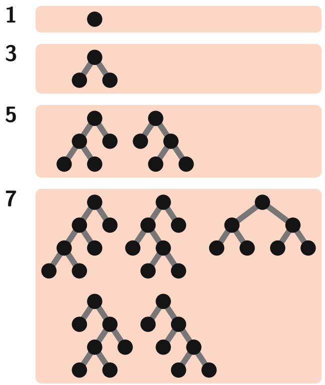


A **Boltzmann Sampler** samples class a uniformly

- Recursive algorithm using GF

## Maximum Likelihood Sampling

- Non-uniform sampling by weighing construction rules
- Obtain weights by parsing an ensemble of structures




*"If you can specify it, you can analyse it."*
*- Philippe Flajolet*

# Boltzmann Combinatorial Object Sampler



**BCOS** is a *work-in progress* Boltzmann sampling libary for:

- Uniform sampling of arbitrary combinatorial classes
- Weighted sampling with predefined weights
- Maximum Likelihood ensemble-based sampling

# Boltzmann Combinatorial Object Sampler



**BCOS** is a *work-in progress* Boltzmann sampling libary for:

- Uniform sampling of arbitrary combinatorial classes
- Weighted sampling with predefined weights
- Maximum Likelihood ensemble-based sampling

**Features**:

- Dynamic arbitrary-precision evaluation of generating functions guarantees correctness
- Ability to sample complex classes even if no closed form GF can be found.
- Python interface for Combinatorial object semantics allows intuitive integration.

# Boltzmann Combinatorial Object Sampler



## Ordered tree

```
import bcos
tree_class = bcos.System(
    "T = N + (N * SEQ(T)),
     N = Atom"
    )
tree_class.sample(size=(5,10))

> (N,((N,(N,N,N,N)),(N,(N)))))
```

# Boltzmann Combinatorial Object Sampler



## Ordered tree

```
import bcos
tree_class = bcos.System(
    "T = N + (N * SEQ(T)),
     N = Atom"
    )
tree_class.sample(size=(5,10))

> (N,((N,(N,N,N,N)),(N,(N))))
```

## RNA secondary structure

```
import bcos
g = bcos.cfg(
    "S -> . S | ( S ) | "
    )
gclass = bocs.System(g)
gclass.sample(size=21)

> ..((.(..)))..((.))(.)
```

# Multiple Context-Free Grammars



However, CFGs are not expressive enough to describe secondary structures with pseudoknots.

Introducing: **Multiple Context Free Grammars** (MCFG)

- Describes a broader class of languages than CFGs
- Allows for non-local correlations

Next step: designing an MCFG for pseudoknot structures.

Example MCFG ( $a^n b^n c^n d^n$ ):

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \rightarrow \begin{pmatrix} ab \\ cd \end{pmatrix}$$

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \rightarrow \begin{pmatrix} aA_1b \\ cA_2d \end{pmatrix}$$

Example derivation:

$$\begin{aligned} & A_1A_2 \\ \rightarrow\ & aA_1bcA_2d \\ \rightarrow\ & aabbccdd \end{aligned}$$

# Pseudoknot Grammars



Problem: Potentially infinite alphabet

- $(), [], \{\}, <>, Aa, Bb, \cdots \quad \cdots, \mathrm{A}\alpha, \mathrm{B}\beta, \Gamma\gamma$?

# Pseudoknot Grammars



Problem: Potentially infinite alphabet

- $(), [], \{\}, <>, Aa, Bb, \cdots \quad \cdots, \mathrm{A}\alpha, \mathrm{B}\beta, \Gamma\gamma$?

Possible solution:

- Parametrised alphabet:

$$S \to A_1 A_2 \mid \bullet \mid (_X\, S\, )_X$$

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \to \begin{pmatrix} SA_1 \\ A_2 \end{pmatrix} \Bigg| \begin{pmatrix} A_1 \\ SA_2 \end{pmatrix} \Bigg| \begin{pmatrix} (_X A_1 \\ )_X A_2 \end{pmatrix} \Bigg| \begin{pmatrix} \epsilon \\ \epsilon \end{pmatrix}$$

where $(_X, )_X = (), [], \cdots$.

# Pseudoknot Grammars



Parsing:

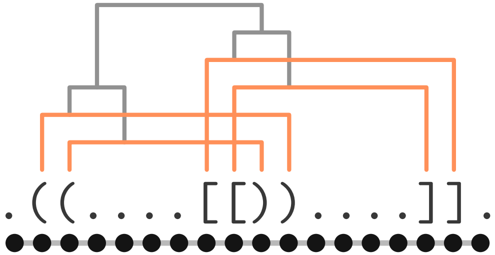
- Instantiate grammar with sufficient degree and parser

  **or**:

- Generate the lowest level of the parse tree and parse this

Generation:

- Generate using simple grammar and create string from parse tree



# Sampling Pseudoknot Structures



## Strategy

- Let $G$ be a parametrised MCFG (**PMCFG**)
- Let $L$ be a library of pseudoknot RNA dot-bracket strings, possibly containing pseudoknots

# Sampling Pseudoknot Structures



**Strategy**

- Let $G$ be a parametrised MCFG (**PMCFG**)
- Let $L$ be a library of pseudoknot RNA dot-bracket strings, possibly containing pseudoknots
- Instantiate $G$ and parse each string in $L$

# Sampling Pseudoknot Structures



**Strategy**

- Let $G$ be a parametrised MCFG (**PMCFG**)
- Let $L$ be a library of pseudoknot RNA dot-bracket strings, possibly containing pseudoknots
- Instantiate $G$ and parse each string in $L$
- Result is a stochastic parametrised MCFG (**SPMCFG**)

# Sampling Pseudoknot Structures



## Strategy

- Let $G$ be a parametrised MCFG (**PMCFG**)
- Let $L$ be a library of pseudoknot RNA dot-bracket strings, possibly containing pseudoknots
- Instantiate $G$ and parse each string in $L$
- Result is a stochastic parametrised MCFG (**SPMCFG**)
- Convert to a weighted combinatorial class $R$

# Sampling Pseudoknot Structures



## Strategy

- Let $G$ be a parametrised MCFG (**PMCFG**)
- Let $L$ be a library of pseudoknot RNA dot-bracket strings, possibly containing pseudoknots
- Instantiate $G$ and parse each string in $L$
- Result is a stochastic parametrised MCFG (**SPMCFG**)
- Convert to a weighted combinatorial class $R$
- Calculate GF based on **SCFG** reduction

# Sampling Pseudoknot Structures



## Strategy

- Let $G$ be a parametrised MCFG (**PMCFG**)
- Let $L$ be a library of pseudoknot RNA dot-bracket strings, possibly containing pseudoknots
- Instantiate $G$ and parse each string in $L$
- Result is a stochastic parametrised MCFG (**SPMCFG**)
- Convert to a weighted combinatorial class $R$
- Calculate GF based on **SCFG** reduction
- Sample $R$, and create dot-bracket string based on the parse tree.

# Thank you!