Efficient algorithms for generating pattern-avoiding combinatorial objects

Torsten Mütze
University of Warwick + Charles University Prague

joint work with Petr Gregor (Charles University), Elizabeth Hartung (MCLA), Hung P. Hoang (ETH Zurich), Arturo Merino (TU Berlin), Namrata (University of Warwick), Aaron Williams (Williams College)

Permutation Patterns 2023
Introduction

• many different classes of combinatorial objects

binary trees
Introduction

- many different classes of combinatorial objects

<table>
<thead>
<tr>
<th>Binary Trees</th>
<th>Permutations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Binary Trees" /></td>
<td><img src="image" alt="Permutations" /></td>
</tr>
</tbody>
</table>

- 123
- 132
- 312
- 321
- ...
Introduction

- many different classes of combinatorial objects

binary trees

permutations

bitstrings
Introduction

- many different classes of combinatorial objects

<table>
<thead>
<tr>
<th>Binary Trees</th>
<th>Permutations</th>
<th>Bitstrings</th>
<th>Set Partitions</th>
</tr>
</thead>
</table>
| ![Binary Trees](image) | 123  
   132  
   312  
   321  
   ... | 000  
   001  
   010  
   011  
   ... | \{1, 2, 3, 4\}  
   \{1, 2, 3\}{4}  
   \{1, 2\}{3, 4}  
   \{1, 2\}{3}{4}  
   ... |
Introduction

- many different classes of combinatorial objects

_binary trees_  

- permutations

- bitstrings

- set partitions

- fundamental tasks:
  counting, sampling, optimization
Introduction

• many different classes of combinatorial objects

\begin{align*}
\text{binary trees} & : \\
& \begin{array}{c}
\text{123} \\
\text{132} \\
\text{312} \\
\text{321} \\
\text{...}
\end{array} \\
\text{permutations} & : \\
& \begin{array}{c}
\text{000} \\
\text{001} \\
\text{010} \\
\text{011} \\
\text{...}
\end{array} \\
\text{bitstrings} & : \\
& \begin{array}{c}
\{1, 2, 3, 4\} \\
\{1, 2, 3\}\{4\} \\
\{1, 2\}\{3, 4\} \\
\{1, 2\}\{3\}\{4\} \\
\text{...}
\end{array}
\end{align*}

• fundamental tasks:
  counting, sampling, optimization
+ exhaustive generation [Knuth TAOCP Vol. 4A]
Exhaustive generation

• **Goal:** generate all objects of a combinatorial class efficiently
Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
- ultimately: each new object in **constant time**
Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
- Ultimately: each new object in **constant time**
- Consecutive objects differ by ‘small amount’ → **Gray code**
Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
- ultimately: each new object in **constant time**
- consecutive objects differ by ‘small amount’ → **Gray code**

- **Examples:**
  - binary trees by **rotations** [Lucas, Roelants van Baronaigien, Ruskey 93]
Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
- ultimately: each new object in **constant time**
- consecutive objects differ by ‘small amount’ → **Gray code**

- **Examples:**
  - binary trees by **rotations** [Lucas, Roelants van Baronaigien, Ruskey 93]
  - permutations by **adjacent transpositions** (Steinhaus-Johnson-Trotter algorithm) [Johnson 64], [Trotter 62]
Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
- ultimately: each new object in **constant time**
- consecutive objects differ by ‘small amount’ → **Gray code**

- **Examples:**
  - binary trees by **rotations** [Lucas, Roelants van Baronaigien, Ruskey 93]
  - permutations by **adjacent transpositions** (Steinhaus-Johnson-Trotter algorithm) [Johnson 64], [Trotter 62]
  - bitstrings by **bitflips** (Binary reflected Gray code) [Gray 53]
Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
- ultimately: each new object in **constant time**
- consecutive objects differ by ‘small amount’ → **Gray code**

- **Examples:**
  - binary trees by **rotations** [Lucas, Roelants van Baronaigien, Ruskey 93]
  - permutations by **adjacent transpositions**
    (Steinhaus-Johnson-Trotter algorithm) [Johnson 64], [Trotter 62]
  - bitstrings by **bitflips** (Binary reflected Gray code) [Gray 53]
  - set partitions by **element exchanges** [Kaye 76]
Flip graphs, lattices & polytopes

- **Flip graph**: vertices are combinatorial objects, edges capture change operations
Flip graphs, lattices & polytopes

- **Flip graph**: vertices are combinatorial objects, edges capture change operations
Flip graphs, lattices & polytopes

- **Flip graph**: vertices are combinatorial objects, edges capture change operations
Flip graphs, lattices & polytopes

- **Flip graph**: vertices are combinatorial objects, edges capture change operations
Flip graphs, lattices & polytopes

- **Flip graph**: vertices are combinatorial objects, edges capture change operations

- many flip graphs can be equipped with a poset structure and realized as polytopes
Flip graphs, lattices & polytopes

- **Flip graph**: vertices are combinatorial objects, edges capture change operations

  - Tamari lattice / associahedron
  - weak order / permutahedron
  - Boolean lattice / hypercube

- many flip graphs can be equipped with a poset structure and realized as polytopes
Flip graphs, lattices & polytopes

- **Flip graph**: vertices are combinatorial objects, edges capture change operations

- Tamari lattice / associahedron
- weak order / permutahedron
- Boolean lattice / hypercube

- many flip graphs can be equipped with a poset structure and realized as polytopes
- exhaustive generation $\leftrightarrow$ Hamilton path (HP)/cycle (HC)
Flip graphs, lattices & polytopes

- **Flip graph**: vertices are combinatorial objects, edges capture change operations

  ![Tamari lattice / associahedron](image1)
  ![weak order / permutahedron](image2)
  ![Boolean lattice / hypercube](image3)

- many flip graphs can be equipped with a poset structure and realized as polytopes

- exhaustive generation \(\leftrightarrow\) **Hamilton path (HP)/cycle (HC)**
Gray code generation

• many tailormade algorithms, few general approaches

[Avis, Fukuda 96], [Barcucci et al. 99], [Li, Sawada 09], [Ruskey, Sawada, Williams 12], [Williams 13]
Gray code generation

- many tailormade algorithms, few general approaches

  [Avis, Fukuda 96], [Barcucci et al. 99], [Li, Sawada 09], [Ruskey, Sawada, Williams 12], [Williams 13]

  - cf. *generating functions* for counting
Gray code generation

- many tailormade algorithms, few general approaches
  
  [Avis, Fukuda 96], [Barcucci et al. 99], [Li, Sawada 09], [Ruskey, Sawada, Williams 12], [Williams 13]

  - cf. generating functions for counting
  - cf. Markov chains for random sampling
Gray code generation

- many tailormade algorithms, few general approaches
  
  [Avis, Fukuda 96], [Barcucci et al. 99], [Li, Sawada 09], [Ruskey, Sawada, Williams 12], [Williams 13]

- cf. generating functions for counting
- cf. Markov chains for random sampling

- This work: a general framework for Gray code generation
Gray code generation

- many tailormade algorithms, few general approaches
  
  [Avis, Fukuda 96], [Barcucci et al. 99], [Li, Sawada 09], [Ruskey, Sawada, Williams 12], [Williams 13]

- cf. generating functions for counting
  
  cf. Markov chains for random sampling

- This work: a general framework for Gray code generation

- Results: all aforementioned algorithms as special cases
Gray code generation

- many tailormade algorithms, few general approaches
  
  [Avis, Fukuda 96], [Barcucci et al. 99], [Li, Sawada 09], [Ruskey, Sawada, Williams 12], [Williams 13]

  ○ cf. generating functions for counting
  ○ cf. Markov chains for random sampling

- This work: a general framework for Gray code generation

- Results: all aforementioned algorithms as special cases
  
  + many new results and algorithms for a multitude of other combinatorial objects and the corresponding lattices / polytopes
Gray code generation

• many tailormade algorithms, few general approaches
  [Avis, Fukuda 96], [Barcucci et al. 99], [Li, Sawada 09], [Ruskey, Sawada, Williams 12], [Williams 13]
  ○ cf. generating functions for counting
  ○ cf. Markov chains for random sampling

• This work: a general framework for Gray code generation

• Results: all aforementioned algorithms as special cases
  + many new results and algorithms for a multitude of other combinatorial objects and the corresponding lattices / polytopes
  + in particular, objects defined by pattern-avoidance
Gray code generation

• many tailormade algorithms, few general approaches
  
  [Avis, Fukuda 96], [Barcucci et al. 99], [Li, Sawada 09], [Ruskey, Sawada, Williams 12], [Williams 13]

  ○ cf. generating functions for counting
  ○ cf. Markov chains for random sampling

• This work: a general framework for Gray code generation

• Results: all aforementioned algorithms as special cases
  + many new results and algorithms for a multitude of other combinatorial objects and the corresponding lattices / polytopes
  + in particular, objects defined by pattern-avoidance

• Idea: Encode objects as a set $F_n \subseteq S_n$ of permutations of length $n$
Jumps

- **Jump**: move an entry in the permutation across some neighboring smaller entries (left or right)
**Jumps**

- **Jump**:= move an entry in the permutation across some neighboring smaller entries (left or right)

```
4 5 1 3 2 6
4 1 3 2 5 6
```
Jumps

• **Jump** := move an entry in the permutation across some neighboring smaller entries (left or right)
Jumps

- **Jump**: move an entry in the permutation across some neighboring smaller entries (left or right)
Jumps

- **Jump**: move an entry in the permutation across some neighboring smaller entries (left or right)
Jumps

- **Jump**: move an entry in the permutation across some neighboring smaller entries (left or right)
Jumps

- **Jump**: move an entry in the permutation across some neighboring smaller entries (left or right)
Jumps

- **Jump**: move an entry in the permutation across some neighboring smaller entries (left or right)
Jumps

- **Jump**: move an entry in the permutation across some neighboring smaller entries (left or right)
Algorithm J

attempts to generate a set of permutations $F_n \subseteq S_n$

• Start with an initial permutation.

• In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from $F_n$ is created.

**Minimal jump:** no shorter jump of the same value produces a permutation in $F_n$
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

**Minimal jump:** no shorter jump of the same value produces a permutation in $F_n$
Algorithm J

Algorithm J attempts to generate a set of permutations \( F_n \subseteq S_n \)

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from \( F_n \) is created.

- **Minimal jump:** no shorter jump of the same value produces a permutation in \( F_n \)
Algorithm J

attempts to generate a set of permutations \( F_n \subseteq S_n \)

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from \( F_n \) is created.

**Minimal jump:** no shorter jump of the same value produces a permutation in \( F_n \)
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from $F_n$ is created.

- **Minimal jump**: no shorter jump of the same value produces a permutation in $F_n$
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

- **Minimal jump**: no shorter jump of the same value produces a permutation in $F_n$
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.

- In the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- **Minimal jump**: no shorter jump of the same value produces a permutation in $F_n$
Algorithm J

attempts to generate a set of permutations $F_n \subseteq S_n$

• Start with an initial permutation.

• In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

• Example: $F_4 = \{1243, 1423, 2134, 4123, 4213\}$
Algorithm J

attempts to generate a set of permutations $F_n \subseteq S_n$

• Start with an initial permutation.

• In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

• Example: $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

1243
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- Example: $F_4 = \{1243, 1423, 2134, 4123, 4213\}$
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

• Start with an initial permutation.

• In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

• Example: $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

\[
\begin{array}{c}
1243 \\
\end{array}
\]
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- **Example:** $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

  1243
  1423
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.
- Stop if no jump is possible or jump direction is ambiguous.

Example: $F_4 = \{1243, 1423, 2134, 4123, 4213\}$
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

**Example:** $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

```
1243
1423
```
Algorithm J

attempts to generate a set of permutations $F_n \subseteq S_n$

• Start with an initial permutation.

• In the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from $F_n$ is created.

  Stop if no jump is possible or jump direction is ambiguous.

• **Example:** $F_4 = \{1243, 1423, 2134, 4123, 4213\}$
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a \textbf{minimal jump} of the \textbf{largest possible value}, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- \textbf{Example:} $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

\[
\begin{align*}
1243 \\
1423 \\
4123
\end{align*}
\]
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

• Start with an initial permutation.

• In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

• Example: $F_4 = \{1243, 1423, 2134, 4123, 4213\}$
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- Example: $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

1243
1423
4123
4213
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

**Example:** $F_4 = \{1243, 1423, 2134, 4123, 4213\}$
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- **Example:** $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

1243
1423
4123
4213
2134
Algorithm J

attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

**Example:** $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

1243
1423
4123
4213
2134
Algorithm J

Algorithm J attempts to generate a set of permutations \( F_n \subseteq S_n \)

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from \( F_n \) is created.

Stop if no jump is possible or jump direction is ambiguous.

**Example:** \( F_4 = \{1243, 1423, 2134, 4123, 4213\} \)

\[
\begin{array}{cccccc}
1243 & 4213 \\
1423 &   \\
4123 &   \\
4213 &   \\
2134 & \checkmark \\
\end{array}
\]
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- Example: $F_4 = \{1243, 1423, 2134, 4123, 4213\}$
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- Example: $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

\[
\begin{array}{cccc}
1243 & 4213 \\
1423 & 2134 \\
4123 & \\
4213 & \\
2134 & \checkmark \\
\end{array}
\]
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

**Example:** $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

<table>
<thead>
<tr>
<th>1243</th>
<th>4213</th>
</tr>
</thead>
<tbody>
<tr>
<td>1423</td>
<td>2134</td>
</tr>
<tr>
<td>4123</td>
<td>X</td>
</tr>
<tr>
<td>4213</td>
<td>no jump possible</td>
</tr>
<tr>
<td>2134</td>
<td>✓</td>
</tr>
</tbody>
</table>
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- **Example:** $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

<table>
<thead>
<tr>
<th>1243</th>
<th>4213</th>
<th>1423</th>
</tr>
</thead>
<tbody>
<tr>
<td>1423</td>
<td>2134</td>
<td>2134</td>
</tr>
<tr>
<td>4123</td>
<td>4213</td>
<td>no jump possible</td>
</tr>
<tr>
<td>2134</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Algorithm J

Algorithm J attempts to generate a set of permutations \( F_n \subseteq S_n \)

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from \( F_n \) is created.

Stop if no jump is possible or jump direction is ambiguous.

**Example:** \( F_4 = \{1243, 1423, 2134, 4123, 4213\} \)

\[
\begin{array}{c}
1243 \\
1423 \\
4123 \\
4213 \\
2134
\end{array}
\begin{array}{c}
4213 \\
2134 \\
\text{no jump possible}
\end{array}
\begin{array}{c}
1243 \\
1423 \\
4123 \\
4213 \\
2134
\end{array}
\]
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- **Example:** $F_4 = \{1243, 1423, 2134, 4123, 4213\}$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1243</td>
<td>4213</td>
<td>1423</td>
<td>4213</td>
</tr>
<tr>
<td>1423</td>
<td>2134</td>
<td>2134</td>
<td>1423</td>
</tr>
<tr>
<td>4123</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4213</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2134</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- no jump possible
- direction ambiguous
- no jump possible
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- If every permutation from $F_n$ is visited, we say that Algorithm J generates $F_n$ (visiting twice is impossible)
Algorithm J

Algorithm J attempts to generate a set of permutations $F_n \subseteq S_n$

- Start with an initial permutation.
- In the current permutation, perform a minimal jump of the largest possible value, so that a previously unvisited permutation from $F_n$ is created.

Stop if no jump is possible or jump direction is ambiguous.

- If every permutation from $F_n$ is visited, we say that Algorithm J generates $F_n$ (visiting twice is impossible)
- **Question:** When does Algorithm J generate $F_n$?
Tree of permutations

- root := empty permutation $\varepsilon$
- given a permutation length $n - 1$, its children are obtained by inserting $n$ in every possible position
Tree of permutations

- root := empty permutation \( \varepsilon \)
- given a permutation length \( n - 1 \), its children are obtained by inserting \( n \) in every possible position

depth \( n = \) all permutations of length \( n \)
Tree of permutations

- root := empty permutation $\varepsilon$
- given a permutation length $n - 1$, its children are obtained by inserting $n$ in every possible position
  - symbol $n$ at leftmost or rightmost position
  - else

$\text{depth } n =$ all permutations of length $n$
Tree of permutations

- root := empty permutation $\varepsilon$
- given a permutation length $n - 1$, its children are obtained by inserting $n$ in every possible position
  - symbol $n$ at leftmost or rightmost position
  - else

depth $n = \text{all permutations of length } n$
Tree of permutations

- root := empty permutation \(\varepsilon\)
- given a permutation length \(n - 1\), its children are obtained by inserting \(n\) in every possible position
  - symbol \(n\) at leftmost or rightmost position
  - else

\[ \text{depth } n = \text{all permutations of length } n \]
Tree of permutations

- we may prune subtrees iff their root is \( \varepsilon \)
Tree of permutations

- we may prune subtrees iff their root is
Tree of permutations

- we may prune subtrees iff their root is
Tree of permutations

- we may prune subtrees iff their root is $\varepsilon$. 

...
Tree of permutations

- we may prune subtrees iff their root is $\varepsilon$. 
Tree of permutations

- we may prune subtrees iff their root is
- given any such pruned tree, a set of permutation $F_n \subseteq S_n$ in depth $n$ is called zigzag language
Tree of permutations

- we may prune subtrees iff their root is
- given any such pruned tree, a set of permutation $F_n \subseteq S_n$ in depth $n$ is called zigzag language
Tree of permutations

- we may prune subtrees iff their root is
- given any such pruned tree, a set of permutation $F_n \subseteq S_n$ in depth $n$ is called zigzag language
Tree of permutations

- We may prune subtrees iff their root is.
- Given any such pruned tree, a set of permutation $F_n \subseteq S_n$ in depth $n$ is called zigzag language.

Examples:
- Prune nothing: $F_n = S_n$, $|F_n| = n!$
Tree of permutations

- we may prune subtrees iff their root is
- given any such pruned tree, a set of permutation $F_n \subseteq S_n$ in depth $n$ is called zigzag language

**Examples:**

- prune nothing: $F_n = S_n$, $|F_n| = n!$
- prune all green nodes: $F_n$ = permutations without peaks, $|F_n| = 2^{n-1}$
Tree of permutations

• we may prune subtrees iff their root is .

• given any such pruned tree, a set of permutation \( F_n \subseteq S_n \) in depth \( n \) is called zigzag language

Theorem: Algorithm J generates any zigzag language, using the identity permutation for initialization.
Tree of permutations

- we may prune subtrees iff their root is
- given any such pruned tree, a set of permutation $F_n \subseteq S_n$ in depth $n$ is called zigzag language

**Theorem:** Algorithm J generates any zigzag language, using the identity permutation for initialization.

**Proof:** Induction over the depth of the tree. □
Tree of permutations

- we may prune subtrees iff their root is
- given any such pruned tree, a set of permutation $F_n \subseteq S_n$ in depth $n$ is called zigzag language

**Theorem:** Algorithm J generates any zigzag language, using the identity permutation for initialization.

**Proof:** Induction over the depth of the tree.

- the number of zigzag languages is enormous:
  \[ \geq 2(n-1)!(n-2) = 2^{\Theta(n \log n)} \]
Tree of permutations

• we may prune subtrees iff their root is.

• given any such pruned tree, a set of permutation \( F_n \subseteq S_n \) in depth \( n \) is called zigzag language.

**Theorem:** Algorithm J generates any zigzag language, using the identity permutation for initialization.

**Proof:** Induction over the depth of the tree. □

• the number of zigzag languages is enormous:

\[
\geq 2^{(n-1)!(n-2)} = 2^{\Theta(n \log n)}
\]

• many of them encode interesting combinatorial objects.
Examples

$F_n = S_n$
$|F_n| = n!$

$F_n = \text{permutations without peaks}$
$|F_n| = 2^{n-1}$
**Examples**

\[ F_n = S_n \]

\[ |F_n| = n! \]

\[ F_n = \text{permutations without peaks} \]

\[ |F_n| = 2^{n-1} \]
Examples

\[ F_n = S_n \]
\[ |F_n| = n! \]

\[ F_n = \text{permutations without peaks} \]
\[ |F_n| = 2^{n-1} \]

Steinhaus-Johnson-Trotter!
minimal jumps
adjacent transpositions
HC on permutahedron
Examples

\[ F_n = S_n \]

\[ |F_n| = n! \]

\[ F_n = \text{permutations without peaks} \]

\[ |F_n| = 2^{n-1} \]

Steinhaus-Johnson-Trotter!

minimal jumps

\[ \leftrightarrow \text{adjacent transpositions} \]

\[ \leftrightarrow \text{HC on permutahedron} \]
Examples

\[ F_n = S_n \]
\[ |F_n| = n! \]

\[ F_n = \text{permutations without peaks} \]
\[ |F_n| = 2^{n-1} \]

Steinhaus-Johnson-Trotter!
minimal jumps
\[ \leftrightarrow \text{adjacent transpositions} \]
\[ \rightarrow HC \text{ on permutahedron} \]
Examples

$$F_n = S_n$$

$$|F_n| = n!$$

$$F_n$$ = permutations without peaks

$$|F_n| = 2^{n-1}$$

Steinhaus-Johnson-Trotter!

minimal jumps

adjacent transpositions

→ HC on permutahedron
Examples

\[ F_n = S_n \]
\[ |F_n| = n! \]

\[ F_n = \text{permutations without peaks} \]
\[ |F_n| = 2^{n-1} \]

Steinhaus-Johnson-Trotter!

minimal jumps

\[ [0 \quad 1 \quad i \text{ right of smaller entries}] \]
\[ [1 \quad 0 \quad i \text{ left of smaller entries}] \]

\[ x_i = \begin{cases} 
0 & \text{i right of smaller entries} \\
1 & \text{i left of smaller entries} 
\end{cases} \]

\( f \) is a minimal jump.

\[ \text{adjacent transpositions} \]

\[ \text{HC on permutahedron} \]
Examples

\[ F_n = S_n \]
\[ |F_n| = n! \]

\[ F_n = \text{permutations without peaks} \]
\[ |F_n| = 2^{n-1} \]

Steinhaus-Johnson-Trotter!

minimal jumps

\[ x_i = \begin{cases} 
0 & \text{i right of smaller entries} \\
1 & \text{i left of smaller entries} 
\end{cases} \]
Examples

\[ F_n = S_n \]
\[ |F_n| = n! \]

\[ F_n = \text{permutations without peaks} \]
\[ |F_n| = 2^{n-1} \]

Steinhaus-Johnson-Trotter!

minimal jumps

\[ x_i = \begin{cases} 
0 & i \text{ right of smaller entries} \\
1 & i \text{ left of smaller entries} 
\end{cases} \]

\[ f \]

adjacent transpositions

\[ \text{HC on permutahedron} \]
Examples

\[ F_n = S_n \]
\[ |F_n| = n! \]

\[ F_n = \text{permutations without peaks} \]
\[ |F_n| = 2^{n-1} \]

Steinhaus-Johnson-Trotter!

minimal jumps

\[ x_i = \begin{cases} 
0 & i \text{ right of smaller entries} \\
1 & i \text{ left of smaller entries} 
\end{cases} \]

Binary reflected Gray code!

minimal jumps

\[ \text{adjacent transpositions} \]

\[ \text{bitflips} \]

\[ \text{HC on permutahedron} \]

\[ \text{HC on hypercube} \]
General approach

Combinatorial objects
General approach

Set of permutations $F_n \subseteq S_n$

$\xrightarrow{f}$

Combinatorial objects
General approach

Set of permutations $F_n \subseteq S_n$

- run Algorithm J

\[ \text{List} = \text{Algo J}(F_n) \]

Combinatorial objects
General approach

Set of permutations
$F_n \subseteq S_n$

- run Algorithm J

List = Algo J($F_n$) $\xrightarrow{f} f^{-1}(\text{List})$

Combinatorial objects
General approach

- Set of permutations
  \[ F_n \subseteq S_n \]

- Combinatorial objects

- Run Algorithm J
  \[ \text{List} = \text{Algo J}(F_n) \]

- Interpret Algorithm J under the bijection
  \[ \text{Algo J} \]

\[ f^{-1}(\text{List}) \]
General approach

Set of permutations
\( F_n \subseteq S_n \)

\[ \text{List} = \text{Algo J}(F_n) \xrightarrow{f^{-1}} f^{-1}(\text{List}) \]

- run Algorithm J

- interpret Algorithm J under the bijection

\[ \text{Algo J} \xrightarrow{f^{-1}} f^{-1}(\text{Algo J}) \]

Combinatorial objects
General approach

Set of permutations \( F_n \subseteq S_n \)

Combinatorial objects

- run Algorithm J
  \[ \text{List} = \text{Algo J}(F_n) \]

- interpret Algorithm J under the bijection
  \[ \text{Algo J} \rightarrow f^{-1}(\text{Algo J}) \]

- minimal jumps
  \[ \text{minimal jumps} \rightarrow \text{‘small changes’} \]
General approach

- run Algorithm J
  \[ \text{List} = \text{Algo } J(F_n) \rightarrow f^{-1}(\text{List}) \]
- interpret Algorithm J under the bijection
  \[ \text{Algo } J \rightarrow f^{-1}(\text{Algo } J) \]
- minimal jumps
  \[ \leftrightarrow \text{‘small changes’} \]
  \[ \leftrightarrow \text{walks on lattices / polytopes} \]

Set of permutations
\[ F_n \subseteq S_n \]

Combinatorial objects

\( f \)
Efficient algorithms

- greedy algorithm as stated very inefficient (store and look-up exponentially many previous permutations)
Efficient algorithms

- greedy algorithm as stated very inefficient (store and look-up exponentially many previous permutations)
- can make it history-free (no look-up needed)
Efficient algorithms

- greedy algorithm as stated very inefficient (store and look-up exponentially many previous permutations)
- can make it history-free (no look-up needed)
- running time in each step governed by membership tests in $F_n$; typically $F_n$ not given explicitly, but by properties (e.g., ‘peak-free’ or ‘231-avoiding’)
Efficient algorithms

• greedy algorithm as stated very inefficient (store and look-up exponentially many previous permutations)
• can make it history-free (no look-up needed)
• running time in each step governed by membership tests in $F_n$; typically $F_n$ not given explicitly, but by properties (e.g., ‘peak-free’ or ‘231-avoiding’)
• in many cases polynomial-time algorithms for concrete objects, sometimes even loopless
Applications

• 1. pattern-avoiding permutations (classical/vincular/mesh patterns, monotone and geometric grid classes) [SODA'20]
Applications

- I. pattern-avoiding permutations (classical/vincular/mesh patterns, monotone and geometric grid classes) [SODA’20]

- VI. pattern-avoiding binary trees
Applications

• **I. pattern-avoiding permutations** (classical/vincular/mesh patterns, monotone and geometric grid classes) [SODA'20]

• **III. pattern-avoiding rectangulations** [SoCG'21]

• **VI. pattern-avoiding binary trees**
Applications

• **I. pattern-avoiding permutations** (classical/vincular/mesh patterns, monotone and geometric grid classes) [SODA’20]

• **II. lattice congruences** of the weak order on $S_n$

• **III. pattern-avoiding rectangulations** [SoCG’21]

• **VI. pattern-avoiding binary trees**
Applications

• I. pattern-avoiding permutations (classical/vincular/mesh patterns, monotone and geometric grid classes) $[^{\text{SODA'20}}]$

• II. lattice congruences of the weak order on $S_n$

• III. pattern-avoiding rectangulations $[^{\text{SoCG'21}}]$

• IV. elimination trees $[^{\text{SODA'22}}]$

• VI. pattern-avoiding binary trees
Applications

- **I. pattern-avoiding permutations** (classical/vincular/mesh patterns, monotone and geometric grid classes) [SODA’20]
- **II. lattice congruences** of the weak order on $S_n$
- **III. pattern-avoiding rectangulations** [SoCG’21]
- **IV. elimination trees** [SODA’22]
- **V. acyclic orientations of graphs** [SODA’23]
- **VI. pattern-avoiding binary trees**
Applications

• **I. pattern-avoiding permutations** (classical/vincular/mesh patterns, monotone and geometric grid classes) [SODA'20]

• **II. lattice congruences** of the weak order on $S_n$

• **III. pattern-avoiding rectangulations** [SoCG'21]

• **IV. elimination trees** [SODA'22]

• **V. acyclic orientations of graphs** [SODA'23]

• **VI. pattern-avoiding binary trees**
Pattern-avoiding permutations

• $S_n(\tau_1, \ldots, \tau_k) \subseteq S_n :=$ set of permutations avoiding each of the patterns $\tau_1, \ldots, \tau_k$
Pattern-avoiding permutations

- A pattern $\tau$ is **tame**, if
Pattern-avoiding permutations

- A pattern $\tau$ is tame, if
  - **classical**: largest entry not at the boundary
Pattern-avoiding permutations

- A pattern $\tau$ is **tame**, if
  
  **classical**: largest entry not at the boundary

  $2413$ ✓
Pattern-avoiding permutations

- A pattern $\tau$ is **tame**, if

  classical: largest entry not at the boundary

  2413  ✓  4213  ❌
Pattern-avoiding permutations

- A pattern $\tau$ is **tame**, if

  - **classical**: largest entry not at the boundary
    
    \[
    2413 \, \checkmark \quad 4213 \, \times
    \]

  - **vincular**: + one vincular pair involving the largest entry
Pattern-avoiding permutations

• A pattern $\tau$ is **tame**, if
  
  **classical**: largest entry not at the boundary
  
  $2413 \, \checkmark \quad 4213 \, \times$

  **vincular**: + one vincular pair involving the largest entry
  
  $2413 \, \checkmark$


Pattern-avoiding permutations

- A pattern $\tau$ is **tame**, if
  
  **classical**: largest entry not at the boundary
  
  $$2413 \, \checkmark \quad 4213 \, \times$$

  **vincular**: + one vincular pair involving the largest entry
  
  $$2413 \, \checkmark \quad 2413 \, \checkmark$$
A pattern $\tau$ is tame, if

- **classical**: largest entry not at the boundary
  - $2413 \, \checkmark \, 4213 \, \times$

- **vincular**: + one vincular pair involving the largest entry
  - $\underline{2}413 \, \checkmark \, \underline{2}413 \, \times$
  - $\underline{2}413 \, \checkmark$
Pattern-avoiding permutations

- A pattern $\tau$ is **tame**, if
  - **classical**: largest entry not at the boundary
    - 2413 ✓ 4213 ❌
  - **vincular**: + one vincular pair involving the largest entry
    - 2413 ✓ 2413 ❌
    - 2413 ✓ 2413 ❌
Pattern-avoiding permutations

- A pattern $\tau$ is **tame**, if
  - **classical**: largest entry not at the boundary
    - \[2413 \text{ ✓} \quad 4213 \text{ ❌}\]
  - **vincular**: $+$ one vincular pair involving the largest entry
    - \[2413 \text{ ✓} \quad 2413 \text{ ❌}\]
    - \[2413 \text{ ✓} \quad 2413 \text{ ❌}\]
  - **mesh**: $+$ no shaded cell in the top row
    - \[2413 \text{ ✓} \quad 2413 \text{ ❌}\]
Pattern-avoiding permutations

- A pattern $\tau$ is **tame**, if
  - classical: largest entry not at the boundary
    - $2413 \checkmark$  
    - $4213 \times$
  - vincular: $+\ $one vincular pair involving the largest entry
    - $2413 \checkmark$
    - $2413 \times$
    - $2413 \checkmark$
    - $2413 \times$
  - mesh: $+\ $no shaded cell in the top row
Pattern-avoiding permutations

- A pattern \( \tau \) is **tame**, if
  - **classical**: largest entry not at the boundary
    
    \[
    \begin{array}{c}
    2413 \quad \checkmark \\
    4213 \quad \times \\
    \end{array}
    \]
  - **vincular**: + one vincular pair involving the largest entry
    
    \[
    \begin{array}{c}
    2413 \quad \checkmark \\
    2413 \quad \checkmark \\
    2413 \quad \times \\
    2413 \quad \times \\
    \end{array}
    \]
  - **mesh**: + no shaded cell in the top row
    
    \[
    \begin{array}{c}
    \checkmark \\
    \checkmark \\
    \end{array}
    \]
Pattern-avoiding permutations

- A pattern $\tau$ is **tame**, if
  - **classical**: largest entry not at the boundary
    - $2413$ ✓
    - $4213$ ✗
  - **vincular**: + one vincular pair involving the largest entry
    - $2413$ ✓
    - $2413$ ✓
    - $2413$ ✓
    - $2413$ ✗
  - **mesh**: + no shaded cell in the top row
    - ✓
    - ✓
    - ✗
Pattern-avoiding permutations

- A pattern $\tau$ is **tame**, if
  - **classical**: largest entry not at the boundary
    
    $2413 \checkmark$  
    $4213 \times$
  
  - **vincular**: + one vincular pair involving the largest entry
    
    $2413 \checkmark$  
    $2413 \checkmark$  
    $2413 \times$
  
  - **mesh**: + no shaded cell in the top row
    
    ![Examples of mesh patterns](image)

**Theorem**: If $\tau_1, \ldots, \tau_k$ are tame patterns, then $S_n(\tau_1, \ldots, \tau_k)$ is a zigzag language.
Pattern-avoiding permutations

Tame patterns $\xrightarrow{f}$ Combinatorial objects
Pattern-avoiding permutations

Tame patterns $f$ Combinatorial objects

231 Catalan families
Pattern-avoiding permutations

Tame patterns \( f \) Combinatorial objects

231 Catalan families

- binary trees by \textit{rotations}
- triangulations by \textit{flips}
- Dyck paths by \textit{hill flips}
Pattern-avoiding permutations

Tame patterns $\rightarrow f \leftarrow$ Combinatorial objects

\begin{align*}
231 & \quad \text{Catalan families} & \quad \text{binary trees by rotations} \\
 & & \quad \text{triangulations by flips} \\
 & & \quad \text{Dyck paths by hill flips} \\
231 & \quad \text{Bell families} 
\end{align*}
Pattern-avoiding permutations

Tame patterns \( f \) Combinatorial objects

| 231       | Catalan families | • binary trees by rotations |
| 231       | Bell families    | • triangulations by flips  |
|           |                 | • Dyck paths by hill flips |
|           |                 | • set partitions by element exchanges |
# Pattern-avoiding permutations

Tame patterns $f$ Combinatorial objects

<table>
<thead>
<tr>
<th>231</th>
<th>Catalan families</th>
<th>Bell families</th>
<th>Bell families</th>
</tr>
</thead>
<tbody>
<tr>
<td>231</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>231,132</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- binary trees by rotations
- triangulations by flips
- Dyck paths by hill flips
- set partitions by element exchanges
- bitstrings by flips (BRGC)
### Pattern-avoiding permutations

#### Tame patterns \( f \) Combinatorial objects

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Catalan families</th>
<th>Bell families</th>
<th>Baxter families</th>
</tr>
</thead>
<tbody>
<tr>
<td>231</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>231</td>
<td>Catalan families</td>
<td></td>
<td></td>
</tr>
<tr>
<td>231,132</td>
<td>Bell families</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2413,3142</td>
<td>Baxter families</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Catalan families:
  - binary trees by rotations
  - triangulations by flips
  - Dyck paths by hill flips
- Bell families:
  - set partitions by element exchanges
  - bitstrings by flips (BRGC)
Pattern-avoiding permutations

Tame patterns $f$ Combinatorial objects

| 231    | Catalan families | • binary trees by rotations |
|        |                 | • triangulations by flips  |
|        |                 | • Dyck paths by hill flips |
| 231    | Bell families   | • set partitions by element exchanges |
| 231,132|                 | • bitstrings by flips (BRGC) |
| 2413,3142 | Baxter families | • diagonal rectangulations |
Pattern-avoiding permutations

Tame patterns $\leftrightarrow$ Combinatorial objects

- 231
  - Catalan families
  - binary trees by rotations
  - triangulations by flips
  - Dyck paths by hill flips

- 231
  - Bell families
  - set partitions by element exchanges

- 231,132
  - Baxter families
  - bitstrings by flips (BRGC)

- 2413,3142
  - Baxter families
  - diagonal rectangulations

- 35124,35142, 2-clumped pms.

- 24513,42513

- 35124,35142, 2-clumped pms.
### Pattern-avoiding permutations

<table>
<thead>
<tr>
<th>Tame patterns</th>
<th>Combinatorial objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>231</td>
<td>Catalan families</td>
</tr>
<tr>
<td></td>
<td>• binary trees by rotations</td>
</tr>
<tr>
<td></td>
<td>• triangulations by flips</td>
</tr>
<tr>
<td></td>
<td>• Dyck paths by hill flips</td>
</tr>
<tr>
<td>231</td>
<td>Bell families</td>
</tr>
<tr>
<td></td>
<td>• set partitions by element exchanges</td>
</tr>
<tr>
<td>231,132</td>
<td>Baxter families</td>
</tr>
<tr>
<td></td>
<td>• bitstrings by flips (BRGC)</td>
</tr>
<tr>
<td>2413,3142</td>
<td>2-clumped pms.</td>
</tr>
<tr>
<td>351214,3142,</td>
<td></td>
</tr>
<tr>
<td>24513,42513</td>
<td></td>
</tr>
</tbody>
</table>
Pattern-avoiding permutations

Tame patterns $f$ Combinatorial objects

231 Catalan families
• binary trees by rotations
• triangulations by flips
• Dyck paths by hill flips

231 Bell families
• set partitions by element exchanges

231,132

2413,3142 Baxter families
• bitstrings by flips (BRGC)

35124,35142, 2-clumped pms.

24513,42513

→ see the Combinatorial Object Server: www.combos.org/jump
Grid classes

- monotone grid class $\text{Grid}_n(M)$ [Huczynska, Vatter 06]
- geometric grid class $\text{Geo}_n(M)$ [Albert et al. 13]
Grid classes

- monotone grid class $\text{Grid}_n(M)$ [Huczynska, Vatter 06]
- geometric grid class $\text{Geo}_n(M)$ [Albert et al. 13]

**Theorem:** If $M = \begin{array}{cc}
-1 & +1 \\
+1 & +1 \\
\end{array}$, then both $\text{Grid}_n(M)$ and $\text{Geo}_n(M)$ are zigzag languages.
Binary trees
Binary trees

- Label vertices with $1, \ldots, n$ according to search tree property: for any vertex $i$, we have $L(i) < i < R(i)$.
Binary trees

- Label vertices with $1, \ldots, n$ according to search tree property: for any vertex $i$, we have $L(i) < i < R(i)$
- $T_n :=$ binary (search) trees with $n$ vertices
Binary trees

- Label vertices with $1, \ldots, n$ according to search tree property: for any vertex $i$, we have $L(i) < i < R(i)$
- $T_n :=$ binary (search) trees with $n$ vertices

**Theorem [Folklore]:** There is a bijection $f$ between $T_n$ and $S_n(231)$. 

![Binary tree diagram]
Binary trees

- Label vertices with $1, \ldots, n$ according to search tree property: for any vertex $i$, we have $L(i) < i < R(i)$
- $T_n :=$ binary (search) trees with $n$ vertices

**Theorem [Folklore]:** There is a bijection $f$ between $T_n$ and $S_n(231)$.

$$f(T) := (r(T), L(T), R(T))$$

‘preorder traversal’
## Binary trees

- Label vertices with $1, \ldots, n$ according to search tree property: for any vertex $i$, we have $L(i) < i < R(i)$
- $T_n := \text{binary (search) trees with } n \text{ vertices}$

**Theorem [Folklore]:** There is a bijection $f$ between $T_n$ and $S_n(231)$.

$$f(T) := (r(T), L(T), R(T))$$

‘preorder traversal’  

$f(T) = (6, 5, 2, 1, 4, 3, 9, 7, 8, 10)$
Binary trees

- Label vertices with $1, \ldots, n$ according to search tree property: for any vertex $i$, we have $L(i) < i < R(i)$
- $T_n :=$ binary (search) trees with $n$ vertices

**Theorem** [Folklore]: There is a bijection $f$ between $T_n$ and $S_n(231)$.

$$f(T) := (r(T), L(T), R(T))$$

‘preorder traversal’

$$f(T) = (6, 5, 2, 1, 4, 3, 9, 7, 8, 10)$$
Binary trees

- Label vertices with $1, \ldots, n$ according to search tree property: for any vertex $i$, we have $L(i) < i < R(i)$
- $T_n :=$ binary (search) trees with $n$ vertices

**Theorem [Folklore]:** There is a bijection $f$ between $T_n$ and $S_n(231)$. 

$f(T) := (r(T), L(T), R(T))$

'preorder traversal' \hspace{1cm} $f(T) = (6, 5, 2, 1, 4, 3, 9, 7, 8, 10)$

- $S_n(231)$ is a zigzag language, so Algorithm J applies
Binary trees

- Label vertices with $1, \ldots, n$ according to search tree property: for any vertex $i$, we have $L(i) < i < R(i)$

- $T_n :=$ binary (search) trees with $n$ vertices

**Theorem** [Folklore]: There is a bijection $f$ between $T_n$ and $S_n(231)$.

$$f(T) := (r(T), L(T), R(T))$$

‘preorder traversal’ $f(T) = (6, 5, 2, 1, 4, 3, 9, 7, 8, 10)$

- $S_n(231)$ is a zigzag language, so Algorithm J applies

**Theorem:** Under $f^{-1}$, minimal jumps of Algorithm J translate to tree rotations, i.e., we obtain a rotation Gray code for binary trees ($\leftrightarrow$ HP on associahedron).

$= \ [\text{Lucas, Roelants van Baronaigien, Ruskey 93}]$
Binary trees

• Label vertices with $1, \ldots, n$ according to search tree property: for any vertex $i$, we have $L(i) < i < R(i)$

• $T_n :=$ binary (search) trees with $n$ vertices

**Theorem** [Folklore]: There is a bijection $f$ between $T_n$ and $S_n(231)$.

$$f(T) := (r(T), L(T), R(T))$$

‘preorder traversal’ $f(T) = (6, 5, 2, 1, 4, 3, 9, 7, 8, 10)$

• $S_n(231)$ is a zigzag language, so Algorithm J applies

**Theorem:** Under $f^{-1}$, minimal jumps of Algorithm J translate to tree rotations, i.e., we obtain a rotation Gray code for binary trees ($\leftrightarrow$ HP on associahedron).

$= \text{[Lucas, Roelants van Baronaigien, Ruskey 93]}$
Patterns in binary trees

pattern tree

host tree

$P$

$T$
Patterns in binary trees

pattern tree       host tree

$P$ contains $T$

$T$ contains $P$
Patterns in binary trees

pattern tree

host tree

$P$

$T$

$T$ contains $P$

$T'$

$T'$ avoids $P$
Patterns in binary trees

Pattern tree  Host tree

contiguous

[Rowland 10]
Patterns in binary trees

contiguous
[Rowland 10]

non-contiguous
[Dairyko, Tyner, Pudwell, Wynn 12]
Patterns in binary trees

contiguous

[Rowland 10]

non-contiguous

[Dairyko, Tyner, Pudwell, Wynn 12]
Mixed tree patterns

mixed (new)

$P$ contains $T$ contains $P$
Mixed tree patterns

mixed \text{(new)}

$P$ contains $T$ avoids $P$

$T$ contains $P$

$T'$ avoids $P$
Theorem: For every (mixed) tree pattern, there is a permutation mesh pattern \( \tau(P) = (f(P), C) \) such that \( f : T_n(P) \rightarrow S_n(231, \tau(P)) \) is a bijection.
Theorem: For every (mixed) tree pattern, there is a permutation mesh pattern $\tau(P) = (f(P), C)$ such that $f : T_n(P) \rightarrow S_n(231, \tau(P))$ is a bijection.

- generalizes result of [Pudwell, Scholten, Schrock, Serrato 14]

mixed (new)
Mixed tree patterns

mixed (new)

Theorem: For every (mixed) tree pattern, there is a permutation mesh pattern $\tau(P) = (f(P), C)$ such that $f : T_n(P) \to S_n(231, \tau(P))$ is a bijection.

- generalizes result of [Pudwell, Scholten, Schrock, Serrato 14]
- classified all tree patterns on $\leq 5$ vertices; interesting bijections to pattern-avoiding lattice paths and set partitions
Tame patterns

• A pattern $P$ is **tame**, if the largest node is neither root nor leaf, and the right branch from the root is non-contiguous.
Tame patterns

- A pattern $P$ is **tame**, if the largest node is neither root nor leaf, and the right branch from the root is non-contiguous

**Theorem:** If $P_1, \ldots, P_k$ are tame patterns, then $f(T_n(P_1, \ldots, P_k))$ is a zigzag language. Under $f^{-1}$, minimal jumps of Algorithm J translate to sequences of rotations.
Tame patterns

- A pattern $P$ is **tame**, if the largest node is neither root nor leaf, and the right branch from the root is non-contiguous.

**Theorem:** If $P_1, \ldots, P_k$ are tame patterns, then $f(T_n(P_1, \ldots, P_k))$ is a zigzag language. Under $f^{-1}$, minimal jumps of Algorithm J translate to sequences of rotations.

→ see www.combos.org/btree
Generic rectangulations

- **Generic rectangulation**: subdivision of a square into $n$ rectangles s.t. no four rectangles meet.
Generic rectangulations

- **Generic rectangulation**: subdivision of a square into $n$ rectangles s.t. no four rectangles meet
Generic rectangulations

- **Generic rectangulation**: subdivision of a square into $n$ rectangles s.t. no four rectangles meet
Generic rectangulations

- **Generic rectangulation**: subdivision of a square into $n$ rectangles s.t. no four rectangles meet

- ‘combinatorial’ equivalence: only incidences between rectangles matter
Generic rectangulations

- **Generic rectangulation**: subdivision of a square into \( n \) rectangles s.t. no four rectangles meet

- ‘combinatorial’ equivalence: only incidences between rectangles matter
Generic rectangulations

- **Generic rectangulation**: subdivision of a square into \( n \) rectangles s.t. no four rectangles meet

- ‘combinatorial’ equivalence: only incidences between rectangles matter

- \( R_n := \) set of all rectangulations with \( n \) rectangles
Generic rectangulations

- **Generic rectangulation**: subdivision of a square into $n$ rectangles s.t. no four rectangles meet

  
  ![Generic rectangulation diagrams]

  - ‘combinatorial’ equivalence: only incidences between rectangles matter

- $R_n :=$ set of all rectangulations with $n$ rectangles

  
  $R_3 =$
Generic rectangulations

**Theorem** [Reading 12]: There is a bijection $f$ between $R_n$ and $S_n(35124, 35142, 24513, 42513)$ (2-clumped permutations).
Generic rectangulations

**Theorem** [Reading 12]: There is a bijection $f$ between $R_n$ and $S_n(35124, 35142, 24513, 42513)$ (2-clumped permutations).

is a zigzag language, so Algorithm J applies
Generic rectangulations

**Theorem** [Reading 12]: There is a bijection $f$ between $R_n$ and $S_n(35124, 35142, 24513, 42513)$ (2-clumped permutations).

is a zigzag language, so Algorithm J applies

**Theorem:** Under $f^{-1}$, minimal jumps of Algorithm J translate to rectangle flips, i.e., we obtain a flip Gray code for generic rectangulations ($\leftrightarrow$ HC on quotientope).
Generic rectangulations

Theorem [Reading 12]: There is a bijection $f$ between $R_n$ and $S_n(35124, 35142, 24513, 42513)$ (2-clumped permutations).

Theorem: Under $f^{-1}$, minimal jumps of Algorithm J translate to rectangle flips, i.e., we obtain a flip Gray code for generic rectangulations ($\leftrightarrow$ HC on quotientope).
Flip Gray code

\[ n = 3 \]

![Graph for n = 3 with 123, 132, 312, 321, 231, 213]

\[ n = 4 \]

![Graph for n = 4 with 1234, 1243, 1423, 4123, 4132, 1432, 1342, 1324, 3124, 3142, 3412, 4312, 4321, 3421, 3241, 3214, 2314, 2341, 2431, 4231, 4213, 2413, 2143, 2134]
Patterns in rectangulations

- **Segment**: maximal sequence of inner edges
Patterns in rectangulations

- **Segment**: maximal sequence of inner edges

- **Pattern**: connected configuration of segments
Patterns in rectangulations

- **Segment**: maximal sequence of inner edges

- **Pattern**: connected configuration of segments

$P = \begin{array}{c}
\includegraphics[width=0.2\textwidth]{pattern1.png} \\
\includegraphics[width=0.2\textwidth]{pattern2.png}
\end{array}$

can be seen as a rectangulation itself
Patterns in rectangulations

• **Segment**: maximal sequence of inner edges

• **Pattern**: connected configuration of segments

\[ P = \]

\[ P \]

can be seen as a rectangulation itself

contains \( P \)
Tame patterns

• A pattern $P$ is tame, if the bottom right corner rectangle does not stretch across the whole bottom or right side.
Tame patterns

- A pattern $P$ is tame, if the bottom right corner rectangle does not stretch across the whole bottom or right side.
Tame patterns

- A pattern $P$ is **tame**, if the bottom right corner rectangle does not stretch across the whole bottom or right side
Tame patterns

- A pattern $P$ is **tame**, if the bottom right corner rectangle does not stretch across the whole bottom or right side.
Tame patterns

- A pattern $P$ is **tame**, if the bottom right corner rectangle does not stretch across the whole bottom or right side.

![Tame patterns examples](image)
Tame patterns

- A pattern $P$ is **tame**, if the bottom right corner rectangle does not stretch across the whole bottom or right side.

**Theorem:** If $P_1, \ldots, P_k$ are tame patterns, then $f(R_n(P_1, \ldots, P_k))$ is a zigzag language. Under $f^{-1}$, minimal jumps of Algorithm J translate to sequences of rectangle flips.
Examples

\[ R_n(\begin{array}{c}
\begin{array}{c}
\text{(a) } X
\end{array}
\end{array}, \begin{array}{c}
\begin{array}{c}
\text{(b) } Y
\end{array}
\end{array}) \]

diagonal rectangulations
Examples

\[ R_n(\begin{array}{cc}
\cdot & \cdot \\
\cdot & \cdot 
\end{array}, \begin{array}{cc}
\cdot & \cdot \\
\cdot & \cdot 
\end{array}) \]

\( \rightarrow \) diagonal rectangulations

\( \leftrightarrow \) HC on quotientope
Examples

$R_n(\begin{array}{c|c|c} & & \\ \hline & & \\ \hline & & \\ \hline \end{array})$ \quad diagonal rectangulations

$\quad \leftrightarrow \quad$ HC on quotientope

$R_n(\begin{array}{c|c|c|c} & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array})$ \quad area-universal rectangulations

[Eppstein, Mumford, Speckmann, Verbeek 2012]
Examples

\( R_n(\text{diagonal rectangulations}) \quad \rightarrow \text{HC on quotientope} \)

\( R_n(\text{area-universal rectangulations}) \quad [\text{Eppstein, Mumford, Speckmann, Verbeek 2012}] \)

\( R_n(\text{guillotine rectangulations}) \)
Examples

$R_n\left(\begin{array}{c}
    \Box \\
    \Box \\
\end{array}\right)$ diagonal rectangulations $\rightarrow$ HC on quotientope

$R_n\left(\begin{array}{c}
    \Box \\
    \Box \\
    \Box \\
    \Box \\
\end{array}\right)$ area-universal rectangulations

$[\text{Eppstein, Mumford, Speckmann, Verbeek 2012}]$

$R_n\left(\begin{array}{c}
    \Box \\
    \Box \\
\end{array}\right)$ guillotine rectangulations

$R_n\left(\begin{array}{c}
    \Box \\
\end{array}\right)$ Catalan staircases

$[\text{Downing, Einstein, Hartung, Williams 2023}]$
Examples

\[ R_n(\begin{array}{c|c}
\hline
\end{array}) \]
\[ \rightarrow \]
diagonal rectangulations

\[ R_n(\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
\hline
\end{array}) \]
\[ \rightarrow \]
area-universal rectangulations

\[ R_n(\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
\hline
\end{array}) \]
\[ \rightarrow \]
guillotine rectangulations

\[ R_n(\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
\hline
\end{array}) \]
\[ \rightarrow \]
Catalan staircases

\[ C_n \]
\[ \rightarrow \]
HP on associahedron

[Eppstein, Mumford, Speckmann, Verbeek 2012]
[Downing, Einstein, Hartung, Williams 2023]
Examples

\[ R_n(\square_1, \square_2) \]

- **diagonal rectangulations**
- \(\rightarrow\) **HC on quotientope**

\[ R_n(\square_1, \square_2, \square_3, \square_4) \]

- **area-universal rectangulations**
- [Eppstein, Mumford, Speckmann, Verbeek 2012]

\[ R_n(\square_1, \square_2) \]

- **guillotine rectangulations**

\[ R_n(\square_1) \]

- **Catalan staircases**
- \(C_n\)
- [Downing, Einstein, Hartung, Williams 2023]
- \(\rightarrow\) **HP on associahedron**

\[ R_n(\square_1, \square_2) \]

- **stacked rectangulations**
- \(2^n\)
Examples

$R_n(\begin{array}{c|c|c} & & \\ \hline & & \\ \hline \end{array})$  \hspace{1cm} diagonal rectangulations

$R_n(\begin{array}{c|c|c|c} & & & \\ \hline & & & \\ \hline \end{array})$  \hspace{1cm} area-universal rectangulations

$R_n(\begin{array}{c|c|c} & & \\ \hline & & \\ \hline \end{array})$  \hspace{1cm} guillotine rectangulations

$R_n(\begin{array}{c|c} & \\ \hline & \\ \hline \end{array})$  \hspace{1cm} Catalan staircases

$R_n(\begin{array}{c|c} & \\ \hline & \\ \hline \end{array})$  \hspace{1cm} stacked rectangulations

$\rightarrow$ HC on quotientope

$\leftrightarrow$ HC on hypercube

[Eppstein, Mumford, Speckmann, Verbeek 2012]

[Downing, Einstein, Hartung, Williams 2023]
Examples

\[ R_n(\begin{array}{c}
  \text{\textbullet} \\
  \text{\textbullet} \\
\end{array},
\begin{array}{c}
  \text{\textbullet} \\
  \text{\textbullet} \\
\end{array}) \]

diagonal rectangulations

\[ \rightarrow \text{HC on quotientope} \]

\[ R_n(\begin{array}{c}
  \text{\textbullet} \\
  \text{\textbullet} \\
  \vdots
\end{array},
\begin{array}{c}
  \text{\textbullet} \\
  \text{\textbullet} \\
  \vdots
\end{array},
\begin{array}{c}
  \text{\textbullet} \\
  \text{\textbullet} \\
\end{array},
\begin{array}{c}
  \text{\textbullet} \\
  \text{\textbullet} \\
\end{array}) \]

area-universal rectangulations

\[ \text{[Eppstein, Mumford, Speckmann, Verbeek 2012]} \]

\[ R_n(\begin{array}{c}
  \text{\textbullet} \\
  \text{\textbullet} \\
\end{array},
\begin{array}{c}
  \text{\textbullet} \\
  \text{\textbullet} \\
\end{array}) \]

guillotine rectangulations

\[ \rightarrow \text{HP on associahedron} \]

\[ R_n(\begin{array}{c}
  \text{\textbullet} \\
\end{array}) \]

Catalan staircases \( C_n \)

\[ \text{[Downing, Einstein, Hartung, Williams 2023]} \]

\[ R_n(\begin{array}{c}
  \text{\textbullet} \\
  \text{\textbullet} \\
\end{array},
\begin{array}{c}
  \text{\textbullet} \\
\end{array}) \]

stacked rectangulations \( 2^n \)

\[ \rightarrow \text{HC on hypercube} \]

\[ \rightarrow \text{see www.combos.org/rect} \]
Open questions

• Generating functions for mixed tree patterns?
Open questions

- Generating functions for mixed tree patterns?
- Third notion of edge type in tree patterns
Open questions

- Generating functions for mixed tree patterns?
- Third notion of edge type in tree patterns
Open questions

• Generating functions for mixed tree patterns?
• Third notion of edge type in tree patterns
• Does every rectangulation pattern correspond to a mesh permutation pattern? → [Asinowski, Cardinal, Felsner, Fusy PP23]
Open questions

- Generating functions for mixed tree patterns?
- Third notion of edge type in tree patterns
- Does every rectangulation pattern correspond to a mesh permutation pattern? → [Asinowski, Cardinal, Felsner, Fusy PP23]
- Applications of the generation framework to other (pattern-avoiding) combinatorial objects
Thank you!