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)

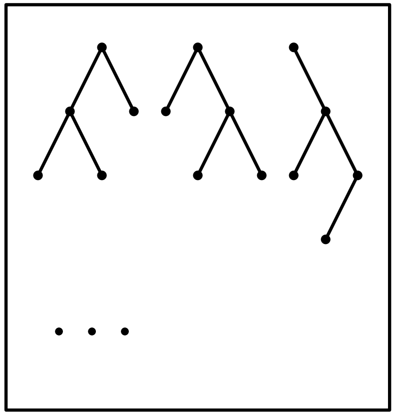





Permutation Patterns 2023

Introduction

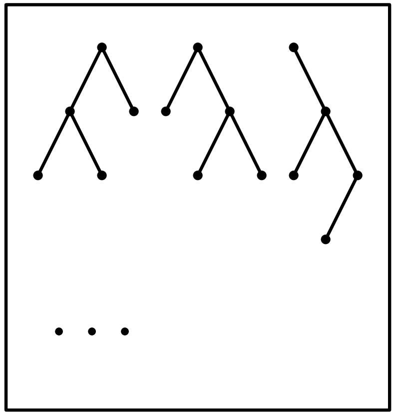
- many different classes of combinatorial objects



binary trees

Introduction

- many different classes of combinatorial objects



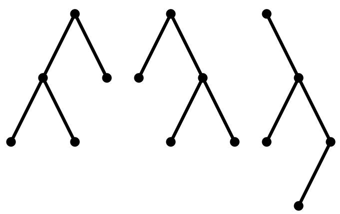
binary trees

123
132
312
321
...

permutations

Introduction

- many different classes of combinatorial objects



...

binary trees

123
132
312
321
...

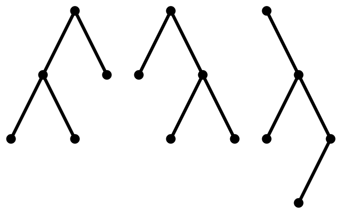
permutations

000
001
010
011
...

bitstrings

Introduction

- many different classes of combinatorial objects



. . .

binary trees

123
132
312
321
. . .

permutations

000
001
010
011
. . .

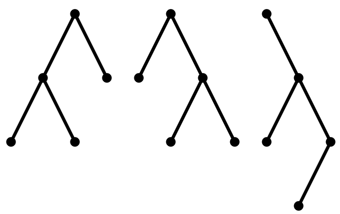
bitstrings

$\{1,2,3,4\}$
$\{1,2,3\}\{4\}$
$\{1,2\}\{3,4\}$
$\{1,2\}\{3\}\{4\}$
. . .

set partitions

Introduction

- many different classes of combinatorial objects



. . .

binary trees

123
132
312
321
. . .

permutations

000
001
010
011
. . .

bitstrings

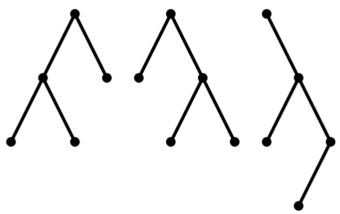
$\{1,2,3,4\}$
$\{1,2,3\}\{4\}$
$\{1,2\}\{3,4\}$
$\{1,2\}\{3\}\{4\}$
. . .

set partitions

- fundamental tasks:
  counting, sampling, optimization

Introduction

- many different classes of combinatorial objects



...

binary trees

123
132
312
321
...

permutations

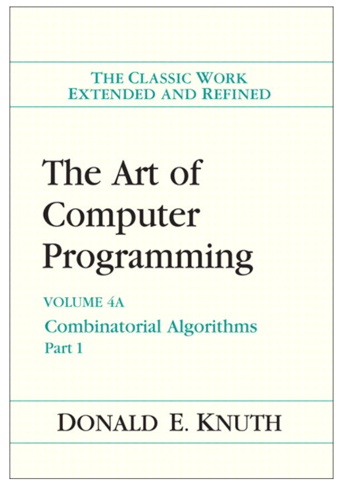
000
001
010
011
...

bitstrings

$\{1,2,3,4\}$
$\{1,2,3\}\{4\}$
$\{1,2\}\{3,4\}$
$\{1,2\}\{3\}\{4\}$
...

set partitions

- 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' $\rightarrow$ **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' $\rightarrow$ **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' $\rightarrow$ **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' $\rightarrow$ **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' $\rightarrow$ **Gray code**

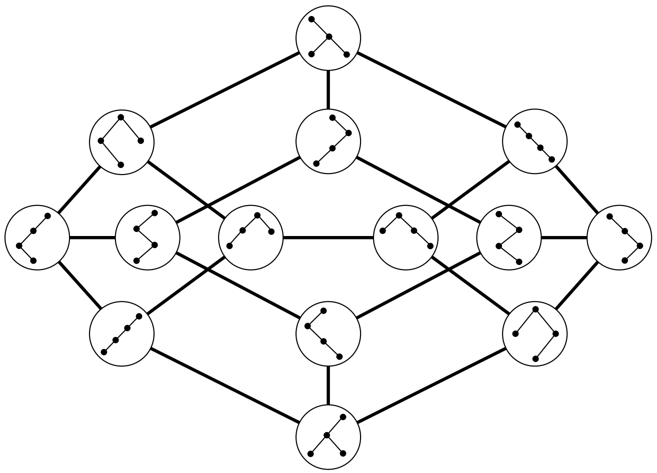
- **Examples:**
  - binary trees by **rotations** **[Lucas, Roelants van Baronaigien, Ruskey 93]**
  - permutations by **adjacent transpositions** (Steinhaus-Johnson-Trotter algorithm) **[Johnson 64], [Trotter 62]**
  - bitstrings by **bitflips** (Binary reflected Gray code) **[Gray 53]**
  - set partitions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

Flip graphs, lattices & polytopes

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

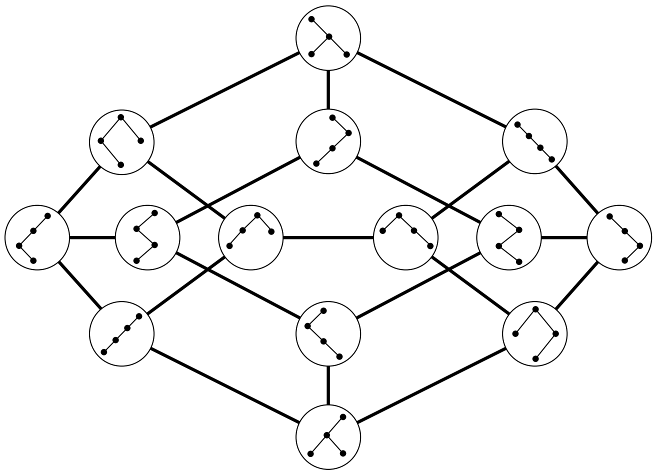
Flip graphs, lattices & polytopes

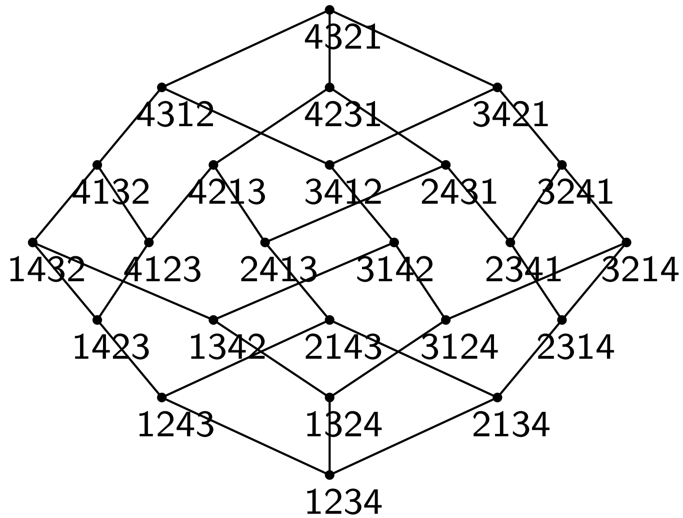
- **Flip graph:** vertices are combinatorial objects, edges capture change operations



Flip graphs, lattices & polytopes

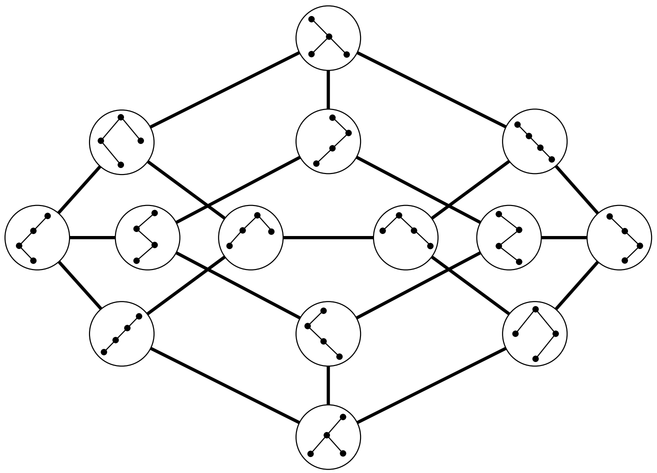
- **Flip graph:** vertices are combinatorial objects, edges capture change operations

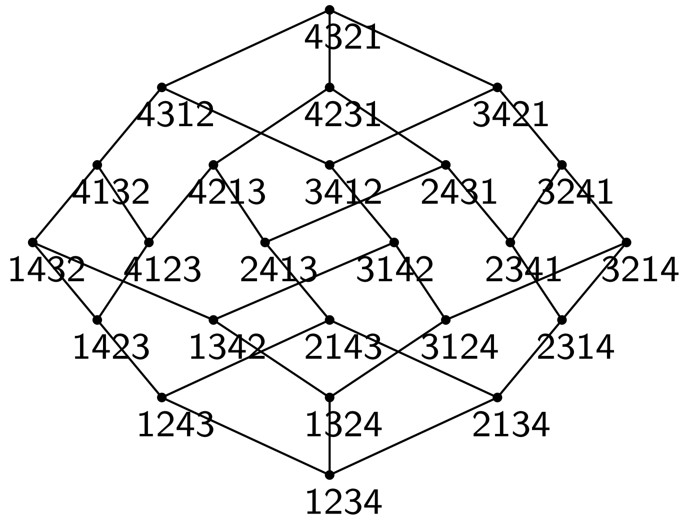


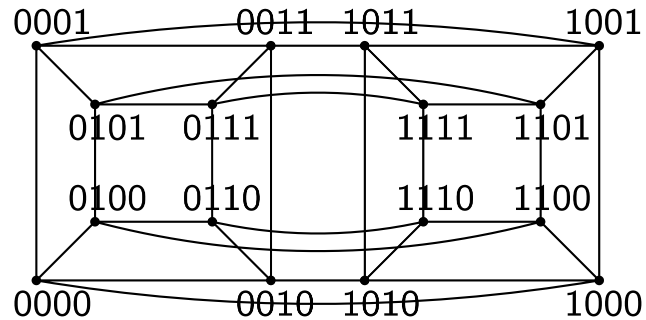


Flip graphs, lattices & polytopes

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

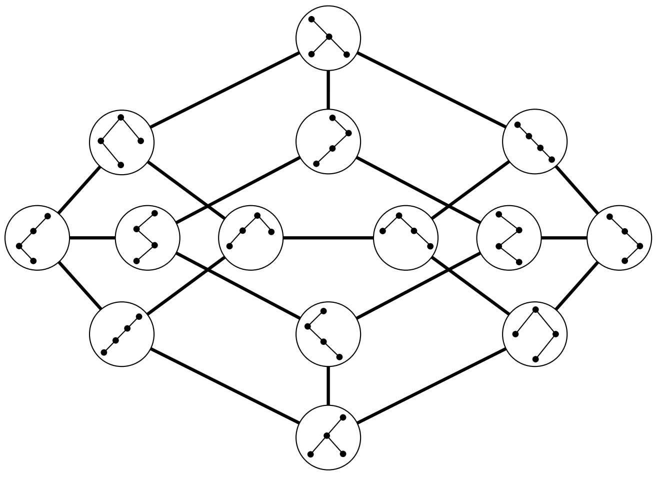


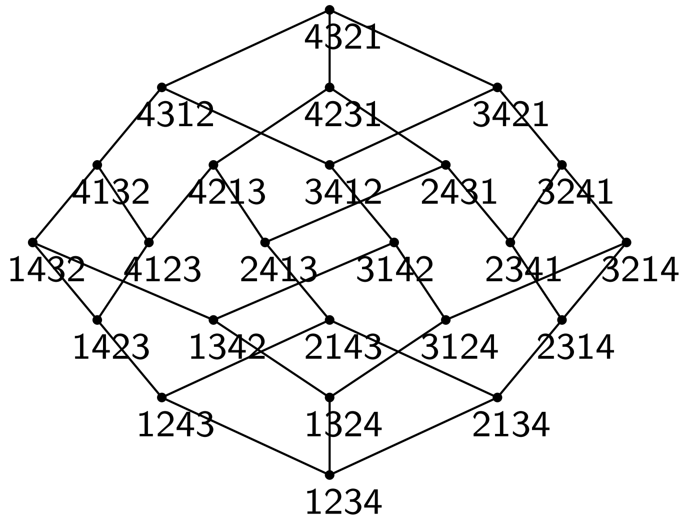


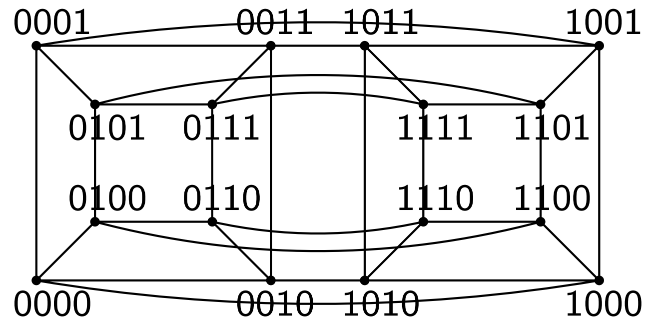


Flip graphs, lattices & polytopes

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



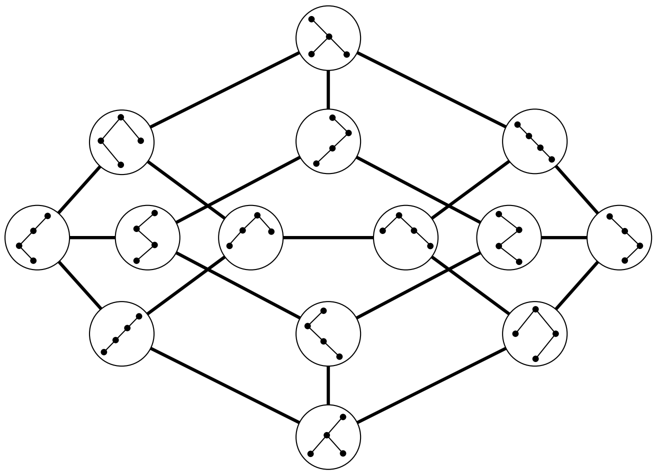




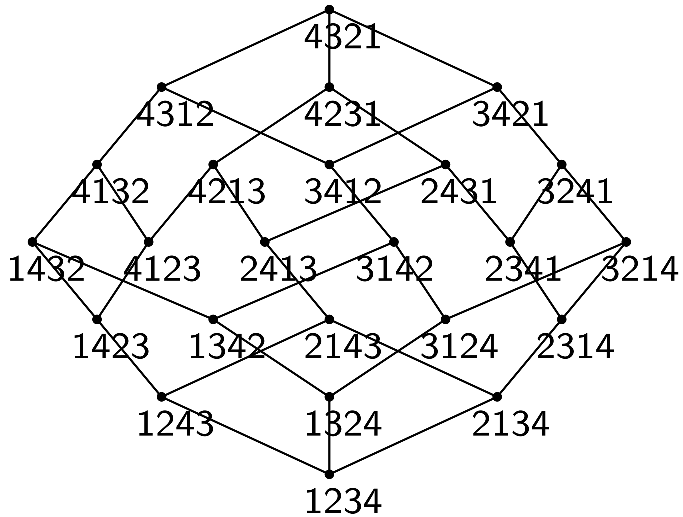
- many flip graphs can be equipped with a poset structure and realized as polytopes

Flip graphs, lattices & polytopes

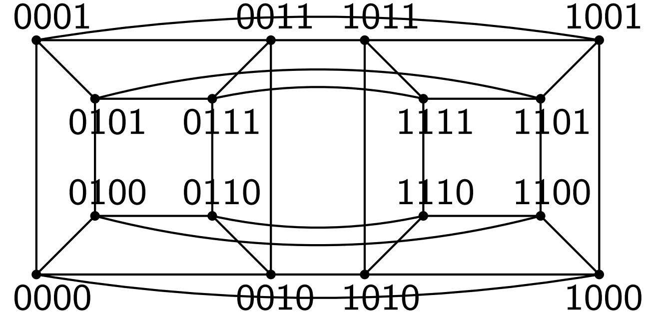
- **Flip graph:** vertices are combinatorial objects, edges capture change operations



Tamari lattice / associahedron



weak order / permutahedron

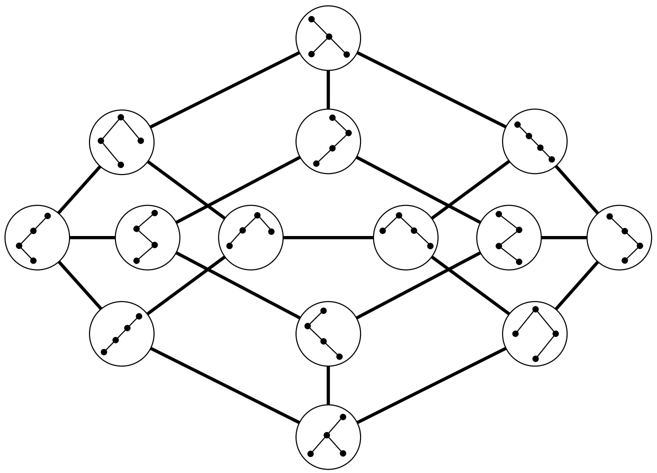


Boolean lattice / hypercube

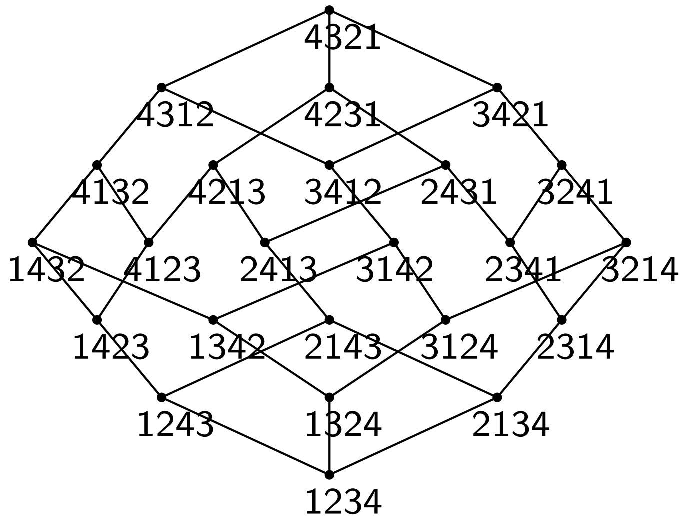
- many flip graphs can be equipped with a poset structure and realized as polytopes

Flip graphs, lattices & polytopes

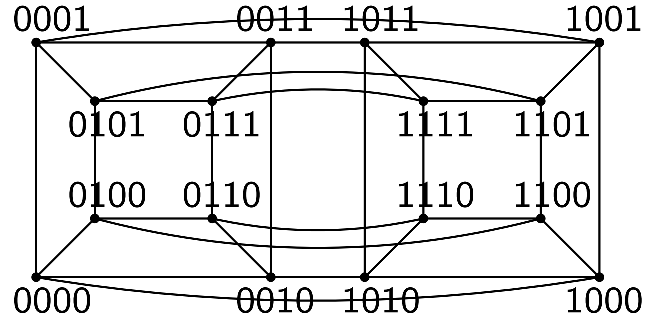
- **Flip graph:** vertices are combinatorial objects, edges capture change operations



Tamari lattice / associahedron



weak order / permutahedron

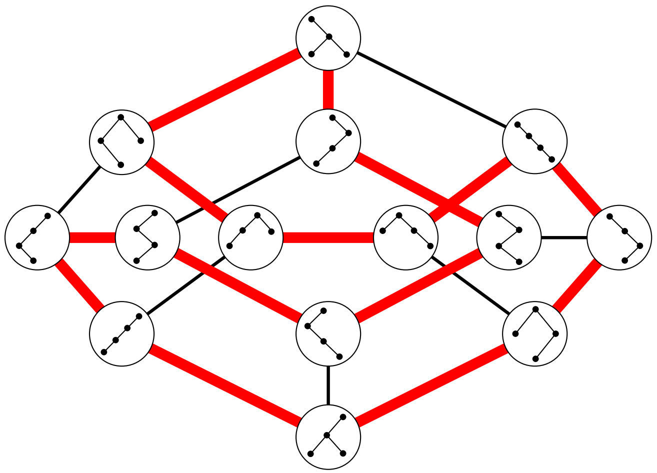


Boolean lattice / hypercube

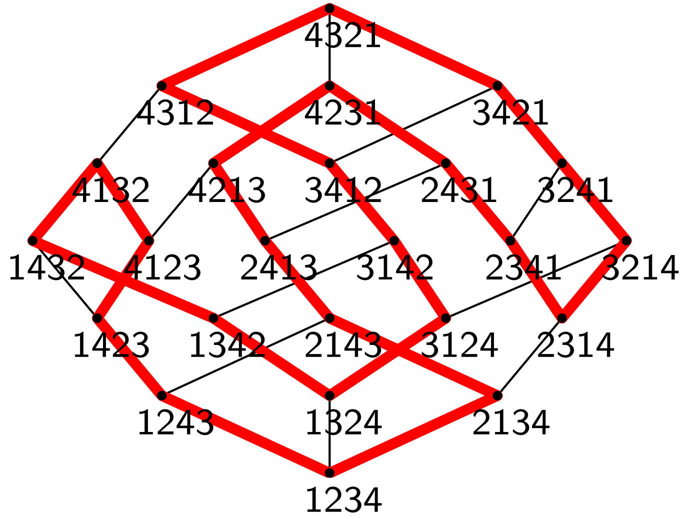
- many flip graphs can be equipped with a poset structure and realized as polytopes
- exhaustive generation $\hookrightarrow$ **Hamilton path (HP)/cycle (HC)**

Flip graphs, lattices & polytopes

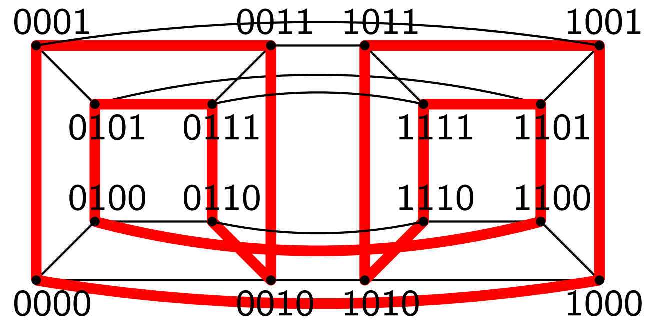
- **Flip graph:** vertices are combinatorial objects, edges capture change operations



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 $\hookrightarrow$ **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)

$$4 \ 5 \ 1 \ 3 \ 2 \ 6$$

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

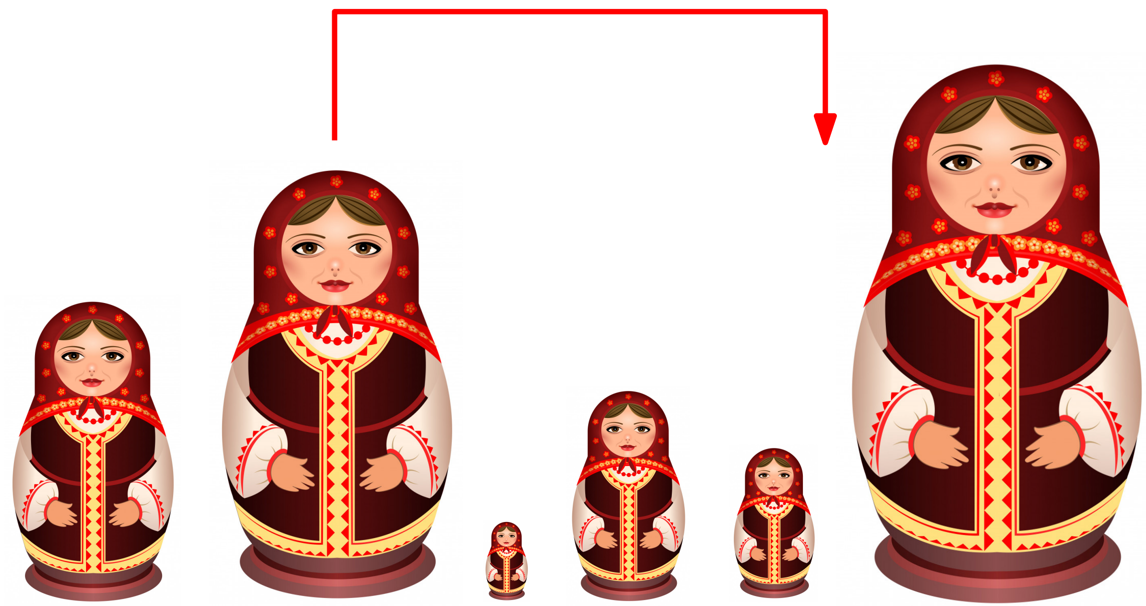
- **Jump:=** move an entry in the permutation across some neighboring smaller entries (left or right)

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)

$$4\ 5\ 1\ 3\ 2\ 6$$
$$4\ 1\ 3\ 2\ 5\ 6$$

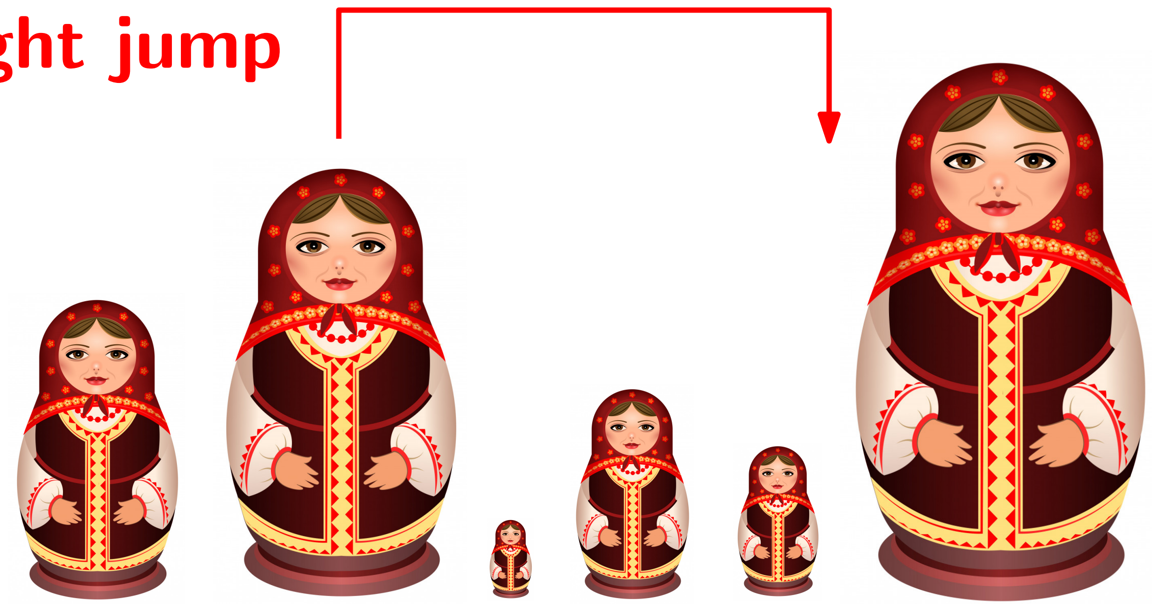


Jumps

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

$$\begin{matrix} 4 & 5 & 1 & 3 & 2 & 6 \\ 4 & 1 & 3 & 2 & 5 & 6 \end{matrix}$$

right jump



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

right jump

Jumps

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

$$\begin{array}{cccccc} 4 & 5 & 1 & 3 & 2 & 6 \\ 4 & 1 & 3 & 2 & 5 & 6 \end{array}$$

right jump

Jumps

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

$$\begin{array}{cccccc} 4 & \color{red}{5} & \color{blue}{1} & \color{blue}{3} & \color{blue}{2} & 6 \\ 4 & \color{blue}{1} & \color{blue}{3} & \color{blue}{2} & \color{red}{5} & 6 \end{array}$$

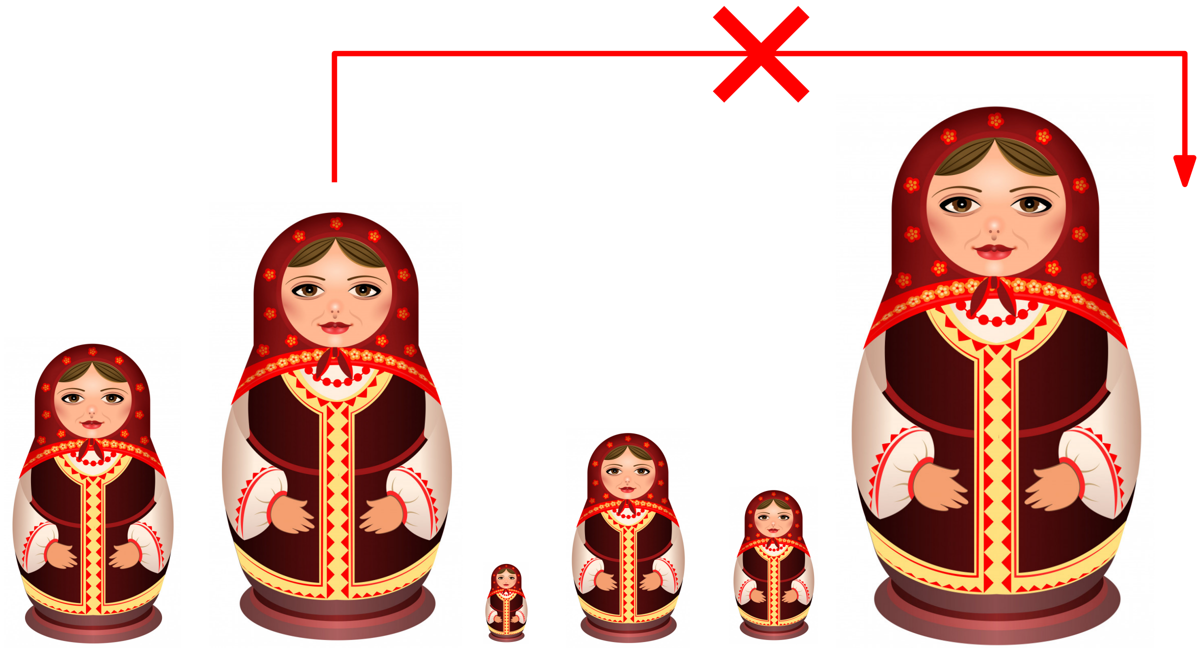
left jump

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



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.

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

$$6\ 7\ 1\ 3\ 2\ 4\ 5\ 8$$

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

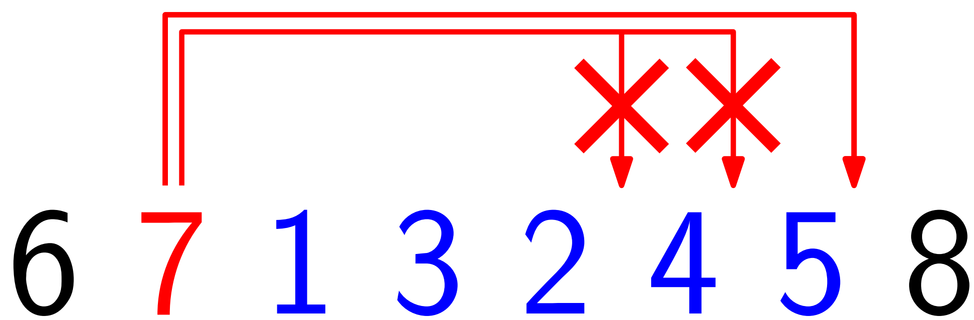
6 7 1 3 2 4 5 8

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

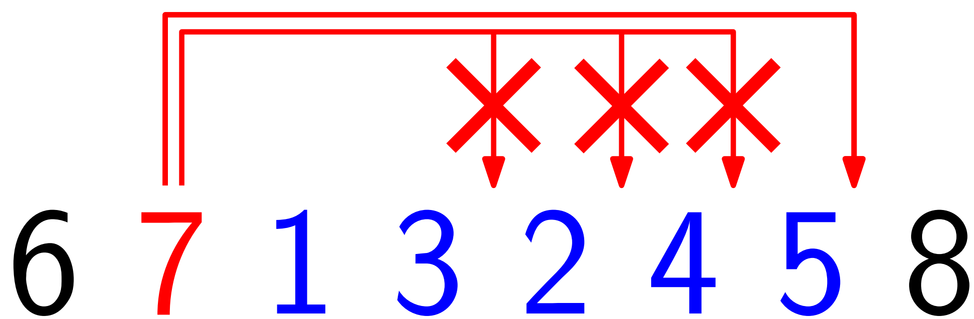


Algorithm J

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

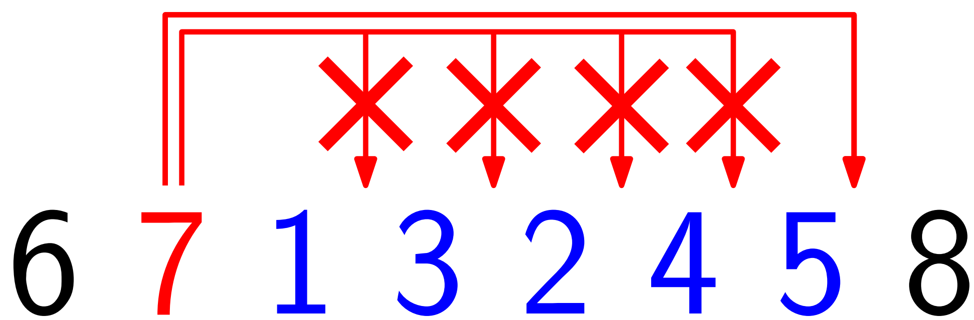


Algorithm J

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



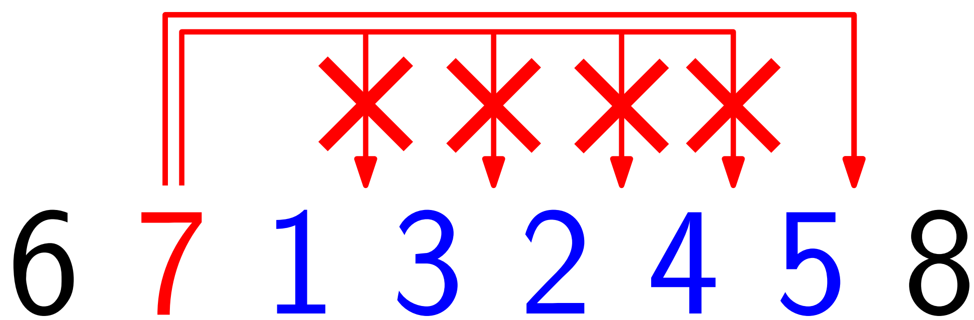
Algorithm J

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

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

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

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

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

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

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

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

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

1243
1423
4123

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

1243
1423
4123

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

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

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

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

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

1243
1423
4123
4213
2134

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

1243
1423
4123
4213
2134 ✓

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

1243
1423
4123
4213
2134

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

1243	**4213**
1423	**2134** ✗
4123	no jump possible
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:** $F_4 = \{1243, \underline{1423}, 2134, 4123, 4213\}$

1243
1423
4123
4213
2134

4213
2134

no jump possible

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

1243
1423
4123
4213
2134

4213
2134

no jump possible

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

1243
1423
4123
4213
2134 ✔

4213
2134 ✘
no jump possible

1423 ✘
direction ambiguous

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 $F_n \subseteq S_n$

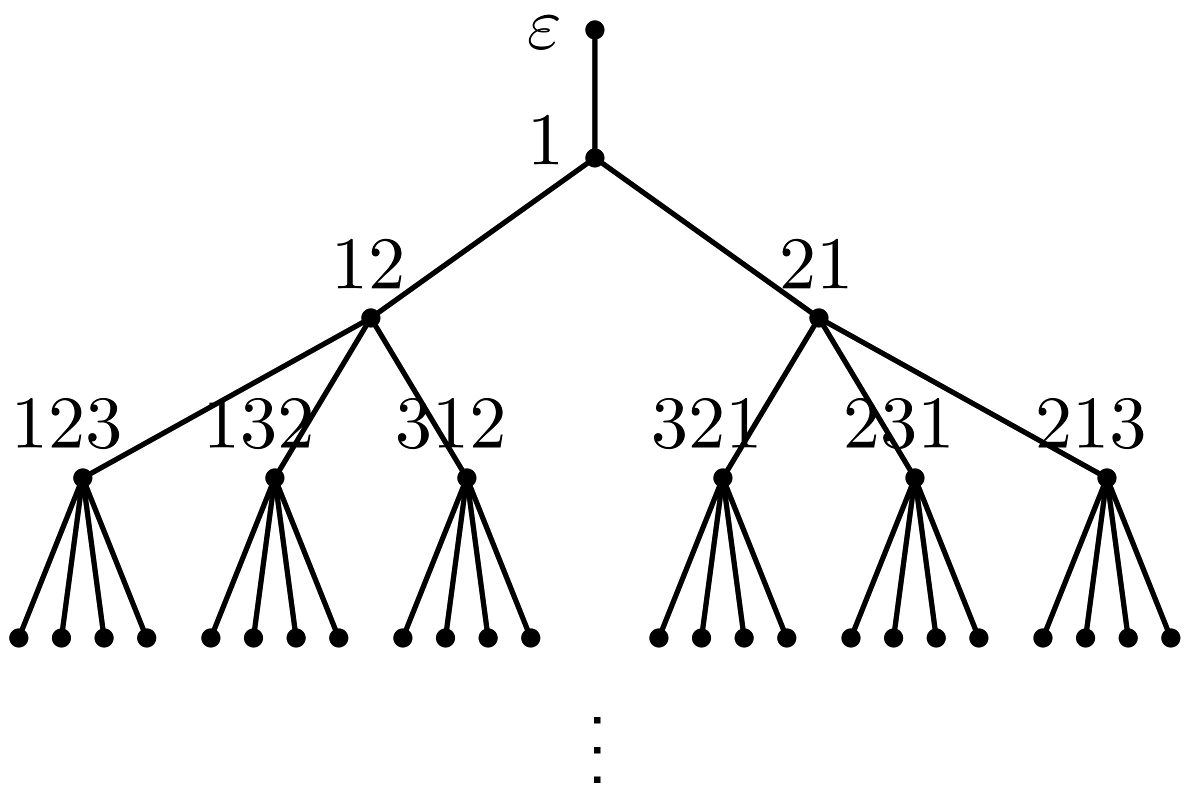
- Start with an initial permutation.
- In the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from F_n is created.

  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

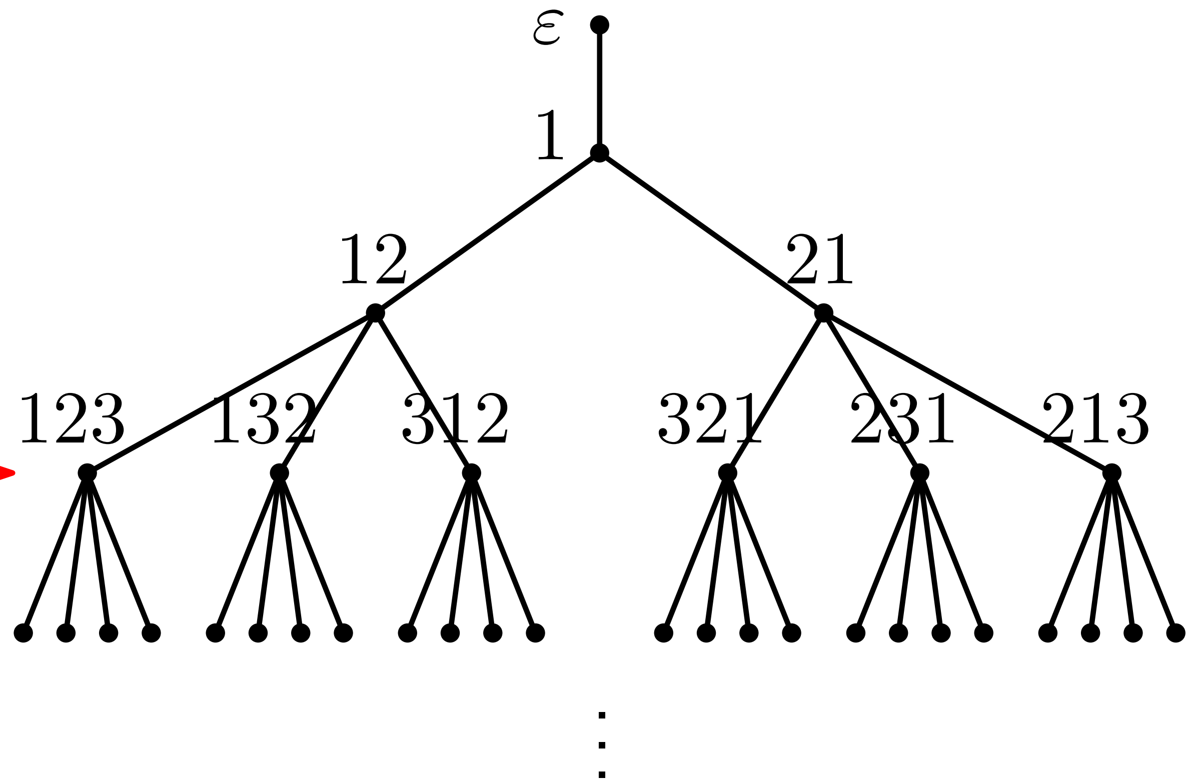
- root := empty permutation ε
- given a permutation length $n-1$, its children are obtained by inserting n in every possible position



Tree of permutations

- root := empty permutation ε
- given a permutation length $n-1$, its children are obtained by inserting n in every possible position

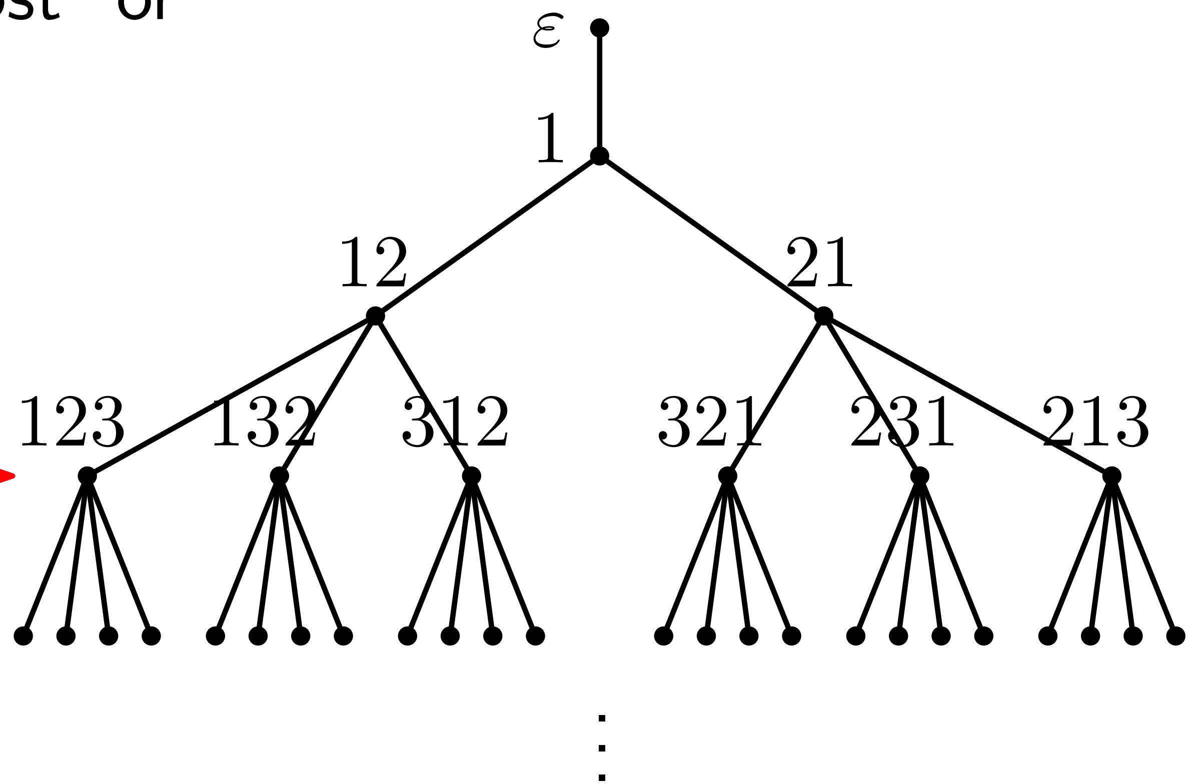
depth $n =$
all permutations
of length n



Tree of permutations

- root := empty permutation ε
- 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

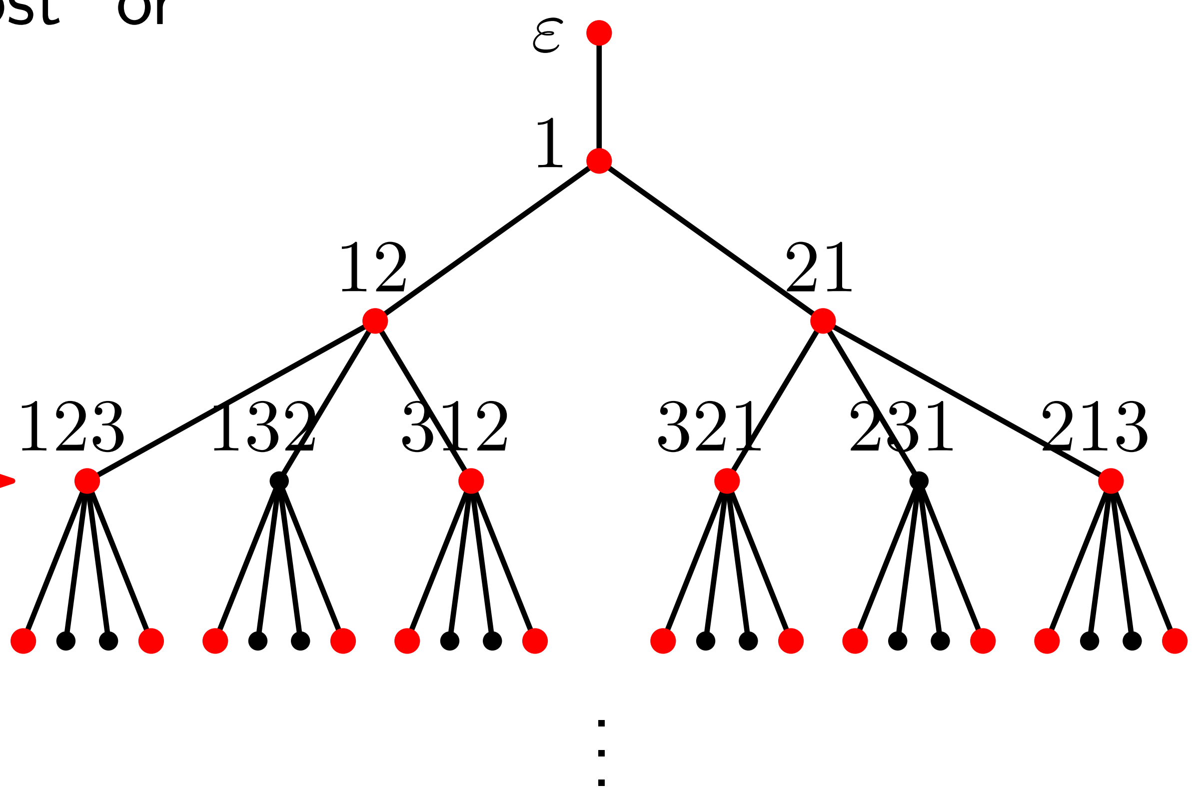
depth n =
all permutations
of length n



Tree of permutations

- root := empty permutation ε
- 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

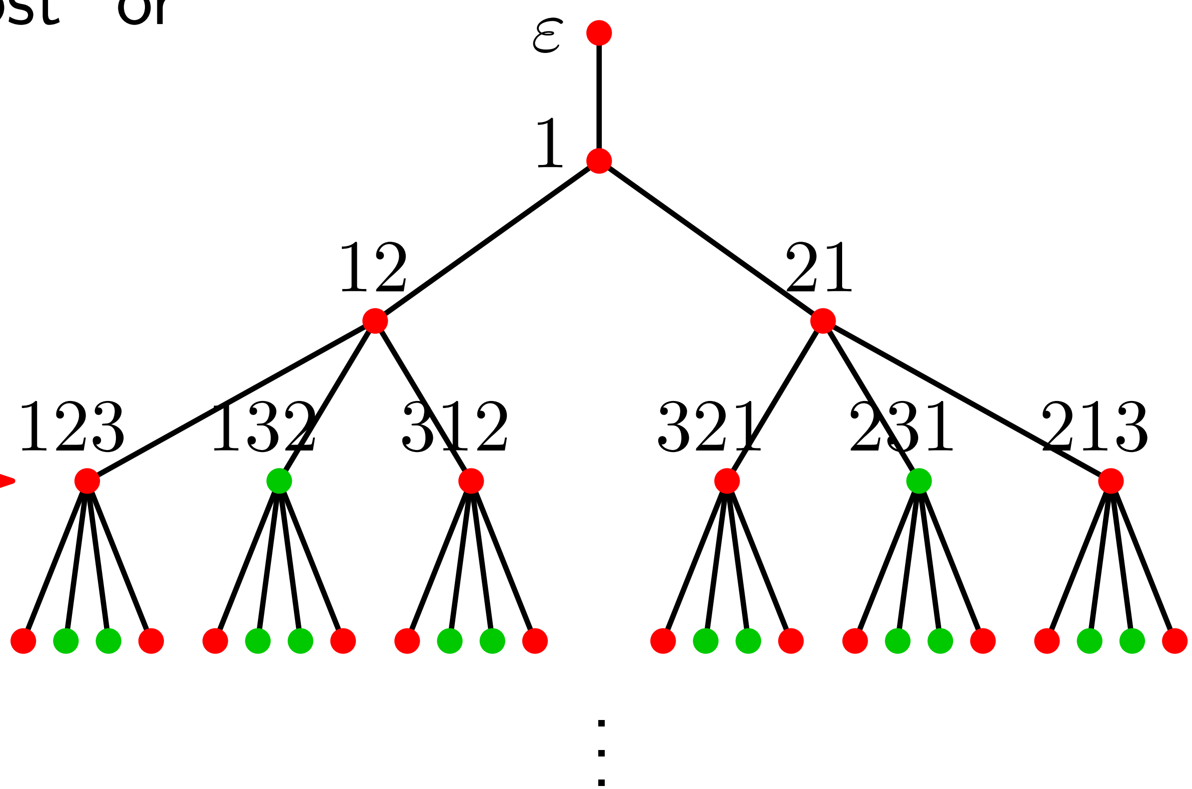
depth n = all permutations of length n



Tree of permutations

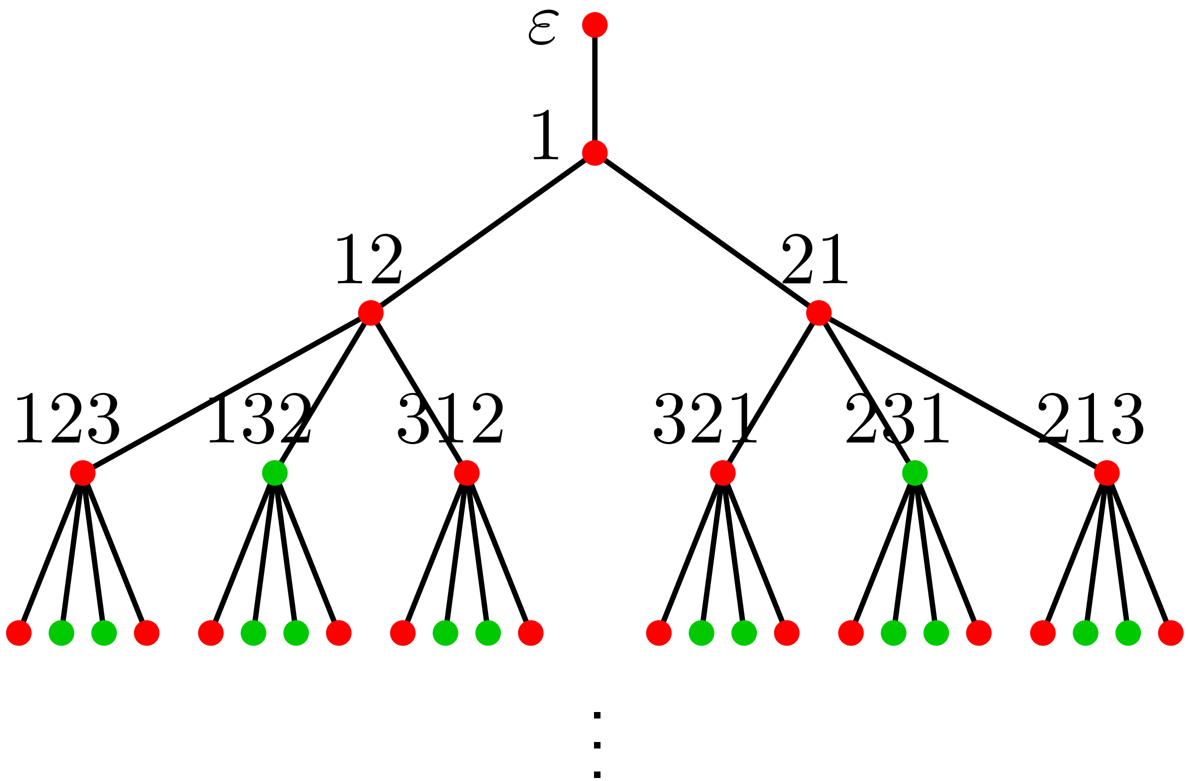
- root := empty permutation ε
- given a permutation length $n-1$, its children are obtained by inserting n in every possible position
  - symbol n at leftmost or rightmost position
  - else

depth $n =$ all permutations of length n



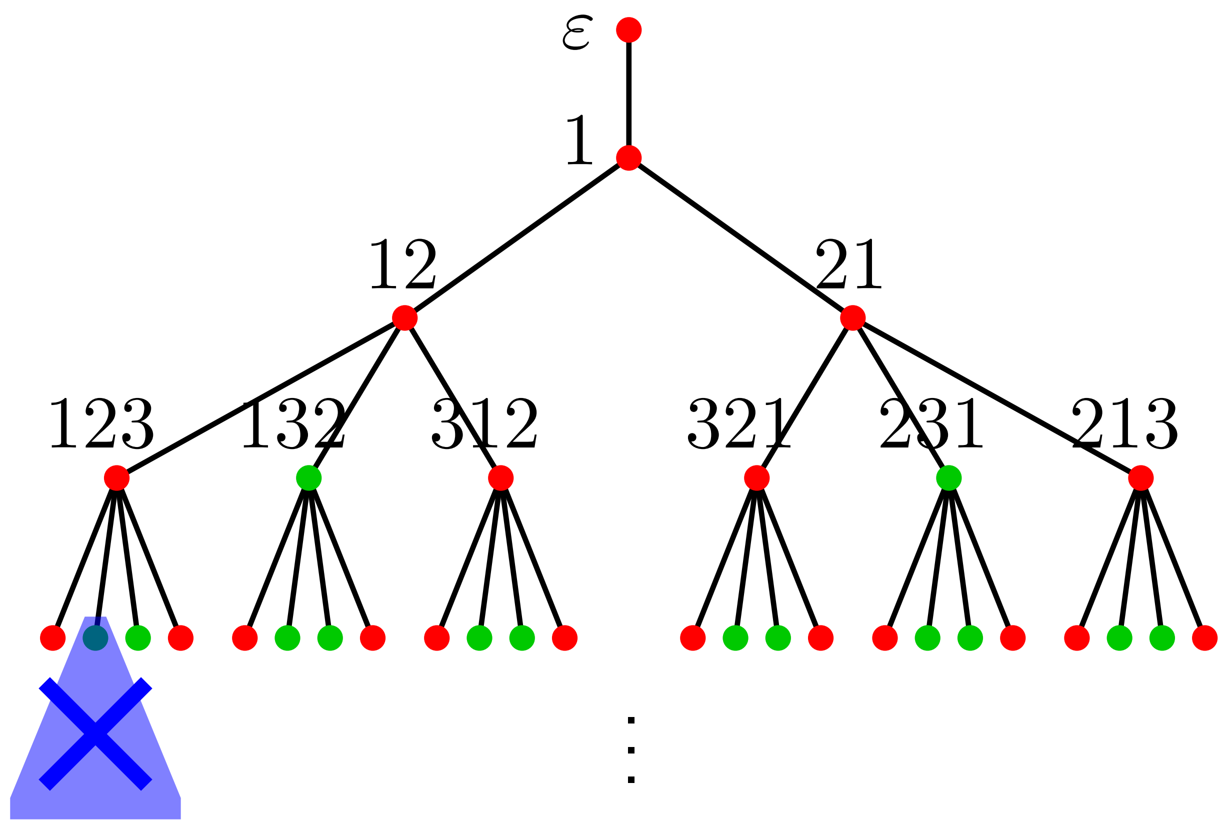
Tree of permutations

- we may prune subtrees iff their root is ●



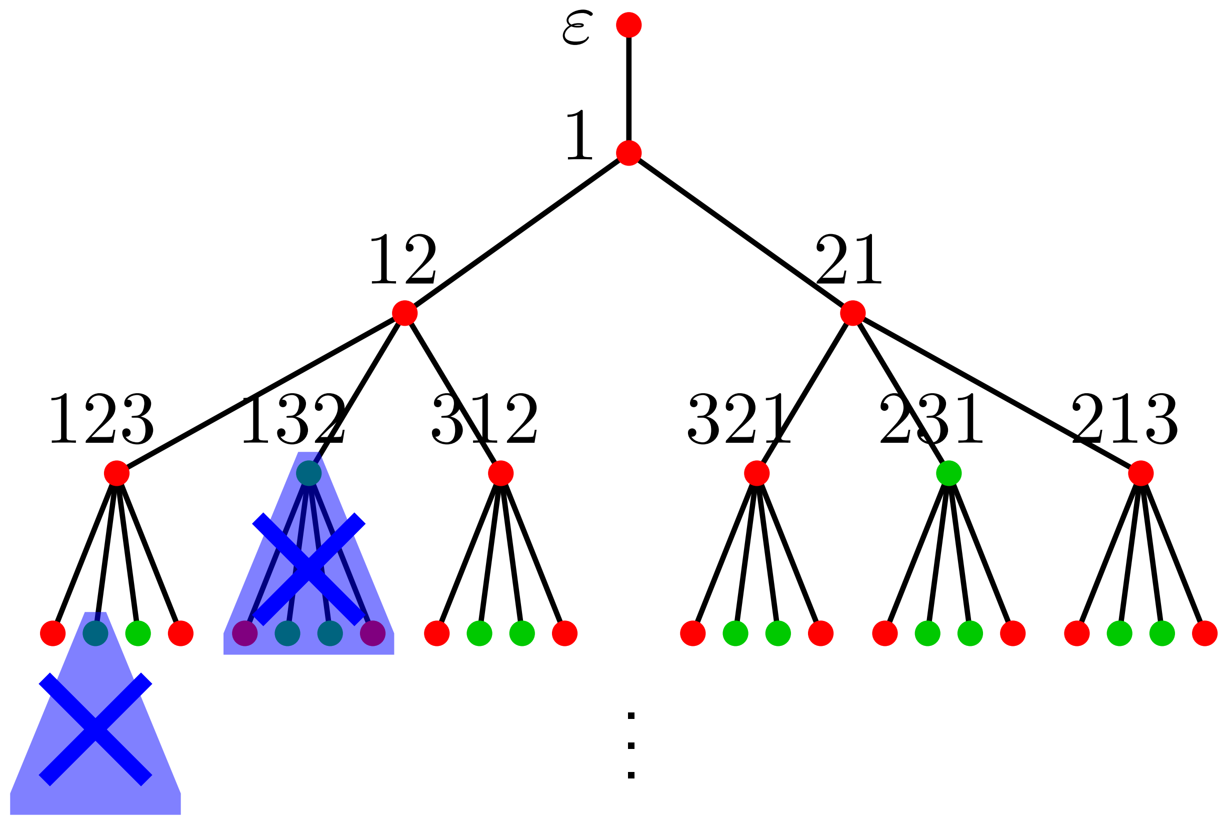
Tree of permutations

- we may prune subtrees iff their root is ●



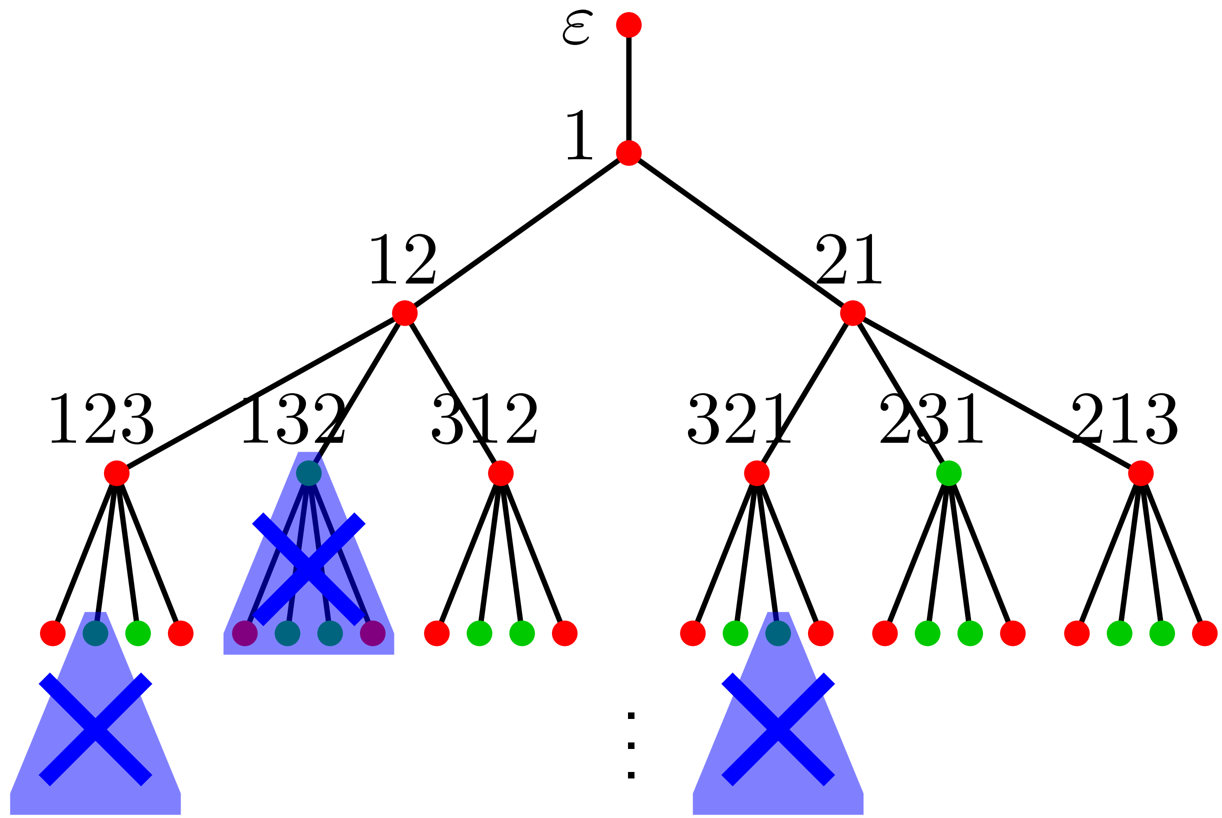
Tree of permutations

- we may prune subtrees iff their root is



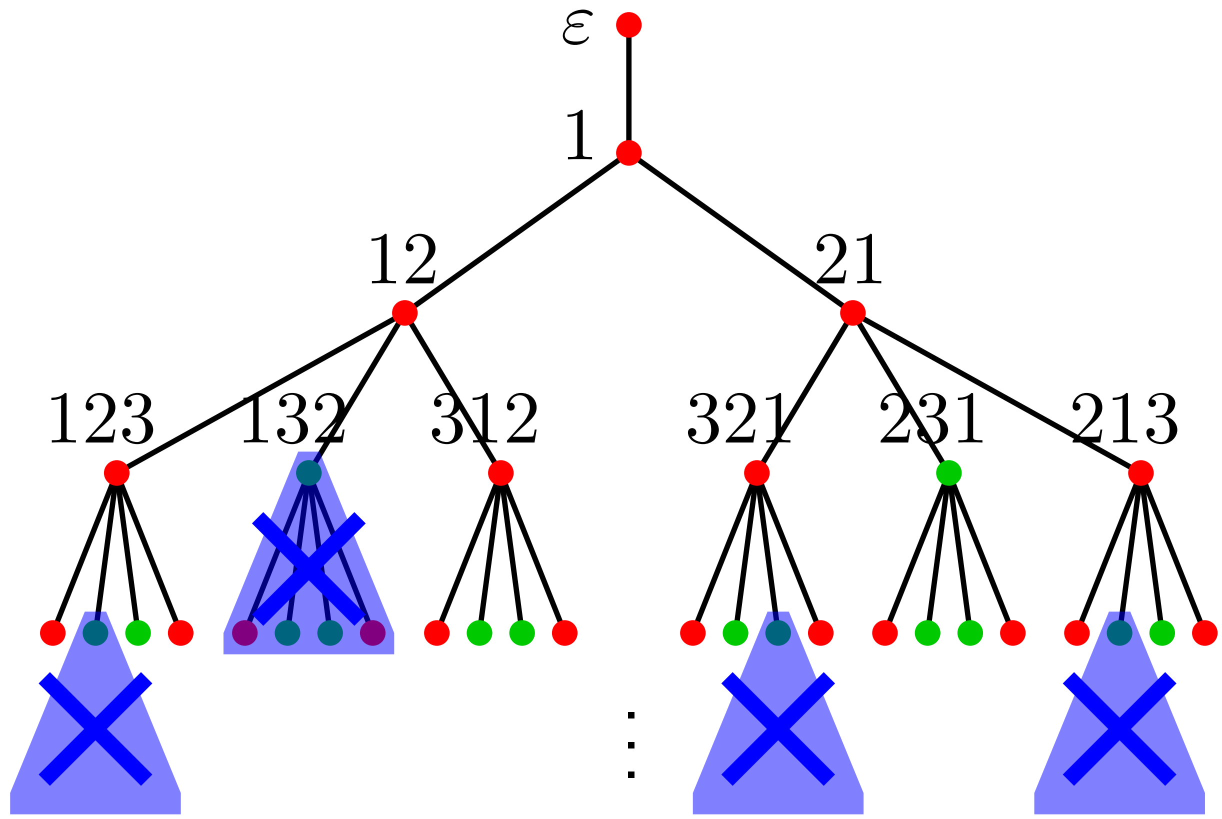
Tree of permutations

- we may prune subtrees iff their root is ●



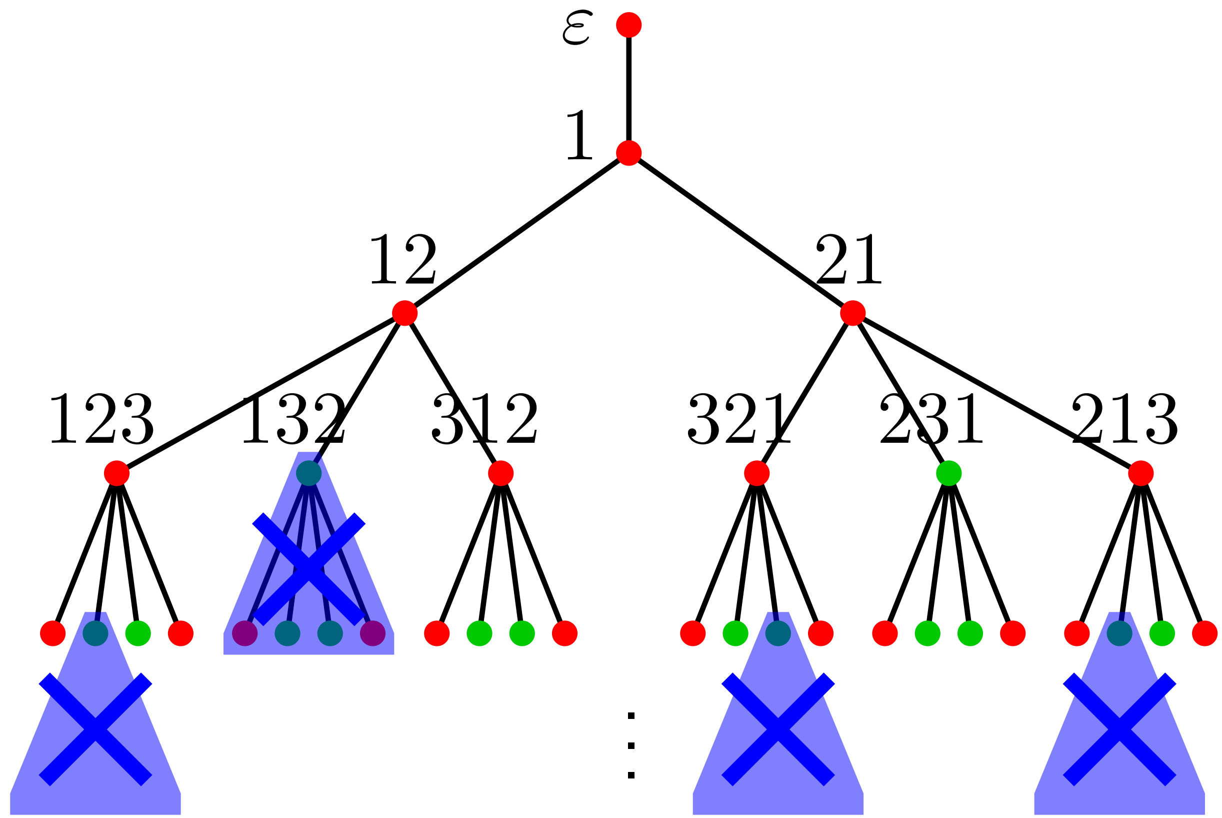
Tree of permutations

- we may prune subtrees iff their root is ●



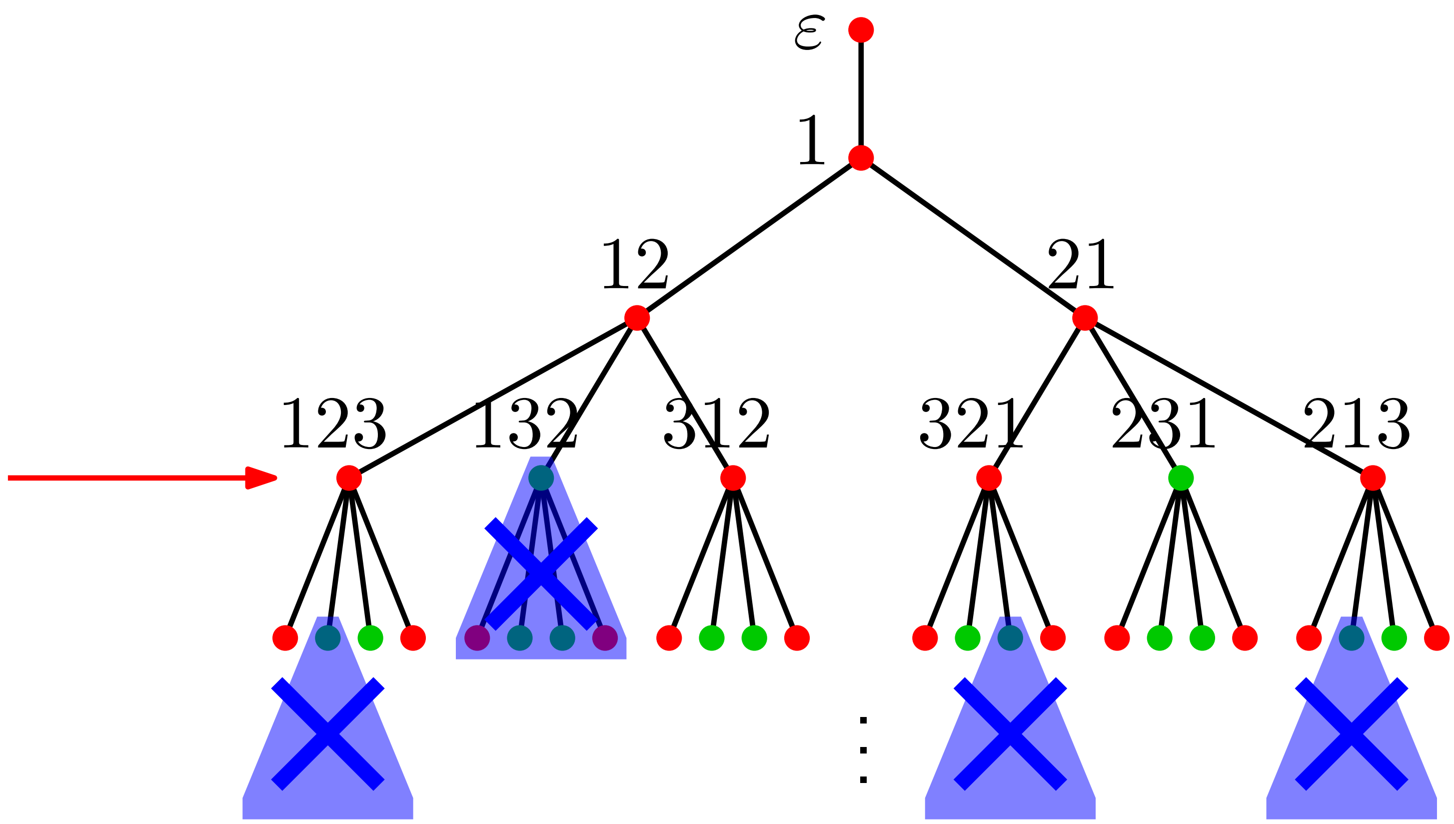
Tree of permutations

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



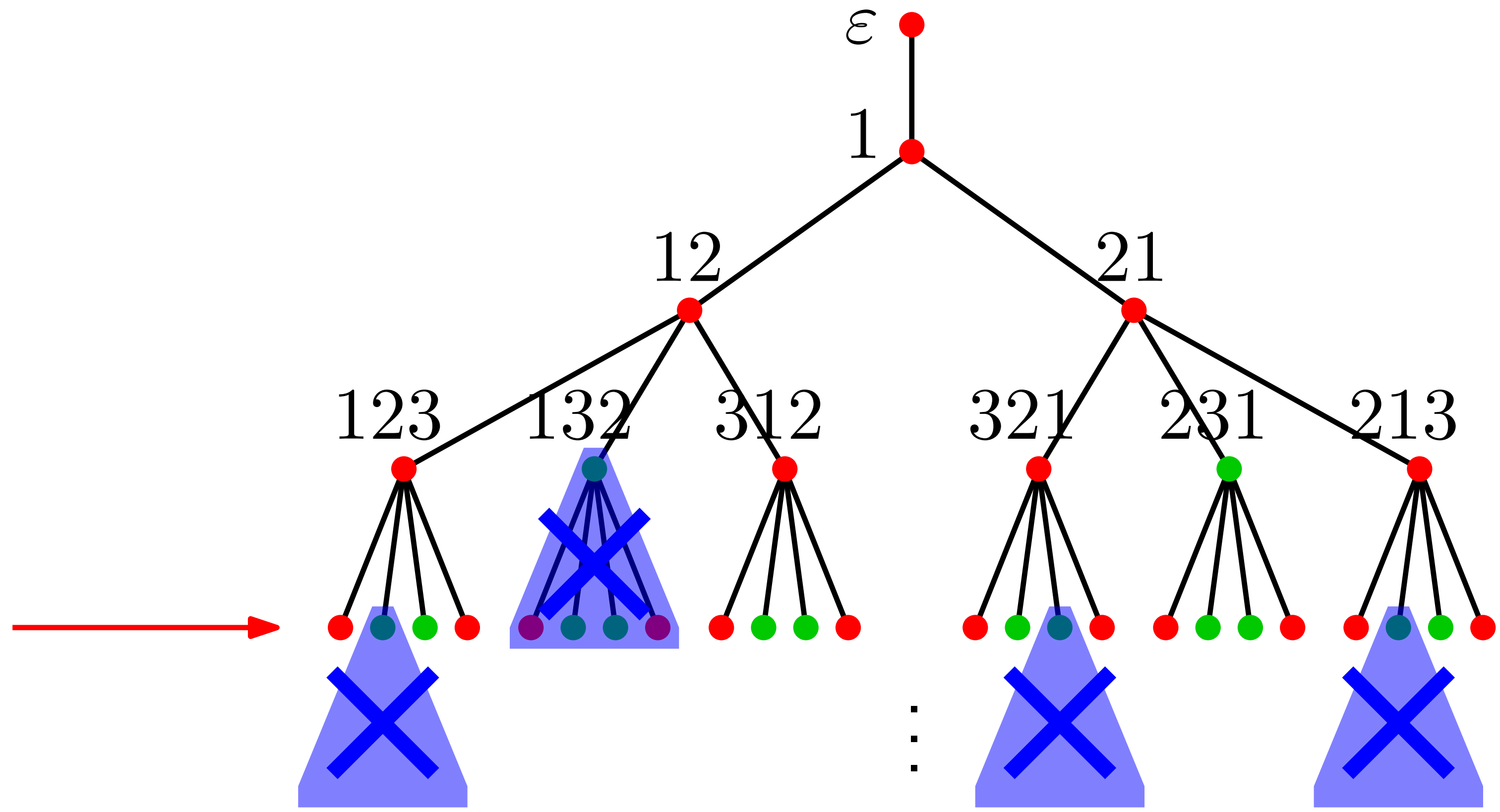
Tree of permutations

- 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**



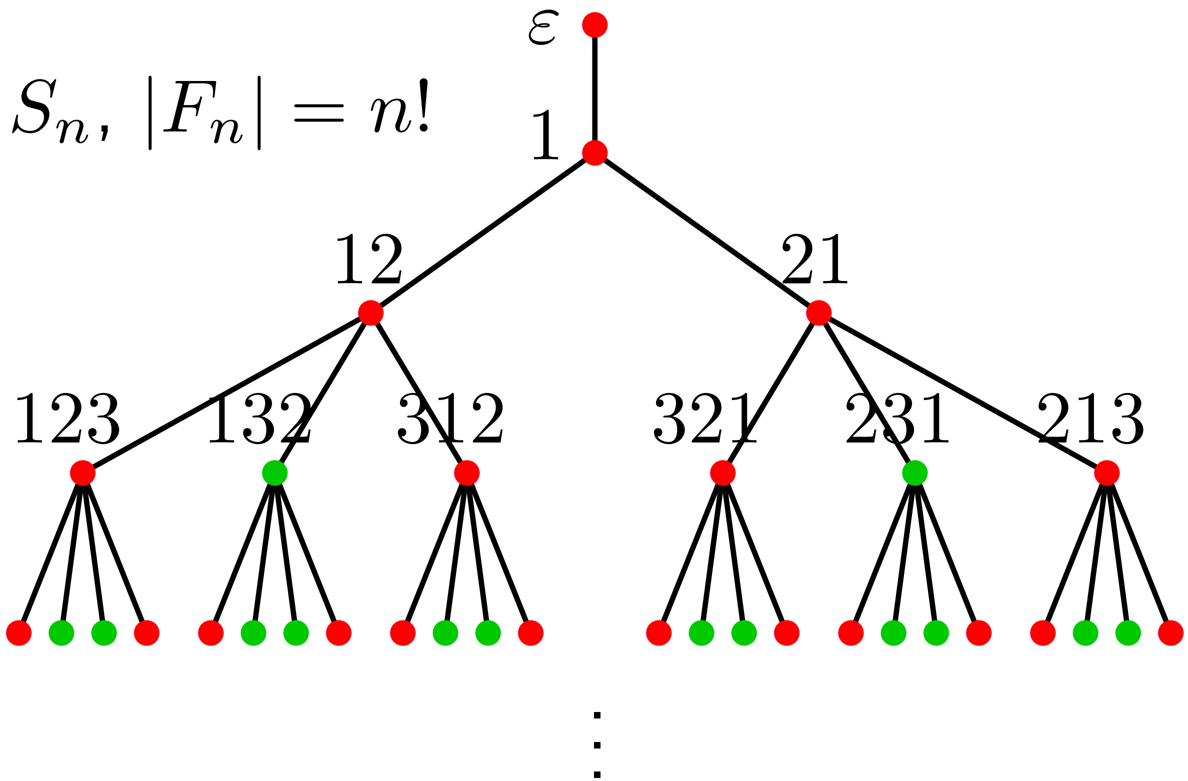
Tree of permutations

- 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**



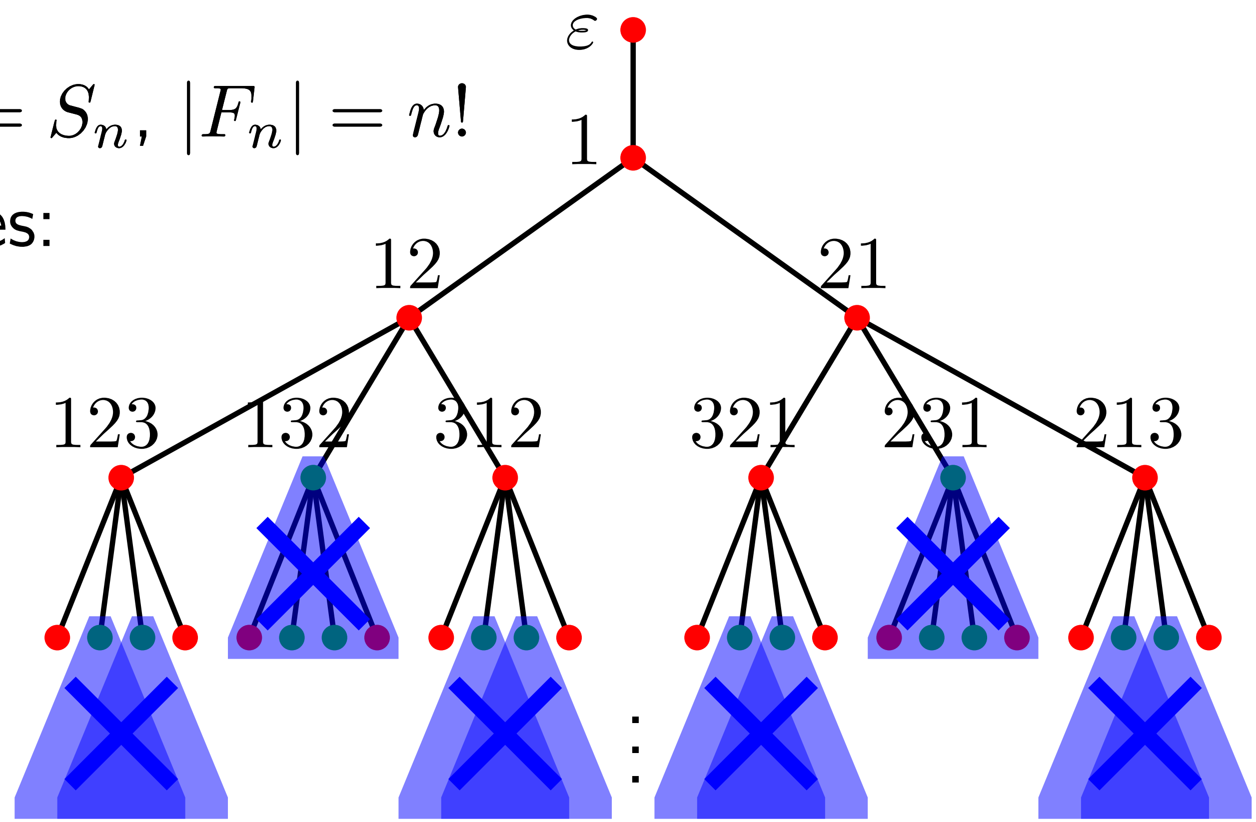
Tree of permutations

- 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 intialization.

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 intialization.

Proof: Induction over the depth of the tree. $\square$

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 intialization.

Proof: Induction over the depth of the tree. $\square$

- the number of zigzag languages is **enormous**:

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

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 intialization.

Proof: Induction over the depth of the tree. $\square$

- the number of zigzag languages is **enormous**:

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

- **many** of them encode interesting combinatorial objects

Examples

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

$F_n =$ permutations without peaks
$|F_n| = 2^{n-1}$

Examples

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

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134

$F_n =$ permutations without peaks
$|F_n| = 2^{n-1}$

Examples

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

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134

Steinhaus-Johnson-Trotter!
minimal jumps
$\leftrightarrow$ adjacent transpositions
$\hookrightarrow$ HC on permutahedron

$F_n =$ permutations without peaks
$|F_n| = 2^{n-1}$

Examples

$F_n = S_n$

$|F_n| = n!$

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134

Steinhaus-Johnson-Trotter!

minimal jumps

$\longleftrightarrow$ adjacent transpositions

$\hookrightarrow$ HC on permutahedron

$F_n =$ permutations without peaks

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

1234
4123
4312
3124
3214
4321
4213
2134

Examples

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

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134

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

$F_n =$ permutations without peaks
$|F_n| = 2^{n-1}$

1234
4123
4312
3124
3214
4321
4213
2134

not a sublist

Examples

$F_n = S_n$

$|F_n| = n!$

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134

Steinhaus-Johnson-Trotter!

minimal jumps

$\longleftrightarrow$ adjacent transpositions

$\hookrightarrow$ HC on permutahedron

$F_n =$ permutations without peaks

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

1234
4123
4312
3124
3214
4321
4213
2134

Examples

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

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134

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

F_n = permutations without peaks
$|F_n| = 2^{n-1}$

<u>234</u>

1234
4123
4312
3124 ↔ f
3214
4321
4213
2134

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

Examples

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

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134

Steinhaus-Johnson-Trotter!
minimal jumps
$\leftrightarrow$ adjacent transpositions
$\hookrightarrow$ HC on permutahedron

F_n = permutations without peaks
$|F_n| = 2^{n-1}$

	234
1234	0
4123	1
4312	1
3124 $\overset{f}{\longleftrightarrow}$	0
3214	0
4321	1
4213	1
2134	0

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

Examples

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

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134

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

F_n = permutations without peaks
$|F_n| = 2^{n-1}$

		234
1234		000
4123	f	001
4312		011
3124	↔	010
3214		110
4321		111
4213		101
2134		100

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

Examples

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

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134

Steinhaus-Johnson-Trotter!
minimal jumps
$\longleftrightarrow$ adjacent transpositions
$\hookrightarrow$ HC on permutahedron

F_n = permutations without peaks
$|F_n| = 2^{n-1}$

		234
1234		000
4123	f	001
4312		011
3124	$\longleftrightarrow$	010
3214		110
4321		111
4213		101
2134		100

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

Binary reflected Gray code!
minimal jumps $\longleftrightarrow$ bitflips
$\hookrightarrow$ HC on hypercube

General approach

Combinatorial objects

General approach

Set of permutations $F_n \subseteq S_n$ $\xleftarrow{\;\;f\;\;}$ Combinatorial objects

General approach

$$\boxed{\text{Set of permutations } F_n \subseteq S_n} \xleftarrow{f} \boxed{\text{Combinatorial objects}}$$

- run Algorithm J

  List $=$ **Algo J**(F_n)

General approach

Set of permutations $F_n \subseteq S_n$ $\xleftarrow{\quad f \quad}$ Combinatorial objects

- run Algorithm J

  $\textbf{List} = \textbf{Algo J}(F_n) \longrightarrow f^{-1}(\textbf{List})$

General approach

Set of permutations $F_n \subseteq S_n$	$\xleftarrow{\quad f \quad}$	Combinatorial objects

- run Algorithm J

  $\textbf{List} = \textbf{Algo J}(F_n) \longrightarrow f^{-1}(\textbf{List})$

- interpret Algorithm J under the bijection

  Algo J

General approach

Set of permutations $F_n \subseteq S_n$ $\xleftarrow{\quad f \quad}$ Combinatorial objects

- run Algorithm J

  $\textbf{List} = \textbf{Algo J}(F_n) \longrightarrow f^{-1}(\textbf{List})$

- interpret Algorithm J under the bijection

  $\textbf{Algo J} \longrightarrow f^{-1}(\textbf{Algo J})$

General approach

$\boxed{\text{Set of permutations } F_n \subseteq S_n} \xleftarrow{\quad f \quad} \boxed{\text{Combinatorial objects}}$

- run Algorithm J

  List $=$ **Algo J**(F_n) $\longrightarrow$ $f^{-1}($**List**$)$

- interpret Algorithm J under the bijection

  Algo J $\longrightarrow$ $f^{-1}($**Algo J**$)$

- minimal jumps $\longrightarrow$ ‘small changes’

General approach

$$\boxed{\text{Set of permutations } F_n \subseteq S_n} \xleftarrow{\;f\;} \boxed{\text{Combinatorial objects}}$$

- run Algorithm J

  $\textbf{List} = \textbf{Algo J}(F_n) \longrightarrow f^{-1}(\textbf{List})$

- interpret Algorithm J under the bijection

  $\textbf{Algo J} \longrightarrow f^{-1}(\textbf{Algo J})$

- minimal jumps $\longrightarrow$ 'small changes'

  $\hookrightarrow$ walks on lattices / polytopes

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

- **I. 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) **[SODA'20]**
- **II. lattice congruences** of the weak order on S_n
- **III. pattern-avoiding rectangulations** **[SoCG'21]**
- **IV. elimination trees** **[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 τ is **tame**, if

Pattern-avoiding permutations

- A pattern τ is **tame**, if

  classical: largest entry not at the boundary

Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

$2\textcolor{red}{4}13$ ✓

Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

$2\mathbf{4}13$ ✓ $\qquad$ $\mathbf{4}213$ ✗

Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

2413 ✓ 4213 ✗

vincular: $+$ one vincular pair involving the largest entry

Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

$$2\mathbf{4}13 \;✅ \qquad\qquad \mathbf{4}213 \;❌$$

vincular: $+$ one vincular pair involving the largest entry

$$2\underline{41}3 \;✅$$

Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

2413 ✓ $\qquad\qquad$ 4213 ✗

vincular: $+$ one vincular pair involving the largest entry

$2\underline{41}3$ ✓

$\underline{24}13$ ✓

Pattern-avoiding permutations

- A pattern τ is **tame**, if

  classical: largest entry not at the boundary

  2413 ✓ $\qquad$ 4213 ✗

  vincular: $+$ one vincular pair involving the largest entry

  $2\underline{41}3$ ✓ $\qquad$ $24\underline{13}$ ✗

  $\underline{24}13$ ✓

Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

$2\mathbf{4}13$ ✓ $\mathbf{4}213$ ✗

vincular: $+$ one vincular pair involving the largest entry

$2\underline{41}3$ ✓ $2\mathbf{4}\underline{13}$ ✗

$\underline{24}13$ ✓ $\underline{24}\,\underline{13}$ ✗

Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

2413 ✓ $\qquad$ 4213 ✗

vincular: $+$ one vincular pair involving the largest entry

$2\underline{41}3$ ✓ $\qquad$ $24\underline{13}$ ✗

$\underline{24}13$ ✓ $\qquad$ $\underline{24}\,\underline{13}$ ✗

mesh: $+$ no shaded cell in the top row

Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

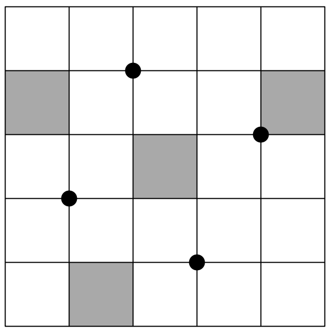
2413 ✓ $\qquad$ 4213 ✗

vincular: $+$ one vincular pair involving the largest entry

$2\underline{41}3$ ✓ $\qquad$ $24\underline{13}$ ✗

$\underline{24}13$ ✓ $\qquad$ $\underline{24}\,\underline{13}$ ✗

mesh: $+$ no shaded cell in the top row

 ✓

Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

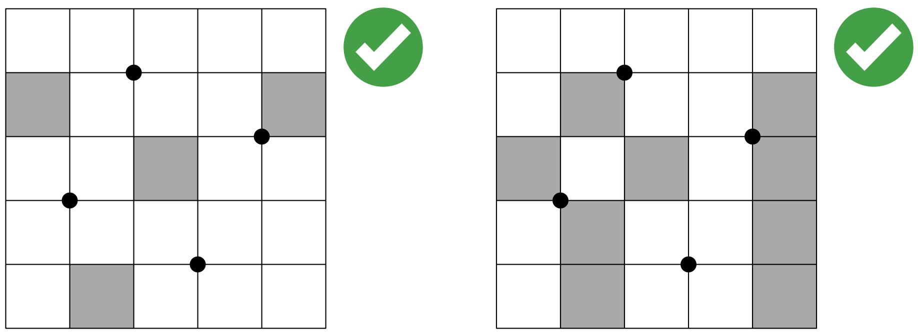
$2\mathbf{4}13$ ✓ $\mathbf{4}213$ ✗

vincular: $+$ one vincular pair involving the largest entry

$2\underline{41}3$ ✓ $24\underline{13}$ ✗

$\underline{24}13$ ✓ $\underline{24}\,\underline{13}$ ✗

mesh: $+$ no shaded cell in the top row



Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

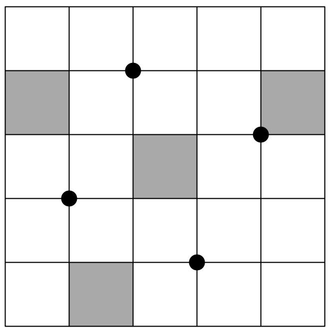
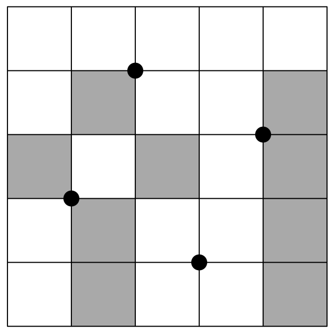
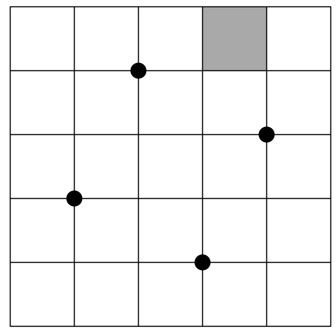
$2\textcolor{red}{4}13$ ✓ $\qquad$ $\textcolor{red}{4}213$ ✗

vincular: $+$ one vincular pair involving the largest entry

$2\underline{\textcolor{red}{4}1}3$ ✓ $\qquad$ $2\textcolor{red}{4}\underline{13}$ ✗

$\underline{2\textcolor{red}{4}}13$ ✓ $\qquad$ $\underline{2\textcolor{red}{4}}\,\underline{13}$ ✗

mesh: $+$ no shaded cell in the top row

 ✓  ✓  ✗

Pattern-avoiding permutations

- A pattern τ is **tame**, if

classical: largest entry not at the boundary

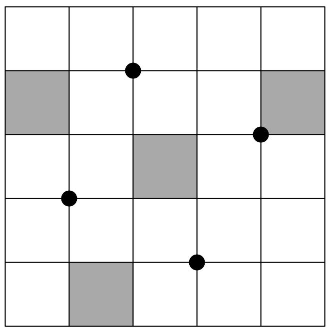
2413 ✓ 4213 ✗

vincular: $+$ one vincular pair involving the largest entry

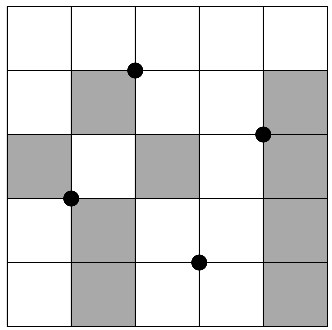
$2\underline{41}3$ ✓ $24\underline{13}$ ✗

$\underline{24}13$ ✓ $\underline{24}\,\underline{13}$ ✗

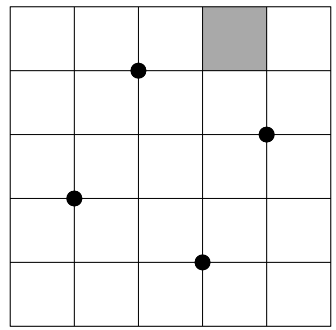
mesh: $+$ no shaded cell in the top row



✓



✓



✗

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 $\xleftrightarrow{f}$ Combinatorial objects

Pattern-avoiding permutations

Tame patterns $\xleftrightarrow{f}$	Combinatorial objects
231	Catalan families

Pattern-avoiding permutations

Tame patterns $\overset{f}{\longleftrightarrow}$ Combinatorial objects		
231	Catalan families	• binary trees by **rotations** • triangulations by **flips** • Dyck paths by **hill flips**

Pattern-avoiding permutations

Tame patterns $\xleftrightarrow{f}$	Combinatorial objects	
231	Catalan families	• binary trees by **rotations** • triangulations by **flips** • Dyck paths by **hill flips**
$\underline{23}1$	Bell families	

Pattern-avoiding permutations

Tame patterns $\xleftrightarrow{f}$		Combinatorial objects
231	Catalan families	• binary trees by **rotations** • triangulations by **flips** • Dyck paths by **hill flips**
$\underline{23}1$	Bell families	• set partitions by **element exchanges**

Pattern-avoiding permutations

Tame patterns $\xleftrightarrow{f}$	Combinatorial objects	
231	Catalan families	• binary trees by **rotations** • triangulations by **flips** • Dyck paths by **hill flips**
$\underline{23}1$	Bell families	• set partitions by **element exchanges**
231,132		• bitstrings by **flips** (BRGC)

Pattern-avoiding permutations

Tame patterns $\xleftrightarrow{f}$ Combinatorial objects

---

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

$\underline{23}1$ — Bell families
- set partitions by **element exchanges**

$231, 132$
- bitstrings by **flips** (BRGC)

$2\underline{41}3, 3\underline{14}2$ — Baxter families

Pattern-avoiding permutations

Tame patterns $\xleftrightarrow{f}$	Combinatorial objects	
231	Catalan families	• binary trees by **rotations** • triangulations by **flips** • Dyck paths by **hill flips**
$\underline{23}1$	Bell families	• set partitions by **element exchanges**
$231, 132$		• bitstrings by **flips** (BRGC)
$2\underline{41}3, 3\underline{14}2$	Baxter families	• diagonal rectangulations

Pattern-avoiding permutations

Tame patterns $\xleftrightarrow{f}$ Combinatorial objects		
231	Catalan families	• binary trees by **rotations** • triangulations by **flips** • Dyck paths by **hill flips**
$\underline{23}1$	Bell families	• set partitions by **element exchanges**
231, 132		• bitstrings by **flips** (BRGC)
$2\underline{41}3, 3\underline{14}2$	Baxter families	• diagonal rectangulations
$3\underline{51}24, 3\underline{51}42,$ $24\underline{51}3, 42\underline{51}3$	2-clumped pms.	

Pattern-avoiding permutations

Tame patterns $\xleftrightarrow{f}$ Combinatorial objects		
231	Catalan families	• binary trees by **rotations** • triangulations by **flips** • Dyck paths by **hill flips**
$\underline{23}1$	Bell families	• set partitions by **element exchanges**
231, 132		• bitstrings by **flips** (BRGC)
$2\underline{41}3, 3\underline{14}2$	Baxter families	• diagonal rectangulations
$3\underline{51}24, 3\underline{51}42,$ $24\underline{51}3, 42\underline{51}3$	2-clumped pms.	• generic rectangulations

Pattern-avoiding permutations

Tame patterns $\xleftrightarrow{f}$		Combinatorial objects
231	Catalan families	• binary trees by **rotations** • triangulations by **flips** • Dyck paths by **hill flips**
$\underline{23}1$	Bell families	• set partitions by **element exchanges**
231,132		• bitstrings by **flips** (BRGC)
$2\underline{41}3,3\underline{14}2$	Baxter families	• diagonal rectangulations
$3\underline{51}24,3\underline{51}42,$ $24\underline{51}3,42\underline{51}3$	2-clumped pms.	• generic rectangulations

$\rightarrow$ see the Combinatorial Object Server: `www.combos.org/jump`

Grid classes

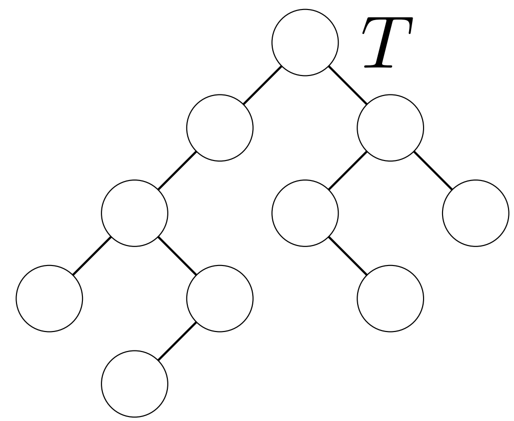
- monotone grid class $\mathrm{Grid}_n(M)$ **[Huczynska, Vatter 06]**
- geometric grid class $\mathrm{Geo}_n(M)$ **[Albert et al. 13]**

Grid classes

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

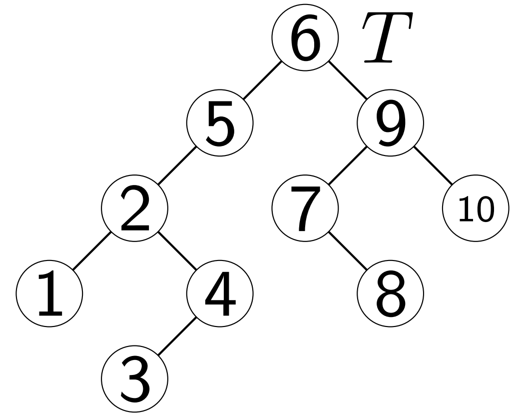
Theorem: If $M = \begin{array}{|c|c|c|c|}\hline -1 & & & +1 \\ \hline & & & \\ \hline & & & \\ \hline \end{array}$, then both $\mathrm{Grid}_n(M)$ and $\mathrm{Geo}_n(M)$ are zigzag languages.

Binary trees



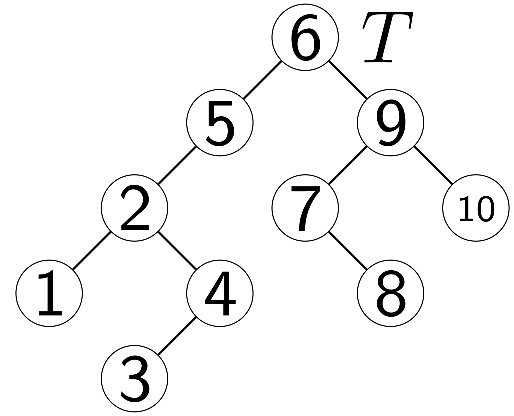
Binary trees

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



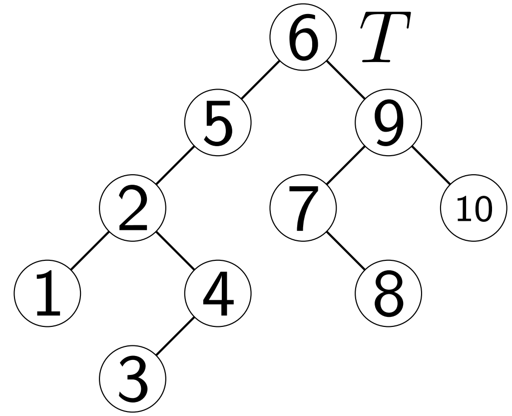
Binary trees

- 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

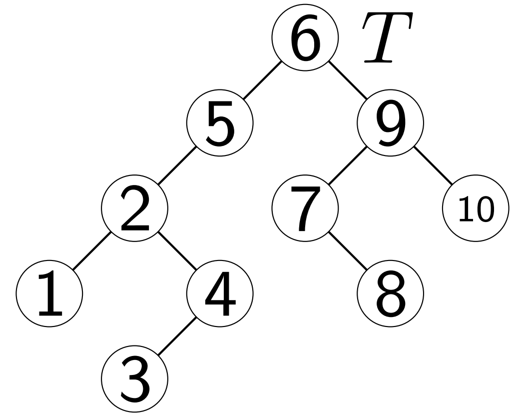
- Label vertices with $1, \ldots, n$ according to search tree property: for any vertex i, we have $L(i) < i < R(i)$
- $T_n :=$ binary (search) trees with n vertices



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

Binary 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



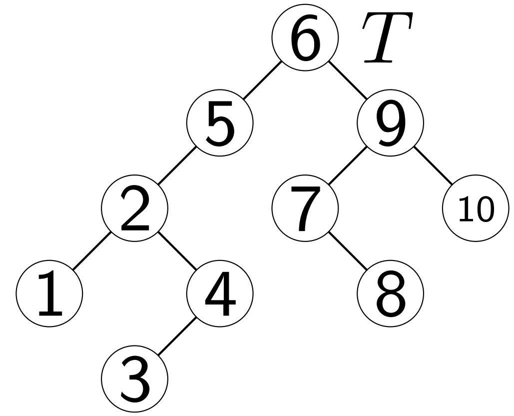
Theorem **[Folklore]**: There is a bijection f between T_n and $S_n(231)$.

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

'preorder traversal'

Binary trees

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



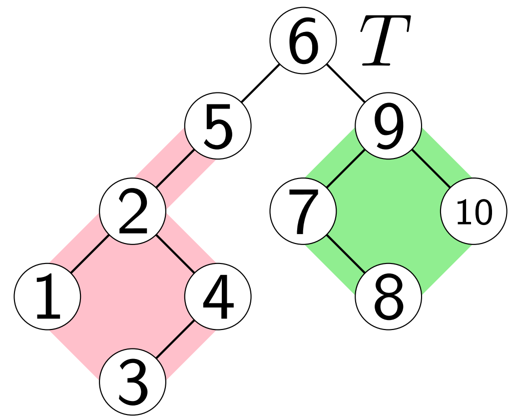
Theorem [Folklore]**:** There is a bijection f between T_n and $S_n(231)$.

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

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

Binary trees

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



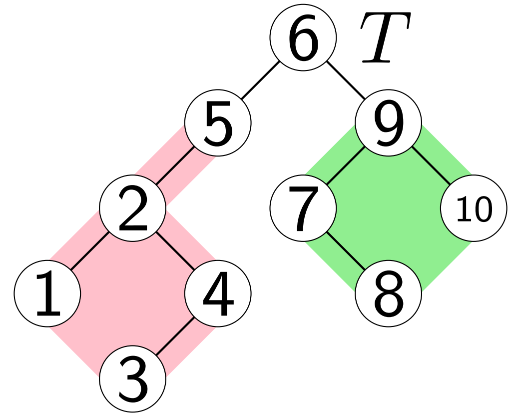
Theorem **[Folklore]**: There is a bijection f between T_n and $S_n(231)$.

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

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

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 $S_n(231)$.

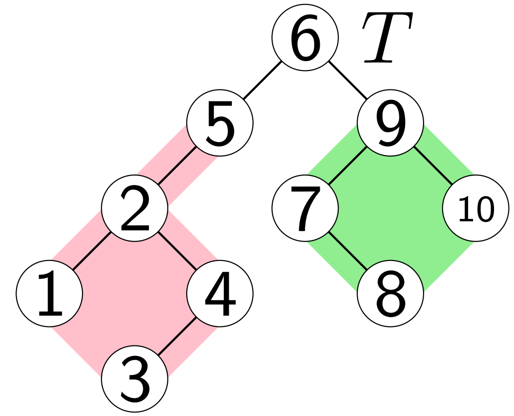
$$f(T) := (r(T), L(T), R(T))$$

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

- $S_n(231)$ 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' $\quad f(T) = (6, 5, 2, 1, 4, 3, 9, 7, 8, 10)$

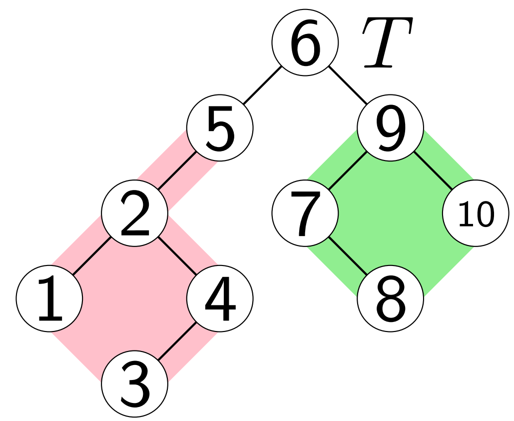
- $S_n(231)$ is a zigzag language, so Algorithm J applies

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

= **[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 $S_n(231)$.

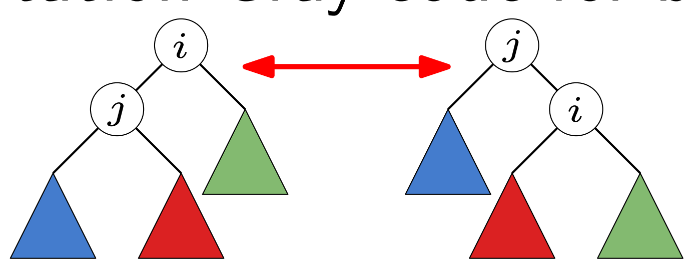
$$f(T) := (r(T), L(T), R(T))$$

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

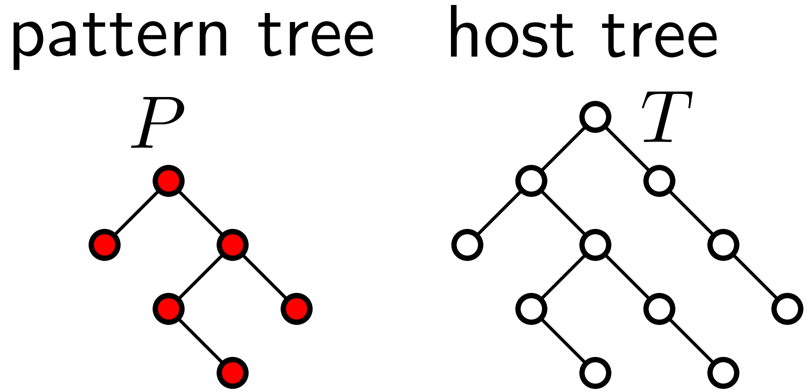
- $S_n(231)$ is a zigzag language, so Algorithm J applies

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

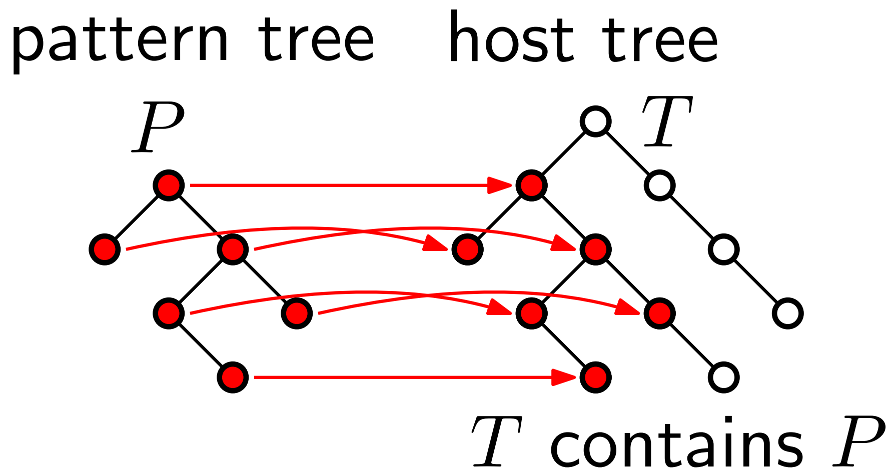
= [Lucas, Roelants van Baronaigien, Ruskey 93]



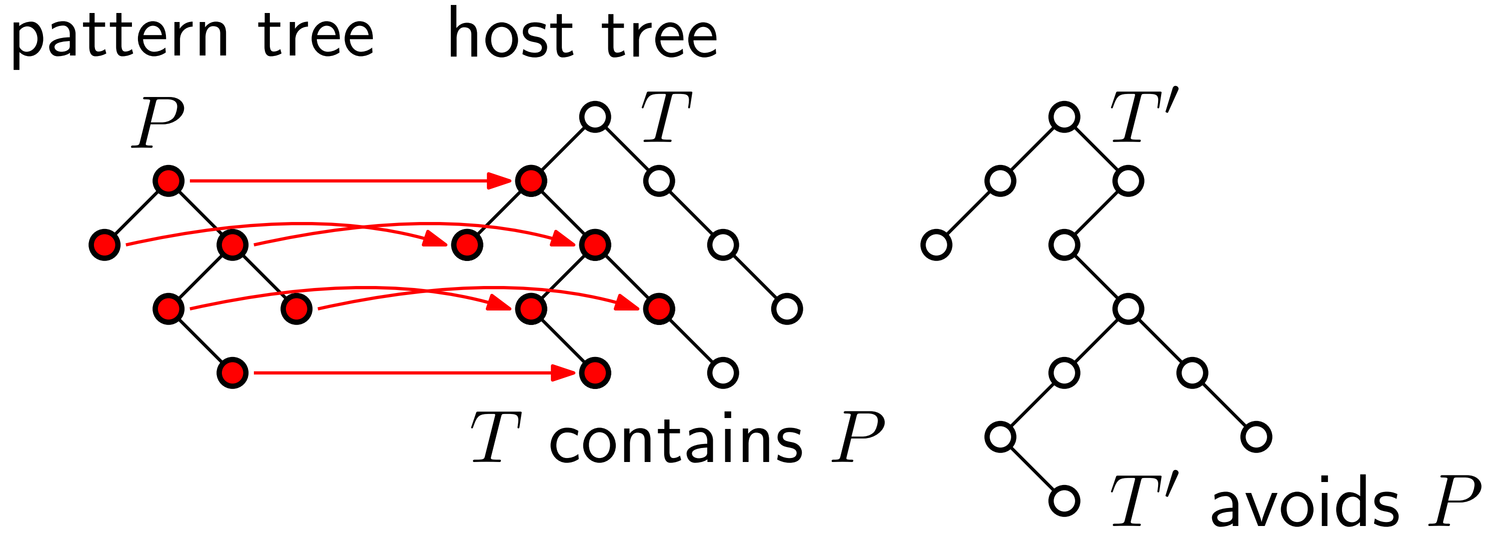
Patterns in binary trees



Patterns in binary trees

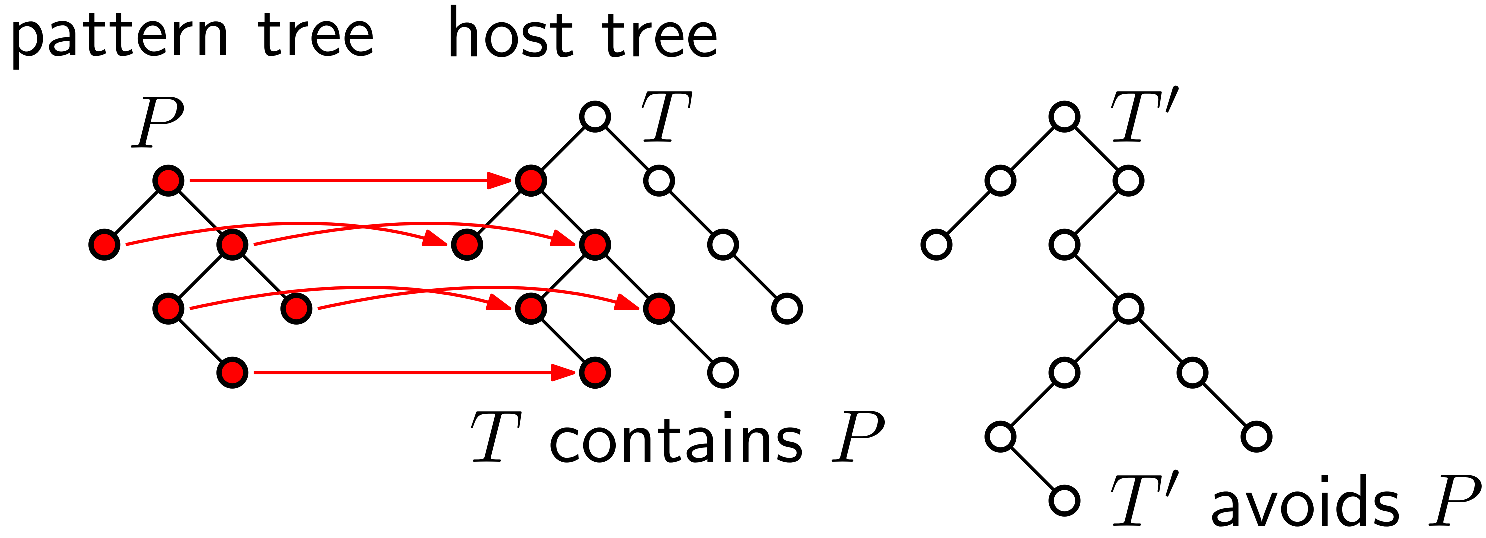


Patterns in binary trees



Patterns in binary trees

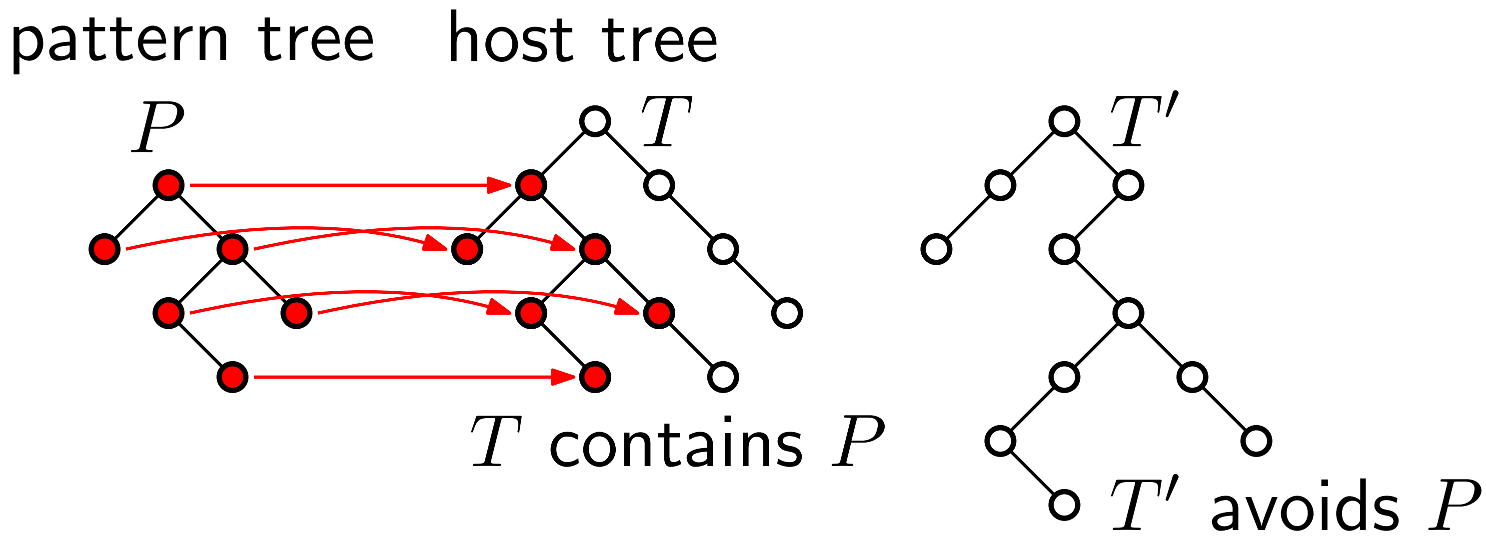
contiguous
[Rowland 10]



Patterns in binary trees

contiguous

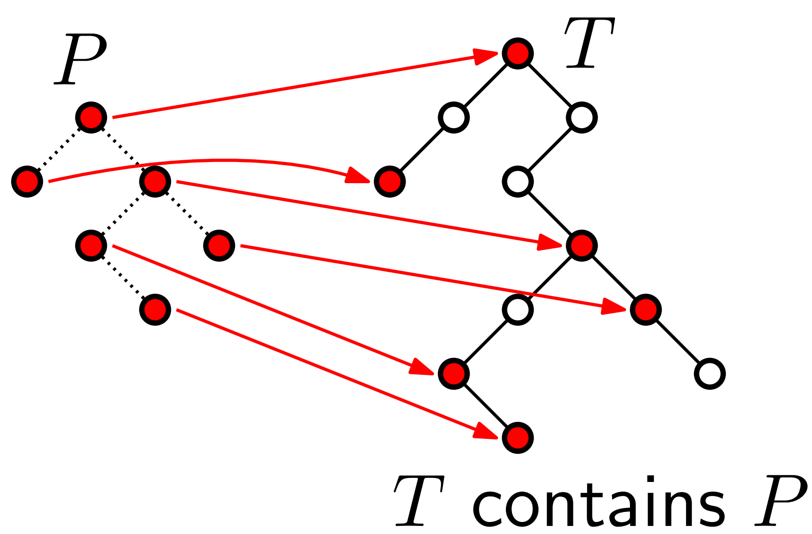
[Rowland 10]



---

non-contiguous

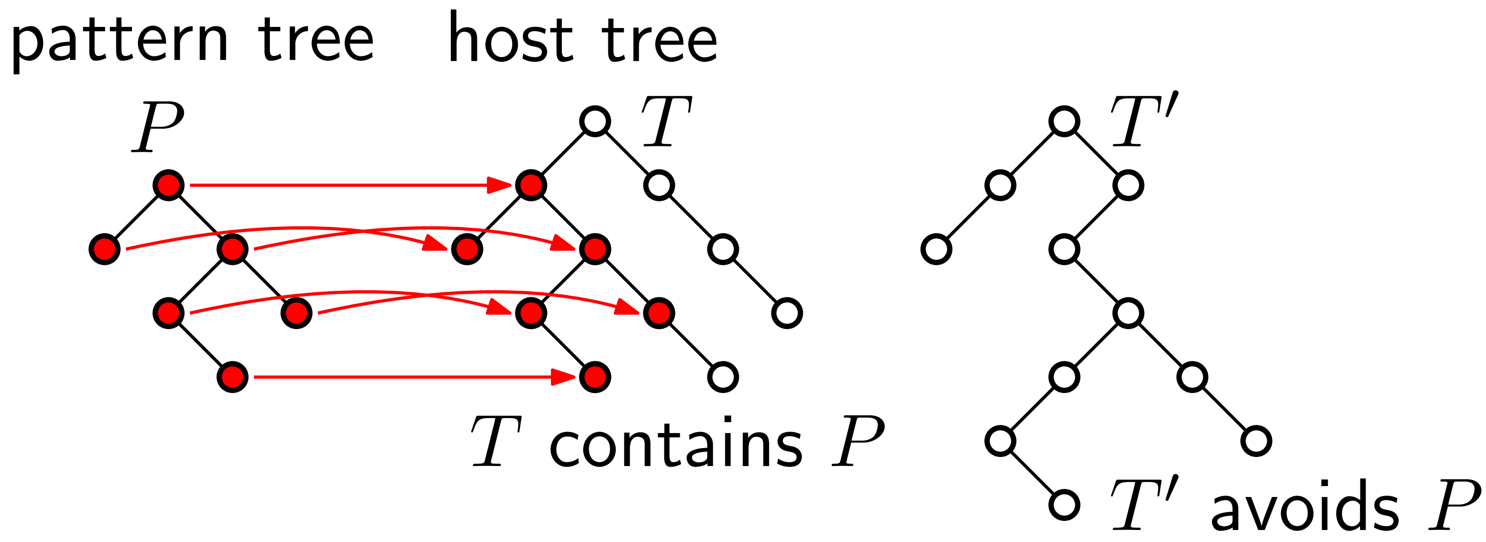
[Dairyko, Tyner, Pudwell, Wynn 12]



Patterns in binary trees

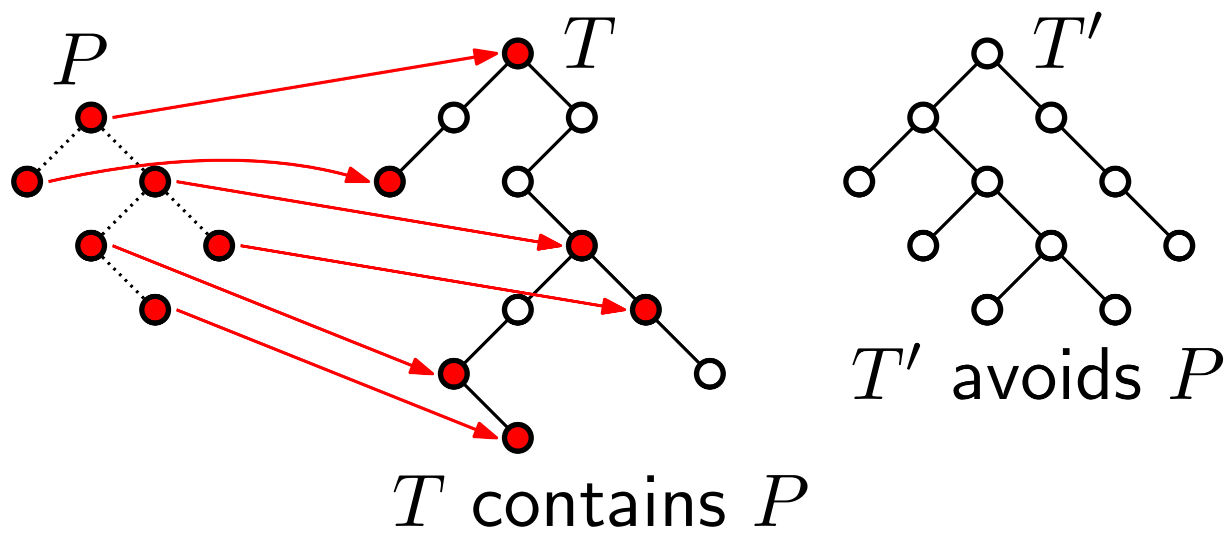
contiguous

[Rowland 10]



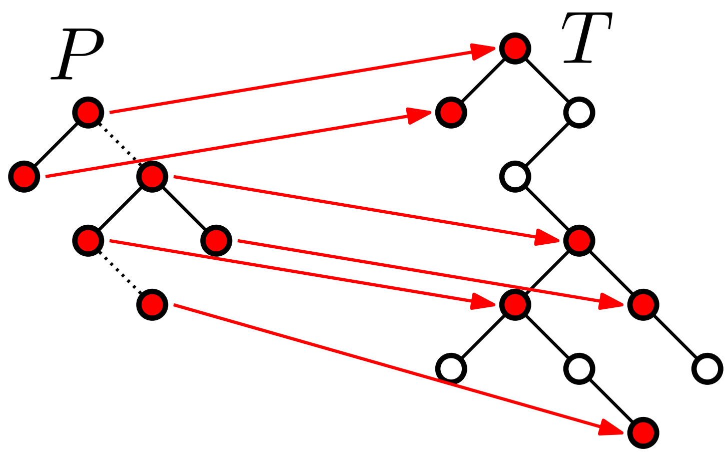
non-contiguous

[Dairyko, Tyner, Pudwell, Wynn 12]



Mixed tree patterns

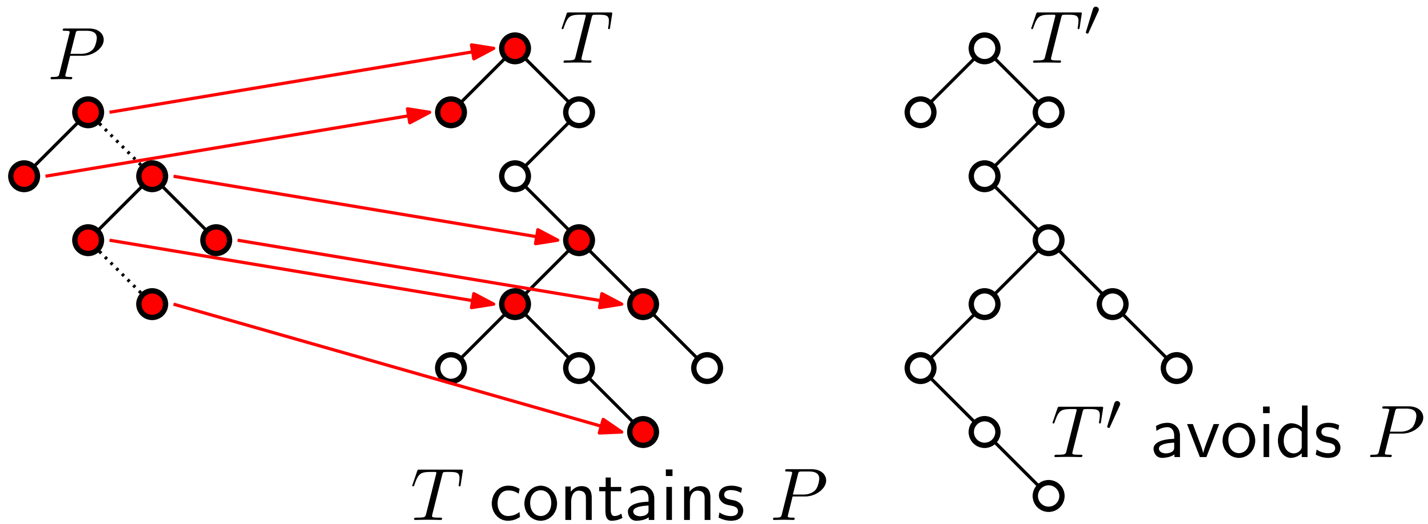
mixed (new)



T contains P

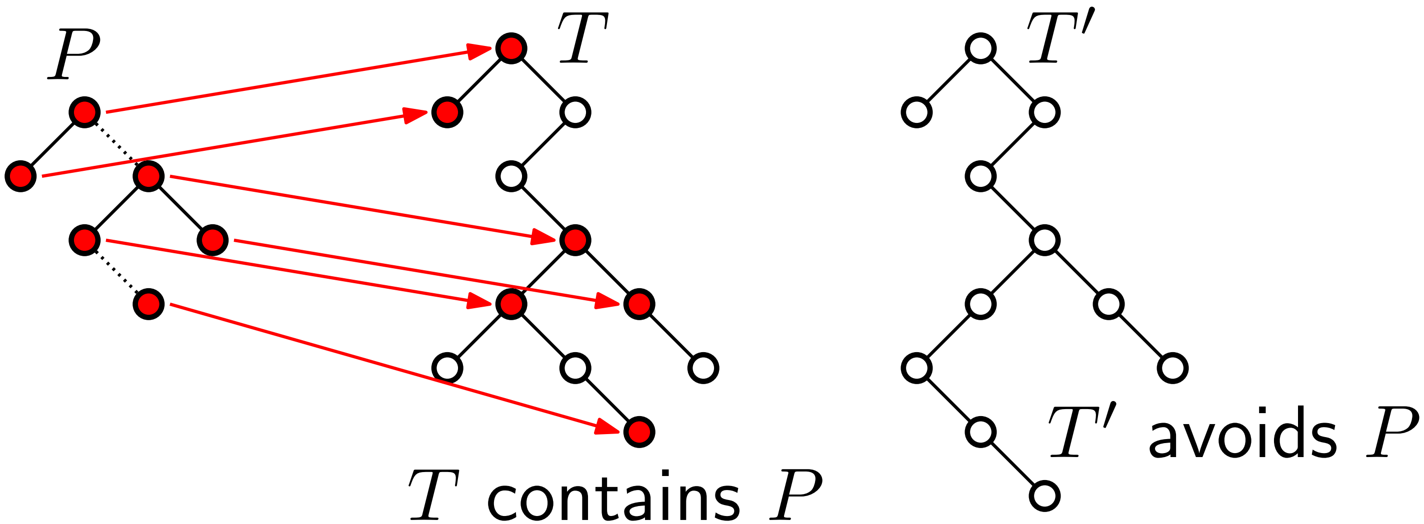
Mixed tree patterns

mixed (new)



Mixed tree patterns

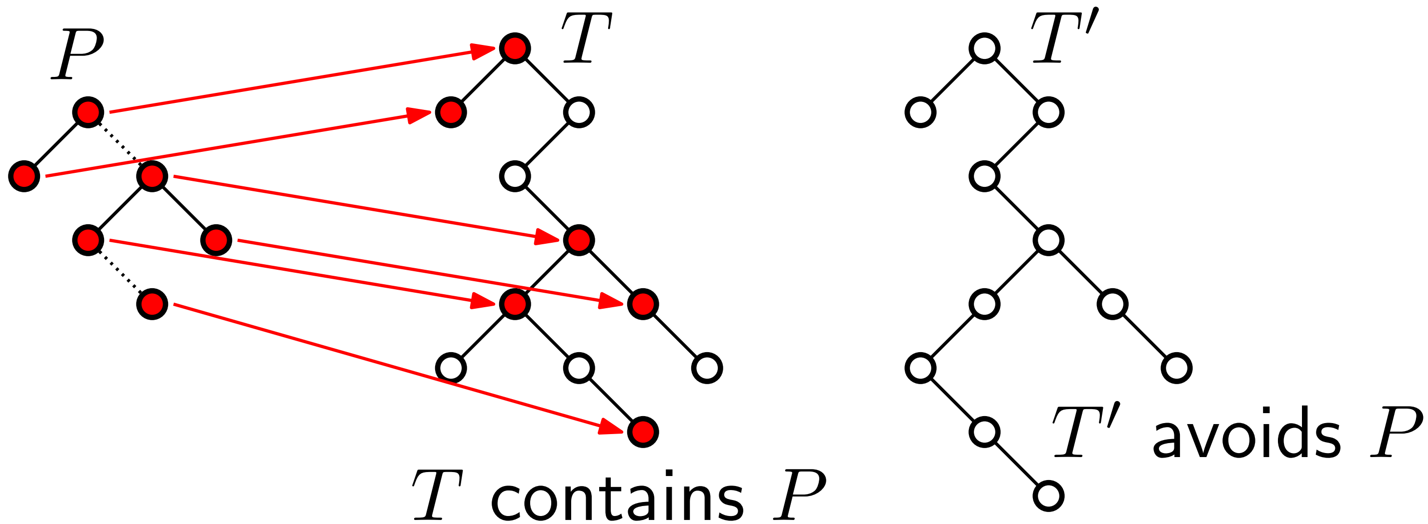
mixed (new)



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.

Mixed tree patterns

mixed (new)

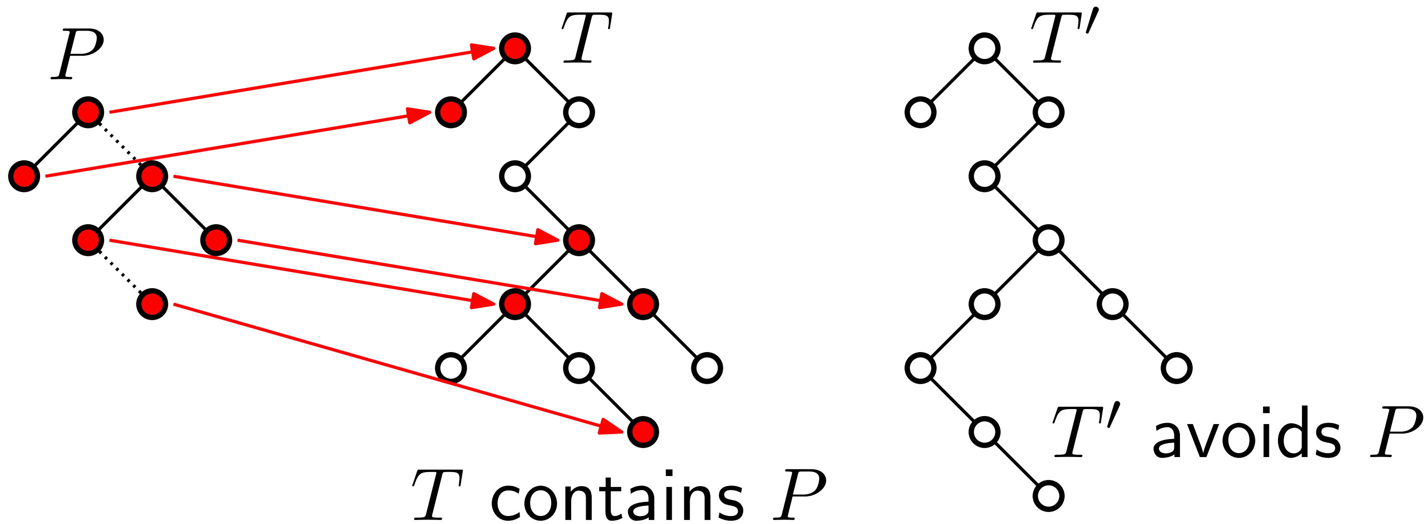


Theorem: For every (mixed) tree pattern, there is a permutation mesh pattern $\tau(P) = (f(P), C)$ such that $f : T_n(P) \to S_n(231, \tau(P))$ is a bijection.

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

Mixed tree patterns

mixed (new)

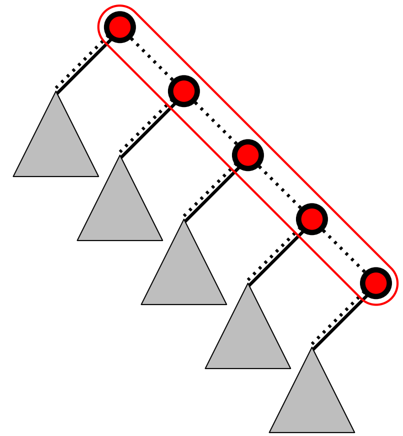


Theorem: For every (mixed) tree pattern, there is a permutation mesh pattern $\tau(P) = (f(P), C)$ such that $f : T_n(P) \to S_n(231, \tau(P))$ is a bijection.

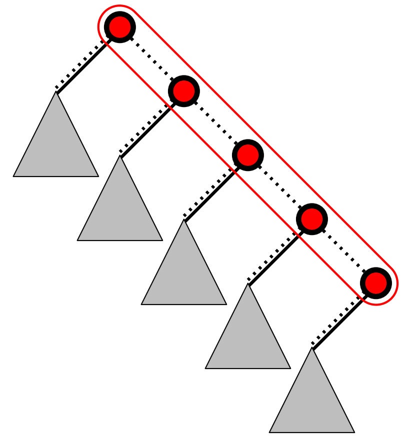
- generalizes result of [Pudwell, Scholten, Schrock, Serrato 14]
- classified all tree patterns on ≤ 5 vertices; interesting bijections to pattern-avoiding lattice paths and set partitions

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



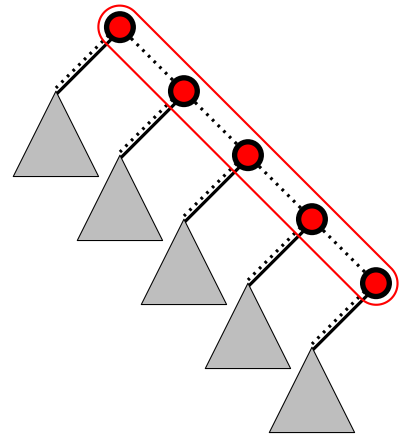
Tame patterns



- 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



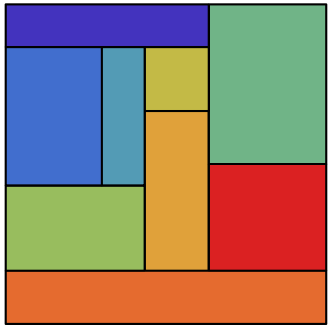
- A pattern P is **tame**, if the largest node is neither root nor leaf, and the right branch from the root is non-contiguous

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.

$\rightarrow$ see `www.combos.org/btree`

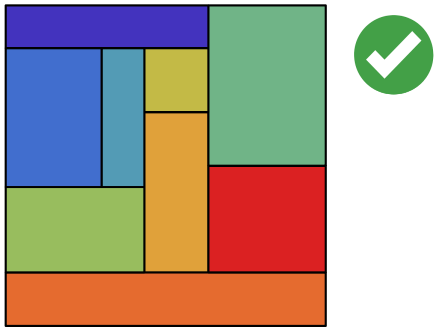
Generic rectangulations

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



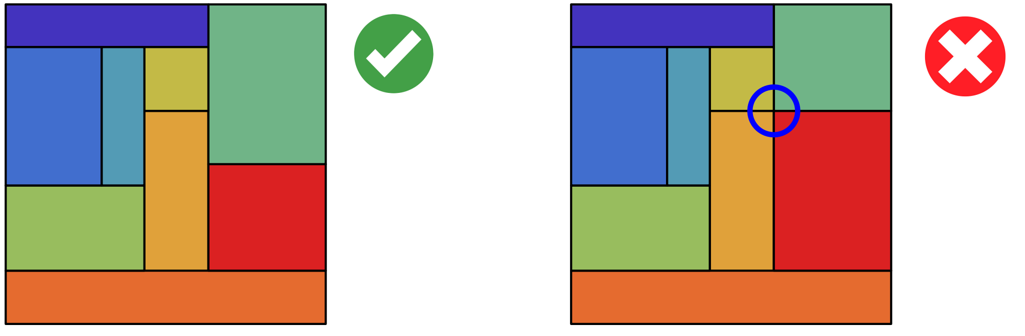
Generic rectangulations

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



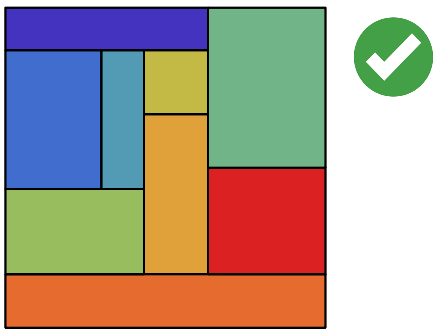
Generic rectangulations

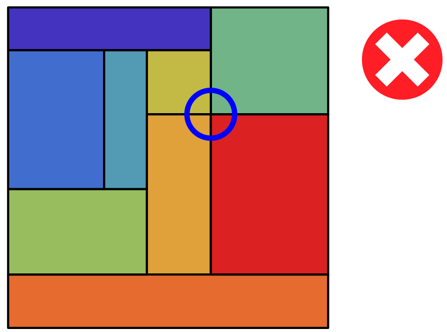
- **Generic rectangulation:** subdivision of a square into n rectangles s.t. no four rectangles meet



Generic rectangulations

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

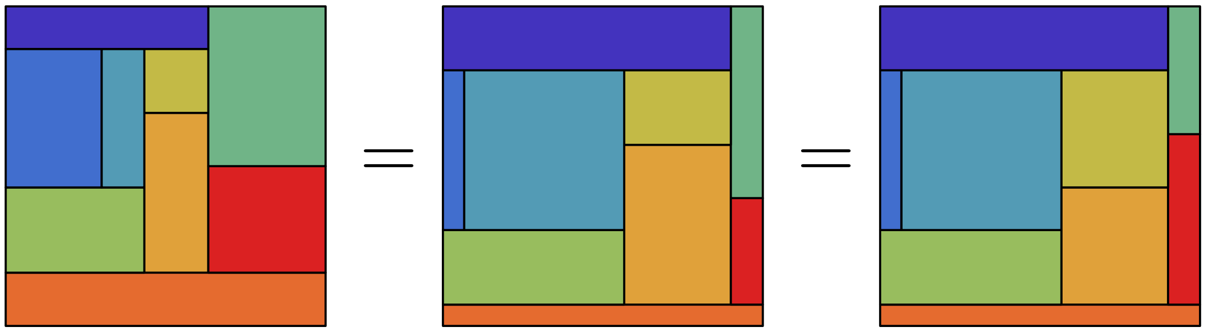




- ‘combinatorial’ equivalence: only incidences between rectangles matter

Generic rectangulations

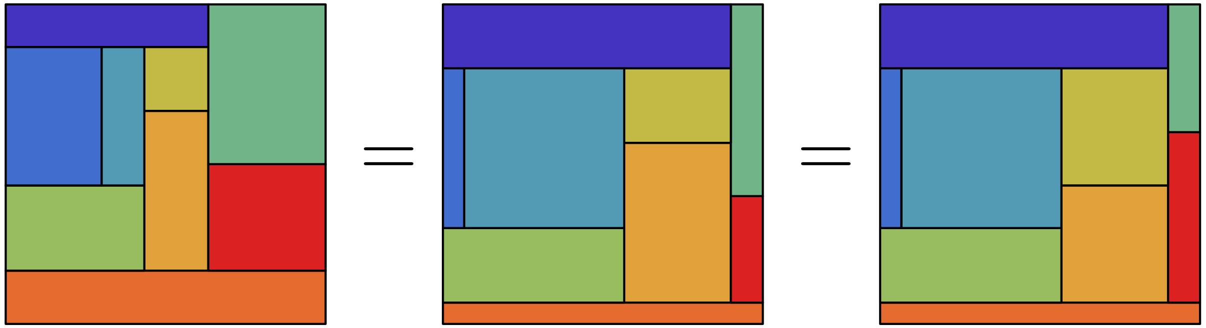
- **Generic rectangulation:** subdivision of a square into n rectangles s.t. no four rectangles meet



- 'combinatorial' equivalence: only incidences between rectangles matter

Generic rectangulations

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

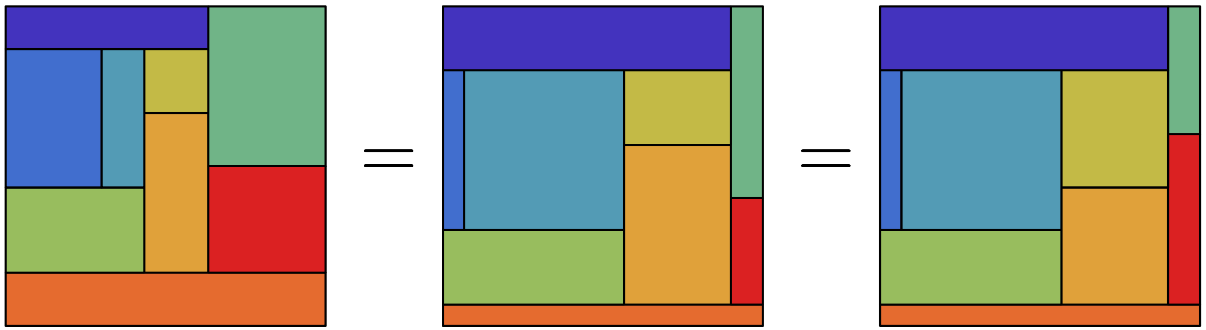


- 'combinatorial' equivalence: only incidences between rectangles matter

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

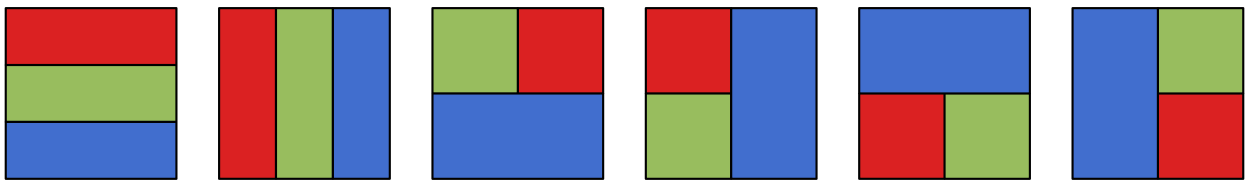
Generic rectangulations

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



- '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 $S_n(3\underline{51}24, 3\underline{51}42, 24\underline{51}3, 42\underline{51}3)$ (2-clumped permutations).

Generic rectangulations

Theorem **[Reading 12]:** There is a bijection f between R_n and $\underbrace{S_n(3\underline{51}24, 3\underline{51}42, 24\underline{51}3, 42\underline{51}3)}_{\text{is a zigzag language, so Algorithm J applies}}$ (2-clumped permutations).

Generic rectangulations

Theorem [Reading 12]**:** There is a bijection f between R_n and $\underbrace{S_n(3\underline{51}24, 3\underline{51}42, 24\underline{51}3, 42\underline{51}3)}_{\text{is a zigzag language, so Algorithm J applies}}$ (2-clumped permutations).

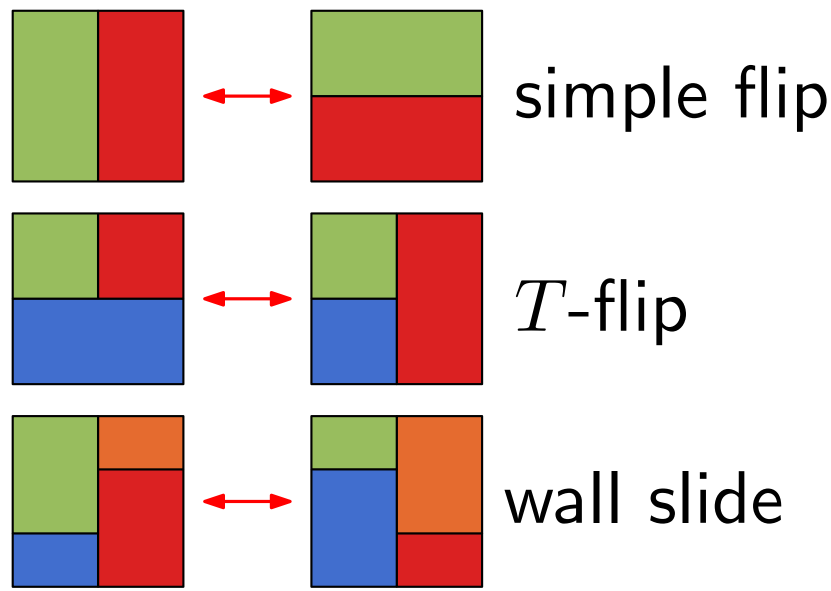
Theorem: Under f^{-1}, minimal jumps of Algorithm J translate to rectangle flips, i.e., we obtain a flip Gray code for generic rectangulations ($\hookrightarrow$ HC on quotientope).

Generic rectangulations

Theorem **[Reading 12]:** There is a bijection f between R_n and $\underbrace{S_n(3\underline{51}24, 3\underline{51}42, 24\underline{51}3, 42\underline{51}3)}_{\text{is a zigzag language, so Algorithm J applies}}$ (2-clumped permutations).

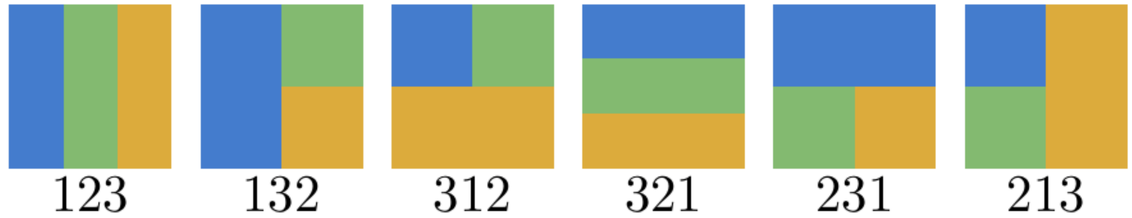
Theorem: Under f^{-1}, minimal jumps of Algorithm J translate to rectangle flips, i.e., we obtain a flip Gray code for generic rectangulations ($\hookrightarrow$ HC on quotientope).

rectangle flips

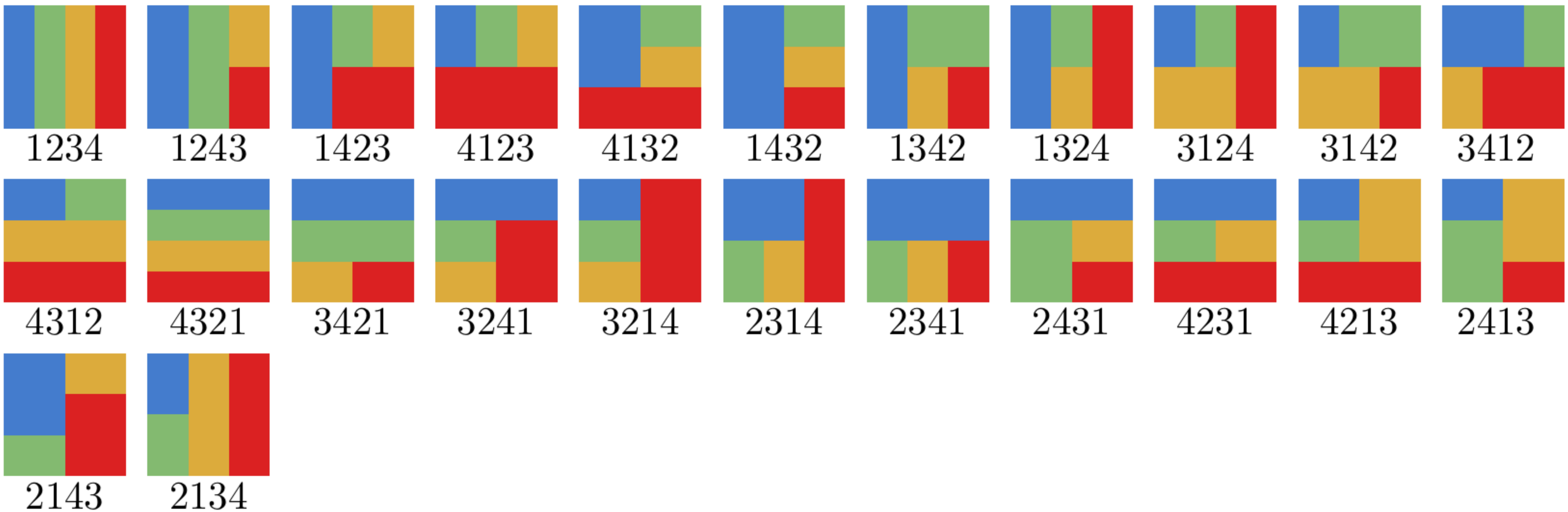


Flip Gray code

$n = 3$

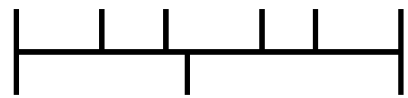


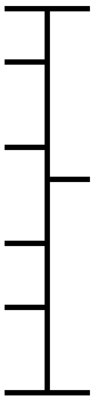
$n = 4$



Patterns in rectangulations

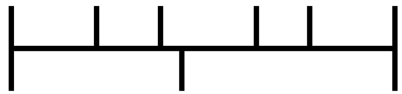
- **Segment:** maximal sequence of inner edges

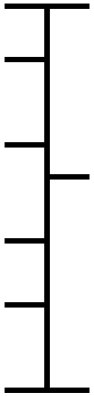




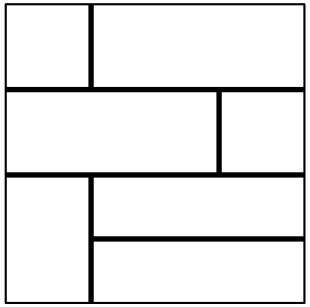
Patterns in rectangulations

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



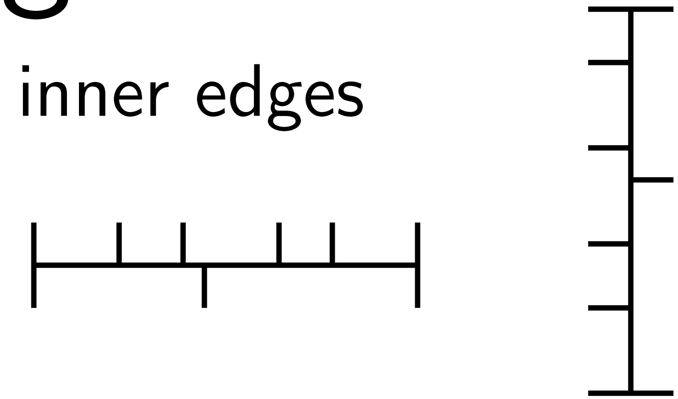


- **Pattern:** connected configuration of segments

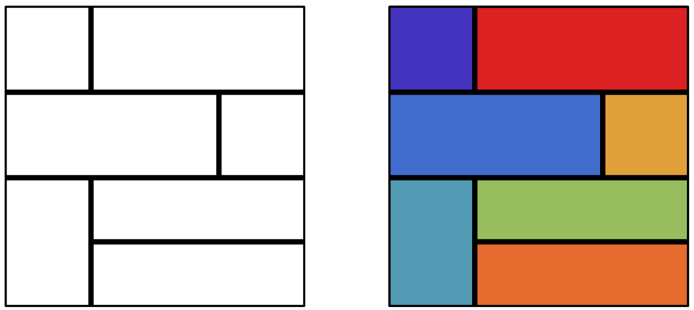
$P =$ 

Patterns in rectangulations

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



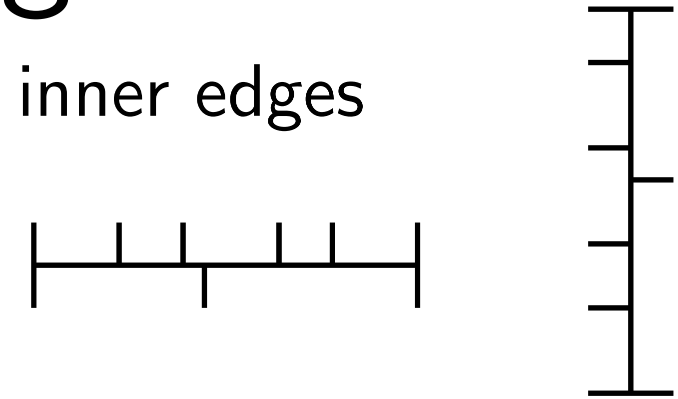
- **Pattern:** connected configuration of segments

$P =$ 

can be seen as a rectangulation itself

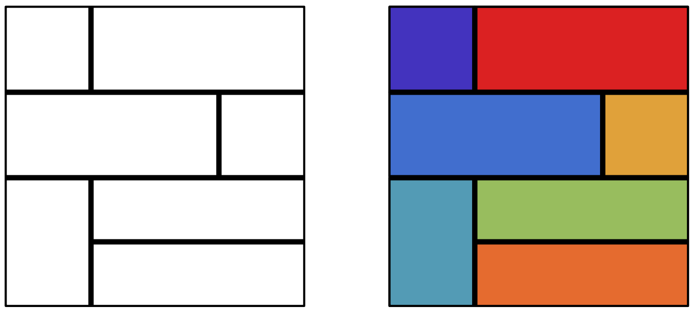
Patterns in rectangulations

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

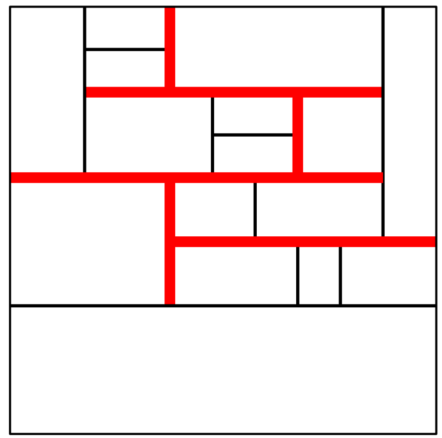


- **Pattern:** connected configuration of segments

$P =$



can be seen as a rectangulation itself



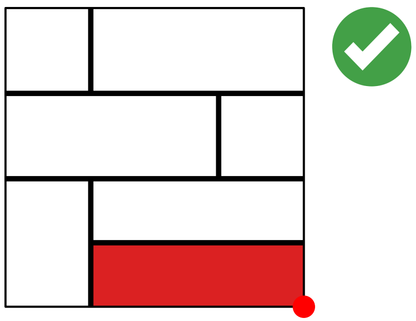
contains P

Tame patterns

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

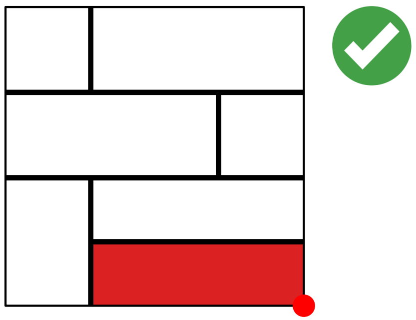
Tame patterns

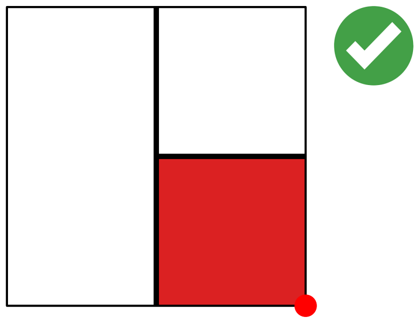
- A pattern P is **tame**, if the bottom right corner rectangle does not stretch across the whole bottom or right side



Tame patterns

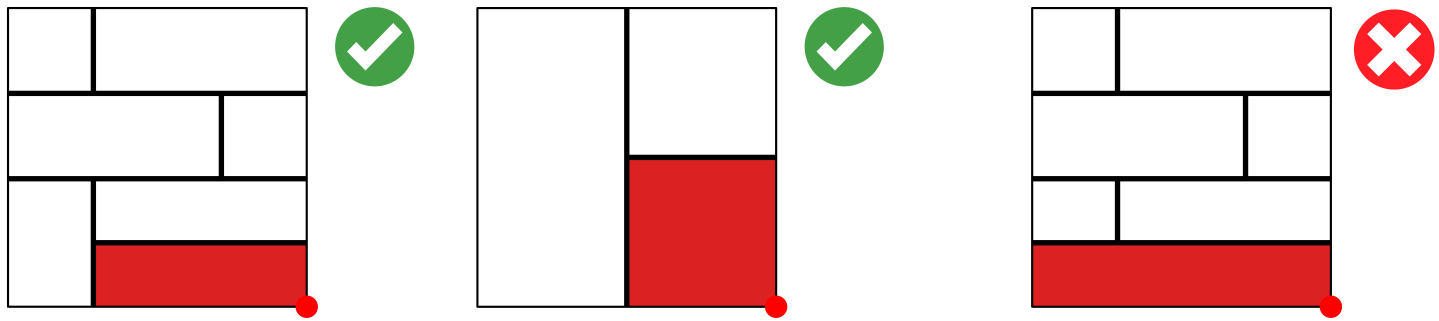
- A pattern P is **tame**, if the bottom right corner rectangle does not stretch across the whole bottom or right side





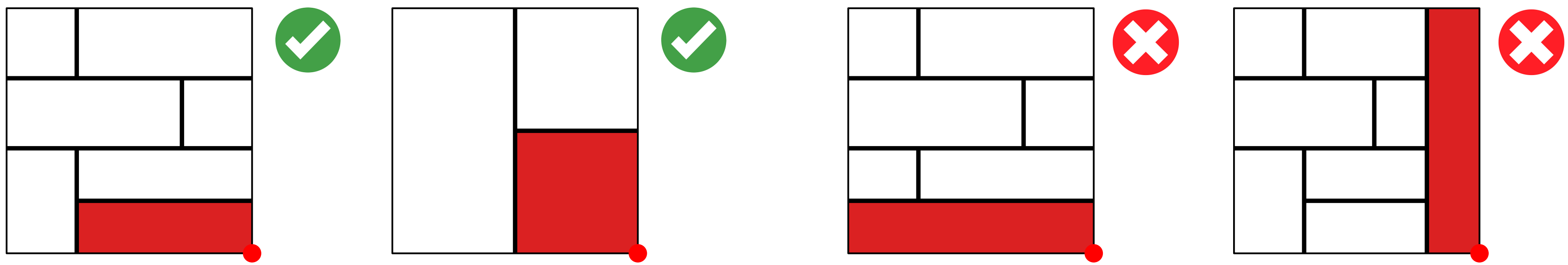
Tame patterns

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



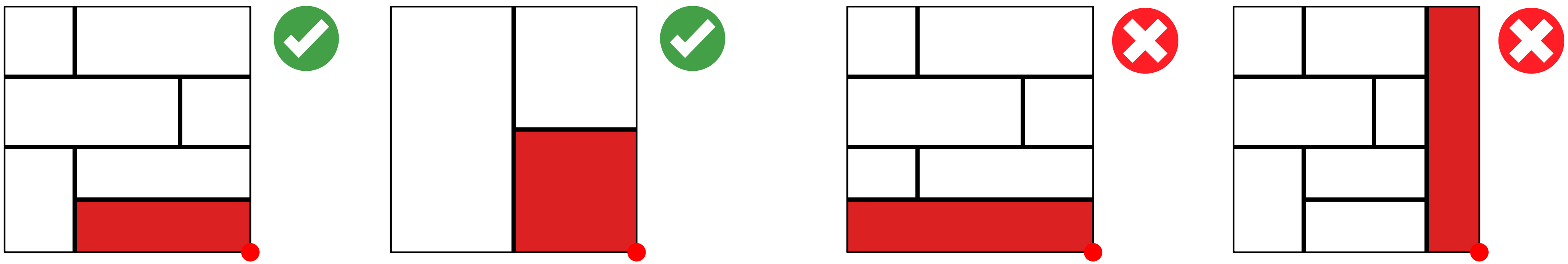
Tame patterns

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



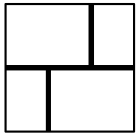
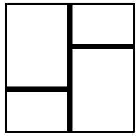
Tame patterns

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

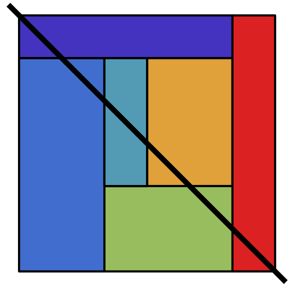


Theorem: If $P_1, \ldots, P_k$ are tame patterns, then $f(R_n(P_1, \ldots, P_k))$ is a zigzag language. Under f^{-1}, minimal jumps of Algorithm J translate to sequences of rectangle flips.

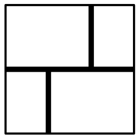
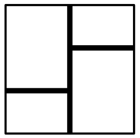
Examples

$R_n($  $,$  $)$

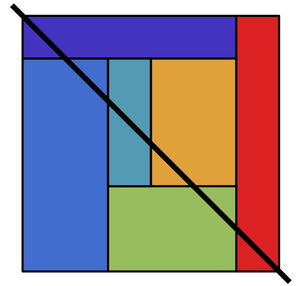
diagonal rectangulations



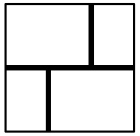
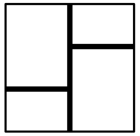
Examples

$R_n($  $,$  $)$

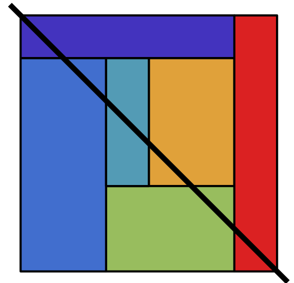
diagonal rectangulations
$\hookrightarrow$ HC on quotientope

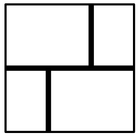
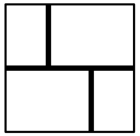
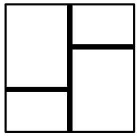
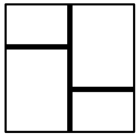


Examples

$R_n($  ,  $)$

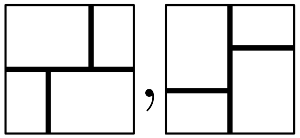
diagonal rectangulations
$\hookrightarrow$ HC on quotientope



$R_n($  ,  ,  ,  $)$

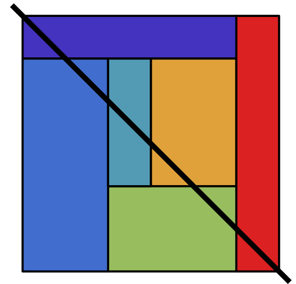
area-universal rectangulations
[Eppstein, Mumford, Speckmann, Verbeek 2012]

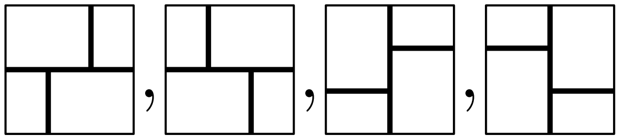
Examples

$R_n($  $)$

diagonal rectangulations

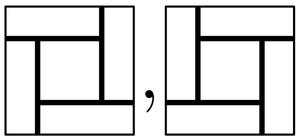
$\hookrightarrow$ HC on quotientope



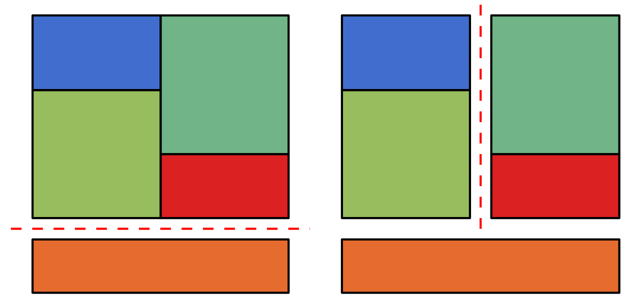
$R_n($  $)$

area-universal rectangulations

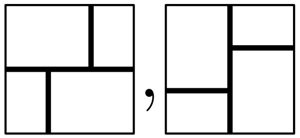
[Eppstein, Mumford, Speckmann, Verbeek 2012]

$R_n($  $)$

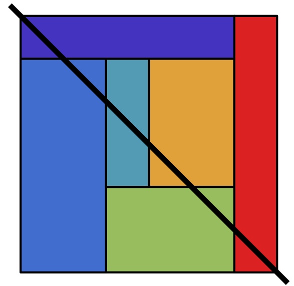
guillotine rectangulations

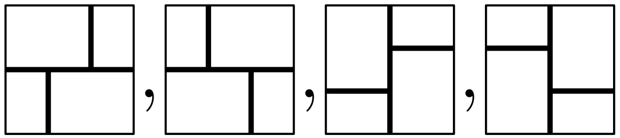


Examples

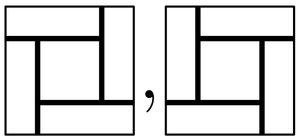
$R_n($  $)$

diagonal rectangulations
$\hookrightarrow$ HC on quotientope

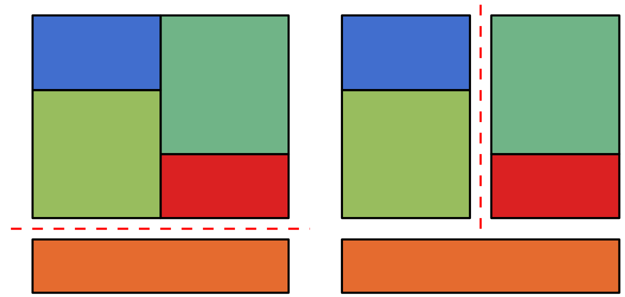


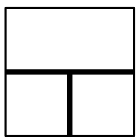
$R_n($  $)$

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

$R_n($  $)$

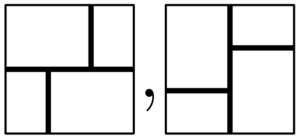
guillotine rectangulations



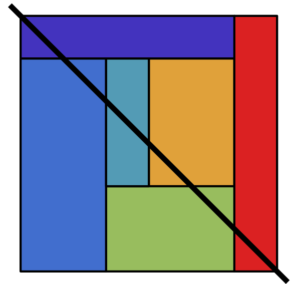
$R_n($  $)$

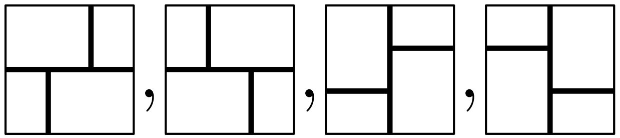
Catalan staircases C_n
[Downing, Einstein, Hartung, Williams 2023]

Examples

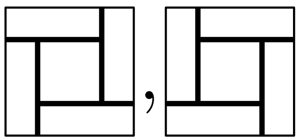
$R_n($  $)$

diagonal rectangulations
$\hookrightarrow$ HC on quotientope

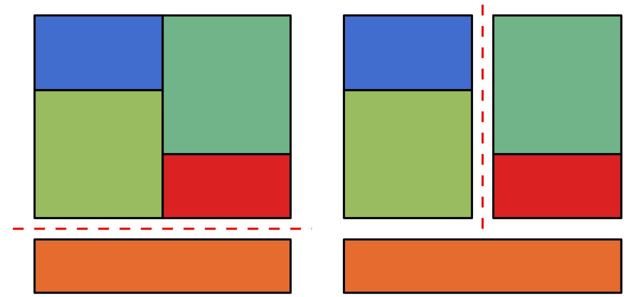


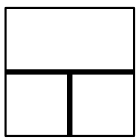
$R_n($  $)$

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

$R_n($  $)$

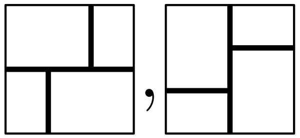
guillotine rectangulations



$R_n($  $)$

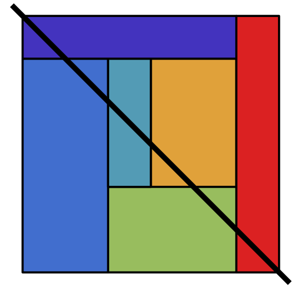
Catalan staircases $\quad C_n$
[Downing, Einstein, Hartung, Williams 2023]
$\hookrightarrow$ HP on associahedron

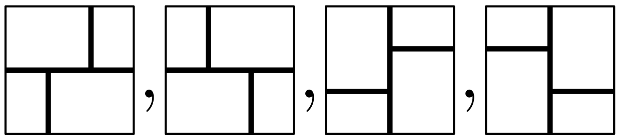
Examples

$R_n($  $)$

diagonal rectangulations

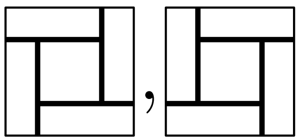
$\hookrightarrow$ HC on quotientope



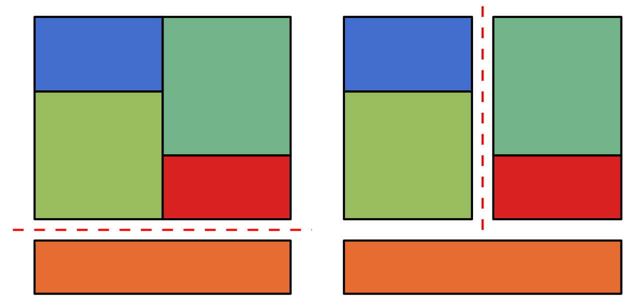
$R_n($  $)$

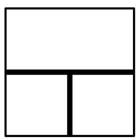
area-universal rectangulations

[Eppstein, Mumford, Speckmann, Verbeek 2012]

$R_n($  $)$

guillotine rectangulations

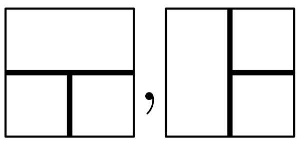


$R_n($  $)$

Catalan staircases $\quad C_n$

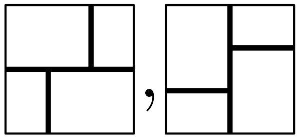
[Downing, Einstein, Hartung, Williams 2023]

$\hookrightarrow$ HP on associahedron

$R_n($  $)$

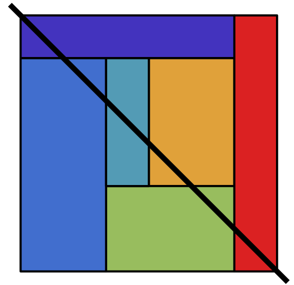
stacked rectangulations $\quad 2^n$

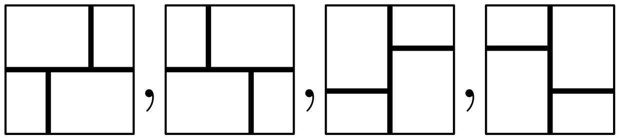
Examples

$R_n($  $)$

diagonal rectangulations

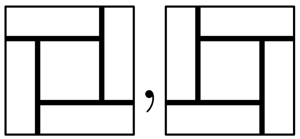
$\hookrightarrow$ HC on quotientope



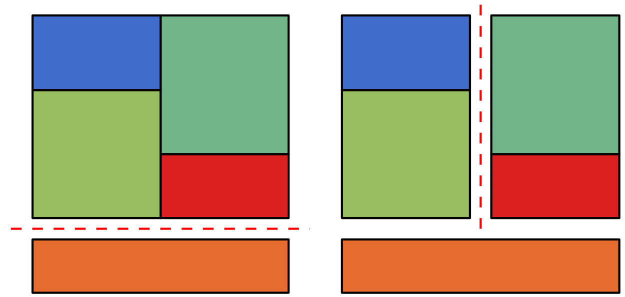
$R_n($  $)$

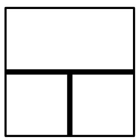
area-universal rectangulations

[Eppstein, Mumford, Speckmann, Verbeek 2012]

$R_n($  $)$

guillotine rectangulations

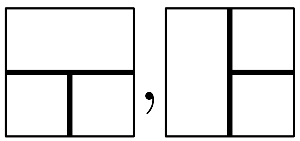


$R_n($  $)$

Catalan staircases $\quad C_n$

[Downing, Einstein, Hartung, Williams 2023]

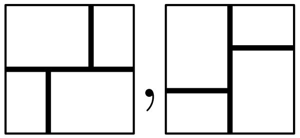
$\hookrightarrow$ HP on associahedron

$R_n($  $)$

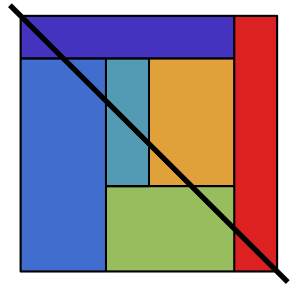
stacked rectangulations $\quad 2^n$

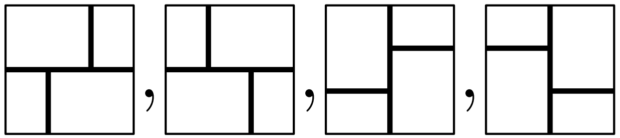
$\hookrightarrow$ HC on hypercube

Examples

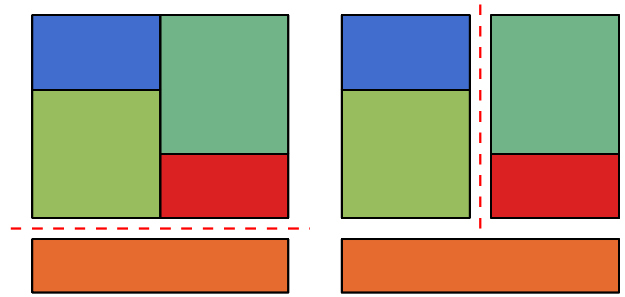
$R_n($  $)$

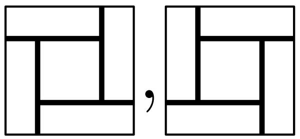
diagonal rectangulations
$\hookrightarrow$ HC on quotientope



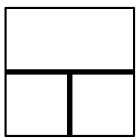
$R_n($  $)$

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

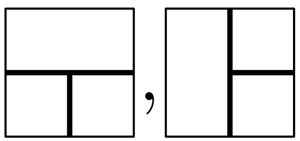


$R_n($  $)$

guillotine rectangulations

$R_n($  $)$

Catalan staircases C_n
[Downing, Einstein, Hartung, Williams 2023]
$\hookrightarrow$ HP on associahedron

$R_n($  $)$

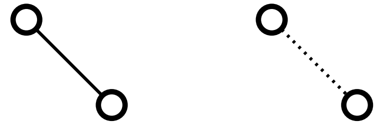
stacked rectangulations 2^n
$\hookrightarrow$ HC on hypercube

$\rightarrow$ see `www.combos.org/rect`

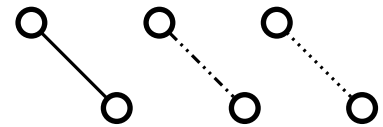
Open questions

- Generating functions for mixed tree patterns?

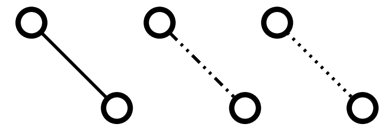
Open questions

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

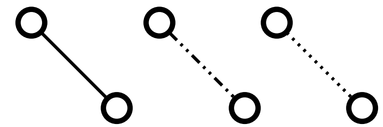
Open questions

- 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? $\rightarrow$ **[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? $\rightarrow$ **[Asinowski, Cardinal, Felsner, Fusy PP23]**
- Applications of the generation framework to other (pattern-avoiding) combinatorial objects

Thank you!