



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Combinatorial generation via permutation languages

Torsten Mütze

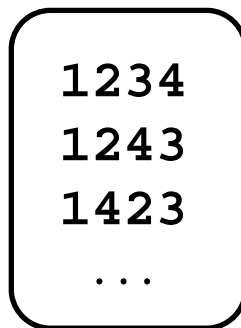
(joint work with Liz Hartung, Hung P. Hoang, and Aaron Williams)

Introduction

- In computer science, we frequently encounter different kinds of **combinatorial objects**

- **Examples:**

- permutations
- binary strings
- binary trees
- graphs
- spanning trees
- ...



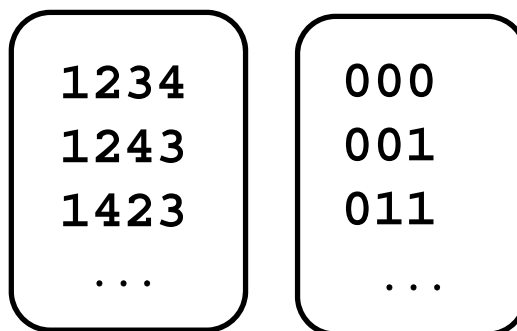
1234
1243
1423
...

Introduction

- In computer science, we frequently encounter different kinds of **combinatorial objects**

- **Examples:**

- permutations
- binary strings
- binary trees
- graphs
- spanning trees
- ...

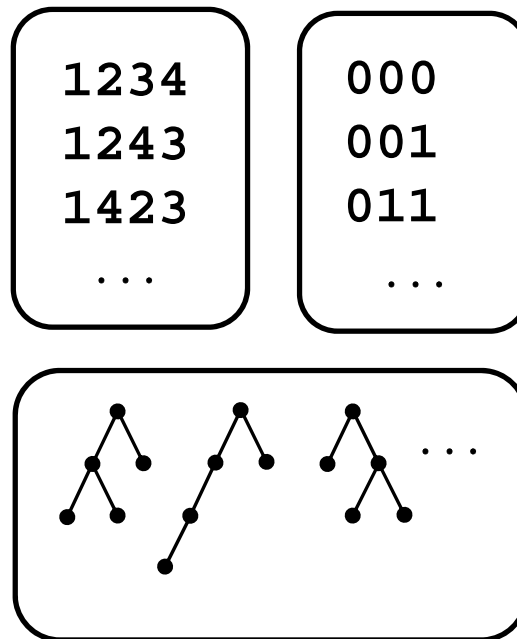


Introduction

- In computer science, we frequently encounter different kinds of **combinatorial objects**

- **Examples:**

- permutations
- binary strings
- binary trees
- graphs
- spanning trees
- ...

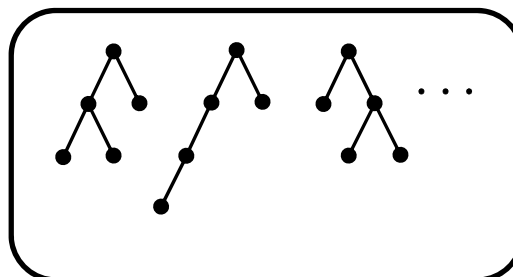
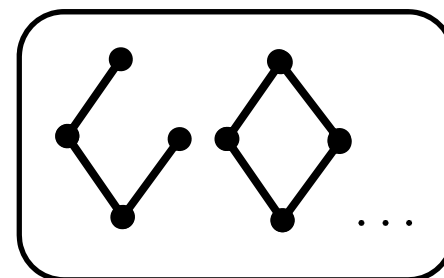
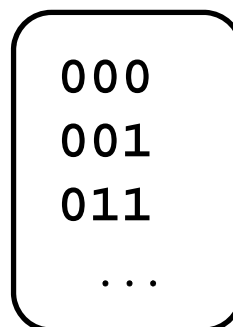
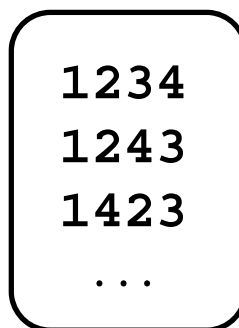


Introduction

- In computer science, we frequently encounter different kinds of **combinatorial objects**

- **Examples:**

- permutations
- binary strings
- binary trees
- graphs
- spanning trees
- ...

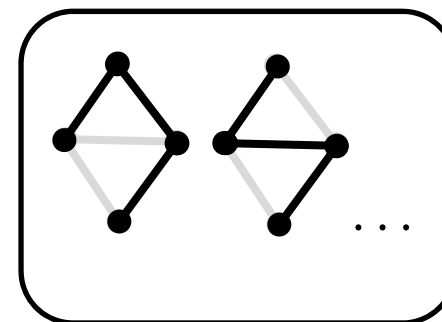
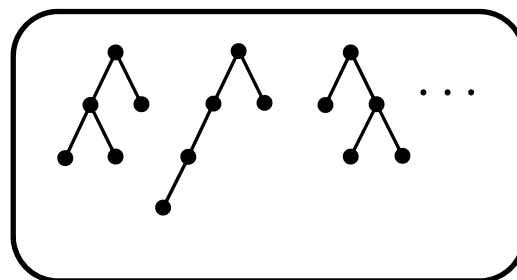
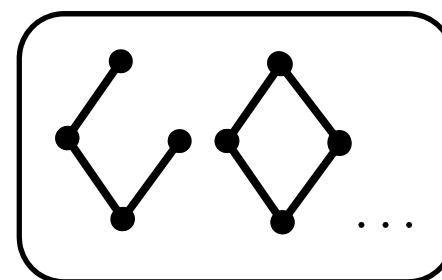
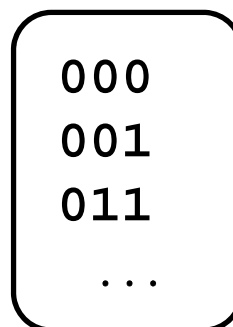
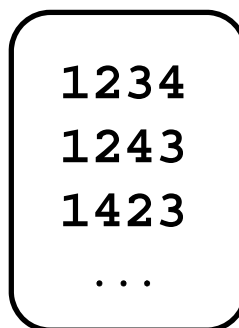


Introduction

- In computer science, we frequently encounter different kinds of **combinatorial objects**

- **Examples:**

- permutations
- binary strings
- binary trees
- graphs
- spanning trees
- ...

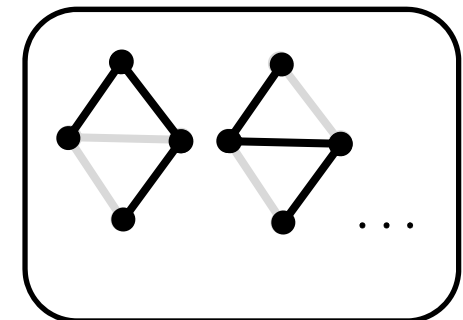
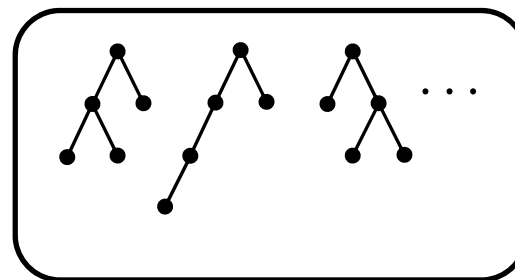
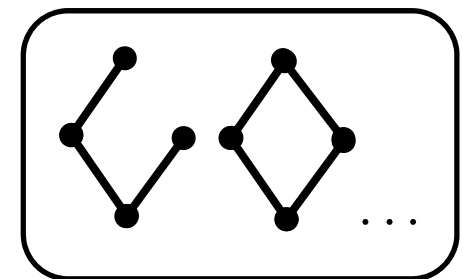
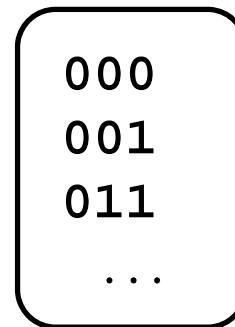
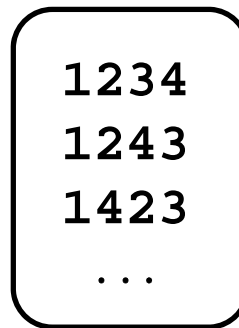


Introduction

- In computer science, we frequently encounter different kinds of **combinatorial objects**

- **Examples:**

- permutations
- binary strings
- binary trees
- graphs
- spanning trees
- ...



- **Common tasks:**

counting + random sampling + **exhaustive generation**

Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
[Knuth, The Art of Computer Programming Vol. 4A]
- ultimately: each new object in **constant** time
- consecutive objects differ only by **'small changes'** → **Gray code**

Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
[Knuth, The Art of Computer Programming Vol. 4A]
- ultimately: each new object in **constant** time
- consecutive objects differ only by **'small changes'** → **Gray code**
- **Examples:**
 - binary trees by **rotations** *[Lucas, van Baronaigien, Ruskey 93]*

Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
[Knuth, The Art of Computer Programming Vol. 4A]
- ultimately: each new object in **constant** time
- consecutive objects differ only by **'small changes'** → **Gray code**
- **Examples:**
 - binary trees by **rotations** *[Lucas, van Baronaigien, Ruskey 93]*
 - permutations by **adjacent transpositions**
(Steinhaus-Johnson-Trotter algorithm *[Johnson 63], [Trotter 62]*)

Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
[Knuth, The Art of Computer Programming Vol. 4A]
- ultimately: each new object in **constant** time
- consecutive objects differ only by **'small changes'** → **Gray code**
- **Examples:**
 - binary trees by **rotations** *[Lucas, van Baronaigien, Ruskey 93]*
 - permutations by **adjacent transpositions**
(Steinhaus-Johnson-Trotter algorithm *[Johnson 63], [Trotter 62]*)
 - binary strings by **bitflips** (binary reflected Gray code *[Gray 53]*)

Exhaustive generation

- **Goal:** generate all objects of a combinatorial class efficiently
[Knuth, The Art of Computer Programming Vol. 4A]
- ultimately: each new object in **constant** time
- consecutive objects differ only by **'small changes'** → **Gray code**
- **Examples:**
 - binary trees by **rotations** *[Lucas, van Baronaigien, Ruskey 93]*
 - permutations by **adjacent transpositions**
(Steinhaus-Johnson-Trotter algorithm *[Johnson 63], [Trotter 62]*)
 - binary strings by **bitflips** (binary reflected Gray code *[Gray 53]*)
 - set partitions by **element exchanges** *[Kaye 76]*

Exhaustive generation

- many tailormade algorithms, few general approaches
[Avis, Fukuda 92], [Li, Sawada 09], [Ruskey, Sawada, Williams 12]

Exhaustive generation

- many tailormade algorithms, few general approaches
[Avis, Fukuda 92], [Li, Sawada 09], [Ruskey, Sawada, Williams 12]
- This work: A **general framework** for **exhaustive generation**

Exhaustive generation

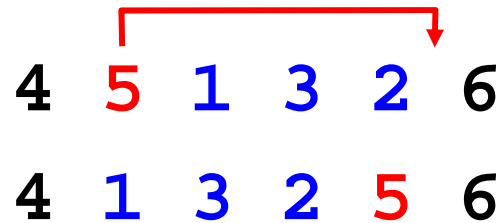
- many tailormade algorithms, few general approaches
[Avis, Fukuda 92], [Li, Sawada 09], [Ruskey, Sawada, Williams 12]
- **This work:** A **general framework** for **exhaustive generation**
- **Results:** all aforementioned algorithms as special cases
+ many new results for other families of objects

Exhaustive generation

- many tailormade algorithms, few general approaches
[Avis, Fukuda 92], [Li, Sawada 09], [Ruskey, Sawada, Williams 12]
- **This work:** A **general framework** for **exhaustive generation**
- **Results:** all aforementioned algorithms as special cases
+ many new results for other families of objects
- **Idea:** Encode objects as a subset $F_n \subseteq S_n$ of permutations of length n

Jumps

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



Jumps

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

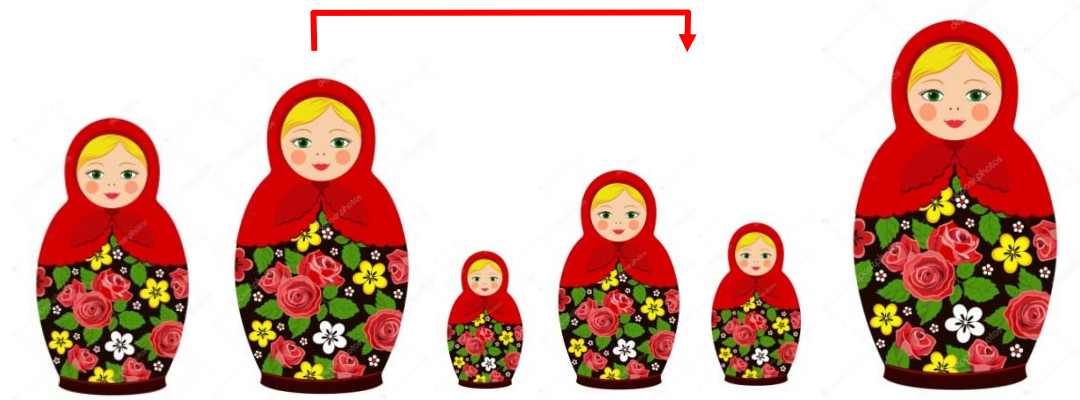
4 5 1 3 2 6
4 1 3 2 5 6



Jumps

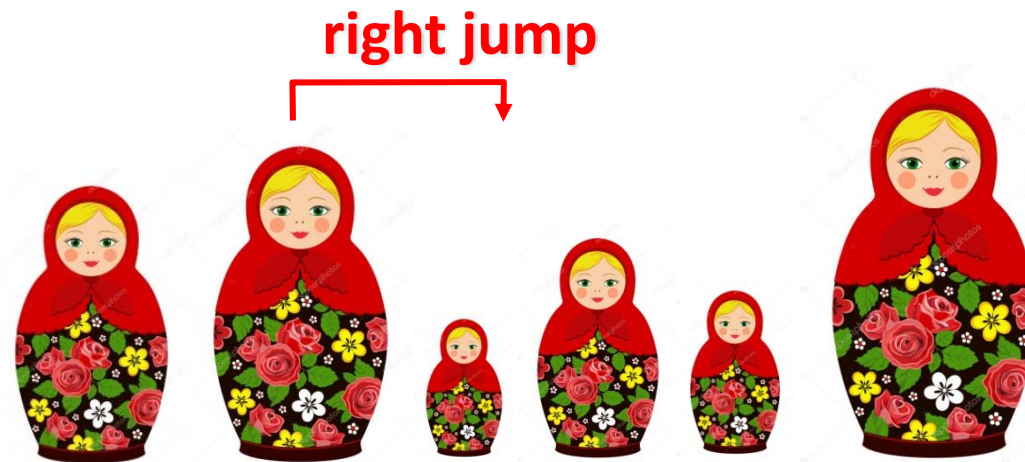
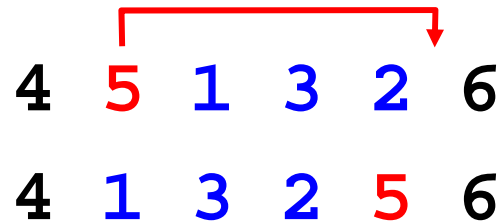
- **Jump** := move an entry in the permutation across some neighboring smaller entries (right or left)

4 5 1 3 2 6
4 1 3 2 5 6



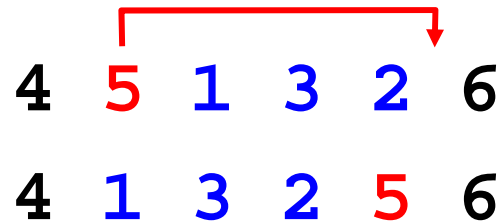
Jumps

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



Jumps

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

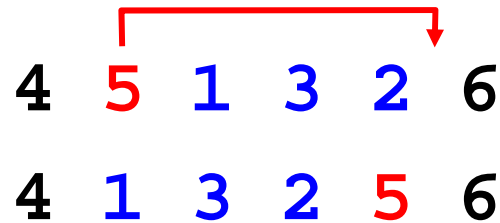


left jump



Jumps

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



Algorithm J

Algorithm J

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

- start with an initial permutation from F_n

Algorithm J

Algorithm J

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

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

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

Algorithm J

Algorithm J

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

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

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

1~~2~~43 ✓ **minimal jumps**

4~~2~~13 ✓

Algorithm J

Algorithm J


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

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

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

 1243 ✓ minimal jumps

 1243 ✗ not minimal

 4213 ✓

Algorithm J

Algorithm J

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

- start with an initial permutation from F_n
- 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: $F_4 = \{1243, 1423, 4123, 4213, 2134\}$

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

1243

1423

4123

4213

2134



4213

2134



no jump
possible

Algorithm J

Algorithm J

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

- start with an initial permutation from F_n
- 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: $F_4 = \{1243, 1423, 4123, 4213, 2134\}$

1243
1423
4123
4213
2134



4213
2134



no jump
possible

1423



direction
ambiguous

Algorithm J

Algorithm J

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

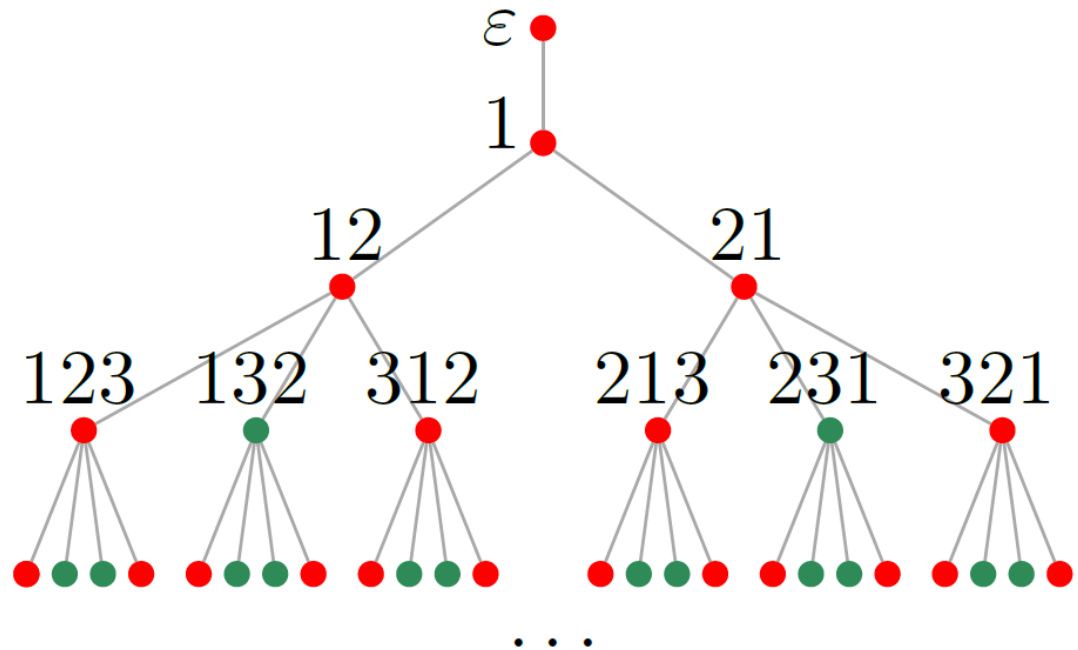
- start with an initial permutation from F_n
- in the current permutation, perform a **minimal jump** of the **largest possible value**, so that a previously unvisited permutation from F_n is created.

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

- If every permutation from F_n is visited and no ambiguity arises, we say that Algorithm J **generates** F_n (visiting twice is impossible)
- **Question:** When does Algorithm J generate F_n ?

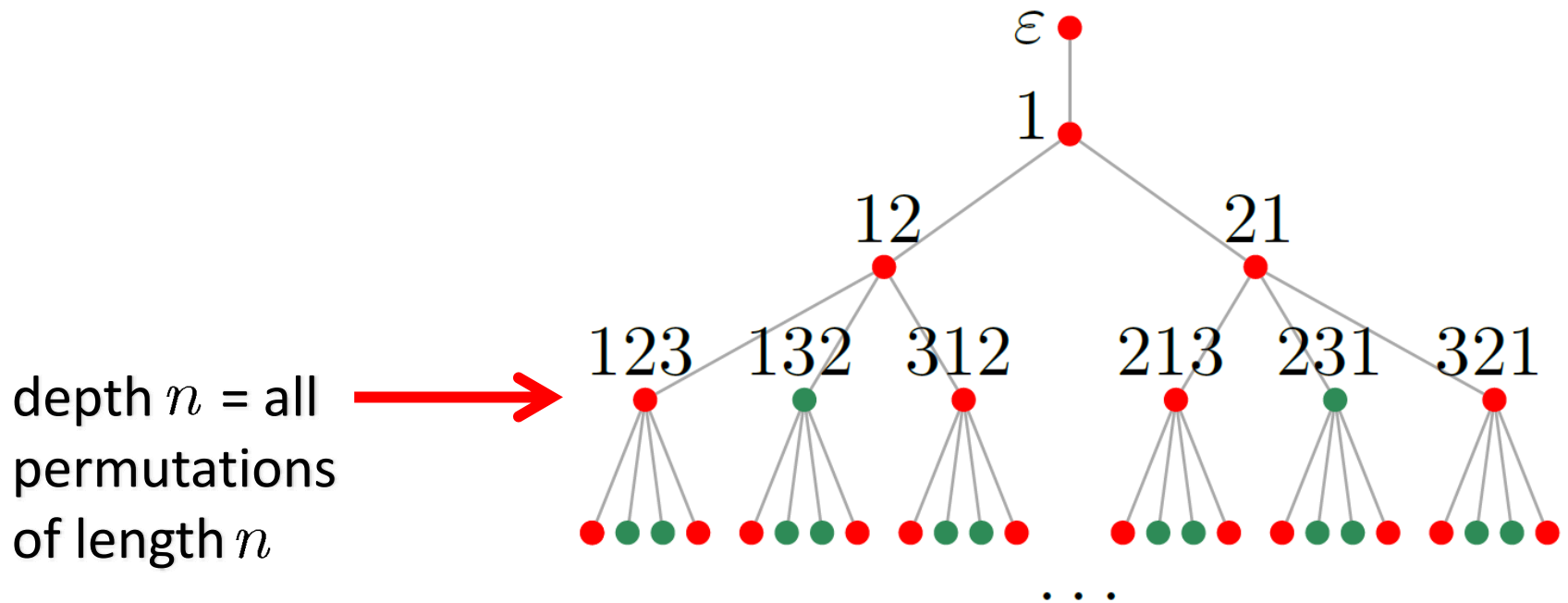
Tree of permutations

- root := empty permutation



Tree of permutations

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



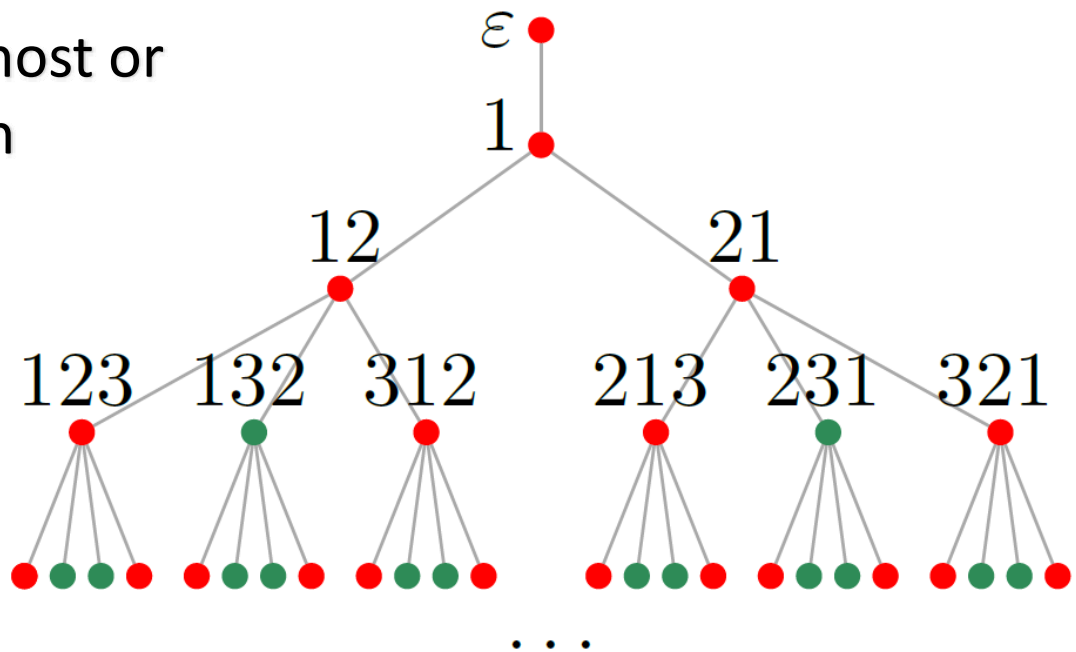
Tree of permutations

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

● symbol n at leftmost or rightmost position

● else

depth $n =$ all permutations of length n

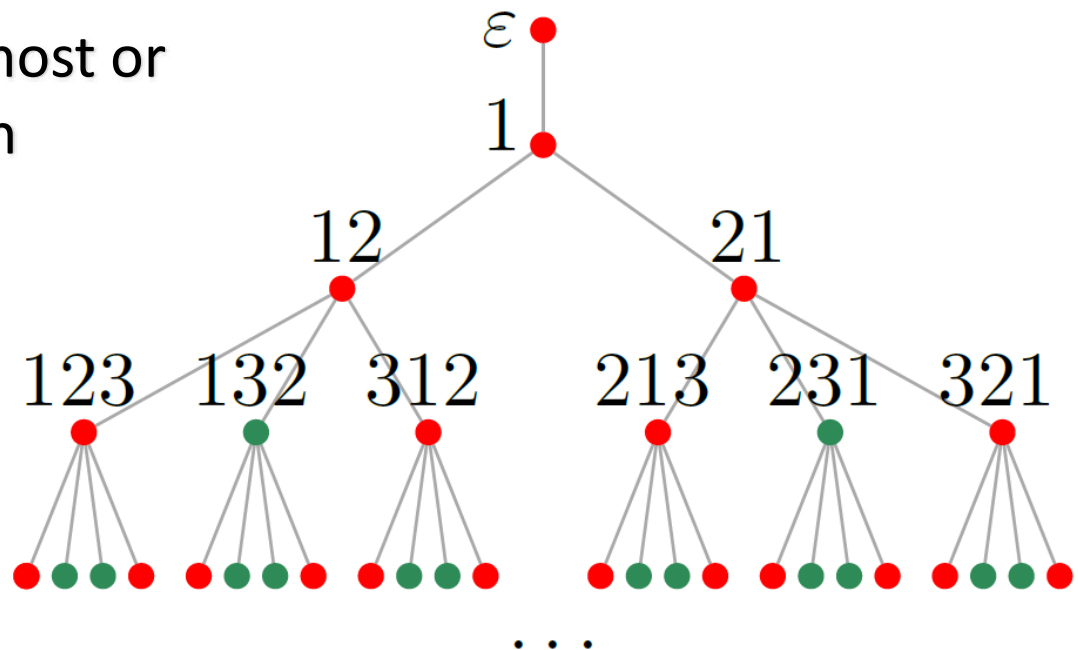


Zigzag languages

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

● symbol n at leftmost or rightmost position

● else

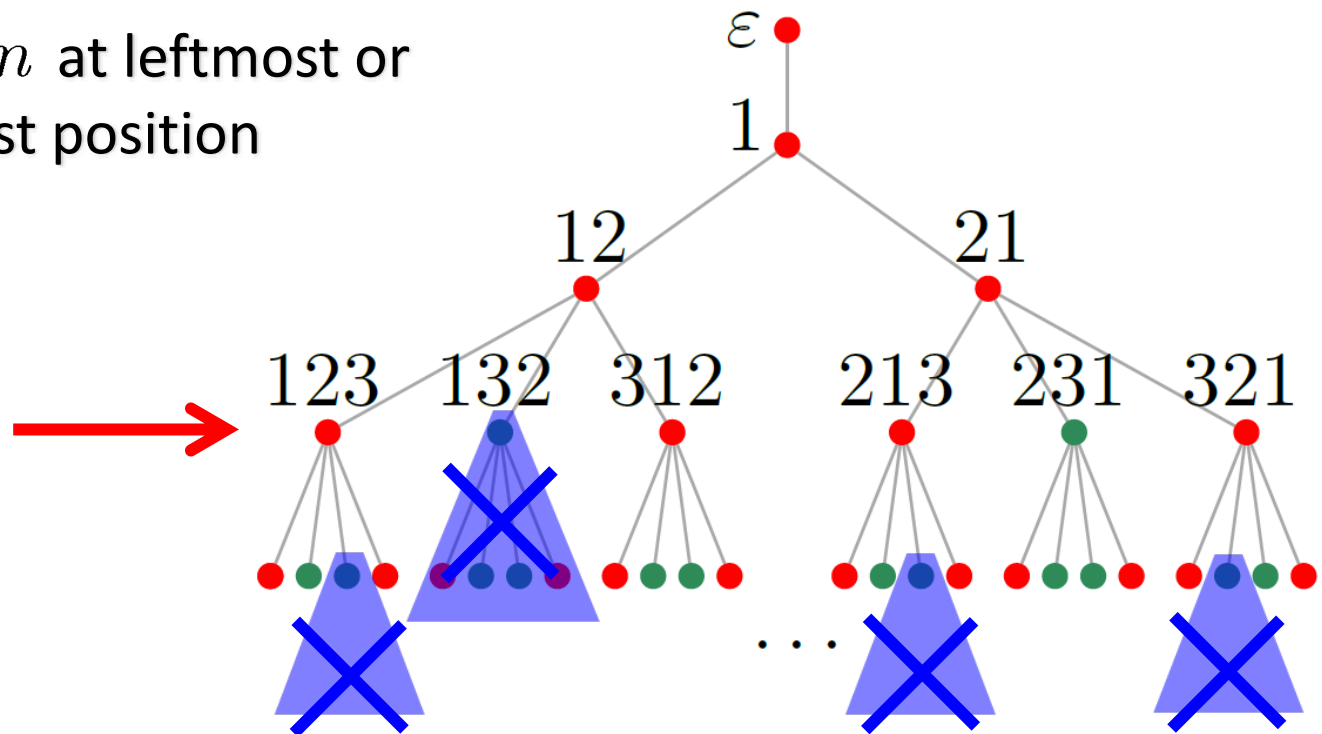


Zigzag languages

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

● symbol n at leftmost or rightmost position

● else

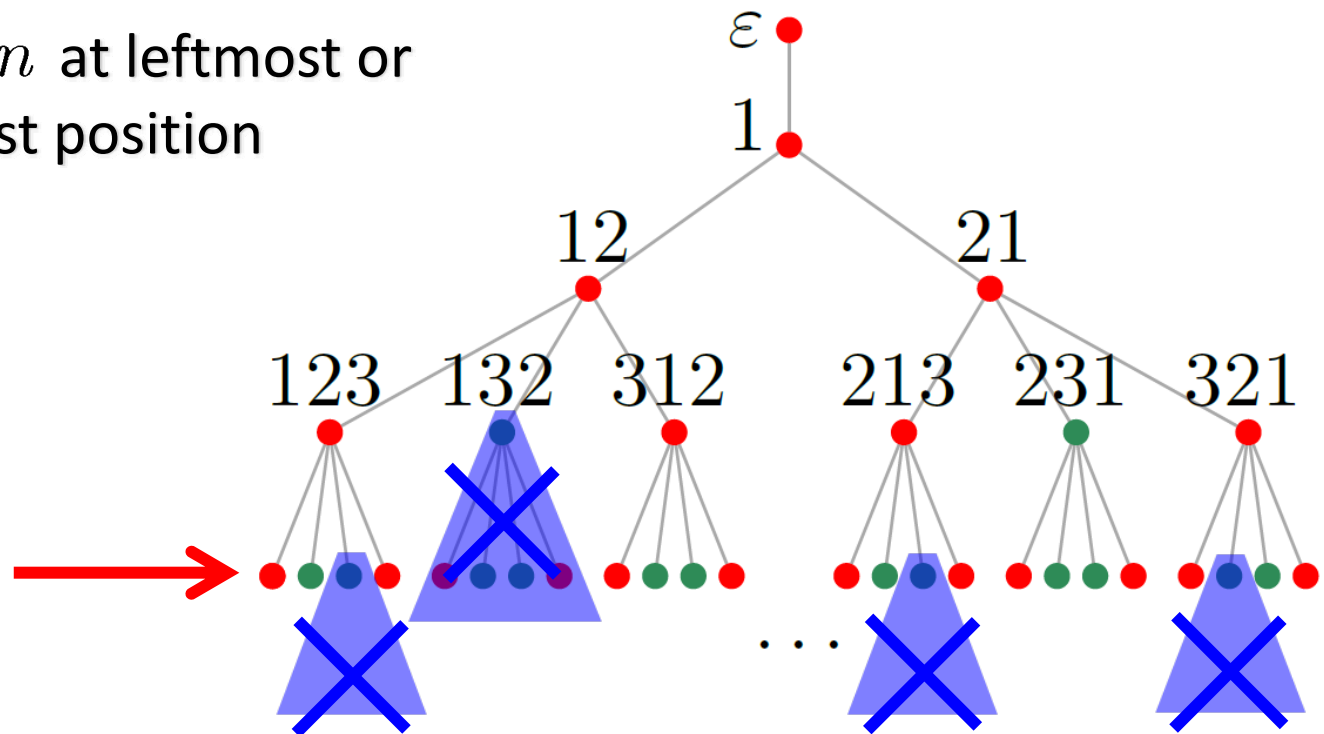


Zigzag languages

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

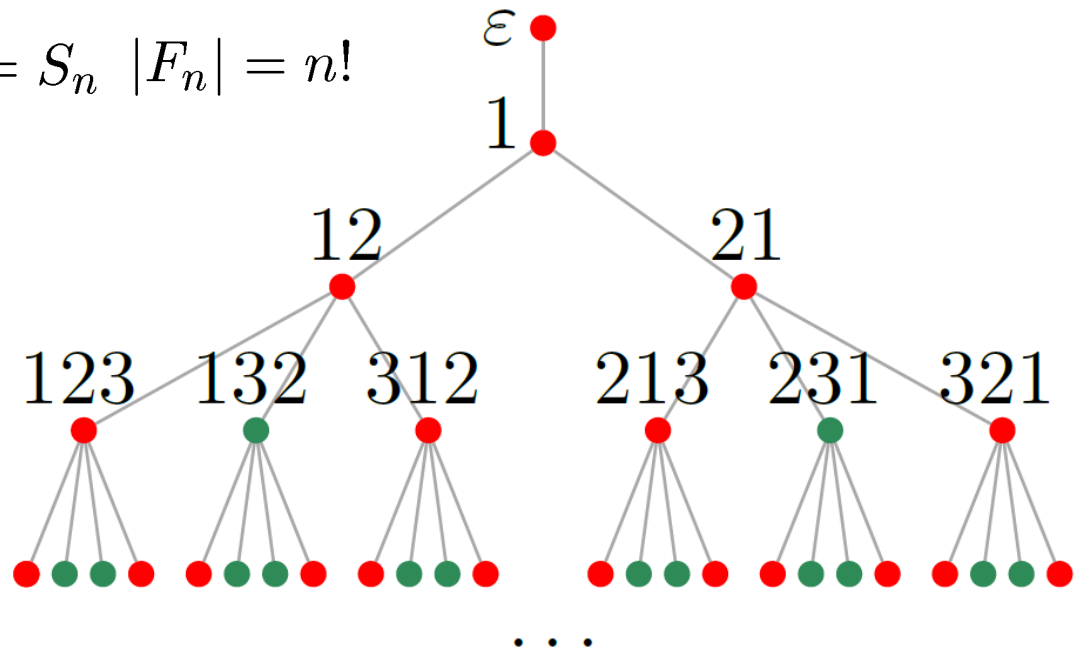
● symbol n at leftmost or rightmost position

● else



Zigzag languages

- we may prune subtrees iff their root is ●
- given any such pruned tree, a set of permutations $F_n \subseteq S_n$ in depth n is called **zigzag language**
- **Examples:**
 - prune nothing: $F_n = S_n$ $|F_n| = n!$



Zigzag languages

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

- **Examples:**

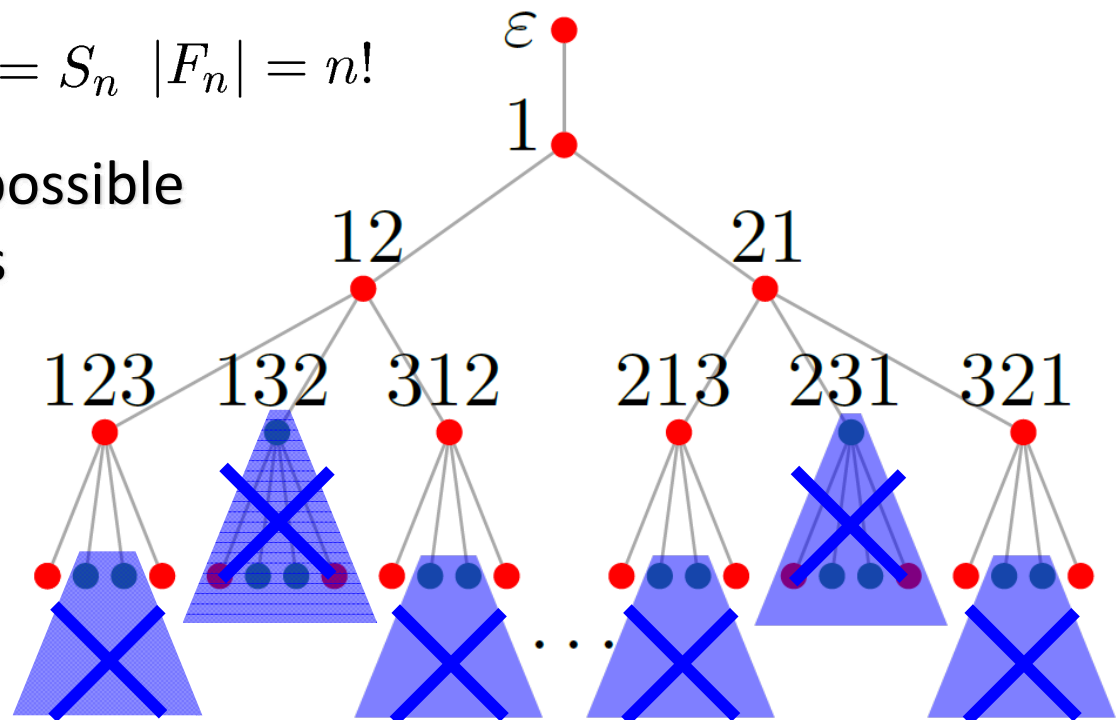
- prune nothing: $F_n = S_n$ $|F_n| = n!$

- prune everything possible


$F_n =$ permutations

without peaks

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




Zigzag languages

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

Theorem:

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

Zigzag languages


- we may prune subtrees iff their root is 
- given any such pruned tree, a set of permutations $F_n \subseteq S_n$ 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. \square

Zigzag languages

- we may prune subtrees iff their root is 
- given any such pruned tree, a set of permutations $F_n \subseteq S_n$ 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. \square

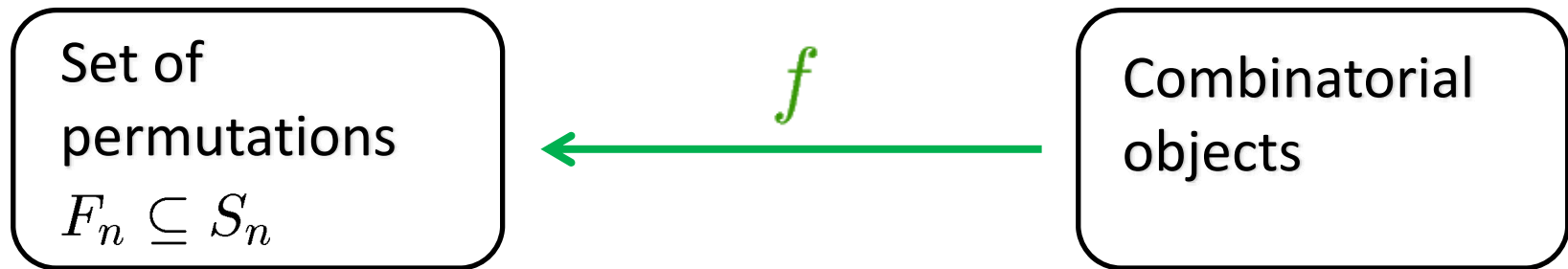
Remark: The number of zigzag languages is **enormous**

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

General approach



General approach



- Run Algorithm J

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

General approach



- Run Algorithm J

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

- Directly interpret Algorithm J under the bijection

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

General approach



- Run Algorithm J

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

- Directly interpret Algorithm J under the bijection

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

$$\mathbf{Minimal jumps} \longrightarrow \mathbf{,Small changes'}$$

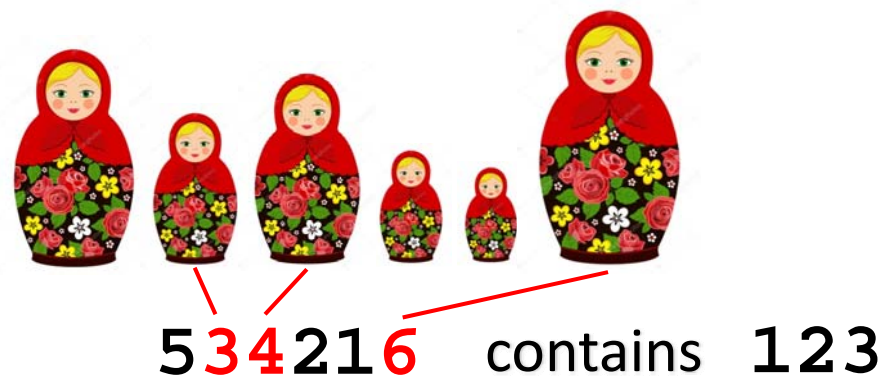
„Interesting“ zigzag languages

Applications of our framework

- pattern-avoiding permutations
- lattice congruences of the weak order on S_n

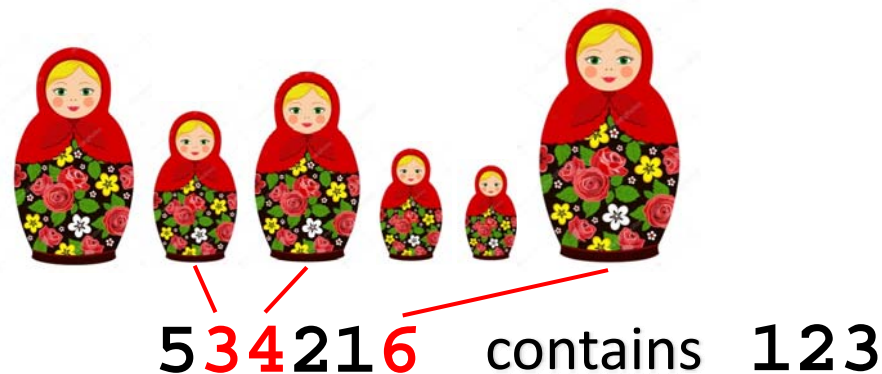
Pattern-avoiding permutations

- **Def:** A permutation π **contains a pattern** τ , if π contains a substring of entries in the same relative order as τ . Otherwise π **avoids** τ .



Pattern-avoiding permutations

- **Def:** A permutation π **contains a pattern** τ , if π contains a substring of entries in the same relative order as τ . Otherwise π **avoids** τ .



Pattern-avoiding permutations

- **Def:** A permutation π **contains a pattern** τ , if π contains a substring of entries in the same relative order as τ . Otherwise π **avoids** τ .
- $S_n(\tau_1, \tau_2, \dots, \tau_k) \subseteq S_n$ set of permutations avoiding τ_1, \dots, τ_k .

Pattern-avoiding permutations

- **Def:** A permutation π **contains a pattern** τ , if π contains a substring of entries in the same relative order as τ . Otherwise π **avoids** τ .
- $S_n(\tau_1, \tau_2, \dots, \tau_k) \subseteq S_n$ set of permutations avoiding τ_1, \dots, τ_k .
- A pattern τ is called **tame**, if the largest symbol is not at the leftmost or rightmost position.

231 ✓

123 ✗

Pattern-avoiding permutations

- **Def:** A permutation π **contains a pattern** τ , if π contains a substring of entries in the same relative order as τ . Otherwise π **avoids** τ .
- $S_n(\tau_1, \tau_2, \dots, \tau_k) \subseteq S_n$ set of permutations avoiding τ_1, \dots, τ_k .
- A pattern τ is called **tame**, if the largest symbol is not at the leftmost or rightmost position.

231 

123 

Theorem:

If τ_1, \dots, τ_k are all tame patterns, then $S_n(\tau_1, \tau_2, \dots, \tau_k)$ is a zigzag language.

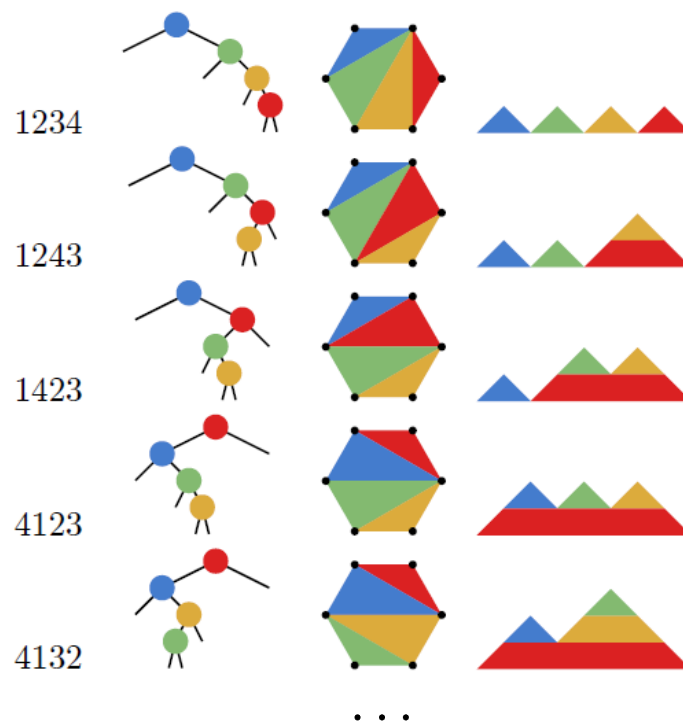
Pattern-avoiding permutations

Tame patterns \xleftrightarrow{f} Combinatorial objects

231

Catalan families

- binary trees by **rotations**
- triangulations by **edge flips**
- Dyck paths by **hill flips**



Pattern-avoiding permutations

Tame patterns \xleftrightarrow{f} Combinatorial objects

231

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

231

Set partitions by **element exchanges**

↑
**positions must
be adjacent**

1234 1|2|3|4

1243 1|2|34

1423 1|24|3

4123 14|2|3

4132 14|23

1432 1|234

1324 1|23|4

...

Pattern-avoiding permutations

Tame patterns $\overset{f}{\longleftrightarrow}$ Combinatorial objects

231

Catalan families

- binary trees by **rotations**
- triangulations by **edge flips**
- Dyck paths by **hill flips**

231

Set partitions by **element exchanges**

231, 132

Binary strings by **bitflips** (BRGC)

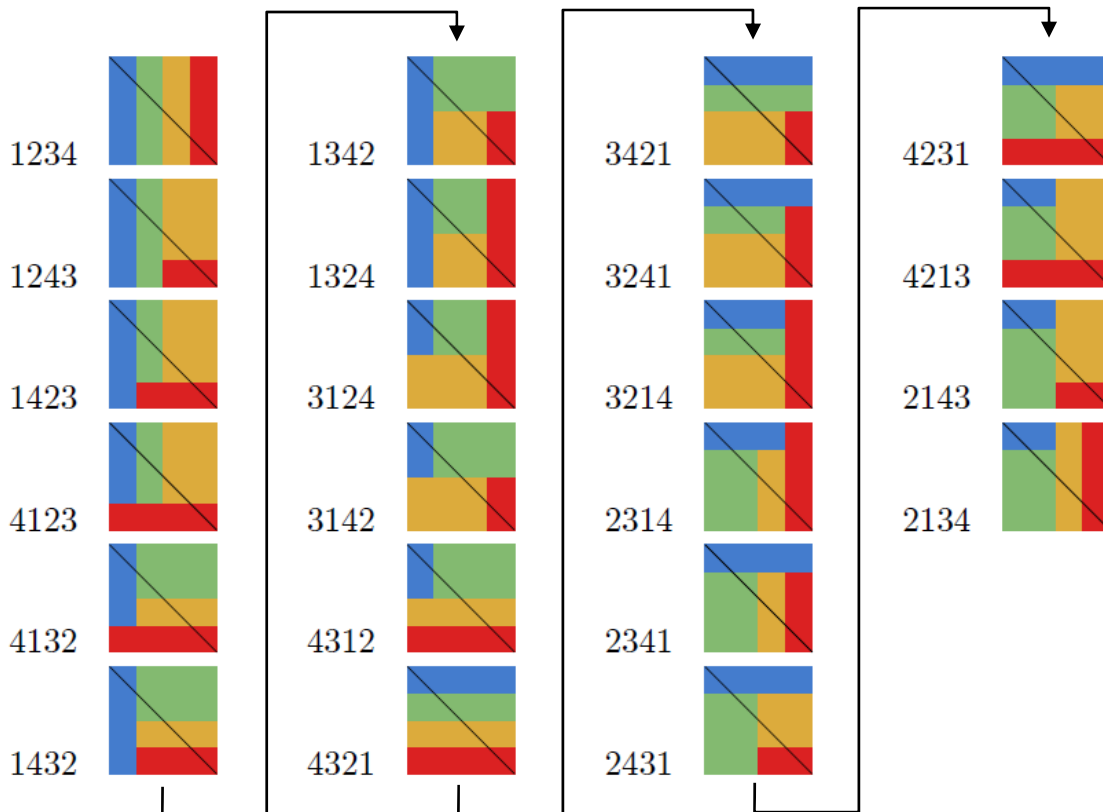
Pattern-avoiding permutations

Tame patterns \xleftrightarrow{f} Combinatorial objects

2413, 3142

Baxter permutations

Diagonal rectangulations by **flips**



Pattern-avoiding permutations

Tame patterns $\overset{f}{\longleftrightarrow}$ Combinatorial objects

2413, 3142 Diagonal rectangulations by **flips**
Baxter permutations

In addition to classical and vincular patterns:

- bivincular patterns [*Bousquet-Mélou et al. 10*]
- barred patterns [*West 90*]
- mesh patterns [*Brändén, Claesson 11*]
- monotone grid classes [*Huczynska, Vatter 06*]
- geometric grid classes [*Albert et al. 13*]
- etc.



The (Combinatorial) Object Server

Object Server



HOT



- website invented by Frank Ruskey 1995-2003 for generating combinatorial objects



The (Combinatorial) Object Server

Object Server



HOT

COS

Object

f Object



- website invented by Frank Ruskey 1995-2003 for generating combinatorial objects
- UVIC server shut-down since several years



The (Combinatorial) Object Server

Object Server



HOT



- website invented by Frank Ruskey 1995-2003 for generating combinatorial objects
- UVIC server shut-down since several years
- revived jointly with Aaron Williams and Joe Sawada
- we proudly present the new Combinatorial Object Server:
<http://combos.org>



The (Combinatorial) Object Server

Object Server

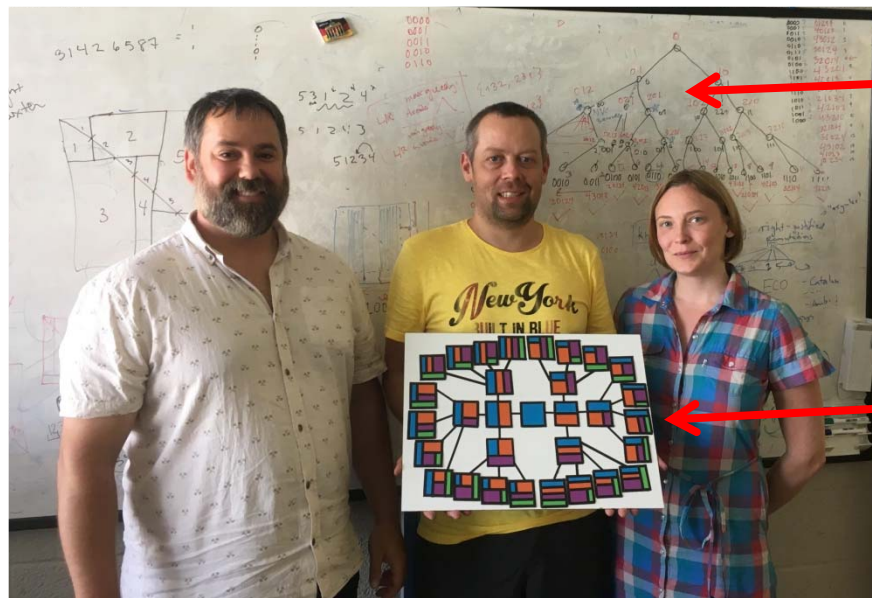


HOT



- website invented by Frank Ruskey 1995-2003 for generating combinatorial objects
- UVIC server shut-down since several years
- revived jointly with Aaron Williams and Joe Sawada
- we proudly present the new Combinatorial Object Server:
<http://combos.org>
- community project (open source, welcome your contributions)

Thank you!



tree of
permutations

Gray code for
diagonal
rectangulations

Aaron Torsten Liz