Simple permutations in permutation classes

⋆ ⋆ ⋆

A polynomial algorithm for deciding the finiteness of the number of simple permutations in permutation classes

Frédérique Bassino[1], Mathilde Bouvel[2], Adeline Pierrot[2], Dominique Rossin[3]

Permutation Patterns 2010, August 9 - 13, Dartmouth College

1: LIPN, Université Paris 13 and CNRS
2: LIAFA, Université Paris Diderot and CNRS
3: LIX, École Polytechnique and CNRS

Outline

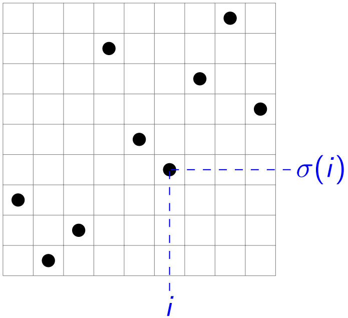
Introduction

- Context of the study
- Definitions

Permutation classes and their enumeration

Permutation: $\sigma = \sigma(1)\sigma(2)\ldots\sigma(n) = \sigma_1\sigma_2\ldots\sigma_n \in S_n$

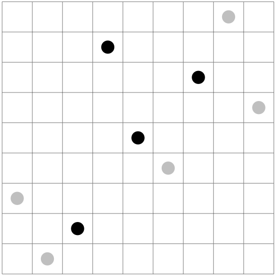
Pattern: $\pi \in S_k$ is a pattern of $\sigma \in S_n$ if $\exists\ 1 \leq i_1 < \ldots < i_k \leq n$ such that $\sigma_{i_1}\ldots\sigma_{i_k}$ is order-isomorphic to π. Denoted $\pi \leq \sigma$.



Permutation classes and their enumeration

Permutation: $\sigma = \sigma(1)\sigma(2)\ldots\sigma(n) = \sigma_1\sigma_2\ldots\sigma_n \in S_n$

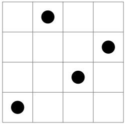
Pattern: $\pi \in S_k$ is a pattern of $\sigma \in S_n$ if $\exists\ 1 \leq i_1 < \ldots < i_k \leq n$ such that $\sigma_{i_1}\ldots\sigma_{i_k}$ is order-isomorphic to π. Denoted $\pi \leq \sigma$.



Permutation classes and their enumeration

Permutation: $\sigma = \sigma(1)\sigma(2)\ldots\sigma(n) = \sigma_1\sigma_2\ldots\sigma_n \in S_n$

Pattern: $\pi \in S_k$ is a pattern of $\sigma \in S_n$ if $\exists\ 1 \leq i_1 < \ldots < i_k \leq n$ such that $\sigma_{i_1}\ldots\sigma_{i_k}$ is order-isomorphic to π. Denoted $\pi \leq \sigma$.



Permutation classes and their enumeration

Permutation: $\sigma = \sigma(1)\sigma(2)\ldots\sigma(n) = \sigma_1\sigma_2\ldots\sigma_n \in S_n$

Pattern: $\pi \in S_k$ is a pattern of $\sigma \in S_n$ if $\exists\ 1 \leq i_1 < \ldots < i_k \leq n$ such that $\sigma_{i_1}\ldots\sigma_{i_k}$ is order-isomorphic to π. Denoted $\pi \leq \sigma$.

Permutation Class: Set $\mathcal{C}$ downward closed for $\leq$.
Characterized by its **basis** B: $\mathcal{C} = Av(B) = \{\sigma : \forall \beta \in B, \beta \not\leq \sigma\}$.
The (finite or infinite) basis is an antichain and is unique:

$$B = \{\beta \notin \mathcal{C} : \forall \pi \leq \beta \text{ such that } \pi \neq \beta, \pi \in \mathcal{C}\}.$$

Permutation classes and their enumeration

Permutation: $\sigma = \sigma(1)\sigma(2)\ldots\sigma(n) = \sigma_1\sigma_2\ldots\sigma_n \in S_n$

Pattern: $\pi \in S_k$ is a pattern of $\sigma \in S_n$ if $\exists\ 1 \leq i_1 < \ldots < i_k \leq n$ such that $\sigma_{i_1}\ldots\sigma_{i_k}$ is order-isomorphic to π. Denoted $\pi \leq \sigma$.

Permutation Class: Set $\mathcal{C}$ downward closed for $\leq$.
Characterized by its **basis** B: $\mathcal{C} = Av(B) = \{\sigma : \forall \beta \in B, \beta \not\leq \sigma\}$.
The (finite or infinite) basis is an antichain and is unique:

$$B = \{\beta \notin \mathcal{C} : \forall \pi \leq \beta \text{ such that } \pi \neq \beta, \pi \in \mathcal{C}\}.$$

Enumeration of class $\mathcal{C} = Av(B)$, with **finite** basis B:

- closed formula for $c_n = |S_n \cap \mathcal{C}|$
- generating function $\sum c_n z^n$
- recurrence on the c_n's
- ...

NB: Enumeration without being given the basis is less frequent.

Permutation classes and generating functions

Enumerating class $\mathcal{C}$ by its generating function $C(z) = \sum c_n z^n$

Structure of $\mathcal{C} \hookrightarrow$ Equations on $C(z) \hookrightarrow$ Properties of $C(z)$

Permutation classes and generating functions

Enumerating class $\mathcal{C}$ by its generating function $C(z) = \sum c_n z^n$

Structure of $\mathcal{C} \hookrightarrow$ Equations on $C(z) \hookrightarrow$ Properties of $C(z)$

Example: $\mathcal{C} = Av(231)$

- Sequence $c_n = \frac{1}{n+1}\binom{2n}{n}$
- **Algebraic** generating function $C(z) = \frac{1-\sqrt{1-4z}}{2z}$

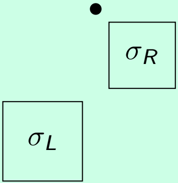
Proof:

$\sigma \in \mathcal{C} \cap S_n \Leftrightarrow \exists k \in [0..n-1]$ s.t. $\sigma = \sigma_L n \sigma_R$

with $\sigma_L \in \mathcal{C}$ on $[1..k]$

and $\sigma_R \in \mathcal{C}$ on $[k+1..n-1]$

$\Rightarrow C(z) = 1 + zC(z)^2$



Permutation classes and generating functions

Enumerating class $\mathcal{C}$ by its generating function $C(z) = \sum c_n z^n$

Structure of $\mathcal{C}$ $\hookrightarrow$ Equations on $C(z)$ $\hookrightarrow$ Properties of $C(z)$

Example: $\mathcal{C} = Av(231)$

- Sequence $c_n = \frac{1}{n+1}\binom{2n}{n}$
- **Algebraic** generating function $C(z) = \frac{1-\sqrt{1-4z}}{2z}$

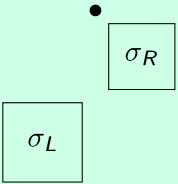
Proof:

$\sigma \in \mathcal{C} \cap S_n \Leftrightarrow \exists k \in [0..n-1]$ s.t. $\sigma = \sigma_L n \sigma_R$

with $\sigma_L \in \mathcal{C}$ on $[1..k]$

and $\sigma_R \in \mathcal{C}$ on $[k+1..n-1]$

$\Rightarrow C(z) = 1 + zC(z)^2$



Properties of the generating function $\equiv$ Structure of the class

A general sufficient condition for algebricity

Thm [Albert, Atkinson '05]
$\mathcal{C}$ contains finitely many simple permutations
$\Rightarrow \mathcal{C}$ is finitely based and has an algebraic generating function.

Sketch of the proof

Use substitution decomposition of permutations ($\equiv$ represent uniquely every permutation by its decomposition tree)

Recursive structure of the permutations in $\mathcal{C}$ ($\equiv$ Tree grammar)
$\Rightarrow$ System of equations satisfied by the generating function $C(z)$
$\Rightarrow$ Algebricity of the generating function

Finite number of simple permutations: decision

Thm [Brignall, Ruškuc, Vatter '08]
For a class $\mathcal{C} = Av(B)$ with finite basis B, it is decidable whether $\mathcal{C}$ contains a finite number of simple permutations.

Sketch of the proof

$\mathcal{C}$ contains infinitely many simple permutations iff $\mathcal{C}$ contains:

1. either infinitely many parallel alternations
2. or infinitely many wedge simple permutations
3. or infinitely many proper pin-permutations

	Decision procedure	Complexity
1. and 2. :	pattern matching of patterns of size 3 or 4 in the $\beta \in B$.	Polynomial
3. :	Decidability with automata techniques on pinwords	Decidable 2ExpTime

Main result: polynomial-time decision

Thm

For a class $\mathcal{C} = Av(B)$ with finite basis B, it is polynomial to check whether $\mathcal{C}$ contains a finite number of simple permutations.

NB: Result known for wreath-closed classes since PP2009

With $n = \max\{|\beta| : \beta \in B\}$ and $k =$ number of patterns in B, the complexity is: Steps 1. and 2.: $\mathcal{O}(k \cdot n \log n)$
Step 3.: $\mathcal{O}(n^{3k})$

NB: Step 3. in the previous procedure: $\mathcal{O}(2^{n \cdot k \cdot 2^n})$

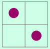
Tools for the proof

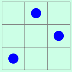
- Substitution decomposition
- Encoding by pinwords and automata techniques
- Previous results on the class of pin-permutations

Substitution for permutations

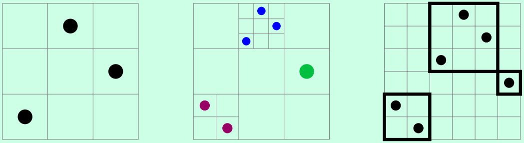
Substitution or **inflation** : $\sigma = \pi[\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(k)}]$.

Example : Here, $\pi = 1\,3\,2$, and

$\alpha^{(1)} = 2\,1 =$ 

$\alpha^{(2)} = 1\,3\,2 =$ 

$\alpha^{(3)} = 1 =$  .

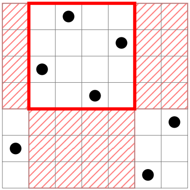


Hence $\sigma = 1\,3\,2[2\,1, 1\,3\,2, 1] = 2\,1\,4\,6\,5\,3$.

Simple permutations

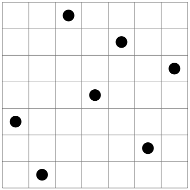
Interval (or **block**) = set of elements of σ whose positions **and** values form intervals of integers

Example: 5 7 4 6 is an interval of 2 5 7 4 6 1 3



Simple permutation = permutation that has no interval, except the trivial intervals: $1, 2, \ldots, n$ and σ

Example: 3 1 7 4 6 2 5 is simple.



The smallest simple: 1 2, 2 1, 2 4 1 3, 3 1 4 2

Substitution decomposition of permutations

Thm [AA '05]: Every σ ($\neq 1$) is uniquely decomposed as

- $\oplus[\alpha^{(1)}, \dots, \alpha^{(k)}]$, where the $\alpha^{(i)}$ are $\oplus$-indecomposable
- $\ominus[\alpha^{(1)}, \dots, \alpha^{(k)}]$, where the $\alpha^{(i)}$ are $\ominus$-indecomposable
- $\pi[\alpha^{(1)}, \dots, \alpha^{(k)}]$, where π is simple of size $k \geq 4$

NB: $\oplus = 12\dots$ and $\ominus = k\dots21$, for any $k \geq 2$

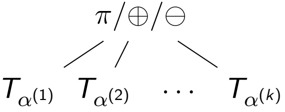
Decomposition tree:

Recursively defined as

$T_1 = \bullet$

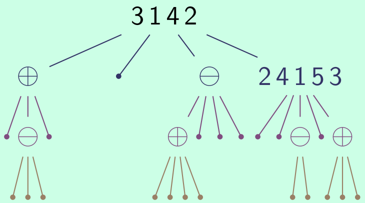
and

$T_\sigma =$



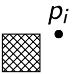
Example: Decomposition tree of

$\sigma =$ 10 13 12 11 14 1 18 19 20 21 17 16 15 4 8 3 2 9 5 6 7

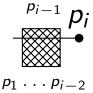


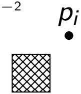
Pin representations

Pin representation of σ = sequence $(p_1, \ldots, p_n)$ s. t. each p_i satisfies

- the externality condition 

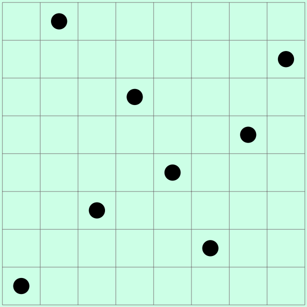
- and

• the separation condition 

• or the independence condition 

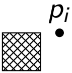
 = *bounding box* of $\{p_1, \ldots, p_{i-1}\}$

Example:

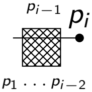


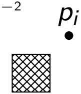
Pin representations

Pin representation of σ = sequence $(p_1, \ldots, p_n)$ s. t. each p_i satisfies

- the externality condition 

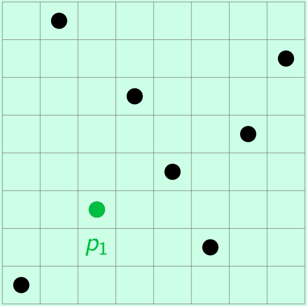
- and

• the separation condition 

• or the independence condition 

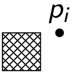
 = *bounding box* of $\{p_1, \ldots, p_{i-1}\}$

Example:

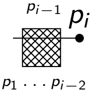


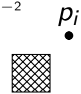
Pin representations

Pin representation of σ = sequence $(p_1, \dots, p_n)$ s. t. each p_i satisfies

- the externality condition 

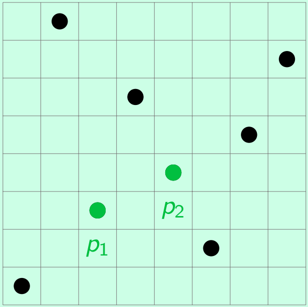
- and

• the separation condition 

• or the independence condition 

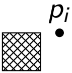
 = *bounding box* of $\{p_1, \dots, p_{i-1}\}$

Example:

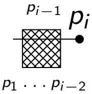


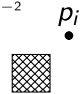
Pin representations

Pin representation of σ = sequence $(p_1, \ldots, p_n)$ s. t. each p_i satisfies

- the externality condition 

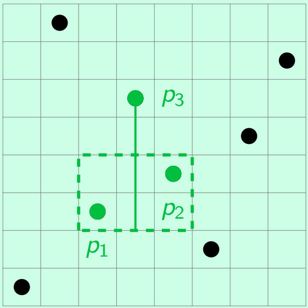
- and



- the separation condition
- or the independence condition 

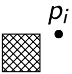
 = *bounding box* of $\{p_1, \ldots, p_{i-1}\}$

Example:

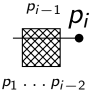


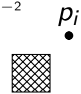
Pin representations

Pin representation of σ = sequence $(p_1, \ldots, p_n)$ s. t. each p_i satisfies

- the externality condition 

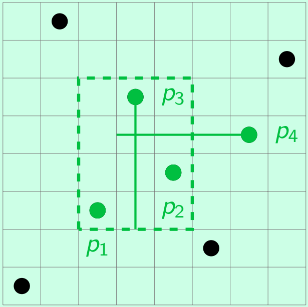
- and

• the separation condition 

• or the independence condition 

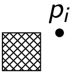
 = *bounding box* of $\{p_1, \ldots, p_{i-1}\}$

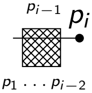
Example:

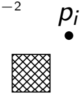


Pin representations

Pin representation of σ = sequence $(p_1, \ldots, p_n)$ s. t. each p_i satisfies

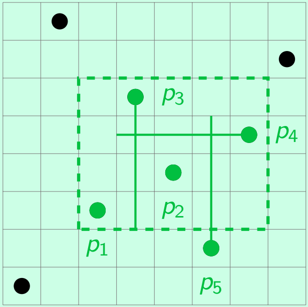
- the externality condition 
- and

- the separation condition 

- or the independence condition 

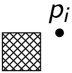
 = *bounding box* of $\{p_1, \ldots, p_{i-1}\}$

Example:

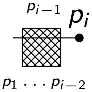


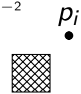
Pin representations

Pin representation of σ = sequence $(p_1, \ldots, p_n)$ s. t. each p_i satisfies

- the externality condition 

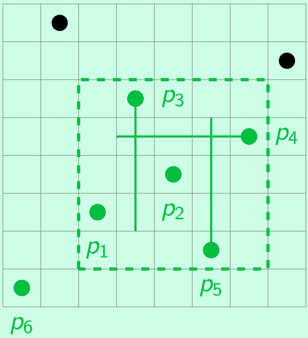
- and

- the separation condition 

- or the independence condition 

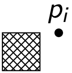
 = *bounding box* of $\{p_1, \ldots, p_{i-1}\}$

Example:

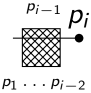


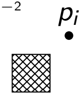
Pin representations

Pin representation of σ = sequence $(p_1, \ldots, p_n)$ s. t. each p_i satisfies

- the externality condition 

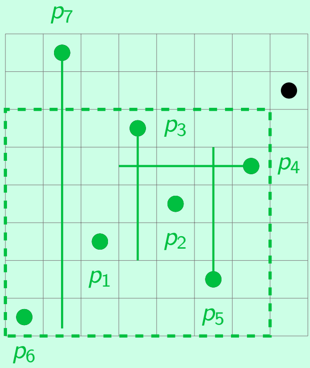
- and

- the separation condition 

- or the independence condition 

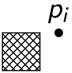
 = *bounding box* of $\{p_1, \ldots, p_{i-1}\}$

Example:

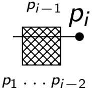


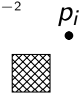
Pin representations

Pin representation of $\sigma =$ sequence $(p_1, \ldots, p_n)$ s. t. each p_i satisfies

- the externality condition 

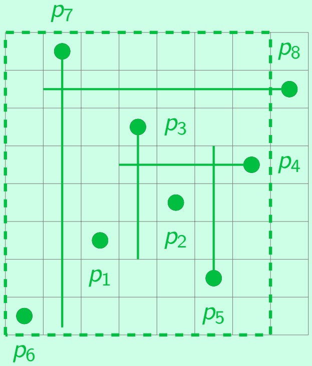
- and

• the separation condition 

• or the independence condition 

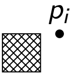
 $=$ *bounding box* of $\{p_1, \ldots, p_{i-1}\}$

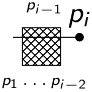
Example:

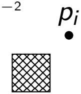


Pin representations

Pin representation of σ = sequence $(p_1, \ldots, p_n)$ s. t. each p_i satisfies

- the externality condition 

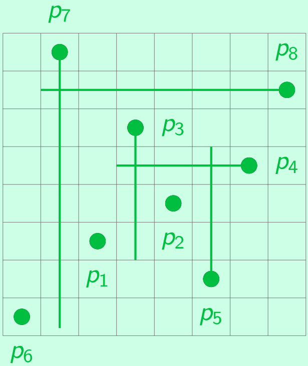
- and
- the separation condition 

- or the independence condition 

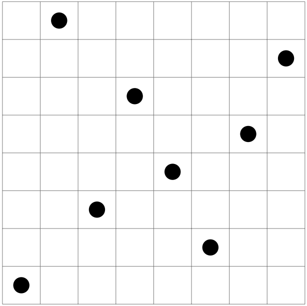
 = *bounding box* of $\{p_1, \ldots, p_{i-1}\}$

Proper pin representation = pin representation where each p_i satisfies the separation condition

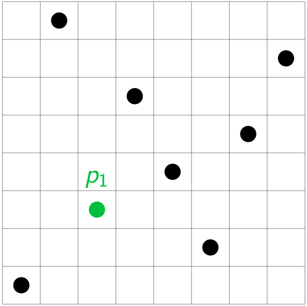
Example:



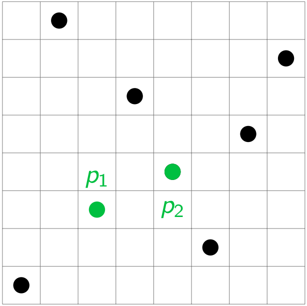
Encoding of pin representations by pinwords



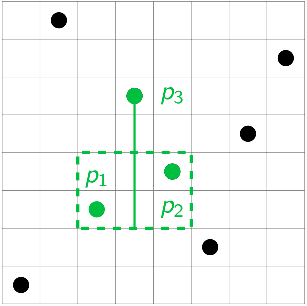
Encoding of pin representations by pinwords



Encoding of pin representations by pinwords



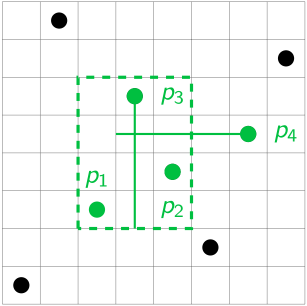
Encoding of pin representations by pinwords



U
p_3

U = up

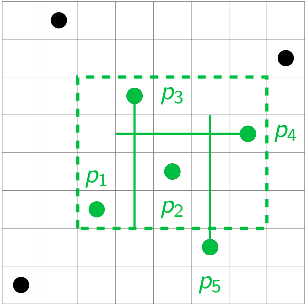
Encoding of pin representations by pinwords



U = up
R = right

U R
p_3 p_4

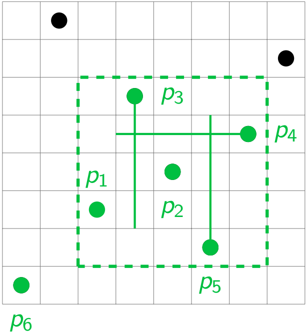
Encoding of pin representations by pinwords



U R D
p_3 p_4 p_5

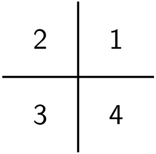
U = up
R = right
D = down

Encoding of pin representations by pinwords

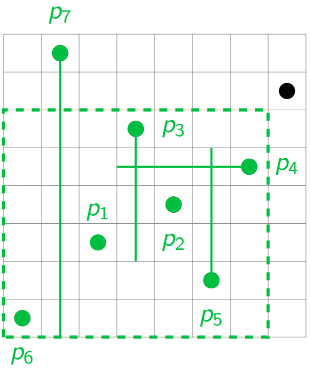


$$\begin{array}{cccc} U & R & D & 3 \\ p_3 & p_4 & p_5 & p_6 \end{array}$$

U = up
R = right
D = down
L = left

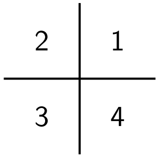


Encoding of pin representations by pinwords

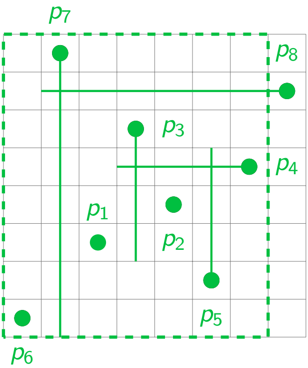


$U \quad R \quad D \quad 3 \quad U$
$p_3 \quad p_4 \quad p_5 \quad p_6 \quad p_7$

U = up
R = right
D = down
L = left

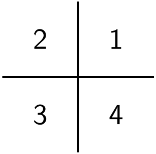


Encoding of pin representations by pinwords

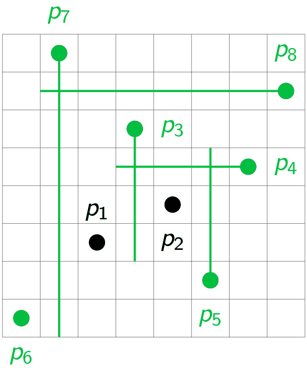


$$\begin{array}{cccccc} U & R & D & 3 & U & R \\ p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \end{array}$$

$$\begin{aligned} U &= \text{up} \\ R &= \text{right} \\ D &= \text{down} \\ L &= \text{left} \end{aligned}$$

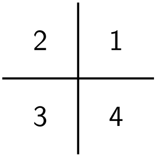


Encoding of pin representations by pinwords

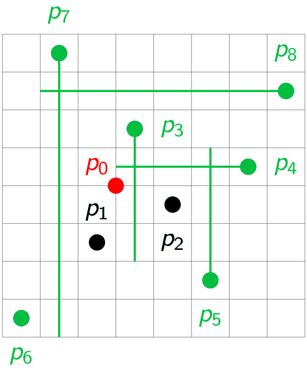


?	U	R	D	3	U	R
	p_3	p_4	p_5	p_6	p_7	p_8

U = up
R = right
D = down
L = left

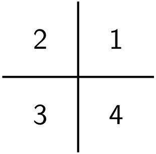


Encoding of pin representations by pinwords

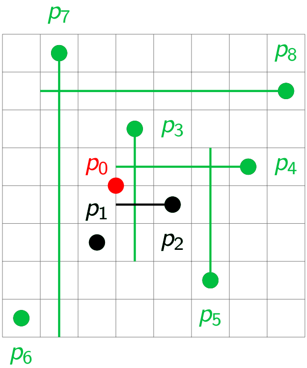


?	U	R	D	3	U	R
	p_3	p_4	p_5	p_6	p_7	p_8

U = up
R = right
D = down
L = left

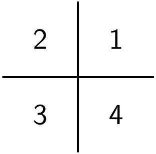


Encoding of pin representations by pinwords

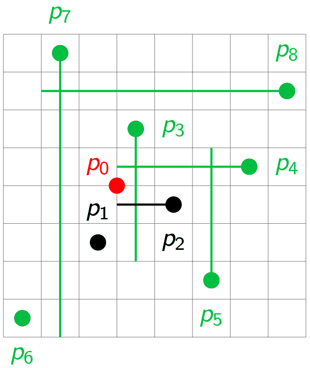


$$\begin{array}{ccccccc} R & U & R & D & 3 & U & R \\ p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \end{array}$$

U = up
R = right
D = down
L = left

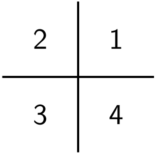


Encoding of pin representations by pinwords

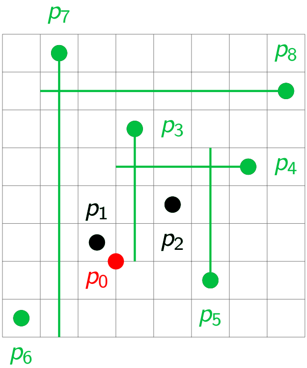


3	R	U	R	D	3	U	R
p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8

U = up
R = right
D = down
L = left



Encoding of pin representations by pinwords



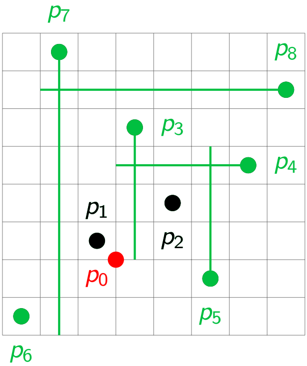
2	1	U	R	D	3	U	R
p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8

Ambiguous encoding

U = up
R = right
D = down
L = left

2	1
3	4

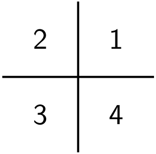
Encoding of pin representations by pinwords



$$\begin{array}{cccccccc} 2 & 1 & U & R & D & 3 & U & R \\ p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \end{array}$$

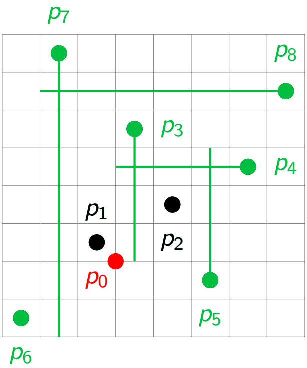
Ambiguous encoding

U = up
R = right
D = down
L = left



NB: Pinwords = words with no factor in $\{L,R\}\cdot\{L,R\} \cup \{U,D\}\cdot\{U,D\}$

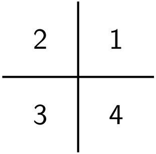
Encoding of pin representations by pinwords



2	1	U	R	D	3	U	R
p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8

Ambiguous encoding

U = up
R = right
D = down
L = left



NB: Pinwords = words with no factor in $\{L, R\} \cdot \{L, R\} \cup \{U, D\} \cdot \{U, D\}$

Strict pinwords: the only numeral is the first letter.

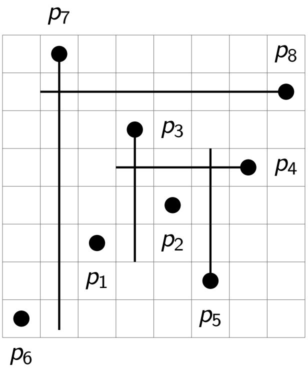
- Encode proper pin representations.
- But proper pin representations are encoded **not only** by strict pinwords!

The class of pin-permutations

Fact: Not every permutation admits (proper) pin representations.

Def: Pin-permutation = that has a pin representation.

Def: Proper pin-permutation = that has a proper pin representation.



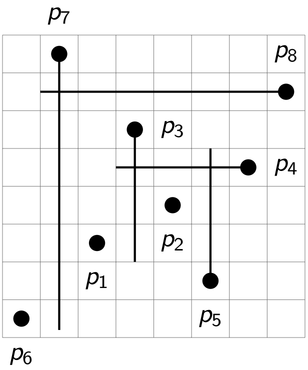
The class of pin-permutations

Fact: Not every permutation admits (proper) pin representations.

Def: Pin-permutation = that has a pin representation.

Def: Proper pin-permutation = that has a proper pin representation.

Thm: Pin-permutations are a permutation class (but proper pin-permutations are not).



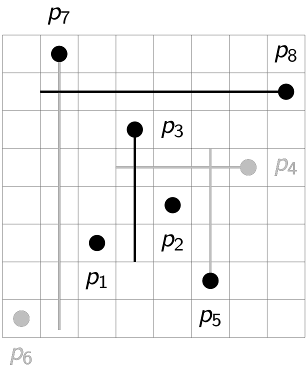
The class of pin-permutations

Fact: Not every permutation admits (proper) pin representations.

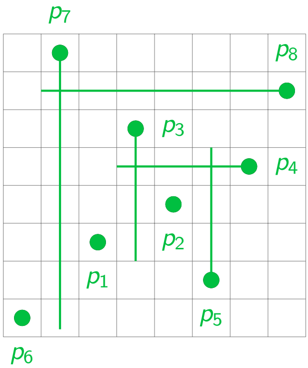
Def: Pin-permutation = that has a pin representation.

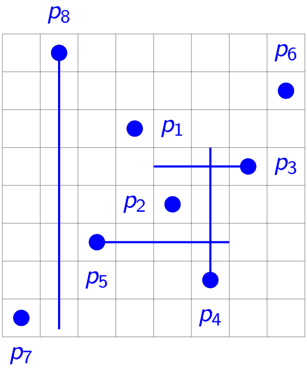
Def: Proper pin-permutation = that has a proper pin representation.

Thm: Pin-permutations are a permutation class (but proper pin-permutations are not).



Ambiguity of the encoding of pin-permutations by pinwords





Several pin representations for a single pin-permutation

Ambiguity of the encoding of pin-permutations by pinwords

σ a pin-permutation of S_n:

- at least one and possibly many pin representations of σ
- at least one and possibly many pinwords (at most 8^n)

Ambiguity of the encoding of pin-permutations by pinwords

σ a proper pin-permutation of S_n:

- at least one and possibly many proper pin representations of σ
- at least one and possibly many strict pinwords (at most 2^{n+2})

Ambiguity of the encoding of pin-permutations by pinwords

σ a proper pin-permutation of S_n:

- at least one and possibly many proper pin representations of σ
- at least one and possibly many strict pinwords (at most 2^{n+2})

- Every **proper** pin-permutations is encoded by at least one and at most 2^{n+2} strict pinwords.
- Every strict pinword encodes a **proper** pin-permutation.

Hence: Infinitely many proper pin-permutations in $\mathcal{C}$
$\Leftrightarrow$ infinitely many strict pinwords encoding permutations in $\mathcal{C}$

Proof of the main result

Thm

For a class $\mathcal{C} = Av(B)$ with finite basis B, it is polynomial to check whether $\mathcal{C}$ contains a finite number of simple permutations.

Lemma

For a class $\mathcal{C} = Av(B)$ with finite basis B, it is polynomial to check whether $\mathcal{C}$ contains a finite number of proper pin-permutations.

- Patterns on permutations and factors on words
- Computing pinwords
- Automata recognizing pinword languages
- Assembling the algorithm

How to read permutation patterns in pinwords

$\forall$ (proper pin-)permutation σ: $\sigma \in \mathcal{C} = Av(B)$ iff $\forall \beta \in B, \beta \not\leq \sigma$

How to read permutation patterns in pinwords

$\forall$ (proper pin-)permutation σ: $\sigma \in \mathcal{C} = Av(B)$ iff $\forall \beta \in B, \beta \not\leq \sigma$

Thm [BRV '08]

$\beta \in B$, σ a (proper) pin-permutation, w a (strict) pinword of σ.

$\beta \leq \sigma$ iff β is a pin-permutation and $\exists$ a pinword u encoding β s.t. $u \preceq w$

How to read permutation patterns in pinwords

$\forall$ (proper pin-)permutation σ: $\sigma \in \mathcal{C} = Av(B)$ iff $\forall \beta \in B, \beta \not\leq \sigma$

Thm [BRV '08]
$\beta \in B$, σ a (proper) pin-permutation, w a (strict) pinword of σ.
$\beta \leq \sigma$ iff β is a pin-permutation and
$\exists$ a pinword u encoding β s.t. $u \preceq w$

Def $u = u^{(1)} \dots u^{(j)}$ with each $u^{(i)}$ strict pinword.
$u \preceq w$ when $w = v^{(1)} w^{(1)} \dots v^{(j)} w^{(j)} v^{(j+1)}$ s.t. $\forall i \in \{1, \dots, j\}$:

- if $w^{(i)}$ begins with a numeral then $w^{(i)} = u^{(i)}$
- if $w^{(i)}$ begins with a direction, then
  - $v^{(i)}$ is nonempty
  - the first letter of $w^{(i)}$ corresponds to a point lying in the quadrant specified by the first letter of $u^{(i)}$
  - and all letters except the first one in $u^{(i)}$ and $w^{(i)}$ agree

Patterns as factors of ϕ(strict pinwords)

Replace numerals by directions $\Rightarrow$ factors instead of “almost factors”

ϕ: $u = u_1 u_2 \ldots u_n$ strict pinword $\mapsto \phi(u) \in \mathcal{M}$ with

$\mathcal{M} = \{L, R, U, D\}^*$ with no factor in $\{L, R\} \cdot \{L, R\} \cup \{U, D\} \cdot \{U, D\}$

Patterns as factors of ϕ(strict pinwords)

Replace numerals by directions $\Rightarrow$ factors instead of “almost factors”

ϕ: $u = u_1 u_2 \dots u_n$ strict pinword $\mapsto \phi(u) \in \mathcal{M}$ with
$\mathcal{M} = \{L, R, U, D\}^*$ with no factor in $\{L, R\} \cdot \{L, R\} \cup \{U, D\} \cdot \{U, D\}$

$\phi(u) = u'_0 u'_1 u_2 \dots u_n$ with $u'_0 u'_1$ given by

u_1	u_2	$u'_0 u'_1$
1	D or U ($\updownarrow$)	UR
	L or R ($\leftrightarrow$)	RU
	ϵ	$\{UR, RU\}$

Patterns as factors of ϕ(strict pinwords)

Replace numerals by directions $\Rightarrow$ factors instead of "almost factors"

ϕ: $u = u_1 u_2 \dots u_n$ strict pinword $\mapsto \phi(u) \in \mathcal{M}$ with

$\mathcal{M} = \{L, R, U, D\}^*$ with no factor in $\{L, R\} \cdot \{L, R\} \cup \{U, D\} \cdot \{U, D\}$

$\phi(u) = u'_0 u'_1 u_2 \dots u_n$ with $u'_0 u'_1$ given by

u_1	u_2	$u'_0 u'_1$
1	D or U ($\updownarrow$)	UR
	L or R ($\leftrightarrow$)	RU
	ϵ	$\{UR, RU\}$

u_1	u_2	$u'_0 u'_1$
2	$\updownarrow$ or $\leftrightarrow$ or ϵ	$\subseteq \{UL, LU\}$
3	$\updownarrow$ or $\leftrightarrow$ or ϵ	$\subseteq \{DL, LD\}$
4	$\updownarrow$ or $\leftrightarrow$ or ϵ	$\subseteq \{RD, DR\}$

Patterns as factors of ϕ(strict pinwords)

Replace numerals by directions $\Rightarrow$ factors instead of "almost factors"

ϕ: $u = u_1 u_2 \dots u_n$ strict pinword $\mapsto \phi(u) \in \mathcal{M}$ with
$\mathcal{M} = \{L, R, U, D\}^*$ with no factor in $\{L, R\} \cdot \{L, R\} \cup \{U, D\} \cdot \{U, D\}$

$\phi(u) = u'_0 u'_1 u_2 \dots u_n$ with $u'_0 u'_1$ given by

u_1	u_2	$u'_0 u'_1$
1	D or U ($\updownarrow$)	UR
	L or R ($\leftrightarrow$)	RU
	ϵ	$\{UR, RU\}$

u_1	u_2	$u'_0 u'_1$
2	$\updownarrow$ or $\leftrightarrow$ or ϵ	$\subseteq \{UL, LU\}$
3	$\updownarrow$ or $\leftrightarrow$ or ϵ	$\subseteq \{DL, LD\}$
4	$\updownarrow$ or $\leftrightarrow$ or ϵ	$\subseteq \{RD, DR\}$

For strict pinwords, $u \preceq w$ iff (some $x \in$) $\phi(u)$ is a factor of $\phi(w)$
(See also PP2009)

Patterns as piecewise factors of ϕ(pinwords)

Thm
For u a pinword and w a strict pinword, $u \preceq w$ iff $\phi(w) \in \mathcal{L}(u)$

Def For $u = u^{(1)}u^{(2)}\ldots u^{(j)}$ with each $u^{(i)}$ strict pinword,
$\mathcal{L}(u) = \Sigma^*\phi(u^{(1)})\Sigma^*\phi(u^{(2)})\ldots\Sigma^*\phi(u^{(j)})\Sigma^*$ with $\Sigma = \{L, R, U, D\}$

$\mathcal{L}(u)$ = words that contain $\phi(u) = (\phi(u^{(1)}), \phi(u^{(2)}), \ldots, \phi(u^{(j)}))$ as "piecewise factor"

Patterns as piecewise factors of ϕ(pinwords)

Thm
For u a pinword and w a strict pinword, $u \preceq w$ iff $\phi(w) \in \mathcal{L}(u)$

Def For $u = u^{(1)}u^{(2)} \dots u^{(j)}$ with each $u^{(i)}$ strict pinword,
$\mathcal{L}(u) = \Sigma^*\phi(u^{(1)})\Sigma^*\phi(u^{(2)})\dots\Sigma^*\phi(u^{(j)})\Sigma^*$ with $\Sigma = \{L, R, U, D\}$

$\mathcal{L}(u) =$ words that contain $\phi(u) = (\phi(u^{(1)}), \phi(u^{(2)}), \dots, \phi(u^{(j)}))$ as "piecewise factor"

Thm
$\beta \in B$, σ a proper pin-permutation, w a strict pinword of σ.
$\beta \leq \sigma$ iff β is a pin-permutation and $\exists$ a pinword u encoding β s.t. $\phi(w) \in \mathcal{L}(u)$

One step further: computing pinwords of $\beta \in B$

So far:

$\forall$ proper pin-permutation σ: $\sigma \in \mathcal{C} = Av(B)$ iff $\forall \beta \in B, \beta \not\leq \sigma$

$\beta \in B$, σ a proper pin-permutation, w a strict pinword of σ.

$\beta \not\leq \sigma$ iff β is not a pin-permutation or for all pinwords u encoding β, $\phi(w) \notin \mathcal{L}(u)$

One step further: computing pinwords of $\beta \in B$

So far:

$\forall$ proper pin-permutation σ: $\sigma \in \mathcal{C} = Av(B)$ iff $\forall \beta \in B, \beta \not\leq \sigma$

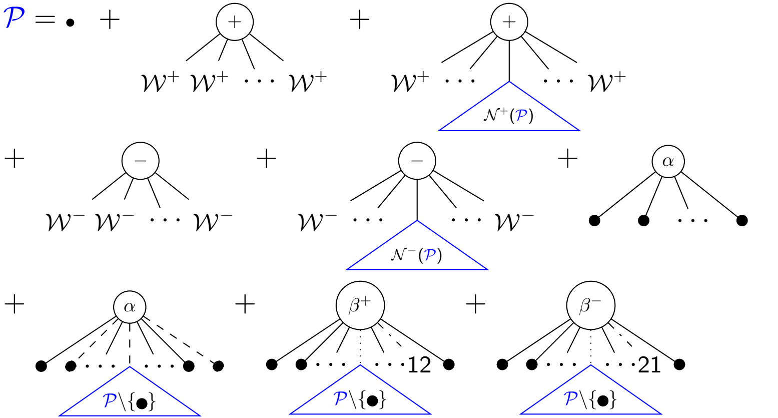
$\beta \in B$, σ a proper pin-permutation, w a strict pinword of σ.
$\beta \not\leq \sigma$ iff β is not a pin-permutation or for all pinwords u encoding β, $\phi(w) \notin \mathcal{L}(u)$

Next step:

When $\beta \in B$ is a pin-permutation, find its pinwords.
$\hookrightarrow$ Use the characterization of pin-permutations of [BBR09]

Characterization of the pin-permutation class

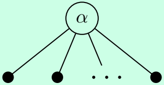
The set $\mathcal{P}$ of decomposition trees of pin-permutations satisfies:



Pinwords $P(\sigma)$ of any pin-permutation σ

For each shape of tree, compute recursively the corresponding set of pinwords.

Example:

For $\sigma =$  , i.e. σ a simple pin-permutation

$P(\sigma)$ contains at most 64 pinwords

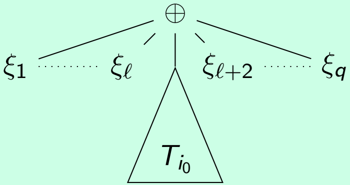
$P(\sigma)$ can be effectively computed in time $\mathcal{O}(n)$, with $n = |\sigma|$

Pinwords $P(\sigma)$ of any pin-permutation σ

For each shape of tree, compute recursively the corresponding set of pinwords.

Example:

For $\sigma =$



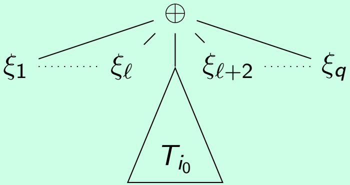
, $T_{i_0} \notin \mathcal{W}^+$, $\forall i \ \xi_i \in \mathcal{W}^+$

Pinwords $P(\sigma)$ of any pin-permutation σ

For each shape of tree, compute recursively the corresponding set of pinwords.

Example:

For $\sigma =$



, $T_{i_0} \notin \mathcal{W}^+$, $\forall i\ \xi_i \in \mathcal{W}^+$

Set $P^{(k)}(\xi_i) =$ pinwords of ξ_i with origin p_0 in quadrant k,

$\mathfrak{P}^{(1)}_{(j)} = \left(P^{(1)}(\xi_j), P^{(1)}(\xi_{j-1}), \ldots, P^{(1)}(\xi_1)\right)$

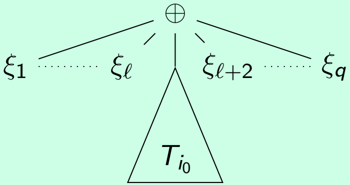
and $\mathfrak{P}^{(3)}_{(j)} = \left(P^{(3)}(\xi_j), P^{(3)}(\xi_{j+1}), \ldots, P^{(3)}(\xi_q)\right)$

Pinwords $P(\sigma)$ of any pin-permutation σ

For each shape of tree, compute recursively the corresponding set of pinwords.

Example:

For $\sigma =$



, $T_{i_0} \notin \mathcal{W}^+$, $\forall i \ \xi_i \in \mathcal{W}^+$

If σ does not satisfy any special condition (H)

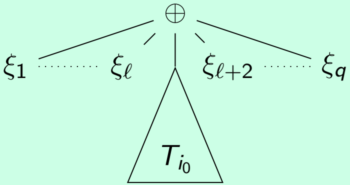
then $P(\sigma) = P_0 = P(T_{i_0}) \cdot \mathfrak{P}^{(1)}_{(\ell)} \sqcup \mathfrak{P}^{(3)}_{(\ell+2)}$

Pinwords $P(\sigma)$ of any pin-permutation σ

For each shape of tree, compute recursively the corresponding set of pinwords.

Example:

For $\sigma =$



, $T_{i_0} \notin \mathcal{W}^+$, $\forall i\ \xi_i \in \mathcal{W}^+$

$$(2H1) \begin{cases} \xi_\ell = \bullet = y \\ \xi_{\ell+2} = \bullet = x \\ T_{i_0} = \ominus[\bullet, S] \end{cases}$$

If σ satisfies Condition $(2H1)$ then $P(\sigma) = P_0 \cup P_1 \cup P_2$, with

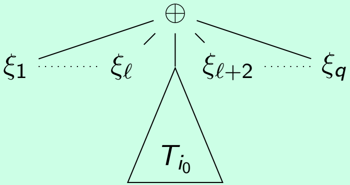
$$P_1 = \underbrace{P(S)\cdot 1 \cdot L}_{x \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \sqcup \mathfrak{P}^{(3)}_{(\ell+3)},\ P_2 = \underbrace{P(S)\cdot 3 \cdot U}_{y \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell-1)} \sqcup \mathfrak{P}^{(3)}_{(\ell+2)}$$

Pinwords $P(\sigma)$ of any pin-permutation σ

For each shape of tree, compute recursively the corresponding set of pinwords.

Example:

For $\sigma =$



$, T_{i_0} \notin \mathcal{W}^+, \forall i \; \xi_i \in \mathcal{W}^+$

If σ satisfies Condition ...

Proof: Technical... and many cases...
Analyze the behavior of a pin representation w.r.t. the block of σ

One more step: automata recognizing $\cup_{u \in P(\beta)} \mathcal{L}(u), \beta \in B$

So far:

$\forall$ proper pin-permutation σ: $\sigma \in \mathcal{C} = Av(B)$ iff $\forall \beta \in B, \beta \not\leq \sigma$

β pin-permutation $\mapsto P(\beta) =$ set of pinwords encoding β

$\beta \in B$, σ a proper pin-permutation, w a strict pinword of σ.

$\beta \not\leq \sigma$ iff β is not a pin-permutation or $\phi(w) \notin \cup_{u \in P(\beta)} \mathcal{L}(u)$

One more step: automata recognizing $\cup_{u \in P(\beta)} \mathcal{L}(u), \beta \in B$

So far:

$\forall$ proper pin-permutation σ: $\sigma \in \mathcal{C} = Av(B)$ iff $\forall \beta \in B, \beta \not\leq \sigma$

β pin-permutation $\mapsto P(\beta) =$ set of pinwords encoding β

$\beta \in B$, σ a proper pin-permutation, w a strict pinword of σ.
$\beta \not\leq \sigma$ iff β is not a pin-permutation or
$\phi(w) \notin \cup_{u \in P(\beta)} \mathcal{L}(u)$

Next step: When $\beta \in B$ is a pin-permutation, describe the language $\cup_{u \in P(\beta)} \mathcal{L}(u)$ by a **deterministic** automaton

Determinism and mirror languages

- As before, use the recursive characterization of the pin-permutation class:

$\hookrightarrow$ For each shape of tree of a pin-permutation σ, compute a deterministic automaton recognizing $\mathcal{L}(\sigma) = \cup_{u \in P(\sigma)} \mathcal{L}(u)$.

Determinism and mirror languages

- As before, use the recursive characterization of the pin-permutation class:

$\hookrightarrow$ For each shape of tree of a pin-permutation σ, compute a deterministic automaton recognizing $\overleftarrow{\mathcal{L}(\sigma)} = \cup_{u \in P(\sigma)} \overleftarrow{\mathcal{L}(u)}$.

Determinism and mirror languages

- As before, use the recursive characterization of the pin-permutation class:

$\hookrightarrow$ For each shape of tree of a pin-permutation σ, compute a deterministic automaton recognizing $\overleftarrow{\mathcal{L}(\sigma)} = \cup_{u \in P(\sigma)} \overleftarrow{\mathcal{L}(u)}$.

Why the mirror?

- Common suffixes in pinwords of $P(\sigma)$
- But several choices for the beginning of $u \in P(\sigma)$

$\hookrightarrow$ Reading for the end allows determinism

Determinism is key to have a polynomial complexity.

Determinism and mirror languages

- As before, use the recursive characterization of the pin-permutation class:

$\hookrightarrow$ For each shape of tree of a pin-permutation σ, compute a deterministic automaton recognizing $\overleftarrow{\mathcal{L}(\sigma)} = \cup_{u \in P(\sigma)} \overleftarrow{\mathcal{L}(u)}$.

Why the mirror?

- Common suffixes in pinwords of $P(\sigma)$
- But several choices for the beginning of $u \in P(\sigma)$

$\hookrightarrow$ Reading for the end allows determinism

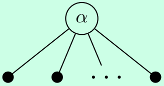
Determinism is key to have a polynomial complexity.

Recall that $\mathcal{L}(u) = \Sigma^* \phi(u^{(1)}) \Sigma^* \phi(u^{(2)}) \dots \Sigma^* \phi(u^{(j)}) \Sigma^*$

Deterministic automaton recognizing $\overleftarrow{\mathcal{L}(\sigma)}$

Recursive construction on the shape of the tree of σ:

Example:

For $\sigma =$ 

, i.e. σ a simple pin-permutation

Compute $P(\sigma)$ (at most 64 pinwords, strict or quasi-strict)

$\overleftarrow{\mathcal{L}(\sigma)} =$ words with a **factor** in $\{\overleftarrow{\phi(u)} : u \in P(\sigma)\}$

NB: small extension of ϕ to quasi-strict pin-words

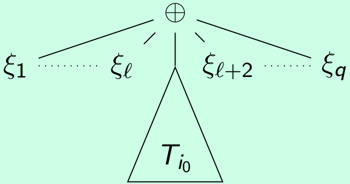
Aho-Corasick: linear-time construction of a deterministic automaton $\mathcal{A}_\sigma$ recognizing $\overleftarrow{\mathcal{L}(\sigma)}$

Deterministic automaton recognizing $\overleftarrow{\mathcal{L}(\sigma)}$

Recursive construction on the shape of the tree of σ:

Example:

For $\sigma =$



, $T_{i_0} \notin \mathcal{W}^+$, $\forall i\ \xi_i \in \mathcal{W}^+$

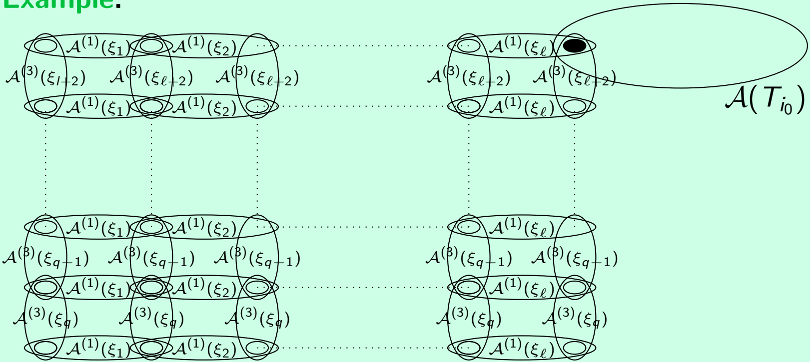
If σ does not satisfy any special condition (H)

then $P(\sigma) = P_0 = P(T_{i_0}) \cdot \mathfrak{P}^{(1)}_{(\ell)} \sqcup \mathfrak{P}^{(3)}_{(\ell+2)}$

Deterministic automaton recognizing $\overleftarrow{\mathcal{L}(\sigma)}$

Recursive construction on the shape of the tree of σ:

Example:

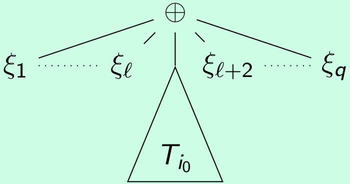


Deterministic automaton recognizing $\overleftarrow{\mathcal{L}(\sigma)}$

Recursive construction on the shape of the tree of σ:

Example:

For $\sigma =$



, $T_{i_0} \notin \mathcal{W}^+$, $\forall i\ \xi_i \in \mathcal{W}^+$

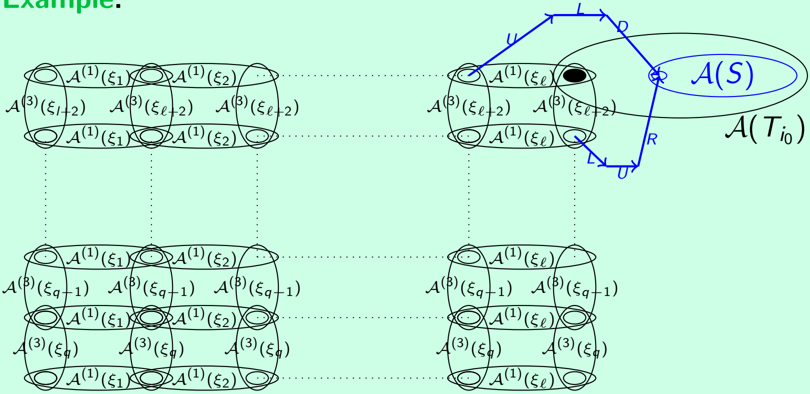
$$(2H1) \begin{cases} \xi_\ell = \bullet = y \\ \xi_{\ell+2} = \bullet = x \\ T_{i_0} = \ominus[\bullet, S] \end{cases}$$

If σ satisfies Condition $(2H1)$ then $P(\sigma) = P_0 \cup P_1 \cup P_2$, with ...
$\Rightarrow$ Add shortcuts to marked states of $\mathcal{A}(T_{i_0})$, corresponding to words added to $P(\sigma)$

Deterministic automaton recognizing $\overleftarrow{\mathcal{L}(\sigma)}$

Recursive construction on the shape of the tree of σ:

Example:



Complexity of the construction

	Time complexity	Size of $\mathcal{A}_\sigma$
Non recursive cases	up to $\mathcal{O}(n^3)$	up to $\mathcal{O}(n^3)$
Recursive cases	up to $\mathcal{O}(n^2)$ $+$ recursive computation	up to $\mathcal{O}(n^2)$ $+$ recursive size

Thm For any pin-permutation σ, we can build a deterministic automaton $\mathcal{A}_\sigma$ recognizing $\overleftarrow{\mathcal{L}(\sigma)} = \cup_{u \in P(\sigma)} \overleftarrow{\mathcal{L}(u)}$
Complexity (time and space): $\mathcal{O}(n^3)$ where $n = |\sigma|$

Almost there

So far:

$\forall$ proper pin-permutation σ: $\sigma \in \mathcal{C} = Av(B)$ iff $\forall \beta \in B, \beta \not\leq \sigma$

β pin-permutation $\mapsto P(\beta) =$ set of pinwords encoding β

$\beta \in B$, σ a proper pin-permutation, w a strict pinword of σ.

$\beta \not\leq \sigma$ iff β is not a pin-permutation or $\phi(w) \notin \cup_{u \in P(\beta)} \mathcal{L}(u)$

iff β is not a pin-permutation or $\phi(w)$ is not accepted by $\mathcal{A}_\beta$

Almost there

So far:

$\forall$ proper pin-permutation σ: $\sigma \in \mathcal{C} = Av(B)$ iff $\forall \beta \in B, \beta \not\leq \sigma$

β pin-permutation $\mapsto P(\beta) =$ set of pinwords encoding β

$\beta \in B$, σ a proper pin-permutation, w a strict pinword of σ.
$\beta \not\leq \sigma$ iff β is not a pin-permutation or $\phi(w) \notin \cup_{u \in P(\beta)} \mathcal{L}(u)$
iff β is not a pin-permutation or $\phi(w)$ is not accepted by $\mathcal{A}_\beta$

Final step:

- Build the automaton accepting the language of words of the form $\phi(w)$ (for w strict pinword) that are not accepted by any $\mathcal{A}_\beta$ (for $\beta \in B$ **and** β **pin-permutation**)
- Test the finiteness of the corresponding language

The missing first step

Find the pin-permutations $\beta \in B$!

Algorithm to test if a simple permutation σ is a pin-permutation

- using properties of pin representation in [BBR '09]

$\hookrightarrow$ linear-time procedure

Algorithm to test if a permutation σ is a pin-permutation:

- compute the decomposition tree of σ
- test whether its shape corresponds to pin-permutation trees
- check that the simple permutations in the tree are pin-permutations

$\hookrightarrow$ linear-time procedure

Overview of the algorithm

Goal: Check the finiteness of the number of proper pin-permutations in $\mathcal{C} = Av(B)$, i.e. check the finiteness of the number of strict pinwords encoding permutations in $\mathcal{C}$

Overview of the algorithm

Goal: Check the finiteness of the number of proper pin-permutations in $\mathcal{C} = Av(B)$, i.e. check the finiteness of the number of ϕ(strict pinwords) encoding permutations in $\mathcal{C}$

Overview of the algorithm

Goal: Check the finiteness of the number of proper pin-permutations in $\mathcal{C} = Av(B)$, i.e. check the finiteness of the number of ϕ(strict pinwords) encoding permutations in $\mathcal{C}$

Procedure:

- Find the pin-permutations $\beta \in B$
- Compute the automata $\mathcal{A}_\beta$
- Compute the automaton $\mathcal{A} = (\cup \mathcal{A}_\beta)^c \cap \mathcal{A}(\mathcal{M})$

  NB Use product construction for union to preserve determinism
- Test whether $L(\mathcal{A})$ is infinite i.e. whether $\mathcal{A}$ contains a cycle

Overview of the algorithm

Goal: Check the finiteness of the number of proper pin-permutations in $\mathcal{C} = Av(B)$, i.e. check the finiteness of the number of ϕ(strict pinwords) encoding permutations in $\mathcal{C}$

Procedure:

- Find the pin-permutations $\beta \in B$
- Compute the automata $\mathcal{A}_\beta$
- Compute the automaton $\mathcal{A} = (\cup \mathcal{A}_\beta)^c \cap \mathcal{A}(\mathcal{M})$

  NB Use product construction for union to preserve determinism
- Test whether $L(\mathcal{A})$ is infinite i.e. whether $\mathcal{A}$ contains a cycle

Complexity: $\mathcal{O}(n^{3k})$ in time and space
where $n = \max\{|\beta| : \beta \in B\}$ and $k =$ number of patterns in B

Main result

Thm There is a $\mathcal{O}(k \cdot n \log n)$ procedure to test whether $\mathcal{C} = Av(B)$ contains finitely many parallel alternations (resp. wedge simple permutations).

Thm There is a $\mathcal{O}(n^{3k})$ procedure to test whether $\mathcal{C} = Av(B)$ contains finitely proper pin-permutations

Thm There is a $\mathcal{O}(n^{3k})$ procedure to test whether $\mathcal{C} = Av(B)$ contains finitely simple permutations
(which is a sufficient condition for $C(z)$ to be algebraic)

Conclusion

So far:

- Finite number of simple permutations in $\mathcal{C}$: sufficient condition for $C(z)$ to be algebraic
- Polynomial procedure to test this condition

Next step:

- Compute the set of simple permutations in $\mathcal{C}$

$\hookrightarrow$ [AA '05] gives a procedure, but very high complexity

- Compute the generating function $C(z)$

$\hookrightarrow$ Provide an algorithm from the proof of [AA '05]

Further perspectives:

- Random generation in (wreath-closed) permutation classes
- Implementation in a library