Pattern avoidance in lattice paths

Valerie Roitner

VSM, 14.5.2020

Lattices

A *lattice* $\Lambda = (V, E)$ is a mathematical model of a discrete space. It consists of a set $V \subset \mathbb{R}^d$ of vertices and a set $E \subseteq V \times V$ of edges. If two vertices are connected via an edge, we call them *nearest neighbours*.
An important subclass of lattices are *periodic* lattices. A lattice is called periodic if the there are vectors $v_1, \ldots, v_k$ such that the lattice is mapped to itself under any translation of the form $\sum_{j=1}^{k} \alpha_j v_j$ where $\alpha_j \in \mathbb{Z}$ for $j = 1, \ldots, k$.

Lattices

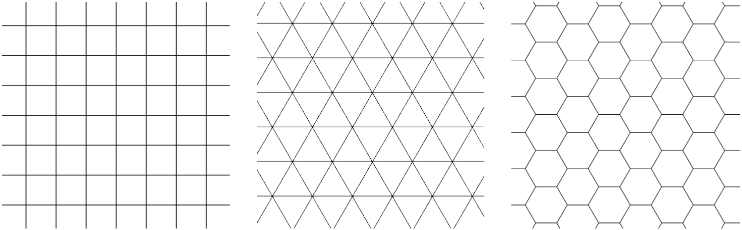
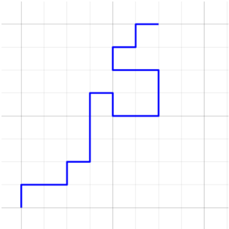


Figure: Three examples of periodic lattices. From left to right: the Euclidean (or square) lattice $\mathbb{Z}^2$, the triangular lattice and the hexagonal lattice.

Lattice paths

A n-step lattice path or lattice walk on a lattice $\Lambda = (V, E)$ from $s \in V$ to $x \in V$ is a sequence $w = (w_0, w_1, \ldots, w_n)$ of vertices such that

1. $w_0 = s$ and $w_n = x$
2. $(w_i, w_i + 1) \in E$ for $i = 0, \ldots, n - 1$

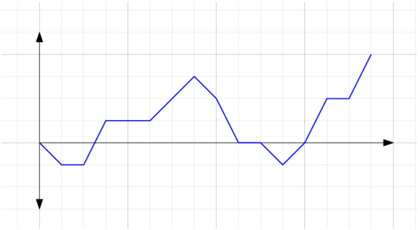


Lattice paths

Alternative definition (in $\mathbb{Z}^d$):
An n-step lattice path from $s \in \mathbb{Z}^d$ to $x \in \mathbb{Z}^d$ relative to a step set $\mathcal{S}$ is a sequence $w = (w_0, w_1, \ldots, w_n)$ of points in $\mathbb{Z}^d$ such that

1. $w_0 = s$ and $w_n = x$
2. $(w_i, w_i + 1) \in \mathcal{S}$ for $i = 0, \ldots, n-1$

Advantage: more compact form.
Note: step set defined globally, same structure at each vertex.
In this talk: step set always finite. Underlying lattice: $\mathbb{Z}^2$



Lattice paths

Applications of lattice paths in mathematical models:

- in physics: wetting and melting processes, Brownian motion
- in biology / biochemistry: models for polymers (e.g. DNA)
- birth-death-processes
- in computer sciences: queues, analysis of algorithms
- ...

Bijections with other mathematical objects:

- trees
- Young tableaux
- triangulations of n-gons
- ...

Lattice paths

length of a step: its first entry u_i
length of a walk/path: sum of the length of its steps, $|w| = u_1 + \cdots + u_m$
size of a walk: number of steps (does not always coincide with length)
final altitude of a walk: sum of altitudes of its steps (second entry v_i), i.e., $\text{alt}(w) = v_1 + \cdots + v_m$.

A lattice path in $\mathbb{Z}^2$ is called *directed* if all its steps have positive first coordinate.
A lattice path is in $\mathbb{Z}^2$ called *simple* if all of its steps are of the form $(1, b)$. These objects are essentially one-dimensional objects and their size always corresponds to their length.

Lattice paths

Weighted lattice paths: each step is associated with a weight.
weight of a path: product of the weight of its steps.
Often used choices of weights are:

- Combinatorial paths in the standard sense: $w_j = 1$ for all steps.
- Paths with coloured steps: $w_j \in \mathbb{Z}^+$.
- Probabilistic models: $\sum_j w_j = 1$ and $w_j \in (0, 1]$.

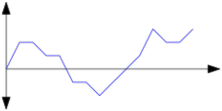
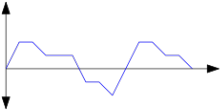
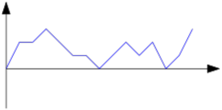
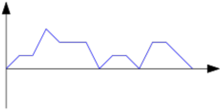
Step polynomial:

$$P(t, u) = \sum_{s \in \mathcal{S}} w_s t^{|s|} u^{\mathrm{alt}(s)}.$$

Lattice paths

- walk: unconstrained lattice path.
- bridge: lattice path whose endpoint lies on the x-axis.
- meander: lattice path that lies in the quarter-plane $\mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$. For directed lattice paths, this is equivalent to lattice paths that never attain negative altitude.
- excursion: bridge and meander.

Lattice paths

	ending anywhere	ending at 0
unconstrained on $\mathbb{Z}$	walks  $W(t,1) = \frac{1}{1-tP(1)}$	bridges  $B(t) = W_0(t) = t\sum_{i=1}^{c} \frac{u_i'(t)}{u_i(t)}$
constrained on $\mathbb{Z}_{\geq 0}$	meanders  $M(t,1) = \frac{1}{1-tP(1)} \prod_{i=1}^{c}(1-u_i(t))$	excursions  $E(t) = M_0(t) = \frac{(-1)^{c-1}}{p_{-c}t} \prod_{i=1}^{c} u_i(t)$

Generating functions for walks, bridges, excursions and meanders (Banderier, Flajolet, 2002).

Patterns

A pattern p is a fixed path/word

$$p = [a_1, \ldots, a_\ell]$$

where $a_i \in \mathcal{S}$.
Length of a pattern …sum of the lengths of its steps.

Patterns

A pattern p is a fixed path/word

$$p = [a_1, \ldots, a_\ell]$$

where $a_i \in \mathcal{S}$.
Length of a pattern … sum of the lengths of its steps.

Occurrence of a pattern p … contiguous sub-string of the path w, which coincides with p.
A path w avoids the pattern p … no occurrence of p in w.

Patterns

A pattern p is a fixed path/word

$$p = [a_1, \ldots, a_\ell]$$

where $a_i \in \mathcal{S}$.
Length of a pattern ... sum of the lengths of its steps.

Occurrence of a pattern p ... contiguous sub-string of the path w, which coincides with p.
A path w avoids the pattern p ... no occurrence of p in w.

Example: $w = [1, 3, 3, 1, -2, 3, 1]$ (where i stands for the step $(1, i)$) has two occurrences of the pattern $p = [3, 1]$ but avoids the pattern $\tilde{p} = [-2, -2]$

Formal power series, generating functions

Formal power series

$$A(z) := \sum_{n\geq 0} a_n z^n = a_0 + a_1 z + a_2 z^2 + \dots$$

Correspondence: sequence $\leftrightarrow$ formal power series (generating functions)

$$(a_0, a_1, a_2, \dots) \leftrightarrow a_0 + a_1 z + a_2 z^2 + \dots$$

Combinatorial constructions correspond to arithmetic operations

- disjoint union $\leftrightarrow$ sum of power series
- Cartesian product $\leftrightarrow$ Cauchy product of series
- sequences of objects from class $\mathcal{A}$ $\leftrightarrow$ geometric series $\frac{1}{1-A(x)}$
- ...

What is the kernel method?

The kernel method is a tool to study generating functions that satisfy functional equations.

Main idea: bind variables in a way such that one side of the equation vanishes.

What the kernel method is not

The (combinatorial) kernel method has nothing do do with the kernel method or kernel trick in statistics or machine learning.

The Beginnings

Exercise:

Consider a word composed of n 'S' symbols and n $'X'$ symbols, where S stands for 'add an element' to some specific stack and X stands for 'remove an element' from the stack. Such a word is called *admissible* if it specifies no operations that cannot be performed – i.e. if the number of X's never exceeds the number of S's when read from left to right. Find the number of admissible words as a function of n.

The Beginnings

Exercise:

Consider a word composed of n 'S' symbols and n $'X'$ symbols, where S stands for 'add an element' to some specific stack and X stands for 'remove an element' from the stack. Such a word is called *admissible* if it specifies no operations that cannot be performed – i.e. if the number of X's never exceeds the number of S's when read from left to right. Find the number of admissible words as a function of n.

D.E. Knuth. *The art of computer programming. Vol 1: Fundamental algorithms.* Addison-Wesley Publishing Co., 1968. Exercise 2.2.1.4

Old Problem – New Solution

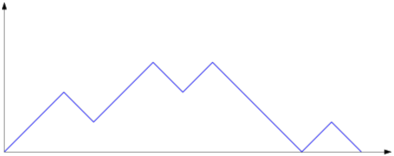
"We present here a new method for solving the ballot problem with the use of double generating functions, since this method lends itself to the solution of more difficult problems ..." – D. E. Knuth

Old Problem – New Solution

A rephrasing of the problem: Find the number of lattice paths with (1,1) and (1,-1) steps that never go below the x-axis and end on the x-axis.

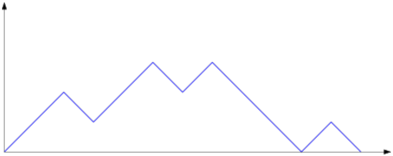
Old Problem – New Solution

A rephrasing of the problem: Find the number of lattice paths with (1,1) and (1,-1) steps that never go below the x-axis and end on the x-axis.



Old Problem – New Solution

A rephrasing of the problem: Find the number of lattice paths with (1,1) and (1,-1) steps that never go below the x-axis and end on the x-axis.



Ways to solve this:

- reflection principle
- first passage decomposition
- ...

How to do it with the kernel method

1. Enlarge the class of objects. Add catalytic/auxiliary variable.

How to do it with the kernel method

1. Enlarge the class of objects. Add catalytic/auxiliary variable.
2. Establish a functional equation. Rewrite it in kernel form.

How to do it with the kernel method

1. Enlarge the class of objects. Add catalytic/auxiliary variable.
2. Establish a functional equation. Rewrite it in kernel form.
3. Eliminate one of the unknowns.

How to do it with the kernel method

1. Enlarge the class of objects. Add catalytic/auxiliary variable.
2. Establish a functional equation. Rewrite it in kernel form.
3. Eliminate one of the unknowns.
4. Extract the generating function.

Steps 1 and 2: introduce new variable, functional equation

z ... length of the walk
s ... final altitude (this is our new variable!)

Steps 1 and 2: introduce new variable, functional equation

z ... length of the walk
s ... final altitude (this is our new variable!)

Use a step-by-step construction to obtain the functional equation

$$F(z,s) = 1 + z(s+\bar{s})F(z,s) - z\bar{s}F(z,0).$$

Steps 1 and 2: introduce new variable, functional equation

z ... length of the walk
s ... final altitude (this is our new variable!)

Use a step-by-step construction to obtain the functional equation

$$F(z,s) = 1 + z(s+\bar{s})F(z,s) - z\bar{s}F(z,0).$$

Rewrite in kernel form: "Bulk on the left, initial and boundary on the right"

$$\underbrace{(1 - z(s+\bar{s}))}_{\text{kernel}}F(z,s) = 1 - z\bar{s}F(z,0).$$

We are interested in $F(z,0)$ (walks that end on the x-axis).

Step 3: eliminate unknowns

Kernel equation:

$$(1 - z(s + \bar{s}))F(z, s) = 1 - z\bar{s}F(z, 0)$$

Step 3: eliminate unknowns

Kernel equation:

$$(1 - z(s + \bar{s}))F(z, s) = 1 - z\bar{s}F(z, 0)$$

Two unknowns: $F(z, s)$ and $F(z, 0)$.

Step 3: eliminate unknowns

Kernel equation:

$$(1 - z(s + \bar{s}))F(z, s) = 1 - z\bar{s}F(z, 0)$$

Two unknowns: $F(z, s)$ and $F(z, 0)$.

- LHS: contains s-dependent unknowns

Step 3: eliminate unknowns

Kernel equation:

$$(1 - z(s + \bar{s}))F(z, s) = 1 - z\bar{s}F(z, 0)$$

Two unknowns: $F(z, s)$ and $F(z, 0)$.

- LHS: contains s-dependent unknowns
- RHS: contains s-independent unknowns

Step 3: eliminate unknowns

Kernel equation:

$$(1 - z(s + \bar{s}))F(z, s) = 1 - z\bar{s}F(z, 0)$$

Two unknowns: $F(z, s)$ and $F(z, 0)$.

- LHS: contains s-dependent unknowns
- RHS: contains s-independent unknowns

There are two ways to get rid of one of the unknowns:

Step 3: eliminate unknowns

Kernel equation:

$$(1 - z(s + \bar{s}))F(z, s) = 1 - z\bar{s}F(z, 0)$$

Two unknowns: $F(z, s)$ and $F(z, 0)$.

- LHS: contains s-dependent unknowns
- RHS: contains s-independent unknowns

There are two ways to get rid of one of the unknowns:

- Eliminate the s-dependent unknown

Step 3: eliminate unknowns

Kernel equation:

$$(1 - z(s + \bar{s}))F(z, s) = 1 - z\bar{s}F(z, 0)$$

Two unknowns: $F(z, s)$ and $F(z, 0)$.

- LHS: contains s-dependent unknowns
- RHS: contains s-independent unknowns

There are two ways to get rid of one of the unknowns:

- Eliminate the s-dependent unknown
- Eliminate the s-independent unknown

Step 3: eliminate unknowns

Eliminate the s-dependent unknown $F(z,s)$.
Multiply the kernel equation by $(-s)$:

$$(zs^2 - s + z)F(z,s) = zF(z,0) - s.$$

We have that

$$zs^2 - s + z = z\left(s - \frac{1-\sqrt{1-4z^2}}{2z}\right)\left(s - \frac{1-\sqrt{1+4z^2}}{2z}\right).$$

Substitute

$$s = s_0(z) = \frac{1-\sqrt{1-4z^2}}{2z}$$

in the kernel equation and obtain

$$0 = zF(z,0) - s_0(z).$$

Step 4: extract generating function

Thus

$$F(z,0) = \frac{s_0(z)}{z} = \frac{1-\sqrt{1-4z^2}}{2z^2}.$$

Generating function for walks ending at height 0. Read off coefficients to obtain solution for n.

More generally

$$F(z,s) = \frac{s_0(z)-s}{zs^2-s+z} = \frac{1-\sqrt{1-4z^2}-2zs}{2z(zs^2-s+z)}.$$

Why not ...?

Why not

$$\tilde{s}_0(z) = \frac{1+\sqrt{1-4z^2}}{2z}?$$

Plugging this solution into the kernel equation gives

$$0 = zF(z,0) - \tilde{s}_0(z).$$

Thus

$$F(z,0) = \frac{\tilde{s}_0(z)}{z} = \frac{1+\sqrt{1-4z^2}}{2z^2} = \frac{1}{z^2} - 1 - z^2 - 2z^4 - \dots$$

Not a power series!

Small and large roots

Small roots: roots $s_i(z)$ which tend to zero as $z \to 0$.
Large roots: roots $s_i(z)$ which tend to infinity as $z \to 0$.

For the kernel method: use small roots.

Patterns: Prefixes and Suffixes

prefix of length k of a string/pattern . . . contiguous sub-string that matches the first k letters
Similarly: suffix . . . matches the last k letters.
Presuffix . . . is both prefix and suffix.

Patterns: Prefixes and Suffixes

prefix of length k of a string/pattern ... contiguous sub-string that matches the first k letters
Similarly: suffix ... matches the last k letters.
Presuffix ... is both prefix and suffix.

Example: Consider

$$p = [1, 3, 3, 1, -2, 3, 1]$$

- $[1, 3, 3]$ is a prefix of p (of length 3).
- $[-2, 3, 1]$ is a suffix p.

Patterns: Prefixes and Suffixes

prefix of length k of a string/pattern ... contiguous sub-string that matches the first k letters
Similarly: suffix ... matches the last k letters.
Presuffix ... is both prefix and suffix.

Example: Consider

$$p = [1, 3, 3, 1, -2, 3, 1]$$

- $[1, 3, 3]$ is a prefix of p (of length 3).
- $[-2, 3, 1]$ is a suffix p.
- $[1]$ is the only presuffix of p.

Finite automata

A *finite automation* is a quadruple $(\Sigma, \mathcal{M}, s_0, \delta)$ where

- Σ is the input alphabet
- $\mathcal{M}$ is a finite, nonempty set of states
- $s_0 \in \mathcal{M}$ is the initial state
- $\delta : \mathcal{M} \times \Sigma \to \mathcal{M}$ is the state transition function (or partial function, i.e., not every $\delta(S_i, x)$ is defined).

Sometimes: set $F \subseteq \mathcal{M}$ of final states also given.

Ways to describe an automation:

- as weighted graph (states are vertices, edge weights are sums of values of the transition function)
- as adjaceny matrix

Patterns and automata

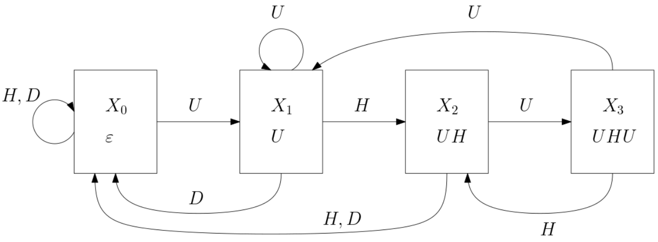
Example: $\mathcal{S} = \{U, H, D\}$ where $U = (1, 1), H = (1, 0)$ and $D = (1, -1)$, $p = [U, H, U, D]$ forbidden pattern.
Automation:

- States are proper prefixes of the pattern p
  Here: $X_0 = \varepsilon, X_1 = U, X_2 = UH, X_3 = UHU$
  In general: $X_i = [a_1, \ldots, a_i]$ first i letters of the pattern, $i = 0, \ldots, \ell(p) - 1$
- Transitions: $\delta(X_i, \lambda) = X_j$ if j is the maximal number such that X_j is a suffix of $X_i \lambda$

When the automaton reads a path w, it ends in the state labeled with the longest prefix of p that coincides with a suffix of w. The automaton is completely determined by the step set and the pattern.

Patterns and automata

Example: $\mathcal{S} = \{U, H, D\}$ where $U = (1,1)$, $H = (1,0)$ and $D = (1,-1)$, $p = [U, H, U, D]$ forbidden pattern.



Adjacency matrix and kernel

Adjacency matrix:

$$A = A(u) = \begin{pmatrix} 1+u^{-1} & u & 0 & 0 \\ u^{-1} & u & 1 & 0 \\ 1+u^{-1} & 0 & 0 & u \\ 0 & u & 1 & 0 \end{pmatrix}.$$

In each row except the last one, all entries sum up to the step polynomial $P(u)$. The *kernel* of an automaton is defined as

$$K(t,u) := \det(I - tA(u)).$$

Generating function for walks avoiding a pattern

Theorem

Let $\mathcal{S}$ be a simple set of steps and let p be a pattern with steps from $\mathcal{S}$. Then the bivariate generating function for walks avoiding the pattern p is given by

$$W(t,u) = \frac{(1,0,\ldots,0)\,\mathrm{adj}(I - tA)\vec{\mathbf{1}}}{K(t,u)}.$$

Generating function for walks avoiding a pattern

Proof. Step-by-step construction → obtain functional equation

$$(W_1, \ldots, W_\ell) = (1, 0, \ldots, 0) + t(W_1, \ldots, W_\ell)A$$

Rewrite as

$$\begin{aligned}(W_1, \ldots, W_\ell)(I - tA) &= (1, 0, \ldots, 0)\\ (W_1, \ldots, W_\ell) &= (1, 0, \ldots, 0)\frac{\mathrm{adj}(I - tA)}{\det(I - tA)}.\end{aligned}$$

$W(t, u)$ is the sum of all the GFs $W_\alpha(t, u)$ over all states. Thus

$$W(t, u) = \sum_{\alpha=1}^{\ell} W_\alpha = (W_1, \ldots, W_\ell)\vec{\mathbf{1}} = \frac{(1, 0, \ldots, 0)\,\mathrm{adj}(I - tA)\vec{\mathbf{1}}}{\det(I - tA)}.$$

Since $K(t, u)$ was defined as $\det(I - tA)$ we obtain

$$W(t, u) = \frac{(1, 0, \ldots, 0)\,\mathrm{adj}(I - tA)\vec{\mathbf{1}}}{K(t, u)}.$$

Generating function for meanders avoiding a pattern

Theorem

Let $\mathcal{S}$ be a simple set of steps and let p be a pattern with steps from $\mathcal{S}$. The bivariate generating function of meanders avoiding the pattern p is

$$M(t,u) = \frac{G(t,u)}{u^e K(t,u)} \prod_{i=1}^{e} (u - u_i(t)), \qquad (1)$$

where $u_1(t), \ldots, u_e(t)$ are the small roots of the kernel $K(t,u)$ and $G(t,u)$ is a polynomial in u which will be characterized in the proof.

1. Introduce catalytic variable (u) ... done
2. Functional equation + rewrite in kernel form:

$$(M_1, \dots, M_\ell) = (1, 0, \dots, 0) + t(M_1, \dots, M_\ell)A \\ - t\{u^{<0}\}((M_1, \dots, M_\ell)A).$$

Rewriting

$$(M_1, \dots, M_\ell)(I - tA) = \underbrace{(1, 0, \dots, 0) - t\{u^{<0}\}((M_1, \dots, M_\ell)A)}_{=:\vec{F}=(F_1,\dots,F_\ell)}. \quad (2)$$

The right hand side of 2 is a vector, its components are power series in t and Laurent polynomials in u (their lowest degree is the value of largest negative step).

Multiply (2) from the right by $(I - tA)^{-1} = \frac{(\mathrm{adj}(I-tA))\cdot\vec{\mathbf{1}}}{\det(I-tA)}$.
Furthermore, denote $\vec{\mathbf{v}} := \vec{\mathbf{v}}(t,u) = (\mathrm{adj}(I - tA)) \cdot \vec{\mathbf{1}}$. We obtain

$$M(t,u) = \frac{(F_1, \ldots, F_\ell)\vec{\mathbf{v}}}{K(t,u)}. \tag{3}$$

Write

$$\Phi(t,u) := u^e(F_1(t,u), \ldots, F_\ell(t,u)) \cdot \vec{\mathbf{v}} \tag{4}$$

where e is the number of small roots of $K(t,u)$ and multiply 3 with $u^e K(t,u)$ to get rid of the denominator and negative u-powers. We obtain

$$u^e K(t,u)M(t,u) = \Phi(t,u). \tag{5}$$

3. Eliminate one of the unknowns:
want to make LHS of $u^e K(t,u) M(t,u) = \Phi(t,u)$.vanish. This can be done by plugging in $u = u_i(t)$ where u_i is any small root of the kernel. Thus, the equation

$$\Phi(t,u) = 0$$

is satisfied by every small root of the kernel. Φ is a Laurent polynomial since F_i and $\vec{\mathbf{v}}$... Laurent polynomials by construction. Since $\Phi = u^e M(t,u) K(t,u)$ and M is a power series in u and $u^e K(t,u)$ is a polynomial in u, the function $\Phi(t,u)$ has no negative powers of $u \Rightarrow \Phi$ polynomial in u.
$u_i(t)$ root of the polynomial equation $\Phi(t,u) = 0 \Rightarrow$

$$\Phi(t,u) = G(t,u) \prod_{i=1}^{e} (u - u_i(t)) \tag{6}$$

for some $G(t,u)$ which is a power series in t and a polynomial in u (can be computed via comparing coefficients).

4. Extract generating function:
Substituting this into 3 we obtain

$$M(t, u) = \frac{G(t, u)}{u^e K(t, u)} \prod_{i=1}^{e} (u - u_i(t)).$$

$\square$

Bridges and excursions:

$$B(t) = W(t, 0)$$

$$E(t) = M(t, 0)$$

Extensions

Previously: several patterns studied individually (Deutsch (1998); Sun (2002); Sapounakis, Tasoulas, Tsikouras (2006); Mansour, Shattuck (2013), . . .) Asinowski, Bacher, Banderier, Gittenberger (2019): vectorial kernel method – unified approach that works for any pattern (simple step set, one pattern) Extensions

- Asinowski, Bacher, Banderier, Gittenberger (2019): Number of occurrences of a pattern can also be counted by VKM – introduce new variable that marks completion of the pattern
- Asinowski, Banderier, R. (2020): Avoidance of several patterns at once
- R. (2020): Avoidance of patterns in walks with longer steps
- other conditions that can be modeled by automata (height restrictions, non-contiguous patterns, . . .)

Thank you!