Oritatami systems assemble shapes no less complex than tile assembly model (aTAM)

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Abstract

Oritatami systems are a model of molecular co-transcriptional folding: the transcript (the "molecule") folds as it is synthesized according to a local energy optimisation process, in a similar way to how actual biomolecules such as RNA fold into complex shapes and functions while being synthesized (transcribed). We introduce a new model, called *turedo*, which is a self-avoiding Turing machine on the plane that evolves by marking visited positions and that can only move to unmarked positions, hence growing a self-avoiding path. Any oritatami can be seen as a particular turedo. We show that any turedo with lookup radius 1 can conversely be simulated by an oritatami, using a universal bead type set. Our notion of simulation is strong enough to preserve the geometrical and dynamical features of these models up to a constant spatio-temporal rescaling (as in intrinsic simulation). As a consequence, turedo can be used to build readily oritatami "smart robots", using our explicit simulation result as a compiler. Furthermore, as our gadgets are simple enough, this might open the way to a readable oritatami programming, and these ingredients could be regarded as a promising direction to implement computation in co-transcribed RNA nanostructures in wetlab.

As an application of our simulation result, we prove two new complexity results on the (infinite) limit configurations of oritatami systems (and radius-1 turedos), assembled from a finite seed configuration. First, we show that such limit configurations can embed any recursively enumerable set, and are thus exactly as complex as aTAM limit configurations. Second, we characterize the possible densities of occupied positions in such limit configurations: they are exactly the Π_2-computable numbers between 0 and 1. We also show that all such limit densities can be produced by one single oritatami system, just by changing the finite seed configuration.

None of these results is implied by previous constructions of oritatami embedding tag systems or 1D cellular automata, which produce only computable limit configurations with constrained density.

Note that, reframing our results, we prove that doodling without lifting the pen nor intersecting lines and using only a 1-local view to decide for the drawing directions produce drawings as complex and as dense as can be.

2012 ACM Subject Classification Author: Please fill in 1 or more \ccsdesc macro

Keywords and phrases Molecular Self-assembly, Co-transcriptional folding, Intrinsic simulation, Arithmetical hierarchy of real numbers, 2D Turing machines, Computability

Digital Object Identifier 10.4230/LIPIcs...

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Oritatami systems were introduced in [10, 11] to investigate the computational power of molecular co-transcriptional folding, in which an RNA sequence (transcript) folds upon itself into an intricate structure while being synthesized (transcribed). This phenomenon has proven programmable *in-vitro* [13], in which Geary, Rothemund, and Andersen demonstrated how to encode a rectangular tile-like structure in a transcript and its folding pathway so that this transcript folds cotranscriptionally along the pathway into the encoded structure. This *RNA Origami* architecture has recently been highly automated by their software ROAD (RNA Origami Automated Design) [9]. ROAD extends the scale and functional diversity of RNA scaffolds, and is thus a promising direction for the design of RNA-based computation. DNA tile self-assembly did rely on the cellular automata theory to build up the abstract Tile Assembly model (aTAM) [24] which in turn allowed to develop experimental settings simple enough to be implement in vitro, such as the Sierpinski triangle [22]. On the opposite, RNA origami was born first *in-vitro* and the oritatami system was created [12] to answer the lack of theoretical framework to design computations for cotranscriptional-based assembly systems. In this paper, we introduce the *turedo* model, implementable in oritatami, which, as opposed to oritatami, is simple enough to program, to wish for a design equivalent to the Sierpinski triangle experiment for cotranscription-based *in-vitro* systems.

An oritatami system consists of a "molecule" (the *transcript*) made of "beads" that attract each other. The molecule grows by one bead per step and, at each step, the δ most recently produced beads are free to move around to look for the position that maximizes the number of bonds they can make with each other (hence the folding is co-transcriptional). This process ends up self-assembling a shape incrementally. It is known from [12, 21] that oritatami systems are Turing universal. They can also build arbitrary shapes [5] modulo a small universal constant upscaling, or specific fractals [18]. However, oritatami systems remain notably challenging to design. Indeed, the only shapes that can be built by [12, 21] are space-time diagrams of cyclic tag-systems or 1D cellular automata; and [5] requires to hardcode the whole shape in the transcript. In this article, we introduce a new computational model (*turedo*) that abstracts away the technical details of attraction rules and bead sequence of oritatami, but embraces the geometrical aspects of them, as opposed to the classical one-dimensional computational models. We demonstrate that turedos can be simulated up to upscaling by oritatami systems. Our simulation allows thus to take full advantage of turedo computations in building shapes, and can be used as a compiler to design powerful oritatami systems as demonstrated below.

Oritatami systems and Turedos. The classical model of Turing machines has already been considered in other settings than the one dimensional bi-infinite tape, in particular in higher dimensions [2]. A popular class of Turing machines in $\mathbb{Z}^2$ is that of turmites [17], which are free to move on the plane but do it by just looking at their current internal state and the tape content at their current position. In this paper we introduce a somewhat orthogonal class of Turing machines on the plane, that we call *turedos*[1], which can look at the tape content around their position to decide their move (like in [2]), but are constrained to move only in a self-avoiding way.

Both our models (oritatami and turedos) have two strong constraints: they are sequential

[1] Inspired by the nicely coined terminology for turmites, as a reference to *toredo navalis* (shipworms) that would only grow self-avoiding tunnels in wood if they were infinite.

and self-avoiding (*i.e.* each position of the plane can only be visited once and becomes an obstruction for future moves). They can be seen as the sequential counterpart of aTAM model of self-assembly [20, 6] or freezing cellular automata [14, 3, 19]. But they are not just finite state automata growing a self-avoiding path in a regular way, their computational power is in their ability to make moves depending on the configuration of neighboring positions.

Our main result is that oritatami can simulate turedos of lookup radius 1. Our notion of simulation is strong enough to preserve the geometrical and dynamical features of these models up to a constant spatio-temporal rescaling: the oritatami reproduces the whole dynamics of the turedo using macro-cells and a constant spatio-temporal rescaling. This definition is similar to intrinsic simulations developed for cellular automata [4] or self-assembly tilings [6].

▶ **Theorem 1.1** (Main result 1). *There is a universal bead type set $\mathcal{B}$ such that for any turedo $\mathcal{T}$ of radius 1 with alphabet of size Q, there is a delay-3 oritatami system based on $\mathcal{B}$ with period $\Lambda = \Theta(Q^6 \log Q)$ which simulates intrinsically $\mathcal{T}$ at space-scale $\Theta(Q^3\sqrt{\log Q})$ and time-scale Λ.*

Theorem 1.1 is proved in section 3.

Complexity of limit configurations. The Turing universality results in [12, 21] induce undecidability results of the form: given an oritatami, a seed and a position, determining whether the position will be visited is undecidable. However these embeddings are such that the limit configurations obtained are always computable, because the space-time of the simulated tag system (or cellular automaton) computation is progressively constructed in a predictable way in a fixed region of oritatami's space. Precisely, in any limit configuration c^∞ obtained this way, the map $z \mapsto c^\infty(z)$ is computable because there is a computable time bound $\tau(z)$ such that if position z is not visited after $\tau(z)$ steps of the run, then it will never be visited (see Lemma 4.1).

The first application of our simulation result is to prove that we can produce uncomputable limit configurations from finite seeds with oritatami (section 4). This implies that there are oritatami runs from finite seeds where there is no computable time bound $\tau(z)$ on the visit time of position z. Results on uncomputable limit configurations were already obtained in the model of directed aTAM [16]. However the construction used takes full advantage of the massive parallelism allowed in the aTAM model and *cannot* be translated to the turedo settings. Our construction is actually simpler than that of [16] and shows that sequential self-avoiding models can organize information in the plane in such a way that some regions allow 'uncomputable come backs.'

▶ **Theorem 1.2** (Main result 2). *There exists a fixed oritatami with delay 3 and a fixed finite seed σ such that the limit configuration c^∞_σ produced is uncomputable as a map.*

The second application of our simulation result is about (upper) density of occupied positions in the limit configurations obtained from finite seeds. Density is a natural geometrical parameter to test the ability of our models to produce complex infinite self-avoiding paths from finite seeds. We show that such densities are exactly the Π_2-computable numbers between 0 and 1 (Theorem 5.3), where Π_2-computable means being the limsup of a computable sequence of rational numbers [25]. In particular turedos and oritatami can produce limit densities which are not recursively approximable (i.e. not the limit of any computable sequence of rational numbers). We actually show that the whole spectrum of density can be obtained in a single turedo by varying the seed (Theorem 5.3). Using our simulation framework, the following result is shown for oritatamis in section 5.

▶ **Theorem 1.3** (Main result 3). *For any $\epsilon > 0$, there exists an oritatami of delay 3 such that for any Π_2-computable number $d \in [0, 1-\epsilon]$, there is a finite seed σ such that the limit configuration c_σ^∞ reached from it has density of occupied positions exactly d.*

Note that the densities that can be produced in the (directed) aTAM model or freezing cellular automata from finite initial configurations cannot be more complex (see Lemma 5.1).

The organization of the paper is as follows: we first present oritatami and turedo models and the notion of simulation (section 2); then, we establish our main simulation result (section 3) and its two applications (sections 4 and 5).

2 Definitions and Models

Oritatami systems. Let B be a finite set of *bead types*. A *configuration* c of a bead type sequence $p \in B^* \cup B^{\mathbb{N}}$ is a directed self-avoiding path $c_0 c_1 c_2 \cdots$ in the triangular lattice $\mathbb{T}$,[2] where for all integer i, the vertex c_i of c is labeled by p_i and refers to the *position* in $\mathbb{T}$ of the $(i+1)$-th bead in the configuration. A *partial configuration* of p is a configuration of a prefix of p.

For any partial configuration c of some sequence p, an *elongation* of c by k beads (or *k-elongation*) is a partial configuration of p of length $|c| + k$ extending by k positions the self-avoiding path of c. We denote by $\mathcal{C}_p$ the set of all partial configurations of p (the index p will be omitted whenever it is clear from the context). We denote by $c^{\triangleright k}$ the set of all k-elongations of a partial configuration c of sequence p.

An *oritatami system* $\mathcal{O} = (p, \heartsuit, \delta)$ is composed of (1) a (possibly infinite) bead type sequence p, called the *transcript*, (2) an *attraction rule*, which is a symmetric relation $\heartsuit \subseteq B^2$, and (3) a parameter δ called the *delay*. $\mathcal{O}$ is said *periodic* if p is infinite and periodic. Periodicity ensures that the "program" p embedded in the oritatami system is finite (does not hardcode unbounded behavior) and at the same time allows arbitrarily long computation.[3]

We say that two bead types a and b *attract* each other when $a \heartsuit b$. Furthermore, given a (partial) configuration c of a bead type sequence q, we say that there is a *bond* between two adjacent positions c_i and c_j of c in $\mathbb{T}$ if $q_i \heartsuit q_j$ and $|i-j| > 1$. The *number of bonds* of configuration c of q is denoted by $H(c) = |\{(i,j) : c_i \sim c_j,\ j > i+1,\ \text{and } q_i \heartsuit q_j\}|$.

Oritatami dynamics. The folding of an oritatami system is controlled by the delay δ. Informally, the configuration grows from a *seed configuration* (the input), one bead at a time. This new bead adopts the position(s) that maximize(s) the potential number of bonds the configuration can make when elongated by δ beads in total. This dynamics is *oblivious* as it keeps no memory of the previously preferred positions [12].

Formally, given an Oritatami system $\mathcal{O} = (p, \heartsuit, \delta)$ and a *seed configuration* σ of a *seed bead type sequence* s, we denote by $\mathcal{C}_{\sigma,p}$ the set of all partial configurations of the sequence $s \cdot p$ elongating the seed configuration σ. The considered *dynamics* $\mathscr{D} : 2^{\mathcal{C}_{\sigma,p}} \to 2^{\mathcal{C}_{\sigma,p}}$ maps every subset S of partial configurations of length ℓ elongating σ of the sequence $s \cdot p$ to the

[2] The triangular lattice is defined as $\mathbb{T} = (\mathbb{Z}^2, \sim)$, where $(x,y) \sim (u,v)$ if and only if $(u,v) \in \cup_{\epsilon=\pm 1}\{(x+\epsilon, y), (x, y+\epsilon), (x+\epsilon, y+\epsilon)\}$. Every position (x,y) in $\mathbb{T}$ is mapped in the euclidean plane to $x \cdot \vec{e} + y \cdot \vec{sw}$ using the vector basis $\vec{e} = (1,0)$ and $\vec{sw} = \text{RotateClockwise}(\vec{e}, 120^\circ) = (-\frac{1}{2}, -\frac{\sqrt{3}}{2})$. We will denote by $\vec{nw}, \vec{ne}, \vec{e}, \vec{se}, \vec{w}, \vec{sw}$ the six canonical unit vectors in $\mathbb{T}$.

[3] Note that we do not impose here a maximal number of bonds per bead (called arity).

subset $\mathscr{D}(S)$ of partial configurations of length $\ell + 1$ of $s \cdot p$ as follows:

$$\mathscr{D}(S) = \bigcup_{c \in S} \arg\max_{\gamma \in c^{\triangleright 1}} \left(\max_{\eta \in \gamma^{\triangleright(\delta-1)}} H(\eta) \right)$$

The possible configurations at time t of the oritatami system $\mathcal{O}$ are the elongations of the seed configuration σ by t beads in the set $\mathscr{D}^t(\{\sigma\})$.

We say that the Oritatami system is *deterministic* if at all time t, $\mathscr{D}^t(\{\sigma\})$ is either a singleton or the empty set. In this case, we denote by c^t the configuration at time t, such that: $c^0 = \sigma$ and $\mathscr{D}^t(\{\sigma\}) = \{c^t\}$ for all $t > 0$; we say that the partial configuration c^t *folds (co-transcriptionally) into* the partial configuration c^{t+1} deterministically. In this case, at time t, the $(t+1)$-th bead of p is placed at c^{t+1} at the position that maximises the number of bonds that can be made in a δ-elongation of c^t.

Turedos: Self-avoiding Turing Machines. A *turedo* is a Turing machine working on the plane with a lookup neighborhood (like in [2]), that can only move in a self-avoiding way. We fix the following set of elementary hexagonal[4] moves $N_H = \{\vec{\mathrm{N}} = (1,1), \vec{\mathrm{NE}} = (1,0), \vec{\mathrm{SE}} = (0,-1), \vec{\mathrm{S}} = (-1,-1), \vec{\mathrm{SW}} = (-1,0), \vec{\mathrm{NW}} = (0,1)\}$ in $\mathbb{Z}^2$ and denote by $B(r)$ the hexagonal ball of radius r centered on $(0,0)$, *i.e.* the set of positions in $\mathbb{Z}^2$ that can be written as a sum of at most r vectors from N_H. We also denote by $b(r)$ the size of $B(r)$, and $c_z(r) = (u \in B(r) \mapsto c(z+u))$ the restriction of a configuration c to the ball of radius r centered on z. Finally, we fix a universal blank symbol $\perp$ representing unoccupied positions.

▶ **Definition 2.1.** *A* turedo *is defined by* $\mathcal{T} = (A, Q, q_0, r, \delta)$ *where* A *is the tape alphabet,* $\perp \in A$*,* Q *is the set of head states with initial state* $q_0 \in Q$*,* r *is the lookup radius,* $\delta : Q \times A^{B(r)} \to Q \times N_H \times A \setminus \{\perp\}$ *is the local transition map.*

A tape configuration *is an element of* $A^{\mathbb{Z}^2}$*. A* global state *is an element of* $\mathcal{S}_\mathcal{T} = A^{\mathbb{Z}^2} \times \mathbb{Z}^2 \times Q$ *(tape configuration, position of head and head state). The turedo* $\mathcal{T}$ *induces a global map* $F_\mathcal{T} : \mathcal{S}_\mathcal{T} \to \mathcal{S}_\mathcal{T}$ *defined as follows:*

$$F_\mathcal{T}(c,z,q) = \begin{cases} (c, z, q) & \text{if } c(z) \neq \perp \text{ or } c(z+d) \neq \perp, \\ (c', z+d, q') & \text{else,} \end{cases}$$

where $(q', d, a) = \delta(q, c_z(r))$ *and* $c'(z) = a$ *and* $c'(u) = c(u)$ *for* $u \neq z$*. When the first case occurs, we say that the machine is blocked.*

The key point of the above definition (which justifies the qualification of 'self-avoiding') is that the only way tape configurations can be altered is by turning a blank symbol into a non-blank symbol, and therefore the head cannot go back to a previously visited position (except when the machine is blocked in which case the global state is a fixed point). Positions holding a blank symbol are therefore seen as empty positions where the head can possibly move to.

[4] The triangular lattice for oritatami uses orientation east-west while the set of elementary moves N_H in turedo contains north-south. It is of course harmless since oritatami are invariant by rotation and could be defined with another triangular lattice. This choice is justified by the main simulation result of the paper where macrocells in oritatami in our figures appear in the same orientation as the hexagonal cells in turedos.

Limit configuration and freezing time. Given an initial global state $s \in \mathcal{S}_\mathcal{T}$ for a turedo of global map $F_\mathcal{T}$, let us consider the sequence $(c^n, z_n, q_n) = F_\mathcal{T}^n(s)$ for $n \in \mathbb{N}$. By the self-avoiding property, it holds that for any $z \in \mathbb{Z}^2$ the sequence of symbols $(c^n(z))_{n\in\mathbb{N}}$ is ultimately constant, and, denoting its limit $c_s^\infty(z)$, we then have defined a tape configuration $c_s^\infty \in A^{\mathbb{Z}^2}$ which is called the *limit configuration* reached by F starting from s. Said differently, using the standard Cantor topology for tape configurations [15], we have that the sequence of configurations $(c^n)_n$ converges to c_s^∞. Moreover, we can associate to the system and the initial global state s, the *freezing time* map $\tau_s : \mathbb{Z}^2 \to \mathbb{N}$ such that $\tau_s(z)$ is the minimal t for which the tape content of cell z at time t is $c_s^\infty(z)$.

Programming turedos. Thanks to the freedom allowed in their local maps, turedos are in general much easier to design than oritatami systems. The basic building block to design complex turedos is the zigzag movement which allows to embed any 1D Turing machine/cellular automaton computation. They can also be used as thick wires to transport information from one region to another. Our zigzag toolbox is detailed in appendix.

Simulations. Any oritatami with delay δ can be seen as a particular turedo of radius $\delta + 1$: indeed, an oritatami transition is completely determined by the position in the sequence of beads, coded as a state of the turedo, and the local configuration in a ball of radius $\delta + 1$.

Our main result proven in the next section is a converse to this observation: any turedo of radius 1 can be simulated by an oritatami system of delay 3. The general idea is to reproduce the dynamics up to a linear spatio-temporal scale factor like in similar notions already considered for cellular automata or self-assembly tilings [4, 7, 3]. More precisely, each cell of the simulated system is represented by a macro-cell in the simulator system, the macro-cells form a linearly distorted hexagonal lattice, and a constant number of time steps is allowed for the simulator to reproduce one step of the simulated system. This notion of simulation is very strict and allows to relate properties of the limit configurations in the simulated system to the corresponding limit configuration in the simulator. This can be done without further hypothesis for computability of limit configurations, but can also be done for the density of non-blank states as soon as the simulation uses macro-cells that are filled densely and constantly.

The complete formalization of the notion of simulation used is given in appendix together with lemmas on computability and density of limit configurations.

3 Delay-3 oritatami systems simulate radius-1 Turedos

This section provides an overview of the design implying main Theorem 1.1. As for the 1D cellular automaton simulation in [21], our simulation proceeds in three phases: 1) reading the neighboring letters, 2) preparing for writing the new letter on the boundaries of the macro-cell and 3) exiting to the computed next location. However, we must solve a significant number of new challenges to adapt to turedos. Turedos are free to move in every direction: the shape of the macrocells must then be isotropic. Furthermore, the reading process must be non-blocking. Thus we cannot use the reading mechanism in [21], nor the writing flip-flap mechanism which would block any further return to a previously visited border; we cannot use its hardcoded exit mechanism, as the exit direction has to deduced from the symbols read. Moreover, as we need to return to a random side after reading and writing on all sides, our oritatami system must be able to absorb up to 4 times the side length before exiting to the new macrocell and starting the next period of the transcript. It follows that we cannot

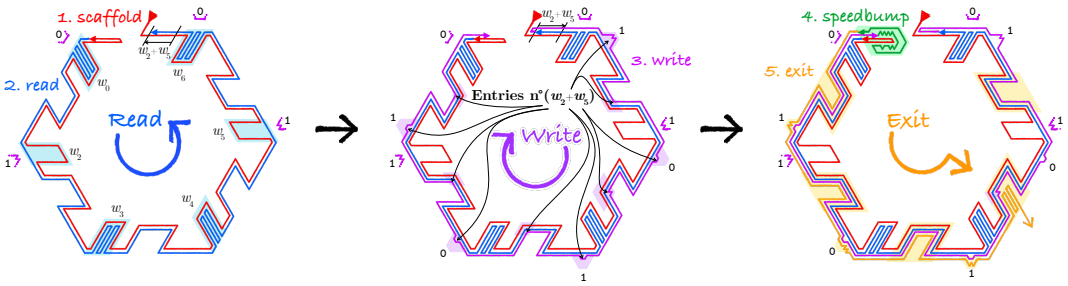


Figure 1 Principle of the macrocell operation. The shift of the reading layer at the end of its folding (and thus of the writing layer) is $\sum_{i:\text{bit read}_i=1} w_i = w_2 + w_5$.

store information on the boundary of the macro-cell as in [21], but need to store information *inside* the macrocell to avoid increasing the macrocell side length uncontrollably. Similarly the speedbump module introduce in [21] must be adapted to fit inside a compact space.

To solve all those issues, we have developed new tools that we believe to be simple, powerful and generic enough to have their own interest. We also believe that some of them could serve as a guideline for a first biochemical implementation of computation using RNA co-transcription. We have a fully functional implementation of our system which can be freely downloaded from [1] to be run on the oritatami simulator [23].

Bit-weight encoding of a Turedo. Consider a radius-1 turedo. First, we get rid of its internal state and orientation by encoding them in the symbols of the tape configuration. We then encode each symbol of the resulting tape alphabet $\mathcal{A}$ as a string of q bits where $q = \lceil \log_2 \#\mathcal{A} \rceil$. The blank symbol $\perp$ is encoded by the reserved word 0^q. Let $Q = 2^q$. In the following we assume that the neighboring cells of the current position are numbered in counterclockwise (CCW) order from 0 to 5 where 5 denotes the cell previously visited by the turedo. Our simulation assumes that the turedo transition function is a function $F : (2^q)^6 \to 2^q \times \{0, \ldots, 4\}$, that reads the q bits $b_{i,0}, \ldots, b_{i,q-1}$ encoding the symbol in the ith CCW neighboring cell for $i = 0..5$, and outputs the q bits of the symbol to be written and the CCW index of the next cell to go to.[5] Furthermore, we assume that F is encoded as a tuple $((w_{ij}), \Phi)$ such that $F((b_{ij})) = \Phi(\sum_{i,j} w_{ij} b_{ij})$ where the $6q$ bit-weights (w_{ij}) are non-negative integers. All transition function F can be encoded this way using the weights $w_{ij} = 2^{qi+j}$. We denote by $\mathcal{W} = \sum_{i,j} w_{ij}$ the sum of the weights of the bits. Encoded this way, the size of the transition table of F is exactly $\mathcal{W} + 1$ for every bit and exit direction.

Principle of the macrocell operation. Fig. 1 presents a schematic overview of the key operations in the macrocell. The transcript consists in five parts:

1. the *scaffold* of the macrocell folds, on each side of the macrocell, in front of the position of each bit to be read, "read pockets" (in blue) of size egal to the weight given by the transition function to that bit; it also builds one "exit pocket" (in orange) per side.
2. the *read layer* folds counterclockwise and fills the read pockets (outlined in blue) when it senses a `0`, and jumps over it when it senses a `1`, pushing the transcript forward by a shift corresponding to the sum Δ of the sizes of the pockets sensing a `1` ($\Delta = w_2 + w_5$ in the figure);

[5] The case of attempting to exit towards the CCW neighboring cell n°5 from which the turedo came, is purposely ignored as it would unnecessarily complicate the construction.

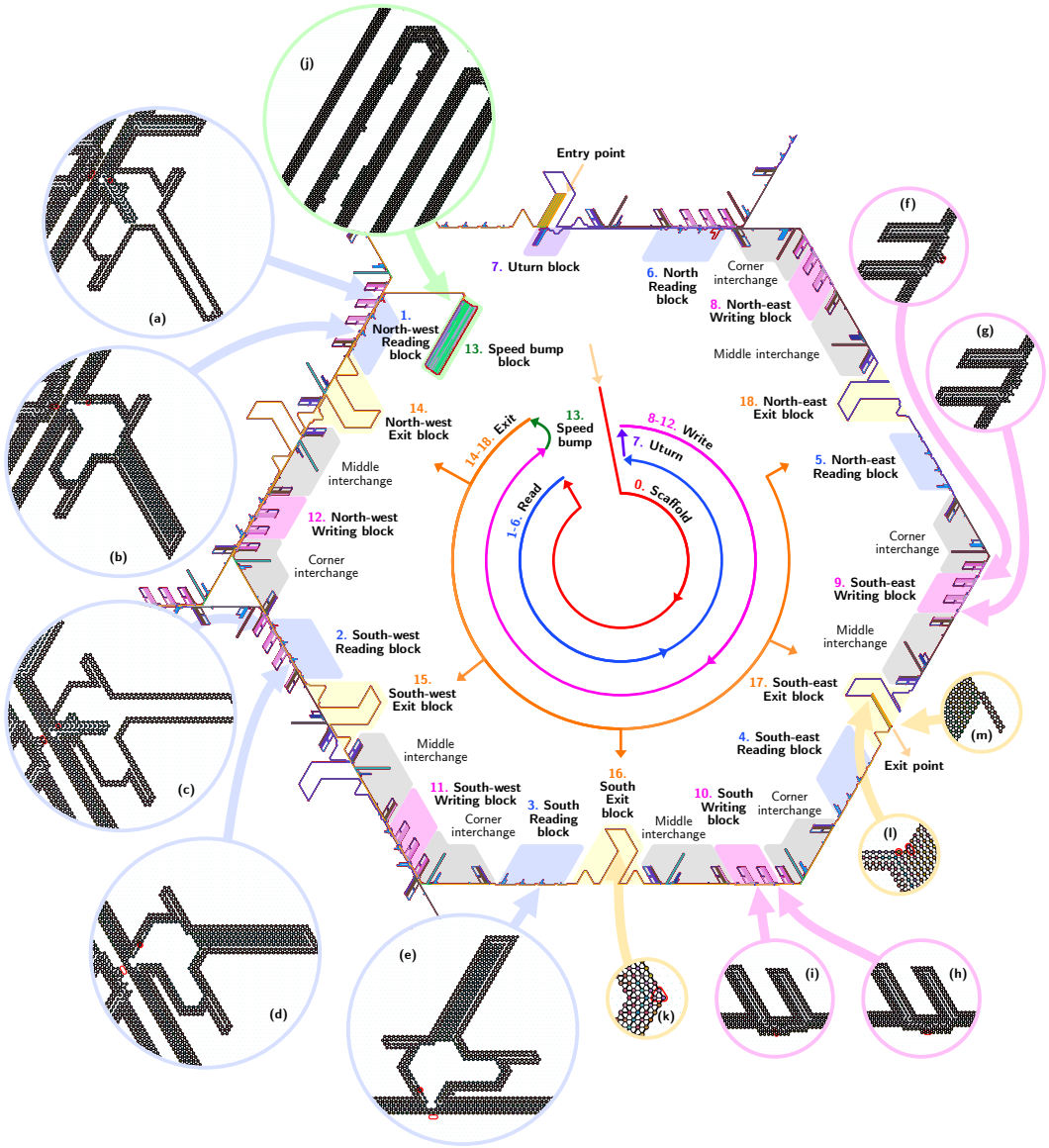


Figure 2 A macrocell for a turedo with $q = 3$ bits ($Q = 8$ tape symbols) together with the order in which layers and modules are used along its boundary as well as snapshot of important modules: (a)-(e) the read pocket in all possible situations: reading a 0/⊥(b,d,e) or a 1 (a,c) from a neighboring cell (a–d) (or not (e)) and through its exit layer (a,b) or directly from its write layer (c,d) – (f)-(h) all possible situations for the write module: writing a 0 (g,i) or a 1 (f,h), through the exit layer (h,i) or directly (f,g) – (j) the shift-absorbing speedbump – (k) the exit layer folds along the exit pocket – (l)-(m) the write layer has placed a kicking ↻76 bead in the corner that detaches the exit layer from the pocket and concludes the folding by exiting to the SW. Zoom in for details

3. the *write layer* contains all the transition tables of the simulated turedo, one for each bit to write on each side, and one for each exit-or-not decision on each side; this layer folds clockwise, and as it is translated forward by Δ, it folds the Δth entry of each transition table at the writing spots (in purple) that trigger the foldings of the transition table entries. The shift Δ accumulated by the reading layer allows then to write the place output pattern on each side. It also places a "kicking bead" (in purple) in the exit pocket on the computed exit side and no-kicking beads in the other using the same shift-principle;
4. the *speedbump module* (outlined in green) absorbs the shift so that the next layer starts without any shift regardless of the values read by the read layer;
5. the *exit layer* folds counterclockwise, following the border until it hits the kick (outlined in yellow) and folds upon itself to the next macrocell.

Observe that the reading layer needs to "read" the bit from neighboring cells and still make room for the two next layers to fold between its layer and the neighboring cells. This explains why our oritatami systems has delay 3: it has to read through 3 layers.

This presentation was just an overview of the macrocell. The complete description of the macrocell is given in Fig. 2. An actual execution of 20 steps of the simulation of a turedo is illustrated in Fig. 7 in appendix. We will now present some of the key tools used in our design.

Folding meter and pockets. Our construction relies on two new simple and powerful tools:

- a *folding meter* is a $4n$-periodic transcript whose period has 4 equally spaced articulation points, so that it can either: 1) follow a border if it is strongly attracted to it; 2) fold upon itself in a compact zig-zag form if the attraction to the border is weak; 3) reveal an hardcoded structure if the attraction to its surrounding is mild.
- a *pocket* is a box which triggers the compact folding of a folding meter and which allows to hide a portion of it in a compact space. The entrance to such a pocket can be conditionned by the surrounding. For instance, the read folding meter enters a read pocket if and only if its reading head rq88 or rq36 is not attracted by the beads encoding a 1 in its neighborhood (see Fig. 2(a–e)), otherwise it folds into an hardcoded glider and exit the pocket rightaway.

Furthermore, several folding meters can be layered on top of each other in opposite direction as long as their periods match. Synchronizing and desynchronizing the two layers allow to trigger the various behaviors as well, by varying the strength of their bonds. For instance, the write layer folds into spikes encoding 0 or 1 when it passes over the read layer in Fig. 2(f–i) because its bonds are weaker with the read layer when the latter is desynchronized after having been suck into the pitfalls that surround this area. Folding meters are presented in details in appendix J.2

The read and write blocks. Fig. 3 shows in details the actual oritatami implementation of the read and write blocks and how write pockets of size equal to the size of the transition tables are used as interconnected vessels to place the correct entry of the table over the write module.

Layer interchange. Each layer is heavily interacting with its neighboring layers inside a macrocell. It follows that unwanted interferences may occur between layers of neighboring macrocells. For this purpose, we use three different variants of bead types in each layer: one for each half of each side, plus one in the middle (see Appendix J.16 and J.17).

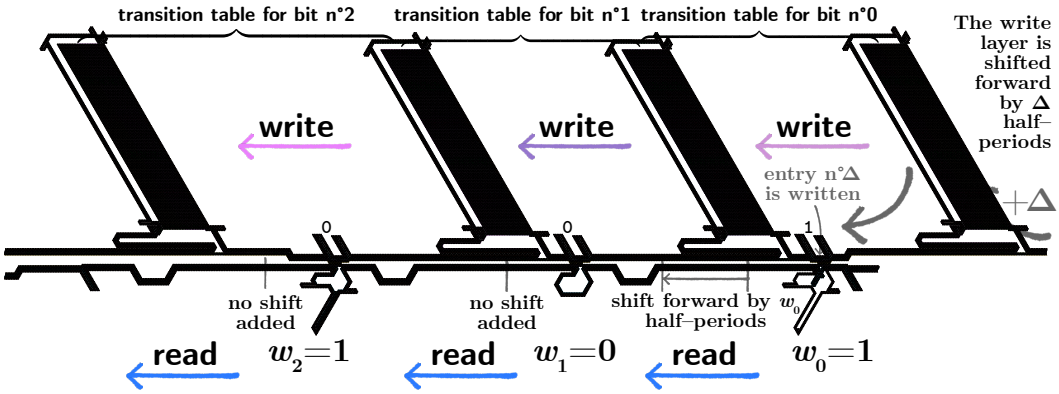


Figure 3 The read and write blocks: (bottom) when the read layer folds, it fills every read pocket facing a 0 on the neighboring macrocell, but skips the read pocket if it faces a 1, which increases the shift forward of the read layer by the weight of the corresponding bit; this yields a total additional shift of w_0 half-periods in the figure – (top) the transition tables are stored in the write layer transcript: one per half-period; the size of the write pockets is set to $\mathcal{W}$ half-periods to accomodate all the unused entries in the transition tables and the transition table is located in the write layer so that its first entry is aligned with the write module when the shift is 0; when the write layer starts to fold, it is shifted forward by $\Delta = \sum_{\text{bit read}(i,j)=1} w_{ij}$ half-periods, the total shift accumulated by its preceding read layer (each transition table is highlighted in a different color in the figure); this implies that the part of the write layer folding over each write module (highlighted in blue) is the one encoding the Δ-th entry of the transition table for each bit to write on the macrocell side as expected; this part will fold into a prescribed shape which will be read as 0 or a 1 by the read layer of its upcoming neighboring macrocell. This is an actual oritatami simulation. Zoom in for details

Setting up the exit module. As the exit pocket needs to accommodate the remaining of the exit layer before it exits, it must have room to fold in a compact shape a folding meter of length up to four macrocell-side long. As the exit pocket belongs to the macrocell side, we need to solve a fix point problem. Moreover, as a different amount of the exit layer will fold into each exit pocket, we need a mechanism to make sure that in all cases, the transcript will exit at the same position on the macrocell side, without interfering with the fix point resolution above. The latter problem is solved by using a pair of "loose ropes" of equal length, one on each side, "pulling" on the exit pocket to adapt its position to the macrocell side. These two important points are detailed in Appendices J.15 and J.18. This concludes the overview of the proof of Theorem 1.1.

4 Uncomputable Limit Configurations and Freezing Time

A configuration $c \in A^{\mathbb{Z}^2}$ is *computable* if there is a Turing machine which on input $z \in \mathbb{Z}^2$ computes $c(z)$. We are interested in the computability of limit configurations obtained from finite initial configurations (*i.e.* everywhere $\perp$ except on a finite region).

As said in the introduction, constructions of Turing universal oritatami systems known so far [21, 12] *do not* produce uncomputable limit configurations. The key reason is that they have a computable *escape direction*: a direction $u \in \mathbb{Z}^2$ and a computable non-decreasing function μ such that $\mu(t) \to \infty$ and for any $t \in \mathbb{N}$, the position z_t of the head after t steps verifies $u \cdot z_t \geqslant \mu(t)$ where '$\cdot$' denotes the scalar product (i.e. the head globally moves away along the direction u). Such a computable escape direction appears naturally in these simulations because they are fundamentally simulations of space-time of one-dimensional systems: they work by growing successive 1D finite configurations and stacking them along a

direction u that corresponds to the time of the simulated system. The simulation never goes back to previously stacked layers simply because computing one step of the 1D system is performed using the last stacked 1D configuration only. More generally (proof in appendix):

▶ **Fact 4.1.** *For any turedo reaching limit configuration c_s^∞ from a finite global state s, the maps $z \mapsto c_s^\infty(z)$ and $z \mapsto \tau_s(z)$ are Turing-equivalent. Moreover they are both computable if the dynamics admits a computable escape direction.*

In the next result, we construct a turedo that goes back uncomputably close to the origin uncomputably often in spite of following a self-avoiding trajectory. Precisely, we prove that turedos of radius 1 and therefore oritatami are powerful enough to embed any recursively enumerable set into their limit configurations reached from a finite initial configuration. As a consequence, both models produce uncomputable limit configurations.

▶ **Theorem 4.2.** *There exists a fixed turedo of radius 1 which, when started from a fixed global state s with a blank tape configuration, reaches an uncomputable limit configuration and therefore has an uncomputable freezing time map τ_s.*

Proof sketch. The basic idea, illustrated in Fig. 4a, is to build a turedo which runs a Turing machine simulation to test all Turing machines for halt in parallel and that, when it finds that some machine i has halted, interrupts momentarily its computation and goes to write a flag in a prefabricated area $p(i)$ located at a computable position in i (initially all areas $p(i)$ are empty). Areas of type $p(i)$ are progressively filled in some uncomputable and unknown order, but, at the limit, it holds that $p(i)$ contains a flag if and only if the machine i halts. Therefore the limit configuration is uncomputable because it can solve the halting problem when used as an oracle.

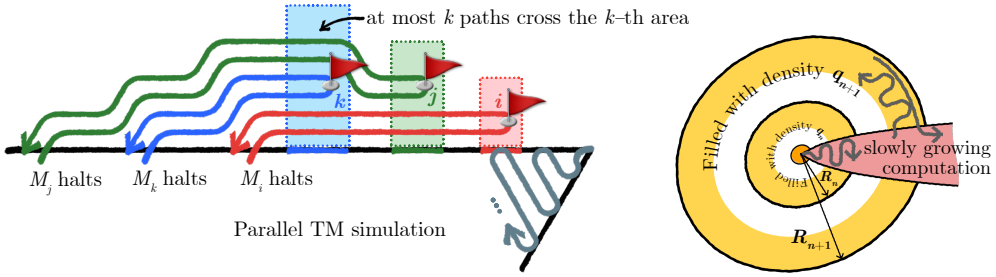
The key to implementing this idea is the layout of the paths to reach the areas $p(i)$: when we proceed as shown in Fig. 4a, no more than i paths will go across the area $p(i)$, i.e. the ones that correspond to the halting Turing machines with $j < i$. As a zigzag of thickness $O(j)$ is enough for the turedo to reach area j, place a flag and go back, then the flag in area $p(i)$ (if any) will never be placed higher than $O(i^2)$ (see appendix). It follows that these area have quadratic size and their ground basis can be set up in advance by the turedo as it simulates the Turing machines in parallel (in particular, the turedo will start the simulation of machine i only after the ground basis of area $p(i)$ is set up). Of course, Figure 4a is a simplification and does not represent all movements of the turedo's head: in particular, when moving towards area $p(i)$, the turedo needs to carry on the information i and to bubble up the ground basis of each area crossed over along the way, and it cannot carry those in its state set. All the implementation details are given in appendix. Using our simulation framework (and in particular Lemma C.2 in appendix), main Theorem 1.2 follows directly from Theorems 1.1 and 4.2.

5 Characterization of Possible Densities of Limit Configurations

We can define the (upper) density $\overline{d}(c)$ of non-blank cells in configuration c as follows:

$$\overline{d}(c) = \limsup_n \frac{\#\{z \in B(n) : c(z) \neq \bot\}}{b(n)}.$$

This choice is natural and gives a translation-invariant notion, but it is not unique (we could replace the sequence $(B(n))_n$ by another Følner sequence [8]). The problem is that, in a simulation, the lattice of cells is distorted into a macro-lattice of macro-cells in such a way



(a) Sketch of the turedo building an uncomputable limit configuration.

(b) Sketch of the turedo building a limit configuration with an arbitrary density $d \in \Pi_2$.

Figure 4 Sketch of the two turedo constructions in sections 4 and 5.

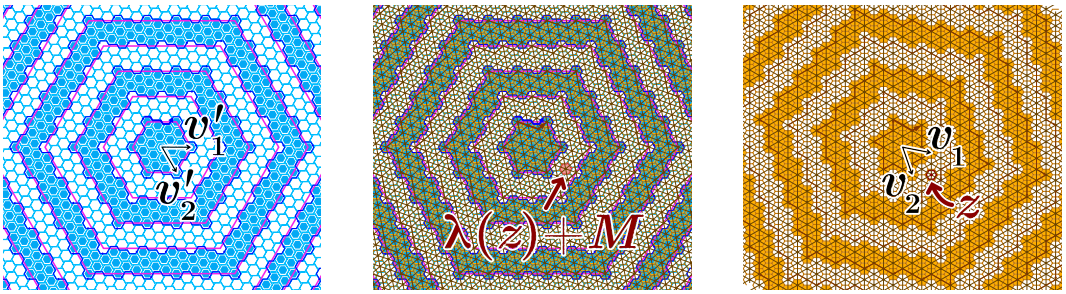


Figure 5 The linear map λ between the turedo world and the oritatami world induces a tilt between the concentric balls in the both worlds. This simulation tilt must be compensated by providing to the simulated turedo a pair of vectors as an input, so that it fills a proper discretization of the oritatami world balls when simulated: (left) the shortest radius vectors v'_1, v'_2 of a ball in the oritatami world that can be mapped exactly in the turedo world – (right) when the two corresponding vectors v_1, v_2 in the turedo world are supplied to the turedo as an input, the turedo can use them to build a proper discretization of large enough balls in the oritatami world – (middle) both turedo and oritatami worlds superposed (the target balls are drawn in purple and the discretized turedo ones in blue).

that the macro-balls do not have the same shape as genuine balls, as shown in Fig. 5. Said differently, the reference Følner sequence is distorted into another one and this can change the density. To circumvent this problem and produce more robust results, we will consider all possible linearly distorted balls from the start: for any pair $v_1, v_2 \in \mathbb{Z}^2$ of non-colinear vectors, we consider the (upper) density $\overline{d}_{v_1,v_2}$ of non-blank state after distortion of the lattice by the pair v_1 and v_2 (formal definition in appendix).

We first prove that the computational complexity of $\overline{d}_{v_1,v_2}(c)$ is Π_2-bounded as soon as c is produced as the limit of a computable process on finite configurations such that the set of non-blank positions is monotonically increasing and with diameter growing in a computable way. This bound applies to turedos but also all systems cited in section 1.

▶ **Lemma 5.1** (Densities of any self-assembling systems are Π_2 – proof in appendix). *Let c^∞ be the limit configuration reached from some finite seed by some system among oritatami, turedos, freezing cellular automata or directed aTAM. Then for any pair of non-colinear vectors v_1, v_2, the upper density $\overline{d}_{v_1,v_2}(c^\infty)$ is a Π_2-computable number.*

For non-deterministic systems (both turedos and aTAM), we can state a similar lemma saying that, starting from any finite seed, there is always one orbit converging to a configuration with Π_2 density.

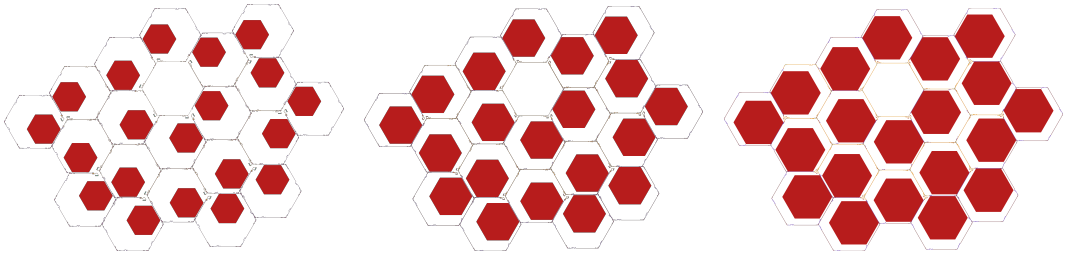


Figure 6 Increasing the density by folding a filled hexagon inside the macrocell expanded by 50, 100 and 200 extra $2n$-periods on each side. Actual oritatami simulation Zoom in for details

Arbitrarily dense simulation. The next theorem is a stronger version of Theorem 1.1, enforcing a *constant* and arbitrarily large density inside each macrocell of the oritatami simulation of a given turedo. Precisely, if we consider the *cell partition* of the oritatami plane into disjoint identical copies of a *macrocell tile* M induced by the map λ from the turedo world to the oritatami world, where each copy $\lambda(z) + M$ covers exactly the macrocell corresponding to the turedo position z (see Fig. 5 and Appendix C), then:

▶ **Theorem 5.2.** *For any turedo $\mathcal{T}$ of radius 1, and for any $\epsilon > 0$, there exists an oritatami system of delay 3 that simulates $\mathcal{T}$ and such that the number of occupied positions in each macrocell tile $\lambda(z) + M$ in the oritatami limit configuration is exactly k for all non-$\perp$ position z of the turedo limit configuration (and 0 for $\perp$ position), with $k \geqslant (1-\epsilon) \cdot \#M$.*

This result is obtained by 1) expanding of the macrocell with a straight line of length L in the middle of each side so that the empty triangles between the macrocells become negligible and 2) inserting a sequence in the scaffold that folds into a filled hexagon of radius $L(1+\alpha)$ inside the space freed inside the macrocell by the expansion. The factor $\alpha > 0$ is necessary to account for the increase of the exit pocket induced by the increase of the side length (more transcript needs to fit into the pocket) (see Fig. 6). Picking L large enough concludes the proof. The case of density 1 is treated in Appendix I.

Arbitrary Π_2-density. We conclude with the construction of a turedo of radius 1 that is able to produce limit configurations with any possible density when starting from the appropriate finite configuration. By possible density we mean any real number $d \in [0,1]$ which is Π_2-computable [25], *i.e.* such that there exists a computable sequence of rational numbers (q_n) with $d = \limsup_n q_n$. The construction is rather technical but the overall idea is simple (see Fig. 4b): at step n, leave a large annulus empty then densely fill another large annulus in such a way that the surface ratio between these annuli is q_n and that their sizes are large enough to dominate all the previously constructed annulus in anterior steps. The exact sequence of annuli is computed by the turedo in a sublinearly growing (hence negligible) corridor.

▶ **Theorem 5.3.** *There exists a turedo of radius 1 such that for any Π_2-computable number $d \in [0,1]$ and any pair of non-colinear vectors v_1, v_2, there is a finite initial global state such that the limit tape configuration c^∞ reached from it verifies: $\overline{d}_{v_1,v_2}(c^\infty) = d$.*

The Π_2-computability limitation is unavoidable as shown in Lemma 5.1, hence our result is optimal and actually gives a characterization of densities of limit configurations of continuous sequential self-avoiding systems (resp. turedo, resp. oritatami) started from finite configurations. Using our simulation framework and Theorem 5.2 we directly deduce Theorem 1.3.

References

1. Anonymous. Turedo to oritatami compiler: `https://hub.darcs.net/turedo2oritatami/turedo2oritatami`.
2. Sebastián Barbieri, Jarkko Kari, and Ville Salo. The group of reversible turing machines. In *Cellular Automata and Discrete Complex Systems*, pages 49–62. Springer International Publishing, 2016.
3. Florent Becker, Diego Maldonado, Nicolas Ollinger, and Guillaume Theyssier. Universality in freezing cellular automata. In *Sailing Routes in the World of Computation - 14th Conference on Computability in Europe, CiE 2018, Kiel, Germany, July 30 - August 3, 2018, Proceedings*, pages 50–59, 2018.
4. Marianne Delorme, Jacques Mazoyer, Nicolas Ollinger, and Guillaume Theyssier. Bulking ii: Classifications of cellular automata. *Theor. Comput. Sci.*, 412(30):3881–3905, 2011.
5. Erik D. Demaine, Jacob Hendricks, Meagan Olsen, Matthew J. Patitz, Trent A. Rogers, Nicolas Schabanel, Shinnosuke Seki, and Hadley Thomas. Know when to fold 'em: Self-assembly of shapes by folding in oritatami. In *DNA Computing and Molecular Programming - 24th International Conference, DNA 24, Jinan, China, October 8-12, 2018, Proceedings*, volume LNCS 11145, pages 19–36, 2018.
6. D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, and S. M. Summer. The tile assembly model is intrinsically universal. In *Proceedings of the 53rd Annual Foundations of Computer Science (FOCS)*, 2012.
7. David Doty, Jack H. Lutz, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *FOCS2012: Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*, pages 302–310, 2012.
8. Erling Følner. On groups with full banach mean value. *MATHEMATICA SCANDINAVICA*, 3:243, dec 1955.
9. Cody Geary, Guido Grossi, Ewan K. S. McRae, Paul W. K. Rothemund, and Ebbe S. Andersen. RNA origami design tools enable cotranscriptional folding of kilobase-sized nanoscaffolds. *Nature Chemistry*, 13:549–558, 2021.
10. Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinnosuke Seki. Programming biomolecules that fold greedily during transcription. In *MFCS2016: Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science*, volume 58 of *LIPIcs*, pages 43:1–43:14, 2016.
11. Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinnosuke Seki. Oritatami: A computational model for molecular co-transcriptional folding. *International Jounal of Molecular Sciences*, 9(2259), 2019.
12. Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinonsuke Seki. Proving the Turing universality of oritatami cotranscriptional folding. In *ISAAC 2018: Proceedings of the 29th International Symposium on Algorithms and Computation*, volume 123 of *LIPIcs*, pages 23:1–23:13, 2018.
13. Cody Geary, Paul W. K. Rothemund, and Ebbe S. Andersen. A single-stranded architecture for cotranscriptional folding of RNA nanostructures. *Science*, 345:799–804, 2014.
14. E. Goles, N. Ollinger, and G. Theyssier. Introducing freezing cellular automata. In J. Kari, I. Törmä, and M. Szabados, editors, *Exploratory Papers of Cellular Automata and Discrete Complex Systems (AUTOMATA 2015)*, pages 65–73, 2015.
15. Petr Kůrka. *Topological and symbolic dynamics*. Société Mathématique de France, 2003.
16. James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers. Computability and complexity in self-assembly. *Theory Comput. Syst.*, 48(3):617–647, 2011.
17. Diego Maldonado, Anahí Gajardo, Benjamin Hellouin de Menibus, and Andrés Moreira. Nontrivial turmites are turing-universal. *J. Cell. Autom.*, 13(5-6):373–392, 2018.
18. Yusei Masuda, Shinnosuke Seki, and Yuki Ubukata. Towards the algorithmic molecular self-assembly of fractals by cotranscriptional folding. In *CIAA2018: the 23rd International*

Conference on Implementation and Application of Automata, volume 10977 of *LNCS*, pages 261–273. Springer, 2018.

19 Nicolas Ollinger and Guillaume Theyssier. Freezing, bounded-change and convergent cellular automata. *CoRR*, abs/1908.06751, 2019.

20 Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.

21 Daria Pchelina, Nicolas Schabanel, Shinnosuke Seki, and Yuki Ubukata. Simple intrinsic simulation of cellular automata in oritatami molecular folding model. In Yoshiharu Kohayakawa and Flávio Keidi Miyazawa, editors, *LATIN 2020: Theoretical Informatics - 14th Latin American Symposium, São Paulo, Brazil, January 5-8, 2021, Proceedings*, volume 12118 of *Lecture Notes in Computer Science*, pages 425–436. Springer, 2020.

22 Paul W. K. Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2:2041–2053, 2004.

23 Nicolas Schabanel. Simple OS simulator: `http://perso.ens-lyon.fr/nicolas.schabanel/OSsimulator/`.

24 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

25 Xizhong Zheng and Klaus Weihrauch. The arithmetical hierarchy of real numbers. In Mirosław Kutyłowski, Leszek Pacholski, and Tomasz Wierzbicki, editors, *Mathematical Foundations of Computer Science 1999*, pages 23–33, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

Appendix

Table of Contents

A Omitted figures of the macrocell **16**

B Appendix: Zigzag Toolkit for Turedos **17**

C Appendix: Formalizing Simulation **21**

D Appendix: Definition of $\overline{d}_{v_1,v_2}$ and B_{v_1,v_2}, and transfert of density through simulation **22**

E Appendix: Proof of Fact 4.1 **23**

F Proof of Lemma 5.1 **23**

G Appendix: Construction of Theorem 4.2 **25**

H Appendix: Construction of Theorem 5.3 **29**

I Achieving density 1: a delay-3 oritatami filling the plane **35**

J The oritatami modules **38**

J.1 Notations . . . 39
J.2 Folding meter and Pocket . . . 40
J.3 Multi-layer interactions . . . 42
J.4 Transcript . . . 44
J.5 Scaffold . . . 46
J.6 Layer ordering and pocket sizes . . . 48
J.7 Read pocket . . . 49
J.8 Write module . . . 55
J.9 Write pocket . . . 57
J.10 U-turn module . . . 60
J.11 Corner module . . . 62
J.12 Exit pocket . . . 63
J.13 Exit interchange trap . . . 67
J.14 Step and shift modules . . . 68
J.15 Exit module . . . 70
J.16 Corner interchange block . . . 72
J.17 Middle interchange block . . . 73
J.18 Determining the macro-cell size: solving the fix point . . . 74
J.19 Tracker speed bump . . . 76

K The turedo-to-oritatami compiler **89**

A Omitted figures of the macrocell

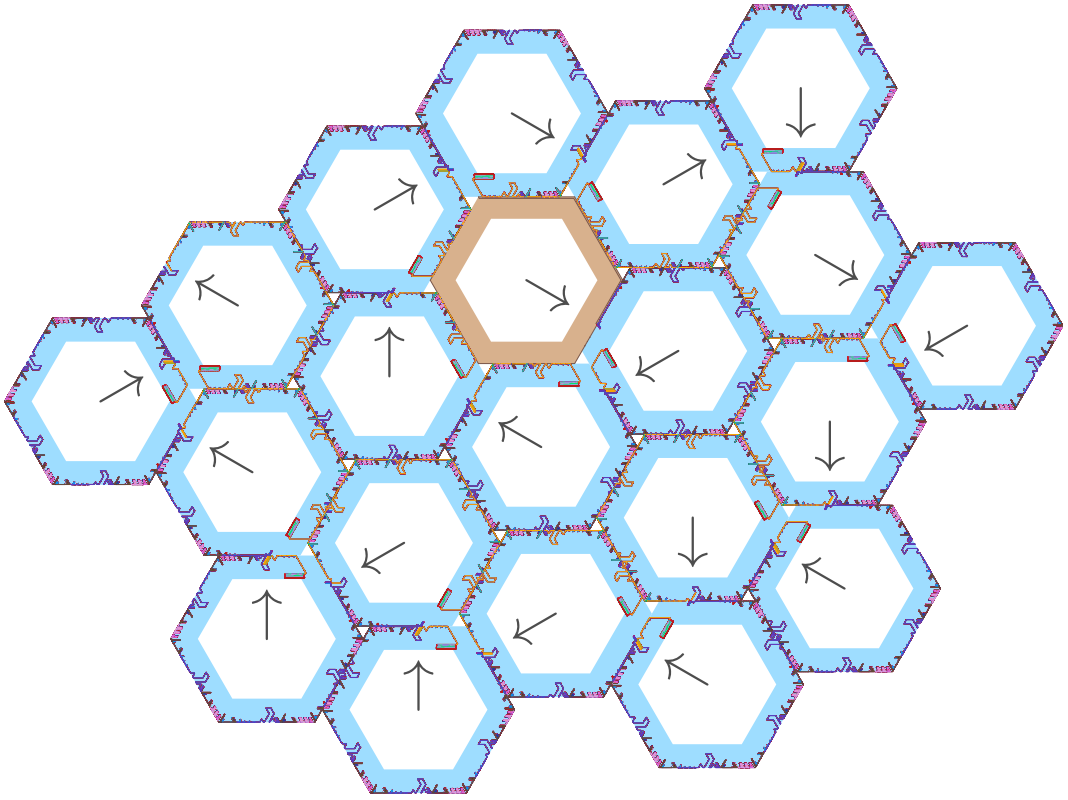


Figure 7 A path of 20 macrocells for q = 3-bits turedo whose internal state is always 1 and whose exit direction depends on the number f of unoccupied neighborings macrocells: if $f \leqslant 2$, it exits to the CCW-most free macrocell; if $f = 3$, it exits to the free macrocell in the middle; and otherwise, it exits to the CW-most free macrocell. This is an actual oritatami simulation. Zoom in for details

B Appendix: Zigzag Toolkit for Turedos

Zigzag transducers. First, if turedos are heavily constrained compared to general 2D Turing machines, they can easily embed 1D finite-state transducers and 1D Turing machines: by progressively filling a region of space by making zigzags, a turedo can compute the iterated applications of a 1D transducer on a finite word where each successive zigzag represents the result of an iteration of the 1D transducer. Moreover any finite-state machine can be used to expose useful information at the end of each of the zigzag for latter use. More precisely, if we fix two non co-linear directions $\overrightarrow{d_T}$ (for time) and $\overrightarrow{d_S}$ (for space) in N_H with $\overrightarrow{d_T} + \overrightarrow{d_S} \in N_H$, a zigzag construction starting from position z_0 if such that the beginnings of zigs and ends of zags will stick to the base line $z_0 + \mathbb{N} \cdot \overrightarrow{d_T}$ and the zigzags will only occupy a cone of space between $z_1 + \mathbb{N} \cdot \overrightarrow{d_T}$ and $z_1 + \mathbb{N} \cdot (\overrightarrow{d_T} + \overrightarrow{d_S})$ for some $z_1 = z_0 + x\overrightarrow{d_T}$. Concretely, it is straightforward to build a turedo $\mathcal{M}$ with neighborhood N_H such that (see Figure 8):

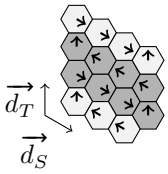


Figure 8 Example of a zigzag computation by a turedo: the t-th zigzag is represented in dark gray, its length l_t verifies $l_t = l_{t+1} = 4$ and $l_{t-1} = 3$.

- the t-th zig visits cells $z_0 + (2t-1)\overrightarrow{d_T}$ to $z_0 + (2t-1)\overrightarrow{d_T} + (l_t - 1)\overrightarrow{d_S}$ and the t-th zag visits cells $z_0 + 2t\overrightarrow{d_T} + (l_t - 1)\overrightarrow{d_S}$ to $z_0 + 2t\overrightarrow{d_T}$ where l_t denotes the width of the t-th zigzag;
- in some component of states of $\mathcal{M}$, the t-th zig contains a word u_t of length l_t, the t-th zag a word v_t of length l_t; v_t is obtained by applying some finite-state letter-to-letter transducer on u_t, and u_{t+1} is obtained by applying another finite-state letter-to-letter transducer on v_t and possibly append a new letter at the end determined by the state of some finite-state automaton run on v_t (so we have $|l_{t+1} - l_t| \leqslant 1$);
- in some other component of states of $\mathcal{M}$ position $z_0 + (2t+1)\overrightarrow{d_T}$ (first position of $(t+1)$-th zig) encodes the state of some finite-state automaton run on v_t and position $z_0 + (2t+2)\overrightarrow{d_T}$ (last position of $(t+1)$-th zag) encodes the state of some other finite-state automaton run on u_{t+1}.

Note that, in particular, one can simulate a 1D Turing machine by progressively building its space-time diagram in such a zigzag construction. In this case, the zig steps allow to move the simulated Turing head in one direction of the 1D tape, and the zag steps in the other direction. The additional information encoded at the beginning of zigs and at the end of zags can also be linked to the Turing machine: for instance, we can encode the information of whether some counter in the simulated Turing machine is non-zero.

Zigzag snakes. As seen above, turedos can perform arbitrary 1D computations in some organized regions of space. Zigzag snakes are a means to transport unbounded information (*i.e.* information that might not fit inside turedo's state set) from one region of space to another following possibly complicated paths. They are constant width zigzags where the width represents the information transmitted in unary. They globally move in some direction $\overrightarrow{d_T}$ while their zigzag oscillations of constant width are done in direction $\overrightarrow{d_S}$, where $\overrightarrow{d_T}$ and $\overrightarrow{d_S}$ verify $\overrightarrow{d_T} + \overrightarrow{d_S} \in N_H$. They can additionally do two things:

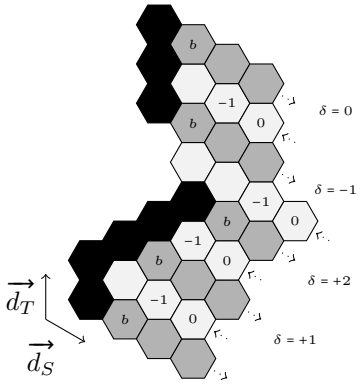


Figure 9 Example of zigzag snakes trajectory following obstacles. Zig phases are in dark gray, zag phases in light gray, and obstacles in black. Initially only black cells are occupied.

- they can globally shift along direction $\overrightarrow{d_S}$ to climb obstacles or in direction $-\overrightarrow{d_S}$ when there is no occupied cell to block them (as if there was a gravity field in direction $-\overrightarrow{d_S}$). The possible $\overrightarrow{d_S}$ shifts are limited to one unit per zigzag, so the snake can only follow 'smooth' curves. To do so, the turedo holds a shift value $\delta \in \{-1, 0, 1, 2\}$ in its state and the tape alphabet can possibly hold a *marking information* from $\{b, -1, 0\}$ used to detect the beginning and the end of the snake along direction $\overrightarrow{d_S}$: b stands for beginning, and -1 and 0 are used to mark the one but last and last position in direction $\overrightarrow{d_S}$. The δ value is used during the zig phase and updated during the zag phase. The zigzag cycle is as follows for a snake at position $z \in \mathbb{Z}^2$ with shift value δ at the beginning of the cycle (see Figure 9):

1. the zig phase starts by putting the mark 'b' in position z and moving in direction $\overrightarrow{d_S}$ until one of the following happens: either $\delta \leqslant 0$ and it reaches a position z' such that $z' - \overrightarrow{d_T}$ has a marking information equal to δ, in which case the next move is in direction $\overrightarrow{d_T}$ and the zag phase begins; or $\delta \geqslant 1$ and it reaches a position z' with $z' - \overrightarrow{d_T}$ empty, in which case the next move is in direction $\overrightarrow{d_T}$ if $\delta = 1$ and $\overrightarrow{d_T} + \overrightarrow{d_S}$ if $\delta = 2$ and the zag phase begins.
2. the zag phase consists in moving in direction $-\overrightarrow{d_S}$; it marks its two first steps by 0 and -1 respectively, and it goes on until one of the following happens:
   a. **terminating with no shift:** if position $z - \overrightarrow{d_T}$ is marked by b and $z - \overrightarrow{d_S}$ is occupied and $z + \overrightarrow{d_T}$ is empty, then move to $z + \overrightarrow{d_T}$, update δ to 0 and start a new zig;
   b. **terminating with simple obstacle in current zag:** else if $z - \overrightarrow{d_S}$ is occupied but $z + \overrightarrow{d_T}$ is free, then move to $z + \overrightarrow{d_T}$, update δ to +1 and start a new zig;
   c. **terminating with simple obstacle in next zig:** else if $z + \overrightarrow{d_T}$ is occupied but $z - \overrightarrow{d_T}$ has the mark 'b', then move to $z + \overrightarrow{d_T} + \overrightarrow{d_S}$, update δ to +1 and start a new zig;
   d. **terminating with double obstacle:** else if $z + \overrightarrow{d_T}$ is occupied and $z - \overrightarrow{d_T}$ has no mark 'b', then move to $z + \overrightarrow{d_T} + \overrightarrow{d_S}$, update δ to +2 and start a new zig;
   e. **terminating with free fall:** else if $z - \overrightarrow{d_T}$ is marked by b and $z - \overrightarrow{d_S}$ is empty, then move to $z - \overrightarrow{d_S}$, next move to $z - \overrightarrow{d_S} + \overrightarrow{d_T}$ which is supposed to be empty;

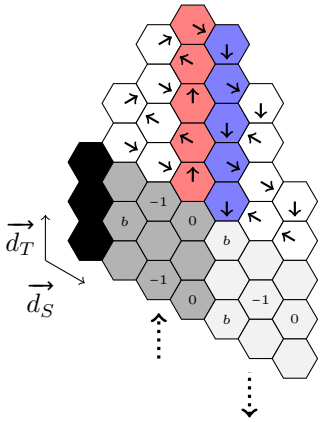


Figure 10 U-turn of a zigzag snake. In dark gray the forward snake and in light gray the backward snake.

update δ to -1 and starts a new zig;

f. any other situation is unspecified and will not be used in our constructions.

- zigzag snakes can also make U-turns, *i.e.* change their global movement vectors from $(\overrightarrow{d_T}, \overrightarrow{d_S})$ (forward snake) to $(-\overrightarrow{d_T}, \overrightarrow{d_S})$ (backward snake) while preserving their width (see Figure 10). After the U-turn, the positions that were at the end of zigs or beginning of zags for the forward snake become obstacles for the backward snake. The decision to make a U-turn is done at the end of a zag when encountering an obstacle, it is triggered by a combination of the internal state of the head and the symbol read on the obstacle. We will only use U-turns in regions were the snake is going straight ($\delta = 0$) and no obstacle is present that would block the backward snake. Suppose the width of the forward snake is w. The algorithm of the U-turn consists in first going on in the same global direction while reducing the size of the snake of one unit after each zigzag and marking a line of direction $\overrightarrow{d_T}$ above the 0 mark of the last zag of the forward snake. Do this until the snake is reduced to one unit. Then use that marked vertical line as reference to do the opposite operation: go in the backward direction and increase by one unit after each zigzag, until the 0 mark of the last zag of the forward snake is seen, precisely when reaching a position z such that $z - \overrightarrow{d_T} - \overrightarrow{d_S}$ is marked by 0. Then move to position $z - \overrightarrow{d_T}$ and at this point the first zig of the backward snake can begin with the correct reference width given by occupied cells z to $z + (w-1)\overrightarrow{d_S}$.

Zigzag snakes are useful because of the information they carry in their width. To use them in the context of a complex turedo with many components, we need two additional constructions:

- **Creating a snake of a given length:** this is done by first marking a segment of the desired length along some direction and then using the second phase of the U-turn construction. More precisely, suppose that some zigzag transducer is working with base line $z_0 + \mathbb{N} \cdot \overrightarrow{d_T}$ and space direction $-\overrightarrow{d_S}$ and that it reaches position $z_0 + (t + 2w - 1) \cdot \overrightarrow{d_T}$ after having marked $2w - 1$ position along this baseline. Then the second phase of the U-turn is triggered using this marked baseline to create a snake of global movement $(-\overrightarrow{d_T}, \overrightarrow{d_S})$, of width w and that starts a zig at position $z_0 + t \cdot \overrightarrow{d_T} + \overrightarrow{d_S}$ (see Figure 11).

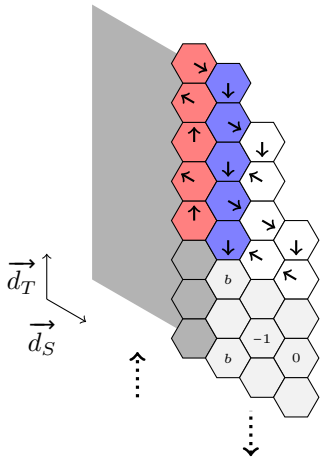


Figure 11 Creation of a zigzag snake of width 3. In dark gray the zone occupied by the zigzag transducer that marked 5 positions along its base line (in red). In light gray the created snake.

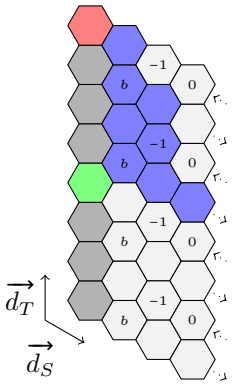


Figure 12 A successful equality test on the width of a zigzag snake. In dark gray the obstacles, in green the start marker and in red the stop marker. In blue the unary counter inside the snake which is represented in light gray.

- **Equality test on the width of a snake:** when the snake moves without shift ($\delta = 0$), it can realize some simple computational task inside an additional component of states while maintaining its standard movement like a constant space zigzag transducer would do. In particular, it can hold a unary counter no larger than its width, and updating it according to some information read on the obstacles that force its $\overrightarrow{d_T}$ movement. Using this technique, an equality test between the width of the snake and the distance between some start/stop markers read on the obstacles can be done as follows (see Figure 12):
  - the test subroutine is launched at a zig after the end of the previous zag reads some start marker on the adjacent obstacle;
  - then the subroutine run inside a subset of states and consists in reducing some unary counter by one unit at each zag starting from the full width of the snake;
  - when the unary counter is reduced to 1 at the end of a zag, the result of the test is *true* is the blocking obstacle has a stop mark, and *false* otherwise.

C Appendix: Formalizing Simulation

Recall that every oritatami system with delay δ is a turedo with radius $\delta + 1$, it follows that the definitions next apply to oritatami systems simulating turedo as well.

In the following definition, we formalize a notion of simulation that captures the ability of a turedo $\mathcal{T}_1$ to reproduce all the dynamics of a turedo $\mathcal{T}_2$ up to a spatio-temporal re-scaling. More concretely, to each step of $\mathcal{T}_2$ that modifies the configuration at some position, $\mathcal{T}_1$ responds in a constant number of steps by adding a constant size pattern in the corresponding position in some scaled-up hexagonal lattice of macro-cells (to have a concrete idea in mind the reader can think of macro-cells as hexagonal balls of radius r, but our definition below allows other kind of macro-cells). In this way, the precise evolution of $\mathcal{T}_2$ can be recovered from the evolution of $\mathcal{T}_1$.

The following definition implements the above idea allowing redundancy of coding (several macro-cell contents in the simulator can encode the same cell letter of the simulated system) and representation of state of the simulated head as a mixed coding into both tape and head state of the simulator.

▶ **Definition C.1.** *We say that a turedo* $\mathcal{T}_1 = (A_1, Q_1, q_1^0, r_1, \delta_1)$ *of global map* F_1 simulates *another turedo* $\mathcal{T}_2 = (A_2, Q_2, q_2^0, r_2, \delta_2)$ *of global map* F_2 *if:*

- *there exists a finite* macrocell tile $M \subseteq \mathbb{Z}^2$ *and a linear transformation* $\lambda : \mathbb{Z}^2 \to \mathbb{Z}^2$ *associated to two non-colinear integer vectors (i.e. a* 2×2 *matrix with integer coefficients), which defines a macro-lattice such that macro-cell tiles tile the plane hexagonally when placed on the macro-lattice, precisely:*
  - $M + \lambda(z) \cap M + \lambda(u) = \varnothing$ *as soon as* $z \neq u$ *and* $\mathbb{Z}^2 = \cup_{z\in\mathbb{Z}^2} \lambda(z) + M$,
  - $M + N_H \cap M + \lambda(z) \neq \varnothing \iff z \in \lambda(N_H) \cup \{(0,0)\}$,
- *for each* $a \in A_2$ *there is a collection of* tape patterns $P_a \subseteq A_1^M$ *with* $P_a \cap P_b = \varnothing$ *whenever* $a \neq b$, *and* $P_\perp = \{\perp^M\}$; *this defines a* tape decoding map $\phi : X \to (A_2)^{\mathbb{Z}^2}$ *defined on the set* X *of configurations* $c \in A_1^{\mathbb{Z}^2}$ *such that* $(\forall z \in \mathbb{Z}^2)\, c_{|\lambda(z)+M} \in \cup_{a\in A_2} P_a$, *as follows:* $\phi(c)(z) = a$ *if* $c_{|\lambda(z)+M} \in P_a$;
- *for each* $q_2 \in Q_2$ *there is a collection of state patterns* $R_{q_2} \subseteq A_1^M \times M \times Q_1$ *with* $R_{q_2} \cap R_{p_2} = \varnothing$ *whenever* $q_2 \neq p_2$; *this defines the set* $X \subseteq \mathcal{S}_{\mathcal{T}_1}$ *of* decodable global states (c, x, q_1) *of* $\mathcal{T}_1$ *which are exactly those verifying: for all* z, $c_{|\lambda(z)+M} \in \cup_{a\in A_2} P_a$ *if* $x \notin \lambda(z) + M$ *and* $(c_{|\lambda(z)+M}, x - \lambda(z), q_1) \in \cup_{q_2\in Q_2} R_{q_2}$ *if* $x \in \lambda(z) + M$;
- *the above elements define a* global state decoding map $\phi^+ : X \to (A_2)^{\mathbb{Z}^2} \times \mathbb{Z}^2 \times Q_2$ *by* $\phi^+(c_1, z_1, q_1) = (c_2, z_2, q_2)$ *where* z_2 *is the unique point such that* $z_1 \in \lambda(z_2) + M$ *and* q_2 *is the unique state such that* $(c_{|\lambda(z_2)+M}, z_1 - \lambda(z_2), q_1) \in R_{q_2}$, *and* $c_2(z_2) = \perp$ *and, for all* $z \neq z_2$, $c_2(z) = a$ *where* a *is the unique symbol such that* $P_a \ni c_{|\lambda(z)+M}$;
- *there is a set of local constraints defining the subset of* valid global states $X^+ \subseteq X$ *as follows: for each "oriented domino"* $(a, v, b) \in A_2 \times N_H \times A_2$ *there is a subset of valid tape patterns* $P_{(a,v,b)} \subseteq P_a \subseteq A_1^M$, *and*

$$X^+ = \left\{(c_1, z_1, q_1) \in X : \forall z \in \mathbb{Z}^2, \forall v \in N_H, c_{1|\lambda(z)+M} \in P_{(c_2(z),v,c_2(z+v))}, \text{ where } (c_2, z_2, q_2) = \phi^+(c_1, z_1, q_1)\right\};$$

  moreover we ask that for any global state (c_2, z_2, q_2) *there is a corresponding global state* $(c_1, z_1, q_1) \in X^+$ *with* $\phi^+(c_1, z_1, q_1) = (c_2, z_2, q_2)$;
- *a time rescaling factor* $T \geqslant 1$,

such that for any configuration $c_2 \in (A_2)^{\mathbb{Z}^2}$ *and* $z_2 \in \mathbb{Z}^2$ *such that* $c_2(z_2) = \perp$, *and for any global state* $(c_1, z_1, q_1) \in X^+$ *such that* $\phi^+(c_1, z_1, q_1) = (c_2, z_2, q_2)$, *it holds that* $F_1^T(c_1, z_1, q_1) \in X^+$ *and* $\phi^+(F_1^T(c_1, z_1, q_1)) = F_2(c_2, z_2, q_2)$.

Note that the relation between F_1^T and F_2 through ϕ^+ can be iterated along the considered orbits and that blank letter $\perp$ is represented by a $\perp$ pattern only. This makes Definition C.1 strong enough to preserve computability and density of limit configurations as shown in the following lemmas.

▶ **Lemma C.2.** *Taking notations of definition C.1, suppose that turedo $\mathcal{T}_1$ simulates turedo $\mathcal{T}_2$ and that c_2^∞ is the limit configuration reached by $\mathcal{T}_2$ from global state (c_2, z_2, q_2) and c_1^∞ is the limit configuration reached by $\mathcal{T}_1$ from global state $(c_1, z_1, q_1) \in X^+$ such that $\phi^+(c_1, z_1, q_1) = (c_2, z_2, q_2)$. Suppose also that c_2 is finite and c_2^∞ infinite. Then it holds $\phi(c_2^\infty) = c_1^\infty$. In particular, c_1^∞ is computable from c_2^∞.*

Proof. Take any $z'' \in \mathbb{Z}^2$. By the definition of limit configurations, there is a freezing time $t^1_{z''}$ such the value of $c_1^\infty(z'')$ is obtained after $t^1_{z''}$ steps at position z'' in the orbit of $\mathcal{T}_1$. The same holds for $\mathcal{T}_2$ and c_2^∞ and we denote by $t^2_{z''}$ the corresponding convergence times. Now take $\tau \geqslant \max(t^2_z, \max_{z''' \in \lambda(z'')+M} t^1_{z'''})$. By the simulation, we have $\phi^+(F_1^{T\tau}(c_1, z_1, q_1)) = F_2^\tau(c_2, z_2, q_2)$. We also know by the assumption that the run of F_2 starting from (c_2, z_2, q_2) is infinite (because it must produce an infinite c_2^∞ starting from a finite c_2). The same is true for F_1 starting from (c_1, z_1, q_1). We deduce that the heads of F_2 will not visit cell z'' after time τ and similarly for all cells $\lambda(z'') + M$ of F_2. It follows that $(c_1^\infty)_{|\lambda(z'')+M} \in P_{c_2^\infty(z'')}$. This being true whatever the choice of z'', we thus showed $\phi(c_1^\infty) = c_2^\infty$. The fact that c_2^∞ is computable from c_1^∞ follows immediately. ◀

D Appendix: Definition of $\overline{d}_{v_1,v_2}$ and B_{v_1,v_2}, and transfert of density through simulation

Let $v_1, v_2 \in \mathbb{Z}^2$ be two non-colinear vectors. Denote by $H(v_1, v_2)$ the closed pseudo-hexagon in the real Euclidean plane with extremal points: $v_1, v_2, v_2 - v_1, -v_1, -v_2, v_1 - v_2$, and by $B_{v_1,v_2}(n)$ the pseudo-ball made of integer points inside $H(nv_1, nv_2)$: $B_{v_1,v_2}(n) = \mathbb{Z}^2 \cap H(nv_1, nv_2)$. We then define the associated (upper) density of non-blank cells in configuration c by:

$$\overline{d}_{v_1,v_2}(c) = \limsup_n \frac{\#\{z \in B_{v_1,v_2}(n) : c(z) \neq \perp\}}{\#B_{v_1,v_2}(n)}.$$

We can now state a density transfer lemma for simulations that use coding patterns with a constant number of non-blank cells.

▶ **Lemma D.1.** *Taking notations of definition C.1, suppose that turedo $\mathcal{T}_1$ simulates turedo $\mathcal{T}_2$ in such a way that, for some $k \in \mathbb{N}$, the number of occurrences of $\perp$ in each pattern of P_a for $a \in A_2$ and $a \neq \perp$ is always k. Then for any non-colinear pair of vectors (v_1, v_2), there is a pair (v_1', v_2') such that, if c_2^∞ is the infinite limit configuration reached by $\mathcal{T}_2$ from global state (c_2, z_2, q_2) with c_2 finite and c_1^∞ is the limit configuration reached by $\mathcal{T}_1$ from global state $(c_1, z_1, q_1) \in X^+$ with $\phi^+(c_1, z_1, q_1) = (c_2, z_2, q_2)$, then $\overline{d}_{v_1,v_2}(c_1^\infty) = \frac{|M|-k}{|M|} \cdot \overline{d}_{v_1',v_2'}(c_2^\infty)$ where $|M|$ denotes the size of M.*

Proof. First, the hypothesis of Lemma C.2 are fulfilled so we have $\phi(c_1^\infty) = c_2^\infty$. The 2×2 matrix defining map λ being non-singular (on field $\mathbb{Q}$) we deduce that there are vector (v_1', v_2') such that $(Nv_1, Nv_2) = \lambda(v_1', v_2')$ for some integer $N \geqslant 1$ ((v_1, v_2) can be reached from a $\mathbb{Q}$-vector and then it suffices to multiply by the least common multiple of denominators). This means that the cells of $\mathcal{T}_2$ in pseudo-ball $B_{v_1',v_2'}(n)$ are simulated by macro-cells in $\mathcal{T}_1$

whose union is

$$U(n) = \bigcup_{(a,b)\in B_{v_1',v_2'}(n)} \lambda(a,b) + M.$$

Let us denote by $d_{S,1}$ the density of non-blank cells in c_1^∞ restricted to some finite set S:

$$d_{S,1} = \frac{\#\{z'' \in S : c_1^\infty(z'') \neq \bot\}}{\#S}$$

and similarly $d_{S,2}$ for c_2^∞. From the hypothesis and the choice of $U(n)$ above we have:

$$d_{B_{v_1',v_2'}(n),2} = \frac{|M|-k}{|M|} \cdot d_{U(n),1}$$

because to each blank cell in $B_{v_1',v_2'}$ corresponds a blank macro-cell in $U(n)$ and to each non-blank cell in $B_{v_1',v_2'}$ corresponds a macro cell with exactly $|M|-k$ non-blank cells. Remark that $\#U(n) \in \Omega(n^2)$ and $\#\big(U(n) \setminus B_{Nv_1,Nv_2}\big) \in O(n)$ (by the tiling property of macro-cells on the macro-lattice), so $d_{U(n),1} = d_{B_{Nv_1,Nv_2}(n),1} + o(1)$. Moreover for any integer i with $|i| \leqslant N$ it also holds $d_{B_{v_1,v_2}(Nn+i),1} = d_{B_{Nv_1,Nv_2}(n),1} + o(1)$. We conclude that

$$\overline{d}_{v_1',v_2'}(c_2^\infty) = \limsup_n d_{B_{v_1',v_2'}(n),2} = \limsup_n \frac{|M|-k}{|M|} \cdot d_{B_{nv_1,nv_2},1} = \frac{|M|-k}{|M|} \cdot \overline{d}_{v_1,v_2}(c_1^\infty).$$

◀

E Appendix: Proof of Fact 4.1

Proof. To compute $\tau_s(z)$ from $c^\infty(z)$ one just computes step by step the orbit until reaching the first step t such that the configuration at z is equal to $c^\infty(z)$. Then $\tau_s(z) = t$. Conversely, one computes $c^\infty(z)$ from $\tau_s(z)$ by just computing the orbit for $\tau_s(z)$ steps and returning the value obtained at position z.

If we suppose that $u \in \mathbb{Z}^2$ is a computable escape direction associated with computable map μ, then the map

$$\alpha : z \in \mathbb{Z}^2 \mapsto \max\{t : \mu(t) \leqslant u \cdot z\}$$

is also computable because $\mu(t) \to \infty$ and μ is non-decreasing. Then $\tau_s(z) \leqslant \alpha(z) + 1$ and thus τ_s is computable (by simulation during $\alpha(z)+1$ steps). ◀

F Proof of Lemma 5.1

To prove Lemma 5.1 we actually show a more general result.

▶ **Lemma F.1.** *Let $(c^t)_{t\in\mathbb{N}}$ be a sequence of configurations of $A^{\mathbb{Z}^2}$ converging towards c^∞ and verifying the following:*

1. *$c^t(z) = \bot$ implies $c^{t'}(z) = \bot$ for any $t' \leqslant t$,*
2. *$(t,z) \mapsto c^t(z)$ is computable,*
3. *there is ϕ computable such that for all t and any $z \notin B(\phi(t))$ it holds $c^t(z) = \bot$.*

This lemma applies to turedo starting from finite initial configurations (in this case the sequence (c^t) is the orbit of tape configurations, and ϕ is just a linear map), but also to directed aTAM self-assembly system [20, 16] and freezing cellular automata [19].

Proof. Denote by $d_{t,n}$ the proportion of non-blank cells in $B_{v_1,v_2}(n)$ for the configuration c^t, and denote by $d_{\infty,n}$ the proportion of non-blank cells in $B_{v_1,v_2}(n)$ for configuration c^∞. Clearly $d_{t,n}$ is a rational number computable from t and n by computability of $(t,z) \mapsto c^t(z)$. Moreover, by hypothesis 1, $(d_{t,n})_t$ is monotone increasing so we have $d_{\infty,n} = \sup_t d_{t,n}$ and therefore

$$\overline{d}_{v_1,v_2}(c^\infty) = \inf_m \sup_{n \geqslant m} \sup_t d_{t,n} = \inf_m \sup_t \sup_{n \geqslant m} d_{t,n}.$$

Note also that if $n_1 \geqslant n_2 \geqslant \phi(t)$ then d_{t,n_1} is smaller than d_{t,n_2} by hypothesis 3. We deduce that $q_{m,t} = \sup_{n \geqslant m} d_{t,n}$ is a rational number computable from t and m. Finally we have $\overline{d}_{v_1,v_2}(c^\infty) = \inf_m \sup_t q_{m,t}$ and [25, Lemma 3.2] shows that it is a Π_2-computable number. ◀

G Appendix: Construction of Theorem 4.2

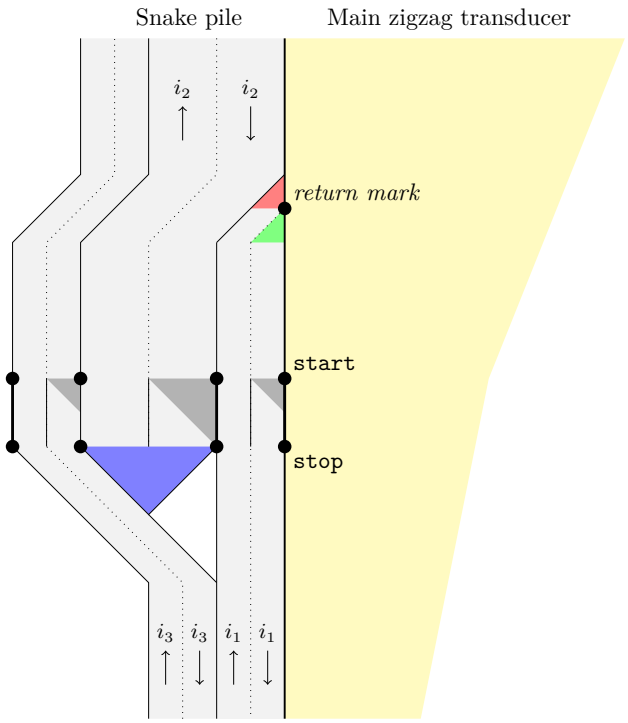


Figure 13 Schematic overview of the execution around a marking zone: snakes (in light gray) corresponding to machines i_1, i_2 and i_3 with halting time $T(i_1) \leqslant T(i_2) \leqslant T(i_3)$ but $i_1 < i_2$ and $i_3 < i_2$. In green a snake launching zone, in red a snake ending zone, in blue a U-turn after a successful equality test. Tests are represented in dark gray.

Main zigzag transducer for the Turing simulation. The zigzag transducer has global directions $(\overrightarrow{d_T}, \overrightarrow{d_S})$ and run two algorithms in parallel that share the global variable m:

Algorithm 1 — **Turing simulation**

- MACHINES ← ∅
- $n \leftarrow 0$
- loop forever:
  1. $n \leftarrow n + 1$
  2. MACHINES ← MACHINES ∪ $\{n\}$
  3. for each $i \in$ MACHINES do:
     - **a.** simulate n steps of machine i starting from an empty tape
     - **b.** if i has halted during the simulation then
       - **i.** if necessary, wait until $m > i$
       - **ii.** MACHINES ← MACHINES $\setminus \{i\}$
       - **iii.** launch a **zigzag snake subroutine** of width i

Algorithm 2 — **Marking**

- $m \leftarrow 0$
- $T \leftarrow \varnothing$
- loop forever:
  - wait $2m + 3$ zigzags
  - $m \leftarrow m + 1$
  - at a zag do $T \leftarrow$ `stop`
  - at the next zig do $T \leftarrow \varnothing$
  - wait $m - 1$ zigzags and at the last zag do $T \leftarrow$ `start`
  - at the next zig do $T \leftarrow \varnothing$
  - wait $2m + 3$ zigzags

Moreover, at each zig or zag, it writes the value of variable $T \in \{\varnothing, \texttt{start}, \texttt{stop}\}$ in some component of the alphabet at the positions of the base line of the zigzag transducer. From the algorithm above the marks follow the following pattern for successive values of m along direction $\overrightarrow{d_T}$ on the base line: a stop marked at the end of a zag then m zigzags finished by a start mark at the last zag, then m zigzags without marks.

Zigzag snake subroutine for flagging. The zigzag snake subroutine is as follows:

- when the subroutine is triggered by the main zigzag transducer on value i (halting machine and desired snake width) it freezes the Turing simulation algorithm (Algorithm 1) but it waits for the end of a loop of the marking algorithm (Algorithm 2) before freezing it ; freezing means that the zigzag movements continues but the content of the simulation tape is copied unchanged from one zigzag to the next; then it does i more zigzags to ensure that the last marked zone is far enough;
- next, it marks a segment of length i along $\overrightarrow{d_T}$ and puts a *return mark* at the end of this segment;
- next the zigzag movement of the transducer is stopped and a snake creation is launched as described earlier (see Figure 11);
- once created the snake of width i moves in direction $-\overrightarrow{d_T}$ and uses space direction $-\overrightarrow{d_S}$; it then adapts by small shifts its trajectory to obstacles it encounters (see Figure 9 for the detailed mechanism, in the present case obstacles are actually either the base line of the main zigzag transducer or previously launched zigzag snake as shown in Figure 13);
- when reading a `start` marker on an obstacle, it starts an equality test (see Figure 12) and replicates the `start` marker at the end of the next zig, on the "back" of the snake: the replicated marker is therefore shifted by one unit in direction $-\overrightarrow{d_T}$; the same is done for the `stop` marker;
- if the test is successful, then a U-turn is launched (see Figure 10); the construction is such that any snake will eventually encounter a `start/stop` test zone of length corresponding to its width;
- when the backward snake move in direction $\overrightarrow{d_T}$ after the U-turn, it does not perform tests when encountering `start/stop` test marker by it copies it like the forward snake did; the shift induced by the copy mechanism in the forward snake is therefore compensated exactly;
- when the backward snake arrives at the *return mark* and during i steps, the backward snake disappears progressively like in first phase of U-turn (see Figure 10 for the details, and the red part in Figure 13 for an overview) ; during this i steps the zigzags cross the base line but do nothing in the region of the zigzag transducer (algorithms are

frozen and the simulated tape content is just copied from one zigzag to the next) ; this progressive disappearance of the snake ensures that the next snake that will come in that region encounters a smooth path of obstacles that it can climb with its shift mechanism (Figure 9);

- the control is transferred back to the main zigzag transducer and Algorithms 1 and 2 are unfrozen.

Proof of Theorem 4.2. We consider the machine described above started from a blank tape configuration with the head in the suitable initial state at position $(0,0)$. First, the marking algorithm (Algorithm 2) and the writing process on the tape ensures that the base line $\mathbb{N}\overrightarrow{d_T}$ contains a succession of *test segments* $S_m = \{t_m\overrightarrow{d_T}, (t_m+1)\overrightarrow{d_T}, \dots, (t_m+2m-2)\overrightarrow{d_T}\}$ with t_m odd and $t_m\overrightarrow{d_T}$ marked with `stop` (written a the end of a zag), $(t_m+2m-2)\overrightarrow{d_T}$ marked with `start` and all other positions of S_m marked with $\varnothing$. Segment S_m is such that it will produce a successful equality test for snakes of width m (see Figure 12). All positions outside test segments are also marked with $\varnothing$, so a snake of width m can only have a successful equality test at the level of segment S_m. Moreover the distance between two consecutive test segments S_m and S_{m+1} is strictly more than $2(m+1)$ (the lower bound comes from Algorithm 2, but the distance can be larger when there is an interruption by a zigzag snake subroutine) and the distance between a test segment S_m and any return point of a zigzag snake subroutine is also at least $2(m+1)$ (because the subroutine freezes Algorithm 2 end the end of a loop and the next loop begins by a waiting instruction after the subroutine terminates).

Besides, Algorithm 1 guaranties that if machine i halts during the simulation and launches a zigzag snake subroutine when the head is at position $t\overrightarrow{d_T}$ on the baseline then $t > t_i$ (instruction 3.b.i of Algortihm 1), which means that the head is beyond the test segment whose size corresponds to snakes of width i.

At any given step of the execution when the machine is not in the zigzag snake subroutine, let T be maximal such that the Turing head has reached position $T\overrightarrow{d_T}$ and define the *snake pile profile* as positions $p(t)$ for $0 \leqslant t \leqslant T$ such that $p(t) = t\overrightarrow{d_T} - s\overrightarrow{d_S}$ where s is maximal such that the tape at this position $p(t)$ is not b.

Now, one can check that the following invariants are maintained between to launches of the zigzag snake subroutines:

- the snake pile profile is smooth for a snake that run on top of it in direction $-\overrightarrow{d_T}$ (thus allowing a future snake to follow this profile); precisely: $p(t-1)$ is closer to the baseline than $p(t)$, or $p(t-1) = p(t) - \overrightarrow{d_T} - \overrightarrow{d_S}$;
- at the level of each test segment S_m already marked, the snake pile profile is flat ($p(t+1) - p(t) = \overrightarrow{d_T}$); this is because the only part of the zigzag snake subroutine that produces a non-flat profile is the creation of the snake, the U-turn and the progressive disappearance; they are all granted to be far away from test segments (the subroutine waits until leaving a test segment before creating a snake, and as said above a test segment S_m is separated by at least $2m+1$ from S_{m-1} which is enough room for a snake of width m to make a U-turn);
- the positions $p(t)$ for t such that $t\overrightarrow{d_T} \in S_m$ have the same marks as $t\overrightarrow{d_T}$ (thus allowing the equality test to run properly for future snakes);
- at the level of each test segment S_m the snake pile profile thickness is bounded: if $t\overrightarrow{d_T} \in S_m$ then $p(t) = t\overrightarrow{d_T} - s\overrightarrow{d_S}$ with $s \leqslant 2m^2$. This is because no snake of width strictly larger than m can reach the level of S_m (it has to make a U-turn at the level of another

test segment placed further along direction $\overrightarrow{d_T}$ and is launched even further).

From the above discussion, we deduce the key property of the execution of the constructed turedo: for each machine i that halts, there is a snake of width i making a U-turn inside the region

$$R_i = \{(t-k)\overrightarrow{d_T} - s\overrightarrow{d_S} : 0 \leqslant k \leqslant 2i \text{ and } 0 \leqslant s \leqslant 2i^2\}.$$

R_i is clearly computable from i and testing the presence of a U-turn in that region can be done by testing whether there exists a position $p \in R_i$ such that the tape is blank at p but not at $p - \overrightarrow{d_S}$ (indeed, U-turns are the only part of the construction where the snake pile contains blank holes, as shown in Figure 13).

We deduce that the halting problem can be decided when given as oracle the limit configuration reached by the execution. The theorem follows. ◀

Note that the choice of the halting problem in the above construction can easily be replaced by any recursively enumerable set.

H Appendix: Construction of Theorem 5.3

General idea. We first describe an ideal target configuration to achieve the desired density. Let us fix the pair of vectors v_1, v_2 and remove the dependence on these vectors in pseudo-ball notation: $B(r) = B_{v_1,v_2}(r)$. Consider the target density $d = \limsup_n \frac{p_n}{q_n}$. We discard density 0 and 1 which are easily treated as special cases. We can therefore suppose that $\frac{p_n}{q_n}$ is eventually bounded away from 0 and 1. We can also suppose that $1 \leqslant p_n \leqslant q_n - 1$ for all n (the inequality must hold infinitely often because $0 < d < 1$ and we can replace the computable sequence (p_n, q_n) by the computable sequence whose nth term is (p_m, q_m) where $m \geqslant n$ is the first index such that the inequality holds). The basic ingredient of our construction is a well chosen sequence of annuli:

- define the annulus between two pseudo-balls of respective radii $r < R$ by $C(r, R) = B(R) \setminus B(r)$. Note that the size of pseudo-balls and annulus verify: $b(r) = \#B(r) \in \alpha r^2 + o(r^2)$ and $c(r, R) = \#C(r, R) \in \alpha(R^2 - r^2) + o(R^2)$.
- choose $r_{n+1} \geqslant q_{n+1}^2 R_n^2$ and let R_{n+1} be the smallest radius greater than r_{n+1} such that $\frac{c(r_{n+1},R_{n+1})}{b(R_{n+1})} \geqslant \frac{p_n}{q_n}$.

This choice ensures the two following properties:

1. a good enough approximation of density: $\frac{c(r_{n+1},R_{n+1})}{b(R_{n+1})} \in \left[\frac{p_n}{q_n}, \frac{p_n}{q_n} + \epsilon(n)\right]$ with $\epsilon(n) \in o(1)$, because $\frac{c(r_{n+1},R_{n+1}-1)}{b(R_{n+1}-1)} < \frac{p_n}{q_n}$ and the derivative of $R \mapsto \frac{c(r_{n+1},R)}{b(R)}$ is $O(\frac{1}{R})$;
2. a thick enough annulus (useful condition for the turedo construction): $R_n - r_n \in \Omega(R_n^{\frac{1}{4}})$. Indeed the condition $c(r_n, R_n) \geqslant \frac{p_n \cdot b(R_n)}{q_n}$ implies $R_n - r_n \in \Omega(\frac{r_n}{q_n})$ (because $p_n \geqslant 1$) so $R_n - r_n \in \Omega(r_n^{\frac{1}{2}})$. Moreover, we have $R_n \in O(q_n r_n)$ because

$$\frac{c(r_n, q_n r_n)}{b(q_n r_n)} \geqslant \frac{q_n - 1}{q_n + 1} + o(1)$$

and we know that $\frac{p_n}{q_n}$ is eventually bounded away from 1 because we supposed that the target density d is strictly less than 1. Therefore we conclude $R_n - r_n \in \Omega(R_n^{\frac{1}{4}})$.

Our ideal target configuration is non-blank exactly on $\bigcup_n C(r_n, R_n)$. For a classical (non self-avoiding) computation process, the next step would be to construct such a configuration and conclude from here. However it is impossible to produce such a configuration with a turedo from a finite initial configuration (because in particular it contains infinitely many N_H-connected components). The following lemma proves that it is sufficient to produce a good enough approximation of this sequence of annuli to achieve the correct density.

▶ **Lemma H.1.** *If a configuration γ is such that there is a set X with:*

1. $\#X \cap B(r) \in o(r^2)$,
2. *for any $z \in C(r_n, R_n) \setminus X$ we have $\gamma(z) \neq \perp$,*
3. *for any $z \in C(R_n, r_{n+1}) \setminus X$ we have $\gamma(z) = \perp$,*

then $\overline{d}_{v_1,v_2}(\gamma) = d$.

Proof. Denote by d_r the non-blank density inside pseudo-ball $B(r)$ for configuration γ:

$$d_r(\gamma) = \frac{\#\{z \in B(r) : \gamma(z) \neq \perp\}}{b(r)}.$$

First, we can reduce to the case where $X = \varnothing$: if $\gamma_\varnothing$ is a configuration verifying the hypothesis of the lemma with X empty, and γ is any configuration equal to $\gamma_\varnothing$ everywhere except on a

set X verifying the hypothesis, then:

$$|d_r(\gamma) - d_r(\gamma_\varnothing)| \leqslant \frac{\#X \cap B(r)}{b(r)} \in o(1)$$

so the two sequences have same limsup.

Now suppose that $X = \varnothing$, denote $d_r = d_r(\gamma)$, and estimate d_{R_n} as follows:

$$d_{R_n} = \frac{c(r_n, R_n) + \beta(n)}{b(R_n)}$$

where $0 \leqslant \beta(n) \leqslant b(R_{n-1})$. The choice or r_n and R_n above then gives $d_{R_n} = \frac{p_n}{q_n} + o(1)$. Therefore $\limsup_n d_{R_n} = d$. To conclude that $\overline{d}_{v_1,v_2}(c) = \limsup_n(d_n) = d$ it is sufficient to verify that the lim sup is actually realized on the subsequence $(R_n)_n$. Indeed for large enough n, density d_r is increasing on interval $[r_n, R_n]$ and decreasing on interval $[R_n, r_{n+1}]$ because, since d_r is eventually bounded away from both 0 and 1, we have:

- $d_{r+1} - d_r \geqslant \frac{d_r \cdot b(r) + c(r,r+1)}{b(r+1)} - d_r = \frac{c(r,r+1)(1-d_r)}{b(r+1)} > 0$ when $r \in [r_n, R_n]$,
- $d_{r+1} - d_r \leqslant \frac{d_r \cdot b(r)}{b(r+1)} - d_r = \frac{-d \cdot c(r,r+1)}{b(r+1)} < 0$ when $r \in [R_n, r_{n+1}]$.

◀

Description of the turedo. We now describe a turedo able to produce limit configurations that satisfy the hypothesis of Lemma H.1. We want a single turedo for all choices of d and vectors v_1, v_2, so the machine computing the sequence $(p_n, q_n)_n$ and v_1 and v_2 are not fixed, cannot be stored in the internal states of the turedo, and will be part of the initial configuration. However, in a given run of the turedo, v_1 and v_2 are constant compared to the growing radius of the annuli. Any suitable pre-computation can be done on the representation of v_1 and v_2 and associated vectors to simplify the work of the turedo.

The turedo has two components: a main Turing computation to compute parameters of successive annuli, and a annulus filling routine that visits approximately all cells of a given annulus and gives back control to the Turing component once finished. See Figure 14 for an overview of the turedo's behavior.

The turedo runs the main Turing computation using a zigzag transducer technique in time direction $\overrightarrow{d_T} = v_1$ and space direction $\overrightarrow{d_S}$ (geometrical details below). While doing the computation, the turedo keeps track of a step counter τ that contains the exact v_1 coordinate of the head (this is done by incrementing said counter at each zigzag). All Turing computations will be done in space $O(\log(\tau))$. To ensure this, we first suppose that the Turing space required to compute (p_n, q_n) is logarithmic in the Turing time required (space complexity is always at most time complexity and it is always possible to slow it down exponentially if necessary). The main Turing computation is as follows:

Algorithm 3 — **Computing annuli**

- $n \leftarrow 0$
- $r \leftarrow 0$
- $R \leftarrow 0$
- loop forever
  1. $n \leftarrow n + 1$
  2. compute (p_n, q_n)
  3. compute $r \leftarrow q_n^2 R^2$
  4. compute the smallest $x \geqslant r$ such that $\frac{c(r,x)}{b(x)} \geqslant \frac{p_n}{q_n}$
  5. $R \leftarrow x$
  6. compute $w \in \Omega(log(R))$ large enough to hold the binary representation of R and r
  7. do zigzags until the internal step counter τ is exactly r
  8. call the filling routine of inner radius r, outer radius R, and step increment w

Step 1 to 6 inside the loop can be done in polynomial time in the size of the Turing tape content (all integers are represented in binary), which guaranties that the value of τ when starting step 7 is less than r. For the computation in step 3, we already saw a $O(q_n r)$ bound on the value of x so we can do a dichotomy search (the numbers $c(r, x)$ and $b(x)$ can be efficiently computed using Pick's theorem). We deduce the key property of the algorithm: when the annulus filling routine is called at the nth iteration of the loop, the turedo's head is exactly at position $r_n v_1$ and the Turing tape contains the values r_n and R_n and w.

The annulus filling routine aims at visiting approximately all cells of $C(r, R)$ and does it by filling successive layers which are annuli $C(r + kw, r + (k + 1)w)$ for $0 \leqslant k \leqslant O(R/\log(R))$ with $w \in \Omega(\log(R))$. The reason to subdivide into layers is that the filling process will make errors at each extremity of the pseudo-hexagonal shape (around positions $(r + kw)v_1$, $(r + kw)v_2$, etc) in order to deal with change of direction between two consecutive sides of the pseudo-hexagon. This error will be $O(w^2)$ for each layer of width w so, by choice of $w \in \log(R)$ the accumulation of errors is small enough in any pseudo ball $B(r + i)$ for $O \leqslant i \leqslant R - r$ to apply Lemma H.1 (recall that r and R are polynomially related as shown above). The filling routine is thus an alternation between forward phases where the turedo's head is making zigzags of constant width $O(\log(R))$ in some direction during $R - O(\log(n))$ steps, then a phase of direction change using only space $O(\log(R)^2)$ inside the current layer. Note that the width of each layer is enough to hold all the information about the shape of the annulus layer to fill, in particular the length of its sides. Note also that after completing an entire layer, the turedo's head is back to the computation zone where it can compute and move to the next reference position $(r + kw)v_1$ to start the next layer. The outer layer has a possibly different, up to two times larger, width to complete exactly the annulus $C(r, R)$.

Geometrical details. The turedo's algorithm involves a finite number of directions: v_1 (for the main Turing computation), $-v_2$, $-v_1$, $v_2 - v_1$, v_2, v_1 and $v_1 - v_2$ for the sides of the annuli. Each vector v from this finite set is represented by a finite word of moves in $m_v \in \{a, b\}^*$ where a and b are two consecutive vectors in N_H (ordered by their angle). To v we associate $v^* = a - b$ which is used as a normal vector. If v is colinear to a base vector (*i.e.* $v = a^k$ for $a \in N_H$), we choose $v^* \in N_H$ so that $v + v^* \in N_H$. If T_v is the set of positions reached by the moves m_v, then T_v tiles the plane by translations $\mathbb{Z}v + \mathbb{Z}v^*$ (see Figure 15). In the same way as already explained for snakes following irregular path in the construction of Theorem 4.2, we can have zigzag transducer doing arbitrary computations while following the periodic path m_v^ω for its temporal direction and using v^* (or $-v^*$) for its spatial direction. The situation here is actually simpler because the turedo's head knows the

next move (encoded in the tape) when arriving at either extremety of zigzags. In particular it is possible to make constant width zigzags (contrary to Figure 9 where obstacles cannot be anticipated and induce small changes in the width of zigzags). This is the behavior of the turedo in the forward phases to fill with density 1 the sides of a annulus layer. The Turing computation is implemented by a straightforward adaptation of the case where the time/space directions are vectors of N_H.

To complete the technical description of the turedo, it is sufficient to describe how to change the pair of directions (v, v^*) of a zigzag to another one, either at the beginning of the annulus filling routine when leaving the Turing direction v_1 and starting to fill a side of the annulus, or between two forward phases to fill two successive sides. Let say the direction change is from (u, u^*) to (v, v^*). The situation is easy when $u^* = v^*$ since the zigzag movement can go on without interruption and it is only the sequence of shifts that changes from m_u to m_v. When $u^* \neq v^*$, we can suppose $u = a^{m_u}b^{n_u}$ and $v = c^{m_v}d^{n_v}$ where $(a, b, c, d) \in N_H$ are ordered counter-clockwise and $a \neq d$. The direction change started with head position z with tape extending in direction u^* is done in three phases (see Figure 15):

- go on with zigzag tape in direction u^* and use the sequence of moves a^w; while doing this copy the relevant content of the tape in the starting situation onto the segment from z to $z + wa$;
- from $z + wa$ move back following the border in $-a$ direction until reaching $z + v^*$ and copy along the way the data to expose it again; then start a zigzag triangle to copy the data on the segment of the same length starting at z but rotated by $\pi/3$; repeat this rotation if needed until the data is copied on segment from z to $z + wv^*$;
- from there, start a zigzag progressing with a periodic sequence of moves m_v and using space direction v^*, and progressively increase its width until it reaches the correct one (one extremity of the zigzags is on the inner side of the annulus layer, the other is on the outer side).

This procedure misses at most $O(\log(w)^2)$ positions in the layer as required. When finishing a layer, the turedo's head is back to the Turing computation zone. It needs to change a last time its zigzag direction, but this last direction change is simple because no data as to be conserved since all the relevant information is already present on the Turing tape, ready to resume the computation. Before starting the next layer, Turing zigzags are executed to move the head at the correct position to start again the layer filling process: precisely, if z was the head position at the beginning of the layer filling process, it needs to reach position $z + wv_1$ and, noting $m_{v_1} = a^k b^l$, the sequence of moves a^w as already been done at the beginning of the layer filling process, it remains to move according to the sequence $a^{(k-1)w}b^{lw}$.

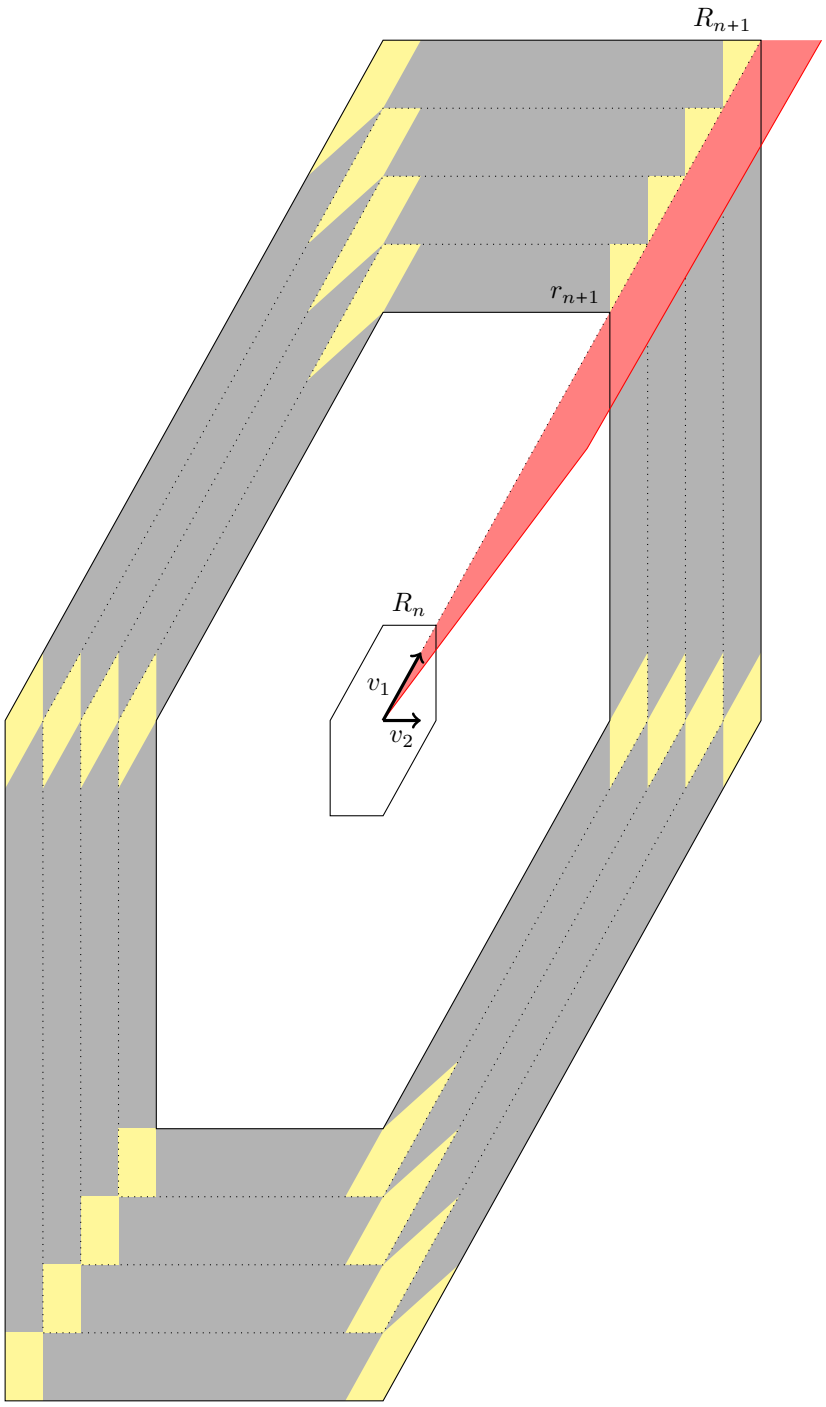


Figure 14 Overview of a run of the turedo to produce an approximate sequence of annuli in the sense of Lemma H.1. In red the Turing computation zone. In yellow the direction change error zone of each layer of the annulus filling routine (each yellow polygon is of area $O(\log(R_{n+1})^2)$). The part in gray is filled with density 1.

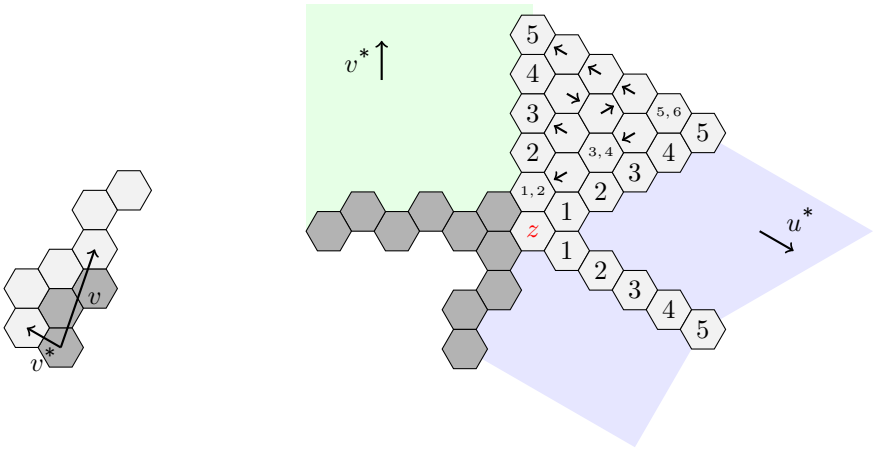


Figure 15 Geometrical details of the turedo's construction: on the left, the tiling by T_v with $m_v = \overrightarrow{\text{N}}\,\overrightarrow{\text{N}}\,\overrightarrow{\text{NE}}$ and $v^* = \overrightarrow{\text{NW}}$; on the right, an example of change of direction from u with $m_u = \overrightarrow{\text{N}}\,\overrightarrow{\text{NE}}$ to v with $m_v = \overrightarrow{\text{SW}}\,\overrightarrow{\text{NW}}$; in light blue the zone where the space direction of zizags is u^*, in light green the one where space direction is v^*; numbered cells indicates how the initial tape content is copied through the direction change process.

I Achieving density 1: a delay-3 oritatami filling the plane

Surprisingly enough, simple ideas like "one single bead type attracting itself" do not fill the plane because they either lead to non-determinism or prefer to fold upon itselves rather than expanding towards new areas. It follows that we had to implement "some intelligence" to obtain an oritatami that fills every position of the plane, namely to implement a counter-clockwise search. We could not use our turedo simulation since the macrocell cannot be fully filled (the pockets cannot be fully filled). So we designed a dedicated oritatami as shown in Fig. 16 and 17. It consists of two parts that always have the same external contour, but not the same orientation: the "crib" (in blue) which orient the next part towards the next counter-clockwise cell, and the "baby" (in red) which always fold the same way to exit where the "crib" points to. The principle for the crib is to adopt two different configurations depending on whether the next free counter-clockwise cell is to the NW or to the SW: by defaults C4 and C5 are attracted to C0 and fold backwards building the crib in the "upright" position, pointing to exit to the first CCW cell; however, if the first CCW is occupied, then C4 and C5 are attracted by it (to B10, B12, B15, B23, B25, or B32) and the crib is built in its "lying down" position, pointing towards the second CCW cell (see Fig.16). The baby just folds the same way in both cases and exits to the next cell pointed by the crib.

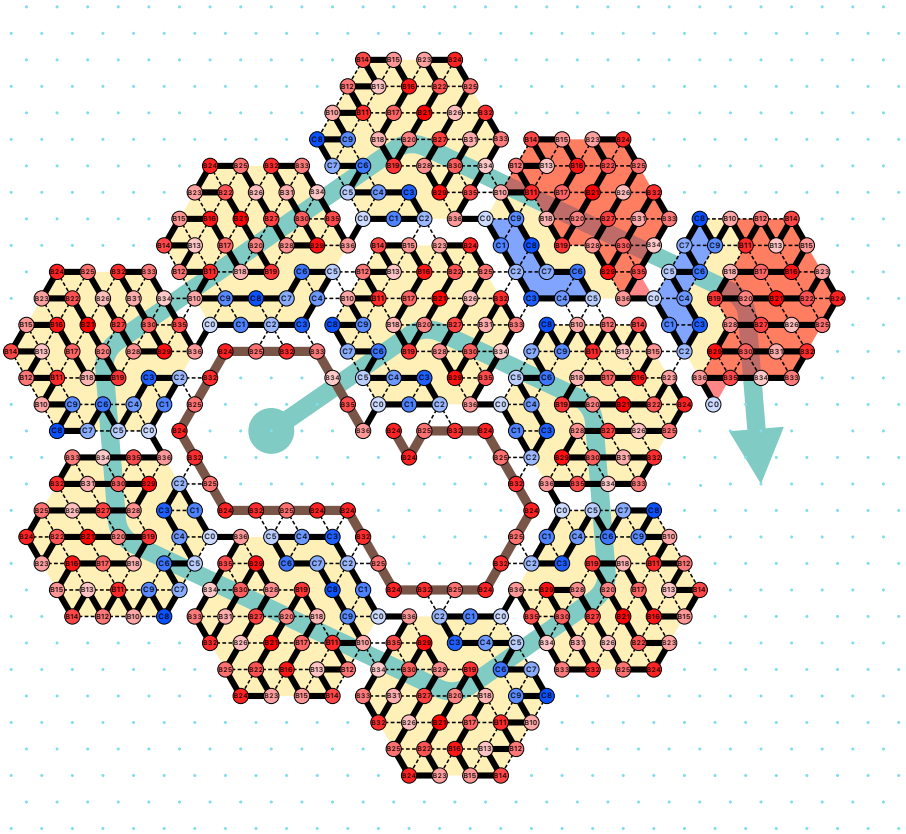


Figure 16 A delay-3 oritatami filling the plane: the crib and the baby are highlighted in blue and red respectively. The two cribs configuration "lying down" (first) and "upright" (second) are highlighted in blue on the top.

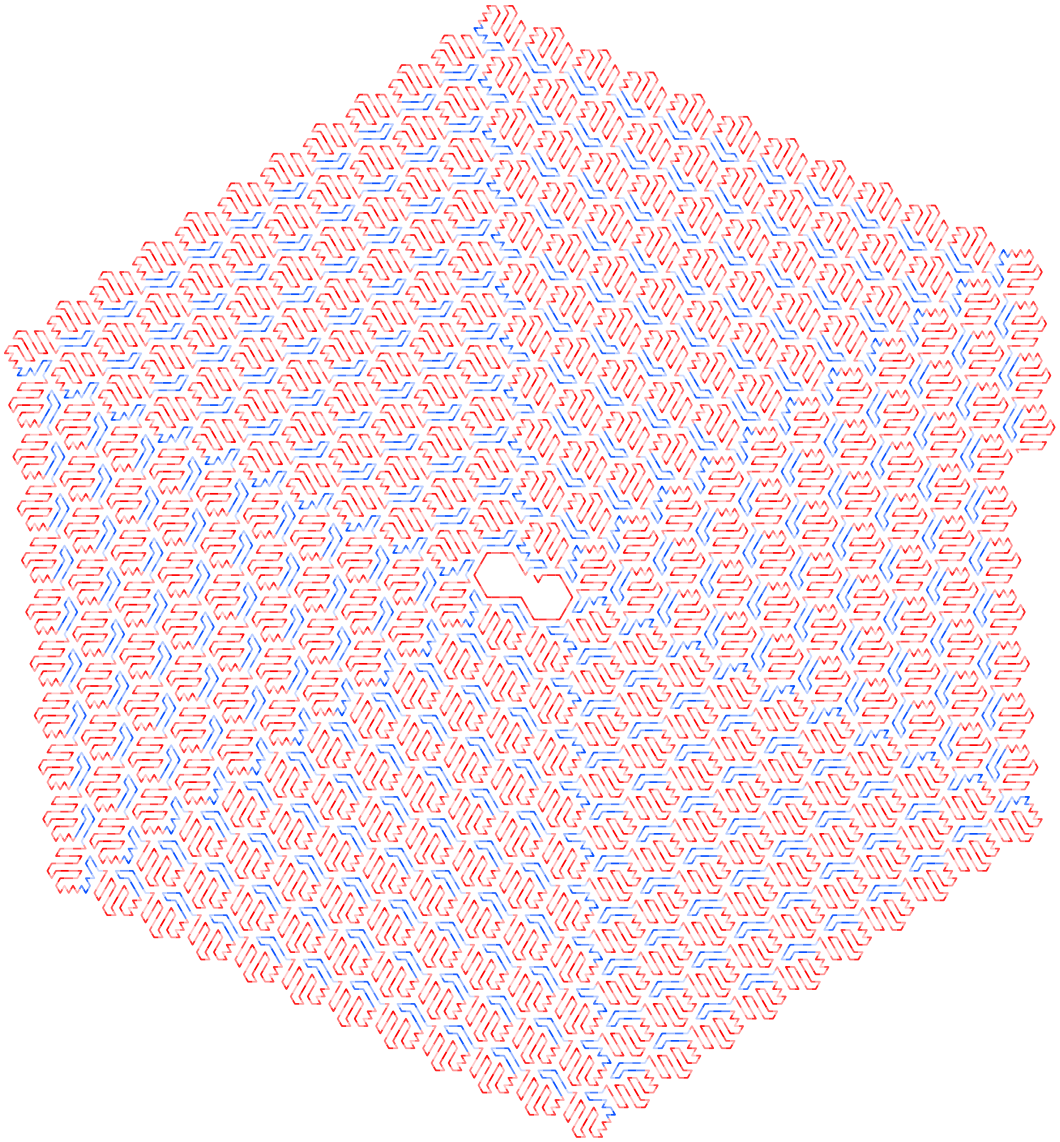


Figure 17 An execution of the delay-3 oritatami filling the plane from a seed consisting of two cells at the center (not filled for clarity).

Full description of the oritatami. The transcript consists is periodic of period C0,...,C9,B10,...,B36. The cells are hexagons of radius 3.

B10 💔 B12,B34,B35,C2,C4,C5,C8
B11 💔 B13,B17,B18,B31,C9
B12 💔 B10,B14,C2,C5
B13 💔 B11,B15,B16
B14 💔 B12,C3
B15 💔 B13,C2,C4,C5
B16 💔 B13,B22,B23
B17 💔 B11,B21
B18 💔 B11,B20
B19 💔 B28
B20 💔 B18,B27
B21 💔 B17,B26
B22 💔 B16,B24,B25
B23 💔 B16,C2,C5
B24 💔 B22,C3
B25 💔 B22,C2,C4,C5
B26 💔 B21,B32
B27 💔 B20,B31
B28 💔 B19,B30
B29 💔 B35,B36
B30 💔 B28,B34,B35
B31 💔 B11,B27,B33
B32 💔 B26,C2,C5
B33 💔 B31,C3
B34 💔 B10,B30
B35 💔 B10,B29,B30
B36 💔 B29
C0 💔 C4,C5,C9
C1 💔 C3,C9
C2 💔 B10,B12,B15,B23,B25,B32,C7,C8
C3 💔 B14,B24,B33,C1
C4 💔 B10,B15,B25,C0,C6
C5 💔 B10,B12,B15,B23,B25,B32,C0,C7
C6 💔 C4,C9
C7 💔 C2,C5,C9
C8 💔 B10,C2
C9 💔 B11,C0,C1,C6,C7

J The oritatami modules

In this section, we present the exact design of each module involved in the Turedo simulation by delay 3 oritatami systems.

J.1 Notations

for two integers $x \geqslant 0$ and $y \geqslant 1$

- x.nextMultiple(of: y) = $y\lceil x/y \rceil$ is the least multiple of y larger or equal to x
- x.complement(to: y) = $y\lceil x/y \rceil - x$ so that $x + x$.complement(to: y) = x.nextMultiple(of: y)

In the figures, the numbers in the same color as a given layer refer to the *lengths* of the corresponding segments of the layer. Black numbers refer to distances.

J.2 Folding meter and Pocket

A *n-folding meter* is a $4n$-periodic transcript with period Φ of the form:

$$\Phi = \mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2, \mathbf{p}_3, \ldots, \mathbf{p}_{n-1}, \mathbf{b}_n, \mathbf{b}_{n+1}, \mathbf{b}_{n+2}, \mathbf{q}_{n+3}, \ldots, \mathbf{q}_{2n-1},$$
$$\mathbf{t}_{2n+0}, \mathbf{t}_{2n+1}, \mathbf{t}_{2n+2}, \mathbf{p}_{2n+3}, \ldots, \mathbf{p}_{3n-1}, \mathbf{b}_{3n}, \mathbf{b}_{3n+1}, \mathbf{b}_{3n+2}, \mathbf{q}_{3n+3}, \ldots, \mathbf{q}_{4n-1}$$

where the letters **t** and **b** stand for *top* and *bottom*. Indeed, the internal interactions:

$$\underbrace{\Phi_i \heartsuit \Phi_{-i-1}, \quad \Phi_i \heartsuit \Phi_{-i-2},}_{\text{Going down}} \quad \underbrace{\Phi_{n+i} \heartsuit \Phi_{n-i}, \quad \text{and } \Phi_{n+i} \heartsuit \Phi_{n-i-1}}_{\text{Going up}} \quad \text{for all } i$$

ensures that it will either (see Fig. 18):

- follow a border if every bead sticks to the beads on the border;
- or fold upon itself in the manner of the "folding meter" tool, when entering into a *pocket* such as the pink area in Fig. 18, where the bead $\mathbf{P}_1$ at the bottom, does not attract the **b**-beads and as a consequence kicks the **b**-beads up and initiates the switchback folding between the **b**-beads, at the bottom, and the **t**-beads at the top. The switchback folding ends when the folding meter reaches the end of the pocket. It then resumes following the border.

As the bonds inside the switchbacks of a n-folding meter are strong, this switchback can flatten sophisticated interactions inside **p**-beads or **q**-beads of the n-folding meter as long as they do not involve more than 3 bonds. This allows us to hide or expose on-demand specific behaviors depending of the context: a specific behavior will happen only if the n-folding meter does not stick to the border or is not in switchback form into a pocket. This is how we manage to hide the transition table into pockets and to make it happen only at specific places.

In this article $n = 26$. Note that the folding meter is essentially $2n$-periodic as the $2n$-period is repeated twice only to prevent unwanted interactions when in switchback form. This is why everywhere in the paper the true unit of length is an half-period of the folding meter and not a full period. Furthermore every bead type $\mathsf{R}i$ in a folding meter R behaves the same as the beadtype $\mathsf{R}(i + 2n) = \mathsf{R}(i + 52)$. For this reason, we will adopt the following notation: given a folding meter R, $\mathsf{R}[\![i]\!]$ will refer to either bead types $\mathsf{R}i$ or $\mathsf{R}i + 2n$; for instance $\mathsf{R}[\![12]\!]$ refers to both R12 and R64.

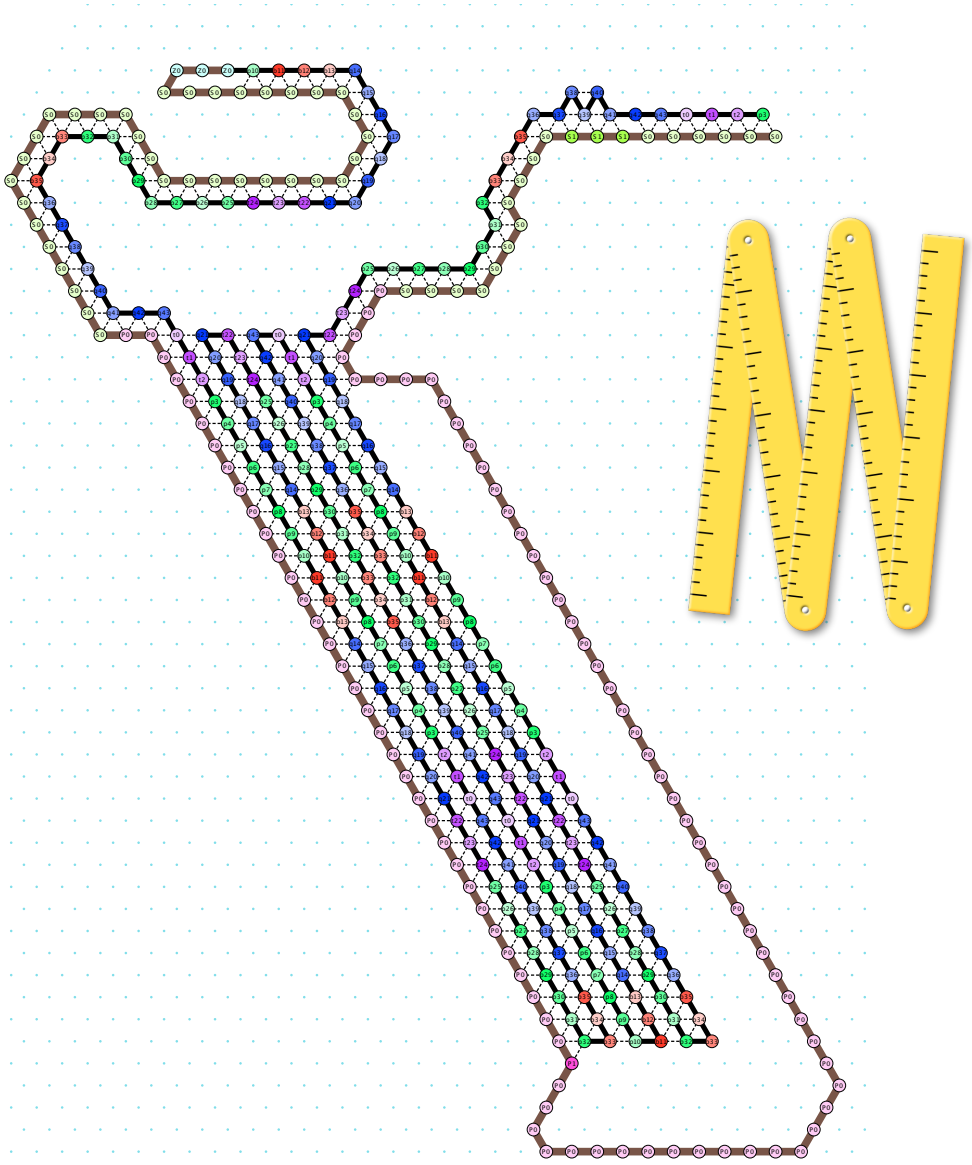


Figure 18 Example of a folding meter and pocket with $n = 11$, and of the tool which inspired its name. When along the light green border, it is flat; when in the pink pocket, it folds upon itself in a very compact form; when along the darker green border, it reveals a secret specific shape (two ears).

J.3 Multi-layer interactions

n-Folding meters have another welcome feature: they can be layered on top of each other in opposite directions. As long as they stick with two bonds to the lower layer, they will behave just as expected.

Our macrocell consists in 3 layers: read, write and exit, folding one after the other on top of the previous one. When the scaffold of the macrocell makes turns, the three layers shift with respect to each other. Fig. 19 illustrates the only possible offsets between the layers in all of our designs:

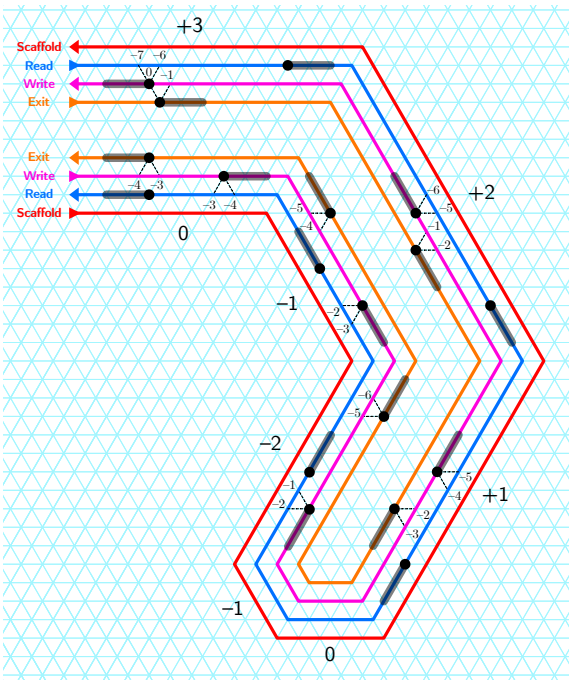


Figure 19 Layer offsets for every considered path orientation.

If $t \in \{-2, \ldots, +3\}$ denotes the counter-clockwise orientation of the scaffold border we get that the binding must be:

- for Write→Read: $(0, -3 - t)$ and $(0, -4 - t)$
- for Exit→Write: bond offsets are $(0, -4 + t)$ and $(0, -3 + t)$.

Now, the observed offsets between layers in the different modules (see the next sections) are:

Offsets	Read/Write	Write/Exit	Concerned layers	
Delay	$-1\ldots+1$	$-1\ldots+1$		
Write pocket	$-1\ldots+3$	0	R1/W2, R1/W12, R2/W12, R2/W1	W2/X1, W12/X1, W12/X2, W1/X2
Write module	$0; n-1\ldots n+1$ (specific)	$-1\ldots+1$	R2/W1	W1/X2
Read Pocket + intermediate delay	$0; n-1\ldots n+1$	$-1\ldots+1$	R1/W2, R2/W12, R1/W12	W2/X1, W12/X1, W12/X2
Exit interchange	$-2\ldots+1$	$-2\ldots+1$	R2/W12, R1/W12	W12/X12
Exit pocket	$-2\ldots+2$	$-2\ldots+2$	R1/W2	W2/X1
Uturn pocket	$0\ldots1$	—	R1/W2	

As a consequence:

Offsets	
R1/W2	$-2\ldots+3$
R1/W12	$-2\ldots+3$
R2/W12	$-2\ldots+3$
R2/W1	$-1\ldots+3$

Offsets	
W2/X1	$-2\ldots+2$
W12/X1	$-1\ldots+1$
W12/X2	$-1\ldots+1$
W1/X2	$-1\ldots+1$
W12/X12	$-2\ldots+1$

This defines the binding attractions between every pair of layers. Note that all of them are local, bounded to beads within a range of at most $-2..+3$ indices, there are no long-range interaction between different layers.

J.4 Transcript

The transcript of this system is periodic, and one period folds into one macrocell. Its period can be divided semantically as:

Scaffold · Read · Write · SpeedBump · Exit

Scaffold hardcodes a skeleton of the macrocell and folds into it clockwise. Read goes around the skeleton counterclockwise while reading inputs from adjacent macrocells and being shifted by an offset accordingly. Write goes around the read layer cw. and write outputs according to the offset. SpeedBump absorbs the offset. Finally, Exit goes around the write layer ccw. up until the macrocell's side on which an "exit signal" is coded. For ease in implementation, these five factors share no bead type.

Read is a repetition of n-folding meters, and so are Write and Exit at least macroscopically. Each of them is fundamentally bi-colored[6] such that the left half of macrocell's sides is painted in one color and the right half is in the other. Interchange modules, located at every corner and in the middle of every side (see Sect. J.16 and J.17), enable the even coloring no matter how large an offset gets as inputs are being read and the read and write layers slide accordingly. In the absence of rules to let beads in different colors bind, the resulting colored macrocells never interact as long as they are center-aligned face-to-face. A specific bead on the read layer certainly needs to be capable of probing bit-encoding beads, but these facing beads vary in color. Rules for this bit-reading are the only exception to the principle of non-intervention across colors. In order for macrocells not to interfere otherwise, the system keeps them far enough away from each other anywhere but the bit-reading sites by utilizing step-up and step-down modules (Sect. J.14).

Read, Write, and Exit consist of n-folding meters so that they can be layered one after another as explained in Sect. J.3. Furthermore, a specific geometry hardcoded in Scaffold and some extra rules enable them to interlock with each other for some functional purposes. These functional modules shall be explained in the rest of this section along with their geometry and extra rules. Note that some of the modules need variants of n-folding meters for Write and Exit which play a functional role at a designated site in the module but behave exactly as their original anywhere else.

J.4.1 Bead types

These four layers and speedbump involve the following bead types, respectively:

[6] An auxiliary third color is employed so as for n-folding meters of different colors not to be next to each other along the transcript. Functional roles of this color will be explained in Sect. J.16 and J.17.

Scaffold	S0..7, S10..17, S20..22, eo0..3, ea0..3, oo0..3, oa0..3, Ci0..3, Ci10..13, Co0..1, Co10..11, F0..5, Ex0..3, J0..23, J36..43, B0, T0;
Read1	rt0..2, rp3..25, rb26..28, rq29..51, rt52..54, rp55..77, rb78..80, rq81..103;
Read12	#t0..2, #p3..25, #b26..28, #q29..51, #t52..54, #p55..77, #b78..80, #q81..103;
Read2	Rt0..2, Rp3..25, Rb26..28, Rq29..51, Rt52..54, Rp55..77, Rb78..80, Rq81..103;
Write1	wt0..2, wp3..25, wb26..28, wq29..51, wt52..54, wp55..77, wb78..80, wq81..103, 1p3..18, 1p55..70, 𝍐p3..18, 𝍐p55..70, 0p3..18, 0p55..70;
Write12	@t0..2, @p3..25, @b26..28, @q29..51, @t52..54, @p55..77, @b78..80, @q81..103;
Write2	Wt0..2, Wp3..25, Wb26..28, Wq29..51, Wt52..54, Wp55..77, Wb78..80, Wq81..103, ⍉p24, ⍉p76;
Speedbump (scaffold)	ψ0..2, β0..17, θ0..3, γ0
Speedbump (layer)	Λ0..2, λ0, χ0..2, ϕ0..5
Exit1	xt0..2, xp3..25, xb26..28, xq29..51, xt52..54, xp55..77, xb78..80, xq81..103, Gp4..16, Gp56..68, ⍉b26..27, ⍉b78..79
Exit12	#t0..2, #p3..25, #b26..28, #q29..51, #t52..54, #p55..77, #b78..80, #q81..103,
Exit2	Xt0..2, Xp3..25, Xb26..28, Xq29..51, Xt52..54, Xp55..77, Xb78..80, Xq81..103, Gp4..16, Gp56..68, ☆q29..35, ★q36..37, ☆q38..46, ☆q81..87, ★q88..89, ☆q90..98;

J.5 Scaffold

This section presents the beadtypes used to build the scaffold. The scaffold is hardcoded at the beginning of periods of the transcript as an instance of a delay 3 oritatami system developed for this turedo simulation. This system, which we call *scaffold maker*, provides an oritatami system with a scaffold of intricate geometry, on which computation is to take place. It mainly consists of two modules that fold into line segments and turns, respectively. Arbitrary beads of the resulting scaffold can be specialized for the sake of computation above at the cost of extra bead types, and the turedo simulation involves such bead type modifications. Being irrelevant to the implementation of the scaffold maker, these modifications are explained not here but rather when the modules of the Turedo simulation are described. Note that, in the figures in the rest of this section, scaffold beads thus modified are highlighted in green (not the color of bead itself). Other two colors are also used to specify how scaffold beads attract: beads highlighted in blue are inert (no attraction) while those highlighted in orange are sticky, that is, attracting anyone in the succeeding layers. See Fig. 22, where read pocket is illustrated. Beads at the entrance of the pocket are modified in order to implement the function of reading a bit, thus they are highlighted in green. The rightside wall of the pocket is highlighted in orange, along which Read layer goes once it steps into this pocket (reading 0), while the lower half of the leftside wall is highlighted in blue, letting Read layer fold back upon itself into switchback.

J.5.1 Line-segment module (16 bead types: S0-15)

This module folds a transcript of period 8, say S0-S1-…-S7, in a zigzag manner into a straight line segment of width 2 according to the 8 rules which let Si bind with S$(i + 2 \bmod 8)$ for all $0 \leqslant i < 8$; see figures. One side of the resulting line segment is hence provided with beads of types S0, S2, S4, and S6, while the opposite side with S1, S3, S5, and S7; hence we call them the even and odd sides, respectively. Being implemented as a zigzag, the line segment is self-standing but is not stable enough not to be interrupted by another line segment in the proximity upon its folding. Therefore, this system duplicates this module using pairwise-distinct bead types S8, S9, …, S15; needless to say, it involves no rule to let these two variants interact. The third variant seems unnecessary for it is highly unlikely for three line segments to meet in the proximity.

J.5.2 Turn module (18 bead types: ea0-3, oa0-3, eo0-3, oo0-3, cb0, cb1), cushions (20 bead types: Ci0-7, Co0-3, F0-3, Ex0-3)

The system involves 4 types of turn module corresponding to the four possible turn types: towards the even/odd side of the preceding line segment acutely/obtusely. Their transcripts are ea0-1-2-3, oa0-1-2-3, eo0-1-2-3, and oo0-1-2-3, respectively, and their unique conformation is hardcoded in the rule set; all of them can be observed in Fig. 26.

A turn module is connected to a line-segment module not directly but via two beads as a cushion. There are eight bead types Ci0, Ci1, …, Ci7 (cushion-in) to concatenate a turn module of any kind to a line-segment module: Ci0-3 are for the line-segments of S0-7 while Ci4-7 are for those of S8-15. The cushion Ci$(k-1 \bmod 4)$-Cik is used if the preceding line-segment consists of S0-7 and ends with Sk or S$(k+4)$. Thus, a line segment can end arbitrarily. As for cushion-out, just the two fragments Co0-1 and Co2-3 are enough to concatenate a line segment of two possible kinds to a turn, on the assumption that line segments begin with S0 or S8. These cushions prevent turns from interacting with line segments. The rule set is so designed as not to let cushion-ins interact with cushion-outs, and hence, any line segment of

length 3 or shorter cannot be implemented by simply combining line segments, cushions, and turns in the straightforward manner; we shall explain how to implement them by utilizing a u-turn when joints are explained.

It is quite useful to introduce two fragments F0-1 and F2-3 as respective variants of Co0-1 and Co2-3 in order to flip the succeeding line segment. Oritatami computations require few intricate encodings on the scaffold. It is hence almost always enough for a bead on the scaffold to attract all or nothing, and this "all-or-nothing" property can be even relaxed further for the line segment as a unit. Since each kind of turn module geometrically determines which side of the succeeding line segment to be faced towards the "reaction surface" (here, we suppose that only one side of a line segment is used), the system replaces Co0-1 by F0-1 (resp. Co2-3 by F2-3) to let the succeeding line segment begin rather with S1 (resp. S9).

The four bead types Ex0-3 are variants of cushion-out used to make sure that every module ends with S3 or S7.

Obtuse turns may need to "bump" for the sake of succeeding layers. Two bead types cb0, cb1 and local rerouting of the cushion-in and turn as Ci13-cb1-oo0-Ci10-oo1-3 from Ci13-Ci10-oo0-3 yield a bump. They serve exclusively for even-side turns and for odd-side turns.

J.5.3 Joints

Joints help the system to implement a succession of turns at short intervals. For example, a u-turn is implemented as illustrated in Fig. 21; note that this implementation requires only one special bead type J7. The succeeding line segments proceed along the preceding one, and hence, they are implemented by pairwise-distinct bead types. An important application of this u-turn is the implementation of a line segment of length 3 or shorter (in fact, of an arbitrary length) as a "difference" between two long-enough line segments. Note that no extra special bead type is needed here.

J.6 Layer ordering and pocket sizes

Figure 20 illustrates how the different layers Scalfold (S), then Read1/Read12/Read2 (R1/R12/R2), then Write1/Write12/Write2 (W1/W12/W2), then Speedbump1/Speedbump2 (SB1/SB2) and finally Exit1/Exit12/Exit2 (X1,X12,X2) succeed to each other around the macrocell. The layers R12, W12, X12 serve as a cushion to cancel any need of interactions between the layers R1/R2, W1/W2 and X1/X2.

Figure 20 also displays the capacities of the various pockets in terms of half-period of folding meters ($2n$ beads). Note that the capacities of the write pockets are all equal to $\mathcal{W}$, the maximum shift, because they are all filled once every bit on every side is read; whereas the capacities of the read pocket are either equal to the weight w_{ij} of the bit read (when reading – highlighted in blue), or to the current sum of the weights of the bits already read (for the interchanges in the middle and in the corners – highlighted in grey). Regarding the exit layer, as it folds after the speedbump, the shift is now zero, and the only thing that dictates the size of the exit traps is the height of the exit pocket, that is $2k_{\text{exit}} + 1$. Note that the exit layer never goes beyond the last exit pocket on the NE side.

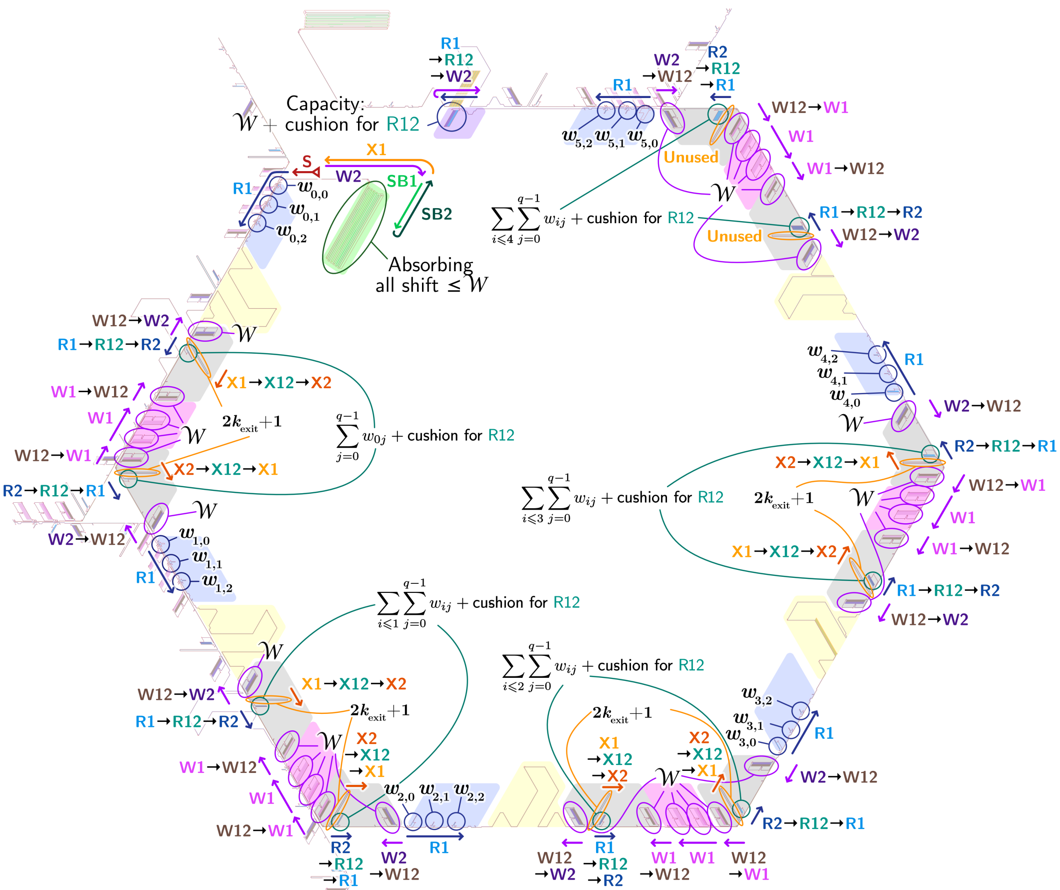


Figure 20 The order in which the layers succeed each other and the capacities of the various read, write and exit pockets. The ordering starts at the end of the scaffold layer (S) in the NW corner.

J.7 Read pocket

Read pocket operation. The primary purpose of the read pocket is namely to read a bit (0/1) and to create a shift in the the read layer by the amount equivalent to its size if the readlayer reads a 1. The read layer folds from right to left. When the read layer reaches the entrance (see Fig. 21), its "reading head", the beads r⟦36⟧, senses whether there is a 1 written on the adjacent cell at this location. If there is a 1, encoded by the presence of the pair of beads (⅄⟦10⟧, ⅄⟦12⟧) or (★⟦36⟧, ★⟦37⟧), then the reading head is attracted upwards, allowing the read layer to folding into a glider shape that will immediately escape the read pocket (Fig. 21). Otherwise, if there is a 0, encoded by the absence of these bead types at the expected location, the reading head is attracted by the pocket border downwards and the read layer ends up filling up the pocket entirely before leaving it (Fig. 22). This results in a shift forward of the read layer by an amount corresponding to the pocket capacity if and only if the bit written on the adjacent macro cell is 1.

Remark that this novel bit reading method, using a reading head, does not obstruct the way between adjacent cells unlike the method used in [21]; this allows the write and exit layers to pass and reach the exit at an arbitrary side. Note that this is the reason why our simulation uses delay 3.

Ease of design. Note finally that the interactions between the scaffold and the read layer are extremely simple: the only places where these interactions are carefully designed are at the entrance and at the end of the pocket (the three areas highlighted in green in Fig. 21), all the other interactions are either "attract-them-all" (the areas highlighted in yellow) or "attract-none-of-them" (the areas highlighted in blue). This demonstrates the simplicity of the folding meter/pocket concept.

Cushion. The 0/1-parameter addCushion is 1 when a read pocket is used inside the interchange blocks, where the color of Read layer changes between Read1 (r-type) and Read2 (R-type), and the extra capacity $2n(2k+1)$ allows to accommodate the cushion sequence Read12 (#-type) of this length to prevent beads of different colors to have to interact with each other inside the pocket (see sections J.6, J.16 and J.17). This is illustrated in Fig. 23.

Geometry. The pocket involves the other two parameters x and y; they are adjusted for the sake of n-folding meters as follows:

- The length of the read layer path from the last **B** before entering a read pocket to the first **T** after getting out of it should be equal to n modulo $2n$. Let $w' = w + \mathsf{addCushion}$. The length, when reading a 1, is $n+2nw'(2k+1)+\rho n+5+x+2+3+\rho n+3+2w'+x+2+5+1+17 = n + 2n(w'(2k+1)+\rho) + 2(w'+x+19)$; thus, x should be set as:

$$x = (w' + 19).\mathsf{complement}(\mathsf{to}:\ n).$$

- A read pocket is accompanied with two "pitfalls" to create the offset n between the read and write layers before reading and back after. In order to prevent the left pitfall from colliding with the pocket, y should be set as

$$y = \frac{(w' + \lfloor x/2 \rfloor + n).\mathsf{nextMultiple}(\mathsf{of}:\ n)}{n}$$

Capacity. The *capacity* of a read pocket is defined to be the difference in length between the paths taken by the transcript upon reading 1 and upon reading 0, and it is determined

by the parameters k, w, ρ with $\rho < 2k+1$, and $\mathsf{addCushion} \in \{0,1\}$ as

$$\mathsf{capacity} = 2n\left(w(2k+1) + \rho - 1 + \frac{(w+19).\mathsf{nextMultiple}(\mathsf{of}:\ n)}{n}\right).$$

Building a read pocket of a given capacity. Conversely, for the read pocket that is supposed to yield a shift of Δ $2n$-periods, thus we want its capacity to be Δ. Its parameters k, w, ρ can be computed as:

$$\begin{aligned} k &= \left\lceil \sqrt{1 + (8\,\Delta + 10)/n - 151/n^2} - n/4 - 1/4 \right\rceil \\ w &= \left\lfloor \frac{\Delta - 19/n + 1}{2k + 1 + 1/n} \right\rfloor \\ \rho &= \Delta - (w(2k+1) + (w + \mathsf{addCushion} + 19).\mathsf{nextMultiple}(\mathsf{of}:\ n)/n - 1) \end{aligned}$$

Since ρ must be smaller than $2k+1$, if ρ obtained as above does not satisfy this inequality, then x is modified as

$$x = (w + \mathsf{addCushion} + 19).\mathsf{complement}(\mathsf{to}:\ n) + \rho - 2k$$

and ρ can be set to $2k$ and y must be updated. The cases when $\Delta \leqslant 2$ are exceptional when these parameters should be set simply as $k = \rho = 0$ and $w = \Delta + 1$ when this pocket is used in the interchange block or $w = \Delta$ otherwise. It follows:

▶ **Proposition J.1.** *On input Δ, the algorithm above outputs parameters (k, w, ρ, x) such that the pocket swallows exactly Δ $2n$-periods of the read folding meter layer when reading a 1 with respect to reading a 0, and such that parameters k, w are $O(\sqrt{\Delta})$.*

Jumping over the entrance. For the read pocket to work, its entrance must have a minimum width. However, the defaults interactions between the read and write layers are too strong to allow the write to jump over it. For this purpose, we place on both side of the read pockets two *pitfalls* (Fig. 24) whose role is to desynchronize the read and write layers by a quarter of a period (zoom in Fig. 2). Normally (when Read and Write are in sync), p parts of Write layer is in front of q parts of Read layer, while q parts of Write layer is in front of p parts of Read layer. However, due to the sandwiching pitfalls, inside the reading block, p parts of these layers are face-to-face, and so are their q parts, and their interaction can be programmed especially so as for Write layer to jump over the read pockets.

Read interchange. The read pocket is also used to prevent read layers of adjacent macrocells from interfering with each other (see sections J.6, J.16 and J.17). The read layers are "bi-colored" and the rule set is designed so as for beads of different colors not to bind. Except those for actually reading bits, the macrocell is provided with read pockets in the halfway along every side and at every corner, in which read layer changes its color (type), no matter how large an offset has been accumulated so far. In the absence of any rule to let beads in read layers of the different types interact with each other, the two read layers of adjacent macrocells never interfere. The write pocket (Sect. J.8) and exit trap (Sect. J.9) serve the analogous purposes for write layer and exit layer, respectively.

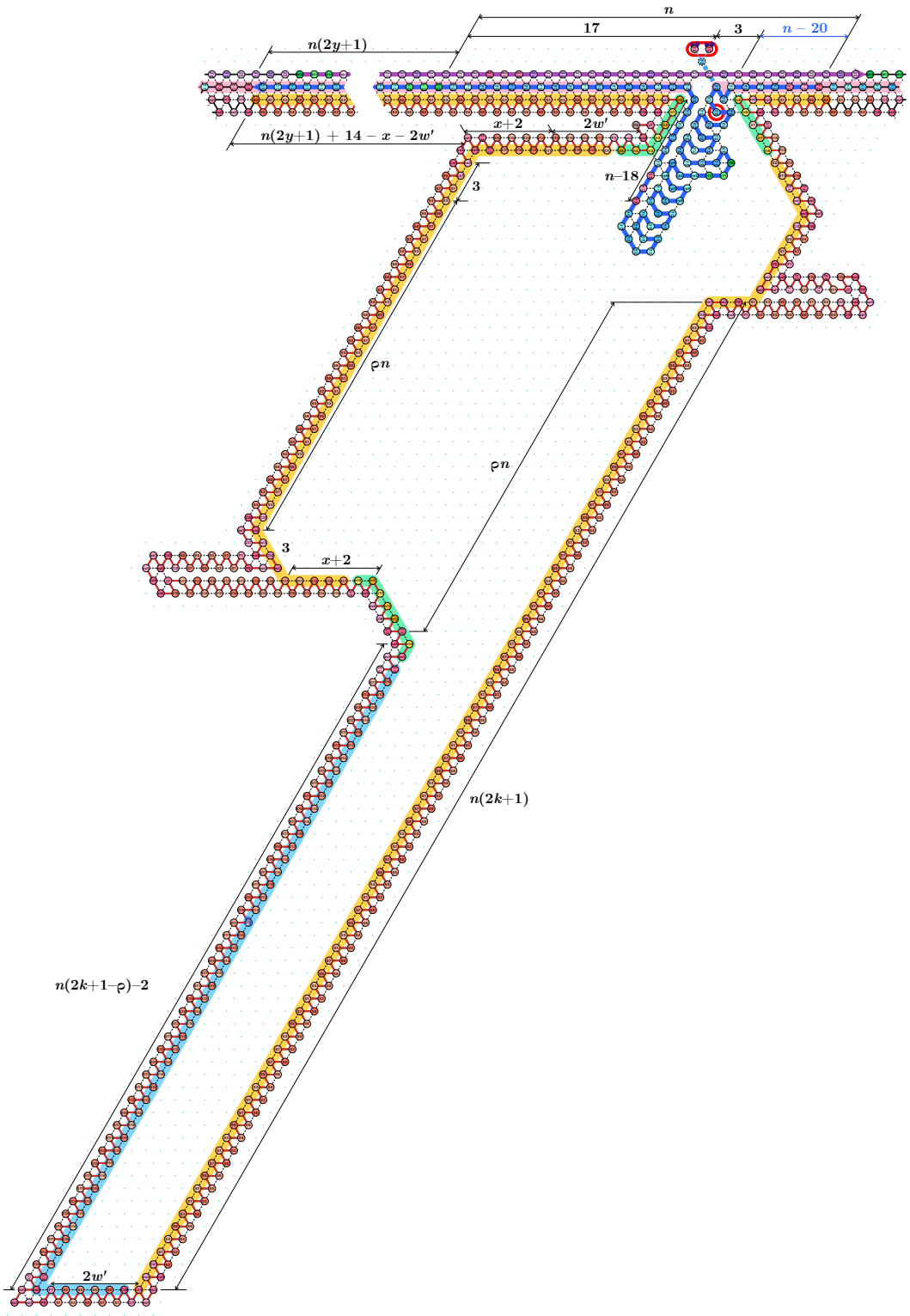


Figure 21 Read pocket reading 1. Recall that $w' = w +$ addCushion. Given w, parameters x and y must be adjusted so as to match the period of the folding meter.

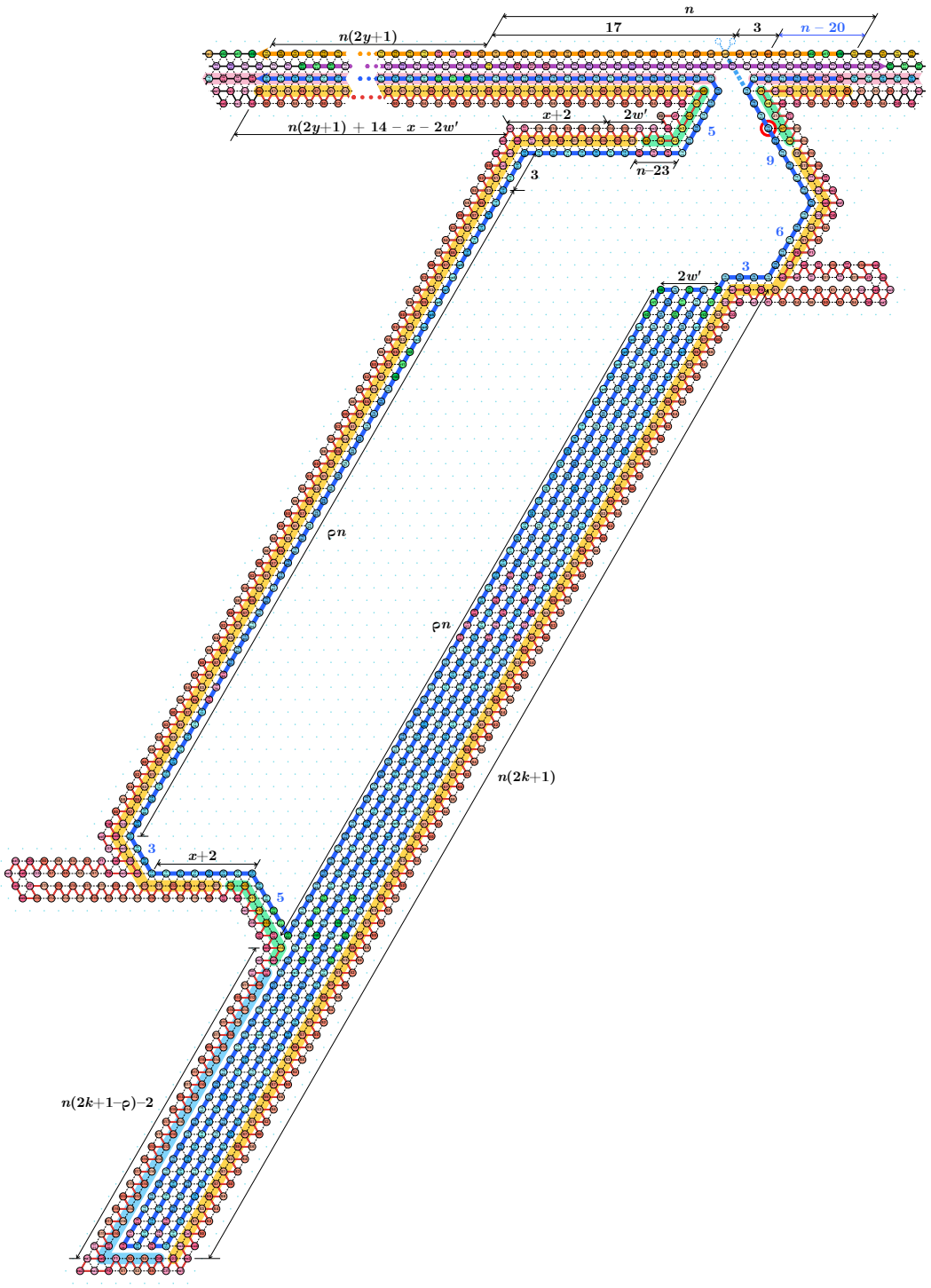


Figure 22 Read pocket reading 0. Recall that $w' = w + \mathsf{addCushion}$. Given w, parameters x and y must be adjusted so as to match the period of the folding meter.

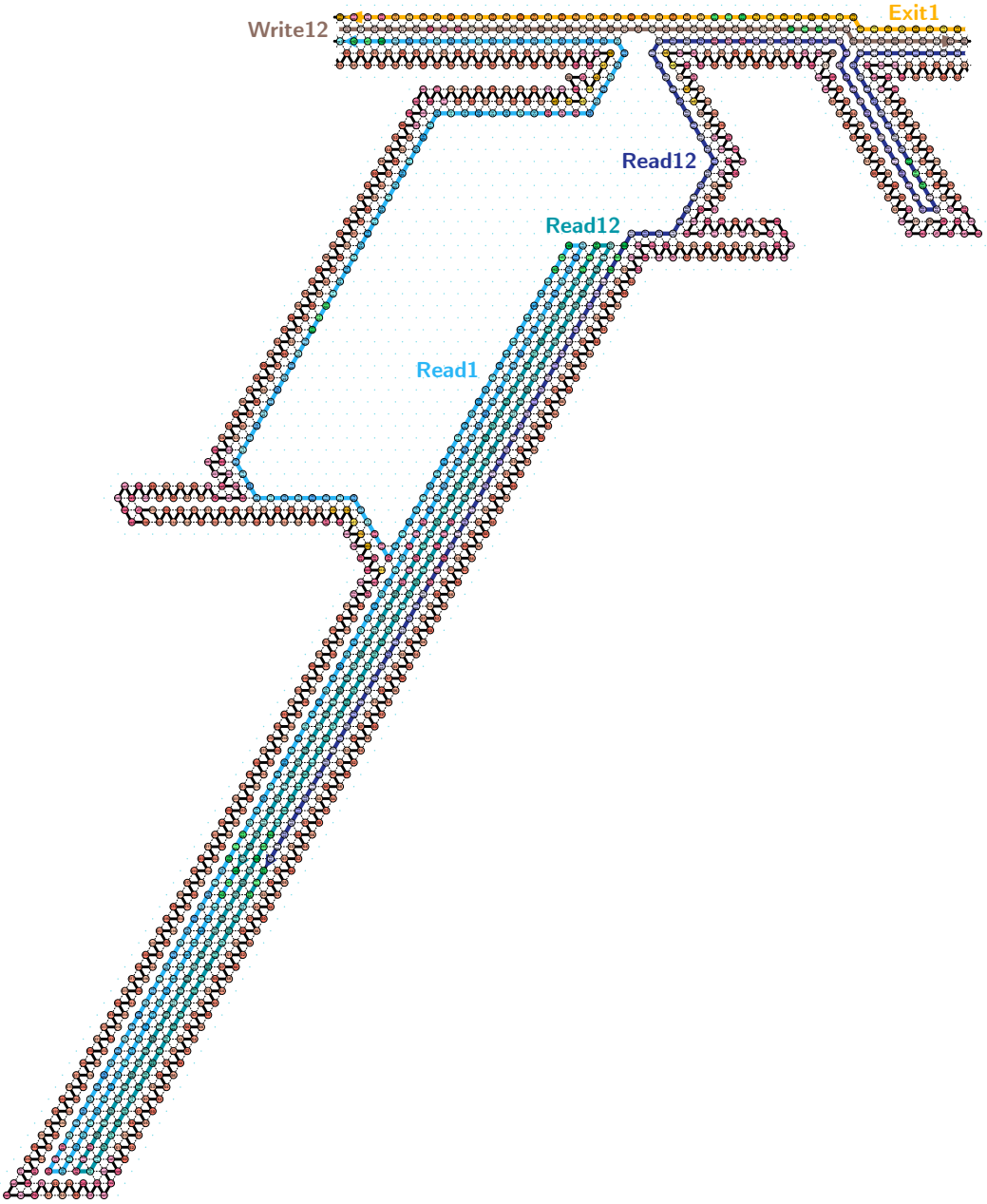
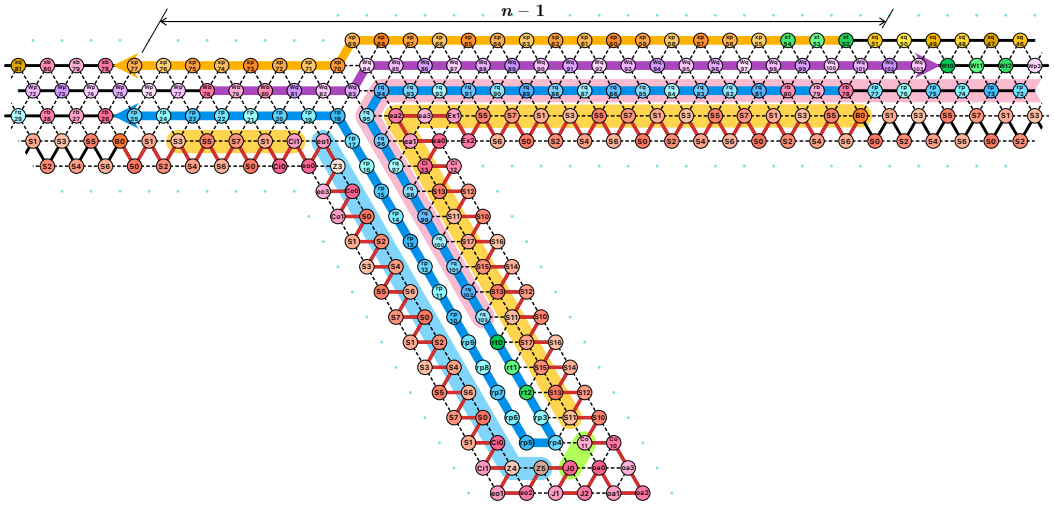
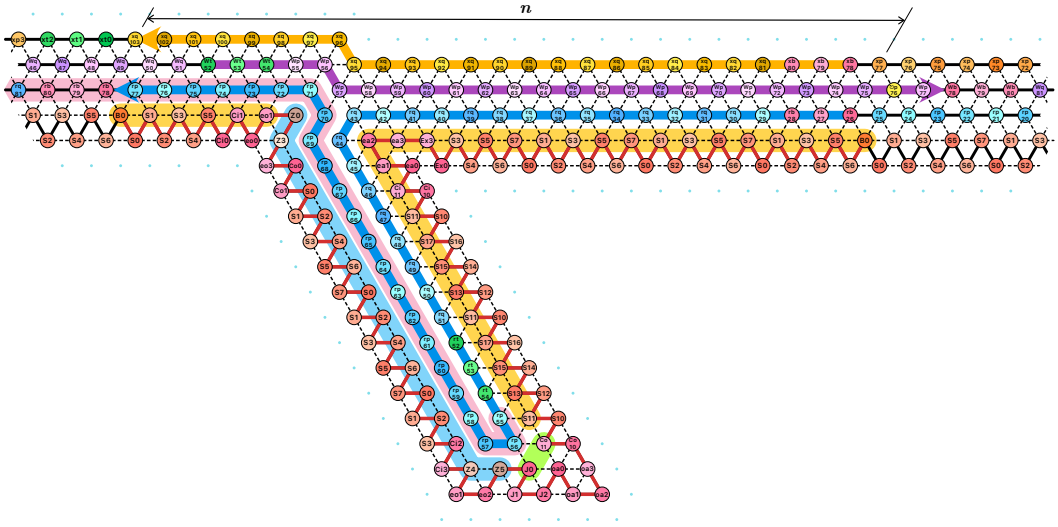


Figure 23 Read pocket in an interchange block whose width has been extended by 2 to accomodate the "cushion" layer Read12 insulating the layers Read1 and Read2 from each other. Note that the length of the insulating layer is precisely twice the height of the pocket, which is strictly enough to insulate the two layers inside the pocket whatever the shift is.



(a) *The left pitfall* dephases the Read and Write layers so that their p and q parts on both sides face each other (highlighted in pink). The resulting reduced attractions is allows for the Write layer to jump over the opening of the upcoming read pockets.



(b) *The right pitfall* resynchronizes the Read and Write layers.

Figure 24 Read pocket surrounding pitfalls.

J.8 Write module

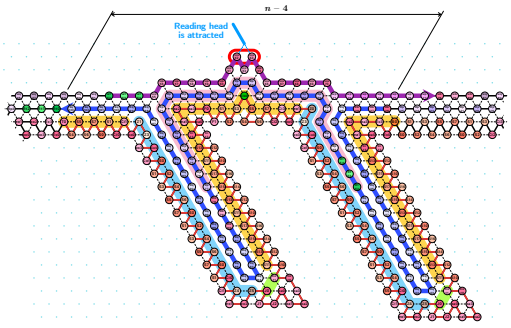
Every side of a macrocell is provided with q write modules, each of which is responsible for one of the q bits to be output along the side. This module places two beads of special type (circled in red in figures) at a designated readable site (Figs. 25a and 25b), corresponding to writing 1, or deliberately out of the site so that they cannot attract the reading head no matter what types they are (Figs. 25c and 25d), which has the same effect as writing 0.

Depending on which side to exit at a given input read, the exit layer may cover each write module or not; it does along all and only the sides ccw. prior to the side to exit. Each write module is equipped with a transition table that is encoded on the repetitions of $\mathcal{W}/2$ n-folding meters (2 entries per period of folding meter). Each table entry (of length $2n$) "knows" from the exit-direction-transition table whether this write module is to be covered by the exit layer or not, and encodes the bit 0/1 using different variants of n-folding meter so as to achieve the following behaviors of this module:

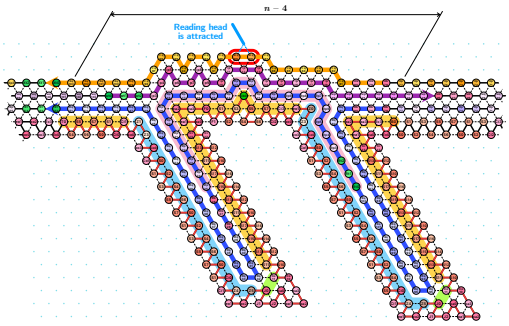
- In case the write layer is to be covered, two spikes are placed to the left (Fig. 25b) or right (Fig. 25d) of the hill; the exit layer folds from right to left, and it hits the brake before the hill if spikes are tp the right, sliding the special beads out of the readable site, equivalent to writing `0`.
- Otherwise, the bit 0/1 is encoded simply as of whether two big spikes are formed at the designated site (Fig. 25a) or not (Fig. 25c).

The write module is equipped with two pitfalls swallowing each an quarter of period of the read folding meter. When the write layer, folding rightward, reaches the left pitfall, it is desynchronised with the read layer by n: the p-parts of the read and write folding meters are facing each other, allowing the hardcoded pattern (with two ears) inside the p-part of the write to fold instead of been glued to the read layer: this allows to write the entry of the transition table at this precise location. The two layers are then immediately resynchronised by the second pitfall, preventing the other entries of the transition table to be written at improper locations.

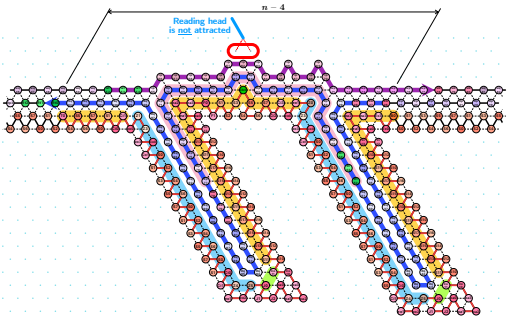
Along a side of a macrocell, q write modules and $q + 1$ write pockets (see Sect. J.9) of capacity $O(\mathcal{W})$ are placed alternately, and each of the q transition tables is stored compactly in the two write pockets that sandwiches the write module for the corresponding bit and slides between them.



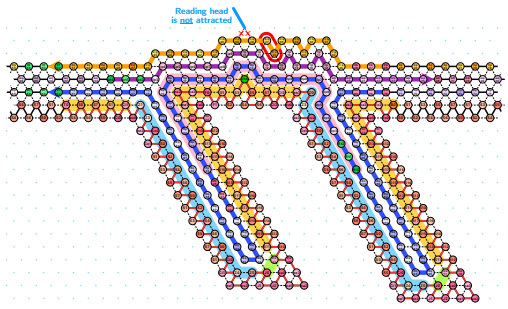
(a) *Write module – Top variant:* the write layer writes a `1` by forming two ears on the top of the module, with two active beads aligned with the reading head of the adjacent macrocell.



(b) *Write module – Left variant:* the write layer writes a `1` by forming two ears to the right of the module so that the active beads of the exit layer are aligned with the reading head of the adjacent macrocell.



(c) *Write module – Right variant 1:* on a side before the exit that will be taken later, the write layer writes a `0` by forming two ears to the right of the module, and as the exit layer will exit before reaching this position, the reading positions will stay empty, which will be interpreted as a `0` by the reading head of the adjacent macrocell.



(d) *Write module – Right variant 2:* the write layer writes a `0` by forming two ears to the left of the module so that the active beads of the exit layer are misaligned with the reading head of the adjacent macrocell.

Figure 25 The four variants of the Write module: **(a, b)** writing 1 and **(c, d)** writing 0.

J.9 Write pocket

Write pocket operation. Its primary purpose is to hide the unused entries of the transition tables. These pockets are placed in between the write modules so that only the entries to be written are exposed on the border, at the locations of the write modules, all the other are hidden in the write pockets. The write pocket is simply "coated" by the read layer. But, the write layer will enter it and fold into its compact switchback form, hiding away the $\mathcal{W}$ unused entries of each transition table (each encoded in half a period of the write folding meter). The exit layer will then pass over it, folding a hardcoded bridge to get across its entrance.

As announced earlier in Sect. J.7, the write pocket is also used to let rather the write layer flip its "color" to prevent interference between macrocells.

Write pocket design. This pocket differs from the read pocket in two ways: 1) as opposed to the read pocket, both layers read and write will enter the pocket unconditionally; 2) the read and write layers will entre the pocket from opposite directions; 2) we must design the pocket so that only the write layer fills the pocket: the read layer must leave the pocket intact for the write to fill. The write pocket is basically the read pocket without reading, that is, read and write layers always enter.

- The first condition implies that both sides of the read-layer-coated entrance will attract the write layer and for this reason we need to make it wider to cancel these unwanted attractions. This means that the exit layer will not be able to jump over the entrance. Fortunately, as the exit layer is never shifted, we can hardcode a glider bridge $\mathsf{G}[\![4..16]\!]$ in this layer at this precise location to solve this issue (see top of Fig. 26). Note that, as the exit layer must stay in sync with the underlying layers on both side of the pocket, the length of this bridge, that is the width of the entrance of the pocket, conditions the positions of the beginnings and ends of the folding meter periods in the pocket.
- The second condition implies that the shape of pocket must be somehow "reversible" this implies that the phase difference between the read and write layers even in sync might increase significantly. Fortunately the folding meter design is flexible enough and increasing n allows to widen the number of bearable phase difference between layers.
- The third condition is much more problematic as we cannot program the read layer as easily as the scaffold as 1) it is a folding meter with delicate structures hidden (such as the the read head glider) and 2) it will enter the write pocket with an uncontrollable shift, making it impossible to hardcode. We solve this problem by a) designing carefully the geometry of the pocket, and b) desynchronising the read and write layer at carefully chosen moments. Essentially, we will keep the read and write layers in sync where the scaffold was attracting in the read pocket (the parts highlighted in yellow parts in Fig. 21), and desynchronise them at the places where the scaffold did not attract the read layer at all in the read pocket (the parts highlighted in blue in Fig. 21). We still need to deal with the end of the switchback form which required special interactions with the scaffold in the read pocket (the parts highlighted in green in Fig. 21): as we will be using desynchronised phases that do not occur anywhere else, we will use these phases to program specific interactions between the read and write layers that will emulate the same behaviour. Again, thanks to the flexibility of the folding meter design, increasing n allows to create enough separated spaces in the layer sequences to implement all these specific behaviours.

We solve these issues using the design in Fig. 26. The main difference with the read pocket is that the scaffold is attracting the read layer everywhere, but in two pitfalls. These pitfalls desynchronize the read and write layers and are placed so as to sandwich the part

of the pocket that should not attract the write layer. The proper folding of the end of the switchback form of the write layer is accomplished by programming carefully the interactions between the read beads rp⟦24..25⟧, rb⟦26..29⟧, rq⟦29⟧ with the write beads @b/Wb⟦78..79⟧, @q/Wq⟦81..83⟧ which meet nowhere else. Note that also the lower bubble has to get away (rightwards and downwards) from the switchbacks of the write layer to avoid unwanted interactions, as opposed to the shape of the read pocket.

All these constraints contribute to the choice of $n = 26$ for the period of our folding meters.

Geometry. The pocket involves the other two parameters x, y, which are adjusted for the sake of n-folding meters as follows:

- The length of the write layer path from **T**/**B** to **T** inside the upper bubble should be equal to 0 modulo $2n$ if ρ is even or equal to n modulo $2n$ if ρ is odd. The length is $2(x + w + 17) + \rho n$; thus, x should be set as:

$$\mathbf{x} = (\mathbf{w} + 17).\mathsf{complement}(\mathsf{to}\colon n). \tag{1}$$

- The length of the read layer path from **T**/**B** to **B** inside the lower bubble should be equal to 0 modulo $2n$ if ρ is odd, or equal to n modulo $2n$ if ρ is even. The length is $2(w + y + 12) + (2k + 1 - \rho)n$; thus, y should be set as:

$$\mathbf{y} = (\mathbf{w} + 12).\mathsf{complement}(\mathsf{to}\colon n). \tag{2}$$

Lastly, in order for the top part of the pocket to end to the right of both the upper and lower bubbles, ℓ should be set as:

$$\ell = \frac{(2w + \max(x + 8, y - 3)).\mathsf{nextMultiple}(\mathsf{of}\colon 2n)}{2n} \tag{3}$$

Capacity. The *capacity* of a write pocket is defined to be the length of the path taken by the write layer from the rightmost **T** to the leftmost **T**, and it is determined by the three independent parameters k, w, ρ with $\rho < 2k + 1$, and one dependent parameter ℓ as:

$$\mathsf{capacity} = 2n((2k + 1)w + \rho) + 2(w + 17).\mathsf{nextMultiple}(\mathsf{of}\colon n) + 2n\ell.$$

Building a write pocket with a given capacity. Computing conversely these parameters from an expected capacity $2nL$ should take the length of the underlying read layer into account. Let us solve $\mathsf{capacity} \geqslant 2nL$ so as to minimize the Read layer length, that is asymptotically as k and w go to ∞: $4nk + 6w$. The ideal ratio is thus: $w \sim 2nk/3$. The parameter ℓ should be at least $(w + 2)/n$ in order for a write pocket not to collide with anything to its right. Plugging in these lower bounds into the formula above for capacity with $x = \rho = 0$, we get that we are looking for a value of k verifying:

$$2nk/3 \cdot (2n(2k + 1) + 2) + 2n + 4 \geqslant 2nL$$

Solving this equation gives:

$$k \geqslant k_0 = \frac{\sqrt{12Ln + n^2 + 4n - 224}}{4\,n} - \frac{1}{2n} - \frac{1}{4}$$

Now we set

$$w = \left\lfloor \frac{nL - 19}{n(2k + 1) + 2} \right\rfloor.$$

From this value of w, we can compute x, y, and ℓ according to the formulas (1), (2), and (3). We finally set

$$\rho = 2nL - \big(2n((2k+1)w+1) + 2(w+17).\mathsf{nextMultiple}(\mathsf{of}\colon\ n) + 2n\ell\big),$$

or set $\rho := 0$ if $\rho < 0$ above so that $\mathsf{capacity} \geqslant 2nL$ is ensured. Note that this process guarantees that $\mathsf{capacity} \leqslant 2n(L+2)$.

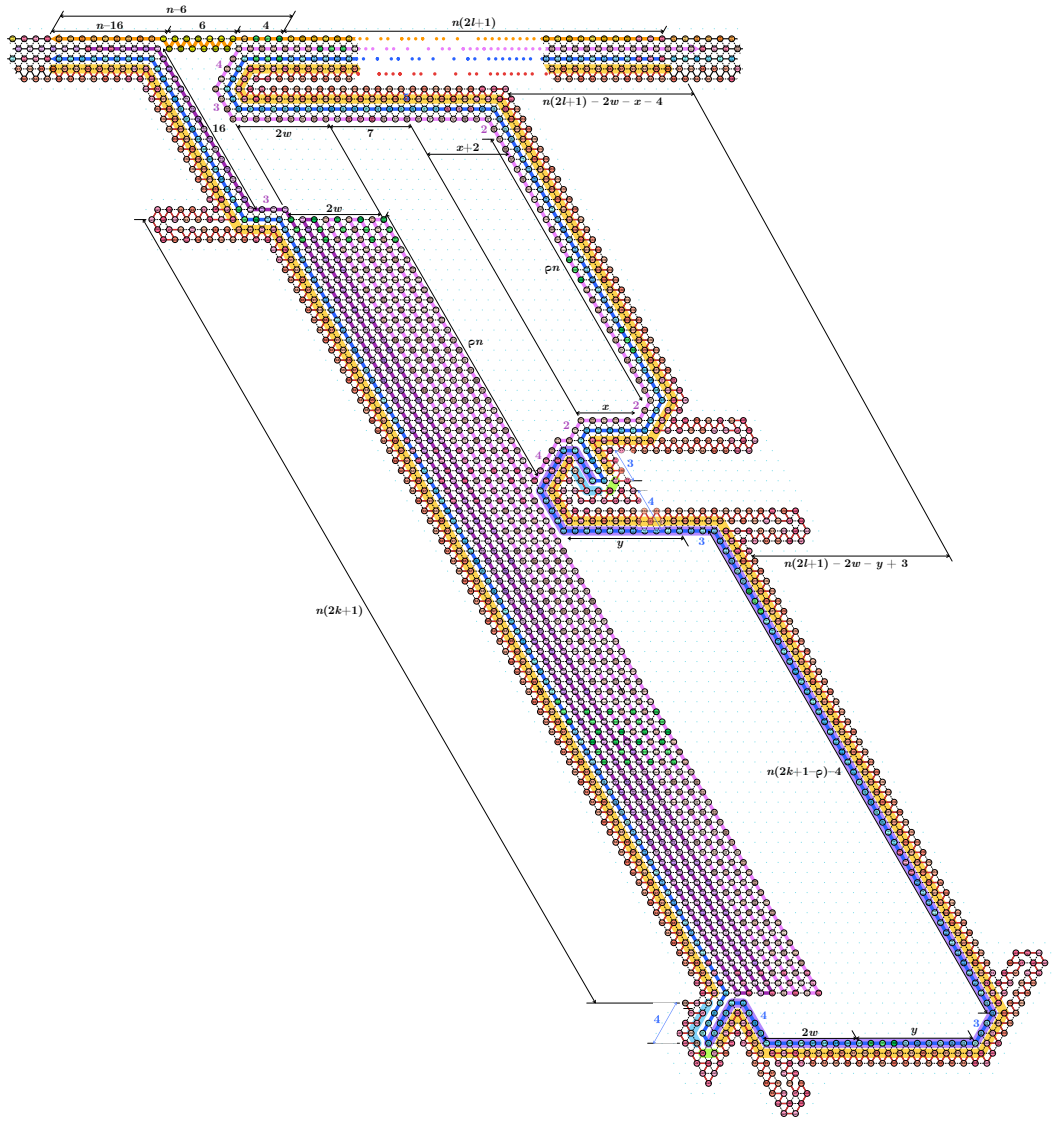


Figure 26 Write pocket. Given w, parameters x, y and l must be adjusted so as to match the period of the folding meter. The part of the read layer which is desynchronized with the write layer is highlighted in purple.

▶ **Proposition J.2.** *Given some $L \geqslant 0$, the algorithm above outputs parameters (k, w, ρ, x, y, ℓ) such that the box swallows at least L $2n$-periods of the write folding meter layer and such that parameters k, w, ℓ are $O(\sqrt{L})$.*

J.10 U-turn module

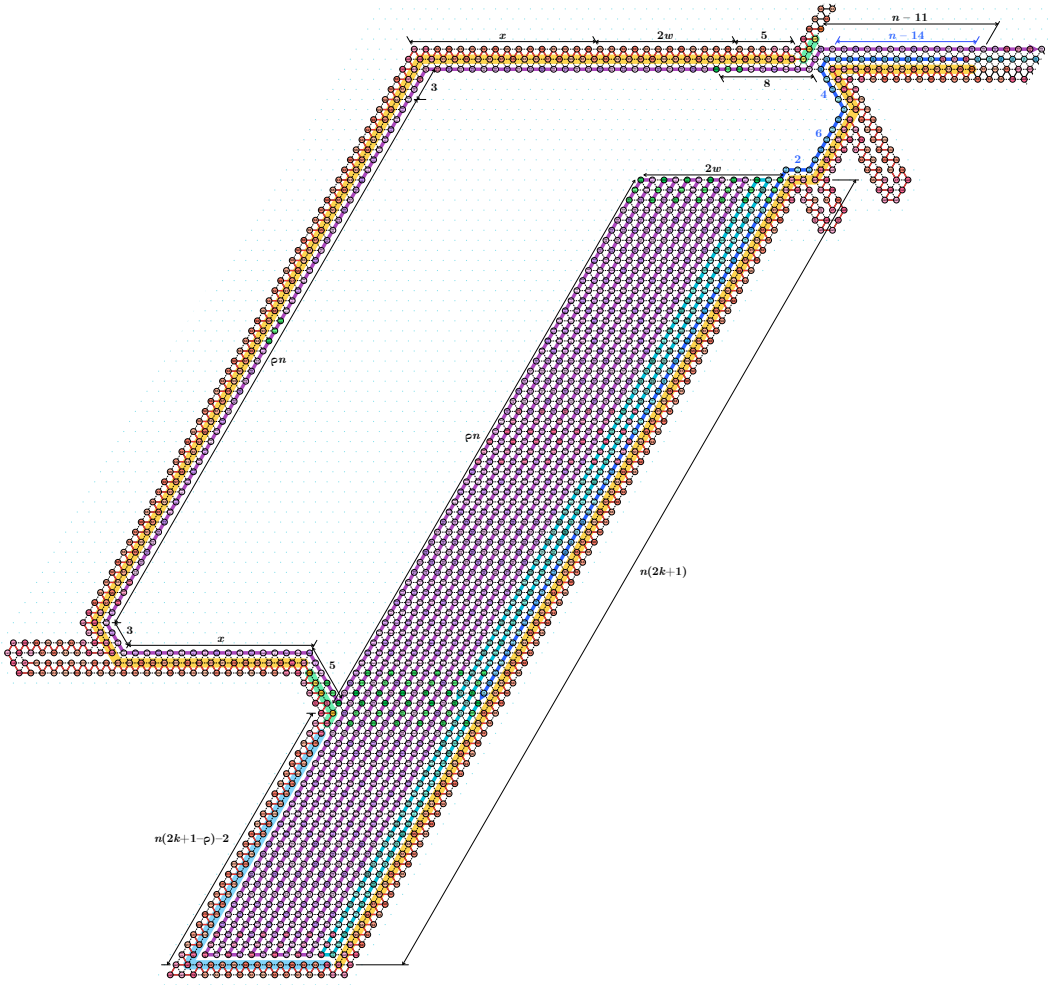


Figure 27 U-turn pocket. Given w, parameter x must be adjusted so as to match the period of the folding meter.

Inside this module, the transcript transitions from the read layer to the write layer. Between these layers is inserted a cushion fragment of length $2n(2k+1)$, which is long enough to guarantee that these two layers never interact inside this module as long as the switchback region is large enough so for the read layer to terminate inside this region even at the largest possible offset read.

The *capacity* of a U-turn pocket is defined to be the length of a path taken by the read and write layers from the rightmost **T** on the read layer to the rightmost **T** on the write layer. It is determined by the three independent parameters k, w, ρ with $\rho < 2k + 1$, and one dependent parameter x as:

$$\mathsf{capacity} = 2n(w(2k + 1) + \rho + 1) + 2(w + 4 + x).$$

For the sake of n-folding meters, x should be set as:

$$\mathbf{x} = (\mathbf{w} + 4).\mathsf{complement}(\mathsf{to}\colon n)$$

so that the capacity becomes a multiple of $2n$.

For $\mathsf{maxshift} = \mathcal{W} = \sum_{i,j} w_{ij}$, let us solve $\mathsf{capacity} \geqslant 2n\,\mathsf{maxShift}$ so as to shorten the scaffold as much as possible; the ideal ratio is $w \sim nk$. Plugging in this value into the formula of capacity together with the lower bounds $w + 4 + x \geqslant n$ and $\rho \geqslant 0$, we get that we are looking for a value of k verifying

$$nk \cdot (2k + 1 + 1/n) + 1 \geqslant \mathsf{maxShift}\,.$$

Solving this inequality implies that it suffices for k to be at least :

$$k \geqslant \max\left\{0, \frac{\sqrt{2n(4 \cdot \mathsf{maxShift} - 3) + n^2 + 1} - 1 - n}{4n}\right\}.$$

Now the remaining parameters can be computed one after another as:

$$w = 1 + \max\left\{0, \left\lfloor \frac{\mathsf{maxShift} - 1}{2k + 1 + 1/n} \right\rfloor\right\},$$

$$\rho = \max\{0, \mathsf{maxShift} - (1 + (2k + 1)(w - 1) + (w + 4 + x)/n)\}.$$

▶ **Proposition J.3.** *The algorithm above outputs parameters (k, w, ρ, x) such that the box swallows at least $\mathcal{W} + (2k + 1)$ $2n$-periods of the folding meter layer and such that k, w are $O(\sqrt{\mathcal{W}})$.*

J.11 Corner module

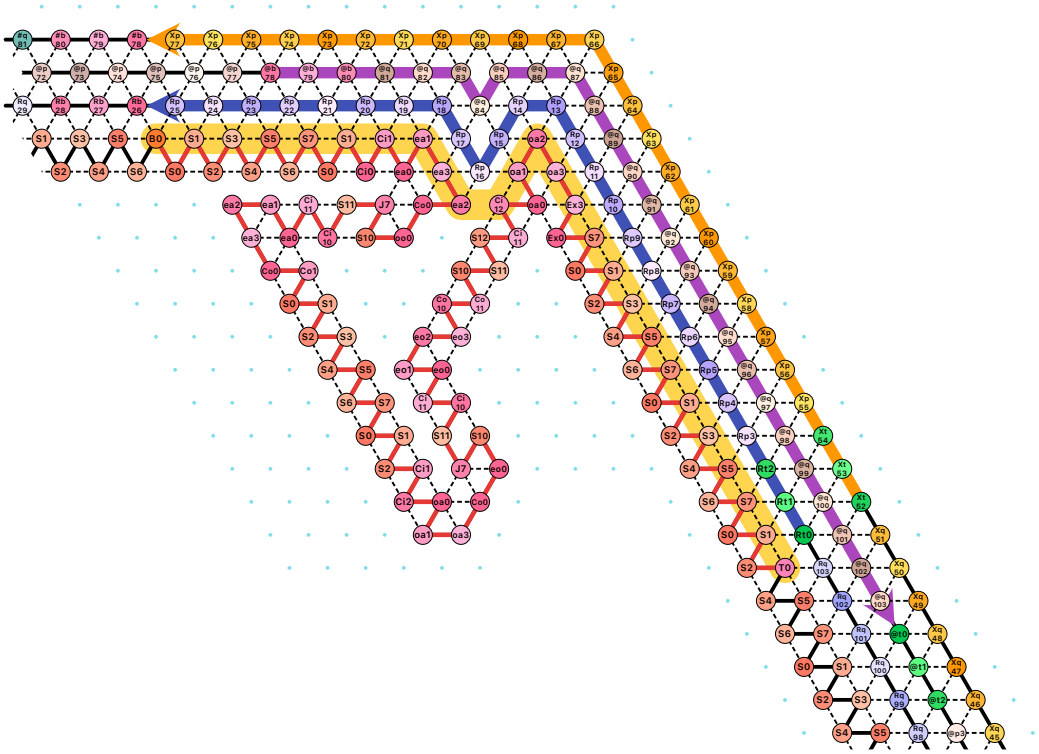


Figure 28 Corner module.

At a corner, outer layers need to go farther. The corner module prevents the resulting dissynchronization by a dent on its scaffold, which counteracts the difference in distance.

J.12 Exit pocket

All the sides but the north one, at which the transcript enters from the previous macrocell or from the seed are provided with an exit pocket, at which the transcript can finalize the current macrocell and go to the next macrocell. The exit layer is as long as the four sides of the macrocell so that it can reach even the northeast side starting after the speedbump, which is at the northwest side. Thus, in order to leave earlier, the remaining portion of the exit layer must be consumed. The exit layer lets the portion be folded into switchbacks, but then the earlier the system terminates, its output point shifts leftward. In order to counteract this for the sake of upcoming macrocell, every exit pocket is sandwiched by two shift modules (this shall be explained in details soon in Sect. J.15). The exit layer decides whether to exit now or later at the designated kick (see Fig. 29). The transition table for this decision shifts around and the table entry corresponding to an input read comes exactly at this point to either attract the exit layer downward, which means that this pocket is not chosen, or not, when the exit layer folds back upward and exits here.

The *capacity* of an exit pocket is defined to be the distance from the first **B** after the branch of the two possible paths of the exit layer to the end of the transcript, and it is determined by its parameters k, w, r as:

$$\mathsf{capacity} = (2nw - 1)(2k + 1) + 2n(2r + 1).$$

The exit pocket involves other two parameters W and x. The parameter W is namely the width of the pocket and is set as:

$$W = n - 10 + 2n\eta$$

for some η large enough to guarantee $w \leqslant W$. The other parameter x is dependent on η; once η is fixed, $x < (2k + 1)n$ is set so as for the length of the read, write, and exit layers each between the leftmost **T** and the rightmost **B** to be equal to n modulo $2n$, that is,

$$n + 2n(2k + 1) + 2(x + W + 10) = n \bmod 2n.$$

The one at the northwest corner requires the largest capacity to afford the remaining portion of the current period of transcript, which amounts to the four macrocell's sides. In order to compute the parameters k and w, however, we need to know the size of the resulting macrocell size, that is, a fixed point equation must be solved. We shall settle this issue in Sect. J.18. By appending some extra n-folding meters at the end of the transcript, r can be set to k. At the other macrocell's sides, the system also employs the exit pocket of this size.

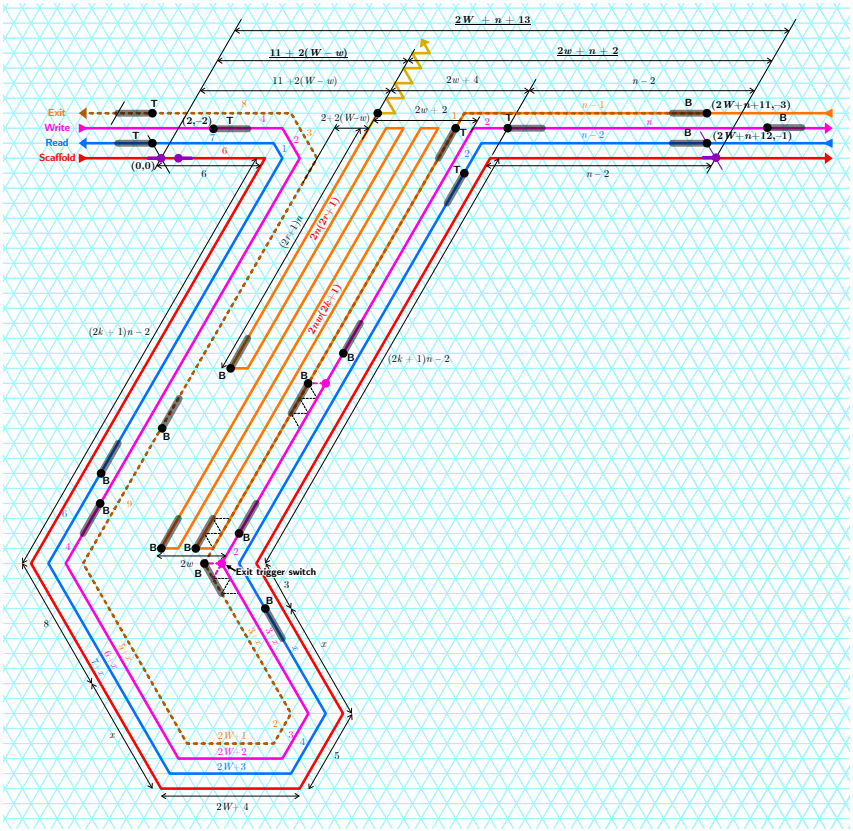


Figure 29 Exit pocket. Given k and w, parameter x must be adjusted so as to match the period of the folding meter.

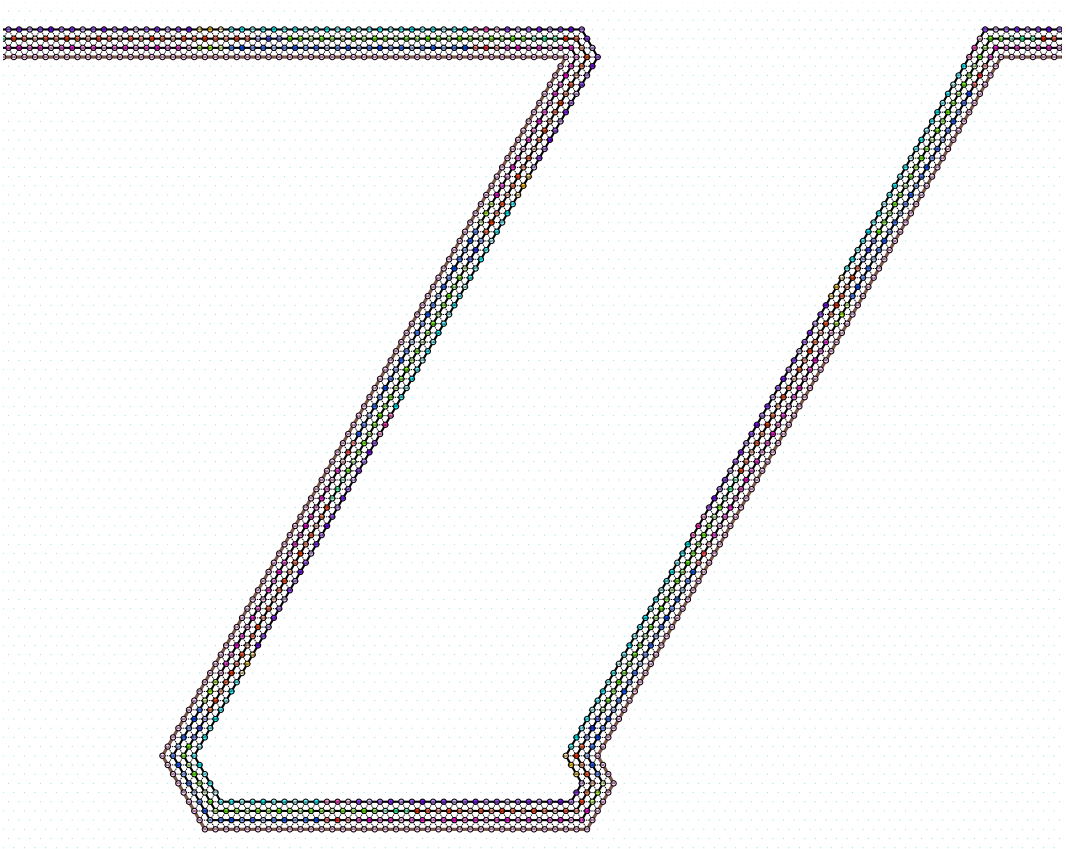


Figure 30 Exit pocket: the exit layer does not exit if its **xb**-beads are attracted by the trigger bead at the bottom corner of the pocket, namely $\mathbf{Wp}_{76}$ here.

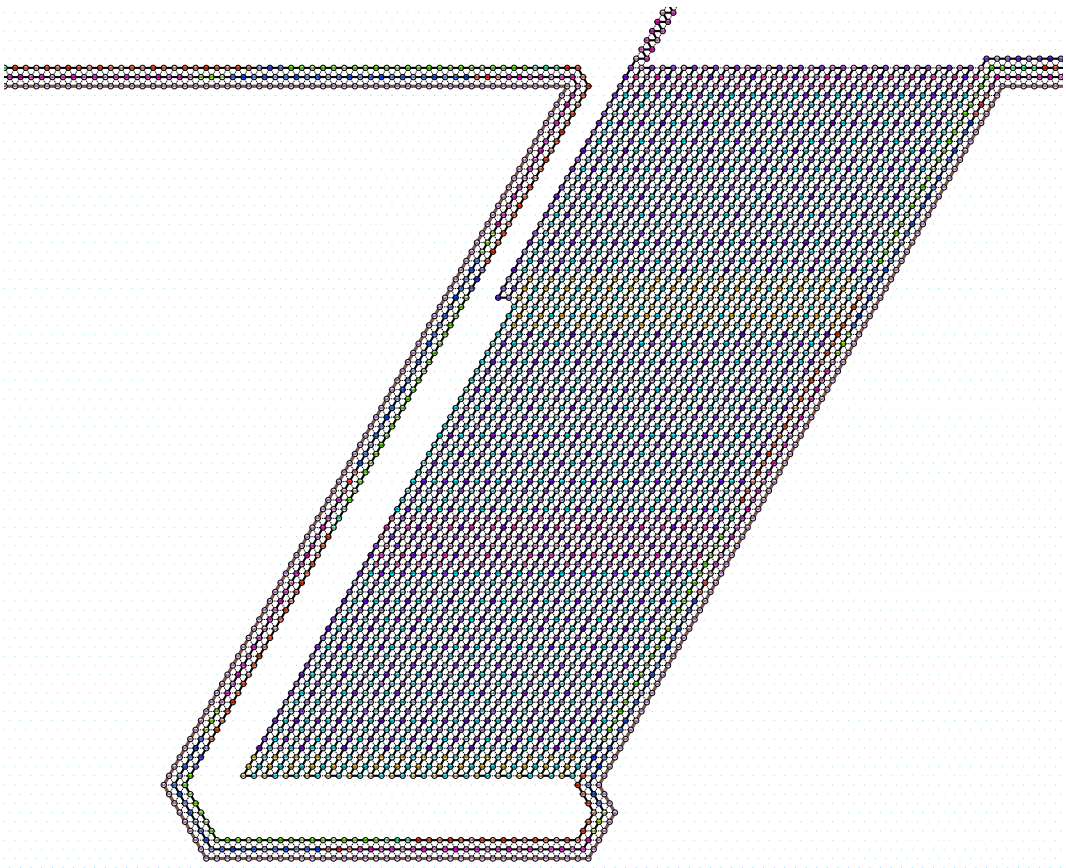


Figure 31 Exit pocket: the exit layer does exit if its **xb**-beads are not attracted by the trigger bead at the bottom corner of the pocket, namely $\mathbf{Wp}_{76}$ here.

J.13 Exit interchange trap

This trap lets the system color its exit layers into non-interacting regions, as read and write pockets do for the corresponding layers. The only place where the two exit colors are in contact are inside the exit pocket. Since the exit layer does not shift, this trap only needs to be as high as the exit pocket. Its design is thus super simple as shown in Fig. 32. The parameter ρ will be set to $2k$ where k is the parameter of the exit pocket.

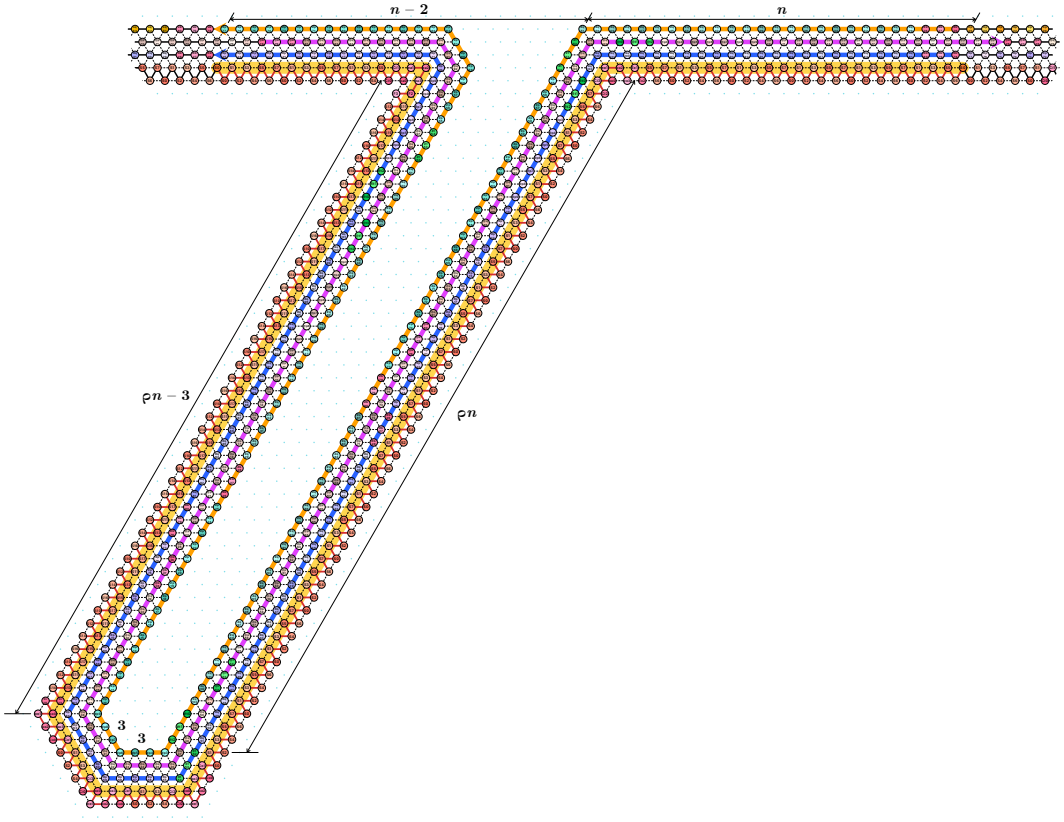
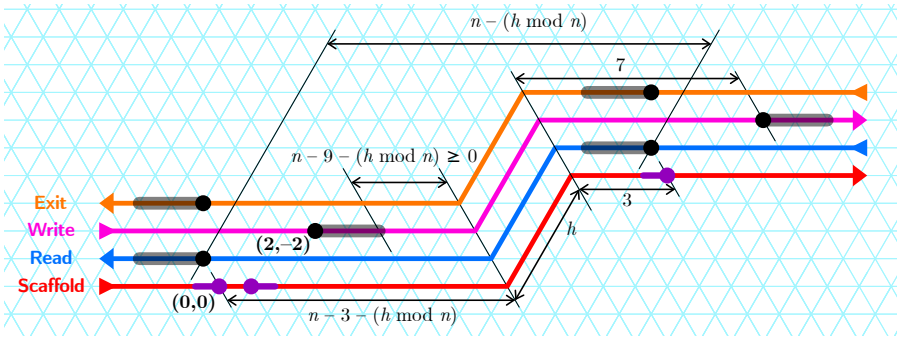


Figure 32 Exit interchange trap.

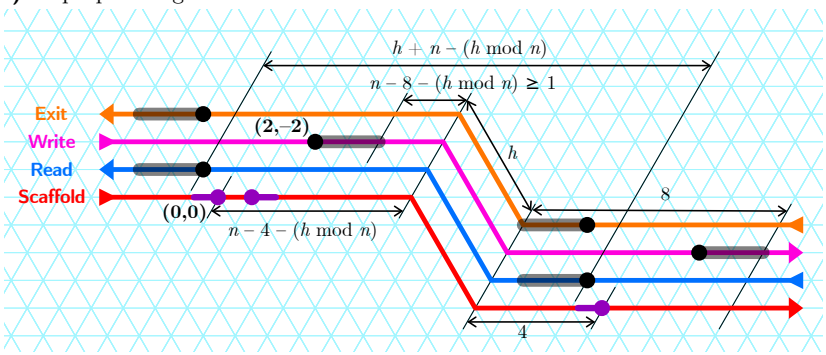
J.14 Step and shift modules

Step up and step down modules. The designs in Fig.33a and 33a fold as indicated for any h such that $0 \leqslant (h \bmod n) \leqslant n-9$. The total length of each layer inside this modules is $n + h - (h \bmod n)$.

For steps of height h with $n - 9 \leqslant (h \bmod n) < n$, we concatenate two such modules, one with $h' = n - 9$ and one with $h'' = h - (n-9)$. The total length of each layer is then $2n + h - (h \bmod n)$.



(a) Step up of height h.



(b) Step down of height h.

Figure 33 The step up and down modules.

Shift module. Shift module will be useful to shift the exit pocket left or right so that the exit glider exits always at the same position, regardless of the number of switchbacks of the exit layer packed inside. It consists in concatenating a line module, one step down module with and one step up module of the same height (see Fig. 34a and 34b).

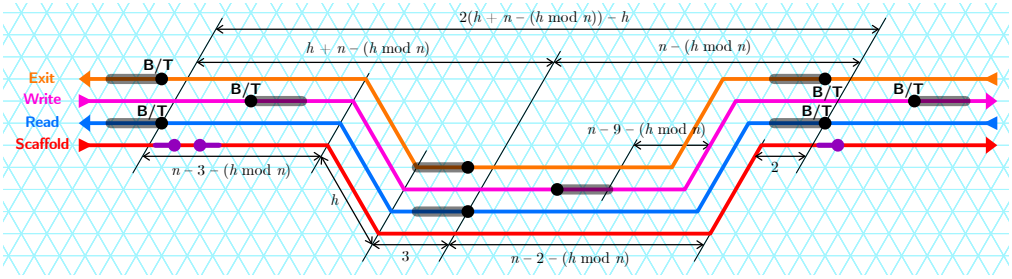
▶ **Lemma J.4.** *Given a positive integer L and $Ln + 9 \leqslant h \leqslant 2nL$ written as $h = qn + r$ where $0 \leqslant r < n$, then the distance between the leftmost T and rightmost T of the concatenation of:*

- *a horizontal line of length $2(L - q - 1)n$, a step down and a step up, both of height h if $r \leqslant n - 9$;*
- *a horizontal line of length $2(L - q - 2)n$, two step down and two step up, both pairs of heights $n - 9$ and $h - n + 9$ respectively, if $r > n - 9$;*

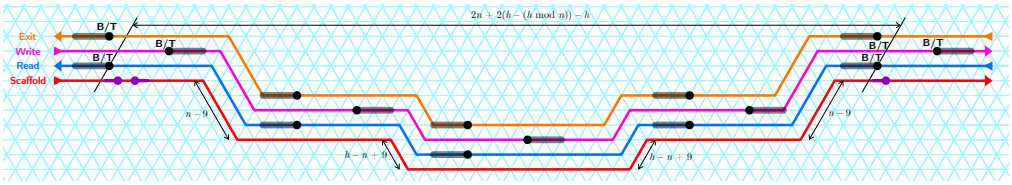
is precisely $2Ln - h$ and the total length of each layer is $2Ln$.

Proof. In both cases, the total length of each layer is $2Ln$. In the first case, the distance between the two extremities is $2(L-q-1)n + 2(h+n-r) - h = 2Ln - h$. In the second case,

note that since $h \geqslant Ln + 9$, then $q \leqslant L - 2$, thus the line initial segment has positive length. Furthermore, the distance between the extremities is $2(L - q - 2)n + 2n + 2(h + n - r) - h = 2Ln - h$. ◀



(a) Single-level shift module of height h for $(h \bmod n) \leqslant n - 9$.



(b) Two-levels shift module of height h for $(h \bmod n) > n - 9$.

Figure 34 The two variants of the shift module.

J.15 Exit module

Consider an exit pocket with parameters W and w where $w = w_0 + i\omega$ for some $i \in \{0, \dots, 4\}$ with $W \geqslant w_0 + 4\omega$. Consider $L = (8\omega + 9).\mathsf{nextMultiple}(\mathsf{of}\colon n)$ such that $8\omega \leqslant Ln - 9$. The exit module for i consists in sandwiching this exit pocket between two shift modules of length $2nL$ and height $2i\omega$ and $2Ln - 2i\omega$. Then, the distances from the leftmost T to the glider exit location and to the rightmost T are both independent of i, as illustrated on Fig. 35. Indeed, the shift of the glider by $2i\omega$ is balanced by the left shift module whose extension is in turn balanced by the right shift module. Note that all layers have the same length which is also independent of i.

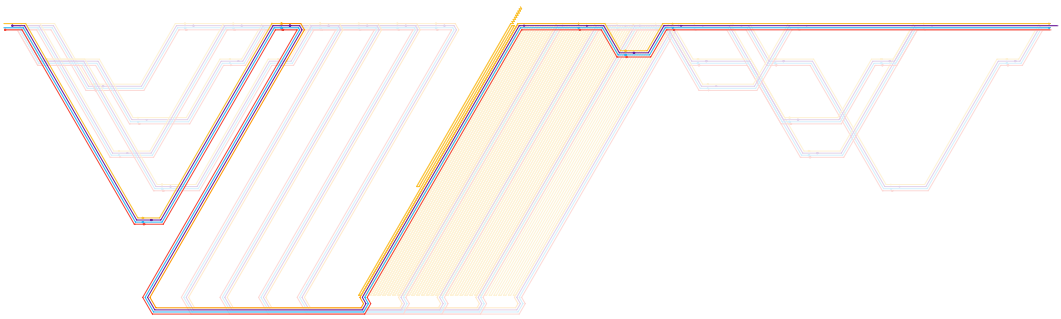


Figure 35 Exit module for the five values of $w = w_0 + i\omega$ for $i \in \{0, \dots, 4\}$ (here, $w_0 = 1$ and $\omega = 8$).

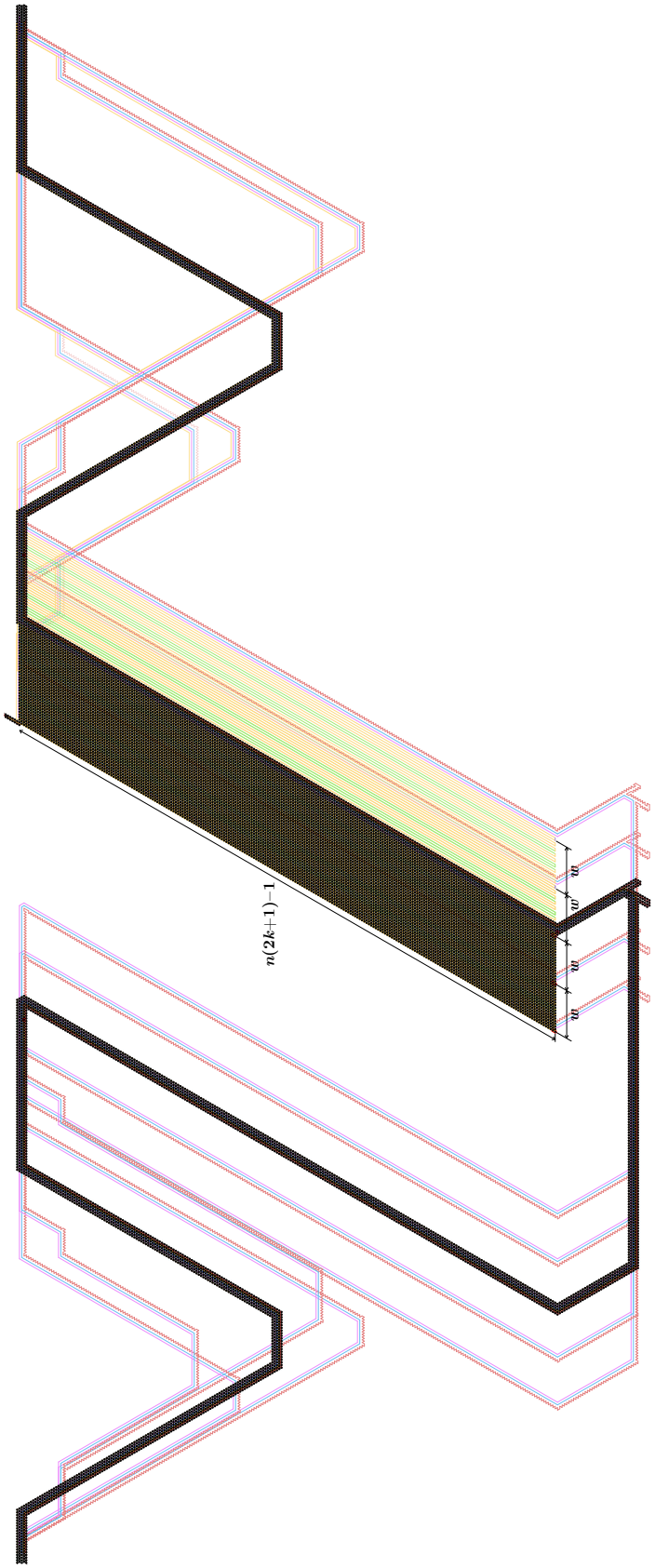


Figure 36 Actual shapes of the sliding exit module.

J.16 Corner interchange block

This block provides enough space to accommodate the junctions between different colors along all the layers so as to free the other more fundamental modules from the unnecessary load to handle transcript of multiple colors. A corner interchange block is provided with one read pocket, two write pockets, and one exit trap (Fig. 37); among these modules, only the read pocket varies in capacity as it is of capacity $Q^i - 1$ at the i-th corner. Since the offset yielded so far does not exceed $2nQ^i$, this is large enough to guarantee that the color shift on the read layer from Read2 to Read1 occurs inside the pocket via the auxiliary third color Read12. The junction between Read1 and Read2 is colored in Read12, and if the colored factor is twice as long as the depth of this read pocket, then the transcript in color Read1 never directly interacts with the transcript in color Read2 while being packed compactly inside the pocket. The write layer changes the color also via the auxiliary third color Write12, but not in one pocket locally but rather throughout the whole block. Folding cw., this layer changes the color from Write2 to Write12 in the first write pocket, and then from Write12 to Write1 in the second one. Since the write layer can shift at most by $2nQ^6$, these pockets must be of capacity $Q^6 - 1$. The exit layer changes the color from Exit2 to Exit1 via the auxiliary color Exit12 inside the exit trap.

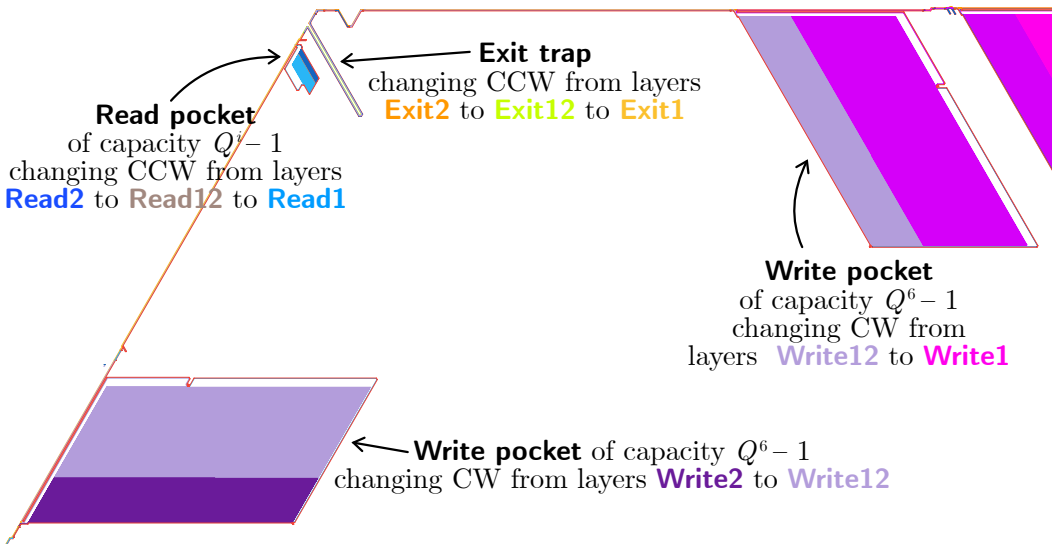


Figure 37 Corner interchange block for $q = 2$ ($Q = 4$ states).
Zoom in for details

J.17 Middle interchange block

This block offers enough space to reverse the color change done at the corner interchange blocks. It works exactly in the same manner as the corner interchange block does.

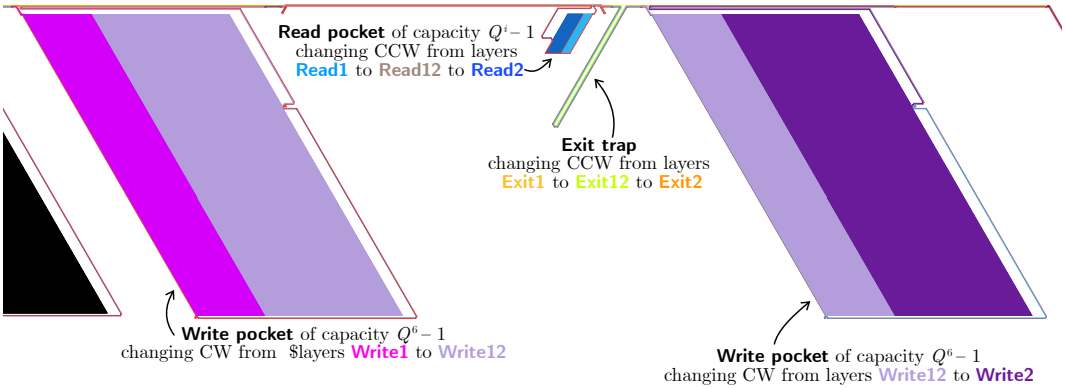


Figure 38 Middle interchange block for $q = 2$ ($Q = 4$ states).
Zoom in for details

J.18 Determining the macro-cell size: solving the fix point

In order to conclude the construction of the read/write part of the macro-cell, we need to compute the length of its side. But, as the size of the exit module depends on the length of the side of the macro-cell (recall that the whole length of the side must fit in it), we need to solve a fixed point equation.

Combining all the blocks described so far (i.e., Writing, Middle interchange, Exit, Reading, and Corner interchange) yields a macro-cell as depicted in Fig. 2, where $q = 3$ (bits) and hence, the macro-cell is provided with one writing block with 3 write modules and one Reading block with 3 read pockets per side.

Two adjacent write modules in the Writing block must be equally distanced as two adjacent read pockets in the Reading block. The distance is hence set to the maximum of $\ell_{\mathsf{write}} - 1$ and all the y's of all reading pockets involved; thus these modules never collide. Concatenating $q + 1$ write pockets and q write modules alternately yields the Writing block with this proper spacing. Concatenating q read modules does not suffice for the Reading block because these vary in size (doubling their sizes for reading in binary), and hence, their y's may differ. Padding between them a straight line of length multiple of $2n$ places these modules spaced equally and properly. Interchange blocks each involves two write pockets and one read pocket. The y parameter of the read pocket of Middle interchange block is set to $1 +$ the largest possible y and the largest possible k computed for all the read pockets involved (see Section J.7). Thus, the left write pocket, this read pocket, and the exit trap, which is concatenated directly to the right of the read pocket at all sides but the northeast one never collide; at the northeast side, the exit trap is replaced by a proper combination of line and step up/down modules so as not to change the length of macro-cell side). The same y is used for the read pocket of Corner interchange block. Finally, the Exit block consists of a line of $2n(k_{\mathsf{exit}} + 1)$ and the Exit module, which consists of an exit pocket parameterized by $W = w_{\mathsf{exit}}$ and $k = r = k_{\mathsf{exit}}$ and sandwiching two shift modules of maximum shift $2w_{\mathsf{exit}}$.

These five types of blocks are concatenated into a side (though the resulting "side" bends due to the Corner interchange block, it is more convenient for our sake to consider it as a side). In order for the read and write pockets to be stuck up towards the macro-cell surface, each of them is sandwiched by step-up and -down modules, each of length $2n$. The exit layer path of each of these blocks is of the following length:

- Read block: $2n(q + 2)$ plus $2nq$ times the distance between the adjacent write modules;
- Corner interchange block: $2n(2\ell_{\mathsf{write}} + y + r + 5)$, where y is the one computed for the interchange above;
- Write block: $2n(q - 1)(\ell_{\mathsf{write}} + 1)$;
- Middle interchange block: $2n(2\ell_{\mathsf{write}} + y + r + 7)$, where y is the y computed for the interchange before;
- Exit block: $2n(4n + 2(2k_{\mathsf{exit}} + 1)) + 2n \times 10\eta$

Taking the step-up and -down modules for the read and write pocket into account, all of these lengths sum up to the length of one side measured along the exit layer path. For the sake of upcoming arguments, let us represent this total length as $2n(A + 10\eta)$.

Now it suffices to set the parameters k_{exit}, η, and x of the exit module properly, and the description of the macro-cell at least size-wise shall be completed. First, note that the asymptotic length of the side is $O(q\sqrt{\mathcal{W}})$. Thus, let us set k_{exit} to $\sqrt{q\sqrt{\mathcal{W}}}$ to ensure that the exit pocket will have its height and width balanced. The exit pocket must be large enough in order to accommodate four sides of the exit layer being folded into switchback; formally speaking, $4(A + 10\eta) \leqslant (2k_{\mathsf{exit}} + 1)w_{\mathsf{exit}} = (2k_{\mathsf{exit}} + 1)(n - 10 + 2n\eta)$. Solving this inequality justifies $\eta = \lceil (4A - 16(2k_{\mathsf{exit}} + 1))/(n(2k_{\mathsf{exit}} + 1) - 40) \rceil$. Now it suffices to fix x as

$n \cdot ((A + 10\eta).\mathsf{complement}(\mathsf{to}\colon\ 2k_{\mathsf{exit}} + 1))$ so that the length of layers along the exit pocket between its leftmost **T** and rightmost **B** becomes n modulo $2n$.

J.19 Tracker speed bump

This module is an adaptation of the speed bump from [21] to delay 3 and along an arbitrary track path. As opposed to [21], where the speedbump is located along the side of the macrocell, we need the speedbump to occupy a compact space. For this purpose, we want this module to adopt a "snake-like" shape to fit into an area of radius the square root of its total length. Together with the delay 3 (as opposed to delay 2 in [21]), this imposed to redesign completely this module, yielding, after a lot of struggling, to a surprinsingly simple and modular conception allowing arbitrary complex path for the track.

A speedbump consists in an area of the macrocell in which the transcript can enter with some shift upper bounded by a predefined maximum value, and will exit with a zero shift, whatever the initial shift was, allowing to realign the transcript regardless of what happened before.

The speedbump is a collaboration of a speedbump scaffold and speedbump layer. The speedbump scaffold is provided with an alternation of flat area of period 4 (θ0..3) and bumper area of period 4 (β0..3) except at the so-called fish tail, where more β types are employed. The speedbump layer is mainly composed of the three modules; bumping sequence, flat sequence, and rephaser. The flat sequence merely crawls along the scaffold. The bumping sequence is a repetition of Λ0Λ1Λ0Λ2, which bumps in bumper areas, as long as it is synchronized properly with the period-4 scaffold patterns just explained. The synchronization is guaranteed by the preceding rephaser ϕ0..5, which absorbs any small shift up to 3.

Correctness of the speedbump. The bead types involves in the fional macrocell are confusing because they are the result of a product of the scaffold bead type (used to build the shape of the scaffold) and the speedbump bead type (used to functionalise the scaffold). In this proof we will just consider the speedbump bead types, disregarding the actual bead in the macrocell to focus only on the speedbump behavior proof, using thus bead types that do exist as is in the macrocell. Here the table of translation between those bead types:

Here	Macrocell
Transcript:	
D0..2	Λ0..2
C0	λ0
E0..5	ϕ0..5
Scaffold:	
A0	θ0, 5, S7, 17, F0, 2, Ci1, J4
A1	θ1, S5, 15, Co2
A2	θ2, S3, 13, eo1, Ex1, Co1
A3	θ3, 4, S1, 11, oo3,
F0	ϕ0, 3
F1	ϕ1, 2

As in the original solid speedbump in [21], any shift decays logarithmically. The transcript consists in a sequence of bumping (D0D2D0D1) and stubbornly-flat (C0) sequences, whereas the speedbump scaffold consists in an alternation of bumper (B0/1) and flattening sequences (A0..3), all (both transcript and scaffold) exponentially decreasing in length. In addtion, we need a rephaser module E0..5 in the transcript between the C0- and the D0D1D0D2-parts

The following figures show that in every situation the speedbump transcript behaves as expected. Fig 39 shows it operating in a real macrocell.

The delay-3 speedbump rule.

A0 💔 C0,D0,D2,E0,E1
A1 💔 C0,D0,D2,E0,E1,E3,E4
A2 💔 C0,D0,D1,E0,E1,E4,E5
A3 💔 C0,D0,D1,E0,E1,E5
B0 💔 C0,D0,E0,E1
B1 💔 C0,D1,D2,E0,E1
C0 💔 A0,A1,A2,A3,B0,B1,F0,F1
D0 💔 A0,A1,A2,A3,B0,D0,F0
D1 💔 A2,A3,B1,D2
D2 💔 A0,A1,B1,D1,F0,F1
E0 💔 A0,A1,A2,A3,B0,B1,E2,F0,F1
E1 💔 A0,A1,A2,A3,B0,B1,E3,E5,F0,F1
E2 💔 E0,F0
E3 💔 A1,E1,F1
E4 💔 A1,A2,F1
E5 💔 A2,A3,E1
F0 💔 C0,D0,D2,E0,E1,E2
F1 💔 C0,D2,E0,E1,E3,E4

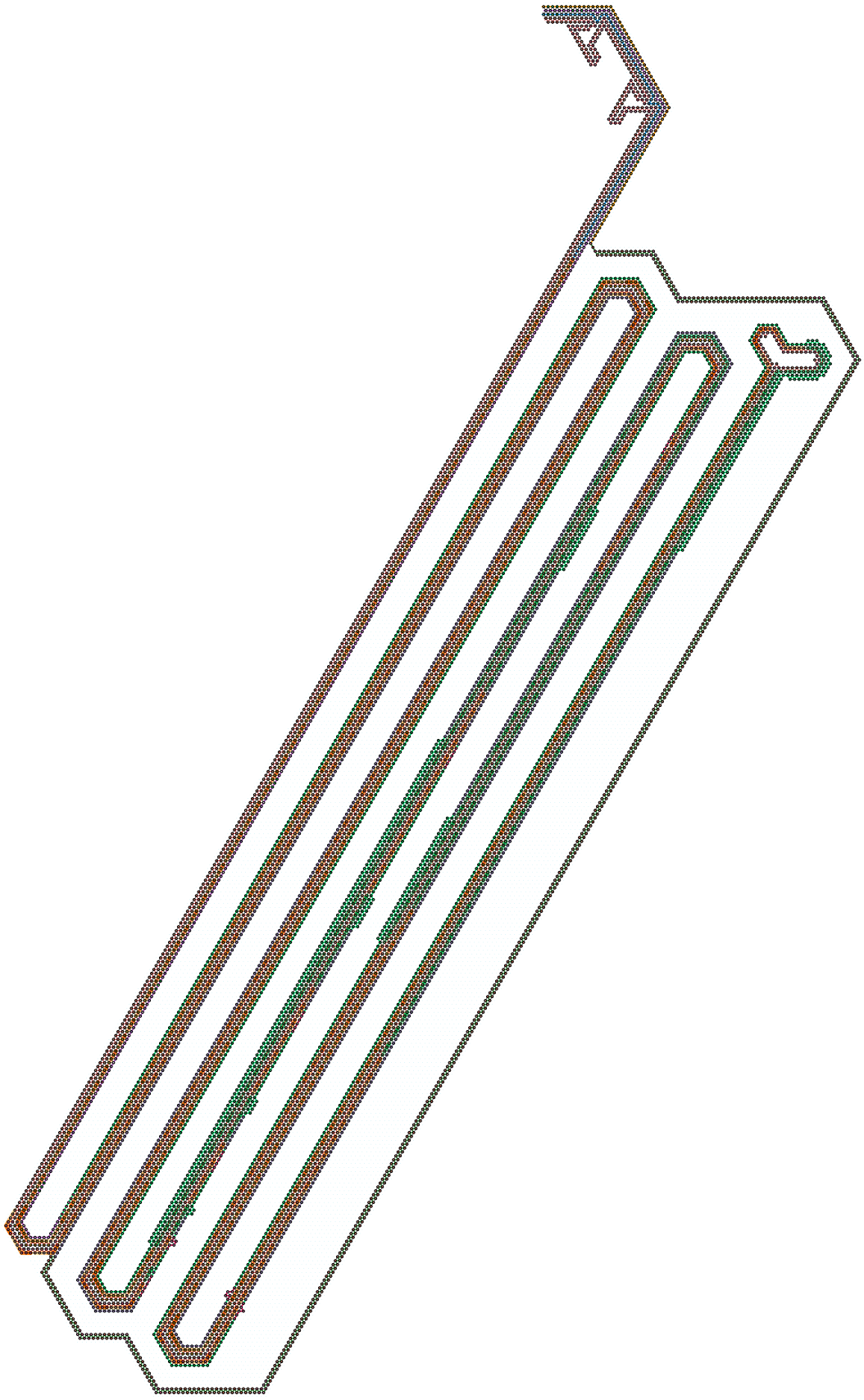


Figure 39 Speed bump module.

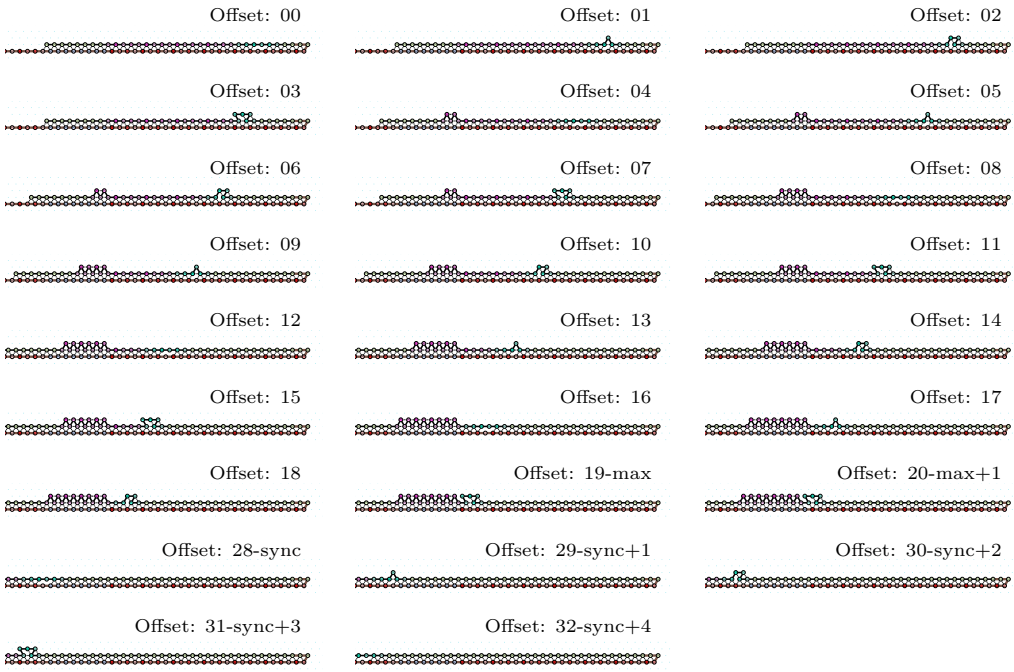


Figure 40 Self-rephasing speedbump on flat surface

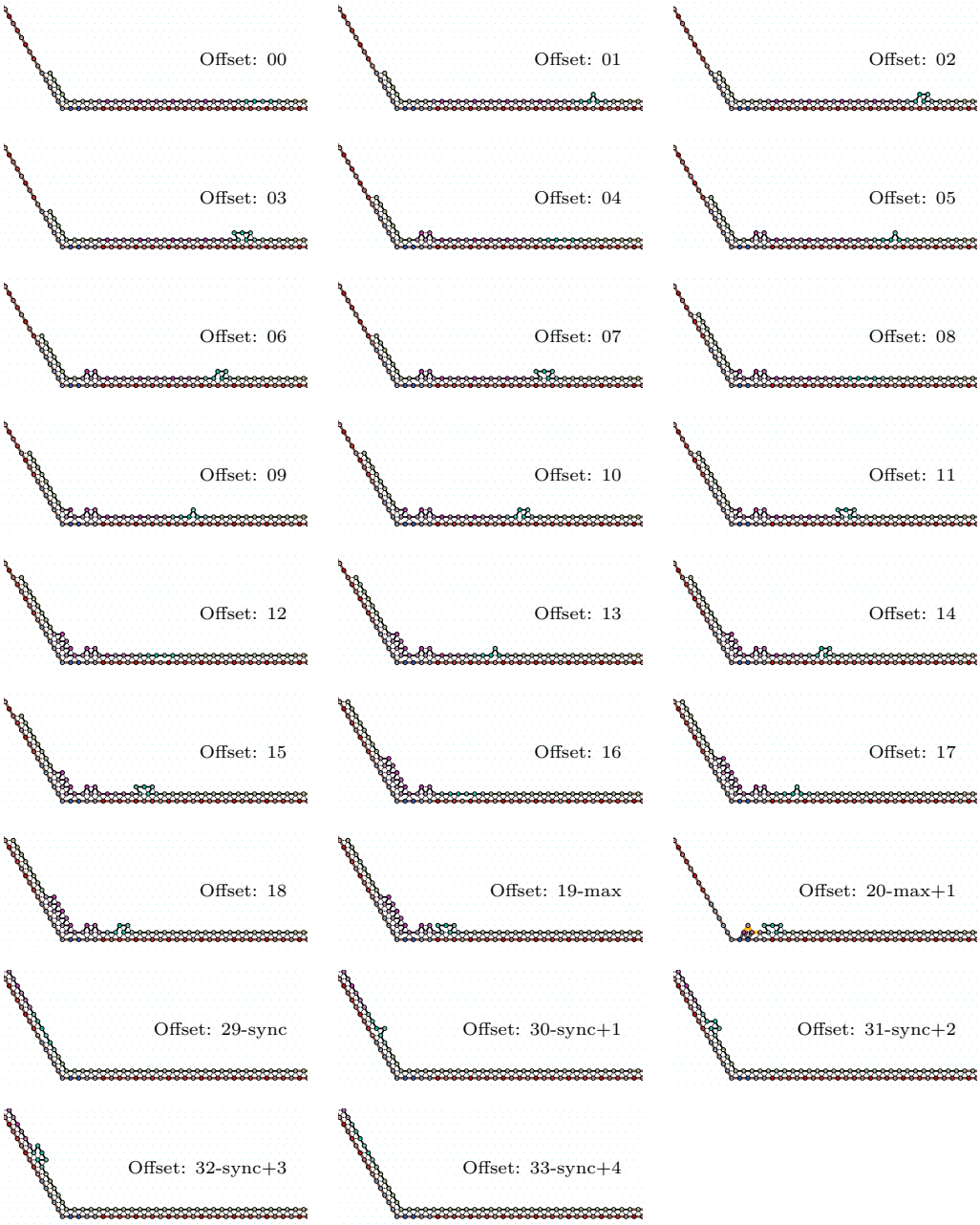


Figure 41 Self-rephasing speedbump on inward surface

Figure 42 Self-rephasing speedbump on inward2 surface

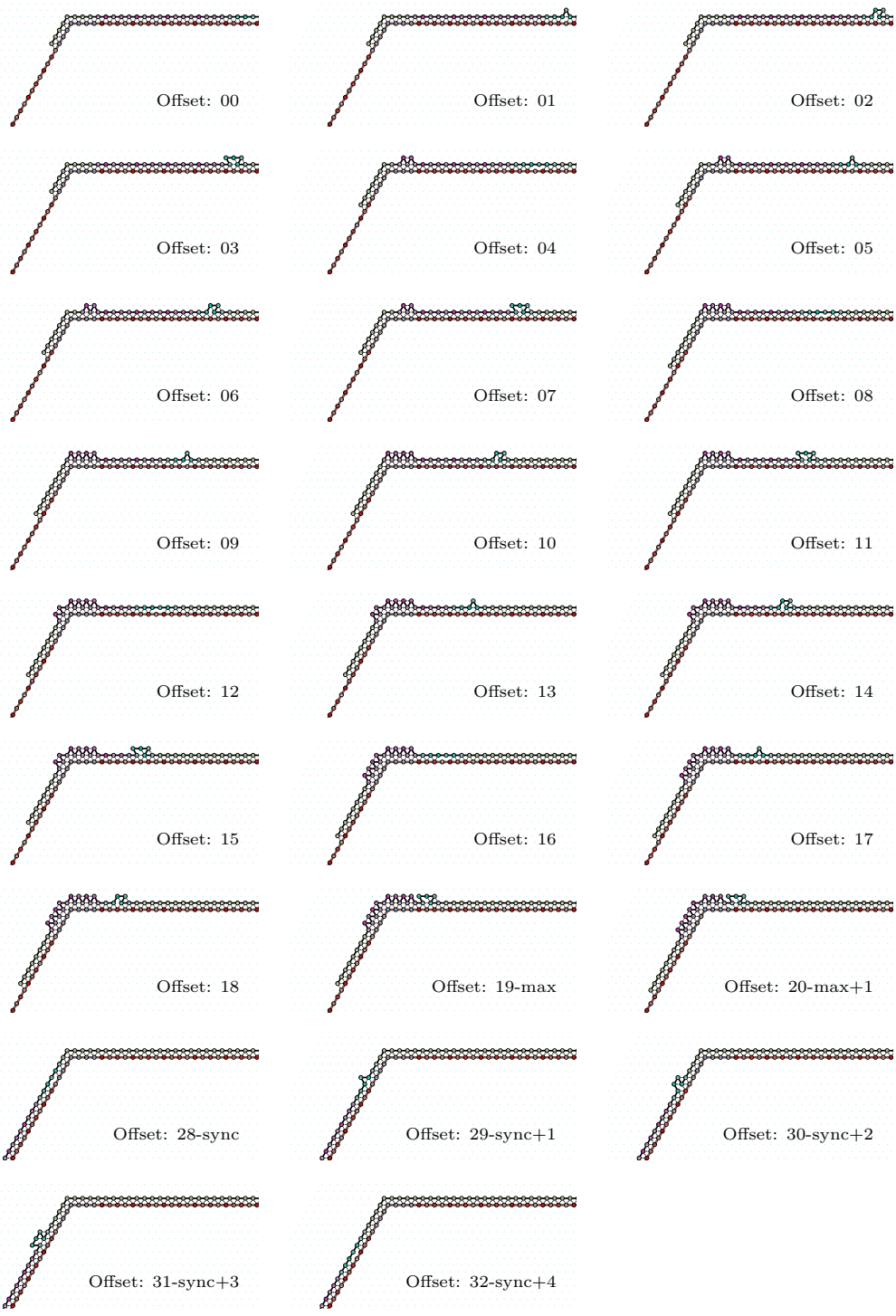


Figure 43 Self-rephasing speedbump on outward surface

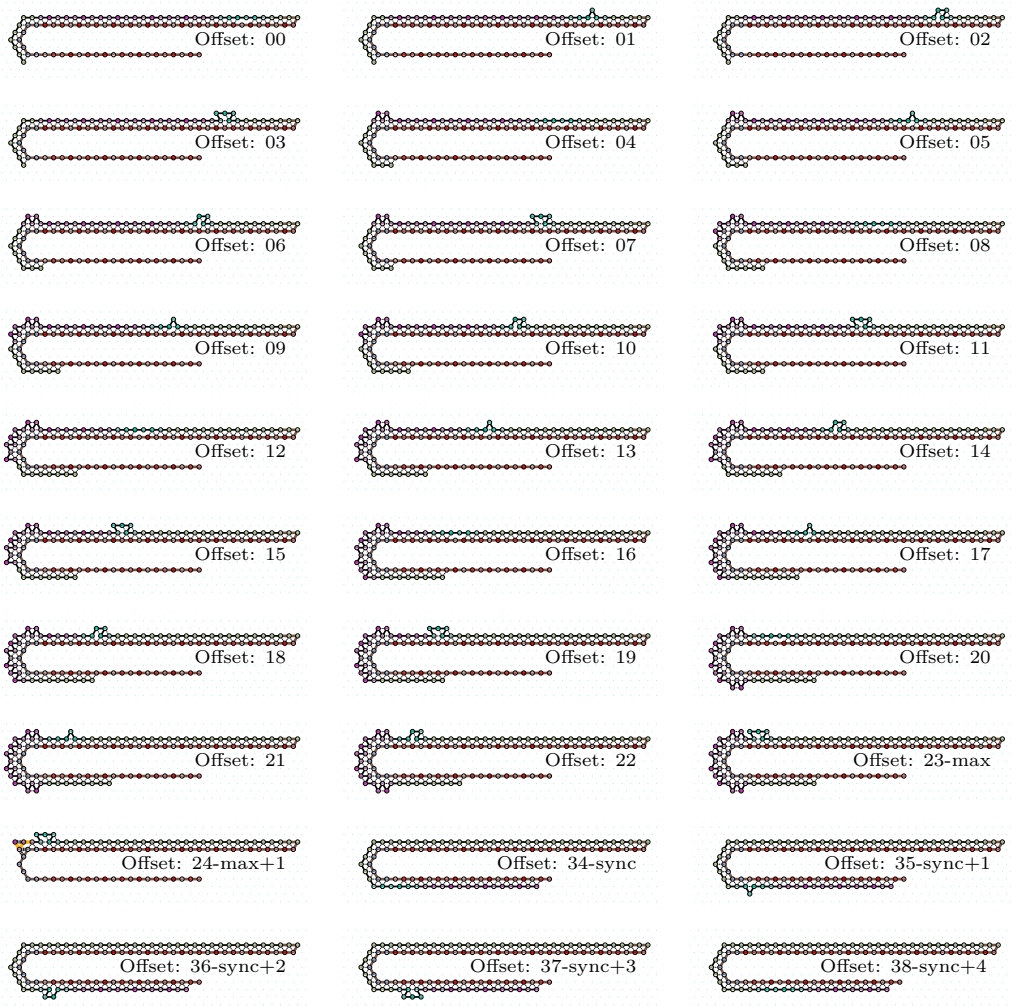


Figure 44 Self-rephasing speedbump on outward2 surface

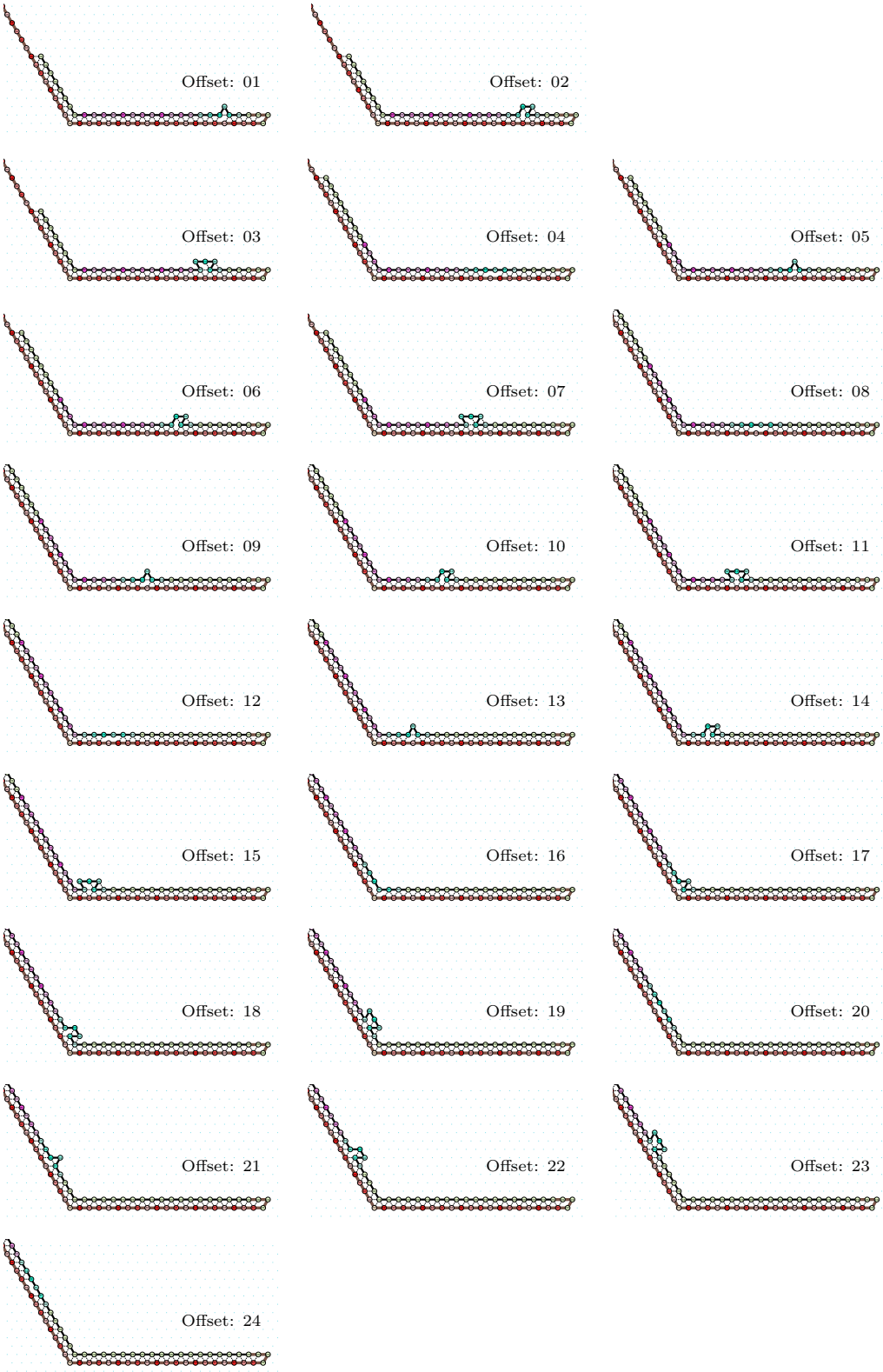


Figure 45 Self-rephasing speedbump on inward red surface

Figure 46 Self-rephasing speedbump on inward2 red surface

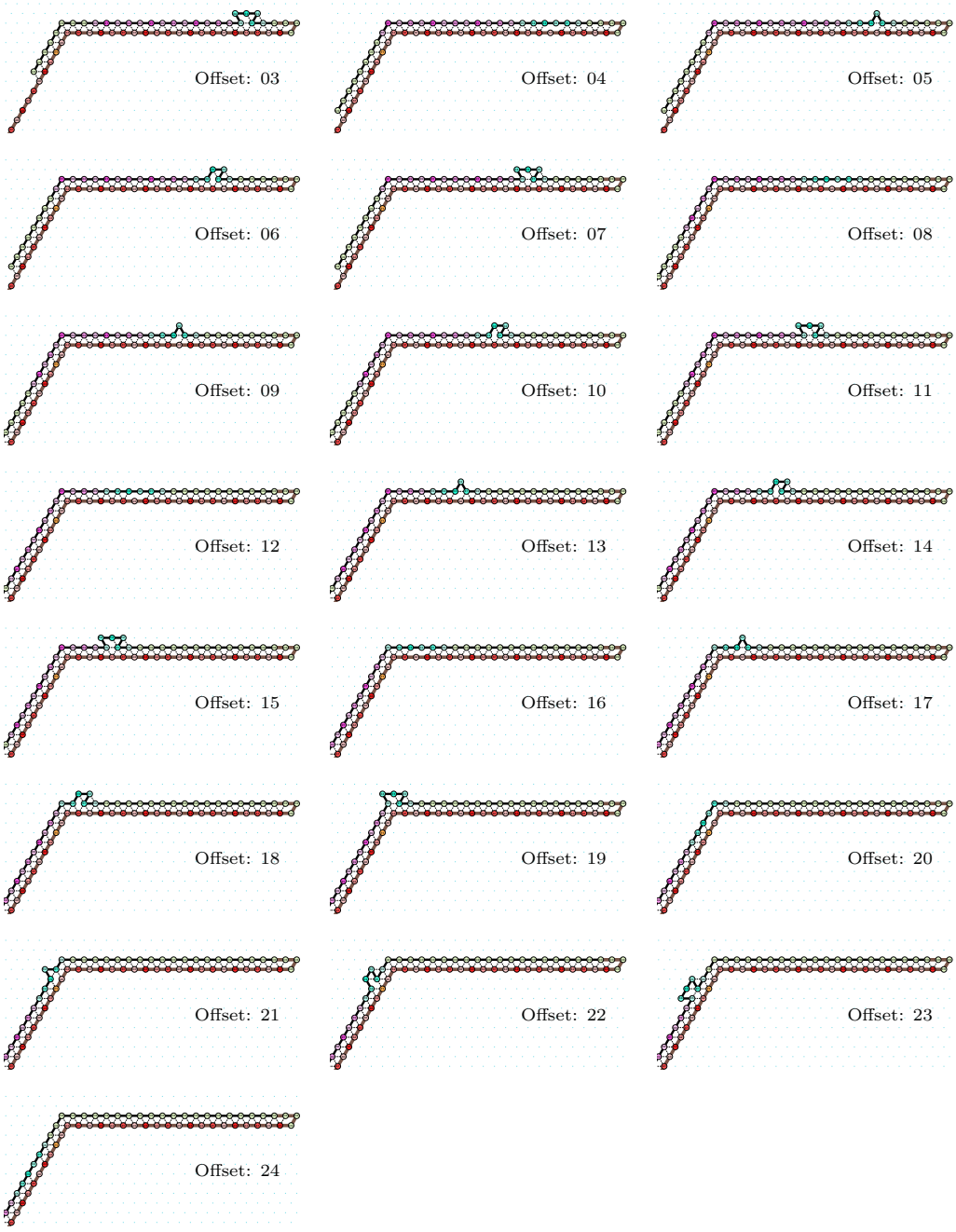


Figure 47 Self-rephasing speedbump on outward red surface

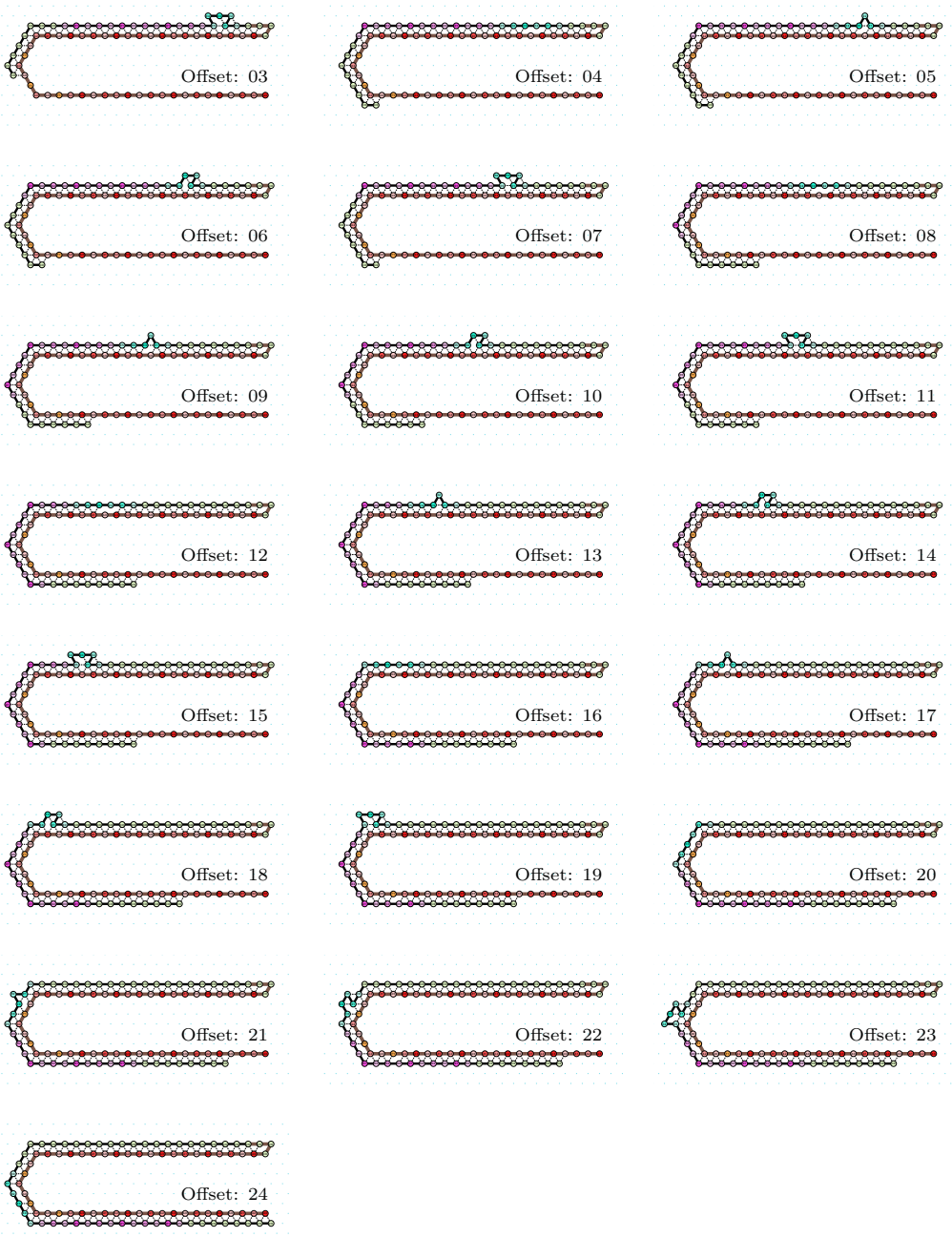


Figure 48 Self-rephasing speedbump on outward2 red surface

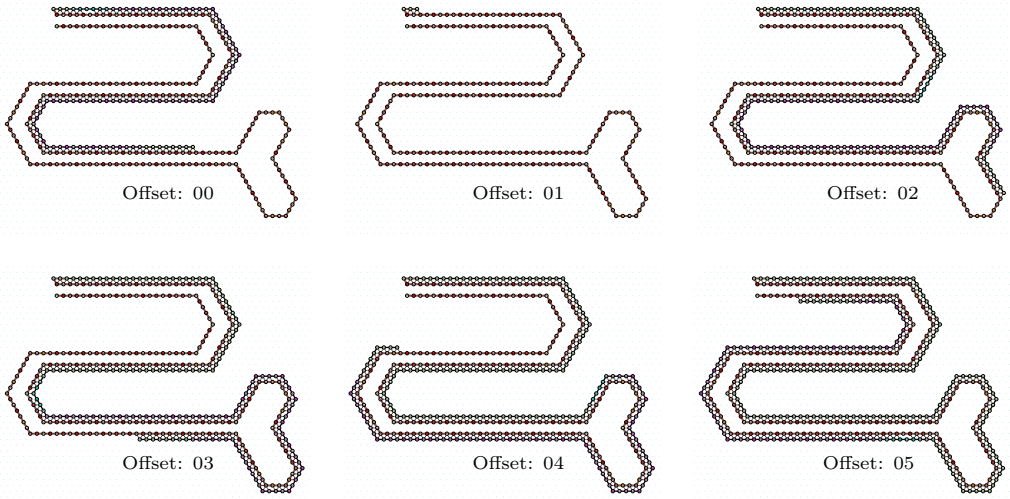


Figure 49 Self-rephasing speedbump on full red surface

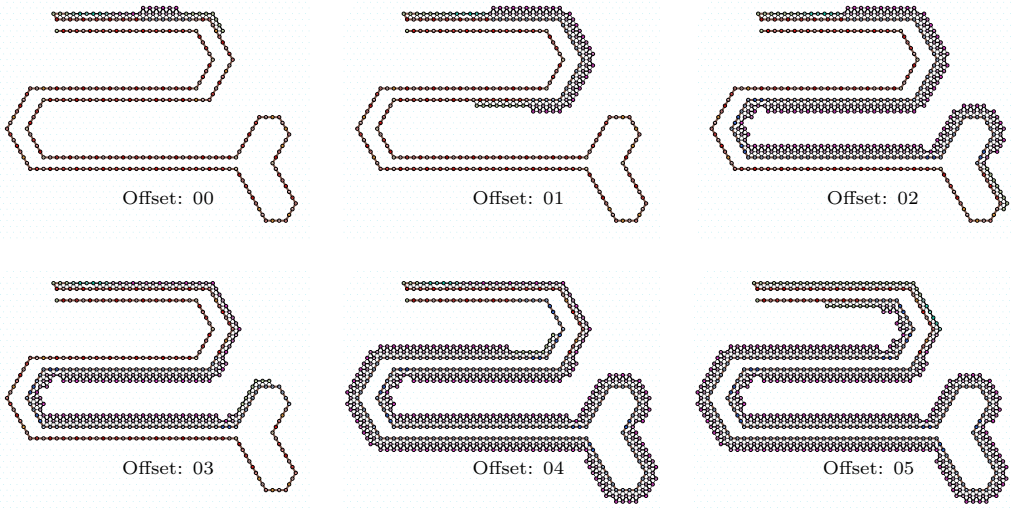


Figure 50 Self-rephasing speedbump on full blue surface

K The turedo-to-oritatami compiler

The current implementation includes 1735 bead types. Examples of turedo compiled as oritatami as well as a fully functional python compiler (soon available) can be downloaded from [1]:

`https://hub.darcs.net/turedo2oritatami/turedo2oritatami/`

The resulting `.os` files are to be run on the oritatami simulator by [23].

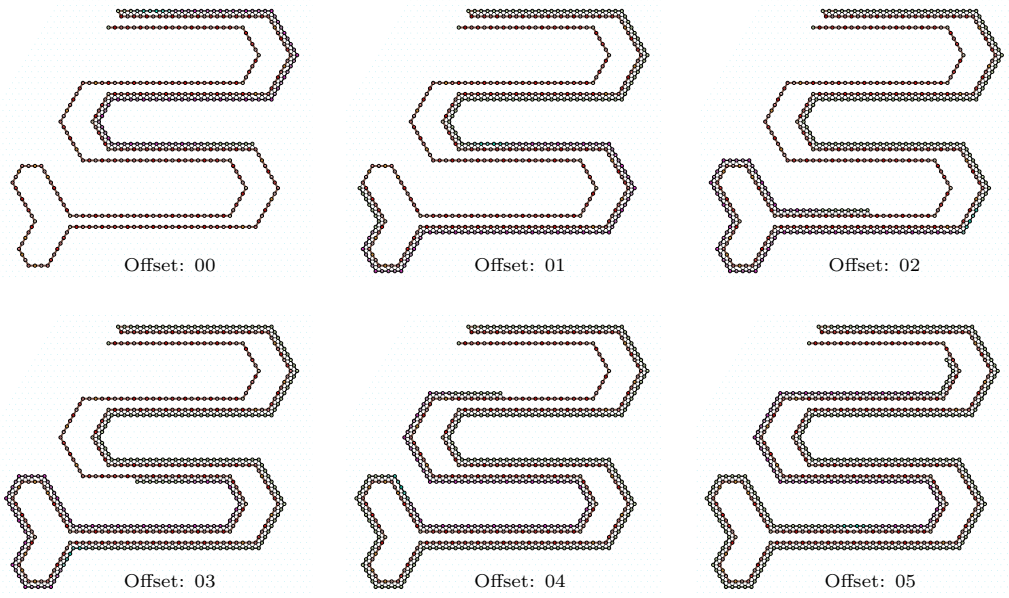


Figure 51 Self-rephasing speedbump on full red surface

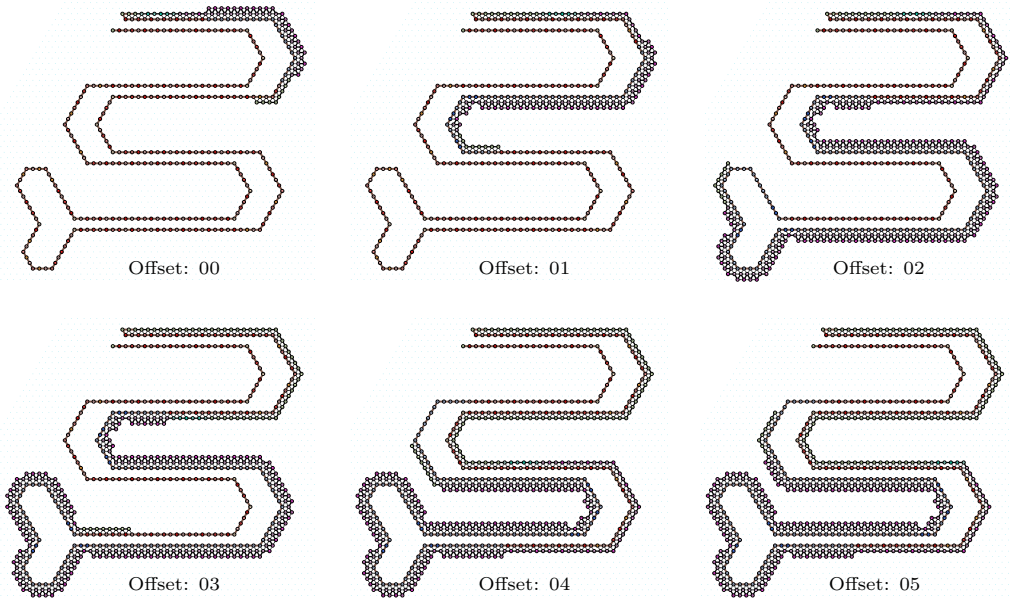


Figure 52 Self-rephasing speedbump on full blue surface