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

Daria Pchelina (Дарья С Пчелина)
LIPN, Institut Galilée – Université Paris 13, France

Nicolas Schabanel
École Normale Supérieure de Lyon (LIP UMR5668 and IXXI, MC2), France

Shinnosuke Seki (関 新之助)
University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo 1828585, Japan

Guillaume Theyssier
Aix-Marseille Université, CNRS, I2M, Marseille, France

Abstract

Different models have been proposed to understand natural phenomena at the molecular scale from a computational point of view. 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. 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. 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 as a readable oritatami "higher-level" programming language to build readily oritatami "smart robots", using our explicit simulation result as a compiler.

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.

2012 ACM Subject Classification Computer systems organization → Molecular computing; Computing methodologies → Molecular simulation; Applied computing → Molecular structural biology; Theory of computation → Computability; Theory of computation → Complexity theory and logic

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.STACS.2022.23

Supplementary Material Radius-1 Turedo to Delay-3 Oritatami compiler [23]: `https://hub.darcs.net/turedo2oritatami/turedo2oritatami/` ; Oritatami simulator [22]: `http://perso.ens-lyon.fr/nicolas.schabanel/OSsimulator`.

Funding *Nicolas Schabanel*: CNRS MITI MoPrExProgMol; IXXI Molécal; CNRS MITI AMARP; CNRS MITI DNASALGO; CNRS INS2I Algadène.
Shinnosuke Seki (関 新之助): JSPS KAKENHI Grant-in-Aids for Scientific Research (B) No. 20H04141 and (C) No. 20K11672.

1 Introduction

A major trend of natural computing is the study of computational models inspired by molecular biology that are both theoretically rich and realistic enough to allow *in-vitro* implementations. Oritatami systems were introduced in [9, 10] 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* by [12], 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) [8]. 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 [21]. On the opposite, RNA origami was born first *in-vitro* and the oritatami system was created [11] 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 [11, 20] that oritatami systems are Turing universal. They can also build arbitrary shapes [4] modulo a small universal constant upscaling, as well as specific fractals [17]. However, oritatami systems remain notably challenging to design. Indeed, the only shapes that can be built by [11, 20] are space-time diagrams of cyclic tag-systems or 1D cellular automata; and [4] requires to hardcode the whole shape in the transcript. The new computational model introduced in this article (*turedo*) not only abstracts away the technical details of attraction rules and bead sequence of oritatami, but embraces the geometrical aspects of them, as opposed to the simulation of 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 [1]. A popular class of Turing machines in $\mathbb{Z}^2$ is that of turmites [16], 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

[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.

content around their position to decide their move (like in [1]), but are constrained to move only in a self-avoiding way.

Both our models (oritatami and turedos) have two strong constraints: they are sequential 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 [19, 5] or freezing cellular automata [13, 2, 18]. 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 [3] or self-assembly tilings [5]. Theorem 1.1 is proved in section 3.

▶ **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.*

Complexity of limit configurations. The Turing universality results in [11, 20] 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 obtained limit configurations 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 [15]. Nevertheless, the construction used takes full advantage of the massive parallelism allowed in the aTAM model and *cannot* be translated into the turedo settings. Our construction is actually simpler than that of [15] and shows that sequential self-avoiding models can organize information in the plane in such a way that some regions allow 'uncomputable comebacks'.

▶ **Theorem 1.2** (Main result 2). *There exists a fixed oritatami with delay 3 and a fixed finite seed σ such that the produced limit configuration c^∞_σ 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 first for turedos and then for oritatami 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. Oritatami systems are embedded in the triangular lattice $\mathbb{T} = (\mathbb{Z}^2, \sim)$, where $(x, y) \sim (u, v)$ if and only if $(u, v) \in \cup_{\epsilon=\pm1}\{(x+\epsilon, y), (x, y+\epsilon), (x+\epsilon, y+\epsilon)\}$. Every position (x, y) in $\mathbb{T}$ is mapped in the euclidean plane to $x \cdot \vec{\mathsf{e}} + y \cdot \overrightarrow{\mathsf{sw}}$ using the vector basis $\vec{\mathsf{e}} = (1, 0)$ and $\overrightarrow{\mathsf{sw}} = \text{RotateClockwise}\left(\vec{\mathsf{e}}, 120^\circ\right) = (-\frac{1}{2}, -\frac{\sqrt{3}}{2})$. We will denote by $\overrightarrow{\mathsf{nw}}, \overrightarrow{\mathsf{ne}}, \vec{\mathsf{e}}, \overrightarrow{\mathsf{se}}, \overrightarrow{\mathsf{w}}, \overrightarrow{\mathsf{sw}}$ the six canonical unit vectors in $\mathbb{T}$. 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 $c_0c_1c_2\cdots$ in $\mathbb{T}$, 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 to be *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.

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 [11].

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

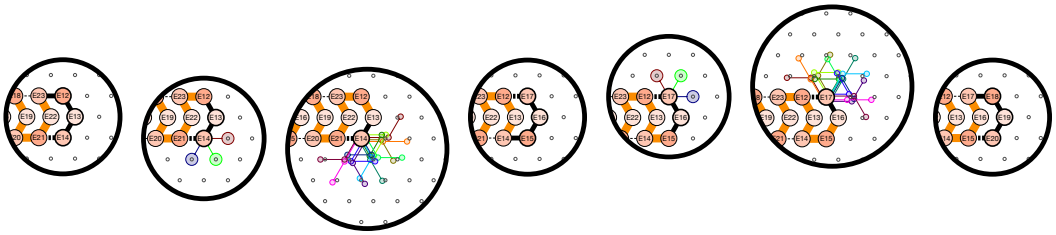


Figure 1 Oritatami model: From left to right, the growth from bead E12 to bead E18 of a self-supported oritatami glider with delay $\delta = 3$, transcript $p = \mathsf{E12}\dots\mathsf{E23}$ and rule $\{\mathsf{E12} \heartsuit \mathsf{E17}, \mathsf{E14} \heartsuit \mathsf{E21}, \mathsf{E18} \heartsuit \mathsf{E23}, \mathsf{E20} \heartsuit \mathsf{E15}\}$. At each step, the set of nascent paths and maximizing the number of bonds is shown. The nascent beads are highlighted in bold black. The nascent paths are drawn in bold black until the last bond made and ends in colors when their tail is free to move (i.e., is not bounded by any bond).

every subset S of partial configurations of length ℓ elongating σ of the sequence $s \cdot p$ to the subset $\mathscr{D}(S)$ of partial configurations of length $\ell + 1$ of $s \cdot p$ as follows:

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

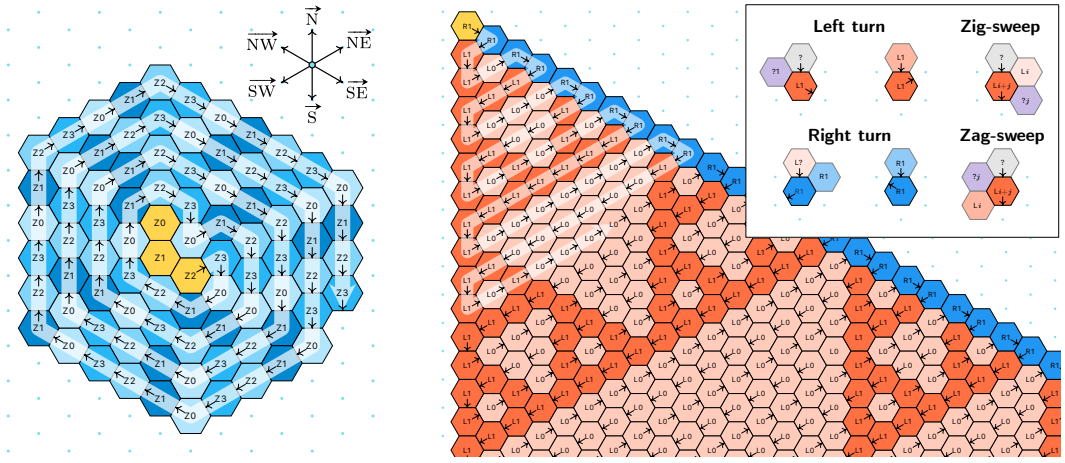
The possible configurations at time t of the oritatami system $\mathcal{O}$ are the elongations of the seed configuration σ by t beads in the set $\mathscr{D}^t(\{\sigma\})$.

We say that the Oritatami system is *deterministic* if at each 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}, that is at the position that maximises the number of bonds that can be made in a δ-elongation of c^t. Figure 1 illustrates the folding steps of a delay-3 oritatami glider.

Turedos: Self-avoiding Turing Machines. A *turedo* is a Turing machine working on the plane with a lookup neighborhood (like in [1]), that can only move in a self-avoiding way. Turedos are embedded in the hexagonal lattice $\mathbb{H} = (\mathbb{Z}^2, \dot{\sim})$ whose 6 unit vectors are $N_H = \{\vec{\mathrm{N}} = (1,1), \vec{\mathrm{NE}} = (1,0), \vec{\mathrm{SE}} = (0,-1), \vec{\mathrm{S}} = (-1,-1), \vec{\mathrm{SW}} = (-1,0), \vec{\mathrm{NW}} = (0,1)\}$. Note that $\mathbb{H}$'s underlying grid is rotated by 30° with respect to $\mathbb{T}$'s . This choice is motivated 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. We 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 and state of the head). The turedo* $\mathcal{T}$



(a) A 4-cyclic clockwise walker turedo **(b)** A 3-states Sierpinski triangle turedo

Figure 2 Two examples of radius-1 turedos. The seed configurations are displayed in yellow. The path followed by the turedo is highlighted in white. Empty (blank) positions are marked with a blue dot. **(a)** The turedo exits to the counterclockwise-most empty cell starting from its entry side. The states are just cycling in $\mathsf{Z0}, \ldots, \mathsf{Z3}$. **(b)** The turedo uses three states $\mathsf{L0}, \mathsf{L1}, \mathsf{R1}$ to perform a zigzag sweeping drawing the Sierpinski triangles pattern using the local rule given above.

induces a global map $F_{\mathcal{T}} : \mathcal{S}_{\mathcal{T}} \to \mathcal{S}_{\mathcal{T}}$ *defined as follows:*

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

where $(q', d, a) = \delta(q, c_z(r))$ *and image configuration* c' *is defined by:* $c'(z) = a$ *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. Two examples of radius-1 turedos are given in Figure 2.

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 $(c^t, z_t, q_t) = F_{\mathcal{T}}^t(s)$ for $t \in \mathbb{N}$. By the self-avoiding property, it holds that for any $z \in \mathbb{Z}^2$ the sequence of symbols $(c^t(z))_{n \in \mathbb{N}}$ 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 [14], we have that the sequence of configurations $(c^t)_t$ 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 $c_s^\infty(z)$ (in particular, $\tau_s(z) = 0$ if $c_s^\infty(z) = \bot$).

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 sweeping movement which allows us to embed any 1D Turing machine/cellular automaton computation (see Fig. 2b). They can also be used as thick wires to transport information from one region to another.

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 [3, 6, 2]. More precisely, each cell of the simulated system is represented by a macrocell in the simulator system, the macrocells 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 macrocells that are filled with the same high enough density.

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 [20], our simulation proceeds in three phases: 1) reading the neighboring letters, 2) preparing for writing the new letter on the boundaries of the macrocell 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, as the exit direction has to be deduced from the symbols read, the reading process must be non-blocking. Thus we cannot use the reading mechanism in [20], nor the writing flip-flap mechanism which would block any further return to a previously visited border; we cannot use its hardcoded exit mechanism either. 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 park unused information on the boundary of the macrocell as in [20], but need to store information *inside* the macrocell to avoid increasing the macrocell side length uncontrollably. Similarly the speedbump module introduced in [20] 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. Our current implementation `turedo2oritatami` uses 1735 bead types. Examples of radius-1 turedos compiled as oritatami as well as a fully functional `python` compiler can be downloaded from [23]: `https://hub.darcs.net/turedo2oritatami/turedo2oritatami/python`. The resulting `.os` files are to be run with the oritatami simulator by [22].

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.

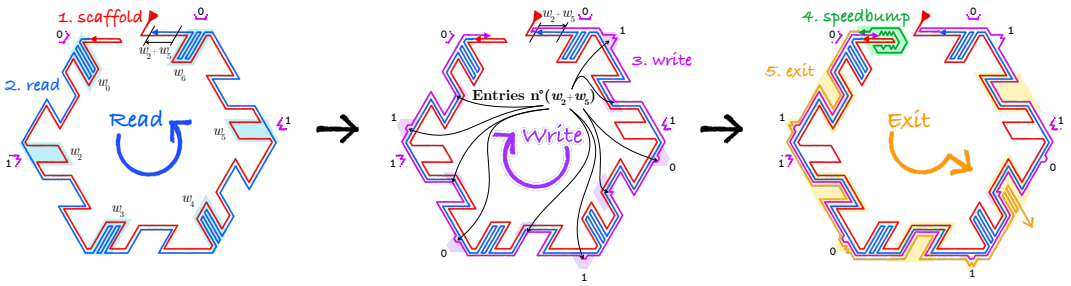


Figure 3 Principle of the macrocell operation. The shift of the reading layer at the end of its folding (and thus of the writing layer) is $\sum_{i:\text{bit read}_i=1} w_i = w_2 + w_5$.

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 $F : (2^q)^6 \to 2^q \times \{0, \dots, 4\}$, that reads the q bits $b_{i,0}, \dots, b_{i,q-1}$ 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.[2] 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 the exit direction.

Principle of the macrocell operation. Fig. 3 presents a schematic overview of the key operations in the macrocell. The transcript consists of 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 equal 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);
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 folding of the selected transition table entries. The shift Δ accumulated by the read layer allows then to write the 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.

[2] 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.

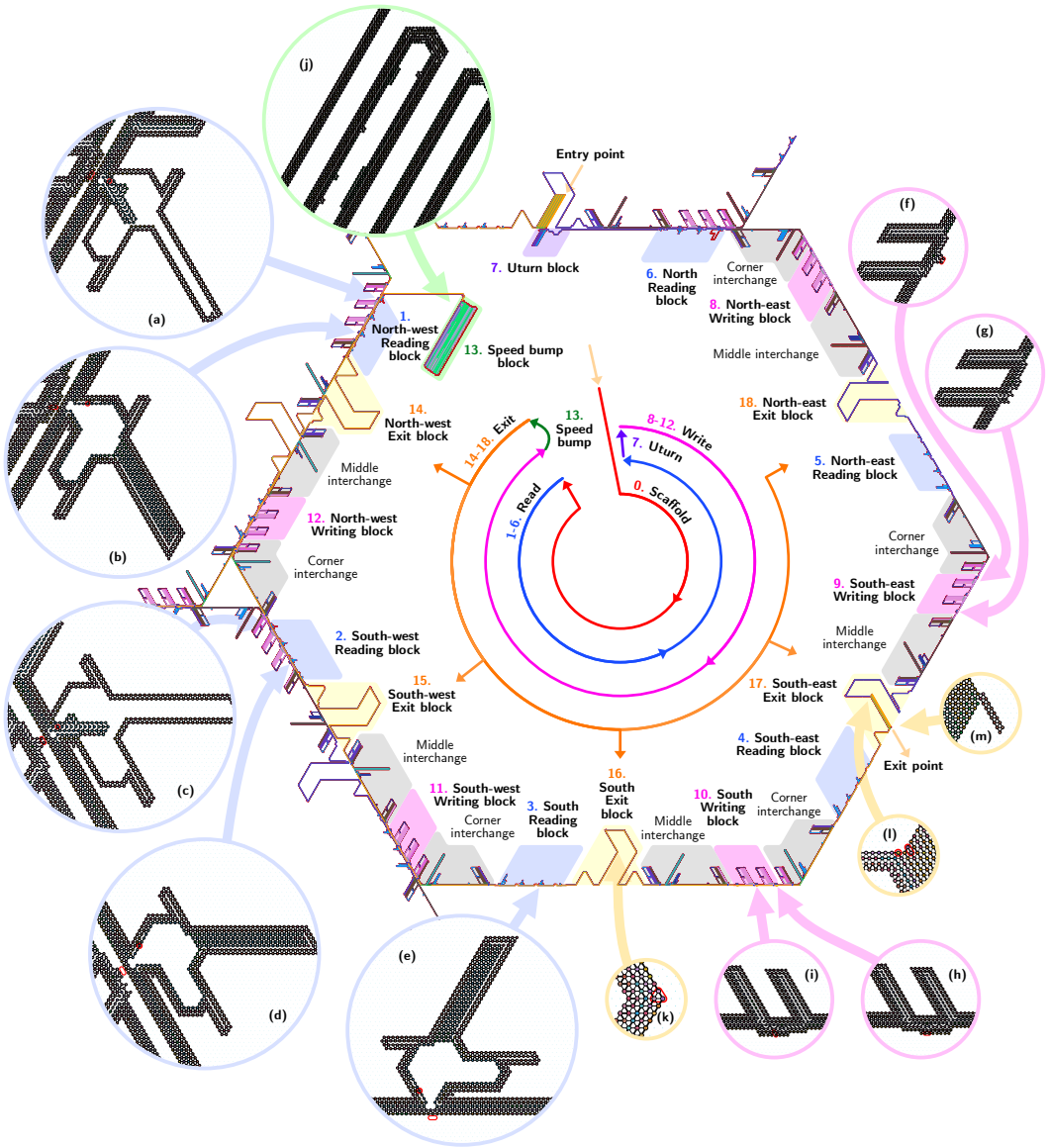


Figure 4 A macrocell for a turedo with $q = 3$ bits ($Q = 8$ tape symbols) together with the order in which layers and modules are used along its boundary as well as snapshots of important modules: (a)-(e) the read pocket in all possible situations: reading a 0/⊥ (fig. b, d and e) or a 1 (fig. a and c) from a neighboring cell (fig. a–d) (or not (fig. e)) and through its exit layer (fig. a and b) or directly from its write layer (fig. c and d) – (f)-(h) all possible situations for the write module: writing a 0 (fig. g and i) or a 1 (fig. f and h), through the exit layer (fig. h and i) or directly (fig. f and g) – (j) the shift-absorbing speedbump – (k) the exit layer folds along the exit pocket – (l)-(m) the write layer has placed the kicking bead ↻76 in the corner that detaches the exit layer from the pocket and concludes the folding by exiting to the SW. Zoom in for details

Observe that the reading layer needs to "read" the bit from neighboring cells while still making room for the two next layers to fold between the reading 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. 4. We will now present some of the key tools used in our design.

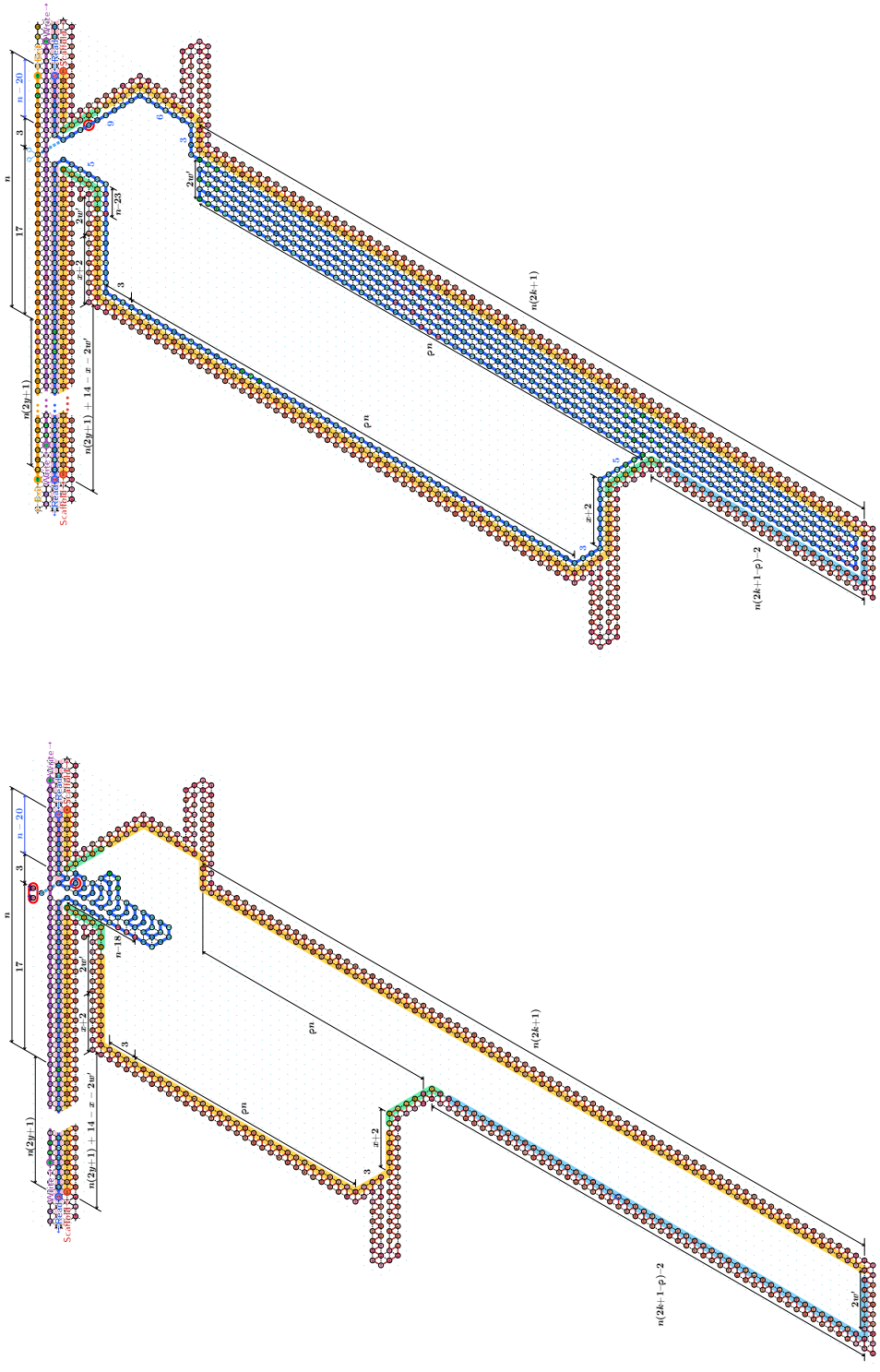
Folding meter and pockets. Our construction relies on two new simple and powerful tools:

- 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 (see Appendix A).
- 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 conditioned 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 two beads encoding a 1 in its neighborhood (see Fig. 4(a–e)), otherwise it folds into an hardcoded glider and exits the pocket rightaway.

Furthermore, several folding meters can be layered on top of each other in opposite directions 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. 4(f–i) because its bonds are weaker with the read layer when the latter is desynchronized after having been sucked into the pitfalls that surround this area. The length n of the each segment is a priori arbitrary as long as $n \geqslant 6$. In the present design, n was set to 26 in order to accommodate all the desired configurations — the most demanding being the glider at the entrance of a read pocket when reading a 0, and the writing of 0 or 1 over the write module.

Read layer and pocket. Let us illustrate the folding meter/pocket mechanism with the read pocket used to induce a shift of $2n\kappa$ in the transcript every time it reads a 1 with bit-weight κ. The primary purpose of the read pocket is namely to read a bit (0/1) and to push forward the read layer by the amount equivalent to its capacity if the read layer reads a 1. The read layer folds from right to left. When the read layer reaches the entrance (see Fig. 5a), its "reading head", the bead rq88, "senses" whether there is a 1 written on the adjacent macrocell at this location. If there is a 1, encoded by the presence of the pair of beads 凵p62 and 凵p64 in Fig. 5a, then the reading head rq88 is momentarily attracted upwards (making two temporary bonds), which results in placing bead rq86 away from the border of the read pocket; the read layer gets then too far from the read pocket border to get attracted to it anymore, and folds into its natural hidden shape: a glider that will immediately escape from the read pocket (Fig. 5a). Otherwise, if there is a 0, encoded by the absence of these bead types at the expected location in Fig. 5b, the reading head rq88 gets attracted downwards inside the pocket by making one bond; the read layer pursues its course downwards along the border until it reaches the bottom of the pocket which does not attract it anymore; the read layer prefers then to fold upon itself into a switchback pattern, filling the pocket completely, until it reaches the other side of the pocket which attracts it again; the read layer follows this border until it exits the pocket. This results in a shift forward of the read layer by an amount corresponding to the pocket capacity $2n\kappa$ if and only if the bit written on the adjacent macrocell is 1.

Remark that this novel bit reading method, using a reading head, does not obstruct the



(a) Read pocket reading 1.

(b) Read pocket reading 0.

Figure 5 Read pocket. 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 $\rho < 2k + 1$ as capacity $= 2n\left(w(2k+1) + \rho - 1 + \frac{(w+19).\mathsf{nextMultipleOf}(n)}{n}\right)$.

way between adjacent cells unlike the method used in [20]; 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.

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. 5), 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.

The read and write blocks. Fig. 6 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 each write module. The write module (responsible for writing `0` or `1` on the sides of the macrocell) and write pocket (responsible for hiding the unused entries of the transition table) are presented in details in Appendix B.

Layer interchange. Each layer is heavily interacting with its neighboring layers inside a macrocell. It follows that unwanted interferences may occur between facing layers from neighboring macrocells. For this purpose, we use three different variants of bead types in each layer: one for each half of each side (e.g. Read1 and Read2 for the read layer), plus one in the middle and the corners which interconnects one to the other and cancel the need for interactions between them (e.g., Read12 to switch between Read1 and Read2).

Setting up the exit block and computing the macrocell size. 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-sides long. Since the exit pocket belongs to the macrocell side, extending the exit pocket extends the macrocell side as well: we have thus to solve a fix point problem. Moreover, since a different amount of the exit layer will fold into each exit pocket depending on its location in the macrocell, we need a mechanism to make sure that, in all cases, the transcript will exit at the same position on every 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 (see the two triangles of varying depth surrounding each of the five exit pockets in Fig. 4). Making the exit pocket deep enough allows to solve this fix point issue, which concludes the proof overview of Thm. 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 [20, 11] *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

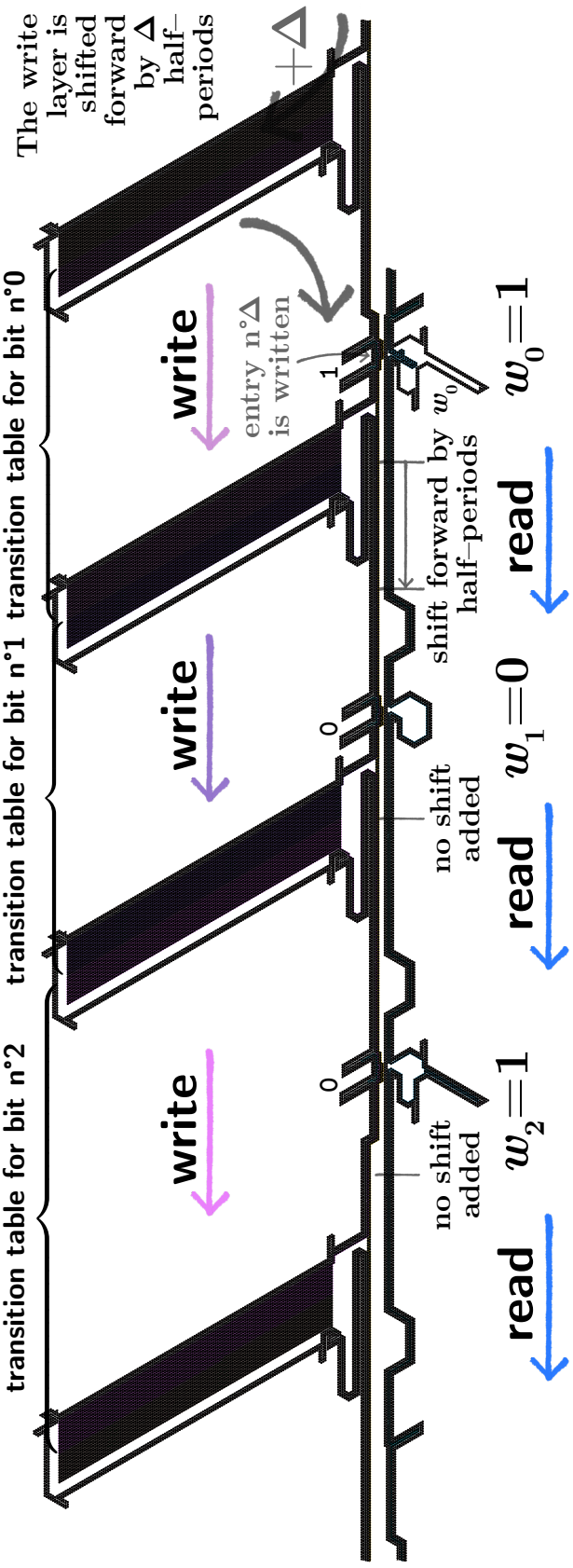
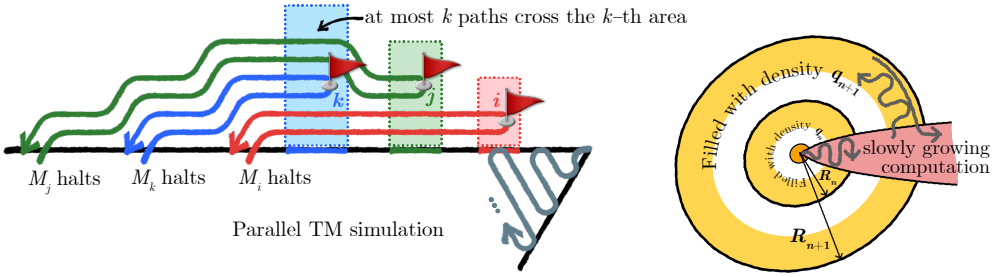


Figure 6 The read and write blocks: (bottom) when the read layer folds, it fills every read pocket facing a 0 on the neighboring macrocell, but skips the read pocket if it faces a 1, which increases the shift forward of the read layer by the weight of the corresponding bit; this yields a total additional 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 $(\mathcal{W}+1)$-long 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



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

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:

▶ **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. 7a, 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 in i position (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. 7a, no more than i paths will go across the area $p(i)$, i.e. the ones that correspond to the halting Turing machines j 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 $O(i^2)$. It follows that these areas 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 7a is a simplification and does not represent all movements of the turedo's head: in particular, when

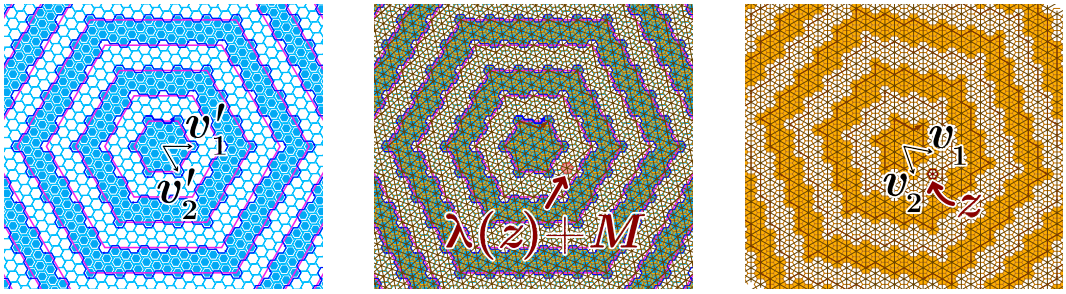


Figure 8 The linear map λ between the turedo world and the oritatami world induces a tilt between the concentric balls in the both worlds. This simulation tilt must be compensated by providing to the simulated turedo a pair of vectors as an input, so that it fills a proper discretization of the oritatami 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).

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. Using our simulation framework, Theorem 1.2 follows 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:

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

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 [7]). The problem is that, in a simulation, the lattice of cells is distorted into a macro-lattice of macro-cells in such a way that the macro-balls do not have the same shape as genuine balls, as shown in Fig. 8. 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.

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). *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.

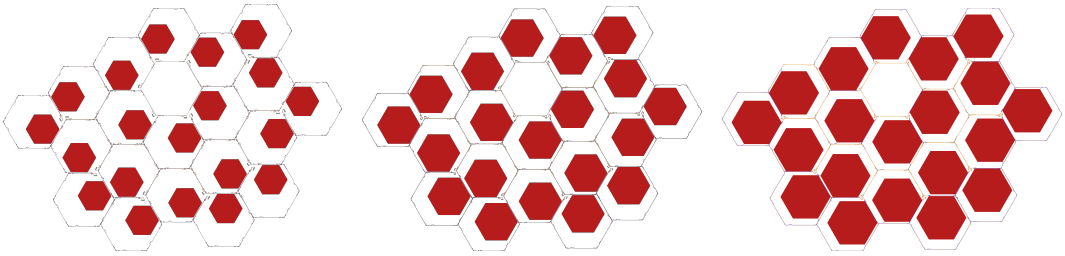


Figure 9 Increasing the density by folding a filled hexagon inside the macrocell expanded by 50, 100 and 200 extra $2n$-periods on each side. Actual oritatami simulation Zoom in for details

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. 8), 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. 9). Picking L large enough concludes the proof. The case of density 1 is treated separately by an ad hoc solution.

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. 7b): 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 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.

2 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.

3 Marianne Delorme, Jacques Mazoyer, Nicolas Ollinger, and Guillaume Theyssier. Bulking ii: Classifications of cellular automata. *Theor. Comput. Sci.*, 412(30):3881–3905, 2011.

4 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.

5 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.

6 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.

7 Erling Følner. On groups with full banach mean value. *MATHEMATICA SCANDINAVICA*, 3:243, dec 1955.

8 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.

9 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.

10 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.

11 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.

12 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.

13 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.

14 Petr Kůrka. *Topological and symbolic dynamics.* Société Mathématique de France, 2003.

15 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.

16 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.

17 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.

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

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

20 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.

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

22 Nicolas Schabanel. Simple OS simulator, 2021. `http://perso.ens-lyon.fr/nicolas.schabanel/OSsimulator/`.

23 Nicolas Schabanel and Shinnosuke Seki. Turedo to oritatami compiler, 2022. `https://hub.darcs.net/turedo2oritatami/turedo2oritatami`.

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.

A Transcript, Folding meter and Pocket

In our design, the oritatami transcript is periodic, and one period folds into one macrocell. The period is divided semantically in five parts:

$$\text{Transcript} = \text{Scaffold} \cdot \text{Read} \cdot \text{Write} \cdot \text{Speedbump} \cdot \text{Exit}$$

Scaffold hardcodes the "skeleton" of the macrocell and folds clockwise. Read folds around the scaffold counterclockwise. It reads the states of the adjacent macrocells, which induces a shift of the transcript equal to the total weight of the bits read with value 1. Write folds on top of the Read layer clockwise, and, according to the shift read, writes the bits to be output on each side and marks the exit side. Speedbump annihilates the shift using a process similar to [20]. Finally, Exit folds on top of the Write layer counterclockwise until it reaches the "exit mark" that has been placed by the Write layer.

Each of the Read, Write, Exit layers have the same periodic structure that we call a folding meter. A *n-folding meter* is a $4n$-periodic transcript with a period R of the form:

$$\begin{aligned}\mathsf{R} = {} & \mathsf{Rt}_0,\ \mathsf{Rt}_1,\ \mathsf{Rt}_2,\ \mathsf{Rp}_3,\ \ldots,\ \mathsf{Rp}_{n-1},\ \mathsf{Rb}_n,\ \mathsf{Rb}_{n+1},\ \mathsf{Rb}_{n+2},\ \mathsf{Rq}_{n+3},\ \ldots,\ \mathsf{Rq}_{2n-1},\\ & \mathsf{Rt}_{2n},\ \mathsf{Rt}_{2n+1},\ \mathsf{Rt}_{2n+2},\ \mathsf{Rp}_{2n+3},\ \ldots,\ \mathsf{Rp}_{3n-1},\ \mathsf{Rb}_{3n},\ \mathsf{Rb}_{3n+1},\ \mathsf{Rb}_{3n+2},\ \mathsf{Rq}_{3n+3},\ \ldots,\ \mathsf{Rq}_{4n-1}\end{aligned}$$

where the letters t and b stand for *top* and *bottom*. The internal interactions are:

$$\underbrace{\mathsf{R}_i \heartsuit \mathsf{R}_{-i-1},\quad \mathsf{R}_i \heartsuit \mathsf{R}_{-i-2},}_{\text{Going down}}\quad \underbrace{\mathsf{R}_{n+i} \heartsuit \mathsf{R}_{n-i},\quad \text{and } \mathsf{R}_{n+i} \heartsuit \mathsf{R}_{n-i-1}}_{\text{Going up}}\quad \text{for all } i. \tag{1}$$

They ensure that the folding meter will either:

- *follow a border* if all of its beads bind to every bead on the border. Two examples are the Read layer along the right border in Fig. 5, and the Write layer along the left border in Fig. 11. This process allows as well to stack several n-folding meters on top of each other in opposite direction and that are in-sync, i.e. such that one's p-parts face other's q-parts. As an example, observe the folding of the three layers Read, Write and Exit at the top right of the write pocket in Fig. 11;
- or *fold upon itself in the manner of the "folding meter" tool*, when reaching the bottom of a *pocket* that no longer attract the layer. Two examples are the Read layer in Fig. 5b or the Write layer in Fig. 11. Indeed, in Fig. 11, the beads rp15..12 do not attract the beads Wb26..28 which thus fold upwards thanks to the interactions listed in Eq. (1) yielding to a switchback pattern that continues until the Write layer reaches the right side, to which the Write layer is attracted and thus resumes following the border.

As the binding between the switchbacks of a n-folding meter are strong, they can flatten any custom interactions encoded internally in either the p- or the q-beads of an n-folding meter as long as these interactions are weak, i.e. do not involve more than 3 bonds per bead. This allows us to hide or expose on-demand specific behaviors when the binding with the lower layer is weak enough: for instance, in Fig. 10, the binding between the p-parts of the Write and Read layers is weak enough to let the p-part of the Write layer fold into various two-spikes patterns encoding 0 or 1 that flatten anywhere else in the macrocell.

In this article, $n = 26$. Note that each folding meter is essentially $2n$-periodic with period $(\mathsf{t}, \mathsf{p}, \mathsf{b}, \mathsf{q})$. This $2n$-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 $2n$, half-period of the folding meter, and not a full period. Furthermore every bead type Ri 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, R$\overline{\mathsf{i}}$ will refer to either bead types Ri or $\mathsf{R}(i + 2n)$; for instance R$\overline{12}$ refers to both bead types R12 and R64.

Notations. For any pair of 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)

B The write block

As seen in Fig. 6, the *write block* on each side of a macrocell consists in an alternation of $q+1$ write pockets of capacity $2n\mathcal{W}$ beads, and q write modules, one for each of the q bits to be written. Each write pocket hides the $\mathcal{W}$ entries of the $(\mathcal{W}+1)$-long transition tables that are unused, while the write module writes the selected entry. Let us start by describing the write module.

B.1 Write module

Write modules are the places where the oritatami writes the bits output on each side. 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. Precisely, this module places two beads of special type (circled in red in figures) at a designated readable site to write a `1` (Figs. 10a and 10b), or deliberately out of the site so that they cannot attract the reading head no matter what types they are to write a `0` (Figs. 10c and 10d).

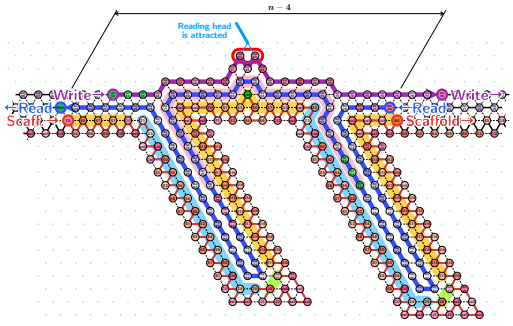
Depending on whether the module is located before of after the side by which the oritatami will exit the macrocell, the module will be covered or not by the Exit layer, leading to the 4 possible configurations in total presented in Fig. 10. As the side by which the oritatami system will exit is known as soon as the states of the neighboring macrocell are read, we can use, in the Write layer description, the appropriate variant of the encoding for each bit on each bit according of the relative position of the module with the exit direction to be taken:

- In the case where the Write layer is to be covered, the Write layer folds into two spikes either to the left (Fig. 10b) or the right (Fig. 10d) of the blue hill at the center of the module; then, when the Exit layer folds from right to left, the two special bead types ★$\overline{32}$ and ★$\overline{33}$ in the Exit layer will either be placed at the top of the hill (where they will attract the reading head) or hidden in the right side of the hill (where they cannot attract the reading head), which will be interpreted by the facing macrocell as a `1` or a `0` respectively.
- Otherwise, the bit `0`/`1` is encoded directly either by two big spikes bearing the special attracting bead types $\llcorner\!\lrcorner\overline{10}$ and $\llcorner\!\lrcorner\overline{12}$ at the top of the hill, which will be read as a `1` (Fig. 10a), or by two spikes to the left of the hill leaving the designated site empty, which will be read as a `0` (Fig. 10c).

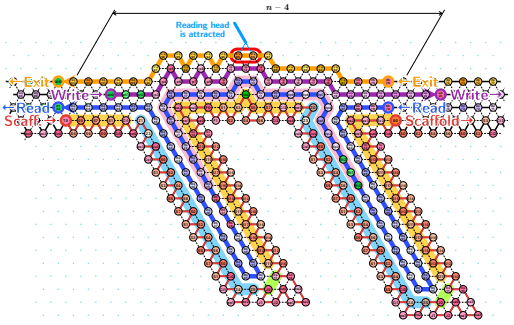
In order for the Write layer to adopt these peculiar configurations, we need it to take its independence from the underlying Read layer. This is accomplished thanks to the two pitfalls surrounding the write module. Each of them have a capacity of n beads exactly, which induces a phase difference of n between the Read and the Write layers, resulting in the p-part of the Write layer to fold on top of the p-part of Read layer over the write modules. These two p-parts have specific interactions between them allowing the specific configurations to be folded there and only there.

B.2 Write pocket

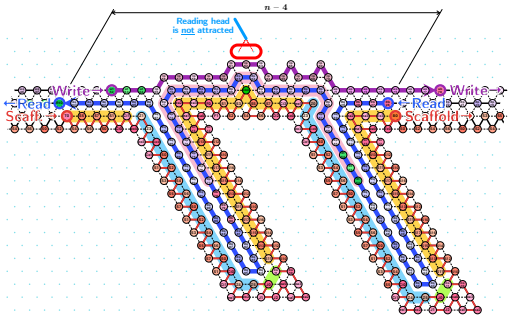
Write pocket operation. Its primary purpose is to hide the $\mathcal{W}$ unused entries of the $(\mathcal{W}+1)$-long transition tables as can be observed in Fig. 6. These pockets are placed between



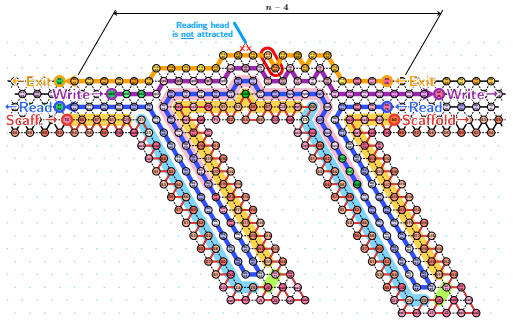
(a) *Write module – Top variant:* the Write layer writes a 1 by forming two spikes on the top of the module, with two active beads aligned with the reading head of the adjacent macrocell.



(b) *Write module – Left variant:* the Write layer prepares for writing a 1 by forming two spikes to the left of the module so that the two active beads of the Exit layer get aligned with the reading head of the facing macrocell.



(c) *Write module – Right variant 1:* on a side located before the exit that will be taken later, the Write layer writes a 0 by forming two spikes to the right of the module; 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 facing macrocell.



(d) *Write module – Right variant 2:* the Write layer prepares for writing a 0 by forming two spikes to the right of the module, so that the two active beads of the Exit layer get misaligned with the reading head of the facing macrocell.

Figure 10 The four variants of the Write module: **(a,b)** writing a 1 and **(c,d)** writing a 0.

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 others are hidden in the write pockets. The Read layer does not fill the write pocket but simply "coats" its border. The Write layer, however, enters it and folds inside into its compact switchback form, hiding away the $\mathcal{W}$ unused entries of each transition table, encoded each in one p-part of the Write layer. Finally, the Exit layer simply jumps over it by folding into a hardcoded bridge to get across its entrance.

As for the read pocket, write pockets are also used in the interchange blocks to switch between the three variants of the write layer Write1 ↔ Write12 ↔ Write2 (see Fig. 4).

Write pocket design. This pocket differs from the read pocket in three 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 enter the pocket from opposite directions; 3) only the Write layer fills the pocket: the Read layer must not fill the pocket. It follows that:

- Been "coated" by the Read layer, both sides of the write pocket entrance will attract the Write layer. To avoid unwanted interactions, we need thus to make the entrance wider. It follows that the Exit layer will not be able to jump over the entrance as the Write layer

does over the read pocket. Fortunately, as the Exit layer is never shifted, we can hardcode a bridge $\mathsf{G}\overline{4..16}$ in this layer at this precise location to solve this issue (see top of Fig. 11).

Furthermore, as the Read layer will enter the pocket with an arbitrary shift, the interactions of the pocket with the Write layer cannot be directly hardcoded. As illustrated in Fig. 5, a pocket has essentially three kinds of interactions with the layer that fills it: 1) full attraction (highlighted in yellow), 2) no attraction (in blue), and 3) localised specific attractions (in green). We create similar interactions using two mechanisms: A) two pitfalls are located at the bottom left and at the middle right of the pocket (see Fig. 11) that introduce and then cancel a phase difference between the Read and Write layers; B) the bottom and bottom right borders of the pocket are moved out of reach of the Write layer switchbacks. A) and B) ensure at the bottom right that the Write layer is not

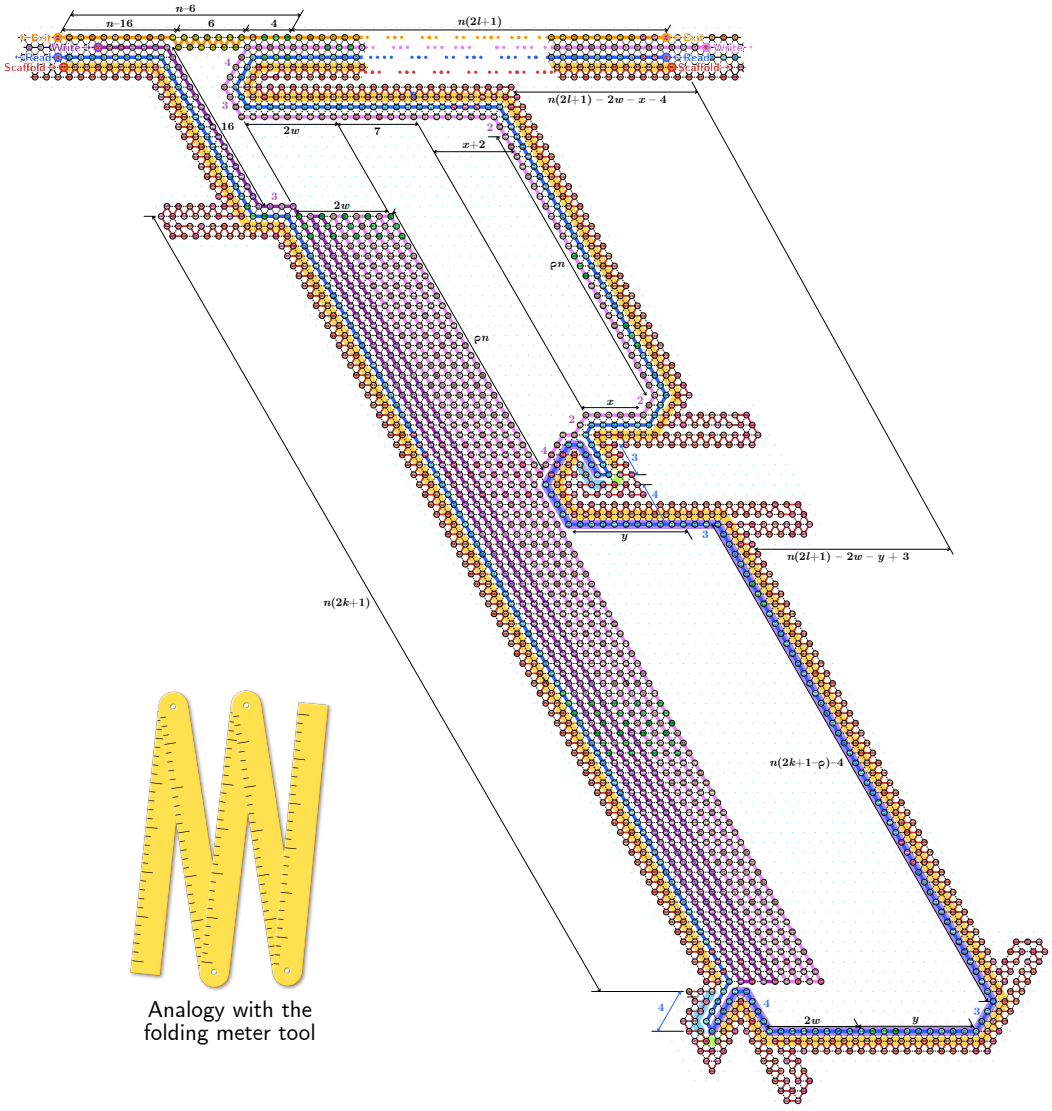


Figure 11 Write pocket. Given w, parameters x, y and ℓ must be adjusted so as to match the period of the folding meter. The part of the Read layer which is desynchronized with the Write layer is highlighted in purple.

attracted by the bottom nor opposite border while it folds in switchbacks. Specifically programmed interactions between the in-sync write bead types @b$\overline{78..80}$@q$\overline{81..83}$ and the out-of-sync read bead types rp$\overline{24..25}$rb$\overline{26..28}$rq$\overline{29}$ that appear just before the second pitfall in the middle of the right border, ensure that the Write layer glues back to the border once it has ended its switchback pattern and not earlier.

Fitting all these constraints together contributes to the choice of $n = 26$ for the period of our folding meters.

Geometry. The pocket geometry is determined by four integer parameters: its width w, height k, remainder $\rho < 2k + 1$, and extension ℓ. The two "bubbles" to the right of the pocket are used to keep synchronised the three layers Read, Write and Exit. Their sizes are determined by two extra integer parameters x and y which are adjusted as follows:

(a) The length of the Write layer path from Wt$\overline{0}$/Wb$\overline{26}$ to Wt$\overline{0}$ 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 total length is $2(x + w + 17) + \rho n$; thus, x should be set to: $\mathbf{x} = (\mathbf{w} + \mathbf{17}).\mathsf{complement}(\mathsf{to}\colon n)$.

(b) The length of the Read layer path from Rt$\overline{0}$/Rb$\overline{26}$ to Rb$\overline{26}$ 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 total length is $2(w + y + 12) + (2k + 1 - \rho)n$; thus, we set: $\mathbf{y} = (\mathbf{w} + \mathbf{12}).\mathsf{complement}(\mathsf{to}\colon n)$.

(c) Lastly, in order to avoid collision with other modules, we set the extension ℓ so that the pocket module ends to the right of the two bubbles, that is: $\ell = (\mathbf{2w} + \max(\mathbf{x} + \mathbf{8}, \mathbf{y} - \mathbf{3})).\mathsf{nextMultiple}(\mathsf{of}\colon 2n)/\mathbf{2n}$.

Capacity. The *capacity* of a write pocket is defined as the length of the path taken by the Write layer from the leftmost Wt$\overline{0}$ to the rightmost Wt$\overline{0}$, and it is determined by the three independent parameters k, w, ρ with $\rho < 2k + 1$, and one dependent parameter ℓ as:

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

Building a write pocket with a given capacity. Various parameters can yield the same capacity, thus we aim for the parameters that yield the shortest transcript. The length of the transcript is given by the Read layer, whose asymptotic length is $\sim 4nk + 6w$. Minimizing this value subject to a fixed asymptotic capacity of $2w(2k + 1)n$ yields to the ideal ratio of $w \sim 2nk/3$. Now, to obtain a write pocket of target capacity $2n\mathcal{W}$ we proceed as follows:

- solving $\mathsf{capacity}(w = 2nk/3, k, \rho = 0, \ell = 0) = 2n\mathcal{W}$ yields a suggested value for k of:

$$k := \max\left(0, \left\lfloor \frac{\sqrt{12n\mathcal{W} + n^2 + 4n - 224}}{4\,n} - \frac{1}{2n} - \frac{1}{4} \right\rfloor\right)$$

- we then set w by solving $2n\mathcal{W} = \mathsf{capacity}(w, k, \rho = 0, \ell \sim w/n)$ which yields:

$$w := \max\left(1, \left\lfloor \frac{n\mathcal{W} - 19}{n(2k + 1) + 2} \right\rfloor\right)$$

- x, y, and ℓ are then computed according to the formulas (a), (b), and (c).
- we conclude by setting:

$$\rho := \max\left(0, 2n\mathcal{W} - \overbrace{\left(2n((2k + 1)w + 1) + 2(w + 17).\mathsf{nextMultiple}(\mathsf{of}\colon n) + 2n\ell\right)}^{\mathsf{capacity}(w,k,\rho=0,\ell)}\right) \Big/ n,$$

if $\rho > 2k$, then rerun the two last steps with $w := w + 1$.

This ensures that: $2n\mathcal{W} \leqslant \mathsf{capacity}(w, k, \rho, \ell) \leqslant 2n(\mathcal{W} + 2)$ and that k, w, and ℓ are $O(\sqrt{\mathcal{W}})$.