Oritatami simulates self-avoiding 2D Turing machines

Daria Pchelina¹, Nicolas Schabanel², Shinnosuke Seki³, and Guillaume Theyssier⁴

1 École Normale Supérieure de Paris, Paris, France
2 École Normale Supérieure de Lyon (LIP UMR5668, MC², ENS de Lyon), France
3 University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo 1828585, Japan
4 Aix-Marseille Université, CNRS, I2M, France

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. A self-avoiding Turing machine on the plane (called turedo) is a machine 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 conversely that any turedo with lookup radius 1 can be simulated by an oritatami. 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, the formalism of turedo can be used as a higher level programming language to build oritatami “smart robots”, using our explicit simulation result as a compiler.

As an illustration, we prove two new complexity results on the limit (infinite) configurations that can be produced starting from finite initial configurations, both for turedos of radius 1 and oritatamis. First, we show that such limit configurations can embed any recursively enumerable set, and thus can be uncomputable. Second, we characterize the possible densities of occupied positions in such limit configurations: they are exactly the \( \Pi_2 \)-computable numbers between 0 and 1. We also show that all such limit densities can be produced by a single turedo by varying the finite initial configuration. The same holds for oritatami for densities bounded away from 1. None of these results are implied by previous constructions of oritatami embedding tag systems or 1D cellular automata, which produce only computable limit configurations with constrained density.

1998 ACM Subject Classification F.1.1 Models of Computation, F.2.2 Nonnumerical Algorithms and Problems

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

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p
1 Introduction

Oritatami systems were introduced in [6, 7] to investigate the computational power of molecular co-transcriptional folding as demonstrated in vitro by [9]. It 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 $\delta$ most recently produced beads are allowed 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 [10, 18] that oritatami systems are Turing universal. They can also build arbitrary shapes [4] modulo some upscaling, or specific fractals [15]. However, oritatami systems remain notably challenging to design. Indeed, the only shapes that can be built by [10, 18] are space-time diagrams of cyclic tag-systems or 1D cellular automata; and [4] requires to hardcode the whole shape in the transcript. In this article, we introduce a new computational model (turedo) for building 2D shapes and demonstrate that it can be simulated up to upscaling by oritatami systems. Our simulation allows thus to take full advantage of Turing computation 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 machine in $\mathbb{Z}^2$ are turmites [14] which are free to move on the plane but does 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 machine on the plane, that we call turedos\(^1\), which can look at the tape 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 a TAM model of self-assembly [17, 5] or freezing cellular automata [11, 2, 16]. 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 raidus 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].

The Turing universality results in [19, 10] 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 $c_\infty$ 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

\(^1\) 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.
that there are oritatami runs from finite seeds where there is no computable time bound (Corollary 6.2). 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 [13]. 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 [13] and shows that sequential self-avoiding models can organize information in the plane in such a way that some regions allows 'uncomputable come backs'.

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 \( \Pi_2 \)-computable numbers between 0 and 1 (Theorem 7.1 and Corollary 7.3), where \( \Pi_2 \)-computable means being the limsup of a computable sequence of rational numbers [20]. 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 number). We actually show that the whole spectrum of density can be obtained in a single turedo by varying the seed (Theorem 7.1). A corresponding but slightly weaker result is shown for oritatami (Corollary 7.3).

Note that the densities that can be produced in the aTAM model or freezing cellular automata from finite initial configurations cannot be more complex (one can check that the \( \Pi_2 \) upper bound of Lemma 7.2 still holds).

The organization of the paper is as follows: we first present oritatami and turedo models (sections 2 and 3), and the notion of simulation (section 4); then, we establish our main simulation result (section 5) and its two applications (sections 6 and 7).

## 2 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 \( c_0c_1c_2 \cdots \) in the triangular lattice \( \mathbb{T} \) 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 \( p \). The class of all the configurations obtained by applying an isometry of \( \mathbb{T} \) to a given configuration is called a conformation.

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 \( c^{\geq k} \) the set of all \( k \)-elongations of a partial configuration \( c \) of sequence \( p \).

### Oritatami systems

An oritatami system \( \mathcal{O} = (p, \mathcal{E}, \delta) \) is composed of (1) a (possibly infinite) bead type sequence \( p \), called the transcript, (2) an attraction rule, which is a symmetric

---

\( z \) is not visited after \( \tau(z) \) steps of the run, then it will never be visited.

The first application of our simulation result is to prove that we can produce uncomputable limit configurations from finite seeds with oritatami (Corollary 6.2). 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 [13]. 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 [13] and shows that sequential self-avoiding models can organize information in the plane in such a way that some regions allows 'uncomputable come backs'.

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

---

The organization of the paper is as follows: we first present oritatami and turedo models (sections 2 and 3), and the notion of simulation (section 4); then, we establish our main simulation result (section 5) and its two applications (sections 6 and 7).
relation $\mathcal{O} \subseteq B^2$, and (3) a parameter $\delta$ 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 any specific behavior) and at the same time allows arbitrarily long computation.\footnote{Note that we do not impose here a maximal number of bonds per bead (called arity).}

We say that two bead types $a$ and $b$ attract each other when $a \mathcal{O} b$. 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 $q_i, q_j$ and $|i-j| > 1$. The number of bonds of configuration $c$ of $q$ is denoted by $H(c) = |\{(i,j) : c_i \sim c_j, j > i + 1, \text{ and } q_i, q_j\}|$.

Oritatami dynamics. The folding of an oritatami system is controlled by the delay $\delta$. 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 $\delta$ beads in total. This dynamics is oblivious as it keeps no memory of the previously preferred positions [8].

Formally, given an Oritatami system $\mathcal{O} = (p, \mathcal{O}, \delta)$ and a seed configuration $\sigma$ of a seed bead type sequence $s$, we denote by $C_{\sigma,p}$ the set of all partial configurations of the sequence $s \cdot p$ elongating the seed configuration $\sigma$. The considered dynamics $\mathcal{D} : 2^{C_{\sigma,p}} \rightarrow 2^{C_{\sigma,p}}$ maps every subset $S$ of partial configurations of length $\ell$ elongating $\sigma$ of the sequence $s \cdot p$ to the subset $\mathcal{D}(S)$ of partial configurations of length $\ell + 1$ of $s \cdot p$ as follows:

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

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

We say that the Oritatami system is deterministic if at all time $t$, $\mathcal{D}^t(\{\sigma\})$ 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 $\mathcal{D}^t(\{\sigma\}) = \{c^t\}$ 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 in $c^{t+1}$ at the position that maximises the number of bonds that can be made in a $\delta$-elongation of $c^t$.

## 3 Turedos: Self-avoiding Turing Machines

A turedo is a Turing machine with lookup neighborhood (like in [1]) that can only move in a self-avoiding way. We fix the following set of elementary hexagonal\footnote{The triangular lattice for oritatami use orientation east-west while the set of elementary moves $N_H$ contain 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 hexagonal cells in turedos.} moves $N_H = \{ \overline{n} = (1,1), \overline{w} = (1,0), \overline{e} = (0,-1), \overline{w} = (-1,0), \overline{w}_{\overline{w}} = (0,1) \}$ in $\mathbb{Z}^2$ and denote by $B(r)$ the hexagonal ball of radius $r$, i.e. the set of position 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 $B(r)$.

▶ **Definition 3.1.** A turedo is defined by $\mathcal{M} = (A, b, Q, q_0, r, \delta)$ where:
Figure 1: 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 \( l_{t-1} = 3 \).

- \( A \) is the tape alphabet,
- \( b \in A \) is the blank symbol,
- \( Q \) is the set of head states with initial state \( q_0 \in Q \),
- \( r \) is the lookup radius,
- \( \delta : Q \times A^{B(r)} \to Q \times N_H \times A \setminus \{b\} \) is the local transition map.

A tape configuration is an element of \( A^{Z_2} \). A global state is an element of \( S_M = A^{Z_2} \times Z_2 \times Q \) (tape configuration, position of head and head state). The Turing machine \( M \) induces a global map \( F_M : S_M \to S_M \) defined as follows:

\[
F_M(c, z, q) = \begin{cases} 
(c, z, q) & \text{if } c(z) \neq b \text{ or } c(z + d') \neq b, \\
(c', z + d', q') & \text{else},
\end{cases}
\]

where \( (q', d', a') = \delta(q, z', \rightarrow c(z + z')) \) and \( c'(z) = a' \) and \( c'(z') = c(z') \) for \( z \neq z' \).

When the first case occur, 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.

The basic building block to design complex turedos is the zigzag (see Figure 1) movement which allows to embed any 1D Turing machine computation. They can also be used as thick wires to transport information from one region to another. A zigzag toolbox is detailed in appendix.

4 Simulations

Both oritatami and turedos are examples of a sequential hexagonal self-avoiding system. By this we mean a map \( F : S_M \to S_M \) acting on global states \( S_M = A^{Z_2} \times Z_2 \times Q \) for some alphabet \( A \) with blank letter \( b \) and state set \( Q \), and such that, for any \( (c, z, q) \in A^{Z_2} \), the image \( F(c, z, q) = (c', z', q') \) verifies

- either \( (c', z', q') = (c, z, q) \) (the system is blocked),
- or \( c(z) = c(z') = c'(z') = b \) and \( c'(z) \neq b \) and \( c(z'') = c'(z'') \) for all \( z'' \) with \( z'' \neq z \) (the configuration grows).

Such a system can only move to a position in a blank state and change the state of the leaved position to a non-blank state. Therefore when the system leaves some position it will never visit it again (hence the name of self-avoiding systems). Moreover, we say that such a
Oritatami simulates self-avoiding 2D Turing machines

system is continuous if there is some finite neighborhood $N \subseteq \mathbb{Z}^2$ such that $z' - z$ and $c'(z)$ depend only on $c_{z+N}$.

Taredos directly fit in that settings. For oritatami, it suffices to rotate their lattice by $\pi/2$ and consider as internal states the current position in the transcript modulo its period.

We can see sequential self-avoiding systems as sequential deterministic mechanisms that generate larger and larger shapes in space. Given an initial global state $s \in S_M$, let us consider the sequence $(c_n, z_n, q_n) = F^n(s)$ 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 $c_\infty(z)$, we then have defined a tape configuration $c_\infty \in \mathbb{Z}^2$ which is called the limit configuration reached by $F$ starting from $s$. Said differently, using the standard Cantor topology for tape configurations [12], we have that the sequence of configurations $(c_n)_n$ converges to $c_\infty$. 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 time steps $t$ for which the tape content of cell $z$ at time $t$ is $c_\infty(z)$. In the following definition, we formalize a notion of simulation that captures the ability of a system $S$ to reproduce all the growing shape dynamics of a system $S'$ up to a spatio-temporal re-scaling. More concretely, to each step of $S'$ that modifies the configuration at some position, $S$ responds in a constant number of steps by adding a constant size pattern in the corresponding position in some scaled-up hexagonal lattice made of hexagonal macro-cells. In this way, the precise evolution of $S'$ can be recovered from the evolution of $S$.

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 4.1.** We say that a sequential self-avoiding system $S = (A, b, Q, q_0, F)$ simulates another system $S' = (A', b', Q', q_0', F')$ if there exist:

- a space rescaling factor $r \in \mathbb{N}$, so that the hexagonal lattice in $S$ is tiled by macro-cells $B(r)$ forming a sub-lattice given by vectors $\{m(v) : v \in N_H \}$, and which induces by linearity a space rescaling map $m : \mathbb{Z}^2 \to \mathbb{Z}^2$;
- for each $a' \in A'$ a collection of tape patterns $P_{a'} \subseteq A^{B(r)}$ with $P_{a_1} \cap P_{a_2} = \emptyset$ whenever $a_1 \neq a_2$, and $P_B = \{b^{B(r)}\}$; this defines a tape decoding map $\phi : X \to (A')^{\mathbb{Z}^2}$ defined on the set $X$ of configurations $c \in (A')^{\mathbb{Z}^2}$ such that, $\forall z \in \mathbb{Z}^2$, $c_{m(z)+B(r)} \subseteq \cup a' \in A'P_{a'}$ as follows: $\phi(c)(z) = a'$ if $c_{m(z)+M} \in P_{a'}$;
- for each $q' \in Q'$ a collection of state patterns $R_{q'} \subseteq A^{B(r)} \times B(r) \times Q$ with $R_{q_1} \cap R_{q_2} = \emptyset$ whenever $q_1 \neq q_2$; this defines the set $X^+$ of valid global states $(c, z, q)$ which are those verifying: $c_{m(z')+B(r)} \subseteq \cup a' \in A'P_{a'}$ if $z \notin m(z') + B(r)$ and $(c_{m(z')+B(r)}, z, q) \in \cup q' \in Q'R_{q'}$ if $z \in m(z') + B(r)$;
- the above elements define a global state decoding map $\phi^+ : X^+ \to (A')^{\mathbb{Z}^2} \times \mathbb{Z}^2 \times Q'$ by $\phi^+(c, z, q) = (c'(z'), q')$ where $z'$ is such that $z \in m(z') + B(r)$ and $q'$ such that $(c_{m(z')+B(r)}, z, q) \in R_{q'}$ and $c'(z') = b'$ and for all $z'' \neq z'$, $c'(z'') = a'$ whenever $c_{m(z'')+B(r)} \in P_{a'}$;
- a time rescaling factor $T \geq 1$, such that for any initial configuration $c' \in (A')^{\mathbb{Z}^2}$ and $z' \in \mathbb{Z}^2$ such that $c'(z') = b'$, and for any global state $(c, z, q) \in X^+$ such that $\phi^+(c, z, q) = (c'_0, z'_0, q_0)$, it holds that $F^T(c, z, q) \in X^+$ and $\phi^+(F^T(c, z, q)) = F'^+(c', z', q')$.

Note that the relation between $F^T$ and $F'$ through $\phi^+$ can be iterated along the considered orbits and that blank letter $b'$ is represented by a $b$ pattern only. This makes
Definition 4.1 strong enough to preserve computability and density of limit configurations as shown in the following lemmas.

First, we say a configuration $c \in A^{Z^2}$ is computable if there is a Turing machine which on input $z \in Z^2$ computes $c(z)$. For any finite initial configuration and a continuous self-avoiding sequential system, the associated limit configuration $c_\infty$ and freezing time map $\tau$ are Turing equivalent: $c_\infty$ can be computed from oracle $\tau (\tau(z))$ time steps of the system are enough to determine $c_\infty(z)$ and reciprocally $\tau$ can be computed with oracle $c_\infty$ (if $c_\infty(z)$ is blank then $\tau(z) = 0$ otherwise it suffices to run the system until time $t$ when cell $z$ change its state and $\tau(z) = t$). We are only interested in infinite limit configurations reached from finite initial configurations (this corresponds to a non-blocking run of the system where the head escapes towards infinity).

**Lemma 4.2.** Taking notations of definition 4.1, suppose system $S$ simulates system $S'$, and $c_\infty'$ is the limit configuration reached by $S'$ from global state $(c', z', q')$ and $c_\infty$ is the limit configuration reached by $S$ from global state $(c, z, q) \in X^+$ such that $\phi^+(c, z, q) = (c', z', q')$. Suppose also that $c'$ is finite and $c_\infty'$ infinite. Then it holds $\phi(c_\infty) = c_\infty'$. In particular, $c_\infty'$ is computable from $c_\infty$.

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

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

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). 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. 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 distorted balls from the start.

Let $v_1, v_2 \in Z^2$ be two non-colinear vectors. Denote by $H(v_1, v_2)$ the pseudo-hexagonal closed curve 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 points in the interior of $H(nv_1, nv_2)$. We then define the associated (upper) density of non-blank cells in configuration $c$ by:

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

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

**Lemma 4.3.** Suppose system $S$ simulates system $S'$ with scale factor $r$ in such a way that, for some $k \in \mathbb{N}$, the number of occurrences of $b$ in each pattern of $P_{a'}$ for $a' \neq b$ 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 $c'_\infty$ is the infinite limit configuration reached by $S'$ from global state $(c', z', q')$ with $c'$ finite and $c'_\infty$ is the limit configuration reached by $S$ from global state $(c, z, q) \in X^+$ with $\phi^+(c, z, q) = (c', z', q')$, then $\overline{d}_{v_1, v_2}(c'_\infty) = \frac{\overline{d}_{v'_1, v'_2}(c'_\infty)}{r^{k-1}} \cdot \overline{d}_{v_1, v_2}(c_\infty)$.

For any oritatami with delay $\delta$ there is a turéd of radius $\delta$ simulating it with scale factor 1: indeed, an oritatami transition is completely determined by the position in the sequence of beads, coded as a state of the turéd, and the local configuration in a ball of radius $\delta$. 
5 Delay-3 oritatami systems simulate radius-1 Turedos

Consider a radius-1 turedo. First, we get rid of the internal state and orientation of the turedo by encoding them in the letters written by the turedo. We then encode each letter of the resulting turedo alphabet $A$ as a string $q$ bits where $q = \lceil \log_2 |A| \rceil$. Let $Q = 2^q$. Note that the size of turedo’s transition table is $\Theta(qQ^6)$. As in [18], 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 towards the correct computed 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 macro-cells must then be isotropic. Furthermore, the reading process must be non-blocking. We cannot thus use the hardcoded exit mechanism in [18], nor the flip-flap mechanism which would block any further return to a previously visited border. Moreover, as we need to return to a random side after reading, our oritatami system must be able to absorb up to 4 times the side length before exiting to the new macro-cell and starting the next period of the transcript. As a consequence, we cannot offered to store information on the boundary of the macro-cell as in [18], but need to store information inside the macro-cell.

Folding meters and pockets. For these purposes, we develop two new generic oritatami tools, the folding meters and pockets, which allow to expose specific prescribed structures or to hide them in a very compact manner, depending of the context. A folding meter can either: be flat, when following a border to which it is strongly attracted; build a specific structure when it follows a border to which it is mildly attracted; or folded upon itself in a very compact manner, when it enters a pocket (hence the name). Fig. 6 on page 18 illustrates this behavior. Furthermore, several folding meters can be layered on top of each other in opposite direction as long as their periods match.

Principle of the simulation. Figures 2 and 3 show an actual simulation and Figure 4 details the reading and writing mechanism. Our simulation proceeds by building hexagonal macro-cells of asymptotically optimal size $O(\sqrt{qQ^3})$. The delay is 3, allowing to sense beads at distance 4, and thus allowing two more layers on top of the reading layer, namely the write and exit layers. The transcript is built as follows:

- It starts with an hardcoded scaffold that builds the skeleton of the macro-cell, placing pockets at the appropriate locations for the next layers to come. In particular, it ends with an enhanced version of the speed bump in [18] allowing to resynchronize the molecule along a non-flat path.

- Then, it continues with the reading layer which consists of equally spread reading heads. The scaffold has build $q$ equally spread reading pockets on each border of the macro-cell, of increasing size $2^0 Q^i, \ldots, 2^q - 1 Q^i$ periods of the folding meter on the $i$th border. The reading layer either jumps over the reading pocket if the beads present on the opposite side encode a 0, or folds compactly inside the pocket if they encode a 1 (or the blank symbol) (see Fig. 8 on page 22). It follows that when it reaches the starting point of the scaffold of the macro-cell (the counterclockwise semi part of the north side), the reading layer is shifted by $\Delta = \sum_{i=1}^6 x_i Q^i \in \{0, \ldots, Q^6 - 1\}$ periods (of the folding meter) where $x_i$ is the integer encoded by the $q$ bits read from the opposite side of the $i$th border.

- then, the transcript enters the uturn pocket which allows to switch seamlessly from the reading layer to the writing layer, regardless of the shift $\Delta$. The scaffold layer has placed
$q-1$ pockets of capacity $Q^6-1$ periods between each writing location along every border. These pockets will be used to hide the unused entries of the transition table for each bit written on the sides. The bit written is encoded by a specific shape that the write layer adopts when it passes above the write locations (see Fig. 11 on page 25).

- When reaching the clockwise end of the macro-cell, the write layer enters the speed bump module (Sec. A.16 on page 48), which absorbs the shift $\Delta$.
- The scaffold has placed one exit pocket in each of the five possible exit sides of the cell.

The transcript concludes with the exit layer that follows the border until it reaches a pocket in which the write layer has placed a specific bead (thanks to the shift $\Delta$) that will trigger its folding as a compact form. This compact form concludes with a glider that exits to the next cell. The size of each exit pocket is designed so that it exits at the exact same position along the exit side regardless of its clockwise position (see Sec. A.12 on page 43 for details).

Many details are omitted due to the space constraints and may be found in the section A on page 15.

▶ Theorem 5.1. There is a universal bead type set $B$ such that for any turedo of radius 1 with alphabet of size $Q$, there is a periodic transcript of length $\Lambda = \Theta(Q^6 \log Q)$ with bead types in $B$ and a triangular seed of size 3, that simulates intrinsically the turedo at space-scale $\Theta(Q^3 \sqrt{\log Q})$ and time-scale $\Lambda$.

Arbitrary dense simulation. Note that the geometry of the macro-cell in the simulation can be extended by an arbitrary length to accommodate in its center an arbitrary large hardcoded hexagon filled with filling beads by the scaffold at the beginning of its folding. This allows us to simulate any turedo with macrocells of constant fixed density arbitrary close to 1.

▶ Theorem 5.2. For any turedo of radius 1, and for any $\epsilon > 0$ there exist $k,r \in \mathbb{N}$ with $k \geq (1-\epsilon)b(r)$ and an oritatami of delay 3 that simulates it with the following parameters: the scale factor is $r$ and the number of occurrences of $b$ (blank letter of the oritatami) in each pattern of $P_{a'}$ for $a' \neq b'$ (blank letter of the turedo) is always $k$.

6 Uncomputable Limit Configurations and Freezing Time

The goal of this section is to prove that turedos of radius 1 and therefore oritatmi are powerful enough to produce uncomputable limit configuration and freezing time maps starting from finite initial states.

The basic idea is to run a Turing machine simulation that tests all Turing machines for halt and, when it finds that some machine $i$ has halted, goes and write a flag at some position $p(i)$ which is computable in $i$ (initially all position $p(i)$ are empty). Positions of type $p(i)$ are progressively filled in some unknown order, but, at the limit, it holds that $p(i)$ contains a flag if and only if machine $i$ halts. Therefore the limit configuration is uncomputable because it can solve the halting problem when used as an oracle.

The turedo we are going to describe uses zigzag transducers to make the global Turing simulation mentioned above and zigzag snakes to go and flag positions corresponding to halting machines. The technical difference with the above description is that there is no fixed flagging position $p(i)$ for machine $i$, but rather a flagging zone which is computable from $i$ and allows to determine whether machine $i$ halts or not, but whose precise contents depends on the halting time of all machines $j \leq i$ (see Figure 5 for an overview of the construction). Said differently, we are using a Turing reduction and not a many-one reduction.
Oritatami simulates self-avoiding 2D Turing machines.

Figure 2 A macro-cell for $q = 2$ bits ($Q = 4$ states) together with the order in which layers and modules are used along its boundary. Actual real size – Zoom in for details.

Figure 3 A path of 7 macrocells for $q = 2$ bits: starting from the top macrocell, it goes $\uparrow$, $\uparrow\leftarrow$, $\downarrow$, $\downarrow\leftarrow$, $\downarrow\rightarrow$, $\downarrow$ and $\downarrow\rightarrow$. Actual real size – Zoom in for details.
Figure 4 Reading block (at the bottom) and Writing block (on top) for $q = 3$ ($Q = 8$ states): The bits 011 are read on the $i$-th border, filling only the central read pocket which results in a shift of the read layer by precisely $(0 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^0) \cdot Q$. The $q-1$ intermediate write pockets (on top) hide the $Q^0 - 1$ unused transition table entries between two consecutive bits. Note the positions of the two ears made by the write layer which either display (bit set to 0) or hide (bit set to 1) the special beads in the exit layer, or raise up to the lower macro-cell (bit set to 0) when no exit layer will show up because this border is after the exit border of the macro-cell in counter clockwise order. Actual real size – Zoom in for details.

Figure 5 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 $T(i_1) \leq T(i_2) \leq T(i_3)$ 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.
Theorem 6.1. There exists a turedo of radius 1 which, when started from the global state with a blank tape configuration, reaches an uncomputable limit configuration and therefore as an uncomputable freezing time map \( \tau_s \).

Using Lemma 4.2 and Theorem 5.2, we get the following corollary for oritatami.

Corollary 6.2. There exists an oritatami with delay 3 and a finite seed \( \sigma \) such that both the limit configuration \( c_\infty \) reached from \( \sigma \) and the freezing time \( \tau_\sigma \) are uncomputable.

7 Density of Limit Configurations

We can construct a turedo of radius 1 that is able to produce limit configurations with any possible density when starting from well-chosen finite configurations. By possible density we mean any real number \( d \in [0, 1] \) which is \( \Pi_2 \)-computable [20], i.e. such that there exists a computable sequence of rational numbers \( (q_n) \) with \( d = \lim \sup_n q_n \). The construction is rather technical but the overal idea is simple: at step \( n \), leave a large corona empty then densely fill a large corona in such a way that the surface ratio between these coronas is \( q_n \) and that their size is large enough to dominate all the previously constructed corona in anterior steps.

Theorem 7.1. There exists a turedo of radius 1 such that for any \( \Pi_2 \)-computable number \( d \in [0, 1] \) and any pair of non-colinear vectors \( v_1, v_2 \), there is an initial global state whose tape configuration is finite such that the limit tape configuration \( c_\infty \) reached from it verifies: \( \overline{d}_{v_1, v_2}(c_\infty) = d \).

The \( \Pi_2 \)-computability limitation is unavoidable as shown in the next lemma, 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 configuration.

Lemma 7.2. Let \( F \) be any continuous sequential self-avoiding system and \( x \in S_M \) be any initial global state whose tape configuration is finite. Denote by \( c_\infty \) the limit tape configuration reached by \( F \) from \( x \). 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 \( \Pi_2 \)-computable number.

By simulation, we get the corresponding result for oritatami which immediately follows from Lemma 4.3 and Theorem 5.2.

Corollary 7.3. For any \( \epsilon > 0 \), there exists an oritatami of delay 3 such that for any \( \Pi_2 \)-computable number \( d \in [0, 1 - \epsilon] \) and any pair of non-colinear vectors \( v_1, v_2 \), there is a finite seed such that the limit configuration \( c_\infty \) reached from it verifies: \( \overline{d}_{v_1, v_2}(c_\infty) = d \).

8 A conjecture

We conclude by the following conjecture to stimulate future research: Denote by \( TUR(r) \) the set of turedos of radius \( r \), by \( ORI(d) \) the set of oritatmis of delay \( d \), and by \( X \geq Y \) the fact that any system in class \( Y \) is simulated by some system in class \( X \). We make the following conjecture (in 2D):

- \( TUR(1) \not\geq TUR(2) \not\geq TUR(3) \not\geq \cdots \);
- \( ORI(d) \not\geq TUR(2) \) whatever \( d \).

However we conjecture that in 3D all classes \( TUR(r) \) are equivalent and that \( ORI(d) \) simulate them for some \( d \leq 3 \).
References


Oritatami simulates self-avoiding 2D Turing machines


A Geometry of the oritatami modules

In this section, we first present the exact geometry of each module involved in the Turedo simulation by delay 3 oritatami systems.
A.1 Notations

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

- $x.\text{nextMultiple(of: } y) = y \lfloor x/y \rfloor$ is the least multiple of $y$ larger or equal to $x$
- $x.\text{complement(to: } y) = y \lfloor x/y \rfloor - x$ so that $x+x.\text{complement(to: } y) = x.\text{nextMultiple(of: } y)$

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.
A.2 Folding meter and Pocket

A \( n \)-folding meter is a \( 4n \)-periodic transcript with period \( \Phi \) of the form:

\[
\Phi = t_0, t_1, t_2, p_3, \ldots, p_{n-1}, b_n, b_{n+1}, q_{n+3}, \ldots, q_{2n-1},
\]
\[
t_{2n+0}, t_{2n+1}, t_{2n+2}, p_{2n+3}, \ldots, p_{3n-1}, b_{3n}, b_{3n+1}, b_{3n+2}, q_{3n+3}, \ldots, q_{4n-1}
\]

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

\[
\Phi_i \bowtie \Phi_{i-1}, \quad \Phi_{i+1} \bowtie \Phi_{i-2}, \quad \Phi_{n+i} \bowtie \Phi_{n-i}, \quad \text{and} \quad \Phi_{n+i} \bowtie \Phi_{n-i-1}
\]

for all \( i \)

ensures that it will either (see Fig. 6):

- 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. 6, where the bead \( P_1 \) 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.
Figure 6 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).
A.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. 7 illustrates the only possible offsets between the layers in all of our designs:

![Figure 7](image)

**Figure 7** 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:
Oritatami simulates self-avoiding 2D Turing machines

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Read/Write</th>
<th>Write/Exit</th>
<th>Concerned layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay</td>
<td>$-1 \ldots +1$</td>
<td>$-1 \ldots +1$</td>
<td></td>
</tr>
<tr>
<td>Write pocket</td>
<td>$-1 \ldots +3$</td>
<td>0</td>
<td>R1/W2, R1/W12, R2/W12, R2/W1</td>
</tr>
<tr>
<td>Write pocket</td>
<td></td>
<td></td>
<td>W2/X1, W12/X1, W12/X2</td>
</tr>
<tr>
<td>Write pocket</td>
<td></td>
<td></td>
<td>W1/X2</td>
</tr>
<tr>
<td>Write module</td>
<td>0; $n - 1 \ldots n + 1$ (specific)</td>
<td>$-1 \ldots +1$</td>
<td>R2/W1</td>
</tr>
<tr>
<td>Write module</td>
<td></td>
<td></td>
<td>W1/X2</td>
</tr>
<tr>
<td>Read Pocket + intermediate delay</td>
<td>0; $n - 1 \ldots n + 1$</td>
<td>$-1 \ldots +1$</td>
<td>R1/W2, R2/W12, R1/W12</td>
</tr>
<tr>
<td>Read Pocket + intermediate delay</td>
<td></td>
<td></td>
<td>W2/X1, W12/X1, W12/X2</td>
</tr>
<tr>
<td>Exit interchange</td>
<td>$-2 \ldots +1$</td>
<td>$-2 \ldots +1$</td>
<td>R2/W12, R1/W12</td>
</tr>
<tr>
<td>Exit pocket</td>
<td>$-2 \ldots +2$</td>
<td>$-2 \ldots +2$</td>
<td>R1/W2</td>
</tr>
<tr>
<td>Exit pocket</td>
<td></td>
<td></td>
<td>W2/X1</td>
</tr>
<tr>
<td>Uturn pocket</td>
<td>0; $n - 1 \ldots n + 1$ (specific)</td>
<td>$-1 \ldots +1$</td>
<td>R2/W12, R1/W12</td>
</tr>
<tr>
<td>Uturn pocket</td>
<td></td>
<td></td>
<td>W12/X12</td>
</tr>
<tr>
<td>Uturn pocket</td>
<td></td>
<td></td>
<td>W12/X2</td>
</tr>
</tbody>
</table>

As a consequence:

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Offsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1/W2</td>
<td>-2\ldots +3</td>
</tr>
<tr>
<td>R1/W12</td>
<td>-2\ldots +3</td>
</tr>
<tr>
<td>R2/W12</td>
<td>-2\ldots +3</td>
</tr>
<tr>
<td>R2/W1</td>
<td>-1\ldots +3</td>
</tr>
<tr>
<td>W2/X1</td>
<td>-2\ldots +2</td>
</tr>
<tr>
<td>W12/X1</td>
<td>-1\ldots +1</td>
</tr>
<tr>
<td>W12/X2</td>
<td>-1\ldots +1</td>
</tr>
<tr>
<td>W1/X2</td>
<td>-2\ldots +2</td>
</tr>
<tr>
<td>W12/X12</td>
<td>-2\ldots +3</td>
</tr>
</tbody>
</table>

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 $-2\ldots +3$ indices, there are no long-range interaction between different layers.
### A.4 Read pocket

The primary purpose of the read pocket is to read a bit (0/1) and create an offset of unit $2^n$ and linearly proportional to its size in collaboration with $n$-folding meters. At an entrance, the read layer transcript is attracted towards the read pocket, but more strongly upward by a facing macrocell or two beads of its write module (shall be explained in Sect. A.5) that encode 0. In this case (see Fig. 10), the transcript gets out of the read pocket immediately after yielding a glider of size $2^n$ plus some constant, dotted in Fig. 8. If the two beads encode rather 1 (Fig. 10), then it is not until the lower rectangular structure of the read pocket is filled fully with switchbacks that the transcript can escape from the pocket. The difference in length between these paths results in an offset. The offset depends on parameters $k$, $w$, $\rho$, and $x$, and the minimum offset is $2^n$. This novel method of bit reading does not leave any bead fixed permanently around a bit read unlike the method used in [19] and enables us to make use of the neighbors for future computation; the proposed system lets the write and exit layers go through there to exit at an arbitrary side.

The read pocket is also used to prevent read layers of adjacent macrocells from interfering with each other. The read layers are “colored” in two ways and the rule set is designed so as for a bead of one color and a bead of the other color 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 and exit trap, which shall be explained shortly, are used for the respective analogous purposes for write layers and exit layers.

Relations among the parameters are the following:

- $0 \leq \rho < 2k + 1$
- We want the length of the Read layer from the last $B$ before entering a read pocket to the first $T$ after getting out of it to be equal to $n$ modulo $2n$. The length, when reading a 1, is $n + 2nw(2k + 1) + p + 5 + x + 2 + 3 + p + m + 3 + 2w + x + 2 + 5 + 1 + 17 = n + 2n(w(2k + 1) + \rho) + 2(w + x + 19) = n \mod 2n$; thus, \(x = (w + 19).\text{complement}(|: n)\)
- A read pocket is accompanied with two “ears” to create the offset $n$ between Read and Write layers before reading and back after. In order to prevent the left ear from colliding with the pocket, $y$ should be chosen so as to satisfy $2ny + 1 - x - 2w \geq 3$, that is, \(y \geq (w + x/2 + 1)/n\)

A read pocket that swallows exactly $\Delta 2^n$-periods of the read folding meter layer when reading a 1 with respect to reading a 0 is implemented by setting the parameters $k$, $w$, and $\rho$ as follows:

- $k = \left\lceil \sqrt{1 + (8\Delta + 10)/n - 151/n^2 - n/4 - 1/4} \right\rceil$
- $w = (\Delta - 19/n + 1)/(2k + 1 + 1/n)$
- $\rho = \Delta - (w(2k + 1) + (w + 19).\text{nextMultiple}(|: n)/n - 1)$

The case when $\Delta \leq 2$ is exceptional when these parameters should be set simply as $k = \rho = 0$ and $w = \Delta$.

**Lemma 1.1.** On input $\Delta$, the algorithm outputs parameters $(k, w, \rho, x)$ such that the box swallows exactly $\Delta 2^n$-periods of the read folding meter layer when reading a 1 with respect to reading a 0.
Figure 8 Read pocket.

Figure 9 Read pocket reading bit 0.
Figure 10 Read pocket reading bit 1.
A.5 Write module

Write module locates two beads of special type (denoted by blue bullets in figures) at a designated readable site (Figs. 11a and 11b) or deliberately out of the site; so they cannot attract the reading head no matter what types they are (Fig. 11c), which has the same effect as writing 1. Depending on which side to exit at a given input read, the exit layer may cover each write module or not. The transition table is prepared for each write module. Hence, the write layer “knows” whether it shall be covered, when the bit 0/1 is encoded by bead types at the tip of totoro ears, or not covered. In the latter case, mini totoro duos are placed to the left (Fig. 11b) or right (Fig. 11c) of the hill. The exit layer folds from right to left, and it hits the brake before the hill if mini totoros are there, sliding the special beads out of the readable site, equivalent to writing 1.
(a) **Write module – Top variant:** the write layer writes a 0 by forming two ears on the top of the module, with two active beads aligned with the reading head of the neighboring macrocell.

(b) **Write module – Left variant:** 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 aligned with the reading head of the neighboring macrocell.

(c) **Write module – Right 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 misaligned with the reading head of the neighboring macrocell.

**Figure 11** The three variants of the Write module.
Figure 12 Write module writing 0 – Top variant.

Figure 13 Write module writing 0 – Left variant.

Figure 14 Write module writing 1 – Right variant.
A.6 Write pocket

The write pocket is basically the read pocket without reading, that is, read and write layers always enter. Its primary purpose is to store a transition table, or more precisely, a part of one transition table and a part of the next transition table, so that only some short fragment of the tables that correspond to the input read (and some fragments corresponding to close inputs, though they shall never be read). The exit layer may or may not reach a write pocket, but if it does, then it jumps over the pocket by a zigzag glider that is preprogrammed.

As announced earlier in Sect. A.4, the write pocket is also used to let rather the write layer flip its “color” to prevent interference between macrocells.

Relation between the parameters are as follows:

- $0 < \rho < 2k + 1$
- We want the length of the Read layer from $B$ to $B$ inside the lower bubble to be a multiple of $2n$, thus: $1 + 4 + 1 + 4 + 1 + 4 + 2w + y + 3 + (2k + 1 - \rho)n - 4 + 3 + y + 4 + 3 = n + \rho n$ mod $2n$, that is $2(w + y + 12) = 0 \mod 2n$ that is $y = (w + 12).\text{complement}(to: n)$
- We want the length of the Write layer from $T/B$ to $T$ inside the upper bubble to be $\rho n$ mod $2n$, thus: $4 + 1 + 2 + x + 2 + \rho n + 2 + x + 2w + 7 + 3 + 4 + 7 = \rho n$ mod $2n$, that is $2(x + w + 17) = 0 \mod 2n$, i.e. $x = (w + 17).\text{complement}(to: n)$
- We want the top part of the pocket to end to the right of the bubbles, thus: $2n\ell = (2w + \max(x + 4, y - 3)).\text{nextMultiple}(of: 2n)$

Total lengths of each layer:

- Read, from rightmost $T$ to leftmost $B$: $2n(\ell + 2k + 1) + 2 \left(\begin{array}{c} (w + x + 17) + w + y + 12 \end{array}\right) + n$
- Write, from leftmost $B$ to rightmost $T$: $n + 2nw(2k + 1) + \rho n + 34 + 2x + 2w + \rho n + 2n\ell = n + 2n(2k + 1)w + \rho + \ell + 2(w + 17).\text{nextMultiple}(of: n)$
- Exit, from rightmost $T$ to leftmost $B$: $2n\ell + n$

The capacity of the Write pocket is the length of write layer from the rightmost $T$ to the leftmost $T$:

$$\text{capacity} = 2n((2k + 1)w + \rho) + 2(w + 17).\text{nextMultiple}(of: n) + (2w + \max(x - 4, y - 3)).\text{nextMultiple}(of: 2n)$$

Computing the parameters to get the desired capacity:

$$\text{capacity} \geq 2n((2k + 1)w + 1) + 2(w + 17) + 2w + 1 = \text{lowerbound}$$
and $\rho < \text{lowerbound} + 4n - 2$

$$\text{lowerbound} = w(2n(2k + 1) + 4) + 2n + 35$$

We want capacity $\geq 2nL$ for some positive integer $L$ so as to minimize the Read layer length, that is asymptotically as $k$ and $w$ go to $\infty$: $4nk + 6w$. The ideal ratio is thus: $w \sim 2nk/3$. Plugging in this value into the lowerbound we get that we are looking for a value of $k$ verifying:

$$2nk/3 \cdot (2n(2k + 1) + 4) + 2n + 35 \geq 2nL$$
Solving this equation gives:

\[ k \geq k_0 = \frac{\sqrt{4(3L-2)n + n^2 - 206 - n - 2}}{4n} \]

Now we set \( w = \lceil ((2nL - 6n + 33)/(2n(2k + 1) + 4)) \rceil \). We finally set \( \rho = \max(0, 2nL - [2n((2k + 1)w + 1) + 2(w + 17)\text{.nextMultiple(of: n)} + (2w + \max(x + 4, y - 3))\text{.nextMultiple(of: 2n)}]) \), so that we ensure that \( \text{capacity} \geq 2nL \). Note that this process guarantees that \( \text{capacity} \leq 2nL + O(\sqrt{2nL}) \) (in fact we can even guarantee that \( \text{capacity} \leq 2n(L + 2) \) by adjusting the parameters carefully).

\[ \begin{array}{l}
(0,0) \quad (n-6,0) \\
(2,-2) \quad (n-7,-1) \\
(n-8,-3) \\
\end{array} \]

**Figure 15** Write pocket.

**Lemma 1.2.** The algorithm outputs parameters \((k, w, \rho, x, y, \ell)\) such that the box swallows at least \(Q^6 - 1\) 2n-periods of the write folding meter layer.
Figure 16 Write pocket.
A.7 U-turn module

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.

These properties are achieved by setting the parameters as follows:

- $0 \leq \rho < 2k + 1$

The capacity of the U-turn module is the sum of the length of Read layer and that of write layer from the rightmost $T$ on the Read layer to the rightmost $T$ on the Write layer, which is $n + 2nw(2k + 1) + \rho n + 5 + x + 3 + \rho n + 3 + x + 2w + 5 + 2 + n - 10 = 2n(w(2k + 1) + \rho + 1) + 2(w + 4 + x)$. We want this capacity to be a multiple of $2n$, thus:

$$2(w + 4 + x) \equiv 0 \mod 2n$$

that is $x = (w + 4).\text{complement(to: n)}$. 

\[\text{Figure 17 U-turn pocket.}\]
Figure 18 U-turn pocket.
The parameters $k$, $w$, and $\rho$ are computed as:

\[
\begin{align*}
  k &= \max\{1, \sqrt{Q^6/2n}\} \\
  w &= \max\{1, Q^6/(2k + 1)\} + 1 \\
  \rho &= \max\{0, Q^6 - (2k + 1)(w - 1)\}
\end{align*}
\]

\textbf{Lemma 1.3.} The algorithm outputs parameters $(k, w, \rho, x)$ such that the box swallows at least $Q^6 - 1 + (2k + 1)2n$-periods of the folding meter layer.
A.8 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.
Figure 20 Corner module.
A.9 Exit interchange trap

This trap lets the system “color” its exit layers into non-interacting regions, as done by read and write pockets for the corresponding layers.

Parameters:

- \( n = 26 \)
- \( w = 3 \)
- \( p = 3 \)
- \( \mu = (w + 2p + 5) \ldots \) complement \( n = 14 \) complement \( 26 = 12 \)

Total length from \( B \) to \( B \): \( 2n(\rho + 1) \).

Figure 21 Exit interchange trap.
Oritatami simulates self-avoiding 2D Turing machines

Figure 22 Exit interchange trap.
A.10 Exit pocket

All the sides but the north one, at which the transcript enters from the previous macrocell, if any, 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. 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. A.12). The exit layer decides whether to exit now or later at the point designated Exit trigger switch in Fig. 23. 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 box is not chosen, or not, when the exit layer folds back upward and exits here.

Total length from $B$ to $T$:

- Read, Write, Exit (inside, in brown): $n + 2n(2k+1)+2x+2W+20$, must be $n \mod 2n$,
  thus: $2x + 2W + 20 = 0 \mod 2n$, i.e. $x = (W + 10)$ complement (to: $n$).
- Exit (going out, orange): $n + 2n((2k+1)w + 2r + 1)$

Parameters:

- $n = 26$
- $W = n - 10 + 2\eta = -10 \mod n$ for some integer $\eta \geq 0$ so that $x = 0$
- $0 \leq w \leq W$
- $r = k$ so that the last switchback has the same height

Total length from $B$ to $T$:

- Read, Write, Exit (inside, in brown): $n + 2n(2k+2+\eta) = n \mod 2n$
- Exit (going out, orange): $n + 2n((2k+1)(w+1) = n \mod 2n$
Figure 23 Exit pocket.
Figure 24 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 $Wp_{76}$ here.
Figure 25 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 $Wp_7b$ here.
A.11 Step and shift modules

Step up and step down modules. The designs in Fig.26a and 26a fold as indicated for any \( h \) such that \( 0 \leq (h \mod n) \leq n - 9 \). The total length of each layer inside this modules is \( n + h - (h \mod n) \).

For steps of height \( h \) with \( n - 9 \leq (h \mod n) < n \), 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 \( 2n + h - (h \mod n) \).

![Diagram](image)

(a) Step up of height \( h \).

(b) Step down of height \( h \).

Figure 26 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. 27a and 27b).

Lemma 1.4. Given a positive integer \( L \) and \( Ln + 9 \leq h \leq 2L \) written as \( h = qn + r \) where \( 0 \leq r < n \), 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 \( r \leq n - 9 \);
- 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 $h \geq Ln + 9$, then $q \leq L - 2$, thus the line initial segment has positive length. Furthermore, the distance between the extremities is $2(L - q - 2)n + 2n + 2(h + n - r) - h = 2Ln - h$. ◀

Figure 27 The two variants of the shift module.
A.12 Exit module

Consider an exit pocket with parameters $W$ and $w$ where $w = w_0 + i\omega$ for some $i \in \{0, \ldots, 4\}$ with $W \geq w_0 + 4\omega$. Consider $L = (8\omega + 9).nextMultiple(of: n)$ such that $8\omega \leq Ln - 9$. 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. 28. 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 $i$.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{exit_module.png}
\caption{Exit module for the five values of $w = w_0 + i\omega$ for $i \in \{0, \ldots, 4\}$ (here, $w_0 = 1$ and $\omega = 8$).}
\end{figure}
A.13 Corner interchange block

Figure 29 Corner interchange block for $q = 2$ ($Q = 4$ states).

Actual real size – Zoom in for details
A.14 Middle interchange block

Figure 30 Middle interchange block for \( q = 2 \) (\( Q = 4 \) states).

Actual real size – Zoom in for details
A.15 Determining the macro-cell size

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. Fortunately, setting the height $k_{\text{exit}}$ of the exit appropriately guarantees the existence of a solution as demonstrated bellow.

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 $q = 2$ (bits) and hence, the macro-cell is provided with one writing block with 2 write modules and one Reading block with 2 read pockets per side. In order for the resulting macro-cell to behave as expected, the parameters $k_{\text{exit}}$, $\ell_{\text{write}}$ must be set as follows.

- $k_{\text{exit}}$ is set to $Q + 1$ if $Q \mod 5 = 2$ or $Q$ otherwise, so that $2k_{\text{exit}} + 1 \neq 0 \mod 5$ (this will be crucial for the existence of a solution later).
- $k_{\text{write}}$ and $\ell_{\text{write}}$ are set respectively to $k$ and $\ell$ of the write pocket with the largest capacity needed, that is, $2n(Q^6 - 1)$ (see Section A.6).

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 $\ell_{\text{write}} - 1$ 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 the maximum of $k_{\text{exit}} + 1$ and the value $y + k + 1$ computed for all the read pockets involved (see Section A.4). 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 $y$ parameter of the read pocket of Corner interchange block is set to the maximum between the $y$ parameter just computed for Middle interchange and $k_{\text{write}} + \ell_{\text{write}}/2$. In Middle interchange block, another write pocket is concatenated directly to the read pocket, while in Corner interchange block, between them are padded a line of length $2n(y + 2) + n + 36$, where $y$ is the one just computed particularly for this block (not the one for Middle). Finally, the Exit block consists of a line of $2n(k_{\text{exit}} + 1)$ and the Exit module, which consists of an exit pocket parameterized by $W = w_{\text{exit}}$ and $k = r = k_{\text{exit}}$ and sandwiching two shift modules of maximum shift $2w_{\text{exit}}$.

These five types of blocks are concatenated into a side (though the resulting “side” turns 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: $2n(q - 1) + 2nq$ times the distance between the adjacent write modules;
- Corner interchange block: $n + 2n(2\ell_{\text{write}} + 2y + 2k_{\text{exit}} + 8)$, where $y$ is the one computed for this block above;
- Write block: $n + 2n(q - 1)(\ell_{\text{write}} + 1)$;
Middle interchange block: $n + 2n(2\ell_{\text{write}} + y' + 2k_{\text{exit}} + 4)$, where $y'$ is the $y$ computed for this block before;

Exit block: $n + 2n(3k_{\text{exit}} + 7) + 2n \times 5\eta$

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 $2n(A + 5\eta)$.

Now it suffices to set the parameter $\eta$ of the exit module properly, and the description of the macro-cell at least size-wise shall be complete. Firstly, the exit pocket must be large enough in order to accommodate four sides of the exit layer being folded into switchback; formally speaking, $4(A + 5\eta) \leq (2k_{\text{exit}} + 1)w_{\text{exit}} = (2k_{\text{exit}} + 1)(16 + \eta)$. Moreover, $A + 5\eta = 0 \mod 2k + 1$ should hold so that one side folds precisely into a rectangle of height $(2k_{\text{exit}} + 1)n$.

Recall that $k_{\text{exit}}$ was chosen so as for $2k_{\text{exit}}$ and 5 to be coprime. Thus, starting from $\eta$ being set to $\lceil (4A - 16(2k_{\text{exit}} + 1))/n(2k_{\text{exit}} + 1) - 20 \rceil$, incrementing $\eta$ by 1 until the congruence holds, results in a proper value of $\eta$ certainly after at most $2k_{\text{exit}}$ steps (or simply by applying Bezout theorem).
A.16 Ongoer speed bump

This module is an adaptation of the speed bump from [18] to delay 3 and arbitrary path.

\[ w \equiv 8 \mod 13 \]
\[ w - 6 \equiv 26 \mod 52 = n \mod 2 \]
\[ n \equiv 4w + 22 \]

Figure 31 Speed bump module.

Rephasing module.

Bumping transcript.

Flat bumper.

Outer-turn bumper.

Inner-turn bumper.

\[ \Delta_{in} \rightarrow \Delta_{out} \text{ tables.} \]

Lemma 1.5. The \( \Delta_{in} \rightarrow \Delta_{out} \) tables combine additively.
Ongoer speed bump design algorithm.
Figure 32 Speed bump module with $w = 20, \Delta_{\text{max}} = 500$ and $\Delta = 500$. 
Figure 33 Speed bump module with $w = 20$, $\Delta_{\text{max}} = 500$ and $\Delta = 214$. 
Figure 34 Speed bump module with $w = 20$, $\Delta_{\text{max}} = 500$ and $\Delta = 0$. 
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 $d_T$ (for time) and $d_S$ (for space) in $N_H$ with $d_T + d_S \in N_H$, 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 $z_0 + N_T d_T$ and the zigzags will only occupy a cone of space between $z_1 + N_T d_T$ and $z_1 + N_T (d_T + d_S)$ for some $z_1 = z_0 + x d_T$. Concretely, it is straightforward to build a turedo $M$ with neighborhood $N_H$ such that (see Figure 1):

- the $t$-th zig visits cells $z_0 + (2t-1)d_T^2$ to $z_0 + (2t-1)d_T^2 + (l_t-1)d_T^1$ and the $t$-th zag visits cells $z_0 + 2td_T^1 + (l_t-1)d_S^1$ to $z_0 + 2td_T^1$ where $l_t$ denotes the width of the $t$-th zigzag;
- in some component of states of $M$, 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 $|l_{t+1} - l_t| \leq 1$);
- in some other component of states of $M$ position $z_0 + (2t+1)d_T^1$ (first position of $(t+1)$-th zig) encodes the state of some finite-state automaton run on $v_t$ and position $z_0 + (2t+1)d_T^1$ (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 $d_T$ while their zigzag oscillations of constant width are done in direction $d_S$, where $d_T$ and $d_S$ verify $d_T + d_S \in N_H$. They can additionally do two things:

- they can globally shift along direction $d_S$ to climb obstacles or in direction $-d_S$ when there is no occupied cell to block them (as if there was a gravity field in direction $-d_S$). The possible $d_S$ 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 $d_S$: $b$ stands for beginning, and $-1$ and $0$ are used to mark the one but last and last position in direction $d_S$. The $\delta$ value is used during the zig phase and updated during the zag phase. The zigzag cycle is
Figure 35 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.

as follows for a snake at position $z \in \mathbb{Z}^2$ with shift value $\delta$ at the beginning of the cycle (see Figure 35):

1. the zig phase starts by putting the mark ’$b$’ in position $z$ and moving in direction $\vec{d}_S$ until one of the following happens: either $\delta \leq 0$ and it reaches a position $z'$ such that $z' - \vec{d}_T$ has a marking information equal to $\delta$, in which case the next move is in direction $\vec{d}_T$ and the zag phase begins; or $\delta \geq 1$ and it reaches a position $z'$ with $z' - \vec{d}_T$ empty, in which case the next move is in direction $\vec{d}_T$ if $\delta = 1$ and $\vec{d}_T + \vec{d}_S$ if $\delta = 2$ and the zag phase begins.

2. the zag phase consists in moving in direction $-\vec{d}_S$; 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 $z - \vec{d}_T$ is marked by $b$ and $z - \vec{d}_S$ is occupied and $z + \vec{d}_T$ is empty, then move to $z + \vec{d}_T$, update $\delta$ to 0 and start a new zig;

   b. **terminating with simple obstacle in current zag**: else if $z - \vec{d}_S$ is occupied but $z + \vec{d}_T$ is free, then move to $z + \vec{d}_T$, update $\delta$ to +1 and start a new zig;

   c. **terminating with simple obstacle in next zig**: else if $z + \vec{d}_T$ is occupied but $z - \vec{d}_S$ has the mark ’$b$’, then move to $z + \vec{d}_T + \vec{d}_S$, update $\delta$ to +1 and start a new zig;

   d. **terminating with double obstacle**: else if $z + \vec{d}_T$ is occupied and $z - \vec{d}_T$ has no mark ’$b$’, then move to $z + \vec{d}_T + \vec{d}_S$, update $\delta$ to +2 and start a new zig;

   e. **terminating with free fall**: else if $z - \vec{d}_T$ is marked by $b$ and $z - \vec{d}_S$ is empty, then move to $z - \vec{d}_T$, next move to $z - \vec{d}_S + \vec{d}_T$ which is supposed to be empty; update $\delta$ 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 $(\vec{d}_T, \vec{d}_S)$ (forward snake) to $(-\vec{d}_T, \vec{d}_S)$ (backward snake) while preserving their width (see Figure 36). 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
Daria Pchelina, Nicolas Schabanel, Shinnosuke Seki, and Guillaume Theyssier

Figure 36  U-turn of a zigzag snake. In dark gray the forward snake and in light gray the backward snake.

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 \( z_0 + N \cdot \overrightarrow{d_T} \) 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 37).

- **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 38):
  - 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;
Oritatami simulates self-avoiding 2D Turing machines

Figure 37 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 38 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.

- 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: Proofs of All Technical Lemmas

C.1 Proof of Lemma 4.2

Proof. Take any $z'' \in \mathbb{Z}^2$. By the definition of limit configurations, there is a time $t_{z''}$ such that for any $t \geq t_{z''}$ we have $c_t(z'') = c_{\infty}(z'')$. The same holds for $c'_{\infty}$ and let us denote by $t'_{z''}$ the corresponding convergence times. Now take $\tau \geq \max(t'_{z''}, \max_{z'\in m(z'')}B(r)(t_{z''}))$. By the simulation, we have $\phi^+(F^T \tau(c, z, q)) = (F')^+(c', z', q')$. We also know by the assumption that the run of $(F')^+$ starting from $(c', z', q')$ is infinite (because it must produce an infinite $c'_{\infty}$ starting from a finite $c'$). The same is true for $F$ starting from $(c, z, q)$. We deduce that the heads of $F'$ will not visit cell $z''$ after time $\tau$ and similarly for all cells $m(z'') + B(r)$ of $F'$. It follows that $(c_{\infty})_{m(z'') + B(r)} \in \mathcal{P}_{c'_{\infty}}(z'')$. This being true whatever the choice of $z''$, we thus showed $\phi(c_{\infty}) = c'_{\infty}$. The fact that $c'_{\infty}$ is computable from $c_{\infty}$ follows immediately. □

C.2 Proof of Lemma 4.3

Proof. First, the hypothesis of Lemma 4.2 are fulfilled so we have $\phi(c_{\infty}) = c'_{\infty}$. In a simulation of scale factor $r$, the original lattice of $S'$ is distorted into a lattice of macro cells by the following linear transformation:

$$M_r = \begin{pmatrix} r+1 & r \\ -r & 2r+1 \end{pmatrix}$$

This linear map being non-singular (on field $\mathbb{Q}$) we deduce that there are vector $(v'_1, v'_2)$ such that $(Nv_1, Nv_2) = M_r(v'_1, v'_2)$ for some integer $N \geq 1$ $(v_1, v_2)$ 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 $S'$ in pseudo-ball $B_{v_1', v_2'}(n)$ are simulated by macro-cells in $S$ whose union

$$M(n) = \bigcup_{(a, b) \in B_{v_1', v_2'}(n)} M_r(a, b) + B(r)$$

verifies $M(n) = B_{Nv_1, Nv_2}(n) + B(r)$. Let us denote by $d_X$ the density of non-blank cells in $c_{\infty}$ restricted to $X$:

$$d_X = \frac{\#\{z'' \in X : c_{\infty}(z'') \neq b\}}{\#X}$$

and similarly $d'_{X}$ for $c'_{\infty}$. From the hypothesis and the choice of $M(n)$ above we have:

$$d'_{B_{v_1', v_2'}(n)} = \frac{b(r) - k}{b(r)} \cdot d_M(n)$$

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

$$\bar{d}_{v_1', v_2'}(c_{\infty}) = \limsup_n d'_{B_{v_1', v_2'}(n)} = \limsup_n \frac{b(r) - k}{b(r)} \cdot d_{v_1, v_2} = \frac{b(r) - k}{b(r)} \cdot \bar{d}_{v_1, v_2}(c_{\infty}).$$

□
C.3 Proof of Lemma 7.2

Proof. Suppose that the initial tape configuration has all its non-blank cells inside $B_{v_1,v_2}(n_0)$. Denote by $d_{t,n}$ the proportion of non-blank cells in $B_{v_1,v_2}(n)$ for the tape configuration reached after $t$ steps, and denote by $d_{\infty,n}$ the proportion of non-blank cells in $B_{v_1,v_2}(n)$ for configuration $c_\infty$. Clearly $d_{t,n}$ is a rational number computable from $t$ and $n$ (by the continuity of $F$), and by the self-avoiding property $(d_{t,n})_t$ is monotone increasing so we have $d_{\infty,n} = \sup_t d_{t,n}$ and therefore

$$d_{v_1,v_2}(c_\infty) = \inf_m \sup_{n \geq m} \sup_t d_{t,n} = \inf_m \sup_{n \geq m} d_{t,n}$$

Note also that if $n \geq 2(t + n_0)$ then $d_{t,n}$ is smaller than $d_{t,t+n_0}$ because after only $t$ steps of computation, all cells outside $B_{v_1,v_2}(t + n_0)$ are still blank. We deduce that $q_{m,t} = \sup_{n \geq m} d_{t,n}$ is a rational number computable from $t$ and $m$. Finally we have $d_{v_1,v_2}(c_\infty) = \inf_m \sup_t q_{m,t}$ and [20, Lemma 3.2] shows that it is a $\Pi_2$-computable number. \hfill ▷
Appendix: Construction of Theorem 6.1

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

**Algorithm 1 — Turing simulation**

\[
\begin{align*}
&= \text{MACHINES} \leftarrow \emptyset \\
&= n \leftarrow 0 \\
&= \text{loop forever:} \\
&\quad 1. \ n \leftarrow n + 1 \\
&\quad 2. \ \text{MACHINES} \leftarrow \text{MACHINES} \cup \{n\} \\
&\quad 3. \ \text{for each } i \in \text{MACHINES} \text{ do:} \\
&\quad\quad a. \ \text{simulate } n \text{ steps of machine } i \text{ starting from an empty tape} \\
&\quad\quad b. \ \text{if } i \text{ has halted during the simulation then} \\
&\quad\quad\quad i. \ \text{if necessary, wait until } m > i \\
&\quad\quad\quad ii. \ \text{MACHINES} \leftarrow \text{MACHINES} \setminus \{i\} \\
&\quad\quad\quad iii. \ \text{launch a zigzag snake subroutine of width } i
\end{align*}
\]

**Algorithm 2 — Marking**

\[
\begin{align*}
&= m \leftarrow 0 \\
&= T \leftarrow \emptyset \\
&= \text{loop forever:} \\
&\quad \text{wait } 2m + 3 \text{ zigzags} \\
&\quad m \leftarrow m + 1 \\
&\quad \text{at a zag do } T \leftarrow \text{stop} \\
&\quad \text{at the next zig do } T \leftarrow \emptyset \\
&\quad \text{wait } m - 1 \text{ zigzags and at the last zag do } T \leftarrow \text{start} \\
&\quad \text{at the next zig do } T \leftarrow \emptyset \\
&\quad \text{wait } 2m + 3 \text{ zigzags}
\end{align*}
\]

Moreover, at each zig or zag, it writes the value of variable \(T \in \{\emptyset, \text{start}, \text{stop}\}\) 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 \(d_T\) 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 \(d_T\) and put 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 37);
once created the snake of width $i$ moves in direction $-d_T$ and use space direction $-d_S$; it then adapt by small shifts its trajectory to obstacles it encounters (see Figure 35 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 5);

- when reading a start marker on an obstacle, it starts an equality test (see Figure 38) and replicate the start marker at the end of the next: the replicated marker is therefore shifted by one unit in direction $-d_T$; the same is done for the stop marker;

- if the test is successful, then a U-turn is launched (see Figure 36); 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 $d_T$ 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 36 for the details, and the red part in Figure 5 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 35);

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

**Proof of Theorem 6.1.** 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 $N d_T$ contains a succession of test segments $S_m = \{t_m d_T, (t_m + 1) d_T, \ldots, (t_m + 2m - 2) d_T\}$ with $t_m$ odd and $t_m d_T$ marked with stop (written a the end of a zig), $(t_m + 2m - 2) d_T$ marked with start and all other positions of $S_m$ marked with $\emptyset$. Segment $S_m$ is such that it will produce a successful equality test for snakes of width $m$ (see Figure 38). All positions outside test segments are also marked with $\emptyset$, 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 $t d_T$ on the baseline then $t > t_i$ (instruction 3.b.i of Algorithm 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 $T d_T$ and define the snake pile profile as positions $p(t)$ for $0 \leq t \leq T$ such that $p(t) = t d_T - s d_S$ 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 $-\vec{d}_T$ (thus allowing a future snake to follow this profile); precisely: $p(t - 1)$ is closer to the baseline than $p(t)$, or $p(t - 1) = p(t) - \vec{d}_T - \vec{d}_S$;
- at the level of each test segment $S_m$ already marked, the snake pile profile is flat ($p(t + 1) - p(t) = \vec{d}_T$); 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 $t \vec{d}_T \in S_m$ have the same marks as $t \vec{d}_T$ (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 $t \vec{d}_T \in S_m$ then $p(t) = t \vec{d}_T - s \vec{d}_S$ with $s \leq 2m^2$. 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 $\vec{d}_T$ 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

$$R_i = \{(t - k) \vec{d}_T - s \vec{d}_S : 0 \leq k \leq 2i \text{ and } 0 \leq s \leq 2i^2\}.$$

$R_i$ 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 $p - \vec{d}_S$ (indeed, U-turns are the only part of the construction where the snake pile contains blank holes, as shown in Figure 5).

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.
E Appendix: Construction of Theorem 7.1

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: $B(r) = B_{v_1, v_2}(r)$, the target density $d = \limsup \frac{p_n}{q_n}$. 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 $1 \leq p_n \leq q_n - 1$ 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 $n$th term is $(p_m, q_m)$ where $m \geq n$ is the first index such that the inequality holds). The basic ingredient of our construction is a well chosen sequence of coronas:

- define the corona 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 corona verify: $b(r) = \#B(r) \in o(r^2)$ and $c(r, R) = \#C(r, R) \in o(R^2 - r^2) + o(R^2)$.
- choose $r_{n+1} \geq q_n^2 R_n^2$ and let $R_{n+1}$ be the smallest radius greater than $r_{n+1}$ such that $c(r_{n+1}, R_{n+1}) \geq \frac{p_n}{q_n}$.

This choice ensures the two following properties:

1. a good enough approximation of density: $\frac{c(r_{n+1}, R_{n+1})}{b(R_{n+1})} \in [\frac{p_n}{q_n}, \frac{p_n}{q_n} + \epsilon(n)]$ with $\epsilon(n) \in o(1)$, because $\frac{b(r_{n+1}, R_{n+1}) - 1}{b(R_{n+1} - 1)} < \frac{p_n}{q_n}$ and the derivative of $R \mapsto \frac{c(r_{n+1}, R)}{b(R)}$ is $O(\frac{1}{R})$;

2. a thick enough corona (useful condition for the turedo construction): $R_n - r_n \in \Omega(R_n^2)$. Indeed the condition $c(r_n, R_n) \geq \frac{p_n \cdot b(R_n)}{q_n}$ implies $R_n - r_n \in \Omega(\frac{p_n}{q_n})$ (because $p_n \geq 1$) so $R_n - r_n \in \Omega(R_n^2)$. Moreover, we have $R_n \in O(q_n r_n)$ because $\frac{c(r_n, q_n r_n)}{b(q_n r_n)} \geq \frac{q_n - 1}{q_n + 1} + o(1)$

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 $R_n - r_n \in \Omega(R_n^2)$.

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 coronas to achieve the correct density.

Lemma 5.1. If a configuration $\gamma$ 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 $\gamma(z) \neq b$,
3. for any $z \in C(R_n, r_{n+1}) \setminus X$ we have $\gamma(z) = b$,

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

Proof. Denote by $d_+^r$ the non-blank density inside pseudo-ball $B(r)$ for configuration $\gamma$:

$$d_+^r(\gamma) = \frac{\# \{ z \in B(r) : \gamma(z) \neq b \}}{b(r)}.$$
First, we can reduce to the case where \( X = \emptyset \): if \( \gamma_0 \) is a configuration verifying the hypothesis of the lemma with \( X \) empty, and \( \gamma \) is any configuration equal to \( \gamma_0 \) everywhere except on a set \( X \) verifying the hypothesis, then:

\[
|d_r(\gamma) - d_r(\gamma_0)| \leq \frac{\#X \cap B(r)}{b(r)} + o(1)
\]

so the two sequences have same limsup.

Now suppose that \( X = \emptyset \); 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 \( 0 \leq \beta(n) \leq b(R_{n-1}) \). The choice or \( r_n \) and \( R_n \) above then gives \( d_{R_n} = \frac{d_n}{2n} + o(1) \).

Therefore \( \limsup_n d_{R_n} = d \). To conclude that \( d_{v_1,v_2}(c) = \limsup_n (d_n) = d \) 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:

\[
\begin{align*}
\text{when } r \in [r_n, R_n], & \quad \cdot \quad d_{r+1} - d_r \geq \frac{d_r b(r) + c(r,r+1)}{b(r+1)} - d_r = \frac{c(r,r+1)(1-d_r)}{b(r+1)} > 0 \\
\text{when } r \in [R_n, r_{n+1}], & \quad \cdot \quad d_{r+1} - d_r \leq \frac{d_r b(r)}{b(r+1)} - d_r = \frac{-d_r c(r+1)}{b(r+1)} < 0
\end{align*}
\]

\( \blacksquare \)

**Description of the turedo.** We now describe a turedo able to produce limit configurations that satisfy the hypothesis of Lemma 5.1. 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 coronas. 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 coronas, and a corona filling routine that visits approximately all cells of a given corona and gives back control to the Turing component once finished. See Figure 39 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 \( \tau \) 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 \( O(\log(r)) \). 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 coronas

\begin{align*}
\text{1. } n &\leftarrow n + 1 \\
\text{2. } \text{compute } (p_n, q_n) \\
\text{3. } \text{compute } r \leftarrow q_n^2 R^2 \\
\text{4. } \text{compute the smallest } x \geq r \text{ such that } \frac{(x, x)}{\pi r} \geq \frac{m}{r} \\
\text{5. } R &\leftarrow x \\
\text{6. } \text{compute } w \in \Omega(\log(R)) \text{ large enough to hold the binary representation of } R \text{ and } r \\
\text{7. } \text{do zigzags until the internal step counter } \tau \text{ is exactly } r \\
\text{8. } \text{call the filling routine of inner radius } r, \text{ outer radius } R, \text{ and step increment } w
\end{align*}

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 guarantees that the value of \( \tau \) 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 corona filling routine is called at the \( n \)th 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 corona filling routine aims at visiting approximately all cells of \( C(r, R) \) and does it by filling successive layers which are coronas \( C(r + kw, r + (k + 1)w) \) for \( 0 \leq k \leq O(R/\log(R)) \) 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 \( O \leq i \leq R - r \) to apply Lemma 5.1 (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 corona 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 corona \( C(r, R) \).

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 coronas. 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 \( Zv + Zv^* \) (see Figure 40). In the same way as already explained for snakes following irregular path in the construction of Theorem 6.1, we can have zigzag transducer doing arbitrary computations while following the periodic path \( m_v^\infty \) 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 extremity of zigzags. In particular it is possible to make constant width zigzags (contrary to Figure 35 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 corona 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 corona filling routine when leaving the Turing direction $v_1$ and starting to fill a side of the corona, 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 $u = a^{m_u}b^{m_u}$ and $v = c^{m_v}d^{m_v}$ 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 40):

1. go on with zigzag tape in direction $u^*$ and use the sequence of moves $a^{w_u}$; while doing this copy the relevant content of the tape in the starting situation onto the segment from $z$ to $z + wa$;
2. 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 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 $z + vw^*$;
3. from their, start a zigzag progressing with periodic sequence of moves $m_v$ and using space direction $v^*$, and progressively increase its width until it reached the correct one (one extremity of the zigzags is on the inner side of the corona 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 zigzag are done 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 reached position $z + wv_1$ and, noting $m_{v_1} = a^kb^l$, the sequence of moves $a^{w}$ as already been done at the beginning of the layer filling process, so it remains to move according to the sequence $a^{(k-1)w}b^{lw}$. 
Figure 39 Overview of a run of the turedo to produce an approximate sequence of coronas in the sense of Lemma 5.1. In red the Turing computation zone. In yellow the direction change error zone of each layer of the corona filling routine (each yellow polygon is of area $O(\log (R_{n+1})^2)$). The part in gray is filled with density 1.
Figure 40 Geometrical details of the turedo’s construction: on the left, the tiling by $T_v$ with $m_u = \overrightarrow{nw} \overrightarrow{ne}$ and $v^* = \overrightarrow{nw}$; on the right, an example of change of direction from $u$ with $m_u = \overrightarrow{nw}$ to $v$ with $m_v = \overrightarrow{sw} \overrightarrow{nw}$; 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.
The oritatami rule

The current implementation includes 1093 bead types. The rule is given only for completeness purpose and can be downloaded at: