Proving the Turing Universality of Oritatami Co-Transcriptional Folding

Cody Geary
California Institute of Technology, Pasadena, CA, USA
cody@dna.caltech.edu

Pierre-Étienne Meunier
Maynooth University, Ireland
pierre-etienne.meunier@mu.ie

Nicolas Schabanel[1]
CNRS, ÉNS de Lyon (LIP, UMR 5668), France and IXXI, U. Lyon, France
http://perso.ens-lyon.fr/nicolas.schabanel/

Shinnosuke Seki[2]
Oritatami Lab, University of Electro-Communications, Tokyo, Japan
http://www.sseki.lab.uec.ac.jp/

Abstract

We study the oritatami model for molecular co-transcriptional folding. In oritatami systems, the transcript (the "molecule") folds as it is synthesized (transcribed), according to a local energy optimisation process, which is similar to how actual biomolecules such as RNA fold into complex shapes and functions as they are transcribed. We prove that there is an oritatami system embedding universal computation in the folding process itself.

Our result relies on the development of a generic toolbox, which is easily reusable for future work to design complex functions in oritatami systems. We develop "low-level" tools that allow to easily spread apart the encoding of different "functions" in the transcript, even if they are required to be applied at the same geometrical location in the folding. We build upon these low-level tools, a programming framework with increasing levels of abstraction, from encoding of instructions into the transcript to logical analysis. This framework is similar to the hardware-to-algorithm levels of abstractions in standard algorithm theory. These various levels of abstractions allow to separate the proof of correctness of the global behavior of our system, from the proof of correctness of its implementation. Thanks to this framework, we were able to computerise the proof of correctness of its implementation and produce certificates, in the form of a relatively small number of proof trees, compact and easily readable/checkable by human, while encapsulating huge case enumerations. We believe this particular type of certificates can be generalised to other discrete dynamical systems, where proofs involve large case enumerations as well.

2012 ACM Subject Classification Theory of computation → Models of computation, Applied computing → Life and medical sciences, Hardware → Biology-related information processing

Keywords and phrases Molecular computing, Turing universality, co-transcriptional folding

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.23

Related Version A full version of the paper is available at [7], https://arxiv.org/abs/1508.00510.

[1] Supported by CNRS grants MoPrExProgMol and AMARP.
[2] Supported by JST Program No. 6F36, and JSPS Grants Nos. 16H05854, 18K19779, and YB29004.

1 Introduction

Oritatami model was introduced in [6] to try to understand the kinetics of co-transcriptional folding. This process has been shown to play an important role in the final shape of biomolecules [1], especially in the case of RNA [4]. The rationale of this choice is that the wetlab version of Oritatami already exists, and has been successfully used to engineer shapes with RNA in the wetlab [8].

In oritatami, we consider a finite set of *bead types*, and a periodic sequence of *beads*, each of a specific bead type. Beads are attracted to each other according to a fixed symmetric relation, and in any folding (a folding is also called a *configuration*), whenever two beads attracted to each other are found at adjacent positions, a *bond* is formed between them.

At each step, the latest few beads in the sequence are allowed to explore all possible positions, and we keep only those positions that minimise the energy, or otherwise put, those positions that maximise the number of bonds in the folding. "Beads" are a metaphor for domains, i.e. subsequences, in RNA and DNA.

Previous work on oritatami includes the implementation of a binary counter [6], the Heighway dragon fractal [12], folding of shapes at small scale [3], and NP-hardness of the rule minimization [15, 9] and of the equivalence of non-deterministic oritatami systems [10].

Main result. In this paper, we construct a "universal" set of 542 bead types, along with a single universal attraction rule for these bead types, with which we can simulate any tag system, and therefore any Turing machine $\mathcal{M}$, within a polynomial factor of the running time $\mathcal{M}$. The reduction proceeds as follows:

$$\text{Turing machine} \xrightarrow{[16,\ 13]} \text{Cyclic tag system} \xrightarrow{\text{Prop. 2}} \text{Skipping cyclic tag system} \xrightarrow{\text{Lem. 3}} \text{Oritatami system}$$

Our result relies on the development of a generic toolbox, which is easily reusable for future work to design complex functions in oritatami systems.

Proving our designs. The main challenge we faced in this paper was the size of our constructions: indeed, while we developed higher-level geometric constructs to program useful shapes, there is a large number of possible interactions between all different parts of the sequence. Getting solid proofs on large objects is a common problem in discrete dynamical systems, for instance on cellular automata [5, 2] or tile assembly systems [11]. In this paper, we introduce a general framework to deal with that complexity, and prove our constructions rigorously. This method proceeds by decomposing the sequence into different *modules*, and the space into different areas: *blocks*, where exactly one step of the simulation is performed, which are composed of *bricks*, where exactly one module grows. We can then reason on the modules separately, and only deal with interactions at the border between all possible modules that can have a common border.

2 Definitions and Main results

2.1 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 in the triangular lattice $\mathbb{T}$,[3] where for all integer i, vertex c_i of c

[3] The triangular lattice is defined as $\mathbb{T} = (\mathbb{Z}^2, \sim)$, where $(x, y) \sim (u, v)$ if and only if $(u, v) \in \cup_{\epsilon=\pm 1}\{(x+\epsilon, y), (x, y+\epsilon), (x+\epsilon, y+\epsilon)\}$. Every position (x, y) in $\mathbb{T}$ is mapped in the euclidean plane to $x \cdot \vec{E} + y \cdot \vec{SW}$ using the vector basis $\vec{E} = (1, 0)$ and $\vec{SW} = \text{RotateClockwise}\left(\vec{E}, 120°\right) = (-\frac{1}{2}, -\frac{\sqrt{3}}{2})$.

is labelled by p_i. c_i is the *position* in $\mathbb{T}$ of the $(i+1)$th bead, of type p_i, in configuration c. A *partial configuration* of a sequence p is a configuration of a prefix of p.

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

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

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

Oritatami dynamics. The folding of an oritatami system is controlled by the delay δ. Informally, the configuration grows from a *seed configuration* (the input), one bead at a time. This new bead adopts the position(s) that maximise(s) the potential number of bonds the configuration can make when elongated by δ beads in total. This dynamics is *oblivious* as it keeps no memory of the previously preferred positions; it differs thus slightly from the hasty dynamics studied in [6]; it might also be considered as closer to experimental conditions such as in [8].

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

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

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

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

We say that the oritatami system *halts* at time t if t is the first time for which $\mathscr{D}^t(\{\sigma\}) = \varnothing$. The folding process may only stop because of a geometric obstruction (no more elongation is possible because the configuration is trapped in a closed area).

Please refer to Fig. 1(d) and 1(e) for examples of the dynamical folding of a transcript. Observe that a given transcript may fold (deterministically) into different paths because of its interactions with its local environment (see section 2.3 for more details).

2.2 Main result

Our main result consists in proving the following theorem that demonstrates that oritatami systems are able to complete arbitrary Turing computation. It shows in particular that deciding whether a given oritatami system folds into a finite size shape for a given seed is undecidable.

▶ **Theorem 1** (Main result). *There is a fixed set B of 542 bead types with a fixed attraction rule 💔 on B, together with two encodings:*

- *π that maps in polynomial time, any single tape Turing machine $\mathcal{M}$ to a bead type sequence $\pi_{\mathcal{M}} \in B^*$;*
- *(s, σ) that maps in polynomial-time, any single-tape Turing machine $\mathcal{M}$ and any input x of $\mathcal{M}$ to a seed configuration $\sigma_{\mathcal{M}}(x)$ of a bead type sequence $s_{\mathcal{M}}(x)$ of length $O_{\mathcal{M}}(|x|)$, linear in the size of the input x (and polynomial in $|\mathcal{M}|$);*

such that: For any single tape Turing machine $\mathcal{M}$ and every input x of $\mathcal{M}$, the deterministic and periodic oritatami system $\mathcal{O}_{\mathcal{M}} = ((\pi_{\mathcal{M}})^\infty, 💔, 3)$ whose transcript has period $\pi_{\mathcal{M}}$ and whose delay is $\delta = 3$, halts its folding from the seed configuration $\sigma_{\mathcal{M}}(x)$ if and only if $\mathcal{M}$ halts on input x. Furthermore, for all t and all input x of $\mathcal{M}$, if $\mathcal{M}$ halts on x after t steps, then the folding of $\mathcal{O}_{\mathcal{M}}$ from seed configuration $\sigma_{\mathcal{M}}(x)$ halts after folding $O_{\mathcal{M}}(t^4 \log^2 t)$ beads.

There is one Turing-universal periodic transcript. Note that if we apply this theorem to an intrinsically universal single tape Turing machine $\mathcal{U}$ (see [14]), then we obtain one single *absolutely fixed* transcript $\pi_{\mathcal{U}}$ such that the deterministic and periodic oritatami system $\mathcal{O}_{\mathcal{U}} = ((\pi_{\mathcal{U}})^\infty, 💔, 3)$ with 542 bead types can simulate efficiently the halting of any Turing machine $\mathcal{M}$ on any input x using a suitable seed configuration obtained via the encoding of $\mathcal{M}$ and x in $\mathcal{U}$. It follows that this absolutely fixed oritatami system consisting of one single periodic transcript is able of arbitrary Turing computation.

From now on, we only consider deterministic periodic oritatami systems with delay $\delta = 3$.

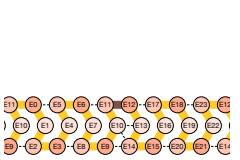
2.3 Basic design tool: Glider/Switchback

As a warm-up, let us introduce a special type of bead sequence (see Fig. 1) that, depending on the initial context of its folding, either folds as a *glider* (a long and thin self-supported shape heading in a fixed direction) or as *switchbacks* (a narrow and high shape allowing compact storage). This only requires a small number of distinct beads types (12 per switchbacks, that can be repeated every 4 switchbacks). This is achieved by designing a rule with minimum interactions ensuring minimum interferences between both folding patterns. Compatibility between the glider and the turns in switchbacks is ensured by aligning the switchback turns with the turns of the glider, exploiting thus the similarity of their finger-like shape there.

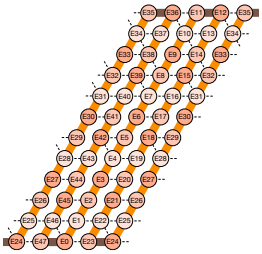
This glider/switchback sequence will be used to store (as switchbacks) and expose (as glider) specific information encoded in the transcript when needed.

2.4 Skipping Cyclic Tag Systems and Turing-Universality

Our proof of the Turing-universality of oritatami systems consists in simulating a special kind of cyclic tag systems (CTS), called skipping cyclic tag system. Cook introduced CTS in [2] and proved that they combined the tremendous advantage of simulating efficiently any Turing machines, while not requiring a random access lookup table, which makes simulation a lot easier.

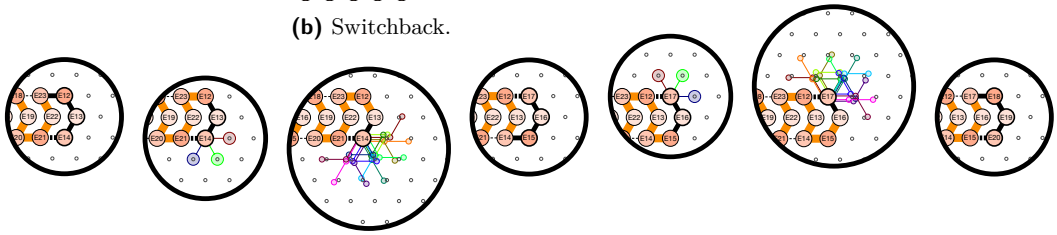


(a) Glider.

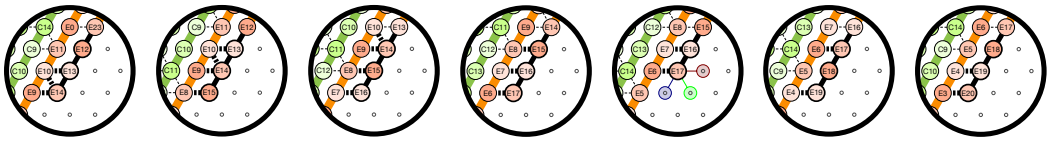


(b) Switchback.

(c) Glider/Switchback turn folding compatibility.



(d) from left to right: the folding of the subsequence as a glider.



(e) from left to right: the folding of the subsequence as a switchback.

Figure 1 Glider/switchback subsequence. The folding path of the transcript is represented as the thick colorful line and the 💗-bonds between beads are represented as dashed lines. The bond-maximizing path for the $\delta = 3$ lastly produced beads is represented by a thick black line, possibly terminated by several colorful paths if several paths realize the maximum of number of bonds.

A skipping cyclic tag system (SCTS). consists of a cyclic list of n words $\alpha = \langle \alpha^0, \ldots, \alpha^{n-1} \rangle \in \{0,1\}^*$, called *appendants*, and an initial *dataword* $u^0 \in \{0,1\}^*$. Intuitively, α encodes the program and u^0 encodes the input. Its configuration at time t consists of a *marker* m^t, recording the index of the current appendant at time t, and a dataword u^t. Initially, $m^0 = 0$ and the dataword is u^0. At each time step t, the SCTS acts deterministically on configuration (m^t, u^t) in one of three ways:

(Halt step) If u^t is the empty word ϵ, then the SCTS halts;[4]

(Nop step) If the first letter u_0^t of u^t is 0, then u_0^t is deleted and the marker moves to the next appendant cyclically: i.e., $m^{t+1} = (m^t + 1) \bmod n$ and $u^{t+1} = u_1^t \cdots u_{|u^t|-1}^t$;

(Skip-append step) If $u_0^t = 1$, then u_0^t is deleted, the next appendant $\alpha^{(m^t+1 \bmod n)}$ is appended onto the right end of u^t, and the marker moves to the second next appendant: i.e., $u^{t+1} = u_1^t \cdots u_{|u^t|-1}^t \cdot \alpha^{(m^t+1 \bmod n)}$ and $m^{t+1} = (m^t + 2) \bmod n$.

For example, consider the SCTS $\mathcal{E} = (\langle 110, \epsilon, 11, 0 \rangle; u^0 = 010)$. Its execution $({}^{[m^t]}u^t)_t$ is:

$$ {}^{[0]}010 \to {}^{[1]}10 \xrightarrow[[2:11]]{\texttt{Append}} {}^{[3]}011 \to {}^{[0]}11 \xrightarrow[[1:\epsilon]]{\texttt{Append}} {}^{[2]}1 \xrightarrow[[3:0]]{\texttt{Append}} {}^{[0]}0 \to {}^{[1]}\texttt{Halt} $$

[4] Note that SCTS halting condition requires the dataword to be empty as opposed to [2, 16] where the computation of a cyclic tag system is said to end also if it repeats a configuration.

Turing universality. A sequence of articles and thesis by Cook [2], and Neary and Woods [16, 13], allows to show that SCTS are able to simulate any Turing machine efficiently in the following sense: (proof omitted see [7])

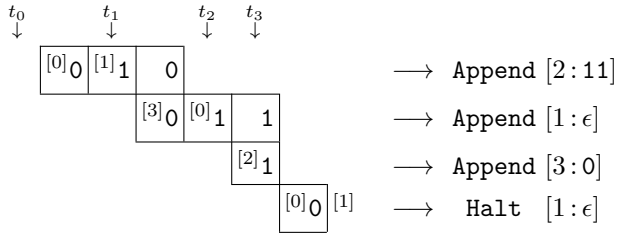
▶ **Proposition 2** ([16, 13]). *Let $\mathcal{M}$ be a deterministic Turing machine using a single tape. There is a polynomial algorithm that computes a skipping cyclic tag system $\mathcal{S}_\mathcal{M}$, together with a linear-time encoding $u_\mathcal{M}(x)$ of the input x of $\mathcal{M}$ as an input dataword for $\mathcal{S}_\mathcal{M}$, such that for all input x: $\mathcal{S}_\mathcal{M}$ halts on input dataword $u_\mathcal{M}(x)$ if and only if $\mathcal{M}$ halts on input x. Furthermore, for all t, if $\mathcal{M}$ halts after t steps, then $\mathcal{S}$ halts after $O_\mathcal{M}(t^2 \log t)$ steps. Moreover, the number of appendants of $\mathcal{S}$ is a multiple of 4.*

In order to prove Theorem 1, we are thus left with proving that there is an oritatami system that simulates in quadratic time any SCTS system (see Theorem 6 in [7] for a precise statement).

3 The block simulation of SCTS: Proving the correctness of local folding is enough

Given a SCTS $\mathcal{S}$, we design an oritatami system $\mathcal{O}_\mathcal{S}$ that folds into a version, at a larger scale, of the *annotated trimmed space-time diagram* of $\mathcal{S}$ (or *trimmed diagram* for short) defined as follows:

Trimmed diagram of SCTS. Any SCTS proceeds as follows: it trims all the leading 0s in the dataword and then appends the currently marked appendant when it reads the first 1 (if any; otherwise it halts). It is thus natural to group all these steps (trim leading 0s and process the leading 1) as one single macro step. This motivates the following representation. Given a SCTS $(\alpha^0, \ldots, \alpha^{n-1}; u^0)$, we denote by $0 \leqslant t_1 < t_2 < \cdots$ all the times t such that the dataword u^t starts with letter 1 and set $t_0 = -1$ by convention. Let us now group all deletion steps occurring during steps $t_i + 1$ to $t_{i+1} - 1$ by simply indicating in exponent the marker m^t before each letter read. In the case of our STCS $\mathcal{E}$, we have $t_0 = -1, t_1 = 1, t_2 = 3, t_3 = 4$ and its execution is now represented as: $^{[0]}0^{[1]}10 \xrightarrow[[2:11]]{\texttt{Append}} {}^{[3]}0^{[0]}11 \xrightarrow[[1:\epsilon]]{\texttt{Append}} {}^{[2]}1 \xrightarrow[[3:0]]{\texttt{Append}} {}^{[0]}0^{[1]}\,\texttt{Halt}$. Now, let's align the resulting datawords in a 2D diagram according to their common parts:



This defines the *annotated trimmed space-time diagram* for the SCTS $\mathcal{E}$. We refer to Lemma 4 in [7] for the formal definition for an arbitrary SCTS.

The transcript. The proof of Theorem 1 (see Theorem 6 in [7]) relies on constructing a transcript (and a fixed rule) that will reproduce faithfully the trimmed diagram of the simulated STCS. Figure 2 illustrates the folded configuration of the transcript corresponding to SCTS $\mathcal{E}$. Macroscopically, the transcript folds into a zig-zag sequence of *blocks*, each performing a specific operation.

The current dataword is encoded at the bottom of each row of blocks: 0s are encoded by a spike, and 1s are encoded by a flat surface.

The seed configuration encodes the initial dataword and opens the first zig row at which the folding of the transcript starts. Letters 0 and 1 are encoded by a *spike* (see Fig. 3(a)) and a *flat surface* (see Fig. 3(b)) respectively.

In each zig row (left to right), the transcript folds into a series of Read0▶ blocks (trimming the leading 0s from the dataword encoded above), and then into a Read1▶ block, if the dataword contains a 1, or into a Halt block terminating the folding, otherwise; this is the *zig-up phase*. Then, the transcript starts the *zig-down phase* which consists in folding into Copy▶ block copying the remaining letters of the dataword encoded above to the bottom of the row; once the end of the dataword is reached, the transcript folds into an Append◥Return block which encodes, at the bottom of the row, the currently marked appendant, and finally, opens the next zig row.

In each zag row (right to left), the transcript folds into Copy◀ blocks copying the new dataword encoded above to the bottom of the row. For the leftmost letter, the transcript folds into the special Copy◤LineFeed block which also opens the next zig row.

The transcript is a periodic sequence whose period is the concatenation of n bead type sequences **Appendant** α^0, ..., **Appendant** α^{n-1} called *segments*, each encoding one appendant.

Encoding of the marker. Read▶ and Append◥Return blocks consist of the folding of *exactly one* segment, whereas Copy▶, Copy◀ and Copy◤LineFeed consist of the folding of *exactly* n segments. It follows that the appendant encoded in the *leading* segment folded inside each block corresponds to the *currently marked* appendant in the simulated SCTS. As a consequence, the appendant contained in the folded Append◥Return block is indeed the appendant to be appended to the dataword.

The segment sequence. Each segment **Appendant** α^i encodes the appendant α^i as a sequence of $6 + |\alpha^i|$ modules: one of each module A, B, and C, then $|\alpha^i|$ of module D, then one of each module E, F and G. Each module is a bead type sequence that plays a particular role in the design:

Module A folds into the initial scaffold upon which the next modules rely.

Module B detects if the dataword is empty: if so, it folds to the left so as the folding gets trapped in a closed space and halts; otherwise, it folds to the right and the folding continues.

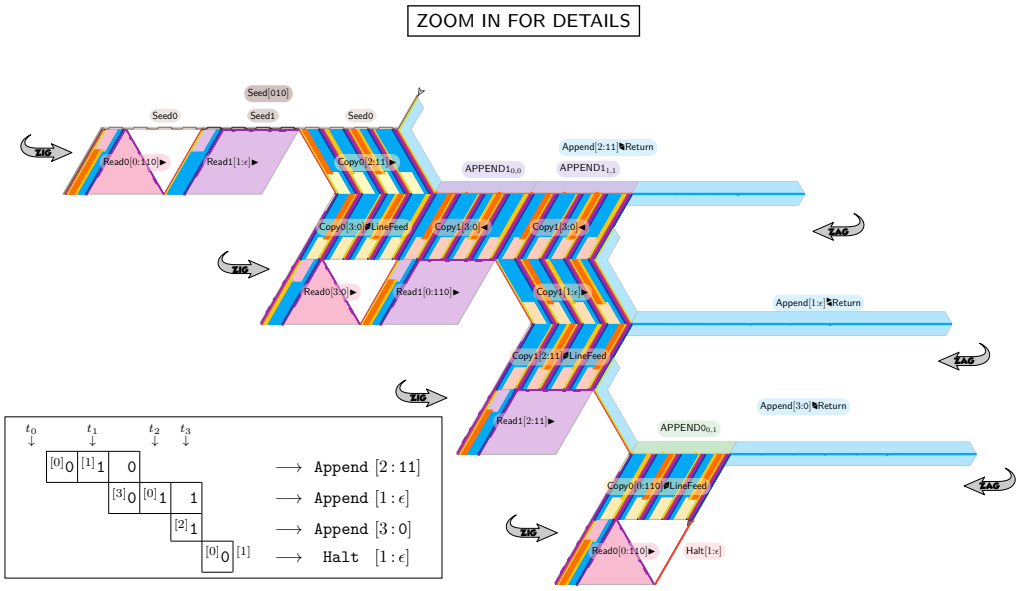
Module C detects the end of the dataword and triggers the appending of the marked appendant accordingly.

Module D encodes each letter of the appendant: its two variants D0 and D1 encode respectively 0s and 1s .

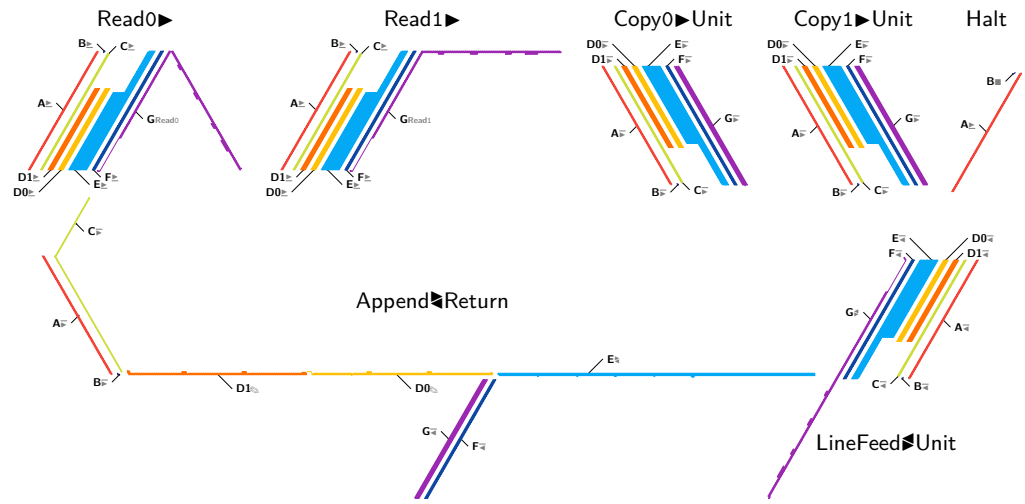
Module E ensures by padding that all appendant sequences have the same length when folded (even if the appendant have different length). It serves two other purposes: Module B senses its presence to detect if the dataword is empty; and its folding initiates the opening of the zag row once the marked appendant has been appended to the dataword.

Module F is the scaffold upon which module G folds. It is specially designed to induce two very distinct shapes on G depending on the initial shift of G. Furthermore, when module F is exposed, module C folds along F which triggers the appending of the marked appendant encoded by the modules D following C.

Module G is the "logical unit" of the transcript. It implements three distinct functions which are triggered by its interactions with its environment: (1) reading the $0^*(1|\epsilon)$ prefix of the dataword, (2) copying a letter of the dataword, and (3) opening the next zig row at the leftmost end of each zag row.



(a) Folding of the oritatami system simulating the STCS $\mathcal{E}$.



(b) Exploded view of the bricks and modules inside the blocks involved in the simulation above.

Figure 2 Folding of the transcript simulating the STCS $\mathcal{E}$, and some block internal structures.

We call *bricks* the folding of each of these modules. The blocks into which the transcript folds, depend on the bricks in which its modules fold, as illustrated in Fig. 2(b). We refer to sections C to F in [7] for the description of blocks in terms of bricks and of how they articulate with each other to produce the desired macroscopic folding pattern.

The full description of each of these sequences is given in Section F in [7].

Let $\mathcal{S} = (\alpha^0, \alpha^1, \ldots, \alpha^{n-1}; u^0)$ be a SCTS, and, as before, let for all integer $i \geqslant 0$, t_i be the i^{th} step where u^t starts with `1` (starting from 0, i.e. t_0 is the first step where u^{t_0} starts with `1`). The following lemma shows that the transcript described above folds indeed into blocks that simulate the trimmed diagram of $\mathcal{S}$. Proposition 2 and Theorem 1 are direct corollaries of this lemma.

▶ **Lemma 3** (Key lemma). *There is a bead type set B and a rule 💔 such that: for every SCTS $\mathcal{S}$, there are $\pi_{\mathcal{S}}$ and $(\sigma_{\mathcal{S}}, s_{\mathcal{S}})$ defined as in Theorem 1 such that, for every initial dataword u^0, the (possibly infinite) final folded path of the oritatami system $\mathcal{O}_{\mathcal{S}} = ((\pi_{\mathcal{S}})^\infty, 💔, \delta = 3)$ from the seed configuration $(\sigma(u^0), s(u^0))$ is exactly structured as the following sequence of blocks organized in zig and zag rows as follows: (recall Fig. 2(a))*

- *First, the block* Seed(u^0) *ending at coordinates* $(-1, 0)$.
- *Then, for $i \geqslant 0$, the i-th row consists of a zig row located between $y = 2(i-1)h + 1$ and $y = 2ih$, and a zag row located between $y = 2ih + 1$ and $y = 2(i+1)h$, composed as follows:*
  - **(Compute)** ***if*** $u^{1+t_i} = 0^r 1 \cdot s$ ***and if*** $s \neq \epsilon$ ***or*** $\alpha^{1+i+t_{i+1}} \neq \epsilon$***:*** *then $r = t_{i+1} - t_i - 1$ and:*
    - *the i-th zig-row consists from left to right of the following sequence of blocks whose origins are located at the following coordinates:*

$\swarrow y$	$2ih$				$(2i-1)h+1$			
$\rightarrow x$	$ih + (1+t_i)W$	$\cdots$	$ih + (t_{i+1}-1)W$	$ih + t_{i+1}W$	$ih + (1+t_{i+1})W - 1$	$\cdots$	$ih + (\lvert s\rvert + t_{i+1})W - 1$	$ih + (1 + \lvert s\rvert + t_{i+1})W - 1$
Blocks	Read0▶	$\cdots$	Read0▶	Read1▶	Copy(s_0)▶	$\cdots$	Copy($s_{\lvert s\rvert-1}$)▶	Append[$\alpha^{1+i+t_{i+1}}$]◥Return
Marker	$i+1+t_i$	$\cdots$	$i+r+t_i$	$i+t_{i+1}$	$i+1+t_{i+1}$	$\cdots$	$i+1+t_{i+1}$	$i+1+t_{i+1}$

    This row ends at position $((i+1)h + (1 + |s| + |\alpha^{i+1+t_{i+1}}| + t_{i+1})W - 7, 2ih + 2)$.
    - *the i-th zag-row consists from right to left of the following sequence of blocks whose origins are located at the following coordinates:*

$\swarrow y$	$2ih+1$		
$\rightarrow x$	$(i+1)h + (2+t_{i+1})W - 8$	$(i+1)h + (3+t_{i+1})W - 8$	$\cdots$	$(i+1)h + (1 + \lvert v\rvert + t_{i+1})W - 8$
Blocks	Copy(v_0)◤LineFeed	Copy(v_1)◀	$\cdots$	Copy($v_{\lvert v\rvert-1}$)◀
Marker	$i+2+t_{i+1}$	$i+2+t_{i+1}$	$\cdots$	$i+2+t_{i+1}$

    where $v = u^{1+t_{i+1}} = s \cdot p_{i+1+t_{i+1}} \neq \epsilon$ (as s and $\alpha^{i+1+t_{i+1}}$ are not both ϵ). This row ends at position $((i+1)h + (1+t_{i+1})W - 1, 2(i+1)h)$.
  - **(Halt 1)** ***if*** $u^{1+t_i} = 0^r 1$ ***and*** $\alpha^{1+i+t_{i+1}} = \epsilon$***:*** *then $r = t_{i+1} - t_i - 1$ and the last rows of the configuration consists from left to right of the following sequence of blocks located at the following coordinates:*

$\swarrow y$	$2ih$				$(2i-1)h+1$	$2(i+1)h$
$\rightarrow x$	$ih + (1+t_i)W$	$\cdots$	$ih + (t_{i+1}-1)W$	$ih + t_{i+1}W$	$ih + (1+t_{i+1})W - 1$	$(i+1)h + (1+t_{i+1})W$
Blocks	Read0▶	$\cdots$	Read0▶	Read1▶	CarriageReturn◥LineFeed◢	Halt
Marker	$i+1+t_i$	$\cdots$	$i+t_{i+1}-1$	$i+t_{i+1}$	$i+1+t_{i+1}$	$i+2+t_{i+1}$

- ***finally,*** **(Halt 2)** ***if*** $u^{1+t_i} = 0^r$ ***for some*** $r \geqslant 0$***:*** *then the i-th zig-row is last row of the configuration and consists of the following sequence of blocks located at the following coordinates:*

$\swarrow y$	$2ih$			
$\rightarrow x$	$ih + (1+t_i)W$	$\cdots$	$ih + (r+t_i)W$	$ih + (1+r+t_i)W$
Blocks	Read0▶	$\cdots$	Read0▶	Halt
Marker	$i+1+t_i$	$\cdots$	$i+r+t_i$	$i+r+1+t_i$

The following sections are dedicated to the proof of Key Lemma 3.

4 Advanced Design Tool box

In this section, we present several key tools to program Oritatami design and which we believe to be generic as they allowed us to get a lot of freedom in our design.

4.1 Implementing the logic

As in [6], the internal state of our "molecular computing machinery" consists essentially of two parameters: 1) the *position inside the transcript* of the part currently folding; and 2) the *entry point* of transcript inside the environment. Indeed, depending on the entry point or the position inside the transcript, different beads will be in contact with the environment and thus different *functions* will be applied as a result of their interactions. This happens during the zig phase: in the first (zig-up) part, the transcript starts folding at the bottom, forcing the modules G to fold into G▶Read bricks; whereas during the second (zig-down) part, the transcript starts folding at the top, forcing the modules G to fold into G▶̄Copy bricks instead. Similarly, the *memory* of the system consists of the beads already placed on the surrounding of the area currently visited (the *environment*). This happens in every row of the folding: depending on the letter encoded at the bottom of the row above, the modules G fold into G▶Read0 or G▶Read1 bricks (zig-up phase), G▶̄Copy0 or G▶̄Copy1 bricks (zig-down phase), and G◀Copy0 or G◀Copy1 bricks (zag phase).

At different places, we need the transcript to read information from the environment and trigger the appropriate folding. This is obtained through different mechanisms.

Default folding. By default, during the zig-up phase, B is attracted to the left by F and folds to the right only in presence of E above. This allows to continue the folding only if the tape word is not empty or to halt it otherwise (see Figure 27 in [7]).

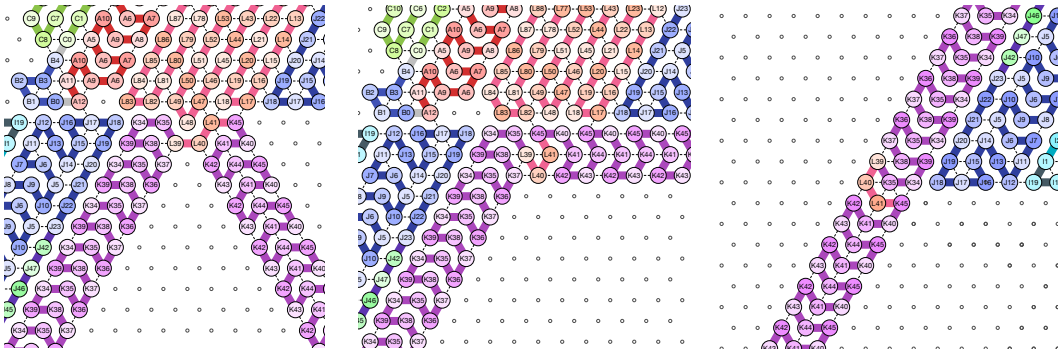
Geometry obstruction. A typical example is illustrated by G. During the zig-up phase where the absence of environment below the block Read▶ allows G to fold downward at the beginning (see Figure 41 in [7]) which shift the transcript by 7 beads along F resulting in G to adopt the glider-shape (more details on this mechanism in the next section). Whereas during the zig-down phase, G cannot make this loop because it is occupied by a previously placed G. This results in a perfect alignment of G with F whose strong attraction forces G to adopt the switchback shape (see Figure 43 in [7]).

Geometry of the environment. Figure 3 shows how the shape of the environment is used to change the direction of G in glider-shape. This results in modifying the entry point in the environment and allows the Oritatami system to trim the leading 0s in the tape word by going back to the same entry point (Fig. 3(a)), switch from zip-up to zig-down phase when reading a 1 by opening the next block from the top (Fig. 3(b)), and from zag to zig-up phase when it has rewound to the beginning of the tape word, by getting down to the bottom of the next zig row (Fig. 3(c)).

4.2 Easing the design: getting the freedom you need

Several key tools allowed to ease considerably our design, and even in some cases to make it feasible. These tools are generic enough to be considered as *programming paradigms*. One main difficulty we faced is that the different functions one wants to implement tend to concentrate at the same "hot-spots" in the transcript. A typical example is the midpoint of G where one wants to implement all the functions: Read, Copy and Line Feed. The following powerful tools allow to overcome these difficulties:

Socks work by letting a glider/switchback module fold into a switchback turn conformation for some time when it would otherwise fold into a glider. Examples are given in Figure 4. They are easy to implement: indeed, the socks naturally adopt the same shape as the corresponding switchback turn and require thus *no extra interfering bonds*. They allow a lot of freedom in the design, for several reasons:

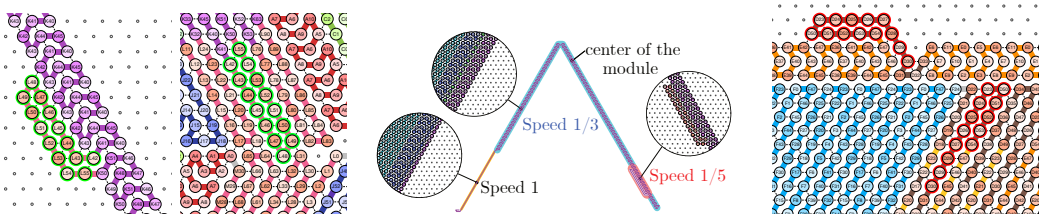


(a) G bounces southeastward in presence of a spike encoding a 0 and folds into G ▶ Read0.

(b) G bounces eastward on a flat surface encoding a 1, and folds into G ▶ Read1.

(c) G goes straight southwestward in absence of obstacle, and folds into G ⧩ LineFeed.

Figure 3 The interactions of module G in "glider"-mode with different environments result in heading to different entry points to the next area of the folding.



(a) Easier bond design.

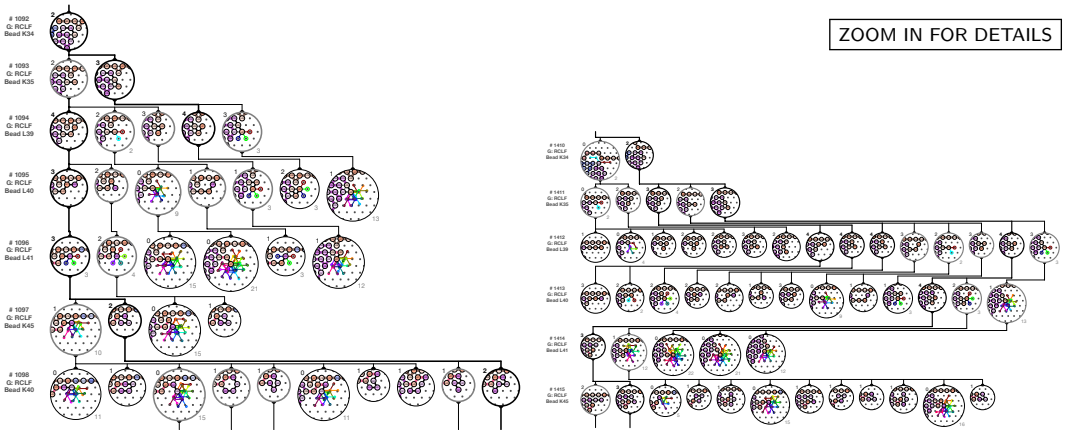
(b) Delaying gliders.

(c) Confinement.

Figure 4 Different uses of socks: (a) Easing the design of switchback/glider by letting the switchback (in green) folds into its natural shape at its extremities even in "glider"-mode.; (b) Module G: Realigning a pattern by slowing its folding down at the end to compensate speeding it at the beginning.; (c) Module D: Preventing unwanted interactions between the beads outlined in red and the ones below, by concealing the red ones on top of the glider.

- First, they simplify the design of important switchback part by *lifting the need for implementing the glider configuration* for that part, as shown in Figure 4(a).
- Second, a glider naturally progresses at speed 1/3. Adding a sock allows us to *slow its progression down* to speed 1/5 for some time (see Fig. 4(b)) and therefore to realign them. We used that feature repeatedly to "shift" some modules: starting the folding at an initial speed-1 (i.e. straight line) and then compensating for that speed later on by introducing socks (see Fig. 4(b)). This is a key point in our design, as it allowed us to *spread apart* the Read and Copy functions into different subsequences of module G, and therefore to get less constraints on our rule design. In the specific case of module G, the Copy-function occurs at the center of the module, while the Read-function is implemented earlier in module! (see section F.10 in [7] for full details)
- Finally, socks allow to prevent unwanted interactions between beads by *concealing* potentially harmful beads in an unreachable area as in Figure 4(c).

Exponential bead type coloring is a key tool to allow module G to fold into different shapes, glider or switchback, along module F, when folding in the Read▶ configuration. The problem it solves is that in order for G to fold into switchbacks, we need strong interactions between G and its neighboring module F (see Fig. 41 in [7]), whereas in order for G to fold as glider, we want to avoid those interactions (see Fig. 43 in [7]).



(a) Proof tree for the glider turn in G ▶ Read0. **(b)** Proof tree for the glider turn in G ▶ Read1.

Figure 5 Two examples of proof trees for the same subsequence in two different environements. The number at the upper-left corner of every ball stands for the number of bonds for the path inside the ball. The number at the lower right corner of each ball stands for the number of paths grouped in the ball, allowing to check that no path was omitted. Balls highlighted in black bold contain the bonds-maximizing paths. Balls highlighted in grey bold contain the paths that places the bead at the same location as the bonds-maximizing paths, and which must thus be considered in the next level as well.

This is made possible because gliders progress at speed 1/3 while switchbacks progress at speed 1. Using a power-of-3 coloring, we manage to easily achieve these contradicting goals altogether (the construction is analysed in Lemma 11 in Section G.1 in [7]).

5 Correctness of local folding: Proof tree certificates

The goal of this section is to conclude the proof of our design by proving Key Lemma 3. The proof works by induction, assuming that the preceding beads of the transcript fold at the locations claimed by the lemma. We proceed in 3 steps:

- We first enumerate all the possible environments for every part of the transcript. As, we carefully aligned our design, most of the beads only see a small number of different environments.
- For the few cases (three in total) where the number of environments is unbounded, we give an explicit proof of correctness of their folding (Lemmas 9–11 in section G.1 in [7]). This is where the concealing feature of socks and the exponential bead type coloring play a crucial role.
- For all the other cases, we designed human-checkable computer-generated certificates, called *proof trees*. It consists in listing in a compact but readable manner all the possible paths for the transcript in every possible environment. In order to match human readability, paths with identical bonding patterns are grouped into one single ball. Balls containing the paths maximizing the number of bonds are highlighted in bold and organized in a tree. This reduces the number of cases to less than 5 balls in most of the levels of the tree, achieving human-checkability of the computed certificate (see Fig. 5). Proof trees are available at `https://www.irif.fr/~nschaban/oritatami/`.

References

1 J. Boyle, G. Robillard, and S. Kim. Sequential folding of transfer RNA. A nuclear magnetic resonance study of successively longer tRNA fragments with a common 5' endA nuclear magnetic resonance study of successively longer tRNA fragments with a common 5' end. *J. Mol. Biol.*, 139:601–625, 1980.

2 M. Cook. Universality in Elementary Cellular Automata. *Complex Systems*, 15:1–40, 2004.

3 E.D. Demaine, J. Hendricks, M. Olsen, M.J. Patitz, T. Rogers N. Schabanel, S. Seki, and H. Thomas. Know When to Fold 'Em: Self-Assembly of Shapes by Folding in Oritatami. In *DNA*, 2018. To be published.

4 K. L. Frieda and S. M. Block. Direct observation of cotranscriptional folding in an adenine riboswitch. *Science*, 338(6105):397–400, 2012.

5 P. Gács. Reliable cellular automata with self-organization. In *FOCS*, pages 90–99, 1997.

6 C. Geary, P.-É. Meunier, N. Schabanel, and S. Seki. Programming Biomolecules that Fold Greedily during Transcription. In *MFCS*, volume LIPIcs 58, pages 43:1–43:14, 2016.

7 C. Geary, P-É Meunier, N. Schabanel, and S. Seki. Proving the Turing Universality of Oritatami Co-Transcriptional Folding (Full Text). *CoRR*, 2018. `arXiv:1508.00510`.

8 C. Geary, P. W. K. Rothemund, and E. S. Andersen. A Single-Stranded Architecture for Cotranscriptional Folding of RNA Nanostructures. *Science*, 345:799–804, 2014.

9 Y-S. Han and H. Kim. Ruleset Optimization on Isomorphic Oritatami Systems. In *Proc. DNA23*, volume LNCS 10467, pages 33–45. Springer, 2017.

10 Y.-S. Han, H. Kim, M. Ota, and S. Seki. Non-deterministic Seedless Oritatami Systems and Hardness of Testing Their Equivalence. *Natural Computing*, 17(1):67–79, 2018.

11 L. Kari, S. Kopecki, P.-É. Meunier, M. J. Patitz, and S. Seki. Binary Pattern Tile Set Synthesis Is NP-hard. In *ICALP*, volume LNCS 9134, pages 1022–1034, 2015.

12 Y. Masuda, S. Seki, and Y. Ubukata. Towards the Algorithmic Molecular Self-Assembly of Fractals by Cotranscriptional Folding. In *CIAA*, volume LNCS 10977, pages 261–273, 2018.

13 T. Neary. *Small universal Turing machines*. PhD thesis, NUI, Maynooth, 2008.

14 N. Ollinger. The quest for small universal cellular automata. In *ICALP*, volume LNCS 2380, pages 318–329, 2002.

15 M. Ota and S. Seki. Ruleset Design Problems for Oritatami Systems. *Theor. Comput. Sci.*, 671:26–35, 2017.

16 D. Woods and T. Neary. On The Time Complexity of 2-tag Systems and Small Universal Turing Machines. In *FOCS*, pages 439–448, 2006.