Natural Computing (2023) 22:563–585
https://doi.org/10.1007/s11047-023-09963-0

Programmable single-stranded architectures for computing

Yu Kihara[1] · Shinnosuke Seki[1]

Accepted: 3 August 2023 / Published online: 26 September 2023

Abstract
Oritatami is a mathematical model of co-transcriptional folding, a phenomenon in which, while being synthesized (*transcribed*) sequentially, an RNA sequence folds upon itself into complex structures via hydrogen bonds between its nucleotides (A, C, G, and U). RNA sequences fold co-transcriptionally to perform computations in-vivo such as gene expression regulation and splicing. Co-transcriptional folding has been recently proven modularly programmable for assembling structures in-vitro in the RNA origami framework as well as for computing arbitrary computable functions in-silico using the oritatami model. In this tutorial, we overview computations in oritatami and their "bricks" to build up from, that is, modules, and then discuss what should be done along with concrete open problems as a seed for further fruitful developments in computation by co-transcriptional folding.

Keywords RNA co-transcriptional folding · RNA origami · Algorithmic molecular self-assembly · Oritatami model

1 Modular approach to the programming of co-transcriptional folding

Transcription is a process by which an enzyme called RNA polymerase synthesizes a single-stranded RNA sequence according to its DNA template sequence (see Figs. 1 and 2), in which the template is read nucleotide by nucleotide from its 3'-end to 5'-end, and the growing RNA sequence is extended by one of the four types of nucleotide (A, C, G, U) energetically favored by the DNA nucleotide read, that is, A on the DNA with U on the RNA, T with A, C with G, and G with C. The product is called a *(RNA) transcript*. While being thus synthesized sequentially, the transcript tends to fold upon itself via binding between nucleotides to get stabilized, where nucleotides can bind no matter which of the 4 types they are, though A-U, C-G, and G-U are favored considerably. This phenomenon is called *(RNA) co-transcriptional folding*.

This work is supported in part by KAKENHI Grand-in-Aid for Scientific Research (B) No. 20H04141 and (C) No. 20K11672 to S. S. Let us express our sincere gratitudes towards anonymous referees for their valuable comments and suggestions on the previous drafts. Some of the artworks in this article were generated by using the Simple OS Simulator developed by Nicolas Schabanel (2016).

✉ Shinnosuke Seki
s.seki@uec.ac.jp

[1] University of Electro-Communications, Tokyo, Japan, 1-5-1, Chofu 1828585, Chofugaoka, Japan

Co-transcriptional folding can be actually even *programmed* in-silico in order to self-assemble various structures out of a single-stranded RNA transcript in-vitro. This breakthrough is founded on the *RNA origami* architecture by Geary et al. (2014). In this architecture, an artificial RNA tile structure of 450 nucleotides was programmed for the first time into an RNA transcript, or more precisely, into its DNA template sequence in such a manner that the tile structure can be retrieved later by letting the transcript fold cotranscriptionally in-vitro, as shown in Fig. 2. Taking the *modular programming approach*, which has been proven quite useful in nucleic acids structure design (see, e.g., Jaeger and Chworos 2006), nowadays copies of basic RNA origami tiles can be merged via structural motifs such as kissing loops (KLs) into tiles of several thousand nucleotides in length with the aid of RNA Origami Automated Design (ROAD) (Geary et al. 2021). ROAD provides a software package to enable programmers to build-up RNA origami structures with various scale and functionality modularly from a library of structural modules such as helical turns, antiparallel crossovers, and KLs of several kinds and to optimize the design of a transcript and folding pathway, along which the transcript folds into the target structure.

What comes next? Computation, of course! Co-transcriptional folding is a major driving force of computing in-vivo including gene expression regulation (Watters et al. 2016) and splicing (Merkhofer et al. 2014), to name a few examples, but yet to be applied for human-made computation. Note

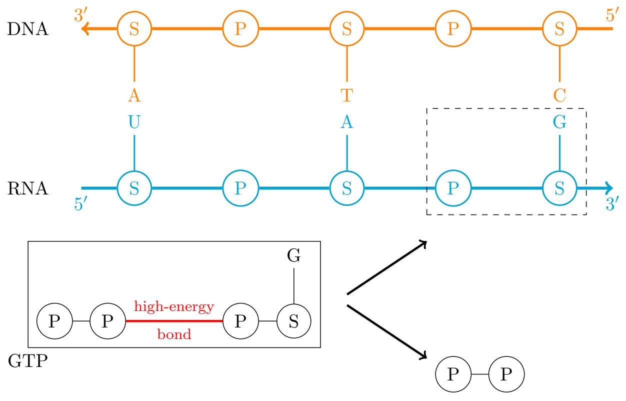

Fig. 1 A “Feynman” diagram of transcription (Feynman 1996). S and P stand for sugar and phosphate, respectively. The DNA template sequence is read and the corresponding RNA sequence (transcript) is synthesized nucleotide by nucleotide according to the mapping A → U, C → G, G → C, T → A

that folding and transcription have been already incorporated into computational protocols in-vitro such as whiplash PCR (Hagiya et al. 1997) and its isothermal variants (Reif and Majumder 2010; Rose et al. 2006), but their computations go through hairpin (de)formation and transcription alternately, rather than being driven by co-transcriptional folding. Being co-transcriptional poses a novel challenge; once being written on a transcript, a code cannot be modified at run-time anyhow. Hence, if the folding pathway were designed so finely as done in the RNA origami, the transcript would be entirely deprived of capability to do any computation. In fact, the co-transcriptional folding pathway for the gene expression regulation mentioned above undergoes bifurcation that either leads to the formation of a terminator stem, which inhibits gene expression, or its collapse into a pseudoknot (Watters et al. 2016). Multiple folding pathways are indispensable for computing by a single linear molecule.

Oritatami is the first mathematical platform to study programming multiple co-transcriptional folding pathways. It lets a transcript of abstract molecules (called *beads*) of finite kinds be elongated one bead per unit time and fold into a directed self-avoiding path on the triangular grid graph co-transcriptionally. An oritatami system has been designed for the first time by Geary et al. to count in binary (Geary et al. 2016). It alternately transcribes a half-adder module, which accommodates four folding pathways pairwise distinct enough to let the succeeding computation know what was computed, and a structural module, which guides the transcript into a zigzag folding pathway and propagates a 1 bit information between half-adders. Turing-universal oritatami systems (Geary et al. 2018; Pchelina et al. 2022, 2020) are a few orders of magnitude greater in size and functionality but still built up from modules that play basic functions such as 1-bit read/write, 1-bit propagation, and referencing to a lookup table. Oritatami is however “too close to the molecular hardware” to let us call these “basic” functions as handily as we do in conventional programming by high-level programming languages such as C++ or Java. Indeed, these Turing universal oritatami systems have been developed over years, after bringing home to us more than a few times how non-trivial it is to compute co-transcriptionally something simple in-silico.

As a high-level programming language for co-transcriptio nal computations, *Turedo* has been proposed recently by Pchelina et al., together with a compiler to translate a Turedo program into an oritatami machine code (Pchelina et al. 2022). It is a novel variant of 2D Turing machines. Turedo programmers can utilize a wealth of knowledge and techniques accumulated over almost a century. After the oritatami model and Turedo are described in Sect. 2 and at the beginning of Sect. 3, respectively, we shall present in the rest of Sect. 3 a basic principle of how the compiler works with the aid of Example 2, which describes how the discrete Sierpinski triangle can self-assemble in Turedo. The compiler can handle only Turedos at radius 1, that is, those which can observe only the adjacent cells, and furthermore their translation in oritatami thus compiled operates only at a specific transcription speed that has not been justified anyhow experimentally yet. Therefore, it should be of some help to explain in Sect. 4 how oritatami systems have been designed.

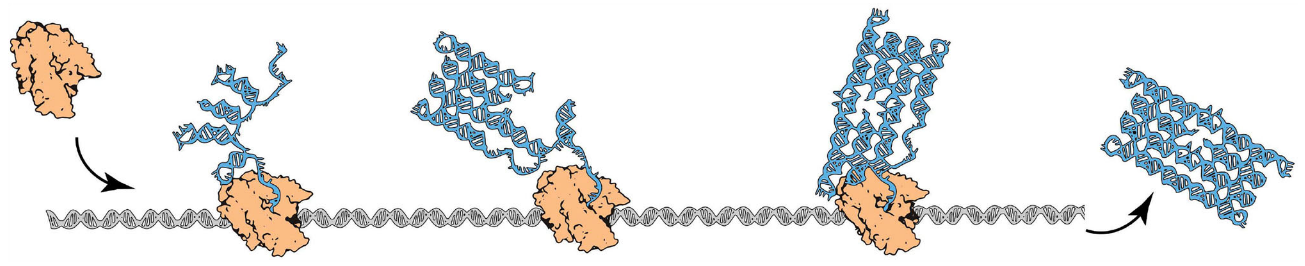

Fig. 2 Co-transcriptional folding and RNA origami (Geary et al. 2014). The RNA polymerase scans the DNA template (colored in grey) sequentially and maps its nucleotides one by one to its RNA counterpart as illustrated in Fig. 1, synthesizing (*transcribing*) the encoded RNA sequence (blue), sequentially. The resulting RNA transcript folds upon itself *co-transcriptionally*

This tutorial proposes open problems at the end of every section, which hopefully will serve as a seed to open fruitful future research. In order to focus on computations, shape self-assembly in oritatami (Demaine et al. 2018; Fazekas et al. 2022; Han and Kim 2018, 2021; Masuda et al. 2018) is barely covered. In order to make best use of this tutorial, readers are assumed to have basic knowledge in theory of computation (Hopcroft et al. 2001) and complexity (Arora and Barak 2009; Geary and Johnson 1979), graph theory (Diestel 2010), and molecular biology (Alberts et al. 2014; Elliott and Ladomery 2016).

2 Oritatami model

Let Σ be a finite set (alphabet) of types of an abstract molecule called a *bead*. Beads can represent whatever capable of binding in RNA origami including nucleotides (unit of transcription), kissing loops, and even larger subunits. Modified alphabets such as $\Sigma' = \{a' \mid a \in \Sigma\}$ are of use; unless otherwise noted, we suppose that an alphabet modified anyhow is disjoint from its original, e.g., $\Sigma \cap \Sigma' = \emptyset$. By Σ^* and Σ^ω, we denote the set of finite sequences of letters in Σ and that of infinite sequences of letters in Σ, respectively. Let $w = a_1a_2\cdots$ be a sequence of letters $a_1, a_2, \ldots \in \Sigma$. For indices i, j, the infix $a_ia_{i+1}\cdots a_j$ of w is referred to as $w[i..j]$; if $i = j$, this notation is simplified as $w[i]$ and refers to the i-th letter of w. If w is finite as $w = a_1a_2\cdots a_n$ for some $a_1, \ldots, a_n \in \Sigma$, then by $|w|$ we denote its length, that is, $|w| = n$. The Watson-Crick complementarity in RNA, A-U and C-G, which induces the helical structure, can be modeled by the antimorphic involution $\theta : \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}^* \to \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}^*$ that is defined based on the involution $\theta(\mathtt{A}) = \mathtt{U}, \theta(\mathtt{C}) = \mathtt{G}, \theta(\mathtt{G}) = \mathtt{C}$, and $\theta(\mathtt{U}) = \mathtt{A}$ and extended to the antimorphism, that is, $\theta(xy) = \theta(y)\theta(x)$ for any words $x, y \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}^*$. This maps, for instance, an RNA sequence UACG to $\theta(\mathtt{UACG}) = \theta(\mathtt{G})\theta(\mathtt{UAC}) = \cdots = \theta(\mathtt{G})\theta(\mathtt{C})\theta(\mathtt{A})\theta(\mathtt{U}) = \mathtt{CGUA}$. For $x, y \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}^*$, the RNA sequence $xy\theta(x)$ folds into a hairpin via nested base pairs between x and $\theta(x)$ (the k-th base of x with the k-th last base of $\theta(x)$) as long as y is of length at least 3, serving as a loop of the hairpin.

In the oritatami model, a pre-programmed sequence w of letters in Σ (beads) folds over the triangular grid graph $\mathbb{T} = (\mathbb{Z}^2, \sim)$ into a directed self-avoiding path while being synthesized (*transcribed*) one bead per unit time, where for two lattice points $p, q \in \mathbb{Z}^2$, $p \sim q$ means that they are adjacent to each other. Folding *co-transcriptionally* in oritatami means that the prefix of w transcribed so far lets only its suffix of length δ, that is, the most recently-synthesized (*nascent*) δ beads, move as long as the backbone between beads does not get broken or intersect, where δ is a system parameter explained later (see Fig. 3 for the case $\delta = 3$). An *oritatami system* $\mathcal{O} = (w, \heartsuit, \delta, \alpha)$ is composed of a (possibly infinite) bead type sequence $w \in \Sigma^* \cup \Sigma^\omega$ called a *transcript*, a symmetric relation $\heartsuit \subseteq \Sigma^2$ called an *(attraction) rule (set)*, and two integer parameters $\delta, \alpha \geq 1$ called *delay* and *arity*, respectively. It is said to be *periodic* with a period $p \geq 1$, or more simply *of period* p, if its transcript w is periodic as $w[i + p] = w[i]$ for all $i \geq 1$.

Beads of types $a, b \in \Sigma$ *attract* each other if $a \heartsuit b$, and a hydrogen bond can form between them as long as they are at unit distance. A bead type $a \in \Sigma$ is said to be *inert* if there is no bead type $b \in \Sigma$ such that $a \heartsuit b$. See Fig. 3. A *conformation* c of a bead-type sequence $u \in \Sigma^* \cup \Sigma^\omega$ is a pair (p, B) of a directed self-avoiding path p in $\mathbb{T}$ along which u folds, that is, $u[i]$ at $p[i]$, and a subset B of the set of all possible bonds: $\{(i, j) \mid j - i \geq 2, u[i] \heartsuit u[j], \text{ and } p[i] \sim p[j]\}$ (this is exactly a configuration of an oritatami system but we prefer the terminology "conformation" as it abstracts a yield of folding an RNA sequence, which is called a conformation

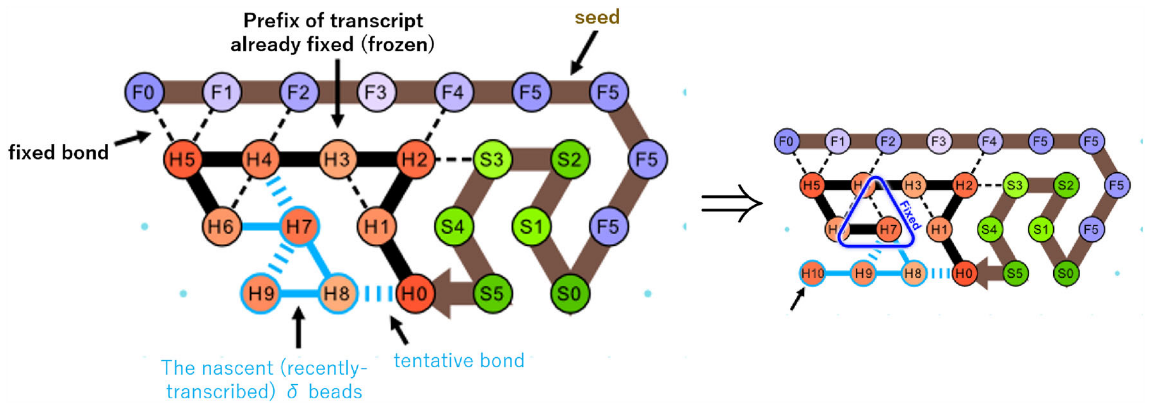

Fig. 3 A conformation of a bead-type sequence F0−···−F4−F5−F5−F5−F5 −S0−···−S5−H0−···−H6 and its (3-)elongation by H7−H8−H9. Components in oritatami are coded in color and line style here as: a seed in brown, a transcript solid, bonds dashed, beads and bonds already fixed in black, and those not fixed yet in cyan

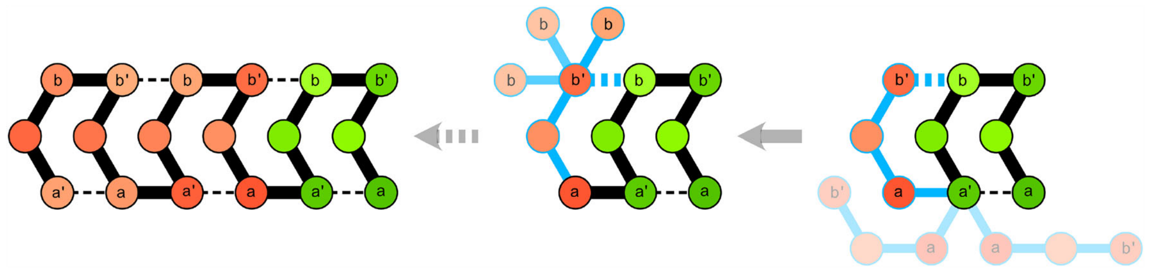

Fig. 4 Glider in oritatami. A transcript of period 6, `a•b'b•a'a···`, folds deterministically into a directed linear structure of height 3 and width 2 per period

in molecular biology and engineering). Its stability $H(c)$ is measured by the number of bonds in it, that is, $H(c) = |B|$. A *partial* conformation of c is a conformation of a prefix of u. An *elongation* of c by a finite bead-type sequence $v \in \Sigma^*$ is a conformation of uv. It is particularly called a k-*elongation* if $|v| = k$. The set of all elongations of c by v is denoted by $c^{\triangleright v}$. The conformation c is said to be of *arity at most* α if every bead of c forms at most α bonds. By C_α, we denote the set of all conformations of arity at most α.

Remark 1 As mentioned above, even a larger molecule than a nucleotide can be represented by a bead unless it violates the order of transcription by, for instance, being scattered over a transcript. A more complex rule than the hydrogen bond rule of the RNA origami, which meaningfully excludes any pair of nucleotides but the Watson-Crick complementarity (`A`-`U`, `C`-`G`) and the pair `G`-`U`, has been thus justified. Needless to say, a rule should be simple for the wetlab implementation. It might be interesting to implement a Turing-universal oritatami system, for example, with a complementary (i.e., bijective) rule (Open Problem 2-1)?

This freedom in level of abstraction makes it impossible as of this moment to tell which value the delay δ should be set to. The value 3 has been chosen in many works (Fazekas et al. 2022; Geary et al. 2018; Han et al. 2021; Maruyama and Seki 2021; Masuda et al. 2018; Pchelina et al. 2022), but this is simply because the *glider* motif, which shall be introduced shortly in Example 1, was discovered at this delay and found quite useful during early days of oritatami research in order not only to sustain computations but also to serve as a foundation of computational modules such as a half-adder as shown in Fig. 12.

2.1 (Oblivious) dynamics in oritatami

Two dynamics have been studied in oritatami so far: *inertial* and *oblivious*. Among them, here we introduce only the prevailing oblivious dynamics. Indeed, the first binary counter (Geary et al. 2016) is the only system implemented so far on the inertial dynamics. As for the inertial dynamics, we just point out how different it is from the oblivious one.

An oritatami system $\mathcal{O} = (w, \heartsuit, \delta, \alpha)$ is given as input an initial (seed) conformation $\sigma \in C_\alpha$ (arity at most α) of a finite bead-type sequence $s \in \Sigma^*$ and folds its transcript w upon σ co-transcriptionally as a sequence of partial conformations of sw as $\sigma = c_0, c_1, c_2, \ldots$, where c_i is a (1-)elongation of c_{i-1} by the i-th bead $w[i]$ such that

$$c_i \in \mathop{\arg\max}_{\gamma \in c_{i-1}^{\triangleright w[i]} \cap C_\alpha} \left(\max_{\eta \in \gamma^{\triangleright w[i+1..i+\delta-1]} \cap C_\alpha} H(\eta) \right). \tag{1}$$

This means that at the i-th step, w has been transcribed up to its $(i + \delta - 1)$-th bead, all but the most nascent δ beads, $w[i], w[i+1], \cdots, w[i+\delta-1]$, have already been fixed as c_{i-1}, and according to those with the largest number of bonds among all the elongations η of c_{i-1} by $w[i]w[i+1..i+\delta-1]$ that are of arity at most α, $w[i]$ and the bonds it forms are fixed. Note that conformations of arity $\alpha + 1$ or larger are excluded in (1). Hence, any conformation with a bead that binds to α or more beads cannot be reached. Bead binding is thus on the first-come-first-served basis; once a bead has formed α bonds *permanently*, that is, in some reachable conformation c_i, it cannot bind any more even if the rule set $\heartsuit$ allows it. If α is set to the degree of the underlying grid graph, beads never use up their capability of binding. In that case, we say that the arity is inexhaustible and denoted as $\alpha = \infty$ in case for model modification over other grid graphs. Note that the simulator (Schabanel 2016) presumes inexhaustible arity.

Example 1 Glider. (Geary et al. 2018) Directional and self-supportive structures and patterns have been called *gliders* in the research of computational models such as cellular automata. In oritatami, a glider was implemented for the first time in order to underpin the simulation of a cyclic tag system at delay 3. Its transcript is of period 6 as `a•b'b•a'a···` and its rule is to let `a` bind with `a'` and `b` bind with `b'`; that is, • is inert.

As long as the previous period folds as shown in Fig. 4 (right), a period recursively folds alike. The nascent fragment `a•b'`, colored in cyan, folds into all the possible conformations, three of which are illustrated in the figure, and finds the most stable ones. Note that this fragment cannot form more than one new bond because there is no `a'` around to which the nascent `a` can bind (note that no bead can bind to its predecessor) and there is only one `b` around to which the nascent `b'` can adjoin. Accordingly, the nascent `a` is stabilized, as shown in Fig. 4 (center). This temporary decision to locate the nascent `b'` to the left of the `b` and bind them will not be overridden until when the `b'` is finally stabilized because the two beads transcribed by then, that is, `b` and •, cannot form any new bond.

The glider motif by itself operates at an arbitrary arity $\alpha \geq 1$. Note that any bead of it admits within its range at most one bead to which it can actually bind; a glider is provided with `a`'s and at any arity strictly larger than 1 these beads remain capable of binding even after being stabilized, but all of them but the latest one are hindered sterically from attracting a nascent `a'`.

If there is i such that the set in the right-hand-side of (1) is composed of two or more elongations, then this system folds *non-deterministically* on σ; otherwise, the folding is *deterministic* on σ. All the oritatami systems published except those studied in (Han et al. 2018) are deterministic (Open Problem 2-2) but only in the sense that they remain deterministic on any supposed seed; in an extreme case, any system immediately gets non-deterministic on a seed made of a single bead. At i when the RHS of (1) gets empty, the system *halts*.

This dynamics is oblivious in the sense that after the bead $w[i]$ is stabilized according to (1), the system entirely forgets how the nascent fragment $w[i..i+\delta-1]$ has folded/bonded and at time $i+1$, it considers afresh all the possible elongations of c_i by $w[i+1..i+\delta]$. The inertial dynamics rather remembers how $w[i..i+\delta-1]$ has been folded/bonded so as to maximize the number of bonds, and at time $i+1$, it considers only their elongations by $w[i+\delta]$. These dynamics hence coincide at delay 1.

2.2 Oritatami classes

The class of deterministic oritatami systems that has been studied the most is that of periodic (cyclic) systems (Fazekas et al. 2022; Geary et al. 2016, 2018, 2019; Maruyama and Seki 2021; Masuda et al. 2018; Pchelina et al. 2022, 2020). This class is motivated by the possible synthesis of a periodic transcript from a cyclic DNA template (Geary and Andersen 2014). It has been proved Turing-universal at delay 2 and 3 as summarized in Table 1 (see also Open Problems 2-3, 2-4, and 2-5; see, e.g, (Fazekas et al. 2021) for some elementary results on computationally-weak subclasses of delay-1 oritatami systems).

Another class of interest is that of complementary oritatami systems. An oritatami system is *complementary* if its rule set $\heartsuit$ is so, that is, for all $a, b, c \in \Sigma$, $a \heartsuit b$ and $a \heartsuit c$ imply $b = c$. It was considered in the context of transcript design (Han et al. 2020), which asks to complete an oritatami system all of whose components but transcript are given along with a seed and a directed self-avoiding path by designing a transcript with which the system folds upon the seed deterministically along the given path. This problem was proved NP-hard even if the given system is promised to be complementary with just three binding pairs $a \heartsuit \overline{a}$, $b \heartsuit \overline{b}$, and $c \heartsuit \overline{c}$ with no other inert bead type involved (Han et al. 2020) (Open Problem 2–6).

The class of oritatami systems with exhaustible arity has been hardly studied though it seems more intuitive for a bead to use up its capability of binding and a bead to be transcribed later cannot help but break into a bond already formed in order to bind to the bead as the DNA strand displacement. This kind of local reconfiguration has been recently incorporated in oritatami (Marcus et al. 2023) but in the ordinary oritatami model, binding is on a first-come-first-served basis. Diminishing capabilities of binding have not been utilized for any computation in oritatami yet. For instance, at arity 1, gliders can propagate an extra one bit beyond and independently of their known capability to propagate one bit as a position of their first and last beads by replacing their inert beads with a self-attractive bead. See the six unlabeled beads in Fig. 4 (left). Suppose they are all of this self-attractive bead type; then they admit two binding patterns as ○ ○ — ○ ○— ○ ○ and ○— ○ ○— ○ ○—○; they are certainly capable of propagating 1-bit depending on whether the leftmost bead is still capable of binding or not.

Open problems

2-1 Can we simulate some Turing-universal model of computation by an oritatami system with a complementary rule? (Page 4)
2-2 Does nondeterminism help us save bead types, and if so, how? (Page 5)
2-3 For $\delta \geq 4$, prove that the class of delay-δ periodic deterministic oritatami systems is Turing universal. (Page 5)
2-4 Prove that the class of delay-1 deterministic oritatami systems is not Turing-universal. (Page 5)
2-5 What is the minimum number of bead types necessary for Turing-universal computation in oritatami? (Page 5)
2-6 Can we solve the transcript design problem (Han et al. 2020) efficiently when only two binding pairs are involved? (Page 5)

3 Turedo: a high-level programming language for oritatami

A Turedo (Turing + *Teredo navalis*) is a finite-state machine with its head crawling on the 2-dimensional hexagonal lattice

Table 1 A summary of Turing universal oritatami systems. The Period column involves variables W, Q, r, and A. W is the sum of the lengths of all productions (words) in a simulated skipping cyclic tag system. By Q and r, we denote the number of states in the 1D cellular automaton and the radius, respectively, and A is a tape alphabet of a simulated Turedo

	To be simulated	δ	α	$\lvert\Sigma\rvert$	Period
Geary et al. (2018)	Skipping cyclic tag system	3	∞	542	$O(W^4)$
Pchelina et al. (2020)	1D-CA (Q states, radius r)	2	∞	183	$O\left(rQ^{4r+2}\log Q\right)$
Pchelina et al. (2022)	(Head-up stateless) Turedo of radius 1	3	∞	1735	$O(\lvert A\rvert^6 \log \lvert A\rvert)$

$\mathbb{Z}^2$, or equivalently but more intuitively, on the honeycomb grid each of whose hex cell is centered at a lattice point, *without crossing its trail* Pchelina et al. (2022). Its *tape configuration* is an element of $A^{\mathbb{Z}^2}$, where A is a finite tape alphabet including the blank symbol $\perp$. Suppose the underlying lattice $\mathbb{Z}^2$ is equipped with a set $D = \{\vec{\mathtt{N}}, \vec{\mathtt{NE}}, \vec{\mathtt{SE}}, \vec{\mathtt{S}}, \vec{\mathtt{SW}}, \vec{\mathtt{NW}}\}$ of six unit vectors. We denote by $B(r)$ the ball of radius r centered at $(0, 0)$. A radius-r Turedo can see only the cells within the distance r. That is, even in a tape configuration $c \in A^{\mathbb{Z}^2}$, at a position $p \in \mathbb{Z}^2$, it can only observe for each $u \in B(r)$ what $c(p+u)$ is (the symbol written at the position $p + u$). This observable ball can be represented by the restriction of c to the ball of radius r centered at p, that is, $c_p(r) = (u \in B(r) \mapsto c(p+u))$. It then refers to its lookup table δ by using $c_p(r)$ as a key in order to decide which state it changes to, with which of the non-blank symbols it marks the current cell, and which of the unvisited neighbor cells to visit next, which have been initialized by the blank symbol $\perp$. Turedos are thus self-avoiding; they can visit only where a blank symbol $\perp$ is and they rewrite the symbol into a non-blank one (in $A \setminus \{\perp\}$) before leaving.

Definition 1 **((North-up) Turedo** Pchelina et al. 2022**)** A *Turedo* is a tuple $\mathcal{T} = (A, Q, q_0, r, \delta)$, where A is a tape alphabet including the blank symbol $\perp$, Q is a finite set of head states including the initial state q_0, $r \geq 0$ is a lookup radius, and $\delta : Q \times A^{B(r)} \to Q \times D \times (A\backslash\{\perp\})$ is a lookup table used as a local transition function.

A *global state* is an element of $A^{\mathbb{Z}^2} \times \mathbb{Z}^2 \times Q$ (tape configuration, head position, and state of the head). In a global state (c, p, q) with $\delta(q, c_p(r)) = (q', d, a)$, only if $c(p+d) = \perp$, the Turedo can transition to a global state $(c', p+d, q')$, where c' is defined by $c'(p) = a$ and $c'(u) = c(u)$ for all $u \neq p$; otherwise, it is *blocked*.

Turedos can be initialized by an arbitrary global state, even with more than one connected component of cells labeled by a non-blank symbol (Nalin and Theyssier 2022), though as a programming language of oritatami, it is supposed to start with all the cells being blank but the one where the head is placed initially, or at least with only one connected component of non-blank cells, and that is what the Turedo-to-oritatami compiler, which we shall introduce in Sect. 3.1, expects.

Turedos are "north-up" by Definition 1. However, unless they forget where they have come from immediately after a transition is made or they are blind in the sense of $r = 0$, no compass is needed. Indeed, a north-up Turedo can be simulated intrinsically by a so-called "track-up" Turedo with the same radius, which reads its neighbors in some specific order with respect to the direction of travel such as reading the six neighbors at $r = 1$ counter-clockwise starting from the diagonally-backward right, the diagonally-forward right next, and so on until the backward, where the previous cell is. In this specific order, the previous cell is read last, and hence, can let the current cell know by its label in which of the possible six directions in $D = \{\vec{\mathtt{N}}, \vec{\mathtt{NE}}, \vec{\mathtt{SE}}, \vec{\mathtt{S}}, \vec{\mathtt{SW}}, \vec{\mathtt{NW}}\}$ the system is heading, and even which state the system is in.

Definition 2 **((Stateless) Track-up Turedo with radius 1)** Let us define the $\pi/3$-degree (CCW) rotation $\rho_{\pi/3} : D \to D$ as $\rho_{\pi/3}(\vec{\mathtt{N}}) = \vec{\mathtt{NW}}$, $\rho_{\pi/3}(\vec{\mathtt{NW}}) = \vec{\mathtt{SW}}$, $\rho_{\pi/3}(\vec{\mathtt{SW}}) = \vec{\mathtt{S}}$, $\rho_{\pi/3}(\vec{\mathtt{S}}) = \vec{\mathtt{SE}}$, $\rho_{\pi/3}(\vec{\mathtt{SE}}) = \vec{\mathtt{NE}}$, and $\rho_{\pi/3}(\vec{\mathtt{NE}}) = \vec{\mathtt{N}}$.

A stateless radius-1 track-up Turedo is a pair $\mathcal{T} = (A, \delta)$, where A is defined as in the ordinary Turedo, while δ is a lookup table $\delta : A \times A \times A \times A \times A \times A \to \{0, 1, 2, 3, 4, 5\} \times (A\backslash\{\perp\})$. Being stateless, its *global state* is just a pair of a tape configuration $c \in A^{\mathbb{Z}^2}$ and a head position $p \in \mathbb{Z}^2$. Suppose $\mathcal{T}$ is in a global state (c_t, p_t) and its head was at a position p_{t-1} in the previous step. For $1 \leq k \leq 6$, let $a_k = c_t(p_t + \rho^k_{\pi/3}(\overrightarrow{p_t p_{t-1}}))$, that is, a_1, a_2, a_3, a_4, a_5 are possibly-blank letters written in the respective diagonally-backward right, diagonally-forward right, forward, diagonally-forward left, diagonally-backward left, and backward neighbor cells of p_i with respect to the direction of travel (from p_{t-1} to p_t); that is, $a_6 = c_t(p_{t-1})$. Then according to the rule $(k, a) = \delta(a_1, a_2, a_3, a_4, a_5, a_6)$, the Turedo transitions to the global state (c_{t+1}, p_{t+1}) unless it is blocked (defined as done in Definition 1), where $c_{t+1}(p_t) = a$ and $c_{t+1}(q) = c_t(q)$ for all $q \in \mathbb{Z}^2\backslash\{p\}$, $p_{t+1} = p_t + \rho^k_{\pi/3}(\overrightarrow{p_t p_{t-1}})$.

The stateless track-up Turedo is similar in dynamics to oritatami. An input Turedo to the Turedo-to-oritatami compiler is expected to be stateless, track-up, and with radius 1 (though, technically speaking, being stateless is dispensable as noted above). The compiled oritatami system folds each period of its transcript into a macrocell that simulates one transition of the input Turedo. The macrocell reads the states of the neighboring macrocells, which are written in binary, weighs the bits read, and sums them up in a uniquely decodable manner. It is highly non-trivial in oritatami to implement a mechanism to take the weighted-sum of integral inputs but in Turedo the mechanism can be programmed in a straightforward manner in the lookup table. Hence, here we present an example track-up Turedo as if it would rather explicitly take a weighted-sum of bits read from the adjacent cells as a table reference key in order to give an idea of how the oritatami macrocell works.

Example 2 **(Sierpinski triangle by a Turedo)** Self-assembly of self-similar fractals has been intensively investigated in molecular programming. Based on the abstract tile assembly model (aTAM), Rothemund, Papadakis, and Winfree programmed "white and black" DNA tiles that assemble into a discrete Sierpinski triangle as a pattern in-vitro (Rothemund et al. 2004). Lathrop, Lutz, and Summers proved that it is impossible to assemble this fractal more *strictly* as a shape, that is, on condition that wherever colered in white in the pattern is rather not to be visited. They then demonstrated an aTAM system that assembles rather a *fibered* Sierpinski triangle, which has the same fractal dimension as the standard one (Lathrop et al. 2009).

Let us introduce a track-up, stateless, radius-1 Turedo that assembles a discrete Sierpinski triangle on the hex grid. This Turedo employs the alphabet $A = \{(0, 0, 0), (1, 0, 0), (1, 0, 1), (1, 1, 0)\}$, where $(0, 0, 0)$ is reserved as the blank symbol $\perp$, whereas among the other three non-blank symbols, the two with the first two bits being $(1, 0)$ should be interpreted as black while the other one should be interpreted as white; hence, hereafter, we rather denote this alphabet as $A = \{\perp, (\bullet, 0), (\bullet, 1), (\circ, 0)\}$. It starts from a global state with all the cells labeled with the blank symbol except the one at the origin (the top cell in Fig. 5 (right)), which is labeled with $(\bullet, 0)$. Its head sweeps the region bounded by the $\overrightarrow{\texttt{SW}}$- and $\overrightarrow{\texttt{SE}}$-axes in a zigzag manner. During zigs and zags, it colors the current cell by $(\circ, 0)$ or by $(\bullet, 0)$, depeding on whether the two cells above are the same in color or not. Along the border, which is colored in black, it makes a U-turn downward.

Let us explain its technicalities. As noted above, this Turedo labels cells with a triple of 3 bits $c_1, c_0, s \in \{0, 1\}$, where the first 2-bits (c_1, c_0) represent the color of the cell as $\circ = (1, 1)$, $\bullet = (1, 0)$, and $(0, 0)$ being uncolored, and hence, we call them *color bits*, while the LSB s is set to 1 as a signal to start a U-turn along the right border of the triangle; we call it a *state bit*. Its head reads the 3-bit labels of the surrounding 6 cells, weighs them according to the weights shown in Fig. 5 (Left), and sums them up to a key to refer to its lookup table $\delta : \{0, 1, 2, 3, 4, 5, 7, 8\} \to \{1, 2, 3, 4, 5\} \times \{(\bullet, 0), (\bullet, 1), (\circ, 0)\}$. If (c_1, c_0, s) is read from diagonally-backward left (northeast in this figure) or forward right (SW), which is weighed by $(1, 1, 0)$, it contributes to the key by $1 \times c_1 + 1 \times c_0 + 0 \times s = c_1 + c_0$, while if it is read from diagonally-forward left (SE) or backward right (NW), which is weighed by $(3, 3, 0)$, its contribution is by $3 \times c_1 + 3 \times c_0 + 0 \times s = 3(c_1 + c_0)$; in these cases, the state bit is thus ignored. If it is read from the back (north), it is weighted-summed as $0 \times c_1 + 0 \times c_0 + 2 \times s = 2s$; in this case, the color bits are thus ignored. Therefore, each cell reads only the state bit from its predecessor and only color bits from the others. Being labeled with $(0, 0, 0)$, no blank neighbor contributes to this key. Since this Turedo is not expected to collide head-on with a visited cell, the corresponding weight (south) is set to $(0, 0, 0)$. All the environments that this Turedo can encounter are illustrated in Fig. 6 and the corresponding transitions are:

$$\begin{aligned}
\delta(0) &= (5, (\bullet, 0)) \\
\delta(1) &= (2, (\bullet, 1)) \\
\delta(2) &= (1, (\bullet, 0)) \\
\delta(3) &= (3, (\bullet, 0)) \\
\delta(4), \delta(8) &= (3, (\circ, 0)) \\
\delta(5), \delta(7) &= (3, (\bullet, 0))
\end{aligned} \tag{2}$$

During the left and right U-turns, the Turedo encounters exactly the same environment, where nothing but the previous cell is at distance 1; it is the state bit that enables the system to distinguish these two kinds of U-turns by having set the state bit of the previous cell to 1 for turns along the right border (the bottom-leftmost case in Fig. 6).

3.1 Turedo-to-oritatami compiler (Pchelina et al. 2022)

The compiler developed by Pchelina et al. (2022) translates a stateless track-up Turedo with radius 1 over an alphabet A to a deterministic oritatami system at delay 3 and of period $O(|A|^6 \log |A|)$. The compiled oritatami system intrinsically simulates the Turedo, that is, behaves as the Turedo does modulo scaling, by simulating each Turedo hex cell by a hexagonal "macrocell" of side length $O(|A|^3 \log |A|)$. The macrocell encodes in binary a symbol in A written on a simulated Turedo cell along its macrosides; $\underbrace{00 \cdots 0}_{\lceil \log |A| \rceil}$ is reserved for the blank symbol $\perp$. Each period of the transcript folds into a macrocell through the following five phases (see Fig. 7):

1. Scaffold layer
2. Read layer
3. Write layer
4. Speedbump
5. Exit layer.

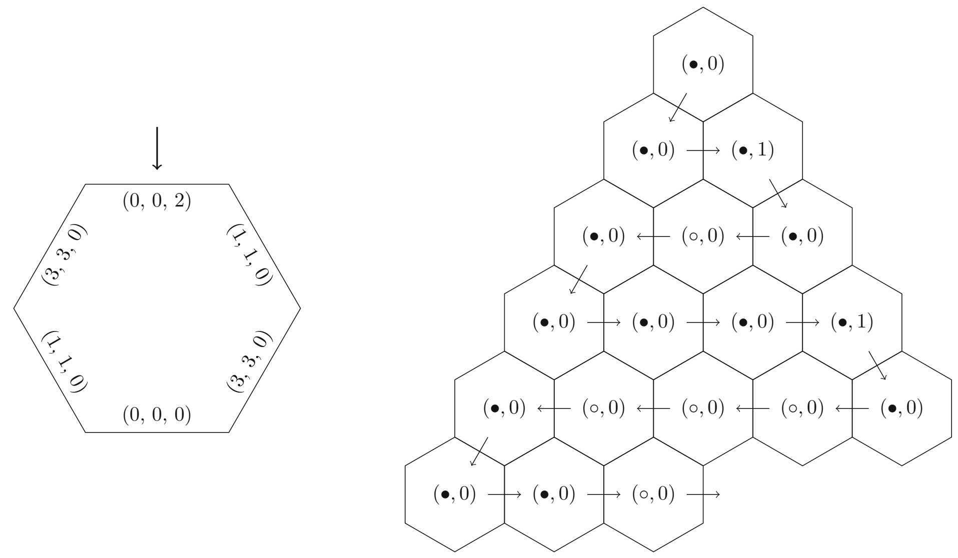

Fig. 5 The Sierpinski triangle self-assembles from the track-up Turedo with radius 1 in Example 2. (Left) Weights that the Turedo uses to compute an integral key from the 3-bit labels (c_1, c_0, s) of the possibly-empty 6 neighboring cells to refer to its own lookup table, which is in Fig. 6. (Right) The first 18 transitions of this Turedo, starting from the initial configuration all of whose cells but the one at the top in this figure are blank

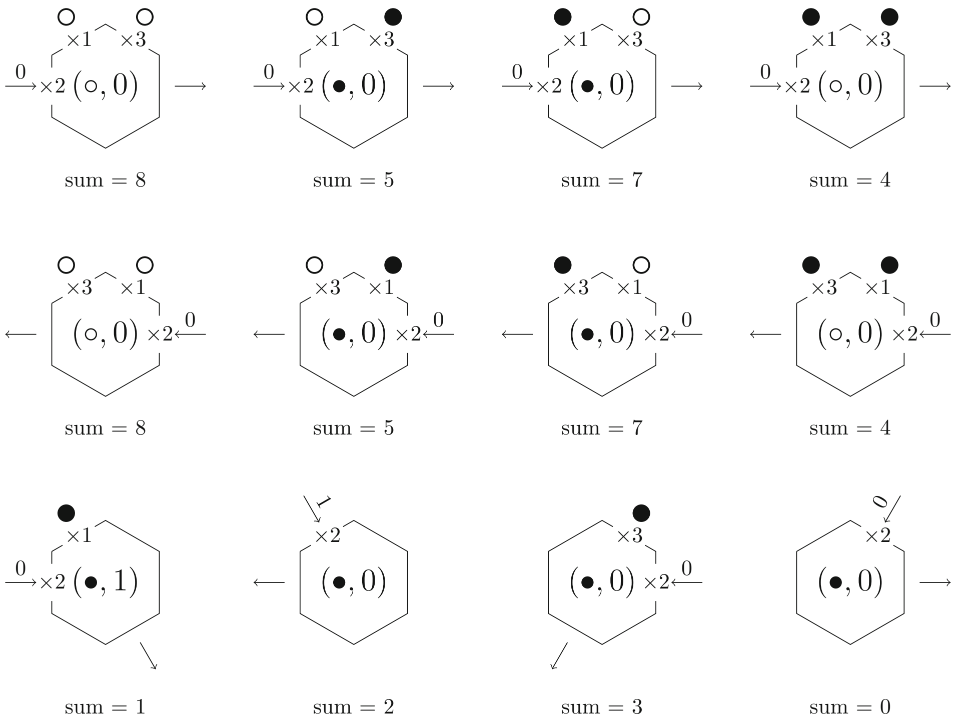

Fig. 6 All the environments that the Sierpinski-triangle Turedo can encounter and the corresponding transitions; each of those in the second row is nothing but the horizontal reflection of the one above, though. Any neighbor with nothing written is supposed not to have been visited yet, that is, to be labeled with (0, 0, 0). The sum noted below each environment is computed by weighing integral labels of the adjacent cells (⊥ as 0, • as 1, and ∘ as 2) and used as a key to refer to the transition function (2)

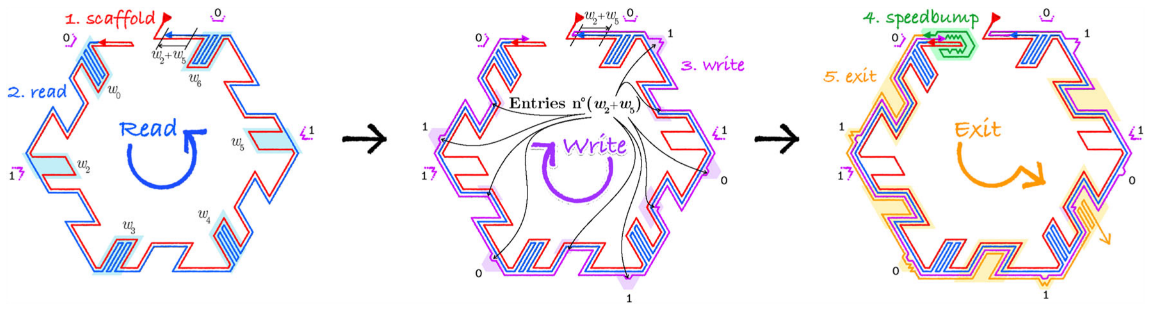

Fig. 7 Folding of a macrocell in the oritatami system translated by the compiler (Pchelina et al. 2022) from a radius-1 Turedo

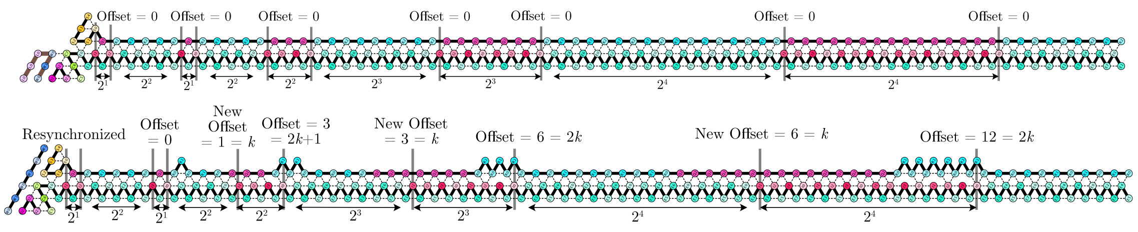

Fig. 8 Straight speedbump developed to simulate a 1D cellular automaton in oritatami at delay 2 (Pchelina et al. 2020). The "car" transcript folds from right to left over the smooth (blue) and bumpy (pink) ground. (Top) With no offset, the car stretches entirely straight, with blue over blue and red over red. (Bottom) Being pushed leftward by an offset (12 in this case), the car hits the brake with the offset divided by half whenever its blue sequence goes over the bumpy ground

Throughout the explanation, the previous macrocell is assumed to be the north of the current one, and the (macro)sides of the current one are indexed as 0, 1, 2, 3, 4, and 5 counter-clockwise (CCW), starting from the northwest; that is, the side 5 points to where the system has come.

The Scaffold layer folds clockwise (CW) from the north to the northwest into a macroscopically-hexagonal core of the macrocell in a hardcoded manner and underpins succeeding computations. The succeeding three layers (Read, Write, and Exit) wrap around this core one over another CCW, CW, and CCW, respectively. The scaffolds of adjacent cells must be distanced even at the nearest by 6 lattice points to accommodate these layers as SRWE|EWRS. Macrocells are actually lumpy, and they get as close as 6 points between only for bit I/O; that is, a bit is written on the top of a hilly structure. By the way, the Scaffold layer was designed by using a library of structural motifs with a unified interface called *scaffold builder* (see Sect. 4.3).

The Read layer folds CCW from the northwest to the north, and then the Write layer folds back CW to the northwest. The Read layer reads a symbol in A written in binary along the side of each neighboring macrocell, if any, or $\lceil \log |A| \rceil$ 0's otherwise, from the LSB as an input $(\mathtt{w}_{\mathtt{NW}}, \mathtt{w}_{\mathtt{SW}}, \mathtt{w}_{\mathtt{S}}, \mathtt{w}_{\mathtt{SE}}, \mathtt{w}_{\mathtt{NE}}, \mathtt{w}_{\mathtt{N}})$ for $\mathtt{w}_{\mathtt{NW}}, \mathtt{w}_{\mathtt{SW}}, \mathtt{w}_{\mathtt{S}}, \mathtt{w}_{\mathtt{SE}}, \mathtt{w}_{\mathtt{NE}}, \mathtt{w}_{\mathtt{N}} \in \{0,1\}^{\lceil \log |A| \rceil}$ CCW starting from the northwestern neighbor, and takes their weighted sum by simply concatenating them into $\mathtt{w}_{\mathtt{N}}\mathtt{w}_{\mathtt{NE}}\mathtt{w}_{\mathtt{SE}}\mathtt{w}_{\mathtt{S}}\mathtt{w}_{\mathtt{SW}}\mathtt{w}_{\mathtt{NW}}$ as done in (Pchelina et al. 2022) or in another ad-hoc manner like the one in Example 2. The resulting sum *offsets* the succeeding transcript, that is, pushes it forward. Lookup tables, which are hardcoded along the Write layer as a *folding meter* (Sect. 3.1.1), thus shift and expose their entries referred by the input at the points where they are supposed to be read later by another macrocell, expressing the symbol written and specifying from which side to exit. After the offset is absorbed entirely through the speedbump, which we shall explain next, the Exit layer faithfully folds along the Write layer until the exiting signal where the rest of the period for this macrocell is absorbed before exiting.

The offset is no longer in use after shifting the lookup tables, that is, by the end of the Write layer. The speedbump module was developed in Pchelina et al. (2020) for the offset-based I/O in oritatami in order to cancel out all the expected offsets. It consists of a "ground" and "car" (see Fig. 8). First, the ground folds from left to right, resulting in an alternation of flat (blue) and bumpy (pink) grounds, getting doubled in length at every alternation. The "car," which folds later from right to left, is colored alike so that with no offset, its color pattern matches completely the color pattern on the ground, leaving no bump as shown in the top of Fig. 8. With a non-zero offset, the car is pushed forward (leftward), forcing a blue fragment to drive over a bumpy ground (pink) and halving the offset. For example, Fig. 8 (bottom) depicts how the offset 12 decreases to 0 after driving over a logarithmic number of bumpy surfaces as 12, 6, 3, 1, down to 0.

The longest bumpy ground on the speedbump is indeed as long as the maximum offset expected. Since a macrocell is supposed to read at most 6 symbols of A in binary, with a straight speedbump, its side would be as long as $O\left(\left(2^{\log \lceil |A| \rceil}\right)^6\right) = O(|A|^6)$. With a bendable speedbump, the offset can be absorbed much more compactly. A zigzag speedbump has been developed in Pchelina et al. (2022) to downsize the macrocell to the one half power; in fact, its side is of length $O(|A|^3 \cdot \log |A|)$, where $\log |A|$ derives from that number of lookup tables responsible for each output bit per side.

Before introducing the *folding meter*, let us raise one open problem (Open Problem 3-1). The proposed compiler is implemented at delay 3, but could be done in principle even at delay 2. Recall how far two adjacent macrocells are. The boundary between them can be represented by the capitals of their layers as SRWE|EWRS; that is to say, when one macrocell is to read the other, the boundary is SR··|EWRS. Note that at delay δ, even a bead at distance $\delta + 1$ can be "sensed" by letting the most recently transcribed bead, that is, the one at the tip of the nascent fragment bind to the bead, unless being hindered geometrically. All of its other functional modules seem modifiable so as to work at delay 2; for instance, the straight speedbump (Pchelina et al. 2020) was implemented at delay 2.

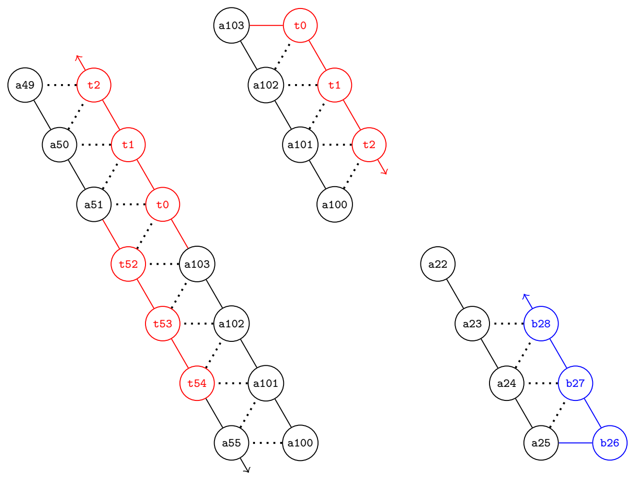

Fig. 9 Basic binding patterns of a folding meter

3.1.1 Folding meter

The compact macrocell must store $6\lceil \log |A| \rceil + 5$ lookup tables of $|A|^6$ entries (one table per output bit or exit-decision bit), somehow inside and expose only the called entries outside. *Folding meter* (Pchelina et al. 2022) was developed for this purpose, but it constituted all the computational mechanisms except the speedbump in the end.

A folding meter consists of a sequence of period 104 as

—t0—t1—t2—a3— · · · —a25—b26—b27—b28—a29— · · · —a51 →
→ —t52—t53—t54—a55— · · · —a77—b78—b79—b80—a81— · · · —a103 →

The period is partitioned into the quarters (of length 26) at `t0`, `b26`, `t52`, and `b78`, respectively. Let us call them *storage units*, and more particularly, the first and third `t`-*type* and the second and fourth `b`-*type*. A rule is designed in such a way that

1. the folding meter traces any surface as long as every nascent fragment (of length $\delta = 3$) can form 6 bonds, as shown in Fig. 9 (left);
2. `t`-type and `b`-type storage units are capable of folding back to their previous storage unit being initiated by their first three beads (`t0-t1-t2`, `t52-t53-t54`, `b26-b27-b28`, `b78-b79-b80`) forming 5 and 4 bonds, respectively, as illustrated in Fig. 9 (center, right); and
3. they never fold back anywhere else.

These basic properties enable a folding meter to be stored compactly in a box with u-turns at the top by t-type storage units while those at the bottom by b-type ones, as exemplified in Fig. 10, where a folding meter of the Write layer folds into switchbacks inside a *write pocket*.

The write pocket consists of a folding meter with $|A|^6$ storage units on the Write layer and a box of size $O(|A|^3) \times O(|A|^3)$ that has been founded by the Scaffold layer and traced by the Read layer. The Read layer is in fact made entirely of a folding meter, and hence, of period 104. For ease in implementation, the folding meters of the Read and Write layers share no bead type with each other, as their bead types are distinguished by their initials `W` and `R` (see Fig. 10). The Read layer has shifted dependently on an input and pushed the Write layer forward, but the offset is guaranteed to be a multiple of 52 by the bit-reading mechanism. This means that no matter what was input, every bead of the Write folding meter faces the same bead types of the Read folding meter, modulo 52. The binding rule can be hence orthogonalized between these folding meters and spares a considerable amount of room to program functions that can be called by de-synchronizing these meters.

De-synchronization and re-synchronization between the Write and Read folding meters enable a macrocell to store lookup tables compactly in write pockets and to write a 1-bit geometrically at the top of a mountainous bit I/O site. See Fig. 10. An appendix at the bottom of the left wall of the box traps a portion of the Read layer and shall cause a de-synchronization when the Write layer would have been tracing the Read layer along the wall down to that point. As long as the nascent fragment at the de-synchronization is either —b26—b27—b28 or —b78—b79—b80, it stably folds back upward, thus initiating the first switchback. Once back to the top of the box, the folding meter is already far enough from the Read layer to be attracted, and hence, it folds afterward into as many switchbacks as the width of the box until it gets re-synchronized with the Read layer on the opposite side of the box. A box of size $O(|A|^3) \times O(|A|^3)$ thus accommodates the whole lookup table for one bit.

Geometrical bit writing is also programmed outside the diagonal. A bit I/O site consists of a mountain and de-synchrozing and re-synchrozing holes respectively at its left and right bases, sandwiched by two write pockets. The lookup table for this bit is accommodated in these pockets, slides between them according to an offset like a movie film on the classical two-reel projector (though inside a box it folds into switchbacks rather than being reeled), and exposes the table entry called by the offset at the summit. Bits must be expressed *geometrically*. This is because a side is not always covered by the Exit layer. Recall that the Exit layer is designed so as to fold CCW from the northwest after an offset being absorbed through the speedbump, and hence, given an input on which the simulated Turedo is to exit southward, for instance, neither the southeast nor northeast side is covered, but for another input, they may be. No matter whether being covered by the Exit layer or not, bits must be readable by a hardcoded mechanism (it should be uncomputable in Turedo how the facing cell has transitioned). The following three conformations are programmed in each of the $|A|^6$ storage units of this table: two bumps (1) at the summit, (2) on

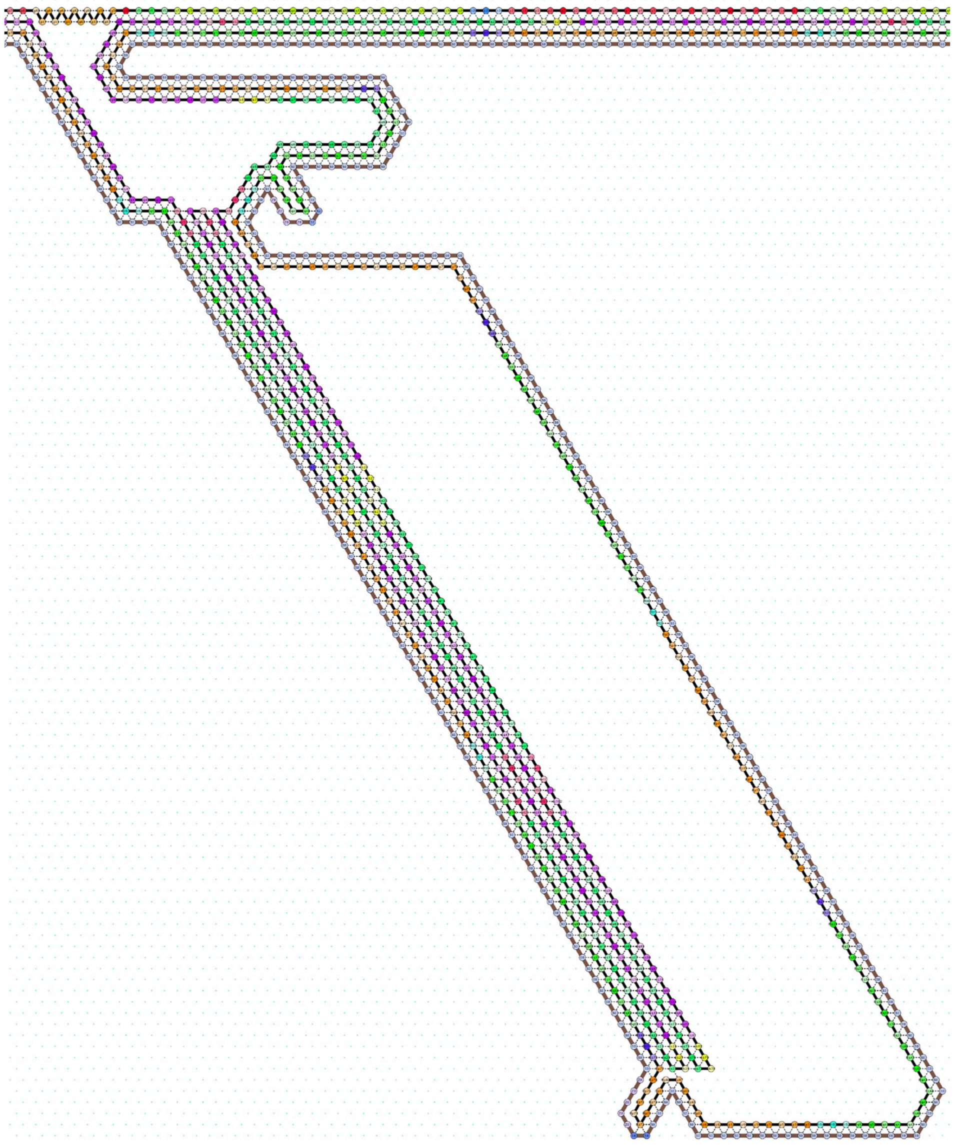

Fig. 10 A write pocket. The Scaffold, Read, Write, and Exit layers overlap one after another outside the pocket. A write folding meter folds from left to right, traces the Read layer as long as they are synchronized, that is, up to the appendix at the bottom of the pocket's left wall, which traps a small portion of the Read layer for de-synchronization, and then folds into switchbacks

the right base, or (3) on the left base. The bumps in the first conformation directly attract a bit-reading probe, which we shall explain in the next paragraph, in case of the bit not being covered by the Exit layer, while those in the second conformation slide the Exit layer so as to expose specific beads on the summit that attract the probe. These thus amount to writing 1 while the third one is to write 0. Note that we know in advance in which way each 1 entry should be programmed.

Folding meters are utilized not only to write a bit but also to read it. A *read pocket* is positioned in face of the summit of the mountain where a bit is written. The read pocket is almost a carbon copy of the write pocket; inside a box made by the Scaffold layer, the Read layer is to fold into switchbacks *unconditionally*. Both its first half period (from `t0` to `a51`) and second one are endowed with the capability to fold into a glider-like conformation by being attracted to the beads that encode 1 in order to jump over a read pocket. Without such beads, that is, when either 0 is written or no macrocell is adjacent, the Read layer rather falls into the write pocket and folds into as many switchbacks as the width of the pocket. The difference in length between these two folding pathways amounts to an offset, which is proportional to the pocket capacity.

Last but not the least, it is the folding meter that enables the macrocell to exit from an arbitrary side. The Exit layer must be at least 4 times as long as a side of the macrocell in order to wrap around the macrocell from the northwest corner to the northeast side. Therefore, when exiting earlier, the remaining portion of the Exit layer must be accommodated somehow along the side (or somewhere earlier, though this option was not chosen). The Exit layer is hence implemented as a folding meter as a whole. Every side of the macrocell but the north one is provided with an *exit pocket*, which consists of a box and a lookup table on the Write layer like the write pocket. The box is provided with an outer corner at the bottom of its right wall, which we call an *exiting corner*, and the lookup table is to inform the Exit layer of which side to exit from. The lookup table slides along the contour of the box and positions its entry called around the exiting corner. The Exit layer, which traces the contour of the box, is peeled off from the layer below and initiates switchbacks, not by being desynchronized as done in the write pocket but by a bead of specific type that attracts the fragments −626−627−628 and −678−679−680 moderately to let them fold back but only at an outer corner. The lookup table is programmed by modifying each storage unit by replacing its bead that can be positioned at the exiting corner by a bead of this mildly-attractive type if on the corresponding input, the simulated Turedo is supposed to exit from this side. Being thus peeled off, the remaining whole portion of the current period folds into switchbacks inside the box and then exits from the side to start the next macrocell. Otherwise, it keeps tracing the contour and go for the next side.

Open problems

3-1 Can we modify the compiler so that the translated oritatami system operates rather at delay 2? (Page 9)

3-2 Expand the Turedo-to-oritatami compiler for Turedos with longer radii than 1, or prove that it is impossible to intrinsically simulate such a Turedo in oritatami.

4 Single-stranded architectures for computing

The Turedo-to-oritatami compiler outlined in Sect. 3 has drastically facilitated oritatami programming as exemplified by the Sierpinski triangle assembler (Example 2) and a clockwise walker, which is a track-up Turedo that faithfully follows the right “wall” of visited cells (Pchelina et al. 2022). In particular, clockwise walkers are a promising candidate for the first wetlab implementation of a computational system driven by RNA co-transcriptional folding because as long as going around something convex, only two kinds of move are needed: going straight and diagonally turning forward right. However, being thus compiled from a Turedo, the resulting oritatami system comes nowhere near the wetlab realization as it operates only at delay 3 and requires maximum (inexhaustible) arity and as many as 1735 bead types according to the current implementation of the compiler as of August 2023. The wetlab realization of computations driven by RNA co-transcriptional folding requires such a compiler to spare as many bead types as possible, or otherwise encourages us to drop the idea of using high-level programming languages and rather to program an oritatami system directly. Two oritatami clockwise walkers, shown in Fig. 11, were hence programmed from scratch. Practical compilers highly probably require novel architectures for computing.

In this section, we overview single-stranded architectures for computing in oritatami implemented so far with emphasis on a modular approach in programming and modules involved. Nucleic acid structures have been successfully built up by modularly combining small motifs (see, e.g., Jaeger and Chworos 2006). A modular approach in oritatami programming is to factorize a transcript semantically so as for each factor (infix) to play relatively simple roles such as

- processing a few bits of information (logic gate, half-adder, etc.),
- information propagation (wire),
- guiding the transcript physically towards a specific direction (90-degrees turn, U-turn, etc.), and
- providing a scaffold for succeeding computations.

One module could be rather scattered along the transcript, but such a deconcentrated module would severely limit the freedom in the design of a directed path along which the transcript folds (*folding pathway*) and potentially entail undesirable long distance interaction.

Modules can be either hardcoded or computational. Turing universal computations in oritatami (Geary et al. 2018; Pchelina et al. 2022) are sustained by a scaffold composed of structural modules each of which hardcodes a unique shape like a line and a turn (for details, see Sect. 4.3); cf. the fact that the first Turing universal oritatami system (Geary et al. 2018) is free from such a hardcoded scaffold as all of its modules serve as a scaffold for the succeeding computation. In contrast to structural modules, computational modules fold differently in different environments. Half-adders, the proof-of-concept oritatami module, fold macroscopically into one shape but microscopically into one of the 4 possible (intramodular) pathways depending on the values of their 2-bits input as shown in Fig. 12.

4.1 Algorithmic design of modules in oritatami

Modular programming in oritatami is encouraged also from the algorithmic viewpoint by a particular affinity of the incremental feature of co-transcriptional folding with dynamic programming. A problem to design a deterministic oritatami system has been formulated as follows:

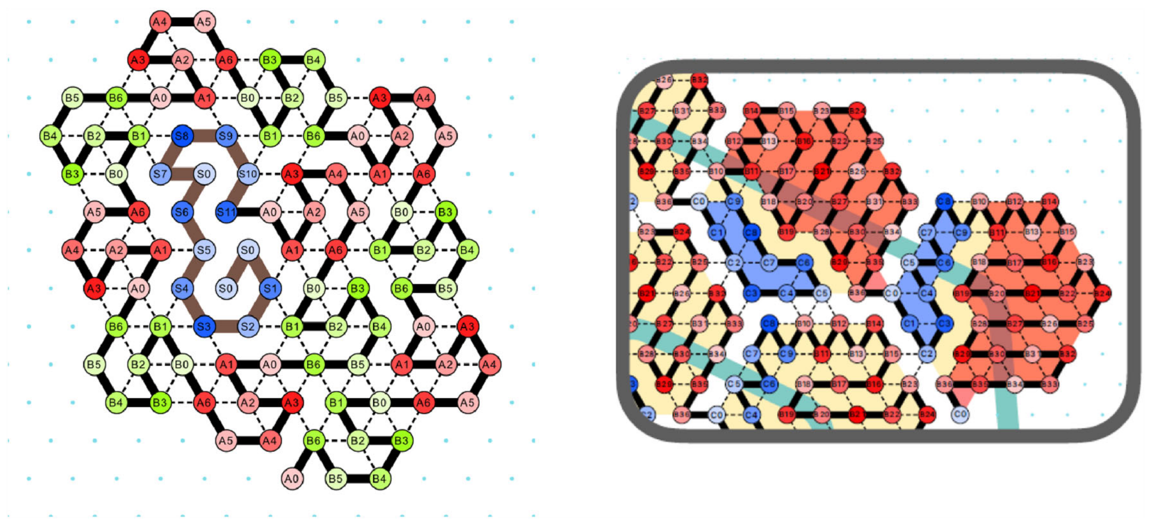

Fig. 11 Oritatami clockwise walkers (left) at delay 2 and of period 14 and (right) at delay 3 and of period 37. They merely check whether there is a wall diagonally forward right or not. For instance, a period of the system on the right consists of a right-wall sensor (colored in blue, of length 10), which admits two conformations, and a hardcoded structural motif (colored in red, of length 27)

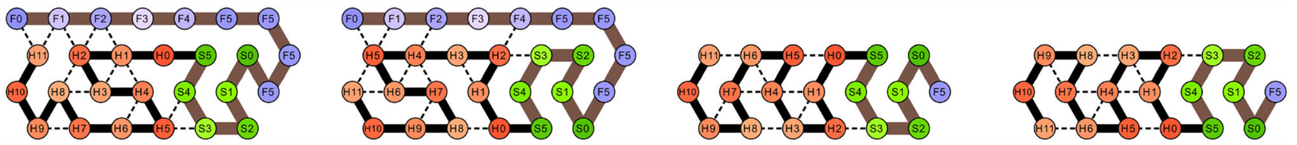

Fig. 12 The latest design of a half-adder in oritatami, whose transcript is of length 12, colored in red (Fazekas et al. 2022). Unlike the previous half-adders (Geary et al. 2019; Maruyama and Seki 2021; Masuda et al. 2018), the self-supportive glider motif (3rd and 4th) enables the 1-bit input from above to be encoded as whether a specific sequence of bead types is above (0) or nothing is there (1). Carry is fed as done conventionally as the height of the first bead (of type `H0`), top for not being carried (1st and 3rd) and bottom for being carried (2nd and 4th)

Problem 1 (Module Design Problem (Geary et al. 2016, 2019))
Given

- length $n \geq 1$ of the transcript of an oritatami system to be designed,
- two disjoint sets B and $\{1, 2, \ldots, n\}$ of bead types,
- delay $\delta \geq 1$,
- arity $\alpha \geq 1$, and
- a set of k pairs $\{(\sigma_1, P_1), (\sigma_2, P_2), \ldots, (\sigma_k, P_k)\}$ of a seed over B and an extension of its directed path by n lattice points in a non-self-crossing manner (folding pathway),

output a rule with which an oritatami system at delay δ with arity α folds the transcript $w = ① - ② - \cdots - ⓝ$ deterministically upon the seed σ_i along P_i for every $1 \leq i \leq k$.

Geary et al. (2016, 2019) have proposed an algorithm to solve Problem 1 with the promise that arity α be maximum (see Open Problem 4-1). It runs in time $O(n \cdot 5^{\delta} 2^{3k(\delta^3+\delta^2+4\delta+1)})$ and in space $O(n \cdot \delta^2 2^{3k(\delta^3+\delta^2+4\delta+1)})$. These complexities motivate us to build up a system from a small (in n) and monofunctional (small k) components, matching perfectly with the spirit of modular programming as long as a sufficient number of bead types are available. Note that the algorithm supposes the inertial dynamics, which has not been examined after its proposal in Geary et al. (2016), Geary et al. (2019). It should be modified so as to function in the oblivious dynamics (Open problem 4-2).

This algorithm requires the transcript to be programmable in the sense that its bead types are pairwise-distinct and never appear on a seed. In fact, if a transcript is specified rather as input, the algorithm cannot run even polynomially in n unless P = NP. The resulting rule set design problem is known to be NP-hard in n even for $k = 1$ except in the following two cases (1) delay is 1 and (2) delay is 2 and arity is 1 (Ota and Seki 2017). This problem is proved there to remain as hard even if configurations are rather given, which tell not only how the transcript should fold but also how its beads should bind. This is due to the so-called *phantom* bond, which forms only intermittently in order to stabilize a bead but does not remain in the product. The NP-hardness for $k = 1$ means that it is not trivial even to hardcode something which involves more beads than the bead types available. As arbitrarily-large macrocells are indispensable for the Turing-universal simulations in oritatami in Pchelina et al. (2022, 2020), their scaffolds have been modularized. The scaffold builder (Pchelina et al. 2022) provides a library of structural motifs with a unified interface (see Sect. 4.3).

With the wetlab experiments in mind, the assumed pairwise-distinctness of bead types gets hardly justified even though beads can represent an oligonucleotide or even something larger. Then the module design problem should be formulated rather as: with a rule R rather given as input together with an alphabet Σ, delay δ, arity, a seed σ, and a target conformation C, how can we sequentially combine beads of a small number of types such that starting from σ, the resulting transcript deterministically folds into C. Han, Kim, and Seki have thus formulated the transcript design problem (TDP) and particularly investigated its sub-problem where a given rule is guaranteed to be complementary (Han et al. 2020). They proved that the complementary TDP (CTDP) is $\mathtt{NP}$-hard even if (1) $|R| = 2$ but delay δ is not bounded, or (2) $|\Sigma| = 6, \delta = 3$, and $|R| = 3$. The latter $\mathtt{NP}$-hardness was obtained by reducing an instance of the planar 3-coloring problem, which is known to be $\mathtt{NP}$-hard even on the square grid as a planar graph of n vertices can be embedded in a square grid graph of size $O(n^2)$ (Geary and Johnson 1979; Harel and Sardas 1998), into a glider-based zigzag folding pathway. It can be easily adapted to delays larger than 3 by making gliders in the folding pathway taller, but not to 2 or 1. This problem is known to be solvable in polynomial time only under the very strict condition that $|R| = 1, \delta = 1$, and α is either 1 or at least 4; an algorithm that solves this problem under this condition in linear time in the length of transcript is given in (Han et al. 2020) (Open Problem 4-3).

4.2 Oritatami design process

Oritatami is a lower-level programming language for co-transcriptional folding than Turedo. Programming in oritatami is not yet something that one can do just for fun from the comfort of one's living room. It has been hardly automated; in fact, the module-design algorithm (Sect. 4.1) is the only algorithm available so far. Oritatami systems are being designed fundamentally by hand. In order to solve a computational task, oritatami engineers must contemplate the following general problems, possibly along with task-specific technicalities:

1. (Modularization) What kinds of modules are necessary/sufficient to solve the task?
2. (Circuit layout) How should these modules be deployed on the reaction surface, that is, on the 2D plane in the current oritatami model?
3. (Folding pathway design) How should these modules be traversed?
4. (Module implementation) How should these modules be implemented?

These numbers do not suggest any sequential order of engineering but are merely for references. Oritatami design is rather a cycle of solving these problems repeatedly based on the feedback from previous trials while untangling the web of mutual dependence and conflicts among various design criteria like

Cr-1 A module should expect as few environments as possible for the sake of algorithmic module implementation (see Sect. 4.1);
Cr-2 Instances of each module should be deployed at regular intervals;
Cr-3 Two instances of the same module should not be adjacent.

The number, k, of environments in Cr-1 translates directly to the running time of the module design algorithm mentioned in Sect. 4.1 as its time complexity involves the factor $\left(2^{3(\delta^3+\delta^2+4\delta+1)}\right)^k$. It should hence be as small as the theoretical lower-bound, that is, the minimum number of inputs for this module to operate. Being deployed regularly, as concerned in Cr-2, modules can be programmed compactly along the transcript. As for Cr-3, it enables us to orthogonalize the so-called inter-modular rules from intra-modular ones. With two instances of A next to each other, modifying a rule *intra-modularly*, that is, to help an instance fold into a specific conformation, may prevent these instances from interacting as expected *inter-modularly*, and vice versa. In addition, it is not until this criterion is satisfied that the module design algorithm can be applied.

Once Cr-3 is met, it suffices to design a set of folding pathways for each module, with which the algorithm in Sect. 4.1 yields the module (or says these pathways are not compatible with each other in one module). These pathways must be distinct enough to let the succeeding computation know what the module has computed. This *intra-modular folding pathway* design has been done manually but became much less laborious nowadays thanks to the glider motif. The glider motif is self-standing but made of few enough bonds to remain plastic to serve as a default conformation of a functional module, as exemplified in the latest implementation of a half-adder (Fig. 12), two of whose four conformations are a glider and its upside-down. Their plasticity was utilized already in the first Turing-universal oritatami system (Geary et al. 2018). Bit-width expansion in an oritatami counter (Maruyama and Seki 2021) should illustrate better how powerful gliders are. No matter how large the seed to begin with, a zigzag counter eventually overflows, when it cannot rely on the pre-built "ceiling." All the modules of the counter

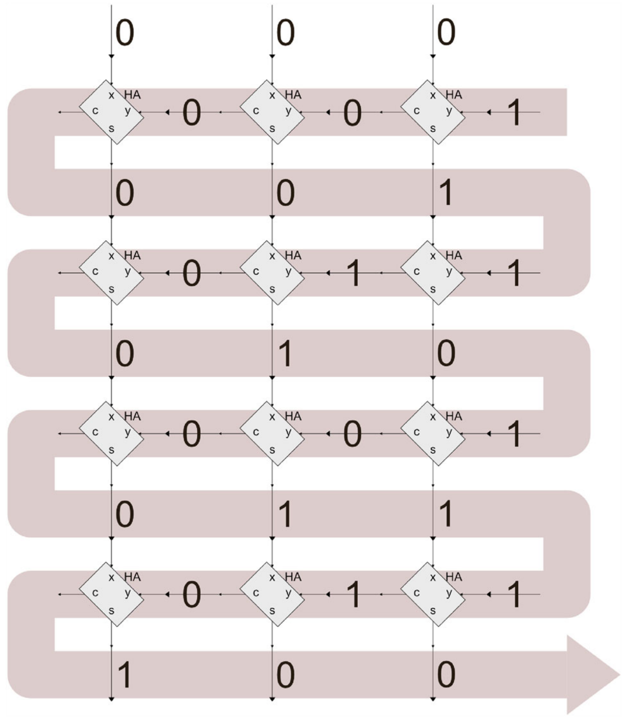

Fig. 13 Deployment of half-adders and wires for a circuit to count in binary and sequential "zigzag" assembly of the circuit in the aTAM (Rothemund and Winfree 2000) and in oritatami (Geary et al. 2016, 2019; Maruyama and Seki 2021). The half-adder takes binary inputs x and y and outputs $s = x \oplus y$ and $c = x \wedge y$ (carry)

proposed in Maruyama and Seki (2021) are glider-based and fold into a glider under the open sky. Thus, as illustrated in Fig. 14, one whole period (pink, purple, green, and orange) of the transcript folds into a glider at an overflow. The next period does not keep proceeding forward but rather turns back because a signal to have the pink module fold into the left-turn conformation, `F27-F26-F21`, gets exposed along the previous orange glider, which is usually hidden beneath the ceiling sterically; in Fig. 14, both of the instances are surrounded by a rectangle and magnified.

4.2.1 Case study: binary counter in oritatami

Let us see how the fixed bit-width binary counter has been designed as a proof-of-concept demonstration of the oritatami model (Geary et al. 2016). It is based on the zigzag binary counter in the aTAM (see, e.g., Rothemund and Winfree 2000), which is designed so as to self-assemble in a sequential manner, in spite of intrinsic parallelism to the aTAM. The aTAM counter increments by 1 in every zig ($\leftarrow$) and the incremented count is propagated across the succeeding zag ($\rightarrow$) to the next zig. It saved the designers of the oritatami counter from contemplating the first two problems, modularization and deployment, to a considerable extent, by suggesting to modularize the counter into half-adders (how else, though), and then how to deploy them and traverse them as shown in Fig. 13.

In programming an aTAM system, or more generally, in a model of self-assembly by molecular aggregation, it suffices at this stage of system design to implement DNA tiles that play a role of each module. Here we do not mean that with a circuit layout, the aTAM programming would be straightforward; it is in fact `NP`-hard to design a system in the aTAM that yields a given planar circuit layout out of at most given number of tile types even if the layout is composed of only two types of components (Kari et al. 2017). In the aTAM, however, at least it does not really matter, and actually should not matter, in what order tiles (modules) attach to a half-built circuit unless causality is violated. The order in which modules are synthesized does matter in programming co-transcriptional folding. Fortunately, the aTAM counter even suggests a folding pathway. Indeed, every reachable configuration of it admits at most one site at which a tile can attach; that is, the tile attachment proceeds as if tiles were "transcribed" sequentially one by one and the "transcript" of tiles folded into zigzags. (This is, of course, no more than an analogy; a series of tile attachments cannot be pre-programmed as an attaching tile at each site may vary in type depending on the initial count.)

The resulting layout, however, does not deploy the half-adders regularly enough in the sense that as long as the transcript is to fold as suggested and the module that increments the count by 1 in a zig is different from the one for propagation in a zag, its period cannot be shorter than the length of one zigzag, and such a transcript can handle only inputs of specific width in bits. Note that without being carried ($y = 0$), a half-adder merely outputs its input x as $s = x \oplus y$. By letting half-adders serve as a vertical wire in this way, the period can get as short as one half-adder and one wire.

"Pillars of half-adders," which result from this substitution, violate Cr-3 and prevent the module design algorithm from being applied. This problem has been solved by a technique called *doppelgänger*. In a nutshell, making a doppelgänger of a module A in an oritatami system means to "prime" each bead type occurring in the transcript of A as $a \to a'$ and to augment the original rule with its A-primed version, which is obtained from the rule by priming all the occurrences of bead types included in A. Needless to say, doppelgänger costs more bead types and longer period of the transcript.

The transcript of the fixed bit-width binary counter is of period 60 as ⓪−①−···−(59), which consists of a half-adder `H` = ⓪−①−···−⑪, its doppelgänger `D` = ㉚−㉛−···−㊶, and the other two factors `L` = ⑫−⑬−···−㉙ and `R` = ㊷−㊸−···−(59) called an L-turner and an R-turner, respectively. A period folds into a linear structure of width 20 and height 3 except at the end of a zig

Table 2 Binary counters in oritatami.

	δ	α	$\|\Sigma\|$	Period	Dynamics	At overflow
Geary et al. (2019, 2016)	4	∞	60	60	Inertial	undefined
Maruyama and Seki (2021)	3	∞	132	132	Oblivious	Bit expansion
Iwano (2023)	3	∞	894	1494	Oblivious	Undefined

The columns δ, α, and $|\Sigma|$ represent delay, arity, and alphabet size (the number of bead types used), respectively, where $\alpha = \infty$ means that in the corresponding system, a bead can form a bond no matter how many bonds it has already formed. Note that the most recent one (Iwano 2023) has not been optimized yet; hence the numbers in the corresponding row are highly unlikely minimum

and a zag, where an L-turner and an R-turner take a special conformation to guide the transcript to the next zig and zag, respectively. The half-adder folds into one of four possible conformations in a zig, depending on a 1-bit input x, which is given as to whether it folds below a specific sequence of bead types ($x = 0$) or not ($x = 1$), and a carry-in y, which is given as to whether it starts folding at the top ($y = 0$) or at the bottom ($y = 1$) of the zig, and outputs $s = x \oplus y$ as a sequence of four bead types exposed below while the carry-out $c = x \wedge y$ is 1 if and only if the last bead is stabilized at the bottom; see Fig. 12, which illustrates a more recent half-adder but serves our purpose here as this I/O principle had been inherited through the oritatami binary counters until the shift-based counter by Iwano (2023) (unpublished yet). L- and R-turners propagate a 1-bit carry from a half-adder to the next half-adder by folding into a glider of even length, which starts and ends along the same side, during zigs and zags (though in zags, carry is always 0). At the ends of a zig and a zag, they rather fold into another conformation for U-turn, respectively. An L-turner is informed of being at the end of a zig from the L-turner above, which has also folded into a U-turn conformation and exposed some signal below, and so is an R-turner from the R-turner above. In order to make sure that an L-turner is transcribed at the left end, the bit-width must be set to an odd integer as illustrated below, where 4.5 periods (`HLDRHLDRHLDRHLDRHL`) fold into one and half zigzag in a 3bit-wide oritatami binary counter:

```
↜ H R D L H ↲
L
↳ D R H L D ↴
            R
↜ H R D L H ↲
L
```

(See also Fig. 14, where `H` is colored in red, `L` in pink, `D` in blue, and `R` in green.) `H` and `D` modules function as a half-adder in a zig whereas they merely propagate a 1-bit downward in a zag as designed in Fig. 13. Note that one period of this counter can handle two bits. In some of the more recent counters in oritatami, which shall be introduced in the next paragraph, half-adders encounter a larger number of environments, and hence, `D`'s are replaced by a module dedicated for the vertical 1-bit propagation; as a result, their period can handle only 1-bit.

Counting in binary is the most well-studied computation in oritatami so far; the binary counter explained above has been modified or extended functionally, and served as a component of larger-scale oritatami systems (Fazekas et al. 2022; Iwano 2023; Maruyama and Seki 2021; Masuda et al. 2018). Hence, it might be worthy of introducing them briefly (see also Table 2).

Binary counters went through three principal functional expansions. Firstly, the counter was modified so as to work under the oblivious dynamics towards self-assembly of the approximated Heighway dragon fractal (Masuda et al. 2018). The resulting counter of width m in bits was combined with a 4-state deterministic finite automaton (DFA) into an oritatami system that folds into the first 2^m turns of the fractal (Open Problem 4-4).

It was then endowed with the capability of detecting an overflow in order to widen by 1-bit at every overflow for counting infinitely (Maruyama and Seki 2021) (Fig. 14). The detectability also enables counters to trigger a phase transition at an overflow. For example, an oritatami square self-assembler (Fazekas et al. 2022) involves two phases and thus transitions from Phase 1 to Phase 2. Given $n \geq 1$, it actually folds into the *rectangle* of size $(wn + c_w) \times (hn + c_h)$ for some scale factors w and h, independent of n, and some constants c_w, c_h (Open Problem 4-5). Upon a seed of length $O(\log n)$ on which n is encoded (in $\lceil \log n \rceil$-bits), it first measures one side of the rectangle by counting in the conventional zigzag manner until being overflowed, yielding a thin rectangle of size $O(\log n) \times (hn + c_h)$. At the overflow, it transitions into Phase 2, where it restarts counting from 0 perpendicularly to the long side of the thin rectangle, folding in the zigzag manner until the target rectangle is completed. In order to halt at the completion, it keeps tracking a diagonal from the start. Recall that the capacity of a zig to propagate a 1-bit of information is already used up to wire half-adders for carry. How can the square assembler propagate one more bit diagonally while counting?

A novel *elastic glider* motif (Fig. 15) is a solution. An elastic glider of length 12ℓ for $\ell \geq 1$ is a glider that is capa-

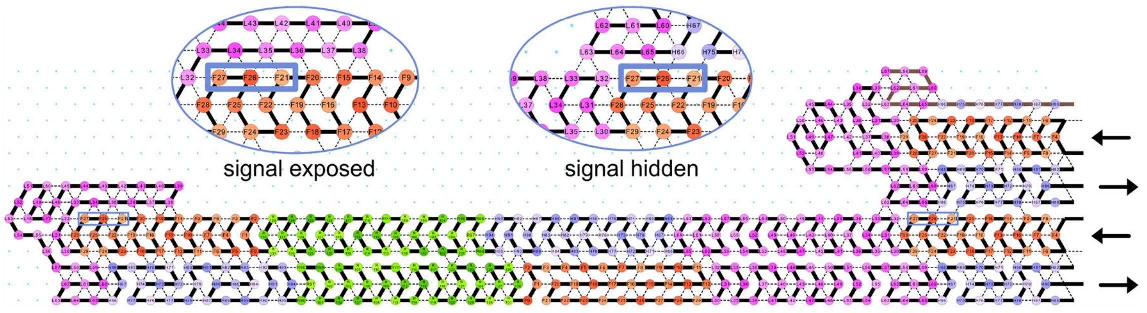

Fig. 14 Bit-expansion at an overflow in the binary counter proposed in Maruyama and Seki (2021). One period consists of a half-adder (colored in red), an L-turner (pink), a doppelgänger (blue), and an R-turner (green). The last half-adder at the end of the second zig (←) ends at the bottom, that is, with a carry. This means that the counter has overflowed. The ceiling is too far there for the next L-turner to make a U-turn (cf. the L-turner above). This L-turner takes its default glider conformation and so do the succeeding three modules `D`, `R`, and `H` of this period. The next left-turner thus begins at the bottom, but thanks to the absence of the zag above, it can now interact with the signal `F27−F26−F21` exposed along the glider conformation of `H` and folds into the alternative U-turn conformation

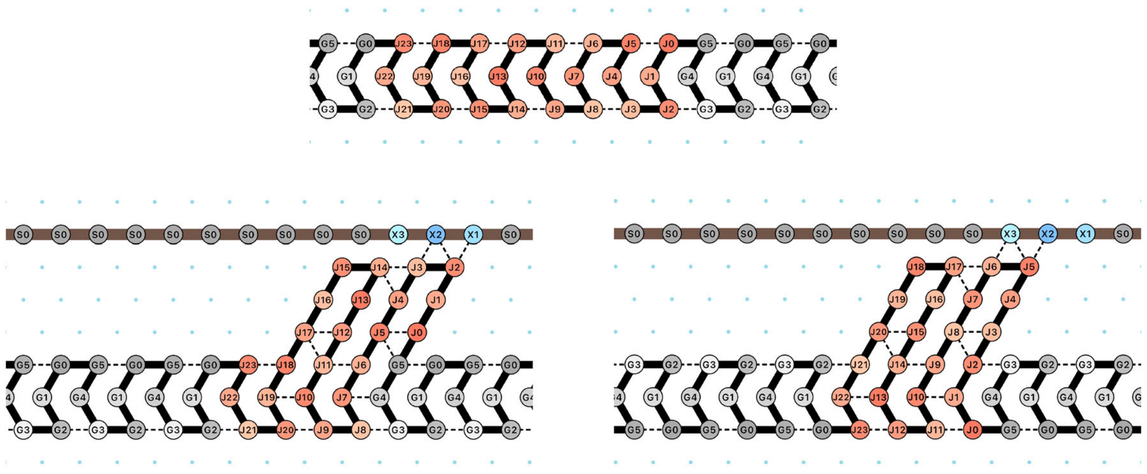

Fig. 15 Three conformations of an elastic glider: (Top) an ordinary glider conformation; (Bottom) shrunk conformations that start at top and at bottom, respectively, under a ceiling with beads of some specific types (here `X1`, `X2`, and `X3`)

ble of shrinking from its default ordinary "stretched" glider conformation of width 4ℓ and height 3 into another narrower-but-taller glider-like conformation of width $2w$ and height 6 (its instance of $\ell = 2$ is colored in red in Fig. 15) by being pulled upward by a sequence of three beads of special types. Note that the sequence can be as short as 3 no matter how large ℓ is, and is still capable of shrinking the whole elastic glider because the shrunk conformation of the unit ($\ell = 1$) elastic glider pulls the next unit, if any, upward instead as shown in Fig. 15. Furthermore, it serves the same purpose even if the elastic glider is upside down. This means that an elastic glider can shrink and propagate a 1-bit at the same time independently. The square self-assembler makes use of this capability. Its half-adder is composed of a tandem of two instances, say $H_{\rm d}$ and $H_{\rm nd}$, of an ordinary half-adder like the one in Fig. 12 with two elastic gliders sandwiching them. Its $H_{\rm d}$ and $H_{\rm nd}$ shift back and forth by making always one of the elastic gliders be shrunk while the other be stretched in order to ensure that its $H_{\rm d}$ is interlocked in the computation if and only if it is on the diagonal. It thus tracks the diagonal.

Elastic gliders can really be a game changer. They have certainly non-negligible potential for facilitating oritatami engineering to such an extent in a year or two that even novices become capable of programming large-scale systems after a few months of learning basics. Iwano, a bachelor student in our group, has indeed implemented a (fixed-bit width) counter that propagates a carry unconventionally by having its half-adder push the succeeding transcript forward (Iwano 2023) exactly when its carry-out is 1 (Open Problem 4-6). As shifts can be superposed on top of each other (as demonstrated in the shift-based Turing-universal simulations Pchelina et al. 2022, 2020), this shift-based counter, or more generally, shift-based computational modules would highly likely be embeddable as a component of another shift-based computation. Being implemented by merely combining elastic gliders, its half-adder was spared the laborious process of

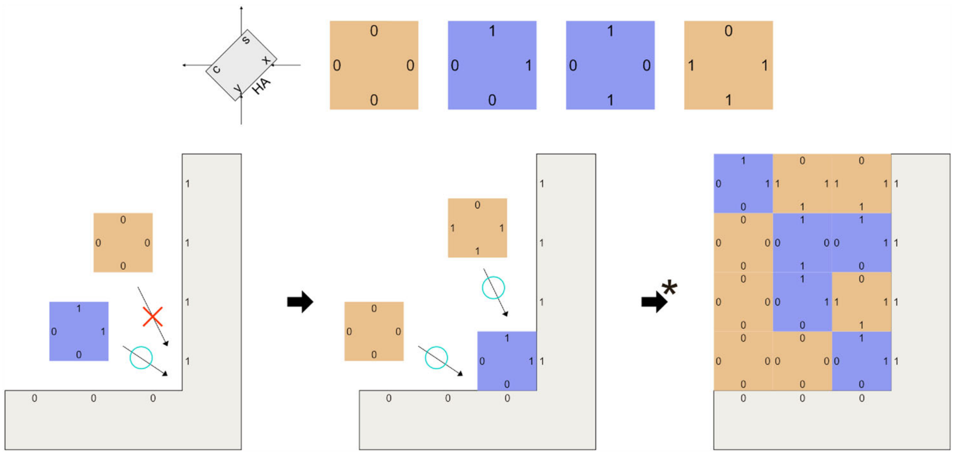

Fig. 16 The basic implementation of a half-adder in the aTAM at temperature 2. A tile is equipped with one strength-1 glue per side (0 or 1) and cannot rotate or flip. At the temperature 2, it can attach only if two or more of its glues match with neighboring glues (*cooperative tile attachment*)

designing from scratch a set of its intra-modular folding pathways like those in Fig. 12. To say the least, elastic gliders may soon eliminate the need for the module design algorithm or intra-modular folding pathway design.

4.2.2 Limit of recycling system designs from the aTAM

Research on the aTAM is a gold mine of well-thought designs as exemplified above. Events of tile attachment are so well-ordered according to those designs that a surprisingly small number of so-called tile-attachment pathways is actually possible, facilitating the theoretical analysis and verification.

Modularization in the aTAM might be however too "coarse." In the aTAM, a module with n-bit input can be, and usually is, implemented as a *set* of 2^n tile types, which encode respective 2^n possible inputs as glue types (half-adders are a typical example for $n = 2$, see Fig. 16). These constituent tiles compete for every position where this module is to be deployed and those with matching inputs (i.e., glues) win. Two such modules M_1 and M_2 can be even merged into one by taking a direct product of every pair of a tile type from M_1 and a tile type from M_2, letting multiple computations run on top of another in parallel.

In contrast, an oritatami module is just one fragment of a transcript. As the transcript is pre-programmed and cannot be modified at run-time, the fragment has to handle all the inputs expected, unless the system is designed so as to change its order of visiting modules, though more than one modular-level folding pathway has yet to be accommodated in any oritatami system. In order to handle n bits, a module has to be provided with a set of 2^n intra-modular folding pathways, which is however hardly accessible even for $n = 3$. A larger number of bits have been hence handled by larger-scale architectures such as binary counters (Geary et al. 2016, 2019; Iwano 2023; Maruyama and Seki 2021), finite automata (Han et al. 2021), multi-bit-width wires and bifurcater (Masuda et al. 2018), and simulators of Turing-universal models (Geary et al. 2018; Pchelina et al. 2022, 2020). It should be quite applicative to study systematically how to merge two simple modules in a compact manner. Simulating them simultaneously in a Turing-universal model and emulating it further on one of the general-purpose simulators is certainly a way but results in something humongous. Elastic gliders and additivity of shifts make shift-based computations a promising approach to this problem.

4.3 How to scaffold computations in oritatami?

All the functional modules of the zigzag oritatami counter serve also as a scaffold for the computation in the next zig or zag, as shown in Fig. 14. That is also the case in the simulation of skipping cyclic tag system (Geary et al. 2018), which self-assembles the space-time diagram of a simulated skipping cyclic tag system in a zigzag manner, and in the assembly of a finite portion of Heighway dragon fractal (Masuda et al. 2018).

Macro-cells to simulate the 1D cellular automaton (Pchelina et al. 2020) or Turedo (Pchelina et al. 2022) are rather sustained by a dedicated scaffold. For the Turedo simulation, a library of structural motifs has been developed called *scaffold builder*. It mainly consists of two types of line segment motif, two types of obtuse turn motifs, and two types of acute turn motifs, as shown in Fig. 17, which can be jointed arbitrarily as long as turns and line segments are alternated. As of now, it operates exclusively at delay 3 but should be easily modifiable for other delays. Its line segment motif is a repetition of `S0`−`S1`−···−`S7` or that of `S10`−`S11`−···−`S17`; they are a doppelgänger of each other. These two types are necessary for two line segments close to each other such as those sandwiching an acute turn. Note that one side of the resulting line segment is numbered oddly and the other evenly. The two types of obtuse turn are dedicated towards the "odd" side and "even" side of the preceding line segment, respectively, and so are those of acute one. Four "cushion-in"

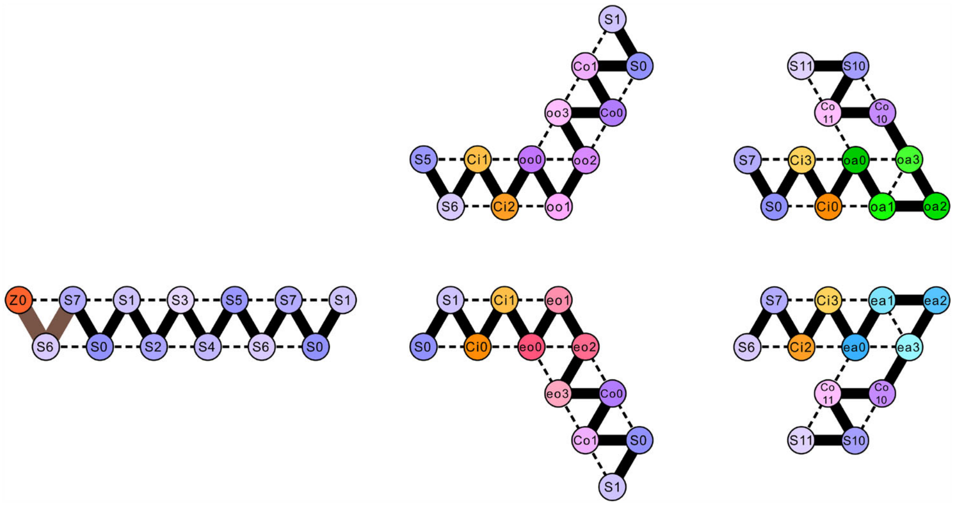

Fig. 17 Structural motifs used in the scaffold builder (Pchelina et al. 2022): line-segment, two types of obtuse turn `oo0-3` and `eo0-3`, respectively towards the odd and even sides of the preceding line segment, and their acute variants `oa0-3` and `ea0-3`

bead types `Ci0-3` let a line segment end arbitrarily to make a turn. The cushion-out is always `Co0-1` to start `S0-7` segment or `Co10-11` to start `S10-17` segment because they can be assumed to begin with `S0` and with `S10`, respectively. Even with this basic library, we can describe fairly complex scaffolds, though the Turedo-to-oritatami compiler (Pchelina et al. 2022) could not help but employ an extended library of more complex motifs rigged by using dedicated bead types.

4.4 Towards the wet-lab implementation

Folding pathway design in oritatami addresses two essentially different problems respectively at modular level and beneath: that is, to determine in what order modules should be traversed and to provide each module with a set of self-avoiding directed paths along which the transcript of the module is to fold. Neither of them has been studied systematically yet, though the second problem seems pretty inconsequential in oritatami programming after usefulness and applicability of elastic gliders have been witnessed in Fazekas et al. (2022), Iwano (2023). We should not, however, jump to this conclusion before observing what the oritatami model fails to capture.

Geary et al. (2021) have revealed experimentally how significantly the choice of co-transcriptional folding pathway affects the yield of RNA origami tiles. Any co-transcriptionally folded structure is underpinned primarily by double helices, which staple two factors of a transcript together. A double helix occurs at a factor $ux\theta(u)$ of a transcript by letting u and $\theta(u)$ base-pair in the nested manner (the k-th base of u bonded with the k-th *last* base of $\theta(u)$ for all $1 \leq k \leq |u|$). The longer u is, the more stable this double helix gets, whereas the longer x is, the less stable the helix gets. Hence, in order for this double helix to be stabilized even intermittently, x needs to be either short, when the helix is closed as a hairpin-loop, or stapled by extra base-pairs otherwise. In the latter case, the additional base-pairs easily confine u topologically and compel the RNA polymerase to twine all the way around u backward while synthesizing $\theta(u)$. Bead-binding in oritatami is neither hindered thus topologically nor favored; no matter how far they are along the transcript, beads can bind as long as they have not used up their capability of binding according to the arity (Open Problem 4-7).

The class of oritatami systems suitable for wet-lab implementation should respect the properties of RNA. A system should not waste any bead type or adopt an unnecessarily-complex rule. It may have to be modified so as to fit certain criteria. *Simulation* should play a central role in the system modification and provide us with a deeper understanding of computations by co-transcriptional folding, as done in DNA tile self-assembly (Doty et al. 2012; Hader et al. 2020).

4.5 Towards intrinsically universal oritatami systems

Any oritatami system can be simulated indirectly via a Turing machine using as few as 183 bead types at delay 2 or 542 beads at delay 3 in oritatami (see Table 1) but the simulation may not behave as the one simulated does. Turedos of radius δ can intrinsically simulate any deterministic oritatami system at delay $\delta - 1$. However, the current compiler (Pchelina et al. 2022) can handle only radius-1 Turedos, and delay being 0 makes no sense in oritatami.

Direct (intrinsic) simulation between oritatami systems has been studied only in the context of design optimization based on homomorphisms between conformations (modulo transcript) (Han and Kim 2019; Rogers and Seki 2017). Some specific delay(s) should be favored for certain wet-lab implementations. Nevertheless, it is not known whether a delay-δ system can be simulated intrinsically at delay $\delta + c$ even for any single value of δ or c (Open Problem 4-8), except impossibility under a quite restricted notion of simulation

(Rogers and Seki 2017). Being combined with the existing Turing universal oritatami systems at delay 2 (Pchelina et al. 2020) and 3 (Geary et al. 2018; Pchelina et al. 2022), such a simulation with $\delta \in \{2, 3\}$ proves that the class of oritatami systems at delay $\delta + c$ is as strong computationally.

A rule could be downsized and simplified by an algorithm without changing the behavior, delay, or arity of an oritatami system. Rule (set) minimization is proved to be NP-hard in general (Han and Kim 2019), but the oritatami system reduced from an instance of 1- IN- 3- SAT in the proof requires delay linearly proportional to the number of SAT variables. A certain kind of binding patterns is subject of prior removal from a rule over others even at the cost of bead types or more tractable binding patterns. *Self-attraction* $a \heartsuit a$ is such a kind. A naive approach to intrinsic simulation is to simulate each bead a of a simulated system by a designated *meta-bead*, which may fold into a hexagon-like shape and interact with other meta-beads as the corresponding simulated beads do. The self-attraction $a \heartsuit a$ implies that being adjacent, the simulating meta-beads interact somehow to simulate the bond $a - a$, but its intra-modular rules severely restrict the possible implementations of the interaction (violation of Cr-3). Self-attractions can be removed from any delay-δ deterministic *finite* oritatami system by using $3(\delta + 1)(\delta + 2)$ doppelgängers of each bead type (Han et al. 2019) (Open Problem 4-9). A nascent bead can observe at most $3(\delta + 1)(\delta + 2)$ beads (the number of lattice points inside the ball of radius $\delta + 1$) around, and hence, when the original system transcribes a bead a, the polymerase can find among it and its doppelgängers one that is not around and rather transcribe it; thus, the self-attraction becomes unnecessary. The quadratic cost was reduced further to linear and proved to be asymptotically optimal (Han et al. 2019).

With the use of meta-beads in mind, simulation requires the following functions:

- *Scaling function* $\tau : \mathbb{Z}^2 \to 2^{\mathbb{Z}^2}$, which maps a lattice point in the triangular lattice $\mathbb{T} = (\mathbb{Z}^2, \sim)$, on which the simulated oritatami system operates, to a region of the triangular lattice where a simulator works, without violating the adjacency relation $\sim$ among the lattice points. The region $\tau(p)$ is where a meta-bead is to be folded in order to simulate a bead at p.
- *Meta-bead function* $\mu : C_\alpha \times 2^{\mathbb{T}} \to \Sigma \cup \{\perp\}$, which masks a conformation of a simulator at arity α and tells which of the beads in Σ the masked (partial) conformation corresponds to, if any, or says no meta-bead has been formed there (yet).

The meta-bead function should be monotone in the sense that for a simulator's conformation c_1 and its elongation c_2, if $\mu(c_1, \tau(p)) = b$ for some bead type $b \in \Sigma$, then $\mu(c_2, \tau(p)) = b$, that is, once a meta-bead that represents b is folded in the region corresponding to a point $p \in \mathbb{T}$, then the region will not change to another bead type or get empty. This monotonicity enables to define the time step when each meta-bead is stabilized. A simulator is said to *simulate* a run of an oritatami system by its run in which

1. for any two points $p_1, p_2 \in \mathbb{T}$, if the simulated system stabilizes a bead at p_1 first and at p_2 later, then the simulator stabilizes a meta-bead at $\tau(p_1)$ first and then at $\tau(p_2)$ later, and
2. if a bead $b \in \Sigma$ is stabilized at a point $p \in \mathbb{T}$, then the simulator reaches in the run a conformation $c \in C_\alpha$ such that $\mu(c, \tau(p)) = b$.

How can a simulator be informed of which oritatami system to be simulated among those in a class? The simulator's (fixed) transcript cannot accommodate a universal table for "genomes" of all the oritatami systems unless the envisioned class is finite. Hence, an oritatami system cannot help but be input to the simulator as a seed. Such a genome encoded on a seed must be replicated at arbitrary distance, arbitrarily many times, highly probably one or more copies per meta-bead. Indeed, in order to observe a bead at delay δ, the transcript must enter a circle of radius $\delta + 1$ centered at the bead at the cost of at least one more bead stabilized permanently inside the circle. This means that, after being observed a quadratic number of times in δ, one copy of genome becomes unobservable. The genome replication is thus indispensable (Open Problem 4-10).

The scaling function τ cannot be independent of a simulated system because a meta-bead must be large enough to hold a copy of the genome, which encodes a rule of size $\Omega(|\Sigma|^2)$. Such a resizable meta-bead is to span more than one, and indeed a considerable number of, periods of the transcript (Open Problem 4-11). Images of this function, that is, regions that accommodate one meta-bead, should be not only as regular as possible but also scalable; the regular hexagon is certainly the primary choice. These regions can overlap and do not have to tessellate the lattice $\mathbb{T}$ as long as the adjacency relation is respected. The function τ itself should be simple enough to provide as little support as possible to the computation it underpins.

Beyond these macroscopically-static challenges lies a formidable one on oritatami dynamics. Local optimization (1) to stabilize a bead cannot be spared pieces of information that are not written on the genome but have been acquired *a posteriori*: what beads have been stabilized around and also how many bonds each of them has been bound with so far unless arity is inexhaustible. Recall the cost of observations in oritatami: permanent stabilization of a bead around a bead observed. Folding a nascent fragment of δ beads could be simulated by a nascent fragment of δ meta-beads at the delay of δ multiplied by the size of a meta-bead, that is, $\delta \times \max_{p \in \mathbb{Z}^2} |\tau(p)|$. This naive idea is however to be turned down immediately. Such a simulator would need to know during its design phase the delay and alphabet of an oritatami system to be simulated (note that even such a "weak" simulation is highly nontrivial: Open Problem 4-12). In addition, at such a large delay, even a single bead could not be stabilized in-silico (Schabanel 2016) within a realistic time frame as local optimization entails an exponential number of pathways to be checked in delay.

At a fixed small delay, meta-beads face no choice but to explore much farther than the delay around in order to be informed of which type of bead is at each observable neighboring point, and moreover, of whether or not it is still capable of binding unless arity is inexhaustible. Such exploration leaves a lot of beads stabilized permanently and easily blocks the transcript or confines it in a bounded region. An engineering challenge rises here of how to let trails of exploration cross each other as well as the simulated backbone of the transcript. This is highly probably an avatar of nemesis, who enacts retribution against our blasphemous trial to create intrinsically universal oritatami systems. The transcript cannot self-cross by definition. Trails of exploration thus amount to impregnable walls, no matter how slender they are. Exploration should be hence carried out meticulously only on as-needed-basis, and at the same time not too passively to keep necessary information within close reach. Meta-beads that are nascent according to the function μ cannot hang onto the shirttails of those already stabilized. They rather have to start assembling its dendrite because bead stabilization also takes into account bindings between nascent beads.

Without the idea of offset-based computing in oritatami and modules founded on it (Fazekas et al. 2022; Pchelina et al. 2022, 2020), some of which have been explained in this tutorial, we would lay our pen down on the desk at this point. They look however too promising not to be applied to engineer an intrinsically universal system at least for a certain non-trivial class in oritatami. Let us briefly share some ideas in conclusion. They may enable to equip meta-beads with a "dendrite." An arbitrarily large amount of information can be encoded as an offset and transported arbitrarily far without being impaired by deformation as tested along the bumpy outer shell of the Turedo macrocell (Pchelina et al. 2022). The offset control along the thin outer shell of the macrocell could be therefore adapted to implement a bendable axon with unlimited transmission capacity by, for example, letting the transcript be shifted at one end, carry the resulting offset down to the other end, and expose there the entry specified by the offset. The resulting axon could be as thin as several beads in width. It could propagate a piece of information even perpendicularly by bending into bumps and dents. Its terminal could be further equipped with the non-invasive bit-reading (Pchelina et al. 2022) to establish a non-blocking communication channel as narrow as delay, through which something thin enough like the transcript or another axon can pass later even twice or more times so that the transcript can avoid being trapped in a bounded region. Furthermore, dendrites could be even overlaid on top of each other, as suggested in the construction of uncomputable limit configuration in Turedo (Pchelina et al. 2022).

These types of junction are certainly not powerful enough to make all kinds of dendrite compatible with each other as well as the folding pathway of the transcript. In fact, it should be relatively easy to invent an example to outwit them. While diversifying such information transmission channels further in function and geometry, we should also tackle the problem of saving an oritatami system from exponential number of nascent folding pathways in delay to be searched in stabilizing a bead. Observe that the half-adder in Fig. 12 and its beads "know" beforehand that they are going to be stabilized at one of the four possible points, or more strongly, that the half-adder admits only four folding pathways as none of its beads prefers intermittently anywhere but the point at which it will be stabilized, that is, no phantom. Phantom is currently required to read a bit in a non-blocking manner (Pchelina et al. 2022), but could be dispensed with (Open Problem 4-13). Modular programming thus spares oritatami systems exponential-cost optimization and enables us to simulate them with simpler and more compatible dendrites. The resulting simulator should be far from an intrinsically universal oritatami system (Open Problem 4-14), but grappling with challenges mentioned above among others should enrich the field of programming computations by co-transcriptional folding further towards heritable in-vivo computation.

Open problems

4-1 Remove the promise of α being maximum from this algorithm. (Page 13)

4-2 Modify the algorithm by Geary et al. (2016, 2019) so as to solve Problem 1 in the oblivious dynamics. (Page 13)

4-3 How hard is the transcript design problem in any of unsettled cases such as being at delay 2? (Page 14)

4-4 Can we self-assemble not a prefix but the whole Heighway dragon fractal in oritatami? (Page 16)

4-5 Design an oritatami system that, given n, deterministically folds into the square of size $sn + c$ for some scaling factor s and constant c for all $n \geq 1$. (Page 16)

4-6 Endow the shift-based counter with the capability of widening by 1-bit at an overflow. (Page 17)

4-7 Can we modify the oritatami model so that bead binding takes topological barrier into account? (Page 19)

4-8 Can we simulate a delay-δ oritatami system by a delay-$(\delta + c)$ oritatami system for some δ and c? (Page 19)

4-9 Can we remove self-attraction from a deterministic *infinite* periodic oritatami system? (Page 20)
4-10 Design an oritatami system that allows to replicate a seed arbitrarily far. (Page 20)
4-11 Design an infinite periodic oritatami system that reads an integer ℓ from a seed and keeps folding hexagonal cells of side length ℓ one after another. (Page 20)
4-12 Given a delay δ and an alphabet Σ, can we construct a universal oritatami system that can simulate any deterministic oritatami system over Σ at delay δ? (Page 21)
4-13 Can we simulate in oritatami a radius-1 Turedo without relying on phantoms? Can we characterize so-called dispensable phantoms? (Page 21)
4-14 Can we construct for a certain (non-trivial) class of oritatami systems a universal oritatami system? (Page 21)

References

Alberts B, Johnson A, Lewis J, Morgan D, Raff M, Roberts K, Walter P (2014) Molecular biology of the cell, 6th edn. Garland Science

Arora S, Barak B (2009) Computational complexity: a modern approach. Cambridge University Press, Cambridge

Demaine ED, Hendricks J, Olsen M, Patitz MJ, Rogers TA, Schabanel N, Seki S, Thomas H (2018) Know when to fold 'em: self-assembly of shapes by folding in oritatami. In: Proceedings of the 24th international conference on DNA computing and molecular programming (DNA 24), volume 11145 of LNCS, Springer, pp 19–36

Diestel R (2010) Graph theory, 4th edn. Springer, Cham

Doty D, Lutz J H, Patitz M J, Schweller RT, Summers SM, Woods D (2012) The tile assembly model is intrinsically universal. In: Proceedings of the 53rd annual IEEE symposium on foundations of computer science (FOCS 2012), pp 302–310

Elliott D, Ladomery M (2016) Molecular biology of RNA. Oxford University Press, Oxford

Fazekas SZ, Kim H, Matsuoka R, Morita R, Seki S (2021) Linear bounds on the size of conformations in greedy deterministic oritatami. Int J Found Comput Sci 32(5):575–596

Fazekas SZ, Kim H, Matsuoka R, Seki S, Takeuchi H (2022) On algorithmic self-assembly of squares by co-transcriptional folding. In: Proceedings of the 33rd international symposium on algorithms and computation (ISAAC 2022), volume 248 of LIPIcs, pp 37:1–37:15

Feynman RP (1996) Feynman lectures on computation. Addison-Wesley, London

Geary C, Andersen E S (2014) Design principles for single-stranded RNA origami structures. In: Proceedings of the 20th international conference on DNA computing and molecular programming (DNA 20), volume 8727 of LNCS, pp 1–19. Springer

Geary C, Grossi G, McRae EKS, Rothemund PWK, Andersen ES (2021) RNA origami design tools enable cotranscriptional folding of kilobase-sized nanoscaffolds. Nat Chem 13:549–558

Geary C, Meunier P-É, Schabanel N, Seki S (2018) Proving the turing universality of oritatami co-transcriptional folding. In: Proceedings of the 29th international symposium on algorithms and computation (ISAAC 2018), volume 123 of LIPIcs, pp 23:1–23:13

Geary C, Meunier PÉ, Schabanel N, Seki S (2019) Oritatami: a computational model for molecular cotranscriptional folding. Int J Mol Sci 20(9):2259

Geary C, Meunier P-É, Schabanel N, Seki S (2016) Programming biomolecules that fold greedily during transcription. In: Proceedings of the 41st international symposium on mathematical foundations of computer science (MFCS 2016), volume 58 of LIPIcs, pp 43:1–43:14

Geary C, Rothemund PWK, Andersen ES (2014) A single-stranded architecture for cotranscriptional folding of RNA structures. Science 345(6198):799–804

Geary MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman & Co

Hader D, Koch A, Patitz MJ, Sharp M (2020) The impacts of dimensionality, diffusion, and directedness on intrinsic universality in the abstract tile assembly model. In: Proceedings of the 2020 ACM-SIAM symposium on discrete algorithms (SODA 2020), pp 2607–2624

Hagiya M, Arita M, Kiga D, Sakamoto K, Yokoyama S (1997) Towards parallel evaluation and learning of boolean μ-formulas with molecules. In: Proceedings of the DIMACS workshop on DNA based computers, volume 48 of DIMACS series in discrete mathematics and theoretical computer science, pp 57–72

Han Y-S, Kim H (2018) Construction of geometric structure by oritatami system. In: Proceedings of the 24th international conference on DNA computing and molecular programming (DNA 24), volume 11145 of LNCS, pp 173–188

Han Y-S, Kim H (2019) Ruleset optimization on isomorphic oritatami systems. Theoret Comput Sci 128–139

Han Y-S, Kim H (2021) Impossibility of strict assembly of infinite fractals by oritatami. Nat Comput 20(4):691–701

Han Y-S, Kim H, Masuda Y, Seki S (2021) A general architecture of oritatami systems for simulating arbitrary finite automata. Theoret Comput Sci 870:29–52

Han Y-S, Kim H, Ota M, Seki S (2018) Nondeterministic seedless oritatami systems and hardness of testing their equivalence. Nat Comput 17(1):67–79

Han Y-S, Kim H, Rogers TA, Seki S (2019) Self-attraction removal from oritatami systems. Int J Found Comput Sci 30(6–7):1047–1067

Han Y-S, Kim H, Seki S (2020) Transcript design problems of oritatami systems. Nat Comput 19(2):323–335

Harel D, Sardas M (1998) An algorithm for straight-line drawing of planar graphs. Algorithmica 20(2):119–135

Hopcroft JE, Motwani R, Ullman JD (2001) Introduction to automata theory, languages, and computation, 2nd edn. Addison Wesley, London

Iwano N (2023) Concurrent signal passing by co-transcriptional folding. Bachelor's thesis, The University of Electro-Communications. Tokyo, Japan

Jaeger L, Chworos A (2006) The architectonics of programmable RNA and DNA nanostructures. Curr Opin Struct Biol 16(4):531–543

Kari L, Kopecki S, Meunier PÉ, Patitz MJ, Seki S (2017) Binary pattern tile set synthesis is NP-hard. Algorithmica 78(1):1–46

Lathrop JI, Lutz JH, Summers SM (2009) Strict self-assembly of discrete Sierpinski triangles. Theoret Comput Sci 410:384–405

Marcus P, Schabanel N, Seki S (2023) Ok, a kinetic model for locally reconfigurable molecular systems. In: Visins of DNA nanotechnology at 40 for the next 40, pp 229–240. Springer

Maruyama K, Seki S (2021) Counting infinitely by oritatami co-transcriptional folding. Nat Comput 20(2):329–340

Masuda Y, Seki S , Ubukata Y (2018) Towards the algorithmic molecular self-assembly of fractals by cotranscriptional folding. In: Proceedings of the 23rd international conference on implementation and application of automata (CIAA 2018), volume 10977 of LNCS, pp 261–273

Merkhofer EC, Hu P, Johnson TL (2014) Introduction to cotranscriptional RNA folding. In: Methods in molecular biology, volume 1126, pp 83–96. Springer

Nalin S, Theyssier G (2022) On turedo hierarchies and intrinsic universality. In: Proceedings of the 28th international conference on

DNA computing and molecular programming (DNA 28), volume 238 of LIPIcs, pp 6:1–6:18

Ota M, Seki S (2017) Ruleset design problems for oritatami systems. Theoret Comput Sci 671:26–35

Pchelina D, Schabanel N, Seki S, Theyssier G (2022) Oritatami systems assemble shapes no less complex than tile assembly model (aTAM). In: Proceedings of the 39th international symposium on theoretical aspects of computer science (STACS 2022), volume 219 of LIPIcs, pp 51:1–51:23

Pchelina D, Schabanel N, Seki S, Ubukata Y (2020) Simple intrinsic simulation of cellular automata in oritatami molecular folding model. In: Proceedings of the 14th Latin American symposium on theoretical informatics (LATIN 2020), volume 12118 of LNCS, pp 425–436

Reif JH, Majumder U (2010) Isothermal reactivating whiplash PCR for locally programmable molecular computation. Nat Comput 9(1):183–206

Rogers TA, Seki S (2017) Oritatami system; a survey and the impossibility of simple simulation at small delays. Fund Inf 154(1–4):359–372

Rose JA, Komiya K, Yaegashi S, Hagiya M (2006) Displacement whiplash PCR: optimized architecture and experimental validation. In: Proceedings of the 12th international meeting on DNA computing (DNA12), volume 4287 of LNCS, pp 393–403

Rothemund PWK, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangle. PLoS Biol 2:e424

Rothemund PWK, Winfree E (2000) The program-size complexity of self-assembled squares (extended abstracts). In: Proceedings of the 32nd annual ACM symposium on theory of computing (STOC 2000), pp 459–468. ACM

Schabanel N (2016) Simple OS simulator. http://perso.ens-lyon.fr/nicolas.schabanel/OSsimulator/

Watters KE, Strobel EJ, Yu AM, Lis JT, Lucks JB (2016) Cotranscriptional folding of a riboswitch at nucleotide resolution. Nat Struct Mol Biol 23(12):1124–1131