Towards the Algorithmic Molecular Self-Assembly of Fractals by Cotranscriptional Folding*

Yusei Masuda, Shinnosuke Seki**, and Yuki Ubukata

Department of Computer and Network Engineering, The University of Electro-Communications, 1-5-1, Chofugaoka, Chofu, Tokyo, 1828585, Japan
s.seki@uec.ac.jp

Abstract. RNA cotranscriptional folding has been recently proven capable of self-assembling a rectangular tile *in vitro* (RNA origami). The oritatami system is a novel computational model of cotranscriptional folding. In this paper, we initiate the theoretical study on the algorithmic self-assembly of shapes by cotranscriptional folding using the oritatami system. We propose an oritatami system that folds into an arbitrary finite portion of the Heighway dragon, which is a fractal also-known as the paperfolding sequence $P = \mathrm{RRLRRLLR}\cdots$. The i-th element of P can be obtained by feeding i in binary to a 4-state deterministic finite automaton with output (DFAO). We implement this DFAO and a bit-sequence bifurcation component as modules of oritatami system. Combining them with a known binary counter module yields the proposed oritatami system.

1 Introduction

An RNA sequence, over nucleotides of four kinds `A`, `C`, `G`, `U`, is synthesized (*transcribed*) from its template DNA sequence over `A`, `C`, `G`, `T` nucleotide by nucleotide by an RNA polymerase (RNAP) enzyme according to the one-to-one mapping `A` $\to$ `U`, `C` $\to$ `G`, `G` $\to$ `C`, and `T` $\to$ `A` (for details, see, e.g., [2]). The yield, called *transcript*, starts folding immediately after it emerges from RNAP. This is the *cotranscriptional folding*. Geary, Rothemund, and Andersen have recently demonstrated the capability of cotranscriptional folding to manufacture an RNA molecule of an intended shape at nano-scale [8]. They actually proposed an architecture of a DNA sequence whose transcript folds cotranscriptionally into an RNA rectangular tile highly likely *in vitro* (see Fig. 1).

In this paper, we shall initiate the theoretical study on algorithmic self-assembly of shapes by cotranscriptional folding using a novel computational model of cotranscriptional folding called the *oritatami system* [7]. Algorithms

* This work is in part supported by JST Program to Disseminate Tenure Tracking System, MEXT, Japan, No. 6F36 and by JSPS KAKENHI Grant-in-Aid for Young Scientists (A) No. 16H05854 to S. S.

** Corresponding author

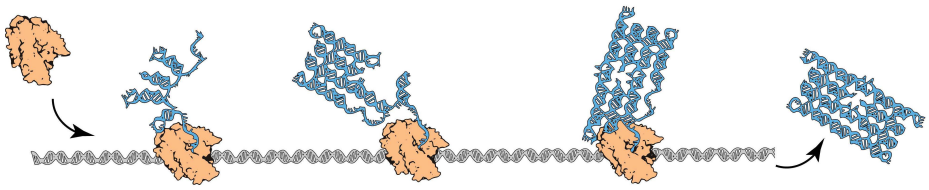

Fig. 1. RNA cotranscriptional folding. An RNA polymerase attaches to a template DNA sequence (gray spiral), scans it through, and synthesizes its RNA copy. The RNA sequence begins to fold upon itself immediately as it emerges from polymerase.

and computation are fundamental to molecular self-assembly as illustrated in an enormous success of their use in DNA tile self-assembly (see, e.g., [4, 14, 17] and references therein). A fractal pattern called Sierpinski triangle was algorithmically self-assembled from coalescence of DNA tiles that compute XOR [15]. Cotranscriptional folding exhibits highly sophisticated computational and algorithmic behaviors. Fluoride riboswitches in the *Bacillus cereus* bacteria cotranscriptionally fold a terminator stem that suppresses gene expression or not, depending on ligand concentration [16]. Cotranscriptional folding is in fact proved Turing-universal by the oritatami system [6]. The Turing-machine simulator is gigantic and intricate but oritatami systems have implemented basic computational devices such as binary counter [7] as a module comparable in size to the gene expression regulator. The binary counter module consists of half-adder components, which fold into one of possible four conformations depending on a 1-bit input and a 1-bit carry/non-carry encoded in their surroundings somehow. It can be diverted as a copier for binary sequences by being fed with the non-carry. They shall be reused in this paper.

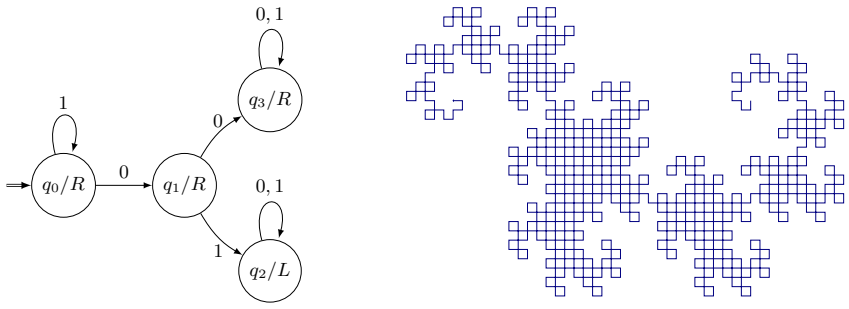

Fig. 2. Heighway dragon. (Left) A DFAO that reads the binary representation of i from the LSB and outputs the direction to turn. (Right) The first $2^{10}-1$ turns of the dragon.

Algorithmic cotranscriptional folding of Sierpinski triangle would allow oritatami systems to borrow rich insights from the DNA tile self-assembly. On the other hand, in order to cut more directly to the heart of algorithmic self-assembly by cotranscriptional folding, we should study the fabrication of shapes that are traversable in some algorithmic way. One such way is to feed a turtle program (see [1]) with a binary *automatic sequence* as commands (drawing a line segment, rotation, etc.), whose i-th bit (starting from 0) can be obtained by giving a binary representation of i from the least significant bit (LSB) to one deterministic finite automaton with output (DFAO) [3]. Shapes thus describable include the Heighway dragon [3] and von Koch curve [11]. A DFAO for the Heighway dragon is shown in Fig. 2 (Left). It is to read the binary representation of $i \geq 0$ from the LSB and outputs the i-th direction $P[i]$ to turn (L or R) assigned to the state finally reached as follows:

$$P = \text{RRLRRLLRRRLLRLLRRRLRRLLLRRLLRLL} \cdots .$$

(The notation P is after its appellative *paperfolding sequence* [3].) A turtle should interpret an L (resp. R) as "move forward by unit distance and turn left (resp. right) 90 degrees." An arbitrary portion of the Heighway dragon can be represented by a factor of P; for instance, $P[0..1022]$ represents the first $2^{10} - 1$ turns of the dragon, shown in Fig. 2 (Right).

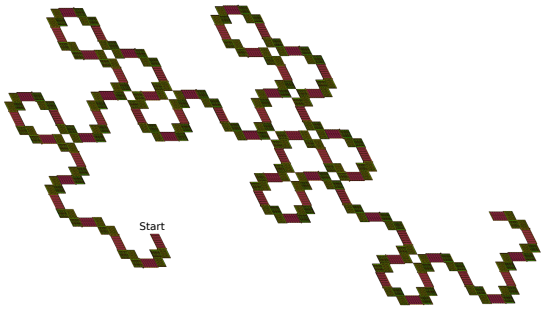

Fig. 3. The portion $P[0..62]$ of the Heighway dragon folded by the proposed oritatami system.

In this paper, we propose a generic design of oritatami system for the algorithmic cotranscriptional folding of an arbitrary finite portion of the Heighway dragon. Fig. 3 shows the portion $P[0..62]$ thus folded (the dragon is slanted, but this is because the oritatami system operates on the triangular grid). The systems consist of three modules: binary counter, DFAO module, and turning module. The binary counter is a technical modification of the one proposed in [7]. It folds into one zigzag and increments a given binary number by 1 if carried-in or just propagates the number otherwise. Binary counters fold into zigzags, yielding a (red) line segment of the dragon. The system feeds only the first counter with carry so that the line segment amounts to increment the current count i by 1. At the end of the segment is a DFAO module, which computes the turn direction $P[i]$ while propagating the count i to the following turning module. A (green) L-shaped block is the turning module. It is a concatenation of three bit-string bifurcation submodules, each of which folds into a rhombus, bifurcates the count i and the

direction $P[i]$ leftward as well as rightward, and directs the growth of further folding according to $P[i]$. We shall implement the DFAO and turning modules and verify them.

A JavaScript program to execute this oritatami system and websites on which this program is executable are freely available [12].

2 Preliminaries

Let Σ be a set of types of abstract molecules, or *beads*, and Σ^* be the set of finite sequences of beads. A bead of type $a \in \Sigma$ is called an a-bead. Let $w = b_1 b_2 \cdots b_n \in \Sigma^*$ be a string of length n for some integer n and bead types $b_1, \ldots, b_n \in \Sigma$. The *length* of w is denoted by $|w|$, that is, $|w| = n$. For two indices i, j with $1 \leq i \leq j \leq n$, we let $w[i..j]$ refer to the subsequence $b_i b_{i+1} \cdots b_{j-1} b_j$; if $i = j$, then we simplify $w[i..i]$ as $w[i]$. For $k \geq 1$, $w[1..k]$ is called a *prefix* of w.

Oritatami systems fold their transcript, a sequence of beads, over the triangular grid as suggested in Fig. 4 cotranscriptionally based on hydrogen-bond-based interactions (*h-interactions* for short) which the rule set of the system allow for between adjacent beads of particular types. Let $\mathbb{T} = (V, E)$ be the triangular grid graph. A directed path $P = p_1 p_2 \cdots p_n$ in $\mathbb{T}$ is a sequence of *pairwise-distinct* points $p_1, p_2, \ldots, p_n \in V$ such that $\{p_i, p_{i+1}\} \in E$ for all $1 \leq i < n$. Its i-th point is referred to as $P[i]$. A *rule set* $\mathcal{H} \subseteq \Sigma \times \Sigma$ is a symmetric relation over the set of pairs of bead types, that is, for all bead types $a, b \in \Sigma$, $(a, b) \in \mathcal{H}$ implies $(b, a) \in \mathcal{H}$.

A *conformation* C is a triple (P, w, H) of a directed path P in $\mathbb{T}$, $w \in \Sigma^*$ of the same length as P, and a set of h-interactions $H \subseteq \{\{i, j\} \mid 1 \leq i, i+2 \leq j, \{P[i], P[j]\} \in E\}$. This is to be interpreted as the sequence w being folded in such a manner that its i-th bead $w[i]$ is placed on the i-th point $P[i]$ along the path and there is an h-interaction between the i-th and j-th beads if and only if $(i, j) \in H$. The condition $i + 2 \leq j$ represents the topological restriction that two consecutive beads along the path cannot form an h-interaction between them. Let $\mathcal{H}$ be a rule set. An h-interaction $(i, j) \in H$ is *valid with respect to* $\mathcal{H}$, or simply $\mathcal{H}$*-valid*, if $(w[i], w[j]) \in \mathcal{H}$. This conformation C is $\mathcal{H}$-valid if all of its h-interactions are $\mathcal{H}$-valid. For an integer $\alpha \geq 1$, C is *of arity* α if the maximum number of h-interactions per bead is α, that is, if for any $k \geq 1$, $|\{i \mid (i, k) \in H)\}| + |\{j \mid (k, j) \in H\}| \leq \alpha$ and this inequality holds as an equation of some k. By $\mathcal{C}_{\leq \alpha}$, we denote the set of all conformations of arity at most α.

Oritatami systems grow conformations by elongating them under their own rule set. Given a rule set $\mathcal{H}$ and an $\mathcal{H}$-valid finite conformation $C_1 = (P, w, H)$, we say that another conformation C_2 is an *elongation of* C_1 *by a bead* $b \in \Sigma$, written as $C_1 \xrightarrow{\mathcal{H}}_b C_2$, if $C_2 = (Pp, wb, H \cup H')$ for some point p not along the path P and set of h-interactions $H' \subseteq \big\{\{i, |w|+1\} \;\big|\; 1 \leq i < |w|, \{P[i], p\} \in E, (w[i], b) \in \mathcal{H}\big\}$, which can be empty. Note that C_2 is also $\mathcal{H}$-valid. This operation is recursively extended to the elongation by a finite sequence of beads as: for any conformation

C, $C \xrightarrow{\mathcal{H}}{}^{*}_{\lambda} C$; and for a finite sequence of beads $w \in \Sigma^*$ and a bead $b \in \Sigma$, a conformation C_1 is elongated to a conformation C_2 by wb, written as $C_1 \xrightarrow{\mathcal{H}}{}^{*}_{wb} C_2$, if there is a conformation C' that satisfies $C_1 \xrightarrow{\mathcal{H}}{}^{*}_{w} C'$ and $C' \xrightarrow{\mathcal{H}}_{b} C_2$.

A finite *oritatami system* (OS) is a 5-tuple $\Xi = (\mathcal{H}, \alpha, \delta, \sigma, w)$, where $\mathcal{H}$ is a rule set, α is an arity, $\delta \geq 1$ is a parameter called the *delay*, σ is an initial $\mathcal{H}$-valid conformation of arity α called the *seed*, upon which its finite *transcript* $w \in \Sigma^*$ is to be folded by stabilizing beads of w one at a time so as to minimize energy collaboratively with the succeeding $\delta - 1$ nascent beads. The energy of a conformation $C = (P, w, H)$, denoted by $\Delta G(C)$, is defined to be $-|H|$; the more h-interactions a conformation has, the more stable it gets. The set $\mathcal{F}(\Xi)$ of conformations *foldable* by this system is recursively defined as: the seed σ is in $\mathcal{F}(\Xi)$; and provided that an elongation C_i of σ by the prefix $w[1..i]$ be foldable (i.e., $C_0 = \sigma$), its further elongation C_{i+1} by the next bead $w[i+1]$ is foldable if

$$C_{i+1} \in \underset{\substack{C \in \mathcal{C}_{\leq \alpha} \, s.t. \\ C_i \xrightarrow{\mathcal{H}}_{w[i+1]} C}}{\arg\min} \; \min \left\{ \Delta G(C') \mid C \xrightarrow{\mathcal{H}}{}^{*}_{w[i+2 \ldots i+k]} C', k \leq \delta, C' \in \mathcal{C}_{\leq \alpha} \right\}. \quad (1)$$

We say that the bead $w[i+1]$ and the h-interactions it forms are *stabilized* according to C_{i+1}. Note that an arity-α OS cannot fold any conformation of arity larger than α.

The OS Ξ is *deterministic* if for all $i \geq 0$, there exists at most one C_{i+1} that satisfies (1). Let us provide an example of deterministic OS that folds into a motif of great use called the *glider*.

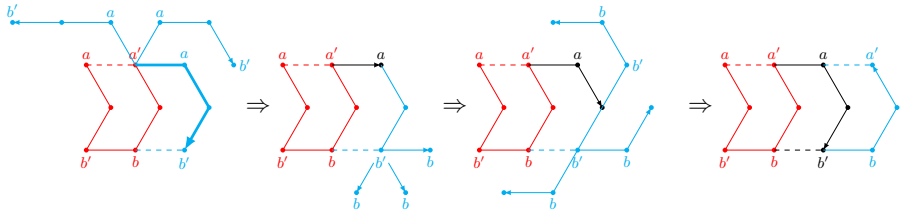

Fig. 4. Progression of a glider by distance 1.

Example 1. Let $\Sigma = \{a, a', b, b', \bullet\}$. Consider a delay-3 OS whose seed is colored in red in Fig. 4, transcript is a repetition of $a \bullet b'b \bullet a'$, and the rule set is $\mathcal{H} = \{(a, a'), (b, b')\}$, which makes $\bullet$-beads inert.

By the fragment $a \bullet b'$ of the first three beads of the transcript, the seed can be elongated in various ways, only three of which are shown in Fig. 4 (left). The only bead on the fragment capable of a new h-interaction is b' (with a b-bead according to $\mathcal{H}$), and there is only one elongation to let the b'-bead to interact, which is bolded in Fig. 4 (left). Thus, the first bead, a, is stabilized to the east

of the previous bead. The next bead, b, is then transcribed but it is not capable of any h-interaction because any b' around is either too far or too close. The bead transcribed afterward is inert. As a result, energy minimization ignores them and even the second and third bead, $\bullet b'$, are stabilized as in the bolded elongation. The first three beads $a, \bullet, b'$ have been thus stabilized and the glider has moved forward by unit distance. It is easily induced inductively that gliders of arbitrary "flight distance" can be folded.

Gliders also provide a medium to propagate 1-bit at arbitrary distance as the position of their last beads, which is determined by the height (top or bottom) of the first bead and a flying distance. For instance, the glider in Fig. 4 launches top and thus its last bead (the a') also comes top after traveling the distance 2. The oritatami system we shall propose exploits this information-carrying capability.

3 Heighway dragon oritatami system

We propose a generic design of deterministic oritatami system that allows us to fold an arbitrary finite portion of the Heighway dragon. The folded dragon is actually slanted as illustrated in Fig. 3, which is more natural than the conventional (upright) one to be folded over the triangular grid. Let $P[j_1..j_2]$ be the target portion. Let $n = \min\{m \mid j_2 < 2^m\}$. Independently of n, the design sets both delay and arity to 3 and employs 567 bead types and a fixed rule set $\mathcal{H}$ (some of the bead types might be saved but not easily due to the NP-hardness of minimizing the number of bead types [9]). The design challenges an extra requirement that the transcript be periodic; a periodic RNA transcript is likely to be transcribed out of a circular DNA sequence [5]. Without requiring the periodicity, one could simply design left-turn and right-turn modules and concatenate their copies according to the paperfolding sequence P. Such a "hardcoding" goes against the spirit of algorithmic self-assembly, and an OS that folds into the infinite Heighway dragon, if any, could not take this approach in order to be describable by a finite mean.

One period folds into successive two line segments of the dragon. Why do these two segments have to be distinguished from each other? The answer lies in the slant of the dragon. The slanted dragon involves two types of left turn as well as two types of right turn: acute and obtuse. Capability of one turning module to make all of the four possible turns would halve the period. Such a turning module, however, would have to take quite a number of conformations; recall that what the module has to turn is not a straw but a thick wire through which the current count i propagates. Such a module is too advanced for the level of current oritatami design techniques. Our approach makes it enough for a turning module to handle just two tasks: turning acutely and obtusely. Observe that after the (slanted) vertical segment, certainly the left turn is obtuse while the right turn is acute, whereas after the horizontal segment, the left turn is acute while the right turn is obtuse. In addition, vertical and horizontal segments occur alternately on the dragon. Therefore, we can attach a proper interpreter *a priori* to a DFAO module to convert its output L/R into a signal A(cute)/O(btuse). Two types of

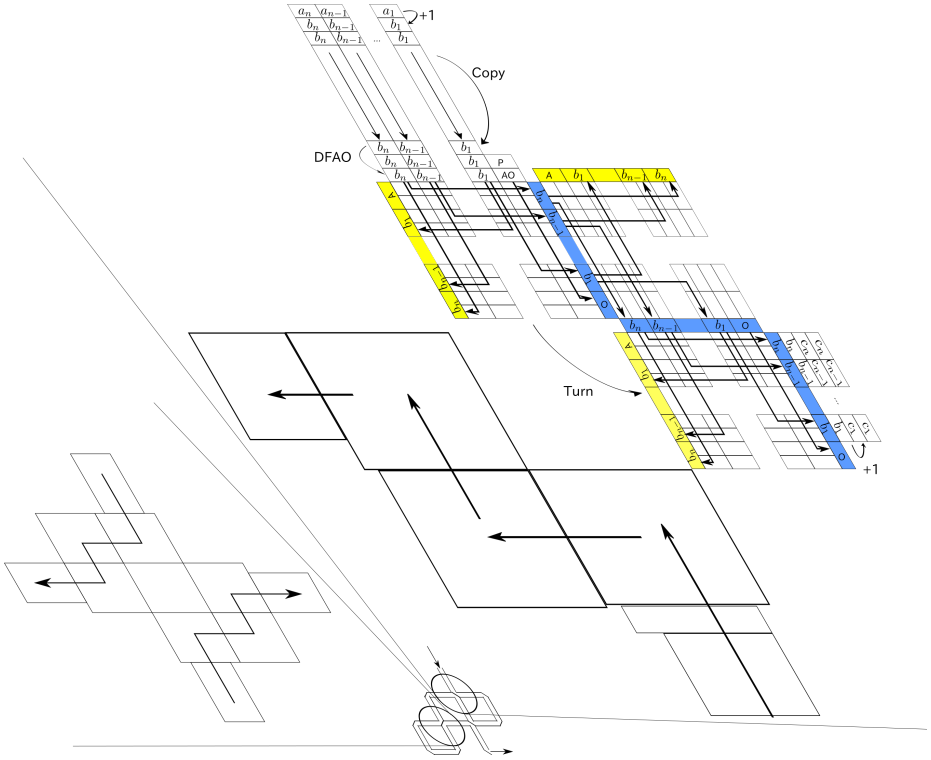

Fig. 5. Folding of one segment plus turn of the Heighway dragon, flow of information through it, and two ways of collision avoidance between two turns.

interpreters are hence needed: vertical interpreter converts L into obtuse and R into acute, while horizontal one converts them the other way around.

The sole difference between the first and second halves of a period is anyway the type of interpreter (AO or $\overline{\text{AO}}$). Thus, it should suffice to explain only the first half. The half, or more precisely, its transcript, consists of three subsequences for the counter, DFAO, and turning modules. It folds as abstracted in Fig. 5 into one line segment plus one turn of the dragon and has its three modules accomplish the following tasks, respectively:

1. $i \leftarrow i + 1$ (count-up) and copy i (drawing a line segment);
2. Compute $P[i]$ and interpret it as either A or O;
3. Make a turn accordingly.

The length of the half is proportional to n^2 and hence so is that of a period. The seed of the system replaces the first counter module of the first period and encodes the initial count j_1 as a sequence of bead types in a format that the following counters can "read." We shall explain how modules read something in Sect. 3.1. Before explaining the implementation of each module, we should

point out one significant issue specific to the folding by oritatami systems. It rises when the dragon makes a turn where it has already turned before, that is, when two turns share a point. By definition, oritatami systems cannot put a bead anywhere occupied by another bead. This is the reason of the L-shape of the turning module. As shown in Figs. 3 and 5, the proposed system makes an acute turn by having three bifurcation components direct the growth of further folding acutely one after another, while it makes an obtuse turn by having them direct the growth rather obtusely.

3.1 Modules

Now it suffices to explain how modules and their components have been implemented, interlocked with each other, and collaborate. Using the simulator developed for [10], we verified that all of the components fold correctly in all possible environments, which are abstracted in Figs. 5, 7, and 13.

Counter module The existing binary counter [7] was modified so as to operate in the dynamics (1), which is more prevailing [6, 9, 10, 13] but less tractable.

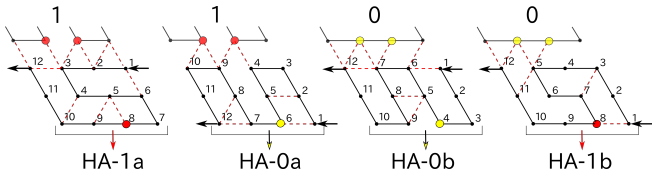

Fig. 6. All conformations of the half-adder component.

The essential component of this module is the half-adder. In Fig. 5, half-adders are abstracted as small rhombuses at the top labeled with $a_n, a_{n-1}, \ldots, a_1$, where the one with a_j is for the j-th bit of the current count i. Though not described, two horizontally adjacent half-adders sandwitch a glider of even length as a space to keep them away from each other sufficiently to prevent any undesirable interference. Figure 6 illustrates its four possible conformations. The whole oritatami system is designed in such a manner that a 1-bit input (1 or 0) is exposed to the specific position (colored red or yellow in Figure 6) as a pair of bead types by the previous turning module. The system is also designed in such a manner that a half-adder starts folding either at the top, as in the first and third conformations, or at the bottom as in the other two. The half-adder considers the top start position as non-carry and the bottom start position as carry. Thus, only when a half-adder takes 1 and carry, it ends at the bottom (second conformation in Figure 6), which is propagated to the next half-adder as it is via the spacer between them. These four conformations expose sequences of four bead types below that are distinct enough to convey the intended 1-bit

output to another computing component, or in other words, we can design a component that can "read" the output. From this point forward, conformations for other components will be illustrated; their input and output will be labeled by their meaning mutually understood by the components that exchange them.

Being fed with non-carry, the counter module serves as the copier module. Combining them one after another yields a line segment of arbitrary length, through which the current count i is propagated.

DFAO module A DFAO module receives the current count i from the last copier module, computes $P[i]$, interprets it properly either A or O, and outputs it together with the count i at the very end of a red line segment.

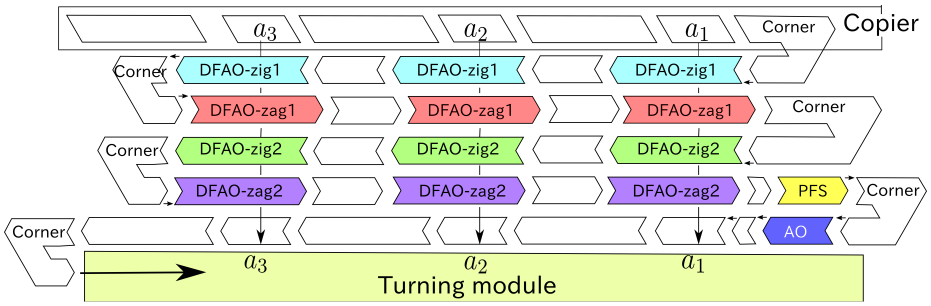

Fig. 7. Component-level abstraction of the folding of DFAO module.

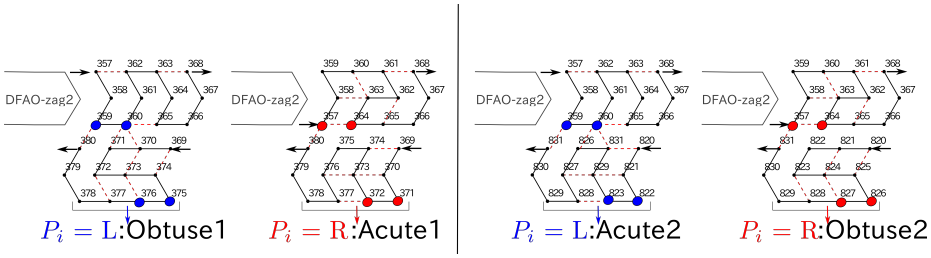

Fig. 8. (left) The possible two conformations of PFS: PFS-L, PFS-R. (right) The possible two conformations of AO.

DFAO is composed of the six components: DFAO-zig, DFAO-zag1, DFAO-zig2, DFAO-zag2, PFS, AO (or "$\overline{\text{AO}}$"). At the component-level, it folds into two zig-zags as abstracted in Figure 7. What the module does in the first zig-zag is to have DFAO-zig1s and DFAO-zag1s read the current count i from its LSB and "mark" the first 0, while propagating i. In the second zig-zag, it employs

DFAO-zig2s and DFAO-zag2s to check whether the automaton transitions to the state q_2 ($P_i = L$) or else ($P_i = R$). The second zag is to end at the top if $P_i = L$ or at the bottom if $P_i = R$. PFS takes one of the two conformations in Fig. 8 and outputs P_i downward. In vertical segments, PFS is followed by AO, while in horizontal segments, it is followed by $\overline{\text{AO}}$. AO interprets the PFS' output $P_i = R$ as acute and $P_i = L$ as obtuse, as shown in Fig. 8. $\overline{\text{AO}}$ interprets them the other way around. Let us explain briefly how each of the components folds to fulfill its roles.

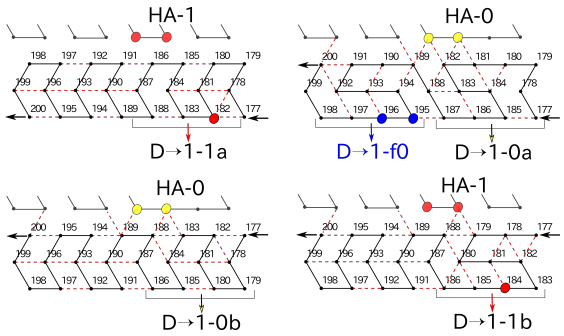

Fig. 9. The possible four conformations of DFAO-zig1: (top) Dzig1-1 and Dzig1-f0; (bottom) Dzig1-20 and Dzig1-21.

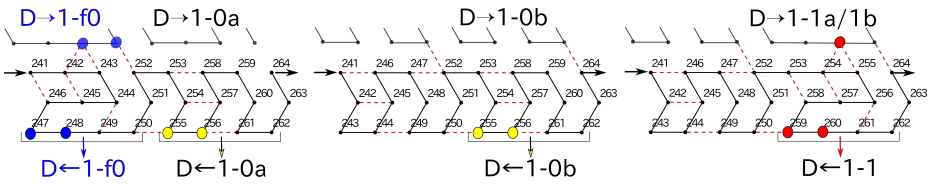

Fig. 10. The possible three conformations of DFAO-zag1: Dzag1-f0, Dzag1-0, and Dzag1-1 from left.

DFAO-zig1's collaboratively detect the first 0 in two phases. Phase1 is to copy all the 1's before the first 0 and Phase2 is to copy all the bits after the first 0. These phases are distinguished by the relative position at which a DFAO-zig1 starts folding to the copier above (in Phase1 it is at the bottom, while in Phase2 it is at the top, as suggested in Figure 9). In Phase1, DFAO-zig1s certainly take the conformation Dzig1-1 (the top left conformation in Figure 9). In Phase1, if the half-adder above outputs 0, this is the first 0. Then the DFAO-zig1 takes the conformation Dzig1-f0 instead, ending at the top to transition to Phase 2. Each of the succeeding DFAO-zig1s takes one of the other two

conformations Dzig1-20 and Dzig1-21 to copy all the remaining bits. Note that there is a cushion between two DFAO-zig1s called *spacer*. Spacers have been already used to prevent undesirable interference among components. They are implemented as a glider (see Example 1), hence capable of propagating 1bit on which phase the system is in. In the first zag, DFAO-zag1's just propagate 0's, 1's, and the first 0 by taking the proper one of the three conformations in Figure 10.

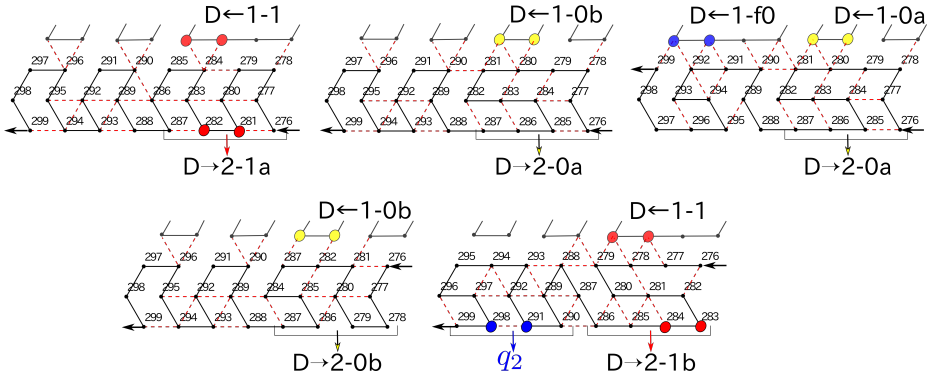

Fig. 11. The possible five conformations of DFAO-zig2: (top) Dzig2-1, Dzig2-0, Dzig2-f0, (bottom) Dzig2-f00, and Dzig2-f01.

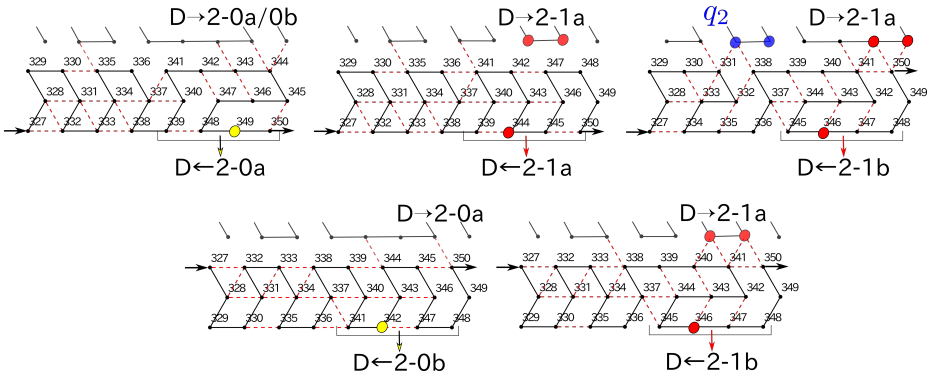

Fig. 12. The possible five conformations of DFAO-zag2: (top) Dzag2-L0, Dzag2-L1, Dzag2-T1, (bottom) Dzag2-R0, and Dzag2-R1.

In the second zig, DFAO-zig2's check whether the first 0 is followed by 0 or 1, being read from LSB. They first copy all the 1's up to the first 0 by taking the conformation Dzig2-1 (top left in Figure 11). The next letter is the first 0,

which is distinguished from other 0's by the special conformation Dzag1-f0 of the DFAO-zag1 responsible for the bit, or more precisely, by its marker f0. Starting at the bottom and reading 0, the DFAO-zig2 can take two conformations Dzig2-0 and Dzig2-f0. These conformations share the first half. The marker f0 folds the second half so as to end at the top, yielding Dzig2-f0. The next DFAO-zig2 therefore starts to fold at the top so that it takes one of the two conformations Dzig2-f00 and Dzig2-f01 depending on the bit read. Recall that reading 1 here is equivalent to transitioning to q_2, that is, $P[i] = L$. Observe that Dzig2-f01 is provided with the marker q_2, which lets the DFAO-zag2 component below know $P[i] = L$. These conformations end at the bottom. The remaining 0's and 1's are copied by Dzig2-0 and Dzig2-1, respectively. The second zag starts at the bottom and copy 0's and 1's by the two conformations Dzag2-L0 and Dzag2-L1 of DFAO-zag2 (top left and center in Figure 12) until a DFAO-zag2 encounters a 1, or more precisely, its marker q_2, if any. Such DFAO-zag2 takes the special conformation Dzag2-T1 and changes the ending position to the top, letting the remaining DFAO-zag2s rather take Dzag2-R0 and Dzag2-R1 for copying, which end at the top. As such, the second zag can feed $P[i]$ to PFS as explained before, while propagating the current count i.

The third zig lets $P[i]$ go through its AO (or $\overline{\mathrm{AO}}$) component to be reinterpreted either as A(cute) or as O(btuse) and propagates the current count i.

Turning module Lastly, we explain the turning module. It consists of two parts: bit-sequence bifurcation submodule and steering arm component (colored in blue in Figure 13).

The bifurcation submodule sends bits of the current count i and the reinterpreted signal (A or O) as shown in Figure 5 while folding into zig-zags. For that, it employs the following four types of components:

1. components to propagate 1-bit vertically: body-rpx (Figure 24), body-lpx1 (Figure 15), body-lpx2 (Figure 16);
2. a component to let 1-bit cross another 1-bit: body-gx (Figure 17);
3. components to fork 1-bit vertically and horizontally: body-rgy (Figure 18) and body-lgy (Figure 19);
4. components to undergo transition between a zig and a zag and exposes 1-bit outside: turn-rgp (Figure 20) and turn-lgp (Figure 21).

The first two types have already been implemented (see, e.g., [10]) so that we shall explain the others.

The component body-rgy takes one of the two conformations in Figure 18 depending on the 1-bit encoded in the two beads above. Output below, the 1-bit is encoded as a type of the second bead from left, while output right, it is encoded as the position of its last bead (top or bottom). Its zag-variant, body-lgy, is implemented analogously; for its conformations.

The 1bit forked rightward by a body-rgy transfers till the end of the zig without being jammed because all remaining modules in the zig are designed in

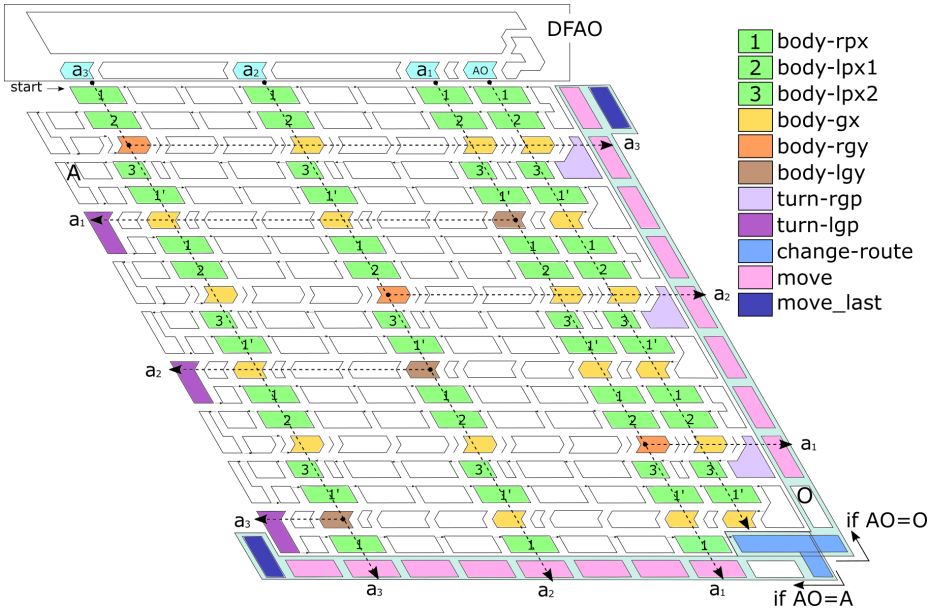

Fig. 13. Component-level abstraction of folding of turning module. All the white components in the middle are spacers, some of which are implemented in the shape of parallelogram instead of glider.

such a way that they start and end at the same height like the even-distance glider. The module turn-rgp receives the 1bit (top or bottom), and exposes it by taking one of the two comformations in Figure 20. The module turn-lgp functions analogously in zags as being folded in Figure 21.

The bifurcation submodule also propagates the 1-bit A or O, output by the DFAO, to tell the steering arm which way it should take. Specifically, the signal has the first module, change-route, of the steering arm take one of the two conformations in Figure 22, guiding the rest of the steering arm towards the ordered direction. The steering arm is provided with move modules (Figure 23), which let bits of the bifurcated bit sequence pass through. Note that the turning module does not have to bifurcate AO. Indeed, the second and third turning modules are supposed to turn in the same manner as the first one. It hence suffices to append A and O to the bifurcated bit sequences on the acute side and obtuse side, respectively, as shown in Figure 13.

Remark 1. In fact, as suggested in Figure 13, the bifurcation component outputs an input bit sequence also downward. That is, it bifurcates the sequence into three ways. This provides a more space-efficient way to replicate a bit sequence many-folds.

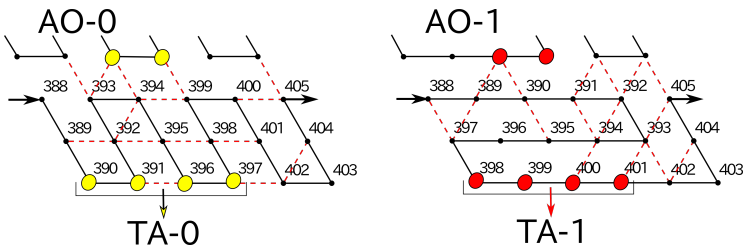

Fig. 14. The possible two conformations of body-rpx component.

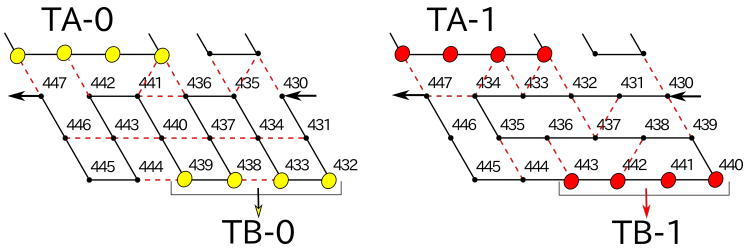

Fig. 15. The possible two conformations of body-lpx1 component.

Acknowledgements

We would like to thank Hwee Kim and Aleck Johnsen for their helpful advices and discussions.

References

1. Abelson, H., diSessa, A.: Turtle Geometry The Computer as a Medium for Exploring Mathematics. MIT Press Series in Artificial Intelligence, The MIT Press (1981)
2. Alberts, B., Johnson, A., Lewis, J., Morgan, D., Raff, M., Roberts, K., Walter, P.: Molecular Biology of the Cell. Garland Science, 6th edn. (2014)
3. Allouche, J.P., Shallit, J.: Automatic Sequences: Theory, Applications, Generalizations. Cambridge University Press (2003)
4. Doty, D.: Theory of algorithmic self-assembly. Communications of the ACM 55(12), 78–88 (2012)
5. Geary, C., Andersen, E.S.: Design principles for single-stranded RNA origami structures. In: Proc. DNA20. vol. LNCS 8727, pp. 1–19. Springer (2014)
6. Geary, C., Meunier, P.E., Schabanel, N., Seki, S.: Efficient universal computation by greedy molecular folding (2015), coRR abs/1508.00510
7. Geary, C., Meunier, P.E., Schabanel, N., Seki, S.: Programming biomolecules that fold greedily during transcription. In: Proc. 41st International Symposium on Mathematical Foundations of Computer Science (MFCS2016). pp. 43:1–43:14. Leibniz International Proceedings in Informatics (LIPIcs) 58 (2016)

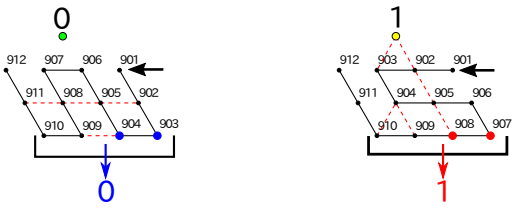

Fig. 16. The possible two conformations of body-lpx2 component.

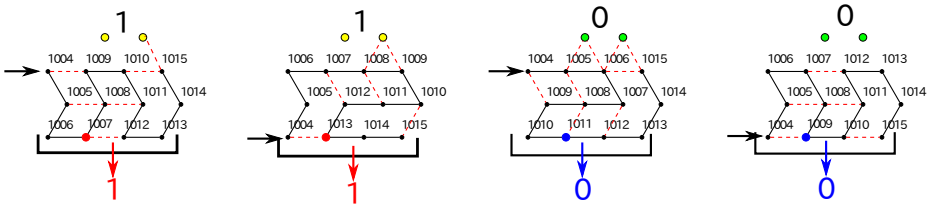

Fig. 17. The possible four conformations of body-gx component.

8. Geary, C., Rothemund, P.W.K., Andersen, E.S.: A single-stranded architecture for cotranscriptional folding of RNA nanostructures. Science 345(6198), 799–804 (2014)
9. Han, Y.S., Kim, H.: Ruleset optimization on isomorphic oritatami systems. In: Proc. 23rd International Conference on DNA Computing and Molecular Programming (DNA23). LNCS, Springer (2017), in press
10. Han, Y.S., Kim, H., Ota, M., Seki, S.: Nondeterministic seedless oritatami systems and hardness of testing their equivalence. In: Proc. 22nd International Conference on DNA Computing and Molecular Programming (DNA22). LNCS, vol. 9818, pp. 19–34. Springer (2016)
11. Ma, J., Holdener, J.: When Thue-Morse meets Koch. Fractals 13, 191–206 (2005)
12. Masuda, Y., Seki, S., Ubukata, Y.: The simulator of heighway dragon
13. Ota, M., Seki, S.: Ruleset design problems for oritatami systems. Theoretical Computer Science 671, 26–35 (2017)
14. Patitz, M.J.: Self-assembly of fractals. In: Encyclopedia of Algorithms, pp. 1918–1922. Springer (2016)
15. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangle. PLoS Biology 2(12), e424 (2004)
16. Watters, K.E., Strobel, E.J., Yu, A.M., Lis, J.T., Lucks, J.B.: Cotranscriptional folding of a riboswitch at nucleotide resolution. Nature Structural & Molecular Biology 23(12), 1124–1133 (2016)
17. Winfree, E.: Algorithmic Self-Assembly of DNA. Ph.D. thesis, California Institute of Technology (June 1998)

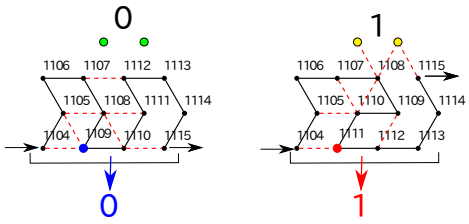

Fig. 18. The possible two conformations of body-rgy component.

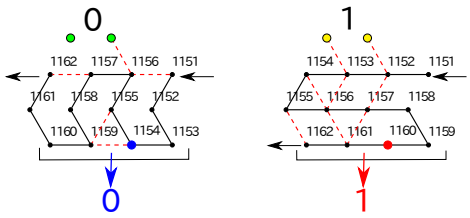

Fig. 19. The possible two conformations of body-lgy component.

Appendix

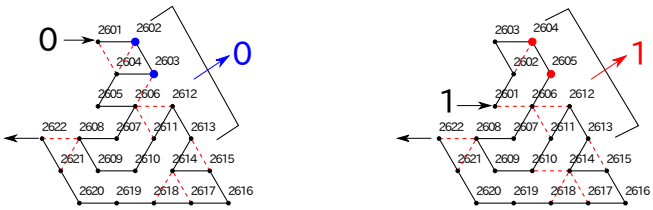

Fig. 20. The possible two conformations of turn-rgp component.

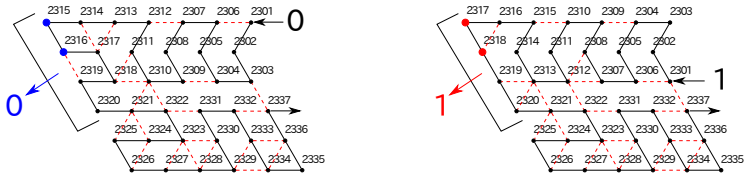

Fig. 21. The possible two conformations of turn-lgp component.

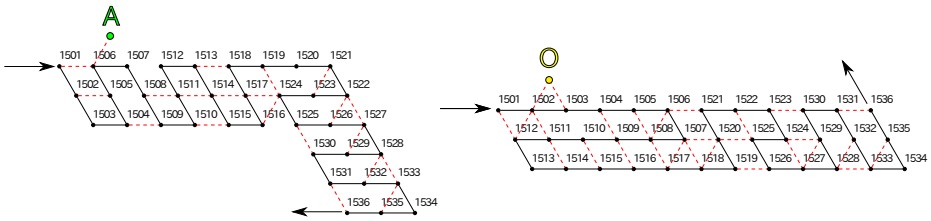

Fig. 22. The possible two conformations of change-route component.

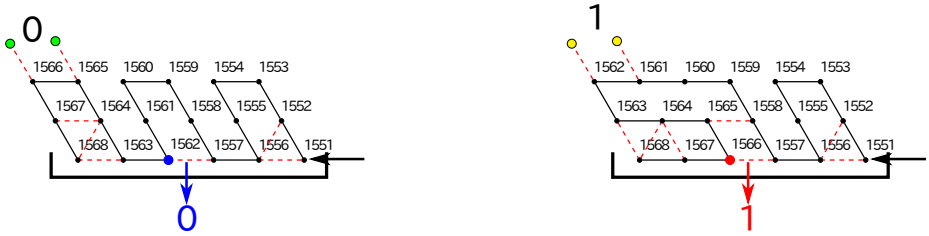

Fig. 23. The possible two conformations of move component.

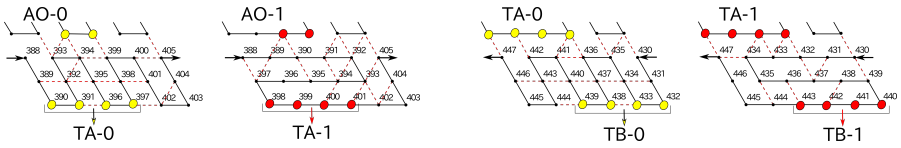

Fig. 24. (Left) The two conformations of body-rpx component. (Right) The two conformations of body-lpx1 component.