Counting infinitely by Oritatami co-transcriptional folding

Kohei Maruyama[1⋆] and Shinnosuke Seki[1,2⋆⋆]

[1] The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo, 1828585, Japan
[2] École Normale Superiéure de Lyon, 46 allée d'Italie, 69007, Lyon, France

Abstract. A fixed bit-width counter was proposed as a proof-of-concept demonstration of an oritatami model of cotranscriptional folding [Geary et al., Proc. MFCS 2016, LIPIcs 58, 43:1-43:14], and it was embedded into another oritatami system that self-assembles a finite portion of Heighway dragon fractal. In order to expand its applications, we endow this counter with capability to widen bit-width at every encounter with overflow.

1 Introduction

Counting is one of the most essential tasks for computing; as well known, it suffices to count for Turing universality [8]. Nature has been counting billions of days using molecular "circadian clockwork" which is "as complicated and as beautiful as the wonderful chronometers developed in the 18th century" [7]. Nowadays, developments in molecular self-assembly technology enable us to design molecules to count. Evans has recently demonstrated a DNA tile self-assembly system that counts accurately in binary from a programmed initial count until it overflows [3]. In its foundational theory of molecular self-assembly, such binary counters have proved their versatility, being used to assemble shapes of particular size [1,9], towards self-assembly of fractals [6], as an infinite scaffold to simulate all Turing machines in parallel in order to prove undecidability of nondeterminism in the abstract tile-assembly model [2], to name a few.

⋆ Primary corresponding author (k.maruyama@uec.ac.jp)
⋆⋆ Secondary corresponding author (s.seki@uec.ac.jp)

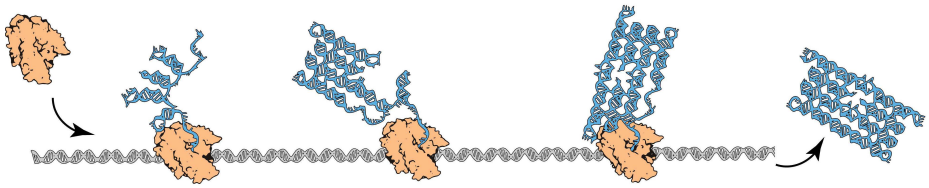

Fig. 1. RNA origami

A fixed bit-width (finite) binary counter has been implemented as a proof-of-concept demonstration of oritatami model of cotranscriptional folding [4]. As shown in Fig. 1, an RNA transcript folds upon itself while being transcribed (synthesized) from its corresponding DNA template strand. Geary et al. programmed a specific RNA rectangular tile structure into a DNA template in such a way that the corresponding RNA transcript *folds cotranscriptionally* into the programmed tile structure highly likely *in vitro* at room temperatures (*RNA origami*) [5]. An oritatami system folds a transcript of abstract molecules called *beads* of finitely many types cotranscriptionally over the 2-dimensional triangular lattice cotranscriptionally according to a rule set that specifies which types of molecules are allowed to bind. The transcript of the binary counter in [4] is of period 60 as ⓪−①−②−$\cdots$−㊽−㊾−⓪−①$\cdots$ and its period is semantically divided into two half-adder (HA) modules A = ⓪−①−$\cdots$−⑪ and C = ㉚−㉛−$\cdots$−㊶ and two structural modules B and D, which are sandwiched by half-adder modules. While being folded cotranscriptionally in zigzags, HA modules increment the current count i by 1, which is initialized on a linear *seed* structure, alike the Evans' counter, whereas structural modules B and D align HA modules properly and also make a turn at an end of the count i; B guides the transcript from a zig to a zag ($\hookrightarrow$) while D does from a zag to a zig ($\hookleftarrow$). This counter was embedded as a component of an oritatami system to self-assemble an arbitrary finite portion of Heighway dragon fractal [6]. Its applications are limited, however, by lack of mechanism to widen bit-width; its behavior is undefined when its count overflows. In this paper, we endow this counter, or more precisely, its structural module B, with capability to widen the count by 1 bit at every encounter with overflow.

2 Preliminaries

Let Σ be a finite alphabet, whose element should be regarded as type of abstract molecule, or *beads*. A bead of type $a \in \Sigma$ is called an a-bead. By Σ^* and Σ^ω, we denote the set of finite sequences of beads and that of one-way infinite sequences of beads, respectively. The empty sequence is denoted by λ. Let $w = b_1 b_2 \cdots b_n \in \Sigma^*$ be a sequence 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 $w[i..i]$ is simplified as $w[i]$. For $k \geq 1$, $w[1..k]$ is called a *prefix* of w.

Oritatami systems fold their transcript, which is a sequence of beads, over the triangular grid graph $\mathbb{T} = (V, E)$ cotranscriptionally. For a point $p \in V$, let $\hexagon_p^d$ denote the set of points which lie in the regular hexagon of radius d centered at the point p. Note that $\hexagon_p^d$ consists of $3d(d+1)+1$ points. 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 $P[i]$. Now we are ready to abstract RNA single-stranded structures in the name of conformation. A *conformation* C (over Σ) 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

$H \subseteq \big\{\{i,j\} \;\big|\; 1 \leq i, i+2 \leq j, \{P[i], P[j]\} \in E\big\}$. This is to be interpreted as the sequence w being folded along the path P in such a manner that its i-th bead $w[i]$ is placed at the i-th point $P[i]$ and the i-th and j-th beads are bound (by a hydrogen-bond-based interaction) 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 be bound. The *length* of C is defined to be the length of its transcript w (that is, equal to the length of the path P). A *rule set* $R \subseteq \Sigma \times \Sigma$ is a symmetric relation over Σ, that is, for all bead types $a, b \in \Sigma$, $(a,b) \in R$ implies $(b,a) \in R$. A bond $\{i,j\} \in H$ is *valid with respect to* R, or simply R-valid, if $(w[i], w[j]) \in R$. This conformation C is *R-valid* if all of its bonds are R-valid. For an integer $\alpha \geq 1$, C is *of arity* α if it contains a bead that forms α bonds but none of its beads forms more. By $\mathcal{C}_{\leq\alpha}(\Sigma)$, we denote the set of all conformations over Σ whose arity is at most α; its argument Σ is omitted whenever Σ is clear from the context.

The oritatami system grows conformations by an operation called elongation. Given a rule set R and an R-valid 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 $C_1 \xrightarrow{R}_b C_2$, if $C_2 = (Pp, wb, H \cup H')$ for some point $p \in V$ not along the path P and set $H' \subseteq \big\{\{i, |w|+1\} \;\big|\; 1 \leq i < |w|, \{P[i], p\} \in E, (w[i], b) \in R\big\}$ of bonds formed by the b-bead; this set H' can be empty. Note that C_2 is also R-valid. This operation is recursively extended to the elongation by a finite sequence of beads as: for any conformation C, $C \xrightarrow{R}{}^*_\lambda C$; 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 $C_1 \xrightarrow{R}{}^*_{wb} C_2$, if there is a conformation C' that satisfies $C_1 \xrightarrow{R}{}^*_w C'$ and $C' \xrightarrow{R}_b C_2$.

An *oritatami system* (OS) Ξ is a tuple $(\Sigma, R, \delta, \alpha, \sigma, w)$, where Σ and R are defined as above, while

- a positive integer δ called the *delay*,
- a positive integer α called the *arity*,
- an initial R-valid conformation $\sigma \in C_{\leq\alpha}(\Sigma)$ called the *seed*,
- a (possibly infinite) *transcript* $w \in \Sigma^* \cup \Sigma^\omega$, which is to be folded upon the seed 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 bonds a conformation has, the more stable it gets. The set $\mathcal{F}(\Xi)$ of conformations *foldable* by the system Ξ 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

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

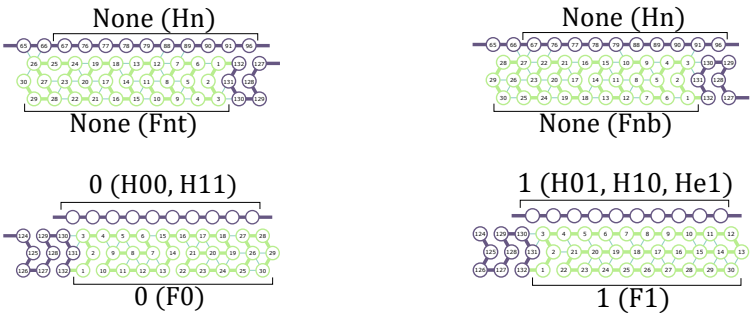

Fig. 2. All the 4 bricks of module F: The 2 bricks at the top are for zigs while the 2 bottom are for zags.

Then we say that the bead $w[i+1]$ and the bonds it forms are *stabilized* according to C_{i+1}. The easiest way to understand this stabilization process should be the video available at `https://www.dailymotion.com/video/x3cdj35`, in which the Turing universal oritatami system by Geary et al. [11], whose delay is 3, is running. Note that an arity-α oritatami system cannot fold any conformation of arity larger than α. A conformation foldable by Ξ is *terminal* if none of its elongations is foldable by Ξ.

3 Folding an infinite binary counter

3.1 General idea

Between two consecutive overflows, the proposed system behaves in the same way as the finite binary counter of Geary et al. [4]. Its transcript folds in a zigzag manner macroscopically (downward in figures throughout this paper). A zig, folding from right to left, increments the current value of the counter by 1. The succeeding zag, folding from left to right, formats the incremented value for the sake of next zig and copies it downward. Unlike the counter of Geary et al., when a zig encounters an overflow, it does not abort but extends the current value by 1 bit.

The transcript of our counter is periodic. Its period 1-2-3- $\cdots$ -132 is semantically divided into the following 4 subsequences, called *modules*:

- 1–30: Format module or F
- 31–66: Left-Turn module or L
- 67–96: Half-Adder module or H
- 97–132: Right-Turn module or R

Modules are to play their roles in expected environments by folding into respective conformations which should be pairwise-distinct enough to be distinguishable by other modules transcribed later. Such expected conformations are called a *brick*. For example, module F encounters only the four environments shown in Fig. 2 where it takes the four bricks, respectively.

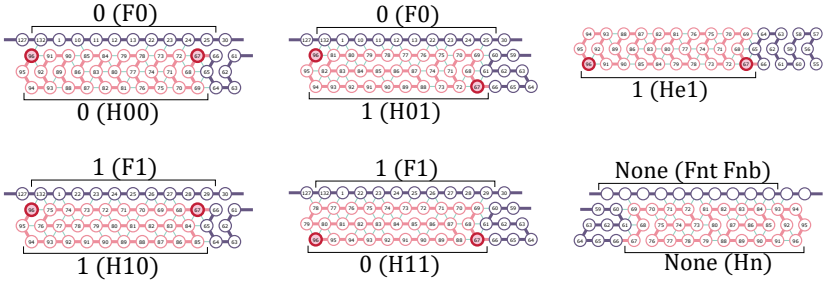

Fig. 3. All the six bricks of module H: `H00`, `H01`, `He1`, `H10`, `H11`, and `Hn` from top left to bottom right. In zags, H always folds into `Hn` while in zigs, it folds into one of the other five bricks.

Seed and Encoding. The initial counter value is encoded as $b_{k-1}b_{k-2}\cdots b_1b_0$ in binary on the seed in the following format

$$64{-}65{-}66{-}\left(\prod_{i=k-1}^{0}\left(w_{Hn}w_{Rb}w_{Fb_i}w_{Lbn}\right)\right)w_{Hn} \tag{2}$$

where $w_{Hn}, w_{Rb}, w_{Fb_i}, w_{Lbn}$ are sequences of bead types exposed downward by modules H, R, F, L when they fold into bricks Hn, Rb, Fb_i, Lbn, respectively, which can be found in Figs. 3, 5, 2, and 4. The seed for $k = 1$ and $b_0 = 0$ is colored in purple in Fig. 6.

In the zig (←) Here, the counter lets instances of module H increment its current value, which is encoded in the format (2) below the seed or previous zag. When the current value is k bits in width, a subsequence $(FLHR)^kF$ of the transcript folds into the zig. All the bricks which modules take in a zig are of height 3 (see Fig. 2, 3, 4, and 5) so that the zig turns out to be a linear structure of height 3. Carry is propagated throughout the zig as a height for modules to start. All modules but H end at the same height as it began, that is, do not change carry. Module H can take the five bricks `H00`, `H01`, `H10`, `H11`, and `He1` shown in Fig. 3 in zigs but nothing else. This is because the whole system is designed in such a way that module H encounters only the five environments. Until the count overflows, module H encounters only four of them, which encode input 0 as w_{F0} or 1 as w_{F1} and carry or no-carry as the height for the module to start folding (bottom or top), where it takes `H00`, `H01`, `H10`, and `H11`, respectively, as shown in Fig. 3 (Hxc is folded when the input is x and the carry is given if $c = 1$ or not otherwise).

Let us see how the subsequence $(FLHR)^kF$ folds into a zig; for $k = 1$ and the current value 0, see Fig. 6. The zig is initially fed with carry by being forced to start folding at the bottom and the carry is propagated through the first instances of F and L in the way just explained toward the first instance of H. This H is thus fed with carry and folds into `H01` if the bit encoded above

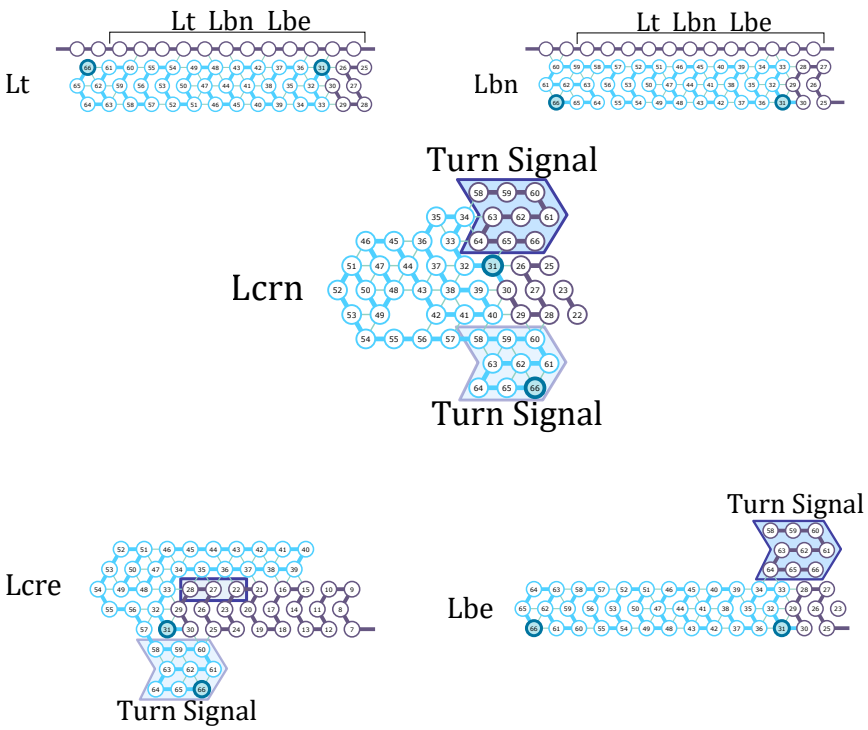

Fig. 4. All the five bricks of module L: `Lt`, `Lbn`, `Lcrn`, `Lcre`, and `Lbe` from top left to bottom right. In zigs, L folds into either `Lt` or `Lbn` depending on where it starts, until the transcript reaches the left end, where L folds either into `Lcrn` if the current value has not been overflowed, or into `Lbe` at an overflow. In the case of overflow, the next L folds into `Lcre`. In zags, L always folds into `Lbn`.

is 0, as illustrated in Fig. 6, or `H11` if the bit is rather 1. `H01` ends at the top, corresponding to no-carry. This absence of carry propagates through the succeeding modules leftward. As a result, the zig ends, or more precisely an instance of F ends folding at the bottom if the current value is overflowed (Fig. 8), or at the top otherwise. (Fig. 6). An instance of L is to be transcribed next. It folds either into `Lcrn` for (normal) carriage-return unless the current value is overflowed, or into `Lbe` at an overflow.

In the zag (→) Here, the value incremented in the previous zig is formatted so as to be exposed below in the format (2) for the next zig. A subsequence $(HRFL)^kH$ of the transcript folds into the zag, where k is the bit width of the incremented value. Both of the bricks of L for carriage-return, i.e., `Lcrn` and `Lcre`, ends at the bottom so that zags start at the bottom. The first instances of H and R fold into `Hn` and `Rb`, respectively. Hence, the next F starts at the

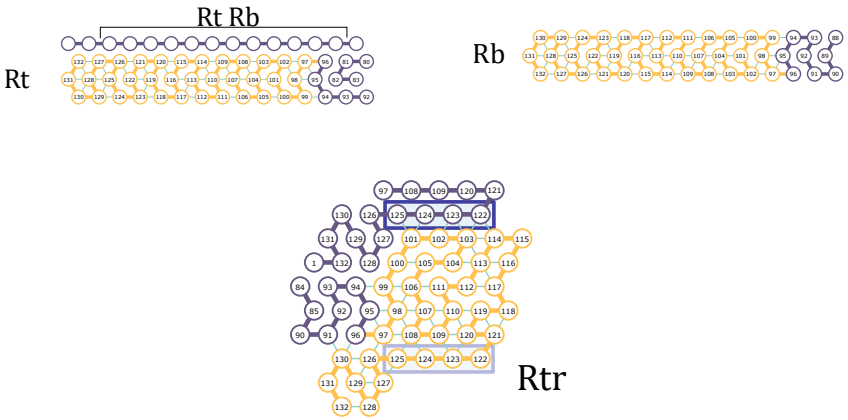

Fig. 5. All the three bricks of module R: `Rt`, `Rb`, and `Rcr`. In zigs, R folds into `Rt` or `Rb`. In zags, R always folds into `Rb` until the transcript reaches the right end, where R folds into `Rcr`.

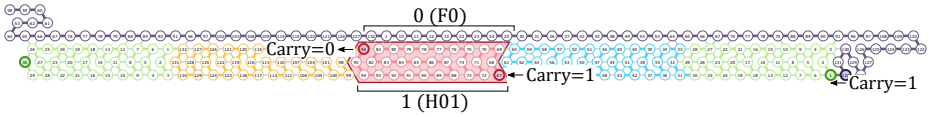

Fig. 6. The first zig pass. The seed is encoded with "0" same as bottom of F0(①-⑩-⑪-⑫-⑬-㉒-㉓-㉔-㉕-㉚), and the counter increments it by giving a carry. Module H outputs "1" as a sum and cancels the carry.

bottom and takes the brick `F`y under the module H that takes the brick `H`xc in the previous zig, where y is the output of the module, that is, $y = (x + c)$ mod 2. Both of these bricks `F0` and `F1` end at the bottom so that the instance of L folds into `Lbn`.

In the case of overflow, instance of H, R, and F are transcribed following `Lbc` and they fold into `He1`, `Rb`, and `Fnb`, respectively. Next instance of L which is transcribed after `Fnb` carriage-return for the next zag. However, if its instance turn, Turn Signal must be present around there so the top of `Fnb` is given the role of Turn Signal. Unless the count overflows, there is a brick above `Fnb` as shown in Fig. 8, so the top of `Fnb` is hidden, but when it overflows, Turn Signal is exposed.

Module H read from the row above the value encoded into folding in the module F during the previous zag-phase, and fold into a shape H00, H10, H01, H11. Hxc is corresponding to the case where x is the bit read in the row above and c is the carry. In the zig pass module R, F, and L just propagate the carry value to the next.

When the zig pass reaches the left most part, module L collaborates with above turn module and it is folded turn module which reverses the folding direction for next zag pass except carry = 1. At this time, if module L has a carry,

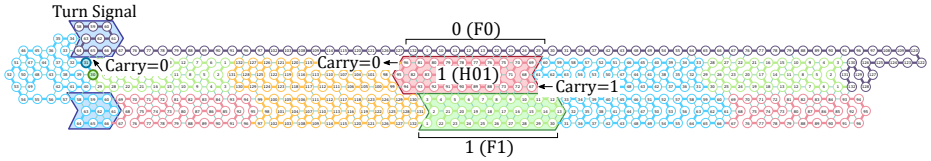

Fig. 7. Module L turns and start the first zag pass. Since there is a Turn Signal at the left end of the seed, when the carry is 0, module L turns and at the end of L forms Turn Signal. In zag pass, module F reads the output of module H and copies it down,

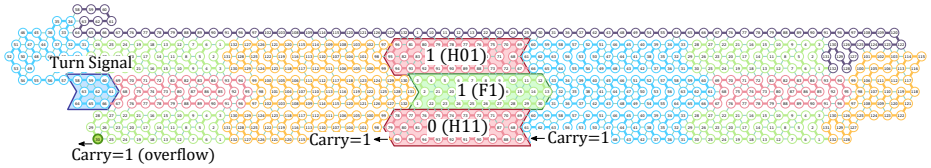

Fig. 8. Reach the left end with carry. Even if transcript of module L sticks to Turn Signal, it can not bind because the distance is long.

folding of module L is glider and it go straight in order to expand bit width after that next module L turns to zag pass while folding turn module.

In the zag (→) This part, instances of module F copy the current value, which incremented in the previous zig and encoded at below the zig. Module F is always folded under module H in zags and F takes the two bricks F0 and F1 shown in Fig. 2 which depending on a brick of module H. Other modules H,R, and L take bricks which do nothing Hn, Rb, and Lbc, respectively, shown in Fig. 3, 5, 4.

a

Module F read form the row above Hxc, and fold into a shape F0, F1. Fy is corresponding to the case where $y = (x + c) \bmod 2$. There are no carry propagation and the module L, H, R fold into Lbc, Hn, Rb in this pass.

When the zag pass reaches the right most part. module R collaborates with above turn module and it is folded turn module which reverses the folding direction for next zag pass.

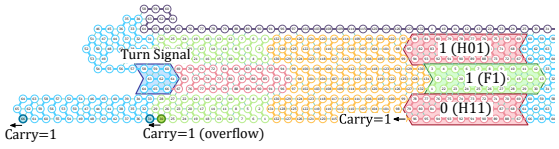

Fig. 9. Module L forms a glider shape without binding to Turn Signal and goes straight on.

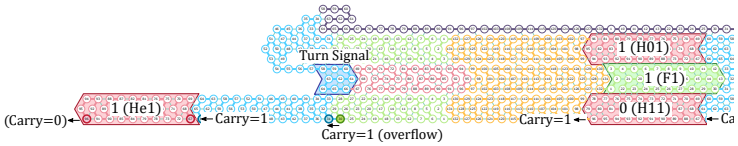

Fig. 10. Module H catches the overflowed carry but module H can not cancel the carry because it must be a self standing glider. That is way the last bead of this module H is folded at the bottom but carry = 0.

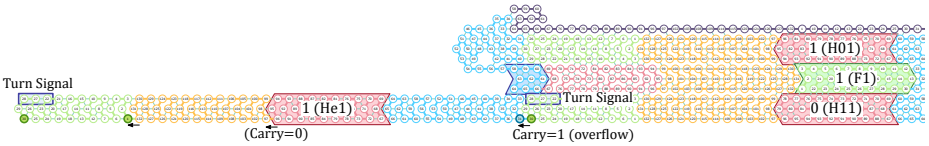

Fig. 11. Module R and F are folded, and then folding of module L starts, but without Turn Signal, L can not turn. Then, turn module L by making the top of module F the Turn Signal.

3.2 Example of folding first zigzag

Let's run an infinite counter from 1 bit width. The seed is encoded with "0", and the starts from bottom row, that is, giving a carry for the least significant bit.

The first zig pass (←) In Fig, 6, the zig pass folds as follows.

- [First module F (Light Green)]
  Normally, module H is folded above module F, and the seed corresponding to this module H now represents None. Then, module F forms a shape which represents None and propagates carry equal to 1.
- [First module L (Light Blue)]
  Since the seed here does not have a turn signal, module L forms glider and propagates carry while extending straight.

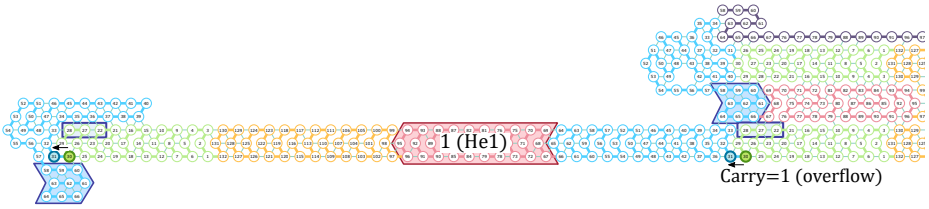

Fig. 12. Module L turns by module F's Turn Signal and zag pass starts. Also, the end of L takes the form of Turn Signal.

– [First module H (Pink)]
  The seed is encoded 0 and module folding starts from the bottom row, that is, the carry equal to 1. Then, HA folds into H01, which represents 1 and is output of carry equal 0.
– [First module R (Yellow) and 2nd module F]
  Both module form glider which start from top row and end to top row, in order to propagate carry equal 0.

The first zag pass (→) In Fig. 6, it is folded until second module F, and module L is transferred as the next transcript. Fig. 7 shows how zag pass is folded. Now, there is a Turn Signal at the left end of the seed, so L's transcript binds to its signal and loses the shape of glider in order to turn. In addition, the turned module L forms a Turn Signal at its end.

The zag pass folds as follows.

– [module H and module R]
  Since the module F of last module in zig pass represents None, module H folds into Hn also representing None. The module R form glider for margin.
– [module F, L, and H]
  There are three types of brick, H01, H10 and He1, representing 1 in module H, and different signals are defined as 1 any. Then, module F folds into brick F1 no matter which module H is above, representing 1. The module L folds glider and the module H folds brick Hn.

3.3 Conformations of Module H

The half adder module H forms five different paths as shown in Fig. 3 in zig pass. Module H reads the already folded bead (F0 or F1) at above as one of the inputs, and determines the other input depending on the folding starts (top or bottom). Module H forms four bricks of H00, H01, H10, and H11 for each input. Then, the bricks outputs the sum as beads of bottom row and also outputs carry depending on whether module H is folded top or bottom except He1. Brick He1 is the first folded module H after overflow. Since this counter uses three row of gliders as a self-standing method, module H consisting of 30 beads has an even width, and when it starts folding from the bottom, folding ends at the bottom. That is why brick He1 ends folding at the bottom and outputs carry equal to 0.

In the output sum, module H has 2 types of beads representing "0" and 3 types of beads representing "1". Then, module F is folded to F0 and F1 in Fig. 2 according to the beads corresponding to 0 (H00, H11) and 1 (H01, H10, He1), respectively, to format beads representing a value.

3.4 Example of overflow

Oritatami system proceeds to further fold the counter transcript form the state shown in Fig. 7, it overflows (Fig. 8). If it does not overflow, the next module

L will bind to above Turn Signal and turn. However, since the carry overflows, module L's folding starts from bottom row and does not reach Turn Signal, so L forms a self standing glider shape, goes straight and extends the zig pass (Fig. 9). Then the folding of module H begins and this H also becomes a glider shape (Fig. 10). There is nothing above the folded module H (brick He1), but the original value is recognized as "0" and He1 catches the carry. However, the end of folding He1 is bottom row because He1 is glider, so for He1 only, the counter determins that the folding end position is bottom row as carry=0. In this way, the infinite counter extends the bit width after it overflows.

Subsequently, module R and F are folded, and then module L folding starts (Fig. 11). Since module L turns by binding to Turn Signal, it is necessary to give signal for turn in order to move to zag pass from here. Therefore, let the top of folded module F be Turn Signal. Normally, there are folded beads above module F, so module L can not bind to F's Turn Signal. However, only when it overflows, above F becomes blank, so L turns by being folded there (Fig. 12). The turned module L forms a Turn Signal at the end, and folding of zag pass starts.

3.5 Conformations of Module L

The left turn module L forms five different paths as shown in Fig 4. In normal zig zag folding, module L is a glider like Lt or Lbc, but when transcript of L binding to Turn Signal, it formed as a turn module like Ltrc or Ltre. Module L in the form as turn module has Turn Signal in its ends and it helps other L turn at the next zig-zag. In zig pass, when module L reaches the left end, if it does not overflow, it bind to this Turn Signal and turned. If module L overflows, it forms Lbe and continues the zig pass. Then, module L which is folded next to Lbe, forms Ltre and turns by binding to the top of module F that is just before it. At this time, the top of module F functions as Turn Signal, but this is the behavior only overflow because otherwise there is some folded module above module F.

4 The brick automaton

Beads which has already been folded around, that is, the environment decide which module transcript will be. The environmental transition is represented Fig. 13. In addition, each symbol written as a state transition is conformation such as Fig. **??**

References

1. Leonard Adleman, Qi Chang, Ashish Goel, and Ming-Deh Huang, Running time and program size for self-assembled squares, In Proc. STOC 2001, ACM, 740-748, 2001.

2. Nathaniel Bryans, Ehsan Chiniforooshan, David Doty, Lila Kari, and Shinnosuke Seki, The power of nondeterminism in self-assembly, *Theory of Computing*, vol. 9, 1-29, 2013.
3. Constantine Glen Evans, Crystals that Count! Physical Principles and Experimental Investigations of DNA Tile Self-Assembly, Ph.D. thesis, Caltech, 2014.
4. Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinnosuke Seki, Oritatami: A computational model for molecular co-transcriptional folding, *International Journal of Molecular Sciences* vol. 20(9), 2259, 2019. Its conference version was published in Proc. MFCS 2016.
5. Cody Geary, Paul W. K. Rothemund, and Ebbe S. Andersen, A single-stranded architecture for cotranscriptional folding of RNA nanostructures, *Science* vol 345(6198), 799-804, 2014.
6. Yusei Masuda, Shinnosuke Seki, and Yuki Ubukata, Towards the algorithmic molecular self-assembly of fractals by cotranscriptional folding, In Proc. CIAA 2018, LNCS 10977, Springer, 261-273, 2018.
7. C. Robertson McClung, Plant circadian rhythms, *The Plant Cell*, vol. 18, 792-803, 2006.
8. Marvin Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Inc., 1967.
9. Paul W. K. Rothemund and Erik Winfree, The program-size complexity of self-assembled squares (extended abstract), In Proc. STOC 2000, ACM, 459-468, 2000.
10. Erik Winfree, Algorithmic Self-Assembly of DNA, Ph.D. thesis, Caltech, 1998.
11. Cody Geary and Pierre-Étienne Meunier and Nicolas Schabanel and Shinonsuke Seki, Proving the Turing Universality of Oritatami Cotranscriptional Folding, Proc. ISAAC 2018, 23:1 – 23:13,2018.

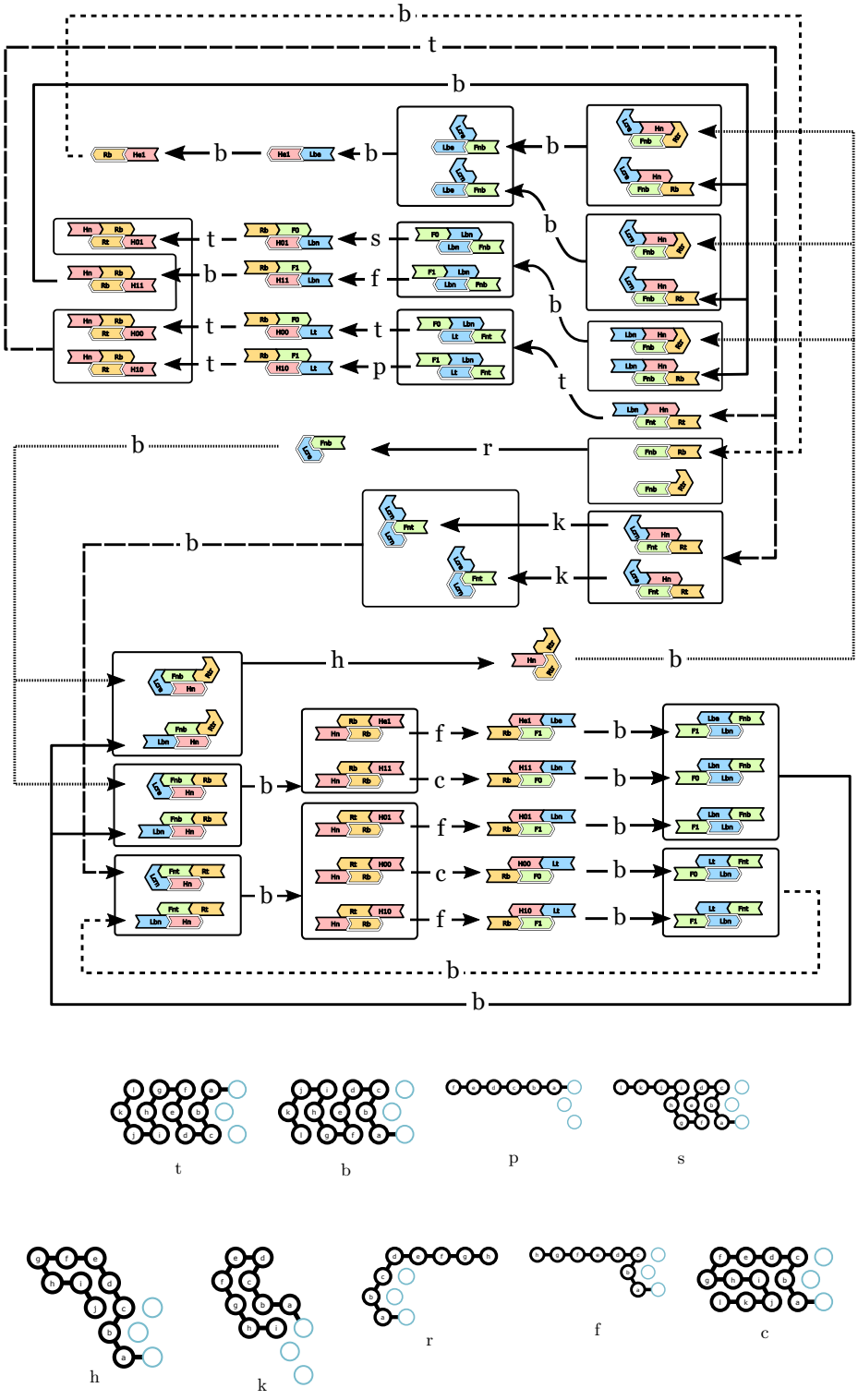

Fig. 13. The brick automaton of infinite binary counter