Proving the Turing Universality of Oritatami Co-Transcriptional Folding
Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, Shinnosuke Seki

To cite this version:
Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, Shinnosuke Seki. Proving the Turing Universality of Oritatami Co-Transcriptional Folding. 2017. hal-01567227

HAL Id: hal-01567227
https://hal.science/hal-01567227
Preprint submitted on 21 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract

We prove that the Oritatami model of molecular folding is capable of embedding arbitrary computations in the folding process itself, by a local energy optimisation process, similar to how actual biomolecules such as DNA or RNA fold into complex shapes and functions.

This result is the first principled construction in this research direction, and motivated the development of a generic toolbox, easily reusable in future work. One major challenge addressed by our design is that choosing the interactions to get the folding to react to its environment is NP-complete. Our techniques bypass this issue by allowing some flexibility in the shapes, which allows to encode different “functions” in different parts of the sequence (instead of using the same part of the sequence).

However, the number of possible interactions between these functions remains quite large, and each interaction also involves a small combinatorial optimisation problem. One of the major challenges we faced was to decompose our programming into various levels of abstraction, allowing to prove their correctness thoroughly in a human readable/checkable form. We hope this framework can be generalised to other discrete dynamical systems, where proofs of such large objects are often difficult to get.
Introduction

Molecular folding is the biological process that turns one-dimensional sequences into three-dimensional shapes. In the particular context of proteins and RNA, this process has attracted a lot of attention, as it could allow us to engineer our own molecules, and therefore to interact with biological functions. The potential applications range from using bacteria as computing devices or nano-factories, to producing targeted drugs to cure specific diseases with little to no side effects. More fundamentally, understanding “molecular programming” by engineering our own molecules will shed a new light on how these mechanisms, and evolution in particular, work in nature.

If we are to have such an engineering discipline crafting “computing molecules” with arbitrary shapes, we need a theory of these systems to inform of their capabilities and give hints for building actual molecules in the wet lab.

Unfortunately, we seem quite far from a full understanding of these mechanisms. From a practical perspective, the latest efforts to solve the protein design problem [19] are still quite far from a complete general methodology. From a theoretical perspective, it has been shown that, in different variants of the hydrophobic-hydrophilic (HP) model [6], the problem of predicting the most likely geometry (or conformation) of a sequence is NP-complete [20, 17, 2, 3, 5], both in two and three dimensional models. Approximation algorithms have also been developed [1, 16].

However, the effective speed of molecular folding in actual cells seems to contradict these hardness results. Moreover, its reliability and relative robustness to small changes in conditions or sequences seem to rule out approximations as well.

To understand these phenomena, two essential ingredients of molecular dynamics need to be considered: thermodynamics, which governs probability distributions over shapes in the long run, and kinetics, which is the step-by-step movements of molecules in solution. Some models of tile assembly, such as the abstract Tile Assembly Model [21, 18] chose to ignore thermodynamics and focused on kinetics, and got excellent results in the lab. Models of molecular folding, like the HP model, focus on hardness results, and for that reason ignore kinetics and work entirely on thermodynamics.

Our goal with Oritatami [11] is to try to understand the kinetics of folding, and in the future get a more complete picture including both aspects. The rationale of this choice is that the wetlab version of Oritatami already exists, and has been successfully used to engineer shapes with RNA in the wetlab [10]. The main feature of RNA that motivates our approach is the fact that RNA folds while being produced, which is known as co-transcriptional folding. This process has been shown to play an important role in the final shape of biomolecules [12], especially in the case of RNA [7].

1.1 Brief overview of the model

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

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

1.2 Main results

In this paper, we construct a “universal” set of 520 bead types, along with a single universal attraction rule for these bead types, with which we can simulate any tag system, and therefore any Turing machine $M$, within a polynomial factor of the running time $M$.

This construction motivated the development of a toolbox composed of two things: common structures that can react to their environment, and solutions to combine these structures into larger constructions.
Theorem 1. There is a finite universal set of beads $B$ and an attraction rule $\bowtie \subseteq B^2$ such that for any Turing machine $M$ running in time $t$ on input $x$ (where $t$ is possibly infinite), there is a seed structure $\sigma \in B^{x^2}$ of size $O(|x|)$, and a periodic sequence $w$ of beads from $B$ (with period of length $O(1)$) such that $w$ folds into a structure of size $O(t^3 \log t)$.

Our construction is composed of different modules, or subsequences, each building different “sub-shapes” of the global conformation. This result had to overcome a number of important challenges, presented in Sections 1.3 and 1.4.

1.3 Proving our designs

The main challenge we faced in this paper was the size of our constructions: indeed, while we developed higher-level geometric constructs to program useful shapes, there is a large number of possible interactions between all different parts of the sequence.

Getting solid proofs on large objects is a common problem in discrete dynamical systems, for instance on cellular automata [8, 4] or tile assembly systems [13]. In this paper, we introduce a general framework to deal with that complexity, and prove our constructions rigorously. This method proceeds by decomposing the sequence into different modules, and the space into different areas where exactly one module grows. We can then reason on the modules separately, and only deal with interactions at the border between all possible modules that can have a common border.

1.4 Design challenges

As shown in our previous results [9], the problem of choosing an attraction rule so that a single sequence folds into different shapes depending on its environment, is NP-complete. That problem is called the sequence design problem in [9].

In the present paper, since our sequence is periodic and has a small number of bead types, a single module can interact with a large number of other modules (including previous copies of itself).

We introduce a tool to cope with such situations, called socks. The goal is to “shift” the sequence, so that different parts interact with the various environments. Socks work as follows: whenever different copies of a single subsequence $s_i, s_{i+1}, \ldots, s_{i+k}$ (for some integer $i$, and with $k$ equal to a few dozens) have to interact with a large number of different unrelated environments (where an environment is a local configuration of beads), we reduced the number of environments by folding a small part of the molecule, before $i$, into a compact useless shape (with the shape of a “sock”), so that only a later part $s_j, s_{j+1}, \ldots$ with $j > i + k$ of the sequence interact with a subset of all environments.

This allows us to dispatch the different interactions to different parts of the sequence.

1.5 Relationship to other work

This construction generalises our previous results, where we built an arbitrary-width counter with a fixed periodic sequence [9]. In that result, all parts of the structure are densely packed into parallelograms, and these structures react to their environment by folding into a different Hamiltonian path in each parallelogram.

That result required tedious manual tweaking of the rule, so that different parts of the sequence interacted nicely with each other. Moreover, finding useful Hamiltonian paths is hard, which means that our techniques could not scale well.

In this paper, we solve these issues to a large extent, using the toolbox we introduce in Section 6. Note that the dynamics used is slightly changed compared to [9]. We believe the dynamics used here to be more intuitive, and our previous negative results (NP-completeness of rule design) still hold.

---

1 The constants in the $O(\cdot)$s only depend on the size of the simulated Turing machine.
2 Definitions and Preliminary Results

The empty word is denoted by $\varepsilon$. For $1 \leq i \leq j \leq n$, by $w[i..j]$, we refer to the factor $w_iw_{i+1}\cdots w_j$ of $w$.

2.1 Oritatami Systems

Let $B$ be a finite set of bead types. A conformation $c$ of a bead sequence $w \in B^* \cup B^\mathbb{N}$ is a directed self-avoiding path in the triangular lattice $\mathbb{T}$, where for all integer $i$, vertex $v_0$ of $c$ is labelled by $w_i$. $v_i$ is the position in $\mathbb{T}$ of the $(i+1)$th bead, of type $w_i$, in conformation $c$. A partial conformation of a sequence is a conformation of a prefix of $w$.

For any partial conformation $c$ of some sequence $w$, an elongation of $c$ by $k$ beads (or $k$-elongation) is a partial conformation of $w$ of length $|c| + k$. We denote by $C_w$ the set of all partial conformations of $w$ (the index $w$ will be omitted when the context is clear). We denote by $E(c, k)$ the set of all $k$-elongations of a partial conformation $c$ of a sequence $w$.

Oritatami systems. An Oritatami system $O = (w, \bullet, \delta, \sigma)$ is composed of (1) a (possibly infinite) bead sequence $w$, called the primary structure, (2) an attraction rule, which is a symmetric relation $\bullet \subseteq B^2$, (3) a parameter $\delta$ called the delay time and (4) $\sigma$, an initial conformation of $w$, called the seed. $O$ is said periodic if $w$ is infinite and its suffix $w_{|\sigma|}w_{|\sigma|+1}\cdots$ is a periodic bead sequence. Periodicity ensures that the “program” embedded in the oritatami system is finite (does not hardcode any specific behavior) and at the same time allows arbitrary long computation.

We say that two bead types $a$ and $b$ attract each other when $a \bullet b$. Furthermore, given a conformation $c$ of $w$, we say that there is a bond between two adjacent positions $c_i$ and $c_j$ of $c$ in $\mathbb{T}$ if $w_i \neq w_j$ and $|i-j| > 1$. The number of bonds of conformation $c$ of $w$ is denoted by $H(c) = |\{(i, j) \in \mathbb{Z}^2 : c_i \sim c_j, j > i + 1, \text{ and } w_i \neq w_j\}|$.

Oritatami dynamics. The folding of an oritatami system is controlled by the delay time $\delta$. Informally, the conformation grows from the seed conformation, one bead at a time. This new bead adopts the position(s) that maximise the potential number of bonds the conformation can make when elongated by $\delta$ beads in total. This dynamics is oblivious as it keeps no memory of the previously preferred positions; it differs thus slightly from the hasty dynamics studied in [11]; it might also be considered as closer to experimental conditions.

Formally, given an Oritatami system $O = (p, \bullet, \delta, \sigma)$, we consider the dynamics $D : 2^\ell \to 2^\ell$ that maps every subset $S$ of partial conformations of length $\ell$ of $w$ to the subset $D(S)$ of partial conformations of length $\ell + 1$ of $w$ as follows:

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

The possible conformations at time $t$ of the Oritatami system $O$ are the elongations of the seed conformation $\sigma$ by $t$ beads in the set $D^t(\{\sigma\})$.

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

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

In this article, we will only consider deterministic periodic Oritatami systems with delay time $\delta = 3$.

---

2 The triangular lattice is defined as $\mathbb{T} = (\mathbb{Z}^2, \sim)$, where $(x, y) \sim (u, v)$ if and only if $(u, v) \in \{(x - 1, y), (x + 1, y), (x, y + 1), (x, y - 1), (x + 1, y + 1), (x - 1, y - 1)\}$. Every position $(x, y)$ in $\mathbb{T}$ is mapped in the euclidean plane to $x \cdot X + y \cdot Y$ using the vector basis $X = (1, 0)$ and $Y = \text{rotation} -120^\circ(X)$. 
2.2 Skipping Cyclic Tag systems

In the next sections, we demonstrate the existence of a single periodic primary structure that can simulate any Turing computation. Precisely, our construction simulates the following particular type of tag systems which are known to simulate in $O(T^2 \ln T)$ steps any Turing machine running in $T$ steps [22].

Skipping Cyclic Tag systems A skipping cyclic tag system consists of a set of $n$ productions $p_0, \ldots, p_{n-1} \in \{0,1\}^*$ and an initial word $u^0 \in \{0,1\}^*$. At each time step, the tag system cycles through the productions and decides to append the current production or not according to the letter read. We denote by $u^t$ the word at time $t$. Formally, at time $t = 0$, its pointer $q^0$ is set to 0. At all time $t$,

**Halting step:** If $u^t$ is the empty word $\epsilon$, then the tag system halts and outputs $q^t$; otherwise

**Deletion step:** If the first letter $u_0^t$ of $u^t$ is $\emptyset$, then set $q^{t+1} := (q^t + 1) \mod n$ and $u^{t+1} := u_1^t \cdots u_{|u^t|}^t$,

the suffix of $u^t$ without its first letter; finally

**Appending step:** if $u_0^t = 1$, then the tag system appends the next production to $u^t$ and skips to the following production, i.e.: $u^{t+1} := u_1^t \cdots u_{|u^t|}^t \cdot p_{q^{t+1} \mod n}$ and set $q^{t+1} := (q^t + 2) \mod n$.

For instance, the skipping tag system $p = (11\emptyset, \epsilon, 11, \emptyset)$ has the following execution $(u^t)^t$ on input word $u^0 = 01\emptyset$:

\[
\begin{align*}
0 \emptyset 1 \emptyset &\rightarrow [1] 1 \emptyset \quad \text{Append} \quad [2] 11 \\
[3] 0 \emptyset 1 1 &\rightarrow [1] 11 \quad \text{Append} \quad [2] 1 \\
[3] 0 \emptyset &\rightarrow [1] \emptyset \quad \text{Append} \quad [3] \emptyset
\end{align*}
\]

and outputs thus production $p_1 = \epsilon$.

Annotated trimmed space-time diagram. Given a SCTS $(p_0, \ldots, p_{n-1}; u^0)$, we denote by $0 \leq t_1 < t_2 < \cdots$ all the times $t$ such that the word $u_t$ starts with letter 1 and set $t_0 = -1$ by convention. Let us now compress the deletion steps occurring during steps $t_i + 1$ and $t_{i+1} - 1$ by simply indicating in exponent the production index for each deleted letter:

\[
\begin{align*}
0 \emptyset &\rightarrow [1] 1 \emptyset \quad \text{Append} \quad [2] 11 \\
[3] 0 \emptyset &\rightarrow [1] 11 \quad \text{Append} \quad [2] 1 \\
[3] 0 &\rightarrow [1] \emptyset \quad \text{Append} \quad [3] \emptyset
\end{align*}
\]

and align the resulting words in a 2D diagram according to their common parts:

![Annotated trimmed space-time diagram](image)

we obtain the annotated trimmed space-time diagram for the SCTS $(p, 01\emptyset)$. The following lemma gives a formal definition:

**Lemma 2.** The annotated word on row $i$ (indexed from $i = 0$) of the annotated trimmed space-time diagram is: (the production indices in exponent are computed modulo $n$)

- if $u^{t_i+t_i} = 0^* 1 \cdot s$ for some $r \geq 0$ and $s \in \{0,1\}^*$: then, $r = t_{i+1} - t_i - 1$ and the annotated word on row $i$ is $[i+1+t_i]_0 \cdots [i-1+t_{i+1}]_0 [i+t_{i+1}]_1 \cdots s$ whose first letter is placed in column $t_i+1$ (where the leftmost column is indexed by 0);

- if $u^{t_i+t_i} = 0^r$ for some $r > 0$: then, row $i$ is the last row of the diagram and its annotated word is $[i+1+t_i]_0 \cdots [i+r+t_i]_0 [i+t_i+r+1]$ and starts at column $t_i+1$.

**Proof.** Observe that $q^t = i + t_i \mod n$ as exactly $t_i$ letters have been read and $i$ appending steps occurred before reading the $i$-th 1. △
Finally, we use a result by Cook [4], and Neary and Woods [22, 14] to show that simulating a skipping cyclic tag system is sufficient to simulate universal Turing machines efficiently:

**Lemma 3.** Let \( M \) be a Turing machine running in time \( t \). There is a skipping cyclic tag system \( S \) simulating \( M \) in \( O(t^2 \log t) \) steps. Moreover, the number of productions of \( S \) is a multiple of 4.

**Proof.** The original cyclic tag system by Cook [4] differs from the skipping cyclic tag system only in that in the original, the list rotates by 1 no matter which letter the current word begins with. By a classical result about tag systems, 2-tag system with \( m \) productions (i.e. over an \( m \) letter alphabet) can be simulated by a cyclic tag system with \( 2m \) productions: that simulation works by encoding the \( m \) letters as \( 10^{m-1}, 010^{m-2}, 0210^{m-3}, \ldots, 0^{m-1}1 \), respectively. We can in turn simulate a cyclic tag system with \( n \) productions \( p_0, p_1, \ldots, p_{n-1} \), starting from an input \( u \), by a skipping cyclic tag system with \( 2n \) productions \( \varepsilon, f(p_0), \varepsilon, f(p_1), \varepsilon, \ldots, f(p_{n-1}) \), starting from the word \( f(u) \), where \( f \) is the automorphism over \( \{0,1\}^* \) defined as \( f(0) = 00 \) and \( f(1) = 1 \).

Finally, the result by Neary and Woods [15, 22] on cyclic tag systems implies that 2-tag systems can simulate \( t \) steps of a Turing machines in \( O(t^2 \log t) \).

### 3 Proof structure

Figure 1 shows the global structure of our construction: at the abstract level (Section 4), we will show how the geometrical arrangement of big blocks (regions of the plane) simulates a tag system. The construction then becomes local: we only have to construct a molecule that correctly folds into the blocks, and interacts with neighbouring blocks as planned.

In Section 5, we will introduce modules (parts of the sequence), functions (possible conformations of a module in response to the environment in which it is folded), and bricks (partial conformations contained inside a block). Finally, in Section 7, we will show the actual sequences and attraction rule that implement all bricks, and show that our choice of geometric parameters guarantees that important parts of the sequence always fold in the same environments, in all possible conformations and inputs.

Section 8 gives a proof that the bricks actually implement the blocks, and shows how we verified the assembly level using a program on a specific tag system that exhibited all possible interactions between modules. This last step of our proof will follow the following steps:

1. Enumerate all the surroundings for each brick of each module
2. Enumerate all possible modules following the module
3. Generate automatically human-readable certificate of the correctness of the folding for each possibility, in the form of proof trees.
4. In the few cases where the surrounding may vary, prove that it has no incidence in the folding of the brick.

![Figure 1 Programming framework.](image-url)
4 Simulating Tag Systems with Blocks

This section presents the simulation logic and the global geometry, without any folding. The simulation is decomposed into blocks, which are regions of the plane.

In the simulation, these blocks only interact at their border. In Section 5, we will show how to implement each block, and interactions between blocks, using actual folding. The names of blocks, introduced here, will be the same between the different parts of the paper.

4.1 The Blocks

4.1.1 Overview

Our simulation proceeds by mimicking the annotated trimmed space-time diagram of the skipping cyclic tag system to be simulated. The folding walks to the South, i.e. each new step is below the previous step.

At each step, the simulation starts in a general movement from left to right. Leading zeros are trimmed off, and the simulation halts if the remaining word is empty. If the remaining word is not empty, there is at least one 1 in the word. The simulation removes the leading 1, skips over (and copies) the rest of the word, and appends the relevant production at the end of the word. Finally, the sequence is folded again in the opposite direction (i.e. right to left), and copies the computed word for the next step to start. See Figure 2 for an example.

![Figure 2](image-url) Execution of the block automaton simulating the SCTS \((p = (\epsilon, 100, 1.0); u^0 = 010)\). Every other row is shaded in blue. Each row encodes one step of the tag system, and each row is divided into a “zig part” (on top) and a zag part (on the bottom). The word at the end of each step can be read at the bottom of the zag part for the corresponding row. In this figure, 1 is pictured as a flat border with a red rectangle and 0 is pictured as a red spike with a red circle. Green and cyan ovals mark the presence of a letter, on zig and zag rows respectively.
4.1.2 Types of blocks

There are 10 different types of blocks, shown in Figure 31. All of them are used in our example in Figure 2:

- **Seed** encodes the initial word into a conformation of beads where 0s are represented as red-dot spikes and 1s as red rectangles.
- **Read0** and **Read1** are responsible for reading and copying the letters of the current word during the zig phase (left to right). When clear from the context, we will sometimes refer to both **Read0** and **Read1** collectively as **Read**.
- **Copy0** and **Copy1** are responsible for copying the letters of the new word over to the next step, during the zag pass (right to left). When clear from the context, we will sometimes refer to both **Copy0** and **Copy1** collectively as **Copy**.
- **AppEnd & C2** are responsible for appending a production, carriage return and line-feed.
- **Halt** is the last block produced corresponding to the halt of the simulated tag system.

4.1.3 The block automaton

A visual depiction of the logic is shown in Figure 3 in the form of an automaton: starting from the seed block, blocks attach at the orange anchors (and one next to the other as described by the block automaton in Figure 3. Each block is labelled with the current production index of the tag system which determines the production to be appended. An example of execution of the block automaton for the SCTS $(p = \langle 110, \epsilon, 11, 0 \rangle; u^0 = \langle 0 \rangle)$ is illustrated in Figure 2.

4.1.4 Simulating tag systems with blocks

Let $S = (p_0, p_1 \ldots p_n; u^0)$ be a skipping cyclic tag system, and for all integer $i \geq 0$, let $t_i$ be the $i$th step where $u^i$ starts with 1 (starting from 0, i.e. $t_0$ is the first step where $u^0$ starts with 1). The following lemma describes the encoding of $S$ into blocks (i.e. generalises Figure 2 to arbitrary tag systems).

**Lemma 4.** The (possibly infinite) final block configuration consists of: (see illustration on Figure 6)

- The seed row consists of the block $Seed(\langle u^0 \rangle)$ anchored at its end point at coordinates $(0, 1)$.
- For $i \geq 0$, the $i$-th row consists of a zig row anchored at height $Y = 2i + 1$, and a zag row anchored at height $Y = 2i + 1$ defined as follows:

  - (Compute) if $u^{i+1} = 0^*1 \cdot s$ and if $s$ and $p_{1+i+t_{i+1}}$ are not both $\epsilon$: then $r = t_{i+1} - t_i - 1$ and, as illustrated in Figure 4(b) and Figure 4(c):
    - the $i$-th zig-row, growing from left to right, contains the sequence of annotated blocks located at the following coordinates (with respect to their anchor point, shown in Figure 31):

    \[
    \begin{array}{cccccc}
    Y & 2i + 1 & X & i + 1 + t_i & \cdots & i + r + t_i & i + t_{i+1} & \cdots & i + |s| + t_{i+1} & i + |s| + 1 + t_{i+1} \\
    \text{Blocks} & \text{Read0} & \cdots & \text{Read1} & \text{Copy}(v_0) & \cdots & \text{Copy}(v_{|s|-1}) & \text{AppEnd & C2} & \text{AppEnd & C2} \\
    \{g\} & [i + 1 + t_i] & \cdots & [i + r + t_i] & [i + t_{i+1}] & \cdots & [i + |s| + t_{i+1}] & [i + |s| + 1 + t_{i+1}] & [i + 1 + t_{i+1}] \\
    \end{array}
    \]

    - the $i$-th zag-row consists of the sequence of blocks located at the following coordinates (with respect to their anchor point, see Figure 31):

    \[
    \begin{array}{cccccc}
    Y & 2i + 1 & X & i + 3 + t_{i+1} - \Delta & \cdots & i + 4 + t_{i+1} - \Delta & \cdots & i + 2 + |s| + t_{i+1} - \Delta & \cdots & i + 2 + t_{i+1} \\
    \text{Blocks} & \text{CopyLineFeed}(v_0) & \cdots & \text{Copy}(v_1) & \cdots & \text{Copy}(v_{|s|-1}) & \cdots & \text{Copy}(v_{|s|-1}) \\
    \{g\} & [i + 2 + t_{i+1}] & \cdots & [i + 2 + t_{i+1}] & \cdots & [i + 2 + t_{i+1}] & \cdots & [i + 2 + t_{i+1}] \\
    \end{array}
    \]

    where \( v = u^{i+1+t_{i+1}} = s \cdot p_{1+i+t_{i+1}} \neq \epsilon \) (as $s$ and $p_{1+i+t_{i+1}}$ are not both $\epsilon$).

  - (Halt) if $u^{i+1} = 0^*1$ and $p_{1+i+t_{i+1}} = \epsilon$: then $r = t_{i+1} - t_i - 1$ and the last rows of the block configuration consist from left to right in the sequence of annotated blocks located at the following
Figure 3 The block automaton.
Here, we understand “align” both as “align in the plane” and “adjust the length of sequences to match modulo common parameters”.

<table>
<thead>
<tr>
<th>X</th>
<th>(i + 1 + t_i)</th>
<th>(i + r + t_i)</th>
<th>(i + l + t_i)</th>
<th>(i + 2 + t_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>(i + 1 + t_i)</td>
<td>(i + r + t_i)</td>
<td>(i + l + t_i)</td>
<td>(i + 2 + t_i)</td>
</tr>
<tr>
<td>([q])</td>
<td>([i + 1 + t_i])</td>
<td>([i + r + t_i])</td>
<td>([i + l + t_i])</td>
<td>([i + 2 + l + t_i])</td>
</tr>
</tbody>
</table>

- **Finally**, (Halt 2) if \(u^{1+t_i} = \theta^n\) for some \(r \geq 0\): then the \(i\)-th zig-row is last row of the block configuration and consists of the sequence of annotated blocks located at the following coordinates, as illustrated in Figure 4(e):

\[
\begin{array}{c|c|c|c}
Y & 2i + 1 & 2i & 2i + 3 \\
\hline
X & i + 1 + t_i & i + r + t_i & i + l + t_i \\
Blocks & \text{Read}\text{\textcircled{B}} & \text{Read}\text{\textcircled{B}} & \text{CarriageReturn\&LineFeed} \\
q & [i + 1 + t_i] & [i + r + t_i] & [i + l + t_i] \\
\end{array}
\]

**Proof.** This follows from an easy induction on the number of rows. The induction hypothesis at step \(i\) is that the word encoded by the blocks at the bottom of the zag-row \((i - 1)\) is \(u^{1+t_i}\) and the production index in this zag row is \(q^{1+t_i} = 1 + i + t_i \pmod{n}\).

First, since we choose the conformation of the seed, we choose the encoding of the initial word in the tag system. Then, showing that if the induction hypothesis holds at step \(i\), it also holds at step \(i + 1\), follows from the case enumeration in Figure 6 and the block automaton in Figure 3.

### 4.2 General geometry of the Blocks

The precise geometry of each block is given by the figures 5 and 32–39. We begin by introducing a number of parameters we will use to align bricks properly\(^3\) for all possible tag systems and inputs.

We first define the write position of a block the position on its border where its value is “written”, i.e. as either a spike (\(\theta\), red circles on Figure 4) or a dent (1, red squares on Figure 4). Similarly, read positions are positions where the shape of the folding depends on whether there is a spike or a dent on the adjacent block. See Figure 5 for an illustration.

Starting from a skipping cyclic tag system \(\mathcal{S}\), we first build a tag system \(\mathcal{T}\) by turning \(\mathcal{S}\) into a skipping cyclic tag system such that \(n\), the number of productions of \(\mathcal{T}\), is a multiple of 4, and moreover \(n \geq 8\). We build \(\mathcal{T}\) by duplicating all the productions of \(\mathcal{S}\) and all the \(\theta\)s in all productions of \(\mathcal{S}\) until \(4\mid n\) and \(n \geq 8\).

- \(n\) is the number of productions in \(\mathcal{T}\), hence \(n\) is a multiple of 4, and \(n \geq 8\).
- \(L\) is the length of the longest production in \(\mathcal{T}\).
- \(P\) is the length of an extra padding on each production. We let \(P = 11 + (L \mod 2)\), hence \(L + P\) is even.
- \(w\) is an atomic width we need to define other constants \(W\) and \(h\). For now, let \(w = 6(L + P) + 18\). We will later use the fact that \(w \mod 12 = 6\).
- \(W\) is the width of the \text{Copy}\text{\textcircled{B}} and \(<\text{Copy}\text{\textcircled{B}} blocks.
  
  Let \(W = n \cdot (w + 6)\). We will later use the fact that \(W \mod 48 = 0\) (because \(n\) is a multiple of 4 and \(w + 6\) is a multiple of 12).
- \(h\) is the height of the \text{Read}\text{\textcircled{B}}, \text{Copy}\text{\textcircled{B}}, and \(<\text{Copy}\text{\textcircled{B}} blocks, not counting the small bumps.
  
  Let \(h = W - (w + 3)\). Note that \(h \mod 12 = 3\).

We can now translate Lemma 4, to give blocks their actual coordinates in the simulation:

---

\(^3\) Here, we understand “align” both as “align in the plane” and “adjust the length of sequences to match modulo common parameters”.

(a) The seed row anchored at coordinates $(0, 0)$.

(b) The case where $u^{1+i} = \emptyset^r 1 \cdot s$ and the production to be appended is $p_{1+i+i+1} \neq \epsilon$.

(c) The case where $u^{1+i} = \emptyset^r 1 \cdot s$ with $s \neq \epsilon$ and the production to be appended is $p_{1+i+i+1} = \epsilon$.

(d) The case where $u^{1+i} = \emptyset^r 1$ and the production to be appended is $p_{1+i+i+1} = \epsilon$.

(e) The case where $u^{1+i} = \emptyset^r$.

Figure 4 The $i$-th row of the final block configuration (the previous and next rows are shaded in blue). Production index in the label are computed modulo $n$. Observe that the Read and Copy in the $i$-th zig row correspond readily the $i$-th line in the annotated trimmed space-time diagram of the simulated SCTS.
Figure 5 Geometry of the Read blocks. Note that the internal structures (the lines in white) of both blocks Read0 and Read1 agree until position \((w + 2, -h + 1)\) where the presence or absence of a spike, encoding a \(0\), at the bottom of the row above forces the block to adopt the shape Read0 or Read1 respectively.

(a) The Read0 block has the shape of a trapezium whose bottom basis has length \(W\) and top basis has length \(w + 5\), with height \(h\). It has a dent (an empty position) located at \((w + 2, -h + 1)\) (w.r.t. to its origin at the bottom left corner), in which plugs the spike of the block from the row above it, encoding the letter \(0\). The next block will start folding at the bottom right corner, at \((W, 0)\).

(b) The Read1 block has the shape of a parallelogram with horizontal side length \(W\) and vertical side length \(h\). The red rectangle area at position \((w + 2, -h + 1)\) (w.r.t. its origin at the bottom left corner) aligns with the flat bottom block above encoding the letter \(1\) (as opposed to a spiked-block encoding a \(0\)). The next block will start folding at the top right corner, at \((W - 1, -h + 1)\).
Lemma 5. The (possibly infinite) final block configuration consists of: (see illustration on Figure 6)

The seed row, i.e., the block \( \text{Seed}(u^0) \) ending at coordinates \((-1, h)\).

For \( i \geq 0 \), the \( i \)-th row consists of a zig row located between \( y = 2ih + 1 \) and \( y = (2i + 1)h \), and a zag row located between \( y = (2i + 1)h + 1 \) and \( y = 2(i + 1) \), defined as follows:

- (Compute) if \( u^{1+t_i} = \emptyset \cdot 1 \cdot s \) and if \( s \) and \( p_{1+i+t_{i+1}} \) are not both \( \epsilon \) then \( r = t_{i+1} - t_i - 1 \) and, as illustrated in Figure 6(a) and Figure 6(b):
  - the \( i \)-th zig-row consists from left to right of the sequence of geometrical blocks whose origin is located at the following coordinates:

\[
\begin{align*}
\sqrt{y} & \rightarrow x \quad \begin{array}{c}
(2i + 1)h \\
\vdots
\end{array} \quad \begin{array}{c}
h + (1 + t_i)W - t_i + 1 \\
\vdots
\end{array} \\
\text{Blocks} & \quad \text{ReadBlock} \quad \text{ReadBlock} \quad \text{ReadBlock} \quad \text{CopyBlock} \quad \text{CopyBlock} \quad \text{Append \& CR[|p_{1+i+t_{i+1}}|]}
\end{align*}
\]

This row ends at position \( ((i + 1)h + 1) + |p_{i+1+t_{i+1}}| + t_{i+1})W - 7, (2(i + 1) + 1)h \).

- the \( i \)-th zag-row consists from left to right of the sequence of geometrical blocks whose origins are located at the following coordinates:

\[
\begin{align*}
\sqrt{y} & \rightarrow x \quad \begin{array}{c}
(i + 1)h + 1 \\
\vdots
\end{array} \\
\text{Blocks} & \quad \text{CopyLineFeed}(t_0) \quad \text{CopyLineFeed}(v_1) \quad \text{CopyLineFeed}(v_{i-1})
\end{align*}
\]

where \( v = u^{1+t_i+1} = s \cdot p_{i+1+t_{i+1}} \neq \epsilon \) (as \( s \) and \( p_{i+1+t_{i+1}} \) are not both \( \epsilon \)). This row ends at position \( ((i + 1)h + 1) + (1 + t_{i+1})W - 1, (2(i + 1) + 1)h \).

- (Halt) if \( u^{1+t_i} = \emptyset^*1 \) and \( p_{1+i+t_{i+1}} = \epsilon \) then \( r = t_{i+1} - t_i - 1 \) and the last rows of the geometrical block configuration consist from left to right of the sequence of geometrical blocks located at the following coordinates, as illustrated in Figure 6(c):

\[
\begin{align*}
\sqrt{y} & \rightarrow x \quad \begin{array}{c}
(2i + 1)h \\
\vdots
\end{array} \\
\text{Blocks} & \quad \text{ReadBlock} \quad \text{ReadBlock} \quad \text{ReadBlock} \quad \text{CarriageReturn} \quad \text{LineFeed}
\end{align*}
\]

Finally, (Halt 2) if \( u^{1+t_i} = \emptyset^* \) for some \( r \geq 0 \) then the \( i \)-th zig-row is last row of the geometrical block configuration and consists of the sequence of geometrical blocks located at the following coordinates, as illustrated in Figure 6(d):

\[
\begin{align*}
\sqrt{y} & \rightarrow x \quad \begin{array}{c}
(2i + 1)h \\
\vdots
\end{array} \\
\text{Blocks} & \quad \text{ReadBlock} \quad \text{ReadBlock} \quad \text{Halt}
\end{align*}
\]

Hence, the read positions and write positions of blocks in consecutive rows are adjacent.

Proof. We map each block from Lemma 4 to its actual position, using the following table to compute the space taken by each block:

<table>
<thead>
<tr>
<th>Block</th>
<th>( \Delta x )</th>
<th>( \Delta y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadBlock</td>
<td>( W )</td>
<td>0</td>
</tr>
<tr>
<td>Read1</td>
<td>( W - 1 )</td>
<td>( 1 - h )</td>
</tr>
<tr>
<td>CopyBlock and Copy1</td>
<td>( W )</td>
<td>0</td>
</tr>
<tr>
<td>Append &amp; CR(</td>
<td>u</td>
<td>)</td>
</tr>
<tr>
<td>\text{CopyBlock and Copy1}</td>
<td>(-W)</td>
<td>0</td>
</tr>
<tr>
<td>\text{CopyLineFeed0} and \text{CopyLineFeed1}</td>
<td>(-W + 8)</td>
<td>(2h - 1)</td>
</tr>
</tbody>
</table>

Corollary 6. The geometrical blocks simulate the associated skipping cyclic tag system.
XX:14 Proving the Turing Universality of Oritatami Co-Transcriptional Folding

(a) The case where $u^{1+t_i} = 0 \cdot 1 \cdot s$ and the production to be appended is $p_{1+t_i+t_{i+1}} \neq \epsilon$.

(b) The case where $u^{1+t_i} = 0 \cdot 1 \cdot s$ with $s \neq \epsilon$ and the production to be appended is $p_{1+t_i+t_{i+1}} = \epsilon$.

(c) The case where $u^{1+t_i} = 0 \cdot 1$ and the production to be appended is $p_{1+t_i+t_{i+1}} = \epsilon$.

(d) The case where $u^{1+t_i} = 0$.

Figure 6 The $i$th row of the final geometrical block configuration (the previous and next rows are shaded in blue). Production index in the label are computed modulo $n$. Observe that the $\text{Read} \uparrow$ and $\text{Copy} \uparrow$ in the $i$-th zig row correspond readily the $i$-th line in the annotated trimmed space-time diagram of the simulated SCTS.
4.3 Production segments: encoding the production index

The primary structure we use to simulate a skipping cyclic tag system with \( n \) productions \( p_0, p_1, \ldots, p_{n-1} \), is a periodic sequence of \( n \) strings of beads of equal length called *production segments* \( [p_0], \ldots, [p_{n-1}] \), where for all \( i \), \( [p_i] \) encodes production \( p_i \). The first module of production segments is written as black lines on the figures in Section B.

The primary sequence of the oritatami system corresponding to the skipping cyclic tag system with productions \( \langle p_0, \ldots, p_{n-1} \rangle \) is the infinite sequence with period \( [p_0] \cdot [p_1] \cdot \cdots \cdot [p_{n-1}] \).

Each block is the result of folding a number of production segments (depending on the block type):

- Read\(\uparrow\) and Append\&CR\(\uparrow\)(\(u\)) take one production segment each,
- Halt stops before folding one full production segment,
- all other blocks take \( n \) production segments each.

We call the internal state of a block \( B \) the production index \( q \) of the first (and possibly only, for Read\(\uparrow\), Append\&CR\(\uparrow\)(\(u\)) and Halt blocks) production segment \( [p_q] \) of \( B \).

\[\blacktriangleright\text{Lemma 7. At each step, the internal state of every block is equal to the state variable } q \text{ in the block automaton in Figure 3.}\]

Therefore, the block automaton simulates exactly the SCTS.

\[\blacktriangleright\text{Proof. In the construction of our blocks, the internal state is increased by one (modulo } n \text{ each time a block consisting of one production segment is folded (Read\(\uparrow\) or Append\&CR\(\uparrow\)), and is unchanged (modulo } n \text{ otherwise (Copy\(\uparrow\), \text{□Copy or } \text{□CopyLineFeed}). The case of Halt, which stops the entire simulation, is special. This is exactly the same as in the block automaton (Figure 3). Moreover, the zag phase contains only blocks of } n \text{ productions segments (i.e. of width } W \text{), hence does not change the internal state, again as in the block automaton.}\]

\[\blacktriangleright\]

5 The Structure of the Sequence: the Modules and the Bricks

5.1 Modules

Each production segment is split into seven *modules* \( \text{A} \ldots, \text{G} \), each serving one or several purposes:

- **Module A** (3\(h\) – 2 beads long) is the initial scaffold upon which the other modules fold.
- **Module B** (5 beads long) is responsible for the detection of an empty tape word: if it is empty, it folds to the left and the molecule gets trapped in a closed space and the computation halts; otherwise, it folds to the right and the computation continues.
- **Module C** (3\(h\)–10 beads long) is responsible for the detection of the end of the tape word to start appending to it the production word.
- **Modules D0/1** (3\(W\) + 30 beads long each) encodes each letter of the production word inside the production segment. It adopts two shapes: compact inside reading and copying blocks, or expanded in appending blocks.
- **Module E** (3\(W\)(\(L\) – \(a\) + \(P\)) + 8\(h\) – 1 beads long) ensures by padding that all production segment have the same length (even if the production word have different length). It serves two other purposes: its presence indicates to \( \text{G} \) and \( \text{B} \) that the end of tape is not yet reached; and it accomplishes the carriage return initiating the Zag-phase once the current production word has been appended.
- **Module F** (4\(h\) beads long) is the scaffold upon which \( \text{G} \) folds. It is specially designed to induce two very distinct shapes on \( \text{G} \) depending on the initial shift of \( \text{G} \).
- **Module G** (6\(h\) – 1 beads long) is the real “brain” of the molecule. It implements three distinction functions which are triggered by its interaction with its environments: in the zig-up phase, it reads the current letter of the tape word, ignoring the \( \emptyset \)s and moves to the zig-down phase when it reads a 1; it copies...
the 0 and 1 in the zig-down and zag phases; it accomplishes the line feed when the molecule reaches the beginning of the tape word at the end of each zag-phase.

The bead-by-bead description of each of these sequences will be given in Section 7.

Each production segment \([p_i]\) is split into a sequence of modules:
\[
D(p_i)_0 \cdot \ldots \cdot D(p_i)_1 \cdot E(p_i)_1 \cdot F \cdot G
\]
where \(\cdot\) denotes the concatenation of two bead sequences.

### 5.2 Bricks

**Definition 8 (Brick).** Each module adopts different conformations to accomplish each of its tasks. We call *brick* every conformation that a given module adopts when folded in a valid environment.

Figure 7 lists all the bricks that adopt the modules in our design and how they are organized inside each block. The exact geometry of the bricks will be given together with their beads sequence in Section 7. Bricks are the lowest design level we will consider in this article before going to the beads level. Figure 8 presents the brick automaton which details how the bricks articulate with each other. The lemma below shows that if the modules folds into bricks according to this automaton, then our design simulates indeed the block automaton and thus the SCTS. We are left with proving that each module folds into the expected brick for every possible environment to complete the proof of our main theorem.

**Lemma 9.** Starting from a wellformed seed (see section 7), the brick automaton in Figure 8 simulates the Skipping Cyclic Tag System.

**Proof sketch.** Starting from a wellformed seed, we prove by induction that the brick automaton implements precisely the block automaton which simulates the SCTS by Lemma 7.

We are left with designing sequences implementing the bricks. We can forget about the simulation itself and focus on the local folding of each module in every possible environment.

### 6 Design Toolbox

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

#### 6.1 Expanding shapes: Glider and Switchback

In our design we need to store many letters in a very compact space inside the blocks Read\(\uparrow\), Copy\(\uparrow\) and \(\downarrow\)Copy, and to expand each of them to the width of a block in the Append\&CR\(\uparrow\) blocks. This is achieved using the glider/switchback device illustrated in Figure 9. The key in this design is that both shapes use a small enough number of bonds so that they don’t interfere once the beginning of the molecule is folded in one way, it keeps folding that way. The design of modules \(D\), \(E\) and \(G\) is based on this bonding pattern (see Section 7). This behavior is best observed in the proof-trees (see Section 8.2 or the companion website of this paper\(^4\)).

#### 6.2 Implementing the logic

As in [11], the internal state of our “molecular computing machinery” consists essentially of two parameters: 1) the *position inside the primary structure* of the part currently folding; and 2) the *entry point* of molecule inside the environment. Indeed, depending on the entry point or the position inside the primary structure, different beads will be in contact with the environment and thus different “functions” will be applied as a

\(^4\) https://www.irif.fr/ nschaban/oritatami/prooftrees/
<table>
<thead>
<tr>
<th></th>
<th>Zig-Up</th>
<th>Zig-Down</th>
<th>Write</th>
<th>◄Zag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Halt</td>
<td>Read△</td>
<td>Copy△</td>
<td>Append &amp; CR</td>
<td>◄Copy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CarriageReturn &amp; LineFeed &amp; Halt</td>
<td>◄Copy1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CopyLineFeed0</td>
<td>◄CopyLineFeed1</td>
</tr>
<tr>
<td>A</td>
<td>A△ZigUp (see Figure 13)</td>
<td>A△ZigDown</td>
<td>A△Zag</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>B△Halt</td>
<td>B△Zig (see Figure 14)</td>
<td>B△ZigDown</td>
<td>B△Zag</td>
</tr>
<tr>
<td>C</td>
<td>N/A</td>
<td>C△ZigDown</td>
<td>C△EndofTape (see Figure 15(a))</td>
<td>C△Zag</td>
</tr>
<tr>
<td>D</td>
<td>N/A</td>
<td>D△ZigUp (see Figure 17(e))</td>
<td>D△ZigDown</td>
<td>D△Write (see Figure 17)</td>
</tr>
<tr>
<td>E</td>
<td>N/A</td>
<td>E△ZigUp (see Figure 18)</td>
<td>E△ZigDown</td>
<td>E△CarriageReturn (see Figure 20)</td>
</tr>
<tr>
<td>F</td>
<td>N/A</td>
<td>F△ZigUp (see Figure 21)</td>
<td>F△ZigDown</td>
<td>F△Zag</td>
</tr>
<tr>
<td>G</td>
<td>N/A</td>
<td>G△Read0 (see Figure 22)</td>
<td>G△ZigCopy (see Figure 24)</td>
<td>G△ZagCopy</td>
</tr>
</tbody>
</table>

**Figure 7** The bricks inside each block.
Figure 8 The brick automaton implementing the block automaton. Note that in the Zig-Down-phase, each letter of the word above is copied by the first module \textbf{G} of the Copy\textsuperscript{1} block and the end of the word is detected by the first module \textbf{G} of the block. In the Zag-phase however, each letter of the word above is copied by the penultimate module \textbf{G} of the Copy block and the beginning of the word is detected by the last \textbf{G} of the block.

Note that this automaton is presented as a “transducer” producing the block diagram: the variables \(c\) and \(b\), which counts up to \(n\), are introduced only to output the right module at the right time during the zig-down and zag phases (assuming the seed is wellformed).
result of their interactions. Similarly, the memory of the system consists of the beads already placed in the area currently visited (the environment).

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

**Default folding.** By default, during the zig-up phase, $B$ is attracted to the left and folds to the right only in presence of $E$ above. This allows to continue the folding only if the tape word is not empty or to halt it otherwise (see Figure 14).

**Geometry obstruction.** An typical example is illustrated by $G$. During the zig-up phase where the absence of environment below the block $\text{Read} \uparrow$ allows $G$ to fold downward at the beginning (see Figure 22) which shift the molecule by 7 beads along $F$ resulting in $G$ to adopt the glider-shape (more details on this mechanism in the next section). Whereas during the zig-down phase, $G$ cannot make this loop because it is occupied by a previously placed $G$. This results in a perfect alignment of $G$ with $F$ whose strong attraction forces $G$ to adopt the switchback shape.

**Geometry of the environment.** Figure 10 shows how the shape of the environment is used to change the direction of $G$ in glider-shape. This results in modifying the entry point in the environment and allows the Oritatami system to trim the leading 0s in the tape word, switch from zip-up to zig-down phase when reading a 1 and from zig- to zig-up phase when it has rewind to the beginning of the tape word.

### 6.3 Easing the design: getting the freedom you need

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

**Socks** work by letting a glider/switchback module fold into a switchback conformation for some time when it would otherwise fold into a glider. Examples are shown in Figure 11. They are easy to implement, since the socks naturally adopt the same shape as turn that part of the module has in the switchback conformation. They offer a lot of freedom in the design, for several reasons:

- First, they simplify the design of important switchback part by *lifting the need for implementing the glider conformation* for that part, as shown in Figure 11(a).
Second, a glider naturally progresses at speed 1/3. Adding a sock allows us to slow its progression down to speed 1/5 for some time, as in Figure 11(b), and therefore realign them. We used that feature repeatedly to “shift” some modules, by starting them with an initial speed-1 (i.e. straight line) progression, as in Figure 11(b), and then compensate for that speed by introducing a socket later on, and realign the brick with others. This is a key point in the design, as it allowed us to separate the Read and Copy functions into different parts of module $G$, and therefore to get less constraints on rule design. In the specific case of module $G$, the Copy-function occurs at the center of the module, while the Read-function is implemented earlier in module!

Finally, socks allow to prevent unwanted interactions between beads by concealing potentially armful beads in unreachable area as in Figure 11(c).

**Exponential coloring** is a key tool to allow module $G$ to fold into different shapes, glider or switchback, along module $F$, when folding in the Read configuration. This trick is described in greater detail in Section 7.10. The problem it solves is that in order for $G$ to fold into its switchback shape, we need strong interactions between $G$ and neighboring module $F$, whereas in order for $G$ to adopt the glider shape, we want to avoid those interactions. This is made possible because gliders progress at speed 1/3 while switchbacks progress at speed 1. Using a power-of-3 coloring allows to realise these contradicting goals altogether (precise construction is analysed in Lemma 15 in Section 8).
7 The Sequences for the Bricks

We now define the primary structure we use to simulate a skipping cyclic tag system. The complete rule is given in the appendix in Section C.

7.1 Extra notations for sequences

In order to do so, we need a few extra notations to manipulate sequences: if \( u \) and \( v \) are finite sequences, we write their concatenation as \( u \cdot v \). For any two integers \( 0 \leq i \leq j < |u| \), we also write \( u_{[i...j]} \) for \( u_i u_{i+1} \ldots u_j \). The reverse sequence of \( u \), written as \( u^R \), is \( u_{|u|} u_{|u|-1} \ldots u_1 u_0 \).

Finally, given a sequence \( u \), we write \( u\langle\langle a_1@i_1, \ldots, a_k@i_k\rangle\rangle \) for the sequence \( w \) where for all \( j \in \{1, 2, \ldots, k\} \), bead \( i_j \) of \( u \) has been replaced by \( a_j \):

\[
w_i = \begin{cases} a_j & \text{if } i = i_j \text{ for some } j \\ u_i & \text{otherwise} \end{cases}
\]

By extension, we write \( u\langle\langle v@k..l\rangle\rangle \) for the sequence \( w \) where for all \( i \in \{k, k+1, \ldots, l\} \), the beads at indices \( k \) to \( l \) of \( u \) have been replaced by the word \( v \) (of length \( l - k + 1 \)):

\[
w_i = \begin{cases} v_{i-k} & \text{if } k \leq i \leq l \\ u_i & \text{otherwise} \end{cases}
\]

For an infinite sequence of (finite) words \((u_i)_{i \geq 1}\), we denote by \( \bigcirc_{i \geq 1} u_i \) the infinite word \( u_1 u_2 \ldots u_1 \ldots \) obtained by containing all the words \( u_1, \ldots \).

7.2 More constants: \( k \), \( \lambda \) and \( \kappa \)

We also define three new constants as helpers for the module sequences:

\[
k = \frac{h-3}{6}. \text{ Note that by the definition of } h \text{ in Section 4.2, } k \text{ is even.}
\]

\[
\lambda = W/2. \text{ By the definition of } W \text{ in Section 4.2, } \lambda \mod 24 = 0.
\]

\[
\kappa = W/24. \text{ By the definition of } W \text{ in Section 4.2, } \kappa \text{ is even.}
\]

7.3 Seed:\Seed: Seed for input \( u \).

We first describe the seed \Seed, which is essentially an encoding of the input word \( u \) to the skipping cyclic tag system we are simulating. As per the definition of oritatami systems, this is a conformation, thus a sequence of beads together with positions (i.e. all other sequences have their positions defined by the folding dynamics). These positions will be encoded incrementally, using the following notation, relative to the axes define in Figure 6:

\[
a \stackrel{\sw}{\searrow} b \text{ means a bead of type } a, \text{ followed by a bead of type } b, \text{ such that } \text{pos } b = \text{pos } a + \begin{pmatrix} 0 \\ 1 \end{pmatrix}.
\]

\[
a \stackrel{\se}{\searrow} b \text{ means a bead of type } a, \text{ followed by a bead of type } b, \text{ such that } \text{pos } b = \text{pos } a + \begin{pmatrix} 1 \\ 0 \end{pmatrix}.
\]

\[
a \stackrel{\se}{\nearrow} b \text{ means a bead of type } a, \text{ followed by a bead of type } b, \text{ such that } \text{pos } b = \text{pos } a + \begin{pmatrix} 1 \\ -1 \end{pmatrix}.
\]

\[
a \stackrel{\sw}{\nearrow} b \text{ means a bead of type } a, \text{ followed by a bead of type } b, \text{ such that } \text{pos } b = \text{pos } a + \begin{pmatrix} 0 \\ -1 \end{pmatrix}.
\]

\[
a \stackrel{\nw}{\nw} b \text{ means a bead of type } a, \text{ followed by a bead of type } b, \text{ such that } \text{pos } b = \text{pos } a + \begin{pmatrix} -1 \\ 0 \end{pmatrix}.
\]
For example, a sequence such as $a \xrightarrow{\text{SegSeedLineFeed}} b \xrightarrow{\text{SegSeedPrefix}} c \xrightarrow{\text{SegSeedLineFeed}} d \xrightarrow{\text{SegSeedPrefix}} e$, starting at $\left( \begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right)$ is a sequence of five beads: a bead of type $a$ at $\left( \begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right)$, a bead of type $b$ at $\left( \begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right)$, a bead of type $c$ at $\left( \begin{array}{c} 1 \\ 1 \\ 1 \end{array} \right)$, a bead of type $d$ at $\left( \begin{array}{c} 0 \\ 1 \\ 1 \end{array} \right)$, and a bead of type $e$ at $\left( \begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right)$.

We are now ready to describe module $\text{Seed}_a$, built by combining 4 types of conformation segments, see Figure 12:

$$\text{SegSeedTerm} = (J_{18} \xrightarrow{\text{SegSeedPrefix}} J_{17} \xrightarrow{\text{SegSeedLineFeed}} J_{16} \xrightarrow{\text{SegSeedPrefix}} J_{15} \xrightarrow{\text{SegSeedLineFeed}} J_{14} \xrightarrow{\text{SegSeedPrefix}} J_{13} \xrightarrow{\text{SegSeedLineFeed}} J_{12} \xrightarrow{\text{SegSeedPrefix}} J_{11} \xrightarrow{\text{SegSeedLineFeed}} J_{10} \xrightarrow{\text{SegSeedPrefix}} J_{9} \xrightarrow{\text{SegSeedLineFeed}} J_{8} \xrightarrow{\text{SegSeedPrefix}} J_{7} \xrightarrow{\text{SegSeedLineFeed}} J_{6} \xrightarrow{\text{SegSeedPrefix}} J_{5} \xrightarrow{\text{SegSeedLineFeed}} J_{4} \xrightarrow{\text{SegSeedPrefix}} J_{3} \xrightarrow{\text{SegSeedLineFeed}} J_{2} \xrightarrow{\text{SegSeedPrefix}} J_{1} \xrightarrow{\text{SegSeedLineFeed}} J_{0})^2 + \frac{b-3}{2}$$

$$\text{SegSeedPrefix} = A_9 \xrightarrow{\text{SegSeedLineFeed}} A_8 \xrightarrow{\text{SegSeedPrefix}} A_7 \xrightarrow{\text{SegSeedLineFeed}} A_6 \xrightarrow{\text{SegSeedPrefix}} A_5 \xrightarrow{\text{SegSeedLineFeed}} A_4 \xrightarrow{\text{SegSeedPrefix}} A_3 \xrightarrow{\text{SegSeedLineFeed}} A_2 \xrightarrow{\text{SegSeedPrefix}} A_1 \xrightarrow{\text{SegSeedLineFeed}} A_0 \xrightarrow{\text{SegSeedPrefix}}$$

$$\text{SegSeed} = (H_{21} \xrightarrow{\text{SegSeedLineFeed}} H_{20} \xrightarrow{\text{SegSeedPrefix}} H_{19} \xrightarrow{\text{SegSeedLineFeed}} H_{18} \xrightarrow{\text{SegSeedPrefix}} H_{17} \xrightarrow{\text{SegSeedLineFeed}} H_{16} \xrightarrow{\text{SegSeedPrefix}} H_{15} \xrightarrow{\text{SegSeedLineFeed}} H_{14} \xrightarrow{\text{SegSeedPrefix}} H_{13} \xrightarrow{\text{SegSeedLineFeed}} H_{12} \xrightarrow{\text{SegSeedPrefix}} H_{11} \xrightarrow{\text{SegSeedLineFeed}} H_{10} \xrightarrow{\text{SegSeedPrefix}} H_{9} \xrightarrow{\text{SegSeedLineFeed}} H_{8} \xrightarrow{\text{SegSeedPrefix}} H_{7} \xrightarrow{\text{SegSeedLineFeed}} H_{6} \xrightarrow{\text{SegSeedPrefix}} H_{5} \xrightarrow{\text{SegSeedLineFeed}} H_{4} \xrightarrow{\text{SegSeedPrefix}} H_{3} \xrightarrow{\text{SegSeedLineFeed}} H_{2} \xrightarrow{\text{SegSeedPrefix}} H_{1} \xrightarrow{\text{SegSeedLineFeed}} H_{0})^{w - 17}$$

Each letter $a \in \{\emptyset, 1\}$ is encoded in the seed by the conformation:

$$\text{SegSeedLetter}(a) = (\text{SegSeed(1)} \xrightarrow{\text{SegSeedPrefix}} \text{SegSeedPrefix})^{n - 1} \text{SegSeed(a)} \xrightarrow{\text{SegSeedPrefix}}$$

Then, the module $\text{Seed}_a$ is:

$$\text{Seed}_a = \text{SegSeedTerm} \xrightarrow{\text{SegSeedPrefix}} \left( \bigcup_{i=1}^{u} \text{SegSeedLetter}(u_i - 1) \xrightarrow{\text{SegSeedPrefix}} \right) \text{SegSeedLineFeed}$$

### 7.4 $A$: Zig-Init.

The first module, $A$, is defined as:

$$A = A_{0.4} \cdot (A_{5.10})^{3k-1} \cdot A_{5.7} \cdot A_{9.10} \cdot A_{11.12}.$$  

The length of $A$ is therefore $5 + 6(3k - 1) + 3 + 1 + 2 + 2 = 3h - 2$. The proof trees in Section 8.2 prove that $A$ always has height $H = 2 + 2(3k - 1) + 3 = h$, and width $3$, and folds as in Figure 13.

### 7.5 $B$: Empty word detector.

The next module is $B$, whose purpose is to test whether the word is empty, and orient the folding either into a closed connected component of the place, if the word is empty, or to the outside of that connected component. This module is defined as $B = B_{0.4}$, which is of length 5, and its two possible functions are shown in Figure 14.

### 7.6 $C$: End of word detector.

If $B$ does not detect an empty word, the folding goes on to $C$, whose purpose is to detect the end of the word: if the current position is at the end of the current word, a production segment (encoded by a sequence of $D_0$ and $D_1$) needs to fold into a word appended at the end of the current word. Else, $C$ folds into a switchback conformation.
Figure 12 The brick $\text{Seed}(u)$. 
7.7 \textbf{D: Letters}

Module \textbf{D} defines the encoding of the letters in productions of the skipping cyclic tag system. It takes three parameters:

- a letter \( x \in \{0, 1\} \),
- a parameter \( r \in \{0, 1, 2\} \) to indicate whether this letter is the first letter in the production word (in which case \( r = 0 \)), at an odd position in the production word (in which case \( r = 1 \)), or at an even position in the production word (in which case \( r = 2 \)),

This module is defined as \( \textbf{D}^{x} = (\textbf{C}0..2)^{2k} \cdot (\textbf{C}3..5)^{k} \cdot \textbf{C}7 \cdot \textbf{C}8 \cdot \textbf{C}6..8^{k-1} \cdot \textbf{C}9..14^{k-1} \cdot \textbf{C}9..10 \cdot \textbf{C}15..16 \cdot \textbf{C}13. \)

The length of \( \textbf{D}^{x} \) is \( 3h - 10 = 3 \times 2k + 3k + 3(k - 1) + 6(k - 1) + 5 \).

Its two possible conformations in the zig phase are shown in Figure 15:

- The left-hand side figure shows the conformation at the end of the word. Its height is \( H_{exp} = \frac{3h - 9}{2} - 2 \), and its width is \( W_{exp} = 2 \).
- The right-hand side figure shows the conformation in the other case. Height (upright): \( H_{up} = h - 3 \). Width (upright): \( W_{up} = 3 \).
(a) The conformation of C at the end of the word (before D folds into the appended production).

(b) The conformation of C when not at the end of the word.

Figure 15 The folding of C in the zig phase.

and a parameter \( t \) indicates whether that letter is the last letter of the word (\( t = 1 \)) or not (\( t = 0 \)).

This module therefore comes in twelve different versions, all of the same length \( 3W + 6 \times 5 = 6(\lambda + 5) = 6 \times (12\kappa + 5) \).

We first describe four helper sequences, each of length \( \lambda + 5 = 12\kappa + 5 \):

- \( \text{SegD0} = D23..33 \cdot E6..11 \cdot (E0..11)^{\kappa-1} \).
- \( \text{SegD1} = (E12..23)^{\kappa} \cdot D49..45 \).
- \( \text{SegD2} = D34..44 \cdot E30..35 \cdot (E24..35)^{\kappa-1} \).
- \( \text{SegD3} = (E36..47)^{\kappa} \cdot D54..50 \).
7.7.1 Encoding of $x = 1$

Then, we define two particular versions of $D_1$, from which the other versions are derived by replacing a few beads:

$$D_{120} = \text{SegD}0 \cdot \text{SegD}1 \cdot \text{SegD}2 \cdot \text{SegD}3 \cdot \text{SegD}0 \cdot \text{SegD}1.$$  
$$D_{110} = \text{SegD}2 \cdot \text{SegD}3 \cdot \text{SegD}0 \cdot \text{SegD}1 \cdot \text{SegD}2 \cdot \text{SegD}3.$$

Module $D_{100}$ is obtained by modifying the 17 first beads of $D_{120}$:

$$D_{100} = D_{120} \langle \text{SegD}0..16 \rangle.$$

Then, for all $r \in \{0, 1, 2\}$, the trailing versions $D_{1r1}$ of $D_1$ are obtained by modifying the 8th and 5 last beads of $D_{1r0}$, as follows:

$$D_{1r1} = D_{1r0} \langle \text{SegD}17+(3W+22), \text{SegD}18..22+(3W+25)..(3W+29) \rangle.$$

7.7.2 Encoding of $x = 0$

For all $r \in \{0, 1, 2\}$ and $t \in \{0, 1\}$, module $D_{0rt}$ is obtained by replacing most of the beads in the range $3w+1..3w+13$ as follows:

$$D_{0rt} = D_{0r} \langle \text{SegL}17+(3w+1), \text{SegL}18+(3w+2), \text{SegD}55..62+(3w+6)..(3w+13) \rangle.$$

7.7.3 Possible conformations

The possible conformations of $D$ are shown in Figures 16 and 17.

7.7.4 Size and alignment of the module

First note that the height of module $D$ (i.e. the encoding of a single letter of a production), when folded into its switchback conformation, is $H_{up} = L/6 = W/2 + 5$, and its width is $W_{up} = 6$.

We will now prove a small lemma to make the proofs of a claim in Section 8 easier:

**Lemma 10.** The segment of $D_{1r1}$ encoding the “bump” in the expanded conformation is always adjacent to the same beads of $C$, when both $C$ and $D_{1r1}$ are in their switchback conformation.

**Proof.** Note that $w = 6 \mod 12$, hence index $i = 3w+1$, the first index of the bump, is such that $i \mod 12 = 7$. This corresponds to index $j = (11 - i) + 5 = 9 \mod 12$ in the previous column, and hence we get the following table:

<table>
<thead>
<tr>
<th>$i \mod 12$</th>
<th>$j = (11 - i) + 5 \mod 12$</th>
<th>Neighboring bead in previous and next columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3w+1 = 7$</td>
<td>$9$</td>
<td>$36 + 9 = \text{E}45 / \text{L}17 / \text{E}21 = 12 + 9$</td>
</tr>
<tr>
<td>$3w+2 = 8$</td>
<td>$8$</td>
<td>$36 + 8 = \text{E}44 - \text{L}18 - \text{E}20 = 12 + 8$</td>
</tr>
<tr>
<td>$3w+6 = 0$</td>
<td>$4$</td>
<td>$36 + 4 = \text{E}40 - \text{D}55 - \text{E}16 = 12 + 4$</td>
</tr>
<tr>
<td>$3w+13 = 7$</td>
<td>$9$</td>
<td>$36 + 9 = \text{E}45 - \text{D}62 - \text{E}21 = 12 + 9$</td>
</tr>
</tbody>
</table>

This proves the lemma statement.
7.8 **E₃**: Padding with \( L - a \) extra blanks for \( 0 \leq a \leq L \).

The purpose of module \( E₃ \) is to make sure that all production segments are of the same length, independently from the length of production words in the simulated skipping cyclic tag system \( S \). Recall from Section 4.2 that \( L \) is the length of the longest production word of \( S \).

This module has two possible conformations: one in switchback, as shown in Figure 18, and one expanded at the end of the appended production. An outline of the latter conformation is shown in Figure 19.

We will now define the different parts of module \( E₃ \), composed of 4 parts:

- the two first parts are based on the two infinite sequences:
  
  \[
  \text{SegEA} = ((F₀..11) κ \cdot (F₁₂..23) κ \cdot (F₂₄..35) κ \cdot (F₃₆..47) κ) ^ {∞} \\
  \text{SegEB} = ((G₀..11) κ \cdot (G₁₂..2₃) κ \cdot (G₂₄..3₅) κ \cdot (G₃₆..4₇) κ) ^ {∞}
  \]

- \( \text{SegEC} = H₀..4 \cdot (H₅..1₆) ^ {q-1} \cdot H₅..1₀ \cdot H₁₇..₂₄ \) of length \( 5 + 12(q-1) + 6 + 8 = 3h - 2 \), where
  
  \[
  q = \frac{h - 3}{4} = 0 \ mod \ 3.
  \]
Proving the Turing Universality of Oritatami Co-Transcriptional Folding

Figure 17: Module [D] expanded to append the new production word at the end of the current word.
(a) Blueprint of the $\mathbf{E}_-$-Zig-Up brick.

(b) Precise description of the $\mathbf{E}_-$-Zig-Up brick.

Figure 18 The $\mathbf{E}_-$-Zig-Up brick.
XX:30 Proving the Turing Universality of Oritatami Co-Transcriptional Folding

(a) The blueprint of the \textit{CarriageReturn Brick}, made of four bricks \textit{EA0}, \textit{EB}, \textit{EC} and \textit{ED}.

(b) The blueprint of the \textit{CarriageReturn Brick}, made of bricks \textit{EA1/2}, \textit{EB}, \textit{EC} and \textit{ED}.

\textbf{Figure 19} Outline of the four different parts of module \textit{E}, when folded at the end of the appended production word. See Figure 20 for the detailed beads of each part.

The last part is composed of five sequences:

- \text{SegED}_0 = \text{I15} \cdot \text{I16} \cdots \text{(I0..5)}^{k-1} \cdot \text{I0} \cdot \text{I1} \cdot \text{I18} of length 6 + 6(k - 1) + 3 = h
- \text{SegED}_1 = \text{I19} \cdot \text{I7} \cdots \text{(I6..8)}^{2k-1} \cdot \text{I6} \cdot \text{I7} \cdot \text{I15} \cdot \text{I16} of length 3 + 3(2k - 1) + 4 = h + 1
- \text{SegED}_2 = \text{I17} \cdot \text{I10} \cdot \text{I11} \cdots \text{(I9..11)}^{2k-1} \cdot \text{I9} \cdot \text{I10} \cdot \text{I19} of length 3 + 3(2k - 1) + 3 = h
- \text{SegED}_3 = \text{I18} \cdot \text{I13} \cdots \text{(I12..14)}^{2k-1} \cdot \text{I12} \cdot \text{I13} \cdot \text{I19} of length 3 + 3(2k - 1) + 3 = h
- and \text{SegED}_4 = \text{I19} \cdot \text{I1} \cdot \text{I2} \cdots \text{(I0..2)}^{2k} of length 3 + 3 \times 2k = h

We may now define the sequence for the module \textit{E}_0 for 0 \leq a \leq L by letting \( K = 3W(L - a + P) \), and:

\[ \text{HeadE}_a = (\text{SegEA})[b,b+3c-2] \cdot \text{F51} \cdot (\text{SegEB})[b+3c,b+K-2], \] of length \( K - 1 \), where \( b = 0 \) if \( a \) is even, and \( b = 2\lambda \) if \( a \) is odd.

\[ \text{TailE} = \text{SegEC} \cdot \text{SegED}_0 \cdot \text{SegED}_1 \cdot \text{SegED}_2 \cdot \text{SegED}_3 \cdot \text{SegED}_4. \]

Then the module \textit{E}_a is:

\[ \text{E}_a = (\text{HeadE}_a) \langle \text{F48..49}@0\cdots1, \text{F50}@11 \rangle \cdot \text{G48} \cdot \text{TailE} \]

and for \( a > 0 \):

\[ \text{E}_a = (\text{HeadE}_a) \cdot \text{G48} \cdot \text{TailE}. \]

The length of module \textit{E}_a is:

\[ |\text{E}_a| = K + 8h - 1 = 6\lambda(L - a + P) + 8h - 1 \]
\[ = 6 \times 12 \lambda \times (L - a + P) + \frac{8h}{mod (3 \times 48)} = 24 \frac{8h}{mod (2 \times 48)} - 1 \]

Therefore, for any word \( a \), \( |\text{E}_a| \mod 48 = 23 \).

We will now prove a lemma, used later in Section 8 to prove that the two conformations (switchback and expanded) can be obtained at the same time:

\textbf{Lemma 11}. When folded in the switchback conformation (i.e. as in Figure 18), all beads of \textit{SegEC} are far enough from \textit{SegEA} to be attracted by them.

Hence, the attractions between these two parts can be freely chosen to “force” \textit{SegEC} to fold into a straight line instead of a glider in the expanded conformation.
Proof. Let $c = \frac{E_a - 15}{4}$, and note that $c \mod 12 = 2$. Now, the width of $\text{SegEB}$, when folded in the switchback conformation is $K - (3c - 1)$. Moreover, $3c - 1 = 5 \mod 36$. Therefore, for any padding length $a$:

$$K - (3c - 1) = \frac{6}{4} \lambda(L - a + P) - 6h + 13 = \frac{3}{2} \lambda(L - a + P) - 12\lambda + 6(w + 3) + 13$$

$$> \lambda \left( \frac{3P}{2} - 12 \right) \geq 4.5\lambda, \quad \text{since} \quad P \geq 11.$$  

Therefore, the width of $\text{SegEB}$ is at least 5 columns, and since the delay of our simulation is 3, no bead of $\text{SegEC}$ can ever be attracted to a bead of $\text{SegEA}$ in the switchback conformation.

Lemma 12. **Bead F51 is never on an edge of brick E2.**

Therefore, that bead is not involved in a turn in that brick, which means that the attraction rule can be decided mostly based on its $\text{EA0}$ brick, to initiate the turn at the end of the padding, in the expanded conformation of $E$.

Proof. That bead is at index $3c - 1$ from the beginning of the module, and moreover between the $\text{SegEA}$ and $\text{SegEB}$ segments, which are both folded into switchbacks of height $\lambda$ in that brick. Now, note that $\lambda \mod 12 = 0$, and that $c \mod 12 = 2$. Therefore, $(3c - 1) \mod 12 = 5$, which means that bead F51 cannot be on the edge of the switchback.
(a) The subbrick EA0, when $a = L$.

(b) The subbrick EA1, when $L - a$ is odd.

(c) The subbrick EA2, when $L - a$ is even and $a < L$.

(d) The subbrick EB.

Here, each $i$ can take any value with suitable parity
in $\{a + b : a \in \{0, 5, 6, 11\} \text{ and } b \in \{0, 12, 24, 36\}\}$

(e) The subbrick EC.

here, $j$ is either 5 or 29; and each $i$ can take any value in $\{a + b : a \in \{0, 5, 6, 11\} \text{ and } b \in \{0, 12, 24, 36\}\}$

(f) The subbrick ED.

Figure 20 The $E_\text{br}^\text{carr}$-carriageReturn bricks.
7.9 \( F \): Zag-Init.

Finally, after the end of the padding, module \( F \) is used to start the copying phase. Module \( F \) has the same conformation in the zig and zag phases, up to a rotation of 180 degrees. The conformation of \( F \) in the zig phase is shown in Figure 21.

Module \( F \) is composed of three parts. The beginning is \( \text{Head}_F \) and the end is \( \text{Tail}_F \), defined as:

- \( \text{Head}_F = J_{0..4} \cdot (J_{5..10})^{3k-1} \cdot J_{5..7} \cdot J_{11..23} \) of length \( 5 + 6(3k - 1) + 3 + 13 = 3h + 6 \)
- \( \text{Tail}_F = J_{48} \cdot (J_{51..48})^{9} \cdot J_{51} \cdot J_{52} \cdot J_{49..48} \) of length \( 1 + 4 \times 10 = 41 \)

The middle part is made of the following “exponential” sequence:

- for even \( i \geq 2 \), let \( \text{SegExp}(i) = J_{24..29} \cdot (J_{30..35})^{3^{i-1}-1} \) of length \( 6 \cdot 3^{i-1} \);
XX:34 Proving the Turing Universality of Oritatami Co-Transcriptional Folding

and for odd \( i \geq 3 \), let \( \text{SegExp}(i) = J_{36..41} \cdot (J_{42..47})^{3i-1-1} \) of length \( 6 \cdot 3i-1 \);
so as to define the following infinite sequence, which is the concatenation of \( \text{SegExp}(2) \), \( \text{SegExp}(3) \), \( \text{SegExp}(4) \) . . . :

\[
\text{SegExpF} = \bigcirc_{i \geq 2} \text{SegExp}(i)
\]

Finally, the beads sequence for \( F \) is:

\[
F = \text{HeadF} \cdot (J_{39..41} \cdot \text{SegExpF}_{[0..(h-51)]})^R \cdot \text{TailF}.
\]

Therefore, the length of module \( F \) is \( 3h + 6 + (h - 50) + 3 + 41 = 4h \).
We claim that for all \( i \), the pattern \( \text{SegExp}(i) \) starts at index \( 3i - 9 \) of \( \text{SegExpF} \). Indeed:

\[
\sum_{j=2}^{i-1} 6 \cdot 3^{j-1} = 2 \cdot \frac{3^i - 9}{3 - 1} = 3^i - 9
\]

Finally, notice \( |\text{SegExpF}_{[0..(h-51)]}| \mod 12 = (h - 50) \mod 12 = 1 \).

7.10 \( G \): Read-Copy-Line Feed module.

Module \( G \) is the last module, and the one in charge of reading and copying the information around. This module can fold into a seven different conformations:

- Figures 22 and 23 show the module reading the encodings of \( 0 \) and \( 1 \), respectively. These conformations only happen during the zig phase.
- Figures 24 and 25 show the module copying the encodings of \( 0 \) and \( 1 \), respectively. These conformations are shown on these figures in the zig phase, but are the same in the zag phase, rotated by 180 degrees.
- Figures 26 only happens at the end of the zag phase, after copying the word, and before starting the next step.

Module \( G \) is of length exactly \( 6h - 1 \), and consists of six parts, each of length approximately \( h \). We described these parts now:

The first part is the most sophisticated since it can be folded either in a straight line, and hence “progress vertically” at speed 1 (i.e. one row per bead), or in a glider, which progresses vertically at speed \( 1/3 \) (i.e. two rows every six beads).

The other parts just contains a small delay loop (a sock) that allow to separate crucial sensing function from basic geometry, as explained in Section 6.3. For the rest of this section, let \( k = \frac{h - 3}{6} = 0 \mod 2 \).

Part 1: As for Module \( F \) we define the following exponential pattern:

- for even \( i \geq 2 \), let \( \text{SegExp}'(i) = K_{4..9} \cdot (K_{10..15})^{3i-1-1} \) of length \( 6 \cdot 3i-1 \);
- and for odd \( i \geq 3 \), let \( \text{SegExp}'(i) = K_{16..21} \cdot (K_{22..27})^{3i-1-1} \) of length \( 6 \cdot 3i-1 \);
- so as to define the infinite sequence:

\[
\text{SegExpG} = \bigcirc_{i \geq 2} \text{SegExp}'(i)
\]

As for \( \text{SegExpF} \), the pattern \( \text{SegExp}'(i) \) starts at index \( 3^i - 9 \) exactly in \( \text{SegExpG} \).

The first part of \( G \) is:

\[
\text{SegG1} = L_{0..6} \cdot K_{3} \cdot (K_{0..3})^{9} \cdot K_{0..2} \cdot L_{7..10} \cdot \text{SegExpG}_{[8..(h-51)]}
\]

of length \( 7 + 1 + 9 \times 4 + 3 + 4 + h - 51 - 7 = h - 7 \). Note that \( h - 51 = 0 \mod 12 \) and thus the index of the last bead of the exponential part is a multiple of 12.
Figure 22: The brick $G$-read0.
Part 2: SegG2 = K32 · K33 · (K28..33)\textsuperscript{k–3} · K28..32 of length \(2 + 6(k – 3) + 5 = h – 14 = 1 \mod 12\).

Part 3: SegG3 = K35..39 · (K34..39)\textsuperscript{k–14} · K34 · K35 · L39..41 · K45 · (K40..45)\textsuperscript{10} · K40 · K41 of length \(5 + 6(k – 14) + 6 + 10 \times 6 + 2 = h – 14 = 1 \mod 12\).

Part 4: SegG4 = K50..51 · (K46..51)\textsuperscript{k–3} · K46..48 of length \(2 + 6(k – 3) + 3 = h – 13 – 3 = h – 16 = 5 \mod 6\).

Part 5: SegG5 = K55..57 · (K52..57)\textsuperscript{k–6} · K52..53 · L74 · L75 · K56..57 · (K52..57)\textsuperscript{2} · K52..53 of length \(3 + 6(k – 6) + 2 + 2 + 2 + 2 + 6 + 2 = h – 16 = 5 \mod 6\).

Part 6: SegG6 = K63 · (K58..63)\textsuperscript{8–19} · K58..61 · L91..99 · M0..19 · K67..69 · (K64..69)\textsuperscript{10} · M20..30 of length \(1 + 6(k – 19) + 4 + 9 + 20 + 3 + 60 + 11 = h – 9 = 6 \mod 12\).

Finally,

\[G = \text{SegG1} \cdot L11..24 \cdot \text{SegG2} \cdot L25..38 \cdot \text{SegG3} \cdot L42..55 \cdot \text{SegG4} \cdot L56..73 \cdot \text{SegG5} \cdot L76..90 \cdot \text{SegG6}\]

of total length = \(h – 7 + 14 + h – 14 + 14 + h – 14 + 14 + h – 16 + 18 + h – 16 + 15 + h – 9 = 6h – 1\).

8 Correctness of the folding

We will now resume and expand the explanation given in Section 3. Here is how we proceeded to ensure the correctness of our design:

1. Enumerate all the surrounding for each brick of each module
2. Enumerate all possible modules following the module
3. Generate automatically human-readable certificate of the correctness of the folding for each possibility, in the form of proof trees.
4. In the few cases where the surrounding may vary, prove that it has no incidence on the folding of the brick. This happens only for three bricks exactly: when the brick G\textsuperscript{Read} zig-folds along F\textsuperscript{ZigUp}, when the top of the brick G\textsuperscript{Read1} folds, and when the zag-bricks fold under D\textsuperscript{Write}. 
The various bond patterns repeated inside each part of this brick

Figure 24 The brick @-copy@.
The various bond patterns repeated inside each part of this brick.

Concatenate for $j = 1$ to $\infty$

Truncate this infinite sequence to height $h$.
Figure 26 The brick.
The lemmas in Section 7 have proved that the bead alignment in each brick does not change when $n$ and $L$ vary. This implies that the figures of the bricks are indeed generic. It follows that with the exception of the three cases listed in point 4 above, and handled in Section 8.1, it is enough to prove the folding of each brick only once. And as most of them are made of repeating patterns, only a finite number of environments have to be considered. That last case will be treated in Section 8.2 using an automatic procedure which produces human-readable certificates called proof-trees.

8.1 The three bricks with varying environments

The following lemma show that it is enough to proof one folding of the bricks under a $\text{D}^{\text{Write}}$, all the other are the same since there are no interaction between the D-write brick and any brick folding immediately below it.

Lemma 13 (Zag-folding under $\text{D}^{\text{Write}}$). The modules ZAG-folding under the bricks $\text{D}^{\text{Write}}$ have no interaction with $\text{D}^{\text{Write}}$, with the only exceptions of:

- the beads $A_0$ and $A_1$ of module $\text{A}$ which have bonds with beads $E(2+12i)$ and $L_{17}$ for $A_0$, and $E(9+12i)$ for $A_1$, for all $0 \leq i \leq 3$.
- the beads $L_{17}, L_{18}, D_{57}, D_{58}$ (the bump in module $\text{D}_{0}$) which bond with the beads $L_{65}, L_{64}, L_{31}$ so that the corresponding module $\text{G}$ folds into the expected brick $\text{G}^{\text{Zag Copy 0}}$.

Proof. Figure 27 lists all the possible interactions between the beads accessible from below the $\text{D}^{\text{Write}}$ bricks (to the left) with the beads at the top the modules zag-folding below it that can interact with them (to the right).

The only possible bonds are thus:

- with beads $D_{17}$ and $D_{22}$: (in green on Figure 27) these are only present at the junction between the bricks $\text{D}^{\text{Write}}$ and $\text{E}^{\text{Carriage Return}}$, at the end of the rightmost $\text{D}^{\text{Write}}$ brick. The correctness of the zag-folding of the $\text{F}^{\text{Zag}}$ brick below is given next in the proof-trees section.

- with beads $L_{17}, L_{18}, D_{56}, D_{57}, D_{58}, D_{62}$: (in blue on Figure 27) these beads are only present in the spike encoding a $\emptyset$ in the brick $\text{D}^{\text{Write}}$, and these interactions are the one expected to ensure the copy of the encoding of $\emptyset$ by the module $\text{G}$ that will Zag-fold below.

- and finally between beads $A_0$ and $A_1$, and 4 groups of beads: $E_{2}, E_{3}, E_{8}, E_{9}$, then $E_{14}, E_{15}, E_{20}, E_{21}$, then $E_{26}, E_{27}, E_{32}, E_{33}$, and finally $E_{38}, E_{39}, E_{44}, E_{45}$ (in red on Figure 27). As the width of a zag-folded production segment is $w + 6 = 0 \mod 12$, the beads $A_0$ and $A_1$ are always aligned with the same beads within each of these groups (see Figure 17), namely $A_0$ with $E_{2}, E_{14}, E_{26}$ and $E_{38}$, and $A_1$ with $E_{9}, E_{21}, E_{33}$ and $E_{45}$. Furthermore as the interactions of $A_0$ and $A_1$ are the same with each of them, it is enough to prove that the module $\text{A}$ zag-folds correctly between one of these groups only, which is done next in the proof-trees section.

It follows that outside these three cases (each handled by a proof-tree, see later), no interactions are possible and the modules will zag-fold below the $\text{D}^{\text{Write}}$ bricks independently of the exact beads that are present inside. It is thus enough to show that each module zag-folds correctly at any location to ensure that it zag-folds correctly anywhere below the $\text{D}^{\text{Write}}$ brick.

Lemma 14 (Top of $\text{G}^{\text{Read 1}}$). During the folding of the brick $\text{G}^{\text{Read 1}}$, no bead in $\text{G}$ interacts with the row above but at its two extremities, i.e. the 82 top-leftmost beads and the 11 last ($K_{34}..L_{55}$ and $M_{20}..M_{30}$ resp. in Figure 23).

Proof. Figure 28(a) lists the only beads exposed and accessible from below above $\text{G}^{\text{Read 1}}$. And Figure 28(b) lists all the possible interactions between them (to the left) and the beads of the brick $\text{G}^{\text{Read 1}}$ zig-folding below (to the right).

According to the rule in Figure 28(b), besides the interactions at the 82 first beads at the very top-leftmost part of $\text{G}^{\text{Read 1}}$ ($K_{34}..L_{55}$ in Figure 23, interactions in green in Figure 28(b)) and the 11 beads
Figure 27 The ϕ-rule between the beads accessible from below of brick D►Write and the beads that will get in touch with them from all the modules Zag-folding below.
(a) The beads accessible when the brick $G_{\text{Read}}$ zig-folds itself.

(b) The $\bullet$-rule for the beads accessible by the beads in $G_{\text{Read}}$ as it zig-folds.

(c) The closest bead $L74$ in brick $G_{\text{Read}}$ can get from one bead $L82$ above (case $n = 1 \mod 3$).
Note that each bead makes exactly one bond, with a bead of the same shade, red, blue, yellow or green (see Figure 22 and 23) and of the same rank within the shade.

at the very end of $G\xrightarrow{\text{Read}1}$ (M20. M30 in Figure 23, interactions in blue in Figure 28(b)), the only possible interaction between $G\xrightarrow{\text{Read}1}$ and the already present beads above it is: $L82\xrightarrow{\text{L74}}$. But $L74$ appears only once in $G\xrightarrow{\text{Read}1}$, at coordinates $(w + 10 + 4k, 1 - h)$ (see Figure 23), while $L82$ appears above $G\xrightarrow{\text{Read}1}$ at coordinates $(w + 1 + i(w + 6), 2 - h)$ for $i = 0..n$. The minimal $x$-distance between $L82$ and $L74$ is thus $\min_{i = 0..n} |9 + 4k - i(w + 6)|$. But $9 + 4k - i(w + 6) = 9 + 4(n - 1)(w + 6)/6 - i(w + 6) = 9 + 2(n - 1 - 2i)(2(L + P) + 8)$. It follows that the minimum difference in $x$-coordinate between $L82$ and $L74$ is:

\[
\begin{align*}
&= 17 + 2(L + P) \geq 41, \text{ if } n = 0 \mod 3; \\
&= 9, \text{ if } n = 1 \mod 3; \text{ and} \\
&= 1 - 2(L + P) \leq -23, \text{ if } n = 2 \mod 3.
\end{align*}
\]

As a consequence, $L74$ never gets close enough to interact with $L82$ above (see Figure 28(c) for the closest situation). It follows that one only need to take into account the environment for the folding of the top-leftmost and top-rightmost part of brick $G\xrightarrow{\text{Read}1}$, which is done next using proof-trees, the glider between them, zig-folds regardless of the beads above in the environment.

$\textbf{Lemma 15 (G\xrightarrow{\text{Read}1} along F\xrightarrow{\text{ZigUp}}).}$ When $G$ folds into the brick $G\xrightarrow{\text{Read}}$, no bead in $\text{SegExpG}$ can make bonds with the beads in $F\xrightarrow{\text{ZigUp}}$ nearby and thus folds regardless of the beads nearby (as a glider).

$\textbf{Proof.}$ Figure 29 lists the interactions between the beads in $\text{SegExpG}$ and the beads in $\text{SegExpF}$: these are exactly $K(4 + i)\xrightarrow{\text{J}}(24 + i)$ for $i = 0..23$; in particular red-shaded beads $K4..K9$ in $G$ (resp. yellow, $K10..K15$; blue, $K16..K21$; and green, $K22..K27$) can only bond with beads of the same shade $J24..J29$ in $F$ (resp. $J30..J35$; $J36..J41$; $J42..J47$).

As shown on Figure 21 and 22 the $y$-coordinates explored by these beads are as follows when $G$ zig-folds into $G\xrightarrow{\text{Read}0}$ or $G\xrightarrow{\text{Read}1}$:

$\textbf{Red :}$ the $y$-coordinates of beads $J24..J29$ in $F$ belong to $\{-40 - 3j, \ldots, -35 - 3j\}$ for $j \geq 1$, while the corresponding beads $K4..K9$ in $G$ explore $y$-coordinates in $\{-38 - 3j + 1, \ldots, -34 - 3j + 1\}$ for $j' \geq 1$.

$\textbf{Yellow :}$ the $y$-coordinates of beads $J30..J35$ in $F$ belong to $\{-34 - 3j + 1, \ldots, -41 - 3j\}$ for $j \geq 1$, while the corresponding beads $K10..K15$ in $G$ explore $y$-coordinates in $\{-36 - 3j + 1, \ldots, -36 - 3j + 1\}$ for $j' \geq 1$.

$\textbf{Blue :}$ the $y$-coordinates of beads $J36..J41$ in $F$ belong to $\{-40 - 3j + 1, \ldots, -35 - 3j + 1\}$ for $j \geq 1$, while the corresponding beads $K16..K21$ in $G$ explore $y$-coordinates in $\{-38 - 3j, \ldots, -34 - 3j\}$ for $j' \geq 1$. 
Green: the y-coordinates of beads J42, J47 in F belong to \{-34 - 3^{2j+2}, \ldots, -41 - 3^{2j+1}\} for \( j \geq 1 \), while the corresponding beads K2, K27 in G explore y-coordinates in \{-36 - 3^{2j+1}, \ldots, -36 - 3^{2j}\} for \( j' \geq 1 \).

Now, as for all \( j \geq 1 \) (with the notation, \( a \leq b \) iff \( a \leq b - 2 \))

\[
-35 - 3^{2j+2} \leq -38 - 3^{2j+1} \leq -34 - 3^{2j+1} \leq -40 - 3^{2j}
\]
and \( -36 - 3^{2j+1} \leq -34 - 3^{2j+1} \leq -41 - 3^{2j} \leq -36 - 3^{2j} \)
and \( -34 - 3^{2j+2} \leq -40 - 3^{2j+1} \leq -35 - 3^{2j+1} \leq -38 - 3^{2j} \)
and \( -41 - 3^{2j+1} \leq -36 - 3^{2j+1} \leq -36 - 3^{2j} \leq -34 - 3^{2j} \)

none of the (same-shade) interacting beads ever get close enough to each other and the beads in the segment SegExpG folds without making any bond (into a glider), regardless of the beads next to them in F→ZigUp when G zig-folds into brick G→Read.

\section{8.2 Proof-trees}

A proof-tree is a compact representation of the enumeration of all the possible paths the molecule explores as it folds. Figure 30 presents the proof-tree for the folding of G when bouncing on a bump encoding a \( \emptyset \) in G→Read\( \emptyset \). For the sake of readability, several paths are drawn in the same ball when they share the same beginning up to their last bond with the environment; then, as a sanity check, the grey number at the bottom left of the ball indicates how many paths are drawn in this ball. The black number in the top right corner of each ball indicates how many bonds are made by the paths with the environment. The ball(s) with the maximum number of bonds is(are) highlighted in black and go to the next round, together with the balls that place the first bead at the same position.

These proof-trees are automatically generated as the molecule folds. Each environment (surrounding + the three beads currently folding) is given a number (written #xxxx). When an already studied environment is encountered, the proof-tree is stopped, and the next (already encountered) environment number is written, allowing easy navigation in the proof — note that Figure 30 is an excerpt from a larger proof-tree and does not show its beginning nor its end, this is why the navigation tag cannot be observed in this figure.

The complete proof certificates may be found on the website:

https://www.irif.fr/~nschaban/oritatami/prooftrees/

\section*{References}


Figure 30: Excerpt from the proof-tree certificate for the folding of $G$ into $\texttt{G\textasciitilde}$ when bouncing on a spike encoding a $\emptyset$. 


A  Types of blocks

![Diagram of different types of blocks]

- **Seed**: Input = \(010\)  
- **Read0**  
- **Read1**  
- **Halt**

**Figure 31** The different type of blocks. The orange circles locate their anchors on the underlying triangle grid. The orange chevrons shows where they plug into each other. The current row of each block is shaded in white while the previous and the next rows are shaded in blue in the underlying triangular grid.

B  Geometry of the blocks

The following figures 32–39 describe the geometry of each block (except for the **Read** blocks presented in Figure 5). Note that they display an idealized version of the real path inside them, omitting details (mainly, socks) that are vital for computing but irrelevant to the block general geometry – see Section 7 for the exact geometry of each brick inside each block.
Figure 32 Geometry of the Seed block. This block encodes the initial word so that the oritatami system simulates properly the corresponding tag system. It consists of placing the different letter at the expected Write positions. Its rightmost part consists in a northeast-bound segment signalling the end of the (initial) word. Its leftmost part ends at the position \((0, 0)\) where the molecule will start folding the first zig-row.
Figure 33 Geometry of the Copy blocks. The Copy0 and Copy1 blocks have both the shape of a parallelogram with horizontal side length $W$ and vertical side length $h$. For both, the next block will start folding at the top right corner, at $(W,0)$. Note that the Copy0 and Copy1 blocks have identical internal structure apart from the line joining the two red areas at $(w+3,0)$ and $(h+w+2,h)$. Indeed, when folding, the part of the molecule located in the red area, either: (1) detects a spike on top (encoding a 0) and then folds into a dent on top which induces spike at the bottom (copying the 0 below, the block Copy0); or (2) folds flat (encoding a 1) on top which induces a flat folding at the bottom, copying the 1 from the top to the bottom of the Zig-row (the block Copy1).

(a) The Copy0 block has a dent (an empty position) located at $(w+3,0)$ (w.r.t. to its origin at the top left corner), in which plugs the spike of the block from the row above it, and which induces (when folding) a spike at the bottom at $(h+w+2,h)$, copying the letter 0 from the top to the bottom of the Zig-row.

(b) The Copy1 block is flat at $(w+3,0)$ (w.r.t. to its origin at the top left corner), which, aligned with a flat block above (encoding a 1), indues (when folding) a flat bottom at $(h+w+1,h-1)$, copying the letter 1 from the top to the bottom of the Zig-row.
Figure 34 Geometry of the ▶ Copy blocks. The ▶ Copy0 and ▶ Copy1 blocks are the horizontal mirror images of the Copy0► and Copy1► blocks (see Figure 33).

(a) The ◀ Copy0 block is the horizontal mirror image of the Copy0► block (see Figure 33(a)).

(b) The ◀ Copy1 block is the horizontal mirror image of the Copy1► block (see Figure 33(b)).
Figure 35 Geometry of the \textit{CopyLineFeed} blocks. These blocks adopt the shape of a \((W - 6) \times h\)-parallelogram prolonged by a southwestbound 'arm' hoping to the beginning of the next zig-row. Folding from right to left, the \textit{CopyLineFeed} blocks are identical to the \textit{Copy} blocks until position \((-W + 6, 0)\) where it detects that there are no more blocks (encoding letter) in the row above (the detection of the absence of a block on top is made possible by the \(\Delta = 7\) horizontal offset between the zig- and zag-rows). Then, instead of completing a parallelogram, the end of the \textit{CopyLineFeed} blocks is attracted upwards and then folds into a southwestbound glider pattern to reach the opening position of the next zig-row. The next block will start folding at \((-W + 8, 2h - 1)\).

(a) The \textit{CopyLineFeed0} block proceeds as \textit{Copy0} to copy the spike encoding a \(\emptyset\) from the row above to the row below. It has a dent (an empty position) at \((-W + w + 9, 0)\) in which plugs the spike (encoding a \(\emptyset\)) of the block above. When folding, this dent induces a spike at the bottom at position \((-W + w + 10, h)\) w.r.t. to the origin of the block. Note that the spike below is at position \((w + 2, -h + 1)\) w.r.t. to the beginning of the following block, which is consistent with the position of the dent in the \textit{Read0} block (see Figure 5(a)).

(b) The \textit{CopyLineFeed1} block.
Figure 36 Geometry of the Append\&OR(u) blocks. The folding into this block is triggered by the absence of a block in row above (indicating the end of the word). It has one northeastbound 2-beads wide arm climbing along the east side of the block in the row above then a southeastbound 4-beads wide arm stopping at the bottom of the current zig-row. Then, the block consists in an 3-beads wide |u|W-beads long eastbound glider path going along the bottom of the current zig-row and encoding each letter of u: the path contains a spike (below, and a dent on top) for each \( u_j = 0 \) at position \((jW + w + h + 2, h)\) (1s are encoded by the absence of spike). It then expanded up to position \((c - 1, h - 1)\) where \( c = \frac{3}{2}W(L - |u| + P) + 2h - 4 \) and go back to its origin and grows a 10-beads wide h-beads high southwestbound arm opening the next zag-row to end at the position \((|u|W + h - 8, h)\), at the top right corner of the upcoming zag-row. The next block will start at \((|u|W + h - 7, h)\).
Figure 37 Geometry of the Append&CR$(\epsilon)$ block. This block is the special case of Figure 36 where $u = \epsilon$. It is given for clarity.
Figure 38 Geometry of the Carriage Return & Line Feed & Halt block. This block is identical to the Append & CR & LF block until it reaches position \((h, h)\). Then, when folding, it detects the absence of a block above which indicates that the current word is empty. It then folds as the leftmost part of the \(\llcorner\) Copy\(\llcorner\) line feed blocks (see Figure 35) to open a new zig-row at \((h+1, 2h-1)\). It then goes up to \((h+3, 2h+1)\). And as there are no block on the zag-row above, it is attracted inside itself and gets blocked at \((h, 3h-4)\).
Figure 39 Geometry of the HALT block. This block appears at the end of the computation. It starts as a Read block with a 3-beads wide \( h \)–beads high southeastbound glider until it reaches position \((2,1-h)\). But, as there are no block in the zag-row above, the next beads are attracted to the left and the construction stops there.

\[ \text{HALT} \]

\[(2,1-h)\]

\(z\)

\(y\)

\(x\)

C The complete rule

We first gives the rule in text. Fig. 40 displays it as a matrix.
C. Geary, P.-É. Meunier, N. Schabanel, S. Seki

XX:57
Proving the Turing Universality of Oritatami Co-Transcriptional Folding
Proving the Turing Universality of Oritatami Co-Transcriptional Folding
**D Enumeration of the environments together with their proof-trees**

The following tables refer to the proof-trees on the website:

https://www.irif.fr/~nschaban/oritatami/prooftrees/

proving the correctness of the folding of our design in every possible surroundings.
Figure 40 The rule of the SCTS Oritatami system: in this diagram, we have $b \bullet b'$ iff there is a bullet $\bullet$ at the intersection of one the two lines coming from $b$ and from $b'$; for instance, we have $A0 \bullet A2$ but not $A0 \bullet A5$. 

XX:62 Proving the Turing Universality of Oritatami Co-Transcriptional Folding