Oritatami: A Computational Model for Molecular Co-Transcriptional Folding

Cody Geary, Pierre-Etienne Meunier, Nicolas Schabanel, Shinnosuke Seki

International Journal of
Molecular Sciences

MDPI

Article

Oritatami: A Computational Model for Molecular Co-Transcriptional Folding

Cody Geary [1], Pierre-Étienne Meunier [2], Nicolas Schabanel [3,*] and Shinnosuke Seki [4]

[1] Computer Science Computation and Neural Systems Bioengineering Caltech, MS 136-93, Moore Building, Pasadena, CA 91125, USA; codyge@gmail.com
[2] Computer Science Dept, Hamilton Institute, Maynooth University, Co. Kildare, Ireland; pierre-etienne.meunier@mu.ie
[3] CNRS, École normale supérieure de Lyon (LIP), CEDEX 07, 69364 Lyon, France
[4] Computer and Network Engineering Dept, University of Electro-Communications, 1-5-1, Chofugaoka, Chofu, Tokyo 1828585, Japan; s.seki@uec.ac.jp
* Correspondence: nicolas.schabanel@ens-lyon.fr

Received: 15 April 2019; Accepted: 30 April 2019; Published: 7 May 2019

Abstract: We introduce and study the computational power of Oritatami, a theoretical model that explores greedy molecular folding, whereby a molecular strand begins to fold before its production is complete. This model is inspired by our recent experimental work demonstrating the construction of shapes at the nanoscale from RNA, where strands of RNA fold into programmable shapes during their transcription from an engineered sequence of synthetic DNA. In the model of Oritatami, we explore the process of folding a single-strand bit by bit in such a way that the final fold emerges as a space-time diagram of computation. One major requirement in order to compute within this model is the ability to program a single sequence to fold into different shapes dependent on the state of the surrounding inputs. Another challenge is to embed all of the computing components within a contiguous strand, and in such a way that different fold patterns of the same strand perform different functions of computation. Here, we introduce general design techniques to solve these challenges in the Oritatami model. Our main result in this direction is the demonstration of a periodic Oritatami system that folds upon itself algorithmically into a prescribed set of shapes, depending on its current local environment, and whose final folding displays the sequence of binary integers from 0 to $N = 2^k - 1$ with a seed of size $O(k)$. We prove that designing Oritatami is NP-hard in the number of possible local environments for the folding. Nevertheless, we provide an efficient algorithm, linear in the length of the sequence, that solves the Oritatami design problem when the number of local environments is a small fixed constant. This shows that this problem is in fact fixed parameter tractable (FPT) and can thus be solved in practice efficiently. We hope that the numerous structural strategies employed in Oritatami enabling computation will inspire new architectures for computing in RNA that take advantage of the rapid kinetic-folding of RNA.

Keywords: natural computing; self-assembly; molecular folding

1. Introduction

The process by which one-dimensional sequences of nucleotides or amino-acids acquire their complex three-dimensional geometries, which are key to their *function*, is a major puzzle of biology today. Understanding molecular folding will not only shed light on the origin and functions of the molecules existing in nature, it will also enable us to *control* the process more finely, and engineer artificial molecules with a wide range of uses, from performing missing functions inside living organisms, to producing precisely targeted drugs.

Biomolecular nano-engineering includes DNA self-assembly, which gave rise to an impressive number of successful experimental realizations, from arbitrary 2D shapes [1] to molecule cyclic machines [2], or counters [3]. First pioneered by Seeman [4], DNA nanotechnologies only really started to take off once a computer science model was devised by Winfree [5] to *program* molecular self assembly in a computer science way.

Since then, many models have been designed to refine different features of experiments: hierarchical self-assembly [6,7], modeling the absence of a *seed*, kinetic tile assembly [5,8], 3D and probabilistic tile assembly [9], among others.

However, the potential applications of this type of DNA technology inside cells are limited by the high thermal stability of the DNA duplex, where consequently DNA nanostructures are typically formed through the process of *annealing*: Namely, by heating the molecules up to high temperatures in a precisely controlled environment and then cooling them down at a precisely controlled rate. This assembly paradigm allows researchers to design DNA structures with amazing precision by taking advantage of computational techniques to optimize the thermodynamics of strand-strand interactions, see for instance [10].

However at the same time the method requires precise control over temperature and other variables that are not always possible to control in every environment, such as for example within living cells. Other biopolymers such as RNA and proteins do not share this limitation, as nature uses both mediums to produce exquisite structures on a continuous basis. Researchers have successfully assembled functional nanoparticles and scaffolds out of these alternatives [11–14]. However, their assembly process is harder for us to program: while the shape that folds depends on both the sequence and the environment, the sequence is read *linearly* and expressed regardless of the environment. This contrasts with more classical programming models such as Turing machines, or even tile assembly, which are able to *jump* to different parts of the program depending on the input. Another important difficulty is that even predicting the final folded shape of a biopolymer given the sequence is still the center of active research, especially for proteins [15–19].

In particular, a large body of computer science literature is focused on energy optimization, one of the main drivers of folding. For example, in different variants of the *hydrophobic-hydrophilic* (HP) model [20], it has been shown that the problem of predicting the most likely geometry (or *configuration*) of a sequence is NP-complete [21–26], both in two and three dimensions, and in different variants of the model.

The *kinetics* of folding, which is the step-by-step dynamics of the reaction, has been demonstrated by biochemists to play a major role in the final shape of molecules [27], and even play a *prevalent role* at the heart of the mechanism of RNA switches in biology [28]. In recent experimental results, researchers have even been able to *control* this mechanism to engineer various things out of RNA represented by the RNA origami architecture for single-stranded rectangular tiles [29], and followed by more complex structural motifs [30] and RNA-origami-based FRET system [31].

This paper introduces a new model of computation and molecular folding inspired by RNA folding, intended to capture the kinetics of folding and model the experiments in [29]. In particular, it focuses on the *co-transcriptional* nature of RNA folding, whereby molecules fold concurrently while being transcribed (see Figure 1): in computer science terms, the folding process is a *local energy optimization*, or otherwise put, a *greedy algorithm*. A key limitation of this model is that the strand folds irreversibly once beads are locked into place, a simplification that makes working with the model tractable, but that also eliminates an important aspect of natural RNA folding—the possibility for structural rearrangement.

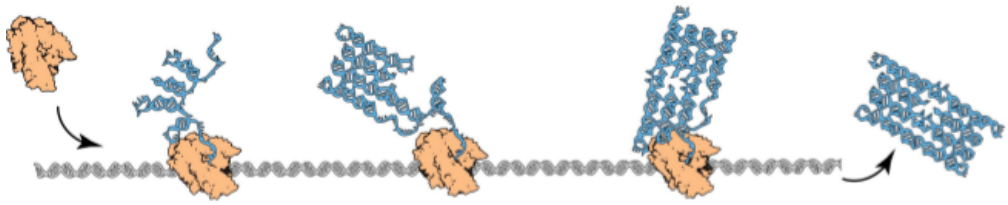


Figure 1. An RNA molecule folding over itself while being transcribed, as the experiments in [29].

The first experimental results have used a standard benchmark: making simple shapes, such as squares (as shown for instance on Figure 2). With the new model introduced in this paper, our goal is twofold: first, explore the engineering possibilities of this mechanism, in order to make arbitrary shapes and structures. Then, the other aim of our study is to understand the complexity of sequence operations, to understand the computational processes which led to the creation of complex molecular networks.

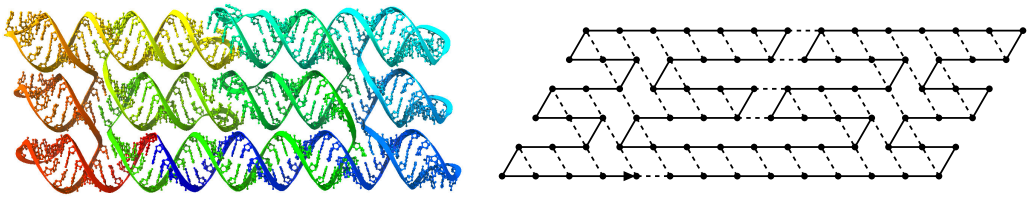


Figure 2. The design of a square, co-transcriptionally folded with RNA, and the corresponding path on the triangular lattice.

1.1. Main Contributions

In our model, called Oritatami, we consider a sequence of "beads", which are abstract basic components, standing for nucleotides or even sequences of nucleotides (also called *domains*). In oritatami, only the latest produced beads of the molecules are allowed to move in order to adopt a more favorable configuration. The folding is driven by the respective attraction between the beads.

Our main construction is a *binary counter*. Counters are an essential component of many sophisticated constructions in biological computing, in particular in tile assembly [32,33]. Counters are also an important benchmark in experiments [3].

Theorem 1. *There is a fixed periodic sequence* $0, 1, \ldots, 59, 0, 1, \ldots$ *of period 60 whose rule is given in Figure 3, which, when started from a seed encoding an integer* x *in binary with at most* $2k + 1$ *bits for some* k*, folds into a structure encoding* $x + 1$, $x + 2$, $\ldots$, $2^{2k+1} - 1$*, on the successive rows of the triangular grid.*

We *prove the correctness* of this construction by designing an abstract module system to handle the complexity of the base mechanism of the model, which is about as low-level as assembly code in more standard computing models.

We then show a generic construction method in this model, which we applied to automate parts of the design of the counter. Moreover, this result helps understanding the computational complexity of sequence programming. Precisely, we prove two results in this direction:

Theorem 2. *Designing a single sequence that folds into different target shapes in a set of surrounding environments, is NP-complete in the number of environments.*

More surprisingly, it turns out that there is an algorithm to solve this problem in time *linear in the length of the sequence*. This algorithm is also practical, as we were able to use it to find sequences for our main construction:

Theorem 3. *The sequence design problem is FPT with respect to the length ℓ of the sequence: there is an algorithm linear in ℓ (but exponential in the number of environments) to design a single sequence that folds into the target shapes in the given environments.*

As Winfree's thesis [5] on the very abstract tile assembly model did inspire many successful experimental works (see for instance [3,10,34–36]), we hope that the numerous structural strategies employed in Oritatami enabling computation will inspire new architectures for computing in RNA that take advantage of the rapid kinetic-folding of RNA.

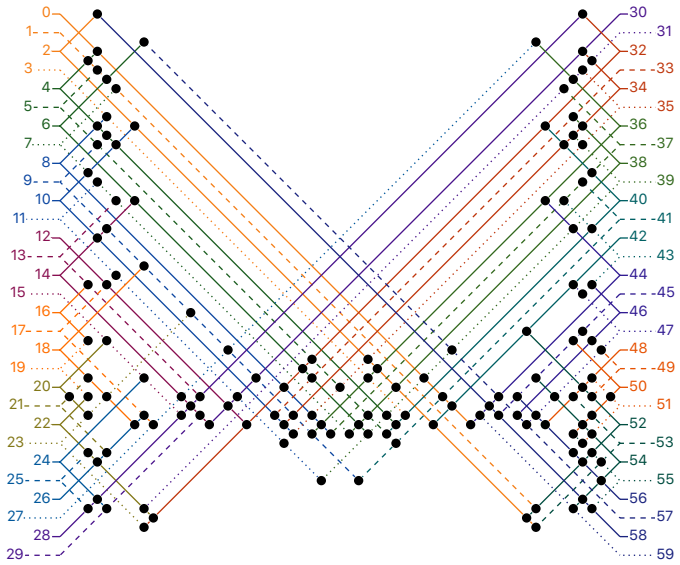


Figure 3. *The rule 💗 of the Counter oritatami system*: in this diagram, we have b 💗 b' iff there is a bullet • at the intersection of one the two lines coming from b and from b'; for instance, we have 4 💗 8 but not 4 💗 7.

1.2. Related Work

The oritatami system has received considerable attention since its proposal in the conference abstract of this paper [37]. As mentioned above, counters serve as a key component in tile self-assembly, especially for assembling shapes. Masuda et al. implemented a small oritatami system that serves as a 4-state automaton with output and integrated it with our binary counter (Theorem 1) into a larger-scale oritatami system towards the self-assembly of an arbitrary finite portion of Heighway dragon fractal [38]. Note that our binary counter works under inertial dynamics and they proposed a modified implementation working under oblivious dynamics with a different ratio of transcription speed to folding (technically speaking, ours works with delay 4 while theirs does with delay 3; delay is formally defined in Section 2). This illustrates the flexibility of oritatami designs. The transcript of their system is periodic as our binary counter; the period is linearly proportional to the size of the target portion. Demaine et al. [39] and Han and Kim [40] independently conducted more comprehensive studies of the shape self-assembly by oritatami systems recently. In particular, Demaine et al. proved that there is a universal set of 114 bead types with a rule set with which an arbitrary finite shape can be folded by an oritatami system, as long as the shape is scaled-up by a small factor.

The FPT algorithm (Theorem 3) has brought about useful modules not only for the binary counter but also for subsequent oritatami systems including the above-mentioned Heighway dragon fractal assembler, Satisfiability (SAT) tautology checker [41], and the time-efficient universal Turing machine simulation in [42]. The construction in [42] is the most intricate oritatami system implemented so far, developing much further the paradigms for co-transcriptional molecular programming introduced here.

Other variants of the rule design problem have been investigated by [43,44]. An alternative heuristic to the FPT algorithm to optimize the rule set has been proposed by Han and Kim [45].

2. Model and Main Results

Given two words $a, b \in B^*$, we denote by ab their concatenation.

2.1. Model

2.1.1. Oritatami System

Oritatami is about the folding of finite sequences of beads, each from a finite set B of *bead types*, using an attraction rule 💗, on the triangular lattice graph $\mathbb{T} = (\mathbb{Z}^2, \sim)$ where $(x,y) \sim (u,v)$ if and only if $(u,v) \in \{(x-1,y),(x+1,y),(x,y+1),(x+1,y+1),(x-1,y-1),(x,y-1)\}$.

A *configuration* c of a sequence $w \in B^*$ is a self-avoiding path of length $n = |w|$ labelled by w in $\mathbb{T}$, i.e., a path whose vertices $c_1, \ldots, c_n$ are pairwise distinct and labelled by the letters of w. We denote by $\mathrm{bt}(c_i) = w_i$ the bead-type of the i-th bead of c. A *partial configuration* of a sequence w is a configuration of a prefix of w. For any partial configuration c of some sequence w, an *elongation* of c by k beads is a partial configuration of w of length $|c| + k$. We denote by $\mathcal{C}_w$ the set of all partial configurations of w (the index w will be omitted when the context is clear). We denote by $c^{\rhd k}$ the set of all elongations by k beads of a partial configuration c of a sequence w and by $c^{\lhd k}$ the singleton containing the prefix of length $|c| - k$ of c.

An *oritatami system* $\mathcal{O} = (p, 💗, \delta)$ is composed of (1) a (possibly infinite) *transcript* p, which is a sequence of *beads*, of a type chosen from a finite set B, (2) an *attraction rule*, which is a symmetric relation $💗 \subseteq B^2$ and (3) a parameter δ called the *delay time*.

Given an attraction rule 💗 and a configuration c of a sequence w, we say that there is a *bond* between two adjacent positions c_i and c_j of c in $\mathbb{T}$ if $w_i \, 💗 \, w_j$. The *number of bonds* in a configuration c of w, written $\mathrm{E}(c)$, is the negation of the number of bonds within c: formally,

$$\mathrm{E}(c) = -|\{(i,j) : c_i \sim c_j,\ j > i+1,\ \text{and } w_i \, 💗 \, w_j\}|.$$

2.1.2. Oritatami Dynamics

A *dynamics* for a sequence w is a function $\mathscr{D}_w : 2^{\mathcal{C}_w} \to 2^{\mathcal{C}_w}$ such that for all subset S of partial configurations of length ℓ of w, $\mathscr{D}(S)$ is a subset of the elongations by one bead of the partial configurations in S (thus, partial configurations of length $\ell + 1$).

Given an oritatami system $\mathcal{O} = (p, 💗, \delta)$ and a *seed configuration* σ of a seed sequence s of length ℓ, a dynamics $\mathscr{D}_{sp}$ gives at each time step the set of *favored nascent configurations* of transcript p, as a function of the set of favored nascent configurations at the previous time step. Initially, the set of favored nascent configurations is $\sigma^{\rhd(\delta-1)}$, that is all the possible elongations of the seed configuration by $\delta - 1$ beads. Then, the set of favored nascent configurations at step t is $\mathscr{D}^t_{sp}(\sigma^{\rhd(\delta-1)})$, that is the set of all elongations by $(t + \delta - 1)$ beads of the seed configuration prolongated by the transcript according to dynamics $\mathscr{D}$.

We explore greedy folding dynamics where only the most recently transcribed beads can move, all other beads remain in place. These still unsettled beads are said *nascent* and their number is controlled by the integer parameter δ (in most of this article, $\delta \leqslant 4$). We consider two different dynamics to model the "greedy" nature of the process:

The inertial dynamics was also called *hasty dynamics* in a preliminary version of this work [37]. It does not question previous choices but chooses the energy-minimal positions for the δ nascent beads among all elongations of the previously adopted partial configurations. It lets the $\delta - 1$ already placed nascent beads where they are and abandons the extension of a configuration if no

extension with the newly transcribed bead allows to reach a lowest energy configuration available for the δ nascent beads.

Formally, given a set of currently favored nascent configurations, $\mathscr{I}$ elongates each of them by one bead, and keeps the elongated configurations that have minimum energy among those who share the same prefix of length $|\sigma| + t$:

$$\mathscr{I}(S) = \bigcup_{\gamma \in S^{\lhd(\delta-1)}} \operatorname*{arg\,min}_{c \in \gamma^{\rhd\delta} \cap S^{\rhd 1}} \mathrm{E}(c)$$

The oblivious dynamics is *oblivious* in the sense that it consists of always choosing the best available positions for the nascent δ beads regardless of the previously preferred choices, as opposed to $\mathscr{I}$. Formally, $\mathscr{O}$ takes a set of currently favored nascent configurations, removes the last $\delta - 1$ positions from all of them, and selects the minimal energy configurations among all of their elongations by δ beads. Precisely:

$$\mathscr{O}(S) = \bigcup_{\gamma \in S^{\lhd(\delta-1)}} \operatorname*{arg\,min}_{c \in \gamma^{\rhd\delta}} E(c)$$

Oritatami systems seem less governable under the oblivious dynamics than under the inertial dynamics, as a larger number of configurations are to be considered for energy minimization under the oblivious dynamics and also as previously discarded configurations may be reconsidered later on.

With the exception of Section 5, in this article, we choose the *inertial dynamics*. Our binary counter is described for the inertial dynamics in Section 3. Many subsequent articles [38–46] chose the oblivious dynamics. Masuda et al. [38] adapted our binary counter to the oblivious dynamics and used it as a component of their oritatami system folding arbitrary finite portion of Heighway dragon fractal.

Determinism, Halt and Resulting Folding. An oritatami system $\mathcal{O} = (p, \heartsuit, \delta)$ is *deterministic* for dynamics $\mathscr{D}$ and seed σ of sequence s if for all $i \geqslant 1$, the position of the i-th bead of p is uniquely determined at time i, i.e., if for all $i \geqslant 1$, $|\{c_{|\sigma|+i} : c \in \mathscr{D}^i_{sp}(\sigma^{\rhd(\delta-1)})\}| = 1$.

We say that $\mathcal{O}$ *stops* at time t with seed σ and dynamics $\mathscr{D}$, if $\mathscr{D}^t_{sp}(\sigma^{\rhd(\delta-1)}) = \varnothing$ and $\mathscr{D}^z_{sp}(\{\sigma\}) \neq \varnothing$ for $z < t$. If $\mathcal{O}$ stops at time t, the set $\mathscr{D}^t_{sp}(\sigma^{\rhd(\delta-1)})$ is the *result* of the folding process; if it consists of a single configuration c, we say that c is the *resulting folding* and that the oritatami system $\mathcal{O}$ *folds deterministically* into configuration c. If the transcript p is finite, the folding process will stop at time $t = |p| - \delta + 1$ or before in case of a geometric obstruction (no more elongation is possible because the configuration gets trapped in a closed area). If the transcript p is infinite, the folding process may only stop because of a geometric obstruction.

Notation for configurations. We will use $a \overset{\mathtt{NW}}{\nwarrow} b$, $a \overset{\mathtt{NE}}{\nearrow} b$, $a \overset{\rightarrow}{\mathtt{E}} b$, $a \overset{\searrow}{\mathtt{SE}} b$, $a \overset{\swarrow}{\mathtt{SW}} b$, $a \overset{\leftarrow}{\mathtt{W}} b$ to denote a configuration consisting of two beads with types a and b where b is placed respectively at the NW, NE, E, SE, SW or W of a. As an example, the configuration in Figure 4 is described as $30 \overset{\mathtt{NW}}{\nwarrow} 31 \overset{\mathtt{NW}}{\nwarrow} 32 \overset{\rightarrow}{\mathtt{E}} 33 \overset{\searrow}{\mathtt{SE}} 34 \overset{\mathtt{NE}}{\nearrow} 35 \overset{\rightarrow}{\mathtt{E}} 36 \overset{\searrow}{\mathtt{SE}} 37 \overset{\leftarrow}{\mathtt{W}} 38 \overset{\swarrow}{\mathtt{SW}} 39 \overset{\rightarrow}{\mathtt{E}} 40 \overset{\rightarrow}{\mathtt{E}} 41$.

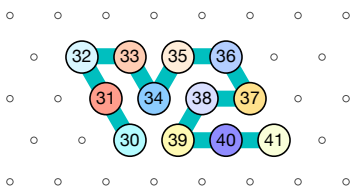


Figure 4. A configuration encoded by $30 \overset{\mathtt{NW}}{\nwarrow} 31 \overset{\mathtt{NW}}{\nwarrow} 32 \overset{\rightarrow}{\mathtt{E}} 33 \overset{\searrow}{\mathtt{SE}} 34 \overset{\mathtt{NE}}{\nearrow} 35 \overset{\rightarrow}{\mathtt{E}} 36 \overset{\searrow}{\mathtt{SE}} 37 \overset{\leftarrow}{\mathtt{W}} 38 \overset{\swarrow}{\mathtt{SW}} 39 \overset{\rightarrow}{\mathtt{E}} 40 \overset{\rightarrow}{\mathtt{E}} 41$.

3. Folding a Binary Counter

3.1. General Idea of The Construction

Our construction works with $\delta = 4$. The counter is implemented by folding the periodic sequence of bead types $0, 1, \ldots, 58, 59, 0, 1, \ldots$ with period 60. Our construction proceeds in zig-zags as the classic implementation of a counter with a sweeping Turing machine whose head goes back and forth between the two ends of the coding part of the tape. Each pass is 3-rows thick and folds each part of the molecule into a parallelogram of size 4×3 or 6×3 except for the last and the first parts of each pass which are folded into parallelograms of size 3×6 to accomplish the U-turn downwards and start the next pass. The *zig pass*, folding three rows at a time from right to left, computes the carry propagation in the current value of the counter. The *zag pass*, folding three rows at a time from left to right, writes down the bits of the newly incremented value, and gets the folding to resume at the right-hand side of the configuration.

The molecule is semantically divided into 4 parts, called *modules*:

- Module A (beads 0–11, in blue in all figures): the First Half-Adder
- Module B (beads 12–29, in red in all figures): the Left-Turn module
- Module C (beads 30–41, in blue in all figures): the Second Half-Adder
- Module D (beads 42–59, in red in all figures): the Right-Turn module

Encoding. The current value of the counter is encoded in standard binary with the most significant bit to the left. Each bit is encoded into a specific folding of the modules A and C of the molecule in the rows corresponding to a zag pass: namely folding A0 and C0 for 0, and A1 and C1 for 1. During the zig pass, the *value of the carry* is encoded by the position of the molecule when it starts to fold Module A or C: `carry` $= 0$ if Module A or C starts to fold in the top row; `carry` $= 1$ if Module A or C starts to fold from the bottom row.

In the zig pass (←), modules A and C "read" from the row above the value encoded into the folding in the row above during the previous zag-phase (or in the seed configuration for the first zig pass), and fold into a shape (called a *brick*, see Section 4.1) A00, A10, A01, A11 or C00, C10, C01, C11 accordingly where Axc is the brick corresponding to the case where x is the bit read in the row above and c is the carry. In the zig pass, modules B and D just propagate the carry value (0 or 1, i.e., start from top or bottom row) output by the preceding module A or C to the next.

When the zig pass reaches the leftmost part of the row on top, the beads there forces the module B to adopt the Left-turn shape which reverses the folding direction and starts the next zag pass.

In the zag pass (→), modules A and C "read" the bricks above Axc or Cxc and folds into the bricks that encodes the corresponding bits, namely Ay or Cy where $y = (x + c) \mod 2$. There are no carry propagation and all the modules B and D fold into the same brick B2 or D2 in this pass.

When the molecule reaches the rightmost part of the row on top of it, the special beads there force the module D to fold into the Right-turn brick which reverses the folding direction and starts the next zig pass.

3.2. The First Two Passes of The Folding

Let's run the first passes of the 3 bits counter to get acquainted with the process.

The seed configuration is shown in Figure 5. The seed configuration for the $(2k + 1)$-bit counter is composed of $4k + 3$ parts:

- The first part $20 \overset{\searrow}{\text{SE}} 21 \overset{\searrow}{\text{SE}} 26 \overset{\searrow}{\text{SE}} 27 \overset{\rightarrow}{\text{E}} 28 \overset{\rightarrow}{\text{E}} 29$, made of beads from Module B, encodes a sequence that will trigger the carriage return at the end of the next zig pass.
- The central part consists in k repetitions of the same sequence of 4 patterns, plus an extra repetition of the first pattern at the end (the central part consists thus in $4k + 1$ parts in total):

  - 30 $\overrightarrow{\text{E}}$ 39 $\overrightarrow{\text{E}}$ 40 $\overrightarrow{\text{E}}$ 41 encoding a bit 0 using beads from Module C,
  - followed by 42 $\overrightarrow{\text{E}}$ 47 $\overrightarrow{\text{E}}$ 48 $\overrightarrow{\text{E}}$ 53 $\overrightarrow{\text{E}}$ 54 $\overrightarrow{\text{E}}$ 59 encoding nothing but padding using beads from Module B,
  - followed by 0 $\overrightarrow{\text{E}}$ 9 $\overrightarrow{\text{E}}$ 10 $\overrightarrow{\text{E}}$ 11 encoding a bit 0 using beads from Module A,
  - followed by 12 $\overrightarrow{\text{E}}$ 17 $\overrightarrow{\text{E}}$ 18 $\overrightarrow{\text{E}}$ 23 $\overrightarrow{\text{E}}$ 24 $\overrightarrow{\text{E}}$ 29 encoding nothing but padding using beads from Module D.

  Note the symmetry by a shift of 30 of the beads values in the patterns involving Modules A and C, and Modules B and D.
- The last part 42 $\overrightarrow{\text{E}}$ 48 $\overrightarrow{\text{E}}$ 50 $\swarrow_{\text{SW}}$ 51 $\overleftarrow{\text{W}}$ 52 $\searrow_{\text{SE}}$ 53 $\overrightarrow{\text{E}}$ 54 $\overset{\text{NE}}{\nearrow}$ 55 $\searrow_{\text{SE}}$ 56 $\searrow_{\text{SE}}$ 57 $\overleftarrow{\text{W}}$ 58 $\overleftarrow{\text{W}}$ 59, made of beads from Module D, encodes a sequence that will first initiate the next zig pass and later trigger the carriage return at the end of the next zag pass.

Note that the seed configuration ends at the bottom row of the upcoming zig pass, which encodes thus that initially the carry is 1.

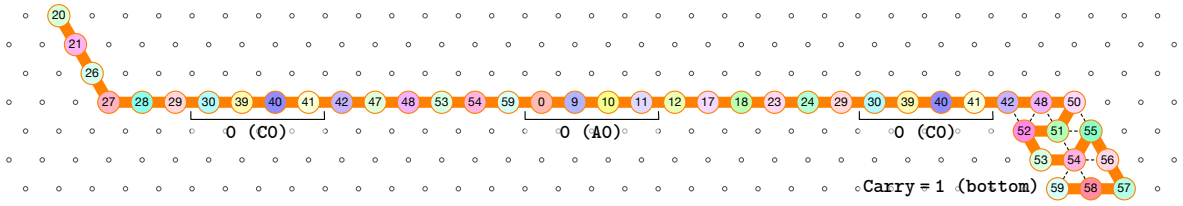


Figure 5. The seed configuration for the 3-bits counter encoding the three bits 000 as the initial value of the counter.

The first zig pass (←). Each zig pass starts with a carry equal to 1, i.e., starts folding from the bottom row. In the first zig pass, the first module A (see Figure 6) folds into the brick A01, encoding the bit $1 = 0 + 1$ with no carry propagation, as a consequence of the carry being 1 and of reading the first bit, 0, from the seed above. Note that the folding A01 ends at the top row, encoding that the carry is now 0. The reading of the bit from the seed is accomplished by the way the module binds to the seed which shapes the module accordingly as we will see in details in Section 3.3.

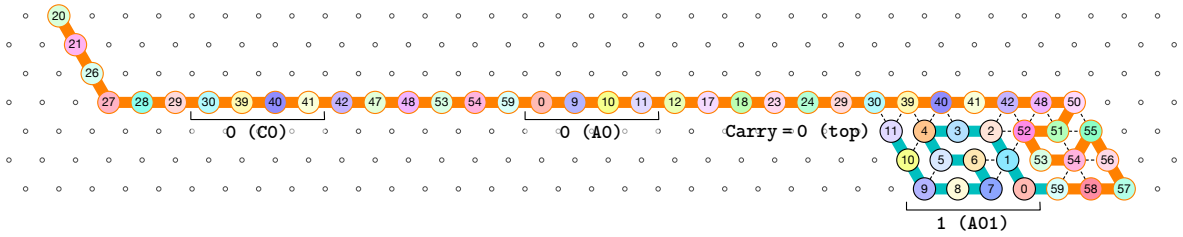


Figure 6. The folding of the first module, A: starting with a carry 1, encoded by the position of the first bead (on the bottom row), this module "reads" a 0 from the seed by binding to the seed, and folds into A01, encoding a 1 with no carry propagation, as encoded by the position of the last bead (on the top row of the module).

Then, as illustrated in Figure 7, the next modules B, C, D, and A fold into shapes B0, C00, D0 and A00 respectively: B0 and D0 meaning that no carry is propagated (they start and end on the top row); and C00 and A00 meaning that the (input) carry is 0 and the bit read from the seed is 0.

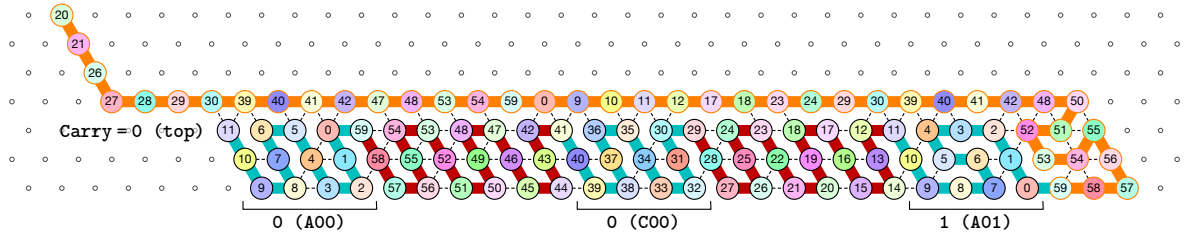


Figure 7. The folding of the central part of the first zig pass in the 3-bits counter.

Finally, as illustrated in Figure 8, the last module, B, of the zig pass binds to the 3-beads long carriage-return pattern at the leftmost part of the seed and folds into the shape BT conducing the molecule to go down by 6 rows, reverse direction and start the following zag pass. Note that the bottom of the shape BT contains the exact same carriage-return pattern.

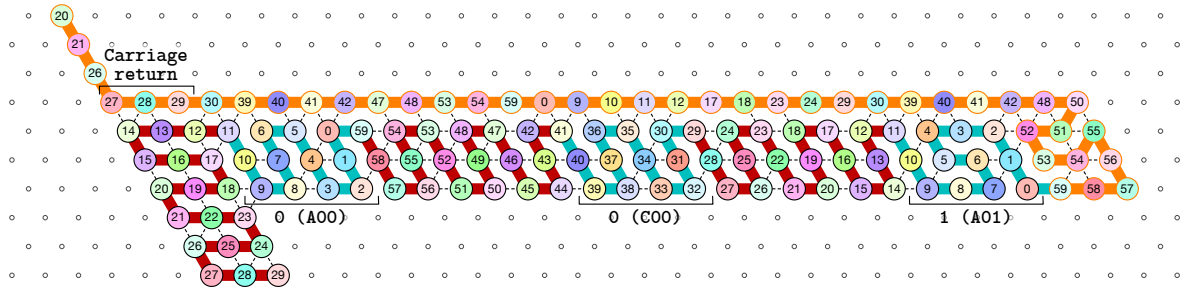


Figure 8. In our construction, the leftmost three beads of any configuration are different from the other beads the left U-turn module binds to inside the zig or zag pass: when the left U-turn module folds next to these bead types, it "triggers" the production of an actual U-turn.

The first zag pass (→). The zag pass is mostly a normalization pass as illustrated in Figures 9 and 10. It progresses from left to right and computes the new value of each bit by rewriting each shape A00 and A11 as C0, C00 and C11 as A0, A10 and A01 as C1, and C10 and C10 as A0. Shapes A0 and C0 encode 0, and Shapes A1 and C1 encode 1, both to be read during the upcoming zig pass. Modules B and D just fold into the shapes B2 and D2 respectively, encoding nothing but padding.

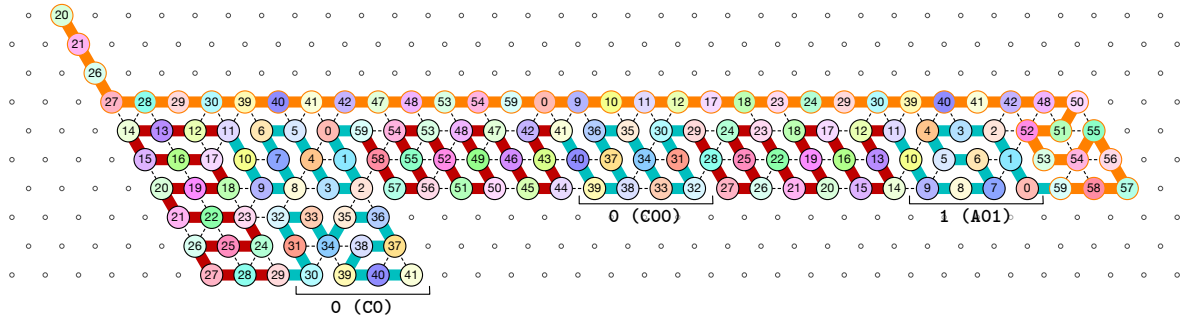


Figure 9. During the zag pass, all modules start from the bottom row, computing the value of each new bit by rewriting shapes A00 and A11 as C0, C00 and C11 as A0, A10 and A01 as C1, and C10 and C10 as A1.

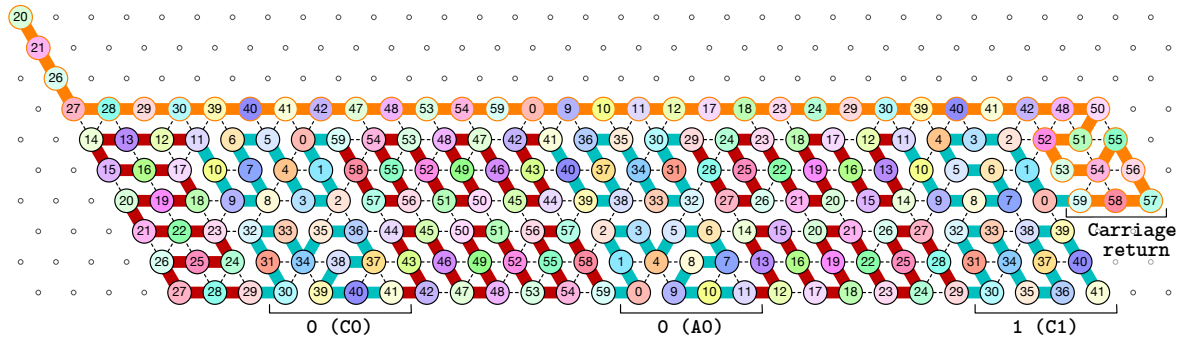


Figure 10. At the end of the first zag pass, the new value of each bit have been encoded into shapes: A0 or C0 for bits equal to 0, A1 or C1 for bit equal to 1.

Finally, as illustrated in Figure 11, the last module, D, of the zag pass binds to the 3-beads long carriage-return pattern in the rightmost part of the seed and folds into the shape DT conducing the molecule to go down by 6 rows, reverse direction and start the next zig pass. Note that, as for the shape BT, the bottom of the shape DT contains the exact same carriage-return pattern.

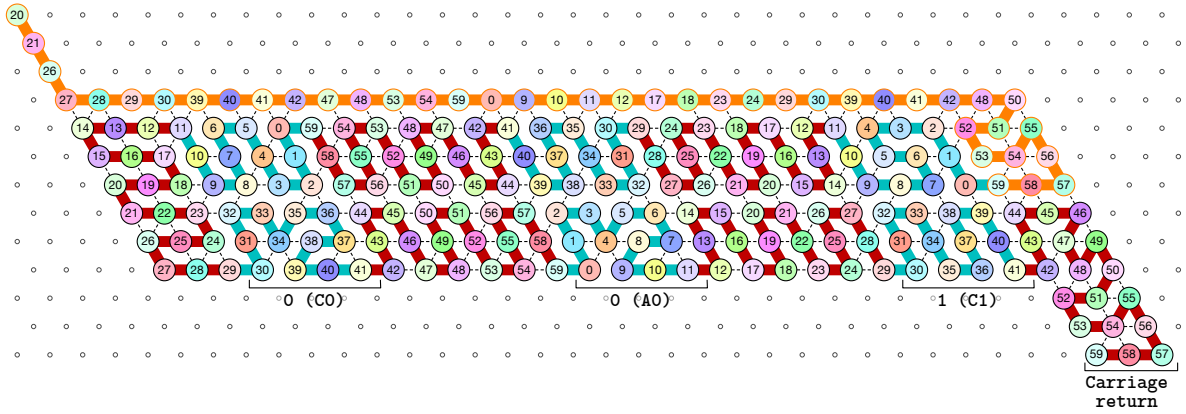


Figure 11. Finally, at the end of the first zag pass, the last module D binds to the carriage-return pattern in the seed and fold into the shape DT to accomplish the right U-turn from which the next zig pass can start.

Figure 12 shows the 3-bits counter folded upto the value $3 = 011$ in binary. One can observe the shape A11 in the second zig pass. A11 corresponds to reading a 1 with a carry 1 which propagates the carry: indeed, the folding ends at the bottom row which propagates the carry to the next module C which folds into C01 as it reads a 0 from above with carry 1. Note that shape A11 is then rewritten as C0 in the following zag pass below.

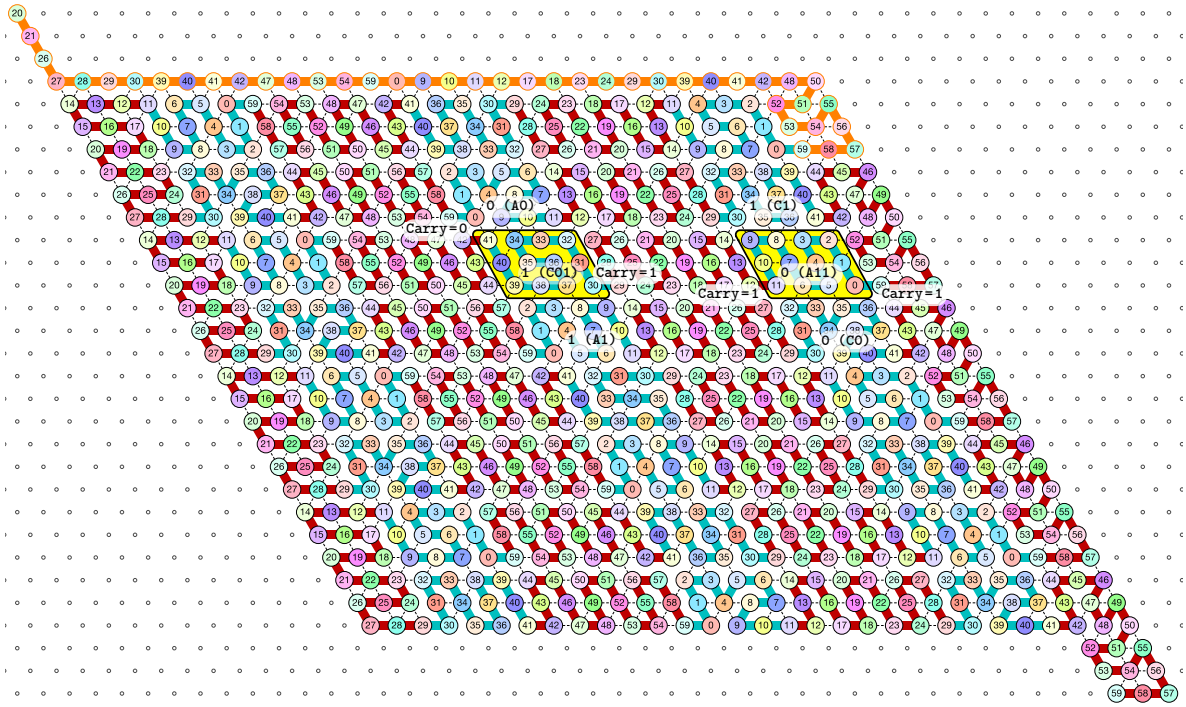


Figure 12. The folding of the 3-bits counter upto value 3 = 011 in binary. Observe the carry propagation in the second zig pass.

3.3. How Does Computation Take Place: Modules, Functions, States and Environment

Each module A, B, C or D implements various "functions" that are "called" when the molecule is folded. Which function is called depends on two things:

the current "state" of the molecule: here, the *state* is whether the carry is 0 or 1. As mentionned earlier this is encoded in the position of the molecule when the module starts to fold: it starts in the top row if the carry is 0; in the bottom row if the carry is 1.

the local environment of the molecule: the *environment*, i.e., the beads already placed around the current area where the folding takes place, acts as the memory in the computation.

The position where the folding of a module starts, determines which beads of a given module will be exposed to and interact with the environment. Then, by creating bonds (or not) with the environment, each module will adopt a specific shape. Therefore, the possible binding schemes will be different depending on this initial position. Similarly, depending on the beads already placed in the environment, the part of the module exposed to it will adopt one form or another depending on how many bonds it can create with the environment. Adopting the language of computer science: the position at which a module starts to fold, determines which *function* of the module is called; the function then *reads the input* encoded by the beads already placed in the environment.

Figure 13 provides a precise description on how the function of the Half-Adder Module C are implemented in the zig pass. As the zig pass goes from right to left, the figure is meant to be read from right to left. In the zig pass, Module C implements two functions: (1) Add 1 to the bit above and propagate the carry if needed, or (2) Copy the bit above unchanged. Add is called if the carry is 1 at the beginning of the folding and Copy is called if the carry is 0. The following step-by-step description of the folding explains how:

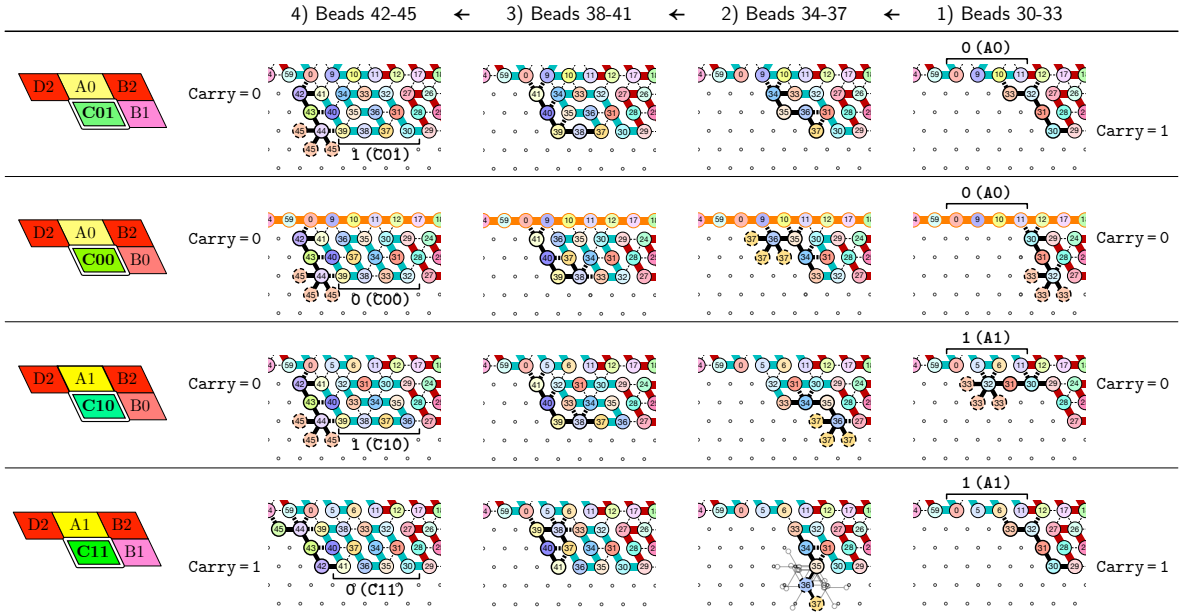


Figure 13. An illustration of how the module C applies a different function which results in different foldings according to the initial state of the molecule (carry = 0 or 1) at the beginning of the folding of the module, and to the environment (the bit 0 or 1 encoded) read above by the function. This figure is meant to be read from right to left (zig pass ←).

Beads 30–33 (rightmost column in Figure 13):

if the carry is 0 *at start*, then bead 30 is able to bind with beads 11 and 12 from the environment and depending on whether the input encodes a bit 0 or 1, bead 32 will be able to bind either to 28 or to 5 and 6 respectively. *Whereas if the carry is* 1, then bead 30 cannot reach the beads 11 and 12. Thus, these are beads 31 and 32 that will bind with beads 10 and 12 from the environment, giving to the molecule a completely different shape.

Beads 34–37 with carry = 0 at start:

as bead 34 is attracted by both beads 30 and 31, the molecule folds upon itself similarly but with a different rotation depending on whether it has read a 0 or a 1 in the environment above: vertically if it has read a 0, horizontally if it has read a 1. Bead 36 attracted by beads 9 and 27 fixes the end of the tip in place leaving bead 37 free to move for now.

Beads 34–37 with carry = 1 at start:

Bead 34 is attracted by beads 9 and 10 encoding a bit 0 above which allows beads 36 and 37 to bind with 31 as well, but prefers to bind with 31 together with 35 otherwise. This induces two different shapes: the beginning of a "wave" pattern ( ) if the bit read above is 0; or the beginning of a "switchback" pattern ( ) if the bit read is 1.

Beads 38–41, without carry propagation (carry = 0, or carry = 1 and bit read = 0):

in these three cases the folding of the beads 38–41 starts from the same position. As the environment is different for each of them, we could design the rule so that this part of the module prefers to adopt the same shape, climing along the part already folded to the top row to start the next module with a carry $= 0$.

Beads 38-41, with carry propagation (carry = 1 and bit read = 1):

because the switchback pattern is upside down in this case, bead 37 stays low and bead 38 can firmly attach to the top with beads 5, 6 and 33, and the tip of the module

folds downwards as 40 and 41 are attracted by 37. This ensures that the folding of the module ends at the bottom row, propagating the carry $= 1$ to the next module.

Note that the bottom rows of the four resulting foldings differ significantly: 39–38–37–30 for C01, 39–38–33–32 for C00, 39–38–37–36 for C10, and 41–36–35–30 for C11. This will allow to distinguish between them in the following zag pass to write the correct bit for the new value of the counter.

4. Proof of Correctness (Theorem 1)

4.1. Overview

As mentioned earlier, each 60-period of the molecule is semantically divided into four modules:

- Module A (beads 0–11): the First Half-Adder
- Module B (beads 12–29): the Left-Turn module
- Module C (beads 30–41): the Second Half-Adder
- Module D (beads 42–59): the Right-Turn module

Depending on the environment in which they fold upon themselves, each of these modules may adopt the following different configurations. We call *bricks* the various configurations each module will adopt in the final folding, as they are the bricks upon which the whole folding is built.

There are six bricks for Modules A and C:

- Axc and Cxc in the zig pass where $x, c \in \{0, 1\}$ are the bit read from the brick above (Ax or Cx) and the (input) carry from the preceding module (Bc or Dc);
- Ay and Cy in the zag pass where y is the bit written: namely $y = x + c \bmod 2$ if the brick above is Axc or Cxc.

There are four bricks for Modules B and D:

- Bc' and Dc' in the zig pass where $c' \in \{0, 1\}$ is the carry output by the preceding brick Cxc or Axc, namely $c' = x \wedge c$;
- and B2 and D2 in the zag pass.

The proof works in three stages: (1) we describe the targeted final folding and show that this target folding implements correctly a binary counter: i.e., that the bricks implement correctly the relationships between the y, x, c and c' described above. Then, we show that the molecule folds indeed as prescribed by the target folding. We proceed in two more stages: (2) we enumerate all the possible surroundings for each module in the target folding; and (3) we show by recurrence that each module folds into the desired brick in each of the possible surroudings. For stage 3, our program produces automatically a human-readable *folding certificate*, which shows the correctness of each step of the folding in each possible environment. It consists in displaying in a compact but readable way, all the possible extensions of the current configuration, and displaying for each of them the number of bonds created (the number in the north-east corner); the maximum bonding configurations are displayed in bold. For readability, we group together extensions when they share a common path up to their last bond (the number of paths grouped is then displayed in the south-east corner for cross-checking). The resulting enumeration is displayed as a tree where only the maximum bond configuration (in bold) gives birth to configurations at the next level. Following the bold path in the tree certificate shows the folding adopted by the molecule. See Figures 14 and 15 for two examples of folding certificates for Brick A01 in the surrounding consisting of the bricks B2, C0, D2 and D1, and for Brick A11 in the surrounding consisting of the bricks B2, C1, D2 and D1. Figure 14 corresponds to the half-adder A reading a 0 from above (Brick C0) and a carry from the right (Brick D1); whereas Figure 15 corresponds to the half-adder A reading a 1 from above (Brick C1) and a carry from the right (Brick D1). Note that

the output configuration is exiting the region from the top for Brick A01 (no carry is propagated) and from the bottom for Brick A11 (a carry is propagated).

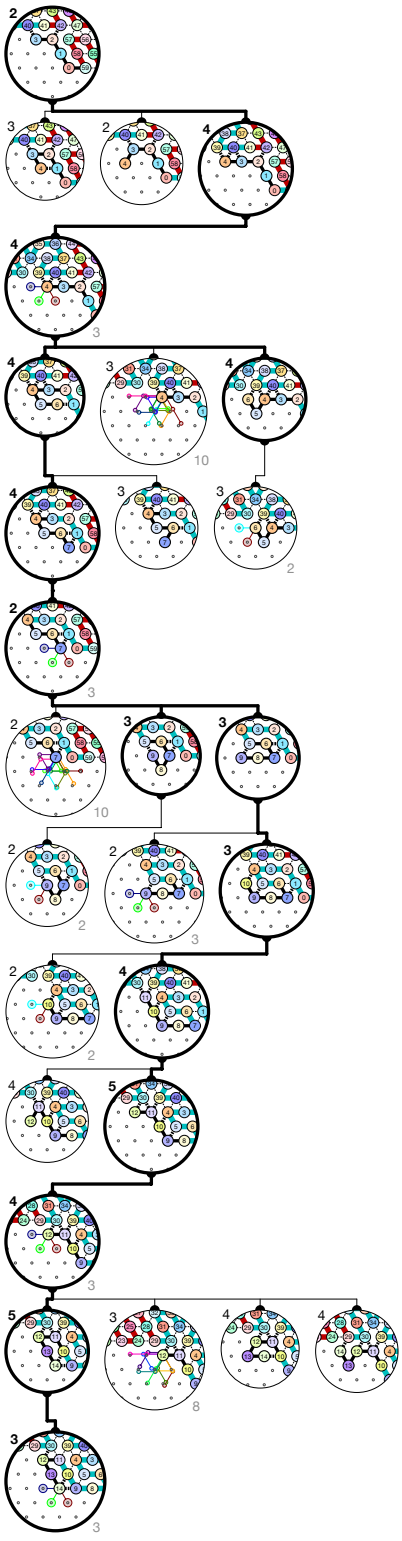
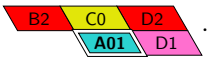


Figure 14. The folding certificate for the brick A01 in the environment: 

.

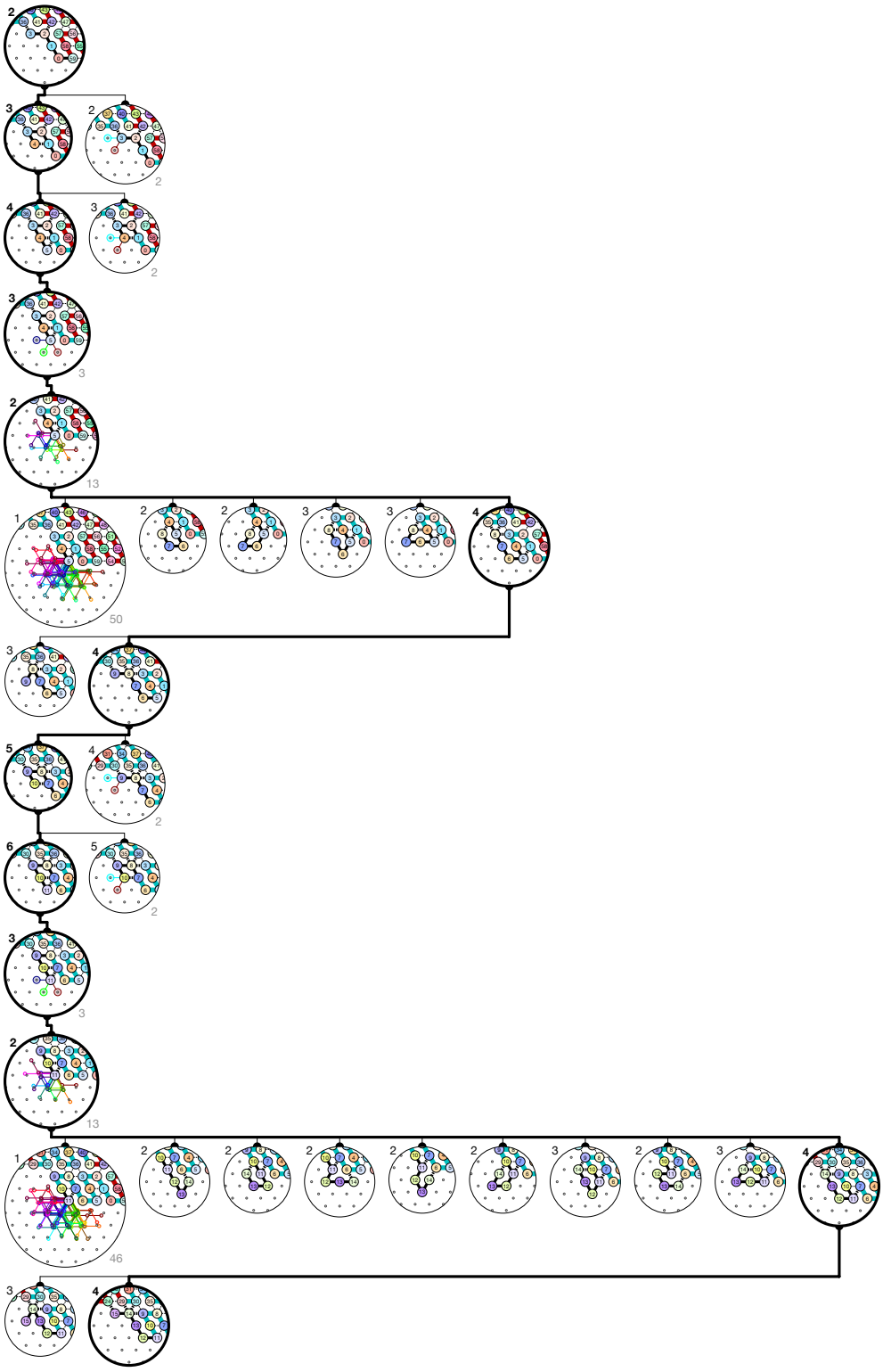
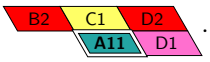


Figure 15. The folding certificate for the brick A11 in the environment: 

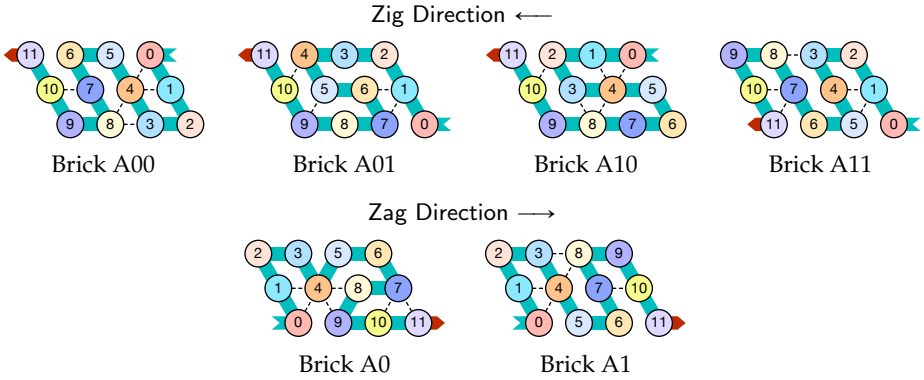
.

In the next subsections, we use our new tools to prove the correctness of the Oritatami System presented in Section 3. Precisely, we show that, starting from a proper seed of length $21 + 20k$, the 60-periodic molecule $(0, \ldots, 59)^\omega$ folds upon itself into $2 \cdot 2^{2k+1} - 1$ rows of height 3 implementing a $(2k + 1)$-bits counter counting from 0 to 2^{2k+1}.

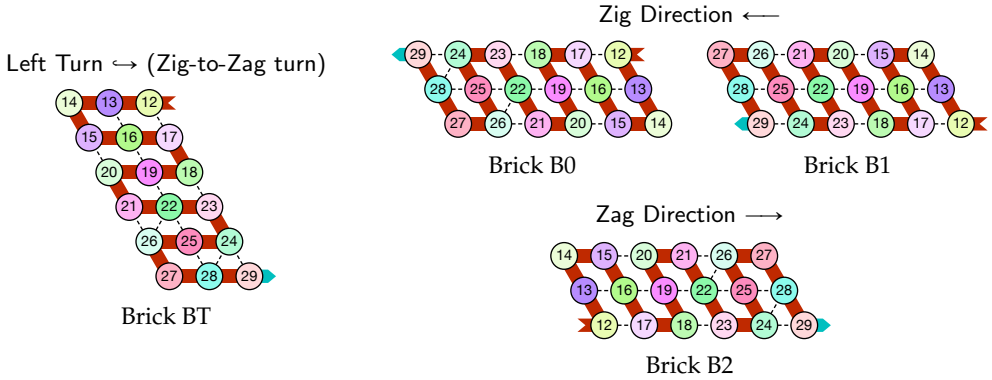
4.2. Description of the Final Configuration (I.e., the Resulting Folding)

Let us first describe each of the possible bricks for each module:

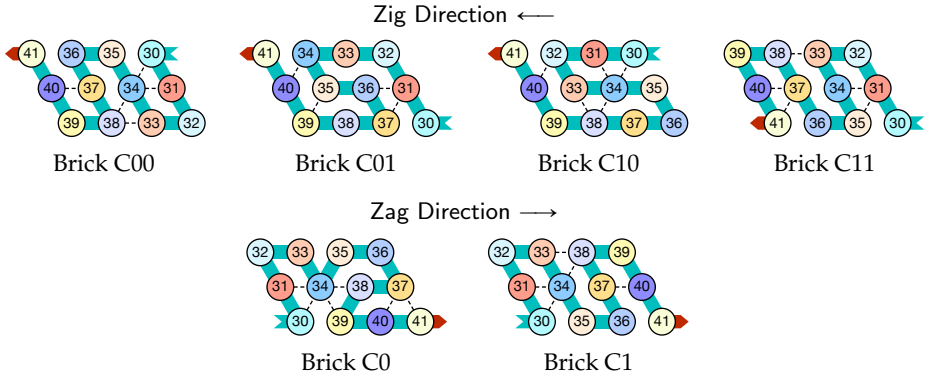
- Module *A*, First Half-Adder (beads 0–11):



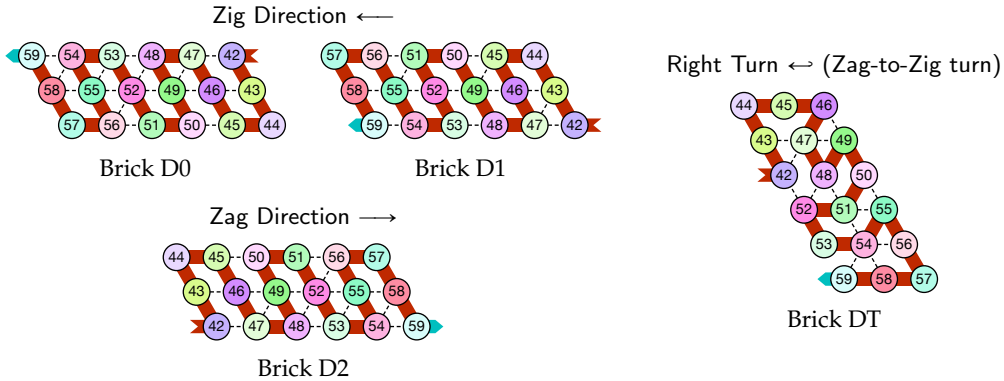
- Module *B*, Left-Turn module (beads 12–29)



- Module *C*, Second Half-Adder (beads 30–41)



- Module *D*, Right-Turn module (beads 42–59)



Note the similarities between the two half-adders A and C in their folding. Using two similar but different modules A and C allow to avoid interference and simplify the design.

The final configuration. In order to program the molecule, we have partitioned the triangular grid into *regions* that will be populated each by a module adopting one of the brick configurations above. The regions and the bricks populating them are displayed in Figure 16. With the exception of the seed region on top, the regions consist in parallelograms of size 4×3, 6×3 or 3×6 organized into $2 \times 2^{2k+1}$ rows of height 3 and $4k+3$ columns of widths 3 or 6. The regions are to be populated as follows to implement a $(2k+1)$-bits counter, as illustrated on Figure 16:

- The rightmost and leftmost columns have width 3:
  - The leftmost column consists in 2^{2k+1} (3×6)-regions all populated with the brick BT (Left Turn).
  - The rightmost column consists in 2^{2k+1} (3×6)-regions all populated with the brick DT (Right Turn).
- The $4k+1$ inner columns consist in 4×3-parallelogram regions if odd and 6×3-parallelogram regions if even. The rows consist of an alternation of *Zig-* and *Zag-rows* to be read *from right to left* and *from left to right*, respectively. The rows $2i+1$ and $2i+2$ take care of reading i in binary from the row $2i$ above, incrementing it in row $2i+1$, and writing $i+1$ in row $2i+2$. In order to describe precisely the folding, let us denote by i_j the jth lowest weight bits of $i \in \mathbb{N}$ when written in binary and by ρ_i the position of the lowest-weight 0-bit of i: $\rho_i = \min\{j : i_j = 0\}$. ρ_i is the position up to which the carry propagates when one increments i: $\rho = (1,2,1,4,1,2,1,4,1,2,1,8,1,2,\ldots)$. Precisely, the region in the pth inner row, $1 \leqslant p \leqslant 2 \cdot 2^{2k+1} - 1$ and the qth inner column, $1 \leqslant q \leqslant 4k+1$ is:
  - if $p = 2i+1$ is odd and $q = 2r$ is even: a 3×6-parallelogram populated with Brick Kc, where:
    - $K =$ B if r is even, and $K =$ D if r is odd;
    - $c = 1$ if $r/2 < \rho_i$ (there still is carry to propagate), and $c = 0$ if $r/2 \geqslant \rho_i$ (there is no more carry to propagate).
  - if $p = 2i$ and $q = 2r$ are even: a 3×6-parallelogram populated with Brick B2 if r is even, or D2 if r is odd.
  - if $p = 2i$ is even and $q = 2r+1$ is odd: a 4×3-parallelogram populated with Brick Ai_r if r is odd and Ci_r if r is even.
  - if $p = 2i+1$ and $q = 2r+1$ are odd: a 4×3-parallelogram populated with Brick Ki_rc where:
    - $K =$ A if r is even, and $K =$ C if r is odd;
    - $c = 1$ if $r/2 \leqslant \rho_i$ (there still is carry to propagate), and $c = 0$ if $r/2 > \rho_i$ (there is no more carry to propagate).
- The seed row on top consists in the bottom row of the brick sequence BT,C0,(D2,A0,B2,C0)^k,DT.
- The leftmost region of the last row is where the folding stops (counter capacity exceeded).

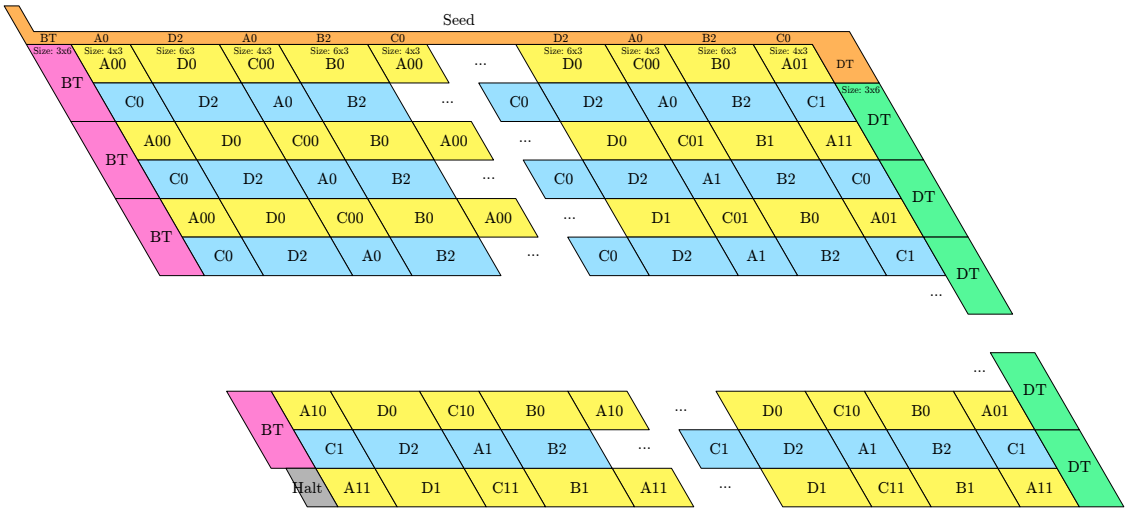


Figure 16. Triangular grid partition into regions populated with the proper bricks. Color coding: Seed-row in orange; Zig-rows (←) in yellow; Zag-rows (→) in blue; Left Turns (↪) in pink; and Right Turns (↩) in green.

4.3. Input and Output Nascent Configurations

Recall that the inertial dynamics extends only the favored nascent configurations from the previous time step (i.e., the one that had the maximum number of bonds). We call this set of configurations, the *output nascent configurations* of the previous time step, or the *input nascent configurations* of the present time step. In this section, we list all the possible input/output nascent configurations according to our design.

The lemmas in Supplementary Materials Section S.1.2 sum up the results and yield by a simple induction that the construction is correct. Let us denote $\alpha, \alpha', \alpha'', \alpha''', \beta, \beta', \gamma, \gamma', \theta, \theta', \lambda_2, \lambda_2', \lambda_3, \lambda_3', \lambda_4, \lambda_4', \lambda_5, \mu, \mu'$ all the possible sets of *output nascent configurations* for all bricks as illustrated in Figure 17. Note that output nascent configurations $\beta, \beta', \gamma, \gamma', \theta, \theta'$ do not represent a single configuration but a set of configurations as the position of the last bead d is not fixed.

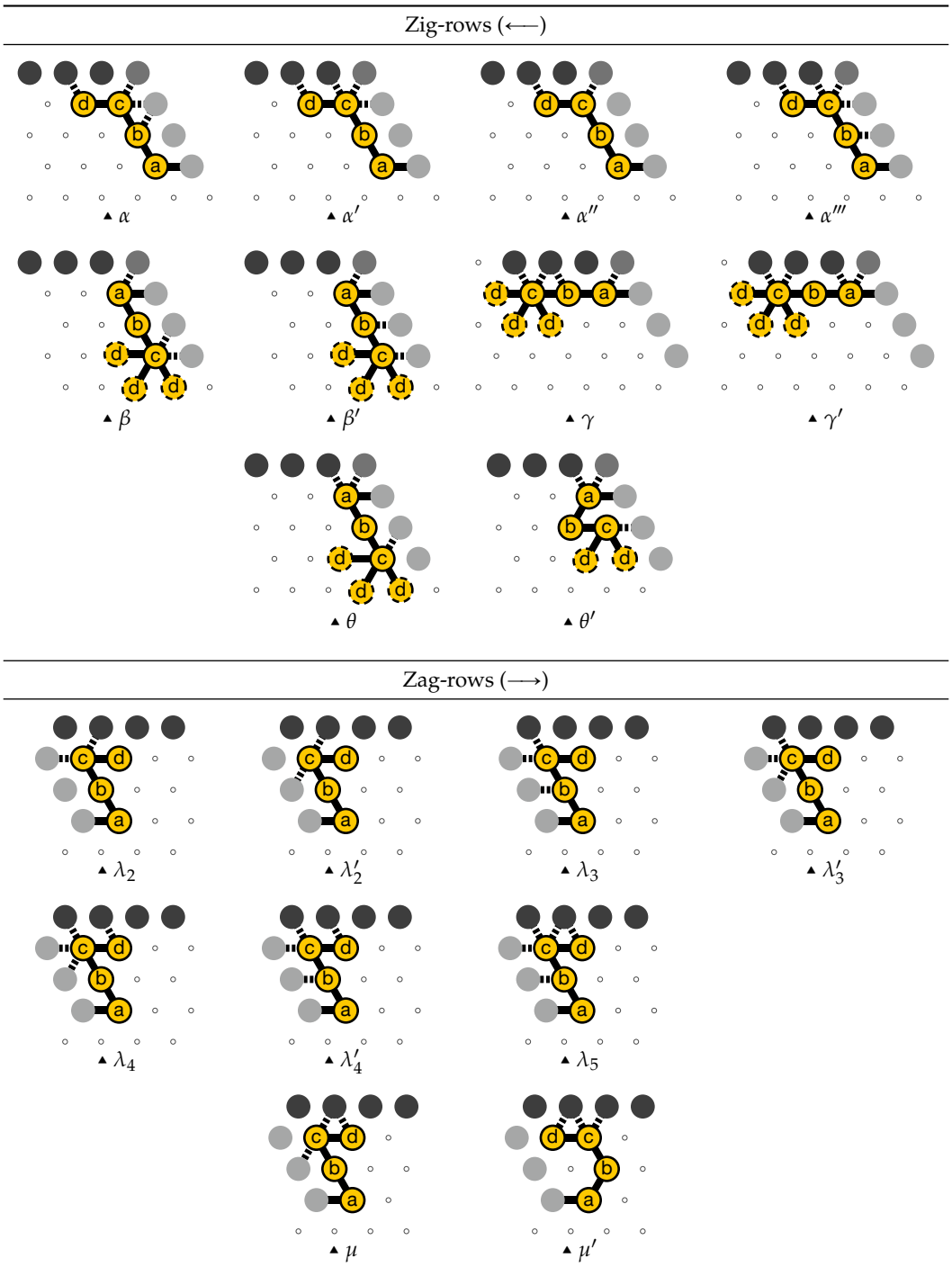


Figure 17. Notation for the various possible sets of output nascent configurations.

Figure 18 illustrates the results proven in the following lemmas and demonstrates that the induction is correct and proves that the bricks fold one after the other so that the molecule folds indeed into the claimed final configuration presented in Figure 16.

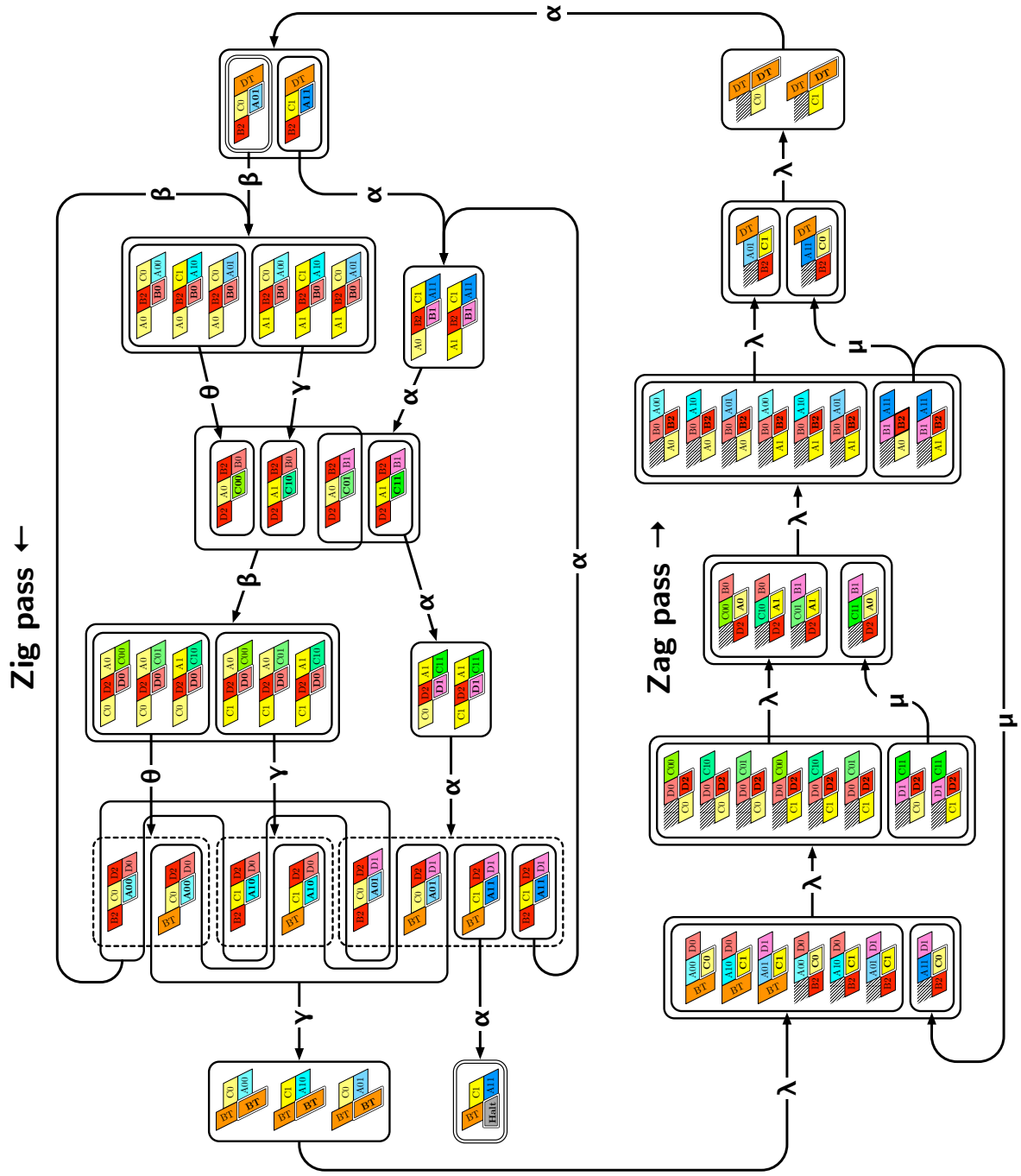


Figure 18. The brick automaton illustrating how Lemmas 1–10 (Supplementary Materials) work together to prove Theorem 1 by induction.

Proof of Correctness. The proof of Theorem 1 proceeds by induction: it is enough to show that each module folds into the expected brick in each region. The folding of each module depends on its environment, i.e., on the bricks already folded nearby and on the current minimum energy configurations output by the previous step. First, Supplementary Materials Section S.1.1 enumerates all the possible environments for each module. Then, Supplementary Materials Section S.1.2 shows that if each module folds as expected, then Theorem 1 is correct. Finally, Supplementary Materials Section S.2 provides all the folding certificates proving that each module folds as expected in every possible environment which concludes the proof of correctness of our 60-beads long periodic oritatami system implementing a binary counter using inertial dynamics.

5. Rule Design Is NP-Hard and FPT

Our second main result concerns the design of a rule for achieving a set of given foldings depending on the environment.

The rule design problem (RDP) consists in the following:

Input: Two *disjoint* sets of bead types B and $\{1,\dots,n\}$, a delay δ, k seed configurations $\sigma^1,\dots,\sigma^k$ of sequences $s^1,\dots,s^k \in B^*$ (with possibly different lengths) and k target configurations $c^1,\dots,c^k$ of sequence $p = \langle 1,\dots,n\rangle$ of length n, and a dynamics $\mathscr{D} \in \{\mathscr{O},\mathscr{I}\}$.

Output: A rule $\heartsuit \subseteq (B \sqcup \{1,\dots,n\})^2$ such that for all $i = 1..k$, the Oritatami system $\mathcal{O} = (p,\heartsuit,\delta)$ folds deterministically into the configuration $\sigma^i c^i$ from the seed configuration σ^i according to dynamics $\mathscr{D}$, i.e., such that: $\mathscr{D}^{n-\delta+1}_{s^i p}\left((\sigma^i)^{\rhd(\delta-1)}\right) = \{\sigma^i c^i\}$ for all $i = 1..k$.

5.1. NP-Completeness

We begin by showing that the rule design problem is NP-complete:

Theorem 4. *For any positive delay and transcript length, the rule design problem is NP-complete.*

Proof. We reduce from 3-SAT with q variables $x_1,\dots,x_q$ and m 3-clauses $C_1,\dots,C_m$ to the rule design problem by designing $3+2q$ bead types $\mathcal{B} = \{1,\dots,n\} \sqcup \{r,z,x_1,\overline{x_1},\dots,x_q,\overline{x_q}\}$, the (fixed length) transcript $p = \langle 1,\dots,n\rangle$, and $q+m$ pairs of seed-target configurations $(\sigma^1,c^1),\dots,(\sigma^q,c^q),(\sigma'^1,c'^1),\dots,(\sigma'^m,c'^m)$, such that $\phi = C_1 \wedge \cdots \wedge C_m$ is satisfiable if and only if there is a rule such that p folds into c^i starting from σ^i, and into c'^i from σ'^i. It will follow that finding a rule is NP-hard (in the number of pairs of seed-target configuration pairs).

Figure 19 shows the seed-target configuration pairs for delay $\delta = 1$. Here, $p = \langle 1\rangle$ of length $n = 1$. This reduction verifies the following property: there is a rule that folds every one of them deterministically iff there is a rule such that 1 is attracted by at least one literal of each clause, and at most one of each variable or its negation. It follows that such a rule exists if and only if ϕ is satisfiable (by setting to true every literal to which 1 is attracted).

Figure 19. The polynomial-time reduction for 3-SAT to the rule design problem for delay $\delta = 1$. The seed configurations are linked in orange, and the target configurations are linked in turquoise. (**a**) The seed-target configuration pair for clause $\ell_i \vee \ell_j \vee \ell_k$: it is deterministically foldable iff 1 is attracted by at least one of ℓ_i, ℓ_j, ℓ_k; (**b**) The seed-target configuration pair for variable x_i: it is deterministically foldable iff 1 is attracted by r and 1 is attracted by at most one of z, x_i and $\overline{x_i}$.

Finally, extending this proof to a larger delay time $\delta \geqslant 2$ can be done as shown in Figure 20 by setting $p = \langle 1,\dots,\delta\rangle$ (of length $n = \delta$) and augmenting in the seed configurations with a tunnel of length $\delta - 1$ that funnels the $\delta - 1$ first beads of the transcript p. This reduces to the delay 1 case. □

(**a**) (**b**)

Figure 20. The reduction for 3-SAT to the rule design problem for delay $\delta \geqslant 2$. (**a**) The seed-target configuration pair for clause $\ell_i \vee \ell_j \vee \ell_k$: it is deterministically foldable iff δ is attracted by at least one of ℓ_i, ℓ_j, ℓ_k. (**b**) The seed-target configuration pair for variable x_i: it is deterministically foldable iff δ is attracted by r and δ is attracted by at most one of x_i and $\overline{x_i}$.

5.2. An Efficient Algorithm in Practice

As can be observed in the proof of Theorem 4, the NP-hardness of the Rule Design Problem depends on the number of desired configurations (k) and *not on the length of the transcript (n)*. As Theorem 3 will show next, the problem is indeed linear-time in the length n of the transcript when the number of desired configuration is a constant, and exponential-time only in k and δ. Such a problem is said *fixed parameter tractable (FPT)* as it can be solved in polynomial time when some parameters (here k and δ) are small constants. As one usually only desires a small number of different configurations for every given part of the transcript, the algorithm described below is very efficient in practice and allowed us to solve various key parts of our design in spite of its NP-hardness in general!

Theorem 5. *thm:fpt The rule design problem (RDP) with k seed-target configurations, each of length n, and delay time δ is fixed-parameter tractable, as it can be solved in time and space complexity $O_{k,\delta}(n)$. More precisely . Algorithm 1 solves RDP for the oblivious dynamics $\mathscr{O}$ in time $O(n \cdot 5^{\delta} 2^{3k(\delta^3+\delta^2+4\delta+1)})$ time and uses $O(n \cdot \delta^2 2^{3k(\delta^3+\delta^2+4\delta+1)})$ space; and Algorithm 2 solves RDP for the inertial dynamics $\mathscr{I}$ in $O(n \cdot 5^{\delta} 2^{k5^{\delta-1}+3k(\delta^3+\delta^2+4\delta+1)})$ time and uses $O(n \cdot \delta^2 5^{\delta} 2^{k5^{\delta-1}+3k(\delta^3+\delta^2+4\delta+1)})$ space.*

The bounds given on the memory use seem to indicate that the algorithm might be impractical even if its time complexity is linear in the length of the transcript. However, we implemented lazily (i.e., allocating memory only when needed) and then, its memory usage remains modest because the number of output nascent configurations remains small. We have used this procedure successfully to solve key parts of our design. Indeed, note that in our design, there are only very few input configurations (most of the time just one) which makes the potentially heavy extension step very fast in practice, at least for this design.

Algorithm 1 Rule Design Problem FPT Algorithm—Oblivious dynamics

function EXTENDOBLIVIOUS(i, a partial rule R)
  ▷ *This procedure computes all the possible extensions of rule R s.t. bead p_i is placed at its desired location in all k target configurations*
  Let $\Sigma = \varnothing$
  Let $\mathcal{N}$ be the bead types reachable by the $(i+\delta-1)$-th beads of the transcript p in any of the δ-elongations of the $(i-1)$-prefixes of the target configurations $\sigma^1 c^1_{1..i-1}, \ldots, \sigma^k c^k_{1..i-1}$
  for all partial rule $\rho : \{p_{i+\delta-1}\} \times \mathcal{N} \to \{\texttt{true}, \texttt{false}\}$ **do**
    Let $R' = R \cup \rho$
    try
      for $j = 1..k$ **do**
        let $S_j = \left(\sigma^j c^j_{1..i-1}\right)^{\rhd(\delta-1)}$
        let $S'_j = \mathscr{O}_{R'}(S_j) = \bigcup_{\gamma \in \sigma_j c^j_{1..i-1}} \underset{c \in \gamma^{\rhd\delta}}{\arg\min} \mathrm{E}_{R'}(c)$
        if $S'_j = \varnothing$ or $\exists c' \in S'_j$ s.t. $c'_i \neq c^j_i$ **then**
          throw *rejectRule* ▷ The rule R' fails to place the bead p_i at its desired position
    $\Sigma := \Sigma \cup \{R'\}$ ▷ Add R' to Σ
    catch *rejectRule*: **continue** ▷ Do not add R' to Σ
  return Σ

function PROJECTOBLIVIOUS(i, Set $\mathcal{R}$ of partial rules R))
  ▷ *This procedure retains only one representative partial rule for every possible subrule of the last $\delta-1$ nascent beads*
  Let $\Pi := \varnothing$
  Let $\mathcal{N}$ be the bead types reachable by the $\delta-1$ nascent beads $p_{i+1}, \ldots, p_{i+\delta-1}$ in any of the $(\delta-1)$-elongations of the i-prefixes of the target configurations $\sigma^1 c^1_{1..i}, \ldots, \sigma^k c^k_{1..i}$
  for all partial rule $\rho : \{p_{i+1}, \ldots, p_{i+\delta-1}\} \times \mathcal{N} \to \{\texttt{true}, \texttt{false}\}$ **do**
    Let R_ρ be the subset of all partial rules R in $\mathcal{R}$ matching with ρ.
    if $R_\rho \neq \varnothing$ **then**
      Pick one rule $R \in R_\rho$ to be the representative of set R_ρ and add it to Π: $\Pi := \Pi \cup \{R\}$
  return Π

function FINDRULEOBLIVIOUS( )
  Let $\mathcal{N}$ be the bead types reachable by the $\delta-1$ nascent beads $p_1, \ldots, p_{\delta-1}$ in any of the $(\delta-1)$-elongations of the k seed configurations $\sigma^1, \ldots, \sigma^k$
  Initialize $\mathcal{R}$ as the set of all partial rules $R : \{p_1, \ldots, p_{\delta-1}\} \times \mathcal{N} \to \{\texttt{true}, \texttt{false}\}$
  for $i = 1..n-\delta+1$ **do** ▷ Extend the rule to place correctly the bead p_i
    Let $\Sigma = \varnothing$
    for all partial rule $R \in \mathcal{R}$ **do**
      $\Sigma := \Sigma \cup$ EXTENDOBLIVIOUS(i, R)
    Update $\mathcal{R} :=$ PROJECTOBLIVIOUS(i, Σ)
  for all $R \in \mathcal{R}$ and $j = 1..k$ **do** ▷ Check the positions of the last $\delta-1$ nascent beads
    if $\mathscr{O}_R(\sigma^j (c^j)^{\lhd 1}) \neq \{\sigma^j c^j\}$ **then**
      Remove R from $\mathcal{R}$: $\mathcal{R} := \mathcal{R} \backslash \{R\}$
  if $\mathcal{R} \neq \varnothing$ **then**
    Pick a rule $R \in \mathcal{R}$ and **return** R
  else
    return "There is no rule building the k target configurations together obliviously"

Algorithm 2 Rule Design Problem FPT Algorithm—Inertial dynamics

function EXTENDINERTIAL(i, a partial rule R, the k sets of nascent configurations $S^1, \ldots, S^k$ up to bead $p_{i+\delta-2}$ favored by R in each of the k target environments)
  ▷ *This procedure computes all the possible extensions of rule R s.t. bead p_i is placed at its desired location in all k target configurations*
  Let $\Sigma = \varnothing$
  Let $\mathcal{N}$ be the bead types reachable by the $(i+\delta-1)$-th bead of the transcript p in any of the 1-elongations of the favored nascent configurations in $S^1, \ldots, S^k$
  for all partial rule $\rho : \{p_{i+\delta-1}\} \times \mathcal{N} \to \{\texttt{true}, \texttt{false}\}$ **do**
    Let $R' = R \cup \rho$
    try
      for $j = 1..k$ **do**
        let $S'^j = \mathscr{I}_{R'}(S^j) = \bigcup_{\gamma \in S^{j \lhd (\delta-1)}} \underset{c \in \gamma^{\rhd\delta} \cap S^{j \rhd 1}}{\arg\min} \mathrm{E}_{R'}(c)$
        if $S'^j = \varnothing$ or $\exists c' \in S'^j$ s.t. $c'_i \neq c^j_i$ **then**
          throw *rejectRule* ▷ The rule R' fails to place the bead p_i at its desired position
      $\Sigma := \Sigma \cup \{\langle R', S'^1, \ldots, S'^k \rangle\}$ ▷ Add R' and its favored nascent configurations to Σ
    catch *rejectRule*: **continue** ▷ Do not add R' to Σ
  return Σ

function PROJECTINERTIAL(i, Set $\mathcal{S}$ of tuples $\langle$partial rule R, the k sets of nascent configurations $S^1, \ldots, S^k$ up to bead $p_{i+\delta-1}$ favored by R in each of the k target environments$\rangle$)
  ▷ *This procedure retains only one representative partial rule for every possible subrule and resulting favored nascent configurations of the last $\delta-1$ nascent beads in every environments*
  Let $\Pi := \varnothing$
  Let $\mathcal{N}$ be the bead types reachable by the $\delta-1$ nascent beads $p_{i+1}, \ldots, p_{i+\delta-1}$ in any of the $\delta-1$ elongations of the partial target configurations $\sigma^1 c^1_{1..i}, \ldots, \sigma^k c^k_{1..i}$
  for all partial rule $\rho : \{p_{i+1}, \ldots, p_{i+\delta-1}\} \times \mathcal{N} \to \{\texttt{true}, \texttt{false}\}$ **and all** subsets $\Gamma^1, \ldots, \Gamma^k$ of $(\delta-1)$-elongations of the partial target configurations $\sigma^1 c^1_{1..i}, \ldots, \sigma^k c^k_{1..i}$ **do**
    Let $R_{\rho,\Gamma}$ be the subset of all partial rules R in $\mathcal{S}$ which match with ρ and whose favored nascent configurations are the sets $\Gamma^1, \ldots, \Gamma^k$, i.e.:
$$\text{let } R_{\rho,\Gamma} = \{R : R \text{ matches with } \rho \text{ and } \langle R, \Gamma^1, \ldots, \Gamma^k \rangle \in \mathcal{S}\}$$
    if $R_{\rho,\Gamma} \neq \varnothing$ **then**
      Pick one rule $R \in R_{\rho,\Gamma}$ to be the representative of set $R_{\rho,\Gamma}$ and add it to Π:
$$\Pi := \Pi \cup \{\langle R, \Gamma^1, \ldots, \Gamma^k \rangle\}$$
  return Π

function FINDRULEINERTIAL( )
  Let $\mathcal{N}$ be the bead types reachable by the $\delta-1$ nascent beads $p_1, \ldots, p_{\delta-1}$ in any of the $(\delta-1)$-elongations of the k seed configurations $\sigma^1, \ldots, \sigma^k$
  Initialize $\mathcal{S}$ as the set of all tuples $\langle R, \sigma^{1 \rhd \delta-1}, \ldots, \sigma^{k \rhd \delta-1} \rangle$ for all partial rules $R : \{p_1, \ldots, p_{\delta-1}\} \times \mathcal{N} \to \{\texttt{true}, \texttt{false}\}$
  for $i = 1..n-\delta+1$ **do** ▷ Extend the rule to place correctly the bead p_i
    Let $\Sigma = \varnothing$
    for all tuple $\langle R, S^1, \ldots, S^k \rangle \in \mathcal{S}$ **do**
      $\Sigma := \Sigma \cup$ EXTENDINERTIAL(i, R, $S^1, \ldots, S^k$)
    Update $\mathcal{S}$:= PROJECTINERTIAL(i, Σ)
  if $\exists \langle R, \{\sigma^1 c^1\}, \ldots, \{\sigma^k c^k\} \rangle \in \mathcal{S}$ **then** ▷ Check the positions of the last $\delta-1$ nascent beads
    return R
  else
    return "There is no rule building the k target configurations together inertially"

Proof sketch (complete proof in Supplementary Materials Section S.3). The key is that every step of the folding is computed locally in a fixed and known environment: at each step i, the δ beads to be folded look for their best positions by interacting with beads with fixed and known positions within a radius $\delta + 1$. It follows that one can compute the set of all suitable subrules, considering these $O(\delta^2)$ bead types only (i.e., that place the $i - \delta + 1$-th bead of the molecule at the correct position). Oblivious $\mathscr{O}$ and inertial $\mathscr{I}$ dynamics differ as $\mathscr{I}$ only consider input nascent configurations which are output by the previous step, whereas $\mathscr{O}$ does not use any information from the previous step. This implies that one need to remember some information to connect one step to the next in $\mathscr{I}$, whereas $\mathscr{O}$ needs no memory at all.

Formally, a *subrule* $R : B^2 \to \{\texttt{true}, \texttt{false}, \bot\}$ is a symmetric function that states for each pair of beads if they attract each other (`true`) or not (`false`), or if this is undefined ($\bot$). We denote by $\operatorname{dom} R = \{(a, b) \in B^2 : R(a, b) \neq \bot\}$ the *domain* of R. Two subrules R_1 and R_2 are *compatible*, denoted by $R_1 \sim R_2$ if they agree for every pair where they are both defined, i.e., if for all $a, b \in \operatorname{dom} R_1 \cap \operatorname{dom} R_2$, $R_1(a, b) = R_2(a, b)$. We say that R_1 *matches with* R_2 if for all $(a, b) \in \operatorname{dom} R_2$, $R_1(a, b) = R_2(a, b)$. If $R_1 \sim R_2$, we denote by $R_1 \cup R_2$ the subrule obtained by *merging* R_1 and R_2, i.e. defined by $R_1 \cup R_2\,(a, b) = R_1(a, b)$ if $(a, b) \in \operatorname{dom} R_1$, and $= R_2(a, b)$ otherwise.

For all $0 \leqslant 1 \leqslant |c|$, we denote by $c_{1..i}$ the prefix of length i of a configuration c. Algorithm 1 solves RDP for the oblivious dynamics in time linear in the length of the sequence as follows. It incrementally constructs a set of rules that place each bead at its desired position in each of the k environments. It proceeds by maintaining a set $\mathcal{R}$ of candidate subrules that place correctly the first i beads in every of the k environments. At the i-th step, the main procedure FindRuleOblivious() first extends each candidate subrule R in $\mathcal{R}$ by calling procedure ExtendOblivious() which scans all the possible attraction rule extensions of R for the new nascent bead (the $(i + \delta - 1)$-th) with the bead types it can reach in each of the k configurations, and retains only the ones that place the i-th bead at its correct position for all k configurations. Note that this extension of the rule does not change the positioning of the $(i - 1)$th first beads since the $(i + \delta - 1)$-th bead is not yet produced when they are placed. Now, in order to keep the processing time constant for each bead, main procedure FindRuleOblivious() calls procedure ProjectOblivious() which retains only one representative of each subset of rules that define the same attractions for the $\delta - 1$ nascent beads (indexed from $i + 1$ to $i + \delta - 1$), i.e., for the only bead types for which the rule matters in order to determine the positions of the upcoming beads. Once the $n - \delta$-th bead is placed, the main procedure concludes by checking that the surviving rules place the last $\delta - 1$ beads at their desired positions.

Algorithm 2 works similarly for the inertial dynamics $\mathscr{I}$. In fact, we just extend the subrule technics by testing subrules for each possible input and corresponding output nascent configurations set, for each seed-target configurations pair and each time step. □

5.3. Comparison with Related Works

A variant of the rule design problem was studied by Ota and Seki [43] with one single seed/target configurations pair and where the transcript can be arbitrary (any bead type can be used, including repetition and one from the seed). In the oblivious dynamics, they fully characterize for all possible combinations of delay δ and arity α for which this variant either can be solved in a polynomial time or is NP-hard. This problem remains as hard even if the transcript is required to be periodic.

The uniqueness of the bead types used in the transcript is thus crucial for the FPT algorithm to exist. Thus, it is not clear whether their variant admits an FPT algorithm or not. Note that other algorithms were proposed, such as the heuristic rule set optimization algorithm by Han and Kim [45].

6. Perspectives

The purpose of our new model is not to be entirely accurate with respect to phenomena observed in nature, but instead to start developing an intuition about the *kind of problem* that need to be solved in order to engineer RNA shapes, and later, even proteins.

In the future, a number of extensions of this model seem natural. In particular, extending it with a more realistic notion of *thermodynamics and molecular agitation*. Using existing works in molecular dynamics [47], would allow to explore stochastic optimization processes. It would be of highest interest if one could come up with a stochastic extension of this model that is still able of Turing-complete computation as was shown in [42] for the present model.

Supplementary Materials: The following are available online at http://www.mdpi.com/1422-0067/20/9/2259/s1.

Author Contributions: All authors contributed to: conceptualization, methodology, validation, formal analysis, investigation, resources, writing–original draft preparation and review and editing, supervision, project administration. Software and visualization, P.-É.M. and N.S.

Funding: C. Geary is a Carlsberg Postdoctoral Fellow supported by the Carlsberg Foundation. N. Schabanel is supported by Grants ANR-12-BS02-005 RDAM, IXXI MOLECAL, IXXI CALCASMOL, CNRS MoPrExProgMol and CNRS AMARP grants. S. Seki is in part supported by the Academy of Finland, Postdoctoral Researcher Grant 13266670/T30606, JST Program to Disseminate Tenure Tracking System, MEXT, Japan, No. 6F36 and JSPS KAKENHI Grant-in-Aid for Young Scientists (A) No. 16H05854 and for Challenging Research (Exploratory) No. 18K19779.

Acknowledgments: The authors thank Abdulmelik Mohammed, Andrew Winslow, Damien Woods for discussions and encouragements.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rothemund, P.W.K. Folding DNA to create nanoscale shapes and patterns. *Nature* **2006**, *440*, 297–302. [CrossRef] [PubMed]
2. Yurke, B.; Turberfield, A.J.; Mills, A.P.; Simmel, F.C.; Neumann, J.L. A DNA-fuelled molecular machine made of DNA. *Nature* **2000**, *406*, 605–608. [CrossRef]
3. Evans, C.G. Crystals that Count! Physical Principles and Experimental Investigations of DNA Tile Self-Assembly. Ph.D. Thesis, California Institute of Technology, Pasadena, CA, USA, 2014.
4. Seeman, N.C. Nucleic-acid junctions and lattices. *J. Theor. Biol.* **1982**, *99*, 237–247. [CrossRef]
5. Winfree, E. Algorithmic Self-Assembly of DNA. Ph.D. Thesis, Caltech, Pasadena, CA, USA, 1998.
6. Cannon, S.; Demaine, E.D.; Demaine, M.L.; Eisenstat, S.; Patitz, M.J.; Schweller, R.; Summers, S.M.; Winslow, A. Two Hands Are Better Than One (up to constant factors). In Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science, Kiel, Germany, 27 February–2 March 2013; pp. 172–184.
7. Chen, H.L.; Doty, D. Parallelism and Time in Hierarchical Self-Assembly. In Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, Kyoto, Japan, 17–19 January 2012; pp. 1163–1182.
8. Fujibayashi, K.; Zhang, D.Y.; Winfree, E.; Murata, S. Error suppression mechanisms for DNA tile self-assembly and their simulation. *Nat. Comput.* **2009**, *8*, 589–612. [CrossRef]
9. Cook, M.; Fu, Y.; Schweller, R.T. Temperature 1 Self-Assembly: Deterministic Assembly in 3D and Probabilistic Assembly in 2D. In Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, USA, 23–25 January 2011.
10. Woods, D.; Doty, D.; Myhrvold, C.; Hui, J.; Zhou, F.; Yin, P.; Winfree, E. Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. *Nature* **2019**, *567*, 366–372. [CrossRef] [PubMed]
11. Afonin, K.A.; Bindewald, E.; Yaghoubian, A.J.; Voss, N.; Jacovetty, E.; Shapiro, B.A.; Jaeger, L. In vitro Assembly of Cubic RNA-Based Scaffolds Designed in silico. *Nat. Nanotechnol.* **2010**, *5*, 676–682. [CrossRef]
12. Afonin, K.A.; Kireeva, M.; Grabow, W.W.; Kashiev, M.; Jaeger, L.; Shapiro, B.A. Co-transcriptional Assembly of Chemically Modified RNA Nanoparticles Functionalized with siRNAs. *Nano Lett.* **2012**, *12*, 5192–5195. [CrossRef]
13. Li, M.; Zheng, M.; Wu, S.; Tian, C.; Weizmann, Y.; Jiang, W.; Wang, G.; Mao, C. In vivo production of RNA nanostructures via programmed folding of single-stranded RNAs. *Nat. Commun.* **2018**, *9*, 2196. [CrossRef]
14. Schwarz-Schilling, M.; Dupin, A.; Chizzolini, F.; Krishnan, S.; Mansy, S.S.; Simmel, F.C. Optimized Assembly of a Multifunctional RNA-Protein Nanostructure in a Cell-Free Gene Expression System. *Nano Lett.* **2018**, *18*, 2650–2657. [CrossRef] [PubMed]

15. Zuker, M.; Sankoff, D. RNA secondary structures and their prediction. *Bull. Math. Biol.* **1984**, *46*, 591–621. [CrossRef]
16. Mathews, D.H.; Disney, M.D.; Childs, J.L.; Schroeder, S.J.; Zuker, M.; Turner, D.H. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proc. Natl. Acad. Sci. USA* **2004**, *101*, 7287–7292. [CrossRef]
17. Mathews, D. Revolutions in RNA secondary structure prediction. *J. Mol. Biol.* **2006**, *359*, 526–532. [CrossRef]
18. Rivas, E. The four ingredients of single-sequence RNA secondary structure prediction. A unifying perspective. *RNA Biol.* **2013**, *10*, 1185–1196. [CrossRef]
19. Jabbari, H.; Condon, A. A fast and robust iterative algorithm for prediction of RNA pseudoknotted secondary structures. *BMC Bioinform.* **2014**, *15*, 147. [CrossRef]
20. Dill, K. Theory for the folding and stability of globular proteins. *Biochemistry* **1985**, *24*, 1501–1509. [CrossRef] [PubMed]
21. Unger, R.; Moult, J. Finding the lowest free energy conformation of a protein is an NP-hard problem: Proof and implications. *Bull. Math. Biol.* **1993**, *55*, 1183–1198. [CrossRef]
22. Paterson, M.; Przytycka, T. On the complexity of string folding. In Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming, Paderborn, Germany, 8–12 July 1996; Meyer, F., Monien, B., Eds.; Springer: Berlin/Heidelberg, Germany, 1996; Volume 1099, pp. 658–669.
23. Atkins, J.; Hart, W.E. On the Intractability of Protein Folding with a Finite Alphabet of Amino Acids. *Algorithmica* **1999**, *25*, 279–294. [CrossRef]
24. Berger, B.; Leighton, T. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *J. Comput. Biol.* **1998**, *5*, 27–40. [CrossRef]
25. Crescenzi, P.; Goldman, D.; Papadimitriou, C.; Piccolboni, A.; Yannakakis, M. On the complexity of protein folding. *J. Comput. Biol.* **1998**, *5*, 423–465. [CrossRef]
26. Aichholzer, O.; Bremner, D.; Demaine, E.D.; Meijer, H.; Sacristán, V.; Soss, M. Long proteins with unique optimal foldings in the H-P model. *Comput. Geom.* **2003**, *25*, 139–159. [CrossRef]
27. Boyle, J.; Robillardl, G.; Kim, S. Sequential folding of transfer RNA. A nuclear magnetic resonance study of successively longer tRNA fragments with a common 5′ end. *J. Mol. Biol.* **1980**, *139*, 601–625. [CrossRef]
28. Frieda, K.L.; Block, S.M. Direct observation of cotranscriptional folding in an adenine riboswitch. *Science* **2012**, *338*, 397–400. [CrossRef]
29. Geary, C.; Rothemund, P.W.K.; Andersen, E.S. A Single-Stranded Architecture for Cotranscriptional Folding of RNA Nanostructures. *Science* **2014**, *345*, 799–804. [CrossRef]
30. Geary, C.; Chworos, A.; Verzemnieks, E.; Voss, N.R.; Jaeger, L. Composing RNA Nanostructures from a Syntax of RNA Structural Modules. *Nano Lett.* **2017**, *17*, 7095–7101. [CrossRef] [PubMed]
31. Jepsen, M.D.E.; Sparvath, S.M.; Nielsen, T.B.; Langvad, A.H.; Grossi, G.; Gothelf, K.V.; Andersen, E.S. Development of a Genetically Encodable FRET System using Fluorescent RNA Aptamers. *Nat. Commun.* **2018**, *9*, 18. [CrossRef]
32. Doty, D.; Lutz, J.H.; Patitz, M.J.; Schweller, R.T.; Summers, S.M.; Woods, D. The tile assembly model is intrinsically universal. In Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science, New Brunswick, NJ, USA, 20–23 October 2012 ; pp. 439–446.
33. Meunier, P.E.; Patitz, M.J.; Summers, S.M.; Theyssier, G.; Winslow, A.; Woods, D. Intrinsic universality in tile self-assembly requires cooperation. In Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, Portland, OR, USA, 5–7 January 2014.
34. Rothemund, P.W.K.; Papadakis, N.; Winfree, E. Algorithmic Self-Assembly of DNA Sierpinski Triangles. *PLoS Biol.* **2004**, *2*, e424. [CrossRef] [PubMed]
35. Fujibayashi, K.; Hariadi, R.; Park, S.H.; Winfree, E.; Murata, S. Toward Reliable Algorithmic Self-Assembly of DNA Tiles: A Fixed-Width Cellular Automaton Pattern. *Nano Lett.* **2008**, *8*, 1791–1797. [CrossRef]
36. Wei, B.; Dai, M.; Yin, P. Complex shapes self-assembled from single-stranded DNA tiles. *Nature* **2012**, *485*, 623–626. [CrossRef]
37. Geary, C.; Meunier, P.E.; Schabanel, N.; Seki, S. Programming Biomolecules That Fold Greedily During Transcription. In Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science, Krakow, Poland, 22–26 August 2016; Volume 58, p. 43.

38. Masuda, Y.; Seki, S.; Ubukata, Y. Towards the Algorithmic Molecular Self-assembly of Fractals by Cotranscriptional Folding. In Proceedings of the 23rd International Conference on Implementation and Application of Automata, Charlottetown, PE, Canada, 30 July–2 August 2018; Volume 10977, pp. 261–273.
39. Demaine, E.; Hendricks, J.; Olsen, M.; Patitz, M.J.; Rogers, T.A.; Nicolas, S.; Seki, S.; Thomas, H. Know When to Fold 'Em: Self-assembly of Shapes by Folding in Oritatami. In Proceedings of the 24th International Conference on DNA Computing and Molecular Programming, Jinan, China, 8–12 October 2018; Volume 11145, pp. 19–36.
40. Han, Y.S.; Kim, H. Construction of Geometric Structure by Oritatami System. In Proceedings of the International Conference on DNA Computing and Molecular Programming, Jinan, China, 8–12 October 2018; Volume 11145, pp. 173–188.
41. Han, Y.S.; Kim, H.; Ota, M.; Seki, S. Nondeterministic seedless oritatami systems and hardness of testing their equivalence. *Nat. Comput.* **2018**, *17*, 67–79. [CrossRef]
42. Geary, C.; Meunier, P.E.; Schabanel, N.; Seki, S. Proving the Turing Universality of Oritatami Co-Transcriptional Folding. In Proceedings of the 29th International Symposium on Algorithms and Computation, Taiwan, China, 16–19 December 2018; Volume 123, p. 23.
43. Ota, M.; Seki, S. Rule Set Design Problems for Oritatami Systems. *Theor. Comput. Sci.* **2017**, *671*, 26–35. [CrossRef]
44. Han, Y.S.; Kim, H.; Seki, S. Transcript Design Problem of Oritatami Systems. In Proceedings of the International Conference on DNA Computing and Molecular Programming, Jinan, China, 8–12 October 2018; Volume 11145, pp. 139–154.
45. Han, Y.S.; Kim, H. Ruleset Optimization on Isomorphic Oritatami Systems. In Proceedings of the 23rd International Conference on DNA Computing and Molecular Programming, Austin, TX, USA, 24–28 September 2017; Volume 10467, pp. 33–45.
46. Rogers, T.A.; Seki, S. Oritatami System: A Survey and the Impossibility of Simple Simulation at Small Delays. *Fund. Inform.* **2017**, *154*, 359–372. [CrossRef]
47. Woods, D.; Chen, H.L.; Goodfriend, S.; Dabby, N.; Winfree, E.; Yin, P. Active Self-Assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time. In Proceedings of the ITCS 2013: Innovations in Theoretical Computer Science, Berkeley, CA, USA, 10–12 January 2013; pp. 353–354.

International Journal of
Molecular Sciences

MDPI

Article

Oritatami: A Computational Model for Molecular Co-Transcriptional Folding

Cody Geary [1], Pierre-Étienne Meunier [2], Nicolas Schabanel [3,*] and Shinnosuke Seki [4]

[1] Computer Science Computation and Neural Systems Bioengineering Caltech, MS 136-93, Moore Building, Pasadena, CA 91125, USA; codyge@gmail.com
[2] Computer Science Dept, Hamilton Institute, Maynooth University, Co. Kildare, Ireland; pierre-etienne.meunier@mu.ie
[3] CNRS, École normale supérieure de Lyon (LIP), CEDEX 07, 69364 Lyon, France
[4] Computer and Network Engineering Dept, University of Electro-Communications, 1-5-1, Chofugaoka, Chofu, Tokyo 1828585, Japan; s.seki@uec.ac.jp
* Correspondence: nicolas.schabanel@ens-lyon.fr

Received: 15 April 2019; Accepted: 30 April 2019; Published: 7 May 2019

Supplementary Materials:

S.1. Correctness of the Folding

This section contains the missing part of the proof of correctness of the counter Oritatami system in Section 4 proving our main Theorem 1.

S.1.1. Enumerating All the Possible Environments

The proof of Theorem 1 proceeds by induction: it is enough to show that each module folds into the expected brick in each region. The folding of each module depends on its environment, i.e., on the bricks already folded nearby and on the current minimum energy configurations output by the previous step. Recall that the Zig-rows (the one that begin and end with an A-brick before the turns) fold from right to left and that the Zag-rows (the one that begin and end with a C-brick before the turns) fold from left to right. Let us now review the possible environments for each module and the bricks they are expected to fold into in each possible environment (for each of them we refer to the figure providing with the corresponding folding tree certificate):

- Module A, First Half-Adder:
  - In Zig-rows (←):
    - *when Module A folds in the rightmost column of a Zig-row,* its environment is *to the right:* always DT; and *above:* either B2,C0 or B2,C1. For each of these two environments, Module A is expected to fold into the following brick:

		Bricks above:	
		B2,C0	B2,C1
Brick to the right:	DT	A01 (Figure S.3)	A11 (Figure S.4)

    - *when Module A folds in an inner column of a Zig-row,* its environment is *to the right:* either D0 or D1; *above:* either B2,C0 or B2,C1. For each of these four environments, Module A is expected to fold into the following brick:

		Bricks above:	
		B2,C0	B2,C1
Brick to the right:	D0	A00 (Figure S.5)	A10 (Figure S.7)
	D1	A01 (Figure S.6)	A11 (Figure S.8)

  - ∗ *when Module A folds in the leftmost column of a Zig-row,* its environment is *to the right:* either D0 or D1; and *above:* either BT,C0 or BT,C1. For each of these four environments, Module A is expected to fold into the following brick:

		Bricks above:	
		BT,C0	BT,C1
Brick to the right:	D0	A00 (Figure S.9)	A10 (Figure S.11)
	D1	A01 (Figure S.10)	A11 (Figure S.12)

- In Zag-rows (→): Module A folds in a 4 × 3-region surrounded *to the left:* always by D2; and *above:* by either C00,B0 or C01,B1 or C10,B0 or C11,B1. For each of these four environments, Module A is expected to fold into the following brick:

		Bricks above:			
		C00,B0	C01,B1	C10,B0	C11,B1
Brick to the left:	D2	A0 (Figure S.13)	A1 (Figure S.14)	A1 (Figure S.15)	A0 (Figure S.16)

- Module B, Left Turn:
  - In the inner columns of a Zig-row (←): Module B folds in a 6 × 3-region surrounded *to the right:* by either A00, A01, A10, or A11 and *above:* by A0,B2 or A1,B2. For each of these height environments, Module B is expected to fold into the following brick:

		Bricks above:	
		A0,B2	A1,B2
Brick to the right:	A00	B0 (Figure S.17)	B0 (Figure S.18)
	A01	B0 (Figure S.19)	B0 (Figure S.20)
	A10	B0 (Figure S.21)	B0 (Figure S.22)
	A11	B1 (Figure S.23)	B1 (Figure S.24)

  - In the leftmost column of a Zig-row (←): Module B folds in a 3 × 6-region surrounded *to the right:* by either A00, A01, A10, or A11 and *above:* by BT. For each of these four environments, Module B is expected to fold into the following brick:

		Bricks above:
		BT
Brick to the right:	A00	BT (Figure S.25)
	A01	BT (Figure S.26)
	A10	BT (Figure S.27)
	A11	Halting non-deterministic configuration (Figure S.28)

  - In the columns of a Zag-row (→): Module B folds in a 6 × 2-region surrounded *to the left:* by either A0 or A1 and *above:* by B0,A00, or B0,A01, or B0,A10 or B1,A11. For each of these height environments, Module B is expected to fold into the brick B2:

		Bricks above:			
		B0,A00	B0,A01	B0,A10	B1,A11
Brick to the left:	A0	B2 (Figure S.29)	B2 (Figure S.30)	B2 (Figure S.31)	B2 (Figure S.32)
	A1	B2 (Figure S.33)	B2 (Figure S.34)	B2 (Figure S.35)	B2 (Figure S.36)

- Module C, Second Half-Adder:
  - In Zig-rows (←): Module C folds in a 4 × 3-region surrounded *to the right:* either by B0 or B1 ; and *above:* by either D2,A0 or D2,A1. For each of these four environments, Module C is expected to fold into the following brick:

		Bricks above:	
		D2,A0	D2,A1
Brick to the right:	B0	C00 (Figure S.37)	C10 (Figure S.38)
	B1	C01 (Figure S.39)	C11 (Figure S.40)

  - In Zag-rows (→):

* *when Module C folds in the leftmost column of a Zag-row,* its environment is *to the left:* always BT; and *above:* either A00,D0 or A01,D1 or A10,D0 or A11,D1. For each of these four environments, Module C is expected to fold into the following brick:

		Bricks above:			
		A00,D0	A01,D1	A10,D0	A11,D1
Brick to the left:	BT	C0 (Figure S.41)	C1 (Figure S.42)	C1 (Figure S.43)	Never occurs

* *when Module C folds in an inner column of a Zag-row,* its environment is *to the left:* always B2; and *above:* either A00,D0 or A01,D1 or A10,D0 or A11,D1. For each of these four environments, Module C is expected to fold into the following brick:

		Bricks above:			
		A00,D0	A01,D1	A10,D0	A11,D1
Brick to the left:	B2	C0 (Figure S.44)	C1 (Figure S.45)	C1 (Figure S.46)	C0 (Figure S.47)

* *when Module C folds in the rightmost column of a Zag-row,* its environment is *to the right:* always B2; *above:* either A01,DT or A11,DT. For each of these two environments, Module C is expected to fold into the following brick:

		Bricks above:	
		A01,DT	A11,DT
Brick to the right:	B2	C1 (Figure S.48)	C0 (Figure S.49)

- Module D, Right Turn:
  - In a column of a Zig-row (←): Module D folds in a 6 × 3-region surrounded *to the right:* by either C00, C01, C10, or C11 and *above:* by C0,D2 or C1,D2. For each of these height environments, Module D is expected to fold into the following brick:

		Bricks above:	
		C0,D2	C1,D2
Brick to the right:	C00	D0 (Figure S.50)	D0 (Figure S.51)
	C01	D0 (Figure S.52)	D0 (Figure S.53)
	C10	D0 (Figure S.54)	D0 (Figure S.55)
	C11	D1 (Figure S.56)	D1 (Figure S.57)

  - In an inner column of a Zag-row (→): Module D folds in a 6 × 3-region surrounded *to the left:* by either C0 or C1 and *above:* by D0,C00 or D0,C01 or D0,C10 or D1,C11. For each of these height environments, Module D is expected to fold into the brick D2:

		Bricks above:			
		D0,C00	D0,C01	D0,C10	D1,C11
Brick to the left:	C0	D2 (Figure S.58)	D2 (Figure S.59)	D2 (Figure S.60)	D2 (Figure S.61)
	C1	D2 (Figure S.62)	D2 (Figure S.63)	D2 (Figure S.64)	D2 (Figure S.65)

  - In the rightmost column of a Zag-row (→): Module D folds in a 3 × 6-region surrounded *to the left:* by either C0 or C1 and *above:* by DT. For each of these two environments, Module D is expected to fold into the brick DT:

		Bricks above:
		DT
Brick to the left:	C0	DT (Figure S.66)
	C1	DT (Figure S.67)

Figure S.1 presents the positions of all the possible bricks in all the possible environments (written in white) in the folding of a 5-bits counter that contains all of them.

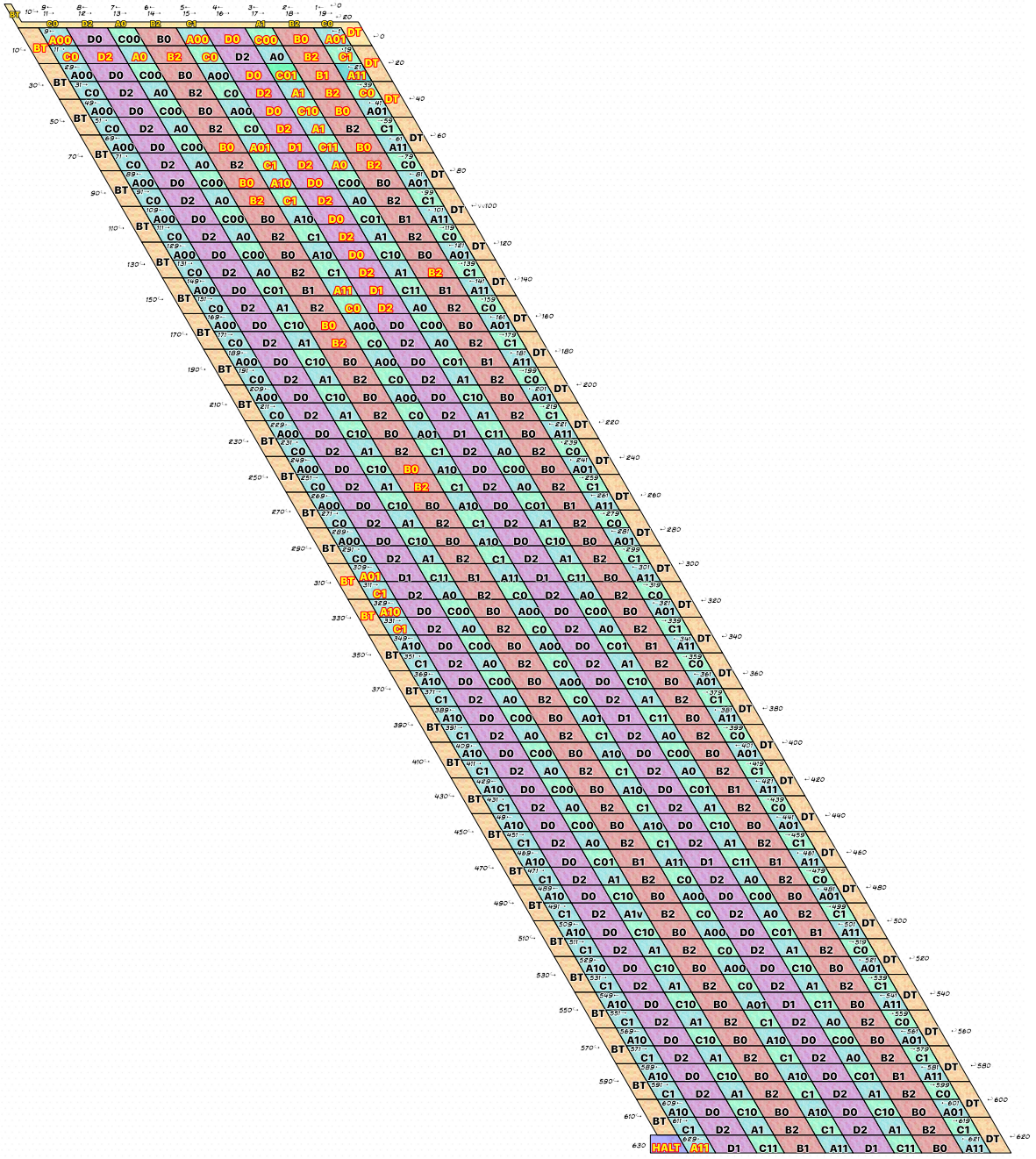


Figure S.1. The positions in the folding of the 5-bits counter of all the bricks analyzed in the following.

S.1.2. Proof of the Correctness of the Local Folding

S.1.2.1. Zig Rows

The following lemmas prove that the molecule folds as claimed in the zig-rows.

Lemma 1 (Modules A and C-inner column-Zig row-no carry propagation)**.** *When folded in an inner column of a Zig-row from the output configurations of the previous brick Dc (resp. Bc) and under a brick Cx (resp. Ax) s.t.* $x + c \leqslant 1$ *(i.e., no carry propagation), module A (resp. C) folds into the brick* $A(x + c)$ *(resp.* $C(x + c)$*) with output configurations* β*.*

Proof. This property is proved by the folding certificates in Figures:

- Module A: Figures S.3 and S.5– S.7.
- Module C: Figures S.37–S.39. □

Lemma 2 (Module A-leftmost column-no carry propagation)**.** *When folded in the leftmost column of a Zig-row from the output configurations of the previous brick Dc and under a brick Cx s.t.* $x + c \leqslant 1$ *(i.e., no carry propagation), module A folds into the brick* $A(x + c)$ *with output configuration* γ*.*

Proof. The folding certificates in Figures S.9–S.11 prove this property. □

Lemma 3 (Modules A and C-Zig row (all columns)-with carry propagation)**.** *When folded in a Zig-row from the output configurations of the previous brick D1 (resp. B1) and under a brick C1 (resp. A1), module A (resp. C) folds into the brick A11 (resp. C11) with output configuration* α' *(resp.* α'''*).*

Proof. This property is proved by the folding certificates in Figures:

- Module A: Figures S.4, S.8 and S.12.
- Module C: Figure S.40. □

Lemma 4 (Modules B and D-inner column-Zig row-no carry propagation)**.** *When folded in an inner column of a Zig-row from the output configurations of the previous brick Axc (resp. Cxc) and under bricks CzD2 (resp. AzB2) s.t.* $x + c \leqslant 1$*, module D (resp. B) folds into the brick D0 (resp. B0) with output configurations:* $\theta \cup \theta'$ *if* $z = 0$*; and* γ' *if* $z = 1$*.*

Proof. This property is proved by the folding certificates in Figures:

- Module D: Figures S.50, S.52 and S.54 for $z = 0$; and Figures S.51, S.53 and S.55 for $z = 1$.
- Module B: Figures S.17, S.19 and S.21 for $z = 0$; Figures S.18, S.20 and S.22 for $z = 1$. □

Lemma 5 (Modules B and D-inner column-Zig row-with carry propagation)**.** *When folded in an inner column of a Zig-row from the output configurations of the previous brick A11 (resp. C11) and under a brick D2 (resp. B2), module D (resp. B) folds into the brick D1 (resp. B1) with output configuration* α''*.*

Proof. This property is proved by the folding certificates in Figures:

- Module D: Figures S.56 and S.57.
- Module B: Figures S.23 and S.24. □

Lemma 6 (Modules B-leftmost column-Zig row-with no carry propagation)**.** *When folded in the leftmost column of a Zig-row from the output configurations of the previous brick Axc with* $x + c \leqslant 1$ *and under a brick BT, module B folds into the brick BT with output configuration* λ_2'*.*

Proof. This property is proved by the folding certificates in Figures S.25–S.27. □

Lemma 7 (Modules B-leftmost column-Zig row-with carry propagation)**.** *When folded in the leftmost column of a Zig-row from the output configurations of the previous brick A11 and under a brick BT, module B folds into the halting non-deterministic configuration with output configuration.*

Proof. This property is proved by the folding certificates in Figure S.28. □

S.1.2.2. Zag Rows

The following lemmas show that the molecule folds as expected in the Zag-rows. As Zag-row does not have to deal with carry propagation, its analysis is significantly simpler than the Zig-rows and fewer lemmas are needed to prove the correctness of the folding.

Lemma 8 (Modules A and C-Zag row)**.** *When folded in Zag-row from the output configurations of the preceding brick BT or B2 and under a brick Cxc (resp. Axc), module A (resp. C) folds into the brick Az (resp. Cz) where* $z = x + c \mod 2$ *with output configuration* λ_3 , λ'_3 , λ_4 , λ'_4 *or* λ_5 *which only differ by the positions of their bonds with the environment.*

Proof. This property is proved by the folding certificates in Figures:

- Module A: Figures S.13–S.16.
- Module C: Figures S.41–S.43 for the leftmost column; Figures S.44–S.47 for the inner columns; Figures S.48 and S.49 for the rightmost column. □

Lemma 9 (Modules A and C-Zag row)**.** *When folded in Zag-row from the output configurations of the preceding brick Ax (resp. Cx) and under the bricks BzAyc (resp. DzCyc in an inner column or DT in the rightmost column), module B (resp. D) folds into the brick B2 (resp. D2) with output configuration* λ'_2 *if* $y + c \leqslant 1$ *and output configurations* $\mu \cup \mu'$ *if* $y + c = 2$*.*

Proof. This property is proved by the folding certificates in Figures:

- Module B: Figures S.29–S.31 for $x = 0$ and $y + c \leqslant 1$; Figures S.33–S.35 for $x = 1$ and $y + c \leqslant 1$; Figures S.32 and S.36 for $y + c = 2$.
- Module D: Figures S.58–S.60 for $x = 0$ and $y + c \leqslant 1$; Figures S.62–S.64 for $x = 1$ and $y + c \leqslant 1$; Figures S.61 and S.65 for $y + c = 2$; Figures S.66 and S.67 for the rightmost column. □

Lemma 10 (Modules A and C-inner column-Zag row-no carry propagation)**.** *When folded in an inner column of a Zig-row from the output configurations of the preceding brick Dc (resp. Bc) and under a brick Cx (resp. Ax) s.t.* $x + c \leqslant 1$ *(i.e., no carry propagation), module A (resp. C) folds into the brick* $A(x + c)$ *(resp.* $C(x + c)$*) with output configurations* β*.*

Proof. This property is proved by the folding certificates in Figures:

- Module A: Figures S.3 and S.5–S.7.
- Module C: Figures S.37–S.39. □

All the folding tree certificates are given in the next supplementary section.

S.2. Folding Certificates

The following sections contain the certificates for the folding of each of modules in all possible environments in a human-readable and -checkable form (up to zooming in the PDF file for some of them). When reading these certificates, the top circles represent the input nascent configurations inherited from the end of folding of the previous module (there might be several of them). Then, the number of new bonds of each of the local configurations is written in the top left corner of that configuration. The path in bold shows the configurations with the maximum number of bonds, that is to say the only ones that are allowed to continue to grow by the inertial dynamics. To improve readability, we group in the same ball all paths that share a common prefix up to their last bond; the free end of each path in the group is drawn in random color; the size of the group is given in the lower right corner for cross-checking. The last level of the folding tree represents the output nascent configurations, that is the only configurations allowed to start the folding of the next module.

S.2.1. Folding Certificate for the Output Nascent Configuration of the Seed

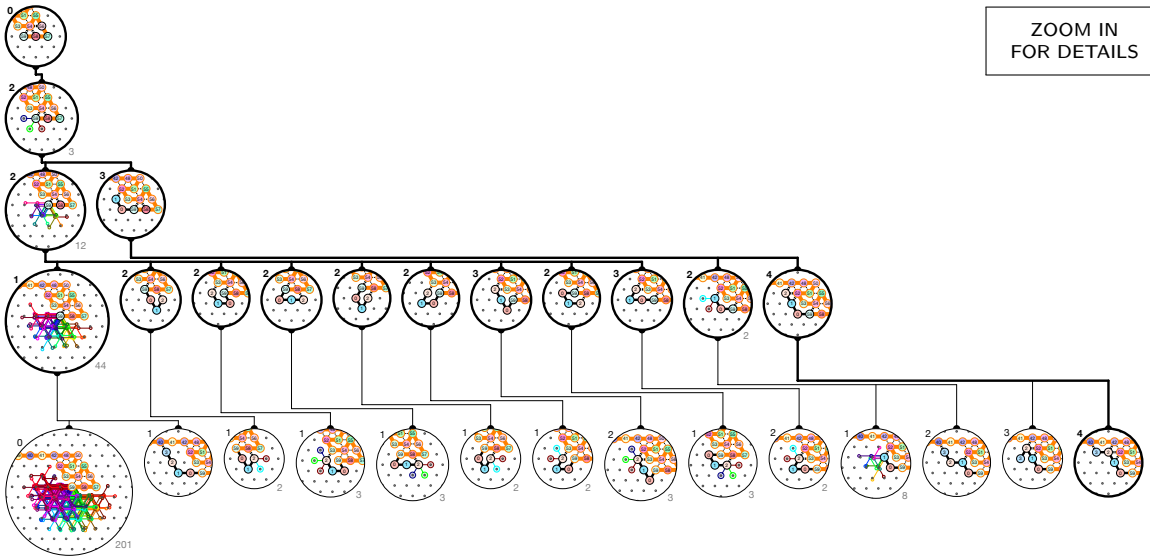
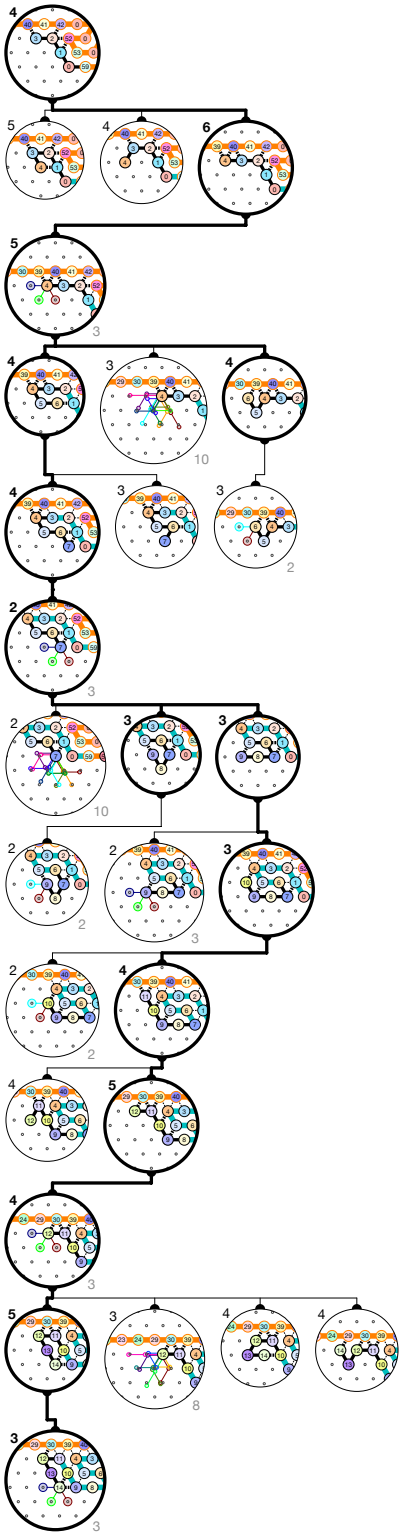


Figure S.2. The output nascent configurations of the seed brick–Region #0 in the 5-bits counter. The first level consists of the seed configuration σ. The next three levels consist of all the possible extensions of σ by $i = 1..3$ beads (recall $3 = \delta - 1$). The last level is the first of the inertial dynamics: it outputs only one nascent configuration, α , with maximum number of bonds.

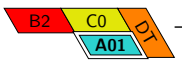
S.2.2. Folding Certificates for Module A: First Half-Adder

Input nascent configurations: α

ZOOM IN
FOR DETAILS



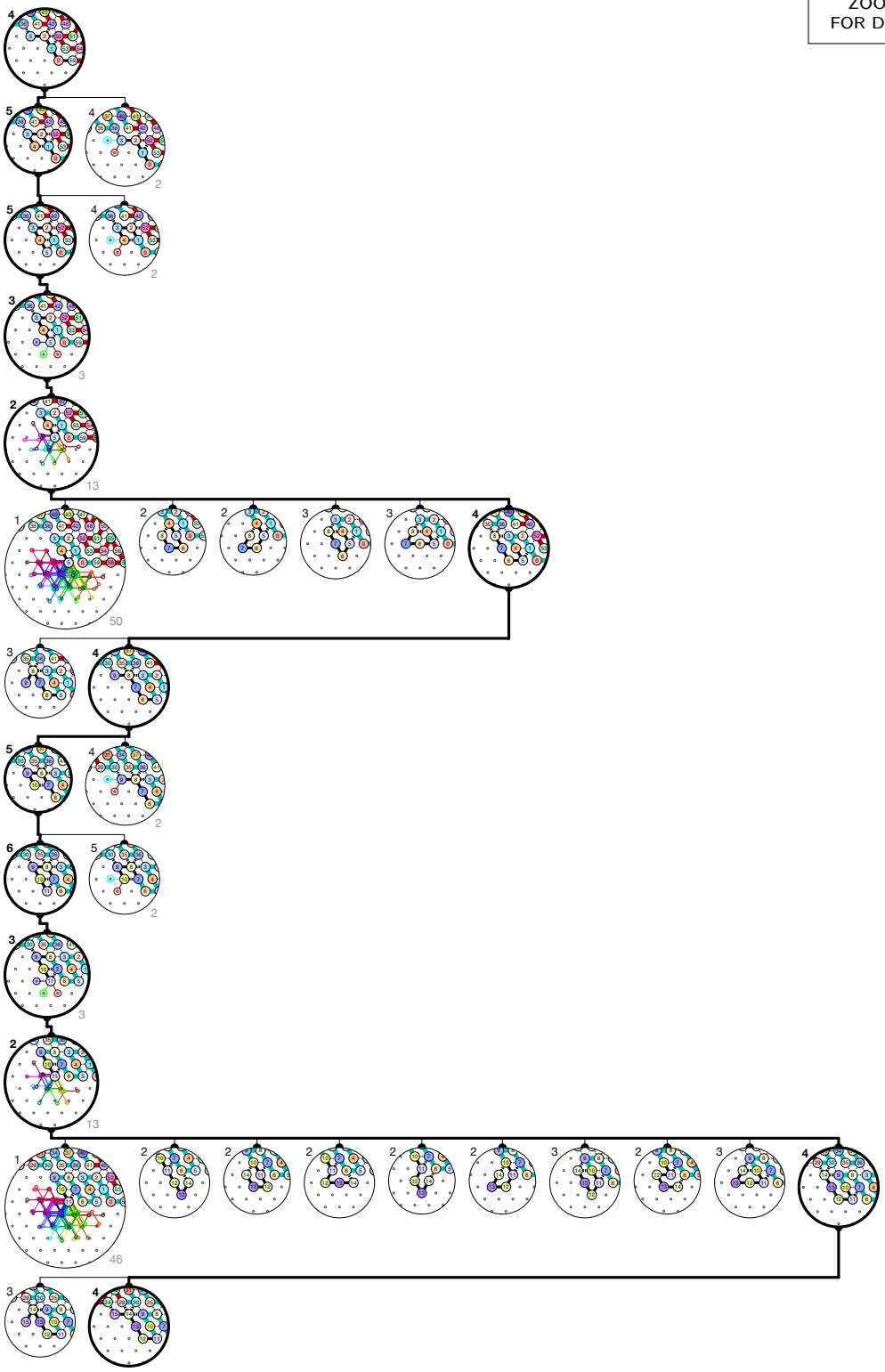
Output nascent configurations: β

Figure S.3. Folding of Brick A01 in 

–First occurrence in 5-bits counter: Region #1.

Input nascent configurations: α

ZOOM IN
FOR DETAILS

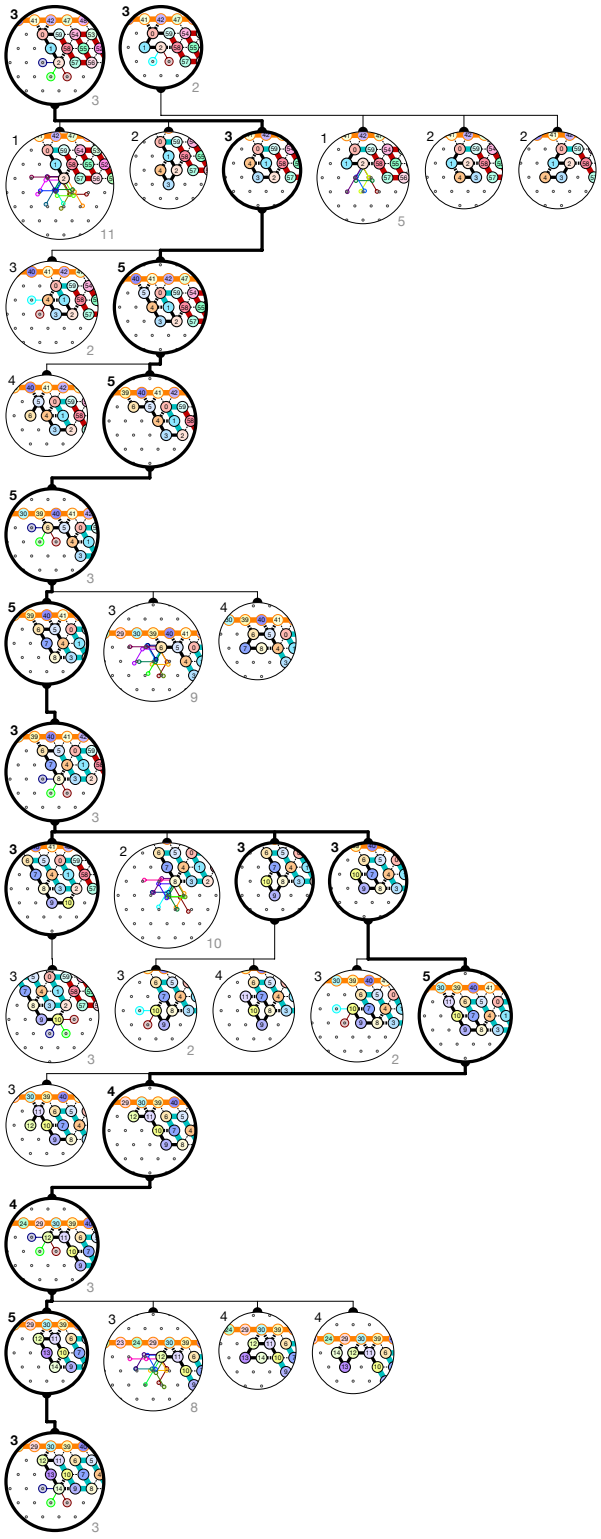


Output nascent configurations: α'

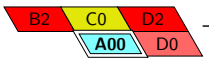
Figure S.4. Folding of Brick A11 in B2 C1 DT A11 –First occurrence in 5-bits counter: Region #21.

Input nascent configurations: $\theta \cup \theta'$

ZOOM IN
FOR DETAILS



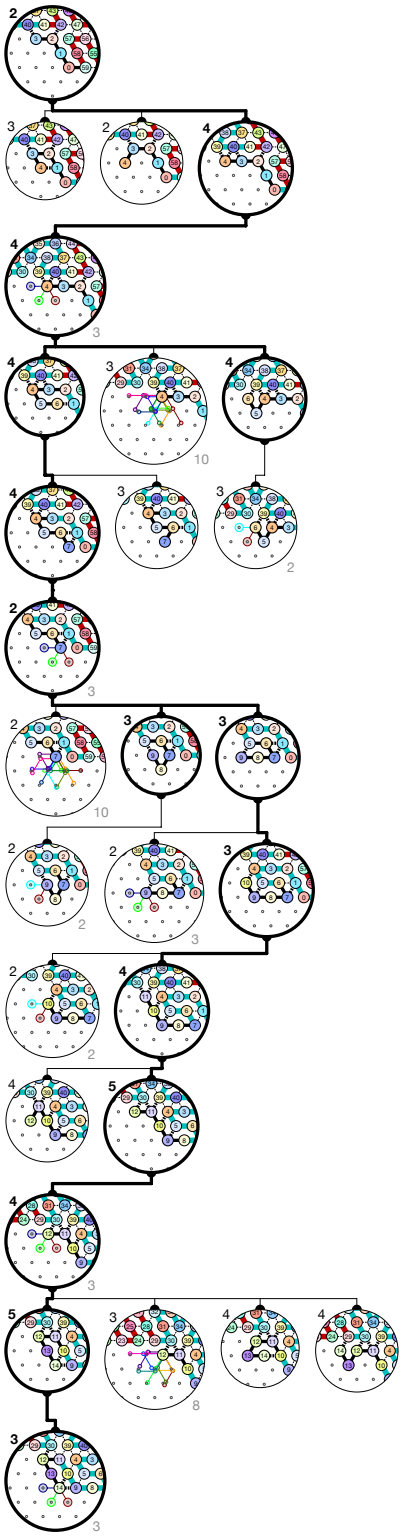
Output nascent configurations: β

Figure S.5. Folding of Brick A00 in 

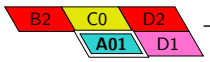
–First occurrence in 5-bits counter: Region #5.

Input nascent configurations: α''

ZOOM IN
FOR DETAILS



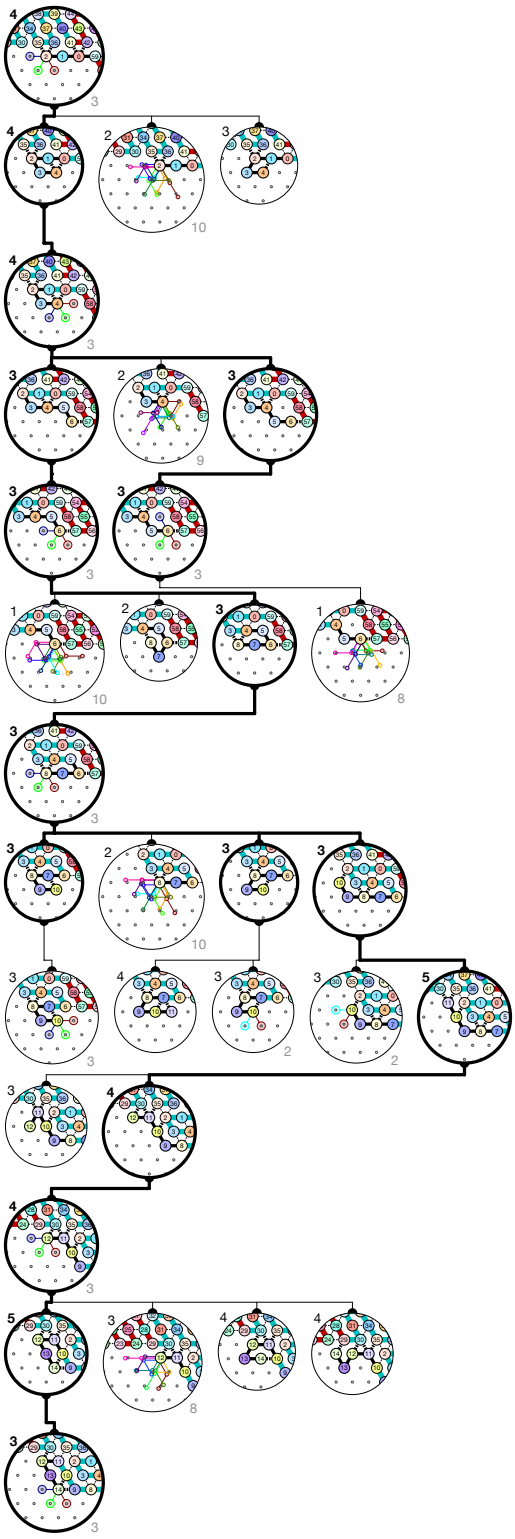
Output nascent configurations: β

Figure S.6. Folding of Brick A01 in 

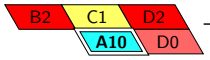
–First occurrence in 5-bits counter: Region #65.

Input nascent configurations: γ'

ZOOM IN
FOR DETAILS



Output nascent configurations: β

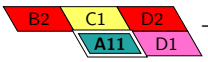
Figure S.7. Folding of Brick A10 in 

–First occurrence in 5-bits counter: Region #85.

Input nascent configurations: α''

ZOOM IN
FOR DETAILS

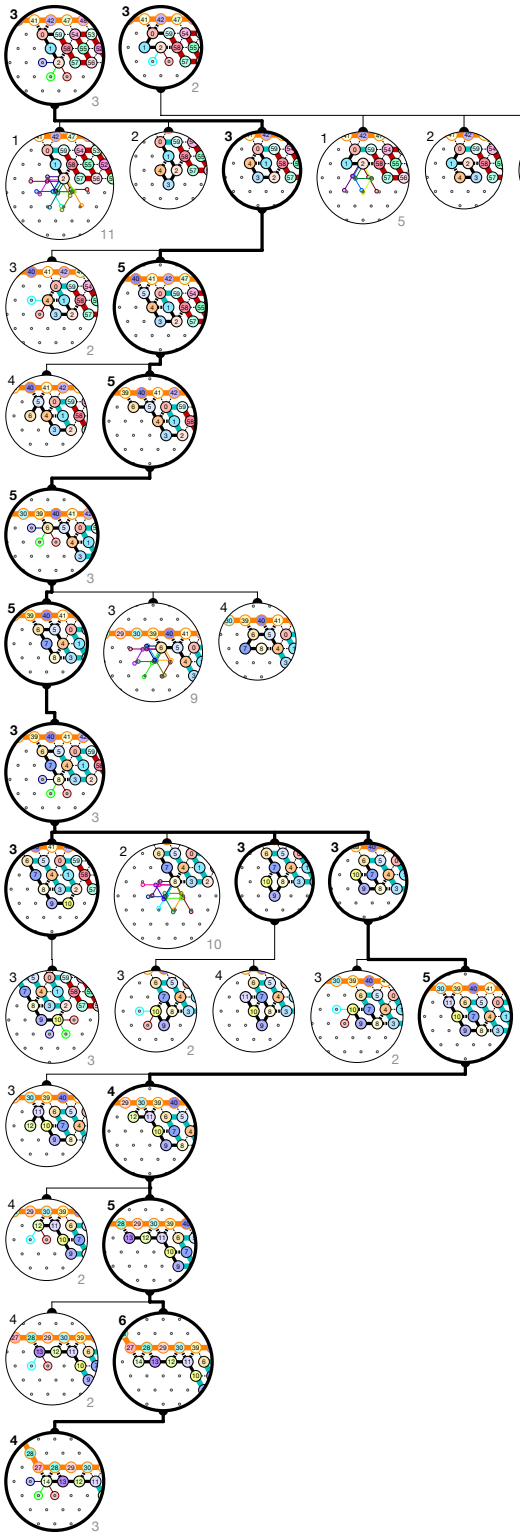
Output nascent configurations: α'

Figure S.8. Folding of Brick A11 in 

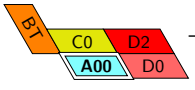
–First occurrence in 5-bits counter: Region #145.

Input nascent configurations: $\theta \cup \theta'$

ZOOM IN
FOR DETAILS



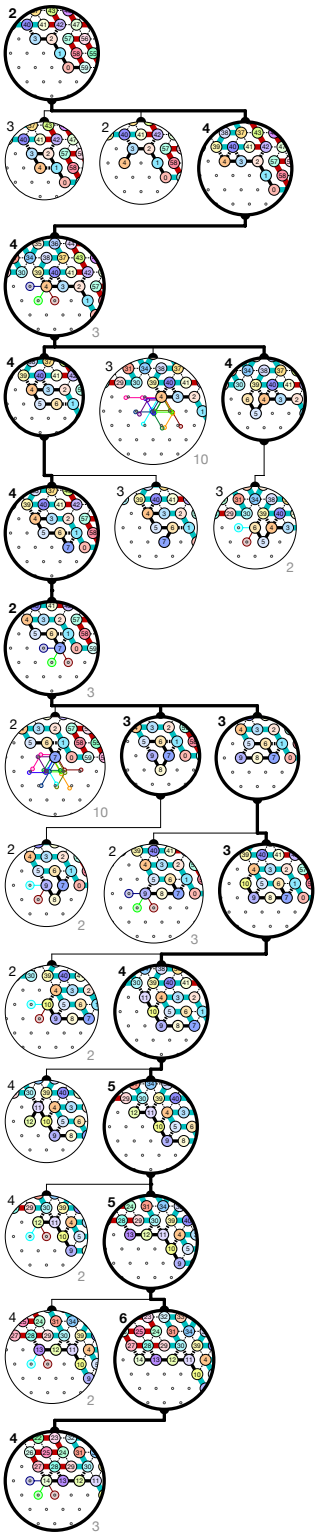
Output nascent configurations: γ

Figure S.9. Folding of Brick A00 in 

–First occurrence in 5-bits counter: Region #9.

Input nascent configurations: α''

ZOOM IN
FOR DETAILS

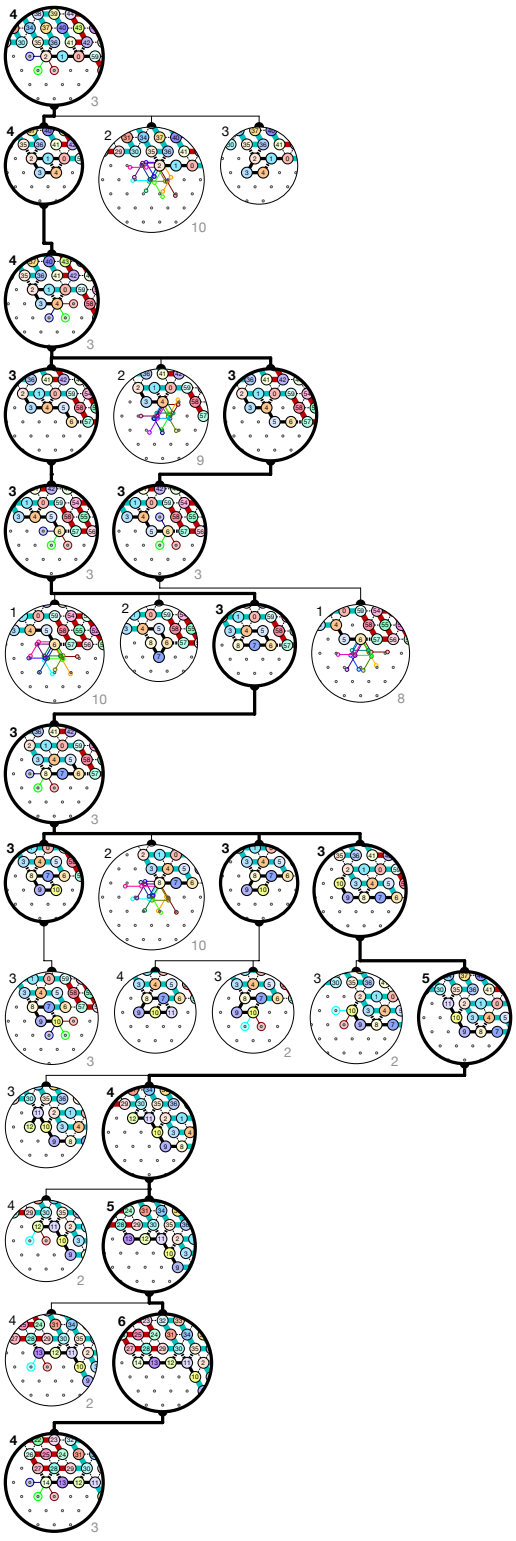


Output nascent configurations: γ

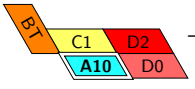
Figure S.10. Folding of Brick A01 in 

–First occurrence in 5-bits counter: Region #309.

Input nascent configurations: γ'



Output nascent configurations: γ

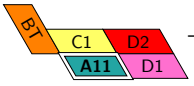
Figure S.11. Folding of Brick A10 in 

–First occurrence in 5-bits counter: Region #329.

Input nascent configurations: α''

ZOOM IN
FOR DETAILS

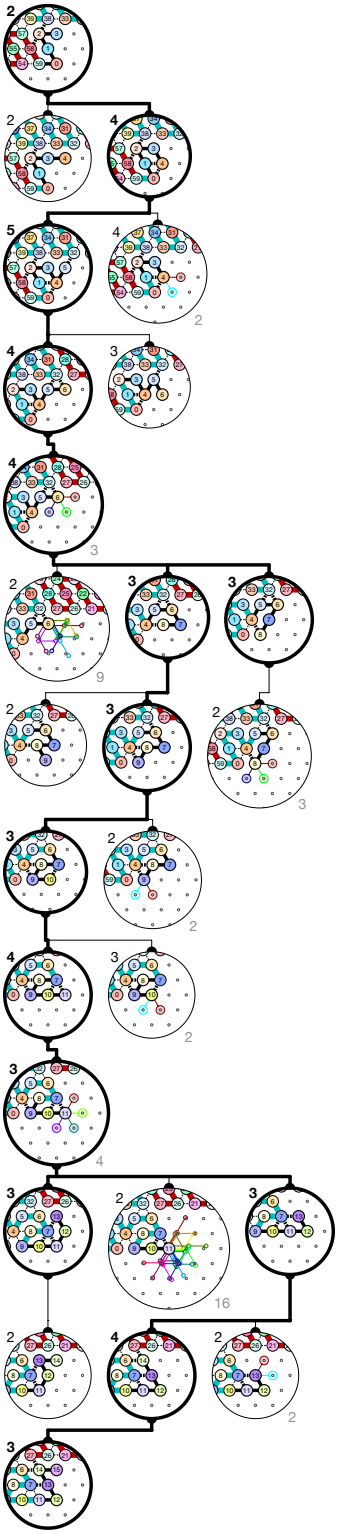
Output nascent configurations: α'

Figure S.12. Folding of Brick A11 in 

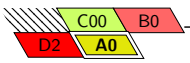
–First occurrence in 5-bits counter: Region #629.

Input nascent configurations: λ'_2

ZOOM IN
FOR DETAILS

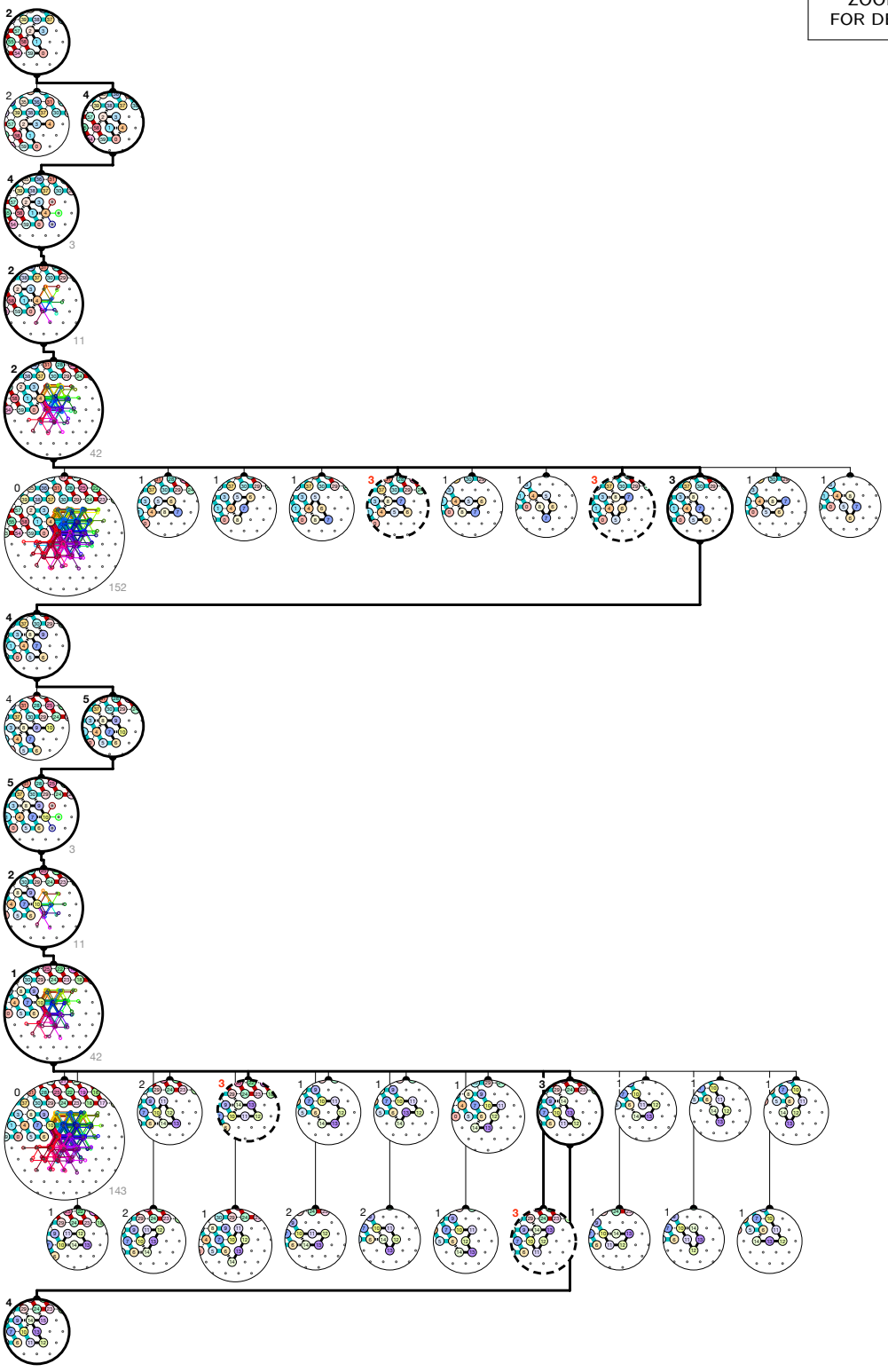


Output nascent configurations: λ_3

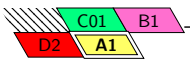
Figure S.13. Folding of Brick A0 in 

–First occurrence in 5-bits counter: Region #13.

Input nascent configurations: λ_2'



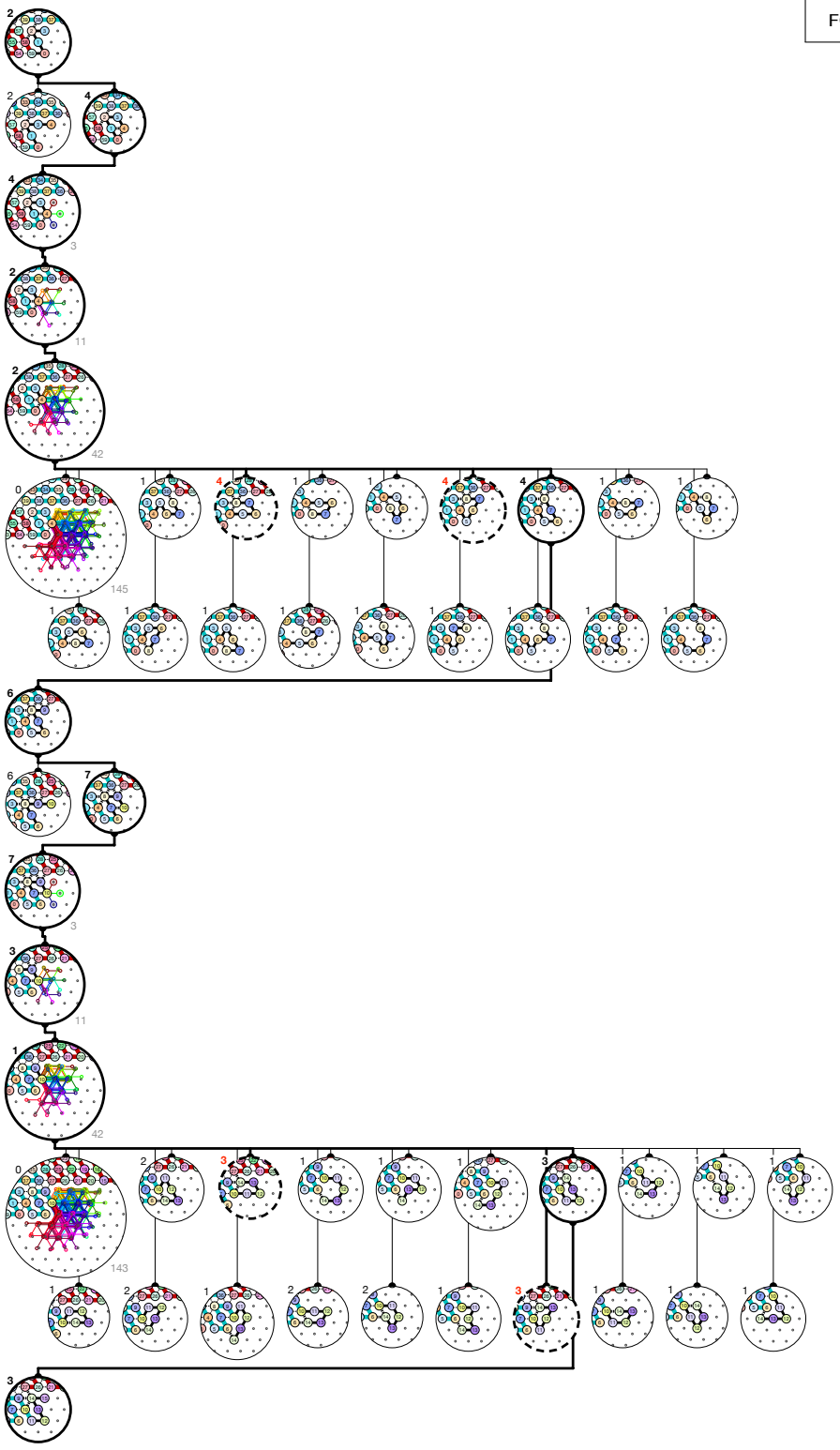
Output nascent configurations: λ_4

Figure S.14. Folding of Brick A1 in 

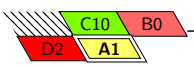
–First occurrence in 5-bits counter: Region #37.

Input nascent configurations: λ'_2

ZOOM IN
FOR DETAILS



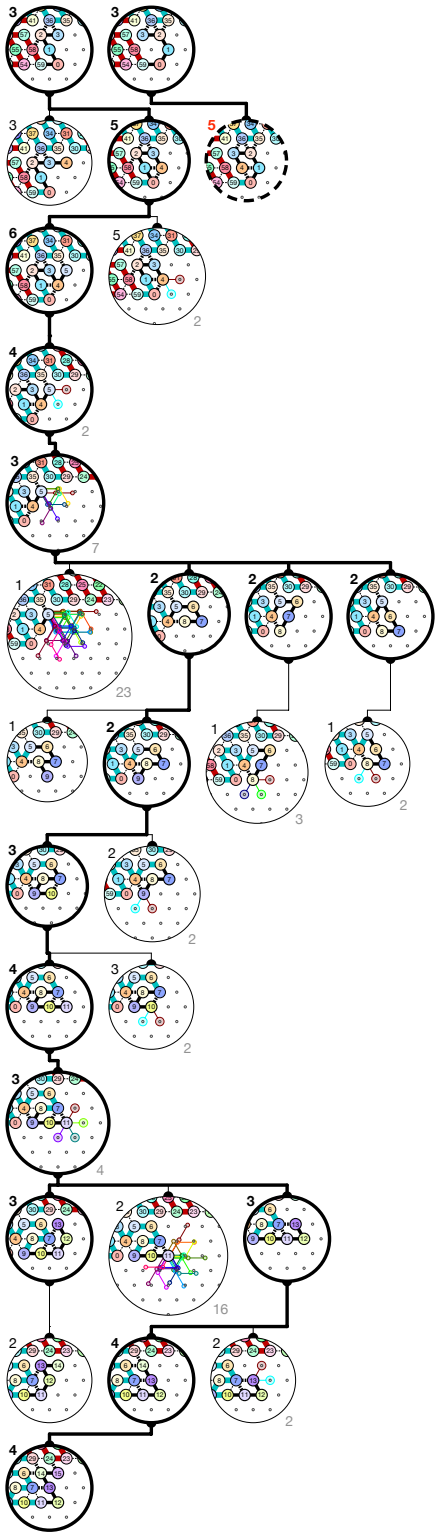
Output nascent configurations: λ'_3

Figure S.15. Folding of Brick A1 in 

–First occurrence in 5-bits counter: Region #57.

Input nascent configurations: $\mu \cup \mu'$

ZOOM IN
FOR DETAILS



Output nascent configurations: λ'_4

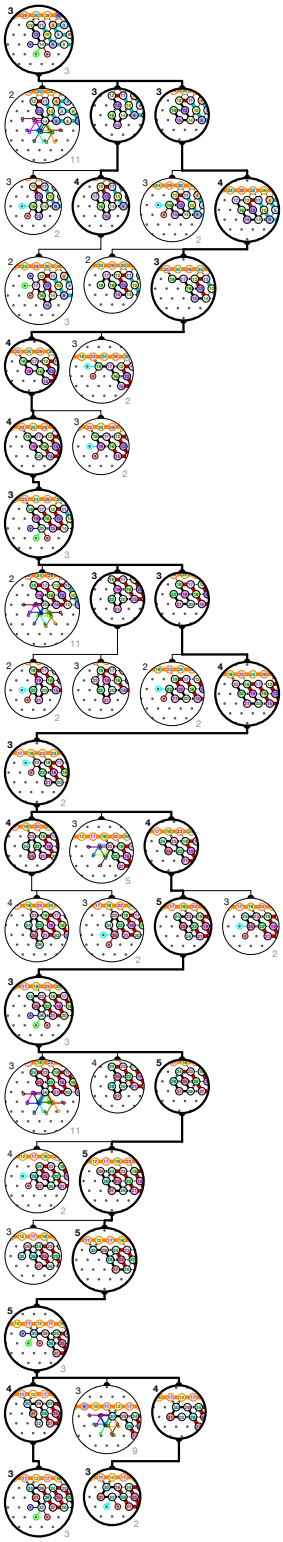
Figure S.16. Folding of Brick A0 in 

–First occurrence in 5-bits counter: Region #77.

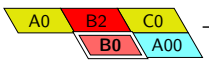
S.2.3. Folding certificates for Module B: Left Turn

Input nascent configurations: β

ZOOM IN
FOR DETAILS



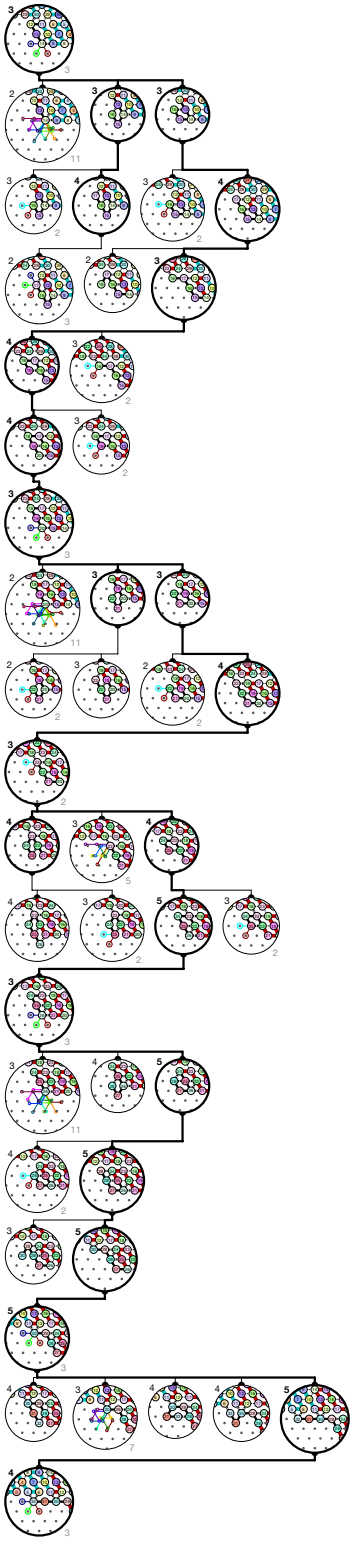
Output nascent configurations: $\theta \cup \theta'$

Figure S.17. Folding of Brick B0 in 

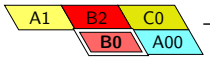
–First occurrence in 5-bits counter: Region #2.

Input nascent configurations: β

ZOOM IN
FOR DETAILS



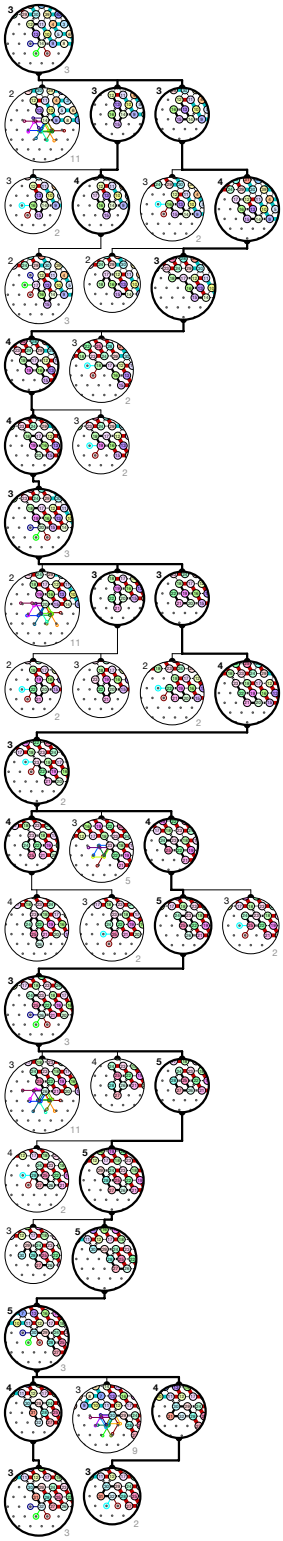
Output nascent configurations: γ'

Figure S.18. Folding of Brick B0 in 

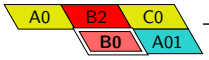
–First occurrence in 5-bits counter: Region #166.

Input nascent configurations: β

ZOOM IN
FOR DETAILS



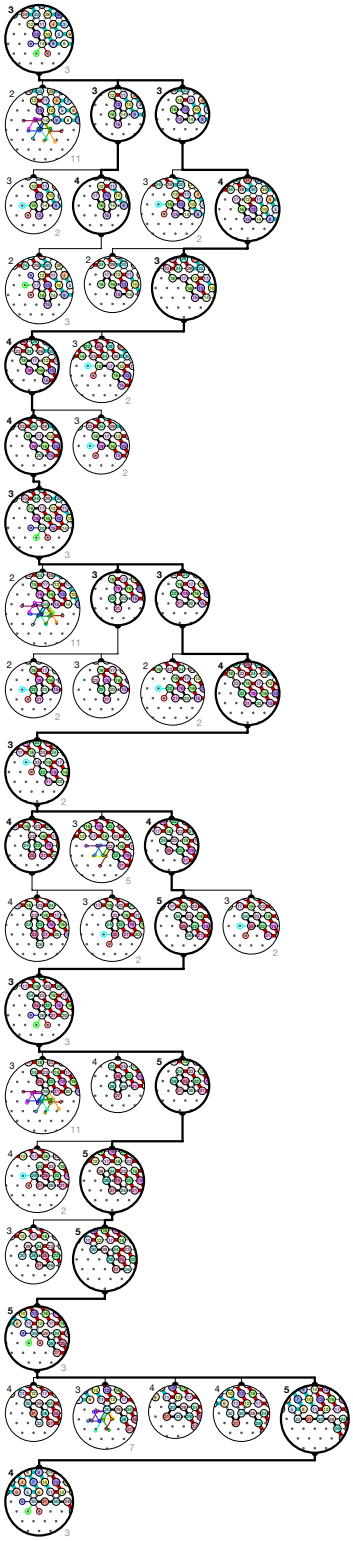
Output nascent configurations: $\theta \cup \theta'$

Figure S.19. Folding of Brick B0 in 

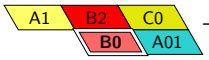
–First occurrence in 5-bits counter: Region #66.

Input nascent configurations: β

ZOOM IN
FOR DETAILS



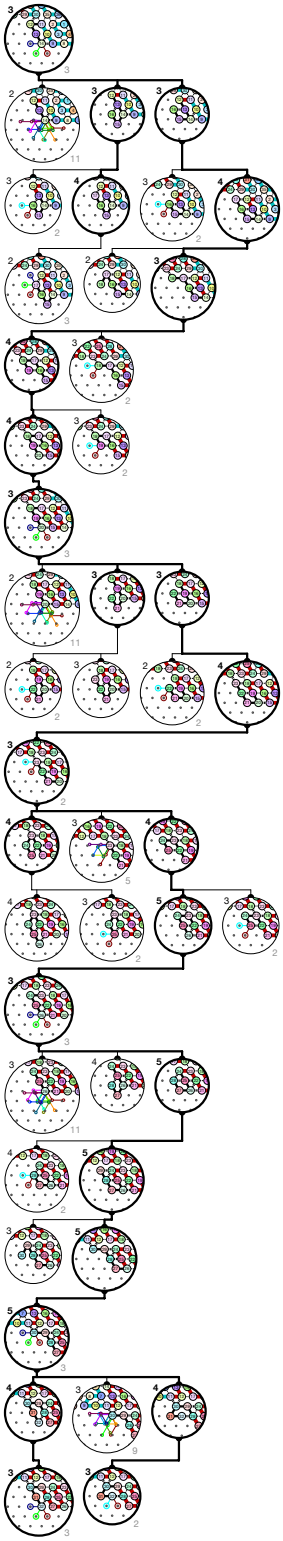
Output nascent configurations: γ'

Figure S.20. Folding of Brick B0 in 

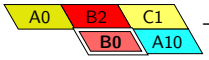
–First occurrence in 5-bits counter: Region #42.

Input nascent configurations: β

ZOOM IN
FOR DETAILS



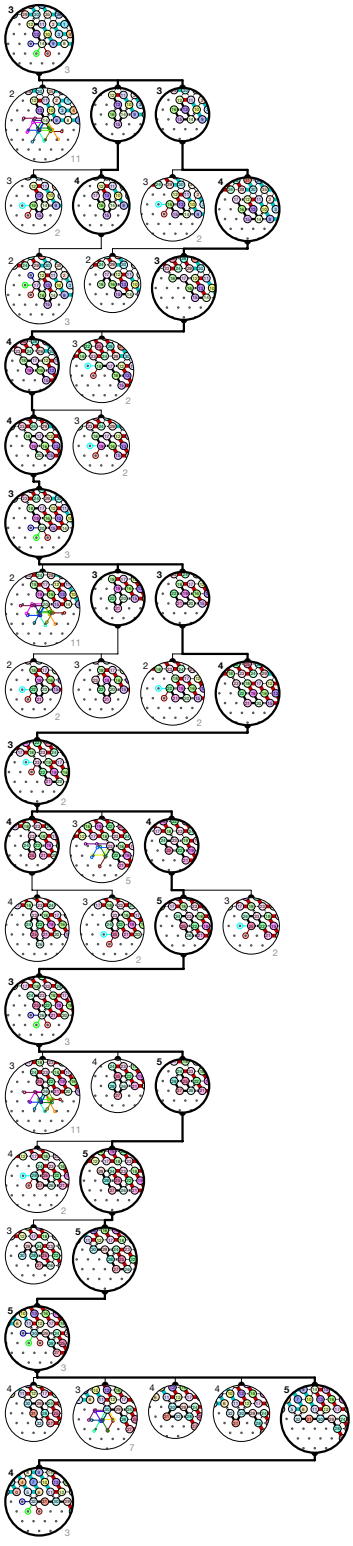
Output nascent configurations: $\theta \cup \theta'$

Figure S.21. Folding of Brick B0 in 

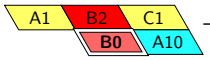
–First occurrence in 5-bits counter: Region #86.

Input nascent configurations: β

ZOOM IN
FOR DETAILS



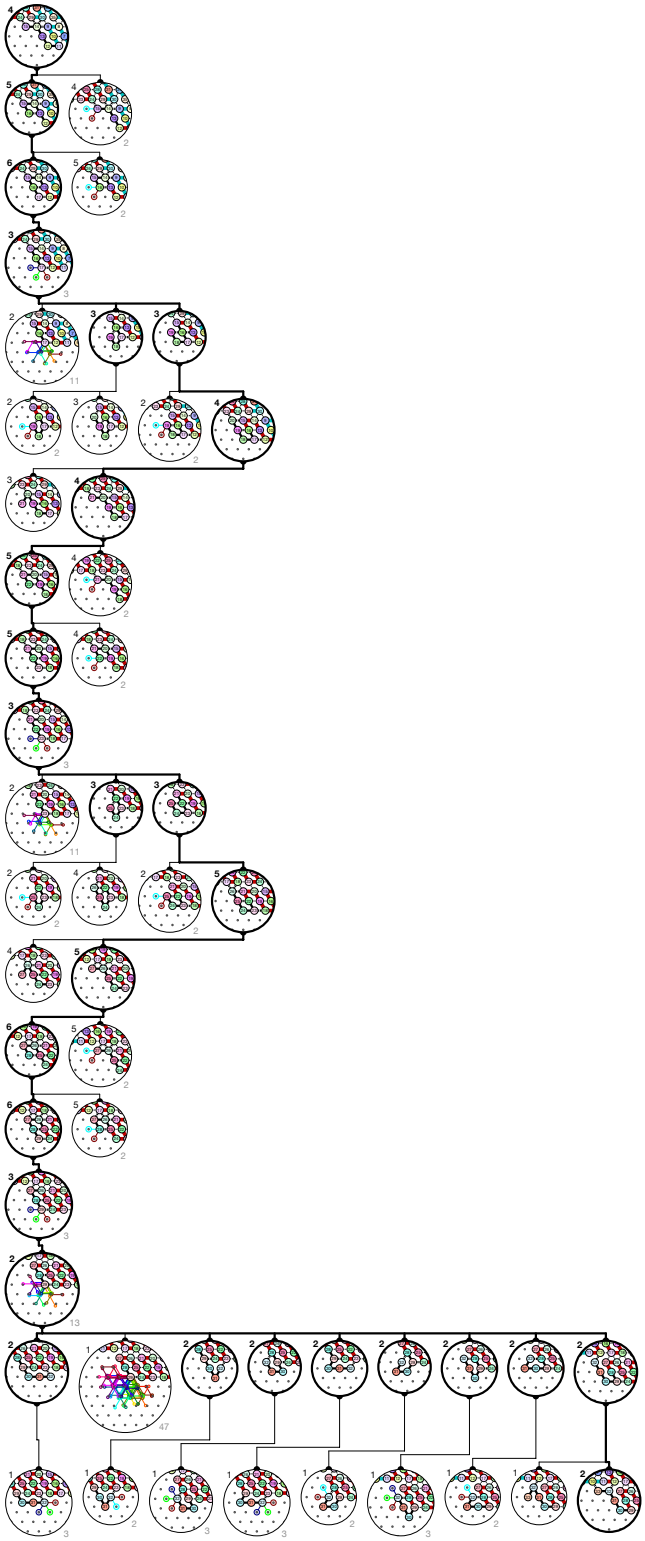
Output nascent configurations: γ'

Figure S.22. Folding of Brick B0 in 

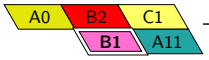
–First occurrence in 5-bits counter: Region #246.

Input nascent configurations: α'

ZOOM IN
FOR DETAILS



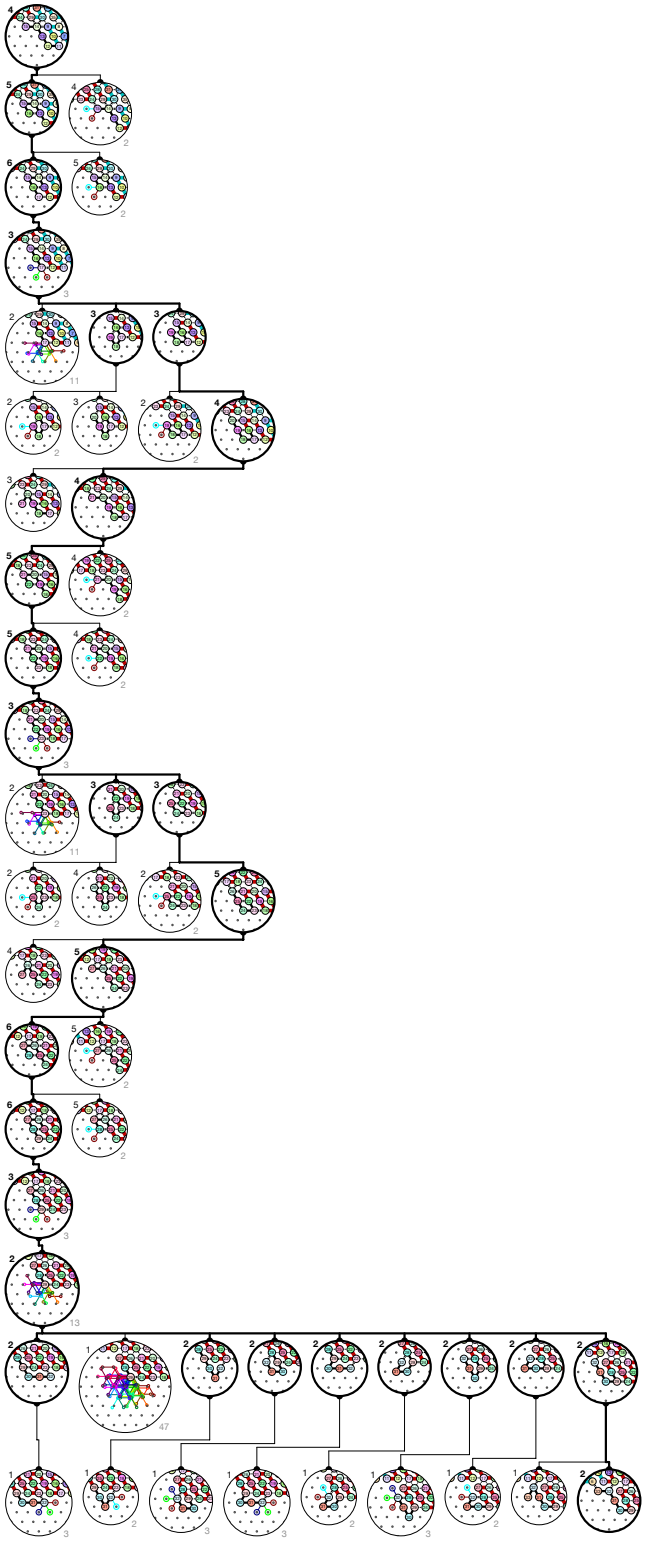
Output nascent configurations: α''

Figure S.23. Folding of Brick B1 in 

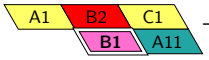
–First occurrence in 5-bits counter: Region #22.

Input nascent configurations: α'

ZOOM IN
FOR DETAILS



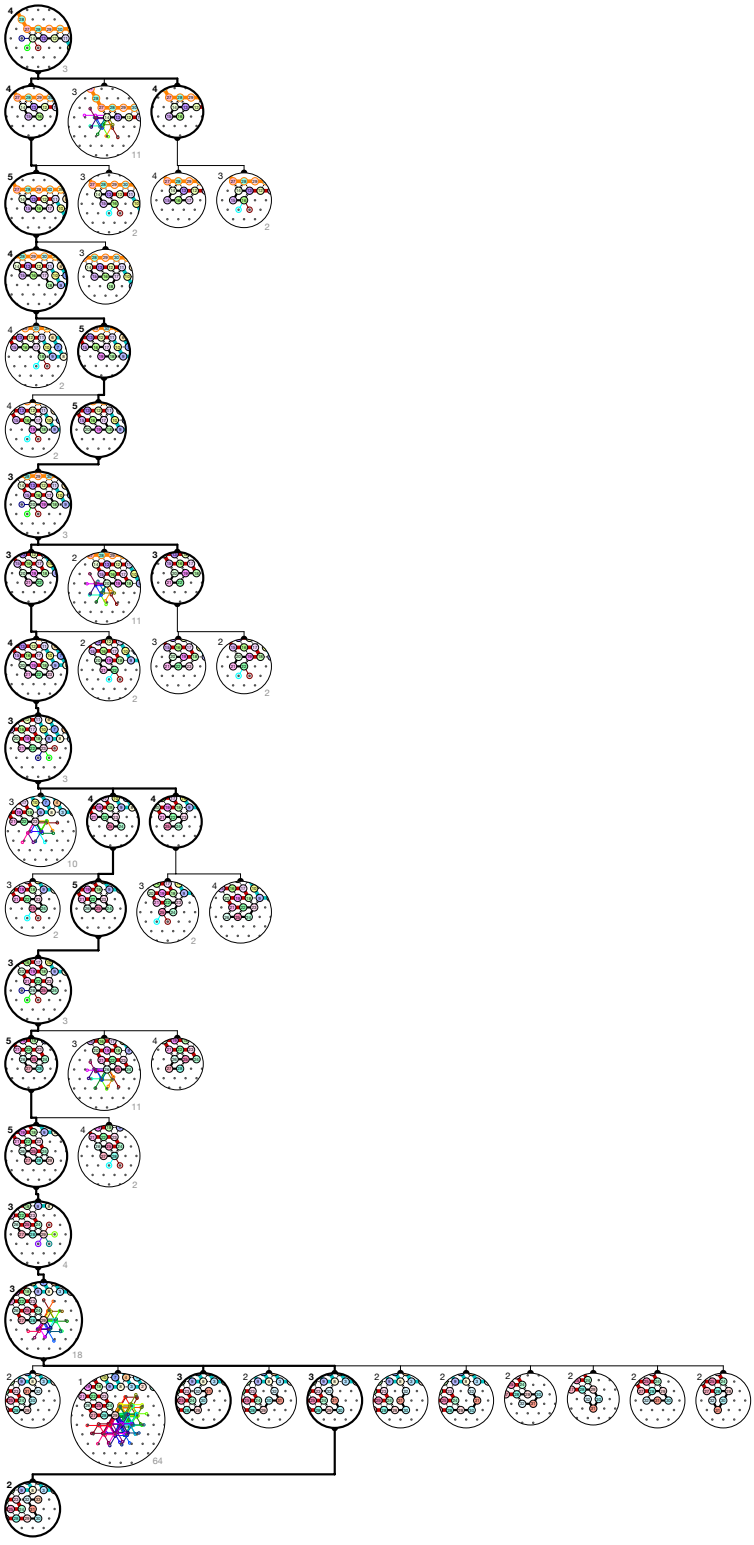
Output nascent configurations: α''

Figure S.24. Folding of Brick B1 in 

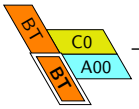
–First occurrence in 5-bits counter: Region #62.

Input nascent configurations: γ

ZOOM IN
FOR DETAILS



Output nascent configurations: λ_2

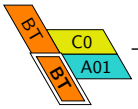
Figure S.25. Folding of Brick BT in 

–First occurrence in 5-bits counter: Region #10.

Input nascent configurations: γ

ZOOM IN
FOR DETAILS

Output nascent configurations: λ_2

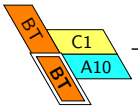
Figure S.26. Folding of Brick BT in 

–First occurrence in 5-bits counter: Region #310.

Input nascent configurations: γ

ZOOM IN
FOR DETAILS

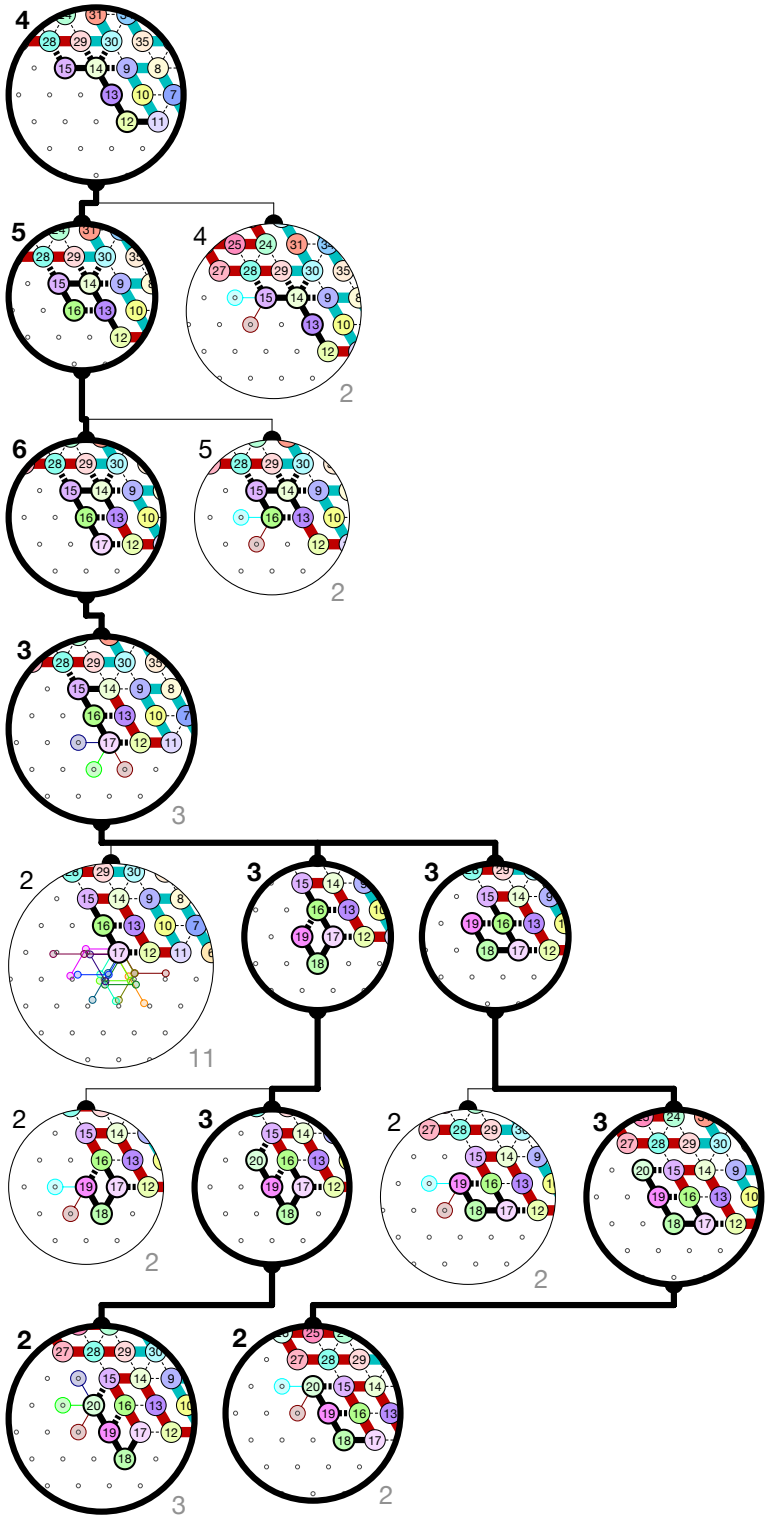
Output nascent configurations: λ_2

Figure S.27. Folding of Brick BT in 

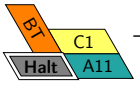
–First occurrence in 5-bits counter: Region #330.

Input nascent configurations: α'

ZOOM IN
FOR DETAILS



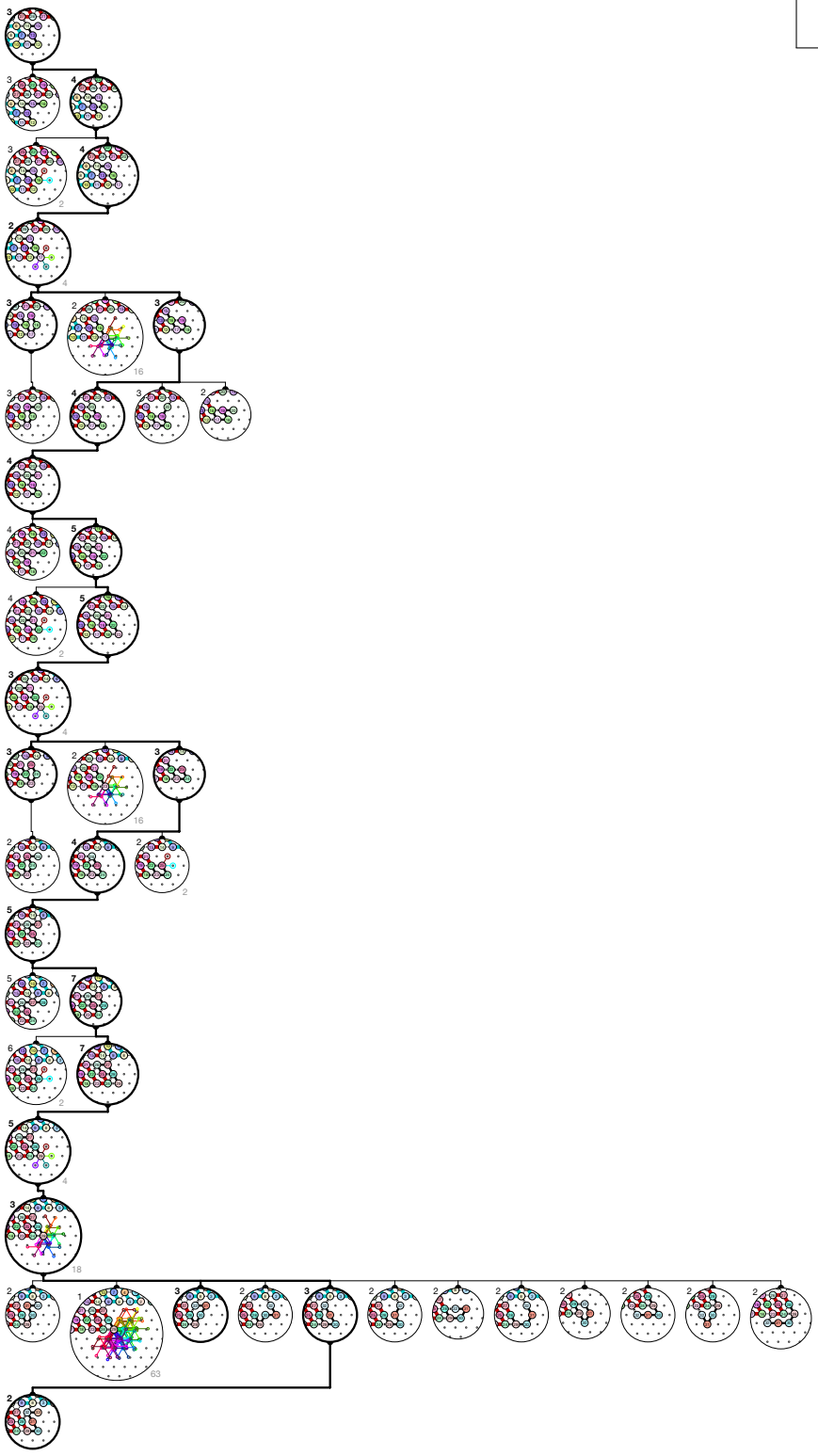
Output nascent configurations: Halt

Figure S.28. Folding of Brick Halt in 

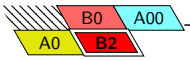
–First occurrence in 5-bits counter: Region #630.

Input nascent configurations: λ_3

ZOOM IN
FOR DETAILS



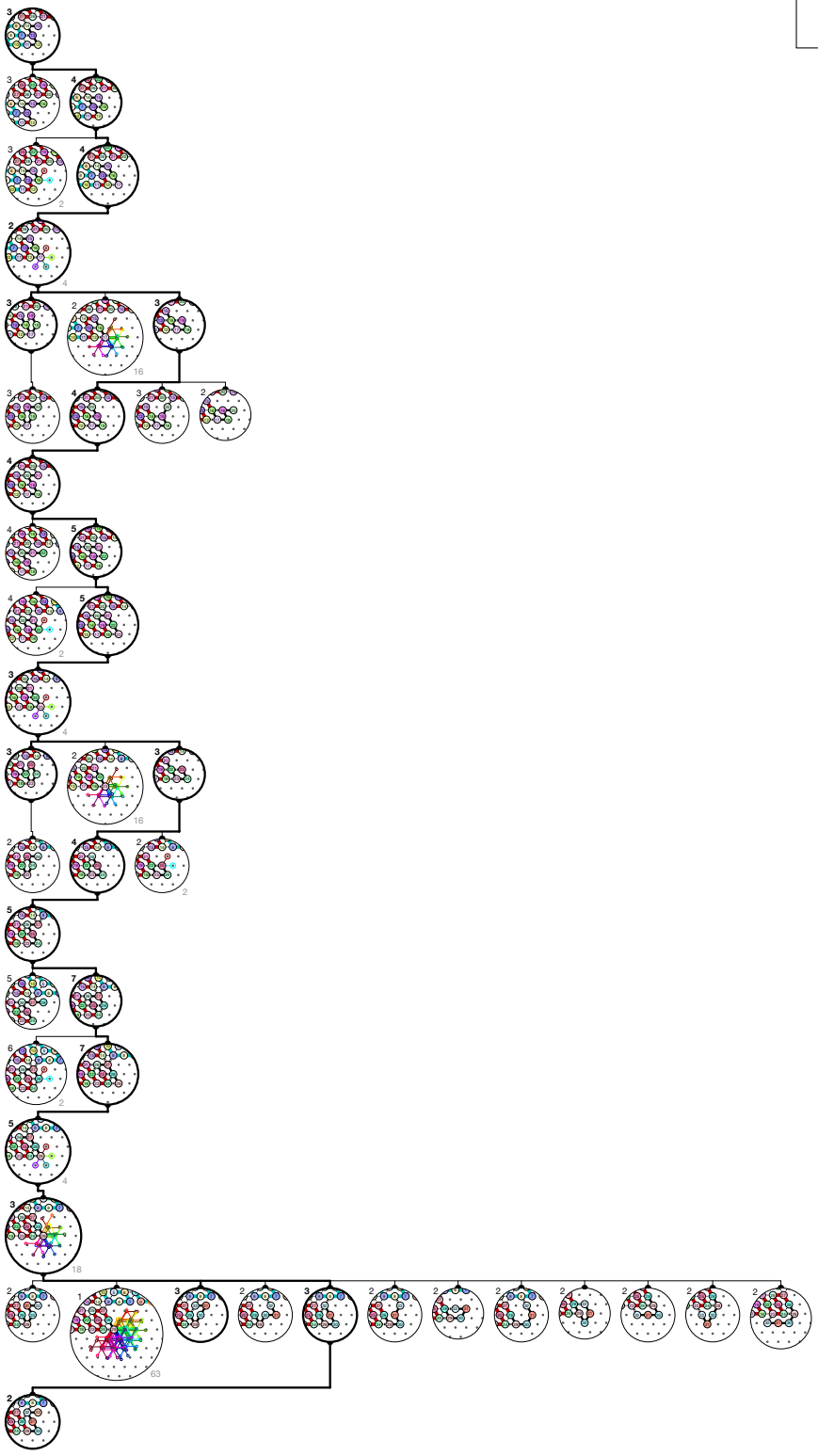
Output nascent configurations: λ_2'

Figure S.29. Folding of Brick B2 in 

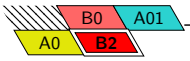
–First occurrence in 5-bits counter: Region #14.

Input nascent configurations: λ_3

ZOOM IN
FOR DETAILS



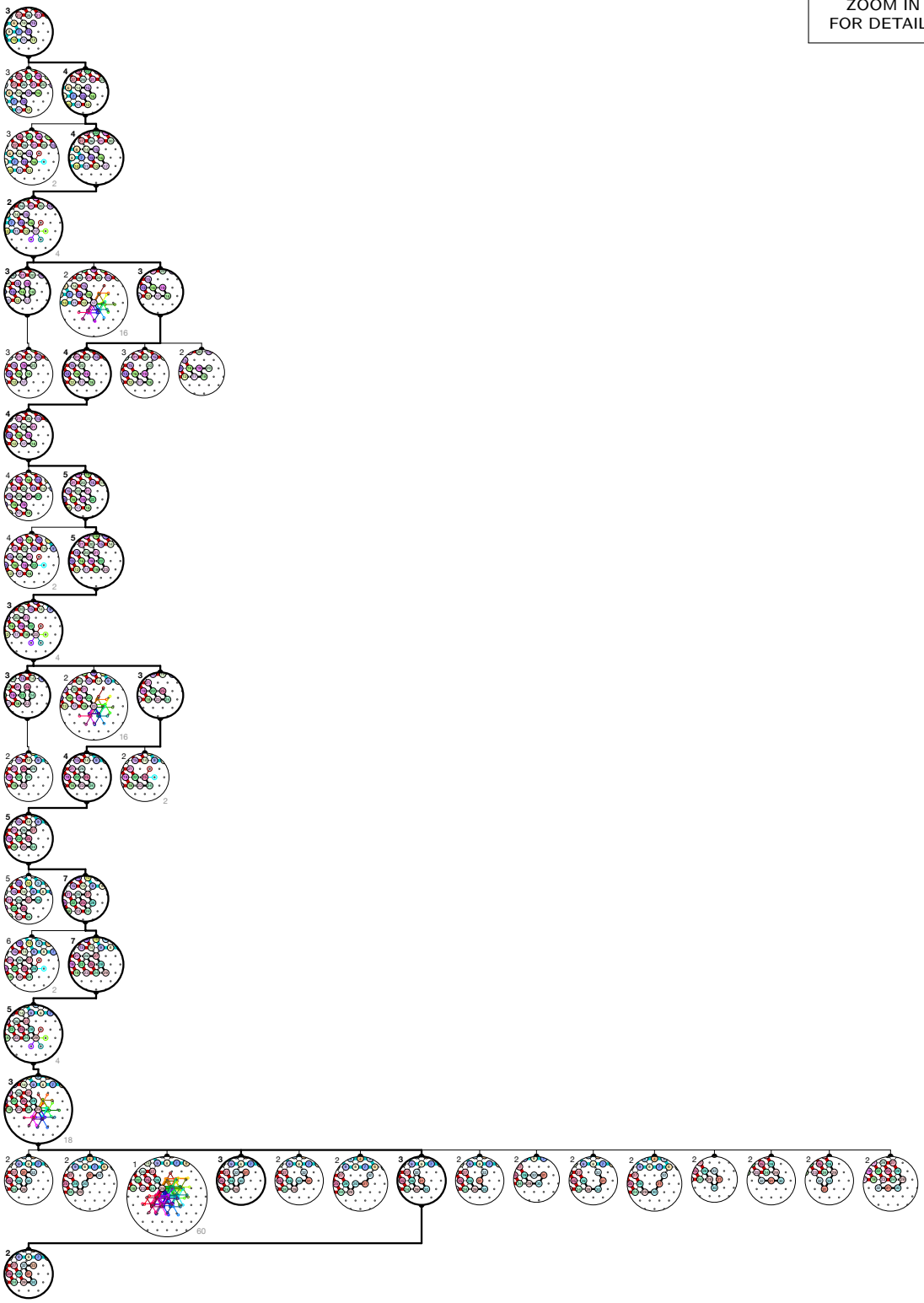
Output nascent configurations: λ_2'

Figure S.30. Folding of Brick B2 in 

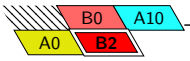
–First occurrence in 5-bits counter: Region #18.

Input nascent configurations: λ_3

ZOOM IN
FOR DETAILS



Output nascent configurations: λ_2'

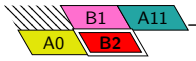
Figure S.31. Folding of Brick B2 in 

–First occurrence in 5-bits counter: Region #94.

Input nascent configurations: λ'_4

ZOOM IN
FOR DETAILS

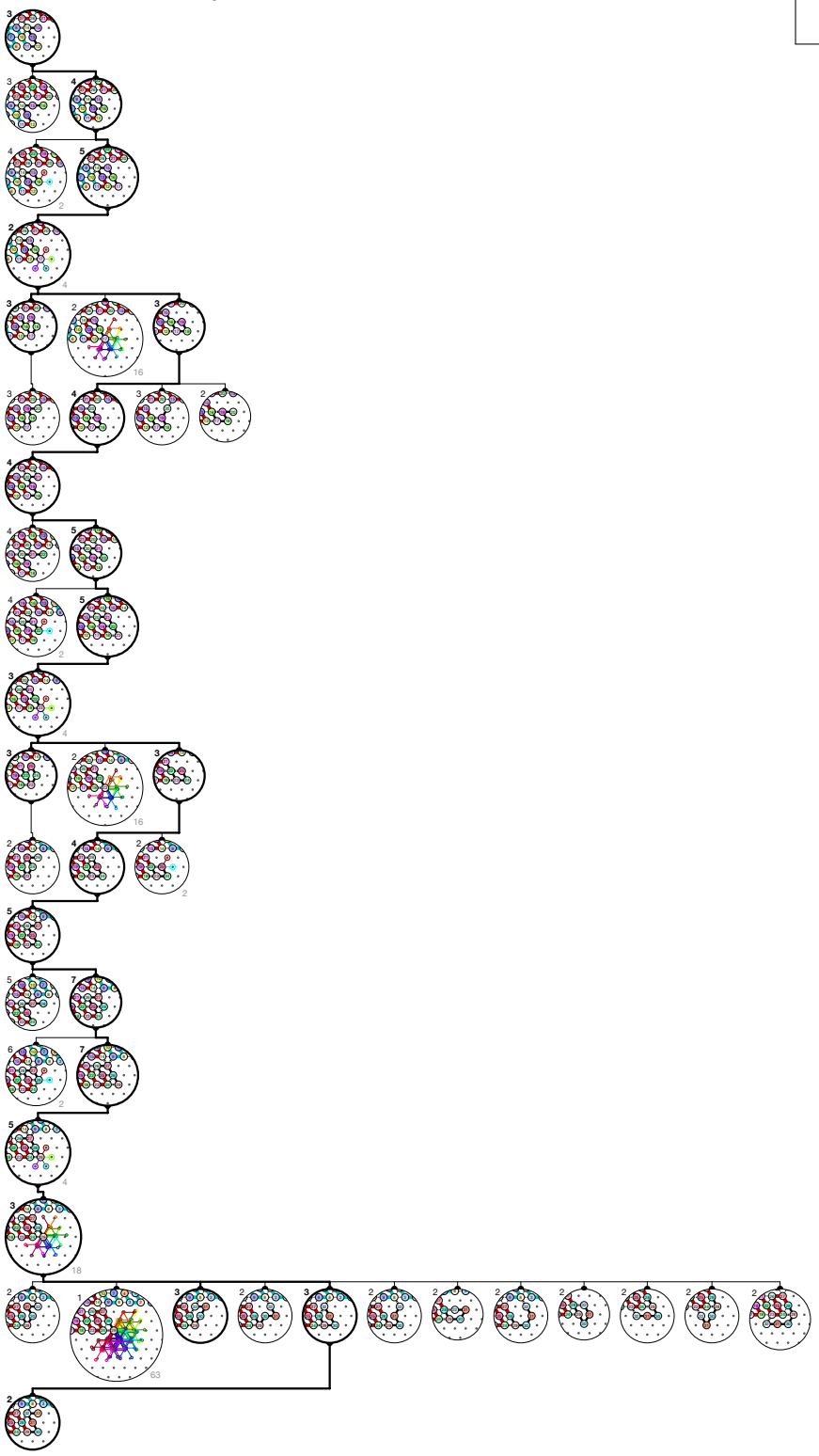
Output nascent configurations: $\mu \cup \mu'$

Figure S.32. Folding of Brick B2 in 

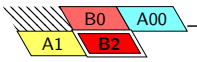
–First occurrence in 5-bits counter: Region #78.

Input nascent configurations: λ'_3

ZOOM IN
FOR DETAILS



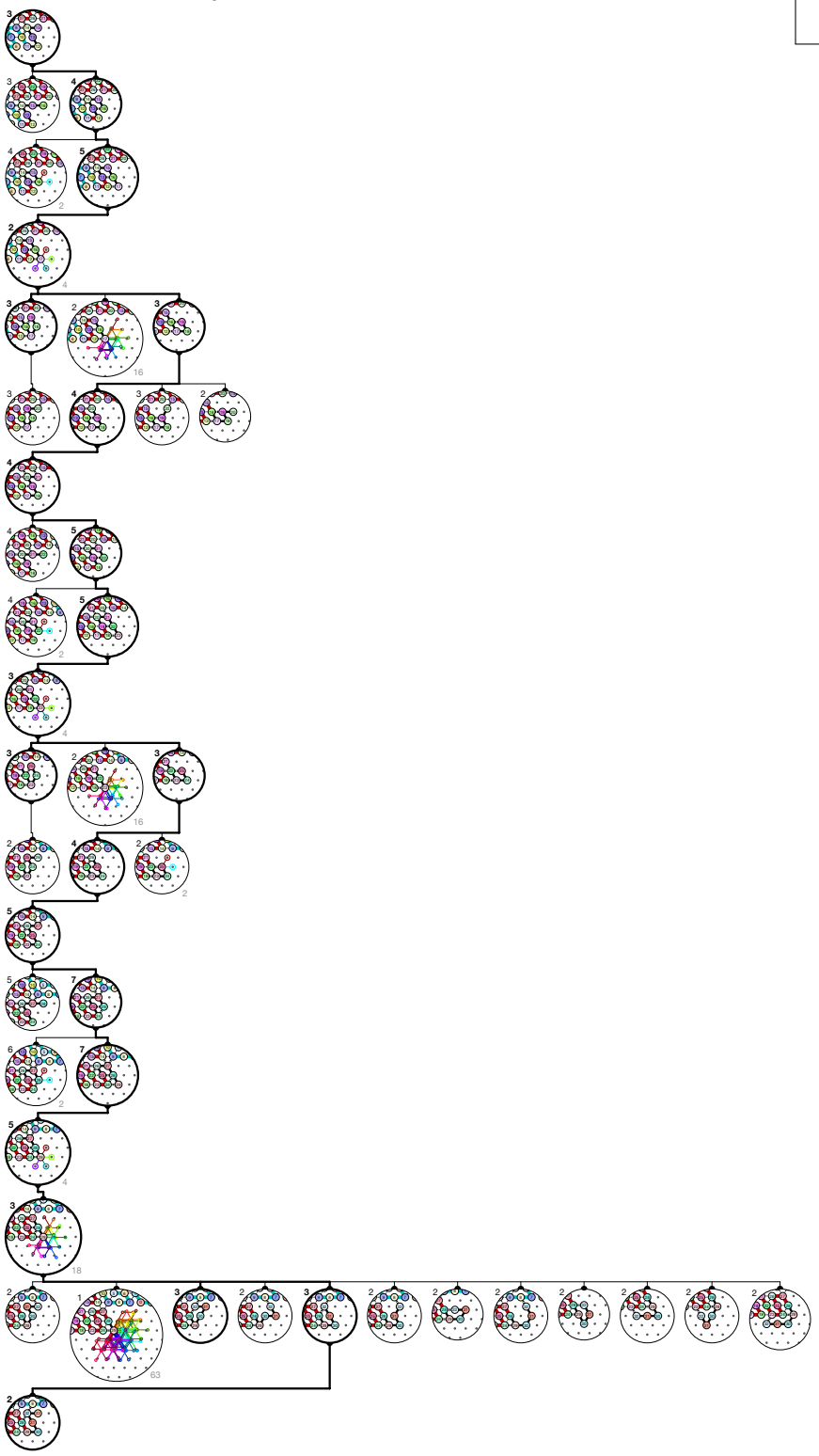
Output nascent configurations: λ'_2

Figure S.33. Folding of Brick B2 in 

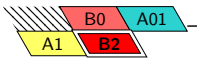
–First occurrence in 5-bits counter: Region #174.

Input nascent configurations: λ'_3

ZOOM IN
FOR DETAILS



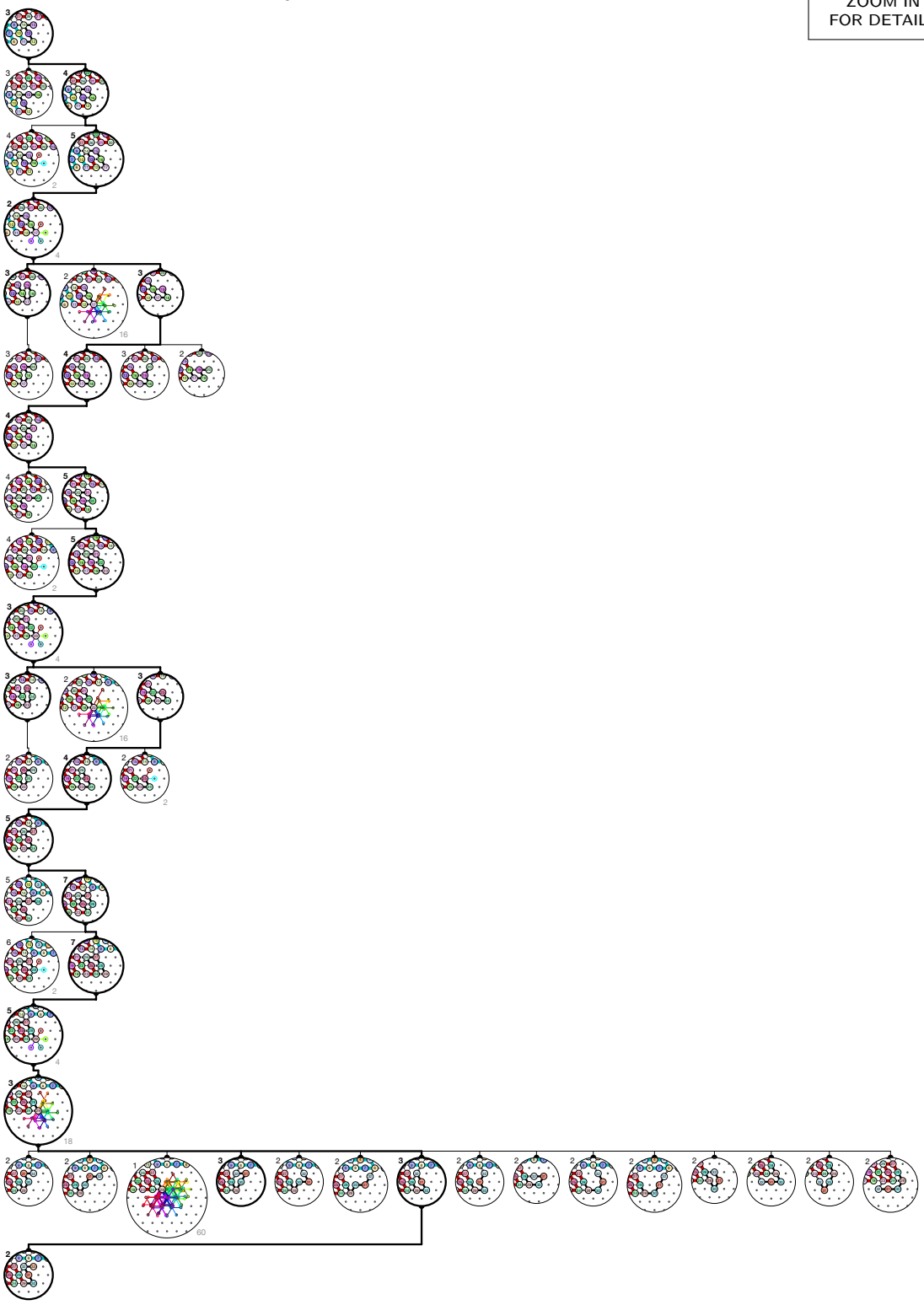
Output nascent configurations: λ'_2

Figure S.34. Folding of Brick B2 in 

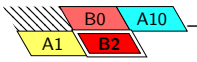
–First occurrence in 5-bits counter: Region #138.

Input nascent configurations: λ'_3

ZOOM IN
FOR DETAILS



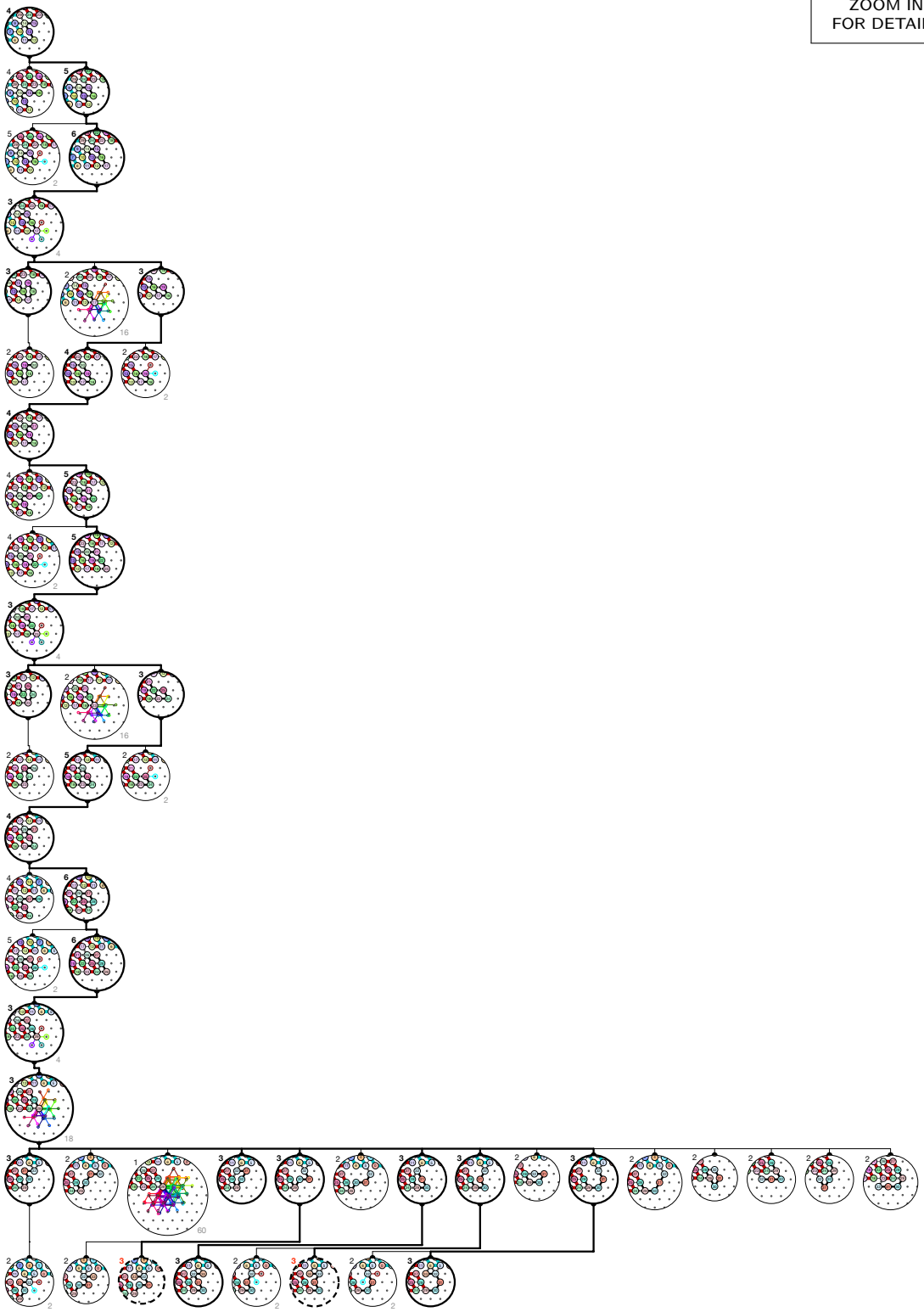
Output nascent configurations: λ'_2

Figure S.35. Folding of Brick B2 in 

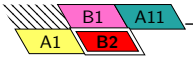
–First occurrence in 5-bits counter: Region #254.

Input nascent configurations: λ_4

ZOOM IN
FOR DETAILS



Output nascent configurations: $\mu \cup \mu'$

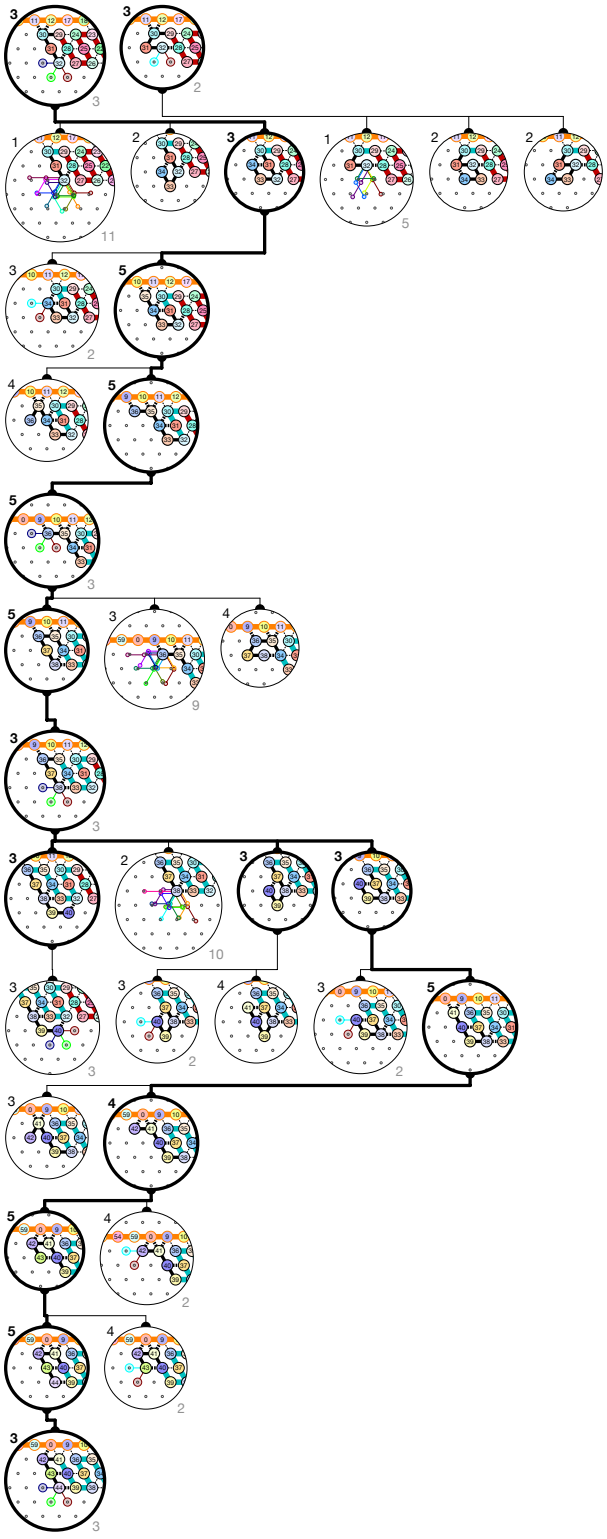
Figure S.36. Folding of Brick B2 in 

–First occurrence in 5-bits counter: Region #38.

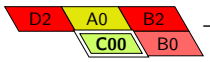
S.2.4. Folding certificates for Module C: Second Half-Adder

Input nascent configurations: $\theta \cup \theta'$

ZOOM IN
FOR DETAILS



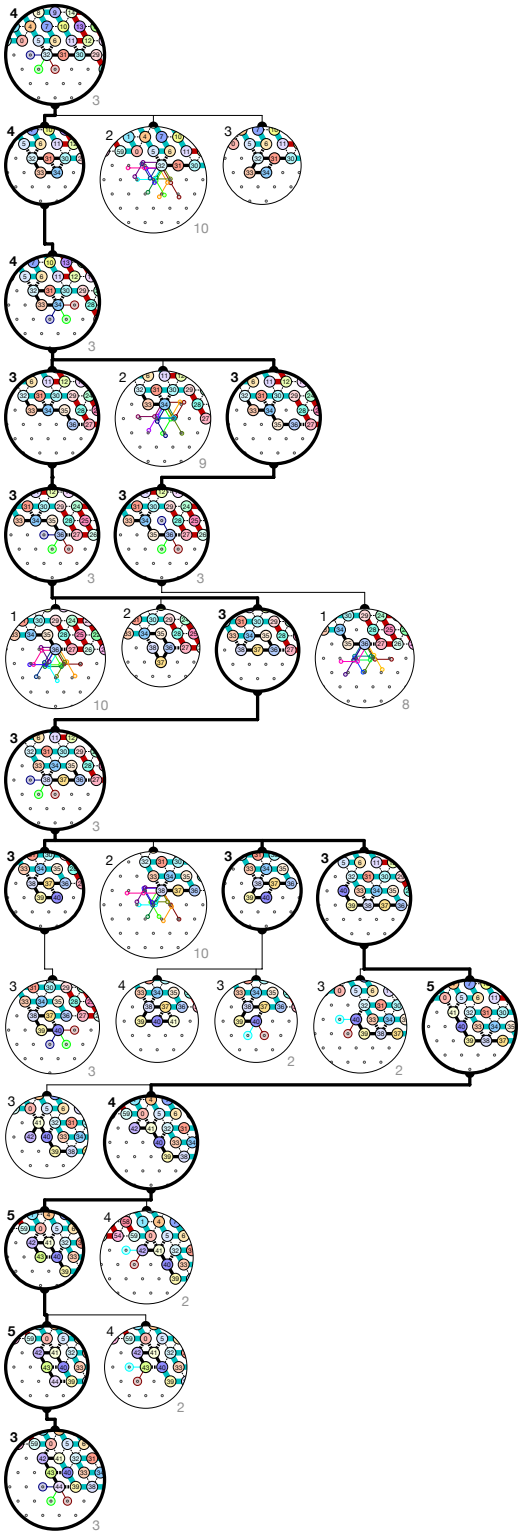
Output nascent configurations: β'

Figure S.37. Folding of Brick C00 in 

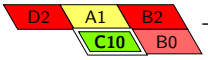
–First occurrence in 5-bits counter: Region #3.

Input nascent configurations: γ'

ZOOM IN
FOR DETAILS



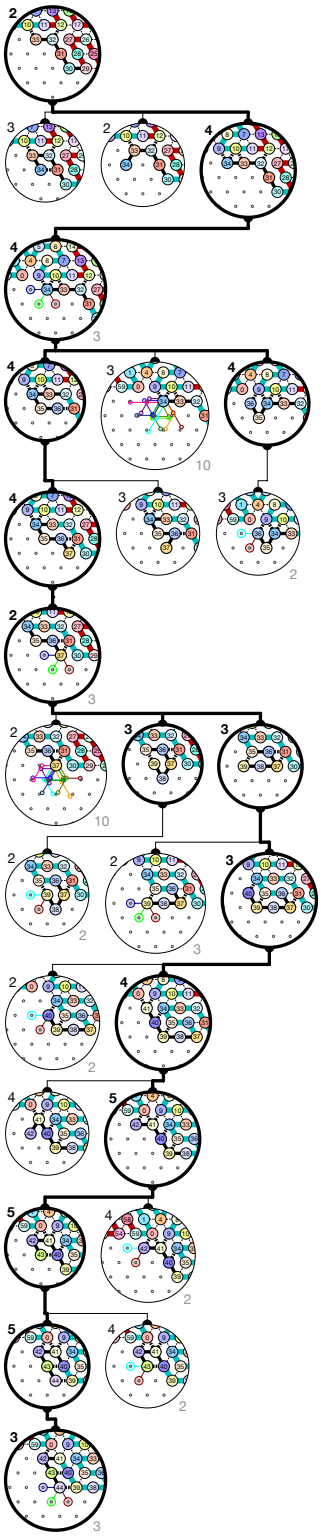
Output nascent configurations: β'

Figure S.38. Folding of Brick C10 in 

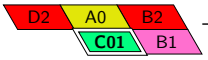
–First occurrence in 5-bits counter: Region #43.

Input nascent configurations: α''

ZOOM IN
FOR DETAILS



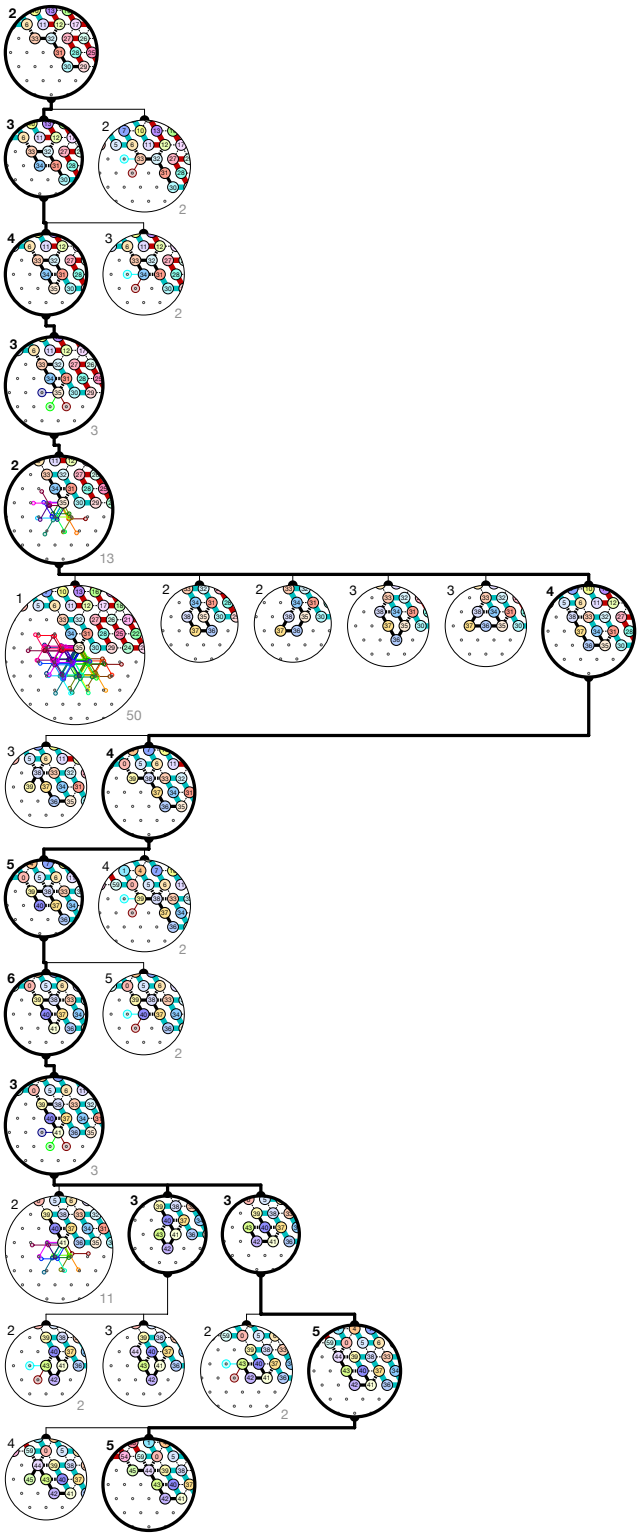
Output nascent configurations: β'

Figure S.39. Folding of Brick C01 in 

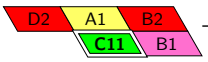
–First occurrence in 5-bits counter: Region #23.

Input nascent configurations: α''

ZOOM IN
FOR DETAILS



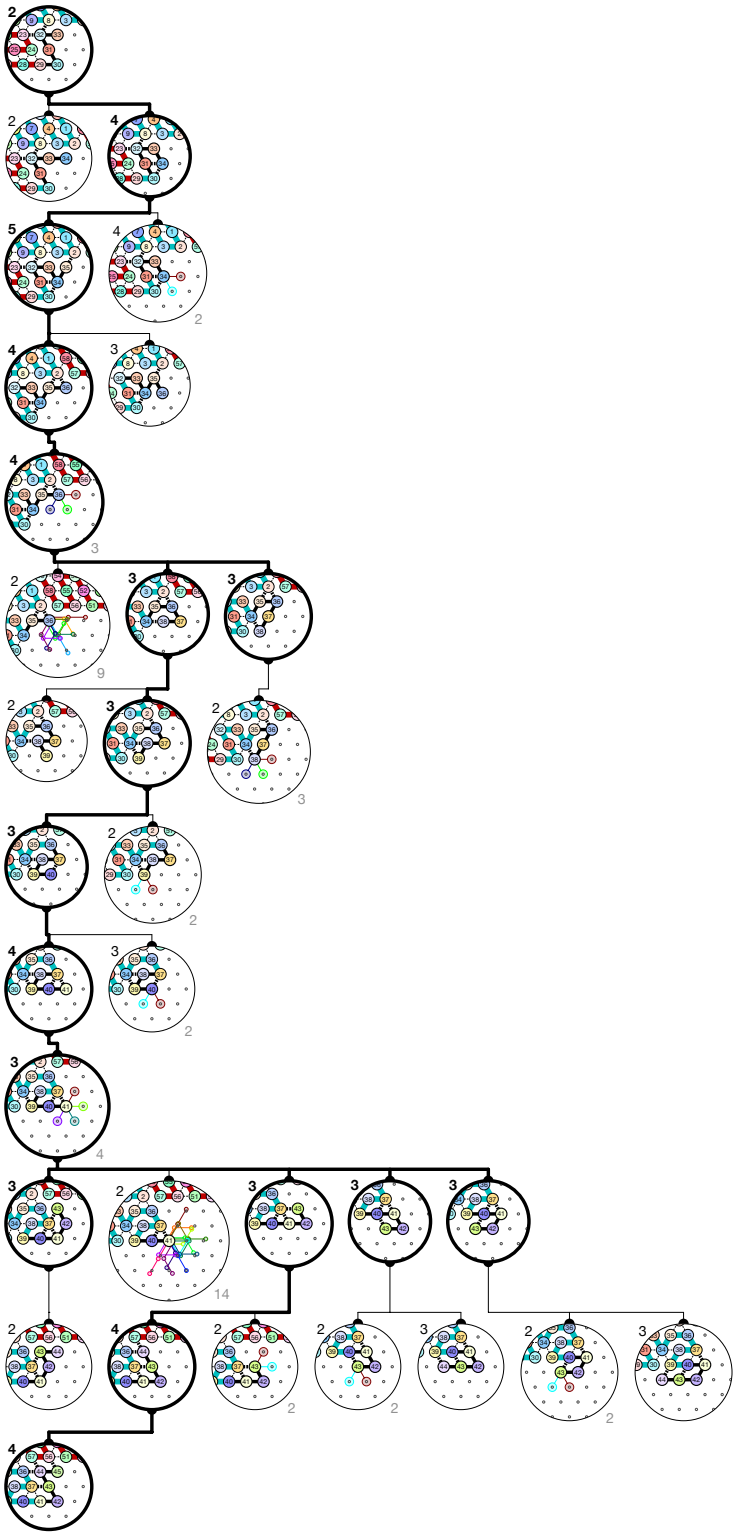
Output nascent configurations: α'''

Figure S.40. Folding of Brick C11 in 

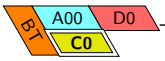
–First occurrence in 5-bits counter: Region #63.

Input nascent configurations: λ_2

ZOOM IN
FOR DETAILS



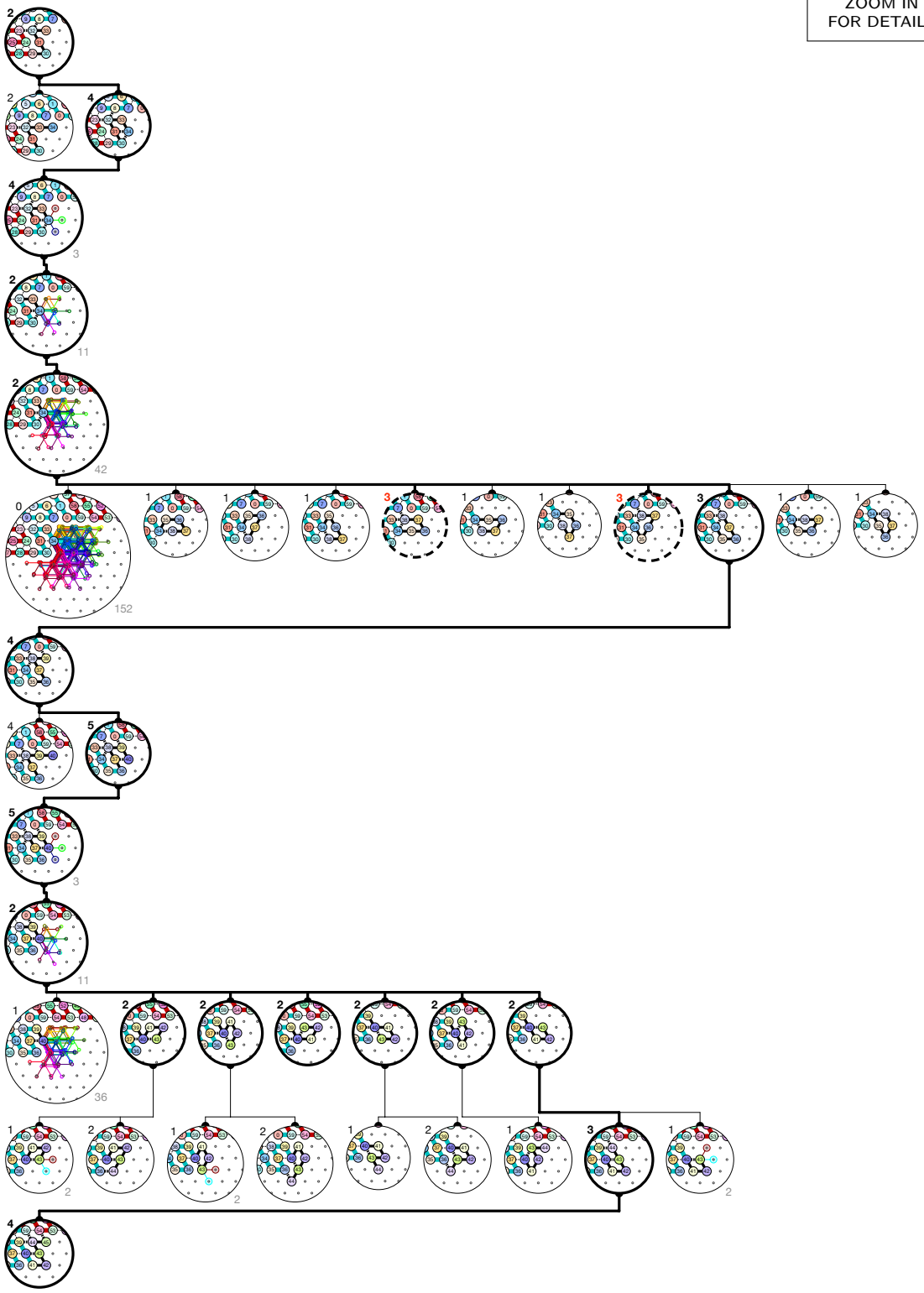
Output nascent configurations: λ'_4

Figure S.41. Folding of Brick C0 in 

–First occurrence in 5-bits counter: Region #11.

Input nascent configurations: λ_2

ZOOM IN
FOR DETAILS



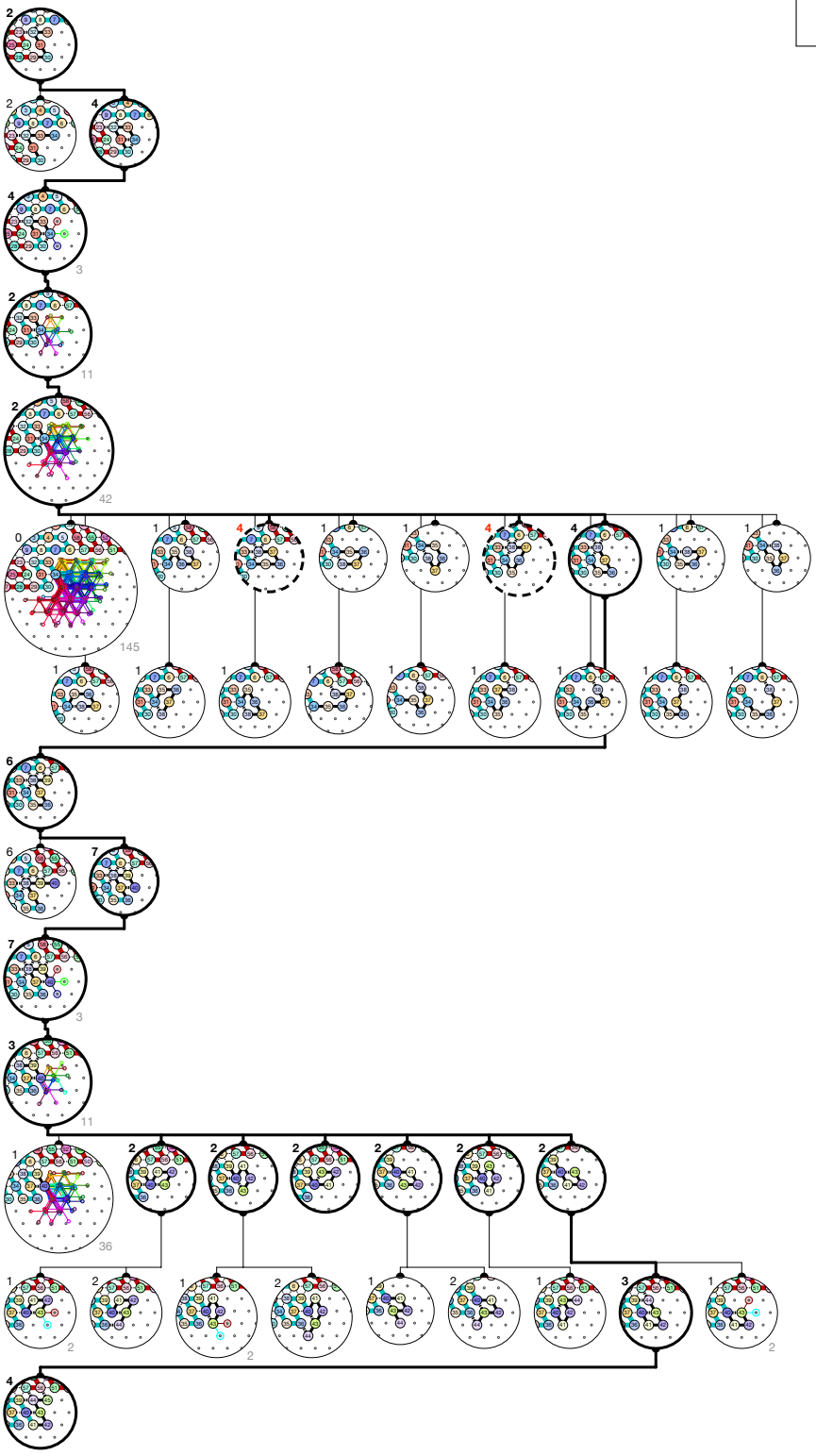
Output nascent configurations: λ'_4

Figure S.42. Folding of Brick C1 in 

–First occurrence in 5-bits counter: Region #311.

Input nascent configurations: λ_2

ZOOM IN
FOR DETAILS



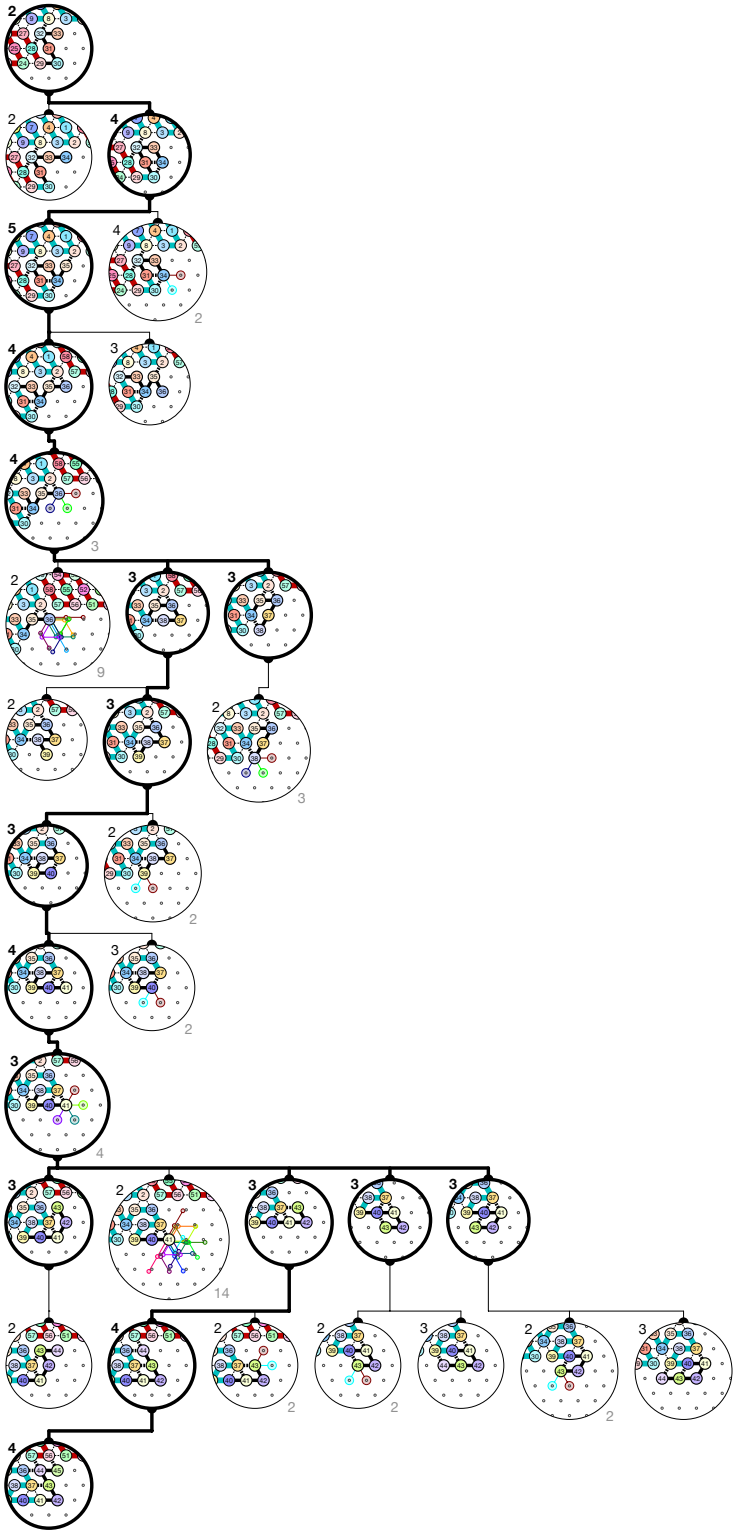
Output nascent configurations: λ'_4

Figure S.43. Folding of Brick C1 in 

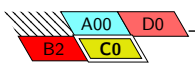
–First occurrence in 5-bits counter: Region #331.

Input nascent configurations: λ'_2

ZOOM IN
FOR DETAILS



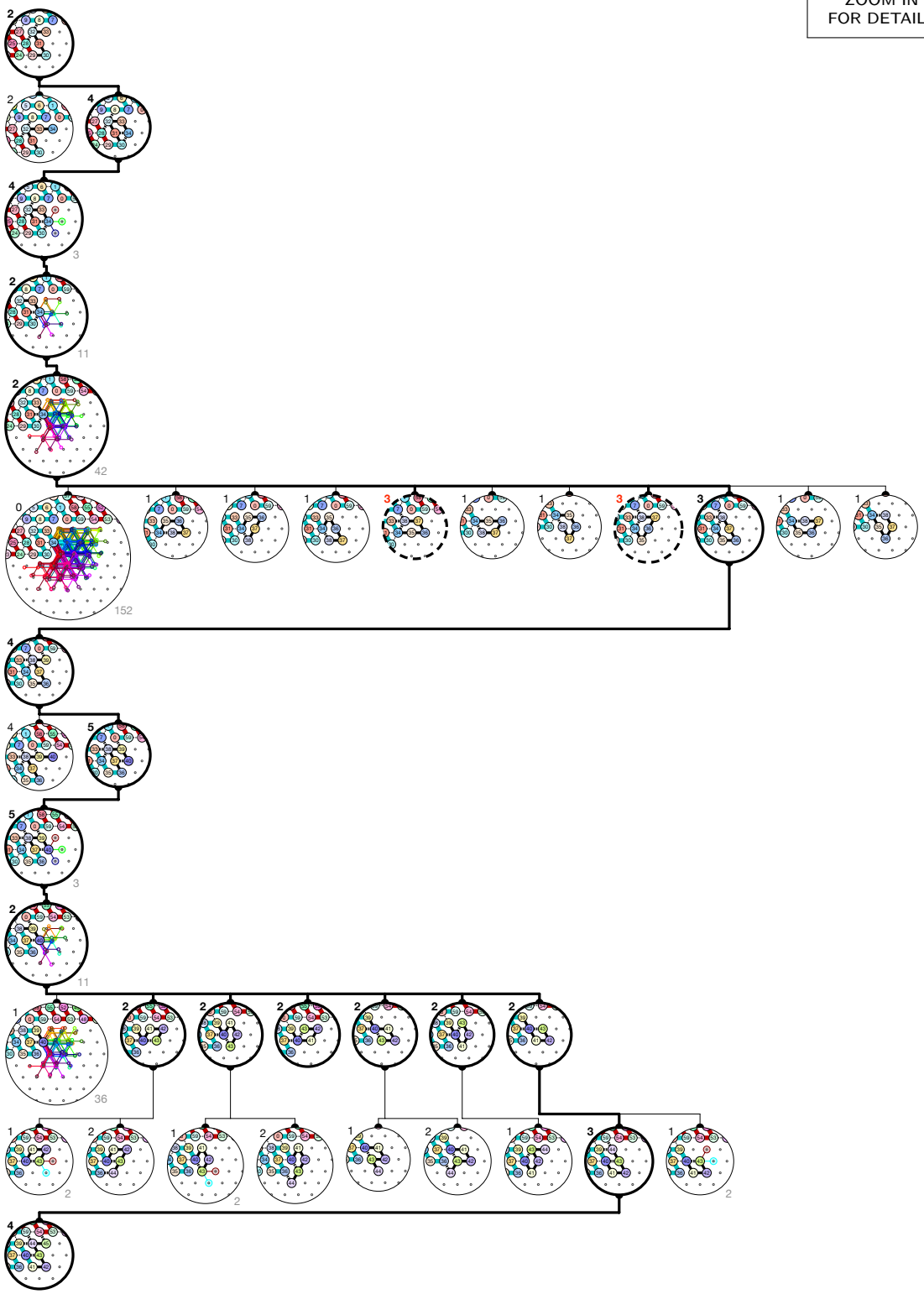
Output nascent configurations: λ'_4

Figure S.44. Folding of Brick C0 in 

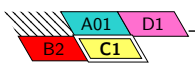
–First occurrence in 5-bits counter: Region #15.

Input nascent configurations: λ'_2

ZOOM IN
FOR DETAILS



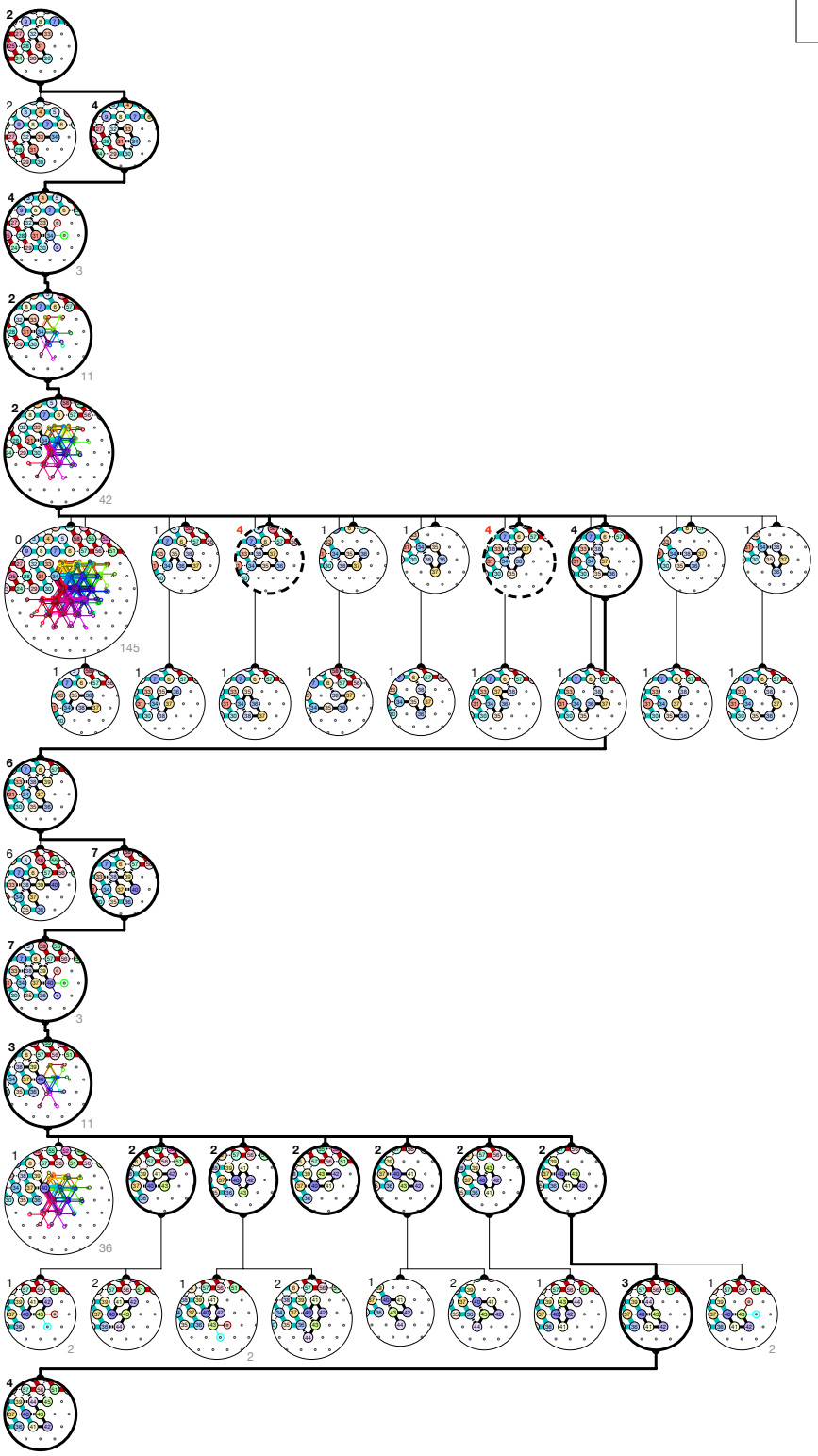
Output nascent configurations: λ'_4

Figure S.45. Folding of Brick C1 in 

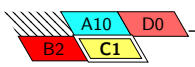
–First occurrence in 5-bits counter: Region #75.

Input nascent configurations: λ'_2

ZOOM IN
FOR DETAILS



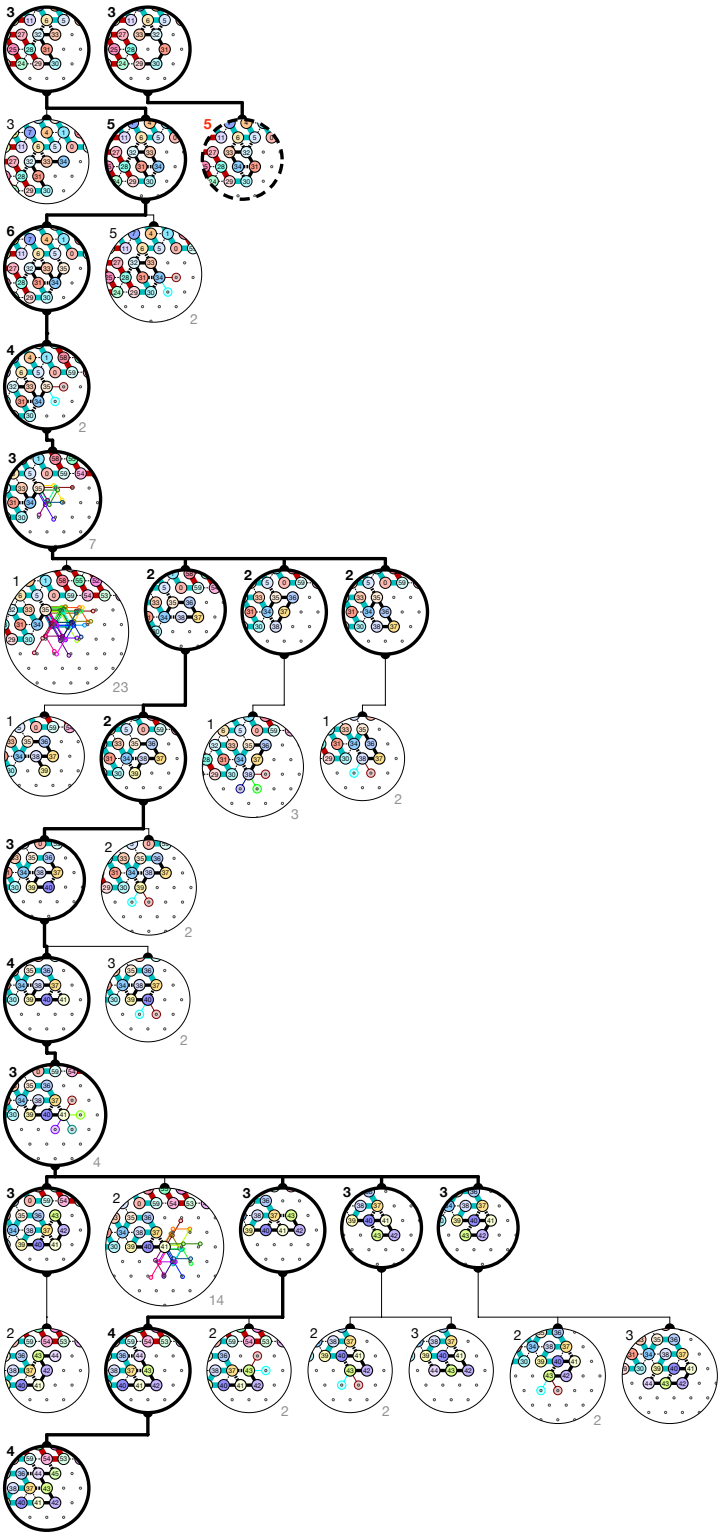
Output nascent configurations: λ'_4

Figure S.46. Folding of Brick C1 in 

–First occurrence in 5-bits counter: Region #95.

Input nascent configurations: $\mu \cup \mu'$

ZOOM IN
FOR DETAILS



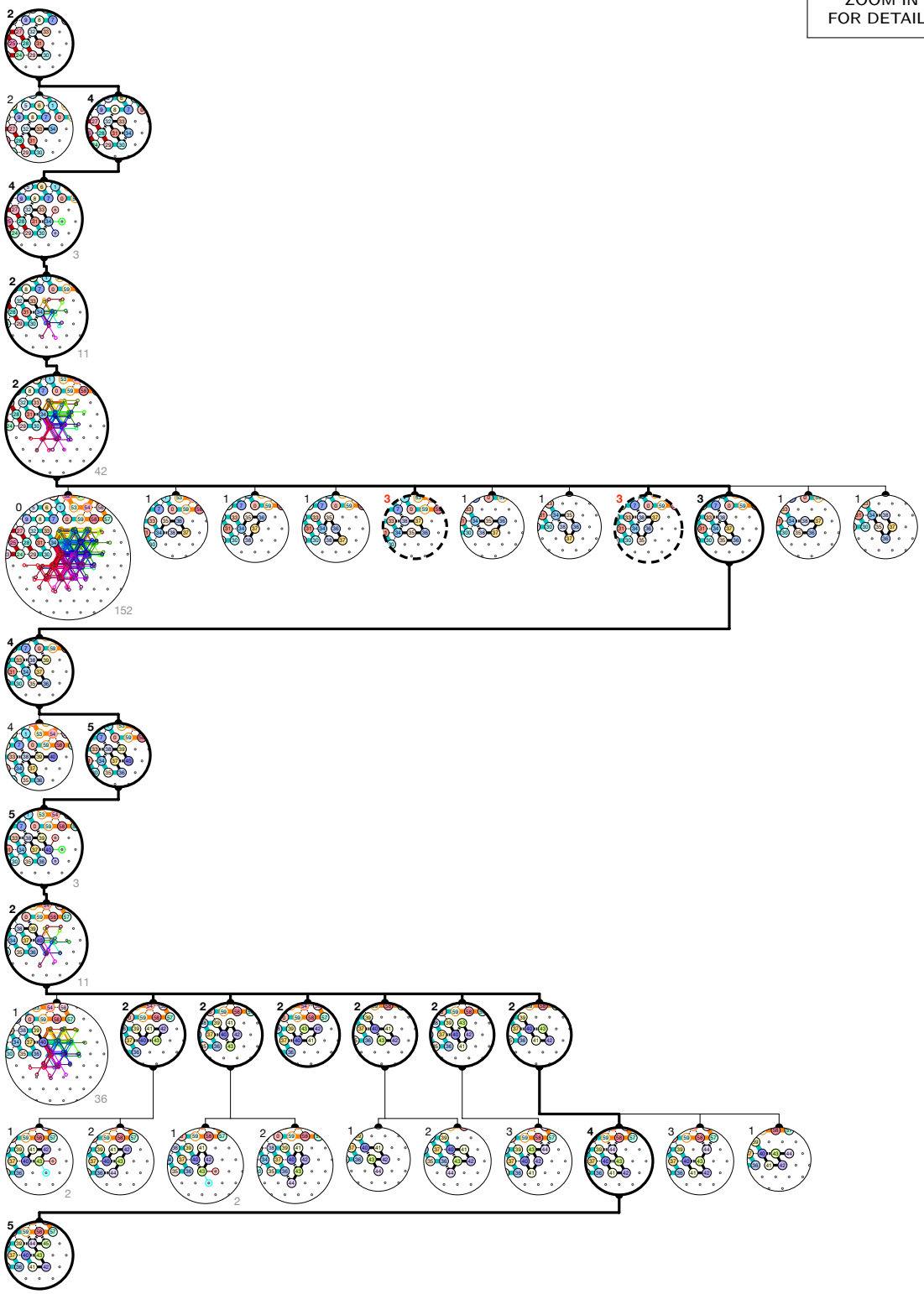
Output nascent configurations: λ'_4

Figure S.47. Folding of Brick C0 in 

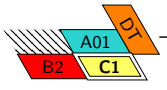
–First occurrence in 5-bits counter: Region #155.

Input nascent configurations: λ'_2

ZOOM IN
FOR DETAILS



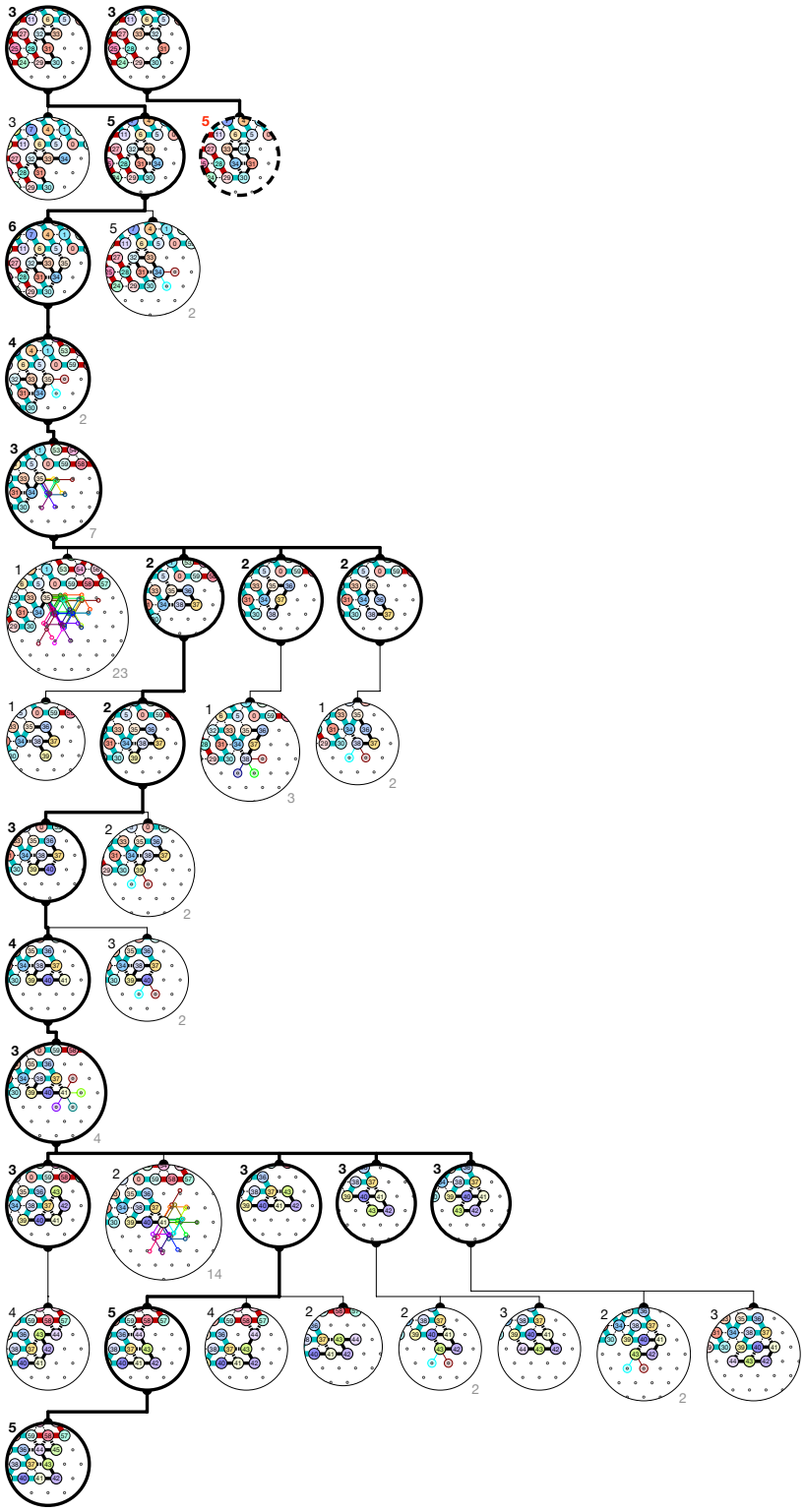
Output nascent configurations: λ_5

Figure S.48. Folding of Brick C1 in 

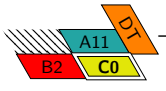
–First occurrence in 5-bits counter: Region #19.

Input nascent configurations: $\mu \cup \mu'$

ZOOM IN
FOR DETAILS



Output nascent configurations: λ_5

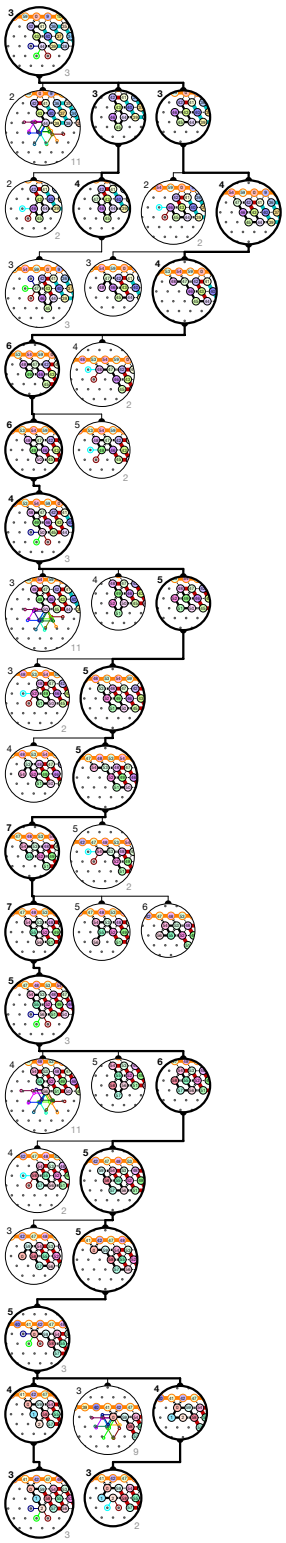
Figure S.49. Folding of Brick C0 in 

–First occurrence in 5-bits counter: Region #39.

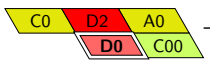
S.2.5. Folding Certificates for Module D: Right Turn

Input nascent configurations: β'

ZOOM IN
FOR DETAILS



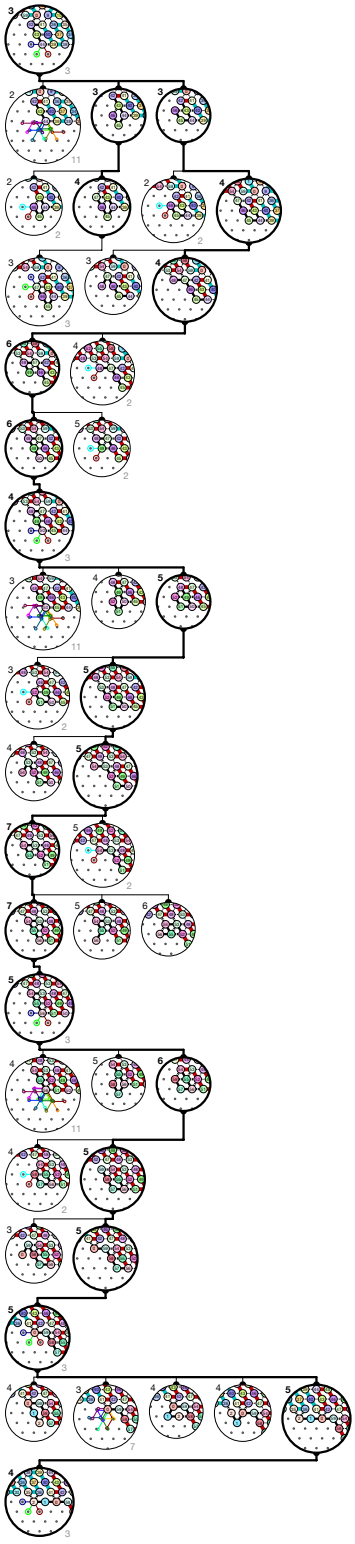
Output nascent configurations: $\theta \cup \theta'$

Figure S.50. Folding of Brick D0 in 

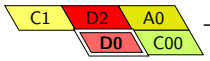
–First occurrence in 5-bits counter: Region #4.

Input nascent configurations: β'

ZOOM IN
FOR DETAILS



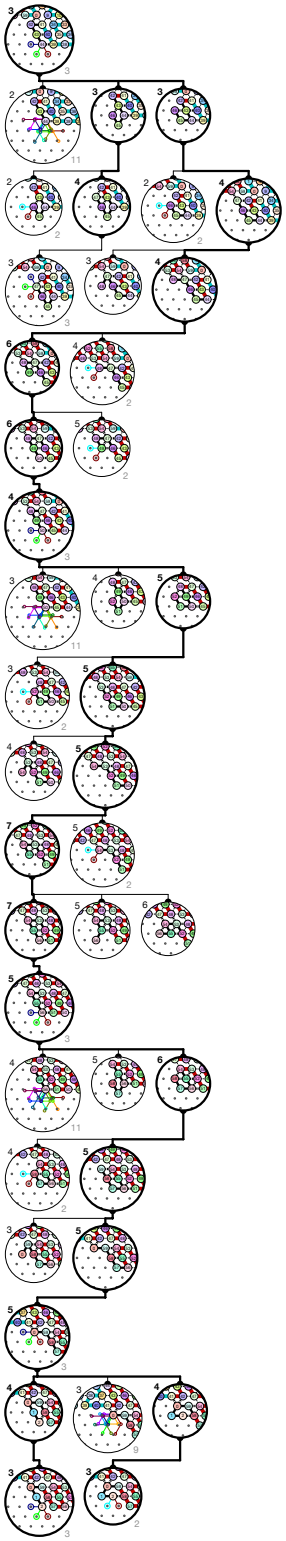
Output nascent configurations: γ'

Figure S.51. Folding of Brick D0 in 

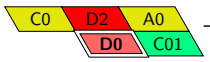
–First occurrence in 5-bits counter: Region #84.

Input nascent configurations: β'

ZOOM IN
FOR DETAILS



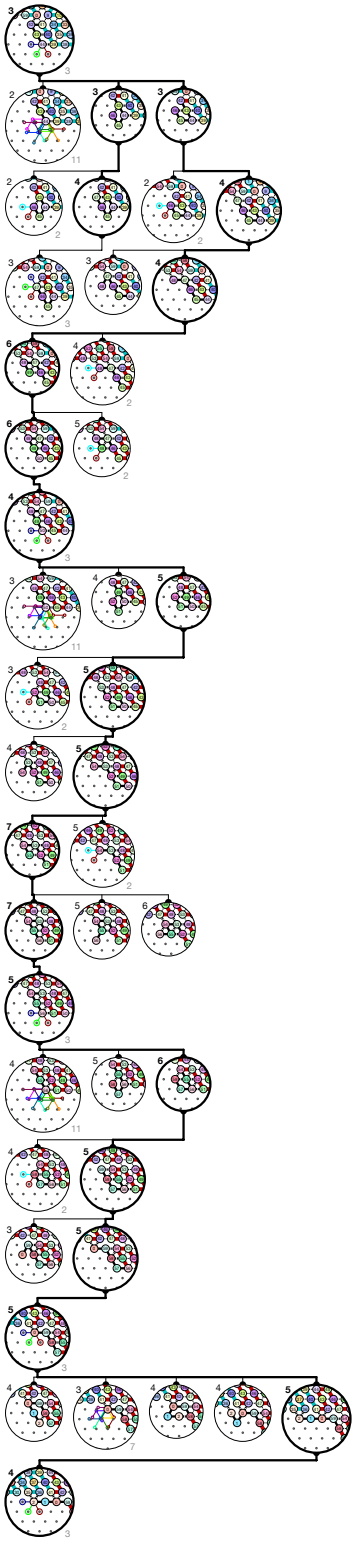
Output nascent configurations: $\theta \cup \theta'$

Figure S.52. Folding of Brick D0 in 

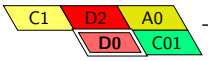
–First occurrence in 5-bits counter: Region #24.

Input nascent configurations: β'

ZOOM IN
FOR DETAILS



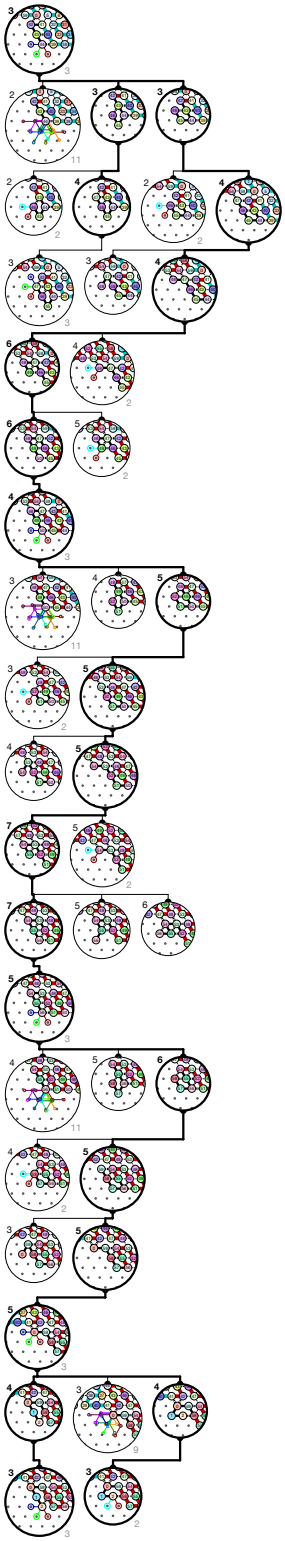
Output nascent configurations: γ'

Figure S.53. Folding of Brick D0 in 

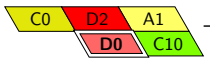
–First occurrence in 5-bits counter: Region #104.

Input nascent configurations: β'

ZOOM IN
FOR DETAILS



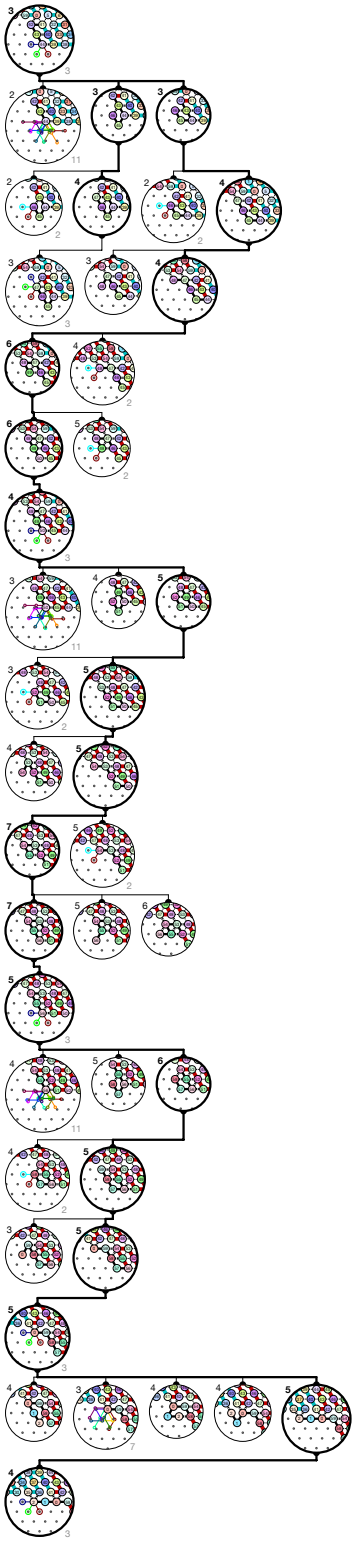
Output nascent configurations: $\theta \cup \theta'$

Figure S.54. Folding of Brick D0 in 

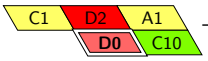
–First occurrence in 5-bits counter: Region #44.

Input nascent configurations: β'

ZOOM IN
FOR DETAILS



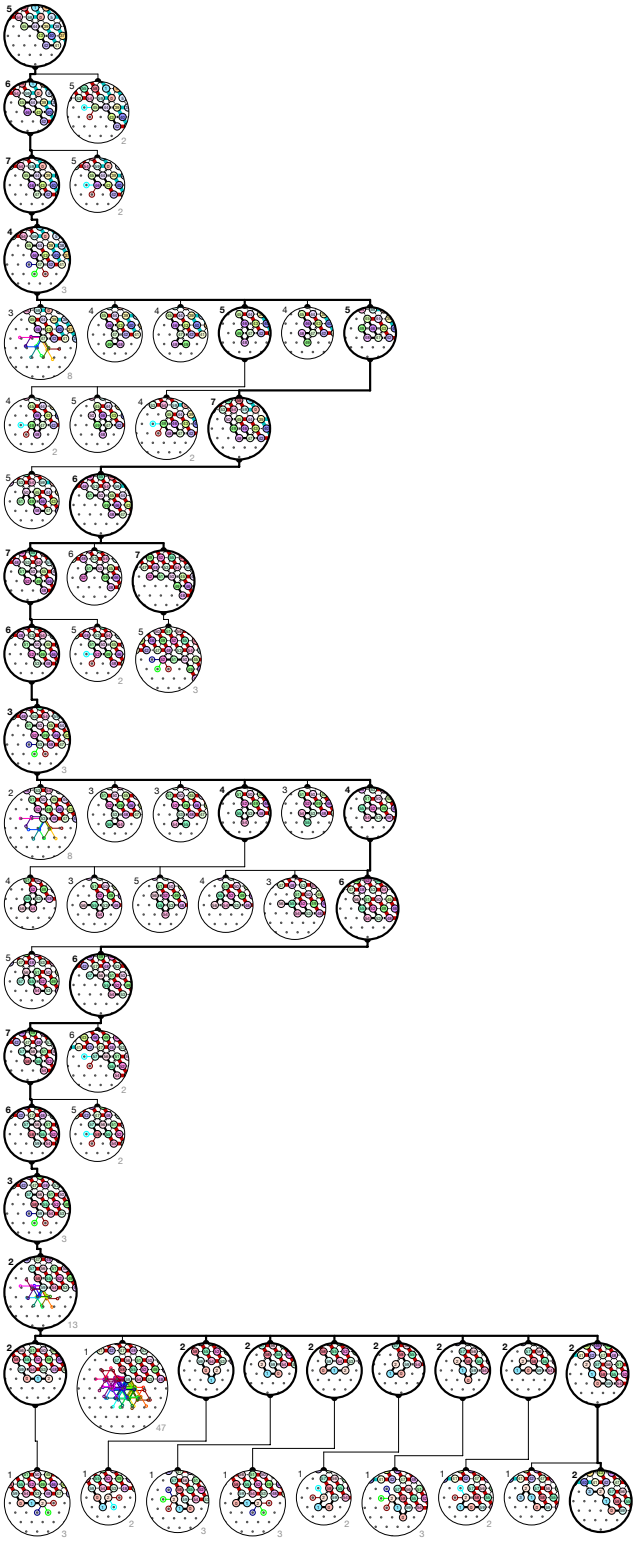
Output nascent configurations: γ'

Figure S.55. Folding of Brick D0 in 

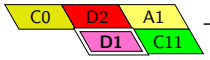
–First occurrence in 5-bits counter: Region #124.

Input nascent configurations: α'''

ZOOM IN
FOR DETAILS



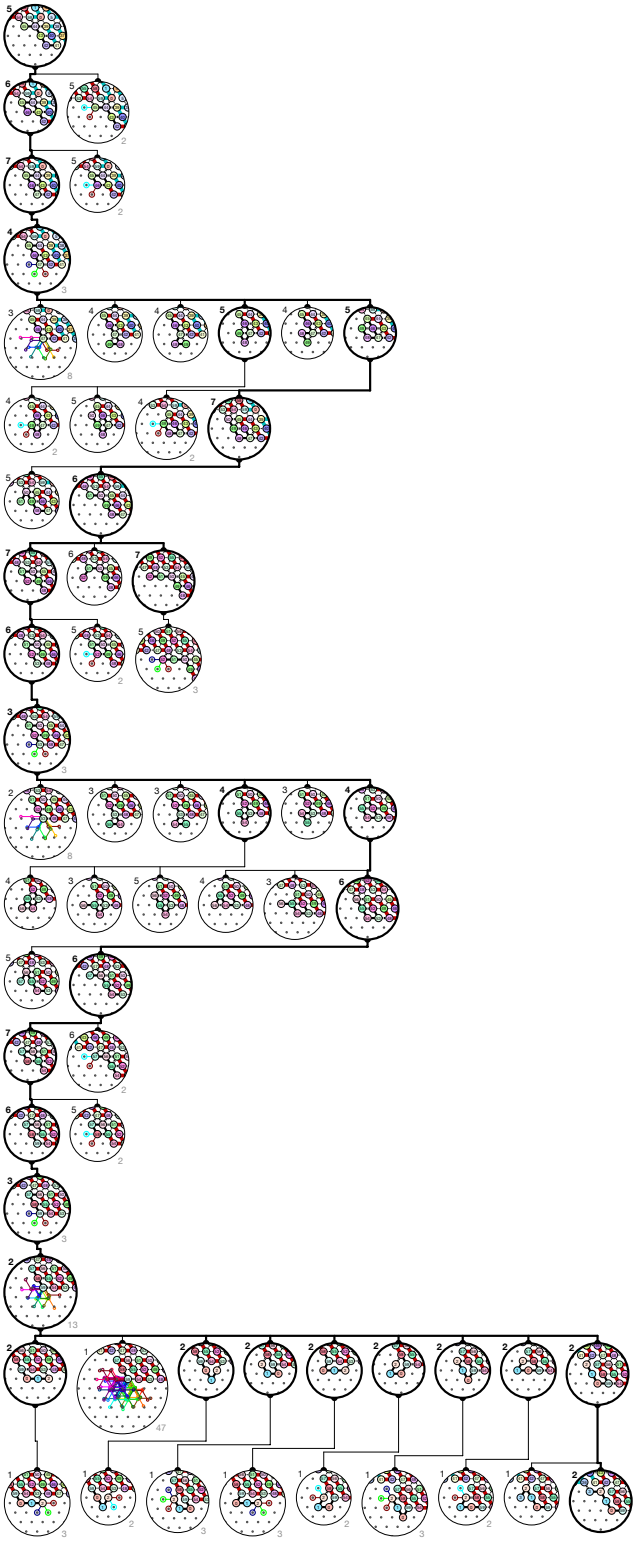
Output nascent configurations: α''

Figure S.56. Folding of Brick D1 in 

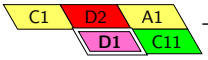
–First occurrence in 5-bits counter: Region #64.

Input nascent configurations: α'''

ZOOM IN
FOR DETAILS



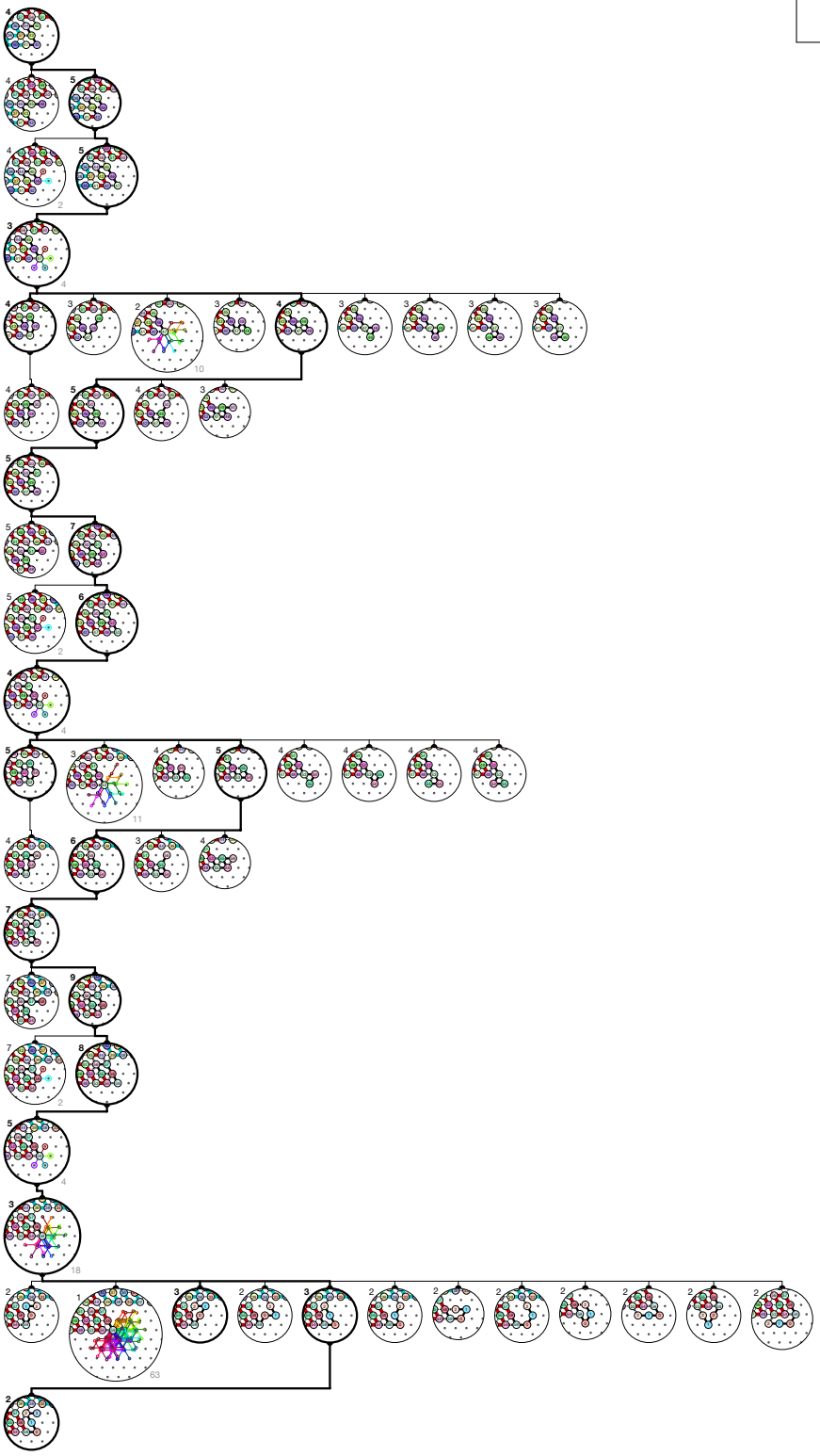
Output nascent configurations: α''

Figure S.57. Folding of Brick D1 in 

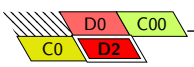
–First occurrence in 5-bits counter: Region #144.

Input nascent configurations: λ'_4

ZOOM IN
FOR DETAILS



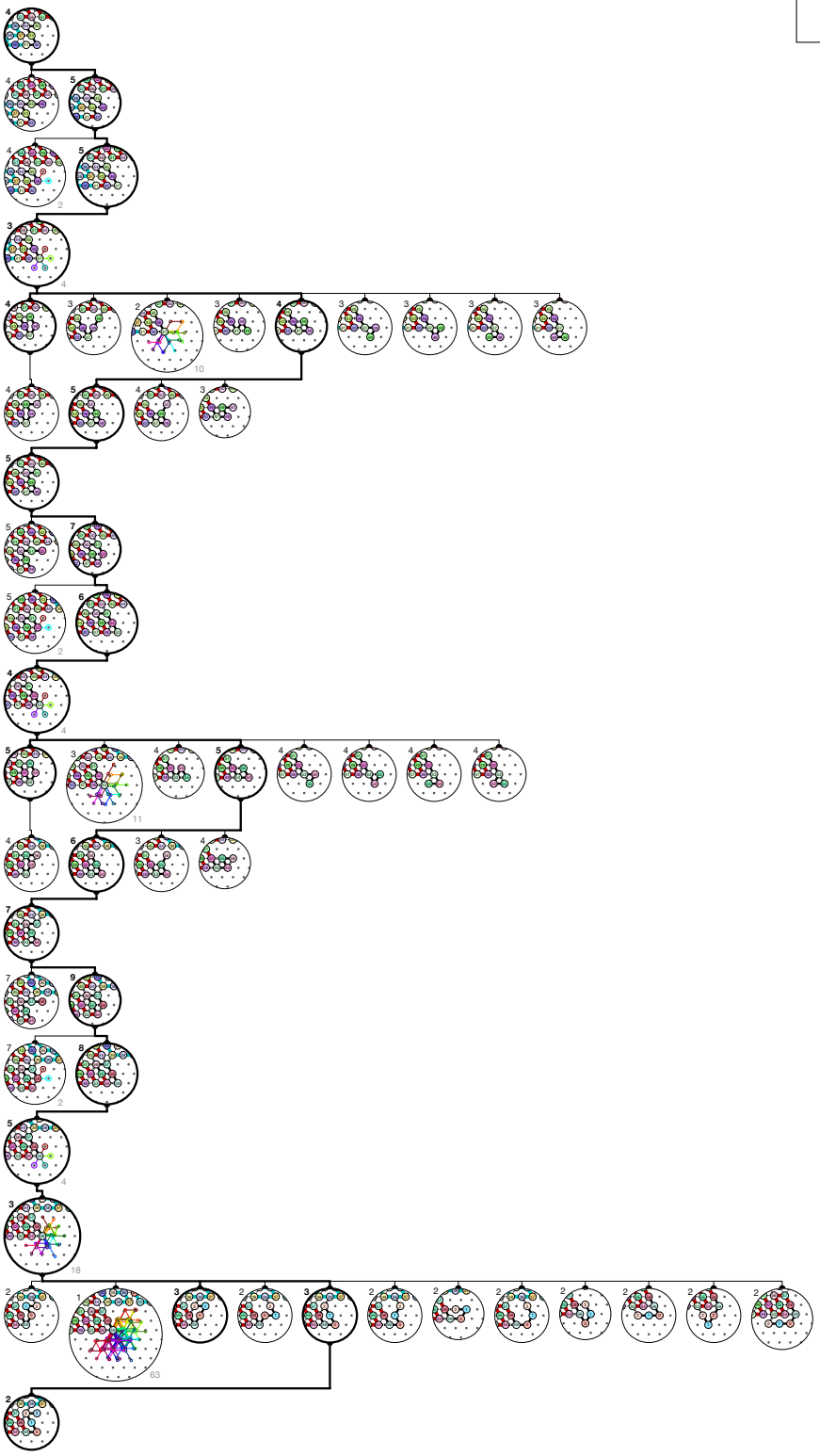
Output nascent configurations: λ'_2

Figure S.58. Folding of Brick D2 in 

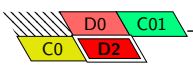
–First occurrence in 5-bits counter: Region #12.

Input nascent configurations: λ'_4

ZOOM IN
FOR DETAILS



Output nascent configurations: λ'_2

Figure S.59. Folding of Brick D2 in 

–First occurrence in 5-bits counter: Region #36.

Input nascent configurations: λ'_4

ZOOM IN
FOR DETAILS

Output nascent configurations: λ'_2

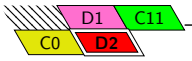
Figure S.60. Folding of Brick D2 in 

–First occurrence in 5-bits counter: Region #56.

Input nascent configurations: λ'_4

ZOOM IN
FOR DETAILS

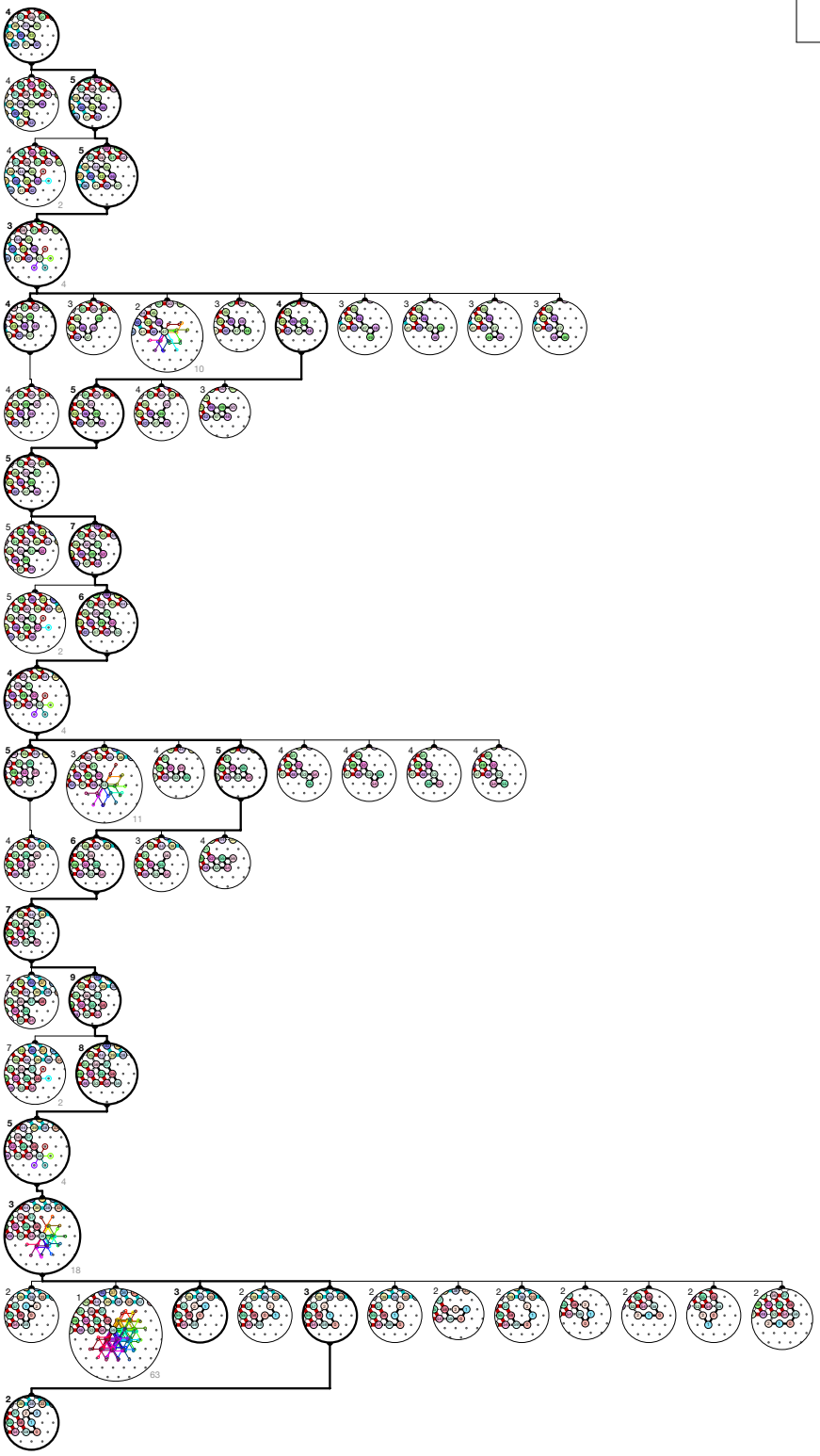
Output nascent configurations: $\mu \cup \mu'$

Figure S.61. Folding of Brick D2 in 

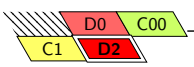
–First occurrence in 5-bits counter: Region #156.

Input nascent configurations: λ'_4

ZOOM IN
FOR DETAILS



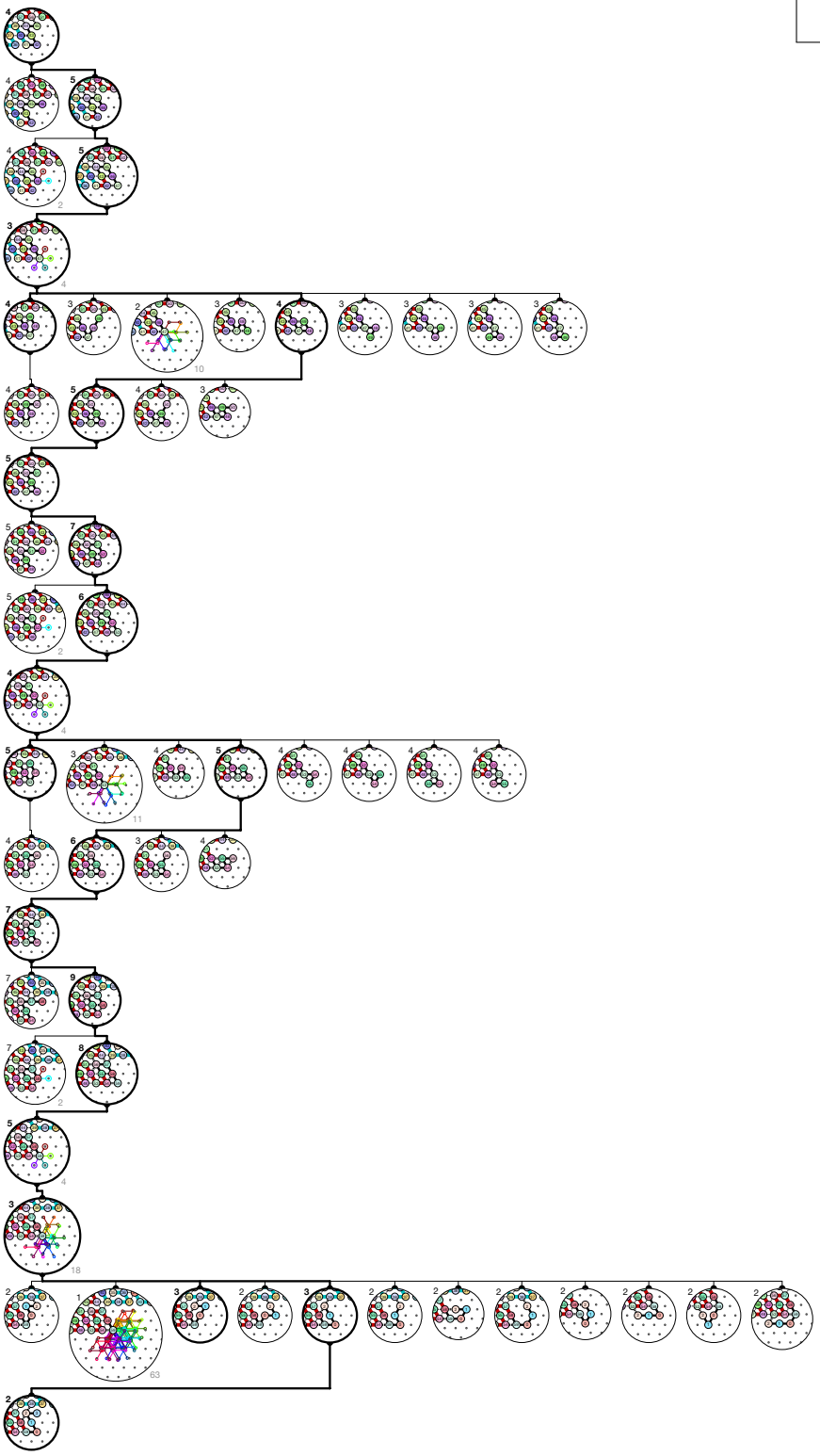
Output nascent configurations: λ'_2

Figure S.62. Folding of Brick D2 in 

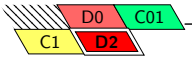
–First occurrence in 5-bits counter: Region #96.

Input nascent configurations: λ'_4

ZOOM IN
FOR DETAILS



Output nascent configurations: λ'_2

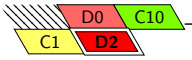
Figure S.63. Folding of Brick D2 in 

–First occurrence in 5-bits counter: Region #116.

Input nascent configurations: λ'_4

ZOOM IN
FOR DETAILS

Output nascent configurations: λ'_2

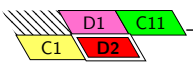
Figure S.64. Folding of Brick D2 in 

–First occurrence in 5-bits counter: Region #136.

Input nascent configurations: λ'_4

ZOOM IN
FOR DETAILS

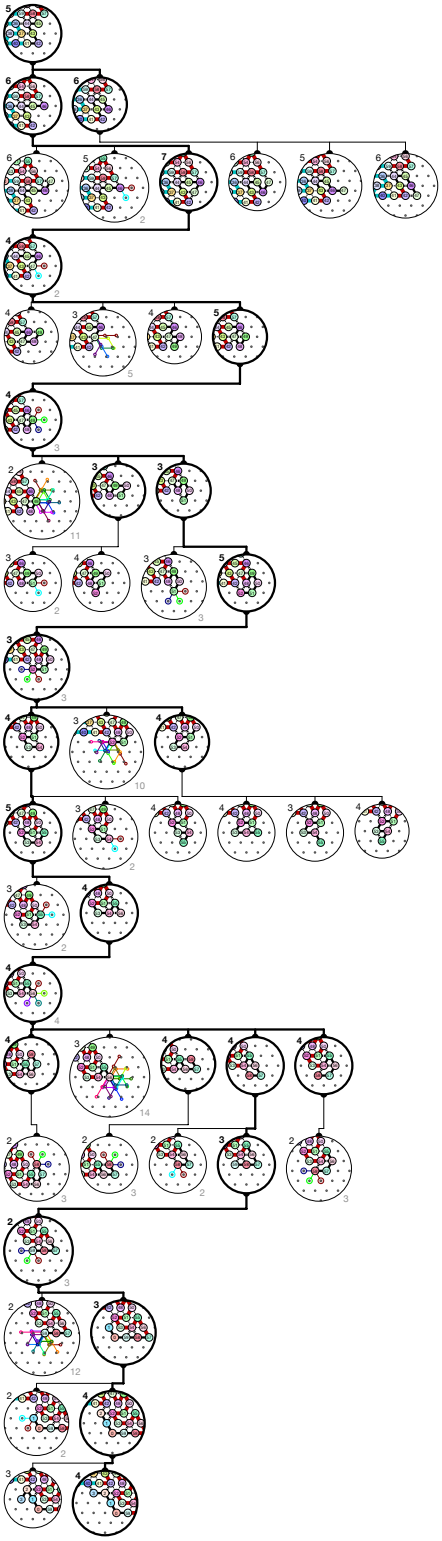
Output nascent configurations: $\mu \cup \mu'$

Figure S.65. Folding of Brick D2 in 

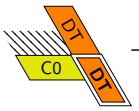
–First occurrence in 5-bits counter: Region #76.

Input nascent configurations: λ_5

ZOOM IN
FOR DETAILS



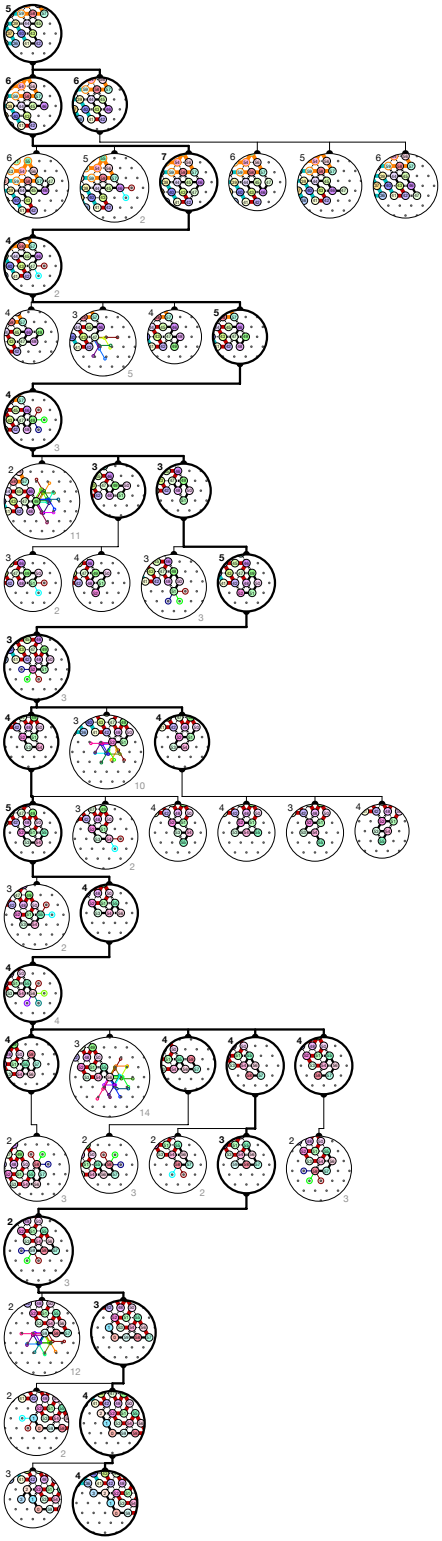
Output nascent configurations: α

Figure S.66. Folding of Brick DT in 

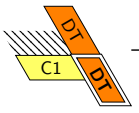
–First occurrence in 5-bits counter: Region #40.

Input nascent configurations: λ_5

ZOOM IN
FOR DETAILS



Output nascent configurations: α

Figure S.67. Folding of Brick DT in 

–First occurrence in 5-bits counter: Region #20.

S.3. Analysis of Algorithms 1 and 2

Proof of Theorem 3. The key is that every step of the folding is computed locally in a fixed and known environment: at each step i, the δ beads to be folded look for their best positions by interacting with beads with fixed and known positions within a radius $\delta + 1$. It follows that one can compute the set of all suitable subrules, considering these $O(\delta^2)$ bead types only (i.e., that place the $i - \delta + 1$-th bead of the molecule at the correct position). Oblivious $\mathscr{O}$ and inertial $\mathscr{I}$ dynamics differ as $\mathscr{I}$ only consider input nascent configurations which are output by the previous step, whereas $\mathscr{O}$ does not use any information from the previous step. This implies that one need to remember some information to connect one step to the next in $\mathscr{I}$, whereas $\mathscr{O}$ needs no memory at all.

Formally, a *subrule* $R : B^2 \to \{\texttt{true}, \texttt{false}, \bot\}$ is a symmetric function that states for each pair of beads if they attract each other (`true`) or not (`false`), or if this is undefined ($\bot$). We denote by $\operatorname{dom} R = \{(a, b) \in B^2 : R(a, b) \neq \bot\}$ the *domain* of R. Two subrules R_1 and R_2 are *compatible*, denoted by $R_1 \sim R_2$ if they agree for every pair where they are both defined, i.e., if for all $a, b \in \operatorname{dom} R_1 \cap \operatorname{dom} R_2$, $R_1(a, b) = R_2(a, b)$. We say that R_1 *matches with* R_2 if for all $(a, b) \in \operatorname{dom} R_2$, $R_1(a, b) = R_2(a, b)$. If $R_1 \sim R_2$, we denote by $R_1 \cup R_2$ the subrule obtained by *merging* R_1 and R_2, i.e. defined by $R_1 \cup R_2\,(a, b) = R_1(a, b)$ if $(a, b) \in \operatorname{dom} R_1$, and $= R_2(a, b)$ otherwise.

For all $0 \leqslant 1 \leqslant |c|$, we denote by $c_{1..i}$ the prefix of length i of a configuration c. Algorithm 1 solves RDP for the oblivious dynamics in time linear in the length of the sequence as follows. It incrementally constructs a set of rules that place each bead at its desired position in each of the k environments. It proceeds by maintaining a set $\mathcal{R}$ of candidate subrules that place correctly the first i beads in every of the k environments. At the i-th step, the main procedure FINDRULEOBLIVIOUS() first extends each candidate subrule R in $\mathcal{R}$ by calling procedure EXTENDOBLIVIOUS() which scans all the possible attraction rule extensions of R for the new nascent bead (the $(i + \delta - 1)$-th) with the bead types it can reach in each of the k configurations, and retains only the ones that place the i-th bead at its correct position for all k configurations. Note that this extension of the rule does not change the positioning of the $(i - 1)$th first beads since the $(i + \delta - 1)$-th bead is not yet produced when they are placed. Now, in order to keep the processing time constant for each bead, main procedure FINDRULEOBLIVIOUS() calls procedure PROJECTOBLIVIOUS() which retains only one representative of each subset of rules that define the same attractions for the $\delta - 1$ nascent beads (indexed from $i + 1$ to $i + \delta - 1$), i.e., for the only bead types for which the rule matters in order to determine the positions of the upcoming beads. Once the $n - \delta$-th bead is placed, the main procedure concludes by checking that the surviving rules place the last $\delta - 1$ beads at their desired positions.

Time and space complexity analysis. Let us first analyse the procedure EXTENDOBLIVIOUS. All the beads reachable by the $(i + \delta - 1)$th bead are located at distance at most $\delta + 1$ from c^j_{i-1} in the j-th target environment. The size of $\mathcal{N}$ is thus at most $3k(\delta + 1)^2$. It follows that there are at most $2^{3k(\delta+1)^2}$ subrules ρ to consider. Testing each subrule takes $O(5^\delta)$ time and $O(\delta^2)$ space. It follows that each execution of procedure EXTENDOBLIVIOUS takes $O(5^\delta 2^{3k(\delta+1)^2})$ time and $O(\delta^2 2^{3k(\delta+1)^2})$ space.

Let us now analysis the procedure PROJECTOBLIVIOUS. Again, all the beads reachable by the $(i + \delta - 1)$th bead are located at distance at most δ from c^j_i in the j-th target environment. The size of $\mathcal{N}$ is thus at most $3k\delta^2$. It follows that there are at most $2^{3k(\delta-1)\delta^2}$ subrules ρ to consider. It follows that the size of the output set Π is bounded by $2^{3k(\delta-1)\delta^2}$. Procedure PROJECTOBLIVIOUS runs then in $O(|\mathcal{R}| + 2^{3k(\delta-1)\delta^2})$ and uses at most $O(\delta^2 2^{3k(\delta-1)\delta^2})$ space.

Note that the size of $\mathcal{R}$ in the main procedure FINDRULEOBLIVIOUS is always bounded by the size of the output Π of PROJECTOBLIVIOUS times the size of the set Σ output of of EXTENDOBLIVIOUS. The size of $\mathcal{R}$ is thus at most $2^{3k(\delta^3+2\delta+1)}$ at all time. As the main procedure calls $n - \delta + 1$ times the procedures EXTENDOBLIVIOUS and the procedures PROJECTOBLIVIOUS, we conclude that Algorithm 1 runs in $O(n \cdot 5^\delta 2^{3k(\delta^3+2\delta+1+(\delta+1)^2)})$ time and uses $O(n \cdot \delta^2 2^{3k(\delta^3+2\delta+1+(\delta+1)^2)})$ space, which are both linear in n for constant k and δ.

Algorithm 2 works similarly for the inertial dynamics $\mathscr{I}$. In fact, we just extend the subrule technics by testing subrules for each possible input and corresponding output nascent configurations set, for each seed-target configurations pair and each time step. This multiplies the memory and time complexities by $2^{k5^{\delta-1}}$. □