Natural Computing (2018) 17:67–79
https://doi.org/10.1007/s11047-017-9661-y

Nondeterministic seedless oritatami systems and hardness of testing their equivalence

Yo-Sub Han[1] · Hwee Kim[1] · Makoto Ota[2] · Shinnosuke Seki[2]

Published online: 1 December 2017

Abstract
The oritatami system (OS) is a model of computation by cotranscriptional folding, being inspired by the recent experimental success of RNA origami to self-assemble an RNA tile cotranscriptionally. The OSs implemented so far, including the binary counter and Turing machine simulator, are deterministic, that is, uniquely fold into one conformation, while nondeterminism is intrinsic in biomolecular folding. We introduce nondeterminism to the OS and propose a nondeterministic OS (NOS) that chooses an assignment of Boolean values nondeterministically and evaluates a logical formula on the assignment. This NOS is seedless in the sense that it does not require any initial conformation like the RNA origami. The NOS enables proving the coNP-hardness of deciding, given two NOSs, if there exists no conformation that one of them folds but the other does not.

Keywords Oritatami system · Self-assembly · RNA cotranscriptional folding · Nondeterminism

1 Introduction

In nature, a one-dimensional RNA sequence—a primary structure—folds itself autonomously into a highly dimensional secondary structure. It has been a constant quest to predict the secondary structure from a given primary structure, and based on experimental observations, researchers have established various RNA structure-prediction models including RNAfold (Zuker and Stiegler 1981), Pknots (Rivas and Eddy 1999), mFold (Zuker 2003) and KineFold (Xayaphoummine et al. 2005). Traditional models tend to rely on the energy optimization of the whole structure.

Recently, biochemists showed that the kinetics—the step-by-step dynamics of a reaction—plays an essential role in the geometric shape of the RNA foldings (Frieda and Block 2012), since the folding caused by intermolecular reactions is faster than the RNA transcription rate (Lai et al. 2013). By controlling cotranscriptional foldings, researchers succeeded in cotranscriptionally assembling a rectangular tile out of RNA, which is called RNA origami (Geary et al. 2014), as depicted in Fig. 1. From this kinetic point of view, Geary et al. (2016) proposed a new folding model called oritatami. In general, an oritatami system (OS) defines a sequence of beads (which is the primary structure) and a set of rules for possible intermolecular reactions between beads. For each bead in the sequence, the system takes a look ahead at a few upcoming beads and determines the best location of the bead that maximizes the number of possible interactions from the look ahead. Note that the look ahead represents the reaction rate of the cotranscriptional folding and the number of interactions represents the energy level. In an OS, we call the secondary structure *the conformation*, and the resulting secondary structure *the terminal conformation*.

✉ Hwee Kim
kimhwee@yonsei.ac.kr

Yo-Sub Han
emmous@yonsei.ac.kr

Makoto Ota
o1111032@edu.cc.uec.ac.jp

Shinnosuke Seki
s.seki@uec.ac.jp

[1] Department of Computer Science, Yonsei University, 50 Yonsei-Ro, Seodaemum-Gu, Seoul 03722, Republic of Korea

[2] Graduate School of Informatics and Engineering, University of Electro-Communications, 1–5–1, Chofugaoka, Chofu, Tokyo 1828585, Japan

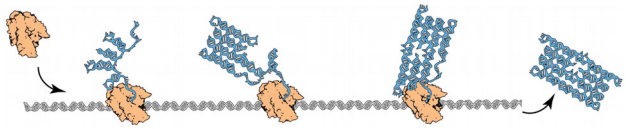

Fig. 1 RNA origami (Geary et al. 2014). The artwork is by Cody Geary

Geary et al. implemented an OS to count in binary (Geary et al. 2016) and an OS to simulate a cyclic tag system (Geary et al. 2015). These OSs uniquely fold into one conformation, and in this sense, they are deterministic. The binary counter and the cyclic tag system are all deterministic systems and thus successfully implemented by deterministic OSs, but it becomes challenging to implement a nondeterministic system using a deterministic OS. For example, if we want to implement a nondeterministic finite automaton (NFA) by a deterministic OS without converting an NFA to a deterministic finite automaton (DFA), we would need an exponential number of OSs with different seeds and primary structures to simulate all possible cases. Thus, we introduce nondeterminism to the OS, which is intrinsic in biomolecular folding and helpful to implement nondeterministic systems within one OS. We define the nondeterministic OS (NOS) in this paper, and examine its power. It turns out that nondeterminism can be made use of for OSs to execute randomized algorithms. We propose an NOS that evaluates a Boolean formula in disjunctive normal form (DNF formula) on a nondeterministic assignment. This NOS is, in fact, seedless like the RNA origami.

For computational models such as NFA, PDA (pushdown automata) or Turing machines, it is a fundamental question to ask whether or not two different systems of the model are equivalent, where the definition of equivalence is dependent on the model. The reachability problem is another fundamental problem to decide whether or not a certain state of a system is reachable from a given initial state of the system. We introduce these problems for the OS and exploit the NOS to prove their hardness. We propose two decision problems.

1. *Oritatami system equivalence problem (OSEQP):* Decide if two nondeterministic OSs have the same set of terminal conformations.
2. *Oritatami reachability problem (ORP)*: Decide if a given nondeterministic OS has a terminal conformation reaching a given point.

We prove the coNP-hardness of the OSEQP and NP-hardness of the ORP.

2 Preliminaries

Let Σ be a set of bead types, and Σ^* be the set of finite strings of beads, i.e., strings over Σ, including the empty string λ. Let $w = a_1 a_2 \cdots a_n$ be a string of length n for some integer n and bead types $a_1, \ldots, a_n \in \Sigma$. The *length* of w, which is n, is denoted by $|w|$. For two indices i, j with $1 \leq i \leq j$, we let $w[i, j]$ refer to the substring $a_i a_{i+1} \cdots a_{\min(j,n)}$; if $i = j$, then we simplify the notation as $w[i]$. We use w^n to denote the string $\underbrace{ww \cdots w}_{n}$.

OSs operate on the hexagonal lattice, where there exists a triplet of points neighboring each other. The *grid graph* (V, E) of the lattice V is the graph whose vertices correspond to the lattice points and are connected if the corresponding lattice points are at a unit distance hexagonally. For a point p and a bead type $a \in \Sigma$, we call the pair (p, a) an *annotated point*, or simply a *point* if being annotated is clear from the context. Two points p, q [or annotated points (p, a), (q, b)] are *adjacent* if $(p, q) \in E$.

A *path* is a sequence $P = p_1 p_2 \cdots p_n$ of *pairwise-distinct* points $p_1, p_2, \ldots, p_n$ such that $p_i p_{i+1}$ is at unit distance for all $1 \leq i < n$. Given a string $w \in \Sigma^*$ of bead types of length n, a *path annotated by* w, or simply *w-path*, is a sequence P_w of annotated points $(p_1, w[1]), \ldots, (p_n, w[n])$, where $p_1 \cdots p_n$ is a path and $(p_i, w[i])$, $(p_{i+1}, w[i+1])$ are adjacent for all i's. Annotated points of the w-path are regarded as a bead, and hence, we call them beads and, in particular, we call the ith point $(p_i, w[i])$ the ith bead of the w-path.

Let $\mathcal{H} \subseteq \Sigma \times \Sigma$ be a symmetric relation, specifying which types of beads can form a hydrogen bond-based interaction (h-interaction for short). This relation $\mathcal{H}$ is called the *rule set*. It is convenient to assume a special *inert* bead type $\bullet \in \Sigma$ that never forms any h-interaction according to $\mathcal{H}$.

A *conformation* C is a pair of a w-path $P_w = (p_1, w[1])\ (p_2, w[2]) \cdots$ and a set H of h-interactions, where $H \subseteq \{\{i, j\} \mid 1 \leq i, i+2 \leq j, (p_i, p_j) \in E\}$ and $\{i, j\} \in H$ implies that the ith and jth beads of the w-path form an h-interaction between them. An example conformation is found in Fig. 2(right). The condition $i + 2 \leq j$ represents the topological restriction that two beads $(p_i, w[i]), (p_{i+1}, w[i+1])$, following each other along the w-path, cannot form an h-interaction between them. The condition that p_i be adjacent to p_j prevents two beads from forming an h-interaction unless they are at unit distance. We say C is *finite* if its path is finite. From now on, when a conformation is illustrated, any unlabeled bead is assumed to be labeled with $\bullet$, that is, inert. For an integer $\alpha \geq 1$, C is of *arity* α if none of its beads interact with more than α beads. On the hexagonal lattice where every point is adjacent to six points, $\alpha > 6$ is merely meaningless, but on

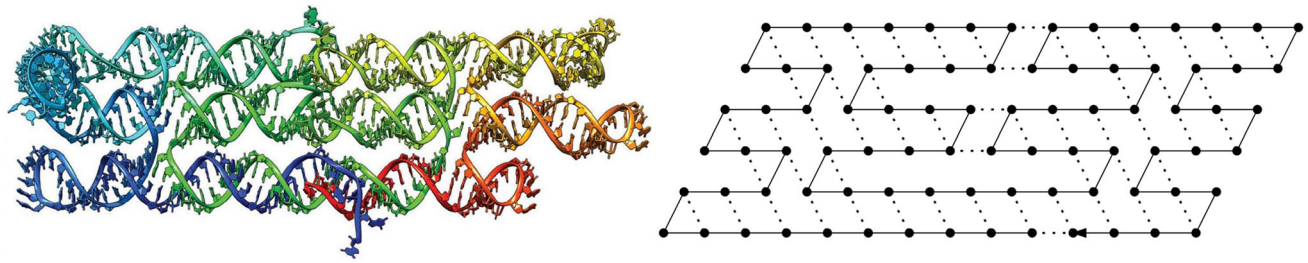

Fig. 2 (Left) An example of an RNA tile assembled by RNA origami. (Right) A conformation abstracting the RNA tile in the OS. The directed solid line represents a path, dots represent beads, and dotted lines represent h-interactions. The idea and artwork were provided by Cody Geary

another lattice larger αs may. Let $\mathcal{C}_\alpha$ be the set of all conformations of arity $= \alpha$.

A rule (a, b) in the rule set $\mathcal{H}$ is *used* in the conformation C if there exists $\{i,j\} \in H$ such that $w[i] = a$ and $w[j] = b$ or $w[i] = b$ and $w[j] = a$. A conformation C is *valid (with respect to $\mathcal{H}$)* if for all $\{i,j\} \in H$, $(w[i], w[j]) \in \mathcal{H}$. In a context with one fixed rule set, only valid conformations with respect to the rule set are considered, and we may not specify with respect to what rule set they are valid.

Given a rule set $\mathcal{H}$ and a valid finite conformation $C_1 = (P_w, H)$ with respect to $\mathcal{H}$, we say that another conformation C_2 is an *elongation of C_1 by a bead $a \in \Sigma$* if $C_2 = (P_w \cdot (p, a), H \cup H')$ for some lattice point p and (possibly empty) set of h-interactions $H' \subseteq \{\{i, |w|+1\} \mid 1 \leq i \leq |w|, (w[i], a) \in \mathcal{H}, (p_i, p) \in E\}$. Note that C_2 is also valid. For a conformation C and a finite string $w \in \Sigma^*$, by $\mathcal{E}(C, w)$, we denote the set of all elongations of C by w, that is, $\mathcal{E}(C, w) = \{C' \in \mathcal{C} \mid C \to_w^* C'\}$. For an arity α, let $\mathcal{E}_\alpha(C, w) = \mathcal{E}(C, w) \cap \mathcal{C}_\alpha$.

2.1 Oritatami system

An OS is a 5-tuple $\Xi = (\mathcal{H}, \alpha, d, \sigma, w)$, where $\mathcal{H}$ is a rule set, α is an *arity*, $d \geq 1$ is a positive integer called the *delay*, σ is an initial valid conformation of arity α called the *seed*, and w is a possibly-infinite string of beads called a *primary structure*.

The delay d, arity-α OS Ξ cotranscriptionally folds its primary structure in the following way. For a string $x \in \Sigma^*$, a conformation C_1, and an elongation C_2 of C_1 by $x[1]$, we say that Ξ *(cotranscriptionally) folds x upon C_1 into C_2* if

$$C_2 \in \operatorname*{argmin}_{C \in \mathcal{E}_\alpha(C_1, x[1])} \min\left\{\Delta G(C') \;\middle|\; C' \in \mathcal{E}_\alpha(C, x[2,d])\right\}, \tag{1}$$

where $\Delta G(C')$ is an energy function that assigns C' with the negation of the number of h-interactions within C' as energy. Informally speaking, C_2 is a conformation obtained by elongating C_1 by the bead $x[1]$ so as for the beads $x[1], x[2], \ldots, x[d]$ to create as many h-interactions as possible. Then we write $C_1 \overset{\Xi}{\hookrightarrow}_x C_2$, and the superscript Ξ is omitted whenever Ξ is clear from the context. Through the folding, the first bead of x is *stabilized*. In figures, we conventionally color x—the fragment to be stabilized—in violet.

Example 1 (*Glider*) Let us explain how an OS cotranscriptionally folds a motif called the *glider*. Gliders offer a directional linear conformation and have been heavily exploited in the existing studies on OS (Geary et al. 2015). Consider a delay-3 OS whose seed is the black conformation in Fig. 3a, primary structure is $w = (b \bullet ac \bullet bd \bullet ca \bullet d)^*$, and the rule set is $\mathcal{H} = \{(a,a), (b,b), (c,c), (d,d)\}$.

By the fragment $w[1,3] = b \bullet a$, the seed can be elongated in many ways; three of them are shown in Fig. 3a. The only bead on the fragment that may form a new h-interaction is a (b is also capable according to $\mathcal{H}$ but no other b is around). In order for the a to get next to the other a, on the seed, the b on the fragment must be located to the east of the last bead of the seed; thus, the b is stabilized there, as shown in Fig. 3b. The stabilization transcribes the next bead $w[4] = c$. The sole other c around is on the seed, but is too far for the c just transcribed to get adjacent to it. Thus, the only way for the fragment $w[2,4] = \bullet ac$ to form an h-interaction is to put the two as next to each other as before, and for that, the $\bullet$ must be located to the southeast of the preceding b as shown in Fig. 3c. The next bead to be transcribed, $w[5]$, is inert, and hence, cannot override the previous decision. The first six beads have been thus stabilized as shown in Fig. 3d, and the glider has thus moved forward by a distance of 2.

It is easily induced inductively that gliders of arbitrary "flight distance" d can be folded by a delay-3 OS; such long-distance gliders have been used in Geary et al. (2015). Moreover, as suggested in Fig. 3, a constant number of bead types are enough for that (in this example, $a, b, c, d, \bullet$).

Gliders also provide a medium to propagate 1-bit information at arbitrary distance. The height (up or down)

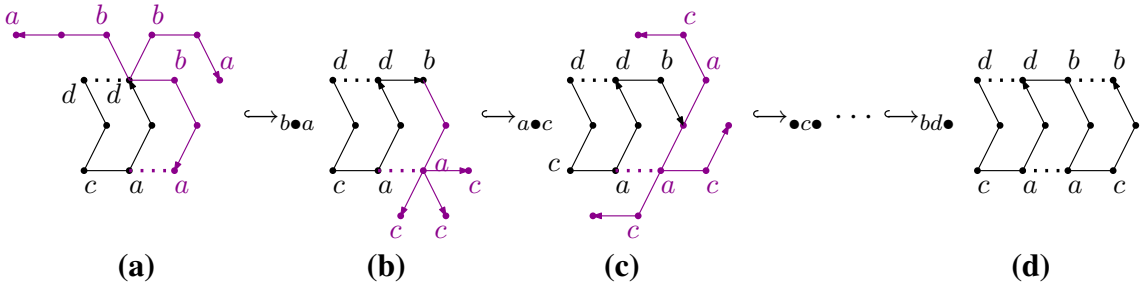

Fig. 3 A glider folded by a delay-3 OS. **a** Three ways to elongate the current conformation by the fragment $b \bullet a$ among many. **b** The three most stable elongations by the fragment $\bullet ac$. **c** Three ways to elongate the current conformation by the fragment $ac\bullet$ among many. **d** The stabilization of $b \bullet ac \bullet b$

of the first bead determines whether the last bead is stabilized up or down after the glider traverses the distance d. For instance, the glider in Fig. 3 launches up and thus its last bead (the second b) also comes up after traveling the distance $d = 2$; being launched down implies being terminated down. This capability has been exploited for an OS to simulate a cyclic tag system for Turing universality (Geary et al. 2015) and the NOS that we shall propose in Sect. 3 also uses it.

The set $\mathcal{F}(\Xi)$ of all conformations foldable by Ξ is now defined recursively as follows: $\sigma \in \mathcal{F}(\Xi)$, and if $C_i \in \mathcal{E}_\alpha(\sigma, w[1,i])$ is in $\mathcal{F}(\Xi)$ and $C_i \overset{\Xi}{\hookrightarrow}_{w[i+1,i+d]} C_{i+1}$, then $C_{i+1} \in \mathcal{F}(\Xi)$. A finite conformation $C_i \in \mathcal{E}_\alpha(\sigma, w[1,i])$ foldable by Ξ is *terminal* if one of the following conditions holds:

1. The primary structure w is finite and $i = |w|$;
2. Either w is infinite or $i < |w|$, and there exists no conformation C_{i+1} such that $C_i \overset{\Xi}{\hookrightarrow}_{w[i+1,i+\delta]} C_{i+1}$.

Note that the conformation in Fig. 4f is terminal by the second condition. By $\mathcal{F}_\square(\Xi)$, we denote the set of all terminal conformations foldable by Ξ.

The OS Ξ is *deterministic* if any foldable conformation $C_i \in \mathcal{E}_\alpha(\sigma, w[1,i])$ is either terminal or admits a unique conformation C_{i+1} such that $C_i \overset{\Xi}{\hookrightarrow}_{w[i+1,i+\delta]} C_{i+1}$, that is, every bead is stabilized uniquely. For example, the system in Fig. 4 is nondeterministic. Note that nondeterministic systems fold into multiple terminal conformations as suggested in Fig. 4. On the other hand, finite deterministic systems fold into exactly one terminal conformation by definition. Thus, an OS is deterministic if and only if the system folds into one terminal conformation. Given a delay-δ OS whose primary structure is of length n, it is decidable in $O(5^\delta n)$ time whether the OS is deterministic or not. Indeed, it suffices to run the OS and checks whether it encounters nondeterminism or not.

Example 2 (*Assignor*) Let us exhibit here how nondeterminism is used in the OS that we shall propose, or more particularly, in its module called the *assignor*. The OS is of delay 3, with a rule set including $(10^{\mathrm{eb}}, 3^{\mathrm{a}}), (10^{\mathrm{eb}}, 9^{\mathrm{a}}), (12^{\mathrm{eb}}, 3^{\mathrm{eb}}), (12^{\mathrm{eb}}, 9^{\mathrm{a}}), (12^{\mathrm{eb}}, 4^{\mathrm{a}})$. The OS folds the assignor uniquely as shown in Fig. 5a, up to its fourth last bead. The last three beads of the assignor are 10^{eb}, 11^{eb} and 12^{eb}.

The fragment $10^{\mathrm{eb}}{-}11^{\mathrm{eb}}{-}12^{\mathrm{eb}}$ can be folded in two ways equally stably with three h-interactions as shown in

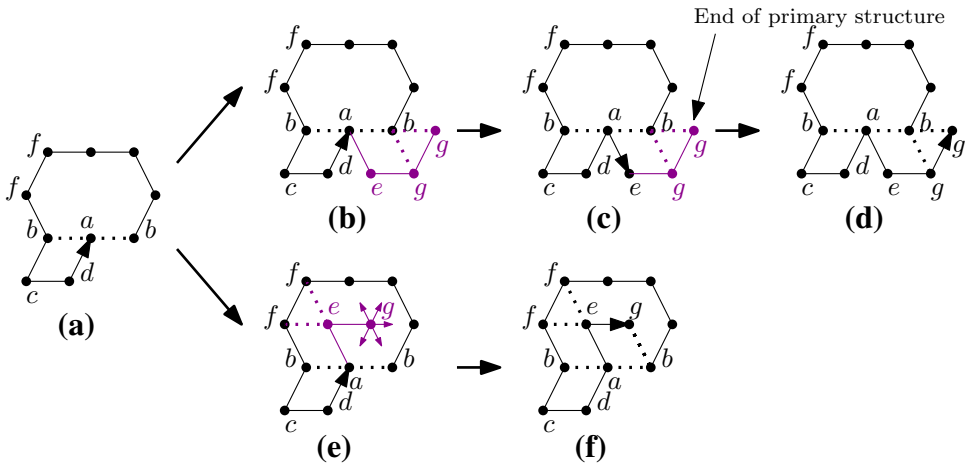

Fig. 4 The two cases in which cotranscriptional folding considers "halfway" elongations. **b**, **e** Two most stable elongations of the current conformation (**a**), the one in **e** is a halfway elongation. **b**–**d** How the primary structure is fully transcribed and folded. **e**, **f** The alternative, in which the last g-bead cannot be transcribed due to the lack of space. The conformations in **d** and **f** are both terminal, though the one in **f** is "shorter"

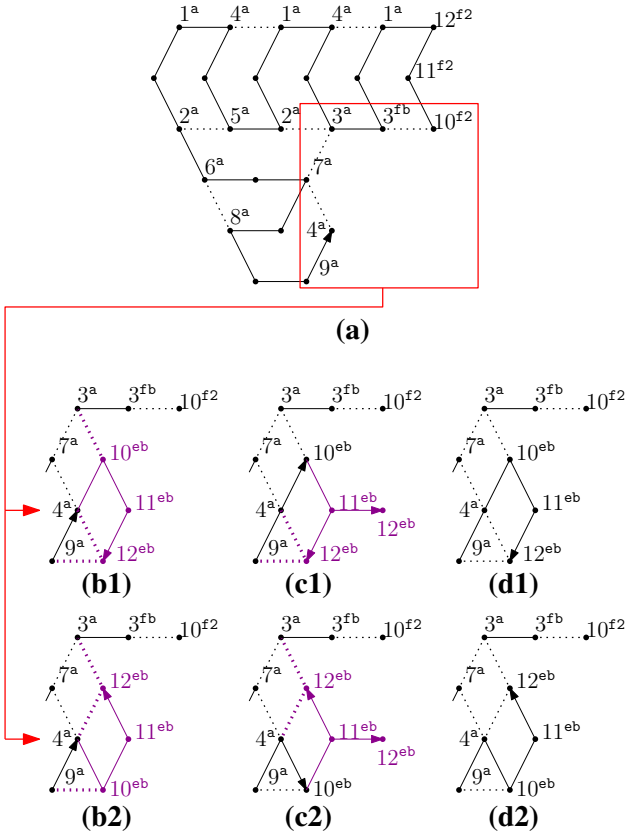

Fig. 5 An assignor folded by a delay-3 OS. While stabilizing the bead 10^{eb}, two elongations equally give three interactions and the bead nondeterministically stabilizes at two different points. The bead 10^{eb} is stabilized nondeterministically at two different points because the two elongations in **b1** and **b2** equally give three interactions. **a** The conformation up to the fourth last bead. **b1**, **b2** Two ways to elongate the current conformation by the fragment $10^{eb}-11^{eb}-12^{eb}$. **c1**, **c2** Ways to elongate the current conformation by the fragment $11^{eb}-12^{eb}$. **d1**, **d2** Two final conformations

Fig. 5(b-1), (b-2). The bead 10^{eb} is stabilized accordingly in the two ways shown in Fig. 5(c-1), (c-2). The remaining beads 11^{eb} and 12^{eb} are stabilized uniquely one after another as shown in Fig. 5(d-1), (d-2). As a result, the assignor nondeterministically stabilizes the last bead 12^{eb} up or down. In our NOS, this nondeterministic assignment of 1-bit information is propagated by gliders in the way mentioned in Example 1.

We say that an OS is *seedless* if its seed is $(\lambda, \emptyset)$. A seedless OS can start folding at any point of the lattice. If a conformation C is foldable, then any of its congruence, that is, a conformation obtained by applying a combination of translation, rotation, and reflection to C is also foldable. Therefore, fixing the first bead to the origin of the lattice and the second bead to one specific neighbor of the origin does not cause any loss of generality. In this sense, we can regard an OS with a seed of at most two beads seedless. Furthermore, a seed of three beads can make an OS seedless. Imagine an OS whose seed consists of two beads. If an elongation of the seed by the first bead is foldable, then its reflection across the seed, which is just a line segment, is also foldable. Hence, if the first bead is stabilized uniquely, then it can be rather considered as a part of the seed. That is the case for the OS we shall propose in Sect. 3. In this sense, the OS is seedless.

We formally define the equivalence of two OSs.

Definition 1 Two OSs Ξ_1 and Ξ_2 are *equivalent* if $\mathcal{F}_\square(\Xi_1) = \mathcal{F}_\square(\Xi_2)$, namely, if two OSs fold the same set of terminal conformations.

Then, we define oritatami System Equivalence Problem.

Definition 2 Given two OSs Ξ_1 and Ξ_2, the OSEQP is used to determine whether or not they are equivalent.

Given a constant delay d, the OSEQP can be solved in linear time for the class of delay-d deterministic OSs. On the other hand, the problem for NOSs turns out to be coNP-hard even for delay-3 OSs, and we shall prove the hardness in Sect. 3.

3 Seedless NOS as a DNF verifier

We propose a seedless NOS that evaluates a given DNF formula, i.e., a Boolean formula in the disjunctive normal form, and then make use of it to prove the coNP-hardness of deciding if two given NOSs are equivalent even under a strict constraint— a difference of just one rule in the rule set.

A DNF formula ϕ is written as $\bigvee_{1 \le i \le n} C_i$ for some clauses $C_1, \ldots, C_n$ that is a logical AND ($\wedge$) of some of the Boolean variables $v_1, \ldots, v_m$ and their negations. The *DNF tautology problem* asks if a given DNF formula is evaluated to TRUE on all possible assignments. The problem is coNP-hard, since it can be polynomially reduced from the dual problem of the satisfiability problem, which is NP-complete (Kleinberg and Tardos 2011).

Algorithm 1: Evaluate a DNF formula with m variables and n clauses formula on a nondeterministically chosen assignment

```
1 for k = 1 to n do c[k] ← *
2 for i = 1 to m do
3 |   Nondeterministically assign TRUE or FALSE into v_i.
4 |   for k = 1 to n do
5 |   |   if The k-th clause involves v_i and v_i = FALSE
  |   |     then c[k] ← U
6 |   |   else if The k-th clause involves ¬v_i and
  |   |     v_i = TRUE then c[k] ← U
7 for k = 1 to n do
8 |   if c[k] = * then return Satisfied
9 return Unsatisfied
```

Algorithm 1 evaluates the DNF formula ϕ on a nondeterministic assignment. For the ease of explanation, we assume m is even (otherwise we just assume one more imaginary variable that occurs nowhere). The seedless NOS Ξ_τ evaluates ϕ using this algorithm. Both its delay and arity are set to 3 (in fact, the arity can be set to any value larger). We conventionally use the term *context* to denote beads and interactions around the current bead that we consider during the stabilization.

3.1 Overview of the NOS

The NOS Ξ_τ consists of modules and submodules, which are fragments of the primary structure that folds differently according to the context. Figure 6 gives an outline of Ξ_τ. For a given DNF formula with m variables and n clauses, the primary structure of Ξ_τ is of the form $w_s w_1 w_2 \cdots w_m w_v$, where we call the factors $w_s, w_1, \ldots, w_m, w_v$ *modules*. The NOS Ξ_τ implements Algorithm 1 by the following modules:

- *Starter* The starter w_s folds into the glider as shown in Fig. 7 and offers a linear scaffold of width $O(n)$, which serves as a "seed" for the succeeding modules. The role of the starter corresponds to the initialization of the array c in line 1 of Algorithm 1.
- The next modules $w_i (1 \le i \le m)$ consist of the following submodules:
  - *Assignor* The basic structure of the assignor was explained in Example 2. It nondeterministically stabilizes its last bead up or down, and the OS interprets up as TRUE being assigned to v_i and down as false being assigned to v_i (see Fig. 6, where TRUE is assigned to v_1, for instance). The assignor is followed by submodules called evaluators and buffers, which occur n times alternately.
    - *Evaluator* Each evaluator corresponds to a clause. The kth evaluator, for C_k, takes the value (T/F) assigned to v_i from the left and the evaluation from the top as inputs, and outputs the value of v_i to the right faithfully and the updated evaluation of whether the clause C_k is still satisfiable or not according to the value of v_i to the bottom.
    - *Buffer* The buffer is just a glider. Buffers keep a sufficient distance between the consecutive submodules horizontally so as for them not to interact with each other.
  - *Turner* The turner is used to turn the direction of the folding. The turner is followed by submodules called buffers and formatters, which occur n times alternately.
  - *Formatter* Since evaluators output each evaluation in two distinct formats, these outputs are reformatted by formatters to be inputs for the following evaluators.
- The folding of w_i corresponds to the ith iteration of the **for**-loop at line 1–4 of Algorithm 1.
- *Verifier* The final module w_v verifies if there is a clause still evaluated to be satisfiable, corresponding to the termination process at line 5–7. The module is named the *verifier* after this role.

3.2 Detailed construction

Note that we denote each bead in a module by its order, and we use superscripts to indicate sets of bead types used for different modules, i.e., $1^{\mathtt{f2}}$ is used for formatters and $3^{\mathtt{s}}$ is used for starters.

3.2.1 Starter

The conformation that the starter folds is illustrated in Fig. 7. This module admits no other conformation, and it is almost straightforward to design a module that folds uniquely by hardcoding the target conformation into the primary structure and rule set. In fact, all the rules used in the glider in Fig. 7 are sufficient (and necessary) for the primary structure of w_s to fold into the glider. We have the primary structure w_s and the rule set $\mathcal{H}_s$ as shown in Fig. 7.

Being thus folded, the starter exposes below n copies of the sequence of bead types $10^{\mathtt{f2}}$–$9^{\mathtt{f2}}$–$8^{\mathtt{f2}}$–$7^{\mathtt{f2}}$ at a fixed interval (every eight points), which shall be translated by succeeding modules as the corresponding clause being satisfiable (denoted by $*$).

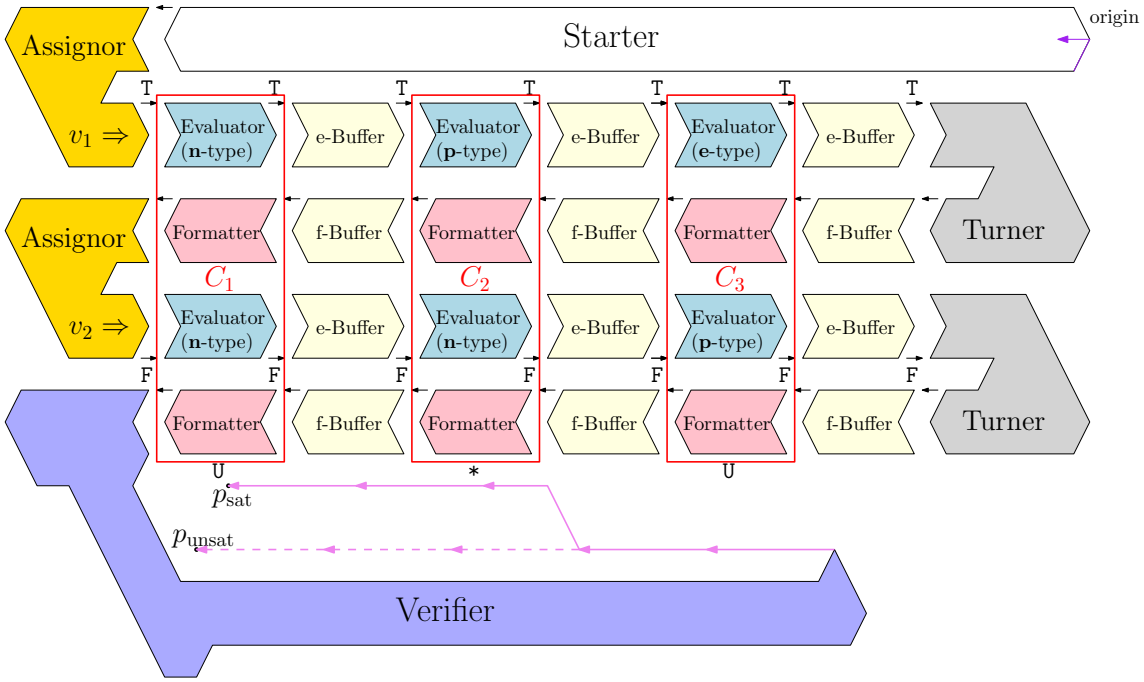

Fig. 6 An overview of the NOS that evaluates a given DNF formula $\phi = (\neg v_1 \wedge \neg v_2) \vee (v_1 \wedge \neg v_2) \vee (v_2)$, with two variables v_1, v_2 and three clauses C_1, C_2, C_3, on the assignment $(v_1, v_2) = (\mathtt{T}, \mathtt{F})$ that it chooses nondeterministically. The folding starts from the purple arrow in the starter. In the last module called the verifier, the conformation folds to p_{sat} since the formula is satisfied. Pink dashed lines represent an alternate conformation that stops at p_{unsat} when the formula is not satisfied. (Color figure online)

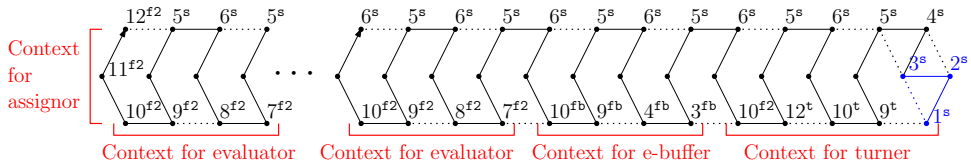

Fig. 7 The linear scaffold conformation into which the starter w_s deterministically folds. Three blue beads indicate the seed. (Color figure online)

3.2.2 Assignor

The first submodule of w_i, for the ith zigzag, is the assignor w_a. We have the primary structure w_a and rule set $\mathcal{H}_a$ as shown in Fig. 5. Its role is to assign a Boolean value to the ith variable v_i *nondeterministically*, corresponding to line 2 of Algorithm 1. The submodule folds into two different conformations in Fig. 5, where the upper conformation represents $v_i = \mathtt{T}$ and the lower conformation represents $v_i = \mathtt{F}$. All conformations that evaluators/formatters admit place their first and last beads at the same height in order to propagate the assigned value as illustrated in Fig. 8. It is only the assignor that makes use of nondeterminism; nondeterminism occurs nowhere else in the NOS (Fig. 9).

3.2.3 p-, n-, and e-Evaluators

Note that the evaluator takes the value assigned to v_i from the left and the evaluation from the top as inputs, and outputs the value of v_i to the right and the updated evaluation to the bottom. Therefore, four distinct conformations are sufficient for evaluators, and we chose those in Fig. 8. There are three possible ways to update, depending on if C_k includes v_1, or its negation, or none of them. Hence, the OS employs three types of evaluators: p, n, and e. For example, as shown in Fig. 8, the p-evaluator folds into the conformation (a) no matter how the clause has been evaluated so far if $v_1 = \mathtt{F}$, while it folds into (c) or (d) depending on the evaluation if $v_1 = \mathtt{T}$. Hence, the p-evaluator never folds into (b). The clauses C_1, C_2 and C_3 of the formula evaluated in Fig. 6 include $\neg v_1$, v_1, and none of them, respectively, and hence the first three evaluators are of type n, p, and e, respectively. Note that evaluators output each evaluation in two distinct formats ($\mathtt{U}_1, \mathtt{U}_2$ for unsatisfied, $*_1, *_2$ for satisfiable), which will be reformatted by the *formatter* in the succeeding zag, as 10-9-8-7 for $*$ and 10-9-4-3 for U. Analogously, for any $2 \leq i \leq m$, the module w_i first assigns T or F randomly to v_i and updates the evaluation of clauses provided by the previous zag (of w_{i-1}).

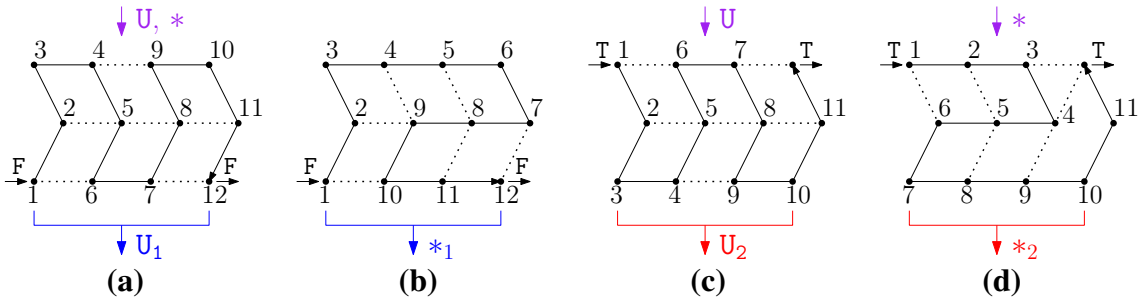

Fig. 8 The possible four conformations of evaluators and formatters. In order to propagate the Boolean value, all of the conformations start and end at the same height. Arrows at the top and bottom denote respectively the previous and updated evaluations of the corresponding clause, though they are in different formats. The arrows from the top indicate which conformations the p-evaluator takes, depending on whether F or T is input from the left; hence, the p-evaluator never takes the conformation (**b**)

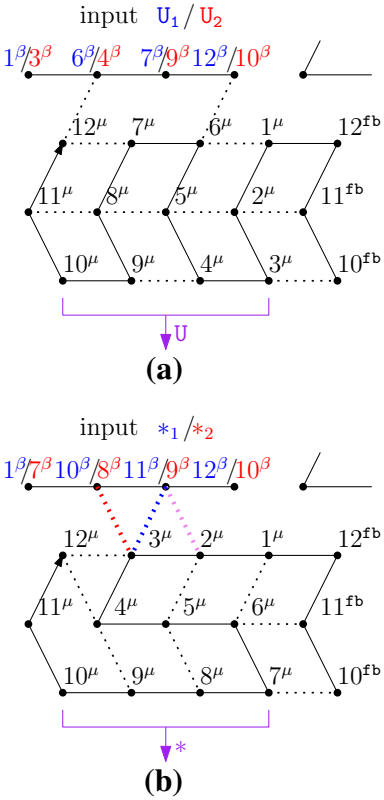

Fig. 9 **a** The glider-like conformation that a formatter takes on the input $\mathtt{U}_1/\mathtt{U}_2$. **b** The alternative conformation of a formatter on the input $*_1/*_2$. Blue and red interactions are for the corresponding colored bead only. (Color figure online)

3.2.4 Turner

The turner w_t is a hardcoded submodule, and the primary structure w_t and the rule set $\mathcal{H}_t$ follow the conformation shown in Fig. 10. Its two possible conformations let multiple possible paths arisen from the nondeterministic value assignment to v_i converge into one path (at this point, the system is allowed to "forget" the value).

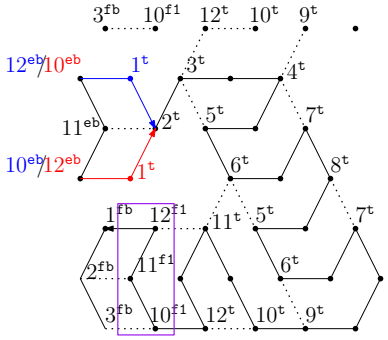

Fig. 10 The two possible conformations of turners, which converge multiple possible paths due to nondeterministic value assignment into one path. The purple box shows the context for the next *f*-buffer. (Color figure online)

3.2.5 Verifier

The verifier first folds into the conformation shown in Fig. 11. Like the starter, this conformation is also hard-coded; all the rules used are sufficient for the conformation to be folded as shown in Fig. 11. We have the primary structure w_{v1} and the rule $\mathcal{H}_{v1}$ as shown in Fig. 11. The rest of the verifier is to thread its way from right to left through the recess between the last zag and the floor just made by the first part of the verifier, as shown in Fig. 6. It has the primary structure

$$w_{v2} = (22^{\mathtt{v}}15^{\mathtt{v}}16^{\mathtt{v}}17^{\mathtt{v}}18^{\mathtt{v}}19^{\mathtt{v}}20^{\mathtt{v}}21^{\mathtt{v}})^{n-1} \cdot (22^{\mathtt{v2}}15^{\mathtt{v2}}16^{\mathtt{v2}}17^{\mathtt{v2}})$$

and the rule set

$$\mathcal{H}_{v2} = \left\{ \begin{array}{c} (15^{\mathtt{v}}, 9^{\mathtt{v}}), (15^{\mathtt{v2}}, 9^{\mathtt{v}}), (16^{\mathtt{v}}, 7^{\mathtt{f2}}), (16^{\mathtt{v2}}, 7^{\mathtt{f2}}), \\ (17^{\mathtt{v}}, 7^{\mathtt{v}}), (17^{\mathtt{v2}}, 7^{\mathtt{v}}), (17^{\mathtt{v}}, 7^{\mathtt{f2}}), (17^{\mathtt{v2}}, 7^{\mathtt{f2}}), \\ (17^{\mathtt{v}}, 8^{\mathtt{f2}}), (17^{\mathtt{v2}}, 8^{\mathtt{f2}}), (17^{\mathtt{v}}, 4^{\mathtt{f2}}), (17^{\mathtt{v2}}, 4^{\mathtt{f2}}), \\ (19^{\mathtt{v}}, 13^{\mathtt{v}}), (21^{\mathtt{v}}, 11^{\mathtt{v}}) \end{array} \right\}.$$

More precisely, it is stretched straight along the floor and once it "detects" a satisfiable clause, or the encoding of $*$, i.e., $10^{\mathtt{f2}}$–$9^{\mathtt{f2}}$–$8^{\mathtt{f2}}$–$7^{\mathtt{f2}}$, it is pulled up and starts being stretched straight along the zag above. The detection is

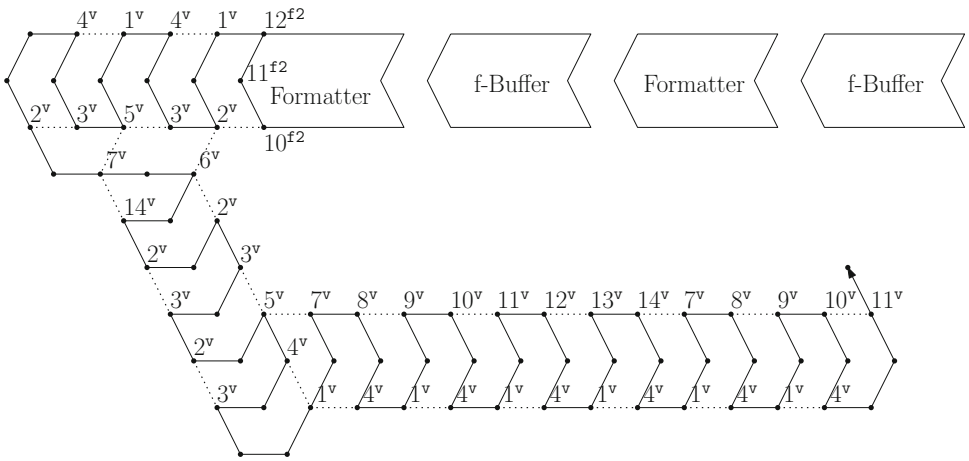

Fig. 11 The first part of the verifier uniquely folds thus and provides a scaffold on which the rest of the verifier serves its role to verify the clauses

done by the segment $15^{\mathrm{v}}-16^{\mathrm{v}}-17^{\mathrm{v}}$, which we name the *probe*. The probe forms two *h*-interactions with the floor, but more *h*-interactions with the encoding of $*$ due to the rules $(17^{\mathrm{v}}, 8^{\mathrm{f2}}), (17^{\mathrm{v}}, 7^{\mathrm{f2}})$, and $(16^{\mathrm{v}}, 7^{\mathrm{f2}})$ [see Fig. 12 (Left)]. In contrast, the probe can form only one *h*-interaction with the encoding of U, as shown in Fig. 12(Middle) due to lack of rules. As a result, the last bead of the probe is stabilized close to the zag above (at the point $p_{\mathtt{sat}}$ in Fig. 6) if and only if ϕ is satisfied by the chosen assignment. The last probe is of distinct bead types as $15^{\mathrm{v2}}-16^{\mathrm{v2}}-17^{\mathrm{v2}}$ for the sake of proving the hardness of the OSEQP later.

3.2.6 The *i*th zigzag

It now suffices to explain the module w_i for the *i*th zigzag ($1 \leq i \leq m$) into detail. Its primary structure is made up as $w_a u_{i,1} \triangleright u_{i,2} \triangleright \cdots \triangleright u_{i,n} \triangleright w_t \triangleleft f_{i,n} \triangleleft \cdots \triangleleft f_{i,2} \triangleleft f_{i,1}$, where w_a is the assignor, w_t is the turner, and $u_{i,k}$ and $f_{i,k}$ are the evaluator and formatter for the *k*th clause, respectively, and triangles ($\triangleright$ and $\triangleleft$) indicate sequences of 12 beads called *e-buffers* and *f-buffers* respectively. Buffers keep a sufficient distance between the consecutive submodules horizontally so as for them not to interact with each other. As shown in Fig. 6, buffers in a consecutive zig and zag may get adjacent to each other vertically. Should they involve a common bead type, an inter-buffer interaction could prevent them from folding into a glider. Therefore, e-buffers and f-buffers use pairwise-distinct sets of bead types.

3.2.7 Evaluator and formatter

The *k*th evaluator $u_{i,k}$ and the *k*th formatter $f_{i,k}$ in the *i*th zig and zag cooperatively update the evaluation of whether the *k*th clause is still satisfiable or already unsatisfied. The role of formatters is auxiliary: as we already explained, the output of evaluators ($*$/U) is encoded (as a sequence of beads exposed below) redundantly, and formatters reformat them and ensure that evaluators in the next zig suffice to be capable of reading one sequence of beads for $*$ and one for U. The glider-based foundation, which we will explain shortly, is modified in such a way that the evaluator and formatter inherit the information transfer capability, which enables the n evaluators for the *i*th zig, that is,

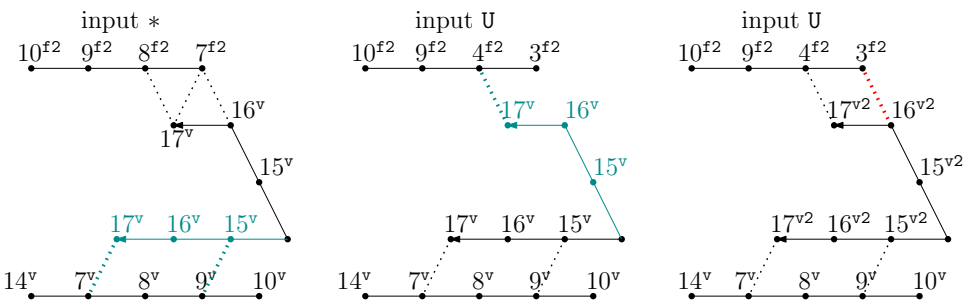

Fig. 12 Detection of satisfiable clauses by the probe segment $15^{\mathrm{v}}-16^{\mathrm{v}}-17^{\mathrm{v}}$. (Left) The sequence $8^{\mathrm{f2}}-7^{\mathrm{f2}}$, a part of the encoding of $*$, pulls the probe strongly by three *h*-interactions, one more than the number of *h*-interactions between the probe and the floor. (Middle) The sequence $4^{\mathrm{f2}}-3^{\mathrm{f2}}$, a part of the encoding of U, cannot pull the probe as strongly as the floor does. (Right) The rule $(3^{\mathrm{f2}}, 16^{\mathrm{v2}})$ added to $\Xi_{\tau 1}$ allows the probe for C_1 to be pulled upward as well as leftward equally strongly when C_1 is evaluated to be unsatisfied

$u_{i,1}, u_{i,2}, \ldots, u_{i,n}$, to transfer the value nondeterministically assigned to v_i one after another.

The basic module is to fold its primary structure $1^{\mu}2^{\mu}\cdots 11^{\mu}12^{\mu}$ (we use Greek letters to represent a set of different bead types) into one of the two gliders shown in Fig. 13 deterministically depending on the two possible contexts (in another context, it could admit another conformation, but in the proposed NOS, the such context is never encountered). It is implemented using the rule set

$$\mathcal{H}_m = \left\{ \begin{array}{c} (2^{\mu}, 11^{\gamma}), (3^{\mu}, 1^{\alpha}), (3^{\mu}, 3^{\alpha}), (3^{\mu}, 10^{\gamma}), \\ (5^{\mu}, 2^{\mu}), (6^{\mu}, 11^{\gamma}), (6^{\mu}, 1^{\mu}), (6^{\mu}, 10^{\beta}), \\ (6^{\mu}, 12^{\beta}), (7^{\mu}, 10^{\gamma}), (8^{\mu}, 5^{\mu}), (9^{\mu}, 2^{\mu}), \\ (9^{\mu}, 4^{\mu}), (9^{\mu}, 7^{\beta}), (9^{\mu}, 9^{\beta}), (9^{\mu}, 11^{\beta}), \\ (10^{\mu}, 1^{\mu}), (11^{\mu}, 8^{\mu}), (12^{\mu}, 4^{\beta}), (12^{\mu}, 6^{\beta}), \\ (12^{\mu}, 8^{\beta}), (12^{\mu}, 10^{\beta}), (12^{\mu}, 3^{\mu}), (12^{\mu}, 4^{\mu}), \\ (12^{\mu}, 7^{\mu}) \end{array} \right\}$$

where

$$\{(\alpha, \beta, \gamma, \mu)\} = \left\{ \begin{array}{l} (\mathtt{fb}, \mathtt{f2}, \mathtt{eb}, \{\mathtt{p1}, \mathtt{n1}, \mathtt{e1}\}), \\ (\mathtt{eb}, \{\mathtt{p1}, \mathtt{n1}, \mathtt{e1}\}, \mathtt{fb}, \mathtt{f1}), \\ (\mathtt{fb}, \mathtt{f1}, \mathtt{eb}, \{\mathtt{p2}, \mathtt{n2}, \mathtt{e2}\}), \\ (\mathtt{eb}, \{\mathtt{p2}, \mathtt{n2}, \mathtt{e2}\}, \mathtt{fb}, \mathtt{f2}) \end{array} \right\}.$$

The evaluator and formatter are derived by "equipping" the basic module with the capability of "reading" the output of the module above; formally speaking, we add some rules to the basic rule set $\mathcal{H}_m$ that attracts some factor of the primary structure (called *input reader*) towards the output so that the resulting module favors another conformation over the glider. Here, one design criterion should be noted: we designed the NOS in such a manner that a

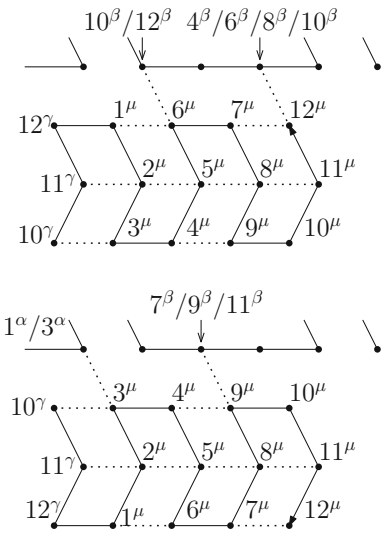

Fig. 13 The basic module is a modification of the glider and folds its primary structure $1^{\mu}2^{\mu}\cdots 12^{\mu}$ into these two conformations deterministically depending on whether the first bead 1^{μ} is up or down

module (evaluator/formatter) taking a glider represents the evaluation `U`. In Fig. 8, the two non-glider conformations are illustrated. Note that these conformations properly propagate the value (`F`/`T`) nondeterministically assigned to v_i by the assignor.

Let us focus attention on the evaluator $u_{i,k}$, which evaluates the kth clause C_k according to the value nondeterministically assigned to v_i and the evaluation made so far by the previous evaluators $u_{1,k}, u_{2,k}, \ldots, u_{i-1,k}$. There are three possibilities to be taken into account depending on whether C_k contains the positive literal v_i, its negation $\neg v_i$, or none of them. That is, three types of evaluators (`p`, `n`, and `e`) are needed. The `p`-type evaluator is supposed to fold into the glider (`U`) no matter what the previous evaluation is if $v_i = \mathtt{F}$, but be capable of taking both the glider and a non-glider conformation so as to propagate the previous evaluation as it is when $v_i = \mathtt{T}$, corresponding to line 4. The `n`-type evaluator is supposed to behave analogously, but the roles of `F` and `T` are flipped, corresponding to line 4. The `e`-type evaluator should propagate the previous evaluation as it is no matter which value is assigned to v_i.

The evaluator $u_{i,k}$ is sandwiched from above and below by two formatters. Since the evaluator interacts with both of them, a bead type common in these formatters would cause misfolding. Therefore, the NOS implements evaluators in consecutive zigs using two pairwise-distinct sets of bead types, even if they are of the identical type. This results in, for instance, two sets of bead types $\{\mathtt{p1}, \mathtt{p2}\}$ for the type-`p` evaluators. Similarly, the NOS uses two distinct sets of bead types for each type of evaluators and the formatter, which we distinguish with numbers.

We propose the following two sets $\mathcal{H}_{\mathtt{T}}, \mathcal{H}_{\mathtt{F}}$ of extra rules, which enable the module to read the output when $v_i = \mathtt{T}$ and when $v_i = \mathtt{F}$, respectively:

$$\mathcal{H}_{\mathtt{T}} = \{(2^{\mu}, 9^{\beta}), (3^{\mu}, 8^{\beta}), (4^{\mu}, 7^{\beta}), (5^{\mu}, 10^{\mathtt{fb}})\}$$

where

$$\{(\beta, \mu)\} = \{(\mathtt{f2}, \mathtt{p1}), (\mathtt{f2}, \mathtt{e1}), (\mathtt{f1}, \mathtt{p2}), (\mathtt{f1}, \mathtt{e2})\},$$

and

$$\mathcal{H}_{\mathtt{F}} = \{(5^{\mu}, 8^{\beta}), (6^{\mu}, 7^{\beta}), (7^{\mu}, 10^{\mathtt{fb}})\}$$

where

$$\{(\beta, \mu)\} = \{(\mathtt{f2}, \mathtt{n1}), (\mathtt{f2}, \mathtt{e1}), (\mathtt{f1}, \mathtt{n2}), (\mathtt{f1}, \mathtt{e2})\}.$$

Rules in $\mathcal{H}_{\mathtt{T}}$ convert the module with sets of bead types $\{\mathtt{p1}, \mathtt{p2}\}$ into the type-`p` evaluator, while rules in $\mathcal{H}_{\mathtt{F}}$ convert the module with $\{\mathtt{n1}, \mathtt{n2}\}$ into the type-`n` evaluator. We claim that these extra rule sets do not interfere with each other, and adding both of them converts the module with $\{\mathtt{e1}, \mathtt{e2}\}$ into the type-`e` evaluator. Figure 14 shows foldings with newly added rules.

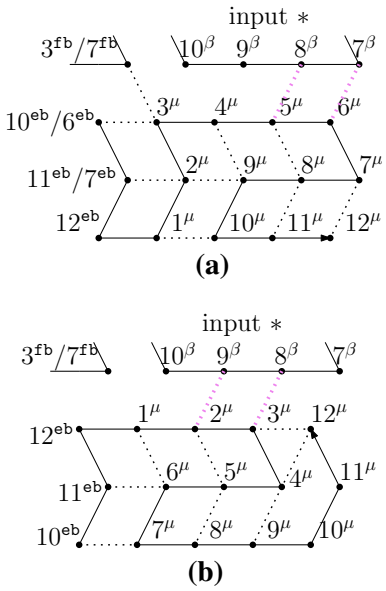

Fig. 14 **a** Folding of n-evaluators and e-evaluators when $v_i = \mathtt{F}$ and the input is $*$. Newly added rules are colored in pink. **b** Folding of p-evaluators and e-evaluators when $v_i = \mathtt{T}$ and the input is $*$

The outputs of an evaluator are redundant: $*$ is encoded as $1^\mu 10^\mu 11^\mu 12^\mu$ or $7^\mu 8^\mu 9^\mu 10^\mu$ whereas U is encoded as $1^\mu 6^\mu 7^\mu 12^\mu$ or $3^\mu 4^\mu 9^\mu 10^\mu$. The redundant output is reformatted by formatters in the ith zag so as for an input to the evaluators in the next (i+1th) zig to be encoded in a unique format. Unlike the evaluator, formatters do not have to propagate 1-bit information horizontally. The conformation of the turner fixes the first bead of the first formatter $f_{i,n}$ in the ith zag up. The following rule set $\mathcal{H}_{\mathtt{format}}$ converts the module with sets of bead types $\{\mathtt{f1}, \mathtt{f2}\}$ into the formatter, which takes the conformation (c) in Fig. 8 if the output of the evaluator above is U or the conformation (d) if the output is $*$:

$$\mathcal{H}_{\mathtt{format}} = \left\{ \begin{array}{l} (2^\mu, 9^\beta), (2^\mu, 11^\beta), (3^\mu, 8^\beta), \\ (3^\mu, 11^\beta), (4^\mu, 8^\beta), (4^\mu, 1^\beta) \end{array} \right\}$$

where

$$\{(\beta, \mu)\} = \left\{ \begin{array}{l} (\mathtt{p1}, \mathtt{f1}), (\mathtt{n1}, \mathtt{f1}), (\mathtt{e1}, \mathtt{f1}), \\ (\mathtt{p2}, \mathtt{f2}), (\mathtt{n2}, \mathtt{f2}), (\mathtt{e2}, \mathtt{f2}) \end{array} \right\}.$$

Figure 9 shows foldings with newly added rules.

3.3 Construction of the primary structure

Given a DNF formula $\phi = \bigvee_{1 \le i \le n} C_i$ with m variables $v_1, v_2, \ldots, v_m$, we can construct Ξ_τ as follows: $\Xi_\tau = (\mathcal{H}, 3, 3, 1^{\mathtt{s}} 2^{\mathtt{s}} 3^{\mathtt{s}}, w)$ where $\mathcal{H}$ is the union of all rule sets for modules and submodules, and w is generated by Algorithm 2.

Algorithm 2: Construct the primary structure for Ξ_τ

```
w ← w_s
w_μ ← 1^μ 2^μ ··· 12^μ for
    μ ∈ {p1, n1, e1, p2, n2, e2, eb, fb, f1, f2}
for j = 1 to m do
    w_i ← w_a
    for i = 1 to n do
        if C_i has v_j then
            if j is odd then w_i ← w_i · w_p1
            else w_i ← w_i · w_p2
        else if C_i has ¬v_j then
            if j is odd then w_i ← w_i · w_n1
            else w_i ← w_i · w_n2
        else
            if j is odd then w_i ← w_i · w_e1
            else w_i ← w_i · w_e2
        w_i ← w_i · w_eb
    w_i ← w_i · w_t
    for i = n to 1 do
        if j is odd then w_i ← w_i · w_fb w_f1
        else w_i ← w_i · w_fb w_f2
    w ← w · w_i
w ← w · w_v1 w_v2 return w
```

We establish the following theorem for Ξ_τ.

Theorem 1 *Let Ξ_τ be the seedless NOS generated from a DNF formula ϕ. Then, ϕ is tautology if and only if there is no conformation of Ξ_τ that reaches the point p_{unsat}.*

4 Utilization of Ξ_τ for OS decision problems

We have established a design of Ξ_τ to solve the DNF tautology problem. Now we utilize Ξ_τ to solve or reduce other problems including the OSEQP.

First, we prove the hardness of the OSEQP by variations of Ξ_τ. Note that the size of the rule set in Ξ_τ is constant, and it takes $O(nm)$ time to construct the primary structure of Ξ_τ from a DNF formula with n clauses and m variables. Thus, we can construct Ξ_τ that represents the given formula in $O(nm)$ time. The coNP-hardness of the OSEQP is proved by reduction from DNF tautology. We derive $\Xi_{\tau 1}$ from Ξ_τ by adding one additional rule $(16^{\mathtt{v2}}, 3^{\mathtt{f1}})$, which makes the verification of $\Xi_{\tau 1}$ nondeterministic when ϕ is not tautology as illustrated in Fig. 12. Thus, ϕ is tautology if and only if Ξ_τ and $\Xi_{\tau 1}$ are equivalent. This proves that the OSEQP is coNP-hard even if two OSs are seedless and identical except for their rule sets $\mathcal{H}_1, \mathcal{H}_2$ such that $\mathcal{H}_1 \subseteq \mathcal{H}_2$ and $|\mathcal{H}_2| - |\mathcal{H}_1| = 1$.

Theorem 2 *OSEQP is coNP-hard, even if the two input NOSs differ only in rule set, and their rule*

sets $\mathcal{H}_1, \mathcal{H}_2$ *satisfy* $\mathcal{H}_1 = \mathcal{H}_2 \cup \{(a, b)\}$ *for some rule* (a, b).

Next, to prove the NP-hardness of the ORP, we claim that Ξ_τ can be used to solve the SAT problem.

Definition 3 (*SAT*) Given a Boolean formula of CNF, where the formula is the conjunction of clauses, with each clause being the disjunction of variables, determine whether or not there exists an assignment that makes the formula evaluate to TRUE.

It is well known that the SAT problem is NP-complete. Note that the input formula for the SAT problem is made of clauses and variables, same as the input formula for tautology problem. Thus, we may use the same idea of Ξ_τ that represents each pair of a clause and a variable as an evaluator module.

Table 1 is the truth table required for the (i, k)-evaluator for tautology and SAT problems. In the SAT case, S represents that the clause is satisfied, and $*$ represents that the value of the clause is not decided yet. Note that there exists a homomorphism that maps the truth table for the SAT problem to the truth table for the tautology problem: We may exchange type p and n, and treat U as S. Thus, if we encode the existence of v_i in C_k by a type-n evaluator, and $\neg v_i$ by a type-p evaluator, Ξ_τ evaluates all clauses autonomously. For the verification, the verifier now reaches p_{unsat} if all clauses returns S (which is encoded as U in the tautology problem). Therefore, the given formula is satisfiable if and only if there exists a conformation that reaches p_{unsat}, and we establish the following theorems.

Table 1 Outputs of (i, k)-evaluators for tautology and SAT problems

Tautology				SAT			
Type	Input	v_i	Output	Type	Input	v_i	Output
p	$*$	*T*	$*$	n	$*$	*T*	$*$
		F	U			*F*	S
	U	*T*	U		S	*T*	S
		F	U			*F*	S
n	$*$	*T*	U	p	$*$	*T*	S
		F	$*$			*F*	$*$
	U	*T*	U		S	*T*	S
		F	U			*F*	S
e	$*$	*T*	$*$	e	$*$	*T*	$*$
		F	$*$			*F*	$*$
	U	*T*	U		S	*T*	S
		F	U			*F*	S

p-evaluator represents that v_i is in the clause, n-evaluator represents that $\neg v_i$ is in that clause, and e-evaluator represents that neither of them is in the clause. In the evaluation of the clause, $*$ represents that the value is not yet decided, U represents that the clause is unsatisfied, and S represents that the clause is satisfied

Theorem 3 *Let* Ξ_τ *be the seedless NOS generated from a CNF formula* ϕ. *Then,* ϕ *is satisfiable if and only if there is a conformation of* Ξ_τ *that reaches the point* p_{unsat}.

Theorem 4 *ORP is NP-hard.*

5 Conclusions

We have designed a seedless NOS that solves the DNF tautology problem by checking if the NOS can fold a terminal conformation that reaches a designated point. We also have demonstrated the hardness of decision problems for OSs, including the OSEQP and ORP. Since this is the first attempt to exploit nondeterminism and seedlessness in the design of an OS, our future work includes applying nondeterminism and seedlessness to solve other problems. It is an open problem whether we can design an equivalent seedless OS from a given OS or not. Also, note that we map an instance of DNF formulas to an NOS that is unique to that input. This notion is called semi-uniformity (Pérez-Jiménez et al. 2003) compared to circuit uniformity (Borodin 1977), where we provide a computing device solely according to the length of the input. Introducing circuit uniformity to the design of OS is another open problem.

Acknowledgements We would like to thank the anonymous reviewers for the careful reading of the paper and many valuable suggestions.

Han was supported by the International Cooperation Program managed by NRF of Korea (2017K2A9A2A08000270), the Basic Science Research Program through NRF funded by MEST (2015R1D1A1A01060097) and the Yonsei University Future-leading Research Initiative of 2016. Kim was supported by an NRF grant funded by the Korean Government (NRF-2013-Global Ph.D. Fellowship Program). The work of S.S. was supported in part by the JST Program to Disseminate Tenure Tracking System, MEXT, Japan, no. 6F36, and by a JSPS Grant-in-Aid for Research Activity Start-up no. 15H06212 and for Young Scientists (A) no. 16H05854.

References

Borodin A (1977) On relating time and space to size and depth. SIAM J Comput 6(4):733–744

Frieda KL, Block SM (2012) Direct observations of cotranscriptional folding in an adenine riboswitch. Science 338(6105):397–400

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

Geary C, Meunier PE, Schabanel N, Seki S (2015) Efficient universal computation by greedy molecular folding. CoRR. arXiv:1508.00510

Geary C, Meunier PE, Schabanel N, Seki S (2016) Programming biomolecules that fold greedily during transcription. In:

Proceedings of the 41st international symposium on mathematical foundations of computer science, pp 43:1–43:14

Kleinberg J, Tardos É (2011) Algorithm design. Addison-Wesley, Reading

Lai D, Proctor JR, Meyer IM (2013) On the importance of cotranscriptional RNA structure formation. RNA 19:1461–1473

Pérez-Jiménez MJ, Romero-Jiménez A, Sancho-Caparrini F (2003) Complexity classes in models of cellular computing with membranes. Nat Comput 2(3):265–285

Rivas E, Eddy SR (1999) A dynamic programming algorithm for RNA structure prediction including pseudoknots. J Mol Biol 285(5):2053–2068

Xayaphoummine A, Bucher T, Isambert H (2005) Kinefold web server for RNA/DNA folding path and structure prediction including pseudoknots and knots. Nucleic Acids Res 33:W605–W610

Zuker M (2003) Mfold web server for nucleic acid folding and hybridization prediction. Nucleic Acids Res 31(13):3406–3415

Zuker M, Stiegler P (1981) Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Res 9(1):133–148