Theoretical Computer Science 696 (2017) 52–64

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Pseudoknot-generating operation ☆

Da-Jung Cho [a], Yo-Sub Han [a,*], Timothy Ng [b], Kai Salomaa [b]

[a] Department of Computer Science, Yonsei University, Seoul 120-749, Republic of Korea
[b] School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada

ARTICLE INFO

Article history:
Received 16 February 2016
Received in revised form 28 February 2017
Accepted 1 July 2017
Available online 8 July 2017
Communicated by N. Jonoska

Keywords:
Pseudoknots
Pseudoknot-generating operation
Closure and decision properties
Formal languages

ABSTRACT

A pseudoknot is a crucial intra-molecular structure formed primarily in RNA strands and closely related to important biological processes. This motivates us to define an operation that generates all pseudoknots from a given sequence and consider algorithmic and language theoretic properties of the operation. We design an efficient algorithm that decides whether or not a given string is a pseudoknot of a regular language L. Our algorithm runs in linear time if L is given by a deterministic finite automaton. We study closure and decision properties of the pseudoknot-generating operation. For DNA encoding applications, pseudoknot structures are undesirable. We give polynomial-time algorithms that check whether or not a regular language L contains a pseudoknot or a pseudoknot generated by some string of L. Furthermore, we show that the corresponding questions for context-free languages are undecidable.

1. Introduction

A ribonucleic acid (RNA) consists of four base types Adenine (A), Uracil (U), Guanine (G) and Cytosine (C), and has hydrogen bonds with the strongest complementary pairs A–U and C–G, which is called the *Watson–Crick pairing* [6]. Besides, in an RNA sequence, G–U might be paired as well—a wobble pair [27]. These paired bases lead an RNA to form secondary structures. An RNA secondary structure generally has stems that form a double helix with paired bases and various kinds of loops of unpaired bases as a structural motif, which then gives rise to well-known structures such as hairpin or pseudoknot. RNA secondary structures play an important role in cells and give insights to molecular evolution and function of RNA molecule [25]. A pseudoknot structure contains at least two stem-loops that occur in RNA with intramolecular base-pairing: Second half of one stem is embedded in between the two halves of another stem. (See Fig. 1 for an example of pseudoknot structure.)

Pseudoknot structures appear in many natural RNA molecules and are closely related with the ribosomal frameshifting that allows viruses to create many protein structures from a relatively small genome [10]. Since the ribosomal frameshifting affects on encoding protein and the pseudoknot structure gives a tertiary structure of molecule, it is a major topic of biomolecular computing to predict pseudoknot structures [4,10].

A *thermodynamic free energy* of RNA secondary structure is estimated from energy values of loops on a secondary structure. Note that an RNA secondary structure may have several loops—hairpins, bulges, interior loops, stacks and multi-

☆ A preliminary version appeared in *Proceedings of the 42nd International Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM 2016, LNCS **9587**, 241–252, Springer-Verlag, 2016.

* Corresponding author.
E-mail addresses: dajungcho@yonsei.ac.kr (D.-J. Cho), emmous@yonsei.ac.kr (Y.-S. Han), ng@cs.queensu.ca (T. Ng), ksalomaa@cs.queensu.ca (K. Salomaa).

http://dx.doi.org/10.1016/j.tcs.2017.07.001
0304-3975/

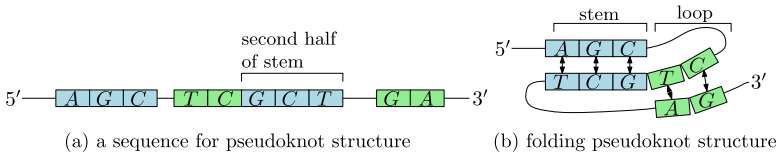


Fig. 1. A pseudoknot structure example: (a) A sequence contains a pseudoknot structure in which the second half of stem (blue box) exists between the two halves of another stem (green boxes). (b) A sequence folds into a pseudoknot. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

loops—and each loop has a corresponding energy function [19]. The thermodynamic free energy model of RNA secondary structure shows that an RNA secondary structure with the minimum-free energy is the most stable structure [29]. Researchers considered the minimum-free energy of RNA secondary structures and designed efficient dynamic-programming algorithms for RNA secondary structure prediction problem [2,22,29].

A *genus* is a taxonomic rank used in the biological classification and the pseudoknot structure is closely related to genus by characterizing the topological pairing of the RNA secondary structure [3]. A simple pseudoknot corresponds to genus 1 that can be drawn on a torus without crossing. The topological genus of RNA structures can be translated into an efficient dynamic-programming algorithm for the predicting RNA structures, and it leads researchers to consider the topological classification of RNA structures [23].

From a formal language viewpoint, several researchers [9,16,20,24] characterized the pseudoknot structure and suggested pseudoknot predicting algorithms. Given an input, the problem of predicting or aligning arbitrary pseudoknot structures is NP-hard [2,12]. The pseudoknot predicting problem over nearest-neighbor energy model that considers base pairing status and partners of its consecutive positions is non-approximable within any positive ratio [26]. Möhl et al. [20] presented an algorithm that computes the edit-distance of two RNA structures with arbitrary pseudoknots and showed that the algorithm is applicable in practice. Kari and Seki [16] formalized a particular case of pseudoknot structures over the Watson–Crick pairing and investigated its properties under formal language theory. Evans [9] proposed the first polynomial-time algorithm for finding maximum common substructures that include pseudoknots. Rinaudo et al. [24] generalized several RNA structures and presented an alignment algorithm based on the tree decomposition approach.

While most researchers considered the problem of predicting pseudoknot structures from a (long) sequence, we examine pseudoknot structures from a different angle: A sequence may be expanded (namely, append a new sequence to itself) to form a pseudoknot structure. We consider this process and define a new operation *pseudoknot-generating* operation that generates all pseudoknot structures (from now we just call *pseudoknots* in short) from a given sequence. The sequences resulting from the operation fold themselves into pseudoknots. In other words, the input string becomes a seed string to generate pseudoknots. Indeed, researchers in molecular biology deliberately generate pseudoknot structures—site-directed mutagenesis of pseudoknot—for understanding their structural features and interaction between hepatitis C virus (HCV) and frameshift mutation that causes the pseudoknot structure [5,18].

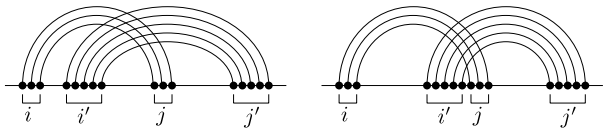
We establish the closure properties of the pseudoknot-generating operation on a string and present an algorithm that determines whether or not a string is a pseudoknot. We also study the closure properties of pseudoknot-generating operation on languages and examine several questions related to pseudoknots with respect to languages. From a biological view point, we can think of the pseudoknot-generating operation on a language as a procedure to generate all possible pseudoknots that may cause a mutation from a set of subsequences. In particular, we theoretically demonstrate that one can check whether or not two sets of DNA sequences contain common mutational seed sequences. Furthermore, we define the pseudoknot-freeness and investigate the decidability problem for pseudoknot-freeness for regular and context-free languages.

In Section 2, we recall some notation and define the pseudoknot-generating operation. We consider the pseudoknot-generating operation and design several algorithms for recognizing generated pseudoknots from strings and finite automata in Section 3. Then, we study closure and decision properties of the pseudoknot-generating operation, and investigate the pseudoknot-free languages in Section 4.

2. Preliminaries

Let Σ denote a finite alphabet of characters and Σ^* denote the set of all strings over Σ. A language over Σ is a subset of Σ^*. The symbol $\emptyset$ denotes the empty language and the symbol λ denotes the null string. Given a string $x = x_1 \cdots x_n$, $|x|$ is the number of characters in x, $x(i)$ denotes the ith character x_i of x and $x(i, j) = x_i x_{i+1} \cdots x_j$ is the substring of x from position i to position j, where $i \leq j$. Given two strings x and y in Σ^*, x is a *prefix* of y if there exists $z \in \Sigma^*$ such that $xz = y$ and x is a *suffix* of y if there exists $z \in \Sigma^*$ such that $zx = y$. Furthermore, x is said to be a *substring* or an *infix* of y if there are two strings u and v such that $uxv = y$.

A *finite automaton* (FA) A is specified by a tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $s \in Q$ is the start state and $F \subseteq Q$ is a set of final states. If F consists of a single state f, then we use f instead of $\{f\}$ for simplicity. Let $|Q|$ be the number of states in Q and $|\delta|$ be the number of transitions in δ. Then, the size of A is $|A| = |Q| + |\delta|$. For a transition $\delta(p, a) = q$ in A, we say that p has an *out-transition* and q has an *in-transition*. If $\delta(q, a)$ has a single element q', then we denote $\delta(q, a) = q'$ instead of $\delta(q, a) = \{q'\}$ for simplicity.



(a) simple pseudoknot (b) restricted pseudoknot

Fig. 2. An example of two types of pseudoknot on a sequence of length n: (a) a simple pseudoknot with two base-pairings (i, j) and (i', j'); (b) a restrict version of pseudoknot with two base-pairings (i, j) and (i', j') such that the first half of (i', j') is immediately followed by the second half of (i, j), where $0 \le i < i' < j < j' \le n$.

A string x over Σ is accepted by A if there is a labeled path from s to a final state such that this path spells out x. We call this path an *accepting path*. Then, the language $L(A)$ of A is the set of all strings spelled out by accepting paths in A. We say that a state of A is *useful* if it appears in an accepting path in A; otherwise, it is *useless*. Unless otherwise mentioned, in the following we assume that all states of A are useful.

A simple pseudoknot structure found in E. Coli transfer-messenger-RNA forms $v_1xv_2yv_3\theta(x)v_4\theta(y)v_5$, where $\theta(x)$ is a reverse-complement of x [16].[1] Then, we say that a string x has a pseudoknot if x has a substring w such that w is a simple pseudoknot.

We consider a restricted pseudoknot in which $v_3 = \lambda$—half of one stem is adjacent to half of another stem. (See Fig. 2 for an example of restricted pseudoknot.)

Given a string x, we define the restricted pseudoknot-generating operation

$$\mathbb{PK}_{\mathbb{R}}(x) = \{x_1x_2x_3x_1^Rx_4x_3^R \mid x = x_1x_2x_3 \text{ and } x_1, x_2, x_3, x_4 \in \Sigma^+\}.$$

Note that, for simplicity, we regard $\theta(w)$ as a reverse of w instead of a reverse-complement of w; namely $\theta(w) = w^R$.

For a language L,

$$\mathbb{PK}_{\mathbb{R}}(L) = \bigcup_{x \in L} \mathbb{PK}_{\mathbb{R}}(w).$$

We define the iterated operation of $\mathbb{PK}_{\mathbb{R}}$ to be, for $i \ge 0$,

$$\mathbb{PK}_{\mathbb{R}}^{(0)}(L) = L, \quad \mathbb{PK}_{\mathbb{R}}^{(i+1)}(L) = \mathbb{PK}_{\mathbb{R}}(\mathbb{PK}_{\mathbb{R}}^{i}(L)), \quad \mathbb{PK}_{\mathbb{R}}^{*}(L) = \bigcup_{i=0}^{\infty} \mathbb{PK}_{\mathbb{R}}^{i}(L).$$

In the following, we only consider restricted pseudoknots and call them simply pseudoknots unless we need to distinguish restricted pseudoknots from general pseudoknots. The reader may refer to Wood [28] for more knowledge in finite automata and formal languages.

3. Algorithms for recognizing generated pseudoknots

We first study the problem for checking whether or not a string $w = w_1w_2 \cdots w_n$ is a pseudoknot; namely, is $w = x_1x_2x_3x_1^Rx_4x_3^R$ for some $x_1, x_2, x_3, x_4 \in \Sigma^+$. The main idea of our approach is to check if there exists a substring $x_3x_1^R$ of w such that x_1 is a prefix and x_3^R is a suffix of w. A naive approach is to consider all possible substrings and check this condition. We design a better algorithm that checks the required condition more efficiently based on the Aho–Corasick algorithm [1]. (See Fig. 3.)

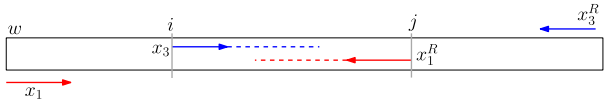


Fig. 3. A naive approach for checking whether or not w is a pseudoknot. For each substring $w(i, j)$, we check whether or not $w(i, j)$ is a catenation of x_3 and x_1^R for a prefix x_1 and a suffix x_3 of $w - w(i, j) = x_3x_1^R$—by comparing characters from both directions.

Before we describe the whole algorithm, we first present an algorithm that finds the shortest length of the matching prefix of the input pattern string with respect to the input for each index of the input. This algorithm is crucial for checking whether or not w is a pseudoknot.

Given a pattern string w and a text T, **Proc.** ShortestMatchingLength is a modified Aho–Corasick algorithm that finds the shortest length of the matching prefix of w at each index of T; if u the shortest matching prefix of w, then the reversal u^R of u appears as an infix of T. The two main differences from the original Aho–Corasick algorithm are

[1] In general, $\theta(x)$ does not necessarily give rise to a single string since there might be a wobble pairing [27]. However, we only consider the Watson–Crick pairing and assume that there is no wobble paring.

Procedure ShortestMatchingLength(w, T).

```
/* w is a length m pattern and T is a length n text                    */
Construct a DFA A = (Q, Σ, δ, 0, Q \ {0}) for w, where Q = {0, 1, ..., m}

/* construct the goto function 𝔾                                       */
𝔾(0, a ≠ w_1 ∈ Σ) ← 0
for i ← 0 to m − 1 do
    𝔾(i, w_{i+1}) ← i + 1

/* construct the failure function 𝔽 and the output function 𝕆          */
𝔽(1) ← 0
for i ← 1 to m do
    if 𝔾(i, a) = i+1 then
        v ← 𝔽(i)
        while 𝔾(v, a) ≠ ∅ do
            v ← 𝔽(v)
        𝔽(i+1) ← 𝔾(v, a)
        𝕆(i+1) ← min(𝕆(i+1), 𝕆(k))

/* read T using 𝔾, 𝔽, 𝕆                                               */
q ← 0
for i ← 1 to n do
    while 𝔾(q, T(i)) ≠ ∅ do
        q ← 𝔽(q)
    q = 𝔾(q, T(i))
    if 𝕆(q) ≠ ∅ then
        SML[q] ← 𝕆(q)
return SML
```

1. it receives only one string w as an input pattern and regards all prefixes of w as matching patterns
2. the output function $\mathbb{O}$ returns the shortest length of the matching pattern instead of reporting all matching patterns: $\mathbb{O}(i{+}1) \leftarrow \min(\mathbb{O}(i{+}1), \mathbb{O}(\mathbb{F}(i{+}1)))$.

It is easy to verify that **Proc.** ShortestMatchingLength runs in $O(m{+}n)$ time, where $m = |w|$ and $n = |T|$.

Now we design the whole algorithm that determines whether or not w is a pseudoknot using **Proc.** ShortestMatchingLength. First, we consider all prefixes of w up-to length $\frac{n}{2}$—candidates for being w_1 in the pseudoknot—and compute the set $w_p[i]$ of the shortest length of the matching prefix of each index using **Proc.** ShortestMatchingLength with $w = w_1 w_2 \cdots w_{\frac{n}{2}}$ and $T = w^R$. Next, we similarly consider all suffixes of w up-to length $\frac{n}{2}$—candidates for being w_3^R in the pseudoknot—and compute the set $w_s[i]$ of the shortest length of the matching suffix for each index $1 \le i \le n$ using **Proc.** ShortestMatchingLength with $w = w_{\frac{n}{2}+1} \cdots w_{n-1} w_n$ and $T = w$.

Algorithm 1: PseudoknotChecker.

```
Input: A string w of length n
Output: Y/N
x = w_1 w_2 ··· w_{n/2}                          /* the half prefix of w */
y = w_{n/2+1} ··· w_{n-1} w_n                    /* the half suffix of w */
w_p[n] = w_s[n] ← 0                              /* initialization: empty the sets */

w_p[n] ← ShortestMatchingLength(x, w^R)
w_s[n] ← ShortestMatchingLength(y^R, w)

for i ← 1 to n − 1 do
    if i < w_s[i] + w_p[i + 1] + 1 then
        return Y
return N
```

Fig. 4 is an example of running **Proc.** ShortestMatchingLength for a string w and obtain $w_p[n]$ and $w_s[n]$. In this example, because of $w_s[9]$ and $w_p[10]$, we know that w is a pseudoknot. However, a pair of $w_s[5]$ and $w_p[6]$ is invalid since, at index 6, w cannot have a prefix of size 6 ($= w_p[6]$). Similarly, a pair of $w_s[11]$ and $w_p[12]$ is invalid for checking the pseudoknot structure for w since, after index 11, w cannot have a $w_1^R w_4 w_3^R$, where $|w_3^R| = 6$. Algorithm 1 is a

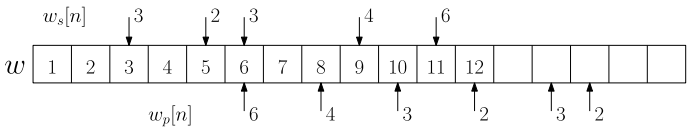


Fig. 4. An example of running **Proc.** ShortestMatchingLength for checking whether or not w is a pseudoknot.

pseudo-description of our algorithm for checking whether or not w is a pseudoknot. It is straightforward to verify that the algorithm runs in $O(n)$ time.

Lemma 3.1. *Given a string w of length n, we can determine whether or not w is a pseudoknot in $O(n)$ time.*

If w is a pseudoknot, then we can find all indices i of w such that $w(1,i) = x_1x_2x_3$ and $w(i+1,n) = x_1^R x_4 x_3^R$ from the algorithm. Let $I_{pk}(w)$ be the set of such indices.

Corollary 3.2. *Given two pseudoknot strings x and y, we can determine whether or not both $x, y \in \mathbb{PK}_{\mathbb{R}}(w)$ for a string w in linear-time in the size of x and y. We can also identify such w using $I_{pk}(x)$ and $I_{pk}(y)$ within the same runtime.*

We next consider a problem of determining whether or not w is in $\mathbb{PK}_{\mathbb{R}}(L(A))$ of a given FA A. Our approach is simple: we read w character by character with A and find all indices j of w when we enter a final state of A while reading w. Namely, $w(1,j) \in L(A)$. Let $I_p(w,A)$ be the set of such indices.

Lemma 3.3. *Given a string w and an FA A,*

$$w \in \mathbb{PK}_{\mathbb{R}}(L(A)) \text{ iff } I_{pk}(w) \cap I_p(w,A) \neq \emptyset.$$

Proof. We prove the statement for each direction separately:
($\Longleftarrow$) Since $I_{pk}(w) \cap I_p(w,A) \neq \emptyset$, there is an index i such that

- $w(1,i) = x_1x_2x_3$ and $w(i+1,n) = x_1^R x_4 x_3^R$ [from $I_{pk}(w)$]
- $w(1,i) \in L(A)$ [from $I_p(w,A)$].

This guarantees that $w \in \mathbb{PK}_{\mathbb{R}}(L(A))$ because A accepts $w(1,i)$.

($\Longrightarrow$) Since $w \in \mathbb{PK}_{\mathbb{R}}(L(A))$, we know that $w \in \mathbb{PK}_{\mathbb{R}}(u)$ for a string $u \in L(A)$. Note that u is a prefix of w. This implies that $|u| \in I_{pk}(w)$ as well as $|u| \in I_p(w,A)$. □

A pseudoknot string may have several different pseudoknots. Therefore, even if $w(1,i) = x_1x_2x_3$ and $w(i+1,n) = x_1^R x_4 x_3^R$ for an index i of w, $w(1,i)$ may not be accepted by A. This is why we have considered all possible indices in $I_{pk}(w)$. We now establish the following result based on the pseudoknot checking algorithm and Lemma 3.3.

Theorem 3.4. *Given a string w of size n and an FA A of size $m = |A|$, we can determine whether or not $w \in \mathbb{PK}_{\mathbb{R}}(L(A))$ in $O(mn)$ time. If A is a DFA, then the runtime becomes $O(n)$.*

Proof. It takes $O(n)$ time to compute $I_{pk}(w)$ and $O(mn)$ time to compute $I_p(w,A)$. If A is a DFA, then we can compute $I_p(w,A)$ in $O(n)$ time. □

Pseudoknots of RNA are closely related with the frameshifting mutation of protein expressions and commonly found in viral genomes, in particular influenza virus [8]. This leads researchers to consider the structural comparison among several sequences to find the common mutational pattern, in particular, pseudoknots [7]. Here, we investigate a necessary condition of $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$ for two strings x and y and show that it is decidable to check whether or not two strings have a common pseudoknot in $\mathbb{PK}_{\mathbb{R}}(x)$ and $\mathbb{PK}_{\mathbb{R}}(y)$.

Let x and y be two strings, where $|x| < |y|$. Fig. 5 shows that $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$ if and only if x is a prefix of y, and $y(|x|+1, |y|)$ ($= u$ in the figure) appears as an infix of $x(1, |x|-2)$—we consider $x(1, |x|-2)$ to ensure $t, v, z \in \Sigma^+$, if exists. There are two possible cases for being an infix as follows:

(a) Since u^R appears as a prefix of $x(1, |x|-2)$, we can select an arbitrary prefix t of $x(1, |x|-2)$ for y_1 as depicted in Fig. 5(a).
(b) Since u^R appears an infix (but not a prefix) of $x(1, |x|-2)$, we can select the prefix t of x_1 such that $x_1 = tu^R$ for y_1 as depicted in Fig. 5(b).

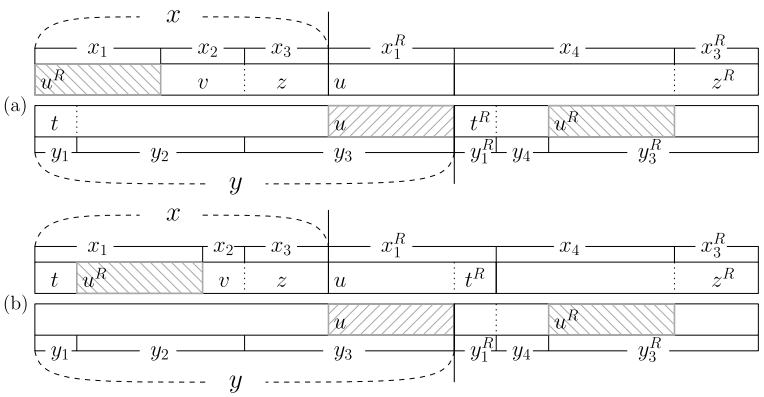


Fig. 5. An example of two strings x and y such that $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$. First, x is a prefix of y. Second, the longer part u (slanted line box in the figure) of y appears as (a) a prefix of x or (b) an infix (but not prefix) of x in the reversed form u^R, where $t, v, z \in \Sigma^+$.

We can check this in $O(|x|)$ time using the KMP algorithm [17] and, therefore, obtain the following result.

Lemma 3.5. *Given two strings x and y such that $|x| < |y|$, we can determine whether or not $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$ in $O(|x|)$ time.*

Proof. The proof is immediate from Fig. 5. □

We know if two strings have a common pseudoknot in $\mathbb{PK}_{\mathbb{R}}(x)$ and $\mathbb{PK}_{\mathbb{R}}(y)$. We next investigate the inclusion between $\mathbb{PK}_{\mathbb{R}}(x)$ and $\mathbb{PK}_{\mathbb{R}}(x)$ when $|x| < |y|$.

Lemma 3.6. *Given two strings x and y, if $|x| < |y|$, then it is impossible that $\mathbb{PK}_{\mathbb{R}}(x) \subset \mathbb{PK}_{\mathbb{R}}(y)$.*

Proof. First of all, x should be a prefix of y—otherwise, $\mathbb{PK}_{\mathbb{R}}(x) \not\subset \mathbb{PK}_{\mathbb{R}}(y)$. Let u be the longer part of y; namely $y = xu$. Then, u^R must appear as an infix of y in every index of x. If not, say when u^R is aligned between index $i - |u|$ and index i, then we can make $x_1 = x(1, i)$ and x_1^R is different from u of x. This follows that $\mathbb{PK}_{\mathbb{R}}(x) \not\subset \mathbb{PK}_{\mathbb{R}}(y)$.

Now since u^R must appear as an infix of y in every index of x, we know that x is a unary string. Moreover, this implies that y is also a unary string. Consider the following two unary strings: $x = a^3$ and $y = a^4$.

$$\mathbb{PK}_{\mathbb{R}}(a^3) = \{a^i \mid i \geq 6\}, \quad \mathbb{PK}_{\mathbb{R}}(a^4) = \{a^i \mid i \geq 7\}.$$

In other words, it is impossible that $\mathbb{PK}_{\mathbb{R}}(x) \not\subset \mathbb{PK}_{\mathbb{R}}(y)$ when $|x| < |y|$. □

Lemma 3.6 shows that if $|x| < |y|$, then $\mathbb{PK}_{\mathbb{R}}(x) \not\subset \mathbb{PK}_{\mathbb{R}}(y)$ always holds.

Given a string z, it is straightforward to verify that $\mathbb{PK}_{\mathbb{R}}(z)$ is regular and $\mathbb{PK}_{\mathbb{R}}(z)$ is infinite from the definition of the operation. We consider the case of applying the $\mathbb{PK}_{\mathbb{R}}$ operation on the resulting language several times, and prove that the iterated $\mathbb{PK}_{\mathbb{R}}$ does not preserve the regularity.

Theorem 3.7. *There exists a string z such that $\mathbb{PK}_{\mathbb{R}}^2(z)$ and $\mathbb{PK}_{\mathbb{R}}^*(z)$ are not regular.*

Proof. Choose $\Sigma = \{a, ¢, \$\}$ and $z = ¢a\$$. Now

$$\mathbb{PK}_{\mathbb{R}}(¢a\$) = \{¢a\$¢w\$ \mid w \in \Sigma^+\}.$$

Define

$$M = ¢a\$¢a^+\$¢a^+\$a^+¢.$$

We claim that

$$\mathbb{PK}_{\mathbb{R}}^2(¢a\$) \cap M = \{¢a\$¢a^m\$¢u\$a^m¢ \mid m \geq 1, u \in a^+\}. \tag{1}$$

All strings $¢a\$¢a^m\$$ are in $\mathbb{PK}_{\mathbb{R}}(¢a\$)$ and by choosing $x_1 = ¢$, $x_2 = a\$$ and $x_3 = ¢a^m\$$, we see that the inclusion from right to left in (1) holds.

To verify the converse inclusion, consider an arbitrary string $x = ¢a\$¢w\$$ in $\mathbb{PK}_{\mathbb{R}}(¢a\$)$. By writing $x = x_1x_2x_3$ we know that $x_1x_2x_3x_1^Rx_4x_3^R$ is in $\mathbb{PK}_{\mathbb{R}}^2(¢a\$)$ for all $x_4 \in \Sigma^+$.

If x_1 contains more than one symbol ¢, the string $x_1x_2x_3x_1^Rx_4x_3^R$ must contain at least 5 symbols ¢, because strings of M end with a ¢ and thus x_3^R must end with a symbol ¢. Since strings of M have 4 occurrences of the symbol ¢, we conclude that the only possibility where $x_1x_2x_3x_1^Rx_4x_3^R \in M$ is that $x_1 = ¢$.

Second, consider the possibility that $w = w_1¢w_2$ contains an occurrence of the symbol ¢ and we choose $x_3 = ¢w_2\$$. (We know that x_3 must begin with the symbol ¢.) Now, recalling that $x_1 = ¢$ must hold, the string $x_1x_2x_3x_1^Rx_4x_3^R$ looks as follow:

$$¢a\$¢w_1¢w_2\$¢u\$w_2^R¢, \quad u \in \Sigma^+.$$

Again we note that the string $x_1x_2x_3x_1^Rx_4x_3^R$ has 5 occurrences of the symbol ¢ and cannot be in M. Thus, we conclude that the only possibility where $x_1x_2x_3x_1^Rx_4x_3^R$ in M, is when $w \in a^+$ and we choose $x_3 = ¢w\$$. This gives the inclusion from left to right in (1).

Since $\mathbb{PK}_{\mathbb{R}}^2(¢a\$) \cap M$ is not regular, also $\mathbb{PK}_{\mathbb{R}}^2(¢a\$)$ must be nonregular. To verify the claim for iterated pseudoknot-generation, we note that

$$\mathbb{PK}_{\mathbb{R}}^*(¢a\$) \cap M = \mathbb{PK}_{\mathbb{R}}^2(¢a\$) \cap M.$$

This follows from the observation that for all $i \neq 2$, $\mathbb{PK}_{\mathbb{R}}^i(¢a\$) \cap M = \emptyset$. First we note that all strings of $\mathbb{PK}_{\mathbb{R}}(¢a\$)$ end with a symbol \$ and hence $\mathbb{PK}_{\mathbb{R}}(¢a\$)$ has no strings in M. Second we note that since all strings of $\mathbb{PK}_{\mathbb{R}}(¢a\$)$ contain two occurrences of ¢ and begin with the symbol ¢, it follows that all strings of $\mathbb{PK}_{\mathbb{R}}^2(¢a\$)$ contain at least three occurrences of ¢ (since the first symbol of a string must always be repeated). Any string $u \in \mathbb{PK}_{\mathbb{R}}^3(¢a\$) \cap M$ is obtained from a string $v \in \mathbb{PK}_{\mathbb{R}}^2(¢a\$)$ by applying the pseudo-knot operation. The string u must repeat the first symbol of v (which is a ¢). Furthermore, in order for u to be in M, u must end with ¢ which means that another occurrence of ¢ in v is repeated. Thus, u would need to have at least five occurrences of ¢ and cannot be in M. Exactly the same argument implies then that $\mathbb{PK}_{\mathbb{R}}^i(¢a\$) \cap M = \emptyset$ for all $i \geq 3$. □

4. Pseudoknot-generating operation on languages

We investigate the properties of pseudoknot on a set of strings. The pseudoknot operation on a string implies that we generate a pseudoknot family related by a common structural motif for pseudoknot. Note that a given string expands and becomes a pseudoknot string by the pseudoknot operation on a string. We extend this view point into languages in which a set of strings represents a set of all subsequences of a long RNA sequence. This implies that the pseudoknot operation on a language generates all possible pseudoknots that may partially occur as a mutation.

4.1. Closure and decision properties of the pseudoknot-generating operation

We first consider the closure property of the pseudoknot operation and determine whether or not two sets of pseudoknots on languages contain a common string.

Theorem 4.1. *Regular and context-free languages are not closed under* $\mathbb{PK}_{\mathbb{R}}$.

Proof. Consider a regular language over $\Sigma = \{a, b, \$_1, \$_2, \#, ¢_1, ¢_2\}$:

$$L = L(\$_1a^+\$_2\#¢_1b^+¢_2).$$

Let M be a set of strings that have two $\$_1$'s, two $\$_2$'s, two $¢_1$'s, two $¢_2$'s and one # in the following order: $\$_1, \$_2, \#, ¢_1, ¢_2, \$_2, \$_1, ¢_2, ¢_1$. Then, we have

$$M \cap \mathbb{PK}_{\mathbb{R}}(L) = \{\$_1a^n\$_2\#¢_1b^m¢_2\$_2a^n\$_1w¢_2b^m¢_1 \mid w \in \{a, b\}^+ \text{ and } n, m \geq 1\}.$$

Note that $M \cap \mathbb{PK}_{\mathbb{R}}(L)$ is not even context-free. □

Next, given regular languages L and R we consider the problem of checking whether or not there is a pseudoknot generated both by a string of L and a string of R. Note that, by Theorem 4.1, we know that $\mathbb{PK}_{\mathbb{R}}(L)$ need not be even context-free in general. This means that we cannot simply first produce a representation of the languages $\mathbb{PK}_{\mathbb{R}}(L)$ and $\mathbb{PK}_{\mathbb{R}}(R)$, respectively, and then check whether or not they have a non-empty intersection. Instead, our algorithm is based directly on finite automata for the original language L and R.

Let $A = (Q_A, \Sigma, \delta_A, s_A, F_A)$ and $B = (Q_B, \Sigma, \delta_B, s_B, F_B)$ be two FAs for L and R. Then, we first construct an FA $C = (Q_A \times Q_B, \Sigma, \delta_C, s_A \times s_B, F_A \times F_B)$ for $L(A) \cap L(B)$ by the standard Cartesian product, where

$$\delta_C((p, q), a) = \{(p', q') \mid p' \in \delta_A(p, a) \text{ and } q' \in \delta_B(q, a)\}.$$

Our algorithm is similar to the idea illustrated in Fig. 5: We check if there exists a pair of strings—say $x \in L(A)$, $y \in L(B)$ and $|x| < |y|$ (the other case is symmetric)—such that $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$. Since x is a prefix of y, there exists a path

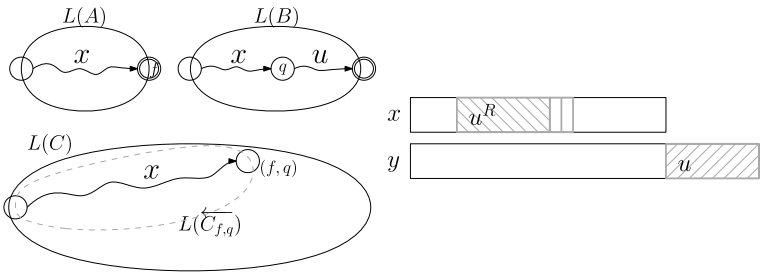


Fig. 6. An example of two FAs A and B such that $\mathbb{PK}_{\mathbb{R}}(L(A)) \cap \mathbb{PK}_{\mathbb{R}}(L(B)) \neq \emptyset$. Note that u^R is an infix of the string $x(2, |x|-2)$.

from s_B to a nonfinal state q that spells out x in B. We search for such paths in C. For each state (f, q) of C, where $f \in F_A, q \in Q_B$, we define two FAs as follows:

- $\overleftarrow{C_{f,q}} = (Q_A \times Q_B, \Sigma, \delta_C, s_A \times s_B, \{(f, q)\})$; in other words, (f, q) is the only final state of C.
- $\overrightarrow{B_q} = (Q_B, \Sigma, \delta_B, q, F_B)$; in other words, q is the new start state of B.

Lemma 4.2. *There exists a state (f, q) of C such that*

$$L(\overleftarrow{C_{f,q}}) \quad \cap \quad \Sigma^* \cdot (L(\overrightarrow{B_q})^R \cdot \Sigma^2) \cdot \Sigma^* \quad \neq \quad \emptyset$$

or a state (p, f') of C such that

$$L(\overleftarrow{C_{p,f'}}) \quad \cap \quad \Sigma^* \cdot (L(\overrightarrow{A_p})^R \cdot \Sigma^2) \cdot \Sigma^* \quad \neq \quad \emptyset$$

if and only if $\mathbb{PK}_{\mathbb{R}}(L(A)) \cap \mathbb{PK}_{\mathbb{R}}(L(B)) \neq \emptyset$.

Proof. We only prove the (f, q) case. The other case is symmetric.
($\Longleftarrow$) Since $\mathbb{PK}_{\mathbb{R}}(L(A)) \cap \mathbb{PK}_{\mathbb{R}}(L(B)) \neq \emptyset$, there are two strings $x \in L(A)$ and $y \in L(B)$ such that $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$ and x is a prefix of y. This implies that there is a state (f, q) in C reached from the start state (s_A, s_B) by reading x. Now since x and y can be aligned as in Fig. 6, $u \in L(\overrightarrow{B_q})$ and u^R is an infix of x. Therefore, x is in both $L(\overleftarrow{C_{f,q}})$ and $\Sigma^* \cdot (L(\overrightarrow{B_q})^R \cdot \Sigma^2) \cdot \Sigma^*$.
($\Longrightarrow$) From the assumption, we have $x \in L(\overleftarrow{C_{f,q}}) \cap \Sigma^* \cdot (L(\overrightarrow{B_q})^R \cdot \Sigma^2) \cdot \Sigma^*$ and $u \in L(\overrightarrow{B_q})$, where u^R is an infix of $x(2, |x|-2)$. Then, as we have illustrated in Fig. 5, we can decompose both $x \in L(A)$ and $xu = y \in L(B)$ into $x_1x_2x_3$ and $y_1y_2y_3$, respectively, where both decompositions make the same pseudoknot $x_1x_2x_3x_1^Rx_4x_3^R = y_1y_2y_3y_1^Ry_4y_3^R$. This guarantees that $\mathbb{PK}_{\mathbb{R}}(L(A)) \cap \mathbb{PK}_{\mathbb{R}}(L(B)) \neq \emptyset$. □

Once we have an intersection FA C, there are at most $|Q_A||Q_B|$ states in the form of (f, q) or (p, f'). Then, for each state, say (f, q), we need to check whether or not $L(\overleftarrow{C_{f,q}}) \cap \Sigma^* \cdot (L(\overrightarrow{B_q})^R \cdot \Sigma^2) \cdot \Sigma^*$ is empty. Since the size of $C_{f,q}$ is at most $|A||B|$ and the size of $\overrightarrow{B_q}$ is at most $|B|$, it takes $O(|A||B|^2)$ time. Therefore, in the worst-case, the total runtime is

$$O(n^2) \text{ (the number of states)} \times O(n^6) \text{ (intersection test)} = O(n^8),$$

where n is the maximum number of states between A and B.

Theorem 4.3. *Given two FAs A and B, we can determine whether or not*

$$\mathbb{PK}_{\mathbb{R}}(L(A)) \cap \mathbb{PK}_{\mathbb{R}}(L(B)) \neq \emptyset$$

in polynomial-time.

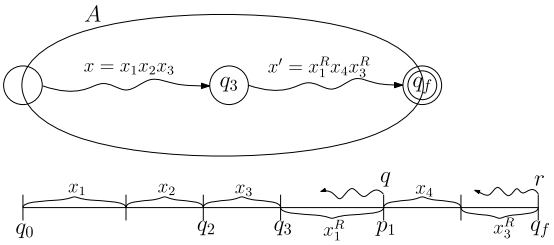
Often we need to verify if there exists a pseudoknot in the input set—a set of strings. In biology, an RNA sequence might first fold into non-pseudoknot, and then form a more complex structure including pseudoknots. According to this phenomenon, Jabbari and Condon [11] considered non-pseudoknots for predicting pseudoknots capturing all possible pre-structures of pseudoknots.

When an input set has a finite number of elements, we may check them one by one. However, if the set is infinite, then we need a better algorithm. We consider this problem when the set is a regular language. Before we present an algorithm, we define the inverse restricted pseudoknot-generating operation $\mathbb{PK}_{\mathbb{R}}^{-1}$ to be

$$\mathbb{PK}_{\mathbb{R}}^{-1}(w) = \{x_1x_2x_3 \mid w \in x_1x_2x_3x_1^Rx_4x_3^R, x_1, x_2, x_3, x_4 \in \Sigma^+\}.$$

Lemma 4.4. *Let A be an NFA. Then there exists an NFA A' such that*

$$L(A') = \mathbb{PK}_{\mathbb{R}}^{-1}(L(A)).$$



Proof. Let $A = (Q, \Sigma, \delta, q_0, F)$ be an NFA. We construct an NFA $A' = (Q', \Sigma, \delta', I', F')$ that recognizes the set of pseudoknot generators for all pseudoknots in $L(A)$. The state set $Q' \subseteq Q \times (Q \cup \{q_\alpha\})^2 \times Q^4$ is a 7-tuple $(p, q, r, p_1, q_2, q_3, q_f)$, where

- p simulates the computation of the input string in A,
- q simulates the computation of x_1^R in A in reverse,
- r simulates the computation of x_3^R in A in reverse,
- p_1 denotes the state in which the computation of x_1^R in A ends,
- q_2 denotes the state from which the computation of x_3 in A begins,
- q_3 denotes the state in which the computation of x_3 ends and the computation of x_1^R begins,
- $q_f \in F$ denotes the state in which the computation of x_3^R ends.

The state q_α serves as a placeholder for computations on x_1^R and x_3^R when they are inactive. The set of start states I' is defined by

$$I' = \{(q_0, p_1, q_\alpha, p_1, q_2, q_3, q_f) \in Q'\},$$

where q_0 is the start state of A. The set of final states F' is defined by

$$F' = \{(q_3, q_\alpha, p_2, p_1, q_2, q_3, q_f) \in Q' \mid (\exists w \in \Sigma^*) p_2 \in \delta(w, p_1)\}.$$

The transition function δ' is defined for $a \in \Sigma$ and $p, q, r \in Q$ as follows:

- $\delta'((p, q, q_\alpha, p_1, q_2, q_3, q_f), a) = \{(s, t, q_\alpha, p_1, q_2, q_3, q_f) \mid s \in \delta(p, a) \text{ and } q \in \delta(t, a)\}$;
- $\delta'((p, q_3, q_\alpha, p_1, q_2, q_3, q_f), a) = \{(s, q_\alpha, q_\alpha, p_1, q_2, q_3, q_f) \mid s \in \delta(p, a)\}$;
- $\delta'((p, q_\alpha, q_\alpha, p_1, q_2, q_3, q_f), a) = \{(s, q_\alpha, q_\alpha, p_1, q_2, q_3, q_f) \mid s \in \delta(p, a)\}$;
- $\delta'((p, q_\alpha, q_\alpha, p_1, q_2, q_3, q_f), a) = \{(q_2, q_\alpha, q_f, p_1, q_2, q_3, q_f) \mid q_2 \in \delta(p, a)\}$;
- $\delta'((p, q_\alpha, r, p_1, q_2, q_3, q_f), a) = \{(s, q_\alpha, t, p_1, q_2, q_3, q_f) \mid s \in \delta(p, a) \text{ and } r \in \delta(t, a)\}$.

All other transitions are undefined.

The machine A' begins by nondeterministically selecting four states and are stored as the final four components of the state. The operation of A' on an input string w occurs in three phases, decomposing the string w into three parts $x_1x_2x_3$.

In the first phase, A' reads x_1 by simulating the computation of w in A from the start state q_0. The machine simultaneously simulates a computation of w in A from the state p_1, but in reverse. If x_1^R is not a substring which ends in state p_1 of some string accepted by A, then the computation fails. The machine stops simulating x_1^R when the simulation of x_1^R reaches the state q_3. If q_3 is not reached by the simulation of x_1^R, the computation terminates. Once q_3 is reached, the second component goes to state q_α to denote that the simulation of x_1^R is tentatively successful.

In the second phase, A' reads x_2 by continuing to read w until the first component reaches the state q_2. Once q_2 is reached, the third phase begins and the simulation of x_3^R begins in the state q_f, which is some final state of A. Again, if x_3^R is not a suffix of some string accepted by A, then the computation fails. Once the computation of w ends, we must check which states the computation has terminated in.

The simulation of $w = x_1x_2x_3$ on A should have ended with state q_3 in the first component as it is the state in which the computation of x_1^R begins. The simulation of x_3^R in reverse in A in the third component must end in a state p_2 which is reachable from the state p_1, which is the state in which the computation of x_1^R ends. The state p_2 can be reachable from p_1 by any string $x_4 \in \Sigma^+$. □

Corollary 4.5. *Given an NFA A, we can determine if A accepts a pseudoknot.*

Proof. We construct A' as in Lemma 4.4. Now A accepts at least one pseudoknot if and only if $L(A') \neq \emptyset$. □

Note that it is decidable to determine whether or not a given regular language contains a pseudoknot. Here, we contrast the result of Corollary 4.5 by showing that it is undecidable whether or not a context-free language contains a pseudoknot. We use a reduction from the *Post Correspondence Problem* (PCP) [21]. Recall that an instance of PCP consists of two lists of strings $((u_1, \ldots, u_n), (v_1, \ldots, v_n))$, $u_i, v_i \in \Sigma^*$, $1 \leq i \leq n$, and a solution of this instance is a sequence of integers $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$. It is well known that PCP is unsolvable [21].

Proposition 4.6. *For a given context-free language L, it is undecidable whether or not L contains a pseudoknot.*

Proof. Let $I_{\text{PCP}} = ((u_1, \ldots, u_n), (v_1, \ldots, v_n))$, where $u_i, v_i \in \{a, b\}^*$, $1 \leq i \leq n$, be an arbitrary instance of PCP. Choose

$$\Sigma = \{a, b, 1, \ldots, n, \$, \$', \#, \#', \%\}.$$

Now define

$$L_1 = \{\$i_k i_{k-1} \cdots i_1 \$' \% \# u_{i_1} u_{i_2} \cdots u_{i_k} \#' \$' j_1 j_2 \cdots j_\ell \$ \% \#' v_{j_\ell}^R v_{j_{\ell-1}}^R \cdots v_{j_1}^R \# \mid k, \ell \geq 1, i_1, \ldots, i_k, j_1, \ldots, j_\ell \in \{1 \ldots, n\}\}.$$

It is easy to verify that L_1 is context-free since the language L_1 is the catenation of two linear context-free languages.

We verify that L_1 contains a pseudoknot if and only if I_{PCP} has a solution. ($\Longleftarrow$) For the "if" direction assume that the sequence $i_1, \ldots, i_k$ is a solution of I_{PCP}. Now L_1 contains the string

$$w_1 = \$i_k i_{k-1} \cdots i_1 \$' \% \# u_{i_1} u_{i_2} \cdots u_{i_k} \#' \$' i_1 i_2 \cdots i_k \$ \% \#' v_{i_k}^R v_{i_{k-1}}^R \cdots v_{i_1}^R \#.$$

By selecting $x_1 = \$i_k i_{k-1} \cdots i_1 \$'$, $x_2 = x_4 = \%$, $x_3 = \#u_{i_1} u_{i_2} \cdots u_{i_k} \#'$, and recalling that $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$ we note that $w_1 = x_1 x_2 x_3 x_1^R x_4 x_3^R$, i.e., w_1 is a pseudoknot.

($\Longrightarrow$) Conversely, if L contains a pseudoknot, then PCP has a solution. We assume that some string $w \in L_1$ can be written in a form $w = x_1 x_2 x_3 x_1^R x_4 x_3^R$, where $x_1, x_2, x_3, x_4 \in \Sigma^+$. Since w is in L_1 it must be of the following general form:

$$w = \$i_k i_{k-1} \cdots i_1 \$' \% \# u_{i_1} u_{i_2} \cdots u_{i_k} \#' \$' j_1 j_2 \cdots j_\ell \$ \% \#' v_{j_\ell}^R v_{j_{\ell-1}}^R \cdots v_{j_1}^R \#,$$

$1 \leq i_1, \ldots, i_k, j_1, \ldots, j_\ell \leq n$. The string w begins with the symbol \$ and contains exactly two occurrences of \$. This means that in the decomposition of w the substring occurrence x_1^R must end with the second occurrence of \$. Similarly since w ends with symbol # and contains exactly two occurrences of # the substring occurrence x_3 must begin with the first symbol #. Now if x_3 does not contain the first symbol #', this would need to be part of x_1^R which is clearly impossible. Symmetrically, if the second \$' is not part of x_1^R, this would need to be part of x_3 which is impossible (since the following substring x_3^R cannot have symbols \$').

Combining these two observations we have that the only possibility is that $x_1^R = \$' j_1 j_2 \cdots j_\ell \$$ and $x_3 = \#u_{i_1} u_{i_2} \cdots u_{i_k} \#'$. With these choices then x_1 must be the substring $\$i_k i_{k-1} \cdots i_1 \$'$ and x_3^R the substring $\#' v_{j_\ell}^R v_{j_{\ell-1}}^R \cdots v_{j_1}^R \#$.

It follows that $k = \ell$ must hold and $i_y = j_y$, $y = 1, \ldots, k$. Furthermore, the equations for x_3 and x_3^R give $u_{i_1} u_{i_2} \cdots u_{i_k} = v_{i_1} v_{i_2} \cdots v_{i_k}$ and $i_1, \ldots, i_k$ is then a solution for the instance I_{PCP}. □

4.2. Pseudoknot-free languages

Analogously with the definition of restricted code classes, such as prefix- or suffix-free codes [13], we define that a language L is $\mathbb{PK}_{\mathbb{R}}$-free (informally just pseudoknot-free) if no string of L is a pseudoknot generated by another string of L.

Definition 4.7. We define a language L to be $\mathbb{PK}_{\mathbb{R}}$-free if $L \cap \mathbb{PK}_{\mathbb{R}}(L) = \emptyset$.

In DNA coding applications, pseudoknots are in general undesirable because they can result in undesired bonds in DNA sequences [14,15]. This means that if we can efficiently check the property of $\mathbb{PK}_{\mathbb{R}}$-freeness, it might be worthwhile to add a preprocessing stage for predicting pseudoknot-structures. Note that some approaches for prediction pseudoknots consider also "non-pseudoknots" because an RNA sequence can fold a non-pseudoknot to form a pseudoknot.

We consider the case when L is regular and show that we can decide whether or not L is $\mathbb{PK}_{\mathbb{R}}$-free. We use a similar construction for constructing an FA for $\mathbb{PK}_{\mathbb{R}}^{-1}$.

Lemma 4.8. *Let A be an FA. Then there exists an NFA A' that accepts a set of strings $u = u_1 u_2 u_3$ such that $u_1^R u_4 u_3^R u_1 u_2 u_3 \in L(A)$.*

Proof. Let $A = (Q, \Sigma, \delta, q_0, F)$ be an FA for L. We construct an NFA $A' = (Q', \Sigma, \delta', I', F')$ that recognizes the set of strings $u = u_1 u_2 u_3$ such that $u_1^R u_4 u_3^R u_1 u_2 u_3$ is in L, where $u_1, u_2, u_3, u_4 \in \Sigma^+$.

The state set $Q' \subseteq Q \times (Q \cup \{q_\alpha\})^2 \times Q^3$ is a 6-tuple (p, q, r, p_1, q_2, q_f), where

- p simulates the computation of the string $u = u_1u_2u_3$ from a final state in A,
- q simulates the computation of u_1^R in A in reverse,
- r simulates the computation of u_3^R in A in reverse,
- p_1 denotes the state in which the computation of u_1^R in A ends,
- q_2 denotes the state from which the computation of u_3 in A begins,
- $q_f \in F$ denotes the state in which the computation of u_3^R ends.

The state q_α serves as a placeholder for computations on u_1^R and u_3^R when they are inactive. The set I' of start states is defined by

$$I' = \{(f, p_1, q_\alpha, p_1, q_2, q_f) \in Q'\},$$

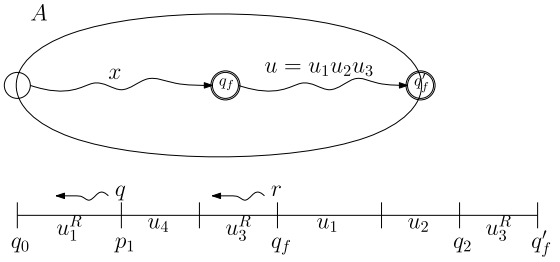
where f is a final state of A. The set F' of final states is defined by

$$F' = \{(q_f', q_\alpha, p_2, p_1, q_2, q_f) \in Q' \mid p_2 \in \delta(p_1, w) \text{ for some } w \in \Sigma^+ \text{ and } q_f' \in F\}.$$

The transition function δ' is defined for $a \in \Sigma$ and $p, q, r \in Q$ as follows:

- $\delta'((p, q, q_\alpha, p_1, q_2, q_f), a) = \{(s, t, q_\alpha, p_1, q_2, q_f) \mid s \in \delta(p, a) \text{ and } q \in \delta(t, a)\}$;
- $\delta'((p, q_0, q_\alpha, p_1, q_2, q_f), a) = \{(s, q_\alpha, q_\alpha, p_1, q_2, q_f) \mid s \in \delta(p, a)\}$;
- $\delta'((p, q_\alpha, q_\alpha, p_1, q_2, q_f), a) = \{(s, q_\alpha, q_\alpha, p_1, q_2, q_f) \mid s \in \delta(p, a)\}$;
- $\delta'((p, q_\alpha, q_\alpha, p_1, q_2, q_f), a) = \{(q_2, q_\alpha, q_f, p_1, q_2, q_f) \mid q_2 \in \delta(p, a)\}$;
- $\delta'((p, q_\alpha, r, p_1, q_2, q_f), a) = \{(s, q_\alpha, t, p_1, q_2, q_f) \mid s \in \delta(p, a) \text{ and } r \in \delta(t, a)\}$.

All other transitions are undefined.



The FA A' begins by nondeterministically selecting three states, which are stared as the final three components of the state. The operation of A' on a string u occurs in three phases, decomposing the string u into three parts $u_1u_2u_3$.

In the first phase, A' reads u_1 by simulating the computation of u in A from a final state f. The FA simultaneously simulates a computation of u in A from the state p_1, but in reverse. If u_1^R is not a substring which ends in state p_1 of some string accepted by A, then the computation fails. If the simulation of u_1^R does not each the start state q_0, then the computation terminates. If the simulation reaches q_0, then either it may continue the computation, or set the second component as state q_α to denote that the simulation of u_1^R is tentatively successful.

In the second phase, A' reads u_2 by continuing to read u until the first component reaches the state q_2. Once q_2 is reached, the third phase begins and the simulation of u_3^R begins in the state q_f, which is some final state of A. Again, if x_3^R is not a suffix of some string accepted by A, then the computation fails. Once the computation of u ends, we must check which states the computation has terminated in.

The simulation of $u = u_1u_2u_3$ on A should have ended in a final state q_f' in the first component. The simulation of u_3^R in reverse in A in the third component must end in a state p_2 that is reachable from the state p_1, which is the state in which the computation of u_1^R ends. The state p_2 can be reachable from p_1 by any string $u_4 \in \Sigma^+$. □

The construction of Lemma 4.8 guarantees that $L(A')$ accepts a string $u = u_1u_2u_3$ if and only if there exists a string $x = u_1^Ru_4u_3^Ru_1u_2u_3 \in L(A) \cap \mathbb{PK}_\mathbb{R}(L(A))$ for some strings $u_1, u_2, u_3, u_4 \in \Sigma^+$.

Theorem 4.9. *Given an FA A, we can determine whether or not $L(A)$ is $\mathbb{PK}_\mathbb{R}$-free in polynomial-time.*

Here, we observe that deciding $\mathbb{PK}_\mathbb{R}$-freeness of a context-free language is undecidable based on Proposition 4.6.

Theorem 4.10. *For a given context-free language L it is undecidable whether or not L is $\mathbb{PK}_\mathbb{R}$-free.*

Proof. Let $I_{\mathrm{PCP}} = ((u_1, \ldots, u_n), (v_1, \ldots, v_n))$, $u_i, v_i \in \{a, b\}^*$, $1 \le i \le n$, be an instance of PCP. Let Σ and L_1 be as in the proof of Proposition 4.6. Define

$$L_{\mathrm{add}} = \{\$i_k i_{k-1} \cdots i_1 \$'\%\# u_{i_1} u_{i_2} \cdots u_{i_k} \#' \mid k \ge 1,\ 1 \le i_1, \ldots, i_n \le n\},$$

and let $L_2 = L_1 \cup L_{\mathrm{add}}$. Since L_1 and L_{add} are context-free, so is L_2.

The proof of Proposition 4.6 verifies that the PCP instance I_{PCP} has a solution if and only if L_1 contains a pseudoknot, and from the same argument it follows that always when L_1 contains a pseudoknot $x_1 x_2 x_3 x_1^R x_4 x^R$, $x_1, x_2, x_3, x_4 \in \Sigma^+$, it must be the case the $x_1 x_2 x_3 \in L_{\mathrm{add}} (\subseteq L_2)$. Note that the strings of L_{add} are "the first halves" of strings of L_1. The strings in L_{add} are never themselves pseudoknots because they begin with a \$ and contain no other occurrences of \$.

Combining these observations it follows that L_2 is not $\mathbb{PK}_{\mathbb{R}}$-free if and only if I_{PCP} has a solution. □

5. Conclusions

We have considered an RNA structure called pseudoknot and specific phenomenon in which a sequence expands itself and forms a pseudoknot. We have defined a restricted version of the pseudoknot-generating operation: For a string x, $\mathbb{PK}_{\mathbb{R}}(x)$, roughly speaking, consists of all possible continuations of x that can fold back onto x to form a pseudoknot.

We have investigated (closure-)properties of pseudoknot-generating operation on a string and designed linear-time algorithm for determining whether or not given string is a pseudoknot. We have shown that for two strings x and y, it is decidable whether or not $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$. Moreover, we have examined the pseudoknot-generating operation on languages, and showed that regular and context-free languages are not closed under pseudoknot-generating. On the other hand, we have established that given two FAs A and B, it is decidable whether or not $\mathbb{PK}_{\mathbb{R}}(L(A)) \cap \mathbb{PK}_{\mathbb{R}}(L(B)) \neq \emptyset$ in polynomial-time in the size of A and B. Furthermore, we have shown that it is decidable whether or not a given regular language is $\mathbb{PK}_{\mathbb{R}}$-free in polynomial-time. However, it is undecidable to determine whether or not a given context-free language is $\mathbb{PK}_{\mathbb{R}}$-free.

Acknowledgements

We wish to thank the referees for the careful reading of the paper and useful suggestions including relevant references [3,26].

Cho and Han were supported by the Basic Science Research Program through NRF funded by MEST (2015R1D1A1A01060097), the Yonsei University Future-leading Research Initiative of 2016 and the IITP grant funded by the Korea government (MSIP) (R0124-16-0002), and Ng and Salomaa were supported by the Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

References

[1] A.V. Aho, M.J. Corasick, Efficient string matching: an aid to bibliographic search, Commun. ACM 18 (6) (1975) 333–340.
[2] T. Akutsu, Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots, Discrete Appl. Math. 104 (1) (2000) 45–62.
[3] M. Bon, G. Vernizzi, H. Orland, A. Zee, Topological classification of RNA structures, J. Mol. Biol. 379 (4) (2008) 900–911.
[4] I. Brierley, P. Digard, S.C. Inglis, Characterization of an efficient coronavirus ribosomal frameshifting signal: requirement for an RNA pseudoknot, Cell 57 (4) (1989) 537–547.
[5] M. Chamorro, N. Parkin, H.E. Varmus, An RNA pseudoknot and an optimal heptameric shift site are required for highly efficient ribosomal frameshifting on a retroviral messenger RNA, Nat. Acad. Sci. 89 (2) (1992) 713–717.
[6] R.M. Dirks, M. Lin, E. Winfree, N.A. Pierce, Paradigms for computational nucleic acid design, Nucleic Acids Res. 32 (4) (2004) 1392–1403.
[7] G. Doose, D. Metzler, Bayesian sampling of evolutionarily conserved RNA secondary structures with pseudoknots, Bioinformatics 28 (17) (2012) 2242–2248.
[8] Z. Du, D.W. Hoffman, An NMR and mutational study of the pseudoknot within the gene 32 mRNA of bacteriophage t2: insights into a family of structurally related RNA pseudoknots, Nucleic Acids Res. 25 (6) (1997) 1130–1135.
[9] P.A. Evans, Finding common RNA pseudoknot structures in polynomial time, J. Discrete Algorithms 9 (4) (2011) 335–343.
[10] D.P. Giedroc, C.A. Theimer, P.L. Nixon, Structure, stability and function of RNA pseudoknots involved in stimulating ribosomal frameshifting, J. Mol. Biol. 298 (2) (2000) 167–185.
[11] H. Jabbari, A. Condon, A fast and robust iterative algorithm for prediction of RNA pseudoknotted secondary structures, BMC Bioinform. 15 (1) (2014) 147.
[12] T. Jiang, G. Lin, B. Ma, K. Zhang, A general edit distance between RNA structures, J. Comput. Biol. 9 (2) (2002) 371–388.
[13] H. Jürgensen, S. Konstantinidis, Codes, in: Word, Language, Grammar, in: Handbook of Formal Languages, vol. 1, 1997, pp. 511–607.
[14] L. Kari, S. Konstantinidis, S. Kopecki, Transducer descriptions of DNA code properties and undecidability of antimorphic problems, in: Proceedings of the 17th International Workshop on Descriptional Complexity of Formal Systems, 2015, pp. 141–152.
[15] L. Kari, K. Mahalingam, Watson–Crick palindromes in DNA computing, Nat. Comput. 9 (2) (2010) 297–316.
[16] L. Kari, S. Seki, On pseudoknot-bordered words and their properties, J. Comput. System Sci. 75 (2) (2009) 113–121.
[17] D.E. Knuth, J.H. Morris Jr., V.R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (2) (1977) 323–350.
[18] B. Marczinke, R. Fisher, M. Vidakovic, A.J. Bloys, I. Brierley, Secondary structure and mutational analysis of the ribosomal frameshift signal of rous sarcoma virus, J. Mol. Biol. 284 (2) (1998) 205–225.
[19] D.H. Mathews, J. Sabina, M. Zuker, D.H. Turner, Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure, J. Mol. Biol. 288 (5) (1999) 911–940.
[20] M. Möhl, S. Will, R. Backofen, Fixed parameter tractable alignment of RNA structures including arbitrary pseudoknots, in: Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching, 2008, pp. 69–81.

[21] E. Post, A variant of a recursively unsolvable problem, Bull. Amer. Math. Soc. 52 (1946) 264–268.
[22] B. Rastegari, A. Condon, Parsing nucleic acid pseudoknotted secondary structure: algorithm and applications, J. Comput. Biol. 14 (1) (2007) 16–32.
[23] C.M. Reidys, F.W.D. Huang, J.E. Andersen, R.C. Penner, P.F. Stadler, M.E. Nebel, Topology and prediction of RNA pseudoknots, Bioinformatics 27 (8) (2011) 1076–1085.
[24] P. Rinaudo, Y. Ponty, D. Barth, A. Denise, Tree decomposition and parameterized algorithms for RNA structure-sequence alignment including Tertiary interactions and pseudoknots, in: Proceedings of the 12th International Workshop on Algorithms in Bioinformatics, 2012, pp. 149–164.
[25] A.A. Saraiya, T.N. Lamichhane, C.S. Chow, J.S. Jr, P.R. Cunningham, Identification and role of functionally important motifs in the 970 loop of escherichia coli 16s ribosomal RNA, J. Mol. Biol. 376 (3) (2008) 645–657.
[26] S. Sheikh, R. Backofen, Y. Ponty, Impact of the energy model on the complexity of RNA folding with pseudoknots, in: Proceedings of the 23rd Annual Symposium on Combinatorial Pattern Matching, 2012, pp. 321–333.
[27] E. Westhof, V. Fritsch, RNA folding: beyond Watson–Crick pairs, Structure 8 (3) (2000) R55–R65.
[28] D. Wood, Theory of Computation, John Wiley & Sons, Inc., New York, NY, 1987.
[29] M. Zuker, P. Stiegler, Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information, Nucleic Acids Res. 9 (1) (1981) 133–148.