Generalizations of Code Languages with Marginal Errors

Yo-Sub Han[1(✉)], Sang-Ki Ko[1], and Kai Salomaa[2]

[1] Department of Computer Science, Yonsei University, 50, Yonsei-Ro, Seoul, Seodaemun-Gu 120-749, Republic of Korea
{emmous,narame7}@cs.yonsei.ac.kr
[2] School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Abstract. We study k-prefix-free, k-suffix-free and k-infix-free languages that generalize prefix-free, suffix-free and infix-free languages by allowing marginal errors. For example, a string x in a k-prefix-free language L can be a prefix of up to k different strings in L. Namely, a code (language) can allow some marginal errors. We also define finitely prefix-free languages in which a string x can be a prefix of finitely many strings. We present efficient algorithms that determine whether or not a given regular language is k-prefix-free, k-suffix-free or k-infix-free, and analyze their runtime. Lastly, we establish the undecidability results for (linear) context-free languages.

Keywords: Codes · Marginal errors · Decision algorithms · Undecidability · Regular languages · Context-free languages

1 Introduction

Codes are useful in many areas including information processing, data compression, cryptography and information transmission [2,12]. Many researchers have studied various codes such as *prefix codes*, *suffix codes* and *infix codes*. Since a code is a set of strings—a language, a code property defines a subset of languages preserving the corresponding property. For instance, for regular langauges, the prefix-freeness defines a proper subset of regular languages, *prefix-free regular languages*. Recently, Kari et al. [13] considered the problem of deciding whether or not a given regular language is maximal with respect to certain combined types of code properties.

There are different applications based on different properties subfamilies [1,3,4,7,9,11]. For example, Huffman codes [11] are prefix-free languages and useful for lossless data compression; Han [6] proposed an efficient pattern matching algorithm for the prefix-free regular expressions based on the prefix-freeness of the input pattern. Infix-free languages have been used in text searching [3] and computing forbidden words [1,4]. Han et al. [8] observed the structural properties of FAs for infix-free regular languages and designed a decision algorithm.

I. Potapov (Ed.): DLT 2015, LNCS 9168, pp. 264–275, 2015.
DOI: 10.1007/978-3-319-21500-6_21

Table 1. The upper bound for the time complexity of decision algorithms for the generalizations of prefix-, suffix- and infix-free regular languages, where n is the size of FAs

	DFA	NFA
prefix-free	$O(n)$	$O(n^2)$
suffix-free	$O(n^2)$	$O(n^2)$
infix-free	$O(n^2)$	$O(n^2)$
k-prefix-free	$O(n^2 \cdot \log k)$	PSPACE-complete
k-suffix-free	PSPACE-complete	PSPACE-complete
k-infix-free	PSPACE-hard	PSPACE-hard
finitely prefix-free	$O(n)$	$O(n^2)$
finitely suffix-free	$O(n^2)$	$O(n^2)$
finitely infix-free	$O(n^2)$	$O(n^2)$

When people design a code in practice—for instance, designing a DNA code by experiments [14,16]—it may not always be successful. There may be a few number of undesired code words in the resulting code. This motivates us to examine a relaxed version of codes; in other words, we allow some marginal errors in the code. We generalize prefix-free, suffix-free and infix-free languages and define new codes, *k-prefix-free, k-suffix-free and k-infix-free* languages. Recall that there is no prefix of any other string in the language L if L is prefix-free [2]. In a k-prefix-free language L, we allow at most k strings in L to have another string in L as a prefix. We also introduce *finitely prefix-free*, *finitely suffix-free* and *finitely infix-free* languages in which we allow a finite number of such strings.

Given a family $\mathcal{L}$ of languages and a language L, the *decision problem* of L with respect to $\mathcal{L}$ is to decide whether or not L belongs to $\mathcal{L}$. For the decision problems of finitely prefix-free, finitely suffix-free and finitely infix-free regular languages, we use the *state-pair graph* used for the decision problems of prefix-free, suffix-free and infix-free regular languages [5]. Table 1 summarizes the time complexity of the prior algorithms and our algorithms based on the state-pair graph. Interestingly, we have a polynomial time algorithm only for deciding the k-prefix-freeness of a language recognized by a DFA. We prove that the complexity for determining whether or not an NFA is k-prefix-free is PSPACE-complete. For both k-suffix-free cases, it is already PSPACE-hard for DFAs. For the decision problems of finitely prefix-free, suffix-free and infix-free regular languages, we present polynomial algorithms based on the state-pair graph construction. We also establish the undecidability results for context-free languages by the reduction from the Post's Correspondence Problem [17].

We define some basic notions in Section 2. We define the generalizations of prefix-free, suffix-free and infix-free regular languages in Section 3 and observe the hierarchy between the proposed subfamilies. In Section 4, we present decision algorithms and complexity results for regular languages. We also establish undecidability result for context-free languages in Section 5.

2 Preliminaries

We briefly present definitions and notations. We refer to the books [10,18] for more knowledge on automata theory. For more details on coding theory, refer to Berstel and Perrin [2] or Jürgensen and Konstantinidis [12].

Let Σ be a finite alphabet and Σ^* be the set of all strings over Σ. A language over Σ is any subset of Σ^*. The symbol $\emptyset$ denotes the empty language, the symbol λ denotes the null string and Σ^+ denotes $\Sigma^* \setminus \{\lambda\}$. Let $|w|$ be the length of w.

For strings x, y and z, we say that x is a *prefix* of y and z is a *suffix* of y if $y = xz$. For strings x, y, w and z, we say that z is an infix of y if $y = xzw$. We define a language L to be prefix-free if a string $x \in L$ is not a prefix of any other strings in L. Similarly, we define a language L to be suffix-free (or infix-free) if a string $x \in L$ is not a suffix (or infix) of any other strings in L. The reversal of a string w is denoted w^R and the reversal of a language L is $L^R = \{w^R \mid w \in L\}$.

A *nondeterministic finite automaton with λ-transitions* (λ-NFA) is a five-tuple $A = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is a finite alphabet, δ is a multi-valued transition function from $Q \times (\Sigma \cup \{\lambda\})$ into 2^Q, $s \in Q$ is the initial state and $F \subseteq Q$ is the set of final states. By an NFA, we mean a nondeterministic automaton without λ-transitions, that is, A is an NFA if δ is a function from $Q \times \Sigma$ into 2^Q. The automaton A is *deterministic* (a DFA) if δ is a single-valued function $Q \times \Sigma \to Q$. The language $L(A)$ recognized by A is the set of strings w such that some sequence of transitions spelling out w takes the initial state of A to a final state. We define the size $|A|$ of A to be $|Q| + |\delta|$. For $q \in Q$, we denote by A_q the NFA that is obtained from A by replacing the initial state s with q. For $q, p \in Q$, we also denote by $A_{q,p}$ the NFA that is as A except has q as the initial state and p as the only final state. We assume that an FA A has only useful states; that is, each state appears on some path from the initial state to some final state.

3 *k*-prefix-free, *k*-suffix-free and *k*-infix-freeness

We consider generalizations of prefix-free, suffix-free and infix-free languages. First, we define a generalization of prefix-free languages called the *k-prefix-free languages*.

Definition 1. *We define a language L to be k-prefix-free if there is no string x in L that is a prefix of more than k strings in $L \setminus \{x\}$.*

Recently, Konitzer and Simon [15] defined the *maximum activity level* of a language L as follows: Given a language L, the maximum activity level l_L of L is

$$l_L = \sup\{l : (\exists w_1, \ldots, w_l \in \Sigma^+, \forall = 1, \ldots, l : w_1 \cdots w_l \in L)\}.$$

Since we consider the number of strings in L that have a common prefix from L and the activity level is defined as the number of proper prefixes of a string in L, the k-prefix-freeness is a different concept from the activity level of a language.

From the definition, we can say that prefix-free languages are in fact, 0-prefix-free. We define similar generalizations for suffix-free and infix-free languages as follows:

Definition 2. *We define a language L to be k-suffix-free (or k-infix-free) if there is no string x in L that is a suffix (or an infix) of more than k strings in $L \setminus \{x\}$.*

We have the following observations from the definition. Note that similar statements hold for suffix-free and infix-free languages.

Observation 1. *If a language L is k-prefix-free, then L is $(k+n)$-prefix-free for all $n \geq 1$.*

Observation 2. *If a language L is prefix-free, then L is k-prefix-free for all $k \geq 1$.*

We also define the following code properties from the k-prefix (suffix, infix)-freeness.

Definition 3. *We define a language L to be* minimally k-prefix (suffix, infix)-free *if L is k-prefix (suffix, infix)-free but not $(k-1)$-prefix (suffix, infix)-free.*

Definition 4. *We define a language L to be finitely prefix-free (suffix-free or infix-free) if L is k-prefix-free (k-suffix-free or k-infix-free) for a constant k.*

4 Decision Algorithms

We determine whether or not a regular language given by a DFA is k-prefix-free or finitely prefix-free. First, we consider decision problems where k is given as part of the input in Section 4.1 and consider the problem of deciding whether the property holds for some finite value of k in Section 4.2.

4.1 For k-prefix-free, k-suffix-free and k-infix-free Regular Languages

Given a DFA A, we can determine whether or not $L(A)$ is k-prefix-free or finitely prefix-free by examining the following properties:

Lemma 1. *Given a DFA $A = (Q, \Sigma, \delta, s, F)$, $L(A)$ is k-prefix-free if and only if there is no final state $f \in F$ satisfying the following conditions:*

$$L(A_{s,f}) \neq \emptyset \quad \textit{and} \quad |L(A_f) \setminus \{\lambda\}| > k. \tag{1}$$

Proof. ($\Longrightarrow$) We prove that $L(A)$ is k-prefix-free if there is no final state f satisfying Condition (1). Assume that there is such a final state f satisfying the condition. This means that there is a string w spelled out by the path from s to f and

$$|L(A_f) \setminus \{\lambda\}| > k.$$

Since there are more than k different strings that can be computed after w, there exist more than k strings in $L(A)$ that contain w as a proper prefix. Therefore, $L(A)$ is not k-prefix-free. We have a contradiction.
($\Longleftarrow$) We prove that there is no final state f satisfying Condition (1) if $L(A)$ is k-prefix-free. Assume that $L(A)$ is not k-prefix-free. This implies that there exist a string w in L and more than k strings—let P_w denote the set of these strings—that contain w as a prefix in $L(A)$. Since A is a DFA, there is a unique final state f that we reach by reading w and all the accepting paths for strings in P_w must pass through f. This guarantees that $|L(A_f) \setminus \{\lambda\}| > k$ since $|P_w| > k$. Therefore, we have a contradiction.

Lemma 2. *Let A be a DFA (or an NFA) over an alphabet Σ with n states such that $L(A)$ is minimally k-prefix-free. Then*

$$k \leq |\Sigma| \cdot \frac{|\Sigma|^{n-1} - 1}{|\Sigma| - 1}.$$

Conversely, there exists an n-state DFA A over Σ that is minimally $|\Sigma| \cdot \frac{|\Sigma|^{n-1}-1}{|\Sigma|-1}$-prefix-free.

From Lemma 1, we can design a quadratic algorithm for deciding whether or not a language recognized by a DFA is k-prefix-free.

Theorem 1. *Given a DFA A and $k \in \mathbb{N}$, we can determine whether or not $L(A)$ is k-prefix-free in $O(n^2 \cdot \log k)$ time, where $n = |A|$.*

Note that the family of k-prefix-free languages form a proper hierarchy with respect to k. For example, if a regular language L is k-prefix-free, then L is not necessarily $(k-1)$-prefix-free. This leads us to a new problem that determines whether or not a regular language L is minimally k-prefix-free.

Theorem 2. *Given a DFA $A = (Q, \Sigma, \delta, s, F)$, we can find $k \in \mathbb{N}$ such that $L(A)$ is minimally k-prefix-free in $O(n^3)$ time, where $n = |A|$.*

Proof. For an arbitrary final state f of A, if A_f is not acyclic, then we know that $L(A)$ is not k-prefix-free for any $k \in \mathbb{N}$. Otherwise, based on the property of Lemma 1, it is possible to compute in polynomial time the cardinality of $L(A_f)$, for each final state f, and pick the maximum value M. Then $L(A)$ is minimally M-prefix-free. Since M is at most $O(|\Sigma|^{|Q|})$ by Lemma 2, the runtime is

$$O(n^2 \cdot \log |\Sigma|^n) = O(n^3)$$

by assuming $|\Sigma|$ as a constant. □

Now we consider the case when the input is specified by an NFA instead of a DFA.

Lemma 3. *Given an NFA $A = (Q, \Sigma, \delta, s, F)$, $L(A)$ is k-prefix-free if and only if there is no set $Q' \subseteq Q$ of states satisfying the following conditions:*

(i) $Q' \cap F \neq \emptyset$,

(ii) $\bigcap_{q \in Q'} L(A_{s,q}) \neq \emptyset$ *and* $\left| \bigcup_{q \in Q'} L(A_q) \setminus \{\lambda\} \right| > k$.

Proof. ($\Longrightarrow$) We prove that $L(A)$ is k-prefix-free if there is no set $Q' \subseteq Q$ of states satisfying the two conditions (i) and (ii). Assume that there is such a set Q' satisfying all these conditions. Then, there exists a string $w \in L(A)$ such that $Q' \subseteq \delta(s, w)$ by the conditions. Furthermore, we have more than k non-empty strings that can be spelled out by a path from a state in Q' to a final state in F because of condition (ii). This implies that $L(A)$ has more than k strings that contain w as a proper prefix. Therefore, $L(A)$ is not k-prefix-free.
($\Longleftarrow$) We prove that if $L(A)$ is k-prefix-free, then there is no set Q' of states satisfying these three conditions. Assume that $L(A)$ is not k-prefix-free. This implies that there exist a string w in L and more than k strings having w as a prefix in $L(A)$. Since A accepts w, there should exist a final state f such that $f \in \delta(s, w)$. We denote the set of all states that can be reached by reading w from the initial state s in A by P. Obviously, f is in P. Since there are more than k strings that have w as a proper prefix, there are also more than k non-empty strings that can be spelled out by paths from states in P to some final states. Since P satisfies conditions (i) and (ii), we have a contradiction. □

Now we show that the problem of determining whether or not a regular language accepted by an NFA is k-prefix-free is PSPACE-complete.

Theorem 3. *Given an NFA A and $k \in \mathbb{N}$, it is* PSPACE*-complete to determine whether or not $L(A)$ is k-prefix-free.*

We also establish that given an NFA A, the problem of finding a non-negative integer k where $L(A)$ is minimally k-prefix-free is also PSPACE-complete.

Theorem 4. *Given an NFA A, the problem of finding $k \in \mathbb{N}$ where $L(A)$ is minimally k-prefix-free is* PSPACE*-complete.*

Now we consider the k-suffix-free case. The properties that characterize the k-suffix-freeness of a language recognized by a DFA turn out to be similar to the properties that characterize the k-prefix-freeness of an NFA. Intuitively, suffix-freeness corresponds to the prefix-freeness of the reversed language and reversing the transitions of a DFA makes it nondeterministic. We present necessary and sufficient conditions for DFAs to accept k-suffix-free regular languages as follows:

Lemma 4. *Given an NFA $A = (Q, \Sigma, \delta, s, F)$, $L(A)$ is k-suffix-free if and only if there is no set $Q' \subseteq Q$ of states satisfying the following conditions:*

$$\left| \bigcup_{q \in Q'} L(A_{s,q}) \setminus \{\lambda\} \right| > k \quad \textit{and} \quad \bigcap_{q \in Q' \cup \{s\}} L(A_q) \neq \emptyset. \tag{2}$$

Proof. ($\Longrightarrow$) We prove that $L(A)$ is k-suffix-free if there is a set $Q' \subseteq Q$ of states satisfying Condition (2). Assume that there is a set $Q' \subseteq Q$ of states satisfying the condition. From the condition

$$\bigcap_{q \in Q' \cup \{s\}} L(A_q) \neq \emptyset,$$

we know that there is a common string $w \in L(A)$ that is also spelled out by paths from the states in Q' to one of the final states. Since the cardinality of the set of strings spelled out by the paths from the initial state to the states in Q' is greater than k, there are more than k strings that have w as a suffix. Since the string w is also in $L(A)$, $L(A)$ is not k-suffix-free. We have a contradiction.
($\Longleftarrow$) We prove that there is no set $Q' \subseteq Q$ of states satisfying Condition (2) if $L(A)$ is k-suffix-free. Assume that $L(A)$ is not k-suffix-free. This implies that there exist a string $w \in L(A)$ and more than k strings containing w as a suffix in $L(A)$. Assume that there exist $k_1 > k$ strings that contain w as a suffix and denote the strings by $w_1, w_2, w_3, \ldots, w_{k_1}$. For all strings $w_i, 1 \leq i \leq k_1$, we can decompose w_i into $w'_i w$ since they have the common suffix w. Then, the strings from w'_1 to w'_{k_1} should be distinct and non-empty since $w_1, w_2, w_3, \ldots, w_{k_1}$ are distinct and properly containing w as a suffix. Let us denote the set of states that are reachable from the initial state s by reading $w'_1, w'_2, \ldots, w'_{k_1}$ by $P \subseteq Q$. Therefore, the following inequality holds:

$$\left| \bigcup_{q \in P} L(A_{s,q}) \setminus \{\lambda\} \right| > k.$$

Moreover, there should be at least one path from each state in P and the initial state to one of the final states of A spelling out the string w since A accepts the strings $w, w_1, w_2, w_3, \ldots, w_{k_1}$ that are in fact, $\lambda \cdot w, w'_1 \cdot w, w'_2 \cdot w, \ldots, w'_{k_1} \cdot w$. Therefore, the following inequality also holds:

$$\bigcap_{q \in P \cup \{s\}} L(A_q) \neq \emptyset.$$

Now we have a contradiction since there is a set $P \subseteq Q$ of states satisfying Condition (2). □

We discuss the computational complexity of the decision problem for k-suffix-free regular languages. Interestingly, it turns out to be much more complicated than the k-prefix-free case even for DFAs.

Theorem 5. *Given a DFA (or an NFA) A and $k \in \mathbb{N}$, it is* PSPACE*-complete to determine whether or not $L(A)$ is k-suffix-free.*

Theorem 6. *Given a DFA (or an NFA) A, the problem of finding $k \in \mathbb{N}$ where $L(A)$ is minimally k-suffix-free is* PSPACE*-complete.*

Moreover, it is immediate from the k-suffix-free case that the problem of determining whether or not a regular language $L(A)$ is k-infix-free is also PSPACE-hard.

Corollary 1. *Given a DFA (or an NFA) A and $k \in \mathbb{N}$, it is* PSPACE*-hard to determine whether or not $L(A)$ is k-infix-free. Given a DFA (or an NFA) A, the problem of finding $k \in \mathbb{N}$ where $L(A)$ is minimally k-infix-free is* PSPACE*-hard.*

Proof. We show that the problem of determining whether or not $L(A)$ is k-infix-free is PSPACE-hard by the reduction from the k-suffix-free case. Let $L \subseteq \Sigma^*$ be a regular language. Then, we can check the k-suffix-freeness of L by checking the k-infix-freeness of $L \cdot \{\$\}$, where $\$ \notin \Sigma$ is a new symbol.

It remains open to show that the problem of determining whether or not $L(A)$ is k-infix-free is in PSPACE.

4.2 For Finitely Prefix-free, Finitely Suffix-free and Finitely Infix-free Regular Languages

We establish the following corollary in a similar way to the proof of Lemma 1.

Corollary 2. *Given a DFA $A = (Q, \Sigma, \delta, s, F)$, $L(A)$ is finitely prefix-free if and only if there is no final state $f \in F$ satisfying the following conditions:*

$$L(A_{s,f}) \neq \emptyset \quad \textit{and} \quad |L(A_f)| = \infty.$$

By Corollary 2, we can determine whether or not a regular language is finitely prefix-free in linear time when the regular language is given by a DFA.

Theorem 7. *Given a DFA A, we can determine whether or not $L(A)$ is finitely prefix-free in $O(n)$ time, where $n = |A|$.*

For the NFA case, on the other hand, we need to use algorithms based on the *state-pair graphs* [5] to determine whether or not a regular language accepted by an NFA is finitely prefix-free. For example, for DFAs, we can easily decide the prefix-freeness (including k-prefix-freeness and finitely prefix-freeness) by observing the accepting path after the final states since all strings having the same prefix should share the same path to spell out the prefix. However, in NFAs, we cannot guarantee such properties. There can be several completely disjoint paths even for a single string. Han [5] showed that it is possible to determine whether or not a regular language given by an NFA is prefix-free, suffix-free or infix-free using the *state-pair graph* in quadratic time in the size of an input.

Definition 5 (Han [5]). *Given an NFA $A = (Q, \Sigma, \delta, s, F)$, we define the state-pair graph $G_A = (V, E)$ of A, where V is a set of nodes and E is a set of labeled edges, as follows:*

- $V = \{(i, j) \mid i, j \in Q\}$ *and*
- $E = \{((i, j), a, (x, y)) \mid x \in \delta(i, a), y \in \delta(j, a)$ *and* $a \in \Sigma\}$.

Then, we have $|G_A| \leq |Q|^2 + |\delta|^2$; namely $|G_A| = O(|A|^2)$.

Without loss of generality, we assume that A has one final state f since we can always make any NFA to have only one final state by introducing a new final state f and making f to be the target state of all final states by a λ-transition.

Lemma 5. *Given an NFA $A = (Q, \Sigma, \delta, s, f)$ and $q, p \in Q$, $L(A)$ is finitely prefix-free if and only if there is no path labeled by a string from (s, s) to (f, p) in G_A satisfying $|L(A_{p,f})| = \infty$.*

Proof. ($\Longrightarrow$) We first prove that $L(A)$ is finitely prefix-free if there is no such path. Assume that there is a such path labeled by a string w. By the assumption, the cardinality of $L(A_{p,f})$ is infinite. This implies that A accepts an infinite number of strings having w as a proper prefix. Since $w \in L(A)$ by assumption, $L(A)$ is not finitely prefix-free.
($\Longleftarrow$) We prove that if $L(A)$ is finitely prefix-free there is no such path. Assume that $L(A)$ is not finitely prefix-free. This implies that there are an infinite number of strings having a prefix w and w is accepted by A. Consider a set Q' of states that are reachable by reading w from the initial state of A. Since there are an infinite number of strings having w as a prefix, there should be an infinite number of distinct accepting paths starting from the states in Q'. If $L(A_{p,f})$ is finite for all $p \in Q'$, it is impossible to have an infinite number of accepting paths from the states in Q'. Thus at least one state $p \in Q'$ should have an infinite number of accepting paths. Now we have a contradiction. □

From the properties of the state-pair graph observed in Lemma 5, we can determine whether or not $L(A)$ is finitely prefix-free by exploring the existence of such paths in the state-pair graph. Since the size of the state-pair graph is quadratic in the size of the given NFA, the time complexity of the algorithm is also quadratic.

Theorem 8. *Given an NFA $A = (Q, \Sigma, \delta, s, f)$, we can determine whether or not $L(A)$ is finitely prefix-free in $O(n^2)$ time, where $n = |A|$.*

Proof. First, we check whether or not $L(A_q)$ is finite for each $q \in Q$. This procedure takes $O(n^2)$ time since we can check the existence of a cycle of an NFA in linear time by depth-first traversal. Then, we explore the state-pair graph to check whether or not there is a path from (s, s) to (f, p) in the state-pair graph G_A when $L(A_p)$ is infinite. As the size of G_A is quadratic in the size of A, the algorithm takes $O(n^2)$ time. □

Note that the decision algorithm for finitely infix-freeness is relatively more complicated than the finitely prefix-free case.

Lemma 6. *Given an NFA $A = (Q, \Sigma, \delta, s, f)$ and $q, p \in Q$, $L(A)$ is finitely infix-free if and only if there is no path labeled by a string from (s, q) to (f, p) where $f \neq p$ in G_A satisfying $|L(A_{s,q}) \cup L(A_{p,f})| = \infty$.*

Proof. ($\Longrightarrow$) We prove that $L(A)$ is finitely infix-free if there is no such path. Assume that there is a such path labeled by a string w. Then, w is in $L(A)$ and

$L(A_{q,p})$. Since the cardinality of $L(A_{s,q}) \cup L(A_{p,f})$ is infinite by assumption, the cardinality of $L(A_{s,q})$ or $L(A_{p,f})$ should be infinite. We can easily see that there are an infinite number of strings containing w as an infix in $L(A)$. Therefore, $L(A)$ is not finitely infix-free. We have a contradiction.
($\Longleftarrow$) Here we prove that if $L(A)$ is finitely infix-free, then there is no such path. Assume that $L(A)$ is not finitely infix-free. This means that for a string $w \in L(A)$, we have an infinite number of strings in $L(A)$ containing w as an infix. Consider all state pairs q, p where $w \in L(A_{q,p})$. If $L(A_{s,q}) \cup L(A_{p,f})$ is finite for all state pairs, the number of strings containing w as an infix cannot be infinite. Therefore, $L(A_{s,q}) \cup L(A_{p,f})$ should be infinite. We conclude the proof. □

Even though the decision algorithm for finitely infix-freeness is a bit more complicated than the finitely prefix-free case, we still have a quadratic algorithm for the problem as follows:

Theorem 9. *Given an NFA $A = (Q, \Sigma, \delta, s, f)$, we can determine whether or not $L(A)$ is finitely infix-free in $O(n^2)$ time, where $n = |A|$.*

Proof. We first construct a state-pair graph G_A of A. Before exploring the existence of the paths, we first check whether or not $L(A_{s,q})$ and $L(A_{q,f})$ is finite. This procedure takes $O(n^2)$ time.

Once we finish the checking procedure, for all states $q \in Q$ except s, we run the depth-first search from (s, q) to (f, p) to check whether or not there exists a such path. If there exists a such path, then we check whether or not $L(A_{s,q}) \cup L(A_{p,f})$ is finite by simply checking whether or not both languages are finite. Then, for all states $q \in Q$ except f, we run the depth-first search from (s, s) to (f, q) to check whether or not there exists a such path. Since the depth-first search takes $O(|V|+|E|)$ time when we are given a graph $G = (V, E)$, we can check whether or not there exists a path in $O(n^2)$ time. Overall, our algorithm takes $O(n^2)$ time. □

We note that the finitely suffix-freeness can be checked in a converse way to the finitely prefix-free case.

Lemma 7. *Given an NFA $A = (Q, \Sigma, \delta, s, f)$ and $q \in Q$, $L(A)$ is finitely suffix-free if and only if there is no path labeled by a string from (s, q) to (f, f) in G_A satisfying $|L(A_{s,q})| = \infty$.*

Proof. The proof is immediate from Lemma 5. We just exchange the initial state s and the final state f from A, and reverse the directions of all transitions. Then the new NFA accepts $L(A)^R$—the reverse of $L(A)$. Then, we can decide whether or not $L(A)$ is finitely suffix-free by testing the finitely prefix-freeness of $L(A)^R$. □

Since the finitely suffix-freeness can be tested by reversing the NFA and testing the finitely prefix-freeness, we establish the following result.

Theorem 10. *Given an NFA A, we can determine whether or not $L(A)$ is finitely suffix-free $O(n^2)$ time, where $n = |A|$.*

5 Undecidability Results for (Linear) Context-free Languages

It is known that given a (linear) context-free language L, it is undecidable whether or not L is prefix-free [12]. We establish similar undecidability results for the code properties considered here.

Theorem 11. *There is no algorithm that determines whether or not a given linear language L is k-prefix (suffix, infix)-free.*

Theorem 12. *There is no algorithm that determines whether or not a given linear language L is finitely prefix (suffix, infix)-free.*

6 Conclusions

We have introduced an extension of prefix-free, suffix-free and infix-free languages by allowing marginal errors. We have defined k-prefix-free, k-suffix-free and k-infix-free languages. For example, a k-prefix-free language L can have at most k strings containing any other string w in L as a prefix. We, then, have considered more general versions of k-prefix-free, k-suffix-free and k-infix-free languages allowing a finite number of such strings.

We have examined the time complexity of the decision problems for these subfamilies. Given a DFA, we have a quadratic algorithm for the decision problem of k-prefix-freeness. However, we have shown that it is PSPACE-complete to determine whether or not a regular language given by an NFA is k-prefix-free. For the k-suffix-free and k-infix-free cases, we have shown that it is PSPACE-hard for both DFAs and NFAs. We have designed polynomial time algorithms based on the state-pair graph construction that decide whether or not a regular language (given by an NFA) is finitely prefix- (respectively, suffix- or infix-) free. We also have established the undecidability results for (linear) context-free languages. In future, we plan to investigate the state complexity of these new subfamilies of regular languages.

References

1. Béal, M.P., Crochemore, M., Mignosi, F., Restivo, A., Sciortino, M.: Computing forbidden words of regular languages. Fundamenta Informaticae **56**(1,2), 121–135 (2002)
2. Berstel, J., Perrin, D.: Theory of codes. Academic Press, Inc. (1985)
3. Clarke, C.L.A., Cormack, G.V.: On the use of regular expressions for searching text. ACM Transactions on Programming Languages and Systems **19**(3), 413–426 (1997)
4. Crochemore, M., Mignosi, F., Restivo, A.: Automata and forbidden words. Information Processing Letters **67**(3), 111–117 (1998)
5. Han, Y.S.: Decision algorithms for subfamilies of regular languages using state-pair graphs. Bulletin of the European Association for Theoretical Computer Science **93**, 118–133 (2007)

6. Han, Y.S.: An improved prefix-free regular-expression matching. International Journal of Foundations of Computer Science **24**(5), 679–687 (2013)
7. Han, Y.S., Salomaa, K., Wood, D.: Intercode regular languages. Fundamenta Informaticae **76**(16), 113–128 (2007)
8. Han, Y.S., Wang, Y., Wood, D.: Infix-free regular expressions and languages. International Journal of Foundations of Computer Science **17**(2), 379–393 (2006)
9. Han, Y.S., Wang, Y., Wood, D.: Prefix-free regular languages and pattern matching. Theoretical Computer Science **389**(1–2), 307–317 (2007)
10. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley Longman Publishing Company Incorporated (2006)
11. Huffman, D.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE **40**(9), 1098–1101 (1952)
12. Jürgensen, H., Konstantinidis, S.: Codes. Word, Language, Grammar, Handbook of Formal Languages **1**, 511–607 (1997)
13. Kari, L., Konstantinidis, S., Kopecki, S.: On the maximality of languages with combined types of code properties. Theoretical Computer Science **550**, 79–89 (2014)
14. Kari, L., Mahalingam, K.: DNA Codes and Their Properties. In: Mao, C., Yokomori, T. (eds.) DNA12. LNCS, vol. 4287, pp. 127–142. Springer, Heidelberg (2006)
15. Konitzer, M., Simon, H.U.: DFA with a Bounded Activity Level. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) LATA 2014. LNCS, vol. 8370, pp. 478–489. Springer, Heidelberg (2014)
16. Marathe, A., Condon, A.E., Corn, R.M.: On combinatorial dna word design. Journal of Computational Biology **8**(3), 201–219 (2001)
17. Post, E.L.: A variant of a recursively unsolvable problem. Bulletin of the American Mathematical Society **52**(4), 264–268 (1946)
18. Wood, D.: Theory of Computation. Harper & Row (1987)