Union-Complexities of Kleene Plus Operation

Benedek Nagy(✉)

Department of Mathematics, Faculty of Arts and Sciences,
Eastern Mediterranean University, Famagusta, North Cyprus, Mersin-10, Turkey
nbenedek.inf@gmail.com

Abstract. Union-free expressions are used in union normal form to decompose any regular language to a finite union of union-free languages. Based on the automata characterisation of the union-free languages, by restricting the 1CFPAs not to have transitions by the empty word, or to be deterministic, the n-union-free and the deterministic union-free languages are defined. Union-complexity as a measure of descriptional complexity of regular languages was introduced recently. By the minimum number of union-free/n-union-free/deterministic union-free languages needed to get a regular language as their union, its union-complexity/n-union-complexity/d-union-complexity is defined. It is already known that union-complexity and n-union-complexity are finite for every regular language, however there are regular languages with infinite d-union-complexity. Operational union-complexity, that is, to predict the union-complexity of a language obtained by a language operation from languages with known union-complexity is an important and interesting question belonging to the field of descriptional complexity of formal systems. In the present paper, the Kleene plus, the positive Kleene closure operator is studied. As the Kleene star and plus operations have very different effects on the union-free languages, it is an interesting problem to investigate how the union-complexities may change under this operation. In particular, we show that the union-complexity of a regular language is not growing when this operation is being applied on it. On the other hand, the n-union-complexity of the Kleene plus of an n-union-free language remains 1, but the n-union-complexity of the Kleene plus of other regular languages may grow. Further, the deterministic union-complexity may jump to an infinite value even if the original language had a relatively small deterministic union-complexity, e.g., 4.

Keywords: union-complexity · union-free languages · regular expressions · Kleene closure

1 Introduction

Various classes of subregular languages are important from various points of view, see, e.g., [5,9]. The union-free languages are defined by regular expressions without the union, they are the star-dot regular languages [2]. Automata

© IFIP International Federation for Information Processing 2022
Published by Springer Nature Switzerland AG 2022
Y.-S. Han and G. Vaszil (Eds.): DCFS 2022, LNCS 13439, pp. 197–211, 2022.
https://doi.org/10.1007/978-3-031-13257-5_15

theoretical characterisation [11] allowed to define the deterministic counterpart: the deterministic union-free languages [3,7,8] and by the nondeterministic λ-transition-free automata, the n-union-free languages. The classes of the union-free, n-union-free and d-union-free languages form a proper hierarchy [14] and they were studied in [2,4,7,11], [14] and [3,8], respectively. Based on possible decomposition of regular languages to finite unions of those languages, the union-complexity, n-union-complexity and d-union-complexity are defined [1,10,12,13,15] (note that this latter could be infinite according to [8] even if the language is regular). The operational union-complexity is studied in details under various operations in [15], except, e.g. Kleene plus. On the other hand, the class of union-free languages is closed under concatenation, Kleene plus and also under Kleene star. Moreover, for any regular language, its Kleene star is union-free, but a similar statement does not hold for Kleene plus. Further, this class is not closed under union and this gives the possibility to define the union normal form and union-complexity of regular languages. Since Kleene plus and Kleene star behave in different ways from our point of view, it is worth to study Kleene plus and we concentrate on this issue in this paper. Closure, or indeed, more precisely, anti-closure properties of n-union-free and d-union-free languages were studied in [8,14], respectively. The non trivial closure properties of the classes of n-union-free and d-union-free languages also give the challenge to analyse the analogous union-complexity measures under various operations. Here as we already mentioned, the Kleene plus is in our focus.

While another usual measure of descriptional complexity of regular languages is connected to the minimal number of states of the accepting finite automata, the union-complexity is closely connected to the union normal form and thus to the regular expressions describing the language [10,12,13].

2 Preliminaries

In this section, first we recall the definition of the union-free languages and the corresponding class of finite automata. We assume that the reader is familiar with the basic concepts of formal languages and automata, thus for each unexplained concepts she/he is referred to any standard textbook on the topic, e.g., to [6] or to the Handbook chapter [17]. Here we show only specific notions closely related to the topic of this paper. The empty word is denoted by λ; Σ is a finite alphabet, while $\cup, \cdot, ^*, ^+$ denote the usual operations on languages, i.e., the union, the concatenation, the Kleene star and the Kleene plus. Now we recall some (formal) concepts, definitions and notions from earlier mentioned studies.

A regular expression is a *union-free expression* if only the operators concatenation and Kleene star are used in its description. A regular language is a *union-free language* if there is a union-free expression that defines it.

We note here that in the literature sometimes a wider class of languages are called union-free, those which have a description by operations concatenation, Kleene star and complement [9], somewhat similarly as the description of star-free languages goes by concatenation, union and complement [17].

Now we briefly recall the concept of finite automata and fix some notations.

A 5-tuple $\mathbf{A} = (Q, S, \Sigma, \delta, F)$ is a *non-deterministic finite automaton*, with the finite set of states Q. Further, $S \in Q$ is the initial state, Σ is the (input) alphabet and $F \subset Q$ is the set of final (or accepting) states. The function $\delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$ is the transition function.

A *path* $Q_0 a_1 Q_1 a_2 Q_2 \dots a_{n-1} Q_{n-1} a_n Q_n$ where $Q_{i+1} \in \delta(Q_i, a_{i+1})$ for every $0 \leq i < n$ (with $n > 0$) is called a *cycle* if $Q_0 = Q_n$. A path without any repeated state is called a *cycle-free path*.

A path is called an *accepting path*, if it ends in a final state. Further, it is an accepting path of a word w if it is written as $(S = Q_0)a_1 Q_1 a_2 Q_2 ... a_{n-1} Q_{n-1} a_n Q_n$ with $Q_n \in F$ and $w = a_1 a_2 ... a_n$ $(a_i \in \Sigma \cup \{\lambda\})$, i.e., it is an accepting path starting at the initial state. A word is accepted by the finite automata if it has an accepting path.

Definition 1 (1CFPA, n-1CFPA, d-1CFPA). *A nondeterministic finite automaton* $\mathbf{A}$ *is a* ***1*** ***c****ycle-****f****ree* ***p****ath* ***a****utomaton, a* 1CFPA*, for short, if there is a unique cycle-free accepting path from each of its states. Moreover, if the automaton* $\mathbf{A}$ *does not have any* λ*-transitions, then it is an* n-1CFPA*, and if* $\mathbf{A}$ *is deterministic, then it is a* d-1CFPA*.*

In this paper, we use only automata with the following property: for each state Q_i of the automaton there is a word such that it has an accepting path that contains Q_i. Consequently, there is no useless or sink state and the automaton may not be fully determined, i.e., it may happen that for a state Q_i and an input letter a the transition function assigns the empty set.

As a consequence of the definition above, a 1CFPA has exactly one final state. From now on F will refer not only to the set of final states, but to its unique element, as well, in case of a 1CFPA. One of the main results of [11] states that the family of languages which are described by union-free expressions and the family of languages recognized by 1CFPAs are exactly the same. Based on this relation, two further classes of union-free languages are defined as follows:

Definition 2 (d-union-free and n-union-free languages). *A language is* deterministic union-free *if there is a deterministic 1CFPA which accepts it [8,13]. The short form* d-union-free *will also be used for these languages. Further, the* n-union-free *languages are exactly those union-free languages that can be accepted by n-1CFPA [14].*

Observe that by definition, in a 1CFPA, a transition between two distinct states cannot be part of the unique cycle-free path from any state to the final state, if there is a parallel transition between the same two states. The issue with parallel transitions can be resolved by a construction duplicating some parts of the automaton. Based on this, every x-union-free language ($x \in \{\lambda$, n, d$\}$) is accepted by an x-1CFPA such that for any two distinct states P, R there is at most one letter such that there is a transition from P to R with that letter.Therefore, in various constructions and proofs, w.l.o.g., we may assume that there is no transition with two different letters between two distinct states of the automaton.

Since in a 1CFPA, from every state R, there is exactly one transition that goes to the direction of F (without cycle), the word which transfers the state R to F in a cycle-free path is unique for each state. We recall that the *backbone* of the automaton is the cycle-free path from the initial state (S) to the final state (F). The word accepted by the backbone is called the *backbone word*.

The following facts are known about union-free languages [11]:

- An x-union-free ($x \in \{\lambda, \text{n}, \text{d}\}$) language is either infinite or contains at most one word.
- The shortest word of a union-free language L is unique and it is the backbone word. In a union-free language each word contains the backbone word (maybe in a scattered way).

The usual expression tree concept can be used to represent regular expressions. To each leaf of the tree a letter of the alphabet or the empty word is assigned. To each other vertex the sign of a regular operation is assigned such that a vertex with assigned $\cup$ or $\cdot$ has exactly two children in tree, while a vertex with assigned * or $^+$ has exactly one child. Let L be a union-free language. Note that $\lambda \in L$ if and only if the backbone word is the empty word. This implies that every letter is under a Kleene star in the tree of the regular expression. Under these circumstances the language can be accepted by a 1CFPA with backbone word λ. If L is n-union-free and $\lambda \in L$, then $S = F$ in the corresponding n-1CFPA. Since every 1CFPA (and thus d-1CFPA) has exactly one accepting state, languages which cannot be accepted by deterministic finite automata with only one final state are not d-union-free languages.

It is known (see, [8,11]) that the family of union-free languages is closed under the operations concatenation, Kleene star and Kleene plus. The family of n-union-free languages is not closed under union and concatenation, but it is closed under Kleene plus [14]. Furthermore, the class of unary n-union-free languages is closed under concatenation, Kleene star, Kleene plus. On the other hand, we have only anti-closure properties for the d-union-free languages: e.g., their class is not closed under union, concatenation, Kleene star, [8] and Kleene plus [14].

In fact, all d-union-free languages are n-union-free languages and all n-union-free languages are union-free. The language a^*b^* is union-free. However, it is not n-union-free [14]. The language $a(b \cup ba)^*$ is n-union-free, but not d-union-free [8]. Thus, there is a proper hierarchy among the three mentioned union-free classes.

By the decomposition result mentioned in [2,10,16], the union-complexity of regular languages is defined in [10,12]. As one of the main results of [14] states, every regular language is a union of finitely many n-union-free languages. Based on these analogies, we present the definition in a general way (based also on [15]). However, it should be noted that while every regular language can be expressed as a union of a finite number of union-free and also as a union of a finite number of n-union-free languages, similar statement does not hold in general for the d-union-free languages (as proven in [8]), therefore, the d-union-complexity may be infinite although the studied language is regular. The following definition gives

back the original definition by the choice $x = \lambda$; gives the n-union-complexity with $x =$ n; moreover it also gives the d-union-complexity with $x =$ d.

Definition 3 (Union-complexity, n-union-complexity and d-union-complexity). *Let $x \in \{\lambda, n, d\}$. The form $L = \bigcup_{i=1}^{k} L_i$ is a minimal x-decomposition of the language L if each L_i is an x-union-free language and there is no $m < k$ such that $L = \bigcup_{i=1}^{m} K_i$, where each K_i is x-union-free. Then, k is called the* x-union-complexity *of the language L. However, in the case that L cannot be written in the form $\bigcup_{i=1}^{k} L_i$ with any natural number k for deterministic union-free languages L_i ($1 \leq i \leq k$), then L has an infinite deterministic union-complexity.*

The class of union-free languages is an interesting class including several languages since for each regular language L, the language L^* is union-free regular. We can summarise some others of the simplest known results about the union-complexities (see, e.g., [14]):

- The x-union-complexity of an x-union-free languages is at most 1 ($x \in \{\lambda,$ n, d$\}$); it is 0 for the empty language and 1 for every nonempty x-union-free language.
- For every finite language, its x-union-complexity is exactly the cardinality of the language.
- A language is regular if and only if its union-complexity is finite.
- A language is regular if and only if its n-union-complexity is finite.

As we already mentioned, the d-union-complexity could be infinite, as, e.g., one of the main results of [8] states:

Proposition 1. *The language defined by the regular expression $((a \cup b)(a \cup b))^*$ cannot be expressed as a union of a finitely many deterministic union-free languages.*

In [1] it has been proven that the union-complexity of regular languages is computable. However, the method is very complex and cannot be used in practical applications. Some bounds may be computed much faster, e.g., an x-decomposition (may also be called *x-union normal form*) of a regular language defines an upper bound for its x-union-complexity.

Before continuing with further more technical concepts, we are already at the stage that all the necessary concepts are shown to understand an example that could highlight the non-trivial nature of the problem we investigate here.

Example 1. Let us consider the language L defined by $(a^*b \cup dc^*)$. On the one hand, the language has 2 shortest words, b and d, and thus it is not union-free. On the other hand, Fig. 1 shows the two d-1CFPAs that accept a^*b and dc^*, respectively proving that L has union-complexity, n-union-complexity and d-union-complexity 2.

Fig. 1. Two deterministic 1-cycle-free-path-automata: the accepted languages, a^*b (left) and dc^* (right) are d-union-free.

Let us consider now L^+, i.e., $(a^*b \cup dc^*)^+$. As it has again 2 shortest words, b and d, this is neither union-free. On the one hand, one can easily check that L^+ is the union of the two languages accepted by 1CFPAs of Fig. 2, as the 1CFPA on the left accepts exactly those words of L^+ which start with a^*b and the 1CFPA on the right accepts exactly those which start with d. On the other hand, both of the 1CFPAs use λ-transitions, i.e., they are not n-1CFPAs and not d-1CFPAs.

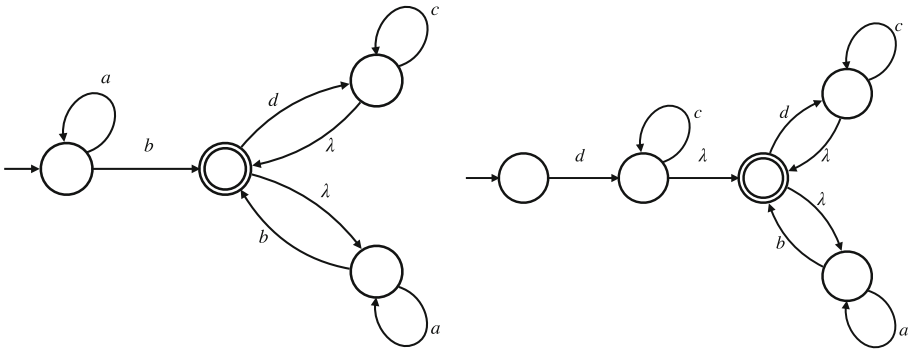

Fig. 2. Two nondeterministic 1-cycle-free-path-automata with λ-transitions such that the union of their accepted languages is exactly $(a^*b \cup dc^*)^+$.

Furthermore, we show that the n-union-complexity of L^+ is greater than 2. The proof is by contradiction, thus assume that there are 2 n-1CFPAs such that the union of their accepted languages is exactly L^+. As there are two shortest words in L^+, the backbone of one of the n-1CFPAs, let us say, **A**, must be S_AbF_A and the backbone of the other, let us say, **B**, must be S_BdF_B. The word db is also in L^+, thus one of the 1CFPAs **A** or **B** must accept it. The n-1CFPA accepts db, must use the above described backbone transition, and another transition to process the other letter, thus this other transition must be a self-loop transition. Thus, in the former case, there is a cycle in **A** as S_AdS_A, while in the latter case there is a cycle in **B** as F_BbF_B. Now, on the one hand, $ab \in L^+$, and this must be accepted by **A** which implies the cycle S_AaS_A in **A**. However, if **A** has also the cycle with letter d on its initial state (as we assumed in the first case), then **A** would also accept the word adb which is clearly not in L^+. Thus the first

case cannot hold, db cannot be accepted by **A**. Now, on the other hand, we have that $dc \in L^+$, and this word must be accepted by **B**. However, it implies the cycle $F_B c F_B$ in **B**. But, now, **B** would also accept $dbc \notin L^+$, which provides the contradiction and the proof that L^+ has a larger n-union-complexity than 2.

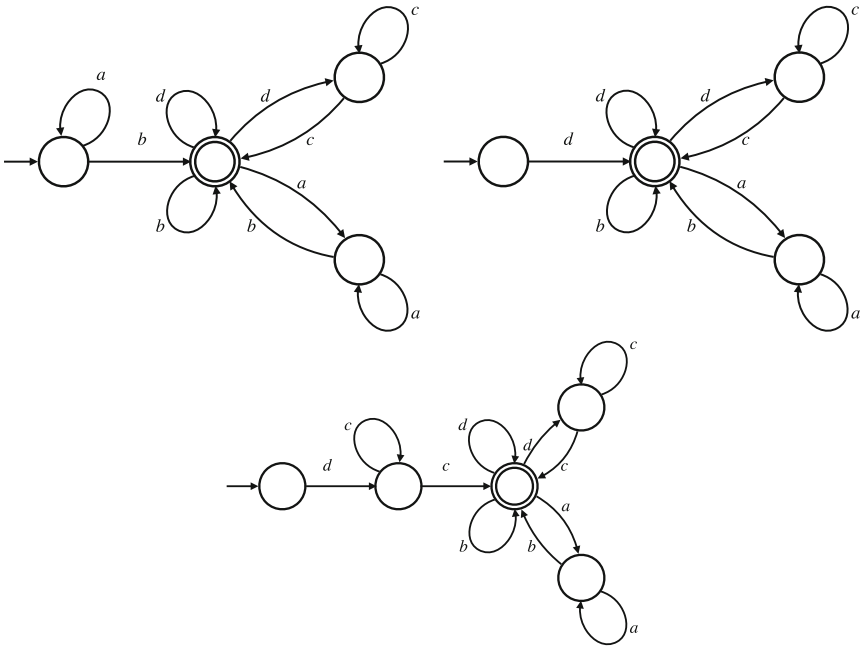

Fig. 3. Three n-1CPAs such that the union of their accepted languages is exactly $(a^*b \cup dc^*)^+$.

Now, to prove that L^+ has n-union-complexity 3, consider the union of the languages accepted by n-1CFPAs shown in Fig. 3. In fact the automaton on the left accepts the words of L^+ that start with a^*b, the n-1CFPA in the middle accepts those which start with d, but not with dc^+, while the n-1CFPA on the right is accepting the words that start with a word of dc^+.

Observing that the 1CFPAs we used in the previous descriptions are highly not deterministic, i.e., in many of them there are more than one transition from some states by the same letter, the d-union-complexity of the language L^+ could be even much higher than 3. We may conjecture it here, without any other explanations, that it is infinite.

Now, we also give new concepts, the tail (and tail-cycles) of the 1CFPAs and another technical concept, the branching (states and transitions).

Definition 4 (branching, head, tail). *A state $P \neq F$ of a 1CFPA is called a* branching state *if there are at least two different transitions from this state, i.e.,*

there is a transition which does not follow the only cycle-free accepting path from P, these transitions are called branching transitions. *The final state F is called a branching state if there is a transition from it, moreover, all of the transitions from F are branching transitions (as none of them is part of the empty path F, i.e., the shortest cycle-free path from F to F).*

If the initial state is a branching state, then the cycle(s) in one of the following forms are called head-cycles*:*

- *either it contains exactly one transition step of the form SaS with a letter $a \in \Sigma$; or*
- *it starts with a branching transition step SaR and continues with the cycle-free accepting path from R till S is reached again.*

A cycle starting from the final state F is called a tail-cycle *of the 1CFPA if it is in one of the following forms:*

- *either it contains exactly one transition step of the form FaF with a letter $a \in \Sigma$; or*
- *it starts with a transition step FaR ($R \neq F$) and continues with the cycle-free accepting path from R.*

If a 1CFPA does not have any tail-cycles, we say that it is tail-cycle-free *or* tailless *(or without a tail).*

The following facts are due to the structure of 1CFPAs.

- Any self-loop transition is a branching transition.
- If there is a branching transition from a state P on the backbone to another state R in the backbone, then R has a longer cycle-free accepting path than P has.
- If there is a branching transition from a state P on the backbone and its transition goes to the state R that is not on the backbone, then the cycle-free accepting path from R reaches the backbone before or on P, i.e., maybe some of the last steps of the cycle starting from P with the branching transition is already on the backbone to reach P again.
- If there is a branching transition from a state $P \neq F$ in a tail-cycle to another state R, then R has a longer cycle-free accepting path than P has, moreover, this cycle-free path arrives back to the tail-cycle before or in P (i.e., on one of the states that the cycle already touched after F by reaching P).
- A tail-cycle may reach the backbone in any of its states P and then, it must follow the backbone till reaching F. (In a special case, it may contain only F from the backbone.)
- A branching transition from a state P going to R always implies that all accepting paths from R will reach P again.
- The number of tail-cycles of a 1CFPA is the number of the (branching) transitions from F.

Now we also define another new concept, the set of substates:

Definition 5. *A state $R \neq P$ is a substate of P if there is a branching transition from P to R. Moreover, the states which are not in the shortest (i.e., cycle-free) path from S to P, but are in the cycle-free accepting path from R are also substates of P, the set of these states is denoted as $sub(P_R)$. Further, the set of states of the cycle starting with the branching transition from P to R and then following the cycle-free accepting path till P is reached again is denoted by $sub^{\circ}(P_R)$.*

We have the following:

- The states on the backbone are not substates of any state of the 1CFPA.
- Each state of an 1CFPA is either on the backbone or it is a substate of another state.
- A tail-cycle contains F, some of the substates of F (if there are more tail-cycles) and maybe some other states of the backbone.

The language $\emptyset$ is a very special language, its union-complexity is 0, as well as its n-union-complexity and d-union-complexity are also 0, since we need 0 languages to unite them to obtain it. On the other hand, the Kleene plus of $\emptyset$ is itself, that is, $\emptyset^+ = \emptyset$. There is not so much about to say this special languages, and thus, in the rest of the paper we may assume that the language we consider is not the empty one.

We recall some of the main results of [15], the operational union-complexity of the three regular operations, union, concatenation and Kleene star.

Proposition 2. *Let L_1 and L_2 be two regular languages with union-complexities n and m, respectively. Then the union of them, i.e., $L = L_1 \cup L_2$ could have the union-complexity at most $n + m$. Moreover, this bound is tight, i.e., for any two positive integers n, m, there are languages L_1 and L_2 with union-complexities n and m, such that their union has union-complexity exactly $n + m$.*

Proposition 3. *Let L_1 and L_2 be two regular languages with union-complexities n and m, respectively. Then the concatenation of them, i.e., $L = L_1 \cdot L_2$, could have the union-complexity at most $n \cdot m$, and this bound is tight.*

Proposition 4. *Let L be a regular language with union-complexity n. Then the language L^* has the union-complexity exactly 1 independently of the value of n.*

Now, we are ready to present our main results concerning the union-complexity of languages created by Kleene plus operation.

3 Operational Union-Complexity of the Kleene Plus Operation

First, we present the case of the general union-complexity, and then in subsections we show the cases of the n-union-complexity and the d-union-complexity. One of our main result, complementing the results shown in [15] is as follows.

Theorem 1. *Let L be a regular language with union-complexity n. Then the language L^+, the Kleene plus of L has a union-complexity of at most n.*

Proof. Obviously $L^+ = L \cdot L^*$. From Propositions 3 and 4 and one may establish the upper bound as $n \cdot 1 = n$.

To prove that this bound is tight, let us start with a language over an n-ary alphabet Σ_n. Let $L_n = \Sigma_n = \{a_1, \ldots, a_n\}$. Then, $L_n^+ = \Sigma_n^+$ has clearly n shortest words, i.e., the letters of the alphabet as words showing that its union-complexity cannot be less than n.

However, this construction uses a larger and larger alphabet with growing value of n. After this initial result on the tightness, let us consider the binary alphabet $\Sigma_2 = \{0, 1\}$. Let us encode n different letters with a binary block code. More precisely, let $L_2 = \{10^k, 10^{k-1}1, \ldots, 1w_i, \ldots, 1w_{n-1}\}$, where w_i is the binary representation of number i with k digits. (The value of k should be at least $\lceil \log_2(n) \rceil$ to make this possible.)

Clearly, L_2 is a finite language with n words, thus its union-complexity is n. Now, the union-complexity of L_2^+ can be estimated from below by the number of its shortest words which is n and gives the proof of the tightness already for the case of a binary alphabet. □

The unary alphabet plays some special importance and it is also interesting since already some of the closure properties of the union-free languages works in a different manner for this special case, consider, e.g., the closure under concatenation [14].

Although over the unary alphabet, the properties of the Kleene star and the Kleene plus are usually very similar, we intend to show that they work in a different way when the union-complexity is studied. It is well-known, and we have already mentioned, that L^* of any regular language has the union-complexity 1. Now we show that this is not the case with L^+ even if the regular language is over a unary alphabet.

Theorem 2. *The regular language L described by $a^4(a^9)^* \cup a^7(a^5)^*$ has the union-complexity 2. Further its Kleene plus, L^+ has also union-complexity 2.*

Proof. As 4 and 9 are co-primes, the language L_1 defined by $(a^4(a^9)^*)^+$ is co-finite, i.e., there are only finitely many natural numbers ℓ such that a^ℓ is not in the language L_1, and thus, the difference of a^* and L_1 is a finite language. A similar statement is true for $(a^7(a^5)^*)^+$. In the former, the following positive lengths are missing: 1, 2, 3, 5, 6, 7, 9, 10, 11, 14, 15, 18, 19, 23, 27. As usual over the unary alphabet, the words can be identified by their lengths. By a theorem of Frobenius (actually, Sylvester has published in the 1880's its solution for the case we need), the longest word that cannot be given in the form $4k_1 + 9k_2$ by nonnegative integer values of k_1 and k_2 is $4 \times 9 - (4 + 9) = 36 - 13 = 23$, however, we have the condition $k_1 \geq 1$ which shifts this limit a little bit. In fact, $(a^4(a^9)^*)^+$ has all the words of length $4k_1$ with $k_1 \geq 1$, all the words of length $4k_1 + 9$, $4k_1 + 18$ and $4k_1 + 27$. These for sets of integers contain all the integers $\ell > 27$.

For the following lengths of the above list, a word with length exists in the language L^+:

$7 : 7$, $11 : 4 + 7$,
$14 : 7 + 7$, $15 : 4 + 4 + 7$,
$18 : 4 + 7 + 7$, $19 : 7 + 7 + 5$,
$23 : 4 + 7 + 7 + 5$, $27 : 4 + 4 + 7 + 7 + 5$.

The co-finite language L^+ does not have the lengths 1, 2, 3, 5, 6, 9 and 10, i.e., $L^+ = \{a^4, a^7, a^8\} \cup \{a^k \mid k > 10\}$. Now, we show that L^+ is not union-free. If it would be, then the backbone word must be $aaaa$, as it is the shortest word of L^+. Further the 1CFPA must also accept a^7 meaning that there is a cycle accessible from the backbone by pumping 3 as into the word. However, then by doing this pumping cycle again, the word a^{10} would be obtained and accepted. This is contradicting to the fact that a^{10} is not in L^+. Thus, this language is not union-free, it has a union-complexity of at least 2. By applying Theorem 1, since L has union-complexity 2, it cannot be more than 2. Therefore, it has been proven that L^+ has union-complexity 2. □

The precise investigation of the unary case is left for the future:
Open Problem 1. Whether the result stated in Theorem 1 is also tight in the case of the unary alphabet for larger union-complexities, is left open.

3.1 On n-Union-Complexity

In this subsection, the n-union-complexity is studied. As we already mentioned, for each regular language, its n-union-complexity is a finite number. On the one hand, the closure of the class of the n-union-free languages under Kleene plus ([14]) gives the immediate corollary:

Corollary 1. *Let L be a language with n-union-complexity 1. Then, the n-union-complexity of the language L^+ is also 1.*

On the other hand, as we have seen in Example 1, the union-complexity and the n-union-complexity may behave in a different manner. Moreover, the constructions in the proofs of Propositions 3 and 4 were based on language operations (like regular expressions) [15], which can be translated to automata only with intensive usage of λ-transitions: remember that the class of n-union-free languages is not closed under concatenation. Thus, we may need to find new constructions to estimate the n-union-complexity.

Theorem 3. *Let a regular language L be given with an n-union-complexity k. Further, let $L_1, \ldots, L_k$ be some n-union-free languages such that $L = \bigcup_{i=1}^{k} L_i$. Let $\mathbf{A}_i$ be an n-1CFPA accepting L_i for each $1 \leq i \leq k$. Let the number of tail-cycles of $\mathbf{A}_i$ be t_i. Then, the n-union-complexity of L^+ is at most $m = k + \sum_{i=1}^{k} t_i$.*

Proof. The proof is a construction to show that L^+ is the union of m n-union-free languages. Based on the condition given in the theorem, let $\mathbf{A}_1, \ldots, \mathbf{A}_k$, be k n-1CFPAs that accept the languages $L_1, \ldots, L_k$ such that $L = \bigcup_{i=1}^{k} L_i$.

Let the structure of each $\mathbf{A}_i$ be given with its backbone states and the sets of the substates including the substates of the tail-cycles (if any). See Fig. 4, top left.

Now, let us have $t_i + 1$ copies of each $\mathbf{A}_i$, a copy $\mathbf{B}_i$ that is similar to the original, but does not include any of the tail-cycles of $\mathbf{A}_i$ (see Fig. 4, top right); and a copy $\mathbf{C}_i^\ell$ for each tail-cycle which are expanded variants of $\mathbf{A}_i$ as it is explained below (ℓ is the numbering of the tail-cycles of the automaton, $1 \leq \ell \leq t_i$). Let the backbone of $\mathbf{C}_i^\ell$ be $S_i^\ell a_i^\ell \ldots F_i^\ell b_i^\ell \overline{R}_i^\ell \ldots \overline{F}_i^\ell$ where $S_i^\ell a_i^\ell \ldots F_i^\ell$ is a copy of the backbone of $\mathbf{A}_i$ and $F_i^\ell b_i^\ell \ldots \overline{F}_i^\ell$ is a copy of the ℓ-th tail-cycle, that is starting with the copy of the branching transition $F_i^\ell b_i^\ell R_i^\ell$. Thus, the backbone of $\mathbf{C}_i^\ell$ contains a copy of each backbone state of $\mathbf{A}_i$ and also an (additional) overlined copy of the states of $sub^\circ(F_{i\,R_i^\ell}^{\ell})$. Let now all the cycles of $\mathbf{A}_i$ be added by adding the substates of each not overlined state, and their substates iteratively by their transitions including all tail-cycles from the state F_i^ℓ. The substates and the cycles of each overlined state should also be added, but the copy $\overline{F}_i^\ell$ of the final state (that actually is the final state of $\mathbf{C}_i^\ell$, but in this way, it will not be part of any cycles in $\mathbf{C}_i^\ell$. (See the second line of Fig. 4.)

So far, by our constructions, each 1CFPA $\mathbf{B}_i$ accepts exactly those words of the language L_i that are accepted without using any transitions of the final state (i.e., without using any of the tail-cycles); and each 1CFPA $\mathbf{C}_i^\ell$ accepts exactly those words of L_i which are accepted in such a way that from the final state the last used branching transition defines the tail-cycle ℓ in $\mathbf{A}_i$. Now let us take copies of each of these automata, from each one more as the number of its head-cycles (i.e., the number of branching transitions of the initial state), let these copies be $\underline{\mathbf{B}}_i^j$ and $\underline{\mathbf{C}}_i^{\ell,j}$, where j is a nonnegative integer not more than h_i, the number of head-cycles of $\mathbf{A}_i$. Let all $\underline{\mathbf{B}}_i^0$ and $\underline{\mathbf{C}}_i^{\ell,0}$ be identical to $\mathbf{B}_i$ and $\mathbf{C}_i^\ell$, respectively, but without any head-cycles (the transitions and the states of the head-cycles and their substates are simply removed, see the third line of Fig. 4). Further, let $\underline{\mathbf{B}}_i^j$ and $\underline{\mathbf{C}}_i^{\ell,j}$ with $j > 0$ be defined as follows. (In the next part we use $\ell = 0$ to index the states of $\underline{\mathbf{B}}_i^j$, while $\ell > 0$ for the states of $\underline{\mathbf{C}}_i^{\ell,j}$.) Let the backbone of these automata be $\tilde{S}_i^{\ell,j} b_{i,1}^{\ell,j} \ldots S_i^{\ell,j} a_{i,1}^\ell \ldots F_i^{\ell,j}$ where the part $\tilde{S}_i^{\ell,j} b_{i,1}^{\ell,j} \ldots S_i^{\ell,j}$ is a copy of the j-th head-cycle that starts with branching transition from $S_i^{\ell,j}$ in $\mathbf{A}_i$ and in this way it is becoming not a cycle, but part of the backbone in the new 1CFPA from $\tilde{S}_i^{\ell,j}$ with letter $b_{i,1}^{\ell,j}$; and the rest of the backbone, $S_i^{\ell,j} a_{i,1}^\ell \ldots F_i^{\ell,j}$, is the original backbone of $\underline{\mathbf{B}}_i$ or $\underline{\mathbf{C}}_i^\ell$. Further, all states of the 1CFPA $\underline{\mathbf{B}}_i$ or $\underline{\mathbf{C}}_i^\ell$ are kept, respectively, with their transitions. Also all the substates of the states of the j-th head-cycle (but the initial state) are copied and reached from the states of the first part $\tilde{S}_i^{\ell,j} b_{i,1}^{\ell,j} \ldots S_i^{\ell,j}$ of the backbone, with their transitions (between pairs of copied states). Moreover, all head-cycles and their substates

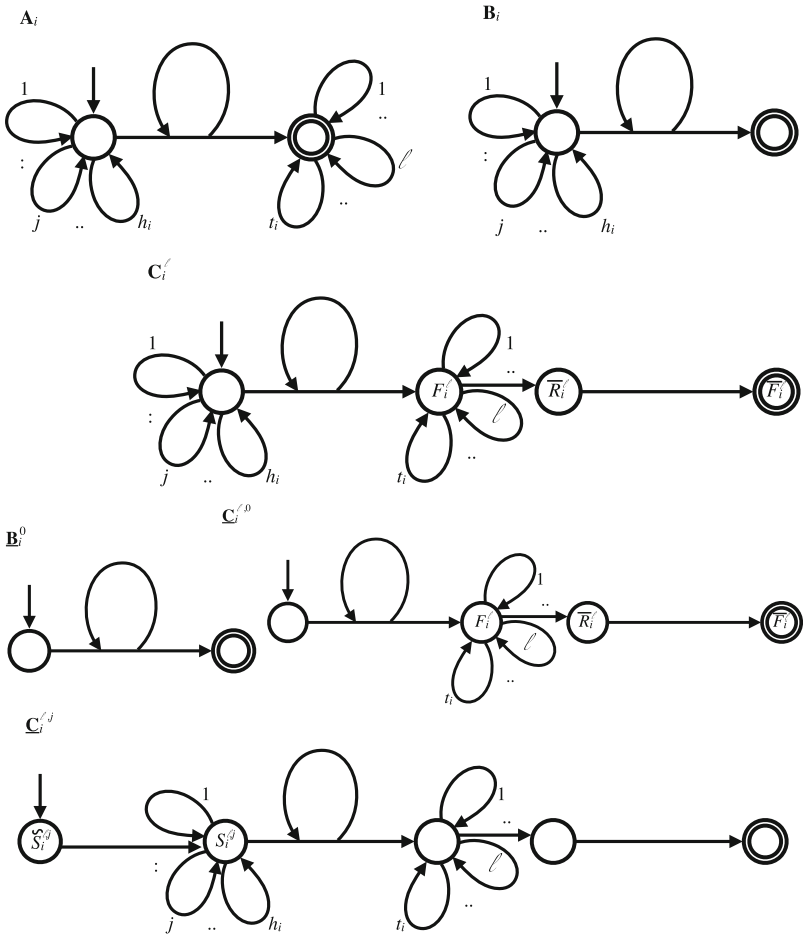

Fig. 4. Some parts of the construction in the proof of Theorem 3.

with their transitions are also copied with a branching transition from $S_i^{\ell,j}$. (See the bottom of Fig. 4.)

In this way, each 1CFPA $\underline{\mathbf{B}}_i^0$ accepts all words of L_i that can be accepted by a path neither containing any branching transition from the initial state of $\mathbf{A}_i$, nor from its final state. Further, each 1CFPA $\underline{\mathbf{B}}_i^j$ accepts exactly those words of L_i that are accepted by a path starting by the j-th branching transition (identifying the j-th head-cycle) from the initial state of $\mathbf{A}_i$ and do not use any branching transitions from the final state of $\mathbf{A}_i$. Also, each $\underline{\mathbf{C}}_i^{\ell,0}$ accepts those words of L_i that are accepted by using none of the head-cycles of $\mathbf{A}_i$, but the ℓ-th tail-cycle was used in the end of the word (the last used branching transition from the final state of $\mathbf{A}_i$ used the branching transition defining the tail-cycle ℓ). Finally, each $\underline{\mathbf{C}}_i^{\ell,j}$ accepts exactly those words of L_i that have an accepting path in $\mathbf{A}_i$

that starts with the j-th head-cycle that is defined by a branching transition at the initial state of $\mathbf{A}_i$ and the last branching transition at the final state used in the accepting path defining the tail-cycle ℓ.

We continue the construction by modifying every $\mathbf{B}_i$ and $\mathbf{C}_i^\ell$ in a very similar way. Thus, let us consider one, let us say $\mathbf{X}$ of the 1CFPAs $\mathbf{B}_i$ or $\mathbf{C}_i^\ell$, let its initial state be denoted by S and its final state be denoted by F. It is clear by the construction, that none of those 1CFPAs has tail-cycles, i.e., all of them are tailless. Now, consider all $\underline{\mathbf{B}}_i^j$ $(1 \leq i \leq k,\ 0 \leq j \leq h_i)$ and $\underline{\mathbf{C}}_i^{\ell,j}$ $(1 \leq i \leq k,\ 1 \leq \ell \leq t_i$, and $0 \leq j \leq h_i)$. For each of these 1CFPAs, let us say $\underline{\mathbf{Y}}$, we expand the 1CFPA $\mathbf{X}$ as follows. For the final state F, let us put a branching transition for each $\underline{\mathbf{Y}}$ copying all states and transitions of $\underline{\mathbf{Y}}$ into $\mathbf{X}$ such that all the transitions from the initial state of $\underline{\mathbf{Y}}$ are coming from F in $\mathbf{X}$, moreover all the transitions reaching the final state of $\underline{\mathbf{Y}}$ are going to F instead in $\mathbf{X}$.

Clearly, each of the constructed automata accepts only words of the language L^+. Moreover, any word of L^+ is accepted by at least one of the previously constructed automata, actually, depending on the structure of the first word that is used to compose the given word from the words of the languages L_i. □

Based on the fact that over a unary alphabet any union-free language is also an n-union-free language [14], we can restate and reformulate Theorem 2.

Corollary 2. *There is a regular language such that its Kleene plus has n-union-complexity greater than 1. For example, considering* $L = \{a^k \mid k = 4+9n$ *or* $k = 7+5n$ *for all* $n \geq 0\}$*, both the n-union-complexities of* L *and* L^+ *are* 2.

3.2 On Deterministic Union-Complexity

Now we give our result about deterministic union-complexity.

Theorem 4. *There is a regular language* L *with finite d-union-complexity such that* L^+ *has infinite d-union-complexity.*

Proof. Consider the language $L = \{aa, ab, ba, bb\}$, clearly its d-union-complexity is a 4. Now, let us consider $L^+ = (aa \cup ab \cup ba \cup bb)^+$, which actually contains all nonempty words over $\Sigma = \{a, b\}$ with even lengths. Moreover, every word of Σ^* is a prefix of some words of L^+, thus based on an analogous proof of Proposition 1, L^+ cannot be written as a finite union of deterministic union-free languages [8].

Open Problem 2. Is there any language L with smaller d-union-complexity than 4 such that its Kleene plus, L^+ has already infinite d-union-complexity? May, e.g., the language of Example 1 have this property?

Since the class of deterministic union-free languages is not closed under any of the usual language operations (union, complement, concatenation, Kleene star etc., see [8,13]), it seems to be a non-trivial task, to find operational d-union-complexity of languages.

References

1. Afonin, S., Golomazov, D.: Minimal union-free decompositions of regular languages. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 83–92. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00982-2_7
2. Brzozwski, J.A.: Regular expression techniques for sequential circuits. Ph.D Dissertation, Department of Electrical Engineering, Princeton University, Princeton, NJ, June 1962
3. Brzozowski, J.A., Davies, S.: Most complex deterministic union-free regular languages. In: Konstantinidis, S., Pighizzini, G. (eds.) DCFS 2018. LNCS, vol. 10952, pp. 37–48. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94631-3_4
4. Crvenković, S., Dolinka, I., Ésik, Z.: On equations for union-free regular languages. Inform. Comput. **164**(1), 152–172 (2001)
5. Holzer, M., Kutrib, M.: Structure and complexity of some subregular language families. In: The Role of Theory in Computer Science, pp. 59–82 (2017)
6. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley Publishing Company, Reading MA (1979)
7. Jirásková, G., Masopust, T.: Complexity in union-free regular languages. Int. J. Found. Comput. Sci. **22**, 1639–1653 (2011)
8. Jirásková, G., Nagy, B.: On union-free and deterministic union-free languages. In: Baeten, J.C.M., Ball, T., de Boer, F.S. (eds.) TCS 2012. LNCS, vol. 7604, pp. 179–192. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33475-7_13
9. Kutrib, M., Wendlandt, M.: Expressive capacity of subregular expressions. RAIRO ITA: Theory Inf. Appl. **52**(2–3–4), 201–218 (2018)
10. Nagy, B.: A normal form for regular expressions. In: Calude, C.S., Calude, E., Dinnen M.J. (eds.) Supplemental Papers for DLT 2004 (8th International Conference Developments in Language Theory), CDMTCS Report 252, Auckland, pp. 51–60 (2004)
11. Nagy, B.: Union-free regular languages and 1-cycle-free-path-automata. Publ. Math. Debrecen **68**, 183–197 (2006)
12. Nagy, B.: On union-complexity of regular languages, CINTI. In: 11th IEEE International Symposium on Computational Intelligence and Informatics **2010**, pp. 177–182 (2010)
13. Nagy, B.: Union-freeness, deterministic union-freeness and union-complexity. In: Hospodár, M., Jirásková, G., Konstantinidis, S. (eds.) DCFS 2019. LNCS, vol. 11612, pp. 46–56. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23247-4_3
14. Nagy, B.: Union-freeness revisited – between deterministic and non-deterministic union-free languages. Int. J. Found. Comput. Sci. **32**, 551–573 (2021)
15. Nagy, B.: Operational union-complexity. Inform. Comput. **284**, 104692 (2022)
16. Shallit, J.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press, Cambridge (2008)
17. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, pp. 41–110. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59136-5_2