On Union-Free and Deterministic Union-Free Languages

Galina Jirásková [1,⋆] and Benedek Nagy [2,⋆⋆]

[1] Mathematical Institute, Slovak Academy of Sciences,
Grešákova 6, 040 01 Košice, Slovakia
jiraskov@saske.sk
[2] Department of Computer Science, Faculty of Informatics,
University of Debrecen, Debrecen, Hungary
nbenedek@inf.unideb.hu

Abstract. The paper continues the study of union-free and deterministic union-free languages. In contrast with the fact that every regular language can be described as a finite union of union-free languages, we show that the finite unions of deterministic union-free languages define a proper subfamily of regular languages. Then we examine the properties of this subfamily.

1 Introduction

The regular languages are the most common, well-known, and well-applicable languages. They are the simplest languages in the Chomsky-hierarchy, and can be represented by regular expressions or finite automata. Nowadays, some subfamilies of the regular languages have become important in various fields [4–6]. One of them is the family of *union-free languages*: those languages described by regular expressions without the union operation.

Since the Parikh images of regular languages coincide with semi-linear sets, it is an interesting question if the Parikh images of languages in a subregular class still contain every semi-linear set. We address this question in Section 3, and show that only certain special semi-linear sets — the so-called *conditional-linear sets* — can be obtained by Parikh images of union-free languages.

The union-free languages are accepted by special nondeterministic finite automata, the so called *one-cycle-free-path automata*, in which there is exactly one cycle-free path from each state to the final state [10]. The deterministic versions of one-cycle-free-path automata are not as powerful, and define the class of deterministic union-free languages [8].

⋆ Research supported by the Slovak Research and Development Agency under contract APVV-0035-10 "Algorithms, Automata, and Discrete Data Structures".

⋆⋆ Research supported by the TÁMOP 4.2.1/B-09/1/KONV-2010-0007 project. The project is implemented through the New Hungary Development Plan, co-financed by the European Social Fund and the European Regional Development Fund.

J.C.M. Baeten, T. Ball, and F.S. de Boer (Eds.): TCS 2012, LNCS 7604, pp. 179–192, 2012.

One of the most important results on union-free languages states that every regular language can be expressed as a union of a finite number of union-free languages [9]. Moreover, the minimal number of required union-free languages is algorithmically computable [3].

Motivated by these results, we address an analogous question for deterministic union-free languages, and provide a negative answer by describing a regular language which cannot be expressed as a union of a finite number of deterministic union-free languages. Our proof uses the representation of deterministic union-free languages by so-called balloon automata.

We also define the classes dU_n as the classes of regular languages that can be described as a union of n deterministic union-free languages, and show that they define a proper hierarchy.

Finally, we consider the class dU_* as the union of classes dU_n, and conclude the paper with some closure properties.

2 Preliminaries

We assume that the reader is familiar with the basic concepts of formal languages and automata theory. For all unexplained notions, we refer the reader to [7, 11].

For a finite non-empty set of symbols Σ, called an alphabet, Σ^* denotes the set of all strings over Σ including the empty string ε. A *language* over Σ is any subset of Σ^*. We denote the size of a finite set A by $|A|$ and its powerset by 2^A.

A *regular expression* over an alphabet Σ is defined inductively as follows: $\emptyset$, ε, and a, for a in Σ, are regular expressions. If r and t are regular expressions, then also $(r+t)$, $(r \cdot t)$, and $(r)^*$ are regular expressions. A regular expression is *union-free* if no symbol $+$ occurs in it. A regular language is *union-free* if there exists a union-free regular expression describing the language.

A *nondeterministic finite automaton* (nfa) is a quintuple $M = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, s is the initial state, F is the set of accepting states, and δ is the transition function that maps $Q \times (\Sigma \cup \{\varepsilon\})$ into 2^Q. The transition function is naturally extended to the domain $Q \times \Sigma^*$. The language *accepted* by nfa M is $L(M) = \{w \in \Sigma^* \mid \delta(s, w) \cap F \neq \emptyset\}$.

Automaton M is *deterministic* (dfa) if it has no ε-transitions, and $|\delta(q, a)| \leq 1$ for all states q in Q and symbols a in Σ. Hence, we consider incomplete dfa's, in which some transitions may be undefined.

A path from state p to state q in an nfa/dfa M is a sequence $p_0 a_1 p_1 a_2 \cdots a_n p_n$, where $p_0 = p$, $p_n = q$, and $p_i \in \delta(p_{i-1}, a_i)$ for $i = 1, 2, \ldots, n$. The length of this path is n; note that p_0 is a path of length 0. The path is a *cycle* if $n > 0$ and $p_0 = p_n$. The path is called a *cycle-free accepting path* if p_n is an accepting state, and either the path is of length 0, or $p_i \neq p_j$ whenever $i \neq j$. An nfa/dfa is a *one-cycle-free-path* (1cfp) nfa/dfa if there is a unique accepting cycle-free path from each of its states. It is known that the family of union-free languages coincides with the set of languages accepted by 1cfp nfa's [10]. The 1cfp dfa's define the family of *deterministic union-free* languages [8].

The accepting path from the initial state of a 1cfp automaton defines the *backbone* of the automaton, and the string accepted by this path is called the *backbone string*. The backbone string is contained in each string of the accepted language in a scattered way, and therefore the shortest string is *unique* in every union-free language [10].

3 On Parikh Images of Union-Free Languages

The Parikh image of a string w over an ordered alphabet $\{a_1, \ldots, a_k\}$ is the vector $\Psi(w) = (m_1, \ldots, m_k)$ of non-negative integers such that m_i is the number of occurrences of a_i in w. The Parikh image of a language L is the set of vectors $\Psi(L) = \{\Psi(w) \mid w \in L\}$.

A set of the form $\{\alpha_0 + n_1\alpha_1 + \cdots + n_m\alpha_m \mid n_j \geq 0 \text{ for } j = 1, 2, \ldots, m\}$, where $\alpha_0, \alpha_1, \ldots, \alpha_m$ are vectors of non-negative integers, is said to be a *linear* set. A *semilinear* set is a finite union of linear sets. It is well-known that the Parikh images of regular languages coincide with semilinear sets.

The aim of this section is to analyse the Parikh images of union-free languages. We show that the Parikh images of union-free languages are somewhere between linear and semilinear sets. Then we define the so-called conditional-linear sets, and prove that such sets coincide with the Parikh image of union-free languages.

Lemma 1. *Every linear set is the Parikh image of a union-free language.*

Proof. Let a linear set be given by the vectors α_j for $j = 0, 1, \ldots, m$. For each vector α_j, consider a string w_j with Parikh image α_j. First, construct an nfa accepting the singleton set $\{w_0\}$. Now, in the final state, add m disjoint cycles labelled by strings w_j for $j = 1, \ldots, m$. This results in a one-cycle-free-path nfa accepting a union-free language with desired Parikh image. □

We have seen that the set of Parikh images of union-free languages is a superset of linear sets, now we give 'upper and lower bounds':

Lemma 2. *There exists a union-free language whose Parikh image is not linear.*

Proof. Consider the language given by the regular expression $a(bb(aab)^*ba)^*a$. Its Parikh image is $\{(2,0)+n(1,3) \mid n \in \mathbb{N}\} \cup \{(3,3)+m(1,3)+k(2,1) \mid m, k \in \mathbb{N}\}$. This set cannot be described by only one linear set. Moreover, this language is even deterministic union-free. □

Lemma 3. *There exists a semilinear set W such that there is no union-free language L with $\Psi(L) = W$.*

Proof. Consider the semilinear set $W = \{(3,1), (4,0)\}$. Every language with Parikh image W is finite and contains at least two shortest strings. □

Lemma 4. *Let a language be accepted by a one-cycle-free-path nfa that consists of a backbone and several, up to starting state, disjoint cycles starting and ending in a state of the backbone; each cycle contains just one state of the backbone. Then the Parikh image of the language is a linear set.*

Proof. The Parikh image of the language is a linear set given by the Parikh vector of the backbone, and the Parikh vectors of the labels of all the cycles. □

The previous result can be stated as follows: The Parikh image of a union-free language with star-height 1 is linear. Now we define conditional-linear sets and show that they coincide exactly with Parikh images of union-free languages.

Definition 1. *A set of vectors W is conditional-linear if every vector α is in W if and only if it can be written in the form*

$$\alpha = \alpha_0 + \delta_1 n_1 \alpha_1 + \delta_2 n_2 \alpha_i + \cdots + \delta_m n_m \alpha_m,$$

where n_j are non-negative integers and α_j are fixed vectors of non-negative integers, and δ_i are conditional coefficients defined in the following way: $\delta_1 = 1$, and if $i > 1$, then δ_i is either without any condition and equals 1, or depends on the coefficient of some α_j with $j < i$, and in such a case it is equal to 1 if $\delta_j n_j > 0$ and to 0 if $\delta_j n_j = 0$:

$$\delta_i = 1, \qquad \text{if there is no condition for } \alpha_i,$$

$$\delta_i = \begin{cases} 1, & \text{if } \delta_j n_j > 0, \\ 0, & \text{if } \delta_j n_j = 0, \end{cases} \qquad \text{if } \alpha_i \text{ depends on the coefficient of } \alpha_j.$$

Having $\delta_i = 1$ for all i without any conditions, the linear sets can be obtained. Thus conditional-linear sets are a kind of extension of linear sets. Moreover, all conditional linear sets are semilinear; however not every semilinear set is conditional linear. In a conditional-linear set some vectors α_i have conditions, i.e., they can be present only if another vector with a smaller index is present.

Theorem 1. *Conditional-linear sets coincide with the Parikh images of union-free languages.*

Proof. Let a union-free language be given by a regular expression

$$x_1 (r_1)^* x_2 \cdots x_n (r_n)^* x_{n+1},$$

where the parts x_i are star-free; it is allowed that $x_i = \varepsilon$ for some i. Let α_0 be the Parikh vector of $x_1 x_2 \cdots x_{n+1}$, that is, of the backbone string; these symbols are not under any Kleene-star in the tree for the expression. Next, let α_i be the Parikh image of the string obtained from the child-subexpression r_i by substituting ε for the parts under Kleene-star inside. These symbols are below exactly one Kleene-star in the tree. The obtained vectors α_i are without conditions, because their mother expression is the original expression, so let $\delta_1 = \cdots = \delta_n = 1$.

Now, for any subexpression r^* which has not been considered yet, using the order of subexpressions as they are in the tree form of the regular expression starting from the top, let the next vector α_i be the Parikh image of the symbols under this Kleene-star using ε for its child-subexpressions of the form p^*. Let the conditional coefficient δ_i depend on the coefficient of the vector α_j where j is the index of the mother subexpression. Since the union-free expression contains finitely many stars this procedure terminates after finitely many steps.

Next, let a conditional-linear set W be given by its vectors and δ's. Construct a one-cycle-free-path nfa with Parikh image W as follows. Let A be the automaton accepting only a backbone string with Parikh vector α_0. Consider every vector α_i which is without condition, that is, $\delta_i = 1$ independently on any other values. Add a cycle labelled by a string with Parikh vector α_i starting and ending at the same state of the backbone of A.

Then consider each other vector α_i in order of increasing i. Let the condition for this vector depend on the coefficient of a vector α_j. The cycle for vector α_j is already included in the automaton, since $j < i$. Add new states to form a subcycle labelled by a string with Parikh vector α_i in a state of the cycle for α_j; it always can be done by adding new states and ε-transitions to the cycle for α_j. Finally, after adding subcycles for all the vectors, we get a one-cycle-free-path nfa that accepts a language with Parikh image W. □

4 Representation of Deterministic Union-Free Languages

Let us turn our attention to deterministic union-free languages defined as languages accepted by deterministic one-cycle-free-path dfa's [8].

First we fix further terms used in the paper. A state p of a 1cfp automaton is a *branching state* if at least two transitions are defined from state p, thus if $\delta(p, a) = q_1$ and $\delta(p, b) = q_2$, where q_1, q_2 are states of the automaton and a, b are symbols in the input alphabet such that $a \neq b$. Since the automaton is a 1cfp automaton, we must have $q_1 \neq q_2$. The accepting state is a *branching state* if there is at least one transition defined from it. If p is not the accepting state, then exactly one transition from p lies on the cycle-free accepting path from state p. All the other transition defined in state p start different cycles. All the transitions in the accepting state start new cycles.

Let a path paq contain a transition (p, a, q) starting a cycle. In the case $p = q$, the cycle has length 1. Otherwise, let us consider the cycle-free accepting path from state q. This path must contain state p, thus the path is of the form qz_1pz_2, where $z_1 \in \Sigma(Q\Sigma)^*$ and $z_2 \in (\Sigma Q)^*$. The cycle $paqz_1p$ is a *starting cycle* at state p. As an example, consider the cycles $4a6b4$ and $4c2a3a4$ that are starting cycles at state 4 in Fig. 2 (left).

Cycles may only start at branching states. The state where a cycle contains a previously (surely) visited state is the *returning state* of the cycle. A cycle may return to a previously (surely) visited state in various ways: A cycle may return at the same state as it starts like the cycle $4a6b4$ in Fig. 2 (left), or a

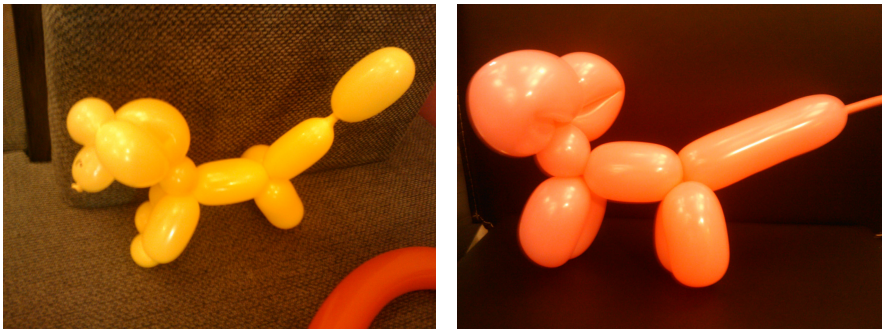

Fig. 1. A toydog from long balloon

cycle may return to another state like the cycle $2b0a1a2$ in Fig. 2; here state 0 is the returning state.

Since a one-cycle-free-path dfa over a unary alphabet cannot have a branching state, except the accepting state, the automaton may only have one cycle starting at the accepting state. Consequently, a deterministic union-free language over a unary alphabet either contains at most one string or is of the form $a^m(a^n)^*$ for some positive integers m, n. The other direction also holds: All these languages are accepted by 1cfp dfa's. Therefore, the unary case seems to be not so interesting. In what follows, we always assume that an alphabet has at least two symbols.

There can be several one-cycle-free-path dfa's for the same language. One of them plays an important role for us. We call it a *balloon* automaton. This name comes from the toys made by clowns for children from long balloons; for example, toydog (see Fig. 1).

Now we give the definition of a balloon automaton. We use balloon automata later to get regular expressions for deterministic union-free languages.

Definition 2 (Balloon DFA). *A backbone 1cfp dfa, that is, a dfa consisting of states of the backbone connected through the symbols of the backbone string is a balloon dfa. If A is a balloon automaton, then any extension of A obtained in the following way is a balloon automaton:*

- *pick a state p of A and a string $a_1 \cdots a_k$ of length k with $k \geq 1$ such that there is no transition on a_1 from p in A;*
- *add $k-1$ new states $p_1, \ldots, p_{k-1}$ to A connected through transitions*

$$p \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} p_{k-1} \xrightarrow{a_k} p.$$

Thus a deterministic one-cycle-free-path automaton is called a balloon dfa, if every starting cycle returns at the same (branching) state as it starts. Fig. 2 shows a one-cycle-free-path dfa and the corresponding balloon automaton.

The following result helps us to characterize deterministic union-free languages by regular expressions.

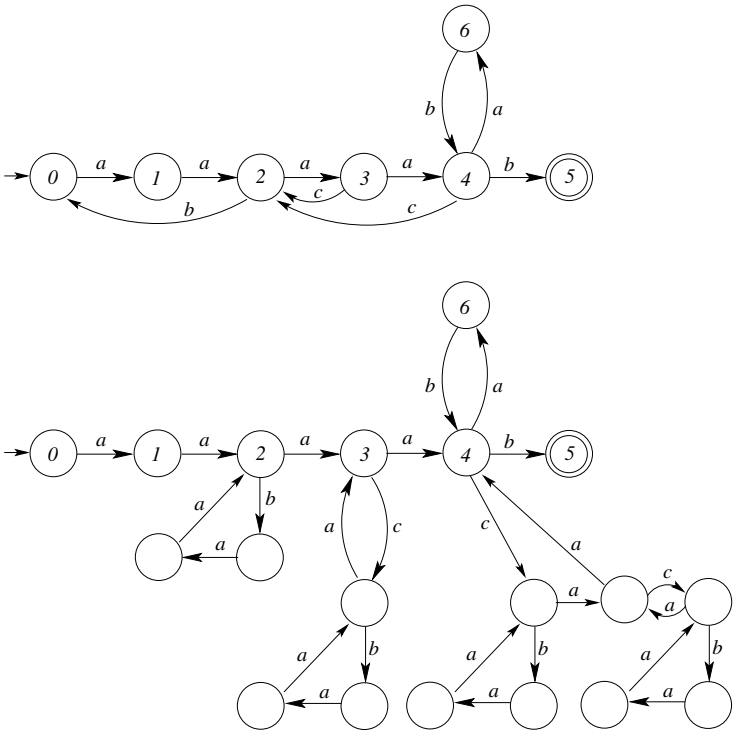

Fig. 2. A 1cfp dfa (up) and the corresponding balloon automaton (down)

Theorem 2. *Every deterministic union-free language is accepted by a balloon dfa.*

Proof. Let a deterministic union-free language L be given by a 1cfp dfa A. We are going to construct a regular expression and a balloon automaton for L in parallel from 1cfp dfa A. We use mixed-form expressions that contain symbols and names of the states of the automaton alternatively. We start with the expression $r_0 = sx_1q_1x_2 \cdots q_{m-1}x_mf$ that contains the symbols of the backbone string $x_1x_2 \cdots x_m$ and the names of the states $s, q_1, \ldots, q_{m-1}, f$ that occur on the backbone. All the states in expression r_0 are pairwise distinct.

As an example, consider the 1cfp dfa shown in Fig. 2. In this example, we have $r_0 = 0a1a2a3a4b5$.

The construction of the balloon automaton starts with the deterministic automaton A_0 that accepts only the backbone string: accordingly, it has states $s', q'_1, \ldots, q'_{m-1}, f'$.

We continue recursively while the automaton has branching states, that is, states that go to at least two distinct states by some symbols.

If the initial automaton A has no branching state, then it only accepts the backbone string, and it is a balloon automaton.

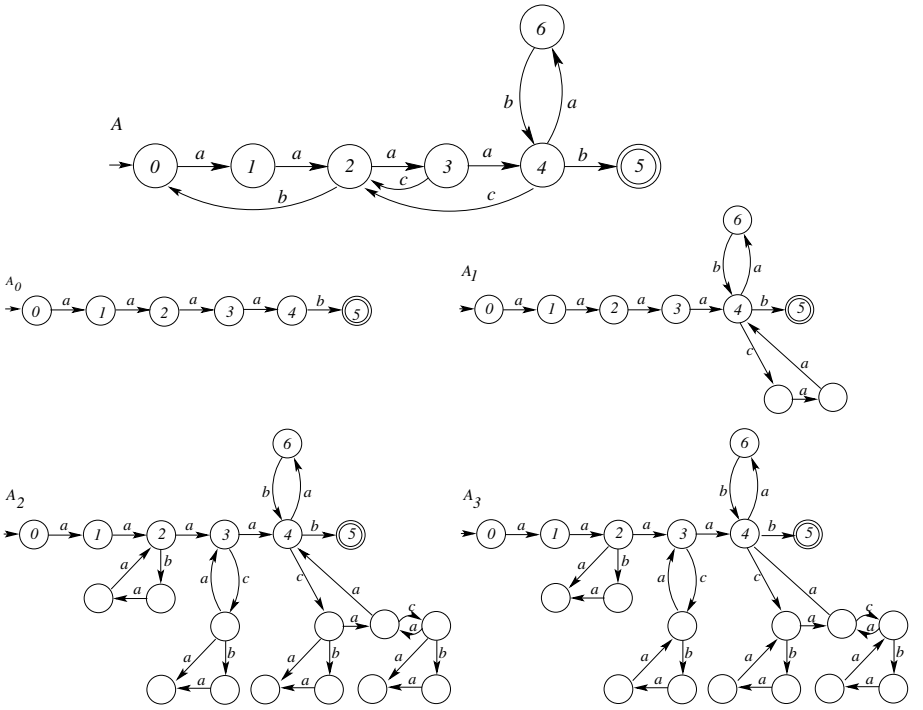

Fig. 3. The construction of a balloon automaton

Assume that our current expression is r_i, and our current automaton is A_i. Construct expression r_{i+1} by modifying expression r_i as follows. Choose the last branching state p that has not been considered yet, that is, the branching state that has the last occurrence in the mixed form r_i among the not yet considered mixed states. Put a pair of brackets into expression r_i to the points immediately after each occurrence of p. Put a star after the brackets, that is, use the following form:

$$u_1 p()^* u_2 \cdots u_m p()^* u_{m+1},$$

where p has m occurrences in r_i, and u_j's do not contain any p. Then put into these brackets as many sequences $()^*$ as many starting cycles exist at state p. After this, write into these brackets the mixed form expressions for the starting cycles using all the starting branches at state p without symbol p. The resulting expression is r_{i+1}. If only one cycle starts from a branching state, then the form $((z)^*)^*$ can be simplified to $(z)^*$, where z denotes the mixed form representing the cycle. In our example, we have

$$r_1 = 0a1a2a3a4((a6b)^*(c2a3a)^*)^*b5,$$
$$r_2 = 0a1a2a3(c2a)^*a4((a6b)^*(c2a3(c2a)^*a)^*)^*b5,$$
$$r_3 = 0a1a2(b0a1a)^*a3(c2(b0a1a)^*a)^*a4((a6b)^*(c2(b0a1a)^*a3(c2(b0a1a)^*a)^*a)^*)^*b5.$$

Notice that after each step, the number of occurrences of p remains m; and all the states which occur in, say, k cycles starting from p have km new occurrences. Moreover, since the automaton is a 1cfp dfa, there are no states appearing in the new brackets which have already been considered.

Now construct automaton A_{i+1} corresponding to expression r_{i+1} by modifying automaton A_i. Extend automaton A_i by cycles corresponding to the new subexpressions $(z)^*$; the ones that are not in r_i, but which are in r_{i+1}. For each of these new subexpressions, add a new cycle to every copy of state p' starting and ending in this state (in a similar way as we draw the backbone). In this step, we use various copies of the original states. Fig. 3 shows automata A_i in our example.

Repeat the above procedure until no more branching states exist. Since there are only finitely many branching states in the given automaton, this procedure terminates after finitely many steps. Let r_n be the resulting expression. After deleting the names of the states in r_n, we get a union-free expression which describes language L. The resulting automaton A_n accepts language L. Moreover, after each step, automaton A_i is a balloon automaton. □

The proof of the previous theorem gives also a regular expression for every one-cycle-free-path dfa. By this construction, we can characterize the deterministic union-free languages by regular expressions as follows. Every deterministic union-free language can be expressed by a regular expression of the following form:

(i) there is at most 1 symbol that continues the expression "in the same level";
(ii) the other symbols may enter for brackets: if there is one such symbol, then we use b: $(b\cdots)^*$; if there are two or more symbols, then we use $b_1, b_2, \ldots, b_\ell$: $((b_1\cdots)^*(b_2\cdots)^*\cdots(b_\ell\cdots)^*)^*)$.

On the other hand, every regular expression that satisfies (i) and (ii) describes a deterministic union-free language since a 1cfp dfa for such a language can be constructed by our proof.

We conjecture that the balloon dfa has maximal number of states among all the incomplete dfa's that accept the language and for each of its state there is exactly one accepting cycle free path.

A rough idea for the proof of the conjecture could be the following: By the construction of the balloon automata for a 1cfp dfa, the number of its states is not smaller than the number of the states of the original automaton. Moreover, the balloon automaton cannot be extended further by states. Since it is deterministic, by adding a new state, some new transitions are needed. This modifies the backbone or a cycle, or creates a new cycle; thus modifies the accepted language.

One of the most important complexity measures of regular languages is the star height that is connected to the number of nested stars in the regular expressions [11]. To measure the complexity of balloon automata, we define the following concept.

Definition 3 (Cycle Depth of Balloon Automata). *The cycle depth of a balloon automaton is the maximal number of its nested cycles.*

The cycle depth of a balloon automaton is 0 for cycle-free automata. For example, the depth of the automaton A_0 in Figure 3 is 0. Such automata only accept singleton languages. The cycle depth of A_1, A_2, A_3 is 1, 2 and 3, respectively.

The cycle depth of the balloon automata and the (nested) star-height of the obtained regular expressions have a strong relation: Actually, if expressions of the form $((r_1)^*(r_2)^*)^*$ are rewritten of the form $(r_1 + r_2)^*$, then this new expression has the same star height as the cycle depth of the original balloon automaton.

A cycle pzp of a 1cfp dfa, where $z \in \Sigma(Q\Sigma)^*$, is called an *inner cycle* if there is no branching state in z. For example, the cycle $2b0a1a2$ in Figure 2 is an inner cycle. In balloon automata, these cycles are at the deepest level. For example, the cycles of length 2 of A_2 in Figure 3 are inner cycles.

We have some important observations about graphs of 1cfp deterministic automata, in particular, we formulate them for balloon automata.

Lemma 5. *Let $A = (Q, \Sigma, \delta, s, f)$ be a 1cfp dfa accepting a non-empty language. Then*

1. *The branching factor at each state of A is at most $|\Sigma|$.*
2. *If the cycle depth of a balloon automaton is zero, then there is a state, in which at most one symbol in Σ defines a transition; the transition on the other symbols are undefined.*
3. *If the cycle depth of a balloon automaton is at least one, and the length of the/an inner cycle is at least two, then there is a state where only one of the symbols defines a transition.*

Proof. 1. In every state of a dfa, at most $|\Sigma|$ transitions may be defined.

2. Balloon automata with cycle depth zero accept exactly one string, and all the transitions in the final state f are undefined.

3. If the cycle depth of a balloon automaton is one, then there is a cycle starting from a state on the backbone with, by our assumption, length at least two. This means that after the first transition of this cycle, there is a state that is outside of the backbone. In this inner state, exactly one transition is defined.

Now assume that the cycle depth of a balloon automaton is at least two. The argument is quite similar to the previous case. In a balloon automaton, one transition could go to the direction of the final state; the first symbol of the unique cycle-free accepting path gives this transition. All the other symbols may start a new cycle going more deeply in the cycle depth of the automaton. Consider the/an inner cycle; its length is at least two by the assumption of the lemma. In this cycle, there is no new starting cycle. Therefore, in the/an inner state, there exists only one transition, which starts the cycle-free accepting path from that state. □

We conclude this section with the following result showing that the class of deterministic union-free languages is not closed under basic regular operations.

Theorem 3 (Closure properties). *The class of deterministic union-free languages is not closed under boolean operations, concatenation, square, star, reversal, cyclic shift, homomorphism, and inverse morphism.*

Proof. For each operation, we present deterministic union-free languages such that the language resulting from the operation is not deterministic union-free. Recall that if a language has at least two shortest strings, or if its minimal dfa has at least two final states, then the language is not deterministic union-free.

Complement:	$\{\varepsilon\}^c$	$= \Sigma^+$,
Union:	$\{a\} \cup \{b\}$	$= \{a, b\}$,
Intersection:	$b^*ab^* \cap a^*ba^*$	$\subseteq \{ab, ba\} \cup \{a, b\}^{\geq 3}$
Symmetric difference:	$\{a\} \oplus \{b\}$	$= \{a, b\}$,
Cyclic shift:	$\text{Shift}(\{ab\})$	$= \{ab, ba\}$,
Shuffle:	$\{a\} \sqcup\!\sqcup \{b\}$	$= \{ab, ba\}$,
Inverse morphism:	$h^{-1}(\{aa\})$	$= \{aa, ab, ba, bb\}$

$h(a) = h(b) = a$

For square and concatenation, consider the deterministic union-free language L_1 accepted by the 1cfp dfa shown in Fig. 4 (left). For reversal, consider the deterministic union-free language L_2 accepted by the 1cfp dfa shown in Fig. 4 (middle), and for star, the deterministic union-free language L_3 accepted by the 1cfp dfa shown in Fig. 4 (right). The minimal dfa's for L_1^2, L_2^R, and L_3^* have two final states, and therefore the resulting languages are not deterministic union-free.

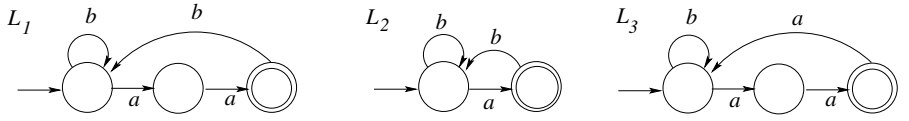

Fig. 4. The 1cfp dfa languages L_1, L_2, L_3 such that L_1^2, L_2^R, and L_3^* are not deterministic union-free

For homomorphisms, consider the deterministic union-free language ab^*ac^* and homomorphism $h(a) = a, h(b) = ab, h(c) = c$. Then $h(ab^*ac^*) = a(ab)^*ac^*$, the minimal dfa for which has two final states. □

5 Finite Union of Deterministic Union-Free Languages

Every regular language can be expressed as the union of a finite number of union-free languages [9]. This is one of the most important results on union-free languages. We can ask whether or not a similar result also holds for *deterministic* union-free languages. The next theorem provides a negative answer, and states one of the main results of this paper.

Theorem 4. *The language L described by the regular expression $((a+b)(a+b))^*$ cannot be expressed as a union of a finite number of deterministic union-free languages.*

Proof. The language L contains exactly the strings over $\{a, b\}$ of an even length. Therefore, each string over $\{a, b\}$ is a prefix of infinitely many strings in L.

Assume for the contradiction that L is given as a finite union of some deterministic union-free languages. Let us consider the balloon automata for these languages. None of these automata has a cycle of length one because otherwise this loop would allow to pump some strings of the accepted language symbol by symbol which would lead to the acceptance of strings of an odd length.

Now let us order all the balloon automata in a list $A_1, A_2, \ldots, A_\ell$. By Lemma 5, each of these automata contains at least one state, in which at most one symbol defines a transition. Moreover, such a state is reachable from every state of a balloon automaton: If there is no cycle at the final state, then the final state is such a state. If there is a cycle starting at the final state, then there is an inner cycle here with at least one such state. And since this state is reachable from the final state, it can be reached from each state of the automaton. Let q_i refer to such a state in automaton A_i.

Let us construct a string w in the following way: Let the prefix of w be the string w_1 that leads to the state q_1 in automaton A_1. Then let the next symbol of w be the one for which there is no transition from q_1. Now let $i = 2$, and hence the next automaton in the list is considered. Let us continue the construction of w by appending w_i to it, where w_i is defined as follows. If the already constructed initial part of w cannot be processed by the automaton A_i, then w_i is the empty string. Elsewhere let q_i' be the state reached by automaton A_i after reading w. Then w_i is the string that leads automaton A_i from state q_i' to state q_i. The next symbol of w is the/a symbol, for which there is no transition from q_i in A_i. Then, we increase i and consider the next automaton in the list until all of them are considered.

Finally, we add one or two symbols to the end of w, depending on length of w; for example, we add a if the length of w is odd, and aa in the other case. The constructed string has an even length, and therefore it is in L. However, our construction proves that no automaton in our list accepts the constructed string, which is a contradiction to our assumption that L is expressed as the union of languages $L(A_i)$. The theorem is proved. □

The proof also works for any language L that has the the following properties:

- the minimal complete dfa for the language has no dead state, that is, each string w in Σ^* is a prefix of a string in L;
- L contains only strings of an even/odd length.

To conclude the paper, let us consider the finite unions of deterministic union-free languages.

Definition 4. *For every positive integer n, we define dU_n as the family of languages that can be expressed as a union of n deterministic union-free languages. Furthermore, let*

$$dU_* = \bigcup_{i=1}^{\infty} dU_i.$$

The following result shows that the classes dU_n define a proper hierarchy.

Theorem 5. *For every n, there exists a language L_n such that $L_n \in dU_n \setminus dU_{n-1}$.*

Proof. Let $L_n = \{a^i b^{n-i} \mid i = 1, 2, \ldots, n\}$ be a language consisting of n strings over $\{a, b\}$ of length n. Then L_n is accepted by the union of n backbone automata. On the other hand, this language cannot be accepted by the union of any $n-1$ deterministic one-cycle-free-path automata because otherwise one of them would accept a language containing at least two shortest strings. □

Now we give some non-closure properties of the language classes dU_n.

Theorem 6. *The classes dU_n are not closed under union, concatenation, square, cyclic shift. If $n \geq 4$, then the class dU_n is not closed under star.*

Proof. For $n = 1$, that is, for the class of deterministic union-free languages, we have already proved these non-closure properties. Otherwise, consider the languages $K = \{b^i \mid 1 \leq i \leq n\}$ and $L = \{a^i b \mid 1 \leq i \leq n\}$ in dU_n. The union of these languages, as well as their concatenation, is a finite language, however, it contains more than n strings. The square of the language L, as well as its cyclic shift, is a finite language containing more than n strings.

The star of the language $aa + ab + ba + bb$ is not in dU_* by Theorem 4, and therefore the last statement of the theorem holds. □

The next theorem gives some closure properties of the class dU_*.

Theorem 7. *The class dU_* is closed under union, and it is not closed under star, intersection, and complement.*

Proof. For star, consider the language $\{\varepsilon, aa, ab, ba, bb\}$ which is in dU_5. After applying the star operation, we get the language $((a+b)(a+b))^*$. By Theorem 4, this language is not in dU_*.

For intersection, consider the languages $K = ((b^*a)^2)^*b^*$ and $L = ((a^*b)^2)^*a^*$, the first of which contains the strings with an even number of a's, while the second one consists of strings with an even number of b's. The proof of Theorem 4 works for the language $K \cap L$ as well since every string in $\{a, b\}$ is a prefix of a string in $K \cap L$, and $K \cap L$ contains only strings of an even length.

Hence the class dU_* is not closed under complement because otherwise, since it is closed under union, it would be closed also under intersection. □

We leave as an open problem whether or not the class dU_* is closed under other regular operations.

6 Conclusions

We examined in detail the classes of union-free and deterministic union-free languages. First we studied the Parikh images of union-free languages, and we proved that they coincide with so-called conditional-linear sets.

Then we defined balloon automata for deterministic union-free languages, and we used them to get regular expressions for deterministic union-free languages, as well as to prove one of the main results of our paper. This result shows that the finite unions of deterministic union-free languages describe a proper subfamily of regular languages. We also investigated the properties of classes represented as a finite union of deterministic union-free languages.

Some closure properties remain open. The characterization of the subregular language class containing the regular languages that cannot be expressed as a finite union of deterministic union-free languages is an interesting further task. Providing minimal 1cfp dfa for deterministic union-free languages, and proving/disproving our conjecture about the maximality of the balloon automata seem to be interesting challenges as well.

References

1. http://www.funstufffordogs.com/Qstore/Qstore.cgi?CMD=011&PROD=1296626935&PNAME=Balloon+Animal+Dog+Toy+-+Dog
2. http://www.facebook.com/pages/Black-Balloon-Shop-baguio-Only/183737488331381
3. 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)
4. Dassow, J.: Contextual Grammars with Subregular Choice. Fundamenta Informaticae 64, 109–118 (2005)
5. Dassow, J., Manea, F., Truthe, B.: Networks of Evolutionary Processors with Subregular Filters. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 262–273. Springer, Heidelberg (2011)
6. Bordihn, H., Holzer, M., Kutrib, M.: Henning Bordihn, Markus Holzer, Martin Kutrib: Determination of finite automata accepting subregular languages. Theoretical Computer Science 410(35), 3209–3222 (2009)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley Publishing Company, Reading MA (1979)
8. Jirásková, G., Masopust, T.: Complexity in union-free regular languages. International Journal of Foundations of Computer Science 22, 1639–1653 (2011)
9. Nagy, B.: A normal form for regular expressions. Calude, C.S., Calude, E., Dinnen, M.J. (eds.) Supplemental Papers for DLT 2004, CDMTCS, Auckland (2004)
10. Nagy, B.: Union-free regular languages and 1-cycle-free-path-automata. Publ. Math. Debrecen 68, 183–197 (2006)
11. Yu, S.: Chapter 2: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. I, pp. 41–110. Springer, Heidelberg (1997)