Publ. Math. Debrecen
68/1-2 (2006), 183–197

Union-free regular languages and 1-cycle-free-path-automata

By BENEDEK NAGY (Debrecen and Tarragona)

Abstract. In this paper, we analyze a subclass of the regular languages, namely the union-free regular languages. These languages can be given by regular expressions without the operation union. In a union-free language the words look like each other, each word contains the so-called "backbone" word of the language in scattered way. The family of special type of finite automata is investigated to recognize these languages. These automata are the 1-cycle-free-path-automata. In these class from each state there is exactly one cycle-free path going to the final state. We also have result about regular expressions with union describing union-free languages.

1. Introduction

The regular languages are the most common, well-known and well-applicable languages. They are the simplest languages in the Chomsky-hierarchy. In this paper we will consider a special subclass of the regular languages. A regular language can contain words which are completely different from each other. This can happen if the regular expression of the language contains the operation union (in regular expression we use +). For example $a + b^*$. The operation union (+) is very powerful, when we

Mathematics Subject Classification: 68Q45.
Key words and phrases: union-free languages, regular expressions, regular languages, finite automaton, directed graphs.
The research is supported by the grant S-061-2004 of the International Visegrad Fund and by the grant T049409 of the National Foundation of Scientific Research of Hungary.

allow infinite sums the expressions can describe all the type 0 languages in Chomsky-hierarchy (i.e. the whole recursive enumerable class). We will investigate the languages which can be described by regular expressions without $+$. The words of a language of this union-free family have the same "shape". In [1] the algebraic properties of union-free languages were examined, in this paper we use another approach.

The structure of the paper is as follows. In the next section we define the union-free regular languages. In Section 3 a special class of finite automata is defined. We prove that these, so-called, 1-cycle-free-path automata accept exactly the union-free subfamily of the regular languages. In Section 4 some properties of these languages are presented. In the last section we summarize our results and we raise some interesting open questions.

2. The union-free languages

In this section we recall the definitions of regular expression and regular languages [2], [5]. We define the union-free languages as well. We start this section with the basic definitions.

In the next definition we use the well-known regular operators, such as union, concatenation and Kleene-star ($+$, $\cdot$, $*$ respectively).

Definition 1. The finite expressions are regular expressions using the letters of the alphabet and symbols $+$, $\cdot$, $*$ in the following way.

The letters of the alphabet together with the empty word (signed by λ) and the empty set (empty language) are regular expressions.

If r, q are regular expressions, then $r + q$, $r \cdot q$ and r^* are regular expressions as well.

Note that the brackets can be used in regular expressions to show the order of the operations ($+$, $\cdot$, $*$). If it is obvious, then we omit the sign of the operator concatenation ($\cdot$), as usual.

We call a language regular if there is a regular expression which describes it.

We call a regular expression union-free (regular) expression if only the operators $\cdot$, $*$ are used in it. Consequently, a language which can be defined by a union-free expression is a union-free (regular) language.

Note that another important and well-examined class of the regular languages is the class of finite languages. Each of them contains only finite number of words. They can be described by the (strongly) star-free regular expressions, in which only the concatenation and the union are the allowed operations and the Kleene-star is not used. In this paper we will use the star-freeness in this strong sense (in the literature other [set-theoretical] operations such as intersection and complement are allowed to use at the [extended] star-free expressions).

Each regular expression can be written in a tree form, in which exactly the leaves are the terminal symbols of the language (λ is also allowed), and other nodes are some operations.

Note that the operation Kleene-plus is used sometimes. It is an abbreviation: $r^+ = r \cdot r^*$.

Example 1. Let $V = \{a_1, a_2, \ldots, a_n\}$ be a finite alphabet. The language $V^* = (a_1 + a_2 + \cdots + a_n)^*$ is union-free. One can write it in the following form: $L = (a_1^* a_2^* \ldots a_n^*)^*$. It is trivial that λ is in L. Moreover L contains all the letters of the alphabet V. With one more iteration of the whole bracket we can concatenate any letters to the previous string. Therefore $L = V^*$.

The language V^+ is union-free if and only if V is singular, i.e. it contains exactly one letter (and the language V is union-free only in this case also). Having at least two letters in the alphabet, V^+ contains the letters but it does not contain λ. Therefore the shortest word in V^+ is not unique. As we will show in Lemma 4 this fact implies that the language is not union-free.

The previous example shows, that the Kleene-star and Kleene-plus have different properties.

Example 2. Let $V = \{a, b, c\}$. The language containing the words bab, $baba, babc^*, \ldots$ given by the regular expression $bab(a + c^*)^*$ is union-free. The union-free expression $bab(a^*c^*)^*$ describes it as well. In Figure 1 the tree forms are presented for both regular expressions.

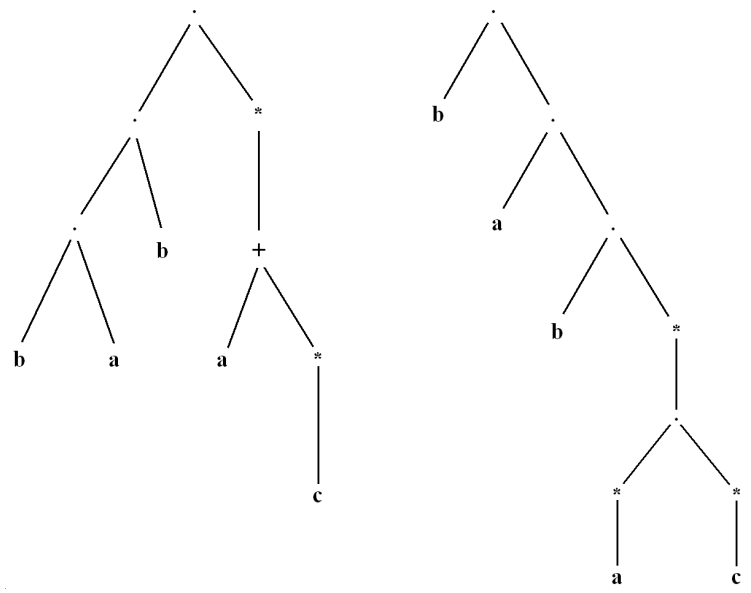


Figure 1. Examples for regular expressions in tree form

As Figure 1 shows one can express the regular expressions by trees.

In the next section we are going to define a subfamily of finite automata to accept union-free languages.

3. The class of 1-cycle-free-path-automata

The concept of finite automata is recalled in the next definition.

Definition 2. A 5-tuple $\mathbf{A} = (Q, S, V, \delta, F)$ is a non-deterministic finite automaton, with the finite (non-empty) set of states Q; $S \in Q$ is the initial state; V is the (input) alphabet and $F \subset Q$ is the set of final (or accepting) states. The function $\delta : Q \times (V \cup \lambda) \to 2^Q$ is the transition function. A path is called accepting path by the word w if it is written as $(S = Q_0)a_1Q_1a_2Q_2 \dots a_{n-1}Q_{n-1}a_nQ_n$ where $\delta(Q_i, a_{i+1}) \ni Q_{i+1}$ for every $0 \leq i < n$ and $Q_n \in F$ moreover $w = a_1a_2 \dots a_n$ (deleting the possible symbols $a_i = \lambda$ as usual). A word is accepted by the finite automata if there is an accepting path for it.

The paths written as sequences of letters and states will be called mixed-form.

Now we define a class of finite automata, and after this we prove that this class of automaton defines exactly the union-free languages.

Definition 3. Let **A** be a nondeterministic finite automaton. **A** is a *1cfpa* (**1** **c**ycle-**f**ree-**p**ath **a**utomaton) if there is a unique cycle-free accepting path from each of its states.

Figure 2 shows an example.

As a consequence of the definition above, a 1cfpa has exactly one final state. (From now on F will refer not only for the set of final states, but for its unique element as well.)

Note that we allow only transitions with exactly one symbol (from $V \cup \{\lambda\}$).

The transitions allowing symbols more than one – like $P \overset{a,b}{\to} R$ – mean at least two different paths (in this case from P to R, these paths are PaR and PbR using mixed form).

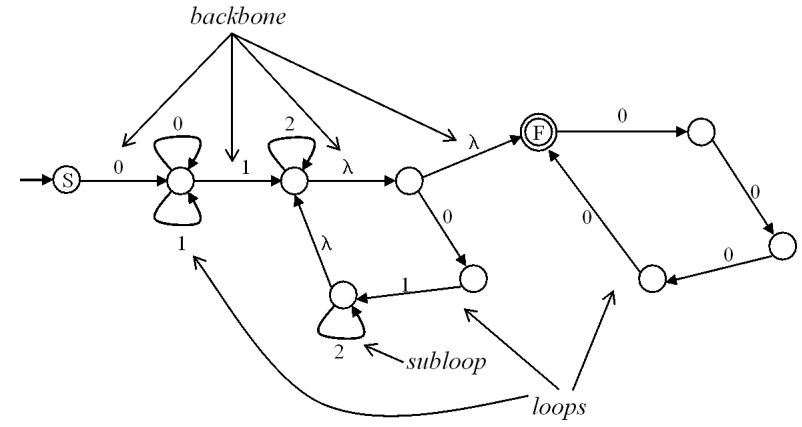


Figure 2. An example for a 1-cycle-free-path-automaton

Graph-theoretically these automata can be characterized as follows.

Proposition 1. *The graphs of the 1cfp automata are exactly those ones in which there exists a node R with the following property. For every node P in the graph there is exactly one directed path from P to R without repetition of any node.*

Proof. It is trivial from the definition. □

Note here that it can be several nodes with this property in a graph. For instance in the graph in Figure 2 there are four such nodes. They are F and the other nodes of the loop containing F.

There is a special consequence of the previous definition and fact.

Proposition 2. *Since from every state R there is exactly one transition going to the direction of F (without cycle), the word which transfers the state R to F in the cycle-free way is unique for each state.*

Proof. It is evident. □

The word which transfers the initial state to the final one has a special importance, therefore we name it.

Definition 4. We will call the word accepted by the cycle-free path from the initial state (S) to the final state (F) as a *backbone* of the automaton. The other parts of the automaton are the loops, sub-loops etc.

Theorem 1. *The family of languages which are described by union-free expressions and the family of languages recognized by 1cfpas are exactly the same.*

Proof. We will prove this in two parts. First we show that each automaton of this type recognizes a union-free language. After that we explain how we can construct a 1cfp automaton for each union-free expression.

Now, we are going to prove that each automaton of this type recognizes a union-free language. Let **A** be a 1cfpa. We will construct a union-free expression which describes the language accepted by **A**.

We will use mixed-form words, which contain terminals and names of the states of the automaton. Initially let our expression $r_0 = Sx \dots F$ be a word which contains the name of states and the terminals of the transitions of the backbone (from the initial state S to the final state F, $x \in V \cup \{\lambda\}$). At λ-transitions we allow that two neighbors in r_0 are state names.

In the next part we use the recursion as many times as the automaton branching (i.e. at least two states can follow the actual [so-called branching] state). If the automaton has no branching, then the backbone is the exactly one word of the accepted language. Assume that our expression is r_i. Then choose the last state P (which has the last occurrence) in the mixed form r_i among the states which has starting branch(es) and has not been examined yet. Now we construct r_{i+1} by modifying r_i.

Put a pair of brackets to r_i to the point immediately after each occurrence of P. Put a star after the brackets. (The following form:

$$x_1 P()^* x_2 \dots x_n P()^* x_{n+1},$$

where P has n occurrences, and x_i does not contain any P for any $i = 1, \dots, n+1$.) Then put into these brackets as many sequence $()^*$ as many starting branch the state P has. After this write into these brackets the mixed form of the words reading by the shortest circles (loops) using all the starting branches at state P without the symbol P. The obtained expression is r_{i+1}.

It is easy to show that after a step the number of occurrences of P remains n; and all the states which are in number k loops starting from P have kn new occurrences. Moreover, since the automaton is a 1cfpa there are no states appearing in the new brackets which have already examined.

Repeat the procedure above if there is at least one non-examined branching state.

We have only finitely many branching states in the automaton, therefore this procedure will be finished. Let r_n be its result. Deleting the names of the states from r_n we get a union-free expression which describes our language. It is easy to show that due to the construction there is a one-to-one mapping from the running of the automaton to reading the union-free expression for each accepted word.

And now, we construct a 1cfp automaton for each union-free expression q by an inductive method.

Iteration step:

We start from the deterministic automaton which accepts the shortest word (i.e. each sub-expression in the form r^* is the empty word.) This will be the backbone of the automaton. Specially, let

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

where the parts x_i are star-free (it is allowed that $x_i = \lambda$). (Let each $x_i = a_{i,1}a_{i,2}\dots a_{i,k_i}$, where k_i is the length of x_i.) Then the initial automaton $\mathbf{A_1}$ is the following.

$$S \overset{a_{1,1}}{\to} P_{1,1} \overset{a_{1,2}}{\to} \dots P_{1,k_1-1} \overset{a_{1,k_1}}{\to} R_1 \overset{a_{2,1}}{\to} P_{2,1} \overset{a_{2,2}}{\to} \cdots \overset{a_{2,k_2}}{\to} R_2 \dots$$

$$\dots R_n \overset{a_{n+1,1}}{\to} P_{n+1,1} \overset{a_{n+1,2}}{\to} \cdots \overset{a_{n+1,k_{n+1}}}{\to} F$$

Iteration step:

Let r^* be a subexpression which have not created yet. (We can use the order of subexpressions as they are in the tree form of the regular expression starting from the top.) Then draw the loop to the state R_j according to this subexpression (using λ for all subexpression of r^* which are in the form r_i^*) to the automaton by the similar way as we draw the backbone.

Since the union-free expression contains finitely many stars this procedure will be finished after finitely many steps. It is evident that the constructed automaton accepts the language defined by the given union-free expression. By the inductive construction, it is trivial that from each state there is exacly 1 cycle-free path goes to the final state. □

Note that the accepted languages of the sequence of automata constructed in the proof are in strictly monotonous increasing sequence.
The constructions can be seen in the following example.

Example 3. Let our automaton be given as the table and Figure 3 show.

State	A (initial)	B	C	D	E	F(inal)
Transition	$\overset{a}{\to}$ B	$\overset{b}{\to}$ C	$\overset{c}{\to}$ D	$\overset{d}{\to}$ E, $\overset{\lambda}{\to}$ B	$\overset{e}{\to}$ F, $\overset{\lambda}{\to}$ C	–

It is easy to check that this automaton is a 1cfpa. Let us construct a union-free regular expression. Initial step: the backbone in mixed form is:

$$AaBbCcDdEeF.$$

There is a loop starting from E, therefore we get:

$$AaBbCcDdE(CcDd)^*eF.$$

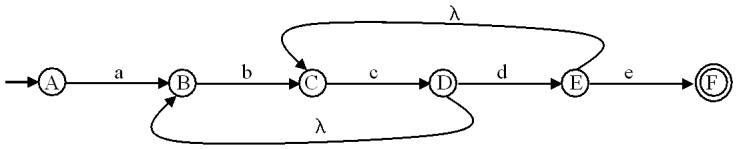


Figure 3. The automaton of Example 3

There is another loop at D, we write it to both occurrences of D:

$$AaBbCcD(BbCc)^*dE(CcD(BbCc)^*d)^*eF.$$

There are no other loops, deleting the states our expression is:

$$abc(bc)^*d(c(bc)^*d)^*e.$$

Now, we will construct a 1cfpa for this union-free expression. The backbone will be: $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D \xrightarrow{d} E \xrightarrow{e} F$. Then using the the first and the third Kleene-stars (the third one is at higher level in the tree than the second one) we get two loops: $D \xrightarrow{b} G \xrightarrow{c} D$, and $E \xrightarrow{c} H \xrightarrow{d} E$. Now using the second star we get: $H \xrightarrow{b} I \xrightarrow{c} H$.

4. Properties of union-free languages

Now we detail some properties of the languages defined above.

Lemma 1. *There are infinitely many non-comparable union-free languages.*

Proof. All the languages containing exactly one word are union-free languages. There are infinitely many of them. □

Lemma 2. *A union-free language is infinite if and only if there is no star-free regular expression to describe it.*

Proof. Without $*$ the language contains exactly one word. □

Corollary 1. *A union-free language is either infinite or contains at most one word.*

Lemma 3. *Let L be an infinite union-free language. There are infinitely many sequences of union-free languages starting with L, in which each language is a proper subset of the previous one.*

PROOF. First we construct an infinite union-free language L_1, which is strictly included in L. According to Lemma 2 there is a Kleene-star in the union-free regular expression of L and so there is at least one loop in the corresponding 1cfpa. It is easy to construct a new 1cfpa with a new backbone. Let us go through the original 1cfpa in such a way that we use exactly 1 of its (non-empty) loops. Let the read word is the backbone of the new automaton. This backbone includes the backbone of the original 1cfpa containing the letters of the loop with new states. Use at least the read loop of the original automaton in the new 1cfpa. The new automata recognizes a strictly smaller language than L, so we get a new infinite union-free language, which is a proper subset of L. And now, the procedure can be continued for the language L_1, which is also an infinite union-free one. Let $L_0 = L$. It is evident that for any infinite subsequence of $L = L_0, L_1, \ldots, L_i, \ldots$ starting with L hold the conditions of the lemma. Therefore their number is infinite. □

Corollary 2. *There is no smallest infinite union-free language. (For each infinite union-free language L there is an infinite union-free language which is a proper subset of L.)*

Now, using the backbone, we describe some other interesting properties of the union-free languages.

One of the simple similarity facts of the words of a union-free language is the following.

Proposition 3. *In a union-free language each word contains the backbone in scattered way.*

PROOF. It is obvious by using the corresponding 1cfpa. □

Let L be a union-free language. Note that $\lambda \in L$ if and only if the backbone is the empty word. This implies that every terminal is under a Kleene-star in the tree of the regular expression. Under these circumstances the language can be accepted by a 1cfpa with $S = F$.

From the previous facts we have the following useful lemma, which can be applied to decide if a language cannot be union-free.

Lemma 4. *The shortest word of a union-free language L is unique and it is the backbone.*

PROOF. Trivial. □

Now, we are in the position to claim the theorem about the closure properties of union-free languages.

Theorem 2. *The union-free language-family is closed under the following operations: concatenation, Kleene-star, substitution by union-free language. It is not closed under the following operations: union (of course), complementation, intersection, substitution by regular language.*

PROOF. The cases of concatenation and Kleene-star: trivial by using regular expressions in sense of the definition of union-free languages.
The substitution by union-free expression is also trivial.
Union: consider the following two languages: $\{a\}$, $\{b\}$.
Complementation: assume the language a^* over the alphabet $\{a, b, c\}$. Then the complement is $V^*bV^* + V^*cV^*$, which has two shortest words, namely b and c. (Or for binary alphabet, the complement of the language defined by $(aa)^*$ has the following two shortest words: a, b.)
Intersection: V^*aV^*, V^*bV^* (we cannot do without union because the letters a and b can be in two kinds of order in the words). The intersection language has two shortest words: ab, ba.
Consequence: it is not closed under substitution by regular expression. □

As a consequence of the previous theorem we have:

Corollary 3. *The union-free language-family is closed under Kleene +, and for any fixed natural number n it is closed under the n-th power.*

And now we have some notes about the non union-free regular expressions describing union-free regular languages.

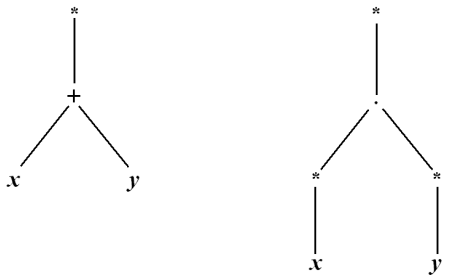


Figure 4. A possible rewriting of regular expressions to a union-free form

Let r be a given regular expression. For the sake of simplicity assume that, it is fully bracketed (i.e. its tree is a binary tree, which means that all unions and concatenations have exactly two components, while the Kleene-stars have exactly one).

Well known, and it is easy to prove that the next statement is true.

Proposition 4. *The following equivalence holds. (See Figure 4 as well.)*

$$(1) \qquad (x+y)^* \quad \textit{can be written in the form } (x^*y^*)^*$$

where x and y are arbitrary regular expressions.

Using the above equivalence the union can be removed under a Kleene-star operation. Moreover we have the following theorem about the relation between regular languages defined by regular expressions and the union-free ones.

Theorem 3. *Let r be a regular expression. If all operations union are under some Kleene-star operations in the tree form of r, then r defines a union-free regular language.*

Proof. Using the following well-known equivalences among regular expressions (see the table; x, y, z and v are arbitrary regular expressions)

one can move the operations union above the concatenations in the tree of the regular expression.

Original form	*Form with union on upper level*
$(x+y)\cdot z$	$x\cdot z+y\cdot z$
$x\cdot(z+v)$	$x\cdot z+y\cdot z$
$(x+y)\cdot(z+v)$	$(x\cdot z+x\cdot v)+(y\cdot z+y\cdot v)$

Using these equivalences the operations union move up to the level immediately below a Kleene-star. With the equivalence of the previous proposition the union can be removed from that level. □

With the previous theorem we have a nice characterization of the union-free languages. Suppose that the regular language L is given by a regular expression r. Then L is union-free if in r there is no union operation which is not under a Kleene-star (i.e. each $+$ is under a $*$ in the tree form of the regular expression). Therefore a 1cfpa can recognize each of these languages.

As a special consequence of the previous facts we have:

Corollary 4. *For each regular language L the language L^* is a union free regular one.*

5. Conclusions, further remarks

The regular languages are the most well-known languages in computer science. The description of them by regular expressions is well known. In this paper we analyzed a subclass of the regular languages. The properties of the union-free regular languages were described. These languages are defined by union-free regular expressions. The family of automata 1cfpa was investigated, in these automata there is exactly one cycle-free path to the final state from each of their states. It was shown by constructive methods that this family of automata can recognize exactly the family of union-free languages. Moreover, it was shown that regular expressions using union can define union-free languages as well, especially when each union operation is under Kleene-star operations.

We have the following open question. What is the connection between the language families accepted by 1cfpa and deterministic 1cfpa, respectively? (Note that we used only non-deterministic 1cfpa which recognize union-free languages. So the question is that the language family accepted by deterministic 1cfpas is the same, or smaller than the union-free language family?)

It will be interesting to examine how the union-free languages are related to the pattern languages. It looks, that the linear star-pattern languages have a strong relation to the union-free languages.

It is an interesting problem as well, to describe the (extended) union-free languages allowing operation intersection or complement. It is easy to show that allowing both of them, via De Morgan law one get the whole regular class.

Another further direction is to analyse the relation between bounded or polynomial-density regular languages and union-free languages ([4]).

There are some possibly interesting questions about decidability, complexity of decidability, state complexity of operations etc. Is it decidable or not, that a given language is a union-free one? If so, then also interesting problem to find an (efficient) algorithm which decides whether a given language is a union-free or not.

Note, that based on the equivalences of regular expressions their normal form and the union-complexity of languages can be defined ([3]).

Acknowledgements. The author wishes to thank the advices of the referee.

References

[1] Siniša Crvenković, Igor Dolinka and Zoltán Ésik, On equations for union-free regular languages, *Inform. and Comput.* **164**, no. 1 (2001), 152–172.

[2] J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages and Computation, *Addison-Wesley Publishing Company, Reading MA*, 1979.

[3] Benedek Nagy, A Normal Form for Regular Expressions, DLT'04, Eighth International Conference on Developments in Language Theory, *Auckland, New Zealand-finalinfo CDMTCS Report 252*, 2004, 51–60.

[4] Andrew Szilard, Sheng Yu, Kaizhong Zhang and Jeffrey Shallit, Characterizing Regular Languages with Polynomial Densities, In: Proceedings of MFCS 1992, (I. Havel and V. Koubek, eds.), 494–503, (LNCS 629).

[5] Sheng Yu, Regular languages, Vol. 3, In: Handbook of formal languages, (G. Rozenberg, A. Salomaa, eds.), *Springer-Verlag*, *Berlin*, 1997.

BENEDEK NAGY
FACULTY OF INFORMATICS
UNIVERSITY OF DEBRECEN
DEBRECEN
HUNGARY
AND
RESEARCH GROUP ON MATHEMATICAL LINGUISTICS
ROVIRA I VIRGILI UNIVERSITY
TARRAGONA
SPAIN

E-mail: nbenedek@inf.unideb.hu

(Received August 4, 2004; revised July 13, 2005)