

## CONVERSION OF REGULAR EXPRESSIONS INTO REALTIME AUTOMATA \*

VILIAM GEFFERT<sup>1</sup> AND ĽUBOMÍRA IŠTOŇOVÁ<sup>1</sup>

**Abstract.** We consider conversions of regular expressions into  $k$ -realtime finite state automata, *i.e.*, automata in which the number of consecutive uses of  $\varepsilon$ -transitions, along any computation path, is bounded by a fixed constant  $k$ . For 2-realtime automata, *i.e.*, for automata that cannot change the state, without reading an input symbol, more than two times in a row, we show that the conversion of a regular expression into such an automaton produces only  $O(n)$  states,  $O(n \log n)$   $\varepsilon$ -transitions, and  $O(n)$  alphabet-transitions. We also show how to easily transform these 2-realtime machines into 1-realtime automata, still with only  $O(n \log n)$  edges. These results contrast with the known lower bound  $\Omega(n(\log n)^2 / \log \log n)$ , holding for 0-realtime automata, *i.e.*, for automata with no  $\varepsilon$ -transitions.

**Mathematics Subject Classification.** 68Q45.

### 1. INTRODUCTION

The analysis of different tools describing formal languages not only with respect to their expressive power, but taking also into account descriptiveness complexity of these tools, is a classical topic of formal language theory. Still, not all relations among descriptiveness complexity of different kinds of formalism are known, even when restricting to devices describing regular languages only. Despite the simplicity of regular languages, some important problems concerning them are still open.

---

*Keywords and phrases.* Descriptiveness complexity, finite-state automata, regular expressions.

\* This work was supported by the Science and Technology Assistance Agency under contract APVT-20-004104, and by the Slovak Grant Agency for Science (VEGA) under contract "Combinatorial Structures and Complexity of Algorithms."

<sup>1</sup> Department of Computer Science, P. J. Šafárik University, Jesenná 5, 04001 Košice, Slovakia; geffert@upjs.sk, lubomira.istonova@upjs.sk

© EDP Sciences 2006

Regular expressions and finite automata without  $\varepsilon$ -transitions are two of the most popular formalisms for regular languages. The size of an automaton is measured by the number of its transitions, while the size of a regular expression is the number of occurrences of alphabet symbols in it.

It is known that a conversion of a nondeterministic  $\varepsilon$ -free finite automaton into an equivalent regular expression may cause an exponential blow-up [1]. For the converse direction, it was conjectured that the best conversion is an  $O(n^2)$  conversion [8]. Only quite recently [5, 6], it was proved that, for each regular expression of size  $n$ , there exists an equivalent  $\varepsilon$ -free automaton with only  $O(n(\log n)^2)$  transitions. After that, it has been shown that if we consider regular languages over a binary input alphabet, the above upper bound can be reduced to  $O(n \log n)$  transitions [2]. This result was also generalized to the case of regular languages over a fixed alphabet with at most  $s$  input symbols:  $O(sn \log n)$  edges are sufficient here.

The upper bounds, which were presented above, are complemented in [7] by the lower bound  $\Omega(n(\log n)^2 / \log \log n)$ .

Comparing  $O(n(\log n)^2)$  with  $\Omega(n(\log n)^2 / \log \log n)$ , that is, the upper and lower bounds, it seems at the first glance that no significant improvement is possible.

However, if we allow  $\varepsilon$ -transitions, the standard conversion of a regular expression into an equivalent automaton does not produce more than  $O(n)$  transitions. The crucial drawback of such automaton is that, due to cycles consisting of  $\varepsilon$ -edges only, there may exist computation paths of unbounded length.

Therefore, it is quite natural to ask what we have to add to an  $\varepsilon$ -free automaton to break the lower bound  $\Omega(n(\log n)^2 / \log \log n)$ , and still have an automaton with a computation time that is linear in the length of the input. This condition can be made even more restrictive, by requiring the automaton to execute only a constant number of steps in between reading any two consecutive symbols from the input.

We have named such machines, with the length of all  $\varepsilon$ -paths bounded by a constant, as “realtime” automata<sup>1</sup>. An automaton is *k-realtime*, if the length of any  $\varepsilon$ -path in it does not exceed the given fixed constant  $k$ .

In this paper, we shall show that already for  $k = 2$ , *i.e.*, for automata in which no  $\varepsilon$ -path is longer than 2, the conversion of a regular expression produces only  $O(n)$  states,  $O(n \log n)$   $\varepsilon$ -transitions, and  $O(n)$  alphabet-transitions. We also show an easy transformation of these 2-realtime machines into 1-realtime automata, where each  $\varepsilon$ -path degenerates into a single  $\varepsilon$ -edge. The total number of edges in the resulting automaton is still below  $O(n \log n)$ , however, the number of alphabet-transitions increases, from  $O(n)$  to  $O(n \log n)$ .

In the general case, for  $k > 2$ , the descriptonal complexity of *k-realtime* automata is still an open problem.

---

<sup>1</sup>Realtime devices play an important role in practice. Their formal definition dates back many years, down to [3]. Here we shall concentrate on the simplest realtime devices, without any auxiliary memory, which are one-way finite state automata.

## 2. PRELIMINARIES

Here we give some basic definitions and notation used throughout the paper. For a more detailed exposition on finite automata, the reader is referred to [4] or any other standard textbook.

A *nondeterministic finite automaton* is a quintuple  $M = (Q, \Sigma, \Delta, q_s, F)$ , where  $Q$  is a finite set of states,  $\Sigma$  a finite set of input symbols,  $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$  a set of transitions (edges),  $q_s \in Q$  an initial state, and  $F \subseteq Q$  a set of final (accepting) states. Here  $\varepsilon$  denotes an empty string. The *language accepted* by  $M$  is the set  $L(M)$  consisting of all strings  $w = a_1 \dots a_k \in \Sigma^*$ , for which there exists a path of transitions connecting the state  $q_s$  with some state in  $F$  and labeled by  $a_1 \dots a_k$ .

A transition  $x = (s_x, a_x, t_x) \in \Delta$  is called a  $\Sigma$ -*transition* (or an *alphabet-transition*), if  $a_x \neq \varepsilon$ , that is, if  $a_x$  is a standard input alphabet symbol. Otherwise, *i.e.*, for  $a_x = \varepsilon$ ,  $x$  is an  $\varepsilon$ -*transition*. The automaton is  $\varepsilon$ -*free*, if it is free of  $\varepsilon$ -transitions.

To make the notation of the paper more readable, the transition  $x$  will be presented in the form  $s_x \xrightarrow{a_x} t_x$ . If  $a_x = \varepsilon$ , it will be omitted, *i.e.*, the edge will be in the form  $s_x \rightarrow t_x$ . The states  $s_x, t_x \in Q$  are called the *source* and *target* states of the edge  $x$ , respectively,  $a_x \in \Sigma \cup \{\varepsilon\}$  is a *label* of  $x$ .

A path consisting of several transitions will be displayed in a more compact form  $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \xrightarrow{a_3} \dots \xrightarrow{a_{m-1}} q_m$ , with the obvious meaning. If all edges along this path are  $\varepsilon$ -transitions, we call such path an  $\varepsilon$ -*path*. Here we use the analogous notation  $q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow \dots \rightarrow q_m$ . The fact that the state  $q_m$  is reachable by some  $\varepsilon$ -path from  $q_1$  can also be expressed in the form  $q_1 \rightarrow^* q_m$ . The same notation can be used to represent that two *edges*  $x, y \in \Delta$  are connected by an  $\varepsilon$ -path:  $x \rightarrow^* y$  indicates that there exists an  $\varepsilon$ -path from the target state of  $x$  to the source state of  $y$ . The meaning of combinations like  $x \rightarrow^* q$  or  $q \rightarrow^* x$ , for  $q \in Q$  and  $x \in \Delta$ , should be obvious.

The *length* of a path is, by definition, the number of edges along this path.

$\Delta_\varepsilon$  and  $\Delta_\Sigma$  will denote the sets of all  $\varepsilon$ -transitions and  $\Sigma$ -transitions of  $M$ , respectively. Clearly, these two sets are disjoint, and  $\Delta_\varepsilon \cup \Delta_\Sigma = \Delta$ .

A *k-realtime automaton*  $M$ , where  $k \geq 0$  is an integer constant, is a nondeterministic finite state automaton in which the length of each  $\varepsilon$ -path is bounded by  $k$ . Therefore, a  $k$ -realtime automaton processes the input in "real time," with at most a constant number of executed steps in between reading any two consecutive symbols from the input. For this reasons, an automaton is called *realtime*, if it is  $k$ -realtime for some constant  $k$ .

We shall investigate the properties of the first few levels. As a special case, the automaton is 2-realtime, if each  $\varepsilon$ -path is of length at most 2. In other words, the machine cannot change its state, without reading an input symbol, more than two times in a row, and hence an input of length  $\ell$  must be processed in at most  $3\ell + 2$  steps. Similar, but even more restrictive, properties are typical for 1-realtime automata, where each  $\varepsilon$ -path degenerates into a single  $\varepsilon$ -edge. It should be clear that for  $k = 0$  we get exactly the class of  $\varepsilon$ -free automata.

A *regular expression* over an alphabet  $\Sigma$  is defined in the usual way. That is, “ $\varepsilon$ ,” “ $\emptyset$ ,” and each single letter  $a \in \Sigma$  are regular expressions. Second, if  $\alpha_1$  and  $\alpha_2$  are regular expressions, then so are  $\alpha_1^*$ ,  $\alpha_1 + \alpha_2$ , and  $\alpha_1 \cdot \alpha_2$ . The language  $L(\alpha)$ , represented by the regular expression  $\alpha$ , is defined by structural induction on  $\alpha$  in the usual way (see, e.g., [4]). We also introduce, for technical reasons, a new unary operator of option “ $\diamond$ ”, which is defined by

$$\alpha^\diamond \stackrel{\text{df.}}{=} \alpha + \varepsilon.$$

Binary operators are written in infix notation, unary operators in postfix notation and “ $\cdot$ ” is often omitted. Parentheses are used to indicate grouping.

The *size of a regular expression*  $\alpha$ , denoted by  $s(\alpha)$ , is the number of occurrences of alphabet symbols in  $\alpha$ , defined by structural induction on  $\alpha$ , i.e.:  $s(\emptyset) = s(\varepsilon) = 0$ ,  $s(a) = 1$ ,  $s(\alpha_1^*) = s(\alpha_1^\diamond) = s(\alpha_1)$ , and  $s(\alpha_1 \cdot \alpha_2) = s(\alpha_1 + \alpha_2) = s(\alpha_1) + s(\alpha_2)$ .

We shall also use a *representation of a regular expression by a binary tree*. In this tree, each binary operator is represented by an inner node with two sons corresponding to its two subexpressions, each unary operator by an inner node with one son for its only subexpression, and each occurrence of an alphabet symbol or special symbol “ $\emptyset$ ” or “ $\varepsilon$ ” represented by a leaf.

By a *subtree below a node*  $x$  we mean the tree consisting of the node  $x$  itself and all its descendants in the given tree.

In what follows, we shall need the following technical lemma. The lemma shows that each binary tree can be decomposed into two subtrees with a balanced number of leaves that were, initially, marked as “attended.” One of these subtrees is below the separating inner node  $x$ . The second one consists of all remaining nodes of the original tree.

**Lemma 2.1.** *Let  $\rho$  be a finite binary tree, with each leaf marked either as “attended” or as “ignored.” The total number of leaves marked as “attended” is  $k \geq 2$ . Then there exists a node  $x$  such that the number of attended leaves in the subtree below the node  $x$  is at most  $2/3 \cdot k$ , but more than  $1/3 \cdot k$ .*

We present this lemma without a proof, which can be found in [2], but give a simple algorithm for finding the node  $x$ . We start from the root and proceed downward in the tree. In each inner node, we go to the left or right subtree depending on which of them contains more leaves, initially marked as “attended.” In [2], Lemma 2.1, it was shown that, along this path, we shall find a node satisfying the required property. Furthermore, this will happen before we reach a leaf.

### 3. CONVERSION INTO SMALL AUTOMATA

In this section, we briefly describe the preprocessing phase, which converts the regular expression into a normal form and then into a nondeterministic automaton with at most  $2n$  states and  $n$  alphabet-transitions. This nondeterministic automaton will also contain some  $\varepsilon$ -transitions.

TABLE 1. Rewriting rules for putting a regular expression into the normal form. Here  $\alpha_1, \alpha_2$  represent arbitrary subexpressions in  $\alpha$ .

(a)	$\emptyset^* \Rightarrow \varepsilon,$ $\emptyset \cdot \alpha_1 \Rightarrow \emptyset,$ $\emptyset + \alpha_1 \Rightarrow \alpha_1,$	$\emptyset^\diamond \Rightarrow \varepsilon,$ $\alpha_1 \cdot \emptyset \Rightarrow \emptyset,$ $\alpha_1 + \emptyset \Rightarrow \alpha_1,$	(c)	$\alpha_1^{**} \Rightarrow \alpha_1^*,$ $\alpha_1^{*\diamond} \Rightarrow \alpha_1^*,$	$\alpha_1^{\diamond*} \Rightarrow \alpha_1^*,$ $\alpha_1^{\diamond\diamond} \Rightarrow \alpha_1^\diamond,$
(b)	$\varepsilon^* \Rightarrow \varepsilon,$ $\varepsilon \cdot \alpha_1 \Rightarrow \alpha_1,$ $\varepsilon + \alpha_1 \Rightarrow \alpha_1^\diamond,$	$\varepsilon^\diamond \Rightarrow \varepsilon,$ $\alpha_1 \cdot \varepsilon \Rightarrow \alpha_1,$ $\alpha_1 + \varepsilon \Rightarrow \alpha_1^\diamond,$	(d)	$(\alpha_1 + \alpha_2)^* \Rightarrow (\alpha_1^\diamond \cdot \alpha_2^\diamond)^*,$ $(\alpha_1 + \alpha_2)^\diamond \Rightarrow \alpha_1 + \alpha_2^\diamond.$	

The conversion of the original regular expression into a normal form reduces the number of binary operators to at most  $n-1$ , moreover, the converted expression  $\alpha$  satisfies the following properties: (i) either  $\alpha$  degenerates to  $\emptyset$  or  $\varepsilon$ , (ii) or, in the tree representation of  $\alpha$ , each leaf corresponds to an alphabet symbol (all symbols  $\emptyset$  and  $\varepsilon$  have been eliminated), and a son of a unary node corresponds either to a simple alphabet symbol or to a binary node for concatenation (the other unary node or a binary node for a union have also been eliminated).

The rules for this conversion are displayed in Table 1, originally presented in Lemma 3.1 in [2]. The application of these rules is repeated, while possible.

After that, the unary nodes in the tree representation of  $\alpha$  are eliminated by unifying them with their sons. Thus, we get a binary tree in which we can find only the following types of nodes. An inner node corresponds either to a union  $w_1 + w_2$ , a simple concatenation  $w_1 \cdot w_2$ , an optional concatenation  $(w_1 \cdot w_2)^\diamond$ , or an iterated concatenation  $(w_1 \cdot w_2)^*$ . A leaf represents either a simple alphabet symbol  $a \in \Sigma$ , an optional symbol  $a^\diamond$ , or an iterated symbol  $a^*$ .

A regular expression being represented by a such binary tree can be converted into a nondeterministic automaton as the following theorem shows.

**Theorem 3.1.** *Each regular expression  $\alpha$  of size  $n \geq 1$  can be replaced by an equivalent nondeterministic automaton  $M$  with at most  $2n$  states and  $n$  alphabet-transitions, such that:*

- (a) *For each subexpression  $\beta$  in  $\alpha$ , corresponding to a subtree below some node in the tree representation of  $\alpha$ , there exists a subautomaton  $M_\beta$  in  $M$ , which is a subgraph in the graph representation of  $M$ .*
- (b) *For each  $\beta'$ , a subexpression of  $\beta$  corresponding to some subtree with the top node located in the subtree for  $\beta$ ,  $M_{\beta'}$  is a subautomaton of  $M_\beta$ , i.e., a subgraph nested in the subgraph for  $M_\beta$ .*
- (c)  *$M_\beta$  has a single entry point, a state  $\text{en}_\beta \in Q$ , and a single exit point, a state  $\text{ex}_\beta \in Q$ , with  $\text{en}_\beta \neq \text{ex}_\beta$ , such that a string  $a_1 \dots a_k \in \Sigma^*$  is in  $L(\beta)$  if and only if there exists a path of edges connecting, within the subgraph for  $M_\beta$ , the state  $\text{en}_\beta$  with  $\text{ex}_\beta$  and labeled by  $a_1 \dots a_k$ .*

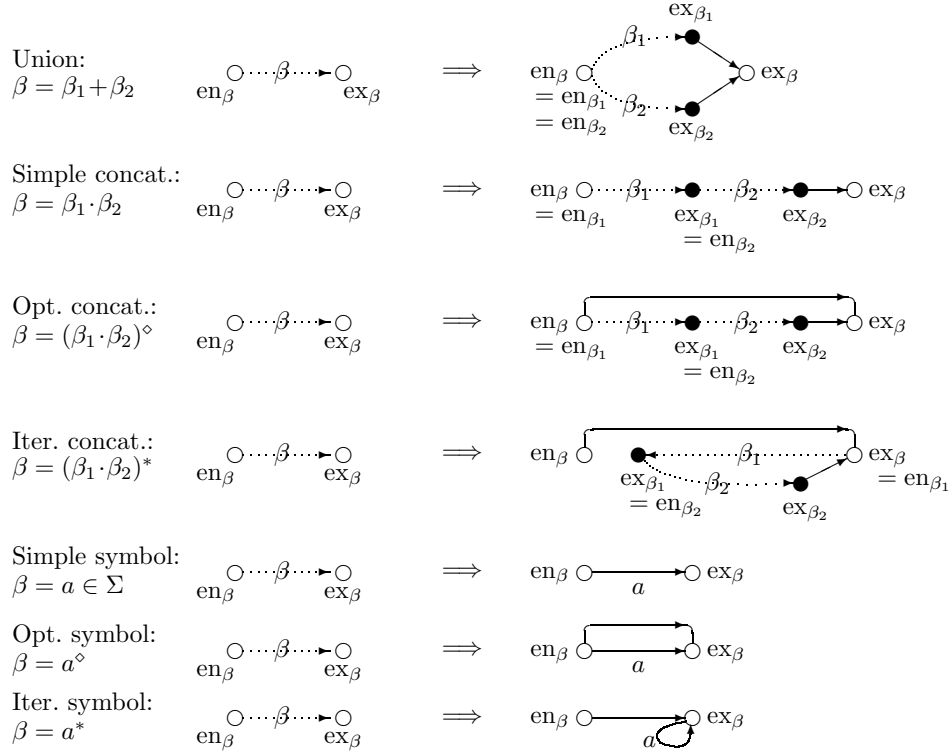


FIGURE 1. Graph rewriting rules for producing a nondeterministic automaton. Edges without labels represent  $\varepsilon$ -transitions, dotted edges are “temporary pseudo-transitions”, to be replaced by subgraphs corresponding to given subexpressions. Filled bullets represent allocated “new” states.

- (d) Any path going into the subgraph for  $M_\beta$  from the surrounding environment must pass through the state  $en_\beta$ . Further,  $M_\beta$  has no edges ending in  $en_\beta$ .

For a detailed proof of the above theorem, the reader is referred to Theorem 3.2 in [2], so we only briefly present the construction of the desired automaton. First, we have to put the expression  $\alpha$  into the normal form and eliminate unary nodes in the tree representation of  $\alpha$  by “unifying”, as described above. After that, we allocate  $q_S$  and  $q_F$ , the initial and final states for  $M$ , which are also the entry and exit points for  $\beta = \alpha$ , the root in the tree representation of  $\alpha$ . Then we connect  $q_S$  with  $q_F$  by a “temporary pseudo-transition” labeled by  $\alpha$ .

Starting from the root in the tree representation of  $\alpha$ , we then proceed downward in the tree and replace temporary pseudo-transitions in the graph, using the corresponding rules presented in Figure 1.

Before advancing further, we shall present a simple transformation of the automaton constructed in Theorem 3.1 into a realtime automaton, which will be used for some small values of  $n$ . Note that the automaton presented below is 1-realtime.

**Lemma 3.2.** *For each regular expression of size  $n \geq 1$ , there exists an equivalent nondeterministic 1-realtime automaton with at most  $2n + 1$  states,  $n$  alphabet-transitions, and  $n^2 + 1$   $\varepsilon$ -transitions.*

*Proof.* For the given regular expression of size  $n$ , we shall use the automaton from Theorem 3.1 as a starting point. This automaton uses at most  $n$  alphabet-transitions, denoted here by  $s_{a_i} \xrightarrow{a_i} t_{a_i}$ , for  $i = 1, \dots, n$ , some  $\varepsilon$ -transitions (the number of which is not important for the construction below), and  $2n$  states. Recall also that there is no edge ending in the initial state  $q_S$ . (See item (d) in Th. 3.1.)

First, replace each  $s_{a_i} \xrightarrow{a_i} t_{a_i}$  by a path  $s_{a_i} \rightarrow s'_{a_i} \xrightarrow{a_i} t'_{a_i} \rightarrow t_{a_i}$ , where  $s'_{a_i}$  and  $t'_{a_i}$  are new states. This temporarily increases the number of states and of  $\varepsilon$ -transitions. The purpose of this transformation is to guarantee that, except for the single  $\Sigma$ -transition labeled by the symbol  $a_i$ , there is no other transition (hence, neither an  $\varepsilon$ -transition) starting from the state  $s'_{a_i}$ . Similarly, no  $\varepsilon$ -transition ends in  $t'_{a_i}$ . This also ensures that  $\{s'_{a_1}, \dots, s'_{a_n}\} \cap \{t'_{a_1}, \dots, t'_{a_n}\} = \emptyset$ .

Second, remove all original  $\varepsilon$ -transitions, as well as all states except for the initial state  $q_S$  and the states  $s'_{a_1}, t'_{a_1}, \dots, s'_{a_n}, t'_{a_n}$ . Keep all  $\Sigma$ -transitions  $s'_{a_i} \xrightarrow{a_i} t'_{a_i}$ . The missing  $\varepsilon$ -paths are replaced as follows: If, for some  $i, j \in \{1, \dots, n\}$ , there existed a path of  $\varepsilon$ -transitions  $t'_{a_i} \rightarrow^* s'_{a_j}$ , include a new  $\varepsilon$ -edge  $t'_{a_i} \rightarrow s'_{a_j}$ . Similarly, if there existed a path  $q_S \rightarrow^* s'_{a_i}$ , for some  $i$ , include  $q_S \rightarrow s'_{a_i}$ . Finally, the effect of a missing path of the form  $t'_{a_i} \rightarrow^* q_F$  is imitated by making  $t'_{a_i}$  one of the final states. For the same reasons, make  $q_S$  a final state, if there existed an  $\varepsilon$ -path  $q_S \rightarrow^* q_F$ .

Since (i) all  $\varepsilon$ -transitions starting from the states  $q_S, t'_{a_1}, \dots, t'_{a_n}$  end in the states  $s'_{a_1}, \dots, s'_{a_n}$ , and (ii) there are no  $\varepsilon$ -transitions starting from  $s'_{a_1}, \dots, s'_{a_n}$ , the resulting automaton is 1-realtime. Note also that no state from among  $s'_{a_1}, \dots, s'_{a_n}$  is final.

Third, we can slightly reduce the number of  $\varepsilon$ -transitions, from  $n^2 + n$  to  $n^2 + 1$ . For each  $q \in \{s'_{a_1}, \dots, s'_{a_n}\}$ , the number of  $\varepsilon$ -edges ending in  $q$  is at most  $n + 1$ . However, if we have more than one state  $q$  with this number exactly equal to  $n + 1$ , we have more than one  $q$  with an  $\varepsilon$ -edge  $\tilde{q} \rightarrow q$  for each  $\tilde{q} \in \{q_S, t'_{a_1}, \dots, t'_{a_n}\}$ . But then we can integrate all states  $q$  having the full list of  $\varepsilon$ -edges (hence, the same list of predecessors) into a single new state, redirecting also the edges from/to these states. After this modification, there may exist at most one state  $q \in \{s'_{a_1}, \dots, s'_{a_n}\}$  with the number of  $\varepsilon$ -edges ending in  $q$  exactly equal to  $n + 1$ . Thus, there are at most  $1 \cdot (n + 1) + (n - 1) \cdot n = n^2 + 1$   $\varepsilon$ -edges. The above modification does not increase the number of states, nor  $\Sigma$ -transitions, bounded by  $2n + 1$  and  $n$ , respectively.  $\square$

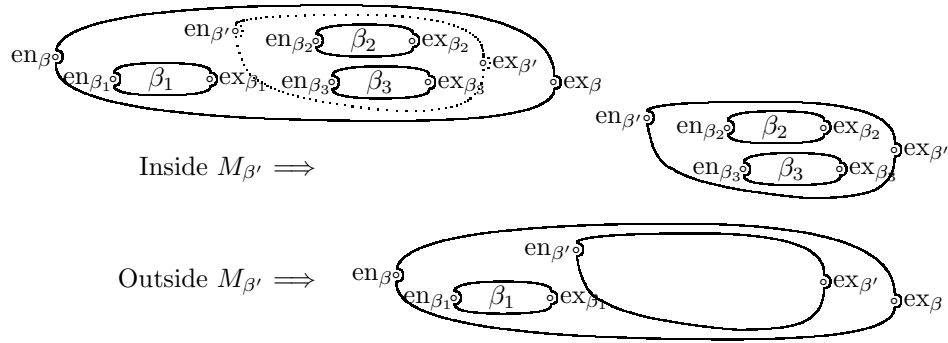


FIGURE 2. Splitting a region  $\beta - \beta_1, \beta_2, \beta_3$  into two subregions;  $\beta' - \beta_2, \beta_3$  and  $\beta - \beta', \beta_1$ .

#### 4. CONVERSION INTO 2-REALTIME AUTOMATA

Note that the automaton  $M$ , constructed in Theorem 3.1, reflects the structure of the regular expression and also of its tree representation. Recall that for each subexpression  $\beta$  in  $\alpha$ , corresponding to a subtree below some node in the graph representing  $\alpha$ , we have a subautomaton  $M_\beta$  in the automaton  $M$ .  $M_\beta$  has a single entry and a single exit point, denoted by  $en_\beta, ex_\beta$ , respectively.

Thus, by fixing some subexpression  $\beta$ , the automaton  $M$  is divided into two parts: the *inside* of  $M_\beta$ , consisting of all states and edges created by the graph rewriting rules of Figure 1 while producing  $M_\beta$  as well as all corresponding descendants in the subtree for  $\beta$ , and the *outside* of  $M_\beta$ , consisting of all remaining states and edges in  $M$ . The states  $en_\beta$  and  $ex_\beta$  form a *boundary*. By definition,  $ex_\beta$  belongs to the inside part while  $en_\beta$  to the outside part of  $M_\beta$ . The next definition generalizes this terminology.

**Definition 4.1.** Let  $\beta$  and  $\beta_1, \dots, \beta_\ell$  be some subexpressions of  $\alpha$ , such that  $\beta_1, \dots, \beta_\ell$  are subexpressions of  $\beta$ , but  $\beta_i$  is not a subexpression of  $\beta_j$ , for  $i \neq j$ . The list  $\beta_1, \dots, \beta_\ell$  may also be empty. Then a *region*  $\beta - \beta_1, \dots, \beta_\ell$  is a subgraph  $\rho$  in the graph representing  $M$ , consisting of the inside part of  $M_\beta$  after removing the inside parts of  $M_{\beta_1}, \dots, M_{\beta_\ell}$ .

We call this subgraph an *inside* part of the region. All remaining states and edges in  $M$  form an *outside* part. A *boundary* of a region consists of the boundary states for  $\beta$  and  $\beta_1, \dots, \beta_\ell$ . The state  $en_\beta$  and  $ex_{\beta_1}, \dots, ex_{\beta_\ell}$  are entry points and  $ex_\beta, en_{\beta_1}, \dots, en_{\beta_\ell}$  are exit points of the region.

Note that a state may be, at the same time, an entry and also an exit point (for example, if  $ex_{\beta_i} = en_{\beta_j}$ , for some  $i \neq j$ ). A relevant property of the regions is that they can be split into the subregions with a balanced number of  $\Sigma$ -transitions. This is formalized in the next lemma, which is a variant of Lemma 2.1. (An illustrating picture is shown in Fig. 2.)

**Lemma 4.2.** Let  $M$  be the automaton constructed in Theorem 3.1, let  $\rho = \beta - \beta_1 \dots \beta_\ell$  be a region in  $M$ , for some subexpressions  $\beta$  and  $\beta_1 \dots \beta_\ell$ , and let the total



number of  $\Sigma$ -transitions inside the region  $\varrho$  be  $k \geq 2$ . Then  $\varrho$  can be decomposed into two disjoint regions  $\varrho_1$  and  $\varrho_2$  such that the number of  $\Sigma$ -transitions inside each of them is at least  $1/3 \cdot k$ , but at most  $2/3 \cdot k$ . More precisely, the list  $\beta_1, \dots, \beta_\ell$  can be partitioned into two disjoint lists  $\beta'_1, \dots, \beta'_{\ell'}$  and  $\beta''_1, \dots, \beta''_{\ell''}$ , such that, for some subexpression  $\beta'$  in  $\beta$ ,  $\varrho_1 = \beta' - \beta'_1, \dots, \beta'_{\ell'}$  and  $\varrho_2 = \beta - \beta', \beta''_1, \dots, \beta''_{\ell''}$ .

Note that the algorithm derived from Lemma 2.1, computing the separating inner node in the tree, can be easily modified to work directly with the regions in  $M$ , and thus to find a boundary between  $\varrho_1$  and  $\varrho_2$ , *i.e.*, the boundary states of  $M_{\beta'}$ . This only requires, in the tree representation of  $\beta$ , to mark all leaves in this tree that are not contained in the subtrees for  $\beta_1, \dots, \beta_\ell$  as “attended,” but those in the subtrees for  $\beta_1, \dots, \beta_\ell$  as “ignored.” Note that the leaves marked as “attended” correspond exactly to the  $k$  alphabet-transitions in the region  $\varrho = \beta - \beta_1, \dots, \beta_\ell$ . (For more details, see Lem. 4.2 in [2].)

**Definition 4.3.** Let  $M$  be the automaton constructed in Theorem 3.1. Then, for each  $\Sigma$ -transition  $x \in \Delta_\Sigma$ , construct two sets  $\text{In}(x), \text{Out}(x) \subseteq Q$  by the use of the following procedure:

- (a) Initially, the sets  $\text{In}(x), \text{Out}(x)$  are empty. We shall also keep track of a “current region”  $\varrho$ . Initially,  $\varrho$  is equal to the entire graph for  $M$ , *i.e.*  $\varrho = \alpha - \emptyset$ .
- (b) Now let  $\varrho = \beta - \beta_1, \dots, \beta_\ell$  be the current region, containing some  $k \geq 2$  alphabet-transitions. Using the procedure described in Lemma 4.2 (based on Lem. 2.1), find  $\beta'$  splitting  $\varrho$  into two subregions  $\varrho_1 = \beta' - \beta'_1, \dots, \beta'_{\ell'}$  and  $\varrho_2 = \beta - \beta', \beta''_1, \dots, \beta''_{\ell''}$ , so that the number of  $\Sigma$ -transitions in each subregion is between  $1/3 \cdot k$  and  $2/3 \cdot k$ .
- (c) Next, use the transition  $x$  as a criterion for branching.
  - (i) If the transition  $x$  is located in the inside of  $\varrho_1$ , *i.e.*, inside  $M_{\beta'}$ , then:
    - $\varrho_1$  becomes the new current region, *i.e.*, let  $\varrho := \varrho_1$ .
    - If, in the *original* graph of  $M$ , there exists an  $\varepsilon$ -path  $\text{en}_{\beta'} \rightarrow^* x$ , insert the state  $\text{en}_{\beta'}$  into  $\text{In}(x)$ .
    - Similarly, if there exists  $x \rightarrow^* \text{ex}_{\beta'}$  in  $M$ , insert  $\text{ex}_{\beta'}$  into  $\text{Out}(x)$ .
 (Note that if such paths do not exist, no states are inserted into the respective sets  $\text{In}(x)$  and/or  $\text{Out}(x)$ .)
  - (ii) If the transition  $x$  is located in the inside of  $\varrho_2$ , *i.e.*, outside  $M_{\beta'}$ , then:
    - $\varrho_2$  becomes the current region, *i.e.*, let  $\varrho := \varrho_2$ .
    - If there exists  $x \rightarrow^* \text{en}_{\beta'}$  in  $M$ , insert  $\text{en}_{\beta'}$  to  $\text{Out}(x)$ .
    - If there exists  $\text{ex}_{\beta'} \rightarrow^* x$  in  $M$ , insert  $\text{ex}_{\beta'}$  to  $\text{In}(x)$ .
- (d) Repeat the Steps (b) and (c) until the number of  $\Sigma$ -transitions in the current region  $\varrho$  is reduced to  $k = 1$ . (At this moment, the unique  $\Sigma$ -transition remaining in  $\varrho$  is  $x$ .)

The sets  $\text{In}(x)$ ,  $\text{Out}(x)$  in Definition 4.3 are similar to the sets  $\text{In}(q)$ ,  $\text{Out}(q)$  in Definition 5.1 in [2]. The only substantial difference is that here we consider the sequence of the current regions along a trajectory zooming in the given  $\Sigma$ -transition  $x$ , rather than in a state  $q \in Q$ .

**Lemma 4.4.** *Let  $M$  be the automaton constructed in Theorem 3.1, with  $n$  alphabet-transitions, for some  $n \geq 6$ . Then, for each alphabet-transition  $x \in \Delta_\Sigma$ , neither of the sets  $\text{In}(x)$ ,  $\text{Out}(x)$  contains more than  $\log_{3/2} n - \log_{3/2}(2^7/3^4)$  states.*

*Proof.* We use the same argument as Lemma 5.2 in [2]. Recall how the sets  $\text{In}(x)$ ,  $\text{Out}(x)$  are built. The procedure starts with the region which is equal to the entire graph for  $M$ , with  $k = n$  alphabet-transitions, and with  $\text{In}(x)$ ,  $\text{Out}(x)$  being empty. This region is then, repeatedly, split into two subregions, each of them containing at most  $2/3 \cdot k$  alphabet-transitions. In each iteration, the current region becomes the one in which the edge  $x$  is located. In each iteration, at most one state is inserted in  $\text{In}(x)$  and at most one state is inserted in  $\text{Out}(x)$ . Thus, after  $i$  iterations, neither  $\text{In}(x)$  nor  $\text{Out}(x)$  can contain more than  $i$  states. The number of  $\Sigma$ -transitions in the current region is bounded by  $k \leq (2/3)^i \cdot n$ . The procedure stops when the current region contains only a single  $\Sigma$ -transition, namely, the edge  $x$ .

Thus, to reduce the number of  $\Sigma$ -transitions below 8, it is sufficient to iterate the procedure  $i$  times, where  $i$  is the *smallest integer* satisfying  $(2/3)^i \cdot n < 8$ . That is,  $i \leq \frac{1}{\log(3/2)} \cdot \log n - \frac{\log 8}{\log(3/2)} + 1$ . When the number of  $\Sigma$ -transitions has been reduced below 8, *i.e.*, to at most 7, the procedure must terminate in the next 3 iterations, since  $\lfloor 7 \cdot 2/3 \rfloor = 4$ ,  $\lfloor 4 \cdot 2/3 \rfloor = 2$ , and  $\lfloor 2 \cdot 2/3 \rfloor = 1$ . Thus,

$$\|\text{In}(x)\|, \|\text{Out}(x)\| \leq \frac{1}{\log(3/2)} \cdot \log n - \frac{\log 8}{\log(3/2)} + 4 = \frac{1}{\log(3/2)} \cdot \log n - \frac{\log(2^7/3^4)}{\log(3/2)},$$

for each  $n \geq 8$ . For  $n = 6$  or  $7$ , we have  $\|\text{In}(x)\|, \|\text{Out}(x)\| \leq 3$ , but  $\log_{3/2} n - \log_{3/2}(2^7/3^4) > 3$ . Thus, the bound holds also for  $n = 6, 7$ .  $\square$

**Lemma 4.5.** *Let  $M$  be the automaton constructed in Theorem 3.1. Let  $x, y \in \Delta$  be two  $\Sigma$ -transitions in  $M$ ,  $x \neq y$ , such that there exists an  $\varepsilon$ -path  $x \rightarrow^* y$  in  $M$ . Then  $\text{Out}(x) \cap \text{In}(y) \neq \emptyset$ .*

*Proof.* Let  $x, y$  be two transitions satisfying the assumptions of the lemma. Now consider, in parallel, sequences of regions for two instances of the procedure described in Definition 4.3, computing the sets  $\text{In}(x)$ ,  $\text{Out}(x)$ , and  $\text{In}(y)$ ,  $\text{Out}(y)$ .

Both processes start with  $\varrho$  being the entire graph for  $M$  and hence the complete path  $x \rightarrow^* y$  being located in the inside of the current region  $\varrho$ . The procedure from Lemma 4.2, based on Lemma 2.1, finds  $\beta'$  splitting  $\varrho$  into two subregions  $\varrho_1, \varrho_2$  with a balanced number of  $\Sigma$ -transitions. It should be clear that, while both transitions  $x, y$  are located in the same subregion, the two processes computing  $\text{In}(x)$ ,  $\text{Out}(x)$  and  $\text{In}(y)$ ,  $\text{Out}(y)$  follow the same trajectory of current regions.

The shared trajectory starts branching at the moment when, after splitting the current region  $\varrho$  into two subregions  $\varrho_1, \varrho_2$ , the transitions  $x, y$  fall into different subregions. This moment must come, since  $x$  does not coincide with  $y$  and the

number of  $\Sigma$ -transitions in the current region goes down to  $k = 1$ . At the moment of branching, there are two cases to consider:

(a)  $x$  falls in  $\varrho_1$ , but  $y$  in  $\varrho_2$ . Since  $x, y$  are located in different subregions, the path  $x \rightarrow^* y$  must pass through the boundary separating the regions  $\varrho_1$  and  $\varrho_2$ . More precisely, it crosses the boundary from the inside of  $\varrho_1$  out, and thus it has to pass through the state  $\text{ex}_{\beta'}$ , the exit point of  $\varrho_1$ . But then we have a path  $x \rightarrow^* \text{ex}_{\beta'}$  and hence the state  $\text{ex}_{\beta'}$  is inserted into the set  $\text{Out}(x)$ . Similarly, we have  $\text{ex}_{\beta'} \rightarrow^* y$ , a path from the entry point of  $\varrho_2$ . Hence,  $\text{ex}_{\beta'}$  is also inserted into  $\text{In}(y)$ . Thus,  $\text{ex}_{\beta'} \in \text{Out}(x) \cap \text{In}(y)$ .

(b)  $x$  falls in  $\varrho_2$ , but  $y$  in  $\varrho_1$ . Here  $x \rightarrow^* y$  has to pass through the state  $\text{en}_{\beta'}$  which is, at the same time, the exit point of  $\varrho_2$  and also the entry point of  $\varrho_1$ . Thus, we have  $x \rightarrow^* \text{en}_{\beta'} \rightarrow^* y$ , and hence  $\text{en}_{\beta'}$  is inserted in  $\text{Out}(x)$ , as well as in  $\text{In}(y)$ .

Summing up, we have that  $\text{Out}(x) \cap \text{In}(y) \neq \emptyset$ , which completes the proof.  $\square$

Now we are ready to construct a 2-realtime automaton.

**Definition 4.6.** Let  $M$  be the automaton constructed in Theorem 3.1. Then an automaton  $M'$ , a variant of  $M$ , is constructed as follows. The set of states in  $M'$  is  $Q' = Q \cup Q_s \cup Q_t \cup \{q'_s, q'_f\}$ , where  $Q_s$  and  $Q_t$  denote, respectively, the new copies of source and target states for  $\Sigma$ -transitions in  $M$ . The initial state is a new state  $q'_s$ . The set of final states is  $F' = \{q'_f\}$ , if  $\varepsilon \notin L(M)$ , but  $F' = \{q'_s, q'_f\}$ , if  $\varepsilon \in L(M)$ . Here  $q'_f$  is also a new state. Then, for each  $\Sigma$ -transition  $x = \tilde{s}_x \xrightarrow{a_x} \tilde{t}_x$ , include the following transitions in  $\Delta'$ :

- (a)  $s_x \xrightarrow{a_x} t_x$ , connecting  $s_x \in Q_s$  with  $t_x \in Q_t$ , the new copies of the original states, and labeled by the same alphabet symbol  $a_x \in \Sigma$ ;
- (b)  $q \rightarrow s_x$ , for each  $q \in \text{In}(x)$ ;
- (c)  $t_x \rightarrow q$ , for each  $q \in \text{Out}(x)$ ;
- (d)  $q'_s \rightarrow s_x$ , if there exists an  $\varepsilon$ -path  $q_s \rightarrow^* x$  in  $M$ ;
- (e)  $t_x \rightarrow q'_f$ , if there exists an  $\varepsilon$ -path  $x \rightarrow^* q_f$ ;
- (f)  $t_x \rightarrow s_x$ , if there exists an  $\varepsilon$ -path  $x \rightarrow^* x$ .

It should be pointed out that, in  $M'$ , there are no edges starting from  $q'_f$ , nor ending in  $q'_s$ . Similarly, there are no  $\varepsilon$ -edges starting from any state of  $Q_s$ , nor any  $\varepsilon$ -edges ending in  $Q_t$ . This follows from the fact that  $Q_s \cup Q_t \cup \{q'_s, q'_f\}$  is a set of new states, and hence none of these states is in  $\text{In}(x)$  or  $\text{Out}(x)$ , for no  $\Sigma$ -transition  $x$ . Figure 3 shows the different types of edges arising from the above definition.

**Lemma 4.7.** *The automaton  $M'$ , constructed in Definition 4.6, is 2-realtime.*

*Proof.* Let us recall that an automaton is 2-realtime if and only if the length of each  $\varepsilon$ -path is at most 2. Using Definition 4.6, we examine all possible  $\varepsilon$ -paths in  $M'$  (see also Fig. 3). There are the following cases to consider:

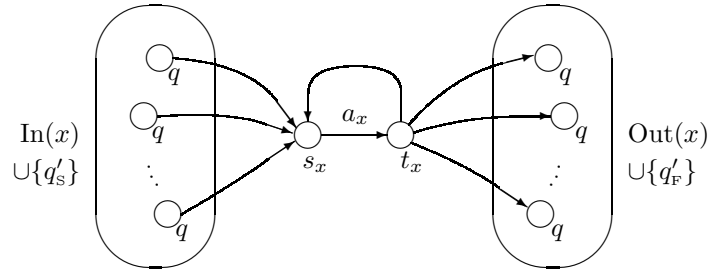


FIGURE 3. Transitions in the automaton  $M'$ , related to a  $\Sigma$ -transition  $x$  in  $M$ . The states  $s_x$  and  $t_x$  are “new” copies of the source and target states for the edge  $x$ . Edges without labels represent  $\varepsilon$ -transitions. Such edge is included only if there exists a path of  $\varepsilon$ -transitions connecting the corresponding states in  $M$ .

- (a) There are no  $\varepsilon$ -edges starting from  $q'_f$  or from  $s_x \in Q_s$ .
- (b) Starting from the initial state  $q'_s$ , we can use an  $\varepsilon$ -edge  $q'_s \rightarrow s_x$ , for some  $s_x \in Q_s$ , by (d) in Definition 4.6. Since there are no  $\varepsilon$ -edges starting from  $Q_s$ , this  $\varepsilon$ -path must end here, and hence its length is bounded by 1.
- (c) Starting from some state  $t_x \in Q_t$ , we can find the following  $\varepsilon$ -paths:
  - (i)  $t_x \rightarrow q \rightarrow s_y$ , for some  $q \in Q$  and  $s_y \in Q_s$ , by (c) and (b) in Definition 4.6. There are no  $\varepsilon$ -edges starting from  $s_y$ , and hence such  $\varepsilon$ -path is of length 2.
  - (ii)  $t_x \rightarrow q'_f$ , by (e) in Definition 4.6. Since there are no edges starting from  $q'_f$ , we have the  $\varepsilon$ -path of length 1.
  - (iii)  $t_x \rightarrow s_x$ , with  $s_x \in Q_s$ , by (f) in Definition 4.6. But there are no  $\varepsilon$ -edges starting from  $s_x$ , so this  $\varepsilon$ -path is also of length 1.
- (d) For completeness, an  $\varepsilon$ -path starting from some  $q \in Q$  can appear only as a part of a path already described in item (c-i).

There are no other  $\varepsilon$ -paths in  $M'$ , except for those mentioned above. But then the length of any  $\varepsilon$ -path in  $M'$  is at most 2.

In particular, the only case producing an  $\varepsilon$ -path of length 2 is item (c-i), representing a path connecting two  $\Sigma$ -transitions.  $\square$

We shall now prove the equivalence of the automata  $M$  and  $M'$ .

**Lemma 4.8.** *The automata  $M$  and  $M'$ , constructed in Theorem 3.1 and Definition 4.6, respectively, are equivalent.*

*Proof.* Recall that, by Theorem 3.1,  $M$  accepts an input  $w = a_1 \dots a_k$  if and only if there exists a path connecting the initial state  $q_s$  with the unique final state  $q_f$ , and labeled by  $a_1 \dots a_k$ . Similarly,  $M'$  accepts  $w$  if and only if it has a corresponding path from  $q'_s$  to an accepting state.

Suppose that  $w = a_1 \dots a_k \in L(M)$ . First, if  $w = \varepsilon$ , that is,  $k = 0$ , then, by Definition 4.6,  $q'_s$  has become a final state and hence  $\varepsilon \in L(M')$ . So assume

that  $k \geq 1$ . Then we must have an accepting path

$$q_S \rightarrow^* \tilde{s}_{a_1} \xrightarrow{a_1} \tilde{t}_{a_1} \rightarrow^* \tilde{s}_{a_2} \xrightarrow{a_2} \tilde{t}_{a_2} \rightarrow^* \dots \rightarrow^* \tilde{s}_{a_k} \xrightarrow{a_k} \tilde{t}_{a_k} \rightarrow^* q_F$$

in  $M$ , where  $\tilde{s}_{a_i}, \tilde{t}_{a_i}$  denote the source and target states of an edge labeled by the symbol  $a_i$ , for  $i = 1, \dots, k$ . Recall that, by (a) in Definition 4.6, the  $\Sigma$ -transition  $\tilde{s}_{a_i} \xrightarrow{a_i} \tilde{t}_{a_i}$  is replaced by  $s_{a_i} \xrightarrow{a_i} t_{a_i}$  in  $M'$ , where  $s_{a_i}, t_{a_i}$  are new copies of  $\tilde{s}_{a_i}, \tilde{t}_{a_i}$ .

Consider now the path  $\tilde{s}_{a_{i-1}} \xrightarrow{a_{i-1}} \tilde{t}_{a_{i-1}} \rightarrow^* \tilde{s}_{a_i} \xrightarrow{a_i} \tilde{t}_{a_i}$  in  $M$ . There are two cases:

If the edges  $\tilde{s}_{a_{i-1}} \xrightarrow{a_{i-1}} \tilde{t}_{a_{i-1}}$  and  $\tilde{s}_{a_i} \xrightarrow{a_i} \tilde{t}_{a_i}$  coincide, *i.e.*, they represent the same edge  $x \in \Delta_\Sigma$ , then, by (f) in Definition 4.6, we have an edge  $t_{a_{i-1}} \rightarrow s_{a_{i-1}}$  in  $M'$ , with  $s_{a_{i-1}} = s_{a_i}$ . Thus, in  $M'$ , there exists a path  $s_{a_{i-1}} \xrightarrow{a_{i-1}} t_{a_{i-1}} \rightarrow s_{a_i} \xrightarrow{a_i} t_{a_i}$ .

On the other hand, if the edges  $\tilde{s}_{a_{i-1}} \xrightarrow{a_{i-1}} \tilde{t}_{a_{i-1}}$  and  $\tilde{s}_{a_i} \xrightarrow{a_i} \tilde{t}_{a_i}$  do not coincide, *i.e.*, they represent two different edges  $x \neq y$  in  $\Delta_\Sigma$ , then, by Lemma 4.5, there exists a state  $q \in Q$ , such that  $q \in \text{Out}(\tilde{s}_{a_{i-1}} \xrightarrow{a_{i-1}} \tilde{t}_{a_{i-1}})$  and, at the same time,  $q \in \text{In}(\tilde{s}_{a_i} \xrightarrow{a_i} \tilde{t}_{a_i})$ . But then, by (c) and (b) in Definition 4.6, we have included the edges  $t_{a_{i-1}} \rightarrow q$  and  $q \rightarrow s_{a_i}$  in  $M'$ . Thus, even in this case, we can find a path  $s_{a_{i-1}} \xrightarrow{a_{i-1}} t_{a_{i-1}} \rightarrow^* s_{a_i} \xrightarrow{a_i} t_{a_i}$ .

Finally, since there are  $\varepsilon$ -paths  $q_S \rightarrow^* \tilde{s}_{a_1} \xrightarrow{a_1} \tilde{t}_{a_1}$  and  $\tilde{s}_{a_k} \xrightarrow{a_k} \tilde{t}_{a_k} \rightarrow^* q_F$  in  $M$ , we have also, by (d) and (e) in Definition 4.6,  $\varepsilon$ -edges  $q'_S \rightarrow s_{a_1}$  and  $t_{a_k} \rightarrow q'_F$  in  $M'$ . Summing up, we can compose the following path in  $M'$ :

$$q'_S \rightarrow s_{a_1} \xrightarrow{a_1} t_{a_1} \rightarrow^* s_{a_2} \xrightarrow{a_2} t_{a_2} \rightarrow^* \dots \rightarrow^* s_{a_k} \xrightarrow{a_k} t_{a_k} \rightarrow^* q'_F.$$

Thus,  $w = a_1 \dots a_k \in L(M')$ , and hence  $L(M) \subseteq L(M')$ .

Conversely, suppose now that  $w = a_1 \dots a_k \in L(M')$ .

If  $w = \varepsilon$ , there must exist an  $\varepsilon$ -path from  $q'_S$  to an accepting state, in  $F' = \{q'_S, q'_F\}$  or  $F' = \{q'_F\}$ . But there is no  $\varepsilon$ -path from  $q'_S$  to  $q'_F$ : starting from  $q'_S$ , we could use an  $\varepsilon$ -edge  $q'_S \rightarrow s_x$ , for some  $s_x \in Q_S$ , by (d) in Definition 4.6, but there is no way to extend this path by another  $\varepsilon$ -edge. Thus,  $w = \varepsilon$  must be accepted by a path ending in  $q'_S$ , that is,  $q'_S \in F'$ . But this holds only if  $\varepsilon \in L(M)$ , by Definition 4.6.

On the other hand,  $w = a_1 \dots a_k \neq \varepsilon$  can be accepted only by a path ending in  $q'_F$ , even if  $q'_S \in F'$ , since there are no edges ending in  $q'_S$ . This gives

$$q'_S \rightarrow^* s_{a_1} \xrightarrow{a_1} t_{a_1} \rightarrow^* s_{a_2} \xrightarrow{a_2} t_{a_2} \rightarrow^* \dots \rightarrow^* s_{a_k} \xrightarrow{a_k} t_{a_k} \rightarrow^* q'_F,$$

for some  $\Sigma$ -transitions  $s_{a_i} \xrightarrow{a_i} t_{a_i}$ , with  $s_{a_i} \in Q_S$  and  $t_{a_i} \in Q_T$ , for  $i = 1, \dots, k$ , by (a) in Definition 4.6. But  $s_{a_i} \xrightarrow{a_i} t_{a_i}$  is included in  $M'$  only if there exists a corresponding  $\Sigma$ -transition in  $M$ , *i.e.*,  $\tilde{s}_{a_i} \xrightarrow{a_i} \tilde{t}_{a_i} \in \Delta_\Sigma$ .

Consider now what types of  $\varepsilon$ -transitions can be used along the  $\varepsilon$ -path  $t_{a_{i-1}} \rightarrow^* s_{a_i}$ . (The structure of such  $\varepsilon$ -paths has already been described in the proof of Lem. 4.7.)

- An  $\varepsilon$ -edge  $t_{a_{i-1}} \rightarrow q$ , for some  $q \in Q$ , can be included in  $M'$  only by item (c) of Definition 4.6. This can happen only if  $q \in \text{Out}(x)$ , for some  $x \in \Delta_\Sigma$ , such

that  $\tilde{t}_{a_{i-1}}$  is the target state of  $x$ . But, by Definition 4.3,  $q \in \text{Out}(x)$  only if there exists an  $\varepsilon$ -path  $x \rightarrow^* q$  in  $M$ . But then we have an  $\varepsilon$ -path  $\tilde{t}_{a_{i-1}} \rightarrow^* q$ .

- Similarly, by item (b), an  $\varepsilon$ -edge  $q \rightarrow s_{a_i}$ , for  $q \in Q$ , can be included only if  $q \in \text{In}(y)$ , for some  $y \in \Delta_\Sigma$ , such that  $\tilde{s}_{a_i}$  is the source state of  $y$ . But  $q \in \text{In}(y)$  only if there exists an  $\varepsilon$ -path  $q \rightarrow^* y$ , and hence also  $q \rightarrow^* \tilde{s}_{a_i}$ , in  $M$ .
- A direct edge  $t_{a_{i-1}} \rightarrow s_{a_i}$  can be included only by (f) of Definition 4.6. This can happen only if, for some  $x \in \Delta_\Sigma$ , such that  $\tilde{s}_{a_i}, \tilde{t}_{a_{i-1}}$  are the source and target states of  $x$ , respectively, there exists an  $\varepsilon$ -path  $x \rightarrow^* x$ , and hence also  $\tilde{t}_{a_{i-1}} \rightarrow^* \tilde{s}_{a_i}$ , in  $M$ .

Summing up, we can compose a path  $\tilde{t}_{a_{i-1}} \rightarrow^* \tilde{s}_{a_i}$  in  $M$ , for each  $i = 2, \dots, k$ .

Finally, we have  $\varepsilon$ -paths  $q'_S \rightarrow^* s_{a_1}$  and  $t_{a_k} \rightarrow^* q'_F$  in  $M'$ . It is easy to see that they are both of length 1, by Definition 4.6, items (d) and (e), using also the fact that no  $\varepsilon$ -edges start in  $Q_s$ , nor end in  $Q_t$ . But the edges  $q'_S \rightarrow s_{a_1}$  and  $t_{a_k} \rightarrow q'_F$  are included in  $M'$  only if there exist some  $\varepsilon$ -paths  $q_S \rightarrow^* \tilde{s}_{a_1}$  and  $\tilde{t}_{a_k} \rightarrow^* q_F$  in  $M$ . This gives

$$q_S \rightarrow^* \tilde{s}_{a_1} \xrightarrow{a_1} \tilde{t}_{a_1} \rightarrow^* \tilde{s}_{a_2} \xrightarrow{a_2} \tilde{t}_{a_2} \rightarrow^* \dots \rightarrow^* \tilde{s}_{a_k} \xrightarrow{a_k} \tilde{t}_{a_k} \rightarrow^* q_F,$$

in  $M$ . Thus,  $w = a_1 \dots a_k \in L(M)$ , and hence  $L(M') \subseteq L(M)$ .  $\square$

Now we are ready to show the size of the automaton  $M'$ .

**Theorem 4.9.** *For each regular expression of size  $n \geq 1$ , there exists an equivalent nondeterministic 2-realtime automaton with at most  $4n + 2$  states,  $n$  alphabet-transitions, and  $n \cdot (2 \cdot \log_{3/2} n + \log_{3/2}(3^{11}/2^{17})) \leq 3.420 \cdot n \cdot \log_2 n + 0.743 \cdot n$   $\varepsilon$ -transitions.*

*Proof.* As shown before, the automaton  $M'$ , constructed in Definition 4.6, is equivalent to the original regular expression. We only have to present upper bounds for the number of states and transitions. The number of states is equal to the sum of the states in  $Q$ ,  $Q_s$ ,  $Q_t$ , plus two special states  $q'_S$  and  $q'_F$ . Recall that the automaton  $M$ , constructed in Theorem 3.1, has  $n$  alphabet-transitions and that for each alphabet-transition  $x$  we have introduced two new states,  $s_x \in Q_s$ , and  $t_x \in Q_t$ . The number of states in  $Q$  is at most  $2n$ , which bounds the number of states in  $Q'$  by  $4n + 2$ .

Second, each  $\Sigma$ -transition in  $M$  produces exactly one  $\Sigma$ -transition in  $M'$ , by (a) in Definition 4.6. Thus, the number of  $\Sigma$ -transitions in  $M'$  is at most  $n$ .

It remains to bound the number of  $\varepsilon$ -edges. By Definition 4.6, we have introduced the sets of the following types of  $\varepsilon$ -edges:

$$\begin{aligned} E_1 &\subseteq \{q \rightarrow s_x; x \in \Delta_\Sigma, q \in \text{In}(x)\}, \\ E_2 &\subseteq \{t_x \rightarrow q; x \in \Delta_\Sigma, q \in \text{Out}(x)\}, \\ E_3 &\subseteq \{q'_S \rightarrow s_x; x \in \Delta_\Sigma\}, \\ E_4 &\subseteq \{t_x \rightarrow q'_F; x \in \Delta_\Sigma\}, \\ E_5 &\subseteq \{t_x \rightarrow s_x; x \in \Delta_\Sigma\}. \end{aligned}$$

Since  $\|\Delta_\Sigma\| \leq n$ , by Theorem 3.1, and  $\|\text{In}(x)\|, \|\text{Out}(x)\| \leq \log_{3/2} n - \log_{3/2}(2^7/3^4)$ , by Lemma 4.4, the number  $\varepsilon$ -transitions in  $M'$  is bounded by

$$\|\Delta'_\varepsilon\| \leq 2n \cdot \left( \frac{1}{\log(3/2)} \cdot \log n - \frac{\log(2^7/3^4)}{\log(3/2)} \right) + 3n = n \cdot \left( \frac{2}{\log(3/2)} \cdot \log n + \frac{\log(3^{11}/2^{17})}{\log(3/2)} \right),$$

for each  $n \geq 6$ . The restriction on  $n$  is due to the fact that the above argument uses Lemma 4.4.

For  $n = 2, \dots, 13$ , we use a different construction, described in Lemma 3.2, giving us an automaton with  $2n+1 \leq 4n+2$  states,  $n$  alphabet-transitions, and  $n^2+1$   $\varepsilon$ -transitions.<sup>2</sup> It is quite easy to verify that  $n^2 + 1 < n \cdot (2 \log_{3/2} n + \log_{3/2}(3^{11}/2^{17}))$  for each  $n = 2, \dots, 13$ . It only remains to prove the statement for  $n = 1$ . However, each regular expression of size  $n = 1$  reduces to  $a, a^\diamond$ , or  $a^*$ . For each of these expressions we can easily design an automaton with at most 2 states, a single  $\Sigma$ -transition, and no  $\varepsilon$ -transitions at all. Therefore, the automaton is  $\varepsilon$ -free and hence also 1- and 2-realtime. Thus, all upper bounds are valid for each  $n \geq 1$ .  $\square$

### 5. CONVERSION INTO 1-REALTIME AUTOMATA

In the previous sections, we have presented a conversion of a regular expression into a 2-realtime automaton  $M'$ , with  $O(n \log n)$  transitions. Now it is easy to convert  $M'$  into a 1-realtime automaton  $M''$ , with the total number of edges still below  $O(n \log n)$ . The main idea of this conversion consists in replacing each path  $s_x \xrightarrow{a_x} t_x \rightarrow q$ , where  $s_x \in Q_s, t_x \in Q_t, q \in \text{Out}(x) \subseteq Q$ , by a single transition  $s_x \xrightarrow{a_x} q$ . The transition  $t_x \rightarrow q$  is removed. This idea is precisely described in the next definition:

**Definition 5.1.** Let  $M'$  be the automaton constructed in Definition 4.6. Then a 1-realtime automaton  $M''$ , a variant of  $M'$ , is constructed as follows. The set of states is  $Q'' = Q' = Q \cup Q_s \cup Q_t \cup \{q'_s, q'_f\}$  of  $M'$ , with  $q''_s = q'_s$ . Also the final states are the same, that is,  $F'' = \{q'_f\}$ , if  $\varepsilon \notin L(M')$ , but  $F'' = \{q'_s, q'_f\}$  otherwise.

- (a) If, for some  $\Sigma$ -transition  $s_x \xrightarrow{a_x} t_x \in \Delta'$ , there exists an  $\varepsilon$ -transition  $t_x \rightarrow q \in \Delta'$ , for some  $q \in \text{Out}(x)$ , include a transition  $s_x \xrightarrow{a_x} q$  in  $\Delta''$ .
- (b) All other transitions of  $M'$ , except for  $t_x \rightarrow q, t_x \in Q_t, q \in \text{Out}(x)$ , are also included in  $M''$ . More precisely, include the following transitions in  $\Delta''$ :
  - (i) each  $s_x \xrightarrow{a_x} t_x \in \Delta'$ , for  $s_x \in Q_s, t_x \in Q_t, a_x \in \Sigma$ ;
  - (ii) each  $q \rightarrow s_x \in \Delta'$ , for  $s_x \in Q_s, q \in \text{In}(x)$ ;
  - (iii) each  $q'_s \rightarrow s_x \in \Delta'$ , for  $s_x \in Q_s$ ;

---

<sup>2</sup>Note that the cases  $n \geq 6$  and  $n = 2, 3, \dots, 13$ , are not mutually exclusive. That is, for  $n = 6, \dots, 13$ , both Definition 4.6 and Lemma 3.2 give automata satisfying the required upper bounds. The upper bounds of Lemma 3.2 are not tight, e.g., for  $n = 2$ , one can show that two  $\Sigma$ - and two  $\varepsilon$ -transitions are sufficient, by constructing manually the corresponding 1-realtime automata, for all possible regular expressions of this size. We leave such tedious enumeration to the interested reader.

- (iv) each  $t_x \rightarrow q'_F \in \Delta'$ , for  $t_x \in Q_t$ ;
- (v) each  $t_x \rightarrow s_x \in \Delta'$ , for  $s_x \in Q_s$ ,  $t_x \in Q_t$ .

(Recall that such transitions had been included in  $M'$  only if there existed corresponding paths in  $M$ .)

**Lemma 5.2.** *The automaton  $M''$ , constructed in Definition 5.1, is 1-realtime.*

*Proof.* The argument is very similar to the proof of Lemma 4.7. In fact, the only case producing an  $\varepsilon$ -path of length 2, in  $M'$  of Definition 4.6, is described in item (c-i) of this lemma. More precisely, such a path was of the form  $s_x \xrightarrow{a_x} t_x \rightarrow q \rightarrow s_y$ , for some  $\Sigma$ -transitions  $x, y$  and some  $q \in \text{Out}(x) \subseteq Q$ , with no  $\varepsilon$ -edges starting from  $s_y$ . But this critical path no longer exists in  $M''$ , since the edge  $t_x \rightarrow q$  has been removed, and the path  $s_x \xrightarrow{a_x} t_x \rightarrow q$  is replaced by a new  $\Sigma$ -transition  $s_x \xrightarrow{a_x} q$  in  $M''$ . (See item (a) in Def. 5.1.) However, the transition  $q \rightarrow s_y$  has been preserved, by item (b-ii), and hence the new  $\varepsilon$ -path is of length 1.

All other cases described in the proof of Lemma 4.7 are the same, producing  $\varepsilon$ -paths of length 1. Thus, the automaton  $M''$  is 1-realtime.  $\square$

**Lemma 5.3.** *The automata  $M'$  and  $M''$ , constructed in Definitions 4.6 and 5.1, respectively, are equivalent.*

*Proof.* Using Definitions 4.6 and 5.1, it is easy to prove the following statement. Each path in  $M'$ , beginning in a state  $q \in \{q'_S\} \cup Q_s$ , consisting of at most one  $\Sigma$ -transition followed by at most two  $\varepsilon$ -transitions (hence, reading at most one symbol from the input), and ending in a state  $q'$ , can be replaced, in  $M''$ , by a path beginning and ending in the same states, and reading the same symbol (if any). Since  $M'$  is 2-realtime, each accepting path in  $M'$  can be divided into such segments, and hence it can be replaced by an accepting computation in  $M''$ . Therefore,  $L(M') \subseteq L(M'')$ .

It is also easy to see that an edge  $q \xrightarrow{a} q'$  has been included in  $M''$ , for some states  $q, q'$  and  $a \in \Sigma \cup \{\varepsilon\}$ , only if, in  $M'$ , there exists a path connecting  $q$  with  $q'$  and scanning the symbol  $a$ . Therefore,  $L(M'') \subseteq L(M')$ .  $\square$

**Theorem 5.4.** *For each regular expression of size  $n \geq 2$ , there exists an equivalent nondeterministic 1-realtime automaton with at most  $4n + 2$  states,  $n \cdot (\log_{3/2} n - \log_{3/2}(2^8/3^5)) \leq 1.710 \cdot n \cdot \log_2 n - 0.128 \cdot n$   $\Sigma$ -transitions, and  $n \cdot (\log_{3/2} n + \log_{3/2}(3^7/2^{10})) \leq 1.710 \cdot n \cdot \log_2 n + 1.872 \cdot n$   $\varepsilon$ -transitions.*

*Proof.* For  $n \geq 6$ , we can use the automaton  $M''$  constructed in Definition 5.1. It only remains to count the number of states and transitions in  $M''$ .

The number of states in  $M''$  is at most  $4n + 2$ , since  $Q'' = Q'$ .

Next, we count the number of  $\Sigma$ -transitions. These transitions are introduced by the rules (a) and (b-i) of Definition 5.1, respectively:

$$E'_1 \subseteq \{s_x \xrightarrow{a_x} q; x \in \Delta_\Sigma, q \in \text{Out}(x)\},$$

$$E'_2 \subseteq \{s_x \xrightarrow{a_x} t_x; x \in \Delta_\Sigma\}.$$



Since  $\|\Delta_\Sigma\| \leq n$ , by Theorem 3.1, and  $\|\text{In}(x)\|, \|\text{Out}(x)\| \leq \log_{3/2} n - \log_{3/2}(2^7/3^4)$ , for  $n \geq 6$ , by Lemma 4.4, the total number of  $\Sigma$ -transitions in  $M''$  is bounded by

$$\|\Delta''_\Sigma\| \leq n \cdot \left( \frac{1}{\log(3/2)} \cdot \log n - \frac{\log(2^7/3^4)}{\log(3/2)} \right) + n = n \cdot \left( \frac{1}{\log(3/2)} \cdot \log n - \frac{\log(2^8/3^5)}{\log(3/2)} \right).$$

Finally, the  $\varepsilon$ -transitions are included by items (b-ii)–(b-v). These edges must fall in one of the following sets:

$$\begin{aligned} E'_3 &\subseteq \{q \rightarrow s_x; x \in \Delta_\Sigma, q \in \text{In}(x)\}, \\ E'_4 &\subseteq \{q'_s \rightarrow s_x; x \in \Delta_\Sigma\}, \\ E'_5 &\subseteq \{t_x \rightarrow q'_f; x \in \Delta_\Sigma\}, \\ E'_6 &\subseteq \{t_x \rightarrow s_x; x \in \Delta_\Sigma\}. \end{aligned}$$

But then the total number of  $\varepsilon$ -edges in  $M''$  is bounded by

$$\|\Delta''_\varepsilon\| \leq n \cdot \left( \frac{1}{\log(3/2)} \cdot \log n - \frac{\log(2^7/3^4)}{\log(3/2)} \right) + 3n = n \cdot \left( \frac{1}{\log(3/2)} \cdot \log n + \frac{\log(3^7/2^{10})}{\log(3/2)} \right).$$

For  $n = 2, \dots, 6$ , we can use the same argument as in the proof of Theorem 4.9. That is, we use the construction described in Lemma 3.2, giving us an automaton with  $2n + 1$  states,  $n$  alphabet-transitions, and  $n^2 + 1 < n \cdot (\log_{3/2} n + \log_{3/2}(3^7/2^{10}))$   $\varepsilon$ -transitions. Thus, all derived upper bounds are valid for each  $n \geq 2$ .  $\square$

It should be pointed out that the construction presented in Definition 5.1 is not the only way how to make the automaton  $M'$  1-realtime. It is possible to replace some (or all) paths of the form  $s_x \xrightarrow{a_x} t_x \rightarrow q$ , where  $s_x \xrightarrow{a_x} t_x \in \Delta'_\Sigma$  and  $q \in \text{Out}(x) \cup \{q'_f, s_x\}$ , by transitions  $s_x \xrightarrow{a_x} q$ . Using this, we can construct several 1-realtime automata which differ in the number of states,  $\Sigma$ -transitions, and  $\varepsilon$ -transitions, with a reduced number of states or  $\varepsilon$ -transitions paid by the increased number of  $\Sigma$ -transitions. For example, we can use the following construction:

- (a) The set of states of  $M''_2$  is  $Q''_2 = Q \cup Q_s \cup \{q'_s, q'_f\}$  of  $M'$ . The initial and final states are defined in the same way as in Definition 5.1.
- (b) If, in  $M'$ , for some  $\Sigma$ -transition  $s_x \xrightarrow{a_x} t_x$ , there exists an  $\varepsilon$ -transition  $t_x \rightarrow q$ , for some  $q \in \text{Out}(x) \cup \{q'_f, s_x\}$ , include a transition  $s_x \xrightarrow{a_x} q$  in  $M''_2$ .
- (c) All  $\varepsilon$ -transitions of  $M'$ , except for  $t_x \rightarrow q$  with  $t_x \in Q_t$ , are also included in  $M''_2$ . More precisely, include  $q \rightarrow s_x$ , for each  $s_x \in Q_s$  and  $q \in \text{In}(x) \cup \{q'_s\}$ .

This gives:

**Corollary 5.5.** *For each regular expression of size  $n \geq 6$ , there exists an equivalent nondeterministic 1-realtime automaton with at most  $3n+2$  states,  $n \cdot (\log_{3/2} n + \log_{3/2}(3^6/2^9)) \leq 1.710 \cdot n \cdot \log_2 n + 0.872 \cdot n$   $\Sigma$ -transitions, and  $n \cdot (\log_{3/2} n - \log_{3/2}(2^8/3^5)) \leq 1.710 \cdot n \cdot \log_2 n - 0.128 \cdot n$   $\varepsilon$ -transitions.*

TABLE 2. Conversion of regular expressions into different types of nondeterministic automata and their descriptonal complexity.

Type of automata	Number of states	Number of $\Sigma$ -transitions	Number of $\varepsilon$ -edges
$\varepsilon$ -free automata (0-realtime)	$O(n)$	$O(n \log^2 n)$	0
1-realtime automata	$O(n)$	$O(n \log n)$	$O(n \log n)$
2-realtime automata	$O(n)$	$O(n)$	$O(n \log n)$
$k$ -realtime automata ( $k > 2$ )	$O(n)$	$O(n)$	?
$\varepsilon$ -automata	$O(n)$	$O(n)$	$O(n)$

Since the proof is analogous to the proof of Theorem 5.4, we leave it to the reader. We only remark that the construction of Lemma 3.2 cannot be used here. Thus, the result is proved only for  $n \geq 6$ . However, for larger values, we have saved  $n$  states and also  $n$  transitions.

## 6. CONCLUSION

We have presented a conversion of regular expressions into nondeterministic 2-realtime automata, using at most  $4n + 2$  states,  $n$  alphabet-transitions and  $2n \cdot \log_{3/2} n + 0.743 \cdot n$   $\varepsilon$ -transitions. We have also shown a construction of a 1-realtime automaton with at most  $4n + 2$  states,  $n \cdot \log_{3/2} n - 0.128 \cdot n$   $\Sigma$ -transitions and  $n \cdot \log_{3/2} n + 1.872n$   $\varepsilon$ -transitions. Alternatively, there exists also another construction of a 1-realtime automaton, with a smaller number of states and  $\varepsilon$ -transitions, but with a slightly increased number of  $\Sigma$ -transitions. In all above cases, the total number of all transitions is bounded by  $O(n \log n)$ .

How many transitions are needed for a general case of  $k$ -realtime automata, where  $k > 2$ , is still an open problem. Lower bounds for the realtime automata are not known either. It might be possible that, by using  $\varepsilon$ -paths longer than 2, we may potentially save some transitions. This is related to the open question of whether there exists a fixed constant  $k$  such that, for each  $n$ , the resulting  $k$ -realtime automaton will have only  $O(n)$  transitions. The best (rather trivial) construction of a realtime automaton with a linear number of transitions, known to the authors, can be obtained by using the automaton of Theorem 3.1 as a starting point, in which we eliminate all cycles composed of  $\varepsilon$ -edges by unifying all states in such a cycle into a single state. However, this results in an  $(n - 1)$ -realtime automaton, that is, in  $k = n - 1$ , which is not a fixed constant.

In Table 2, we give a survey on complexity of different types of automata resulted from conversions of regular expressions.

## REFERENCES

- [1] A. Ehrenfeucht and P. Zieger, Complexity measures for regular expressions. *J. Comput. Syst. Sci.* **12** (1976) 134–46.
- [2] V. Geffert, Translation of binary regular expressions into nondeterministic  $\varepsilon$ -free automata with  $O(n \log n)$  transitions. *J. Comput. Syst. Sci.* **67** (2003) 451–72.
- [3] S. Ginsburg, *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland (1975).
- [4] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley (1979).
- [5] J. Hromkovič, S. Seibert and T. Wilke, Translating regular expressions into small  $\varepsilon$ -free nondeterministic automata, in *Proc. Symp. Theoret. Aspects Comput. Sci. Lect. Notes Comput. Sci.* **1200** (1997) 55–66.
- [6] J. Hromkovič, S. Seibert and T. Wilke, Translating regular expressions into small  $\varepsilon$ -free nondeterministic finite automata. *J. Comput. Syst. Sci.* **62** (2001) 565–88.
- [7] Yu. Lifshits, A lower bound on the size of  $\varepsilon$ -free NFA corresponding to a regular expression. *Inform. Process. Lett.* **85** (2003) 293–99.
- [8] S. Sippu and E. Soisalon-Soininen, *Parsing Theory, Vol. I: Languages and Parsing. EATCS Monographs in Theoret. Comput. Sci.* **15** (1988).

Communicated by J. Hromkovic.

Received April 25, 2005. Accepted March 9, 2006.