ACADEMIC PRESS

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Information and Computation 186 (2003) 140–162

Information and Computation

www.elsevier.com/locate/ic

Follow automata[☆]

Lucian Ilie[*,1] and Sheng Yu [2]

Department of Computer Science, University of Western Ontario, London, ON, Canada N6A 5B7

Received 11 October 2002; revised 27 March 2003

Abstract

We give two new algorithms for constructing small nondeterministic finite automata (NFA) from regular expressions. The first constructs NFAs with ε-transitions (εNFA) which are smaller than all the other εNFAs obtained by similar constructions. Their size is at most $\frac{3}{2}|\alpha| + \frac{5}{2}$, where α is the regular expression. This is very close to optimal since we prove also the lower bound $\frac{4}{3}|\alpha| + \frac{5}{2}$. The second constructs NFAs. It uses ε-elimination in the εNFAs we just introduced and builds a quotient of the well-known position automaton w.r.t. the equivalence given by the follow relation; therefore giving the name of *follow* automaton. The new automaton uses optimally the information from the positions of a regular expression. We compare the follow automaton with the best constructions to date and show that it has important advantages over those.
© 2003 Elsevier Science (USA). All rights reserved.

MSC: 68P20, 68Q19, 68Q45, 68Q70, 68W05

Keywords: Regular expressions; Nondeterministic finite automata; Partial derivatives; Quotients; Right-invariant equivalences; ε-Elimination

1. Introduction

The importance of regular expressions for applications is well known. They describe lexical tokens for syntactic specifications and textual patterns in text manipulation systems. Regular expressions have become the basis of standard utilities such as scanner generators (lex), editors (emacs, vi), or programming

[☆] An extended abstract of this paper has been presented at *The 13th Annual Symposium on Combinatorial Pattern Matching (CPM'02)* (Fukuoka, 2002); see [15].

[*] Corresponding author. Fax: +519-661-3515.

E-mail addresses: ilie@csd.uwo.ca (L. Ilie), syu@csd.uwo.ca (S. Yu).

[1] Research partially supported by NSERC Grant R3143A01.

[2] Research partially supported by NSERC Grant OGP0041630.

languages (perl, awk), see [1,9]. While regular expressions provide an appropriate notation for regular languages, their implementation is done using finite automata. The size of the automata is crucial for the efficiency of the algorithms using them; e.g., for regular expression matching. Since the deterministic finite automata obtained from regular expressions can be exponentially larger in size, in many cases nondeterministic finite automata are used instead. Minimization of NFAs is PSPACE-complete, see [20], so other methods need to be used to obtain small NFAs. Probably the most famous such constructions are the ones of Thompson [19] which builds a nondeterministic finite automaton with ε transitions (εNFA) and the one of Glushkov and McNaughton-Yamada [10,17] which outputs a nondeterministic finite automaton without ε-transitions (NFA), called position automaton. While Thompson's automaton has linear size (in terms of the size of the regular expression), the position automaton has size at most quadratic and can be computed in quadratic time by the algorithm of Brüggemann-Klein [4]. We note that throughout the paper the size of automata will include both transitions and states.

Antimirov [2] generalized Brozozowski's derivatives and built the partial derivative automata. Champarnaud and Ziadi [6,7] improved very much Antimirov's $\mathcal{O}(n^5)$ algorithm for the construction of such NFA; their algorithm runs in quadratic time. They proved also that the partial derivative automaton is a quotient of the position automaton and so it is always smaller than or equal to the position automaton.

The best worst case comes with the construction of Hromkovič et al. [14]; their NFA, called common follow sets automaton, has size at most $\mathcal{O}(n(\log n)^2)$ and, by the algorithm of Hagenah and Muscholl [11], it can be computed in time $\mathcal{O}(n(\log n)^2)$. This construction artificially increases the number of states in order to reduce the number of transitions.

In this paper, we propose new algorithms to construct very small nondeterministic finite automata, with or without ε-transitions, from regular expressions. Our first algorithm constructs εNFAs which are smaller than all the others obtained by similar constructions; e.g., the one of Thompson [19] or the one of Sippu and Soisalon-Soininen [18] (which builds a smaller εNFA than Thompson's). Given a regular expression α, the size of our εNFA for α is at most $\frac{3}{2}|\alpha| + \frac{5}{2}$. This is very close to the optimal; we prove a lower bound of $\frac{4}{3}|\alpha| + \frac{5}{2}$.

We give then a method for constructing NFAs. It uses ε-elimination in the εNFA newly introduced. The obtained NFAs have several remarkable properties. First, although the construction of this NFA has, apparently, nothing to do with positions, it turns out, unexpectedly, that the NFA is a quotient of the position automaton with respect to the equivalence given by the follow relation; therefore giving the name of *follow automaton*. Second, we show that the follow automaton uses optimally the information from the positions of the regular expression and thus it cannot be improved this way. Third, the follow automaton is, conceptually, the simplest compared to the best similar constructions. Finally, the follow automaton seems to perform very well in practical applications. Even if the worst case is quadratic in what concerns both the size of the automaton and the running time of the algorithm, in practice it performs much better. For instance, it seems to outdo on most examples the common follow sets automaton which, as we mentioned, has the best worst case size and running time. The worst case seems to be quite irrelevant here. On the other hand, it seems very difficult to compute the average case size and running time of such constructions. Therefore, we have to rely on examples to make comparisons. For most examples, the common follow sets automaton reaches its upper bound of $\mathcal{O}(n(\log n)^2)$, while the follow automaton is linear. (Precisely, we consider parameterized examples.)

The paper is organized as follows. Section 2 contains the basic definitions we need. In Section 3 we give an algorithm to reduce regular expressions such that many redundant elements are eliminated. Section 4 gives our construction of εNFAs. It also gives the proof that it is always smaller than the well known constructions of [18,19] and the lower bound showing that it is very close to optimal. Section 5 recalls the position and partial derivative automata. The fact that the partial derivative automaton is a quotient of the position automaton is given a simpler proof in Section 6. The construction of our follow NFAs is given in Section 7. Section 8 contains the proof that our NFA is a quotient of the position automaton. The optimal use of positions in the construction of the follow NFA is shown in Section 9. Some examples are given in Section 10 to compare our constructions with the position, partial derivative, and common follow sets automata. Finally, we discuss in Section 11 some of the most important problems which should be clarified about follow automata and related constructions.

2. Regular expressions and automata

We recall here the basic definitions we need throughout the paper. For further details we refer to [13] or [20].

Let A be an alphabet and A^* the set of all words over A; ε denotes the empty word and the length of a word w is denoted $|w|$. A *language* over A is a subset of A^*. A *regular expression* over A is $\emptyset$, ε, or $a \in A$, or is obtained from these applying the following rules finitely many times: for two regular expressions α and β, the *union*, $\alpha + \beta$, the *catenation*, $\alpha \cdot \beta$, and the *star*, α^*, are regular expressions. The regular language denoted by a regular expression α is $L(\alpha)$. Also, we define $\varepsilon(\alpha)$ to be ε if $\varepsilon \in L(\alpha)$ and $\emptyset$ otherwise. The *size* of α is denoted $|\alpha|$ and represents the number of symbols in α when written in postfix (parentheses are not counted).

A *finite automaton* is a quintuple $M = (Q, A, \delta, q_0, F)$, where Q is the set of states, A is the input alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta \subseteq Q \times (A \cup \{\varepsilon\}) \times Q$ is the transition mapping; we shall denote, for $p \in Q, a \in A \cup \{\varepsilon\}$, $\delta(p, a) = \{q \in Q \mid (p, a, q) \in \delta\}$. The automaton M is called *deterministic* (DFA) if $\delta : Q \times A \to Q$ is a (partial) function, *nondeterministic* (NFA) if $\delta \subseteq Q \times A \times Q$, and *nondeterministic with ε-transitions* (εNFA) if there are no restrictions on δ. The language recognized by M is denoted $L(M)$. The *size* of a finite automaton M is $|M| = |Q| + |\delta|$; we count both states and transitions.

Let $\equiv \subseteq Q \times Q$ be an equivalence relation. For $q \in Q$, $[q]_\equiv$ denotes the equivalence class of q w.r.t. $\equiv$ and, for $S \subseteq Q$, $S/_\equiv$ denotes the quotient set $S/_\equiv = \{[q]_\equiv \mid q \in S\}$.

We say that $\equiv$ is *right invariant* w.r.t. M iff

(i) $\equiv \subseteq (Q - F)^2 \cup F^2$ (final and non-final states are not $\equiv$-equivalent) and

(ii) for any $p, q \in Q$, $a \in A$, if $p \equiv q$, then $\delta(p, a)/_\equiv = \delta(q, a)/_\equiv$.

If $\equiv$ is right invariant, the *quotient automaton* $M/_\equiv$ is constructed as $M/_\equiv = (Q/_\equiv, A, \delta_\equiv, [q_0]_\equiv, F/_\equiv)$, where $\delta_\equiv = \{([p]_\equiv, a, [q]_\equiv) \mid (p, a, q) \in \delta\}$; notice that $Q/_\equiv = (Q - F)/_\equiv \cup F/_\equiv$, so we do not merge final with non-final states. Notice that $L(M/_\equiv) = L(M)$.

3. Reduced regular expressions

We give in this section an algorithm for reducing regular expressions. The intent is to reduce the number of $\emptyset$'s and ε's, as well as the total size of the expression. Such reductions are often mentioned in literature, but we want to make things more precise here. The reduced form of regular expressions is used later in the paper where precise assumptions about the structure of the regular expressions are needed. As it will be seen, our results hold as well for expressions which are not reduced.

We first introduce several notations. For a regular expression α over A, we denote by $|\alpha|_A$ and $|\alpha|_\varepsilon$ the number of occurrences in α of letters from A and ε, respectively.

Given a regular expression α over A, assume we have the syntax tree for it; when building the tree we assume '$+$' left associative (that is, $a+b+c=(a+b)+c$), which will enable us to reduce further the number $|\alpha|_\varepsilon$. We also assume that each vertex in the tree is labelled by the corresponding symbol from $A \cup \{\varepsilon, +, \cdot, *\}$ and has associated with it the subexpression corresponding to the subtree rooted at the vertex.

The regular expressions are reduced according to the algorithm below.

Algorithm 1.
(a) $\emptyset$-reduction: compute, for each vertex β, whether or not $L(\beta)=\emptyset$ and then modify α such that, at the end, either $\alpha=\emptyset$ or α contains no $\emptyset$.
(b) ε-reduction: compute, for each vertex β, whether $\varepsilon \in L(\beta)$ and whether $L(\beta)=\{\varepsilon\}$; for each vertex β with $L(\beta)=\{\varepsilon\}$, replace the subtree rooted at β by ε and then:
- if the parent of β is labelled by '$\cdot$', then replace the parent by the other child,
- if the parent is labelled by '$*$', then replace the parent by the child,
- if the parent of β is labelled '$+$' and ε is in the language of the other child, then replace the parent by the other child.

(c) '$*$'-reduction: for any vertex labelled by '$*$', if its child is also labelled by '$*$', then replace it by its child.

We shall call α obtained after applying Algorithm 1 *reduced*. We give next two observations concerning the size of reduced regular expressions followed by some examples proving their optimality.

Proposition 2. *For any reduced regular expression α such that $\alpha \notin \{\emptyset, \varepsilon\}$, we have*
(i) $|\alpha|_A \geqslant |\alpha|_\varepsilon$,
(ii) $|\alpha| \leqslant 6|\alpha|_A - 2$.

Proof.
(i) We prove by structural induction that, for any reduced $\alpha \neq \varepsilon$, if $\varepsilon \notin L(\alpha)$, then $|\alpha|_A \geqslant |\alpha|_\varepsilon + 1$ and if $\varepsilon \in L(\alpha)$, then $|\alpha|_A \geqslant |\alpha|_\varepsilon$.
The property is true for $\alpha = a$, $a \in A$. When α has at least one operator, we assume the property true for all subexpressions of α different from ε and prove it for α.
First, assume $\alpha = \beta + \gamma$. If both β and γ are different from ε, the property is shown true for α by the inductive hypothesis on β and γ. If $\beta = \varepsilon$ (the case $\gamma = \varepsilon$ is symmetric), then, since α is reduced, we have $\varepsilon \notin L(\gamma)$. The inductive hypothesis gives $|\alpha|_A = |\gamma|_A \geqslant |\gamma|_\varepsilon + 1 = |\beta|_\varepsilon + |\gamma|_\varepsilon = |\alpha|_\varepsilon$.
If $\alpha = \beta \cdot \gamma$, then none of β and γ can be ε, and the property follows from the inductive hypothesis.
If $\alpha = \beta^*$, then $\beta \neq \varepsilon$ and, by the inductive hypothesis, $|\alpha|_A = |\beta|_A \geqslant |\beta|_\varepsilon = |\alpha|_\varepsilon$.
(ii) We prove the following assertions simultaneously by structural induction:

- if $\varepsilon \notin L(\alpha)$, then $|\alpha| \leqslant 6|\alpha|_A - 5$,
- if the root of α's tree is labelled by '$*$', then $|\alpha| \leq 6|\alpha|_A - 2$,
- if the root of α's tree is labelled by '$+$' or '$\cdot$', then $|\alpha| \leq 6|\alpha|_A - 3$.

For $\alpha = a, a \in A$, the property is true. Assume the property true for all subexpressions of α different from ε and prove it for α.
First, take $\alpha = \beta + \gamma$. If both β and γ are different from ε, then the property follows by the inductive hypothesis on β and γ. If $\beta = \varepsilon$ (similarly for $\gamma = \varepsilon$), then the inductive hypothesis gives $|\alpha| = |\gamma| + 2 \leqslant 6|\gamma|_A - 5 + 2 = 6|\alpha|_A - 3$.
Assume $\alpha = \beta \cdot \gamma$. If $\varepsilon \notin L(\alpha)$, then at least one of $L(\beta)$ and $L(\gamma)$ does not contain ε and the inductive hypothesis gives $|\alpha| = |\beta| + |\gamma| + 1 \leq 6|\beta|_A + 6|\gamma|_A - 5 - 2 + 1 < 6|\alpha|_A - 5$. If $\varepsilon \in L(\alpha)$, then ε must be in both $L(\beta)$ and $L(\gamma)$ and we have, by the inductive hypothesis, $|\alpha| = |\beta| + |\gamma| + 1 \leqslant 6|\beta|_A - 2 + 6|\gamma|_A - 2 + 1 = 6|\alpha|_A - 3$.
Finally, if $\alpha = \beta^*$, then $\beta \neq \varepsilon$ and $|\alpha| = |\beta| + 1 \leqslant 6|\beta|_A - 3 + 1 = 6|\alpha|_A - 2$. $\square$

Example 3. Consider $\alpha_1 = (a_1 + \varepsilon)^*$ and define inductively, for all $i \geqslant 1$, $\alpha_{i+1} = (\alpha_i + \beta_i)^*$, where β_i is obtained from α_i by replacing each a_j by $a_{j+|\alpha_i|_A}$. For instance,

$$\alpha_3 = (((a_1 + \varepsilon)^* + (a_2 + \varepsilon)^*)^* + ((a_3 + \varepsilon)^* + (a_4 + \varepsilon)^*)^*)^*.$$

Then, for any $n \geqslant 1$, α_n is reduced and $|\alpha_n|_A = 2^{n-1}$, $|\alpha_n|_\varepsilon = 2^{n-1}$, and $|\alpha_n| = 6 \cdot 2^{n-1} - 2$.

We shall assume that all regular expressions throughout the paper are reduced. This will not affect the complexity of our algorithms since reducing an expression takes only linear time and the size of the reduced expression is less than or equal to the size of the initial expression. Also, Proposition 2 says that all complexities can be expressed in terms of the number of letters in the regular expression, that is, $|\alpha|_A$.

4. Small εNFAs from regular expressions

We give in this section our new construction of εNFAs from regular expressions. As in the previous constructions, we construct the εNFA by induction using the structure of the regular expression.

Algorithm 4. Given a regular expression α, the algorithm constructs an εNFA for α inductively, following the structure of α, and is shown in Fig. 1. The steps should be clear from the figure but we bring some further improvements at each step:

(a) After catenation (Fig. 1(v)): denote the state common to the two automata by p; (a1) if there is a single transition outgoing from p, say $p \xrightarrow{\varepsilon} q$, then the transition is removed and p and q merged; otherwise (a2) if there is a single transition incoming to p, say $q \xrightarrow{\varepsilon} p$, then the transition is removed and p and q merged.

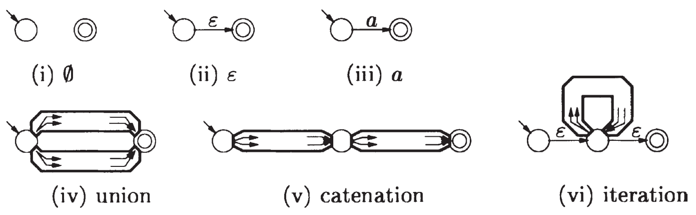

Fig. 1. The construction of $\mathbf{A}_{\mathrm{f}}^{\varepsilon}$.

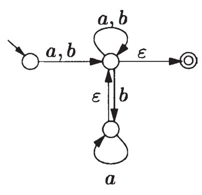

Fig. 2. $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\tau)$ for $\tau = (a+b)(a^* + ba^* + b^*)^*$.

(b) After iteration (Fig. 1(vi)), denote the middle state by p. If there is a cycle containing p such that all its transitions are labelled by ε, then all transitions in the cycle are removed and all states in the cycle are merged.
(c) After the end of all steps in Fig. 1; if there is only one transition leaving the initial state and is labelled ε, say $q_0 \overset{\varepsilon}{\to} p$, then the transition is removed and q_0 and p merged.
(d) In case of multiple transitions, that is, transitions with the same source, target, and label, only one transition is kept, the others are removed.

Example 5. An example of the construction in Algorithm 4 is given in Fig. 2. The regular expression τ used there will be our running example throughout the paper. The example was carefully contrived such that any two constructions which are, in general, different will be different on τ.

We call the automaton returned by Algorithm 4 *follow* εNFA (the reason for this name will be clear later) and denote it

$$\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha) = (Q_f^{\varepsilon}, A, \delta_f^{\varepsilon}, 0_f, q_f).$$

The next theorem proves the correctness and running time of the Algorithm 4.

Theorem 6. *For any regular expression α we have*:
(i) $L(\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)) = L(\alpha)$ *and*
(ii) $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$ *can be computed in time* $\mathcal{O}(|\alpha|)$.

Proof. (i) is clear by construction. For (ii), we just point out how the improvements at (b) can be done in linear time. Anytime a '$*$' corresponding to a subexpression β^* of α is processed, we attempt finding ε-cycles. Because all previous ε-cycles have been removed, the only possible cycles are those containing the state obtained by merging the initial and final state of the follow εNFA for β. We can do a complete

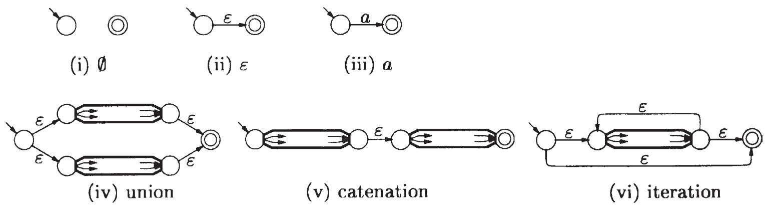

Fig. 3. The construction of Thompson [19].

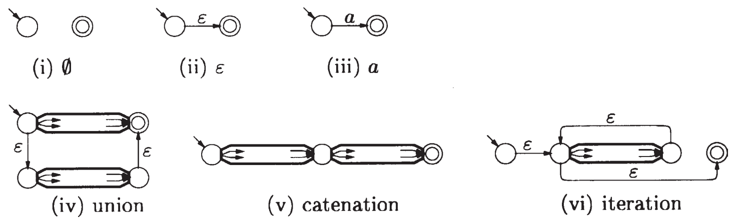

Fig. 4. The construction of Sippu and Soisalon-Soininen [18].

search using backtracking on the ε-transitions in β's automaton; when a cycle is found, it is removed and the states are merged; when we backtrack on an ε-transition, we mark that ε-transition such that it will not be tried second time. This is correct because such ε-transitions cannot be involved in other ε-cycles during the remaining of the construction. Consequently, all improvements at (b) can be done together in time $\mathcal{O}(|\alpha|)$. $\square$

The next theorem says that this εNFA is always smaller than the ones obtained by the constructions of Thompson [19] and Sippu and Soisalon-Soininen [18]. We give also an example showing that it can be much smaller. Notice that in the example we do not use the improvements (a)–(d) at all since we want to emphasize the superiority of the core of our construction. (It is easy to construct artificial expressions for which our construction, using (a)–(d), gives an arbitrarily smaller automaton.)

Theorem 7. *For any regular expression α, the size of $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$ is smaller than the size of the εNFAs obtained from α using the constructions of Thompson or Sippu and Soisalon-Soininen.*

Proof. Recall first the other two constructions. They are inductive and should be clear from Figs. 3 and 4.

All three constructions start the same way and at each inductive step (according to the structure of the regular expression), ours adds less transitions and less states. Precisely, the total number of states and transitions added by each of the three constructions for an operation '$+$', '$\cdot$', and '$*$', respectively, is (a negative number means that the size decreases):

- for our construction: -2, -1, 3;
- for Thompson's construction: 6, 1, 6;
- for Sippu and Soisalon-Soininen's construction: 2, -1, 5. $\square$

Example 8. For the regular expression $\alpha = a_1 + a_2 + \cdots + a_n$, $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$ has size $n + 2$ (2 states, n transitions), Thompson's has size $9n - 6$ ($4n - 2$ states, $5n - 4$ transitions), and Sippu and Soisalon-Soininen's has size $5n - 2$ ($2n$ states, $3n - 2$ transitions).

We discuss next an upper bound on the size of our εNFA.

A first remark concerns the following invariant of our construction. For any subexpression of α, the automaton constructed by our algorithm has one starting state of indegree zero and one final state of outdegree zero, except for the improvement at (c) which is done only at the very end of the construction.

A technical remark concerns the inductive proofs about the follow εNFA. When using induction, we shall tacitly work with $\mathbf{A}_{\mathrm{f}}^{\varepsilon}$ obtained without the improvement (c), as this is the way induction is done.

The improvements in the above steps (a)–(d) are actually very important since they can reduce significantly the size of the $\mathbf{A}_{\mathrm{f}}^{\varepsilon}$; especially the one at (b). As a consequence, for any subexpression of the form $(\beta_1^* + \cdots + \beta_m^* + \alpha_1^*\alpha_2^* \cdots \alpha_n^*)^*$, the '$*$'s for α_i's and β_i's do not increase the size of the automaton. For instance, the constructed automata for the expressions $(a^* + b^*)^*$, $(a^*b^*)^*$, and $(a + b)^*$ are identical. The same is true for any ε in an expression like $(\beta_1 + \cdots + \varepsilon + \cdots + \beta_m)^*$.

We see next a very general case when '$*$'s in the regular expression do not change the size of the automaton and we shall be able to make important assumptions on the structure of the expressions. We say that a regular expression α is $*$-*avoidable* if there is a path in α's tree from the root to a leaf such that no vertex on this path (including the root and the leaf) is labelled by '$*$'. Otherwise α is called $*$-unavoidable.

Assume β is $*$-unavoidable and construct a regular expression, $\mathsf{remove}(\beta)$, as follows. For any path from the root of β's tree to a leaf, consider the '$*$' which is closest to the root (there is at least one '$*$'). We remove this '$*$' and change all '$\cdot$'s on the path from the removed '$*$' to the root into '$+$'s. For instance, if $\beta = a^*b^* + c^*$, then $\mathsf{remove}(\beta) = a + b + c$. Now, for any regular expression α, we construct another expression $\mathsf{avoid}(\alpha)$ as follows. As long as there are subexpressions of the form β^* in α with β $*$-unavoidable, we choose a minimal such β, i.e., β has no subexpression γ^* with γ $*$-unavoidable, and replace β by $\mathsf{remove}(\beta)$. As an example, if $\alpha = \big((a^*(bc)^* + d^*)^* + (c(a + b))^*b^*\big)^*a + b$, then $\mathsf{avoid}(\alpha) = (a + bc + d + c(a + b) + b)^*a + b$.

The idea is to remove '$*$'s from α such that the language of α remains unchanged but the size decreases. As we shall see in a moment, the automaton εNFA remains the same but for an expression of smaller size. This will help us when proving an upper bound on the size of $\mathbf{A}_{\mathrm{f}}^{\varepsilon}$.

Lemma 9. *For any regular expression α, $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha^*)$ and $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\mathsf{avoid}(\alpha)^*)$ are identical.*

Proof. It is enough to show that, for any $*$-unavoidable expression β, $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\beta^*)$ and $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\mathsf{remove}(\beta)^*)$ are the same. As β is $*$-unavoidable, there are $\beta_i, i \leqslant i \leqslant n$, subexpressions of β such that β is obtained from $\beta_i^*, 1 \leqslant i \leqslant n$, by using only '$+$' and '$\cdot$'. When building the follow εNFA for β_i^*, the initial and final states of the follow εNFA for β_i are merged to a single state, say q_i. This q_i is on a path labelled ε from the initial to the final state of the follow εNFA for β. Therefore, in the automaton of β^*, all q_is will be merged. Clearly, the same happens in the follow automaton of $\mathsf{remove}(\beta)^*$. $\square$

Before proving the upper bound on the size of the follow εNFA, we need several notations and a technical lemma. For a regular expression α over A, we denote by $|\alpha|_+$, $|\alpha|_\bullet$, $|\alpha|_*$ the number of occurrences in α of '$+$', '$\cdot$', '$*$', respectively. Thus $|\alpha| = |\alpha|_A + |\alpha|_\varepsilon + |\alpha|_+ + |\alpha|_\bullet + |\alpha|_*$.

We partition the set of vertices in α's tree that are labelled by '$*$' into four classes: the first contains the root, if labelled by '$*$', and those whose parent is labelled by '$\cdot$' and whose sibling is not labelled by '$*$' – let their number be c_1; the second contains those whose parent is labelled by '$\cdot$' and whose sibling is also labelled by '$*$' – their number is $2c_2$; the third and fourth sets are defined as the previous two by replacing the label '$\cdot$' of the parent by '$+$' – their numbers are p_1 and $2p_2$, respectively.

Lemma 10. *Let α be a regular expression such that for any subexpression β^* of it, β is $*$-avoidable. Then $|\alpha|_* + p_2(\alpha) \leqslant \frac{1}{2}(|\alpha| + 1)$.*

Proof. We prove the following properties, which imply the statement; it is assumed that α below has the property in the statement, i.e., for any subexpression β^* of it, β is $*$-avoidable:
(1) if α is $*$-unavoidable and the root of α's tree is not labelled by '$*$', then $|\alpha|_* + p_2(\alpha) \leqslant \frac{1}{2}(|\alpha| + 1)$,
(2) if the root of α's tree is labelled by '$*$', then $|\alpha|_* + p_2(\alpha) \leqslant \frac{1}{2}|\alpha|$,
(3) if α is $*$-avoidable, then $|\alpha|_* + p_2(\alpha) \leqslant \frac{1}{2}(|\alpha| - 1)$.
We use structural induction. If $\alpha \in \{\emptyset, \varepsilon\} \cup \{a \mid a \in A\}$, then $|\alpha|_* + p_2(\alpha) = 0$ and (3) is satisfied. When α has at least one operator, we assume the properties true for all subexpressions of α and prove them for α.
(1) Consider first the case $\alpha = \beta + \gamma$. If at least one of β and γ has the root of the syntax tree not labelled by '$*$', then, by the inductive hypothesis, $|\alpha|_* + p_2(\alpha) = |\beta|_* + |\gamma|_* + p_2(\beta) + p_2(\gamma) \leqslant \frac{1}{2}(|\beta| + 1) + \frac{1}{2}(|\gamma| + 1) = \frac{1}{2}(|\alpha| + 1)$. If both roots of the syntax trees of β and γ are labelled by '$*$', then the inductive hypothesis gives $|\alpha|_* + p_2(\alpha) = |\beta|_* + |\gamma|_* + p_2(\beta) + p_2(\gamma) + 1 \leqslant \frac{1}{2}|\beta| + \frac{1}{2}|\gamma| + 1 = \frac{1}{2}(|\alpha| + 1)$. The case $\alpha = \beta \cdot \gamma$ is similar.
(2) Put $\alpha = \beta^*$. Then, by hypothesis, β is $*$-avoidable and we have, using the inductive hypothesis, $|\alpha|_* + p_2(\alpha) = |\beta|_* + 1 + p_2(\beta) \leqslant \frac{1}{2}(|\beta| - 1) + 1 = \frac{1}{2}|\alpha|$.
(3) In this case, either $\alpha = \beta \cdot \gamma$ or $\alpha = \beta + \gamma$ and at least one out of β and γ is $*$-avoidable. In particular, $p_2(\alpha) = p_2(\beta) + p_2(\gamma)$. We have then $|\alpha|_* + p_2(\alpha) = |\beta|_* + |\gamma|_* + p_2(\beta) + p_2(\gamma) \leqslant \frac{1}{2}|\beta| + \frac{1}{2}|\gamma| + \frac{1}{2} - \frac{1}{2} = \frac{1}{2}(|\alpha| - 1)$. □

Theorem 11. *For any reduced regular expression α, $|\mathbf{A}_\mathrm{f}^\varepsilon(\alpha)| \leqslant \frac{3}{2}|\alpha| + \frac{5}{2}$.*

Proof. Using the notations introduced above, we have

$$|\mathbf{A}_\mathrm{f}^\varepsilon(\alpha)| \leqslant 3|\alpha|_A + 3|\alpha|_\varepsilon - 2|\alpha|_+ - |\alpha|_\bullet + c_1 + 4c_2 + 3p_1 + 6p_2.$$

Using the equality $|\alpha|_A + |\alpha|_\varepsilon - 1 = |\alpha|_+ + |\alpha|_\bullet$, we can write

$$|\mathbf{A}_\mathrm{f}^\varepsilon(\alpha)| \leqslant |\alpha| + 2 - |\alpha|_+ + 2c_2 + 2p_1 + 4p_2 = |\alpha| + 2 - |\alpha|_+ + |\alpha|_* - c_1 + p_1 + 2p_2.$$

By Lemma 9, we may assume α has no subexpression β^* with $\beta$$*$-unavoidable (as otherwise we have the same automaton but for a longer expression) and may apply Lemma 10. Using also the inequality $p_1 + p_2 \leqslant |\alpha|_+$, we get $|\mathbf{A}_\mathrm{f}^\varepsilon(\alpha)| \leqslant |\alpha| + 2 + |\alpha|_* + p_2 \leqslant \frac{3}{2}|\alpha| + \frac{5}{2}$, which was to be proved. □

We move next to proving a lower bound which is very close to the upper bound in Theorem 11.

Theorem 12. *Let* $\alpha_n = (a_1^* + a_2^*)(a_3^* + a_4^*) \cdots (a_{2n-1}^* + a_{2n}^*)$. *Every* ε*NFA accepting* $L(\alpha_n)$ *has size at least* $8n - 1 = \frac{4}{3}|\alpha| + \frac{1}{3}$.

Proof. Let A_n be an εNFA accepting $L(\alpha_n)$. For any i, $1 \leqslant i \leqslant 2n$, there must be a state q_i of A_i and a cycle containing q_i and labelled by a nontrivial power of a_i. Moreover, all q_is are different and all these cycles are disjoint. Also, for any i, $1 \leqslant i \leqslant n-1$, there is a path from either of q_{2i-1} and q_{2i} to either of q_{2i+1} and q_{2i+2}. The first transitions on these paths belong to no others. So far we have shown that $|A_n| \geqslant 4n + 4(n-1)$. The rest comes from the fact that we have only one initial state. □

Using Theorem 11 and Proposition 2(ii), we obtain that $|\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)| \leqslant 9|\alpha|_A - \frac{1}{2}$. However, this result does not seem to be close to optimal and investigating upper bounds for the size of $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$ in terms of the number of letters in α remains to be further investigated.

5. Positions and partial derivatives

We recall in this section two well-known constructions of NFAs from regular expressions. The first is the *position automaton*, discovered independently by Glushkov [10] and McNaughton and Yamada [17].

Let α be a regular expression. Put $\mathsf{pos}(\alpha) = \{1, 2, \ldots, |\alpha|_A\}$ and $\mathsf{pos}_0(\alpha) = \mathsf{pos}(\alpha) \cup \{0\}$. All letters in α are made different by marking each letter with its position in α; denote the obtained expression $\overline{\alpha} \in \overline{A}^*$, where $\overline{A} = \{a_i \mid a \in A, 1 \leqslant i \leqslant |\alpha|_A\}$. For instance, if $\alpha = a(baa + b^*)$, then $\overline{\alpha} = a_1(b_2a_3a_4 + b_5^*)$. Notice that $\mathsf{pos}(\alpha) = \mathsf{pos}(\overline{\alpha})$. The same notation will also be used for removing indices, that is, for unmarked expressions α, the operator $\overline{\cdot}$ adds indices, while for marked expressions $\overline{\alpha}$ the same operator $\overline{\cdot}$ removes the indices: $\overline{\overline{\alpha}} = \alpha$. We extend the notation for arbitrary structures, like automata, in the obvious way. It will be clear from the context whether $\overline{\cdot}$ adds or removes indices.

Three mappings first, last, and follow are then defined as follows. For any regular expression α and any $i \in \mathsf{pos}(\alpha)$, we have:

$$\begin{aligned} &\mathsf{first}(\alpha) = \{i \mid a_i w \in L(\overline{\alpha})\}, \\ &\mathsf{last}(\alpha) = \{i \mid w a_i \in L(\overline{\alpha})\}, \\ &\mathsf{follow}(\alpha, i) = \{j \mid u a_i a_j v \in L(\overline{\alpha})\}. \end{aligned} \tag{1}$$

The three mappings have also an inductive definition, which we shall give later, when needed in the proofs. For future reasons, we extend $\mathsf{follow}(\alpha, 0) = \mathsf{first}(\alpha)$. Also, let $\mathsf{last}_0(\alpha)$ stand for $\mathsf{last}(\alpha)$ if $\varepsilon(\alpha) = \emptyset$ and $\mathsf{last}(\alpha) \cup \{0\}$ otherwise.
The *position automaton* for α is

$$\mathbf{A}_{\mathrm{pos}}(\alpha) = (\mathsf{pos}_0(\alpha), A, \delta_{\mathrm{pos}}, 0, \mathsf{last}_0(\alpha))$$

with $\delta_{\mathrm{pos}} = \{(i, a, j) \mid j \in \mathsf{follow}(\alpha, i), a = \overline{a_j}\}$. As shown by Glushkov [10] and McNaughton and Yamada [17], $L(\mathbf{A}_{\mathrm{pos}}(\alpha)) = L(\alpha)$. Brüggemann-Klein [4] gave an algorithm which computes the position automaton in quadratic time.

Example 13. Consider the regular expression $\tau = (a + b)(a^* + ba^* + b^*)^*$. The marked version of τ is $\overline{\tau} = (a_1 + b_2)(a_3^* + b_4a_5^* + b_6^*)^*$. The values of the mappings first, last, and follow for τ and the corresponding position automaton $\mathbf{A}_{\mathrm{pos}}(\tau)$ are given in Fig. 5.

$\mathsf{first}(\tau) = \{1, 2\}$

$\mathsf{last}(\tau) = \{1, 2, 3, 4, 5, 6\}$

i	$\mathsf{follow}(\tau, i)$
1	$\{3, 4, 6\}$
2	$\{3, 4, 6\}$
3	$\{3, 4, 6\}$
4	$\{3, 4, 5, 6\}$
5	$\{3, 4, 5, 6\}$
6	$\{3, 4, 6\}$

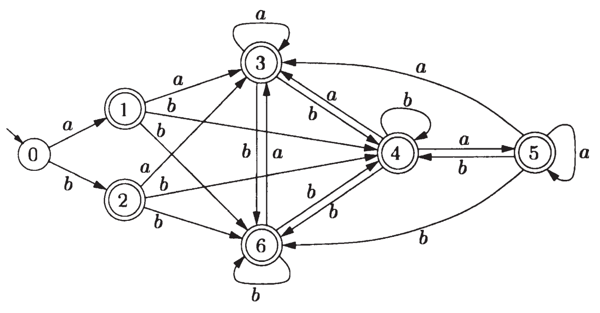

Fig. 5. $\mathbf{A}_{\mathrm{pos}}(\tau)$ for $\tau = (a + b)(a^* + ba^* + b^*)^*$.

The second construction we recall in this section is the *partial derivative automaton*, introduced by Antimirov [2]. Recall the notion of partial derivative introduced by him. For a regular expression α and a letter $a \in A$, the set $\partial_a(\alpha)$ of partial derivatives of α w.r.t. a is defined inductively as follows:

$$\begin{aligned}
&\partial_a(\varepsilon) = \partial_a(\emptyset) = \emptyset,\\
&\partial_a(b) = \begin{cases}\{\varepsilon\} & \text{if } a = b,\\ \emptyset & \text{otherwise,}\end{cases}\\
&\partial_a(\alpha + \beta) = \partial_a(\alpha) \cup \partial_a(\beta),\\
&\partial_a(\alpha\beta) = \begin{cases}\partial_a(\alpha)\beta & \text{if } \varepsilon(\alpha) = \emptyset,\\ \partial_a(\alpha)\beta \cup \partial_a(\beta) & \text{if } \varepsilon(\alpha) = \varepsilon.\end{cases}\\
&\partial_a(\alpha^*) = \partial_a(\alpha)\alpha^*.
\end{aligned} \tag{2}$$

The definition of partial derivatives is extended to words by $\partial_\varepsilon(\alpha) = \{\alpha\}$, $\partial_{wa}(\alpha) = \partial_a(\partial_w(\alpha))$, for any $w \in A^*$, $a \in A$. The set of all partial derivatives of α is denoted $\mathrm{PD}(\alpha) = \{\partial_w(\alpha) \mid w \in A^*\}$. Antimirov [2] showed that the cardinality of this set is less than or equal to $|\alpha|_A + 1$ and constructed the *partial derivative automaton*

$$\mathbf{A}_{\mathrm{pd}}(\alpha) = (\mathrm{PD}(\alpha), A, \delta_{\mathrm{pd}}, \alpha, \{q \in \mathrm{PD}(\alpha) \mid \varepsilon(q) = \varepsilon\}),$$

where $\delta_{\mathrm{pd}}(q, a) = \partial_a(q)$, for $q \in \mathrm{PD}(\alpha)$, $a \in A$; he proved $L(\mathbf{A}_{\mathrm{pd}}(\alpha)) = L(\alpha)$.

Champarnaud and Ziadi [6,7] proved that the partial derivative automaton is a quotient of the position automaton and showed how the partial derivative automaton can be computed in quadratic time, improving very much Antimirov's quintic time bound. We shall see in the next section a simplified presentation of some of their results.

Example 14. Consider the regular expression τ from Example 5. The partial derivatives of τ are computed in Fig. 6 where also its partial derivative automaton $\mathbf{A}_{\mathrm{pd}}(\tau)$ is shown.

$\partial_a(\tau) = \{\tau_1\}$ $\qquad \tau_1 = (a^* + ba^* + b^*)^*$

$\partial_b(\tau) = \{\tau_1\}$

$\partial_a(\tau_1) = \{\tau_2\}$ $\qquad \tau_2 = a^*\tau_1$

$\partial_b(\tau_1) = \{\tau_2, \tau_3\}$ $\qquad \tau_3 = b^*\tau_1$

$\partial_a(\tau_2) = \{\tau_2\}$

$\partial_b(\tau_2) = \{\tau_2, \tau_3\}$

$\partial_a(\tau_3) = \{\tau_2\}$

$\partial_b(\tau_3) = \{\tau_2, \tau_3\}$

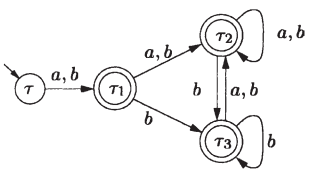

Fig. 6. $\mathbf{A}_{\mathrm{pd}}(\tau)$ for $\tau = (a + b)(a^* + ba^* + b^*)^*$.

6. $\mathbf{A}_{\mathrm{pd}}$ revisited

In this section we give a simplified proof of the fact, proved by Champarnaud and Ziadi, that the partial derivative automaton $\mathbf{A}_{\mathrm{pd}}$ is a quotient of $\mathbf{A}_{\mathrm{pos}}$. Essentially, we rely only on the work of Berry and Sethi [3]. We shall not use the notions of canonical derivative and c-continuation of [6] but show that, under certain hypotheses, they are in fact the same as the continuations of Berry and Sethi.

We assume in the following that the *rules for $\emptyset$ and ε* hold: $\alpha + \emptyset = \emptyset + \alpha = \alpha$, $\alpha \cdot \emptyset = \emptyset \cdot \alpha = \emptyset$, and $\alpha \cdot \varepsilon = \varepsilon \cdot \alpha = \alpha$. Two regular expressions α and β which reduce to the same expression using associativity, commutativity, and idempotence of $+$ are called *similar* [5]; this is denoted $\alpha \sim_{\mathrm{aci}} \beta$.

We recall also the definition of the *(total) derivative*, due to Brzozowski [5]. The derivative of α w.r.t. a letter a, $a^{-1}(\alpha)$, is defined inductively as:

$$\begin{aligned}
&a^{-1}(\varepsilon) = a^{-1}(\emptyset) = \emptyset,\\
&a^{-1}(b) = \begin{cases} \varepsilon & \text{if } a = b,\\ \emptyset & \text{otherwise,}\end{cases}\\
&a^{-1}(\alpha + \beta) = a^{-1}(\alpha) + a^{-1}(\beta),\\
&a^{-1}(\alpha\beta) = a^{-1}(\alpha)\beta + \varepsilon(\alpha)a^{-1}(\beta),\\
&a^{-1}(\alpha^*) = a^{-1}(\alpha)\alpha^*.
\end{aligned} \tag{3}$$

The definition of the total derivatives is extended to words by $\varepsilon^{-1}(\alpha) = \alpha$, $(wa)^{-1}(\alpha) = a^{-1}(w^{-1}(\alpha))$, for any $w \in A^*$, $a \in A$.

Consider the marked version of α, $\overline{\alpha} \in \overline{A}^*$ which has all letters different. Berry and Sethi proved, for a fixed $a_i \in \overline{A}$, that for all words $w \in \overline{A}^*$, $(wa_i)^{-1}(\overline{\alpha})$ is either $\emptyset$ or unique modulo $\sim_{\mathrm{aci}}$. It is clear that, for any two disjoint subexpressions β_1 and β_2 of $\overline{\alpha}$, at most one of the expressions $(wa_i)^{-1}(\beta_1)$ and $(wa_i)^{-1}(\beta_2)$ is different from $\emptyset$. Therefore, when computing total derivatives using (3), we get at each moment at most one term different from $\emptyset$. Hence, it is natural to require that we apply, whenever possible, the rules for $\emptyset$ and ε during the computation of the total derivatives. What we get is that the derivative $(wa_i)^{-1}(\overline{\alpha})$ so computed is either $\emptyset$ or unique; we got rid of the $\sim_{\mathrm{aci}}$-similarity.

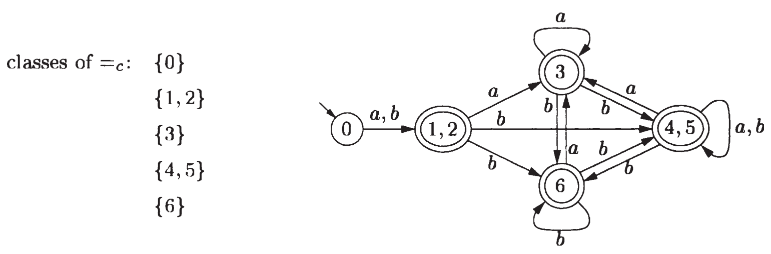

Fig. 7. $\overline{\mathbf{A}_{\rm pd}(\overline{\tau})} \simeq \overline{\mathbf{A}_{\rm pos}(\overline{\alpha})/_{=_c}}$ for $\tau = (a+b)(a^* + ba^* + b^*)^*$.

The same can be done for the computation of the partial derivatives: when using (2), we apply the rules for $\emptyset$ and ε after each step. Since they are computed in the same way, we have $\partial_{wa_i}(\overline{\alpha}) = (wa_i)^{-1}(\overline{\alpha})$.

Recall next the notion of a *continuation*, also from Berry and Sethi. For a letter $a_i \in \overline{A}$, the continuation of a_i in α, denoted $c_i(\overline{\alpha})$, is any expression $(wa_i)^{-1}(\overline{\alpha}) \not\equiv \emptyset$. From the above, this notion is well defined. Notice again that we are not talking about $\sim_{\rm aci}$-equivalent expressions because, by our assumption, there is only one. Denote also $c_0(\overline{\alpha}) = \overline{\alpha}$. Berry and Sethi's *continuation automaton* is then $\mathbf{A}_{\rm con}(\overline{\alpha}) = (Q, A, \delta, q_0, F)$, where $Q = \{c_i(\overline{\alpha}) \mid i \in \mathsf{pos}_0(\alpha)\}$, $q_0 = \overline{\alpha}$, $F = \{q \mid \varepsilon(q) = \varepsilon\}$, and $\delta = \{(c_i(\overline{\alpha}), a_j, c_j(\overline{\alpha})) \mid a_j \in \mathsf{first}(c_i(\overline{\alpha}))\}$. As Berry and Sethi proved.

Proposition 15. $\mathbf{A}_{\rm con}(\overline{\alpha}) \simeq \mathbf{A}_{\rm pos}(\overline{\alpha})$ *and* $\mathbf{A}_{\rm con}(\alpha) \simeq \mathbf{A}_{\rm pos}(\alpha)$.

The difference between the continuation or position automaton, for α or $\overline{\alpha}$ is that the labels on transitions are unmarked or marked, respectively. Obviously, if two automata with marked letters are isomorphic, so are the unmarked versions.

It is worth mentioning that the language accepted by the two automata for $\overline{\alpha}$ is $L(\overline{\alpha})$. Also, $\overline{L(\overline{\alpha})} = L(\alpha)$. Notice that for the continuation and position automata, it makes no difference whether we work first with $\overline{\alpha}$ and unmark the obtained automaton or we work with α. However, as we shall see in a moment, the same is not valid for the partial derivative automaton.

Now, from the definition of $\mathbf{A}_{\rm pd}(\overline{\alpha})$, the difference w.r.t. $\mathbf{A}_{\rm con}(\overline{\alpha})$ is that whenever two continuations of $\overline{\alpha}$ (including $\overline{\alpha}$) are the same, they represent different states in $\mathbf{A}_{\rm con}(\overline{\alpha})$ but the same in $\mathbf{A}_{\rm pd}(\overline{\alpha})$. Define then the equivalence $=_c \subseteq (\mathsf{pos}(\alpha))^2$ by $i =_c j$ iff $c_i(\overline{\alpha}) = c_j(\overline{\alpha})$; $=_c$ is right-invariant w.r.t. the position automaton. What we have so far is that

Proposition 16.
(i) $\mathbf{A}_{\rm pd}(\overline{\alpha}) \simeq \mathbf{A}_{\rm con}(\overline{\alpha})/_{=_c} \simeq \mathbf{A}_{\rm pos}(\overline{\alpha})/_{=_c}$
(ii) $\overline{\mathbf{A}_{\rm pd}(\overline{\alpha})} \simeq \overline{\mathbf{A}_{\rm con}(\overline{\alpha})/_{=_c}} \simeq \overline{\mathbf{A}_{\rm pos}(\overline{\alpha})/_{=_c}}$.

Example 17. For the regular expression τ from Example 5, we construct in Fig. 7 the automaton $\overline{\mathbf{A}_{\rm pd}(\overline{\tau})}$; the classes of the equivalence $=_c$ are also shown.

We have worked so far in this section only with regular expressions which have all letters different. We shall now remove the marking and see what happens. Define another equivalence, $\equiv_c \subseteq (\mathsf{pos}(\alpha))^2$ by $i \equiv_c j$ iff $\overline{c_i(\overline{\alpha})} = \overline{c_j(\overline{\alpha})}$; $\equiv_c$ is also right-invariant w.r.t. the position automaton and $=_c \subseteq \equiv_c$.

classes of $\equiv_c$: $\{0\}$
$\{1,2\}$
$\{3,4,5\}$
$\{6\}$

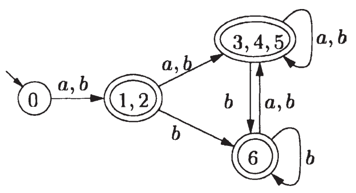

Fig. 8. $\mathbf{A}_{\mathrm{pd}}(\tau) \simeq \mathbf{A}_{\mathrm{pos}}(\tau)/_{\equiv_c}$ for $\tau = (a+b)(a^* + ba^* + b^*)^*$.

For any letter a and regular expression β, it is clear that $\partial_a(\alpha) = \{\overline{\partial_{a_i}(\overline{\alpha})} \mid \overline{a_i} = a\}$. Therefore, the partial derivative automaton is obtained by merging those states in the continuation automaton which have the same continuation when indices are removed. We therefore have the result of Champarnaud and Ziadi [6]

Theorem 18. $\mathbf{A}_{\mathrm{pd}}(\alpha) \simeq \mathbf{A}_{\mathrm{pos}}(\alpha)/_{\equiv_c}$.

Notice that we gave also a proof for the result of Antimirov [2] that $|PD(\alpha)| \leqslant |\alpha|_A + 1$.

Example 19. For the regular expression τ from Example 5, we construct in Fig. 8 the automaton $\mathbf{A}_{\mathrm{pd}}(\tau)$; the classes of the equivalence $\equiv_c$ are also shown. According to Theorem 18, we have $\mathbf{A}_{\mathrm{pd}}(\tau) \simeq \mathbf{A}_{\mathrm{pos}}(\tau)/_{\equiv_c}$ as it can be seen by comparing with Fig. 5, where $\mathbf{A}_{\mathrm{pos}}(\tau)$ is shown.

7. Follow automata

In this section we give our new algorithm for constructing NFAs from regular expressions. The idea is very simple: just eliminate (in a certain way, to be made precise below) the ε-transitions from the $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$. Essentially, for any path labelled ε, $p \overset{\varepsilon}{\rightsquigarrow} q$, and any transition $q \xrightarrow{a} r$, we add a transition $p \xrightarrow{a} r$. The obtained automaton is called *follow* NFA, denoted

$$\mathbf{A}_{\mathrm{f}}(\alpha) = (Q_f, A, \delta_f, 0_f, F_f).$$

We give below the precise details of the elimination of ε-transitions from $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$. We notice that, due to improvement (b) in Algorithm 4, there are no ε-cycles in $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$.

Algorithm 20. Given $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$, the algorithm constructs $\mathbf{A}_{\mathrm{f}}(\alpha)$.

```
1. $F_f \leftarrow \{q_f\}$
2. sort topologically $Q_f^{\varepsilon}$ w.r.t. the order $p \leqslant q$ iff $p \xrightarrow{\varepsilon} q \in \delta_f^{\varepsilon}$;
3. denote the ordered $Q_f^{\varepsilon} = (q_1, q_2, \ldots, q_r)$
4. for $i$ from $r$ downto 1 do
5.     for each transition $q_i \xrightarrow{\varepsilon} p$ do
6.         for each transition $p \xrightarrow{a} q$ do
7.             if $q_i \xrightarrow{a} q \notin \delta_f^{\varepsilon}$ then add $q_i \xrightarrow{a} q$ to $\delta_f^{\varepsilon}$
8.         if $p \in F_f$ then add $q_i$ to $F_f$
9.         remove the transition $q_i \xrightarrow{\varepsilon} p$
```

```
10. for each $q \in Q_f^\varepsilon - \{0_f\}$ such that there is no $p \xrightarrow{a} q$ in $\delta_f^\varepsilon$ do
11.     eliminate $q$ from $Q_f^\varepsilon$ and all transitions involving $q$ from $\delta_f^\varepsilon$
12. $Q_f \leftarrow Q_f^\varepsilon$; $\delta_f \leftarrow \delta_f^\varepsilon$
13. return $\mathbf{A}_\mathrm{f}(\alpha) = (Q_f, A, \delta_f, 0_f, F_f)$
```

Theorem 21. *For any regular expression α, $\mathbf{A}_\mathrm{f}(\alpha)$ is an NFA accepting $L(\alpha)$ which can be constructed in time and space $\mathcal{O}(|\alpha|^2)$.*

Proof. For the first assertion, it should be clear from Algorithm 20 that $L(\mathbf{A}_\mathrm{f}(\alpha)) = L(\mathbf{A}_\mathrm{f}^\varepsilon(\alpha))$. We then use Theorem 6(i).

The complexity is given by the number of pairs $(p \xrightarrow{\varepsilon} q, q \xrightarrow{a} r)$ which are considered in the algorithm. There are $\mathcal{O}(|\alpha|)$ ε-transitions and $\mathcal{O}(|\alpha|)$ transitions labelled by the same letter which leave a certain state. Assuming A is fixed, we obtain the result. □

Example 22. We give an example of an application of Algorithm 20. For the same regular expression $\tau = (a + b)(a^* + ba^* + b^*)^*$ from Example 5, we build in Fig. 9 the automaton $\mathbf{A}_\mathrm{f}(\tau)$; compare with Example 5 to see the ε-elimination.

We conclude this section with some very important comments concerning both the size of $\mathbf{A}_\mathrm{f}(\alpha)$ and the running time of Algorithm 20 which builds it. The worst case in Theorem 21 is reached for instance for the regular expression of [14], that is, $\alpha = (a_1 + \varepsilon)(a_2 + \varepsilon) \cdots (a_n + \varepsilon)$. However, in most examples (see also the examples at the end) both the size of $\mathbf{A}_\mathrm{f}(\alpha)$ and the running time of Algorithm 20 are linear. Also, we do not have examples where the ε-elimination requires essentially more time than the size of $\mathbf{A}_\mathrm{f}(\alpha)$. This remains an open problem. We finally notice that our ε-elimination algorithm is different from, and faster than, the classical one of [13]. The difference is that we do not compute ε-closures.

8. $\mathbf{A}_\mathrm{f}$ is a quotient of $\mathbf{A}_\mathrm{pos}$

We prove in this section that $\mathbf{A}_\mathrm{f}(\alpha)$ introduced above is a quotient of $\mathbf{A}_\mathrm{pos}(\alpha)$. This is unexpected because the construction of $\mathbf{A}_\mathrm{f}(\alpha)$ does not have, apparently, anything to do with positions. However, the consequences of this result are very important.
We start by defining the equivalence $\equiv_f \subseteq \mathsf{pos}_0(\alpha)^2$ by

$$i \equiv_f j \quad \text{iff (i) both } i, j \text{ or none belong to } \mathsf{last}(\alpha) \text{ and} \\ \text{(ii) } \mathsf{follow}(\alpha, i) = \mathsf{follow}(\alpha, j)$$

Notice that we restrict the equivalence so that we do not make equivalent final and non-final states in $\mathbf{A}_\mathrm{pos}(\alpha)$. The maim result of this section follows.

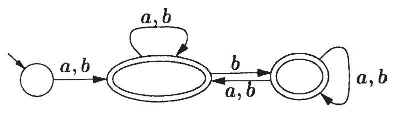

Fig. 9. $\mathbf{A}_\mathrm{f}(\tau)$ for $\tau = (a + b)(a^* + ba^* + b^*)^*$.

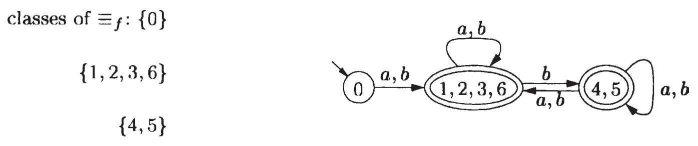

Fig. 10. $\mathbf{A}_{\mathrm{f}}(\tau) \simeq \mathbf{A}_{\mathrm{pos}}(\tau)/_{\equiv_f}$ for $\tau = (a+b)(a^* + ba^* + b^*)^*$.

Theorem 23. $\mathbf{A}_{\mathrm{f}}(\alpha) \simeq \mathbf{A}_{\mathrm{pos}}(\alpha)/_{\equiv_f}$.

We notice first that the restriction we imposed on $\equiv_f$ so that final and non-final states in $\mathsf{pos}_0(\alpha)$ cannot be $\equiv_f$-equivalent is essential, as shown by the expression $\alpha = (a^*b)^*$. Here $\mathsf{follow}(\alpha, i) = \{1, 2\}$, for any $0 \leqslant i \leqslant 2$. However, merging all three states of $\mathbf{A}_{\mathrm{pos}}(\alpha)$ is an error as the resulting automaton would accept the language $(a+b)^*$.

Example 24. Here is an example of an application of Theorem 23. For the same regular expression $\tau = (a+b)(a^* + ba^* + b^*)^*$ from Example 5, we build in Fig. 10 the $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\tau)$ and then give the equivalence classes of $\equiv_f$ and the automaton $\mathbf{A}_{\mathrm{f}}(\tau)$.

We move next to the proof of Theorem 23. First of all we need to see that we are allowed to make the quotient of the position automaton by the equivalence $\equiv_f$.

Lemma 25. *The equivalence $\equiv_f$ is right invariant w.r.t. $\mathbf{A}_{\mathrm{pos}}(\alpha)$.*

Proof. The first condition, compatibility with the set of final states $\mathsf{last}_0(\alpha)$, is verified by the definition of $\equiv_f$. For the second condition, consider $i \in \mathsf{last}_0(\alpha)$, $a \in A$. We have $\delta_{\mathrm{pos}}(i, a) = \{k \in \mathsf{pos}(\alpha) \mid k \in \mathsf{follow}(\alpha, i), \overline{a_k} = a\}$ and so, if $i \equiv_f j$, then $\delta(i, a) = \delta(j, a)$ and the claim follows. $\square$

The following well-known properties of these mappings will be used in the sequel:

$$
\begin{aligned}
\mathsf{first}(\beta+\gamma) &= \mathsf{first}(\beta) \cup \mathsf{first}(\gamma),\\
\mathsf{first}(\beta\gamma) &= \mathsf{first}(\beta) \cup \varepsilon(\beta)\mathsf{first}(\gamma),\\
\mathsf{first}(\beta^*) &= \mathsf{first}(\beta),\\
\mathsf{last}(\beta+\gamma) &= \mathsf{last}(\beta) \cup \mathsf{last}(\gamma),\\
\mathsf{last}(\beta\gamma) &= \mathsf{last}(\gamma) \cup \varepsilon(\gamma)\mathsf{last}(\beta),\\
\mathsf{last}(\beta^*) &= \mathsf{last}(\beta),\\
\mathsf{follow}(\beta+\gamma, i) &= \begin{cases} \mathsf{follow}(\beta, i) \text{ if } i \in \mathsf{pos}(\beta),\\ \mathsf{follow}(\gamma, i) \text{ if } i \in \mathsf{pos}(\gamma),\end{cases}\\
\mathsf{follow}(\beta\gamma, i) &= \begin{cases} \mathsf{follow}(\beta, i) \text{ if } i \in \mathsf{pos}(\beta) - \mathsf{last}(\beta),\\ \mathsf{follow}(\beta, i) \cup \mathsf{first}(\gamma) \quad \text{if } i \in \mathsf{last}(\beta),\\ \mathsf{follow}(\gamma, i) \text{ if } i \in \mathsf{pos}(\gamma),\end{cases}\\
\mathsf{follow}(\beta^*, i) &= \begin{cases} \mathsf{follow}(\beta, i) \quad \text{if } i \in \mathsf{pos}(\beta) - \mathsf{last}(\beta),\\ \mathsf{follow}(\beta, i) \cup \mathsf{first}(\beta) \quad \text{if } i \in \mathsf{last}(\beta).\end{cases}
\end{aligned}
\tag{4}
$$

Also, we shall need several results before proving Theorem 23. First, it is clear that $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$ is obtained from $\overline{\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})}$ by eliminating multiple transitions, if any. Therefore, $\mathbf{A}_{\mathrm{f}}(\alpha)$ is obtained from $\overline{\mathbf{A}_{\mathrm{f}}(\overline{\alpha})}$ in the same way. Also, $\mathbf{A}_{\mathrm{pos}}(\alpha) = \overline{\mathbf{A}_{\mathrm{pos}}(\overline{\alpha})}$, which implies that $\mathbf{A}_{\mathrm{pos}}(\alpha)/_{\equiv_f}$ is obtained by eliminating multiple transitions from $\overline{\mathbf{A}_{\mathrm{pos}}(\overline{\alpha})/_{\equiv_f}}$. Consequently, it is enough to prove that $\mathbf{A}_{\mathrm{f}}(\overline{\alpha}) \simeq \mathbf{A}_{\mathrm{pos}}(\overline{\alpha})/_{\equiv_f}$. Notice that $\equiv_f$ is right invariant w.r.t. $\mathbf{A}_{\mathrm{pos}}(\overline{\alpha})$.
We define the function

$$\mathsf{m} : \mathsf{pos}_0(\alpha) \longrightarrow Q_f^{\varepsilon} \qquad \mathsf{m}(0) = 0_f \text{ and}$$
$$\mathsf{m}(i) = p, \text{ if } i \neq 0 \text{ and } q \xrightarrow{a_i} p, \text{ for some } q \in Q_f^{\varepsilon}.$$

There is a single transition labelled a_i in $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$, so $\mathsf{m}(i)$ is well defined as its target. Because the initial states of $\mathbf{A}_{\mathrm{f}}(\overline{\alpha})$ and $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$ are the same and all transitions labelled a_i in $\mathbf{A}_{\mathrm{f}}(\overline{\alpha})$ have the same target state, m can be equivalently defined as $\mathsf{m} : \mathsf{pos}_0(\alpha) \longrightarrow Q_f$ by $\mathsf{m}(0) = 0_f$ and, for $i \neq 0$, $\mathsf{m}(i) = p$, for any $p \in Q_f$ such that there is an transition labelled a_i which is incoming to p. Notice that m is onto Q_f as the states of $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$ which have all incoming transitions labelled ε were removed by Algorithm 20. The function m will be the isomorphism we look for.
We prove next several results concerning the function m. For two states p and q, we denote the fact that there is a path labelled ε form p to q by $p \overset{\varepsilon}{\rightsquigarrow} q$; this path can also empty, that is, $p = q$.

Lemma 26. *For any $i, j \in \mathsf{pos}(\alpha)$, we have*
(i) $i \in \mathsf{first}(\alpha)$ *iff there is* $0_f \overset{\varepsilon}{\rightsquigarrow} p \xrightarrow{a_i} \mathsf{m}(i)$ *in* $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$.
(ii) $i \in \mathsf{last}(\alpha)$ *iff there is* $\mathsf{m}(i) \overset{\varepsilon}{\rightsquigarrow} q_f$ *in* $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$.
(iii) $j \in \mathsf{follow}(\alpha, i)$ *iff there is* $\mathsf{m}(i) \overset{\varepsilon}{\rightsquigarrow} p \xrightarrow{a_j} \mathsf{m}(j)$ *in* $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$.

Proof. The assertions follow from the definitions of first, last, and follow in (1) and the equality $L(\overline{\alpha}) = L(\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha}))$ in Theorem 6(i). □

Lemma 26 implies that, to show the isomorphism $\mathbf{A}_{\mathrm{f}}(\overline{\alpha}) \simeq \mathbf{A}_{\mathrm{pos}}(\overline{\alpha})/_{\equiv_f}$, it is enough to prove that, for any $i, j \in \mathsf{pos}_0(\alpha)$, $i \equiv_f j$ iff $\mathsf{m}(i) = \mathsf{m}(j)$. If we define the equivalence $\equiv_m = \{(i, j) \mid \mathsf{m}(i) = \mathsf{m}(j)\}$, then we have to show $\equiv_m = \equiv_f$. Indeed, assume this holds. Lemma 26(ii) assures that final states of $\mathbf{A}_{\mathrm{pos}}(\overline{\alpha})/_{\equiv_f}$ are mapped to final states of $\mathbf{A}_{\mathrm{f}}(\overline{\alpha})$. Then, we have a transition $[i]_{\equiv_f} \xrightarrow{a_j} [j]_{\equiv_f}$ in $\mathbf{A}_{\mathrm{pos}}(\overline{\alpha})/_{\equiv_f}$ iff $j \in \mathsf{follow}(\alpha, i)$ iff (by Lemma 26(i)(iii)) $\mathsf{m}(i) \overset{\varepsilon}{\rightsquigarrow} p \xrightarrow{a_j} \mathsf{m}(j)$ in $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$ iff $\mathsf{m}(i) \xrightarrow{a_j} \mathsf{m}(j)$ in $\mathbf{A}_{\mathrm{f}}(\overline{\alpha})$. The isomorphism follows.
The next result concerning initial states follows from Lemma 26.

Corollary 27. *We have that* $\mathsf{m}^{-1}(0_f) = \{0\}$ *iff there is no incoming transition labelled by some* a_i *to* 0_f *in* $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$ (*or, equivalently, in* $\mathbf{A}_{\mathrm{f}}(\overline{\alpha})$). *Also, if* $i \in \mathsf{m}^{-1}(0_f)$, *then* $\mathsf{follow}(\alpha, i) = \mathsf{first}(\alpha)$.

We make an observation concerning notations, such as 0_f, q_f, m, $\equiv_f$, etc. They depend on α but we omit α when it is understood; when it is not clear from the context, we add it as a further subscript, e.g., $0_{f,\alpha}$, $q_{f,\alpha}$, m_α, $\equiv_{f,\alpha}$, etc.
We shall need several further lemmata to prove our goal.

Lemma 28. *The final state $q_f \in Q_f^{\varepsilon}$ remains as a state in Q_f after Algorithm* 20 *iff there is $i \in \mathsf{last}(\alpha)$ such that $\mathsf{follow}(\alpha, i) = \emptyset$; moreover, in this case, for any $i \in \mathsf{last}(\alpha)$, $\mathsf{m}(i) = q_f$ iff $\mathsf{follow}(\alpha, i) = \emptyset$.*

Proof. The state q_f is not eliminated by Algorithm 20 if and only if there is a transition $q \xrightarrow{a_i} q_f$ in $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$. By definition of m, we have $\mathsf{m}(i) = q_f$ and Lemma 26(ii)(iii) give that $i \in \mathsf{last}(\alpha)$ and $\mathsf{follow}(\alpha, i) = \emptyset$. Conversely, assume $i \in \mathsf{last}(\alpha)$ with $\mathsf{follow}(\alpha, i) = \emptyset$. By Lemma 26(ii), there is a path $\mathsf{m}(i) \overset{\varepsilon}{\rightsquigarrow} q_f$ in $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$. Assume this path is not empty and consider the last transition of it, $q \xrightarrow{\varepsilon} q_f$. According to the construction of $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$ in Algorithm 4, this ε-transition may appear in two ways: from an ε initial y in α or from a ‘*’ in α. In the former case, there must be (because α is reduced) a path from q to q_f which has at least one transition labelled by some a_j. Thus, by Lemma 26(iii), $\mathsf{follow}(\alpha, i) \neq \emptyset$, a contradiction. In the latter case, we obtain a similar contradiction; as α is reduced, there must be a path as before from q to q. Therefore, it must be that $\mathsf{m}(i) = q_f$ and so q_f remains in $\mathbf{A}_{\mathrm{f}}(\overline{\alpha})$. Notice that we proved also the second statement. $\square$

Lemma 29. *For any $i \in \mathsf{last}(\alpha)$ such that $\emptyset \neq \mathsf{follow}(\alpha, i) \subseteq \mathsf{first}(\alpha)$, there is $0_f \overset{\varepsilon}{\rightsquigarrow} \mathsf{m}(i) \overset{\varepsilon}{\rightsquigarrow} q_f$ in* $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\overline{\alpha})$.

Proof. By induction on α. Denote the property to be proved $P_1(\alpha, i)$.

If $\alpha \in \{\emptyset, \varepsilon, a\}$, then the property is true. When α has at least one operator, assume P_1 true for all subexpressions of α and let us prove it for α. We shall use (4).

(1) $\alpha = \beta + \gamma$. Assume $i \in \mathsf{last}(\beta)$. The case $i \in \mathsf{last}(\gamma)$ is similar. Then $\emptyset \neq \mathsf{follow}(\beta, i) \subseteq \mathsf{first}(\beta) \cup \mathsf{first}(\gamma)$ and hence $\emptyset \neq \mathsf{follow}(\beta, i) \subseteq \mathsf{first}(\beta)$. Therefore, by the inductive hypothesis, $P_1(\beta, i)$ is true and so is $P_1(\alpha, i)$.

(2) $\alpha = \beta\gamma$. If $\varepsilon(\gamma) = \emptyset$, then $i \in \mathsf{last}(\gamma)$ with $\emptyset \neq \mathsf{follow}(\gamma, i) \subseteq \mathsf{first}(\beta) \cup \varepsilon(\beta)\ \mathsf{first}(\gamma)$ and so it must be that $\varepsilon(\beta) = \varepsilon$ and $\emptyset \neq \mathsf{follow}(\gamma, i) \subseteq \mathsf{first}(\gamma)$. Now, the inductive hypothesis gives that $P_1(\gamma, i)$ holds, in particular $\varepsilon(\gamma) = \varepsilon$, a contradiction. Thus, we have $\varepsilon(\gamma) = \varepsilon$.
Now, if $i \in \mathsf{last}(\gamma)$, then, as above, we get $\varepsilon(\beta) = \varepsilon$ and the inductive hypothesis gives $P_1(\gamma, i)$. Together, these imply $P_1(\alpha, i)$.
When $i \in \mathsf{last}(\beta)$, we have $\emptyset \neq \mathsf{follow}(\beta, i) \cup \mathsf{first}(\gamma) \subseteq \mathsf{first}(\beta) \cup \varepsilon(\beta)\ \mathsf{first}(\gamma)$ and $\varepsilon(\beta) = \varepsilon$, as $\gamma \neq \varepsilon$ (α is reduced). If $\mathsf{follow}(\beta, i) = \emptyset$, then Lemma 28 gives that $\mathsf{m}(i) = q_{f,\beta}$ (the final state of $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\beta)$). Hence, $P_1(\alpha, i)$ holds. If $\mathsf{follow}(\beta, i) \neq \emptyset$, then the inductive hypothesis gives $P_1(\beta, i)$ which will give again $P_1(\alpha, i)$.

(3) $\alpha = \beta^*$. Then $i \in \mathsf{last}(\beta)$ and $\emptyset \neq \mathsf{follow}(\beta, i) \cup \mathsf{first}(\beta) \subseteq \mathsf{first}(\beta)$ which implies $\mathsf{follow}(\beta, i) \subseteq \mathsf{first}(\beta)$. If $\mathsf{follow}(\beta, i) \neq \emptyset$, then $P_1(\alpha, i)$ follows from the inductive hypothesis on β. If $\mathsf{follow}(\beta, i) = \emptyset$, we use again Lemma 28 and obtain $P_1(\alpha, i)$. $\square$

Lemma 30. *For any $i, j \in \mathsf{last}(\alpha)$ with $\emptyset \neq \mathsf{follow}(\alpha, i) - \mathsf{first}(\alpha) = \mathsf{follow}(\alpha, j) - \mathsf{first}(\alpha)$, we have either $i \equiv_f j$ or $\mathsf{m}(i) = \mathsf{m}(j)$.*

Proof. By induction on α. Denote the property to be proved by $P_2(\alpha, i, j)$. For $\alpha \in \{\emptyset, \varepsilon, a\}$ there is nothing to prove. We assume next α has at least one operator and P_2 is true for all subexpressions of α.

(1) $\alpha = \beta + \gamma$. Then i and j are both in either $\mathsf{last}(\beta)$ or $\mathsf{last}(\gamma)$ and the property follows from (4) and the inductive hypothesis.

(2) $\alpha = \beta\gamma$. We use (4). Assume first $i, j \in \mathsf{last}(\gamma)$. If $\varepsilon(\beta) = \varepsilon$, then the inductive hypothesis gives $P_2(\gamma, i, j)$ which, in turn, implies $P_2(\alpha, i, j)$. If $\varepsilon(\beta) = \emptyset$, then $i \equiv_{f,\alpha} j$. If $i, f \in \mathsf{last}(\beta)$, then $\varepsilon(\gamma) = \varepsilon$. If $\varepsilon(\beta) = \varepsilon$, we can use inductive hypothesis on β. Assume $\varepsilon(\beta) = \emptyset$. Then $\mathsf{follow}(\beta, i) - \mathsf{first}(\beta) = \mathsf{follow}(\beta, j) - \mathsf{first}(\beta)$. If both members of the last equality are non-empty, then we can again use the inductive hypothesis on β. Otherwise, for any $k \in \{i, j\}$, $\mathsf{follow}(\beta, k) = \emptyset$; if non-empty, then Lemma 29 would give $P_1(\beta, k)$, implying $\varepsilon(\beta) = \varepsilon$, a contradiction. Therefore, by Lemma 28, we get $\mathsf{m}(i) = \mathsf{m}(j)$.

The remaining possibility is $i \in \mathsf{last}(\beta)$, $j \in \mathsf{last}(\gamma)$; we have also $\varepsilon(\gamma) = \varepsilon$. The equality $\mathsf{follow}(\alpha, i) - \mathsf{first}(\alpha) = \mathsf{follow}(\alpha, j) - \mathsf{first}(\alpha)$ is possible only if $\varepsilon(\beta) = \emptyset$, $\mathsf{follow}(\beta, i) \subseteq \mathsf{first}(\beta)$, and $\mathsf{follow}(\gamma, j) = \mathsf{first}(\gamma)$. Also, it must be that $\mathsf{follow}(\beta, i) = \emptyset$, as otherwise Lemma 29 would give $\varepsilon(\beta) = \varepsilon$, a contradiction. Therefore, $i \equiv_{f,\alpha} j$.

(3) $\alpha = \beta^*$. Then $P_2(\alpha, i, j)$ follows from the inductive hypothesis on β. $\square$

Proof of Theorem 23. We can start now the proof of the equality $\equiv_f = \equiv_m$ which, as argued before, is enough to prove the statement of Theorem 23. We do this again by induction on α. If $\alpha \in \{\emptyset, \varepsilon, a\}$, then $\equiv_{f,\alpha} = \equiv_{m,\alpha} = \emptyset$. Assume α has at least one operator and that the property holds for all subexpressions of α. We shall tacitly use (4). Also, recall that all expressions are assumed to be reduced.

(1) $\alpha = \beta + \gamma$. Corollary 27 gives $\mathsf{m}^{-1}(0_{f,\alpha}) = \{0\}$. Consider first the case when $\beta = \varepsilon$; the case $\gamma = \varepsilon$ is symmetric. If $i \neq 0$ and $i \equiv_{f,\alpha} 0$, then $\mathsf{follow}(\gamma, i) = \mathsf{first}(\gamma) \neq \emptyset$, and so, by Lemma 29, $\varepsilon \in L(\gamma)$, contradiction with α reduced. Therefore, $\equiv_{f,\alpha} = \equiv_{f,\gamma} \cap \mathsf{pos}(\alpha)^2$. Since also $\equiv_{m,\alpha} = \equiv_{m,\gamma} \cap \mathsf{pos}(\alpha)^2$, the inductive hypothesis implies $\equiv_{f,\alpha} = \equiv_{m,\alpha}$.

Assume now $\beta \neq \varepsilon$, $\gamma \neq \varepsilon$. We know that no $i \neq 0$ can have $i \equiv_{f,\alpha} 0$. Take $i \neq 0$, $j \neq 0$ such that $i \equiv_{f,\alpha} j$. If i and j are both in $\mathsf{pos}(\beta)$ or $\mathsf{pos}(\gamma)$, then $i \equiv_{f,\beta} j$ or $i \equiv_{f,\gamma} j$, respectively. If not, then $i \in \mathsf{last}(\beta)$, $j \in \mathsf{last}(\gamma)$, and $\mathsf{follow}(\beta, i) = \emptyset = \mathsf{follow}(\gamma, j)$. Therefore,

$$\begin{aligned}\equiv_{f,\alpha} = {} & ((\equiv_{f,\beta} \cup \equiv_{f,\gamma}) \cap \mathsf{pos}(\alpha)^2) \\ & \cup \{(i, j) \in \mathsf{last}(\beta) \times \mathsf{last}(\gamma) \mid \mathsf{follow}(\beta, i) = \emptyset = \mathsf{follow}(\gamma, j)\}.\end{aligned}$$

According to Algorithm 4 and Corollary 27, $\mathsf{m}_\beta^{-1}(0_{f,\alpha})$ is either $\{0\}$ or empty (in the case of (a1)). Similarly, using Lemma 28, $\mathsf{m}_\beta^{-1}(q_{f,\alpha})$ contains those $i \in \mathsf{pos}(\alpha)$ with $\mathsf{follow}(\alpha, i) = \emptyset$. Therefore, $\equiv_{f,\alpha} = \equiv_{m,\alpha}$.

(2) $\alpha = \beta\gamma$. Since α is reduced, both β and γ are different from ε. Hence, for $i \neq 0$, we have $i \equiv_{f,\alpha} 0$ iff $i \in \mathsf{pos}(\beta)$, $i \equiv_{f,\beta} 0$. This implies $\equiv_{f,\alpha} \cap (\{0\} \times \mathsf{pos}(\alpha)) = \equiv_{f,\beta} \cap (\{0\} \times \mathsf{pos}(\alpha))$.

Take $i \neq 0$, $j \neq 0$, such that $i \equiv_{f,\alpha} j$. If both i and j are in $\mathsf{pos}(\beta)$ or $\mathsf{pos}(\gamma)$, then $i \equiv_{f,\beta} j$ or $i \equiv_{f,\gamma} j$, respectively. The converse holds as well. If $i \in \mathsf{pos}(\beta)$, $j \in \mathsf{pos}(\gamma)$, then it must be that $i \in \mathsf{last}(\beta)$, $\mathsf{follow}(\beta, i) = \emptyset$, and $j \equiv_{f,\gamma} 0$. The converse is also true. Therefore, we have

$$\begin{aligned}\equiv_{f,\alpha} = {} & \equiv_{f,\beta} \\ & \cup (\equiv_{f,\gamma} \cap \mathsf{pos}(\gamma)^2) \\ & \cup \{(i, j) \in \mathsf{last}(\beta) \times \mathsf{pos}(\gamma) \mid \mathsf{follow}(\beta, i) = \emptyset,\ j \equiv_{f,\gamma} 0\}.\end{aligned}$$

Consider now $\equiv_{m,\alpha}$. The positions mapped to the same states by m_β or m_γ will also be mapped the same by m_α. Also, the positions mapped by m_α to $0_{f,\alpha}$ are precisely those mapped this way by m_β. According to Algorithm 4 (and its improvement (a)) and Lemma 28, the positions i in β with $\mathsf{follow}(\beta, i) = \emptyset$ and those j in γ with $m_\gamma(j) = 0_{f,\gamma}$ are mapped to the same state. Now the inductive hypothesis shows that $\equiv_{f,\alpha} = \equiv_{m,\alpha}$.

(3) $\alpha = \beta^*$. Consider first $i \neq 0$, $i \equiv_{f,\alpha} 0$. Lemma 29 gives that $i \in \mathsf{pos}(\beta) - \mathsf{last}(\beta)$ is not possible. Thus $i \in \mathsf{last}(\beta)$ with $\mathsf{follow}(\beta, i) \subseteq \mathsf{first}(\beta)$. So, either $\mathsf{follow}(\beta) = \emptyset$ or, by Lemma 29, there is $0_{f,\beta} \overset{\varepsilon}{\rightsquigarrow} \mathsf{m}(i) \overset{\varepsilon}{\rightsquigarrow} q_{f,\beta}$ in $\mathbf{A}_\mathrm{f}^\varepsilon(\overline{\beta})$. The converse holds true because of Lemma 26. Therefore

$$\begin{aligned}\equiv_{f,\alpha} \cap(\{0\} \times \mathsf{pos}(\alpha)) = \{(0,i) \mid i \in \mathsf{last}(\beta), 0_{f,\beta} \overset{\varepsilon}{\rightsquigarrow} \mathsf{m}(i) \overset{\varepsilon}{\rightsquigarrow} q_{f,\beta} \text{ in } \mathbf{A}_\mathrm{f}^\varepsilon(\overline{\beta})\}\\ \cup\{(0,i) \mid i \in \mathsf{last}(\beta), \mathsf{follow}(\beta,i) = \emptyset\}.\end{aligned}$$

It can be seen now that $\equiv_{f,\alpha} \cap(\{0\} \times \mathsf{pos}(\alpha)) = \equiv_{m,\alpha} \cap(\{0\} \times \mathsf{pos}(\alpha))$ because of the definition of $\mathbf{A}_\mathrm{f}^\varepsilon$ in Algorithm 4.

Consider next $i \neq 0$, $j \neq 0$ such that $i \equiv_{f,\alpha} j$. If $i, j \in \mathsf{pos}(\beta) - \mathsf{last}(\beta)$, then $i \equiv_{f,\beta} j$. If $i, j \in \mathsf{last}(\beta)$, then $\mathsf{follow}(\beta, i) \cup \mathsf{first}(\beta) = \mathsf{follow}(\beta, j) \cup \mathsf{first}(\beta)$. If one of $\mathsf{follow}(\beta, i)$ and $\mathsf{follow}(\beta, j)$ is a subset of $\mathsf{first}(\beta)$, then the other is also and, for any $k \in \{i, j\}$ we have that either $\mathsf{follow}(\beta, k) = \emptyset$ or $\mathsf{follow}(\beta, k) \neq \emptyset$; in the latter case, by Lemma 29, there is $0_{f,\beta} \overset{\varepsilon}{\rightsquigarrow} \mathsf{m}(k) \overset{\varepsilon}{\rightsquigarrow} q_{f,\beta}$ in $\mathbf{A}_\mathrm{f}^\varepsilon(\overline{\beta})$. On the other hand, if none of $\mathsf{follow}(\beta, i)$ and $\mathsf{follow}(\beta, j)$ is included in $\mathsf{first}(\beta)$, then $\emptyset \neq \mathsf{follow}(\beta, i) - \mathsf{first}(\beta) = \mathsf{follow}(\beta, j) - \mathsf{first}(\beta)$, which gives, by Lemma 30 and the inductive hypothesis on β, that $i \equiv_{f,\beta} j$.

We have proved that

$$\begin{aligned}cl \equiv_{f,\alpha} \cap\, \mathsf{pos}(\alpha)^2 = (\equiv_{f,\beta} \cap \mathsf{pos}(\beta)^2)\\ \cup\{(i,j) \in \mathsf{last}(\beta)^2 \mid \forall k \in \{i,j\},\ \text{either } \mathsf{follow}(\beta,k) = \emptyset\\ \text{or there is } 0_{f,\beta} \overset{\varepsilon}{\rightsquigarrow} \mathsf{m}(i) \overset{\varepsilon}{\rightsquigarrow} q_{f,\beta} \text{ in } \mathbf{A}_\mathrm{f}^\varepsilon(\overline{\beta})\}.\end{aligned}$$

Now, again by the definition of $\mathbf{A}_\mathrm{f}^\varepsilon$ in Algorithm 4 and the improvement in (b), we have $\equiv_{f,\alpha} \cap\, \mathsf{pos}(\alpha)^2 = \equiv_{m,\alpha} \cap\, \mathsf{pos}(\alpha)^2$. Therefore, $\equiv_{f,\alpha} = \equiv_{m,\alpha}$, and the proof of Theorem 23 is completed. □

So, we have that both follow and partial derivative automata are quotients of the position automaton. As it will be seen in Section 10, the two quotients are incomparable. Let us further remark that [16] investigates further such quotients and shows how to build the largest right-invariant equivalence w.r.t. the position automaton, which gives the smallest quotient, therefore smaller than either of follow or partial derivative automaton. However, it is an open problem how to compute that quotient fast; according to [16], it can be computed in polynomial time.

9. $\mathbf{A}_\mathrm{f}$ uses optimally the positions

Finally, we show that the follow automaton $\mathbf{A}_\mathrm{f}(\alpha)$ uses the whole information which comes from positions of α. Indeed, the follow automaton for marked expressions cannot be improved. $\mathbf{A}_\mathrm{f}(\overline{\alpha})$ is a deterministic automaton and let the minimal automaton equivalent to it be $\mathsf{min}(\mathbf{A}_\mathrm{f}(\overline{\alpha}))$. Then $\overline{\mathsf{min}(\mathbf{A}_\mathrm{f}(\overline{\alpha}))}$ is an NFA accepting $L(\alpha)$ which can be computed in time $\mathcal{O}(|\alpha|^2 \log |\alpha|)$ using the minimization algorithm of Hopcroft [12]. This is, in fact, another way of using positions to compute NFAs for regular expressions. However, it is interesting to see that $\overline{\mathsf{min}(\mathbf{A}_\mathrm{f}(\overline{\alpha}))}$ brings no improvement over $\mathbf{A}_\mathrm{f}(\alpha)$.

Theorem 31. $\overline{\mathsf{min}(\mathbf{A}_\mathrm{f}(\overline{\alpha}))} \simeq \mathbf{A}_\mathrm{f}(\alpha)$.

Proof. It is enough to show that $\mathsf{min}(\mathbf{A}_\mathrm{f}(\overline{\alpha})) \simeq \mathbf{A}_\mathrm{f}(\overline{\alpha})$, that is, $\mathbf{A}_\mathrm{f}(\overline{\alpha})$ is already minimal. We first complete the automaton $\mathbf{A}_\mathrm{f}(\overline{\alpha})$; we add a new non-final state, denoted $\emptyset$, and all missing transitions will go to it. Denote the completed automaton by $\mathbf{A}_\mathrm{f}^\emptyset(\overline{\alpha})$. Consider two positions i and j which have different follow sets and, with no loss of generality, take $k \in \mathsf{follow}(\alpha, i) - \mathsf{follow}(\alpha, j)$. Then, there is a word $w \in \overline{A}^*$ such that $a_k w$ takes the automaton $\mathbf{A}_{\mathrm{f}}^{\emptyset}(\overline{\alpha})$ from the state i to a final state. On the other hand, $a_k w$ takes the automaton $\mathbf{A}_{\mathrm{f}}^{\emptyset}(\overline{\alpha})$ from the state j to $\emptyset$. Therefore, i and j cannot be merged. Since they have been arbitrarily chosen, the automaton $\mathbf{A}_{\mathrm{f}}(\overline{\alpha})$ is minimal. □

Notice also that computing $\mathbf{A}_{\mathrm{f}}(\alpha)$ by ε-elimination in $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha)$ is faster than using Hopcroft's algorithm [12] plus unmarking.

10. Comparing $\mathbf{A}_{\mathrm{f}}$ with other constructions

We discuss in this section some examples to compare the follow automaton with the best constructions to date. We shall include also comparison with the common follow sets automaton of [14], denoted below by $\mathbf{A}_{\mathrm{cfs}}(\alpha)$. We do not include here the very long description of $\mathbf{A}_{\mathrm{cfs}}$ which can be found in [14] or [11].

We start with some examples showing that $\mathbf{A}_{\mathrm{f}}$ can be much smaller than either of $\mathbf{A}_{\mathrm{pos}}$ and $\mathbf{A}_{\mathrm{pd}}$ and that $\mathbf{A}_{\mathrm{f}}$ is incomparable with either of $\mathbf{A}_{\mathrm{pd}}$ and $\mathbf{A}_{\mathrm{cfs}}$.

Example 32. Consider α_n from Example 3. The follow automaton is smaller than all the others:
$|\mathbf{A}_{\mathrm{pos}}(\alpha_n)| = |\mathbf{A}_{\mathrm{pd}}(\alpha_n)| = \Theta(|\alpha_n|^2)$,
$|\mathbf{A}_{\mathrm{f}}(\alpha_n)| = \Theta(|\alpha_n|)$,
$|\mathbf{A}_{\mathrm{cfs}}(\alpha_n)| = \Theta(|\alpha_n|(\log(|\alpha_n|))^2)$.

Example 33. Consider the regular expression

$$\alpha_n = a_1(b_1 + \cdots + b_n)^* + a_2(b_1 + \cdots + b_n)^* + \ldots + a_n(b_1 + \cdots + b_n)^*.$$

We have now that the partial derivative automaton is the smallest:
$|\mathbf{A}_{\mathrm{pos}}(\alpha_n)| = \Theta(|\alpha_n|^{3/2})$,
$|\mathbf{A}_{\mathrm{f}}(\alpha_n)| = \Theta(|\alpha_n|)$,
$|\mathbf{A}_{\mathrm{pd}}(\alpha_n)| = \Theta(|\alpha_n|^{1/2})$, and
$|\mathbf{A}_{\mathrm{cfs}}(\alpha_n)| = \Theta(|\alpha_n|(\log(|\alpha_n|))^2)$.

Example 34. Consider the regular expression of [14]

$$\alpha_n = (a_1 + \varepsilon)(a_2 + \varepsilon)\cdots(a_n + \varepsilon).$$

In this case the common follow sets automaton is the smallest:
$|\mathbf{A}_{\mathrm{pos}}(\alpha_n)| = |\mathbf{A}_{\mathrm{f}}(\alpha_n)| = |\mathbf{A}_{\mathrm{pd}}(\alpha_n)| = \Theta(|\alpha_n|^2)$, and
$|\mathbf{A}_{\mathrm{cfs}}(\alpha_n)| = \Theta(|\alpha_n|(\log(|\alpha_n|))^2)$.

Next, we give some real-life examples which have some interesting common properties. For all of them, the follow automaton and the partial derivative automaton are isomorphic and smaller than the other two. These examples are:
- C-comments: `/*((A-{*})+**`*`(A-{*,/}))`*`**`*`/`
- floating point numbers:

`(0+···+9)(0+···+9)`*`.((0+···+9)(0+···+9)`*`+`ε`)(e+E)(++-+`ε`)`
`(0+···+9)(0+···+9)`*

- programming languages identifiers:
  `(a+··· z+A+··· Z)(a+··· z+A+··· Z+0+···+9)`*

If these examples are generalized to some parametrized examples we still have that $\mathbf{A}_\mathrm{f}$ and $\mathbf{A}_\mathrm{pd}$ are isomorphic and have linear size; the position automaton has quadratic size and the common follow sets automaton has size linear times the square of the logarithm. We show it only for the last example. Conclusions of these results are discussed in the next section.

Example 35. Consider the regular expression (generalized identifiers in programming languages)

$$\alpha_{n,m} = (a_1 + a_2 + \cdots + a_n)(a_1 + a_2 + \cdots + a_n + b_1 + b_2 + \cdots + b_m)^*.$$

We have
$|\mathbf{A}_\mathrm{f}(\alpha_{n,m})| = |\mathbf{A}_\mathrm{pd}(\alpha_{n,m})| = \Theta(|\alpha_{n,m}|)$,
$|\mathbf{A}_\mathrm{pos}(\alpha_{n,m})| = \Theta(|\alpha_{n,m}|^2)$, and
$|\mathbf{A}_\mathrm{cfs}(\alpha_{n,m})| = \Theta(|\alpha_{n,m}|(\log(|\alpha_{n,m}|))^2)$.

We finally notice that we did not compare our construction with the one of Chang and Paige [8] since we do not work with compressed automata.

11. Conclusions and further research

We gave two new algorithms to construct nondeterministic finite automata from regular expressions. The first constructs εNFAs which are smaller than all other similar constructions and also very close to optimal. The second constructs the follow NFAs which are conceptually by far the simplest compared to all the others: we construct the follow εNFA, which is elementary, and then eliminate the ε-transitions, which is again elementary. However, the resulting automata have interesting properties. The follow automaton is always a quotient of the position automaton, is very easy to compute, and is at least as small as all the other similarly constructed automata in most cases. We believe that these features will make these automata very attractive for practical purposes. Several problems should be investigated further.

First, it seems that the time required to build the follow automaton is linear in terms of its size. At least we do not have examples to prove the converse. We remark that the assertion is not true in general. There are examples of εNFAs for which the ε-elimination takes longer than both size of input and size of output.

Second, the follow automaton seems to have linear size in most cases. It is of interest to see which are those cases and when the size is far from linear. Also, the common follow sets automaton seems to have size linear times the logarithm squared in most cases. Some lower bounds on its size might bring some light here.

Third, a more rigorous comparison between the follow automaton and common follow sets or partial derivative automaton should be done. This seems difficult because average case analysis is, very likely, too complicated. Probably the only way to decide which one is better is by testing all of them in real-life applications.

References

[1] A. Aho, R. Sethi, J. Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, MA, 1988.
[2] V. Antimirov, Partial derivatives of regular expressions and finite automaton constructions, Theoret. Comput. Sci. 155 (1996) 291–319.
[3] G. Berry, R. Sethi, From regular expressions to deterministic automata, Theoret. Comput. Sci. 48 (1986) 117–126.
[4] A. Brüggemann-Klein, Regular expressions into finite automata, Theoret. Comput. Sci. 120 (1993) 197–213.
[5] J. Brzozowski, Derivatives of regular expressions, J. ACM 11 (1964) 481–494.
[6] J.-M. Champarnaud, D. Ziadi, New finite automaton constructions based on canonical derivatives, in: Proceedings of the CIAA 2000, LNCS 2088, Springer, Berlin, 2001, pp. 94–104.
[7] J.-M. Champarnaud, D. Ziadi, Computing the equation automaton of a regular expression in $\mathcal{O}(s^2)$ space and time, in: Proceedings of the 12th Combinatorial Pattern Matching (CPM 2001), LNCS 2089, Springer, Berlin, 2001, pp. 157–168.
[8] C.-H. Chang, R. Paige, From regular expressions to DFA's using compressed NFA's, Theoret. Comput. Sci 178 (1997) 1–36.
[9] J. Friedl, Mastering Regular Expressions, O'Reilly, 1998.
[10] V.M. Glushkov, The abstract theory of automata, Russian Math. Surveys 16 (1961) 1–53.
[11] C. Hagenah, A. Muscholl, Computing ϵ-free NFA from regular expressions in $O(n \log^2(n))$ time, Theor. Inform. Appl. 34 (4) (2000) 257–277.
[12] J. Hopcroft, An $n \log n$ algorithm for minimizing states in a finite automaton, in: Proceedings of the International Symposium on Theory of machines and computations, Technion, Haifa, Academic Press, New York, 1971, pp. 189–196.
[13] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, Mass, 1979.
[14] J. Hromkovic, S. Seibert, T. Wilke, Translating regular expressions into small ϵ-free nondeterministic finite automata, J. Comput. System Sci. 62 (4) (2001) 565–588.
[15] L. Ilie, S. Yu, Constructing NFAs by optimal use of positions in regular expressions, in: A. Apostolico, M. Takeda (Eds.), Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching (CPM) (Fukuoka, 2002), Lecture Notes in Comput. Sci., 2373, Springer, Berlin, 2002, pp. 279–288.
[16] L. Ilie, S. Yu, Algorithms for computing small NFAs, in: K. Diks, W. Rytter (Eds.), Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS), (Warszawa, 2002), Lecture Notes in Comput. Sci., 2420, Springer, Berlin, 2002, pp. 328–340.
[17] R. McNaughton, H. Yamada, Regular expressions and state graphs for automata, IEEE Trans. on Electronic Computers 9 (1) (1960) 39–47.
[18] S. Sippu, E. Soisalon-Soininen, Parsing Theory: I Languages and Parsing, EATCS Monographs on Theoretical Computer Science, vol. 15, Springer-Verlag, New York, 1988.
[19] K. Thompson, Regular expression search algorithm, Comm. ACM 11 (6) (1968) 419–422.
[20] S. Yu, Regular Languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. I, Springer-Verlag, Berlin, 1997, pp. 41–110.