Theoretical Computer Science 412 (2011) 5802–5807

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Note

# From regular expressions to smaller NFAs☆

Pedro García [a], Damián López [a,*], José Ruiz [a], Gloria I. Álvarez [b]

[a] Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 46022 Valencia, Spain
[b] Pontificia Universidad Javeriana, Cali, Colombia

ARTICLE INFO

*Article history:*
Received 21 December 2010
Accepted 27 May 2011
Communicated by D. Perrin

*Keywords:*
Regular expression
Finite automata
Position automata quotients

ABSTRACT

Several methods have been developed to construct λ-free automata that represent a regular expression. Among the most widely known are the position automaton (Glushkov), the partial derivatives automaton (Antimirov) and the follow automaton (Ilie and Yu). All these automata can be obtained with quadratic time complexity, thus, the comparison criterion is usually the size of the resulting automaton. The methods that obtain the smallest automata (although, for general expressions, they are not comparable), are the follow and the partial derivatives methods. In this paper, we propose another method to obtain a λ-free automaton from a regular expression. The number of states of the automata we obtain is bounded above by the size of both the partial derivatives automaton and of the follow automaton. Our algorithm also runs with the same time complexity of these methods.



## 1. Introduction

One of the problems that have been studied in automata theory is the development of algorithms to construct automata that represent regular expressions. The solution to this problem permits the efficient implementation of useful tools in fields like text processing. Recently, programming languages such as Perl, Python, Java, C♯ or PHP consider regular expressions as an extra tool that helps to decrease the programming effort.

One of the first methods for this task was the Thompson automaton [12], which is an inductive tool that defines the automata for the basic regular expressions together with rules to construct the automata for the different operations involved in a regular expression $\alpha$. The cost of this construction is linear in the number of symbols and operators involved in $\alpha$, which will be denoted $|\alpha|$, whereas in what follows, $\|\alpha\|$ will denote just the number of symbols.

The position automaton was proposed independently by Glushkov [9] and McNaughton and Yamada [10]. An intuitive algorithm to construct it starts considering the linearized version $\overline{\alpha}$ of a regular expression $\alpha$, that is, one in which the symbols are distinguished according to their position in $\alpha$. The number of states of the automaton is the number of occurrences of the symbols in $\alpha$ plus one (the initial state) and $(a_i, b, b_j)$ is a transition of the automaton if $b_j$ is a successor of $a_i$ in a word of $L(\overline{\alpha})$ and the symbol in the position $j$ of $\alpha$ is $b$. Several methods have been proposed to obtain this automaton with quadratic time complexity [13,8,4].

The partial derivatives automaton was proposed by Antimirov [1]. The concept of partial derivative can be seen as a non-deterministic extension of the Brzozowski's derivatives. The difference with the deterministic version arises when the result of a derivative is a union of regular expressions. This union is changed by a set containing the expressions. The construction of the automaton is very similar to Brzozowski's construction. Antimirov proposes a $\mathcal{O}(|\alpha|^2\|\alpha\|^3)$ algorithm to

☆ This work was partially supported by the Spanish Ministerio de Educación y Ciencia under project TIN2007-60769.
* Corresponding author. Tel.: +34 96 3877007; fax: +34 963877359.
*E-mail addresses:* pgarcia@dsic.upv.es (P. García), dlopez@dsic.upv.es (D. López), jruiz@dsic.upv.es (J. Ruiz), galvarez@dsic.upv.es (G.I. Álvarez).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.05.058

construct the partial derivatives automaton. Concerning this construction, it is shown in [6] that this automaton is a quotient of the position automaton by a certain equivalence relation and that it can be constructed in $\mathcal{O}(|\alpha|^2|\alpha\|)$ space and time complexities. This method, aimed to improve the time complexity of the partial derivatives algorithm, is called the equation automaton method and when it is applied to $\overline{\alpha}$ obtains the same automaton as the partial derivatives method applied to $\alpha$. Champarnaud and Ziadi propose in [5] an improved algorithm that runs in $\mathcal{O}(|\alpha|^2)$ space and time complexity.

The *follow automaton*, proposed by Ilie and Yu [11] is the quotient of the position automaton by the following equivalence relation: two states are equivalent if they have the same successors (follow) and the same membership to the set of final states. The algorithm they propose constructs an automaton in a similar way to the Thompson automaton, but with fewer states and without λ-loops. This allows the authors to develop an algorithm to eliminate the λ-transitions that works in $\mathcal{O}(|\alpha|^2)$.

We note that all these methods have quadratic time complexity. Therefore, in order to compare all these approaches it is important to take into account the size of the resulting automaton. Under this criterion, the best behavior is achieved by the partial derivatives and the follow methods. In [7] the authors prove that, when a normal form version of the regular expressions is considered, the partial derivatives method obtains automata with size bounded above by the size of the follows automaton. This transformation into a normal form can be carried out in linear time. When general expressions are considered, the size of the automata obtained from these methods cannot be compared.

In this paper we propose a new method to construct a λ-free automaton from a regular expression whose size is bounded above by the size of both the partial derivatives and of the follow automaton. Our method runs also with quadratic time complexity with respect to the size of the regular expression.

## 2. Definitions and notation

Let $A$ be a finite alphabet and $A^*$ the free monoid generated by $A$ with concatenation as the binary operation and $\lambda$ as neutral element. A language $L$ is any subset of $A^*$, the elements $x \in A^*$ are called *words*.

For any given language $L$ over $A^*$ and a word $u \in A^*$, the left quotient of $L$ by $u$ is defined as $u^{-1}L = \{v \in A^* \ : \ uv \in L\}$.

A *regular expression* can be recursively defined as follows:

1. $\emptyset$, $\lambda$ and $a \in A$ are regular expressions.
2. If $\alpha$ and $\beta$ are regular expressions then $\alpha + \beta, \alpha \cdot \beta, \alpha^*$ and $(\alpha)$ are also regular expressions.
3. All regular expressions can be obtained by applying the rules 1 and 2 finitely many times.

The regular language denoted by a regular expression $\alpha$ is $L(\alpha)$. Then $L((\alpha)) = L(\alpha), L(\emptyset) = \emptyset, L(\lambda) = \{\lambda\}, L(a) = \{a\}$ for $a \in A, L(\alpha + \beta) = L(\alpha) \cup L(\beta), L(\alpha \cdot \beta) = L(\alpha) \cdot L(\beta)$ and $L(\alpha^*) = L(\alpha)^*$. The alphabet of a regular expression $\alpha$ will be denoted with $A_\alpha$. We define $\Lambda(\alpha) = \{\lambda\} \cap L(\alpha)$. Derivatives and left quotients are denoted in the same way. This should not lead to confusion as $L(u^{-1}\alpha) = u^{-1}L(\alpha)$.

The linearized expression of a regular expression $\alpha$, denoted by $\overline{\alpha}$, is obtained by marking each letter with a subindex denoting its position in $\alpha$. Thus, if the set of positions of $\alpha$ is $pos(\alpha) = \{1, 2, \ldots, \|\alpha\|\}$ and $pos_0(\alpha) = pos(\alpha) \cup \{0\}$, then $A_{\overline{\alpha}}$ is the alphabet of symbols $a_i$ such that there is an $a$ in position $i$ of the regular expression $\alpha$.

A *finite automaton* (*NFA*) is a 5-tuple $\mathcal{A} = (Q, A, \delta, q_0, F)$, where $Q$ is a finite set of states, $A$ is an alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta : Q \times (A \cup \{\lambda\}) \to 2^Q$ is the transition function, which will also be seen as $\delta \subseteq Q \times (A \cup \{\lambda\}) \times Q$. Given an automaton $\mathcal{A}$ and a state $q \in Q$, we denote the right language of $q$ in $\mathcal{A}$ as $R_q^{\mathcal{A}} = \{x \in A^* \ : \ \delta(q, x) \cap F = \emptyset\}$.

If an automaton has no empty transitions and for every state $q$ and every symbol $a$, the number of transitions $\delta(q, a)$ is at most one, it is called *deterministic* (*DFA*).

Given a NFA $\mathcal{A}$ and any two states $p\ q$, the equivalence relation $\equiv_R$ defined as $p \equiv_R q$ if and only if $R_p^{\mathcal{A}} = R_q^{\mathcal{A}}$. This relation defines a partial reduction of $\mathcal{A}$. When a DFA is considered, this equivalence relation produces the minimal DFA.

Given two equivalence relations $E_1$ and $E_2$, the *join* relation, denoted with $E_1 \vee E_2$, is defined as the smallest equivalence relation that contains $E_1$ and $E_2$, that is, $E_1 \vee E_2$ is the transitive closure of the relation $E_1 \cup E_2$. Finally, we say that $E_1$ refines $E_2$ (denoted $E_1 \preceq E_2$) when $p \equiv_{E_1} q$ implies that $p \equiv_{E_2} q$.

## 3. Position, follow and partial derivatives automaton

In this section we summarize the most relevant previous results on this matter. We also recall a previous method by Champarnaud and Ziadi [5] that runs with quadratic time complexity.

The position automaton of a regular expression $\alpha$, which we will denote $\mathcal{A}_{pos}(\alpha)$, was introduced independently by Glushkov [9] and McNaughton and Yamada [10]. This construction, for a given a regular expression $\alpha$, for $u, w \in A_{\overline{\alpha}}^*$ and $i \in pos(\alpha)$, uses the following mappings:

- $first(\alpha) = \{i : a_i w \in L(\overline{\alpha})\}$.
- $last(\alpha) = \{i : w a_i \in L(\overline{\alpha})\}$.
- $follow(\alpha, i) = \{j : u a_i a_j w \in L(\overline{\alpha})\}$.

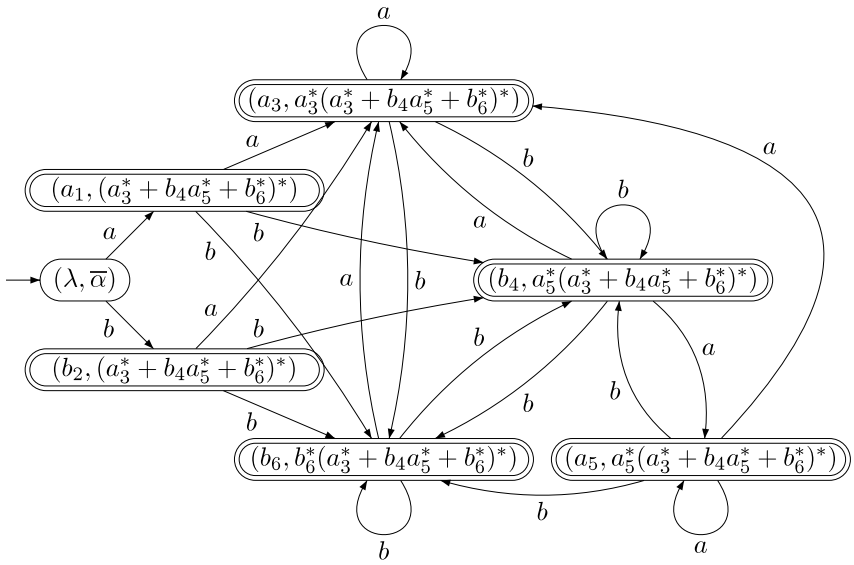


**Fig. 1.** C-continuation automaton for $\alpha = (a + b)(a^* + ba^* + b^*)^*$.

Berstel and Pin [3] related this construction to the concept of local languages. They established that the position automaton could be obtained from a standard local automaton for $\overline{\alpha}$, applying a strictly alphabetical morphism $h : A_{\overline{\alpha}}^* \to A_\alpha^*$ that erases the subindexes in $\overline{\alpha}$. Clearly $h(\overline{\alpha}) = \alpha$.

Although they were not concerned on the development of efficient algorithms, Berstel and Pin [3], and also Berry and Sethi [2] in an implicit way, have proved that $\overline{\alpha}$ defines a local language for any regular expression $\alpha$.

The partial derivatives automaton of an expression $\alpha$, denoted $\mathcal{A}_{pd}(\alpha)$ in the following, was introduced by Antimirov [1]. Champarnaud and Ziadi propose in [5] an efficient method to build the position and the partial derivatives automaton. Their method is based in the notion of *c-continuations* of a linear regular expression. Intuitively, for any linear regular expression $\overline{\alpha}$, the computation of $c_a(\overline{\alpha})$ traverses the regular expression, and searches for a non-empty derivative with respect to any word whose last symbol is $a$. Note that, given any linear regular expression $\overline{\alpha}$, the c-continuation of $\overline{\alpha}$ with respect to $a$ returns an expression of all the (non-null) derivatives with respect to $(ua)$, no matter which $u \in A_{\overline{\alpha}}^*$ is considered.

The c-continuations allows the definition of the *c-continuation automaton* $\mathcal{A}_c(\alpha) = (Q, A_\alpha, \delta, q_0, F)$, where:

- $Q = \{(a_i, c_{a_i}(\overline{\alpha}))$, where $a_i$ is the symbol in the $i$-th position in $\overline{\alpha}$ or $\lambda$ and $c_{a_i}(\overline{\alpha})$ is the c-continuation of $\overline{\alpha}$ with respect to the symbol whose position is $i$.
- $q_0 = (\lambda, \overline{\alpha})$
- $F = \{(a_i, c_{a_i}(\overline{\alpha})) \ : \ \Lambda(c_{a_i}(\overline{\alpha})) \neq \emptyset\}$
- $\delta((a_i, c_{a_i}(\overline{\alpha})), b) = \{(b_j, c_{b_j}(\overline{\alpha})) \ : \ h(b_j) = b \ \wedge \ j \in follow(\alpha, i)\}$.

The c-continuation automaton by Champarnaud and Ziadi provides an efficient way to obtain the partial derivatives automaton. Briefly, given a regular expression $\alpha$, the method considers equivalent those states $p = (a_i, c_{a_i}(\overline{\alpha}))$ and $q = (b_j, c_{b_j}(\overline{\alpha}))$ (i.e. $p \equiv_{pd} q$) such that $h(c_{a_i}(\overline{\alpha})) = h(c_{b_j}(\overline{\alpha}))$, with $h : A_{\overline{\alpha}}^* \to A_\alpha^*$ where $h(a_i) = a$. In other words, those states whose c-continuation are identical when the subindexes are erased.

The authors propose a quadratic algorithm to obtain the c-continuation automaton. This algorithm allows them to obtain (also with quadratic time complexity) both the position and the partial derivatives automata for a given regular expression.

An example of the c-continuation automaton for a regular expression $\alpha$ is shown in Fig. 1. The depicted automaton is the position automaton for $\alpha$. Example 1 illustrates the process to obtain the partial derivatives automaton.

**Example 1.** Let $\alpha = (a + b)(a^* + ba^* + b^*)^*$. The linearized expression is $\overline{\alpha} = (a_1 + b_2)(a_3^* + b_4a_5^* + b_6^*)^*$. The partial derivatives automaton can be obtained from the c-continuation automaton shown in Fig. 1.

Note that the states $(a_1, (a_3^* + b_4a_5^* + b_6^*)^*)$ and $(b_2, (a_3^* + b_4a_5^* + b_6^*)^*)$ are merged. The resulting state is identified with the state $q_1$ in Fig. 2. In the same way, the states $(a_3, a_3^*(a_3^* + b_4a_5^* + b_6^*)^*)$, $(b_4, a_5^*(a_3^* + b_4a_5^* + b_6^*)^*)$ and $(a_5, a_5^*(a_3^* + b_4a_5^* + b_6^*)^*)$ are also merged (state $q_2$ in Fig. 2). The state $(b_6, b_6^*(a_3^* + b_4a_5^* + b_6^*)^*)$ of the c-continuation automaton is not merged with anyone and is denoted with $q_3$ in the partial derivatives automaton.

**Proposition 2** (*Champarnaud and Ziadi [6]*). $\mathcal{A}_{pos}(\alpha)/\equiv_{pd} = L(\alpha)$.

**Proof.** The proof is based in the fact that $\equiv_{pd} \preceq \equiv_R$. □

In [11], Ilie and Yu propose a new algorithm to construct NFAs from regular expressions named *follow automaton*. The authors propose a constructive algorithm, and also prove that the follow automaton is a quotient of the position automaton

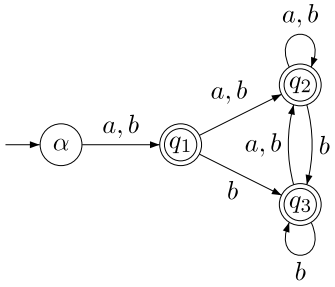


**Fig. 2.** $\mathcal{A}_{pd}(\alpha)$ for $\alpha = (a+b)(a^* + ba^* + b^*)^*$.

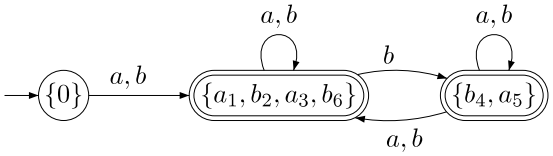


**Fig. 3.** Follow automaton for $\alpha = (a+b)(a^* + ba^* + b^*)^*$. The equivalence classes are inside the states.

by the equivalence relation $\equiv_f$ defined as:

$$a_i \equiv_f a_j \Leftrightarrow \begin{cases} a_i \in last(\alpha) \leftrightarrow a_j \in last(\alpha) & \text{and} \\ follow(\alpha, i) = follow(\alpha, j). \end{cases}$$

That is, given a NFA $A = (Q, \Sigma, \delta, q_0, F)$, for any pair of states $p$ and $q$, $p \equiv_f q$ if and only if $\forall a \in \Sigma, \delta(p,a) = \delta(q,a) \wedge (p \in F \Leftrightarrow q \in F)$. Thus, it is easy to see that $\equiv_f \preceq \equiv_R$. Proposition 3 follows from this fact.

**Proposition 3** (*Ilie and Yu [11]*). $L(\mathcal{A}_{pos}(\alpha)/\equiv_f) = L(\alpha)$.

We note here that the quotient automaton $\mathcal{A}_{pos}(\alpha)/\equiv_f$ (i.e. the follow automaton) can also be computed efficiently using the c-continuation automaton.

**Example 4.** Let us consider the position automaton shown in Fig. 1. We identify three equivalence classes of the follow relation: $\{\lambda\}$, $\{a_1, b_2, a_3, b_6\}$ and $\{b_4, a_5\}$ (for the sake of brevity, we do not show the second component of the states). The quotient of the position automaton by the relation $\equiv_f$ is depicted in Fig. 3.

## 4. A new method to obtain λ-free NFAs from regular expressions

In this section we will describe a new method that obtains finite automata from regular expressions. It uses the concepts of follow [11] and equation automata [6]. The size of these automata for a given regular expression are upper bounds of the size of the automaton obtained by the method we propose below.

Let us define $\equiv_\vee$ as the join of the relations $\equiv_{pd}$ and $\equiv_f$.

**Proposition 5.** $L(\mathcal{A}_{pos}(\alpha)/\equiv_\vee) = L(\alpha)$.

**Proof.** To prove the proposition it will be enough to prove that $\equiv_\vee \preceq \equiv_R$.

Given any pair of states $p$ and $q$ of $\mathcal{A}_{pos}(\alpha)$, if $p \equiv_\vee q$, then two cases arise:

1. if $p \equiv_f q$ or $p \equiv_{pd} q$, then $p \equiv_R q$,
2. if $p \not\equiv_f q$ and $p \not\equiv_{pd} q$, then there exists $r$ such that $p \equiv_f r$ and $r \equiv_{pd} q$. Due to the fact that, $\equiv_f \preceq \equiv_R$ and $\equiv_{pd} \preceq \equiv_R$, it follows that $R_r^{\mathcal{A}_{pos}} = R_p^{\mathcal{A}_{pos}}$ and $R_r^{\mathcal{A}_{pos}} = R_q^{\mathcal{A}_{pos}}$, therefore, $R_p^{\mathcal{A}_{pos}} = R_p^{\mathcal{A}_{pos}}$ and $p \equiv_R q$. □

**Proposition 6.** *The size of the automaton* $\mathcal{A}_{pos}/\equiv_R$ *is bounded above by the size of* $\mathcal{A}_{pos}/\equiv_f$ *and* $\mathcal{A}_{pos}/\equiv_{pd}$.

**Proof.** Note that it follows from the fact that $\equiv_\vee$ is coarser than both $\equiv_{pd}$ and $\equiv_f$. □

Algorithm 4.1 shows how the new automaton can be obtained. This algorithm first merges the states of the c-continuation automaton that are equivalent under the follow relation. In this merging step the algorithm does not discard the different c-continuations of the merged states. This provides, for each state $q$ in the resulting automaton , several expressions for the same language. The second step uses the morphism that erases the subindexes of the expression in each state. Those states that have in common a regular expression (i.e. their right languages are the same) are also merged.

**Example 7.** Let us consider the regular expression $(a+b)(a^* + ba^* + b^*)^*$. Fig. 1 shows the c-continuation automaton for $\alpha$. The quotient of this automaton by the relation $\equiv_f$ is shown in Fig. 4.

Note that $h(a_3^*(a_3^* + b_4a_5^* + b_6^*)^*) = h(a_5^*(a_3^* + b_4a_5^* + b_6^*)^*)$. Thus, the resulting automaton is shown in Fig. 5.

**Algorithm 4.1** Algorithm to obtain small NFA for any given regular expression.

**Input**: A regular expression $\alpha$.
**Output**: A non-deterministic automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\alpha)$ .

**Method**:
Obtain the c-continuation automaton $\mathcal{A}_c(\alpha)$
Compute the relation $\equiv_f$
Obtain $\mathcal{A}_c(\alpha)/\equiv_f$.
(* The c-continuations of the merged states are not discarded *)
Erase the subindexes to each c-continuation
Merge those states which have at least one expression in common
**Return** the resulting automaton
**EndMethod**:

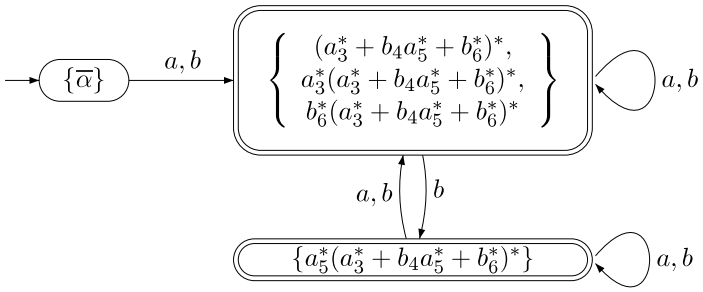


**Fig. 4.** Quotient automata $\mathcal{A}_c(\alpha)/\equiv_f$, for the expression $(a + b)(a^* + ba^* + b^*)^*$. First step performed by our algorithm.

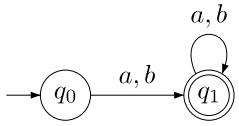


**Fig. 5.** Automaton for $\alpha = (a + b)(a^* + ba^* + b^*)^*$ with our method.

Our algorithm can take profit from the result in [5] to achieve also quadratic complexity with respect to the size of the regular expression. Note that computing the relation $\equiv_f$ as well as obtaining the quotient automaton does not increase the quadratic complexity. The c-continuations are not discarded in the first step, therefore the quotient automaton has, at most, the same number of c-continuations than the c-continuation automaton, and thus it is possible to check which states to merge without increasing the complexity.

### *4.1. Relations between approaches*

As mentioned, for any given regular expression, the follow and partial derivatives methods are not comparable with respect to the size of the output automata. In [11] Ilie and Yu state the difficulty of such a study and propose an empirical study using real-life applications.

In [7] the authors tackle this comparative task, and they prove that, whenever the regular expression is *normalized*, then the partial derivatives automaton is a quotient of the follow automaton.

For any given regular expression $\alpha$, it is said that $\alpha$ is a normalized expression if it is *reduced* and it is in *Star Normal Form* (SNF) [4]. A expression is reduced if it contains neither $\emptyset$ nor unnecessary $\lambda$ and it has no nested star operators. For any expression, it is possible to use the syntactic tree of it to obtain a reduced version in linear time.

We refer the interested reader to [4] for more details on the algorithm to obtain the SNF version of any given regular expression. We only note that: first, this computation can be done in linear time; and second, a regular expression $\alpha$ is said to be in SNF if and only if, for every subexpression $\beta^*$ of $\alpha$ the next condition holds:

$$\forall i \in last(\beta),\ follow(\beta, i) \cap first(\beta) = \emptyset.$$

**Proposition 8** (*Champarnaud et al. [7]*)**.** *When normalized expressions are considered,* $\equiv_f \preceq \equiv_{pd}$.

**Proposition 9.** *When normalized expressions are considered. The size of* $\mathcal{A}_{pos}/\equiv_\vee$ *is equal to the size of* $\mathcal{A}_{pd}$.

**Proof.** By Proposition 8 and the definition of $\equiv_\vee$. □

When normalized expressions are considered, our algorithm returns the same automaton output by the partial derivatives method. Note that this is easy to prove because our approach applies the follow and partial derivatives equivalence relations.

It is worth to be noted that, when a normalized regular expression is considered, there is no difference between the output obtained by ours and the partial derivatives methods. We also note that, normalized expressions does not offer always an advantage. The following example illustrates this fact.

**Example 10.** Let us consider the regular expression of Example 1. The normalized version of this expression is $\alpha_n = (a + b)(a + ba^* + b)^*$. Both the follow and partial derivatives methods output the automata shown in Fig. 4 when they run with the normalized expression as input. We recall that the partial derivatives method output a four-states automaton when the non-normalized version of $\alpha$ is used.

Algorithm 4.1 returns the same automaton when the normalized version of $\alpha$ is used. Nevertheless, as shown in Example 7, it is possible to obtain a smaller automaton using the original expression.

## 5. Conclusions

Although the time complexity of both the follow automaton and the equation automaton (partial derivatives) method is the same, taking into account general expressions, the size of the automata they obtain cannot be compared. When normalized expressions are considered, the size of the automata output by the partial derivatives method is upper bounded by the size of the follows automaton.

In this paper we propose a new method to construct automata from regular expressions. The algorithm runs also with quadratic time complexity and assures that the size of the automata obtained is upper bounded by the size of the smallest automata obtained by the previous methods.

## References

[1] V. Antimirov, Partial derivatives of regular expressions and finite automata constructions, Theoretical Computer Science 155 (1996) 291–319.
[2] G. Berry, R. Sethi, From regular expressions to deterministic automata, Theoretical Computer Science 48 (1) (1986) 117–126.
[3] J. Berstel, J.-E. Pin, Local languages and the Berry–Sethi algorithm, Theoretical Computer Science 155 (2) (1996) 439–446.
[4] A. Brüggemann-Klein, Regular expressions into finite automata, Theoretical Computer Science 120 (1993) 117–126.
[5] J.M. Champarnaud, D. Ziadi, From c-continuations to new quadratic algorithms for automaton synthesis, International Journal of Algebra and Computation 6 (2001) 707–736.
[6] J.M. Champarnaud, D. Ziadi, Canonical derivatives, partial derivatives and finite automaton constructions, Theoretical Computer Science 289 (2002) 137–163.
[7] J.M. Champarnaud, F. Ouardi, D. Ziadi, Normalized expressions and finite automata, International Journal of Algebra and Computation 17 (1) (2007) 141–154.
[8] C.H. Chang, R. Paige, From regular expressions to DFA's using compressed NFA's, Theoretical Computer Science 178 (1997) 1–36.
[9] V.M. Glushkov, The abstract theory of automaton, Russian Mathematical Surveys 16 (1961) 1–53.
[10] R. McNaughton, H. Yamada, Regular expressions and state graphs for automata, IEEE Transactions on Electronic Computers 9 (1) (1960) 39–47.
[11] L. Ilie, S. Yu, Follow automata, Information and Computation 186 (2003) 140–162.
[12] K. Thompson, Regular expression search algorithm, Communications of the ACM 11 (6) (1968) 419–422.
[13] D. Ziadi, J-L. Ponty, J.M. Champarnaud, Passage d'une expression rationnelle à un automate fini non-déterministe, Bulletin of the Belgian Mathematical Society (1997) 177–203.