A new algorithm to obtain small NFAs from regular expressions

Pedro García (1), Damián López (1), José Ruiz (1) and Gloria I. Álvarez (2)

(1) Departamento de Sistemas Informáticos y Computación.
Universidad Politécnica de Valencia. Valencia (Spain).
(2) Pontificia Universidad Javeriana. Cali (Colombia).
email: \{pgarcia,jruiz,dlopez,galvarez\}@dsic.upv.es

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 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. As a byproduct we describe a new method to obtain the position automaton based in the proof of Berstel and Pin [4] that linearized regular expressions represent local languages.

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 as Perl, Phyton, Java, C♯ or PHP consider regular expressions as an extra tool that helps to decrease the programming effort.

*Work partially supported by Spanish Ministerio de Educación y Ciencia under project TIN2007-60769
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 the sequel, $\|\alpha\|$ will denote just the number of symbols.

The position automaton was proposed independently by Glushkov [8] and McNugton and Yamada [11]. An intuitive algorithm to construct it starts considering the linearized version $\pi$ 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 $L(\pi)$ and the symbol in the position $j$ of $\alpha$ is $b$. Several methods have been proposed to obtain this automaton with quadratic time complexity: Champarnaud and Ziadi [5]; Chang and Paige [7]; and Brügge-Klein [2].

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 $O(|\alpha|^2 \|\alpha\|^3)$ algorithm to 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 $O(|\alpha|^2 \|\alpha\|)$ 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 $\pi$ obtains the same automaton as the partial derivatives method applied to $\alpha$. Champarnaud and Ziadi propose in [5] an improved algorithm that runs in $O(|\alpha|^2)$ space and time complexity.

The follow automaton, proposed by Ilie and Yu [10] 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 $\lambda$-loops. This allows the authors to develop an algorithm to eliminate the $\lambda$-transitions that works in $O(|\alpha|^2)$.

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. Nevertheless, the size of the automata obtained from these methods can not be
compared.

In this paper we propose a new method to construct a $\lambda$-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. As a byproduct, we also describe a new method to construct the position automaton. It is based in the proof of Berstel and Pin that linearized regular expressions are local languages.

## 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. The only way of obtaining a regular expression is to apply 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 $L(\alpha^*) = L(\alpha)^*$. When necessary, the alphabet of a regular expression $\alpha$ will be denoted with $A_\alpha$. We define $\Lambda(\alpha) = \{\lambda\} \cap L(\alpha)$.

The derivative of a regular expression with respect to a symbol is defined recursively as follows:

- $a^{-1}\emptyset = \emptyset$
- $a^{-1}\lambda = \emptyset$
- $a^{-1}b = \begin{cases} \lambda & \text{if } b = a \\ \emptyset & \text{otherwise} \end{cases}$
- $a^{-1}(\alpha + \beta) = a^{-1}\alpha + a^{-1}\beta$
- $a^{-1}(\alpha \cdot \beta) = \begin{cases} a^{-1}(\alpha)\beta & \text{if } \Lambda(\alpha) = \emptyset \\ a^{-1}(\alpha)\beta + a^{-1}\beta & \text{otherwise} \end{cases}$
- $a^{-1}(\alpha)^* = a^{-1}(\alpha)\alpha^*$

Derivatives of regular expressions can be extended to a words as follows:

- $\lambda^{-1}\alpha = \alpha$
- $(ua)^{-1}\alpha = a^{-1}(u^{-1}\alpha)$, $u \in A^*$, $a \in A$. 3
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 $\alpha$ is denoted $\overline{\alpha}$. It is obtained marking each letter with a subindex denoting its position in $\alpha$. Thus, if the set of positions of $\alpha$ is $\text{pos}(\alpha) = \{1, 2, \cdots, |\alpha|\}$ and $\text{pos}_0(\alpha) = \text{pos}(\alpha) \cup \{0\}$, then $A_{\overline{\alpha}}$ is the alphabet of symbols $a_i$ such that $i$ is the position of $a$ in the regular expression $\alpha$.

A finite automaton (NFA) is a 5-tuple $A = (Q, A, \delta, q_0, F)$, 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 $\delta : Q \times (A \cup \{\lambda\}) \rightarrow 2^Q$ is the transition function, which will also be seen as $\delta \subseteq Q \times (A \cup \{\lambda\}) \times Q$. Given an automaton $A$ and a state $q \in Q$, we denote $R^A_q$, the right language of $q$ in $A$, that is, the language accepted by the automaton $A = (Q, A, \delta, q, F)$.

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).

A language $L$ is local if there exist sets $P, S \subseteq A$ and $N \subseteq A^2$ such that $L - \{\lambda\} = (PA^* \cap A^*S) - A^*NA^*$. The family of local languages is a subclass of the regular languages. Every regular language is the image of a local language under a morphism.

Another important result [9] states that a language $L$ is local if and only if, for any given words $u, v$ such that $u \neq v$ and any symbol $a$, $(ua)^{-1}L = \emptyset$ and $(va)^{-1}L = \emptyset$ implies that $(ua)^{-1}L = (va)^{-1}L$.

Linearized expressions denote local languages and therefore, the derivatives of linearized expressions with respect to the words ending with the same symbol are either empty or represent the same language.

A DFA $A = (Q, A, \delta, q_0, F)$ is local if for any $a \in A$, the set $\{\delta(q, a) : q \in Q\}$ contains at most one element. If there is no transition arriving to the initial state, the automaton is called standard local.

The following standard local automaton $A = (Q, A, \delta, q_0, F)$ recognizes a local language $L$ defined by the sets $P, S \subseteq A$ and $N \subseteq A^2$: $Q = A \cup \{\lambda\}$, $q_0 = \lambda$, $F = S$ and $\delta(\lambda, a) = a$ for $a \in P$ and $\delta(a, b) = b$ for $ab \notin N$.

3 Position, partial derivatives and follow automata

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. This result will allow us to propose our improved algorithm.

3.1 Position automaton

The position automaton was introduced independently by Glushkov [8] and McNaughton-Yamada [11]. This construction, for a given a regular expres-
sion \( \alpha \), for \( u, w \in A^*_\alpha \) and \( i \in \text{pos}(\alpha) \), uses the following mappings:

- \( \text{first}(\alpha) = \{ i : a_iw \in L(\overline{\alpha}) \} \).
- \( \text{last}(\alpha) = \{ i : wa_i \in L(\overline{\alpha}) \} \).
- \( \text{follow}(\alpha, i) = \{ j : ua_ia_jw \in L(\overline{\alpha}) \} \).

Berstel and Pin [4] related this construction to the concept of local languages. They established that the position automaton could be obtained from a standard local automaton for \( \alpha \), applying a strictly alphabetical morphism \( h : A^*_\alpha \rightarrow A^*_\alpha \) 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 [4], and also Berry and Sethi [3] in an implicit way, have proved that \( \alpha \) defines a local language for any regular expression \( \alpha \). This can be seen doing \( P = \{ a_i : i \in \text{first}(\alpha) \} \), \( S = \{ a_i : i \in \text{last}(\alpha) \} \) and \( N = A^*_\alpha - \{ a_ia_j : j \in \text{follow}(\alpha, i) \} \) and applying the construction for a standard local automaton explained above. In the sequel, we will denote this automaton as \( A_{sl}(\alpha) \). Applying the morphism that erases the subindexes of the symbols to this automaton, the position automaton for \( \alpha \), which we will denote as \( A_{pos}(\alpha) \), is obtained.

**Example 1** Let \( \alpha = (a + b)(a^* + ba^* + b^*)^* \). The linearized expression is \( \overline{\alpha} = (a_1 + b_2)(a_3^* + b_4a_5^* + b_6^*)^* \). We have then \( \text{first}(\alpha) = \{ 1, 2 \} \), \( \text{last}(\alpha) = \{ 1, 2, 3, 4, 5, 6 \} \), \( \text{follow}(\alpha, 1) = \text{follow}(\alpha, 2) = \text{follow}(\alpha, 3) = \text{follow}(\alpha, 6) = \{ 3, 4, 6 \} \), whereas \( \text{follow}(\alpha, 4) = \text{follow}(\alpha, 5) = \{ 3, 4, 5, 6 \} \). The standard local automaton \( A_{sl}(\overline{\alpha}) \) is shown in Figure 1. The position automaton \( A_{pos}(\alpha) \), which is obtained eliminating the subindexes, is depicted in Figure 2.

![Figure 1: \( A_{sl}(\overline{\alpha}) \) for \( \overline{\alpha} = (a_1 + b_2)(a_3^* + b_4a_5^* + b_6^*)^* \).](image-url)
3.2 An alternative algorithm to obtain the position automaton

We recall the construction that Berstel and Pin [4] used to proof that linear regular expressions denote local languages.

The automata constructions for the basic cases are the standard. Let $\alpha$ and $\beta$ be two linear regular expressions. Let also $A_{sl}(\alpha) = (Q_1, A_1, \delta_1, q_1, F_1)$ and $A_{sl}(\beta) = (Q_2, A_2, \delta_2, q_2, F_2)$, with $A_1 \cap A_2 = \emptyset$, be standard local automata for these expressions. The constructions for the regular expressions involving operators can be seen in Figure 3. Precise definitions are given below:

1. $L(\alpha) \cup L(\beta)$ is recognized by the local automaton $A = (Q, A, \delta, q_0, F)$, where the new initial state $q_0$ is obtained merging $q_1$ and $q_2$, that is,
   \begin{itemize}
   \item $Q = (Q_1 - \{q_1\}) \cup (Q_2 - \{q_2\}) \cup \{q_0\}$, $q_0 \notin Q_1 \cup Q_2$.
   \item $\delta = \{(q, a, q') \in \delta_1 \cup \delta_2 : q \notin \{q_1, q_2\}\} \cup \{(q_0, a, q) : (q_1, a, q) \in \delta_1 \lor (q_2, a, q) \in \delta_2\},$
   \end{itemize}
\[ F = \begin{cases} \quad F_1 \cup F_2 & \text{if } q_1 \notin F_1 \land q_2 \notin F_2 \\ \quad (F_1 - \{q_1\}) \cup (F_2 - \{q_2\}) \cup \{q_0\} & \text{otherwise}. \end{cases} \]

2. \( L(\alpha) \cdot L(\beta) \) is recognized by the local automaton \( A = (Q, A, \delta, q_0, F) \), where:

- \( Q = (Q_1 \cup Q_2) - \{q_2\} \),
- \( \delta = \delta_1 \cup \{(q, a, q') \in \delta_2 : q \neq q_2\} \cup \{(q, a, q') : q \in F_1 \land (q_2, a, q') \in \delta_2\} \),
- \( q_0 = q_1 \)
- \( F = \begin{cases} \quad F_2 & \text{if } q_2 \notin F_2 \\ \quad F_1 \cup (F_2 - \{q_2\}) & \text{otherwise}. \end{cases} \)

3. If \( A = (Q, A, \delta, q_0, F) \) recognizes \( L(\alpha_1) \), then \( L(\alpha_1)^* \) is recognized by the local automaton \( A' = (Q, A, \delta', q_0, F \cup \{q_0\}) \), where \( \delta' = \delta \cup \{(q, a, q') : q \in F \land (q_0, a, q') \in \delta\} \).

Note that to obtain this standard local automaton it is enough to follow the syntactic tree of the regular expression in the usual operator precedence order (which can be done in \( \mathcal{O}(|\alpha|) \)) and apply the given rules. The most expensive rule is the star. The application of this rule would lead to a \( \mathcal{O}(||\alpha||^2) \) time complexity in the worst case. Therefore, we conclude that it is possible to obtain the standard local automaton in \( \mathcal{O}(||\alpha||^2|\alpha|) \).

This algorithm has more time complexity than the best algorithms that obtains such automaton (for instance, Champarnaud and Ziadi [5]). Nevertheless, we consider the construction interesting enough since, first, the method takes profit from the fact that linearized regular expressions denote local languages; and second, the method works recursively on the structure of the regular expression, using in each step very simple construction rules.

**Example 2** Let \( \alpha = (a + b)(a^* + ba^* + b^*)^* \). Then \( \overline{\alpha} = (a_1 + b_2)(a_5^* + b_4a_5^* + b_6^*)^* \). Some of the elementary automata are depicted in Figure 4.

![Figure 4: Automata for a1, b2, a1 + b2 and a5*](image)

The automaton for \((a_5^* + b_4a_5^* + b_6^*)^* \) is shown in Figure 5. The order of construction of the automaton from the elementary items is standard: parenthesis, star, concatenation and addition.

To construct the position automaton for \( \alpha \) one needs to concatenate the automaton for \((a_1 + b_2)\) with the automaton for \((a_5^* + b_4a_5^* + b_6^*)^* \). It is the automaton depicted in Fig 1.
3.3 Partial Derivatives automaton

The partial derivatives automaton was introduced by Antimirov [1]. For \( a \in A \) and a regular expression \( \alpha \), the set of partial derivatives of \( \alpha \) with respect to \( a \) is denoted \( \partial_a(\alpha) \) and defined recursively as:

- \( \partial_a(\emptyset) = \partial_a(\lambda) = \emptyset \),
- \( \partial_a(b) = \begin{cases} \{ \lambda \} & \text{if } a = b, \\ \emptyset & \text{otherwise}. \end{cases} \)
- \( \partial_a(\alpha + \beta) = \partial_a(\alpha) \cup \partial_a(\beta) \)
- \( \partial_a(\alpha \beta) = \partial_a(\alpha) \beta \cup \partial_a(\beta) \) if \( \Lambda(\alpha) = \emptyset \),
- \( \partial_a(\alpha^*) = \partial_a(\alpha)\alpha^* \)

This definition is extended to words \( w \in A^* \) as follows:

- \( \partial_w(\alpha) = \begin{cases} \alpha & \text{if } w = \lambda \\ \partial_a(\partial_w(\alpha)) & \text{if } w = xa. \end{cases} \)

We call the \( PD(\alpha) \) to the (finite) set of partial derivatives of \( \alpha \), that is, \( PD(\alpha) = \{ \partial_w(\alpha) : w \in A^* \} \). The partial derivatives automaton was defined as: \( A_{pd}(\alpha) = (PD(\alpha), A, \delta_{pd}, \alpha, \{ q \in PD(\alpha) : \Lambda(q) = \{ \lambda \} \}) \), where \( \delta_{pd}(q, a) = \partial_a(q) \).

**Example 3** Let us consider the regular expression of Example 1. The partial derivatives of \( \alpha \) are: \( \partial_a(\alpha) = \partial_b(\alpha) = (a^* + ba^* + b^*)^* = \{ \alpha_1 \} \), \( \partial_{a_1}(a_1) = a^*a_1 \), \( \partial_b(a_1) = (a^*a_1, b^*a_1) \) and so on. The partial derivatives automaton for \( \alpha \) is depicted in Figure 6.

3.4 C-continuation automaton

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, which are defined as follows:
The authors propose a quadratic algorithm to obtain the c-continuation automaton. The algorithm achieves this complexity using a new identifier for each star (sub)expression with a new identifier. Thus, at a cost of increase the alphabet with as many symbols as the number of stars in the regular expression, the length of each c-continuation is bounded by the length of the regular expression. Further computations do not increase this complexity.
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 Figure 7. Note that it is isomorphic to the position automaton for $\alpha$.

![Diagram of the c-continuation automaton for $\alpha = (a + b)(a^* + ba^* + b^*)^*$](image)

**Figure 7:** C-continuation automaton for $\alpha = (a + b)(a^* + ba^* + b^*)^*$.

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 $(a_i, c_{a_i}(\alpha))$ and $(b_j, c_{b_j}(\alpha))$ such that $c_{a_i}(\alpha)$ and $c_{b_j}(\alpha)$ are identical when the subindexes are erased.

**Example 4** Let us consider the regular expression $\alpha = (a + b)(a^* + ba^* + b^*)^*$. The partial derivatives automaton can be obtained from the c-continuation automaton shown in Figure 7.

Note that the states $(a_1, (a_2^* + b_4a_5^* + b_6^*))$ and $(b_2, (a_2^* + b_4a_5^* + b_6^*))$ are merged. The resulting state is identified with the state $q_1$ in Figure 6. 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 Figure 6). The state $(6, b_6^*(a_3^* + b_4a_5^* + b_6^*))$ of the c-continuation automaton is not merged with anyone and is denoted with $q_3$ in the partial derivatives automaton.
3.5 Follow automaton

In [10], Ilie and Yu propose a new algorithm to construct NFAs from regular expressions. They start building a NFA with \(\lambda\)-transitions which prove of size at most \(O(|\alpha|)\). This automaton is denoted \(A_\lambda(\alpha)\). The algorithm obtains a \(\lambda\)-automaton with no \(\lambda\)-loops. This allows the authors to use an ad-hoc \(\lambda\)-elimination method of time and space complexity of \(O(|\alpha|^2)\) to build the follow automaton for \(L(\alpha)\), which is denoted \(A_f(\alpha)\).

**Example 5** Let us consider again the regular expression \(\alpha\) of Example 1. The NFA with \(\lambda\)-transitions \(A_\lambda(\alpha)\) and the follow automaton \(A_f(\alpha)\) are depicted in Figure 8 (a) and (b) respectively. See [10] for the details of the construction.

![Diagram](image)

Figure 8: (a) \(A_\lambda(\alpha)\) and (b) \(A_f(\alpha)\) for \(\alpha = (a + b)(a^* + ba^* + b^*)^*\).

They also prove that the follow automaton is a quotient of the position automaton by the equivalence relation \(\equiv_f\) defined as:

\[ a_i \equiv_f a_j \text{ if and only if } (a_i, a_j \in F \text{ or } a_i, a_j \in Q \setminus F) \text{ and } \text{follow}(\alpha, i) = \text{follow}(\alpha, j). \]

3.6 Relations between the follow and the standard local automata.

Let us see now that the equivalence relation that defines the follow automaton is equivalent to minimize the standard local automaton for \(\alpha\) and then, to apply the homomorphism that eliminates the subindexes of the symbols.

**Proposition 6** Let \(\mathcal{A} = (Q, \delta, q_0, F)\) be a DFA such that \(L(\mathcal{A}) = L(\alpha)\). Let \(q, q' \in Q\). Then \(R^\mathcal{A}_q\) is a local language for every \(q \in Q\).

**Proof.** Let \(q \in Q\) and let \(x \in \Sigma^*\) such that \(\delta(q_0, x) = q\). It follows that \(R^\mathcal{A}_q = x^{-1}L\). Let \(a \in \Sigma\), for every \(y \in \Sigma^*\) we have that \((ya)^{-1}R^\mathcal{A}_q = (ya)^{-1}(x^{-1}L) = (xya)^{-1}L\). Note that this language is empty or the same for every \(y \in \Sigma^*\). Therefore, \(R^\mathcal{A}_q\) is a local language. ■

**Proposition 7** Let \(\overline{\alpha}\) be a linearized regular expression and let the automaton \(\mathcal{A} = (Q, \delta, q_0, F)\) be a DFA such that \(L(\mathcal{A}) = L(\overline{\alpha})\). Let \(q, q' \in Q\). Then \(R^\mathcal{A}_q = R^\mathcal{A}_{q'}\) if and only if \(\text{follow}(\overline{\alpha}, q) = \text{follow}(\overline{\alpha}, q')\) and \(q \in F\) if and only if \(q' \in F\).
Proof. First we note that \( L(\mathcal{A}) \) is a local language because the automaton \( \mathcal{A} \) accepts the language denoted by a linearized regular expression.

The direct part is evident. For the reciprocal, let \( R^A_q = R^A_{q'} = L' \). As \( R^A_q = R^A_{q'} \), \( \lambda \in R^A_q \) if and only if \( \lambda \in R^A_{q'} \) and \( q \subseteq F \) if and only if \( q' \subseteq F \). \( L' \) is a local language and therefore it can be characterized by the sets \( P, S, N \).

The existence of \( P \) implies that \( \text{follow}(\overline{a}, q) = \text{follow}(\overline{a}, q') \). \( \square \)

In a minimization process, two states are merged whenever they have the same right language. These Propositions prove that, if an automaton accepts the language denoted by a linear regular expression \( \overline{\alpha} \) (a local language), then, two states \( p \) and \( q \) of the automaton are equivalent if and only if they have the same right language. Thus, if the states \( p \) and \( q \) of \( A_{sl}(\overline{\alpha}) \) are merged when this automata is minimized, then both have the same right language, and therefore, \( p \equiv_f q \).

Example 8 The equivalence classes of the position automaton of the expression \( \alpha \) of Figure 2 are shown in Table 1.

<table>
<thead>
<tr>
<th>i</th>
<th>state</th>
<th>follow ((\alpha, i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \lambda )</td>
<td>( {1, 2} )</td>
</tr>
<tr>
<td>1</td>
<td>( a_1 )</td>
<td>( {3, 4, 6} )</td>
</tr>
<tr>
<td>2</td>
<td>( b_2 )</td>
<td>( {3, 4, 6} )</td>
</tr>
<tr>
<td>3</td>
<td>( a_3 )</td>
<td>( {3, 4, 6} )</td>
</tr>
<tr>
<td>4</td>
<td>( b_4 )</td>
<td>( {3, 4, 5, 6} )</td>
</tr>
<tr>
<td>5</td>
<td>( a_5 )</td>
<td>( {3, 4, 5, 6} )</td>
</tr>
<tr>
<td>6</td>
<td>( b_6 )</td>
<td>( {3, 4, 6} )</td>
</tr>
</tbody>
</table>

Table 1: Successors of every state of the position automaton for \( \alpha = (a + b)(a^* + ba^* + b^*)^* \)

We identify three equivalence classes \( \{\lambda\}, \{a_1, b_2, a_3, b_6\} \) and \( \{b_4, a_5\} \). The quotient of the position automaton by the relation \( \equiv_f \) is depicted in Figure 9. Note that this quotient is equivalent to minimize the automaton \( A_{sl}(\overline{\alpha}) \) of Figure 1 and then calculate the image under morphism \( h \) (eliminate the subindexes). Note also that the c-continuation automaton have all the information needed to obtain the relation \( \equiv_f \).

Figure 9: Follow automaton for \( \alpha = (a + b)(a^* + ba^* + b^*)^* \). The equivalence classes are inside the states.
4 A new method to obtain \( \lambda \)-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 [10] 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.

**Definition 9** Let \( \alpha \) be a regular expression, let \( \overline{\alpha} \) be the linearized version of \( \alpha \) and let \( h : A_{\overline{\alpha}}^+ \to A_{\alpha}^+ \) be the morphism that converts one expression in the other. For every symbols \( a_i \) and \( a_j \) in \( A_{\overline{\alpha}} \) we define the following equivalence relation: \( a_i \equiv a_j \) if and only if \( a_i \equiv f a_j \) and \( h(a_i) = h(a_j) \).

**Lemma 10** Let \( \psi : A^* \to B^* \) be any strictly alphabetical morphism (i.e., \( \psi(a) = b \in B \) for every \( a \in A \)) and let \( L \subseteq A^* \). Then, we have that \( \psi(\bigcup_{a \in \underbrace{a^{-1}L}}) = b^{-1}\psi(L) \).

**Proof.** Let \( x \in \psi(\bigcup_{a \in \underbrace{a^{-1}L}}) \), then there exists \( y \in \bigcup_{a \in \underbrace{a^{-1}L}} \) such that \( \psi(y) = x \). Thus, there exists \( a \in A \) such that \( \psi(a) = b \) and there exists \( y \in a^{-1}L \) with \( \psi(y) = x \). Then \( b\psi(y) = bx \in \psi(L) \) and thus \( x \in b^{-1}\psi(L) \).

For the converse, let \( x \in b^{-1}\psi(L) \), then \( bx \in \psi(L) \) and there exists \( a \in A \) and \( y \in A^* \) with \( \psi(y) = x \) and \( ay \in L \). Then \( y \in \bigcup_{a \in \underbrace{a^{-1}L}} \) and thus \( x = \psi(y) \in \psi(\bigcup_{a \in \underbrace{a^{-1}L}}) \). \[\blacksquare\]

The previous statement can be easily extended to words by induction, that is, \( \psi(\bigcup_{a \in \underbrace{a^{-1}L}}) = y^{-1}\psi(L) \).

For any regular expression \( \alpha \), and its linearized version \( \overline{\alpha} \), we will denote the quotient set \( A_{\overline{\alpha}}/ \equiv \equiv A_{\overline{\alpha}} \). Let \( \varphi : A_{\overline{\alpha}}^+ \to A_{\overline{\alpha}}^+ \) be the morphism that maps every symbol in its equivalent class, that is, for every \( a_j \in A_{\overline{\alpha}}, \varphi(a_j) = a, \) where:

- \( a_i \in [a_j]_{\equiv} \).
- \( i \) is the smallest index for every \( a_k \in [a_j]_{\equiv} \).

\( \varphi(\overline{\alpha}) \) will be denoted as \( \overline{\alpha} \). It is clear that \( h(L(\overline{\alpha})) = h(L(\overline{\alpha})) = L(\alpha) \).

**Example 11** Let \( \alpha = (a + b)(a^* + ba^* + b^*)^* \).

- The linearized expression of \( \alpha \) is \( \overline{\alpha} = (a_1 + b_2)(a_3^* + b_4a_5^* + b_6^*)^* \).
- The classes of the equivalence relation \( \equiv_f \) of \( \overline{\alpha} \) are \( [\lambda], [a_1, b_2, a_3, b_6] \) and \( [b_4, a_5] \) (see the Table 1 of Example 8).
- The equivalence classes of the relation \( \equiv \) are then \( [\lambda], [a_1, a_3], [b_2, b_6], [b_4] \) and \( [a_5] \).

13
\begin{itemize}
  \item \(\varphi(a_1) = \varphi(a_3) = a_1, \varphi(b_2) = \varphi(b_6) = b_2, \varphi(b_4) = b_4, \varphi(a_5) = a_5\) and thus \(\overline{\alpha} = (a_1 + b_2)(a_1^* + b_1a_5^* + b_2^*)^*\).
\end{itemize}

**Proposition 12** \(\overline{\alpha}\) is a local language.

**Proof.** The morphism \(\varphi\) defined above is a strictly alphabetical morphism so, from Lemma 10, for any \(y \in A^+_{\overline{\alpha}}\) and any \(\overline{\alpha} \in A^*_{\overline{\alpha}}\) we have \((y\overline{\alpha})^{-1}\overline{\alpha} = \varphi(\bigcup_{\psi(y) = \alpha}(x\overline{\alpha})^{-1}\overline{\alpha})\). Then, given any \(x \in A^+_{\overline{\alpha}}\) and any \(a \in A_{\overline{\alpha}}\), the set \((x\overline{\alpha})^{-1}\overline{\alpha}\) is empty or denotes the same language. The same happens for any \(\overline{\alpha}\) such that \(\varphi(\overline{\alpha}) = \varphi(\overline{\alpha}) = \overline{\alpha}\), that is, if \((x\overline{\alpha})^{-1}\overline{\alpha} \neq \emptyset\) and \((x'\overline{\alpha})^{-1}\overline{\alpha} \neq \emptyset\) we have \((x\overline{\alpha})^{-1}\overline{\alpha} = (x'\overline{\alpha})^{-1}\overline{\alpha}\) and thus, \((y\overline{\alpha})^{-1}\overline{\alpha}\) is an empty set or is unique, so \(\overline{\alpha}\) denotes a local language. \(\blacksquare\)

### 4.1 The proposed method

The previous proposition and the fact that \(h(\overline{\alpha}) = \alpha\) show that, if an automaton for \(L(\overline{\alpha})\) is obtained and the subindexes are eliminated, the resulting automaton recognizes \(L(\alpha)\). Algorithm 4.1 summarizes the method we propose to obtain a non-deterministic automaton \(A\) from a regular expression \(\alpha\).

First, the algorithm we propose considers the equivalence relation \(\equiv_f\) to obtain the relation \(\equiv\) and compute \(\overline{\alpha}\). Then, the standard local automaton for this expression is obtained.

Note that if \(a, b\) are symbols in \(A_{\alpha}\), where \(a_i, b_j\) are their counterparts in \(A_{\overline{\alpha}}\) (that is, \(h(a_i) = a\) and \(h(b_j) = b\)), and such that \([a_i]_{\equiv_f} = [b_j]_{\equiv_f}\), then, for any \(x, x' \in A^+_{\overline{\alpha}}\) such that \((xa_i)^{-1}L(\overline{\alpha}) \neq \emptyset\) and \((x'b_j)^{-1}L(\overline{\alpha}) \neq \emptyset\) it follows that the language \((xa_i)^{-1}L(\overline{\alpha})\) is equal to the language \((x'b_j)^{-1}L(\overline{\alpha})\). Therefore, the states \(q\) and \(q'\) related, respectively to the derivatives \((xa_i)^{-1}L(\overline{\alpha})\) and \((x'b_j)^{-1}L(\overline{\alpha})\), are the same although the derivatives may look different.

Let us to abuse of notation and denote in the same way a state \(q\) of the automaton and the expression that represent the right language \(R_q\). Thus, finally, we apply the equivalence \(q \sim q'\) if \(h(q) = h(q')\) over the set of derivatives. This is correct because in \(L(h(\overline{\alpha})) = L(\alpha)\) the right languages of \(q\) and \(q'\) will be the same. After merging the states that fulfill this condition, the method applies the homomorphism that erases the subindexes and ends returning the resulting automaton.

In order to compare the size of the follow automaton, the partial derivatives automaton and our proposal, we first recall that:

- From Propositions 6 and 7, the follow method merges two states \(p\) and \(q\) if the right languages of \(p\) and \(q\) in \(L(\overline{\alpha})\) are the same.
- To do the same thing, the partial derivatives method takes into account if the regular expressions related to \(p\) and \(q\) are the same when the
Algorithm 4.1 Proposed algorithm to obtain small NFA for any given regular expression.

Input: A regular expression \( \alpha \).
Output: A non-deterministic automaton \( A \) such that \( L(A) = L(\alpha) \).

Method:

1. Obtain \( \overline{\alpha} \)
2. Build the standard local automaton \( A_{sl}(\overline{\alpha}) \)
3. Compute the relation \( \equiv_f \) from \( A_{sl}(\overline{\alpha}) \)
4. Apply the morphism \( \varphi \) to \( \overline{\alpha} \) obtaining \( \overline{\alpha} \)
5. Build the standard local automaton \( A_{sl}(\overline{\alpha}) \)
6. Obtain the quotient automaton \( A_{sl}(\overline{\alpha})/\equiv_f = (Q, A, \delta, q_0, F) \)

For each state \( p \) in \( Q \):

- \( \text{DerivativeSet}[p] = \emptyset \)

endFor

Store \( \overline{\alpha} \) in \( \text{DerivativeSet}[q_0] \)

For each transition \( (q, a, p) \) in \( \delta \):

- For each \( \beta \) in \( \text{DerivativeSet}[q] \):
  - Store \( a^{-1}\beta \) in \( \text{DerivativeSet}[p] \)

endFor

For each pair of states \( p \) and \( q \) in \( Q \):

- If \( h(\text{DerivativeSet}[p]) \cap h(\text{DerivativeSet}[p]) \neq \emptyset \):
  - Merge the states \( p \) and \( q \)

endIf

endFor

Return the resulting automaton without subindexes

EndMethod:

subindexes are erased. Therefore, this method considers whether or not the right languages of \( p \) and \( q \) in \( L(\alpha) \) are the same.

The method we propose considers first the follow equivalence to obtain \( \equiv \). We recall that, for any two symbols \( a_i, b_j \) in \( A_{sl}(\overline{\alpha}) \) such that \([a_i]_{\equiv_f} = [b_j]_{\equiv_f}\), the derivatives \((xa_i)^{-1}L(\overline{\alpha})\) and \((xb_j)^{-1}L(\overline{\alpha})\) will be the same. Therefore, the number of states of the obtained automaton is bounded above by the number of states of the follows automaton.

Furthermore, our proposal merges those states that have some derivative in common when the subindexes are erased. Thus, the number of states obtained by our method also is bounded above by the partial derivatives method.

Example 13 Let us consider the regular expression \( \alpha = (a + b)(a^* + ba^*) + \)
The linearized expression of $\alpha$ is $\overline{\alpha} = (a_1 + b_2)(a_3^* + b_4a_5^* + b_2^*)^*$. The classes of the equivalence relation $\equiv_f$ of $\overline{\alpha}$ are $[\lambda]$, $[a_1, b_2, a_3, b_6]$ and $[b_4, a_5]$ (see the Table 1 of Example 8). The equivalence classes of the relation $\equiv$ are then $[\lambda]$, $[a_1, a_3]$, $[b_2, b_6]$, $[b_4]$ and $[a_5]$. Their representative elements are respectively denoted as $\lambda$, $a_1$, $b_2$, $a_3$ and $a_5$ and thus, we obtain $\overline{\alpha} = (a_1 + b_2)(a_3^* + b_4a_5^* + b_2^*)^*$. The standard local automaton $A_{sl}(\overline{\alpha})$ is depicted in Fig 10.

Figure 10: $A_{sl}(\overline{\alpha})$ for $\overline{\alpha} = (a_1 + b_2)(a_3^* + b_4a_5^* + b_2^*)^*$.

The algorithm merges first those states that are $\equiv_f$ equivalent. Therefore, states denoted $a_1$ and $b_2$, as well as states $b_4$ and $a_5$, are merged. The resulting quotient automaton is shown in Figure 11.

Figure 11: Quotient automaton $A_{sl}(\overline{\alpha})/ \equiv_f$ for $\alpha = (a + b)(b^* + a^* + b^*)^*$.

In order to compute the derivatives it is possible to follow the transitions of the standard local automaton. Thus, for instance, it is not necessary to compute the derivatives of $q_0$ with respect to $b_4$ or $a_5$.

It is worth to be noticed that the expressions obtained as the derivatives of $q_0$ with respect to $a_1$ or $b_2$ might be different to the expressions obtained when the derivatives of $q_1$ with respect to $a_1$ or $b_2$ are computed. Nevertheless, as the automaton states, all of them represent the same language. We note this below together with all the derivatives.
The fact that each state may have different expressions, all of them representing the same language, is summarized by the algorithm in a list of derivatives indexed by the state identifiers, thus:

\[
\text{DerivativeSet}[q_0] = \{p\}
\]

\[
\text{DerivativeSet}[q_1] = \begin{cases} 
    h((a_1^* + b_4a_5^* + b_2^*)^*), \\
    h(a_1^*(a_1^* + b_4a_5^* + b_2^*)^*), \\
    h(b_2^*(a_1^* + b_4a_5^* + b_2^*)^*)
\end{cases}
\]

\[
\text{DerivativeSet}[q_2] = \{h(a_2^*(a_1^* + b_4a_5^* + b_2^*)^*)\}
\]

States \(q_1\) and \(q_2\) are to be merged because, when the subindexes are erased, both share the expression \(a^*(a^* + b^* + b^*)^*\). Once the morphism that eliminates the subindexes is applied, we obtain the automaton of Figure 12.

\[
\begin{array}{c}
q_0 \\
\downarrow \quad a,b \\
q_1
\end{array}
\]

Figure 12: Automaton for \(\alpha = (a + b)(a^* + ba^* + b^*)^*\) with our method.

### 4.2 An efficient algorithm

In order to improve the time complexity of the Algorithm 4.1, we take profit from the c-continuation automaton proposed by Champarnaud and Ziadi because it has all the information needed to determine the states to be merged.

We recall that, on the one hand, in order to merge the states \(p\) and \(q\), the follows method takes into account whether or not the right languages \(R_p^{A_d(\overline{\alpha})}\) and \(R_q^{A_d(\overline{\alpha})}\) are equal (using the relation \(\equiv_f\)). On the other hand, the partial derivatives method merges those states \(p\) and \(q\) such that the expressions for the right languages \(R_p^{h(A_d(\overline{\alpha}))}\) and \(R_q^{h(A_d(\overline{\alpha}))}\) are equal. Note that this criterion does not assure all the possible merges to be done.
Our improved algorithm combines both criteria in a two steps method. Algorithm 4.2 summarizes our approach.

Algorithm 4.2 Improved algorithm to obtain small NFA for any given regular expression.

**Input:** A regular expression \( \alpha \).

**Output:** A non-deterministic automaton \( A \) such that \( L(A) = L(\alpha) \).

**Method:**

1. Obtain the c-continuation automaton \( A_c(\alpha) \)
2. Compute the relation \( \equiv_f \)
3. Obtain \( A_c(\alpha)/\equiv_f \).
   (* The c-continuations of the merged states are not discarded *)
4. Erase the subindexes to each c-continuation
5. Merge those states which have at least one expression in common

**Return** the resulting automaton

**EndMethod:**

The first step of this new algorithm 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 (that is \( A_{sl}(\overline{\alpha})/\equiv_f \)), 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 are also merged.

**Example 14** Let us consider the regular expression \((a+b)(a^*+ba^*+b^*)^*\). Figure 7 shows the c-continuation automaton for \( \alpha \). The quotient of this automaton by the relation \( \equiv_f \) is shown in Figure 13.

Note that \( h(a_3^*(a_4^* + b_4a_5^* + b_6^*))^*) = h(a_5^*(a_3^* + b_4a_5^* + b_6^*))^* \). Thus, the resulting automaton is shown in Figure 12.

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.

**Example 15** Let us consider the regular expression \( \alpha = a(a+b)^*ab + b(a+b)^*ab \). Its linearized version is \( \overline{\alpha} = a_1(a_2 + b_3)^*a_4b_5 + b_6(a_7 + b_8)^*a_9b_{10} \). The c-continuation automaton for \( \alpha \) is shown in Figure 14.
\[
\begin{align*}
\{a_3^5 + b_4 a_5^5 + b_6^5\}^* \\
\{a_3^2 (a_3 + b_4 a_5^5 + b_6^5)\}^* \\
\{b_6^5 (a_3^3 + b_4 a_5^5 + b_6^6)\}^*
\end{align*}
\]

Figure 13: Quotient automata \(A_c(\alpha)/\equiv_f\), for the expression \((a + b)(a^* + ba^* + b^*)^*\). First step performed by our improved algorithm.

\[
\begin{align*}
(a_1, (a_2 + b_3)^* a_4 b_5) \\
(a_2, (a_2 + b_3)^* a_4 b_5) \\
(b_1, (a_2 + b_3)^* a_4 b_5) \\
\{a_4, b_5\} \\
\{b_5, \lambda\}
\end{align*}
\]

\[
\begin{align*}
(a_7, (a_7 + b_8)^* a_9 b_{10}) \\
(b_8, (a_7 + b_8)^* a_9 b_{10}) \\
\{a_9, b_{10}\} \\
\{b_{10}, \lambda\}
\end{align*}
\]

Figure 14: \(A_c(\alpha)\) for \(\alpha = (a(a + b)^* ab + b(a + b)^* ab)\).

The classes of the follows relation are enumerated below:

\([\lambda], [a_1, a_2, b_3], [a_4], [b_6, a_7, b_8], [a_9], [b_5, b_{10}]\)

and the quotient automata (that is, the follows automaton) is shown in Figure 15.
Note that \( h( (a_2 + b_3)^*a_4b_5) = h( (a_7 + b_8)^*a_9b_{10}) \), (the merged state is named \( q_1 \) in the automaton in Figure 16) and \( h(b_5) = h(b_{10}) \) (state \( q_2 \) in Figure 16). Thus, the final automaton is shown in Figure 16.

Note that, in this case, our method and the partial derivatives one obtain the same automaton.

5 Conclusions

Although the time complexity of both the follow automaton and the equation automaton (partial derivatives) method is the same, the size of the automata they obtain can not be compared.

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 bounded above by the size of the smallest automata obtained by the previous methods.

We have also described a method to construct the position automaton based in the fact that linearized regular expressions are local languages. The time complexity of this method is cubic with respect to the size of the regular expression.
References


