algorithms

MDPI

Article

Efficient Construction of the Equation Automaton

Faissal Ouardi [1,*,†], Zineb Lotfi [1,†] and Bilal Elghadyry [1,2,†]

1 Department of Computer Science, Faculty of Sciences, Mohammed V University in Rabat, Rabat 10000, Morocco; lotfi.ziineb@gmail.com (Z.L.); bilal.el-ghadyry@univ-littoral.fr (B.E.)
2 EA 4491-LISIC-Lab., Laboratoire d'Informatique Signal et Image de la Côte d'Opale Université Littoral Côte d'Opale, 62100 Calais, France
* Correspondence: f.ouardi@um5r.ac.ma
† These authors contributed equally to this work.

Abstract: This paper describes a fast algorithm for constructing directly the equation automaton from the well-known Thompson automaton associated with a regular expression. Allauzen and Mohri have presented a unified construction of small automata and gave a construction of the equation automaton with time and space complexity in $O(m \log m + m^2)$, where m denotes the number of Thompson automaton transitions. It is based on two classical automata operations, namely epsilon-removal and Hopcroft's algorithm for deterministic Finite Automata (DFA) minimization. Using the notion of c-continuation, Ziadi et al. presented a fast computation of the equation automaton in $O(m^2)$ time complexity. In this paper, we design an output-sensitive algorithm combining advantages of the previous algorithms and show that its computational complexity can be reduced to $O(m \times |Q_{\equiv_e}|)$, where $|Q_{\equiv_e}|$ denotes the number of states of the equation automaton, by an epsilon-removal and Bubenzer minimization algorithm of an Acyclic Deterministic Finite Automata (ADFA).

Keywords: regular expressions; finite automata; efficient algorithms

Citation: Ouardi, F.; Lotfi, Z.; Elghadyry, B. Efficient Construction of the Equation Automaton. *Algorithms* **2021**, *14*, 238. https://doi.org/10.3390/a14080238

Academic Editor: Frank Werner

Received: 27 May 2021
Accepted: 6 August 2021
Published: 11 August 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The equation automaton (also known as derived terms automaton or Antimirov automaton) was first introduced in Mirkin's paper [1]. In [2], Antimirov introduced the notion of partial derivative of a regular expression, that lead to another definition and construction of the equation automaton. It is an ϵ-free NFA which has in general smaller number of states and transitions than the well-known position automaton [3–5]. The complexity of the original construction algorithm of [2], which is based on the computation of the set of partial derivatives of the expression, is in $O(n^5)$, where n denotes the size of the regular expression. In 2001, Champarnaud and Ziadi [6] introduced the notion of canonical derivatives and constructed a new automaton called the c-continuation automaton. They also proved that this automaton is isomorphic to the position automaton and that the equation automaton is its quotient for some equivalence relation.

The notion of c-derivative has been introduced in [6] to derive the equation automaton from the position automaton via the c-continuation automaton. A unique regular expression over indexed and ordered letters, called c-continuation, is assigned to each state of the position automaton. The resulting automaton is called the c-continuation automaton [6]. After that, one can define the equivalence relation between two c-continuations i.e., two states of the c-continuation automaton as follows: if deleting the indices of letters from two c-continuations results in the same regular expression, they correspond to the same partial derivative. Hence, the equation automaton would be a quotient of the c-continuation automaton w.r.t. the previously defined equivalence relation. From the algorithmic point of view, this result allows the construction of the equation automaton in $O(n^2)$ time and space [6,7]. Therefore, this improves the Antimirov's algorithm by a factor of $O(n^3)$.

In [8], Allauzen and Mohri present simple and unified constructions of the position automata [3–5], follow automata [9,10], and the equation automata [2,6,7] from regular

expressions. Their algorithms are based on two standard automata operations applied to the Thompson automata [11,12] called: epsilon-remove and Hopcroft's algorithm for DFA minimization [13]. The complexity of their construction for the equation automaton is in $O(m \log\ m + m^2)$, where m is the number of Thompson automaton transitions. Notice that, by construction the number of transitions of the Thompson automaton m and the size of a regular expression n are proportional. Thus, we have $m = O(n)$.

To improve the time complexity of computing the equation automaton from a regular expression E, we design an algorithm, combining advantages of previous methods [6–8], with a worst-case time complexity in $O(m \times |Q_{\equiv_e}|)$, where $|Q_{\equiv_e}|$ denotes the number of its states. Our approach is based on Bubenzer minimization of an acyclic DFA instead Hopcroft's algorithm for DFA minimization step used in Allauzen and Mohri's method. The main idea is to associate implicitly each c-continuation to a corresponding state, called *position state* in the Thompson automaton by a special marking of ε-transitions. As a consequence of this marking, the right language of each position state in the Thompson automaton represents implicitly its c-continuation, called *pseudo-continuation*. After that, we disable temporarily the cyclic ε-transition in the Thompson automaton and perform Bubenzer minimization of an acyclic DFA to compute efficiently partial derivatives equivalence relation over the set of position states. Finally, we remove indexed ε-transitions, enable the cyclic ε-transition and then compute the ε-closure of states in the produced automaton from the previous step to get the equation automaton. The implementation of the proposed algorithm is available under the repository https://github.com/FaissalOuardi/Equation-automaton, (accessed on 27 May 2021).

The paper is organized as follows. Section 2 contains some basic definitions and necessary preliminaries. Section 3 summarizes theoretical results that lead to c-continuations of a regular expression, and their relations with the partial derivatives. The definition of the c-continuation automaton is recalled, as well as the way it is connected to the equation automaton. Section 4 is a recall to the algorithm due to Allauzen and Mohri. We detail then in Section 5 the algorithmic refinements leading to an $O(m \times |Q_{\equiv_e}|)$ time complexity of the efficient construction of the equation automaton where $|Q_{\equiv_e}|$ is the number of its states.

2. Preliminaries

In this section, we introduce briefly the notion of finite automata. For further details on formal aspects of finite automata theory, we particularly recommend reading classical books [14,15].

2.1. Regular Expressions and Finite Automata

Let A be a non-empty finite set of letters, called an *alphabet*. The set of all words over A is denoted by A^*. ε is the empty word. A language over A is a subset of A^*.

2.1.1. Regular Expressions and Languages

A regular expression over the alphabet A is a term of the algebra $\mathcal{T}_{reg(A)}$ defined over the set $A \cup \{0,1\}$ with the symbols of functions $*, +, \cdot$, where $*$ is unary and $+$ and $\cdot$ are binary. Properties of the constants $0, 1$ and the operators $*, +$, and $\cdot$ lead to identities on this algebra. Each regular expression denotes a language. L is the function that assigns to each regular expression the regular language it denotes. $\mathrm{L} : T_{reg(A)} \longrightarrow reg(A^*)$ is defined as follows:

$$\begin{aligned} L(0) &= \emptyset \\ L(1) &= \{\varepsilon\} \\ L(a) &= a, \text{ for each } a \text{ in } A \\ L(\mathrm{F}+\mathrm{G}) &= L(\mathrm{F}) \cup L(\mathrm{G}) \\ L(\mathrm{F}\cdot\mathrm{G}) &= L(\mathrm{F})L(\mathrm{G}) \\ L(\mathrm{F}^*) &= L(\mathrm{F})^* \end{aligned}$$

The following identities are classically used:

$$0 + \mathrm{E} = \mathrm{E} = \mathrm{E} + 0, \;\; 1 \cdot \mathrm{E} = \mathrm{E} = \mathrm{E} \cdot 1, \;\; 0 \cdot \mathrm{E} = 0 = \mathrm{E} \cdot 0.$$

Let E be a regular expression. The set of letters occurring in E is denoted by A_{E}. To specify their position in the expression, letters are subscripted following the order of reading. The resulted expression is *the linearized form* of E, denoted by $\overline{\mathrm{E}}$. For example, starting from $\mathrm{E} = (a+b)^* aba + 1$, one obtains the linearized version $\overline{\mathrm{E}} = (a_1 + b_2)^* a_3 b_4 a_5 + 1$ of E. The subscripted letters are called *positions*; the set of all position in the expression E is denoted by $\mathrm{pos}(\mathrm{E})$. For the previous example, we have $\mathrm{pos}(\mathrm{E}) = \{a_1, b_2, a_3, b_4, a_5\}$. If F is a subexpression of E, we denote by $\mathrm{pos}_{\mathrm{E}}(\mathrm{F})$ the subset of positions of E that are letters of F. We say that a regular expression is in linear form if each letter of the expression occurs only once. We denote by h the function that maps each position in $\mathrm{pos}(\mathrm{E})$ to the letter of A_{E} that appears at this position in E. For $\mathrm{E} = (a+b)^* aba + 1$, we have $h(a_1) = h(a_3) = h(a_5) = a$ and $h(b_2) = h(b_4) = b$. The *size* of the regular expression E, denoted by $|\,\mathrm{E}\,|$, is the number of nodes in its syntax tree. We call *alphabetic width* of E, denoted by $||\,\mathrm{E}\,||$, the number of occurrences of letters in the expression i.e., the cardinality of $\mathrm{pos}(\mathrm{E})$. The alphabetic width of the expression $(a+b)^* aba + 1$ is equal to 5; its size is equal to 12.

Notice that the alphabetic width and the size of a regular expression are independent parameters. Therefore complexities are expressed w.r.t. both of these two parameters. However, it is usual to preprocess the input expression in order to reduce its size and to make its size proportional to its alphabetic width. So, if we consider a reduced regular expression E w.r.t. the following rules:

- $1 + 1 = 1$,
- $1 + \mathrm{E} + 1 = 1 + \mathrm{E}$,
- E is in Star-Normal Form (SNF) [16].

Thus, we have in this case $|\,\mathrm{E}\,| = O(||\,\mathrm{E}\,||)$. It is known that regular expressions can be transformed to SNF in linear time [16]. $\lambda(\mathrm{E})$ denote the null term of E, that is

$$\lambda(\mathrm{E}) = \begin{cases} 1 & \text{if } \varepsilon \in \mathrm{L}(\mathrm{E}), \\ 0 & \text{otherwise.} \end{cases}$$

By $T(\mathrm{E})$ we denote the syntax tree associated with the regular expression E. A node in $T(\mathrm{E})$ will be denoted by ν. We write $\mathit{Nodes}(\mathrm{E})$ for the set of nodes of $T(\mathrm{E})$. If $\nu \in \mathit{Nodes}(\mathrm{E})$ is a node in $T(\mathrm{E})$, $\mathit{sym}(\nu)$, $\mathit{father}(\nu)$ and $\mathit{right}(\nu)$ denote respectively the symbol, the father and the right son of the node ν. If $\mathit{sym}(\nu)$ is an operator, E_ν will denote the subexpression that corresponds to the subtree with the root ν.

2.1.2. Finite Automata and Recognizable Languages

A nondeterministic finite automaton (NFA) is a quintuple $\mathcal{A} = \langle Q, A, q_0, \delta, F \rangle$ where Q is a finite set of states, A is the alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times (A \cup \{\varepsilon\}) \to 2^Q$ is the transition function. The size of an automaton $\mathcal{A}$, denoted by $|\mathcal{A}|$, is the number of its states. The automaton $\mathcal{A}$ is called *deterministic* (DFA) if there is only one initial state, $|\delta(q, \varepsilon)| = 0$ and $|\delta(q, a)| = 1$, for any $q \in Q$, for any $a \in A$. A path in $\mathcal{A}$ is a sequence (q_i, a_i, q_{i+1}), $i = 1, \cdots, n$, of consecutive transitions. Its label is the word $w = a_1 a_2 \cdots a_n$. A word $w = a_1 a_2 \cdots a_n$ is recognized by the automaton $\mathcal{A}$ if there exists a path labeled w such that $q_1 = q_0$ and $q_{n+1} \in F$.

The language recognized by the automaton $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of words it recognizes. The right language of a state q in the automaton $\mathcal{A}$, denoted by $\overrightarrow{L}_q(\mathcal{A})$, is obtained by setting q to be the initial state, i.e., $\overrightarrow{L}_q(\mathcal{A}) = \{w \in A^* \mid \delta(q, w) \cap F \neq \emptyset\}$.

We say that $\mathcal{A}$ is acyclic if the underlying graph is acyclic. The language associated with an acyclic automaton is finite.

Let $\sim$ be an equivalence relation over Q. For $q \in Q$, $[q]$ denotes the equivalence class of q w.r.t. $\sim$ and, for $C \subseteq Q$, $C/_\sim$ denotes the quotient set $C/_\sim = \{[q] | q \in C\}$. We say that $\sim$ is right invariant w.r.t. $\mathcal{A}$ if and only if the following conditions hold:

- $\sim \subseteq (Q - F)^2 \cup F^2$ (final and non-final states are not $\sim$-equivalent),
- for any $p, q \in Q$, $a \in A$, if $p \sim q$, then $\delta(p, a)/_\sim = \delta(q, a)/_\sim$.

2.2. Thompson Automaton

In [12], Thompson gave a linear time and space algorithm to convert a regular expression E to an NFA with ε-transitions, denoted by $\mathcal{T}_{\mathrm{E}}$. The recursive steps of the construction of Thompson NFA are pictured in Figure 1.

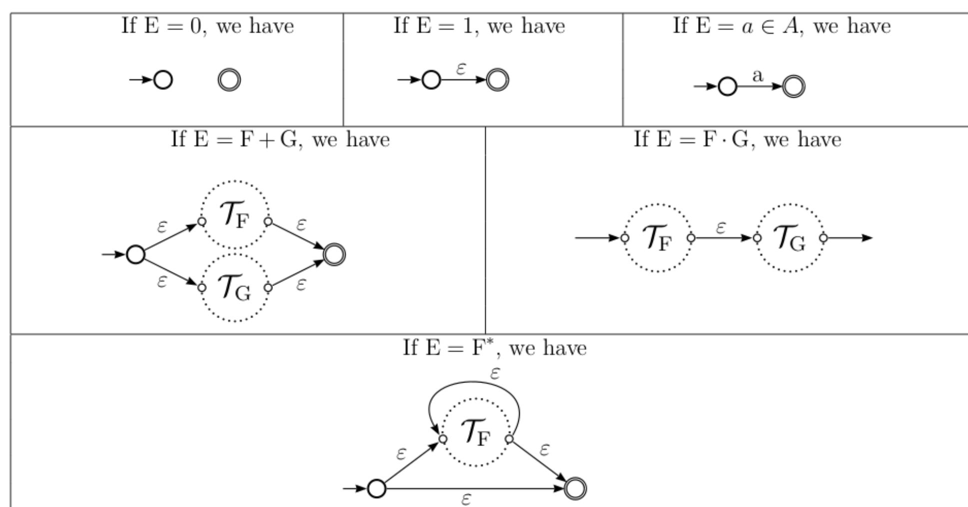


Figure 1. Thompson construction of an NFA.

Example 1. *Let us consider the regular expression* $\mathrm{E} = (a^* + ba^* + b^*)^*$. *The Thompson automaton* $\mathcal{T}_{\mathrm{E}}$ *associated with* E *is shown in the Figure 2.*

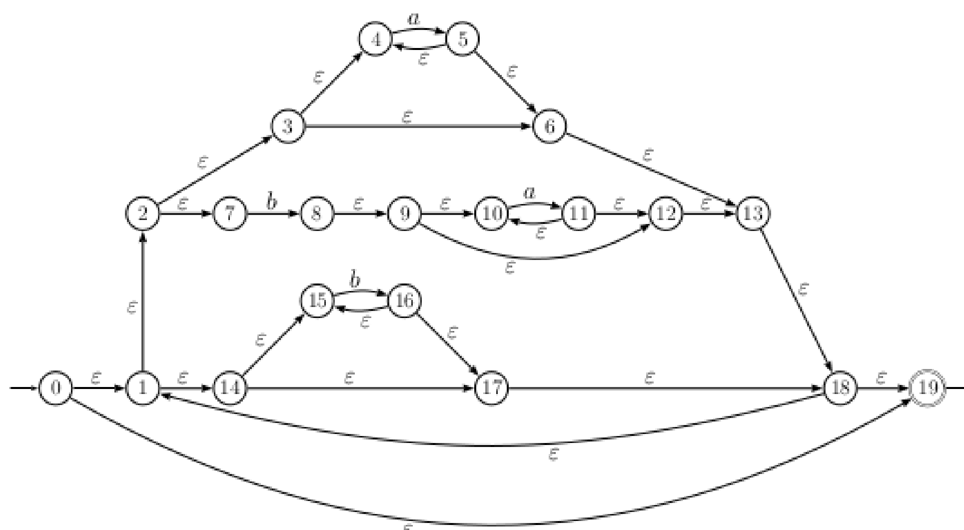


Figure 2. Thompson automaton $\mathcal{T}_{\mathrm{E}}$.

There are some disadvantages of Thompson's NFA when it is used in practice: it has many redundant states and ε-transitions, its number of states is in $O(|\mathrm{E}|)$ while other constructions offer NFAs with $O(||\mathrm{E}||)$ states.

In the next section, we will present the construction of a reduced ε-free automaton, named equation automaton, sometimes called Antimirov automaton or derived terms automaton.

3. Equation Automaton

The equation automaton has been introduced for the first time by Mirkin in [1]. In 1996, Antimirov introduced the notion of partial derivatives and used it to define the equation automaton [2]. Champarnaud and Ziadi [6] defined the notion of canonical derivatives of a linear expression and constructed a new automaton called the c-continuation automaton. They also proved that this automaton is isomorphic to the position automaton in the sense that the two automata have identical sets of states, identical initial and final states, and transitions Theorem 6 in [6]. Using an equivalence relation over the set of states of the c-continuation automaton, they derive the equation automaton in quadratic time.

The definition of the equation automaton of a regular expression is based on that of the partial derivatives of regular expressions, which are multisets of regular expressions over A. The partial derivative of E with respect to $a \in A$ is defined recursively on the structure of E as follows:

$$\begin{aligned}
\partial_a(0) &= \partial_a(1) = \emptyset, \\
\partial_a(x) &= \ \textbf{if } a = x \textbf{ then } \{1\} \textbf{ else } \emptyset, \\
\partial_a(\mathrm{F}+\mathrm{G}) &= \partial_a(\mathrm{F}) \cup \partial_a(\mathrm{G}), \\
\partial_a(\mathrm{F}\cdot\mathrm{G}) &= \partial_a(\mathrm{F})\cdot\mathrm{G} \cup \lambda(\mathrm{F})\cdot\partial_a(\mathrm{G}), \\
\partial_a(\mathrm{F}^*) &= \partial_a(\mathrm{F})\cdot\mathrm{F}^*.
\end{aligned}$$

The partial derivative of E with respect to the string $u \in A^*$ is denoted by $\partial_u(\mathrm{E})$ and recursively defined by $\partial_\varepsilon(\mathrm{E}) = \mathrm{E}$ and $\partial_{ua}(\mathrm{E}) = \partial_a(\partial_u(\mathrm{E}))$.

Let $D(\mathrm{E}) = \{\mathrm{E}\} \cup \{\mathrm{E}' \mid \mathrm{E}' \in \partial_u(\mathrm{E}) \text{ with } u \in A^*\}$.

Theorem 1. *(Antimirov [2]). The cardinality of the set $D(\mathrm{E})$ of all partial derivatives of a regular expression* E *is less than or equal to* $||\,\mathrm{E}\,|| + 1$.

The equation automaton $\mathcal{E}_\mathrm{E} = \langle D(\mathrm{E}), A_\mathrm{E}, \mathrm{E}, \delta, F\rangle$ of E is defined by:

- $\delta(\mathrm{E}', a) = \{\mathrm{E}'' \in D(\mathrm{E}) \mid \mathrm{E}'' \in \partial_a(\mathrm{E}')\}$,
- $F = \{\mathrm{E}' \in D(\mathrm{E}) \mid \lambda(\mathrm{E}') = 1\}$.

Example 2. *Let us consider the regular expression* $\mathrm{E} = (a^* + ba^* + b^*)^*$. *The partial derivatives of* E *are as follows:*

$$\begin{aligned}
\partial_a(\mathrm{E}) &= \{a^*(a^* + ba^* + b^*)^*\} \\
\partial_b(\mathrm{E}) &= \{a^*(a^* + ba^* + b^*)^*,\ b^*(a^* + ba^* + b^*)^*\}
\end{aligned}$$

The computation of the transitions of the equation automaton $\mathcal{E}_\mathrm{E}$ are as follows:

$$\begin{aligned}
\partial_a(a^*(a^*+ba^*+b^*)^*) &= \partial_a(a^*)(a^*+ba^*+b^*)^* \cup \partial_a((a^*+ba^*+b^*)^*) \\
&= \partial_a(a)a^*(a^*+ba^*+b^*)^* \cup (\partial_a(a^*) \cup \partial_a(ba^*) \cup \partial_a(b^*))(a^*+ba^*+b^*)^* \\
&= \{a^*(a^*+ba^*+b^*)^*\} \\
\partial_b(a^*(a^*+ba^*+b^*)^*) &= \partial_b(a^*)(a^*+ba^*+b^*)^* \cup \partial_b((a^*+ba^*+b^*)^*) \\
&= \partial_b(a)a^*(a^*+ba^*+b^*)^* \cup (\partial_b(a^*) \cup \partial_b(ba^*) \cup \partial_b(b^*))(a^*+ba^*+b^*)^* \\
&= (\partial_b(b)a^* \cup \partial_b(b)b^*)(a^*+ba^*+b^*)^* \\
&= \{a^*(a^*+ba^*+b^*)^*, b^*(a^*+ba^*+b^*)^*\} \\
\partial_a(b^*(a^*+ba^*+b^*)^*) &= \partial_a(b^*)(a^*+ba^*+b^*)^* \cup \partial_a((a^*+ba^*+b^*)^*) \\
&= (\partial_a(a^*) \cup \partial_a(ba^*) \cup \partial_a(b^*))(a^*+ba^*+b^*)^* \\
&= \{a^*(a^*+ba^*+b^*)^*\} \\
\partial_b(b^*(a^*+ba^*+b^*)^*) &= \partial_b(b^*)(a^*+ba^*+b^*)^* \cup \partial_b((a^*+ba^*+b^*)^*) \\
&= \partial_b(b)b^*(a^*+ba^*+b^*)^* \cup (\partial_b(a^*) \cup \partial_b(ba^*) \cup \partial_b(b^*))(a^*+ba^*+b^*)^* \\
&= \{b^*(a^*+ba^*+b^*)^*\} \cup (\partial_b(b)a^* \cup \partial_b(b)b^*)(a^*+ba^*+b^*)^* \\
&= \{b^*(a^*+ba^*+b^*)^*, a^*(a^*+ba^*+b^*)^*\}
\end{aligned}$$

The equation automaton $\mathcal{E}_\mathrm{E}$ associated with E is shown in Figure 3.

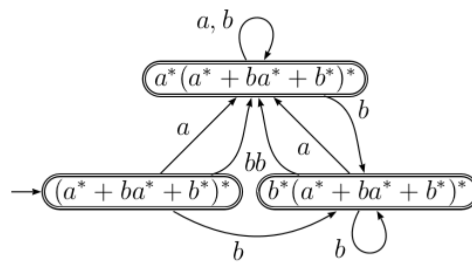


Figure 3. The equation automaton $\mathcal{E}_{\mathrm{E}}$.

In the following, we recall the definition and properties of the c-continuation automaton. Next, we show how it can be bound to the equation automaton.

3.1. C-Continuation Automaton

This automaton has been introduced by Champarnaud and Ziadi [6] to efficiently compute the equation automaton. Let us recall the notion of c-derivative, c-continuation and c-continuation automaton.

Definition 1. *(c-derivative with respect to a letter). The c-derivative of a regular expression* E *with respect to a letter* a *is the regular expression* $d_a(\mathrm{E})$ *defined by:*

$$\begin{array}{rcl} d_a(0) &=& d_a(1) = 0, \\ d_a(x) &=& \textbf{if } a = x \textbf{ then } 1 \textbf{ else } 0, \\ d_a(\mathrm{F}+\mathrm{G}) &=& \textbf{if } d_a(\mathrm{F}) \neq 0 \textbf{ then } d_a(\mathrm{F}) \textbf{ else } d_a(\mathrm{G}), \\ d_a(\mathrm{F}\cdot\mathrm{G}) &=& \textbf{if } d_a(\mathrm{F}) \neq 0 \textbf{ then } d_a(\mathrm{F})\cdot\mathrm{G} \textbf{ else } \lambda(\mathrm{F})\cdot d_a(\mathrm{G}), \\ d_a(\mathrm{F}^*) &=& d_a(\mathrm{F})\cdot\mathrm{F}^*. \end{array}$$

The c-derivative with respect to a word $u = u_1 \cdots u_n$ is defined recursively by the rules: $d_\varepsilon(\mathrm{E}) = \mathrm{E}$ and $d_{u_1\cdots u_n}(\mathrm{E}) = d_{u_2\cdots u_n}(d_{u_1}(\mathrm{E}))$.

Theorem 2. *(Theorem 4 in [6]). Let* E *be a linear regular expression and* a *be a letter from* E*. Then all non-zero c-derivatives of the form* $d_{ua}(\mathrm{E})$*, where* u *is an arbitrary word, are equal.*

Theorem 2 allows us to define the c-continuation $c_a(\mathrm{E})$ of a in a linear expression E as the unique value of the non-zero c-derivatives $d_{ua}(\mathrm{E})$.

Proposition 1. *(Proposition 6 in [6]). For every letter* a *of a linear expression* E*, the c-continuation* $c_a(\mathrm{E})$ *is such that:*

$$\begin{array}{rcl} c_a(a) &=& 1, \\ c_a(\mathrm{F}+\mathrm{G}) &=& \textbf{if } c_a(\mathrm{F}) \texttt{ exists } \textbf{then } c_a(\mathrm{F}) \textbf{ else } c_a(\mathrm{G}), \\ c_a(\mathrm{F}\cdot\mathrm{G}) &=& \textbf{if } c_a(\mathrm{F}) \texttt{ exists } \textbf{then } c_a(\mathrm{F})\cdot\mathrm{G} \textbf{ else } c_a(\mathrm{G}), \\ c_a(\mathrm{F}^*) &=& c_a(\mathrm{F})\cdot\mathrm{F}^*. \end{array}$$

Corollary 1. *(Corollary 5 in [6]). For every letter* a *of a linear expression* E*, the c-continuation* $c_a(\mathrm{E})$ *is either 1 or a subexpression of* E *or a product of subexpressions.*

More precisely, for a linear regular expression E, we have $c_a(\mathrm{E}) = H_0 \cdots H_k$, where H_i is a subexpression of E, for all $0 \leq i \leq k$.

We now consider a regular expression E over A. Let $\overline{\mathrm{E}}$ be the linearized form of E over pos(E) and h be the mapping from pos(E) onto A_{E}.

In order to simplify the writing for a regular expression E, we consider by convention that $c_0(\mathrm{E}) = d_\varepsilon(\overline{\mathrm{E}}) = \overline{\mathrm{E}}$ and $c_x(\mathrm{E})$ will denote $c_x(\overline{\mathrm{E}})$.

Definition 2. *(c-continuation automaton) The c-continuation automaton of* E, $\mathcal{C}_{\mathrm{E}} = \langle Q, A_{\mathrm{E}}, i, \delta, F \rangle$*, is defined by:*

- $Q = \{(x, c_x(\mathrm{E})) \mid x \in \mathrm{pos}(\mathrm{E}) \cup \{0\}\}$,
- $i = (0, c_0(\mathrm{E}))$,
- $F = \{(x, c_x(\mathrm{E})) \mid \lambda(c_x(\mathrm{E})) = 1\}$,
- $\delta((x, c_x(\mathrm{E})), a) = \{(y, c_y(\mathrm{E}) \mid h(y) = a \text{ and } d_y(c_x(\mathrm{E})) \equiv c_y(\mathrm{E})\}, \forall x \in \mathrm{pos}(\mathrm{E}) \cup \{0\}$ *and* $\forall a \in A_{\mathrm{E}}$.

We note that the number of states of $\mathcal{C}_{\mathrm{E}}$ is exactly $||\,\mathrm{E}\,|| + 1$.

Corollary 2. *(Corollary 7 in [6]). Let* E *be a regular expression. One has:* $\mathrm{L}(\mathcal{C}_{\mathrm{E}}) = \mathrm{L}(\mathrm{E})$.

Example 3. *Let us consider the regular expression* $\mathrm{E} = (a^* + ba^* + b^*)^*$ *from Example 1. The linearized form of* E *is* $\overline{\mathrm{E}} = (a_1^* + b_2 a_3^* + b_4^*)^*$ *and the c-continuations of* E *are as follows:*

$$\begin{aligned}
c_0(\mathrm{E}) &= (a_1^* + b_2 a_3^* + b_4^*)^* & c_{a_3}(\mathrm{E}) &= c_{a_3}(a_1^* + b_2 a_3^* + b_4^*) \cdot (a_1^* + b_2 a_3^* + b_4^*)^* \\
c_{a_1}(\mathrm{E}) &= a_1^*(a_1^* + b_2 a_3^* + b_4^*)^* & &= c_{a_3}(b_2 a_3^*) \cdot (a_1^* + b_2 a_3^* + b_4^*)^* \\
c_{b_2}(\mathrm{E}) &= a_3^*(a_1^* + b_2 a_3^* + b_4^*)^* & &= c_{a_3}(a_3^*) \cdot (a_1^* + b_2 a_3^* + b_4^*)^* \\
c_{b_4}(\mathrm{E}) &= b_4^*(a_1^* + b_2 a_3^* + b_4^*)^* & &= a_3^*(a_1^* + b_2 a_3^* + b_4^*)^*
\end{aligned}$$

The outgoing transitions from the state $(0, c_0(\mathrm{E}))$ *are computed using the c-derivatives of* $c_0(\mathrm{E})$ *as follows:*

$$\begin{aligned}
&d_{a_1}(c_0(\mathrm{E})) = c_{a_1}(\mathrm{E}),\ d_{b_2}(c_0(\mathrm{E})) = c_{b_2}(\mathrm{E}),\ d_{a_3}(c_0(\mathrm{E})) = 0,\ d_{b_4}(c_0(\mathrm{E})) = c_{b_4}(\mathrm{E}) \\
&d_{a_1}(c_{a_1}(\mathrm{E})) = c_{a_1}(\mathrm{E}),\ d_{b_2}(c_{a_1}(\mathrm{E})) = c_{b_2}(\mathrm{E}),\ d_{a_3}(c_{a_1}(\mathrm{E})) = 0,\ d_{b_4}(c_{a_1}(\mathrm{E})) = c_{b_4}(\mathrm{E}) \\
&d_{a_1}(c_{b_2}(\mathrm{E})) = c_{a_1}(\mathrm{E}),\ d_{b_2}(c_{b_2}(\mathrm{E})) = c_{b_2}(\mathrm{E}),\ d_{a_3}(c_{b_2}(\mathrm{E})) = c_{a_3}(\mathrm{E}),\ d_{b_4}(c_{b_2}(\mathrm{E})) = c_{b_4}(\mathrm{E}) \\
&d_{a_1}(c_{a_3}(\mathrm{E})) = c_{a_1}(\mathrm{E}),\ d_{b_2}(c_{a_3}(\mathrm{E})) = c_{b_2}(\mathrm{E}),\ d_{a_3}(c_{a_3}(\mathrm{E})) = c_{a_3}(\mathrm{E}),\ d_{b_4}(c_{a_3}(\mathrm{E})) = c_{b_4}(\mathrm{E}) \\
&d_{a_1}(c_{b_4}(\mathrm{E})) = c_{a_1}(\mathrm{E}),\ d_{b_2}(c_{b_4}(\mathrm{E})) = c_{b_2}(\mathrm{E}),\ d_{a_3}(c_{b_4}(\mathrm{E})) = 0,\ d_{b_4}(c_{b_4}(\mathrm{E})) = c_{b_4}(\mathrm{E})
\end{aligned}$$

Then we get the c-continuation automaton $\mathcal{C}_{\mathrm{E}}$ *in Figure 4.*

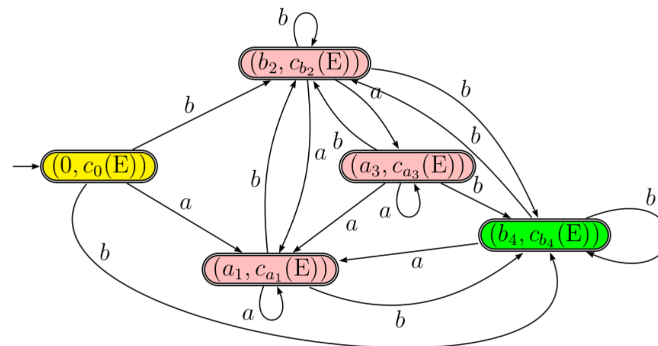


Figure 4. The c-continuation automaton $\mathcal{C}_{\mathrm{E}}$.

3.2. Equation Automaton as a Quotient of C-Continuation Automaton

Champarnaud and Ziadi [6] have proved that the equation automaton is a quotient of the c-continuation automaton. Let us consider the equivalence relation $\equiv_e$ defined by

$$(x, c_x(\mathrm{E})) \equiv_e (y, c_y(\mathrm{E})) \Leftrightarrow h(c_x(\mathrm{E})) \equiv h(c_y(\mathrm{E})) \tag{1}$$

Sometimes we write $x \equiv_e y \Leftrightarrow h(c_x(\mathrm{E})) \equiv h(c_y(\mathrm{E}))$.

Proposition 2. *The relation* $\equiv_e$ *is right-invariant, i.e., for all letters* a *in* A*, for all pairs of states* $(x, c_x(\mathrm{E})), (y, c_y(\mathrm{E}))$ *in* Q *such that* $(x, c_x(\mathrm{E})) \equiv_e (y, c_y(\mathrm{E}))$*, we have:* $\delta((x, c_x(\mathrm{E})), a)/_{\equiv_e} = \delta((y, c_y(\mathrm{E})), a)/_{\equiv_e}$.

Moreover, if two states are equivalent w.r.t. $\equiv_e$, then they are either both final or both non-final, since $(x, c_x(\mathrm{E})) \in F \Leftrightarrow \lambda(c_x(\mathrm{E})) = 1 \Leftrightarrow \lambda(h(c_x(\mathrm{E}))) = 1$.

The equivalence class of the state $(x, c_x(\mathrm{E}))$ is represented by $\mathrm{C}_x = h(c_x(\mathrm{E}))$. Since the relation $\equiv_e$ is right-invariant, we can define the quotient automaton $\mathcal{C}_\mathrm{E}/_{\equiv_e} = \langle Q_{\equiv_e}, A_\mathrm{E}, q_0, \delta, F\rangle$ as follows:

- $Q_{\equiv_e} = \{\mathrm{C}_x \mid x \in \mathrm{pos(E)} \cup \{0\}\}$,
- $q_0 = \mathrm{C}_0$,
- $F = \{\mathrm{C}_x \mid \lambda(c_x(\mathrm{E})) = 1\}$,
- $\delta(\mathrm{C}_x, a) = \{\mathrm{C}_y \mid h(y) = a \text{ and } d_y(c_x(\mathrm{E})) \equiv c_y(\mathrm{E})\}$, $\forall \mathrm{C}_x \in Q_{\equiv_e}$ and $\forall a \in A_\mathrm{E}$.

Theorem 3. *(Theorem 10 in [6]). Let* E *be a regular expression. The automaton $\mathcal{C}_\mathrm{E}/_{\equiv_e}$ deduced from the c-continuation automaton is isomorphic to the equation automaton $\mathcal{E}_\mathrm{E}$.*

We note that the number of states of $\mathcal{E}_\mathrm{E}$ is majorized by $||\,\mathrm{E}\,|| + 1$.

Example 4. *Let us consider the regular expression* $\mathrm{E} = (a^* + ba^* + b^*)^*$ *from Example 1. There are three $\equiv_e$-equivalence classes when applying the function h that remove indices from letters for different c-continuations of* E*:*

$$h(c_0(\mathrm{E})) = (a^* + ba^* + b^*)^*$$
$$h(c_{a_1}(\mathrm{E})) = h(c_{b_2}(\mathrm{E})) = h(c_{a_3}(\mathrm{E})) = a^*(a^* + ba^* + b^*)^*$$
$$h(c_{b_4}(\mathrm{E})) = b^*(a^* + ba^* + b^*)^*$$

The c-continuation automaton $\mathcal{C}_\mathrm{E}$ and the quotient automaton $\mathcal{C}_\mathrm{E}/_{\equiv_e}$ which is isomorphic to the equation automaton $\mathcal{E}_\mathrm{E}$ are schematized in Figure 5:

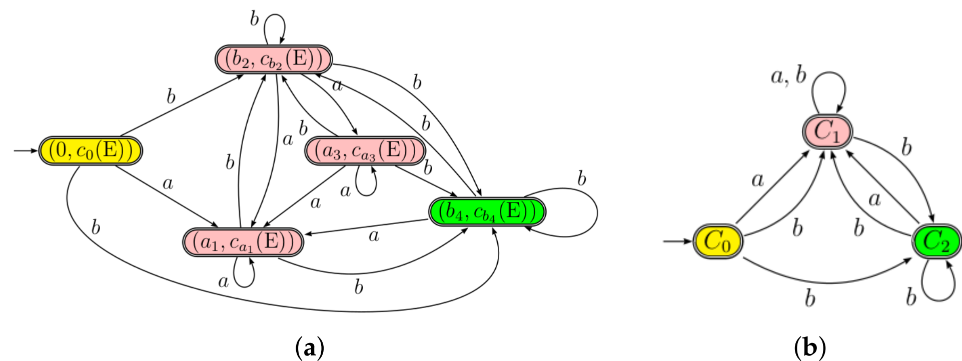
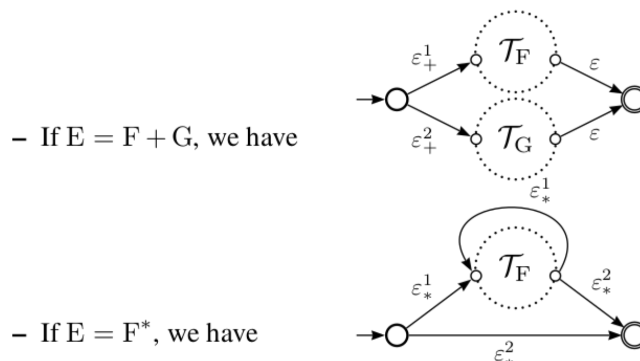


Figure 5. (**a**) The c-continuation automaton $\mathcal{C}_\mathrm{E}$ versus (**b**) The quotient automaton $\mathcal{C}_\mathrm{E}/_{\equiv_e}$.

4. Allauzen and Mohri's Algorithm

In [8], Allauzen and Mohri compute the equation automaton from the Thompson automaton of a regular expression E in $O(m\log\, m + m^2)$ time. Their algorithm is based on some combinations of ε-transitions removal and Hopcroft's algorithm for DFA minimization to the classical Thompson automata [13]. In the next, we briefly describe their method.

Let $\widehat{A} = A_\mathrm{E} \cup \{\varepsilon^1_+, \varepsilon^2_+, \varepsilon^1_*, \varepsilon^2_*\}$. We denote by $\widehat{\mathcal{T}_\mathrm{E}}$ the automaton over $\widehat{A}$ obtained by recursively marking some of the ε-transitions of the Thompson automaton $\mathcal{T}_\mathrm{E}$ as follows:



Allauzen and Mohri have shown that the equation automaton can be obtained using some ε-transitions marking of the Thompson automaton and then apply two classical automata operations, namely epsilon removal, denoted by the function *rmeps* (resp. the function $\widehat{rmeps}$ for marked epsilon removal) and the Hopcroft's algorithm for DFA minimization [13], denoted by $min_{\mathbb{B}}$.

Proposition 3. *(Proposition 3 in [8]). We have* $\mathcal{E}_{\mathrm{E}} = \widehat{rmeps}(min_{\mathbb{B}}(rmeps(\widehat{\mathcal{T}_{\mathrm{E}}})))$.

Note that after removing ε-transitions from the automaton $\widehat{\mathcal{T}_{\mathrm{E}}}$, we obtain a deterministic finite automaton $rmeps(\widehat{\mathcal{T}_{\mathrm{E}}})$. After that, the Hopcroft's algorithm for DFA minimization is applied to derive the automaton $min_{\mathbb{B}}(rmeps(\widehat{\mathcal{T}_{\mathrm{E}}}))$ such that the set of its states is in bijection with the set of partial derivatives of E. Finally, to compute transitions of the equation automaton from $min_{\mathbb{B}}(rmeps(\widehat{\mathcal{T}_{\mathrm{E}}}))$, marked ε-transitions are removed using $\widehat{rmeps}$ operation.

Theorem 4. *(Theorem 3 in [8]). Let* E *be a regular expression over* A*. The equation automaton of* E *can be computed in* $O(m \log m + m^2)$ *time.*

5. Efficient Conversion Algorithm

In this section, we will show that the equation automaton $\mathcal{E}_{\mathrm{E}}$ of a regular expression E can be deduced from the associated Thompson automaton in $O(|\,\mathrm{E}\,| \cdot |Q_{\equiv_e}|)$ time, where $|Q_{\equiv_e}|$ denotes the number of states of $\mathcal{E}_{\mathrm{E}}$. Algorithm 1 summarizes the different steps of our approach.

Algorithm 1 Computation of the equation automaton.

input : The Thompson automaton $\mathcal{T}_{\mathrm{E}} = \langle Q, A_{\mathrm{E}}, I, \delta, F\rangle$ associated with a regular expression E.
output: The equation automaton $\mathcal{E}_{\mathrm{E}}$ associated with E.

```
/*                    Computation of states                    */
```

Compute $Id(\mathcal{T}_{\mathrm{E}})$:

Subexpressions identification over *states* of $\mathcal{T}_{\mathrm{E}}$.

- Define the sub-automaton $Id(\mathcal{T}_{\mathrm{E}})$ by marking recursively some ε-transitions of $\mathcal{T}_{\mathrm{E}}$ according to the following rules:
  if $\mathrm{E} = \mathrm{F} + \mathrm{G}$ **then**
  mark the ε-transitions by ε_{l+} (resp. ε_{r+}) from the initial state I to $I_{\mathcal{T}_{\mathrm{F}}}$ (resp. $I_{\mathcal{T}_{\mathrm{G}}}$)
  if $\mathrm{E} = \mathrm{F}^*$ **then**
  - mark the ε-transitions by ε_*^1 from the initial state I (resp. the final state $F_{\mathcal{T}_{\mathrm{F}}}$) to $I_{\mathcal{T}_{\mathrm{F}}}$ (resp. F) and by ε_*^2 from the initial state I to F.
  - temporarily disable the ε-transition from $F_{\mathcal{T}_{\mathrm{F}}}$ to $I_{\mathcal{T}_{\mathrm{F}}}$.
- Compute the function $N(q)$ that maps each state $q \in Q$ to a unique integer identifying the associated subexpression E_q, if it exists.

Compute $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$:

- Compute pseudo-continuations for all *position states* of $Id(\mathcal{T}_{\mathrm{E}})$.
- Merge equivalent states having the same pseudo-continuation.

```
/*        Computation of transitions and final states          */
```

Compute $rmeps(C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}})))$:

- Perform epsilon removal operation using $rmeps()$ function over $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$.

For convenience, we assume that the k states of a given finite automaton are identified by the integers $1, \cdots, k$.

From Corollary 1, the c-continuation $c_x(\mathrm{E}) = H_0 \cdots H_l$ associated with a position x is a concatenation of distinct subexpressions H_i of E, possibly reduced to a single subexpression or to 1. In the Thompson automaton $\mathcal{T}_{\mathrm{E}}$, we can associate a position x to a particular state q, called *position state* and define the associated pseudo-continuation $C(q) = N(H_0) \cdots N(H_l)$, where $N(H_i)$ denotes the integer that identify the initial state of the Thompson automaton $\mathcal{T}_{H_i}$. So, the first step, compute $Id(\mathcal{T}_{\mathrm{E}})$, of our algorithm consists on computing the function $N(.)$ such that for two subexpressions H_i and H_j of E, we have: $H_i \equiv H_j \Leftrightarrow N(I_{\mathcal{T}_{H_i}}) = N(I_{\mathcal{T}_{H_j}})$. This step can be done using a special marking of the ε-transitions of $\mathcal{T}_{\mathrm{E}}$ that makes it acyclic and deterministic and such that the right languages of its states represent the structure of the corresponding subexpressions. In the next step, Compute $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$, we re-mark the ε-transitions such that the resulted automaton is acyclic and deterministic and the right language of a position state in $Id(\mathcal{T}_{\mathrm{E}})$ represents a pseudo-continuation. After that, one can merge equivalent position states having the same right language. The final step is the computation of final states and transitions of the equation automaton using an epsilon removal operation, denoted by $rmeps(.)$, from the resulted automaton in the previous step.

In the next, we will show that the equation automaton $\mathcal{E}_{\mathrm{E}}$ can be computed efficiently from the Thompson automaton $\mathcal{T}_{\mathrm{E}}$ using the following operations $rmeps(C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}})))$.

5.1. Computation of States

In the following, We will show that the computation of the relation $\equiv_e$ over the states of the Thompson automaton can be performed in linear time w.r.t. the size of the expression using the minimization of an acyclic deterministic finite automaton. This minimization can be performed efficiently in $O(|\,\mathrm{E}\,|)$ time using Bubenzer's algorithm [17,18].

Before computing the equivalence classes $C_{\equiv_e}$ over states of the Thompson automaton, we will perform a preprocessing step to identify all identical sub-expressions of E. In the next, we will show that this identification can be done in $O(|\,\mathrm{E}\,|)$ time.

5.1.1. Sub-Expressions Identification

Let Exp the set of all subexpressions of E. In this preprocessing step, we will mark each state in the Thompson automaton by a unique letter in the set $\{1, 2, \ldots, |\,\mathrm{Exp}\,|\}$.

Let us define a bijection N between the set Exp and a finite set of letters $\{1, 2, \ldots, |\,\mathrm{Exp}\,|\}$. Consequently, if E_1 and E_2 are two sub-expressions of E, then we have:

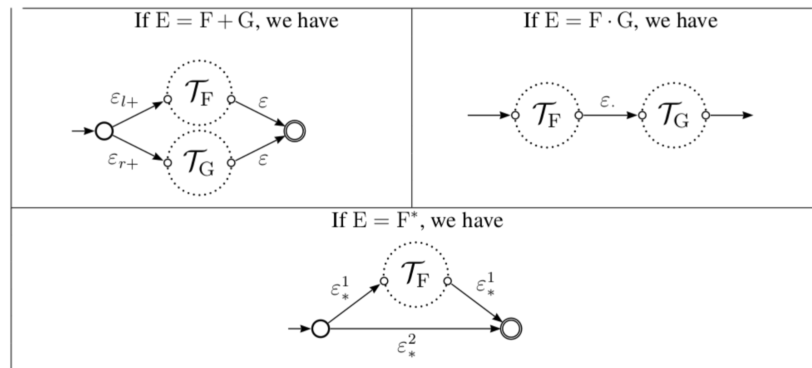
$$\mathrm{E}_1 \equiv \mathrm{E}_2 \Leftrightarrow \mathrm{N}(\mathrm{E}_1) = \mathrm{N}(\mathrm{E}_2) \tag{2}$$

Based on the parsing method, introduced in Section 6 in [19], that derive an equivalent regular expression from Thompson automaton, each subexpression H_i of E is associated with an integer identifying the initial state $I_{\mathcal{T}_{H_i}}$ in the Thompson automaton $\mathcal{T}_{\mathrm{E}}$.

Let q be a state in $\mathcal{T}_{\mathrm{E}}$, we denote by E_q the subexpression associated with q, if it exists. For abbreviation, $\mathrm{N}(q)$ represents $\mathrm{N}(\mathrm{E}_q)$.

In the following, we will show that the computation of the function N over the states of $\mathcal{T}_{\mathrm{E}}$ turns into a minimization of the acyclic deterministic sub-automaton of the Thompson automaton, $Id(\mathcal{T}_{\mathrm{E}}) = \langle Q', A_{\mathrm{E}}, I, \delta', F \rangle$, defined by:

- $A = \{\varepsilon_{l+}, \varepsilon_{\cdot}, \varepsilon_{r+}, \varepsilon^1_*, \varepsilon^2_*\} \cup A_{\mathrm{E}}$ where l (resp. r), denote left (resp. right),
- $Q' = \{(q, N(q)) |\; q \in Q\}$, i.e., a state in $\mathcal{T}_{\mathrm{E}}$ is augmented by the letter $N(q)$.
- The transition function δ' is defined over the Thompson automaton as follows:



Notice that this automaton is an acyclic deterministic sub-automaton of the Thompson automaton where ε-transitions are indexed and the cyclic transitions in the case when $\mathrm{E} = \mathrm{F}^*$ are temporarily disabled.

To compute identical subexpressions, we define the equivalence relation $\sim$ over the states of $Id(\mathcal{T}_\mathrm{E})$ as follows:

$$q \sim q' \Leftrightarrow \mathrm{N}(q) = \mathrm{N}(q') \tag{3}$$

Thus we have:

$$[q]_\sim = [q']_\sim \Leftrightarrow \mathrm{N}(\mathrm{E}_q) = \mathrm{N}(\mathrm{E}_{q'}) \tag{4}$$

Lemma 1. *Let q and q' be two states in $Id(\mathcal{T}_\mathrm{E})$. We have:*

$$\overrightarrow{L}_q(Id(\mathcal{T}_\mathrm{E})) = \overrightarrow{L}_{q'}(Id(\mathcal{T}_\mathrm{E})) \Leftrightarrow \mathrm{E}_q \equiv \mathrm{E}_{q'}$$

Proof. Obvious, by construction. □

Proposition 4. *The function $\mathrm{N}(.)$ can be computed over $Id(\mathcal{T}_\mathrm{E})$ in $O(|\,\mathrm{E}\,|)$ time.*

Proof. Let q and q' two states in $Id(\mathcal{T}_\mathrm{E})$. One has:

$$\begin{aligned} q \sim q' &\overset{(3)}{\Longleftrightarrow} \mathrm{N}(q) = \mathrm{N}(q') \\ &\overset{(2)}{\Longleftrightarrow} \mathrm{E}_q \equiv \mathrm{E}_{q'} \\ &\overset{\text{Lemma 1}}{\Longleftrightarrow} \overrightarrow{L}_q(Id(\mathcal{T}_\mathrm{E})) = \overrightarrow{L}_{q'}(Id(\mathcal{T}_\mathrm{E})) \end{aligned}$$

Thus, the equivalence relation $\sim$ coincides with Myhill-Nerode equivalence relation [20,21] over the states of $Id(\mathcal{T}_\mathrm{E})$. Since the automaton $Id(\mathcal{T}_\mathrm{E})$ is deterministic and acyclic, its minimization using Bubenzer's algorithm [17] requires $O(|\,\mathrm{E}\,|)$ time and space complexity. □

Example 5. *The automaton $Id(\mathcal{T}_\mathrm{E})$ obtained after performing the subexpression identification step for the regular expression $\mathrm{E} = (a^* + ba^* + b^*)^*$ through $\mathcal{T}_\mathrm{E}$.*

As shown in Figure 6, for the states 3 and 9 we have:

$$\left.\begin{aligned} \overrightarrow{L}_3(Id(\mathcal{T}_\mathrm{E})) &= \{\varepsilon_*^1 a \varepsilon_*^1 \varepsilon_*^1, \varepsilon_*^2 \varepsilon_*^1\} \\ \overrightarrow{L}_9(Id(\mathcal{T}_\mathrm{E})) &= \{\varepsilon_*^1 a \varepsilon_*^1 \varepsilon_*^1, \varepsilon_*^2 \varepsilon_*^1\} \end{aligned}\right. \Longleftrightarrow 3 \sim 9 \Longleftrightarrow N(3) = N(9)$$

As a consequence, we have $\mathrm{E}_3 \equiv \mathrm{E}_9$.

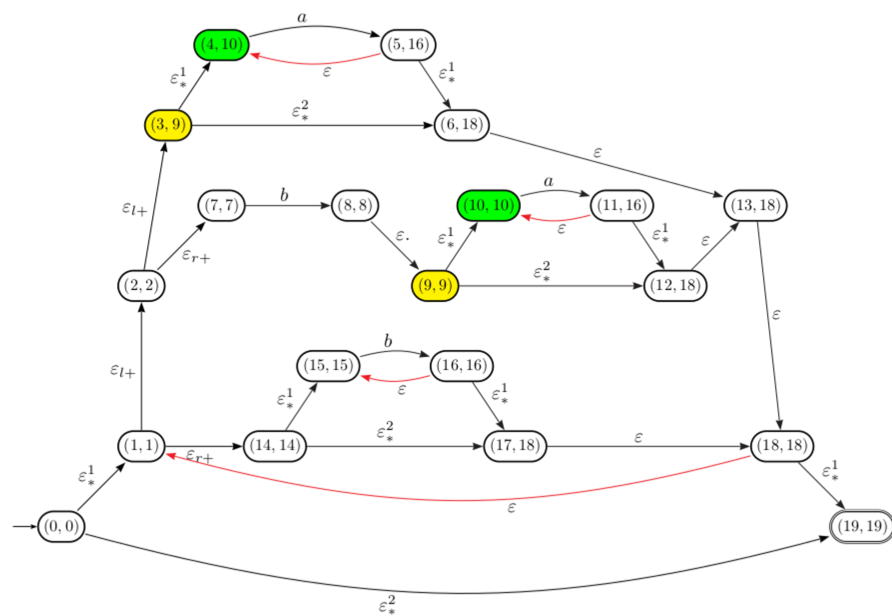


Figure 6. The automaton $Id(\mathcal{T}_{\mathrm{E}})$ associated with $\mathrm{E} = (a^* + ba^* + b^*)^*$.

5.1.2. $\equiv_e$-Equivalent States Merging

Let us now turn to the computation of the set of states of the equation automaton. From Corollary 1, the c-continuation $c_x(\mathrm{E})$ is a concatenation of distinct subexpressions of E, possibly reduced to a single subexpression or to 1. The following proposition shows that the c-continuation $c_x(\mathrm{E})$ can be computed over the syntactic tree $\mathrm{T}(\overline{\mathrm{E}})$ associated with the linearized version $\overline{\mathrm{E}}$.

Proposition 5. *(Ref. [6]). Let* E *be a regular expression and x a position in* $\mathrm{pos}(\mathrm{E})$*. Let* ν_x *be a node in* $\mathrm{T}(\overline{\mathrm{E}})$ *such that* $sym(\nu_x) = x$*. The c-continuation* $c_x(\mathrm{E})$ *is as follows:*

$$c_x(\mathrm{E}) = \bigodot_{\substack{\nu_x \preceq \nu \preceq \nu_{\overline{\mathrm{E}}} \\ f(\nu) \neq \perp}} \mathrm{E}_{f(\nu)}$$

where $\odot$ *is the concatenation operator.*

The function $f\ :\ Nodes(\mathrm{E}) \cup \{\perp\} \longrightarrow Nodes(\mathrm{E}) \cup \{\perp\}$ *is defined as follows:*

$$f(\nu) = \begin{cases} father(\nu) & \text{if } sym(father(\nu)) = * \text{ and } \nu \neq \nu_{\overline{\mathrm{E}}} \\ \mathrm{right}(father(\nu)) & \text{if } sym(father(\nu)) = \cdot \\ \perp & \text{otherwise.} \end{cases}$$

with $\perp$ *is an artificial node such that* $f(\perp) = \perp$*.*

Using Proposition 5, the computation of the set of states requires $O(|\,\mathrm{E}\,|^3)$ time and space complexity. This is due to the fact that the size of a c-continuation is in $O(|\,\mathrm{E}\,|^2)$. In order to reduce this complexity, we introduce a modified definition of pseudo-continuation introduced in [6] over an acyclic deterministic sub-automaton of the Thompson automaton, denoted by $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$. When merging $\equiv_e$-equivalent states over $Id(\mathcal{T}_{\mathrm{E}})$, we get the automaton $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$. This step requires a linear time w.r.t the size of E using Bubenzer's algorithm [17], since the automaton $Id(\mathcal{T}_{\mathrm{E}})$ is acyclic and deterministic.

In the following, a state $(q, \mathrm{N}(q))$ in the automaton $Id(\mathcal{T}_{\mathrm{E}})$ is called a *position state*, if there exists $a \in A_{\mathrm{E}}$ and $(q', \mathrm{N}(q')) \in Q'$ such that $\delta'((q', \mathrm{N}(q')), a) = (q, N(q))$. The state $(0, N(0))$ is also considered as a *position state*.

It is obvious to see that each *position state* is associated with a unique position in $\mathrm{pos}(\mathrm{E}) \cup \{0\}$ and then it's can be associated with a c-continuation.

Let $(q, N(q))$ and $(q', N(q'))$ two position states associated with the positions x and x' in pos(E). One can extend the $\equiv_e$ relation over *position states* in $Id(\mathcal{T}_{\mathrm{E}})$ as follows:

$$(q, N(q)) \equiv_e (q', N(q')) \Longleftrightarrow h(c_x(\mathrm{E})) \equiv h(c_{x'}(\mathrm{E}))$$

In the next, we will prove that the computation of the equivalence classes $C_{\equiv_e}$ can be performed in a linear time w.r.t. the size of the regular expression over $Id(\mathcal{T}_{\mathrm{E}})$ using the notion of *pseudo-continuations*.

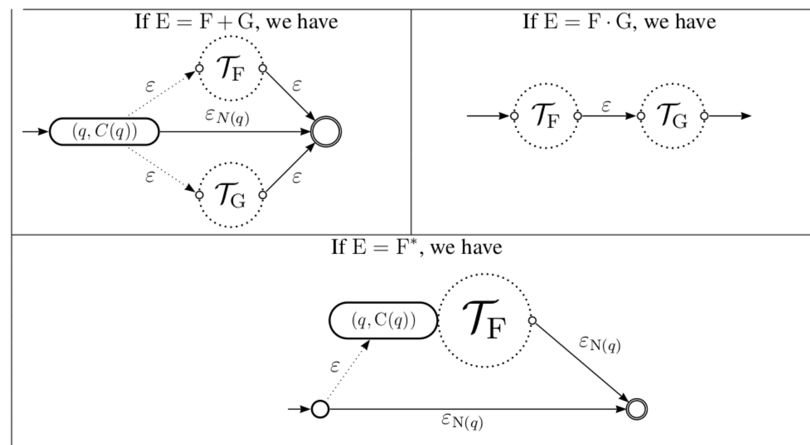
For abbreviation, a state $(q, N(q))$ in $Id(\mathcal{T}_{\mathrm{E}})$ will be denoted by q. We denote by $\mathrm{C}(q)$ *the pseudo-continuation* associated with the *position state* q which is an implicit representation of its c-continuation $c_x(\mathrm{E})$, where x is the position letter of q. We will show that the computation of the equivalence classes $C_{\equiv_e}$ turns on the computation of *pseudo-continuation* $C(q)$ over a particular ε-transitions marking of the automaton $Id(\mathcal{T}_{\mathrm{E}})$.

Definition 3. *The pseudo-continuation* $\mathrm{C}(q)$ *associated with a position state* $(q, \mathrm{N}(q)) \in Q'$ *in* $Id(\mathcal{T}_{\mathrm{E}})$ *is recursively defined by:*

$$\mathrm{C}(q) \quad = \quad \begin{cases} \mathrm{N}(q) & \text{if } \delta'(q, \varepsilon_{l+}) = q' \text{ and } \delta'(q, \varepsilon_{r+}) = q'', \\ \mathrm{N}(q') \cdot \mathrm{C}(q'') & \text{if } \delta'(q, \varepsilon_1^*) = q'' \text{ and } \delta'(q', a) = q \text{ for some } a \in A_{\mathrm{E}}, \\ \mathrm{N}(q') \cdot \mathrm{C}(q'') & \text{if } \delta'(q, \varepsilon_1^*) = q' \text{ and } \delta'(q, \varepsilon_2^*) = q'', \\ a \cdot \mathrm{C}(q') & \text{if } \delta'(q, a) = q', \text{ for all } a \in A_{\mathrm{E}}, \\ \mathrm{C}(q') & \text{if } \delta'(q, \varepsilon) = q'. \end{cases} \tag{5}$$

In order to compute efficiently the set of pseudo-continuations associated with the position states in $Id(\mathcal{T}_{\mathrm{E}})$, we define an acyclic deterministic sub-automaton $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}})) = \langle Q_{\equiv_e}, A_{\mathrm{E}}, I, \delta'', F \rangle$ of the Thompson automaton as follows:

- $A = \{\varepsilon_0, \cdots, \varepsilon_{|\,\mathrm{Exp}\,|}\} \cup A_{\mathrm{E}}$.
- $Q_{\equiv_e} = \{(q, \mathrm{C}(q)) |\ (q, \mathrm{N}(q)) \in Q'\}$, i.e., a state in $Id(\mathcal{T}_{\mathrm{E}})$ is replaced by $(q, C(q))$.
- The transition function δ'' is defined as follows:



Let us define the equivalence relation $\approx$ over the *position states* of $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$ as follows:

$$(q, \mathrm{C}(q)) \approx (q', \mathrm{C}(q')) \Leftrightarrow \mathrm{C}(q) \equiv \mathrm{C}(q') \tag{6}$$

Thus we have:

$$[q]_{\approx} = [q']_{\approx} \Leftrightarrow \mathrm{C}(q) \equiv \mathrm{C}(q') \tag{7}$$

Let $\overline{h}$ be the application that maps a letter ε_i to the letter i. By construction, the following proposition holds.

Proposition 6. *Let* $(q, \mathrm{C}(q))$ *be a position state in* $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$. *We have:*

$$\overline{h}(\overrightarrow{L}_q(C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}})))) = C(q)$$

Example 6. *Let us consider the Thompson automaton* $\mathcal{T}_{\mathrm{E}}$ *defined in previous examples. Figure 7 schematizes the derived automaton from* $Id(\mathcal{T}_{\mathrm{E}})$ *after pseudo continuations computation for position states.*

Notice that dotted ε-transitions are temporarily disabled and dashed ones are temporarily added.

The position states in the automaton $Id(\mathcal{T}_{\mathrm{E}})$ *are* $\{0, 5, 8, 11, 16\}$. *According to the definition of a pseudo continuation (see Formula (7)), the pseudo-continuations associated with position states are computed over the automaton* $Id(\mathcal{T}_{\mathrm{E}})$ *as follows:*

$$\mathrm{C}(5) = \mathrm{C}(8) = \mathrm{C}(11) = 10 \cdot 1$$
$$\mathrm{C}(16) = 15 \cdot 1$$

On the other hand, we have:

$$\overrightarrow{L}_5(C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))) = \overrightarrow{L}_8(C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))) = \overrightarrow{L}_{11}(C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))) = \varepsilon_{10} \cdot \varepsilon_1$$
$$\overrightarrow{L}_{16}(C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))) = \varepsilon_{15} \cdot \varepsilon_1$$

The following proposition is fundamental to prove that the equivalence relation $\equiv_e$ using the notion of c-continuation is the same when using pseudo-continuations $\mathrm{C}(q)$.

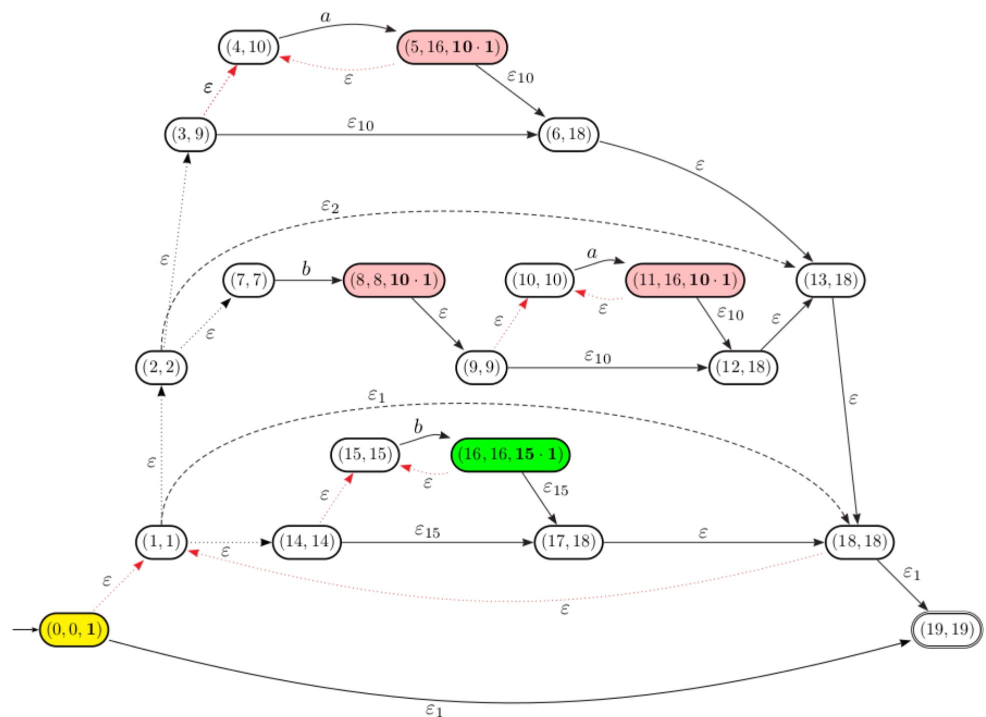


Figure 7. Pseudo-continuations computation for position states in $Id(\mathcal{T}_{\mathrm{E}})$.

Proposition 7. *Let* q *(resp.* q'*) be a position state associated with a position* $x \in \mathrm{pos(E)}$ *(resp.* $x' \in \mathrm{pos(E)}$*) in* $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$. *One has:*

$$\mathrm{C}(q) \equiv \mathrm{C}(q') \Leftrightarrow h(c_x(\mathrm{E})) \equiv h(c_{x'}(\mathrm{E}))$$

As a consequence, the following proposition holds.

Proposition 8. *Let q and q' be two position states in $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$. One has:*

$$q \equiv_e q' \Leftrightarrow \mathrm{C}(q) \equiv \mathrm{C}(q')$$

Theorem 5. *Let E be a regular expression and $\mathcal{T}_{\mathrm{E}}$ the associated Thompson automaton. The relation $\equiv_e$ can be computed over $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$ in $O(|\,\mathrm{E}\,|)$ time.*

Proof. From Proposition 8, one can deduce that the computation of the equivalence relation $\equiv_e$, turn to apply the Myhill-Nerode relation on the states of the automaton $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$. By definition, this last is acyclic and deterministic. Then, its minimization using Bubenzer's algorithm [17] requires $O(|\,\mathrm{E}\,|)$ time and space complexity. □

Example 7 (Continues)**.** *The automaton $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$, schematized in the Figure 8, is obtained from $Id(\mathcal{T}_{\mathrm{E}})$ after merging $\equiv_e$-equivalent position states* 5, 8 *and* 11.

The next step of our approach consists of the transitions and final states computation of the equation automaton $\mathcal{E}_{\mathrm{E}}$ using epsilon removal operation, denoted by $rmeps()$, over the automaton $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$.

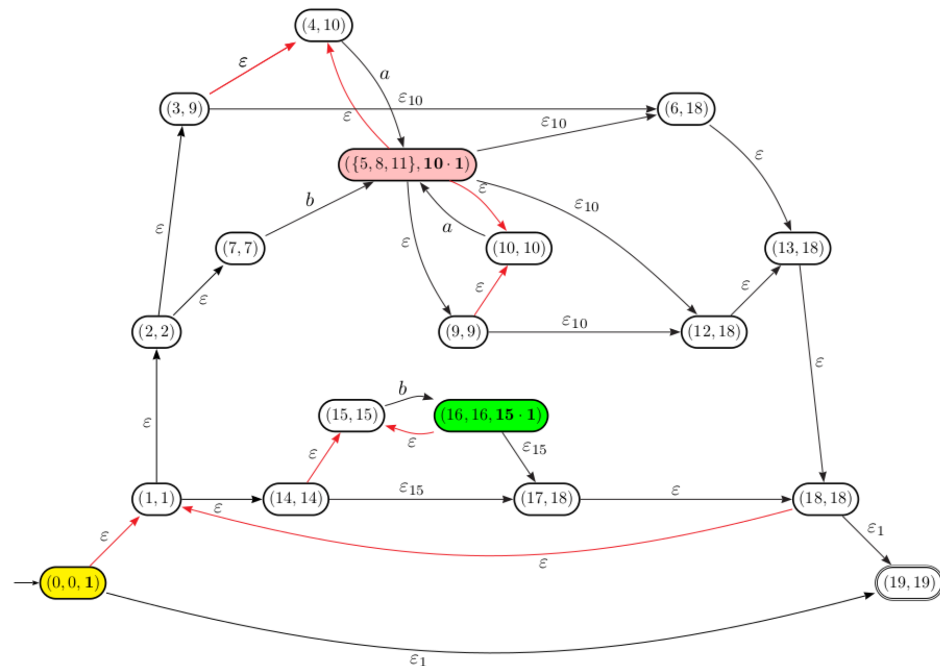


Figure 8. The automaton $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$ associated with $\mathrm{E} = (a^* + ba^* + b^*)^*$.

5.2. Computation of Transitions and Final States

After merging $\equiv_e$-equivalent states in the previous step, we obtain a reduced automaton $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$ having the same set of states as the equation automaton. To compute the transition function, we first enable the cyclic transitions previously disabled in the case when $\mathrm{E} = \mathrm{F}^*$ on $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$.

Let $Q_{\equiv_e}$ be the set of states of $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$. Recall that epsilon removal operation is denoted by, $rmeps(p)$ for a state $p \in Q$ and $rmeps(\mathcal{A})$ denotes the resulted automaton after removing marked and non-marked ε-transitions from the automaton $\mathcal{A}$.

As a consequence of Lemma 5 from [8], the following Lemma yeilds.

Lemma 2. *(Lemma 5 in [8]). Let q and q' be two position states in $Q_{\equiv_e}$ associated respectively with the positions x and x' in $\mathrm{pos}(\mathrm{E}) \cup \{0\}$, we have:*

$$q \in rmeps(q') \text{ iff } h(c_x(\mathrm{E})) \in \partial_a(h(c_{x'}(\mathrm{E}))), \text{ for some } a \in A_{\mathrm{E}}.$$

The set of destination states of the outgoing transitions from a state $q \in Q_{\equiv_e}$ is then equal to

$$\{p \in Q_{\equiv_e} \mid p \in rmeps(q) \text{ and } p \text{ is a position state}\}$$

Lemma 3. *Let q be a position state in $Q_{\equiv_e}$ associated with a position x in $\mathrm{pos}(\mathrm{E}) \cup \{0\}$, we have:*

$$F \in rmeps(q) \textit{ iff } \lambda(c_x(\mathrm{E})) = 1.$$

Proposition 9. *We have $\mathcal{E}_{\mathrm{E}} = rmeps(C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}})))$*

Example 8. *(Continues). Let us consider the automaton $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$ of the Example 7. The final states and the transitions of the equation automaton are computed over $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$ using epsilon removal operation rmeps as follows:*

- *The set of states of the equation automaton are $\{0, \{5, 8, 11\}, 16\}$.*
- *Since the final state of $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$ is the state 19 and $19 \in rmeps(0)$, $19 \in rmeps(\{5, 8, 11\})$, and $19 \in rmeps(16)$, then the set of final states are $\{0, \{5, 8, 11\}, 16\}$.*
- *There are two paths in $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$ from the state 0 to the state $\{5, 8, 11\}$ labeled respectively by $\varepsilon \cdot \varepsilon \cdot \varepsilon \cdot \varepsilon \cdot a$ and $\varepsilon \cdot \varepsilon \cdot \varepsilon \cdot b$, then $\{5, 8, 11\} \in rmeps(0)$. Consequently, two transitions $(0, a, \{5, 8, 11\})$ and $(0, b, \{5, 8, 11\})$ are added to the equation automaton. The same process will be applied for other transitions.*

Since there are $O(|\,\mathrm{E}\,|)$ states in $C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}}))$ and the operation $rmeps(C_{\equiv_e}(Id(\mathcal{T}_{\mathrm{E}})))$ is performed on exactly $|Q_{\equiv_e}|$ states, the following theorem holds.

Theorem 6. *Let E be a regular expression. The equation automaton of E can be computed in $O(|\,\mathrm{E}\,| \cdot |Q_{\equiv_e}|)$.*

6. Conclusions

In this paper, we presented a fast and sophisticated construction of the equation automaton from a regular expression over its associated Thompson automaton. The time complexity of our algorithm is at least as favorable as that of the best previously known algorithm. It is based on the minimization of acyclic deterministic finite automata and epsilon removal operations. This allowed us a construction of the equation automaton in $O(|\,\mathrm{E}\,| \cdot |Q_{\equiv_e}|)$ time and space complexity where $|Q_{\equiv_e}|$ denotes the number of transitions of the produced automaton. The implementation of the proposed algorithm is available under the following repository: https://github.com/FaissalOuardi/Equation-automaton (accessed on 27 May 2021).

Author Contributions: Conceptualization, F.O., Z.L. and B.E.; methodology, F.O., Z.L. and B.E.; validation, F.O., Z.L. and B.E.; formal analysis, F.O., Z.L. and B.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Source code can be found under the following link: https://github.com/FaissalOuardi/Equation-automaton (accessed on 27 May 2021).

Acknowledgments: We wish to thank the referees for the care they put into reading the previous versions of this manuscript. Their comments were invaluable in depth and detail, and the current version owes much to their efforts.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mirkin, B.G. Novyj algoritm postroéniá bazisa v ázyké régularnyh vyražénij. Izvéstiá Akadémii Nauk SSSR. Engineering cybernetics, no. 5 (1966). English translation of the preceding: Brzozowski, J. An algorithm for constructing a base in a language of regular expressions. pp. 110–116. *J. Symb. Log.* **1971**, *36*, 694.
2. Antimirov, V. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.* **1996**, *155*, 291–319. [CrossRef]
3. Glushkov, V.M. The abstract theory of automata. *Russ. Math. Surv.* **1961**, *16*, 1–53. Available online: https://iopscience.iop.org/article/10.1070/RM1961v016n05ABEH004112 (accessed on 27 May 2021). [CrossRef]
4. McNaughton, R.F.; Yamada, H. Regular expressions and state graphs for automata. *IEEE Trans. Electron. Comput.* **1960**, *9*, 39–57. [CrossRef]
5. Ziadi, D.; Ponty, J.-L.; Champarnaud, J.-M. A New Quadratic Algorithm to Convert a Regular Expression into an Automaton. In Proceedings of the Workshop on Implementing Automata, London, ON, Canada, 29–31 August 1996; pp. 109–119.
6. Champarnaud, J.-M.; Ziadi, D. Canonical derivatives, partial derivatives and finite automaton constructions. *Theor. Comput. Sci.* **2002**, *289*, 137–163. [CrossRef]
7. Khorsi, A.; Ouardi, F.; Ziadi, D. Fast equation automaton computation. *J. Discret. Algorithms* **2008**, *6*, 433–448. [CrossRef]
8. Allauzen, C.; Mohri, M. A Unified Construction of the Glushkov, Follow, and Antimirov Automata. In Proceedings of the International Conference of Mathematical Foundations of Computer Science, Stará Lesná, Slovakia, 28 August–1 September 2006; pp. 110–121.
9. Ilie, L.; Yu, S. Follow automata. *Inf. Comput.* **2003**, *186*, 140–162. [CrossRef]
10. Champarnaud, J.-M.; Nicart, F.; Ziadi, D. From the ZPC Structure of a Regular Expression to its Follow Automaton. *IJAC* **2006**, *16*, 17–34. [CrossRef]
11. Kleene, S. *Representation of Events in Nerve Nets and Finite Automata*; Automata Studies, Ann. Math. Studies 34; Princeton University Press: Princeton, NJ, USA, 1956; pp. 3–41.
12. Thompson, K. Regular Expression Search Algorithm. *Commun. ACM* **1968**, *11*, 410–422. [CrossRef]
13. Hopcroft, J. *An n log n Algorithm for Minimizing States in a Finite Automaton*; Technical Report; Stanford University, CS Dept.: Stanford, CA, USA, 1971.
14. Hopcroft, J.E.; Ullman, J.D. *Introduction to Automata Theory, Languages and Computation*; Addison-Wesley: Reading, MA, USA, 1979.
15. Sakarovitch, J.; Thomas, R. *Elements of Automata Theory*; Cambridge University Press: Cambridge, UK, 2009.
16. Brüggemann-Klein, A. Regular expressions into finite automata. *Theor. Comp. Sci.* **1993**, *120*, 117–126. [CrossRef]
17. Bubenzer, J. Cycle-aware minimization of acyclic deterministic finite-state automata. *J. Discret. Appl. Math.* **2014**, *163*, 238–246. [CrossRef]
18. Revuz, D. Minimization of acyclic deterministic automata in linear time. *Theor. Comput. Sci.* **1992**, *92*, 181–189. [CrossRef]
19. Giammarresi, D.; Ponty, J.-L.; Wood, D.; Ziadi, D. A characterization of Thompson digraphs. *Discret. Appl. Math.* **2004**, *134*, 317–337. [CrossRef]
20. Myhill, J. Finite automata and the representation of events. In *WADD TR-57-624*; Wright Patterson AFB: Dayton, OH, USA, 1957; pp. 112–137.
21. Nerode, A. Linear automata transformation. *Proc. AMS* **1958**, *9*, 541–544. [CrossRef]