

Follow Automaton versus Equation Automaton

J.-M. Champarnaud, F. Ouardi*, D. Ziadi

LIFAR, University of Rouen, 76821 Mont-Saint-Aignan – France
{jmc,fouardi,ziadi}@dir.univ-rouen.fr

Abstract

There exist two well-known quotients of the position automaton of a regular expression. The first one, called the equation automaton, has first been introduced by Mirkin from the notion of prebase and has been redefined by Antimirov from the notion of partial derivative. The second one, due to Ilie and Yu and called the follow automaton, can be obtained by ε -elimination in an ε -NFA that is always smaller than the classical ε -NFAs (Thompson, Sippu and Soisalon-Soininen). Ilie and Yu discuss of the difficulty to succeed in a theoretical comparison between the size of the follow automaton and the size of the equation automaton and conclude that it is very likely necessary to realize experimental studies. In this paper we solve the theoretical question, by first defining a set of regular expressions, called normalized expressions and such that every regular expression can be normalized in linear time, and proving then that the equation automaton of a normalized expression is always smaller than its follow automaton.

1 Introduction

For the last fifty years, the conversion of regular expressions into automata has raised the interest of computer scientists. In 1960-61, Glushkov [6] and McNaughton and Yamada [9] have defined the position automaton of a regular expression. In 1964, Mirkin [10] has introduced the notion of prebase, described a construction of the equation automaton and proved that this automaton is smaller in number of states than the position automaton.

Since the nineties, researchers have been interested by designing efficient algorithms for the computation of these automata. First of all, there exist

*Corresponding author. E-mail address: fouardi@dir.univ-rouen.fr

three well-known algorithms for computing the position automaton. The first one, due to Brüggemann-Klein [2] makes use of the notion of Star Normal Form of a regular expression. The second one is due to Chang and Paige [5] and it is based on a lazy computation technique. The third one has been designed by Champarnaud, Ziadi and Ponty [12] and it is built on the so-called ZPC-structure. The complexity of these three algorithms is quadratic w.r.t. the size of the regular expression. Concerning the equation automaton, there are two main algorithms. Antimirov's algorithm [1] is based on the computation of the set of partial derivatives of the expression, whereas the algorithm designed by Champarnaud and Ziadi [4] computes the equation automaton as the quotient of the c-continuation automaton by an equivalence relation that will be denoted by \equiv_e .

Recently, Ilie and Yu [8] have introduced a new automaton called the follow automaton and shown that it is a quotient of the position automaton by an equivalence relation denoted by \equiv_f . They have produced families of expressions for which the follow automaton has a smaller number of states than the equation automaton and other families for which it is the contrary. They finally conjecture that the only way to decide which automaton is better is probably by testing them in real-life applications. In this paper, we solve this challenging question. Indeed, we show that any regular expression can be reduced in a normalized form in linear time, and that the equation automaton of a normalized expression has a smaller number of states than the follow automaton.

The paper is organized as follows. Section 2 presents terminology, recalls the definition of the the three above-mentioned automata as well as the computation of the c-continuations of a regular expression, which turns out to be a useful tool for proving our results. In Section 3, we first define the set of normalized expressions and then we show that for a normalized expression the equivalence relation \equiv_e is coarser than the equivalence relation \equiv_f .

2 Preliminaries

In this section we recall some basic definitions and properties about regular expressions and finite automata. For more details, we refer to [7, 11].

Regular expressions and automata Let Σ be a non-empty finite set of symbols, called an *alphabet*. The set of all the words over Σ is denoted by Σ^* . The *empty word* is denoted by ε . A *language* over Σ is a subset of Σ^* . *Regular expressions* over an alphabet Σ and *regular languages* that they

denote are inductively defined as follows:

- (1) 0 is a regular expression denoting the language $L(0) = \emptyset$.
 - (2) 1 is a regular expression denoting the language $L(1) = \{\varepsilon\}$.
 - (3) a , for all $a \in \Sigma$, is a regular expression denoting the language $L(a) = \{a\}$.
- Let F (resp. G) be a regular expression denoting the language $L(F)$ (resp. $L(G)$). Then we have:
- (4) $(F + G)$ is a regular expression denoting the language $L(F + G) = L(F) \cup L(G)$.
 - (5) $(F \cdot G)$ is a regular expression denoting the language $L(F \cdot G) = L(F)L(G)$.
 - (6) (F^*) is a regular expression denoting the language $L(F^*) = (L(F))^*$.
- We write $Sym(E) = "+"$ (resp. $."$, $"^*"$) if $E = F + G$ (resp. $E = FG$, $E = F^*$). The following identities are classically used: $0 + E = E = E + 0$, $1 \cdot E = E = E \cdot 1$, $0 \cdot E = 0 = E \cdot 0$. We write: $F \equiv G$ if two regular expressions are identical (following Mirkin [10], E and F "graphically coincide").

Let E be a regular expression over an alphabet Σ . We call *alphabetic width* of E , denoted by $\|E\|$, the number of occurrences of symbols of Σ in E whereas we call *size* of E , denoted by $|E|$, the number of nodes of the syntax tree of E . The alphabetic width of the expression $(a + b)^* \cdot a \cdot b \cdot a + 1$ is equal to 5; its size is equal to 12.

In order to specify their position in the expression, symbols are subscripted following the order of reading. For example, starting from $E = (a + b)^*aba + 1$, one obtains the *linearized version* $\bar{E} = (a_1 + b_2)^*a_3b_4a_5 + 1$ of E . The set of positions for an expression E is denoted by Pos_E . For the previous example, we have $Pos_E = \{a_1, b_2, a_3, b_4, a_5\}$. If F is a subexpression of E , we denote by $Pos_E(F)$ the subset of positions of E that are symbols of F . We denote by h the application that maps each position in Pos_E to the symbol of Σ that appears at this position in E . For $E = (a + b)^*aba + 1$, we have $h(a_1) = h(a_3) = h(a_5) = a$ and $h(b_2) = h(b_4) = b$. We use the following definition: $\lambda(E) = \text{if } \varepsilon \in L(E) \text{ then } 1 \text{ else } 0$.

A *finite automaton* is a quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is the set of states, Σ is the alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \rightarrow 2^Q$ is the transition mapping. The language recognized by \mathcal{A} is denoted $L(\mathcal{A})$. The *size* of a finite automaton \mathcal{A} is the number of its states.

Position automaton In order to construct a non-deterministic finite automaton recognizing $L(E)$, Glushkov [6] has introduced three position sets: $First(E)$ is the set of initial positions of words of $L(\bar{E})$, $Last(E)$ is the set of final positions of words of $L(\bar{E})$, and $Follow(E, x)$ is the set of positions that immediately follow the position x in some word of $L(\bar{E})$.

Formally $First$ is defined by induction according to the following rules:

$$\begin{aligned}
First(0) &= First(1) = \emptyset, \\
First(x) &= \{x\}, \\
First(F + G) &= First(F) \cup First(G), \\
First(F \cdot G) &= First(F) \cup \lambda(F) First(G), \\
First(F^*) &= First(F).
\end{aligned}$$

In order to obtain rules for $Last(E)$, just substitute $First$ by $Last$ and replace the last but one rule by $Last(F \cdot G) = Last(G) \cup \lambda(G) Last(F)$.

The set $Follow(E, x)$ can be inductively computed as follows:

$$\begin{aligned}
Follow(0, x) &= Follow(1, x) = \emptyset, \\
Follow(F + G, x) &= \begin{cases} Follow(F, x) & \text{if } x \in Pos_E(F), \\ Follow(G, x) & \text{otherwise,} \end{cases} \\
Follow(F \cdot G, x) &= \begin{cases} Follow(F, x) & \text{if } x \in Pos_E(F) \setminus Last(F), \\ Follow(F, x) \cup First(G) & \text{if } x \in Last(F), \\ Follow(G, x) & \text{if } x \in Pos_E(G), \end{cases} \\
Follow(F^*, x) &= \begin{cases} Follow(F, x) & \text{if } x \in Pos_E(F) \setminus Last(F), \\ Follow(F, x) \cup First(F) & \text{if } x \in Last(F). \end{cases}
\end{aligned}$$

We add a specific position, denoted by 0 (where $0 \notin Pos_E$) to the set Pos_E and we set $Pos_0(E) = Pos_E \cup \{0\}$; the set $Last_0(E)$ is equal to $Last(E)$ if $\lambda(E) = 0$ and to $Last(E) \cup \{0\}$ otherwise; the set $Follow_0(E, x)$ is equal to $Follow(E, x)$ if $x \in Pos_E$ and to $First(E)$ if $x = 0$. The position automaton of E [6, 9], denoted by \mathcal{P}_E , whose states are $Pos_0(E)$, and that recognizes $L(E)$, is derived from the above positions sets as follows.

Definition 1. *The position automaton of a regular expression E is the automaton $\mathcal{P}_E = (Pos_0(E), \Sigma, \delta, \{0\}, Last_0(E))$ such that:*

$$\delta(x, a) = \{y \mid y \in Follow_0(E, x) \text{ and } h(y) = a\}, \forall x \in Pos_0(E), \forall a \in \Sigma.$$

Follow automaton Recently, Ilie and Yu [8] introduced the *follow automaton* $\mathcal{A}_f(E)$ of a regular expression E . This automaton is smaller than the position automaton and, in average, faster to compute. The follow automaton $\mathcal{A}_f(E)$ can be computed in quadratic time [8, 3]. Let us set $x \equiv_f y \Leftrightarrow (Follow_0(E, x) = Follow_0(E, y) \text{ and } x \in Last_0(E) \Leftrightarrow y \in Last_0(E))$. The follow automaton is a quotient of the position automaton [8]. Indeed we have $\mathcal{A}_f(E) \simeq \mathcal{P}_E / \equiv_f$.

Equation automaton The equation automaton has been defined by Mirkin [10] and by Antimirov [1]. Champarnaud and Ziadi have introduced the notion of c-continuation and proved that the equation automaton is a quotient of the c-continuation automaton [4]. In this section, we recall the notions of c-derivative, c-continuation and c-continuation automaton.

Definition 2 (c-derivative w.r.t. a symbol). *The c-derivative of a regular expression E w.r.t. a symbol a , written $d_a(E)$, is defined by*

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

The extension to a word $u = u_1 \dots u_n$ follows the equations: $d_\varepsilon(E) = E$ and $d_{u_1 \dots u_n}(E) = d_{u_2 \dots u_n}(d_{u_1}(E))$.

Theorem 1. [4] *If E is linear, for every symbol a and every word u , the c-derivative $d_{ua}(E)$ of E w.r.t. the word ua is either 0 or unique.*

Theorem 1 allows us to define the c-continuation $c_a(E)$ of a in E , that is the unique value of the non-null c-derivatives $d_{(ua)}(E)$.

Proposition 1. [4] *For every symbol a of a linear expression E , the c-continuation $c_a(E)$ is such that:*

$$\begin{aligned} c_a(a) &= 1, \\ c_a(F + G) &= \text{if } c_a(F) \neq 0 \text{ then } c_a(F) \text{ else } c_a(G), \\ c_a(F \cdot G) &= \text{if } c_a(F) \neq 0 \text{ then } c_a(F) \cdot G \text{ else } \lambda(F) \cdot c_a(G), \\ c_a(F^*) &= c_a(F) \cdot F^*. \end{aligned}$$

Corollary 1. [4] *For every symbol a of a linear expression E , the c-continuation $c_a(E)$ is either 1 or a subexpression of E or a product of subexpressions.*

More precisely, $c_a(E) = H_1 \dots H_n$, where H_i is a subexpression of E , for all $1 \leq i \leq n$. This decomposition is fundamental for the main theorem of this paper. Let E be a regular expression over Σ , \overline{E} the linearized version of E over Pos_E and h the mapping from Pos_E onto Σ . We assume that 0 is a symbol not in Pos_E . Let $c_0(\overline{E}) = d_\varepsilon(\overline{E}) = \overline{E}$.

Definition 3 (C-continuation automaton). *The c-continuation automaton of E , $\mathcal{C}_E = (Q, \Sigma, i, T, \delta)$, is defined by:*

- $Q = \{(x, c_x(\overline{E})) \mid x \in Pos_0(E)\}$,
 - $i = (0, c_0(\overline{E}))$,
 - $T = \{(x, c_x(\overline{E})) \mid \lambda(c_x(\overline{E})) = 1\}$,
 - $\delta((x, c_x(\overline{E})), a) = \{(y, c_y(\overline{E})) \mid h(y) = a \text{ and } d_y(c_x(\overline{E})) \equiv c_y(\overline{E})\}$,
- $\forall x \in Pos_0(E) \text{ and } \forall a \in \Sigma$.

The position and c-continuation automata are isomorphic. The relation that exists between the *First*, *Last* and *Follow* sets and the c-continuations is enlightened by the following proposition, which will be useful in the sequel.

Proposition 2. [4] *The following equalities hold:*

1. $First(E) = \{y \in Pos_E \mid d_y(\overline{E}) \neq 0\}$;
2. $Last(E) = \{y \in Pos_E \mid \lambda(c_y(\overline{E})) = 1\}$;
3. $Follow(E, x) = \{y \in Pos_E \mid d_y(c_x(\overline{E})) \neq 0\}$.

Let us set $(x, c_x(\overline{E})) \equiv_e (y, c_y(\overline{E})) \Leftrightarrow h(c_x(\overline{E})) \equiv h(c_y(\overline{E}))$. The equation automaton is a quotient of the c-continuation automaton [4]. Indeed we have $\mathcal{E}_E \simeq \mathcal{C}_E / \equiv_e$.

3 Follow automaton versus equation automaton

In this section, we show that the equation automaton associated to a *normalized* regular expression has always less states than the follow automaton.

3.1 Normalized expressions

Definition 4. *A regular expression E is said to be normalized if the following conditions hold:*

- 1) *The expression E is a reduced one according to:*
 - the identities: $0 + E = E = E + 0$, $1 \cdot E = E = E \cdot 1$, $0 \cdot E = 0 = E \cdot 0$,
 - the rule: for all subexpressions H of E , $H = F + 1 \Rightarrow \lambda(F) = 0$.
- 2) *The operation “.” is left associative i.e. $H = F \cdot G \Rightarrow Sym(G) \neq \text{“.”}$.*
- 3) *The expression E is in SNF i.e. for every expression F such that F^* is a subexpression of E , it holds: $\forall x \in Last(F)$, $Follow(F, x) \cap First(F) = \emptyset$.*

It was shown in [3] that for a regular expression E , it is possible to construct an equivalent normalized regular expression in $O(|E|)$ time and space. In the following, we consider only linear regular expressions.

Proposition 3. *Let x be a position of a linear regular expression E . Then one has $\left. \begin{array}{l} x \in First(E) \\ \lambda(c_x(E)) = 1 \end{array} \right\} \Rightarrow \lambda(E) = 1$.*

Proof. Proof is by induction on the alphabetic width of E . Let us first consider the case $\|E\| = 1$. Since E is normalized, its form is either $a + 1$ or a^* and it is easy then to prove that the proposition is true. We now suppose that the proposition is satisfied for normalized expressions F and G and we prove it is satisfied for normalized expressions $F + G$, $F \cdot G$ and F^* .

Case $E = (F + G)$: without loss of generality we suppose that $x \in Pos_E(F)$. In this case $c_x(E) = c_x(F)$. Then $\lambda(c_x(E)) = \lambda(c_x(F)) = 1$. Since $x \in First(E)$ one has $x \in First(F)$. By the inductive hypothesis $\lambda(F) = 1$. Consequently $\lambda(E) = 1$.

Case $E = (F \cdot G)$: if $x \in Pos_E(F)$, then $x \in First(F)$ and $c_x(F \cdot G) = c_x(F) \cdot G$. It implies that $\lambda(c_x(E)) = \lambda(c_x(F) \cdot G) = 1$. Thus $\lambda(c_x(F)) = 1$ and $\lambda(G) = 1$. Applying the inductive hypothesis on F we get $\lambda(F) = 1$. Thus $\lambda(E) = 1$.

If $x \in Pos_E(G)$, then $x \in First(G)$ and since $x \in First(E)$, $\lambda(F) = 1$. Since $c_x(F \cdot G) = c_x(G)$ and $\lambda(c_x(F \cdot G)) = 1$, it comes $\lambda(c_x(G)) = 1$. By induction hypothesis we have $\lambda(G) = 1$. Consequently $\lambda(E) = 1$.

Case $E = F^$:* it is obvious. ■

Theorem 2. *Let x and y be two positions of a normalized regular expression E . Then the following conditions are equivalent:*

- i) $c_x(E) \equiv c_y(E)$,*
- ii) $\forall a \in Pos_E, d_a(c_x(E)) \equiv d_a(c_y(E))$ and $\lambda(c_x(E)) = \lambda(c_y(E))$,*
- iii) $Follow(E, x) = Follow(E, y)$ and $[x \in Last(E) \Leftrightarrow y \in Last(E)]$.*

Proof. *i) \Rightarrow ii)* is obvious.

ii) \Leftrightarrow iii) straightforward consequence of Proposition 4.

ii) \Rightarrow i) we proceed by absurd. We suppose that $\forall a \in Pos_E, d_a(c_x(E)) \equiv d_a(c_y(E))$ and $[\lambda(c_x(E)) = \lambda(c_y(E))]$ and also that $c_x(E) \not\equiv c_y(E)$.

Since x and y are two different positions of E , there exists a subexpression $E_x \diamond E_y$ of E with $\diamond \in \{+, \cdot\}$ such that $x \in Pos_E(E_x)$ and $y \in Pos_E(E_y)$.

Let us suppose that $\diamond = "+"$. Since $c_x(E) \not\equiv c_y(E)$, one has $c_x(E) = A_1 \cdots A_\alpha \cdot C_1 \cdots C_m$ and $c_y(E) = B_1 \cdots B_\beta \cdot C_1 \cdots C_m$ with $\alpha + \beta \geq 1$, $Pos_E(A_i) \subseteq Pos_E(E_x)$ and $Pos_E(B_j) \subseteq Pos_E(E_y)$ for all $1 \leq i \leq \alpha$ and $1 \leq j \leq \beta$.

In the case where $C_1 \cdots C_m \neq 0$, we have three sub-cases: (a) $\alpha = 0$,

$\beta \geq 1$, (b) $\alpha \geq 1, \beta = 0$ and (c) $\alpha \geq 1, \beta \geq 1$. We limit ourselves to the sub-case (a); proof for (b) and (c) can be done in a similar way. In the sub-case (a), $c_x(E)$ and $c_y(E)$ have the form $c_x(E) = C_1 \cdots C_m$ and $c_y(E) = B_1 \cdots B_\beta \cdot C_1 \cdots C_m$.

Let $b \in First(B_1)$. By Proposition 2, we have $d_b(c_y(E)) \neq 0$. By *ii*) it implies that $d_b(c_x(E)) \neq 0$. Consequently we have $b \in First(C_1 \cdots C_m)$. Since we have $b \in Pos_E(E_y)$, there exists $k, 1 \leq k \leq m$ such that $C_k = F^*$ and $\lambda(C_1 \cdots C_k) = 1$ with $b \in First(F)$. It comes

$$First(B_1) \cap First(F) \neq \emptyset \quad (1)$$

In the other hand, since $c_y(E) = B_1 \cdots B_\beta \cdot C_1 \cdots C_m$, there exists a subexpression $A_y, A_y \cdot B_1$ or $A_y^* = B_1$ of E such that $y \in Pos_E(A_y)$ and A_y contains no occurrence of “.” nor of “*”. Consequently $y \in Last(A_y)$.

Since $x \in First(E_x)$ and $\lambda(C_1 \cdots C_k) = 1$ we have $x \in First(F)$, which implies by Proposition 2 that $d_x(c_x(E)) \neq 0$. By *ii*) one has $d_x(c_y(E)) \neq 0$. Since $x \notin Pos_E(E_y)$, we have $\lambda(B_1 \cdots B_\beta C_1 \cdots C_k) = 1$. Hence $y \in Last(F)$. Thus

$$Last(A_y) \cap Last(F) \neq \emptyset \quad (2)$$

Finally, for the subexpression $F^* = C_k$ of E , there exists $y \in Last(F)$ such that $b \in First(B_1) \subseteq First(F) \cap Follow(F, y)$. Hence a contradiction with E is in SNF.

In the case where $C_1 \cdots C_m = 0$, $c_x(E)$ and $c_y(E)$ have the form $c_x(E) = A_1 \cdots A_\alpha$ et $c_y(E) = B_1 \cdots B_\beta$ with $\alpha \geq 1$ et $\beta \geq 1$. Let $b \in First(B_1)$. By Proposition 2 we have $d_b(c_y(E)) \neq 0$. By *ii*) one has $d_b(c_x(E)) \neq 0$. It holds: $b \notin Pos_E(E_x) \Rightarrow b \notin Pos_E(A_i)$, for all $1 \leq i \leq \alpha$. Hence a contradiction.

Let us suppose that $\diamond = \text{“.”}$. Then $Sym(E_y) \neq \text{“.”}$ and $c_x(E)$ and $c_y(E)$ have the form

$$\begin{aligned} c_x(E) &= A_1 \cdots A_\alpha \cdot E_y \cdot C_1 \cdots C_m \\ c_y(E) &= \begin{cases} B_1 \cdots B_\beta \cdot E_y \cdot C_1 \cdots C_m & \text{where } E_y = F^* \\ \text{or} \\ B_1 \cdots B_\beta \cdot C_1 \cdots C_m & \text{where } E_y = F + G \end{cases} \end{aligned}$$

Let us discuss the case where $C_1 \cdots C_m = 0$. We have

$$\begin{aligned} c_x(E) &= A_1 \cdots A_\alpha \cdot E_y \\ c_y(E) &= \begin{cases} B_1 \cdots B_\beta \cdot E_y & \text{where } E_y = F^* \\ \text{or} \\ B_1 \cdots B_\beta & \text{where } E_y = F + G \end{cases} \end{aligned}$$

If $\alpha \neq 0$, let $a \in First(A_1)$. By Proposition 2 we have $d_a(c_x(E)) \neq 0$. By *ii*) it holds $d_a(c_y(E)) \neq 0$. One has $a \notin Pos_E(E_y)$. Hence a contradiction.

If $\alpha = 0$, then $c_x(E) = E_y$ and

$$c_y(E) = \begin{cases} B_1 \cdots B_\beta \cdot E_y & \text{where } E_y = F^* \\ \text{or} \\ B_1 \cdots B_\beta & \text{where } E_y = F + G \end{cases}$$

Case $c_y(E) = B_1 \cdots B_\beta \cdot E_y$. The demonstration is similar as in the sub-case (a) of the case where $\diamond = "+"$. There exists b , $b \in First(B_1) \subseteq Follow(F, y) \cap First(F)$ with $F^* = E_y$ and $y \in Last(F)$. Hence a contradiction with E is in SNF.

Case $c_y(E) = B_1 \cdots B_\beta$. We proceed in a similar way and we deduce that there exists k , $1 \leq k \leq \beta$ such that $B_k = F^*$ and $b \in First(B_1) \subseteq Follow(F, y) \cap First(F)$ and $y \in Last(F)$. Which is a contradiction with E is in SNF.

Let us discuss the case where $C_1 \cdots C_m \neq 0$. There are two cases. The first one is where $E_y = H^*$. Let $c_x(E) = A_1 \cdots A_\alpha \cdot E_y \cdot C_1 \cdots C_m$ and $c_y(E) = B_1 \cdots B_\beta \cdot E_y \cdot C_1 \cdots C_m$ with $\alpha + \beta \geq 1$. We consider two sub-cases: (d) $\alpha = 0$ and (e) $\alpha \geq 1$. Proof for (d) can be done in a similar way as in the case where $C_1 \cdots C_m = 0$. In the sub-case (e), by a similar reasoning, there exist k , $1 \leq k \leq \alpha$ such that $C_k = F^*$ and $a \in First(A_1) \subseteq Follow(F, x) \cap First(F)$ and $x \in Last(F)$. Which is a contradiction with E is in SNF.

The second one is when $E_y = F + G$. Let us suppose that $y \in Pos_E(F)$. We have two sub-cases:

Case $G \neq 1$: by a similar way, there exist k , $1 \leq k \leq \alpha$ such that $C_k = F^*$, $t \in First(E_y) \subseteq Follow(F, z) \cap First(F)$ and $z \in Last(F) \cap Last(E_x)$. Hence a contradiction with E is in SNF.

Case $G = 1$: let $a \in First(A_1)$. By Proposition 2 we have $d_a(c_x(E)) \neq 0$. By *ii*) one has $d_a(c_y(E)) \neq 0$. Thus, there exists k , $1 \leq k \leq m$ such that $\lambda(B_1 \cdots B_\beta C_1 \cdots C_k) = 1$ and $C_k = H^*$ with $a \in First(H)$. Hence $y \in First(H)$. Since $\lambda(B_1 \cdots B_\beta C_1 \cdots C_k) = 1$, we have $\lambda(B_1 \cdots B_\beta) = 1$. One has $c_y(F) = B_1 \cdots B_\beta$ and $\lambda(c_y(F)) = \lambda(B_1 \cdots B_\beta) = 1$, which implies that $y \in First(F)$. By Proposition 3 one has $\lambda(F) = 1$. Which is a contradiction with the Definition 4-1 for the subexpression $E_y = F + 1$ of E .

■

The equation automaton is a quotient of the position automaton by the equivalence relation $x \equiv_e y \Leftrightarrow h(C_x(E)) \equiv h(C_y(E))$. By Theorem 2, for a normalized expression it holds that $x \equiv_f y \Leftrightarrow C_x \equiv C_y$. Then we conclude that $x \equiv_f y$ implies that $x \equiv_e y$. Hence the theorem:

Theorem 3. *The number of states in the equation automaton of a normalized regular expression is smaller than in its follow automaton. The same property holds for the number of transitions.*

References

- [1] Antimirov V., *Partial derivatives of regular expressions and finite automaton constructions*, Theoret. Comput. Sci., 155, 2917–319, 1996.
- [2] Brüggemann-Klein A., *Regular Expressions into Finite Automata*, Theoret. Comput. Sci., 120, 197–213, 1993.
- [3] Champarnaud J.-M., F. Nicart and D. Ziadi, *Computing the Follow Automaton of a regular Expression*, accepted in CIAA2004.
- [4] Champarnaud J.-M. and D. Ziadi, *Canonical derivatives, partial derivatives and finite automaton constructions*, Theoret. Comput. Sci., 289, 137-163, 2002.
- [5] Chang C.-H. and Paige R., *From Regular Expressions to DFA's Using Compressed NFA's*, Theoret. Comput. Sci., 178, 1–36, 1997.
- [6] Glushkov V.M., *The Abstract Theory of Automata*, Russian Math. Surveys, 16, 1–53, 1961.
- [7] Hopcroft J.E. and Ullman J.D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.
- [8] Ilie L. and Yu S., *Follow automata*, Information and computation, 186, 140-162, 2003.
- [9] McNaughton R., Yamada H., *Regular Expressions and State Graphs For Automata*, IEEE Trans. on Electronic Computers, 9-1, 39–47, 1960.
- [10] Mirkin B. G. *An algorithm for constructing a base in a language of regular expressions*, Engineering Cybernetics, 5:110–116, 1966.
- [11] S. Yu, *Regular languages*, in: G. Rozenberg, A. Salomaa, Handbook of Formal Languages, Vol. I, Springer-Verlag, Berlin, 41–110, 1997.
- [12] Ziadi D., Ponty J.-L. and Champarnaud J.-M., *Passage d'une expression rationnelle à un automate fini non-déterministe*, Journées Montoises (1995), Bull. Belg. Math. Soc. 4, 177-203, 1997.