Extended to Multi-Tilde-Bar Regular Expressions and Efficient Finite Automata Constructions

Jean-Marc Champarnaud\textsuperscript{a}, Faissal Ouardi\textsuperscript{b,}\textsuperscript{b}, Djelloul Ziadi\textsuperscript{a}

\textsuperscript{a}LITIS, University of Rouen, 76821 Saint-Etienne-du-Rouvray, France
\textsuperscript{b}Department of Computer Science, Faculty of Science, Mohammed V-Agdal University, Morocco

Abstract

Several algorithms have been designed to convert a regular expression into an equivalent finite automaton. One of the most popular constructions, due to Glushkov and to McNaughton and Yamada, is based on the computation of the Null, First, Last and Follow sets (called Glushkov functions) associated with a linearized version of the expression. Recently Mignot considered a family of extended expressions called Extended to multi-tilde-bar Regular Expressions (EmtbREs) and he showed that, under some restrictions, Glushkov functions can be defined for an EmtbRE. In this paper we present an algorithm which efficiently computes the Glushkov functions of an unrestricted EmtbRE. Our approach is based on a recursive definition of the language associated with an EmtbRE which enlightens the fact that the worst case time complexity of the conversion of an EmtbRE into an automaton is related to the worst case time complexity of the computation of the Null function. Finally we show how to extend the $ZPC$ -structure to EmtbREs, which allows us to apply to this family of extended expressions the efficient constructions based on this structure (in particular the construction of the c-continuation automaton, the position automaton, the follow automaton and the equation automaton).

Keywords: Regular Expressions and languages, Finite automata, Computation Complexity

1. Introduction

According to Kleene’s theorem \cite{16}, regular expressions and finite automata are two equivalent representations of regular languages. The conversion from a representation into the other one raised numerous research works. Concerning the
conversion of a regular expression into a finite automaton we can cite the following references: [1–5, 9–11, 13, 14, 17, 19], for which a common aim is to reduce the space and/or worst case time complexity of the result of the conversion. In this paper we are particularly interested by the implementation of conversion algorithms which are based on the notion of position, such as the five first ones in the above list. Following [13, 17], these algorithms are based on the the computation of the Null, First, Last and Follow sets (called Glushkov functions) associated with a linearized version of the expression. Recently Mignot [18] considered a family of extended expressions called Extended to multi-tilde-bar Regular Expressions (EmtbREs) and he showed that, under some restrictions, the Glushkov functions can be defined for an EmtbRE (see also [6, 7]). In this paper we present an algorithm which efficiently computes the Glushkov functions of an unrestricted EmtbRE. Our approach is based on a recursive definition of the language associated with an EmtbRE which enlightens the fact that worst case time complexity of the conversion of an EmtbRE into an automaton is related to the worst case time complexity of the computation of the Null function. Finally we show how to extend the \( ZPC \)-structure [19] to EmtbREs, which allows us to apply to this family of extended expressions the efficient constructions based on this structure (in particular the construction of the c-continuation automaton [10], the position automaton [19], the follow automaton [9] and the equation automaton [10, 15]).

The structure of the paper is as follows. In Section 2, we recall some basic definitions concerning regular expressions and finite automata, and we recall the notion of multi-tilde-bar expression. New properties concerning the language of a multi-tilde-bar expression are stated in Section 3. In Section 4, we give the definition of the position automaton associated with an arbitrary multi-tilde-bar expression. Section 5 is devoted to an efficient computation of the position automaton of an EmtbRE, through the extension of the notion of \( ZPC \)-structure of a regular expression.

2. Preliminaries

2.1. Regular expressions and finite automata

Let \( A \) be a non-empty finite set of symbols, called an alphabet. The set of all the words over \( A \) is denoted by \( A^* \). The empty word is denoted by \( \varepsilon \). A language over \( A \) is a subset of \( A^* \). Regular expressions over an alphabet \( A \) and regular languages that they denote are inductively defined as follows:

- \( \emptyset \) is a regular expression denoting the language \( L(\emptyset) = \emptyset \).
- \( x \), for all \( x \in A \cup \{ \varepsilon \} \), is a regular expression denoting the language \( L(x) = \{ x \} \).
Let \( F \) (resp. \( G \)) be a regular expression denoting the language \( L(F) \) (resp. \( L(G) \)); then we have:

- \((F + G)\) is a regular expression denoting the language \\
  \( L(F + G) = L(F) \cup L(G) \).
- \((F \cdot G)\) is a regular expression denoting the language \\
  \( L(F \cdot G) = L(F) \cdot L(G) \).
- \((F^*)\) is a regular expression denoting the language \( L(F^*) = (L(F))^* \).

The following identities are classically used:
\( \emptyset + E = E = E + \emptyset, \epsilon \cdot E = E = E \cdot \epsilon, \emptyset \cdot E = \emptyset = E \cdot \emptyset. \)

Let \( E \) be a regular expression. Its linearized form, denoted by \( E' \), is obtained by ranking every letter occurrence with a subindex denoting its position in \( E \). We say that a regular expression is in linear form if each letter of the expression occurs only once. Subscripted letters are called positions and the set of positions is denoted by \( \text{Pos}(E) \). We denote by \( h \) the application that maps each position in \( \text{Pos}(E) \) to the symbol of \( A \) that appears at this position in \( E \). The size of \( E \), denoted by \( |E| \), is the size of its syntactical tree. We call alphabetic width of \( E \), denoted by \( ||E|| \), the number of occurrences of letters in the expression.

**Definition 1.** Let \( E \) be a regular expression denoting the language \( L \). The set \( \text{Null}(E) \) is defined by:

\[
\text{Null}(E) = \begin{cases} 
\{\epsilon\} & \text{if } \epsilon \in L, \\
\emptyset & \text{otherwise.} 
\end{cases}
\]

A finite automaton (NFA) is a 5-tuple \( A = \langle Q, A, \delta, q_0, F \rangle \), 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 \( \delta : Q \times A \rightarrow 2^Q \) is the transition function. The language recognized by \( A \) is denoted by \( L(A) \).

2.2. Multi-tilde-bar expressions

We now recall the syntactical definition of extended to multi-tilde-bar regular expressions (EmtbREs) \[6\]. Notice that these expressions will be proven to be regular later (see Corollary \[1\]).

Let \( E \) be a regular expression. The language \( L(E) \setminus \{\varepsilon\} \) is denoted by the expression \( \overline{E} \) (bar operator) and the language \( L(E) \cup \{\varepsilon\} \) is denoted by the expression \( \overline{E} \) (tilde operator). Without loss of generality, any regular expression can be considered as a product of concatenation \( E_1 \cdot E_2 \cdots E_n \) of \( n \) subexpressions, with \( n \geq 1 \). Such a product is denoted by \( E_{1,n} \) and the set of its factors is denoted by \( F \).
Let us consider the set of pairs \( F' = \{(i, j) \mid 1 \leq i \leq j \leq n\} \). For \( 1 \leq i \leq j \leq n \), the factor \( E_{i,j} \) is represented by the pair \((i, j) \in F'\). A bar operator (resp. a tilde operator) applying on the factor \( E_{i,j} \) is also represented by the pair \((i, j) \in F'\).

Given two disjoint subsets \( F_1 \) and \( F_2 \) of \( F \), a multi-tilde-bar operator is defined by two subsets of \( F' \): the set \( B^n_1 \) of bar operators applying on the factors of \( F_1 \) and the set \( T^n_1 \) of tilde operators applying on the factors of \( F_2 \). Finally, a multi-tilde-bar expression \( E'_{1,n} \) is defined as a product \( E_{1,n} \) equipped with a set \( B^n_1 \) of bars and a set \( T^n_1 \) of tildes.

**Definition 2.** An Extended to multi-tilde-bar Regular Expression (EmtbRE) over an alphabet \( A \) is inductively defined by:

\[
\begin{align*}
E &= \emptyset, \\
E &= x, \text{ with } x \in A \cup \{\varepsilon\}, \\
E &= (F + G), \text{ with } F \text{ and } G \text{ two EmtbREs}, \\
E &= (F \cdot G), \text{ with } F \text{ and } G \text{ two EmtbREs}, \\
E &= (F^*), \text{ with } F \text{ an EmtbRE}, \\
E'_{1,n} \text{ is a EmtbRE with } B^n_1 \text{ define the set of Bar operators,} \\
T^n_1 \text{ define the set of Tilde operators,} \\
\text{and } E_{1,n} \text{ a concatenation product of EmtbREs.}
\end{align*}
\]

The EmtbRE \( E'_{i,j} \) is deduced from the expression \( E'_{1,n} \) by taking as set of bars the subset \( B^n_i = \{(k_1, k_2) \in B^n_1 \mid i \leq k_1 \leq k_2 \leq j\} \) of \( B^n_1 \) and as set of tildes the subset \( T^n_i = \{(k_1, k_2) \in T^n_1 \mid i \leq k_1 \leq k_2 \leq j\} \) of \( T^n_1 \). The size of \( E'_{1,n} \) denoted \( |E'_{1,n}| \) is the size of \( E_{1,n} \) added with the term \( |T^n_1| + |B^n_1| \). The alphabetic width \( ||E'_{1,n}|| \) of \( E'_{1,n} \) is the number of occurrences of letters in the expression.

**Example 1.** Consider the regular expression \( E_{1,5} = E_1 \cdot E_2 \cdot E_3 \cdot E_4 \cdot E_5 \). Let us consider the set of bars \( B^n_1 = \{(2, 3), (3, 5)\} \) and the set of tildes \( T^n_1 = \{(1, 2), (4, 5)\} \). The EmtbREs \( E'_{1,5} \) and \( E'_{1,3} \) can be represented graphically as follows:

\[
\begin{align*}
E'_{1,5} &= E_1 \cdot E_2 \cdot E_3 \cdot E_4 \cdot E_5 \\
E'_{1,3} &= E_1 \cdot E_2 \cdot E_3
\end{align*}
\]

3. The language of a multi-tilde-bar expression

The original semantical definition of the language of an EmtbRE \( [6] \) is based on the description of how words are generated by overlapping tildes and bars. Our approach is different: we provide a recursive definition of the language of an EmtbRE.

**Definition 3.** Let \( E'_{1,n} \) be a multi-tilde-bar expression. The language associated with \( E'_{1,n} \) is recursively defined as follows:
L(E_{i,j}) = \begin{cases} 
L \cup \{\varepsilon\} & \text{if } (i,j) \in T_1^n, \\
L \setminus \{\varepsilon\} & \text{if } (i,j) \in B_1^n, \\
L & \text{otherwise.}
\end{cases}

With \( L = \bigcup_{k=1}^{j-1} L(E_{k,k}) \cdot L(E_{k+1,j}) \) and
L(E'_{k,k}) = \begin{cases} 
L(E_{k,k}) \cup \{\varepsilon\} & \text{if } (k,k) \in T_1^n, \\
L(E_{k,k}) \setminus \{\varepsilon\} & \text{if } (k,k) \in B_1^n, \\
L(E_{k,k}) & \text{otherwise.}
\end{cases}

for all \( 1 \leq i < j \leq n \).

**Corollary 1.** The language of a multi-tilde-bar expression \( E_{1,n}' \) is regular.

As we will see in the following, this recursive definition will allow us to provide the construction of the Glushkov automaton of any EmtbRE. It is worthwhile noticing that in [6], this construction is restricted to saturated EmtbREs, that is expressions such that in each EmtbRE subexpression every factor is equipped with either a tilde or a bar.

Let us define a particular concatenation operator, denoted by \( \odot_{\varepsilon} \), as follows:

\[
L(E'_{i,j}) \odot_{\varepsilon} L(E'_{j+k,n}) = \begin{cases} 
L(E'_{i,j}) \cdot L(E'_{j+k,n}) \setminus \{\varepsilon\} & \text{if } (1,n) \in B_1^n, \\
L(E'_{i,j}) \cdot L(E'_{j+k,n}) & \text{otherwise.}
\end{cases}
\]

**Proposition 1.** Let \( E_{1,n}' \) be an EmtbRE. The language associated with \( E_{1,n}' \) can be recursively computed as follows:

\[
L(E_{1,k}') = \left( L(E_{1,k-1}') \odot_{\varepsilon} L_k \right) \cup \left( \bigcup_{j=1}^{k-1} L(E'_{i,j}) \odot_{\varepsilon} \text{Null}(E'_{j+1,k}) \right) \cup \text{Null}(E_{1,k}), \quad \forall 1 < k \leq n,
\]

with \( L_i = L(E'_{i,i}) \cup \text{Null}(E'_{i,i}), \forall 1 \leq i \leq n \).

**PROOF.** The proof is by induction on \( k \), i.e. the number of factors in \( E_{1,k}' \). Let us consider the case where \( k = 2 \). It is easy to prove that the proposition is true:

\[
L(E'_{1,2}) = \left( L(E_{1,1}') \odot_{\varepsilon} L_2 \right) \cup \left( L(E_{1,1}') \odot_{\varepsilon} \text{Null}(E'_{2,2}) \right) \cup \text{Null}(E_{1,2})
\]

We now suppose that the proposition is satisfied for the EmtbE \( E_{1,k-1}' \) and we prove it is satisfied for \( E_{1,k}' \):

\[
L(E_{1,k}') \overset{\text{Def.}}{=} \bigcup_{j=1}^{k-1} L(E_{1,j}') \odot_{\varepsilon} L(E_{j+1,k}) \cup \text{Null}(E_{1,k}) = \left( L(E_{1,k-1}') \odot_{\varepsilon} L_k \right) \cup \left( \bigcup_{j=1}^{k-2} L(E_{1,j}') \odot_{\varepsilon} L(E_{j+1,k}) \right) \cup \text{Null}(E_{1,k})
\]

\[
= \overset{\text{Ind.Hyp.}}{=} \left( L(E_{1,k-1}') \odot_{\varepsilon} L_k \right) \cup \left( \bigcup_{j=1}^{k-2} L(E_{1,j}') \odot_{\varepsilon} L(E_{j+1,k}) \right) \cup \text{Null}(E_{1,k})
\]

\[
= \bigcup_{j=1}^{k-2} L(E_{1,j}') \odot_{\varepsilon} L(E_{j+1,k}) \cup \left( \bigcup_{j=1}^{k-2} L(E_{1,j}') \odot_{\varepsilon} \text{Null}(E'_{j+1,k}) \right) \cup \text{Null}(E_{1,k})
\]

\[
= \bigcup_{j=1}^{k-1} L(E_{1,j}') \odot_{\varepsilon} \left( L(E_{j+1,k-1}) \cup \text{Null}(E'_{j+1,k}) \right) \cup \text{Null}(E_{1,k})
\]

\[
= \bigcup_{j=1}^{k-1} L(E_{1,j}') \odot_{\varepsilon} \left( L(E_{j+1,k-1}) \cup \text{Null}(E'_{j+1,k}) \right) \cup \text{Null}(E_{1,k})
\]
\[
\begin{align*}
\text{Def. 3:} & \quad \text{First}(E) = \{x \in \text{Pos}(E) \mid xv \in L(E)\} \\
\text{Last}(E) & = \{x \in \text{Pos}(E) \mid ux \in L(E)\} \\
\text{Follow}(x, E) & = \{y \in \text{Pos}(E) \mid uxyv \in L(E)\}
\end{align*}
\]

The position automaton \( P_E \) of a regular expression \( E \) is defined by the 5-uple \( \langle \text{Pos}_0(E), A, \delta, q_0, \text{Last}_0(E) \rangle \) such that:

\[\delta(x, a) = \{y \mid y \in \text{Follow}_0(x, E) \text{ and } h(y) = a\}, \forall x \in \text{Pos}_0(E), \forall a \in A.\]
The position automaton $P_E$ recognizes the language $L(E)$ [13, 17].

Glushkov functions can be defined for bar expressions and tilde expressions as follows, where $x \in \text{Pos}(E)$:

\[
\begin{align*}
\text{First}(E) &= \text{First}(\overline{E}) = \text{First}(\overline{\overline{E}}), \\
\text{Last}(E) &= \text{Last}(\overline{E}) = \text{Last}(\overline{\overline{E}}), \\
\text{Follow}(x, E) &= \text{Follow}(x, \overline{E}) = \text{Follow}(x, \overline{\overline{E}}).
\end{align*}
\]

As a consequence the computation of Glushkov functions can be extended to the family of EmtbREs. Such an extension is described in [6]; it addresses the subfamily of saturated EmtbREs for which every factor is equipped with either a tilde or a bar.

4.2. Glushkov functions for a multi-tilde-bar expression

In this section, we address the general case: we show how to compute the Glushkov functions of an EmtbRE for which there is no restriction on the distribution of tilde and bar operators over the factors of the expression.

**Proposition 2.** Let $E = E_1 \cdot E_2 \cdots E_n$, with $n \geq 1$, and $E' = E'_{1,n}$ be an EmtbRE in linearized form.

Let $k$ be an integer such that $1 \leq k \leq n$ and $x$ be a position in $\text{Pos}(E_k)$. The Glushkov functions associated with $E'$ are recursively computed according to the following formulas:

\[
\begin{align*}
\text{Pos}(E'_{1,n}) &= \bigcup_{k=1}^{n} \text{Pos}(E_k) \\
\text{First}(E'_{1,n}) &= \text{First}(E_1) \uplus \biguplus_{j=1}^{n-1} \text{Null}(E'_{1,j}) \cdot \text{First}(E'_{j+1,j+1}), \\
\text{Last}(E'_{1,n}) &= \text{Last}(E_n) \uplus \biguplus_{j=1}^{n-1} \text{Null}(E'_{j+1,n}) \cdot \text{Last}(E'_{j,j}), \\
\text{Follow}(x, E'_{1,n}) &= \begin{cases} \\
\text{Follow}(x, E_k) & \text{if } (k = n) \lor (x \notin \text{Last}(E_k)) \\
\text{Follow}(x, E_k) \uplus \text{First}(E'_{k+1,n}) & \text{otherwise}.
\end{cases}
\end{align*}
\]
PROOF. Proof is restricted to the non-classical cases:

(1) from the definition of the function First, one has:
First(E′_1,n) = \{ x \in Pos(E′_1,n) | xv \in L(E′_1,n) \}. Using the Proposition and by induction on n, one can deduce the following equalities:

First(E′_1,n) = \{ x \in Pos(E′_1,n) | xv \in \left( \bigcup_{j=1}^{n-1} L(E′_1,j) \cdot Null(E′_{j+1,n}) \right) \} 
\bigcup \bigcup_{j=1}^{n-1} First(E′_1,j) 
Ind.Hyp. = First(E′_1,n-1) \cup Null(E′_{n-1,n}) \cdot First(E′_{n,n}) \cup \bigcup_{j=1}^{n-1} First(E′_1,j) 

(2) from the definition of the function Last, one has:
Last(E′_1,n) = \{ x \in Pos(E′_1,n) | xv \in L(E′_1,n) \}. Using the Proposition and by induction on n, one can deduce the following equalities:

Last(E′_1,n) = \{ x \in Pos(E′_1,n) | xv \in \left( \bigcup_{j=1}^{n-1} L(E′_1,j) \cdot Null(E′_{j+1,n}) \right) \} 
\bigcup \bigcup_{j=1}^{n-1} Last(E′_1,j) \cdot Null(E′_{j+1,n}) 
Ind.Hyp. = Last(E′_{1,n}) \cup \bigcup_{j=1}^{n-1} Last(E′_1,j) \cdot Null(E′_{j+1,n}) 

(3) proof is similar as for (1) and (2).

Corollary 2. The Glushkov functions of a multi-tilde-bar expression can be written as a disjoint union which involves the First, Last, and Follow sets associated with sub-expressions of E′_1,n (not of E_1,n) and the value of the function Null(E′_{i,j}) for all 1 \leq i \leq j \leq n.

The following proposition can be deduced from the Definition:

Proposition 3. Let E′ be an EmbRE in linearized form. The function Null(E′) can be recursively computed as follows:
Thus, by the definition of the set Null, we have Null(E) = \{ \varepsilon \}.

Null(E') = \begin{cases} \emptyset & \text{if } (1, n) \in T_1^n, \\ \{ \varepsilon \} & \text{if } (1, n) \in B_1^n, \\ \bigcup_{j=1}^{n-1} \text{Null}(E'_1) \cdot \text{Null}(E'_{j+1,n}) & \text{otherwise}. \end{cases}

\textbf{Proof.} Proof is by induction on the size of E. It is restricted to the non-classical case (4).

If (1, n) \in T_1^n, then E'_{1,n} can be written as F. Thus, by the definition of the set Null, we have Null(E'_{1,n}) = \{ \varepsilon \}. If (1, n) \in B_1^n, then E'_{1,n} can be written as \overline{F}. Thus, by the definition of the set Null, we have Null(E'_{1,n}) = \emptyset.

Let us suppose that (1, n) \notin T_1^n \cup B_1^n, one has:

\varepsilon \in L(E'_{1,n}) \iff \varepsilon \in \bigcup_{j=1}^{n-1} L(E'_{1,j}) \cdot L(E'_{j+1,n})

\iff \varepsilon \in \bigcup_{j=1}^{n-1} (\text{Null}(E'_{1,j}) \cdot \text{Null}(E'_{j+1,n}))

\iff \varepsilon \in \text{Null}(E'_{1,n})

\textbf{Example 2.} Let us consider the following EmtbRE:

\[ E_{1,7} = a_1^* \cdot b_2 \cdot (c_3 + \varepsilon) \cdot (d_4 + \varepsilon) \cdot (e_5 + \varepsilon) \cdot f_6 \cdot g_7^* \]

The language associated with E_{1,7} is:

\{ a_1 b_2, a_1 b_2 g_7, b_2, b_2 c_3, b_2 g_7, \cdots, b_2 e_5, b_2 c_3 d_4 e_5, b_2 c_3 d_4 e_5 f_6, b_2 c_3 f_6 g_7, b_2 d_4 e_5 f_6 g_7, \cdots, d_4, d_4 e_5, d_4 e_5 f_6, d_4 e_5 f_6 g_7, \cdots, e_5, e_5 f_6, e_5 f_6 g_7, \cdots \}

The associated Glushkov functions are:

\begin{align*}
\text{Pos}(E') &= \{ a_1, b_2, c_3, d_4, e_5, f_6, g_7 \} \\
\text{Null}(E') &= \emptyset \\
\text{First}(E') &= \{ a_1, b_2, d_4, e_5 \} \\
\text{Last}(E') &= \{ b_2, c_3, d_4, e_5, f_6, g_7 \} \\
\text{Follow}(a_1, E') &= \{ a_1, b_2 \} \\
\text{Follow}(b_2, E') &= \{ c_3, d_4, g_7 \} \\
\text{Follow}(c_3, E') &= \{ d_4, e_5, f_6 \} \\
\text{Follow}(d_4, E') &= \{ e_5, f_6 \} \\
\text{Follow}(e_5, E') &= \{ f_6 \} \\
\text{Follow}(f_6, E') &= \{ g_7 \} \\
\text{Follow}(g_7, E') &= \{ g_7 \} 
\end{align*}

\textbf{Figure 1: The Position automaton } \mathcal{A}_{E_{1,7}}
5. Efficient computations of the position automaton and of the c-continuation automaton

In this section, we present efficient algorithms to compute the Glushkov functions of a multi-tilde-bar expression $E'$, based on the formulas of the Proposition 2. According to the Corollary 2, the worst case time complexity of these algorithms depends on the worst case time complexity of the function $\text{Null}(E')$ that we first study.

5.1. Computation of $\text{Null}(E')$

According to the Proposition 3, a naive computation of the function $\text{Null}$ of the $E'_{i,j}$ can be performed using the following Algorithm.

**Data:** $E'_{i,j}$  
**Result:** $\text{Null}(E'_{i,j})$

for $i \leftarrow 1$ to $n$ do
  if $(i, i) \in B^*_1$ then
    $\text{Null}(E'_{i,i}) = \emptyset$
  else
    if $(i, i) \in T^*_1$ then
      $\text{Null}(E'_{i,i}) = \{\varepsilon\}$
    else
      $\text{Null}(E'_{i,i}) = \text{Null}(E_{i,i})$
  end
end

for $k \leftarrow 1$ to $n - 1$ do
  for $i \leftarrow 1$ to $n - k$ do
    if $(i, i + k) \in B^*_1$ then
      $\text{Null}(E'_{i,i+k}) = \emptyset$
    else
      if $(i, i + k) \in T^*_1$ then
        $\text{Null}(E'_{i,i+k}) = \{\varepsilon\}$
      else
        $\text{Null}(E'_{i,i+k}) = \bigcup_{j=1}^{i+k-1} \text{Null}(E'_{i,j}) \cdot \text{Null}(E'_{j+1,i+k})$
      end
    end
  end
end

The different steps of the algorithm are illustrated through the following example.

**Example 3.** Consider the EmtbRE $E'_{1,3}$ such that $T^*_1 = \{(1, 1), (2, 3)\}$, $B^*_1 = \{(1, 2), (3, 3)\}$, and $E_1 = a$, $E_2 = (b + \varepsilon)$, $E_3 = (c + \varepsilon)$. The diagram below is a graphical representation of the recursive dependency between different values of $\text{Null}(E'_{i,j})$. 

10
It holds: 
\[ \text{Null}(E'_{1,3}) = (\text{Null}(E'_{1,1}) \cdot \text{Null}(E'_{2,3})) \cup (\text{Null}(E'_{1,2}) \cdot \text{Null}(E'_{3,3})) \cup \{\varepsilon\} = \{\varepsilon\} \]

Let us consider the case of an EmtbRE \( E'_{1,n} \). There are \((n - k)\) vertices on the \( k^{th} \) line, corresponding to tilde or bar operators \((1, 1 + k), (2, 2 + k), \ldots\). The computation of the associated \( \text{Null}(E'_{i,j+k}) \) functions requires:

- a constant number of elementary test operations:
  - if \((i, i + k) \in T^n_1\) or \((i, i + k) \in B^n_1\),
  - \((k - 1)\) concatenations of \((\text{Null}(E'_{i,j}) \cdot \text{Null}(E'_{j+1,i+k}))\),
  - \((k - 2)\) unions.

Finally, \( \sum_{k=2}^{n} 2 \cdot k \cdot (n - k + 1) \) operations are needed to compute the function \( \text{Null}(E'_{1,n}) \).

**Proposition 4.** Let \( E'_{1,n} \) be an EmtbRE. The function \( \text{Null}(E'_{1,n}) \) can be computed in \( O(\vert E'_{1,n} \vert + n^3) \) time.

Notice that the function \( \text{Null}(E'_{1,n}) \) can be computed by making use of one of the numerous algorithms which compute the transitive closure of a DAG (see for example [12]). Although these algorithms have the same \( O(n^3) \) worst case time complexity as the naive algorithm they likely have a better running time performance than the naive algorithm.

**5.2. Computation of the Glushkov functions**

According to Corollary 2 for an EmtbRE \( E'_{1,n} \), the functions First(\( E'_{1,n} \)), (Resp. Last(\( E'_{1,n} \))), and Follow(\( x, E'_{1,n} \)) can be written as disjoint unions of some First(\( E'_{i,j} \)) (Resp. Last(\( E'_{i,j} \))) sets. Thus, the following proposition holds.

**Proposition 5.** Let \( E'_{1,n} \) be an EmtbRE and \( x \in \text{Pos}(E'_{1,n}) \). The functions First(\( E'_{1,n} \)), Last(\( E'_{1,n} \)), and Follow(\( x, E'_{1,n} \)) can be computed in \( O(|E'_{1,n}| + n^3) \) time.
5.3. Computation of a c-continuation over a ZPC-structure

According to Corollary 2, a multi-tilde-bar expression can be viewed as a standard regular expression equipped with a specific computation for the function Null. The computation of the Glushkov functions of a multi-tilde-bar expression obviously depends on the definition of the function Null: for example, an alternative interpretation of the tilde operator can be associated with the following definition of Null:

\[
\text{Null}(E_1 \cdot E_2 \cdots E_n) = \{\varepsilon\} \iff (\varepsilon \in L(E_1)) \land (\varepsilon \in L(E_n)).
\]

The ZPC-structure [19] can be extended to multi-tilde-bar expressions in a natural way (see Figure 2), by representing the tilde and bar operators by edges connecting the \(^{1,4}\)-nodes of the product. Therefore, all the algorithms based on the ZPC-structure, i.e. the construction of the c-continuation automaton [10], of the equation automaton [10], of the follow automaton [9] and of the weighted position automaton [8] also work for multi-tilde-bar expressions. Moreover the worst case time complexity in the case of multi-tilde-bar expressions is the worst case time complexity of the standard case augmented with the worst case time complexity of the function Null. Therefore, the following theorem can be stated.

**Theorem 1.** Let \(E'\) be a multi-tilde-bar expression and \(N\) the worst case time complexity of the function Null. The position automaton, the c-continuation automaton, the follow automaton and the equation automaton associated with \(E'\) can be computed in 

\[O(|E'| \times ||E'|| + N)\]

time.

The computation of a c-continuation through a ZPC-structure is illustrated by the following example.

**Example 4.** Let us consider the following EmtbRE:

\[
E'_{1,6} = \left( \varepsilon_1 \cdot (b_2 + \varepsilon) \cdot (c_3 + \varepsilon) \cdot (d_4 + \varepsilon) \cdot e_5 \cdot f_6^* \right)^*.
\]

Let us explain how to compute the c-continuation of \(E'\) associated with some position \(x\), denoted by \(c_x(E')\). The ZPC-structure of \(E'\) is partially shown in Figure 2 with all the links which are necessary to computes \(c_{a_1}(E')\) and \(c_{b_2}(E')\). On the right-hand side, the standard First tree is added with blue (resp. green) links between some \(^{1,4}\)-nodes which represent bar (resp. tilde) operators over factors of \(E\). The edge connecting any \(^{1,4}\)-node to its right son is marked by the value of the function Null associated with its left son, and all other edges are marked by \(\varepsilon\).
On the left-hand side, the standard Last tree is added with blue (resp. green) links between '·'-nodes which represent bar (resp. tilde) operators over factors of $E$. The edge connecting any '·'-node to its left son is marked by the value of the function Null associated with its right son, and all other edges are marked by $\varepsilon$. The two trees are connected by the so-called Follow links (red links). For each '·'-node, there is a Follow link going from its left son in the Last tree to its right son in the First tree, and for each *-node, there is a Follow link going from its son in the Last tree to the *-node itself in the First tree.

Figure 2: The ZPC-structure associated with the multi-tilde-bar expression $E'_{1,6}$.

The computation of a c-continuation using a ZPC-structure can be done in a similar way as in the standard case. Let $<(l_1, r_1), (l_2, r_2), \ldots, (l_k, r_k)>$ be the list of follow links in the path going from a position $x$ to the root of the Last tree. Let us denote by $F_i$ the subexpression associated with the node $r_i$ in the First tree. Then the c-continuation $c_x$ associated with $x$ is the expression $F_1 \cdots F_k$. In our example we have:

$$c_{a_1}(E) = \left((b_2 + \varepsilon) \cdot (c_3 + \varepsilon) \cdot (d_4 + \varepsilon) \cdot e_5 \cdot f^*_6\right) \cdot E'_{1,6}$$

$$c_{b_2}(E) = \left((c_3 + \varepsilon) \cdot (d_4 + \varepsilon) \cdot e_5 \cdot f^*_6\right) \cdot E'_{1,6}$$

6. Conclusion

In this paper, we give some answers to open questions raised in [6]. First, we formalize an explicit definition of the language associated with a multi-tilde-bar
expression, which allows us to give a recursive computation of its Glushkov functions. Next, we show that the worst case time complexity to construct the position automaton depends on the worst case time complexity of the function $\text{Null}(E)$. This function can straightforwardly be replaced by another type of function in order to control the application of each tilde or bar. Last, we provide an algorithm to convert a multi-tilde-bar expression into its position automaton, with a cubic worst case time complexity with respect to the size of the multi-tilde-bar expression.

References

[1] V. Antimirov. *Partial derivatives of regular expressions and finite automaton constructions*. Theoret. Comput. Sci., 155:291-319, 1996.

[2] Berry, G. and Sethi, R. *From regular expressions to deterministic automata*. Theoret. Comp. Sci. 48(1), pp. 117-126. 1986.

[3] Berstel, J. and Pin, J-E. *Local languages and the Berry-Sethi algorithm*. Theoret. Comp. Sci. 155(2), pp. 439-446. 1996.

[4] J. A. Brzozowski. *Derivatives of regular expressions*. J. ACM, 11(4):481494, 1964.

[5] Brüggemann-Klein, A. *Regular expressions into finite automata*. Theoret. Comp. Sci. 120, pp. 117-126. 1993.

[6] P. Caron, J.-M. Champarnaud, L. Mignot. *Multi-tilde-bar expressions and their automata*. Acta Inf. 49(6):413-436, 2012.

[7] P. Caron, J.-M. Champarnaud, L. Mignot. *Acyclic automata and small expressions using multi-tilde-bar operators*. Theor. Comput. Sci. 411(38-39):3423-3435, 2010.

[8] J.-M. Champarnaud, E. Laugerotte, F. Ouardi, D. Ziadi. *From Regular Weighted Expressions To Finite Automata*. Int. J. Found. Comput. Sci. 15(5):687-700, 2004.

[9] J.-M. Champarnaud, F. Nicart, D. Ziadi. *From the ZPC Structure of a Regular Expression to its Follow Automaton*. IJAC, 16(1):17-34, 2006.

[10] J.-M. Champarnaud and D. Ziadi. *Canonical derivatives, partial derivatives and finite automaton constructions*. Theor. Comput. Sci. 289(1):137-163, 2002.
[11] J.-M. Champarnaud, J.-L. Ponty, and D. Ziadi. *From regular expressions to finite automata*. Internat. J. Comput. Math., 72:415431, 1999.

[12] Y. Chen. *A new algorithm for computing transitive closures*. ACM Symposium on Applied Computing, 1091-1092, 2004.

[13] V. M. Glushkov. *The abstract theory of automata*. Russian Mathematical Surveys, 16:1-53, 1961.

[14] L. Ilie and S. Yu. *Follow automata*. Inf. Comput., 186(1):140162, 2003.

[15] A. Khorsi, F. Ouardi, D. Ziadi. *Fast equation automaton computation*. J. Discrete Algorithms, 6(3):433-448, 2008.

[16] S. Kleene. *Representation of events in nerve nets and finite automata*. Automata Studies, Ann. Math. Studies 34:341, 1956. Princeton U. Press.

[17] R. F. McNaughton and H. Yamada. *Regular expressions and state graphs for automata*. IEEE Transactions on Electronic Computers, 9:3957, March 1960.

[18] L. Mignot. *Des Codes Barres pour les Langages Rationnels*. PhD thesis, LITIS, Université de Rouen, France, 2010. available online, URL: [http://ludovicmignot.free.fr](http://ludovicmignot.free.fr).

[19] D. Ziadi, J.-L. Ponty and J.-M. Champarnaud, *Passage d’une expression rationnelle à un automate fini non-déterministe*. Journées Montoises, Bull. Belg. Math. 1995.