

Evaluation of three implicit structures to implement nondeterministic automata from regular expressions

Jean-Marc Champarnaud
LIFAR, Université de Rouen
F-76821 Mont-Saint-Aignan Cedex, France

Abstract

The aim of this paper is to compare three efficient representations of the position automaton of a regular expression: the Thompson ε -automaton, the \mathcal{ZPC} -structure and the \mathcal{F} -structure, an optimization of the \mathcal{ZPC} -structure. These representations are linear w.r.t. the size s of the expression, since their construction is in $O(s)$ space and time, as well as the computation of the set $\delta(X, a)$ of the targets of the transitions by a of any subset X of states. The comparison is based on the evaluation of the number of edges of the underlying graphs respectively created by the construction step or visited by the computation of a set $\delta(X, a)$.

Keywords: Regular Expression, Finite Automaton, NFA, Implicit Structure.

1 Introduction

An efficient implementation of nondeterministic finite automata (NFA's) computation is based on two main features: the data structure to represent the NFA and the process to compute the set $\delta(X, a)$ of the targets of the transitions by a of an arbitrary subset X of states. In the general case, the $\delta(X, a)$ sets are independent a priori and the NFA is memorized by a table whose (q, a) -entry is the set $\delta(q, a)$. The computation of any $\delta(X, a)$ set can be deduced from the transition table. The complexity of a NFA is therefore generally measured by the size of its transition table, i.e. by the number of transitions.

For instance, the number of transitions of an arbitrary n -states NFA over an alphabet Σ is bounded by $|\Sigma|n^2$. If E is a regular expression of alphabetic width n the number of states in the position automaton [7, 10] of E is equal to $n + 1$ and the number of transitions is bounded by n^2 ; on the other hand, the common follow sets automaton [9, 8] of E has $O(n)$ states and $O(n \log^2(n))$ transitions.

According to this usual definition of NFA's complexity, the position automaton of a regular expression of alphabetic width n has the same $O(n^2)$ complexity as an arbitrary automaton, whereas the common follow sets automaton has an $O(n \log^2(n))$ complexity.

However, this classical definition does not take in consideration the specific properties which allow to provide a more efficient implementation of some NFA's families. This is the case when the NFA is obtained from a regular expression: the syntactic structure of the expression induces dependencies among the $\delta(X, a)$ sets. Thanks to this property, a shared representation of the information can be designed, which leads both to save memory space and to speed up the process to compute the $\delta(X, a)$ sets. For instance, the position automaton of an expression of alphabetic width n can be implemented with an $O(n)$ space and time complexity, leading to an $O(n)$ computation of any $\delta(X, a)$ set [2, 5, 14, 11].

In this paper we examine three representations of the position automaton yielding a linear complexity: the Thompson ε -automaton [12], the \mathcal{ZPC} -structure [14, 11, 4], and an optimization of the \mathcal{ZPC} -structure, the \mathcal{F} -structure. We explain the relationship between these structures and compare the number of elementary operations involved by their construction and by the computation of the $\delta(X, a)$ sets. Notice that the relationship between the position automaton and the Thompson ε -automaton has been studied in [6]; our approach is a more algorithmic one: in particular, the Thompson ε -automaton is compared to other linear representations rather than to the position automaton itself.

In order to deepen the comparison, some assumptions are made concerning input regular expressions. These hypothesis are presented in the following section. In Section 3, we recall the definition of the position automaton and the complexity of the computations performed on its table. Section 4 is devoted to the \mathcal{ZPC} -structure: we present an inductive definition of this representation, and give an accurate analysis of its complexity. The construction of the Thompson ε -automaton is recalled in Section 5. Section 6 provides a comparison between the \mathcal{ZPC} -structure of an expression and its Thompson ε -automaton, based on the fact that the state graph of the ε -automaton can be deduced from the syntax tree of the expression. The \mathcal{F} -structure and its complexity are described in Section 7. In order to provide a visual comparison of the various constructions, the figures have been gathered in Annex A.

2 Hypothesis

The complexity of a regular expression is generally measured by its *size* s , i.e. the length of its prefixed form, or by its (alphabetic) *width* w , i.e. the number of occurrences of alphabet symbols. An arbitrary regular expression, such as the argument of a pattern matching command, may contain an arbitrary number of empty set, empty word and Kleene star operator occurrences. Its complexity should be measured by s , which may be arbitrarily greater than w . It is the

reason why the complexity of the structures we study are firstly given w.r.t. s . In practical applications, it is profitable to preprocess the input expression in order to reduce the number of empty set, empty word and Kleene star operator occurrences. We define *reduced* expressions and show that they have linearly dependent size and width.

Definition 1 *A regular expression E is said to be reduced if it is such that:*

1. *Either E is the expression \emptyset or E contains no occurrence of the empty set.*
2. *Either E is the expression ε or the empty word only occurs in subexpressions $F + \varepsilon$ or $\varepsilon + F$, with $F \neq \varepsilon$ ('+' is denoted by '+ $_\varepsilon$ ' in these expressions).*
3. *Two consecutive operations in E cannot be both either an '*' or a '+ $_\varepsilon$ ' operation.*

Proposition 1 *Let E' be a regular expression of size s' and width w' . It is possible to construct a reduced regular expression E equivalent to E' , of size s and width w , such that:*

1. *The expression E is deduced from E' in $O(s')$ space and time.*
2. *The expressions E and E' have an identical width: $w = w'$.*
3. *If $E \neq \emptyset$ and $E \neq \varepsilon$, then E is such that:*
 - (a) *The overall number of '.' and '+' operators is equal to $w - 1$.*
 - (b) *The overall number of '*' and '+ $_\varepsilon$ ' operators is bounded by $2w - 1$.*
 - (c) *The overall number of '.' et '*' operators is bounded by $3w - 2$.*
 - (d) *The size of E is such that: $2w - 1 \leq s \leq 6w - 3$. The lower bound is reached when there is no occurrence of the empty word in E and no occurrence of the '*' operator. The upper bound is reached when there are $2w - 1$ occurrences of the empty word in E .*

The Thompson ε -automaton and the ZPC -structure are known to be linear structures w.r.t. s since their construction and the computation of $\delta(X, a)$ sets are space and time linear. If the input expression is a reduced one, the complexity can be expressed w.r.t. w and thus made more precise. Moreover, a detailed implementation will be provided for each structure, in order to obtain an exact measure of the space. Lastly, the number of elementary operations will be bounded under the following assumption:

Hypothesis H₁ : *The number of elementary operations to construct a structure (resp. to compute a $\delta(X, a)$ set) is proportional to the number of created (resp. visited) edges.*

3 The position automaton of a regular expression

3.1 Definition of the position automaton

A regular expression over an alphabet Σ is *linear* if and only if all its symbols are distinct. The following sets of symbols are associated to a linear expression E :

- $Null(E) = \{\varepsilon\}$ if $\varepsilon \in L(E)$ and \emptyset otherwise,
- the set $First(E)$ of symbols matching the first symbol of some word in $L(E)$,
- the set $Last(E)$ of symbols matching the last symbol of some word in $L(E)$,
- the sets $Follow(E, x)$ of symbols following x in some word of $L(E)$, $\forall x \in \Sigma$.

Now, let E be a regular expression over Σ . If a is the j^{th} occurrence of a symbol in E , a_j is the *position* associated to a . The set of positions of E is denoted by $Pos(E)$. The linear expression deduced from E by substituting each symbol by its position is called *the linearized version* of E and is denoted by \overline{E} . Let h be the mapping from $Pos(E)$ to Σ such that $h(x)$ is the symbol related to the position x . For instance, let $E = a \cdot (a+b) + (a+b) \cdot (\varepsilon+b)$. We have: $Pos(E) = \{a_1, a_2, b_3, a_4, b_5, b_6\}$, $\overline{E} = a_1 \cdot (a_2 + b_3) + (a_4 + b_5) \cdot (\varepsilon + b_6)$, $h(a_1) = h(a_2) = h(a_4) = a$ and $h(b_3) = h(b_5) = h(b_6) = b$. The sets of positions associated to E are straightforwardly deduced from the sets of symbols associated to \overline{E} : $Null(E) = Null(\overline{E})$, $First(E) = First(\overline{E})$, $Last(E) = Last(\overline{E})$ and, $\forall x \in Pos(E)$, $Follow(E, x) = Follow(\overline{E}, x)$. Notice that the second argument of $Follow(E, x)$ is a position when E is an arbitrary expression and is a symbol when E is a linear expression. Finally, the position automaton \mathcal{P}_E of E is deduced from these sets as follows.

Definition 2 *The position automaton of E , $\mathcal{P}_E = (Q, \Sigma, I, F, \delta)$, is defined by:*

- $Q = Pos(E) \cup \{0\}$, where 0 is not in $Pos(E)$,
- $I = \{0\}$,
- $F = \begin{cases} Last(E) & \text{if } Null(E) = \emptyset \\ Last(E) \cup \{0\} & \text{otherwise} \end{cases}$
- $\delta(0, a) = \{x \in First(E) \mid h(x) = a\}$, $\forall a \in \Sigma$,
- $\delta(x, a) = \{y \mid y \in Follow(E, x) \text{ and } h(y) = a\}$, $\forall x \in Pos(E)$, $\forall a \in \Sigma$.

In the following, an expression E such that $Null(E) = \{\varepsilon\}$ is said to be *nullable*. Let us notice that, in a position automaton, for all state y , the label of each transition going into y is equal to $h(y)$: the position automaton is said to be *homogeneous*.

3.2 Recursive computation of the position sets

The sets $Null(E)$, $First(E)$, $Last(E)$ and $Follow(E, x)$ can be computed according to recursive formulas similar to the following ones, which hold for the case $E = F \cdot G$:

$$\begin{aligned}
Null(F \cdot G) &= Null(F) \cap Null(G) \\
First(F \cdot G) &= \begin{cases} First(F) \cup First(G) & \text{if } Null(F) = \{\varepsilon\} \\ First(F) & \text{otherwise} \end{cases} \\
Last(F \cdot G) &= \begin{cases} Last(F) \cup Last(G) & \text{if } Null(G) = \{\varepsilon\} \\ Last(G) & \text{otherwise} \end{cases} \\
Follow(F \cdot G, x) &= \begin{cases} Follow(F, x) \cup First(G) & \text{if } x \in Last(F) \\ Follow(F, x) \cup Follow(G, x) & \text{otherwise} \end{cases}
\end{aligned}$$

3.3 Complexity

The complexity of the position automaton is the following. The size of the table is bounded by $w(w+1)$. A direct computation of the $Follow(E, x)$ sets is in $O(w^2)$ time, hence an $O(w^3)$ construction of the table. The computation of $\delta(x, a)$ can be performed in $O(w)$ time, via three means: the conversion of the expression into its star normal form [2], the lazy transition evaluation defined in [5], or the elimination of the redundant follow links in the \mathcal{ZPC} -structure [14, 11]. It leads to an $O(w^2)$ construction of the table. Moreover, the computation of $\delta(X, a)$, where X is an arbitrary set of states, is performed in $O(w^2)$ time using the table.

4 The \mathcal{ZPC} -structure of a regular expression

The \mathcal{ZPC} -structure is described in [14, 11]. It is made of two copies of the syntax tree and of a collection of links connecting their nodes and implementing the computation of the *First*, *Last* and *Follow* sets of the subexpressions of E .

4.1 Definition of the \mathcal{ZPC} -structure

Let $E_0 = \$ \cdot (E) \cdot \#$, where ‘\$’ and ‘#’ are symbols not in Σ , and let $Pos(E_0) = \{x_0, x_1, x_2, \dots, x_w, x_{w+1}\}$, with $h(x_0) = ‘\$’$ and $h(x_{w+1}) = ‘\#’$, be the set of positions of E_0 . The \mathcal{ZPC} -structure of E is defined by: $\mathcal{ZPC}_E = zpc(E_0)$; the

inductive construction of $zpc(E)$ is illustrated by Figure A.1. Notice that the definition we give here is slightly different from the original one [14, 11]. The links used to compute the *Last* sets have been discarded, since our main goal is to design an efficient pattern matcher. The graphical presentation has been modified too, to facilitate the comparison with the Thompson ε -automaton.

Let us give some indications concerning this construction. The two copies of the syntax tree of E_0 associated to the structure \mathcal{ZPC}_E are denoted by $Firsts(E)$ and $Lasts(E)$. If F is a subexpression of E_0 the corresponding node in $Firsts(E)$ (resp. $Lasts(E)$) is denoted by φ_F (resp. λ_F). As recalled below, the computation of $\delta(X, a)$ involves a bottom-up (resp. top-down) traversal of $Lasts(E)$ (resp. $Firsts(E)$). Hence the following implementation of \mathcal{ZPC}_E , illustrated by Figure A.3:

- an array *lpositions* of size $w + 1$, such that *lpositions*[k], $k = 0$ to w , is a pointer to the leaf λ_{x_k} associated to the position x_k (notice it is not necessary to store a pointer to the leaf $\lambda_{x_{w+1}}$);
- for a node λ in $Lasts(E)$:
 - a pointer *lparent* to the parent node; if λ is the root of a tree in the forest $Lasts(E)$ then *lparent*(λ) is set to NULL,
 - a pointer *follow*, from the son of a node ‘*’ in $Lasts(E)$ (resp. the left son of a node ‘.’) to its copy in $Firsts(E)$ (resp. the right son of its copy),
 - a boolean *notvisited* initialized to *true* before any $\delta(X, a)$ computation;
- an array *roots* of size s , such that *root*[i] is a pointer to the root of a tree in the forest $Firsts(E)$;
- for a node φ of $Firsts(E)$:
 - the pointers *leftson* and *frightson* deduced from the syntax tree,
 - the pointers *fbegin*, *fend* and *fnext* used to compute $First(\varphi)$,
 - an integer *ftree*, which is the index of the tree the node φ belongs to;
- for each leaf associated to a position: the character *symbol* and the integer *frank* associated to the position.

4.2 Complexity of the construction of \mathcal{ZPC}_E

The forests $Firsts(E)$ and $Lasts(E)$ are generally drawn with distinct nodes. They can however be implemented with a shared set of $s + 2$ nodes. Each node is equipped with six pointers: *leftson*, *frightson*, *fbegin*, *fend*, *lparent* and

follow, and three data: the booleans *null* and *notvisited* and the index *ftree*. Moreover, each of the $w + 1$ leaves of $Firsts(E)$ is equipped with two data: the pointer *fnext* and the character *symbol*. Lastly, the size of the array *lpositions* is equal to $w + 1$, and the size of the array *froots*, which is less than the number of operators '.', is bounded by s if E is an arbitrary expression, and by w if E is reduced. Hence the following proposition:

Proposition 2 *Let e be the space taken up by the structure \mathcal{ZPC}_E . If the expression is an arbitrary one, we have $e \leq 8s + 3w$. If the expression is a reduced one, we have $e \leq 7s + 4w$ and $18w < e < 46w$.*

As far as the time is concerned, the computation of the boolean *null* must be taken in account. Conversely, some pointers are only computed for a subset of the set of nodes: the overall number of non-null *leftson* and *rightson* pointers added with the number of trees in the forest $Firsts(E)$ is equal to $s - 1$, and the pointers *follow* are only generated by the '.' and '*' nodes. According to Proposition 1, the overall number of operators '.' and '*', and thus of non-null *follow* pointers, is bounded by $3w - 2$. Lastly, the computation of each pointer is in $O(1)$ time. Under the hypothesis \mathbf{H}_1 , we get the following proposition:

Proposition 3 *Let t be the time taken up by the construction of the structure \mathcal{ZPC}_E . If the expression is an arbitrary one, we have $t \leq 7s + 3w$. If the expression is a reduced one, we have $t \leq 7s$ and $14w < t < 42w$.*

4.3 Computation of $\delta(X, a)$

The computation of the $\delta(X, a)$ sets on a \mathcal{ZPC} -structure is examined in [4]. Since the position automaton is homogeneous, $\delta(X, a)$ can be obtained in $O(w)$ time from $\delta(X) = \bigcup_{a \in \Sigma} \delta(X, a)$, according to the formula:

$$\delta(X, a) = \{y \in \delta(X) \mid h(y) = a\}$$

Moreover, the set $Y = \delta(X)$ can be computed by the following algorithm:

Algorithm deltaZPC:

- **Step 1:** Compute the set Λ of nodes λ in $Lasts(E)$ such that $Last(\lambda) \cap X \neq \emptyset$ and there exists a *follow* link exiting from node λ .
- **Step 2:** Compute the set Φ of nodes φ in $Firsts(E)$ such that there exists a *follow* link in Λ entering in φ . The set $Y = \delta(X)$ is such that $Y = \bigcup_{\varphi \in \Phi} First(\varphi)$.
- **Step 3:** Deduce a set Φ' from Φ so that the set Y is computed according to the formula: $Y = \biguplus_{\varphi \in \Phi'} First(\varphi)$.

Let us take for instance the expression $E = ((a \cdot (a + b + \varepsilon))^* \cdot b)^*$ and consider the structure \mathcal{ZPC}_E of Figure A.3. Assuming that $X = \{a_1, b_4\}$, the following sets are computed: $\Lambda = \{\lambda_5, \lambda_4, \lambda_3, \lambda_2, \lambda_1\}$, $\Phi = \{\varphi_6, \varphi_4, \varphi_{11}, \varphi_2, \varphi_{\#}\}$, $\Phi' = \{\varphi_6, \varphi_2, \varphi_{\#}\}$ and $Y = \{a_1, a_2, b_3, b_4, \#\}$.

4.4 Complexity of the computation of $\delta(X, a)$

Let us examine the complexity of the Algorithm *deltaZPC*. Let L_X be the set of the roots of the trees in $Lasts(E)$ containing at least one node of Λ . For all λ in L_X , λ_X is defined as the partial subtree rooted in λ and made of all the paths going from λ to the leaves labelled by a position in X . Similarly, let F_X be the set of the roots of the trees in $Firsts(E)$ containing *at least two* nodes of Φ . For all φ in F_X , φ_X is defined as the partial subtree rooted in φ and obtained by deletion of the subtrees rooted in a node belonging to Φ . The number of edges in λ_X (resp. φ_X) is denoted by $|\lambda_X|$ (resp. $|\varphi_X|$). The size of the set Λ (resp. Φ) is bounded by the number f_X of *follow* links. Hence the following proposition:

Proposition 4 *Under the hypothesis \mathbf{H}_1 , the computation time of the different steps of the Algorithm *deltaZPC* is the following:*

- *Step 1:* $t_1 = 2 \sum_{\lambda \in L_X} |\lambda_X|$
- *Step 2:* $t_2 = f_X$
- *Step 3:*
 - *Computation of Φ' :* $t'_3 = \sum_{\varphi \in F_X} |\varphi_X|$
 - *Computation of Y :* $t''_3 < w + 2f_X$

Let us remark that the 2 coefficient in t_1 is due to the necessary initialization of the boolean *notvisited* before each $\delta(X, a)$ computation. In Step 3, if a $Firsts(E)$ tree exactly contains one node of Φ , this node is straightforwardly added to Φ' . The computation of $First(\varphi)$, for all φ in Φ' , visits the two edges $fbegin(\varphi)$ and $fend(\varphi)$. The size of Φ' is bounded by the size of Φ . Moreover the computation of Y as a disjoint union implies to visit $|Y| \leq w + 1$ *next* edges. Hence the bound on the time t_3 .

5 The i-automaton of a regular expression

In an ε -automaton, instantaneous transitions, i.e. transitions on the empty word ε , are allowed. Thompson has designed in [12] the construction of an ε -automaton recognizing the language of a given expression. We here present the definition of the *i-automaton* of an expression, a variant of the Thompson ε -automaton. The interest of the i-automaton is that its structure is closely

related to the syntax tree of the expression, which makes the comparison to the \mathcal{ZPC} -structure more obvious.

5.1 Definition of the i-automaton

The i-automaton $(Q, \Sigma, i_E, t_E, \delta_i)$ of the expression E is defined inductively by the schemas of Figure A.2, where the unlabelled edges correspond to ε -transitions. For each subexpression F of E , the inductive construction produces the two states i_F and t_F . The set of states of the i-automaton is the union of the set of states i_F and of the set of states t_F .

5.2 Properties of the i-automaton

Let us first notice that the Thompson ε -automaton can be viewed as an optimization of the i-automaton: in the case $E = F \cdot G$, Thompson merges the states i_E and i_F , as well as the states t_E and t_G . The properties we state for i-automata in the following are well-known properties of Thompson ε -automata.

Let us recall that an ε -path is a path where all the edges are labelled by ε . The ε -closure of a state q , denoted by $\varepsilon(q)$, is the set of the states which are accessible from q by an ε -path. The set of the states from which q can be reached by an ε -path is denoted by $\varepsilon^{-1}(q)$. The relationship between the i-automaton and the position automaton is stated by the following proposition, whose proof is by induction on the size of E .

Proposition 5 *Let $Pos(E) = \{x_1, x_2, \dots, x_p\}$ be the set of positions of E and $(Q, Pos(E), i_E, t_E, \delta_i)$ be the i-automaton of \overline{E} . Let I (resp. T) the set of states i_{x_k} (resp. t_{x_k}) generated by the expressions x_k . The following properties hold:*

1. $Null(E)$ is equal to $\{\varepsilon\}$ if and only if there exists an ε -path from i_E to t_E ,
2. $First(E) = \varepsilon(i_E) \cap I$,
3. $Last(E) = \varepsilon^{-1}(t_E) \cap T$,
4. $Follow(E, x_k) = \{x_\ell \mid i_{x_\ell} \in \varepsilon(t_{x_k})\}$,
5. $\delta(x_k, a) = \{x_\ell \mid t_{x_\ell} \in \delta_i(\varepsilon(t_{x_k}) \cap I, a)\}$.

5.3 Complexity of the construction of the i-automaton

An i-automaton can be represented by a table with $2s$ entries. Each state (except t_E) is the origin either of one symbol-transition, or of one or two ε -transitions. Moreover, the computation of each transition is in $O(1)$ time. Therefore, the space and time taken up by the construction of the table are equal to $4s$. More precisely, according to Proposition 1, and under the hypothesis \mathbf{H}_1 , it can be

proved that the time is bounded by $15w$ when the expression is reduced. Moreover, due to the fact that the number of edges going out of a state is bounded by 2, the computation of the set $\delta(X, a)$ is performed in $O(s)$ time.

6 \mathcal{ZPC} -structure vs. i-automaton

We now use the fact that the i-automaton is deduced from the syntax tree of the expression to show similarities and differences with the \mathcal{ZPC} -structure.

6.1 The structure of the i-automaton

Let I_E and T_E be two copies of the syntax tree of E . Let X_I (resp. X_T) be the set of nodes of I_E (resp. T_E). Let $\mathcal{G}_E = (U, V, \Sigma \cup \{\varepsilon\})$ be a labelled digraph such that $U = X_I \cup X_T$ and V is obtained from the set Γ_I of the edges *ileftson* and *irightson* of I_E , the set Γ'_T of the edges *tparent* of T_E and the following sets:

1. E_1 : the set of the edges *irightson* in I_E with a node ‘.’ as origin,
2. E_2 : the set of the edges *tparent* in T_E associated to the *tleftson* edge of a node ‘.’,
3. E_3 : the set of the edges from the son of a node ‘*’ in T_E to its copy in I_E ,
4. E_4 : the set of the edges from the left son of a node ‘.’ in T_E to the right son of its copy in I_E ,
5. E_5 : the set of the edges from a node ‘*’ in I_E to its copy in T_E ,
6. E_6 : the set of the edges from a leaf (symbol, empty word) in I_E to its copy in T_E .

We have: $V = (\Gamma_I \setminus E_1) \cup (\Gamma'_T \setminus E_2) \cup (E_3 \cup E_4 \cup E_5 \cup E_6)$. All the edges are labelled by ε except the symbol-edges of E_6 . The set V can be computed through a traversal of the syntax tree. The following proposition is a direct consequence of the inductive definition of the i-automaton.

Proposition 6 *The graph \mathcal{G}_E is isomorphic to the state graph of the i-automaton of E .*

The comparison between \mathcal{G}_E and \mathcal{ZPC}_E yields the following similarities:

1. Identical sets of nodes (as far as \mathcal{ZPC}_E is defined on two distinct copies).
2. Identical sets of edges connecting the second copy to the first one: the set $E_3 \cup E_4$ in \mathcal{G}_E is equivalent to the set of *follow* links in \mathcal{ZPC}_E .

3. Closely related sets of syntactic edges:

- (a) the sets of *ileftson* edges and of *leftson* edges are equivalent,
- (b) the set of *tparent* edges is equivalent to the set of *lparent* edges minus the edges (λ_F, λ_E) in the case $E = F \cdot G \wedge \text{Null}(G) = \{\varepsilon\}$,
- (c) the set of *irightson* edges is equivalent to the set of *frightson* edges minus the edges (φ_E, φ_G) in the case $E = F \cdot G \wedge \text{Null}(F) = \{\varepsilon\}$.

On the other hand, the main differences are the following:

1. *The handling of the positions:* in \mathcal{G}_E , each leaf of I_E associated to a position is connected to the corresponding leaf of T_E by an edge labelled by the associated symbol. In \mathcal{ZPC}_E , the array *lpositions* provides the addresses of the leaves associated to the positions (which are shared by $\text{First}(E)$ and $\text{Last}(E)$).
2. *The processing of the nullable subexpressions:* in \mathcal{G}_E , a subexpression F is nullable if and only if there exists at least one ε -path from i_F to t_F . The existence of such a path is related to the ε -edges connecting ε -leaves in I_E to the corresponding leaves in T_E , and by the edges of the set E_ε coming from the processing of the starred expressions. In \mathcal{ZPC}_E , the boolean $\text{null}(F)$ is computed for each subexpression F .
3. *The computation of the set $\text{First}(F)$, for F a non-nullable expression of E :* in \mathcal{G}_E , we have: $\text{First}(F) = \varepsilon(i_F) \cap Y$; the computation of $\text{First}(F)$ implies the exploration of the subtree of I_E rooted in i_F . In \mathcal{ZPC}_E , $\text{First}(F)$ is directly computed from the pointers *fbegin* et *fend* associated to F and from *fnext* links.

6.2 Complexity of the construction of \mathcal{G}_E and \mathcal{ZPC}_E

The two constructions are in $O(s)$ space and time. The following table makes this result more precise, by giving the space taken by each structure, and the construction time under the hypothesis \mathbf{H}_1 . The input is either an arbitrary expression, or a reduced one. Identical assumptions have been made to evaluate the complexity of the two constructions (cf. Propositions 2 and 3).

Table 1: Complexity of the \mathcal{G}_E and \mathcal{ZPC}_E structures.

	<i>arbitrary expression</i>		<i>reduced expression</i>	
	\mathcal{G}_E	\mathcal{ZPC}_E	\mathcal{G}_E	\mathcal{ZPC}_E
space	$4s + 2w$	$8s + 3w$	$10w < e < 26w$	$18w < e < 46w$
time	$4s + 2w$	$7s + 3w$	$6w < t < 17w$	$14w < t < 42w$

The construction of \mathcal{ZPC}_E is about two times more expensive than the construction of \mathcal{G}_E . The question is to know whether the additional information it computes allows a fairly large speedup of $\delta(X, a)$ sets computation or not.

6.3 Computation of $\delta(X, a)$

The comparison of the operations performed to compute the sets $\delta(X, a)$ in \mathcal{G}_E and in \mathcal{ZPC}_E is made possible by the following proposition:

Proposition 7 *Let ρ and ρ' be two nodes in the same forest of \mathcal{ZPC}_E and let r and r' be the corresponding nodes in \mathcal{G}_E . The following properties are equivalent:*

1. *There exists a path from ρ to ρ' .*
2. *There exists an ε -path from r to r' .*

Proof. Let us first verify that the property holds in $Lasts(E)$ and T_E . The critical case is when $F = G \cdot H$ and $Null(H) = \{\varepsilon\}$. In this case, there exists a link *lparent* from λ_G to λ_F and no link *tparent* from t_G to t_F . However, there exists a *follow* link from t_G to i_H and a *tparent* link from t_H to i_F . Moreover, there exists an ε -path from i_H to t_H since $Null(H) = \{\varepsilon\}$. Finally, there exists an ε -path from t_G to t_F . It can be proved in a similar way that if there exists a link *frightson* from φ_F to φ_H in $Firsts(E)$ and no link *irightson* from i_F to i_H in I_E , then there exists an ε -path from i_F to i_H .

For example, on Figures A.3 and A.4 which respectively represent the \mathcal{ZPC} -structure and the \mathcal{G} -structure of the expression $E = ((a \cdot (a + b + \varepsilon))^* \cdot b)^*$, it appears that:

- The ε -path $(t_5, i_6, i_8, i_{10}, t_{10}, t_8, t_6, t_4)$ acts as the *lparent* link (λ_5, λ_4) .
- The ε -path (i_2, i_3, t_3, i_{11}) acts as the *frightson* link $(\varphi_2, \varphi_{11})$.
- There exists no ε -path from t_3 to t_2 and no *lparent* link from λ_3 to λ_2 .
- There exists no ε -path from i_4 to i_6 and no *frightson* link from φ_4 to φ_6 .

Finally, the computation of $\delta(X, a)$ in \mathcal{G}_E amounts to compute sets which are equivalent to Λ , Φ and Y sets, as in the Algorithm *deltaZPC*, and with the following complexity:

- Step 1: The number of visited edges is greater in T_E than in $Lasts(E)$ due to the ε -paths which act as *lparent* or *frightson* links.
- Step 3:
 - Since the boolean *notvisited* is used in I_E , each edge must be visited a second time due to a necessary re-initialization step.

- If a tree in $Firsts(E)$ contains only one node φ of Φ , then $First(\varphi)$ is obtained directly from the pointers $fbegin(\varphi)$ and $fend(\varphi)$ and from the $fnext$ linking. On the opposite, the corresponding subtree in I_E is explored.
- If a tree in $Firsts(E)$ contains at least two nodes of Φ , the subsets of edges of this tree respectively visited in each structure are complementary.

This comparison can be summarized as follows:

Proposition 8 *The average computation time of $\delta(X, a)$ is smaller when performed on the structure \mathcal{ZPC}_E than on the structure \mathcal{G}_E ; the ratio between the average number of edges visited in $Firsts(E)$ and in I_E is equal to $2/3$.*

7 The forest-structure of an expression

From now on, we assume that E is a reduced expression and that f is the number of follow links. Figure A.5 illustrates the construction of \mathcal{F}_E , the forest-structure of the expression $E = ((a \cdot (a + b + \varepsilon))^* \cdot b)^*$.

The structure \mathcal{F}_E is derived from \mathcal{ZPC}_E by the following optimizations:

- *A new handling of occurrences of the empty word:* We only consider reduced expressions, and we introduce a unary operator ‘ \diamond ’ which is such that: $Null(F^\diamond) = \{\varepsilon\}$, $First(F^\diamond) = First(F)$, $Last(F^\diamond) = Last(F)$ and, $\forall x \in Pos(F)$, $Follow(F^\diamond, x) = Follow(F, x)$, and F^\diamond is substituted to each occurrence of $F + \varepsilon$ and $\varepsilon + F$. Since there are no longer leaves associated to the empty word, the pointers $fbegin$ and $fend$ are necessarily non-null, hence a faster computation. Moreover the size of a reduced expression is now bounded by $4w - 2$.
- *A compression of the forests $Firsts(E)$ and $Lasts(E)$:* The forest $Lasts(E)$ (resp. $Firsts(E)$) is compressed so that only the heads (resp. the tails) of follow links are kept. This processing leads to the following properties:
 1. the compressed forests do not necessarily share the same set of nodes,
 2. the number of sons of a node may be more than 2,
 3. in each forest, the number of nodes is equal to f and the number of leaves is bounded by $w + 1$.

Finally, we get the following evaluation for the construction of a forest-structure:

Proposition 9 *Let e (resp. t) be the space (resp. time) taken up by the construction of the \mathcal{F}_E structure. We have: e and t are approximatively equal to $7f + 3w$ and $3w < e, t < 24w$.*

Proposition 9 holds under the same assumptions as for the analysis of the structures \mathcal{G}_E and \mathcal{ZPC}_E . Moreover, the Algorithm *deltaZPC* can be readily made suitable for a \mathcal{F}_E structure. Due to the reduction of the size of the forests, the computation of a $\delta(X, a)$ set is faster on \mathcal{F}_E than on \mathcal{ZPC}_E .

8 Conclusion and perspectives

The comparative analysis of the structures \mathcal{G}_E , \mathcal{ZPC}_E and \mathcal{F}_E has been conceived as a preliminary work to their implementation and to an experimental study of the performances. For the construction step, the main results are the following:

Table 2: Complexity of the \mathcal{G}_E , \mathcal{ZPC}_E and \mathcal{F}_E structures.

	<i>reduced expression</i>		
	\mathcal{G}_E	\mathcal{ZPC}_E	\mathcal{F}_E
space	$10w < e < 26w$	$18w < e < 46w$	$3w < e < 24w$
time	$6w < t < 17w$	$14w < t < 42w$	$3w < t < 24w$

Concerning the computation of the $\delta(X, a)$ sets, we have shown that the structure \mathcal{ZPC}_E is more efficient in average than \mathcal{G}_E . Moreover the structure \mathcal{F}_E is expected to be more efficient than \mathcal{ZPC}_E , since its forests have a smaller size.

References

- [1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.
- [2] A. Brüggemann-Klein, Regular Expressions into Finite Automata. *Theoret. Comput. Sci.*, 120(1993), 197–213.
- [3] D. Beauquier, J. Berstel, and P. Chrétienne. *Éléments d'Algorithmique*. Masson, Paris, 1992.
- [4] J.-M. Champarnaud, Subset Construction Complexity for Homogeneous Automata, Position Automata and ZPC-Structures, *Theoret. Comp. Sc.*, 269/1-2, to appear.
- [5] C.-H. Chang and R. Paige. From Regular Expressions to DFAs using Compressed NFAs, in Apostolico. Crochemore. Galil. and Manber. editors. *Lecture Notes in Computer Science*, 644(1992), 88-108.

- [6] G. Giammarresi , J.-L. Ponty and D. Wood, The Glushkov and Thompson Constructions: A Synthesis. TCSC Research Report 1998-11, HKUST, Hong Kong, 1998, <http://www.cs.ust.hk/tcsc/RR/1998-11.ps.gz>.
- [7] V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1961.
- [8] Ch. Hagenah and A. Muscholl, Computing ε -free NFA from Regular Expressions in $O(n \log^2(n))$ Time, in: L. Prim *et al.* (eds.), MFCS'98, *Lecture Notes in Computer Science*, 1450(1998), 277-285, Springer.
- [9] J. Hromkovič, S. Seibert, and T. Wilke. Translating regular expressions into small ε -free nondeterministic finite automata. In R. Reischuk and M. Morvan, eds, STACS 97, *Lecture Notes in Computer Science*, 1200(1997), 55–66, Springer.
- [10] R. F. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9:39–57, March 1960.
- [11] J.-L. Ponty, D. Ziadi, and J.-M. Champarnaud. A new quadratic algorithm to convert a regular expression into an automaton. In D. Raymond *et al.*, eds, WIA'96, *Lecture Notes in Computer Science*, 1260(1997), 109–119, Springer.
- [12] K. Thompson, Regular Expression Search Algorithm, *Comm. Assoc. Comput. Mach.* 11(1968) 419–422.
- [13] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume I, Word, Language, Grammar, pages 41–110. Springer, Berlin, 1997.
- [14] D. Ziadi, J.-L. Ponty and J.-M. Champarnaud. Passage d'une expression rationnelle à un automate fini non-déterministe, Journées Montoises (1995), *Bull. Belg. Math. Soc.*, 4:177-203, 1997.

Figures

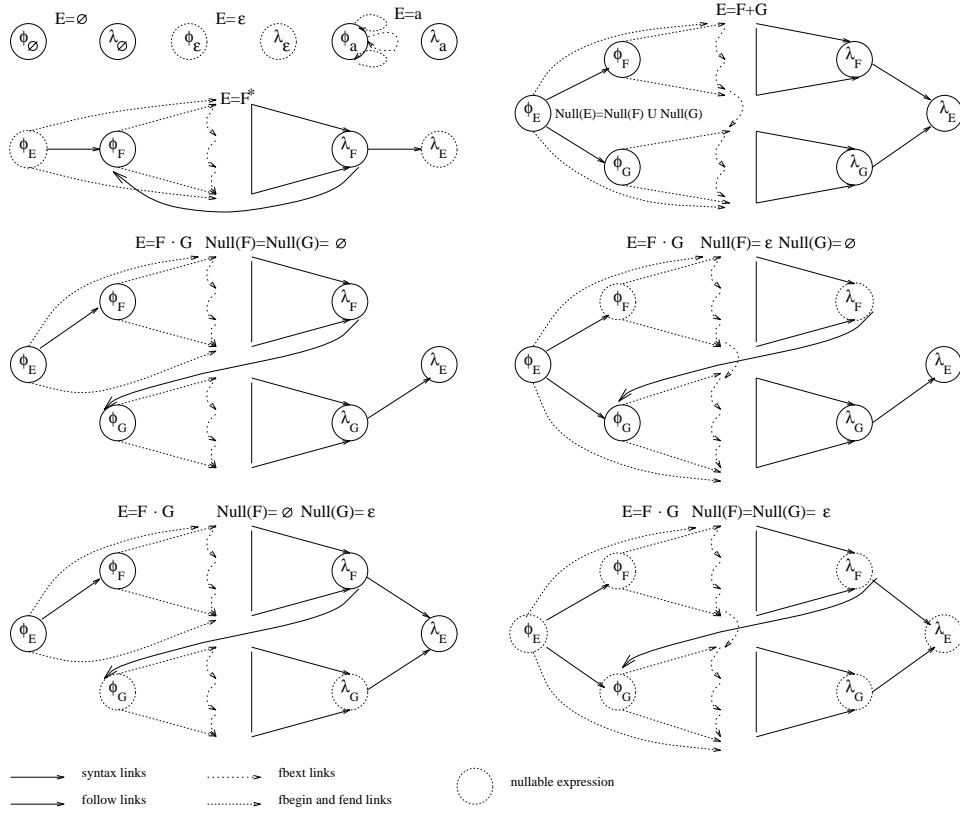


Figure 1: The inductive definition of the structure $zpc(E)$.

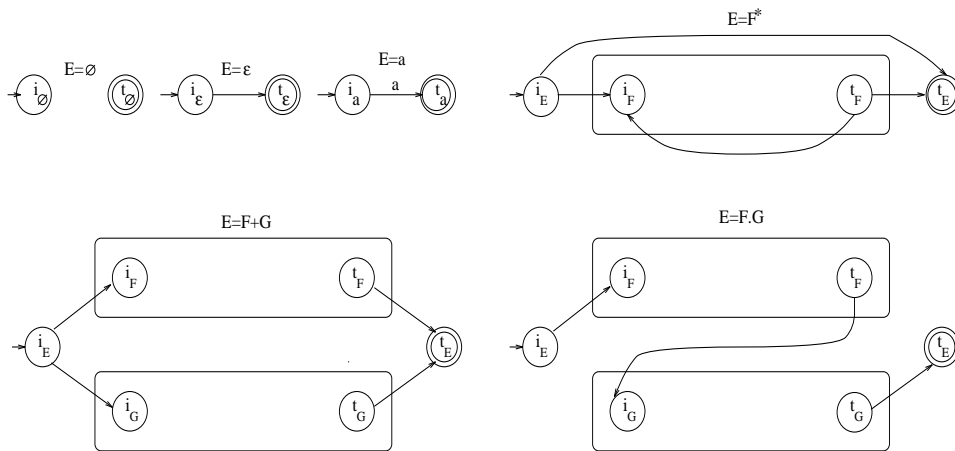


Figure 2: The inductive definition of the i-automaton of a regular expression.

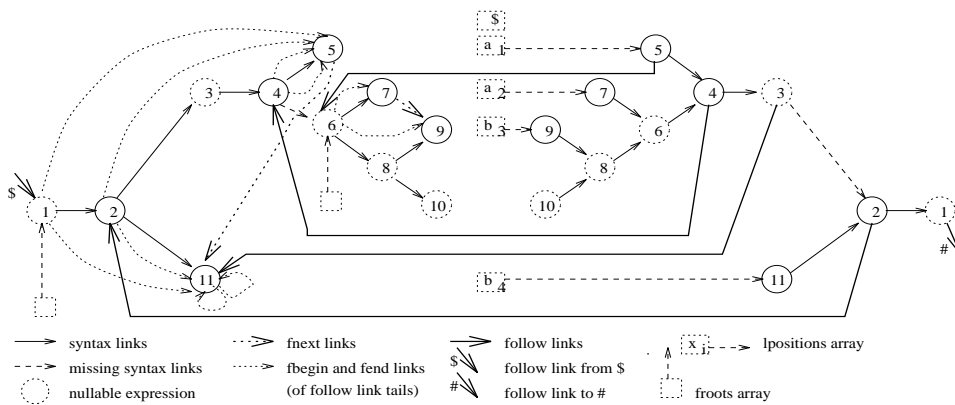


Figure 3: The structure ZPC_E of $E = ((a \cdot (a + b + \varepsilon))^* \cdot b)^*$.

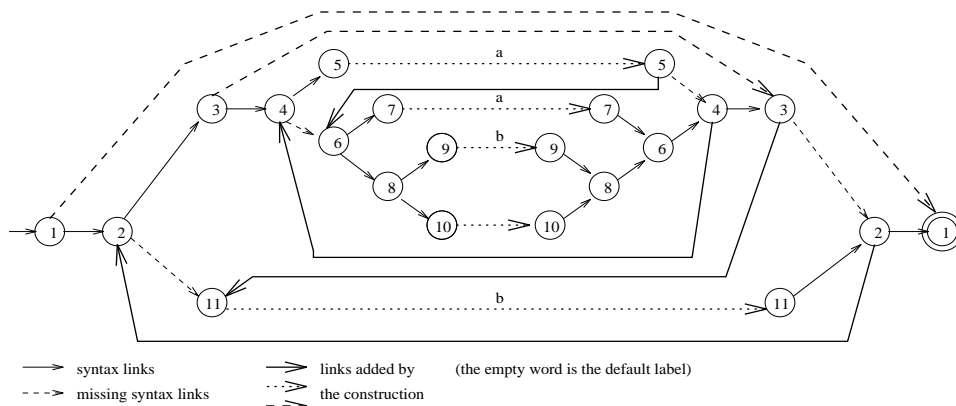


Figure 4: The i-automaton of $E = ((a \cdot (a + b + \varepsilon))^* \cdot b)^*$.

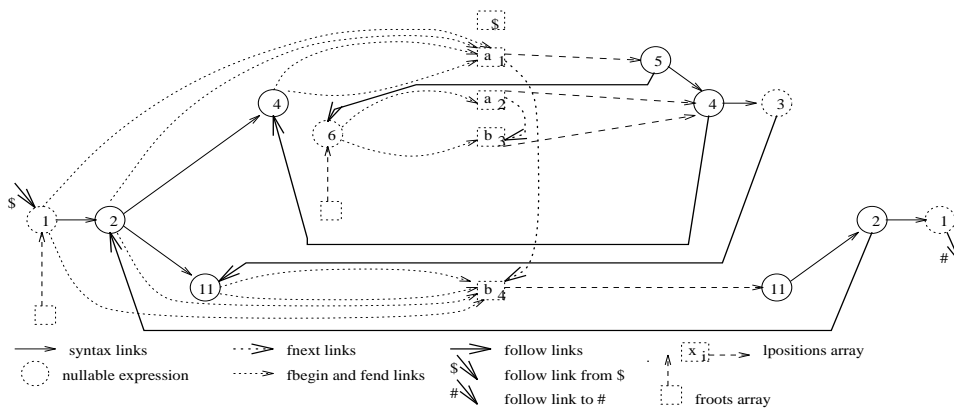


Figure 5: The structure \mathcal{F}_E of $E = ((a \cdot (a + b + \varepsilon))^* \cdot b)^*$.