The Bottom-Up Position Tree Automaton, the Father Automaton and their Compact Versions

Samira Attou[1], Ludovic Mignot[2], and Djelloul Ziadi[2]

[1] USTHB, Faculty of Mathematics, RECITS Laboratory,
BP 32, El Alia, 16111 Bab Ezzouar, Algiers, Algeria
[2] Groupe de Recherche Rouennais en Informatique Fondamentale,
Université de Rouen Normandie,
Avenue de l'Université, 76801 Saint-Étienne-du-Rouvray, France
sattou@usthb.dz,{ludovic.mignot,djelloul.ziadi}@univ-rouen.fr

Abstract. The conversion of a given regular tree expression into a tree automaton has been widely studied. However, classical interpretations are based upon a Top-Down interpretation of tree automata. In this paper, we propose new constructions based on the Gluskov's one and on the one of Ilie and Yu one using a Bottom-Up interpretation. One of the main goals of this technique is to consider as a next step the links with deterministic recognizers, consideration that cannot be performed with classical Top-Down approaches. Furthermore, we exhibit a method to factorize transitions of tree automata and show that this technique is particularly interesting for these constructions, by considering natural factorizations due to the structure of regular expression.

1 Introduction

Automata are recognizers used in various domains of applications especially in computer science, *e.g.* to represent (non necessarily finite) languages, or to solve the membership test, *i.e.* to verify whether a given element belongs to a language or not. Regular expressions are compact representations for these recognizers. Indeed, in the case where elements are words, it is well known that each regular expression can be transformed into a finite state machine recognizing the language it defines. Several methods have been proposed to realize this conversion. As an example, Glushkov [8] (and independently Mc-Naughton and Yamada [13]) showed how to construct a non deterministic finite automaton with $n+1$ states where n represents the number of letters of a given regular expression. The main idea of the construction is to define some particular sets named First, Follow and Last that are computed with respect to the occurrences of the symbols that appear in the expression.

These so-called Glushkov automata (or Position automata) are finite state machines that have been deeply studied. They have been structurally characterized by Caron and Ziadi [6], allowing us to invert the Glushkov computation by constructing an expression with n symbols from a Glushkov automaton with $n+1$ states. They have been considered too in the theoretical notion of one-unabiguity by Bruggemann-Klein and Wood [5], characterizing regular languages recognized by a deterministic Glushkov automaton, or with practical thoughts, like expression updating [4]. Finally, it is also related to combinatorial research topics. As an example, Nicaud [16] proved that the average number of transitions of Glushkov automata is linear.

Moreover, the Glushkov automata can be easily reduced into the Follow automata [10] in the case of word by applying an easy-to-compute congruence from the position functions.

The Glushkov construction was extended to tree automata [12,15], using a Top-Down interpretation of tree expressions. This interpretation can be problematic while considering determinism. Indeed, it is a folklore that there exist regular tree languages that cannot be recognized by Top-Down deterministic tree automata. Extensions of one-ambiguity are therefore incompatible with this approach.

In this paper, we propose a new approach based on the construction of Glushkov and of Ilie and Yu in a Bottom-Up interpretation. We also define a compressed version of tree automata in order to factorize the transitions, and we show how to apply it directly over these computations using natural factorizations due to the structure of the expressions. The paper is structured as follows: in Section 2, we recall some properties related to regular tree expressions; we also introduce some basics definitions. We define, in Section 3, the position functions used for the construction of the Bottom-Up position tree automaton. Section 4 indicates the way that we construct the Bottom-Up position tree automaton with a linear number of states using the functions shown in Section 3. In Section 5, we propose the notion of compressed automaton and show how to reduce the size of the Position automaton computed in the previous section. Finally, we show in Section 6 and in Section 7 how to extend the notion of Follow automaton in a Bottom-Up interpretation.

This paper is an extended version of [3]. It is a full-proof version that contains two new constructions: the father automaton and its compressed version.

2 Preminiaries

Let us first introduce some notations and preliminary definitions. For a boolean condition ψ, we denote by $(E \mid \psi)$ E if ψ is satisfied, $\emptyset$ otherwise. Let $\Sigma = (\Sigma_n)_{n\geq 0}$ be a finite ranked alphabet. A *tree* t over Σ is inductively defined by $t = f(t_1, \ldots, t_k)$ where $f \in \Sigma_k$ and $t_1, \ldots, t_k$ are k trees over Σ. The relation "s is a subtree of t" is denoted by $s \prec t$ for any two trees s and t. We denote by $\mathrm{root}(t)$ the root symbol of the tree t, *i.e.*

$$\mathrm{root}(f(t_1, \ldots, t_k)) = f. \tag{1}$$

The *predecessors* of a symbol f in a tree t are the symbols that appear directly above it. We denote by $\mathrm{father}(t, f)$, for a tree t and a symbol f the pairs

$$\mathrm{father}(t, f) = \{(g, i) \in \Sigma_l \times \mathbb{N} \mid \exists g(s_1, \ldots, s_l) \prec t, \mathrm{root}(s_i) = f\}. \tag{2}$$

These couples link the predecessors of f and the indices of the subtrees in t that f is the root of. Let us consider a tree $t = g(t_1, \ldots, t_k)$ and a symbol f. By definition of the structure of a tree, a predecessor of f in t is a predecessor of f in a subtree t_i of t, or g if f is a root of a subtree t_i of t. Consequently:

$$\mathrm{father}(t, f) = \bigcup_{i\leq n} \mathrm{father}(t_i, f) \cup \{(g, i) \mid f \in \mathrm{root}(t_i)\}. \tag{3}$$

We denote by T_Σ the set of trees over Σ. A tree language L is a subset of T_Σ.

For any 0-ary symbol c, let $t \cdot_c L$ denote the tree language constituted of the trees obtained by substitution of any symbol c of t by a tree of L. By a linear extension, we denote by $L \cdot_c L' = \{t \cdot_c L' \mid t \in L\}$. For an integer n, the n-th substitution ^{c,n} of a language L is the language $L^{c,n}$ recursively defined by

$$L^{c,n} = \begin{cases} \{c\}, & \text{if } n = 0, \\ L \cdot_c L^{c,n-1} & \text{otherwise.} \end{cases}$$

Finally, we denote by $L(E_1^{*_c})$ the language $\bigcup_{k\geq 0} L(E_1)^{c,k}$.

An *automaton* over Σ is a 4-tuple $\mathrm{A} = (Q, \Sigma, Q_F, \delta)$ where Q is a set of states, $Q_F \subseteq Q$ is the set of final states, and $\delta \subset \bigcup_{k\geq 0}(Q^k \times \Sigma_k \times Q)$ is the set of transitions, which can be seen as the function from $Q^k \times \Sigma_k$ to 2^Q defined by

$$(q_1, \ldots, q_k, f, q) \in \delta \Leftrightarrow q \in \delta(q_1, \ldots, q_k, f).$$

It can be linearly extended as the function from $(2^Q)^k \times \Sigma_k$ to 2^Q defined by

$$\delta(Q_1, \ldots, Q_n, f) = \bigcup_{(q_1,\ldots,q_n)\in Q_1\times\cdots Q_n} \delta(q_1, \ldots, q_n, f). \tag{4}$$

Finally, we also consider the function Δ from T_Σ to 2^Q defined by

$$\Delta(f(t_1, \ldots, t_n)) = \delta(\Delta(t_1), \ldots, \Delta(t_n), f).$$

Using these definitions, the language $L(A)$ recognized by the automaton A is the language

$$\{t \in T_\Sigma \mid \Delta(t) \cap Q_F \neq \emptyset\}.$$

A tree automaton $A = (\Sigma, Q, F, \delta)$ is *deterministic* if for any symbol f in Σ_m, for any m states $q_1, \ldots, q_m$ in Q, $|\delta(q_1, \ldots, q_m, f)| \leq 1$.

A regular expression E over the alphabet Σ is inductively defined by:

$$E = f(E_1, \ldots, E_k), \qquad E = E_1 + E_2,$$
$$E = E_1 \cdot_c E_2, \qquad E = E_1^{*_c},$$

where $k \in \mathbb{N}$, $c \in \Sigma_0$, $f \in \Sigma_k$ and $E_1, \ldots, E_k$ are any k regular expressions over Σ. In what follows, we consider expressions where the subexpression $E_1 \cdot_c E_2$ only appears when c appears in the expression E_1. The *language denoted* by E is the language $L(E)$ inductively defined by

$$\begin{aligned} L(f(E_1, \ldots, E_k)) &= \{f(t_1, \ldots, t_k) \mid t_j \in L(E_j), j \leq k\}, \\ L(E_1 + E_2) &= L(E_1) \cup L(E_2), \\ L(E_1 \cdot_c E_2) &= L(E_1) \cdot_c L(E_2), \\ L(E_1^{*_c}) &= L(E_1)^{*_c}, \end{aligned}$$

with $k \in \mathrm{N}$, $c \in \Sigma_0$, $f \in \Sigma_k$ and $E_1, \ldots, E_k$ any k regular expressions over Σ.

A regular expression E is *linear* if each symbol Σ_n with $n \neq 0$ occurs at most once in E. Note that the symbols of rank 0 may appear more than once. We denote by $\overline{E}$ the linearized form of E, which is the expression E where any occurrence of a symbol is indexed by its position in the expression. The set of indexed symbols, called *positions*, is denoted by $\mathrm{Pos}(\overline{E})$. We also consider the *delinearization* mapping h sending a linearized expression over its original unindexed version.

Let ϕ be a function between two alphabets Σ and Σ' such that ϕ sends Σ_n to Σ'_n for any integer n. By a well-known adjunction, this function is extended to an *alphabetical morphism* from $T(\Sigma)$ to $T(\Sigma')$ by setting $\phi(f(t_1, \ldots, t_n)) = \phi(f)(\phi(t_1), \ldots, \phi(t_n))$. As an example, one can consider the delinearization morphism h that sends an indexed alphabet to its unindexed version. Given a language L, we denote by $\phi(L)$ the set $\{\phi(t) \mid t \in L\}$. The *image by* ϕ of an automaton $A = (\Sigma, Q, Q_F, \delta)$ is the automaton $\phi(A) = (\Sigma', Q, Q_F, \delta')$ where

$$\delta' = \{(q_1, \ldots, q_n, \phi(f), q) \mid (q_1, \ldots, q_n, f, q) \in \delta\}.$$

By a trivial induction over the structure of the trees, it can be shown that

$$\phi(L(A)) = L(\phi(A)). \tag{5}$$

An alphabetical morphism is a particular case of *automaton morphism* between two automata $A = (\Sigma, Q, F, \delta)$ and $B = (\Sigma', Q', F', \delta')$, that is a function ϕ that sends Σ_n to Σ'_n for any integer n, Q to Q', F to F' and δ to δ' such that

$$\delta'((\phi(q_1), \ldots, \phi(q_n)), \phi(f)) = \{\phi(q) \mid q \in \delta((q_1, \ldots, q_n), f)\}. \tag{6}$$

In this case, we set $\phi(A) = (\phi(\Sigma), \phi(Q), \phi(F), \phi(\delta))$. Let us first show that morphisms are stable w.r.t. transition composition.

Lemma 1. *Let $A = (\Sigma, \_, \_, \delta)$ be an automaton and ϕ be an automaton morphism from A. Let $\phi(A) = (\Sigma', \_, \_, \delta')$. For any tree t in $T(\Sigma)$,*

$$\Delta'(\phi(t)) = \{\phi(q) \mid q \in \Delta(t)\}.$$

Proof. Let us proceed by induction over the structure of the trees in $T(\Sigma)$. Let $t = f(t_1, \ldots, t_n)$ be a tree in $T(\Sigma)$. Let us set $S_i = \{\phi(q) \mid q \in \Delta(t_i)\}$ for any integer $i \leq n$. Notice that $S_i = (\Delta'(\phi(t_i))$ for any integer $i \leq n$ from the induction hypothesis. For a given state q' of $\phi(A)$, we denote by $\phi^{-1}(q)$ the set of the states q' in A satisfying $\phi(q) = q'$. Then: from ,

$$\begin{aligned}
\Delta'(\phi(f)(\phi(t_1), \ldots, \phi(t_n))) &= \delta'((\Delta'(\phi(t_1)), \ldots, \Delta'(\phi(t_n))), \phi(f)) \\
&= \delta'((S_1, \ldots, S_n), \phi(f)) && \textbf{(Induction Hypothesis)} \\
&= \bigcup_{(q_1, \ldots, q_n) \in S_1 \times \cdots \times S_n} \delta'((q_1, \ldots, q_n), \phi(f)) \\
&= \bigcup_{(p_1, \ldots, p_n) \in \Delta(t_1) \times \cdots \times \Delta(t_n)} \delta'((\phi(p_1), \ldots, \phi(p_n)), \phi(f)) \\
&= \bigcup_{(p_1, \ldots, p_n) \in \Delta(t_1) \times \cdots \times \Delta(t_n)} \{\phi(q) \mid q \in \delta((p_1, \ldots, p_n), f)\} && \textbf{(Equation (6))} \\
&= \{\phi(q) \mid q \in \delta((\Delta(t_1), \ldots, \Delta(t_n)), f)\} \\
&= \{\phi(q) \mid q \in \Delta(f(t_1, \ldots, t_n))\}.
\end{aligned}$$

□

As a direct consequence of the previous lemma,

Proposition 1. *Let $A = (\Sigma, \_, \_, \_)$ be an automaton and ϕ be an automaton morphism from A. If for any symbol f in Σ, $\phi(f) = f$, then*

$$L(A) = L(\phi(A)).$$

Two automata A and B are *isomorphic* if there exist two morphisms ϕ and ϕ' satisfying

$$A = \phi'(\phi(A)), \qquad B = \phi(\phi'(B)).$$

3 Position Functions

In this section, we define the position functions that are considered in the construction of the Bottom-Up automaton in the next sections. We show how to compute them and how they characterize the trees in the language denoted by a given expression.

Let E be a linear expression over a ranked alphabet Σ and f be a symbol $\in \Sigma_k$. The set $\mathrm{Root}(E)$, subset of Σ, contains the roots of the trees in $L(E)$, *i.e.*

$$\mathrm{Root}(E) = \{\mathrm{root}(t) \mid t \in L(E)\}. \tag{7}$$

The set $\mathrm{Father}(E, f)$, subset of $\Sigma \times \mathbb{N}$, contains a couple (g, i) if there exists a tree in $L(E)$ with a node labeled by g the i-th child of is a node labeled by f:

$$\mathrm{Father}(E, f) = \bigcup_{t \in L(E)} \mathrm{father}(t, f). \tag{8}$$

Example 1. Let us consider the ranked alphabet defined by $\Sigma_2 = \{f\}$, $\Sigma_1 = \{g\}$, and $\Sigma_0 = \{a, b\}$. Let E and $\overline{E}$ be the expressions defined by

$$E = (f(a, a) + g(b))^{*_a} \cdot_b f(g(a), b), \quad \overline{E} = (f_1(a, a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b).$$

Hence,

$$\begin{aligned}
&\mathrm{Root}(\overline{E}) = \{a, f_1, g_2\},\\
\mathrm{Father}(\overline{E}, f_1) &= \{(f_1, 1), (f_1, 2)\}, \quad \mathrm{Father}(\overline{E}, a) = \{(f_1, 1), (f_1, 2), (g_4, 1)\},\\
\mathrm{Father}(\overline{E}, g_2) &= \{(f_1, 1), (f_1, 2)\}, \quad \mathrm{Father}(\overline{E}, b) = \{(f_3, 2)\},\\
\mathrm{Father}(\overline{E}, f_3) &= \{(g_2, 1)\}, \qquad\qquad \mathrm{Father}(\overline{E}, g_4) = \{(f_3, 1)\}.
\end{aligned}$$

Let us show how to inductively compute these functions.

Lemma 2. *Let E be a linear expression over a ranked alphabet Σ. The set* $\mathrm{Root}(E)$ *is inductively computed as follows:*

$$\begin{aligned}
\mathrm{Root}(f(E_1, ..., E_n)) &= \{f\},\\
\mathrm{Root}(E_1 + E_2) &= \mathrm{Root}(E_1) \cup \mathrm{Root}(E_2),\\
\mathrm{Root}(E_1 \cdot_c E_2) &= \begin{cases} \mathrm{Root}(E_1) \setminus \{c\} \cup \mathrm{Root}(E_2) & \text{if } c \in L(E_1),\\ \mathrm{Root}(E_1) & \text{otherwise,} \end{cases}\\
\mathrm{Root}(E_1^{*_c}) &= \mathrm{Root}(E_1) \cup \{c\},
\end{aligned}$$

where $E_1, \ldots, E_n$ are n regular expressions over Σ, f is a symbol in Σ_n and c is a symbol in Σ_0.

Proof. Let us consider the following cases.

1. The case when $E = f(E_1, \ldots, E_n)$ is a direct consequence of Equation (1) and Equation (7).
2. Let us consider a tree t in $L(E_1 + E_2)$. Consequently, t is in $L(E_1)$ or in $L(E_2)$. And we can conclude using Equation (7).
3. Let us consider a tree t in $t_1 \cdot_c L(E_2)$ with $t_1 \in L(E_1)$. If $t_1 = c$ (resp. $t_1 \neq c$), then it holds by definition of $\cdot_c$ that $\mathrm{root}(t) \in \mathrm{Root}(L(E_2))$ (resp. $\mathrm{root}(t) = \mathrm{root}(t_1) \setminus \{c\}$). Once again, we can conclude using Equation (7).
4. Let us consider a tree t in $L(E_1^{*_c})$. By definition, either $t = c$ or t is in $t_1 \cdot_c L(E_1^{*_c})$ with $t_1 \in L(E_1) \setminus \{c\}$. Therefore either $\mathrm{root}(t) = c$ or $\mathrm{root}(t) = \mathrm{root}(t_1)$. Once again, we can conclude using Equation (7).

□

Lemma 3. *Let E be a linear expression and f be a symbol in Σ_k. The set* $\mathrm{Father}(E, f)$ *is inductively computed as follows:*

$$\begin{aligned}
\mathrm{Father}(g(E_1, ..., E_n), f) &= \bigcup_{i \leq n} \mathrm{Father}(E_i, f) \cup \{(g, i) \mid f \in \mathrm{Root}(E_i)\},\\
\mathrm{Father}(E_1 + E_2, f) &= \mathrm{Father}(E_1, f) \cup \mathrm{Father}(E_2, f),\\
\mathrm{Father}(E_1 \cdot_c E_2, f) &= (\mathrm{Father}(E_1, f) \mid f \neq c) \cup \mathrm{Father}(E_2, f)\\
&\quad \cup (\mathrm{Father}(E_1, c) \mid f \in \mathrm{Root}(E_2))\\
\mathrm{Father}(E_1^{*_c}, f) &= \mathrm{Father}(E_1, f) \cup (\mathrm{Father}(E_1, c) \mid f \in \mathrm{Root}(E_1)),
\end{aligned}$$

where $E_1, \ldots, E_n$ are n regular expressions over Σ, g is a symbol in Σ_n and c is a symbol in Σ_0.

Proof. Let us consider the following cases.

1. The case when $E = g(E_1, \ldots, E_n)$ is a direct consequence of Equation (3) and Equation (7).
2. Let us consider a tree t in $L(E_1 + E_2)$. Consequently, t is in $L(E_1)$ or in $L(E_2)$. And we can conclude using Equation (3).
3. Let us consider a tree $t = t_1 \cdot_c L(E_2)$ with $t_1 \in L(E_1)$. By definition, t equals t_1 where the occurrences of c have been replaced by some trees t_2 in $L(E_2)$. Two cases may occur.
   (a) If $c \neq f$, then a predecessor of the symbol f in t can be a predecessor of the symbol f in a tree t_2 in $L(E_2)$, a predecessor of the symbol f in t_1, or a predecessor of c in t_1 if an occurrence of c in t_1 has been replaced by a tree t_2 in $L(E_2)$ the root of which is f.
   (b) If $c = f$, since the occurrences of c have been replaced by some trees t_2 of $L(E_2)$, a predecessor of the symbol c in t can be a predecessor of the symbol c in a tree t_2 in $L(E_2)$, or a predecessor of c in t_1 if an occurrence of c has been replaced by itself (and therefore if it appears in $L(E_2)$).

   And we can conclude using Equation (3) and Equation (7).
4. By definition, $L(E_1^{*_c}) = \bigcup_{k \geq 0} L(E_1)^{c,k}$. Therefore, a tree t in $L(E_1^{*_c})$ is either c or a tree t_1 in $L(E_1)$ where the occurrences of c have been replaced by some trees t_2 in $L(E_1)^{c,k}$ for some integer k. Let us then proceed by recursion over this integer k. If $k = 1$, a predecessor of f in t is a predecessor of f in t_1, a predecessor of f in a tree t_2 in $L(E_1)^{c,1}$ or a predecessor of c in t_1 if an occurrence of c in t_1 was substituted by a tree t_2 in $L(E_1)^{c,1}$ the root of which is f, *i.e.*

   $$\mathrm{Father}(E_1^{c,2}, f) = \mathrm{Father}(E_1, f) \cup (\mathrm{Father}(E_1, c) \mid f \in \mathrm{Root}(E_1)).$$

   By recursion over k and by applying the same reasoning, it can be shown that each recursion step adds $\mathrm{Father}(E_1, f)$ to the result of the previous step, and therefore

   $$\mathrm{Father}(E_1^{c,k}, f) = \mathrm{Father}(E_1, f) \cup (\mathrm{Father}(E_1, c) \mid f \in \mathrm{Root}(E_1)).$$

□

Let us now show how these functions characterize, for a tree t, the membership of t in the language denoted by an expression.

Definition 1. *Let E be a linear expression over a ranked alphabet Σ and t be a tree in $T(\Sigma)$. The property $P(t)$ is the property defined by*

$$\forall s = f(t_1, \ldots, t_n) \prec t, \forall i \leq n, (f, i) \in \mathrm{Father}(E, \mathrm{root}(t_i)).$$

Proposition 2. *Let E be a linear expression over a ranked alphabet Σ and t be a tree in $T(\Sigma)$. Then* ***(1)*** *t is in $L(E)$ if and only if* ***(2)*** $\mathrm{root}(t)$ *is in* $\mathrm{Root}(E)$ *and $P(t)$ is satisfied.*

Proof. Let us first notice that the proposition $1 \Rightarrow 2$ is direct by definition of Root and Father. Let us show the second implication by induction over the structure of E. Hence, let us suppose that $\mathrm{root}(t)$ is in $\mathrm{Root}(E)$ and $P(t)$ is satisfied.

– Let us consider the case when $E = g(E_1, \ldots, E_n)$ and let us set $t = f(t_1, \ldots, t_n)$. Since $\mathrm{root}(t)$ is in $\mathrm{Root}(E)$, $f = g$ from Lemma 2. From $P(t)$, it holds that for any $i \leq n$, $(f, i) \in \mathrm{Father}(E, \mathrm{root}(t_i))$. Since E is linear, and following Lemma 3, $\mathrm{root}(t_i) \in \mathrm{Root}(E_i)$. Consequently, from the induction hypothesis, t_i is in $L(E_i)$ for any integer $i \leq n$ and t belongs to $L(E)$.

– The case of the sum is a direct application of the induction hypothesis.
– Let us consider the case when $E = E_1 \cdot_c E_2$. Let us first suppose that $\mathrm{root}(t)$ is in $\mathrm{Root}(E_2)$. Then c is in $L(E_1)$ and $P(t)$ is equivalent to

$$\forall s = f(t_1, \ldots, t_n) \prec t, \forall i \leq n, (f, i) \in \mathrm{Father}(E_2, \mathrm{root}(t_i)).$$

By induction hypothesis t is in $L(E_2)$ and therefore in $L(E)$.
Let us suppose now that $\mathrm{root}(t)$ is in $\mathrm{Root}(E_1)$. Since E is linear, let us consider the subtrees t_2 of t with only symbols of E_2 and a symbol of E_1 as a predecessor in t. Since $P(t)$ holds, according to induction hypothesis and Lemma 3, each of these trees belongs to $L(E_2)$. Hence t belongs to $t_1 \cdot_c L(E_2)$ where t_1 is equal to t where the previously defined t_2 trees are replaced by c. Once again, since $P(t)$ holds and since $\mathrm{root}(t)$ is in $\mathrm{Root}(E_1)$, t_1 belongs to $L(E_1)$.
In these two cases, t belongs to $L(E)$.
– Let us consider the case when $E = E_1^{*_c}$. Let us proceed by induction over the structure of t. If $t = c$, the proposition holds from Lemma 2 and Lemma 3. Following Lemma 3, each predecessor of a symbol f in t is a predecessor of f in E_1 (case **1**) or a predecessor of c in E_1 (case **2**). If all the predecessors of the symbols satisfy the case **1**, then by induction hypothesis t belongs to $L(E_1)$ and therefore to $L(E)$. Otherwise, we can consider (similarly to the catenation product case) the smallest subtrees t_2 of t the root of which admits a predecessor in t which is a predecessor of c in E_1. By induction hypothesis, these trees belong to $L(E_1)$. And consequently t belongs to $t' \cdots L(E_1)$ where t' is equal to t where the subtrees t_2 have been substituted by c. Once again, by induction hypothesis, t' belongs to $L(E_1^{*_c})$. As a direct consequence, t belongs to $L(E)$.

□

4 Bottom-Up Position Automaton

In this section, we show how to compute a Bottom-Up automaton with a linear number of states from the position functions previously defined.

Definition 2. *The* Bottom-Up Position automaton $\mathcal{P}_E$ *of a linear expression* E *over a ranked alphabet* Σ *is the automaton* $(\Sigma, \mathrm{Pos}(E), \mathrm{Root}(E), \delta)$ *defined by:*

$$((f_1, \ldots, f_n), g, g) \in \delta \Leftrightarrow \forall i \leq n, (g, i) \in \mathrm{Father}(E, f_i).$$

Notice that due to the linearity of E, $\mathcal{P}_E$ is deterministic.

Example 2. The Bottom-Up Position automaton $(\mathrm{Pos}(\overline{E}), \mathrm{Pos}(\overline{E}), \mathrm{Root}(\overline{E}), \delta)$ of the expression $\overline{E}$ defined in Example 1 is defined as follows:

$$\begin{aligned} \mathrm{Pos}(E) = \{a, b, f_1, g_2, f_3, g_4\}, \quad \mathrm{Root}(\overline{E}) = \{a, f_1, g_2\}, \\ \delta = \{(a,a), (b,b), ((a,a), f_1, f_1), ((a, f_1), f_1, f_1), ((a, g_2), f_1, f_1), ((f_1, a), f_1, f_1), \\ ((f_1, f_1), f_1, f_1), ((f_1, g_2), f_1, f_1), ((g_2, a), f_1, f_1), ((g_2, f_1), f_1, f_1), \\ ((g_2, g_2), f_1, f_1), (f_3, g_2, g_2), ((b, g_4), f_3, f_3), (a, g_4, g_4)\}. \end{aligned}$$

Let us now show that the Position automaton of E recognizes $L(E)$.

Lemma 4. *Let* $\mathcal{P}_E = (\Sigma, Q, Q_F, \delta)$ *be the Bottom-Up Position automaton of a linear expression* E *over a ranked alphabet* Σ*,* t *be a tree in* T_Σ *and* f *be a symbol in* $\mathrm{Pos}(E)$*. Then* ***(1)*** $f \in \Delta(t)$ *if and only if* ***(2)*** $\mathrm{root}(t) = f \wedge P(t)$*.*

Proof. Let us proceed by induction over the structure of $t = f(t_1, \ldots, t_n)$. By definition, $\Delta(t) = \delta(\Delta(t_1), \ldots, \Delta(t_n), f)$. For any state f_i in Δ_i, it holds from the induction hypothesis that

$$f_i \in \Delta(t_i) \Leftrightarrow \mathrm{root}(t_i) = f_i \wedge P(t_i). \tag{*}$$

Then, suppose that **(1)** holds (*i.e.* $f \in \Delta(t)$). Equivalently, there exists by definition of $\mathcal{P}_E$ a transition $((f_1, \ldots, f_n), f, f)$ in δ such that f_i is in $\Delta(t_i)$ for any integer $i \leq n$. Consequently, f is the root of t. Moreover, from the equivalence stated in Equation (*), $\mathrm{root}(t_i) = f_i$ and $P(t_i)$ holds for any integer $i \leq n$. Finally and equivalently, $P(t)$ holds as a consequence of Equation (3). The reciprocal condition can be proved similarly since only equivalences are considered.

□

As a direct consequence of Lemma 4 and Proposition 2,

Proposition 3. *The Bottom-Up Position automaton of a linear expression E recognizes $L(E)$.*

The Bottom-Up Position automaton of a (not necessarily linear) expression E can be obtained by first computing the Bottom-Up Position automaton of its linearized expression $\overline{E}$ and then by applying the alphabetical morphism h. As a direct consequence of Equation (5),

Proposition 4. *The Bottom-Up Position automaton of an expression E recognizes $L(E)$.*

5 Compressed Bottom-Up Position Automaton

In this section, we show that the structure of an expression allows us to factorize the transitions of a tree automaton by only considering the values of the Father function. The basic idea of the factorizations is to consider the cartesian product of sets. Imagine that a tree automaton contains four binary transitions (q_1, q_1, f, q_3), (q_1, q_2, f, q_3), (q_2, q_1, f, q_3) and (q_2, q_2, f, q_3). These four transitions can be factorized as a *compressed transition* $(\{q_1, q_2\}, \{q_1, q_2\}, f, q_3)$ using set of states instead of sets. The behavior of the original automaton can be simulated by considering the cartesian product of the origin states of the transition.

We first show how to encode such a notion of compressed automaton and how it can be used in order to solve the membership test.

Definition 3. *A* compressed tree automaton *over a ranked alphabet Σ is a 4-tuple (Σ, Q, Q_F, δ) where Q is a set* of states*, $Q_F \subset Q$ is the set of* final states*, $\delta \subset (2^Q)^n \times \Sigma_n \times 2^Q$ is the set of* compressed transitions *that can be seen as a function from $(2^Q)^k \times \Sigma_k$ to 2^Q defined by*

$$(Q_1, \ldots, Q_k, f, q) \in \delta \Leftrightarrow q \in \delta(Q_1, \ldots, Q_k, f).$$

Example 3. Let us consider the compressed automaton $A = (\Sigma, Q, Q_F, \delta)$ shown in Figure 1. Its transitions are

$$\delta = \{(\{1,2,5\}, \{3,4\}, f, 1), (\{2,3,5\}, \{4,6\}, f, 2), (\{1,2\}, \{3\}, f, 5), (\{6\}, g, 4), (\{6\}, g, 5), (a, 6), (a, 4), (b, 3)\}.$$

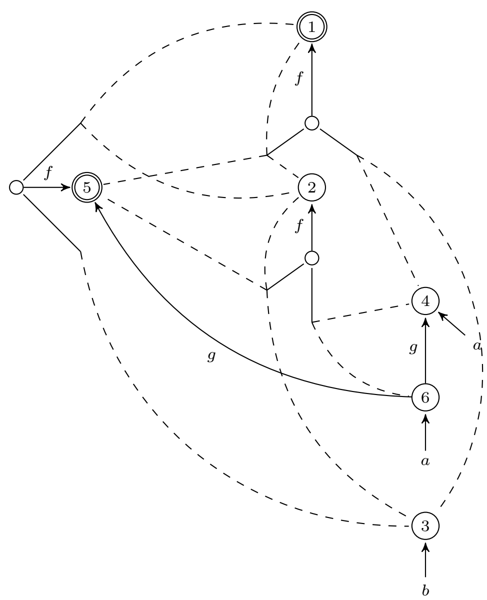

Fig. 1. The compressed automaton A.

The transition function δ can be restricted to a function from $Q^n \times \Sigma_n$ to 2^Q (*e.g.* in order to simulate the behavior of an uncompressed automaton) by considering for a tuple $(q_1, \ldots, q_k)$ of states and a symbol f in Σ_k all the "active" transitions $(Q_1, \ldots, Q_k, f, q)$, that are the transitions where q_i is in Q_i for $i \leq k$. More formally, for any states $(q_1, \ldots, q_k)$ in Q^k, for any symbol f in Σ_k,

$$\delta(q_1, \ldots, q_k, f) = \bigcup_{\substack{(Q_1, \ldots, Q_k, f, q) \in \delta, \\ \forall i \leq k, q_i \in Q_i}} \{q\}. \tag{9}$$

The transition set δ can be extended to a function Δ from $T(\Sigma)$ to 2^Q by inductively considering, for a tree $f(t_1, \ldots, t_k)$ the "active" transitions $(Q_1, \ldots, Q_k,\ f, q)$ once a subtree is read, that is when $\Delta(q_i)$ and Q_i admits a common state for $i \leq k$. More formally, for any tree $t = f(t_1, \ldots, t_k)$ in $T(\Sigma)$,

$$\Delta(t) = \bigcup_{\substack{(Q_1, \ldots, Q_k, f, q) \in \delta, \\ \forall i \leq k, \Delta(t_i) \cap Q_i \neq \emptyset}} \{q\}.$$

As a direct consequence of the two previous equations,

$$\Delta(f(t_1, \ldots, t_n)) = \bigcup_{(q_1, \ldots, q_n) \in \Delta(t_1) \times \cdots \times \Delta(t_n)} \delta(q_1, \ldots, q_n, f). \tag{10}$$

The *language recognized by* a compressed automaton $A = (\Sigma, Q, Q_F, \delta)$ is the subset $L(A)$ of $T(\Sigma)$ defined by

$$L(A) = \{t \in T(\Sigma) \mid \Delta(t) \cap Q_F \neq \emptyset\}.$$

Example 4. Let us consider the automaton of Figure 1 and let us show that the tree $t = f(f(b, a), g(a))$ belongs to $L(A)$. In order to do so, let us compute $\Delta(t')$ for each subtree t' of t. First, by definition,

$$\Delta(a) = \{4, 6\}, \qquad \Delta(b) = \{3\}.$$

Since the only transition in δ labeled by f containing 3 in its first origin set and 4 or 6 in its second is the transition $(\{2, 3, 5\}, \{4, 6\}, f, 2)$,

$$\Delta(f(b, a)) = \{2\}.$$

Since the two transitions labeled by g are $(\{6\}, g, 4)$ and $(\{6\}, g, 5)$,

$$\Delta(g(a)) = \{4, 5\}.$$

Finally, there are two transitions labeled by f containing 2 in their first origin and 4 or 5 in its second: $(\{2, 3, 5\}, \{4, 6\}, f, 2)$ and $(\{1, 2, 5\}, \{3, 4\}, f, 1)$. Therefore

$$\Delta(f(f(b, a), g(a)) = \{1, 2\}.$$

Finally, since 1 is a final state, $t \in L(A)$.

Let ϕ be an alphabetical morphism between two alphabets Σ and Σ'. The *image by* ϕ of a compressed automaton $A = (\Sigma, Q, Q_F, \delta)$ is the compressed automaton $\phi(A) = (\Sigma', Q, Q_F, \delta')$ where

$$\delta' = \{(Q_1, \ldots, Q_n, \phi(f), q) \mid (Q_1, \ldots, Q_n, f, q) \in \delta\}.$$

By a trivial induction over the structure of the trees, it can be shown that

$$L(\phi(A)) = \phi(L(A)). \tag{11}$$

Due to their inductive structure, regular expressions are naturally factorizing the structure of transitions of a Glushkov automaton. Let us now define the compressed Position automaton of an expression.

Definition 4. *The* compressed Bottom-Up Position automaton $\mathcal{C}(E)$ *of a linear expression* E *is the automaton* $(\Sigma, \mathrm{Pos}(E), \mathrm{Root}(E), \delta)$ *defined by*

$$\delta = \{(Q_1, \ldots, Q_k, f, \{f\}) \mid Q_i = \{g \mid (f, i) \in \mathrm{Father}(E, g)\}\}.$$

Example 5. Let us consider the expression $\overline{E}$ defined in Example 1. The compressed automaton of $\overline{E}$ is represented at Figure 2.

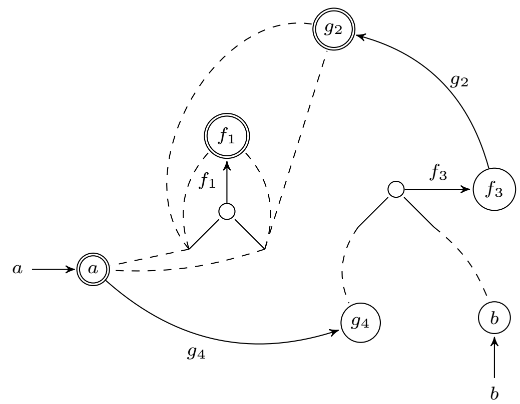

Fig. 2. The compressed automata of the expression $(f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b)$.

As a direct consequence of Definition 4 and of Equation (9),

Lemma 5. *Let E be a linear expression over a ranked alphabet Σ. Let $\mathcal{C}(E) = (\Sigma, Q, Q_F, \delta)$. Then, for any states $(q_1, \ldots, q_n)$ in Q^n, for any symbol f in Σ_k,*

$$\delta(q_1, \ldots, q_n, f) = \{f\} \Leftrightarrow \forall i \leq n, (f, i) \in \mathrm{Father}(E, q_i).$$

Consequently, considering Definition 2, Lemma 5 and Equation (10),

Proposition 5. *Let E be a linear expression over a ranked alphabet Σ. Let $\mathcal{P}_E = (\_, \_, \_, \delta)$ and $\mathcal{C}(E) = (\_, \_, \_, \delta')$. For any tree t in $T(\Sigma)$,*

$$\Delta(t) = \Delta'(t).$$

Since the Bottom-Up Position automaton of a linear expression E and its compressed version have the same states and the same final states,

Corollary 1. *The Glushkov automaton of an expression and its compact version recognize the same language.*

The compressed Bottom-Up Position automaton of a (not necessarily linear) expression E can be obtained by first computing the compressed Bottom-Up Position automaton of its linearized expression $\overline{E}$ and then by applying the alphabetical morphism h. Therefore, considering Equation (11),

Proposition 6. *The compressed Bottom-Up Position automaton of a regular expression E recognizes $L(E)$.*

6 The Father Automaton

In this section, we define the Father automaton associated with an expression that is an extension of the classical follow (word) automaton [10] that have been already extended in the case of (Top-Down) tree automata [15].

We embed the computation of the function Root in the function Father by adding a unary symbol \$ that is not in Σ at the top of the syntactic tree of an expression. Indeed,

$$f \in \mathrm{Root}(E) \Leftrightarrow (\$, 1) \in \mathrm{Father}(\$(E), f). \tag{12}$$

Equivalenty,

Lemma 6. *Let f be a state of the Position automaton of a linear expression E. The two following conditions hold:*

1. *f is a final state,*
2. *$(\$, 1)$ is in $\mathrm{Father}(\$(E), f)$.*

With this notation, the Father automaton is defined as follows.

Definition 5. *The* Father Automaton *of a linear expression E over an alphabet Σ is the automaton $\mathcal{F}_E = (\Sigma, Q, F, \delta)$ defined by*

$$Q = \{\mathrm{Father}(\$(E), f) \mid f \in \Sigma\}, \quad F = \{q \in Q \mid \$ \in q\},$$
$$((\mathrm{Father}(\$(E), f_1), \ldots, \mathrm{Father}(\$(E), f_n)), g, \mathrm{Father}(\$(E), g)) \in \delta \Leftrightarrow \forall i \leq n, (g, i) \in \mathrm{Father}(E, f_i).$$

Notice that due to the linearity of E, $\mathcal{P}_E$ is deterministic.

In the word case, it has been shown that the Follow automaton is a quotient of the Position automaton. Let us proceed in the same way in order to show, using Proposition 1, that the Father automaton of an expression E recognizes $L(E)$. Consequently, let us first extend the notion of congruence for tree automata.

Let $A = (\Sigma, Q, F, \delta)$ be a deterministic tree automaton and $\sim$ be an equivalence relation over Q such that for any two equivalent states p and p',

$$p \in F \Leftrightarrow p' \in F.$$

Given an equivalence relation $\sim$ over a set S, we denote by $S_\sim$ the set of equivalence classes of S and by $[p]_\sim$ (or $[p]$ when there is no ambiguity) the equivalence classe of an element p in P. By a notation abuse, we extend $\sim$ to the subsets of size 0 or 1 as follows:

$$\emptyset \sim \emptyset, \quad \forall s \in S, \emptyset \not\sim \{s\},$$
$$\forall s, s' \in S, \{s\} \sim \{s'\} \Leftrightarrow s \sim s'.$$

The relation $\sim$ is a *Bottom-Up congruence* for δ if and only if for any two states p and p' in Q, the two following conditions are equivalent:

1. $p \sim p'$,
2. for any symbol f in Σ_m, for any integer $n \leq m$, for any $m-1$ states $q_1, \ldots, q_{n-1}, q_{n+1}, \ldots, q_m$ in Q,

$$\delta((q_1, \ldots, q_{n-1}, p, q_{n+1}, \ldots, q_m), f) \sim \delta((q_1, \ldots, q_{n-1}, p', q_{n+1}, \ldots, q_m), f).$$

Two Bottom-Up congruent states can be said *interchangeable* in the litterature [1]. The *quotient automaton* of A w.r.t. $\sim$ is the automaton $A_\sim = (\Sigma, Q_\sim, F_\sim, \delta')$ with

$$\delta'(([q_1], \ldots, [q_m]), f) = \{\phi(q) \mid q \in \delta((q_1, \ldots, q_m), f)\}. \tag{13}$$

Since $A_\sim$ can be computed from a canonical morphism associated with ϕ, and as a direct corollary of Proposition 1,

Proposition 7. *Quotienting by a Bottom-Up congruence preserves the recognized language.*

Let us now show how to obtain the Father automaton by quotienting the Position automaton w.r.t. the following congruence.

Definition 6. *The* Father congruence *associated with a linear expression E over an alphabet Σ is the congruence $\sim$ defined by*

$$p \sim p' \Leftrightarrow \mathrm{Father}(\$(E), p) = \mathrm{Father}(\$(E), p').$$

Equivalently, the Father congruence is the *kernel* of the function sending a symbol p to $\mathrm{Father}(\$(E), p)$.

Proposition 8. *The Father congruence of a linear expression E is a Bottom-Up congruence for the transition function of the Position automaton of E.*

Proof. First, let us notice that following Lemma 6, two equivalent states have the same finality. Moreover, two states are equivalent if and only if they admit the same fathers. Consequently, from the construction of the Position automaton (Definition 2), for any two states p and p', the two following conditions are equivalent:

1. $p \sim p'$,
2. for any symbol f in Σ_m, for any integer $n \leq m$, for any $m-1$ states $q_1, \ldots, q_{n-1}, q_{n+1}, \ldots, q_m$ in Q,

$$\delta((q_1, \ldots, q_{n-1}, p, q_{n+1}, \ldots, q_m), f) = \delta((q_1, \ldots, q_{n-1}, p', q_{n+1}, \ldots, q_m), f).$$

Since $\sim$ is reflexive, it is a Bottom-Up congruence. $\square$

Proposition 9. *The Father Automaton associated with a linear expression E is isomorphic to the quotient of the Position automaton of E w.r.t. the Father congruence.*

Proof. Let us set

$$\mathcal{P}_E = (\_, \_, \_, \delta), \qquad \mathcal{P}_{E\sim} = (\_, \_, \_, \delta_\sim) \qquad \mathcal{F}_E = (\_, \_, \_, \delta').$$

Let us consider the functions ϕ and ϕ' defined as follows

$$\phi([f]) = \text{Father}(\$(E), f), \qquad \phi'(\text{Father}(\$(E), f)) = [f].$$

Notice that since $f \sim f' \Leftrightarrow \text{Father}(\$(E), f) = \text{Father}(\$(E), f')$, the functions are both well-defined. Moreover, they are trivially the inverse of each other. Furthermore, since

$$f \sim f' \Rightarrow ((\$, 1) \in \text{Father}(\$(E), f) \Leftrightarrow (\$, 1) \in \text{Father}(\$(E), f')),$$

ϕ preserves the finality of the states. Finally,

$$\begin{aligned}
(([f_1], \ldots, [f_n]), g, [g]) \in \delta_\sim &\Leftrightarrow ((f_1, \ldots, f_n), g, g) \in \delta && \textbf{(Equation (13))}\\
&\Leftrightarrow \forall i \leq n, (g, i) \in \text{Father}(E, f_i) && \textbf{(Definition 2)}\\
&\Leftrightarrow ((\text{Father}(\$(E), f_1), \ldots, \text{Father}(\$(E), f_n)), g, \text{Father}(\$(E), g)) \in \delta' && \textbf{(Definition 5)}.
\end{aligned}$$

Hence ϕ and ϕ' are two inverse automata morphisms between $\mathcal{P}_{E\sim}$ and $\mathcal{F}_E$. □

As a direct consequence of Proposition 7, Proposition 8 and Proposition 9,

Corollary 2. *The Father Automaton associated with a linear expression E recognized $L(E)$.*

Applying the delinearization morphism h from $\mathcal{F}_{\overline{E}}$ produces the Father automaton of any expression E. Finally,

Corollary 3. *The Father Automaton associated with an expression E recognizes $L(E)$.*

Example 6. The Father automaton $(\text{Pos}(\overline{E}), \text{Pos}(\overline{E})_\sim, \text{Root}(\overline{E})_\sim, \delta)$ of the expression $\overline{E}$ defined in Example 1 is obtained by merging the states f_1 and g_2 of $\mathcal{P}_E$, *i.e.*:

$$\begin{aligned}
&\text{Pos}(E) = \{[a], [b], \{f_1, g_2\}, [f_3], [g_4]\}, \quad \text{Root}(\overline{E}) = \{[a], [f_1]\},\\
\delta = \{&(a, [a]), (b, [b]), (([a], [a]), f_1, [f_1]), (([a], [f_1]), f_1, [f_1]), (([f_1], a), f_1, [f_1]),\\
&(([f_1], [f_1]), f_1, [f_1]), ([f_3], g_2, [g_2]), (([b], [g_4]), f_3, [f_3]), ([a], g_4, [g_4])\}.
\end{aligned}$$

7 The Compressed Father Automaton

Finally, let us show that similarly to the Position automaton, the Father automaton can be compressed too.

Definition 7. *The* compressed Father automaton $\mathcal{CF}(E)$ *of a linear expression E is the automaton* $(\Sigma, \text{Pos}(E), \text{Root}(E), \delta)$ *defined by*

$$\delta = \{(Q_1, \ldots, Q_k, f, \{\text{Father}(\$(E), f)\}) \mid Q_i = \{\text{Father}(\$(E), g) \mid (f, i) \in \text{Father}(\$(E), g)\}\}.$$

In order to show that $\mathcal{CF}(E)$ recognizes $L(E)$, we can apply the same method as for the Father automaton. First, due to Equation 9, the definition of a Bottom-Up congruence for A is exactly the same (Equation (2)).

The *quotient* of a compressed automaton $A = (\Sigma, Q, F, \delta)$ w.r.t. a Bottom-Up congruence $\sim$ is the compressed automaton $A_\sim = (\Sigma, Q_\sim, F_\sim, \delta')$ where

$$\delta'((Q_1, \ldots, Q_m), f) = \{\phi(q) \mid q \in \delta((q_1, \ldots, q_m), f) \wedge \forall i \leq m, [q_i] \in Q_i\}.$$

Similarly to Lemma 5 and Proposition 5, it can be shown that

Proposition 10. *The compressed Father automaton is a quotient of the compressed Position automaton w.r.t. the Father congruence.*

As a direct corollary,

Corollary 4. *The compressed Father automaton and the Father automaton of a linear expression E recognize $L(E)$.*

Applying the delinearization morphism h from $\mathcal{CF}_{\overline{E}}$ produce the compressed Father automaton of any expression E. Finally, according to Equation (11),

Proposition 11. *The compressed Father automaton and the Father automaton of an expression E recognize $L(E)$.*

Example 7. Let us consider the expression $\overline{E}$ defined in Example 1. The compressed Father automaton of $\overline{E}$ is represented at Figure 3.

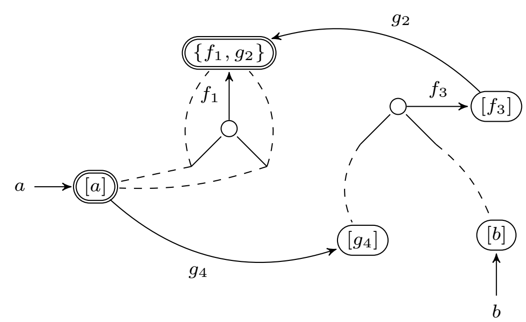

Fig. 3. The compressed Father automaton of the expression $(f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b)$.

8 Web Application

The computation of the position functions and the Glushkov and Father constructions have been implemented in a web application (made in Haskell, compiled into Javascript using the REFLEX PLATFORM, represented with VIZ.JS) in order to help the reader to manipulate the notions. From a regular expression, it computes the classical Top-Down Glushkov defined in [12], and both the normal and the compressed versions of the Glushkov and Father Bottom-Up automata.

This web application can be found here [14]. As an example, the expression $(f(a,a) + g(b))^{*_a} \cdot_b f(g(a), b)$ of Example 1 can be defined from the literal input `(f(a,a)+g(b))*a.bf(g(a),b)`.

9 Conclusion and Perspectives

In this paper, we have shown how to compute the Bottom-Up Position and Father automata associated with a regular expression. This construction is relatively similar to the classical ones defined over a word expression [8,10]. We have also proposed two reduced versions, the compressed Bottom-Up Position and Father automata, that can be easily defined for word expressions too.

Since this construction is related to the classical one, one can wonder if all the studies involving Glushkov word automata can be extended to tree ones ([4,5,6,16]). The classical Glushkov construction was also studied *via* its morphic links with other well-known constructions. The next step of our study is to extend Antimirov partial derivatives [2] in a Bottom-Up way too (in a different way from [11]), using the Bottom-Up quotient defined in [7].

Moreover, we can also consider the transition compression in a Top-Down way, that is to also aggregate the destination states. This way, the Follow tree automaton [15] can also be compressed too.

Finally, this compression method can lead to heuristic study in order to choose a good aggregation to optimize any reduction. To this aim, the decomposition techniques implemented to compute the Common Follow Set automaton [9] can be considered.

References

1. Adámek, J., Trnková, V.: Automata and Algebras in Categories. Mathematics and its Applications. Springer Netherlands (1990)
2. Antimirov, V.M.: Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. **155**(2) (1996) 291–319

3. Attou, S., Mignot, L., Ziadi, D.: The bottom-up position tree automaton and its compact version. In: accepted at CIAA 2018
4. Bouchou, B., Duarte, D., Alves, M.H.F., Laurent, D., Musicante, M.A.: Schema evolution for XML: A consistency-preserving approach. In: MFCS. Volume 3153 of Lecture Notes in Computer Science, Springer (2004) 876–888
5. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. Inf. Comput. **140**(2) (1998) 229–253
6. Caron, P., Ziadi, D.: Characterization of Glushkov automata. Theor. Comput. Sci. **233**(1-2) (2000) 75–90
7. Champarnaud, J., Mignot, L., Sebti, N.O., Ziadi, D.: Bottom-up quotients for tree languages. Journal of Automata, Languages and Combinatorics **22**(4) (2017) 243–269
8. Glushkov, V.M.: The abstract theory of automata. Russian Mathematical Surveys **16** (1961) 1–53
9. Hromkovic, J., Seibert, S., Wilke, T.: Translating regular expressions into small -free nondeterministic finite automata. J. Comput. Syst. Sci. **62**(4) (2001) 565–588
10. Ilie, L., Yu, S.: Follow automata. Inf. Comput. **186**(1) (2003) 140–162
11. Kuske, D., Meinecke, I.: Construction of tree automata from regular expressions. RAIRO - Theor. Inf. and Applic. **45**(3) (2011) 347–370
12. Laugerotte, É., Sebti, N.O., Ziadi, D.: From regular tree expression to position tree automaton. In: LATA 2013. (2013) 395–406
13. McNaughton, R.F., Yamada, H.: Regular expressions and state graphs for automata. IEEE Transactions on Electronic Computers **9** (March 1960) 39–57
14. Mignot, L.: Application: Glushkov tree automata. `http://ludovicmignot.free.fr/programmes/fatherBotUp/index.html` Accessed: 2018-05-18.
15. Mignot, L., Sebti, N.O., Ziadi, D.: Tree automata constructions from regular expressions: a comparative study. Fundam. Inform. **156**(1) (2017) 69–94
16. Nicaud, C.: On the average size of Glushkov's automata. In: LATA. Volume 5457 of Lecture Notes in Computer Science, Springer (2009) 626–637