Ahlem Belabbaci; Hadda Cherroun; Loek Cleophas; Djelloul Ziadi

Tree pattern matching from regular tree expressions

Kybernetika, Vol. 54 (2018), No. 2, 221–242

Persistent URL: http://dml.cz/dmlcz/147190

KYBERNETIKA — VOLUME *54* (2018), NUMBER 2, PAGES 221–242

TREE PATTERN MATCHING FROM REGULAR TREE EXPRESSIONS

Ahlem Belabbaci, Hadda Cherroun, Loek Cleophas and Djelloul Ziadi

In this work we deal with tree pattern matching over ranked trees, where the pattern set to be matched against is defined by a regular tree expression. We present a new method that uses a tree automaton constructed inductively from a regular tree expression. First we construct a special tree automaton for the regular tree expression of the pattern E, which is somehow a generalization of Thompson automaton for strings. Then we run the constructed automaton on the subject tree t. The pattern matching algorithm requires an $\mathcal{O}(|t||E|)$ time complexity, where $|t|$ is the number of nodes of t and $|E|$ is the size of the regular tree expression E. The novelty of this contribution besides the low time complexity is that the set of patterns can be infinite, since we use regular tree expressions to represent patterns.

Keywords: tree automata, Thompson Tree automata, regular tree expressions, tree pattern matching

Classification: 68Q45

1. INTRODUCTION

Tree pattern matching algorithms play an important role in many applications such as compilers, validation of XML documents, automatic proofs, etc. This problem can be defined as the search for all occurrences of one or more given trees (patterns) in a subject tree.

Tree pattern matching can be considered as an extension of string pattern matching. Several algorithms have appeared in the literature [9, 12, 20, 21, 25, 26, 29].

While most tree pattern matching literature appeared from the early 1980s onwards, the 1975 PhD thesis of Kron was an earlier exception [22]. Kron uses so-called orthogonal tree automata, automata that read the sequence of states assigned to the sequence of children of a node to determine the state to be assigned to the parent node.

In 1982, Hoffmann and O'Donnell in [20] presented a more extensive study of tree pattern matching. They proposed several algorithms solving the tree pattern matching problem for both bottom-up and top-down approaches. First, the bottom-up method generalizes string matching. The idea here is to find at each node in the subject tree, all patterns or parts of patterns matching this node. Interestingly, Hoffmann and O'Donnell do not mention bottom-up tree automata, even though their bottom-up method in

DOI: 10.14736/kyb-2018-2-0221

essence corresponds to tabulating such an automaton. Their top-down approach reduces tree matching to a string-matching problem by using a string-path matching automaton.

Another solution to the tree pattern matching problem exists. The principle is to encode the subject tree as strings, allowing the use of string pattern matching techniques. In [25] a generalization of the Knuth-Morris-Pratt algorithm of string pattern matching into trees is given. In the Knuth-Morris-Pratt algorithm the method used is the precomputation of shifts [29]. A more recent work using linearization is presented in [31], where a backward tree pattern matching technique is approached that uses ideas from Boyer-Moore and Horspool's string pattern matching algorithms.

Other techniques use a pushdown automaton [12, 26]. In [12] the method used is analogous to the construction of string pattern matchers: for given patterns, a non-deterministic pushdown automaton is created which is then determinized for efficiency reasons.

In a different way, Itokawa et al [21], use a depth-first unary degree sequence (DFUDS), which is a data structure associated with an ordered tree and expressing the features of a graph structure. They then propose a pattern matching algorithm that uses the DFUDS data structure to determine whether or not a given tree has features of a tree pattern.

All the previously discussed tree pattern matching algorithms consider the pattern as a tree or a *finite* set of trees. In our work, we focus on a more generalized problem by using patterns represented by a regular tree expression.

A basic and well-known algorithm for regular expression pattern matching for strings is Thompson algorithm [30]. The principle of this algorithm consists of building, by induction, an automaton from the regular expression of the pattern and in a second step to process the subject text using the constructed automaton in order to identify the different pattern occurrences. This algorithm has been used in many practical tools such as the *grep* utility on Linux [1, 17].

In this paper we propose a tree pattern matching algorithm inspired by Thompson pattern matching for strings. This algorithm is based on the construction of a special tree automaton that can be viewed as a generalization of Thompson one for strings. Our proposal might be useful in all fields that need a lookup for one or multiple patterns in a subject tree, especially when these patterns are represented by a regular tree expression. For example: instruction selection in automatic code generation [2, 13], genetics [11, 15], term rewriting [19, 20], verification of network protocol and cryptography [14, 18].

A similar method constructing an automaton from a regular tree expression was described in [28], in which the authors use rather a different sort of automata (pushdown automata) and require a linearization of the trees involved; that is, the automata used there do not directly process trees but instead an encoding of trees into strings (postfix notation), a level of indirectness not required in our approach.

In the case of words, several algorithms were proposed in order to convert a regular expression into an automaton. The most common construction is the standard or position automaton [16, 27]. Brzozowski's construction [6] of a deterministic finite automaton uses derivatives of regular expressions. This approach was modified by Antimirov [3] who defined partial derivatives to construct a non-deterministic automaton from a regular expression E. Another construction was proposed by Thompson [30] based on induction over the structure of a regular expression.

By analogy to words, some algorithms were proposed for trees. Among these works is the one of Laugerotte et al. [24], who gave an algorithm to compute the position tree automaton. The work of Kuske and Meinecke [23] consists of the definition of partial derivatives for regular tree expressions and then building a non-deterministic finite tree automaton recognizing the language denoted by such an expression. They adapt and modify the approach of Champarnaud and Ziadi [7, 8] in the word case. Tree derivatives were introduced by Levine in [4, 5] and extend the concept of Brzozowski's string derivatives.

The rest of the paper is organized as follows. In Section 2, some preliminaries are presented. In Section 3, we give the inductive construction of the generalization of Thompson automaton to the tree case. In Section 4 we describe our algorithm for tree pattern matching with its complexity. Section 5 concludes the paper.

2. PRELIMINARIES

Let (Σ, ar) be *a ranked alphabet*, where Σ is a finite set of symbols and ar represents the *rank* of Σ which is a mapping from Σ into $\mathbb{N}$. The set of symbols of rank n is denoted by Σ_n. The elements of rank 0 are called *constants*. A *tree* t over Σ is inductively defined as follows: $t = a,\ t = f(t_1, \ldots, t_k)$ where a is a constant, k is any integer satisfying $k \geq 1$, f is any symbol in Σ_k and $t_1, \ldots, t_k$ are any k trees over Σ. We denote by $|t|$ the number of nodes of a tree t and by T_Σ the set of trees over Σ. *A tree language* is a subset of T_Σ. For our Thompson tree automaton-based pattern matching algorithm later in this paper, we want to index the nodes of the subject tree. To do so, we mark each node as follows: $\mathrm{Mark}(f) = f_1$ if f is the root symbol, $\mathrm{Mark}(f) = f_{\mathrm{Mark}(Parent(f))\mathrm{pos}(f)}$ otherwise, where $\mathrm{pos}(f)$ is the position of a node f among its sibling. For example, let $t = f(f(a, b), h(g(d)))$ be a tree. After marking t , we get the following indexed tree: $f_1(f_{11}(a_{111}, b_{112}), h_{12}(g_{121}(d_{1211})))$. Let Σ_M be the set of marked symbols of Σ. We define the mapping h from Σ_M to Σ, which for a marked symbol $f_{u_1 \ldots u_k}$ gives its corresponding symbol in Σ, that is f.

A (Bottom Up) Finite Tree Automaton $\mathcal{A}$ is a tuple (Q, Σ, Q_T, Δ) where Q is a finite set of states, $Q_T \subseteq Q$ is the set of *final states* and $\Delta \subseteq \bigcup_{n \geq 0}(Q \times \Sigma_n \times Q^n)$ is the set of *transition rules* [10, 23]. This set is equivalent to the function Δ from $Q^n \times \Sigma_n$ to 2^Q defined by $(q, f, q_1, \ldots, q_n) \in \Delta \Leftrightarrow q \in \Delta(q_1, \ldots, q_n, f)$. The domain of this function can be extended to $(2^Q)^n \times \Sigma_n$ as follows: $\Delta(Q_1, \ldots, Q_n, f) = \bigcup_{(q_1, \ldots, q_n) \in Q_1 \times \cdots \times Q_n} \Delta(q_1, \ldots, q_n, f)$. Finally, we denote by Δ^* the function from T_Σ to 2^Q defined for any tree in T_Σ as follows: $\Delta^*(t) = \Delta(a)$ if $t = a$ with $a \in \Sigma_0$, $\Delta^*(t) = \Delta(f, \Delta^*(t_1), \ldots, \Delta^*(t_n))$ $\Delta^*(t) = \Delta(\Delta^*(t_1), \ldots, \Delta^*(t_n), f)$ if $t = f(t_1, \ldots, t_n)$ with $f \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma$. A tree is *accepted* by $\mathcal{A}$ if and only if $\Delta^*(t) \cap Q_T \neq \emptyset$. The language $\mathcal{L}(\mathcal{A})$ *recognized* by $\mathcal{A}$ is the set of trees accepted by $\mathcal{A}$, that is $\mathcal{L}(\mathcal{A}) = \{t \in T_\Sigma \mid \Delta^*(t) \cap Q_T \neq \emptyset\}$.

The tree substitution of the constant $c \in \Sigma_0$ by the language $L \subseteq T_\Sigma$ in the tree $t \in T_\Sigma$, denoted by $t\{c \leftarrow L\}$, is the tree language L if $t = c$; the language $\{d\}$ if $t = d$ where $d \in \Sigma_0$ and $(d \neq c)$; and finally the language $f(t_1\{c \leftarrow L\}, \ldots, t_n\{c \leftarrow L\})$ if $t = f(t_1, \ldots, t_n)$. Then, the *c−product* language $L_1 \cdot_c L_2$ of two languages $L_1, L_2 \subseteq T_\Sigma$ is defined as: $L_1 \cdot_c L_2 = \bigcup_{t \in L_1} \{t\{c \leftarrow L_2\}\}$. The sequence of successive iterations is defined for $L \subseteq T_\Sigma$ as: $L^{0_c} = \{c\}$ and $L^{(n+1)_c} = L^{n_c} \cup L \cdot_c L^{n_c}$. The $c-$closure of L is defined as

$L^{*_c} = \bigcup_{n\geq 0} L^{n_c}$. The constant c is called the symbol of the $c-$*closure operator*.

A *regular tree expression* E over a ranked alphabet Σ is inductively defined by $\mathrm{E} = 0$, $\mathrm{E} \in \Sigma_0$, $\mathrm{E} = f(\mathrm{E}_1, \cdots, \mathrm{E}_n)$, $\mathrm{E} = \mathrm{E}_1 + \mathrm{E}_2$, $\mathrm{E} = \mathrm{E}_1 \cdot_c \mathrm{E}_2$, $\mathrm{E} = {\mathrm{E}_1}^{*_c}$, where $c \in \Sigma_0$, $n \in \mathbb{N}$, $f \in \Sigma_n$ and $\mathrm{E}_1, \mathrm{E}_2, \ldots, \mathrm{E}_n$ are any n regular expressions over Σ. We call $\mathrm{E} = f(\mathrm{E}_1, \cdots, \mathrm{E}_n)$ the arity operation. We denote by $|E|$ the size of the regular tree expression E. Every regular tree expression E can be seen as a tree over the ranked alphabet $\Sigma \cup \{+, \cdot_c, *_c \mid c \in \Sigma_0\}$ where $+$ and $\cdot_c$ can be seen as symbols of rank 2 and $*_c$ has rank 1. The *language* $[\![\mathrm{E}]\!]$ *denoted by* E is inductively defined by $[\![0]\!] = \emptyset$, $[\![c]\!] = \{c\}$, $[\![f(\mathrm{E}_1, \mathrm{E}_2, \cdots, \mathrm{E}_n)]\!] = f([\![\mathrm{E}_1]\!], \ldots, [\![\mathrm{E}_n]\!])$, $[\![\mathrm{E}_1 + \mathrm{E}_2]\!] = [\![\mathrm{E}_1]\!] \cup [\![\mathrm{E}_2]\!]$, $[\![\mathrm{E}_1 \cdot_c \mathrm{E}_2]\!] = [\![\mathrm{E}_1]\!] \cdot_c [\![\mathrm{E}_2]\!]$, $[\![{\mathrm{E}_1}^{*_c}]\!] = [\![\mathrm{E}_1]\!]^{*_c}$ where $n \in \mathbb{N}$, $\mathrm{E}_1, \mathrm{E}_2, \ldots, \mathrm{E}_n$ are any n regular expressions, $f \in \Sigma_n$ and $c \in \Sigma_0$. A tree language is accepted by some tree automaton if and only if it can be denoted by a regular tree expression [10, 23].

A tree pattern (for sake of simplicity we use pattern) is a tree in which variable leaves may exist, that is leaves with a symbol which can be substituted by any other sub-tree of the tree language. This symbol is referred to as v. Tree pattern matching problem is the identification of occurrences of one or more tree patterns in a subject tree. For each tree $t \in T_\Sigma$ and each pattern $p \in T_{\Sigma\cup\{v\}}$, a sub-tree in t matches the pattern p in a node n means that $p = t/n$, where t/n is the sub-tree of t rooted by n.

The graphical representation of the tree automaton is similar to the one of strings. The states are represented by circles with double circles for the final states, and transitions between states by edges labeled with a symbol of the alphabet Σ_0 or ε. For tree automata, some changes are made in the representation of transitions. A transition from state q_0 to states $q_1, q_2, ..., q_n$ with a symbol or an ε-transition is represented by i) an edge connecting the state q_0 to a small circle unlabeled with a symbol or ε, ii) n edges connecting the small circle to the state q_i labeled by i where $i : 1..n$. In the case of directed automata (top-down or bottom- up) edges are directed [9] (see for example Figure 12).

3. CONSTRUCTION OF TREE AUTOMATON FROM A REGULAR TREE EXPRESSION

Before presenting the proposed construction, we introduce some notations and definitions. We add a symbol ε of rank 1 to the alphabet Σ. This symbol has the same meaning as ϵ in the case of strings. For example, the tree $f(\varepsilon(\varepsilon(a)), b)$ is equal to $f(a, b)$.

Definition 3.1. Let t be a tree, we define the function $\varepsilon-$closure(t) that removes ε-nodes from t as follows:

$$\begin{aligned} \varepsilon-\text{closure}(a) &= a \quad \text{for each } a \text{ in } \Sigma_0 \\ \varepsilon-\text{closure}(\varepsilon(t)) &= \varepsilon-\text{closure}(t) \\ \varepsilon-\text{closure}(g(t_1, \ldots, t_n)) &= g(\varepsilon-\text{closure}(t_1), \ldots, \varepsilon-\text{closure}(t_n)). \end{aligned}$$

From the definition of $\varepsilon-$closure(t) we can deduce the following properties:

Property 3.2. Let t_1, $t_2 \in T_\Sigma$. We have then

$$\varepsilon-\text{closure}(t_1 \cdot_c t_2) = \varepsilon-\text{closure}(t_1) \cdot_c \varepsilon-\text{closure}(t_2).$$

This property can be extended to sets of trees.

Property 3.3. Let $s, t_1, \ldots, t_n \in T_\Sigma$. We have then

$$\varepsilon\text{−closure}(s \cdot_c \{t_1, \ldots, t_n\}) \quad = \quad \varepsilon\text{−closure}(s) \cdot_c \{\varepsilon\text{−closure}(t_1), \ldots, \varepsilon\text{−closure}(t_n)\}.$$

Given the difference between strings and trees concerning the concatenation and closure operations, we should take care when constructing a tree automaton. Indeed, we have designed a special form of tree automaton that allows us to inductively construct a tree automaton from a regular tree expression in a straightforward way. For the sake of simplicity we will use the name Thompson tree automaton to refer to this construction.

The basic idea of our construction is to build, from a given regular tree expression E, a finite bottom-up tree automaton which has the form illustrated by Figure 1. The main characteristic of this automaton is that it contains one initial state for each element of Σ_0 (the frame Q_{Σ_0}). This condition makes more sense for dealing with the concatenation operation, since as a result we have to perform concatenation in just one state.

In order to keep this form, some ε-transitions are added during the inductive constructions.

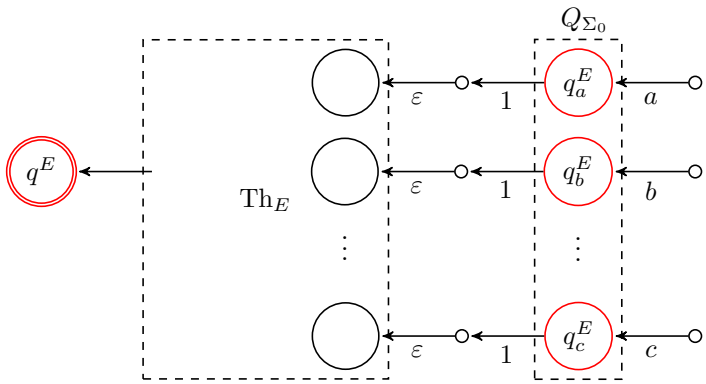


Fig. 1: General Form of Thompson Tree Automaton.

Let E be a regular tree expression. The generalized Thompson automaton $\mathrm{Th}_E = (Q^E, \Sigma \cup \{\varepsilon\}, \{q^E\}, \Delta^E)$ over the alphabet $\Sigma \cup \{\varepsilon\}$ associated with E is defined inductively as follows. Let Th_F, Th_G and Th_{E_i}, be the generalized Thompson automaton associated respectively with the tree expressions F, G and E_i, for $i = 1 \ldots n$, then:

The empty language $E = 0$ (Figure 2): $Q^E = \{q^E\}$ and $\Delta_E = \emptyset$

Fig. 2: The Empty Language Thompson Tree Automaton.

The leaf tree $E = a$, $a \in \Sigma_0$ (Figure 3): $Q^E = \{q^E\}$ and $\Delta_E = \{a \to q^E\}$

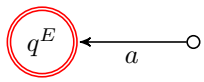


Fig. 3: The Leaf Tree Thompson Automaton.

For the purpose of clarity, we refer hereafter to the set $\Delta^X \setminus \{a \to q_a^X \mid a \in \Sigma_0\}$ by $\Delta^{>0,X}$, where X is a regular tree expression.

The arity function $E = f(E_1, \ldots, E_n)$ (Figure 4):
$Q^E = \bigcup_{i=1}^n Q^{E_i} \cup \{q^E\} \cup \{q_a^E \mid a \in \Sigma_0\}$ and

$$\begin{aligned}\Delta^E = &\bigcup_{i=1}^{n} (\Delta^{>0,E_i}) \cup \{a \to q_a^E \mid a \in \Sigma_0\} \\ &\cup \{f(q^{E_1}, q^{E_2}, \ldots, q^{E_n}) \to q^E\} \cup \{\varepsilon(q_a^E) \to q_a^{E_i} \mid a \in \Sigma_0\,, i = 1 \ldots n\}.\end{aligned}$$

The union $E = F + G$ (Figure 5): $Q^E = Q^F \cup Q^G \cup \{q^E\} \cup \{q_a^E \mid a \in \Sigma_0\}$ and

$$\begin{aligned}\Delta^E = &\Delta^{>0,F} \cup \Delta^{>0,G} \cup \{a \to q_a^E \mid a \in \Sigma_0\} \\ &\cup \{\varepsilon(q_a^E) \to q_a^F \mid a \in \Sigma_0\} \cup \{\varepsilon(q_a^E) \to q_a^G \mid a \in \Sigma_0\} \\ &\cup \{\varepsilon(q^F) \to q^E\} \cup \{\varepsilon(q^G) \to q^E\}.\end{aligned}$$

The concatenation $E = F \cdot_c G$ (Figure 6): $Q^E = Q^F \cup Q^G \cup \{q^E\} \cup \{q_a^E \mid a \in \Sigma_0\}$ and

$$\begin{aligned}\Delta^E = &\Delta^{>0,F} \cup \Delta^{>0,G} \cup \{a \to q_a^E \mid a \in \Sigma_0\} \\ &\cup \{\varepsilon(q_a^E) \to q_a^G \mid a \in \Sigma_0\} \cup \{\varepsilon(q_a^E) \to q_a^F \mid a \in \Sigma_0 \setminus \{c\}\} \\ &\cup \{\varepsilon(q^G) \to q_c^F\} \cup \{\varepsilon(q^F) \to q^E\}.\end{aligned}$$

The closure $E = F^{*_c}$ (Figure 7): $Q^E = Q^F \cup \{q^E\} \cup \{q_a^E \mid a \in \Sigma_0\}$ and

$$\begin{aligned}\Delta^E = &\Delta^{>0,F} \cup \{a \to q_a^E \mid a \in \Sigma_0\} \\ &\cup \{\varepsilon(q_c^E) \to q^E\} \cup \{\varepsilon(q^F) \to q_c^F\} \cup \{\varepsilon(q^F) \to q^E\} \\ &\cup \{\varepsilon(q_a^E) \to q_a^F \mid a \in \Sigma_0\}.\end{aligned}$$

Let $q \in Q$ be a state. Let $Q_-^\varepsilon(q) = \{p \in Q \mid \varepsilon(p) \longrightarrow q\}$.
From the construction of Thompson tree automata, we deduce the following property.

Property 3.4. For a state $q \in Q$, we have $|Q_-^\varepsilon(q)| \leqslant 2$.

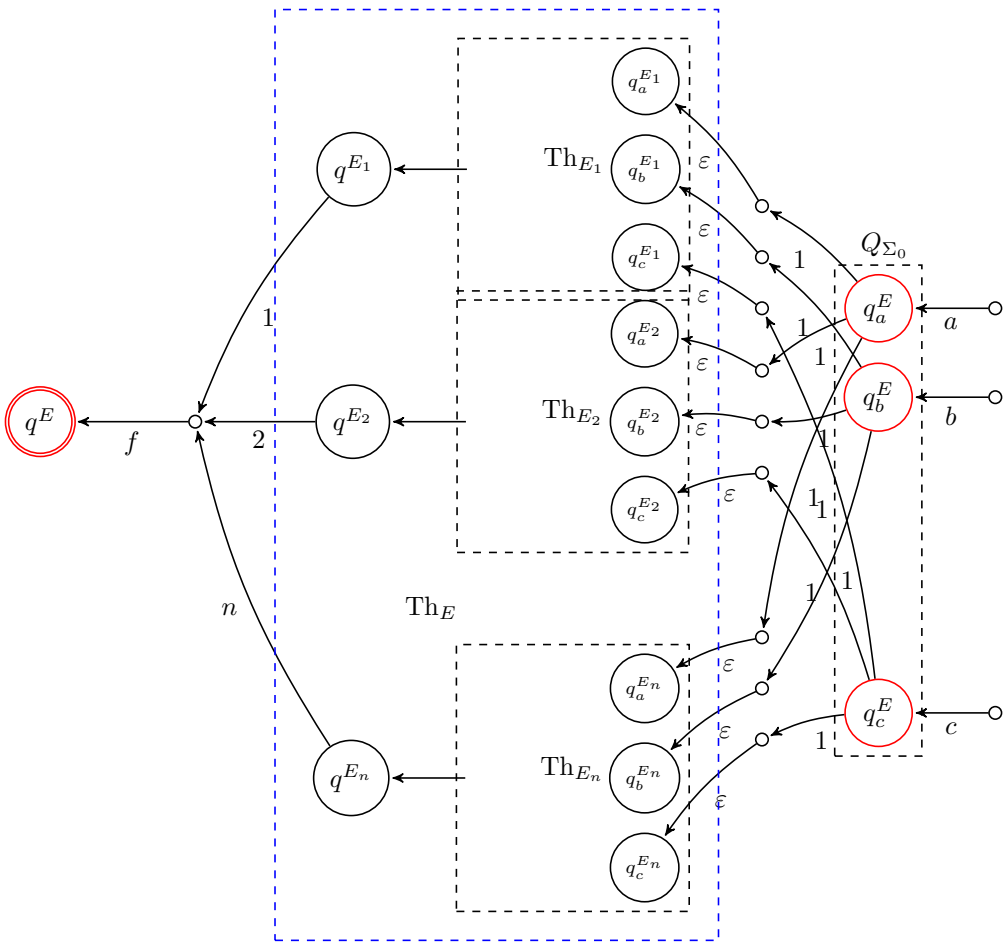


Fig. 4: Thompson Tree Automaton For $E = f(E_1, E_2, \ldots, E_n)$.

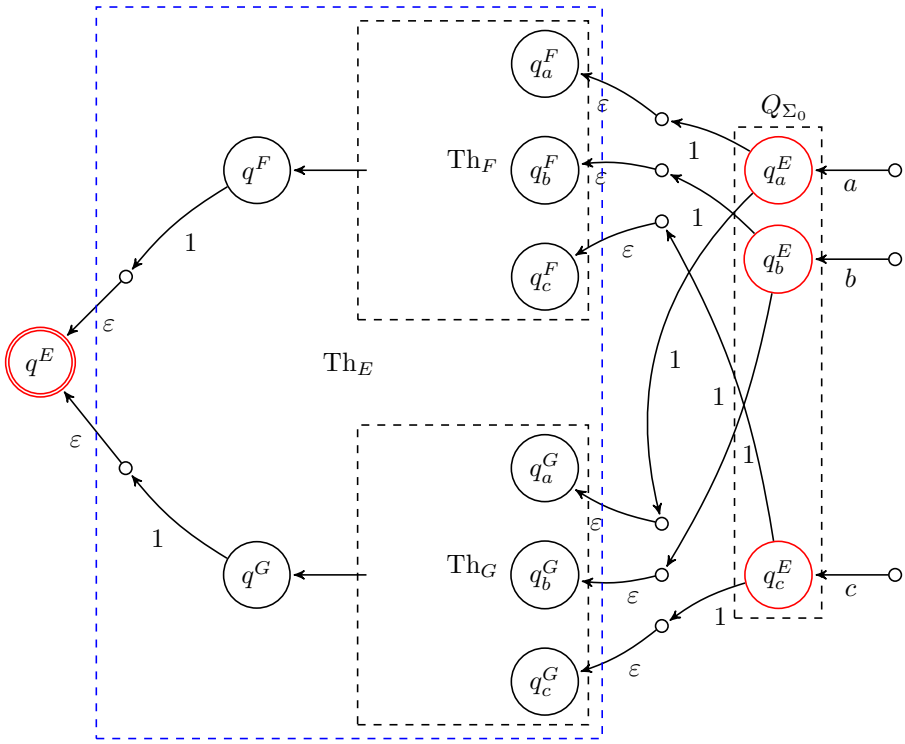


Fig. 5: Thompson Tree Automaton for $E = F + G$.

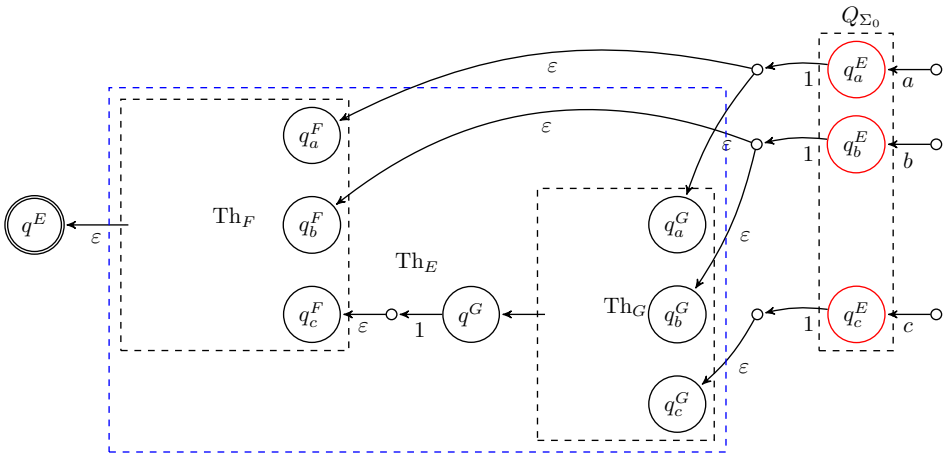


Fig. 6: Thompson Tree Automaton for $E = F \cdot_c G$.

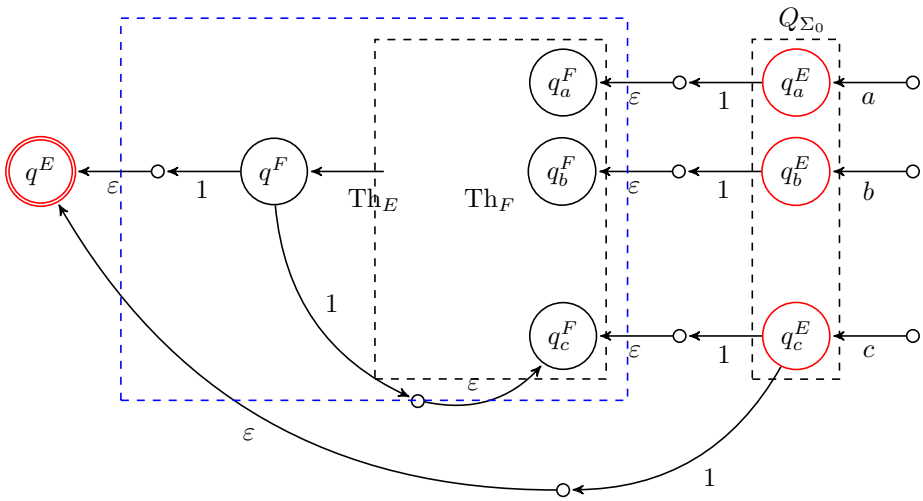


Fig. 7: Thompson Tree Automaton for $E = F^{*,c}$.

Theorem 3.5. For a regular tree expression E, $\varepsilon - closure(\mathcal{L}(\mathrm{Th}_E)) = [\![E]\!]$.

Before proving this theorem, we prove the two next propositions.
Let $\Sigma_0 = \{x_1, x_2, \cdots, x_n\}$ and $t, \overline{t}, \widetilde{t} \in T_\Sigma$ such that

$$\overline{t} = \Big(\cdots \Big(\big(t \cdot_{x_1} \varepsilon(x_1) \big) \cdot_{x_2} \varepsilon(x_2) \Big) \cdots \Big) \cdot_{x_n} \varepsilon(x_n)$$

and

$$\widetilde{t} = \Big\{ \cdots \Big\{ \big\{ t\{\varepsilon(x_1) \leftarrow x_1\} \big\} \varepsilon(x_2) \leftarrow x_2 \Big\} \cdots \Big\} \varepsilon(x_n) \leftarrow x_n.$$

Let $\mathcal{A}$, $\overline{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$ be tree automata such that $\mathcal{A} = (Q, \Sigma, q^{\mathcal{A}}, \Delta), \overline{\mathcal{A}} = (\overline{Q}, \Sigma, q^{\overline{\mathcal{A}}}, \overline{\Delta})$ and $\widetilde{\mathcal{A}} = (\widetilde{Q}, \Sigma, q^{\widetilde{\mathcal{A}}}, \widetilde{\Delta})$, where

$$\begin{aligned}
&\overline{Q} = Q \cup \{ {q'}_a^{\mathcal{A}} \mid \forall\, a \in \Sigma_0 \wedge q_a^{\mathcal{A}} \in Q \} \\
&q^{\overline{\mathcal{A}}} = q^{\mathcal{A}} \\
&\overline{\Delta} = \Delta \setminus \{ a \to q_a^{\mathcal{A}} \mid a \in \Sigma_0 \wedge q_a^{\mathcal{A}} \in Q \} \\
&\quad \cup \{ a \to {q'}_a^{\mathcal{A}} \mid a \in \Sigma_0 \wedge {q'}_a^{\mathcal{A}} \in \overline{Q} \} \\
&\quad \cup \{ \varepsilon({q'}_a^{\mathcal{A}}) \to q_a^{\mathcal{A}} \}
\end{aligned}$$

and

$$\begin{aligned}
&\widetilde{Q} = Q \setminus \{ {q'}_a^{\mathcal{A}}, \mid \forall\, a \in \Sigma_0 \wedge {q'}_a^{\mathcal{A}} \in Q \} \\
&q^{\widetilde{\mathcal{A}}} = q^{\mathcal{A}}
\end{aligned}$$

$$\begin{aligned}\widetilde{\Delta} = \Delta \setminus \{a \to {q'}_a^{\mathcal{A}} \mid a \in \Sigma_0 \wedge {q'}_a^{\mathcal{A}} \in Q\}\\ \setminus \{\varepsilon({q'}_a^{\mathcal{A}}) \to q_a^{\mathcal{A}} \mid q_a^{\mathcal{A}} \in Q\}\\ \cup \{a \to q_a^{\mathcal{A}} \mid a \in \Sigma_0 \wedge q_a^{\mathcal{A}} \in Q\}.\end{aligned}$$

These automata are introduced in order to improve the readability of the proofs of Theorem 3.5. $\overline{\mathcal{A}}$ is the automaton to which we have added ε-transitions after initial transitions, and $\widetilde{\mathcal{A}}$ is the one from which we have removed these ε-transitions.

Proposition 3.6. If $t \in \mathcal{L}(\mathcal{A})$, then $\overline{t} \in \mathcal{L}(\overline{\mathcal{A}})$.

Proof. We have $t \in \mathcal{L}(\mathcal{A})$, so $\Delta^*_{\mathcal{A}}(t) = q^{\mathcal{A}}$. According to the construction of $\overline{\mathcal{A}}$, every transition of the form $a \to q_a^{\mathcal{A}}$ such that $a \in \Sigma_0 \wedge q_a^{\mathcal{A}} \in Q$ in the path recognizing t in $\mathcal{A}$ is replaced by the two transitions $a \to {q'}_a^{\mathcal{A}}$ and $\varepsilon({q'}_a^{\mathcal{A}}) \to q_a^{\mathcal{A}}$ with $a \in \Sigma_0 \wedge {q'}_a^{\mathcal{A}} \in \overline{Q}$. Then $\overline{\Delta}^*(\overline{t}) = q^{\mathcal{A}} = q^{\overline{\mathcal{A}}}$, that is $\overline{t} \in \mathcal{L}(\overline{\mathcal{A}})$. □

Remark 3.7. Using Property 3.2, it is obvious that $\varepsilon{-}\text{closure}(t) = \varepsilon{-}\text{closure}(\overline{t})$.

$$\begin{aligned}\varepsilon{-}\text{closure}(\overline{t}) &= \varepsilon.-\text{closure}\Big(\Big(\Big((t \cdot_{x_1} \varepsilon(x_1)) \cdot_{x_2} \varepsilon(x_2)\Big)\cdots\Big) \cdot_{x_n} \varepsilon(x_n)\Big)\\ &= \Big(\cdots\Big(\Big(\varepsilon{-}\text{closure}(t) \cdot_{x_1} \varepsilon{-}\text{closure}(\varepsilon(x_1))\Big) \cdot_{x_2} \varepsilon{-}\text{closure}(\varepsilon(x_2))\Big)\\ &\qquad \cdots\Big) \cdot_{x_n} \varepsilon{-}\text{closure}(\varepsilon(x_n))\\ &= \Big(\cdots\Big(\Big(\varepsilon{-}\text{closure}(t) \cdot_{x_1} (x_1)\Big) \cdot_{x_2} (x_2)\Big)\cdots\Big) \cdot_{x_n} (x_n)\\ &= \varepsilon{-}\text{closure}(t).\end{aligned}$$

Proposition 3.8. If $t \in \mathcal{L}(\mathcal{A})$, then $\widetilde{t} \in \mathcal{L}(\widetilde{\mathcal{A}})$.

Proof. We have $t \in \mathcal{L}(\mathcal{A})$, that is $\Delta^*_{\mathcal{A}}(t) = q^{\mathcal{A}}$. According to the construction of the automaton $\widetilde{\mathcal{A}}$, in the path recognizing t in $\mathcal{A}$ we substitute all transitions of the form $a \to {q'}_a^{\mathcal{A}}$ and $\varepsilon({q'}_a^{\mathcal{A}}) \to q_a^{\mathcal{A}}$ with $a \in \Sigma_0 \wedge {q'}_a^{\mathcal{A}} \in Q$ by the transition $a \to q_a^{\mathcal{A}}$ for each $a \in \Sigma_0 \wedge q_a^{\mathcal{A}} \in Q$. Then $\widetilde{\Delta}^*(\widetilde{t}) = q^{\mathcal{A}} = q^{\widetilde{\mathcal{A}}}$, that is $\widetilde{t} \in \mathcal{L}(\widetilde{\mathcal{A}})$. □

Remark 3.9. Like Remark 3.7, it is clear that $\varepsilon{-}\text{closure}(t) = \varepsilon{-}\text{closure}(\widetilde{t})$.

In order to prove Theorem 3.5, we prove the two next lemmas. This is accomplished by induction on the structure of the tree automaton using Propositions 3.6 and 3.8. We give the proof for the cases of a leaf tree, arity and concatenation operations. The case of union is straightforward and the closure is similar to the concatenation.

Lemma 3.10. For each tree $t \in [\![E]\!]$, there exists a tree $t' \in \mathcal{L}(\text{Th}_E)$ such that $\varepsilon{-}\text{closure}(t') = t$.

Proof.
Case $E = a$ Let $t \in [\![E]\!]$, $t = a$. From Thompson leaf tree automaton construction, we have $\mathcal{L}(\mathrm{Th}_E) = \{a\}$, so $t' = a$. We have also $\varepsilon-\mathrm{closure}(t') = \varepsilon-\mathrm{closure}(a) = t$.

Case $E = g(E_1, E_2, \ldots, E_n)$ where n is the rank of g. Let $t \in [\![E]\!]$, so $t = g(e_1, e_2, \ldots, e_n)$, with $t_i \in [\![E_i]\!]$ and $1 < i \leqslant n$. According to the induction hypothesis, there exist $t'_i \in \mathcal{L}(\mathrm{Th}_{E_i})$ such that $\varepsilon-\mathrm{closure}(t'_i) = t_i$ with $i = 1 \ldots n$. We assume that $\overline{t}_i = t'_i$ since according to the construction of Thompson automaton generalization, t'_i has the same structure as $\overline{t}_i$. This assumption will be used in the concatenation case as well.

According to the construction of Thompson automaton for the arity, and using Proposition 3.6, we have $\overline{\Delta}^*(\overline{t_i}) = q^{E_i}$ for $i = 1 \cdots n$. Moreover, we have $g(q^{E_1}, q^{E_2}, \ldots, q^{E_n}) \to q^E \in \Delta$, then $\Delta^*(\overline{t}) = q^E$ where $\overline{t} = g(\overline{t_1}, \overline{t_2}, \cdots, \overline{t_n})$, that is $\overline{t} \in \mathcal{L}(\mathrm{Th}_E)$. We show that $\varepsilon-\mathrm{closure}(\overline{t}) = t$. From Remark 3.7, we have $\varepsilon-\mathrm{closure}(\overline{t_i}) = \varepsilon-\mathrm{closure}(t_i)$, so

$$\varepsilon-\mathrm{closure}(\overline{t}) = \varepsilon-\mathrm{closure}(g(\overline{t_1}, \overline{t_2}, \cdots, \overline{t_n})).$$

Using Property 3.2, we get

$$\varepsilon-\mathrm{closure}(\overline{t}) = g(\varepsilon-\mathrm{closure}(\overline{t_1}), \varepsilon-\mathrm{closure}(\overline{t_2}), \cdots, \varepsilon-\mathrm{closure}(\overline{t_n})).$$

Using Remark 3.7, we have

$$\varepsilon-\mathrm{closure}(\overline{t}) = g(\varepsilon-\mathrm{closure}(t_1), \varepsilon-\mathrm{closure}(t_2), \cdots, \varepsilon-\mathrm{closure}(t_n)).$$

Using the induction hypothesis, we get

$$\varepsilon-\mathrm{closure}(\overline{t}) = g(t_1, t_2, \cdots, t_n) = t.$$

Case $E = F \cdot_c G$ Let $t \in [\![E]\!]$, $t \in [\![F]\!] \cdot_c [\![G]\!]$ means $t \in \{(t_f) \cdot_c \{t_1, \ldots, t_k\}\}$, such that $t_i \in [\![G]\!]$, $i = 1 \ldots k$ and $t_f \in [\![F]\!]$. According to the induction hypothesis, there exist $t'_f, t'_1, \ldots, t'_k$ with $t'_f \in \mathcal{L}(\mathrm{Th}_F)$ and $t'_i \in \mathcal{L}(\mathrm{Th}_G)$, $i = 1 \ldots k$, such that $\varepsilon-\mathrm{closure}(t'_f) = t_f$ and $\varepsilon-\mathrm{closure}(t'_k) = t_k$. Let $\overline{t}_i = t'_i$, $\overline{t}_f = t'_f$ and $\overline{t} \in \{(\overline{t}_f) \cdot_c \{\overline{t}_1, \overline{t}_2, \cdots, \overline{t}_k\}\}$. According to the construction of Thompson automaton of concatenation and using Proposition 3.6, we have

$$\Delta^*(\overline{t_i}) = q^G \tag{1}$$

According to the same construction and also using Proposition 3.6, the same transitions are replaced except for $a = c$ where c is the concatenation symbol, which is replaced by $\varepsilon(q^G) \to q_c^F$. Then, we have

$$\Delta^*(\overline{t_f}) = q^F. \tag{2}$$

From 1 and 2 we get $\Delta^*(\overline{t}) = q^E$. We show that $\varepsilon-\mathrm{closure}(\overline{t}) = t$.

$\varepsilon-\mathrm{closure}(\overline{t}) = \varepsilon-\mathrm{closure}(\{(\overline{t}_f \cdot)_c \{\overline{t}_1, \overline{t}_2, \cdots, \overline{t}_k\}\})$.

Using Property 3.3, we get

$$\varepsilon-\mathrm{closure}(\overline{t}) = \varepsilon-\mathrm{closure}(\overline{t}_f) \cdot_c \{\varepsilon-\mathrm{closure}(\overline{t}_1), \varepsilon-\mathrm{closure}(\overline{t}_2), \cdots, \varepsilon-\mathrm{closure}(\overline{t}_k)\}.$$

Using Remark 3.7, we have

$$\varepsilon\text{−closure}(\bar{t}) = \varepsilon\text{−closure}(t_f) \cdot_c \{\varepsilon\text{−closure}(t_1), \varepsilon\text{−closure}(t_2), \cdots, \varepsilon\text{−closure}(t_k)\}.$$

And finally using the induction hypothesis, we get

$$\varepsilon\text{−closure}(\bar{t}) = t_f \cdot_c \{t_1, t_2, \cdots, t_k\} = t.$$

□

Lemma 3.11. If $t \in \mathcal{L}(\mathrm{Th}_E)$, then $\varepsilon\text{−closure}(t) \in [\![E]\!]$.

Proof.
Case of leaf tree automaton Let $t \in \mathcal{L}(\mathrm{Th}_E)$. According to the construction of the generalized Thompson automaton of leaf tree we have $t = a$. Moreover, we have $\varepsilon\text{−closure}(t) = \varepsilon\text{−closure}(a) = a$ and $a \in [\![E]\!]$.
Case of arity automaton Let $t \in \mathcal{L}(\mathrm{Th}_E)$, $t = g(t_1, t_2, \ldots, t_n)$. According to the induction hypothesis and using Proposition 3.8, there exist $t_1, t_2, \ldots, t_n \in T_\Sigma$ such that $\widetilde{t}_1 \in \mathcal{L}(\mathrm{Th}_{E_1})$ and $\varepsilon\text{−closure}(\widetilde{t}_k) \in [\![E_k]\!]$, for $k = 1 \ldots n$.

Furthermore, we have

$$\varepsilon\text{−closure}(t) = \varepsilon\text{−closure}(g(t_1, t_2, \ldots, t_n)).$$

From Definition 3.1, we have

$$\varepsilon\text{−closure}(t) = g(\varepsilon\text{−closure}(t_1), \varepsilon\text{−closure}(t_2), \ldots, \varepsilon\text{−closure}(t_n)).$$

And using Remark 3.9, we get

$$\varepsilon\text{−closure}(t) = g(\varepsilon\text{−closure}(\widetilde{t}_1), \varepsilon\text{−closure}(\widetilde{t}_2), \ldots, \varepsilon\text{−closure}(\widetilde{t}_n)).$$

Using the induction hypothesis, we have $\varepsilon\text{−closure}(\widetilde{t}_i) \in [\![E_i]\!]$, for $i = 1 \ldots n$ where n is the rank of g. Then,

$$g(\varepsilon\text{−closure}(\widetilde{t}_1), \varepsilon\text{−closure}(\widetilde{t}_2), \ldots, \varepsilon\text{−closure}(\widetilde{t}_n)) \in [\![E]\!].$$

Case of concatenation automaton Let $t \in \mathcal{L}(\mathrm{Th}_E)$, then $t \in \{(t_f) \cdot_c \{t_1, \ldots, t_k\}\}$, such that $t_i \in [\![G]\!]$, $i = 1 \ldots k$. According to the induction hypothesis and using Proposition 3.8, there exist $t_1, \ldots, t_k, t_f \in T_\Sigma$ such that $\widetilde{t}_i \in \mathcal{L}(\mathrm{Th}_G)$ with $\varepsilon\text{−closure}(\widetilde{t}_i) \in [\![G]\!]$, $i = 1 \ldots k$, and $\widetilde{t}_f \in \mathcal{L}(\mathrm{Th}_F)$ with $\varepsilon\text{−closure}(\widetilde{t}_f) \in [\![F]\!]$.

Moreover, we have

$$\varepsilon\text{−closure}(t) = \varepsilon\text{−closure}(t_{f \cdot_c} \{t_1, \ldots, t_k\}).$$

Using Property 3.3, we get

$$\varepsilon\text{−closure}(t) = \varepsilon\text{−closure}(t_f) \cdot_c \{\varepsilon\text{−closure}(t_1), \ldots, \varepsilon\text{−closure}(t_k)\}.$$

Using Remark 3.9, we have

$$\varepsilon\text{−closure}(t) = \varepsilon\text{−closure}(\widetilde{t_f}) \cdot_c \{\varepsilon\text{−closure}(\widetilde{t_1}), \ldots, \varepsilon\text{−closure}(\widetilde{t_k})\}.$$

And finally using the induction hypothesis, we get

$$\varepsilon\text{−closure}(t) \in [\![F]\!] \cdot_c [\![G]\!] \in [\![F \cdot_c G]\!] \in [\![E]\!].$$

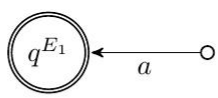
□

Corollary 3.12. The number of transitions in the generalized Thompson tree automaton for a regular tree expression E is linear in $|E|$.

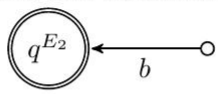
P r o o f . As each symbol in the regular tree expression E generates a constant number of transitions in the inductive construction of Thompson tree automaton, and since we can consider each regular tree expression as a tree where leaves represent the symbols of the alphabet and internal nodes represent the regular tree expression operators, so the number of transitions generated for this automaton's construction is $\mathcal{O}(|E|)$. □

Example of constructing a Thompson tree automaton

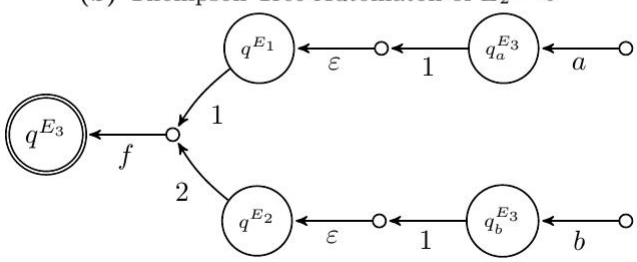
Let E be a regular tree expression such that $E = (f(a, b) + g(c) \cdot_c d)^{*,d}$. Figures 8 – 11 show the successive construction of Thompson tree Automaton for E.



(a) Thompson Tree Automaton of $E_1 = a$

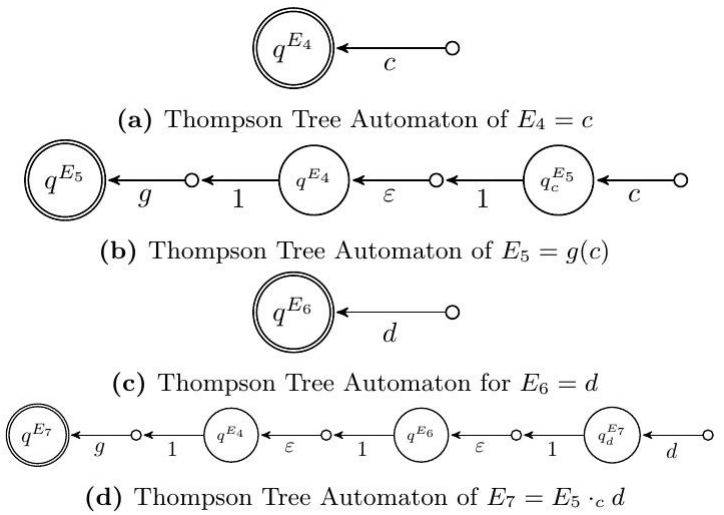


(b) Thompson Tree Automaton of $E_2 = b$



(c) Thompson Tree Automaton of $E_3 = f(a, b)$

Fig. 8: Thompson Tree Automaton of $f(a, b)$.



(a) Thompson Tree Automaton of $E_4 = c$

(b) Thompson Tree Automaton of $E_5 = g(c)$

(c) Thompson Tree Automaton for $E_6 = d$

(d) Thompson Tree Automaton of $E_7 = E_5 \cdot_c d$

Fig. 9: Thompson Tree Automaton of $g(c) \cdot_c d$.

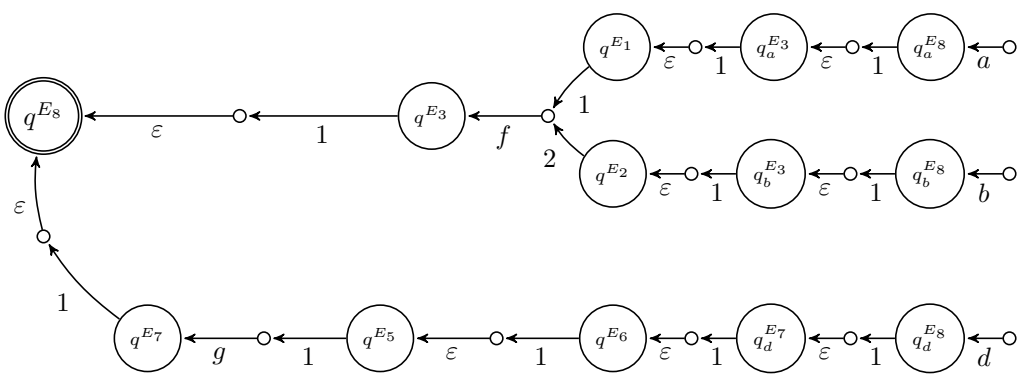


Fig. 10: Thompson Tree Automaton of $E_8 = E_3 + E_7 = f(a, b) + g(c) \cdot_c d$.

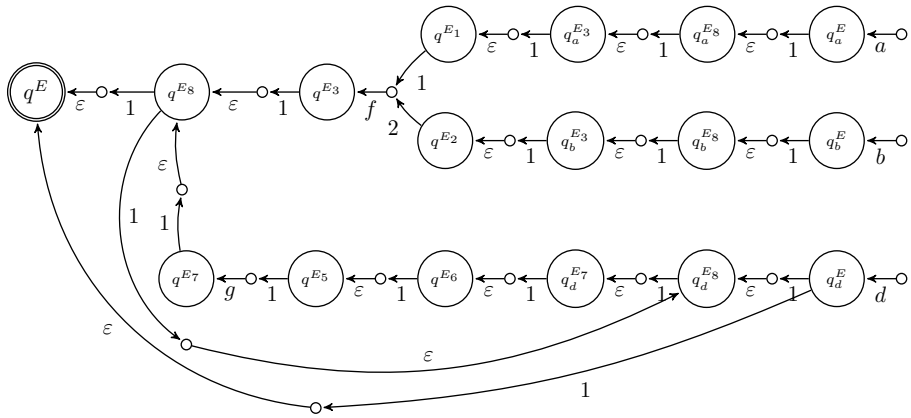


Fig. 11: Thompson Tree Automaton of $E = E_8^{*,d} = (f(a,b) + g(c) \cdot_c d)^{*,d}$.

4. APPLICATION TO TREE PATTERN MATCHING

By analogy to words, we will use the extended Thompson automaton in tree pattern matching. Let E be a regular tree expression and t a marked and linearized tree.

Searching occurrences of E in a subject tree t is done in two steps:

- Constructing Thompson tree automaton for E,
- Running the constructed automaton on t: each time the final state is reached, an occurrence of the pattern E has been recognized.

The construction proposed in Section 3 allows us to perform two kinds of pattern matching:

1. Top-down pattern matching by constructing a Thompson tree automaton for the regular tree expression $E \cdot_\upsilon (\Sigma \cup \{\upsilon\})^{*,\upsilon}$, where E is the regular expression of the tree pattern and $\upsilon \in \Sigma_0$ represents the variable leaf.

2. Bottom-up pattern matching by using the Thompson tree automaton of the regular tree expression E. We don't need to add a variable leaf as we perform the pattern matching from leaves and if a pattern exists it must be a sub-tree of the subject tree.

Note that both the top-down and the bottom-up versions of Thompson tree automaton are non-deterministic, but the top-down one is highly non-deterministic compared to the bottom-up one, thus the pattern matching takes more time. (Furthermore, in general the bottom-up automaton could be determinized while the top-down one could not be, given the limited power of deterministic top-down tree automata). For this reason we will develop hereafter only the bottom-up approach.

4.1. Tree pattern matching algorithm

We will not make the constructed automaton deterministic, in order to keep the reduced number of states and transitions. In fact, to verify that a tree t is recognized by a tree automaton Th_E, we will simulate a determinization by activating in each step of the traversal of the constructed automaton, states that are reachable from initial states.

Definition 4.1. For a state $q \in Q$, we define the set $Q_q^\varepsilon = \{p \mid \varepsilon(q) \longrightarrow p \in \Delta\}$.

This definition can be extended to set of states.

Definition 4.2. For a subset $P \subseteq Q$, we define the set $Q_P^\varepsilon = \bigcup_{q \in P} Q_q^\varepsilon$.

These two definitions are implemented in Algorithm 1 (*Skip_Epsilon*). Let $P \subseteq Q$ be a set of states. We define the set $\lambda(P) = \{q \in P \mid Q_q^\varepsilon = \emptyset\}$.

Algorithm 1: Function *Skip_Epsilon*

Input: P : Set of states.
Output: Set of reachable states from q after skipping ε-transitions.
1 $R \leftarrow P$;
2 $X \leftarrow \emptyset$;
3 **While** $(R \neq \emptyset)$ **Do**
4 $\quad X \leftarrow X \cup \lambda(R)$;
5 $\quad R \leftarrow Q_R^\varepsilon$;}
6 **EndWhile**
7 **Return**(X);

In function *Skip_Epsilon* (Algorithm 1), we calculate the set of reachable states from a set of states P. In every iteration of the loop *while* (line 3), we add the set $\lambda(R)$, which represents the subset of states of P that don't lead to any other state by an ε-transition, to the output set (line 4). The set R, initialized by P, contains at each step the reachable states after skipping one ε-transition (line 5). These instructions are repeated until there is no more ε-transitions to be reached.

According to the inductive construction of Thompson tree automata and using Definition 4.2, we can deduce the following property which guarantees that in a Thompson tree automaton, no cycles of ε-transitions exist.

Property 4.3. For a subset $P \subset Q$, $Q_P^\varepsilon \neq P$

Corollary 4.4. The function *Skip_Epsilon* has $\mathcal{O}(|E|)$ time complexity.

Proof. From Property 4.3 we deduce that the loop *while* has a finite number of iterations. Let M be this number of iterations and R_i be the subset of states calculated in the ith iteration. So, this loop is calculated in $\sum_{i=1}^{M} |\lambda(R_i)|$. Using Property 3.4, $\sum_{i=1}^{M} |\lambda(R_i)| \leqslant 2|Q|$. Therefore, the function *Skip_Epsilon* has $\mathcal{O}(|Q|)$ time complexity. Using Corollary 3.12, this complexity can be bounded by $\mathcal{O}(|E|)$. □

Definition 4.5. Let Q_f be the set of reachable states after reading f, that is $Q_f = \{p \in Q \mid \exists\, q_1, \cdots, q_n \in Q, f(q_1, \cdots, q_n) \longrightarrow p \in \Delta\}$.

Definition 4.6. Let Q_f^i be the set of states that are the ith child of f, that is $Q_f^i = \{q_i \in Q \mid f(q_1, \cdots, q_i, \cdots, q_n) \longrightarrow p \in \Delta\}$.

These two definitions are used in Algorithm 2 (function *Move*) in order to define the set of reachable states after reading a symbol $f \in \Sigma_n$.

In the function *Move* (Algorithm 2) we start by computing sets Q_f and Q_f^i for $i = 1 \ldots n$, where n is the arity of the symbol f. Then, we select from the set Q_f states for which all children already exist in the underlying set Q_f^i. Therefore, we get the output set of states reachable by reading the symbol f.

Algorithm 2: Function *Move*

Input: $P_1, P_2, \ldots, P_n$: Sets of states. $f \in \Sigma_n$.
Output: R : Set of states.

```
1  Compute Q_f and Q_f^i for i = 1, ..., n ;
2  R ← ∅ ;
3  Foreach (p ∈ Q_f) Do
4     //Let q_1, ..., q_i, ..., q_n ∈ Q such that f(q_1, ..., q_i, ..., q_n) ⟶ p ∈ Δ
5     If (q_i ∈ (P_i ∩ Q_f^i), for i = 1, ..., n)) Then
6        R ← R ∪ {p}
7     EndIf
8  EndForeach
9  Return(R);
```

Corollary 4.7. The function *Move* has $\mathcal{O}(|E|)$ time complexity.

P r o o f. Both the **foreach** loop (line 3) and the membership test (line 5) require $\mathcal{O}(r|Q|)$ time complexity, where r is the maximal arity for Σ, that is an $\mathcal{O}(|Q|)$ time complexity. As we want to express the complexity in terms of the input regular tree expression, we use Corollary 3.12 to bound $|Q|$ by $|E|$. So Algorithm 2 requires $\mathcal{O}(|E|)$ time complexity. □

Tree pattern matching algorithm using Thompson tree automaton is presented in Algorithm 3. This algorithm takes as input a *marked* subject tree t and the Thompson tree automaton Th_E of the pattern's regular tree expression E. We run the automaton using the functions *Skip_Epsilon* (Algorithm 1) and *Move*(Algorithm 2) in order to find occurrences of the pattern in the subject tree. Each time the final state q^E is reached, an occurrence of the pattern is found, and the symbol leading to the final state is added to the output set.

Algorithm 3: Algorithm TPM

Input: $t_{u_1 \ldots u_k}$: a node of the subject tree, Th_E : Thompson automaton for E.
Output: $P_{u_1 \ldots u_k}$: set of states.

If $(ar(t) = 0)$ **Then**
  $P_{u_1 \ldots u_k} \leftarrow Skip\_Epsilon(Move(h(t_{u_1 \ldots u_k})))$;
Else
  $P_{u_1 \ldots u_k} \leftarrow Skip\_Epsilon(Move(TPM(t_1), TPM(t_2), \ldots, TPM(t_n), h(t_{u_1 \ldots u_k})))$;
  // $n = ar(t)$
EndIf
If $(q^E \in P_{u_1 \ldots u_k})$ **Then**
  $occ \leftarrow occ \cup \{t_{u_1 \ldots u_k}\}$; $P_{u_1 \ldots u_k} \leftarrow P_{u_1 \ldots u_k} \setminus \{q^E\}$;
EndIf
Return$(P_{u_1 \ldots u_k})$;

In the TPM algorithm (Algorithm 3) we associate a set $P_{u_1 \ldots u_k}$ to each node $t_{u_1 \ldots u_k}$ of the subject tree t. As the algorithm goes on these sets maintain the reachable states during the traversal of the automaton by reading nodes of t. For each node $t_{u_1 \ldots u_k}$, we call the functions *Move* and *Skip_Epsilon* in order to calculate the new set $P_{u_1 \ldots u_k}$ taking as input parameters the symbol $t_{u_1 \ldots u_k}$ and the sets calculated recursively $P_{u_1 \ldots u_k}$. If the final state of the pattern's Thompson tree automaton is included in the set $P_{u_1 \ldots u_k}$, we add the symbol $t_{u_1 \ldots u_k}$ to the set of nodes matching the pattern *occ*.

Theorem 4.8. The tree pattern matching algorithm using Thompson tree automaton requires $\mathcal{O}(|t||E|)$ time complexity.

Proof. In Algorithm 3, the recursive call of TPM allows the process of all nodes in t, that is $|t|$. Using the functions *Move* and *Skip_Epsilon* for each node, we get an $\mathcal{O}(|t||E|)$ time complexity. □

4.2. Example of tree pattern matching using Thompson automaton

Let us consider the previous regular tree expression $E = (f(a, b) + g(c) \cdot_c d)^{*,d}$ a pattern's regular tree expression, and $t = a_{111}b_{112}d_{1211}f_{11}g_{121}h_{12}f_1$ a *marked* and *linearized* subject tree. Figure 12 recalls Thompson tree automaton Th_E constructed for E.

In order to determine nodes that match the pattern's regular tree expression, we run Algorithm 3 with t and Th_E as input parameters. Figure 13 shows the successive construction of sets $P_{u_1 \ldots u_k}$ by using the functions *Move* and *Skip_Epsilon*.

5. CONCLUSION

In this paper we have presented a new algorithm for tree pattern matching problem where we look for one or multiple occurrences of trees from some tree language, that is matched by the pattern represented by a regular tree expression in a target tree: the subject tree.

We have addressed this problem in two steps. First, we have proposed a generalization of Thompson automaton for strings to trees. The automaton used was constructed inductively on the structure of the pattern's regular tree expression. Then, we have presented a tree pattern matching algorithm that runs the extended Thompson automaton on the subject tree.

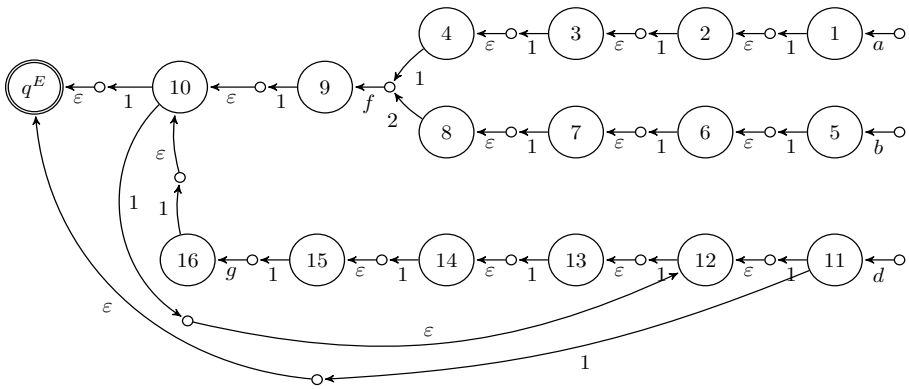


Fig. 12: Thompson Tree Automaton of the Pattern.

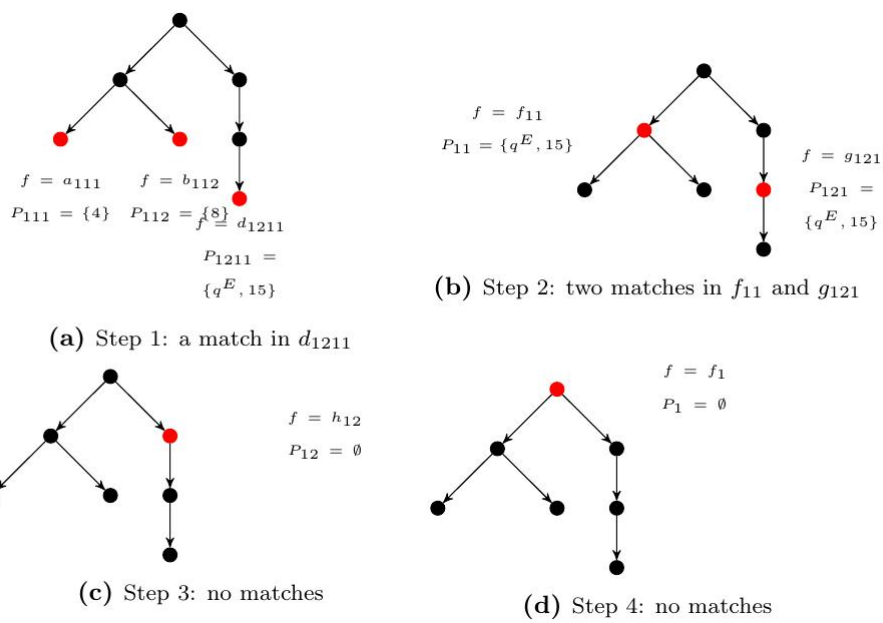


Fig. 13: Example of Running TPM Algorithm Using Thompson Tree Automaton.

The tree pattern matching can be done in $\mathcal{O}(|t||E|)$ time complexity, where t is the subject tree and E is the regular tree expression representing the pattern language. Despite the low theoretical complexity of our tree pattern matching algorithm, it might be necessary to get an idea about its practical one, which we estimate to be much lower since we have bounded all subsets of states by the set of all states Q of Thompson tree automaton. In order to do that, we aim to implement besides the tree pattern matching algorithm and the tree automaton acceptor, a parametrized regular tree expressions generator that generates random regular tree expressions as patterns and a random subject tree. The latter should contain at least one occurrence of the underlying pattern.

6. ACKNOWLEDGMENT

This work is supported by a South Africa-Algeria Cooperation Project funded by the South African National Research Foundation and the Algerian MESRS-DGRSDT under project A/AS-2013-002. Any opinion, finding and conclusion or recommendation expressed in this material is that of the author(s) and the NRF/MESRS-DGRSDT do not accept any liability in this regard.

The authors would like to thank Younes Guellouma for helpful comments and suggestions on an earlier draft of this paper.

(Received October 20, 2016)

REFERENCES

[1] T. A. Assaleh and W. Ai: Survey of Global Regular Expression Print (GREP) Tools. 2004. http://www.cosc.brocku.ca/~taa/papers/abou-assaleh_csci6306a.pdf

[2] A. V. Aho and M. Ganapathi: Efficient tree pattern matching (extended abstract): An aid to code generation. In: Proc. 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 1985, pp. 334–340.

[3] V. M. Antimirov: Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. *155* (1996), 291–319. DOI:10.1016/0304-3975(95)00182-4

[4] L. Barry: Derivatives of tree sets with applications to grammatical inference. IEEE Trans. Pattern Analysis and Machine Intelligence, IEEE Computer Soc. *3* (1981), 285-293. DOI:10.1109/tpami.1981.4767101

[5] L. Barry: The use of tree derivatives and a sample support parameter for inferring tree systems. IEEE Trans. Pattern Analysis and Machine Intelligence, IEEE Computer Soc. *4* (1982), 25–34. DOI:10.1109/tpami.1982.4767191

[6] J. A. Brzozowski: Derivatives of regular expressions. J. ACM *11* (1964), 481–494. DOI:10.1145/321239.321249

[7] J.-M. Champarnaud and D. Ziadi: From C-continuations to new quadratic algorithms for automaton synthesis. IJAC *11* (2001), 707–736. DOI:10.1142/s0218196701000772

[8] J.-M. Champarnaud and D. Ziadi: Canonical derivatives, partial derivatives and finite automaton constructions. Theor. Comput. Sci. *289* (2002), 137–163. DOI:10.1016/s0304-3975(01)00267-5

[9] L. Cleophas: Tree Algorithms: Two Taxonomies and a Toolkit. PhD Thesis, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, 2008.

[10] H. Comon, M. Dauchet, R. Gilleron, C. Loding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi: Tree Automata Techniques and Applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2008.

[11] J.-F. Dufayard, L. Duret, S. Penel, M. Gouy, F. Rechenmann, and G. Perrière: Tree pattern matching in phylogenetic trees: automatic search for orthologs or paralogs in homologous gene sequence databases. Bioinformatics, Oxford Univ Press, *21* (2005), 2596–2603. DOI:10.1093/bioinformatics/bti325

[12] T. Flouri, C. S. Iliopoulos, J. Janoušek, B. Melichar, and S. P. Pissis: Tree template matching in ranked ordered trees by pushdown automata. J. Discrete Algorithms *17* (2012), 15–23. DOI:10.1016/j.jda.2012.10.003

[13] Ch. W. Fraser, R. R. Henry, and T. A. Proebsting: BURG: Fast optimal instruction selection and tree parsing. SIGPLAN Not, ACM *27* (1992), 68–76. DOI:10.1145/131080.131089

[14] T. Genet and F. Klay: Rewriting for cryptographic protocol verification. In: Proc. 17th International Conference on Automated Deduction, CADE-17, Springer-Verlag, London 2000, pp. 271–290. DOI:10.1007/10721959_21

[15] R. Giegerich: A Declarative Approach to the Development of Dynamic Programming Algorithms, Applied to RNA Folding. Report, 1998.

[16] V. M. Glushkov: The abstract theory of automata. Russian Math. Surveys *16* (1961) 1–53. DOI:10.1070/rm1961v016n05abeh004112

[17] E. Goebelbecker: Using grep: Moving from DOS? Discover the power of this Linux utility. Linux Journal, Belltown Media *18* (1995).

[18] J. Goubault-Larrecq: A Method for Automatic Cryptographic Protocol Verification. In: Parallel and Distributed Processing, Springer 2000, pp. 977–984. DOI:10.1007/3-540-45591-4_134

[19] A. Gräf: Left-to-right Tree Pattern Matching. In: Rewriting Techniques and Applications, Springer 1991, pp. 323–334. DOI:10.1007/3-540-53904-2_107

[20] Ch. M. Hoffmann and M. J. O'Donnell: Pattern matching in trees. J. ACM *29* (1982), 68–95. DOI:10.1145/322290.322295

[21] Y. Itokawa, M. Wada, T. Ishii, and T. Uchida: Pattern Matching Algorithm Using a Succinct Data Structure for Tree-Structured Patterns. In: Intelligent Control and Innovative Computing, Lecture Notes in Electrical Engineering, Springer US 2012, pp. 349–361. DOI:10.1007/978-1-4614-1695-1_27

[22] H. H. Kron: Tree Templates and Subtree Transformational Grammars. PhD Thesis, University of California, Santa Cruz 1975.

[23] D. Kuske and I. Meinecke: Construction of tree automata from regular expressions. RAIRO – Theor. Inf. and Appl. *45* (2011), 347–370. DOI:10.1051/ita/2011107

[24] É. Laugerotte, N. O. Sebti, and D. Ziadi: From regular tree expression to position tree automaton. In: Language and Automata Theory and Applications – 7th International Conference, LATA, Bilbao 2013, pp. 395–406. DOI:10.1007/978-3-642-37064-9_35

[25] H.-T. Lu and Y. Wuu: A simple tree pattern-matching algorithm. In: Proc. Workshop on Algorithms and Theory of Computation, Citeseer 2000.

[26] M. Madhavan and P. Shankar: Optimal regular tree pattern matching using pushdown automata. In: Foundations of Software Technology and Theoretical Computer Science, 18th Conference, Chennai 1998, pp. 122–133. DOI:10.1007/978-3-540-49382-2_11

[27] R. McNaughton and H. Yamada: Regular expressions and finite state graphs for automata. Electronic Computers, IRE Trans. *EC-9* (1960), 39–47. DOI:10.1109/tec.1960.5221603

[28] R. Polách, J. Janoušek, and B. Melichar: Regular tree expressions and deterministic pushdown automata. In: Mathematical and Engineering Methods in Computer Science – 7th International Doctoral Workshop, MEMICS, Lednice 2011, pp. 70–77.

[29] E. M. Reingold, K. J. Urban, and D. Gries: K-M-P string matching revisited. Inf. Process. Lett. *64* (1997), 217–223. DOI:10.1016/s0020-0190(97)00173-7

[30] K. Thompson: Regular expression search algorithm. Commun. ACM *11* (1968), 419–422. DOI:10.1145/363347.363387

[31] J. Trávníček, J. Janoušek, B. Melichar, and L. G. Cleophas: Backward linearised tree pattern matching. In: Language and Automata Theory and Applications – 9th International Conference, LATA, Nice 2015, pp. 599–610. DOI:10.1007/978-3-319-15579-1_47

Ahlem Belabbaci, Laboratoire d'informatique et de mathématiques – Université Amar Telidji, Laghouat. Algérie.
e-mail: ah.belabbaci@lagh-univ.dz

Hadda Cherroun, Laboratoire d'informatique et de mathématiques – Université Amar Telidji, Laghouat. Algérie.
e-mail: Hadda_Cherroun@lagh-univ.dz

Loek Cleophas, FASTAR Research Group, Stellenbosch University, South Africa and Foundations of Language Processing Group, Umeå University. Sweden.
e-mail: loek@fastar.org

Djelloul Ziadi, LITIS – Université de Rouen. France.
e-mail: djelloul.Ziadi@univ-rouen.fr