Tree pattern matching from regular tree expressions

Citation for published version (APA):

DOI:
10.14736/kyb-2018-2-0221

Document status and date:
Published: 01/01/2018

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

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 $O(|t| |E|)$ 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
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
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 Champannaud and Ziadi [27, 28] in the word case. Tree derivatives were introduced by Levine in [41, 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 $(\Sigma, ar)$ be a ranked alphabet, where $\Sigma$ is a finite set of symbols and $ar$ represents the rank of $\Sigma$ which is a mapping from $\Sigma$ into $\mathbb{N}$. The set of symbols of rank $n$ is denoted by $\Sigma_n$. The elements of rank 0 are called constants. A tree $t$ over $\Sigma$ 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 $\Sigma_k$ and $t_1, \ldots, t_k$ are any $k$ trees over $\Sigma$. We denote by $|t|$ the number of nodes of a tree $t$ and by $T_\Sigma$ the set of trees over $\Sigma$. A tree language is a subset of $T_\Sigma$. 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: $\text{Mark}(f) = f_1$ if $f$ is the root symbol, $\text{Mark}(f) = f\text{Mark( Parent(f))pos(f)}$ otherwise, where $\text{pos}(f)$ 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: $f_1(f_1(a_{111}, b_{112}), h_{12}(g_{121}(d_{1211})))$. Let $\Sigma_M$ be the set of marked symbols of $\Sigma$. We define the mapping $h$ from $\Sigma_M$ to $\Sigma$, which for a marked symbol $f_{u_1 \ldots u_k}$ gives its corresponding symbol in $\Sigma$, that is $f$.

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

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: $L_1 \cdot_c L_2 = \bigcup_{l_1 \in L_1} \{ t[c \leftarrow L_2] \}$. 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.
\(L^* = \bigcup_{n \geq 0} L^n\). The constant \(c\) is called the symbol of the \(c\)-closure operator.

A regular tree expression \(E\) over a ranked alphabet \(\Sigma\) is inductively defined by \(E = 0, E \in \Sigma_0, E = f(E_1, \cdots, E_n), E = E_1 + E_2, E = E_1 \cdot c E_2, E = E_1^*,\) where \(c \in \Sigma_0, n \in \mathbb{N}, f \in \Sigma_n\) and \(E_1, E_2, \ldots, E_n\) are any \(n\) regular expressions over \(\Sigma\). We call \(E = f(E_1, \cdots, E_n)\) 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 \(\Sigma \cup \{+ \cdot c, \cdots, 1^* | c \in \Sigma_0\}\) where \(+\) and \(\cdot c\) can be seen as symbols of rank 2 and \(1^*\) has rank 1. The language \(E\) denoted by \(E\) is inductively defined by \(\emptyset = \{\}\), \([c] = \{c\}, \[f(E_1, E_2, \cdots, E_n)] = f([E_1], \ldots, [E_n]), [E_1 + E_2] = [E_1] \cup [E_2], [E_1 \cdot c E_2] = [E_1] \cdot c [E_2], [E_1^*] = [E_1]^*\) where \(n \in \mathbb{N}, E_1, E_2, \ldots, E_n\) 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 \(\Sigma_0\) or \(\varepsilon\). For tree automata, some changes are made in the representation of transitions. A transition from state \(q_0\) to states \(q_1, q_2, \ldots, q_n\) with a symbol or an \(\varepsilon\)-transition is represented by i) an edge connecting the state \(q_0\) to a small circle unlabeled with a symbol or \(\varepsilon\), 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 \(\varepsilon\) of rank 1 to the alphabet \(\Sigma\). This symbol has the same meaning as \(\varepsilon\) 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 \(\varepsilon\)-nodes from \(t\) as follows:

\[
\varepsilon\text{-closure}(a) = a \quad \text{for each } a \in \Sigma_0
\]

\[
\varepsilon\text{-closure}(\varepsilon(t)) = \varepsilon\text{-closure}(t)
\]

\[
\varepsilon\text{-closure}(g(t_1, \ldots, t_n)) = g(\varepsilon\text{-closure}(t_1), \ldots, \varepsilon\text{-closure}(t_n)).
\]

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 \( s, t_1, \ldots, t_n \in T_S \). We have then

\[
\varepsilon\text{-closure}(s \cdot c \{t_1, \ldots, t_n\}) = \varepsilon\text{-closure}(s) \cdot c \{\varepsilon\text{-closure}(t_1), \ldots, \varepsilon\text{-closure}(t_n)\}.
\]

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 \( \Sigma_0 \) (the frame \( Q_{\Sigma_0} \)). 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 \( \varepsilon \)-transitions are added during the inductive constructions.

![Fig. 1: General Form of Thompson Tree Automaton.](image)

Let \( E \) be a regular tree expression. The generalized Thompson automaton \( \text{Th}_E = (Q^E, \Sigma \cup \{\varepsilon\}, \{q^E\}, \Delta^E) \) over the alphabet \( \Sigma \cup \{\varepsilon\} \) associated with \( E \) is defined inductively as follows. Let \( \text{Th}_F, \text{Th}_G \) and \( \text{Th}_{E_i} \), 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.](image)
The leaf tree $E = a, a \in \Sigma_0$ (Figure 3): $Q^E = \{q^E\}$ and $\Delta_E = \{a \to q^E\}$

![Diagram](image)

**Fig. 3:** The Leaf Tree Thompson Automaton.

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

The arity function $E = f(E_1, \ldots, E_n)$ (Figure 4):

$Q^E = \bigcup_{i=1}^{n} Q^{E_i} \cup \{q^E\} \cup \{q^E_a \mid a \in \Sigma_0\}$ and

$$\Delta^E = \bigcup_{i=1}^{n} (\Delta^{>0,E_i}) \cup \{a \to q^E_a \mid a \in \Sigma_0\}$$

$$\cup \{f(q^{E_1}, q^{E_2}, \ldots, q^{E_n}) \to q^E\} \cup \{\epsilon(q^E_a) \to q^E_a \mid a \in \Sigma_0, i = 1 \ldots n\}.$$

The union $E = F + G$ (Figure 5):

$Q^E = Q^F \cup Q^G \cup \{q^E\} \cup \{q^F_a \mid a \in \Sigma_0\}$ and

$$\Delta^E = \Delta^{>0,F} \cup \Delta^{>0,G} \cup \{a \to q^E_a \mid a \in \Sigma_0\}$$

$$\cup \{\epsilon(q^E_a) \to q^F_a \mid a \in \Sigma_0\} \cup \{\epsilon(q^E_a) \to q^G_a \mid a \in \Sigma_0\}$$

$$\cup \{\epsilon(q^F) \to q^E\} \cup \{\epsilon(q^G) \to q^E\}.$$

The concatenation $E = F \cdot_c G$ (Figure 6):

$Q^E = Q^F \cup Q^G \cup \{q^E\} \cup \{q^E_a \mid a \in \Sigma_0\}$ and

$$\Delta^E = \Delta^{>0,F} \cup \Delta^{>0,G} \cup \{a \to q^E_a \mid a \in \Sigma_0\}$$

$$\cup \{\epsilon(q^E_a) \to q^G_a \mid a \in \Sigma_0\} \cup \{\epsilon(q^E_a) \to q^F_a \mid a \in \Sigma_0 \setminus \{c\}\}$$

$$\cup \{\epsilon(q^G) \to q^F_a \} \cup \{\epsilon(q^E) \to q^E\}.$$

The closure $E = F^*$ (Figure 7):

$Q^E = Q^F \cup \{q^E\} \cup \{q^E_a \mid a \in \Sigma_0\}$ and

$$\Delta^E = \Delta^{>0,F} \cup \{a \to q^E_a \mid a \in \Sigma_0\}$$

$$\cup \{\epsilon(q^E_a) \to q^E\} \cup \{\epsilon(q^F) \to q^F_a \} \cup \{\epsilon(q^F) \to q^E\}$$

$$\cup \{\epsilon(q^E_a) \to q^F_a \mid a \in \Sigma_0\}.$$

Let $q \in Q$ be a state. Let $Q^\varepsilon_-(q) = \{p \in Q \mid \varepsilon(p) \to q\}$.

From the construction of Thompson tree automata, we deduce the following property.

**Property 3.4.** For a state $q \in Q$, we have $|Q^\varepsilon_-(q)| \leq 2$. 
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 $E = F \cdot cG$. 
Theorem 3.5. For a regular tree expression $E$, $\varepsilon$-closure($L(\text{Th}_E)$) = $[E]$.

Before proving this theorem, we prove the two next propositions.
Let $\Sigma_0 = \{x_1, x_2, \ldots, x_n\}$ and $t, \tilde{t}, \tilde{\tilde{t}} \in T_{\Sigma}$ such that
\[
\tilde{t} = \left( \cdots \left( (t \cdot x_1 \varepsilon(x_1)) \cdot x_2 \varepsilon(x_2) \right) \cdots \right) \cdot x_n \varepsilon(x_n)
\]
and
\[
\tilde{\tilde{t}} = \left\{ \cdots \left\{ t \{ \varepsilon(x_1) \leftrightarrow x_1 \} \varepsilon(x_2) \leftrightarrow x_2 \right\} \cdots \right\} \varepsilon(x_n) \leftrightarrow x_n.
\]
Let $A, \overline{A}$ and $\widetilde{A}$ be tree automata such that $A = (Q, \Sigma, q^A, \Delta)$, $\overline{A} = (\overline{Q}, \Sigma, q^\overline{A}, \overline{\Delta})$ and $\widetilde{A} = (\tilde{Q}, \Sigma, q^{\tilde{A}}, \tilde{\Delta})$, where
\[
\overline{Q} = Q \cup \{ q^{\prime A}_a \mid \forall a \in \Sigma_0 \wedge q^A_a \in Q \}
\]
\[
q^{\overline{A}} = q^A
\]
\[
\overline{\Delta} = \Delta \setminus \{ a \rightarrow q^A_a \mid a \in \Sigma_0 \wedge q^A_a \in Q \}
\]
\[
\cup \{ a \rightarrow q^{\prime A}_a \mid a \in \Sigma_0 \wedge q^{\prime A}_a \in \overline{Q} \}
\]
\[
\cup \{ \varepsilon(q^{\prime A}_a) \rightarrow q^A_a \}
\]
and
\[
\tilde{Q} = Q \setminus \{ q^{\prime A}_a \mid \forall a \in \Sigma_0 \wedge q^{\prime A}_a \in Q \}
\]
\[
q^{\tilde{A}} = q^A
\]
\[ \tilde{\Delta} = \Delta \setminus \{ a \to q^\mathcal{A}_a \mid a \in \Sigma_0 \land q^\mathcal{A}_a \in Q \} \setminus \{ \varepsilon(q^\mathcal{A}_a) \to q^\mathcal{A}_a \mid q^\mathcal{A}_a \in Q \} \cup \{ a \to q^\mathcal{A}_a \mid a \in \Sigma_0 \land q^\mathcal{A}_a \in Q \}. \]

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

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

**Proof.** We have \( t \in \mathcal{L}(\mathcal{A}) \), so \( \Delta^*_\mathcal{A}(t) = q^\mathcal{A} \). According to the construction of \( \tilde{\mathcal{A}} \), every transition of the form \( a \to q^\mathcal{A}_a \) such that \( a \in \Sigma_0 \land q^\mathcal{A}_a \in Q \) in the path recognizing \( t \) in \( \mathcal{A} \) is replaced by the two transitions \( a \to q^\mathcal{A}_a \) and \( \varepsilon(q^\mathcal{A}_a) \to q^\mathcal{A}_a \) with \( a \in \Sigma_0 \land q^\mathcal{A}_a \in Q \). Then \( \Delta^*_\mathcal{A}(\tilde{t}) = q^\mathcal{A} = q^\tilde{\mathcal{A}} \), that is \( \tilde{t} \in \mathcal{L}(\tilde{\mathcal{A}}) \).

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

\[
\varepsilon \text{-closure}(\tilde{t}) = \varepsilon \text{-closure} \left( \left( (t \cdot x_1 \varepsilon(x_1)) \cdot x_2 \varepsilon(x_2) \right) \cdots \cdot x_n \varepsilon(x_n) \right)
= \left( \cdots \left( \varepsilon \text{-closure}(t) \cdot x_1 \text{-closure}(\varepsilon(x_1)) \right) \cdot x_2 \varepsilon \text{-closure}(\varepsilon(x_2)) \right)
\cdots \cdot x_n \varepsilon \text{-closure}(\varepsilon(x_n))
= \left( \cdots \left( \varepsilon \text{-closure}(t) \cdot x_1 (x_1) \right) \cdot x_2 (x_2) \right) \cdots \cdot x_n (x_n)
= \varepsilon \text{-closure}(t).
\]

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

**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 \( \tilde{\mathcal{A}} \), in the path recognizing \( t \) in \( \mathcal{A} \) we substitute all transitions of the form \( a \to q^\mathcal{A}_a \) and \( \varepsilon(q^\mathcal{A}_a) \to q^\mathcal{A}_a \) with \( a \in \Sigma_0 \land q^\mathcal{A}_a \in Q \) by the transition \( a \to q^\mathcal{A}_a \) for each \( a \in \Sigma_0 \land q^\mathcal{A}_a \in Q \). Then \( \tilde{\Delta}^*_\mathcal{A}(\tilde{t}) = q^\mathcal{A} = q^\tilde{\mathcal{A}} \), that is \( \tilde{t} \in \mathcal{L}(\tilde{\mathcal{A}}) \).

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

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 \( \varepsilon \)-closure\( (t') = t \).
Let $\Delta t$ where $t \in [E]$, $t = a$. From Thompson leaf tree automaton construction, we have $\mathcal{L}(\text{Th}_E) = \{a\}$, so $t' = a$. We have also $\varepsilon$-closure($t'$) = $\varepsilon$-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 $1 \leq i \leq n$. According to the induction hypothesis, there exist $t_i' \in \mathcal{L}(\text{Th}_{E_i})$ such that $\varepsilon$-closure($t_i'$) = $t_i$ with $i = 1 \ldots n$. We assume that $\bar{t}_i = t_i'$ since according to the construction of Thompson automaton generalization, $t_i'$ has the same structure as $\bar{t}_i$. 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 $\Delta^*(\bar{t}_i) = q^E_i$ for $i = 1 \ldots n$. Moreover, we have $g(q^E_1, q^E_2, \ldots, q^E_n) \rightarrow q^E \in \Delta$, then $\Delta^*(\bar{t}) = q^E$ where $\bar{t} = g(\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_n)$, that is $\bar{t} \in \mathcal{L}(\text{Th}_E)$. We show that $\varepsilon$-closure($\bar{t}$) = $t$. From Remark 3.7 we have $\varepsilon$-closure($\bar{t}_i$) = $\varepsilon$-closure($t_i$), so $\varepsilon$-closure($\bar{t}$) = $\varepsilon$-closure($g(\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_n)$).

Using Property 3.2 we get
$$\varepsilon$-closure($\bar{t}$) = g(\varepsilon$-closure($\bar{t}_1), \varepsilon$-closure($\bar{t}_2), \ldots, \varepsilon$-closure($\bar{t}_n)$).$$

Using Remark 3.7 we have
$$\varepsilon$-closure($\bar{t}$) = g(\varepsilon$-closure($t_1), \varepsilon$-closure($t_2), \ldots, \varepsilon$-closure($t_n)$).$$

Using the induction hypothesis, we get
$$\varepsilon$-closure($\bar{t}$) = g(t_1, t_2, \ldots, t_n) = t.$$

Case $E = F_c G$ 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 $t_f' \in \mathcal{L}(\text{Th}_F)$ and $t_i' \in \mathcal{L}(\text{Th}_G)$, $i = 1 \ldots k$, such that $\varepsilon$-closure($t_f'$) = $t_f$ and $\varepsilon$-closure($t_i'$) = $t_i$. Let $\bar{t}_i = t_i'$, $\bar{t}_f = t_f'$ and $\bar{t} \in \{(\bar{t}_f) \cdot_c \{\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_k\}\}$. According to the construction of Thompson automaton of concatenation and using Proposition 3.6 we have
$$\Delta^*(\bar{t}_i) = q^G$$
(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 $\varepsilon(q^G) \rightarrow q^F_c$. Then, we have
$$\Delta^*(\bar{t}_f) = q^F.$$
(2)

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

$$\varepsilon$-closure($\bar{t}$) = \varepsilon$-closure($\{(\bar{t}_f) \cdot_c \{\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_k\}\}$).

Using Property 3.3 we get
$$\varepsilon$-closure($\bar{t}$) = $\varepsilon$-closure($\bar{t}_f \cdot_c \{\varepsilon$-closure($\bar{t}_1), \varepsilon$-closure($\bar{t}_2), \ldots, \varepsilon$-closure($\bar{t}_k)\}$).
Using Remark 3.7, we have
\[ \varepsilon - \text{closure}(\tilde{t}) = \varepsilon - \text{closure}(t_f) \cdot_c \{ \varepsilon - \text{closure}(t_1), \varepsilon - \text{closure}(t_2), \ldots, \varepsilon - \text{closure}(t_k) \}. \]

And finally using the induction hypothesis, we get
\[ \varepsilon - \text{closure}(\tilde{t}) = t_f \cdot_c \{ t_1, t_2, \ldots, t_k \} = t. \]

\[ \square \]

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

Proof. Let \( t \in \mathcal{L}(\text{Th}_E) \). 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 \( t \in \mathcal{L}(\text{Th}_E) \), \( t = g(t_1, t_2, \ldots, t_n) \). According to the induction hypothesis and using Proposition 3.8, there exist \( t_1, t_2, \ldots, t_n \in T_\Sigma \) such that \( \tilde{t}_1 \in \mathcal{L}(\text{Th}_{E_1}) \) and \( \varepsilon - \text{closure}(\tilde{t}_k) \in [E_k] \), for \( k = 1 \ldots n \).

Using the induction hypothesis, 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
\[ \varepsilon - \text{closure}(t) = g(\varepsilon - \text{closure}(\tilde{t}_1), \varepsilon - \text{closure}(\tilde{t}_2), \ldots, \varepsilon - \text{closure}(\tilde{t}_n)). \]

Case of concatenation automaton. Let \( t \in \mathcal{L}(\text{Th}_E) \), then \( t \in \{(t_f)_{-c} \{ t_1, \ldots, t_k \} \} \), 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 \( \tilde{t}_i \in \mathcal{L}(\text{Th}_G) \) with \( \varepsilon - \text{closure}(\tilde{t}_i) \in [G], i = 1 \ldots k \), and \( \tilde{t}_f \in \mathcal{L}(\text{Th}_F) \) with \( \varepsilon - \text{closure}(\tilde{t}_f) \in [F] \).

Moreover, we have
\[ \varepsilon - \text{closure}(t) = \varepsilon - \text{closure}(t_f) \cdot_c \{ \varepsilon - \text{closure}(t_1), \ldots, \varepsilon - \text{closure}(t_k) \}. \]

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

\[ \epsilon - \text{closure}(t) = \epsilon - \text{closure}(\tilde{t}_f) \cdot \{\epsilon - \text{closure}(\tilde{t}_1), \ldots, \epsilon - \text{closure}(\tilde{t}_k)\} \].

And finally using the induction hypothesis, we get

\[ \epsilon - \text{closure}(t) \in [F] \cdot c [G] \in [F \cdot c \cdot G] \in [E]. \]

\[ \square \]

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

**Proof.** 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 \( O(|E|) \). \[ \square \]

**Example of constructing a Thompson tree automaton**

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

\[ \text{(a) Thompson Tree Automaton of } E_1 = a \]

\[ \text{(b) Thompson Tree Automaton of } E_2 = b \]

\[ \text{(c) Thompson Tree Automaton of } E_3 = f(a, b) \]

**Fig. 8:** Thompson Tree Automaton of \( f(a, b) \).
Fig. 9: Thompson Tree Automaton of $g(c) \cdot_c d$.

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

(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 $E_7 = E_5 \cdot_c d$
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 \( E \cdot \nu (\Sigma \cup \{\nu\})^* \cdot \nu \), where \( E \) is the regular expression of the tree pattern and \( \nu \in \Sigma_0 \) 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 \( Q^\epsilon_q = \{ p \mid \epsilon(q) \rightarrow p \in \Delta \} \).

This definition can be extended to set of states.

**Definition 4.2.** For a subset \( P \subseteq Q \), we define the set \( Q^\epsilon_P = \bigcup_{q \in P} Q^\epsilon_q \).

These two definitions are implemented in Algorithm 1 (\textit{Skip Epsilon}). Let \( P \subseteq Q \) be a set of states. We define the set \( \lambda(P) = \{ q \in P \mid Q^\epsilon_q = \emptyset \} \).

**Algorithm 1:** Function \textit{Skip Epsilon}

\begin{verbatim}
Input: \( P \) : Set of states.
Output: Set of reachable states from \( q \) after skipping \( \epsilon \)-transitions.
1 \( R \leftarrow P ; \)
2 \( X \leftarrow \emptyset ; \)
3 \textbf{While} \( (R \neq \emptyset) \) \textbf{Do}
4 \( X \leftarrow X \cup \lambda(R) ; \)
5 \( R \leftarrow Q^\epsilon_R ; \}
6 \textbf{EndWhile}
7 \textbf{Return}(X);
\end{verbatim}

In function \textit{Skip Epsilon} (Algorithm 1), we calculate the set of reachable states from a set of states \( P \). In every iteration of the loop \textit{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 \( \epsilon \)-transition, to the output set (line 4). The set \( R \), initialized by \( P \), contains at each step the reachable states after skipping one \( \epsilon \)-transition (line 5). These instructions are repeated until there is no more \( \epsilon \)-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 \( \epsilon \)-transitions exist.

**Property 4.3.** For a subset \( P \subset Q \), \( Q^\epsilon_P \neq P \)

**Corollary 4.4.** The function \textit{Skip Epsilon} has \( O(|E|) \) time complexity.

\textbf{Proof.} From Property 4.3, we deduce that the loop \textit{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 \( i \)th iteration. So, this loop is calculated in \( \sum_{i=1}^{M} |\lambda(R_i)| \). Using Property 3.4, \( \sum_{i=1}^{M} |\lambda(R_i)| \leq 2|Q| \). Therefore, the function \textit{Skip Epsilon} has \( O(|Q|) \) time complexity. Using Corollary 3.12, this complexity can be bounded by \( O(|E|) \). \( \square \)
Definition 4.5. Let $Q_f$ be the set of reachable states after reading $f$, that is $Q_f = \{ p \in Q \mid \exists q_1, \cdots, q_n \in Q, f(q_1, \cdots, q_n) \rightarrow p \in \Delta \}$.

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

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

In the function $\text{Move}$ (Algorithm 2) we start by computing sets $Q_f$ and $Q^i_f$ 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 $Q^i_f$. 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 $Q^i_f$ for $i = 1, \ldots, n$;
2. $R \leftarrow \emptyset$;
3. Foreach $(p \in Q_f)$ Do
4. //Let $q_1, \ldots, q_i, \ldots, q_n \in Q$ such that $f(q_1, \ldots, q_i, \ldots, q_n) \rightarrow p \in \Delta$
5. If $(q_i \in (P_i \cap Q^i_f), \text{for } i = 1, \ldots, n))$ Then
6. $R \leftarrow R \cup \{ p \}$
7. EndIf
8. EndForeach
9. Return(R);

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

**Proof.** Both the foreach loop (line 3) and the membership test (line 5) require $O(r|Q|)$ time complexity, where $r$ is the maximal arity for $\Sigma$, that is an $O(|Q|)$ 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 $O(|E|)$ time complexity. $\square$

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 $\text{Skip_Epsilon}$ (Algorithm 1) and $\text{Move}$ (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.
1 If \((ar(t) = 0)\) Then
2 \(P_{u_1...u_k} \leftarrow \text{Skip.Epsilon}(\text{Move}(h(t_{u_1...u_k})))\);
3 Else
4 \(P_{u_1...u_k} \leftarrow \text{Skip.Epsilon}(\text{Move}(\text{TPM}(t_1), \text{TPM}(t_2), \ldots, \text{TPM}(t_n), h(t_{u_1...u_k})))\);
\(/n = ar(t)\)
5 EndIf
6 If \((q^E \in P_{u_1...u_k})\) Then
7 \(\text{occ} \leftarrow \text{occ} \cup \{t_{u_1...u_k}\}; P_{u_1...u_k} \leftarrow P_{u_1...u_k} \setminus \{q^E\}\)
8 EndIf
9 Return\((P_{u_1...u_k});\)

In the TPM algorithm (Algorithm 3) we associate a set \(P_{u_1...u_k}\) to each node \(t_{u_1...u_k}\) 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 \(t_{u_1...u_k}\) we call the functions \text{Move} and \text{Skip.Epsilon} in order to calculate the new set \(P_{u_1...u_k}\) taking as input parameters the symbol \(t_{u_1...u_k}\) and the sets calculated recursively \(P_{u_1...u_k}\). If the final state of the pattern’s Thompson tree automaton is included in the set \(P_{u_1...u_k}\), we add the symbol \(t_{u_1...u_k}\) to the set of nodes matching the pattern \text{occ}.

**Theorem 4.8.** The tree pattern matching algorithm using Thompson tree automaton requires \(O(|t||E|)\) 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 \text{Move} and \text{Skip.Epsilon} for each node, we get an \(O(|t||E|)\) time complexity. \(\square\)

**4.2. Example of tree pattern matching using Thompson automaton**

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

In order to determine nodes that match the pattern’s regular tree expression, we run Algorithm 3 with \(t\) and \(\text{Th}_E\) as input parameters. Figure 13 shows the successive construction of sets \(P_{u_1...u_k}\) by using the functions \text{Move} and \text{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.

(a) Step 1: a match in \( d_{1211} \)

(b) Step 2: two matches in \( f_{11} \) and \( g_{121} \)

(c) Step 3: no matches

(d) Step 4: no matches

Fig. 13: Example of Running TPM Algorithm Using Thompson Tree Automaton.
The tree pattern matching can be done in $O(|t||E|)$ 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


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