

# Subtree Matching by Deterministic Pushdown Automata

Tomáš Flouri, Bořivoj Melichar

Department of Computer Science and Engineering  
 Faculty of Electrical Engineering  
 Czech Technical University in Prague  
 Karlovo nám. 13, 121 35 Prague 2, Czech Republic  
 Email: {flourt1,melichar}@fel.cvut.cz

J. Janoušek

Department of Computer Science  
 Faculty of Information Technologies  
 Czech Technical University in Prague  
 Kolejni 550/2, 160 00 Prague 6, Czech Republic  
 Email: Jan.Janousek@fit.cvut.cz

**Abstract**—Subtree matching is an important problem in Computer Science on which a number of tasks, such as mechanical theorem proving, term-rewriting, symbolic computation and non-procedural programming languages are based on. A systematic approach to the construction of subtree pattern matchers by deterministic pushdown automata, which read subject trees in prefix notation, is presented. The method is analogous to the construction of string pattern matchers: for a given pattern, a nondeterministic pushdown automaton is created and then it is determined. In addition, it is shown that the size of the resulting deterministic pushdown automata directly corresponds to the size of the existing string pattern matchers based on finite automata.

## I. INTRODUCTION

THE THEORY of formal string (or word) languages [1], [12], [19] and the theory of formal tree languages [4], [7], [9] are important parts of the theory of formal languages [20]. The most famous models of computation of the theory of tree languages are various kinds of tree automata [4], [7], [9]. Trees can also be seen as strings, for example in their prefix (also called preorder) or postfix (also called postorder) notation. [13] shows that the deterministic pushdown automaton (PDA) is an appropriate model of computation for labelled, ordered, ranked trees in postfix notation and that trees in postfix notation which are accepted by deterministic PDAs form a proper superclass of the class of regular tree languages, which are accepted by finite tree automata.

Tree pattern matching is often declared to be analogous to the problem of string pattern matching [4]. One of the basic approaches used for string pattern matching can be represented by finite automata constructed for the pattern, which means that the pattern is preprocessed. Examples of these automata can be the string matching automata [5], [6], [17], [21]. Given a pattern  $P$  of size  $m$ , the string matching automaton can be constructed for the pattern  $P$  in time linear to  $m$ . The constructed string matching automaton accepts the set of all words containing pattern  $P$  as a suffix and thus it can find all occurrences of string  $P$  in a given text  $T$ . The main advantage of this kind of finite automata is that the deterministic string

matching automaton can be constructed in time linear to the size of the given pattern  $P$  and the search phase is in time linear to the input text.

Although there are many tree pattern matching methods (see [11] for these methods along with their time and space complexities), they fail to represent a simple and systematic approach with a linear time searching phase which would also be directly analogous to the basic string pattern matching method.

This paper presents a new kind of PDAs for trees in prefix notation, which is directly analogous to string matching automata and their properties: *subtree matching PDAs* constructed over a tree  $t_1$  can find all occurrences of subtree  $t_1$  within a given tree  $t_2$  in time  $\mathcal{O}(n)$ , where  $n$  is the number of nodes of  $t_2$ . The basic idea of the subtree matching PDAs has been presented in [16]. This paper deals with the subtree matching PDAs in more detail.

By analogy with the string matching automaton, the subtree matching PDA is constructed over a given subtree  $t_1$  having  $m$  nodes.

Moreover, the presented subtree matching PDAs have the following two other properties. First, they are input-driven PDAs [23], which means that each pushdown operation is determined only by the input symbol. The input-driven PDAs can be always determined [23]. Second, their pushdown symbol alphabets contain just one pushdown symbol and therefore their pushdown store can be implemented by a single integer counter. This means that the presented PDAs can be transformed to counter automata [3], [22], which is a weaker and simpler model of computation than the PDA.

The rest of the paper is organised as follows. Basic definitions are given in section II. Some properties of subtrees in prefix notation are discussed in the third section. The fourth section deals with the subtree matching PDA. Fifth section acts as an overview of the time and space complexities and the last section is the conclusion.

## II. BASIC NOTIONS

### A. Ranked alphabet, tree, prefix notation, subtree matching

We define notions on trees similarly as they are defined in [1], [4], [7], [9].

This research has been partially supported by the Ministry of Education, Youth and Sports under research program MSMT 6840770014, and by the Czech Science Foundation as project No. 201/09/0807.

We denote the set of natural numbers by  $\mathbb{N}$ . A *ranked alphabet* is a finite, nonempty set of symbols, each of which has a unique nonnegative *arity* (or *rank*). Given a ranked alphabet  $\mathcal{A}$ , the arity of a symbol  $a \in \mathcal{A}$  is denoted  $\text{Arity}(a)$ . The set of symbols of arity  $p$  is denoted by  $\mathcal{A}_p$ . Elements of arity  $0, 1, 2, \dots, p$  are respectively called nullary (constants), unary, binary,  $\dots$ ,  $p$ -ary symbols. We assume that  $\mathcal{A}$  contains at least one constant. In the examples we use numbers at the end of identifiers for a short declaration of symbols with arity. For instance,  $a_2$  is a short declaration of a binary symbol  $a$ .

Based on concepts of the graph theory (see [1]), a labelled, ordered, ranked tree over a ranked alphabet  $\mathcal{A}$  can be defined as follows:

An *ordered directed graph*  $G$  is a pair  $(N, R)$ , where  $N$  is a set of nodes and  $R$  is a set of linearly ordered lists of edges such that each element of  $R$  is of the form  $((f, g_1), (f, g_2), \dots, (f, g_n))$ , where  $f, g_1, g_2, \dots, g_n \in N$ ,  $n \geq 0$ . This element would indicate that, for node  $f$ , there are  $n$  edges leaving  $f$ , the first entering node  $g_1$ , the second entering node  $g_2$ , and so forth.

A sequence of nodes  $(f_0, f_1, \dots, f_n)$ ,  $n \geq 1$ , is a *path* of length  $n$  from node  $f_0$  to node  $f_n$  if there is an edge which leaves node  $f_{i-1}$  and enters node  $f_i$  for  $1 \leq i \leq n$ . A *cycle* is a path  $(f_0, f_1, \dots, f_n)$ , where  $f_0 = f_n$ . An ordered *dag* (dag stands for Directed Acyclic Graph) is an ordered directed graph that has no cycle. A *labelling* of an ordered graph  $G = (A, R)$  is a mapping of  $A$  into a set of labels. In the examples we use  $a_f$  for a short declaration of node  $f$  labelled by symbol  $a$ .

Given a node  $f$ , its *out-degree* is the number of distinct pairs  $(f, g) \in R$ , where  $g \in A$ . By analogy, *in-degree* of the node  $f$  is the number of distinct pairs  $(g, f) \in R$ , where  $g \in A$ .

A *labelled, ordered, ranked and rooted tree*  $t$  over a ranked alphabet  $\mathcal{A}$  is an ordered dag  $t = (N, R)$  with a special node  $r \in A$  called the *root* such that

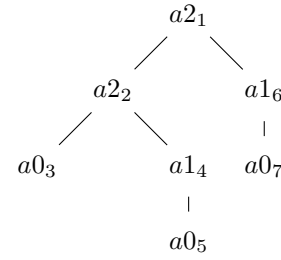
- (1)  $r$  has in-degree 0,
- (2) all other nodes of  $t$  have in-degree 1,
- (3) there is just one path from the root  $r$  to every  $f \in N$ , where  $f \neq r$ ,
- (4) every node  $f \in N$  is labelled by a symbol  $a \in \mathcal{A}$  and out-degree of  $a_f$  is  $\text{Arity}(a)$ .

Nodes labelled by nullary symbols (constants) are called *leaves*.

*Prefix notation*  $\text{pref}(t)$  of a labelled, ordered, ranked and rooted tree  $t$  is obtained by applying the following *Step* recursively, beginning at the root of  $t$ :

*Step:* Let this application of *Step* to be node  $a_f$ . If  $a_f$  is a leaf, list  $a$  and halt. If  $a_f$  is not a leaf, let its direct descendants be  $a_{f_1}, a_{f_2}, \dots, a_{f_n}$ . Then list  $a$  and subsequently apply *Step* to  $a_{f_1}, a_{f_2}, \dots, a_{f_n}$  in that order.

*Example 2.1:* Consider a ranked alphabet  $\mathcal{A} = \{a_2, a_1, a_0\}$ . Consider a tree  $t_1$  over  $\mathcal{A}$   
 $t_1 = (\{a_{21}, a_{22}, a_{03}, a_{14}, a_{05}, a_{16}, a_{07}\}, R)$ , where  $R$  is a set of the following ordered sequences of pairs:



$$\text{pref}(t_1) = a_2 a_2 a_0 a_1 a_0 a_1 a_0$$

Fig. 1. Tree  $t_1$  from Example 2.1 and its prefix notation.

$$\begin{aligned} &((a_{21}, a_{22}), (a_{21}, a_{16})), \\ &((a_{22}, a_{03}), (a_{22}, a_{14})), \\ &((a_{14}, a_{05})), \\ &((a_{16}, a_{07})) \end{aligned}$$

Tree  $t_1$  in prefix notation is string  $\text{pref}(t_1) = a_2 a_2 a_0 a_1 a_0 a_1 a_0$ . Trees can be represented graphically and tree  $t_1$  is illustrated in Fig. 1. ■

The height of a tree  $t$ , denoted by  $\text{Height}(t)$ , is defined as the maximal length of a path from the root of  $t$  to a leaf of  $t$ .

A subtree  $p$  *matches* an object tree  $t$  at node  $n$  if  $p$  is equal to the subtree of  $t$  rooted at  $n$ .

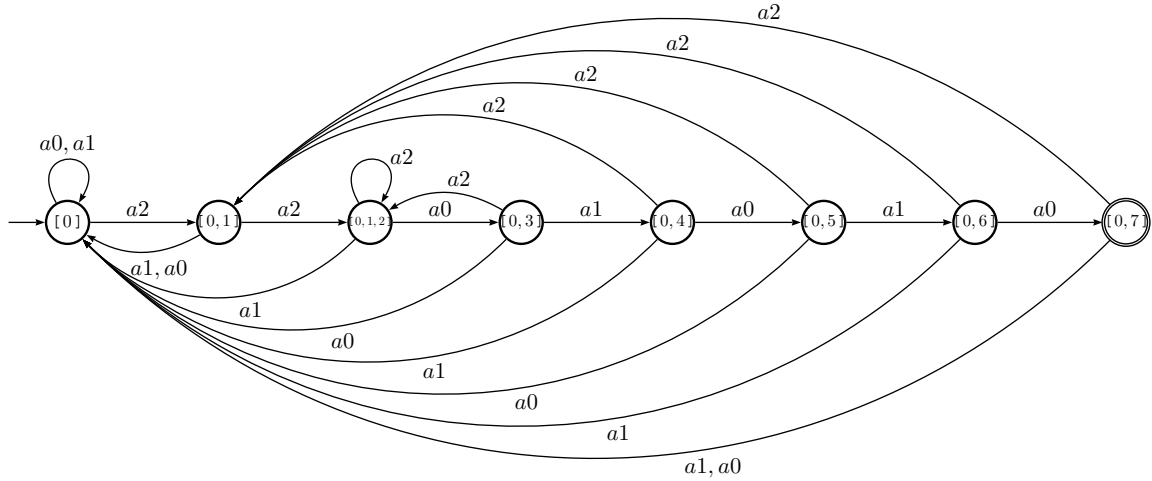
## B. Alphabet, language, pushdown automaton

We define notions from the theory of string languages similarly as they are defined in [1], [12].

Let an *alphabet* be a finite nonempty set of symbols. A *string*  $x$  over a given alphabet is a finite, possibly empty sequence of symbols. A *language* over an alphabet  $\mathcal{A}$  is a set of strings over  $\mathcal{A}$ . Set  $\mathcal{A}^*$  denotes the set of all strings over  $\mathcal{A}$  including the empty string, denoted by  $\varepsilon$ . Set  $\mathcal{A}^+$  is defined as  $\mathcal{A}^+ = \mathcal{A}^* \setminus \{\varepsilon\}$ . Similarly for string  $x \in \mathcal{A}^*$ ,  $x^m$ ,  $m \geq 0$ , denotes the  $m$ -fold concatenation of  $x$  with  $x^0 = \varepsilon$ . Set  $x^*$  is defined as  $x^* = \{x^m : m \geq 0\}$  and  $x^+ = x^* \setminus \{\varepsilon\} = \{x^m : m \geq 1\}$ .

An (extended) *nondeterministic pushdown automaton* (nondeterministic PDA) is a seven-tuple  $M = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$ , where  $Q$  is a finite set of *states*,  $\mathcal{A}$  is an *input alphabet*,  $G$  is a *pushdown store alphabet*,  $\delta$  is a mapping from  $Q \times (\mathcal{A} \cup \{\varepsilon\}) \times G^*$  into a set of finite subsets of  $Q \times G^*$ ,  $q_0 \in Q$  is an initial state,  $Z_0 \in G$  is the initial contents of the pushdown store, and  $F \subseteq Q$  is the set of final (accepting) states. Triplet  $(q, w, x) \in Q \times \mathcal{A}^* \times G^*$  denotes the configuration of a pushdown automaton. In this paper we will write the top of the pushdown store  $x$  on its left hand side. The initial configuration of a pushdown automaton is a triplet  $(q_0, w, Z_0)$  for the input string  $w \in \mathcal{A}^*$ .

The relation  $\vdash_M \subset (Q \times \mathcal{A}^* \times G^*) \times (Q \times \mathcal{A}^* \times G^*)$  is a *transition* of a pushdown automaton  $M$ . It holds that  $(q, aw, \alpha\beta) \vdash_M (p, w, \gamma\beta)$  if  $(p, \gamma) \in \delta(q, a, \alpha)$ . The  $k$ -th power, transitive closure, and transitive and reflexive closure


 Fig. 2. Transition diagram of deterministic string matching automaton for pattern  $x = a2 a2 a0 a1 a0 a1 a0$  from Example 2.2.

of the relation  $\vdash_M$  is denoted  $\vdash_M^k$ ,  $\vdash_M^+$ ,  $\vdash_M^*$ , respectively. A pushdown automaton  $M$  is *deterministic* pushdown automaton (deterministic PDA), if it holds:

- 1)  $|\delta(q, a, \gamma)| \leq 1$  for all  $q \in Q$ ,  $a \in \mathcal{A} \cup \{\varepsilon\}$ ,  $\gamma \in G^*$ .
- 2) If  $\delta(q, a, \alpha) \neq \emptyset$ ,  $\delta(q, a, \beta) \neq \emptyset$  and  $\alpha \neq \beta$  then  $\alpha$  is not a suffix of  $\beta$  and  $\beta$  is not a suffix of  $\alpha$ .
- 3) If  $\delta(q, a, \alpha) \neq \emptyset$ ,  $\delta(q, \varepsilon, \beta) \neq \emptyset$ , then  $\alpha$  is not a suffix of  $\beta$  and  $\beta$  is not a suffix of  $\alpha$ .

A pushdown automaton is *input-driven* if its each pushdown operation is determined only by the input symbol.

A language  $L$  accepted by a pushdown automaton  $M$  is defined in two distinct ways:

- 1) *Accepting by final state:*

$$L(M) = \{x : \delta(q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \gamma) \wedge x \in \mathcal{A}^* \wedge \gamma \in G^* \wedge q \in F\}.$$

- 2) *Accepting by empty pushdown store:*

$$L_\varepsilon(M) = \{x : (q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \varepsilon) \wedge x \in \mathcal{A}^* \wedge q \in Q\}.$$

If PDA accepts the language by empty pushdown store then the set  $F$  of final states may be the empty set. The subtree PDAs accept the languages by empty pushdown store.

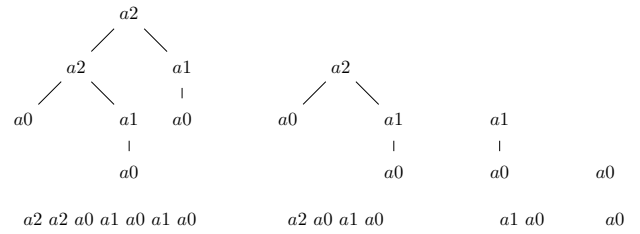
For more details on pushdown automata see [1], [12].

### C. Example of string matching automaton

*Example 2.2:* The transition diagram of the deterministic string matching automaton constructed for string  $a2 a2 a0 a1 a0 a1 a0$  is illustrated in Fig. 2. (See [1],[5] and [17] for definitions of finite automata and construction of the deterministic string matching automaton.)

## III. PROPERTIES OF SUBTREES IN PREFIX NOTATION

In this section we describe some general properties of the prefix notation of a tree and of its subtrees. These properties are important for the construction of the subtree matching PDA, which is described in the next section.


 Fig. 3. All subtrees of tree  $t_1$  from Example 2.1, and their prefix notations.

*Example 3.1:* Consider tree  $t_1$  in prefix notation  $\text{pref}(t_1) = a2 a2 a0 a1 a0 a1 a0$  from Example 2.1, which is illustrated in Fig. 1. Tree  $t_1$  contains only subtrees shown in Fig. 3.

Generally, for any tree, the following theorem holds.

*Theorem 1:* Given a tree  $t$  and its prefix notation  $\text{pref}(t)$ , all subtrees of  $t$  in prefix notation are substrings of  $\text{pref}(t)$ .

*Proof:* By induction on the height of the subtree.

- 1) If a subtree  $t'$  has just one node  $a$ , where  $\text{Arity}(a) = 0$ , then  $\text{Height}(t') = 0$ ,  $\text{pref}(t') = a$  and the claim holds for that subtree.
- 2) Assume that claim holds for subtrees  $t_1, t_2, \dots, t_p$ , where  $p \geq 1$ ,  $\text{Height}(t_1) \leq m$ ,  $\text{Height}(t_2) \leq m, \dots, \text{Height}(t_p) \leq m$ ,  $m \geq 0$ . We have to prove that the claim holds also for each subtree  $t' = at_1t_2 \dots t_p$ , where  $\text{Arity}(a) = p$ ,  $\text{Height}(t') = m + 1$ :  
As  $\text{pref}(t') = a \text{pref}(t_1) \text{pref}(t_2) \dots \text{pref}(t_p)$ , the claim holds for the subtree  $t'$ .

Thus, the theorem holds.  $\blacksquare$

However, not every substring of a tree in prefix notation is its subtree in prefix notation. This can be easily seen on the fact that for a given tree with  $n$  nodes in prefix notation, there can be  $\mathcal{O}(n^2)$  distinct substrings but there is just  $n$  subtrees – each node of the tree is the root of just one subtree. Just those substrings which themselves are trees in prefix notation are those which are the subtrees in prefix notation. This property

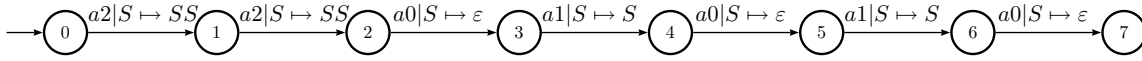


Fig. 4. Transition diagram of deterministic PDA  $M_p(t_1)$  accepting tree  $t_1$  in prefix notation  $pref(t_1) = a2 a0 a2 a0 a0 a0$  from Example 4.1.

is formalised by the following definition and theorem.

**Definition 2:** Let  $w = a_1 a_2 \dots a_m$ ,  $m \geq 1$ , be a string over a ranked alphabet  $\mathcal{A}$ . Then, the *arity checksum*  $ac(w) = Arity(a_1) + Arity(a_2) + \dots + Arity(a_m) - m + 1 = \sum_{i=1}^m Arity(a_i) - m + 1$ .

**Theorem 3:** Let  $pref(t)$  and  $w$  be a tree  $t$  in prefix notation and a substring of  $pref(t)$ , respectively. Then,  $w$  is the prefix notation of a subtree of  $t$ , if and only if  $ac(w) = 0$ , and  $ac(w_1) \geq 1$  for each  $w_1$ , where  $w = w_1 x$ ,  $x \neq \epsilon$ .

*Proof:* It is easy to see that for any two subtrees  $st_1$  and  $st_2$  it holds that  $pref(st_1)$  and  $pref(st_2)$  are either two different strings or one is a substring of the other. The former case occurs if the subtrees  $st_1$  and  $st_2$  are two different trees with no shared part and the latter case occurs if one tree is a subtree of the other tree. No partial overlapping of subtrees is possible in ranked ordered trees. Moreover, it holds for any two subtrees which are adjacent siblings that their prefix notations are two adjacent substrings.

– *If:* By induction on the height of a subtree  $st$ , where  $w = pref(st)$ :

- 1) We assume that  $Height(st) = 1$ , which means we consider the case  $w = a$ , where  $Arity(a) = 0$ . Then,  $ac(w) = 0$ . Thus, the claim holds for the case  $Height(st) = 1$ .
- 2) Assume that the claim holds for the subtrees  $st_1, st_2, \dots, st_p$  where  $p \geq 1$ ,  $Height(st_1) \leq m$ ,  $Height(st_2) \leq m$ ,  $\dots$ ,  $Height(st_p) \leq m$ ,  $ac(pref(st_1)) = 0$ ,  $ac(pref(st_2)) = 0$ ,  $\dots$ ,  $ac(pref(st_p)) = 0$ .

We are to prove that it holds also for a subtree of height  $m + 1$ . Assume  $w = a pref(st_1) pref(st_2) \dots pref(st_p)$ , where  $Arity(a) = p$ . Then

$$ac(w) = p + ac(pref(st_1)) + ac(pref(st_2)) + \dots + ac(pref(st_p)) - (p + 1) + 1 = 0 \text{ and } ac(w_1) \geq 1 \text{ for each } w_1, \text{ where } w = w_1 x, x \neq \epsilon.$$

Thus, the claim holds for the case  $Height(st) = m + 1$ .

– *Only if:* Assume  $ac(w) = 0$ , and  $w = a_1 a_2 \dots a_k$ , where  $k \geq 1$ ,  $Arity(a_1) = p$ . Since  $ac(w_1) \geq 1$  for each  $w_1$ , where  $w = w_1 x$ ,  $x \neq \epsilon$ , none of the substrings  $w_1$  can be a subtree in prefix notation. This means that the only possibility for  $ac(w) = 0$  is that  $w$  is of the form  $w = a pref(t_1) pref(t_2) \dots pref(t_p)$ , where  $p \geq 0$ , and

$t_1, t_2 \dots t_p$  are subtrees which are adjacent siblings. In such case  $ac(w) = p + 0 - (p + 1) + 1 = 0$ .

No other possibility of the form of  $w$  for  $ac(w) = 0$  is possible. Thus, the claim holds. ■

Thus, the theorem holds. ■

We note that in subtree matching PDAs, the arity checksum is computed by pushdown operations, where the contents of the pushdown store represents the corresponding arity checksum. For example, the empty pushdown store means that the corresponding arity checksum is equal to 0.

#### IV. SUBTREE MATCHING PUSHDOWN AUTOMATON

This section deals with the subtree matching PDA for trees in prefix notation: algorithms and theorems are given and the subtree matching PDA and its construction are demonstrated with an example.

**Definition 4:** Let  $s$  and  $pref(s)$  be a tree and its prefix notation, respectively. Given an input tree  $t$ , a subtree pushdown automaton constructed over  $pref(s)$  accepts all matches of tree  $s$  in the input tree  $t$  by final state.

First, we start with a PDA which accepts the whole subject tree in prefix notation. The construction of the PDA accepting a tree in prefix notation is described by Alg. 1. The constructed PDA is deterministic.

**Algorithm 1.** Construction of a PDA accepting a tree  $t$  in prefix notation  $pref(t)$ .

**Input:** A tree  $t$  over a ranked alphabet  $\mathcal{A}$ ; prefix notation  $pref(t) = a_1 a_2 \dots a_n$ ,  $n \geq 1$ .

**Output:** PDA  $M_p(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$ .

**Method:** 1.

- 1) For each state  $i$ , where  $1 \leq i \leq n$ , create a new transition

$$\delta(i - 1, a_i, S) = (i, S^{Arity(a_i)}), \text{ where } S^0 = \epsilon. \blacksquare$$

**Example 4.1:** The PDA constructed by Alg. 1, accepting the prefix notation  $pref(t_1) = a2 a2 a0 a1 a0 a1 a0$  of tree  $t_1$  from Example 2.1, is the deterministic PDA  $M_p(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_1, 0, S, \emptyset)$ , where the mapping  $\delta_1$  is a set of the following transitions:

State	Input	Pushdown Store
0	$a2\ a2\ a0\ a1\ a0\ a1\ a0$	$S$
1	$a2\ a0\ a1\ a0\ a1\ a0$	$S\ S$
2	$a0\ a1\ a0\ a1\ a0$	$S\ S\ S$
3	$a1\ a0\ a1\ a0$	$S\ S$
4	$a0\ a1\ a0$	$S\ S$
5	$a1\ a0$	$S$
6	$a0$	$S$
7	$\varepsilon$	$\varepsilon$
accept		

Fig. 5. Trace of deterministic PDA  $M_p(t_1)$  from Example 4.1 for tree  $t_1$  in prefix notation  $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ .

$$\begin{aligned}
\delta_1(0, a2, S) &= (1, SS) \\
\delta_1(1, a2, S) &= (2, SS) \\
\delta_1(2, a0, S) &= (3, \varepsilon) \\
\delta_1(3, a1, S) &= (4, S) \\
\delta_1(4, a0, S) &= (5, \varepsilon) \\
\delta_1(5, a1, S) &= (6, S) \\
\delta_1(6, a0, S) &= (7, \varepsilon)
\end{aligned}$$

The transition diagram of deterministic PDA  $M_p(t_1)$  is illustrated in Fig. 4. In that figure, for each transition rule  $\delta_1(p, a, \alpha) = (q, \beta)$  from  $\delta$ , the edge leading from state  $p$  to state  $q$  is labelled by the triplet of the form  $a|\alpha \mapsto \beta$ .

Fig. 5 shows the sequence of transitions (trace) performed by deterministic PDA  $M_p(t_1)$  for tree  $t_1$  in prefix notation. ■

**Theorem 5:** Let  $M = (\{Q, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$  be an input-driven PDA whose each transition from  $\delta$  is of the form  $\delta(q_1, a, S) = (q_2, S^i)$ , where  $i = \text{Arity}(a)$ .

Then, if  $(q_3, w, S) \vdash_M^+ (q_4, \varepsilon, S^j)$ , then  $j = ac(w)$ .

*Proof:* By induction on the length of  $w$ :

- 1) Assume  $w = a$ . Then,  $(q_3, a, S) \vdash_M (q_4, \varepsilon, S^j)$ , where  $j = \text{Arity}(a) = ac(a)$ . Thus, the claim holds for the case  $w = a$ .
- 2) Assume that claim holds for a string  $w = a_1a_2 \dots a_k$ , where  $k \geq 1$ . This means that  $(q_3, a_1a_2 \dots a_k, S) \vdash_M^k (q_4, \varepsilon, S^j)$ , where  $j = ac(a_1a_2 \dots a_k)$ . We have to prove that the claim holds also for  $w = a_1a_2 \dots a_k a$ . It holds that  $(q_3, a_1a_2 \dots a_k a, S) \vdash_M^k (q_4, a, S^j) \vdash_M (q_5, \varepsilon, S^l)$ , where  $l = j + \text{Arity}(a) - 1 = ac(w) + \text{Arity}(a) - 1 = \text{Arity}(a_1) + \text{Arity}(a_2) + \dots + \text{Arity}(a_k) - k + 1 + \text{Arity}(a) - 1 = ac(a_1a_2 \dots a_k a)$ .

Thus, the claim holds for the case  $w = a_1a_2 \dots a_k a$ .

Thus, the theorem holds. ■

The correctness of the deterministic PDA constructed by Alg. 1 which accepts trees in prefix notation, is described by the following lemma.

**Lemma 6:** Given a tree  $t$  and its prefix notation  $pref(t)$ , the PDA  $M_p(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$ , where  $n \geq 0$ , constructed by Alg. 1 accepts  $pref(t)$ .

*Proof:* By induction on the height of the tree  $t$ :

- 1) If tree  $t$  has just one node  $a$ , where  $\text{Arity}(a) = 0$ , then

$Height(t) = 0$ ,  $pref(t) = a$ ,  $\delta(0, a, S) = (1, \varepsilon) \in \delta$ ,  $(0, a, S) \vdash_{M_p(t)} (1, \varepsilon, \varepsilon)$  and the claim holds for that tree.

- 2) Assume that claim holds for trees  $t_1, t_2, \dots, t_p$ , where  $p \geq 1$ ,  $Height(t_1) \leq m$ ,  $Height(t_2) \leq m, \dots, Height(t_p) \leq m$ ,  $m \geq 0$ .

We have to prove that the claim holds also for each tree  $t$  such that

$pref(t) = a\ pref(t_1)pref(t_2) \dots pref(t_p)$ ,  $\text{Arity}(a) = p$ , and  $Height(t) \geq m + 1$ :

Since  $\delta(0, a, S) = (1, S^p) \in \delta$ , and

$(0, a\ pref(t_1)pref(t_2) \dots pref(t_p), S)$

$\vdash_{M_p(t)} (1, pref(t_1)pref(t_2) \dots pref(t_p), S^p)$

$\vdash_{M_p(t)}^* (i, pref(t_2) \dots pref(t_p), S^{p-1})$

$\vdash_{M_p(t)}^* \dots$

$\vdash_{M_p(t)}^* (j, pref(t_p), S)$

$\vdash_{M_p(t)}^* (k, \varepsilon, \varepsilon)$ ,

the claim holds for that tree.

Thus, the lemma holds. ■

We present the construction of the deterministic subtree matching PDA for trees in prefix notation. The construction consists of two steps. First, a nondeterministic subtree matching PDA is constructed by Alg. 2. This nondeterministic subtree matching PDA is an extension of the PDA accepting trees in prefix notation, which is constructed by Alg. 1. Second, the constructed nondeterministic subtree matching PDA is transformed to the equivalent deterministic subtree matching PDA. In spite of the fact that the determinisation of a nondeterministic PDA is not possible generally, the constructed nondeterministic subtree matching PDA is an input-driven PDA and therefore can be determinised [23].

**Algorithm 2.** Construction of a nondeterministic subtree matching PDA for a tree  $t$  in prefix notation  $pref(t)$ .

**Input:** A tree  $t$  over a ranked alphabet  $\mathcal{A}$ ; prefix notation  $pref(t) = a_1a_2 \dots a_n$ ,  $n \geq 1$ .

**Output:** Nondeterministic subtree matching PDA  $M_{nps}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \{n\})$ .

**Method:** 1.

- 1) Create PDA  $M_{nps}(t)$  as PDA  $M_p(t)$  by Alg. 1.
- 2) For each symbol  $a \in \mathcal{A}$  create a new transition  $\delta(0, a, S) = (0, S^{\text{Arity}(a)})$ , where  $S^0 = \varepsilon$ . ■

**Example 4.2:** The subtree matching PDA, constructed by Alg. 2 from tree  $t_1$  having prefix notation  $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ , is the nondeterministic PDA  $M_{nps}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_2, 0, S, \{7\})$ , where mapping  $\delta_2$  is a set of the following transitions:

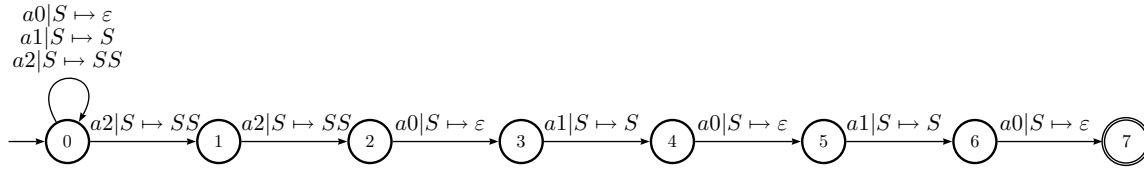


Fig. 6. Transition diagram of nondeterministic subtree matching PDA  $M_p(t_1)$  for tree  $t_1$  in prefix notation  $pref(t_1) = a2 a2 a0 a1 a0 a1 a0$  from Example 4.2.

$$\begin{aligned}
 \delta_2(0, a2, S) &= (1, SS) \\
 \delta_2(1, a2, S) &= (2, SS) & \delta_2(0, a2, S) &= (0, SS) \\
 \delta_2(2, a0, S) &= (3, \epsilon) & \delta_2(0, a1, S) &= (0, S) \\
 \delta_2(3, a1, S) &= (4, S) & \delta_2(0, a0, S) &= (0, \epsilon) \\
 \delta_2(4, a0, S) &= (5, \epsilon) \\
 \delta_2(5, a1, S) &= (6, S) \\
 \delta_2(6, a0, S) &= (7, \epsilon)
 \end{aligned}$$

The transition diagram of the nondeterministic PDA  $M_{nps}(t_1)$  is illustrated in Fig. 6. Again, in that figure, for each transition rule  $\delta_2(p, a, \alpha) = (q, \beta)$  from  $\delta_2$ , the edge leading from state  $p$  to state  $q$  is labelled by the triplet of the form  $a|\alpha \mapsto \beta$ . ■

**Theorem 7:** Given a tree  $s$  and its prefix notation  $pref(s)$ , the PDA  $M_{nps}(s)$  constructed by Alg. 2 is a subtree matching PDA for  $pref(s)$ .

*Proof:* According to Theorem 3, given an input tree  $t$ , each subtree in prefix notation is a substring of  $pref(t)$ . Since the PDA  $M_{nps}(s)$  has just states and transitions equivalent to the states and transitions, respectively, of the string matching automaton, the PDA  $M_{nps}(t)$  accepts all matches of subtree  $s$  in tree  $t$  by final state. ■

For the construction of deterministic subtree PDA, we use the transformation described by Alg. 3. This transformation is based on the well known transformation of nondeterministic finite automaton to an equivalent deterministic one, which constructs the states of the deterministic automaton as subsets of states of the nondeterministic automaton and selects only a set of accessible states (i.e. subsets) [12]. Again, states of the resulting deterministic PDA correspond to subsets of states of the original nondeterministic PDA.

**Algorithm 3.** Transformation of an input-driven nondeterministic PDA to an equivalent deterministic PDA.

**Input:** Input-driven nondeterministic PDA

$$M_{nx}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, F)$$

**Output:** Equivalent deterministic PDA

$$M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', q_I, S, F')$$

**Method:** 1.

- 1) Initially,  $Q' = \{\{0\}\}$ ,  $q_I = \{0\}$  and  $\{0\}$  is an unmarked state.
- 2) (a)
  - a) Select an unmarked state  $q'$  from  $Q'$ .
  - b) For each input symbol  $a \in \mathcal{A}$ :

- i)  $q'' = \{q : \delta(p, a, \alpha) = (q, \beta) \text{ for all } p \in q'\}$ .
  - ii) Add transition  $\delta'(q', a, S) = (q'', Arity(a))$ .
  - iii) If  $q'' \notin Q$  then add  $q''$  to  $Q$  and set it as unmarked state.
- c) Set state  $q'$  as marked.

3) Repeat step 2 until all states in  $Q'$  are marked.

4)  $F' = \{q' \mid q' \in Q' \wedge q' \cap F \neq \emptyset\}$ . ■

The deterministic subtree matching automaton  $M_{dps}(t)$  for a tree  $t$  with prefix notation  $pref(t)$  is demonstrated by the following example.

**Example 4.3:** The deterministic subtree matching PDA for tree  $t_1$  in prefix notation  $pref(t_1) = a2 a2 a0 a1 a0 a1 a0$  from Example 2.1, which has been constructed by Alg. 3 from nondeterministic subtree matching PDA  $M_{nps}(t_1)$  from Example 4.2, is the deterministic PDA  $M_{dps}(t_1) = (\{[0], [0, 1], [0, 1, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7]\}, \mathcal{A}, \{S\}, \delta_3, [0], S, \{\{0, 7\}\})$ ,

where mapping  $\delta_3$  is a set of the following transitions:

$$\begin{aligned}
 \delta_3([0], a0, S) &= ([0], \epsilon) & \delta_3([0, 4], a0, S) &= ([0, 5], \epsilon) \\
 \delta_3([0], a1, S) &= ([0], S) & \delta_3([0, 4], a1, S) &= ([0], S) \\
 \delta_3([0], a2, S) &= ([0, 1], SS) & \delta_3([0, 4], a2, S) &= ([0, 1], SS) \\
 \delta_3([0, 1], a0, S) &= ([0], \epsilon) & \delta_3([0, 5], a0, S) &= ([0], \epsilon) \\
 \delta_3([0, 1], a1, S) &= ([0], S) & \delta_3([0, 5], a1, S) &= ([0, 6], S) \\
 \delta_3([0, 1], a2, S) &= ([0, 1, 2], SS) & \delta_3([0, 5], a2, S) &= ([0, 1], SS) \\
 \delta_3([0, 1, 2], a0, S) &= ([0, 3], \epsilon) & \delta_3([0, 6], a0, S) &= ([0, 7], \epsilon) \\
 \delta_3([0, 1, 2], a1, S) &= ([0], S) & \delta_3([0, 6], a1, S) &= ([0], S) \\
 \delta_3([0, 1, 2], a2, S) &= ([0, 1, 2], SS) & \delta_3([0, 6], a2, S) &= ([0, 1], SS) \\
 \delta_3([0, 3], a0, S) &= ([0], \epsilon) & \delta_3([0, 7], a0, S) &= ([0], \epsilon) \\
 \delta_3([0, 3], a1, S) &= ([0, 4], S) & \delta_3([0, 7], a1, S) &= ([0], S) \\
 \delta_3([0, 3], a2, S) &= ([0, 1, 1], SS) & \delta_3([0, 7], a2, S) &= ([0, 1], SS)
 \end{aligned}$$

The transition diagram of deterministic PDA  $M_{dps}(t_1)$  is illustrated in Fig. 7. Again, in that figure, for each transition rule  $\delta_3(p, a, \alpha) = (q, \beta)$  from  $\delta_3$ , the edge leading from state  $p$  to state  $q$  is labelled by the triple of the form  $a|\alpha \mapsto \beta$ .

We note that the deterministic subtree matching PDA  $M_{dps}(t_1)$  has a very similar transition diagram to the string matching automaton constructed for  $pref(t_1)$  [5], [17], as can be seen by comparing Figs. 2 and 7.

Fig. 8 shows the sequence of transitions (trace) performed by the deterministic subtree PDA  $M_{dps}(t_1)$  for an input tree  $t_2$  in prefix notation  $pref(t_2) = a2 a2 a2 a0 a1 a0 a1 a0 a1 a1 a2 a0 a0$ . The accepting state is  $\{0, 7\}$ . Fig. 9 depicts the pattern subtree  $t_1$  and input tree  $t_2$ . ■

**Theorem 8:** Given a nondeterministic input-driven PDA  $M_{nx}(t) = (Q, \mathcal{A}, \{S\}, \delta, q_0, S, F)$ , the deterministic PDA

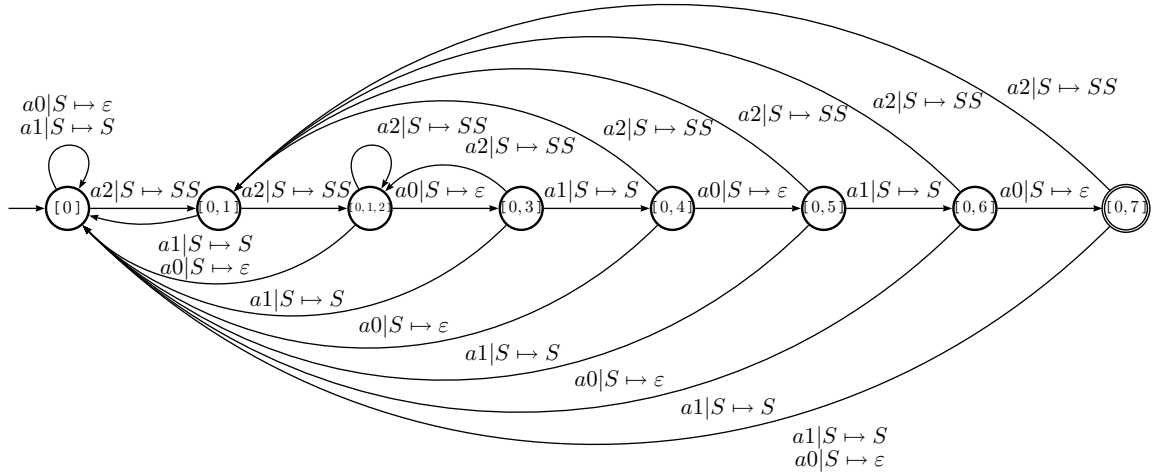


Fig. 7. Transition diagram of deterministic PDA  $M_{dps}(t_1)$  for tree  $t_1$  in prefix notation  $pref(t_1) = a2 a2 a0 a1 a0 a1 a0$  from Example 4.3.

State	Input	PDS
[0]	$a2 a2 a2 a0 a1 a0 a1 a0 a1 a1 a2 a0 a0$	$S$
[0, 1]	$a2 a2 a0 a1 a0 a1 a0 a1 a1 a2 a0 a0$	$SS$
[0, 1, 2]	$a2 a0 a1 a0 a1 a0 a1 a1 a2 a0 a0$	$SSS$
[0, 1, 2]	$a0 a1 a0 a1 a0 a1 a1 a2 a0 a0$	$SSSS$
[0, 3]	$a1 a0 a1 a0 a1 a1 a2 a0 a0$	$SSS$
[0, 4]	$a0 a1 a0 a1 a1 a2 a0 a0$	$SSS$
[0, 5]	$a1 a0 a1 a1 a2 a0 a0$	$SS$
[0, 6]	$a0 a1 a1 a2 a0 a0$	$SS$
[0, 7]	$a1 a1 a2 a0 a0$	match $S$
[0]	$a1 a2 a0 a0$	$S$
[0]	$a2 a0 a0$	$S$
[0, 1]	$a0 a0$	$SS$
[0]	$a0$	$S$
[0]	$\epsilon$	$\epsilon$

Fig. 8. Trace of deterministic subtree PDA  $M_{dps}(t_1)$  from Example 4.3 for an input subtree  $t_2$  in prefix notation  $pref(t_2) = a2 a2 a2 a0 a1 a0 a1 a0 a1 a1 a2 a0 a0$ .

$M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', \{q_0\}, S, F')$  constructed by Alg. 3 is equivalent to PDA  $M_{nx}(t)$ .

*Proof:* First, we prove the following claim by induction on  $i$ :

(\*):  $(q'_1, w, S) \vdash_{M_{dx}(t)}^i (q'_2, \epsilon, S^j)$  if and only if  $q'_2 = \{p : (q, w, S) \vdash_{M_{nx}(t)}^i (p, \epsilon, S^j) \text{ for some } q \in q'_1\}$ .

1) Assume  $i=1$ .

- *if:* if  $(q'_1, a, S) \vdash_{M_{dx}(t)} (q'_2, \epsilon, S^j)$ , then there exists a state  $q \in q'_1$ , where  $(q, a, S) \vdash_{M_{nx}(t)} (p, \epsilon, S^j)$ ,  $p \in q'_2$ .
- *only if:* if  $(q, a, S) \vdash_{M_{nx}(t)} (p, \epsilon, S^j)$ , then for each  $q'_1 \in Q'$ , where  $q \in q'_1$ , it holds that  $(q'_1, a, S) \vdash_{M_{dx}(t)} (q'_2, \epsilon, S^j)$ , where  $p \in q'_2$ .

2) Assume that claim (\*) holds for  $i = 1, 2, \dots, k, k \geq 1$ . This means that  $(q'_1, w, S) \vdash_{M_{dx}(t)}^k (q'_2, \epsilon, S^j)$  if and only if

$q'_2 = \{p : (q, S, w) \vdash_{M_{nx}(t)}^k (p, \epsilon, S^j) \text{ for some } q \in q'_1\}$ . We have to prove that claim (\*) holds also for  $i = k + 1$ .

- *if:* if  $(q'_1, w, S) \vdash_{M_{dx}(t)}^k (q'_2, a, S^l) \vdash_{M_{dx}(t)} (q'_3, \epsilon, S^j)$ , then there exists a state  $q \in q'_2$ , where  $(q, a, S^l) \vdash_{M_{nx}(t)} (p, \epsilon, S^j)$ ,  $p \in q'_3$ .
- *only if:* if  $(q_0, pref(t), S) \vdash_{M_{nx}(t)}^k (q, a, S^l) \vdash_{M_{nx}(t)} (p, \epsilon, S^j)$ , then for each  $q'_1 \in Q'$ , where  $q \in q'_1$ , it holds that  $(q'_1, a, S^l) \vdash_{M_{dx}(t)} (q'_2, \epsilon, S^j)$ , where  $p \in q'_2$ .

As a special case of claim (\*),  $(\{q_0\}, pref(t), S) \vdash_{M_{dx}(t)}^i (q', \epsilon, \epsilon)$  if and only if  $(q_0, S, pref(t)) \vdash_{M_{nx}(t)}^i (q_1, \epsilon, \epsilon)$ . Thus, the theorem holds.  $\blacksquare$

We note that trees with the structure  $pref(t) = (a1)^{n-1}a0$  represent strings. The deterministic subtree matching PDA for such trees has the same number of states and transitions as the deterministic substring matching automaton constructed for  $pref(t)$  and accepts the same language.

## V. COMPLEXITIES

In this section we present the time and space complexity for the construction phase of the deterministic subtree matching automaton along with the time complexity of the searching phase.

*Theorem 9:* Given a tree  $t$  with  $n$  nodes and its prefix notation  $pref(t)$ , the deterministic subtree matching PDA  $M_{pds}(t)$  constructed by Alg. 2 and 3 has exactly  $n + 1$  states, one pushdown symbol and  $|\mathcal{A}|(n + 1)$  transitions.

*Proof:* Since the subtree matching PDA is directly analogous to the string matching automaton, we can use the proof from [6], [17].  $\blacksquare$

*Theorem 10:* Given an input tree  $t$  with  $n$  nodes, the searching phase of the deterministic subtree matching automaton constructed by Algs. 2 and 3 over tree  $s$  with  $m$  nodes, that is finding all occurrences of subtree  $s$  in tree  $t$ , is  $\mathcal{O}(n)$ .

*Proof:* The searching phase consists of reading tree  $t$  once, symbol by symbol from left to right. The appropriate transition is taken each time a symbol is read, resulting in exactly  $n$  transitions. Occurrences of  $s$  are matched by transitions leading to the final state.  $\blacksquare$

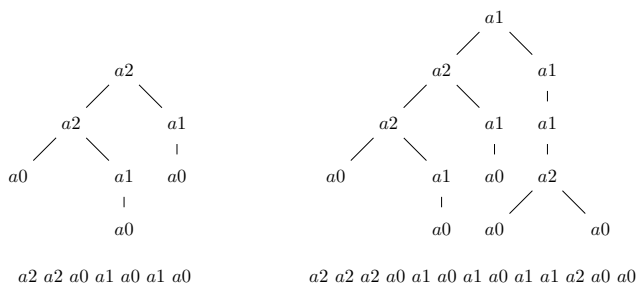


Fig. 9. Tree  $t_1 = a2\ a2\ a0\ a1\ a0\ a1\ a0$  and tree  $t_2 = a2\ a2\ a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$  from Example 4.3 along with their prefix notations.

## VI. CONCLUSION

In this paper, we have introduced a new kind of pushdown automata: subtree matching PDAs for trees in prefix notation. These pushdown automata are in their properties analogous to string matching automata, which are widely used in stringology [5], [6], [17], [21]. The presented subtree matching PDA is constructed over a tree  $t_1$  having  $m$  nodes and serving as the pattern, and the deterministic version allows to find all occurrences of the pattern  $t_1$  in a given tree  $t_2$  with  $n$  nodes in time linear in  $n$ .

Regarding specific tree algorithms whose model of computation is the standard deterministic pushdown automaton, recently we have introduced principles of three new basic algorithms. First, the tree pattern matching PDA [8], [16], which is an extension of the subtree matching PDA presented in this paper. Second, so-called subtree and tree pattern PDAs, which represent a complete index of the tree and the search phase of all occurrences of a subtree or a tree pattern, respectively, of size  $m$  is performed in time linear in  $m$  and not depending on the size of the tree [14], [15], [16]. These automata representing indexes of trees are analogous in their properties to the string suffix and factor automata [5], [6], [17], [21]. Third, a method how to find all repeats of connected subgraphs in trees with the use of subtree or tree pattern PDAs [18], [16]. More details on these results and related information can also be found on [2].

## REFERENCES

- [1] Aho, A. V., Ullman, J. D. *The Theory of Parsing, Translation and Compiling*. Vol. 1: Parsing, Vol. 2: Compiling, Prentice-Hall, New York, 1972.
- [2] *Arbology www pages*, Available on: <http://www.arbology.org>, August 2009.
- [3] Berstel, J. *Transductions and Context-Free Languages*. Teubner StudienbÄucher, Stuttgart, 1979.
- [4] Cleophas, L. *Tree Algorithms. Two Taxonomies and a Toolkit*. PhD thesis, Technische Universiteit Eindhoven, Eindhoven, 2008.
- [5] Crochemore, M., Hancart, Ch. Automata for Matching Patterns, In: *Vol 2: Linear Modeling: Background and Application. Handbook of Formal Languages*. pp. 399–462, Springer, Berlin, Heidelberg, 1997.
- [6] Crochemore, M., Rytter, W. *Jewels of Stringology*. World Scientific, New Jersey, 1994.
- [7] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M. *Tree Automata Techniques and Applications*, Available on: <http://www.grappa.univ-lille3.fr/tata>, release October, 12th 2007, 2007.
- [8] Flouri, T., Janoušek, J., Melichar, B. *Tree pattern matching by deterministic pushdown automata*, unpublished.
- [9] Geceşeg, F, Steinby, M. Tree Languages, In: *Vol 3: Beyond Words. Handbook of Formal Languages*. pp. 1–68, Springer, Berlin, Heidelberg, 1997.
- [10] Gibbons, A. and Rytter, W. *Efficient Parallel Algorithms*. Cambridge University Press, Cambridge, 1988.
- [11] Hoffman, C. M., O'Donnell M. J. *Pattern Matching in Trees*. Journal of ACM, vol. 29, pp. 68–95, 1982.
- [12] Hopcroft, J. E., Motwani, R., Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation*. 2nd edition, Addison-Wesley, New York, 2001.
- [13] Janoušek, J., Melichar, B. *On Regular Tree languages and Deterministic Pushdown Automata*, In: Acta Informatica, Springer, Berlin, in press.
- [14] Janoušek, J. String Suffix Automata and Subtree Pushdown Automata, In: *Proceedings of the Prague Stringology Conference 2009*, pp. 160–172, Czech Technical University in Prague, Prague, 2009.
- [15] Janoušek, J., Melichar, B. *Subtree and Tree Pattern Pushdown Automata for Trees in Prefix Notation*, unpublished.
- [16] *London Stringology Days 2009 Conference presentations*, Available on: <http://www.dcs.kcl.ac.uk/events/LSD&LAW09/>, King's College London, London, February 2009.
- [17] Melichar, B., Holub, J., Polcar, J. *Text Searching Algorithms*. Available on: <http://stringology.org/athens/>, release November 2005, 2005.
- [18] Melichar, B., Janoušek, J. *Repeats in Trees and Deterministic Tree Pattern Pushdown Automaton*. draft. 2009.
- [19] Sikkel, K., Salomaa, A. (Eds.) Vol. 1: Word, Language, Grammar, In: *Handbook of Formal Languages*. Springer, Berlin, Heidelberg, 1997.
- [20] Sikkel, K., Salomaa, A. (Eds.) *Handbook of Formal Languages*. Springer, Berlin, Heidelberg, 1997.
- [21] Smyth, B. *Computing Patterns in Strings*. Pearson, Addison Wesley, New York, 2000.
- [22] Valiant, L. G., Paterson, M. S. Deterministic one-counter automata. In: *LNCS*, vol. 2, Springer, Berlin, pp. 104–115, 1973.
- [23] Wagner, K., Wechsung, G. *Computational Complexity*. Springer, Berlin, Heidelberg, 2001.