KYBERNETIKA — VOLUME *48* (2012), NUMBER 3, PAGES 402–428

ARBOLOGY: TREES AND PUSHDOWN AUTOMATA

Bořivoj Melichar, Jan Janoušek and Tomáš Flouri

We present a unified and systematic approach to basic principles of Arbology, a new algorithmic discipline focusing on algorithms on trees. Stringology, a highly developed algorithmic discipline in the area of string processing, can use finite automata as its basic model of computation. For various kinds of linear notations of ranked and unranked ordered trees it holds that subtrees of a tree in a linear notation are substrings of the tree in the linear notation. Arbology uses pushdown automata reading such linear notations of trees as its basic model of computation. Basic principles known from stringology are used for the construction of particular arbology algorithms, in which the underlying tree structure is processed with the use of the pushdown store. Arbology results are shown for the basic problems subtree matching and tree indexing for ranked and unranked ordered trees.

Keywords: trees, pushdown automata, tree pattern matching, indexing trees, arbology

Classification: 05C05, 68Q68

1. INTRODUCTION

Trees are one of the fundamental hierarchical data structures used in Computer Science. Many methods have been described for solving various tree–related problems. However, most of them lack clear references to a systematic approach of the theory of formal languages, grammars and automata.

The theory of formal tree languages has been extensively studied and developed since the 1960s [5, 6, 10, 13]. Models of computation of this theory are various kinds of tree automata, which represent the generalisation of automata on strings to automata on trees. The most researched kind of tree automata are finite tree automata, which recognize regular tree languages, and their implementation is based on recursive procedures.

Linearising trees and using standard string automata reading the linear notations represents another approach for solving tree algorithms. Generally, every sequential algorithm working on a tree traverses nodes of the tree in a sequential order of nodes, which follows a corresponding linear notation of the tree. It holds that linear notations of trees are context–free languages. In [20] it is proved that (string) deterministic pushdown automata reading linear notations of trees are more powerful than finite tree automata: the class of tree languages whose postfix notation can be accepted by deterministic pushdown automata is a proper superclass of regular tree languages. These ways of reasoning can lead to the conclusion that a pushdown automaton reading linear notations

of trees can be an appropriate model of computation for tree algorithms.

Some particular tree algorithms based on pushdown automata are known, for example the Graham–Glanville technique used for code selection [14] or other similar methods [21, 25] using an LR–like pushdown automaton for ranked trees. Stringology [8, 9, 23, 26] is a highly developed systematic algorithmic discipline in the area of string processing and it can use finite automata as its basic model of computation. However, a systematic theory for tree algorithms which would be analogous to that of the stringology did not exist. In 2008 we founded a new algorithmic discipline called *arbology* from the Latin, Spanish, French words (arbor, arbol, arbre) meaning *tree*. Our main motivation for founding arbology was to apply some of the well–known principles of algorithms from stringology to trees so that effective analogous tree algorithms based on pushdown automata would be created. As is proved in this paper, for various kinds of linear notations of both ranked and unranked ordered trees, it holds that subtrees of a tree in a linear notation are substrings of the tree in the linear notation. This is a key property, which allows the use of some principles from stringology in the area arbology.

This paper presents an unified and systematic approach to basic arbology principles and results. Arbology results are shown for the basic problems subtree matching and tree indexing for ranked and unranked ordered trees. The paper is an extended journal version of contributions of two invited talks from two conferences [22, 19]. In comparison with [22], this paper contains the most important proofs which are omitted in [22] and also principles for processing unranked ordered trees.

The rest of the paper is organised as follows. Section 2 contains basic definitions. Section 3 deals with linear notations of ranked and unranked trees and some of their important properties. Basic pushdown automata for processing these linear notations are presented in Section 4. Section 5 is devoted to determinising pushdown automata. Section 6 deals with exact subtree matching. Subtree and tree pattern pushdown automata, which represent a complete index of a tree for subtrees and for tree patterns, respectively, are presented in Section 7. Sections 6 and 7 present the results for ranked trees in prefix notation. Section 8 describes analogous algorithms for processing the other linear notations, also for unranked trees. The last section is the conclusion.

2. BASIC NOTIONS

We define notions on trees similarly as they are defined in [1, 6, 13, 15].

2.1. Alphabet

An *alphabet* is a finite nonempty set of *symbols*. A *ranked alphabet* is a finite nonempty set of symbols each of which has a unique non–negative *arity* (or *rank*). Given a ranked alphabet $\mathcal{A}$, the arity of a symbol $a \in \mathcal{A}$ is denoted $Arity(a)$. The set of symbols of arity p is denoted by $\mathcal{A}_p$. Elements of arity $0, 1, 2, \ldots, p$ are respectively called nullary (constants), unary, binary, ..., p-ary symbols. We assume that $\mathcal{A}$ contains at least one constant. In the examples we use numbers at the end of the identifiers for a short declaration of symbols with arity. For instance, $a2$ is a short declaration of a binary symbol a.

2.2. Tree, tree pattern, tree template

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

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

A sequence of nodes $(f_0, f_1, \ldots, f_n)$, $n \geq 1$, 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 $(f_0, f_1, \ldots, f_n)$, 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 $G = (N, R)$ is a mapping of N 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 $g \in N$. By analogy, the *in-degree* of node f is the number of distinct pairs $(g, f) \in R$, where $g \in N$.

A *tree* is an acyclic connected graph. Any node of a tree can be selected as a *root* of the tree. A tree with a root is called *rooted tree*. Nodes of the tree with out-degree 0 are called *leaves*.

A tree can be *directed*. A *rooted and directed tree* t is a dag $t = (N, R)$ with a special node $r \in N$, 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 $f \neq r$.

A *labelled, (rooted, directed) tree* is a tree having the following property:

(4) every node $f \in N$ is labelled by a symbol $a \in \mathcal{A}$, where $\mathcal{A}$ is an alphabet.

An *ordered, (labelled, rooted, directed) tree* is a tree where direct descendants $a_{f1}, a_{f2}, \ldots, a_{fn}$ of a node a_f having an $Arity(a_f) = n$ are ordered.

A *ranked, (labelled, rooted, directed, ordered) tree* is a tree labelled by symbols from a ranked alphabet and out-degree of a node f labelled by symbol $a \in \mathcal{A}$ is $Arity(a)$. Nodes labelled by nullary symbols (constants) are leaves.

Similarly, an *unranked, (labelled, rooted, directed, ordered) tree* is a tree labelled by symbols from an unranked alphabet, which means that the out-degree of a node f labelled by symbol a is not given by the symbol a.

Example 2.1. Consider a ranked alphabet $\mathcal{A} = \{a2, a1, a0\}$. Consider an ordered, ranked, labelled, rooted, and directed tree $t_1 = (\{a2_1, a2_2, a0_3, a1_4, a0_5, a1_6, a0_7\}, R_1)$ over $\mathcal{A}$, where R_1 is a set of the following ordered sequences of pairs:

$$((a2_1, a2_2), (a2_1, a1_6)),$$
$$((a2_2, a0_3), (a2_2, a1_4)),$$
$$((a1_4, a0_5)),$$
$$((a1_6, a0_7)).$$

The unranked version of tree t_1 is tree $t_2 = (\{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}, R_2)$ over $\{a\}$, where R_2 is a set of the following ordered sequences of pairs:

$((a_1, a_2), (a_1, a_6)),$
$((a_2, a_3), (a_2, a_4)),$
$((a_4, a_5)),$
$((a_6, a_7)).$

Trees can be represented graphically, and trees t_1 and t_2 are illustrated in Figure 1.

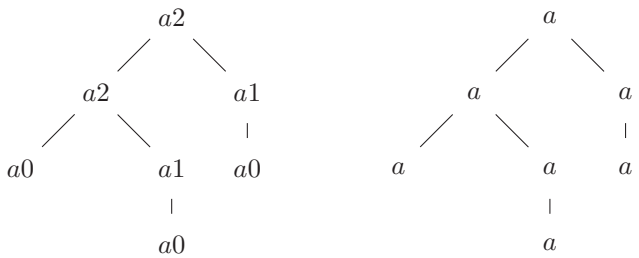


Fig. 1. Tree t_1 (left) and its unranked version tree t_2 (right) from Example 2.1.

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

To define a *tree pattern*, we use a special nullary symbol S, not in $\mathcal{A}$, $Arity(S) = 0$, which serves as a placeholder for any complete subtree. A tree pattern is defined as a labelled ordered tree over an alphabet $\mathcal{A} \cup \{S\}$. We will assume that the tree pattern contains at least one node labelled by a symbol from $\mathcal{A}$. A tree pattern containing at least one symbol S will be called a *tree template*.

A tree pattern p with $k \geq 0$ occurrences of the symbol S *matches* an object tree t at node n if there exist subtrees $t_1, t_2, \ldots, t_k$ (not necessarily the same) of the tree t such that the tree p', obtained from p by substituting the subtree t_i for the ith occurrence of S in p, $i = 1, 2, \ldots, k$, is equal to the subtree of t rooted at n.

Example 2.2. Consider tree $t_1 = (\{a2_1, a2_2, a0_3, a1_4, a0_5, a1_6, a0_7\}, R_1)$ from Example 2.1, which is illustrated in Figure 1. Consider a tree pattern p_1 over $\mathcal{A} \cup \{S\}$ $p_1 = (\{a2_1, a0_2, a1_3, a0_4\}, R_{p1})$, where R_{p1} is a set of the following ordered sequences of pairs:

$((a2_1, a0_2), (a2_1, a1_3)),$
$((a1_3, a0_4)).$

Consider a tree pattern (template) p_2 over $\mathcal{A} \cup \{S\}$ $p_2 = (\{a2_1, S_2, a1_3, S_4\}, R_{p2})$, where R_{p2} is a set of the following ordered sequences of pairs:

$((a2_1, S_2), (a2_1, a1_3)),$
$((a1_3, S_4)).$

Tree patterns p_1 and p_2 are illustrated in Figure 2. Tree pattern p_1 has one occurrence in tree t_1 – it matches at node 2 of t_1. Tree pattern p_2 has two occurrences in tree t_1 – it matches at nodes 1 and 2 of t_1.

Fig. 2. Tree pattern p_1 (left) and tree pattern (template) p_2 (right) from Example 2.2.

2.3. Language, grammar, finite and pushdown automata

We define notions from the theory of string languages similarly as they are defined in [1, 16].

A *language* over an alphabet $\mathcal{A}$ is a set of strings over $\mathcal{A}$. Symbol $\mathcal{A}^*$ denotes the set of all strings over $\mathcal{A}$ including the empty string, denoted by ε. Set $\mathcal{A}^+$ is defined as $\mathcal{A}^+ = \mathcal{A}^* \setminus \{\varepsilon\}$. Similarly, for string $x \in \mathcal{A}^*$, symbol 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 $x^+ = x^* \setminus \{\varepsilon\} = \{x^m : m \geq 1\}$.

A *nondeterministic finite automaton* (NFA) is a five-tuple $FM = (Q, \mathcal{A}, \delta, q_0, F)$, where Q is a finite set of *states*, $\mathcal{A}$ is an *input alphabet*, δ is a mapping from $Q \times \mathcal{A}$ into a set of finite subsets of Q, $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is the set of final (accepting) states. A finite automaton FM is *deterministic* (DFA) if $\delta(q, a)$ has no more than one member for any $q \in Q$ and $a \in \mathcal{A}$. We note that the mapping δ is often illustrated by its transition diagram.

Every NFA can be transformed to an equivalent DFA [1]. The transformation constructs the states of the DFA as subsets of states of the NFA and selects only such accessible states (ie subsets). These subsets are called *d–subsets*.

A *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*, δ 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 pushdown store symbol, and $F \subseteq Q$ is the set of final (accepting) states.

An *extended nondeterministic pushdown automaton* is a seven-tuple $M = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$, where δ is a mapping from $Q \times (\mathcal{A} \cup \{\varepsilon\}) \times G^*$ into a set of finite subsets of $Q \times G^*$ and all other symbols have the same meaning as above.

Triple $(q, w, x) \in Q \times \mathcal{A}^* \times G^*$ denotes the configuration of a pushdown automaton. We will write the top of the pushdown store x on its left hand side. The initial configuration of a pushdown automaton is a triple (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 kth power, transitive closure, and transitive and reflexive closure of the relation $\vdash_M$ is denoted $\vdash_M^k$, $\vdash_M^+$, $\vdash_M^*$, respectively.

An extended pushdown automaton M is an *deterministic* extended pushdown au-

tomaton (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 α is not a suffix of β and β is not a suffix of α.
3. If $\delta(q, a, \alpha) \neq \emptyset$, $\delta(q, \varepsilon, \beta) \neq \emptyset$, then α is not a suffix of β and β is not a suffix of α.

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 the pushdown automaton accepts the language by empty pushdown store, then the set F of final states is the empty set.

2.4. String pattern matching automata

Given a string pattern $p \in \mathcal{A}^*$, the string pattern matching automaton constructed for the string pattern p reads an input text and reaches its final state whenever the end of the string pattern has been read in the input text. In this way every occurrence of the string pattern in the text is found. The searching phase is performed in time linear to the length of the input text.

Example 2.3. Given a string $p_1 = a2\ a0\ a1\ a0$, which is a subtree of tree t_1 from Example 2.1 in prefix notation, the corresponding nondeterministic string pattern matching automaton is $FM_{npat}(p_1) = (\{0, 1, 2, 3, 4\}, \mathcal{A}, \delta_1, 0, \{4\})$, where its transition diagram is illustrated in Figure 3. (For the construction of the nondeterministic string pattern matching automaton, see [23].)

After the standard transformation of a nondeterministic string pattern matching automaton to a deterministic one [16], the deterministic string pattern matching automaton for p_1 is $FM_{dpat}(p_1) = (\{[0], [0, 1], [0, 2], [0, 3], [0, 4]\}, \mathcal{A}, \delta_2, [0], \{[0, 4]\})$, where its transition diagram is illustrated in Figure 4.

2.5. Indexing strings by finite automata

String suffix and factor automata are finite automata, and were introduced in [4, 7] as a mechanism for eliminating redundancy in string suffix trees [8, 9, 23, 26]. Given a string $s \in \mathcal{A}^*$, the suffix and factor automaton constructed for the string s accepts all suffixes and substrings, respectively, of the string s in time linear to the length of the input suffix and the input substring, respectively, and not depending on the length of the string s. In [8, 9, 26], suffix and factor automata are defined as such minimal deterministic finite automata. In [23], their basic nondeterministic versions are also presented. In some literature [9], the deterministic suffix automaton is also called the *directed acyclic word graph (DAWG).*

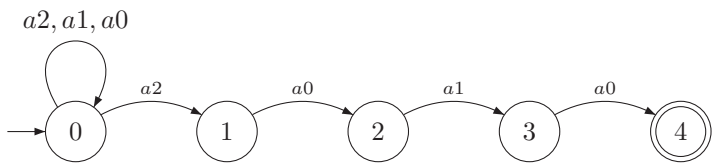


Fig. 3. Transition diagram of the nondeterministic string pattern matching automaton $FM_{npat}(p_1)$ for string pattern $p_1 = a2\ a0\ a1\ a0$ from Example 2.3.

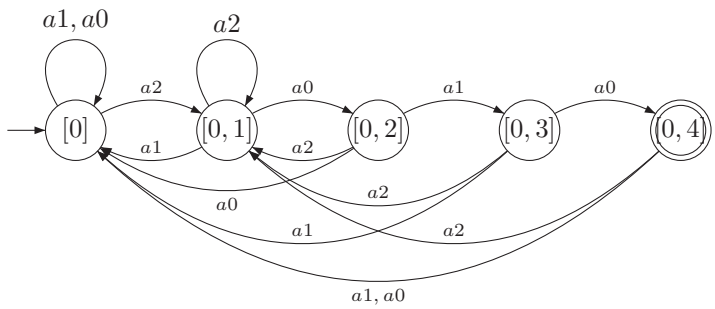


Fig. 4. Transition diagram of the deterministic string pattern matching automaton $FM_{dpat}(p_1)$ for string pattern $p_1 = a2\ a0\ a1\ a0$ from Example 2.3.

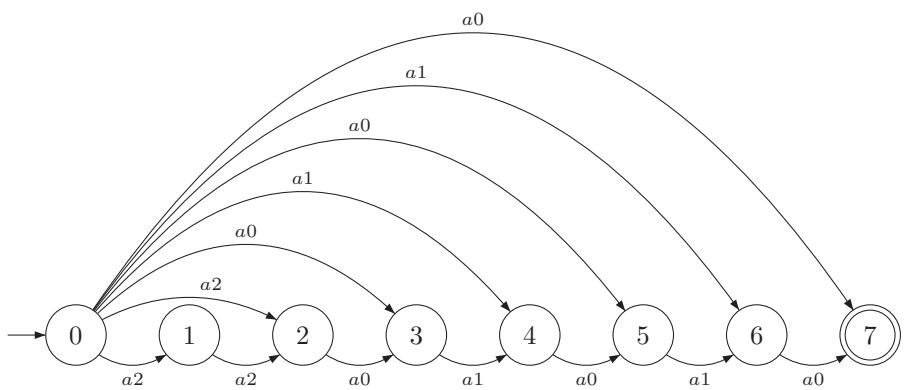


Fig. 5. Transition diagram of the nondeterministic string suffix automaton $FM_{nsuf}(p_2)$ for string $p_2 = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 2.4.

Example 2.4. Given a string $p_2 = a2\ a2\ a0\ a1\ a0\ a1\ a0$, which is tree t_1 from Example 2.1 in prefix notation, the corresponding nondeterministic suffix automaton is $FM_{nsuf}(p_2) = (\{0,1,2,3,4,5,6,7\}, \mathcal{A}, \delta_n, 0, \{7\})$, where its transition diagram is illustrated in Figure 5. (For the construction of the nondeterministic suffix automaton, see [23].)

After the standard transformation of a nondeterministic suffix automaton to a deterministic one [16], the deterministic suffix automaton for p_2 is
$FM_{dsuf}(p_2) = (\{[0],[1,2],[2],[3],[4],[5],[6],[7],[3,5,7],[4,6],[5,7]\},$
$\mathcal{A}, \delta_d, 0, \{[7],[3,5,7],[5,7]\})$, where its transition diagram is illustrated in Figure 6.

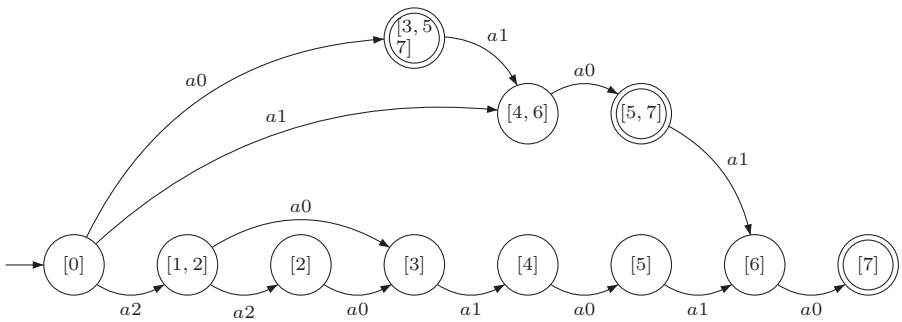


Fig. 6. Transition diagram of the deterministic suffix automaton $FM_{dsuf}(p_2)$ for string $p2 = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 2.4.

3. LINEAR NOTATIONS OF TREES

Every sequential algorithm traverses a processed tree in a sequential order of nodes, which creates a corresponding linear notation of the tree. In this paper we consider depth first oriented traversing of the processed tree in which every node is recorded just once during some visit.

3.1. Ranked trees

There are two standard one–visit depth–first oriented linear notations of ranked trees: prefix (also called preorder) notation, and postfix (also called postorder) notation.

Definition 3.1. The *prefix notation pref*(t) of a tree t is defined as follows:

1. *pref*(t) $= a$ if a_f is a leaf,
2. *pref*(t) $= a$ *pref*(b_1) *pref*(b_2) ... *pref*(b_n), where a is the root of the tree t and $b_1, b_2, \ldots\ b_n$ are direct descendants of a.

We note that in many papers on the theory of tree languages, such as [6, 13, 15], labelled ordered ranked trees are defined with the use of ordered ranked *ground terms*. Ground terms can be regarded as labelled ordered ranked trees in prefix notation.

Definition 3.2. The *postfix notation post*(t) of a tree t is defined as follows:

1. $post(t) = a$ if a_f is a leaf,
2. $post(t) = post(b_1)\ post(b_2) \dots post(b_n)\ a$, where a is the root of the tree t and $b_1, b_2, \dots\ b_n$ are direct descendants of a.

Example 3.3. Consider a ranked alphabet $\mathcal{A} = \{a2, a1, a0\}$. Consider a tree t_1 over $\mathcal{A}$ from Example 2.1, which is illustrated in Figure 1. Prefix and postfix notations of tree t_1 are strings $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ and $post(t_1) = a0\ a0\ a1\ a2\ a0\ a1\ a2$, respectively.

3.2. Unranked trees

Definitions of the basic prefix and postfix notations are very useful for ranked trees. If the tree is not ranked it is necessary to include information regarding the rank of every node. This can be done in two ways: (1) by representing a node in both notations as a pair $(a, Arity(a))$, (2) by using other principles of linearisation based on incorporating some special symbols. The second approach can be done using a prefix and a postfix bar notation in which each subtree is delimited by a bar symbol.

Definition 3.4. The *prefix bar notation pref_bar*(t) and *postfix bar notation post_bar*(t) of a tree t are defined as follows:

1. $pref\_bar(a) = a \uparrow$ and $post\_bar(a) = \uparrow a$ if a is a leaf,
2. $pref\_bar(t) = a\ pref\_bar(b_1)\ pref\_bar(b_2) \dots pref\_bar(b_n) \uparrow$ and $post\_bar(t) = \uparrow\ post\_bar(b_1)\ post\_bar(b_2) \dots post\_bar(b_n)\ a$ for prefix and postfix bar notation, respectively, where a is the root of the tree t and $b_1, b_2, \dots\ b_n$ are direct descendants of a.

Example 3.5. Consider an unranked tree t_2 from Example 2.1, which is illustrated in Figure 1. Prefix and postfix bar notations of tree t_2 are strings $pref\_bar(t_2) = a\ a\ a\ \uparrow\ a\ a\ \uparrow \uparrow \uparrow\ a\ a \uparrow \uparrow \uparrow$ and $post\_bar(t_2) = \uparrow \uparrow \uparrow\ a\ \uparrow \uparrow\ a\ a\ a\ \uparrow \uparrow\ a\ a\ a$, respectively.

3.3. Properties of one–visit linear notations of trees

It holds for any tree that each of its subtrees in a one–visit linear notation is a substring of the tree in the linear notation.

Theorem 1. *Given a tree t and its notations pref(t), post(t), pref_bar(t) and post_bar(t), all subtrees of t in prefix, postfix, prefix bar and postfix bar notation are substrings of pref(t), post(t), pref_bar(t) and post_bar(t), respectively.*

Proof. The proof for prefix notation can be found in [18]. Proofs for the other three notations can be made by analogy. For example, the proof for the prefix bar notation is as follows:
By induction on the height of the subtree:

1. If a subtree t' has just one node a, where $Arity(a) = 0$, then $Height(t') = 0$, $pref\_bar(t') = a \uparrow$ and the claim holds for that subtree.

2. Assume that the claim holds for subtrees $t_1, t_2, \ldots, t_p$, where $p \geq 1$, $Height(t_1) \leq m$, $Height(t_2) \leq m$, ..., $Height(t_p) \leq m$, $m \geq 0$. We have to prove that the claim also holds for each subtree $t' = at_1t_2 \ldots t_p$, where $Arity(a) = p$, $Height(t') = m + 1$:
   As $pref\_bar(t') = a\ pref\_bar(t_1)\ pref\_bar(t_2) \ldots pref\_bar(t_p) \uparrow$, the claim holds for the subtree t'. □

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

4. BASIC PUSHDOWN AUTOMATA FOR PARTICULAR LINEAR NOTATIONS OF TREES

In this section, we present basic pushdown automata which accept the one–visit linear notations of trees defined in the previous section and compute the underlying structure of trees by means of pushdown operations and pushdown store. These basic pushdown automata are defined by Definitions 4.1, 4.2, 4.5, and 4.6 for trees in prefix, postfix, prefix bar and postfix bar notations, respectively. They contain a pushdown symbol S and we would like to note that each pushdown symbol S which is present in the pushdown store during the reading of a tree in a linear notation corresponds to just one subtree.

4.1. Ranked trees

Basic pushdown automata for processing prefix and postfix notations of trees are defined as follows. We would like to note that the language of all trees in prefix notation is prefix–free, which means that no prefix of a sentence from the language is a sentence from the language. Therefore, the end of the input can be recognised by a pushdown automaton using no lookahead symbol. On the other hand, the language of all trees in postfix notation is not prefix–free. Therefore, in pushdown automata processing trees in postfix notation we add an appended right marker $\dashv$ to the input and the resulting language of trees in postfix notation with the appended right marker is prefix–free.

Definition 4.1. Let $\mathcal{A}$ be a ranked alphabet. Then, the *basic pushdown automaton for trees in prefix notation* is a PDA $M_p = (\{q_0\}, \mathcal{A}, \{S\}, \delta_p, q_0, S, \emptyset)$, where each transition from δ_p is of the form $\delta_p(q_0, a, S) = (q_0, S^i)$, where $a \in \mathcal{A}$, $i = arity(a)$.

Definition 4.2. Let $\mathcal{A}$ be a ranked alphabet. Then, the *basic pushdown automaton for trees in postfix notation* with an appended right marker $\dashv$ is a PDA $M_t = (\{q_0\}, \mathcal{A} \cup \{\dashv\}, \{Z_0, S\}, \delta_t, q_0, Z_0, \emptyset)$, where each transition from δ_p is of the form $\delta_t(q_0, a, S^i) = (q_0, S)$ and $\delta_t(q_0, \dashv, Z_0\ S) = (q_0, \varepsilon)$, where $a \in \mathcal{A}$, $i = Arity(a)$.

Example 4.3. Consider a ranked alphabet $\mathcal{A} = \{a2, a1, a0\}$. The transition diagrams of the basic pushdown automata for trees in prefix and postfix notation are illustrated in Figure 7.

Fig. 7. Transition diagrams of the basic pushdown automata for ranked alphabet $\mathcal{A} = \{a2, a1, a0\}$ and trees in prefix (left) and postfix (right) notation from Example 4.3.

The rest of this subsection is devoted to the behaviour of the basic pushdown automata for prefix and postfix notations. Prefix and postfix notations of ranked trees possess the following properties, which are described by a definition and two theorems.

Definition 4.4. 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 2. *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 proper prefix w_1 of w (ie. $w = w_1 x$, $x \neq \varepsilon$).*

Proof. In [18]. □

The dual principle holds for the postfix notation.

Theorem 3. *Let post(t) and w be a tree t in postfix notation and a substring of post(t), respectively. Then, w is the postfix notation of a subtree of t, if and only if $ac(w) = 0$, and $ac(w_1) \leq 0$ for each proper prefix w_1 of w (ie. $w = w_1 x$, $x \neq \varepsilon$).*

Proof. For any two different subtrees st_1 and st_2 it holds that $post(st_1)$ and $post(st_2)$ 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 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 trees. Moreover, it holds for any two subtrees which are adjacent siblings that their postfix notations are two adjacent substrings.

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

1. We assume that $Height(st) = 0$, which means we consider the case $w = a$. where $arity(a) = 0$. Then, $ac(w) = 0$. Thus, the claim holds for the case $Height(st) = 0$.
2. Assume that the claim holds for the subtrees $st_1, st_2, \ldots, st_p$ where $p \geq 1$, $Height(st_1) \leq m$, $Height(st_2) \leq m$, ..., $Height(st_p) \leq m$, $ac(post(st_1)) = 0$, $ac(post(st_2)) = 0$, ..., $ac(post(st_p)) = 0$.
   We are to prove that it also holds for a subtree w of height $m + 1$. Assume $w = post(st_1)\ post(st_2)\ \ldots post(st_p)\ a$, where $arity(a) = p$. Then $ac(w) = ac(post(st_1)) + ac(post(st_2)) + \ldots + ac(post(st_p)) + p - (p + 1) + 1 = 0$ and $ac(w_1) \geq 0$ for each w_1 , where $w = w_1 x$, $x \neq \varepsilon$.
   Thus, the claim holds for the case $Height(st) = m + 1$.

— *Only if*: Assume $ac(w) = 0$, and $w = a_1 a_2 \ldots a_m$, where $m \geq 1$, and $arity(a_m) = p$. Since $ac(w_1) \geq 0$ for each w_1, where $w = w_1 x$, $x \neq \varepsilon$, the only possibility for $ac(w) = 0$ is that w is of the form $w = post(t_1)\ post(t_2) \ldots\ post(t_p)\ a$, where $p \geq 0$, and $t_1, t_2 \ldots t_p$ are subtrees which are adjacent siblings. In such a case $ac(w) = 0 + p - (p + 1) + 1 = 0$.

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

Thus, the theorem holds. □

The basic pushdown automata for prefix and postfix notations compute the arity checksum by pushdown operations. Their behaviour is formally described by the following two theorems.

Theorem 4. *Let* $M_p = (\{\{q_0\}, \mathcal{A}, \{S\}, \delta_p, q_0, S, \emptyset)$ *be the basic pushdown automaton for trees in prefix notation. Then,* $(q_0, w_1 w_2, S) \vdash^+_{M_p} (q_0, w_2, S^j) \vdash^*_{M_p} (q_0, \varepsilon, \varepsilon)$*, where* $w_1 \in \mathcal{A}^+$*,* $w_2 \in \mathcal{A}^*$*, if and only if* $j = ac(w_1)$ *and* $w_1 w_2$ *is a tree in prefix notation.*

Proof. In [18]. □

Theorem 5. *Let* $M_t = (\{q_0\}, \mathcal{A} \cup \{\dashv\}, \{Z_0, S\}, \delta_{post}, q_0, Z_0, \emptyset)$ *be the basic pushdown automaton for trees in postfix notation with an appended right marker. Then,* $(q_0, w_1 w_2 \dashv, Z_0) \vdash^+_{M_t} (q_0, w_2 \dashv, Z_0\ S^j) \vdash^*_{M_t} (q_0, \varepsilon, \varepsilon)$*, where* $w_1 \in \mathcal{A}^+$*,* $w_2 \in \mathcal{A}^*$*, if and only if* $j = -ac(w_1) + 1$ *and* $w_1 w_2$ *is a tree in a postfix notation.*

Proof. The relation $(q_0, w_1 w_2 \dashv, Z_0) \vdash^+_{M_t} (q_0, w_2 \dashv, Z_0\ S^j)$, which matches with Theorem 3, is proved by induction on the length of w_1:

1. Assume $w_1 = a$. Then, $(q_0, a w_2 \dashv, Z_0) \vdash_{M_t} (q_0, w_2 \dashv, Z_0\ S)$, where $Arity(a) = 0$. Thus, the claim holds for the case $w_1 = a$.
2. Assume that the claim holds for a string $w_1 = a_1 a_2 \ldots a_m$, where $m \geq 1$. This means that $(q_0, a_1 a_2 \ldots a_m w_2 \dashv, Z_0) \vdash^m_M (q_0, w_2 \dashv, Z_0 S^j)$, where $j = -ac(a_1 a_2 \ldots a_m) + 1$. We have to prove that the claim also holds for $w_1 =$

$a_1a_2 \dots a_m a$. It holds that
$(q_0, a_1a_2 \dots a_m a w_2\dashv, Z_0) \vdash_M^m (q_0, a w_2\dashv, Z_0 S^j) \vdash_M (q_0, \varepsilon, Z_0 S^l)$,
where $l = j - arity(a) + 1 = -ac(w_1) - arity(a) + 1 = -ac(a_1a_2 \dots a_m a) + 1$.
Thus, the claim holds for the case $w_1 = a_1a_2 \dots a_m a$.

The relation $(q_0, w_2\dashv, Z_0\ S^j) \vdash_{M_t}^+ (q_0, \varepsilon, \varepsilon)$ holds if
$(q_0, w_2\dashv, Z_0\ S^j) \vdash_{M_t}^* (q_0, \dashv, Z_0 S) \vdash_{M_t} (q_0, \varepsilon, \varepsilon)$. This means that $ac(w_1w_2) = 0$, which matches with Theorem 3. □

4.2. Unranked trees

Basic pushdown automata for trees in prefix bar notation and in postfix bar notation are defined by the following two definitions. We would like to note that no appended right marker to the input is needed as in the case of postfix notation, because the languages of trees in prefix bar notation and of trees in postfix bar notation are prefix–free and therefore the end of the input can be recognised by a pushdown automaton using no lookahead symbol.

Definition 4.5. Let $\mathcal{A}$ be an alphabet. Then, the *basic pushdown automaton for trees in prefix bar notation* is a PDA $M_{pb} = (\{q_0\}, \mathcal{A} \cup \{\uparrow\}, \{Z_0, S\}, \delta_{pb}, q_0, Z_0, \emptyset)$, where each transition from δ_{pb} is of the form $\delta_{pb}(q_0, a, Z_0) = (q_0, S)$,
$\delta_{pb}(q_0, a, S) = (q_0, SS)$, $\delta_{pb}(q_0, \uparrow, S) = (q_0, \varepsilon)$, where $a \in \mathcal{A}$.

Definition 4.6. Let $\mathcal{A}$ be an alphabet. Then, the *basic pushdown automaton for trees in postfix bar notation* is a PDA $M_{tb} = (\{q_0\}, \mathcal{A} \cup \{\uparrow\}, \{Z_0, S\}, \delta_{tb}, q_0, Z_0, \emptyset)$, where each transition from δ_{tb} is of the form $\delta_{tb}(q_0, \uparrow, Z_0) = (q_0, S)$,
$\delta_{tb}(q_0, \uparrow, S) = (q_0, SS)$, $\delta_{tb}(q_0, a, S) = (q_0, \varepsilon)$, where $a \in \mathcal{A}$.

Example 4.7. Consider an alphabet $\mathcal{A} = \{a\}$. The transition diagrams of the basic pushdown automata for trees in prefix bar and postfix bar notation are illustrated in Figure 8.

Fig. 8. Transition diagrams of the basic pushdown automata for alphabet $\mathcal{A} = \{a\}$ and trees in prefix bar (left) and postfix bar (right) notation from Example 4.7.

The rest of this subsection is devoted to the behaviour of the basic pushdown automata for prefix bar and postfix bar notations. A similar definition and similar theorems

as for prefix and postfix notations hold for prefix and postfix bar notations. Instead of arity checksum *ac* a bar checksum *bc* is defined.

Definition 4.8. Given a string $w \in (\mathcal{A} \cup \{\uparrow\})^+$, its *bar checksum* $bc(w)$ is defined as follows:

1. $bc(a) = 1$, $a \in \mathcal{A}$, and $bc(\uparrow) = -1$,
2. $bc(b\ w_1) = bc(b) + bc(w_1)$, $b \in \mathcal{A} \cup \{\uparrow\}$, $w_1 \in (\mathcal{A} \cup \{\uparrow\})^*$.

Theorem 6. *Let pref_bar(t) and w be a tree t in prefix bar notation and a substring of pref_bar(t), respectively. Then, w is the prefix bar notation of a subtree of t, if and only if* $bc(w) = 0$, *and* $bc(w_1) \geq 1$ *for each proper prefix* w_1 *of* w *(ie.* $w = w_1 x$, $x \neq \varepsilon$*).*

Proof.

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

1. We assume that $Height(st) = 0$, which means we consider the case $w = a \uparrow$. Then, $bc(w) = 0$. Thus, the claim holds for the case $Height(st) = 0$.
2. Assume that the claim holds for the subtrees $st_1, st_2, \ldots, st_p$ where $p \geq 1$, $Height(st_1) \leq m$, $Height(st_2) \leq m$, …, $Height(st_p) \leq m$, $bc(pref\_bar(st_1)) = 0$, $bc(pref\_bar(st_2)) = 0$, …, $bc(pref\_bar(st_p)) = 0$.
   We are to prove that it also holds for a subtree w of height $m + 1$. Assume $w = a\ pref\_bar(st_1)\ pref\_bar(st_2)\ \ldots pref\_bar(st_p) \uparrow$. Then $bc(w) = bc(pref\_bar(st_1)) + bc(pref\_bar(st_2)) + \ldots + bc(pref\_bar(st_p)) + 1 - 1 = 0$ and $bc(w_1) \geq 1$ for each w_1 , where $w = w_1 x$, $x \neq \varepsilon$.
   Thus, the claim holds for the case $Height(st) = m + 1$.

— *Only if*: Assume $bc(w) = 0$, and $w = a_1 a_2 \ldots a_m$, where $m \geq 1$. Since $bc(w_1) \geq 1$ for each w_1, where $w = w_1 x$, $x \neq \varepsilon$, the only possibility for $bc(w) = 0$ is that w is of the form
$w = a\ pref\_bar(t_1)\ pref\_bar(t_2) \ldots pref\_bar(t_p) \uparrow$, where $p \geq 0$, and $t_1, t_2 \ldots t_p$ are subtrees which are adjacent siblings. In such a case $bc(w) = 1 + 0 - 1 = 0$.

There is no other possible form of w for $bc(w) = 0$.

Thus, the theorem holds. □

Theorem 7. *Let post_bar(t) and w be a tree t in postfix bar notation and a substring of post_bar(t), respectively. Then, w is the postfix bar notation of a subtree of t, if and only if* $bc(w) = 0$, *and* $bc(w_1) \leq -1$ *for each proper suffix* w_1 *of* w *(ie.* $w = x w_1$, $x \neq \varepsilon$*).*

Proof. The proof can be done by analogy with the proof of Theorem 6. □

The basic pushdown automata for prefix bar and postfix bar notations compute the bar checksum by pushdown operations. Their behaviour is formally described by the following two theorems.

Theorem 8. *Let* $M_{pb} = (\{q_0\}, \mathcal{A} \cup \{\uparrow\}, \{Z_0, S\}, \delta_{pb}, q_0, Z_0, \emptyset)$ *be the basic pushdown automaton for trees in prefix bar notation. Then,* $(q_0, w_1 w_2, Z_0) \vdash^+_{M_{pb}} (q_0, w_2, S^j) \vdash^*_{M_{pb}} (q_0, \varepsilon, \varepsilon)$*, where* $w_1 \in (\mathcal{A} \cup \{\uparrow\})^+$*,* $w_2 \in (\mathcal{A} \cup \{\uparrow\})^*$*, if and only if* $j = bc(w_1)$ *and* $w_1 w_2$ *is a tree in a prefix bar notation.*

Proof. The statement $j = bc(w_1)$ follows directly from the transition function and Def. 4.8. The sequence of moves $(q_0, w_1 w_2, Z_0) \vdash^+_{M_{pb}} (q_0, w_2, S^j) \vdash^*_{M_{pb}} (q_0, \varepsilon, \varepsilon)$ is possible because the pushdown store contains at least one symbol before each of its moves. $w_1 w_2$ is a tree in a prefix bar notation because it matches with Theorem 6. □

Theorem 9. *Let* $M_{tb} = (\{q_0\}, \mathcal{A} \cup \{\uparrow\}, \{Z_0, S\}, \delta_{tb}, q_0, Z_0, \emptyset)$ *be the basic pushdown automaton for trees in postfix bar notation. Then,* $(q_0, w_1 w_2, Z_0) \vdash^+_{M_{tb}} (q_0, w_2, S^j) \vdash^*_{M_{tb}} (q_0, \varepsilon, \varepsilon)$*, where* $w_1 \in (\mathcal{A} \cup \{\uparrow\})^+$*,* $w_2 \in (\mathcal{A} \cup \{\uparrow\})^*$*, if and only if* $j = -bc(w_1)$ *and* $w_1 w_2$ *is a tree in a postfix bar notation.*

Proof. The sequence of moves $(q_0, w_1 w_2, Z_0) \vdash^+_{M_{pb}} (q_0, w_2, S^j) \vdash^*_{M_{pb}} (q_0, \varepsilon, \varepsilon)$ is possible because the pushdown store contains at least one symbol before each of its move. $w_1 w_2$ is a tree in a postfix bar notation because it matches with Theorem 7. □

5. DETERMINISING PUSHDOWN AUTOMATA

No universal algorithm for determinising nondeterministic pushdown automata to equivalent deterministic ones exists. We have identified three classes of nondeterministic pushdown automata for which algorithms for determinising are known. They are called input–driven, visible [2] and height–deterministic pushdown automata [24]. This section describes a simple algorithm of the determinisation of input–driven pushdown automata, which is used in the rest of this paper.

The notion of a pushdown operation will frequently be used in the following meaning.

Definition 5.1. Let $M = (Q, A, G, \delta, q_0, Z_0, F)$ be a pushdown automaton. Let $\delta(q, a, \alpha)$ contain a pair (p, β) for $p, q \in Q$, $a \in A \cup \varepsilon$, $\alpha, \beta \in G^*$. Then the notation $\alpha \mapsto \beta$ is used for the operation popping α from the top of the pushdown store and pushing β to the top of the pushdown store. This operation is called a *pushdown operation.*

Definition 5.2. A pushdown automaton $M = (Q, A, G, \delta, q_0, Z_0, F)$ is an input–driven pushdown automaton if each pushdown operation $\alpha \mapsto \beta$ during every transition is explicitly determined by the input symbol. In more formal notation: For each $q \in Q$ and $a \in \mathcal{A} \cup \{\varepsilon\}$ there exists the only mapping $\delta(q, a, \alpha) = \{(p_1, \beta), (p_2, \beta), ..., (p_m, \beta)\}$ for one pair $\alpha, \beta \in G^*$ and $p_1, p_2, \ldots, p_m \in Q$.

Given a nondeterministic input–driven PDA, it can be determinised as follows:

Algorithm 1. Transformation of an input–driven nondeterministic PDA to an equivalent deterministic PDA.
Input: Input–driven nondeterministic PDA $M_n(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, G,$
$\delta, 0, Z_0, \emptyset)$.
Output: Equivalent deterministic PDA $M_d(t) = (Q', \mathcal{A}, G, \delta', q_I, Z_0, \emptyset)$.
Method:

1. Initially, $Q' = \{[0]\}$, $q_I = [0]$, and $[0]$ is an unmarked state.
2. (a) Select an unmarked state q' from Q'.
   (b) Add transition $\delta'(q', a, \alpha) = (q'', \beta)$, where $q'' = \{q : \ \delta(p, a, \alpha) = (q, \beta)$ for all $p \in q'\}$. If q'' is not in Q' then add q'' to Q' as an unmarked state.
   (c) Set the state q' as marked.
3. Repeat step 2 until all states in Q' are marked.

The correctness of Algorithm 1 is proved in [18].

6. EXACT SUBTREE MATCHING

This section deals with subtree matchers by deterministic pushdown automata, which read subject trees in prefix notation. 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 determinised. The pushdown automaton and its pushdown operations are constructed according to Theorems 2 and 4, where the pushdown operations check the prefix notation of the processed tree.

Definition 6.1. Let s and *pref*(s) be a tree and its prefix notation, respectively. Given an input tree t, a *subtree matching pushdown automaton constructed over pref*(s) accepts all matches of tree s in the input tree t by final state.

By analogy with stringology, the nondeterministic subtree matching pushdown automaton can be constructed in the following way.

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 $pref(t) = a_1 a_2 \dots a_n$, $n \geq 1$.
Output: Nondeterministic subtree matching PDA $M_{nps}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \{n\})$.
Method:

1. For each state i, where $1 \leq i \leq n$, create a new transition $\delta(i-1, a_i, S) = (i, S^{Arity(a_i)})$, where $S^0 = \varepsilon$.
2. For each symbol $a \in \mathcal{A}$ create a new transition $\delta(0, a, S) = (0, S^{Arity(a)})$, where $S^0 = \varepsilon$.

The correctness of Algorithm 2 is proved in [12].

For constructing deterministic subtree matching PDA, we use the transformation described by Algorithm 1. In [12] it is proved that given a tree t with n nodes in its prefix or postfix notation, the deterministic subtree matching PDA $M_{pds}(t)$ constructed by Algorithm 2 and 1 is made of exactly $n+1$ states, one pushdown symbol and $|\mathcal{A}|(n+1)$ transitions.

Given an input tree t with n nodes, the searching phase of the deterministic subtree matching automaton constructed by Algorithms 2 and 1 is $\mathcal{O}(n)$, which is also proved in [12].

Example 6.2. Consider a ranked alphabet $\mathcal{A} = \{a0, a1, a2\}$. Consider a subtree pattern p_1 from Example 2.2, where $pref(p_1) = a2\ a0\ a1\ a0$. The corresponding nondeterministic subtree matching PDA constructed by Algorithm 2 is PDA $M_{nps}(p_1) = (\{0, 1, 2, 3, 4\}, \mathcal{A}, \delta_{nps(p_1)}, \{S\}, 0, S, \{4\})$, where its transition diagram is illustrated in Figure 9.

After determinising according to Algorithm 1, the deterministic subtree matching PDA for p_1 is
$FM_{dps}(p_1) = (\{[0], [0, 1], [0, 2], [0, 3], [0, 4]\}, \mathcal{A}, \delta_{dps(p_1)}, \{S\}, [0], S, \{[0, 4]\})$, where its transition diagram is illustrated in Figure 10.

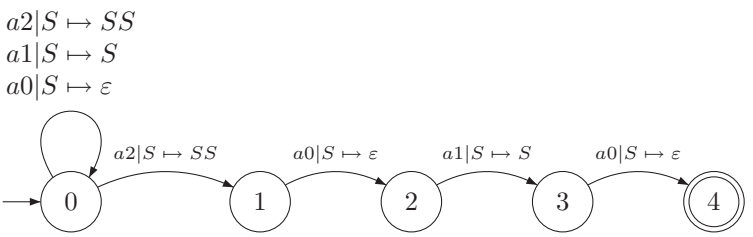


Fig. 9. Transition diagram of the nondeterministic subtree matching automaton $FM_{nps}(p_1)$ for tree pattern in prefix notation $pref(p_1) = a2\ a0\ a1\ a0$ from Example 6.2.

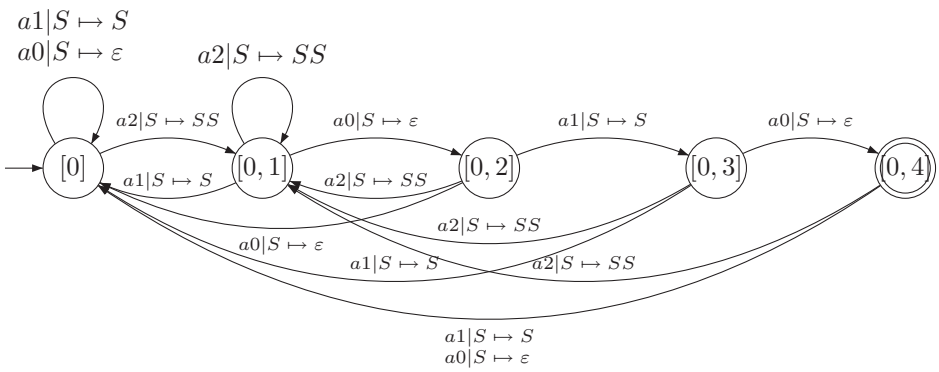
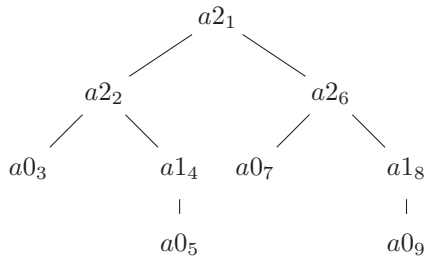


Fig. 10. Transition diagram of the deterministic subtree matching automaton $M_{dps}(p_1)$ from Example 6.2.

Consider a tree t_3 over $\mathcal{A}$ $t_3 = (\{a2_1, a2_2, a0_3, a1_4, a0_5, a2_6, a0_7, a1_8, a0_9\}, R_2)$ illustrated in Figure 11. $pref(t_3) = a2\ a2\ a0\ a1\ a0\ a2\ a0\ a1\ a0$.

Given a tree t_3 the sequence of moves by PDA $M_{dps}(p_1)$ is shown in Figure 12. There are two matching actions in this sequence, which means there are two occurrences of subtree p_1 in tree t_3 at the corresponding positions.



$$pref(t_3) = a2a2a0a1a0a2a0a1a0$$

Fig. 11. Tree t_3 from Example 6.2 and its prefix notation.

State	Input	Pushdown Store	Action
$[0]$	$a2\ a2\ a0\ a1\ a0\ a2\ a0\ a1\ a0$	S	
$[0,1]$	$a2\ a0\ a1\ a0\ a2\ a0\ a1\ a0$	$S\ S$	
$[0,1]$	$a0\ a1\ a0\ a2\ a0\ a1\ a0$	$S\ S\ S$	
$[0,2]$	$a1\ a0\ a2\ a0\ a1\ a0$	$S\ S$	
$[0,3]$	$a0\ a2\ a0\ a1\ a0$	$S\ S$	
$[0,4]$	$a2\ a0\ a1\ a0$	S	match
$[0,1]$	$a0\ a1\ a0$	$S\ S$	
$[0,2]$	$a1\ a0$	S	
$[0,3]$	$a0$	S	
$[0,4]$	ε	ε	match
accept			

Fig. 12. Trace of deterministic pushdown automaton M_{dpt} from Example 6.2.

Subtree matching for multiple subtrees can easily be performed by a PDA created as a union of subtree matching PDAs for the particular subtrees, for more details see [12].

7. INDEXING TREES

This section describes subtree PDAs [17] and tree pattern PDAs [18] for a tree in prefix notation. Subtree pushdown automata accept all subtrees of the tree. Tree pattern pushdown automata accept all tree patterns which match the tree. Given a tree with n nodes, the deterministic subtree and the deterministic tree pattern pushdown automaton 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 n. The total size of the deterministic subtree pushdown automaton is linear in n. Although the number of distinct tree patterns which match the tree can be

exponential in n, for specific cases of trees the total size of the deterministic tree pattern pushdown automaton is linear in n.

7.1. Indexing for subtrees

Definition 7.1. Let t and *pref*(t) be a tree and its prefix notation, respectively. A *subtree pushdown automaton* for *pref*(t) accepts all subtrees of t in prefix notation.

First, we start with a PDA which accepts the whole subject tree in prefix notation by empty pushdown store, the construction of which is described by Algorithm 3. The constructed PDA is deterministic.

Algorithm 3. Construction of a PDA accepting *pref*(t) by empty pushdown store.
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. For each state i, where $1 \leq i \leq n$, create a new transition $\delta(i-1, a_i, S) = (i, S^{Arity(a_i)})$, where $S^0 = \varepsilon$.

The correctness of Algorithm 3 is proved in [18].

The construction of the deterministic subtree PDA for trees in prefix notation consists of two steps. First, a nondeterministic subtree PDA is constructed by Algorithm 4. This nondeterministic subtree PDA is an extension of the PDA accepting tree in prefix notation, which is constructed by Algorithm 3. Second, the constructed nondeterministic subtree PDA is input–driven and therefore can easily be determinised.

Algorithm 4. Construction of a nondeterministic subtree PDA for a tree t in prefix notation *pref*(t).
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation $pref(t) = a_1 a_2 \dots a_n$, $n \geq 1$.
Output: Nondeterministic PDA $M_{nps}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

1. Create PDA $M_{nps}(t)$ as PDA $M_p(t)$ by Algorithm 3.
2. For each state i, where $2 \leq i \leq n$, create a new transition $\delta(0, a_i, S) = (i, S^{Arity(a_i)})$, where $S^0 = \varepsilon$.

The correctness of Algorithm 4 is proved in [18].

Example 7.2. A subtree pushdown automaton 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 Algorithm 4, is nondeterministic pushdown automaton
$M_{nps}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_2, 0, S, \emptyset)$, where its transition diagram is illustrated in Figure 13.

The deterministic subtree pushdown automaton, which has been constructed according to Algorithm 1 from nondeterministic subtree pushdown automaton $M_{nps}(t_1)$, is deterministic pushdown automaton
$M_{dps}(t_1) = (\{[0], [1, 2], [2], [3], [4], [5], [6], [7], [3, 5, 7], [4, 6], [5, 7]\}, \mathcal{A}, \{S\}, \delta_3, [0], S, \emptyset)$,

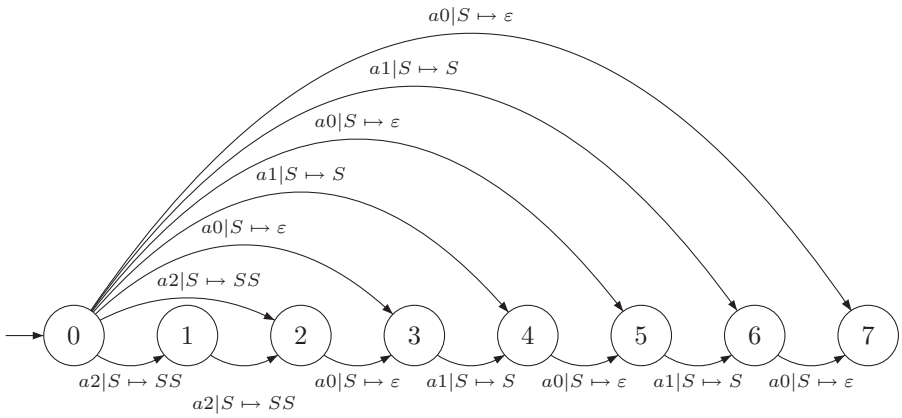


Fig. 13. Transition diagram of nondeterministic subtree pushdown automaton $M_{nps}(t_1)$ for tree t_1 in prefix notation $\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 7.2.

where its transition diagram is illustrated in Figure 14. We note that the nondeterministic subtree pushdown automaton is always acyclic and therefore the contents of the pushdown store of the deterministic pushdown automaton in particular states can be computed beforehand. Consequently, all transitions leading from states $[3, 5, 7]$ and $[5, 7]$ can be omitted because the contents of the pushdown store in these state is always ε and therefore no transition is possible from these states due to the pushdown operations. This means that deterministic subtree pushdown automaton $M_{dps}(t_1)$ has fewer transitions than the deterministic string suffix automaton constructed for $\mathit{pref}(t_1)$ [8, 23, 26] shown in Figure 6.

Figure 15 shows the sequence of transitions (the trace) performed by deterministic subtree pushdown automaton $M_{dps}(t_1)$ for an input subtree p_3 in prefix notation $\mathit{pref}(p_3) = a_1a_0$. The accepting state is $[5, 7]$, which means there are two occurrences of the input subtree st in tree t_1 and their rightmost leaves are nodes $a0_5$ and $a0_7$.

Given a tree t with n nodes, the number of distinct subtrees of tree t is equal or smaller than n. The deterministic subtree PDA has only one pushdown symbol S, and all its states and transitions correspond to the states and the transitions, respectively, of the deterministic suffix automaton constructed for $\mathit{pref}(t)$. Therefore, the total size of the deterministic subtree PDA cannot be greater than the total size of the deterministic suffix automaton constructed for $\mathit{pref}(t)$ [9]. This means that given a tree t with n nodes and its prefix notation $\mathit{pref}(t)$, the deterministic subtree PDA $M_{dps}(t)$ constructed by Algorithms 4 and 1 has just one pushdown symbol, fewer than $N \leq 2n + 1$ states and at most $N + n - 1 \leq 3n$ transitions.

7.2. Indexing for tree patterns

Definition 7.3. Let t and $\mathit{pref}(t)$ be a tree and its prefix notation, respectively. A

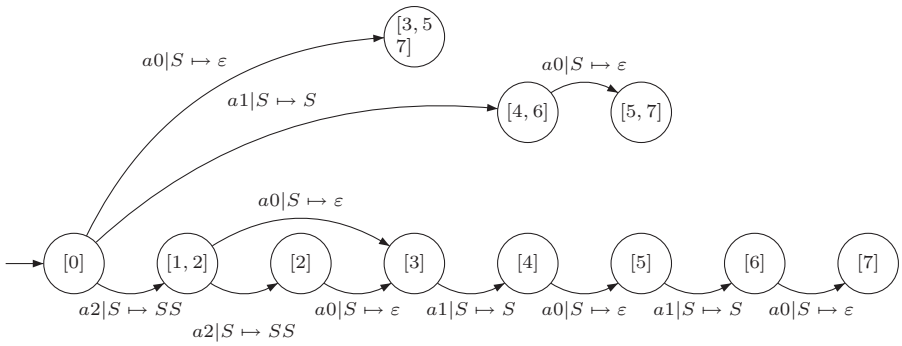


Fig. 14. Transition diagram of deterministic subtree pushdown automaton $M_{dps}(t_1)$ for tree in prefix notation $\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 7.2.

State	Input	Pushdown Store
[0]	$a1\ a0$	S
$[4, 6]$	$a0$	S
$[5, 7]$	ε	ε
accept		

Fig. 15. Trace of deterministic subtree pushdown automaton $M_{dps}(t_1)$ from Example 7.2.

tree pattern pushdown automaton for *pref*(t) accepts all tree patterns in prefix notation which match the tree t.

Given a subject tree, first we construct a so–called deterministic *treetop PDA* for this tree in prefix notation, which accepts all tree patterns that contain the root of the subject tree and match the subject tree. The deterministic treetop PDA is defined as follows.

Definition 7.4. Let t, r and *pref*(t) be a tree, its root and its prefix notation, respectively. A *treetop pushdown automaton* for *pref*(t) accepts all tree patterns in prefix notation which have the root r and match the tree t.

Definition 7.5. Let t and *pref*(t) be a tree and its prefix notation, respectively. The *set srms*(t) *of subtree rightmost states* is defined as $srms = \{i : \mathit{pref}(t) = a_1 a_2 \dots a_n,\ 1 \leq i \leq n,\ \mathit{Arity}(a_i) = 0\}$.

The construction of the treetop PDA is described by the following algorithm.

Algorithm 5. Construction of a treetop PDA for a tree t in prefix notation *pref*(t).
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation $\mathit{pref}(t) = a_1 a_2 \dots a_n$, $n \geq 1$.

Output: Treetop PDA $M_{pt}(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A} \cup \{S\}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

1. Create $M_{pt}(t)$ as $M_p(t)$ by Algorithm 3.
2. Create a set $srms = \{\ i\ : 1 \leq i \leq n,\ \delta(i-1, a, S) = (i, \varepsilon),\ a \in \mathcal{A}_0\}$.
3. For each state i, where $i = n-1, n-2, \ldots, 1$, $\delta(i, a, S) = (i+1, S^p)$, $a \in \mathcal{A}_p$, create a new transition $\delta(i, S, S) = (l, \varepsilon)$ such that $(i, xy, S) \vdash^+_{M_p(t)} (l, y, \varepsilon)$ as follows:
   If $p = 0$, create a new transition $\delta(i, S, S) = (i+1, \varepsilon)$.
   Otherwise, if $p \geq 1$, create a new transition $\delta(i, S, S) = (l, \varepsilon)$, where l is the pth smallest integer such that $l \in srms$ and $l > i$. Remove all j, where $j \in srms$, and $i < j < l$, from $srms$.

The correctness of Algorithm 5 is proved in [18].

Example 7.6. Consider tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 2.1, which is illustrated in Figure 1. The deterministic treetop pushdown automaton, constructed by Algorithm 5, is deterministic pushdown automaton $M_{pt}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_4, 0, S, \emptyset))$, where its transition diagram is illustrated in Figure 16.

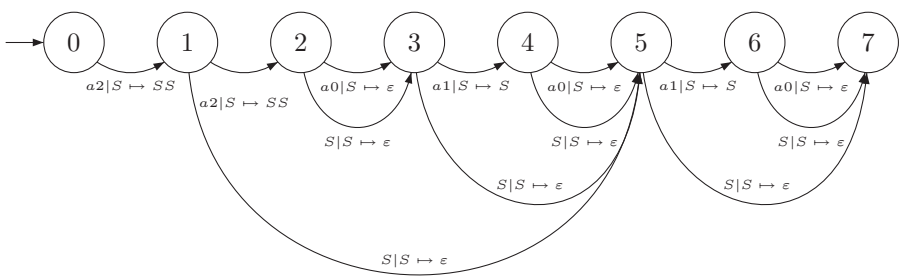


Fig. 16. Transition diagram of deterministic treetop pushdown automaton $M_{pt}(t_1)$ for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 7.6.

The nondeterministic tree pattern PDA for trees in prefix notation is constructed as an extension of the deterministic treetop PDA. The nondeterministic tree pattern PDA $M_{npt}(t)$ is input–driven and therefore can be determinised to an equivalent deterministic tree pattern PDA $M_{dpt}(t)$.

Algorithm 6. Construction of a nondeterministic tree pattern PDA for a tree t in prefix notation $pref(t)$.
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation $pref(t) = a_1 a_2 \ldots a_n$, $n \geq 1$.
Output: Nondeterministic tree pattern PDA $M_{npt}(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A} \cup \{S\}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

1. Create $M_{npt}(t)$ as the treetop PDA $M_{pt}(t)$ by Algorithm 5.

2. For each state i, where $2 \leq i \leq n$, create a new transition $\delta(0, a_i, S) = (i, S^{Arity(a_i)})$, where $S^0 = \varepsilon$.

The correctness of Algorithm 6 is proved in [18].

Example 7.7. Consider tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 2.1, which is illustrated in Figure 1. The nondeterministic tree pattern pushdown automaton accepting all tree patterns matching tree t_1, which has been constructed by Algorithm 6, is nondeterministic pushdown automaton $M_{npt}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_5, 0, S, \emptyset))$, where its transition diagram is illustrated in Figure 17.

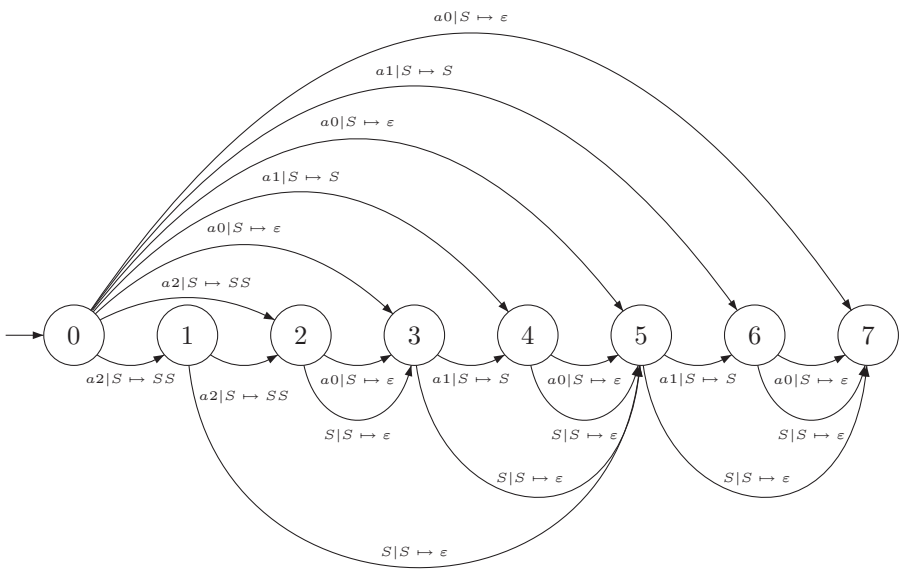


Fig. 17. Transition diagram of nondeterministic tree pattern pushdown automaton $M_{npt}(t_1)$ from Example 7.7 for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$.

The deterministic tree pattern pushdown automaton $M_{dpt}(t_1)$ constructed according to Algorithm 1 is
$M_{dpt}(t_1) = (\{[0], [1, 2], [2], [3], [4], [5], [6], [7], [3, 5, 7], [3, 5], [4, 6], [5, 7]\}, \mathcal{A}, \{S\}, \delta_6, [0], S, \emptyset))$, where its transition diagram is illustrated in Figure 18. Again, all transitions leading from states $[3, 5, 7]$ and $[5, 7]$ can be omitted because the contents of the pushdown store in these state is always ε.

Figure 19 shows the sequence of transitions (the trace) performed by deterministic tree pattern pushdown automaton $M_{dpt}(t_1)$ for input tree pattern p_1, which is illustrated in Figure 2.

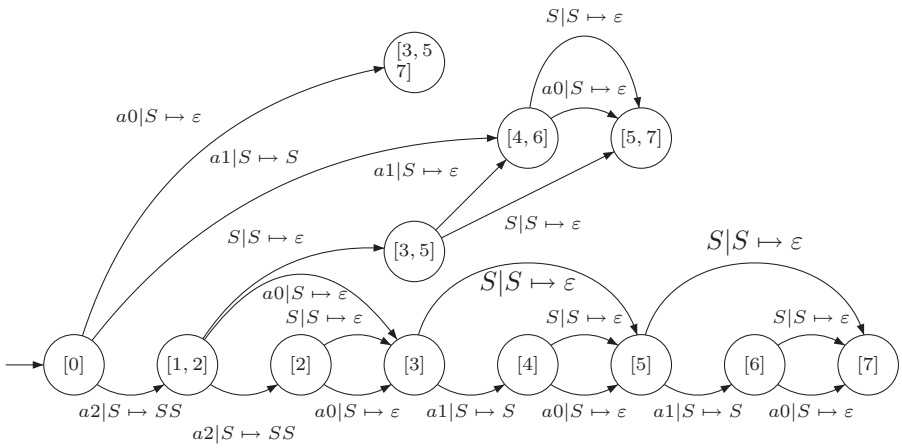


Fig. 18. Transition diagram of deterministic tree pattern pushdown automaton $M_{dpt}(t_1)$ from Example 7.7 for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$.

State	Input	Pushdown Store
$[0]$	$a2\ S\ a1\ S$	S
$[1, 2]$	$S\ a1\ S$	SS
$[3, 5]$	$a1\ S$	S
$[4, 6]$	S	S
$[5, 7]$	ε	ε
accept		

Fig. 19. Trace of deterministic pushdown automaton $M_{dpt}(t_1)$ from Example 7.7 for an input tree template p_1 from Example 2.2.

As is proved in [18], the number of distinct tree patterns which match a tree t with n nodes can be at most $2^{n-1}+n$. For specific cases of trees with an exponential number of tree patterns matching the tree, the total size of the deterministic tree pattern PDA is linear. For a detailed discussion on the total size of the deterministic tree pattern PDA, see [18].

8. PROCESSING TREES IN POSTFIX, PREFIX BAR AND POSTFIX BAR NOTATIONS

In Sections 6 and 7 particular algorithms assume ranked ordered trees in prefix notation. All these algorithms can easily be modified also for trees in the other linear notations whose basic pushdown automata are defined in Section 4. According to Theorem 1, a subtree of a tree in such a linear notation is a substring of the tree in the same linear

notation. The consequence of this theorem is that the same principles from stringology which are used for the prefix notation can also be used for the other three notations. The construction of subtree matching, subtree and tree pattern pushdown automata for trees in postfix, prefix bar and postfix bar notations differ only in pushdown operations. Instead of the pushdown operations according to the basic pushdown automaton for prefix notation defined in Definition 4.1, the pushdown operations according to the other basic pushdown automata defined by Definitions 4.2, 4.5, and 4.6 are used for processing trees in postfix, prefix bar and postfix bar notations, respectively.

9. CONCLUSION AND FUTURE WORK

The basic arbology principles and algorithms have been presented. Arbology uses a pushdown automaton reading a linear notation of a tree as its basic model of computation. The construction of arbology pushdown automata is based on the same principles ar they are used in stringology with the addition that the underlying tree structure is processed by the pushdown operations. In the basic pushdown automata presented in this paper the only one pushdown symbol S (and the initial pushdown symbol Z_0) is used, which means that the pushdown store can be replaced by an integer counter. However, this does not hold for arbology algorithms in general - more complicated algorithms, such as tree pattern matching for tree templates [11], can use more than one pushdown symbol to remember information on different kinds of subtrees. We should point out that such pushdown automata do not have to be input–driven and determinising algorithms different from those shown in Section 5 must be used.

Up-to-date information on arbology research can be found on arbology web pages [3]. Recently we have extended principles presented in this paper and developed algorithms and pushdown automata for other tree problems such as finding repeats of tree patterns in trees, tree compression, nonlinear tree pattern matching, indexing trees for nonlinear tree pattern matching, oracle versions of indexing pushdown automata for trees, and others.

Topics for future arbology research are approximate tree pattern matching which involves operations renaming, inserting, and deleting a node, tasks of XML processing, problems of determinisation of pushdown automata, finding various kinds of regularities in trees, and others.

The well developed theory of tree automata [5, 6, 10, 13] also describes many particular solutions for operations on trees. As is described in [20], every tree automaton can be transformed to an equivalent deterministic pushdown automaton which accepts the same tree language in a linear notation, and the total size of the resulting pushdown automaton is the same as the total size of the deterministic tree automaton. This agrees with the subtree matching pushdown automata presented in Section 6, the total size of which directly corresponds to the total size of the corresponding tree automata [5, 6].

ACKNOWLEDGEMENT

This research has received support from the Czech Technical University as project No. SGS12/092/OHK3/1T/18.

(Received July 6, 2011)

REFERENCES

[1] A. V. Aho and J. D. Ullman: The Theory of Parsing, Translation, and Compiling. Prentice-Hall Englewood Cliffs, N.J. 1972.

[2] R. Alur and P. Madhusudan: Visibly pushdown languages. In: STOC (L. Babai, ed.), ACM (2004), pp. 202–211.

[3] Arbology www pages. Available on: `http://www.arbology.org/` (2012).

[4] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, and J. I. Seiferas: The smallest automaton recognizing the subwords of a text. Theoret. Comput. Sci. *40* (1985), 31–55.

[5] L. Cleophas: Tree Algorithms. Two Taxonomies and a Toolkit. Ph.D. Thesis, Technische Universiteit Eindhoven 2008.

[6] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi: Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (2007).

[7] M. Crochemore: Transducers and repetitions. Theoret. Comput. Sci. *45* (1986), 1, 63–86.

[8] M. Crochemore, C. Hancart: Automata for matching patterns. In: Handbook of Formal Languages (G. Rozenberg and A. Salomaa, eds.), Vol. 2 Linear Modeling: Background and Application, Chap. 9, pp. 399–462. Springer–Verlag, Berlin 1997.

[9] M. Crochemore and W. Rytter: Jewels of Stringology. World Scientific, New Jersey 1994.

[10] J. Engelfriet: Tree Automata and Tree Grammars. University of Aarhus 1975.

[11] T. Flouri, C. Iliopoulos, J. Janoušek, B. Melichar, and S. Pissis: Tree template matching in ranked, ordered trees by pushdown automata. To appear at CIAA 2011.

[12] T. Flouri, J. Janoušek, and B. Melichar: Subtree matching by pushdown automata. Comput. Sci. Inform. System *7* (2010), 2, 331–357.

[13] F. Gecseg and M. Steinby: Tree languages. In: Handbook of Formal Languages (G. Rozenberg and A. Salomaa, eds.), Vol. 3 Beyond Words. Handbook of Formal Languages, pp. 1–68. Springer–Verlag, Berlin 1997.

[14] R. S. Glanville and S. L. Graham: A new method for compiler code generation. In: POPL 1978, pp. 231–240.

[15] C. M. Hoffmann and M. J. O'Donnell: Pattern matching in trees. J. Assoc. Comput. Mach. *29* (1982), 1, 68–95.

[16] J. E. Hopcroft, R. Motwani, and J. D. Ullman: Introduction to Automata Theory, Languages, and Computation. Second edition. Addison-Wesley, Boston 2001.

[17] J. Janoušek: String suffix automata and subtree pushdown automata. In: Proc. Prague Stringology Conference 2009 (J. Holub and J. Žďárek, eds.), Czech Technical University in Prague 2009, pp. 160–172. Available on: `http://www.stringology.org/event/2009`.

[18] J. Janoušek: Arbology: Algorithms on Trees and Pushdown Automata. Habilitation Thesis, TU FIT, Brno 2010.

[19] J. Janoušek: Introduction to arbology. An invited talk at AAMP WIII conference, Prague 2011.

[20] J. Janoušek and B. Melichar: On regular tree languages and deterministic pushdown automata. Acta Inform. *46* (2009), 7, 533–547.

[21] M. Madhavan, P. Shankar, S. Rai, and U. Ramakrishna: Extending graham-glanville techniques for optimal code generation. ACM Trans. Program. Lang. Syst. *22* (2000), 6, 973–1001.

[22] B. Melichar: Arbology: Trees and pushdown automata. In: LATA 2010 (A. H. Dediu, H. Fernau, and C. Martín-Vide, eds.), Lecture Notes in Comput. Sci. *6031* (2010), pp. 32–49. Invited paper.

[23] B. Melichar, J. Holub, and J. Polcar: Text searching algorithms. Available on: `http://stringology.org/athens/` (2005).

[24] D. Nowotka and J. Srba: Height-deterministic pushdown automata. In: MFCS 2007 (L. Kucera and A. Kucera, eds.), Lecture Notes in Comput. Sci. *4708* (2007), pp. 125–134.

[25] P. Shankar, A. Gantait, A. R. Yuvaraj, and M. Madhavan: A new algorithm for linear regular tree pattern matching. Theor. Comput. Sci. *242* (2000), 1–2, 125–142.

[26] B. Smyth: Computing Patterns in Strings. Addison-Wesley-Pearson Education Limited, Essex 2003.

Bořivoj Melichar, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Thákurova 9, 160 00 Praha 6. Czech Republic.
e-mail: Borivoj.Melichar@fit.cvut.cz

Jan Janoušek, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Thákurova 9, 160 00 Praha 6. Czech Republic.
e-mail: Jan.Janousek@fit.cvut.cz

Tomas Flouri, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Thákurova 9, 160 00 Praha 6. Czech Republic.
e-mail: Tomas.Flouri@fit.cvut.cz