String Suffix Automata and Subtree Pushdown Automata*

Jan Janoušek

Department of Computer Science
Faculty of Information Technologies
Czech Technical University in Prague
Zikova 1905/4, 166 36 Prague 6, Czech Republic
Jan.Janousek@fit.cvut.cz

Abstract. String suffix automata accept all suffixes of a given string and belong to the fundamental stringology principles. Extending their transitions by specific pushdown operations results in new subtree pushdown automata, which accept all subtrees of a given subject tree in prefix notation and are analogous to the suffix automata in their properties. The deterministic subtree pushdown automaton accepts an input subtree in time linear to the number of nodes of the subtree and its total size is linear to the number of nodes of the given subject tree.

Keywords: tree, subtree, string suffix automata, tree pattern matching, pushdown automata

1 Introduction

The theory of formal string (or word) languages [1,10,17] and the theory of formal tree languages [4,5,9] are important parts of the theory of formal languages [16]. The most famous models of computation of the theory of tree languages are various kinds of tree automata [4,5,9]. Trees can also be seen as strings, for example in their prefix (also called preorder) or postfix (also called postorder) notation. [11] shows that the deterministic pushdown automaton (PDA) is an appropriate model of computation for labelled ordered ranked trees in postfix notation and that the trees in postfix notation acceptable by deterministic PDA form a proper superclass of the class of regular tree languages, which are accepted by finite tree automata. In the further text we will omit word "string" when referencing to string languages or string 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 text, which means that the text is preprocessed. Examples of these automata are suffix automata [6]. Given a text of size n, the suffix automaton can be constructed for the text in time linear in n. The constructed suffix automaton represents a complete index of the text for all possible suffixes and can find all occurrences of a string suffix and their positions in the text. The main advantage of this kind of finite automata is that the deterministic suffix automaton performs the search phase in time linear in the size of the input subtree and not depending on n.

This paper presents a new kind of acyclic PDAs for trees in prefix notation, which is analogous to string suffix automata and their properties: *subtree PDAs* accept all

* 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.

Jan Janoušek: String Suffix Automata and Subtree Pushdown Automata, pp. 160–172.
Proceedings of PSC 2009, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04403-2

subtrees of the tree. The basic idea of the subtree PDAs has been presented in [13]. This paper deals with the subtree PDAs in more details. [12] contains the detailed description of the subtree PDAs, related formal theorems, lemmas, and their proofs, many of which are skipped in this paper. Moreover, [12] describes an extension of the subtree PDAs – *tree pattern PDAs*, which accept all tree patterns matching the tree and are analogous to string factor automata in their basic properties.

By analogy with the string suffix automaton, the subtree PDA represents a complete index of the tree for all possible subtrees. Given a tree of size n, the main advantage of the deterministic subtree PDA is again that the search phase is performed in time linear in the size of the input subtree and not depending on n. We note that this cannot be achieved by any standard tree automaton because the standard deterministic tree automaton runs always on the subject tree, which means the searching by tree automata can be linear in n at the best.

Moreover, the presented subtree PDAs have the following two other properties. First, they are input-driven PDAs [20], which means that each pushdown operation is determined only by the input symbol. Input-driven PDAs can always be determinised [20]. 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 [3,19], 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 2. Some properties of subtrees in prefix notation are discussed in the third section. The fourth section deals with the subtree PDA. The last section is the conclusion.

2 Basic notions

2.1 Ranked alphabet, tree, prefix notation

We define notions on trees similarly as they are defined in [1,4,5,9].

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 $Arity(a)$. 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.

Based on concepts from 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 $((f, g_1), (f, g_2), \ldots, (f, g_n))$, where $f, g_1, g_2, \ldots, g_n \in N$, $n \geq 0$. 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 $(f_0, f_1, \ldots, f_n)$, $n \geq 1$, 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 $G = (A, R)$ 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, the *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 $Arity(a)$.

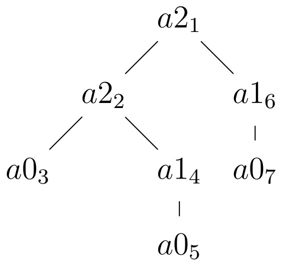
Nodes labelled by nullary symbols (constants) are called *leaves*.

Prefix notation 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* be to 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 $a_{f_1}, a_{f_2}, \ldots, a_{f_n}$. Then list a and subsequently apply *Step* to $a_{f_1}, a_{f_2}, \ldots, a_{f_n}$ in that order.

Example 1. Consider a ranked alphabet $\mathcal{A} = \{a2, a1, a0\}$. Consider a tree t_1 over $\mathcal{A}$ $t_1 = (\{a2_1, a2_2, a0_3, a1_4, a0_5, a1_6, a0_7\}, R)$, where R is a set of the following ordered sequences of pairs:

$$\begin{array}{l}((a2_1, a2_2), (a2_1, a1_6)),\\ ((a2_2, a0_3), (a2_2, a1_4)),\\ ((a1_4, a0_5)),\\ ((a1_6, a0_7))\end{array}$$

Tree t_1 in prefix notation is string $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$. Trees can be represented graphically and tree t_1 is illustrated in Fig. 1. □



$pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

Figure 1. Tree t_1 from Example 1 and its prefix notation

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.

2.2 Alphabet, language, pushdown automaton

We define notions from the theory of string languages similarly as they are defined in [1,10].

Let an *alphabet* be a finite nonempty set of symbols. 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 [1,10]. 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.* In spite of the fact that d-subsets are standard sets, they are often written in square brackets ([ ]) instead of in braces ({ }).

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*, 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 symbol, 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 right 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 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 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 pushdown automaton is *input-driven* if each of its pushdown operations 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:*

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

2. *Accepting by empty pushdown store:*

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

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

For more details see [1,10].

2.3 Example of string suffix automaton

Example 2. Given the prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ of tree t_1 from Example 1, the corresponding nondeterministic suffix automaton is $FM_{nsuf}(pref(t_1)) = (\{0,1,2,3,4,5,6,7\}, \mathcal{A}, \delta_n, 0, \{7\}))$, where its transition diagram is illustrated in Fig. 2. (For the construction of the nondeterministic suffix automaton see [14].)

After the standard transformation of a nondeterministic suffix automaton to a deterministic one [10], the deterministic suffix automaton for $pref(t_1)$ is $FM_{dsuf}(pref(t_1)) = (\{[0],[1,2],[2],[3],[4],[5],[6],[7],[3,5,7],[4,6],[5,7]\}, \mathcal{A}, \delta_d, 0, \{[7],[3,5,7],[5,7]\}))$, where its transition diagram is illustrated in Fig. 3.

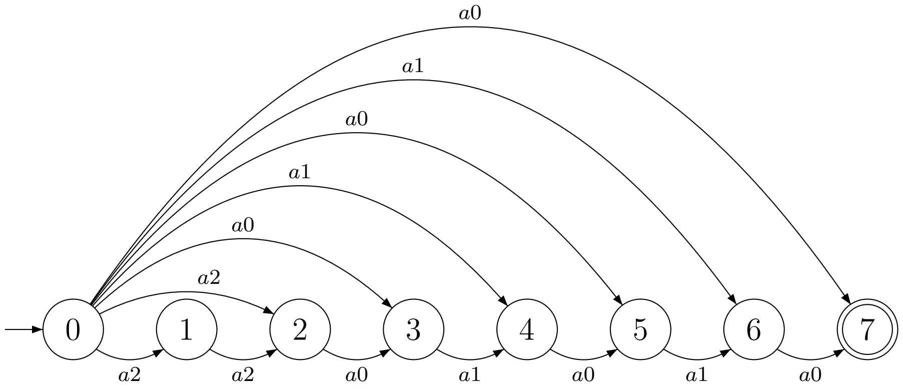


Figure 2. Transition diagram of nondeterministic suffix automaton for string $a2\ a2\ a0\ a1\ a0\ a1\ a0$

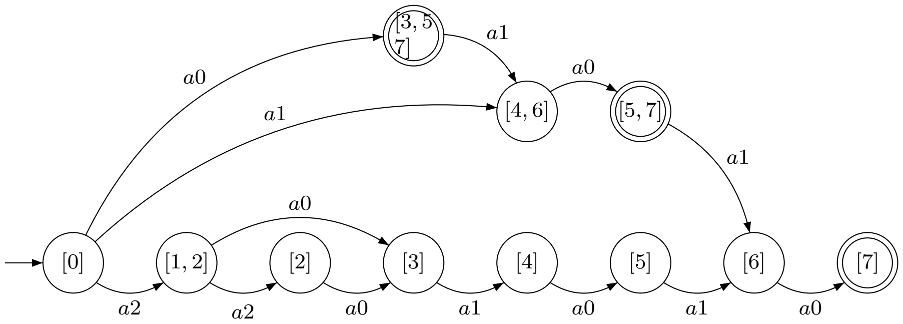


Figure 3. Transition diagram of deterministic suffix automaton for string $a2\ a2\ a0\ a1\ a0\ a1\ a0$

3 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 subtree PDA, which is described in the next section.

Example 3. Consider tree t_1 in prefix notation $\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1, which is illustrated in Fig. 1. Tree t_1 contains only subtrees shown in Fig. 4.

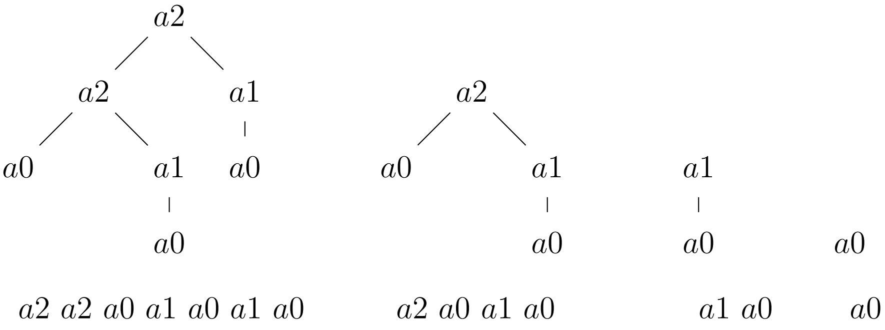


Figure 4. All subtrees of tree t_1 from Example 1, and their prefix notations

Generally, it holds for any tree that each of its subtrees in prefix notation is a substring of the tree in prefix notation.

Theorem 4. *Given a tree t and its prefix notation pref(t), all subtrees of t in prefix notation are substrings of pref(t).*

Proof. In [12]. □

However, not every substring of a tree in prefix notation is a prefix notation of its subtree. This can be easily seen from the fact that for a given tree with n nodes there can be $\mathcal{O}(n^2)$ distinct substrings, but there are 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 is formalised by the following definition and theorem.

Definition 5. *Let* $w = a_1 a_2 \cdots a_m$*,* $m \geq 1$*, be a string over a ranked alphabet* $\mathcal{A}$*. Then, the* arity checksum $ac(w) = \mathit{arity}(a_1) + \mathit{arity}(a_2) + \cdots + \mathit{arity}(a_m) - m + 1 = \sum_{i=1}^{m} \mathit{arity}(a_i) - m + 1$.

Theorem 6. *Let* $\mathit{pref}(t)$ *and* w *be a tree* t *in prefix notation and a substring of* $\mathit{pref}(t)$*, 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* $w = w_1 x$*,* $x \neq \varepsilon$*.*

Proof. In [12]. □

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

4 Subtree pushdown automaton

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

Definition 7. *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. The construction of the PDA accepting a tree in prefix notation by the empty pushdown store 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 $pref(t) = a_1 a_2 \cdots a_n$, $n \geq 1$.
Output: PDA $M_p(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$.
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 $S^0 = \varepsilon$. □

Example 8. A PDA accepting tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1, which has been constructed by Alg. 1, is 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 δ_1 is a set of the following transitions:

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

The transition diagram of deterministic PDA $M_p(t_1)$ is illustrated in Fig. 5. In this figure for each transition rule $\delta_1(p, a, \alpha) = (q, \beta)$ from δ the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$.

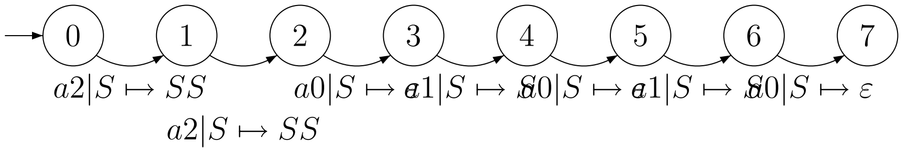


Figure 5. Transition diagram of deterministic PDA $M_p(t_1)$ accepting tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 8

Fig. 6 shows the sequence of transitions (trace) performed by deterministic PDA $M_p(t_1)$ for tree t_1 in prefix notation. □

It holds that every input-driven PDA that has the same pushdown operations as they are defined for the above deterministic PDA $M_p(t)$ for tree t in prefix notation behaves such that the contents of its pushdown store corresponds to the arity checksum. This is described by the following theorem. We note that such pushdown operations correspond to the pushdown operations of the standard top-down parsing algorithm for a context-free grammar with rules of the form

$$S \to a\ S^{arity(a)}.$$

For principles of the standard top–down (LL) parsing algorithm see [1].

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	ε	ε
accept		

Figure 6. Trace of deterministic PDA $M_p(t_1)$ from Example 8 for tree t_1 in prefix notation $\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

Theorem 9. *Let* $M = (\{Q, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$ *be an input-driven PDA of which each transition from* δ *is of the form* $\delta(q_1, a, S) = (q_2, S^i)$*, where* $i = \mathit{arity}(a)$*. Then, if* $(q_3, w, S) \vdash_M^+ (q_4, \varepsilon, S^j)$*, then* $j = \mathit{ac}(w)$*.*

Proof. In [12]. □

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

Lemma 10. *Given a tree t and its prefix notation pref*(t)*, the PDA* $M_p(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$*, where* $n \geq 0$*, constructed by Alg. 1 accepts pref*(t)*.*

Proof. In [12]. □

We present the construction of the deterministic subtree PDA for trees in prefix notation. The construction consists of two steps. First, a nondeterministic subtree PDA is constructed by Alg. 2. This nondeterministic subtree PDA is an extension of the PDA accepting tree in prefix notation, which is constructed by Alg. 1. Second, the constructed nondeterministic subtree PDA is transformed to the equivalent deterministic subtree PDA. Although a nondeterministic PDA cannot generally be determinised, the constructed nondeterministic subtree PDA is an input-driven PDA and therefore can be determinised [20].

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

1. Create PDA $M_{nps}(t)$ as PDA $M_p(t)$ by Alg. 1.
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 $S^0 = \varepsilon$. □

Example 11. A subtree PDA for tree t_1 in prefix notation
$\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1, which has been constructed by Alg. 2, is nondeterministic PDA $M_{nps}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_2, 0, S, \emptyset))$, where mapping δ_2 is a set of the following transitions:

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

The transition diagram of nondeterministic PDA $M_{nps}(t_1)$ is illustrated in Fig. 7. Again, in this figure for each transition rule $\delta_2(p, a, \alpha) = (q, \beta)$ from δ_2 the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$.

A comparison of Figs. 7 and 2 shows that the states and the transitions of nondeterministic subtree PDA $M_{nps}(t_1)$ correspond to the states and the transitions, respectively, of the nondeterministic string suffix automaton for *pref*(t_1); the transitions of the subtree PDA are extended by pushdown operations so that it holds that the number of symbols S in the pushdown store is equal to the corresponding arity checksum. □

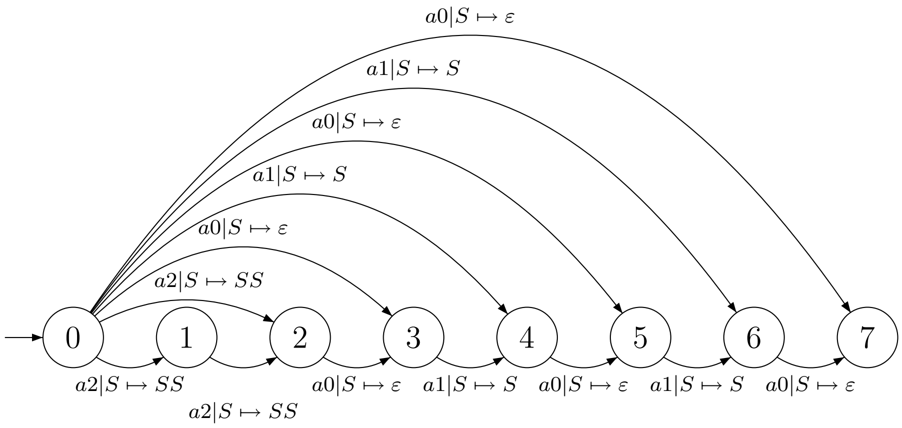

$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a2|S \mapsto SS$
0
1
2
3
4
5
6
7
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a2|S \mapsto SS$

Figure 7. Transition diagram of nondeterministic subtree PDA $M_{nps}(t_1)$ for tree t_1 in prefix notation *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 11

Theorem 12. *Given a tree t and its prefix notation pref(t), the PDA $M_{nps}(t)$ constructed by Alg. 2 is a subtree PDA for pref(t).*

Proof. In [12]. □

It is known that each nondeterministic input-driven PDA can be transformed to an equivalent deterministic input-driven PDA [20]. To construct deterministic subtree or tree pattern PDAs from their nondeterministic versions we use the transformation described by Alg. 3. This transformation is a simple extension of the well known transformation of a nondeterministic finite automaton to an equivalent deterministic one [10]. Again, the states of the resulting deterministic PDA correspond to subsets of the states of the original nondeterministic PDA, and these subsets are again called d-subsets. Moreover, the original nondeterministic PDA is assumed to be acyclic with a specific order of states, and Alg. 3 precomputes the possible contents of the pushdown store in particular states of the deterministic PDA according to pushdown operations and selects only those transitions and accessible states of the deterministic PDA for which the pushdown operations are possible. The assumption that the PDA is acyclic results in a finite number of possible contents of the pushdown store. Furthermore, the assumption of the specific order of states allows us to compute these contents of the pushdown store easily in a one-pass way.

Algorithm 3. Transformation of an input-driven nondeterministic PDA to an equivalent deterministic PDA.
Input: Acyclic input-driven nondeterministic PDA $M_{nx}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$, where the ordering of its states is such that if $\delta(p, a, \alpha) = (q, \beta)$, then $p < q$.
Output: Equivalent deterministic PDA $M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', q_I, S, \emptyset)$.
Method:

1. Let $cpds(q')$, where $q' \in Q'$, denote a set of strings over $\{S\}$. (The abbreviation *cpds* stands for Contents of the PushDown Store.)
2. Initially, $Q' = \{[0]\}$, $q_I = [0]$, $cpds([0]) = \{S\}$ and $[0]$ is an unmarked state.
3. (a) Select an unmarked state q' from Q' such that q' contains the smallest possible state $q \in Q$, where $0 \leq q \leq n$.
   (b) For each input symbol $a \in \mathcal{A}$:
      i. 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' and create $cpds(q'') = \emptyset$. Add ω, where $\delta(q', a, \gamma) \vdash_{M_{dx}(t)} (q'', \varepsilon, \omega)$ and $\gamma \in cpds(q')$, to $cpds(q'')$.
   (c) Set the state q' as marked.
4. Repeat step 3 until all states in Q' are marked. □

The deterministic subtree automaton for a tree in prefix notation is demonstrated by the following example. The PDA reads an input subtree in prefix notation and the accepting state corresponds to the rightmost leaves of all occurrences of the input subtree in the subject tree.

Example 13. The deterministic subtree PDA for tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1, which has been constructed by Alg. 3 from nondeterministic subtree PDA $M_{nps}(t_1)$ from Example 11, is deterministic PDA $M_{dps}(t_1) = (\{[0], [1, 2], [2], [3], [4], [5], [6], [7], [3, 5, 7], [4, 6], [5, 7]\}, \mathcal{A}, \{S\}, \delta_3, \ [0], S, \emptyset))$, where mapping δ_3 is a set of the following transitions:

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

We note that there are no transitions leading from states $[3, 5, 7]$, $[5, 7]$ and $[7]$, because the pushdown store in these state is always empty and therefore no transition is possible from these states due to the pushdown operations. This means that the deterministic subtree PDA $M_{dps}(t_1)$ has fewer transitions than the deterministic string suffix automaton constructed for $pref(t_1)$ [6,14,18], as can be seen by comparing Figs. 3 and 8.

The transition diagram of deterministic PDA $M_{dps}(t_1)$ is illustrated in Fig. 8. Again, in this figure for each transition rule $\delta_3(p, a, \alpha) = (q, \beta)$ from δ_3 the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$.

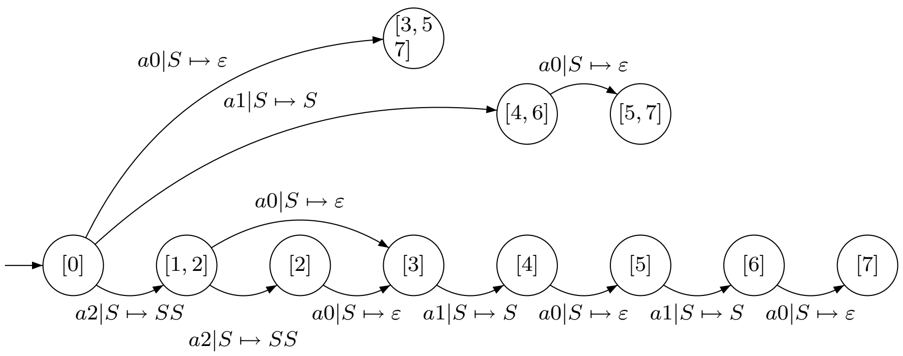

[0]
[1, 2]
[2]
[3]
[4]
[5]
[6]
[7]
[3, 5 7]
[4, 6]
[5, 7]
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$

Figure 8. Transition diagram of deterministic subtree PDA $M_{dps}(t_1)$ for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 13

Fig. 9 shows the sequence of transitions (trace) performed by deterministic subtree PDA $M_{dps}(t_1)$ for an input subtree st in prefix notation $pref(st) = a_1a_0$. 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 $a0_7$. □

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

Figure 9. Trace of deterministic subtree PDA $M_{dps}(t_1)$ from Example 13 for an input subtree st in prefix notation $pref(st) = a1a0$

Theorem 14. *Given an acyclic input-driven nondeterministic PDA* $M_{nx}(t) = (Q, \mathcal{A}, \{S\}, \delta, q_0, S, \emptyset)$*, the deterministic PDA* $M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', \{q_0\}, S, \emptyset)$ *constructed by Alg. 3 is equivalent to PDA* $M_{nx}(t)$*.*

Proof. In [12]. □

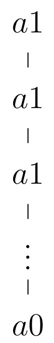
We note that trees with the structure $pref(t) = (a1)^{n-1}a0$ represent strings. Such a tree is illustrated in Fig. 10. It can be simply shown that the deterministic subtree PDAs for such trees have the same number of states and transitions as the deterministic suffix automata constructed for $pref(t)$ and accept the same language.

It is obvious that the number of distinct subtrees in a tree can be at most the number of nodes of the tree.

Lemma 15. *Given a tree t with n nodes, the number of distinct subtrees of tree t is equal or smaller than n.*

Proof. In [12]. □

At the end of this section we discuss the total size of the constructed deterministic subtree PDA, which cannot be greater than the total size of the deterministic suffix automaton constructed for $pref(t)$ [6,7]. We recall that the deterministic subtree PDA can have even fewer states and transitions than the corresponding deterministic string suffix automaton as certain states and transitions need not be accessible due to pushdown operations.



$$pref(t_2) = (a1)^{n-1}a0$$

Figure 10. A tree t_2, which represents a string, and its prefix notation

Theorem 16. *Given a tree t with n nodes and its prefix notation $pref(t)$, the deterministic subtree PDA $M_{dps}(t)$ constructed by Algs. 2 and 3 has just one pushdown symbol, fewer than $N \leq 2n + 1$ states and at most $N + n - 1 \leq 3n$ transitions.*

Proof. The deterministic subtree PDA in question may have only states and transitions which correspond to the states and the transitions, respectively, of the deterministic suffix automaton constructed for $pref(t)$. Therefore, the largest possible numbers of states and transitions of the deterministic subtree PDA are the same as those of the deterministic suffix automaton. The numbers of states and transitions of the deterministic suffix automaton are proved in Theorems 6.1 and 6.2 in [7] or in Theorem 5.3.5 in [18]. We note that these proofs are based on the following principle: Given a substring u, the d-subset of the state in which the deterministic suffix automaton is after reading u is called the *terminator set* of u [18]. It holds for any two substrings u_1 and u_2 that their terminator sets cannot overlap; in other words, the terminator sets of a deterministic suffix automaton correspond to a tree structure. It has been proved that this tree structure is such that the above-mentioned numbers of states and transitions hold. □

5 Conclusion

We have described a new kind of pushdown automata: subtree PDAs for trees in prefix notation. These pushdown automata are in their properties analogous to suffix automata, which are widely used in stringology. The presented subtree PDAs represent a complete index of the subject tree with n nodes for all possible subtrees and the deterministic version allows to find all occurrences of input subtrees of size m in time linear in m and not depending on n.

Regarding specific tree algorithms whose model of computation is the standard deterministic pushdown automaton, recently we have introduced principles of other three new algorithms. First, a new and simple method for constructing subtree pattern matchers as deterministic pushdown automata directly from given subtrees without constructing finite tree automata as an intermediate product [8,13]. Second, tree pattern pushdown automata, which represent a complete index of the tree for all tree patterns matching the tree and the search phase of all occurrences of a tree pattern of size m is performed in time linear in m and not depending on the size of the tree [12,13]. These automata representing indexes of trees for all tree patterns are analogous in their properties to the string factor automata [6,7] and are an extension of the subtree PDA presented in this paper. Third, a method for finding all repeats of connected subgraphs in trees with the use of subtree or tree pattern PDA [15,13]. More details on these results and related information can also be found on [2].

I would like to thank to Bořivoj Melichar and anonymous referees – their comments have contributed to improving the text significantly.

References

1. A. V. Aho and J. D. Ullman: *The theory of parsing, translation, and compiling*, Prentice-Hall Englewood Cliffs, N.J.,, 1972.
2. *Arbology www pages*: Available on: `http://www.arbology.org`, July 2009.
3. J. Berstel: *Transductions and Context-Free Languages*, Teubner Studienbucher, Stuttgart, 1979.
4. L. Cleophas: *Tree Algorithms. Two Taxonomies and a Toolkit.*, PhD thesis, Technische Universiteit Eindhoven, Eindhoven, 2008.
5. 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, release October, 12th 2007.
6. M. Crochemore and C. Hancart: *Automata for matching patterns*, in Handbook of Formal Languages, G. Rozenberg and A. Salomaa, eds., vol. 2 Linear Modeling: Background and Application, Springer-Verlag, Berlin, 1997, ch. 9, pp. 399–462.
7. M. Crochemore and W. Rytter: *Jewels of Stringology*, World Scientific, New Jersey, 1994.
8. T. Flouri, J. Janoušek, and B. Melichar: *Tree pattern matching by deterministic pushdown automata.* accepted for WAPL 2009 conference, 2009.
9. 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, Springer-Verlag, Berlin, 1997, pp. 1–68.
10. J. E. Hopcroft, R. Motwani, and J. D. Ullman: *Introduction to automata theory, languages, and computation*, Addison-Wesley, Boston, 2nd ed., 2001.
11. J. Janoušek and B. Melichar: *On regular tree languages and deterministic pushdown automata.* accepted for publication in Acta Informatica, Springer, 2009.
12. J. Janoušek and B. Melichar: *Subtree and tree pattern pushdown automata for trees in prefix notation.* submitted for publication, 2009.
13. *London stringology days 2009 conference presentations*: Available on: `http://www.dcs.kcl.ac.uk/events/LSD&LAW09/`, King's College London, London, February 2009.
14. B. Melichar, J. Holub, and T. Polcar: *Text searching algorithms.* Available on: `http://stringology.org/athens/`, 2005, release November 2005.
15. B. Melichar and J. Janoušek: *Repeats in trees by subtree and tree pattern pushdown automata.* draft, 2009.
16. G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.
17. G. Rozenberg and A. Salomaa, eds., *Vol. 1: Word, Language, Grammar, Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.
18. B. Smyth: *Computing Patterns in Strings*, Addison-Wesley-Pearson Education Limited, Essex, England, 2003.
19. L. G. Valiant and M. Paterson: *Deterministic one-counter automata*, in Automaten theorie und Formale Sprachen, 1973, pp. 104–115.
20. K. Wagner and G. Wechsung: *Computational Complexity*, Springer-Verlag, Berlin, 2001.