DOI: 10.2298/CSIS111220024T

Indexing Ordered Trees for (Nonlinear) Tree Pattern Matching by Pushdown Automata

Jan Trávníček, Jan Janoušek, and Borivoj Melichar

Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00 Prague 6, Czech Republic
{Jan.Travnicek, Jan.Janousek, melichar}@fit.cvut.cz

Abstract. Trees are one of the fundamental data structures used in Computer Science. We present a new kind of acyclic pushdown automata, the *tree pattern pushdown automaton* and the *nonlinear tree pattern pushdown automaton*, constructed for an ordered tree. These automata accept all tree patterns and nonlinear tree patterns, respectively, which match the tree and represent a full index of the tree for such patterns. Given a tree with n nodes, the numbers of these distinct tree patterns and nonlinear tree patterns can be at most $2^{n-1}+n$ and at most $(2+v)^{n-1}+2$, respectively, where v is the maximal number of nonlinear variables allowed in nonlinear tree patterns. The total sizes of nondeterministic versions of the two pushdown automata are $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$, respectively. We discuss the time complexities and show timings of our implementations using the bit-parallelism technique. The timings show that for a given tree the running time is linear to the size of the input pattern.

Keywords: Tree pattern matching, nonlinear tree pattern matching, indexing trees, pushdown automata

1. Introduction

Trees are one of the fundamental data structures used in Computer Science. Finding occurrences of tree patterns in trees is an important problem with many applications such as compiler code selection, interpretation of nonprocedural languages, implementation of rewriting systems, or various tree finding and tree replacement systems. Tree patterns are trees in which leaves can be labelled also by a special linear variable S, which serves as a placeholder for any subtree. Nonlinear tree patterns can further contain leaves labelled by specific nonlinear variables, where each of nonlinear variables represents a specific subtree. Nonlinear tree pattern matching is used especially in the implementation of term rewriting systems, in which the terms can be represented as tree structures with nonlinear variables.

Generally, there exist two basic approaches to pattern matching problems. The first approach is represented by the use of a pattern matcher which is constructed for patterns. In other words, the patterns are preprocessed. Given a

tree of size n, such tree pattern matcher typically perform the search phase in time linear in n [15, 6, 12, 24]. This approach is suitable for cases when one wants look for occurrences of a given pattern in input subject structures. The second basic approach is represented by the use of an indexing data structure constructed for the subject in which we search. In other words, the subject is preprocessed. Examples of such indexing structures are suffix or factor automata [9, 10, 22, 25] in the area of string processing or subtree pushdown automaton [18], which represents a complete index of an ordered tree for subtrees. This approach is suitable especially for cases when one wants look for occurrences of different input patterns in a given subject structure.

The theory of formal tree languages have been extensively studied and developed since the 1960s and its main models of computation are various kinds of tree automata [14, 6, 8]. However, trees can also be represented as strings, for example in their prefix (also called preorder) or postfix (also called postorder) notation. A linear notation of a tree can be obtained by the corresponding traversing of the tree. Moreover, every sequential algorithm on a tree traverses nodes of the tree in a sequential order and so follows a linear notation of the tree. [20] proves that the deterministic pushdown automaton (PDA) is an appropriate model of computation for labelled ordered trees in linear notation and that the trees in postfix notation acceptable by deterministic PDA form a proper superclass of the class of regular tree languages [14], which are accepted by finite tree automata. Recently, pushdown automata gain a popularity in solving practical problems of processing trees, for example in processing XML documents [13].

In this paper we present a new kind of acyclic pushdown automata for an ordered tree. The *tree pattern pushdown automaton* and the *nonlinear tree pattern pushdown automaton* represent a complete index of the tree for tree patterns and nonlinear tree patterns, respectively, and accept all tree patterns and nonlinear tree patterns, respectively, which match the tree. Given a tree with n nodes, the numbers of distinct tree patterns and nonlinear tree patterns which match the tree can be at most $2^{n-1}+n$ and at most $(2+v)^{n-1}+2$, respectively, where v is the maximal number of distinct nonlinear variables allowed in nonlinear tree patterns. We describe the construction of nondeterministic (nonlinear) tree pattern pushdown automata and discuss their time and space complexities. We are not aware of any other existing pushdown automaton which would represent such an index. The presented nondeterministic pushdown automata are input–driven and therefore can be determinised.

The presented (nonlinear) tree pattern pushdown automata have two kinds of transitions. First, transitions reading symbols of the alphabet of labels of tree nodes. Second, transitions reading the variables in (nonlinear) tree patterns. If our pushdown automata have only the former transitions, they would be analogous to string nondeterministic suffix or factor automata. Efficient methods of implementing nondeterministic string suffix automata by the bit–parallelism technique are well-known [23]. The bit–parallelism technique can be also used for the latter transitions efficiently. We describe our implementations using the

bit-parallel technique and show their timings. The timings show that the running time is for a given tree linear to the size of the input (nonlinear) tree patterns, which is a result similar to the result described in [23].

We note that the presented PDAs have only one pushdown symbol and therefore can be easily transformed to counter automata, which are a weaker and simpler model of computation than the PDA. We present the automata in this paper as PDAs because the PDA is a more fundamental and more widely-used model of computation than the counter automaton.

Since our pushdown automata accept finite languages, which correspond to finite sets of various connected subgraphs of the tree, a finite automaton could also be used instead of a pushdown automaton. However, such finite automaton would have significantly more states than the PDA, in which the underlying tree structure is efficiently processed by the pushdown store.

Early presentations of the tree pattern pushdown automaton and the nonlinear tree pattern pushdown automaton can be found in [21, 19] and [26], respectively. This paper can be considered as an extended version of these publications.

The paper is organised as follows. The second section discusses related works of existing (nonlinear) tree pattern matching algorithms. The third section contains basic definitions. The fourth section is devoted to indexing trees for tree pattern matching by pushdown automata. The fifth section describes indexing trees for nonlinear tree pattern matching by pushdown automata. Section 6 deals with nonlinear tree pattern matching for more than one nonlinear variable in nonlinear tree patterns. The seventh section describes our implementations and show experimental results. The last section is a conclusion.

2. Related Works

Some algorithms for (nonlinear) tree pattern matching are known. All of them use the approach which is represented by the preprocessing of the (nonlinear) tree pattern. For tree pattern matching algorithms see [15, 6, 12].

Nonlinear tree pattern matching algorithm described in [24] reads the Euler linear notation of both a subject tree and a nonlinear tree pattern. Euler notation is a tree linear notation, which contains a node each time it is visited during the preorder traversing of the tree. This means that every node appears exactly $1 + arity(node)$-times in Euler notation. Our method presented in this paper uses a standard tree prefix notation, which contains every node just once.

In [24] factors which represent some subtrees in a subject tree in Euler notation are constructed. The standard Aho-Corasick automaton [1] is then constructed for these factors. The subject tree in Euler notation is processed by the constructed Aho-Corasick automaton and a binary array is constructed for each factor of the nonlinear tree pattern. Locations of factors of the input subject tree in Euler notation are then transformed to arrays of ones and zeros, which describes locations of this factor in the subject tree in Euler notation. In this way the nonlinear variables are matched.

3. Basic notions

We define notions on trees similarly as they are defined in [2, 14, 15].

3.1. 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 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 identifiers for a short declaration of symbols with arity. For instance, $a2$ is a short declaration of a binary symbol a.

3.2. Tree, tree pattern, nonlinear tree pattern

Based on concepts from graph theory (see [2]), a tree over an alphabet $\mathcal{A}$ can be defined as follows:

An *graph* G is a pair (N, R), where N is a set of nodes and R is a set of edges such that each element of R is of the form (f, g), where $f, g \in N$. This element will indicate that, for node f, there is an edge between node f and node g.

A *directed graph* G is a graph, where each element of R of the form (f, g) indicates that, there is an edge leaving node f and entering node g. This edge is ordered from f to g. An *undirected graph* G is a graph in which no such ordering of edges is given.

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 *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.

A directed graph is *connected* if there exists a path from f_u to f_v for each pair of nodes (f_u, f_v), $u \neq v$, of the graph.

A *cycle* is a path $(f_0, f_1, \ldots, f_n)$ in which $f_0 = f_n$.

Given a node f of a directed graph, 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 a connected graph without any cycle. In this paper we assume that the tree has at least one node. Any node of a tree can be selected as a *root* of the tree. A tree with a root is called *rooted tree*.

A tree can be *directed*. A *rooted and directed tree* t is an acyclic connected directed graph $t = (N, R)$ 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 $f \neq r$.

Nodes of a directed tree with out-degree 0 are called *leaves*.

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.

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

An *ordered, (ranked, 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.

Example 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 pairs:

$$R_1 = \{(a2_1, a2_2), (a2_1, a1_6), (a2_2, a0_3), (a2_2, a1_4), (a1_4, a0_5), (a1_6, a0_7)\}.$$

Tree t_1 in prefix notation is $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$.

Trees can be represented graphically, and tree t_1 is illustrated in Figure 1.

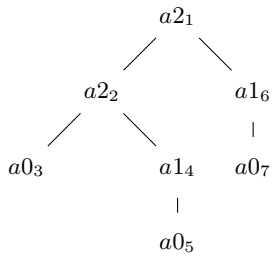

Fig. 1. Tree t_1 from Example 1

A *subtree* of a tree $t = (N, R)$ is any tree $t' = (N', R')$ such that: (1) N' is nonempty and contained in N, (2) $R' = A' \times A' \cap R$, and (3) No node of $A \setminus A'$ is a descendant of a node in A'.

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

To define a *tree pattern*, we use a special nullary symbol S, not in alphabet $\mathcal{A}$, $Arity(S) = 0$, which is a variable and serves as a placeholder for any 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* a subject 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 i-th occurrence of S in p, $i = 1, 2, \ldots, k$, is equal to the subtree of t rooted at n.

The *nonlinear tree pattern* can contain also other special nullary symbols – nonlinear variables, not in alphabet $\mathcal{A}$. These symbols serve as placeholders for

specific subtrees. Every occurrence of a symbol X in a nonlinear tree pattern is matched with the same subtree. A nonlinear tree pattern has to contain at least one symbol from $\mathcal{A}$. A nonlinear tree pattern which contains at least two equal nonlinear variables will be called a *nonlinear tree template*.

A nonlinear tree pattern np with $k \geq 2$ occurrences of a nonlinear variable X *matches* a subject tree t at node n if there exists a subtree t_X of the tree t and subtrees $t_1, t_2, \ldots, t_m$ (not necessarily the same) of the tree t such that the tree np', obtained from np by substituting the subtree t_X for the i-th, $1 \leq i \leq k$, occurrences of X in np, and by substituting the subtree t_i for the i-th occurrence of S in p, $i = 1, 2, \ldots, m$, is equal to the subtree of t rooted at n.

Example 2. Consider a tree $t_1 = (\{a2_1, a2_2, a0_3, a1_4, a0_5, a1_6, a0_7\}, R_1)$ from Example 1, which is illustrated in Figure 1.

Consider a tree pattern p_1 over $\mathcal{A}$, $p_1 = (\{a2_1, a0_2, a1_3, a0_4\}, R_{p1})$. Tree pattern p_1 in prefix notation os $pref(p_1) = a2\ a_0\ a_1\ a_0$.

$$R_{p1} = \{((a2_1, a0_2), (a2_1, a1_3)), ((a1_3, a0_4))\}$$

Consider a tree pattern p_2 over $\mathcal{A} \cup \{S\}$, $p_2 = (\{a2_1, S_2, a1_3, S_4\}, R_{p2})$. Tree pattern p_2 in prefix notation is $pref(p_2) = a2\ S\ a1\ S$. Note that symbol S can occur in a nonlinear tree pattern and it serves as a linear variable.

$$R_{p2} = \{(a2_1, S_2), (a2_1, a1_3), (a1_3, S_4)\}$$

Consider a nonlinear tree pattern p_3 over $\mathcal{A} \cup \{S, X\}$, $p_3 = (\{a2_1, X_2, a1_3, X_4\}, R_{p3})$. Nonlinear tree pattern p_3 in prefix notation is $pref(p_3) = a2\ X\ a1\ X$.

$$R_{p3} = \{(a2_1, X_2), (a2_1, a1_3), (a1_3, X_4)\}$$

Tree patterns p_1, p_2 and p_3 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. Nonlinear tree pattern p_3 has one occurrence in tree t_1 – it matches at node 2 of t_1.

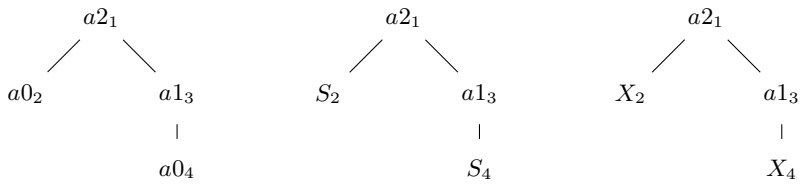

Fig. 2. Tree pattern p_1 (left), tree template p_2 (center) and nonlinear tree template p_3 (right) from Example 2

3.3. Language, finite and pushdown automata

We define notions from the theory of string languages similarly as they are defined in [2, 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 $x^* = \{x^m : m \geq 0\}$, $x^+ = \{x^m : m \geq 1\}$ and $x^* = x^+ \cup \{\varepsilon\}$.

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.

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

Unreachable states are states $p \in Q$ from automaton $M = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$ which are not reachable from the initial state because there is no sequence of transitions from the initial state to that particular state p. Formally, there are no transitions that allow $(q_0, kw, Z_0) \vdash_M^+ (p, w, \gamma)$.

Unnecessary states are states $p \in Q$ from automaton $M = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$ which are not connected to any final state $f \in F$ if automaton accepts by final states, or not connected to any state, where $\gamma \in G^*$ may be ε, if automaton accepts by empty pushdown store.

Pushdown automaton $M = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$ is acyclic if it does not contain transitions $(q, x_1, \gamma_1) \vdash_M^+ (q, x_2, \gamma_2)$, where $xx_2 = x_1$, $x \neq \varepsilon$ and $q \in Q$.

3.4. String suffix and factor automata

String suffix and factor automata are finite automata that were introduced in [4, 7] as a mechanism for eliminating redundancy in string suffix trees [9, 10,

22, 25]. 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 [9, 10, 25], suffix and factor automata are defined as such minimal deterministic finite automata. In [23, 22], their basic nondeterministic versions are also presented. In some literature, the deterministic suffix automaton is also called the *directed acyclic word graph (DAWG)*.

Example 3. Given a string $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$, which is the prefix notation 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 Figure 3. For the construction of the nondeterministic suffix automaton, see [22].

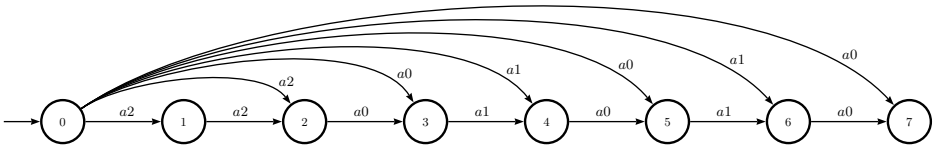

Fig. 3. Transition diagram of the nondeterministic string suffix automaton $FM_{nsuf}(pref(t_1))$ for prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ of tree t_1 from Example 1

4. Indexing trees for tree pattern matching

In this section, algorithms and theorems regarding tree pattern PDAs for trees in prefix notation are given, and the tree pattern PDAs and their construction are demonstrated on an example. A tree pattern can be either a subtree or a tree template, which contains at least one special nullary symbol S representing a subtree. Tree pattern PDAs are an extension of subtree PDAs, introduced in [18]. A subtree PDA is analogous to the string suffix automaton and it accepts a linear notation of all subtrees of a given tree. The pushdown operations are used to process the tree structure. New states and transitions, which are used for processing the special nullary symbols S in tree templates, are additionally present in the tree pattern PDA. The pushdown operations are the same.

Definition 1. *Let t and $pref(t)$ be a tree and its prefix notation, respectively. A* 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 match the subject tree and contain the root of the subject tree. The deterministic treetop PDA is defined by the following definition. States and transitions of the treetop pushdown automaton are computed by Algorithm 1. Finally, the correctness Algorithm 1 is proved by Theorem 1.

Definition 2. *Let t, r and $pref(t)$ be a tree, its root and its prefix notation, respectively. A* treetop pushdown automaton $M_{pt}(t) = (0, 1, 2, ..., n, A \cup S, S, \delta, 0, S, \emptyset)$ *for $pref(t)$ accepts all tree patterns in prefix notation which have the root r and match the tree t.*

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

Algorithm 1 *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 $pref(t) = a_1 a_2 \dots a_n$, $n \geq 1$.*
Output: *Treetop PDA $M_{pt}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A} \cup \{S\}, \{S\}, \delta, 0, S, \emptyset)$.*
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 $S^0 = \varepsilon$.*
2. *Create a set $srms = \{\ i\ : 1 \leq i \leq n,\ \delta(i-1, a, S) = (i, \varepsilon),\ a \in \mathcal{A}_0\}$. The abbreviation* srms *stands for Subtree Right Most States.*
3. *For each state i, where $i = n-1, n-2, \dots, 1$, $\delta(i, a, S) = (i+1, S^p)$, $a \in \mathcal{A}_p$, 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 p-th smallest integer such that $l \in srms$ and $l > i$. Remove all j, where $j \in srms$, and $i < j < l$, from $srms$. □

The treetop PDA is similar to the prefix string finite automaton. Moreover, there exists additional transitions reading symbol S, which represents a subtree, and these transitions skip over parts which are subtrees of the tree in prefix notation. The automaton uses the pushdown store for computing a checksum so that the input would be a valid prefix notation of a tree.

The construction of treetop PDA by Algorithm 1 is illustrated in the following example.

Example 4. Consider tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1, which is illustrated in Figure 1. The deterministic treetop PDA, constructed by Algorithm 1, is deterministic PDA $M_{pt}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_1, 0, S, \emptyset))$, where mapping δ_1 is a set of the following transitions:

$$
\begin{array}{ll}
\delta_1(0, a2, S) = (1, SS) & \\
\delta_1(1, a2, S) = (2, SS) & \delta_1(1, S, S) = (5, \varepsilon) \\
\delta_1(2, a0, S) = (3, \varepsilon) & \delta_3(2, S, S) = (3, \varepsilon) \\
\delta_1(3, a1, S) = (4, S) & \delta_1(3, S, S) = (5, \varepsilon) \\
\delta_1(4, a0, S) = (5, \varepsilon) & \delta_1(4, S, S) = (5, \varepsilon) \\
\delta_1(5, a1, S) = (6, S) & \delta_1(5, S, S) = (6, \varepsilon) \\
\delta_1(6, a0, S) = (7, \varepsilon) & \delta_1(6, S, S) = (7, \varepsilon)
\end{array}
$$

The transition diagram of deterministic treetop PDA $M_{pt}(t_1)$ is illustrated in Figure 4. In this figure for each transition rule $\delta(p, a, \alpha) = (q, \beta)$ from δ the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$.

Deterministic treetop PDA $M_{pt}(t_1)$ has been constructed by Algorithm 1 as follows. We can see that the initial set $srms = \{3, 5, 7\}$. Then, new transitions, which read symbol S, are created in the following order: $\delta_4(6, S, S) = (7, \varepsilon)$, $\delta_4(5, S, S) = (7, \varepsilon)$, $\delta_4(4, S, S) = (5, \varepsilon)$, $\delta_4(3, S, S) = (5, \varepsilon)$, $\delta_4(2, S, S) = (3, \varepsilon)$, and $\delta_4(1, S, S) = (5, \varepsilon)$. □

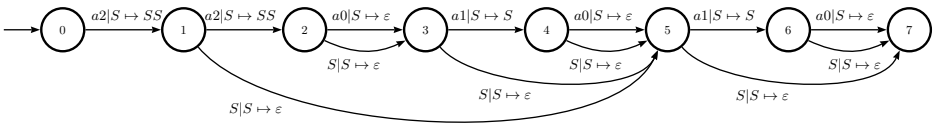

Fig. 4. Transition diagram of deterministic treetop pushdown automaton $M_{pt}(t_1)$ for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 4

Theorem 1. *Given a tree t and its prefix notation $pref(t)$, the PDA $M_{pt}(t)$ constructed by Algorithm 1 is a treetop PDA for $pref(t)$.*

Proof. Let r be the root of t. The PDA $M_{pt}(t)$ is a simple extension of the PDA, which is constructed by step 1 and accepts the tree t in prefix notation. It holds for new transitions added by step 3, which read the special nullary symbol S, that $\delta(q_1, S, S) = (q_2, \varepsilon)$ if and only if $(q_1, w, S) \vdash^+_{M_{pt}(t)} (q_2, \varepsilon, \varepsilon)$ and q_1 is not the initial state 0. This means that the new added transitions reading S correspond just to subtrees not containing the root r. Thus, the PDA $M_{pt}(t)$ accepts all tree patterns in prefix notation which contain the root r and match the tree t. □

The nondeterministic tree pattern PDA for trees in prefix notation is constructed as an extension of the deterministic treetop PDA: for each state of the treetop PDA with an incoming transition which reads a symbol $a \in \mathcal{A}$ we add the same transition from the starting state to that state. This construction is described by the following algorithm.

Algorithm 2 *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 $pref(t) = a_1 a_2 \dots a_n$, $n \geq 1$.*
Output: *Nondeterministic tree pattern PDA $M_{npt}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A} \cup \{S\}, \{S\}, \delta, 0, S, \emptyset)$.*
Method:

1. *Create $M_{npt}(t)$ as $M_{pt}(t)$ by Algorithm 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 $S^0 = \varepsilon$.* □

The tree pattern PDA is similar to the string factor finite automaton. Its construction is based on the treetop PDA and the extension is that the tree pattern accepted by the automaton can be matched on any node of the tree. For this reason, additional transitions are created.

Example 5. Consider tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1, which is illustrated in Figure 1. The nondeterministic tree pattern PDA accepting all tree patterns matching tree t_1, which has been constructed by Algorithm 2, is nondeterministic PDA $M_{npt}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_2, 0, S, \emptyset))$, where mapping δ_2 is a set of the following transitions:

$$\begin{array}{lll}
\delta_2(0, a2, S) = (1, SS) & & \\
\delta_2(1, a2, S) = (2, SS) & \delta_2(1, S, S) = (5, \varepsilon) & \delta_2(0, a2, S) = (2, SS) \\
\delta_2(2, a0, S) = (3, \varepsilon) & \delta_3(2, S, S) = (3, \varepsilon) & \delta_2(0, a0, S) = (3, \varepsilon) \\
\delta_2(3, a1, S) = (4, S) & \delta_2(3, S, S) = (5, \varepsilon) & \delta_2(0, a1, S) = (4, S) \\
\delta_2(4, a0, S) = (5, \varepsilon) & \delta_2(4, S, S) = (5, \varepsilon) & \delta_2(0, a0, S) = (5, \varepsilon) \\
\delta_2(5, a1, S) = (6, S) & \delta_2(5, S, S) = (6, \varepsilon) & \delta_2(0, a1, S) = (6, S) \\
\delta_2(6, a0, S) = (7, \varepsilon) & \delta_2(6, S, S) = (7, \varepsilon) & \delta_2(0, a0, S) = (7, \varepsilon)
\end{array}$$

The transition diagram of nondeterministic tree pattern PDA $M_{npt}(t_1)$ is illustrated in Figure 5. Again, in this figure for each transition rule $\delta(p, a, \alpha) = (q, \beta)$ from δ the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$. □

In the following theorem we prove the correctness of the constructed tree pattern PDA.

Theorem 2. *Given a tree t and its prefix notation $pref(t)$, the PDA $M_{npt}(t)$ constructed by Algorithm 2 is a tree pattern PDA for $pref(t)$.*

Proof. The PDA $M_{npt}(t)$ is a simple extension of the PDA $M_{pt}(t)$, which is constructed by Algorithm 1 and accepts all tree patterns in prefix notation which contain the root r of the tree t and match the tree t by empty pushdown store. The PDA $M_{npt}(t)$ contains new added transitions of the form $\delta(0, a_i, S) = (i, S^{Arity(a_i)})$. These transitions correspond just to the possibility that the first

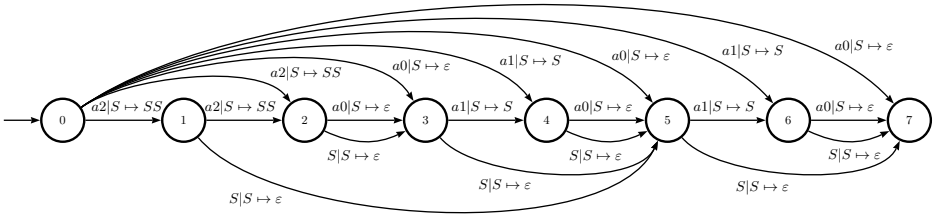

Fig. 5. Transition diagram of nondeterministic tree pattern pushdown automaton M_{npt} (t_1) from Example 5 for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

symbol of a tree pattern to be accepted can be any node of the tree t. Thus, the PDA $M_{npt}(t)$ accepts all tree patterns in prefix notation which match the tree t. □

Lemma 1. *Given a tree t with n nodes, the number of distinct tree patterns which match the tree t can be at most $2^{n-1} + n$.*

Proof. First, subtrees of any subtree of the tree t can be replaced by the special nullary symbol S and the tree template resulting from such a replacement is a tree pattern which matches the tree. Given a tree with n nodes, the maximal number of subsets of subtrees that can be replaced by the special nullary symbol S occurs for the case of a tree t_2 whose structure is given by the prefix notation $pref(t_2) = a(n-1)\ a_1 0\ a_2 0 \ldots a_{n-1} 0$, where $n \geq 2$. Such a tree is illustrated in Figure 6. In this tree, each of the nullary symbols $a_1 0, a_2 0, \ldots, a_{n-1} 0$ can be replaced by nullary symbol S, and therefore we can create 2^{n-1} distinct tree templates which are tree patterns matching the tree t_2.

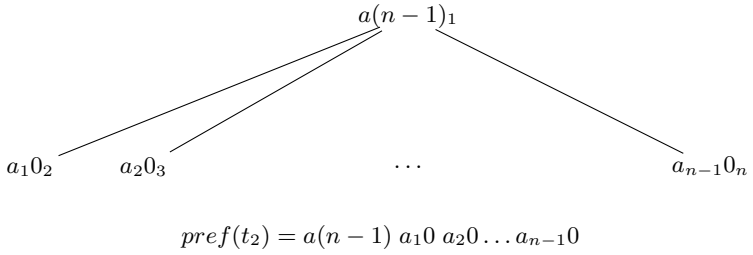

Fig. 6. A tree t_2 with $2^{n-1} + n$ distinct tree patterns matching the tree t_3 and its prefix notation

Second, the tree t itself and all its subtrees not containing the root are tree patterns which match the tree, which gives n other distinct tree patterns (provided all the subtrees are unique).

Thus, the total number of distinct tree patterns matching the tree t can be at most $2^{n-1} + n$. □

Lemma 2. *The number of states of a nondeterministic tree pattern pushdown automaton M_{ntp} is $m + 1$, where m is the number of nodes of a subject tree.*

Proof. There is one state for each symbol in $pref(t)$ plus and the initial state. Thus, the number of states is $m + 1$. □

Lemma 3. *The number of transitions of a nondeterministic tree pattern pushdown automaton M_{ntp} is $3m - 2$, where m is the number of nodes of a subject tree.*

Proof. There is one transition for each symbol in $pref(t)$, which forms the "backbone" of the automaton. There are exactly $m-1$ transitions from the initial state to every other state. Finally, there is one transition for symbol S leading from every state except the initial state. Thus, the number of states is then $3m - 2$. □

5. Indexing trees for nonlinear tree pattern matching

Definition 3. *Let t and $pref(t)$ be a tree and its prefix notation, respectively. A* nonlinear tree pattern pushdown automaton *for $pref(t)$ accepts all nonlinear tree patterns in prefix notation which have at most one nonlinear variable and match the tree t.*

5.1. Basic nonlinear tree pattern pushdown automaton

In our indexing pushdown automata for nonlinear tree pattern matching we construct new parts called tails, which represent parts of the pushdown automaton after reading nonlinear variables.

Definition 4. *Given a tree pattern pushdown automaton $M = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$ and a state $q_t \in Q$, the $tail(M, q_t) = (Q_t, \mathcal{A}, G, \delta_t, q_t, S, F)$. $Q_t = Q \setminus Q_{us}$, Q_{us} is a set of unreachable states from q_t, $\delta_t = \delta \setminus \delta_{us}$, δ_{us} are transitions leading from or to state $q_n \in Q_{us}$.*

Example 6. Consider a tree pattern pushdown automaton $M_{npt}(t_1)$ from Example 5, which is an index of tree t_1 from Example 1. The tail of automaton with initial state $q_t = 3$ is $tail(M_{npt}(t_1), 3) = (Q, \mathcal{A} \cup \{S\}, \{S\}, \delta, 3, S, \varnothing)$ constructed from tree pattern pushdown automaton shown in Figure 5. The corresponding transition diagram is illustrated in Figure 7. □

We note that every node of a tree t is the root of just one subtree, which is represented by symbol S. The prefix notation of such subtree is a factor of $pref(t_1)$. These factors are in the tree pushdown automaton "skipped" by transitions for input symbol S.

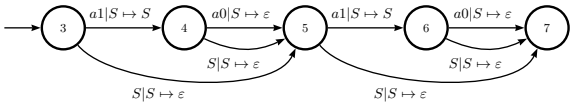

Fig. 7. Tail of tree pattern pushdown automaton $tail(M_{npt}(t_1), 3)$ from Example 6

Definition 5. *Given a tree pattern pushdown automaton* $M_{npt}(t) = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$ *and a state* $q \in Q$*, the* subtree skipped by transition $sst(q) = b_1 b_2 \dots b_m$*, where* $b_1, b_2, \dots, b_m \in \mathcal{A}$*, is given by a labelled path* $b_1, b_2, \dots, b_m$ *in the PDA* $M_{npt}(t)$ *between states* q *and* q_t*, where* $(q_t, \varepsilon) \in \delta(q, S, S)$*.*

Informally, $sst(q)$ is the prefix notation of the subtree which is skipped by transition S leading from the state q. $sst(q)$ is used in Algorithm 3 to determine which subtree of the subject tree was "assigned" to a particular automaton tail.

Example 7. Consider a tree pattern pushdown automaton $M_{npt}(t_1)$ from Example 5, which is an index of tree t_1 from Example 1. The subtree skipped by transition $sst(1) = a2\ a0\ a1\ a0$. □

The construction of basic nonlinear tree pattern PDA consists of two algorithms. Algorithm 3 constructs tails from the original tree pattern pushdown automaton. Algorithm 4 recursively connects these created tails to the pushdown automaton being created.

Algorithm 3 Recursive construction of tail of nondeterministic basic nonlinear tree pattern automaton.
Input: Tail of nondeterministic tree pattern pushdown automaton M_{tnpt}, string representing subtree skipped by transition $x = sst(q)$.
Output: Recursively created tail $nta(M_{tnpt}, x)$.
Method:

1. For each transition $(q_t, \varepsilon) \in \delta(q, S, S)$ in automaton M_{tnpt} where $sst(q) = x$ do:
   1.1. Create $M_{tmp} = nta(tail(M_{tnpt}, q_t), x)$ using Algorithm 3.
   1.2. Add new state q_{id} to M_{tnpt} where q_{id} is copy of state q_t.
   1.3. Add new transition $(q_{id}, \varepsilon) \in \delta(q, X, S)$ to M_{tnpt}.
   1.4. Add M_{tmp} to M_{tnpt} and merge initial state of M_{tmp} with q_{id}.
2. $nta(M_{tnpt}, x)$ is M_{tnpt}.

Algorithm 4 Construction of nondeterministic basic nonlinear tree pattern pushdown automaton.
Input: Nondeterministic tree pattern pushdown automaton $M_{npt}(t)$.
Output: Nondeterministic basic nonlinear tree pattern pushdown automaton $M_b(t)$.
Method:

1. For each transition $(q_t, \varepsilon) \in \delta(q, S, S)$ in automaton $M_{npt}(t)$ do:

1.1. Create $M_{tmp} = nta(tail(M_{npt}(t), q_t), sst(q))$ using Algorithm 3.
1.2. Add new state q_{id} to $M_{npt}(t)$ where q_{id} is copy of state q_t.
1.3. Add new transition $(q_{id}, \varepsilon) \in \delta(q, X, S)$ to $M_{npt}(t)$.
1.4. Add M_{tmp} to $M_{npt}(t)$ and merge initial state of M_{tmp} with q_{id}.
2. $M_b(t)$ is $M_{npt}(t)$.

The nonlinear tree pattern PDA is similar to the tree pattern PDA. The difference between Algorithm 3 and Algorithm 4 is that Algorithm 3 calls itself only when processing transition for symbol S leading from state q, where $sst(q)$ equals its subtree parameter. On the other hand, Algorithm 4 calls Algorithm 3 for each transition for symbol S.

Example 8. Given a string $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$, which is a prefix notation of tree t_1 from Example 1, the corresponding nondeterministic basic nonlinear tree pattern pushdown automaton is
$M_b(t_1) = (Q, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \varnothing)$, where its transition diagram is illustrated in Figure 8. □

5.2. Nonlinear tree pattern pushdown automaton

Some states of the pushdown automaton constructed by Algorithm 4 can be merged so that states in nondeterministic nonlinear tree pattern pushdown automaton M_{nntp} for a subject tree t $M_{nntp}(t) = (\{0, 1, 2, \ldots, n, x_1, \ldots, n_1, y_2, \ldots, n_2, \ldots, z_m, \ldots, n_m\}, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \varnothing)$ would still track both assigned subtree and the same number of nonlinear variables read from the pattern. Merged states are those from tails with the same assigned subtree and the same number of nonlinear variables read.

Definition 6. Let $M_b(t) = (Q, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, q_0, S, \varnothing)$ be a basic nondeterministic nonlinear tree pattern PDA constructed by Algorithm 4. Let x be the longest string over alphabet $\mathcal{A}$, where $(q, x, \alpha) \vdash^*_{Mb} (q_f, \varepsilon, \beta)$. The *tree node state label* $tnsl(q)$ is defined $tnsl(q) = |pref(t)| - |x|$.

Algorithm 5 Algorithm for counting the $tnsl$.
Input: Nondeterministic basic nonlinear tree pattern pushdown automaton $M_b(t)$ and state q for which the $tnsl$ is counted.
Output: Number representing $tnsl$.
Variables: Temporary number n, State $initial$.
Method:

1. n = 0. $initial$ is the starting state of $M_b(t)$.
2. Do:
   2.1. If exists transition $(q, S^{arity(a)}) \in \delta(q_{prev}, a, S)$ where $a \in \mathcal{A}$ and $q_{pref} \neq initial$ do:
      2.1.1. $n = n + 1$, $q = q_{prev}$.
      2.1.2. Goto step [2.].

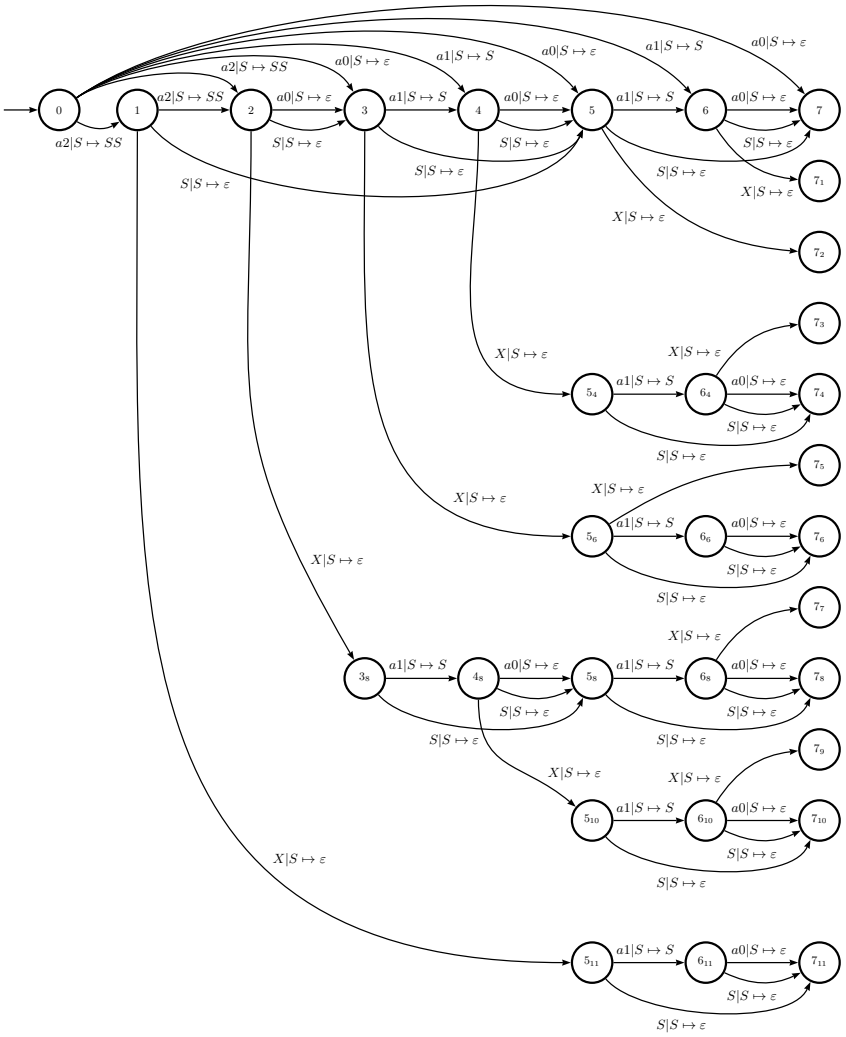

Fig. 8. Basic nondeterministic nonlinear tree pattern pushdown automaton $M_b(t_1)$ from Example 8 constructed for tree t_1 shown in Figure 1

2.2. If exists transition $(q, \varepsilon) \in \delta(q_{prev}, X, S)$ where X is nonlinear variable do:
  2.2.1. $n = n + |sst(q_{prev})|$, $q = q_{prev}$.
  2.2.2. Goto step [2.].
2.4. Output $n + 1$.

Example 9. Given a basic nondeterministic nonlinear tree pattern pushdown automaton is $M_b(t_1) = (Q, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \varnothing)$, its transition diagram is shown in Figure 8.

Then, $tnsl(3) = 3$, $tnsl(5_4) = 5$, $tnsl(7_{11}) = 7$, $tnsl(7_9) = 7$. □

Definition 7. Given a basic nondeterministic nonlinear tree pattern pushdown automaton $M_b(t) = (Q, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \varnothing)$ created by Algorithm 4, the *number of nonlinear variable transitions* $nnv(q, X)$ is the number of transitions reading nonlinear variable X on the path from the initial state q_0 to state q, where q and $q_0 \in Q$.

Algorithm 6 Algorithm for counting the nnv.
Input: Basic nondeterministic nonlinear tree pattern pushdown automaton $M_b(t)$ and state q for which the $tnsl$ is counted.
Output: Number representing nnv.
Variables: Temporary number n, State $initial$.
Method:

1. n = 0. $initial$ is the starting state of $M_b(t)$.
2. Do:
  2.1. If exists transition $(q, S^{arity(a)}) \in \delta(q_{prev}, a, S)$ where $a \in \mathcal{A}$ and $q_{pref} \neq initial$ do:
    2.1.1. $q = q_{prev}$.
    2.1.2. Goto with step [2.].
  2.2. If exists transition $(q, \varepsilon) \in \delta(q_{prev}, X, S)$ where X is nonlinear variable do:
    2.2.1. $n = n + 1$, $q = q_{prev}$.
    2.2.2. Goto with step [2.].
  2.3 Output n.

Definition 8. Given a nondeterministic basic nonlinear tree pattern pushdown automaton $M_b(t)$ created by Algorithm 4, the *mergeable states* $ms(M_b(t))$ is a collection of pairs (key, $value$), where key is a triplet ($sst(q)$, $nnv(u, X)$, $tnsl(u)$) and $value$ is a set of states. $ms(M_b(t))$ stores sets of states with the same number of transitions reading nonlinear variable X $nnv(q, X)$ and subtree skipped by transition $sst(q)$.
$ms(M_b(t)) = \{(sst(q_x), nnv(s_{a1}, X), tnsl(s_{a1})), \{s_{a1}, s_{a2}, \ldots\}),$
$(sst(q_y), nnv(s_{b1}, X), tnsl(s_{b2})), \{s_{b1}, s_{b2}, \ldots\}), \ldots\}$, where the first state s_1 from each set is the main state. State v is $sst(v)$ denoting state for state s_1 given by $(v, X\omega, S\gamma) \vdash (s_1, \omega, \gamma)$, where $\omega = (\mathcal{A} \cup \{S, X\})^*$. All states from that set are given by following: $\{\forall s : nnv(s, X) = nnv(s_1, X)$ and $sst(v) = sst(u)$ and $tnsl(s) = tnsl(s_1); s, s_1, u, v \in Q\}$, where state u is $sst(u)$ denoting state for state s given by $(u, X(\mathcal{A} \cup \{S\})^*\omega, S^\alpha) \vdash^* (s, \omega, S^\beta)$, where $\omega = (\mathcal{A} \cup \{S, X\})^*$.

Each set from the collection of sets of mergeable states $ms(M_b(t))$ defines states from nondeterministic basic nonlinear tree pattern pushdown automaton $M_b(t)$ that can be merged and the resulting automaton is called nondeterministic nonlinear tree pattern pushdown automaton $M_{nntp}(t)$. All states from each set defines the start of a merging process so that states that are reachable by the same sequence of transitions are also merged.

Example 10. Given a string $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$, which is the prefix notation of tree t_1 from Example 1. The corresponding nondeterministic basic nonlinear tree pattern pushdown automaton is $M_b(t_1) = (Q, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \varnothing)$, where its transition diagram and states are illustrated in Figure 8.

All states that occur in one of the set in the collection $ms(M_b(t_1))$ are target states from all transitions for a symbol X and the transitions for a symbol S which shares the source state.

$$ms(M_b(t_1)) = \{((a0, 1, 5), \{5_4, 5_8\}), ((a0, 1, 7), \{7_1, 7_4, 7_8\}),$$
$$((a0, 2, 7), \{7_3, 7_7, 7_{10}\}), ((a1a0, 1, 7), \{7_2, 7_6\})\}.$$

Algorithm 7 Construction of the nondeterministic nonlinear tree pattern pushdown automaton.
Input: Nondeterministic basic nonlinear tree pattern pushdown automaton $M_b(t)$.
Output: Nondeterministic nonlinear tree pattern pushdown automaton $M_{nntp}(t)$.
Variables: Collection of sets of states $ms(M_b(t))$.
Method:

1. For all transitions $(u_1, \varepsilon) \in \delta(q, X, S)$ do:
   1.1. If the collection $ms(M_b(t))$ does not contain a set on a key ($sst(q)$, $nnv(u_1, X)$, $tnsl(u_1)$) create that set as an empty set.
   1.2. Add u_1 to the collection $ms(M_b(t))$ to the set on the key ($sst(q)$, $nnv(u_1, X)$, $tnsl(u_1)$).
2. For all transitions $(u_2, \varepsilon) \in \delta(q, S, S)$, where exists a transition $(u_1, \varepsilon) \in \delta(q, X, S)$ do:
   2.1. If $nnv(u_2, X) \neq 0$ and the collection $ms(M_b(t))$ does not contain a set on a key ($sst(q)$, $nnv(u_2, X)$, $tnsl(u_2)$) create that set as an empty set.
   2.2. Add u_2 to the collection $ms(M_b(t))$ to the set on the key ($sst(q)$, $nnv(u_2, X)$, $tnsl(u_2)$).
3. For each set in the collection $ms(M_b(t))$ do:
   3.1. Merge all states in this set, along with all states that follows-up.

Example 11. Given a string $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$, which is the prefix notation of tree t_1 from Example 1, the corresponding nondeterministic nonlinear tree pattern pushdown automaton is $M_{nntp}(t_1) = (Q, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \varnothing)$, where merged states are in Example 10 and its transition diagram and states are illustrated in Figure 9. □

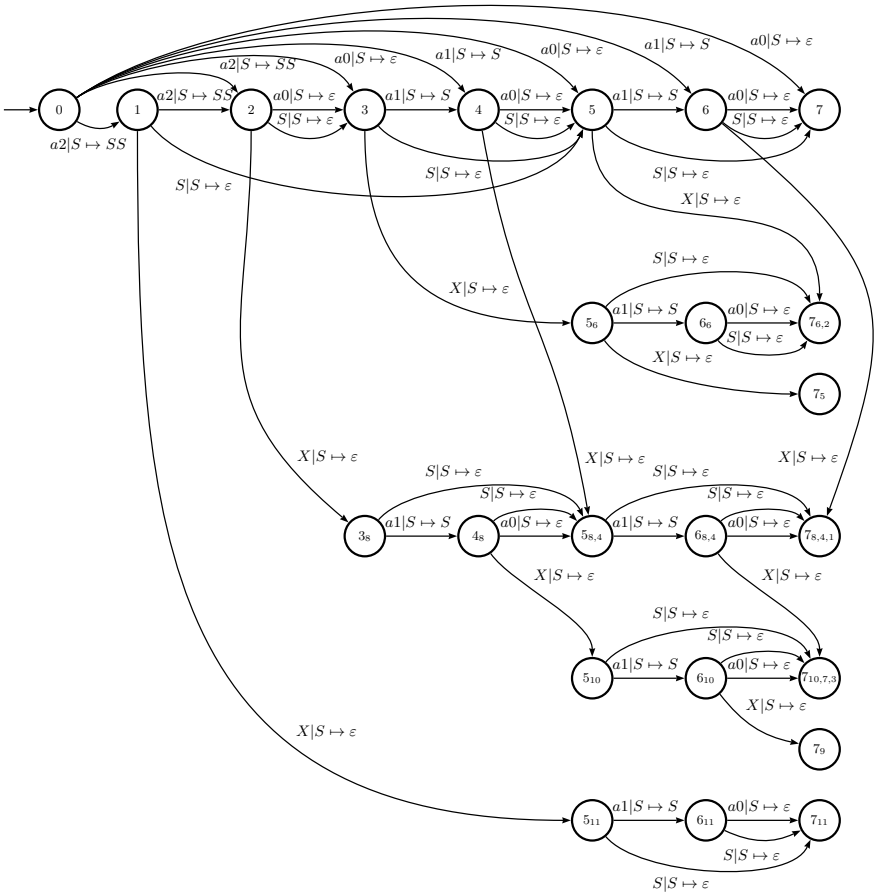

Fig. 9. Nondeterministic nonlinear tree pattern pushdown automaton $M_{nntp}(t_1)$ from Example 11 constructed by Algorithm 7 for subject tree shown in Figure 1

The nondeterministic nonlinear tree pattern pushdown automaton can be even minimalised by omitting the $nnv(q, X)$ part of the key value pairs of the collection $ms(M_b(t))$. The resulting automaton would represent an index of the subject tree for nonlinear tree pattern matching but would not be able to say how many nonlinear variables has been read during processing the nonlinear tree pattern.

5.3. Time and Space Complexity Analysis

Lemma 4. *Time complexity of accepting the nonlinear tree template by automaton created by Algorithm 7 is $\mathcal{O}(\sum_K k_i)$, where K is the set of all prefixes except ε, and k_i is the number of distinct sequences of transitions in automaton $M_{nntp}(t)$ for $k_i \in K$ which ends in a state of automaton M_{nntp}.*

Proof. Automata have to try all possible sequences of transitions according to tree template which occur in the nondeterministic nonlinear tree pattern automaton. Sequences of symbols of these transitions form a prefix of tree template. Prefix of the size of one symbol from tree template is handled by exactly n steps, where n is the number of all possible sequences of transitions in the automaton for that prefix. Prefix of the size of two symbols is handled by $n + m$ steps, where m is the number of all possible sequences of transitions in the automaton for that prefix. Note that handling two symbols prefix requires two transitions to be processed, however the first transition is already accounted by prefix of size of one symbol.

Exact time complexity is then the sum of all possible sequences of transitions in the automaton for all prefixes of nonlinear tree template, which is $\mathcal{O}(\sum_S rs_i)$. □

Lemma 5. *The number of states of nondeterministic nonlinear tree pattern pushdown automaton $M_{nntp}(t)$ created by Algorithm 7 is $\mathcal{O}(n(\sum_{i=0}^{s} r_i)) = \mathcal{O}(n^2)$, where n is the number of nodes of a subject tree and $\sum_{i=0}^{s} r_i$, where s is the number of distinct subtrees, and r_i is the number of repetitions of each subtree.*

Proof. Each occurrence of each unique subtree in tree increments the number of automaton tails, that were created for this subtree. The exact number of tails created for particular subtree is then r_i, where r_i is the number of repetitions of that subtree. Then the total number of tails for one nonlinear variable in automaton is the number of tails created for each unique subtree of indexed tree which is $\sum_{i=0}^{s} r_i$. The total number of tails does not count original automaton. The exact number of states of the automaton for one nonlinear variable is $\mathcal{O}(n(\sum_{i=0}^{s} r_i + 1))$ = $\mathcal{O}(n(\sum_{i=0}^{s} r_i))$. □

Lemma 6. *The number of transitions of nondeterministic nonlinear tree pattern pushdown automaton $M_{nntp}(t)$ created by Algorithm 7 is $\mathcal{O}(n^2 + n + \sum_{i=0}^{s}(\frac{r_i^2 + r_i}{2})) = \mathcal{O}(n^2)$, where n is the number of nodes of a subject tree, s is the number of distinct subtrees and r_i is the number of repetitions of each unique subtree.*

Proof. For all tails for one nonlinear variable, there are transitions reading symbol X between these tails. There is one transition heading to the last tail. There are two transitions heading to the previous tail, and so on. The number of transitions reading symbol X is $\sum_{i=0}^{s}(\frac{r_i^2+r_i}{2})$.

Using Lemma 5 the number of transitions for symbol S is $\frac{1}{2}n^2$ and the number of transitions for symbol $a \in \mathcal{A}$ is $\frac{1}{2}n^2 + n$.

The number of transitions then is $\mathcal{O}(\sum_{i=0}^{s}(\frac{r_i^2+r_i}{2}) + n^2 + n)$. □

Lemma 7. *Given a tree t with n nodes, the number of distinct nonlinear tree patterns which match the tree t can be at most $3^{n-1} + 2$.*

Proof. First, subtrees of any subtree of the tree t can be replaced by the special nullary symbol S and the tree template resulting from such a replacement is a tree pattern which matches the tree. Second, subtrees of any subtree of the tree t can be replaced by the special nullary symbol X and the nonlinear tree template resulting from such replacement is a nonlinear tree pattern which matches the tree. Given a tree with n nodes, the maximal number of subsets of subtrees that can be replaced by the special nullary symbol S, X occurs for the case of a tree t_3 whose structure is given by the prefix notation $pref(t_3) = a(n-1)\ a_1 0\ a_2 0 \ldots a_{n-1}0$, where $n \geq 2$. Such a tree is illustrated in Figure 6. In this tree, each of the nullary symbols $a_1 0, a_2 0, \ldots, a_{n-1}0$ can be replaced by nullary symbol S or X, and therefore we can create 3^{n-1} distinct tree templates which are tree patterns matching the tree t_3.

Third, the tree t itself and all its subtrees not containing the root are tree patterns which match the tree. Subtrees of the tree t must be the same so that the nonlinear tree pattern matched the tree in the first place. These gives 2 other distinct tree patterns.

Thus, the total number of distinct tree patterns matching the tree t can be at most $3^{n-1} + 2$. □

6. Processing more nonlinear variables in nonlinear tree patterns

Indexing for nonlinear tree pattern matching with more than one nonlinear variable can be done by a pushdown automaton created as a pushdown automaton for the intersection of languages. Automaton for two nonlinear variables would be constructed on the basis of two automata – each of them for one nonlinear variable. The disadvantage of this approach would be increasing space complexity.

Another approach is represented by a nondeterministic nonlinear tree pattern pushdown automaton $M_{nntp}(t)$ for one nonlinear variable that can be used as an indexing data structure also for nonlinear tree patterns with more variables. The idea is to compare which transitions of more runs of this single automaton were used to match the pattern. The input pattern needs to be modified because it contains symbols representing the nonlinear variables that the nondeterministic nonlinear tree pattern pushdown automaton can't handle.

Example 12. Consider a ranked alphabet $\mathcal{A} = \{a4, a3, a2, a1, a0\}$. Consider a nonlinear tree template p_4 over $\mathcal{A} \cup \{S, Y, Z\}$ $p_4 = (\{a4_1, X_2, X_3, Y_4, Y_5\}, R_{p4})$ over $\mathcal{A}$, where R_{p4} is a set of the following ordered pairs:

$$R_{p4} = \{(a4_1, X_2), (a4_1, X_3), (a4_1, Y_4), (a4_1, Y_5)\}.$$

Nonlinear tree template p_4 is illustrated in Figure 10.

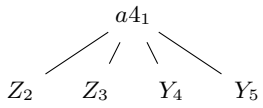

Fig. 10. Nonlinear tree template p_4 from Example 12

Nonlinear tree template p_4 can be decomposed to nonlinear tree templates for one nonlinear variable. These nonlinear tree templates will be over alphabet $\mathcal{A} \cup \{S, X\}$ and are illustrated in Figure 11. □

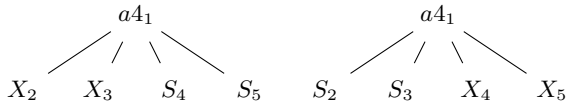

Fig. 11. Decomposition of nonlinear tree template p_4 from Example 12

In the beginning the algorithm decomposes a given nonlinear tree template to nonlinear tree templates of one nonlinear variable. Then, the accepting sequences of transitions are computed using nondeterministic nonlinear tree pattern pushdown automaton $M_{nntp}(t)$ and each decomposed nonlinear tree pattern. These accepting sequences of transitions can be used for filtering real occurrences out of the original tree template for more nonlinear variables.

Algorithm 8 Algorithm of nonlinear tree pattern matching with more nonlinear variables using nondeterministic nonlinear tree pattern pushdown automaton $M_{nntp}(t)$
Input: Nondeterministic nonlinear tree pattern pushdown automaton $M_{nntp}(t)$. Nonlinear tree pattern with more variables p, set $vars$ of variables used in the template p.

Output: Occurrences of the pattern.
Method:

1. Collection of nonlinear templates for one variable po is an empty collection.
2. For each variable var in $vars$ do:
   2.1. pd is a clone of nonlinear tree template p.
   2.2. Change a symbol in leaf nodes to nullary symbol S, where node label $l \in (vars \setminus var)$.
   2.3. Change a symbol in leaf nodes to nullary symbol of nonlinear variable X, where node label $l = var$.
   2.4. Add pd to po.
3. Set Occ contains $\{0, 1, \ldots, n\}$ where n is the size of the tree t.
4. For each nonlinear tree template pd in po do:
   4.1. Determine accepting sequences of transitions ts of tree template t using $M_{nntp}(t)$.
   4.2. Compute a set Occ_{pd} as set of $tnsl(q)$, where q is a target state of the first transition from all ts.
   4.2. Remove all items in Occ which are not in Occ_{pd}.
5. Output Occ.

6.1. Time and Space Complexity Analysis

Lemma 8. *Time complexity of accepting the nonlinear tree template for more nonlinear variables by nondeterministic nonlinear tree pattern pushdown automaton is $\mathcal{O}(v \times m + \sum run(pd) + \sum Occ_{pd})$, where v is the number of nonlinear variables, m is the size of the nonlinear template, $run(pd)$ is the time of locating all accepting transition sequences of each of decomposed templates pd in automaton $M_{nntp}(t)$ and Occ_{pd} is the size of occurrences of decomposed template pd.*

Proof. The nonlinear tree template needs to be decomposed to nonlinear tree templates for one nonlinear variable. This takes $v \times m$ time.

Occurrences of each nonlinear tree template from decomposed nonlinear tree template p are computed in time $\sum run(pd)$.

Composition of partial occurrences Occ_{pd} to Occ can be done in $\sum Occ_{pd}$ time. □

Lemma 9. *Given a tree t with n nodes, the number of distinct nonlinear tree patterns (with more nonlinear variables) which match the tree t can be at most $(2 + v)^{n-1} + 2$.*

Proof. First, subtrees of any subtree of the tree t can be replaced by the special nullary symbol S and the tree template resulting from such a replacement is a tree pattern which matches the tree. Second, subtrees of any subtree of the tree t can be replaced by the special nullary symbols of variables and the nonlinear tree template resulting from such replacement is a nonlinear tree pattern which matches the tree. Given a tree with n nodes, the maximal number of subsets of subtrees that can be replaced by the special nullary symbols of variables occurs for the case of a tree t_3 whose structure is given by the prefix notation $pref(t_3) = a(n-1)\ a_1 0\ a_2 0 \ldots a_{n-1} 0$, where $n \geq 2$. Such a tree is illustrated

in Figure 6. In this tree, each of the nullary symbols $a_1 0, a_2 0, \ldots, a_{n-1} 0$ can be replaced by nullary symbol of variables and therefore we can create $(2+v)^{n-1}$ distinct tree templates which are tree patterns matching the tree t_3.

Third, the tree t itself and all its subtrees not containing the root are tree patterns which match the tree. Subtrees of the tree t must be the same so that the nonlinear tree pattern matched the tree in the first place. These gives 2 other distinct tree patterns.

Thus, the total number of distinct tree patterns matching the tree t can be at most $(2+v)^{n-1}+2$. □

7. Some empirical results

We have implemented the nondeterministic tree pattern pushdown automaton and the nonlinear tree pattern pushdown automaton using the bit-parallelism technique, which was introduced in [27]. For our implementations of transitions reading symbols from $\mathcal{A}$ we use the same approach as it is used in the implementation of nondeterministic string suffix automata in [23]. The transitions reading the special variable nullary symbols must be treated in a special way. When processing these nullary symbols the algorithm takes indexes of each one in configuration bit vector of the bit–parallelism simulation and recompute these indexes according to transitions reading variable symbols in the simulated automaton. A unique id of subtree is stored for each location in the bit vector so that the matching of nonlinear tree patterns is possible.

Our implementations were written in Java programming language; all timings were conducted on a 2 GHz Intel Core i7 with 8 GB of RAM running OpenSUSE GNU/Linux version 12.1 and Java 1.6.0. In Figures 12, 13, and 14, it is shown that the running time for a given tree is linear with the size of the input (nonlinear) tree pattern. In Figures 15, 16, and 17, it is shown that the running time for binary trees is also linear in general but for very small input patterns there is a slowdown caused by recomputing the configuration vector of bit–parallelism for nullary variable symbols.

8. Conclusion

We have presented the tree pattern pushdown automaton and the nonlinear tree pattern pushdown automaton, a new kind of pushdown automata which represent a complete index of a given ordered tree for tree patterns and nonlinear tree patterns, respectively. We have discussed the time and space complexities and have shown timings of our implementations using the bit-parallel technique. The timings are similar to those for the existing bit-parallel implementation of nondeterministic string suffix automata.

Since the presented pushdown automata are input–driven, they can be determinised. However, the space complexities of their deterministic versions are open problems.

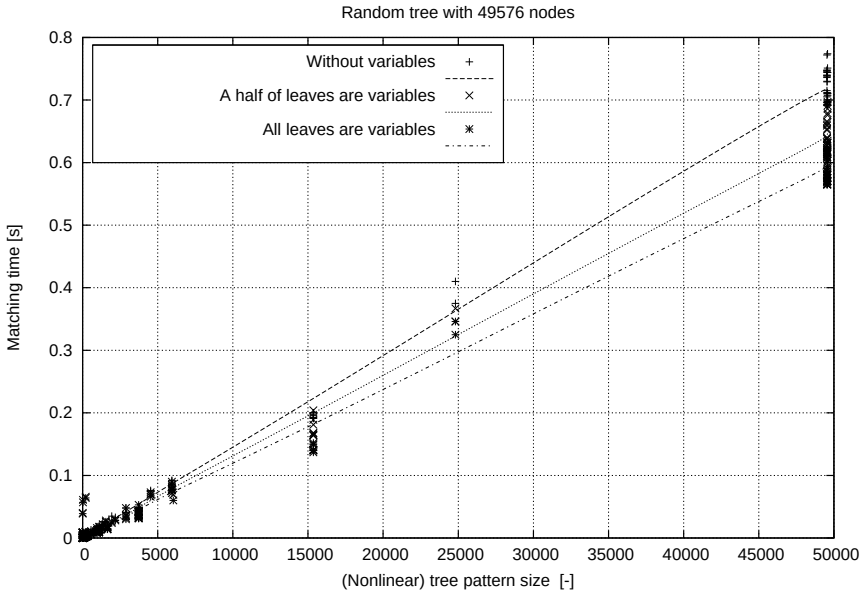

Fig. 12. Random tree with 49576 nodes

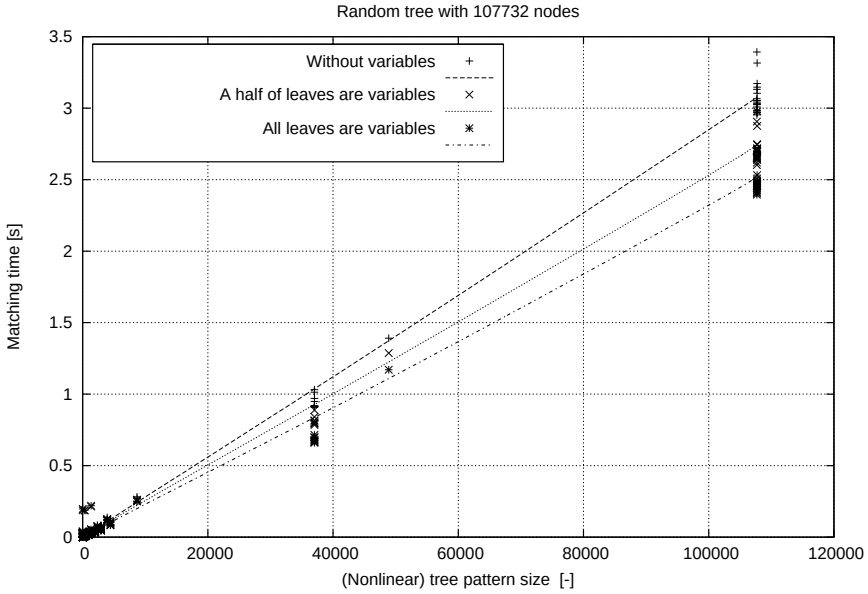

Fig. 13. Random tree with 107732 nodes

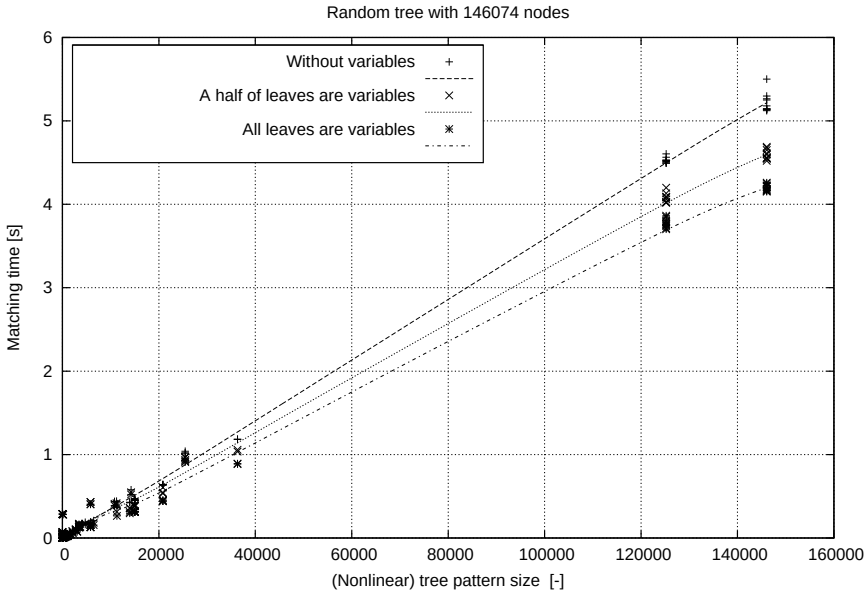

Fig. 14. Random tree with 146074 nodes

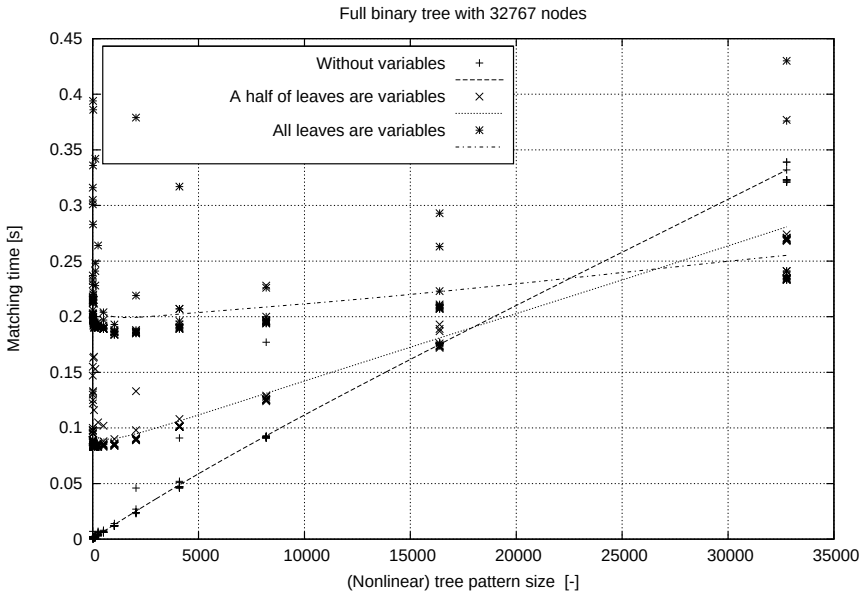

Fig. 15. Full binary tree tree with 32767 nodes

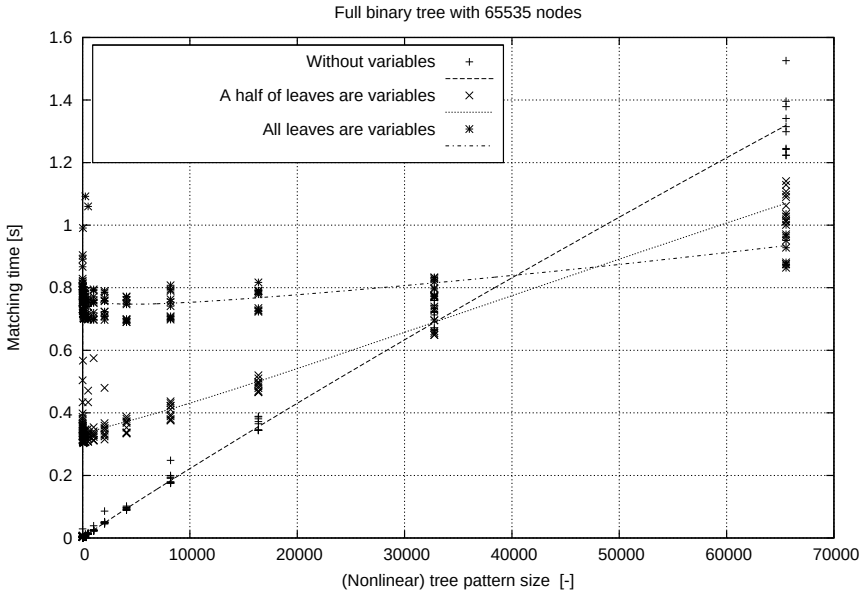

Fig. 16. Full binary tree tree with 65535 nodes

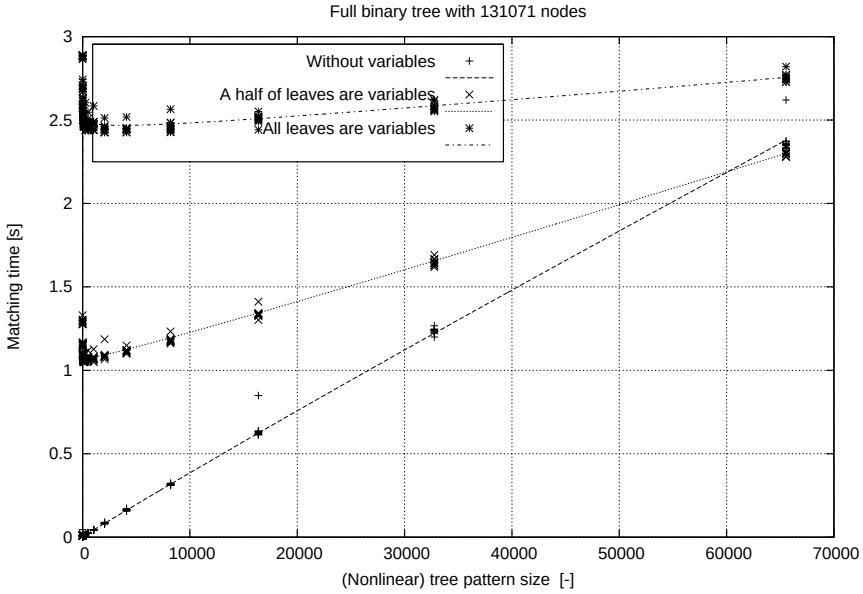

Fig. 17. Full binary tree tree with 131071 nodes

References

1. Aho, Alfred V.; Margaret J. Corasick: Efficient string matching: An aid to bibliographic search. In: *Communications of the ACM*, 18 (6), pp. 333340, 1975.
2. Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling*. Prentice-Hall Englewood Cliffs, N.J., 1972.
3. *Arbology www pages*, Available at: http://www.arbology.org, June 2012.
4. Blumer, A., Blumer, J., Haussler, D., Ehrenfeucht, A., Chen, M. T., Seiferas, J. I., 1985. The smallest automaton recognizing the subwords of a text. Theor. Comput. Sci. 40, 31–55.
5. Christou, M., Crochemore, M., et al., 2011. Computing All Subtree Repeats in Ordered Ranked Trees. In *String Processing and Information Retrieval*, Vol. 7024, pp. 338-343.
6. L.G.W.A. Cleophas: *Tree Algorithms: Two Taxonomies and a Toolkit.* PhD Thesis, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, April 2008.
7. Crochemore, M., 1986. Transducers and repetitions. Theor. Comput. Sci. 45 (1), 63–86.
8. 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.
9. Crochemore, M., Hancart, C., 1997. Automata for matching patterns. In: Rozenberg, G., Salomaa, A. (Eds.), Handbook of Formal Languages. Vol. 2 Linear Modeling: Background and Application. Springer–Verlag, Berlin, Ch. 9, pp. 399–462.
10. Crochemore, M., Rytter, W., 1994. Jewels of Stringology. World Scientific, New Jersey.
11. Domenico Cantone, Simone Faro and Emanuele Giaquinta: A Compact Representation of Nondeterministic (Suffix) Automata for the Bit-Parallel Approach, In: *CPM 2010, LNCS 6129*, Springer, Berlin, 2010.
12. Toms Flouri, Jan Janousek, Borivoj Melichar, Costas S. Iliopoulos, Solon P. Pissis: Tree Template Matching in Ranked Ordered Trees by Pushdown Automata. In: *CIAA 2011, LNCS 6807*, Springer, Berlin, pp. 273-281, 2011.
13. Olivier Gauwin, Joachim Niehren: Streamable Fragments of Forward XPath. In: *CIAA 2011, LNCS 6807*, Springer, Berlin, pp. 3-15, 2011.
14. F Gecseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3 Beyond Words. Handbook of Formal Languages, pages 1–68. Springer–Verlag, Berlin, 1997.
15. Christoph M. Hoffmann and Michael J. O'Donnell. Pattern matching in trees. *J. ACM*, 29(1):68–95, 1982.
16. Hopcroft, J. E., Motwani, R., Ullman, J. D., 2001. Introduction to automata theory, languages, and computation, 2nd Edition. Addison-Wesley, Boston.
17. J. W. Klop. *Term Rewriting Systems*, Handbook of Logic in Computer Science, 1992.
18. Janousek, J. String Suffix Automata and Subtree Pushdown Automata. In: *Proceedings of the Prague Stringology Conference 2009*, pp. 160–172, Czech Technical University in Prague, Prague, 2009.
19. Janousek, J.: *Arbology: Algorithms on Trees and Pushdown Automata*. Habilitation thesis, TU FIT, Brno, 2010.
20. Janousek, J., Melichar, B. On Regular Tree Languages and Deterministic Pushdown Automata. In *Acta Informatica*, Vol. 46, No. 7, pp. 533-547, Springer, 2009.

21. Melichar, B. Arbology: Trees and pushdown automata. In: *LATA 2010 (LNCS 6031)*, invited paper, pp. 32-49, Springer, 2010.
22. Melichar, B., Holub, J., Polcar, J., 2005. Text searching algorithms. Available at: http://stringology.org/athens/, release November 2005.
23. Gonzalo Navarro, Mathieu Raffinot: A Bit-Parallel Approach to Suffix Automata: Fast Extended String Matching. In: *CPM, LNCS 1448*, Springer, Berlin, pp. 14-33, 1998.
24. R. Ramesh, I. V. Ramakrishnan. *Nonlinear Pattern Matching in Trees*, Journal of the Association for Computing Machinery, Vol 39, No 2, April 1992.
25. Smyth, B., 2003. Computing Patterns in Strings. Addison-Wesley-Pearson Education Limited, Essex, England.
26. Travnicek, J. , Janousek, J., Melichar, B. Nonlinear Tree Pattern Pushdown Automata. In*Proceedings of the FEDCSIS 2011*, IEEE Computer Society Press, pp. 871-878, 2011.
27. R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. In: *Commun. ACM*, 35(10), pp. 7482, 1992.

Jan Trávníček has been a PhD student at the Department of Theoretical Computer Science, Czech Technical University in Prague, Faculty of Information Technology. His research interests are focused on algorithms on trees (Arbology).

Jan Janoušek has been an associate professor at the Department of Theoretical Computer Science, Czech Technical University in Prague, Faculty of Information Technology. His research interests include algorithms on trees (Arbology), parsing algorithms, compiler construction, attribute grammars and formal languages and automata theory.

Borivoj Melichar has been a full professor at the Department of Theoretical Computer Science, Czech Technical University in Prague, Faculty of Information Technology. His research interests include algorithms on strings (Stringology), algorithms on trees (Arbology), parsing algorithms and compiler construction.

Received: December 20, 2011; Accepted: June 4, 2012.