ISBN 978-83-60810-22-4

Indexing Trees by Pushdown Automata for Nonlinear Tree Pattern Matching

J. Trávníček, J. Janoušek, B. Melichar
Department of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00 Prague 6, Czech Republic
Email: travnja3@fit.cvut.cz, Jan.Janousek@fit.cvut.cz, melichar@fit.cvut.cz

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

***Abstract*—A new kind of an acyclic pushdown automaton for an ordered tree is presented. The *nonlinear tree pattern pushdown automaton* represents a complete index of the tree for nonlinear tree patterns and accepts all nonlinear tree patterns which match the tree. Given a tree with n nodes, the number of such nonlinear tree patterns is $\mathcal{O}((2+v)^n)$, where v is the number of variables in the patterns. We discuss time and space complexities of the nondeterministic nonlinear tree pattern pushdown automaton and a way of its implementation. The presented pushdown automaton is input–driven and therefore can be determinised.**

I. 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 containing variables which represent specific subtrees are called nonlinear tree patterns. 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 the problem of pattern matching. 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 [10]. The second basic approach is represented by the use of an indexing 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 [6, 7, 17, 19] for strings or subtree pushdown automaton [13], which represents a complete index of a tree for subtrees.

Trees can also be seen 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 follows a linear notation of the tree. [15] shows that the deterministic pushdown automaton (PDA) is an appropriate model of computation for labelled ordered 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 [9], which are accepted by finite tree automata.

This paper describes a new kind of acyclic pushdown automaton, *nonlinear tree pattern pushdown automaton*, which represents a complete index of the tree for nonlinear tree patterns and accepts all nonlinear tree patterns which match the tree. Given a tree with n nodes, the number of such nonlinear tree patterns is $\mathcal{O}((2+v)^n)$, where v is the number of variables in the patterns. We describe the construction of the nonlinear tree pattern pushdown automaton and discuss its time and space complexities. The presented nondeterministic pushdown automaton is input–driven and therefore can be determinised. The deterministic version would accept an input nonlinear tree pattern of size m in time linear in m and not depending on n, but its disadvantage would be its large space complexity.

The presented nonlinear tree pattern pushdown automaton is analogous to string nondeterministic factor automaton [17]. The pushdown symbol alphabet contains just one pushdown symbol and therefore the pushdown store can be implemented by a single integer counter. Therefore, efficient methods for implementing nondeterministic string factor automata, such as [8], can easily be used also for the implementation of nondeterministic nonlinear tree pattern pushdown automata.

We note that the presented PDAs 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 the tree indexing data structure is to accept a finite language, a finite automaton could also be constructed. However, this automaton would have significantly more states than the PDA, in which the underlying tree structure is processed by the pushdown store.

The paper is organised as follows. The second section discusses a related work describing a nonlinear tree pattern matching algorithm. The third section contains basic definitions. Three types of pushdown automata that indexes trees for nonlinear pattern matching are presented. The fourth section describes special case of pushdown automaton. The first type, described in the fifth section, is a basic pushdown automaton that represents the basic idea of indexing trees for nonlinear pattern matching with one variable. The second type, described in the sixth section, is an optimisation of the basic pushdown automaton. The optimised pushdown automaton called nonlinear tree pattern pushdown automaton is smaller but accepts the same language as the basic one. The subsequent section is devoted to the indexing for more than one variable in nonlinear tree patterns. The last section is a conclusion.

II. Related Work

Some algorithms for nonlinear tree pattern matching are known. Nonlinear tree pattern matching algorithm described in [18] uses the approach which is represented by the preprocessing of the nonlinear input tree pattern. The algorithm reads Euler 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 the Euler notation. Our method presented in this paper uses a standard tree prefix notation, which contains every node just once, for the first visit during the preorder traversing of the tree and of the input pattern.

In [18] factors which represent some subtrees in a subject tree in Euler notation are constructed. Aho-Corasick automaton 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. If symbol 1 is at position i in the binary array it means that the corresponding factor of the pattern string is a suffix of the prefix (to i-th symbol) of Euler notation of the subject tree. In this way the nonlinear variables are matched. In our method presented in this paper we construct a complete index of the subject tree for a given maximal number of variables and do not construct any additional matching automata.

III. Basic notions

We define notions on trees similarly as they are defined in [1, 9, 10].

A. 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.

B. Tree, tree pattern, tree template

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

A *directed graph* G is a pair (N, R), where N is a set of nodes and R is a set of 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 leaving f, entering node g.

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

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

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

A tree can be *directed*. A *rooted and directed tree* t is a dag $t = (N, R)$ with a special node $r \in N$, called the *root*, such that (1) r has in-degree 0, (2) all other nodes of t have in-degree 1, (3) there is just one path from the root r to every $f \in N$, where $f \neq r$.

A *labelled, (rooted, directed) tree* is a tree having the following property: (4) every node $f \in N$ is labelled by a symbol $a \in \mathcal{A}$, where $\mathcal{A}$ is an alphabet.

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:

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

The tree t_1 written 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.

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.

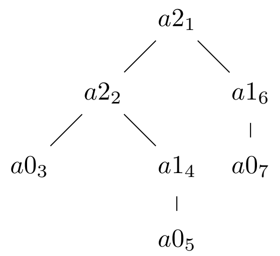

Fig. 1: Tree t_1 from Example 1

To define a *tree pattern*, we use a special nullary symbol S, not in alphabet $\mathcal{A}$, $Arity(S) = 0$, which 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* uses nullary symbols $X, Y, \ldots$ which are not in alphabet $\mathcal{A}$. These symbols have arity equal to zero. These symbols serve as a placeholders for any subtree. Additionally every occurrence of for example symbol X in *nonlinear tree pattern* is matched with the same subject subtree. A nonlinear tree pattern has to contain at least one symbol from $\mathcal{A}$. A nonlinear tree pattern which contains at least two symbols X will be called *nonlinear tree template*. Symbol X is called a *nonlinear variable*.

A nonlinear tree pattern np with $k \geq 2$ occurrences of the symbol X *matches* an subject tree t at node n if there exist the same subtrees $t_1, t_2, \ldots, t_k$ of the tree t such that the tree np', obtained from np by substituting the subtree t_i for the i-th occurrence of X in np, $i = 1, 2, \ldots, k$, 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 template p_1 over $\mathcal{A} \cup \{S\}$,
$p_1 = (\{a2_1, S_2, a1_3, S_4\}, R_{p1})$, where R_{p1} is a set of lists of the following ordered pairs:

$$R_{p1} = \{(a2_1, S_2), (a2_1, a1_3), (a1_3, S_4)\}.$$

The tree template p_1 written in prefix notation is $pref(p_1) = a2\ S\ a1\ S$.

Consider a nonlinear tree template p_2 over $\mathcal{A} \cup \{S, X\}$,
$p_2 = (\{a2_1, X_2,\ a1_3, X_4\},\ R_{p3})$, where R_{p2} is a set of lists of the following ordered pairs:

$$R_{p2} = \{(a2_1, X_2), (a2_1, a1_3), (a1_3, X_4)\}.$$

Note that symbol S can occur in nonlinear tree template and it serves as unbounded variable.

The tree template p_2 written in prefix notation is $pref(p_2) = a2\ X\ a1\ X$.

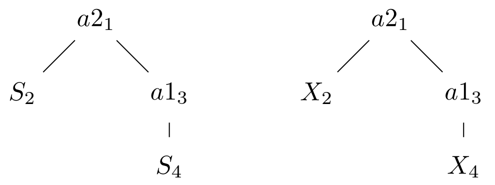

Fig. 2: Tree template p_1 (left) and nonlinear tree template p_2 (right) from Example 2

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

C. Language, finite and pushdown automata

We define notions from the theory of string languages similarly as they are defined in [1, 11].

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 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

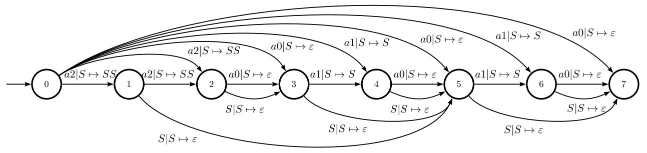
0
1
2
3
4
5
6
7
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$

Fig. 3: Transition diagram of nondeterministic tree pattern pushdown automaton M_{npt} $(pref(t_1))$ from Example 3

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 $\delta(q, x_1, \gamma_1) \vdash^+_M (q, x_2, \gamma_2)$, where $xx_2 = x_1$, $x \neq \varepsilon$ and $q \in Q$.

IV. Indexing trees for tree pattern matching

Tree pattern pushdown automata are introduced in [14, 16] as an extension of subtree PDA. The tree pattern pushdown automaton represents a complete index of a tree for linear tree patterns and accepts all tree patterns that match the tree.

Example 3 Consider a 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 tree pattern pushdown automaton accepting all tree patterns matching tree t_1 is nondeterministic pushdown automaton $M_{npt}(pref(t_1)) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_5, 0, S, \emptyset))$. Its transition diagram is illustrated in Figure 3.

V. Indexing trees by basic nonlinear tree pattern pushdown automaton

A nondeterministic basic nonlinear tree pattern pushdown automaton $M_b = (\{0,\ 1,\ 2,\ \ldots,\ n,\ x_1,\ \ldots,\ n_1,\ y_2,\ \ldots,\ n_2, \ldots, z_m, \ldots, n_m\}, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \emptyset)$ accepts all nonlinear tree patterns which can occur in a subject tree.

A. Construction of basic indexing automaton for nonlinear tree pattern matching

In our indexing pushdown automata we construct special parts called tails, which represent parts accessible after reading an input symbol of a nonlinear variable. Such a symbol selects a particular tail.

The $tail(M, q_t, Z_t)$ of an automaton $M = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$, where M is an acyclic tree pattern pushdown automaton, is defined as $tail(M, q_t, Z_t) = (Q_t, \mathcal{A}, G, \delta_t, q_t, Z_t,\ F)$. $Q_t = Q \setminus Q_{us}$, Q_{us} is a set of unreachable states from q_t when pushdown store operations are omitted, $q_t \in Q_t$ is a new initial state of an automaton, $\delta_t = \delta \setminus \delta_{us}$, δ_{us} are transitions leading from or to state $q_n \in Q_{us}$.

Example 4 Given a tree pattern pushdown automaton M_{npt} $(pref(t_1))$, which is an index of tree t_1 from Example 1 shown in Figure 3. The tail of automaton with initial state 3 is $tail(M_{npt}, 3, S) = (Q, \mathcal{A} \cup \{S\}, \{S\}, \delta, 3, S, \emptyset)$ constructed from tree pattern pushdown automaton shown in Figure 3, S is the initial symbol of the pushdown store. The corresponding transition diagram is illustrated in Figure 4.

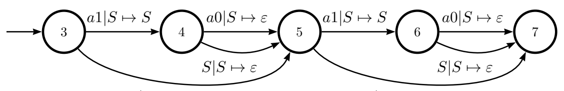
3
4
5
6
7
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$

Fig. 4: Tail of tree pattern pushdown automaton $M_{tail}(M_{npt}, 3, \varepsilon)$ from Example 4

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

A labelled path in the automaton M_{npt} between states q and q_t, where q_t is defined by transition $(q_t, \varepsilon) \in \delta(q, S, S)$ will be denoted $sst(q) = b_1 b_2 \ldots b_m$ and it represents a subtree in the linear notation. $sst(q)$ is used in Algorithm 2 to determine which subtree of subject tree was "assigned" to particular automaton tail.

The construction consists of two algorithms. Algorithm 2 constructs tails from the original tree pattern pushdown automaton. Algorithm 1 connects recursively these created tails to the automaton being created.

Algorithm 1 Construction of nondeterministic basic nonlinear tree pattern pushdown automaton.
Input: Nondeterministic tree pattern pushdown automaton M_{npt}.
Output: Nondeterministic basic nonlinear tree pattern pushdown automaton M_b.
Method:

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

Algorithm 2 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 $M_{tmp} = nta(tail(M_{tnpt}, q_t, S), x)$ using Algorithm 2.
    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 $M_{tnpt}$.
```

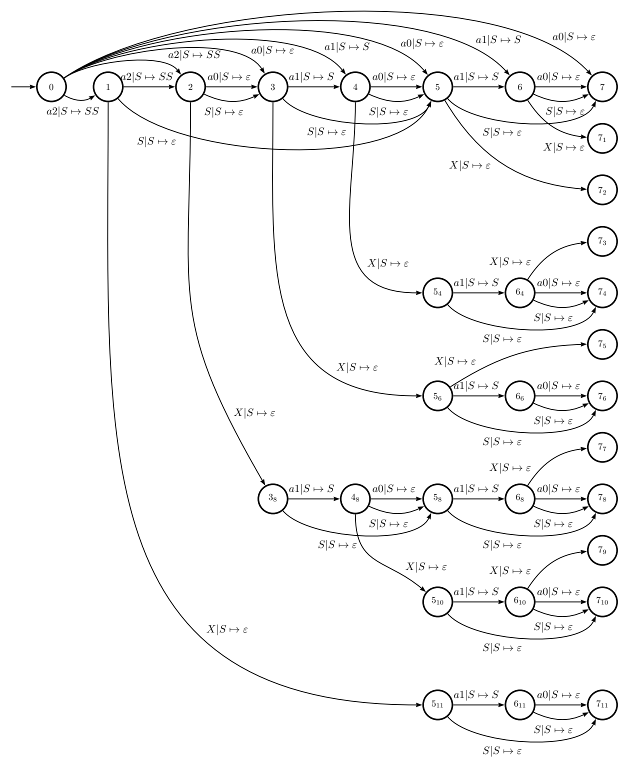

Fig. 5: Nondeterministic basic nonlinear tree pattern pushdown automaton $M_b(t_1)$ from Example 5 constructed for subject tree shown in Figure 1

```
    1.4. Add $M_{tmp}$ to $M_{tnpt}$ and merge initial state of $M_{tmp}$ with $q_{id}$.
2. $nta(M_{tnpt}, x)$ is $M_{tnpt}$.
```

The difference between Algorithm 2 and Algorithm 1 is that Algorithm 2 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 1 calls Algorithm 2 for each transition for symbol S.

Example 5 Given a string $p = a2\ a2\ a0\ a1\ a0\ a1\ a0$, 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 5.

VI. Indexing trees by nonlinear tree pattern pushdown automaton

Some states of an automaton created by Algorithm 1 may be merged so that states in nondeterministic nonlinear tree pattern pushdown automaton $M_o = (\{0, 1, 2, \dots, n, x_1, \dots, n_1, y_2, \dots, n_2, \dots, z_m, \dots, n_m\}, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \varnothing)$ will still track both virtually assigned subtree and the same number of nonlinear variables read from the pattern. Merged states are those from tails with the same virtually assigned subtree and the same number of nonlinear variables read.

A. Constructing indexing automaton for nonlinear tree pattern matching

Definition 1 A *tree node state label* $tnsl(q)$ is the sequence number of tree node of subject tree written in prefix notation. The $tnsl(q)$ is equivalent to automaton state label, where $q \in Q$ from automaton $M_b = (Q, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, q_0, S, \varnothing)$. The $tnsl(q)$ is the main number from state label.

Algorithm 3 Algorithm for counting the $tnsl$.
Input: Nondeterministic basic nonlinear tree pattern pushdown automaton M_b, state q for which count the $tnsl$.
Output: Number representing $tnsl$.
Variables: Temporary number n, State $initial$.
Method:

```
1. $n$ = 0. $initial$ is initial state of automaton $M_b$.
2. While $q \neq initial$ do:
    2.1. If exists transition $(q, S^{arity(a)}) \in \delta(q_{prev}, a, S)$ where $a \in \mathcal{A}$ and $q_{pref}$ is preferably not $q_0$ do:
        2.1.1. $n = n + 1$, $q = q_{prev}$.
        2.1.2. Continue 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. $n = n + |sst(q_{prev})|$, $q = q_{prev}$.
        2.2.2. Continue with step [2.].
```

Example 6 Given a nonlinear nondeterministic basic tree pattern pushdown automaton for pattern matching is $M_b(t_1) = (Q, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \varnothing)$, having its transition diagram shown in Figure 5.

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

Definition 2 A *number of nonlinear variables* $nnv(q, X)$ is the number of transitions for nonlinear variable X on the path from the initial state q_0 to state q, where q and $q_0 \in Q$ of a nondeterministic basic nonlinear tree pattern pushdown automaton $M_b = (Q, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \varnothing)$ created by Algorithm 1.

Algorithm 4 Algorithm for counting the nnv.
Input: Nondeterministic basic nonlinear tree pattern pushdown automaton M_b, state q for which count the $tnsl$.
Output: Number representing nnv.
Variables: Temporary number n, State $initial$.
Method:

```
1. $n$ = 0. $initial$ is initial state of automaton $M_b$.
2. While $q \neq initial$ do:
    2.1. If exists transition $(q, S^{arity(a)}) \in \delta(q_{prev}, a, S)$ where $a \in \mathcal{A}$ and $q_{pref}$ is preferably not $q_0$ do:
        2.1.1. $q = q_{prev}$.
        2.1.2. Continue 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. Continue with step [2.].
```

Definition 3 The *starting states of optimisation* $sso(M_b)$ is a collection of (key, $value$) pairs, where key is a triplet ($sst(q)$, $nnv(u, X)$, $tnsl(u)$) and $value$ is a set of states. The $sso(M_b)$ stores sets of states with the same number of transitions for nonlinear variable X $nnv(q, X)$ and subtree skipped by transition $sst(q)$, which denotes the sets of states from nondeterministic basic nonlinear tree pattern pushdown automaton M_b created by Algorithm 1. The $sso(M_b) = \{(sst(q_x), nnv(s_{a1}, X), tnsl(s_{a1})), \{s_{a1}, s_{a2}, \ldots\})$, $(sst(q_y)$, $nnv(s_{b1}, X), tnsl(s_{b2})), \{s_{b1}, s_{b2}, \ldots\}), \ldots\}$, 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 states $sso(M_b)$ defines states from nondeterministic basic nonlinear tree pattern pushdown automaton M_b that can be merged and the resulting automaton is called nondeterministic nonlinear tree pattern pushdown automaton M_o. 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 7 Given a string $p = a2\ a2\ a0\ a1\ a0\ a1\ a0$, 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 5.

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

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

Algorithm 5 Construction of the nondeterministic nonlinear tree pattern pushdown automaton.
Input: Nondeterministic basic nonlinear tree pattern pushdown automaton M_b.
Output: Nondeterministic nonlinear tree pattern pushdown automaton M_o.
Variables: Collection of sets of states $sso(M_b)$.
Method:

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

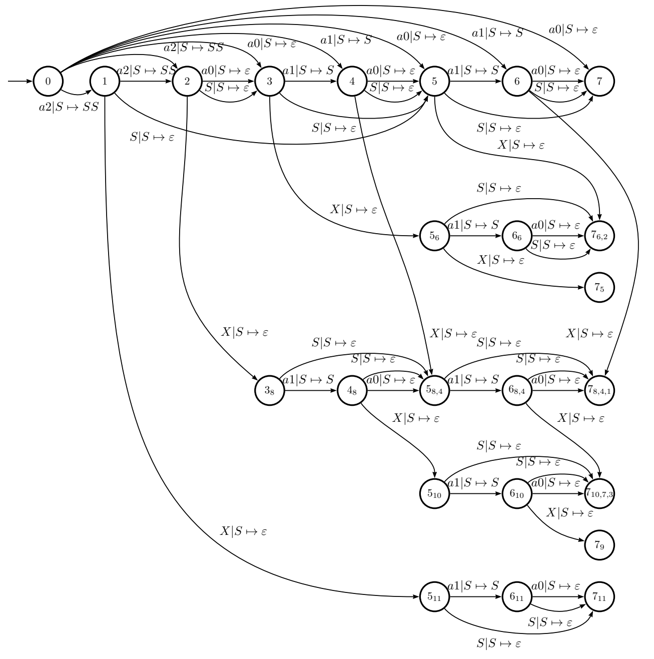

Fig. 6: Nondeterministic nonlinear tree pattern pushdown automaton $M_o(t_1)$ from Example 8 constructed by Algorithm 5 for subject tree shown in Figure 1

Example 8 Given a string $p = a2\ a2\ a0\ a1\ a0\ a1\ a0$, which is the prefix notation of tree t_1 from Example 1, the corresponding nondeterministic nonlinear tree pattern pushdown automaton is $M_o(t_1) = (Q, \mathcal{A} \cup \{S, X\}, \{S\}, \delta, 0, S, \varnothing)$, where merged states are in Example 7 and its transition diagram and states are illustrated in Figure 6.

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 $sso(M_b)$. A 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.

B. Time and Space Complexity Analysis

Lemma 1 Time complexity of accepting pattern by automaton created by Algorithm 5 is $\mathcal{O}(m)$, where m is the number of nodes of a subject tree.

Proof: Automaton created for nonlinear pattern matching reads just one symbol from the input by every transition. Automaton accepts or rejects the input pattern at the latest after reading the last symbol of pattern tree written in prefix notation. ∎

Lemma 2 Time complexity of accepting pattern by automaton created by Algorithm 5 is $\mathcal{O}(\sum_S rs_i)$, where S is the set of all prefixes except ε and rs_i is the number of distinct sequences of transitions in automaton M_o for $s_i \in S$ which ends in valid state.

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

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

Lemma 3 The number of states of nondeterministic nonlinear tree pattern pushdown automaton M_o (space complexity) created by Algorithm 5 is $\mathcal{O}(m(\sum_{i=0}^{s} r_i))$, where m is the number of nodes of a subject tree and $\sum_{i=0}^{s} r_i$, where s is the number of distinct subtrees, is the number of automaton tails, where r_i is the number of repetitions of each unique 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 space complexity of automaton for one nonlinear variable is $\mathcal{O}(m(\sum_{i=0}^{s} r_i + 1)) = \mathcal{O}(m(\sum_{i=0}^{s} r_i))$. ∎

Lemma 4 The number of transitions of nondeterministic nonlinear tree pattern pushdown automaton M_o (space complexity) created by Algorithm 5 is $\mathcal{O}(m^2 + m + \sum_{i=0}^{s}(\frac{r_i^2+r_i}{2}))$, where m 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: Given all tails for one nonlinear variable there are transitions for 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 for symbol X is $\sum_{i=0}^{s}(\frac{r_i^2+r_i}{2})$.

Using Lemma 3 the number of transitions for symbol S is $\frac{1}{2}m^2$ and the number of transitions for symbol $a \in \mathcal{A}$ is $\frac{1}{2}m^2 + m$.

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

Lemma 5 Language defined by nondeterministic nonlinear tree pattern pushdown automaton M_o is $\mathcal{O}(3^m)$, where m is the number of nodes of a subject tree.

Proof: Consider a tree with $m + 1$ nodes. Arity of the first node is m. Remaining nodes are labelled with the same nullary symbol. It is possible to create tree template where on each position of nullary symbol a nonlinear variable X, symbol S or the original symbol can be placed. Therefore on m positions it is possible to chose from three symbols. Language size is then $\mathcal{O}(3^m)$, where $m + 1$ is the number of nodes of a subject tree. ∎

VII. Construction of automata for patterns with more nonlinear variables

Patterns that contain more than one nonlinear variable are more common than those with one nonlinear variable. The algorithm for construction of nondeterministic nonlinear tree pattern pushdown automaton for more nonlinear variables M_{mo} or M_{mb} is basically algorithm for union of automata. Automaton for two nonlinear variables is union of two automata for one nonlinear variable.

A. An algorithm of joining automata

Definition 4 The *nonlinear variable from automaton $nva(M)$* is the nonlinear variable for which the nondeterministic nonlinear tree pattern pushdown automaton M_o was created for.

Consider that nondeterministic basic nonlinear tree pattern pushdown automaton M_{mo} for two nonlinear variables determined by X, Y is to be constructed. Nondeterministic nonlinear tree pattern pushdown automaton M_o for nonlinear variable determined by symbol X and second for nonlinear variable determined by symbol Y are constructed by Algorithm 5. The first automaton for nonlinear variable determined by symbol X handles nonlinear variable determined by symbol Y as linear variable usually determined by symbol S. The second automaton handles nonlinear variables similarly.

Algorithm 6 Construction of Indexing automaton for more nonlinear variables.
Input: Set of nondeterministic nonlinear tree pattern pushdown automata M_o.
Output: Nondeterministic nonlinear tree pattern pushdown automaton for more nonlinear variables $M_{mo} = (Q, \mathcal{A}, \{S\}, \delta', q_I, S, \varnothing)$.
Method:

```
1. Create set of symbols of nonlinear variables $nvs$ using $nva$ from input set of automata.
2. For $i$ = 0 to $sizeof(M_o)$, step 1 do:
    2.1. $M_{oi}$ is automaton from set $M_o$ on index $i$.
    2.2. For each transition $(u, \varepsilon) \in \delta(q, S, S)$ in automaton $M_{oi}$ do:
        2.2.1. For each symbol $s$ in $nvs \setminus nva(M_{oi})$ add transition $(u, \varepsilon) \in \delta(q, s, S)$ to automaton $M_{oi}$.
2. Create automaton $M_{mo}$ as union of all automata in input set.
```

B. Time and space complexity analysis

Lemma 6 The space complexity of a nondeterministic nonlinear pattern pushdown automaton for more nonlinear variables M_{mo} is $\mathcal{O}(t^v * m)$, where t is the number of tails of the nondeterministic nonlinear tree pattern pushdown automaton for one nonlinear variable M_o, v is the number of nonlinear variables and m is the number of nodes of a subject tree.

Proof: The automaton complexity is clear for one nonlinear variable. The number of tails in automaton for more nonlinear variables is result from the Cartesian product of tails in each automaton for one nonlinear variable. Each tail of the first automaton allows the second automaton to move across tails depending on the others nonlinear variables and so on for more automata. The number of tails is t^v for the Cartesian product, where t is the number of tails of original one nonlinear variable automaton. The number of tails is given by union of v automata for v nonlinear variables. So the exact space complexity of nondeterministic nonlinear tree pattern pushdown automaton M_{mo} for more nonlinear variables is $\mathcal{O}(t^v * m)$. ■

Lemma 7 The language accepted by nondeterministic nonlinear tree pattern pushdown automaton M_{mo} for more nonlinear variables contains $\mathcal{O}((2+v)^n)$ sentences, where n is the number of nodes of a subject tree.

Proof: Proof is constructed similarly as in Lemma 5. Consider a tree with $n+1$ nodes. Arity of the first node is n. Remaining nodes are labelled with the same nullary symbol. It is possible to create tree template where on each position of nullary symbol a nonlinear variable X, Y, ..., symbol S or the original symbol can be placed. Therefore, m positions is possible to choose from $2+v$ symbols. Language size is then $\mathcal{O}((2+v)^n)$, where $n+1$ is the number of nodes of a subject tree. ■

VIII. Conclusion

Algorithms creating pushdown automata for nonlinear tree indexing have been presented. Since these pushdown automata are input–driven, they can be determinised. It is shown that a nondeterministic nonlinear tree pattern pushdown automaton for one nonlinear variable has a space complexity polynomial to the size of the subject tree. The algorithm for constructing nondeterministic nonlinear tree pattern pushdown automaton for more nonlinear variables using the union of automata (Cartesian product of tails of automata) have also been presented.

The exact space complexity of the deterministic nonlinear indexing pushdown automata is an open question.

References

[1] Alfred V. Aho and Jeffrey 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, June 2011.

[3] 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.

[4] Crochemore, M., 1986. Transducers and repetitions. Theor. Comput. Sci. 45 (1), 63–86.

[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] 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.

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

[8] 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.

[9] 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.

[10] Christoph M. Hoffmann and Michael J. O'Donnell. Pattern matching in trees. *J. ACM*, 29(1):68–95, 1982.

[11] Hopcroft, J. E., Motwani, R., Ullman, J. D., 2001. Introduction to automata theory, languages, and computation, 2nd Edition. Addison-Wesley, Boston.

[12] J. W. Klop. *Term Rewriting Systems*, Handbook of Logic in Computer Science, 1992.

[13] 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.

[14] Janousek, J.: *Arbology: Algorithms on Trees and Pushdown Automata*. Habilitation thesis, TU FIT, Brno, 2010.

[15] Janousek, J., Melichar, B. On Regular Tree Languages and Deterministic Pushdown Automata. In *Acta Informatica*, Vol. 46, No. 7, pp. 533-547, Springer, 2009.

[16] Melichar, B. Arbology: Trees and pushdown automata. In: *LATA 2010 (LNCS 6031)*, invited speaker, pp. 32-49, Springer, 2010.

[17] Melichar, B., Holub, J., Polcar, J., 2005. Text searching algorithms. Available on: http://stringology.org/athens/, release November 2005.

[18] R. Ramesh, I. V. Ramakrishnan. *Nonlinear Pattern Matching in Trees*, Journal of the Association for Computing Machinery, Vol 39, No 2, April 1992.

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