VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ARBOLOGY: ALGORITHMS ON TREES AND PUSHDOWN AUTOMATA

HABILITAČNÍ PRÁCE
HABILITATION THESIS

AUTOR PRÁCE AUTHOR	Ing. JAN JANOUŠEK, Ph.D.

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ARBOLOGIE: STROMOVÉ ALGORITMY A ZÁSOBNÍKOVÉ AUTOMATY

ARBOLOGY: ALGORITHMS ON TREES AND PUSHDOWN AUTOMATA

HABILITAČNÍ PRÁCE
HABILITATION THESIS

AUTOR PRÁCE
AUTHOR

Ing. JAN JANOUŠEK, Ph.D.

BRNO 2010

Abstrakt

Habilitační práce prezentuje základní výsledky a principy arbologie, nové algoritmické disciplíny v oblasti zpracování stromů. Jako výpočetní model používá arbologie standartní zásobníkový automat, který čte lineární zápis stromu. Jednou z hlavních inspirací pro vytváření arbologických algoritmů je stringologie, která je algoritmickou disciplínou v oblasti zpracování textu. Práce sestává ze dvou částí. První čast práce obsahuje rozšířený strukturovaný abstrakt popisující hlavní arbologické výsledky, známé příbuzné výsledky a témata budoucího výzkumu. Prezentované výsledky jsou rozděleny do čtyř oblastí: formální stromové jazyky, indexování stromu, vyhledávání repetic ve stromu a vyhledávání vzorků ve stromu. Druhou část práce tvoří kolekce původních článků, které byly vytvořeny do konce roku 2009.

Abstract

This habilitation thesis presents basic results and principles of arbology, a new algorithmic discipline dealing with tree processing. As a model of computation arbology uses a standard pushdown automaton which reads a tree in a linear notation. One of the main inspirations for creating arbology algorithms is stringology, which is an algorithmic discipline dealing with string processing. The thesis consists of two parts. The first part contains an extended structured abstract describing main arbology results, related results, and topics for future research. The presented results are divided into the following four areas: formal tree languages, indexing a tree, computing repeats in a tree and tree pattern matching. The second part of the thesis is a collection of original papers, which were created by the end of 2009.

Klíčová slova

Strom, podstrom, stromový vzorek, vyhledávání vzorků ve stromech, indexování stromů, repetice ve stromech, zásobníkový automat, stromový automat, stromový jazyk.

Keywords

Tree, subtree, tree pattern, tree pattern matching, indexing tree, repeats in trees, pushdown automaton, tree automaton, tree language.

Citace

Jan Janoušek: Arbology: Algorithms on Trees and Pushdown Automata, habilitation thesis, Brno, FIT Brno University of Technology, 2010

Arbology: Algorithms on Trees and Pushdown Automata

Prohlášení

Prohlašuji, že jsem tuto habilitační práci vypracoval samostatně a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

....
Jan Janoušek
February 1, 2010

Poděkování

I would like to thank the most important people without whom this thesis would have never been created. I would like to thank my wife Kateřina and my parents, Jarmila and Ivo, for their love, which is one of the fundamentals of my life. I would like also to thank my friend and colleague Bořivoj Melichar – he taught me a lot and I highly appreciate the possibility of working with him during the last 14 years.

Dedication

To my wife Kateřina and my mum Jarmila

Preface

I started dealing with algorithms on trees in the second half of 2007 when I was introduced to the theory of tree automata. A bit later, together with my former Ph.D. supervisor professor Bořivoj Melichar, we started considering a systematic approach to the construction of algorithms for basic operations on trees. In 2008, together with Bořivoj Melichar and Ph.D. student Tomáš Flouri, we founded a new algorithmic discipline, which was given the name *arbology* from the Spanish word *arbol*, meaning *tree*. The most crucial decision to be done in the beginning was to select an appropriate model of computation. We have selected a standard pushdown automaton because of particular reasons described in the introduction of this thesis. Arbology as a new algorithmic discipline was officially first introduced at London Stringology Days 2009 conference. This habilitation thesis presents the basic arbology results and principles. Recently our arbology research group has been growing and topics of our current and future research work are also mentioned in this thesis.

Contents

Part I

Arbology – overview of basic results and principles

Chapter 1

Introduction

Trees are one of the fundamental data structures of Computer Science. They are able to express a hierarchical structure and are widely used in many applications.

The theory of formal string (or word) languages [3, 46, 72] and the theory of formal tree languages [18, 20, 29, 38, 37] have been extensively studied and developed since the 1950s and 1960s, respectively. Both the theories are important parts of the theory of formal languages [71]. The theory of formal string languages and its models of computation even represent the basic and the largest part of the theory of formal languages. Elements of string and tree languages are strings and trees, respectively. Models of computation of the theory of string languages are finite string automata, pushdown string automata, linear bounded automata and Turing machines, whereas models of computation of the theory of tree languages are various kinds of tree automata.

Trees can also be seen as strings, for example in their prefix (also called preorder) or postfix (also called postorder) notation. These linear notations can be respectively obtained by the prefix or postfix traversing of trees. Even, it can be said that every sequential algorithm processing a tree traverses through nodes of the tree in a linear order and so a corresponding linear notation of the tree is followed.

This habilitation thesis presents basic results and principles of *arbology* [9]. Arbology is a new algorithmic discipline focusing on algorithms on trees whose model of computation is a standard pushdown automaton which reads a linear notation of trees. We think that arbology is unique in the sense it represents the first–time systematic approach to solving tree problems by means of pushdown automata, although some particular tree algorithms based on pushdown automata, for example Graham-Glanville technique [41] used for code selection, are known. Pushdown automata seem to be an obvious and natural model of computation for algorithms on trees because of the following three facts:

1. Many algorithms for processing trees use recursive procedures, which means that the pushdown store is used for their implementation. For example, finite tree automata are implemented by recursive procedures.
2. Linear notations of trees are context-free languages and pushdown automata are the corresponding model of computation for context-free languages [3, 46, 72]. One of arbology results proves that the class of tree languages whose linear notation can be accepted by deterministic pushdown automata is a proper superclass of regular tree languages, which are recognized by finite tree automata.
3. The theory of finite automata has been successfully used in stringology [21, 22, 63, 76], which is an algorithmic discipline dealing with string processing. We think that

arbology deals with similar problems which can be solved in a similar fashion.

As is mentioned above the main inspiration for building arbology algorithms can be found in stringology [21, 22, 63, 76]. Stringology uses finite automata as a very useful tool. In arbology we try to apply the stringology principles to trees so that effective tree algorithms using pushdown automata would be created. In this way we have created new methods for the following basic operations on trees: indexing trees, computing repeats in trees, and tree pattern matching. These methods are analogous to methods of string indexing, of computing repeats in strings, and of string pattern matching, respectively.

There are some differences between finite and pushdown automata theories. For every nondeterministic finite automaton there exists an equivalent deterministic finite automaton which can be constructed using well known algorithm. This does not hold generally for the case of pushdown automata – for some nondeterministic pushdown automata their equivalent deterministic versions do not exist. Examples of such pushdown automata are pushdown automata accepting palindromes of the form like ww^R. The reason is that a deterministic automaton reading the palindrome from left to right is not able to find the centre of it. Generally, it is not known how to decide for a given nondeterministic pushdown automaton whether there exists a deterministic equivalent or not. Nevertheless, we have identified three classes of pushdown automata for which such a determinisation is possible. These classes are called input–driven [79], visible [7] and heigth–deterministic pushdown automata [66]. We note that many of particular pushdown automata presented in the subsequent chapters are input–driven pushdown automata.

The thesis consists of two parts. The first part of the thesis is an extended structured abstract of basic arbology results and principles. The second part of the thesis is a collection of original papers, which were created by the end of 2009. These papers describe the results in details.

The first part of the thesis consists of 11 chapters and is organised as follows. Chapter 2 discusses related existing results and algorithms and provides a comparison with arbology results presented in this thesis. Basic notions are defined in Chapter 3. Chapter 4 contains results introduced in [49], which deals with formal tree languages and pushdown automata and is the basic theoretical paper of arbology. Given a finite tree automaton it is proved that an equivalent LR(0) grammar can be constructed, which means that also an equivalent deterministic pushdown automaton can be constructed. Moreover, it is proved that deterministic pushdown automata are more powerful than finite tree automata – the class of tree languages whose linear notation can be accepted by deterministic pushdown automata is a proper superclass of regular tree languages. A new linear notation for unranked trees is described in Chapter 5. Chapter 6 describes general properties of linear notations of trees. These properties are substantial for constructing arbology algorithms. Chapters 8, 9, and 10 contain the descriptions of basic arbology algorithms in brief. An efficient method of indexing trees, of finding repeats in trees, and of tree pattern matching, respectively, are described. These methods were introduced in [48, 50], [51], and [32, 34], respectively.

Chapter 2

Related results and applications

Existing algorithms on trees are described by various formalisms, such as tree automata, term–rewriting systems, or pushdown automata, or they are directly described by programs written in appropriate programming languages. As stated in the previous chapter, we think that arbology represents the first–time systematic approach to the construction of algorithms on trees whose model of computation is a standard pushdown automaton.

Finite tree automata recognize regular tree languages and may be the most researched kind of tree automata [18, 20, 38]. As is shown in Chapter 4 ([49]), any problem which can be solved by a finite tree automaton can also be solved by a deterministic pushdown automaton. In [52, 67] it is shown that so–called pushdown tree–walking automata recognize exactly the class of regular tree languages. The underlying principle of a method of transformation of finite tree automata to pushdown tree–walking automata [67] is similar to the principle which is used in [49] for transformation of finite tree automata to deterministic pushdown automata.

Examples of other existing kinds of tree automata can be pushdown tree automata and generalised tree automata [20].

Comparing arbology algorithms with the known methods from the theory of tree automata [18, 20], the arbology methods of indexing trees and finding repeats in trees have not their equivalent known solutions by means of tree automata. Arbology tree pattern matching method provides directly equivalent solution as the known solution described by means of tree automata.

Formalisms for describing semantics are also used for tree processing: there exists a tool YakYak [53], which is a preprocessor for yacc–compatible generators and serves for generating parsers of the regular tree languages. The output of YakYak is not a syntax defining context–free grammar only, but it is an attributed context–free grammar in which the constraints defining regular tree languages are described not by the syntactic rules but by the semantic attribute rules (see also [4, 26] for the definition and for further information on attributed grammars). As a result, the parser generated by the YakYak + yacc–compatible generator behaves as a deterministic pushdown automaton which recognizes regular tree languages by an extended attribute semantic evaluation.

An example of a well researched problem whose solutions have been described by various models of computation is the code selection problem in compiler backends. The task here is to cover the intermediate program representation, which is in the form of a tree, by appropriate target machine code instructions, which are represented by tree patterns, and to select the “best possible” such covering. The best possible covering is usually selected according to the result of the evaluation of a cost function, which describes the cost of the machine code instructions. For the purpose of tree covering various versions of tree pattern matching are generally used (see [17, 43, 44, 54] for the basic tree pattern matching methods). A code selection method based on deterministic finite tree automata can be found in [30], where the cost function is computed by an additional semantic evaluation. On the other hand, [41, 57, 74] describe the code selection methods based on deterministic pushdown automata performing the tree pattern matching, where the tree patterns are represented by rules of a context–free grammar, and in this way generally ambiguous and non-LR(0) context–free grammars are created. Consequently, the LR(0) parsers for those grammars contain conflicts. In [41] these conflicts are resolved by some heuristics; in [57, 74] a special construction of a deterministic parser is used, which corresponds to a determinisation of the above–mentioned LR(0) parser with the conflicts.

Another code selection method is provided by a family of tools BURG, IBURG, etc. (see [35, 36] for example), which use another model of computation, so–called tree rewriting systems, for the tree pattern matching in the code selection problem.

We note that another code selection method based on the deterministic pushdown automaton would result from the transformation of the deterministic finite tree automaton from [30] in the way described in this thesis (where the evaluation of the cost function would be implemented by an attribute semantic evaluation). In addition, the transformation gives an unambiguous LR(0) grammar, which means the resulting code generator could be implemented easily with the use of an existing (yacc–like) parser generator for that grammar.

Models of computations for various linearised forms of unranked trees and their relationships to regular tree languages have been studied in some papers: for example, so–called nested words and visibly pushdown languages are studied in [6] and [7], respectively.

Chapter 3

Basic notions

We define notions on trees similarly as they are defined in [3, 18, 20, 37, 44].

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 the 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, tree template

Based on concepts from graph theory (see [3]), a tree over an alphabet $\mathcal{A}$ can be defined as follows:

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

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

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

A *tree* is an acyclic connected graph. Any node of a tree can be selected as a *root* of the tree. A tree with a root is called *rooted tree.*

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,

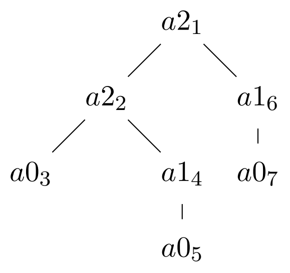


Figure 3.1: Tree t_1 from Example 3.1

(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 $a \in \mathcal{A}$ is $arity(a)$. Nodes labelled by nullary symbols (constants) are called *leaves.*

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

Example 3.1. Consider a ranked alphabet $\mathcal{A} = \{a2, a1, a0\}$. Consider a tree t_1 over $\mathcal{A}$ $t_1 = (\{a2_1, a2_2, a0_3, a1_4, a0_5, a1_6, a0_7\}, R)$, where R is a set of the following ordered sequences of pairs:

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

Trees can be represented graphically, and tree t_1 is illustrated in Fig. 3.1. □

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

To define a *tree pattern*, we use a special nullary symbol S, not in $\mathcal{A}$, which serves as a placeholder for any subtree. A tree pattern is defined as a labelled ordered ranked tree over ranked alphabet $\mathcal{A} \cup \{S\}$. By analogy, a tree pattern in prefix notation is defined as a labelled ordered ranked tree over ranked alphabet $\mathcal{A} \cup \{S\}$ in prefix notation. We will assume that the tree pattern contains at least one node labelled by a symbol from $\mathcal{A}$, i.e. sole S is not allowed to be a tree pattern. A tree pattern containing at least one symbol S will be called a *tree template.*

A tree pattern p with $k \geq 0$ occurrences of the symbol S *matches* an object tree t at node n if there exist subtrees $t_1, t_2, \dots, t_k$ (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, $i = 1, 2, \dots, k$, is equal to the subtree of t rooted at n.

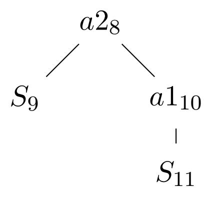


Figure 3.2: Tree pattern (template) p_1 from Examples 3.2

Example 3.2. Consider tree $t_1 = (\{a2_1, a2_2, a0_3, a1_4, a0_5, a1_6, a0_7\}, R)$ from Example 3.1, which is illustrated in Fig. 3.1. Consider a tree pattern (template) p_1 over $\mathcal{A} \cup \{S\}$ $p_1 = (\{a2_8, S_9, a1_{10}, S_{11}\}, R')$, where R' is a set of the following ordered sequences of pairs:

$$((a2_8, S_9), (a2_8, a1_{10})),$$
$$((a1_9, S_{11}))$$

Tree pattern p_1 is illustrated in Fig. 3.2. Tree pattern p_1 has two occurrences in tree t_1 – it matches at nodes $a2_1$ and $a2_2$ of t_1. □

3.3 Trees in prefix and postfix notation, ground terms

Prefix and postfix notations of trees belong to the most popular and used, one–visit, linear notations of trees.

Definition 3.3. The *prefix notation* $pref(t)$ of a tree t is defined in this way:

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

We note that in many papers on the theory of tree languages, such as [18, 20, 37, 44], labelled ordered ranked trees are defined with the use of ordered ranked *ground terms*. Ground terms can be regarded as labelled ordered ranked trees in prefix notation. Therefore, the notions ground term, tree and tree in prefix notation are often used interchangeably in many papers.

Definition 3.4. The *postfix notation* $post(t)$ of a tree t is defined in this way:

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

Example 3.5. Consider a ranked alphabet $\mathcal{A} = \{a2, a1, a0\}$. Consider a tree t_1 over $\mathcal{A}$ from Example 3.1, which is illustrated in Fig. 3.1. Prefix and postfix notations of tree t_1 are strings $pref(t_1) = a2 \; a2 \; a0 \; a1 \; a0 \; a1 \; a0$ and $post(t_1) = a0 \; a0 \; a1 \; a2 \; a0 \; a1 \; a2$, respectively. □

3.4 Finite tree automata, regular tree languages

A *nondeterministic finite (bottom–up) tree automaton* (nondeterministic FTA) over a ranked alphabet $\mathcal{A}$ is a 4–tuple $\mu = (Q, \mathcal{A}, Q_f, \Delta)$, where Q is a finite set of states, $Q_f \subseteq Q$ is the set of final states, and Δ is a set of transition rules of the following type:

$$f(q_1, q_2, \ldots, q_n) \to q,$$

where $f \in \mathcal{A}_n$, $n \geq 0$, and $q, q_1, \ldots, q_n \in Q$.

If there are no two rules with the same left–hand side, the tree automaton is called a *deterministic finite tree automaton* (deterministic FTA).

Example 3.6. A simple example of a deterministic finite tree automaton over an alphabet containing constants b and c, and binary symbol a is finite tree automaton $\mu_1 = (Q, \mathcal{A}, Q_f, \Delta)$, where $Q = \{1, 2, 3\}$, $\mathcal{A} = \{a2, b0, c0\}$, $Q_f = \{3\}$, and Δ contains these transition rules:

$$\begin{array}{rcl} b0 & \to & 1 \\ c0 & \to & 2 \\ a2(1,1) & \to & 3 \\ a2(1,2) & \to & 3 \end{array}$$

□

Finite tree automata over a ranked alphabet $\mathcal{A}$ run on ground terms over $\mathcal{A}$. Finite tree automaton starts at the leaves and moves upward, associating along a run a state with each subterm inductively. A *run* of an automaton on a ground term is defined as follows: The leaves are mapped to states q by the initial transition rules of the form $a \to q$, where $a \in \mathcal{A}_0$. Now, given a node labelled with $f \in \mathcal{A}_n$, $n \geq 1$, suppose its children have been mapped into states $q_1, \ldots, q_n$, where $f(q_1, q_2, \ldots, q_n) \to q$, then this node gets mapped to q.

A ground term is *accepted* by a finite tree automaton if there exists a run on the ground term such that its root is mapped to a final state.

The tree language $L(\mu)$ *recognized* by a finite tree automaton μ is the set of all ground terms accepted by the finite tree automaton μ. Two tree automata are *equivalent* if they recognize the same tree language. A tree language is *recognizable* if it is recognized by some nondeterministic finite tree automaton. A tree language is recognisable if and only if it is a regular tree language (see [18, 20, 37] for the definition of regular tree languages). Furthermore, it holds that each nondeterministic (bottom–up) finite tree automaton can be transformed to an equivalent deterministic (bottom–up) finite tree automaton.

Example 3.7. *Finite tree automaton μ_1 from Example 3.6 recognizes tree language $L(\mu_1) = \{a2\ b0\ b0,\ a2\ b0\ c0\}$. Ground term $t_2 = a2\ b0\ c0$ and the run of finite tree automaton μ_1 on ground term t_2 are illustrated in Fig. 3.3.* □

We note that there exist also *nondeterministic top–down finite tree automata* and *deterministic top–down finite tree automata.* The class of tree languages recognized by nondeterministic top–down finite tree automata is exactly the class of regular tree languages. However, it is not possible to transform every nondeterministic top–down finite tree automaton to an equivalent deterministic top–down finite tree automaton, which is a strictly less powerful model.

For more details on finite tree automata and regular tree languages, see [18, 20, 37].

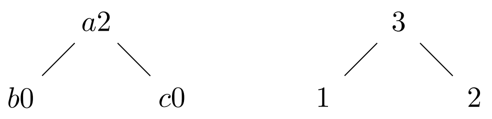


Figure 3.3: Tree t_2 (left) and the run of finite tree automaton μ_1 from Example 3.6 on tree t_2 (right)

3.5 Language, grammar, finite and pushdown automata

We define notions from the theory of string languages similarly as they are defined in [3, 46].

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 *context-free grammar* (CFG) is a 4-tuple $G = (N, \mathcal{A}, P, S)$, where N and $\mathcal{A}$ are finite disjoint sets of *nonterminal* and *terminal (input) symbols*, respectively. P is a finite set of *rules* $A \to \alpha$, where $A \in N$, $\alpha \in (N \cup \mathcal{A})^*$. $S \in N$ is the *start symbol.* Relation $\Rightarrow$ is called *derivation*: if $\alpha A\ \gamma \Rightarrow \alpha\beta\gamma$, $A \in N$, and α, β, $\gamma \in (N \cup \mathcal{A})^*$, then rule $A \to \beta$ is in P. Symbols $\Rightarrow^+$, and $\Rightarrow^*$ are used for the *transitive*, and the *transitive and reflexive* closure of $\Rightarrow$, respectively. The language generated by a G, denoted by $L(G)$, is the set of strings $L(G) = \{w : S\ \Rightarrow^* w,\ w \in \mathcal{A}^*\}$.

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

Every NFA can be transformed to an equivalent DFA [3, 46]. The transformation constructs the states of the DFA as subsets of states of the NFA and selects only such accessible states (ie subsets). These subsets are called *d–subsets.* In spite of the fact that d–subsets are standard sets, they are often written in square brackets ([ ]) instead of in braces ({ }).

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 $w \in \mathcal{A}^*$.

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

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

A pushdown automaton M is a *deterministic* pushdown automaton (deterministic PDA), if it holds:

1. $|\delta(q, a, Z)| \leq 1$ for all $q \in Q$, $a \in \mathcal{A}$, $Z \in inG$ and $\delta(q, \varepsilon, Z) = \emptyset$ or
2. $\delta(q, a, Z) = \emptyset$ for all $a \in \mathcal{A}$ and $|\delta(q, \varepsilon, Z)| \leq 1$.

An extended pushdown automaton M is an *deterministic* extended pushdown automaton (deterministic PDA), if it holds:

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

A pushdown automaton is *input–driven* if each of its pushdown operations is determined only by the input symbol.
A language L accepted by a pushdown automaton M is defined in two distinct ways:

1. *Accepting by final state:*
$$L(M) = \{x : \delta(q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \gamma) \wedge x \in \mathcal{A}^* \wedge \gamma \in G^* \wedge q \in F\}.$$
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 pushdown automaton accepts the language by empty pushdown store, then the set F of final states is the empty set.

For more details see [3, 46].

3.6 LR(0) parsing

Given a string w, an *LR(0) parser* for a context–free grammar $G = (N, T, P, S)$ reads the string w from left to right without any backtracking and is implemented by a deterministic pushdown automaton.

A string γ is a *viable prefix* of G if γ is a prefix of $\alpha\beta$, and $S \Rightarrow_{\text{rm}}^* \alpha A x \Rightarrow_{\text{rm}} \alpha\beta x$ is a rightmost derivation in G; the string β is called the *handle*. We use the term *complete viable prefix* to refer to $\alpha\beta$ in its entirety. During parsing, each contents of the pushdown store correspond to a viable prefix.

The standard LR(0) parser performs two kinds of transitions:

1. When the contents of the pushdown store correspond to a viable prefix containing an incomplete handle, the parser performs a *shift*, which reads one symbol a and pushes a symbol corresponding to a onto the pushdown store.
2. When the contents of the pushdown store corresponds to a viable prefix ending by the handle β, the parser performs a *reduction* by a rule $A \to \beta$. The reduction pops $|\beta|$ symbols from the top of the pushdown store and pushes a symbol corresponding to A onto the pushdown store.

A context–free grammar G is $LR(0)$ if the two conditions for G:
(1) $S \Rightarrow^*_{\mathrm{rm}} \alpha A w \Rightarrow_{\mathrm{rm}} \alpha\beta w$,
(2) $S \Rightarrow^*_{\mathrm{rm}} \gamma B x \Rightarrow_{\mathrm{rm}} \alpha\beta y$,
imply that $\alpha A y = \gamma B x$, that is, $\alpha = \gamma$, $A = B$, and $x = y$.

If the context–free grammar G is not an LR(0) grammar, then the pushdown automaton constructed as an LR(0) parser contains *conflicts*, which means the next transition to be performed cannot be determined according to the contents of the pushdown store only.

For context–free grammars without hidden–left and right recursions the number of consecutive reductions between the shifts of two adjacent symbols cannot be greater than a constant, and therefore the LR(0) parser for such a grammar can be optimized by precomputing all its reductions beforehand. Then, the optimized resulting LR(0) parser reads one symbol on each of its transition [10].

For more details on LR parsing, see [3, 2].

Chapter 4

Regular tree languages, finite tree automata and pushdown automata

The theory of formal string languages and of formal tree languages are both important parts of the theory of formal languages. Regular tree languages are recognized by finite tree automata. Trees in their postfix notation can be seen as strings. This chapter presents a simple transformation from any given (bottom–up) finite tree automaton recognizing a regular tree language to a deterministic pushdown automaton accepting the same tree language in postfix notation. The resulting deterministic pushdown automaton can be implemented easily by an existing parser generator because it is constructed for an LR(0) grammar, and its size directly corresponds to the size of the deterministic finite tree automaton. The class of regular tree languages in postfix notation is a proper subclass of deterministic context-free string languages. Moreover, the class of tree languages which are in their postfix notation deterministic context-free string languages is a proper superclass of the class of regular tree languages.

4.1 Transformation of a (bottom–up) finite tree automaton to an (extended) deterministic pushdown automaton

Definition 4.1. Let $\mu = (Q, \mathcal{A}, Q_f, \Delta)$ be a finite tree automaton. Then, a *context-free grammar generating $L(\mu)$ in postfix notation with appended right marker* $\dashv$ is context–free grammar $G_\mu = (N, T, P, S')$, where $N = \{S'\} \cup \{S_q : q \in Q\}$, $T = \mathcal{A}$, and $P = \{S' \to S_q \dashv : q \in Q_f\} \cup \{S_q \to S_{q_1} S_{q_2} \dots S_{q_n}\ a : a(q_1, q_2, \dots, q_n) \to q \in \Delta\}$.

Acceptance by the empty pushdown store is achieved in this way.

Definition 4.2. Let $\mu = (Q, \mathcal{A}, Q_f, \Delta)$ be a finite tree automaton. Let $G_\mu = (N, T, P, S')$ be the context–free grammar created according to Def. 4.1 for μ. Then, *pushdown automaton accepting $L(\mu)$ in postfix notation with appended right marker* $\dashv$ is pushdown automaton $M_\mu = (\{q\}, T, G, \delta, q, Z_0, \emptyset)$, where $T = \mathcal{A}$, $G = Q \cup \{Z_0, \dashv\}$, and $\delta = \{\delta(q, \dashv, Z_0\alpha) = (q, \varepsilon) : S' \to \alpha \dashv \in P\} \cup \{\delta(q, a, \alpha) = (q, A) : \ A \to \alpha a \in P, A \neq S'\}$.

The transformation in question is demonstrated on the following example:

Example 4.3. Consider tree language L_2 of trees representing logical expressions which are equal to the *true* value and can contain constants *true* and *false*, unary symbol *not*, and binary symbol *or*.

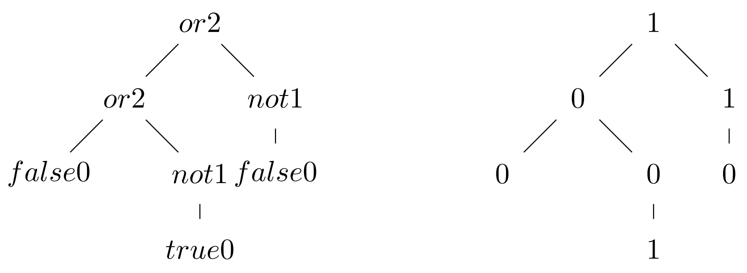


Figure 4.1: Tree $t_3 \in L_2$ (left) and the run of finite tree automaton μ_2 from Example 4.3 on tree t_3 (right)

Consider deterministic finite tree automaton $\mu_2 = (Q, \mathcal{A}, Q_f, \Delta)$, where $L(\mu_2) = L_2$, $Q = \{0, 1\}$, $\mathcal{A} = \{true0,\ false0,\ not1,\ or2\}$, $Q_f = \{1\}$, and Δ contains these rules:

$$\begin{array}{rcl}
false0 & \rightarrow & 0 \\
true0 & \rightarrow & 1 \\
not1(0) & \rightarrow & 1 \\
not1(1) & \rightarrow & 0 \\
or2(0,0) & \rightarrow & 0 \\
or2(0,1) & \rightarrow & 1 \\
or2(1,0) & \rightarrow & 1 \\
or2(1,1) & \rightarrow & 1
\end{array}$$

Fig. 4.1 shows a tree $t_3 \in L_2$ and the run of finite tree automaton μ_2 on tree t_3.

The context–free grammar created according to Def. 4.1 and generating $L(\mu_2)$ in postfix notation is $G_{\mu 2} = (N, T, P, S')$, where $N = \{S', S_0, S_1\}$, $T = \{true0,\ false0,\ not1,\ or2, \dashv\}$, and P contains the following rules (the rules are written in the same order as their corresponding transition rules of deterministic finite tree automaton μ_2):

$$\begin{array}{lcl}
S' & \rightarrow & S_1 \dashv \\
S_0 & \rightarrow & false0 \\
S_1 & \rightarrow & true0 \\
S_1 & \rightarrow & S_0\ not1 \\
S_0 & \rightarrow & S_1\ not1 \\
S_0 & \rightarrow & S_0\ S_0\ or2 \\
S_1 & \rightarrow & S_0\ S_1\ or2 \\
S_1 & \rightarrow & S_1\ S_0\ or2 \\
S_1 & \rightarrow & S_1\ S_1\ or2
\end{array}$$

The pushdown automaton created according to Def. 4.2 for $G_{\mu 2}$ is deterministic pushdown automaton $M_{\mu 2} = (\{q\}, T, G, \delta, q, Z_0, \emptyset)$, where $T = \{true0,\ false0,\ not1,\ or2, \dashv\}$, $G = \{Z_0, S_0, S_1\}$, and δ contains the following transition rules:

State	Pushdown Store	Input
q	Z_0	$false0\ true0\ not1\ or2\ false0\ not1\ or2\ \dashv$
q	Z_0S_0	$true0\ not1\ or2\ false0\ not1\ or2\ \dashv$
q	$Z_0S_0S_1$	$not1\ or2\ false0\ not1\ or2\ \dashv$
q	$Z_0S_0S_0$	$or2\ false0\ not1\ or2\ \dashv$
q	Z_0S_0	$false0\ not1\ or2\ \dashv$
q	$Z_0S_0S_0$	$not1\ or2\ \dashv$
q	$Z_0S_0S_1$	$or2\ \dashv$
q	Z_0S_1	$\dashv$
q	ε	ε
accept		

Figure 4.2: Trace of deterministic pushdown automaton $M_{\mu 2}$ from Example 4.3

$$\begin{array}{rcl}
\delta(q, \dashv, Z_0S_1) & = & (q, \varepsilon) \\
\delta(q, false0, \varepsilon) & = & (q, S_0) \\
\delta(q, true0, \varepsilon) & = & (q, S_1) \\
\delta(q, not1, S_0) & = & (q, S_1) \\
\delta(q, not1, S_1) & = & (q, S_0) \\
\delta(q, or2, S_0S_0) & = & (q, S_0) \\
\delta(q, or2, S_0S_1) & = & (q, S_1) \\
\delta(q, or2, S_1S_0) & = & (q, S_1) \\
\delta(q, or2, S_1S_1) & = & (q, S_1)
\end{array}$$

Tree automaton μ_2 is deterministic. As a consequence, the right–hand side of every rule of grammar $G_{\mu 2}$ is also unique and, moreover, that right–hand side is not a suffix of the right–hand side of any other rule of grammar $G_{\mu 2}$, which means the grammar is LR(0).

Tree t_3 in postfix notation is string $false0\ true0\ not1\ or2\ false0\ not1\ or2$. Fig. 4.2 shows the sequence of transitions (trace) performed by deterministic pushdown automaton $M_{\mu 2}$ for tree t_3 in postfix notation with the appended right marker. □

Lemma 4.4. *Let $\mu = (Q, \mathcal{A}, Q_f, \Delta)$ be a finite tree automaton. Let $G_\mu = (N, T, P, S')$ be the context–free grammar created according to Def. 4.1 for μ. Then, the context–free grammar G_μ generates exactly the language $L(\mu)$ in postfix notation with appended right marker $\dashv$.*

Proof. In Part II of the thesis or in [49]. □

Theorem 4.5. *Let $\mu = (Q, \mathcal{A}, Q_f, \Delta)$ be a finite tree automaton. Let $G_\mu = (N, T, P, S')$ be the context–free grammar created according to Def. 4.1 for μ. Let $M_\mu = (\{q\}, T, G, \delta, q, Z_0, \emptyset)$ be the pushdown automaton created according to Def. 4.2 for G_μ. Then, the pushdown automaton M_μ accepts exactly the language $L(\mu)$ in postfix notation with appended right marker $\dashv$.*

Proof. In Part II of the thesis or in [49]. □

Theorem 4.6. *Let $\mu = (Q, \mathcal{A}, Q_f, \Delta)$ be a deterministic finite tree automaton. Let $G_\mu = (N, T, P, S')$ be the context–free grammar created according to Def. 4.1 for μ. Let $M_\mu = (\{q\}, T, G, \delta, q, Z_0, \emptyset)$ be the pushdown automaton created according to Def. 4.2 for G_μ.*

Then the context–free grammar G_μ is an LR(0) grammar and the pushdown automaton M_μ is deterministic.

Proof. In Part II of the thesis or in [49]. □

Corollary 4.7. *The class of regular tree languages in postfix notation is a proper subclass of deterministic context-free string languages.*

Proof. In Part II of the thesis or in [49]. □

The size of the constructed deterministic pushdown automaton directly corresponds to the size of the deterministic finite tree automaton.

4.2 Beyond the class of regular tree languages

It is demonstrated by the following example that the deterministic pushdown automaton is a model of computation which is powerful enough to accept also some non–regular tree languages in postfix notation.

Example 4.8. Given a ranked alphabet $\mathcal{A} = \{f2, g1, a0\}$, consider tree language $L_3 = \{f2\ g1^i\ a0\ g1^i\ a0 : i > 0\}$, which contains the symmetry between the two children of binary symbol $f2$. It is shown in the details in Example 1.2.1 in [20] that L_3 is not a regular tree language, which means it cannot be recognized by a finite tree automaton.

L_3 in postfix notation contains strings of the form $a0\ g1^i\ a0\ g1^i\ f2$, where $i > 0$, which forms a deterministic context-free language. For example, the following LR(0) grammar G_3 generates L_3 in postfix notation with appended right marker $\dashv$. Context–free grammar $G_3 = (N, T, P, S')$, where $N = \{S', A\}$, $T = \{f2, g1, a0, \dashv\}$, and P contains these rules:

$$\begin{array}{lcl} S' & \rightarrow & S\ \dashv \\ S & \rightarrow & a0\ A\ f2 \\ A & \rightarrow & g1\ A\ g1 \\ A & \rightarrow & g1\ a0\ g1 \end{array}$$

□

Corollary 4.9. *The class of tree languages which are in their postfix notation deterministic context-free string languages is a proper superclass of the class of regular tree languages.*

Proof. The corollary follows from Corollary 4.7 and Example 4.8. □

Chapter 5

Linear notations of unranked trees

Definitions of the basic prefix and postfix notations are very useful for ranked trees. If the tree is not ranked it is necessary to include information concerning the rank of every node. This can be done in two ways:

1. to represent a node in both notations as a pair $(a, arity(a))$,
2. to use another principles of linearisation based on incorporation of some special symbols.

The second approach can be illustrated by a bracketted notation in which each subtree is enclosed in the left and the close bracket. The bar notations are based on the following observations:

1. there is always the root of a subtree just behind the left bracket in prefix notation,
2. there is always the right bracket just behind the root of a subtree in postfix notation.

It follows from this observation that there is the left (right) bracket redundant in prefix (postfix) bracketted notation. The bar notations in both cases reduces the number of symbols in both linear bracketted notations. Instead of different symbols, left or right brackets, the symbol bar ($|$) can be used in both cases.

Definition 5.1. The *prefix bar notation* $pref\_bar(t)$ and *postfix bar notation* $post\_bar(t)$ of a tree t are defined in this way:

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

In arbology we use linear bar notatitions for approximate tree pattern matching [33, 77], where the following three operations on trees are considered:

1. Changing the label of a node.
2. Inserting a node to a tree.
3. Deleting a node from a tree.

The second and the third operation increases and decreases, respectively, the arity of a node.

Chapter 6

Properties of linear notations of trees

In this chapter we describe some general properties of linear notations of trees which are defined in the previous chapters. These properties are substantial for creating arbology algorithms.

6.1 Prefix notation

Example 6.1. Consider tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 3.1, which is illustrated in Fig. 3.1. Tree t_1 contains only subtrees shown in Fig. 6.1. □

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

Theorem 6.2. *Given a tree t and its prefix notation $pref(t)$, all subtrees of t in prefix notation are substrings of $pref(t)$.*

Proof. In Part II of the thesis or in [50, 34]. □

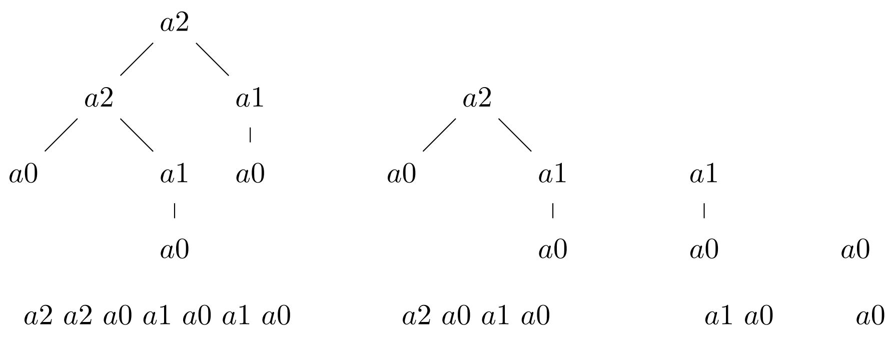


Figure 6.1: All subtrees of tree t_1 from Example 3.1, and their prefix notations

However, not every substring of a tree in prefix notation is a prefix notation of its subtree. Just those substrings which themselves are trees in prefix notation are those which are the subtrees in prefix notation. This property is formalised by the following definition and theorem.

Definition 6.3. Let $w = a_1a_2 \dots a_m$, $m \geq 1$, be a string over a ranked alphabet $\mathcal{A}$. Then, the *arity checksum* $ac(w) = arity(a_1) + arity(a_2) + \dots + arity(a_m) - m + 1 = \sum_{i=1}^{m} arity(a_i) - m + 1$.

Theorem 6.4. *Let $pref(t)$ and w be a tree t in prefix notation and a substring of $pref(t)$, respectively. Then, w is the prefix notation of a subtree of t, if and only if $ac(w) = 0$, and $ac(w_1) \geq 1$ for each w_1, where $w = w_1x$, $x \neq \varepsilon$.*

Proof. In Part II of the thesis or in [50, 34]. □

6.2 Postfix notation

In this section we describe the dual principle for the postfix notation. Theorems 6.5 and 6.6 present the direct analogy of properties of the prefix and postfix notations.

Theorem 6.5. *Given a tree t and its postfix notation $post(t)$, all subtrees of t in postfix notation are substrings of $post(t)$.*

Theorem 6.6. *Let $post(t)$ and w be a tree t in postfix notation and a substring of $post(t)$, respectively. Then, w is the postfix notation of a subtree of t, if and only if $ac(w) = 0$, and $ac(w_1) \leq -1$ for each w_1, where $w = xw_1$, $x \neq \varepsilon$.*

6.3 Linear notations of unranked trees

Similar properties hold also for prefix and postfix bar notations of trees.

Theorem 6.7. *Given a tree t and its prefix bar notation $pref\_bar(t)$, all subtrees of t in prefix bar notation are substrings of $pref\_bar(t)$.*

Theorem 6.8. *Given a tree t and its postfix bar notation $post\_bar(t)$, all subtrees of t in postfix bar notation are substrings of $post\_bar(t)$.*

Definition 6.9. Let $w = a_1a_2 \dots a_m$, $m \geq 1$, be a string over $\mathcal{A} \cup \{|\}$. Then, the *bar checksum* is defined as follows:

1. $bc(a) = 1$, and $bc(|) = -1$.
2. $bc(wa) = bc(w) + 1$, and $bc(w|) = bc(w) - 1$.

Theorem 6.10. *Let $pref\_bar(t)$ and w be a tree t in prefix bar notation and a substring of $pref\_bar(t)$, respectively. Then, w is the prefix bar notation of a subtree of t, if and only if $bc(w) = 0$, and $bc(w_1) \geq 1$ for each w_1, where $w = xw_1$, $x \neq \varepsilon$.*

The dual theorem for the postfix bar notation is as follows.

Theorem 6.11. *Let $post\_bar(t)$ and w be a tree t in postfix bar notation and a substring of $post\_bar(t)$, respectively. Then, w is the postfix bar notation of a subtree of t, if and only if $bc(w) = 0$, and $bc(w_1) \leq -1$ for each w_1, where $w = xw_1$, $x \neq \varepsilon$.*

6.4 Computing arity and bar checksums by pushdown automata

Pushdown automata presented in the next chapters compute arity or bar checksums by pushdown operations during the processing of trees. This computing of checksums is formally described by the following four theorems for prefix, postfix, prefix bar and postfix bar notations of trees.

Theorem 6.12. *Let $M = (\{Q, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$ be an input-driven PDA of which each transition from δ is of the form $\delta(q_1, a, S) = (q_2, S^i)$, where $i = arity(a)$. Then, if $(q_3, w, S) \vdash_M^+ (q_4, \varepsilon, S^j)$, where w is a tree in prefix notation, then $j = ac(w)$.*

Proof. In Part II of the thesis or in [50]. □

We note that such pushdown operations correspond to the pushdown operations of the standard top–down parsing algorithm for a context-free grammar with rules of the form

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

For principles of the standard top–down (LL) parsing algorithm see [3].

Theorem 6.13. *Let $M = (\{Q, \mathcal{A}, \{S\}, \delta, 0, S, F)$ be an input–driven PDA whose each transition from δ is of the form $\delta(q_1, a, S^i) = (q_2, S)$, where $i = arity(a)$. Then, if $(q_3, w, \varepsilon) \vdash_M^+ (q_4, \varepsilon, S^j)$, where w is a tree in postfix notation, then $j = -ac(w) + 1$.*

We note that such pushdown operations correspond to the pushdown operations of the standard bottom-up (LR) parsing algorithm for a context-free grammar with rules of the form

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

Theorem 6.14. *Let $M = (\{Q, \mathcal{A}, \{S\}, \delta, 0, S, F)$ be an input–driven PDA whose each transition from δ is of the form $\delta(q_1, a, \varepsilon) = (q_2, S)$ or $\delta(q_1, |, S) = (q_2, \varepsilon)$. Then, if $(q_3, w, \varepsilon) \vdash_M^+ (q_4, \varepsilon, S^j)$, where w is a tree in prefix bar notation, then $j = bc(w)$.*

Theorem 6.15. *Let $M = (\{Q, \mathcal{A}, \{S\}, \delta, 0, S, F)$ be an input–driven PDA whose each transition from δ is of the form $\delta(q_1, a, S) = (q_2, \varepsilon)$ or $\delta(q_1, |, \varepsilon) = (q_2, S)$. Then, if $(q_3, w, \varepsilon) \vdash_M^+ (q_4, \varepsilon, S^j)$, where w is a tree in postfix bar notation, then $j = -bc(w)$.*

Chapter 7

On determinisation of pushdown automata

It is well known that in the theory of finite automata there exists the algorithm of transformation of any nondeterministic finite automaton to an equivalent deterministic one. The determinisation of a finite automaton contains a creation of sets of states of a nondeterministic automaton. These subsets play the role of states of an equivalent deterministic finite automaton. The number of states of resulting deterministic finite automaton is less or equal to 2^n, where n is the number of states of the original nondeterministic finite automaton.

Such a universal algorithm for the determinisation of pushdown automata does not exist. We identified three classes of nondeterministic pushdown automata for which exist algorithms for determinisation. They are called input–driven [79], visible [7] and heigth–deterministic pushdown automata [66]. Algorithms for determinisation are different for these classes.

The principle of determinisation of finite automata can be used for input–driven pushdown automata. A notion of pushdown operation will be frequently used in the following meaning.

Definition 7.1. Let $M = (Q, A, G, \delta, q_0, Z_0, F)$ be a pushdown automaton. Let $\delta(q, a, \alpha)$ contains pair (p, β) for $p, q \in Q$, $a \in A \cup \varepsilon$, $\alpha, \beta \in G^*$. Then the notation $\alpha \mapsto \beta$ is used for operation popping α from the top of the pushdown store and pushing β to the top of the pushdown store. This operation is called *pushdown operation.*

Input–driven pushdown automata can be defined formally as follows.

Definition 7.2. A pushdown automaton $M = (Q, A, G, \delta, q_0, Z_0, F)$ is an input–driven pushdown automaton if each pushdown operation $\alpha \mapsto \beta$ during every transition is explicitly determined by the input symbol. In more formal notation: For each $Q \in Q$ and $a \in \mathcal{A} \cup \{\varepsilon\}$ there exists the only mapping $\delta(q, a, \alpha) = \{(p_1, \beta), (p_2, \beta), ..., (p_m, \beta)\}$ for one pair $\alpha, \beta \in G^*$ and $p_1, p_2, \ldots, p_m \in Q$.

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

Algorithm 7.3. Transformation of an input–driven nondeterministic PDA to an equivalent deterministic PDA.
Input: Input–driven nondeterministic PDA $M_{nx}(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$, where the ordering of its states is such that if $\delta(p, a, \alpha) = (q, \beta)$, then $p \leq q$.
Output: Equivalent deterministic PDA $M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', q_I, S, \emptyset)$.
Method:

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

Theorem 7.4. *Given a input–driven nondeterministic PDA $M_{nx}(t) = (Q, \mathcal{A}, \{S\}, \delta, q_0, S, \emptyset)$, the deterministic PDA $M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', \{q_0\}, S, \emptyset)$ constructed by Alg. 7.3 is equivalent to PDA $M_{nx}(t)$.*

Proof. In Part II of the thesis or in [50]. □

Chapter 8

Indexing trees

Two new kinds of acyclic pushdown automata for trees in prefix notation are presented in this chapter. First, *subtree pushdown automata* accept all subtrees of the tree. Second, *tree pattern pushdown automata* accept all tree patterns which match the tree. The presented pushdown automata are input–driven and therefore can be determinised. Given a tree with n nodes, the deterministic subtree and the deterministic tree pattern pushdown automaton represent a complete index of the tree, and the search phase of all occurrences of a subtree or a tree pattern, respectively, of size m is performed in time linear in m and not depending on n. This is faster than the time of the existing tree pattern matching algorithms, which depends on n. The total size of the deterministic subtree pushdown automaton is linear in n. Although the number of distinct tree patterns which match the tree can be exponential in n, for specific cases of trees the total size of the deterministic tree pattern pushdown automaton is linear in n.

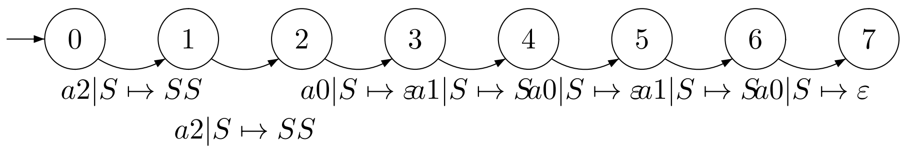
8.1 Subtree pushdown automata

From the global point of view, comparing the subtree pushdown automata with the string suffix automaton [14, 24], the deterministic subtree pushdown automaton constructed for a tree t can have just states and transitions which correspond to states and transitions of the deterministic string suffix automaton constructed for $pref(t)$, where the transitions of the subtree pushdown automaton are extended with pushdown operations. The pushdown operations compute the arity checksum. Moreover, some of the states and the transitions of the string suffix automaton need not be present in the deterministic subtree pushdown automaton because the corresponding pushdown operations cannot be performed and therefore such states are not accessible.

Definition 8.1. *Let t and $pref(t)$ be a tree and its prefix notation, respectively. A* subtree pushdown automaton *for $pref(t)$ accepts all subtrees of t in prefix notation.*

First, we start with a PDA which accepts the whole subject tree in prefix notation by empty pushdown store, whose construction is described by Alg. 8.2. The constructed PDA is deterministic.

Algorithm 8.2. Construction of a PDA accepting $pref(t)$ by empty pushdown store.
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation $pref(t) = a_1a_2 \ldots a_n$, $n \geq 1$.
Output: PDA $M_p(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$.
Method:



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

Figure 8.1: Transition diagram of deterministic pushdown automaton $M_p(t_1)$ accepting tree t_1 in prefix notation $\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 8.3

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

Figure 8.2: Trace of deterministic pushdown automaton $M_p(t_1)$ from Example 8.3 for tree t_1 in prefix notation $\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

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

Example 8.3. A pushdown automaton accepting tree t_1 in prefix notation $\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 3.1, which has been constructed by Alg. 8.2, is deterministic pushdown automaton $M_p(t_1) = (\{0,1,2,3,4,5,6,7\}, \mathcal{A}, \{S\}, \delta_1, 0, S, \emptyset))$, where the mapping δ_1 is a set of the following transitions:

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

The transition diagram of deterministic pushdown automaton $M_p(t_1)$ is illustrated in Fig. 8.1. In this figure, for each transition rule $\delta_1(p, a, \alpha) = (q, \beta)$ from δ the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$.

Fig. 8.2 shows the sequence of transitions (trace) performed by deterministic pushdown automaton $M_p(t_1)$ for tree t_1 in prefix notation. □

Lemma 8.4. *Given a tree t and its prefix notation $\mathit{pref}(t)$, the PDA $M_p(t) = (\{0,1,2,\ldots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$, where $n \geq 0$, constructed by Alg. 8.2 accepts $\mathit{pref}(t)$.*

Proof. In Part II of the thesis or in [50]. □

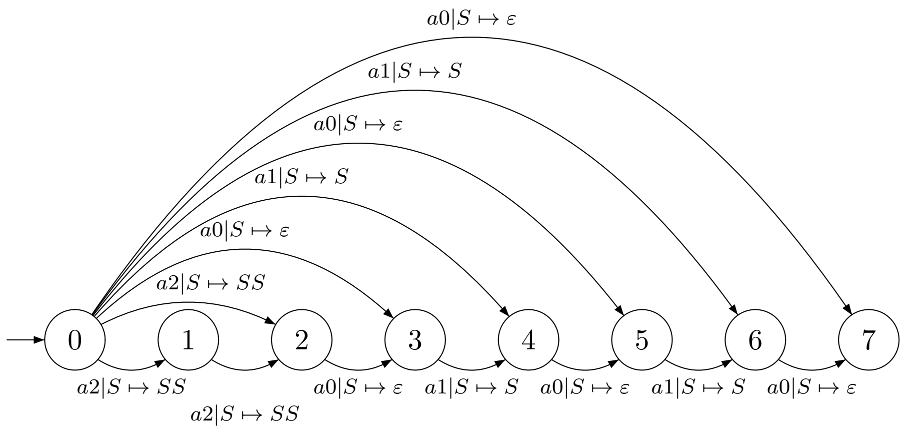

$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a2|S \mapsto SS$
0
1
2
3
4
5
6
7
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$

Figure 8.3: Transition diagram of nondeterministic subtree pushdown automaton $M_{nps}(t_1)$ for tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 8.6

The construction of the deterministic subtree PDA for trees in prefix notation consists of two steps. First, a nondeterministic subtree PDA is constructed by Alg. 8.5. This nondeterministic subtree PDA is an extension of the PDA accepting tree in prefix notation, which is constructed by Alg. 8.2. Second, the constructed nondeterministic subtree PDA is transformed to the equivalent deterministic subtree PDA by Alg. 7.3.

Algorithm 8.5. Construction of a nondeterministic subtree PDA for a tree t in prefix notation $pref(t)$.
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation $pref(t) = a_1a_2 \dots a_n$, $n \geq 1$.
Output: Nondeterministic PDA $M_{nps}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

1. Create PDA $M_{nps}(t)$ as PDA $M_p(t)$ by Alg. 8.2.
2. For each state i, where $2 \leq i \leq n$, create a new transition $\delta(0, a_i, S) = (i, S^{Arity(a_i)})$, where $S^0 = \varepsilon$. □

Example 8.6. A subtree pushdown automaton for tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 3.1, which has been constructed by Alg. 8.5, is nondeterministic pushdown automaton $M_{nps}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_2, 0, S, \emptyset))$, where mapping δ_2 is a set of the following transitions:

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

The transition diagram of nondeterministic pushdown automaton $M_{nps}(t_1)$ is illustrated in Fig. 8.3. Again, in this figure for each transition rule $\delta_2(p, a, \alpha) = (q, \beta)$ from δ_2 the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$.

□

Theorem 8.7. *Given a tree t and its prefix notation $pref(t)$, the PDA $M_{nps}(t)$ constructed by Alg. 8.5 is a subtree PDA for $pref(t)$.*

Proof. In Part II of the thesis or in [50]. □

To construct deterministic subtree or tree pattern PDAs from their nondeterministic versions we use the transformation described by Alg. 7.3.

Example 8.8. The deterministic subtree pushdown automaton for tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 3.1, which has been constructed by Alg. 7.3 from nondeterministic subtree pushdown automaton $M_{nps}(t_1)$ from Example 8.6, is deterministic pushdown automaton $M_{dps}(t_1) = (\{[0], [1,2], [2], [3], [4], [5], [6], [7], [3,5,7], [4,6], [5,7]\}, \mathcal{A}, \{S\}, \delta_3, [0], S, \emptyset))$, where mapping δ_3 is a set of the following transitions:

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

The contents of the pushdown store in particular states are as follows:

$$\begin{array}{ll}
cpds([0]) = \{S\}, & cpds([3,5,7]) = \{\varepsilon\}, \\
cpds([1,2]) = \{SS\}, & cpds([4,6]) = \{S\}, \\
cpds([2]) = \{SSS\}, & cpds([5,7]) = \{\varepsilon\}. \\
cpds([3]) = \{S, SS\}, & \\
cpds([4]) = \{S, SS\}, & \\
cpds([5]) = \{\varepsilon, S\}, & \\
cpds([6]) = \{S\}, & \\
cpds([7]) = \{\varepsilon\}, &
\end{array}$$

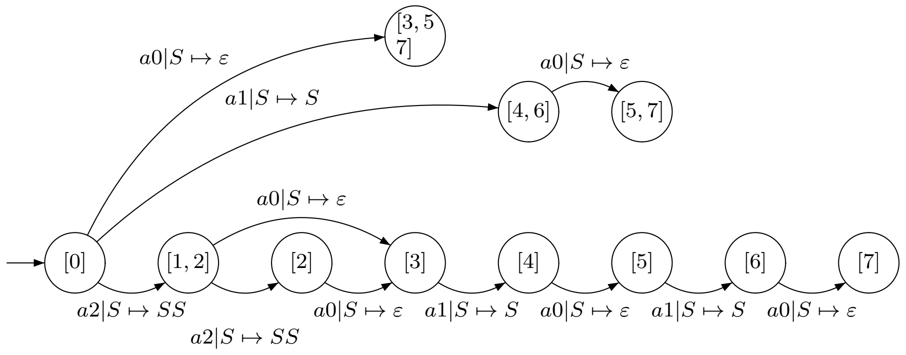
The transition diagram of deterministic pushdown automaton $M_{dps}(t_1)$ is illustrated in Fig. 8.4. Again, in this figure for each transition rule $\delta_3(p, a, \alpha) = (q, \beta)$ from δ_3 the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$.

We note that there are no transitions leading from states $[3,5,7]$, $[5,7]$ and $[7]$, because the contents of the pushdown store (cpds) in these state is always ε and therefore no transition is possible from these states due to the pushdown operations. This means that deterministic subtree pushdown automaton $M_{dps}(t_1)$ has fewer transitions than the deterministic string suffix automaton constructed for $pref(t_1)$ [21, 63, 76].

Fig. 8.5 shows the sequence of transitions (the trace) performed by deterministic subtree pushdown automaton $M_{dps}(t_1)$ for an input subtree st in prefix notation $pref(st) = a_1 a_0$. The accepting state is $[5,7]$, which means there are two occurrences of the input subtree st in tree t_1 and their rightmost leaves are nodes $a0_5$ and $a0_7$. □

Lemma 8.9. *Given a tree t with n nodes, the number of distinct subtrees of tree t is equal or smaller than n.*

Proof. In Part II of the thesis or in [50]. □



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

Figure 8.4: Transition diagram of deterministic subtree pushdown automaton $M_{dps}(t_1)$ for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 8.8

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

Figure 8.5: Trace of deterministic subtree pushdown automaton $M_{dps}(t_1)$ from Example 8.8 for an input subtree st in prefix notation $pref(st) = a1a0$

The deterministic subtree PDA has the only pushdown symbol S, and all its states and transitions correspond to the states and the transitions, respectively, of the deterministic suffix automaton constructed for $pref(t)$. Therefore, the total size of the deterministic subtree PDA cannot be greater than the total size of the deterministic suffix automaton constructed for $pref(t)$.

Theorem 8.10. *Given a tree t with n nodes and its prefix notation $pref(t)$, the deterministic subtree PDA $M_{dps}(t)$ constructed by Algs. 8.5 and 7.3 has just one pushdown symbol, fewer than $N \leq 2n + 1$ states and at most $N + n - 1 \leq 3n$ transitions.*

Proof. In Part II of the thesis or in [50]. □

8.2 Tree pattern pushdown automata

In this section, algorithms and theorems regarding tree pattern pushdown automata for trees in prefix notation are given, and the tree pattern pushdown automata 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 pushdown automata are an extension of subtree pushdown automata, described in the previous section, so that also tree templates would be accepted. New states and transitions, which are used for processing the special nullary symbols S in tree templates, are additionally present in the tree pattern s. The pushdown operations are the same and compute the arity checksum.

Definition 8.11. 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 as follows.

Definition 8.12. Let t, r and $pref(t)$ be a tree, its root and its prefix notation, respectively. A *treetop pushdown automaton* for $pref(t)$ accepts all tree patterns in prefix notation which have the root r and match the tree t.

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

Algorithm 8.13. 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 $pref(t) = a_1a_2 \dots a_n$, $n \geq 1$.
Output: Treetop PDA $M_{pt}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A} \cup \{S\}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

1. Create $M_{pt}(t)$ as $M_p(t)$ by Alg. 8.2.
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 RightMost 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 $srms$. □

Example 8.14. Consider tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 3.1, which is illustrated in Fig. 3.1. The deterministic treetop pushdown automaton, constructed by Alg. 8.13, is deterministic pushdown automaton $M_{pt}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_4, 0, S, \emptyset))$, where mapping δ_4 is a set of the following transitions:

$$\begin{array}{lcl@{\qquad}lcl}
\delta_4(0, a2, S) & = & (1, SS) & & & \\
\delta_4(1, a2, S) & = & (2, SS) & \delta_4(1, S, S) & = & (5, \varepsilon) \\
\delta_4(2, a0, S) & = & (3, \varepsilon) & \delta_3(2, S, S) & = & (3, \varepsilon) \\
\delta_4(3, a1, S) & = & (4, S) & \delta_4(3, S, S) & = & (5, \varepsilon) \\
\delta_4(4, a0, S) & = & (5, \varepsilon) & \delta_4(4, S, S) & = & (5, \varepsilon) \\
\delta_4(5, a1, S) & = & (6, S) & \delta_4(5, S, S) & = & (6, \varepsilon) \\
\delta_4(6, a0, S) & = & (7, \varepsilon) & \delta_4(6, S, S) & = & (7, \varepsilon)
\end{array}$$

The transition diagram of deterministic treetop pushdown automaton $M_{pt}(t_1)$ is illustrated in Fig. 8.6. 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 $a|\alpha \mapsto \beta$.

Deterministic treetop pushdown automaton $M_{pt}(t_1)$ has been constructed by Alg. 8.13 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 $\delta_4(1, S, S) = (5, \varepsilon)$. □

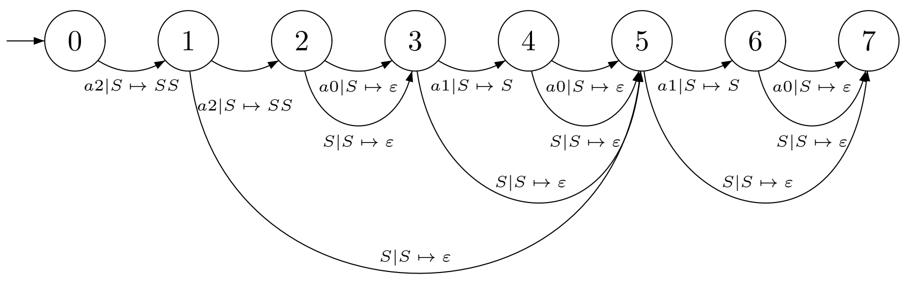


Figure 8.6: Transition diagram of deterministic treetop pushdown automaton $M_{pt}(t_1)$ for tree in prefix notation $\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 8.14

Theorem 8.15. *Given a tree t and its prefix notation $\mathit{pref}(t)$, the PDA $M_{pt}(t)$ constructed by Alg. 8.13 is a treetop PDA for $\mathit{pref}(t)$.*

Proof. In Part II of the thesis or in [50]. □

The nondeterministic tree pattern PDA for trees in prefix notation is constructed as an extension of the deterministic treetop PDA.

Algorithm 8.16. Construction of a nondeterministic tree pattern PDA for a tree t in prefix notation $\mathit{pref}(t)$.
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation $\mathit{pref}(t) = a_1a_2\dots a_n$, $n \geq 1$.
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 Alg. 8.13.
2. For each state i, where $2 \leq i \leq n$, create a new transition
   $\delta(0, a_i, S) = (i, S^{Arity(a_i)})$, where $S^0 = \varepsilon$. □

Example 8.17. Consider tree t_1 in prefix notation $\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 3.1, which is illustrated in Fig. 3.1. The nondeterministic tree pattern pushdown automaton accepting all tree patterns matching tree t_1, which has been constructed by Alg. 8.16, is nondeterministic pushdown automaton $M_{npt}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_5, 0, S, \emptyset))$, where mapping δ_5 is a set of the following transitions:

$$\begin{array}{lcllcllcl}
\delta_5(0, a2, S) & = & (1, SS) & & & & & & \\
\delta_5(1, a2, S) & = & (2, SS) & \delta_5(1, S, S) & = & (5, \varepsilon) & \delta_5(0, a2, S) & = & (2, SS) \\
\delta_5(2, a0, S) & = & (3, \varepsilon) & \delta_3(2, S, S) & = & (3, \varepsilon) & \delta_5(0, a0, S) & = & (3, \varepsilon) \\
\delta_5(3, a1, S) & = & (4, S) & \delta_5(3, S, S) & = & (5, \varepsilon) & \delta_5(0, a1, S) & = & (4, S) \\
\delta_5(4, a0, S) & = & (5, \varepsilon) & \delta_5(4, S, S) & = & (5, \varepsilon) & \delta_5(0, a0, S) & = & (5, \varepsilon) \\
\delta_5(5, a1, S) & = & (6, S) & \delta_5(5, S, S) & = & (6, \varepsilon) & \delta_5(0, a1, S) & = & (6, S) \\
\delta_5(6, a0, S) & = & (7, \varepsilon) & \delta_5(6, S, S) & = & (7, \varepsilon) & \delta_5(0, a0, S) & = & (7, \varepsilon)
\end{array}$$

The transition diagram of nondeterministic tree pattern pushdown automaton $M_{npt}(t_1)$ is illustrated in Fig. 8.7. 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 $a|\alpha \mapsto \beta$. □

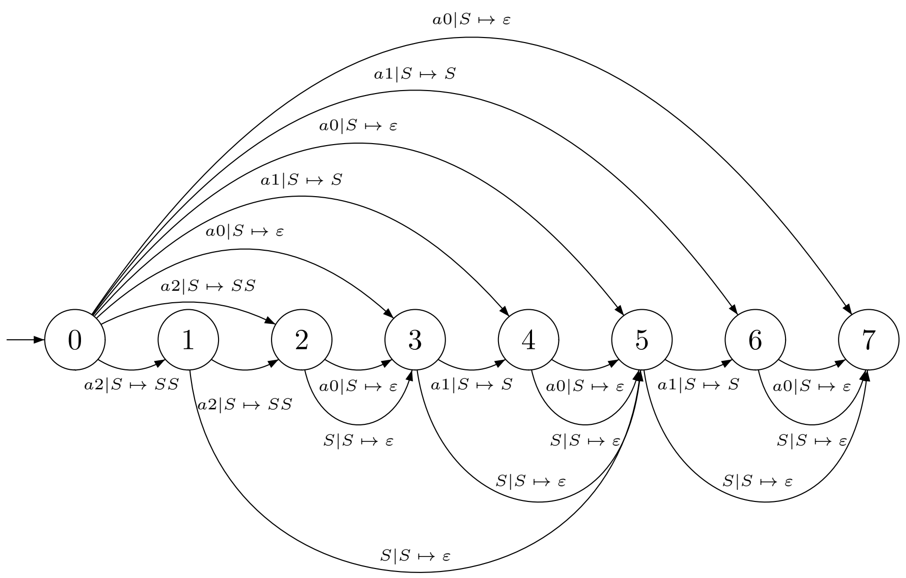


Figure 8.7: Transition diagram of nondeterministic tree pattern pushdown automaton $M_{npt}(t_1)$ from Example 8.17 for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

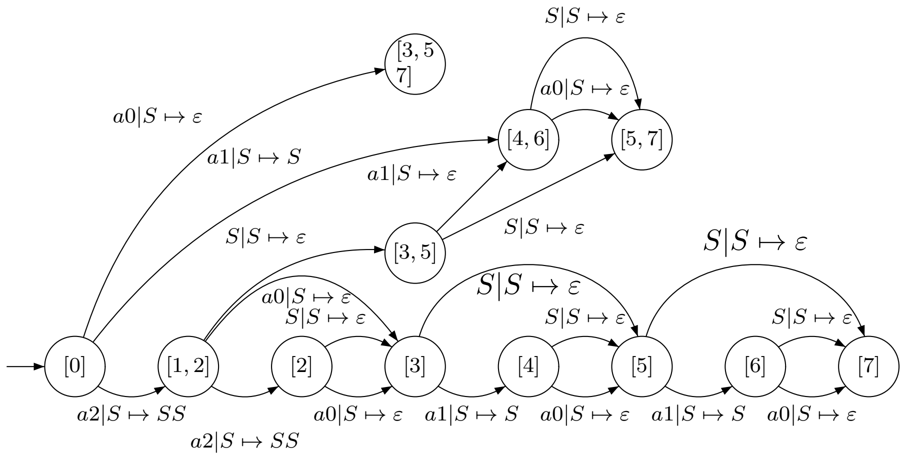
Theorem 8.18. *Given a tree t and its prefix notation $pref(t)$, the PDA $M_{npt}(t)$ constructed by Alg. 8.16 is a tree pattern PDA for $pref(t)$.*

Proof. In Part II of the thesis or in [50]. □

The nondeterministic tree pattern PDA $M_{npt}(t)$ is again an acyclic input-driven PDA, and therefore can be determinised by Alg. 7.3 to an equivalent deterministic tree pattern PDA $M_{dpt}(t)$.

Example 8.19. Consider nondeterministic tree pattern pushdown automaton $M_{npt}(t_1)$ from Example 8.17, the transition diagram of which is illustrated in Fig. 8.7. The deterministic tree pattern pushdown automaton $M_{dpt}(t_1)$ constructed by Alg. 7.3 is $M_{dpt}(t_1) = (\{[0],[1,2],[2],[3],[4],[5],[6],[7],[3,5,7],[3,5],[4,6],[5,7]\}, \mathcal{A}, \{S\}, \delta_6, [0], S, \emptyset))$, where mapping δ_6 is a set of the following transitions:

$$\begin{array}{lcllcl}
\delta_6([0],a0,S) & = & ([3,5,7],\varepsilon) & & & \\
\delta_6([0],a1,S) & = & ([4,6],S) & & & \\
\delta_6([0],a2,S) & = & ([1,2],SS) & & & \\
\delta_6([1,2],a2,S) & = & ([2],SS) & \delta_6([1,2],S,S) & = & ([3,5],\varepsilon) \\
\delta_6([2],a0,S) & = & ([3],\varepsilon) & \delta_6([2],S,S) & = & ([3],\varepsilon) \\
\delta_6([3],a1,S) & = & ([4],S) & \delta_6([3],S,S) & = & ([5],\varepsilon) \\
\delta_6([4],a0,S) & = & ([5],\varepsilon) & \delta_6([4],S,S) & = & ([5],\varepsilon) \\
\delta_6([5],a1,S) & = & ([6],S) & \delta_6([5],S,S) & = & ([7],\varepsilon) \\
\delta_6([6],a0,S) & = & ([7],\varepsilon) & \delta_6([6],S,S) & = & ([7],\varepsilon) \\
\delta_6([3,5],a1,S) & = & ([4,6],S) & \delta_6([3,5],S,S) & = & ([5,7],\varepsilon) \\
\delta_6([4,6],a0,S) & = & ([5,7],\varepsilon) & \delta_6([4,6],S,S) & = & ([5,7],\varepsilon) \\
\delta_6([1,2],a0,S) & = & ([3],\varepsilon) & & &
\end{array}$$



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

Figure 8.8: Transition diagram of deterministic tree pattern pushdown automaton $M_{dpt}(t_1)$ from Example 8.19 for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

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

Figure 8.9: Trace of deterministic pushdown automaton M_{dpt} from Example 8.19

The transition diagram of deterministic tree pattern pushdown automaton $M_{dpt}(t_1)$ is illustrated in Fig. 8.8. 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 $a|\alpha \mapsto \beta$.

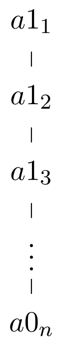
Fig. 8.9 shows the sequence of transitions (the trace) performed by deterministic tree pattern pushdown automaton $M_{dpt}(t_1)$ for input tree pattern $p_1 = a2\ S\ a1\ S$, which is illustrated in Fig. 3.2. □

The rest of this section is devoted to a discussion on the space required by the deterministic tree pattern pushdown automaton.

Lemma 8.20. *Given a tree t with n nodes, the number of distinct tree patterns which match the tree t can be at most $2^{n-1} + n$.*

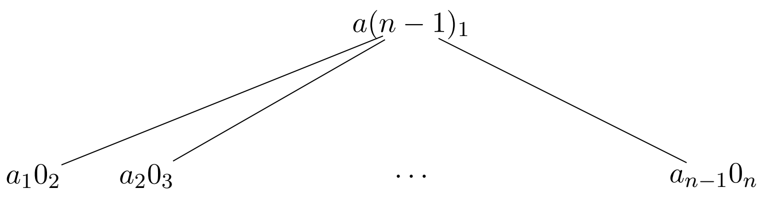
Proof. In Part II of the thesis or in [50]. □

Example 8.21. Figs. 8.12 and 8.13 show the transition diagrams of the deterministic tree pattern pushdown automata constructed by Algs. 8.16 and 7.3 for the two examples of trees with the boundary structures illustrated in Figs. 8.10 and 8.11, respectively. □



$$pref(t_4) = (a1)^{n-1}a0$$

Figure 8.10: A tree t_4, which represents a string, and its prefix notation



$$pref(t_5) = a(n-1)\ a_1 0\ a_2 0 \ldots a_{n-1} 0$$

Figure 8.11: A tree t_5 with $2^{n-1} + n$ distinct tree patterns matching the tree t_5 and its prefix notation

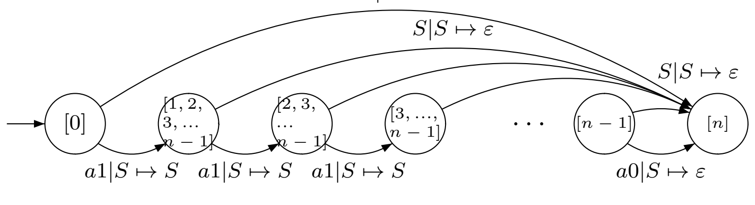
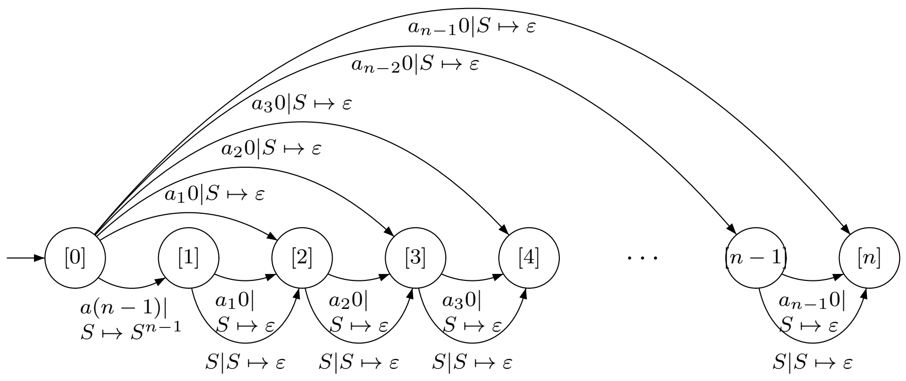


Figure 8.12: Transition diagram of the deterministic tree pattern pushdown automaton $M_{dpt}(t_4)$ for the tree t_4 in prefix notation $pref(t_4) = (a1)^{n-1}a0$, where $n \geq 1$



$a_{n-1}0|S \mapsto \varepsilon$
$a_{n-2}0|S \mapsto \varepsilon$
$a_3 0|S \mapsto \varepsilon$
$a_2 0|S \mapsto \varepsilon$
$a_1 0|S \mapsto \varepsilon$
[0]
[1]
[2]
[3]
[4]
$\cdots$
$[n-1]$
$[n]$
$a(n-1)|$ $S \mapsto S^{n-1}$
$a_1 0|$ $S \mapsto \varepsilon$
$a_2 0|$ $S \mapsto \varepsilon$
$a_3 0|$ $S \mapsto \varepsilon$
$a_{n-1}0|$ $S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$

Figure 8.13: Transition diagram of the deterministic tree pattern pushdown automaton $M_{dpt}(t_3)$ for the tree t_5 in prefix notation $pref(t_5) = a(n-1)\ a_1 0\ a_2 0 \ldots a_{n-1}0$, where $n \geq 2$ and $a_i 0 \neq a_j 0$, $i \neq j$

As the simplest case, where the total size of the deterministic tree pattern pushdown automaton is linear in the number of nodes of the tree, we consider a deterministic tree pattern pushdown automaton that has just the same states as the corresponding deterministic subtree pushdown automaton. This occurs for specific cases of trees which are called trees with periodical subtrees, and are defined by Definition 8.22.

Definition 8.22. Let t be a tree over a ranked alphabet $\mathcal{A}$. The tree t is a *tree with periodical subtrees* if there exists a mapping $\mathcal{Z}$ of $\mathcal{A} \setminus \mathcal{A}_0$ into $\mathcal{A}^+$ such that the prefix notation of each subtree st of the tree t is of the form $pref(st) = (ax)^m y$, where $a \in \mathcal{A}$, $x \in \mathcal{A}^*$, $y \in \mathcal{A}^+$, $m \geq 1$, $\mathcal{Z}(a) = xy$, and y is a subtree in prefix notation.

Example 8.23. Examples of trees with periodical subtrees are illustrated in Figs. 8.10, 8.11 and 8.14, and the transition diagrams of the corresponding deterministic tree pattern pushdown automata are illustrated in Figs. 8.12, 8.13 and 8.15, respectively.

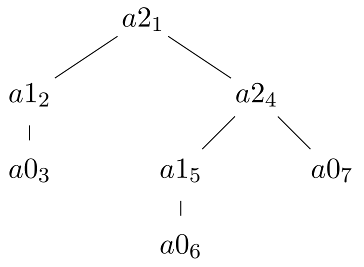
In the case of tree t_6, which is constructed over ranked alphabet $\{a2, a1, a0\}$ and is illustrated in Fig. 8.14, the corresponding mapping $\mathcal{Z}_4$ is the following set:

$$\begin{aligned} \mathcal{Z}_4(a2) &= a1\ a0\ a0, \text{ where } x = a1\ a0 \text{ and } y = a0 \\ \mathcal{Z}_4(a1) &= a0, \text{ where } x = \varepsilon \text{ and } y = a0 \end{aligned}$$

Prefix notations of all subtrees of tree t_6 are: $a2\ a1\ a0\ a2\ a1\ a0\ a0$, $a2\ a1\ a0\ a0$, $a1\ a0$ and $a0$. □

The resulting number of states and of transitions of the deterministic tree pattern pushdown automata for trees with determined subtrees are formally proved in Theorem 8.24. We also present a companion Lemma 8.25 which shows that the deterministic tree pattern pushdown automaton has more states than the deterministic subtree pushdown automaton if the condition for the trees with determined subtrees is violated.

Theorem 8.24. *Let t be a tree over a ranked alphabet $\mathcal{A}$. If the tree t is a tree with periodical subtrees, then the deterministic tree pattern pushdown automaton $M_{dpt}(t)$ constructed by Algs. 8.16 and 7.3 has just one pushdown symbol, fewer than $N \leq 2n+1$ states and at most $2N + n - 3 \leq 5n - 1$ transitions.*



$$pref(t_4) = a2\ a1\ a0\ a2\ a1\ a0\ a0$$

Figure 8.14: Tree t_6 from Example 8.23 and its prefix notation

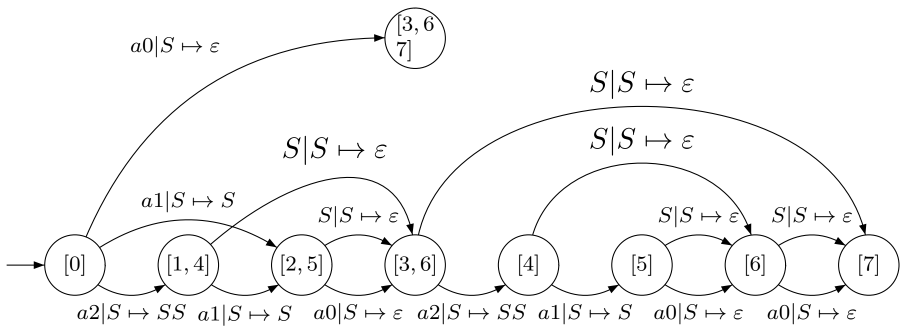
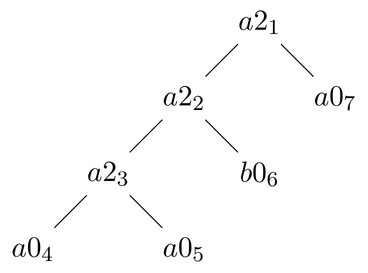

$a0|S \mapsto \varepsilon$
$[3,6,7]$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$a1|S \mapsto S$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$[0]$
$[1,4]$
$[2,5]$
$[3,6]$
$[4]$
$[5]$
$[6]$
$[7]$
$a2|S \mapsto SS$
$a1|S \mapsto S$
$a0|S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

Figure 8.15: Transition diagram of deterministic tree pattern pushdown automaton $M_{dpt}(t_6)$ from Example 8.23 for tree in prefix notation $pref(t_6) = a2\ a1\ a0\ a2\ a1\ a0\ a0$



$pref(t_5) = a2\ a2\ a2\ a0\ a0\ b0\ a0$

Figure 8.16: Tree t_7 from Example 8.26 and its prefix notation

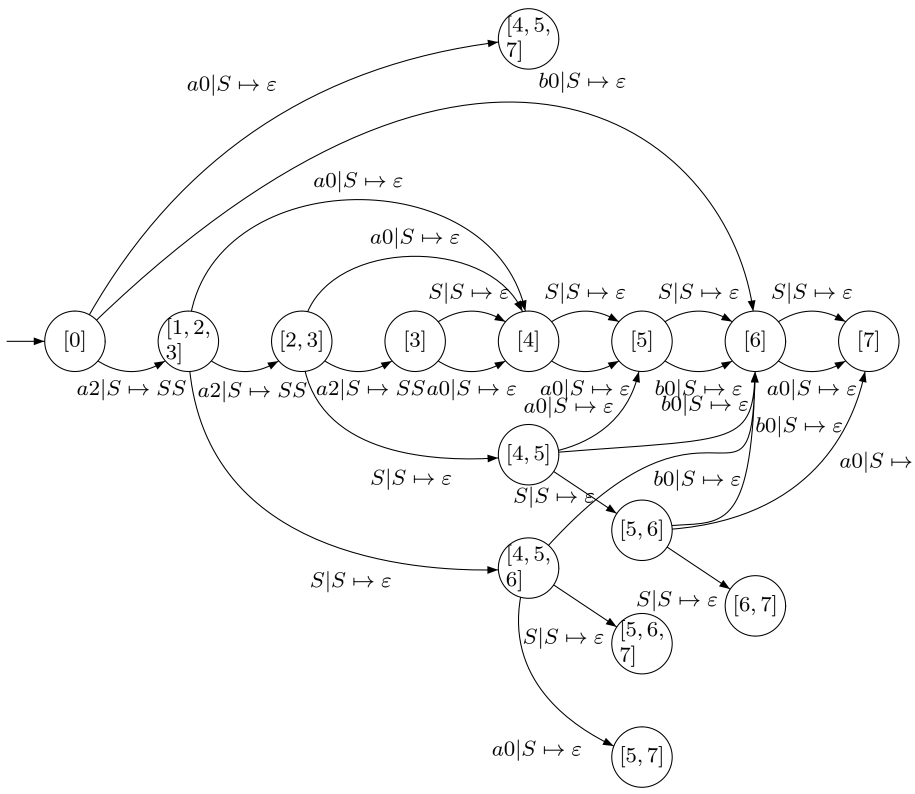
Proof. In Part II of the thesis or in [50]. □

Lemma 8.25. *Let t be a tree over a ranked alphabet $\mathcal{A}$. If the tree t is not a tree with periodical subtrees, then the deterministic tree pattern pushdown automaton $M_{dpt}(t)$ constructed by Algs. 8.16 and 7.3 has more states than the deterministic subtree pushdown automaton $M_{dps}(t)$ constructed by Algs. 8.5 and 7.3.*

Proof. In Part II of the thesis or in [50]. □

Example 8.26. Examples of trees which are not trees with periodical subtrees are illustrated in Figs. 3.1 and 8.16, and the transition diagrams of the corresponding deterministic tree pattern pushdown automata are illustrated in Figs. 8.8 and 8.17, respectively. In Fig. 8.8, state $[3, 5]$ is not in the corresponding deterministic subtree pushdown automaton. In Fig. 8.17, states $[4, 5]$, $[4, 5, 6]$, $[5, 6]$, $[5, 7]$, $[5, 6, 7]$ and $[6, 7]$ are not in the corresponding deterministic subtree pushdown automaton. □

In general, the deterministic tree pattern pushdown automaton can have more than linear number of states. For example, given a tree t in prefix notation $pref(t) = a2^m a0^{m+1}$, $m \geq 1$, the corresponding pushdown automaton $M_{dpt}(t)$ has $N = \frac{m^2+m}{2} + 2m + 2$ states and $2(N - m - 1) = m^2 + 3m + 2$ transitions. This means that the number of distinct tree patterns which match such a tree t is exponential in the number of nodes of the tree and the total size of the corresponding deterministic tree pattern pushdown automaton is quadratic. The maximal numbers of states and transitions of the deterministic tree pattern pushdown automaton in general remain open problems.



$a0|S \mapsto \varepsilon$
$b0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$[4,5,7]$
$[0]$
$[1,2,3]$
$[2,3]$
$[3]$
$[4]$
$[5]$
$[6]$
$[7]$
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$
$b0|S \mapsto \varepsilon$
$b0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$
$b0|S \mapsto \varepsilon$
$[4,5]$
$a0|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$b0|S \mapsto \varepsilon$
$[5,6]$
$S|S \mapsto \varepsilon$
$[4,5,6]$
$S|S \mapsto \varepsilon$
$[6,7]$
$S|S \mapsto \varepsilon$
$[5,6,7]$
$a0|S \mapsto \varepsilon$
$[5,7]$

Figure 8.17: Transition diagram of deterministic tree pattern pushdown automaton $M_{dpt}(t_7)$ from Example 8.26 for tree in prefix notation $pref(t_7) = a2\ a2\ a2\ a0\ a0\ b0\ a0$

Chapter 9

Repeats in trees

Efficient methods of finding various kinds of repeats in a string can be based on constructing and analysing string suffix trees or string suffix automata, which represent complete indexes of the string for substrings. In the previous chapter we describe subtree pushdown automata, which are analogous to the string suffix automata and represent complete indexes of trees for subtrees. This chapter presents a new and simple method of finding various kinds of all repeats of subtrees in a given tree by constructing and analysing the subtree pushdown automaton for the tree. Given a tree with n nodes, the finding of all repeats of subtrees in the tree is performed in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.

Given a tree, the problem is to find all repeating subtrees of the tree and to compute kinds and positions of all occurrences of these subtrees. All repeats of subtrees and their properties are summarised in a subtree repeat table, which is defined by Defs. 9.1, 9.2 and 9.3. We define two versions of the subtree repeat table: the first, basic, version of the table contains basic information on repeats and its size is linear to the number of nodes of the tree. The second one, an extended subtree repeat table, contains also further information such as all the repeating subtrees in prefix notation, which can result in a larger table.

Definition 9.1. Let t be a tree over a ranked alphabet $\mathcal{A}$. A *subtree position set* $\mathrm{sps}(st, t)$, where st is a subtree of t, is the set $sps(st, t) = \{i : pref(t) = x\ pref(st)\ y,\ x, y \in \mathcal{A}^*, i = |x| + 1\}$.

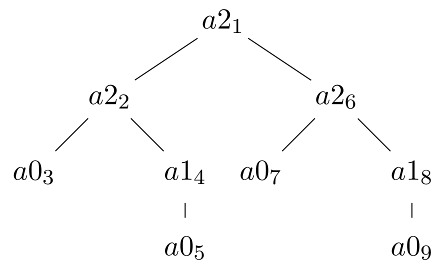
Definition 9.2. Let t be a tree over a ranked alphabet $\mathcal{A}$. Given a subtree st of t, *list of subtree repeats* $lsr(st, t)$ is a relation in $sps(st, t) \times \{F, S, Q\}$ defined as follows:

- $(i, F) \in lsr(st, t)$ iff $pref(t) = x\ pref(st)\ y$, $i = |x| + 1$, $x \neq x_1\ pref(st)\ x_2$,
- $(i, S) \in lsr(st, t)$ iff $pref(t) = x\ pref(st)\ y$, $i = |x| + 1$, $x = x_1\ pref(st)$,
- $(i, G) \in lsr(st, t)$ iff $pref(t) = x\ pref(st)\ y$, $i = |x| + 1$, $x = x_1\ pref(st)\ x_2$, $x_2 \in \mathcal{A}^+$.

Informally, the list of subtree repeats for a subtree st contains kinds of all occurrences of the subtree st. Abbreviations F, S, and G stand for First occurrence of the subtree, repeat as a Square, and repeat with a Gap, respectively. In comparison with kinds of repeats in string [61, 63], repeats of subtrees have no kind which would represent the overlapping of subtrees because any two different occurrences of the same subtree cannot overlap.

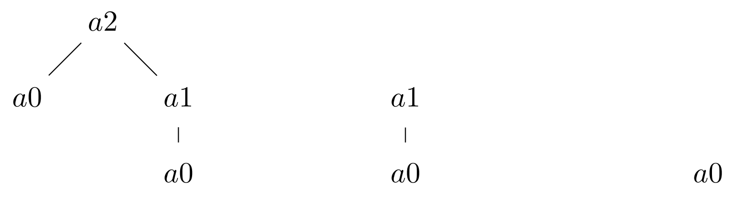
Definition 9.3. Given a tree t, the *basic subtree repeat table* $BSRT(t)$ is the set of all lists of subtree repeats $lsr(st, t)$, where st is a subtree with more than one occurrence in the tree t.

The *extended subtree repeat table* $ESRT(t)$ is the set of all triplets $(sps(st, t), pref(st), lsr(st, t))$, where st is a subtree with more than one occurrence in the tree t.



$pref(t_8) = a2\ a2\ a0\ a1\ a0\ a2\ a0\ a1\ a0$

Figure 9.1: Tree t_8 from Example 9.4 and its prefix notation



$pref(st_1) = a2\ a0\ a1\ a0$ $pref(st_2) = a1\ a0$ $pref(st_3) = a0$

Figure 9.2: Subtrees with more than one occurrence in tree t_8 from Example 9.4, and their prefix notations

Example 9.4. Consider a ranked alphabet $\mathcal{A} = \{a0, a1, a2\}$. Consider a tree t_8 over $\mathcal{A}$ $t_8 = (\{a2_1, a2_2, a0_3, a1_4, a0_5, a2_6, a0_7, a1_8, a0_9\}, R_8)$, where R_8 is a set of the following ordered sequences of pairs:

$$\begin{array}{l}((a2_1, a2_2), (a2_1, a2_6)),\\ ((a2_2, a0_3), (a2_2, a1_4)),\\ ((a1_4, a0_5)),\\ ((a2_6, a0_7), (a2_6, a1_8)),\\ ((a1_8, a0_9))\end{array}$$

Tree t_8 in prefix notation is string $pref(t_8) = a2\ a2\ a0\ a1\ a0\ a2\ a0\ a1\ a0$. Trees tree t_8 is illustrated in Fig. 9.1.

Subtrees with more than one occurrence in tree t_8 are subtrees st_1, st_2 and st_3, where $pref(st_1) = a2\ a0\ a1\ a0$, $pref(st_2) = a1\ a0$, and $pref(st_3) = a0$. These three subtrees are illustrated in Fig. 9.2. All other subtrees of tree t_8 are present just once in tree t_8.

It holds that $sps(st_1) = \{2, 6\}$, $sps(st_2) = \{4, 8\}$, $sps(st_3) = \{3, 5, 7, 9\}$, and the corresponding basic subtree repeat table $BSRT(t_1)$ and extended subtree repeat table $ESRT(t_1)$ are illustrated in Fig. 9.3 and Fig. 9.4, respectively. $\square$

List of subtree repeats
$(2, F), (6, S)$
$(4, F), (8, G)$
$(3, F), (5, G), (7, G), (9, G)$

Figure 9.3: Basic subtree repeat table $BSRT(t_1)$ from Example 9.4

Subtree position set	Subtree in prefix notation	List of subtree repeats
$2, 6$	$a2\ a0\ a1\ a0$	$(2, F), (6, S)$
$4, 8$	$a1\ a0$	$(4, F), (8, G)$
$3, 5, 7, 9$	$a0$	$(3, F), (5, G), (7, G), (9, G)$

Figure 9.4: Extended subtree repeat table $ESRT(t_1)$ from Examples 9.4

Algorithm 9.5. Construction of the basic subtree repeat table for a tree t in prefix notation $pref(t)$.
Input: A tree t; prefix notation $pref(t) = a_1a_2 \dots a_n$, $n \geq 1$.
Output: Basic subtree repeat table $BSRT(t)$.
Method:

1. Initially, $BSRT(t) = \emptyset$.
2. Create $M_{nps}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$ by Alg. 8.5.
3. Let Q' denote a set of states. Let $pdsl(q')$, where $q' \in Q'$, denote a set of pairs of integers (the abbreviation $pdsl$ stands for the number of symbols S in the PushDown Store, and the Length of the subtree.)
4. $Q' = \{[0]\}$, $pdsl([0]) = \{(1, 0)\}$ and $[0]$ is an unmarked state.
5. (a) Select an unmarked state q' from Q' such that q' contains the smallest possible state $q \in Q$, where $0 \leq q \leq n$.
   (b) For each $(0, l) \in pdsl(q')$ to $BSRT(t)$ add pairs (x, Z), where $x = r - l$, $r \in q'$ and:
      i. $Z = F$ if x is the smallest such number x,
      ii. $Z = S$ if $x - 1 \in q''$,
      iii. $Z = G$ otherwise.
   (c) If there is $v > 0$, $(v, w) \in pdsl(q')$, then for each input symbol $a \in \mathcal{A}$:
   Compute state $q'' = \{q : \delta(p, a, \alpha) = (q, \beta)$ for all $p \in q'\}$.
   If q'' is not in Q' and $|q''| > 1$, then add q'' to Q' and create $pdsl(q'') = \emptyset$.
   Add pairs $(j, k + 1)$, where $(i, k) \in pdsl(q')$, $i > 0$, $j = i + arity(a) - 1$, to $pdsl(q'')$.
   (d) Set the state q' as marked.
6. Repeat step 5 until all states in Q' are marked. □

Example 9.6. The basic subtree repeat table $BSRT(t_1)$ constructed by Alg. 9.5 for tree t_8 from Example 9.4 is illustrated in Fig. 9.3. During this construction states $[0]$, $[1, 2, 6]$, $[3, 5, 7, 9]$, $[3, 7]$, $[4, 8]$, $[5, 9]$, and the following set $pdsl$ have been constructed:

$$\begin{aligned} &pdsl([0]) = \{(1,0)\}, && pdsl([3,5,7,9]) = \{(0,1)\}, \\ &pdsl([1,2,6]) = \{(2,1)\}, && pdsl([3,7]) = \{(1,2)\}, \\ & && pdsl([4,8]) = \{(1,1),(1,3)\}, \\ & && pdsl([5,9]) = \{(0,2),(0,4)\}. \end{aligned}$$

Theorem 9.7. *Given a tree t with n nodes, Alg. 9.5 correctly constructs the basic subtree repeat table $BSRT(t)$ in time $\mathcal{O}(n)$ and space $\mathcal{O}(n)$.*

Proof. In Part II of the thesis or in [51]. □

Chapter 10

Tree Pattern Matching

A systematic approach to the construction of subtree pattern matchers by deterministic pushdown automata, which read subject trees in prefix and postfix notation, is presented in this chapter. The method is analogous to the construction of string pattern matchers: for a given pattern, a nondeterministic pushdown automaton is created and then it is determinised. The size of the resulting deterministic pushdown automata directly corresponds to the size of the existing string pattern matchers based on finite automata.

Definition 10.1. Let s and *pref*(s) be a tree and its prefix notation, respectively. Given an input tree t, a subtree pushdown automaton constructed over *pref*(s) accepts all matches of tree s in the input tree t by final state.

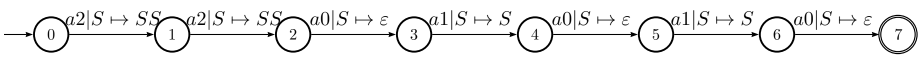
10.1 Subtree matching

First, we start with a PDA which accepts the whole subject tree in prefix notation. The construction of the PDA accepting a tree in prefix notation is described by Alg. 10.2. The constructed PDA is deterministic.

Algorithm 10.2. Construction of a PDA accepting *pref*(t) by final state.
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation *pref*$(t) = a_1a_2 \dots a_n$, $n \geq 1$.
Output: PDA $M_p(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \{n\})$.
Method:

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

Example 10.3. The PDA constructed by Alg. 10.2, accepting the prefix notation *pref*$(t_1) =$ $a2\ a2\ a0\ a1\ a0\ a1\ a0$ of tree t_1 from Example 6.1, is the deterministic PDA $M_p(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_1, 0, S, \{n\}))$, where the mapping δ_1 is a set of the following transitions:



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

Figure 10.1: Transition diagram of deterministic PDA $M_p(t_1)$ accepting tree t_1 in prefix notation *pref*$(t_1) = a2\ a0\ a2\ a0\ a0\ a0$ from Example 10.3

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

Figure 10.2: Trace of deterministic PDA $M_p(t_1)$ from Example 10.3 for tree t_1 in prefix notation *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

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

The transition diagram of deterministic PDA $M_p(t_1)$ is illustrated in Fig. 10.1. Fig. 10.2 shows the sequence of transitions (trace) performed by deterministic PDA $M_p(t_1)$ for tree t_1 in prefix notation. □

Lemma 10.4. *Given a tree t and its prefix notation pref(t), the PDA* $M_p(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, F)$*, where* $n = |t|$*, constructed by Alg. 10.2, accepts pref(t).*

Proof. In Part II of the thesis or in [34]. □

We present the construction of the deterministic subtree matching PDA for trees in prefix notation. The construction consists of two steps. First, a nondeterministic subtree matching PDA is constructed by Alg. 10.5. This nondeterministic subtree matching PDA is an extension of the PDA accepting trees in prefix notation, which is constructed by Alg. 10.2. Second, the constructed nondeterministic subtree matching PDA is transformed to the equivalent deterministic subtree matching PDA by Alg. 7.3.

Algorithm 10.5. Construction of a nondeterministic subtree matching PDA for a tree t in prefix notation *pref*(t).
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation *pref*$(t) = a_1a_2 \ldots a_n$, $n \geq 1$.
Output: Nondeterministic subtree matching PDA $M_{nps}(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, \{S\}, \delta, 0,$

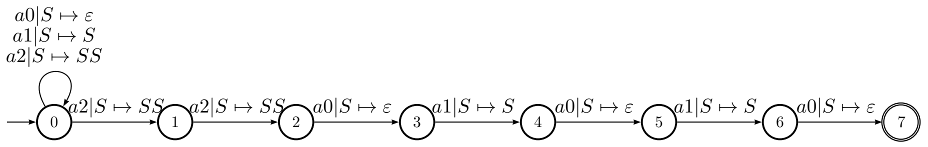


Figure 10.3: Transition diagram of nondeterministic subtree matching PDA $M_p(t_1)$ for tree t_1 in prefix notation *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 10.6

$S, \{n\})$.
Method:

1. Create PDA $M_{nps}(t)$ as PDA $M_p(t)$ by Alg. 10.2.
2. For each symbol $a \in \mathcal{A}$ create a new transition $\delta(0, a, S) = (0, S^{Arity(a)})$, where $S^0 = \varepsilon$. □

Example 10.6. The subtree matching PDA, constructed by Alg. 10.5 from tree t_1 having prefix notation *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$, is the nondeterministic PDA $M_{nps}(t_1) = (\{0,1,2,3,4,5,6,7\}, \mathcal{A}, \{S\}, \delta_2, 0, S, \{7\}))$, where mapping δ_2 is a set of the following transitions:

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

The transition diagram of the nondeterministic PDA $M_{nps}(t_1)$ is illustrated in Fig. 10.3. □

Theorem 10.7. *Given a tree t and its prefix notation pref(t), the PDA $M_{nps}(t)$ constructed by Alg. 10.5 is a subtree matching PDA for pref(t).*

Proof. In Part II of the thesis or in [34]. □

For the construction of deterministic subtree matching PDA, we use the transformation described by Alg. 7.3.

Example 10.8. The deterministic subtree matching PDA for tree t_8 in prefix notation *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$, which has been constructed by Alg. 7.3 from nondeterministic subtree matching PDA $M_{nps}(t_8)$, is the deterministic PDA $M_{dps}(t_8) = (\{[0], [0,1], [0,1,2], [0,3], [0,4], [0,5], [0,6], [0,7]\}, \mathcal{A}, \{S\}, \delta_3, [0], S, \{[0,7]\})$, where its transition diagram is illustrated in Fig. 10.5. Fig. 10.4 shows the sequence of transitions (trace) performed by the deterministic subtree PDA $M_{dps}(t_8)$ for an input tree t_9 in prefix notation *pref*$(t_9) = a2\ a2\ a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$. The accepting state is $[0,7]$. Fig. 10.6 depicts the pattern subtree t_8 and input tree t_9. □

State	Input	PDS
$\{0\}$	$a2\ a2\ a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	S
$\{0,1\}$	$a2\ a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	SS
$\{0,1,2\}$	$a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	SSS
$\{0,1,2\}$	$a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	$SSSS$
$\{0,3\}$	$a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	SSS
$\{0,4\}$	$a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	SSS
$\{0,5\}$	$a1\ a0\ a1\ a1\ a2\ a0\ a0$	SS
$\{0,6\}$	$a0\ a1\ a1\ a2\ a0\ a0$	SS
$\{0,7\}$	$a1\ a1\ a2\ a0\ a0$ match	S
$\{0\}$	$a1\ a2\ a0\ a0$	S
$\{0\}$	$a2\ a0\ a0$	S
$\{0,1\}$	$a0\ a0$	SS
$\{0\}$	$a0$	S
$\{0\}$	ε	ε

Figure 10.4: Trace of deterministic subtree PDA $M_{dps}(t_8)$ from Example 10.8 for an input subtree t_9 in prefix notation *pref*$(t_9) = a2\ a2\ a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$

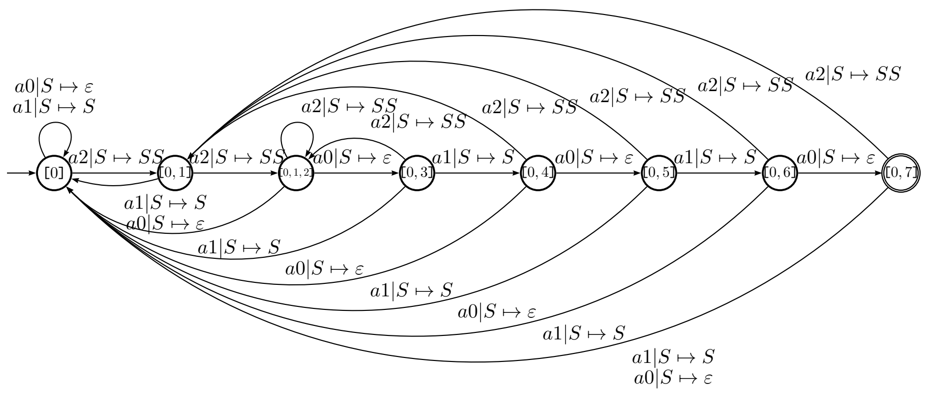


Figure 10.5: Transition diagram of deterministic PDA $M_{dps}(t_8)$ for tree t_8 in prefix notation *pref*$(t_8) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 10.8

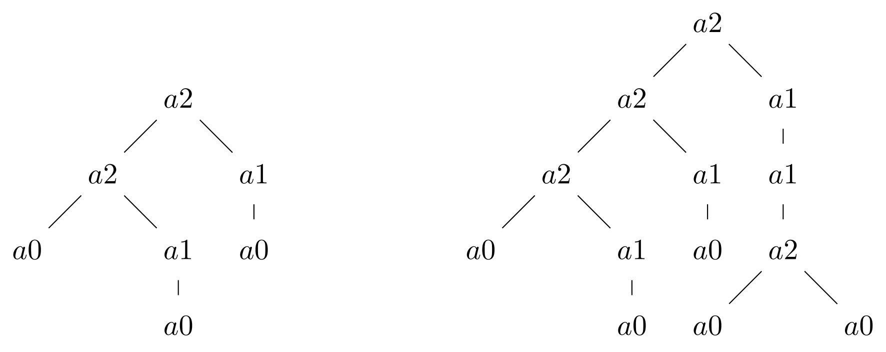


Figure 10.6: Trees t_8 and t_9 from Example 10.8

Theorem 10.9. *Given a tree t with n nodes in its prefix or postfix notation, the deterministic subtree matching PDA $M_{pds}(t)$ constructed by Alg. 10.5 and 7.3 is made of exactly $n+1$ states, one pushdown symbol and $|\mathcal{A}|(n+1)$ transitions.*

Proof. In Part II of the thesis or in [34]. □

Theorem 10.10. *Given an input tree t with n nodes, the searching phase of the deterministic subtree matching automaton constructed by Algs. 10.5 and 7.3 is $\mathcal{O}(n)$.*

Proof. In Part II of the thesis or in [34]. □

10.2 Multiple subtree matching

Definition 10.11. Let $P = \{t_1, t_2, \ldots, t_m\}$ be a set of m trees and $pref(t_i), 1 \leq i \leq m$ be the prefix notation of the i-th tree in P. Given an input tree t, a subtree pushdown automaton constructed over set P accepts all matches of subtrees $t_1, t_2, \ldots, t_m$ in the input tree t by final state.

Similarly as in Subsection 10.1, our method begins with a PDA which accepts trees $t_1, t_2, \ldots, t_m$ in their prefix notation. The construction of this PDA is described by Alg. 10.12.

Algorithm 10.12. Construction of a PDA accepting a set of trees $P = \{t_1, t_2, \ldots, t_m\}$ in their prefix notation.
Input: A set of trees $P = \{t_1, t_2, \ldots, t_m\}$ over a ranked alphabet $\mathcal{A}$; prefix notation $pref(t_i) = a_1a_2 \ldots a_{n_i}$, $1 \leq i \leq m$, $n_i \geq 1$.
Output: PDA $M_p(P) = (\{0, 1, 2, \ldots, q\}, \mathcal{A}, \{S\}, \delta, 0, S, F)$.
Method:

1. Create PDAs $M_p(t_i) = (Q_i, \mathcal{A}, \{S\}, \delta_i, 0_i, S, F_i)$ by Alg. 10.2 for $i = 1, 2, \ldots, m$.
2. Create PDA $M_p(P) = (Q, \mathcal{A}, \{S\}, \delta, 0, S, F)$, where $Q = \bigcup_{i=1}^{m}(Q_i \setminus \{0_i\}) \cup \{0\}$, $\delta(q, a, S) = \delta_i(q, a, S)$,

$\delta(0, a, S) = \delta_i(0_i, a, S)$, where $q \in (Q \setminus \{0\})$, $i = 1, 2, \ldots, m$,
$F = \bigcup_{i=1}^{m} F_i$. □

The correctness of the deterministic PDA constructed by Alg. 10.12, which accepts trees in prefix notation, is described by the following lemma.

Lemma 10.13. *Given a set of k trees $P = \{t_1, t_2, \ldots, t_m\}$ and their prefix notation pref(t_i), $1 \leq i \leq m$, the PDA $M_p(P) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, F)$, where $1 + min(|t_1|, |t_2|, \ldots, |t_m|) \leq n \leq 1 + \sum_{j=1}^{k} |t_j|$, constructed by Alg. 10.12 accepts pref(t_i), where $1 \leq t_i \leq m$.*

Proof. In Part II of the thesis or in [34]. □

The deterministic subtree matching PDA for multiple tree patterns in prefix notation can be constructed in a similar fashion to the subtree matching PDA for a single pattern. First, the PDA accepting a set of trees in their prefix notations, constructed by Alg. 10.12, is used to construct a nondeterministic subtree matching PDA by Alg. 10.14. The constructed nondeterministic subtree matching PDA is then transformed to the equivalent deterministic subtree matching PDA by Alg. 7.3.

Algorithm 10.14. Construction of a nondeterministic subtree matching PDA for a set of trees $P = \{t_1, t_2, \ldots, t_m\}$ in their prefix notation.
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation *pref*(t) $= a_1 a_2 \ldots a_n$, $n \geq 1$.
Output: Nondeterministic subtree matching PDA $M_{nps}(t) = (Q, \mathcal{A}, \{S\}, \delta, 0, S, F)$.
Method:

1. Create PDA $M_{nps}(t)$ as PDA $M_p(t) = (Q, \mathcal{A}, \{S\}, \delta, 0, S, F)$ by Alg. 10.12.
2. For each symbol $a \in \mathcal{A}$ create a new transition $\delta(0, a, S) = (0, S^{Arity(a)})$, where $S^0 = \varepsilon$. □

Theorem 10.15. *Given a set of m trees $P = \{t_1, t_2, \ldots, t_m\}$ and their prefix notation pref(t_i), $1 \leq i \leq m$, the PDA $M_{nps}(P)$ constructed by Algs. 10.12 and 10.14 is a subtree matching PDA for tree patterns $t_1, t_2, \ldots, t_m$.*

Proof. In Part II of the thesis or in [34]. □

Theorem 10.16. *Given a set of m trees $P = \{t_1, t_2, \ldots, t_m\}$ over a ranked alphabet $\mathcal{A}$, the deterministic subtree matching PDA $M_{pds}(P)$ is constructed by Alg. 10.14 and 7.3 in time $\Theta(|\mathcal{A}|s)$, requires $\Theta(|\mathcal{A}|s)$ storage, where $s = \sum_{i=1}^{m} |t_i|$, and its pushdown store alphabet consists of one symbol.*

Proof. In Part II of the thesis or in [34]. □

Theorem 10.17. *Given an input tree t with n nodes, the searching phase of the deterministic subtree matching automaton constructed by Algs. 10.5 and 7.3 over a set of m trees P is $\mathcal{O}(n)$.*

Proof. In Part II of the thesis or in [34]. □

Chapter 11

Conclusions, current and future research

Basic arbology results, principles and algorithms have been created and they are presented in brief in the first part of this thesis. Algorithms for particular problems on trees are presented for prefix notation of trees; these algorithms can be easily modified also for postfix, bar prefix and bar postfix notations in the following way: instead of the pair of Theorems (6.4, 6.12), the pairs of Theorems (6.6, 6.13), (6.10, 6.14), and (6.11, 6.15), respectively, can be considered and the pushdown operations can be changed accordingly. Bořivoj Melichar as an invited speaker [62] is to present a similar overview of basic arbology results at Language and Automata Theory and Applications conference (LATA 2010). The detailed descriptions of the results can be found in [34, 48, 49, 50, 51], which can be found in the second part of this thesis.

Topics for our current and future arbology are:

- Computing repeats of connected subgraphs in a tree. This algorithm has the same principle as the algorithm computing repeats of subtrees in a tree (see Chapter 9 or [51]); instead of the subtree pushdown automaton the tree pattern pushdown automaton is used. The first result on this topic has been presented in [56], a detailed paper is currently under preparation. This result has not an equivalent known solution by means of tree automata.
- Multiple matching for given tree patterns. This algorithm is again presented as an analogy to stringology pattern matching automaton [63]. The first result on this topic has been presented in [56], and a detailed paper, which is an extension of [34], is currently under preparation [33] and should be submitted soon. The results directly correspond to the solutions by means of tree automata [18, 20] in the sense of time and space complexities.
- Constructing deterministic subtree and tree pattern oracle pushdown automata. In analogy with string factor oracle automaton [5], these automata use less memory than the original subtree and tree pattern pushdown automata, respectively, and accept certain additional subtrees and tree patterns, respectively. Our first result on this topic is to be presented in [69].
- Approximate subtree pattern matching. This problem includes three possible operations on trees: renaming a node in a tree, inserting a subtree into a tree and deleting a subtree from a tree. For details of the definition of these three operations see [13]. Our first result is to be presented in [33]. Since the arity of nodes can change as a

result of these operation, the bar notation of trees defined in Chapter 5 is used.
- Approximate tree indexing. Our first result on this topic is to be presented in [77].
- Nonlinear tree pattern matching. For details of the definition of this problem see [70].
- Particular practical algorithms for processing XML data format.
- Processing unordered trees.
- We think that future arbology results might be also contributive to some problems of the theory of pushdown automata, for example to the problem of the determinisation of nondeterministic pushdown automata [7, 66].

For more information on arbology see [9].

Bibliography

[1] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Princiles, Techniques, and Tools.* Addison-Wesley, 1986.

[3] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling.* Prentice-Hall Englewood Cliffs, N.J., 1972.

[4] Henk Alblas and Borivoj Melichar, editors. *Attribute Grammars, Applications and Systems, International Summer School SAGA, Prague, Czechoslovakia, June 4-13, 1991, Proceedings*, volume 545 of *Lecture Notes in Computer Science.* Springer, 1991.

[5] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Factor oracle: A new structure for pattern matching. In Pavelka et al. [68], pages 295–310.

[6] Rajeev Alur. Marrying words and trees. In Diekert et al. [27], page 5.

[7] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In Babai [11], pages 202–211.

[8] Alberto Apostolico and Franco P. Preparata. Optimal off-line detection of repetitions in a string. *Theor. Comput. Sci.*, 22:297–315, 1983.

[9] Arbology www pages. Available on: `<http://www.arbology.org/>`, 2010. February 2010.

[10] John Aycock, R. Nigel Horspool, Jan Janousek, and Borivoj Melichar. Even faster generalized lr parsing. *Acta Inf.*, 37(9):633–651, 2001.

[11] László Babai, editor. *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004.* ACM, 2004.

[12] J. Berstel. *Transductions and Context-Free Languages.* Teubner StudienbÄucher, Stuttgart, 1979.

[13] P. Bille. *Pattern Matching in Trees and Strings.* PhD thesis, FIT University of Copenhagen, Copenhagen, 2008.

[14] Anselm Blumer, J. Blumer, David Haussler, Andrzej Ehrenfeucht, M. T. Chen, and Joel I. Seiferas. The smallest automaton recognizing the subwords of a text. *Theor. Comput. Sci.*, 40:31–55, 1985.

[15] Gerth Stølting Brodal, Rune B. Lyngsø, Christian N. S. Pedersen, and Jens Stoye. Finding maximal pairs with bounded gap. In Crochemore and Paterson [25], pages 134–149.

[16] David R. Chase. An improvement to bottom-up tree pattern matching. In *POPL*, pages 168–177, 1987.

[17] David R. Chase. An improvement to bottom-up tree pattern matching. In *ACM Symp. POPL*, pages 168–177, 1987.

[18] L. Cleophas. *Tree Algorithms. Two Taxonomies and a Toolkit.* PhD thesis, Technische Universiteit Eindhoven, Eindhoven, 2008.

[19] Richard Cole, Ramesh Hariharan, and Piotr Indyk. Tree pattern matching and subset matching in deterministic ($\log^3$)-time. In *SODA*, pages 245–254, 1999.

[20] 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.

[21] M. Crochemore and C. Hancart. Automata for matching patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 2 Linear Modeling: Background and Application, chapter 9, pages 399–462. Springer–Verlag, Berlin, 1997.

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

[23] Maxime Crochemore. An optimal algorithm for computing the repetitions in a word. *Inf. Process. Lett.*, 12(5):244–250, 1981.

[24] Maxime Crochemore. Transducers and repetitions. *Theor. Comput. Sci.*, 45(1):63–86, 1986.

[25] Maxime Crochemore and Mike Paterson, editors. *Combinatorial Pattern Matching, 10th Annual Symposium, CPM 99, Warwick University, UK, July 22-24, 1999, Proceedings*, volume 1645 of *Lecture Notes in Computer Science.* Springer, 1999.

[26] Pierre Deransart and Martin Jourdan, editors. *Attribute Grammars and their Applications, International Conference WAGA, Paris, Fance, September 19-21, 1990, Proceedings*, volume 461 of *Lecture Notes in Computer Science.* Springer, 1990.

[27] Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors. *Computer Science - Theory and Applications, Second International Symposium on Computer Science in Russia, CSR 2007, Ekaterinburg, Russia, September 3-7, 2007, Proceedings*, volume 4649 of *Lecture Notes in Computer Science.* Springer, 2007.

[28] Moshe Dubiner, Zvi Galil, and Edith Magen. Faster tree pattern matching. *J. ACM*, 41(2):205–213, 1994.

[29] Joost Engelfriet. *Tree automata and tree grammars.* University of Aarhus, 1975.

[30] Christian Ferdinand, Helmut Seidl, and Reinhard Wilhelm. Tree automata for code selection. *Acta Inf.*, 31(8):741–760, 1994.

[31] T. Flouri, J. Janousek, and B. Melichar. Tree pattern matching by pushdown automata. Available on: `<http://www.dcs.kcl.ac.uk/events/LSD&LAW09/>`, King's College London, London, 2009. London Stringology Days 2009 Conference presentations.

[32] T. Flouri, J. Janoušek, and B. Melichar. Tree pattern matching by deterministic pushdown automata. In M. Ganzha and M. Paprzycki, editors, *Proceedings of the IMCSIT, Vol. 4*, pages 659–666. IEEE Computer Society Press, 2009.

[33] T. Flouri, J. Janoušek, and B. Melichar. Approximate subtree matching by pushdown automata. presentation, to appear in London strigology days conference 2010,, 2010.

[34] T. Flouri, J. Janoušek, and B. Melichar. Subtree matching by pushdown automata. accepted for publication in Computer Science and Information Systems, ComSIS Consortium, 2010.

[35] Christopher W. Fraser, David R. Hanson, and Todd A. Proebsting. Engineering a simple, efficient code-generator generator. *LOPLAS*, 1(3):213–226, 1992.

[36] Christopher W. Fraser, Robert R. Henry, and Todd A. Proebsting. Burg: fast optimal instruction selection and tree parsing. *SIGPLAN Notices*, 27(4):68–76, 1992.

[37] 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.

[38] Ferenc Gécseg and Magnus Steinby. *Tree Automata.* Akadéniai Kiadó, Budapest, Hungary, 1984.

[39] Ferenc Gécseg and Magnus Steinby. *Tree Automata.* Akadéniai Kiadó, Budapest, Hungary, 1984.

[40] Alan Gibbons and Wojciech Rytter. *Efficient parallel algorithms.* Cambridge University Press, New York, NY, USA, 1988.

[41] R. Steven Glanville and Susan L. Graham. A new method for compiler code generation. In *POPL*, pages 231–240, 1978.

[42] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997.

[43] C. Hemerik and Joost-Pieter Katoen. Bottom-up tree acceptors. *Sci. Comput. Program.*, 13(1):51–72, 1989.

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

[45] J. Holub. Finite automata in stringology. Habilitation Thesis, CTU, Prague, 2007.

[46] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley, Boston, 2nd edition, 2001.

[47] J. Janousek and B. Melichar. Subtree and tree pattern pushdown automata for trees in prefix notation. Available on: <http://www.dcs.kcl.ac.uk/events/LSD&LAW09/>, King's College London, London, 2009. London Stringology Days 2009 Conference presentations.

[48] Jan Janoušek. String suffix automata and subtree pushdown automata. In Jan Holub and Jan Žďárek, editors, *Proceedings of the Prague Stringology Conference 2009*, pages 160–172, Czech Technical University in Prague, Czech Republic, 2009. Available on: <http://www.stringology.org/event/2009>.

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

[50] J. Janoušek and B. Melichar. Subtree and tree pattern pushdown automata for trees in prefix notation. submitted for publication to Acta Informatica, 2009.

[51] J. Janoušek and B. Melichar. Finding repeats of subtrees in a tree in linear time and space. in preparation, 2010.

[52] Tsutomu Kamimura and Giora Slutzki. Parallel and two-way automata on directed ordered acyclic graphs. *Information and Control*, 49(1):10–51, 1981.

[53] Nils Klarlund, Niels Damgaard, and Michael I. Schwartzbach. Yakyak: parsing with logical side constraints. In Rozenberg and Thomas [73], pages 286–301.

[54] Hans Hermann Kron. *Tree templates and subtree transformational grammars.* PhD thesis, 1975.

[55] Ludek Kucera and Antonín Kucera, editors. *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*. Springer, 2007.

[56] London stringology days 2009 conference presentations. Available on: <http://www.dcs.kcl.ac.uk/events/LSD\&LAW09/>, King's College London, London, 2009.

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

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

[59] Michael G. Main and Richard J. Lorentz. An o(n log n) algorithm for finding all repetitions in a string. *J. Algorithms*, 5(3):422–432, 1984.

[60] B. Melichar. Repetitions in text and finite automata. In *Cleophas, L., Watson, B.W. (Eds.) Proceedings of the Eindhoven FASTAR Days 2004, TU Eindhoven*, pages 1–46, 2004.

[61] B. Melichar. Repetitions in text and finite automata. In L. Cleophas and B.W. Watson, editors, *Proceedings of the Eindhoven FASTAR Days 2004*, pages 1–46, TU Eindhoven, The Netherlands, 2004.

[62] B. Melichar. Arbology: Trees and pushdown automata. to appear in Proceedings of Language and Automata Theory and Applications (LATA 2010), invited speaker, 2010.

[63] B. Melichar, J. Holub, and J. Polcar. Text searching algorithms. Available on: `<http://stringology.org/athens/>`, 2005. release November 2005.

[64] B. Melichar and J. Janousek. Repeats in trees and pushdown automata. Available on: `<http://www.dcs.kcl.ac.uk/events/LSD&LAW09/>`, King's College London, London, 2009. London Stringology Days 2009 Conference presentations.

[65] B. Melichar, J. Janousek, and Flouri T. Arbology - basic notions. Available on: `<http://www.dcs.kcl.ac.uk/events/LSD&LAW09/>`, King's College London, London, 2009. London Stringology Days 2009 Conference presentations.

[66] Dirk Nowotka and Jirí Srba. Height-deterministic pushdown automata. In Kucera and Kucera [55], pages 125–134.

[67] Jan pascal Van Best. Tree-walking automata and monadic second order logic, 1999. MSc. Thesis, Leiden University.

[68] Jan Pavelka, Gerard Tel, and Miroslav Bartosek, editors. *SOFSEM '99, Theory and Practice of Informatics, 26th Conference on Current Trends in Theory and Practice of Informatics, Milovy, Czech Republic, November 27 - December 4, 1999, Proceedings*, volume 1725 of *Lecture Notes in Computer Science*. Springer, 1999.

[69] M. Plicka, J. Janoušek, and B. Melichar. Oracle subtree pushdown automaton. presentation, to appear in London strigology days conference 2010,, 2010.

[70] R. Ramesh and I. V. Ramakrishnan. Nonlinear pattern matching in trees. *J. ACM*, 39(2):295–316, 1992.

[71] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages.* Springer–Verlag, Berlin, 1997.

[72] G. Rozenberg and A. Salomaa, editors. *Vol. 1: Word, Language, Grammar, Handbook of Formal Languages.* Springer–Verlag, Berlin, 1997.

[73] Grzegorz Rozenberg and Wolfgang Thomas, editors. *Developments in Language Theory, Foundations, Applications, and Perspectives, Aachen, Germany, 6-9 July 1999.* World Scientific, 2000.

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

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

[76] Bill Smyth. *Computing Patterns in Strings*. Addison-Wesley-Pearson Education Limited, Essex, England, 2003.

[77] J. Stoklasa, J. Janoušek, and B. Melichar. Subtree pushdown automata for trees in bar notation. presentation, to appear in London strigology days conference 2010,, 2010.

[78] Leslie G. Valiant and Mike Paterson. Deterministic one-counter automata. In *Automaten theorie und Formale Sprachen*, pages 104–115, 1973.

[79] K. Wagner and G. Wechsung. *Computational Complexity*. Springer–Verlag, Berlin, 2001.

Part II

Arbology – papers

This part contains papers [49, 48, 50, 51, 34].

Acta Informatica (2009) 46:533–547
DOI 10.1007/s00236-009-0104-9

ORIGINAL ARTICLE

On regular tree languages and deterministic pushdown automata

Jan Janoušek · Bořivoj Melichar

Received: 4 June 2008 / Accepted: 13 August 2009 / Published online: 1 September 2009

Abstract The theory of formal string languages and of formal tree languages are both important parts of the theory of formal languages. Regular tree languages are recognized by finite tree automata. Trees in their postfix notation can be seen as strings. This paper presents a simple transformation from any given (bottom-up) finite tree automaton recognizing a regular tree language to a deterministic pushdown automaton accepting the same tree language in postfix notation. The resulting deterministic pushdown automaton can be implemented easily by an existing parser generator because it is constructed for an LR(0) grammar, and its size directly corresponds to the size of the deterministic finite tree automaton. The class of regular tree languages in postfix notation is a proper subclass of deterministic context-free string languages. Moreover, the class of tree languages which are in their postfix notation deterministic context-free string languages is a proper superclass of the class of regular tree languages.

1 Introduction

The theory of formal string (or word) languages [1,25,37] and the theory of formal tree languages [10,11,15,20,21] have been extensively studied and developed since the 1950s and 1960s, respectively. Both the theories are important parts of the theory of formal languages

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.

J. Janoušek (✉)
Department of Computer Science, Faculty of Information Technologies,
Czech Technical University in Prague, Kolejni 550/2,
160 00 Praha 6, Czech Republic
e-mail: janousej@fel.cvut.cz; Jan.Janousek@fit.cvut.cz

B. Melichar
Department of Computer Science and Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Karlovo nám. 13, 121 35 Prague 2, Czech Republic
e-mail: melichar@fel.cvut.cz

[38] and describe fundamentals for many computer applications. The theory of formal string languages and its models of computation even represent the basic and the largest part of the theory of formal languages. Elements of string and tree languages are strings and trees, respectively. Models of computation of the theory of string languages are finite string automata, pushdown string automata, linear bounded automata and Turing machines, whereas models of computation of the theory of tree languages are various kinds of tree automata. Some of the classes of tree and string languages are:

- Regular tree languages, which are recognized by finite tree automata (FTAs) and generated by regular tree grammars.
- Regular string languages, which are accepted by finite string automata and generated by regular string grammars.
- Context-free string languages, which are accepted by pushdown string automata (PDAs) and generated by context-free string grammars (CFGs). Further, context-free languages accepted by deterministic PDAs are called deterministic context-free string languages. It holds that there exist context-free string languages which are not deterministic, i.e. which cannot be accepted by a deterministic PDA.

The formalisms related to the same class of languages are always mutually transformable, and algorithms of these transformations are well-known. This paper deals with tree languages and with the deterministic PDAs. We show that any FTA can also be simply transformed to a deterministic PDA, and we discuss some related properties and issues, including basic demonstrating examples.

In the further text we will omit word "string" when referencing to string languages, string automata or string grammars. We will use word "tree" whenever referencing to tree languages, tree automata and tree grammars.

Although FTAs were created originally as a natural extension of finite automata, regular tree languages are also strongly related to context-free languages. Most important known relationships between regular tree languages and context-free languages are represented by the following three properties [10,11,21]:

First, the set of derivation trees of a context-free language is a regular tree language.

Second, a regular tree language yield, which is given by the concatenation of leaves of the trees, is a context-free language and, conversely, each context-free language is a regular tree language yield.

Third, there exists a regular tree language which is not the set of derivation trees of a context-free language.

There are two kinds of FTAs according to the direction in which the trees are processed: bottom-up (also called frontier-to-root, or this direction is omitted in the name) and top-down (also called root-to-frontier). FTAs can be deterministic or nondeterministic. Deterministic top-down FTAs are strictly less powerful than the other kinds which are all equally powerful and can recognize every regular tree language. This means that every (bottom-up) nondeterministic FTA can be transformed to an equivalent (bottom-up) deterministic FTA recognizing the same tree language.

An FTA is commonly implemented as a program with recursive functions and appropriate data structures which recursively traverses the tree and evaluates FTA states on the tree [10,16]. It is clear that the deterministic version of the tree automaton is more suitable for an effective implementation than its nondeterministic version.

Trees can also be seen as strings, for example in their prefix (also called preorder) or postfix (also called postorder) notation. We note that prefix or postfix notation of a tree can be obtained by its prefix or postfix traversing, respectively, and that many of the existing algorithms on trees process the trees by prefix or postfix traversing. This paper presents a simple transformation of a given (bottom-up) FTA $\mathcal{A}$ which recognizes a regular tree language L to the deterministic PDA which accepts L in postfix notation. This transformation is presented in the following way: First, a CFG $G_{\mathcal{A}}$ generating L in postfix notation is created. The created CFG $G_{\mathcal{A}}$ is in Reversed Greibach Normal Form and its construction is straightforward: each state of the FTA $\mathcal{A}$ corresponds to one nonterminal symbol of the CFG $G_{\mathcal{A}}$ and each transition rule of the FTA $\mathcal{A}$ corresponds to one rule of the CFG $G_{\mathcal{A}}$. Second, a PDA working in bottom-up fashion is constructed for the CFG $G_{\mathcal{A}}$. The constructed PDA behaves as a (generalised) LR(0) parser for the CFG $G_{\mathcal{A}}$ whose reduce operations have been precomputed beforehand, and therefore it reads one symbol on every transition. We note that this optimization of the (generalised) LR(0) parser by precomputing reductions beforehand was also used in [7]. The size of the constructed PDA directly corresponds to the given FTA: The constructed PDA has just one state and each of its pushdown symbols, except the initial pushdown symbol, corresponds to one state of the given FTA and each of its transition rules, except transition rules operating with the initial pushdown symbol, corresponds to one transition rule of the given FTA. If the given FTA $\mathcal{A}$ is deterministic, then the created CFG $G_{\mathcal{A}}$ is an LR(0) grammar and the resulting PDA is also deterministic. Otherwise, if the given FTA $\mathcal{A}$ is not deterministic, then the created CFG $G_{\mathcal{A}}$ is not an LR(0) grammar and the resulting PDA is also not deterministic. We assume that the given FTA $\mathcal{A}$ to be transformed is deterministic, because any nondeterministic FTA can be transformed to the equivalent deterministic FTA.

The contributions of this paper are:

1. Simple transformation from an FTA to a deterministic PDA, which contributes to a better understanding of the theories of regular tree languages and context-free languages and gives further possibilities to transform theoretical and practical results between the two theories. For example, the transformation allows us to transform any FTA solution of a tree related problem to the equivalent solution described by a deterministic PDA. The construction of the PDA is described by Definition 2 in the third section. Also, the presented theory clarifies the theoretical background for related results of specific problems mentioned in the fifth section.
2. The resulting deterministic PDA represents a simple and fundamental model for effective implementation of FTA (on condition that the access to input trees in postfix notation is suitable). Furthermore, the LR(0) grammar that can be created for any FTA can be directly used as the input of one of the existing deterministic PDA implementing and very well developed bottom-up parser generators, such as bison [8] or yacc [28] (see [2,25] for the use of yacc-like parser generators). The creation of the grammar is described by Definition 1 in the third section.
3. The property of regular tree languages that the class of regular tree languages in postfix notation is a proper subclass of deterministic context-free languages. This and other properties are also formally proved in the third section of this paper.
4. It follows from the possibility of transforming any nondeterministic FTA to an equivalent deterministic FTA that nondeterministic PDAs which can be created by transformation from nondeterministic FTAs can also be transformed to equivalent deterministic PDAs, which are created by transformation from the equivalent deterministic FTAs. This is a contribution to the not yet fully researched problem of how to transform a nondeterministic PDA to an equivalent deterministic PDA (provided that the equivalent deterministic PDA exists).

5. It is demonstrated by Example 3 in the fifth section that the deterministic PDA is a model of computation powerful enough to accept also some tree languages in postfix notation which are beyond the class of regular tree languages. This means that the class of tree languages which are in their postfix notation deterministic context-free languages is a proper superclass of the class of regular tree languages.

The rest of the paper is organised as follows. Basic definitions are given in Sect. 2. The third section describes the transformation of (bottom-up) FTAs to deterministic PDAs, contains a demonstrating example and considers some properties of languages and automata that follow from this transformation. The third section also deals with the size of the resulting deterministic PDA. There are a number of results which study relationships between tree automata and other formalisms or describe solutions of specific, tree related problems both by FTAs in some papers and by PDAs (or PDAs extended with attribute evaluation) in some other papers. Some notes on these results are given in the fourth section. The fifth section demonstrates on an example that deterministic PDAs can also accept some tree languages in postfix notation beyond the class of regular tree languages. The last section is the conclusion.

2 Basic notions

2.1 Ranked alphabet, ground term, tree, finite tree automaton, regular tree language

We use notions from the theory of tree languages similarly as they are defined in [10,11,21].

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

The set $T(\mathcal{F})$ of *ground terms* over the ranked alphabet $\mathcal{F}$ is the smallest set inductively defined in the following way:

1. $\mathcal{F}_0 \subseteq T(\mathcal{F})$,
2. If $p \geq 1$, $f \in \mathcal{F}_p$ and $t_1, \ldots, t_p \in T(\mathcal{F})$, then $f(t_1, \ldots, t_p) \in T(\mathcal{F})$.

Ground terms can be regarded as finite labelled ordered ranked *trees* in prefix notation, where each symbol of $f \in \mathcal{F}$ represents a node with label f, and the arguments are its children. Therefore, in this paper we will use the notions tree and ground term interchangeably.

A *nondeterministic finite (bottom-up) tree automaton* (nondeterministic FTA) over a ranked alphabet $\mathcal{F}$ is a 4-tuple $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$, where Q is a finite set of states, $Q_f \subseteq Q$ is the set of final states, and Δ is a set of transition rules of the following type:

$$f(q_1, q_2, \ldots, q_n) \rightarrow q,$$

where $f \in \mathcal{F}_n$, $n \geq 0$, and $q, q_1, \ldots, q_n \in Q$.

If there are no two rules with the same left-hand side, the tree automaton is called a *deterministic finite tree automaton* (deterministic FTA).

Example 1 A simple example of a deterministic FTA over an alphabet containing constants b and c, and binary symbol a is FTA $\mathcal{A}_1 = (Q, \mathcal{F}, Q_f, \Delta)$, where $Q = \{1, 2, 3\}$,

Fig. 1 Tree t (*left*) and the run of FTA $\mathcal{A}_1$ from Example 1 on tree t (*right*)

$\mathcal{F} = \{a(,), b, c\}$, $Q_f = \{3\}$, and Δ contains these transition rules:

$$b \to 1$$
$$c \to 2$$
$$a(1, 1) \to 3$$
$$a(1, 2) \to 3$$

Finite tree automata over a ranked alphabet $\mathcal{F}$ run on ground terms over $\mathcal{F}$. FTA starts at the leaves and moves upward, associating along a run a state with each subterm inductively. A *run* of an automaton on a ground term is defined as follows: The leaves are mapped to states q by the initial transition rules of the form $a \to q$, where $a \in \mathcal{F}_0$. Now, given a node labelled with $f \in \mathcal{F}_n, n \geq 1$, suppose its children have been mapped into states $q_1, \ldots, q_n$, where $f(q_1, q_2, \ldots, q_n) \to q$, then this node gets mapped to q.

A ground term is *accepted* by a finite tree automaton if there exists a run on the ground term such that its root is mapped to a final state.

The tree language $L(\mathcal{A})$ *recognized* by an FTA $\mathcal{A}$ is the set of all ground terms accepted by the FTA $\mathcal{A}$. Two tree automata are *equivalent* if they recognize the same tree language. A tree language is *recognizable* if it is recognized by some nondeterministic FTA. A tree language is recognisable if and only if it is a regular tree language (see [10,11,21] for the definition of regular tree languages). Furthermore, it holds that each nondeterministic (bottom-up) FTA can be transformed to an equivalent deterministic (bottom-up) FTA.

Example 1, contd. Finite tree automata $\mathcal{A}_1$ recognizes tree language $L(\mathcal{A}_1) = \{a(b, b), a(b, c)\}$. Ground term $t = a(b, c)$ and the run of FTA $\mathcal{A}_1$ on ground term t are illustrated in Fig. 1.

The height of a ground term t, denoted by Height(t) is inductively defined in the following way:

1. Height(t) $= 0$, if $t = a, a \in \mathcal{F}_0$,
2. Height(t) $= 1 + \max(\{\text{Height}(t_i) : i = 1, 2, \ldots, n\})$, if $t = a(t_1, t_2, \ldots, t_n)$, $a \in \mathcal{F}_n$, $n \geq 1$.

We note that there exist also *nondeterministic top-down finite tree automata* and *deterministic top-down finite tree automata*. The class of tree languages recognized by nondeterministic top-down finite tree automata is exactly the class of regular tree languages. However, it is not possible to transform every nondeterministic top-down finite tree automaton to an equivalent deterministic top-down finite tree automaton, which is a strictly less powerful model.

For more details on FTAs and regular tree languages, see [10,11,21].

2.2 Alphabet, language, context-free grammar, pushdown automaton

We use notions from the theory of languages similarly as are defined in [1,25].

Let an *alphabet* be a finite nonempty set of symbols. A *language* over an alphabet T is a set of strings over T. Symbol T^* denotes the set of all strings over T including the empty string, denoted by ε. Set T^+ is defined as $T^+ = T^* \backslash \{\varepsilon\}$. Similarly for string $x \in T^*$, 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 $x^+ = x^* \backslash \{\varepsilon\}$.

A *context-free grammar* (CFG) is a 4-tuple $G = (N, T, P, S)$, where N and T are finite disjoint sets of *nonterminal* and *terminal symbols*, respectively. P is a finite set of *rules* $A \to \alpha$, where $A \in N$, $\alpha \in (N \cup T)^*$. $S \in N$ is the *start symbol*. A CFG $G = (N, T, P, S)$ is said to be in *Reversed Greibach Normal Form* if each rule from P is of the form $A \to \alpha a$, where $a \in T$ and $\alpha \in N^*$.

Relation $\Rightarrow$ is called *derivation*: if $\alpha A \gamma \Rightarrow \alpha\beta\gamma$, $A \in N$, and $\alpha, \beta, \gamma \in (N \cup T)^*$, then rule $A \to \beta$ is in P. Symbols $\Rightarrow^+$, and $\Rightarrow^*$ are used for the *transitive*, and the *transitive and reflexive* closure of $\Rightarrow$, respectively. A *rightmost derivation* $\Rightarrow_{\mathrm{rm}}$ is a relation $\alpha A x \Rightarrow \alpha\beta x$, where $x \in T^*$. Relation $A \Rightarrow^+ \alpha A \beta$ is called *recursion*. *Right recursion* is a $A \Rightarrow^+ \alpha A$. *Hidden-left recursion* is a $A \Rightarrow^+ B\alpha A\beta$, where $B\alpha \Rightarrow^+ \varepsilon$.

The language generated by a CFG G, denoted by $L(G)$, is the set of strings $L(G) = \{w : S \Rightarrow^* w, w \in T^*\}$.

A *context-free language* is a language generated by a CFG.

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

The relation $(q, aw, \beta\alpha) \vdash_M (p, w, \beta\gamma) \subset (Q \times T^* \times G^*) \times (Q \times T^* \times G^*)$ is a *transition* of a pushdown automaton M if $(p, \gamma) \in \delta(q, a, \alpha)$. The k-th power, transitive closure, and transitive and reflexive closure of the relation $\vdash_M$ is denoted $\vdash_M^k$, $\vdash_M^+$, $\vdash_M^*$, respectively. A pushdown automaton M is *deterministic* pushdown automaton (deterministic PDA), if it holds:

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

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

1. *Accepting by final state*:
$$L(M) = \left\{x : \delta(q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \gamma) \wedge x \in T^* \wedge \gamma \in G^* \wedge q \in F\right\},$$
2. *Accepting by empty pushdown store*:
$$L_\varepsilon(M) = \left\{x : (q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \varepsilon) \wedge x \in T^* \wedge q \in Q\right\}.$$

If PDA accepts the language by empty pushdown store then the set F of final states is the empty set. In this paper we will use only PDAs which accept the languages by empty pushdown store.

The class of languages accepted by nondeterministic PDAs is exactly the class of context-free languages. Languages accepted by deterministic PDAs are called *deterministic context-free languages*. There exist context-free languages which are not deterministic, i.e. for which no deterministic PDA can be constructed.

For more details see [1,25].

2.3 LR(0) parsing

Given a string w, an *LR(0) parser* for a CFG $G = (N, T, P, S)$ reads the string w from left to right without any backtracking and is implemented by a deterministic PDA.

A string γ is a *viable prefix* of G if γ is a prefix of $\alpha\beta$, and $S \Rightarrow_{\mathrm{rm}}^{*} \alpha A x \Rightarrow_{\mathrm{rm}} \alpha\beta x$ is a rightmost derivation in G; the string β is called the *handle*. We use the term *complete viable prefix* to refer to $\alpha\beta$ in its entirety. During parsing, each contents of the pushdown store correspond to a viable prefix.

The standard LR(0) parser performs two kinds of transitions:

1. When the contents of the pushdown store correspond to a viable prefix containing an incomplete handle, the parser performs a *shift*, which reads one symbol a and pushes a symbol corresponding to a onto the pushdown store.
2. When the contents of the pushdown store corresponds to a viable prefix ending by the handle β, the parser performs a *reduction* by a rule $A \rightarrow \beta$. The reduction pops $|\beta|$ symbols from the top of the pushdown store and pushes a symbol corresponding to A onto the pushdown store.

A CFG G is $LR(0)$ if the two conditions for G:

(1) $S \Rightarrow_{\mathrm{rm}}^{*} \alpha A w \Rightarrow_{\mathrm{rm}} \alpha\beta w$,
(2) $S \Rightarrow_{\mathrm{rm}}^{*} \gamma B x \Rightarrow_{\mathrm{rm}} \alpha\beta y$, imply that $\alpha A y = \gamma B x$, that is, $\alpha = \gamma$, $A = B$, and $x = y$.

If the CFG G is not an LR(0) grammar, then the PDA constructed as an LR(0) parser contains *conflicts*, which means the next transition to be performed cannot be determined according to the contents of the pushdown store only.

For CFGs without hidden-left and right recursions the number of consecutive reductions between the shifts of two adjacent symbols cannot be greater than a constant, and therefore the LR(0) parser for such a grammar can be optimized by precomputing all its reductions beforehand. Then, the optimized resulting LR(0) parser reads one symbol on each of its transition [7]. In this paper, none of CFGs in the examples contains the above-mentioned recursions and the presented deterministic PDA can be thought of as LR(0) parser optimized with the precomputed reductions.

For more details on LR parsing, see [1,2].

3 Transformation of a (bottom-up) finite tree automaton to an (extended) deterministic pushdown automaton

In this section we present a simple transformation of a given FTA $\mathcal{A}$ to a deterministic PDA which accepts $L(\mathcal{A})$ in postfix notation. The transformation is presented in the following way: First, a CFG $G_{\mathcal{A}}$ generating $L(\mathcal{A})$ in postfix notation is created, which is defined by Definition 1. The created CFG $G_{\mathcal{A}}$ is in Reversed Greibach Normal Form and its construction is simple: each state of FTA $\mathcal{A}$ corresponds to one nonterminal symbol of $G_{\mathcal{A}}$ and each transition rule of $\mathcal{A}$ corresponds to one rule of $G_{\mathcal{A}}$. Second, a PDA working in bottom-up fashion is created for CFG $G_{\mathcal{A}}$, which is defined by Definition 2. The constructed PDA accepts $L(\mathcal{A})$ in postfix notation and behaves as an LR parser for CFG $G_{\mathcal{A}}$ whose reduce operations have been precomputed beforehand and therefore it reads one symbol on every transition. If the given FTA $\mathcal{A}$ is deterministic, then the created CFG $G_{\mathcal{A}}$ is an LR(0) grammar and the resulting PDA is also deterministic. Otherwise, if the given FTA $\mathcal{A}$ is not deterministic, then the created CFG $G_{\mathcal{A}}$ is not an LR(0) grammar and the resulting PDA is also not

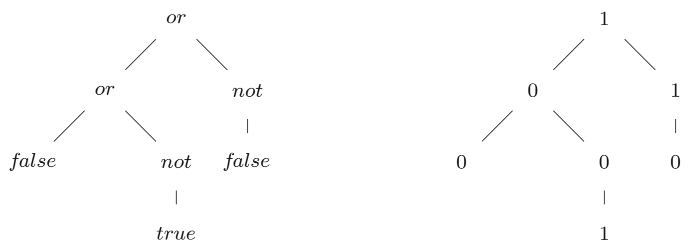


Fig. 2 Tree $t_1 \in L_2$ (*left*) and the run of FTA $\mathcal{A}_2$ from Example 2 on tree t_1 (*right*)

deterministic. We assume that the given FTA $\mathcal{A}$ to be transformed is deterministic because every nondeterministic FTA can be transformed to an equivalent deterministic FTA.

Definition 1 Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be an FTA. Then, a *context-free grammar generating $L(\mathcal{A})$ in postfix notation with appended right marker* $\dashv$ is CFG $G_{\mathcal{A}} = (N, T, P, S')$, where $N = \{S'\} \cup \{S_q : q \in Q\}, T = \mathcal{F}$, and $P = \{S' \to S_q \dashv : q \in Q_f\} \cup \{S_q \to S_{q_1} S_{q_2}, \ldots, S_{q_n}\ a : a(q_1, q_2, \ldots, q_n) \to q \in \Delta\}$.

Let us note that the right marker $\dashv$ is added to the grammar so that the last operation of the PDA would read the right marker and would empty the pushdown store. Acceptance by the empty pushdown store is achieved in this way.

Definition 2 Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be an FTA. Let $G_{\mathcal{A}} = (N, T, P, S')$ be the CFG created according to Definition 1 for $\mathcal{A}$. Then, *PDA accepting $L(\mathcal{A})$ in postfix notation with appended right marker* $\dashv$ is PDA $M_{\mathcal{A}} = (\{q\}, T, G, \delta, q, Z_0, \emptyset)$, where $T = \mathcal{F}$, $G = Q \cup \{Z_0, \dashv\}$, and $\delta = \{\delta(q, \dashv, Z_0\alpha) = (q, \varepsilon) :\ S' \to \alpha \dashv \in P\} \cup \{\delta(q, a, \alpha) = (q, A) : A \to \alpha a \in P, A \neq S'\}$.

The transformation in question is demonstrated on the following example:

Example 2 Consider tree language L_2 of trees representing logical expressions which are equal to the $true$ value and can contain constants $true$ and $false$, unary symbol not, and binary symbol or.

Consider deterministic FTA $\mathcal{A}_2 = (Q, \mathcal{F}, Q_f, \Delta)$, where $L(\mathcal{A}_2) = L_2$, $Q = \{0, 1\}$, $\mathcal{A} = \{true, false, not(), or(,)\}$, $Q_f = \{1\}$, and Δ contains these rules:

$$\begin{aligned} false &\to 0 \\ true &\to 1 \\ not(0) &\to 1 \\ not(1) &\to 0 \\ or(0, 0) &\to 0 \\ or(0, 1) &\to 1 \\ or(1, 0) &\to 1 \\ or(1, 1) &\to 1 \end{aligned}$$

Figure 2 shows a tree $t_1 \in L_1$ and the run of FTA $\mathcal{A}_2$ on tree t_1.

The CFG created according to Definition 1 and generating $L(\mathcal{A}_2)$ in postfix notation is $G_{\mathcal{A}2} = (N, T, P, S')$, where $N = \{S', S_0, S_1\}$, $T = \{true, false, not(), or(,), \dashv\}$, and P

contains the following rules (the rules are written in the same order as their corresponding transition rules of deterministic FTA $\mathcal{A}_2$):

$$
\begin{aligned}
S' &\to S_1 \dashv \\
S_0 &\to false \\
S_1 &\to true \\
S_1 &\to S_0\ not() \\
S_0 &\to S_1\ not() \\
S_0 &\to S_0\ S_0\ or(,) \\
S_1 &\to S_0\ S_1\ or(,) \\
S_1 &\to S_1\ S_0\ or(,) \\
S_1 &\to S_1\ S_1\ or(,)
\end{aligned}
$$

The PDA created according to Definition 2 for $G_{\mathcal{A}2}$ is deterministic PDA $M_{\mathcal{A}2} = (\{q\}, T, G, \delta, q, Z_0, \emptyset)$, where $T = \{true, false, not(), or(,), \dashv\}$, $G = \{Z_0, S_0, S_1\}$, and δ contains the following transition rules:

$$
\begin{aligned}
\delta(q, \dashv, Z_0 S_1) &= (q, \varepsilon) \\
\delta(q, false, \varepsilon) &= (q, S_0) \\
\delta(q, true, \varepsilon) &= (q, S_1) \\
\delta(q, not(), S_0) &= (q, S_1) \\
\delta(q, not(), S_1) &= (q, S_0) \\
\delta(q, or(,), S_0 S_0) &= (q, S_0) \\
\delta(q, or(,), S_0 S_1) &= (q, S_1) \\
\delta(q, or(,), S_1 S_0) &= (q, S_1) \\
\delta(q, or(,), S_1 S_1) &= (q, S_1)
\end{aligned}
$$

Tree automaton $\mathcal{A}_2$ is deterministic because the left-hand side of its every rule is unique. As a consequence, the right-hand side of every rule of grammar $G_{\mathcal{A}2}$ is also unique and, moreover, that right-hand side is not a suffix of the right-hand side of any other rule of grammar $G_{\mathcal{A}2}$, which means the grammar is LR(0). This means that every pop action of pushdown automaton $M_{\mathcal{A}2}$, which reads one symbol on every transition, determines the next transition of PDA $M_{\mathcal{A}2}$ to be performed.

Tree t_1 in postfix notation is string $false\ true\ not()\ or(,)\ false\ not()\ or(,)$. Figure 3 shows the sequence of transitions (trace) performed by deterministic PDA $M_{\mathcal{A}2}$ for tree t_1 in postfix notation with the appended right marker. The top of the pushdown store is on its right hand side. An accept occurs if the automaton has read the whole input string, which is the given tree in postfix notation with the appended right marker, and ends with an empty pushdown store.

As is demonstrated in Example 2, each PDA constructed according to Definition 2 behaves according to the following principle: after reading any subtree in postfix notation the PDA has a symbol S_{q_i} on the top of the pushdown store if and only if a run of the given FTA maps the root of the subtree to the corresponding state q_i.

Since the constructed PDA behaves as an LR(0) parser, it also checks the syntactic structure of the postfix notation of the given tree and reports a possible error as soon as it appears.

State	Pushdown Store	Input
q	Z_0	$false\ true\ not()\ or(,)\ false\ not()\ or(,)\ \dashv$
q	Z_0S_0	$true\ not()\ or(,)\ false\ not()\ or(,)\ \dashv$
q	$Z_0S_0S_1$	$not()\ or(,)\ false\ not()\ or(,)\ \dashv$
q	$Z_0S_0S_0$	$or(,)\ false\ not()\ or(,)\ \dashv$
q	Z_0S_0	$false\ not()\ or(,)\ \dashv$
q	$Z_0S_0S_0$	$not()\ or(,)\ \dashv$
q	$Z_0S_0S_1$	$or(,)\ \dashv$
q	Z_0S_1	$\dashv$
q	ε	ε
accept		

Fig. 3 Trace of deterministic PDA $M_{\mathcal{A}2}$ from Example 2

Finite tree automata are often extended with an output function for their states, for example as is done in FTAs for tree pattern matching, where the output function emits output symbols announcing that particular tree patterns have been found in subject trees (see [10]). We note that the constructed PDA can also be simply extended with the same output function in the following way: when a pushdown symbol S_{q_i} is put onto the top of the pushdown store, the same output symbols as for the corresponding FTA state q_i are emitted. In this way, an equivalent translation PDA can be constructed for any FTA with the output function for the FTA states.

In the rest of this section we formally prove the correctness of the transformation in question, describe some related properties of languages, grammars and automata, and discuss the total size of the resulting deterministic PDA for a regular tree language in postfix notation.

Lemma 1 *Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be an FTA. Let $G_{\mathcal{A}} = (N, T, P, S')$ be the CFG created according to Definition 1 for $\mathcal{A}$. Then, the CFG $G_{\mathcal{A}}$ generates exactly the language $L(\mathcal{A})$ in postfix notation with appended right marker $\dashv$.*

Proof Assume $Q = \{q_1, q_2, \ldots, q_n\}, n \geq 1$, and $N = \{S', S_{q_1}, S_{q_2}, \ldots, S_{q_n}\}$. Let $post(t)$ denote the postfix notation of the ground term t. It holds that:

1. If $t = a$, where $a \in \mathcal{F}_0$, then $\text{post}(t) = a$,
2. If $t = a(t_1, t_2, \ldots, t_m)$, where $m \geq 1, a \in \mathcal{F}_m$ and $t_1, t_2, \ldots, t_m$ are ground terms, then $\text{post}(t) = \text{post}(t_1)\text{post}(t_2), \ldots, \text{post}(t_m)\ a$.

First, we prove the following claim by induction on the height of the ground term:

(*) Given a ground term t over the ranked alphabet $\mathcal{F}$ it holds that $S_{q_i} \Rightarrow^*_{\text{rm}} \text{post}(t)$, $S_{q_i} \in N$, if and only if a run of the FTA $\mathcal{A}$ maps the root of t to the state q_i.

1. If $t = a, a \in \mathcal{F}_0$, then $\text{Height}(t) = 0, \text{post}(t) = a$ and for each transition rule $a \to q_i \in \Delta$ there is the rule $S_{q_i} \to a \in P$, and therefore $S_{q_i} \Rightarrow a = \text{post}(t)$.
2. Assume that claim (*) holds for ground terms $t_1, t_2, \ldots, t_n$, where $n \geq 1$, $\text{Height}(t_1) \leq m, \text{Height}(t_2) \leq m, \ldots, \text{Height}(t_n) \leq m, m \geq 0$. This means there are the derivations $S_{qt1} \Rightarrow^* \text{post}(t_1), S_{qt2} \Rightarrow^* \text{post}(t_2), \ldots, S_{qtn} \Rightarrow^* \text{post}(t_n)$, where the roots of $t_1, t_2, \ldots, t_n$ are mapped to the corresponding states $q_{qt1}, q_{qt2}, \ldots, q_{qtm} \in Q$, respectively, by the FTA $\mathcal{A}$. We have to prove that claim (*) holds also for each term $t = a(t_1, t_2, \ldots, t_n)$, where $\text{Height}(t) = m + 1$:
   For each transition rule $a(q_{qt1}, q_{qt2}, \ldots, q_{qtm}) \to q_i \in \Delta$ there is the rule $S_{q_i} \to S_{qt1}S_{qt2}, \ldots, S_{qtm}\ a \in P$, which means $S_{q_i} \Rightarrow^* \text{post}(t)$.

No other transition rules of the FTA $\mathcal{A}$ exist.

Given a ground term $t \in L(\mathcal{A})$, there is at least one run of the FTA $\mathcal{A}$ such that the root of t is mapped to a state $q_i \in Q_f$. The CFG G contains the rule $S' \Rightarrow S_{q_i} \dashv$ and therefore $S' \Rightarrow^* \text{post}(t) \dashv$.

Given a ground term $t \notin L(\mathcal{A})$, there is no run of the FTA $\mathcal{A}$ such that the root of t is mapped to a state $q_i \in Q_f$. Therefore, the CFG G contains no rule of the form $S' \Rightarrow S_{q_i} \dashv$ and therefore the CFG G does not generate post$(t) \dashv$. □

Theorem 1 *Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be an FTA. Let $G_{\mathcal{A}} = (N, T, P, S')$ be the CFG created according to Definition 1 for $\mathcal{A}$. Let $M_{\mathcal{A}} = (\{q\}, T, G, \delta, q, Z_0, \emptyset)$ be the PDA created according to Definition 2 for $G_{\mathcal{A}}$. Then, the PDA $M_{\mathcal{A}}$ accepts exactly the language $L(\mathcal{A})$ in postfix notation with appended right marker $\dashv$.*

Proof The PDA $M_{\mathcal{A}}$ behaves as the LR(0) parser with the precomputed reductions beforehand for the CFG $G_{\mathcal{A}}$, which is in Reversed Greibach Normal Form. Since the CFG $G_{\mathcal{A}}$ generates exactly the language $L(\mathcal{A})$ in postfix notation with appended right marker $\dashv$, as is proved in Lemma 1, the theorem holds. □

Theorem 2 *Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a deterministic FTA. Let $G_{\mathcal{A}} = (N, T, P, S')$ be the CFG created according to Definition 1 for $\mathcal{A}$. Let $M_{\mathcal{A}} = (\{q\}, T, G, \delta, q, Z_0, \emptyset)$ be the PDA created according to Definition 2 for $G_{\mathcal{A}}$. Then the CFG $G_{\mathcal{A}}$ is an LR(0) grammar and the PDA $M_{\mathcal{A}}$ is deterministic.*

Proof Since the FTA $\mathcal{A}$ is deterministic, the left-hand side of each of its transition rules from Δ is unique. Therefore, the right-hand side of each grammar rule from P is also unique. Each grammar rule from P is of the form $S_q \rightarrow S_{q_1} S_{q_2}, \ldots, S_{q_n}\ a$, where $n = arity(a)$, and therefore no right-hand side of a grammar rule from P is a suffix of the right-hand side of any other grammar rule from P. Furthermore, it holds for any prefix $\alpha\beta$ such that $S' \Rightarrow^*_{\text{rm}} \alpha A w \Rightarrow_{\text{rm}} \alpha\beta w$, $w \in T^*$, that $\alpha\beta \in N^*T$. Therefore, the two conditions:

(1) $S \Rightarrow^*_{\text{rm}} \alpha A w \Rightarrow_{\text{rm}} \alpha\beta w$,
(2) $S \Rightarrow^*_{\text{rm}} \gamma B x \Rightarrow_{\text{rm}} \alpha\beta y$,

imply that $\alpha = \gamma$, $A = B$, and $x = y$. Thus, the CFG $G_{\mathcal{A}}$ is an LR(0) grammar.

Since no right-hand side of a grammar rule from P is a suffix of the right-hand side of any other grammar rule from P, the pop action of each transition of the PDA $M_{\mathcal{A}}$ unambiguously determines the next transition of the PDA $M_{\mathcal{A}}$ to be performed. Thus, the PDA $M_{\mathcal{A}}$ is deterministic. □

Corollary 1 *The class of regular tree languages in postfix notation is a proper subclass of deterministic context-free string languages.*

Proof Any regular tree language L can be recognized by a deterministic FTA $\mathcal{A}$. The deterministic FTA $\mathcal{A}$ can be transformed to the deterministic PDA constructed according to Definition 2, which accepts L in postfix notation with the appended right marker, as is proved in Theorems 1 and 2. Thus, the class of regular tree languages in postfix notation is a subclass of deterministic context-free string languages.

There exist deterministic context-free languages, such as $L = \{a^n : n \geq 0\}$ for an unary symbol a, which are not tree languages in postfix notation. Thus, the subset is a proper subset and the corollary holds. □

It is easy to see that the size of the PDA constructed according to Definition 2 directly corresponds to the size of the given FTA: the constructed PDA has just one state and each of its pushdown symbols, except the initial pushdown symbol Z_0, corresponds to one state of the given FTA. Each transition of the constructed PDA, except transitions operating with the initial pushdown symbol, which read the appended right marker, corresponds to one transition of the given FTA.

Thus, the size of the constructed deterministic PDA directly corresponds to the size of the deterministic FTA. The deterministic FTA, as described in [11], can be constructed for every nondeterministic FTA and has the following size: the transformation from a given nondeterministic FTA to an equivalent deterministic FTA is based on the construction of subsets of the nondeterministic FTA states, by analogy to the transformation from a nondeterministic finite string automaton to an equivalent deterministic finite string automaton [1,25]. Consequently, the number of states of the equivalent deterministic FTA can be exponential in the number of states of the given nondeterministic FTA; however, in practice, it often turns out that many states are not accessible and only the accessible states are considered (see [11]).

4 Notes on some related results and applications

For trees there exist also other models of computation than the finite tree automata. In [39,29] it is shown that so-called pushdown tree-walking automata recognize exactly the class of regular tree languages. The underlying principle of a method of transformation of finite tree automata to pushdown tree-walking automata [39] is similar to the principle which is used in this paper for transformation of finite tree automata to deterministic pushdown automata.

Models of computations for various linearised forms of unranked trees and their relationships to regular tree languages have been extensively studied in many papers: for example, so-called nested words and visibly pushdown languages are studied in [4] and [5], respectively.

As is demonstrated in this paper, any problem which can be solved by an FTA can also be solved by a deterministic PDA. In the rest of this section we discuss some existing results for specific, tree related problems whose solutions are described both by FTAs and by PDAs from the point of view of automata and the theories of formal string and tree languages.

There exists a tool YakYak [30], which is a preprocessor for yacc-compatible generators and serves for generating parsers of the regular tree languages. However, the output of YakYak is not a syntax defining CFG only, but it is an attributed CFG in which the constraints defining regular tree languages are described not by the syntactic rules but by the semantic attribute rules (see also [3,14] for the definition and for further information on attributed grammars). As a result, the parser generated by the YakYak + yacc-compatible generator behaves as a deterministic PDA which recognizes regular tree languages by an extended attribute semantic evaluation.

An example of a problem with solutions that are described by both FTAs and PDAs is the code selection problem in compiler backends. Many code selection methods based on various models of computation have been described. The task here is to cover the intermediate program representation, which is in the form of a tree, by appropriate target machine code instructions, which are represented by tree patterns, and to select the "best possible" such covering. The best possible covering is usually selected according to the result of the evaluation of a cost function, which describes the cost of the machine code instructions. For the purpose of tree covering various versions of tree pattern matching are generally used (see [9,23,24,31] for the basic tree pattern matching methods). A code selection method based on deterministic FTAs can be found in [16], where the cost function is computed by an additional semantic evaluation. On the other hand, [22,33,36] describe the code selection methods based on deterministic PDAs performing the tree pattern matching, where the tree patterns are represented by rules of a CFG, and in this way generally ambiguous and non-LR(0) CFGs are created. Consequently, the LR(0) parsers for those grammars contain conflicts. In [22] these conflicts are resolved by some heuristics; in [33,36] a special construction of a deterministic parser is used, which corresponds to a determinization of the above–mentioned LR(0) parser with the conflicts.

Here we mention also a family of tools BURG, IBURG, etc. (see [18,19] for example), which use another model of computation, so-called tree rewriting systems, for the tree pattern matching in the code selection problem.

Let us note that another code selection method based on the deterministic PDA would result from the transformation of the deterministic FTA from [16] in the way described in this paper (where the evaluation of the cost function would be implemented by an attribute semantic evaluation). In addition, the transformation gives an unambiguous LR(0) grammar, which means the resulting code generator could be implemented easily with the use of an existing (yacc-like) parser generator for that grammar.

5 Beyond the class of regular tree languages

Although non-regular tree languages are beyond the main scope of this paper, it is demonstrated by the following example that the deterministic PDA is a model of computation which is powerful enough to accept also some non-regular tree languages in postfix notation.

Example 3 Given a ranked alphabet $\mathcal{F} = \{f(,), g(), a\}$, consider tree language $L_3 = \{f(g^i(a), g^i(a)) : i > 0\}$, which contains the symmetry between the two children of binary symbol $f(,)$. It is shown in the details in Example 1.2.1 in [11] that L_3 is not a regular tree language, which means it cannot be recognized by an FTA.

L_3 in postfix notation contains strings of the form $ag()^i ag()^i f(,)$, where $i > 0$, which forms a deterministic context-free language. For example, the following LR(0) grammar G_3 generates L_3 in postfix notation with appended right marker $\dashv$. CFG $G_3 = (N, T, P, S')$, where $N = \{S', A\}$, $T = \{f(,), g(), a, \dashv\}$, and P contains the following rules:

$$\begin{aligned} S' &\to S \ \dashv \\ S &\to a\ A\ f(,) \\ A &\to g()\ A\ g() \\ A &\to g()\ a\ g() \end{aligned}$$

Corollary 2 *The class of tree languages which are in their postfix notation deterministic context-free string languages is a proper superclass of the class of regular tree languages.*

Proof The corollary follows from Corollary 1 and Example 3. □

6 Conclusion and future work

Given an FTA recognizing a tree language L, we have described how to create an LR(0) grammar in Reversed Greibach Normal Form which generates the tree language L in postfix notation, and how to construct a deterministic PDA which accepts the tree language L in postfix notation. The presented transformation from the FTA to the deterministic PDA is simple, and the size of the resulting deterministic PDA directly corresponds to the size of the deterministic FTA recognizing the tree language L.

The presented transformation contributes to a better understanding of the theories of tree and string formal languages and allows the transformation of any solution of a problem described by FTA to the equivalent solution described by a deterministic PDA. Also, the created LR(0) grammar can be directly used as the input for the existing and well developed (LA)LR parser generators, such as yacc-like generators.

Further, it is shown that the deterministic PDA as a model of computation is powerful enough to accept also some tree languages beyond the class of regular tree languages.

Regarding specific tree algorithms whose model of computation is the standard deterministic pushdown automaton, recently we have introduced principles of such three new algorithms. First, a new and simple method how to construct tree pattern matchers as deterministic PDAs directly from given tree patterns without constructing finite tree automata as an intermediate product [17,32]. Second, so-called subtree and tree pattern PDAs, which represent a complete index of the tree and the search phase of all occurrences of a subtree or a tree pattern, respectively, of size m is performed in time linear in m and not depending on the size of the tree [26,27,32]. These automata representing indexes of trees are analogous in their properties to the string suffix and factor automata [12,13,34]. Third, a method how to find all repeats of connected subgraphs in trees with the use of subtree or tree pattern PDAs [35,32]. More details on these results and related information can also be found on [6].

Acknowledgments We would like to thank to Loek Cleophas, Tomáš Lusk and Robin Healey for their helpful comments on draft versions of this paper. Anonymous referee very professional comments and information on some related works also have contributed to improving the text significantly.

References

1. Aho, A.V., Ullman, J.D. : The Theory of Parsing, Translation and Compiling, vol. 1: Parsing, vol. 2: Compiling. Prentice Hall, New York (1972)
2. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools. Addison Wesley, Reading, Mass (1986)
3. Alblas, H., Melichar, B. (eds.): Attribute Grammars, Applications and Systems. LNCS 545, Springer, Berlin (1991)
4. Alur, R.: Marrying words and trees. In: 26th ACM symposium on principles of database systems, pp. 233–242. ACM, New York (2007)
5. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: 36th ACM symposium on theory of computing, pp. 202–211. ACM, New York (2004)
6. Arbology www pages: Available on: http://www.arbology.org. Aug (2009)
7. Aycock, J., Horspool, N., Janoušek, J., Melichar, B.: Even faster generalized LR parsing. Acta Inform. **37**(9), 633–651 (2001)
8. Bison-GNU parser generator: Available on: http://www.gnu.org/software/bison/. May (2008)
9. Chase, D.: An improvement to bottom up tree pattern matching. In: Proceedings of 14th annual ACM symposium on principles of programming languages, pp. 168–177 (1987)
10. Cleophas, L.: Tree algorithms. Two taxonomies and a toolkit. Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven (2008)
11. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. Available on: http://www.grappa.univ-lille3.fr/tata (2007). Release 12 Oct 2007
12. Crochemore, M., Hancart, Ch.: Automata for matching patterns. In: Linear Modeling: Backgroung and Application. Handbook of Formal Languages, vol. 2, pp. 399–462. Springer, Berlin, Heidelberg (1997)
13. Crochemore, M., Rytter, W.: Jewels of Stringology. World Scientific, New Jersey (1994)
14. Deransart, P., Jourdan, M. (eds.): Attribute Grammars and Their Applications. LNCS 461, Springer, Berlin (1990)

15. Engelfriet, J.: Tree Automata and Tree Grammars. Lecture Notes. DAIMI FN-IO, University of Aarhus, Denmark (1975)
16. Ferdinand, Ch., Seidl, H., Wilhelm, R.: Tree automata for code selection. Acta Inform. **31**(8), 741–760 (1994)
17. Fleuri, T., Janoušek, J., Melichar, B.: Subtree matching by deterministic pushdown automata. In: Proceedings of WAPL'09 Conference, Mragowo (2009)
18. Fraser, Ch.W., Hanson, D.A., Proebsting, T.A.: Engineering a simple, efficient code generator generator. ACM Lett. Program. Lang. Sys. **1, 3**, 213–226 (1992)
19. Fraser, Ch.W., Henry, R.R., Proebsting, T.A.: BURG: fast optimal instruction selection and tree parsing. ACM SIGPLAN Notices **27**, 68–76 (1992)
20. Gecseg, F.: Tree Automata. Akademiai Kiado, Budapest (1984)
21. Gecseg, F., Steinby, M.: Tree languages. In: Beyond Words. Handbook of Formal Languages, vol. 3, pp. 1–68. Springer, Berlin, Heidelberg (1997)
22. Glanville, R.S., Graham, S.L.: A new approach to compiler code generation. In: Proceedings of 5th ACM Symposium on Principles of Programming Languages, pp. 231–240 (1978)
23. Hemerik, C., Katoen, J.P.: Bottom-up tree acceptors. Sci. Comput. Program. **13**(1), 51–72 (1989)
24. Hoffmann, C.M., O'Donnell, M.J.: Pattern matching in trees. J. ACM **29**(1), 68–95 (1982)
25. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. 2nd edn. Addison-Wesley, New York (2001)
26. Janoušek, J.: String suffix automata and subtree pushdown automata. In: Proceedings of the Prague Stringology Conference 2009, pp. 160–172. Czech Technical University in Prague, Prague (2009)
27. Janoušek, J., Melichar, B.: Subtree and tree pattern pushdown automata for trees in prefix notation. Submitted for publication (2009)
28. Johnson, S.: Yacc-yet another compiler compiler. Tech. Rep. TR 32, AT & T, Bell Laboratories, New Jersey (1975)
29. Kamimura, T., Slutzki, G.: Parallel and two-way automata on directed ordered acyclic graphs. Inform. Control **49**(1), 10–51 (1981)
30. Klarlund, N., Damgaard, N., Schwartzbach, M.I.: YakYak: parsing with logical side constraints. In: Developments in Language Theory, Foundations, Applications, and Perspectives, pp. 286–301. World Scientific, New Jersey (2000)
31. Kron, H.: Tree templates and subtree transformational grammars. Ph.D. thesis, University of California, Santa Cruz (1975)
32. London Stringology Days 2009 Conference presentations: Available on: http://www.dcs.kcl.ac.uk/events/LSD&LAW09/ (2009). King's College London, Feb 2009
33. Madhavan, M., Shankar, P., Siddhartha, R., Ramakrishna, U.: Extending Graham–Glanville techniques for optimal code generation. ACM Trans. Program. Lang. Sys. **22**(6), 973–1001 (2000)
34. Melichar, B., Holub, J., Polcar, J.: Text Searching Algorithms. Available on: http://stringology.org/athens/ (2005). Release Nov 2005
35. Melichar, B., Janoušek, J.: Repeats in Trees by Subtree and Tree Pattern Pushdown Automata. Draft (2009)
36. Shankar, P., Gantait, A., Yuvaraj, A., Madhavan, M.: A new algorithm for linear regular tree pattern matching. Theor. Comput. Sci. **242**(1–2), 125–142 (2000)
37. Sikkel, K., Salomaa, A. (eds.): Word, language, grammar. In: Handbook of Formal Languages, vol. 1. Springer, Berlin, Heidelberg (1997)
38. Sikkel, K., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Berlin, Heidelberg (1997b)
39. van Best, J.P.: Tree-walking automata and monadic second order logic. M.Sc. thesis, Leiden University (1998)

String Suffix Automata and Subtree Pushdown Automata*

Jan Janoušek

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

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

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

1 Introduction

The theory of formal string (or word) languages [1,10,17] and the theory of formal tree languages [4,5,9] are important parts of the theory of formal languages [16]. The most famous models of computation of the theory of tree languages are various kinds of tree automata [4,5,9]. Trees can also be seen as strings, for example in their prefix (also called preorder) or postfix (also called postorder) notation. [11] shows that the deterministic pushdown automaton (PDA) is an appropriate model of computation for labelled ordered ranked trees in postfix notation and that the trees in postfix notation acceptable by deterministic PDA form a proper superclass of the class of regular tree languages, which are accepted by finite tree automata. In the further text we will omit word “string” when referencing to string languages or string automata.

Tree pattern matching is often declared to be analogous to the problem of string pattern matching [4]. One of the basic approaches used for string pattern matching can be represented by finite automata constructed for the text, which means that the text is preprocessed. Examples of these automata are suffix automata [6]. Given a text of size n, the suffix automaton can be constructed for the text in time linear in n. The constructed suffix automaton represents a complete index of the text for all possible suffixes and can find all occurrences of a string suffix and their positions in the text. The main advantage of this kind of finite automata is that the deterministic suffix automaton performs the search phase in time linear in the size of the input subtree and not depending on n.

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

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

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

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

Moreover, the presented subtree PDAs have the following two other properties. First, they are input-driven PDAs [20], which means that each pushdown operation is determined only by the input symbol. Input-driven PDAs can always be determinised [20]. Second, their pushdown symbol alphabets contain just one pushdown symbol and therefore their pushdown store can be implemented by a single integer counter. This means that the presented PDAs can be transformed to counter automata [3,19], which is a weaker and simpler model of computation than the PDA.

The rest of the paper is organised as follows. Basic definitions are given in section 2. Some properties of subtrees in prefix notation are discussed in the third section. The fourth section deals with the subtree PDA. The last section is the conclusion.

2 Basic notions

2.1 Ranked alphabet, tree, prefix notation

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

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

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

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

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

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

A *labelled, ordered, ranked and rooted tree* t over a ranked alphabet $\mathcal{A}$ is an ordered dag $t = (N, R)$ with a special node $r \in A$ called the *root* such that
(1) r has in-degree 0,
(2) all other nodes of t have in-degree 1,
(3) there is just one path from the root r to every $f \in N$, where $f \neq r$,
(4) every node $f \in N$ is labelled by a symbol $a \in \mathcal{A}$ and out-degree of a_f is $Arity(a)$.

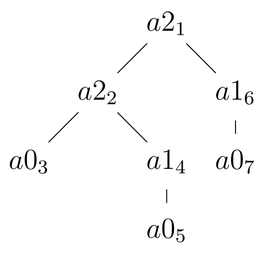
Nodes labelled by nullary symbols (constants) are called *leaves*.

Prefix notation pref(t) of a labelled, ordered, ranked and rooted tree t is obtained by applying the following *Step* recursively, beginning at the root of t:
Step: Let this application of *Step* be to node a_f. If a_f is a leaf, list a and halt. If a_f is not a leaf, let its direct descendants be $a_{f_1}, a_{f_2}, \ldots, a_{f_n}$. Then list a and subsequently apply *Step* to $a_{f_1}, a_{f_2}, \ldots, a_{f_n}$ in that order.

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

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

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



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

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

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

2.2 Alphabet, language, pushdown automaton

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

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

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

Every NFA can be transformed to an equivalent DFA [1,10]. The transformation constructs the states of the DFA as subsets of states of the NFA and selects only such accessible states (ie subsets). These subsets are called *d-subsets*. In spite of the fact that d-subsets are standard sets, they are often written in square brackets ([ ]) instead of in braces ({ }).

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

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

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

A pushdown automaton is *input-driven* if each of its pushdown operations is determined only by the input symbol.
A language L accepted by a pushdown automaton M is defined in two distinct ways:

1. *Accepting by final state:*
$$L(M) = \{x : \delta(q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \gamma) \wedge x \in \mathcal{A}^* \wedge \gamma \in G^* \wedge q \in F\}.$$
2. *Accepting by empty pushdown store:*
$$L_\varepsilon(M) = \{x : (q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \varepsilon) \wedge x \in \mathcal{A}^* \wedge q \in Q\}.$$

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

For more details see [1,10].

2.3 Example of string suffix automaton

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

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

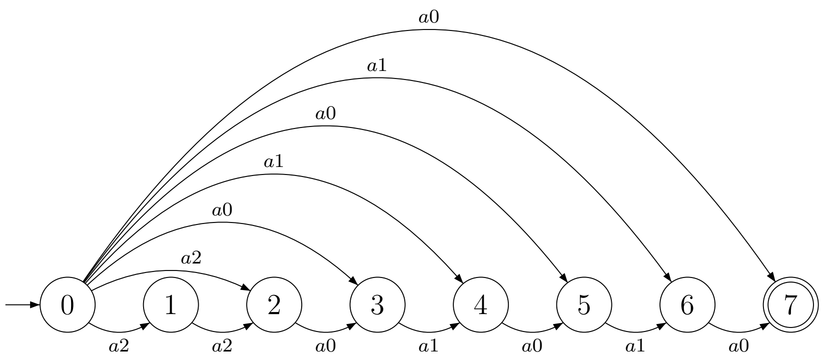


Figure 2. Transition diagram of nondeterministic suffix automaton for string $a2\ a2\ a0\ a1\ a0\ a1\ a0$

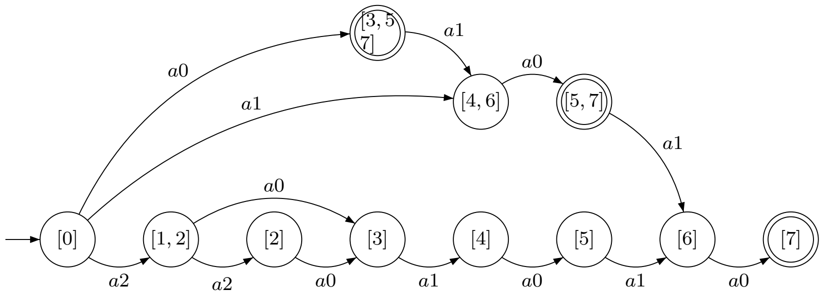


Figure 3. Transition diagram of deterministic suffix automaton for string $a2\ a2\ a0\ a1\ a0\ a1\ a0$

3 Properties of subtrees in prefix notation

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

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

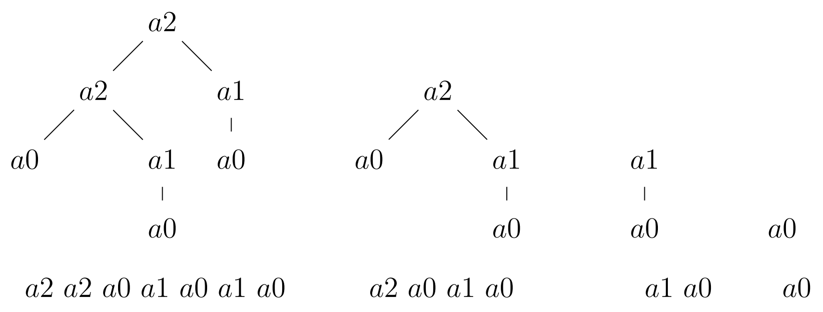


Figure 4. All subtrees of tree t_1 from Example 1, and their prefix notations

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

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

Proof. In [12]. □

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

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

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

Proof. In [12]. □

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

4 Subtree pushdown automaton

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

Definition 7. *Let t and pref(t) be a tree and its prefix notation, respectively. A* subtree pushdown automaton *for pref(t) accepts all subtrees of t in prefix notation.*

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

Algorithm 1. Construction of a PDA accepting a tree t in prefix notation *pref*(t).
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation $\mathit{pref}(t) = a_1a_2\cdots a_n$, $n \geq 1$.
Output: PDA $M_p(t) = (\{0,1,2,\ldots,n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

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

Example 8. A PDA accepting tree t_1 in prefix notation $\mathit{pref}(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1, which has been constructed by Alg. 1, is deterministic PDA $M_p(t_1) = (\{0,1,2,3,4,5,6,7\}, \mathcal{A}, \{S\}, \delta_1, 0, S, \emptyset))$, where the mapping δ_1 is a set of the following transitions:

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

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

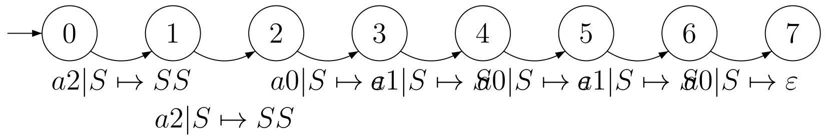

0
1
2
3
4
5
6
7
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$

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

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

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

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

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

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

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

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

Proof. In [12]. □

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

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

Proof. In [12]. □

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

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

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

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

$\delta_2(0, a2, S) = (1, SS)$	
$\delta_2(1, a2, S) = (2, SS)$	$\delta_2(0, a2, S) = (2, SS)$
$\delta_2(2, a0, S) = (3, \varepsilon)$	$\delta_2(0, a0, S) = (3, \varepsilon)$
$\delta_2(3, a1, S) = (4, S)$	$\delta_2(0, a1, S) = (4, S)$
$\delta_2(4, a0, S) = (5, \varepsilon)$	$\delta_2(0, a0, S) = (5, \varepsilon)$
$\delta_2(5, a1, S) = (6, S)$	$\delta_2(0, a1, S) = (6, S)$
$\delta_2(6, a0, S) = (7, \varepsilon)$	$\delta_2(0, a0, S) = (7, \varepsilon)$

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

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

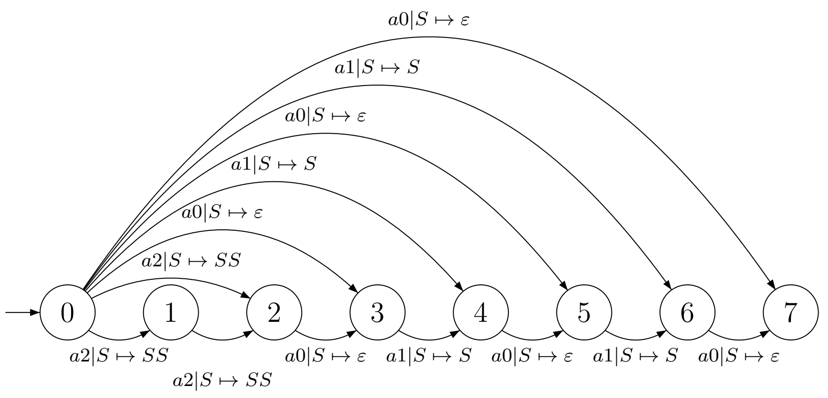

$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a2|S \mapsto SS$
0
1
2
3
4
5
6
7
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$

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

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

Proof. In [12]. □

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

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

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

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

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

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

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

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

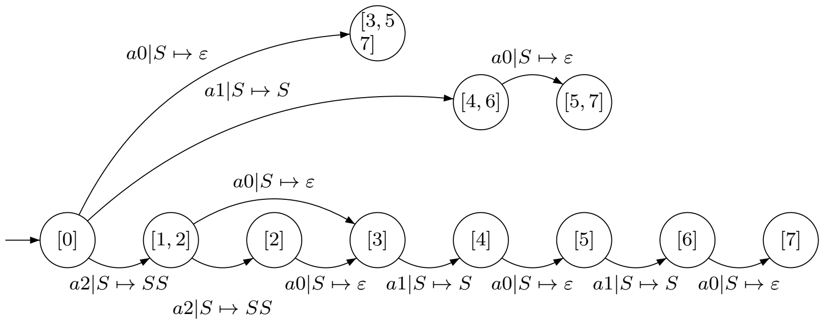

[3,5 7]
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
[4,6]
$a0|S \mapsto \varepsilon$
[5,7]
$a0|S \mapsto \varepsilon$
[0]
[1,2]
[2]
[3]
[4]
[5]
[6]
[7]
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$

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

Fig. 9 shows the sequence of transitions (trace) performed by deterministic subtree PDA $M_{dps}(t_1)$ for an input subtree *st* in prefix notation *pref* $(st) = a_1a_0$. The accepting state is $[5,7]$, which means there are two occurrences of the input subtree *st* in tree t_1 and their rightmost leaves are nodes $a0_5$ and $a0_7$. □

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

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

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

Proof. In [12]. □

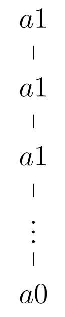
We note that trees with the structure *pref* $(t) = (a1)^{n-1}a0$ represent strings. Such a tree is illustrated in Fig. 10. It can be simply shown that the deterministic subtree PDAs for such trees have the same number of states and transitions as the deterministic suffix automata constructed for *pref*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

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

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

Proof. In [12]. □

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



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

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

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

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

5 Conclusion

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

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

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

References

1. A. V. Aho and J. D. Ullman: *The theory of parsing, translation, and compiling*, Prentice-Hall Englewood Cliffs, N.J.,, 1972.
2. *Arbology www pages*: Available on: `http://www.arbology.org`, July 2009.
3. J. Berstel: *Transductions and Context-Free Languages*, Teubner Studienbucher, Stuttgart, 1979.
4. L. Cleophas: *Tree Algorithms. Two Taxonomies and a Toolkit.*, PhD thesis, Technische Universiteit Eindhoven, Eindhoven, 2008.
5. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi: *Tree automata techniques and applications*. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007, release October, 12th 2007.
6. M. Crochemore and C. Hancart: *Automata for matching patterns*, in Handbook of Formal Languages, G. Rozenberg and A. Salomaa, eds., vol. 2 Linear Modeling: Background and Application, Springer-Verlag, Berlin, 1997, ch. 9, pp. 399–462.
7. M. Crochemore and W. Rytter: *Jewels of Stringology*, World Scientific, New Jersey, 1994.
8. T. Flouri, J. Janoušek, and B. Melichar: *Tree pattern matching by deterministic pushdown automata*. accepted for WAPL 2009 conference, 2009.
9. F. Gecseg and M. Steinby: *Tree languages*, in Handbook of Formal Languages, G. Rozenberg and A. Salomaa, eds., vol. 3 Beyond Words. Handbook of Formal Languages, Springer-Verlag, Berlin, 1997, pp. 1–68.
10. J. E. Hopcroft, R. Motwani, and J. D. Ullman: *Introduction to automata theory, languages, and computation*, Addison-Wesley, Boston, 2nd ed., 2001.
11. J. Janoušek and B. Melichar: *On regular tree languages and deterministic pushdown automata*. accepted for publication in Acta Informatica, Springer, 2009.
12. J. Janoušek and B. Melichar: *Subtree and tree pattern pushdown automata for trees in prefix notation*. submitted for publication, 2009.
13. *London stringology days 2009 conference presentations*: Available on: `http://www.dcs.kcl.ac.uk/events/LSD&LAW09/`, King's College London, London, February 2009.
14. B. Melichar, J. Holub, and T. Polcar: *Text searching algorithms*. Available on: `http://stringology.org/athens/`, 2005, release November 2005.
15. B. Melichar and J. Janoušek: *Repeats in trees by subtree and tree pattern pushdown automata*. draft, 2009.
16. G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.
17. G. Rozenberg and A. Salomaa, eds., *Vol. 1: Word, Language, Grammar, Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.
18. B. Smyth: *Computing Patterns in Strings*, Addison-Wesley-Pearson Education Limited, Essex, England, 2003.
19. L. G. Valiant and M. Paterson: *Deterministic one-counter automata*, in Automaten theorie und Formale Sprachen, 1973, pp. 104–115.
20. K. Wagner and G. Wechsung: *Computational Complexity*, Springer-Verlag, Berlin, 2001.

Noname manuscript No.
(will be inserted by the editor)

Subtree Pushdown Automata and Tree Pattern Pushdown Automata for Trees in Prefix Notation

Jan Janoušek · Bořivoj Melichar

Received: date / Accepted: date

Abstract Two new kinds of acyclic pushdown automata for trees in prefix notation are presented. First, *subtree pushdown automata* accept all subtrees of the tree. Second, *tree pattern pushdown automata* accept all tree patterns which match the tree. The presented pushdown automata are input–driven and therefore can be determinised. Given a tree with n nodes, the deterministic subtree and the deterministic tree pattern pushdown automaton represent a complete index of the tree, and the search phase of all occurrences of a subtree or a tree pattern, respectively, of size m is performed in time linear in m and not depending on n. This is faster than the time of the existing tree pattern matching algorithms, which depends on n. The total size of the deterministic subtree pushdown automaton is linear in n. Although the number of distinct tree patterns which match the tree can be exponential in n, for specific cases of trees the total size of the deterministic tree pattern pushdown automaton is linear in n.

Keywords Tree, Subtree, Tree pattern, Pushdown automaton, Indexing, Tree pattern matching

J. Janoušek
Department of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague
Kolejni 550/2, 160 00 Prague 6, Czech Republic
E-mail: Jan.Janousek@fit.cvut.cz

B. Melichar
Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo nám. 13, 121 35 Prague 2, Czech Republic
E-mail: melichar@fel.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.

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 or various tree finding and tree replacement systems.

The theory of formal string (or word) languages [1, 23, 33] and the theory of formal tree languages [7, 9, 18] are important parts of the theory of formal languages [32]. Elements of string and tree languages are strings and trees, respectively. The main models of computation of the theory of string languages are finite string automata, pushdown string automata (PDAs), linear bounded automata and Turing machines, whereas models of computation of the theory of tree languages are various kinds of tree automata. Trees can also be seen as strings, for example in their prefix (also called preorder) or postfix (also called postorder) notation. We note that the prefix or postfix notation of a tree can be obtained by prefix or postfix traversing, respectively, and that many of the existing algorithms on trees process the trees by prefix or postfix traversing. [25] shows that the deterministic PDA is an appropriate model of computation for labelled ordered ranked trees in postfix notation and that the trees in postfix notation acceptable by deterministic PDA form a proper superclass of the class of regular tree languages, which are accepted by finite tree automata.

In the further text we will omit the word "string" when referring to string languages or string automata.

Tree pattern matching is often declared to be analogous to the problem of string pattern matching [4, 7, 21]. String pattern matching is the problem of finding all occurrences of string patterns and their positions in a text. A model of computation for string pattern matching can be a finite automaton [11]. One of the basic approaches for string pattern matching is represented by the use of finite automata which are constructed for string patterns. In other words, the patterns are preprocessed. Given a text of size n, such finite automata typically perform the search phase in time linear in n (see [11, 12, 29, 35] for a survey).

Another basic approach for string pattern matching is represented by the use of finite automata constructed for the text. In other words, the subject text is preprocessed. Examples of these automata are suffix or factor automata [11, 12, 29, 35]. A factor of a text is defined as a subword of the text. Given a text of size n, the suffix or the factor automaton can be constructed for the text in time linear in n. The constructed suffix or factor automaton represents a complete index of the text for all possible suffixes or factors, respectively, and can find all occurrences of a string suffix or a string factor, respectively, and their positions in the text. The main advantages of this kind of finite automata are:

– Given an input string suffix or string factor of size m, the suffix or factor automaton, respectively, performs the search phase in time linear in m and not depending on n.
– Although the number of possible factors in the text can be quadratic in n, the total size of the suffix or the factor automaton is linear in n.

Thus, suffix and factor automata are advantageous especially in cases when we want to perform string pattern matching very fast and more times in the same text, and they can also be used for large texts. There exist many effective applications of string suffix and factor automata for problems such as data compression, finding repeats in text, backward string matching, and approximate string pattern matching (see [11, 22, 29] for a survey).

Tree pattern matching is defined as the problem of finding all occurrences and their positions of matches of tree patterns in a subject tree. A tree is defined as an acyclic connected graph and a tree pattern represents an acyclic connected subgraph. Although many tree pattern matching methods have been described [6, 7, 8, 13, 14, 16, 17, 20, 21, 27, 31, 34], all of them use the approach where tree patterns are preprocessed and the search phase is performed in time linear to the size of the tree at best. We note that some of these algorithms also preprocess the subject tree. In other words, none of these existing tree pattern matching methods is analogous to the abovementioned approach which is represented by string suffix or factor automata for the string matching problem with all the corresponding analogous advantages.

This paper presents two new kinds of acyclic PDAs for trees in prefix notation, which are analogous to string suffix and factor automata and their properties:

- First, *subtree PDAs* accept all subtrees of the tree.
- Second, *tree pattern PDAs* accept all tree patterns which match the tree. We note that the task of finding occurrences of tree patterns which match a tree is equivalent to the task of tree pattern matching.

Moreover, the subtree PDAs and the tree pattern PDAs can find the rightmost leaves of all occurrences of the input subtree and the input tree pattern, respectively.

An early presentation of the basic idea of subtree and tree pattern PDAs can be found in [26]. A brief description of subtree PDAs, theorems and lemmas regarding subtree PDAs without their proofs can be found in [24]. This paper contains a detailed description, more examples, theorems, lemmas, and their proofs and other related information for both subtree and tree pattern PDAs, in detail.

By analogy with the string suffix or factor automaton, the subtree PDA and tree pattern PDA represent complete indexes of the tree for all possible subtrees and for all tree patterns which match the tree, respectively. Given a tree of size n, the advantages of the deterministic subtree or the deterministic tree pattern PDA are:

- Given an input subtree or input tree pattern of size m, the subtree PDA or the tree pattern PDA, respectively, performs the search phase in time linear in m and not depending on n. This is faster than the time of the existing abovementioned tree pattern matching algorithms, which depends on n. This is also faster than the time that could be theoretically achieved by any standard tree automaton because the standard deterministic tree automaton runs on the subject tree, which means that searching by tree automata can be linear in n at best.
- The number of subtrees of the tree is n and the total size of the deterministic subtree PDA is linear in n.
  Although the number of possible distinct tree patterns which match the tree can be exponential in n, we prove that for specific cases of trees the total size of the deterministic tree pattern pushdown automaton is linear in n. We also show a case of trees for which the total size of the deterministic tree pattern pushdown automaton is quadratic in n. The maximal total size of the deterministic tree pattern PDA in general remains an open problem.

Thus, deterministic subtree PDAs and deterministic tree pattern PDAs are again advantageous especially in cases when we want to perform the searching of subtrees and tree patterns, respectively, very fast and more times in the same tree. Also, they can be used for large trees.

Both kinds of PDAs presented in this paper have the following two other properties:

– They are input–driven PDAs [37], which means that each pushdown operation is determined only by the input symbol. Input–driven PDAs can always be determinised [37]. Moreover, they have their logarithmic-time work–optimal parallel versions, the principles of which are described in [19]. We note that a parallel algorithm is work–optimal if its cost, which is the product of the number of steps and the number of processors, is of the same order as the cost of the best sequential algorithm [19].
– Their pushdown symbol alphabets contain just one pushdown symbol and therefore their pushdown store can be implemented by a single integer counter. This means that the presented PDAs can be transformed to counter automata [3, 36], 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.

The deterministic subtree PDA and the deterministic tree pattern PDA presented in this paper are constructed in the following way: given a tree in prefix notation, a nondeterministic PDA is constructed for this tree. After that, the constructed nondeterministic PDA is transformed to the deterministic PDA. Although a nondeterministic PDA cannot in general be determinised, the constructed nondeterministic subtree and tree pattern PDAs can be determinised because they are input–driven PDAs.

We prove that the prefix notation of any subtree is a substring of the prefix notation of the tree. This important property of trees allows us to use the principles of suffix and factor automata for the construction of subtree PDAs and tree pattern PDAs, respectively. Given a subject tree t in prefix notation $pref(t)$, the nondeterministic subtree PDA has the same states as the nondeterministic string suffix automaton constructed for $pref(t)$ [29]. Moreover, its transitions are also the same but further extended with pushdown operations so that the contents of the pushdown store can determine the ends of subtrees of the subject tree. The resulting deterministic subtree PDA can possibly have fewer states and transitions than the deterministic suffix automaton [11, 12, 29, 35], because some of the states created during the determinisation need not be accessible due to pushdown operations. All the PDAs presented in this paper accept the input tree patterns not by a final state but by empty pushdown store.

A subtree is a special case of a tree pattern, and tree pattern PDAs are an extension of subtree PDAs. This extension adds some new transitions, which represent transitions over subtrees so that the tree pattern PDAs will accept those subsequences of prefix notation which represent subtrees with gaps described by tree patterns that match the subject tree.

The rest of the paper is organised as follows. Basic definitions are given in section 2. Some properties of subtrees in prefix notation are discussed and proved in the third section. The fourth and fifth sections deal with subtree PDAs and tree pattern PDAs, respectively, for trees in prefix notation. The last section is the conclusion.

2 Basic notions

2.1 Ranked alphabet, tree, prefix notation, tree pattern, tree template, tree pattern matching

We define notions on trees similarly as they are defined in [1, 7, 9, 18, 21].

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

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

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

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

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

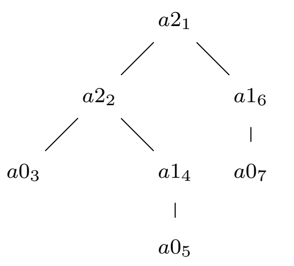
A *labelled, ordered and rooted ranked tree* t over a ranked alphabet $\mathcal{A}$ is an ordered dag $t = (N, R)$ with a special node $r \in A$, called the *root*, such that
(1) r has in-degree 0,
(2) all other nodes of t have in-degree 1,
(3) there is just one path from the root r to every $f \in N$, where $f \neq r$,
(4) every node $f \in N$ is labelled by a symbol $a \in \mathcal{A}$ and the out-degree of a_f is $Arity(a)$.

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

The *prefix notation* $pref(t)$ of a labelled, ordered, ranked and rooted tree t is obtained by applying the following *Step* recursively, beginning at the root of t:
Step: Let this application of *Step* be to node a_f. If a_f is a leaf, list a and halt. If a_f is not a leaf, let its direct descendants be $a_{f_1}, a_{f_2}, \dots, a_{f_n}$. Then list a and subsequently apply *Step* to $a_{f_1}, a_{f_2}, \dots, a_{f_n}$ in that order.

We note that in many papers on the theory of tree languages, such as [7, 9, 18, 21], labelled ordered ranked trees are defined with the use of ordered ranked *ground terms*. Ground terms can be regarded as labelled ordered ranked trees in prefix notation. Therefore, the notions ground term, tree and tree in prefix notation are used interchangeably in these papers.

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



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

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

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

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

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

To define a *tree pattern*, we use a special nullary symbol S, not in $\mathcal{A}$, which serves as a placeholder for any subtree. A tree pattern is defined as a labelled ordered ranked tree over ranked alphabet $\mathcal{A} \cup \{S\}$. By analogy, a tree pattern in prefix notation is defined as a labelled ordered ranked tree over ranked alphabet $\mathcal{A} \cup \{S\}$ in prefix notation. We will assume that the tree pattern contains at least one node from $\mathcal{A}$ (i.e. sole S is not allowed to be a tree pattern). A tree pattern containing at least one symbol S will be called a *tree template*.

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

Example 2 Consider tree $t_1 = (\{a2_1, a2_2, a0_3, a1_4, a0_5, a1_6, a0_7\}, R)$ from Example 1, which is illustrated in Fig. 1. Consider a tree pattern (template) p_1 over $\mathcal{A} \cup \{S\}$
$p_1 = (\{a2_8, S_9, a1_{10}, S_{11}\}, R')$, where R' is a set of the following ordered sequences of pairs:

$$\begin{aligned}&((a2_8, S_9), (a2_8, a1_{10})),\\&((a1_9, S_{11}))\end{aligned}$$

Tree pattern p_1 in prefix notation is string $pref(p_1) = a2\ S\ a1\ S$. Tree pattern p_1 is illustrated in Fig. 2. Tree pattern p_1 has two occurrences in tree t_1 – it matches at nodes $a2_1$ and $a2_2$ of t_1. □

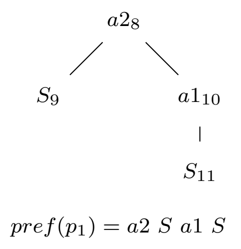


Fig. 2 Tree pattern (template) p_1 from Examples 2 and 9 and its prefix notation

2.2 Alphabet, context–free grammar, language, finite automaton, pushdown automaton

We define notions from the theory of string languages similarly as they are defined in [1, 23].

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

A *context-free grammar* (CFG) is a 4-tuple $G = (N, \mathcal{A}, P, S)$, where N and $\mathcal{A}$ are finite disjoint sets of *nonterminal* and *terminal (input) symbols*, respectively. P is a finite set of *rules* $A \to \alpha$, where $A \in N$, $\alpha \in (N \cup \mathcal{A})^*$. $S \in N$ is the *start symbol*. Relation $\Rightarrow$ is called *derivation*: if $\alpha A\, \gamma \Rightarrow \alpha\beta\gamma$, $A \in N$, and α, β, $\gamma \in (N \cup \mathcal{A})^*$, then rule $A \to \beta$ is in P. Symbols $\Rightarrow^+$, and $\Rightarrow^*$ are used for the *transitive*, and the *transitive and reflexive* closure of $\Rightarrow$, respectively. The language generated by a G, denoted by $L(G)$, is the set of strings $L(G) = \{w : S\ \Rightarrow^* w,\, w \in \mathcal{A}^*\}$.

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

Every NFA can be transformed to an equivalent DFA [1, 23]. The transformation constructs the states of the DFA as subsets of states of the NFA and selects only such accessible states (ie subsets). These subsets are called *d-subsets*. In spite of the fact that d-subsets are standard sets, they are often written in square brackets ([ ]) instead of in braces ({ }).

An (extended) *nondeterministic pushdown automaton* (nondeterministic PDA) is a seven-tuple $M = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$, where Q is a finite set of *states*, $\mathcal{A}$ is an *input alphabet*, G is a *pushdown store alphabet*, δ is a mapping from $Q \times (\mathcal{A} \cup \{\varepsilon\}) \times G^*$ into a set of finite subsets of $Q \times G^*$, $q_0 \in Q$ is an initial state, $Z_0 \in G$ is the initial pushdown 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. In this paper we will write the top of the pushdown store x on its right hand side. The initial configuration of a pushdown automaton is a 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 M is a *deterministic* pushdown automaton (deterministic PDA), if it holds:

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

A pushdown automaton is *input–driven* if each of its pushdown operations is determined only by the input symbol.
A language L accepted by a pushdown automaton M is defined in two distinct ways:

1. *Accepting by final state:*
$$L(M) = \{x : \delta(q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \gamma) \wedge x \in \mathcal{A}^* \wedge \gamma \in G^* \wedge q \in F\}.$$
2. *Accepting by empty pushdown store:*
$$L_\varepsilon(M) = \{x : (q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \varepsilon) \wedge x \in \mathcal{A}^* \wedge q \in Q\}.$$

If PDA accepts the language by empty pushdown store, then the set F of final states is the empty set. In this paper we will use only PDAs which accept the languages by empty pushdown store.

For more details see [1, 23].

2.3 String suffix and factor automata

String suffix and factor automata are finite automata and were introduced in [5, 10] as a mechanism for eliminating redundancy in string suffix trees [11, 12, 29, 35]. 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 [11, 12, 35] suffix and factor automata are defined as such minimal deterministic finite automata. In [29], 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 the prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ of tree t_1 from Example 1, the corresponding nondeterministic suffix automaton is
$FM_{nsuf}(pref(t_1)) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \delta_n, 0, \{7\}))$, where its transition diagram is illustrated in Fig. 3. (For the construction of the nondeterministic suffix automaton see [29].)

After the standard transformation of a nondeterministic suffix automaton to a deterministic one [23], the deterministic suffix automaton for $pref(t_1)$ is
$FM_{dsuf}(pref(t_1)) = (\{[0], [1, 2], [2], [3], [4], [5], [6], [7], [3, 5, 7], [4, 6], [5, 7]\}$,
$\mathcal{A}, \delta_d, 0, \{[7], [3, 5, 7], [5, 7]\}))$, where its transition diagram is illustrated in Fig. 4. $\square$

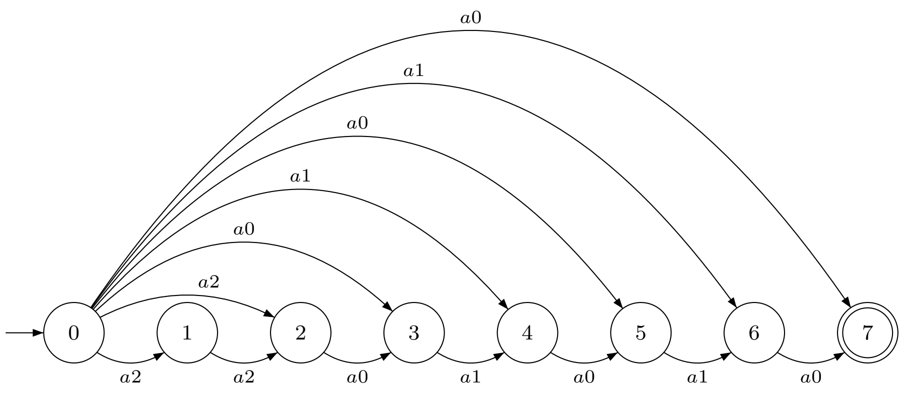


Fig. 3 Transition diagram of the nondeterministic string suffix automaton $FM_{nsuf}(pref(t_1))$ for prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ of tree t_1 from Example 1

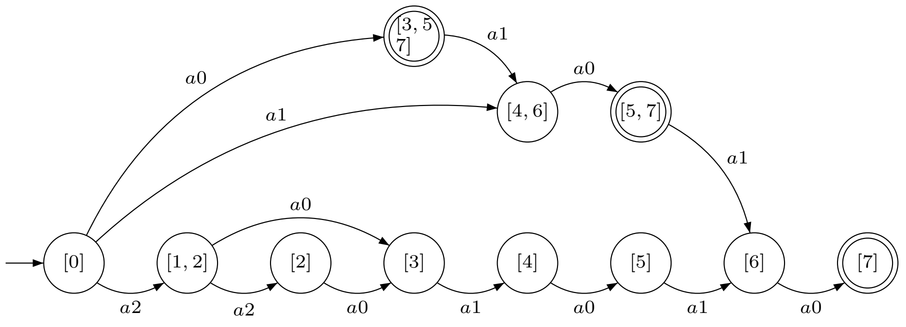


Fig. 4 Transition diagram of the deterministic suffix automaton $FM_{dsuf}(pref(t_1))$ for prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ of tree t_1 from Example 1

3 Properties of subtrees in prefix notation

In this section we prove some general properties of the prefix notation of a tree and of its subtrees. These properties are important for the construction of subtree PDAs and tree pattern PDAs, which are described in the next sections.

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 Fig. 1. Tree t_1 contains only subtrees shown in Fig. 5. $\square$

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

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

Proof By induction on the height of the subtree:

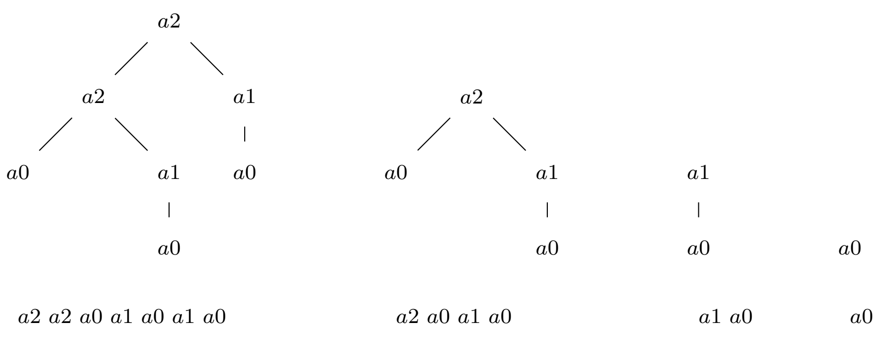


Fig. 5 All subtrees of tree t_1 from Example 5, and their prefix notations

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

Thus, the theorem holds. □

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

Definition 1 Let $w = a_1a_2 \ldots a_m$, $m \geq 1$, be a string over a ranked alphabet $\mathcal{A}$. Then, the *arity checksum* $ac(w) = arity(a_1) + arity(a_2) + \ldots + arity(a_m) - m + 1 = \sum_{i=1}^{m} arity(a_i) - m + 1$.

Theorem 2 *Let $pref(t)$ and w be a tree t in prefix notation and a substring of $pref(t)$, respectively. Then, w is the prefix notation of a subtree of t, if and only if $ac(w) = 0$, and $ac(w_1) \geq 1$ for each w_1, where $w = w_1x$, $x \neq \varepsilon$.*

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

– *If:* By induction on the height of a subtree st, where $w = pref(st)$:
  1. We assume that $Height(st) = 0$, which means we consider the case $w = a$. where $arity(a) = 0$. Then, $ac(w) = 0$. Thus, the claim holds for the case $Height(st) = 0$.

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

– *Only if*: Assume $ac(w) = 0$, and $w = a_1 a_2 \ldots a_m$, where $m \geq 1$, $arity(a_1) = p$. Since $ac(w_1) \geq 1$ for each w_1, where $w = w_1 x$, $x \neq \varepsilon$, none of the substrings w_1 can be a subtree in prefix notation. This means that the only possibility for $ac(w) = 0$ is that w is of the form $w = a\ pref(t_1)\ pref(t_2) \ldots pref(t_p)$, where $p \geq 0$, and $t_1, t_2 \ldots t_p$ are subtrees which are adjacent siblings. In such a case $ac(w) = p + 0 - (p+1) + 1 = 0$.
  No other possibility of the form of w for $ac(w) = 0$ is possible. Thus, the claim holds.

Thus, the theorem holds. □

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

4 Subtree pushdown automata

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

Definition 2 Let t and $pref(t)$ be a tree and its prefix notation, respectively. A *subtree pushdown automaton* for $pref(t)$ accepts all subtrees of t in prefix notation.

From the global point of view, comparing the subtree PDAs with the string suffix automaton, the deterministic subtree PDA constructed for a tree t can have just states and transitions which correspond to states and transitions of the deterministic string suffix automaton constructed for $pref(t)$, where the transitions of the subtree PDA are extended with pushdown operations. The pushdown operations compute the arity checksum, which was defined in the previous section. Moreover, some of the states and the transitions of the string suffix automaton need not be present in the deterministic subtree PDA because the corresponding pushdown operations cannot be performed and therefore such states are not accessible. All PDAs in this paper accept the input by empty pushdown store, which means the arity checksum is 0.

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

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

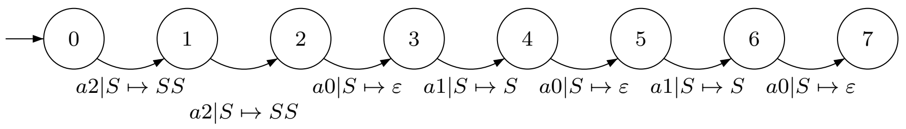

0
1
2
3
4
5
6
7
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$

Fig. 6 Transition diagram of deterministic PDA $M_p(t_1)$ accepting tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 5

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

Fig. 7 Trace of deterministic PDA $M_p(t_1)$ from Example 5 for tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

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

Example 5 A PDA accepting tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1, which has been constructed by Alg. 1, is deterministic PDA $M_p(t_1) = (\{0,1,2,3,4,5,6,7\}, \mathcal{A}, \{S\}, \delta_1, 0, S, \emptyset))$, where the mapping δ_1 is a set of the following transitions:

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

The transition diagram of deterministic PDA $M_p(t_1)$ is illustrated in Fig. 6. In this figure, for each transition rule $\delta_1(p, a, \alpha) = (q, \beta)$ from δ the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$.

Fig. 7 shows the sequence of transitions (trace) performed by deterministic PDA $M_p(t_1)$ for tree t_1 in prefix notation. □

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

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

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

Theorem 3 *Let $M = (\{Q, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$ be an input-driven PDA of which each transition from δ is of the form $\delta(q_1, a, S) = (q_2, S^i)$, where $i = arity(a)$.*
Then, if $(q_3, w, S) \vdash_M^+ (q_4, \varepsilon, S^j)$, then $j = ac(w)$.

Proof By induction on the length of w:

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

   Thus, the theorem holds. □

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

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

Proof By induction on the height of tree t:

1. If tree t has just one node a, where $Arity(a) = 0$, then $Height(t) = 0$, $pref(t) = a$, $\delta(0, a, S) = (1, \varepsilon) \in \delta$, $(0, a, S) \vdash_{M_p(t)} (1, \varepsilon, \varepsilon)$ and the claim holds for that tree.
2. Assume that the claim holds for trees $t_1, t_2, \dots, t_p$, where $p \geq 1$, $Height(t_1) \leq m$, $Height(t_2) \leq m$, ..., $Height(t_p) \leq m$, $m \geq 0$.
   We have to prove that the claim also holds for each tree t such that
   $pref(t) = a\ pref(t_1)pref(t_2) \dots pref(t_p)$, $Arity(a) = p$, and $Height(t) \geq m + 1$:
   Since $\delta(0, a, S) = (1, S^p) \in \delta$, and
   $(0, a\ pref(t_1)pref(t_2) \dots pref(t_p), S)$
   $\vdash_{M_p(t)} (1, pref(t_1)pref(t_2) \dots pref(t_p), S^p)$
   $\vdash^*_{M_p(t)} (i, pref(t_2) \dots pref(t_p), S^{p-1})$
   $\vdash^*_{M_p(t)} \cdots$
   $\vdash^*_{M_p(t)} (j, pref(t_p), S)$
   $\vdash^*_{M_p(t)} (k, \varepsilon, \varepsilon)$,
   the claim holds for that tree.

Thus, the lemma holds. □

We present the construction of the deterministic subtree PDA for trees in prefix notation. The construction consists of two steps. First, a nondeterministic subtree PDA is constructed by Alg. 2. This nondeterministic subtree PDA is an extension of the PDA accepting tree in prefix notation, which is constructed by Alg. 1. Second, the constructed nondeterministic subtree PDA is transformed to the equivalent deterministic subtree PDA. Although a nondeterministic PDA cannot generally be determinised, the constructed nondeterministic subtree PDA is an input–driven PDA and therefore can be determinised [37].

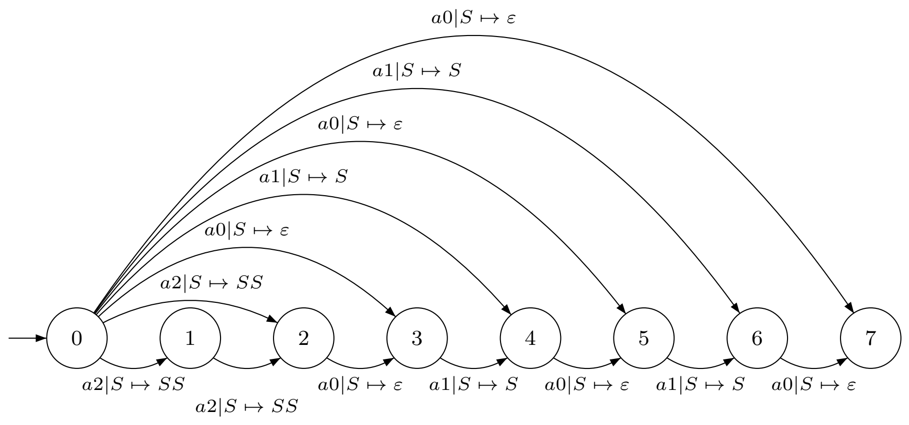

$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a2|S \mapsto SS$
0
1
2
3
4
5
6
7
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$

Fig. 8 Transition diagram of nondeterministic subtree PDA $M_{nps}(t_1)$ for tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 6

Algorithm 2 Construction of a nondeterministic subtree PDA for a tree t in prefix notation $pref(t)$.
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation $pref(t) = a_1a_2 \dots a_n$, $n \geq 1$.
Output: Nondeterministic subtree PDA $M_{nps}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

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

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

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

The transition diagram of nondeterministic PDA $M_{nps}(t_1)$ is illustrated in Fig. 8. Again, in this figure for each transition rule $\delta_2(p, a, \alpha) = (q, \beta)$ from δ_2 the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$.

A comparison of Figs. 8 and 3 shows that the states and the transitions of nondeterministic subtree PDA $M_{nps}(t_1)$ correspond to the states and the transitions, respectively, of the nondeterministic string suffix automaton for $pref(t_1)$; the transitions of the subtree PDA are extended by pushdown operations so that it holds that the number of symbols S in the pushdown store is equal to the corresponding arity checksum. □

We prove the correctness of the constructed subtree PDA.

Theorem 4 *Given a tree t and its prefix notation $pref(t)$, the PDA $M_{nps}(t)$ constructed by Alg. 2 is a subtree PDA for $pref(t)$.*

Proof According to Theorem 1 each subtree in prefix notation is a substring of $pref(t)$. Since the PDA $M_{nps}(t)$ has just states and transitions equivalent to the states and transitions, respectively, of the string suffix automaton, the PDA $M_{nps}(t)$ reads just all the substrings of $pref(t)$.

According to Theorem 3 the PDA $M_{nps}(t)$ behaves such that that if $(q_3, w, S) \vdash^+_{M_{nps}(t)} (q_4, \varepsilon, S^j)$, then $j = ac(w)$. The input is accepted by the empty pushdown store, which agrees with Theorem 2 – input w is the prefix notation of a subtree of t if and only if $ac(w) = 0$, and $ac(w_1) \geq 1$ for each w_1 , where $w = w_1x$, $x \neq \varepsilon$.

Thus, the theorem holds. □

It is known that each nondeterministic input–driven PDA can be transformed to an equivalent deterministic input–driven PDA [37]. To construct deterministic subtree or tree pattern PDAs from their nondeterministic versions we use the transformation described by Alg. 3. This transformation is an extension of the well known transformation of a nondeterministic finite automaton to an equivalent deterministic one [23]. Again, the states of the resulting deterministic PDA correspond to subsets of the states of the original nondeterministic PDA, and these subsets are again called d-subsets. Moreover, the original nondeterministic PDA is assumed to be acyclic with a specific order of states, and Alg. 3 precomputes the possible contents of the pushdown store in particular states of the deterministic PDA according to pushdown operations and selects only those transitions and accessible states of the deterministic PDA for which the pushdown operations are possible. The assumption that the PDA is acyclic results in a finite number of possible contents of the pushdown store. Furthermore, the assumption of the specific order of states allows us to compute these contents of the pushdown store easily in a one-pass way.

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

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

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

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

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

The contents of the pushdown store in particular states are as follows:

$$\begin{array}{ll}
cpds([0]) = \{S\}, & cpds([3,5,7]) = \{\varepsilon\}, \\
cpds([1,2]) = \{SS\}, & cpds([4,6]) = \{S\}, \\
cpds([2]) = \{SSS\}, & cpds([5,7]) = \{\varepsilon\}. \\
cpds([3]) = \{S, SS\}, & \\
cpds([4]) = \{S, SS\}, & \\
cpds([5]) = \{\varepsilon, S\}, & \\
cpds([6]) = \{S\}, & \\
cpds([7]) = \{\varepsilon\}, &
\end{array}$$

The transition diagram of deterministic PDA $M_{dps}(t_1)$ is illustrated in Fig. 9. Again, in this figure for each transition rule $\delta_3(p, a, \alpha) = (q, \beta)$ from δ_3 the edge leading from state p to state q is labelled by the triple of the form $a|\alpha \mapsto \beta$.

We note that there are no transitions leading from states $[3,5,7]$, $[5,7]$ and $[7]$, because the contents of the pushdown store (cpds) in these state is always ε and therefore no transition is possible from these states due to the pushdown operations. This means that deterministic subtree PDA $M_{dps}(t_1)$ has fewer transitions than the deterministic string suffix automaton constructed for $pref(t_1)$ [11, 29, 35], as can be seen by comparing Figs. 4 and 9.

Fig. 10 shows the sequence of transitions (the trace) performed by deterministic subtree PDA $M_{dps}(t_1)$ for an input subtree st in prefix notation $pref(st) = a_1 a_0$. The accepting state is $[5,7]$, which means there are two occurrences of the input subtree st in tree t_1 and their rightmost leaves are nodes $a0_5$ and $a0_7$. □

We prove the correctness of Alg. 3, which constructs the deterministic input–driven PDA for a given acyclic nondeterministic input–driven PDA. The proof is done similarly as the proof of the equivalence of a nondeterministic finite automaton and its corresponding equivalent deterministic finite automaton [1].

Theorem 5 *Given an acyclic input–driven nondeterministic PDA $M_{nx}(t) = (Q, \mathcal{A}, \{S\}, \delta, q_0, S, \emptyset)$, the deterministic PDA $M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', \{q_0\}, S, \emptyset)$ constructed by Alg. 3 is equivalent to PDA $M_{nx}(t)$.*

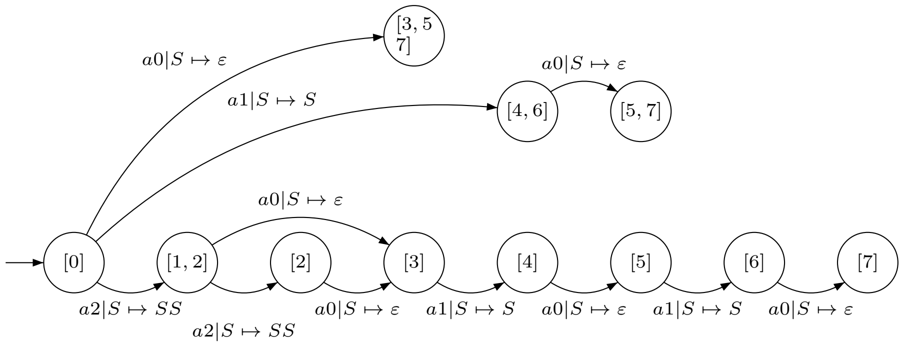


Fig. 9 Transition diagram of deterministic subtree PDA $M_{dps}(t_1)$ for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 7

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

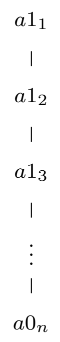
Fig. 10 Trace of deterministic subtree PDA $M_{dps}(t_1)$ from Example 7 for an input subtree st in prefix notation $pref(st) = a1a0$

Proof First, we prove the following claim by induction on i:
(*): $(q'_1, w, S) \vdash^i_{M_{dx}(t)} (q'_2, \varepsilon, S^j)$ if and only if
$q'_2 = \{p : (q, w, S) \vdash^i_{M_{nx}(t)} (p, \varepsilon, S^j)$ for some $q \in q'_1\}$.

1. Assume i=1.
   - *if*: if $(q'_1, a, S) \vdash_{M_{dx}(t)} (q'_2, \varepsilon, S^j)$, then there exists a state $q \in q'_1$, where $(q, a, S) \vdash_{M_{nx}(t)} (p, \varepsilon, S^j)$, $p \in q'_2$.
   - *only if*: if $(q, a, S) \vdash_{M_{nx}(t)} (p, \varepsilon, \beta)$, then for each $q'_1 \in Q'$, where $q \in q'_1$, it holds that $(q'_1, a, S) \vdash_{M_{dx}(t)} (q'_2, \varepsilon, S^j)$, where $p \in q'_2$.
2. Assume that claim (*) holds for $i = 1, 2, \ldots, k$, $k \geq 1$.
   This means that $(q'_1, w, S) \vdash^k_{M_{dx}(t)} (q'_2, \varepsilon, S^j)$ if and only if
   $q'_2 = \{p : (q, w, S) \vdash^k_{M_{nx}(t)} (p, \varepsilon, S^j)$ for some $q \in q'_1\}$. We have to prove that claim (*) also holds for $i = k + 1$.
   - *if*: if $(q'_1, w, S) \vdash^k_{M_{dx}(t)} (q'_2, a, S^l) \vdash_{M_{dx}(t)} (q'_3, \varepsilon, S^j)$, then there exists a state $q \in q'_2$, where $(q, a, S^l) \vdash_{M_{nx}(t)} (p, \varepsilon, S^j)$, $p \in q'_3$.
   - *only if*: if $(q_0, pref(t), S) \vdash^k_{M_{nx}(t)} (q, a, S^l) \vdash_{M_{nx}(t)} (p, \varepsilon, S^j)$, then for each $q'_1 \in Q'$, where $q \in q'_1$, it holds that $(q'_1, a, S^l) \vdash_{M_{dx}(t)} (q'_2, \varepsilon, S^j)$, where $p \in q'_2$.

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

We note that trees with the structure $pref(t) = (a1)^{n-1}a0$ represent strings. Such a tree is illustrated in Fig. 11. It can be simply shown that the deterministic subtree PDAs for such trees have the same number of states and transitions as the deterministic suffix automata constructed for $pref(t)$ and accept the same language.



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

Fig. 11 A tree t_2, which represents a string, and its prefix notation

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

Lemma 2 *Given a tree t with n nodes, the number of distinct subtrees of tree t is equal or smaller than n.*

Proof It is clear that each node of the tree t is the root of a subtree, which gives just n subtrees. Since some subtrees may be equal, the lemma holds. □

At the end of this section we discuss the total size of the constructed deterministic subtree PDA. The deterministic subtree PDA has the only pushdown symbol S, and all its states and transitions correspond to the states and the transitions, respectively, of the deterministic suffix automaton constructed for $pref(t)$. Therefore, the total size of the deterministic subtree PDA cannot be greater than the total size of the deterministic suffix automaton constructed for $pref(t)$. In other words, the total size is linear to the number of nodes of the tree. More precisely, the exact maximal numbers of states and transitions are described by the following theorem.

Theorem 6 *Given a tree t with n nodes and its prefix notation $pref(t)$, the deterministic subtree PDA $M_{dps}(t)$ constructed by Algs. 2 and 3 has just one pushdown symbol, fewer than $N \leq 2n + 1$ states and at most $N + n - 1 \leq 3n$ transitions.*

Proof The deterministic subtree PDA in question may have only states and transitions which correspond to the states and the transitions, respectively, of the deterministic suffix automaton constructed for $pref(t)$. Therefore, the largest possible numbers of states and transitions of the deterministic subtree PDA are the same as those of the deterministic suffix automaton. The numbers of states and transitions of the deterministic suffix automaton are proved in Theorems 6.1 and 6.2 in [12] or in Theorem 5.3.5 in [35]. We note that these proofs are based on the following principle:
Given a substring u, the d-subset of the state in which the deterministic suffix automaton is after reading u is called the *terminator set* of u [35]. It holds for any two

substrings u_1 and u_2 that their terminator sets cannot overlap; in other words, the terminator sets of a deterministic suffix automaton correspond to a tree structure. It has been proved ([12, 35]) that this tree structure is such that the abovementioned numbers of states and transitions hold. $\square$

5 Tree pattern pushdown automata

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, described in the previous section, so that also tree templates would be accepted. New states and transitions, which are used for processing the special nullary symbols S in tree templates, are additionally present in the tree pattern s. The pushdown operations are the same and compute the arity checksum.

Definition 3 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 as follows.

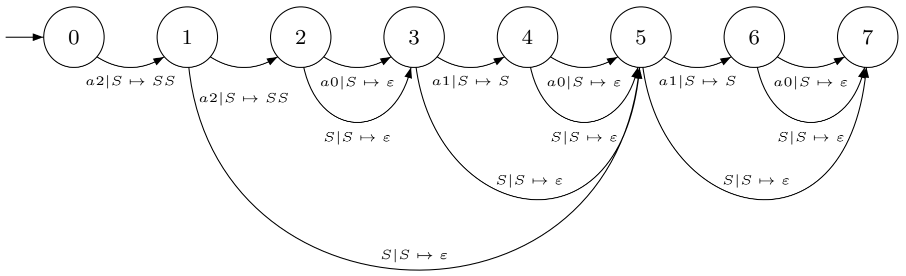
Definition 4 Let t, r and $pref(t)$ be a tree, its root and its prefix notation, respectively. A *treetop pushdown automaton* for $pref(t)$ accepts all tree patterns in prefix notation which have the root r and match the tree t.

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

Algorithm 4 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 $pref(t) = a_1a_2\dots a_n$, $n \geq 1$.
Output: Treetop PDA $M_{pt}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A} \cup \{S\}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

1. Create $M_{pt}(t)$ as $M_p(t)$ by Alg. 1.
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 RightMost 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*. $\square$

The construction of treetop PDA by Alg. 4 is illustrated in the following example.



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

Fig. 12 Transition diagram of deterministic treetop PDA $M_{pt}(t_1)$ for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 8

Example 8 Consider tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1, which is illustrated in Fig. 1. The deterministic treetop PDA, constructed by Alg. 4, is deterministic PDA $M_{pt}(t_1) = (\{0,1,2,3,4,5,6,7\}, \mathcal{A}, \{S\}, \delta_4, 0, S, \emptyset))$, where mapping δ_4 is a set of the following transitions:

$$\begin{array}{lcllcl}
\delta_4(0, a2, S) & = & (1, SS) & & & \\
\delta_4(1, a2, S) & = & (2, SS) & \delta_4(1, S, S) & = & (5, \varepsilon) \\
\delta_4(2, a0, S) & = & (3, \varepsilon) & \delta_3(2, S, S) & = & (3, \varepsilon) \\
\delta_4(3, a1, S) & = & (4, S) & \delta_4(3, S, S) & = & (5, \varepsilon) \\
\delta_4(4, a0, S) & = & (5, \varepsilon) & \delta_4(4, S, S) & = & (5, \varepsilon) \\
\delta_4(5, a1, S) & = & (6, S) & \delta_4(5, S, S) & = & (6, \varepsilon) \\
\delta_4(6, a0, S) & = & (7, \varepsilon) & \delta_4(6, S, S) & = & (7, \varepsilon)
\end{array}$$

The transition diagram of deterministic treetop PDA $M_{pt}(t_1)$ is illustrated in Fig. 12. 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 $a|\alpha \mapsto \beta$.

Deterministic treetop PDA $M_{pt}(t_1)$ has been constructed by Alg. 4 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 $\delta_4(1, S, S) = (5, \varepsilon)$. □

Theorem 7 *Given a tree t and its prefix notation $pref(t)$, the PDA $M_{pt}(t)$ constructed by Alg. 4 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 $M_{pt}(t)$, which is constructed by Alg. 1 and accepts the tree t in prefix notation (see Lemma 1). It holds for new added transitions, 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 matches with Theorem 2 so 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 5 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 $pref(t) = a_1a_2 \dots a_n$, $n \geq 1$.
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 Alg. 4.
2. For each state i, where $2 \leq i \leq n$, create a new transition $\delta(0, a_i, S) = (i, S^{Arity(a_i)})$, where $S^0 = \varepsilon$. □

Example 9 Consider tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1, which is illustrated in Fig. 1. The nondeterministic tree pattern PDA accepting all tree patterns matching tree t_1, which has been constructed by Alg. 5, is nondeterministic PDA $M_{npt}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_5, 0, S, \emptyset))$, where mapping δ_5 is a set of the following transitions:

$\delta_5(0, a2, S) = (1, SS)$		
$\delta_5(1, a2, S) = (2, SS)$	$\delta_5(1, S, S) = (5, \varepsilon)$	$\delta_5(0, a2, S) = (2, SS)$
$\delta_5(2, a0, S) = (3, \varepsilon)$	$\delta_3(2, S, S) = (3, \varepsilon)$	$\delta_5(0, a0, S) = (3, \varepsilon)$
$\delta_5(3, a1, S) = (4, S)$	$\delta_5(3, S, S) = (5, \varepsilon)$	$\delta_5(0, a1, S) = (4, S)$
$\delta_5(4, a0, S) = (5, \varepsilon)$	$\delta_5(4, S, S) = (5, \varepsilon)$	$\delta_5(0, a0, S) = (5, \varepsilon)$
$\delta_5(5, a1, S) = (6, S)$	$\delta_5(5, S, S) = (6, \varepsilon)$	$\delta_5(0, a1, S) = (6, S)$
$\delta_5(6, a0, S) = (7, \varepsilon)$	$\delta_5(6, S, S) = (7, \varepsilon)$	$\delta_5(0, a0, S) = (7, \varepsilon)$

The transition diagram of nondeterministic tree pattern PDA $M_{npt}(t_1)$ is illustrated in Fig. 13. 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 $a|\alpha \mapsto \beta$. □

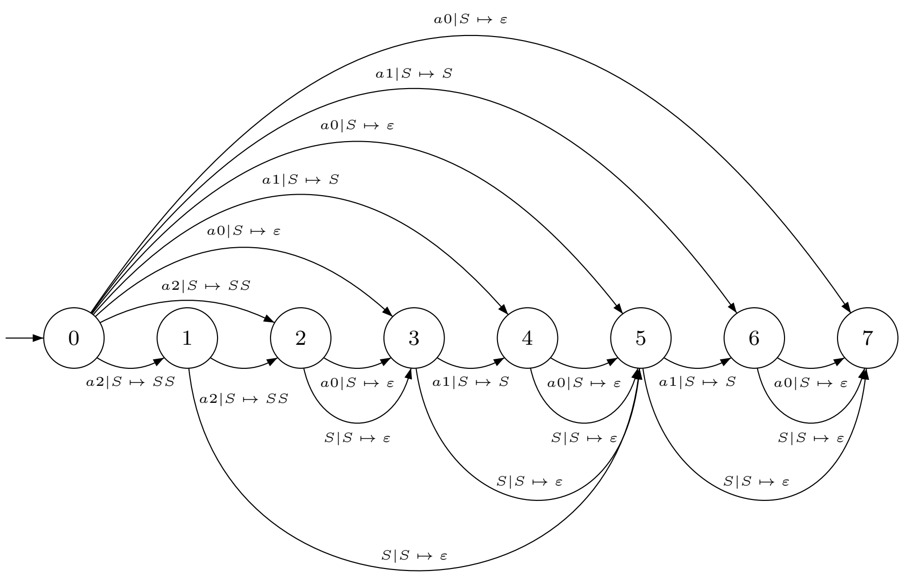
In the following theorem we prove the correctness of the constructed tree pattern PDA.

Theorem 8 *Given a tree t and its prefix notation $pref(t)$, the PDA $M_{npt}(t)$ constructed by Alg. 5 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 Alg. 4 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 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. □

The nondeterministic tree pattern PDA $M_{npt}(t)$ is again an acyclic input-driven PDA, and therefore can be determinised by Alg. 2 to an equivalent deterministic tree pattern PDA $M_{dpt}(t)$.

Example 10 Consider nondeterministic tree pattern PDA $M_{npt}(t_1)$ from Example 9, the transition diagram of which is illustrated in Fig. 13. The deterministic tree pattern PDA $M_{dpt}(t_1)$ constructed by Alg. 2 is
$M_{dpt}(t_1) = (\{[0], [1, 2], [2], [3], [4], [5], [6], [7], [3, 5, 7], [3, 5], [4, 6], [5, 7]\}, \mathcal{A}, \{S\}, \delta_6, [0], S, \emptyset))$, where mapping δ_6 is a set of the following transitions:



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

Fig. 13 Transition diagram of nondeterministic tree pattern PDA $M_{npt}(t_1)$ from Example 9 for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

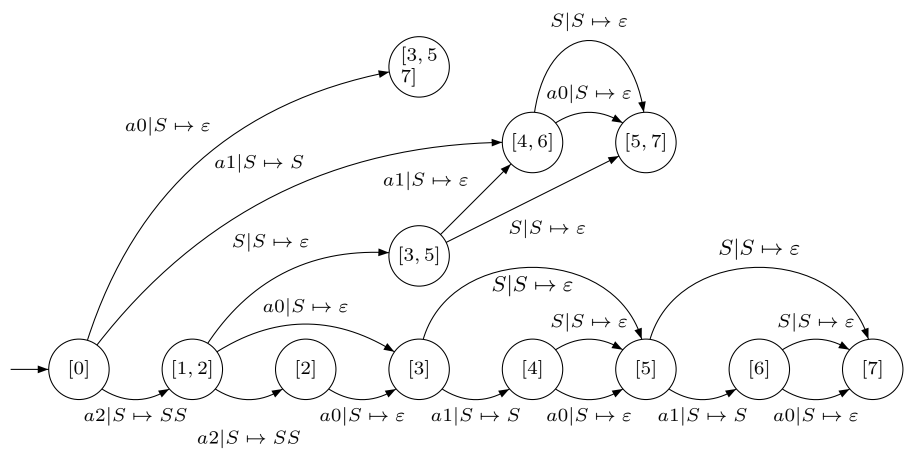
$$\begin{array}{llllll}
\delta_6([0], a2, S) & = & ([1,2], SS) & \delta_6([0], a0, S) & = & ([3,5,7], \varepsilon) \\
\delta_6([1,2], a2, S) & = & ([2], SS) & \delta_6([0], a1, S) & = & ([4,6], S) \\
\delta_6([2], a0, S) & = & ([3], \varepsilon) & \delta_6([1,2], a0, S) & = & ([3], \varepsilon) \\
\delta_6([3], a1, S) & = & ([4], S) & \delta_6([4,6], a0, S) & = & ([5,7], \varepsilon) \\
\delta_6([4], a0, S) & = & ([5], \varepsilon) & \delta_6([4,6], S, S) & = & ([5,7], \varepsilon) \\
\delta_6([5], a1, S) & = & ([6], S) & \delta_6([1,2], S, S) & = & ([3,5], \varepsilon) \\
\delta_6([6], a0, S) & = & ([7], \varepsilon) & \delta_6([3,5], a1, S) & = & ([4,6], S) \\
 & & & \delta_6([3,5], S, S) & = & ([5,7], \varepsilon)
\end{array}$$

The transition diagram of nondeterministic tree pattern PDA $M_{dpt}(t_1)$ is illustrated in Fig. 14. 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 $a|\alpha \mapsto \beta$.

Fig. 15 shows the sequence of transitions (the trace) performed by deterministic tree pattern PDA $M_{dpt}(t_1)$ for input tree pattern $p_1 = a2\ S\ a1\ S$, which is illustrated in Fig. 2. □

The rest of this section is devoted to a discussion on the space required by the deterministic tree pattern PDA. Some other examples are also presented. In comparison with the nondeterministic subtree PDA, the nondeterministic tree pattern PDA has added transitions, which read the special nullary symbol S. As a consequence, the deterministic tree pattern PDA has more transitions than the deterministic subtree PDA and in many cases it has also more states. Although the number of distinct tree patterns matching a tree with n nodes can be exponential in n, we show that for specific cases of trees the total size of the constructed deterministic tree pattern PDA is linear in n.

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



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

Fig. 14 Transition diagram of deterministic tree pattern PDA $M_{dpt}(t_1)$ from Example 10 for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

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

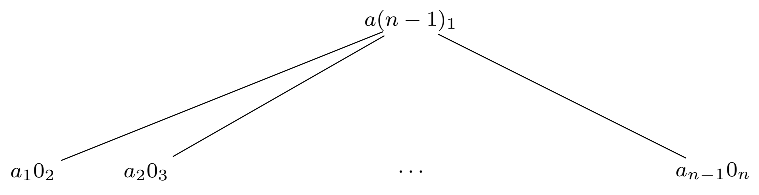
Fig. 15 Trace of deterministic PDA M_{dpt} from Example 10

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_3 whose structure is given by the prefix notation $pref(t_3) = a(n-1)\ a_1 0\ a_2 0 \dots a_{n-1} 0$, where $n \geq 2$. Such a tree is illustrated in Fig. 16. In this tree, each of the nullary symbols $a_1 0, a_2 0, \dots, a_{n-1} 0$ 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_3.

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 $2^{n-1} + n$, and the lemma holds. □

Example 11 Figs. 17 and 18 show the transition diagrams of the deterministic tree pattern PDAs constructed by Algs. 5 and 2 for the two examples of trees with the boundary structures illustrated in Figs. 11 and 16, respectively. □



$pref(t_3) = a(n-1)\ a_1 0\ a_2 0 \dots a_{n-1} 0$

Fig. 16 A tree t_3 with $2^{n-1} + n$ distinct tree patterns matching the tree t_3 and its prefix notation

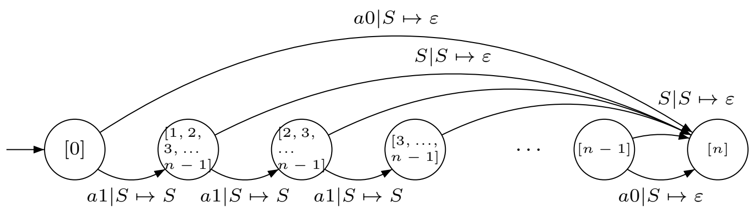


Fig. 17 Transition diagram of the deterministic tree pattern PDA $M_{dpt}(t_2)$ for the tree t_2 in prefix notation $pref(t_2) = (a1)^{n-1}a0$, where $n \geq 1$

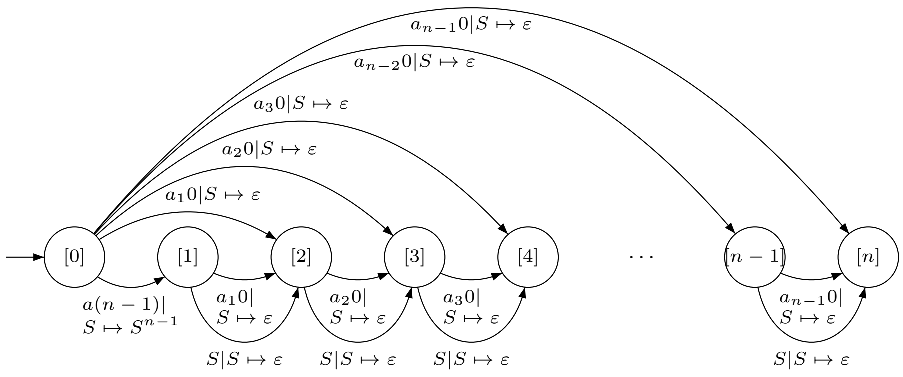
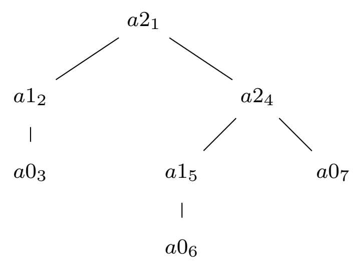

$a_{n-1}0|S \mapsto \varepsilon$
$a_{n-2}0|S \mapsto \varepsilon$
$a_3 0|S \mapsto \varepsilon$
$a_2 0|S \mapsto \varepsilon$
$a_1 0|S \mapsto \varepsilon$
[0]
[1]
[2]
[3]
[4]
…
[n − 1]
[n]
$a(n-1)| S \mapsto S^{n-1}$
$a_1 0| S \mapsto \varepsilon$
$a_2 0| S \mapsto \varepsilon$
$a_3 0| S \mapsto \varepsilon$
$a_{n-1} 0| S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$

Fig. 18 Transition diagram of the deterministic tree pattern PDA $M_{dpt}(t_3)$ for the tree t_3 in prefix notation $pref(t_3) = a(n-1)\ a_1 0\ a_2 0 \dots a_{n-1} 0$, where $n \geq 2$ and $a_i 0 \neq a_j 0$, $i \neq j$

As the simplest case, where the total size of the deterministic tree pattern PDA is linear in the number of nodes of the tree, we consider a deterministic tree pattern PDA that has just the same states as the corresponding deterministic subtree PDA. In such a case, the added transitions reading nullary symbol S have to lead only to states which



$pref(t_4) = a2\ a1\ a0\ a2\ a1\ a0\ a0$

Fig. 19 Tree t_4 from Example 12 and its prefix notation

are already states of the deterministic subtree PDA. This occurs for specific cases of trees which are called trees with periodical subtrees, and are defined by Definition 5.

Definition 5 Let t be a tree over a ranked alphabet $\mathcal{A}$. The tree t is a *tree with periodical subtrees* if there exists a mapping $\mathcal{Z}$ of $\mathcal{A} \setminus \mathcal{A}_0$ into $\mathcal{A}^+$ such that the prefix notation of each subtree st of the tree t is of the form $pref(st) = (ax)^m y$, where $a \in \mathcal{A}$, $x \in \mathcal{A}^*$, $y \in \mathcal{A}^+$, $m \geq 1$, $\mathcal{Z}(a) = xy$, and y is a subtree in prefix notation.

The informal meaning of Def. 5 is that in a tree with periodical subtrees it holds for each label a that all subtrees with the root labelled by a respectively have their subtrees equal from the leftmost to the rightmost but one subtree and string x is the concatenation of prefix notations of these subtrees. String axy in Def. 5 is the prefix notation of the subtree with the root a which has the fewest number of nodes, and string y is the prefix notation of its rightmost subtree.

Example 12 Examples of trees with periodical subtrees are illustrated in Figs. 11, 16 and 19, and the transition diagrams of the corresponding deterministic tree pattern PDAs are illustrated in Figs. 17, 18 and 20, respectively.

In the case of tree t_4, which is constructed over ranked alphabet $\{a2, a1, a0\}$ and is illustrated in Fig. 19, the corresponding mapping $\mathcal{Z}_4$ is the following set:

$$\begin{array}{lcl} \mathcal{Z}_4(a2) & = & a1\ a0\ a0, \text{ where } x = a1\ a0 \text{ and } y = a0 \\ \mathcal{Z}_4(a1) & = & a0, \text{ where } x = \varepsilon \text{ and } y = a0 \end{array}$$

Prefix notations of all subtrees of tree t_4 are: $a2\ a1\ a0\ a2\ a1\ a0\ a0$, $a2\ a1\ a0\ a0$, $a1\ a0$ and $a0$. □

The resulting number of states and of transitions of the deterministic tree pattern PDAs for trees with determined subtrees are formally proved in Theorem 9. We also present a companion Lemma 4 which shows that the deterministic tree pattern PDA has more states than the deterministic subtree PDA if the condition for the trees with determined subtrees is violated.

Theorem 9 *Let t be a tree over a ranked alphabet $\mathcal{A}$. If the tree t is a tree with periodical subtrees, then the deterministic tree pattern PDA $M_{dpt}(t)$ constructed by*

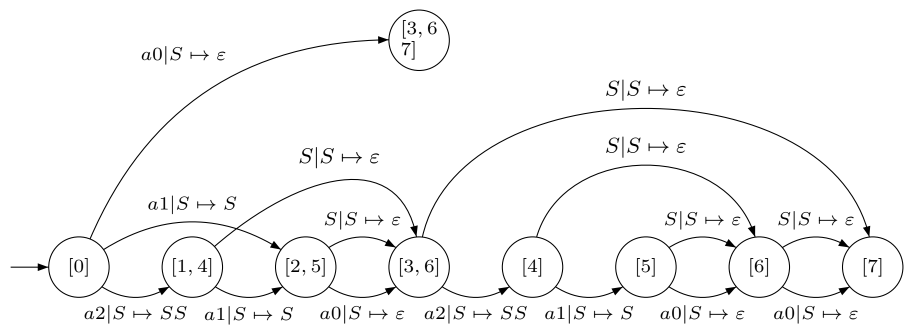

$a0|S \mapsto \varepsilon$
[3, 6 7]
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$a1|S \mapsto S$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
$S|S \mapsto \varepsilon$
[0]
[1, 4]
[2, 5]
[3, 6]
[4]
[5]
[6]
[7]
$a2|S \mapsto SS$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$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

Fig. 20 Transition diagram of deterministic tree pattern PDA $M_{dpt}(t_4)$ from Example 12 for tree in prefix notation $pref(t_4) = a2\ a1\ a0\ a2\ a1\ a0\ a0$

Algs. 5 and 3 has just one pushdown symbol, fewer than $N \leq 2n+1$ states and at most $2N + n - 3 \leq 5n - 1$ transitions.

Proof Part I. In the first part of the proof we describe a general common property of the deterministic string suffix automaton, the deterministic subtree PDA and the deterministic tree pattern PDA.

A well-known general property of the deterministic string suffix automaton constructed for a string over an alphabet T is that the d-subset of a state in which the string suffix automaton is after reading a substring x, where $x \in T^+$, is the set of positions of the last symbols of all occurrences of the substring x [28].

The transitions of the deterministic subtree PDA $M_{dps}(t)$ are extensions of transitions of the deterministic string suffix automaton, and therefore the same general property also holds for the PDA $M_{dps}(t)$: the d-subset of a state in which the PDA $M_{dps}(t)$ is after reading a substring x, where $x \in \mathcal{A}^+$, is the set of the last nodes of all occurrences of x in prefix notations of subtrees.

For example, deterministic tree pattern PDA $M_{dps}(t_1)$ from Example 7, which is illustrated in Fig 9, is in state $[5, 7]$ after reading string $a1\ a0$. This means that nodes $a0_5$ and $a0_7$ are the last nodes of all occurrences of $a1\ a0$ in prefix notations of subtrees of tree t_1, which is illustrated in Fig. 1.

The deterministic tree pattern PDA $M_{dpt}(t)$ has also added transitions reading nullary symbol S, which represents subtrees. This means that the d-subset of a state in which the PDA $M_{dpt}(t)$ is after reading a substring y, where $y \in \mathcal{A}(\mathcal{A} \cup \{S\})^*$, is the union of d-subsets of states in which the PDA $M_{dps}(t)$ is after reading all substrings which are matched with y.

For example, deterministic tree pattern PDA $M_{dpt}(t_1)$ from Example 10, which is illustrated in Fig 14, is in state $[3, 5]$ after reading template $a2\ S$. Template $a2\ S$ matches prefix notations $a2\ a0$ and $a2\ a2\ a0\ a1\ a0$ (we note that tree t_1 is not a tree with periodical subtrees). This means that nodes $a0_3$ and $a0_5$ are the last nodes of all occurrences of $a2\ a0$ and $a2\ a2\ a0\ a1\ a0$ in prefix notations of subtrees of tree t_1, which is illustrated in Fig. 1.

Part II. In the second part of the proof we consider the maximal number of states.

The deterministic tree pattern PDA $M_{dpt}(t)$ is constructed for a tree t with periodical subtrees, which means that for each subtree with a root a, where $Arity(a) \geq 1$, there is always only one substring which can be matched with each symbol S representing a subtree from the leftmost to the rightmost but one subtree. Further, the prefix notation of the rightmost subtree may be only of the form $(ax)^p y$, where $p \geq 0$ and y is a subtree in prefix notation. This means that the rightmost leaf of the rightmost subtree is the rightmost leaf of the subtree in prefix notation y. Thus, every state of the deterministic tree pattern PDA $M_{dpt}(t)$ with an incoming transition reading S is also a state of the deterministic subtree PDA $M_{dps}(t)$ constructed by Algs. 2 and 3, ie. $M_{dpt}(t)$ has fewer than $N \leq 2n+1$ states (see Theorem 6).

Part III. In the third part of the proof we consider the maximal number of transitions. In comparison with the deterministic subtree PDA $M_{dps}(t)$, the PDA $M_{dpt}(t)$ has added transitions reading nullary symbol S. There is just one transition reading nullary symbol S which leads from each state except the initial and the last state, in which the pushdown store is always empty. Thus, there are $N-2$ such added transitions, where N is the number of states. The maximal number of transitions of the PDA $M_{dps}(t)$ is $N+n-1 \leq 3n$ (see Theorem 6) and therefore the overall maximal number of transitions of the PDA $M_{dpt}(t)$ is $2N+n-3 \leq 5n-1$. □

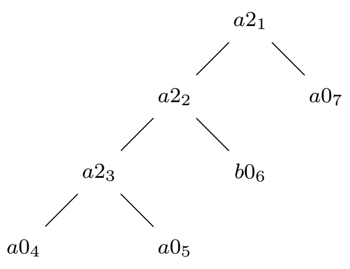
Lemma 4 *Let t be a tree over a ranked alphabet $\mathcal{A}$. If the tree t is not a tree with periodical subtrees, then the deterministic tree pattern PDA $M_{dpt}(t)$ constructed by Algs. 5 and 3 has more states than the deterministic subtree PDA $M_{dps}(t)$ constructed by Algs. 2 and 3.*

Proof The deterministic tree pattern PDA $M_{dpt}(t)$ is an extension of the deterministic subtree PDA $M_{dps}(t)$ and all states of the PDA $M_{dps}(t)$ are also states of the PDA $M_{dpt}(t)$. We show that the PDA $M_{dpt}(t)$ has at least one transition which reads nullary symbol S and leads to a state which is not present in the PDA $M_{dps}(t)$:

The tree t is not a tree with periodical subtrees, which means that one of the following two cases occurs for some two subtrees st_1 and st_2 with a same root $a \in \mathcal{A} \backslash \mathcal{A}_0$:

1. There are two different substrings which can be matched with a symbol S representing the i-th subtree of st_1 and of st_2, where $i < Arity(a)$ (the i-th subtree is from the leftmost to the rightmost but one subtrees). Since the d-subset of a state with an incoming transition reading the symbol S is the union of d-subsets of states of the PDA $M_{dps}(t)$ for these two matched substrings, a new state of the PDA $M_{dpt}(t)$ which is not present in the PDA $M_{dps}(t)$ is created.
   For example, tree t_1, which is illustrated in Fig. 1, has two substrings $a2\ a0$ and $a2\ a2\ a0\ a1\ a0$ which are matched with tree template $a2\ S$, where S does not represent the rightmost subtree. Therefore, new state $[3,5]$ of PDA $M_{dpt}(t_1)$ from Example 10, which is illustrated in Fig 14, has been created.
2. The prefix notations of the rightmost subtrees are not of the form $(ax)^p y$, where $p \geq 0$, and y is the prefix notation of a subtree. This means that these righmost subtrees are neither equal nor have the same rightmost leaf as the subtree in prefix notation axp and a new state of the PDA $M_{dpt}(t)$ which is not present in the PDA $M_{dps}(t)$ is created. □

Example 13 Examples of trees which are not trees with periodical subtrees are illustrated in Figs. 1 and 21, and the transition diagrams of the corresponding deterministic tree pattern PDAs are illustrated in Figs. 14 and 22, respectively. In Fig. 14, state $[3,5]$ is not in the corresponding deterministic subtree PDA, which is illustrated in Fig. 9. In Fig. 22, states $[4,5]$, $[4,5,6]$, $[5,6]$, $[5,7]$, $[5,6,7]$ and $[6,7]$ are not in the corresponding deterministic subtree PDA. □



$pref(t_5) = a2\ a2\ a2\ a0\ a0\ b0\ a0$

Fig. 21 Tree t_5 from Example 13 and its prefix notation

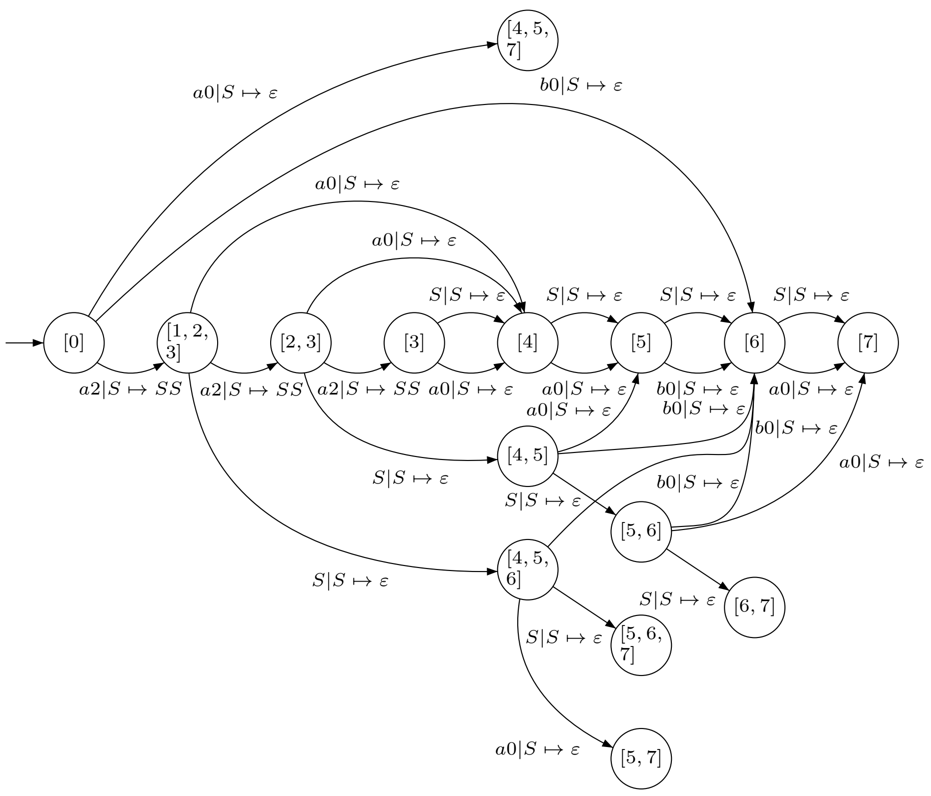


Fig. 22 Transition diagram of deterministic tree pattern PDA $M_{dpt}(t_5)$ from Example 13 for tree in prefix notation $pref(t_5) = a2\ a2\ a2\ a0\ a0\ b0\ a0$

In general, the deterministic tree pattern PDA can have more than linear number of states. For example, given a tree t in prefix notation $pref(t) = a2^m a0^{m+1}$, $m \geq 1$, the corresponding PDA $M_{dpt}(t)$ has $N = \frac{m^2+m}{2} + 2m + 2$ states and $2(N - m - 1) = m^2 + 3m + 2$ transitions. This means that the number of distinct tree patterns which match such a tree t is exponential in the number of nodes of the tree and the total size of the corresponding deterministic tree pattern PDA is quadratic. The maximal numbers of states and transitions of the deterministic tree pattern PDA in general remain open problems.

6 Conclusion

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

The future work should focus on the numbers of states and transitions of deterministic tree pattern PDAs for various cases of trees and on the on-line construction of the deterministic tree pattern PDA directly from a given tree.

Regarding other tree algorithms whose model of computation is the standard deterministic pushdown automaton, we have recently introduced principles for two other new algorithms. First, a new and simple method for constructing tree pattern matchers as deterministic pushdown automata directly from given tree patterns without constructing finite tree automata as an intermediate product [15, 26]. Second, a method for finding all repeats of subtrees and connected subgraphs in trees with the use of subtree PDAs and tree pattern PDAs, respectively [30, 26]. More details on these results and related information can also be found on [2].

Acknowledgements We would like to thank Robin Healey for his helpful comments on drafts of this paper.

References

1. Aho, A.V., Ullman, J.D.: The theory of parsing, translation, and compiling. Prentice-Hall Englewood Cliffs, N.J. (1972)
2. Arbology www pages. Available on: http://www.arbology.org (2009). October 2009
3. Berstel, J.: Transductions and Context-Free Languages. Teubner Studienbucher, Stuttgart (1979)
4. Bille, P.: Pattern matching in trees and strings. Ph.D. thesis, FIT University of Copenhagen, Copenhagen (2008)
5. Blumer, A., Blumer, J., Haussler, D., Ehrenfeucht, A., Chen, M.T., Seiferas, J.I.: The smallest automaton recognizing the subwords of a text. Theor. Comput. Sci. **40**, 31–55 (1985)
6. Chase, D.R.: An improvement to bottom-up tree pattern matching. In: ACM Symp. POPL, pp. 168–177 (1987)

7. Cleophas, L.: Tree algorithms. two taxonomies and a toolkit. Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven (2008)
8. Cole, R., Hariharan, R., Indyk, P.: Tree pattern matching and subset matching in deterministic ($\log^3$)-time. In: SODA, pp. 245–254 (1999)
9. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata (2007). Release October, 12th 2007
10. Crochemore, M.: Transducers and repetitions. Theor. Comput. Sci. **45**(1), 63–86 (1986)
11. Crochemore, M., Hancart, C.: Automata for matching patterns. In: G. Rozenberg, A. Salomaa (eds.) Handbook of Formal Languages, vol. 2 Linear Modeling: Background and Application, chap. 9, pp. 399–462. Springer–Verlag, Berlin (1997)
12. Crochemore, M., Rytter, W.: Jewels of Stringology. World Scientific, New Jersey (1994)
13. Dubiner, M., Galil, Z., Magen, E.: Faster tree pattern matching. J. ACM **41**(2), 205–213 (1994)
14. Ferdinand, C., Seidl, H., Wilhelm, R.: Tree automata for code selection. Acta Inf. **31**(8), 741–760 (1994)
15. Flouri, T., Janoušek, J., Melichar, B.: Tree pattern matching by deterministic pushdown automata. In: M. Ganzha, M. Paprzycki (eds.) Proceedings of the IMCSIT, Vol. 4, pp. 659–666. IEEE Computer Society Press (2009)
16. Fraser, C.W., Hanson, D.R., Proebsting, T.A.: Engineering a simple, efficient code-generator generator. LOPLAS **1**(3), 213–226 (1992)
17. Fraser, C.W., Henry, R.R., Proebsting, T.A.: Burg: fast optimal instruction selection and tree parsing. SIGPLAN Notices **27**(4), 68–76 (1992)
18. Gecseg, F., Steinby, M.: Tree languages. In: G. Rozenberg, A. Salomaa (eds.) Handbook of Formal Languages, vol. 3 Beyond Words. Handbook of Formal Languages, pp. 1–68. Springer–Verlag, Berlin (1997)
19. Gibbons, A., Rytter, W.: Efficient parallel algorithms. Cambridge University Press, New York, NY, USA (1988)
20. Glanville, R.S., Graham, S.L.: A new method for compiler code generation. In: POPL, pp. 231–240 (1978)
21. Hoffmann, C.M., O'Donnell, M.J.: Pattern matching in trees. J. ACM **29**(1), 68–95 (1982)
22. Holub, J.: Finite automata in stringology. Habilitation Thesis, CTU, Prague (2007)
23. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation, 2nd edn. Addison-Wesley, Boston (2001)
24. Janoušek, J.: String suffix automata and subtree pushdown automata. In: J. Holub, J. Žďárek (eds.) Proceedings of the Prague Stringology Conference 2009, pp. 160–172. Czech Technical University in Prague, Czech Republic (2009)
25. Janoušek, J., Melichar, B.: On regular tree languages and deterministic pushdown automata. Acta Inf. **46**(7), 533–547 (2009). DOI 10.1007/s00236-009-0104-9
26. London stringology days 2009 conference presentations. Available on: http://www.dcs.kcl.ac.uk/events/LSD&LAW09/, King's College London, London (2009)
27. Madhavan, M., Shankar, P., Rai, S., Ramakrishna, U.: Extending graham-glanville techniques for optimal code generation. ACM Trans. Program. Lang. Syst. **22**(6), 973–1001 (2000)

28. Melichar, B.: Repetitions in text and finite automata. In: Cleophas, L., Watson, B.W. (Eds.) Proceedings of the Eindhoven FASTAR Days 2004, TU Eindhoven, pp. 1–46 (2004)
29. Melichar, B., Holub, J., Polcar, J.: Text searching algorithms. Available on: http://stringology.org/athens/ (2005). Release November 2005
30. Melichar, B., Janoušek, J.: Repeats in trees by subtree and tree pattern pushdown automata (2009). Draft
31. Ramesh, R., Ramakrishnan, I.V.: Nonlinear pattern matching in trees. J. ACM **39**(2), 295–316 (1992). DOI http://doi.acm.org/10.1145/128749.128752
32. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer–Verlag, Berlin (1997)
33. Rozenberg, G., Salomaa, A. (eds.): Vol. 1: Word, Language, Grammar, Handbook of Formal Languages. Springer–Verlag, Berlin (1997)
34. Shankar, P., Gantait, A., Yuvaraj, A.R., Madhavan, M.: A new algorithm for linear regular tree pattern matching. Theor. Comput. Sci. **242**(1-2), 125–142 (2000). DOI http://dx.doi.org/10.1016/S0304-3975(98)00205-9
35. Smyth, B.: Computing Patterns in Strings. Addison-Wesley-Pearson Education Limited, Essex, England (2003)
36. Valiant, L.G., Paterson, M.: Deterministic one-counter automata. In: Automaten theorie und Formale Sprachen, pp. 104–115 (1973)
37. Wagner, K., Wechsung, G.: Computational Complexity. Springer–Verlag, Berlin (2001)

Finding Repeats of Subtrees in a Tree Using Pushdown Automata

Jan Janoušek and Bořivoj Melichar

Department of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague
Kolejni 550/2, 160 00 Prague 6, Czech Republic
Jan.Janousek|Borivoj.Melichar@fit.cvut.cz
WWW: http://www.arbology.org/

Abstract. Efficient methods of finding various kinds of repeats in a string can be based on constructing and analysing string suffix trees or string suffix automata, which represent complete indexes of the string for substrings. In [9, 11] we have introduced subtree pushdown automata, which are analogous to the string suffix automata and represent complete indexes of trees for subtrees. This paper presents a new and simple method of finding various kinds of all repeats of subtrees in a given tree by constructing and analysing the subtree pushdown automaton for the tree. Given a tree with n nodes, the finding of all repeats of subtrees in the tree is performed in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.

1 Introduction

Trees are one of the fundamental data structures used in Computer Science. Given a tree, finding beforehand unknown repeating subtrees of the tree is an important problem with many applications – data compression, compiler code optimization, processing data tree structures such as XML and so on.

Periodicity in strings have been of interest since the beginning of the 20th century and effective methods for finding various kinds of repetitions and repeats in a string form an important part of well-researched stringology theory [7, 15, 16]. Some of these methods are based on constructing and analysing string suffix trees or string suffix automata, which represent complete indexes of the string for substrings [2, 4, 6, 13, 14].

[10] shows that string pushdown automata are an appropriate model of computation for processing labelled ordered ranked trees in a linear notation. We note that the prefix or postfix linear notation of a tree can be obtained by prefix or postfix traversing, respectively, and that every sequential algorithm processing a tree visits nodes of the tree in a linear order. In [9, 11] we have introduced

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

subtree pushdown automata, which are analogous to string suffix automata and represent complete indexes of trees for subtrees. This paper presents a new and simple method of finding various kinds of all repeats of subtrees in a given tree by constructing and analysing the subtree pushdown automaton for the tree. The presented method is similar to the method of finding repeats of substrings in a string by constructing and analysing the string suffix automaton [14, 15]. Given a tree with n nodes, the finding of all repeats of subtrees in the tree is performed in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space. We are not aware of any other known method which would find all repeats of subtrees in a tree in linear time and space. An early presentation of the basic idea of the presented method of finding repeats of subtrees can be found in [12].

The rest of the paper is organised as follows. Basic definitions are given in section 2. The third and the fourth section deal with subtree pushdown automata and finding repeats of subtrees by constructing and analysing the subtree pushdown automata, respectively. The last section is the conclusion.

2 Basic notions

2.1 Ranked alphabet, tree, prefix notation

We define notions on trees similarly as they are defined in [1, 5, 8].

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

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

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

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

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

A *labelled, ordered and rooted ranked tree* t over a ranked alphabet $\mathcal{A}$ is an ordered dag $t = (N, R)$ with a special node $r \in A$, called the *root*, such that
(1) r has in-degree 0,
(2) all other nodes of t have in-degree 1,
(3) there is just one path from the root r to every $f \in N$, where $f \neq r$,
(4) every node $f \in N$ is labelled by a symbol $a \in \mathcal{A}$ and the out-degree of a_f is $Arity(a)$.

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

The *prefix notation* $pref(t)$ of a labelled, ordered, ranked and rooted tree t is obtained by applying the following *Step* recursively, beginning at the root of t: *Step*: Let this application of *Step* be to node a_f. If a_f is a leaf, list a and halt. If a_f is not a leaf, let its direct descendants be $a_{f_1}, a_{f_2}, \ldots, a_{f_n}$. Then list a and subsequently apply *Step* to $a_{f_1}, a_{f_2}, \ldots, a_{f_n}$ in that order.

We note that in many papers on the theory of tree languages, such as [5, 8], labelled ordered ranked trees are defined with the use of ordered ranked *ground terms*. Ground terms can be regarded as labelled ordered ranked trees in prefix notation.

Example 1. Consider a ranked alphabet $\mathcal{A} = \{a0, a1, a2\}$. Consider a tree t_1 over $\mathcal{A}$ $t_1 = (\{a2_1, a2_2, a0_3, a1_4, a0_5, a2_6, a0_7, a1_8, a0_9\}, R_1)$, where R_1 is a set of the following ordered sequences of pairs:

$$\begin{array}{l}((a2_1, a2_2), (a2_1, a2_6)),\\ ((a2_2, a0_3), (a2_2, a1_4)),\\ ((a1_4, a0_5)),\\ ((a2_6, a0_7), (a2_6, a1_8)),\\ ((a1_8, a0_9))\end{array}$$

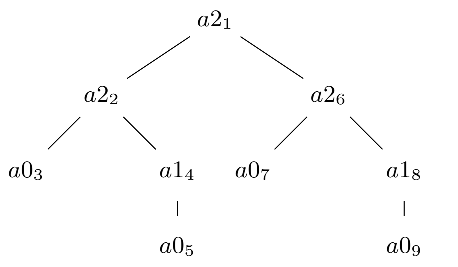
Tree t_1 in prefix notation is string $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a2\ a0\ a1\ a0$. Trees can be represented graphically, and tree t_1 is illustrated in Fig. 1. □

2.2 Alphabet, pushdown automaton

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

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

An (extended) *nondeterministic pushdown automaton* (nondeterministic PDA) is a seven-tuple $M = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$, where Q is a finite set of *states*, $\mathcal{A}$ is an *input alphabet*, G is a *pushdown store alphabet*, δ is a mapping from $Q \times (\mathcal{A} \cup \{\varepsilon\}) \times G^*$ into a set of finite subsets of $Q \times G^*$, $q_0 \in Q$ is an initial state, $Z_0 \in G$ is the initial pushdown store symbol, and $F \subseteq Q$ is the set of final



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

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

(accepting) states. Triple $(q, w, x) \in Q \times \mathcal{A}^* \times G^*$ denotes the configuration of a pushdown automaton. In this paper we will write the top of the pushdown store x on its right hand side. The initial configuration of a pushdown automaton is a 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 M is a *deterministic* pushdown automaton (deterministic PDA), if it holds:

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

A pushdown automaton is *input–driven* if each of its pushdown operations is determined only by the input symbol.
A language L accepted by a pushdown automaton M is defined in two distinct ways:

1. *Accepting by final state:*
$$L(M) = \{x : \delta(q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \gamma) \wedge x \in \mathcal{A}^* \wedge \gamma \in G^* \wedge q \in F\}.$$
2. *Accepting by empty pushdown store:*
$$L_\varepsilon(M) = \{x : (q_0, x, Z_0) \vdash_M^* (q, \varepsilon, \varepsilon) \wedge x \in \mathcal{A}^* \wedge q \in Q\}.$$

If PDA accepts the language by empty pushdown store, then the set F of final states is the empty set. In this paper we use only PDAs which accept the languages by empty pushdown store.

For more details see [1].

3 Subtree pushdown automata

This section gives basic information on nondeterministic and deterministic subtree pushdown automata [9, 11]. A subtree pushdown automaton represents a complete index of a given tree for subtrees and accepts all subtrees of the tree by the empty pushdown store. For the deterministic subtree pushdown automata the search phase of all occurrences of an input subtree is performed in time linear in the size of the input subtree and not depending on the size of the tree. We note that subtree pushdown automaton has only one pushdown symbol and therefore their pushdown store can be implemented by a single integer counter.

Generally, it holds for any tree that each of its subtrees in prefix notation is a substring of the tree in prefix notation. This important property allows to use principles of effective string algorithms for processing trees in prefix notation. However, not every substring of a tree in prefix notation is a prefix notation of its subtree. In subtree pushdown automata their pushdown operations determine which of the substrings of trees in prefix notation are those representing subtrees.

Theorem 1. *Given a tree t and its prefix notation $pref(t)$, prefix notations of all subtrees of the tree t are substrings of $pref(t)$.*

Proof. In [11]. □

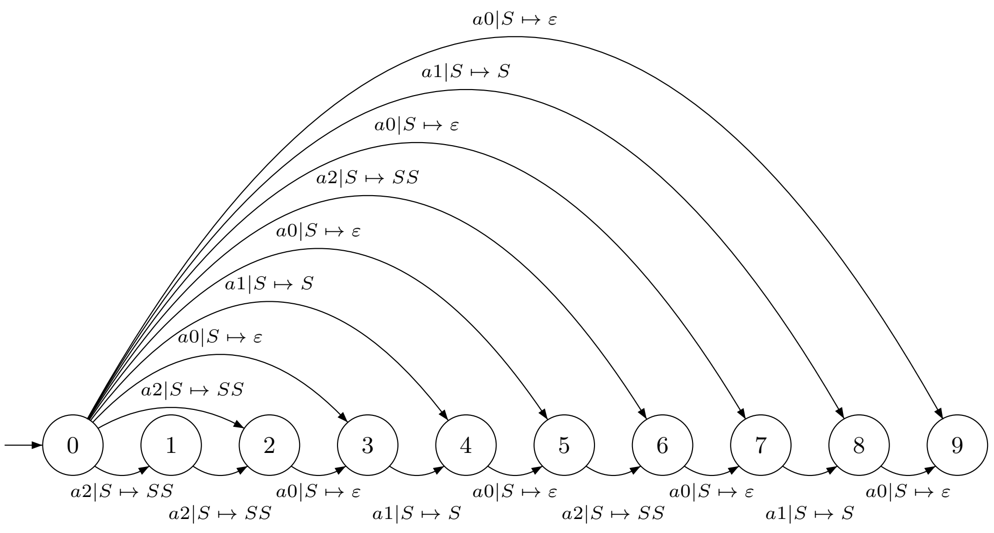
Definition 1. *Let t and $pref(t)$ be a tree and its prefix notation, respectively. A* subtree pushdown automaton *for $pref(t)$ accepts all subtrees of t in prefix notation.*

Algorithm 1. Construction of a nondeterministic subtree PDA for a tree t in prefix notation $pref(t)$.
Input: A tree t; prefix notation $pref(t) = a_1a_2 \ldots a_n$, $n \geq 1$.
Output: Nondeterministic subtree PDA $M_{nps}(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

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

Example 2. Consider tree t_1 from Example 1, which is illustrated in Fig. 1. A subtree PDA for tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a2\ a0\ a1\ a0$ which has been constructed by Alg. 1, is nondeterministic PDA $M_{nps}(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \mathcal{A}, \{S\}, \delta_1, 0, S, \emptyset))$, where mapping δ_1 is illustrated in the transition diagram of $M_{nps}(t_1)$ in Fig. 2. In this figure for each transition rule $\delta(p, a, \alpha) = (q, \beta)$ the edge leading from state p to state q is labelled by the string of the form $a|\alpha \mapsto \beta$. □

Theorem 2. *Given a tree t and its prefix notation $pref(t)$, the PDA $M_{nps}(t)$ constructed by Alg. 1 is a subtree PDA for $pref(t)$.*



$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a2|S \mapsto SS$
0
1
2
3
4
5
6
7
8
9
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$

Fig. 2. Transition diagram of nondeterministic subtree PDA $M_{nps}(t_1)$ for tree t_1 in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a2\ a0\ a1\ a0$ from Example 2

Proof. In [11]. □

It is known that each nondeterministic input–driven PDA can be transformed to an equivalent deterministic input–driven PDA. An algorithm of such a determinisation of the nondeterministic subtree PDA is described by Alg. 2. This algorithm is a simple extension of the standard algorithm of transformation of a nondeterministic finite automaton to an equivalent deterministic one [1].

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

```
1. Let $spds(q')$, where $q' \in Q'$, denote a set of integers (the abbreviation $spds$
   stands for the number of Symbols $S$ in the PushDown Store.)
2. Initially, $Q' = \{[0]\}$, $q_I = [0]$, $spds([0]) = \{1\}$ and $[0]$ is an unmarked state.
3. (a) Select an unmarked state $q'$ from $Q'$ such that $q'$ contains the smallest
       possible state $q \in Q$, where $0 \leq q \leq n$.
   (b) If there is $v > 0$, $v \in spds(q')$, then for each input symbol $a \in \mathcal{A}$:
       i. Add transition $\delta'(q', a, \alpha) = (q'', \beta)$, $q'' = \{q : \delta(p, a, \alpha) = (q, \beta)$ for
          all $p \in q'\}$. If $q''$ is not in $Q'$ then add $q''$ to $Q'$ and create $spds(q'') =
          \emptyset$. Add $j$, where $\delta(q', a, S^i) \vdash_{M_{dx}(t)} (q'', \varepsilon, S^j)$ and $i \in spds(q')$, to
          $spds(q'')$.
```

(c) Set the state q' as marked.
4. Repeat step 5 until all states in Q' are marked. □

Example 3. The deterministic subtree PDA for tree t_1 from Example 1, which has been constructed by Alg. 2 from nondeterministic subtree PDA $M_{nps}(t_1)$ from Example 2, is deterministic PDA $M_{dps}(t_1) = (\{[0], [1, 2, 6], [2], [3], [4], [5], [6], [7], [8], [9], [3, 5, 7, 9], [3, 7], [4, 8], [5, 9]\}, \mathcal{A}, \{S\}, \delta_2, [0], S, \emptyset))$, where mapping δ_2 is illustrated by the transition diagram $M_{dps}(t_1)$ in Fig. 3. Again, in this figure for each transition rule $\delta_3(p, a, \alpha) = (q, \beta)$ from δ_3 the edge leading from state p to state q is labelled by the string of the form $a|\alpha \mapsto \beta$. □

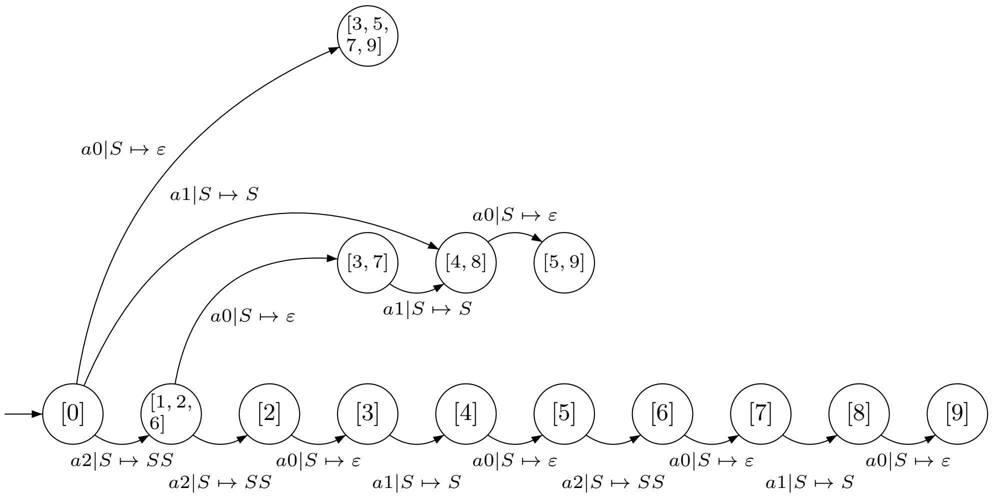

[3, 5, 7, 9]
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
[3, 7]
[4, 8]
[5, 9]
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
[0]
[1, 2, 6]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a1|S \mapsto S$
$a0|S \mapsto \varepsilon$

Fig. 3. Transition diagram of deterministic subtree PDA $M_{dps}(t_1)$ for tree in prefix notation $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a2\ a0\ a1\ a0$ from Example 3

Theorem 3. *Given an acyclic input–driven nondeterministic PDA $M_{nx}(t) = (Q, \mathcal{A}, \{S\}, \delta, q_0, S, \emptyset)$, the deterministic PDA $M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', \{q_0\}, S, \emptyset)$ constructed by Alg. 2 is equivalent to PDA $M_{nx}(t)$.*

Proof. In [11]. □

The maximal total size of deterministic subtree pushdown automata is described by the following theorem.

Theorem 4. *Given a tree t with n nodes and its prefix notation $pref(t)$, the deterministic subtree PDA $M_{dps}(t)$ constructed by Algs. 1 and 2 has just one pushdown symbol, fewer than $N \leq 2n + 1$ states and at most $N + n - 1 \leq 3n$ transitions.*

Proof. In [11]. □

4 Finding repeats of subtrees

4.1 Definition of the problem

Given a tree, the problem is to find all repeating subtrees of the tree and to compute kinds and positions of all occurrences of these subtrees. All repeats of subtrees and their properties are summarised in a subtree repeat table, which is defined by Defs. 2, 3 and 4. We define two versions of the subtree repeat table: the first, basic, version of the table contains basic information on repeats and its size is linear to the number of nodes of the tree. The second one, an extended subtree repeat table, contains also further information such as all the repeating subtrees in prefix notation, which can result in a larger table. We note that the linear size of the table is important for linear time and space complexities of the algorithm constructing the table.

Definition 2. *Let t be a tree over a ranked alphabet $\mathcal{A}$. A* subtree position set *$sps(st, t)$, where st is a subtree of t, is the set $sps(st, t) = \{i : pref(t) = x\ pref(st)\ y,\ x, y \in \mathcal{A}^*, i = |x| + 1\}$.*

Informally, the subtree position set for a subtree st contains positions of the roots of all occurrences of the subtree st in prefix notation.

Definition 3. *Let t be a tree over a ranked alphabet $\mathcal{A}$. Given a subtree st of t,* list of subtree repeats *$lsr(st, t)$ is a relation in $sps(st, t) \times \{F, S, Q\}$ defined as follows:*

- *$(i, F) \in lsr(st, t)$ iff $pref(t) = x\ pref(st)\ y$, $i = |x| + 1$, $x \neq x_1\ pref(st)\ x_2$,*
- *$(i, S) \in lsr(st, t)$ iff $pref(t) = x\ pref(st)\ y$, $i = |x| + 1$, $x = x_1\ pref(st)$,*
- *$(i, G) \in lsr(st, t)$ iff $pref(t) = x\ pref(st)\ y$, $i = |x| + 1$, $x = x_1\ pref(st)\ x_2$, $x_2 \in \mathcal{A}^+$.*

Informally, the list of subtree repeats for a subtree st contains kinds of all occurrences of the subtree st. Abbreviations F, S, and G stand for First occurrence of the substree, repeat as a Square, and repeat with a Gap, respectively. In comparison with kinds of repeats in string [14, 15], repeats of subtrees have no kind which would represent the overlapping of subtrees because any two different occurrences of the same subtree cannot overlap.

Definition 4. *Given a tree t, the* basic subtree repeat table *$BSRT(t)$ is the set of all lists of subtree repeats $lsr(st, t)$, where st is a subtree with more than one occurrence in the tree t.*

The extended subtree repeat table *$ESRT(t)$ is the set of all triplets $(sps(st, t), pref(st), lsr(st, t))$, where st is a subtree with more than one occurrence in the tree t.*

Example 4. Consider tree t_1 from Example 1, which is illustrated in Fig. 1. $pref(t_1) = a2\ a2\ a0\ a1\ a0\ a2\ a0\ a1\ a0$. Subtrees with more than one occurrence in tree t_1 are subtrees st_1, st_2 and st_3, where $pref(st_1) = a2\ a0\ a1\ a0$, $pref(st_2) =$

$a1\ a0$, and $pref(st_3) = a0$. These three subtrees are illustrated in Fig. 4. All other subtrees of tree t_1 are present just once in tree t_1.

It holds that $sps(st_1) = \{2, 6\}$, $sps(st_2) = \{4, 8\}$, $sps(st_3) = \{3, 5, 7, 9\}$, and the corresponding basic subtree repeat table $BSRT(t_1)$ and extended subtree repeat table $ESRT(t_1)$ are illustrated in Fig. 5 and Fig. 6, respectively. □

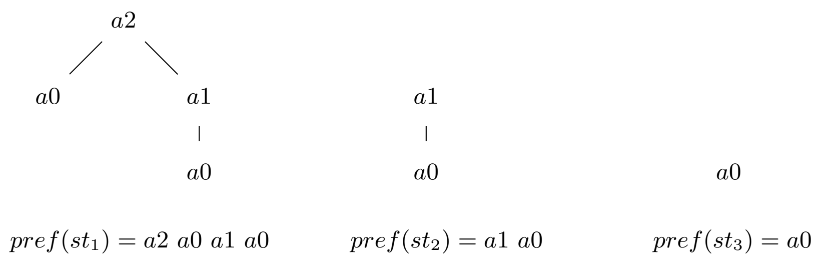


Fig. 4. Subtrees with more than one occurrence in tree t_1 from Example 1, and their prefix notations

List of subtree repeats
$(2, F), (6, S)$
$(4, F), (8, G)$
$(3, F), (5, G), (7, G), (9, G)$

Fig. 5. Basic subtree repeat table $BSRT(t_1)$ from Example 4

Subtree position set	Subtree in prefix notation	List of subtree repeats
$2, 6$	$a2\ a0\ a1\ a0$	$(2, F), (6, S)$
$4, 8$	$a1\ a0$	$(4, F), (8, G)$
$3, 5, 7, 9$	$a0$	$(3, F), (5, G), (7, G), (9, G)$

Fig. 6. Extended subtree repeat table $ESRT(t_1)$ from Examples 4

4.2 Constructing subtree repeat table

A well-known general property of the deterministic string suffix automaton constructed for a string is that a state in which the deterministic string suffix automaton is after reading a substring x corresponds to the set of positions of the last symbols of all occurrences of the substring x in the string [14].

The transitions of the deterministic subtree PDA $M_{dps}(t)$ are extensions of transitions of the deterministic string suffix automaton, and therefore the same general property also holds for the PDA $M_{dps}(t)$: a state in which the PDA $M_{dps}(t)$ is after reading a substring x corresponds to the set of the last nodes of all occurrences of x in prefix notations of subtrees.

The deterministic subtree PDA constructed for a tree t accepts prefix notations of all subtrees of t by the empty pushdown store. States of the deterministic subtree PDA which are multiple subsets and their *spds* sets (see Alg. 2) contain 0, which represents the empty pushdown store, are the positions of the rightmost nodes of all repeating subtrees of the tree. Therefore, the construction of a subtree repetition table for the tree t can be similar to the construction of the deterministic subtree PDA. States of the deterministic subtree PDA which are single subsets are omitted and elements of the subtree repeat table are computed from the above-mentioned states representing the empty pushdown store.

Algorithm 3. Construction of the basic subtree repeat table for a tree t in prefix notation $pref(t)$.
Input: A tree t; prefix notation $pref(t) = a_1 a_2 \dots a_n$, $n \geq 1$.
Output: Basic subtree repeat table $BSRT(t)$.
Method:

```
1. Initially, $BSRT(t) = \emptyset$.
2. Create $M_{npt}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$ by Alg. 1.
3. Let $Q'$ denote a set of states. Let $pdsl(q')$, where $q' \in Q'$, denote a set of
   pairs of integers (the abbreviation pdsl stands for the number of symbols $S$
   in the PushDown Store, and the Length of the subtree.)
4. $Q' = \{[0]\}$, $pdsl([0]) = \{(1, 0)\}$ and [0] is an unmarked state.
5. (a) Select an unmarked state $q'$ from $Q'$ such that $q'$ contains the smallest
       possible state $q \in Q$, where $0 \leq q \leq n$.
   (b) For each $(0, l) \in pdsl(q')$ to $BSRT(t)$ add pairs $(x, Z)$, where
       $x = r - l$, $r \in q'$ and:
         i. $Z = F$ if $x$ is the smallest such number $x$,
        ii. $Z = S$ if $x - 1 \in q''$,
       iii. $Z = G$ otherwise.
   (c) If there is $v > 0$, $(v, w) \in pdsl(q')$, then for each input symbol $a \in \mathcal{A}$:
       Compute state $q'' = \{q : \delta(p, a, \alpha) = (q, \beta)$ for all $p \in q'\}$.
       If $q''$ is not in $Q'$ and $|q''| > 1$, then add $q''$ to $Q'$ and create $pdsl(q'') = \emptyset$.
       Add pairs $(j, k + 1)$, where $(i, k) \in pdsl(q')$, $i > 0$, $j = i + Arity(a) - 1$,
       to $pdsl(q'')$.
   (d) Set the state $q'$ as marked.
6. Repeat step 5 until all states in $Q'$ are marked.                              □
```

Example 5. The basic subtree repeat table $BSRT(t_1)$ constructed by Alg. 3 for tree PDA t_1 from Example 1 is illustrated in Fig. 5. During this construction states [0], [1, 2, 6], [3, 5, 7, 9], [3, 7], [4, 8], [5, 9], and the following set *pdsl* have been constructed:

$$\begin{array}{ll} pdsl([0]) = \{(1,0)\}, & pdsl([3,5,7,9]) = \{(0,1)\}, \\ pdsl([1,2,6]) = \{(2,1)\}, & pdsl([3,7]) = \{(1,2)\}, \\ & pdsl([4,8]) = \{(1,1),(1,3)\}, \\ & pdsl([5,9]) = \{(0,2),(0,4)\}. \end{array}$$

□

It is easy to see that the algorithm of the construction of the extended subtree repeat table is a simple extension of Alg. 3. The extended subtree repeat table contains the repeating subtrees in prefix notation, which can be computed by the following principle: instead of a set *pdsl* of pairs we can use a set *spdsl* of triplets (x, y, z), where $x \in \mathcal{A}^*$, $y, z \geq 0$ (the abbreviation *spdsl* stands for Subtree in prefix notation, the number of symbols S in the PushDown Store, and the Length of the subtree). Initially, $spdsl([0]) = \{(\varepsilon, 1, 0)\}$ (see step 4 of Alg. 3)). When a new element (x, y, z) is added to a $spdsl(q'')$, the subtree in prefix notation x is computed as the concatenation of x', $spdsl(q') = (x', y', z')$, and the input symbol a which is being processed (see step 5c of Alg. 3)).

Theorem 5. *Given a tree t with n nodes, Alg. 3 correctly constructs the basic subtree repeat table $BSRT(t)$ in time $\mathcal{O}(n)$ and space $\mathcal{O}(n)$.*

Proof. The computation of the basic subtree repeat table by Alg. 3 is based on the construction of the deterministic subtree pushdown automaton for $pref(t)$ (see Algs. 1 and 2). The deterministic subtree pushdown automaton accepts all prefix notations of subtrees of the tree t by the empty pushdown store and its states are analogous to states of the deterministic string suffix automaton. Therefore, positions of the rightmost nodes of all occurrences of particular prefix notations correspond to the created states [14, 15]. Only states which are multiple sets and correspond to the rightmost leaves of occurrences of subtrees, which correspond to the empty pushdown store, are added to the basic subtree repeat table. Positions of the roots of these occurrences are computed from the rightmost leaves and from the numbers of nodes of the subtrees.

Both nondeterministic and deterministic subtree pushdown automata have $\mathcal{O}(n)$ states and transitions (see Theorem 4), and the maximal number of repeats of subtrees is also $\mathcal{O}(n)$. Therefore, $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space are used. □

5 Conclusion

More details on tree algorithms using pushdown automata and related information can also be found on [3].

Bibliography

[1] Aho, A.V., Ullman, J.D.: The theory of parsing, translation, and compiling. Prentice-Hall Englewood Cliffs, N.J. (1972)

[2] Apostolico, A., Preparata, F.P.: Optimal off-line detection of repetitions in a string. Theor. Comput. Sci. **22**, 297–315 (1983)

[3] Arbology www pages. Available on: http://www.arbology.org (2009). December 2009

[4] Brodal, G.S., Lyngsø, R.B., Pedersen, C.N.S., Stoye, J.: Finding maximal pairs with bounded gap. In: M. Crochemore, M. Paterson (eds.) CPM, *Lecture Notes in Computer Science*, vol. 1645, pp. 134–149. Springer (1999)

[5] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata (2007). Release October, 12th 2007

[6] Crochemore, M.: An optimal algorithm for computing the repetitions in a word. Inf. Process. Lett. **12**(5), 244–250 (1981)

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

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

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

[10] Janoušek, J., Melichar, B.: On regular tree languages and deterministic pushdown automata. Acta Inf. **46**(7), 533–547 (2009). DOI 10.1007/s00236-009-0104-9

[11] Janoušek, J., Melichar, B.: Subtree and tree pattern pushdown automata for trees in prefix notation (2009). Submitted for publication

[12] London stringology days 2009 conference presentations. Available on: http://www.dcs.kcl.ac.uk/events/LSD&LAW09/, King's College London, London (2009)

[13] Main, M.G., Lorentz, R.J.: An o(n log n) algorithm for finding all repetitions in a string. J. Algorithms **5**(3), 422–432 (1984)

[14] Melichar, B.: Repetitions in text and finite automata. In: L. Cleophas, B. Watson (eds.) Proceedings of the Eindhoven FASTAR Days 2004, pp. 1–46. TU Eindhoven, The Netherlands (2004)

[15] Melichar, B., Holub, J., Polcar, J.: Text searching algorithms. Available on: http://stringology.org/athens/ (2005). Release November 2005

[16] Smyth, B.: Computing Patterns in Strings. Addison-Wesley-Pearson Education Limited, Essex, England (2003)

UDC, DOI: N/A

Subtree Matching by Pushdown Automata*

Tomáš Flouri[1], Jan Janoušek[2], and Bořivoj Melichar[2]

[1] Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo nám. 13, 121 35 Prague 2, Czech Republic
flourtom@fel.cvut.cz,
[2] Department of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague
Kolejní 550/2, 160 00 Prague 6, Czech Republic
{Jan.Janousek,Borivoj.Melichar}@fit.cvut.cz

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

Keywords: subtree, subtree matching, pushdown automata.

1. Introduction

The theory of formal string (or word) languages [2, 16, 24] and the theory of formal tree languages [6, 8, 14] are important parts of the theory of formal languages [23]. While the models of computation of the theory of string languages are finite automata, pushdown automata, linear bounded automata and Turing machines, the most famous models of computation of the theory of tree languages are various kinds of tree automata [6, 8, 14]. Trees, however, can also be seen as strings, for example in their prefix (also called preorder) or postfix (also called postorder) notation. Recently it has been shown that the deterministic pushdown automaton (PDA) is an appropriate model of computation for labelled, ordered, ranked trees in postfix notation and that the trees in postfix notation, acceptable by deterministic PDA, form a proper superclass of the class of regular tree languages, which are accepted by finite tree automata [18].

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

Trees represent one of the fundamental data structures used in Computer Science and thus tree pattern matching, the process of finding occurrences of subtrees in trees, is an important problem with many applications, such as compiler code selection, interpretation of non-procedural languages or various tree finding and tree replacement systems.

Tree pattern matching is often declared to be analogous to the problem of string pattern matching [6]. One of the basic approaches used for string pattern matching can be represented by finite automata constructed for the pattern, which means that the pattern is preprocessed. Examples of these automata are the string matching automata [9, 10, 22, 26]. Given a pattern P of size m, the string matching automaton can be constructed for the pattern P in time linear to m. The constructed string matching automaton accepts the set of words containing pattern P as a suffix, and thus it can find all occurrences of string P in a given text T. The main advantage of this kind of finite automata is that the deterministic string matching automaton can be constructed in time linear to the size of the given pattern P, and the search phase is in time linear to the input text. A generalization of the mentioned string matching problem can be the string matching problem with multiple patterns [1, 22, 26]. Given a set of patterns $P = \{p_1, p_2, \ldots, p_m\}$, the string matching automaton can be constructed in time linear to the number of symbols of patterns in set P. The constructed string matching automaton accepts the set of words having any of the patterns in P as a suffix, and thus it can find all occurrences of strings $p_1, \ldots, p_m$ in a given text T.

Although there are many tree pattern matching methods (see [5–7, 11, 15, 25] for these methods), they fail to present a simple and systematic approach with a linear time searching phase which would also be directly analogous to the basic string pattern matching method.

This paper, being an extended version of [12], presents a new kind of PDAs for trees in prefix and postfix notations called subtree matching PDAs, which are directly analogous to string matching automata and their properties. A subtree matching PDA, constructed from a given tree s, can find all occurrences of subtree s within a given tree t in time $\mathcal{O}(n)$, where n is the number of nodes of t. Subtree matching, as with string matching, can also be generalized to subtree matching with multiple patterns. Subtree matching PDAs can be constructed from a set of trees $P = \{t_1, t_2, \ldots, t_m\}$ in the same manner as string matching automata, retaining their property of linear searching phase $\mathcal{O}(n)$, where n is the number of nodes of the subject tree t.

Moreover, the presented subtree matching PDAs have the following two other properties. First, they are input–driven PDAs [28], which means that each pushdown operation is determined only by the input symbol. The input–driven PDAs can be always determinised [28]. Second, their pushdown symbol alphabets contain just one pushdown symbol and therefore their pushdown store can be implemented by a single integer counter. This means that the presented PDAs can be transformed to counter automata [4, 27], which is a weaker and simpler model of computation than the PDA.

The rest of the paper is organised as follows. Basic definitions are given in section 2. Some properties of subtrees in prefix notation are discussed in the third section. Sections 4 and 5 deal with the subtree matching PDA constructed over a single and multiple patterns, respectively. Section 6 shows the dual principle for the postfix notation and the last section is the conclusion.

2. Basic Notions

2.1. Ranked alphabet, tree, prefix notation, postfix notation, subtree matching

We define notions on trees similarly as they are defined in [2, 6, 8, 14].

We denote the set of natural numbers by $\mathbb{N}$. A *ranked alphabet* is a finite, nonempty set of symbols, each of which has a unique nonnegative *arity* (or *rank*). Given a ranked alphabet $\mathcal{A}$, the arity of a symbol $a \in \mathcal{A}$ is denoted by $Arity(a)$. 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.

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

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

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

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

A *labelled, ordered, ranked and rooted tree* t over a ranked alphabet $\mathcal{A}$ is an ordered dag $t = (N, R)$ with a special node $r \in A$ called the *root* such that
(1) r has in-degree 0,
(2) all other nodes of t have in-degree 1,
(3) there is just one path from the root r to every $f \in N$, where $f \neq r$,
(4) every node $f \in N$ is labelled by a symbol $a \in \mathcal{A}$ and out-degree of a_f is *Arity*(a).

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

Prefix notation pref(t) of a labelled, ordered, ranked and rooted tree t is obtained by applying the following *Step* recursively, beginning at the root of t:
Step: Let this application of *Step* be node a_f. If a_f is a leaf, list a and halt. If a_f is not a leaf, having direct descendants $a_{f_1}, a_{f_2}, \ldots, a_{f_n}$, then list a and subsequently apply *Step* to $a_{f_1}, a_{f_2}, \ldots, a_{f_n}$ in that order.
Postfix notation post(t) of t is formed by changing the last sentence of *Step* to read "Apply *Step* to $a_{f_1}, a_{f_2}, \ldots, a_{f_n}$ in that order and then list a."

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

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

The prefix and postfix notations of tree t_1 are strings *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ and $post(t_1) = a0\ a0\ a1\ a2\ a0\ a1\ a2$, respectively. Trees can be represented graphically, and tree t_1 is illustrated in Fig. 1. □

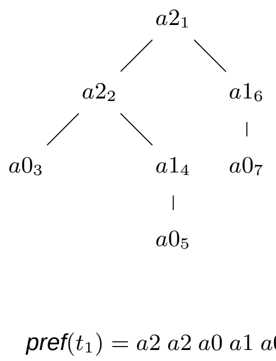


Fig. 1. Tree t_1 from Example 1 and its prefix notation

The number of nodes of a tree t is denoted by $|t|$.

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.

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

2.2. Alphabet, language, pushdown automaton

We define notions from the theory of string languages similarly as they are defined in [2, 16].

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

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

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

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

A pushdown automaton is *input–driven* if its each pushdown operation is determined only by the input symbol.
A language L accepted by a pushdown automaton M is defined in two distinct ways:

1. *Accepting by final state:*

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

2. *Accepting by empty pushdown store:*

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

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

In the rest of the text, we use the following notation for labelling edges when illustrating transition diagrams of various PDAs: For each transition rule $\delta_1(p, a, \alpha) = (q, \beta)$ from the transition mapping δ of a PDA, we label its edge leading from state p to state q by the triplet of the form $a|\alpha \mapsto \beta$.

For more details on pushdown automata see [2, 16].

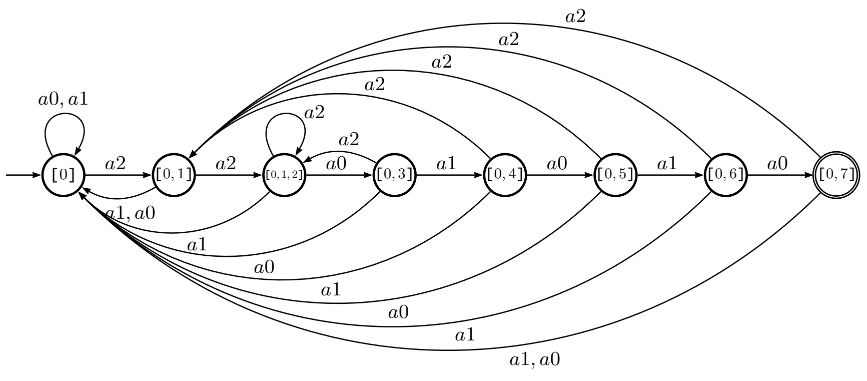


Fig. 2. Transition diagram of deterministic string matching automaton for pattern $x = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 2

2.3. Examples of string matching automaton

Example 2. The transition diagram of the deterministic string matching automaton constructed for string $a2\ a2\ a0\ a1\ a0\ a1\ a0$ is illustrated in Fig. 2. □

Example 3. The transition diagram of the deterministic string matching automaton constructed for a set of strings $P = \{a2\ a2\ a0\ a0\ b0, a2\ b1\ a0\ a0, a2\ a0\ a0\}$ is illustrated in Fig. 3. □

See [2, 9, 22] for definitions of finite automata and construction of the deterministic string matching automaton.

3. Properties of subtrees in prefix notation

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

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 Fig. 1. Tree t_1 contains only subtrees shown in Fig. 4.

Generally, for any tree, the following theorem holds.

Theorem 1. *Given a tree t and its prefix notation pref(t), all subtrees of t in prefix notation are substrings of pref(t).*

Proof. By induction on the height of the subtree.

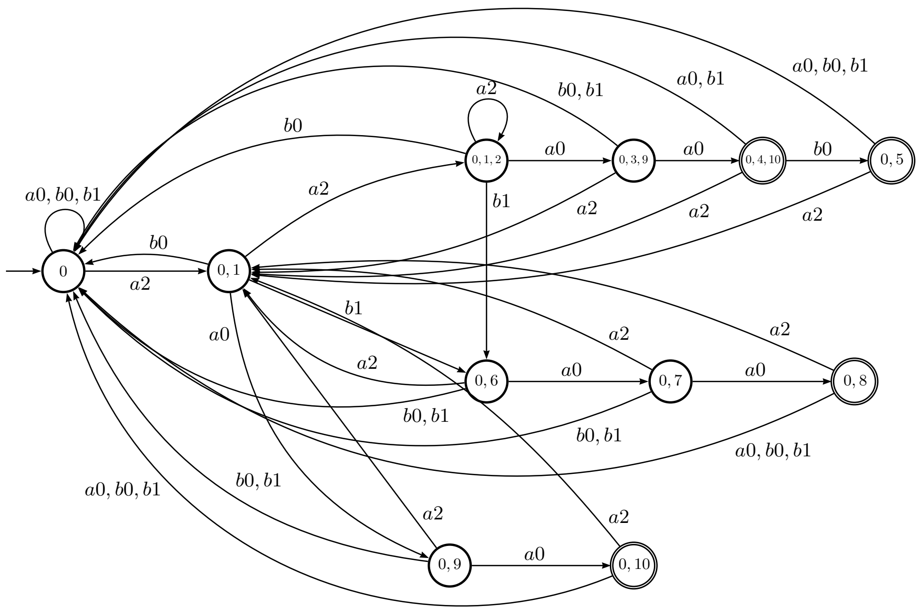
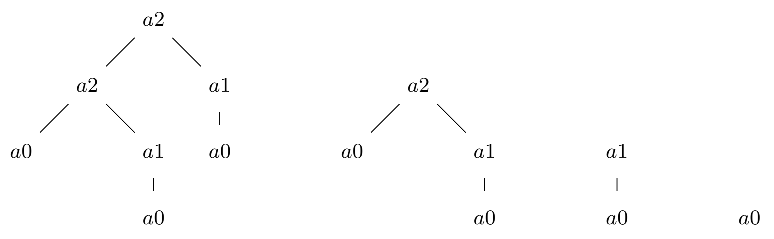


Fig. 3. Transition diagram of deterministic string matching automaton (Aho-Corasick) for patterns $\{a2\ a2\ a0\ a0\ b0,\ a2\ b1\ a0\ a0,\ a2\ a0\ a0\}$

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

Thus, the theorem holds. □

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



pref$(t) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ *pref*$(t) = a2\ a0\ a1\ a0$ *pref*$(t) = a1\ a0$ *pref*$(t) = a0$

post$(t) = a0\ a0\ a1\ a2\ a0\ a1\ a2$ *post*$(t) = a0\ a0\ a1\ a2$ *post*$(t) = a0\ a1$ *post*$(t) = a0$

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

Definition 1. *Let $w = a_1a_2 \ldots a_m$, $m \geq 1$, be a string over a ranked alphabet $\mathcal{A}$. Then, the* arity checksum *ac*$(w) =$ *Arity*$(a_1)+$*Arity*$(a_2)+\ldots+$*Arity*$(a_m)-m+1$= $\sum_{i=1}^{m}$ *Arity*$(a_i) - m + 1$.

Theorem 2. *Let pref(t) and w be a tree t in prefix notation and a substring of pref(t), respectively. Then, w is the prefix notation of a subtree of t, if and only if $ac(w) = 0$, and $ac(w_1) \geq 1$ for each w_1, where $w = w_1x$, $x \neq \varepsilon$.*

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

– *If:* By induction on the height of a subtree st, where $w =$ *pref*(st):
  1. We assume that *Heigth*$(st) = 1$, which means we consider the case $w = a$, where *Arity*$(a) = 0$. Then, *ac*$(w) = 0$. Thus, the claim holds for the case *Height*$(st) = 1$.
  2. Assume that the claim holds for the subtrees $st_1, st_2, \ldots, st_p$ where $p \geq 1$, *Height*$(st_1) \leq m$, *Height*$(st_2) \leq m$, …, *Height*$(st_p) \leq m$ and *ac*(*pref*$(st_1)) = 0$, *ac*(*pref*$(st_2)) = 0$, …, *ac*(*pref*$(st_p)) = 0$.
     We are to prove that it holds also for a subtree of height $m + 1$. Assume $w = a$ *pref*(st_1) *pref*(st_2) …*pref*(st_p), where *Arity*$(a) = p$. Then *ac*$(w) = p+$*ac*(*pref*$(st_1))+$*ac*(*pref*$(st_2))+\ldots+$*ac*(*pref*$(st_p))-(p+1)+1 =$ 0 and *ac*$(w_1) \geq 1$ for each w_1 , where $w = w_1x$, $x \neq \varepsilon$.
     Thus, the claim holds for the case *Height*$(st) = m + 1$.

– *Only if*: Assume *ac*$(w) = 0$, and $w = a_1a_2 \ldots a_k$, where $k \geq 1$, *Arity*$(a_1) = p$. Since *ac*$(w_1) \geq 1$ for each w_1, where $w = w_1x$, $x \neq \varepsilon$, none of the substrings w_1 can be a subtree in prefix notation. This means that the only possibility

for $ac(w) = 0$ is that w is of the form $w = a$ *pref*(t_1) *pref*$(t_2) \dots$*pref*(t_p), where $p \geq 0$, and $t_1, t_2 \dots t_p$ are neighbouring subtrees. In such case, *ac*$(w) = p + 0 - (p + 1) + 1 = 0$.
No other possibility of the form of w for *ac*$(w) = 0$ is possible. Thus, the claim holds.

Thus, the theorem holds. □

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

4. Subtree Matching pushdown automaton

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

Problem 1 (Subtree Matching). Given two trees s and t, find all occurrences of tree s in tree t.

Definition 2. *Let s and pref(s) be a tree and its prefix notation, respectively. Given an input tree t, a subtree pushdown automaton constructed over pref(s) accepts all matches of tree s in the input tree t by final state.*

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

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

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

Example 5. The PDA constructed by Alg. 1, accepting the prefix notation *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ of tree t_1 from Example 1, is the deterministic PDA $M_p(t_1) = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \mathcal{A}, \{S\}, \delta_1, 0, S, \{n\}))$, where the mapping δ_1 is a set of the following transitions:

$$\begin{tikzcd}
\arrow[r] & 0 \arrow[r, "a2|S \mapsto SS"] & 1 \arrow[r, "a2|S \mapsto SS"] & 2 \arrow[r, "a0|S \mapsto \varepsilon"] & 3 \arrow[r, "a1|S \mapsto S"] & 4 \arrow[r, "a0|S \mapsto \varepsilon"] & 5 \arrow[r, "a1|S \mapsto S"] & 6 \arrow[r, "a0|S \mapsto \varepsilon"] & \boxed{7}
\end{tikzcd}$$

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

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

The transition diagram of deterministic PDA $M_p(t_1)$ is illustrated in Fig. 5. Fig. 6 shows the sequence of transitions (trace) performed by deterministic PDA $M_p(t_1)$ for tree t_1 in prefix notation. □

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

Fig. 6. Trace of deterministic PDA $M_p(t_1)$ from Example 5 for tree t_1 in prefix notation *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$

Theorem 3. *Let $M = (\{Q, \mathcal{A}, \{S\}, \delta, 0, S, F)$ be an input–driven PDA whose each transition from δ is of the form $\delta(q_1, a, S) = (q_2, S^i)$, where $i =$ Arity(a). Then, if $(q_3, w, S) \vdash_M^+ (q_4, \varepsilon, S^j)$, then $j =$ **ac**(w).*

Proof. By induction on the length of w:

1. Assume $w = a$. Then, $(q_3, a, S) \vdash_M (q_4, \varepsilon, S^j)$, where $j =$ *Arity*$(a) = ac(a)$. Thus, the claim holds for the case $w = a$.
2. Assume that the claim holds for a string $w = a_1a_2 \ldots a_k$, where $k \geq 1$. This means that $(q_3, a_1a_2 \ldots a_k, S) \vdash_M^k (q_4, \varepsilon, S^j)$, where $j =$ **ac**$(a_1a_2 \ldots a_k)$. We have to prove that the claim holds also for $w = a_1a_2 \ldots a_k\ a$.

It holds that $(q_3, a_1a_2 \ldots a_ka, S) \vdash_M^k (q_4, a, S^j) \vdash_M (q_5, \varepsilon, S^l)$, where $l = j + \textit{Arity}(a) - 1 = ac(w) + \textit{Arity}(a) - 1 = \textit{Arity}(a_1) + \textit{Arity}(a_2) + \ldots + \textit{Arity}(a_k) - k + 1 + \textit{Arity}(a) - 1 = \textit{ac}(a_1a_2 \ldots a_ka)$.
Thus, the claim holds for the case $w = a_1a_2 \ldots a_k\ a$.

Thus, the theorem holds. □

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

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

Proof. By induction on the height of the tree t:

1. If tree t has just one node a, where *Arity*$(a) = 0$, then *Height*$(t) = 0$, *pref*$(t) = a$, $\delta(0, a, S) = (1, \varepsilon) \in \delta$, $(0, a, S) \vdash_{M_p(t)} (1, \varepsilon, \varepsilon)$ and the claim holds for that tree.
2. Assume that claim holds for trees $t_1, t_2, \ldots, t_p$, where $p \geq 1$, *Height*$(t_1) \leq m$, *Height*$(t_2) \leq m$, $\ldots$, *Height*$(t_p) \leq m$, $m \geq 0$.
   We have to prove that the claim holds also for each tree t such that
   pref$(t) = a$ *pref*(t_1)*pref*$(t_2) \ldots$ *pref*(t_p), *Arity*$(a) = p$, and *Height*$(t) \geq m + 1$:
   Since $\delta(0, a, S) = (1, S^p) \in \delta$, and $(0, a\ \textit{pref}(t_1)\textit{pref}(t_2) \ldots \textit{pref}(t_p), S)$
   $\vdash_{M_p(t)} (1, \textit{pref}(t_1)\textit{pref}(t_2) \ldots \textit{pref}(t_p), S^p)$
   $\vdash^*_{M_p(t)} (i, \textit{pref}(t_2) \ldots \textit{pref}(t_p), S^{p-1})$
   $\vdash^*_{M_p(t)} \cdots$
   $\vdash^*_{M_p(t)} (j, \textit{pref}(t_p), S)$
   $\vdash^*_{M_p(t)} (k, \varepsilon, \varepsilon)$,
   the claim holds for that tree.

Thus, the lemma holds. □

We present the construction of the deterministic subtree matching PDA for trees in prefix notation. The construction consists of two steps. First, a nondeterministic subtree matching PDA is constructed by Alg. 2. This nondeterministic subtree matching PDA is an extension of the PDA accepting trees in prefix notation, which is constructed by Alg. 1. Second, the constructed nondeterministic subtree matching PDA is transformed to the equivalent deterministic subtree matching PDA. In spite of the fact that the determinisation of a nondeterministic PDA is not possible generally, the constructed nondeterministic subtree matching PDA is an input–driven PDA and therefore can be determinised [28].

Algorithm 2. Construction of a nondeterministic subtree matching PDA for a tree t in prefix notation *pref*(t).
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation *pref*$(t) = a_1a_2 \ldots a_n$, $n \geq 1$.

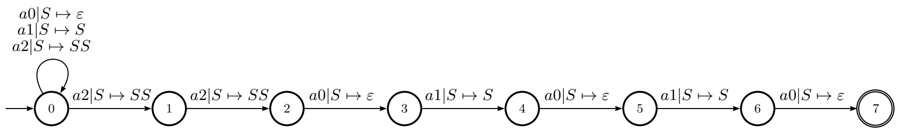


Fig. 7. Transition diagram of nondeterministic subtree matching PDA $M_p(t_1)$ for tree t_1 in prefix notation *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 6

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

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

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

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

The transition diagram of the nondeterministic PDA $M_{nps}(t_1)$ is illustrated in Fig. 7. □

Theorem 4. *Given a tree t and its prefix notation pref(t), the PDA $M_{nps}(t)$ constructed by Alg. 2 is a subtree matching PDA for pref(t).*

Proof. According to Theorem 2, given an input tree t, each subtree in prefix notation is a substring of *pref*(t). Since the PDA $M_{nps}(s)$ has just states and transitions equivalent to the states and transitions, respectively, of the string matching automaton, the PDA $M_{nps}(t)$ accepts all matches of subtree s in tree t by final state. □

For the construction of deterministic subtree PDA, we use the transformation described by Alg. 3. This transformation is based on the well known transformation of nondeterministic finite automaton to an equivalent deterministic one, which constructs the states of the deterministic automaton as subsets of states of the nondeterministic automaton and selects only a set of accessible states (i.e. subsets) [16]. Again, states of the resulting deterministic PDA correspond to subsets of states of the original nondeterministic PDA.

Algorithm 3. Transformation of an input–driven nondeterministic PDA to an equivalent deterministic PDA.
Input: Input–driven nondeterministic PDA $M_{nx}(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, F)$
Output: Equivalent deterministic PDA $M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', q_I, S, F')$.
Method:

```
1. Initially, $Q' = \{\{0\}\}$, $q_I = \{0\}$ and $\{0\}$ is an unmarked state.
2. (a) Select an unmarked state $q'$ from $Q'$.
   (b) For each input symbol $a \in \mathcal{A}$:
       i. $q'' = \{q : \delta(p, a, \alpha) = (q, \beta)$ for all $p \in q'\}$.
       ii. Add transition $\delta'(q', a, S) = (q'', S^{\textit{Arity}}(a))$.
       iii. If $q'' \notin Q$ then add $q''$ to $Q$ and set it as unmarked state.
   (c) Set state $q'$ as marked.
3. Repeat step 2 until all states in $Q'$ are marked.
4. $F' = \{ q' \mid q' \in Q' \wedge q' \cap F \neq \emptyset \}$.
```
□

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

Example 7. The deterministic subtree matching PDA for tree t_1 in prefix notation *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 1 , which has been constructed by Alg. 3 from nondeterministic subtree matching PDA $M_{nps}(t_1)$ from Example 6, is the deterministic PDA $M_{dps}(t_1) = (\{[0], [0, 1], [0, 1, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7]\}, \mathcal{A}, \{S\}, \delta_3, [0], S, \{[0, 7]\})$, where its transition diagram is illustrated in Fig. 9.

We note that the deterministic subtree matching PDA $M_{dps}(t_1)$ has a very similar transition diagram to the deterministic string matching automaton constructed for *pref*(t_1) [9, 22], as can be seen by comparing Figs. 2 and 9. The only difference between the two types of automata are the pushdown operations appearing in the subtree matching PDA, which ensure the validity of the input tree. The input tree is valid only if the pushdown store of the subtree PDA is emptied after the last symbol from the prefix notation of the input tree is read.

Fig. 8 shows the sequence of transitions (trace) performed by the deterministic subtree PDA $M_{dps}(t_1)$ for an input tree t_2 in prefix notation *pref*$(t_2) = a2\ a2\ a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$. The accepting state is $\{0, 7\}$. Fig. 10 depicts the pattern subtree t_1 and input tree t_2. □

Theorem 5. *Given a nondeterministic input–driven PDA $M_{nx}(t) = (Q, \mathcal{A}, \{S\}, \delta, q_0, S, F)$, the deterministic PDA $M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', \{q_0\}, S, F')$ which is constructed by Alg. 3 is equivalent to PDA $M_{nx}(t)$.*

State	Input	PDS
$\{0\}$	$a2\ a2\ a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	S
$\{0,1\}$	$a2\ a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	SS
$\{0,1,2\}$	$a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	SSS
$\{0,1,2\}$	$a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	$SSSS$
$\{0,3\}$	$a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	SSS
$\{0,4\}$	$a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$	SSS
$\{0,5\}$	$a1\ a0\ a1\ a1\ a2\ a0\ a0$	SS
$\{0,6\}$	$a0\ a1\ a1\ a2\ a0\ a0$	SS
$\{0,7\}$	$a1\ a1\ a2\ a0\ a0$ match	S
$\{0\}$	$a1\ a2\ a0\ a0$	S
$\{0\}$	$a2\ a0\ a0$	S
$\{0,1\}$	$a0\ a0$	SS
$\{0\}$	$a0$	S
$\{0\}$	ε	ε

Fig. 8. Trace of deterministic subtree PDA $M_{dps}(t_1)$ from Example 7 for an input subtree t_2 in prefix notation *pref*$(t_2) = a2\ a2\ a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$

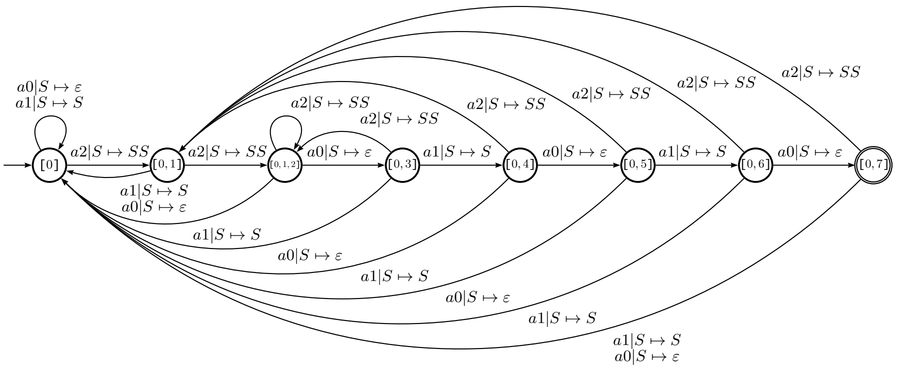
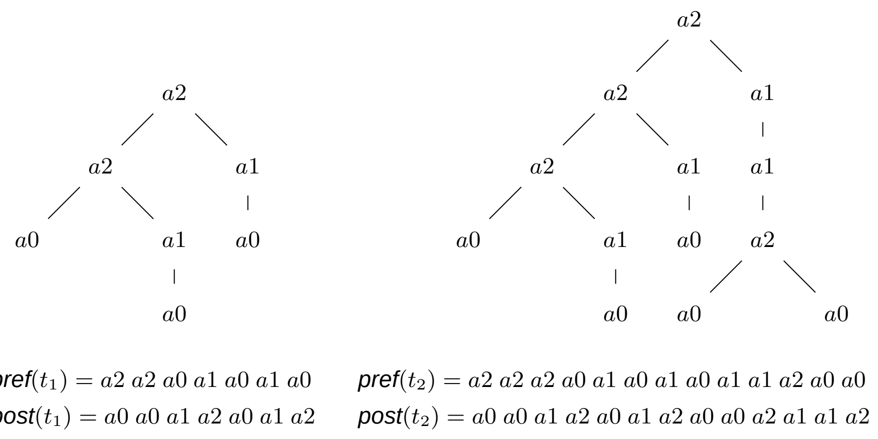


Fig. 9. Transition diagram of deterministic PDA $M_{dps}(t_1)$ for tree t_1 in prefix notation *pref*$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$ from Example 7

Proof. First, we prove the following claim by induction on i:
(*): $(q'_1, w, S) \vdash^i_{M_{dx}(t)} (q'_2, \varepsilon, S^j)$ if and only if
$q'_2 = \{p : (q, w, S) \vdash^i_{M_{nx}(t)} (p, \varepsilon, S^j) \text{ for some } q \in q'_1\}$.

1. Assume i=1.
   - *if*: if $(q'_1, a, S) \vdash_{M_{dx}(t)} (q'_2, \varepsilon, S^j)$, then there exists a state $q \in q'_1$, where $(q, a, S) \vdash_{M_{nx}(t)} (p, \varepsilon, S^j)$, $p \in q'_2$.
   - *only if*: if $(q, a, S) \vdash_{M_{nx}(t)} (p, \varepsilon, \beta)$, then for each $q'_1 \in Q'$, where $q \in q'_1$, it holds that $(q'_1, a, S) \vdash_{M_{dx}(t)} (q'_2, \varepsilon, S^j)$, where $p \in q'_2$.
2. Assume that claim (*) holds for $i = 1, 2, \ldots, k$, $k \geq 1$.
   This means that $(q'_1, w, S) \vdash^k_{M_{dx}(t)} (q'_2, \varepsilon, S^j)$ if and only if
   $q'_2 = \{p : (q, S, w) \vdash^k_{M_{nx}(t)} (p, \varepsilon, S^j) \text{ for some } q \in q'_1\}$. We have to prove that claim (*) holds also for $i = k + 1$.
   - *if*: if $(q'_1, w, S) \vdash^k_{M_{dx}(t)} (q'_2, a, S^l) \vdash_{M_{dx}(t)} (q'_3, \varepsilon, S^j)$, then there exists a state $q \in q'_2$, where $(q, a, S^l) \vdash_{M_{nx}(t)} (p, \varepsilon, S^j)$, $p \in q'_3$.
   - *only if*: if $(q_0, pref(t), S) \vdash^k_{M_{nx}(t)} (q, a, S^l) \vdash_{M_{nx}(t)} (p, \varepsilon, S^j)$, then for each $q'_1 \in Q'$, where $q \in q'_1$, it holds that $(q'_1, a, S^l) \vdash_{M_{dx}(t)} (q'_2, \varepsilon, S^j)$, where $p \in q'_2$.

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



pref$(t_1) = a2\ a2\ a0\ a1\ a0\ a1\ a0$
post$(t_1) = a0\ a0\ a1\ a2\ a0\ a1\ a2$
pref$(t_2) = a2\ a2\ a2\ a0\ a1\ a0\ a1\ a0\ a1\ a1\ a2\ a0\ a0$
post$(t_2) = a0\ a0\ a1\ a2\ a0\ a1\ a2\ a0\ a0\ a2\ a1\ a1\ a2$

Fig. 10. Trees t_1 and t_2 from Example 7 along with their prefix and postfix notations

Theorem 6. *Given a tree t with n nodes in its prefix or postfix notation, the deterministic subtree matching PDA $M_{pds}(t)$ constructed by Alg. 2 and 3 is made of exactly $n+1$ states, one pushdown symbol and $|\mathcal{A}|(n+1)$ transitions.*

Proof. Let $M_{nps}(t) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \{n\}$ be an automaton constructed from tree t with a prefix notation *pref*$(t) = a_1\ a_2\ \dots\ a_n$ over ranked alphabet $\mathcal{A}$ by Alg. 2. We will prove that this automaton is directly analogous to the string matching automaton and accepts the same language if we ignore the pushdown operations, which actually do not affect the process of determinisation as M_{pds} is an input–driven automaton. From Alg 2 and 3, $M_{nps}(t)$ has transitions $\delta(0, a, S) = (0, S^{\textit{Arity}(a)})$ for all $a \in \mathcal{A}$ and $\delta(i-1, a_i, S) = (i, \varepsilon, S^{\textit{Arity}(a_i)})$. The proof is a mutual induction of the following $n + 1$ statements:
(1) $\delta^*(0, w, S) = (0, \varepsilon, S^{\textit{ac}(w)})$, $w \in \mathcal{A}^*$.
(2) $\delta^*(0, w, S) = (1, \varepsilon, S^{\textit{ac}(w)})$ if and only if $w = w_1 a_1$, $w_1 \in \mathcal{A}^*$
(i) $\delta^*(0, w, S) = (i-1, \varepsilon, S^{\textit{ac}(w)})$ if and only if $w = w_1 a_1 a_2 \dots a_{i-1}$, $w_1 \in \mathcal{A}^*$

1. Assume that $|w| = 0$, which means $w = \varepsilon$. Statement (1) holds, since $\delta^*(0, \varepsilon, S) = (0, \varepsilon, S)$. Statements ($i$), $1 < i \leq n+1$, do not hold as $\delta^*(0, \varepsilon, S)$ contains, from its basic definition, only $(0, \varepsilon, S)$.
2. Assume $w = w_1 a$, where $w_1 \in \mathcal{A}^k$, that is $|w_1| = k$ and $a \in \mathcal{A}$. We may assume that statements (i) $1 < i \leq n + 1$ hold for w_1, and we need to prove them for w. We assume the inductive hypothesis for k and prove it for $k + 1$.
   (a) There exists a series of transitions $(0, w_1, S) \vdash^* (0, \varepsilon, S^{\textit{ac}(w_1)})$, since $\delta(0, a, S) = (0, \varepsilon, S^{\textit{Arity}(a)})$ are transitions of automaton M_{nps}.Thus statement (1) is proved for w.
   (b) We now prove statements i, where $1 < i \leq n + 1$:
      – *If:* Assume that $w_1 = w_2 a_1 a_2 \dots a_{i-2}$, where $w_2 \in \mathcal{A}^*$ and $a = a_{i-1}$. By statement ($i - 1$) applied to w_1, we know from our induction hypothesis that there exists a series of transitions $(0, w_1, S) \vdash^* (i - 2, \varepsilon, S^{\textit{ac}(w_1)})$. Since for all $1 \leq j \leq n$ there exists a transition $\delta(j - 1, a_j, S) = (j, S^{\textit{Arity}(a_j)})$, we conclude that $\delta^*(0, w, S) = (i - 1, \varepsilon, S^{\textit{ac}(w)})$.
      – *Only if:* Suppose there exists a series of transitions $(0, w, S) \vdash^* (i - 1, \varepsilon, S^{\textit{ac}(w)})$. From the inductive assumption we know that there exists a series of transitions $(0, w_1, S) \vdash^* (i-2, \varepsilon, S^{\textit{ac}(w_1)})$. By statement $(i-1)$ applied to w_1, we know that $w_1 = w_2 a_1 a_2 \dots a_{i-2}$. Thus $w = w_2 a_1 a_2 \dots a_{i-1}$, and we have proved statement (i).

Thus, from statements $1, \dots, n+1$, if we ignore the pushdown operations, M_{pds} accepts the language $L = \{w.\textit{pref}(t)\}$, where $w \in \mathcal{A}^*$. Since the subtree matching PDA is directly analogous to the string matching automaton, we can use the proof from [10, 22] for space and time complexities. □

Theorem 7. *Given an input tree t with n nodes, the searching phase of the deterministic subtree matching automaton constructed by Algs. 2 and 3 is $\mathcal{O}(n)$.*

Proof. The searching phase consists of reading tree t once, symbol by symbol from left to right. The appropriate transition is taken each time a symbol is read, resulting in exactly n transitions. Each transition consumes a constant time because the time of each pushdown operation is limited by the maximal arity of nodes. Occurrences of the subtree to find are matched by transitions leading to the final states. □

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

5. Multiple subtree matching

In this section we present a generalization of Problem 1. We deal with the construction of subtree matching PDA over a finite set of trees. The whole concept is demonstrated with an example.

Problem 2 (Multiple Subtree Matching). Given a tree t and a set of m trees $P = \{t_1, t_2, \dots, t_m\}$, find all occurrences of trees $t_1, t_2, \dots, t_m$ in tree t.

Definition 3. *Let $P = \{t_1, t_2, \dots, t_m\}$ be a set of m trees and pref$(t_i), 1 \le i \le m$ be the prefix notation of the i-th tree in P. Given an input tree t, a subtree pushdown automaton constructed over set P accepts all matches of subtrees $t_1, t_2, \dots, t_m$ in the input tree t by final state.*

Similarly as in Section 4, our method begins with a PDA which accepts trees $t_1, t_2, \dots, t_m$ in their prefix notation. The construction of this PDA is described by Alg. 4

Algorithm 4. Construction of a PDA accepting a set of trees $P = \{t_1, t_2, \dots, t_m\}$ in their prefix notation.
Input: A set of trees $P = \{t_1, t_2, \dots, t_m\}$ over a ranked alphabet $\mathcal{A}$; prefix notation *pref*$(t_i) = a_1 a_2 \dots a_{n_i}$, $1 \le i \le m$, $n_i \ge 1$.
Output: PDA $M_p(P) = (\{0, 1, 2, \dots, q\}, \mathcal{A}, \{S\}, \delta, 0, S, F)$.
Method:

```
1. Let $q \leftarrow 0$ and $F \leftarrow \emptyset$
2. For each tree $t_i = a^i_1\ a^i_2\ \dots\ a^i_{|t_i|}$, $1 \le i \le m$, do
   (a) Let $l \leftarrow 0$
   (b) For $j = 1$ to $|t_i|$ do
       i. If the transition $\delta(l, a^i_j, S)$ is not defined then
          A. Let $q \leftarrow q + 1$
          B. Create a transition $\delta(l, a^i_j, S) \leftarrow (q, S^{\mathit{Arity}(a^i_j)})$
          C. Let $l \leftarrow q$
       ii. Else if transition $\delta(l, a^i_j, S)$ is defined
          A. $l \leftarrow p$ where $(p, \gamma) \leftarrow \delta(l, a_j, S)$
   (c) $F \leftarrow F \cup \{l\}$
```

Example 8. Consider a set of trees $P = \{t_1, t_2, t_3\}$, with their prefix notations being *pref*$(t_1) = a2\ a2\ a0\ a0\ b0$, *pref*$(t_2) = a2\ b1\ a0\ a0$ and *pref*$(t_3) = a2\ a0\ a0$. The deterministic PDA constructed by Alg. 4 accepting the prefix notation of trees in P is $M_p(P) = (\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}, \mathcal{A}, \{S\}, \delta_1, 0, S, \{5, 8, 10\}))$, where mapping δ_1 is a set of the following transitions:

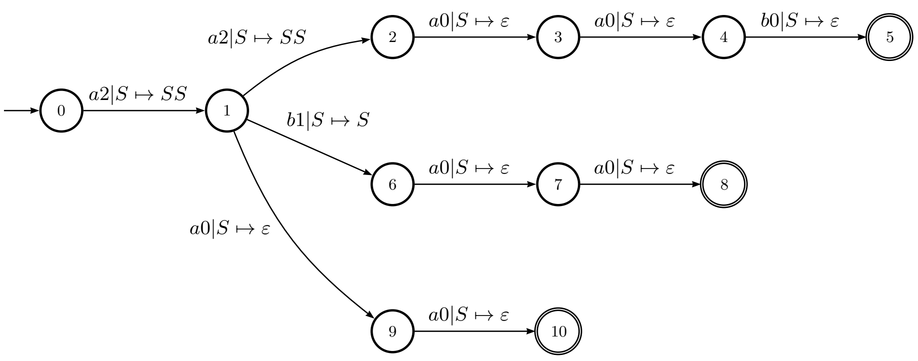

0
1
2
3
4
5
6
7
8
9
10
$a2|S \mapsto SS$
$a2|S \mapsto SS$
$a0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$
$b0|S \mapsto \varepsilon$
$b1|S \mapsto S$
$a0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$

Fig. 11. Transition diagram of deterministic PDA $M_p(P)$ accepting the trees with prefix notation $\{a2\ a2\ a0\ a0\ b0, a2\ b1\ a0\ a0, a2\ a0\ a0\}$ from Example 8

$$\begin{aligned}
\delta_1(0, a2, S) &= (1, SS)\\
\delta_1(1, a2, S) &= (2, SS)\\
\delta_1(2, a0, S) &= (3, \varepsilon)\\
\delta_1(3, a0, S) &= (4, \varepsilon)\\
\delta_1(4, b0, S) &= (5, \varepsilon)\\
\delta_1(1, b1, S) &= (6, S)\\
\delta_1(6, a0, S) &= (7, \varepsilon)\\
\delta_1(7, a0, S) &= (8, \varepsilon)\\
\delta_1(1, a0, S) &= (9, \varepsilon)\\
\delta_1(9, a0, S) &= (10, \varepsilon)
\end{aligned}$$

The transition diagram of deterministic PDA $M_p(P)$ is illustrated in Fig. 11.
Fig. 12 shows the sequence of transitions (trace) performed by deterministic PDA $M_p(P)$ for trees $t_1, t_2, t_3 \in P$ in prefix notation. □

The correctness of the deterministic PDA constructed by Alg. 4, which accepts trees in prefix notation, is described by the following lemma.

Lemma 2. *Given a set of k trees $P = \{t_1, t_2, \dots, t_m\}$ and their prefix notation pref(t_i), $1 \le i \le m$, the PDA $M_p(P) = (\{0, 1, 2, \dots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, F)$, where $1 +$ min$(|t_1|, |t_2|, \dots, |t_m|) \le n \le 1 + \sum_{j=1}^{k} |t_j|$, constructed by Alg. 4 accepts pref(t_i), where $1 \le t_i \le m$.*

Proof. By induction on the height of trees $t_1, t_2, \dots, t_m$:

1. If trees $t_1, t_2, \dots, t_m$ have just one node, $a_1, a_2, \dots, a_k$ respectively, where *Arity*$(a_i) = 0$, for all $1 \le i \le k$, then *Height*$(t_i) = 0$, *pref*$(t_i) = a_i$, $\delta(0, a_i, S) = (i, \varepsilon) \in \delta$, $(0, a_i, S) \vdash_{M_p(P)} (i, \varepsilon, \varepsilon)$ for all $1 \le i \le k$ and the claim holds.

State	Input	Pushdown Store
0	$a2\ a2\ a0\ a0\ b0$	S
1	$a2\ a0\ a0\ b0$	$S\ S$
2	$a0\ a0\ b0$	$S\ S\ S$
3	$a0\ b0$	$S\ S$
4	$b0$	S
5	ε	ε
accept		
0	$a2\ b1\ a0\ a0$	S
1	$b1\ a0\ a0$	$S\ S$
6	$a0\ a0$	$S\ S$
7	$a0$	S
8	ε	ε
accept		
0	$a2\ a0\ a0$	S
1	$a0\ a0$	$S\ S$
9	$a0$	S
10	ε	ε
accept		

Fig. 12. Trace of deterministic PDA $M_p(P)$ from Example 8 for trees in prefix notation $\{a2\ a2\ a0\ a0\ b0,\ a2\ b1\ a0\ a0,\ a2\ a0\ a0\}$

2. Assume that the claim holds for trees $t^1_1, t^1_2, \ldots, t^1_{p_1}, t^2_1, t^2_2, \ldots, t^2_{p_2}, \ldots, t^k_1, t^k_2, \ldots, t^k_{p_k}$ where $p_i \geq 1$ for all $1 \leq i \leq k$, *Height*$(t^i_1) \leq m$, *Height*$(t^i_2) \leq m$, ..., *Height*$(t^i_p) \leq m$, $m \geq 0$, for all $1 \leq i \leq k$.
   We have to prove that the claim holds also for each tree t_i, $1 \leq i \leq k$, such that
   pref$(t_i) = a_i$ *pref*(t^i_1)*pref*$(t^i_2) \ldots$*pref*$(t^i_{p_i})$, *Arity*$(a_i) = p_i$, and *Height*$(t_i) \geq m+1$:
   Since $\delta(0, a_i, S) = (i, S^p) \in \delta$, and $(0, a$ *pref*(t^i_1)*pref*$(t^i_2) \ldots$*pref*$(t^i_{p_i}), S)$
   $\vdash_{M_p(t_i)} (i,$*pref*(t^i_1)*pref*$(t^i_2) \ldots$*pref*$(t^i_{p_i}), S^p)$
   $\vdash^*_{M_p(t_i)} (j^i,$*pref*$(t^i_2) \ldots$*pref*$(t^i_{p_i}), S^{p_i-1})$
   $\vdash^*_{M_p(t_i)} \cdots$
   $\vdash^*_{M_p(t_i)} (\ell^i,$*pref*$(t^i_{p_i}), S)$
   $\vdash^*_{M_p(t_i)} (f^i, \varepsilon, \varepsilon)$
   the claim holds for that tree.

Thus, the lemma holds. □

The deterministic subtree matching PDA for multiple tree patterns in prefix notation can be constructed in a similar fashion to the subtree matching PDA for a single pattern. First, the PDA accepting a set of trees in their prefix notations, constructed by Alg. 4, is used to construct a nondeterministic subtree matching PDA by Alg. 5. The constructed nondeterministic subtree matching PDA is then transformed to the equivalent deterministic subtree matching PDA.

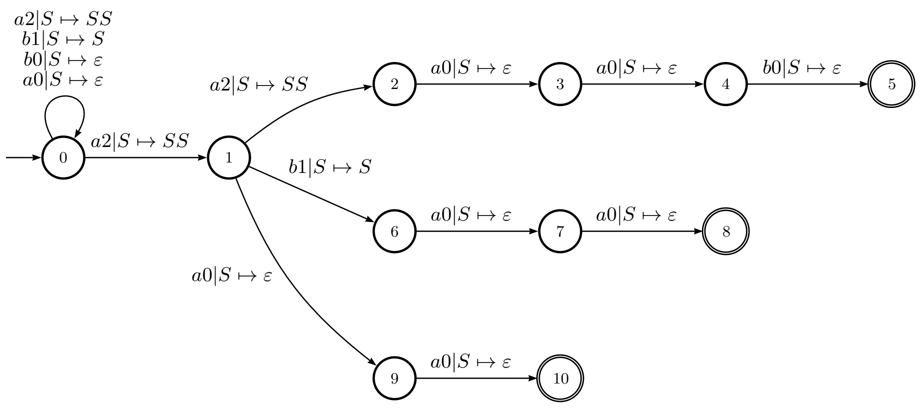

$a2|S \mapsto SS$
$b1|S \mapsto S$
$b0|S \mapsto \varepsilon$
$a0|S \mapsto \varepsilon$
0
$a2|S \mapsto SS$
1
$a2|S \mapsto SS$
2
$a0|S \mapsto \varepsilon$
3
$a0|S \mapsto \varepsilon$
4
$b0|S \mapsto \varepsilon$
5
$b1|S \mapsto S$
6
$a0|S \mapsto \varepsilon$
7
$a0|S \mapsto \varepsilon$
8
$a0|S \mapsto \varepsilon$
9
$a0|S \mapsto \varepsilon$
10

Fig. 13. Transition diagram of nondeterministic subtree matching PDA $M_p(P)$ constructed over trees in set P from Example 9

Algorithm 5. Construction of a nondeterministic subtree matching PDA for a set of trees $P = \{t_1, t_2, \ldots, t_m\}$ in their prefix notation.
Input: A tree t over a ranked alphabet $\mathcal{A}$; prefix notation *pref*$(t) = a_1 a_2 \ldots a_n$, $n \geq 1$.
Output: Nondeterministic subtree matching PDA $M_{nps}(t) = (Q, \mathcal{A}, \{S\}, \delta, 0, S, F)$.
Method:

1. Create PDA $M_{nps}(t)$ as PDA $M_p(t) = (Q, \mathcal{A}, \{S\}, \delta, 0, S, F)$ by Alg. 4.
2. For each symbol $a \in \mathcal{A}$ create a new transition $\delta(0, a, S) = (0, S^{\textit{Arity}(a)})$, where $S^0 = \varepsilon$.

□

Example 9. The subtree matching PDA constructed by Alg. 2 over the set of trees P from Example 8 is the nondeterministic PDA $M_{nps}(P) = (\{0,1,2,3,4,5,6,7,8,9,10\}, \mathcal{A}, \{S\}, \delta_2, 0, S, \{5,8,10\}))$, where mapping δ_2 is a set of the following transitions:

$$\begin{array}{ll}
\delta_2(0,a2,S) = (1,SS) & \\
\delta_2(1,a2,S) = (2,SS) & \delta_2(0,a2,S) = (0,SS) \\
\delta_2(2,a0,S) = (3,\varepsilon) & \delta_2(0,b1,S) = (0,S) \\
\delta_2(3,a0,S) = (4,\varepsilon) & \delta_2(0,b0,S) = (0,\varepsilon) \\
\delta_2(4,b0,S) = (5,\varepsilon) & \delta_2(0,a0,S) = (0,\varepsilon) \\
\delta_2(1,b1,S) = (6,S) & \\
\delta_2(6,a0,S) = (7,\varepsilon) & \\
\delta_2(7,a0,S) = (8,\varepsilon) & \\
\delta_2(1,a0,S) = (9,\varepsilon) & \\
\delta_2(9,a0,S) = (10,\varepsilon) &
\end{array}$$

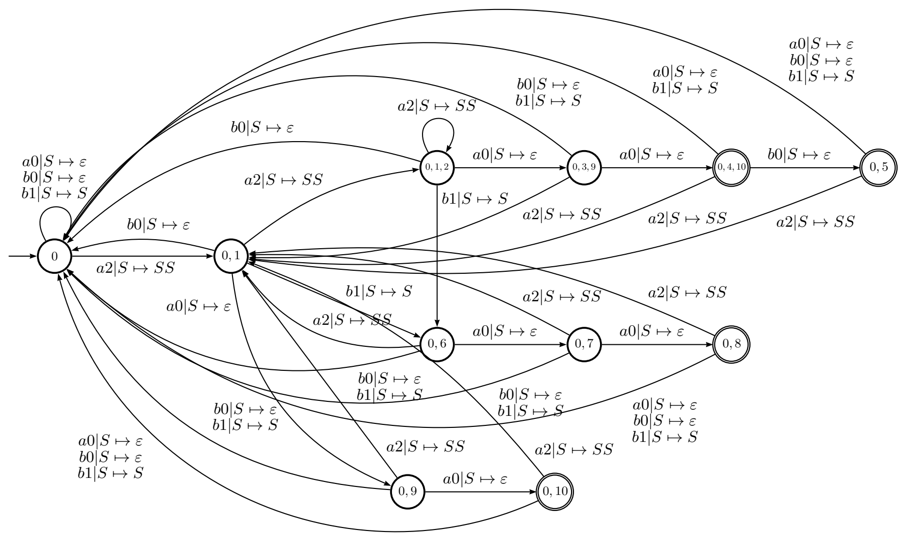


Fig. 14. Transition diagram of deterministic PDA $M_{dps}(P)$ constructed over trees in set P from Example 10

The transition diagram of nondeterministic PDA $M_{nps}(P)$ is illustrated in Fig. 13. □

Theorem 8. *Given a set of m trees $P = \{t_1, t_2, \dots, t_m\}$ and their prefix notation pref(t_i), $1 \le i \le m$, the PDA $M_{nps}(P)$ constructed by Alg. 5 is a subtree matching PDA for tree patterns $t_1, t_2, \dots, t_m$.*

Proof. According to Theorem 2, given an input tree t, each subtree in prefix notation is a substring of *pref*(t). Since the PDA $M_{nps}(P)$ has just states and transitions equivalent to the states and transitions, respectively, of the Aho-Corasick string matching automaton , the PDA $M_{nps}(P)$ accepts all matches of subtrees $t_1, t_2, \dots, t_m$ in tree t by final state. □

For the construction of deterministic subtree PDA, we use the transformation described by Alg. 3 from Section 4.

The deterministic subtree matching automaton $M_{dps}(P)$ for a set of trees $P = \{t_1, t_2, \dots, t_m\}$ with prefix notations *pref*$(t_i), 1 \le i \le k$ is demonstrated by the following example.

Example 10. The deterministic subtree matching PDA for the set of trees P from Example 8, constructed by Alg. 3 from the nondeterministic subtree matching PDA $M_{nps}(P)$ from Example 9, is $M_{dps}(P) = (\{[0], [0,1], [0,1,2], [0,3,9], [0,4,10], [0,5], [0,6], [0,7], [0,8], [0,9], [0,10]\}, \mathcal{A}, \{S\}, \delta_3, [0], S, \{[0,4,10], [0,5], [0,8], [0,10]\})$, with its transition diagram illustrated in Fig. 14.

We note that the deterministic subtree matching PDA $M_{dps}(P)$ has a very similar transition diagram to the Aho-Corasick string matching automaton constructed for the strings representing the prefix notations of trees in set P from Example 8 (see also [1, 9, 22]), as can be seen by comparing Figs. 4 and 14.

Fig. 15 shows the sequence of transitions (trace) performed by the deterministic subtree PDA $M_{dps}(P)$ for the input tree t having prefix notation *pref*$(t) = a2\ a2\ a2\ a0\ a0\ a2\ a2\ a0\ a0\ b0\ \ a2\ b1\ a0\ a0$. The final states are $\{[0, 4, 10], [0, 5], [0, 8], [0, 10]\}$. Fig. 16 depicts the pattern subtrees from set P and the input tree t. □

State	Input		PDS
$\{0\}$	$a2\ a2\ a2\ a0\ a0\ a2\ a2\ a0\ a0\ b0\ a2\ b1\ a0\ a0$		S
$\{0,1\}$	$a2\ a2\ a0\ a0\ a2\ a2\ a0\ a0\ b0\ a2\ b1\ a0\ a0$		SS
$\{0,1,2\}$	$a2\ a0\ a0\ a2\ a2\ a0\ a0\ b0\ a2\ b1\ a0\ a0$		SSS
$\{0,1,2\}$	$a0\ a0\ a2\ a2\ a0\ a0\ b0\ a2\ b1\ a0\ a0$		$SSSS$
$\{0,3,9\}$	$a0\ a2\ a2\ a0\ a0\ b0\ a2\ b1\ a0\ a0$		SSS
$\{0,4,10\}$	$a2\ a2\ a0\ a0\ b0\ a2\ b1\ a0\ a0$	match	SS
$\{0,1\}$	$a2\ a0\ a0\ b0\ a2\ b1\ a0\ a0$		SSS
$\{0,1,2\}$	$a0\ a0\ b0\ a2\ b1\ a0\ a0$		$SSSS$
$\{0,3,9\}$	$a0\ b0\ a2\ b1\ a0\ a0$		SSS
$\{0,4,10\}$	$b0\ a2\ b1\ a0\ a0$	match	SS
$\{0,5\}$	$a2\ b1\ a0\ a0$	match	S
$\{0,1\}$	$b1\ a0\ a0$		SS
$\{0,6\}$	$a0\ a0$		SS
$\{0,7\}$	$a0$		S
$\{0,8\}$	ε	match	ε

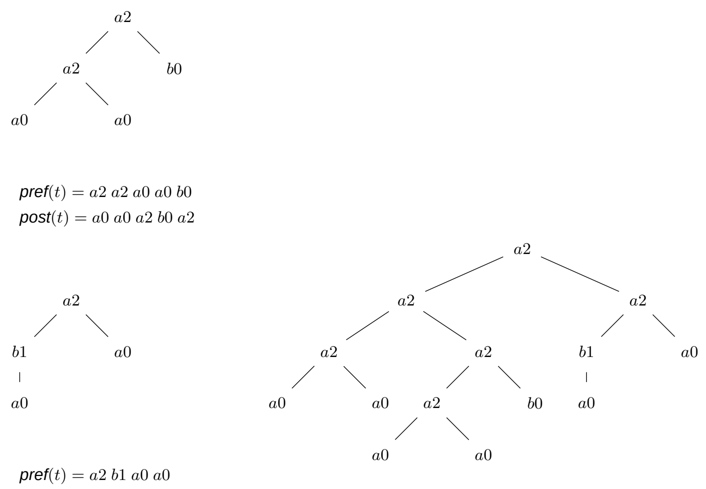
Fig. 15. Trace of deterministic subtree PDA $M_{dps}(P)$ from Example 10 for tree t_2 in prefix notation *pref*$(t) = a2\ a2\ a2\ a0\ a0\ a2\ a2\ a0\ a0\ b0\ a2\ b1\ a0\ a0$.

Theorem 9. *Given a set of m trees $P = \{t_1, t_2, \dots, t_m\}$ over a ranked alphabet $\mathcal{A}$, the deterministic subtree matching PDA $M_{pds}(P)$ is constructed by Alg. 5 and 3 in time $\Theta(|\mathcal{A}|s)$, requires $\Theta(|\mathcal{A}|s)$ storage, where $s = \sum_{i=1}^{m} |t_i|$, and its pushdown store alphabet consists of one symbol.*

Proof. Since the subtree matching PDA for multiple patterns is directly analogous to the Aho-Corasick string matching automaton (this can be proved from proof of Theorem 6), we can use the proof from [1] and [26]. □

Theorem 10. *Given an input tree t with n nodes, the searching phase of the deterministic subtree matching automaton constructed by Algs. 2 and 3 over a set of m trees P is $\mathcal{O}(n)$.*

Proof. The searching phase consists of reading tree t once, symbol by symbol from left to right. The appropriate transition is taken each time a symbol is read,



pref$(t) = a2\ a2\ a0\ a0\ b0$

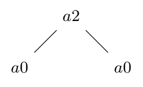
post$(t) = a0\ a0\ a2\ b0\ a2$

pref$(t) = a2\ b1\ a0\ a0$

post$(t) = a0\ b1\ a0\ a2$

pref$(t) = a2\ a2\ a2\ a0\ a0\ a2\ a2\ a0\ a0\ b0\ a2\ b1\ a0\ a0$

post$(t) = a0\ a0\ a2\ a0\ a0\ a2\ b0\ a2\ a2\ a0\ b1\ a0\ a2\ a2$



pref$(t) = a2\ a0\ a0$

post$(t) = a0\ a0\ a2$

Fig. 16. Pattern subtrees from set P and the input tree from Example 10 along with their prefix and postfix notations

resulting in exactly n transitions. Each transition consumes a constant time because the time of each pushdown operation is limited by the maximal arity of nodes. Occurrences of the subtree to find are matched by transitions leading to the final states. □

6. Subtree matching in postfix notation

In this section we show the dual principle for the postfix notation. Theorems 11 and 12 present the direct analogy of properties of the prefix and postfix notations. Theorem 13 is analogous to Theorem 3.

Theorem 11. *Given a tree t and its postfix notation post(t), all subtrees of t in postfix notation are substrings of post(t).*

Theorem 12. *Let post(t) and w be a tree t in postfix notation and a substring of post(t), respectively. Then, w is the postfix notation of a subtree of t, if and only if $ac(w) = 0$, and $ac(w_1) \leq -1$ for each w_1, where $w = xw_1$, $x \neq \varepsilon$.*

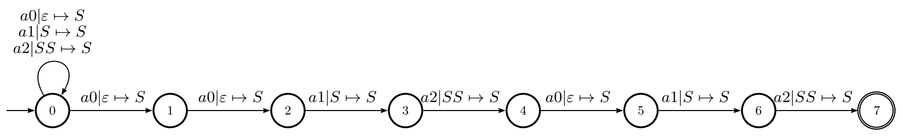


Fig. 17. Transition diagram of nondeterministic subtree matching PDA $M_p(t_1)$ for tree t_1 in postfix notation *post*$(t_1) = a0\ a0\ a1\ a2\ a0\ a1\ a2$ from Example 6

Theorem 13. *Let* $M = (\{Q, \mathcal{A}, \{S\}, \delta, 0, S, F)$ *be an input–driven PDA whose each transition from* δ *is of the form* $\delta(q_1, a, S^i) = (q_2, S)$*, where* $i =$ *Arity*(a)*. Then, if* $(q_3, w, \varepsilon) \vdash_M^+ (q_4, \varepsilon, S^j)$*, then* $j = -ac(w) + 1$*.*

From the above Theorems, we can easily transform Algorithms 1-5 to work with the postfix notation of trees. The only change required is in the pushdown operations. All transitions of the form $\delta(q, a, S) = (p, S^{\mathit{Arity}(a_i)})$ must be changed to the form $\delta(q, a, S^{\mathit{Arity}(a_i)}) = (p, S)$. The subtree matching PDA also requires no initial pushdown store symbol, while after processing a valid tree in postfix notation, the pushdown store contains a single symbol 'S'.

Fig. 17 illustrates the nondeterministic subtree matching PDA $M_p(t_1)$ constructed from the postfix notation of the tree from Example 6.

Fig. 18 illustrates the deterministic subtree matching PDA $M_dps(t_1)$ constructed from the postfix notation of the tree from Example 6.

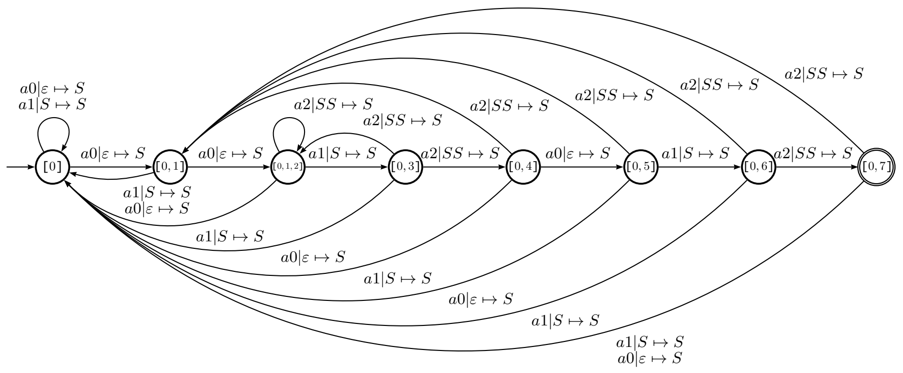


Fig. 18. Transition diagram of deterministic PDA $M_{dps}(t_1)$ for tree t_1 in postfix notation *post*$(t_1) = a0\ a0\ a1\ a2\ a0\ a1\ a2$ from Example 7

7. Conclusion

We have introduced a new kind of pushdown automata: subtree matching PDAs for trees in prefix and postfix notations. These pushdown automata are in their properties analogous to string matching automata, which are widely used in stringology [9, 10, 22, 26].

Regarding specific tree algorithms whose model of computation is the standard deterministic pushdown automaton, we have recently introduced principles of other three new algorithms. First, the tree pattern matching PDA [13, 21] which is an extension of the subtree matching PDA presented in this paper. Second, the subtree and tree pattern PDAs, which represent a complete index of a given tree by preprocessing it. Searching for all occurrences of a subtree or a tree pattern of size m is then performed in time linear to m and not depending on the size of the preprocessed tree [17, 19, 21]. These automata representing indexes of trees are analogous in their properties to the string suffix and factor automata [9, 10, 22, 26]. Third, a method on how to find all repeats of connected subgraphs in trees with the use of subtree or tree pattern PDAs [21, 20]. More details on these results and related information can also be found on [3].

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM 18(6), 333–340 (1975)
2. Aho, A.V., Ullman, J.D.: The theory of parsing, translation, and compiling. Prentice-Hall Englewood Cliffs, N.J. (1972)
3. Arbology www pages. Available on: http://www.arbology.org/ (2009), december 2009
4. Berstel, J.: Transductions and Context-Free Languages. Teubner Studienbucher, Stuttgart (1979)
5. Chase, D.R.: An improvement to bottom-up tree pattern matching. In: POPL. pp. 168–177 (1987)
6. Cleophas, L.: Tree Algorithms. Two Taxonomies and a Toolkit. Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven (2008)
7. Cole, R., Hariharan, R., Indyk, P.: Tree pattern matching and subset matching in deterministic ($\log^3$)-time. In: SODA. pp. 245–254 (1999)
8. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata (2007), release October, 12th 2007
9. Crochemore, M., Hancart, C.: Automata for matching patterns. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 2 Linear Modeling: Background and Application, chap. 9, pp. 399–462. Springer–Verlag, Berlin (1997)
10. Crochemore, M., Rytter, W.: Jewels of Stringology. World Scientific, New Jersey (1994)
11. Dubiner, M., Galil, Z., Magen, E.: Faster tree pattern matching. J. ACM 41(2), 205–213 (1994)
12. Flouri, T., Janoušek, J., Melichar, B.: Subtree matching by deterministic pushdown automata. In: Ganzha, M., Paprzycki, M. (eds.) Proceedings of the IMCSIT, Vol. 4. pp. 659–666. IEEE Computer Society Press (2009)

13. Flouri, T., Janoušek, J., Melichar, B.: Tree pattern matching by deterministic pushdown automata (2009), draft
14. Gecseg, F., Steinby, M.: Tree languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3 Beyond Words. Handbook of Formal Languages, pp. 1–68. Springer–Verlag, Berlin (1997)
15. Hoffmann, C.M., O'Donnell, M.J.: Pattern matching in trees. J. ACM 29(1), 68–95 (1982)
16. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation. Addison-Wesley, Boston, 2nd edn. (2001)
17. Janoušek, J.: String suffix automata and subtree pushdown automata. In: Holub, J., Žďárek, J. (eds.) Proceedings of the Prague Stringology Conference 2009. pp. 160–172. Czech Technical University in Prague, Czech Republic (2009), available on: http://www.stringology.org/event/2009
18. Janoušek, J., Melichar, B.: On regular tree languages and deterministic pushdown automata. Acta Inf. 46(7), 533–547 (2009)
19. Janoušek, J., Melichar, B.: Subtree and tree pattern pushdown automata for trees in prefix notation (2009), submitted for publication
20. Janoušek, J., Melichar, B.: Finding repeats of subtrees in a tree using pushdown automata (2010), submitted for publication
21. London stringology days 2009 conference presentations. Available on: http://www.dcs.kcl.ac.uk/events/LSD&LAW09/, King's College London, London (2009)
22. Melichar, B., Holub, J., Polcar, J.: Text searching algorithms. Available on: http://stringology.org/athens/ (2005), release November 2005
23. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer–Verlag, Berlin (1997)
24. Rozenberg, G., Salomaa, A. (eds.): Vol. 1: Word, Language, Grammar, Handbook of Formal Languages. Springer–Verlag, Berlin (1997)
25. Shankar, P., Gantait, A., Yuvaraj, A.R., Madhavan, M.: A new algorithm for linear regular tree pattern matching. Theor. Comput. Sci. 242(1-2), 125–142 (2000)
26. Smyth, B.: Computing Patterns in Strings. Addison-Wesley-Pearson Education Limited, Essex, England (2003)
27. Valiant, L.G., Paterson, M.: Deterministic one-counter automata. In: Automaten theorie und Formale Sprachen. pp. 104–115 (1973)
28. Wagner, K., Wechsung, G.: Computational Complexity. Springer–Verlag, Berlin (2001)

Author One is Tomáš Flouri

Author Two is Jan Janoušek

Author Three is Bořivoj Melichar

Received: Month DD, 20YY; Accepted: Month DD, 20YY.