On Tree Pattern Matching by Pushdown Automata

T. Flouri

Tree pattern matching is an important operation in Computer Science on which a number of tasks such as mechanical theorem proving, term-rewriting, symbolic computation and non-procedural programming languages are based on. Work has begun on a systematic approach to the construction of tree pattern matchers by deterministic pushdown automata which read subject trees in prefix notation. The method is analogous to the construction of string pattern matchers: for given patterns, a non-deterministic pushdown automaton is created and then it is determinised. In this first paper, we present the proposed non-deterministic pushdown automaton which will serve as a basis for the determinisation process, and prove its correctness.

Keywords: tree, tree pattern, pushdown automaton, tree pattern matching.

1 Introduction

Tree pattern matching is one of the fundamental problems with many applications, and is often declared to be analogous to the problem of string pattern matching [3, 5, 17]. String pattern matching is the problem of finding all occurrences of string patterns and their positions in a given text. A model of computation for string pattern matching is the finite automaton [8]. One of the basic approaches used for string pattern matching is represented by finite automata which are constructed for string patterns, which means that the patterns are preprocessed. Given a text of size n, such finite automata typically perform the search phase in time linear to n (see [8, 9, 19] for a survey).

Tree pattern matching is the problem of finding all occurrences and their positions of matches of tree patterns in a subject tree. Although many tree pattern matching methods have been described [4, 5, 6, 10, 11, 13, 14, 16, 17, 18, 20, 21], most of them fail to provide a search phase in linear time (based on the size of the subject tree) or have huge memory requirements.

This paper presents the first attempt to perform tree pattern matching by a unified and systematic approach using pushdown automata. We present the first, basic, non-deterministic model of a pushdown automaton performing tree pattern matching. The goal of this research is to provide a method for determinising the non-deterministic model of the proposed pushdown automaton, which will make linear time (based on the size of the subject tree) pattern matching possible.

2 Basic notions

2.1 Ranked alphabet, tree, prefix notation, tree pattern

We define notions on trees similarly as they are defined in [1, 5, 7, 15, 17].

We denote the set of natural numbers by \( \mathbb{N} \). A ranked alphabet \( \mathcal{A} \) is a finite, non-empty set of symbols, each of which has a unique, non-negative arity (or rank). Given a ranked alphabet \( \mathcal{A} \), the arity of a symbol \( a \in \mathcal{A} \) is denoted \( \text{Arity}(a) \). 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, 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 identifiers for a short declaration of symbols with arity. For instance, \( a2 \) is a short declaration of a binary symbol \( a \). Based on the concepts of graph theory (see [1]), a labeled, 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 \( f_{l-1} \) and enters node \( f_l \) for \( 1 \leq l \leq n \). A cycle is a path \((f_0, f_1, \ldots, f_n)\), where \( f_0 = f_n \). An ordered Directed Acyclic Graph (DAG) is an ordered directed graph that has no cycle. Labeling 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 \( f \) labeled by symbol \( a \).

A labeled, ordered, ranked tree \( t \) over a ranked alphabet \( \mathcal{A} \) is an ordered DAG \( t = (N, R) \) with a special node 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 root \( r \) to every \( f \in N \), where \( f \neq r \),
4. every node \( f \in N \) is labeled by a symbol \( a \in \mathcal{A} \) and out-degree of \( a_f \) is \( \text{Arity}(a) \),
5. nodes labeled by nullary symbols are called leaves.

The prefix notation \( \text{pref}(0) \) of a labeled, ordered, ranked tree \( t \) is obtained by applying the following Step recursively, beginning at the root of \( t \):
Step: Let this application of Step to be node \( a_j \). If \( a_j \) is a leaf, list \( a_j \) and halt. If \( a_j \) is not a leaf, let its direct descendants be \( a_{f_1}, a_{f_2}, \ldots, a_{f_n} \). Then list \( a_j \) 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, 7, 15, 17], labeled ordered ranked trees are defined with the use of ordered ranked ground terms. Ground terms can be regarded as labeled, 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 \( A = \{a_2, a_1, a_0\} \). Consider a tree \( t_1 \) over \( A \) \( t_1 = \langle \{a_2a_1a_2, a_0a_3, a_1a_4, a_0a_5, a_1a_6, a_0a_7\} \rangle \), where \( R \) is a set of the following ordered sequences of pairs:

- \( ((a_2a_1a_2), (a_0a_3)) \)
- \( ((a_2a_0a_3), (a_2a_1a_4)) \)
- \( ((a_1a_4, a_0a_3)) \)
- \( ((a_1a_6, a_0a_7)) \).

The prefix notation of tree \( t_1 \) is

\[ \text{pref}(t_1) = a2a2a0a1a0a1a0 \].

Trees can also be represented graphically and tree \( t_1 \) is illustrated in Fig. 1. The height of a tree \( t \), denoted by \( \text{Height}(t) \), is defined as the maximal length of a path from the root of \( t \) to a leaf of \( t \).

![Fig. 1: Tree pattern from Example 1 and its prefix notation](image)

To define a tree pattern, we use a special nullary symbol \( S \), where \( S \notin A \), which serves as a placeholder for any subtree. A tree pattern is defined as a labeled, ordered, ranked tree over ranked alphabet \( A \cup \{S\} \). By analogy, a tree pattern in prefix notation is defined as a labeled, ordered, ranked tree over ranked alphabet \( A \cup \{S\} \) in prefix notation.

A pattern \( p \) with \( k \geq 0 \) occurrences of the symbol \( S \) matches a tree \( t \) at node \( n \) if there exist subtrees \( t_{i_1}, t_{i_2}, \ldots, t_{i_k} \) (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 a tree

\( t_1 = \langle \{a_2a_1a_2, a_0a_3, a_1a_4, a_0a_5, a_1a_6, a_0a_7\} \rangle \), from Example 1, which is illustrated in Fig. 1. Consider a tree pattern \( p_1 \) over \( A \cup \{S\} \), \( p_1 = \langle \{a_2a_0a_3, a_1a_6, a_0\} \rangle \), where \( R \) is a set of the following ordered sequences of pairs:

- \( ((a_2a_0a_3), (a_2a_1a_4)) \)
- \( ((a_1a_6, a_0)) \).

The prefix notation of tree pattern \( p_1 \) is

\[ \text{pref}(p_1) = a2S a1S \].

The tree pattern \( p_1 \) is illustrated in Fig. 2 and has two occurrences in tree \( t_1 \), matching at nodes \( a_2 \) and \( a_0 \) of \( t_1 \).

![Fig. 2: Tree pattern from Example 2 and its prefix notation](image)

### 2.2 Alphabet, Language, Context-Free Grammar, Pushdown Automaton

Let an alphabet be a finite, nonempty set of symbols. A language over an alphabet \( A \) is a set of strings over \( A \). The symbol \( A^* \) denotes the set of all strings over \( A \), including the empty string, which is denoted by \( \varepsilon \). Set \( A^+ \) is defined as \( A^+ = A^* \setminus \{\varepsilon\} \). Similarly for string \( x \in A^* \), the 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 \( G = (N, T, P, S) \), where \( N \) and \( T \) are finite, disjoint sets of nonterminal and terminal symbols, respectively. \( P \) is a finite set of rules \( A \rightarrow \alpha \), 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 \( \beta \) from \( P \) is of the form \( A \rightarrow a\alpha \), where \( a \in T \) and \( \alpha \in N^* \).

The relation \( \Rightarrow \) is called derivation, if \( \alpha \gamma \Rightarrow \alpha\beta\gamma \), \( A \in N \), and \( \alpha, \beta, \gamma \in (N \cup T)^* \), then rule \( A \rightarrow \beta \) is in \( P \). Symbols \( \Rightarrow^* \), and \( \Rightarrow^* \) are used for the transitive, and the transitive and reflexive closure of \( \Rightarrow \), respectively. A rightmost derivation \( \Rightarrow_r \) is a relation \( \alpha A \Rightarrow \alpha\beta x \), where \( x \in T^* \). The relation \( A \Rightarrow^* \alpha A \beta \) is called recursion. Right recursion is a \( A \Rightarrow^* \alpha A \).

Hidden-left recursion is a \( A \Rightarrow^* Ba\beta \), where \( Ba \Rightarrow^* \).

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 derivation tree is a labeled, ordered tree representing a syntactic structure of a string \( w \), generated by the grammar \( G \).
Its root is labeled by the start symbol $S$ and its leaves are labeled by terminal symbols or empty strings. Each interior node of the tree is labeled by a non-terminal symbol $A$, and the children of such a node are labeled, from left to right, by symbols from the right-hand side $\beta$ of a rule $A \to \beta \in P$. A derivation $S \Rightarrow^* w$ corresponds to a derivation tree whose leaves, if read from left to right, are labeled with the string $w$.

A CFG $G$ is unambiguous if each string $w \in L(G)$ has just one derivation tree in the CFG $G$.

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

An (extended) non-deterministic pushdown automaton (non-deterministic PDA) is a seven-tuple $M = (Q, \Sigma, G, \delta, q_0, Z_0, F)$, where $Q$ is a finite set of states, $\Sigma$ is an input alphabet, $G$ is a pushdown store alphabet, $\delta$ is a mapping from $Q \times (\Sigma \cup \{\varepsilon\}) \times G^* \times (Q \times A^* \times G\dagger)$ into a set of finite subsets of $Q \times A^* \times G\dagger$, $q_0 \in Q$ is an 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 $(q, w, x) \in Q \times A^* \times G\dagger$ denotes the configuration of a pushdown automaton. In this paper, the top of the pushdown store $x$ is always on the left-hand side. The initial configuration of a pushdown automaton is a triplet $(q_0, w, Z_0)$ for the input string $w \in A^*$. The relation $(q, a.w, \beta.a) \vdash_M (p, w, \beta.a) \subset (Q \times A^* \times G\dagger) \times (Q \times A^* \times G\dagger)$ 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 $\vdash^k_M$, $\vdash^T_M$, $\vdash^R_M$, respectively. A pushdown automaton $M$ is deterministic (deterministic PDA), if it holds:

1. $|\delta(q, a, \gamma)| \leq 1$ for all $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $\gamma \in G\dagger$.
2. If $\delta(q, a, \alpha) \neq \emptyset$, $\delta(q, a, \beta) \neq \emptyset$ and $\alpha \neq \beta$, then $\alpha$ is not a prefix of $\beta$ and $\beta$ is not a prefix of $\alpha$.
3. If $\delta(q, a, \alpha) \neq \emptyset$, $\delta(q, e, \beta) \neq \emptyset$, then $\alpha$ is not a prefix of $\beta$ and $\beta$ is not a prefix of $\alpha$.

The class of languages accepted by non-deterministic 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, that is, for which no deterministic PDA can be constructed.

A pushdown automaton is input-driven if each of its pushdown operations is determined only by the input symbol.

### 2.3 LR(0) parsing

Given a string $w$, an LR(0) parser for a CFG $G = (N, \Sigma, P, S)$ reads the string $w$ from left to right without any backtracking and is implemented by a deterministic PDA. A string $\gamma$ is a viable prefix if $\gamma$ is a prefix of $\alpha \beta$, and $\delta \Rightarrow_{\alpha \beta}^* \alpha A x \Rightarrow_{\alpha \beta}^* \alpha A x$ is a rightmost derivation in $G$; the string $\beta$ is called the handle. We use the term complete viable prefix to refer to $\alpha \beta$ in its entirety. During parsing, each content of the pushdown store corresponds 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 ending by the handle $\beta$, the parser performs a reduction by 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.

2. When the contents of the pushdown store correspond to a viable prefix ending by the handle $\beta$, the parser performs a shift with 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 CFG $G$ is LR(0) if the two conditions for $G$:

1. $S \Rightarrow^*_{\alpha \beta} \alpha A w \Rightarrow^*_{\alpha \beta} \alpha \beta w$,
2. $S \Rightarrow^*_{\alpha \beta} \gamma B x \Rightarrow^*_{\alpha \beta} \alpha \beta y$,

imply that $\alpha Ay = \gamma B x$, that is $\alpha = \gamma$, $A = B$, and $x = y$.

If the CFG $G$ 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 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. Then, the optimized resulting LR(0) parser reads one symbol on each of its transitions [2].

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

1. Accepting by final state:
   
   $L(M) = \{x : \delta(q_0, x, Z_0) \Rightarrow^*_{\alpha \beta} (q, e, \gamma) \wedge x \in A^* \wedge \gamma \in G^* \wedge q \in F\}$.

2. Accepting by empty pushdown store:
   
   $L_e(M) = \{x : \delta(q_0, x, Z_0) \Rightarrow^*_{\alpha \beta} (q, e, \gamma) \wedge x \in A^* \wedge q \in Q\}$.

### 3 A Deterministic Pushdown Automaton accepting trees in prefix notation

The prefix notation of a tree can be generated by a grammar $G = (N, \Sigma, P, S)$, having rules $P$ of the following form:

1. $S \rightarrow a_0$
2. $S \rightarrow a_1S$
3. $S \rightarrow a_2SS$
   
   $\ldots$

4. $S \rightarrow a_nS^{n-1}$

- \[ M = (Q, \Sigma, G, \delta, q_0, Z_0, F) \]
Since the grammar is \( LR(0) \), belonging to the subclass of context-free grammars named as deterministic context-free grammars, the generated language belongs to the class of deterministic context-free languages and can be recognised by a deterministic pushdown automaton.

In this section we present the deterministic pushdown automaton \( M = (\emptyset, \mathcal{A}, (S), \delta, 0, S, 0) \), accepting arbitrary trees in prefix notation by empty pushdown store. The transitions of the automaton are in the form \( \delta(0, x, S) = (0, S^{\text{Arity}(x)}) \), for each \( x \in \mathcal{A} \). Have in mind that \( S^0 = \varepsilon \). The automaton, which is input-driven, is depicted in Fig. 3. This particular automaton will be the basic building block for our non-deterministic automaton, which will serve as a pattern matcher.

\[ a_{i1} | S \rightarrow S^i \]

Fig. 3: Deterministic tree automaton

**Theorem 1.**

The automaton presented in section 3 accepts valid trees in prefix notation by empty pushdown store.

**Proof.** There are three possible types of input to be given to the automaton:

1. A valid input, which represents the prefix notation of a tree.
2. An invalid input, in which there exists a prefix that represents a valid prefix notation of a tree.
3. An invalid input which is the prefix notation of some (unknown) tree.

To show that the first type of (valid) input is accepted by our automaton by empty pushdown store, we use strong induction.

Let \( P(n) \) be a predicate defined over all integers \( n \). Predicate \( P(0) \) is true, if trees of height \( i \) are accepted by the presented deterministic PDA. We define the base case and the inductive step in the following manner:

1. **Base case:** \( P(0) \) is true.
2. **Inductive case:** \( P(0), P(1), \ldots, P(n-1) \Rightarrow P(n) \)

Since the initial pushdown store symbol is symbol \( S \), statement (1) is true, as the only trees of height \( 0 \) are trees with only one node \( x \) having arity 0. The transition to be taken is \( \delta(0, x, S) = (0, \varepsilon) \), removing the initial symbol \( S \) from the pushdown store.

Each tree of depth \( n \) can be represented as a root \( x \) of arity \( k \), where each of its children nodes can be subtrees of height at most \( n - 1 \). According to the inductive case assumption (2), each of the subtree can be accepted by the automaton by empty pushdown store, removing the initial pushdown store symbol \( S \). Since the root appends \( k-1 \) symbols \( S \) on the pushdown store, the \( k \) subtrees remove \( k \) symbols \( S \) from the pushdown store, leaving it empty. As a result, we have proven that our automaton accepts valid input (prefix notations of trees) by empty pushdown store.

In the second case (invalid input, in which there exists such a prefix that represents a valid prefix notation of a tree), the pushdown store will be emptied at the moment the prefix which represents a valid prefix notation is read.

In the third case, which is apparent from the first case, the pushdown store will not be emptied.

We have proved that the automaton accepts only valid input (prefix notations of trees) by empty pushdown store.

**Corollary 1.** Processing an arbitrary tree with the automaton introduced in Section 3 results in one symbol being removed from the top of the pushdown store.

### 4 Searching Non-Deterministic Pushdown Automaton

In this section we present the Searching Non-Deterministic Pushdown Automaton (SNPDA), performing tree pattern matching.

The structure of an SNPDA accepting all occurrences of the tree pattern for a given tree in prefix notation is described by Algorithm 4. We note that the SNPDA accepts the matched patterns by final state.

The SNPDA is loosely based on the searching non-deterministic finite automaton, which is used for pattern matching in strings, as described in [19]. It is constructed by extending the deterministic pushdown automaton presented in Section 3 with states and transitions corresponding to the given tree pattern.

We start by constructing the deterministic pushdown automaton \( M = (q_0, \mathcal{A}, G, \delta, q_0, S, F) \) presented in Section 3, where \( \delta \) is defined as \( \delta(q_0, x, S) = (\{q_0, S^{\text{Arity}(x)}\}) \) for each \( x \in \mathcal{A} \). The prefix notation of the pattern is read from left to right; for each node \( x \) (except the special nullary symbol \( S \)) at position \( i \) (the position of the first node is 1), the following steps are carried out:

1. Create a new state \( q_i \).
2. In case \( i \neq 1 \) do step 3, otherwise do step 4.
3. Define a new transition \( \delta(q_{i-1}, x, S) = (\{q_i, S^{\text{Arity}(x)}\}) \).
4. Append a new transition: \( \delta(q_0, x, S) = \delta(q_0, x, S) \cup (\{q_i, S^{\text{Arity}(x)}\}) \).

In case the nullary symbol \( S \) is found at position \( i \), the following steps are carried out:

1. Create a new state \( q_i \).
2. Define a symbol \( #_j \), where \( #_j \notin G \).
3. Add the new symbol \( #_j \) to the pushdown store symbol set \( G: G = G \cup \{#_j\} \).

© Czech Technical University Publishing House
4. Define a new transition \( \delta(q_{i-1}, \varepsilon, S) = \{(q_0, S \#_j)\} \).

5. Define a new transition \( \delta(q_0, \varepsilon, \#_j) = \{(q_0, \varepsilon)\} \).

The last created state (state \( q_n \)) is set as final (that is, \( F = \{q_n\} \)). Examples of PDAs constructed by Algorithm 1 for various patterns are shown in Fig. 4–6.

input : \( x = x_1x_2\ldots x_n \) – prefix notation of a tree over a ranked alphabet \( \mathcal{A} \)

output : \( M \) – a non-deterministic pushdown automaton

1. \( Q \leftarrow \emptyset \)
2. \( G \leftarrow \{S\} \)
3. \( \delta \leftarrow \emptyset \)
4. for \( i \leftarrow 0 \) to \( n \) do \( Q \leftarrow Q \cup \{q_i\} \)
5. \( F \leftarrow \{q_n\} \)
6. foreach \( y \in \mathcal{A} \) do \( \delta(q_0, y, S) \leftarrow (q_0, S^{arity}(y)) \)
7. \( j \leftarrow 0 \)
8. for \( i = 1 \) to \( n \) do
9. if \( x_i = S \) then
10. \( j \leftarrow j + 1 \)
11. \( G \leftarrow G \cup \{\#_j\} \)
12. \( \delta(q_{i-1}, \varepsilon, S) \leftarrow (q_0, S \#_j) \)
13. \( \delta(q_0, \varepsilon, \#_j) \leftarrow (q_0, \varepsilon) \)
14. else
15. \( \delta(q_{i-1}, x_i, S) \leftarrow (q_j, S^{arity}(x_i)) \)
16. end
17. end
18. \( M = (Q, \mathcal{A}, G, \delta, q_0, S, F) \)

Algorithm 1: Construction of a searching nondeterministic pushdown automaton

Theorem 2.

The SNPDA constructed by Algorithm 1 finds all occurrences of the tree pattern in a subject tree by final state.

Proof. We provide a sketch of the proof.

A tree pattern, for which the SNPDA \( M = (Q, \mathcal{A}, G, \delta, q_0, S, F) \) is constructed, has either the form \( p = x_1x_2\ldots x_n \) (form 1), where \( x_1 \in \mathcal{A} \) and \( x_i \in \mathcal{A} \cup \{S\} \) for \( i > 1 \), or the form \( p = S \) (form 2). The automaton is non-deterministic at state \( q_0 \) due to the transitions

\[
\delta(q_0, x_1, S) = \{(q_0, S^{arity}(x_1))\}
\]

in the case of form 1, or due to the transition

\[
\delta(q_0, \varepsilon, S) = \{(q_0, S \#_j)\}
\]

conflicting with all other transitions in the case of form 2.

Because of this property, the SNPDA can follow more than one path at each input symbol. It can either cycle through state \( q_0 \) or move to state \( q_1 \) and on to \( q_n \), in case the input symbols match the pattern.

At the point of a nullary symbol in the pattern, an \( \varepsilon \)-transition leading to state \( q_0 \) is taken, replacing the top of the pushdown store (which is a symbol \( S \)) with \( S \#_j \), where \( j \) is distinct for each \( S \) in the tree pattern. Using this method, we simulate a new pushdown store on the top of the current pushdown store. Symbol \( \#_j \) denotes the end of the new, simulated pushdown store. From Corollary 1, we know that reading a tree by cycling through state \( q_0 \) removes 1 symbol from the pushdown store. As a result, the top of the pushdown store will be \( \#_j \), which indicates that a tree (required by the respective symbol \( S \) in the tree pattern) has been processed. The \( \#_j \)-transition can now be taken to resume pattern matching at the point after the respective symbol \( S \) in the pattern.

Note that the SNPDA in Fig. 6 is input-driven and thus it can be determinised in the same way as finite automata. The deterministic version is illustrated in Fig. 7.
5 Conclusion and future work

In this paper we have presented an innovative method of tree pattern matching by pushdown automata. We have introduced a non-deterministic model of the searching pushdown automaton, which correctly accepts all occurrences of a pattern in a given tree presented in its prefix notation.

Our goal is to perform determinisation of this automaton, which will lead to linear time (to the size of the subject tree) searching of patterns, as in the case of string pattern matching by deterministic finite automata. Work on the determinisation of this automaton has already begun and the first results were presented at the London Stringology Days conference held at King’s College, London [12].

Acknowledgements

The research described in this paper was supervised by Prof. Bořivoj Melichar, DrSc. of FEE CTU in Prague and supported by the Czech Ministry of Education, Youth and Sports under research program MSM 6840770014, and by the Czech Science Foundation as project No. 201/09/0807.

References


Tomáš Flouri
e-mail: flourt1@fel.cvut.cz

Department of Computer Science and Engineering
Czech Technical University in Prague
Faculty of Electrical Engineering
Karlov Nám. 13
121 35 Prague 2, Czech Republic