Construction of a Pushdown Automaton Accepting a Postfix Notation of a Tree Language Given by a Regular Tree Expression

Tomáš Pecka
Department of Theoretical Computer Science, Faculty of Information Technology
Czech Technical University in Prague
Tomas.Pecka@fit.cvut.cz

Jan Trávníček
Department of Theoretical Computer Science, Faculty of Information Technology
Czech Technical University in Prague
Jan.Travnicek@fit.cvut.cz

Radomír Polách
Department of Theoretical Computer Science, Faculty of Information Technology
Czech Technical University in Prague
Radomir.Polach@fit.cvut.cz

Jan Janoušek
Department of Theoretical Computer Science, Faculty of Information Technology
Czech Technical University in Prague
Jan.Janousek@fit.cvut.cz

Abstract
Regular tree expressions are a formalism for describing regular tree languages, which can be accepted by a finite tree automaton as a standard model of computation. It was proved that the class of regular tree languages is a proper subclass of tree languages whose linear notations can be accepted by deterministic string pushdown automata. In this paper, we present a new algorithm for transforming regular tree expressions to equivalent real-time height-deterministic pushdown automata that accept the trees in their postfix notation.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases tree, regular tree expression, pushdown automaton

Digital Object Identifier 10.4230/OASIcs.SLATE.2018.6

1 Introduction

The theories of formal string languages and formal tree languages are important parts of computer science. Strings and trees are fundamental data structures. Tree languages processing has become very popular in the recent years. For example, we can find practical usages in the area of processing markup languages (like XML) or abstract syntax trees. Traditionally, problems on trees are solved using various kinds of tree automata [5]. However, trees can also be represented by strings, for instance in their prefix or postfix notation obtained by preorder or postorder traversal of the tree, respectively. It was proved by
Janoušek and Melichar [9] that the class of regular tree languages is a proper subclass of tree languages whose linear notations can be accepted by deterministic pushdown automata (PDAs). Thus, the standard (string) PDA is another suitable model of computation for processing regular tree languages in a linear notation. For example, algorithms processing XML with the use of PDAs have been investigated [10, 17].

Regular tree expressions (RTEs) are a natural formalism for the description of regular tree languages [5]. They are analogous to regular (string) expressions. It is well known that regular (string) expressions describe regular languages and can be converted to finite automata. In the case of trees, RTEs can be converted to corresponding finite tree automata.

Finite automata and regular (string) expressions are well-studied [8, 16]. A string language membership problem is a decision problem. Given a regular (string) expression E and a string w, decide whether w is in the language described by the regular (string) expression E. This problem can be decided by converting the expression to an equivalent finite automaton and running the automaton on the input word w.

Many algorithms deal with a problem of converting regular (string) expressions to finite automata in the string domain. Three algorithms by Brzozowski [4], Thompson [18] and Glushkov [7] (also known as a position automaton) are the basic ones. Antimirov’s partial derivatives method [3] (which can be seen as a non-deterministic extension of Brzozowski’s algorithm) must be also mentioned. Conversions by Glushkov’s and Antimirov’s can be done in polynomial time w.r.t. the number of occurrences of symbols in the regular expression.

The language membership problem for trees and RTEs is analogous: Given a regular tree expression E and a tree t, decide whether t is in the language described by the regular tree expression E. As in the string case, one can create a finite tree automaton (or a PDA) equivalent to the RTE E and let the automaton run on (linearised) tree t.

Algorithms for the conversion of RTEs to finite tree automata are inspired by the mentioned algorithms from the string domain. Antimirov’s and Glushkov’s algorithms were adapted to regular tree expressions by Kuske and Meinecke [11] and also later by Laugerotte et al. [12]. The finite tree automaton is constructed in polynomial time w.r.t. the size of the RTE in both adaptations. Thompson’s algorithm was an inspiration for Polách [14], where RTEs are converted to PDAs.

This paper presents a new approach for the conversion of RTEs to PDAs. The presented algorithm was inspired by the Glushkov’s algorithm [7] for regular (string) expressions. To create the equivalent PDA, the RTE is analysed similarly to Glushkov’s algorithm. Resulting PDA accepting linearised trees described by the RTE is constructed in quadratic time w.r.t. the size of the RTE. The constructed automaton is a real-time height-deterministic PDA and therefore it can be always determinised [15].

The paper is organised as follows: Basic definitions are given in Section 2. The conversion algorithm producing the PDA is presented in Section 3. Section 4 discusses complexity improvements to the algorithm and to the size of the constructed PDA. Finally, the results are summarised in the conclusion.

## 2 Basic Definitions

### 2.1 Trees

A labelled, ordered and ranked tree over a ranked alphabet \( \mathcal{A} \) can be defined based on the concepts from graph theory [1].

A ranked alphabet \( \mathcal{A} \) is a finite nonempty set of symbols. Each symbol \( a \) is assigned with a non-negative integer arity denoted by \( \text{arity}(a) \). \( \mathcal{A}_n \) denotes the set of symbols from \( \mathcal{A} \) with arity \( n \). The set \( \mathcal{A}_0 \) is nonempty. Notation \( a2 \) denotes symbol \( a \) with \( \text{arity}(a) = 2 \).
A directed ordered graph $G$ is a pair $(N, R)$ where $N$ is a set of nodes and $R$ is a set of ordered lists of edges. Elements from $R$ are in the form $((f, g_1), (f, g_2), \ldots, (f, g_n))$, where $f, g_1, g_2, \ldots, g_n \in N$, $n \geq 0$. Such element indicates that there are $n$ edges leaving $f$ with the first edge 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 from $f_i$ to $f_{i+1}$ for each $0 \leq i < n$. A cycle is a path where $f_0 = f_n$.

An in-degree of a node $f \in N$ is the number of distinct pairs $(g, f), g \in N$ in elements of $R$. An out-degree of $f \in N$ is the number of distinct pairs $(f, g), g \in N$ in elements of $R$. A node with the out-degree 0 is a leaf.

An ordered directed acyclic graph (DAG) is an ordered directed graph with no cycle. A rooted DAG is a DAG with a special node $r \in N$ called the root. The in-degree of $r$ is 0, in-degree of every other node is 1 and there is just one path from the root $r$ to every $f \in N$, $f \neq r$. A labelled ranked DAG is a DAG where every node is labelled by a symbol $a \in A$ and the out-degree of a node $a \in A$ equals to arity($a$). A directed, ordered, rooted, labelled and ranked tree is rooted, labelled and ranked DAG. All trees in this paper are considered to be directed, ordered, rooted, labelled and ranked.

The postfix notation of a tree $t$ denoted by $post(t)$ is defined recursively:

1. $post(t) = root(t)$ if $root(t)$ is also a leaf,
2. $post(t) = post(c_1) \cdot post(c_2) \cdots post(c_n) \cdot root(t)$, $c_i$ are children of $root(t)$.

The postfix notation of a tree language $L$ is defined as $post(L) = \{post(t) : t \in L\}$. A postfix notation of any subtree of $t$ is a substring of $post(t)$. However, not every substring of a postfix notation of a tree is a postfix notation of its subtree [6].

**Example 1.** Let $t$ from Figure 1 be a directed, rooted, labelled, ranked and ordered tree with labels from ranked alphabet $A = \{a2, b0\}$. The root of $t$ is a node $a2$ with an ordered 2-tuple of children $(a2, b0)$. Postfix notation of $t$ is $post(t) = b0 b0 a2 b0 b0 a2 a2 b0 a2$.

### 2.2 Regular Tree Expressions

Regular tree expressions (RTEs) are defined using a substitution operation as in [5]. The definition of the RTE is similar to the definition of the regular (string) expression.

RTEs are defined over two alphabets, $\mathcal{F}$ and $\mathcal{K}$. $\mathcal{F}$ is a ranked alphabet of symbols, $\mathcal{K}$ is a set of constants (symbols with arity 0), $\mathcal{K} = \{\Box_1, \Box_2, \ldots, \Box_n\}$, $n \geq 0$, $\mathcal{F} \cap \mathcal{K} = \emptyset$. This alphabet is used to indicate the position where substitution operations take place.

Firstly, the substitution, i.e. replacing occurrences of $\Box_i$ by trees from a tree language $L_j$, is defined. Let $\mathcal{K} = \{\Box_1, \ldots, \Box_n\}$ and $t$ be a tree over $\mathcal{F} \cup \mathcal{K}$, and let $L_1, \ldots, L_n$ be tree languages. Then the tree substitution of $\Box_1, \ldots, \Box_n$ by $L_1, \ldots, L_n$ in $t$ denoted by $t[\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n]$ is the tree language defined by the following identities:
Construction of a PDA Accepting a Postfix Notation of Trees Given by a RTE

A sample regular tree expression (RTE).

Example 2. Figure 2 is a RTE and \( \text{F} \) of occurrences. In LISP:

- Figure 2 Examples of RTEs.

- The operation _alternation_ of \( L_1 \) and \( L_2 \) is denoted by \( L_1 \uplus L_2 \). The result is a set of trees obtained by the union of regular tree languages \( L_1 \) and \( L_2 \), i.e., \( L_1 \uplus L_2 \).

The operation _concatenation_ of \( L_2 \) to \( L_1 \) through \( \Box \), denoted by \( L_1 \Box L_2 \), is the set of trees obtained by substituting the occurrence of \( \Box \) in trees of \( L_1 \) by trees of \( L_2 \), i.e., \( \bigcup_{t \in L_1} t \{ \Box \leftarrow t \} \).

Given a tree language \( L \) over \( \mathcal{F} \cup \mathcal{K} \) and \( \Box \in \mathcal{K} \), the sequence \( L^n \Box \) is defined by the equalities \( L^0 \Box = \{ \Box \} \) and \( L^{n+1} \Box = L \uplus L^n \Box \). The operation _closure_ is defined as \( L^* \Box = \bigcup_{n=0}^{\infty} L^n \Box \).

The RTE over alphabets \( \mathcal{F} \) and \( \mathcal{K} \) is defined as follows:

- The empty set (\( \emptyset \)) and a constant \( \{a\} \in \mathcal{F}_0 \uplus \mathcal{K} \) are RTEs,
- if \( E_1, E_2, \ldots, E_n \) are RTEs and \( \Box \in \mathcal{K} \), then: \( E_1 + E_2 \) is a RTE, \( E_1 \Box E_2 \) is a RTE, \( E_1^* \Box \) is a RTE and \( a(E_1, \ldots, E_n) \) is a RTE if \( a \in F_n \) and \( \text{arity}(n) > 0 \).

RTE \( E \) represents a language denoted by \( L(E) \) and defined by the following equalities:

\[
\begin{align*}
L(\emptyset) & = \emptyset, \\
L(\{a\}) & = \{a\} \text{ for } a \in F_0 \uplus K, \\
L(f(E_1, \ldots, E_n)) & = \{ f(s_1, \ldots, s_n) \mid s_1 \in L(E_1), \ldots, s_n \in L(E_n) \}.
\end{align*}
\]

The size of the RTE \( E \) (denoted by \( |E| \)) is the size of the syntax tree of \( E \). The number of occurrences of symbols from \( \mathcal{F} \) and \( \mathcal{K} \) in the RTE \( E \) is denoted by \( ||F_E|| \) and \( ||K_E|| \), respectively.

**Example 2.** Let \( \mathcal{F} = \{ \text{nil}, \text{cons}, \text{int} \} \) where \( \text{arity}(\text{cons}) = 2 \) and other symbols have arity 0. Let \( \mathcal{K} = \{ \Box, \Box_2 \} \). Then the RTE from Figure 2b denotes the language of lists of integers in LISP: \( \{ \text{nil}, \text{cons}(\text{int}, \text{nil}), \text{cons}(\text{int}, \text{cons}(\text{int}, \text{nil})), \ldots \} \)

### 2.3 Pushdown Automata

Notions are used similarly as they are defined in [8].
A nondeterministic pushdown automaton (PDA) is a seven-tuple \((Q, \Sigma, \Gamma, \delta, q_0, \bot, F)\) where \(Q\) is a finite set of states, \(\Sigma\) is a finite set of input symbols (input alphabet), \(\Gamma\) is a finite set of pushdown store symbols (pushdown store alphabet), \(\delta\) is a mapping from \(Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*\) into a set of finite subsets of \(Q \times \Gamma^*\), \(q_0 \in Q\) is the initial state, \(\bot \in \Gamma\) is the initial pushdown store symbol and \(F \subseteq Q\) is a set of final states.

Triplet \((q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*\) is a configuration of a PDA. The initial configuration is \((q_0, w, \bot), w \in \Sigma^*\). Relation \((q, aw, \beta \alpha) \Gamma_M (p, w, \beta \gamma) \in (Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Sigma^* \times \Gamma^*)\) is a transition of a PDA \(M\) if \((p, \gamma) \in \delta(q, a, \alpha)\). \(\Gamma_M^k\) denotes the \(k\)-th power, \(\Gamma_M^0\) is the transitive closure and \(\Gamma_M^+\) is the transitive and reflexive closure. In strings representing the pushdown store in this paper, the top of the pushdown store is situated on the right.

A language accepted by PDA \(M\) can be defined in two distinct ways. PDA can accept
1. either by final states, then \(L(M) = \{w : w \in \Sigma^* \exists \gamma \in \Gamma^* \exists f \in F, (q_0, w, \bot) \vdash^+ (f, \varepsilon, \gamma)\}\)
2. or by an empty pushdown store, then \(L(M) = \{w : w \in \Sigma^* \exists q \in Q, (q_0, w, \bot) \vdash^+ (q, \varepsilon, \varepsilon)\}\)
   and \(F = \emptyset\).

A PDA is deterministic if the following conditions hold:

1. \(|\delta(q, a, \gamma)| \leq 1, \forall q \in Q, \forall a \in (\Sigma \cup \{\varepsilon\}), \forall \gamma \in \Gamma^*\),
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 \(\alpha\) (i.e., \(\alpha \gamma \neq \beta, \alpha \neq \beta \gamma, \gamma \in \Gamma^*\)),
3. If \(\delta(q, a, \alpha) \neq \emptyset, \delta(q, \varepsilon, \beta) \neq \emptyset\), then \(\alpha\) is not a prefix of \(\beta\) and \(\beta\) is not a prefix of \(\alpha\) (i.e., \(\alpha \gamma \neq \beta, \alpha \neq \beta \gamma, \gamma \in \Gamma^*\)).

The class of languages accepted by nondeterministic PDAs is exactly the class of context-free languages. Deterministic PDAs accepts deterministic context-free languages. This class is a proper subset of context-free languages.

A height-deterministic PDA is such PDA which on all of its runs on input word \(w \in \Sigma^*\) leads to the same pushdown store height. Height-deterministic PDAs are a generalization of visibly PDAs [13, 15, 2]. A real-time height-deterministic PDA is such PDA that is height-deterministic and without \(\varepsilon\)-transitions. This class of PDAs is determinisable [13, 15].

### 3 Converting RTE to PDA

In this section, a new method of creating a real-time height-deterministic PDA from RTE is proposed. The constructed PDA accepts postfix ranked linear notation of trees.

#### 3.1 Analysing RTE

To analyse the structure of the expression, the RTE has to be preprocessed similarly to Glushkov’s algorithm. Firstly, every occurrence of symbol from \(\mathcal{F}\) alphabet of the RTE is subscripted with an unique symbol. Subscripted RTE \(E\) is denoted as \(E’\).

Functions First, Follow and Pos are defined to analyse the RTE \(E’\). Function Pos returns a set of occurrences of symbols from \(\mathcal{F}\) alphabet of \(E’\). Function First computes a set of symbols that can be a root of a tree described by \(E’\). Function Follow returns tuples of children of a given symbol. Unlike strings, a symbol can be followed by more than a single symbol. The size of the children tuple is defined by the arity of the symbol.
Definition 3. Based on the definition of RTEs, the function First is defined recursively:

\[
\text{First}(\emptyset) = \emptyset \\
\text{First}(a(E_1, E_2, \ldots, E_n)) = \{a\} \\
\text{First}(E_1 + E_2) = \text{First}(E_1) \cup \text{First}(E_2) \\
\text{First}(E_1 \cdot □ E_2) = \begin{cases} 
\text{First}(E_1) & \text{if } □ \notin \text{First}(E_1) \\
(\text{First}(E_1) \setminus \{□\}) \cup \text{First}(E_2) & \text{if } □ \in \text{First}(E_1) 
\end{cases} \\
\text{First}(E^* □) = \{□\} \cup \text{First}(E)
\]

Theorem 4. The function First(E') returns the set of symbols that can be the root of any tree described by an RTE E.

Proof. The proof is done by induction on the structure of the RTE. The basis: If E = a(E_1, E_2, \ldots, E_n), a ∈ F ∪ K, n ≥ 0: Only a can be a root. If E = ∅: There is no root. Now, assume that the theorem holds for any E_1 and E_2. If E = E_1 + E_2: This operator unifies two sets of trees. Therefore the roots of trees from L(E) are either from First(E_1) or First(E_2). If E = E_1 □: The roots can be only elements from First(E_1) or the substitution symbol □. If E = E_1 □ E_2: Initially, suppose □ \notin First(E_1). Then the root must be from First(E_1). If □ ∈ First(E_1), then □ gets substituted by the roots of the trees from E_2.

The function Follow returns a set of tuples of symbols which can be direct descendants (children) of a symbol a ∈ F. The computation of the function is defined using Algorithm 1. The algorithm recursively traverses the syntax tree of the RTE and maintains a substitution map. The map contains roots of all possible trees that can be substituted for each □ ∈ K. If □ occurs as a child of the symbol a when computing Follow(E', a), it gets substituted by elements of a substitution map for a given □.

It is possible that □_2 ∈ K is present in the subMap[□_1] of any node. Then it is required to include the contents of mapping for key □_2 into □_1 set of that node. Also, if □ ∈ subMap[□] then □ element can be discarded from the set as it brings no new information.

For the purpose of proving the correctness of the computation, the algorithm can be split in a two pass algorithm. In the first pass, the substitution mapping for each node is computed. In the second pass, the computation of Follow can use the computed mapping.

Theorem 5. Algorithm 1 computes a substitution mapping of every node of the RTE.

Proof. If the substitution operation takes place (in concatenation and iteration nodes), it alters the substitution map. The changes in substitution mapping come from the definitions of RTEs. Case E_1 □ E_2: Roots from trees described by E_2 may appear in the place of □ symbols in E_1. Therefore the mapping for the □ symbol in E_1 is replaced. The substitutions for □ symbols in E_2 are determined by the same mapping as in the parent node. Case E_1 □ E_2: Symbol □ is to be replaced by roots of E_1 (this implements the actual iteration) and the iteration is terminated by concatenating a tree from the right operand of the closest substitution or iteration node. In other cases, the existing mapping is simply passed to children as no substitution happens.

Theorem 6. Function Follow(E', a) (defined by Algorithm 1) correctly returns a set of tuples representing all possible tuples of direct children of a node a.

Proof. The proof by induction is straightforward with the use of the previous theorem.
Algorithm 1: Computation of Follow($E', a$) in a single pass.

```latex
Function Follow($E$, $a$) 
| return FollowRec($E$, $a$, NewMap())

Function FollowRec($E$, $a$, subMap)
| switch $E$ do
| case $E_1 + E_2$ do
| return FollowRec($E_1$, $a$, subMap) $\cup$ FollowRec($E_2$, $a$, subMap)
| case $E_1 \cdot □ E_2$ do
| $subMapL \leftarrow subMap$ /* copy map */
| $subMapL[□] \leftarrow \text{First}(E_2)$ /* replace mapping for □ */
| return FollowRec($E_1$, $a$, subMap) $\cup$ FollowRec($E_2$, $a$, subMap)
| case $f(E_1, E_2, \ldots, E_n)$ do
| if $a = f$ then return ReplaceConstants($subMap$, $E_1, E_2, \ldots, E_n$)
| else return $\bigcup_{i=1}^{n} \text{FollowRec}(E_i, a, subMap)$
| case $∅$ do
| return $∅$

Function ReplaceConstants($subMap$, $E_1, E_2, \ldots, E_n$)
| $lst \leftarrow \text{NewList}()$
| for $E_i$ in $E_1, E_2, \ldots, E_n$ do
| if $E_i \in K$ then $lst \leftarrow \text{Append}(lst, subMap[c])$ /* child is a □ */
| else $lst \leftarrow \text{Append}(lst, \text{First}(E_i))$
| return CartesianProduct($lst$)
```

Example 7. Let $E$ be a RTE from Figure 2a. First($E'$) = \{\[b0_2, a2_1, a2_3\]\}. The results of the function First and the substitution map for individual nodes are illustrated in Figure 3. Follow($E', a2_1$) = \{(a2_1, a2_1), (a2_1, a2_1), (a2_1, b0_2), (a2_1, a2_1), (a2_1, a2_1), (a2_1, b0_2), (b0_2, a2_1), (b0_2, a2_1), (b0_2, b0_2)\}. Follow($E', a2_3$) = \{(b0_4, b0_3)\}. Follow of leaves is $∅$.

3.2 Pushdown Automaton Construction

In the previous section, it was shown how to compute First and Follow sets. The First set determines what symbols are the last to be read in the postfix notation. The Follow sets store the information about the direct children of a node. This information is used to create transitions of the two state PDA that accepts by final state. The automaton reads a linear postfix notation of a tree with an end-of-string marker $\vdash$ appended to the end of the input. Technical helper functions $\varphi$ and $\sigma$ are presented first.

Definition 8. Function $\varphi$ maps an element (tuple) of Follow($E'$, $a$) to a string of pushdown store symbols. The resulting string is $ε$ if the size of the tuple is zero. Mapping $\sigma$ strips the unique index from the subscripted symbol.

Example 9. Let $E'$ be a subscripted RTE and let $f = (a2_1, b2_2, c0_3)$ be a follow tuple of some node. Then $\varphi(f) = a2_1 b2_2 c0_3$. Also $\sigma(a2_1) = a2$. 
Construction of a PDA Accepting a Postfix Notation of Trees Given by a RTE

First = \{a_2, b_0, a_2\} \quad \text{First = \{b_0\}}
\{\square \rightarrow \}\quad \{\square \rightarrow \}\n
First = \{a_2, b_0, a_2\} \quad \text{First = \{a_2\}}
\{\square \rightarrow \{b_0\}\} \quad \{\square \rightarrow \{b_0\}\}

First = \{a_2, b_0, \square\} \quad \text{First = \{b_0\}}
\{\square \rightarrow \{a_2\}\} \quad \{\square \rightarrow \{b_0\}\}

First = \{a_2, b_0\} \quad \text{First = \{b_0\}}
\{\square \rightarrow \{a_2, a_2\}\} \quad \{\square \rightarrow \{b_0\}\}

First = \{a_2\} \quad \text{First = \{b_0\}}
\{\square \rightarrow \{a_2, a_2, b_0\}\} \quad \{\square \rightarrow \{a_2, a_2, b_0\}\}

First = \{\square\} \quad \text{First = \{\square\}}
\{\square \rightarrow \{a_2, a_2, b_0\}\} \quad \{\square \rightarrow \{a_2, a_2, b_0\}\}

\textbf{Algorithm 2:} PDA accepting linearised trees described by an RTE \(E\).

\textbf{input :} RTE \(E\).
\textbf{output :} PDA \(A\) such that \(L(A) = post(L(E))\)

Create PDA with the following properties:
- Set of states is equal to \(\{q, f\}\),
- alphabet is equal to the \(F\) alphabet of \(E\), and \(-\) symbol,
- pushdown store alphabet is equal to the symbols \(\text{Pos}(E') \cup \{\perp\}\),
- mapping \(\delta\) can be created by these rules:
  - \(\forall a_i \in \text{Pos}(E'), \sigma(a_i) \in F_0,\) add transition \(\delta(q, \sigma(a_i), \varepsilon) = ((q, a_i))\),
  - \(\forall a_i \in \text{Pos}(E'), \sigma(a_i) \not\in F_0, \forall f \in \text{Follow}(E', a_i),\) add \(\delta(q, \sigma(a_i), \varphi(f)) = ((q, a_i))\),
  - \(\forall a_i \in \text{First}(E')\) add transition \(\delta(q, \perp, \perp a_i) = \{(f, \varepsilon)\}\).

Resulting PDA is \(A = ((q, f), F \cup \{\perp\}, \text{Pos}(E') \cup \{\perp\}, \delta, \perp, q, \{f\})\). Automaton accepts by the final state. The top of the pushdown store is on the right.

Roots of subtrees are stored on the pushdown store to keep track of which subtrees have been read so far. When the root of a subtree is read, its children have to be on the top of the pushdown store. They are replaced by a pushdown store symbol corresponding to the read symbol. PDA recognising postfix notations of trees described by RTE \(E\) (\(post(L(E))\)) is constructed by Algorithm 2.

\textbf{Example 10.} This example expands on Example 7. The PDA \(A = ((q, f), \{a_2, b_0, \perp\}, \{\perp, a_2, b_0, a_2, a_2, b_0, a_2, b_0, \perp\}, \delta, \perp, q, \{f\})\) is constructed by Algorithm 2. \(\delta\) is defined as follows:

\[
\begin{align*}
\delta(q, a_2, a_2, a_2) &= \{(q, a_2)\} & \delta(q, a_2, b_0, a_2) &= \{(q, a_2)\} \\
\delta(q, a_2, b_0, b_2) &= \{(q, a_2)\} & \delta(q, a_2, b_0, b_2) &= \{(q, a_2)\} \\
\delta(q, a_2, a_2, a_2) &= \{(q, a_2)\} & \delta(q, a_2, b_0, a_2) &= \{(q, a_2)\} \\
\delta(q, a_2, b_0, b_2) &= \{(q, a_2)\} & \delta(q, a_2, b_0, b_2) &= \{(q, a_2)\} \\
\delta(q, \perp, \perp a_2) &= \{(f, \varepsilon)\} & \delta(q, \perp, \perp a_2) &= \{(f, \varepsilon)\} \\
\delta(q, \perp, \perp a_2) &= \{(f, \varepsilon)\}
\end{align*}
\]

The RTE from Figure 2a describes, for instance, the tree from Figure 1. This tree in its postfix notation (with \(-\) symbol appended) is accepted by the automaton.

\textbf{Theorem 11.} Algorithm 2 creates PDA \(A\) such that \(L(A) = post(L(E))\).

\textbf{Proof.} The proof consists of two parts: \(post(L(E)) \subseteq L(A)\) and \(L(A) \subseteq post(L(E))\).
Case \( post(\{E\}) \subseteq L(A) \): Proof comes directly from the proof of the Follow and First functions. The functions analyse all possible combinations of parent-children relations. The relations are used in the transition function of the PDA. When an input tree (except \( \hat{\imath} \) symbol) is read, the automaton can continue only if the pushdown store content equals to the string \( \downarrow f (f \in \text{Follow}(E')) \) set to ensure that whole tree was read.

Case \( L(A) \subseteq post(\{E\}) \): If there is a word from \( L(A) \) that is not in \( post(\{E\}) \) then either the computation of First or Follow functions were wrong or the transitions created from Follow sets would allow the automaton to accept something more. The functions First and Follow are proved to be correct. ▶

**Theorem 12.** Algorithm 2 creates a real-time height-deterministic PDA.

**Proof.** Transitions PDA always pop arity(\( a \)) symbols from the pushdown store and push one symbol when reading symbol \( a \). Reading symbol \( \hat{\imath} \) pops two symbols and pushes none. The pushdown store height is predetermined and same for all nondeterministic computations of the PDA on any string. This fulfills the conditions of height-determinism. The PDA never reads \( \varepsilon \), therefore it is also real-time [13, 15]. ▶

The PDA has two properties worth mentioning: The function ReplaceConstants from Algorithm 1 has an exponential output with the number of node’s children that are from \( K \) and the size of \( \text{subMap}[\( \square \)] \) for given node. As every element from the Follow set results in one transition, the PDA has an exponential amount of transitions. Also, Theorem 12 shows that the PDA is determinisable because the PDA is real-time height-deterministic [13, 15].

**Example 13.** RTE \( E' \) (Figure 4a) has the following properties: First(\( E' \)) = \{a21, b02, c03\}, Follow(\( E', a21 \)) = \{a21, a21\}, Follow(\( E', b02 \)) = \{a21, b02, c03\}, Follow(\( E', c03 \)) = \{a21, a21\} and Follow(\( E', b02 \)) = \{a21, b02\}. The equivalent PDA is illustrated in Figure 4b.

## 4 Reducing the Number of Transitions

The PDA created by Algorithm 2 has an exponential number of transitions. The transition function \( \delta \) enumerated all possible tuples of children for every node in the tree.

The idea behind the improvement is to make a better use of the pushdown store. New pushdown store symbols representing all possible symbols that can appear in the place of a
Construction of a PDA Accepting a Postfix Notation of Trees Given by a RTE

Algorithm 3: Improved PDA accepting linearised trees described by a RTE $E$.

\begin{algorithm}
\textbf{input} : RTE $E$.
\textbf{output} : PDA $A$ such that $L(A) = post(L(E))$

Create PDA with following properties:
\begin{itemize}
  \item Set of states is equal to \{q, f\},
  \item input alphabet is $\mathcal{F} \cup \{\dagger\}$,
  \item pushdown store alphabet consists of all sets that appear in substitution mapping in $\square \in \mathcal{K}$ nodes and singletons consisting of indexed occurrences of symbols from $\mathcal{F}$,
  \item transitions ($\delta$) are created by these rules:
    \begin{enumerate}
      \item for all symbols $a_i \in \mathcal{F}$ add transition $\delta(q, \sigma(a_i), \varphi(\text{Follow}(E, a_i))) = \{(q, \{a_i\})\}$,
      \item for all nodes $\square_i$ labelled with a $\square \in \mathcal{K}$, for all symbols $a_i \in \text{subMap}_{\square_i}[\square]$ add $\delta(q, \sigma(a_i), \varphi(\text{Follow}(E, a_i))) = \{(q, \text{subMap}_{\square_i}[\square])\}$,
      \item for all symbols $a_i \in \text{First}(E')$ add transition $\delta(q, \dagger, \bot \{a_i\}) = \{(f, \varepsilon)\}$.
    \end{enumerate}
\end{itemize}

Resulting PDA is $A = (\{q, f\}, \mathcal{F} \cup \{\dagger\}, \{\{a_i\} \mid a_i \in \mathcal{F}\} \cup \{\text{subMap}_{\square_i}[\square] \mid \text{for all nodes } \square_i \text{ labelled with a } \square \in \mathcal{K}\}, \delta, q, \{f\}$. Automaton accepts by the final state.

The top of the pushdown store is on the right.

symbol $\square \in \mathcal{K}$ are introduced. These symbols effectively represent the complete substitution mapping. For every occurrence of the symbol $\square \in \mathcal{K}$ the substitution mapping set for this occurrence is to be added as a new pushdown store symbol.

The only difference in the analysis of the RTE is the Follow algorithm. On line 24, the computation is altered by removing the computation of Cartesian product and returning the list $\text{lst}$ instead. This excludes the need for computing the Cartesian product. Furthermore, every symbol of $\mathcal{F}$ alphabet is now followed by exactly one tuple.

The ideas from previous paragraphs are applied in the Algorithm 3. The algorithm constructs an improved PDA which has an asymptotically lower amount of transitions.

\begin{definition}
Let $\text{subMap}_{\triangle}[\square]$ return the substitution mapping for symbol $\square \in \mathcal{K}$ inside the $\triangle$ node of the syntax tree.
\end{definition}

\begin{theorem}
Algorithm 3 creates a real-time height-deterministic PDA.
\end{theorem}

\begin{proof}
Similar to the proof of Theorem 12.
\end{proof}

The automaton created by Algorithm 3 is determinisable.

\begin{theorem}
Algorithm 3 creates PDA equivalent to RTE $E$ in $O(|E|^2)$ time and the number of transitions of the PDA is $O(\|\mathcal{F}_E\| \|\mathcal{K}_E\|)$.
\end{theorem}

\begin{proof}
Overall time complexity can be determined from the efficient implementation of the algorithm. Computing and saving the First set takes $O(|E| \|\mathcal{F}_E\|)$ time. The substitution mapping can be computed in one traversal over the syntax tree of RTE. It is saved as a mapping from every occurrence of a node from $\mathcal{K}_E$ alphabet to the set of elements from $\mathcal{F}_E$. The Follow elements can be computed in the same traversal. This takes $O(|E|^2)$ time. Rules of type 1 and 3 are created in $O(\|\mathcal{F}_E\|)$ time from the Follow mapping and First set, respectively. While creating type 2 rules, for every $\mathcal{F}_E$ node it is required to iterate over the saved substitution mapping. Therefore, creating type 2 rules takes $O(\|\mathcal{F}_E\| \|\mathcal{K}_E\|)$ time.

The overall time complexity is $O(|E| \|\mathcal{F}_E\| + |E| \|\mathcal{F}_E\| \|\mathcal{K}_E\|) = O(|E|^2)$ as $\|\mathcal{K}_E\| \leq |E|$ and $\|\mathcal{F}_E\| \leq |E|$. The number of transitions is $O(\|\mathcal{F}_E\| \|\mathcal{K}_E\|)$ because there are $O(\|\mathcal{F}_E\|)$ transitions of types 1 and 3, and $O(\|\mathcal{F}_E\| \|\mathcal{K}_E\|)$ transitions of type 2.
\end{proof}
Figure 5 Sample RTE with First set and substitution mapping for important nodes.

Figure 6 PDA equivalent to a RTE from Figure 5 with transitions grouped by type. For readability, symbol $\triangleright$ stands for \( \{a_4, b_0, c_0, d_0, e_0\} \) and symbol $\blacktriangleleft$ stands for \( \{a_4, b_0, c_0, d_0, e_0\} \).

Example 17. RTE $E'$ from Figure 5 converted to equivalent PDA (Figure 6). First($E'$) = \( \{a_4, b_0, c_0, d_0, e_0, p_0, q_0, r_0\} \). Follow($E'$, $a_4$) = \( \{\{a_4, b_0, c_0, d_0, e_0\}, \{a_4, b_0, c_0, d_0, e_0\}, \{a_4, b_0, c_0, d_0, e_0\}, \{a_4, b_0, c_0, d_0, e_0\}, \{a_4, b_0, c_0, d_0, e_0\}, \{a_4, b_0, c_0, d_0, e_0, p_0, q_0, r_0\}\). Follow of other symbols (leaves) is $\emptyset$. Note that if the Follow was computed by Algorithm 1 then $|\text{Follow}(E', a_4)| = 1600$. 

5 Conclusion and Future Work

A new algorithm for the conversion of a RTE to a PDA has been described. The resulted PDA accepts all trees from the language described by the RTE in their linear postfix notation. Presented PDA belongs to the class of real-time height-deterministic PDAs, therefore it can always be determinised [15].

The presented algorithm creates the PDA in quadratic time w.r.t. to the size of input RTE’s syntax tree, i.e. in $O(|E|^2)$ time. The number of transitions in the PDA is $O(|F_E||K_E|)$.

There is also a number of interesting open problems. As the processing of the RTE is similar to processing the regular expression for the Glushkov’s algorithm, we hope to explore more similarities with this algorithm. Although searching thoroughly, we have not found
an algorithm for the reverse conversion (from a finite tree automaton or a PDA to a RTE). Finally, we would like to explore the tree pattern matching problem where the definition of a set of tree patterns is represented by RTEs.

References