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
© Springer-Verlag 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 $0, 1, 2, \dots, p$ 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 $t_1, \dots, t_p \in T(\mathcal{F})$, then $f(t_1, \dots, t_p) \in T(\mathcal{F})$.

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:

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

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

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 $Q = \{1, 2, 3\}$,

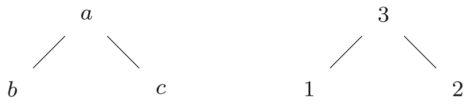

Fig. 1 Tree t (*left*) and the run of FTA $\mathcal{A}_1$ from Example 1 on tree t (*right*)

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

$$\begin{aligned} b &\to 1 \\ c &\to 2 \\ a(1,1) &\to 3 \\ a(1,2) &\to 3 \end{aligned}$$

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* $\Rightarrow_{\text{rm}}$ 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 $A \to \beta$. 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) $S \Rightarrow_{\mathrm{rm}}^{*} \alpha A w \Rightarrow_{\mathrm{rm}} \alpha\beta w$,
(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

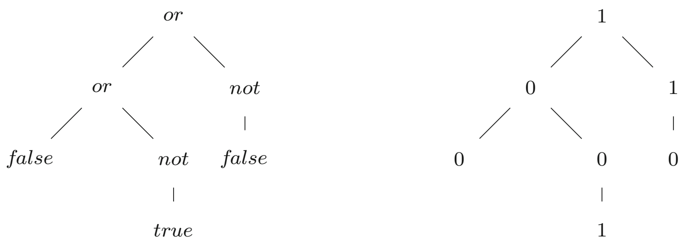

Fig. 2 Tree $t_1 \in L_2$ (*left*) and the run of FTA $\mathcal{A}_2$ from Example 2 on tree t_1 (*right*)

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 $P = \{S' \rightarrow S_q \dashv : q \in Q_f\} \cup \{S_q \rightarrow S_{q_1} S_{q_2}, \ldots, S_{q_n}\ a : a(q_1, q_2, \ldots, q_n) \rightarrow q \in \Delta\}$.

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 $\delta = \{\delta(q, \dashv, Z_0\alpha) = (q, \varepsilon) : \ S' \rightarrow \alpha \dashv \in P\} \cup \{\delta(q, a, \alpha) = (q, A) : A \rightarrow \alpha a \in P, A \neq S'\}$.

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:

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

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 $\text{post}(t) \dashv$. □

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 $S_q \to S_{q_1} S_{q_2}, \ldots, S_{q_n}\ a$, 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)