Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 899–902

ISBN 978-83-60810-22-4

Tree Indexing by Pushdown Automata and Subtree Repeats

Tomáš Flouri
Czech Technical University in Prague, Czech Republic,
Dept. of Theoretical Computer Science

Costas S. Iliopoulos
King's College London, UK, Dept. of Informatics &
Curtin University of Technology, Australia, DEBII

Jan Janoušek
Czech Technical University in Prague,
Czech Republic,
Dept. of Theoretical Computer Science

Bořivoj Melichar
Czech Technical University in Prague,
Czech Republic,
Dept. of Theoretical Computer Science

Solon P. Pissis
King's College London, UK,
Dept. of Informatics

***Abstract*—We consider the problem of finding all subtree repeats in a given unranked ordered tree. We show a new, elegant, and simple method, which is based on the construction of a tree indexing structure called the *subtree pushdown automaton*. We propose a solution for computing all subtree repeats from the deterministic subtree pushdown automaton constructed over the subject tree. The method we present is directly analogous to the relationship between string deterministic suffix automata and factor repeats in a given string.**

I. Introduction

TREES are one of the fundamental data structures used in Computer Science. Given a tree, finding beforehand unknown subtree repeats of the tree, and the positions of their occurrences, is a problem with many applications – data compression of trees, compiler code optimization, locating code clones in software packages, analysing various data tree structures, such as XML, and so on.

Periodicity in strings has been of interest since the beginning of the 20th century, and efficient methods for finding various kinds of repetitions and repeats in a string form an important part of well-researched stringology theory [1], [2], [3]. Some of these methods are based on principles of constructing and analysing string suffix trees or string suffix automata, which represent complete indexes of the suffixes of a string [4], [5], [6], [7], [8].

Trees can also be seen as strings in their linear notation. A linear notation of a tree can be obtained by the corresponding traversing of the tree. Moreover, every sequential algorithm on a tree traverses nodes of the tree in a sequential order, and follows a linear notation of the tree. In [9], the authors show that the deterministic pushdown automaton (PDA) is an appropriate model of computation for labelled ordered trees in postfix notation, and that the trees in postfix notation, acceptable by deterministic PDA, form a proper superclass of the class of regular tree languages [10], which are accepted by finite tree automata.

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

In this article, we present a new, elegant, and simple method for finding all subtree repeats in a given unranked ordered tree, and the positions of their occurrences. This problem can be defined as follows.

Problem 1: Find all subtree repeats within a given subject tree t.

The presented method is based on principles of constructing and analysing the deterministic *subtree pushdown automaton* (SPA), similarly as in the case of the above mentioned methods for finding repeats in strings. The SPA for ranked ordered trees was originally introduced in [11]. The construction of an SPA is based on the fact that the postfix bar notation of each subtree is a factor of the postfix bar notation of the tree. The underlying tree structure is processed by the use of the pushdown store. By analogy with the string factor automaton, the SPA represents a complete index of a given tree for all possible subtrees. Given a tree of size n, the advantages of the deterministic SPA are:

- Given an input subtree of size m, the deterministic SPA performs the search phase in time linear in m, and not depending on n.
- The number of subtrees of the tree is n and the total size of the deterministic SPA is linear in n.

We recall that the problem of tree indexing is defined as follows:

Problem 2: Construct an indexing structure over a subject tree t, so that one can efficiently query whether a given subtree p exists in t.

II. Preliminaries

A. Basic Definitions

An *alphabet* Σ is a finite, nonempty set of symbols. A *string* is a succession of zero or more symbols from an alphabet Σ. The string with zero symbols is denoted by ε. The set of all strings over the alphabet Σ is denoted by Σ^*. A string x of length m is represented by $x_1x_2\dots x_m$, where $x_i \in \Sigma$ for $1 \leq i \leq m$. The length of a string x is denoted by $|x|$. A string w is a *factor* of x if $x = uwv$ for $u, v \in \Sigma^*$, and is represented as $w = x_i \dots x_j$, $1 \leq i \leq j \leq |x|$.

The number of nodes of a tree t is denoted by $|t|$. The *postfix bar notation bar*(t) of a labeled ordered tree t is obtained by applying *Step* recursively, starting at the root of t.
Step: Let this application of *Step* be node v. List a bar. If v is a leaf, list v and halt. If v is an internal node having descendants $v_1, v_2, \ldots, v_r$, apply *Step* to $v_1, v_2, \ldots, v_r$ in that order and then list v. For simplicity, throughout the article we will refer to the postfix bar notation as bar notation.

An (extended) *nondeterministic pushdown automaton* is a seven-tuple $M = (Q, \mathcal{A}, G, \delta, q_0, Z_0, F)$, where Q is a finite set of *states*, $\mathcal{A}$ is the *input alphabet*, G is the *pushdown store alphabet*, δ is a mapping from $Q \times (\mathcal{A} \cup \{\varepsilon\}) \times G^*$ into a set of finite subsets of $Q \times G^*$, $q_0 \in Q$ is the initial state, $Z_0 \in G$ is the initial content of the pushdown store, and $F \subseteq Q$ is the set of final (accepting) states. The triplet $(q, w, x) \in Q \times \mathcal{A}^* \times G^*$ denotes the configuration of a PDA. In this article, we write the top of the pushdown store x on its left hand side. The initial configuration of a PDA is a triplet (q_0, w, Z_0) for the input string $w \in \mathcal{A}^*$. The relation $\vdash_M \subset (Q \times \mathcal{A}^* \times \Gamma^*) \times (Q \times \mathcal{A}^* \times \Gamma^*)$ is a transition of a PDA M. It holds that $(q, aw, \alpha\beta) \vdash_M (p, w, \gamma\beta)$ if $(p, \gamma) \in \delta(q, a, \alpha)$. For simplicity, in the rest of the text, we use the notation $p\alpha \underset{M}{\overset{a}{\longmapsto}} q\beta$ when referring to the transition $\delta_1(p, a, \alpha) = (q, \beta)$ of a PDA M. A PDA is *deterministic*, if:

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

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

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

In the rest of the text, we use the following labeling of edges when illustrating transition diagrams of various PDA: For each transition rule $\delta_1(p, a, \alpha) = (q, \beta)$ from the transition mapping δ of a PDA, we label its edge leading from state p to state q by the triplet of the form $a|\alpha \mapsto \beta$.

B. Properties of unranked ordered trees in bar notation

In this section, we present some basic properties of trees in their bar notation.

Lemma 3: Given a tree t and its bar notation *bar*(t), the bar notations of all subtrees of t are factors of *bar*(t).

However, not every factor of the bar notation of a tree represents a subtree.

Definition 4: Let $x = x_1 x_2 \ldots x_m$, $m \geq 1$, be a string over an alphabet Σ. Then, the *bar checksum* $bc(x) = \sum_{i=1}^{m} b(x_i)$, where

$$b(x_i) = \begin{cases} 1 & : \quad x_i = | \\ -1 & : \quad x_i \in \Sigma \end{cases}$$

Theorem 5: Let *bar*(t) and x be a tree t in bar notation and a factor of *bar*(t), respectively, over an alphabet Σ. Then, x is the bar notation of a subtree of t, if and only if $bc(x) = 0$, and $bc(y) < 0$ for each y, where $x = zy$, $y, z \in \Sigma^+$.

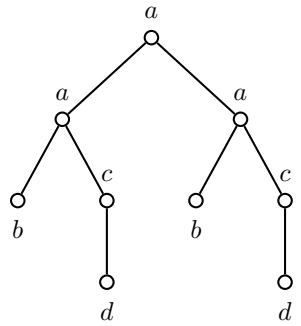

Fig. 2: An unranked ordered tree t having bar notation *bar*$(t) = |\,|\,|\,b\,|\,|\,d\,c\,a\,|\,|\,b\,|\,|\,d\,c\,a\,a$

Input: Tree $x = x_1 x_2 \ldots x_n$ over Σ
Output: Nondeterministic SPA M

1: $Q \leftarrow \bigcup_{i \leftarrow 0}^{n} \{i\}$
2: $B \leftarrow \bigcup_{i \leftarrow 1}^{n} \{\, i : x_i = | \,\}$
3: $C \leftarrow \bigcup_{i \leftarrow 1}^{n} \{\, i : x_i \neq | \,\}$
4: $T \leftarrow \bigcup\{(0, \varepsilon, |, S, i) : \forall i \in B\} \cup \bigcup\{(0, S, x_i, \varepsilon, i) : \forall i \in C\} \cup \bigcup\{(i-1, \varepsilon, |, S, i) : \forall i \in B\} \cup \bigcup\{(i-1, S, |, \varepsilon, i) : \forall i \in C\}$
5: $\delta(p, \alpha, x) \leftarrow \{(q, \beta) : \forall\ (p, \alpha, x, q, \beta) \in T\}$
6: $M \leftarrow (Q, \{S\}, \Sigma, \delta, 0, \varepsilon, \emptyset)$

Fig. 3: Construction of a nondeterministic SPA

III. Tree Indexing by Pushdown Automata

A. Subtree Pushdown Automaton

The SPA represents a complete index of some tree t. The language accepted by such automaton is the set of linearised notations of all subtrees of t – in this case bar notations – and is accepted by an empty pushdown store.

The construction of the nondeterministic SPA is similar to the construction of the classical nondeterministic string suffix automaton (see [2]) and is presented in Fig. 3. However, the transformation to its equivalent deterministic version differs substantially from that of the suffix automaton, as more constraints are placed due to the fact that not every factor of the linearised notation of a tree is a subtree (see Theorem 5).

Example 6: Consider the tree t illustrated in Fig. 2, having bar notation *bar*$(t) = |\,|\,|\,b\,|\,|\,d\,c\,a\,|\,|\,b\,|\,|\,d\,c\,a\,a$. The nondeterministic SPA constructed using the algorithm in Fig. 3, is $M = (\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18\}, \Sigma, \{S\}, \delta, 0, \varepsilon, \emptyset))$, illustrated in Fig. 1.

B. Construction of deterministic SPA

We are now in a position to present a simple method for constructing a deterministic SPA to solve Problem 2, by using a subset construction method, transforming the nondeterministic SPA to a deterministic one.

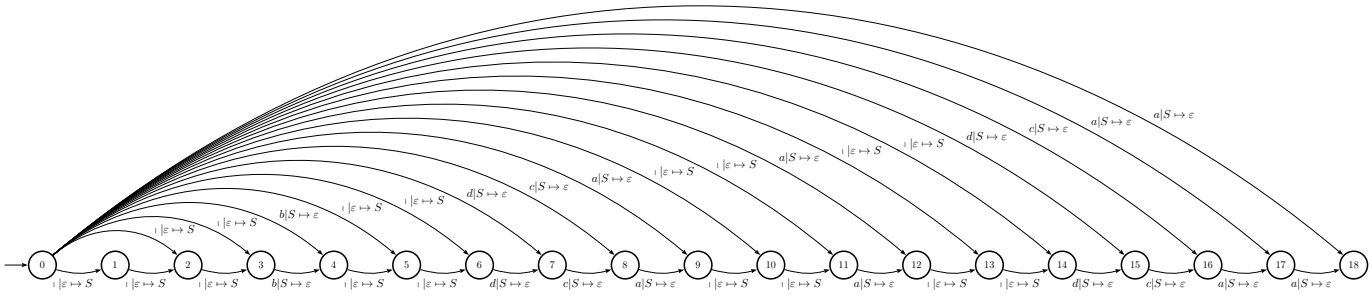

Fig. 1: Non-deterministic SPA constructed from Example 6

Input: Nondeterministic SPA $M = (Q, \Sigma, \{S\}, \delta, 0, \varepsilon, \emptyset)$
Output: Deterministic SPA $M' = (Q', \Sigma, \{S\}, \delta', \{0\}, \varepsilon, \emptyset)$

$Q' \leftarrow \{\{0\}\}$
$L \leftarrow \{\, (E, 1) : \delta(0, \varepsilon, |) = E \times \{S\} \,\}$
while not empty L **do**
  $(q, bc) \leftarrow$ DEQUEUE(L)
  $Q' \leftarrow Q' \cup \{q\}$
  for all $x \in \Sigma$ **do**
    $q_x \leftarrow \{\, E : \delta(p, S, x) = E \times \{\varepsilon\}, \forall p \in q \,\}$
    if $q_x \neq \emptyset$ **then**
      $Q' \leftarrow Q' \cup \{q_x\}$
      $\delta'(q, S, x) \leftarrow (q_x, \varepsilon)$
      if $bc > 1$ **then**
        ENQUEUE$(L, (q_x, bc - 1))$
      end if
    end if
  end for
  $q_| \leftarrow \{\, E : \delta(p, \varepsilon, x) = E \times \{S\}, \forall p \in q \,\}$
  if $q_| \neq \emptyset$ **then**
    $Q' \leftarrow Q' \cup \{q_|\}$
    $\delta'(q, \varepsilon, |) \leftarrow (q_|, S)$
    ENQUEUE$(L, (q_x, bc + 1))$
  end if
end while
$M' \leftarrow (Q', \Sigma, \{S\}, \delta', \{0\}, \varepsilon, \emptyset)$

Fig. 4: Transforming a nondeterministic SPA to an equivalent deterministic

Fig. 4 presents the algorithm for the transformation, which is based on the well-known technique of subset construction [12]. The nondeterministic SPA constructed from Alg. 3 is input–driven, and thus can be transformed to an equivalent deterministic SPA, which will serve as the indexing structure for the given subject tree.

Example 7: Consider the nondeterministic SPA constructed in Example 6. By applying the algorithm in Fig. 4, we obtain a new deterministic SPA with its transition diagram illustrated in Fig. 5.

Theorem 8: Given a tree t of size n, the deterministic SPA constructed from *bar*(t) is input-driven, has exactly one pushdown symbol, and consists of at most $4n + 1$ states.

IV. FINDING SUBTREE REPEATS

A. Definition of the problem

Problem 1 is to find all subtree repeats of a given tree t, along with the positions and the types of their occurrences. The positions and the types of the occurrences are summarised in a table called the *subtree repeats table*.

Definition 9: Let t be a tree over an alphabet Σ. A *subtree position set sps*(s, t), where s is a subtree of t, is the set *sps*$(s, t) = \{i : bar(t) = x\, bar(s)\, y,\ x, y \in (\Sigma \cup \{|\})^*, i = |x| + |bar(s)| + 1\}$.

Informally, the subtree position set for a subtree s contains the positions of the roots of all occurrences of the subtree s.

Example 10: Consider the tree t illustrated in Fig. 2. There are four subtree repeats t_1, t_2, t_3 and t_4 in t having bar notations *bar*$(t_1) = |\,|\,b\,|\,|\,d\,c\,a$, *bar*$(t_2) = |\,|\,d\,c$, *bar*$(t_3) = |\,d$ and *bar*$(t_4) = |\,b$. It holds that *sps*$(t_1, t) = \{9, 17\}$, *sps*$(t_2, t) = \{8, 16\}$, *sps*$(t_3, t) = \{7, 15\}$ and *sps*$(t_4, t) = \{4, 12\}$.

Definition 11: Let t be a tree over an alphabet Σ. Given a subtree s of t, the *list of subtree repeats lsr*(s, t) is a relation in *sps*$(s, t) \times \{F, S, G\}$ defined as follows:

- $(i, F) \in$ *lsr*(s, t) iff $bar(t) = x\, bar(s)\, y,\ i = |x| + |bar(s)|,\ x \neq x_1\, bar(s)\, x_2$,
- $(i, S) \in$ *lsr*(s, t) iff $bar(t) = x\, bar(s)\, y,\ i = |x| + |bar(s)|,\ x = x_1\, bar(s)$,
- $(i, G) \in$ *lsr*(s, t) iff $bar(t) = x\, bar(s)\, y,\ i = |x| + |bar(s)|,\ x = x_1\, bar(s)\, x_2$

where $x, y, x_1, x_2 \in \Sigma^*$.

In other words, the list of subtree repeats can be categorised in three types. F denotes that the subtree at the specific position is the *first* subtree in the list. S denotes a *square*, i.e. the specified subtree is a sibling of its previous subtree repeat, and thus their bar notations are consequtive factors in the bar notation of the subject tree. G stands for *gap*, and denotes that there exists a gap – another different subtree – between its previous repeat. In comparison with the types of repeats found in strings (see [2], [8]), subtree repeats are missing the type *overlapping*, as no two different occurrences of the same subtree can overlap.

Definition 12: Given a tree t, the *subtree repeats table SRT*(t) is the set of all triplets $(sps(s, t), bar(s), lsr(s, t))$, where s is a subtree with more than one occurrence in t.

Example 13: Consider the tree t illustrated in Fig. 2. The subtree repeats table *SRT*(t) is illustrated in Table I.

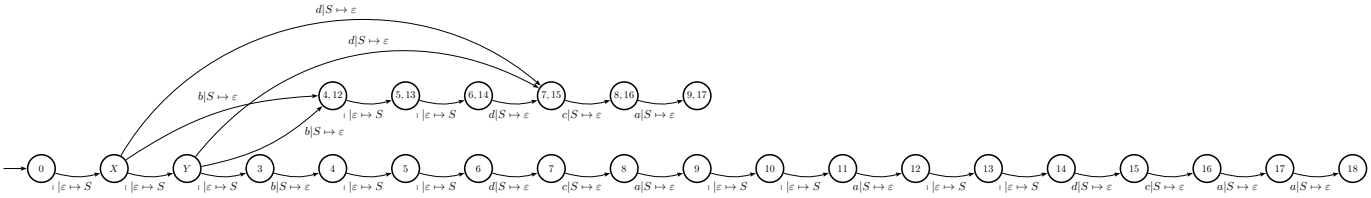

Fig. 5: Deterministic SPA from Example 7. Note that X denotes the state $\{1, 2, 35, 6, 10, 11, 13, 14\}$ and Y the state $\{2, 3, 6, 11, 14\}$

Input: A tree t in bar notation $bar(t) = x_1 x_2 \dots x_n$
Output: Subtree repeats table $SRT(t)$

1: Initialise $SRT(t) = \emptyset$
2: Construct a deterministic SPA $M = (Q, \Sigma, \{S\}, \delta, \{0\}, \varepsilon, \emptyset)$ using the algorithms in Fig. 3 and 4
3: ENQUEUE$(L, \{\{0\}, \varepsilon, 0\})$
4: **while notempty** L **do**
5: $(q, x, bc) \leftarrow$ DEQUEUE(L)
6: **for all** $y \in \Sigma$ **do**
7: **if** $\delta(q, S, y) \neq \emptyset$ **then**
8: ENQUEUE$(L, \{p, xy, bc - 1\} : \delta(q, S, y) = (p, \varepsilon))$
9: **end if**
10: **end for**
11: **if** $\delta(q, \varepsilon, |) \neq \emptyset$ **then**
12: ENQUEUE$(L, \{p, x|, bc + 1\} : \delta(q, \varepsilon, y) = (p, S))$
13: **end if**
14: **if** $bc = 0 \wedge |x| > 0 \wedge |q| > 1$ **then**
15: Let us denote that $q = \{r_1, r_2, \dots, r_{|q|}\}$ such that $r_i > r_{i-1}$
16: $lsr(x, t) \leftarrow \{(r_1, F)\}$
17: **for all** $r_i \in q, i > 1$ **do**
18: **if** $r_i - r_{i-1} = |x|$ **then**
19: $lsr(x, t) \leftarrow lsr(x, t) \cup \{(i, S)\}$
20: **else** $\{r_i - r_{i-1} > |x|\}$
21: $lsr(x, t) \leftarrow lsr(x, t) \cup \{(i, G)\}$
22: **end if**
23: **end for**
24: **end if**
25: $SRT(t) \leftarrow SRT(t) \cup \{\{i : (i, y) \in lsr(x, t)\}, x, lsr(x, t)\}$
26: **end while**

Fig. 6: Construction of the subtree repeats table for a tree t

$sps(s, t)$	$bar(s)$	List of subtree repeats
$9, 17$	$\|\|b\|\|d\,c\,a$	$(9, F), (17, S)$
$8, 16$	$\|\|d\,c$	$(8, F), (16, G)$
$7, 15$	$\|d$	$(7, F), (15, G)$
$4, 12$	$\|b$	$(4, F), (12, G)$

TABLE I: Subtree repeats table $SRT(t)$ from Example 13

B. Construction of the subtree repeats table

A well-known general property of the deterministic string suffix automaton constructed for a string x is that the state in which the deterministic string suffix automaton transits after reading a factor y, corresponds to the set of ending positions of all occurrences of the factor y in x [8].

The transitions of the deterministic SPA M constructed by the algorithm in Fig. 4 for some tree t are extensions of the transitions of the deterministic string suffix automaton [11], and therefore the same general property also holds for the SPA M: the state in which the SPA M transits after reading the postfix notation x of some subtree s, corresponds to the set of ending positions of all occurrences of x in $bar(t)$, which, in turn, corresponds to the positions of the root nodes of all occurrences of s in t.

The deterministic SPA M accepts bar notations of all subtrees of t by the empty pushdown store. The non-singleton subset of some state q having its bc set to 0, which is represented by the empty pushdown store, denotes the positions of the subtree read to transit from the initial state of M to state q. The subtree repeats table can therefore be constructed by traversing the deterministic SPA M and is described by the algorithm in Fig. 6.

REFERENCES

[1] M. Crochemore and W. Rytter, *Jewels of Stringology*. New Jersey: World Scientific, 1994.
[2] B. Melichar, J. Holub, and J. Polcar, "Text searching algorithms," Available on: http://stringology.org/athens/, 2005, release November 2005.
[3] B. Smyth, *Computing Patterns in Strings*. Essex, England: Addison-Wesley-Pearson Education Limited, 2003.
[4] A. Apostolico and F. P. Preparata, "Optimal off-line detection of repetitions in a string," *Theor. Comput. Sci.*, vol. 22, pp. 297–315, 1983.
[5] G. S. Brodal, R. B. Lyngsø, C. N. S. Pedersen, and J. Stoye, "Finding maximal pairs with bounded gap," in *CPM*, ser. Lecture Notes in Computer Science, M. Crochemore and M. Paterson, Eds., vol. 1645. Springer, 1999, pp. 134–149.
[6] M. Crochemore, "An optimal algorithm for computing the repetitions in a word," *Inf. Process. Lett.*, vol. 12, no. 5, pp. 244–250, 1981.
[7] M. G. Main and R. J. Lorentz, "An o(n log n) algorithm for finding all repetitions in a string," *J. Algorithms*, vol. 5, no. 3, pp. 422–432, 1984.
[8] B. Melichar, "Repetitions in text and finite automata," in *Proceedings of the Eindhoven FASTAR Days 2004*, L. Cleophas and B. Watson, Eds., TU Eindhoven, The Netherlands, 2004, pp. 1–46.
[9] J. Janoušek and B. Melichar, "On regular tree languages and deterministic pushdown automata," *Acta Inf.*, vol. 46, no. 7, pp. 533–547, 2009.
[10] F. Gecseg and M. Steinby, "Tree languages," in *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds. Springer–Verlag, Berlin, 1997, vol. 3 Beyond Words. Handbook of Formal Languages, pp. 1–68.
[11] J. Janoušek, "String suffix automata and subtree pushdown automata," in *Proceedings of the Prague Stringology Conference 2009*, J. Holub and J. Žďárek, Eds., Czech Technical University in Prague, Czech Republic, 2009, pp. 160–172, available on: http://www.stringology.org/event/2009.
[12] A. V. Aho and J. D. Ullman, *The theory of parsing, translation, and compiling*. Prentice-Hall Englewood Cliffs, N.J., 1972.