Subtree Oracle Pushdown Automata for Ranked and Unranked Ordered Trees

Martin Plicka, Jan Janoušek, Bořivoj Melichar
martin.plicka@fit.cvut.cz, jan.janousek@fit.cvut.cz, borivoj.melichar@fit.cvut.cz
Department of Theoretical Computer Science,
Faculty of Information Technology,
Czech Technical University in Prague,
Thákurova 9, 160 00 Prague 6,
Czech Republic

***Abstract*—Oracle modification of subtree pushdown automata for ranked and unranked ordered trees is presented. Subtree pushdown automata [1] represent a complete index of a tree for subtrees. Subtree oracle pushdown automata, as inspired by string factor oracle automaton [2], have the number of states equal to $n+1$, where n is the length of a corresponding linear notation of the tree. This makes the space complexity very low. By analogy with the string factor oracle automaton the subtree oracle automata can also accept some subtrees which are not present in the given subject tree. However, the number of such false positive matches is smaller than in the case of the string factor oracle automaton. The presented pushdown automata are input–driven and therefore they can be determinised.**

I. Introduction

TREES are one of the fundamental data structures used in Computer Science, e.g. they are used as intermediate forms in compilers or in the form of XML documents. Trees can also be seen as strings, for example in their prefix (also called preorder) or postfix (also called postorder) notation.

One of basic approaches to pattern matching uses data structures which are constructed for a given subject and represent its index. Examples of such data structures for a subject string can be suffix or factor automata [3], [4], [5]. The main advantage of this kind of deterministic finite automata are that they perform the search phase in time linear in m and not depending on n, where m and n are the length of the input pattern and of the subject string, respectively. However, the implementation of the string factor automaton requires a fairly large amount of memory space. Factor oracle automaton [2] represents a space reduced variant of the string factor automaton. The number of states of factor oracle automaton is equal to $n+1$, where n is the length of the subject string. In comparison with the string factor automaton, it accepts also some additional subsequences of the subject string. Despite this fact, it can be used for a fast and memory efficient indexing of strings and for backward oracle string matching, in which the factor oracle for a reversed input pattern is constructed and then it is used for matching the input pattern in a sliding window from right to left [2].

In [1], we have introduced Subtree PDA, a new kind of acyclic PDA for ordered trees, which represents an index of a subject tree for subtrees and is analogous to the string factor automaton and its properties. The construction of a subtree PDA is based on the fact that each subtree in a specific linear notation is a substring of the tree in the linear notation. The underlying tree structure is processed by the use of the pushdown store. In this paper, by analogy with the string processing, we present an oracle modification of the subtree PDA. The deterministic subtree oracle PDA has the number of states equal to $n+1$, where n is the length of a corresponding linear notation of the tree. It accepts all subtrees of the subject tree and further it may accept also certain subtrees which are not present in the subject tree. Despite this fact, it can be used for a fast and memory efficient indexing of trees, mainly for quick rejection of input patterns that are not subtrees of the subject tree, and for backward oracle subtree matching, which can be done by analogy with the string backward oracle matching algorithm [2]. We present subtree oracle PDAs for both ranked and unranked ordered trees. Although searching thoroughly, we have not found any other existing indexing structure for a tree with the abovementioned space complexity.

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.

II. Some basic notions and notations

Definition 1. *The prefix notation pref(t) of a tree t is defined in this way:*

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

For unranked trees, nodes have no arity so it is not possible to determine the number of node descendants from label. Instead, a bar notation defined bellow can be used.

Definition 2. *Let $]$ be the* prefix bar symbol, $] \notin \mathcal{A}$. Prefix bar notation $pref_{bar}(t)$ *of a tree t is defined as follows:*

1) $pref_{bar}(t) = a\,]$ *for tree t with a single node a.*

2) *$pref_{bar}(t) = r\,pref_{bar}(a_1), \ldots pref_{bar}(a_n)\,]$ for tree with root r and direct descendants $a_1 \ldots a_n$ of node r.*

Example 1. Consider a ranked alphabet $\mathcal{A} = \{b2, a2, b0, a0\}$. Consider a tree t_1 in prefix notation $pref(t_1) = b2\ b0\ a2\ a0\ a2\ a0\ a0$. Tree t_1 is illustrated in Fig. 1. □

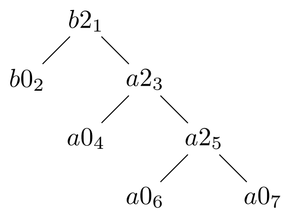


Fig. 1. Tree t_1 from Example 1

Assuming tree t_1 being unranked ($\mathcal{A} = \{a, b\}$), it can be written in prefix bar notation as
$pref_{bar}(t_1) = b\ b\ ]\ a\ a\ ]\ a\ a\ ]\ a\ ]\ ]\ ]\ ]$ □

A. Factor oracle automata in stringology

Given a string x, the deterministic factor automaton is defined as the minimal deterministic finite automaton accepting all factors of x. A factor oracle automaton can be constructed from the factor automaton. This construction is based on merging so-called *corresponding states* in a factor automaton together [5].

Definition 3. *Let M be a factor automaton for string x and q_1, q_2 be different states of M. Let there exist two sequences of transitions in M: $(q_0, x_1) \vdash^* (q_1, \varepsilon)$, and $(q_0, x_2) \vdash^* (q_2, \varepsilon)$.*

If x_1 is a suffix of x_2 and x_2 is a prefix of x then q_1 and q_2 are corresponding states.

The factor oracle automaton can be constructed by merging the corresponding states.

Example 2. We construct factor oracle automaton for the prefix notation $pref(t_1) = b2\ b0\ a2\ a0\ a2\ a0\ a0$ of tree t_1 from Example 1. After merging all pairs of corresponding states, the resulting automaton will be $FM_{ora}(pref(t_1))$ and is transition diagram is depicted in Fig. 2. For more information on its construction, see [5].

The constructed string factor oracle automaton now accepts string $x = b2\ b0\ a2\ a0\ a0$, which is not a factor of $pref(t_1) = b2\ b0\ a2\ a0\ a2\ a0\ a0$. String x was created from $pref(t_1)$ by omitting one repeat of string $a2\ a0$. □

III. Properties of subtrees in prefix and prefix bar notation

In this section we present some general properties of the prefix and the prefix bar notation of a tree.

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

Proof: See [1], [6], [7]. ■

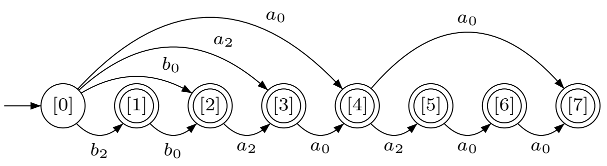


Fig. 2. Transition diagram of the deterministic string factor oracle automaton $FM_{ora}(pref(t_1))$ for prefix notation $pref(t_1) = b2\ b0\ a2\ a0\ a2\ a0\ a0$ of tree t_1 from Example 2

However, not every substring of $pref(t)$ or $pref_{bar}(t)$ is a prefix notation of a subtree. This property is formalised by the following definitions and theorems.

Definition 4. *Let $w = a_1a_2 \ldots a_m$, $m \geq 1$, be a string over a ranked alphabet $\mathcal{A}$. Then, the* arity checksum *$ac(w) = arity(a_1) + arity(a_2) + \ldots + arity(a_m) - m + 1 = \sum_{i=1}^{m} arity(a_i) - m + 1$.*

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

Proof: See [1], [6], [7]. ■

Similar properties can be seen for unranked trees and their prefix bar notation.

Definition 5. *Let $w = a_1a_2 \ldots a_m$, $m \geq 1$, be a string over $\mathcal{A} \cup \{]\}$. Then, the* bar checksum *is defined as follows:*

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

Lemma 1. *Let w, $w = b_1b_2 \ldots b_m$, $m \geq 2$ be a string over alphabet $\mathcal{A} \cup \{]\}$ such that $bc(w) = 0$, and $bc(w_1) \geq 1$ for each w_1, where $w = w_1x$, $x \neq \varepsilon$. Then $b_1 \in \mathcal{A}$ and $b_m =]$.*

Proof: See [7]. ■

Theorem 3. *Let $pref_{bar}(t)$ and w be a tree t in bar notation and a substring of $pref_{bar}(t)$, respectively. Then, w is the bar notation of a subtree of t, if and only if $bc(w) = 0$, and $bc(w_1) \geq 1$ for each w_1, where $w = w_1x$, $x \neq \varepsilon$.*

Proof: See [7]. ■

Theorem 4. *Let $M = (\{Q, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$ be an input–driven PDA of which each transition from δ is of the form $\delta(q_1, a, S) = (q_2, S^i)$, where $i = arity(a)$.*
Then, if $(q_3, w, S) \vdash_M^+ (q_4, \varepsilon, S^j)$, then $j = ac(w)$.

Proof: See [7]. ■

IV. Subtree pushdown automata

A. Subtree PDA for trees in prefix notation

A subtree pushdown automaton for ranked ordered trees has been introduced in [1]. Nondeterministic subtree PDA for trees in prefix notation is an input–driven PDA constructed by Alg. 1.

Algorithm 1. Construction of a nondeterministic subtree PDA for a tree t in prefix notation $pref(t)$.
Input: A tree t; prefix notation $pref(t) = a_1a_2 \ldots a_n$, $n \geq 1$.

Output: Nondeterministic subtree PDA $M_{nps}(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

1) For each state i, where $1 \leq i \leq n$, create a new transition $\delta(i-1, a_i, S) = (i, S^{Arity(a_i)})$, where $S^0 = \varepsilon$.
2) For each state i, where $2 \leq i \leq n$, create a new transition $\delta(0, a_i, S) = (i, S^{Arity(a_i)})$, where $S^0 = \varepsilon$. □

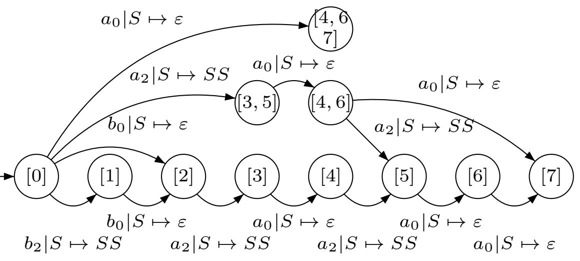
Each nondeterministic input–driven PDA can be transformed to an equivalent deterministic input–driven PDA.

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

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

The deterministic subtree PDA for a tree in prefix notation is demonstrated by the following example.

Example 3. The deterministic subtree PDA for tree t_1 in prefix notation from Example 1, which has been constructed by Alg. 2 and then determinised is PDA $M_{dps}(t_1)$. Its transition diagram is illustrated in Fig. 3.



$a_0|S \mapsto \varepsilon$
$[4,6,7]$
$a_2|S \mapsto SS$
$a_0|S \mapsto \varepsilon$
$a_0|S \mapsto \varepsilon$
$[3,5]$
$[4,6]$
$b_0|S \mapsto \varepsilon$
$a_2|S \mapsto SS$
$[0]$
$[1]$
$[2]$
$[3]$
$[4]$
$[5]$
$[6]$
$[7]$
$b_0|S \mapsto \varepsilon$
$a_0|S \mapsto \varepsilon$
$a_0|S \mapsto \varepsilon$
$b_2|S \mapsto SS$
$a_2|S \mapsto SS$
$a_2|S \mapsto SS$
$a_0|S \mapsto \varepsilon$

Fig. 3. Transition diagram of deterministic subtree PDA $M_{dps}(t_1)$ for tree in prefix notation $pref(t_1) = b2\ b0\ a2\ a0\ a2\ a0\ a0$ from Example 3

During the processing of an input subtree st in prefix notation $pref(st) = a_2 a_0 a_0$, the automaton changes states in sequence $[0]$,$[3,5]$,$[4,6]$ and accepts the input in state $[5]$ by an empty pushdown store. □

Theorem 5. *Given a tree t and its prefix notation $pref(t)$, the PDA $M_{nps}(t)$ constructed by Alg. 1 is a subtree PDA for $pref(t)$.*

Proof: See [1]. ■

Theorem 6. *Given an acyclic input–driven nondeterministic PDA $M_{nx}(t) = (Q, \mathcal{A}, \{S\}, \delta, q_0, S, \emptyset)$, the deterministic PDA $M_{dx}(t) = (Q', \mathcal{A}, \{S\}, \delta', \{q_0\}, S, \emptyset)$ constructed by Alg. 2 is equivalent to PDA $M_{nx}(t)$.*

Proof: See [7]. ■

B. Subtree PDA for Prefix Bar Notation

Similarly, we can also construct subtree PDAs for unranked trees in the prefix bar notation.

Algorithm 3. Construction of a nondeterministic subtree PDA for a tree t in prefix bar notation $pref_{bar}(t)$.
Input: A tree t; prefix bar notation $pref_{bar}(t) = a_1 a_2 \ldots a_n$, $n \geq 2$.
Output: Nondeterministic subtree PDA $M_{npbs}(t) = (\{0, 1, 2, \ldots, n\}, \mathcal{A} \cup \{]\}, \{S\}, \delta, 0, S, \emptyset)$.
Method:

1) For each state i, where $2 \leq i \leq n$, create a new transition
$$\delta(i-1, a_i, S) = \begin{cases} (i, \varepsilon) & \text{for } a_i =] \\ (i, SS) & \text{for } a_i \in \mathcal{A}. \end{cases}$$
2) For each state i, where $1 \leq i \leq n$, create a new transition $\delta(0, a_i, S) = (i, S)$. □

Again, the nondeterministic input–driven PDA can be transformed to an equivalent deterministic PDA by Algorithm 2.

Theorem 7. *Given a tree t and its prefix bar notation $pref_{bar}(t)$, the PDA $M_{npbs}(t)$ constructed by Alg. 3 is a subtree PDA for $pref_{bar}(t)$.*

V. SUBTREE ORACLE PDA

This section deals with subtree oracle pushdown automata. Properties of these pushdown automata will be shown on an example. Similarly to stringology, we construct *subtree oracle automaton* with the use of the definition of corresponding states (see Definition 3). For this purpose, we use a string representing the tree in prefix notation as a part of the definition of corresponding states.

Definition 6. *Let M be a subtree automaton for tree t. Let q_1, q_2 be different states of M. Let there exist two sequences of transitions in M: $(q_0, w_1) \vdash^* (q_1, \varepsilon)$, and $(q_0, w_2) \vdash^* (q_2, \varepsilon)$.*

If w_1 is a suffix of w_2 and w_2 is a prefix of $pref(t)$, then q_1 and q_2 are corresponding states.

Definition 7. *Let $M_{dps}(t)$ be deterministic subtree pushdown automaton (PDA) accepting all subtrees of tree t.* Subtree oracle PDA *$M_{ops}(t)$ is a pushdown automaton created from $M_{dps}(t)$ by merging all corresponding states.*

Using our style of node numbering from 0 to n, it holds that labels of every two mutually corresponding states d_{q_1}, d_{q_2} have

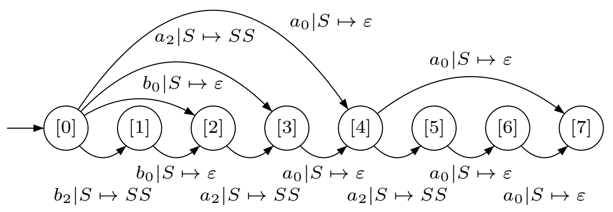

$a_2|S \mapsto SS$
$a_0|S \mapsto \varepsilon$
$a_0|S \mapsto \varepsilon$
$b_0|S \mapsto \varepsilon$
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
$b_0|S \mapsto \varepsilon$
$a_0|S \mapsto \varepsilon$
$a_0|S \mapsto \varepsilon$
$b_2|S \mapsto SS$
$a_2|S \mapsto SS$
$a_2|S \mapsto SS$
$a_0|S \mapsto \varepsilon$

Fig. 4. Transition diagram of deterministic subtree oracle PDA $M_{ops}(t_1)$ for tree t_1 from Example 4

the same minimal value of their d-subsets, ie. $min(d_{q_1}) = min(d_{q_2})$.

Example 4. We construct the subtree oracle automaton for tree t_1 from Example 1 defined in prefix notation as $pref(t_1) = b_2\ b_0\ a_2\ a_0\ a_2\ a_0\ a_0$.

The transition diagram of the deterministic subtree PDA for tree t_1 is illustrated on Fig. 3. We can see three pairs of corresponding states: $([3],[3,5])$, $([4],[4,6])$, $([4],[4,6,7])$

To construct subtree oracle PDA, we will merge all pairs of corresponding states. The resulting automaton is deterministic PDA $M_{ops}(t_1)$, depicted in Fig. 4.

We see that PDA now accepts $Pref(t_2) = b_2 b_0 a_2 a_0 a_0$, which is a subsequence of $pref(t_1)$. This property refers to the property of string factor oracle automata (see Example 2). □

VI. Properties of oracle pushdown automata

Using PDAs instead of finite automata may lead to the elimination of some negative properties which are present in string factor oracle automata:

First, during the process of determinisation, the cancellation condition $cpds(q) = \{\varepsilon\}$ in Algorithm 2 indicates all states from which all outgoing transitions can be omitted because of invalid pushdown operations. This means that some states become inaccessible and can be removed. Omitted transitions would have been responsible for accepting more inputs.

Second, we can define conditions for so-called *safe merging* of corresponding states. This safe merge does not affect the accepted language.

Definition 8. *We define the minimal input arity checksum of state* $q \in Q$ *as* $AC^{-}_{min}(q) = min\{i : (q_0, xy, S) \vdash^* (q, y, S^i), x, y \in \mathcal{A}^*, i \geq 0\}$.

Definition 9. *We define the minimal input arity checksum of state* $q \in Q$ *for input symbol* $a \in \mathcal{A}$ *as* $ac^{-}_{min}(q,a) = min\{i : (q_0, xay, S) \vdash^* (q, y, S^i), x, y \in \mathcal{A}^*, i \geq 0\}$.

AC^{-}_{min} specifies the minimal number of pushdown symbols that can appear in the pushdown store in the state q after reading any input x. ac^{-}_{min} specifies the last read symbol before reaching the state q (ie the label of the last transition is used). By replacing *min* function with *max*, we could define AC^{-}_{max} and ac^{-}_{max}.

Definition 10. *We define maximal output arity checksum of state* $q \in Q$ *as* $AC^{+}_{max}(q) = max\{i : (q, x, S^i) \vdash^* (r, \varepsilon, \varepsilon), x \in \mathcal{A}^*, r \in Q, i \geq 0\}$.

Definition 11. *We define maximal output arity checksum of state* $q \in Q$ *for input symbol* $a \in \mathcal{A}$ *as* $ac^{+}_{max}(q,a) = max\{i : (q, ax, S^i) \vdash^* (r, \varepsilon, \varepsilon), x \in \mathcal{A}^*, r \in Q, i \geq 0\}$.

AC^{+}_{max} specifies the maximal number of pushdown symbols that can be removed starting from the state q by reading arbitrary input. ac^{+}_{max} specifies first symbol of that input. By replacing *max* function with *min*, we could define AC^{+}_{min} and ac^{+}_{min}.

Lemma 2. *Two distinct corresponding states q and r of deterministic subtree pushdown automata can be safely merged if at least one of following conditions is fulfilled:*

1) *For every input symbol* $a \in \mathcal{A}$ *such that* $\delta(q,a,S) \neq \delta(r,a,S)$*, it holds that* $ac^{+}_{max}(r,a) < AC^{-}_{min}(q)$ *and* $ac^{+}_{max}(q,a) < AC^{-}_{min}(r)$
2) *Either* $AC^{-}_{max}(q) = AC^{+}_{max}(q) = 0$ *or* $AC^{-}_{max}(r) = AC^{+}_{max}(r) = 0$.

VII. Conclusion

We have described oracle modification of subtree pushdown automata for ordered ranked and unranked trees in prefix and prefix bar notation, respectively. These pushdown automata are analogous in their properties to string factor oracle automata, which are widely used in stringology. The presented pushdown automata allow quick rejection of input patterns that are not subtrees of the subject tree or can be used for efficient backward oracle subtree matching. There are open questions for future research. First, the tree language accepted by the subtree oracle pushdown automaton should be investigated in details. Second, an algorithm for minimising the deterministic subtree PDA using only the safe merging of states, which is introduced in this paper, should also be researched.

For more information on tree algorithms using PDAs, see [8].

References

[1] 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.

[2] C. Allauzen, M. Crochemore, and M. Raffinot, "Factor oracle: A new structure for pattern matching," in *SOFSEM*, ser. Lecture Notes in Computer Science, J. Pavelka, G. Tel, and M. Bartosek, Eds., vol. 1725. Springer, 1999, pp. 295–310.

[3] M. Crochemore and C. Hancart, "Automata for matching patterns," in *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds. Springer–Verlag, Berlin, 1997, vol. 2 Linear Modeling: Background and Application, ch. 9, pp. 399–462.

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

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

[6] B. Melichar, "Arbology: Trees and pushdown automata," in *LATA*, ser. Lecture Notes in Computer Science, A. H. Dediu, H. Fernau, and C. Martín-Vide, Eds., vol. 6031. Springer, 2010, pp. 32–49, invited paper.

[7] J. Janoušek, *Arbology: Algorithms on Trees and Pushdown Automata*. Brno: Habilitation thesis, TU FIT, 2010.

[8] "Arbology www pages," Available on: http://www.arbology.org/, July 2011.