Oracle Pushdown Automata for Trees in Prefix Notation

Martin Plicka\textsuperscript{1} \hspace{1cm} Jan Janoušek\textsuperscript{2} \hspace{1cm} Bořivoj Melichar\textsuperscript{2}

\textsuperscript{1}Department of Computer Science and Engineering  
Czech Technical University in Prague, Faculty of Electrical Engineering

\textsuperscript{2}Department of Theoretical Computer Science  
Czech Technical University in Prague, Faculty of Information Technology

London Stringology Days & London Algorithmic Workshop, 2010
Outline

1 Introduction
   • Motivation
   • Oracle Automata in Stringology

2 Pushdown Automata for Trees in Prefix Notation
   • Construction
   • Deterministic PDA

3 Oracle Pushdown Automata
   • Construction
   • Safe Oracle
Arbology tries to use the same principles on trees as they are used on strings in stringology, using pushdown automata instead of finite automata as computation model.

Factor oracle automata in stringology reduce the amount of states to minimal value . . .

. . . with some disadvantages.

What about creating oracle automata for trees in prefix notation?
Factor Oracle Automata in Stringology

- Suffix automaton accepts all suffixes of given string. Factor automaton accepts all factors (substrings) of subject string.
- Number of distinct factors of a string is limited by $O(n^2)$ where $n$ is length of the string. Number of states of factor automaton is linear in $n$. Search time complexity is $m$ where $m$ is length of searched string.
- Factor oracle automata have exactly $n + 1$ states.
- Unfortunately, factor oracle automata also accept some subsequences (discontinuous substrings) of subject string.
- Despite this property, they can be still used for indexing of strings.
Corresponding States

Definition

Let $M$ be the factor automaton for string $w$ and $q_1$, $q_2$ be different states of $M$. Let there exist two sequences of transitions in $M$:

$$(q_0, w_1) \vdash^* (q_1, \varepsilon), \text{ and}$$

$$(q_0, w_2) \vdash^* (q_2, \varepsilon).$$

If $w_1$ is a suffix of $w_2$ and $w_2$ is a prefix of $w$ then $q_1$ and $q_2$ are corresponding states.

The factor oracle automaton can be constructed by merging the corresponding states.
Construction of Factor Oracle

**Input:** String $w$,

**Output:** Deterministic factor oracle automaton $Oracle(w)$.

1. Construct nondeterministic factor automaton for given string $w$.
2. Create equivalent deterministic automaton.
3. Merge all corresponding states.
4. If result automaton is not deterministic, repeat steps 2 and 3.
Example

Factor automaton for string $x = abba$.
$Fact(x) = \{a, b, ab, bb, ba, abb, bba, abba\}$

1. Non deterministic version,
2. deterministic version.
Example - cont.

Factor automaton for string "abba".

1. Deterministic version.
2. Deterministic factor oracle. It additionally accepts string "aba" which is not factor of "abba".
Subtree and tree pattern automata have similar behavior to suffix and factor automata, respectively, from stringology.

Grammar of tree prefix notation is LL(1) – we can use pushdown automata for accepting trees or their subtrees in prefix notation.

Arity checksum = number of nodes to be read to have complete tree (number of remaining symbols on pushdown store).
\[ ac(q) = 0 \Rightarrow \text{accept} . \]

We need the only one pushdown symbol \( \Rightarrow \) pushdown store can be replaced by counter.

They represent index for quick searching in trees, using time linear in \( m \) where \( m \) is a length of string representing the tree in prefix notation.
Example

Tree $t_1$, $\text{pref}(t_1) = b_2 b_0 a_2 a_0 a_2 a_0 a_0$

Subtrees are:

1. $b_2 b_0 a_2 a_0 a_2 a_0 a_0$
2. $a_2 a_0 a_2 a_0 a_0$
3. $a_2 a_0 a_0$
4. $a_0$
5. $b_0$

Tree patterns are (for example):

1. $b_2 b_0 S$
2. $b_2 b_0 a_2 S a_2 a_0 a_0$
3. $a_2 a_0 S$
4. $\ldots$
Example - Accepting Whole Tree

\[
\begin{align*}
0 & : b_2 | S \rightarrow SS \\
1 & : b_0 | S \rightarrow \varepsilon \\
2 & : a_2 | S \rightarrow SS \\
3 & : a_0 | S \rightarrow \varepsilon \\
4 & : a_2 | S \rightarrow SS \\
5 & : a_0 | S \rightarrow \varepsilon \\
6 & : a_0 | S \rightarrow \varepsilon \\
7 & : 
\end{align*}
\]
Example - Accepting Subtrees
Example - Accepting Tree Patterns
Making the PDA Deterministic

- Both subtree and tree pattern PDAs can be always transformed into deterministic ones because they are input-driven – pushdown operations are determined by input symbol.
- Algorithm is the same as in case of finite automata.
- States which are accessible by only invalid pushdown operation (reading pushdown symbol from empty pushdown store), are omitted since our PDA accepts input by reaching the empty pushdown store.
- It seems that although the number of possible tree patterns are exponential in $n$, maximum number of states of deterministic tree pattern PDA is quadratic in $n$. 
Example - Deterministic Subtree PDA

\[
\begin{align*}
0 & \rightarrow S \\
1 & \rightarrow S \\
2 & \rightarrow S \\
3 & \rightarrow S \\
4 & \rightarrow S \\
5 & \rightarrow S \\
6 & \rightarrow S \\
7 & \rightarrow S
\end{align*}
\]
Oracle PDA - Construction

- Construction adapts all steps from the construction of string factor oracle automata.
- Corresponding states are defined in similar manner.
- Oracle PDA is constructed by merging all pairs of corresponding states.
Example for Subtree

Corresponding states are marked with distinct colours.
This automaton accepts tree $t_2$.

Tree $t_2$ is not subtree of subject tree $t_1$.

$\text{Pref}(t_2)$ is a subsequence of $\text{Pref}(t_1)$, created by omitting repetition of $a_2a_0$ in $\text{Pref}(t_1)$. 
Two significant properties cause that for trees, the number of false positive accepts should be smaller.

1. During the transformation to PDA, using the stop condition of empty pushdown store, inaccessible states are omitted.
2. Only strings representing trees are accepted.

During the state merge, outgoing transition from one state, in combination with incoming transition to the second state, may cause automaton accepting new tree.

Second property makes room for specifying conditions of "safe" state merge.
Definition

Minimal input arity checksum of state $q \in Q$: 
$$AC_{\min}^{-}(q) = \min\{i : (q_0, \alpha \beta, S) \vdash^* (q, \beta, S^i), \alpha \in \mathcal{A}^*, \beta \in \mathcal{A}^*, i \geq 0\}$$

Definition

Minimal input arity checksum of state $q \in Q$ for input symbol $x \in \mathcal{A}$: 
$$ac_{\min}^{-}(q, x) = \min\{i : (q_0, \alpha x \beta, S) \vdash^* (q, \beta, S^i), \alpha \in \mathcal{A}^*, \beta \in \mathcal{A}^*, i \geq 0\}$$

- The triple represents pushdown automata configuration. $\mathcal{A}$ is a ranked alphabet. $q_0 \in Q$ is initial state of pushdown automata.
- In other words, minimal input arity checksum specifies the minimal number of pushdown symbols that can appear in pushdown store in state $q$ after reading arbitrary input $\alpha$.
- The arity checksum for particular input only specifies the last transition leading to the state $q$.
- Similarly, we can define "max" ($AC_{\max}^{-}$, $ac_{\max}^{-}$) versions as well.
**Output Arity Checksum**

**Definition**

Maximal output arity checksum of state $q \in Q$:

$$AC_{\text{max}}^+(q) = \max\{i : (q, \alpha, S^i) \vdash^* (r, \varepsilon, \varepsilon), \alpha \in A^*, r \in Q, i \geq 0\}$$

**Definition**

Maximal output arity checksum of state $q \in Q$ for input symbol $x \in A$:

$$ac_{\text{max}}^+(q, x) = \max\{i : (q, x\alpha, S^i) \vdash^* (r, \varepsilon, \varepsilon), \alpha \in A^*, r \in Q, i \geq 0\}$$

- The triple represents pushdown automata configuration. $A$ is a ranked alphabet.
- Maximal output arity checksum specifies the maximal number of pushdown symbols that can be read starting from state $q$ after reading arbitrary input $\alpha$.
- The arity checksum for particular input symbol only specifies first transition leading from the state $q$.
- Similarly, we can define "min" ($AC_{\text{min}}^+, ac_{\text{min}}^+$) versions as well.
Example

\[ AC_{\text{min}}([4, 6, 7]) = AC_{\text{max}}([4, 6, 7]) = 0 \]

\[ AC_{\text{min}}([4, 6]) = ac_{\text{max}}([4, 6], a2) = ac_{\text{max}}([4, 6], a0) = 1 \]

\[ AC_{\text{min}}([5]) = 2 \text{ (by reading either } a2a0a2 \text{ or } b2b0a2a0a2) \]
Safe merge

Lemma

Two distinct corresponding states $q$, $r$ of deterministic subtree pushdown automata can be "safely" merged if at least one of following conditions is met:

1. Either $AC_{\text{max}}^{-}(q) = 0$ or $AC_{\text{max}}^{-}(r) = 0$.
2. For every input symbol $x \in A$ such that $\delta(q, x, S) \neq \delta(r, x, S)$, it holds that $ac_{\text{max}}^{+}(r, x) < AC_{\text{min}}^{-}(q)$ and $ac_{\text{max}}^{+}(q, x) < AC_{\text{min}}^{-}(r)$

By sequential safe merging of corresponding states, we may create "safe" oracle PDA.
Example of Safe Merge

\[ \text{Pref}(t_3) = a_3 a_2 a_0 a_0 a_2 a_0 b_0 a_2 a_0 a_0 \]

\[ AC_{max}^-([4, 10]) = 0 \rightarrow \text{condition 1 is satisfied.} \]
Example of Safe Merge - cont.

\[ \text{Pref}(t_3) = a_3 a_2 a_0 a_2 a_0 b_0 a_2 a_0 a_0 \]

\[ ac_{max}^+ ([3], b_0) = 0 \quad < \quad AC_{min}^- ([3, 6, 9]) = 1 \]

\[ ac_{max}^+ ([3, 6, 9], b_0) = 2 \quad < \quad AC_{min}^- ([3]) = 3 \]

\[ \rightarrow \text{condition 2 is satisfied.} \]
Example of Safe Merge - cont.

$$\text{Pref}(t_3) = a_3a_2a_0a_0a_2b_0a_2a_0a_0$$

There is no input symbol $x$ such as $\delta([2, 5, 8], x, S) \neq \delta([2], x, S)$

$\Rightarrow$ condition 2 is satisfied.
Example of Safe Merge - cont.

\[ \text{Pref}(t_3) = a_3a_2a_0a_0a_2a_0b_0a_2a_0a_0 \]

\[ AC_{\text{max}}([3, 4, 6, 9, 10]) = 0 \]

⇒ condition 1 is satisfied.
Example of Safe Merge - cont.

\[ \text{Pref}(t_3) = a_3a_2a_0a_0a_2a_0b_0a_2a_0a_0 \]

This automaton does not accept any additional input.
Oracle automata for trees in prefix notation were introduced.

General properties of such automata were illustrated,
including safe state merging definition.

Open problems:
- Define properties of false accepted trees.
- Find common properties of some kind of input trees (e.g. repetitions).

www.arbology.org
Oracle automata for trees in prefix notation were introduced.
General properties of such automata were illustrated,
including safe state merging definition.

Open problems:
- Define properties of false accepted trees.
- Find common properties of some kind of input trees (e.g. repetitions).

www.arbology.org