Simple Tree Pattern Matching for Trees in the Prefix Bar Notation

Jan Lahoda[1] and Jan Žďárek[2,⋆]

[1] Sun Microsystems Czech,
V Parku 2308/8, 148 00 Praha 4, Czech Republic,
Jan.Lahoda@sun.com
[2] Department of Theoretical Computer Science,
Faculty of Information Technology,
Czech Technical University in Prague,
Kolejní 550/2, 160 00 Prague 6, Czech Republic
Jan.Zdarek@fit.cvut.cz

Abstract. A new pushdown automata based algorithm for searching all occurrences of a tree pattern in a subject tree is presented. The algorithm allows pattern matching with don't care symbols and multiple patterns. A simulation algorithm is also proposed, and practical experimental results are presented.

1 Introduction

Tree pattern matching has numerous applications in computing, for example in program optimization, code generation and refactoring. It has been researched thorougly for several decades, see Janoušek and Melichar [9]. Recently, a new stream of research has been started by Janoušek and Melichar [9]. They consider trees in the postfix notation as strings and present a 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. Based on this fundamental result, Melichar *et al.* started to extend principles of text pattern matching using finite automata into the tree pattern matching domain. They use pushdown automata for matching in trees, where trees are represented by their prefix or postfix notation [8,4,5]. These automata are either constructed directly as deterministic pushdown automata, or they are nondeterministic input-driven pushdown automata. The nondeterminism can be removed in the latter case, as it is known that any input-driven pushdown automaton can be determinised [13]. The prefix bar notation is the prefix notation of a rooted ordered labeled directed tree where only closing bracket of a bracket pair is used. The prefix bar notation was introduced by Stoklasa, Janoušek and Melichar in [10,11]. A detailed overview of the tree matching algorithms based on pushdown automata is due to Janoušek [7].

In this paper we propose a new algorithm for tree pattern matching. The algorithm allows to perform tree pattern matching with don't cares, including multiple tree patterns, by means of pushdown automata. The pushdown automata constructed by our algorithm are visibly pushdown [1] and so can be determinised. As the determinised versions of the visibly pushdown automata can be quite big (see Section 3), a simulation algorithm is also proposed for the constructed automata. The simulation algorithm was evaluated experimentally and the results are presented in the final part of the paper.

⋆ Partially supported by GAČR project No. 201/09/0807 and MŠMT project No. MSM 6840770014.

Jan Lahoda, Jan Žďárek: Simple Tree Pattern Matching for Trees in the Prefix Bar Notation, pp. 25–36 .
Proceedings of PSC 2010, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04597-8

The motivation to invent the algorithm described herein was a tool that allows to quickly search vast amounts of source code and find given patterns in the code. The tool is given one or more AST snippets ("patterns"), including don't cares, and then it processes a huge number of ASTs and searches for occurrences of the pattern(s) in these ASTs. To fulfill this task, the tool preprocesses the pattern once, and then analyses the ASTs, processing them on the fly.

1.1 Definitions

Let A be a finite alphabet and its elements be called symbols. A set of strings over A is denoted by A^*. A language L is any subset of A^*, $L \subseteq A^*$. The empty string is denoted by ε. The "don't care" symbol is a special universal symbol that matches any other symbol including itself [3].

A finite automaton (FA) is a quintuple (Q, A, δ, I, F). Q is a finite set of states, A is a finite input alphabet, $F \subseteq Q$ is a set of final states. If an FA is nondeterministic (NFA), then δ is a mapping $Q \times (A \cup \{\varepsilon\}) \mapsto \mathcal{P}(Q)$ and $I \subseteq Q$ is a set of initial states. A deterministic FA (DFA) is (Q, A, δ, q_0, F), where δ is a (partial) function $Q \times A \mapsto Q$; $q_0 \in Q$ is the only initial state.

The following definitions introduce pushdown automata and related notions. A (nondeterministic) pushdown automaton (PDA), is a septuple $(Q, A, G, \delta, q_0, Z_0, F)$, where Q is a finite set of states, A is a finite input alphabet, G is a finite pushdown store alphabet, δ is a mapping $Q \times (A \cup \{\varepsilon\}) \times G \mapsto \mathcal{P}(Q \times G^*)$, $q_0 \in Q$ is an initial state, $Z_0 \in G$ is the initial pushdown store symbol, $F \subseteq Q$ is a set of final states. A pushdown store operation of PDA M, $M = (Q, A, G, \delta, q_0, Z_0, F)$, is a relation $(A \cup \{\varepsilon\}) \times G \mapsto G^*$. A pushdown store operation produces new contents on the top of the pushdown store by taking one input symbol or the empty string from the input and the current contents on the top of the pushdown store. The pushdown store grows to the right if written as a string x, $x \in G^*$. A transition of PDA M is the relation $\vdash_M \subseteq (Q \times A^* \times G) \times (Q \times A^* \times G^*)$. It holds that $(q, aw, \alpha\beta) \vdash_M (p, w, \gamma\beta)$ 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 PDA is a deterministic PDA if it holds:

1. $|\delta(q, a, \gamma)| \leq 1$ for all $q \in Q$, $a \in A \cup \{\varepsilon\}$, $\gamma \in G$.
2. For all $\alpha \in G$, $q \in Q$, if $\delta(q, \varepsilon, \alpha) \neq \emptyset$, then $\delta(q, a, \alpha) = \emptyset$ for all $a \in A$.

A language L accepted by PDA M is a set of words over finite alphabet A. It is defined in two distinct ways:

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

If the PDA accepts the language by empty pushdown store then the set F of its final states is defined to be the empty set.

Alur and Madhusudan [1] introduced a special type of pushdown automata and languages they accept.

A *visible alphabet* $\tilde{A}$ is a triple $\tilde{A} = (A_c, A_r, A_{int})$. $\tilde{A}$ comprises three categories of symbols, three disjoint finite alphabets: A_c is a finite set of calls (pushdown store

grows), A_r is a finite set of returns (pushdown store shrinks), A_{int} is a finite set of internal actions that do not use the pushdown store.

A *visibly pushdown automaton* (VPA) is a septuple $(Q, \tilde{A}, G, \delta, Q_{in}, Z_0, F)$, where Q is a finite set of states, $\tilde{A}$ is a finite visible alphabet, G is a finite pushdown store alphabet, δ is a mapping: $(Q \times A_c \times \varepsilon) \mapsto (Q \times (G \setminus \{Z_0\})) \cup (Q \times A_r \times G) \mapsto (Q \times \varepsilon) \cup (Q \times A_{int} \times \varepsilon) \mapsto Q \times \varepsilon)$, $Q_{in} \subseteq Q$ is a set of initial states, $Z_0 \in G$ is the initial pushdown store symbol, $F \subseteq Q$ is a set of final states.

All notions related to extended pushdown automata hold for visibly pushdown automata as well. Specifically, the language accepted by visibly pushdown automaton: A *language $L(M)$ accepted by visibly pushdown automaton M* is the set of words accepted by M. A *visibly pushdown language* is a set of words over some finite alphabet A, $L \subseteq A^*$ with respect to $\tilde{A}$ ($\tilde{A}$-VPL) if there exists a visibly pushdown automaton M over $\tilde{A}$ such that $L(M) = L$. Visibly pushdown automata can be determinised.

Let V be a set of nodes, E be a set of edges. A rooted ordered labeled directed tree T, $T = (V, E)$, is a rooted directed tree where every node $v \in V$ is labeled by symbol $a \in A$ and its out-degree is given by the arity of the symbols of A. Nodes labeled by nullary symbols (constants) are called leaves. All trees used in this paper are rooted ordered labeled directed trees.

Let us define the prefix bar notation [11, analogous to Def. 2].

Definition 1. *The prefix bar notation of tree P, with root r and its children $c_1, \ldots, c_n$, denoted by $d(P)$ is defined recursively as follows: $d(P) = rd(c_1) \cdots d(c_n) \uparrow$.*

Note also that r and $\uparrow$ in Definition 1 are symbols of alphabets A_c and A_r, respectively, of a particular visibly pushdown automaton we simulate.

2 Main Idea

In this section, we will describe new algorithms for exact tree pattern matching and for tree pattern matching with don't cares. The algorithms use Euler-like notation to serialize the trees (Ždárek [12, Alg. 3.8]) and finite automata or pushdown automata to perform the matching. The algorithms described herein in fact extend the algorithm described by Flouri, Janoušek and Melichar in [5] where they consider deterministic PDA constructions for subtree pattern matching.

2.1 Exact Pattern Matching

In this section, we will present a simple algorithm for exact tree pattern matching based on finite automata. As noted by Stoklasa, Janoušek and Melichar [11, Theorem 1], the prefix bar notation of a tree contains prefix bar notations of all subtrees of the tree as substrings. The exact pattern matching can therefore be performed easily as follows. First, a finite automaton for exact string pattern matching is constructed for $d(P)$. A string matching algorithm is then used to locate the occurrences of $d(P)$ in $d(T)$, which correspond to occurrences of P in T.

Theorem 2. *Given tree pattern P, containing m total nodes, and subject tree T, containing n total nodes, the aforementioned algorithm for tree pattern matching runs in $\mathcal{O}(n + m)$ time.*

Proof. The deterministic finite automaton constructed for the prefix bar notation of P will have $2m + 1$ states, and can be constructed in $\mathcal{O}(m)$ time (Crochemore [2], Holub [6]). Pattern matching over the prefix bar notation of T using this automaton then takes $\mathcal{O}(n)$ time.

2.2 Pattern Matching with Don't Cares

In this section, we will show how to extend the algorithm described in the previous section to handle don't care tree nodes.

Definition 3. *A leaf node of tree pattern P marked with don't care symbol $+$ matches any single complete subtree in the subject tree T.*

Definition 4. *The prefix bar notation of tree P with don't care symbols, root r and children $c_1, \ldots, c_n$, denoted by $d(P)$ is defined recursively as follows:*

$$d(P) = \begin{cases} rd(c_1) \cdots d(c_n) \uparrow & \text{iff } r \neq + \\ + & \text{iff } r = + \end{cases}$$

The finite automata are not sufficient to model tree pattern matching with don't care symbols. Pushdown automata (PDA) will be used for this task.

Algorithm 1 shows the construction of the pushdown automaton for tree pattern matching with don't cares. The tree pattern matching is then performed over the prefix bar notation of the subject tree.

The PDA constructed by Algorithm 1 is structurally similar to finite automaton for exact tree pattern matching described in the previous section. The pushdown operations for "down" and "up" symbols are as follows. For "down" symbols, pushdown symbol e is pushed to the store, for "up" symbols the same symbol is popped from the store. The reason for pushing and popping this symbol is to ensure that during matching the pushdown store contains as many symbols as is the current depth in the subject tree. This limits the actual pushdown-store non-determinism of the automaton.

The don't cares are translated into the PDA as shown in Figure 1. In the transition from the state s to (inner) state i, the "target" state is remembered in the pushdown store. The loop transitions on the state i ensure that whole subtree will be skipped.

In Algorithm 1, lines 4– 11 construct the states for matching the don't care symbol (as shown in Figure 1). Lines 13–19 construct the basic structure of the automaton. Lines 22–25 construct the loop in the initial state.

For patterns without don't care symbols, the Algorithm 1 constructs automata that are functionally equivalent to finite automata for exact tree pattern matching of trees in prefix bar notation described in Section 2.1. The states and transitions that are created for don't care symbols assure that the automaton will skip symbols that correspond to a complete subtree in the prefix bar notation. The first such symbol pushes a marker symbol at the top of the of the pushdown store. When the closing $\uparrow$ symbol of the complete subtree is processed, this marker is found at the top of the pushdown store, and the matching continues with the following symbol of the pattern's prefix bar notation. The algorithm therefore performs tree pattern matching with don't care symbols for tree in the prefix bar notation.

The PDA constructed by Algorithm 1 is a non-deterministic one. Note, however, that the non-determinism is caused solely by the loop in the initial state. Without it, the automaton would be deterministic and would implement a tree-top search. The PDA is also a visibly pushdown automaton ([1]) and so can always be determinised.

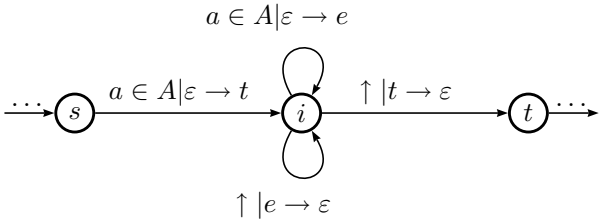

Figure 1. PDA states representing don't care symbol

Algorithm 1 Construction of PDA from tree pattern for tree pattern matching

Input: Pattern tree P in the prefix bar notation
Output: PDA $M = (Q, A, G, \delta, q_0, e, F)$

create state q_0, $q = q_0$, $Q = \{q_0\}$, $G = \{e\}$
for all $a \in d(P)$ **do**
  if a is $+$ **then**
    create new states q_1, q_2
    for all $b \in A$ **do**
      $\delta(q, b, \varepsilon) = \{(q_1, q_2)\}$
      $\delta(q_1, b, \varepsilon) = \{(q_1, e)\}$
    end for
    $\delta(q_1, \uparrow, e) = \{(q_1, \varepsilon)\}$
    $\delta(q_1, \uparrow, q_2) = \{(q_2, \varepsilon)\}$
    $q = q_2$, $G = G \cup \{q_2\}$
  else
    create new state q'
    if a is $\uparrow$ **then**
      $\delta(q, a, e) = \{(q', \varepsilon)\}$
    else
      $\delta(q, a, \varepsilon) = \{(q', e)\}$
    end if
    $q = q'$
  end if
end for
for all $b \in A$ **do**
  $\delta(q_0, b, \varepsilon) = \delta(q_0, b, \varepsilon) \cup \{(q_0, e)\}$
end for
$\delta(q_0, \uparrow, e) = \{(q_0, \varepsilon)\}$
$F = \{q\}$

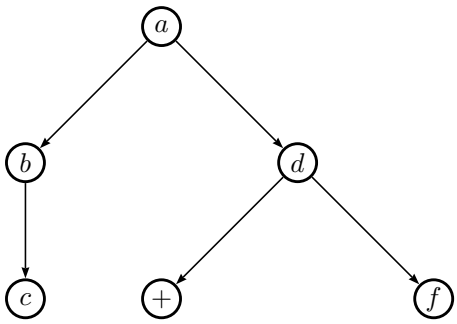

Figure 2. Example tree pattern with don't cares - its prefix bar notation is $abc\uparrow\uparrow d+f\uparrow\uparrow\uparrow$

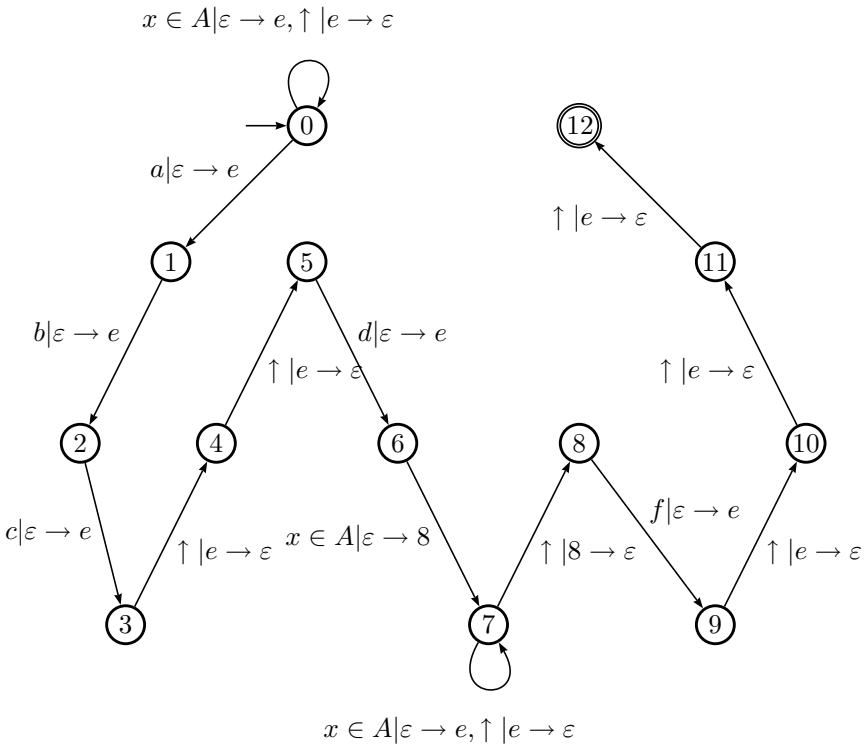

Figure 3. Pattern matching PDA for tree pattern shown in Figure 2

2.3 Multiple Pattern Matching

The algorithms described in the previous sections can be straightforwardly extended to perform the tree pattern matching with don't cares for multiple patterns. For each pattern, the PDA is constructed using Algorithm 1. A new PDA is then constructed from these sub-PDAs by uniting the initial states.

2.4 Simulation

In the previous sections, we have shown a way to construct a non-deterministic pushdown automaton for tree pattern matching with don't cares. The constructed automaton is also visibly pushdown, and so can be determinised. However, as the automaton can become quite big during the determinization process, we will show how to efficiently simulate the non-deterministic automaton. In this section, we will show that the pushdown store non-determinism is limited and base the simulation on this fact.

Lemma 5 (Absence of variable-length branches). *Let T be a tree and $d(T)$ its prefix bar notation. Let $M = (Q, A, G, \delta, q_0, e, F)$ be a PDA constructed by Algorithm 1 for a tree pattern P. Then for each prefix p of $d(T)$ exists an integer l such that for each transition sequence $(q_0, p, e) \vdash^* (q', \varepsilon, s)$, $l = |s|$.*

Proof. In the PDA created by Algorithm 1, all the transitions for $\uparrow$ pop a symbol from the pushdown store and all transitions for the other symbols push a symbol to the pushdown. Consequently, the depth of the pushdown store depends only on the number of $\uparrow$ and non-$\uparrow$ symbols in the currently processed prefix, not on the sequence of transitions.

Note 6 (No interference on the pushdown store). Let $M = (Q, A, G, \delta, q_0, e, F)$ be a PDA constructed by Algorithm 1 for a tree pattern P. Then there are no two states $q_1, q_2 \in Q$, $q_1 \neq q_2$, $a \in A$ and $s, u, w \in G$, $s \neq e$ such that $\delta(q_1, a, w) = (q_1', ws)$, $\delta(q_2, a, u) = (q_2', us)$.

The meaning of Lemma 5 is that there are no variable-length branches of the pushdown store while simulating this automaton. Note 6 points out that no two distinct transitions store the same symbol to the pushdown store. These two observations together assure that the pushdown store of PDA constructed by Algorithm 1 can be simulated using bit parallelism. The simulation algorithm based on bit-parallelism is described below.

The simulation is shown in Algorithm 2. The simulation is based on the simulation of non-deterministic finite automata using bit-parallelism. Variable W consisting of $|Q|$ bits contains the bit mask of active states (one bit of W is assigned to each state, active and inactive states have bit value 1 and 0, respectively). Each entry of pushdown store S, consisting of $|G|$ bits, contains the bit mask of the current pushdown symbols (one bit is assigned to each pushdown symbol except e, if the symbol is in the pushdown store at the given level, value 1 is used, 0 otherwise). Symbol e is always on the pushdown store and does not need to be encoded.

Theorem 7. *Algorithm 2 runs in $\mathcal{O}(nm^2)$ worst case time, where n is the number of nodes of the subject tree and m is the number of nodes of the pattern tree.*

Algorithm 2 Simulation of PDA for tree pattern matching

Input: Subject tree T in the prefix bar notation, PDA $M = (Q, A, G, \delta, q_0, e, F)$
Output: Root nodes of the occurrences of tree pattern P in subject tree T

```
 1: W = {q_0}, S = ∅
 2: for all a ∈ d(T) do
 3:    if a is ↑ then
 4:       W' = ∅
 5:       for all q ∈ W do
 6:          (q', ε) = δ(q, ↑, e)                    /*at most one such entry*/
 7:          W' = W' ∪ {q'}
 8:       end for
 9:       W' = W' ∪ pop element from S
10:       W = W'
11:    else
12:       W' = ∅, S' = ∅
13:       for all q ∈ W do
14:          for all (q', s') in δ(q, a, ε) do
15:             if s' = e then
16:                W' = W' ∪ {q'}
17:             else
18:                S' = S' ∪ {s'}
19:             end if
20:          end for
21:       end for
22:       W = W', push S' to S
23:    end if
24:    if W ∩ F ≠ ∅ then
25:       found occurrences of P in T
26:    end if
27: end for
```

Proof. The main loop starting at line 2 iterates over $2n$ elements of the prefix bar notation of the subject tree T. Both branches of the *if* statement on line 3 iterate over the currently active states, perform transitions in the PDA and merge the results using union. There may be up to $\mathcal{O}(m)$ active states and each union takes up to $\mathcal{O}(m)$ time. As $\delta(q, a, \varepsilon)$ may contain at most two elements (for $q = q_0$), the loop on line 14 is performed at most twice. Therefore, each pass through the if statement on line 3 takes $\mathcal{O}(m^2)$ time. The intersection on line 24 takes $\mathcal{O}(|F|)$ time ($\mathcal{O}(m)$ in the worst case). Therefore, the total worst case time complexity of the algorithm is $\mathcal{O}(nm^2)$.

An example of the simulation is given in Figure 5. The subject tree of the tree pattern matching is depicted in Figure 4, the tree pattern is shown in Figure 2 and the corresponding PDA for tree pattern matching is depicted in Figure 3. The figure presents set of the active states W and pushdown store S, used to identify the end of a subtree matched by a don't care symbol in the pattern. Their values are displayed before and after the reading of each input symbol.

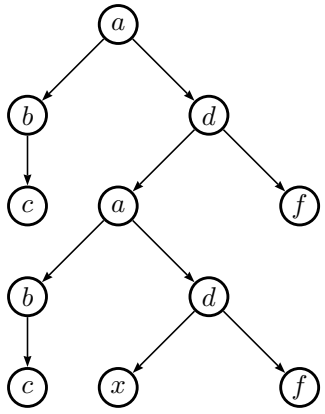

Figure 4. Example subject tree for example shown in Figure 5

text		*a*	*b*	*c*	↑	↑	*d*	*a*	*b*	*c*	↑	↑	*d*	*x*	↑	*f*	↑	↑	↑	*f*	↑	↑	↑	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	2	3	4	5	6													8	9	10	11	**12**
								1	2	3	4	5	6		8	9	10	11	**12**				
S								{8}				{8}		{8}				{8}						
									{8}		{8}		{8}		{8}		{8}							
										{8}				{8}		{8}								

Figure 5. Example of simulation using PDA depicted in Figure 3 in the subject tree depicted in Figure 4; W is the set of active states, S is the pushdown store; $F = \{12\}$

3 Experimental Results

To evaluate the practical properties of the proposed algorithm, we have implemented the algorithm and performed several experiments.

First, let us discuss the number of states and pushdown symbols in non-deterministic and deterministic pushdown automata for tree pattern matching. We have implemented an incremental version of determinization algorithm described by Alur and Madhusudan [1]. Consider for example tree pattern depicted in Figure 6. The corresponding non-deterministic pushdown automaton has 19 states and 6 pushdown symbols. When this automaton is determinised, the resulting deterministic pushdown automaton has 20087 accessible states and 154 used pushdown symbols. Although this is significantly less than the upper bound of number of states and number of pushdown symbols (which is in this case $2^{19 \cdot 6}$ states and and $2^{6 \cdot 6}$ pushdown symbols), the absolute number of states is still significant and it would be impractical to keep such big automata.

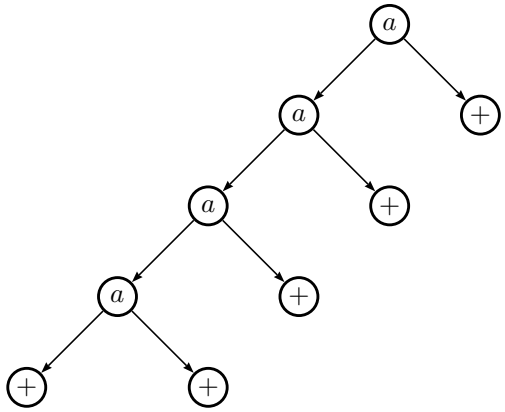

Figure 6. Example tree for which corresponding tree pattern matching PDA is determinised

Furthermore, we tested the impact of the use of bit parallelism for simulating the non-deterministic PDA for tree pattern matching, 46433 Java source files were searched for selected patterns. The tested source files were a complete NetBeans IDE main code base, available at `http://hg.netbeans.org/main-golden`, the changeset on which the experiments were performed was `96f614d8662d`. The tested patterns are shown in Table 1. Let us note that the patterns are expanded before the PDA is constructed using the algorithms denoted above. This expansion produces a set of patterns for each input pattern for the user's convenience. For example the pattern

```
org.openide.util.RequestProcessor.getDefault()
```

is augmented with patterns like

`RequestProcessor.getDefault()` and `getDefault()`.

The experimental results are summarized in Table 2. The results suggest that the number of states active at any given time during the matching is very low in practice. The simulation algorithm is therefore practically viable.

1. method invocation:

```
org.openide.util.RequestProcessor.getDefault()
```

2. double checked locking:

```
if ($var == null) {
     synchronized($lock) {
         if ($var == null) $statements;
     }
}
```

3. 151 standard NetBeans IDE patterns

Table 1. Tested patterns

pattern name	total states	active states		running time [s]
		average	maximum	
method invocation	20	1.54	3	163
double checked locking	449	1.52	12	121
all	3853	6.67	70	254

Table 2. Summary of experimental results. For tested patterns see Table 1

4 Conclusion

In this paper we have presented a new algorithm for tree pattern matching. The algorithm is based on pushdown automata and supports both don't care symbols and multiple patterns. An algorithm for efficient simulation of the automaton is given.

We see several possible directions for future research. One possible direction is to investigate possibility of don't cares which would match any number of complete subtrees (i.e. don't cares with a variable arity). It would also be possible to investigate the behaviour of the determinisation algorithm with regard to the tree pattern matching PDA (not only the one described in this paper), and if it is possible to adjust the PDAs in such a way that the determinisation algorithm would provide smaller results. Finally, the don't cares may not be independent: e.g. it is possible to say that two subtrees that are covered by two don't cares must in fact be equivalent. Would it be possible to extend the tree pattern matching algorithm to understand such constraint?

4.1 Acknowledgements

The authors wish to thank the anonymous referees for their detailed reviews and helpful comments.

References

1. R. Alur and P. Madhusudan: *Visibly pushdown languages*, in Proceedings of the thirty-sixth Annual ACM Symposium on Theory of Computing, New York, NY, 2004, ACM Press, pp. 202–211.
2. M. Crochemore and W. Rytter: *Jewels of Stringology*, World Scientific Publishing, Hong-Kong, 2002, 310 pages.
3. M. J. Fischer and M. S. Paterson: *String matching and other products*, in Complexity of Computation, R. M. Karp, ed., vol. 7, SIAM-AMS Proceedings, 1974, pp. 113–125.
4. T. Flouri, J. Janoušek, and B. Melichar: *Tree pattern matching by deterministic pushdown automata*, in Proceedings of the International Multiconference on Computer Science and Information Technology, Workshop on Advances in Programming Languages, M. Ganzha and M. Paprzycki, eds., vol. 4, IEEE Computer Society Press, 2009, pp. 659–666.
5. T. Flouri, J. Janoušek, and B. Melichar: *Subtree matching by pushdown automata*. Computer Science and Information Systems, 7(2) Apr. 2010.
6. J. Holub: *Simulation of Nondeterministic Finite Automata in Pattern Matching*, PhD thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic, 2000.
7. J. Janoušek: *Arbology: Algorithms on trees and pushdown automata*, habilitation thesis, Brno University of Technology, 2010, submitted.
8. 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, 2009, pp. 160–172.
9. J. Janoušek and B. Melichar: *On regular tree languages and deterministic pushdown automata*. Acta Inform., 46(7) Nov. 2009, pp. 533–547.
10. J. Stoklasa, J. Janoušek, and B. Melichar: *Subtree pushdown automata for trees in bar notation*, 2010, London Stringology Days 2010, London.
11. J. Stoklasa, J. Janoušek, and B. Melichar: *Subtree pushdown automata for trees in bar notation*, 2010, submitted to J. Discret. Algorithms.
12. J. Žďárek: *Two-dimensional Pattern Matching Using Automata Approach*, PhD thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic, 2010, submitted, `http://www.stringology.org/papers/Zdarek-PhD_thesis-2010.pdf`.
13. K. Wagner and G. Wechsung: *Computational Complexity*, Springer-Verlag, Berlin, 2001.