Theoretical Computer Science 562 (2015) 590–605

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Postfix automata

Maohua Jing [a,b,*], Yixian Yang [b,c], Ning Lu [a], Wenbo Shi [a], Changyong Yu [a]

[a] School of Computer and Communication Engineering, Northeastern University at Qinhuangdao, China
[b] College of Information Science and Engineering, Northeastern University, China
[c] Information Security Center, Beijing University of Posts and Telecommunications, China

ARTICLE INFO

Article history:
Received 30 January 2013
Received in revised form 17 March 2014
Accepted 31 October 2014
Available online 6 November 2014
Communicated by G. Ausiello

Keywords:
Regular expression
Finite automata
Follow automata
Thompson's automata

ABSTRACT

We introduce a notion of *postfix automata* and apply it to finite automata constructions. We give a compact method for constructing smaller nondeterministic finite automata (NFAs) from given regular expressions. There are four steps in our method. First, we convert one or more given regular expressions into postfix form, better known as *Reverse Polish* notation, and call it a *postfix regular expression*. Second, we construct a syntax tree for the postfix regular expression. Next, we code the syntax tree using a specific algorithm and finally, we construct an NFA based on the coded syntax tree. Using our method we can obtain smaller NFAs, called *postfix automata*, with only one starting state and one final state. In this paper, we prove several notions and present a classical example to compare the size of a postfix automaton with the NFAs created by several classical construction methods. We also derive the efficiency in reducing the size of the NFA through general theoretical analysis.

The regular expression is an important notation for specific patterns. Owing to its expressive power and flexibility in describing useful patterns [1], regular expression matching technology based on finite automata is widely used in networks and information processing, including applications for network real-time processing, protocol analysis, intrusion detection systems, intrusion prevention systems, deep packet inspection systems, and virus detection systems like Snort [2], Linux L7-filter [3], and Bro [4]. Regular expressions are replacing explicit string patterns as the method of choice for describing patterns. However, with the increasing scale and number of regular expressions in a practical system, it is challenging to achieve good performance for pattern matching based on regular expressions. For example, the number of signatures in Snort has grown from 3166 in 2003 to 15,047 in 2009 and the pattern matching routines in Snort account for up to 70% of the total execution time with 80% of the instructions executed on real traces [5].

Implementation of regular expression matching is done using finite automata, including non-deterministic finite automata (NFAs) and deterministic finite automata (DFAs). Storage space of automata is jointly determined by the number of states and transitions between states. A key issue is that the size of the automaton obtained from a regular expression is large, where size is defined as the number of states and transition arcs between states. The size of an automaton is crucial for the efficiency of the algorithms using it; e.g., for pattern matching based on regular expressions, size directly affects both time and space efficiency.

NFAs and DFAs have their own advantages and disadvantages in regular expression matching. NFAs can provide an exponentially more succinct description than DFAs but equivalence, inclusion, and universality are computationally hard for NFAs, while many of these problems can be solved in polynomial time for DFAs. The processing complexity for each character in the input is $O(1)$ in a DFA, but $O(n^2)$ for an NFA if all n states are active at the same time. The key feature

* Corresponding author at: School of Computer and Communication Engineering, Northeastern University at Qinhuangdao, China.

http://dx.doi.org/10.1016/j.tcs.2014.10.050
0304-3975/

of a DFA is that there is only one active state at any time; but converting an NFA into a DFA may generate $O(\Sigma^n)$ states. The size of a DFA, obtained from a regular expression, can increase exponentially; the DFA of a regular expressions with thousands of patterns yields tens of thousands of states, which means memory consumption of thousands of megabytes. Another problem is that a minimal NFA is hard to compute [6]. How to use the matching efficiency of a DFA and the storage efficiency of an NFA to realize matching is always a pursued goal in the field of regular expression matching.

Given an n-state NFA, one can always construct a semantically equivalent DFA with at most 2^n states [7]. Therefore, NFAs can offer exponential reductions in space compared with DFAs. Since DFAs obtained from regular expressions can be exponentially larger in size, they can in many cases be represented by NFAs. In fact, NFAs offer a number of advantages over DFAs. For example, NFAs are very useful in parallel processing because they can process an input character in multiple branches and may output multiple acceptance states for the input compared with DFAs [8]. Moreover, since NFAs are more compact than DFAs, combined with their parallel processing they are more space efficient [9].

Motivated by the above advantages of NFAs, we propose a method for constructing smaller automata from given regular expressions. There are two related problems in converting a given regular expression into an NFA, namely

- the translation of regular expressions into ε-free NFAs (that is, without ε-transitions); and
- the translation of regular expressions into ε-NFAs (that is, with ε-transitions).

We construct a smaller NFA, with or without ε-transitions, from the given regular expressions based on a specific kind of binary tree: a kind of coded syntax parse tree for the given regular expressions. There are two main algorithms in our method: one is coding the syntax parse tree and the other is constructing the NFA based on the coded binary tree. In addition, preprocessing is carried out in our method before constructing the syntax parse tree.

Using this method, we can obtain smaller NFAs. Here we do not focus on constructing ε-free NFAs; we merely aim to obtain a smaller size by reducing the number of ε-transition arcs in the final NFA as much as possible. In other words, we allow ε-transition arcs in the NFA, but try to reduce the number thereof. Furthermore, we give a typical example showing that we can obtain much smaller NFAs using our method and determine the efficiency in reducing the size by general theoretical analysis.

The most important benefit of postfix automata is that obtaining the state space by coding the syntax parse tree is very compact. Another benefit is obtaining the transition arcs between states based on the coded syntax tree. In addition, a single NFA can be obtained simply from a given group of regular expressions using our method.

The paper is organized as follows. Section 1 introduces related work about this problem, while Section 2 discusses some of the basic concepts needed. Section 3 describes the method for constructing postfix automata in detail. Section 4 compares our postfix automata with several well known classical automata construction methods using the same example to show the effectiveness of the algorithm. We also prove the efficiency thereof by theoretical analysis. Finally, Section 5 concludes the paper.

1. Related work

Several methods have been developed to construct automata from given regular expressions. Some of the better known automata are Thompson's automaton [10], the follow automaton of Ilie and Yu [11,12], the position automaton of Glushkov [13], and the partial derivative automaton of Antimirov [14].

The first and most well-known method for constructing automata is Thompson's automata, which includes ε-transitions. It is a classical solution and obtains linear size ε-NFAs. The cost of this construction is linear in the number of symbols and operators [15]. Based on Thompson's algorithm, Sippu and Soisalon-Soininen [16] proposed a method for constructing an ε-free NFA, which is slightly smaller than the Thompson automaton.

Another well-known method for NFA construction from regular expressions is the position automaton, proposed independently by Glushkov, McNaughton and Yamada. The position automaton is an NFA without ε-transitions. The size thereof falls between linear and quadratic, while the number of states in the automaton is the number of occurrences of the symbols in the regular expressions plus one (the start state). While Thompson's automaton has linear size, the position automaton has quadratic size in the worst case. Several methods have been proposed to obtain this automaton in $O(n^2)$ time complexity [17,18].

Hromkovic et al. [19] proposed a construction method that produces an $O(n(\log n)^2)$ size NFA, called the common follow sets automaton, in $O(n(\log n)^2)$ time using the algorithm of Hagenah and Muscholl [20].

In 2003, Ilie and Yu proposed a new method for constructing small NFAs from regular expressions, called follow automata, based on two algorithms that can create NFAs with size at most $\frac{3}{2}|r| + \frac{5}{2}$ and lower bound $\frac{4}{3}|r| + \frac{5}{2}$, where r is a given regular expression and $|r|$ is the length of the string. The first algorithm constructs NFAs with ε-transitions, which are smaller than all the other ε-free NFAs obtained by similar constructions such as Thompson's automaton.

Our aim is to construct automata that are not only "reasonably" small in size, but can also be constructed simply and efficiently in practice. Our basic idea resembles a postfix expression in arithmetic, which can be effectively computed using a stack. We introduce the notion of a postfix regular expression for a given regular expression and apply it to finite automata constructions. We also introduce a tree representation based on postfix regular expressions, called a regular tree, which can accurately describe the logical recursive and nested relationships and calculation sequence of regular expressions. We can