Information Processing Letters 116 (2016) 750–756

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Compact representations of automata for regular expression matching

Meng Zhang [a,*], Yi Zhang [b], Chen Hou [a]

[a] College of Computer Science and Technology, Jilin University, Changchun, China
[b] Department of Computer Science, Jilin Business and Technology College, Changchun, China

ARTICLE INFO

Article history:
Received 27 September 2015
Received in revised form 19 January 2016
Accepted 9 July 2016
Available online 26 July 2016
Communicated by M. Chrobak

Keywords:
Algorithms
String matching
Regular expressions

ABSTRACT

Glushkov's nondeterministic finite automaton leads to efficient regular expression matching. But it is memory greedy for long regular expressions. We develop space efficient representations for the deterministic finite automata obtained from Glushkov automata. The approach reduces the space of the DFA from $O(m2^m)$ bits to $O(m2^k)$ bits where k is the number of strings in the regular expression, m is the number of characters excluding operators. The average space usage is $O(m(1+1/\sigma)^k)$ bits where σ is the size of the alphabet. The state transition function runs in constant time when the length of a word is not less than m. Experiments show that our method is as efficient as the previous method and uses less space.

1. Introduction

High performance regular expression matching is one of the central problems for applications such as network security [4,20] and bioinformatics [14]. Recently, much research focuses on optimizing regular expression matching by reducing the space usage [20], increasing the memory bandwidth [15] and enhancing the performance either by algorithm design or by hardware [4,6–9]. Throughout the paper, we denote the length of our regular expression by m (without counting the operators), the length of the text by n, and the alphabet by Σ.

Nondeterministic finite automaton (NFA) is a major technique for regular expression matching. Thompson's classic NFA construction [21,22] builds an NFA of $O(m)$ states. It takes $O(mn)$ time to search for occurrences of the regular expression in a text using the NFA. Myers [18] reduced the time complexity of Thompson's algorithm and gave an $O(mn/\log n + (n+m)\log n)$ time solution on a $\log n$-bit uniform RAM. Bille and Thorup [8] improved the time complexity of Thompson's algorithm by a $\log^{1.5} n/\log\log n$ factor. Recently, Bille and Thorup [9] presented a regular expression matching algorithm that processes each character in $O(k\frac{\log w}{w} + \log k)$ time using $O(m)$ space on a RAM with w-bit words, where k is the number of strings in the regular expression.

An alternative NFA construction approach for regular expressions was presented by Glushkov [13,5]. The Glushkov automaton of a regular expression is an ε-transition free NFA of $m+1$ states. The corresponding DFA uses much less space than the worst case using Thompson's NFA. Navarro and Raffinot [19] (NR for short) presented a bit-parallel implementation of the DFA obtained from Glushkov's NFA using $O(m2^m)$ bits. The approach is very efficient for short regular expressions. But for expressions with large m, the space usage becomes too large. The NR-method uses the table splitting technique to reduce the space at the cost of increasing the searching time. Champarnaud et al. [11] improved the space of the NR-method. But the worst case space is still $O(m2^m)$ bits.

* Corresponding author.
E-mail addresses: zhangmeng@jlu.edu.cn (M. Zhang), whdzy2000@vip.sina.com (Y. Zhang), 445962340@qq.com (C. Hou).

http://dx.doi.org/10.1016/j.ipl.2016.07.003
0020-0190/

1.1. Our results

We present an implementation of Glushkov's NFA that uses less space than the NR-method while is as efficient as the NR-method. We explore properties of Glushkov automata and develop a new state transition method. The idea is to classify the states into two classes. We use Shift-And to change the set of active states of the first class and use minimal perfect hash functions (MPHFs) [12] to change the set of active states of the second class. The space usage is $O(m\Sigma_{a\in\Sigma}2^{e(a)})$ bits where $e(a)$ is the number of strings ending with a in the regular expression. The worst case space is $O(m2^k)$ bits. If we assume a uniform frequency distribution of individual characters in regular expressions, the average space usage is $m(1+1/\sigma)^k$ bits. The time for state transition is constant when $m \le w$ and $O(\lceil m/w \rceil)$ otherwise.

2. Background

Regular expressions are constructed from symbols via concatenation, union (|), and Kleene star ($*$) operations. The set of regular expressions over an alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$ is defined as follows. (i) For $a \in \Sigma$ or $a = \varepsilon$, a is a regular expression. (ii) For regular expressions R and S, $(R)|(S)$, $(R)(S)$, and $(R)^*$ are regular expressions where $(R)(S)$ denotes the concatenation of R and S. The language determined by a regular expression R is denoted by $L(R)$. The set $Pos(R) = \{1, \ldots, m\}$ is the set of positions in R not counting operators. Denote by σ_y the indexed character of R that is at position $y \in Pos(R)$, and define $\sigma_0 = \varepsilon$.

The model of computation we adopt is a random access machine with w-bit words, and the instruction set includes unit-cost addition "+", subtraction "−", multiplication "×", and bit-wise Boolean operations including `or` "|", `and` "&", bit complementation "¬", and bit shift "<<" (">>").

We denote the substring $t[i]\ldots t[j]$ $(i \le j)$ of a string $t = t[1]t[2]\ldots t[n]$ by $t[i, j]$. A *bitvector* is a sequence of bits. For a bitvector X and an integer $c > 0$, X^c denotes the repetition of X for c times.

2.1. Glushkov's NFA

We give a brief introduction to Glushkov's NFA following the notions in [19]. More details can be found in [13,5,19]. The Glushkov NFA for a regular expression RE recognizes $L(RE)$. Each position of RE corresponds to a state of the Glushkov NFA (the initial state corresponds to 0). The $m+1$ states are labeled from 0 to m. For a state $x \in \{0, \ldots, m\}$, define $Follow(x) = \{y \in Pos(RE), \exists u, v \in \Sigma^*, u\sigma_x\sigma_y v \in L(RE)\}$, which contains all the states reached by going the transitions from x. Define $Last = \{x \in Pos(RE), \exists u \in \Sigma^*, u\sigma_x \in L(RE)\}$, which is the set of final states. Fig. 1 is an example of the Glushkov NFA.

2.2. Navarro and Raffinot's implementation of Glushkov NFAs

Navarro and Raffinot [19] presented a compact representation of the DFA corresponding to a Glushkov NFA.

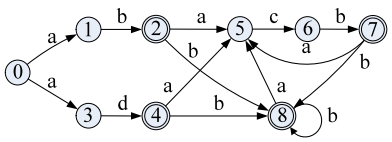

Fig. 1. The Glushkov automaton built on the regular expression $(ab|ad)((acb|b)*)$. State 0 is the initial state. Double-circled states are final.

The sets of NFA states are represented by bit vectors of $m+1$ bits; if state i belongs to the set, the i-th bit is 1 otherwise 0. In scanning a text, the set of current active Glushkov NFA states represents a DFA state; we denote the set of active states by a bit vector D. In the rest of the paper, we will use sets of NFA states, position sets and bit vectors interchangeably.

The NR-method makes state transitions using two tables: the first one is $B : \Sigma \to 2^{m+1}$, such that $B[a]$ is the set of states whose incoming transitions are labeled by a. The second is a table $T : 2^{m+1} \to 2^{m+1}$. Given the current active state set D, define $T[D]$ as the set of states that can be reached from states in D, that is, $T[D] = \bigcup_{i\in D} Follow(i)$. The state transition function of the DFA obtained from the Glushkov's NFA is as follows.

$$\delta(D, a) = T[D]\&B[a]. \tag{1}$$

Table T uses $(m+1)2^{m+1}$ bits. Table B uses $(m+1)\sigma$ bits. The number of total bits required is $(m+1)(2^{m+1}+\sigma)$. By a table splitting approach, the space can be reduced to $O(t2^{m/t}m)$ bits while the searching time increases to $O(tn)$, where t is the number of tables with $2^{m/t}$ entries split from T. The NR-method can process character classes by treating a character class as a single letter.

Mapping from states to bits. In Glushkov's construction, each non-operator character (or position) in the regular expression uniquely corresponds to a state. In the NR method, the i-th non-operator character is mapped to state i, and state i is mapped to the i-th bit in D. Actually, the transition function in (1) works with any bijection from states to bits, as long as all bitvectors in B and T use the same bijection as D.

2.3. Minimal perfect hash function

A perfect hash function (PHF) maps the keys in set S to unique values in the range $[0, v-1]$. A minimal perfect hash function (MPHF) is a PHF with $|S| = v$. The recent result of MPHF [10] on a unit-cost RAM obtains a space usage of approximately $2.62n + o(n)$ bits where n is the number of keys. The evaluation of this MPHF requires constant time for keys with constant number of words. The construction algorithm of the MPHF runs in $O(n)$ time. In this paper, we will use these minimal perfect hash functions without going into the details.

3. Heterogeneous transition function

In this section, we present space economical representations of the DFA obtained from Glushkov's NFA. We employ the Shift-And approach and the hash function.