Information Processing Letters 143 (2019) 56–60

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A faster algorithm for finding shortest substring matches of a regular expression

Hiroaki Yamamoto

Department of Electrical & Computer Engineering, Shinshu University, 4-17-1 Wakasato, Nagano-shi 380-8553, Japan

ARTICLE INFO

Article history:
Received 4 December 2015
Received in revised form 7 September 2018
Accepted 2 December 2018
Available online 5 December 2018
Communicated by Andrzej Tarlecki

Keywords:
Design of algorithms
Regular expression
Matching algorithm
Shortest substring
Finite automaton

ABSTRACT

Consider a regular expression r of length m and a text string T of length n over an alphabet Σ. Then, the *RE shortest substring search problem* is to find all shortest substrings of T matching r. The previous algorithm proposed by Clarke and Cormack uses an ε-free nondeterministic finite automaton (NFA) and runs in $O(ksn)$ time and $O(s)$ space, where k is the maximum number of outgoing transitions for any state and symbol, and s is the number of states. Generally, an ε-free NFA obtained from a regular expression has $s = O(m)$ and $k = O(m)$; thus the algorithm takes $O(m^2n)$ time and $O(m)$ space. We propose a faster algorithm that runs in $O(mn)$ time and $O(m)$ space. The proposed algorithm is based on a Thompson automaton which is an NFA with ε-transitions.

1. Introduction

The regular expression (RE) matching problem plays an important role in computer science and has been studied extensively [1,2,11,13,15]. The RE matching problem is typically defined as follows. Let r be an RE of length m and let T be a text string of length n over an alphabet Σ. Then the RE matching problem is to determine whether there is a substring y of T such that $y \in L(r)$, where $L(r)$ denotes the language denoted by r. The traditional algorithm using a Thompson automaton [14] solves the RE matching problem in $O(mn)$ time and $O(m)$ space. Furthermore more efficient algorithms have been developed. For example, Myers [11] presented an algorithm that runs in $O(mn/\log n)$ time and space, and Bille and Thorup [1,2] proposed efficient algorithms using bit-parallelism.

As a more general problem, there is an *RE search problem* that finds all substrings of T matching r. We can solve the RE search problem using existing RE matching algorithms. However, it requires significant time because these RE matching algorithms only find the end positions of substrings matching an RE. We need to find the start positions in the RE search problem. Therefore, it is difficult to solve the RE search problem in $O(mn)$ time. Han, Wang, and Wood [6] studied a subclass of REs called prefix-free REs and showed that the RE search problem for prefix-free REs can be solved in $O(mn)$ time and $O(m)$ space.

Clarke and Cormack [4] addressed an RE shortest substring search problem for markup languages such as XML (Extensible Markup Language) and developed an efficient algorithm that makes use of ε-free nondeterministic finite automaton (NFA). Their algorithm runs in $O(ksn)$ time and $O(s)$ space, where k is the maximum number of outgoing transitions for any state and symbol, and s is the number of states of an ε-free NFA obtained from r. If an RE is prefix-free, then the RE shortest substring search problem can be solved in $O(mn)$ time and $O(m)$ space. Han [5] and Han, Wang, and Wood [6] state that this problem can be solved in $O(mn)$ time for any RE if a Thompson automaton (T-NFA) is used because k is at most 2. However, we cannot use the Clarke–Cormack algorithm because a T-NFA is an NFA with ε-transitions. In [5,6] no $O(mn)$ time algorithm using a T-NFA was ever shown. If we di-

E-mail address: yamamoto@cs.shinshu-u.ac.jp.

https://doi.org/10.1016/j.ipl.2018.12.001

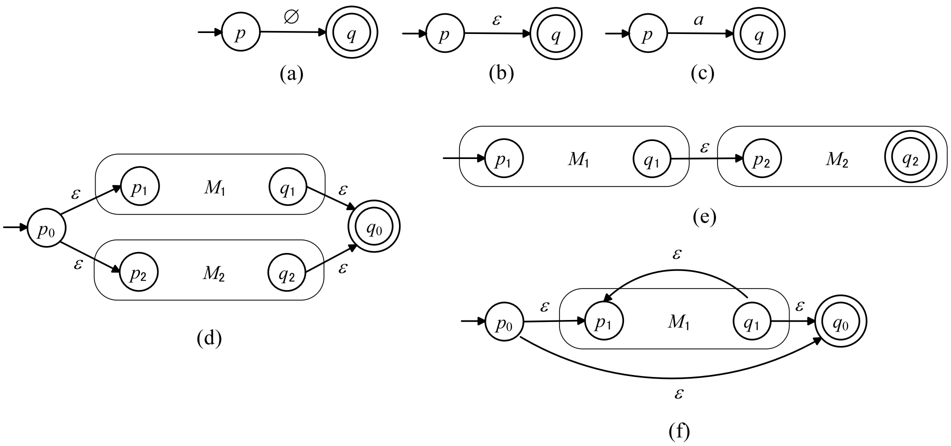

Fig. 1. Construction of Thompson automaton for a regular expression r. (a) $r = \emptyset$, (b) $r = \varepsilon$, (c) $r = a$, (d) union $r = r_1 + r_2$, (e) concatenation $r = r_1 r_2$, (f) star closure $r = r_1^*$.

rectly remove the ε-transitions from a T-NFA, then k can be $\Theta(m)$. A position automaton, an equation automaton, and a follow automaton are widely known as an ε-free NFA obtained from an RE [3,9,16]. However, as seen in [9], there are REs such that the number of transitions is $\Theta(m^2)$ for these automata. Furthermore, Hromkovič, Seibert, and Wilke [7] gave an upper bound of $O(m(\log m)^2)$ and a lower bound of $\Omega(m \log m)$ for the number of transitions of an ε-free NFA obtained from an RE. Lifshits [10] improved the lower bound to $\Omega(m(\log m)^2/\log\log m)$. Thus, it seems to be impossible to achieve $O(mn)$ time and $O(m)$ space by directly applying the Clarke–Cormack algorithm to an ε-free NFA obtained from an RE.

In this paper, we present an algorithm that runs in $O(mn)$ time and $O(m)$ space. The proposed algorithm uses a T-NFA but does not remove the ε-transitions. We achieve $O(mn)$ time and $O(m)$ space by efficiently processing ε-transitions. The remainder of this paper is organized as follows. In Section 2, we give basic definitions of REs and describe the RE shortest substring search problem. In Section 3, we present a T-NFA and describe its basic properties. The proposed algorithm is presented in Section 4.

2. Regular expressions and an RE shortest substring search problem

Here we provide some definitions for REs.

Definition 1. Let Σ be an alphabet. The REs over Σ are defined as follows.

1. $\emptyset$, ε (the empty string), and a ($\in \Sigma$) are REs that denote the empty set, the set $\{\varepsilon\}$, and the set $\{a\}$, respectively.
2. Let r_1 and r_2 be REs denoting the sets R_1 and R_2, respectively. Then $(r_1 + r_2)$, $(r_1 r_2)$, and (r_1^*) are also REs that denote the sets $R_1 \cup R_2$ (union), $R_1 R_2$ (concatenation), and R_1^* (star closure), respectively.

Typically, unnecessary parentheses in an RE are eliminated according to the following preference rules. Star closure has higher preference than concatenation and union, and concatenation has higher preference than union. Here $L(r)$ denotes the language denoted by an RE r. The length of an RE r represents the number of alphabet symbols and operators (union, concatenation, and star) occurring in r.

Let $T = a_1 \cdots a_n$ be a string over Σ. Then, for any $1 \le i \le j \le n$, we define a substring $T[i:j]$ of T as $T[i:j] = a_i \cdots a_j$. Note that the empty string ε is also a *substring* of T. If a substring $T[i:j]$ is not equal to T, then $T[i:j]$ is called a *proper substring* of T.

Definition 2. Let r be an RE. Then, we define the set $\mathcal{P}(r)$ of strings as follows.

$\mathcal{P}(r) = \{x \mid x \in L(r)$ and for any proper substring y of x, $y \notin L(r)\}$.

We define the set $\mathrm{Match}(r, T) = \{(i, j) \mid i \le j, T[i:j] \in \mathcal{P}(r)\}$. From Definition 2, if $\varepsilon \in L(r)$, then $\mathcal{P}(r) = \{\varepsilon\}$; thus, $\mathrm{Match}(r, T) = \emptyset$. Therefore, in this paper, we consider only REs r such that $\varepsilon \notin L(r)$.

Definition 3. For any RE r and any string T, the *RE shortest substring search problem* is to find $\mathrm{Match}(r, T)$.

Consider the following example. Let $r = ab(a+b)^*ba$ be an RE over $\{a, b\}$ and let $T = aababaaaabaaaba$. Then, all substrings matching r consist of $T[2:6] = ababa$, $T[2:11] = ababaaaaba$, $T[2:15] = ababaaaabaaaba$, $T[4:11] = abaaaaba$, $T[4:15] = abaaaabaaaba$, and $T[9:15] = abaaaba$. Thus $\mathrm{Match}(r, T) = \{(2, 6), (4, 11), (9, 15)\}$.

3. Thompson automata

T-NFAs are recursively constructed according to the definition of REs, and the construction algorithm for T-NFAs is widely known (for example, see [8]). Fig. 1 outlines a construction for each operator, where M_1 and M_2 denote T-NFAs for REs r_1 and r_2, respectively. As seen in Fig. 1, a star closure generates a transition to a previous state,

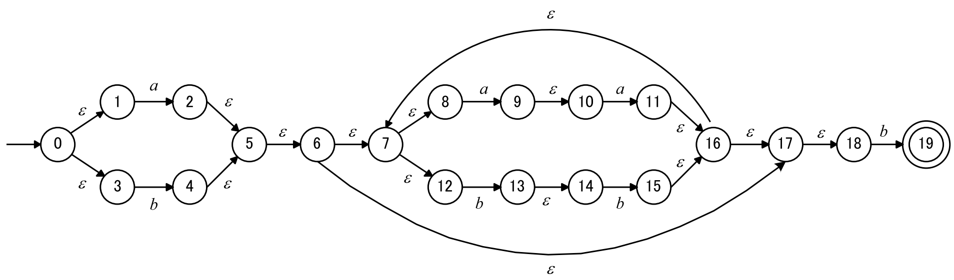

Fig. 2. T-NFA for $r = (a+b)(aa+bb)^*b$.

i.e., a transition from state q_1 to state p_1. We refer to this as a *back transition*.

Let $M = (Q, \Sigma, \delta, q_0, q_f)$ be a T-NFA for an RE r of length m, where Q is a set of states, Σ is an alphabet, δ is a transition function from $Q \times (\Sigma \cup \{\varepsilon\})$ to 2^Q, q_0 is the initial state and q_f is the final state. Thus, a T-NFA has one initial state and one final state. In addition, a T-NFA has the following properties.

Property 1 A T-NFA consists of at most $2m$ states and $4m$ transitions.

Property 2 For any state $q \in Q$, all incoming transitions of q have either the empty string ε or an alphabet symbol in Σ. We call state q an *ε-state* if all incoming transitions have ε, and call q a symbol state (*s-state*) if all incoming transitions have an alphabet symbol.

Property 3 For any state $q \in Q - \{q_0\}$, if q is an ε-state then the number of incoming transitions of q is 1 or 2, and if q is an s-state then the number of incoming transitions of q is just 1.

We define $\tilde{Q} = \{q \in Q \mid q \text{ is an } s\text{-state}\} \bigcup \{q_0\}$. For any state $q \in Q$, if q has two incoming transitions, then q is called a *junction state*. Note that these two incoming transitions are labeled ε. Thus all junction states are ε-states. We call a sequence of transitions of M a *path*. In particular, we call a sequence consisting of only ε-transitions an *ε-path*. Here, for any state $q \in Q$, we define that there is an ε-path from q to q. By removing back transitions from a T-NFA, the T-NFA can be viewed as a directed acyclic graph. The topological order of states of a T-NFA is a linear order obtained from the corresponding directed acyclic graph.

Here, we provide an example of a T-NFA. Consider an RE $r = (a+b)(aa+bb)^*b$ over $\Sigma = \{a, b\}$. Then, Fig. 2 shows the T-NFA of r constructed by the recursive construction given in Fig. 1. $\tilde{Q} = \{0, 2, 4, 9, 11, 13, 15, 19\}$ and the junction states are $5, 7, 16$ and 17. Each state is numbered in topological order. When constructing a T-NFA according to Fig. 1, the following proposition holds.

Proposition 1. *For any RE of length m, we can construct the T-NFA in $O(m)$ time and space.*

Furthermore, T-NFAs have the following property. This property was given in [12].

Lemma 1 *(see Lemma 1 in [12]). Let M be a T-NFA. Then, any loop-free path of M has at most one back transition.*

4. Algorithm for the RE shortest substring search problem

The algorithm given by [4] assumes an ε-free NFA; therefore, we must remove ε-transitions from a T-NFA if we use this algorithm. Generally, by removing ε-transitions, the number of outgoing transitions of a state increases to $O(m)$. Thus, if we remove ε-transitions and then use the algorithm of [4], the time complexity becomes $O(m^2n)$. We will show an $O(mn)$ time algorithm by computing ε-transitions efficiently. In other words, since a T-NFA has a simple structure, we can compute ε-transitions efficiently by using the structure. Our algorithm $\mathsf{REShortSearch}(r, T)$ is shown in Algorithm 1. In Algorithm 1, the array $\mathsf{Start}[q]$ is used to store the start position of the shortest substring for which M can reach state q from q_0. Then, the following theorem holds.

Algorithm 1 $\mathsf{REShortSearch}(r, T)$.

Input: an RE r and a string $T = a_1 \cdots a_n$, where $a_i \in \Sigma$.
Output: $\mathsf{Match}(r, T)$.

```
Generate a T-NFA M = (Q, Σ, δ, q_0, q_f) from r.
Sort Q in topological order.
for all q ∈ Q do
    Start[q] ← 0
end for
Compute Q_f = {q | q is an s-state such that there is an ε-path from
  q to q_f}
for i = 1 to n do
    UpdateStart(M, Start, i)
    pos = max_{q∈Q_f} Start[q]
    if pos > 0 then
        Output (pos, i)
        for all q ∈ Q do
            if Start[q] ≤ pos then
                Start[q] ← 0
            end if
        end for
    end if
end for
```

Theorem 1. *Let r be an RE of length m and T be a string of length n. Then,* $\mathsf{REShortSearch}(r, T)$ *can compute* $\mathsf{Match}(r, T)$ *in $O(mn)$ time and $O(m)$ space.*

Let us prove the theorem. The key point to achieve $O(mn)$ time is the procedure $\mathsf{UpdateStart}(M, \mathsf{Start}, i)$, which is given in Algorithm 2. First, we give the following lemma.

Algorithm 2 Procedure UpdateStart(M, Start, i).

Input: T-NFA M, Start and an input position i.

1: $loop \leftarrow 2$
2: **for all** $q \in Q$ **do**
3: Next[q] $\leftarrow 0$
4: **end for**
5: Start[q_0] $\leftarrow i$
6: **while** $loop \neq 0$ **do**
7: **for all** ε-states $q \in Q - \{q_0\}$ in topological order **do**
8: **if** q is a junction state with $q \in \delta(p_1, \varepsilon)$ and $q \in \delta(p_2, \varepsilon)$ **then**
9: Start[q] $\leftarrow$ max{Start[p_1], Start[p_2]}
10: **else**
11: Start[q] $\leftarrow$ Start[p], where $q \in \delta(p, \varepsilon)$
12: **end if**
13: **end for**
14: $loop \leftarrow loop - 1$
15: **end while**
16: **for all** s-states q with a transition $\delta(p, a_i) = \{q\}$ by a_i **do**
17: **if** Start[p] $>$ Next[q] **then**
18: Next[q] $\leftarrow$ Start[p]
19: **end if**
20: **end for**
21: **for all** $q \in Q$ **do**
22: Start[q] $\leftarrow$ Next[q]
23: **end for**

Lemma 2. *The following properties hold for any $i \geq 1$.*

1. UpdateStart(M, Start, i) *runs in $O(m)$ time and $O(m)$ space.*
2. *After executing* UpdateStart(M, Start, i), *for any state $q \in \tilde{Q}$, $i' =$ Start[q] > 0 if and only if the following (a) and (b) hold:*
   (a) *M can reach state q from q_0 by $T[i' : i]$,*
   (b) *for any i_1 ($i' < i_1 \leq i$), M cannot reach state q from q_0 by $T[i_1 : i]$.*

Proof. Each state of M is checked at most once in the while-loop at lines 6–15. Therefore UpdateStart(M, Start, i) runs in $O(m)$ time and $O(m)$ space because M has at most $2m$ states and the while-loop is only performed twice. Thus, property 1 has been proved.

Next, let us prove property 2 using induction on i. In UpdateStart(M, Start, i), note that Next[q] is used as a temporary array to compute Start[q]. Next[q] is initially set to 0. If there is a substring $T[i' : i]$ on which M can go to q from q_0, then Next[q] is set to the maximum position among these i'. Finally, Next[q] is copied to Start[q] at lines 21–23.

Now let us prove the base case $i = 1$. The procedure UpdateStart(M, Start, i) first sets Start[q_0] to 1 at line 5. For any $p \in Q$, if there is an ε-path from q_0 to p, then Start[p] is set to 1 in the while-loop because each state is processed in topological order. Let q be an s-state with $\delta(p, \sigma) = \{q\}$. Then, if there is an ε-path from q_0 to p, Start[p] is set to 1 prior to processing q. Therefore, Next[q] is set to 1 at lines 16–20 if $\sigma = a_1$. If $\sigma \neq a_1$, then Next[q] $= 0$. Finally, Next[q] is copied to Start[q] at line 22. Thus, if Start[q] > 0, then Start[q] $= 1$ and M can go to q from q_0 by $T[1 : 1]$. Conversely, if M can go to q from q_0 by $T[1 : 1]$, then Start[q] is set to 1. Thus, property 2 holds for the base case.

Induction step. Let $i \geq 1$. Assume that property 2 holds for any position less than $i + 1$. Then, we prove case $i + 1$. Suppose that UpdateStart(M, Start, i) have finished. Then, for any state $p \in Q$ which has a transition to an s-state q, i.e., $\delta(p, \sigma) = \{q\}$ for an alphabet symbol σ, the following claim holds.

Claim. *Let $q_1, \ldots, q_t$ be all states such that they are in $\tilde{Q}$ and there is an ε-path to p. Then,* Start[p] *is set to the maximum value of* Start[q_j] ($1 \leq j \leq t$) *after executing UpdateStart*(M, Start, $i + 1$).

Proof of Claim. For any q_j ($1 \leq j \leq t$), there is always a loop-free ε-path P_j from q_j to p because if an ε-path has a loop, then we can remove the loop. By Lemma 1, P_j has at most one back transition. If P_j does not have a back transition, then the value of Start[q_j] is transmitted to Start[p] because each state is processed in topological order. Now we consider the case where P_j has one back transition. Suppose that $P_j = p_1 \cdots p_l$, where $p_1 = q_j$, $p_l = p$, and a transition p_e ($1 \leq e \leq l - 1$) to p_{e+1} in P_j is a back transition. Then, since each state is processed in topological order, state p_{e+1} is processed prior to p_e. Thus, in the first repetition of the while-loop, Start[p_{e+1}] is computed using the old value of Start[p_e]. However, in the second repetition, Start[p_{e+1}] can be computed using the updated value. Therefore, the value of Start[q_j] is correctly transmitted to Start[p] even if P_j has a back transition. Since a junction state takes the maximum value of two states at line 9, Next[q] is set to the maximum value among Start[q_1], ..., Start[q_t]. Thus the claim holds. □

By the inductive assumption, if Start[q_j] > 0, then M can move from q_0 to q_j on T[Start[q_j] $: i$]. Recall that a state q is an s-state with a transition $\delta(p, \sigma) = \{q\}$. Thus, if $\sigma = a_{i+1}$ and Start[p] $>$ Next[q], then Start[p] is copied to Next[q] at line 18. If $\sigma \neq a_{i+1}$, then Next[q] $= 0$ because Next[q] is first set to 0. Finally Next[q] is copied to Start[q] for all states q. Thus, by the claim, if Start[q] > 0, then M can reach q from q_0 by T[Start[q] $: i + 1$], and, for any i_1 (Start[q] $< i_1 \leq i + 1$), M cannot reach q from q_0 by $T[i_1 : i + 1]$. Furthermore, it is also clear that if (a) and (b) holds, then Start[q] > 0. Thus, property 2 holds for the case $i + 1$, and Lemma 2 has been proved. □

We can now prove Theorem 1.

Proof of Theorem 1. First, let us prove the time and space complexities. The construction of a T-NFA and the topological sort of states can be performed in $O(m)$ time and space. The procedure UpdateStart(M, Start, i) of line 8 is executed n times. Therefore, from Lemma 2, lines 7–18 takes $O(mn)$ time and $O(m)$ space. Thus REShortSearch(r, T) can be executed in $O(mn)$ time and $O(m)$ space.

Let us show that REShortSearch(r, T) computes Match(r, T). First, we show that if REShortSearch(r, T) outputs a pair (pos, i) then (pos, i) $\in$ Match(r, T). Let $q \in \tilde{Q}$. From Lemma 2, after executing UpdateStart(M, Start, i) for each $i \geq 1$, if Start[q] has position $i' > 0$, then it holds that M can reach q from q_0 by $T[i' : i]$ and for any i_1 ($i' < i_1 \leq i$), M cannot reach q from q_0 by $T[i_1 : i]$. Here, if $q \in Q_f$ then $T[i' : i]$ is accepted by M. The variable pos

is set to the maximum value among Start[q] with $q \in Q_f$ at line 9; therefore a substring $T[pos : i]$ is accepted by M, and, for any i_1 ($pos < i_1 \leq i$), $T[i_1 : i]$ is not accepted by M. Thus, we have $(pos, i) \in \mathrm{Match}(r, T)$.

Next, we show the converse relation. Assume that $(i', i) \in \mathrm{Match}(r, T)$. By the definition of $\mathrm{Match}(r, T)$, a substring $T[i', i]$ is accepted by M; however, for any i_1 ($i' < i_1 \leq i$), $T[i_1 : i]$ is not accepted by M. From Lemma 2, after executing UpdateStart(M, Start, i), there is a state $q \in Q_f$ such that Start[q] $= i'$. Furthermore, i' must be the maximum value among Start[q] with $q \in Q_f$. Thus, REShortSearch(r, T) sets pos to i' at line 9 and outputs a pair (pos, i). Therefore, REShortSearch(r, T) correctly computes $\mathrm{Match}(r, T)$. $\square$

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number JP17K00183.

References

[1] P. Bille, New algorithms for regular expression matching, in: Proc. of ICALP 2006, in: Lect. Notes Comput. Sci., vol. 4501, 2006, pp. 643–654.
[2] P. Bille, M. Thorup, Regular expression matching with multi-strings and intervals, in: Proc. of SODA 2011, 2011, pp. 1297–1307.
[3] A. Brüggemann-Klein, Regular expressions into finite automata, Theor. Comput. Sci. 120 (1993) 197–213.
[4] Charles L.A. Clarke, Gordon V. Cormack, On the use of regular expressions for searching text, ACM Trans. Program. Lang. Syst. (TOPLAS) 19 (3) (1997) 413–426.
[5] Yo-Sub Han, On the linear number of matching substrings, J. Univers. Comput. Sci. 16 (5) (2010) 715–728.
[6] Yo-Sub Han, Y. Wang, D. Wood, Prefix-free regular languages and pattern matching, Theor. Comput. Sci. 389 (2007) 307–317.
[7] J. Hromkovič, S. Seibert, T. Wilke, Translating regular expressions into small ε-free nondeterministic finite automata, J. Comput. Syst. Sci. 62 (4) (2001) 565–588.
[8] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory Language and Computation, Addison Wesley, Reading, Mass, 1979.
[9] L. Ilie, S. Yu, Follow automata, Inf. Comput. 186 (2003) 140–162.
[10] Y. Lifshits, A lower bound on the size of ε-free NFA corresponding to a regular expression, Inf. Process. Lett. 85 (3) (2003) 293–299.
[11] G. Myers, A four Russians algorithm for regular expression pattern matching, J. ACM 39 (4) (1992) 430–448.
[12] E. Myers, W. Miller, Approximate matching of regular expressions, Bull. Math. Biol. 51 (1) (1989) 5–37.
[13] G. Navarro, M. Raffinot, New techniques for regular expression searching, Algorithmica 41 (2004) 89–116.
[14] K. Thompson, Regular expression search algorithm, Commun. ACM 11 (6) (1968) 419–422.
[15] H. Yamamoto, Regular expression matching algorithms using dual position automata, J. Comb. Math. Comb. Comput. 71 (2009) 103–125.
[16] H. Yamamoto, T. Miyazaki, M. Okamoto, Bit-parallel algorithms for translating regular expressions into NFAs, IEICE Trans. Inf. Syst. E90-D (2) (2007) 418–427.