Regular Expression Matching Algorithms using Dual Position Automata *

Hiroaki Yamamoto
Department of Information Engineering, Shinshu University,
4-17-1 Wakasato, Nagano-shi, 380-8553 Japan.
yamamoto@cs.shinshu-u.ac.jp

Abstract

This paper introduces an automaton model called *a dual position automaton* (a dual PA), and then gives a bit-parallel algorithm for generating a dual PA from a regular expression (RE). For any RE r over an alphabet Σ, our translation algorithm generates a dual PA consisting of $\tilde{m}(\tilde{m}+1)$ bits in $O(\tilde{m}\lceil\tilde{m}/w\rceil)$ time and space, where w is the length of a computer word, $\tilde{m} = \sum_{a\in\Sigma} m_a$ and m_a is the number of occurrences of an alphabet symbol a in r. Furthermore, we give a method to construct a compact DFA representation from a dual PA. This DFA representation requires only $(\tilde{m}+1)\sum_{a\in\Sigma} 2^{m_a}$ bits. Finally, we show RE matching algorithms using such a DFA representation.

1 Introduction

Regular expressions (REs) pattern matching problems play an important role in the field of computer science, computational biology and so on. For this reason, RE pattern matching algorithms have intensively been studied [1, 3, 4, 12, 13, 14, 15, 18, 19]. We are here concerned with the following RE pattern matching problem: Let r be an RE and let x be a string (which is also called *a text*). Then the RE matching problem is to decide whether or not there is a substring y of x such that $y \in L(r)$, where $L(r)$ denotes the

*This research has been supported in part of Grant-in-Aid for Scientific Research, Ministry of Education, Culture, Sports, Science and Technology, Japan.

language generated by r. In general, RE matching algorithms are divided into two parts, a preprocessing part and a matching part. The preprocessing part translates a given RE into a finite automaton and the matching part does a matching job using the generated finite automaton. This time, many algorithms make use of nondeterministic finite automata (NFAs) because the translation can be efficiently done. Hence constructing NFAs with a smaller size for given REs is crucial in practical applications and a lot of research for efficiently generating smaller NFAs has been done (see [2, 5, 7, 8, 9, 10, 11]). Deterministic finite automata (DFAs) are also useful models for the RE matching problem because we can do an RE matching in a linear time. However, the DFA corresponding to an RE may become an exponential size in the length of the RE in the worst case. Hence constructing as compact a representation of DFAs as possible is desired. In techniques using automata, the time for constructing an NFA or a DFA from an RE is also crucial when the length of a given RE is large. Therefore, a faster translation algorithm from an RE to a finite automaton is also required. The aim of the paper is to present a compact DFA representation and efficient RE matching algorithms by introducing an NFA called a *dual position automaton*. Furthermore, we give a faster bit-parallel translation algorithm from an RE into a dual position automaton.

Two types of NFAs for REs are widely known, one is a *Thompson automaton* and the other is *a position automaton* (PA for short, also called *a Glushkov automaton*). Let r be an RE over an alphabet Σ, and let m be the total number of occurrences of alphabet symbols and operator symbols in r and let $\tilde{m}$ be the number of occurrences of alphabet symbols in r. Without loss of generality, we may assume $m = O(\tilde{m})$ as mentioned in [7]. As seen in [8], a Thompson automaton is an NFA with ϵ-moves and has at most $2m$ states and $4m$ transitions. Thompson automata can recursively be constructed based on the inductive definition of REs in $O(\tilde{m})$ time and space. Given a string of length n, the traditional algorithm using a Thompson automaton solves the RE matching problem in $O(mn)$ time and $O(m)$ space. Myers [12] has improved it using the Four Russians technique so that his algorithm can solve the RE matching problem in $O(mn/\log n)$ time and space. Recently, Bille [4] has proposed a new algorithm which improves $O(mn)$ time while preserving $O(m)$ space. His algorithm is also based on a Thompson automaton. B.W. Watson and R.E. Watson [18] have studied an extension of the Boyer-Moore string matching algorithm and presented a Boyer-Moore-style RE matching algorithm.

On the other hand, a PA is an ϵ-free NFA (that is, an NFA without any ϵ-moves) and has exactly $\tilde{m}+1$ states and at most $\tilde{m}^2+\tilde{m}$ transitions. PAs are also important models in practical applications because they become smaller than Thompson automata for some kinds of REs. For this reason,

some studies for efficiently constructing PAs have been done, and $O(\tilde{m}^2)$ time and space algorithms have been developed [5, 6]. Recently, Yamamoto, Miyazaki and Okamoto [16] have presented a faster bit-parallel algorithm generating a PA and related automata. PAs have another important property that for any state, *all incoming transitions* to the state have the same symbol. This property has a potential for improving complexities of RE matching algorithms. Indeed, Navarro and Raffinot [14, 15] have made use of this property to obtain a compact DFA representation and have presented a faster RE matching algorithm. Their compact DFA representation requires only $O(\tilde{m}2^{\tilde{m}})$ bits while a traditional DFA representation obtained from a Thompson automaton requires $O(m2^{2m})$ bits.

1.1 Contributions of the paper

In this paper, we will introduce an automaton model called *a dual position automaton* (dual PA), and present an efficient translation algorithm from an RE to a dual PA, a compact DFA representation, and efficient RE matching algorithms. Watson also define a similar automaton as *the dual constructions* in [17, Chapter 6, Section 6.7]. A dual PA is also an ϵ-free NFA with exactly $\tilde{m}+1$ states and at most $\tilde{m}^2+\tilde{m}$ transitions. Unlike a PA, however, it has a property that for any state, *all outgoing transitions* from the state have the same symbol. Clearly, by reversing a PA, we can get an NFA satisfying such a property, but it accepts a reversed string.

We first give a faster bit-parallel algorithm for translating an RE into a dual PA using a Thompson automaton. Our algorithm translates a given RE into a dual PA in $O(\tilde{m}\lceil\tilde{m}/w\rceil)$ time and space, where w is the length of a computer word. Furthermore, a generated dual PA is represented with $\tilde{m}(\tilde{m}+1)$ bits. Hence if $\tilde{m}=O(w)$, then the algorithm runs in $O(\tilde{m})$ time and space. Watson also give a construction based on a Thompson automaton in [17, Chapter 6, Section 6.7,Cnstruction 6.65]. However, he does not make a theoretical discussion on the time of the construction. In Chapter 14 of [17], he experimentally evaluates a performance of the construction. Generating a dual PA according to his construction directly, we would take $O(\tilde{m}^2)$ time in the worst case because we must compute $O(\tilde{m}^2)$ transitions of a dual PA. Our bit-parallel algorithm takes advantage of a nice property (see Lemma 1) of a Thompson automaton and achieves a speed-up.

Next we show a compact DFA representation obtained from a dual PA and an efficient RE matching algorithm. We improve an idea of Navarro and Raffinot [14, 15] by grouping all states of a dual PA by a symbol on the outgoing transitions. That is, we partition the set of states into at most

$|\Sigma|$ subsets Q_a according to a symbol a on transitions, where $|\Sigma|$ denotes the number of elements in Σ. In other words, if the outgoing transitions of states q and p have the same symbol, then q and p belong to the same subset. Then, for each subset Q_a, we make DFA transitions. In addition, by introducing an idea of a *lookahead symbol* in a matching process, we can achieve a more compact DFA representation and design an efficient RE matching algorithm. The matching time of the algorithm gets to depend on m_α but not on $\tilde{m}$, where m_α is defined as $m_\alpha = \max\{m_a \mid a \in \Sigma\}$ when m_a denotes the number of occurrences of an alphabet symbol a in r. Our DFA representation requires only $(\tilde{m}+1)\sum_{a\in\Sigma} 2^{m_a}$ bits. Since Navarro and Raffinot's representation needs $(\tilde{m}+1)\prod_{a\in\Sigma} 2^{m_a}$ bits, we improve the space. Especially, the space required would be much smaller when an RE consists of many kinds of symbols. The RE matching algorithm using our DFA representation runs in $O(n\lceil m_\alpha/w\rceil)$ time. Furthermore, the preprocessing time for constructing a DFA representation from r is $O(\lceil\tilde{m}/w\rceil(\tilde{m}+\sum_{a\in\Sigma} 2^{m_a}))$. Hence if $m_\alpha = O(w)$, then the matching time is $O(n)$ and the preprocessing time is $O(\tilde{m}+\sum_{a\in\Sigma} 2^{m_a})$. Note that the matching time of a standard DFA-based algorithm is normally described as $O(n)$, but a factor like $\lceil m_\alpha/w\rceil$ may be hidden. This is because, if a DFA has 2^s states, we need s bits for expressing each state and may need at least $\lceil s/w\rceil$ time to access each state. This time, the matching time becomes $O(n\lceil s/w\rceil)$.

Finally, we apply a decomposition technique to our algorithm. Although Navarro and Raffinot [15] decomposed the whole set of states, we decompose each subset Q_a.x[That is, we introduce a parameter $1 \le K_a \le w$ for each $a \in \Sigma$ and decompose each Q_a by a parameter K_a into $\lceil m_a/K_a\rceil$ subsets. By this improvement, we can present an RE matching algorithm running in $O(\lceil m_\alpha^2/(wK_\alpha)\rceil n)$ time using $O((\tilde{m}+1)\sum_{a\in\Sigma}\lceil m_a/K_a\rceil 2^{K_a})$ bits. If we have $K_a = \log n$ for every $a \in \Sigma$ and $\tilde{m} = O(w)$, then we can get an RE matching algorithm running in $O(\lceil m_\alpha/\log n\rceil n)$ time and $O(\tilde{m}n/\log n)$ space. This time, the preprocessing time is $O(\tilde{m}+\tilde{m}n/\log n)$.

We will rely on a w-bit uniform RAM to evaluate the complexities of algorithms. In general, since most papers assume $w \ge \log n$, we also do so. Note that we will describe algorithms using $\tilde{m}$-bit vectors or m_α-bit vectors for the sake of convenience. In a practical implementation, however, these bit vectors must be divided into w-bit vectors. Hence factors $\lceil\tilde{m}/w\rceil$ and $\lceil m_\alpha/w\rceil$ appear in the time and space complexities.

The paper is organized as follows. In Section 2, we will give basic definitions of REs. In Section 3, we will explain a Thompson automaton and a dual PA, and then give a translation algorithm from an RE into a dual PA. In Section 4 we will give compact DFA representations and RE

matching algorithms.

2 Regular expressions and some notations

We here give some definitions for regular expressions.

Definition 1 *Let Σ be an alphabet. The regular expressions (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 $\{\epsilon\}$ and the set $\{a\}$, respectively.*
2. *Let r_1 and r_2 be REs denoting the sets R_1 and R_2, respectively. Then $(r_1 \vee r_2)$, $(r_1 r_2)$ and (r_1^*) are also REs that denote the sets $R_1 \cup R_2$ (union), $R_1 R_2$ (concatenation), and R_1^* (Kleene closure or star), respectively.*

In this paper, we use the following notations related to the size of an RE.

- By $L(r)$ we denote the language generated by an RE r.
- By m we denote the number of occurrences of alphabet symbols and operator symbols in r. The length of r means this m.
- By m_a we denote the number of occurrences of an alphabet symbol a in r. In addition, we define $\tilde{m} = \sum_{a \in \Sigma} m_a$ and $m_\alpha = \max\{m_a \mid a \in \Sigma\}$.

As mentioned in Introduction, we may assume $m = O(\tilde{m})$. Hence we will use a parameter $\tilde{m}$ when stating our results. We make use of bit vectors for designing algorithms. Then we regard the rightmost bit as the least significant bit.

3 From Thompson automata to dual position automata

3.1 Thompson automata

Thompson automata are recursively constructed based on the definition of REs, whose construction algorithm is widely known (for example, see

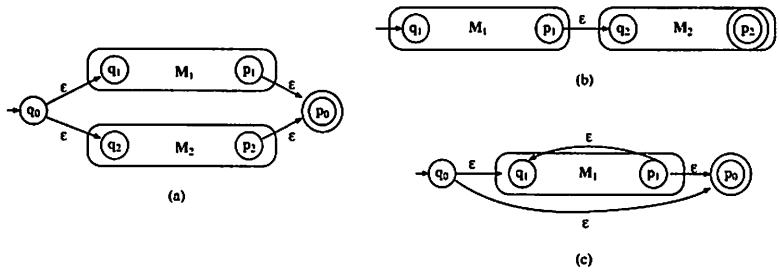


Figure 1: Translation from an RE into an NFA. (a) union ($r_1 \vee r_2$), (b) concatenation ($r_1 r_2$), and (c) Kleene closure (r_1^*).

[8]). We give an outline of the construction in Fig. 1. In Fig.1, (a), (b), and (c) show recursive constructions for union ($r_1 \vee r_2$), concatenation ($r_1 r_2$), and Kleene closure (r_1^*), respectively. Here M_1 and M_2 denote NFAs for REs r_1 and r_2, respectively. Let $M = (Q, \Sigma, \delta, q_0, q_f)$ be the Thompson automata obtained from an RE r of length m, where Q is a set of states, Σ is an alphabet, δ is a transition function, q_0 is the initial state, and q_f is the final state. Note that a Thompson automaton has just one initial state and one final state. Then M has at most $2m$ states and $4m$ transitions. Furthermore, for any state $q \in Q$, all outgoing transitions from q are caused either by the empty string ϵ or by an alphabet symbol $a \in \Sigma$. If the transitions from q are caused by an alphabet symbol, then we call state q a *sym-state*; otherwise *an ϵ-state*. We partition a set of *sym*-states into several subsets according to an alphabet symbol. Let us call a *sym*-state an *a-state* if the alphabet symbol of the *sym*-state is $a \in \Sigma$. Then we define $Q_a = \{q \mid q \text{ is an } a\text{-state}\}$. Such subsets Q_a play an important role in improving an efficiency of an algorithm. We have the following proposition.

Proposition 1 *For any RE r of length m, we can construct the Thompson automaton with at most $2m$ states and $4m$ transitions in $O(m)$ time and space.*

We define the reversed automaton M^R of M by reversing all transitions and interchanging the initial state and the final state. That is, for any states q, p of M and $\sigma \in \Sigma \cup \{\epsilon\}$, $p \in \delta(q, \sigma)$ if and only if $q \in \delta^R(p, \sigma)$, where δ^R is the transition function of M^R. We must note that *sym*-states and ϵ-states are defined in M. Hence if q is a *sym*-state in M, then the incoming transition to q is carried out by an alphabet symbol in M^R. In other words, in M^R, states to which the incoming transition is defined by an alphabet symbol become *sym*-states.

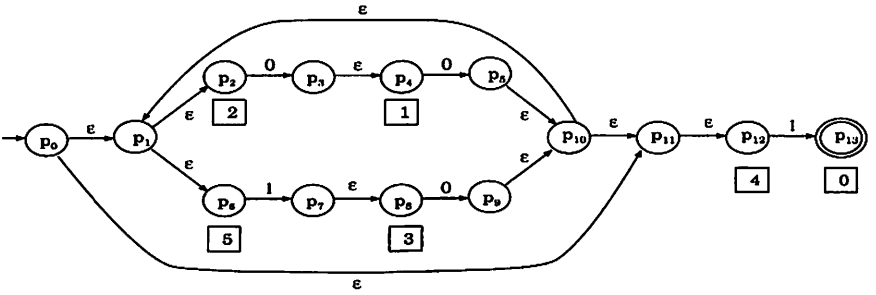


Figure 2: Thompson automaton for $r = (00 \vee 10)^*1$.

Let us introduce some notions for M and M^R. We call a state with two incoming transitions a *junction state*. Also we say that a state p is a *predecessor* of a state q if there is a transition from p to q. Furthermore, a sequences of transitions from a state to a state is called a *path*. As seen in (c) of Fig.1, a star operator generates a transition going back to a previous state. We call this transition a *back transition*. That is, the transition from p_1 to q_1 is a back transition. Note that a back transition in M becomes a back transition also in M^R. By removing such back transitions from M, we can sort the states of M from the initial state in a topological order. Here, by a topological order, we mean that for any states p and q, $p < q$ if and only if there is a directed path from p to q. It is clear that a topological order of states of M^R is defined by reversing that of M.

Thompson automata have the following important property.

Lemma 1 ([13], Lemma 1) *Let M be the Thompson automaton obtained from a given RE. Then, any loop-free path in M has at most one back transition.*

By the symmetrical structure of a Thompson automaton M, we notice that Proposition 1 and Lemma 1 also hold for M^R. With Lemma 1 and a reversed automaton, we can efficiently generate a bit vector representation of a dual position automaton.

Example Let us consider an RE $r = (00 \vee 10)^*1$ over $\Sigma = \{0, 1\}$. Fig. 2 shows the Thompson automaton for r and Fig. 3 shows the reversed automaton. Since *sym*-states are defined for a Thompson automaton, states p_2, p_4, p_6, p_8 and p_{12} are *sym*-states in Fig. 2 and Fig. 3. Furthermore, states p_2, p_4 and p_8 are 0-states, and p_6 and p_{12} are 1-states. In Fig. 2, states p_1, p_{10} and p_{11} are junction states and the transition from p_{10} to p_1

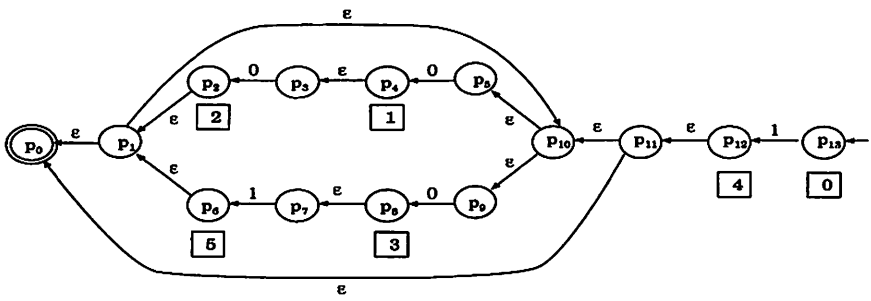


Figure 3: The reversed automaton for Fig. 2.

is a back transition. Similarly, in Fig. 3, states p_0, p_1 and p_{10} are junction states and the transition from p_1 to p_{10} is a back transition.

3.2 Position automata and dual position automata

First we will explain a position automaton (a PA for short), which is an NFA constructed from a regular expression. In general, a PA is defined as follows. We can also see similar definitions in [5, 9, 10, 11, 14]. Let r be an RE with $\tilde{m}$ occurrences of alphabet symbols. We make an expression $\bar{r}$ by numbering each alphabet symbol c occurring in r with its position. That is, if a symbol c is the symbol occurring at the i-th position in r, then it is changed to c_i in $\bar{r}$. We denote by I the set of such positions. For example, we have $\bar{r} = (0_1 0_2 \vee 1_3 0_4)^* 1_5$ and $I = \{1,2,3,4,5\}$ for $r = (00 \vee 10)^* 1$. The existing algorithms for generating a PA have focused on computing subsets of I, **First**, **Last**, and **Follow**(i). Here the set **First** is defined to be $\{i \mid c_i\alpha \in L(\bar{r})\}$, the set **Last** is defined to be $\{i \mid \alpha c_i \in L(\bar{r})\}$, and the set **Follow**$(i)$ is defined to be $\{j \mid \alpha c_i c_j \beta \in L(\bar{r})\}$. Then a PA $G = (Q, \Sigma, \delta, p_0, F)$ is defined as follows: the set Q of states is $\{p_0\} \cup I$, the initial state is p_0, the set F of final states is **Last**. In addition, for any $i \in I$ and $c \in \Sigma$, $\delta(i, c)$ is defined to be $\{j \mid j \in \textbf{Follow}(i) \text{ and } c_j = c\}$, and for any $c \in \Sigma$, $\delta(p_0, c)$ is defined to be $\{i \mid i \in \textbf{First} \text{ and } c_i = c\}$. Thus a PA G is an ϵ-free NFA and has the following properties.

Property 1 For any state q of G, all incoming transitions to q are activated by the same alphabet symbol $a \in \Sigma$.

Property 2 The number of initial states is just one, and the number of final states is more than or equal to one.

Property 3 The number of states is just $\tilde{m}+1$ and the number of transitions is at most $\tilde{m}^2+\tilde{m}$.

Alternatively, a PA can be constructed from a Thompson automaton. Let $M=(Q_1,\Sigma,\delta_1,q_0,q_f)$ be the Thompson automaton obtained from an RE r. We consider only the initial state q_0 and states $q\in Q_1$ such that the incoming transition to q is defined by an alphabet symbol, that is, $\delta_1(p,c)=q$ for a *sym*-state p. Let Q' be the set of these states. Then a PA $G=(Q,\Sigma,\delta,q_0,F)$ is defined as follows: $Q=Q'$, the state q_0 becomes the initial state of G, and the set F of final states consists of states from which one can reach the final state q_f by using only ϵ-moves in M. In addition, the transition function δ is defined: for any $q,p\in Q$, if there is a path from q to p with a string $\epsilon\cdots\epsilon c$ for a symbol $c\in\Sigma$ in M, then a transition from q to p by c is added to δ. Yamamoto et al. [16] gave a bit-parallel algorithm to generate a PA from a Thompson automaton according to this construction. Watson [17] also show a construction from a Thompson automaton.

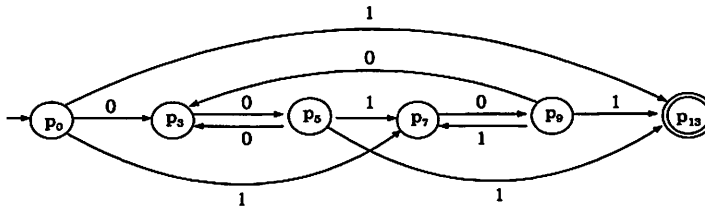
Now let us define a dual version of PAs, called a *dual position automaton* (a dual PA for short). A dual PA $\bar{G}$ is also an NFA constructed from an RE. Again let $M=(Q_1,\Sigma,\delta_1,q_0,q_f)$ be the Thompson automaton obtained from an RE r. Then a dual PA is defined to be an ϵ-free NFA which is constructed from the Thompson automaton M by considering only *sym*-states and the final state q_f in the similar way as a PA. Thus, unlike a PA, we define a dual PA by extracting *sym*-states and the final state q_f. More precisely speaking, when we define Q' to be the set of these states, a dual PA $\bar{G}=(Q,\Sigma,\delta,I,q_f)$ is defined as follows: $Q=Q'$, the set I of initial states consists of states which one can reach from the initial state q_0 by using only ϵ-moves in M, and the state q_f becomes the final state of $\bar{G}$. In addition, the transition function δ is defined: for any $q,p\in Q$, if there is a path from q to p with a string $c\epsilon\cdots\epsilon$ for a symbol $c\in\Sigma$ in M, then a transition from q to p by c is added to δ. In the next section, we will give an algorithm which generates a dual PA from a Thompson automaton according to this definition. A dual PA $\bar{G}$ has the following properties.

Property 1' For any state q of $\bar{G}$, all outgoing transitions from q are activated by the same alphabet symbol $a\in\Sigma$.

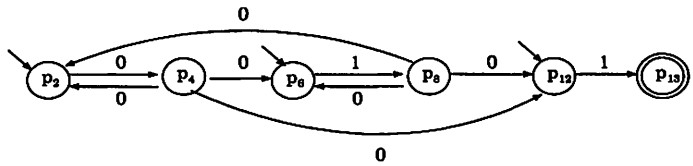
Property 2' The number of initial states is more than or equal to one, and the number of final states is just one.

Property 3 The number of states is just $\tilde{m}+1$ and the number of transitions is at most $\tilde{m}^2+\tilde{m}$.

As we can see, Property 1 and 1', and Property 2 and 2' become symmetric. Property 3 holds for both of a PA and a dual PA.



(a) Position automaton



(b) Dual position automaton

Figure 4: Position automaton and dual position automaton for $r = (00 \vee 10)^*1$. These automata are obtained from the Thompson automaton given in Fig. 2.

We will make use of a dual PA for an RE matching algorithm. Hence, to generate a bit representation of a dual PA, we also make use of a Thompson automaton as in [16].

Example Let us consider an RE $r = (00 \vee 10)^*1$ again. Fig. 4 shows the PA and the dual PA for r. These automata are made from the Thompson automaton given in Fig. 2. The PA is made by considering only states $p_3, p_5, p_7, p_9, p_{13}$ and the initial state p_0. In addition, the set of positions for r is $\{1, 2, 3, 4, 5\}$ because $\bar{r} = (0_1 0_2 \vee 1_3 0_4)^* 1_5$; states p_3, p_5, p_7, p_9 and p_{13} correspond to positions $1, 2, 3, 4$ and 5, respectively. On the other hand, the dual PA is made by considering only states $p_2, p_4, p_6, p_8, p_{12}$ and the final state p_{13}. The PA has one initial state p_0 and one final state p_{13}, while the dual PA has three initial states p_2, p_6, p_{12} and one final state p_{13}.

3.3 A bit-parallel translation algorithm from an RE to a dual PA

We focus on *sym*-states and the final state of a Thompson automaton. By computing the reachability between these states, we can construct a dual

PA from a Thompson automaton. We will present an efficient bit-parallel algorithm for translating an RE into a dual PA by using a Thompson automaton. We number *sym*-states to implement a set of *sym*-states as a bit vector. To do this, when $\Sigma = \{a_1, \cdots, a_l\}$, we number states in order of Q_{a_1}, Q_{a_2}, $\cdots$, Q_{a_l}. That is, states of Q_{a_1} are numbered from 1 to m_{a_1}, states of Q_{a_2} are numbered from $m_{a_1}+1$ to $m_{a_1}+m_{a_2}$, $\cdots$, and states of Q_{a_l} are numbered from $\sum_{j<l} m_{a_j}+1$ to $\sum_{j\leq l} m_{a_j}$. We assign the number 0 to the final state. For any state q, we denote by $num(q)$ a number assigned to q. In Fig. 2, a number attached to each *sym*-state is the number of the state; that is, states p_{13}, p_4, p_2, p_8, p_{12}, and p_6 are numbered 0, 1, 2, 3, 4, and 5, respectively.

Let r be an RE over Σ and let $M = (Q, \Sigma, \delta, q_0, q_f)$ be the Thompson automaton constructed from r. To compute the transition function of a dual PA, we compute an array $NEXT[q, \sigma]$ whose elements are bit vectors of $\tilde{m}+1$ bits, where $q \in Q$ and $\sigma \in \Sigma \cup \{\epsilon\}$. Here note that if $\sigma = a \in \Sigma$, then $q \in Q_a$. In addition, the array $NEXT[q, \sigma]$ satisfies that the i-th bit of $NEXT[q, \sigma]$ is equal to 1 if and only if there is a path from state q to a *sym*-state p with number i. If q is an a-state, then this means that M can move from state q to p by an alphabet symbol a. We can efficiently compute $NEXT[q, a]$ by using the property of Lemma 1 and the reversed automaton M^R of M. Note that Lemma 1 also holds for M^R. Finally, the desired dual PA is represented by $NEXT[q, a]$ on all *sym*-states q and alphabet symbols a. Hence the size of $NEXT[q, a]$ becomes $\tilde{m}(\tilde{m}+1)$ bits.

The algorithm starts with *REtoDPA* given in Fig. 5. In the algorithm, the operator | denotes bitwise OR. Furthermore, we use two functions *BitSet* and *BitCheck*. *BitSet*(v, i) sets the i-th bit of v to 1. *BitCheck*(v, i) checks whether or not the i-th bit of v is equal to 1, and if equal, then it returns 1; otherwise returns 0. Since these functions can easily be implemented so that they can run in $O(1)$ time, the details are omitted here. For any RE r, *REtoDPA* translates r into the Thompson automaton M, and then invokes the procedure *ReachState* for M^R. The procedure *ReachState*, given in Fig. 6, computes array $NEXT[q, a]$ using M^R. This time, Lemma 1 guarantees that we can correctly compute $NEXT[q, a]$ by traversing all states of M^R twice in a topological order. We have the following theorem.

Theorem 2 *Let r be an RE with $\tilde{m}$ occurrences of alphabet symbols. Then the algorithm* REtoDPA *correctly translates r into the dual PA in $O(\tilde{m}\lceil \tilde{m}/w \rceil)$ time and space. If $\tilde{m} = O(w)$, then it runs in $O(\tilde{m})$ time and space.*

Algorithm REtoDPA(r)
Input: an RE r.
Output: a dual PA $\bar{G} = (Q', \Sigma, \delta', I, q_f)$.

Step 1. Generate the Thompson automaton M and the reversed automaton M^R from r. In addition, compute Σ_r which is the set of alphabet symbols occurring in r.

Step 2. For all $a \in \Sigma_r$, number each state of Q_a.

Step 3. Let q_0 be the initial state of M. Then, add a new initial state q_{ini} and a transition from q_{ini} to q_0 by ϵ to M.

Step 4. For all states q of M, if q is an a-state for an alphabet symbol $a \in \Sigma_r$, then $NEXT[q, a] := 0$; otherwise $NEXT[q, \epsilon] := 0$.

Step 5. Do *ReachState*(M^R,*NEXT*,Σ_r).

Step 6. Construct $\bar{G}$ as follows:

1. define Q' to be the set $\{q \mid q$ is a *sym*-state or the final of $M\}$ and $\Sigma = \Sigma_r$,
2. define the final state q_f to be the final state of M,
3. for all $a \in \Sigma$ and all $q \in Q_a$
   define $\delta'(q, a)$ to be $NEXT[q, a]$,
4. for all states $q \in Q'$
   if $BitCheck(NEXT[q_{ini}, \epsilon], num(q)) = 1$, then add q to I.

Figure 5: The algorithm *REtoDPA*.

3.4 Proof of Theorem 2

Let M be the Thompson automaton for a given RE r. First we will prove the following lemma to show the correctness of the algorithm.

Lemma 3 *The i-th bit of $NEXT[q, a]$ becomes 1 if and only if M can move from an a-state q to a sym-state p with $num(p) = i$ by the alphabet symbol a.*

Proof. First let us show the *only-if-part*. For any states q and p of M, it is clear that there is a path from q to p if and only if there is a path from p to q in M^R. We can easily see that if the i-th bit of $NEXT[q, \sigma]$ is set to 1 by *ReachState*, then for a *sym*-state p with $num(p) = i$, there is a path from p to q in M^R. This time, if q is an a-state for any alphabet symbol a, then $\sigma = a$ and the sequence of symbols over the path is $\epsilon \cdots \epsilon \cdot a$. That is, this means that M can move from q to p by the alphabet symbol a.

Next let us show the reverse direction, that is, the *if-part*. To do this, it is sufficient to show the following claim. Here $Q' = \{q \mid q$ is a *sym*-state or the final state of $M\}$.

Claim *For any states $q, p \in Q'$, if M can move from q to p by a symbol*

Procedure ReachState(M^R,$NEXT$,Σ_r)

Repeat the following twice:
for all states q of M^R do the following in a topological order:

1. if q is the initial state, then $BitSet(NEXT[q,\epsilon], num(q))$,
2. if $q \in Q_a$ for a symbol $a \in \Sigma_r$ and q has the incoming transition from a state p_1 (note that in this case q has just one incoming transition), then $NEXT[q,a] := NEXT[p_1,\epsilon]$ and $BitSet(NEXT[q,\epsilon], num(q))$,
3. if q is a junction state, then $NEXT[q,\epsilon] := NEXT[p_1,\epsilon] \mid NEXT[p_2,\epsilon]$, where p_1 and p_2 are two predecessors of q;
4. otherwise $NEXT[q,\epsilon] := NEXT[p_1,\epsilon]$, where p_1 is a predecessor of q.

Figure 6: The procedure *ReachState*.

a, then ReachState *sets the $num(p)$-th bit of $NEXT[q,a]$ to* 1.

Proof of the claim. Since M can move from q to p by a symbol a, there is a loop-free path from q to p in M. Hence the reversed path from p to q is also loop-free in M^R. Furthermore, this reversed path have at most one back transition because Lemma 1 holds for M^R. Now let this path be $Z = p_1(=p), p_2, \ldots, p_t(=q)$. If Z does not contain any back transitions, then *ReachState* sets the $num(p)$-th bit of $NEXT[q,a]$ to 1 in the first traverse of Step 2. This is because we have $p_1 < p_2 < \cdots < p_t$ in a topological order and all states other than q and p are ϵ-states. Next suppose that Z contains one back transition. Here, without loss of generality, let the transition from p_l to p_{l+1} be a back transition. This time, we have $p_1 < \cdots < p_l$ and $p_{l+1} < \cdots < p_t$ in a topological order. Hence, since *ReachState* can see that there is a path from p to p_l in the first traverse of Step 2, it sets the $num(p)$-th bit of $NEXT[p_l,\epsilon]$ to 1. However, at this moment, it may not know whether or not there is a path from p_l to q. In the second traverse of Step 2, *ReachState* can see that there is a path from p to q because it can use the value of $NEXT[p_l,\epsilon]$ by the back transition from p_l to p_{l+1}. Hence *ReachState* sets the $num(p)$-th bit of $NEXT[q,a]$ to 1. Thus we have proven the claim, and hence have proven the lemma.

It follows from Lemma 3 that the algorithm *REtoDPA* correctly computes a dual PA.

Next let us discuss the complexity. By Proposition 1, we can construct the Thompson automaton M and the reversed automaton M^R with $O(\tilde{m})$ states and transitions in $O(\tilde{m})$ time and space. The procedure *ReachState* traverses all states and transitions of M at most twice. When W is word-length of a computer, since it takes $O(\lceil \tilde{m}/w \rceil)$ time to process each state, Step 3 takes $O(\tilde{m}\lceil \tilde{m}/w \rceil)$ time. Step 4 also takes $O(\tilde{m}\lceil \tilde{m}/w \rceil)$ time. Hence

Table 1: DFA representation $D[a, b, I_a]$ for the dual PA of Fig. 4.

			$b=0$	$b=1$
			(p_2, p_4, p_8)	(p_6, p_{12})
$a=0$	$I_0 = (p_2, p_4, p_8)$	$(1,0,0)$	$(0,1,0)$	$(0,0)$
		$(0,1,0)$	$(1,0,0)$	$(1,1)$
		$(0,0,1)$	$(1,0,0)$	$(1,1)$
		$(1,1,0)$	$(1,1,0)$	$(1,1)$
		$(1,0,1)$	$(1,1,0)$	$(1,1)$
		$(0,1,1)$	$(1,0,0)$	$(1,1)$
		$(1,1,1)$	$(1,1,0)$	$(1,1)$
$a=1$	$I_1 = (p_6, p_{12})$	$(1,0)$	$(0,0,1)$	$(0,0)$
		$(0,1)$	$(0,0,0)$	$(0,0)$
		$(1,1)$	$(0,0,1)$	$(0,0)$

the total time becomes $O(\tilde{m}\lceil\tilde{m}/w\rceil)$. The space mainly depends on an array $NEXT[q, a]$, which requires $O(\tilde{m}\lceil\tilde{m}/w\rceil)$ space. Hence the space becomes $O(\tilde{m}\lceil\tilde{m}/w\rceil)$. If $\tilde{m} = O(w)$, then the algorithm runs in $O(\tilde{m})$ time and space. Thus the theorem has been proved.

4 RE matching algorithms

We first give a compact DFA representation and a matching algorithm using it, and then will extend them by grouping parameters K_a. We give algorithms solving an extended version of the RE matching problem. That is, given an RE r and a string x, our algorithms output endpoints of all substrings of x which match r.

4.1 A compact DFA representation and a matching algorithm

Let us give a compact DFA representation for a dual PA $\bar{G}$. Our DFA representation consists of two arrays $D[a, b, I_a]$ and $FINAL[a]$. We generate $D[a, b, I_a]$ and $FINAL[a]$ from a dual PA $\bar{G}$ with $NEXT[p, a]$, where $a, b \in \Sigma$ and $0 \leq I_a \leq 2^{m_a} - 1$. The element of $D[a, b, I_a]$ is a bit vector of m_b bits, and if $D[a, b, I_a] = I_b$ $(0 \leq I_b \leq 2^{m_b} - 1)$, then it means that there is a transition from a subset I_a of a-states to a subset I_b of b-states in $\bar{G}$. Note that I_a is a bit vector representation for subsets of the set Q_a of a-states

Algorithm REMatchDFA(r,x)
Input: an RE r and a string $x = x_1 \cdots x_n$, where $x_i \in \Sigma = \{a_1, \cdots, a_l\}$.
Output: endpoints i of all substrings of x matching r.

Step 1. Generate a dual PA $\bar{G} = (Q, \Sigma, \delta, I_q, q_f)$ using *REtoDPA*(r).

Step 2. Generate a DFA representation $D[a, b, I_a]$ and $FINAL[a]$ using *GenDFA*($\bar{G}$).

Step 3. /* Set the initial states. A bit vector $INIT[a]$ is set for indicating all initial a-states. $INIT$ is an $\tilde{m}+1$-bit vector each bit of which corresponds to a state of Q.*/

1. $INIT := 0$,
2. for all states $q \in I_q$, $BitSet(INIT, num(q))$,
3. $J := 1$,
4. for $a = a_1, \cdots, a_l$,
   (a) $INIT[a] := (INIT \ \& \ B[a]) >> J$,
   (b) $J := J + m_a$.

Step 4. If the final state q_f is included in I_q, then output 0, /* This means that ϵ matches r */

Step 5. $CSTATE := INIT[x_1]$.

Step 6. For $i := 1$ to $n-1$ do

1. if $BitCheck(FINAL[x_i], CSTATE]) = 1$, then output the position i,
2. $CSTATE := D[x_i, x_{i+1}, CSTATE]$,
3. /* set self-loop on the initial states to find all substrings matching r*/ $CSTATE := CSTATE \mid INIT[x_{i+1}]$.

Step 7. If $BitCheck(FINAL[x_n], CSTATE]) = 1$, then output the position n.

Figure 7: The algorithm *REMatchDFA*.

of $\bar{G}$. The element v of array $FINAL[a]$ is a bit vector of 2^{m_a} bits, which satisfies that the i-th bit of v is 1 if and only if there is a transition on a from a state of the i-th subset of Q_a to the final state of $\bar{G}$, where each subset of Q_a is numbered by the corresponding I_a. Then $D[a, b, I_a]$ and $FINAL[a]$ are generated by procedure *GenDFA*($\bar{G}$) given in Fig.8, which is an extension of a technique used in [14, 15]. To compute $D[a, b, I_a]$, we first compute the array $E[I_a]$ denoting a transition from a subset I_a of a-states to *sym*-states, and then compute a transition from a subset I_a of a-states to b-states. The size of $D[a, b, I_a]$ is $\tilde{m} \sum_{a \in \Sigma} 2^{m_a}$ bits because $\tilde{m} = \sum_{a \in \Sigma} m_a$, and the size of $FINAL[a]$ is $\sum_{a \in \Sigma} 2^{m_a}$ bits. Hence the total size is $(\tilde{m}+1) \sum_{a \in \Sigma} 2^{m_a}$ bits. In the algorithm, the operator & denotes bitwise *AND* and the operator >> denotes *Shift Right*. Furthermore, $B[b]$ denotes a bit-mask for b-states, that is, its value is an $\tilde{m}+1$-bit vector in which only bits corresponding to b-states are 1; other bits are 0. By the

Procedure GenDFA($\bar{G}$)
Comment: constructing $D[a,b,I_a]$ and $FINAL[a]$ from a dual PA $\bar{G}$ with $NEXT[q,a]$.

```
1.  I := 1 and E[0] := 0
2.  for a := a_1, ..., a_l /* Σ = {a_1, ..., a_l} */
3.      for i := 0, ..., m_a − 1
4.          for j := 0, ..., 2^i − 1
5.              E[2^i + j] := E[j] | NEXT[I + i, a]
6.              if BitCheck(E[2^i + j], 0) = 1, then BitSet(FINAL[a], 2^i + j)
7.              J := 1
8.              for b := a_1, ..., a_l
9.                  D[a, b, 2^i + j] := (E[2^i + j] & B[b]) >> J
10.                 J := J + m_b
11.             end-for
12.         end-for
13.     end-for
14.     I := I + m_a
15. for-end
```

Figure 8: The procedure *GenDFA*.

discussion above, we have the following result for the size of a compact DFA representation.

Proposition 2 *The total size of* $D[a,b,I_a]$ *and* $FINAL[a]$ *is* $(\tilde{m}+1)\sum_{a\in\Sigma} 2^{m_a}$ *bits.*

Example We give a compact DFA representation $D[a,b,I_a]$ and $FINAL[a]$ for the dual PA given in Fig.4. Table 1 shows $D[a,b,I_a]$, where each entry denotes a bit vector. For instance, $D[0,0,(1,0,0)] = (0,1,0)$. Since only state p_{12} can reach a final state p_{13}, $FINAL[a]$ is defined as follows: $FINAL[0] = (0,0,0,0,0,0,0,0)$ and $FINAL[1] = (1,0,1,0)$ because for (p_6,p_{12}), $(0,1)$ and $(1,1)$ contain state p_{12}. Here note that these binary numbers $(0,1)$ and $(1,1)$ are 1 and 3, respectively, because the rightmost bit is the least significant bit.

The matching algorithm $REMatchDFA(r,x)$, given in Fig.7, outputs endpoints of all substrings of x matching r. The algorithm makes use of one lookahead symbol to hold only a-states in *CSTATE* for some alphabet symbol a at a time. Hence *CSTATE* consists of at most m_α bits. Let x_i be the i-th symbol of x. Then, for any i, *CSTATE* always consists of x_i-states reachable from the initial states by processing $x_1 \cdots x_{i-1}$.

REMatchDFA computes the set of x_{i+1}-states reachable from *CSTATE* by x_i using $D[x_i, x_{i+1}, I_a]$. Thus, after having processed x_i, *CSTATE* consists of x_{i+1}-states reachable from the initial states by $x_1 \cdots x_i$. As you see, in $D[a, b, I_a]$, symbol a corresponds to current symbol and symbol b corresponds to the next symbol (that is, the lookahead symbol).

In *REMatchDFA*(r, x), we call Steps 1 and 2 *a preprocessing part* and call Step 3 to 7 *a matching part.* In the preprocessing part, we generate a DFA representation $D[a, b, I_a]$ and $FINAL[a]$ from a given RE r. We get the following theorem. Here, as mentioned before, $\tilde{m}$ is the number of occurrences of alphabet symbols in an given RE r and m_a is the number of occurrences of an alphabet symbol a, and $m_\alpha = \max\{m_a \mid a \in \Sigma\}$.

Theorem 4 *The algorithm REMatchDFA*(r, x) *can find endpoints of all substrings of x matching r. In addition, the preprocessing part runs in $O(\lceil \tilde{m}/w \rceil (\tilde{m} + \sum_{a \in \Sigma} 2^{m_a}))$ time and the matching part runs in $O(n \lceil m_\alpha / w \rceil)$ time using $O(\lceil \tilde{m}/ w \rceil \sum_{a \in \Sigma} 2^{m_a})$ space. If $\tilde{m} = O(w)$, then the preprocessing part runs in $O(\tilde{m} + \sum_{a \in \Sigma} 2^{m_a})$ time and the matching part runs $O(n)$ time using $O(\sum_{a \in \Sigma} 2^{m_a})$ space.*

Proof. First let us show that if a substring y of x matches r, then the endpoint of y is output. We prove the following claim: for any i_1 and i with $1 \leq i_1 \leq i \leq n-1$, *CSTATE* always consists of all x_i-states reachable from the initial states I_q by $x_{i_1} \cdots x_{i-1}$ when x_i is processed, where $i_1 \geq i - 1$ means the empty string. In Step 3, we first define bit vectors $INIT[a]$ for all symbols a, each of which consists of at most m_α bits. This time, a bit vector $INIT[a]$ satisfies that the i-th bit is 1 if and only if the i-th state of Q_a is an initial state.

For any fixed position $i_1 \geq 1$, we prove the claim by induction on position i. Let us first consider the case $i = i_1$. If $i_1 = 1$, then *CSTATE* is set to the initial x_1-states $INIT[x_1]$ in Step 5; otherwise, in 3 of Step 6, $INIT[x_i]$ is added to *CSTATE* when x_i is processed. Hence it is clear that the claim holds.

Assume that the claim holds for all positions less than or equal to $i \geq i_1$. Then let us show that the claim holds for $i + 1$. *REMatchDFA* computes the set of x_{i+1}-states reachable from *CSTATE* by x_i using $D[x_i, x_{i+1}, I_a]$. Thus, after having processed x_i, *CSTATE* consists of x_{i+1}-states reachable from the initial states by $x_1 \cdots x_i$. Thus the claim holds. Since *CSTATE* is an m_{x_i}-bit vector, we can regard it as an integer t between 0 and $2^{m_{x_i}} - 1$. Then the t-th bit of $FINAL[x_i]$ is 1 if it is possible to reach from states denoted by t to a final state by x_i. Let $y = x_{i_1} \cdots x_i$. Since y is in $L(r)$, after having processed x_{i-1}, *CSTATE* contains an x_i-state reachable to the

final state q_f by the claim because a dual PA $\bar{G}$ can move to q_f by $x_{i_1} \cdots x_i$. Hence 1 of Step 6 becomes true because the *CSTATE*-th bit of $FINAL[x_i]$ is 1. Thus the position i is output. Conversely if the position i is output, then there is an integer i_1 such that $x_{i_1} \cdots x_i$ is in $L(r)$. We omit the proof because it can be done similarly.

Next let us check the time for the preprocessing part. By Theorem 2, it takes $O(\tilde{m})$ time to generate a dual PA $\bar{G}$. The procedure $GenDFA(\bar{G})$ constructs $D[a, b, I_a]$ and $FINAL[a]$ in $O(\lceil \tilde{m}/w \rceil \sum_{a \in \Sigma} 2^{m_a})$ time. Hence it takes $O(\lceil \tilde{m}/w \rceil \ (\tilde{m} + \sum_{a \in \Sigma} 2^{m_a}))$ time in total. Let us see the matching part. The time of the matching part is dominated by Step 6. Since, for each symbol x_i, Step 6 takes $O(\lceil m_\alpha/w \rceil)$ time, it takes $O(n\lceil m_\alpha/w \rceil)$ time in total. The space required depends on $D[a, b, I_a]$, $FINAL[a]$, $B[a]$ and $INIT[a]$. Hence from Proposition 2, we obtain $O((\tilde{m}+1)\sum_{a \in \Sigma} 2^{m_a})$ bits, and hence we obtain $O(\sum_{a \in \Sigma} 2^{m_a})$ space. Thus the theorem has been proved.

4.2 Improvement by a decomposition technique

Navarro and Raffinot [15] partition the whole set of states of a PA into several subsets and construct DFA-like transitions for each subset of states. For each $a \in \Sigma$, we introduce a parameter $1 \leq K_a \leq w$, and then partition each subset Q_a, but not the whole set, into $t = \lceil m_a/K_a \rceil$ subsets $Q_a^0, \cdots, Q_a^{t-1}$ with $|Q_a^0| = \cdots = |Q_a^{t-2}| = K_a$ and $|Q_a^{t-1}| \leq K_a$, where $|Q_a^j|$ denotes the number of elements in Q_a^j. Then we generate a DFA representation for every subset. We call such a DFA representation *a partial DFA representation*, which is represented by an array $D[a, b, I_a^{h_a}, h_a]$ and $FINAL[a, h_a]$, where $a, b \in \Sigma$, $0 \leq I_a^{h_a} \leq 2^{K_a}$ and $0 \leq h_a \leq t-1$. The array $D[a, b, I_a^{h_a}, h_a]$ is computed by the procedure $GenPartial(\bar{G})$ given in Fig. 10 and satisfies

$$D[a, b, I_a] = D[a, b, I_a^0, 0] \mid D[a, b, I_a^1, 1] \mid \cdots \mid D[a, b, I_a^{t-1}, t-1]$$

Furthermore $FINAL[a, h_a]$ is an array of bit vectors for $Q_a^{h_a}$, which is defined in the same way as $FINAL[a]$. The element of $FINAL[a, h_a]$ is a bit vector of $2^{|Q_a^{h_a}|}$ bits. Note that $2^{|Q_a^{h_a}|} \leq 2^{K_a}$. The partial DFA representation leads to a more efficient matching algorithm *REMatchPartial* given in Fig.9.

Example We give a partial DFA representation $D[a, b, I_a^{h_a}, h_a]$ for the dual PA given in Fig.4. Table 2 shows $D[a, b, I_a^{h_a}, h_a]$, where we set $K_0 = K_1 = 2$. Hence $Q_0 = \{p_2, p_4, p_8\}$ is divided into two subsets $Q_0^0 = \{p_2, p_4\}$ and $Q_0^1 = \{p_8\}$. Comparing with $D[a, b, I_a]$ in Table 1, we see that the size of $D[a, b, I_a^{h_a}, h_a]$ becomes smaller.

Algorithm REMatchPartial(r,x)
Input: an RE r and a string $x = x_1 \cdots x_n$, where $x_i \in \Sigma$..
Output: endpoints i of all substrings of x matching r.

Step 1. Generate a dual PA $\tilde{G} = (Q, \Sigma, \delta, I_q, q_f)$ using *REtoDPA*(r).

Step 2. Generate a partial DFA representation $D[a, b, I_a, h_a]$ and $FINAL[a, h_a]$ using *GenPartial*($\tilde{G}$).

Step 3. /* Set the initial states. A bit vector $INIT[a]$ is set for indicating all initial a-states. $INIT$ is an $\tilde{m}+1$-bit vector each bit of which corresponds to a state of Q. */

1. $INIT := 0$,
2. for all states $q \in I_q$, $BitSet(INIT, num(q))$,
3. $J := 1$,
4. for $a := a_1, \cdots, a_l$,
   (a) $INIT[a] := (INIT \ \& \ B[a]) >> J$,
   (b) $J := J + m_a$.

Step 4. If the final state q_f is included in I_q, then output 0, /* This means that ϵ matches r */

Step 5. $CSTATE := INIT[x_1]$.

Step 6. For $i := 1$ to $n-1$ do

1. $Temp := 0$ and $Match := false$,
2. for $h := 0, \cdots, \lceil m_{x_i}/K_{x_i} \rceil - 1$,
   (a) /* extract K_{x_i} bits from $CSTATE$ */
   $KSTATE := CSTATE \ \& \ 0 \cdots 01^{K_{x_i}}$,
   (b) if $BitCheck(FINAL[x_i, h], KSTATE) = 1$, then $Match := true$,
   (c) $Temp := Temp \mid D[x_i, x_{i+1}, KSTATE, h]$,
   (d) $CSTATE := CSTATE >> K_{x_i}$,
3. if $Match = true$, then output i,
4. /* update the current state and set self-loop on the initial states */
   $CSTATE := Temp \mid INIT[x_{i+1}]$.

Step 7. /* Check out a match on the last symbol x_n. */

1. $Match := false$,
2. for $h := 0, \cdots, \lceil m_{x_n}/K_{x_n} \rceil - 1$,
   (a) $KSTATE := CSTATE \ \& \ 0 \cdots 01^{K_{x_n}}$,
   (b) if $BitCheck(FINAL[x_n, h], KSTATE) = 1$, then $Match := true$,
   (c) $CSTATE := CSTATE >> K_{x_n}$,
3. if $Match = true$, then output n.

Figure 9: The algorithm *REMatchPartial*.

Procedure GenPartial($\bar{G}$)
Comment: constructing $D[a, b, I_a, h_a]$ and $FINAL[a, h_a]$ from a dual PA $\bar{G}$ with $NEXT[q, a]$.

1. $I := 1$ and $E[0] := 0$
2. for $a := a_1, \cdots, a_l$ /* $\Sigma = \{a_1, \cdots, a_l\}$ */
3. for $h := 0, \cdots, \lceil m_a/K_a \rceil - 1$
4. if $h \neq \lceil m_a/K_a \rceil - 1$, then $K := K_a$; otherwise $K := m_a - K_a h$
5. for $i := 0, \ldots, K - 1$
6. for $j := 0, \cdots, 2^i - 1$
7. $E[2^i + j] := E[j] \mid NEXT[I + i, a]$
8. if $BitCheck(E[2^i + j], 0) = 1$, then $BitSet(FINAL[a, h], 2^i + j)$
9. $J := 1$
10. for $b := a_1, \cdots, a_l$
11. $D[a, b, 2^i + j, h] := (E[2^i + j]$ & $B[b]) >> J$
12. $J := J + m_b$
13. end-for
14. end-for
15. $I := I + K + 1$
16. end-for
17. end-for
18. end-for

Figure 10: The procedure *GenPartial*.

We have the following result for the size of a partial DFA representation.

Proposition 3 *The total size of* $D[a, b, I_a, h_a]$ *and* $FINAL[a, h_a]$ *is* $O((\tilde{m}+1)\sum_{a\in\Sigma}\lceil m_a/K_a\rceil 2^{K_a})$ *bits.*

In $REMatchPartial(r, x)$, we again call Steps 1 and 2 *a preprocessing part* and call Step 3 to 7 *a matching part.* Then we have the next theorem. We omit the proof because it can be done in the same way as Theorem 4.

Theorem 5 *The algorithm* $REMatchPartial(r, x)$ *can find endpoints of all substrings of* x *matching* r. *In addition, the preprocessing part runs in* $O(\lceil \tilde{m}/w \rceil(\tilde{m} + \sum_{a\in\Sigma}\lceil m_a/K_a\rceil 2^{K_a}))$ *time and the matching part runs in* $O(\lceil m_\alpha^2/(wK_\alpha)\rceil n)$ *time using* $O(\lceil \tilde{m}/w \rceil \sum_{a\in\Sigma}\lceil m_a/K_a\rceil 2^{K_a})$ *space.*

Each parameter K_a can be regarded as a kind of a measure to decide a degree of determinism. The following corollary is directly obtained from Theorem 5.

Table 2: Partial DFA representation $D[a, b, I_a^{h_a}, h_a]$ for the dual PA of Fig. 4.

			$b=0$	$b=1$
			(p_2, p_4, p_8)	(p_6, p_{12})
$a=0$	$I_0^0 = (p_2, p_4)$	$(1,0)$	$(0,1,0)$	$(0,0)$
		$(0,1)$	$(1,0,0)$	$(1,1)$
		$(1,1)$	$(1,1,0)$	$(1,1)$
	$I_0^1 = (p_8)$	(1)	$(1,1,0)$	$(1,1)$
$a=1$	$I_1^0 = (p_6, p_{12})$	$(1,0)$	$(0,0,1)$	$(0,0)$
		$(0,1)$	$(0,0,0)$	$(0,0)$
		$(1,1)$	$(0,0,1)$	$(0,0)$

Corollary 5.1 *Let us set $K_a = \log n$ for every $a \in \Sigma$. Then the preprocessing part runs in $O(\lceil \tilde{m}/w \rceil (\tilde{m}n/\log n))$ time and the matching part runs in $O(m_a^2 n/(w \log n))$ time using $O(\lceil \tilde{m}/w \rceil \tilde{m}n/\log n)$ space.*

We here want to note that, for any symbol a, if the number of elements in the last subset Q_a^{t-1} is less than K_a, then the variable K is set to $|Q_a^{t-1}|$, but not K_a, in step 4 of *GenPartial*. Hence the term $\sum_{a \in \Sigma} \lceil m_a/K_a \rceil 2^{K_a}$ appearing in the preprocessing time and space of Theorem 5 can be expanded into $\sum_{a \in \Sigma} (2^{K_a} + \cdots + 2^{K_a} + 2^{|Q_a^{t-1}|})$. Therefore we have $O(\lceil \tilde{m}/w \rceil (\tilde{m}n/\log n))$ preprocessing time and space in the above corollary.

From the corollary, if $\tilde{m} = O(w)$, then we get an RE matching algorithm running in $O(m_\alpha n/\log n)$ time with preprocessing time $O(\tilde{m}n/\log n)$ and using $O(\tilde{m}n/\log n)$ space.

5 Conclusions

Summarizing the paper, we have shown the following results by introducing a dual PA:

- the bit-parallel translation algorithm from an RE into a dual PA,
- methods to generate a compact DFA representation from a dual PA,
- efficient RE matching algorithms using our compact DFA representation.

Acknowledgments

We would like to thank the referees for useful comments and for showing us the references [17, 18].

References

[1] A.V. Aho, *Algorithms for finding patterns in strings*, In J.V. Leeuwen, ed. Handbook of theoretical computer science, Elsevier Science Pub., 1990.

[2] V. Antimirov, *Partial derivative of regular expressions and finite automaton construction*, Theoret. Comput. Sci., 155(1996), 291–319.

[3] A. Apostolico, Z. Galil ed., *Pattern Matching Algorithms*, Oxford University Press, 1997.

[4] P. Bille, *New Algorithms for Regular Expression Matching*, Proc. of ICALP 2006, LNCS 4501(2006), 643–654.

[5] A. Brüggemann-Klein, *Regular expressions into finite automata*, Theoret. Comput. Sci., 120(1993), 197–213.

[6] C.H. Chang and R. Paige, *From regular expressions to DFA's using compressed NFA's*, Theoret. Comput. Sci., 178(1997), 1–36.

[7] J.-M. Champarnaud, *Evaluation of three implicit structures to implement nondeterministic automata from regular expressions*, Internat. J. Found. Comput. Sci., 13(1)(2002), 99–113.

[8] J.E. Hopcroft and J.D. Ullman, *Introduction to automata theory language and computation*, Addison Wesley, Reading Mass, 1979.

[9] J. Hromkovič, S. Seibert and T. Wilke, *Translating Regular Expressions into Small ϵ-free Nondeterministic Finite Automata*, J. Comput. System Sci., 62(2001), 565–588.

[10] L. Ilie and S. Yu, *Constructing NFAs by optimal use of positions in regular expressions*, Proc. of CPM2002, LNCS 2373(2002),279–288.

[11] L. Ilie and S. Yu, *Follow Automata*, Information and Computation, 186(2003), 140–162.

[12] G. Myers, *A Four Russians Algorithm for Regular Expression Pattern Matching*, J. ACM 39, 4(1992), 430–448.

[13] E. Myers and W. Miller, *Approximate Matching of Regular Expressions*, Bull. of Mathematical Biology, 51(1)(1989), 5–37.

[14] G. Navarro and M. Raffinot, *Compact DFA Representation for Fast Regular Expression Search*, Proc. of WAE2001, LNCS 2141(2001), 1–12.

[15] G. Navarro and M. Raffinot, *New Techniques for Regular Expression Searching*, Algorithmica, 41(2004), 89–116.

[16] H. Yamamoto, T. Miyazaki and M. Okamoto, *Bit-Parallel Algorithms for Translating Regular Expressions into NFAs*, IEICE Trans. Inf. & Syst., Vol.E90-D, No.2(2007), 418–427.

[17] B.W. Watson, *Taxonomies and Toolkits of Regular Language Algorithms*, Ph.D. dissertation, Faculty of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1995.

[18] B.W. Watson and R.E. Watson, *A Boyer-Moore-style algorithm for regular expression pattern matching*, Science of Computer Programming, 48(2003), 99–117.

[19] S. Wu, U. Manber and E. Myers, *A Sub-Quadratic Algorithm for Approximate Regular Expression Matching*, J. of Algorithms, 19(1995), 346–360.