An Efficient Multi-Attribute Pattern Matching Machine

Kazuaki Ando, Masami Shishibori and Jun-ichi Aoe

Department of Information Science & Intelligent Systems
Faculty of Engineering
Tokushima University
2-1 Minami-Josanjima-Cho
Tokushima-Shi 770
Japan

e-mail: {ando, aoe}@is.tokushima-u.ac.jp

Abstract. We describe an efficient multi-attribute pattern matching machine to locate all occurrences of any of a finite number of the sequence of rule structures (called matching rules) in a sequence of input structures. The proposed algorithm enables us to match set representations containing multiple attributes. Therefore, in proposed algorithm, confirming transition is decided by the relationship, whether the input structure includes the rule structure or not. It consists in constructing a finite state pattern matching machine from matching rules and then using the pattern matching machine to process the sequence of input structures in a single pass. Finally, the pattern matching algorithm is evaluated by theoretical analysis and the evaluation is supported by the simulation results with rules for the extraction of keywords.

Key words: string pattern matching, set representation, multi-attribute pattern matching, finite state pattern matching machine, matching algorithm

1 Introduction

String pattern matching [Aho75, Aho90, Knut77, Aoe84, Fan93, Boye77] is an important component of many areas in science and information processing. A string pattern matching machine has been applied to various fields such as the lexical analysis of a compiler [Aho86], voice recognition [Take93], bibliographic search [Aho75], spelling checking [Pete80], text editing and so on. Aho and Corasick describe a simple, efficient string pattern matching machine [Aho75] (hereafter called C machine) to locate all occurrences of finite number of keywords in a text string in a single pass. However, in the AC machine, the input is restricted to characters. In addition, a multi-attribute input is very useful for many applications, such as, extraction of keywords [Kimo91, Ogaw93], document processing [Ikeh93], and so on. Moreover, the multi-attribute pattern matching is necessary for the realization of higher pattern matching.

This paper describes an efficient multi-attribute pattern matching machine (hereafter called MAPM machine) to locate all occurrences of any of a finite number of

matching rules in a sequence of input structures and a method for constructing the multi-attribute pattern matching machine. The proposed algorithm enables us to match set representations containing multiple attributes.

In the following chapters, the multi-attribute pattern matching scheme is described in detail. Chapter **2** explains an efficient string pattern matching machine based on Aho and Corasick. Chapter **3** describes the multi-attribute matching rules for the MAPM machine and an efficient algorithm for these matching rules. Chapter **4** explains the construction of the goto, output and failure functions for the MAPM machine. Chapter **5** shows the theoretical estimations and experimental evaluations for the MAPM machine. Finally, in Chapter **6** the future research is discussed.

2 The Aho-Corasick Algorithm

This chapter explains an efficient string pattern matching machine, where a finite state string pattern matching machine based on Aho and Corasick [Aho75] locates all occurrences of any of a finite number of keywords in a text string.

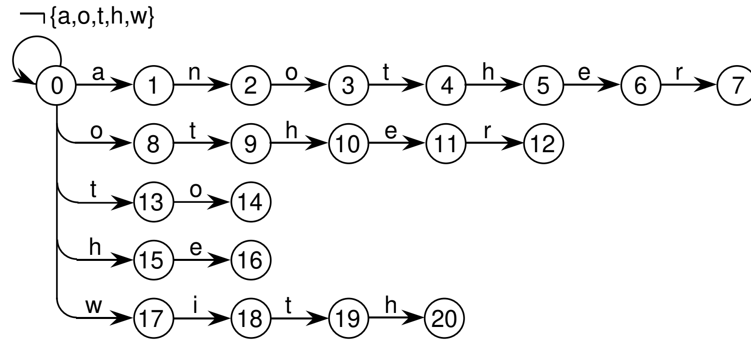
Let K_SET be a finite set of strings which we shall call keywords and let TX be an arbitrary string which we shall call the text string. The AC machine is a program which takes as input the text string TX and produces as output the locations in TX at which the keywords (elements of K_SET) appear as substrings. The AC machine is constructed as a set of states. Each state is represented by a number. The state number 0 represents an initial state.

With I as the set of input symbols, the behavior of the AC machine is defined by next three functions:

goto function $g : S \times I \rightarrow S \cup \{fail\}$,
failure function $f : S \rightarrow S$,
output function $output : S \rightarrow A$, subset of K_SET.

Figure 1 shows the functions, from Aho *et. al.*, used by the machine AC for the set of keywords $K_SET = \{$"another", "other", "to", "he", "with"$\}$. Here, $\neg\{$'a', 'o', 't', 'h', 'w'$\}$ denotes all input symbols other than 'a', 'o', 't', 'h' or 'w'.

The directed graph in Figure 1 (a) represents the goto function. For example, the transition labeled 'a' from state 0 to state 1 indicates that $g(0,\text{'a'}) = 1$. The absence of the arc indicates *fail*. The AC machine has the property that $g(0,\text{'}\sigma\text{'}) \neq fail$ for all input symbols σ. The behavior of the AC machine is summarized below.



(a) The goto function.

s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
f(*s*)	0	0	8	9	10	11	12	0	13	15	16	0	0	8	0	0	0	0	13	15

(b) The failure function.

s	*output* (*s*)
6	{ he }
8	{ another, other }
11	{ he }
12	{ other }
14	{ to }
16	{ he }
20	{ with }

(c) The output function.

Figure 1: The function of the AC machine.

Algorithm 1: The AC machine.
[Input]: A target text $TX = c_1c_2...c_n$, where each c_i, for $1 \leq i \leq n$, is an input symbol and an AC machine with goto function g, failure function f, and output function *output*.
[Output]: Locations at which the keywords occur in TX.
[Method]:

```
begin
   s ← 0;
   for i ← 1 until n do
      begin
         while g(s, c_i) = fail do s ← f(s);
         s ← g(s, c_i);
         if output(s) ≠ φ then
            print i, output(s);
      end
end
```

(Example 1) Consider the behavior of the AC machine that uses the functions in Figure 1 to process the text string "stothe". Since g(0,'s') = 0, the machine enters the state 0. Since g(0,'t') = 13, g(13,'o') = 14, the AC machine enters state 14,

advances to the next input symbol and emits $output(14)$, indicating that it has found the keywords "to" at the end of position 3 in the text string. In state 14 with the input symbol 't', the AC machine makes two state transitions in its operating cycle. Since $g(14,\text{'t'}) = fail$, the AC machine enters the state $8 = f(14)$. At this point, since $g(8,\text{'t'}) = 9$, the AC machine enters state 9. Hereafter since $g(9,\text{'h'}) = 10$, $g(10,\text{'e'}) = 11$, the AC machine enters state 11 and computes the matching operation after detecting keyword "he". (END)

The AC algorithm consists in constructing a finite state pattern matching machine from the keywords and then using the machine to process the text string in a single pass. Construction of the AC machine takes a time proportional to the sum of the lengths of the keywords.

3 A Multi-Attribute Pattern Matching Algorithm

This chapter explains an algorithm of an efficient multi-attribute pattern matching machine (called MAPM machine). The MAPM machine is a extension of the Aho-Corasick Algorithm. Section **3.1** describes the multi-attribute matching rules for the MAPM machine. In Section **3.2**, an efficient algorithm of these matching rules is presented.

3.1 Matching rules for the MAPM machine

Let $ATTR$ be the *attribute name* and let $VALUE$ be the *attribute value*. Let R be a finite set of pairs $(ATTR, VALUE)$, then we shall call R a *rule structure*. For example, the following attributes are considered.

STR : string, that is, word spelling.
CAT : category, or, a part of speech.
SEM : semantic information such as concepts and categories.

For example, the rule structure R of "doctor" is defined using the above attributes as follows:

$$R = \{(STR, \text{"doctor"}), (CAT, Noun), (SEM, Human)\}$$

If $RULE$ is the *matching rule* consisting of a sequence of rule structures, $RULE$ is defined as follows:

$$RULE = R_1 R_2 ... R_n (1 \leq n)$$

Let R_SET be a set of $RULE$.

(Example 2) Consider the following example of a R_SET.

$$R_SET = \{RULE_1, RULE_2\}$$
$$RULE_1 = R_1 R_3, RULE_2 = R_2 R_3$$
$$R_1 = \{(STR, \text{"male"})\}$$
$$R_2 = \{(STR, \text{"female"})\}$$
$$R_3 = \{(SEM1, Male), (SEM2, Imago)\}$$

$RULE_1$ is a rule to detect tautology expressions and $RULE_2$ is a rule to detect contradictory expressions. For example, $RULE_1$ can detect the expression of "male man" or "male bull" and so on, $RULE_2$ can detect the expression of "female ram" or "female stallion" and so on. By using the multiple attributes for pattern matching, it is easy to define an abstraction rule. (END)

Input structures to be matched by rule structures are also defined by the same set representation. N is used as the notation for input structures to distinguish them from R. In order to consider the abstraction of the rule structure, matching of the rule structure R and the input structure N are decided by the relationship such that N includes R ($N \supseteq R$).

3.2 A Matching Algorithm

Let α be a sequence of the input structures. The MAPM machine is a program which takes as input α and produces as output the locations in a α at which every $RULE$ in R_SET appears as subsequences of structures. The MAPM machine consists of a set of states. Each state is represented by a number. One state (usually 0) is designated as the initial state. The MAPM machine processes a α by successively reading the input structure N in a α, making state transitions and occasionally emitting an output.

Let S_e be a set of states and let J be a set of the rule structure R, then the behavior of the MAPM machine is defined by next three functions:

goto function $g_e : S_e \times J \to S_e \cup \{fail\}$,
failure function $f_e : S_e \to S_e$,
output function $output_e : S_e \to A_e, subset\ of R_SET$.

Figure 2 shows the functions used by the machine MAPM for a $R_SET = \{RULE_1, RULE_2, RULE_3, RULE_4\}$. Here $\neg$ indicates all input structures except R_1 and R_3.

As the AC machine, the goto function g_e maps a pair consisting of a state and an input structure into a state or the message $fail$. The directed graph in Figure 2 (a) represents the goto function. The failure function f_e maps a state into a state. The failure function is constructed whenever the goto function reports $fail$. Certain states are designated as output states which indicate that a $RULE$ has been found. The output function formalizes this concept by associating R_SET (possible empty) with every state.

In the MAPM machine, a confirming transition is decided by the relationship, whether the input structure N includes the rule structure R or not. Therefore, in order to confirm a transition from certain state to next_state using the relationship, we define a function CHECK($state, N$) as follows:

[Function CHECK($state, N$)]

For each transition $g_e(state, R) = next_state$ in the goto graph, if some transition labeled R which satisfies the relationship ($N \supseteq R$) exists, then it returns all $next_state$, otherwise it returns $fail$.

(Example 3) Suppose that $g_e(s_1, R_1) = s_2$ and $g_e(s_1, R_2) = s_3$ are defined in the goto graph and R_1 and R_2 are defined as follows:

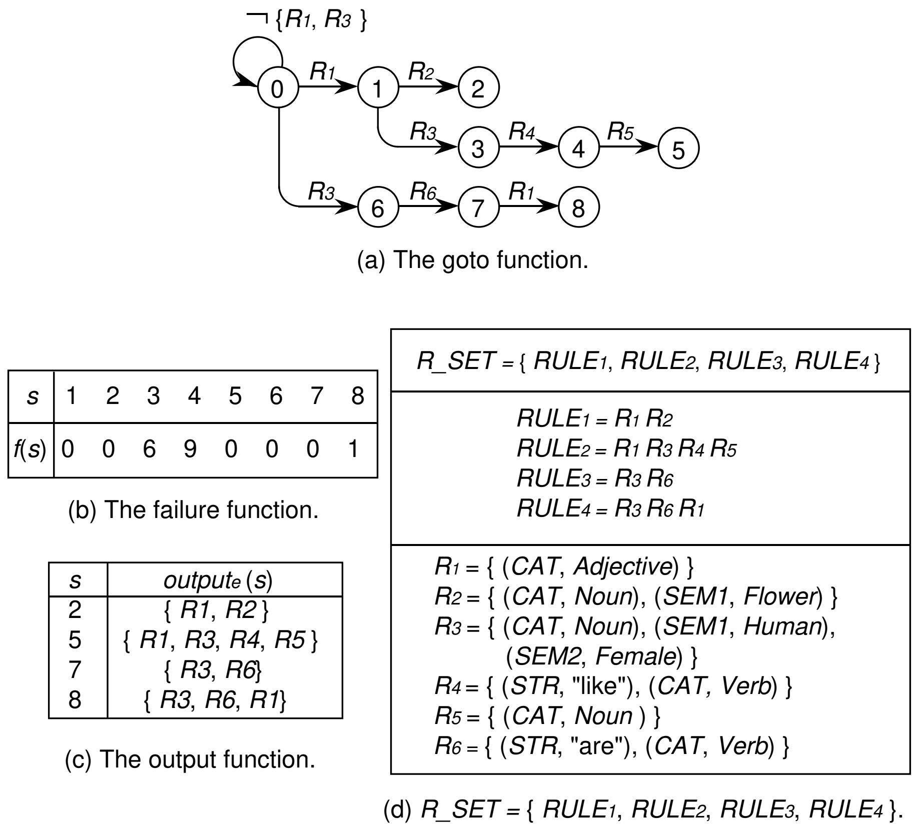


Figure 2: The functions of the MAPM machine for R_SET.

$$R_1 = (SEM, Human)$$
$$R_2 = (CAT, Noun))$$

Consider the behavior of the function CHECK to process the following N_1 and N_2.

$$N_1 = (STR, \text{"mother"}), (CAT, Noun), (SEM, Human)$$
$$N_2 = (STR, \text{"beautiful"}), (CAT, Adjective)$$

For N_1 the function CHECK returns s_2 and s_3 to satisfy $R_1 \subseteq N_1$ and $R_2 \subseteq N_1$. In the case of N_2 , $R_1 \subseteq N_2$ and $R_2 \subseteq N_2$ are not satisfied, therefore the function CHECK returns $fail$. (END)

Algorithm 2 summarizes the behavior of the MAPM machine.

Algorithm 2: A Multi-Attribute Pattern Matching machine (MAPM machine).
[Input]: A sequence of input structures $\alpha = N_1, N_2, ..., N_n$, where each N_i is an input structure and a MAPM machine with goto function g_e, failure function f_e, and output function $output_e$.
[Output]: Locations at which sequences of structure were extracted.
[Method]:

```
begin
  queue_1 ← 0;
  for i ← 1 until n do
    begin
      queue_2 ← empty;
      while queue_1 ≠ φ do
        begin
          temp ← empty;
          let state be the next state in queue_1;
          queue_1 ← queue_1 − state;
          while CHECK(state, N_i) = fail do state ← f_e(state);
          temp ← CHECK(state, N_i);
          queue_2 ← queue_2 ∪ temp;
          while temp ≠ φ do
            begin
              let element be the next state in temp;
              temp ← temp − element;
              if output_e(element) ≠ φ then
                begin
                  print i;
                  output_e(element)
                end
            end
        end;
      queue_1 ← queue_2
    end
end.
```

As we have mentioned before, in this algorithm, confirming transitions are decided by the relationship ($N \supseteq R$). Therefore, it has the possibility that some R such that N includes R exist among those transitions. In order to solve this problem, Algorithm 2 stores all states returned by the function CHECK in a first-in-first-out list denoted by the variable $queue_1$, and the MAPM machine continues to process for each state in $queue_1$.

(Example 4) Figure 2 shows the functions used by the MAPM machine for a $R_SET = RULE_1, RULE_2, RULE_3, RULE_4$. Consider the behavior of the MAPM machine that uses the functions in Figure 2 to process the sequence of the input structures $\alpha = N_1 N_2 N_3 N_4 N_5 N_6$.

$N_1 = (STR$, "beautiful"$), (CAT, Adjective)$
$N_2 = (STR$, "rose"$), (CAT, Noun), (SEM1, Flower)$

$N_3 = (STR,$ "and"$), (CAT, Conjunction)$
$N_4 = (STR,$ "pretty"$), (CAT, Adjective)$
$N_5 = (STR,$ "girl"$), (CAT, Noun), (SEM1, Human), (SEM2, Female)$
$N_6 = (STR,$ "are"$), (CAT, Verb)$

Since N_1 includes R_1 and N_2 includes R_2, CHECK(0,N_1) = 1 and CHECK(1, N_2) = 2, the MAPM machine enters state 2, advances to the next input structure and emits the $output_e(2)$, indicating that it has found the $RULE_1$ at the end of position 2 in the α. Since N_3 doesn't include R_1 or R_3, CHECK(0, N_3) = 0, the MAPM machine enters the state 0. Since N_4 includes R_1 and N_5 includes R_3, CHECK(0, N_4) = 1, CHECK(1, N_5) = 3, the MAPM machine enters state 3. In state 3 with the input structure N_6, the MAPM machine makes two state transitions in this operating fashion. Since N_6 doesn't include R_4, CHECK(3, N_6) = $fail$, the MAPM machine enters the state 6 = $f_e(3)$. At this point, since N_6 includes R_6, CHECK(6, N_6) = 7, the MAPM machine enters state 7, emits $output_e(7)$, indicating that it has found the $RULE_3$ at the end of position 6 in the α. (END)

The MAPM algorithm consists in constructing a finite state pattern matching machine from matching rules and then using the machine to process the sequence of input structures in a single pass.

4 Construction of Goto, Output and Failure Function for the MAPM Machine

This chapter explains the construction of the goto, output and failure function for the MAPM machine. Although the construction of the MAPM machine is similar to the construction of the AC machine [Aho75], the point which changes a transition label for the goto function to a set is different from the AC machine. Therefore, the decision of the same transition when the goto, failure and output functions are constructed is defined as follows:

[Condition for the equivalency of rule structures]
If the number of elements of rule structure R_1 and R_2 are equal and each of the elements $(ATTR, VALUE)$ of R_1 and R_2 are equal, then we define R_1 equal to R_2 ($R_1 = R_2$).

In order to examine the equivalency of rule structures, we define a function ARC($state, R$). The function ARC returns a $next_state$ that satisfies the condition for the equivalency of rule structures. The following shows the function ARC.

[Function ARC($state, R_2$**)]**
For each transition $g_e(state, R_1) = next_state$ in the goto graph, if $R_1 = R_2$, then the function ARC returns a $next_state$, otherwise it returns $fail$.

(Example 5) Suppose that $g_e(state_1, R_1) = state_2$ is defined in the goto graph and R_1 is defined as:

$$R_1 = (CAT, Noun), (SEMHuman)$$

Consider the behavior of the function ARC to process the following rule structures R_2 and R_3:

$$R_2 = (CAT, Noun), (SEM, Human)$$
$$R_3 = (CAT, Noun)$$

For R_2, ARC returns $state_2$ to satisfy the condition for the equivalency of rule structures $R_1 = R_2$. For R_3, the function CHECK doesn't satisfy the condition for the equivalency of rule structures $R_1 = R_3$ and therefore returns $fail$. (END)

Algorithm 3 summarizes the method for the construction of the goto and output functions for the MAMP machine. Algorithm 4 summarizes the method for the construction of the failure and output functions for the MAMP machine.

Algorithm 3: Construction of the goto function.
[Input]: Set of $RULE\ R_SET = RULE_1, RULE_2, ..., RULE_k$.
[Output]: Goto function g_e and a partially computed output function $output_e$.
[Method]: We assume $output_e(s)$ is empty when state s is first created, and $g_e(s, R) = fail$ if R is undefined or if $g_e(s, R)$ has not yet been defined. The procedure $enter(RULE)$ inserts into the goto graph a path that spells out $RULE$.

```
begin
   newstate ← 0;
   for i ← 1 until k do enter(RULE_i);
   for all R such that ARC(0, R) = fail do g_e(0, R) ← 0;
end

procedure enter(R_1, R_2, ..., R_m)
begin
   state ← 0; j ← 1;
   while ARC(state, R_j) ≠ fail do
      begin
         state ← ARC(state, R_j);
         j ← j + 1
      end
   for p ← j until m do
      begin
         newstate ← newstate + 1;
         g_e(state, R_p) ← newstate;
         state ← newstate;
      end
   output_e(state) ← R_1, R_2, ..., R_m
end
```

Algorithm 4: Construction of the failure function.
[Input]: Goto function g_e and output function $output_e$ from Algorithm 3.
[Output]: Failure function f_e and output function $output_e$.
[Method]:

```
begin
   queue ← empty;
   for each R such that ARC(0, R) = s ≠ 0 do
      begin
         queue ← queue ∪ s;
         f_e(s) ← 0
      end
   while queue ≠ empty do
      begin
         let r be the next state in queue;
         queue ← queue − r;
         for each R such that ARC(r, R) = s ≠ fail do
            begin
               queue ← queue ∪ s;
               state ← f_e(r);
               while ARC(state, R) = fail do state ← f_e(state);
               f_e(s) ← ARC(state, R);
               output_e(s) ← output_e(s) ∪ output_e(f_e(s))
            end
      end
end
```

(Example 6) Consider the construction of the MAPM machine for the $R_SET = RULE_1, RULE_2, RULE_3, RULE_4$ in Figure 2 (d). In the first place, the states and the goto function are determined according to Algorithm 3. Second, the failure function is computed according to Algorithm 4. The output function is constructed according to both Algorithms. The decision of the same transition when the goto, failure and output functions are constructed is decided by the equivalency of the rule structures.

Firstly, consider the construction of goto function. Adding the first rule $RULE_1$ to the graph, $g_e(0, R_1) = 1$ and $g_e(1, R_2) = 2$ are constructed as shown in Figure 3 (a). At this point, the output "R_1, R_2" is associated with state 2.

Adding the second rule $RULE_2$, Figure 3 (b) is obtained. Since ARC(0, R_1) = 1, notice that when the rule structure R_1 in $RULE_2$ is added there is already a transition labeled "R_1" from state 0 to state 1. Therefore it is not needed to add another transition labeled "R_1" from state 0 to state 1. After that, $g_e(1, R_3) = 3$, $g_e(3, R_4) = 4$ and $g_e(4, R_5) = 5$ are constructed as shown Figure 3 (b). The output "R_1, R_3, R_4, R_5" is associated with state 5.

Adding the third rule $RULE_3$, $g_e(0, R_3) = 6$ and $g_e(6, R_6) = 7$ are constructed as shown in Figure 3 (c). The output "R_3, R_6" is associated with state 7.

Adding the last rule $RULE_4$, Figure 3 (d) is obtained. The output "R_3, R_6, R_1" is associated with state 8. Here, since ARC(0, R_3) = 6 and ARC(6, R_6) = 7, we have been able to use the existing transition labeled R_3 from state 0 to state 6 and the

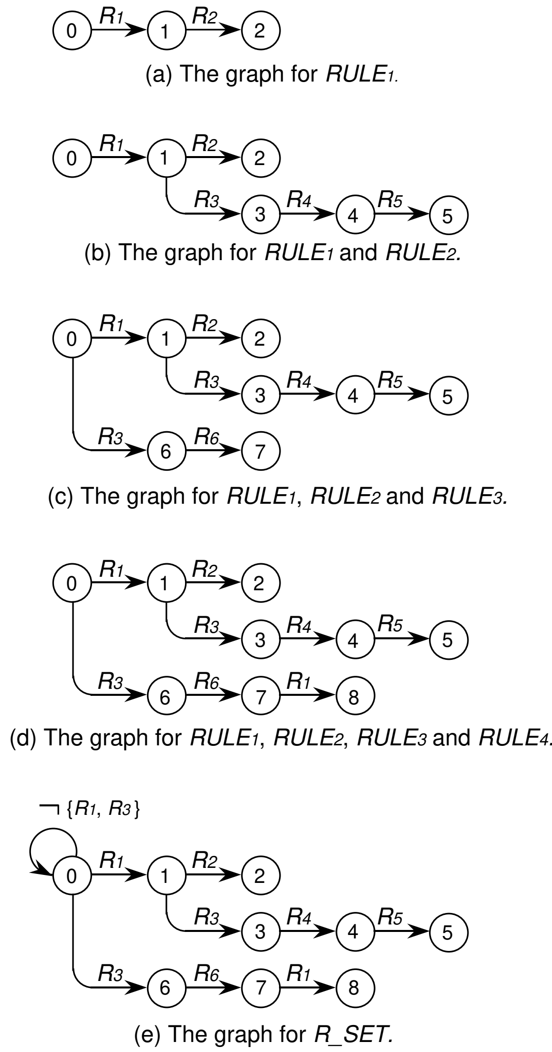


Figure 3: The construction of the goto graph for R_SET.

existing transition labeled R_6 from state 6 to state 7. To complete the construction of the goto function, a loop from state 0 to state 0 on all the input structures other than "R_1" and "R_3", is added to the graph. Finally, Figure 3 (e) is obtained.

Next, consider the construction of the failure function. To compute the failure function from Figure 3 (e), $f_e(1) = f_e(6) = 0$ is set since state 1 and 6 are the states of depth 1. Then the failure function for state 2, 3 and 7, the states of depth 2, is computed. To compute $f_e(2)$, $state = f_e(1) = 0$ is set; and since ARC(0, R_2) = 0, $f_e(2) = 0$ is determined. To compute $f_e(3)$, $state = f_e(1) = 0$ is set, and since ARC(0, R_3) = 6, $f_e(3) = 6$ is determined. To compute $f_e(7)$, state = $f_e(6) = 0$ is set; and since ARC(0, R_6) = 0, $f_e(7) = 0$ is determined. Continuing in this fashion, the failure function shown in Figure 2 (b) is obtained.

Finally, the goto, failure and output functions are constructed as shown in Figure 2 (b) and (c). (END)

5 Evaluation

In this chapter, the theoretical estimations and experimental evaluations for the MAPM machine are presented. Section **5.1**, describes the theoretical estimations for the MAPM machine, and in Section **5.2**, it is evaluated by applying it to the extraction of keywords [Kimo91, Ogaw93].

5.1 Theoretical Estimations

Suppose that the time complexity for confirming a transition in the MAPM machine is O(1). Let m be the length of sequence of input structures α. The time complexity of matching Algorithm 2 by the MAPM machine is O(m), because the matching cost of the AC machine is independent of the number of matching rules (keywords). However, the precise complexity for confirming a transition depends on the cost of the function CHECK and $queue_1$ in Algorithm 2.

Consider the time complexity of the function CHECK. By using the order of attributes names, sets of input and rule structures can be represented as the sorted-list whose nodes are denoted by (attribute-name, attribute-value, pointer). Similarly, the goto function is represented by the list structure. Let K be the kinds of attribute names. In the function CHECK, the time complexity for judging the relationship, whether the input structure includes the rule structure or not, is similar to the cost ($K + K = 2K$) of merging two sorted_lists of maximum length K into one list. Therefore, the time complexity of judging the relationship is O(K). Suppose that B is the maximum number of transitions leaving from certain state s. Then, the time complexity of the function CHECK is O(KB). Although this cost is more than that of the AC machine, an expression ability of rules for the MAPM machine is higher than the rule for the AC machine.

From the above observation, consider the time complexity of confirming a transition. Let D be the maximum number of the states in $queue_1$. The complexity is O(KBD), because $queue_1$ has all states returned by the function CHECK in Algorithm 2.

The time complexity for constructing the AC machine is proportional to the total length h of keywords. On the other hand, the time complexity for the construction of the MAPM machine is O(hK) in the worst-case, because confirming transitions depends on the function ARC and the time complexity for determining the equivalency of the rule structures in the function ARC is the same cost O(K) as the function CHECK.

5.2 Experimental Evaluations

The MAPM machine is evaluated by applying it to the extraction of keywords [Kimo91, Ogaw93]. For experimental evaluations, the MAPM machine has been implemented on a DELL OptiPlex GXMT5133 and it has been written in the C language.

In order to evaluate the efficiency of the proposed algorithm, we defined 112 rules for the extraction of keywords. Table 1 shows the information about the rules for the extraction of keywords. The information about $RULE$ are the values for each

Total number of kinds of attribute names	5
Information about *RULE*	
Total number	112
Average length	2.5
Average number of kind of attributes names	1.9
Construction time of the MAPM machine [sec]	0.543

Table 1: Information about *RULE* and construction time.

	Text1	Text2	Text3	Average
Number of words	411	233	197	280.0
Matching time [ms]	20.9	10.3	7.4	12.9
Number of extracted keywords	18	15	15	16.0

Table 2: The results of the simulation.

rule structure. From the average number, 1.9, of kinds of attribute names, it turns out that the attribute of each structure was abstracted effectively. It seems that the construction time (CPU time), 0.543 second, is practical.

Table 2 shows the results of the simulation using the above rule. To perform the simulation of the extraction of keywords, the following three texts were used.

Text1: General document, such as letter, journal, etc.
Text2: Abstract of a paper.
Text3: Document of patents.

From the average matching time in Table 2, the efficiency of the proposed algorithm could be verified. As shown by the theoretical estimations, the time complexity of the MAPM machine depends on the cost of the function CHECK. In the simulation, the attribute of each structure was abstracted effectively, such that the average of number of kinds of attribute names is 1.9. Consequently, good results could be obtained.

6 Conclusions

We have described an efficient method for multi-attribute pattern matching in this paper. A multi-attribute pattern matching is useful for many applications and the proposed algorithms enable the realization of higher pattern matching. The presented algorithm are evaluated by theoretical estimation and the experimental evaluation is supported by simulation results for the extraction of keywords.

In the proposed algorithms, it takes time to judge whether the input structure includes the rule structure or not. Therefore, as future extension, we are considering an efficient data structure and an efficient decision algorithm for the judging of the relationship, whether the input structure includes the rule structure or not. We

believe the proposed method is very useful for any existing and future computing system that would require an efficient multi-attribute pattern matching.

References

[Aho75] A.V. Aho – M.J. Corasick: Efficient String Matching : An Aid to Bibliographic Search. *Comm. ACM*, Vol.18, No.6, pp.333-340, 1975.

[Aho90] A.V. Aho: Algorithms for Finding Patterns in Strings. in J. Leeuwen, ed., Handbook of Theoretical Computer Science, Elsevier Science Publishers, pp.275-300, 1990.

[Aho86] A.V. Aho – R. Sethi – J.D. Ullman: Compilers Principles, Techniques and Tools. Reading MA: Addison-Wesley, ch. 2, 1986.

[Knut77] D.E. Knuth – J.H. Morris, Jr. – V.R. Pratt: Fast pattern matching in strings. *SIAM J.Comput.*, Vol.6, pp.323-350, June 1977.

[Kimo91] H. Kimoto: Automatic Indexing and Evaluation of Keywords for Japanese Newspapers. Ttrans. of the Institute of Electronics, Information and Communication Engineers of Japan, D-I Vol.J74-D-I, No.8, pp.556-566, Aug. 1991. (in Japanese)

[Aoe84] J. Aoe – Y. Yamamoto – R. Shimada: A Method for Improving String Pattern Matching Machine. *IEEE Trans. Software. Eng.*, Vol.10, No.6, pp.116-120, 1984.

[Fan93] J.-J. Fan – K.-Y. Su: An efficient algorithm for matching multiple patterns. *IEEE Trans. Knowledge and Data Eng.*, KDE-5, 2, pp.339-351, 1993.

[Pete80] J. L. Peterson: Computer Programs for Spelling Correction. Lecture Notes in Computer Science, New York: Springer-Verlag, 1980.

[Boye77] R.S. Boyer – J.S. Moore: A fast string pattern matching algorithm. *Comm. ACM*, Vol.20, No.10, pp.762-772, 1977.

[Ikeh93] S. Ikehara – E. Ohara – S. Takagi: Natural Language Processing for Japanese Text Revision Support System. Journal of Information Processing Society of Japan, Vol.34, No.10, pp.1249-1258, Oct. 1993. (in Japanese)

[Ogaw93] Y. Ogawa – M. Mochinushi – A. Bessho: A Compound Keyword Assignment Method for Japanese Texts. Research Report, Information Processing Society of Japan, 93-NL-97-15, pp.103-109, Sep. 1993. (in Japanese)

[Take93] Y. Takebayashi: Natural Language Processing in Speech Understanding and Dialogue. Journal of Information Processing Society of Japan, Vol.34, No.10, pp.1287-1296, Oct. 1993. (in Japanese)