ONE-WAY SEQUENTIAL SEARCH SYSTEMS AND THEIR POWERS

Arikawa, Setsuo
Research Institute of Fundamental Information Science, Kyushu University

https://doi.org/10.5109/13149

出版情報：統計数理研究. 19 (3/4), pp.69-85, 1981-03. Research Association of Statistical Sciences
バージョン：
権利関係：

ONE-WAY SEQUENTIAL SEARCH SYSTEMS AND THEIR POWERS

By

Setsuo ARIKAWA*

(Received November 1, 1980)

Abstract

One-way sequential search systems based on pattern matching machines are described. The powers of the systems are evaluated from a viewpoint of formal language theory. Their applicability to medical information processing is briefly discussed.

1. Introduction.

Sequential searches based on pattern matching have mainly been studied for text-editing and symbol manipulations [1], [2], [3], and have not been understood so practical for information retrievals, because they, to say the least, require time proportional to the length of file to be scanned. However the newly invented techniques of pattern matching, together with the recent development and diffusion of high-speed external memory units, are making it possible to apply the sequential searches to some practical information retrieval systems [4], [5], [6], [7].

The author, in co-operation with his colleagues, has developed a sequential search system TEXTIR at Kyushu University and at Edinburgh Western General Hospital in order to solve the following problems:

(1) Statistical study on natural languages, particularly on sentence patterns in abstracts of scientific documents.
(2) Study on automatic indexing.
(3) Development of information system for researchers' use.
(4) Patient record system for doctors' use.
(5) Support system for everyday business in laboratories.

We have constructed three variants of TEXTIR, paying attention to getting an efficient realization of the pattern matching machines and enlarging the functions as much as possible without trading the efficiency off to it. They are different in retrieving powers but have two distinctive features in common:

* Research Institute of Fundamental Information Science, Kyushu University 33, Fukuoka 812, Japan

(a) Retrievals of records of the form $x_1 \cdots x_2 \cdots x_3 \cdots x_4$. Every variant of TEXTIR can retrieve records of that form, where the triple dots $\cdots$ mean "some string". The function is particularly useful for check of some contextual conditions concerning the order of occurrences of keywords.

(b) Independence of record formats. Our systems do not assume any definite formats on records or files to be scanned, but assume the whole file of records as one very long string of characters. We can freely set up the so called output delimiters like the usual keywords and can consider strings intervening between a pair of output delimiters as virtual records.

In the present paper we shall describe the three variants of TEXTIR and discuss their powers from viewpoint of formal language theory, and then briefly refer to the realization of our systems and to their applicability to the medical information processing.

2. Preliminaries.

We begin with a simple formalism of records, files and related concepts, and then give a brief overview of Aho-Corasick's pattern matching machine which plays a key role in our search systems.

2.1 Formalism of concepts.

Let Σ be an alphabet, that is, a finite nonempty set of characters, Σ^* be the set of all words or strings over Σ including the empty word ε and Σ^+ be the set $\Sigma^* - \{\varepsilon\}$. Then a record is a string in Σ^* and a file is a subset of Σ^*.

When a file is organized or its elements are collected according to some definite rules, the set of rules can be taken as a grammar G and the file as a language $L(G)$ generated by G. Usually the file is a set of collected samples from $L(G)$ which possibly has infinitely many strings.

Retrieving some set of records which statisfy a query Q corresponds to making an intersection $L(G) \cap L(G')$, where G' is a grammar defined by the query Q. This understanding of the concepts will lead us to a new qualitative evaluation of the powers of one-way sequential search systems.

The class of grammars for queries is convenient if they have ability to define all reasonable subsets of $L(G)$ or Σ^*. We call a one-way sequential search system R-universal if the class of all languages defined by the system and the queries to it contains the class of all regular languages, i. e., languages recognized by finite automata.

2.2. Aho-Corasick machine.

Aho-Corasick's pattern matching machine is an efficient finite automaton which runs on text strings and detects all possible occurrences of keywords. Thus their retrieval system mainly consists of a constructor of the finite automaton from the set K of keywords and an interpreter which makes the automaton run along the text strings in Σ^* and detects the keywords in them.

The constructor is divided into two subsystems—the goto function constructor and the failure function constructor. We show an example of their machine in Fig. 1,

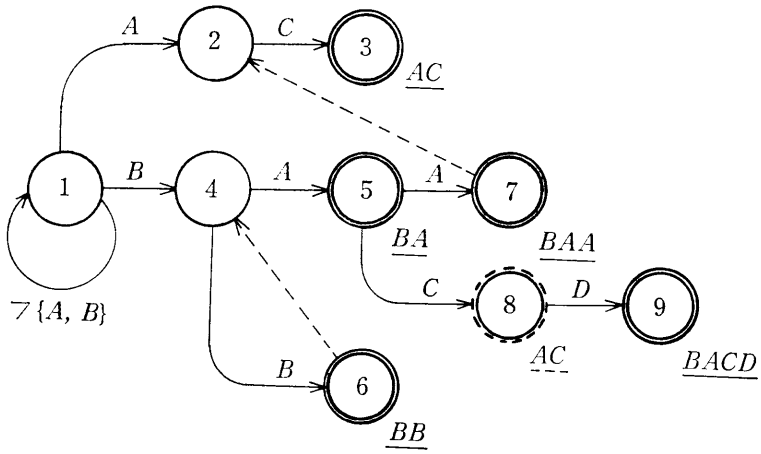
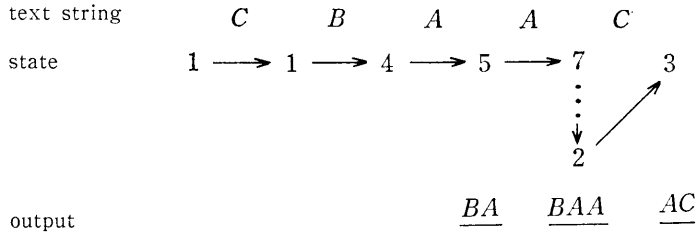


Fig. 1. A pattern matching machine by Aho-Corasick

where the set of keywords

$$K=\{AC,\ BA,\ BB,\ BAA,\ BACD\}$$

are given in this ordering, the unbroken arrows, circles and lines are constructed by the goto function constractor, and the broken ones by the failure function constructor. The machine has been optimized partly and the failure transitions, i. e., the broken arrows to the state 1 from all the states except the states 1, 6 and 7, should be added. The double circles mean the states with real outputs indicated by the underlined strings, and $\nabla\{A, B\}$ under the state 1 means all symbols except A and B. The state diagram works almost like that of finite automaton or sequential machine. The transitions indicated by the broken arrows correspond to the ε-moves; that is, the machine simply changes its state while holding the head in the same place. The machine behaves on text strings, say $CBAAC$ as follows:



The pattern matching machines of this type detect all possible occurrences of keywords in K by a one-way sequential search. In the example, the keywords BA, BAA and AC in K are detected in the text string $CBAAC$ by a one-way sequential search from left to right.

3. TEXTIR and its variants.

TEXTIR (TEXT Information Retrieval system) is a general purpose text infor-

mation retrieval system based on one-way sequential searches. This section describes an outline of TEXTIR and its variants.

3.1. Queries.

A query to TEXTIR consists of lists of input/output delimiters, keywords and logical formulae, and some other instructions such as selections of modes and object files.

3.1.1 Delimiters.

TEXTIR makes use of input/output delimiters, which are strings in Σ^+ assigned to the variables $A, B, \cdots, Y$. The output delimiters work for marking off some virtual records from a long text string, while the input delimiters work for telling the system when to evaluate the logical formulae. The system evaluates the formulae every time it detects input delimiters. According to a condition at the second output delimiter it displays the virtual record between the pair of output delimiters. The work and relation of the input/output delimiters are illustrated in Fig. 2, where u_1, u_2, u_3 are output delimiters, n_1, n_2 are input delimiters and r_1, r_2 are the virtual records. For example, in the usual retrieving, if the value of a logical formula is equal to 1 (true) at the output delimiter u_3, then the system will display the record r_2 as a reply to the query.

By virtue of these input/output delimiters we can use the system for very complicated retrievals if we are familiar with formats peculiar to the object files or text strings.

3.1.2 Keywords.

We can use two kinds of keywords—the usual ones in Σ^+ and ones with some occurrences of triple dots. The keywords with triple dots such as

$$x_1 \cdots x_2 \cdots x_3 \cdots x_4$$

work for defining or detecting virtual records of the form

$$\Sigma^* x_1 \Sigma^* x_2 \Sigma^* x_3 \Sigma^* x_4 \Sigma^*,$$

where x_1, x_2, x_3, x_4 are the usual keywords in Σ^+ without triple dots $\cdots$ as a substring. Thus the triple dots all together mean "some string" in Σ^*. Hereafter we use a symbol $\circ$ instead of the triple dots $\cdots$ in keywords to avoid a confusion, although we use them in real retrievals.

The function of keywords of this type, which is not new in symbol manipulations [4], is particularly useful for checking some contextual conditions as to the order of occurrences of keywords in records. For example, it is very difficult and sometimes impossible for systems without this function to retrieve only the records which contain

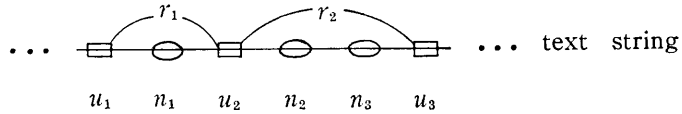


Fig. 2. Input/output delimiters

a keyword x followed by a keyword y, but it is quite simple in the case of our system; it is done by a keyword $x \circ y$.

The listing of keywords should be done by using variables $A, B, \cdots, Y$ issued by the system. We call the variables $A, B, \cdots, Y$ keyword variables.

3.1.3 Logical formulae.

A logical formula is a usual logical expression which is composed by using keyword variables $A, B, \cdots, Y$, logical operators $+$(for 'or'), $\cdot$ (for 'and') and $\sim$ (for 'not'), and some pairs of brackets [and]. All keyword variables in the formula should be defined in the preceding keyword listing. The listing of logical formulae is done after the formula variables $Z_1, Z_2, \cdots, Z_{15}$ issued by the system.

The system allows some abbreviations in listing of logical formulae. By '$Z_i=+$' a logical sum of all keyword variables already defined is produced, by '$Z_i=\cdot$' a logical product, and by '$Z_i=\sim$' a negation of sum.

Examples. Assume that keyword variables A, B, C, D are defined in keyword listing. Then

$$Z_1=[A+B]\cdot[C+D]$$

$$Z_2=\sim$$

$$Z_3=[A+B+C]\cdot[\sim D]$$

are examples of logical formulae, where $Z_2=\sim$ is equivalent to $Z_2=\sim[A+B+C+D]$.

3.2. Basic components.

TEXTIR mainly consists of three components, i.e., query editor, interpreter and output editor. This section describes their functions.

Query editor. The query editor receives the input/output delimiters, keywords and logical formulae from a user's terminal. It successively makes a goto function from these delimiters and keywords, in the course of which it encodes them into some positive integers with the properties:

$$\text{code (keyword)} < \text{code (input delimiter)} < \text{code (output delimiter)},$$

$$\text{code}(A) < \text{code}(B) < \cdots < \text{code}(Y),$$

where $A, B, \cdots, Y$ are variables issued by the system at the stage of listing of delimiters and keywords, and code(A), for example, means the code for the string assigned to the variable A. It also makes an output function in a table form

$$(\text{os}(i), \text{oc}(i))_{i=1,2,\cdots},$$

which means that a string with code oc(i) is to be detected in the state os(i). The editor makes another array ndot(i) to keep the number n of the substrings in the i-th keyword listing $D=x_1 \circ x_2 \circ \cdots \circ x_n$.

Then the editor constructs the failure function, and augments and sorts the output function to complete the pattern matching machine. The logical formuale are

transformed into Polish notations.

Interpreter. The interpreter makes the pattern matching machine run along a designated text string. Let keyword(i, j) be a table to check whether the j-th keyword in the i-th logical formula is detected. Then the interpreter works, using the code, tables and array, as follows:

(1) Clear keyword(i, j) by 0 for all i, j.

(2) Put keyword$(i, j)=1$ when the j-th keyword in the i-th query is detected.

(3) Evaluate all the logical formulae in the order of $Z_1, \cdots, Z_k$ when an input delimiter is detected. Then go to (1).

(4) Send a virtual record to the output editor as a reply to the i-th query, when an output delimiter is detected and if the i-th logical formula Z_i is true. Then go to (1) unless the text string is exhausted.

As seen in the algorithm above, TEXTIR processes many queries (up to 15) at a time. We remark that the order of occurrences of substrings $x_1, x_2, \cdots, x_n$ in a keyword of the form $x_1 \circ x_2 \circ \cdots \circ x_n$ is also checked by the interpreter, and hence the corresponding keyword(i, j) is checked if and only if the substrings $x_1, x_2, \cdots, x_n$ are marked in this ordering.

Output editor. The output editor edits the virtual records sent by the interpreter and displays them.

3.3 Three variants of TEXTIR.

Three variants of TEXTIR—system K, system W and system E have been constructed. The basic queries and functions mentioned above are all common to these three systems. This section describes other queries and functions proper to each variant.

3.3.1 System K.

The system K is a standard version of TEXTIR, in which the output delimiters also work as input ones. Hence users can omit the listing of input delimiters if they want to substitute the output delimiters for input ones. A virtual record intervening between the last two output delimiters is given as an output to a query with a logical formula Z, if Z becomes true at some input delimiters between the two output delimiters.

EXAMPLE 1. Let a text string be

$$\cdots acaabbaacaaabbaaaada \cdots$$

and a query be

OUTPUT DELIMITER

$A=ac$

$B=da$

INPUT DELIMITER

$A=ab$

KEYWORD

$A=aaaa$

$B=ba$

LOGICAL FORMULA
$Z_1 = A \cdot B$
$Z_2 = A \dot{-} B$

Then a virtual record *aaabbaaaa* is obtained as an output for Z_1, and two records *aabba* and *aaabbaaaa* are for Z_2.

3.3.2 System W.

As we shall see in Section 4, the function of the so called triple dots, which is one of the distinctive features of the system K, is more powerful than the sequential search system by Aho-Corasick. However K is not yet R-universal. In order to attain the universality and to enlarge the power much more, we have extended K.

The extended system W can restrict the universe Σ^* into some reasonably small subsets and make use of other variables than the keyword variables. The output delimiters can work properly.

Restrictions of universe. Queries in the system K define some subsets of Σ^*. The real Σ is the set of all characters used in computers. Hence it is not small. We sometimes want to restrict Σ into a small subset. Theoretically such a restriction is easy, but practically it requires a long query. We have realized this retriction in the system W by inhibiting the loop transitions in the initial state incuced by $\neg\{A, B\}$ say.

The system W has two retrieval modes {reset, noreset} related to the restriction. When the noreset mode is selected at the stage of querying and once the loop transition is taken place, the system ignores all keywords and input delimiters until it finds an output delimiter. By use of the noreset mode users can restrict Σ^* nearly into H^*, where H is the set of delimiters and keywords and is considerably small. The reset mode is the same as the implicit mode in the system K.

Use of formula variables in formulae. In the system W it is also possible to use the formula variables $Z_1, Z_2, \cdots, Z_{15}$ and a special variable @ called ε-variable in logical formulae and to indicate some formulae as temporary formula variables. The ε-variable @ plays a special role. Its value is true only when the system, leaving the last output delimiter, is about to start its job and becomes false after the system has started the searching for keywords. A logical formula ending with a slant/ simply works for a temporary memory and hence no records are edited for the query corresponding to the formula.

The system evaluates the set of logical formulae in the order of $Z_1, Z_2, \cdots, Z_{15}$, whenever it finds an input delimiter, and gives those virtual records between the last two delimiters as the outputs for the query when it finds an output delimiter and if the last value of the logical formula corresponding to the query is true.

The system clears all formula variables in use by $Z_i := \mathrm{ivz}(i)$ whenever it finds an output delimiter. The value ivz(i) is determined at the stage of query editing as follows. Let A_1 (=A), A_2 (=B), $\cdots$, A_n be keyword variables in use and $Z_1, Z_2, \cdots, Z_m$ be formula variables in use as well, and let

$$Z_i = f_i\ (@\,;\ A_1, \cdots, An\,;\ Z_1, \cdots, Z_{m-1}, Z_m)$$

for each i. Then the values of ivz are successively determined by

$$\begin{aligned}
&\mathrm{ivz}(1) = f_1(1\,;\,0,\,\cdots,\,0\,;\,0,\,\cdots,\,0,\,0)\\
&\mathrm{ivz}(2) = f_2(1\,;\,0,\,\cdots,\,0\,;\,\mathrm{ivz}(1),\,0,\,\cdots,\,0,\,0)\\
&\qquad\vdots\\
&\mathrm{ivz}(m) = f_m(1\,;\,0,\,\cdots,\,0\,;\,\mathrm{ivz}(1),\,\cdots,\,\mathrm{ivz}(m-1),\,0)\,.
\end{aligned}$$

The extended mode of evaluating is automatically adopted if one of the following conditions is satisfied:

(1) The ε-variable @ is used in some formulae.

(2) There are some formulae $Z_i=f_i$ in which a formula variable Z_j with $j \geqq i$ is used.

If none of them is satisfied, the system W works just like the system K. Hence a query such as

$$Z_i=f_i\ (A_1,\ A_2,\ \cdots,\ A_n)$$

in the system K is equivalent to a query

$$Z_i=Z_i+f_i\ (A_1,\ A_2,\ \cdots,\ A_n)$$

in system W. Hence the value of Z_i in the righthand term at time t is one at time $t-1$ and hence the variable works also as a memory. On the other hand, in a formula such as

$$Z_j=Z_i+f_j \quad (i<j)$$

the variable Z_i does not work as a memory.

EXAMPLE 2. The query to patient records illustrated in the finite state grammar in Fig. 3 is realized as follows. First, select noreset mode and omit the listing of output delimiters.

INPUT DELIMITER
A=b (a blank)

KEYWORD
A=TREAT
B=BLEEDING
C=LEFT
D=RIGHT
E=EXAM

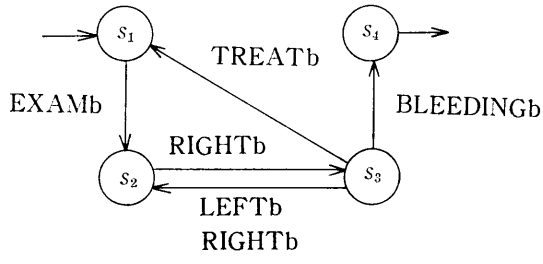


Fig. 3. Finite state grammar (by A.G. Hill)

LOGICAL FORMULA

$Z_1=Z_2\cdot B$

$Z_2=Z_3\cdot D/$

$Z_3=Z_4\cdot E+[C+D]\cdot Z_5/$

$Z_4=Z_5\cdot A+@/$

$Z_5=Z_2/$

Then the query corresponding to Z_1 realizes the grammar in Fig. 3.

3.3.3 System E.

The system E which is a final version of TEXTIR, is obtained from the system W by introducing several counters and by enlarging the operators in logical formulae. The counters are realized only by substituting keyword$(i, j):=$keyword$(i, j)+1$ and $Z:=Z+1$ for keyword$(i, j):=1$ and $Z:=1$, respectively, and hence they count how often a keyword is detected in between a pair of input delimiters and how often a logical formula is satisfied in between a pair of output delimiters.

A logical formula in the system E is composed by using the keyword variables, formula variables and ε-variable @, integers, logical operators ($,$, $\cdot$, $\sim$), comparison operators ($=$, $<=$, $>=$, $<$, $>$, $<>$), arithmetic operators ($+$, $*$, $-$, $/$), and brackets [,]. The precedence and the meaning of these operators are given in Fig. 4.

In the system E a variable α is considered to be true if $\alpha>0$, and hence a logical formula Z is satisfied if $Z>0$. The logical operators are interpreted as follows:

$$\alpha, \beta=1 \quad \text{if} \quad \alpha>0 \quad \text{or} \quad \beta>0,$$

$$\alpha\cdot\beta=1 \quad \text{if} \quad \alpha>0 \quad \text{and} \quad \beta>0,$$

$$\sim\alpha=1 \quad \text{if} \quad \alpha\leqq 0.$$

operator	precedence	meaning	
*	1		
/	1		arithmetic
+	2		operators
−	2		
=	3		
<=	3	$\leqq$	
>=	3	$\geqq$	comparison
<	3		operators
>	3		
<>	3	$\neq$	
$\sim$	4	not	logical
.	5	and	operators
,	6	or	

Fig. 4. Operators in system E

EXAMPLE 3. Let a text string be

$$dabdaaabbbdaaaabbbbaaaad,$$

and select noreset mode. Then consider the following queries:

OUTPUT DELIMITER
$A=d$

INPUT DELIMITER
$A=d$

KEYWORD
$A=a$
$B=b$
$C=ba$
$D=a\circ b\circ a$
$E=b\circ a\circ b$

LOGICAL FORMULA
$Z_1=[A=B]\cdot[\sim C]$
$Z_2=[A=2*B]\cdot D\cdot[\sim E]$

By a query Z_1 two virtual records ab and $aaabbb$ are retrieved, and by Z_2 a virtual record $aaaabbbbaaaa$ is retrieved.

4. Powers of the systems.

In this section we consider the retrieving powers of the three variants of TEXTIR as well as Aho-Corasick system from a formal language theoretic viewpoint. The Aho-Corasick system A is also taken as a variant of our TEXTIR, although it has no function of the triple dots and the delimiters. These systems can define many languages by various queries when we take files as Σ^*. To make our discussion clearer we refer to an interpretation of these systems and queries.

A keyword assignment $A=x$ in the system K as well as the system A can naturally be taken as a definition of a language

$$\|A\|=\|x\|\equiv\Sigma^*x\Sigma^*.$$

The interpretation can be extended to all formulae in the system K. A formula assignment $Z=f(\alpha_1, \alpha_2, \cdots, \alpha_n)$, where α_i $(1\leqq i\leqq n)$ are variables, defines a language

$$\|Z\|=\|f(\alpha_1, \alpha_2, \cdots, \alpha_n)\|$$

by the following interpretation:

$$\|\alpha_i+\alpha_j\|=\|\alpha_i\|\cup\|\alpha_j\|,$$

$$\|\alpha_i\cdot\alpha_j\|=\|\alpha_i\|\cap\|\alpha_j\|,$$

$$\|\sim\alpha_i\|=\Sigma^*-\|\alpha_i\|.$$

A similar interpretation is also valid in the other systems W and E. Hence let S

be a system of the four, Z be a possible query, i.e., a logical formula, in the system and let $L(Z/S)$ be a language

$$L(Z/S)=\|Z\| .$$

Let $C_A(\Sigma)$, $C_K(\Sigma)$, $C_W(\Sigma)$ and $C_E(\Sigma)$ be the classes of languages over Σ defined in the systems A, K, W and E, respectively, and by all possible queries, e.g.

$$C_K(\Sigma)=\{L(Z/K)\,;\,Z\} ,$$

and let $C_R(\Sigma)$ be the class of regular languages over Σ.

REMARK. In this section for the sake of simplicity, we assume that output delimiters are not in Σ^* and release the restrictions on the length and the number of keywords, delimiters and formulae. As a matter of course they must be finite.

THEOREM 1.

(1) $C_A(\Sigma)=C_K(\Sigma)\equiv C_W(\Sigma)\subseteq C_E(\Sigma)$ *if* $\sharp(\Sigma)=1$

(2) $C_A(\Sigma)\subseteq C_K(\Sigma)\subseteq C_W(\Sigma)\subseteq C_E(\Sigma)$ *if* $\sharp(\Sigma)>1$.

PROOF. From the definitions of the systems it is clear that

$$C_A(\Sigma)\subseteq C_K(\Sigma)\subseteq C_W(\Sigma)\subseteq C_E(\Sigma)$$

for any Σ.

(1) In case $\sharp(\Sigma)=1$, clearly

$$\|x_1\circ\cdots\circ x_n\|\equiv\|x_1\|\cdots\|x_n\|=\|x_1\cdots x_n\| .$$

Hence

$$C_A(\Sigma)=C_K(\Sigma) .$$

Let us show that a language $L_1=(aa)^*$ is not in $C_A(\{a\})$. Suppose L_1 is in $C_A(\{a\})$ and a query $Z=f(A_1, A_2, \cdots, A_n)$ with a list of keywords $A_1=a^{r_1}=aa\cdots a$ (r_1 times), $A_2=a^{r_2}$, $\cdots$, $A_n=a^{r_n}$ defines L_1 in the system A, i.e.,

$$L(Z/A)=(aa)^* .$$

The formula can be written in a sum-of-products form

$$Z=f(A_1, A_2, \cdots, A_n)=\sum_{i} A_1^{e_{i1}}\cdot A_2^{e_{i2}}\cdots A_n^{e_{in}} ,$$

where $e_{ij}=0$ or 1 for which $A_i^0=\sim A_i$ and $A_i^1=A_i$. If $e_{ij}=0$, then

$$\sharp(\|A_1^{e_{i1}}\cdots A_j^{e_{ij}}\cdots A_n^{e_{in}}\|)\leqq r_j ,$$

because

$$\|A_1^{e_{i1}}\cdots A_j^{e_{ij}}\cdots A_n^{e_{in}}\|\subseteq\|A_n^0\|=a^*-\|A_j\| .$$

Hence there is at least one term

$$A_1^{e_{k1}}\cdots A_n^{e_{kn}}$$

with $e_{kj}=1$ for all j. Then we have

$$\begin{aligned}\|A_1^{e_{k1}}\cdots A_n^{e_{kn}}\|&=\|a^{\max(r_1,\cdots,r_n)}\|\\&=a^{\max(r_1,\cdots,r_n)}\cdot a^*\end{aligned}$$

and we have

$$a^m,\ a^{m+1}\in\|A_1^{\varepsilon_{k_1}}\cdots A_n^{\varepsilon_{k_n}}\|\subseteq L(Z/A)$$

for all $m\geqq\max(r_1,\cdots,r_n)$, which is a contradiction. Hence L_1 is not in $C_A(\{a\})$.

(2) First let us show that $C_A(\Sigma)\neq C_K(\Sigma)$ for $\Sigma=\{a,b\}$. To show this we prove that a language

$$L_2=\Sigma^*a\Sigma^*a\Sigma^*$$

is in $C_K(\Sigma)$ but not in $C_A(\Sigma)$. Clearly $L_2\in C_K(\Sigma)$, since $L_2=L(Z/K)$ for $Z=A$ with $A=a\circ a$. Now suppose $L_2\in C_A(\Sigma)$ and a query $Z=f(A_1,A_2,\cdots,A_n)$ with a listing of keywords $A_i=x_i$ $(1\leqq i\leqq n)$ defines L_2 in the system A. The logical formula Z can be written again in a sum-of-products form

$$Z=f(A_1,A_2,\cdots,A_n)=\sum_i A_1^{\varepsilon_{i_1}}\cdot A_2^{\varepsilon_{i_2}}\cdots A_n^{\varepsilon_{i_n}}.$$

We may assume without any loss of generality that

$$\|A_1^{\varepsilon_{i_1}}\cdots A_n^{\varepsilon_{i_n}}\|\neq\phi$$

for all i. Without losing generality we may assume that $A_1,\cdots,A_r$ are variables to each of which a keyword with at most one a is assigned and $A_{r+1},\cdots,A_n$ are variables to each of which a keyword with at least two a's is assigned. Suppose $ab^ka\in\|S\|$ for a sufficiently large k and a term S. Then all variables $A_{r+1},\cdots,A_n$ should be negative in S. Since

$$b^kab^k\in\|A_1\cdots A_r\|\quad\text{and}\quad b^kab^k\in\|[\sim A_{r+1}]\cdots[\sim A_n]\|,$$

we have

$$b^kab^k\in\|S\|\subseteq L(Z/A),$$

which is impossible. This completes the proof that L_2 is not in $C_A(\Sigma)$. By $(aa)^*\notin C_A(\{a\})=C_K(\{a\})$ in the proof (1) and by Theorem 4 below, obviously $C_K(\Sigma)\neq C_W(\Sigma)$. The relation $C_W(\Sigma)\neq C_E(\Sigma)$ follows from Theorem 4 and Example 3. In fact,

$$\|Z_1\|=\{a^nb^n\ ;\ n\geqq1\},$$

$$\|Z_2\|=\{a^nb^na^n\ ;\ n\geqq1\},$$

which are not regular and hence not in $C_W(\Sigma)$. □

From Theorem 1 we immediately have the following:

COROLLARY 2. *The function of triple dots in the system K is independent of the logical operators.*

THEOREM 3. *Let $C_K^{(n)}(\Sigma)$ be the class of all languages over Σ each of which is defined in the system K by a query with at most n uses of triple dots in one keyword. Then for Σ with $\sharp(\Sigma)>1$,*

$$C_A(\Sigma)=C_K^{(0)}(\Sigma)\subsetneqq C_K^{(1)}(\Sigma)\subsetneqq C_K^{(2)}(\Sigma)\subsetneqq\cdots\subsetneqq C_K(\Sigma).$$

PROOF. By a discussion analogous to the proof of (2) of Theorem 1, we can prove that a language $\|a\circ a\circ\cdots\circ a\|$ (n times) is in $C_K^{(n)}(\Sigma)$ but not in $C_K^{(n-1)}(\Sigma)$. □

THOREM 4.

$$C_W(\Sigma)=C_R(\Sigma)\qquad \textit{for any}\quad \Sigma.$$

PROOF. From the definition of the system W, obviously $C_W(\Sigma)\subset C_R(\Sigma)$. Hence it suffices to show that every finite automaton can be realized in the system W. Let $M=(S, \Sigma, \delta, s_1, F)$ be a finite automaton. We define a query in W in which $L(M)$ is defined. Let $S=\{s_1, s_2, \cdots, s_m\}$, $\Sigma=\{a_1, a_2, \cdots, a_n\}$ and $A_1(=A)$, $A_2(=B)$, $\cdots$, A_n be keyword variables. Set $a_1, a_2, \cdots, a_n$ as input delimiters and assign a_i to each keyword variable by $A_i=a_i$, and make the logical formulae as follows:

$$Z_1=\sum_{(j,k):\delta(s_k,a_j)=s_1} A_j\cdot Z_{m+k}+@/$$

$$Z_i=\sum_{(j,k):\delta(s_k,a_j)=s_1} A_j\cdot Z_{m+k} \quad (1<i\leqq m)$$

$$Z_{m+i}=Z_i/ \quad (1<i\leqq m),$$

$$Z_{2m+1}=\sum_{s_i\in F} Z_i .$$

The system evaluates in succession the series of logical formulae from Z_i to Z_{2m+1} each time it finds an input delimiter and the formula variables on the right work as memories, their values being ones at the last evaluations, and so we can easily verify that

$$L(Z_{2m+1}/W)=L(M). \quad \square$$

EXAMPLE 4. In the above proof we have used plenty of formula variables for the sake of simplicity. We can reduce the variables nearly down to 3/4 of them. We give an example for a finite automaton in Fig. 5. Let a, b be input delimiters and $A=a$, $B=b$ be a list of keywords, and then make logical formulae as follows:

$$Z_1=[Z_2\cdot A]+[Z_3\cdot B]+@/$$

$$Z_2=Z_4\cdot[A+B]/$$

$$Z_3=[Z_3\cdot A]+[Z_5\cdot B]$$

$$Z_4=Z_1/$$

$$Z_5=Z_2/$$

Then we have $L(Z_3/W)=L(M)$.

Theorem 4 asserts that the system W and hence the system E are R-universal, and applicable to a syntax checker of finite state languages.

As we have just seen in the proof of Theorem 1, the system E is so powerful that it defines context-sensitive languages such as $\{a^nb^na^n;\ n\geqq 1\}$. Since the series of logical formulae are evaluated within $t(n)$ steps, the system E finishes its retrieving within $t(n)\cdot n$ steps, where n is the length of a virtual record and t is a primitive recursive function of n. Hence we have the following theorem:

THEOREM 5. *The class $C_E(\Sigma)$ is properly contained in the class of recursive sets over Σ.*

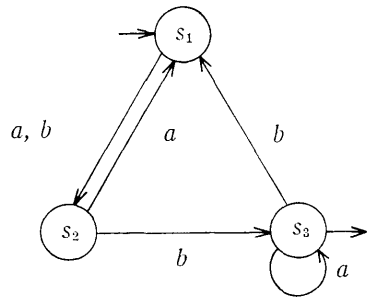


Fig. 5. Finite automaton M

5. Implementations.

The systems K and E have been realized in a small scale computer PANAFACOM U-400, and FACOM M200 in Fortran at Kyushu University, and the system W in GEC 2050 in Coral 66 at Western General Hospital. This section briefly describes how the transition tables of pattern matching machines and the triple dots processor have been realized in these computers.

5.1 Transition tables.

The transition diagram such as in Fig. 1 is realized in an array of size $3\times n$ if Aho-Corasick's algorithm is directly applied, where n is nearly equal to the number of states in a transition diagram. We have decreased the size to $2\times n$ by deleting the row of goto function. The deleted goto function is easily recovered by a simple function $\mathrm{goto}(k)=k+1$ for any state k. The original algorithm needs another big array as a working space at the stage of constructing failure function. We have deleted this working space while keeping the program in the same size and in nearly the same speed. And keeping the transition table in the same size, we have also deleted the redundant transitions as many as possible. Fig. 6 illustrates how the transition table of the pattern matching machine in Fig. 1 is constructed. In Fig. 6 the strings of symbols beside the double arrows show the current inputs to the constructor, and the 0's in the rows of failure function indicate real transitions to state 1, while 1's indicate transitions to state 1 by the ε-moves.

5.2 Triple dots processing.

In principle it is possible to construct directly finite automaton which accepts a language defined by $x_1\circ\cdots\circ x_n$, but it forces us to abandon our present approach and to build a new theory. The directly constructed finite automaton will use many states. Hence we have adopted a method of marking the order of occurrences of sub-keywords $x_1, x_2, \cdots x_n$ at the stage of retrieving by the interpreter.

The interpreter memorizes the largest number j of the sub-keywords in the keyword $x_1\circ x_2\circ\cdots,\circ x_n$ that have already occurred. This works for avoiding twice or more checking of the occurrences of such sub-keywords in the same keyword with triple dots $\circ$. However, the interpreter still works too sensitively, because the pattern

⇓AC

'state'	1	2	3
input	A	C	∀
failure			

⇓BA

'state'	1	2	3	4	5	6
input	A	C	∀	B	A	∀
failure	4					

⇓BB

'state'	1	2	3	4	5	6	7	8
input	A	C	∀	B	A	∀	B	∀
failure	4				7			

⇓BAA (Insert 'A')

'state'	1	2	3	4	5	6	7	8	9
input	A	C	∀	B	A	A	∀	B	∀
failure	4				8				

⇓$BACD$

'state'	1	2	3	4	5	6	7	8	9	10	11	12
input	A	C	∀	B	A	A	∀	B	∀	C	D	∀
failure	4				8	10						

⇓ (Complete the failure function)

'state'	1	2	3	4	5	6	7	8	9	10	11	12
input	A	C	∀	B	A	A	∀	B	∀	C	D	∀
failure	4	1	1	0	8	10	2	1	5	2	3	1

⇓ (Optimize the transition table)

'state'	1	2	3	4	5	6	7	8	9	10	11	12
input	A	C	∀	B	A	A	∀	B	∀	C	D	∀
failure	4	1	1	0	8	10	2	0	5	4	1	1

Fig. 6. Construction of transition table

matching machine finds out all possible occurrences of (sub-) keywords. For example the keyword variable A with an assignment $A=aa\circ aa$ is already marked when the system reads the third occurrence of 'a' in a text $aaaa$. Hence strictly speaking, our triple dots define another combination Δ instead of $\circ$. It is easy to show the following fact:

PROPOSITION 6.

(1) $x\Delta y\equiv x\circ y$ *iff there are no* x_1, u, $y_1(\neq\varepsilon)$, v, $w(\neq\varepsilon)$ *such that*

$$x=x_1u \quad \& \quad y=uy_1 \quad or \quad y=vxw,$$

(2) $x^{(1)}\Delta x^{(2)}\Delta\cdots\Delta x^{(|x|)}\Delta x\Delta y\equiv x\circ y$,

where $x^{(i)}$ *is the i-th symbol in* x *and* $|x|$ *denotes the length of* x, *and* $\equiv$ *reads "works equivalently".*

In accordance with this proposition we have added a procedure to check the condition (1), and to produce the left-hand expression of (2) if the condition (1) holds. Thus the assignment above is automatically replaced by $A=aJaJaaJaa$.

6. Discussion.

At Western General Hospital, a formal grammar for diabetic records has been developed and a syntax checker of records written in the grammar has also been developed. The records have been stored after syntax checking by the checker. The direct motive of realizihg the system W at the hospital was to support the patient record system and to supply a preliminary system of the records to doctors. In this section we briefly discuss a characteristic of patient record systems and an applicability of our systems to such systems.

Compared with other information systems such as document retrieval systems, the patient record system has the following specific features:

(1) Records should be valid and exact. In general, medical data should be stored after some careful checking of validity.

(2) Indexing of records is difficult and nearly meaningless. Since the records are written in some fixed medical terms, they look quite similar to each other and almost all terms are used for almost all records. Hence the usual indexing will prove meaningless and redundant. However, fortunately, doctors' interest will naturally control the file of records to an appropriate amount for the sequential searches.

(3) Each record will increase the items each time a patient has an examination and a treatment. The order of items in a particular record is specially significant. The record is to be taken as a history of the patient concerning his examinations and treatments. In this sense the record is a kind of time series of patterns by strings of symbols.

(4) Users of the information system are not always familiar with the specification of the system. Some grammar for presenters of data should be given in the form not so far away from natural language, and the question-answering should easily be understandable. On the other hand, by analysis of several queries of doctors to the patient record system, we can summarize their requests on the system as follows:

(a) To see an individual patient record. That is to find out an appropriate segment out of one record, to read the n-th examination or treatment in one record for the next treatment and to see some other personal data necessary for the treatment.

(b) To find out all records which satisfy some conditions or have some properties.

(c) To make statistics on their patients, e.g., to make tables and to draw figures.

(d) To organize sets of records under some criteria, classifications or viewpoints.

Here we should note that patient records range from 50 to more than 1000 in length. Hence it will not be so easy for doctors to find out particular places of records unless they use any mechanized system. The requests in (a) will be satisfied by a sequential

search systems such as the systems W and E, and so will be the requests (b) and (c). The request (d) will be satisfied by a kind of MIR-RF system. The notion of tree representation and organization of concepts in that system will be valid for the patient record system.

Acknowledgements

The main part of this work was done while the author was visiting Machine Intelligence Research Unit, University of Edinburgh and working at Edinburgh Western General Hospital supported by the Japan Society of Promotion of Science. The author should like to express his sincere thanks to Prof. D. Michie, Dr. H. R. A. Townsend and Dr. A. G. Hill for their advice, discussion and encouragement. He is also grateful to Dr. S. Takeya and Mr. S. Miyano for the constant co-operation.

References

[1] KNUTH, D. E., MORRIS, J. H. JR. and PRATT, V. R.: Fast pattern matching in strings, TR CS-74-440, Stanford Univ., (1974).

[2] AHO, A. V., HOPCROFT, J. E. and ULLMAN, J. D.: The design and analysis of computer algorithms, Addison-Wesley, (1974).

[3] FARBER, D. J., GRISWOLD, R. E. and POLONSKY, I. P.: SNOBOL, A string manipulation language, J. ACM 11, (1964), 21-30.

[4] AHO, A. V. and CORASICK, M. J.: Efficient string matching: An aid to bibliographic search, C. ACM 18, (1975), 333-340.

[5] ARIKAWA, S., TAKEYA, S. and ISHIBASHI, M.: On an information retrieval system using pattern matching machines, Proc. Japan Inf. Processing Soc., (1976), 253-254 (Japanese).

[6] BOYER, R. S. and MOORE, J. S.: A fast string search algorithm, C. ACM 20, (1977), 762-772.

[7] KAMBAYASHI, Y., NAKATSU, N. and YAJIMA, S.: Hierarchical pattern matching algorithms for strings, Trans. Inst. Electronics and Communication Engineers of Japan, J62-D, (1979), 341-347 (Japanese).

[8] TAKEYA, S. and ARIKAWA, S.: TEXTIR—A text information retrieval system using pattern matching machines, Res. Rept. Res. Inst. Fund. Inform. Sci. Kyushu Univ., 83, (1978), 1-16.

[9] ARIKAWA, S., HILL, A. G. and TOWNSEND, H. R. A.: An information retrieval system based on one-way sequential searches, Report at MIRU, Univ. of Edinburgh (1978, unpublished).

[10] ARIKAWA, S.: A one-way sequential search system and its applicability to medical information processing, Proc. 13th Hawaii International Conference of System Sciences, vol. 3, (1980).

[11] ARIKAWA, S. and KITAGAWA, T.: Multistage information retrieval system based upon researcher files, Proc. 2nd USA-Japan Comp. Conf., (1975), 149-153, and also in Res. Rept. Res. Inst. Fund. Inform. Sci., Kyushu Univ. 51, (1975), 1-34.