SIGMA - AN INFORMATION SYSTEM FOR RESEARCHERS USE

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

Shinohara, Takeshi
九州大学大学院総合理工学研究科情報システム学専攻

Shiraishi, Shuji
Department of Information Systems, Interdisciplinary Graduate School of Engineering Science, Kyushu University

Tamakoshi, Yasushi
Department of Information Systems, Interdisciplinary Graduate School of Engineering Science, Kyushu University

https://doi.org/10.5109/13339

出版情報：Bulletin of informatics and cybernetics. 20 (1/2), pp.97-114, 1982-03. Research Association of Statistical Sciences
バージョン：
権利関係：

Bulletin of Informatics and Cybernetics 20 (1982)

SIGMA—AN INFORMATION SYSTEM FOR RESEARCHERS USE

By

Setsuo ARIKAWA[1,2], Takeshi SHINOHARA[2], Shuji SHIRAISHI[2]

and

Yasushi TAMAKOSHI[2]

(Received November 27, 1981)

Abstract

This paper describes an information system SIGMA implemented on the computer at our University Computer Center. The system creates, for each user, a MEMO space for his private notebook and SIGMA spaces for sharing data with other users. The system assumes, as data structure, only the strings of characters, and hence the main access method is a linear search. We developed a general purpose linear search method called one-way sequential search which extensively makes use of pattern matching machines. Communications between users and the system are automatically recorded in log files, which function as command procedures.

1. Introduction

Scientific information system such as information retrieval systems came into practical use, and there has been understood the importance of such system in scientific research activities. Researchers can now retrieve some database which covers reasonably authorized data in their study fields. However this passive way of use of information systems may not satisfy the researchers who are at once users and producers of scientific data.

To solve the problem, the M-committee of the Special Research Project, named Formation Process of Information Systems and Organization of Scientific Information, proposed in 1977 three levels of scientific information systems which manage PRF (Private Researcher Files), UDL (User Data Library), and CDL (Center Data Library) respectively [1]. Several years before the proposal, one of the authors (S. A.) with his colleagues studied the so called PRF and UDL systems and developed an experimental system MIR-RF at their laboratory [2], [3].

1) Research Institute of Fundamemtal Information Science, Kyushu University 33, Fukuoka 812.
2) Department of Information Systems, Interdisciplinary Graduate School of Engineering Science, Kyushu University 39, Fukuoka 812.

Accepting the proposal and considering our study and experience, we have newly developed an information system SIGMA at our University Computer Center. The SIGMA system was intended to satisfy the following requirements:

1) The system be able to cope with various types of data.

2) The system be easily able to store, retrieve and edit researcher's private data.

3) The system be able to work for constructing and organizing a researcher group's data.

4) The system be able to retrieve some authorized databases and to file necessary data from them.

5) Data typed by users, including commands and their operands, be never lost but be effectively used.

From these viewpoints we have adopted, at first, strings of characters as a basis of data structure. This might sound negative, but the structure of strings is the simplest and most fundamental, and has ability to compose any other data structure. As a basis of access method we adopted the one-way sequential search method [5] based on the pattern matching machines [4], which enables complicated retrievals, editing and refiling of retrieved data. The requirements 3) and 4) have been solved by simplifying copying of files within user's space between users' spaces, and between user's space and external data set which is, of course, still in the computer system. We have devised a physical data structure to prevent the stored files from vanishing by computer system's crash, user's careless hanging up and so on. Finally all communications, excepting some commands, between users and our system are automatically recorded in a log file, which functions as a kind of command procedure.

2. Spaces and Files

First we describe the basic organization of spaces and files of SIGMA system implemented on FACOM M-200 at Computer Center, Kyushu University. The SIGMA system is a computer program which creates, when it is initially invoked, directly accessible data sets under the user identifier, and works on them. Thus the system assumes no proper area in the computer.

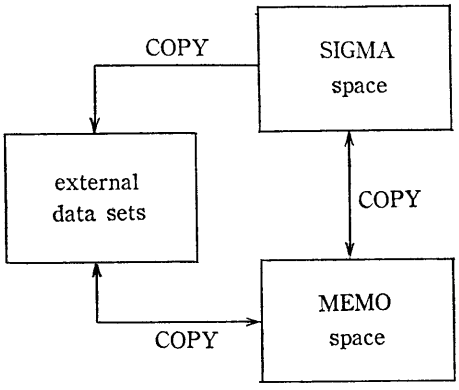


Fig. 1. Spaces in SIGMA System

We call the data set, which SIGMA system creates and manages, as a space. SIGMA system makes use of two kinds of spaces, MEMO space and SIGMA space, and also the usual data sets in the computer system which we call external data sets (See Fig. 1). As in the figure, files are copied by the COPY command.

2.1. MEMO Spaces

A MEMO space obligatory on any user of SIGMA is divided into five subspaces W, M, T, L and D as in Fig. 2, where W is a subspace for work files, M is for the user's files with proper names, T is for work files in executing SEARCH command, L is for log files and D is for deleted files. All the subspaces but M are of a stack structure with bottom, and in each subspace the newest file is put on the top of the stack. When the number of files in one of subspaces W, T and L exceeds a constant, the oldest one is to be moved to the top of the subspace D. When MEMO space is exhausted, the system will actually delete files in D in succession from the bottom to the top to acquire necessary area. These garbage collections are made mostly in real time while commands are executing.

2.3. Physical Structure of Files

Files in the subspace M are maintained by a B-tree which is constructed also in the MEMO space. The B-tree used in SIGMA system is specially designed so that it efficiently works in sorting of records. In fact, we avoided shifts of records within

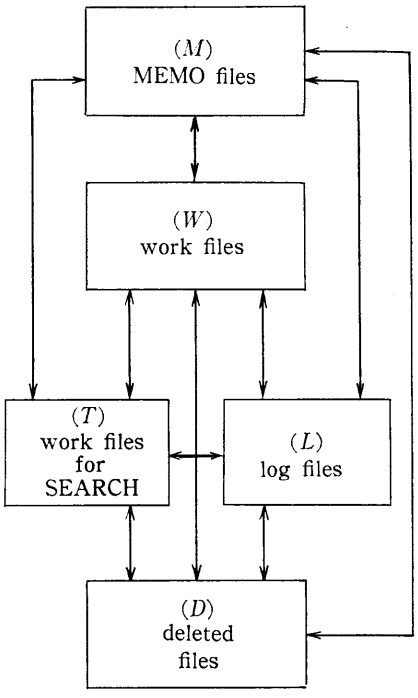


Fig. 2. Structure of MEMO Space

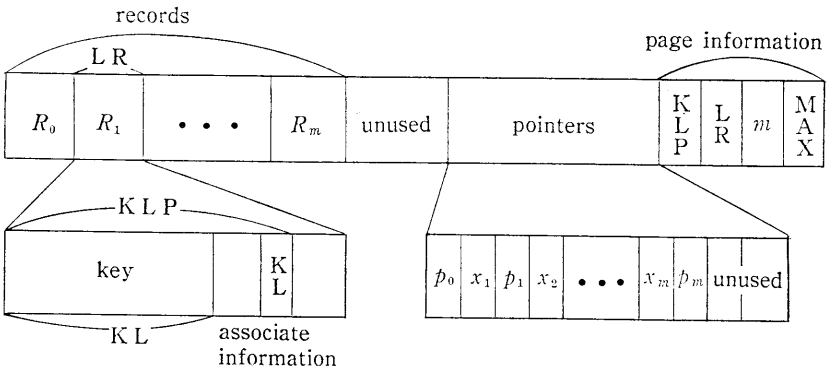


Fig. 3. Organization of *B*-Tree Page
R_i: record, KLP: key length position, LR: length of record, m: number of records in the page, MAX: maximum number of records allowable in the page, KL: length of key, p_i: pointer to page, x_i: pointer to record in the page.

a page of *B*-tree by using pointers to records in the page as shown in Fig. 3, where the size of page is 2048 bytes and KLP (key length position) indicates the point where the length of record is written.

Insertion and deletion of records are carried out in new pages, and when these processes finish the old root page is replaced by a new one. Note that the number of pages influenced in the process of insertion or deletion is at most $O(\log n)$, where n is the total number of records in the *B*-tree. This method protects the *B*-tree against system crashes and careless breakes.

Now the contents of files of SIGMA system are all strings of characters and are stored in bilaterally chained sectors in blocks (See Fig. 4). Here a block is of the same size as of a page in *B*-tree, and is divided into 16 sectors.

The status of the sectors and blocks (pages) is grasped by a bit map. The system issues necessary sectors and pages for a job, checking the bit map, and updates it at the end of the job.

2.4. SIGMA Spaces

A SIGMA space is structually identical with the MEMO space but it has not the subspaces *W*, *T* and *L*. The space is created when the first SIGMA file is made under the user identifier. This space is, as in Fig. 5, for constructing data files for common use and for sharing some data files. In Fig. 5, a dotted rectangle shows a space per one user (identifier), and an arrow shows a permission of access. The privacy in SIGMA system is protected by the permission set up for each SIGMA space and also by pass number set up for each file in SIGMA system. MEMO space can not be shared, for it was intended to be a user's notebook.

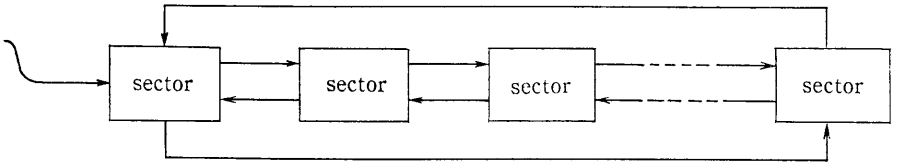


Fig. 4. Physical Structure of Files

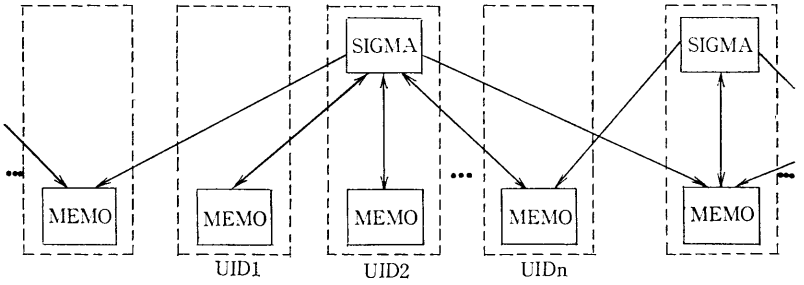


Fig. 5. Sharing Files in SIGMA System

3. Functions of SIGMA System

In this section we discuss the functions of SIGMA system by showing some basic commands.

3.1. Starting and Ending

SIGMA system is invoked by a command procedure SIGMA in a READY state of TSS. At the initial invoking, a MEMO space is created under the user identifier. If MEMO space already exists, that is, at the second or later invoking, a command waiting state SIGMA is immediately entered. Then any command becomes available.

The system is ended by an END command when it is in SIGMA state. If it is not in SIGMA state, the user should get the state by some commands, say one or more ENDs. Now it enters a READY state of TSS. If the user need not keep his MEMO space, he may delete it by a DELETE command of TSS. A MEMO space is of 3 megabytes.

3.2. File Names and Pass Numbers

1) Names of MEMO files are of the form

$$\langle id\rangle_1.\langle id\rangle_2 \cdots \langle id\rangle_n ,$$

where each $\langle id\rangle_j$ is a string of letters and numerals beginning with a letter, n is greater than or equal to one and the total length including the dots is not longer than 33 characters.

2) Files in each of the spaces W, L and T are named, for example, by $W.1$, $W.2$, $\cdots$ in order from the top. Hence they are referred by $W.3$, say. The top files in W, L and T are referred simply by W, L and T, respectively.

3) Names of SIGMA files are nearly the same as those of MEMO files mentioned above, but the user identifiers are needed. The full form is

$$S.'\langle userid\rangle.\langle id\rangle_1.\langle id\rangle_2 \cdots \langle id\rangle_n'.$$

If we use a system constant

$$\text{PREFIX}=\langle userid\rangle.\langle id\rangle_1 \cdots \langle id\rangle_p,$$

we may simply write

$$S.\langle id\rangle_{p+1} \cdots \langle id\rangle_n$$

instead of the full name. The initial value of PREFIX is the user identifier, and the value can be changed by a TERMINAL command.

4) Deleted file names are referred as follows.

(a) Deleted files in a MEMO space are referred just like the files in the spaces in W, L and T, and also by using the old name that it had:

$$D.\,oldname.\langle number\rangle.$$

(b) Deleted files in a SIGMA space are referred just like (a) but S precedes. The PREFIX is also valid.

5) External data sets are referred by

$$X.\,dsname$$

$$X.'dsname'$$

where *dsname* is a usual data set name. Note that these external data sets are referred only through COPY, LOAD and SAVE commands.

6) We may set up pass numbers to MEMO files and SIGMA files. A pass number is a nonnegative integer less than 10000, and is given after the file name as follows:

$$\text{FILE}:=filename$$

$$\text{PASS NUMBER}:=nnnn.$$

Hereafter we use "*filename*" to denote the *filename* including the pass number "*nnnn*" unless otherwise stated.

3.3. Making Files from Keyboards

All files dealt with in SIGMA system, except the work files of the SEARCH command, are strings of characters. To create such a file we may use a KEYIN command, which puts the input string from keyboard on the top of the space W. The end of input is given by ; ; ;CR. The carriage return CR is interpreted as CR as it is, if the symbol just before it is not a continuation (normally "-" and changeable).

3.4. Moving and Copying of Files

1) Moving of files only by changing pointers is carried out by a MOVE command. A general form is

$$\text{MOVE } filename_1\ filename_2,$$

by which the name *filename*$_1$ is deleted from the space and the file of *filename*$_1$ is named *filename*$_2$ instead. This command is naturally invalid between MEMO space and SIGMA space. The system has simple commands PUT, GET and DELETE defined by MOVE as follows:

PUT *filename*=MOVE *W filename*

GET *filename*=MOVE *filename W*

DELETE *filename*=MOVE *filename D*

For example, the command DELETE moves the file named *filename* to the top of the space *D*.

2) Copies of contents of files are made by a COPY command; a general form is

COPY *filename*$_1$ *filename*$_2$.

Since the COPY command makes a copy of the contents, it is valid between any two spaces. The system also has simple commands LOAD and SAVE defined by

LOAD *filename*=COPY *filename W*,

SAVE *filename*=COPY *W filename*.

3.5. Display of File Names and Contents of Files

1) Directories of files are displayed by a DIRECTORY command; a general form is

DIRECTORY *filename*.

If *filename* is omitted, the list of all files in the MEMO space is displayed; if it is one of *W*, *T*, *L* and *D*, the list of all files in the corresponding space is displayed; otherwise the list of all files with *filename* as initial subname (including *filename*) is displayed.

2) Contents of a file is displayed by

LIST *filename*,

and the top file in the space *W* is by

LOOK,

which is equivalent to

LIST *W*.

3.6. Catenation of Files

All files, except some, in SIGMA system are strings of characters. Hence joining is catenating. The catenation is carried out by a CATENATE command; a general form is

CATENATE

$(A_1 \cdot A_2 \cdots A_n \Rightarrow B \ (n<21))$

FILE $A_1 := filename_1$

FILE $A_2 := filename_2$

⋮

FILE $A_m := filename_m$

FILE $A_{m+1} := CR$

FILE $B := filename$,

by means of which a file named *filename* is obtained by successively concatenating files named $filename_1$, $filename_2$, $\cdots$, $filename_m$.

3.7. Multiple Replacement of Strings

Let $X=\{x_1, \cdots, x_n\}$ and $Y=\{y_1, \cdots, y_n\}$ be finite sets of words. Then we can get a file on the top of W space, which is obtained from a file f by replacing all occurrences of word x_i in the f by y_i $(1 \leqq i \leqq n)$. This replacement is carried out by a REPLACE command of the form:

REPLACE

(REPLACE A BY B)

FILE $:= filename$

$A_1 := x_1$

$B_1 := y_1$

$A_2 := x_2$

$B_2 := y_2$

⋮

$A_n := x_n$

$B_n := y_n$

$A_{n+1} := CR$,

where y_i is possibly empty but x_i is not.

The command REPLACE makes use of the pattern matching machines [5]. When REPLACE is invoked, the system constructs a pattern matching machine PMM from the set $X=\{x_1, \cdots x_n\}$ and runs the PMM along the file named *filename* in one-way from left to right. In the meantime the PMM detectes all occurrences of words in X. The PMM devised here will try to detect another longer word (say x_j) even if it has already detected a word. This trial will continue until there are no words longer than those already detected, that is, until the PMM goes again into its initial state by a failure transition or it scans m characters, where m is the maximum length of words

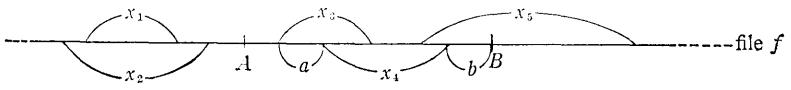


Fig. 6. Multiple Replacement

in X. Then it replaces the longest word x_j it has detected by y_j and repeats the process from the character next to the occurrence of x_j in the file.

We show an example in Fig. 6. Here let $|x_5|=m$, $|x_1|<|x_2|<m$, $|x_3|<m$, $|x_3|<|x_4|$ and $|ax_4b|=m$. Then in the file f, the x_2 is replaced by y_2 somewhere at the point A and x_3 by y_3 at the point B.

3.8. Retrievals

Retrievals of SIGMA system are carried out by a SEARCH command, which makes use of the pattern matching machines [4], [5]. Files are all strings of characters including the carriage return key. Hence some record delimiters are needed to mark off virtual records from a long string. Thus the command requires to set up some record delimiters, keywords to be detected and logical formulae. The SEARCH command is essentially the one-way sequential search system, which we discussed in the first author's previous paper [5]. Here we only list its remarkable features and explain the functions by an example. For further details, the reader should refer to the paper [5].

The SEARCH command has the following features:

1) Detection of records of the form $x_1 \cdots x_2 \cdots x_3 \cdots x_4$, where the triple dot $\cdots$ means "some string". This function is useful for check of some contextual conditions concerning the occurrences of keywords.

2) Independence of record formats. SIGMA system does not assume any definite formats on records, i.e., on files, to be scanned, but assumes the whole file as one very long string of characters. The system successively marks off virtual records intervening between a pair of record delimiters which users can freely set up like the usual keywords.

3) Restriction of universe. The command defines a subset of Σ^* as shown in [5]. The alphabet Σ is the set of characters usable in the computer system. Hence it is not small. We sometimes want to restrict the universe Σ^* into a reasonably small subset. It is not difficult but tedious to restrict the universe. Thus we have introduced a mode (no reset mode) to inhibit loop transitions in the initial state induced by any characters except some characters used in the keywords and delimiters. By this inhibition we can restrict Σ^* into H^*, where H is the set of keywords and delimiters and their initial substrings.

4) Use of two kinds of delimiters. A delimiter is nonempty string of characters. We can use two kinds of delimiters in the SERACH command, record delimiters and item delimiters[3]. The record delimiters work for marking off some virtual records in the file just stated in 2). The item delimiters work for telling when to evaluate the

3) In the previous paper [5] we called the record delimiter an output delimiter and the item delimiter as an input delimiter.

logical formulae. The system evaluates the formulae every time it detects item delimiters. If the value of a logical formula is true i.e., nonnegative at time a record delimiter is detected, then the system will memorize, in the work file in the space T, the data about the virtual record intervened between the present and the last record delimiters.

5) Use of counters. Every keyword variable A with $A:=x$ counts the number of occurrences of the keyword x in a virtual record. The counter was realized simply by $A:=A+1$.

6) Generalization of logical formulae. A logical formula is composed by using *keyword variables, formula variables, ε-variable E, integers, logical operators* (, , ·, ∧), *comparison operators* (=, <=, >=, <, >, < >), *arithmetic operators* (+, *, −, /) and *brackets* [,]. Here the logical operators are interpreted as follows:

$$A, B=1 \quad \textit{if} \quad A>0 \quad \textit{or} \quad B>0$$

$$A\cdot B=1 \quad \textit{if} \quad A>0 \quad \textit{and} \quad B>0$$

$$\wedge A=1 \quad \textit{if} \quad A\leqq 0.$$

The ε-variable E is for detecting the empty string ε. A logical formula may end with a slant (/) like $Z_i:=f/$. In this case the formula variable Z_i works for a temporary memory and hence it is not considered as a question. The system evaluates the set of logical formulae in the order of $Z_1, Z_2, \cdots$. Hence, for example, the value of variable Z_i on the right in a formula

$$Z_i=Z_i+f_i$$

is one at the last evaluation. This suggests that finite automata are easily realizable in our system.

Now we show an example in Fig. 7, where the surrounded parts are inputs by the user and the others are responses from the system. In the second line the user chose a version D, which is a simple version mainly for document retrievals. Record delimiters in the version D function also as item delimiters. The symbol ! in the fourth line stands for a carriage return symbol. The keyword to A_4

$$(\mathrm{AR})\cdots \mathrm{BOOK} \cdots (\mathrm{TI})$$

means that BOOK is not a substring of HANDBOOK, say but is author's name which is between (AR) and (TI), because in this case the file is formed as a string of characters like the second half of the figure.

When the listing of keywords is finished, a PMM is constructed from the set of strings {$!, COMPLEX, NONDETERMINISTIC, (AR), COOK, BOOK, (TI)}. Then the PMM runs on the file named MIYANO with a pass number 1111. The second half of Fig. 7 is a result of the retrieval. The numbers in the parentheses on the right ends mean the lengths of the retrieved records.

3.9. Refiling of Retrieved Records

Data on retrieved records, which includes file names, starting points of the records, lengths and questions satisfied by the records, is memorized in the top file of the work

```
DO: SEARCH
 VERSION (D/E)? D

   RECORD DELIMITERS
 D1:=$I
 D2:=

   KEYWORDS
 A1:=COMPLEX
 A2:=NONDETERMINISTIC
 A3:=(AR)... COOK
 A4:=(AR)... BOOK...(TI)
 A5:=

   LOGICAL FORMULAE
 Z1:=A1.A2.[A3,A4]
 Z2:=A1.A2
 Z3:=A3,A4
 Z4:=A3.^A1
 Z5:=A4
 Z6:=

 FILE:=MIYANO IIII
```

```
  RETRIEVED TEXTS
TOTAL                =     30
QUESTION  1 (Z1 ) =      1
QUESTION  2 (Z2 ) =      5
QUESTION  3 (Z3 ) =     26
QUESTION  4 (Z4 ) =      6
QUESTION  5 (Z5 ) =     18
```

```
 FILE:=
 LIST OF RESULTS (N/Y)?.Y

 QUESTIONS:=1 2
```

```
                                                    QUESTION  1 (Z1 ) =      1
                                                                NO.   1(    97)
(AR) COOK, S.A.
(TI) A HIERARCHY FOR NONDETERMINISTIC TIME COMPLEXITY
(JP) JCSS 7 (1973) 343-353



                                                    QUESTION  2 (Z2 ) =      5
                                                                NO.   1(    97)
(AR) COOK, S.A.
(TI) A HIERARCHY FOR NONDETERMINISTIC TIME COMPLEXITY
(JP) JCSS 7 (1973) 343-353

                                                                NO.   2(   104)
(AR) IBARRA, O.H.
(TI) A NOTE CONCERNING NONDETERMINISTIC TAPE COMPLEXITIES
(JP) JACM 19 (1972) 608-612
...
...
```

Fig. 7. Use of SEARCH Command

space T. Using such data a REFILE command refiles the retrieved records in the top file of the space W. We explain its function by an example in Fig. 8, which is to refile the answers to the question in Fig. 7 in lexicographic order of authors and in order of years for the same authors.

After invoking a REFILE command, the user types question numbers punctuated by blanks, and a new record delimiter, and selects a mode on numbering and a mode on listing. If Y's are selected, records in the new file are numbered and listed. Sorting of records is carried out according to the instruction following the prompt SORT ON:. The instruction is a string of capital letters and minuses (−). The specification of the instruction is given after the prompts SPECIFIED BY and X: =, where X is any

```
DO: REFILE

 QUESTIONS:=1 2
 TOTAL RECORDS =     5
 RECORD DELIMITER:=$
 NUMBERING (N/Y)? Y
 LIST (N/Y)? N
 SORT ON: AY
  SPECIFIED BY
  A:='(AR) ',A8<',','/'ZZZZZZZZ'
  Y:='(JP) ',<'(19',I2/'99'
 CAPS CONVERT (N/Y)? N
DO: _
```

Fig. 8. Use of REFILE Command

letter in the instruction string. In the example above, the specification

$$A:='(\mathrm{AR})\ ',\ A8<',\ '/'ZZZZZZZZ'$$

means the following:

1) Find a string (AR), then
2) extract at most 8 characters before a comma ($A8<',\ '$) and
3) supplement lack if any with necessary Z's ($/'ZZZZZZZZ'$).

In the sequel, a keyword $IBARRAZZ$ is extracted by the specification A and a keyword 72 by Y and hence a keyword $IBARRAZZ72$ by AY from the bottom record in Fig. 7.

A negated letter in the instruction string means to sort in reverse order of the keywords extracted by it. Hence, in the example, if the instruction string is replaced by

$$\mathrm{SORT\ ON}: A-Y,$$

then the records are sorted in lexicographic order of the author and in reverse order of years for the same authors.

In case such instruction is omitted at all, the records are refiled as they are.

Finally lower case letters in the extracted keywords can be converted into the corresponding capital ones by selecting Y at the CAPS CONVERT.

For further details on keyword extractions, readers should refer to Section 4.

3.10. Sorting of Records

Virtual records in the usual files can be sorted by a SORT command just like the above. In this case users should give record delimiters as well as file names. Fig. 9 is an example, which sorts records in the same order as the example in Fig. 8, and should be selfexplaining.

3.11. Use of TSS Command Procedures

Most TSS command procedures under the OS are available from our SIGMA system. A TSS command changes DO state and SIGMA state into a TSS state. Then any TSS

```
DO: SORT
 FILE:=PAPERS
 RECORD DELIMITERS OF THE FILE
 D1:=$!
 D2:=
 NEW RECORD DELIMITER:=$
 NUMBERING (N/Y)? Y
 LIST (N/Y)? N
 SORT ON: AY
  SPECIFIED BY
  A:='(AR) ',A8<',/'ZZZZZZZZ'
  Y:='(JP) ',<'(19',I2/'99'
 CAPS CONVERT (N/Y)? N
DO: _
```

Fig. 9. Use of SORT command

command procedure except a few becomes available. An END command changes the state TSS back to DO state or SIGMA state.

4. Keyword Extractions in Sorting

Keyword extractions in the sorting by the REFILE and SORT commands are carried out according to *extraction_specifications* as we have stated in Section 3.9.

4.1. Syntax of Extraction_Specifications

This *specification* is nearly the same as a format in Fortran and is defined as follows.

An *extraction_specification* is a string of the form

item_delimiter, flist

where the *item_delimiter* is a *pattern* or a *pattern_list*, and the *flist* is an *extraction_item* or a sequence of *extraction_items* punctuated by commas.

A *pattern* is a string of characters beginning and ending with a single quotation mark, or a string of letters *A*, *I* and *D*, which is called a *picture*. The *A*, *I* or *D* stands for a string of alphabetic letters, numerals or characters other than alphanumeric characters, respectively. A *pattern_list* is a parenthesized list of *patterns* which is punctuated by commas. An *item_delimiter* here may possibly be an empty string denoted by a double quotation ".

An *extraction_item* is one of the following

[*r*] *extraction_specifier*,

[*r*] (*flist*),

where *r* is a positive integer called a *repetition specifier*. Omission of *r* is equivalent to $r=1$.

An *extraction_specifier* is a *read_specifier* or a *skip_specifier*. A *read_specifier* is of the form

$$\left\{\begin{matrix} A \\ I \end{matrix}\right\}[r]\left[<\left\{\begin{matrix} pattern \\ pattern\_list \end{matrix}\right\}\right][/'string'],$$

where the length of the *string* is not greater than r, A stands for character strings and I stands for numeral strings. According to the *read_specifier* of the form above, the system reads a character string (A) or numeral string (I) of length not greater than r which is before ($<$) the *pattern* or *pattern_list*, and supplements lack or missing with a necessary suffix of the *string*. If the slant / and the *string* for supplement are omitted, then a string of blanks or 9's for A or I, respectively, is assumed.

A *skip_specifier* is one of the following

$$X[r]\left[<\left\{\begin{matrix} pattern \\ pattern\_list \end{matrix}\right\}\right],$$

$$<\left\{\begin{matrix} pattern \\ pattern\_list \end{matrix}\right\}.$$

According to the first *skip_specifier*, the system skips at most r characters until ($<$) it finds *pattern* or *pattern_list*. The meaning of the second one should be clear.

We show two illustrative examples of *extraction_specification* in Fig. 10.

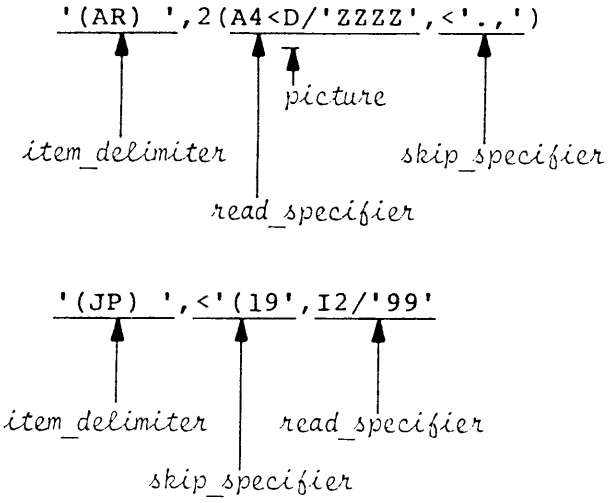


Fig. 10. Extraction-specifications

4.2. Keyword Extractions

The system constructs a PMM (pattern matching machine) from the set of all *patterns* used in the *extraction_specifications* and also in record delimiters in case of SORT command, and then extracts keywords by running the PMM on the records. The SORT command marks off virtual records in a file while the PMM is running. The extracted keywords are successively inserted into our B-tree. Thus the sorting, includ-

ing keyword extractions, is completed while the PMM runs on records or a file to the end. The total time of sorting is still $O(n \log n)$, where n is the length of the file and the coefficient should be very small.

Now according to the *extraction_specification* the system extracts keywords from a record R as follows.

1) Try to find an *item_dlimiter* in R. If the i-th *item_delimeter* is found out, then go to 2). In case an empty *item_delimiter* exists, go directly to 2).

2) Extract a keyword according to the *extraction_specifier* beginning with the i-th *item_delimiter*. If another *item_delimiter* is found out in the meantime, then finish the present extracting by the *extraction_specifier* and supplement the lack if any and then repeat 2) for the newly found *item_delimiter*. If the keyword extractions are completed for all *extraction_specifications* or an end of record is reached, then go to 3).

3) Compose a complete keyword from the (sub)keywords extracted at the stage 2) according to the instructions following the prompt SORT ON: and the selection at the CAPS CONVERT. At this stage a (sub)keyword corresponding to a negated letter should be reversed, and also it is converted into a string of capital letters if Y is selected at the CAPS CONVERT. Then go to 4).

4) Insert the complete keyword into the B-tree to sort.

The process above is repeated until the records are exhausted.

5. Log Files and Their Use

Communications between a user and SIGMA system are automatically recorded in a file of the space L. We call such a file a log file. The log file is a string of characters like other usual files of SIGMA system, and it consists of records of prompts by the system and responses by the users. We can give a proper name to the log file and treat it in exactly the same way as the usual files.

In SIGMA system we can give commands and their operands from any files as well as from keyboards. Thus a log file is used as a kind of command procedure. The essential difference between the usual command procedures and ours is in the point that ours are automatically and unconsciously constructed while we are working on SIGMA system.

Changes of input streams from keyboards to files and vice versa are possible by

$$.\,\textit{filename}$$

at any time. The *filename* is of any file in SIGMA system and the expression above also includes the following special ones

$$.K$$

$$.K.1$$

$$.E,$$

where $.K$ changes the input stream to the keyboard and at the same time starts a new logging, $.K.1$ changes the input stream to the keyboard while just one line is typed, and $.E$ finishes the current input stream and returns to the last input stream.

In the second case no record is stored, and in the last case the current log file is closed and put on the top of L if the current stream is from the keyboard.

The SIGMA system has two main states, SIGMA state and DO state, which are notified by the corresponding prompts SIGMA> and DO:. When SIGMA system is invoked, it is in SIGMA state. Then it opens a log file by a command of SIGMA other than END and some special commands, and it begins to record the communications, and enters DO state. The usual commands are now available. The END command ends the SIGMA system in SIGMA state, but it has the same function as $.E$ in DO state. Note that $.E$ is available at any time except the system is in SIGMA state, which is different from END. The system in DO state goes into SIGMA state when a log file on the top is closed by $.E$ or END. (See Fig. 11 below).

Now we show an illustrative example of input streams in Fig. 12, where $.L$ is a top file in log space L, which is the communication record in the second real line. When the system returns to SIGMA state, three log files are constructed in the log space L in the order of 1, 3 and 2 from the top.

Finally we show a simple use of log file in Fig. 13, which should be self-explaining.

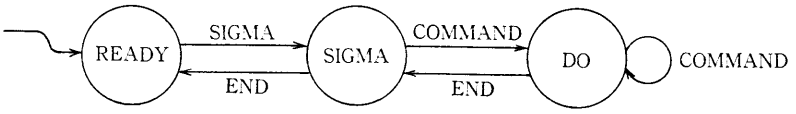
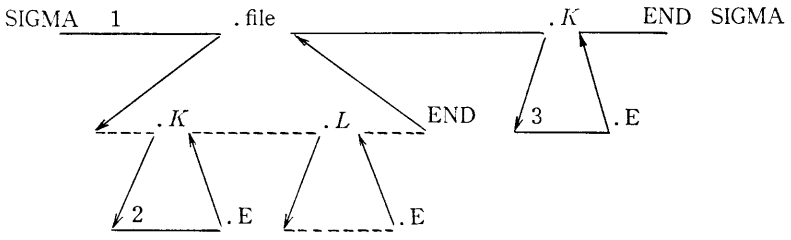


Fig. 11. State Transition of SIGMA System



Real lines are from keyboard and dotted lines are from files.

Fig. 12. Input Streams of SIGMA System

6. Concluding Remarks

We have described an information system SIGMA and discussed its functions. The system assumes no other data structure than that of the strings of characters, and has one-way sequential searches as its access method. The users of the system can share files with other users and can easily copy data from SIGMA files to external data sets

```
READY
SIGMA
SIGMA> LOOK
This is a text.

DO: END
SIGMA> LIST L
DO: LOOK
DO: END

DO: .L
LIST ? (Y/N) Y
DO: LOOK
This is a text.

DO: END
DO: END
SIGMA> END
READY
```

Fig. 13. Use of Log File

and vice versa. Hence they can use any other resources and facilities the computer system has. Thus we expect SIGMA system to be a friendly interface to other data base systems as well as ordinary programs of users.

We have a plan to extend the present version of the system so as to function as a usual database system. In fact, a proper editor has been developed [6] and a data description language and a language for application programs are being studied and developed.

The use of log files discussed here should be a new idea. The users of SIGMA system will easily be possible to have their own system in SIGMA by using this function. We can say that such use of log files is a simple but practical realization of learning systems and is another automatic programming. In this connection we have observed that a practical data entry system with learning ability is realizable. One of the authors (T.S.) has obtained an important class of languages which are easily inferrable from positive data, i.e., input data [7].

Acknowledgements

The authors should like to express their sincere thanks to Professor S. Kano, Director of Computer Center, for his constant support and encouragement. They are grateful to Mr. S. Miyano at Department of Mathematics for his co-operation in implementing the experimental version of SIGMA system and for his valuable discussions. Last but not least they are also grateful to Lecturer F. Matsuo, Head of Research and Development Division, and to his staff, especially to Mr. S. Futamura and Mr. T. Suenaga for their advice and help during the course of implementing at the Center.

References

[1] INOSE, H. (ed) *Scientific Information Systems in Japan*, North-Holland, (1981).

[2] ARIKAWA, S. and KITAGAWA, T. *Multistage Information Retrieval System Based upon Researcher Files*, Proc. 2nd USA-JAPAN Computer Conference, (1975), 149-153.

[3] ARIKAWA, S., KANO, S., KITAGAWA, T. and TAKEYA, S. *Organization and Use of Private Researcher Files in Scientific Research Works*, Scientific Information Systems in Japan (H. Inose, ed.), North-Holland, 43-50, (1981).

[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. *One-Way Sequential Search Systems and Their Powers*, Bull. Math. Stat., **19**, (1981), 69-85.

[6] SHINOHARA, T. *On an Editor TEDIT*, Project Report "Knowledge Representations and Their Applications to Information Retrievals", (S. Arikawa, ed.), (1981), 247-258, (Japanese).

[7] SHINOHARA, T. *On Inferring Pattern Languages*, MC Memorandum, Kyushu Univ., (1981), (Japanese).

[8] ARIKAWA, S. *Pattern Matching Machines to Detect Longest Keys*, Res. Rept., Res. Inst. Fund. Inform. Sci., Kyushu Univ., (1981, to appear).