

研究者向き情報システムSIGMAについて

有川, 節夫
九州大学理学部基礎情報学研究施設

篠原, 武
九州大学大学院総合理工学研究科情報システム学専攻

白石, 修二
九州大学大学院総合理工学研究科情報システム学専攻

玉越, 靖司
九州大学大学院総合理工学研究科情報システム学専攻

<https://doi.org/10.15017/1474967>

出版情報：九州大学大型計算機センター広報. 14 (4), pp.550-573, 1981-12-10. 九州大学大型計算機センター
バージョン：
権利関係：



研究者向き情報システム SIGMA について

有川 節夫^{*}, 篠原 武^{**}, 白石 修二^{**}, 玉越 靖司^{**}

1. はじめに

研究者用の学術情報システムの重要性が指摘され [1], 大学の大型計算機センターにおいて, 文献検索サービスも行われるようになり, 研究者の情報システムに対する関心と必要性の認識も深まってきた. 九州大学大型計算機センターにおいては, 1978 年以来 FAIRS による INSPEC 文献データの検索サービスが行われており, 計算機科学, 物理学関係の研究者にとって重宝されている. また Adbis によるデータベース統合支援も可能になってきた [2].

基礎情報学研究施設では 1973 年以来, 文部省科学研究費による特定研究「広域大量情報の高次処理」及び「学術情報システムの形成過程」, 更に総合研究等を通じて, 研究者用の情報システムのあり方について, 提案や実験システムの開発を行ってきた [3], [4], [5]. また二番目の特定研究において, 学術情報システムを PRF (Private Researcher Files), UDL (User Data Library), CDL (Center Data Library) といった三段階で構成する方式が提案され, その後 PRF, UDL レベルのシステムの研究や開発がなされた [4], [6], [7].

一方, 計算機システムについても, 機能の拡充と端末機器の普及によって, TSS による利用法が定着してきた. これに伴って, 各ユーザ当りのデータセットの使用量も増大し, プログラムの貯蔵だけでなく, 学術データの蓄積のためにも使われるようになってきた.

こうした情報システムに関するユーザの認識の変化, 計算機システムの利用法の変化を考慮して, 更に筆者等の経験を生かして, 新しく SIGMA という名前の研究者向きの情報システムを開発した. このシステムには, 次の段階で, いわゆるデータベース・システムとしての機能を追加する予定であるが, 今回は, 一方向逐字処理に基本を置いた小回りの利くエディタとしても活用できる部分について, その利用法を中心に解説することにする. このシステムは, 先に述べた学術情報システムの三段階構成でいえば, PRF から UDL を指向したもので, CDL とも交流の可能なものである.

2. システムの概要

SIGMA システムは, 文献の蓄積・検索, 論文の作成, 書類の整理, プログラムの作成, その他データの収集・加工等, 研究者の日常的な活動を支援する目的で開発した研究者向きの情報システムである. 本節では, 先ず, その概要をシステムの特徴とファイル構成の面から説明することにする.

2.1 システムの特徴

研究者の日常的な研究活動を支え, 研究者の作業場として活用できる情報システムには以下のよう

* 九州大学理学部基礎情報学研究施設

** 九州大学大学院総合理工学研究科情報システム学専攻

なことが要求される[4]，[8]，[13].

- 1) 多種多様なデータや処理を扱えるような柔軟な構造と効率的な機能を有すること.
- 2) 自由に自分用のデータを蓄積し，それを検索編集できること.
- 3) 研究者同志が協同し，そのグループ用のデータベースを構築できること.
- 4) 網羅性のある汎用のデータベースをアクセスし，その中の必要なデータを自分のファイルに取り込み活用できること.
- 5) 入力したデータは，通常のデータだけでなくコマンド類に至るまで，すべて損われることなく，十分に活用できること.

このような観点に立って，SIGMAシステムでは，データとしては，末端の利用者にとって，最も単純で基本的である文字列を対象にして，アクセスや処理としては，パターン・マッチング・マシン[9]の技法を駆使した一方向逐字処理[10]を基本においた．これによって，小回りの利いた検索や検索結果の編集，再ファイル化が簡単に行えるようになった．ファイルの転送を自分の領域内(2.2参照)，グループにおける領域間，一般のデータセットと領域間で自由に行えるようにした．これによって，要求[3]，[4]を満たす道が開けた．また計算機システムの障害や利用者の不用意な回線の切断によって，ファイルが消滅することがないように十分な配慮をした．更に利用者のキーボードからの入力は，データだけでなくコマンドに至るまで，特別な例外を除いて全て自動的に記録され，これも通常のファイルと同様に編集検索の対象にされ，しかも一種のコマンド・プロシジャとして活用できるようにした．こうした点にSIGMAシステムの特徴がある．

2.2 領域とファイルの構成

SIGMAシステムのデータ・ベースは各ユーザのダイレクト・アクセスできるデータ・セット上に構成される．このデータ・セットにファイルやファイル名を管理するためのB木が作られるわけである．ここでは，このようなデータ・セットを領域と呼ぶことにする．各ユーザ(1ユーザID)から見たSIGMAシステムの領域は，図1に示す通り，MEMO領域とSIGMA領域とからなる．更にこうし

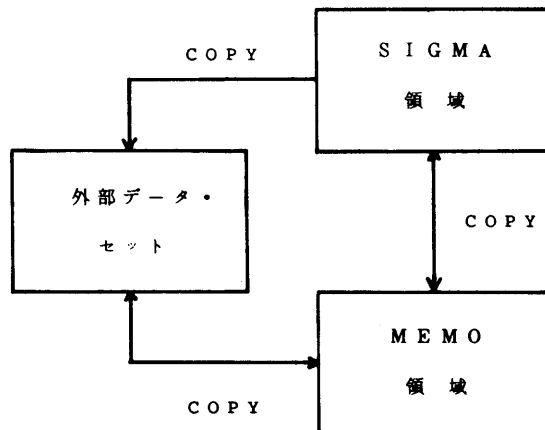


図1. SIGMAシステムの領域

た領域の外部にある一般のデータ・セット（以後、外部データ・セットという）も参照できるようになっている。

MEMO 領域は SIGMA システムのユーザにとっては必須のデータ・セットである。この領域は図 2 に示す通り、作業用のファイルを置くための領域（W）と、自分用の整理された名前付きのファイルを置くための MEMO ファイルの領域、SEARCH コマンドの作業用ファイルの領域（T）、

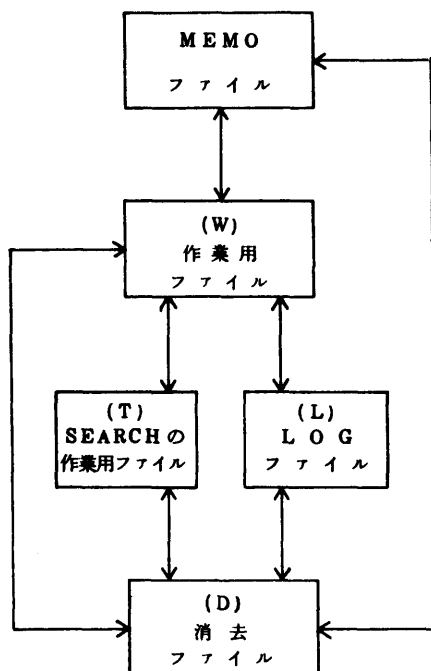


図 2. MEMO 領域におけるファイルの構成

SIGMA システムにおける交信の記録をとるための LOG ファイルの領域（L），更に消去されたファイルを置くための領域（D）とからなる。

領域W，T，L，Dはいずれも底のあるスタックの構造をもち、最近のファイルがそれぞれのトップにくるようになっている。

領域W，T，Lにあるファイルは個数が一定値を越えるとスタックの底にある最古のファイルが消去ファイルの領域のトップに移される。MEMO 領域に空きがなくなった場合には、領域Dの中のファイルが古いものから順に実際に消去されて、必要な領域の確保を行う。こうした操作はコマンドの実行中にすべて自動的に行われる。

SIGMA 領域は、構造的には MEMO 領域から領域W，T，Lを除いたものに等しい。この領域は図 3 にあるように、ユーザ・グループが協同して1つのデータ・ベースを構築したり、できあがったデータ・ベースを共有するための領域である。図において破線で囲まれた部分が1ユーザ ID 当りの領域である。矢印は SIGMA 領域のアクセス権（READ / WRITE）を示している。なお、SIGMA システムにおけるファイルの機密保護は、領域ごとに設定する認可制とファイルごとに設定するパスワードによって行う。MEMO 領域はそれを所有するユーザにとって、文字通りメモ帖として活用

できることを意図したものであるから、この領域内のファイルの共有はできない。

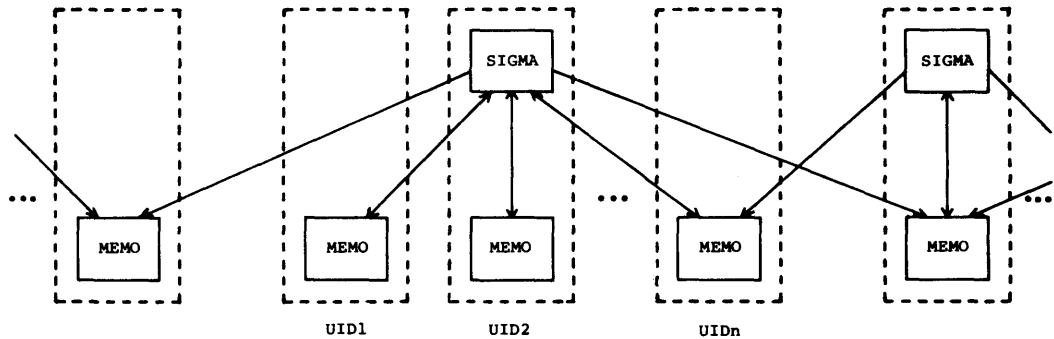


図3. SIGMAファイルの共有

3. システムの利用法

以下に SIGMA システムの利用法について一通り説明する。逐字サーチやソーティング、LOG ファイルの利用、TSS コマンドの呼び出しについて第4節以下でそれぞれ節を改めて詳しく説明することにする。

3.1 起動と終了 (SIGMA, END)

SIGMA システムの起動と終了は次のようにして行う。

起動：1) LOGON 時に SIZE パラメータを 1024 以上に指定する。FAIRS を SIGMA から呼び出すときは、そのための処置しておく。

```
LOGON TSS userid/password S(2048) PROC(FAIRS)
```

2) SIGMA システムを呼び出す。

```
READY
```

```
SIGMA
```

3) 初めて SIGMA システムを呼び出した時には、MEMO 領域が存在しないので、システムから MEMO. DATA という名前のデータ・セットを作成してよいか否を問い合せてくる。YES (Y) の指示をすれば、MEMO 領域を作り始める。約 1 分間の時間を要する。既に MEMO 領域が存在していれば、直ちにコマンド待ちの状態になる。

```
NO MEMO SPACE EXISTS.
```

```
CREATE (Y OR N) ? _
```

終了：4) SIGMA システムには SIGMA 状態と DO 状態とがある。DO 状態の場合には END コマンド等によって、SIGMA 状態に戻した後で END コマンドを投入すると終了する。即ち TSS の READY 状態に戻る。

```
DO: END
```

```
SIGMA > END
```

```
READY
```

- 5) MEMO 領域を保存する必要がない場合には、TSS の DELETE コマンドでデータ・セット MEMO . DATA を消去すればよい。このようにして、MEMO . DATA を消去すると復旧することができないので注意を要する。なお、MEMO . DATA は 3 メガバイトのデータ・セットである。

```
READY
DELETE MEMO . DATA
```

3.2 ファイル名とパス・ナンバー

1) MEMO ファイルの名前は $\langle id \rangle_1 . \langle id \rangle_2 . \dots . \langle id \rangle_n$ の形の表現である。ここに、各 $\langle id \rangle_j$ は英字で始まる長さ 2 以上の英数字列で、 $n \geq 1$ であり、かつ全体の長さはピリオドを含めて 32 文字以下である。

2) MEMO 領域における領域 W, L, T におけるファイルは新しい順に、例えば、W.1, W.2 のように名前付けられている。したがって参照は、W.3 のように、数の形で行う。これらの領域のトップにあるファイルの参照は W, L, D のように、.1 を省略して行うことができる。

3) SIGMA ファイルの名前は、MEMO ファイルの名前と同様であるが、誰れ(どのユーザ ID) の SIGMA 領域にあるものであるかを指定する必要がある。表現の完全な形は

$$S. \nabla \langle userid \rangle . \langle id \rangle_1 . \langle id \rangle_2 . \dots . \langle id \rangle_n \nabla$$

である。これで、SIGMA ファイルであること (S)、どのユーザ ID のものであるか ($\langle userid \rangle$)、また名前は何であるか ($\langle id \rangle_1 . \langle id \rangle_2 . \dots . \langle id \rangle_n$) が識別されることになる。またシステム定数 PREFIX を用いれば、 $\langle userid \rangle . \langle id \rangle_1 . \dots . \langle id \rangle_p$ と引用符 ∇ を省略した

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

でもって、上の完全形を表わすことができる。

PREFIX は SIGMA システムの起動時にはその利用者のユーザ ID となっている。変更は TERMINAL コマンドによって行う。

4) 消去ファイルの名前。

(a) MEMO 領域にある消去ファイルは、2) の W, L, T と同様に参照できる。またそのファイルがもっていた古い名前でも参照できる。この場合のファイル名の指定は

$$D. \text{filename. 数}$$

である。

(b) SIGMA 領域の消去ファイルの参照は先頭に S. を付けることを除けば (a) と同様である。PREFIX については、3) と同様で、

$$S. D. \nabla \langle userid \rangle \nabla$$

$$S. D. \text{filename}$$

のように用いる。

5) 外部データ・セットの名前は

$$X. \text{dsname}$$

$$X. \nabla \text{dsname} \nabla$$

である。dsname は通常用いるデータ・セット名である。なお、外部データへのアクセスは COPY, LOAD, SAVE によってのみ可能である。

6) パス・ナンバーは MEMO ファイルと SIGMA ファイルに対して設定できる。パス・ナンバーは、0 ~ 9999 の整数で、MEMO ファイルや SIGMA ファイルを作るとき、或いは参照するときに指定する。指定は

```
FILE := filename
PASS NUMBER := n n n n
```

のように filename の指定に引き続いて行う。省略されれば、即ち CR は 0 と解釈される。パス・ナンバー 0 をもつファイルの参照にはパス・ナンバーの指定は不要である。

3.3 ファイルの作成 (KEYIN)

SIGMA システムで扱うファイルは SEARCH コマンドの作業用ファイルを除いて全てテキスト (文字列) である。このような文字列としてのファイルをキーボードから作成するためには KEYIN コマンドを使う。これは、キーボードから入力した (長い) 文字列を作業領域 (W) のトップにファイルとして置くためのコマンドである。入力の終了は ; ; ; CR で指示する。テキスト入力中の CR は、そのまま復改記号として解釈される。ただし、CR の直前の入力文字が継続文字 (標準値一で、変更可能) である場合には、復改記号は無視される。

```
DO: KEYIN
>This is a text in English.
>;;
DO: LOOK
This is a text in English.

DO: _
```

3.4 ファイルの転送と複製 (MOVE, COPY, NICKNAME)

1) MOVE (PUT, GET, DELETE) はポインタの付け替えによってファイルの転送を行うもので、ファイルの複製は行われぬ。MEMO 領域と SIGMA 領域間での転送はこのコマンドではできない。一般形は

```
MOVE filename1 filename2
```

である。(もちろん、パス・ナンバーが必要な場合には指示しなければならない。以下の説明において特に断らない限りパス・ナンバーも含めて filename と呼ぶことにする。) これによって filename 1 のファイルが filename 2 に移される。PUT, GET, DELETE コマンドはそれぞれ次のように MOVE コマンドによって定義されるものである。

```
PUT filename      = MOVE W filename
GET filename      = MOVE filename W
DELETE filename = MOVE filename D
```

つまり、PUT は作業領域のトップにあるファイルを filename に移し、GET は filename にあるファイルを作業領域のトップにもってくる。また DELETE は filename にあるファイルを消去ファイルの領域のトップに移す。

2) COPY (LOAD, SAVE)はファイルの複製を作るためのコマンドで、一般形は

```
COPY filename 1 filename 2
```

である。これによって、filename 1 のファイルの複製が filename 2 に作られる。COPYコマンドでは、内容の複製が作られるので、MOVEコマンドと違って、異なる領域間、外部データ・セットと領域間でも使用できる。

LOAD, SAVEコマンドはそれぞれ次のように定義されるものである：

```
LOAD filename = COPY filename W
```

```
SAVE filename = COPY W filename
```

なお、外部データセットに対して有効なコマンドは、ここで挙げた COPY, LOAD, SAVE だけである。

3) NICKNAME は

```
NICKNAME filename 1 filename 2
```

の形で使われるコマンドで、名前 filename 1 をもつファイルに別名 filename 2 を追加する。ただし、filename 1 は領域 W, T, L, D 上のファイル名であってはならない。

```
DO: LOOK
This is a text in English.

DO: PUT TEXT
PASS NUMBER:=1111
DO: LIST TEXT 1111
This is a text in English.

DO: LOAD TEXT 1111
DO: LOOK
This is a text in English.

DO: _
```

3.5 ファイル名及びファイル内容の表示 (DIRECTRY, LOOK, LIST)

1) DIRECTRY はファイルの名前、作成期日、長さを表示するためのコマンドで、一般形は

```
DIRECTRY filename
```

である。filename を省略すると MEMO ファイル全体のファイル名が表示される。filename が W, T, L, D の場合には、それぞれの領域内にあるファイル全体の名前が表示される。これ以外の場合には、filename と filename. で始まるファイルの名前が全て表示される。したがって、例えば、filename が S の場合には、その時の SIGMA ファイルの全体の名前が表示されることになる。

2) LIST はファイルの内容を表示するためのコマンドで、一般形は

```
LIST filename
```

である。

3) LOOK は作業領域のトップにあるファイルの内容を表示するためのもので、LIST W に等しい。

3.6 ファイルの結合 (CATENATE)

SIGMA システムにおけるファイルは文字列であるから、ファイルの結合は文字列の連結 (cate-

nation)である。連結はCATENATEコマンドで行う。一般形は

```
CATENATE
    ( A1 · A2 ··· AN ⇒ B ( N < 21 ) )
FILE A1 := filename 1
FILE A2 := filename 2
    :
FILE Am := filename m
FILE Am+1 := CR
FILE B := filename
```

である。これによって、ファイル filename 1, filename 2, ..., filename m をこの順序で連結した filename という名前のファイルが作られる。

```
DO: LIST DATA
ABC
DO: CATENATE
    ( A1 A2 ... AN ==> B ( N < 21 ) )
FILE A1:=DATA
FILE A2:=DATA
FILE A3:=DATA
FILE A4:=

FILE B:=W
DO: LOOK
ABCABCABC
DO: -
```

3.7 文字列の同時置き換え (REPLACE)

ファイルの中の指定された文字列をことごとく指定された他の文字列で置き換えることができる。これはエディタのサブコマンド CHANGE を一般化したもので、一般形は

```
REPLACE
    ( REPLACE A BY B )
FILE := filename
A1  := x1
B1  := y1
A2  := x2
B2  := y2
    :
An  := xn
Bn  := yn
An+1 := CR
```

である。これによって、filename という名前のファイルの中に含まれている文字列 x_i をことごとく文字列 y_i ($1 \leq i \leq n$) で置き換えたファイルが作業領域のトップに作られる。 $n \leq 99$ である。なお使用に当たって以下のことに注意して欲しい。

- 1) y_i は空列 (すなわち, $B_i := CR$) でもよいが, x_i には空列は許されない.
- 2) ある文字列 x_i が検出され, これが y_i で置き換えられた時点で, 文字列 x_1, x_2, \dots, x_n を探す作業は初期化され, この x_i の次の文字から新しい置き換え作業がくり返される.
- 3) 文字列 x_i, y_i の登録において, $x_i = x_j$ ($i < j$) となった場合には, 始めの方の

$A_i := x_i$

$B_i := y_i$

は無視される. したがってこれを置き換えの指定を訂正に使うこともできる. 即ち, 第 i 番目の指定を訂正するには,

$A_j := x_i$

$B_j := y_j$

のように A_j に訂正したい文字列 x_i をまた B_j に正しく置き換えるべき文字列 y_j を割り当てればよい. これによって, 第 i 番目の指定が無効になるために, $i + 1$ 番目から j 番目までの指定が1つつつ詰め合わされ, 次のプロンプトは再び $A_j :=$ となるが, このコマンドでは A と B の対だけが意味をもつので気にしなくてよい.

- 4) REPLACE コマンドは左から見てなるべく長い文字列を置き換えるように働く, したがって図4におけるファイルに対しては, x_2 の部分が y_2 に置き換えられる. また図の右の部分については, x_4 は無視され, x_3, x_5 に対する置き換えが行われる. これは, 置き換えを左から右への一方向の逐字処理により行い, しかもバック・トラックをしないためである [14].

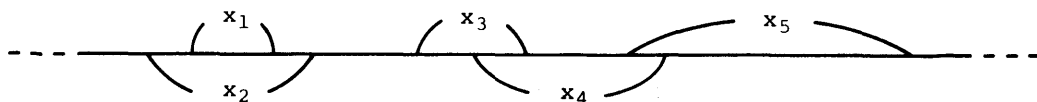


図4. 文字列の同時置き換え

3.8 検索 (SEARCH)

検索は一方向逐字サーチに基づくコマンド SEARCHによって行う. SIGMA システムのファイルは基本的にはすべて1本の文字列である. もちろん復改キー CR も1個の文字として扱われる. こうしたファイルの中に適当な区切り語を設定し, 2つの区切り語にはさまれた部分文字列を仮想的なレコードと考えて検索を行う. したがって, 質問文の作成には, このようなレコード区切り語の指定と検出すべきキーワードの指定, 更に論理式の指定が必要である. SEARCH コマンドの詳細については, 節を改めて第4節で説明することにして, ここでは例に従って簡単な解説をすることにする. 例において, 四角で囲んだ部分がユーザの指定する項目である.

SEARCH コマンドの投入に続いて, VERSION D (4節を参照) を選択する. 引き続いて, レコード区切り語をプロンプト $D_i :=$ に従って入力する. $D_1 := \$!$ における記号! は復改記号である. これによって, $\$ CR$ の間にある文字列が仮想的なレコードと解釈されることになる. レコード区切り語の登録が終われば ($D_2 := CR$), キーワードの登録が始まる. これはプロンプト $A_i :=$ に従って入力する. キーワードには通常文字列 (A_1 における COMPLEX など) の他に, A_3, A_4 にあるよう, トリプルドット...を含む文字列も使用できる. $x \dots y$ は「 x があってその後何

```

DO: SEARCH
VERSION (D/E)? D

RECORD DELIMITERS
D1:=$!
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 1111
    
```

```

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
...
    
```

かあって(空列でもよい)yがあるような文字列」という意味をもつ。A4における

(AR) ... BOOK ... (TI)

は検索しようとしているファイルが上の例の文献リストに見られるような形をしているので、例えば、HANDBOOKなどのBOOKではなくて、著者名としての、つまり(AR)と(TI)の間にあるBOOKであることを記述したことになる。

キーワードの登録に続いて、論理式の登録を行う。論理式はプロンプト Zi:= に従って、キーワード変数(Ai)と論理記号 and(・), or(,), not(へ), 括弧([,])を用いて作る。例にあるように複数の論理式を登録することができる。

Z1 := A1. A2. [A3, A4]

は、「A1 に割り当てられたキーワード(すなわち COMPLEX)を含みかつ A2のキーワードを含み、かつ A3 のキーワード又は A4 のキーワードを含む」レコードを検索せよという意味をもつ。

論理式の登録が終わったら、プロンプト FILE:= に従って検索対象となるファイル名を指示する。これで、システムは検索を開始し、検索結果を例にあるように表示する。全く同じ質問を別のファイルに適用したければ、次々にプロンプト FILE:= に従ってファイル名を指示すればよい。

検索が終了する(FILE:= CR で指示する)と、検索結果を表示するか否かをきいてくるので、必要ならば Y を選り希望する質問番号(論理式の番号)を空白で区切って指示する。検索結果は、例の下の方にあるような形で表示される。右端の()中の数値はレコードの長さ(文字数)である。

3.9 再ファイル化 (REFILE)

SEARCH コマンドによる検索結果は、MEMO 領域の中の SEARCH コマンドの作業領域(T)にファイル名、レコードの開始点、長さ、質問番号等をコード化した形で記録される。

REFILE はこうした記録を基にして、検索結果を再びファイル化するためのコマンドである。これについても、例に従って説明しよう。この例は 3.8 の例による結果を著者のアルファベット順に、しかも同一著者については年代順にソートして、新しいファイルを作るためのものである。

まず REFILE コマンドの投入し、続いて必要な質問番号を指示し、新しく作られるファイルにおけるレコード区切り語を指示する。レコード区切り語は省略されれば、復改記号 CR である。レコード区切り語に続いてレコードに番号を付ける場合には NUMBERING (N/Y) ? において Y を選択する。新しいファイルのリストが必要ならば、LIST のところで Y を選択する。(例においては

```

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<','/'ZZZZZZZ'
Y:='(JP) ',<'(19',I2/'99'
CAPS CONVERT (N/Y)? N

DO: LOOK
$00001
(AR) COOK, S.A.
(TI) A HIERARCHY FOR NONDETERMINISTIC TIME COMPLEXITY
(JP) JCSS 7 (1973) 343-353
$00002
(AR) IBARRA, O.H.
(TI) A NOTE CONCERNING NONDETERMINISTIC TAPE COMPLEXITIES
(JP) JACM 19 (1972) 608-612
$00003
(AR) SAVITCH, W.J.
(TI) RELATIONSHIPS BETWEEN NONDETERMINISTIC AND DETERMINISTIC TAPE
COMPLEXITIES
(JP) JCSS 4 (1970) 177-192
$00004
(AR) SAVITCH, W.J.
(TI) PARALLEL AND NONDETERMINISTIC TIME COMPLEXITY CLASSES
(JP) LNCS 62 (1978) 411-424
$00005
(AR) SEIFERAS, J.E., MEYER, A.R.
(TI) SEPARATING NONDETERMINISTIC TIME COMPLEXITY CLASSES
(JP) JACM 25 (1978) 146-167

DO: _
    
```

Nであるからリストはとられない。))

レコードのソートを行う場合には、プロンプト SORT ON: に従って英字とそれに- (マイナス) を付けたものの列として指示し、その英字の意味を SPECIFIED BY 以下のプロンプトに従って指示する。SORT ON: に続く英字列を省略すると、検索された順序にレコードを並べたファイルが作られる。

```
A := ▽ (AR) ▽, A8 < ▽, ▽ / ▽ ZZZZZZZZ ▽
```

は「文字列 (AR) ▽ を見つけて、その後の、までの高々8文字 (A8 < ▽, ▽) を読みとり、不足分はその個数だけのZで補え」ということを意味する。スラント以下の文字列は欠落又は不足分を補うためのもので、この指定が省略された場合には、文字型 (A8 など) に対しては空白の列が、また整数型 (Yにおける I2 など) に対しては0が仮定される。例において、LOOK 以下はソートされたファイルの内容であるが、例えば、3番目の文献に対しては、Aによって SAVITCHZ が、またYによって70が切り出され、結局ソーティングの対象となるキーとして長さ10の文字列 SAVITCHZ70 が切り出される。

SORT ON: における英字で-を付けられたものに対しては、その逆順にソートされることになる。したがって、例えば、例において

```
SORT ON: A-Y
```

と指示すると、著者のアルファベット順で、同一著者については年代の新しい順にソートされる。

切り出された文字列に大文字小文字が混在している場合には、CAPS CONVERT においてYを指示し大文字に変換することができる。

このコマンドによって作られるファイルは作業領域のトップに作られる。

3.10 ソーティング (SORT)

通常のファイルにおける仮想的なレコードを対象にして、ソーティングを行うことができる。これは SORT コマンドを用いて次の例のように指示する。ソーティングを行うファイル名を指定して、そこでのレコード区切り語をプロンプト Di := に従って登録する。その後は、REFILE コマンドにおけるソーティングの指示と全く同様である。例は、3.8の例において使用した検索対象のファイルと同じ形をしたファイルに対して、3.9の例と同じソーティングを行うためのものである。これによって、ファイル PAPERS 中のレコード区切り語 \$! で囲れたレコードが次々に取り出されて、ソーティングのためのキーが切り出され、それによってソートされたファイルが作業領域のトップに作られる。

```
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<','/'ZZZZZZZ'
Y:='(JP) ',<'(19',I2/'99'
CAPS CONVERT (N/Y)? N
```

3. 11 システム 定数の表示と変更 (TERMINAL)

SIGMA システムのもつ定数の表示や変更は TERMINAL コマンドによって行う、このコマンドを投入すると、プロンプトが TERM: になるので、それに続いて変更したいシステム定数に対応するサブコマンドと変更値を入力する。

- 1) NEWLINE によって代用復改記号を変更する。初期値と省略値は ! である。
- 2) CONTINUE によって継続記号を変更する。初期値は - で、省略すると継続処理は行われなくなる。
- 3) LINESIZE によって端末の論理行サイズを変更する。指定可能なサイズは 1 ~ 136 である。
- 4) DOT によって SEARCH コマンドにおけるトリプルドット . . . に使うドット記号を変更する。
- 5) PREFIX によって SIGMA 領域のファイル参照に用いる PREFIX を変更する。
- 6) DISPLAY によって現在のシステム定数の状況が端末に表示される。

3. 12 SIGMA 領域の設定と消去

- 1) SIGMA 領域の設定. SIGMA 領域は

userid. SIGMAi . DATA (i = 1 , 2 , . . . , 9 , 0)

という名前のデータ・セットである。この領域はユーザが初めて自分の userid の下に SIGMA ファイルを作成しようとしたときに設定される。最初は i=1 のデータ・セットである。以後は不足した場合に i=2, 3, ... に従って、次々に設定される。各領域のサイズは 3 メガバイトである。新設増設に際しては、ユーザに新しく SIGMA 領域を設定していいか否かを問い合せてくるので、それに答えればよい。

最初の SIGMA 領域が設定されるとき、ファイル名を管理するための B 木 (B-tree) 用のデータ・セット

userid. B TREE. DATA

も同時に作成される。このように SIGMA 領域の管理用の B 木は別のデータ・セットに作られるが、MEMO 領域管理用の B 木は同じ MEMO 領域内に作られる。

なお、SIGMA システムで使用するデータ・セットと DD 名は以下の通りである。

データ・セット名	DD 名
MEMO. DATA	FT70F001
B TREE. DATA	FT90F001
SIGMA1. DATA	FT80F001
SIGMA2. DATA	FT81F001
⋮	⋮
SIGMA0. DATA	FT89F001

- 2) SIGMA 領域の共有. SIGMA システムでは SIGMA 領域を共有することができる。アクセス権の設定は、図 3 に示したように READ/WRITE について個別に行いたいわけであるが、現在のところ本システムを記述している言語 FORTRAN 77 に問題点があるために、UPDATE まで含めた

アクセス権の設定をしないと共有ができない。

アクセス権の設定(変更)は TSS コマンド PERMIT を用いて、

```
PERMIT SIGMA1. DATA USER (uid1, uid2, ..., uidn) ACCESS (UPDATE)
```

```
PERMIT BTREE. DATA USER (uid1, uid2, ..., uidn) ACCESS (UPDATE)
```

のように行う。SIGMA 領域は BTREE. DATA と SIGMA_i. DATA ($i=1, \dots, 9, 0$) からなるので、特に BTREE. DATA に対するアクセス権の設定(変更)を忘れないように注意して欲しい。

3) SIGMA 領域の消去。この領域の消去は MEMO 領域の場合と同様に TSS コマンドの DELETE を用いて行う。

3.13 コマンド入力に関する注意

SIGMA システムではコマンドやそれに必要なオペランド類の入力を空白で区切って1行に何個でも並べてよい。ただし、空白が意味をもつような文字列を入力する場合は、その入力行の残りは無視されて、プロンプトが出される。また文字列の入力時には代用復改記号の置き換えが行われる。コマンドやオペランド類を1個ずつ入力していくと、次の入力に対するプロンプトが出されるが、1行の複数の入力をした場合には、途中のプロンプトは出されずに、必要な次のプロンプトが出される。

LIST(N/Y)? などにおける選択は省略すると前の方の文字が選ばれる。この場合はNが選ばれる。(N OR Y)? の場合はN又はYを必ず選択しなければならない。

コマンド名の入力に際しては、他のコマンドと衝突が生じない最小限のプレフィクスを入力すれば十分である。例えば、LIST に対しては LI で十分であり、LOOK に対しては LOO で十分である。

4. SEARCH コマンドの機能

SEARCH コマンドについては、3.8 で簡単に説明したが、その機能についてもう少し詳しく説明することにする。このコマンドは元来一つの独立した検索システムとして開発したもの[11]に改良を重ねて[12]、[10]、SIGMA システムにおける1つのコマンドとして採用したものである。

4.1 特長

SEARCH コマンドの第1の特長は、区切り語、キーワードの集合からパタン・マッチング・マシンという一種の有限オートマトンを作り、それが対象となるファイル、すなわちテキスト系列の上を左から右へ一度走る間に全ての仕事を済ませる点にある。この他の主な特長は次の諸点にある。

1) $x_1 \cdots x_2 \cdots x_3$ の形のキーワードの指定ができる。これによって、キーワードの出現順序に意味があるような検索が可能になる。例えば "x followed by y" の形のテキストの検索はトリプル・ドット \cdots (以後 \circ で示す)を用いて $A_i := x \circ y$ と指定することによって簡単に行える。

2) 自由なレコード形式に対して適用できる。ファイルは文字列であるので、レコードは登録されたレコード区切り語と呼ばれる文字列にはさまれた部分文字列として、検索時に自動的に切り出される。

3) 複数質問を同時に処理できる。SEARCHコマンドは1度ではあるが、ファイルをフルサーチするので、必要な時間は必然的に走査対象のファイルの長さに比例したものになる。しかし論理式を最大99個、結果の得られる質問式(論理式)32個を同時に処理することができるので、質問1個当たりの検索時間は比較的短いことになる。

4) 検索能力が高い。キーワードの出現をマークする代わりに、出現回数をカウントするようにし、必要に応じて、初期状態への failure transition を禁止できるようにした。更に論理式に式変数、整数、種々の演算子を使えるようにしたために、検索能力が飛躍的に向上した。なお能力についてはオートマトン論的観点から定性的に評価してある[10]。

4.2 区切り語

区切り語にはレコード区切り語(RECORD DELIMITER)とアイテム区切り語(ITEM DELIMITER)がある。いずれも空でない文字列である。レコード区切り語は既に説明したように、仮想的なレコードを取り出すためのもので、アイテム区切り語は、論理式の評価の時点を示すためのものである。すなわち、アイテム区切り語を検出した時点で論理式の評価が行われる。区切り語は

```
RECORD DELIMITERS
D1 :=
:
ITEM DELIMITERS
D1 :=
:
OTHER ITEM DELIMITERS
D1 :=
:
```

の順序でそれぞれ最大99個まで登録できるが、次の点に注意して欲しい。

1) ITEM DELIMITERS のところでアイテム区切り語を実際に登録するとレコード区切り語はレコード区切り語固有の働きをし、OTHER ITEM DELIMITERS のプロンプトは出ない。1個もアイテム区切り語を登録しないと、レコード区切り語はアイテム区切り語としても機能する。この場合は OTHER ITEM DELIMITERS のプロンプトが出され、そこで登録したアイテム区切り語は固有の働きをする。

2) レコード区切り語を全く登録しなければ、ファイル全体が1つのレコードと見なされてサーチを行う。この場合は検索結果は1件か0件であるが、この1, 0でもって、ファイル全体の特性を知ることができる。

なお、レコード区切り語とアイテム区切り語の関係は図5に示すようなものである。図において、 R_i はレコード区切り語を E_j はアイテム区切り語と示し、 r_k は仮想的なレコードを示す。

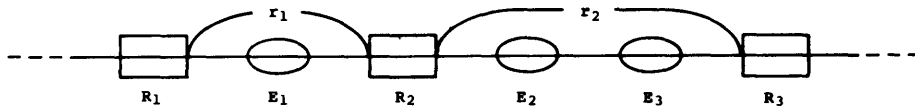


図5. レコード区切り語とアイテム区切り語

4.3 キーワード

キーワードには通常の空列でない文字列としてのキーワードの他に、トリプル・ドット \cdots (\circ で示した) を含む $x_1 \circ x_2 \circ \cdots \circ x_n$ の形のキーワードとがある。キーワードの登録はKEYWORDS に続くプロンプト $A1 := , A2 := \cdots$ に従って行う。個数は最大 99個である。 $A1, A2, \dots$ をキーワード変数と呼ぶ。 A_i のとる値は割り当てられたキーワードの出現回数であり、初期値と論理式が評価された時点での値はともに0である。

4.4 論理式

論理式はキーワード変数 ($A1, A2, \dots$)、式変数 ($Z1, Z2, \dots$)、整数、 ϵ 変数 E 、括弧 $[,]$ 、論理演算子 ($or(,), and(,), not \wedge$)、比較演算子 ($\langle, \rangle, =, \langle =, \rangle =, \langle \rangle$)、算術演算子 ($+, -, *, /$) を用いて作られるものである。論理式の登録は LOGICAL FORMULAE に続くプロンプト $Z1 := , Z2 := \cdots$ に従って行う。個数は最大 99 個である。 $Z1, Z2, \dots$ を式変数と呼ぶ。

論理演算子は次のように解釈される。

$$\alpha, \beta = 1 \quad (\alpha > 0 \text{ 又は } \beta > 0 \text{ のとき})$$

$$\alpha \cdot \beta = 1 \quad (\alpha > 0 \text{ かつ } \beta > 0 \text{ のとき})$$

$$\wedge \alpha = 1 \quad (\alpha \leq 0 \text{ のとき}),$$

ただし真値を1 (偽値は0) で表わしている。

各演算子の優先順位と自明でない演算子の意味は次表で与えられる。

演算子	優先順位	意味
*	1	
/	1	
+	2	
-	2	

=	3	
$\langle =$	3	\leq
$\rangle =$	3	\geq
\langle	3	
\rangle	3	
$\langle \rangle$	3	\neq

\wedge	4	not
\cdot	5	and
,	6	or

論理式の登録において、次のような省略形が許される。

- $Z_i := ,$ (全てのキーワード変数の論理和)
- $Z_i := \cdot$ (全てのキーワード変数の論理積)
- $Z_i := \neg$ (全てのキーワード変数の論理和の否定)
- $Z_i := +$ (全てのキーワード変数の和)。

ϵ 変数 E は空列を検出するためのもので、レコード区切り語が見つかった時点で1になり、アイテム区切り語が見つかり論理式が評価された時点では0の値をもつものである。したがって、レコード区切り語が連続している文字列、つまり空列が検出されることになる。

論理式1個に対して1つの検索結果が得られるので、論理式を質問と見ることができる。或る論理式を中間的な式変数としてだけ使用し、質問とは見なさないようにすることができる。それには、

$$Z_i := A_1 . A_2 /$$

のように、論理式の後にスラント/を置けばよい。スラントの付かない論理式は32個まで許される。

なお、論理式の評価はアイテム区切り語を検出するたびに、 Z_1, Z_2, \dots の順序で行われる。式変数の初期値は、 $Z_i = ivz(i)$ である。ここに $ivz(i)$ は論理式が

$$Z_i = f_i(E; A_1, \dots, A_n; Z_1, \dots, Z_m) \quad (1 \leq i \leq m)$$

で登録されているとき、

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

で定義される。

4.5 論理式の評価に関する注意

論理式はアイテム区切り語を検出するたびに Z_1, Z_2, \dots の順序で評価される。したがって、例えば

$$Z_i := Z_i, A_3$$

における右辺の Z_i は1時点前の値である。このように、式変数を一時記憶として使うこともできる。

このシステムでは、入力した全ての論理式において、

- 1) ϵ 変数 E は使用されていない、しかも
- 2) $Z_i := f(Z_j)$ ならば $j < i$ である

場合には、各アイテム区切り語において評価された論理式の論理和が1であるようなレコードを検索結果として出力するようになっている。すなわち、1), 2) が満される場合には、ユーザは標準的なサーチをしているものと見做して、論理式 $Z_i := f_i$ を

$$Z_i := Z_i, f_i$$

と置き換えて通常の処理をするわけである。

4.6 ユニバースの制限

キーワードの割り当て $A1 := x$ によって, $\Sigma^* X \Sigma^*$ の言語, つまり x は含まねばならないが x の前後は何でもいような文字列が検出されることになる. そこで, パタン・マッチング・マシンにおける初期状態への failure transition を禁止することによって, この Σ^* (何でもいいという部分) を登録された区切り語やキーワードの部分列に限定できるようにした. このような方式の選択は

RESET (Y/N)?

でNを指定することによって行われる. (この機能については文献[10]を参照されたい).

```
DO: LIST EXAMPLE1
.....*aaa/bbbb*aabb/aaaabb*.....
```

```
DO: SEA
VERSION (D/E)? E
RESET (Y/N)?

RECORD DELIMITERS
D1:=*
D2:=

ITEM DELIMITERS
D1:=

OTHER ITEM DELIMITERS
D1:=/
D2:=

KEYWORDS
A1:=a
A2:=b
A3:=

LOGICAL FORMULAE
Z1:=A1.A2
Z2:=A1.A2
Z3:=

FILE:=EXAMPLE1
```

```
RETRIEVED TEXTS
TOTAL = 2
QUESTION 1 (Z1) = 1
QUESTION 2 (Z2) = 2
```

```
FILE:=
LIST OF RESULTS (N/Y)? Y
QUESTIONS:=1 2
```

```
QUESTION 1 (Z1) = 1
NO. 1( 11)
```

aabb/aaaabb

```
QUESTION 2 (Z2) = 2
NO. 1( 8)
```

aaa/bbbb

```
NO. 2( 11)
```

aabb/aaaabb
DO: _

DO: LIST EXAMPLE2

.....DABDAAABBBEDA AAAABBBBAAAAD.....

```

DO: SEARCH
VERSION (D/E)? E
RESET (Y/N)? N

RECORD DELIMITERS
D1:=D
D2:=

ITEM DELIMITERS
D1:=

OTHER ITEM DELIMITERS
D1:=

KEYWORDS
A1:=A
A2:=B
A3:=BA
A4:=A...B...A
A5:=B...A...B
A6:=

LOGICAL FORMULAE
Z1:=[A1=A2].^A3
Z2:=[A1=2*A2].A4.^A5
Z3:=

FILE:=EXAMPLE2
    
```

```

RETRIEVED TEXTS
TOTAL           =      3
QUESTION 1 (Z1) =      2
QUESTION 2 (Z2) =      1
    
```

```

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

QUESTIONS:=1 2

```

AB
AAABBB
    
```

```

QUESTION 1 (Z1) =      2
NO.      1(      2)
QUESTION 2 (Z2) =      1
NO.      2(      6)
    
```

```

AAAABBBBAAAA
DO: _
    
```

```

QUESTION 2 (Z2) =      1
NO.      1(     12)
    
```

4.7 簡易版

以上で述べてきたように SEARCH コマンドは、相当きめ細かい検索を行うために、利用が面倒であるという難点もある。そこで簡易版を用意した。これは

VERSION (D/E)?

において D を選ぶことによって使える。D 版では 3.8 で挙げた例に示されるように、レコード区切り語、キーワード、論理式だけを指定すればいいので比較的簡潔である。この版ではレコード区切り語はアイテム区切り語としても機能する。

SEARCH コマンドの機能を示すために人工的な例を 567 ページと 568 ページに挙げておいた。3.8 の例も併せて参考にして欲しい。最初の例はレコード区切り語 * がアイテム区切り語としても働

き、他にアイテム区切り語/をもつ例である。ファイル EXAMPLE1 からは、レコードとしてドットの部分を除けば、aaa/bbb と aabb/aaaabb がとり出されることになる。アイテム区切り語/で論理式が評価されるので、前者は論理式 Z1 を満たさない。

二番目の例は、RESET でNが選択されているので、この場合にはユニバースが { A, B, D } 上の文字列に制限される。結局 Z1 によって $A^n B^n$ の形の文字列が、Z2 によって $A^n B^n A^n$ の形の文字列が検出されることになる。

5. ソーティングにおけるキーの切出し

REFILEと SORT コマンドによるソーティングにおけるキーの切出しは、キー切出し仕様に従って行われる。この仕様は FORTRAN における format と同様の形式をしている。

キー切出し仕様は、次の形である。

項目区切り, flist

項目区切りは、パタン又はパタンリストである。

パタンは、文字列を▼でくくったもの(▼を表すには▼▼としたもの)か、又はピクチャと呼ばれる、A, I, D から成る文字列である。これら A, I, D はそれぞれ英字、数字、英数字以外の文字を表すのに用いられる。パタンリストはパタンをコンマで区切って並べ、(,) でくくったものである。なお項目区切りにおいてのみ、空列▼▼を指定することができる。

flist は切出し仕様項目、又はこれをコンマで区切って並べたものである。

切出し仕様項目は、次のいずれかの形である。

[r] 切出し記述子

[r] (flist)

ここで r は符号なし整数で、反復子と呼ばれる。反復子は後続の記述子又は flist を r 回繰り返すことを意味する。反復子が省略された場合は 1 とみなされる。

切出し記述子にはソーティングのためのキーを読込むための読込指示子と、文字列を適当に読飛ばすための読飛ばし指示子の 2 つがある。

読込指示子は次の形である。

$$\left\{ \begin{array}{l} A \\ I \end{array} \right\} [r] \left[\left\langle \left\{ \begin{array}{l} \text{パタン} \\ \text{パタンリスト} \end{array} \right\} \right\rangle [/ \text{▼文字列▼}] \right]$$

ただし文字列は長さ r 以下のものである。A は任意の文字を、I は数字を読込むことを指示し、r は読込む文字数の上限を表し、< に続くパタン又はパタンリストは、そこに示される文字列を発見したら読込みを終了することを指示し、/ に続く文字列は、読込まれた文字が r 個に満たないときに、それを補うように指示するものである。/以降を省略した場合には、A, I に対してそれぞれ、空白、9 が補われる。

読飛ばし指示子は次のいずれかの形である。

$$X [r] \left[\left\langle \left\{ \begin{array}{l} \text{パタン} \\ \text{パタンリスト} \end{array} \right\} \right\rangle \right]$$

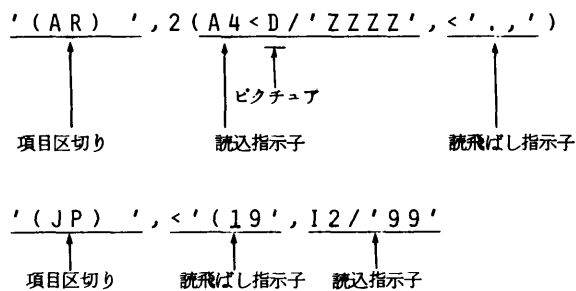
$$\left\langle \begin{array}{l} \text{ボタン} \\ \text{ボタンリスト} \end{array} \right\rangle$$

上の形は、rで示される文字数だけ、ただしくに続くボタン又はボタンリストが示す文字列が見つかるまで読み飛ばすことを指示するもので、下の形は、単にボタン又はボタンリストが示す文字列が見つかるまで読み飛ばすことを指示するものである。

1つのレコードに対するキーの切出しは、このキー切出し仕様に従って次のように行われる。

まず、対象となるレコードの上で、項目区切りをさがし、(項目区切りに空列▼が指示されると、無条件にみつかることになる。) みつかった項目区切りに対応するキー切出し仕様に従ってキーの切出しを行う。そのとき、同時に、他の項目区切りをさがしており、別の項目区切りが見つかったら、現在の仕様の切出しを終了して、(必要ならば省略値を補って) みつかった項目区切りに対応するキー切出し仕様に従ってキーの切出しを行う。以上の作業は、すべてのキー切出し仕様に対する切出しを終了するか、レコードの終りを検出したら終了する。

次にキー切出し仕様の簡単な例を示しておく。



6. LOGファイルの利用

SIGMAシステムにおいては、ユーザとの交信記録が、MEMO領域のLOGファイル用の領域Lに自動的に作られる。こうしてできたファイルをLOGファイルという。LOGファイルは、通常のSIGMAのファイルと同様に文字列の形をしたもので、内容はシステムが端末に出すプロンプトとそれに対するユーザの応答の記録である。LOGファイルに名前を付けて、SIGMAシステムで扱う一般のファイルとして扱うこともできる。

SIGMAシステムではコマンドやオペランド類の入力を端末からだけでなく、SIGMAシステムのファイルからも行うことができる。したがってLOGファイルをそのまま一種のコマンド・プロシジャーとして利用できるわけである。

端末とファイル間での入力流れの切り換えは、

. filename

の入力によって、任意の時点で行うことができる。ここで用いられるファイル名には、通常のSIGMAのファイル名の他に、特別な働きをする次の3つがある。

- 1) .K 以後の入力を端末に切り換えると同時に、新しくLOGファイルの作成を開始する。

- 2) .K.1 次の1行だけを端末に切り換える。このときの通信は記録されない。
- 3) .E 現在の入力の流れを終了させ、もとの入力の流れに戻す。その際現在の入力が端末からのものであれば、その LOG ファイルを閉じて領域 L のトップに置く。

SIGMA システムには、主な状態として SIGMA 状態と DO 状態の 2 状態があり、プロンプトはそれぞれ SIGMA>, DO: となっている。SIGMA システムが呼び出されると、まず SIGMA 状態になり、END 等の特別なもの以外のコマンドを投入すると LOG ファイルを開いて、通信記録がとられ始めて、DO 状態になる。この状態で通常のコマンドを使用できる。END コマンドは SIGMA 状態ではシステムを終了させるが、DO 状態では上に述べた .E と同じ動きをする。 .E は SIGMA 状態以外なら任意の時点で用いることができる点に違いがある。DO 状態で .E 又は END コマンドによってトップにある LOG ファイルを閉じると SIGMA 状態に戻る (図 6 参照)。

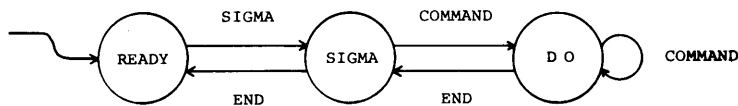
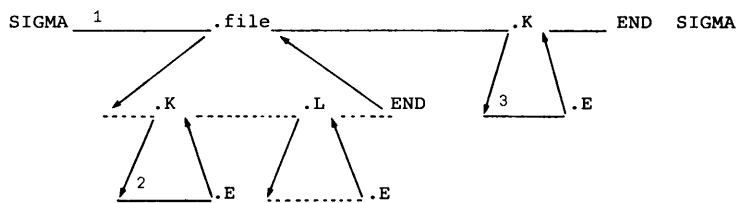


図 6. SIGMA システムにおける状態遷移

次に SIGMA システムの入力の流れを図 7 に示しておこう。図において .L は LOG ファイル用領域のトップにあるファイルを目指すので、実線 2 における通信記録である。SIGMA 状態に戻ったときの LOG ファイルの領域には上から順に 1, 3, 2 での通信記録が作られていることになる。



実線 —— は端末からの入力

点線 はファイルからの入力

図 7. SIGMA システムにおける入力の流れ

最後に簡単な使用例を挙げておこう。

```

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
    
```

7. TSS コマンドの呼び出し (TSS)

DO 状態又は SIGMA 状態から TSS コマンド・プロセッサを呼び出すことができる。これは SIGMA のコマンド TSS によって行われる。TSS を投入するとプロンプトが TSS: となり、それに続いて OS の TSS コマンドが使えるようになる。ただし、次のコマンドは使用できない。

```
LOGOFF
LOGON
EDIT
LIBRARY
TEST
TESTCOB
PROFILE
END
```

なお TSS 状態から抜け出すには、プロンプト TSS: に続いて END CR, 又は CR を入力すればよい。

8. おわりに

データとしては、情報の蓄積・検索、編集といった処理において、最も基本的である文字列を対象にして、処理の手段としては、汎用のものでは最も高速なパタン・マッチング・マシンの技法を駆使した一方向逐字処理に基本を置いた SIGMA システム第一版について解説した。SIGMA という名前の由来は、System for Information Gleaning and MAnipulation などと事故付けることも可能であるが、この第一版では文字列を対象としていて、通常こうした文字の集合を Σ と書くことと、数学の和の記号として馴染みが深いことにある。

ファイルの共有や外部のデータ・セットとの連携も可能であるために、計算機システムの種々の機能を有機的に活用できるわけで、蓄積・検索、編集をはじめとして、他のデータベース・システムに対するインターフェイスとしても有効であるものと期待している。

次の段階として、専用エディタの開発とともに、いわゆるデータベース・システムとしても機能できるシステムを志向して、データ記述言語や応用プログラム用の言語を開発して、拡充をはかりたい。利用者の御批判と御叱責をいただければ幸いである。

このシステムは昭和 55～56 年度における総合研究 (A) 及び昭和 56 年度の一般研究 (C) の一環として開発したものである。開発に当って絶えず助言と激励をいただいた加納省吾センター長に深く感謝します。また大型計算機センターでの開発に際して、援助をいただいた松尾文碩研究開発部長はじめ研究開発部の方々、特に二村祥一氏、末永正氏に感謝の意を表したい。

参考文献

1. Inose, H. (ed) Scientific Information Systems in Japan, North-Holland, (1981)
2. 松尾, 二村, 高木 データベース統合支援システム Adbis(1), 九州大学大型計算機センター 広報, 14, 2, 1981, 172-217.

3. Arikawa, S. and Kitagawa, T. Multistage Information Retrieval System Based upon Researcher Files, Proc. 2nd USA-Japan Computer Conference, 1975, 149-153.
4. Arikawa, S., Kano, S., Kitagawa, T. and Takeya, S. Organization and Use of Private Researcher Files in Scientific Research Work, Scientific Information Systems in Japan (Ed. Inose, H.), North-Holland, 1981, 43-50.
5. 有川 総合研究(A)「知識の表現とそれを利用する情報検索システムの研究」報告書, 1981.
6. 田中他 Period, 情報システム研究会資料, 1980.
7. 松田, 田中 ユーザ指向のデータ・ベース管理システムCOOD, 情報処理学会論文誌, 1980, 347-353.
8. 有川 研究者ファイルに基づく情報検索システム, 北海道大学大型計算機センター・ニュース, 13, 4, 1981, 5-16.
9. Aho, A. V. and Corasick, M. T. Efficient String Matching: An Aid to Bibliographic Search, C. ACM, 18, 1975, 333-340.
10. Arikawa, S. One-Way Sequential Search Systems and Their Powers, Bull. Math. Stat., 19, 1981, 69-85.
11. 有川, 武谷, 石橋 パタン・マッチング・マシンを利用する例文検索システム, 情報処理学会論文集, 1976.
12. Takeya, S. and Arikawa, S. TEXTIR-A Text Information Retrieval System, Res. Rept. Res. Inst. Fund. Inform. Sci., Kyushu Univ., 83, 1978.
13. 有川, 篠原, 白石, 玉越 研究者用のファイルシステム SIGMA について(I)(II), 情報処理学会論文集, 1981.
14. Arikawa, S., Shinohara, T., Shiraishi, S. and Tamakoshi, Y. SIGMA-An Information System for Researchers Use, Bull. Informatics and Cybernetics, 20, 1982 (in press)