APL を拡張した言語 E. APL†

渡辺豊英†† 松田 聡††† 萩原 宏††††

E. APL は APL の特徴の下に，データベース，リスト処理などのデータ処理への適応のために拡張され，APL の構文則・評価則に従っている．拡張の主な視点は会話プロセスでの処理，より簡潔な記述機能，多様なデータ表現を可能とするための機能などである．

E. APL と APL の一番大きな相異は "General Array" を導入したことに因っている．この General Array は配列をリストで連結して階層構造として実現されたものであり，レコード構造・木構造・リスト構造・集合族などの表現を可能としている．General Array の能力とその操作機能により，表構造のデータベースを APL に比べて容易に実現できる．それは，APL の柔軟な配列の扱いに加えて，レコード構造の活用と集合族の扱いができるからである．データベースの構築者は E. APL の機能によって，あたかも文字処理をするかのごとく，小規模，個人用のデータベースを作成できる．さらに，その応用例として本文では，情報検索システムを取上げ，APL との対比の下に検討を加えている．

1. はじめに

計算機技術の進展と共に，数多くのプログラミング言語が設計・開発され，人間と計算機とのインタフェースの役割を演じてきた．そこにはそれぞれの設計思想と目的用途に従った機能性・記述性・表現性・簡潔性などが基本となっている．APL (A Programming Language)[1] もそのひとつであり，アルゴリズムの簡潔な記述を目的として K. E. Iverson によって，考案された言語である．この言語の流用には，マイクロプログラミング，スイッチング理論，オペレーションズ・リサーチなどのアルゴリズムの簡潔な表現があり，彼自身それを試みた[2]．その後，会話型プログラミング言語として改編され，今日 APL といえば会話言語のことを指している（本稿でも APL とは会話型プログラミング言語を指している．）．

APL は簡潔な構文則と数学的な記法に基づいて演算子と被演算数の単純な組合せにより，アルゴリズムを表現することができる．また，データ構造として配列を基本とするが，演算プロセスで動的にその配列構造を変えることや，豊富な演算機能で配列構造を扱えることなどの特徴を有している．このような特徴の下に，各分野へ適応するように拡張され，機能強化の面から検討されて派生言語が開発された．それらには AHPL, APLG などがある．AHPL[3] は並列処理・非同期処理などのハードウェア機能の記述が可能なように拡張されたシステム記述用の言語である．また，APLG[4] はグラフィック用に機能拡張を図った図形表現・表示用の言語である．それらは APL の記述性・機能性，データの柔軟な扱いに注目して APL の拡張を図り，APL の基本概念をその目的に合致させた言語と思われる．しかし，両言語は基本的に操作機能の拡張に留り，データ構造には注視していない．図形データを扱う場合に，画面を構成する要素は階層的なデータの表現となり易く，配列だけでは図形データの表現力に欠ける．また，システム記述でも，データや制御の流れを表現するだけでなく，装置などの表現を行う場合には不十分である．

さて，拡張言語 E. APL (an Extended APL) はデータ構造の視点から検討を加え，階層データやデータ間の関係付けなどの表現を可能とするデータ構造を導入した言語である．すなわち，データベース，リスト処理などのように多重なデータやデータ間の関係を扱うことができる．本稿では，この拡張言語 E. APL の拡張概念を述べ，E. APL で扱うデータ構造の表現力を整理する．さらにデータベースの適応性を APL と比較して論ずる．

2. 拡張機能の概要

E. APL は APL の特徴に基づいて，データベースやリスト処理などのデータ処理への適応を試みて拡張された言語であり，データ構造・操作機能・制御構造などから検討を加えて設計された[5]．

† E. APL: an Extended APL by TOYOHIDE WATANABE (Data Processing Center, Kyoto University), SATOSHI MATSUDA (Nippon Electric Co. Ltd.), and HIROSHI HAGIWARA (Department of Information Science, Kyoto University).

†† 京都大学大型計算機センター
††† 日本電気(株)
†††† 京都大学工学部情報工学教室

2.1 拡張の視点

拡張は，プログラムの記述・手続きの実行における機能，多様なデータを表現するためのデータ構造とそれを扱う機能などからなりたっている．

i）APL で扱う配列は評価の都度，その大きさ・次元などを変え得るが，属性の同一な要素の集りでなければならないので，多様なデータを表現する場合には限界がある．たとえば，レコード構造などは表現できない．すなわち，データとデータ，または集合と集合に存在する関係などは表現できない．これは APL がアルゴリズムの記述に対して強力ではあるが，データの表現に対してそれ程考慮されていなかったことによる．しかし，データベース，リスト処理などのデータそのものの扱いや，その表現に対して適応させようとする場合に，配列だけでは不十分である．E. APL は「General Array」[6] を導入して，データの表現力をたかめている．

ii）一方，会話のプロセスでは，試行錯誤の下に計算処理，データ処理が行われ，評価された結果を確認して次の手続きが実行されるのを「会話の手続き」とすれば，ある手続きの実行以前にもどってその手続きを再実行できることが必要である．すなわち，ある手続きの実行以前のデータ値が何らかの形で再利用できるのでなければならない．たとえば，データベースの処理にはある操作の実行以前までデータを回復することは，容易ではない．E. APL ではこのために，値の退避・回復機能によって対処している．

E. APL は APL をデータ処理への適応性という点から拡張され，操作機能・制御構造なども考慮されている．しかし，E. APL は必ずしも APL の機能すべてを含んで拡張されたのではなく，ある種の機能が割愛されている．追加・拡張された機能は，APL の構文則・評価則に合致させて，見かけ上何ら APL のプログラム構造と変るところはない．

2.2 主な拡張機能

i）General Array

General Array は配列構造の拡張であり，配列がリストで連結されたデータ構造である．すなわち，配列のある要素の値はポインタをとり，他の配列を連結している．したがって，「General Array は配列の要素が他の配列であるかも知れない構造である．」と定義付けられている．しかし，ポインタ値そのものはプログラムでは扱えない．

図1に示す General Array の $A[1;1]$ の値は，

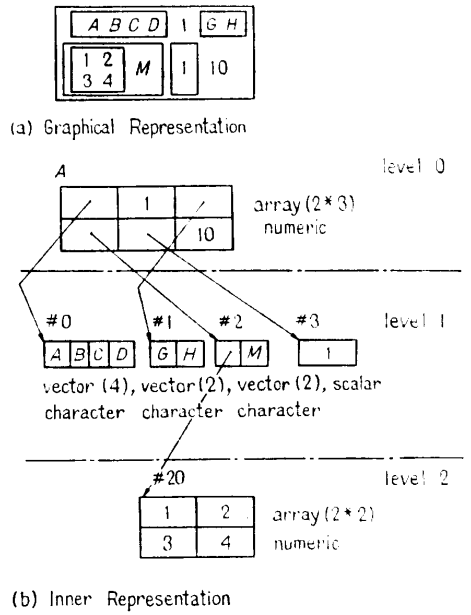

図 1 General Array
Fig. 1 General Array.

#0 へのポインタ値ではなく，#0 の配列である．この #0 は文字属性の配列であり，数値属性の配列 A とはその構造等が異なっている．すなわち，General Array をなす各配列ごとの構造，値属性は任意である．これらの配列で A をレベル 0 といい，General Array の総称名である．#0 はレベル 1 と呼ばれ，A の要素となっている．このようにレベル n までの任意の構造を表現することができる．

General Array の導入により，APL では表現できなかった種々のデータ構造を E. APL で扱うことができる．たとえば，リスト構造・木構造・レコード構造などである．すなわち，ポインタによって各配列を任意に連結し，各配列間の関係付けを行うことが可能である．これはデータ間の関係を扱ったり，データの集りを表現する場合に有効である．

一方，General Array の導入に伴って，それを扱う機能が追加されている．これらは主にこの構造のレベルを扱う機能や，この構造に伴う APL の拡張機能などがある．また，APL 本来の機能は General Array に対してもレベル 0 の視点で操作が可能である．すなわち，General Array の構造をプログラムが意識しなければ，配列と同じような扱いが可能である．

ii）値の退避・回復の機能

会話のプロセスでは試行錯誤的に手続きの検証を行う場合が多いと考えられ，手続きの再実行のために

```
      ∓ FUNC ..... declaration
[1]   .....
      .....
[I0]  FN1 Δ BACK1
[I1]  Δ ..... get-back operator
[I2]  FN2 Δ BACK2
[I3]  Δ
[I4]  .....
      .....
[N]   ∇
```

If *BACK*1 could succeed, *FN*1 can be evaluated and finished. Otherwise, in the next line the global variables are recoverd by Δ. Below here the same process is repeated.

図 2 値の退避・回復機能

Fig. 2 Backtracking mechanism (retirement and recover of values).

E. APL は定義関数の頭文で，一時その関数で使用される外部変数* の値を退避できる．値の回復にはその関数の手続き部分で回復機能を利用する．**図 2** では，頭文で「∇」の代りに「∓」を使用して退避を宣言している．したがって，この関数が呼び出されると，局部変数・定数以外の変数の値が退避され，次に手続きが実行される．そして，手続き中に「Δ」が使用されない限り，退避させられたデータ値はそのまま確保されており，関数の終了時に消去される．一方，途中で「Δ」が使用されれば（図 2 の文番号 I_1, I_3 など），その機能の実行後に変数の値が，この関数の実行以前にもどされる．

このようなプロセスによって，種々の関数を退避宣言された関数の下で同一のデータ値を与えて実行することができる．たとえば，図 2 では FUNC に含まれる関数 *FN*1, *FN*2 などがそれにあたる．パズルや決定問題などを解く場合に，また手続きのデバッグを行う場合にも会話的な処理には有効である．

ⅲ）制御構造の拡張

APL は配列演算のための機能が強力であり，単純なくり返し制御のための文（FOR 文，DO 文など）はなく，制御構造については goto 文（→）のみである．被演算数はラベル・定数ばかりではなく，変数・演算式も可能であり，かなり柔軟なプログラムを構成することができる（条件分岐では，→A/B と →B[A] とは同一である．）．

しかし，単純な判定操作が必要な手続きでは，goto 文のみによるプログラムは記述に冗長さを伴う．E. APL では goto 文に加えて条件文を導入した．この条件文は LISP の COND 関数の各要素 (p→e) に対応する．COND 関数と同等の手続きを goto 文と条件文によって **図 3**（a），(b) に示す．条件文は「IF～THEN～」型と同等の表現である．また，goto 文と組合せることにより，「IF～THEN～ELSE～」や「DO～WHILE～」型の制御構造を構成することができる．それを図 3 (c)，(d) に示す．すなわち，プログラムの構成に対して条件文の導入は，種々の制御構造を実現する．この手法は，A. L. Lim ら[7]の方法（⍳，⍕などの特別な内部関数を追加することによって，IF, REPEAT, DO, WHILE などの定義関数を構成する試み）よりも簡潔に実現できる．

```
      ∇ COND                    ∇ COND
[1]   →(P1≠1)/4          [1]   E1 Δ P1
[2]   E1                  [2]   E2 Δ P2
[3]   →END                [3]   E3 Δ P3
[4]   →(P2≠1)/7                 ⋮
[5]   E2                         ⋮
[6]   →END                [N]   ∇
      ⋮
[N]   END: ∇
(a) by goto statements    (b) by control statements

[K0]  →K2; E1 Δ P         [K0]  →K0; E Δ P
[K1]  E2                  [K1]  .....
[K2]  .....
(c) "IF-THEN-ELSE-"clause (d) "DO-WHILE-"clause
```

図 3 条件文の例

Fig. 3 Examples of control statements.

ⅳ）定義関数のパラメータ

APL では引数の個数と結果を持つか否かによって，**図 4** に示す定義関数の型がある．すなわち，引数は最大 2 個までである．しかし，多くのパラメータを渡したいときには不便である．もちろん，これらの引数は配列でもよいから，定義関数内で配列引数の要素を抽出することにより，多数のパラメータを渡すことができるが，引数に意味付けができない上，冗長なプログラムとなりやすい．この短所に関して，E. APL では

	非代入型 unassigned function	代入型 assigned function
零項 non-arg	*FUNC*	*R*←*FUNC*
単項 unary-arg	*FUNC X*	*R*←*FUNC X*
2 項 binary-arg	*X FUNC Y*	*R*←*X FUNC Y*

図 4 APL の定義関数の型

Fig. 4 Types of APL difined function.

* 外部変数とはその定義関数の局部変数以外のものをさす．

引数のない非代入型零項関数にパラメータ列を次のような形式で許した.

呼出し: $FN\ (e_1;\ e_2;\ e_3;\ \cdots;\ e_n)$

定義関数の頭文: $\nabla\ FN\ (X_1;\ X_2;\ \cdots;\ X_n)$

非代入型関数は演算式中に使用できないので, 構文上何ら矛盾を生じない.

E. APL の基本演算・操作機能は 90 個を越える. また, 会話型処理では数値計算に対して, 単精度の扱いしかないが, E. APL では必要に応じて2倍・4倍精度も扱う. この場合に, システムが自動的に入力データ長から表現可能な精度で, 有効数字を扱う方式を採っている.

3. E. APL のデータ表現

General Array の導入は多様なデータ構造の表現を可能とする. ここでは, それらのデータ構造の実現法と捕え方を述べる.

3.1 各データ構造

i) レコード構造: レコード構造を属性の異なったデータの集りと考えれば, General Array の各要素に配列を対応付けて, 属性の異なったデータの集りを表現することができる. しかし, 基本的には配列構造の拡張であり, 名前による参照はできない. 参照は配列の添字によらなければならない. E. APL は擬似的に General Array をレコード構造として捕えるのであり, 各要素の意味付けはそれを扱う手続きに任せられる. **図5**は PASCAL のレコード記述を General Array によって表現した構造である.

ii) リスト構造: General Array はその要素とし

```
Array [1..n] of
  record of
    a 1: number     ; *numeric
    a 2: city-name  ; *character
    a 3: population ; *numeric
         ⋮
  end;
```

(a) Description of PASCAL

1	TOKYO	1000	.....
2	KYOTO	140	.....
3	OSAKA	250	.....
4	·	·	·
·	·	·	·
·	·	·	·

(b) Representation in E. APL

図5 データ・エントリの例 (レコード構造の表現)

Fig. 5 An example of Data Entries.

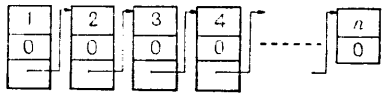

図6 リスト構造の表現

Fig. 6 List structure.

```
∇R←LIST N; T
[1] T←N,0
[2] R←T⍴0=N←N−1
[3] T←N, 0, ⊂T
[4] →2
[5] ∇
```

図7 リスト構造を作成するプログラム

Fig. 7 Program which generates list structure.

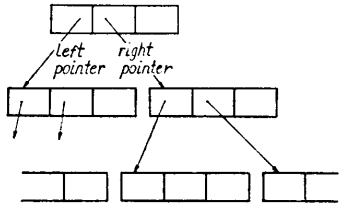

図8 木構造の表現

Fig. 8 Tree structure.

て, 陰にポインタを含んで構成されている. しかし, プログラムでは直接ポインタを扱えないが, ポインタで各要素が連結されることを利用して, 任意長のリストを構成できる. **図6**はひとつのリスト構造であり, **図7**がそれを生成するプログラムである.

iii) 木構造: リスト構造と同様に考える. すなわち, 必要な枝の数だけのポインタを General Array の要素に与える. この場合, 部分木の追加・削除は可変な配列構造の性質により, 容易に行える. **図8**に2分岐木の構造を示す.

iv) 集合の表現: APL の基本機能には集合操作に伴うものが少なくない. これは配列が動的にその要素数を変えることができ, 基本的に集合を扱っていることによる. それらの操作機能は次のようである. General Array の場合も見かけ上, レベル0では配列として扱えるために自然に適応できる.

$x \in y$: x の要素が y の要素か否かのチェック.

ρx : 要素の数, 要素なしにも有効である.

$x \subseteq y$: 包含関数のチェック.

$x \sim y$: y から x の要素を取除く.

$x \simeq y$: y から x を取除く.

$(\sim x \in y)/x, y$: x と y の和集合.

$(x \in y)/x$: y と x の共通集合

さらに, E. APL では General Array によって集合族の表現を可能とする. すなわち, General Array

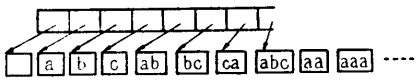

図 9 集合族の表現
Fig. 9 Representation of set family.

は配列を要素とする配列であるからである．たとえば，a, b, c の組合せにより成る集合は**図 9** に示した．レベル 1 の配列が a, b, c から抽出された (ab) などの集合を表現していて，レベル 0 は各要素を要素とする集合である．集合・集合族の扱いが容易に行えるのは，General Array を成す各配列が動的に変るからであり，それが集合（要素）の追加・削除に対して自然に整合するからである．

このように，E. APL では General Array とその操作機能により多様なデータ構造を表現できる．しかし，PASCAL などのようにデータタイプによってデータ構造を明示しないため，操作レベルで構造の把握が必要となってくるが，操作によってひとつの表現を多様な視点で捕え，種々の処理が可能である．

3.2 データベースの表現

E. APL は General Array によって各種のデータ構造を表現することができ，木構造を基本とした MUMPS（医療用データベース言語）と同等のデータベースを構成することもできる[8]．すなわち，General Array によって木構造のデータベースを作成し，E. APL 内の機能によってそのデータベースを扱うことができる能力は，MUMPS とほぼ等価である．しかし，E. APL でデータベースを表現する場合には木構造よりも表構造の方が次の点ですぐれている．

i) E. APL は配列構造を基本として操作する機能が多い．

ii) E. APL はデータの参照手段として配列の添字によるため，特に木構造への参照機能が十分とは言いがたい．

iii) E. APL の算術演算子は配列計算に強力であり，数値データの処理に対しては表形式がよい．

表構造によって表現されたデータベースを E. APL の操作機能により，特にデータベースとして意識することなく扱うことができる．この場合に，General Array の適応はレコード構造・リスト構造・集合族の表現によって，かなり柔軟なデータの表現性に富むからである．また，数値・文字データが混在した表現も可能であり，E. APL の算術処理用の演算子の活用が図れる．

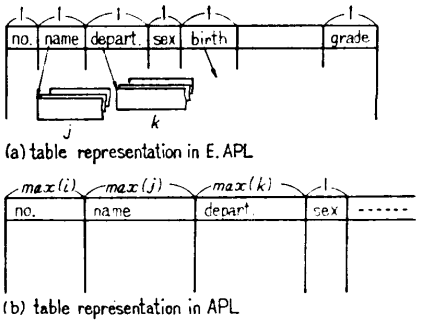

図 10 E. APL と APL のデータベースの表現
Fig. 10 Database representation.

以上のことに基づいて学生に関するデータベースを表現する．**図 10** (a) がそれを示している．各データ項目は General Array のレベル 0 の配列要素で表現され，その実体はレベル 1 で配列として表現される．この場合，レベル 0 は数値属性である．同様な APL の表現を (b) に示す．(a), (b) では General Array の利点が表われている．すなわち，(b) では，まず数値・文字データの混在が許されないので，この配列は文字属性でなければならない．次に，文字データは配列のひとつの要素に対してひとつしか含まれず，ストリングなどの扱いは数要素をひとつのフィールドとして管理しなければならない．したがって，フィールドの扱いに工夫が必要である．また，このデータベースを扱うプログラムが陽に意識してデータ項目の長さを扱わなければならない．ある試みはフィールド名を関数名で扱い，フィールドに参照する場合には関数を介してデータを扱っている[9]．

このように，APL と E. APL のデータベースの表現に対してはかなりの相異がある．加えて，APL ではすべてが文字データであるため，算術用の演算子の強力な機能を直接利用することができない．

一方，E. APL でデータベースを表現する場合に，それを扱うプログラムはデータ項目の数・データ数に依存しない．データの評価時に，その大きさなどを調べて処理できるからである．すなわち，E. APL ではデータと手続きは互いに独立していて，データの追加・削除に対しても何ら手続きの変更を必要としない．

4. 情報検索システムへの適応例

E. APL の適応の一例として，データベースを取上げ，E. APL の機能だけで容易に情報検索（特に，文献検索）機構を構成できることを示す[10]．

4.1 検索のためのデータ表現

情報検索とはデータベースの各レコードに集合演算を施して，条件に合致する目的のレコードを選びだすことであり，一般に検索効率を向上させるために原データ用のファイル（原ファイルと呼ぶ）以外に，各データ項目ごとの転置ファイルを利用している．また，検索結果を管理するための表（集合管理表と呼ぶ）などが必要である．

上記の主なファイル・表を E. APL ではどのように扱うのかを述べる．文献データベースを取上げ，その構成を示す．

i）原ファイル：データは文献情報の各項目から成り立っている．たとえば，文献番号（数），題名（文字），分野（文字）などとする．これらはレコード構造により，**図11**（a）のように作成される．すなわち，General Array のレベル0で各データ項目を x 方向に与え，y 方向に各レコードを与える．そして，レベル1で文字属性のデータの実体を表現する．

ii）転置ファイル：転置ファイルはあるデータ項目に含まれるすべてのデータから抽出された語の登録表であり，語と文献番号列がひとつの対をなしている．キー語のそれを図11（b）に示す．すなわち，語と文献番号のエントリをレベル0で与え，その実体をレベル1で表現する．この場合，ある文献が追加されて，このファイルの再構成が必要なときでも論理的には，データの結合操作を行えばよい．また，このファイルが B-tree で構成されるか否かは今問題としない．必要ならば，木構造で構成する．

iii）集合管理表：この表は検索の結果，選択された文献番号の集合を管理する．そして，いくつかの集合から構成されるか，各集合がいくつの要素から成るかは，General Array の性質から問題とはならない．図11（c）にそれを示す．すなわち，レベル0で検索結果の集合へのエントリを示すポインタ値を管理する．レベル1でその実体を表現する．

各ファイル・表を大域変数として扱い，それぞれの名前を RECORD, KEYWORD, TEMPSET などとする．ここで，本質的なのは多様な構造を表現できるということに加えて，General Array を成す各配列が動的に構造を変えることができることである．

一方，APL では3.2節で述べたように，このような簡潔な表現はできない．

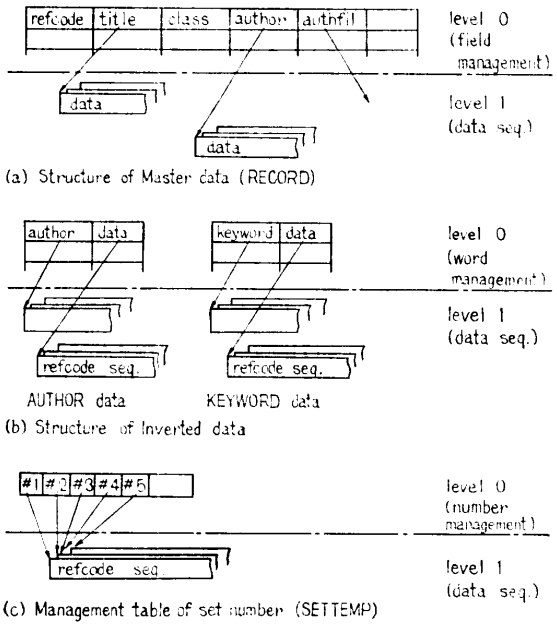

図 11 情報検索システム (IRS) の各データ表現
Fig. 11 General arrays for constructing IRS.

4.2 検索用コマンドの構成

検索コマンドの構文として**図12**のものとする．これらのコマンドの解釈を考える．EQ などの比較演算子で検索条件に合致する文献集合をみつけて，その集合に論理演算子 AND などによって集合演算を施す．最後に，FIND で検索条件に合致した文献番号の集合を管理する．すなわち，FIND コマンドそのもののプロセスと考えずに，EQ, AND などと同一の操作機能レベルとして，FIND の機能を捕えれば，検索コマンドの構文は右から左へと解釈してもよいことになる．これは E. APL（または APL）の構文則「右から左への評価手順」と一致する．たとえば，

FIND (AUTHOR EQ "KYOTO") OR
KEYWORD EQ "DATA"

の構文は，E. APL の演算式

1(AU="K")∨KW="C"

```
⟨FIND command⟩=FIND ⟨search condition⟩
⟨search condition⟩=⟨search term⟩|(⟨search condition⟩)
                   ⟨logical operator⟩ ⟨search term⟩
⟨search term⟩=⟨field name⟩ ⟨relational operator⟩ ⟨word⟩
⟨relational operator⟩=EQ|NE|GT|GE|LT|LE|....
⟨logical operator⟩=AND|OR
⟨field name⟩=⟨variable⟩
⟨word⟩=⟨number⟩|⟨string⟩
⟨AND command⟩=ANDS ⟨set number sequence⟩
⟨set number sequence⟩=⟨eet number⟩|⟨set number sequence⟩,
                       ⟨set number⟩
⟨OR command⟩=ORS ⟨set number sequence⟩
⟨DIF command⟩=DIF ⟨set number sequence⟩
⟨DISPLAY command⟩=DISPLAY ⟨set number⟩
```

図 12 検索コマンドの構文則
Fig. 12 Syntax rules of IR search-command.

```
        ∇R←X EQ Y; X1
[1]   X1←⍳↑X
[2]   R←X1[2]⌷(∧/X1[1]=Y)=1
[3]   X←↓X
[4]   R←  ⌷X=
[5]   →1
[6]   ∇
```

(a) EQ function

```
        ∇R←X AND Y
[1]   R←(X∈Y)/X
[2]   ∇
```

(b) AND function

```
    ∇ FIND X; X1
[1] ··RESULT(S) WERE NOTHING..⌷ =∈X
[2] SETTEMP←SETTEMP,X
[3] ··THERE WERE RESULTS.··
[4] ··      : ··,⍕⍴X,··DOCUMENTS.··
[5] ··THE NUMBER OF THIS SET IS··,⍕⍴SETTEMP,··.··
[6] ∇
```

(c) FIND command

```
        ∇ANDS X; X1; X2
[1]   ··PARAMETER ERROR··  ⌷X=
[2]   X2←↑X
[3]   X←↓X
[4]   →(X=  )/7
[5]   X2←(X2∈↑X)/X2
[6]   →3
[7]   FIND X2
[8]   ∇
```

(d) ANDS command

図 13 検索コマンドなどの定義関数例

Fig. 13 Defined functions of search-command.

と全く同一の構造と考えられる.

E. APL が右から左へと一文を評価するという特徴が英文の命令形の構成として一致していることに帰因する. すなわち, 一番左の演算子が右の各演算子の修飾を受けていることは, 英文に対応付ければ, 副詞や副詞句・節によって命令動作が修飾されていることと同じである.

さて, このような検索コマンド・操作子を E. APL で構成した例を**図 13** に与える. この定義関数は図 12 の FIND, EQ, AND などである. FIND は非代入型単項関数であり, EQ, AND は代入型 2 項関数として記述され, 演算式中に使用される.

EQ, AND を APL で表面的に記述することは可能であるが, 各関数の手続きで扱われているデータが General Array を基本としているために, APL ではこの記述では表現できない. APL では General Array で表現されているデータベースを 3.2 節で述べたように扱い, かつそれを扱う操作子がその項目の構造を把握している必要がある. したがって, APL では EQ, AND などの定義関数ははるかに複雑な手続きとなる. FIND も同様である.

ANDS コマンドも集合番号列の論理和操作であり, 図 13 (d) に示す定義関数である. たとえば,

ANDS 3,5,6

では, 3,5,6 という集合番号の列に対してその論理和を施す. ここで, 集合番号間の区切記号「,」は E. APL の「ストリング結合」を行う基本演算子であり, E. APL の構文上の規約がそのまま生かされている. 図 13 (d) では, [1] で X の要素数 (集合番号の数) が空か否かを判定している. [2] から [6] は集合番号を順次取出して, それらの集合に含まれている文献番号を比較して両集合にあれば, $X2$ に入れる. すべての集合の文献番号列の共通部分がみつかれば, [7] で FIND 関数を呼んで集合管理表に登録し, 検索結果の情報を出力するなどの後処理を行う.

同様な考えが ORS, DISPLAY などにも適用できる. すなわち, General Array を種々の角度から各ファイル・表に適応させ, E. APL の構文則の下にコマンドをそのままの形式で構成できる.

APL ではまず各ファイル・表の表現と扱いが E. APL のごとくに容易ではないため, 付加機能・システムの変更等を行わなければならない.

5. おわりに

本文では APL の機能を拡張した E. APL の主な機能を示し, 特にデータ構造の拡張である General Array のデータの表現によって, データベースの表現などを論じた. そのひとつの適用例として情報検索システムの実現にあたって, E. APL の機能のみにより自然な形式で, 各レコードの構成, コマンドの構成ができることを APL との対比の下に示し, E. APL の記述力の優ることを述べた.

E. APL には約 90 個の基本演算子があり, 主に APL の基本演算子に General Array の操作機能を加えた. しかし, リスト構造・木構造の扱いには十分な能力があるかは, まだ検討がなされていない. リストの扱いでは LISP 程度の機能が必要であり, 検討・実証を行う予定でいる. また, データベースに対しても数値データによる検索・処理の扱いを本文では言及していないが, E. APL の機能からかなり容易に操作可能と思われる. 改めて報告する予定でいる.

E. APL の General Array はかなりの分野に適応性を持つと思われる. コンピュータ・グラフィックや記号処理には大いに検討の余地がある. また, 本文では General Array とその操作子の能力に視点を注い

だが，値の退避・回復などは実際のプログラミングに期待される機能と思われる．

謝辞 当課題をすすめるにあたり，色々とご指示頂いた福井大学渡辺勝正教授，姫路工業大学宮脇冨士夫氏に深謝の意を表わすと共に，萩原研究室の方々に感謝する．また，本稿の細部にわたって不備な点を指摘して頂いた査読者に感謝する．

参考文献

1) Iverson, K.E.: A Programming Language, p. 286, John Wiley & Sons. Inc., New York (1962).
2) Iverson, K.E.: A Programming Language, Proc. AFIPS 1962 SJCC, Vol. 21, pp. 345-351.
3) Hill, F.J.: Introducing AHPL-Introducing Computers Hardware Description Language, Computer, pp. 28-30 (Decem. 1974).
4) Giloi, W.K. and Eneavnacas, J.: APLG-An APL Based System for Interactive Computer Graphics, Proc. AFIPS 1974 NCC, Vol. 43, pp. 521-528.
5) 渡辺ほか：拡張された APL…E. APL…の機能，情報処理学会第17回全国大会，282, pp. 551-552 (1976).
6) Ghandour, Z. and Mezei, J.: General Arrays, Operators and Functions, IBM J. RES. DEVELOP., 17, pp. 335-352 (July 1973).
7) Lim, A. L. and Lewis, G. R.: Toward Structured Programs in APL, The Computer Journal, Vol. 18, No. 2, pp. 140-143 (1974).
8) Watanabe, T., et al.: An Experimental Assessment of an Extended APL to MUMPS Language, MEDIS '78, pp. 443-446.
9) Mahood, C.E.: Data Base Retrieval System (DBRS)-A Personalized Data Base System for the APL User, APL 76: Conference Proceedings, Ottawa, pp. 267-283 (Sept. 1976).
10) 渡辺ほか：E. APL による IR システム構成への一考察，情報処理学会第19回全国大会，3I-11, pp. 865-866 (1978).

(昭和53年10月19日受付)
(昭和54年 4 月19日採録)