統計数理研究所彙報
第31巻 第2号(1983)

APL 未組み込み演算の代替処理

統計数理研究所 仁 木 直 人

(1983年12月 受付)

1. は じ め に

APLは,豊富で強力な演算子群を持ち,処理過程を簡潔に表現できる,応用分野の広い,会話型の言語である.とくに線型計算やデータの操作・作表などが容易に行なえることから「統計計算用言語」といっても良いであろう(例えばNiki(1981)など).

現在多くの計算機で使用されているAPLプロセッサは,K.E. Iversonが創案・研究したプログラミング言語“A Programming Language”(Iverson(1962))に基づいてIBMのWatson研究センタで開発されたプロセッサを起源としている.この「計算機に乗せる」過程で,Iversonの原本にあった言語仕様の内いくつかは捨てられ,逆にいくつかが付け加えられた.また現在も言語としての発展は継続されているが,新しい機能が付け加えられることはあっても,一度捨てた仕様を拾う方向の努力はあまりされていないようである.

捨てられた仕様の中で重要と思われるのは,「木構造」データとその操作,ファイルの操作,陰関数表現およびいくつかの有用な演算である.

「木構造」データに関する部分を現存のデータ構造と演算で補うことは,可能ではあるが,極めて煩わしい部類に入るであろう.リスト処理を受け持つ補助プロセッサをPL/Iなどで作成してAPLから呼出すようにするか,あるいは,計算機メーカなどによる新版APLの開発を待つしかないが,実現する見通しはかなり暗い.

ファイルの操作に関する部分は,ハードウェアやオペレーティング・システムとの関連から,各機種ごとに独自に処理を行なっているので,Iversonの構想をそのまま持ち込むことは無理といわざるを得ない.本稿でも,ファイル操作については言及しない.

陰関数表現も,現在のAPL記法と相入れない面があり,将来も組み込まれることはないであろう.この代替処理は,対象となる演算子によって異なるため,一般形を得ることは困難である.ただ,同等の働きをする陽な表現は,本稿で追加する処理を含めれば,比較的簡単に得ることができるから,個々の陰関数表現に対する特別な代替処理が必要とは思われない.

その他の仕様(主に演算子)で未だ組み込まれていないものについては,そのほとんどに代替処理が実現できる.その中には,比較的使われることが多いと思われるもの,使えると大いに便利なものも少なくない.本稿の第一の目的は,未組み込みの演算の代替処理をまとめて一種のknowhow集として提供し,APLユーザの負担を一層軽くしようとすることにある.さらに,APLプログラムの例題集として,APL教育の一助となれば幸いである.

「代替処理」をユーザ定義関数の形で書くことは比較的たやすい.しかし,多くのAPLプロセッサが,ジェネレータとしてではなく,インタプリタとして作られていることを鑑みると,複数行に書くことは大いに不利である.ましてやループを含むようでは演算子の代替処理とはいい難い.インタプリタ方式では,各行ごとにかかるオーバヘッドがある上,ジャンプ先の行の

サーチにはかなりの時間を要すからである．本稿では，少々頑固のようではあるが，1行中(60字程度以内）に収まる代替プログラムを提示する．これらをユーザ定義関数形式にして使用することも結構であるし，プログラム中にインラインで（適当に変形して）組み込むこともまた自由にできよう．さらに，APLの入門者にとっては，APLの各演算子の典型的な使用例として参考になるのではないかと考える．また多少重複する演算があっても，オーバヘッドを減らすこととユーザの活用の便宜上から，ワーク用の変数はなるべく使わないようにしている．

「代替処理」プログラムは，対象とする演算にもよるが，なるべくデータの種類（数値，文字の別）や形（スカラ，ベクトル，マトリクスなど）によらず，多くの場合にそのまま使えるように考えたつもりである．データの形により演算そのものの働き方が異なる場合や，データの種類や形を考慮することで著しくプログラムが簡単になる場合には，そのときの別法も付記しておいた．また□IO (index origin) が0または1のどちらにセットされていても構わないように，注意を払ってある．

記述を簡単にするため，本稿では以下のコンベンションを採用する．

A, B	任意の変数を表わす．
U, V	ベクトルを表わす．
M, N	マトリクスを表わす．
J, K, L	整数スカラを表わす．
S	任意の形の論理値変数を表わす．
T	論理値ベクトルを表わす．
X, Y	任意の形の数値変数を表わす．
P, Q	数値ベクトルを表わす．
W, Z	ワークとして用いる．

2. 代替処理プログラム

APL演算は，同じ形の変数間の演算（強制的に同形にする場合を含む）と，形（とくに配列の次元）が異なる変数間の演算に大別できる．後者では，特定の軸を指定して，その軸に沿う演算を行なう場合が多い．軸指定が必要な演算に関しては，本稿では，最後の軸が指定された場合のみを提示することとし，他の軸への変更は読者に委ねる．変更方法が分らない，または，あまりに複雑となる場合は，前後に転置“⍉”による軸の入れ替えを行なえばよいだけである．

APLプロセッサによって，演算記号などの記法に相違のある場合がある．ただし，入出力に関係するところを除けば，少し考えれば分かるようなことばかりで，記法上の問題は実際上少ない．本稿は，HITAC Mシリーズ用のAPLプロセッサ（VOS3 APL-05-00（日立製作所(1983)))に準拠したプログラムとしている．もし，他のプロセッサで使用する場合は，言語仕様の違いを確認する必要がある．

プログラムは次の形式に則って記述する．「未組み込み演算」はIversonの原本に準拠した表現としてある．

```
Operation（その簡単な説明）
    未組み込み演算 ： 代替処理プログラム
          注釈（必要ならば）
    未組み込み演算 ： 代替処理プログラム  ─┐
                                          │
          注釈（必要ならば）              ↓  別法がある場合
    ............
```

2.1. 集合に関する演算

「集合」はベクトル（すなわちオーダのついた集合で，スカラは要素1個から成る「集合」とはみなさない）として取扱えばよい．集合に関する演算で組み込まれていないものは，包含関係を調べる述語群，和・差・積集合，および集合要素の直積（Cartesian product：各集合から1要素ずつ取出して作り得る全ての集合の集合）である．

要素の直積は，任意の数の集合に対して定義されているから，このままでは現在の APL 体系に適合しない．そこで，ふたつのマトリクス（ベクトルを含む）の行に関する直積（双方の各1行を取出して連結することにより作り得る総てのベクトルを各行として持つマトリクス）を計算するプログラムを用意することにする．この方が，応用範囲も広く，より一般的であろう（具体例は付録を参照）．

同じ要素が複数個含まれている「集合」については，演算の前後で，同一要素を単数とみなす限りにおいて正しい結果を生じる．ただし，要素の直積については，応用の面からも，総て異なる要素とみることにする．

Inclusion（U が V に含まれれば 1，そうでなければ 0）

`U ⊆ V    : ∧/U∊V`

Strict inclusion（U が V に含まれ V は U に含まれなければ 1，そうでなければ 0）

`U ⊂ V    : (~∧/V∊U)∧∧⌿U∊V`

Similarity（U と V がおなじ集合であれば 1，そうでなければ 0）

`U ■ V    : ∧/(V∊U),U∊V`

Union（合併集合，和集合）

`U ∪ V    : U,(~V∊U)/V`

Difference（補集合，差集合）

`U ∆ V    : (~U∊V)/U`

Intersection（共通部分，積集合）

`U ∩ V    : (U∊V)/U`

Cartesian product（両方の行ベクトルを連結してできるベクトルの集合，行の直積）

`M ⊗ N    : ((1↑⍴N)⌿M),((⍴⍴N)↑(1↑(⍴M)×⍴N),⌽⍴N)⍴N`

一般形．M および N はベクトルでも構わない．なお，ここでは「圧縮 ”／”」の発展である「複写」機能を用いている．この機能を欠くプロセッサに対するプログラムは少々複雑になるが，1行には収まる程度である．

`M ⊗ V    : ((⍴V)⌿M),((1↑⍴M)×⍴V)⍴V`

右辺がベクトル（縦ベクトル）のときは簡単な形にできる．M はベクトルでも構わない（そのときは ”1 ↑” も不要）．

⊗ / M　　: (,⍉M)[((⍴Z)⍴⎕IO++\0,1↓W)+Z←⍉(W←(1↓W)⍴1↑W)⊤(⍳×/W←⍴M)-⎕IO]

マトリクスの各列ベクトルの Cartesian product.

2.2. 特殊ベクトル，特殊行列

特殊ベクトル，特殊行列として，それを表わす特別な記号が割当てられているのは，0 または 1 の要素がある決まった配置に並べられたものである．そのうち，0 のみまたは 1 のみから成るベクトル・マトリクスの作り方は，形 "⍴" の基本的な応用であるから，ここでは取り上げない．その他のものもごく簡単に作ることができ，APL 入門者には，よい練習問題となろう．

Iverson の原本にはないが，与えられたベクトル（スカラは 1 次元ベクトルとしない）を対角とする行列を必要とすることも多いので，その作り方をも付記しておく．

正方行列の場合，三角行列の作り方がより単純になる．どこを省略できるのかは，ただちに分かるであろう．

J-th unit vector（第 J 要素のみが 1 で，他が 0 の K 次元ベクトル）

$\epsilon^J(K)$　　: K↑(-J)↑1

　　または (⍳K)=J+⎕IO-1

Prefix of weight J（始めの J 要素が 1 で，他が 0 の K 次元ベクトル）

$\alpha^J(K)$　　: K↑J⍴1

Suffix of weight J（最後の J 要素が 1 で，他が 0 の K次元ベクトル）

$\omega^J(K)$　　: (-K)↑J⍴1

Identity（対角要素が 1 で，他は総て 0 の $J\times K$ 行列）

I(J,K)　: (⍳J)∘.=⍳K　　　　　　一般に使える

　　または (J,K)⍴1,K⍴0　　　　$J\leqq K+1$ のとき適用可能

Diagonal（U を対角とし，他は 0 の正方行列）

DIAG U　: W⍴U,(W←2⍴⍴U)⍴0

Logical superdiagonal（対角要素の L だけ上にある要素が 1 で，他は 0 の $J\times K$ 行列）

^{L}I(J,K)　: (L+⍳J)∘.=⍳K

　　$L<0$ の場合は，$|L|$ だけ下の要素が 1 となる．

Superdiagonal（対角の $L\geqq 0$ だけ上を U とし，他は 0 の正方行列）

^{L}DIAG U　: (1 ¯1×L+W)↑W⍴U,(W←2⍴⍴U)⍴0

　　$L\geqq 0$ だけ下を U とするものは，"1 ¯1" を "¯1 1" と替えればよい．上記の Logical superdiagonal に従って U の要素の再配置（後述）を行なっても実現できる．

Upper right（対角要素を含まない右上三角行列）

◹(J,K)　: (⍳J)∘.<(⍳K)-(K>J)×K-J

Upper left（対角要素を含まない左上三角行列）

◸(J,K)　: ⌽(⍳J)∘.<(⍳K)-(K>J)×K-J

　　または ((⍳J)∘.+⍳K)≤(2×⎕IO-1)+J⌊K

Lower right（対角要素を含まない右下三角行列）

◿(J,K)　: (⌽⍳J)∘.<(⍳K)-(K>J)×K-J

　　または ((⍳J)∘.+⍳K)≥(2×⎕IO)+J⌈K

Lower left（対角要素を含まない左下三角行列）

◺(J,K)　: (⍳J)∘.>(⍳K)+(J>K)×J-K

2.3. 論理値を結果とする演算

結果が論理値となる演算，とくに後で要素の選択のために使う論理データを作るための演算が，現在の APL プロセッサでは多数落とされている．一般の変数に対する Forward/Backward set selector および Maxima/Minima selector は，ユーザ定義関数として複数行に書く他ないようにも思えるかも知れない．しかし，工夫次第で，下の掲げるような方法が可能である．APL 入門者にとって，ここで行なわれている演算を追跡してみることは，有益であろう．なお，"⌈" および "⌊" は，現存の APL プロセッサでの用法（Ceiling および Floor）とは異なっているので，下線を付して区別しておく．

Maximum prefix（0 が出るまで，始めから 1 が続く部分）

`α / S   : ∧\S`

Maximum suffix（0 が出るまで，後ろから 1 が続く部分）

`ω / S   : ⌽∧\⌽S`

Forward set selector（始めから全て異なる要素が続く部分）

`σ / A   : ∧\∧≠((⍳1↑W)∘.≥(⍴A)⍴⍳¯1↑W)∨(W⍴(1↑0⍴A),,A)≠(W←(¯1+¯1↑⍴A),⍴A)⍴A`

任意の変数（スカラを除く）に適用できる一般形．

`σ / U   : ∧\∧/(W∘.≤W←⍳⍴U)∨U∘.≠U`

ベクトル（重複を許す集合）については比較的簡単にできる．

Backward set selector（後ろから全て異なる要素が続く部分）

`τ / A   : ⌽∧\⌽∧≠((⌽⍳1↑W)∘.<(⍴A)⍴⍳¯1↑W)∨(W⍴1↓,A)≠(W←(¯1+¯1↑⍴A),⍴A)⍴A`

任意の変数（スカラを除く）に適用できる一般形．

`τ / U   : ⌽∧\⌽∧/(W∘.≥W←⍳⍴U)∨U∘.≠U`

ベクトル（重複を許す集合）について適用できる．

Maxima selector（論理データにより選択された要素中（最後の軸に関して）最大であるもの）

`S ⌈ X   : S×W=⍉(⌽⍴X)⍴⍉⌈/S×W←X+1-⌊/,X`

S と X の形は同じである必要がある．もし $X>0$ であれば，W として X そのものが使え，W は不要である．

`T ⌈ X   : T\W=⍉(⌽⍴W)⍴⍉⌈/W←T/X`

論理データがベクトルの場合である．X の形は，最後の軸の次元が T の次元と一致していれば，他の軸に関して任意である．

`T ⌈ P   : T\W=⌈/W←T/P`

右辺もベクトルの場合は，簡単である．

Minima selector（論理データにより選択された要素中(最後の軸に関して)最小であるもの）

`S ⌊ X   : S×W=⍉(⌽⍴X)⍴⍉⌊/S×W←X-1+⌈/,X`

S と X の形は同じである必要がある．もし $X>0$ であれば，W として X そのものが使え，W は不要である．

`T ⌊ X   : T\W=⍉(⌽⍴W)⍴⍉⌊/W←T/X`

論理データがベクトルの場合である．X の形は，最後の軸の次元が T の次元と一致していれば，他の軸に関して任意である．

`T ⌊ P   : T\W=⌊/W←T/P`

右辺もベクトルの場合に適用できる．

2.4. 論理データに従った要素の再配置

論理データに従った要素の再配置の基本的な演算として，Compression と Expansion はどの APL プロセッサにも組み込まれている．しかし，便利で有用な演算である Mask と Mesh は，現在の APL 体系にそぐわない（3個のオペランドがいる）ことからも，全くといっていいほど，組み込まれる可能性は薄い．また，Compression と Expansion についても，論理データとしてはベクトルのみとしているので，一般の場合を述べておくことにする．

Compression（0 に対応する要素を捨て，1 に対応する要素のみで新しい配置をつくる）

`T / B   : T/B`

論理ベクトルに従う圧縮．基本的演算のひとつである．

`S / B   : (,S)/,B`

S と A は同じ形である必要がある．結果は 1 に対応する要素から成るベクトルとなる．

Mask（論理値に従ってふたつの変数のどちらか一方の要素を選択して配置する）

`/A,T,B/ : (ρA)ρ((ρA)ρT)⊖A,[⎕IO-0.5]B`

A と B の形は同じである必要がある．T は A および B の最後の軸と同じ次元のベクトルで，T の要素の値に従って，0 の場合は A，1 の場合は B の各列ベクトルが選択される．

`/A,S,B/ : (ρA)ρS⊖A,[⎕IO-0.5]B`

A, B, S は同じ形である必要がある．

`/X,T,Y/ : ((~T)\(~T)/X)+T\T/Y`

数値データには加算が使えるので，プログラムが簡単になる．

`/X,S,Y/ : (X×~S)+(Y×S)`

X, Y, S が同じ形である場合も同様である．また，文字データも，`⎕AVιA` などにより数値化してからこれらの演算を行ない，その結果を再び `⎕AV[•••]` により文字化することが別法として考えられる．

`/U,T,V/ : (U,V)[(T×ρT)+ιρT]`

ベクトルに対しては，インデクスにより選択が可能である．

Expansion（1 に対する場所に要素を順に配置し，0 に対応する場所は，もとが数値か文字かによって，0 または空白を埋める）

`T \ B   : T\B`

論理ベクトルに従う拡張．基本的演算のひとつである．

`S \ V   : (ρS)ρ(,S)\V`

1 のあるところに，V の要素をバラバラにして順に配置する．

Mesh（0 のところには左側の変数の要素，1 のところには右側の変数の要素を，それぞれ順に取出して配置する）

`\A,T,B\ : (ρW)ρ((ρW)ρT)⊖(W←(~T)\A),[⎕IO-0.5]T\B`

A と B の形は，最後の軸の次元を除いて，同じである必要がある．T の 0 要素の数および 1 要素の数は A および B の最後の軸の次元に等しくなければならない．

`\U,S,V\ : (ρS)ρ(U,V)[((~,S)\ιρU)+(,S)\(ρU)+ιρV]`

U および V の次元は，S に含まれる 0 および 1 の要素数にそれぞれ一致している必要がある．

\X,T,Y\ : ((~T)\X)+T\Y

数値データには加算が使えるので，プログラムが簡単になる．

\P,S,Q\ : (ρS)ρ((~,S)\P)+(,S)\Q

この場合も同様である．なお，文字データに対しては，⎕AVιA などで数値化してからこれらの演算を行ない，その結果を再び⎕AV[•••]などにより文字化することも別法として考えられる．

\U,T,V\ : (U,V)[((~T)\ιρU)+T\(ρU)+ιρV]

全部がベクトルのときは，インデクスをうまく作るだけで済む．

3. おわりに

ここまで見てきたように，Iverson の原本にあるベクトルおよびマトリクスに関する演算のほとんど総てが，1行のプログラムで代替することができた．本稿でとくに採上げなかった演算は，次に挙げるものを除いて，現行の APL プロセッサにすでに採り入れられているものか，あるいは，その代替処理が明らかに分るようなものである．

代替処理が1行では実現できなかった演算とは，マトリクスの対応する行の間の「指標調べ」M ι N である．左側がベクトルの場合は現行の APL プロセッサでそのまま処理でき，また，行単位に「指標調べ」を実行したのち連結を行なう，複数行のプログラムは容易に作成できる．この演算を必要とするケースはごく少ないであろうから，あまり気にすることもないのであるが，ひとつだけ残ってしまったのはやはり形が悪い．うまいアイデアがあれば，ぜひご教授願いたい．

Iverson の原本にはないが，ときに必要となる Kronecker product は，外積の簡単な応用

```
((ρM)×ρN)ρ 1 3 2 4 ⍉M∘.×N
```

でできるので，参考のため，付記しておく．

なお，査読者の方々から有益かつ的確なコメントをいただいた．ここに記して感謝の意を表したいと思います．

参 考 文 献

日立製作所 (1983)．プログラムプロダクト VOS3 APL 言語／使用の手引，8090-3-223-40.

Iverson, K.E. (1962). *A Programming Language*, Wiley, New York.

Niki, N. (1981). APL programmes for canonical correlation analysis of categorical data, *Computer Science Monographs*, **16**.

付録：使用例

適当な検査データを与えたときの実行結果を以下に示す．注釈〔＊＊…＊＊〕中の括弧内は，説明を行なった節の番号である．与えた検査データを

変数名 ＝ $\boxed{\text{データの内容}}$

の形で表わし，代替処理プログラムおよびその計算結果を

〔番号〕 対象演算：代替処理プログラム
$\boxed{\text{計算結果}}$

のように示す．〔番号〕は，本文中に現われた順番に対応している．

```
** SET OPERATIONS (2.1) **

     +-----------+       +---------------+
U =  |2 4 6 8 10|   V =  |1 1 2 3 5 8 13|
     +-----------+       +---------------+

[1]  U ⊆ V     : ∧/U∊V
         0

[2]  U ⊂ V     : (~∧/V∊U)∧∧/U∊V
         0

[3]  U ≡ V     : ∧/(V∊U),U∊V
         0

[4]  U ∪ V     : U,(~V∊U)/V
         2 4 6 8 10 1 1 3 5 13

[5]  U Δ V     : (~U∊V)/U
         4 6 10

[6]  U ∩ V     : (U∊V)/U
         2 8

     +-----------+       +---------------+
U =  |2 4 6 8 10|   V =  |2 10 4 8 2 2 6|
     +-----------+       +---------------+

[1]  U ⊆ V     : ∧/U∊V
         1

[2]  U ⊂ V     : (~∧/V∊U)∧∧/U∊V
         0

[3]  U ≡ V     : ∧/(V∊U),U∊V
         1

[4]  U ∪ V     : U,(~V∊U)/V
         2 4 6 8 10

[5]  U Δ V     : (~U∊V)/U


[6]  U ∩ V     : (U∊V)/U
         2 4 6 8 10
```

```
** CARTESIAN PRODUCTS (2.1) **

     +----+      +-----+      +----+
M =  |1234|  N = |ABCDE|  V = |⊂⊃∩∪|
     |+=+≠|      |FGHIJ|      +----+
     +----+      |KLMNO|
                 |PQRST|
                 +-----+

[7]  M ⊗ N     : ((1↑ρN)⌿M),((ρρN)↑(1↑(ρM)×ρN),ΦρN)ρN
          1234ABCDE
          1234FGHIJ
          1234KLMNO
          1234PQRST
          +=+≠ABCDE
          +=+≠FGHIJ
          +=+≠KLMNO
          +=+≠PQRST

[8]  M ⊗ V     : ((ρV)⌿M),((1↑ρM)×ρV)ρV
          1234⊂
          1234⊃
          1234∩
          1234∪
          +=+≠⊂
          +=+≠⊃
          +=+≠∩
          +=+≠∪

[9]  ⊗ / M     : (,⍉M)[((ρZ)ρ⎕IO++\0,1↓W)+Z←⍉(W←(1↓W)ρ1↑W)⊤(ι*/W←ρM)-⎕IO]
          1234
          123≠
          12+4
          12+≠
          1=34
          1=3≠
          1=+4
          1=+≠
          +234
          +23≠
          +2+4
          +2+≠
          +=34
          +=3≠
          +=+4
          +=+≠
```

```
** SPECIAL VECTORS AND MATRICES (2.2) **

       +-+       +-+       +-+       +--------+
   J = |5|   K = |7|   L = |3|   U = |2 4 6 8|
       +-+       +-+       +-+       +--------+

[10]  ϵ^J(K)    : K↑(-J)↑1
          0 0 0 0 1 0 0

[11]            : (ιK)=J+⎕IO-1
          0 0 0 0 1 0 0

[12]  α^J(K)    : K↑Jρ1
          1 1 1 1 1 0 0

[13]  ω^J(K)    : (-K)↑Jρ1
          0 0 1 1 1 1 1

[14]  I(J,K)    : (ιJ)∘.=ιK
           1 0 0 0 0 0 0
           0 1 0 0 0 0 0
           0 0 1 0 0 0 0
           0 0 0 1 0 0 0
           0 0 0 0 1 0 0

[15]            : (J,K)ρ1,Kρ0
           1 0 0 0 0 0 0
           0 1 0 0 0 0 0
           0 0 1 0 0 0 0
           0 0 0 1 0 0 0
           0 0 0 0 1 0 0

[16]  DIAG U    : WρU,(W←2ρρU)ρ0
           2 0 0 0
           0 4 0 0
           0 0 6 0
           0 0 0 8

[17]  ^L I(J,K) : (L+ιJ)∘.=ιK
           0 0 0 1 0 0 0
           0 0 0 0 1 0 0
           0 0 0 0 0 1 0
           0 0 0 0 0 0 1
           0 0 0 0 0 0 0

[18]  ^L DIAG U : (1 ¯1×L+W)↑WρU,(W←2ρρU)ρ0
           0 0 0 2 0 0 0
           0 0 0 0 4 0 0
           0 0 0 0 0 6 0
           0 0 0 0 0 0 8
           0 0 0 0 0 0 0
           0 0 0 0 0 0 0
           0 0 0 0 0 0 0
```

```
** TRIANGULAR MATRICES (2.2) **

     +-+       +-+
J  = |3|   K = |4|
     +-+       +-+

[19]  ◥(J,K)   : (⍳J)∘.<(⍳K)-(K>J)×K-J
              0 0 1 1
              0 0 0 1
              0 0 0 0

[20]  ◤(J,K)   : ⌽(⍳J)∘.<(⍳K)-(K>J)×K-J
              1 1 0 0
              1 0 0 0
              0 0 0 0

[21]           : ((⍳J)∘.+⍳K)≤(2×⎕IO-1)+J⌊K
              1 1 0 0
              1 0 0 0
              0 0 0 0

[22]  ◢(J,K)   : (⌽⍳J)∘.<(⍳K)-(K>J)×K-J
              0 0 0 0
              0 0 0 1
              0 0 1 1

[23]           : ((⍳J)∘.+⍳K)≥(2×⎕IO)+J⌈K
              0 0 0 0
              0 0 0 1
              0 0 1 1

[24]  ◣(J,K)   : (⍳J)∘.>(⍳K)+(J>K)×J-K
              0 0 0 0
              1 0 0 0
              1 1 0 0

     +-+       +-+
J  = |4|   K = |3|
     +-+       +-+

[19]  ◥(J,K)   : (⍳J)∘.<(⍳K)-(K>J)×K-J
              0 1 1
              0 0 1
              0 0 0
              0 0 0

[20]  ◤(J,K)   : ⌽(⍳J)∘.<(⍳K)-(K>J)×K-J
              1 1 0
              1 0 0
              0 0 0
              0 0 0

[21]           : ((⍳J)∘.+⍳K)≤(2×⎕IO-1)+J⌊K
              1 1 0
              1 0 0
              0 0 0
              0 0 0

[22]  ◢(J,K)   : (⌽⍳J)∘.<(⍳K)-(K>J)×K-J
              0 0 0
              0 0 0
              0 0 1
              0 1 1

[23]           : ((⍳J)∘.+⍳K)≥(2×⎕IO)+J⌈K
              0 0 0
              0 0 0
              0 0 1
              0 1 1

[24]  ◣(J,K)   : (⍳J)∘.>(⍳K)+(J>K)×J-K
              0 0 0
              0 0 0
              1 0 0
              1 1 0
```

```
** LOGICAL SELECTOR GENERATION (2.3) **

     +--------+       +-----------+       +----------------+
A =  |ABCDEFGH|   S = | 1 1 0 0 1 |   X = |  2  8  5  6  3 |
     |ABCAIJKL|       | 0 1 0 1 0 |       |  1  7  7 10  4 |
     |AAAAAAAA|       | 1 0 0 1 1 |       |  6  9  1  1  6 |
     |MNOPQRSS|       | 1 1 1 1 1 |       |  7  1  4  1  5 |
     +--------+       +-----------+       +----------------+
     +--------+       +-----------+       +-----------+
U =  |ABCAIJKL|   T = |0 1 0 1 0|     P = |1 7 7 10 4|
     +--------+       +-----------+       +-----------+

[25]  α / S   : ∧\S
          1 1 0 0 0
          0 0 0 0 0
          1 0 0 0 0
          1 1 1 1 1

[26]  ω / S   : ⌽∧\⌽S
          0 0 0 0 1
          0 0 0 0 0
          0 0 0 1 1
          1 1 1 1 1

[27]  σ / A   : ∧\∧⌿((⍳1↑W)∘.≥(⍴A)⍴⍳¯1↑W)∨(W⍴(1↑0⍴A),,A)≠(W←(¯1+¯1↑⍴A),⍴A)⍴A
          1 1 1 1 1 1 1 1
          1 1 1 0 0 0 0 0
          1 0 0 0 0 0 0 0
          1 1 1 1 1 1 1 0

[28]  σ / U   : ∧\∧/(W∘.≤W←⍳⍴U)∨U∘.≠U
        1 1 1 0 0 0 0 0

[29]  τ / A   : ⌽∧\⌽∧⌿((⌽⍳1↑W)∘.<(⍴A)⍴⍳¯1↑W)∨(W⍴1↓,A)≠(W←(¯1+¯1↑⍴A),⍴A)⍴A
          1 1 1 1 1 1 1 1
          0 1 1 1 1 1 1 1
          0 0 0 0 0 0 0 1
          0 0 0 0 0 0 0 1

[30]  τ / U   : ⌽∧\⌽∧/(W∘.≥W←⍳⍴U)∨U∘.≠U
        0 1 1 1 1 1 1 1

[31]  S ⌈̲ X   : S×W=⍉(⌽⍴X)⍴⍉⌈/S×W←X+1-⌊/,X
          0 1 0 0 0
          0 0 0 1 0
          1 0 0 0 1
          1 0 0 0 0

[32]  T ⌈̲ X   : T\W=⍉(⌽⍴W)⍴⍉⌈/W←T/X
          0 1 0 0 0
          0 0 0 1 0
          0 1 0 0 0
          0 1 0 1 0

[33]  T ⌈̲ P   : T\W=⌈/W←T/P
        0 0 0 1 0

[34]  S ⌊̲ X   : S×W=⍉(⌽⍴X)⍴⍉⌊/S×W←X-1+⌈/,X
          1 0 0 0 0
          0 1 0 0 0
          0 0 0 1 0
          0 1 0 1 0

[35]  T ⌊̲ X   : T\W=⍉(⌽⍴W)⍴⍉⌊/W←T/X
          0 0 0 1 0
          0 1 0 0 0
          0 0 0 1 0
          0 1 0 1 0

[36]  T ⌊̲ P   : T\W=⌊/W←T/P
        0 1 0 0 0
```

```
** COMPRESSION AND MASK OPERATIONS (2.4) **

     +------+       +------+
A =  |ABCDEF|  B =  |123456|
     |GHIJKL|       |7890+-|
     |MNOPQR|       |⊂⊃∩∪⊥⊤|
     |STUVWX|       |<≤=≥>≠|
     +------+       +------+
     +-----------+      +-----------------+      +-----------------+
S =  |0 0 0 0 0 0|  X = | 1  3  5  7  9 11|  Y = | 2  4  6  8 10 12|
     |1 0 1 0 1 0|      |13 15 17 19 21 23|      |14 16 18 20 22 24|
     |1 1 0 0 1 1|      |25 27 29 31 33 35|      |26 28 30 32 34 36|
     |1 1 1 0 1 1|      |37 39 41 43 45 47|      |38 40 42 44 46 48|
     +-----------+      +-----------------+      +-----------------+
     +-----------+     +------------+      +-------------+
T =  |1 1 0 1 0 0| U = |1 3 5 7 9 11|  V = |2 4 6 8 10 12|
     +-----------+     +------------+      +-------------+

[37]  T / B    : T/B
         124
         780
         ⊂⊃∪
         <≤≥

[38]  S / B    : (,S)/,B
         79+⊂⊃⊥⊤<≤=>≠

[39]  /A,T,B/  : (ρA)ρ((ρA)ρT)⊖A,[⎕IO-0.5]B
         12C4EF
         78IOKL
         ⊂⊃O∪QR
         <≤U≥WX

[40]  /A,S,B/  : (ρA)ρS⊖A,[⎕IO-0.5]B
         ABCDEF
         7H9J+L
         ⊂⊃OP⊥⊤
         <≤=V>≠

[41]  /X,T,Y/  : ((~T)\(~T)/X)+T\T/Y
          2  4  5  8  9 11
         14 16 17 20 21 23
         26 28 29 32 33 35
         38 40 41 44 45 47

[42]  /X,S,Y/  : (X×~S)+(Y×S)
          1  3  5  7  9 11
         14 15 18 19 22 23
         26 28 29 31 34 36
         38 40 42 43 46 48

[43]  /U,T,V/  : (U,V)[(T×ρT)+ιρT]
         2 4 5 8 9 11
```

```
** EXPANSION AND MESH OPERATIONS (2.4) **

      +--+      +----+      +-----+       +-------------+
A =  |AB|  B = |1234|  X = |  1 2|  Y = |  11 12 13 14|
     |CD|      |5678|      |  3 4|      |  15 16 17 18|
     |EF|      |90⊂⊃|      |  5 6|      |  19 20 21 22|
     |GH|      |∩∪⊥⊤|      |  7 8|      |  23 24 25 26|
     +--+      +----+      +-----+      +-------------+
     +---------+       +-----------------+
U =  |ABCDEFGH|   V = |1234567890⊂⊃∩∪⊥⊤|
     +---------+       +-----------------+
     +-----------------+
P =  |1 2 3 4 5 6 7 8|
     +-----------------+
     +-----------------------------------------------+
Q =  |11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26|
     +-----------------------------------------------+
     +------------+       +---------+
T =  |1 1 0 1 0 1|   S = | 1 1 1 1|
     +------------+       | 1 0 1 0|
                          | 0 1 1 1|
                          | 0 0 1 1|
                          | 0 1 0 1|
                          | 1 1 1 0|
                          +---------+

[44]  T \ B    : T\B
        12 3 4
        56 7 8
        90 ⊂ ⊃
        ∩∪ ⊥ ⊤

[45]  S \ V    : (ρS)ρ(,S)\V
        1234
        5 6
         789
          0⊂
         ⊃ ∩
        ∪⊥⊤

[46]  \A,T,B\  : (ρW)ρ((ρW)ρT)⊖(W←(~T)\A),[⎕IO-0.5]T\B
        12A3B4
        56C7D8
        90E⊂F⊃
        ∩∪G⊥H⊤

[47]  \U,S,V\  : (ρS)ρ(U,V)[((~,S)\ιρU)+(,S)\(ρU)+ιρV]
        1234
        5A6B
        C789
        DE0⊂
        F⊃G∩
        ∪⊥⊤H

[48]  \X,T,Y\  : ((~T)\X)+T\Y
         11 12  1 13  2 14
         15 16  3 17  4 18
         19 20  5 21  6 22
         23 24  7 25  8 26

[49]  \P,S,Q\  : (ρS)ρ((~,S)\P)+(,S)\Q
         11 12 13 14
         15  1 16  2
          3 17 18 19
          4  5 20 21
          6 22  7 23
         24 25 26  8


      +--+      +----+
U =  |AB|  V = |1234|
      +--+      +----+

[50]  \U,T,V\  : (U,V)[((~T)\ιρU)+T\(ρU)+ιρV]
        12A3B4
```

Substitute Programmes for the Operations Unavailable in Current APL Processors

Naoto Niki

(The Institute of Statistical Mathematics)

This article provides a number of one-line APL programmes to execute the operations not available in APL processors in current use, which are as follows:

a) set operations
   inclusion, strict inclusion, similarity, intersection, difference, union, Cartesian product;
b) generation of special vectors and matrices
   unit vector, prefix and suffix vectors of any weight, diagonal, superdiagonal and triangular matrices;
c) generation of logical selectors
   maximum prefix, maximum suffix, forward/backward selector, maxima/minima selector;
d) compression, expansion, mask and mesh operations.

The operations mentioned above were invented by K.E. Iverson himself in his "A Programming Language" but have not yet been implemented in spite of their usefulness.