

Steps Toward an APL Compiler --

Updated

Alan J. Perlis

Research Report #24

March 1975

Introduction

An APL processor is customarily implemented as an interpreter; each statement is parsed on each call for its execution. APL is typeless, the type of an operand being known (for certain) only at execution time. The shapes, ranks, and types of arrays, being subject to change during execution, make a useful pre-execution compilation phase difficult to define.

Furthermore, APL provides such a condensed notation that the code obtained from compilation would seriously erode the available storage in a user's work space. Remember that APL is usually implemented within a time-sharing system. Hence storage must be shared.

It is the purpose of this paper to describe some techniques that make an APL compiler feasible within a time-sharing system, where limited work space is the rule. The thrust is towards an implementation on any of the standard commercial mini-computers.

ranks taken by a given identifier over several statements.

Let us confine our attention to the assignment statement. If all of the contained identifiers are assumed to be rank- and type-invariant, then so must all sub-expressions not containing occurrences of

(1) a function call

and/or (2) $(X)\rho Q$ where X is an expression other than
 and/or (3) $(X)\phi Q$ a constant vector and Q is an expression.

We make a second hypothesis:

H2) Let us assume that subsequent executions of a function called from a statement yield a result of the same type and rank as on a first call from that statement.

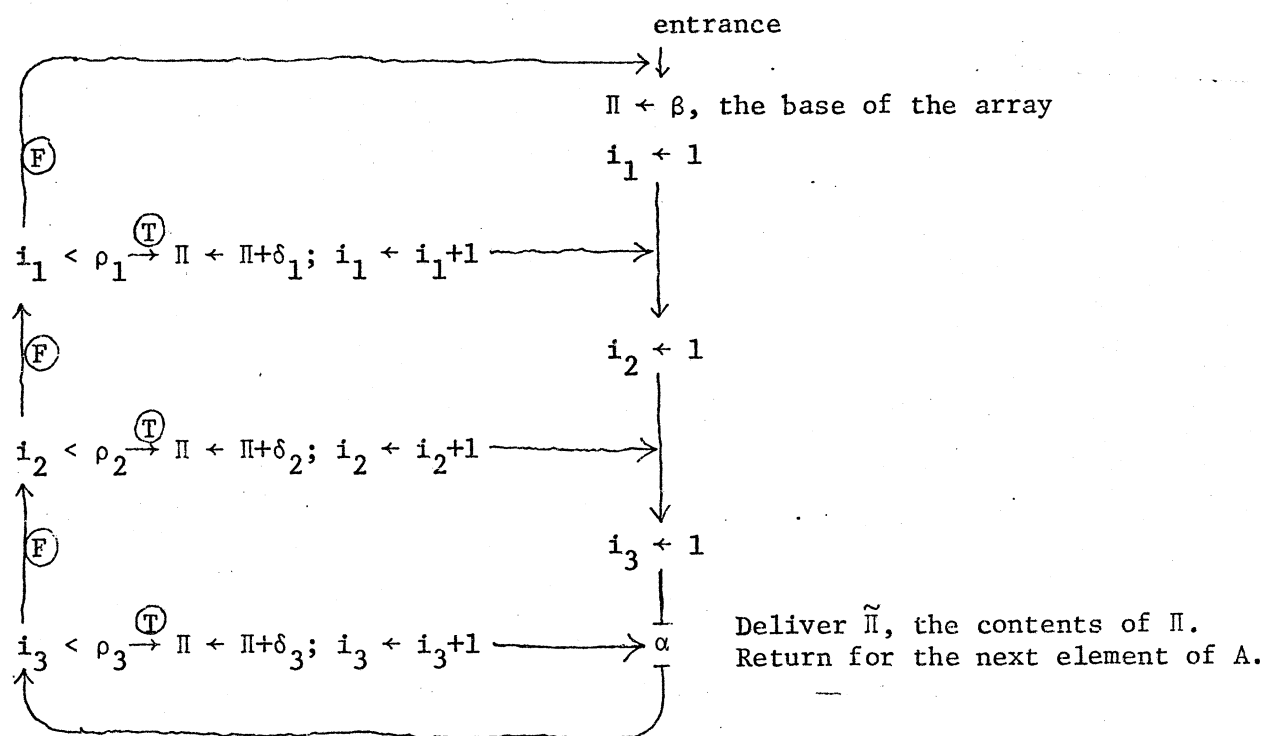
We will say that a statement not containing any occurrence of (2) or (3) and satisfying H1 and H2 is a compilable statement.

Since the hypotheses may be violated, the compilation, or the use of compiled code, is conditional. The relationships between compilation and execution for a given statement are illustrated in the example. (See the figure on page 21.)

The prefix contains the rank and type constraints of each section of the statement (there are three in the example). If any constraints are violated, the statement is placed in the "first passage" category; otherwise, in the "subsequent passage" category.

Ladders

A major preoccupation of an APL processor is the delivery (and receipt) of array elements in some required order. The most common is the ravel order. We may imagine that, for an array A of rank 3, whose shape vector is ρ_1, ρ_2, ρ_3 , the elements are delivered under control of the following network:



We call the delivery point α a splice. The network is linked to the user of the elements by a co-routine control. The network and the user program are linked co-routines.

We will symbolize this network with the following diagram called a

array is to be used but only on its rank. Hence they can be micro-programmed. a, b, ..., g are link addresses. Whenever PC reaches such a link address, a goto that address takes place and PC is augmented. When the code attached to a splice has been executed, the execution of the ladder continues from the ladder location specified by PC.

Given the δ_k , $M[i_1; i_2; \dots; i_n]$ is defined as

$$\beta + \sum_{k=1}^n (i_k - 1) \times G_k$$

where the G_k are defined recursively in terms of the δ_k :

```

for k ← n step -1 until 1 do
  G_k ← δ_k;
  for j ← k+1 step 1 until n do
    G_k ← G_k + G_j × (ρ_j - 1);

```

Conversely, the δ_k can be given in terms of the G_k :

```

for k ← n step -1 until 1 do (*)
  δ_k ← G_k;
  for j ← k+1 step 1 until n do
    δ_k ← δ_k - G_j × (ρ_j - 1);

```

When $\delta_k = 1$, $k = 1, 2, \dots, n$, the G_k 's are computed from the much simpler

```

G_n ← 1;
for k ← n-1 step -1 until 1 do
  G_k ← G_{k+1} × ρ_{k+1};

```

An array is maintained in storage as the ravelled vector of its elements and some attached control information -- base, rank, and shape vector.

g) if $t_1 = \tilde{\Pi}$ then begin $t_3 \leftarrow i_1$; goto h end;

h) $\hookleftarrow y$; $\hookleftarrow z$

i) $t_3 \leftarrow 1 + \rho V$;

j) $\Pi \leftarrow t_3$; $\hookleftarrow y$;

k)

l)

m)

n)

} co not used oc; *(these are NOPs)*

x , y , and z are co-routine link registers. Thus the $\hookleftarrow x$ in (e) simultaneously performs:

PC + 1 of ladder for A goes into x
and PC of ladder for B gets set from x

and control is now in ladder B, having just been in ladder A. Initially, the three link registers point to:

x : 1 of B
 y : n of W
 z : i of V

and the entrance is to m in ladder A.

Control is always within a splice or at an s_i , t_i , v_i , or σ in some ladder. The co-routine links are between splices of different ladders. When a splice code is completed, the PC of the attached ladder points to the next ladder action to be performed.

Thus regardless of the shapes of B, A, V, and W about nineteen machine instructions and about fifty bytes of ladder data make up the compilation.

$$\rho'[I] \leftarrow L/(I=C)/\rho A$$

$$I \leftarrow I+1$$

$$\beta' \underline{\text{is}} \beta$$

To compute δ' : like (a1) with ν' for ν , ρ' for ρ , G' for G .

(b) *Reversal (monadic ϕ)*: $\phi[k]A$

$$G' \underline{\text{is}} G \text{ except } G'_k \leftarrow -G_k$$

$$\rho' \underline{\text{is}} \rho$$

$$\beta' \underline{\text{is}} \beta - G'_k \times (\rho_k - 1)$$

δ' is computed from (*).

(c) *Drop (\dagger)*: $Q \dagger A$

$$G' \underline{\text{is}} G.$$

$$\rho' \underline{\text{is}} \rho - |Q|$$

$$\beta' \underline{\text{is}} \beta + \sum_{k=1}^{\nu} (Q_k > 0) \times Q_k \times G_k$$

δ' is computed from (*).

(d) *Take (\dagger)*: $Q \dagger A$

$$G' \underline{\text{is}} G$$

$$\rho' \underline{\text{is}} |Q|$$

$$\beta' \underline{\text{is}} \beta + \sum_{k=1}^{\nu} (Q_k < 0) \times (\rho_k + Q_k) \times G_k$$

δ' is computed from (*).

(e) *Subscription*:

$$X[M_1; M_2; \dots; M_\nu] \underline{\text{is}}$$

$$(Q_1 \ Q_2 \ \dots \ Q_\nu) \dagger (P_1 \ P_2 \ \dots \ P_\nu) \dagger X$$

(\dagger)

where A and B are delivered in ravel order and \sim refers to the inner product where operands are taken in ravel order. To make specific the order of delivery it is convenient to write

$$C \leftarrow (\phi A) \vee \{1 \ 2 \ \dots \ \underline{\underline{\kappa-1}} \ \underline{\underline{1}} \ \underline{\underline{2}} \ \dots \ \underline{\underline{\delta-1}}; \underline{\underline{\kappa}}, \underline{\underline{\delta}} \sim\} \psi \phi (\delta \ 1 \ 2 \ \dots \ \overline{\delta-1}) \phi B$$

This notation serves to indicate the effect of operations (a) through (f) on ladders delivering inner products.

Thus consider the example:

$$C \leftarrow (3 \ 2 \ 4 \ 1) \phi A \vee \psi B$$

where A and B are each of rank 3. The expression can be transformed into:

$$C \leftarrow (3 \ 2 \ 4 \ 1) \phi (\phi A) \vee \{1, 2, \underline{\underline{1}}, \underline{\underline{2}}, ; 3, \underline{\underline{3}}, \sim\} \psi \phi (3 \ 1 \ 2) \phi B$$

$$C \leftarrow (\phi A) \vee \{(3 \ 2 \ 4 \ 1) \phi 1, 2, \underline{\underline{1}}, \underline{\underline{2}}; 3, \underline{\underline{3}}, \sim\} \psi \phi (3 \ 1 \ 2) \phi B$$

$$C \leftarrow (\phi A) \vee \{\underline{\underline{2}}, \underline{\underline{2}}, \underline{\underline{1}}, \underline{\underline{1}}; 3, \underline{\underline{3}}, \sim\} \psi \phi (3 \ 1 \ 2) \phi B$$

$$C \leftarrow ((2 \ 1 \ 3) \phi \phi A) \vee \langle \underline{\underline{1}} / \underline{\underline{1}} \ \underline{\underline{2}} / \underline{\underline{2}}; 3 \ \underline{\underline{3}} \sim \rangle \psi (2 \ 1 \ 3) \phi \phi (3 \ 1 \ 2) \phi B$$

where / will indicate where ladder splitting is required on the transposed arrays. Finally, if A' is $\underline{\underline{2}} \ \underline{\underline{1}} \ \underline{\underline{3}} \ \phi \phi A$ and B' is $\phi \ \underline{\underline{3}} \ \underline{\underline{2}} \ \underline{\underline{1}} \ \phi B$,

$$C \leftarrow A' \vee \langle \underline{\underline{1}} / \underline{\underline{1}} \ \underline{\underline{2}} / \underline{\underline{2}}; 3 \ \underline{\underline{3}} \sim \rangle \psi B'$$

which has the ladder structure:

Ladder Splitting

As the example in the previous section shows, a ladder may be split into several disjoint ladders, each of which has its own data and control code.

Let the rank of the array be n and let its ladder be split into n sections with associated G 's:

$$G_1, \dots, G_{r_1}; G_{r_1+1}, \dots, G_{r_2}; G_{r_2+1}, \dots, G_{r_n}$$

with $0 = r_0 < r_1 < r_2 < \dots < r_n = n$. Then the δ_k are:

for $i \leftarrow 0$ step 1 until $n-1$ do

for $k \leftarrow r_{i+1}$ step -1 until r_i+1 do

$$\delta_k \leftarrow G_k$$

for $j \leftarrow k+1$ step 1 until r_{i+1} do

$$\delta_k \leftarrow \delta_k - G_j \times (\rho_j - 1)$$

and for each section:

$$\beta_r \text{ is } \prod_{r-1}^{\sim} \quad r \text{ is } 2, \dots, n$$

For example, if G is -5 1 20 and ρ is 4 5 3, the possible ladder splits with associated δ_k are:

Some Basic Operator Transformations

Each of the operations \mathcal{Q} , ϕ , \uparrow , \downarrow , $[v]$ can be absorbed into the ladder of an array in a large class of expressions, i.e. they lead to a recomputation of G , δ , and β for some array and may create ladder splits. We now give the transformations that move these operations to the leaves of the expression tree, where they then modify ladders. We define a class of expressions ξ to which we apply these transformations.

In what follows C denotes an array of rank 1 of constants and X denotes an identifier of an array. τ denotes any of the five operations above. $[k]$ refers to a subscript, each component of which is a constant, blank, or of the form $a \pm ib$, where a and b are integers and $b \geq 0$. Then:

- (1) $X \in \xi$
- (2) If ψ is any monadic and v any dyadic scalar operation, $\psi\xi \in \xi$ and $\xi v \xi \in \xi$
- (3) $v/[k]\xi \in \xi$
- (4) $\xi \circ v \xi \in \xi$
- (5) $\xi v_1 \cdot v_2 \xi \in \xi$
- (6) $\tau\xi \in \xi$
- (7) The mixed monadic functions: $\imath\xi \in \xi$
 $\xi \in \xi$
 $\rho\xi \in \xi$
- (8) The mixed dyadic functions: $C\rho\xi \in \xi$
 $\xi, [\mu]\xi \in \xi$
 $\xi \in \xi \in \xi$
 $\xi/[k]\xi$
 $\xi \setminus [k]\xi$
 $\xi \imath \xi \in \xi$

$$5.1 \quad C \circ A \circ \langle u \rangle B \Leftrightarrow (h \circ A) \circ \langle q \rangle g \circ B$$

where: q is $u[\Delta C]$

h is $\Delta(\kappa \geq \Delta C) / \Delta C$

g is $\Delta(\kappa < \Delta C) / \Delta C$

Example: $6 \ 2 \ 4 \ 3 \ 5 \ 1 \ \circ \ A \circ \cdot \nu B$, κ is 3 and δ is 3

becomes: $(3 \ 1 \ 2 \ \circ \ A) \circ \langle 101010 \rangle 2 \ 3 \ 1 \ \circ \ B$

$$5.2 \quad C \uparrow A \circ \cdot \nu B \Leftrightarrow ((\kappa \uparrow C) \uparrow A) \circ \cdot \nu ((-\delta) \uparrow C) \uparrow B$$

$$5.3 \quad C \downarrow A \circ \cdot \nu B \Leftrightarrow ((\kappa \downarrow C) \downarrow A) \circ \cdot \nu ((-\delta) \downarrow C) \downarrow B$$

$$5.4 \quad \phi[k] A \circ \cdot \nu B \Leftrightarrow \text{if } k \leq \kappa \text{ then } (\phi[k] A) \circ \cdot \nu B \text{ else } A \circ \cdot \phi[k - \kappa] B$$

$$5.5 \quad (A \circ \cdot \nu B)[\mu] \Leftrightarrow (A[\kappa \uparrow \mu]) \circ \cdot \nu B[-\delta \uparrow \mu]$$

6. Transformations on inner product.

$A \nu \langle u \rangle \psi B$ is the notation for inner product. $\langle u \rangle$ has the form

$$\underbrace{0 \dots 0}_{\kappa-1} \underbrace{1 \dots 1}_{\delta-1}; 01$$

or $v; w$

$$6.1 \quad C \circ A \nu \langle u \rangle \psi B \Leftrightarrow (h \circ A) \nu \langle q \rangle \psi g \circ B$$

where: q is $v[\Delta C]; w$

h is $\Delta((-1 + \kappa) \geq \Delta C) / \Delta C, \kappa$

g is $((\Delta(\kappa < \Delta C) / \Delta C), \delta)[\delta, 1 - 1 + \delta]$

$$6.2 \quad C \uparrow A \nu \cdot \psi B \Leftrightarrow ((((-1 + \kappa) \uparrow C), (\rho A)[\kappa]) \uparrow \phi A) \circ \nu \langle u \rangle \psi (((-\delta - 1) \uparrow C), (\rho B)[1]) \uparrow \phi(\delta, 1 - \delta - 1) \phi B$$

$$6.3 \quad C \downarrow A \nu \cdot \psi B \Leftrightarrow ((((-1 + \kappa) \downarrow C), (\rho A)[\kappa]) \downarrow \phi A) \circ \nu \langle u \rangle \psi (((-\delta - 1) \downarrow C), (\rho B)[1]) \downarrow \phi(\delta, 1 - \delta - 1) \phi B$$

$$6.4 \quad \phi[k] A \nu \cdot \psi B \Leftrightarrow \text{if } k \leq \kappa - 1 \text{ then } (\phi[k] \phi A) \nu \langle u \rangle \psi \phi(\delta, 1 - \delta - 1) \phi B$$

else $(\phi A) \nu \langle u \rangle \psi \phi[k - \kappa + 1] \phi(\delta, 1 - \delta - 1) \phi B$

10. Transformations on ϵ . $A \in B$

10.1 $\phi[k]A \in B \iff (\phi[k]A) \in B$

10.2 $C \oslash A \in B \iff (C \oslash A) \in B$

10.3 $C \uparrow A \in B \iff (C \uparrow A) \in B$

10.4 $C \downarrow A \in B \iff (C \downarrow A) \in B$

10.5 $(A \in B)[v] \iff (A[v]) \in B$

11. Transformations on compression. $A/[k]B$

11.1 $\phi[j]A/[k]B \iff \text{if } j \neq k \text{ then } A/[k]\phi[j]B \text{ else } (\phi A)/[k]\phi[k]B$

11.2 $C \oslash A/[k]B \iff A/[C[k]]C \oslash B$

11.3 $C \uparrow A/[k]B \iff (C[k] \uparrow A)/[k]C \uparrow B$

11.4 $C \downarrow A/[k]B \iff (C[k] \downarrow A)/[k]C \downarrow B$

11.5 $(A/[k]B)[v] \iff (A[v_k])/[k](B[v])$

where $[v_k]$ is the k th component of v expressed as a subscript, i.e. as $[]$ or $[C]$ or $[\alpha \pm i\beta]$.

12. Transformations on expansion. $A \backslash [k] B$

These are identical to the transformations on compression, i.e. merely replace "/" by "\" in 11.1 to 11.5.

13. Transformations on index of. $A_1 B$

13.1 $\phi[k]A_1 B \iff A_1(\phi[k]B)$

13.2 $C \oslash A_1 B \iff A_1(C \oslash B)$

13.3 $C \uparrow A_1 B \iff A_1 C \uparrow B$

13.4 $C \downarrow A_1 B \iff A_1 C \downarrow B$

13.5 $(A_1 B)[v] \iff A_1 B[v]$

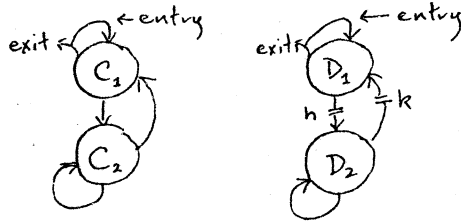
Ladder Networks II

As an expression is parsed, ladder networks are created. In the case of expressions of the form $(\xi_1) \nu \xi_2$, where ν is a dyadic operator, the two independent networks for ξ_1 and ξ_2 may be fused into one network. This is particularly important when ν is a dyadic scalar operator. Let us consider two examples to gain insight into the general fusing algorithm.

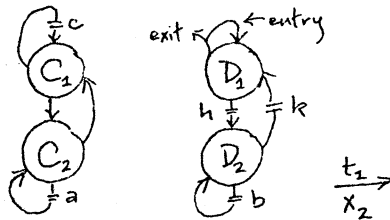
Example 1. $+/(A \times B) + C \div D$

Assume all arrays are of rank 2.

Step 1.

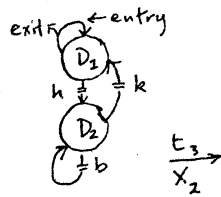


Step 2. The operation \div fuses the two ladders into "one" ladder structure.



- a) $t_1 \leftarrow \tilde{\Pi}; \hookrightarrow X_1$
- b) $\hookrightarrow X_1; t_1 \leftarrow t_1 \div \tilde{\Pi}; \hookrightarrow X_2;$

The result is t_1 linked by co-routine register X_2 . X_1 is c .



- d) $t_2 \leftarrow \tilde{\Pi}; \hookrightarrow X_3$
- e) $\hookrightarrow X_3; t_2 \leftarrow t_2 \times \tilde{\Pi}; \hookrightarrow X_4$
- h) $t_3 \leftarrow 0$
- k) $\hookrightarrow X_2$

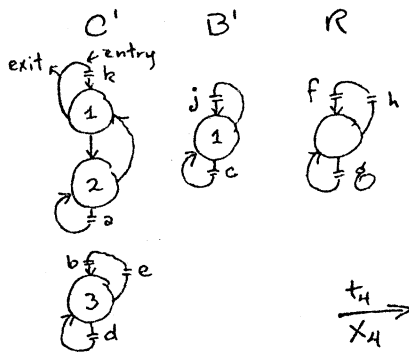
X_{1_0} is c; X_{3_0} is f; X_{4_0} is g.

Example 2. $R_1A+B/[2]C$ where ρC is 3

Under transformations it becomes $R_1(A[2]+B)/[2]A+C$.

$A[2]+B$ becomes B' and $A+C$ becomes C' when their G , ρ , δ , and β values are changed in splice k .

$/[2]$ induces ladder splitting of C' :

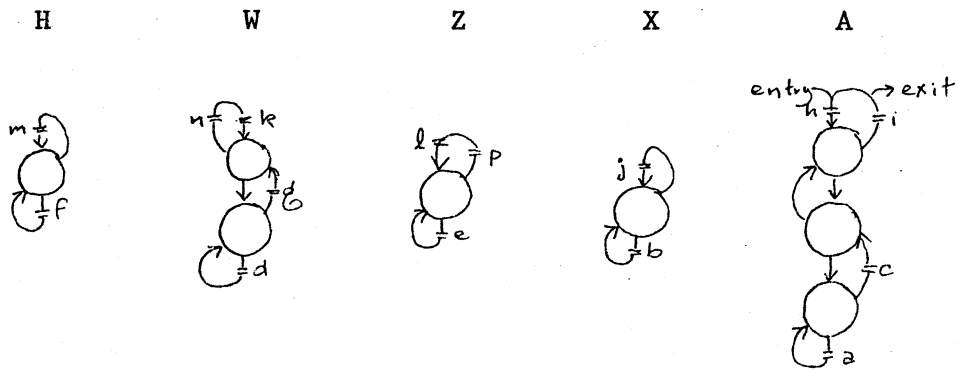


- a) $t_1 \leftarrow \tilde{\Pi}; \hookrightarrow X_1$
- c) if $\tilde{\Pi} = 1$ then $\hookrightarrow X_2;$
 $\hookrightarrow X_1$
- b) $\beta \leftarrow t_1$
- d) $t_3 \leftarrow \tilde{\Pi}; \hookrightarrow X_3$
- e) $\hookrightarrow X_2$
- f) $t_4 \leftarrow 1 + \rho R$
- g) if $t_3 = \tilde{\Pi}$ then $t_4 \leftarrow i_1;$
goto h
- h) $\hookrightarrow X_4; \hookrightarrow X_3$

The result is delivered from h in t_4 and linked by co-routine register X_4 .

X_{1_0} is j. X_{2_0} is b. X_{3_0} is f.

The ladder structure is:



a) $t_1 \leftarrow \tilde{\Pi}$; $\hookrightarrow X_1$

b) $\Pi \leftarrow t_1$; $\hookrightarrow X_1$

c) SRJ F; $\hookrightarrow X_2$ (SRJ: sub-routine jump)

d) $\hookrightarrow X_3$; $t_3 \leftarrow \tilde{\Pi} + t_2$; $\hookrightarrow X_4$

e) $t_2 \leftarrow \tilde{\Pi}$; $\hookrightarrow X_3$

f) $\Pi \leftarrow t_3$; $\hookrightarrow X_4$

g) $\hookrightarrow X_2$

h) entry

i) exit

X_{10} is j

X_{20} is k

X_{30} is l

X_{40} is m

Example 2. $H \leftarrow (F A) \circ . + W$

but now F has the header $\nabla Z(,1) \leftarrow F X(,1)$. X may have any rank and that of Z is computed from X's rank. The ladder structure is:



since all δ_j for H' are no longer 1.

The introduction of this new data structure implies that we have useful answers to the following questions:

- a) How do the standard APL functions extend when one or two operands are ragged arrays?
- b) How are ragged arrays processed by ladder networks?
- c) What programming tasks are made simpler (more transparent) or more economical or both by the presence of ragged arrays?

As to (a), a ragged array is an array of vectors. Consequently APL functions must extend so as to facilitate the control of array-arranged vector-vector operations. As a first example, consider some possible extensions of dyadic ι .

Let A and B be ra 's and $C \leftarrow A \iota B$.

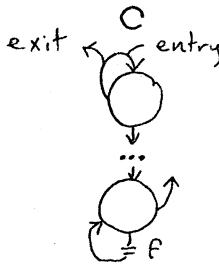
- 1) $\rho \alpha A$ is 2 2 and $\rho \alpha B$ is 3 3 and $\rho \alpha C$ is 2 2 3 3 where $C[I;J;K;L;]$ is $A[I;J;] \iota B[K;L;]$
- 2) $\rho \alpha A$ is 3 1 and $\rho \alpha B$ is 3 3 and $\rho \alpha C$ is 3 3 and $C[I;J;]$ is $A[I;] \iota B[I;J;]$
- 3) $\rho \alpha A$ is $\rho \alpha B$ is 3 3 and $\rho \alpha C$ is 3 3 and $C[I;J;]$ is $A[I;J;] \iota B[I;J;]$

Now ι is basically defined as a vector-vector function. Hence, with that understood, we write

- 1) $\underline{\text{is}} A \circ \iota B$ (Note the use of the outer product notation for composite functions.)
- 2) $\underline{\text{is}} (\alpha B) \alpha \tau 1 2 1 3 \text{ } \text{Q} \iota A \circ \iota B$ since Q is defined only for regular APL arrays.
- 3) $\underline{\text{is}} A \iota B$

As a second example consider Δ . If A is ra , B is ΔA means

Consider the ladder for $C[D]$ where each rag is an integer or 10 or a vector of integers:



$$a) \delta \leftarrow 1$$

$$\rho \leftarrow \text{if } t_4 = 0 \text{ then } t_2 \text{ else } t_4$$

$$\beta \leftarrow \text{if } t_4 = 0 \text{ then } 1 \text{ else } t_3$$

$$b) t_5 \leftarrow \tilde{\Pi} + t_1 - 1$$

$$\hookrightarrow Z$$

$$c) \hookrightarrow Y$$

$$f) t_1 \leftarrow \mu_1(\Pi); \hookrightarrow X$$

$$g) t_3 \leftarrow \mu_1(\Pi); t_4 \leftarrow \mu_2(\Pi)$$

$$\hookrightarrow Y$$

$$\hookrightarrow X$$

The result is t_5 linked by Z . X_0 is e . Y_0 is a .

As to (c), what programming tasks do ragged arrays help with, consider a piece of text that is represented as a ragged array C , the lines being rags of character vectors.

- 1) Insert the ragged array D after the k th rag of C :

$$Q \leftarrow \perp C$$

$$C \leftarrow \tau(k \uparrow Q), (\perp D), k \uparrow Q$$

- 2) Suppose each rag of C consists of character string words separated by single spaces. Let D be a ra each rag of which is a word. Create E , the index of D , i.e. an integer ra such that $E[k;]$ is the set of line numbers of C whose lines contain one or more occurrences of the word $D[k;]$.

Machine Code for APL

Ladders provide some insight into a model for machine code for an APL machine. A line of APL code, as we have seen, produces a ladder network with its associated scalar code. The scalar code is very much like ordinary machine code, except that the required operand address space is generally small.

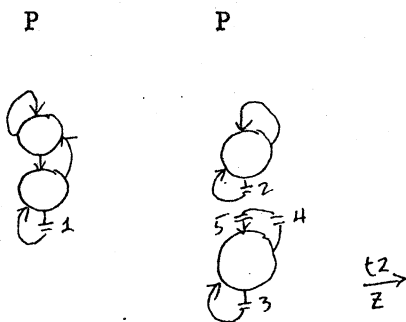
An important feature of ladder code is that ladder networks are easily constructed for expressions that are cumbersome to express in standard APL. For example, let P be an array. ρP is $N \times M$. Each row represents a point in Euclidean M space. We wish to construct D , the matrix of the square of the distances between the points defined by P . Then:

$$D \leftarrow + / ((1 \ 2 \ 3 \ 2) \circ P \circ . - P) * 2$$

$$D \leftarrow + / ((1 \ 2 \ 2 \ 3) \circ P \circ . - \circ P) * 2$$

$$D \leftarrow \tilde{+} / (\circ P) \circ . \{ \underline{1} \ 2 \ \underline{1} \ \underline{2}; \ 2 = \underline{1} \} - \circ P * 2$$

A ladder network is:



$$1) \ t1 \leftarrow \tilde{\Pi}; \ \leftarrow U$$

$$2) \ t2 \leftarrow 0; \ t3 \leftarrow \tilde{\Pi}; \ \leftarrow V; \ \leftarrow U$$

$$3) \ t2 \leftarrow (t1 - \tilde{\Pi}) * 2 + t2$$

$$4) \ \leftarrow Z; \ \leftarrow V$$

$$5) \ \beta \leftarrow t3$$

As a second example, suppose one wishes to prepare the graph of a