TAGS: Trains, Agendas, and Gerunds

Roger K.W. Hui

Morgan Stanley

Kenneth E. Iverson

Iverson Software

Trains, agendas, and gerunds have been available in J for some time, and do not conflict with most APL systems. In particular, since there are no noun trains, the trains in J do not conflict with the strands of APL2.

This paper reviews the definitions and uses of trains, agendas, and gerunds, and presents some new extensions that enhance their utility.

TRAINS

The first example of a train was provided by the *fork*, defined by Iverson and McDonnell [1] as a formalization of the informal use in mathematics of expressions such as `f+g` and `f-g` to denote the sum and the difference of functions. For example:

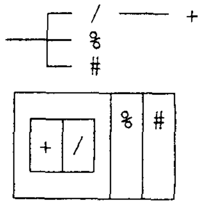
```
   mean=. +/ % #        Sum div by no. of items
   mean a=. 1 2 3 4 5
3
   norm=. ] - mean      Right arg less mean
   norm a
_2 _1 0 1 2
   (] - +/ % #) a
_2 _1 0 1 2
```

The phrase `] - +/ % #` is an example of a train of more than three elements. The definition of a *hook* as a train of two elements provides meanings for trains of even length as well as odd. For example:

```
   pr=. + %             Left plus reciprocal of right
   2 pr 10
2.1
   pr/a                 Continued fraction
1.43312
   pr/1 1 1 1 1         Approx golden mean
1.625
   NORM=. - +/ % #      Train of four items
   (norm = NORM) a      Test functions for
1 1 1 1 1                  equality
```

To make the functions more readable we will set the function display to show both the tree and boxed representations of functions. Thus:

```
   9!:3 ] 4 2
   mean
```



The utility of trains is enhanced by the *cap* (denoted by `[:` and defined on p. 70 of Iverson [2]), which causes the verb that follows it in a train to be applied monadically. For example, the squares of a normalized list and the standard deviation may be defined by single trains as follows:

```
   sqn=. [: *: ] - +/ % #
    std=. [:%:[:(+/%#)[:*:]-+/%#
   sqn 8 10 12
4 0 4
   std 8 10 12
1.63299
```

GERUNDS

The *tie* conjunction (`` ` ``) applied to verbs produces a noun that is called a *gerund* (because it carries the force of a verb). For example:

```
   g=. +`*
   g/a
47

   1+2*3+4*5
47
```


APL 94 - 9/94 Antwerp Belgium
© 1994 ACM 0-89791-675-1/94/0009..$3.50

```
   x=. 10 [ c=. 3 1 4 2
    g/3,x,1,x,4,x,2
2413
   }. , x,.c
3 10 1 10 4 10 2
   +`*/ }. , x,.c
2413
```

The last expression above is Horner's efficient evaluation of a polynomial with coefficients `c`.

The adverb `\` applied to a gerund yields the equivalent train. For example:

```
    ger=.  -`(+/)`%`#
   ger\ a                    Equivalent to norm
_2 _1 0 1 2
   (1 2 3 { ger)\ a        Equivalent to mean
3
```

AGENDA

The *agenda* conjunction (`@.`) selects from its gerund left argument the item indexed by the result of its verb right argument. For example:

```
   h=.(=<.) * 1: + (=|)  Constant fn 1:
   h 4.5 _4 4
0 1 2
   q=. >.`*:`%: @. h
   q 4.5     Ceiling of non-integer
5
   q _4      Square of negative integer
16
   q 4       Square root of positive integer
2
```

NEW EXTENSIONS

Gerunds are merely boxed lists, and may be formed by means other than the tie. For example:

```
   g0=. +`% [ g1=. '+';'%'
   g0\                       g1\
  ┌ +                      ┌ +
──┴ %                    ──┴ %

┌─┬─┐                    ┌─┬─┐
│+│%│                    │+│%│
└─┴─┘                    └─┴─┘

   (5 g0\ 4), (5 g1\ 4)
5.25 5.25
    ]g2=. ;:'&+&.+&:^.@'
┌─┬─┬──┬─┬──┬──┬─┐
│&│+│&.│+│&:│^.│@│
└─┴─┴──┴─┴──┴──┴─┘
```

Repeated boxing in an argument to `\` will indicate parenthesization. For this purpose we will define a recursive indexing function such that the boxing of indices is reflected in the boxing of the result:

```
   from=. ($:&.> <)`from1 @. ifopen
     ifopen=. (-:>) @ {. @ [
     from1=. >@{ ` { @. (1&<@#@[)
```

For example:

```
   k =.(0;_2;_1;(1 3 5;3;5);4;5;6)
   k
┌─┬──┬──┬───────────┬─┬─┬─┐
│0│_2│_1│┌─────┬─┬─┐│4│5│6│
│ │  │  ││1 3 5│3│5││ │ │ │
│ │  │  │└─────┴─┴─┘│ │ │ │
└─┴──┴──┴───────────┴─┴─┴─┘

   k from g2
┌─┬──┬─┬────────────────┬──┬──┬─┐
│&│^.│@│┌──────────┬─┬──┐│&:│^.│@│
│ │  │ ││┌─┬─┬──┐  │+│^.││  │  │ │
│ │  │ │││+│+│^.│  │ │  ││  │  │ │
│ │  │ ││└─┴─┴──┘  │ │  ││  │  │ │
│ │  │ │└──────────┴─┴──┘│  │  │ │
└─┴──┴─┴────────────────┴──┴──┴─┘

   9!:3] 4 2 5  Display: Linear/Boxed/Tree
   q=. (k from g2)\ & ^. (@ ((+
           + ^.) + ^.) (&: ^. @))
   q
┌─┬──┬──────────────────────────────────┐
│&│^.│┌─┬────────────────────┬─────────┐│
│ │  ││@│┌──────────┬─┬──┐   │┌──┬──┬─┐││
│ │  ││ ││┌─┬─┬──┐  │+│^.│   ││&:│^.│@│││
│ │  ││ │││+│+│^.│  │ │  │   │└──┴──┴─┘││
│ │  ││ ││└─┴─┴──┘  │ │  │   │         ││
│ │  ││ │└──────────┴─┴──┘   │         ││
│ │  │└─┴────────────────────┴─────────┘│
└─┴──┴──────────────────────────────────┘

  ┌ &
  ├ ^.
  │    ┌ @
──┤    │       ┌ +
  │    │    ┌──┼ +
  │    │    │  └ ^.
  └────┼────┼ +
       │    └ ^.
       │
       │    ┌ &:
       └────┼ ^.
            └ @
```

Comparison of the two displays of `q` shows the relation between boxing and parenthesization. The conjunction `q` may be used as follows:

```
   2 ^ q ! 3
0.727302
```

The agenda is redefined to make use of these results: `g @. h` is equivalent to `(i from g)\`, where `i` is the result of the function `h`.

Page 70 of Iverson [2] listed twelve basic trains (bidents and tridents), two producing verbs, four producing adverbs, and six producing conjunctions. This list is now extended to the twelve bidents and twenty-five tridents listed below. The four columns show the case, the class of result, the definition, and an informal name, using the letters N, V, A, and C to indicate nouns, verbs, adverbs, and conjunctions. An asterisk marks cases that were previously meaningless:

`N0 A1`	verb	as in APL\360	"apply"
`N0 C1`	adv	`N0 C1 x`	"with"
`V0 N1`	noun	as in APL\360	"apply"
`V0 V1`	verb	hook	"hook"
`V0 A1`	verb	as in APL\360	"apply"
`V0 C1`	adv	`V0 C1 x`	"with"
`A0 V1`	*adv	`(x A0) V1`	
`A0 A1`	adv	`(x A0) A1`	"compose"
`A0 C1`	adv	`(x A0) C1 x`	"both"
`C0 N1`	adv	`x C0 N1`	"with"
`C0 V1`	adv	`x C0 V1`	"with"
`C0 A1`	conj	`(x C0 y) A1`	"atop"
`N0 V1 N2`	noun	as in APL\360	"apply"
`V0 V1 V2`	verb	fork	"fork"
`V0 V1 C2`	*conj	`V0 V1 (x C2 y)`	
`A0 V1 V2`	*adv	`(x A0) V1 V2`	
`C0 V1 V2`	*conj	`(x C0 y) V1 V2`	
`C0 V1 C2`	conj	`(x C0 y) V1 (x C2 y)`	"fork"
`A0 A1 V2`	*conj	`(x A0) (y A1) V2`	
`A0 A1 A2`	adv	`((x A0) A1) A2`	"compose"
`C0 A1 A2`	conj	`((x C0 y) A1) A2`	
`N0 C1 N2`	verb	as in APL\360	"apply"
`N0 C1 V2`	verb	as in APL\360	"apply"
`N0 C1 A2`	*adv	`N0 C1 (x A2)`	
`N0 C1 C2`	*conj	`N0 C1 (x C2 y)`	
`V0 C1 N2`	verb	as in APL\360	"apply"
`V0 C1 V2`	verb	as in APL\360	"apply"
`V0 C1 A2`	*adv	`V0 C1 (x A2)`	
`V0 C1 C2`	*conj	`V0 C1 (x C2 y)`	
`A0 C1 N2`	*adv	`(x A0) C1 N2`	
`A0 C1 V2`	*adv	`(x A0) C1 V2`	
`A0 C1 A2`	conj	`(x A0) C1 (y A2)`	"fork"
`A0 C1 C2`	conj	`(x A0) C1 (x C2 y)`	
`C0 C1 N2`	*conj	`(x C0 y) C1 N2`	
`C0 C1 V2`	*conj	`(x C0 y) C1 V2`	
`C0 C1 A2`	conj	`(x C0 y) C1 (y A2)`	
`C0 C1 C2`	conj	`(x C0 y) C1 (x C2 y)`	"fork"

In order to distinguish a noun such as `'*'` from the function `*` in a gerund, its *atomic representation* must be used. The atomic representation of a noun is given by the function

```
ar=. [: < (,'0')"_ ; ] .
```

The new trains substantially simplify the results provided by the translators from explicit to tacit definition presented in Hui [3].

EXAMPLES

We conclude with two further examples of the use of gerunds, in *recursion* and *amendment*.

Recursion. In the Tower of Hanoi puzzle, a set of **n** discs (each of a different size) is to be moved from post A to post B using a third post C, under the restriction that a larger disc is never to be placed on top of a smaller.

Recursion using the gerund and agenda provides a simple definition of the sequence of moves in Hanoi:

```
   h=. b`(p,.q,.r)@.c
      c=. 1: < [
      b=. 2&,@[ $ ]
         p=. <:@[ h 1: A. ]
         q=. 1: h ]
         r=. <:@[ h 5: A. ]
   3 h x=. 'ABC'
AABACCA
BCCBABB
   0 1 2 3 4 <@h"0 1 x
┌┬─┬───┬───────┬───────────────┐
││A│AAC│AABACCA│AACABBAACCBCAAC│
││B│CBB│BCCBABB│CBBCACCBBAABCBB│
└┴─┴───┴───────┴───────────────┘
```

In the Josephus problem analyzed in Graham et al. [4], items in a circle are eliminated at a fixed interval `i` until only `i-1` remain; the indices of the survivors are to be determined as a function of the interval and the original list of items. This function may be expressed as a recursion in which the odd and even cases are treated differently:

```
   J=. 0:`even`odd @. c
   even=. +:@>:@J@<:@-:
   odd=. +:@J@-:@<:
   c=. * * >:@(2&|)
   J"0 b=. i. 20
0 0 2 0 2 4 6 0 2 4 6 8 10 12 14 0 2 4 6 8
   <;.1 J"0 b
┌─┬───┬───────┬───────────────────┬─────
│0│0 2│0 2 4 6│0 2 4 6 8 10 12 14│0 2 4
└─┴───┴───────┴───────────────────┴─────
```

Amendment. If the *amend* adverb `}` is applied to a numeric argument `m`, then `x m} y` yields `y` *amended* by `x` in the positions specified by the indices in `m`. For example:

```
   '#*' 3 0} 'abcdefg'
*bc#efg
```

If `m` is a gerund the possibilities are much greater: its middle element determines the numeric argument to the adverb `}`, and the others modify the arguments `x` and `y`.

For example, the following functions `E1`, `E2`, and `E3` are the so-called *elementary* linear operations on a matrix, interchanging two rows of a matrix, multiplying a row by a constant, and adding a multiple of one row to another:

```
   E1=. <@] C. [
   E2=. f`g`[}
   E3=. F`g`[}
     f=. {:@]*{.@] { [
     F=. [:+/(1:,{:@])*(}:@] { [)
     g=. {.@] { i.@$@[
   M=. i. 3 3
   a1=. M E1 1 2
   a2=. M E2 1 10
   a3=. M E3 1 2 10
```

```
   M;a1;a2;a3
┌─────┬─────┬────────┬────────┐
│0 1 2│0 1 2│ 0  1  2│ 0  1  2│
│3 4 5│6 7 8│30 40 50│63 74 85│
│6 7 8│3 4 5│ 6  7  8│ 6  7  8│
└─────┴─────┴────────┴────────┘
```

It should be noted that the function `E1` uses a permutation expressed as a cycle rather than the amend adverb. However, it could also be expressed as an amendment using a gerund.

REFERENCES

1. McDonnell E.E. and K.E. Iverson, Phrasal Forms, *APL89 Conference Proceedings*, APL Quote-Quad Vol 19 Number 4.

2. Iverson, K.E. *J Introduction and Dictionary Version 7*, Iverson Software.

3. Hui, Roger K.W. et al, Tacit Definition, *APL91 Conference Proceedings*, APL Quote-Quad Vol 21 Number 4.

4. Graham, Patashnik, and Knuth, *Concrete Mathematics*, Addison-Wesley, 1989.