Appendix: Defined Functions

Original Wagner-Fischer Algorithm

```
    ∇
[0]   z←a wfa b;i;j;m1;m2;m3;n1;n2;D
[1]   ⍝ Global Inputs: alphabet, ins_del, sub, ⎕IO←0
[2]   ⍝ Inputs: strings a and b made up from alphabet
[3]   ⍝ Outputs: distance between a and b
[4]   ⍝ Usage:
[5]   ⍝ 'abbdbbabbcbbabbdbbabbcbb' wfa 'ebbbabbbcbbbabbb'
[6]
[7]    n1←⍴a←alphabet⍳a
[8]    n2←⍴b←alphabet⍳b
[9]    a←¯1,a
[10]   b←¯1,b
[11]   D←((n1+1),n2+1)⍴0
[12]
[13]   i←0
[14]  loop1:→(n1<i←i+1)/end1
[15]   D[i;0]←D[i-1;0]+ins_del[a[i]]
[16]   →loop1
[17]  end1:
[18]
[19]   j←0
[20]  loop2:→(n2<j←j+1)/end2
[21]   D[0;j]←D[0;j-1]+ins_del[b[j]]
[22]   →loop2
[23]  end2:
[24]
[25]   i←0
[26]  loop3:→(n1<i←i+1)/end3
[27]   j←0
[28]  loop4:→(n2<j←j+1)/loop3
[29]   m1←D[i-1;j-1]+sub[a[i];b[j]]
[30]   m2←D[i-1;j]+ins_del[a[i]]
[31]   m3←D[i;j-1]+ins_del[b[j]]
[32]   D[i;j]←m1⌊m2⌊m3
[33]   →loop4
[34]
[35]  end3:
[36]   z←D[n1;n2]
    ∇
```

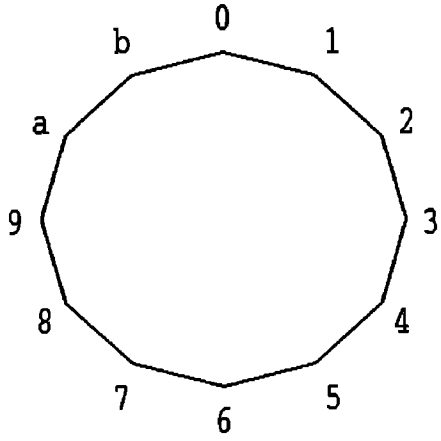
Parallel Wagner-Fischer Algorithm

```
    ∇
[0]   z←a pwfa b;i;j;k;m1;m2;m3;n1;n2;n3;D;from
[1]   ⍝ Global Inputs: alphabet, ins_del, sub, ⎕IO←0
[2]   ⍝ Inputs: strings a and b made up from alphabet
[3]   ⍝ Outputs: distance between a and b
[4]   ⍝ Usage:
[5]   ⍝ 'abbdbbabbcbbabbdbbabbcbb' pwfa 'ebbbabbbcbbbabbb'
[6]
[7]    n1←⍴a←alphabet⍳a
[8]    n2←⍴b←alphabet⍳b
[9]    a←¯1,a
[10]   b←¯1,b
[11]   D←((n1+1),n2+1)⍴0
[12]
[13]   i←0
[14]  loop1:→(n1<i←i+1)/end1
[15]   D[i;0]←D[i-1;0]+ins_del[a[i]]
[16]   →loop1
[17]  end1:
[18]
[19]   j←0
[20]  loop2:→(n2<j←j+1)/end2
[21]   D[0;j]←D[0;j-1]+ins_del[b[j]]
[22]   →loop2
[23]  end2:
[24]
[25]   k←0
[26]   n3←n1+n2-1
[27]  loop3:→(n3<k←k+1)/end3
[28]   i←from←⍳1+(k⌊n1)-(from←1⌈1+k-n2)
[29]   j←1+k-i
[30]   m1←D[(i-1),¨(k-1)]+sub[a[i],¨b[j]]
[31]   m2←D[(i-1),¨j]+ins_del[a[i]]
[32]   m3←D[i,¨k-1]+ins_del[b[j]]
[33]   D[i,¨j]←m1⌊m2⌊m3
[34]   →loop3
[35]
[36]  end3:
[37]   z←D[n1;n2]
    ∇
```

Clock Face

—by David Steinbrook
La Selva Beach, California

THE TWELVE NUMBERS on an analog clock face are close to the twelve digits in base-12:



These digits conceal correspondences that first surfaced in APL in 1975, and were extended in J during1998. While the context for this material was originally music, its mappings hold in general.

Just beneath the surface is a relationship between a *set* (a collection of distinct items) and its subsets. In a musical setting, items in a set (scale) are not heard at the same time: only through subsets (triads) do we infer set-contents. A *sound-rule* (non-adjacent in the scale) is a term to refer to this relationship.

In this presentation, we first describe subsets derived in two ways. We then show that these two groups of subsets have the same sub-types. Listings of a few J verbs appear at the end.

Sets

Base-12 sets have size (number of distinct items) and, within size, they have type. We name sets by the number of distinct items they contain: sets of size 2, for example, are a starting point.

The set 0 1 has the same type as the set 1 2 (transposition, adding a constant to each item, does not change the type). Ordering is not a factor: the set 1 0 has the same content as 0 1. Using these and similar criteria, there are six distinct Two-types:

```
0 1
0 2
0 3
0 4
0 5
0 6
```

When we classify base-12 sets by size and by types within size, we see the domain:

Name	Distinct Types
Two	6
Three	12
Four	29
Five	38
Six	50
Seven	38
Eight	29
Nine	12
Ten	6

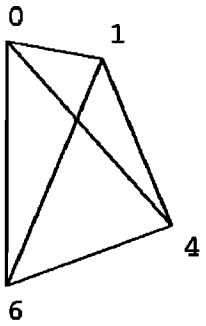
Combinations

The first algorithm we present generates subsets with a combinations technique. For example, to find size-3 subsets in a size-4 set, we generate indices for the subsets:

```
   ] i=. 3 cmb 4
0 1 2
0 1 3
0 2 3
1 2 3
```

and apply them to a size-4 set:

```
   ] t11 =. Tet 11
0 1 4 6
   ] x=. i{t11
0 1 4
0 1 6
0 4 6
1 4 6
```



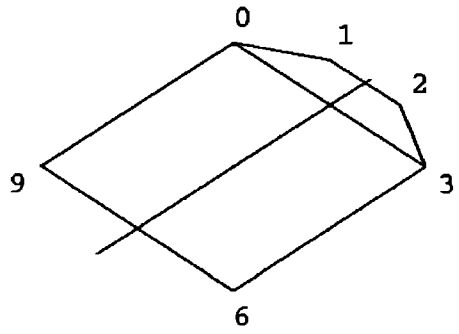
Given Three-type subsets within a Four-type, we scrutinize the Three-types (using index-origin 1):

```
   subtype x
3 5 7 8
```

So far, we speculate that subset Three-types in a Four-type are distinct.

Flex Patterns

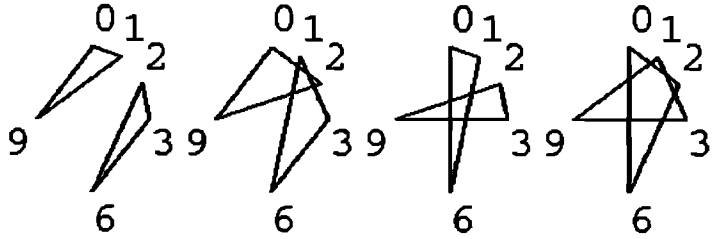
The second algorithm we present generates subsets with *flex* patterns on a symmetrical set (overt bilateral symmetry). When placed on a circle, some sets have an *axis* of symmetry. Consider Six 10, with items `0 1 2 3 6 9`:



Pairs of symmetrical items with the same sum (3 mod 12) are (9 6), (0 3), and (1 2); these are called *dyads*. Four axis-crossings exist for three dyads: `0 0` (shown), `0 1`, `1 0`, and `1 1`:

```
   >flex Six 10
9 0 1     NB. Cross: 0 0
6 3 2
9 0 2     NB. Cross: 0 1
6 3 1
9 3 2     NB. Cross: 1 0
6 0 1
9 3 1     NB. Cross: 1 1
6 0 2
```

Each pattern gives rise to a Three-type and its mirror image.



When we scrutinize these Three-types, we find:

```
   subtype >flex Six 10
3 3 7 7 5 5 8 8
```

Now we are suspicious: combination subsets from Four-type 11 and flex subsets from Six-type 10 correspond to each other by type.

Corresponding Threes

The Four we used here is not symmetrical, while the Six has overt symmetry. When we examine all sets, we find that there are sixteen non-overt Four-types that correspond to the sixteen overt Six-types through subset Three-types:

Four-type	Three-type				Six-type
2	1	2	3	6	1
3	1	3	4	7	12
4	1	4	5	8	20
5	1	5	5	9	25
7	2	6	4	7	43
8	2	7	5	10	47
9	2	5	8	11	39
11	3	5	7	8	10
12	3	5	10	11	32
13	3	4	11	12	41
15	4	5	8	9	11
19	2	3	8	10	28
20	2	4	9	11	35
22	6	7	9	11	49
23	6	8	8	12	50
25	7	10	8	11	42

If we had chosen a bilaterally symmetrical Four-type, of which there are 10, the correspondence is with symmetrical Fives, of which there are also 10. Two of each Three-type occur in each:

Four-type	Three-type				Six-type
1	1	1	2	2	1
6	2	2	3	3	29
10	3	3	4	4	38
14	4	4	5	5	13
16	4	4	11	11	16
18	2	2	7	7	17
21	6	6	8	8	35
24	7	7	9	9	37
27	3	3	11	11	28
28	7	7	11	11	36

Conclusion

Base-12 correspondences open a door to relationships not suspected on the surface, and lead to a new repertoire of options.

Listings

Several verbs written in J appear here. Complete listing of a catalog of base-12 sets is beyond the scope of this presentation, but is available on request. ■

```
cmb=: 4 : 0
NB. size x. combinations of i.y.
NB. R.K.W. Hui
z=.1 0$k=.i.#c=.1,~(y.-x.)$0
for. i.x. do.
 z=.;k,.&.>(-c=.+/\.c){.&.><1+z
end.
)


flex=:3 : 0
(i.#dyads y.) flex y.
:
fl"2&.>x.{dyads y.
)


fl=:3 :0
NB. y. are dyads (2 x n)
r=.agg"1#:i.2^<:{:$y.
r flx y.
)


flx=:4 : 0
>|:&.>(<"1 x.)|."0 1&.><|:y.
)


agg=:[:+/\(+/'')&,
```

---

*A composer since 1956 and APL-literate since 1975, **David Steinbrook** obtained his MFA in Composition from Princeton University in 1965. He taught Music Theory and Composition at Princeton and at Swarthmore College until 1977, then worked for I. P. Sharp Associates, Reuters, NetLabs, Seagate Software, and Motorola. He discovered the first Clock Face correspondence while a visitor at the IBM APL Design Group in Philadelphia (1975–76). He writes music and builds music and other interfaces in J from La Selva Beach, on Monterey Bay, in California. You can reach him at davidst@pacbell.net or at 831-684-1754.*