SYMMETRIES OF THE FIRING SQUAD SYNCHRONIZATION PROBLEM REVEALED IN A NESTED ARRAY

J. Philip Benkard
Staff Programmer, Quality Assurance
International Business Machines Corporation
Dept 17W / Bldg 630
Route 52
Hopewell Junction, NY 12533-0999

ABSTRACT

The firing squad problem of automata theory is described and a simple, nonoptimal solution is given. Examination of output traces shows that the solution has at least one obvious symmetry. If the transition matrix is reshaped and partitioned, the resulting nested array not only reveals and explicates more symmetries but suggests ways in which nested arrays can be used to save space.

INTRODUCTION

A finite state automaton is a device with a finite number each of states, inputs, and outputs. One of the inputs is the current state; one of the outputs is the next state. The other inputs and outputs are external. As time advances in discrete steps, the state of the device at one time is determined entirely by the inputs at the last previous time. The firing squad problem requires the definition of such an automaton, called a soldier, with two external inputs which are the states of his left and right neighbors when arrayed in a firing squad line. The only output is the next state.

From time to time a squad is formed whose size is determined by the prominence of the guest of honor. In view of the inevitable inflation of political values, the possible size of a squad must be considered unlimited. The automata are set to a common initial state, and an officer at one end of the line then gives the command "Synch and Shoot". The soldiers, following the rules of finite state automata and each receiving input only from his two neighbors, exchange enough information to fire simultaneously as soon as possible. Firing is represented by a particular state of the automaton. The optimal solution, which takes ¯2+2×N steps, where N is the length of the squad, is quite complex. A simple solution with only twelve internal states will be shown. For squads of length two or greater it requires no more than ¯2+3×N states.

The problem is nontrivial because of the combination of synchronization and finiteness of the individual automata. It is not hard to design soldiers who will fire in sequence down the line. It is not too much harder to synchronize if the automata can "count off", just like real soldiers. When the soldier at the far end finds out that he is number 576273054, the soldiers can easily count down in sequence and synchronize. But a finite state automaton has a maximum limit to his countability; a squad beyond that limit cannot so synchronize itself.

A much simplified problem will be analyzed first to introduce the method of message transmission. The solution to this simple problem has no symmetries. A solution to the original problem is then presented and analyzed. When the shallow transition matrix, on which the solution is based, is transformed into a depth three array, the symmetries of the internal states of the automata and of the connections between automata are revealed as different forms of reflections.

The shallow matrix, furthermore, is rather sparse. The distribution of significant elements in items and subitems can be used to effect significant storage savings.

A TRIVIAL CASE

Let the requirement be that only the soldier at the opposite end of the line from the officer must fire his weapon. The command to synch and shoot can be thought of as a shoulder tap of the first soldier. The internal states must be defined so that the shoulder tap is transmitted to the next soldier during each time interval. For this case, three internal state suffice. Before the command is given all soldiers are in a common idling state, which is selfperpetuating; if a soldier and his neighbors are in the idling state, the soldier remains idle for the next time period.

The next state is determined by three values, each of which has a range of three. A 3 by 3 by 3 transition matrix is necessary. The values in the matrix are the same three values with one addition, the firing state. The iterative solution halts as soon as one or more soldiers enter that state so that this state is never an input to another soldier.

The states are represented by characters, which can be used to create a suggestive display. This representation was chosen long ago at a time when the possibility of a few 20 by 20 by 20 four byte integer matrices was fraught with hints of WSFULL. The representation turned out to be serendipitous. The characters are in a variable called SFSCH, standing for Simple Firing Squad CHaracters.

```
          SFSCH  ←-→  '.=⎕⍟'
```

The logarithm symbol represents the firing state; the period, the idling state; the equals, the shoulder tap state. The quad remains in the same state regardless of what happens to his neighbors; for that reason, he is sometimes called a blockhead. Before the iterative solution begins, a blockhead is affixed to each end of the line to provide neighbors for the end soldiers. Blockheads will be found to play a role at other than end points in the general solution.

The transition matrix is called SFSM, for Simple Firing Squad Matrix. It is conveniently displayed by lowering its last two axes

```
          ⊂[1 2] SFSM

 ...  .==  ==⎕
 =-⍟  ===  ==⎕
 ...  .-⍟  ⎕⎕-
```

The three axes represent, respectively, a soldier, his left neighbor, and his right neighbor. Thus, the left hand 3 by 3 matrix represents a soldier in state zero (.). Rows stand for values of the left neighbor; columns, the right. For example, the middle rows of the three displayed matrices represent a soldier whose left neighbor is in state one (=); the left columns, a soldier whose right neighbor is in state zero (.).

The character in each position gives the state of the soldier at the next point in time. One (=) will be the state of the soldier at the next time if the soldier and his right neighbor are in state zero (.), and his left neighbor is in state one (=). Thus an equals sign appears in plane zero, row one, and column zero. The minus sign is used at locations which will never be used; that is, for configurations of states which will never occur.

The character representations can be converted to small integers by finding their indices in SFSCH. The numbers for a particular time can then be used as indices into the transition matrix to find the next internal state. For example, at time zero a squad of length three is displayed

```
        □=..□
```

Soldier zero is in state one (=); his left neighbor is in state two (□); his right, in state zero (.). For soldier one, the integers are zero (.), one (=), and zero (.), respectively. Using these values to select from the transition matrix

```
        (⊂¨(1 2 0)(0 1 0))⊃¨⊂SFSM
    .=
```

Soldiers zero and one are in idle and shoulder tap states, respectively, one time interval later. The firing squad at this time would be represented

```
        □.=.□
```

The shoulder tap has been passed on from soldier zero to soldier one.

A simple expression creates a few examples

```
        SFSGO¨1+⍳6

  □=□  □=.□  □=..□  □=...□  □=....□  □=.....□
  □⊖□  □.⍟□  □.=.□  □.=..□  □.=...□  □.=....□
             □..⍟□  □..=.□  □..=..□  □..=...□
                    □...⍟□  □...=.□  □...=..□
                            □....⍟□  □....=.□
                                     □.....⍟□
```

```
     ∇
 [0]    Z←SFSGO Y
 [1]    Z←SFSM SFSCH FSGO Y
     ∇ 1987-08-26 13.39.07 (GMT-4)
```

SFSGO, standing for Simple FSGO uses the variables defined above as left argument to the solution to the general problem, FSGO. The code for FSGO appears in the appendix. The right argument is the length of the squad. The reader may wish to confirm that the log symbol at SFSM[1;2;2] is only used for the one man squad. The starting position for a one man squad is the only instance of an equals sign between two quads.

The iterated line, FSGO[6], is the core of the solution. Before iteration starts, the transition matrix is converted to integer form using INDEX OF so that the look up is not required at each step, and the variable L is set to one less than the number of characters, that is, to the index of the firing state. L is used in line [6] to detect an occurrence of the firing state (⍟) so that iteration will stop.

The indices for the state of each individual automaton are determined in FSGO[6] by using the squad together with its left and right shifts. The two block heads appear together in both shifted forms. Thus, the blockhead entries in the transition matrix, which correspond to a soldier and one of his neighbors in state two, preserve that state over each time interval.

THE GENERAL CASE

If FSGO is called monadically it uses the global variables FSM and FSCH. It also uses the sign of its right argument to determine at which end of the line the officer stands. The conceit is simple. Two messages are sent, one traveling at one third the speed of the other. The delay in the slow message comes from having three waiting states before each soldier taps the shoulder of an adjacent soldier. When the slow message meets the reflection of the fast, the midpoint of the squad has been found. One or two soldiers, depending on parity, become blockheads, defining two squads of the same length, and fast and slow messages are dispatched from the midpoint in opposite directions. A blockhead in the general solution can leave his inert state only when one of a set of prefiring conditions is met; at the next moment he fires. Since a squad of any length can be called as a subdivision of at least one larger squad and will in that case be called from both left and right ends, the solution must be symmetric.

The symmetry of the overall solution and the use of blockheads to define subdivisions are illustrated in the following small example.

```
            □←(X Y)←(⊂'. ')REP_IN¨FSGO¨8 ¯8
    □=       □  □       =□
    □>→      □  □      ←<□
    □⊃ →     □  □     ← ⊂□
    □≥  →    □  □    ←  ≤□
    □ >  →   □  □   ←  < □
    □ ⊃   →  □  □  ←   ⊂ □
    □ ≥    → □  □ ←    ≤ □
    □  >    ←□  □→    <  □
    □  ⊃   ← □  □ →   ⊂  □
    □  ≥  ←  □  □  →  ≤  □
    □   >←   □  □   →<   □
    □   ||   □  □   ||   □
    □  =□□=  □  □  =□□=  □
    □ ←<□□>→ □  □ ←<□□>→ □
    □→ ⊂□□⊃ ←□  □→ ⊂□□⊃ ←□
    □ |≤□□≥| □  □ |≤□□≥| □
    □|□|□□|□|□  □|□|□□|□|□
    ⍟⍟⍟⍟⍟⍟⍟⍟⍟⍟  ⍟⍟⍟⍟⍟⍟⍟⍟⍟⍟
```

The function REP_IN replaces the dots with blanks, which removes clutter from the figure. There is obvious symmetry in these two figures; one is just the reflection of the other

```
        X ≡ ⌽Y
  0
```

Oops! It is also necessary to reverse the individual characters using

```
        ∇REVCH[⎕]∇
     ∇
 [0]    Z←REVCH Y ⍝  Where defined, REVCH¨ ←→ REVCH
 [1]    (,Z)←(' -',FSCH)[(' -',FS¯CH)⍳,Z←Y]
     ∇ 1987-08-24 15.08.57 (GMT-4) and
              ⊃FSCH FS¯CH
        →>⊃≥=.□|←<⊂≤⍟
        ←<⊂≤=.□|→>⊃≥⍟
```

The high minus sign in the name FS¯CH suggests a reversal. The directed symbols are interchanged in the two variables; the symmetric symbols are unchanged.

```
        X ≡ REVCH ⌽Y
  1
```

That's better! The rich APL symbol set led to characters, originally chosen to save space in an APLSV workspace, which themselves express a symmetry. Figure 1, which follows the text, gives the complete run of a seventeen man squad and the end of a seventy-five man squad. Parts of the smaller example occur in the larger. The seventy-five man squad divides five times.

DISCOVERING INTERNAL SYMMETRY

Once the notion of the reflected messages and the key sixth line of FSGO were in hand, there remained only the entry of items in the transition matrix. This was done experimentally by changing locations, lines, and planes in the matrix. The first solution looked something like Figure 2. Not much symmetry there.

The reader might have noticed that the array in Figure 2 has length fifteen in each axis rather than length twelve. The first step in improving the solution was to use a trace matrix to keep track of locations in the transition matrix which were actually used during a set of runs of FSGO. The trace matrix was a bit array, originally all zeroes, with the same shape as the transition matrix. A line of code was added to the then current version of FSGO to enter ones in the trace matrix at each time step. It was quickly clear that few of the locations were in fact used, for example, no opposing left and right arrows are ever adjacent. With this information states were combined, reducing the size to 12 by 12 by 12. From fifteen to twelve may not seem like much but

```
      12 15 ÷.× 3
0.512
```

so that for a rank three array, nearly half the space is saved.

The twelve remaining states were of three distinct types, representing respectively, right moving, stationary, and left moving states. The set of characters was reordered to form the variable FSCH given above. Figure 3 shows the transition matrix after each of its axes had been correspondingly reordered.

Little symmetry shows here, but symmetry requires the right point of view. In the form shown in Figure 3, the three axes represent left neighbor, soldier, right neighbor. This means that the rotation control vector in FSGO[6] was ¯1 0 1. That is a reasonable way to arrange a set of consecutive integers, but it is from the point of view of the guy in the middle that symmetry is most readily apparent.

Figure 4 shows the matrix with the first two axes interchanged. This is the form of the matrix used by FSGO. There are some more patterns, but symmetry is still elusive. To appreciate the effect of the axis transposition, the reader is challenged to find in Figure 3 the sort of cut-off rectangle of mostly ⎕ in item [1;2] of Figure 4. A good place to start is the bottom row of Figure 3.

Now notice that in corresponding items in the top and bottom rows the same characters appear reversed. That is reminiscent of the need to reverse characters to obtain an overall symmetry of the result. Figure 5 is the result of partitioning each item of Figure 4 corresponding to the groups of characters representing left moving, stationary, and right moving states. This matrix, FSMD3 is of depth three and satisfies the following identity

```
      FSMD3  ≡̲  ⊖ R̲E̲V̲CH¨¨¨ ⍉¨¨ BT¨ FSMD3
                where

      ∇BT[⎕]∇
    ∇
[0]   Z←BT Y  ⍝  ←→  ○_/ : Transpose on minor axis
[1]   Z←⍉⌽⊖Y
    ∇ 1987-08-06 14.03.03 (GMT-4)
```

In the identity and in line zero of the function the underscore is used between two characters to designate a single overstruck symbol which is not in the character set. This convention is adopted from an APL2 input convention. Using the symbols ⌽ and ○_/ for R̲E̲V̲CH and BT we can write

```
      FSMD3  ≡̲  ⊖ ⌽¨¨¨ ⍉¨¨ ○_/¨ FSMD3
```

This identity expresses the symmetry of the solution transition matrix which underlies the symmetry of results. We shall return to discuss this symmetry later on.

SOME OBSERVATIONS ON STORAGE

The transition matrix has 1728 leaves of which only 102 are not '-'. Sparse array techniques could be used to store this matrix, although for one so small not much would be gained. The arrangement of nontrivial leaves in the matrix is, however, not random. The partitioned matrix FSMD3 has twelve items each of which has nine subitems. Only thirty-two of the one-hundred-eight subitems contain any useful information; those with sixteen minus signs could be replaced by just one. Furthermore, only eight of the thirty-two useful subitems contain instances of more than one character. The other twenty-four could be replaced by the respective character scalars.

The resulting array, FSMRD, is shown in Figure 6. The name stands for Firing Squad Matrix of Ragged Depth. The figure shows the matrix both naked and with each item DISPLAYed using the function provided with APL2. The bare display is catenated to the form with boxes so that rows of items in the two forms are collinear. Since REP_IN has replaced minus signs with blanks, the minus signs which appear in the figure are only those produced by the DISPLAY function. The top and bottom rows of FSMRD each contain eight significant characters. Seven of these are boxed; in the case of the eighth an entire item has been replaced with a scalar, so it is DISPLAYed with only a single accompanying minus sign. The middle row contains the other sixteen subitems, with one, eight, four, and three in its respective items.

The system function ⎕AT can be used to document the space saving of the nested array. With left argument 4 and a variable name as right argument the function returns a two item vector. The first item is the total number of bytes used to store the array; the second item is the number of bytes used to store actual data. The difference of the two numbers is the nesting overhead.

Here is a short terminal session.

```
      Y ← 1E64×X ← 100×FSMRDI ← FSCH ⍳D̲(1 0) FSMRD

      NAMES
 FSM    FSMD3 FSMRD
 FSMRDI X     Y

      ,['']¨⊂[0]⊃NAMES(,¨-/¨Z)(Z←4 ⎕AT¨NAMES)
  FSM               FSMD3             FSMRD
  24                1940              692
  1752 1728         3668 1728         904 212

  FSMRDI            X                 Y
  692               692               692
  904 212           1540 848          2388 1696

      FM ← FSCH , '-'

      ⊃FM Y←+/FM∘.=X←∊FSMRD
 → > ⊃ ≥ =  .  ⎕  | ← < ⊂ ≤ ⍟   -
 4 2 2 1 4 19 13 10 4 2 2 1 6 142

      ⍴X
 212

      212 70÷1728
  0.123 0.041
```

The 2 by 3 array in the middle shows the space requirements for all six arrays. The second line in each item, generated by the minus reductions, gives the structure overhead explicitly.

The variable FSMRDI has the same structure as FSMRD, but for each character has been substituted its index in FSCH. INDEX OF is applied with left argument FSCH to each of the scalars in FSMRD using an experimental depth operator, D̲, which is shown in the appendix. Since all these indices are small positive integers, the array uses the same amount of space as FSMRD. X and Y are defined to show the effect of changing data element size. Uniform nesting of the data results in use of about twice as much space as the shallow form uses. The ragged depth form of the array uses about half as much as the original flat array. When larger integers are used data storage space increases, but overhead is unchanged.

Again referring to the terminal session above, note that of the 212 leaves of the array only 70 are in fact useful, since the remaining 142 are all minus signs, representing locations that are never referred to. Still, only one-eighth of the original 1728 leaves remain. The inutile items are displayed as blanks, which are underscored by the minus signs generated by the DISPLAY function. They are distributed 32, 78, 32, respectively, in the three rows of the ragged array. Perhaps structural techniques are competitive with sparse array techniques for some larger problems.

The symmetry identity

```
    FSMRD  =__  ⊖ REVCH¨¨¨ ⍉¨¨ BT¨ FSMRD
```

still holds for the ragged array because the ENCLOSE of a shallow scalar is the scalar itself.

ANALYZING SYMMETRY

The twelve items in the partitioned array of Figure 5 each correspond to a state of the center soldier according to the pattern

```
        →  >  ⊃  ≥

        =  .  □  |

        ←  <  ⊂  ≤
```

This fact is suggested by the upper diagram to the right in Figure 5 . The leaves of the array correspond to the next state of the automaton. Thus the two left most operations, ⊖ and REVCH¨¨¨, interchange respectively the input and output states of the center automaton. For example, if an automaton is in '>' between two '.' states his next state will be '⊃'. This is represented by the '⊃' in the (0 1)(1 1)(1 1) position of the matrix. The three item nested vector could be used as left argument to the PICK function. Each item in the vector selects an item in a level of the transition matrix, beginning at the top. Thus 0 1 selects item one in row zero, 1 1 selects subitem one in row one of that item, and the second 1 1 selects the '⊃'. The other '⊃' in item 0 1, corresponding to a '>' between a '□' and a '→' is selected by (0 1)(1 0)(2 0). It occurs in the illustrations of FSGO 8 shown earlier. The two operations thus assure that the symmetric cases are handled correctly as far as the center automaton of a triple is concerned.

The remaining two operations take care of the center automaton's view of his neighbors. Let R, S, and L stand for right moving, stationary, and left moving states. Then each subitem corresponds to a combination of two of these types of states

\ω α\	R	S	L
R	RR	RS	RL
S	SR	SS	SL
L	LR	LS	LL

where the rows correspond to the left neighbor; the columns, to the right. In the lower diagram to the right in Figure 5 the breakdown of neighbor states into the three categories is indicated by spaces in the row and column of symbols. If an automaton is in, say, the LS configuration, that is, he has a left moving state on his left and a stationary state on his right, then in the symmetric state he must have a stationary state on his left and a right moving state on his right. To the automaton the important thing is whether the neighbor is bearing an incoming message or an outgoing message. As the diagram just above shows, the operation which makes that switch is BT¨, the backwards transpose on items. Finally, the ⍉¨¨, transpose on subitems, interchanges the left and right automatons. The symmetries of the transition matrix support the symmetry of results mentioned earlier.

USING THE NESTED TRANSITION MATRIX

The function FSGON, for Firing Squad Go Nested, uses the nested transition matrix. It is similar to FSGO but requires two functions to handle the indexing paths which PICK uses to access array items below the first level. All three functions are shown in the appendix. FSMITR translates the indices into FSM, which are derived directly from the firing squad line, into paths of length three which would work to retrieve items from FSMD3. Since all the paths are of length three, an index error would occur with FSMRD for leaves which are not three levels into the array, that is exactly those scalars whose substitution for larger subarrays led to space savings. The function SCPIC is a variant of PICK which ignores all items in the left argument after a shallow scalar is reached.

CONCLUSIONS

In the above simple analysis of a classic problem, use of a nested array to describe the solution provided two quite different kinds of insight:

- The symmetries of the transition array, which underlie the symmetry of the solution, are easily expressed as reflections at different levels of the array. The group of reflections includes one across each of the axes and diagonals. The symmetry expression holds for the ragged depth version of the matrix without change.
- The transition matrix is a small array, yet use of a nested structure results in saving about one half the storage space. This is achieved with no basic change in the program which uses it.

APPENDIX

```
      DISP_FS
                 Miscellaneous functions and data

  Z←IM FSGO FSI;CHR;L  ⍝  Simulate the firing squad
  ⍝ INPUTS: LEFT - Transition matrix / Character list
  ⍝         RIGHT - Squad length; start at right if < 0
  ⍎(0=⎕NC 'IM')/'IM←FSM FSCH'  ⍝  Globals for standard problem.
  (L IM)←(¯1+⍴CHR)((CHR,'-')⍳↑(IM CHR)←IM) ⍝ Set local variables.
  FSI←CHR⍳,Z←,[¯.1]1⌽'□□',(FSI<0)⌽'=',(¯1+|FSI)⍴'.'  ⍝  Starting line.
  ⍎(FSI∧.≠L)/'→↑⎕LC Z←Z,[0]CHR[FSI←(⊂¨⊂[0]⊃0 ¯1 1⌽¨⊂FSI)⊃¨⊂IM]' ⍝ Iterate.

  Z←REVCH Y ⍝  Where defined, REVCH¨ ←→ REVCH
  (,Z)←(' -',FSCH)[(' -',FS¯CH)⍳,Z←Y]

  Z←BT Y  ⍝  ←→  ○_/ : Transpose on minor axis
  Z←⍉⌽⊖Y

  FSCH:   →>⊃≥=.□|←<⊂≤⍟   -:+:-   FS¯CH:  ←<⊂≤=.□|→>⊃≥⍟
```

```
             Assorted identities.

         FSMD3  =__  ⊖ REVCH¨¨¨¨¨¨ ⍉¨¨¨¨ BT¨ FSMD3
1
          FSM   =__    ⊃ , ↑¨ ,[0]/¨ ,/¨ FSMD3
1
        FSMD3=__⊃[0]¨⊃¨¨¨¨¨¨(⊂M)CUT¨¨¨¨⊂[1]¨¨¨¨⊃[0]¨¨¨¨(M←⊂12⍴1+4↑1)CUT¨⊂[0]¨3 4⍴⊂[1 2]FSM
1

         FSMRD  =__  ⊖ REVCH¨¨¨¨¨¨ ⍉¨¨¨¨ BT¨ FSMRD
1

         FSMD3I FSMRDI  =__  (⊂FSCH) ⍳D(1 0)¨ FSMD3 FSMRD
1

⍝       FSMD3   ←-→   ((⊂M) ⊂[1]¨ M ⊂[0]⍵)¨ 3 4⍴⊂[1 2] FSM      (someday ?)
⍝       FSMD3   ←-→    (((⊂M) ⊂) AX∘⍵)¨ 3 4⍴⊂[1 2] FSM      (some later day ?)

        FSM=__1 0 2⍉FSMO[M;M;M←¯1↓FSCHO⍳FSCH]
1
        ⊃FSCH FSCHO   ⍝  As originally conceived
→>⊃≥=.⎕|←<⊂≤⊛
.→>⊃≥←<⊂≤=|⎕⊛



        0 SHLST FSFNLST

∇ 0.   Z←X SFSGO Y  ⍝  Solution for small squad.
  1.   Z←SFSM SFSCH FSGO Y


∇  0.  Z←L(F D G)R;MON;T    ⍝    ¨_.        (DEPTH)
⍝  1. ⍝ NEEDS: IF P J
⍎  2.  ⍎'→0⍴⎕EX''F''IF F=__J' IF 2=⎕NC 'F'
⍎  3.  ⍎'→0⍴⎕EX''G''IF G=__J' IF 2=⎕NC 'G'
   4.  Z←,⍎(4×MON←0=⎕NC 'L')↓'(=__L)(=__R)'  ⍝        Z ← vector of arg. depth(s)
→  5.  →LD LJ RD RJ IF∊(⊂2 0)=⎕NC¨'FG'
⍝  6. ⍝                The f-f invocation is as suggested in [Iverson '83, p 44]
→  7.  →0 Z←⍎MON↓∊'L F ' 'D G¨' 'R' IF 1 0 1∨(0<⌈/Z)>⍎MON↓'L G R'
⍝  8. ⍝→0 Z←⍎∊(1 0 1∨(0<⌈/Z)>⍎T,'G R')/(T,'F ')'D G¨' 'R',0⍴T←MON↓'L '
 ÷ 9. LD:F←1↓4↑0,F  ⍝                            In case empty F is indicated by ''.
⍝ 10. ⍝                 P is the power operator, which cannot take ¨ as operand.
→ 11.  →0 Z←⍎∊MON↓'(⊂P(0⊃F)L)' 'G'((1⊃F)⍴'¨')'⊂P(2⊃F)R'
⍝ 12. ⍝             Alternative using the promoting transform of [Benkard APL84]
⍝ 13. ⍝ ⍎(10×MON↓)'(⊂P(0⊃F)L)   G(P(1⊃F))I¨   ⊂P(2⊃F)R'
 ÷14. LJ:
 ÷15. RJ:⎕ES 5 4 ⍝                               Both operands must be present.
 ÷16. RD: ⍝                Someday, figure out what an ¨ over G means if 0≠=__G!
→ 17.  →SYMM IF 0=⍴⍴G  ⍝        Asymm is the LEVEL operator of [Thomson APL84]
  18.  ⎕ES 5 3 IF MON∧2=⍴G   ⍝                                       Length error.
⍎ 19.  ⍎'→NNEG Z←Z×~T' IF∨/T←0∊¨⍴¨G   ⍝_ Empty operand item means no control.
⍝ 20. ⍝ →0 Z←⍎MON↓'L',(' F ' UNLESS '((¯1⌽1,1⌽Z)D(F D G))' IF∧/Z←Z≤2⍴G),'R'
→ 21.  →0 Z←⍎MON↓'L',((∧/Z←Z≤2⍴G)⊃' F ' '((¯1⌽1,⌽Z)D(F D G))'),'R'
⍝ 22. ⍝                        Alternative using the MESH invocation of  B (\).
⍝ 23. ⍝→0 Z←⍎MON↓'L',((∧/Z←Z≤2⍴G)⊃' F ' '((1(1 0 1\)Z)D(F D G))'),'R'
→÷24. SYMM:→NNEG IF 0≤G            ⍝         Neg does (|G+1)¨S, but stops at
→ 25.  →0 Z←⍎MON↓'L F ',('D G¨' IF(⌈/Z)0∧.>0 G←G+1),'R' ⍝    shallow scalars.
 ÷26. NNEG:Z←⍎MON↓'L F ',('D G¨' IF∊G<⌈/Z),'R' ⍝ applies to depth ≤G objects.


∇ 0.   Z←IM FSGON FSI;CHR;L  ⍝  Simulate the firing squad with nesting.
⍝ 1. ⍝ INPUTS: LEFT - Nested transition matrix / Character list
⍝ 2. ⍝          RIGHT - Squad length; start at right if < 0
⍎ 3.   ⍎(0=⎕NC 'IM')/'IM←FSMRDI FSCH'  ⍝  Globals for standard problem.
  4.   L←¯1+⍴↑⌽(IM CHR)←IM  ⍝ Set local variables.
  5.   FSI←CHR⍳,Z←,[¯.1]1⌽'⎕⎕',(FSI<0)⌽'=',(¯1+|FSI)⍴'.'  ⍝  Starting line.
⍎ 6.   ⍎(FSI∧.≠L)/'→↑⎕LC Z←Z,[0]CHR[FSI←(FSMITR¨⊂[0]⊃0 ¯1 1⌽¨⊂FSI)SCPIC¨⊂IM]'


∇ 0.   Z←FSMITR Y  ⍝  Shallow indices (FSM)  ==>  depth three paths (FSMD3)
  1.   Z←(⊂3 4⊤Y[0]),⊂[1]3 4⊤1↓Y


∇ 0.   Z←X SCPIC Y
⍝ 1. ⍝ Allows use of length three paths with  FSMRD
⍝ 2. ⍝ Z←('(1↑X)⊃Y' ⎕EA '''(2↑X)⊃Y''⎕EA''X⊃Y''')  ⍝  slightly slower
→ 3.   →⍳0==__Z←X[0]⊃Y
→ 4.   →⍳0==__Z←X[1]⊃Z
  5.   Z←X[2]⊃Z
```

```
     (FSGO 17)  (¯43↑[0]FSGO 75)
□=................□   □...............←.⊂□⊃.→...............□...............←.⊂□⊃.→...............□
□>→...............□   □..............←..≤□≥..→..............□..............←..≤□≥..→..............□
□⊃.→..............□   □.............←..<.□.>..→.............□.............←..<.□.>..→.............□
□≥..→.............□   □............←...⊂.□.⊃...→............□............←...⊂.□.⊃...→............□
□.>..→............□   □...........←....≤.□.≥....→...........□...........←....≤.□.≥....→...........□
□.⊃...→...........□   □..........←....<..□..>....→..........□..........←....<..□..>....→..........□
□.≥....→..........□   □.........←.....⊂..□..⊃.....→.........□.........←.....⊂..□..⊃.....→.........□
□..>....→.........□   □........←......≤..□..≥......→........□........←......≤..□..≥......→........□
□..⊃.....→........□   □.......←......<...□...>......→.......□.......←......<...□...>......→.......□
□..≥......→.......□   □......←.......⊂...□...⊃.......→......□......←.......⊂...□...⊃.......→......□
□...>......→......□   □.....←........≤...□...≥........→.....□.....←........≤...□...≥........→.....□
□...⊃.......→.....□   □....←........<....□....>........→....□....←........<....□....>........→....□
□...≥........→....□   □...←.........⊂....□....⊃.........→...□...←.........⊂....□....⊃.........→...□
□....>........→...□   □..←..........≤....□....≥..........→..□..←..........≤....□....≥..........→..□
□....⊃.........→..□   □.←..........<.....□.....>..........→.□.←..........<.....□.....>..........→.□
□....≥..........→.□   □→...........⊂.....□.....⊃...........←□→...........⊂.....□.....⊃...........←□
□.....>..........←□   □.→..........≤.....□.....≥..........←.□.→..........≤.....□.....≥..........←.□
□.....⊃.........←.□   □..→........<......□......>........←..□..→........<......□......>........←..□
□.....≥........←..□   □...→.......⊂......□......⊃.......←...□...→.......⊂......□......⊃.......←...□
□......>......←...□   □....→......≤......□......≥......←....□....→......≤......□......≥......←....□
□......⊃.....←....□   □.....→....<.......□.......>....←.....□.....→....<.......□.......>....←.....□
□......≥....←.....□   □......→...⊂.......□.......⊃...←......□......→...⊂.......□.......⊃...←......□
□.......>..←......□   □.......→..≤.......□.......≥..←.......□.......→..≤.......□.......≥..←.......□
□.......⊃.←.......□   □........→<........□........>←........□........→<........□........>←........□
□.......≥|........□   □........||........□........||........□........||........□........||........□
□.......=□=.......□   □.......=□□=.......□.......=□□=.......□.......=□□=.......□.......=□□=.......□
□......←<□>→......□   □......←<□□>→......□......←<□□>→......□......←<□□>→......□......←<□□>→......□
□.....←.⊂□⊃.→.....□   □.....←.⊂□□⊃.→.....□.....←.⊂□□⊃.→.....□.....←.⊂□□⊃.→.....□.....←.⊂□□⊃.→.....□
□....←..≤□≥..→....□   □....←..≤□□≥..→....□....←..≤□□≥..→....□....←..≤□□≥..→....□....←..≤□□≥..→....□
□...←..<.□.>..→...□   □...←..<.□□.>..→...□...←..<.□□.>..→...□...←..<.□□.>..→...□...←..<.□□.>..→...□
□..←...⊂.□.⊃...→..□   □..←...⊂.□□.⊃...→..□..←...⊂.□□.⊃...→..□..←...⊂.□□.⊃...→..□..←...⊂.□□.⊃...→..□
□.←....≤.□.≥....→.□   □.←....≤.□□.≥....→.□.←....≤.□□.≥....→.□.←....≤.□□.≥....→.□.←....≤.□□.≥....→.□
□→....<..□..>....←□   □→....<..□□..>....←□→....<..□□..>....←□→....<..□□..>....←□→....<..□□..>....←□
□.→...⊂..□..⊃...←.□   □.→...⊂..□□..⊃...←.□.→...⊂..□□..⊃...←.□.→...⊂..□□..⊃...←.□.→...⊂..□□..⊃...←.□
□..→..≤..□..≥..←..□   □..→..≤..□□..≥..←..□..→..≤..□□..≥..←..□..→..≤..□□..≥..←..□..→..≤..□□..≥..←..□
□...→<...□...>←...□   □...→<...□□...>←...□...→<...□□...>←...□...→<...□□...>←...□...→<...□□...>←...□
□...||...□...||...□   □...||...□□...||...□...||...□□...||...□...||...□□...||...□...||...□□...||...□
□..=□□=..□..=□□=..□   □..=□□=..□□..=□□=..□..=□□=..□□..=□□=..□..=□□=..□□..=□□=..□..=□□=..□□..=□□=..□
□.←<□□>→.□.←<□□>→.□   □.←<□□>→.□□.←<□□>→.□.←<□□>→.□□.←<□□>→.□.←<□□>→.□□.←<□□>→.□.←<□□>→.□□.←<□□>→.□
□→.⊂□□⊃.←□→.⊂□□⊃.←□   □→.⊂□□⊃.←□□→.⊂□□⊃.←□→.⊂□□⊃.←□□→.⊂□□⊃.←□→.⊂□□⊃.←□□→.⊂□□⊃.←□→.⊂□□⊃.←□□→.⊂□□⊃.←□
□.|≤□□≥|.□.|≤□□≥|.□   □.|≤□□≥|.□□.|≤□□≥|.□.|≤□□≥|.□□.|≤□□≥|.□.|≤□□≥|.□□.|≤□□≥|.□.|≤□□≥|.□□.|≤□□≥|.□
□|□|□□|□|□|□|□□|□|□   □|□|□□|□|□□|□|□□|□|□|□|□□|□|□□|□|□□|□|□|□|□□|□|□□|□|□□|□|□|□|□□|□|□□|□|□□|□|□
⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛   ⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛⊛
  17  automata                                            75  automata
```

The last eighteen lines of the larger squad comprise eight replications of each half of the smaller squad.
FIGURE 1.

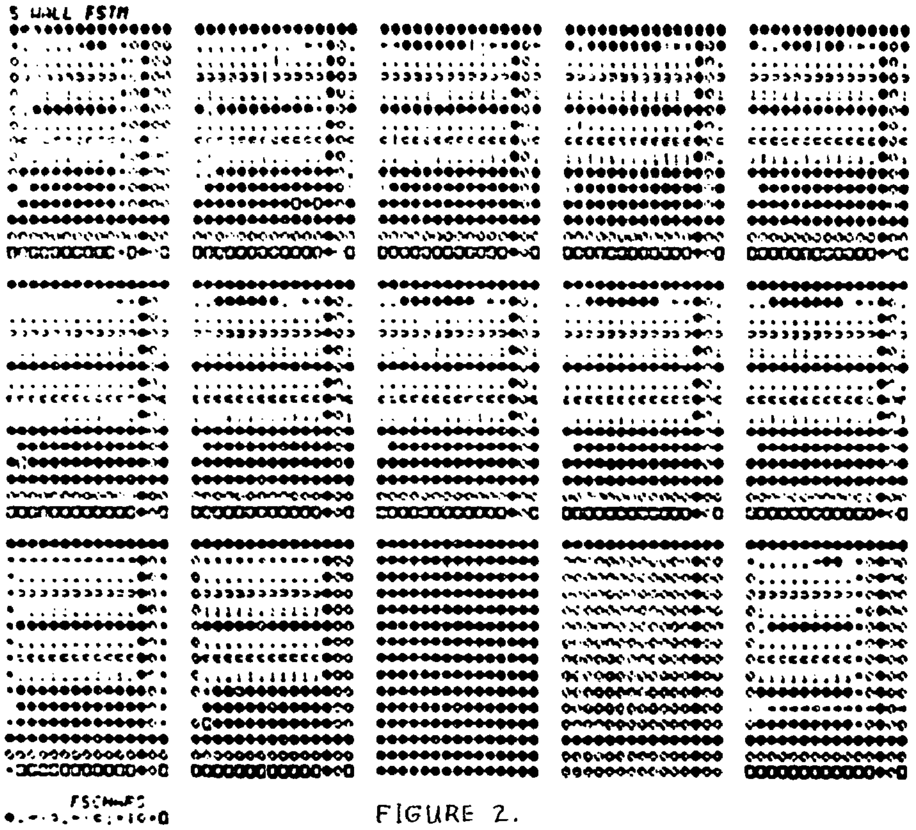


FIGURE 2.

```
      3 4⍴⊂[0 2]FSM     ⍝   Wallpaper display
------------   -----.------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
-----→←---|-   -----.------   .----.--|---   ----->------
------------   ------------   ------------   ------------
------------   ------------   ------------   -----□------
------------   -----||-----   ------------   ------------
-----||-----   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------

------------   -----.---|--   -----.---|--   ------------
------------   -----⊃--|---   ⊃-------|---   ------------
------------   -----≥------   -----≥------   ------------
------------   -----.-=----   -----.-|----   ------------
------------   ------<-----   ----->|-----   ------------
-----→←-----   ....←..=←..<   ....→.-|→---   -----=|-----
----□-□-----   -----□□-----   □□□□□□-⊕----   ------⊕⊕----
------------   -------□---□   ------⊕⊕----   -----□⊕-----
------------   -----..--.--   ------------   ------------
------------   -----⊂------   ------------   ------------
------------   -----≤≤-----   ------------   ------------
------------   -----..-----   ------------   -----=|-----

------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
-----..---.-   -----..-----   -----..-----   -----..-----
□-----□-----   -□----□-----   --□---□-----   ---□--□-----
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------⊂-----   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
```

An intermediate version of the transition matrix.

FIGURE 3.

```
      3 4⍴⊂[1 2]FSM     ⍝   Wallpaper display
------------   ------------   ------------   ------------
-----.------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
-----.---|--   -----⊃--|---   -----≥------   -----.-=----
-----.---|--   ⊃-------|---   -----≥------   -----.-|----
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------

------------   -----→←---|-   ------------   ------------
------------   -----.------   ------------   ------------
------------   .----.--|---   ------------   ------------
------------   ----->------   ------------   -----□------
------------   -----→←-----   ----□-□-----   ------------
------<-----   ....←..=←..<   -----□□-----   -------□---□
----->|-----   ....→.-|→---   □□□□□□-⊕----   ------⊕⊕----
------------   -----=|-----   ------⊕⊕----   -----□⊕-----
------------   -----..---.-   □-----□-----   ------------
------------   -----..-----   -□----□-----   ------------
------------   -----..-----   --□---□-----   ------------
------------   -----..-----   ---□--□-----   ------------

------------   -----||-----   ------------   ------------
-----||-----   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
-----..--.--   -----⊂------   -----≤≤-----   -----..-----
------------   ------------   ------------   ------------
------------   ------------   ------------   -----=|-----
------------   ------⊂-----   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
------------   ------------   ------------   ------------
```

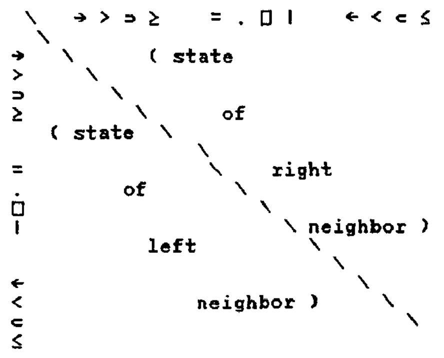
Final shallow transition matrix.

FIGURE 4.

```
FSMD3
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  -.--  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----

----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  -.--  -|--     ----  -⊃--  |---     ----  -≥--  ----     ----  -.-=  ----
----  -.--  -|--     ⊃---  ----  |---     ----  -≥--  ----     ----  -.-|  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----

----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----


----  ----  ----     ----  -→←-  --|-     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  -.--  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     .---  -.--  |---     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ->--  ----     ----  ----  ----     ----  -□--  ----

----  ----  ----     ----  -→←-  ----     ----  □-□-  ----     ----  ----  ----
----  --<-  ----     ....  ←..=  ←..<     ----  -□□-  ----     ----  ---□  ---□
----  ->|-  ----     ....  →.-|  →---     □□□□  □□-⍟  ----     ----  --⍟⍟  ----
----  ----  ----     ----  -=|-  ----     ----  --⍟⍟  ----     ----  -□⍟-  ----

----  ----  ----     ----  -..-  --.-     □---  --□-  ----     ----  ----  ----
----  ----  ----     ----  -..-  ----     -□--  --□-  ----     ----  ----  ----
----  ----  ----     ----  -..-  ----     --□-  --□-  ----     ----  ----  ----
----  ----  ----     ----  -..-  ----     ---□  --□-  ----     ----  ----  ----


----  ----  ----     ----  -||-  ----     ----  ----  ----     ----  ----  ----
----  -||-  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----

----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  -..-  - --     ----  -⊂--  ----     ----  -≤≤-  ----     ----  -..-  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  -=|-  ----

----  ----  ----     ----  --⊂-  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
----  ----  ----     ----  ----  ----     ----  ----  ----     ----  ----  ----
```

The partitioned transition matrix.

```
→   >   ⊃   ≥

=   .   □   |

←   <   ⊂   ≤

( state of center soldier )

    Items in matrix

==================================
```

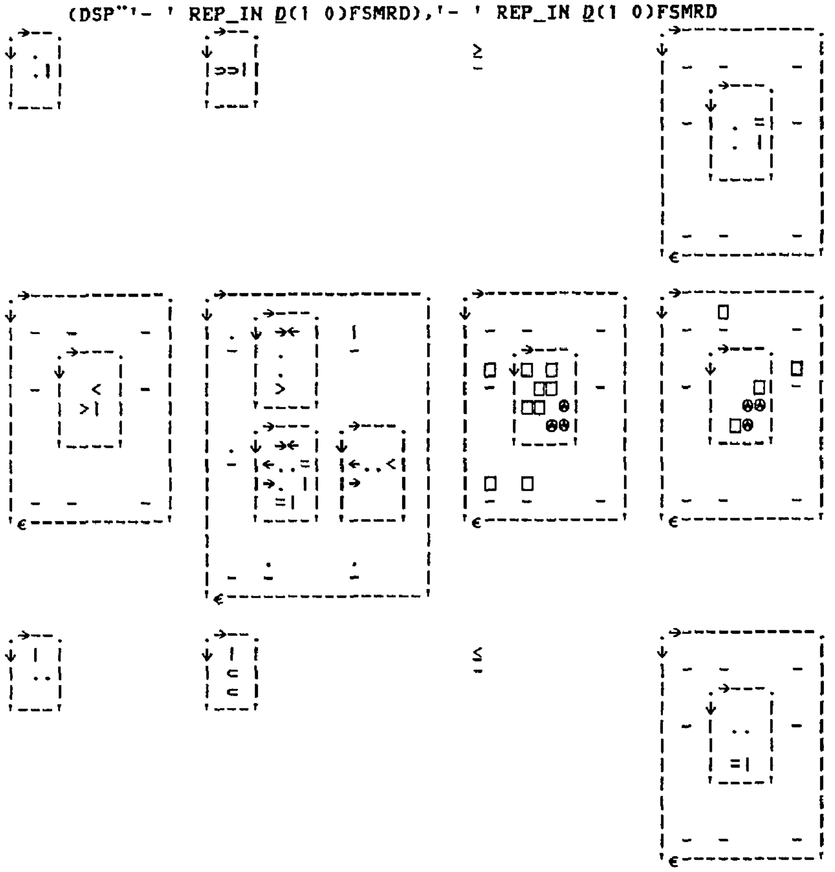


Scalars in each item.

Keys to the partitioned matrix.

FIGURE 5.

```
(DSP¨'- ' REP_IN D(1 0)FSMRD),'- ' REP_IN D(1 0)FSMRD
```


Ragged transition matrix with display boxes.

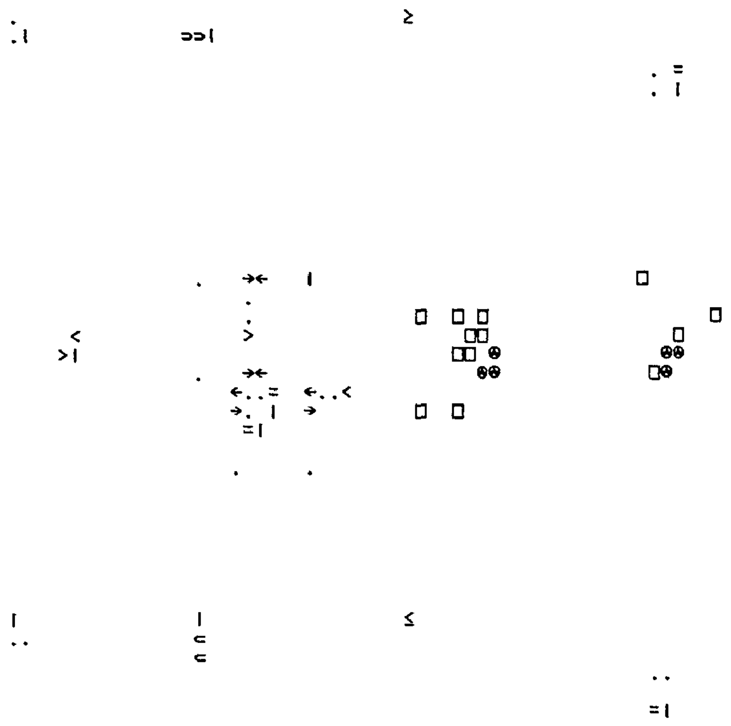

Unadorned ragged depth transition matrix.

FIGURE 6.