A FORTH Finite State Machine

Contributed by James Basile, Department of Computer Science
C. W. Post Center, Long Island University, Greenvale, New York

Finite state machines arise naturally in many applications and are a versatile, easy to understand structure to organize a control application around. For the uninitiated, [SMA82] provides a good background. The code in screens 1098-1101 does not perform state reduction and assumes that one already has worked out a state table or other similar description of the state machine.

The code provided assumes that an action (in the guise of a FORTH word) has been associated with each output. The parameter field for the table contains a pointer to the current state of the machine and a <pointer, size> entry for each state. These fields are created by the defining word STATES. A simple list is created for each state of the machine at a later time and the pointers are updated as the list is defined. In particular, each entry in each state's list consists of 4 bytes: a byte for the input value, a byte for the next state, and a word for the code field address of the output action.

The list for each state is initiated by the word STATE followed by the number associated with the state (e.g., STATE 7). Entries for the state are subsequently defined by !S. The list for each state is terminated by the entry OR-ELSE which provides the machine with code and a next state to associate with an input not explicitly described. This construct can also be used to handle the common situation in which all but a few inputs have the same output and next state.

Once all states for the machine are defined the initial state of the machine must be selected. This can be done by SET-STATE either at definition time or by NEW-STATE in the application.

The simplified example in screens 1102-1103 illustrates use of the state machine for emulating a smart terminal. Suppose a terminal responds to commands in the form of escape sequences beginning with the ASCII character ESC (decimal 27). For simplicity, let us assume there are three commands: ESC C clears the screen; ESC P x y positions the cursor to row x, column y; ESC T n sets the tab to every n columns (i.e., an ASCII TAB (decimal 9) will advance the cursor position to the next multiple of n).

In the initial state the system merely displays characters unless an ESC or TAB is detected. These characters demand special treatment and are defined in state 1. If an ESC is detected the machine is put into state 2. There commands are interpreted. If additional parameters are needed, a transition to another state is specified.

References

[SMA82] Small, C. "Finite-State Machines, Theory, Synthesis, and Minimization." *1982 Rochester Forth Conference on Databases and Process Control.* Rochester: Institute for Applied Forth Research, 1982.

SCR # 1098

```
 0 ( FINITE STATE MACHINE - J. Basile )
 1 ( The following blocks provide 79-STANDARD code to im-
 2 plement a finite state machine in software. The machine expects an
 3 8-bit value on the stack and performs a FORTH output word
 4 corresponding to that input and to the current state and also sets
 5 the next state. The output word has the input still on on the stack.
 6 The code is easily modified to handle larger size inputs. This code
 7 is written to run under Friendly FORTH on an IBM-PC )
 8
 9 VARIABLE >FSM< ( pointer used during table definition to
10       the address of the parameter field of a FSM )
11
12 1099 LOAD  1100 LOAD  1101 LOAD  ( state machine )
13
14 1102 LOAD  1103 LOAD  ( sample application )
15
```

SCR # 1099

```
 0 ( BASIC WORDS )
 1 : SCAN ( input, FSM addr -- input, ptr to action )
 2    SWAP OVER @ @ ( thread )  COUNT ( # of inputs defined )
 3    ?DUP IF 0 DO ( search for input )
 4      2DUP C@ = IF LEAVE  ELSE 2+ 2+  THEN LOOP
 5    THEN  1+ ;
 6 : NEW-STATE ( state#, FSM addr -- )
 7    DUP ROT 2 * + SWAP ! ;
 8
 9 ( interpretive mode only )
10 ( set state of an FSM )
11 : SET-STATE ( USAGE: state# SET-STATE <fsm-name> )
12    [COMPILE] '  NEW-STATE ;
13 ( get state of an FSM )
14 : ?S ( --current-state ; USAGE: ?S <fsm-name> )
15    [COMPILE] '  DUP @ SWAP - 2 / . ;
```

SCR # 1100

```
 0 ( DEFINING WORD TO CREATE AN FSM )
 1 : STATES ( n STATES <name> )
 2    CREATE  HERE >FSM< !
 3      1 + 2 * ALLOT ( ptrs for threads & current one )
 4    DOES>  SCAN  ( find action )
 5      COUNT 4 ROLL NEW-STATE          ( set new next state )
 6      @ EXECUTE ;          ( perform output )
 7
 8
 9
10
11
12
13
14
15
```

```
SCR # 1101

 0 ( CREATING A STATE )
 1 ( these words are designed to be used in interpretive mode )
 2 ( syntax for each word is commented )
 3 : STATE ( STATE n ; begin definition of state n )
 4       32 WORD NUMBER DROP  ( single precision result )
 5       2 * >FSM< @ +  DUP >FSM< @ !  ( make state current )
 6       HERE SWAP !  0 C, ;
 7 : !S  ( input next-state !S <action> ; define an input for
 8       current state )
 9       SWAP C, C,  [COMPILE] '  CFA ,
10       >FSM< @ @ @ ( thread being defined )
11       DUP C@  ( current count )  1+ SWAP C! ;
12 : OR-ELSE  ( next-state OR-ELSE <action> ; define default
13       for the state )
14       0 C,  ( dummy byte )  C,  ( next state )
15       [COMPILE] ' CFA ,  ( default action if input not found ) ;

SCR # 1102

 0 ( SAMPLE APPLICATION - TERMINAL EMULATOR )
 1 ( @CURSOR , !CURSOR , and CLEAR are system words which
 2       set and read the cursor position and clear the screen,
 3       respectively. )
 4 CREATE >TAB<  5 C,
 5 : TAB DROP ( input )  @CURSOR  DUP >TAB< C@ MOD  -
 6       TAB C@ +  !CURSOR ;
 7 : !TAB 20 MIN  ( max tab spacing )  >TAB< C!  ;
 8 : NOOP ;
 9 : BAD-CMD  ." ESC " EMIT ;
10 : CLR  DROP CLEAR ;
11
12
13
14
15

SCR # 1103

 0 ( EMULATOR STATE MACHINE )
 1 5 STATES DISPLAY
 2 STATE 1  ( just displaying )
 3       09 01 !S TAB  27 02 !S DROP  01 OR-ELSE EMIT
 4 STATE 2  ( received an ESC )
 5       67 03 !S CLR  80 03 !S DROP  84 05 !S DROP
 6       01 OR-ELSE BAD-CMD
 7 STATE 3  ( row for cursor command - leave on stack )
 8       04 OR-ELSE NOOP
 9 STATE 4  ( column - set the position )
10       01 OR-ELSE !CURSOR
11 STATE 5  ( tab setting update )
12       01 OR-ELSE !TAB
13 1 SET-STATE DISPLAY  ( set initial state )
14 ( sample emulator loop -- ?PORT get a character from RS232 )
15 : EMULATE  BEGIN  ?PORT  DISPLAY  0 UNTIL ;
```