Experiments on Finite State Machines
(Theory of FSM-Based Testing)

1. Terminologies and Definitions of experiments testing

   D-method
   W-method
   U-method
-- Definition of experiments on FSM

(1) Mealy Machine

A mealy machine is a FSM which can be characterized by a quintuple:

\( M=(I,S,O,f,g) \)

where

- \( I \): input alphabet
- \( S \): state alphabet
- \( O \): output alphabet
- \( f : S \times I \rightarrow S \)
- \( g : S \times I \rightarrow O \)

Example: \( M=((0,1),(S1,S2,S3,S4,S5),(0,1),F,G) \)

Table 1, Mealy machine M1

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S1,0</td>
<td>S4,1</td>
</tr>
<tr>
<td>S2</td>
<td>S1,0</td>
<td>S5,1</td>
</tr>
<tr>
<td>S3</td>
<td>S5,0</td>
<td>S1,1</td>
</tr>
<tr>
<td>S4</td>
<td>S3,1</td>
<td>S4,1</td>
</tr>
<tr>
<td>S5</td>
<td>S2,1</td>
<td>S5,1</td>
</tr>
</tbody>
</table>
1. Terminology

Experiment

Given a sequential:
- a: State table known
- b: Strongly connected
- c: Completely specified
- d: Reduced

Using testing architecture

Deciding unknown parameters (initial state/final state)

Sequential machine M=(I,S,O,F,G)

Experiment (testing)
- Simple experiment
- Multiple experiment
- Preset experiment
- Adaptive experiment
- Length of experiment
- Initial state identification experiment
- Final state identification experiment
- Machine verification experiment
Transition Diagram for M1

Assume initial state is in \((s2,s3,s4,s5)\)

- DS = 000
- HS = 000
- SS = 0000
- CS = (0, 00, 000, 1000)
2. Four Key Sequences

DS-SEQUENCE (Distinguishing Sequence)
An input sequence is a DS for a FSM if the output response of M to it is different for each initial state.

HS-SEQUENCE (Homing Sequence)
An input sequence is a HS for a FSM if its final state can be determined uniquely from the response to DS irrespective of the actual initial state of FSM.

SS-SEQUENCE (Synchronizing Sequence)
A synchronizing sequence of a FSM is an input sequence that takes FSM to a special final state regardless of the output or the initial state of the machine.

Characterizing Sequences
Characterizing sequences are a set of input sequences such that there exists a unique relationship between the observed output responses and the unknown initial state.
Construction of Successor Tree

A successor tree is a tree-like connected graph constructed by the following rule:

1: Apply each input symbol to the states of the machine (or a subset of Q), each of which develop a node that contains the next states obtained from the state transition function of the FSM;

2: Group all the next states in one group if the corresponding outputs are identical; otherwise, arrange them into different groups.

Each group is called an uncertainty (U). Groups in a node are called an uncertainty vector (UV).
Termination rules of successor trees for DS, HS, SS

When a successor tree is used for finding DS, HS and SS, it is called DS-tree, HS-tree and SS-tree respectively.

--DS (distinguishing tree)
(D1) An uncertainty vector in the kth level is also associated with some branch in a preceding with a homogeneous vector (each group has identical states);
(D3) A branch is associated with a simple uncertainty vector (each group has only one state).

--HS (homing tree)
(H1) A UV in the kth level is also associated with some branch in a preceding UV.

--SS (Synchronizing tree)
(S1) The synchronizing uncertainty vector associated with a kth level branch is identical to that of some branch in a preceding the next state for an input with no duplicates.
Table 1. Mealy Machine M1

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td></td>
<td>S1,0</td>
<td>S4,1</td>
</tr>
<tr>
<td>S2</td>
<td></td>
<td>S1,0</td>
<td>S5,1</td>
</tr>
<tr>
<td>S3</td>
<td></td>
<td>S5,0</td>
<td>S1,1</td>
</tr>
<tr>
<td>S4</td>
<td></td>
<td>S3,1</td>
<td>S4,1</td>
</tr>
<tr>
<td>S5</td>
<td></td>
<td>S2,1</td>
<td>S5,1</td>
</tr>
</tbody>
</table>
Table 2. Segmented Responses of M1 to 000

<table>
<thead>
<tr>
<th>Initial state</th>
<th>Responses to 0 0 0</th>
<th>Final state</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>0 0 0</td>
<td>S1</td>
</tr>
<tr>
<td>S3</td>
<td>0 1 0</td>
<td>S1</td>
</tr>
<tr>
<td>S4</td>
<td>1 0 1</td>
<td>S2</td>
</tr>
<tr>
<td>S5</td>
<td>1 0 0</td>
<td>S1</td>
</tr>
</tbody>
</table>
Fig 1. Diagnosing and homing tree for m1 with IUV=(s2,s3,s4,s5)
Fig. 2 Multiple experiment tree for M1

UV = (s1, s2, s3, s4, s5)

\[ l_1 = 0 \]
\( (s_1, s_4) \)

\[ l_2 = 00 \]
\( (s_1, s_3) \)

\[ l_3 = 000 \]
\( (s_4, s_5) \)

\[ l_4 = 1000 \]
\( (s_1, s_2) \)

Characterizing Sequences = \{ l_1, l_2, l_3, l_4 \}
Order of Experiment = 4
Length of Experiment = 10
Multiplicity of Experiment = 4
Fig. 3 Synchronizing tree for M1 with IUV={s2,s3,s4,s5}

SS is 0000
2. Design of Experiment

State Identification with conventional approach
*(successor tree approach)*

A: Initial state identification
   A.1: Simple experiment
       Procedure of experiment
       (1): Construct distinguishing tree
           (Using termination rule D1,D2 and D3)
       (2): Find **DS** and response table
       (3): Perform experiment
       --Simple preset experiment

Apply **DS**, observe output and determine the initial state
B. Final State Identification

Procedure
1: Construct Homing tree using termination rules H1, H2
2: Find the homing sequence HS
3: Perform experiment
    3.1 simple preset experiment
        Apply HS, observe response
        and draw conclusion
    3.2 simple adaptive experiment
        Divide HS into subsequences
        Apply subsequences to M until the
        response can be attributed to a final state.

Example: See Fig. 1 and Table 2

C. Synchronizing Problem Procedure
1: Construct synchronizing tree using rules S1, S2
2: Find SS
3: Perform experiment

Example: See Fig. 3
3. Machine Verification and software testing

Machine verification is to test the machine to see if it conforms to its specification.

The verification consists of the following three phases:

1: Initialization phase.
   Machine to be tested is first brought to a specific state from which the second phase will begin;

2: State identification phase.
   DS-sequence is repeatedly applied to the machine to see if it has n different states, n is the number of states in the machine;

3: Transition verification phase.
   Machine is made to go through all possible transitions.

Assume that TR stands for transfer sequence which takes the machine from one state to another and T is a transition.

Machine verification can be outlined as follows:

\[(HS-TR/SS)(DS-TR)^*(TR-T-DS)^*\]

Note that it is important to make testing sequence as short as possible.
3.2 Application of machine identification to software testing

D-method
State identification
SS-TR-DS (for each state)
Transition verification (b-sequence)
SS-TR-T-DS (for each transition)

W-method
Assume W-set = (w1,w2...wi)
(characterizing sequences)
State identification
SS-TR-w1
SS-TR-w2 (for each state)
: 
SS-TR-Wi

Transition verification
SS-TR-T-w1
SS-TR-T-w2 (for each transition)
: 
SS-TR-T-wi
U-method

Assume that there is a different DS/UIO for each different state.

State identification

SS-TR-UIOi (for each state)

Transition identification

SS-TR-T-UIO (for each transition)

Example, see following pages
Example for: D-method
W-method
U-method

Fig. 1 A transition diagram for a machine M.

<table>
<thead>
<tr>
<th>input state</th>
<th>output</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>λ</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. A transition table for machine in Fig. 1.
D-method DS=BB

α-sequence:
- r B B
- r A A A A B B
- r A A A B B
- r A B B
- r A A B B

β-sequence:
- r A B B
- r B B B
- r A A A A A B B
- r A A A A B B B
- r A A A A B B
- r A A A B B B
- r A A B B
- r A B B B
- r A A A B B
- r A A B B B

<table>
<thead>
<tr>
<th>state</th>
<th>Mls&lt;DS&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>λλ</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>1λ</td>
</tr>
</tbody>
</table>

Table 3. Outputs on DS for M in Fig. 1.

An optimized use case sequence constructed from the above test subsequences is:

rAAAAAABBBrAAAAABBBrAAABBBBrAABBBBrABBBrBBBBBB
U-method

The $\alpha$- and $\beta$-sequences generated by the application of U-method for M in Fig. 1 are given below.

$\alpha$-sequence:

- $0$: r B
- $1$: r A A A A A A A
- $2$: r A A A B
- $3$: r A B B
- $4$: r A A A A A

$\beta$-sequence:

- r A B B
- r B B
- r A A A A A A A
- r A A A A B B
- r A A A A A A A
- r A A A B B B
- r A A A A A
- r A B B B
- r A A A B
- r A A B B
- r A A B B

<table>
<thead>
<tr>
<th>state</th>
<th>UIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B/\lambda</td>
</tr>
<tr>
<td>1</td>
<td>A/1 A/1</td>
</tr>
<tr>
<td>2</td>
<td>B/0</td>
</tr>
<tr>
<td>3</td>
<td>B/1 B/1</td>
</tr>
<tr>
<td>4</td>
<td>A/1 A/0</td>
</tr>
</tbody>
</table>

Table 2. UIO sequences for M in Fig. 1.

An optimized use case sequence constructed from the above test subsequences is:

rAAAAAAArAAAAABBrAAAAABBBrAABBBrABBBrBBB
W-set = \{A, AA, B\}
(characterizing sequence)

\(\alpha\)-sequence:
- r A
- r A A
- r B
- r A A A A A
- r A A A A A A
- r A A A A B
- r A A A A
- r A A A A A

<table>
<thead>
<tr>
<th>state</th>
<th>Mls(A)</th>
<th>Mls(AA)</th>
<th>Mls(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(\lambda)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4. Last output symbols on W for M in Fig. 1.
β-sequence: (cont’d.)

β-sequence:

An optimized use case sequence constructed from the above test subsequences is:

rAAAAAAArAAAAABrAAAAABAArAAAAABBrAArABA
rAAAAABBrAAAAABArAAAAABBrABAArABArABBrBAr
Good News
for Requirements
Analysis &
Validation
Functionally-Targeted Requirements Analysis Methods

Graphical Methods:

1. **CAUSE EFFECT GRAPHING**
   - logical, casual model of user/product interactions
   - use cases track logical effect of combinations of user/product conditions and actions
   - reduced number of use cases selected

2. **FSM-BASED TESTING**
   - state transition model describes system response to user actions at a snapshot in time
   - use cases track means of reaching a goal state from an initial state
Graphical Methods (cont.)

2. Finite State Machine-Based Requirements Analysis

Assumptions

• at high level, user/product interactions can be described as a series of transitions between states of their interface
• current “state of user/product interface is a “snap shot” of key parameter values
• each equivalence class of (a vector of) interface parameter values can be represented by one state
• user and system-generated actions or indications correspond to “events”
• past history is not relevant to decision of next action, only current state and events.
FSM-Based Requirements Analysis (cont.)

For Functionally-Targeted (Black-Box Requirements Analysis), one FSM is often adequate:

- customer command
- PRODUCT RESPONSE
- present state of product (customer viewpoint)
- next state of product (customer viewpoint)
Describing Requirements by Finite State Machines

Finite State Representation

• Describes many possible scenarios at once
• Requires identifying the state of an entity or object
• Event-based: when an event occurs, a response and/or a change of state occurs
• Transitions are changes of state
FSM-Based Requirements Analysis (cont.)

Step 0: Capture Scenarios

Step 1: Derive layered (high-level, as detailed as req’d.) Finite State representation of customer requirements & validate against customer scenarios

Step 2: Select (Control Flow) Coverage Requirements
   – Weak, Untargeted
   – Weak, Targeted (Transition Coverage)
   – Strong

Step 3: Derive Scenarios to Satisfy Coverage Requirements
   – weak coverage for normal behaviours
   – strong or at least targeted coverage for exceptional behaviours

Step 4: Derive use cases to implement the scenarios
Describing Requirements by Finite State Machines

ey. Consider the states of a telephone handset interface:

1. AVAILABLE
2. NOT AVAILABLE

Events defined could include user actions:

1. go off-hook
2. go on-hook
3. dial a number

Handset responses could include:

1. no response
2. sound dial-tone
3. sound ringing tone
4. sound busy tone
Describing Requirements by Finite State Machines (cont.)

Types of Finite State Machine Representations:

1. State Transition Diagrams
Types of Finite State Machine Representations

2. State Transition Tables
(for previous example)

<table>
<thead>
<tr>
<th>Event</th>
<th>State</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
</table>
| State  | 1     | make response 1  
go to state 2 | - |
|        | 2     | -  | make response 2  
go to state 2 |

- means "unknown at this time" or "not specified"
Describing Use-Case Scenarios by Finite State machines (cont.)

Example FSM Diagram for Telephone Handset Interface

- go off-hook/sound dial tone
- go on-hook/no response
- dial number/sound ringing tone or dial number/sound busy tone
## Example FSM Table for Telephone Handset Interface

<table>
<thead>
<tr>
<th>Event</th>
<th>State</th>
<th>go off-hook</th>
<th>dial number</th>
<th>go on-hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 AVAILABLE</td>
<td>sound dial tone.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>next-state: 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 NOT AVAILABLE</td>
<td>sound ringing</td>
<td>X</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tone or sound</td>
<td></td>
<td>next-state: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>busy tone,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>next-state: 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

State indicates availability of user.
Event indicates user action performed on handset.
* means "any action whatsoever" (in this case, no special action)
Requirements Errors and FSM Representations

Ambiguous Response
(to same event "a")

Unspecified Transition
(in State 1, what if event "b" can occur and no response and next state is specified)

Unreachable State impossible to get to

Sink State impossible to escape from

Transition Error incorrect response or next-state
Steps:

1. Build Row Headers:
   Characterize distinct states of the object (service, feature, interface, etc.) you wish to describe. Be careful of the level of detail -- too much detail can lead to errors and misunderstanding.

2. Build Column Headers:
   Determine the events as those occurrences which bring about changes in the object's state.

3. Complete each entry in the table:
   In Row (State) I and Column (Event) J, insert the Response which should occur at that State for that Event. Then, indicate by NEXT-STATE: to which state a transition occurs. If State I/Event J should not occur, insert "X" or "NA".

Deriving FSM Tables from Requirements:
(Customer-Directed View)
Deriving Customer-Directed FSM Tables from Requirements

Workshop Example
3 Way Call:
English Description:

When A dials B and B answers, A and B are in a talking state. When A flash-hooks, and B is placed on hold, A gets a dial-tone and calls C. Assuming C answers, when A issues a flash-hook, B is brought back into the call, and all three parties (A, B, C) are in a talking state. If B is on hold and A hangs up, A is rung back; in all other situations, when A hangs up, the other parties get a disconnect treatment (assuming that A does not have call transfer). If B or C hangs up, appropriate treatment is given and any existing leg is preserved.
Step 1: Build Row Headers (States)

Since the purpose is 3WC, there will be two important attributes to capture, namely:

- call status between A and B
- and call status between A and C.

(assumes A is the “controller” for the call)

Thus, each state is a pair

(status of A-B leg, status of A-C leg)

This is a subjective activity and needs to be validated for accuracy with a customer-agent or domain expert if possible.

Missing information in the spec. may lead to the need for additional state attributes.
**Step 2: Build Column Headers (Events)**

<table>
<thead>
<tr>
<th>Event</th>
<th>A flash</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A talking with B.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 party call.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A talking with C.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B on hold.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A talking with B and C.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Basically, A can flash and any one of the parties can choose to disconnect. Thus, there appears to be only 4 events. But …?
## Example: 3 Way Call (continued)

### Step 2: Build Column Headers (Events)

<table>
<thead>
<tr>
<th>Event</th>
<th>A Flash</th>
<th>A Disconnect</th>
<th>B Disconnect</th>
<th>C Disconnect</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 A talking with B.</td>
<td>B is placed on hold. A gets SDT and calls. C.</td>
<td>A is idled. B goes to disc. Trtmt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 party call</td>
<td>Next-state: 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 A talking with C. B on hold</td>
<td>*A, B, C are talking.</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Next-state: 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 A talking with B and C.</td>
<td>C is removed from call and goes to disc. Trtmt.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Next-state: 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** What happens here? Considering this situation leads to a new state and a new event (one more row, one more column, and two new table entries). Building the FSM Table often leads to a more complete FSM in a systematic way.
## MODULE II  EXERCISES & EXAMPLES

### State-Event Table for Three Way Calling (A is 500/2500 set)

<table>
<thead>
<tr>
<th>Event</th>
<th>A</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>B is placed on hold. A gets SDT and calls C</td>
<td>A is idled. B goes to disc. trtmt.</td>
<td>B is idled. A goes to disc. trtmt.</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>2</td>
<td>* A, B, C are talking. Next-state: 3</td>
<td>C goes to disc. trtmt. A is rung back. Next-state: 4</td>
<td>B is idled. A and C remain talking. Next-state: 1</td>
<td>C is idled. A talking to B upon flashing or after disc. time</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>C is removed from call and goes to disc. trtmt. Next-state: 1</td>
<td>A is idled. B and C go to disc. trtmt.</td>
<td>B is idled. A and C remain talking. Next-state: 1</td>
<td>C is idled. A and B remain talking. Next-state: 1</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>NA</td>
<td>NA</td>
<td>Stop ring back. A is idled and B is idled.</td>
<td>NA</td>
<td>A and B are talking</td>
</tr>
</tbody>
</table>

Table 1: The above table illustrates the functionality of simple three way calling. Assume that the controller (A) does not have call transfer.
Example of Coverage Assessment:

Capturing the UserView functionality of a Link Set

SERVICES

TRANSFER
RING AGAIN
CANCEL RING AGAIN
HOLD
RETRIEVE HOLD
CONFERENCING

CALL PICK-UP
CALL FORWARD
CANCEL CALL FORWARD
PARK
RETRIEVE PARK

Also used in normal user scenarios for soak testing
1. Represent User View of Link-Set Product

**Current State =**

<table>
<thead>
<tr>
<th>Receiver Signal Status</th>
<th>Switch Hook Status,</th>
<th>Lamp Status,</th>
<th>Bell Status,</th>
<th>User Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stutter tone (4)</td>
<td>on</td>
<td>on</td>
<td>silent</td>
<td>initator</td>
</tr>
<tr>
<td>confirmation (2)</td>
<td>off</td>
<td>off</td>
<td>ringing</td>
<td>controller</td>
</tr>
<tr>
<td>dial tone</td>
<td></td>
<td></td>
<td></td>
<td>add-on</td>
</tr>
<tr>
<td>busy tone</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>voice phonems</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hold (pause)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>no sound</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ringing tone</td>
<td>9 x 2 x 2 x 2 x 3 = 216 possible states!</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROH tone</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(from user point of view)
1. Represent User View of Link-Set Product (cont.)

Note that some potential state values may not be available to customer.

   e.g. My set does not use the (message) lamp.

This reduces the number of potentially feasible states to 108.

Also, user role can be separated out for global use cases--reduces number of distinct states to 36.

Finally, only a few use cases need to be made for the cases when the handset is on-hook. So off-hook states will be listed first.
1. Represent User View of Product (cont.)

- Identify local events
  - do nothing
  - push R
  - push L
  - go on
  - off
  - hook
  - push {CODE, EXT, outside line, operator, invalid EXT, invalid CODE}
    (includes 2 keys pressed simultaneously, number not in service, fax number)
  - talk into transmitter
2. Construct Local State Transition Table (reachable states only)

[ but be careful --unreachable states may be reachable in future releases ]
(document as coverage exclusions)

i) identify start (stable) state (n, on, s)
   [no tone, on-hook, silent]
ii) systematically complete state table

“-” means “indeterminate - specification ambiguous or incomplete”
“x” means “must not occur”
“n” means “no effect”
“n/a” means “not applicable”
FPS 2000 # 1: Human Machine Interface

Your organization has acquired by a corporate buy-out, a new product called the “Fax-Plain and Simple” FPS 2000. This product consists of sensors, motors, and lamps which are all entirely controlled by software. An overhead view of the FPS 2000 is shown below. The documentation is poor to non-existent. Your task is to recapture customer requirements and design, and then to re-engineer the product to fit your product line.

Message Display Panel capable of displaying:
"Transmission OK"
"Time" (hour, followed by minutes)
"Transmitting Page" <page number>
"Dialing"
"Error"
"Receiving"
"Reception OK"
"Document Ready"
"Copying"

Input Area to insert transmission document (face down)

Output Area for Received/Copied Document
The purpose of the FPS 2000 is to send and receive facsimile documents in either manual or automatic (auto) mode. To send a document to a destination telephone number (optional 3 digit area code, followed by 7 digit number), a customer performs the following sequence of steps:

i) insert document face down into input area (8). The document should be automatically pulled part way into the machine, and the message panel (1) should display “Document Ready”.

ii) enter the destination telephone number on the telephone key pad (3):
    - if a local call, enter seven digits (not beginning with 0)
    - if a long distance call in the sender’s area code, enter “1”, followed by the seven digits of the number
    - if a long distance call outside the sender’s area code, enter “1”, followed by the 10-digit destination phone number.

Hit the GO AHEAD button (5). The message panel (1) should display “Dialling...the message panel (1) should display “Transmitting Page 1” and the document should be automatically pulled into the machine. If another page is inserted into the input area (8) before the first page is completely inside the machine, this next page will also be automatically pulled into the machine after the first page, and the message panel (1) will display “Transmitting Page 2”. This can be repeated for successive pages of transmission. Each page is transmitted electronically to the destination fax machine.

iv) at the end of a successful, complete transmission, the message panel (1) will display “Transmission OK” for 5 seconds, then display the time of day.

This is the normal sequence of interactions between the user and the machine. If any step is unsuccessful, or if the user hits the STOP button (4), the message panel (1) displays “Error” for 5 seconds, and then displays the time of day. Whenever) displays the time of day.

Note that Telephone Receiver/Speaker (9), Manual/Auto Selection Button (6), and Manual Mode Lamp (7) are not described here -- they will be described if needed.
Example:
FPS 2000 Reqs. : Finite-State Machine (FSM)

If we take an abstract view of the FPS 2000, we can arrive at the following FSM representation of the steps involved in transmitting a document from a User (sender) to a Destination Machine (DM) by means of a Sending Machine (SM). Only the interface between the User (U) and the Sending Machine (SM) is represented (the Sending Machine (SM) is the FPS 2000).

**States:**
- I ↔ Idle
- R ↔ Ready to Transmit
- T ↔ Transmitting
- E ↔ Error

**Events:**
- i ↔ insert page
- d ↔ dial destination number
- g ↔ press GO AHEAD
- s ↔ press STOP
- to ↔ timeout (5 seconds elapsed)
- f ↔ failure of attempted or pending action

**SM Responses:**
- TOK ↔ Transmission OK
- TIME ↔ Time of Day Message
- PAGE ↔ Transmitting Page
- DLNG ↔ Dialling
- ERR ↔ Error
- RDY ↔ Document Ready

The resulting FSM (simplified) is

**Note** that any events which are not specified at a particular state are NOT ALLOWED. e.g., at state R, the event is is not allowed. “-” means “no event”, i.e., just wait for something to happen.

**Note** that reception is not yet specified!
1. FSM Representation of Requirements (Conclusion)

- Continue Link Set Example
  - verify
  - validate

Note that High-Level FSM Representations promote:
  - systematic requirements capture, especially of state-transition oriented systems, products, interfaces
  - straight forward review of requirements
  - simple interpretation by design engineers
  - use of FSM Tools (e.g. SDT for SDL)

(Only the first step in Requirements Capture & Validation)
FSM-Based Requirements Analysis (coverage requirements for use cases)

**Steps for** weak, untargeted (**FSM transition tour**) coverage:
1. Derive and validate FSM representation of allowed customer/product interactions.
2. Identify cycles of transitions in the FSM.
3. Derive as few scenarios as possible which exercise all transitions.

**Steps for** weak targeted (**FSM-transition coverage**):
1. Derive and validate FSM representation of allowed customer/product interactions.
2. Find each transition between states of the FSM.
3. Derive the shortest scenario to reach each state.
4. For each state transition, derive a scenario consisting of:
   i) a *preamble* to reach the state
   ii) the event that causes the transition to occur
   iii) a *postamble* that verifies
      a) the product made the correct response
      b) the product changed to the correct next state
FSM-Based Requirements Analysis (cont.)

Steps for strong *(FSM-path)* use case coverage:

1. Derive and validate FSM representation of allowed customer/product interactions.

2. Identify natural *phases* in customer use of a product. A normal use or session will involve a sequence of phases which work together to achieve the user’s goal.

3. For each phase, identify all start states and all goal states (usually only one of each).

4. Derive Use Case Requirements
   a) For each phase, find the normal behaviour path from each start state to goal state (usually the longest path without any cycles).
   b) For each normal behaviour path, identify each transition that deviates from that path. Normally these are exception-handling transitions.

5. Derive Use Case Scenarios
   a) Cover normal paths with a few use cases as possible.
   b) Cover each exception-handling transition by its own path.
Example: Simple POTS Phone Interface FSM Requirements Analysis

States

Q ↔ quiet
B ↔ bell ringing
P ↔ pending
V ↔ voice interaction

Initial States
\{Q, B\}

Goal States
\{V, Q\}

Events

gof ↔ go off-hook
gon ↔ go on-hook
d ↔ dial
to ↔ timeout

Product Responses

DTN ↔ present dial tone
VCE ↔ carry voice conversation
BSY ↔ present busy tone
ROH ↔ present receiver off hook tone
NRS ↔ no response
Example: Simple POTS (cont.)

**STEP 1:** (common to all coverage levels)

Derive and Validate FSM for POTS Interface
Based on the above States, Events, and Product Responses, complete the FSM diagram below:

![FSM Diagram]

Document N/A Transitions in the Table below:

<table>
<thead>
<tr>
<th>State</th>
<th>Event(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>
Example: Simple POTS Answers

**STEP 1:** (common to all coverage levels)

Derive and Validate FSM for POTS Interface
Based on the above States, Events, and Product Responses, complete the FSM diagram below:

(One transition was omitted for simplicity -- can you find it?)

Note: N/A Transitions (should be documented and validated)

<table>
<thead>
<tr>
<th>State</th>
<th>Event(s)</th>
<th>(internal transition spontaneous)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>gon, d, to</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>gof</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>gof, d, to</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>gon, d</td>
<td></td>
</tr>
</tbody>
</table>
Use Case Paths (Use Case Requirements) which provide weak, untargeted coverage are:

<table>
<thead>
<tr>
<th>Path Number</th>
<th>State Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Scenarios will be discussed later)
Abstract Use Cases which provide *weak, untargeted* coverage are:

<table>
<thead>
<tr>
<th>Path Number</th>
<th>State Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Q, P, P (ROH), Q, P, P (BSY), Q, P, V, Q</td>
</tr>
<tr>
<td>2</td>
<td>B, V, Q</td>
</tr>
<tr>
<td>3</td>
<td>B, Q</td>
</tr>
</tbody>
</table>

(Scenarios will be discussed later)
Abstract Use Cases for weak, transition-targeted coverage are:
Step 3 in method:  
(a) Derive shortest path to each state (preamble)

<table>
<thead>
<tr>
<th>State</th>
<th>Preamble</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

(b) Derive verification path for each state

<table>
<thead>
<tr>
<th>State</th>
<th>Verification path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>(not necessary - why?)</td>
</tr>
</tbody>
</table>
Abstract Use Cases for weak, transition-targeted coverage are:
Step 3 in method: (a) Derive shortest path to each state (preamble)

<table>
<thead>
<tr>
<th>State</th>
<th>Preamble</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>null</td>
</tr>
<tr>
<td>P</td>
<td>gof/DTN</td>
</tr>
<tr>
<td>V</td>
<td>gof/DTN, d/VCE</td>
</tr>
<tr>
<td>B</td>
<td>(external intervention)</td>
</tr>
</tbody>
</table>

(b) Derive verification path for each state

<table>
<thead>
<tr>
<th>State</th>
<th>Verification path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>gof/DTN, gon/NRS</td>
</tr>
<tr>
<td>P</td>
<td>to/ROH, gon/NRS</td>
</tr>
<tr>
<td>V</td>
<td>to/NA, gon/NRS</td>
</tr>
<tr>
<td>B</td>
<td>(not necessary - why?)</td>
</tr>
</tbody>
</table>
Weak, transition-targeted coverage (continued):

Step 4. ii) the event which causes each transition to occur.

Fill in the table below for all transitions.

<table>
<thead>
<tr>
<th>State</th>
<th>State (path)</th>
<th>Event</th>
<th>Verify Response</th>
<th>DVerify Next State (path)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>start</td>
<td>gof</td>
<td>DTN</td>
<td>to, gon</td>
</tr>
</tbody>
</table>

Example:
Weak, transition-targeted coverage (continued):

Step 4. ii) the event which causes each transition to occur. Fill in the table below for all transitions.

<table>
<thead>
<tr>
<th>State</th>
<th>State (path)</th>
<th>Event</th>
<th>Verify Response</th>
<th>DVerify Next State (path)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>P</td>
<td>gof</td>
<td>DTN</td>
<td>to/ROH, gon/NRS</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>to</td>
<td>ROH</td>
<td>gon/NRS</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>d</td>
<td>BSY</td>
<td>to/ROH, gon/NRS</td>
</tr>
<tr>
<td>P</td>
<td>V</td>
<td>d</td>
<td>VCE</td>
<td>to/NA, gon/NRS</td>
</tr>
<tr>
<td>V</td>
<td>Q</td>
<td>gon</td>
<td>NRS</td>
<td>gof/DTDN, gon/NRS</td>
</tr>
<tr>
<td>B</td>
<td>V</td>
<td>gof</td>
<td>VCE</td>
<td>to/NA, gon/NRS</td>
</tr>
<tr>
<td>B</td>
<td>Q</td>
<td>to</td>
<td>NRS</td>
<td>gof/DTDN, gon/NRS</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>gof</td>
<td>NRS</td>
<td>gof/DTDN, gon/NRS</td>
</tr>
</tbody>
</table>
Example:

Abstract Use Cases for Strong (FSM Path) Coverage

Steps 2, 3 (Identify Phases): (fill in the states below)

i) Establish Voice Connection
   (start B, goal V)
   (start , goal )

ii) Terminate Call Session:
    (start , goal )
Abstract use Cases for Strong (FSM Path) Coverage

Steps 2, 3 (Identify Phases): (fill in the states below)

i) Establish Voice Connection
   (start B, goal V)
   (start Q, goal V)

ii) Terminate Call Session:
    (start V, goal Q)
Abstract Use Cases for Strong (FSM Path) Coverage (cont.)

Step 4: (Identify Abstract Use Cases (paths))

a) Identify longest, acyclic (normal behaviour) paths for each phase. (Complete the table below)

<table>
<thead>
<tr>
<th>Phase</th>
<th>#</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>i) B to V</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>i) Q to V</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ii) V to Q</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Abstract Use Cases for Strong (FSM Path) Coverage (cont.)

Step 4: (Identify Abstract Use Cases (paths))

a) Identify longest, acyclic (normal behaviour) paths for each phase. (Complete the table below)

<table>
<thead>
<tr>
<th>Phase</th>
<th>#</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>i) B to V</td>
<td>1</td>
<td>gof/VCE</td>
</tr>
<tr>
<td>i) Q to V</td>
<td>2</td>
<td>gof/DTN, d/VCE</td>
</tr>
<tr>
<td>ii) V to Q</td>
<td>3</td>
<td>gon/NRS</td>
</tr>
</tbody>
</table>

Note: in Q to V phase, we detect an FSM modelling error, namely:
- after $BSY$ or $ROH$, the only applicable event is $gon$.
(how could you correct this? would you? why or why not?)
Example:

Strong Abstract Use Cases (continued)

Step 4 b): For each normal behaviour path, identify exception handling transitions by filling in the table below.

<table>
<thead>
<tr>
<th>Normal Behaviour Path</th>
<th>Exception-handling Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(B, V)</td>
</tr>
<tr>
<td>2</td>
<td>(Q, P, V)</td>
</tr>
<tr>
<td></td>
<td>(Q, P, V)</td>
</tr>
<tr>
<td></td>
<td>(Q, P, V)</td>
</tr>
<tr>
<td></td>
<td>(Q, P, V)</td>
</tr>
<tr>
<td>3</td>
<td>(V, Q)</td>
</tr>
</tbody>
</table>

Step 5: Derive Use Case Scenarios
(to be discussed later)
Answers:

Strong (FSM Path) Coverage (continued)

Step 4 b): For each normal behaviour path, identify exception handling transitions by filling in the table below.

<table>
<thead>
<tr>
<th>Normal Behaviour Path</th>
<th>Exception-handling Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (B, V)</td>
<td>to/NRS</td>
</tr>
<tr>
<td>2 (Q, P, V)</td>
<td>to/ROH</td>
</tr>
<tr>
<td></td>
<td>(Q, P, V)</td>
</tr>
<tr>
<td></td>
<td>(Q, P, V)</td>
</tr>
<tr>
<td>3 (V, Q)</td>
<td>n/a (or is it?)</td>
</tr>
</tbody>
</table>

Step 5: Derive Use Case Scenarios
(to be discussed later)