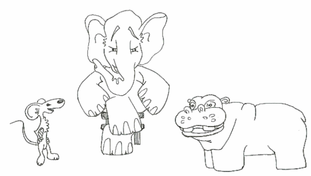
Foundations of Software Testing

Chapter 3: Test Generation: Finite State Models

Aditya P. Mathur
Purdue University



Last update: September 3, 2007

Learning Objectives

- What are Finite State Models?
- The W method for test generation
- The Wp method for test generation

Where are FSMs used?

- Conformance testing of communications protocols--this is where it all started.
- Testing of any system/subsystem modeled as a finite state machine, e.g. elevator designs, automobile components, nuclear plant protection systems, steam boiler control, etc.
- Finite state machines are widely used in modeling of all kinds of systems. Generation of tests from FSM specifications assists in testing the conformance of implementations to the corresponding FSM model.

What is a Finite State Machine? Quick review

- A finite state machine (FSM) is an abstract representation of behavior exhibited by some systems.
- An FSM is derived from application requirements, e.g., a network protocol could be modeled using an FSM.
- Not all aspects of an application's requirements can be specified by an FSM: real time requirements, performance requirements, and computational requirements.

FSM (Mealy machine, 1955): Definition

An FSM (Mealy) is a 6-tuple: (X, Y, Q, q_0, δ, O), where:

- X is a finite set of input symbols also known as the input alphabet.
- Y is a finite set of output symbols also known as the output alphabet,
- Q is a finite set of states,
- q_0 in Q is the initial state,
- δ: Q x X→ Q is a next-state or state transition function, and
- O: Q x X→ Y is an output function

FSM (Moore machine, 1956): Definition

An FSM (Moore) is a 7-tuple: (X, Y, Q, q_0, δ, O, F), where:

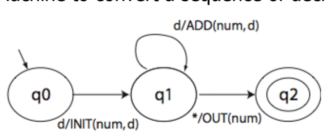
X , Y, Q, q_0, and δ are the same as in FSM (Mealy)

O: Q → Y is an output function

F∈Q is the set of final or accepting or terminating states.

State Diagram Representation of FSM

Nodes: States; Labeled Edges: Transitions
Example: Machine to convert a sequence of decimal digits to an integer



(a) Notice ADD, INIT, ADD,OUT actions.

(b) INIT: Initialize num. ADD: Add to num. OUT: Output num.

Tabular representation of FSM

The table given below shows how to represent functions δ and O for the DIGDEC machine.

Current state	Action		Next state	
	d	*	d	*
q_0	INIT (num, d)		q_1	
q_1	ADD (num, d)	OUT (num)	q_1	q_2
q_2				

Properties of FSM

Completely specified: An FSM M is said to be completely specified if from each state in M there exists a transition for each input symbol.

Strongly connected: An FSM M is considered strongly connected if for each pair of states (q_i q_j) there exists an input sequence that takes M from state q_i to q_j.

Properties of FSM: Equivalence

V-equivalence: Let $M_1=(X, Y, Q_1, m^1_0, T_1, O_1)$ and $M_2=(X, Y, Q_2, m^2_0, T_2, O_2)$ be two FSMs. Let V denote a set of non-empty strings over the input alphabet X i.e. $V \subseteq X^+$.

Let q_i and q_j, be two states of machines M_1 and M_2, respectively. q_i and q_j are considered V-equivalent if $O_1(q_i, s)=O_2(q_j, s)$ for all s in V.

Properties of FSM: Distinguishability

States q_i and q_j are said to be equivalent if $O_1(q_i, r)=O_2(q_j, r)$ for any set V.

If q_i and q_j are not equivalent then they are said to be distinguishable.

Properties of FSM: k-equivalence

k-equivalence: Let $M_1=(X, Y, Q_1, m^1_0, T_1, O_1)$ and $M_2=(X, Y, Q_2, m^2_0, T_2, O_2)$ be two FSMs.

States $q_i \varepsilon Q_1$ and $q_j \varepsilon Q_2$ are considered k-equivalent if, when excited by any input of length k, yield identical output sequences.

States that are not k-equivalent are considered k-distinguishable.

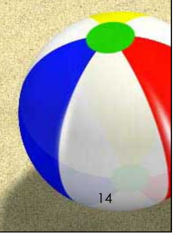
Properties of FSM: Machine Equivalence

Machine equivalence: Machines M_1 and M_2 are said to be equivalent if (a) for each state σ in M1 there exists a state σ' in M_2 such that σ and σ' are equivalent and (b) for each state σ in M_2 there exists a state σ' in M_1 such that σ and σ' are equivalent.

Machines that are not equivalent are considered distinguishable.

Minimal machine: An FSM M is considered minimal if the number of states in M is less than or equal to any other FSM equivalent to M.

Faults Targeted



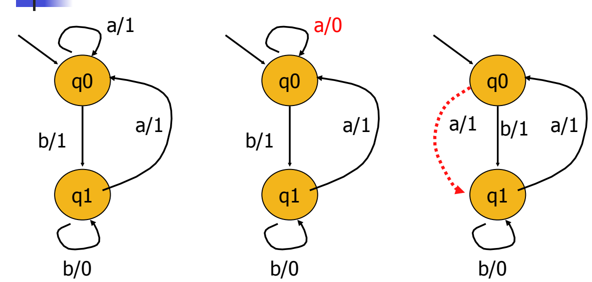
Faults in implementation

An FSM serves to specify the correct requirement or design of an application.

The tests generated from an FSM target faults related to the FSM itself.

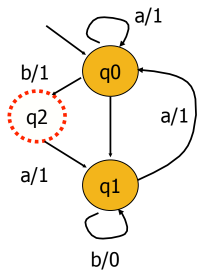
What faults are targeted by the tests generated using an FSM?

Fault model

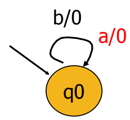


Correct design | Operation error | Transfer error

Fault model (contd.)



Extra state error



Missing state error

Test generation using W method

Assumptions for test generation

1. M is Completely specified, minimal, connected, and deterministic.
2. M starts in a fixed initial state.
3. M and IUT (Implementation Under Test) have the same input alphabet.

Characterization Set of M – W-set

Characterization set for machine M, denoted as W, is a finite set of input sequences that distinguish the behavior of any pair of states in M.

--- W is constructed from the k-equivalence partitions of M.

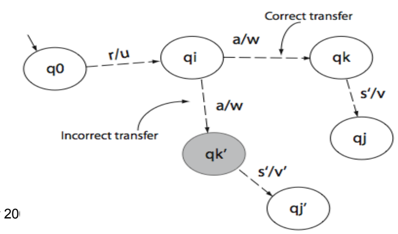
Transition Cover Set of M – P-set

Transition cover set for machine M, denoted as P, is a finite set of input sequences such that exciting M with all elements of P ensures that all states are reached and all transitions are traversed at least once.

--- P is constructed from the testing tree of M.

Error detection process

- Each test case t is of the form r.s where r is in P and s in W.
- r moves the application from initial state q0 to state qj. Then, s=as' takes it from qi to state qj or qj'.



Overall algorithm used in W-method

Step 1: Estimate m, the number of states in the correct implementation of the given FSM M.

Step 2: Construct the characterization set W for M.

Step 3: Construct the testing tree for M and generate the transition cover set P from the testing tree.

Step 4: Construct set Z from W and m.

Step 5: Desired test set=P.Z

Step 1: Estimation of m

Initially we will assume that m = n, where
n is the number of states in FSM M
m is the number of states in correct implementation

Later we discuss how to handle cases where m > n

Step 2: Construction of W

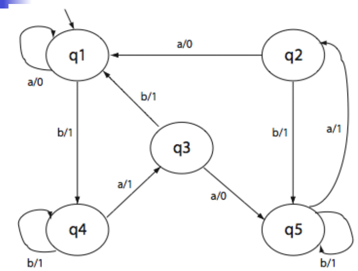
Let M=(X, Y, Q, q1, δ, O) be a minimal and complete FSM.

W is a finite set of input sequences that distinguish the behavior of any pair of states in M. Each input sequence in W is of finite length.

Given states qi and qj in Q, W contains a string s such that:

O(qi, s)≠O(qj, s)

Example of W



W={baaa,aa,aaa}

O(baaa,q1)=1101

O(baaa,q2)=1100

Thus baaa distinguishes state q1 from q2 as O(baaa,q1) ≠ O(baaa,q2)

Steps in the construction of W

Step 1: Construct a sequence of k-equivalence partitions of Q denoted as P1, P2, …Pm, m>0.

Step 2: Traverse the k-equivalence partitions in reverse order to obtain distinguishing sequence for each pair of states.

What is a k-equivalence partition of Q?

A k-equivalence partition of Q, denoted as P_k, is a collection of n finite sets Σ_{k1}, Σ_{k2} … Σ_{kn} such that

$$\cup^{n}_{i=1} \Sigma_{ki} = Q$$

States in Σ_{ki} are k-equivalent.

If state u is in Σ_{ki} and v in Σ_{kj} for i≠j, then u and v are k-distinguishable.

How to construct a k-equivalence partition?

Given an FSM M, construct a 1-equivalence partition, start with a tabular representation of M.

Current state	Output		Next state	
	a	b	a	b
q1	0	1	q1	q4
q2	0	1	q1	q5
q3	0	1	q5	q1
q4	1	1	q3	q4
q5	1	1	q2	q5

Construct 1-equivalence partition

Group states identical in their Output entries. This gives us 1-partition P_1 consisting of Σ_1={q1, q2, q3} and Σ_2 ={q4, q5}.

Σ	Current state	Output		Next state	
		a	b	a	b
1	q1	0	1	q1	q4
	q2	0	1	q1	q5
	q3	0	1	q5	q1
2	q4	1	1	q3	q4
	q5	1	1	q2	q5

Construct 2-equivalence partition: Rewrite P_1 table

Rewrite P_1 table. Remove the output columns. Replace a state entry q_i by q_{ij} where j is the group number in which state q_i lies.

Σ	Current state	Next state	
		a	b
1	q1	q11	q42
	q2	q11	q52
	q3	q52	q11
2	q4	q31	q42
	q5	q21	q52

P_1 Table

Group number

Construct 2-equivalence partition: Construct P_2 table

Group all entries with identical second subscripts under the next state column. This gives us the P_2 table. Note the change in second subscripts.

Σ	Current state	Next state	
		a	b
1	q1	q11	q43
	q2	q11	q53
2	q3	q53	q11
3	q4	q32	q43
	q5	q21	q53

P_2 Table

Construct 3-equivalence partition: Construct P_3 table

Group all entries with identical second subscripts under the next state column. This gives us the P_3 table. Note the change in second subscripts.

P_3 Table

Σ	Current state	Next state a	Next state b
1	q1	q11	q43
	q2	q11	q54
2	q3	q54	q11
3	q4	q32	q43
4	q5	q21	q54

Construct 4-equivalence partition: Construct P_4 table

Continuing with regrouping and relabeling, we finally arrive at P_4 table.

P_4 Table

Σ	Current state	Next state a	Next state b
1	q1	q11	q44
2	q2	q11	q55
3	q3	q55	q11
4	q4	q33	q44
5	q5	q22	q55

k-equivalence partition: Convergence

The process is guaranteed to converge.

When the process converges, and the machine is minimal, each state will be in a separate group.

The next step is to obtain the distinguishing strings for each state.

Finding the distinguishing sequences: Example

Let us find a distinguishing sequence for states q1 and q2.

Find tables P_i and P_{i+1} such that (q1, q2) are in the same group in P_i and different groups in P_{i+1}. We get P_3 and P_4.

Initialize $z=\varepsilon$. Find the input symbol that distinguishes q1 and q2 in table P3. This symbol is b. We update z to z.b. Hence z now becomes b.

Finding the distinguishing sequences: Example (contd.)

The next states for q1 and q2 on b are, respectively, q4 and q5.

We move to the P_2 table and find the input symbol that distinguishes q4 and q5. Let us select a as the distinguishing symbol. Update z which now becomes ba.

The next states for states q4 and q5 on symbol a are, respectively, q3 and q2. These two states are distinguished in P_1 by a and b. Let us select a. We update z to baa.

Finding the distinguishing sequences: Example (contd.)

The next states for q3 and q2 on a are, respectively, q1 and q5.

Moving to the original state transition table we obtain a as the distinguishing symbol for q1 and q5

We update z to baaa. This is the farthest we can go backwards through the various tables. baaa is the desired distinguishing sequence for states q1 and q2. Check that o(q1,baaa)≠o(q2,baaa).

Finding the distinguishing sequences: Example (contd.)

Using the same procedure used for q1 and q2, we can find the distinguishing sequence for each pair of states. This leads us to the following characterization set for our FSM.

W={a, aa, aaa, baaa}

W-method: where are we?

Step 1: Estimate the maximum number of states (m) in the correct implementation of the given FSM M.

Step 2: Construct the characterization set W for M.

Step 3: Construct the testing tree for M and generate the transition cover set P from the testing tree. ⟵

Step 4: Construct set Z from W and m.

Step 5: Desired test set=P.Z

Step 3: Construct the testing tree for M

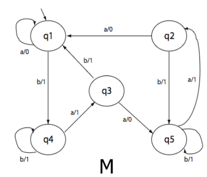
A testing tree of an FSM is a tree rooted at the initial state. It contains at least one path from the initial state to the remaining states in the FSM.

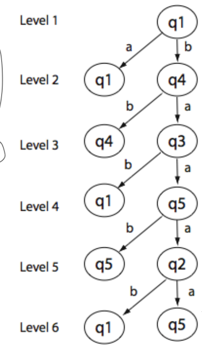
Construction:
State q0, the initial state, is the root of the testing tree. Assuming that the testing tree has been constructed until level k, the (k+1)th level is built as follows.

Select a node n at level k.

- If n appears at any level from 1 through k-1, then *n is a leaf node* and is not expanded any further.
- If n is not a leaf node then we expand it by adding a branch from node n to a new node m if δ(n, x)=m for each x in X . This branch is labeled as x. This step is repeated for all nodes at level k.

Example: Construct the testing tree for M





Start here, initial state is the root.

q1 becomes leaf, q4 can be expanded.

.

.

.

No further expansion possible

W-method: where are we?

Step 1: Estimate the maximum number of states (m) in the correct implementation of the given FSM M.

Step 2: Construct the characterization set W for M.

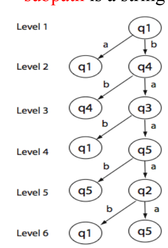
Step 3: (a) Construct the testing tree for M and (b) generate the transition cover set P from the testing tree. ⟵

Step 4: Construct set Z from W and m.

Step 5: Desired test set=P.Z

Step 3: (b) Find the transition cover set from the testing tree

A transition cover set P is a set of all strings representing subpaths, starting at the root, in the testing tree. Concatenation of the labels along the edges of a subpath is a string that belongs to P. The empty string (ε) also belongs to P.



P = {ε, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa}

W-method: where are we?

Step 1: Estimate the maximum number of states (m) in the correct implementation of the given FSM M.

Step 2: Construct the characterization set W for M.

Step 3: Construct the testing tree for M and generate the transition cover set P from the testing tree.

Step 4: Construct set Z from W and m. ⟵

Step 5: Desired test set=P.Z

Step 4: Construct set Z from W and m

Given that X is the input alphabet and W the characterization set, we have:

$Z = X^0.W \cup X^1.W \cup \ldots.. X^{m-1-n}.W \cup X^{m-n}.W$

For m=n=5, we get

$Z = X^0.W=W$

For X={a, b}, W={a, aa, aaa, baaa}, m=6, n=5

$Z = W \cup X^1.W$ = {a, aa, aaa, baaa} ∪ {a, b}.{a, aa, aaa, baaa}
= {a, aa, aaa, baaa, <u>aa</u>, <u>aaa</u>, aaaa, abaaa, ba, baa, <u>baaa</u>, bbaaa}

W-method: where are we?

Step 1: Estimate the maximum number of states (m) in the correct implementation of the given FSM M.

Step 2: Construct the characterization set W for M.

Step 3: (a) Construct the testing tree for M and (b) generate the transition cover set P from the testing tree.

Step 4: Construct set Z from W and m.

Step 5: Desired test set=P.Z ⟵ Next

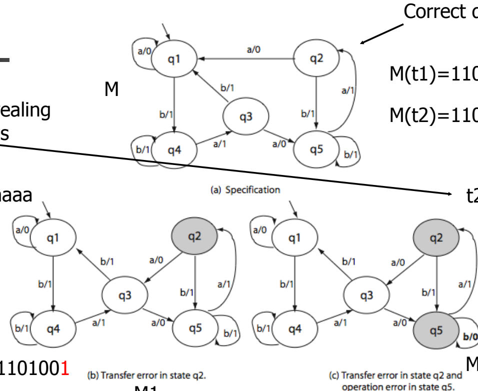
Step 5: Desired test set=P.Z

The test inputs based on the given FSM M can now be derived as:
T=P.Z

Do the following to test the implementation:

1. Find the expected response to each element of T.
2. Generate test cases for the application. Note that even though the application is modeled by M, there might be variables to be set before it can be exercised with elements of T.
3. Execute the application and check if the response matches. Reset the application to the initial state after each test.

Example 1: Testing an erroneous application



Correct design

M

M(t1)=1101000

M(t2)=11011

Error-revealing test cases

t1=baaaaaa

t2=baaba

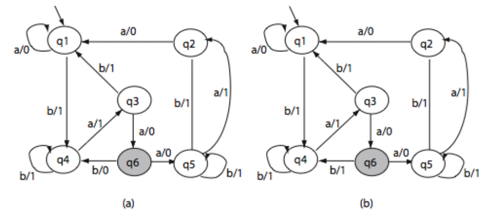
M1(t1)=1101001

M2(t2)=11001

M1

M2

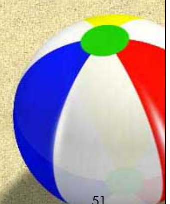
Example 2: Extra state. n=5, m=6.



M1 M2

t1=baaba	M(t1)=11011	M1(t1)=11001
t2=baaa	M(t2)=1101	M2(t2)=1100

The Partial W (Wp) method



The partial W (Wp) method

Tests are generated from minimal, complete, and connected FSM.

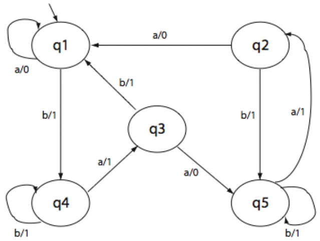
Size of tests generated is *generally smaller* than that generated using the W-method.

Test generation process is divided into two phases:
Phase 1: Generate a test set using the state cover set (S) and the characterization set (W).
Phase 2: Generate additional tests using a subset of the transition cover set and state identification sets.

What is a state cover set? A state identification set?

State cover set

Given FSM M with input alphabet X, a state cover set S is a finite non-empty set of strings over X* such that, for each state qi in Q, there is a string in S that takes M from its initial state to qi.



S={ε, b, ba, baa, baaa}

S is always a subset of the transition cover set P.

S is not necessarily unique.

State identification set

Given an FSM M with Q as the set of states, an identification set W_i for state qi∈Q has the following properties:

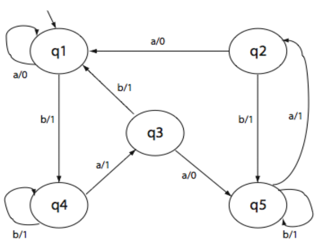
(a) $W_i \subseteq W$, $1\leq i\leq n$ [Identification set is a subset of W.]

(b) O(qi, s)≠ O(qj, s) , for $1\leq j\leq n$, $j\neq i$, $s\in W_i$
[For each state other than qi, there is a string in Wi that distinguishes qi from qj.]

(c) No subset of W_i satisfies property (b). [W_i is minimal.]

State identification set: Example

Last element of the output string



$W_1=W_2$={baaa, aa, a}
W_3={a, aa} $W_4=W_5$={a, aaa}

Si	Sj	x	O(Si,x)	O(Sj,x)
1	2	baaa	1	0
	3	aa	0	1
	4	a	0	1
	5	a	0	1
2	3	aa	0	1
	4	a	0	1
	5	a	0	1
3	4	a	0	1
	5	a	0	1
4	5	aaa	1	0

Wp method: Example:
Step 1: Compute S, P, W, Wi, $\mathcal{W}$

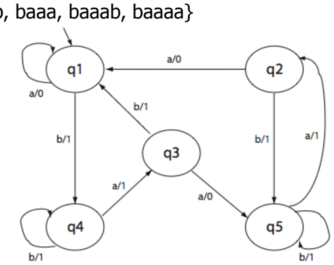
S={ε, b, ba, baa, baaa}

P={ε, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa}

$W_1=W_2$={baaa, aa, a}
W_3={a, aa} $W_4=W_5$={a, aaa}

W={a, aa, aaa, baaa}

$\mathcal{W}$={W1, W2, W3, W4, W5}



Wp method: Example: Step 2: Compute T1 [m=n]

T1 = S.W = {ε, b, ba, baa, baaa}.{a, aa, aaa, baaa}

Elements of T1 ensure that the

- each state of the FSM is covered and
- distinguished from the remaining states.

Wp method: Example: Step 3: Compute R and δ (we assume m=n)

R = P - S = {ε, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa}
- {ε, b, ba, baa, baaa}
= {a, bb, bab, baab, baaab, baaaa}

Let each element of R be denoted as $r_{i1}, r_{i2}, \ldots, r_{ik}$

And let $\delta(q_0, r_{ij}) = q_{ij}$

Wp method: Example: Step 4: Compute T2 [m=n]

$T2 = R \otimes \mathcal{W} = \cup^{k}_{j=1} (\{r_{ij}\}.W_{ij})$,
where W_{ij} is the identification set for state q_{ij}.

δ(q1, a)=q1 δ(q1, bb)=q4 δ(q1, bab)=q1

δ(q1, baab)=q5 δ(q1, baaab)=q5 δ(q1, baaaa)=q1

$T2 = (\{a\}.W_1) \cup (\{bb\}.W_4) \cup (\{bab\}.W_1) \cup (\{baab\}.W_5) \cup (\{baaab\}.W_5) \cup (\{baaaa\}.W_1)$
= {abaaa, aaa, aa} ∪ {bba, bbaaa} ∪ {babbaaa, babaa, baba} ∪
{baaba, baabaaa} ∪ {baaaba, baaabaaa} ∪
{baaaabaaa, baaaaaa, baaaaa}

Wp method: Example: Savings

Test set size using the W method = 44

Test set size using the Wp method = 34 (20 from T1 + 14 from T2)

Testing using the Wp method

Testing proceeds in two phases.

Tests from T1 are applied in phase 1.
Tests from T2 are applied in phase 2.

While tests from phase 1 ensure state coverage, they do not ensure all transition coverage.

Wp method when $m > n$

Sets T1 and T2 are computed a bit differently, as follows:

T1 = S.X[m-n].W, where X[m-n] is the set union of X^i , $1 \leq i \leq (m-n)$

T2 = R.X[m-n] $\otimes \mathcal{W}$

Summary

Behavior of a large variety of applications can be modeled using finite state machines (FSM), e.g. GUIs.

The W and the Wp methods are automata theoretic methods to generate tests from a given FSM model.

Tests so generated are guaranteed to detect all operation errors, transfer errors, and missing/extra state errors in the implementation given that the FSM representing the implementation is complete, connected, and minimal.

The size of tests sets generated by the W method is larger than that generated by the Wp method. However, their fault detection effectiveness are the same.