

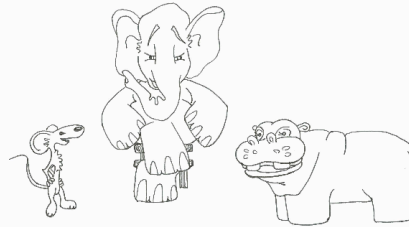
Foundations of Software Testing

Chapter 3: Test Generation: Finite State Models

Aditya P. Mathur
Purdue University



These slides are copyrighted. They are for use with the **Foundations of Software Testing** book by Aditya Mathur. Please use the slides but do not remove the copyright notice.



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 (locks, transmission, stepper motors, etc), 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.



Where are FSMs used?

Modeling GUIs, network protocols, pacemakers, Teller machines, WEB applications, safety software modeling in nuclear plants, and many more.

While the FSM's considered in examples are abstract machines, they are abstractions of many real-life machines.



What is an Finite State Machine?

Quick review

- A finite state machine, abbreviated as FSM, is an abstract representation of behavior exhibited by some systems.
- An FSM is derived from application requirements. For example, a network protocol could be modeled using an FSM.
- Not all aspects of an application's requirements are specified by an FSM. Real time requirements, performance requirements, and several types of computational requirements cannot be specified by an FSM.

© Aditya P. Mathur 2007

5



FSM (Mealy machine, 1955): Definition

An FSM (Mealy) is a 6-tuple: $(X, Y, Q, q_0, \delta, O)$, 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 **states**,

q_0 in Q is the **initial state**,

$\delta: Q \times X \rightarrow Q$ is a next-state or **state transition function**, and

$O: Q \times X \rightarrow Y$ is an **output function**

© Aditya P. Mathur 2007

6

FSM (Moore machine, 1956): Definition

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

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

$O: Q \rightarrow Y$ is an **output function**

$F \subseteq Q$ is the set of final or accepting or terminating states.

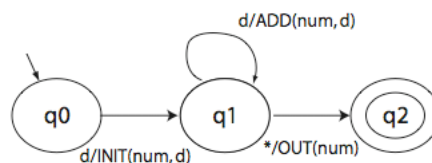
© Aditya P. Mathur 2007

7

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.

© Aditya P. Mathur 2007

8



Tabular representation of FSM

A table is often used as an alternative to the state diagram to represent the state transition function δ and the output function O .

The table consists of two sub-tables that consist of one or more columns each. The leftmost sub-table is the output or the **action** sub-table. The rows are labeled by the states of the FSM. The rightmost sub-table is the **next state** sub-table.

© Aditya P. Mathur 2007

9



Tabular representation of FSM: Example

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				

© Aditya P. Mathur 2007

10



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

Stated differently, states q_i and q_j are considered V-equivalent if M_1 and M_2 , when excited with an element of V in states q_i and q_j , respectively, yield identical output sequences.

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**. This definition of equivalence also applies to states within a machine. Thus machines M_1 and M_2 could be the same machine.



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



Properties of FSM: k-equivalence (contd.)

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

Once again, M_1 and M_2 may be the same machines implying that k-distinguishability applies to any pair of states of an FSM.

It is also easy to see that if two states are **k-distinguishable** for any $k > 0$ then they are also distinguishable for any $n \geq k$. If M_1 and M_2 are not k-distinguishable then they are said to be **k-equivalent**.

© Aditya P. Mathur 2007

15



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 M_1 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.

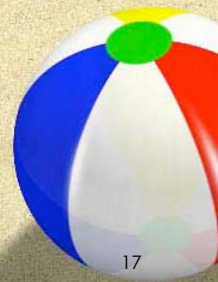
Machines that are not equivalent are considered distinguishable.

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 .

© Aditya P. Mathur 2007

16

Faults Targeted



© Aditya P. Mathur 2007

17



Faults in implementation

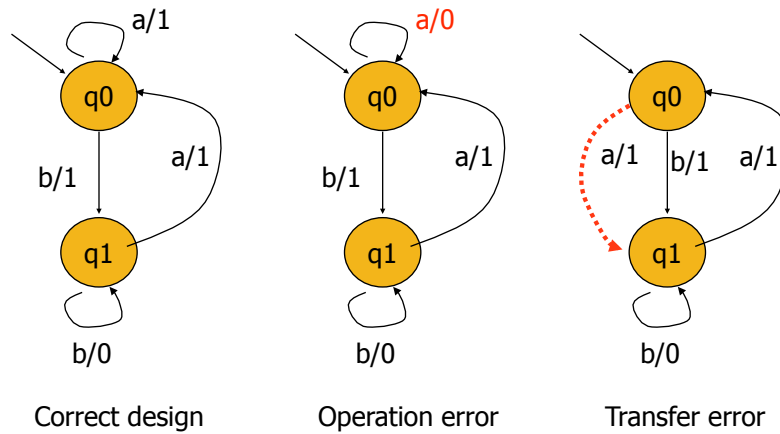
An FSM serves to specify the correct requirement or design of an application. Hence tests generated from an FSM target faults related to the FSM itself.

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

© Aditya P. Mathur 2007

18

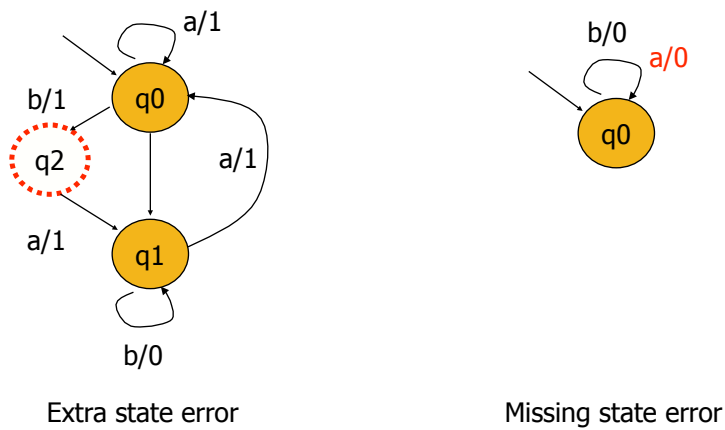
Fault model



© Aditya P. Mathur 2007

19

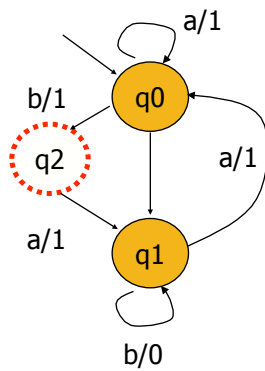
Fault model (contd.)



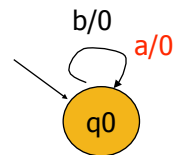
© Aditya P. Mathur 2007

20

Fault model (contd.)



Extra state error



Missing state error

© Aditya P. Mathur 2007

21

Test generation using Chow's method

© Aditya P. Mathur 2007

22



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.



Overall algorithm used in Chow's method

- 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 1: Estimation of m

This is based on a knowledge of the implementation.
In the absence of any such knowledge,
let m = number of states in M .



Step 2: Construction of W . What is W ?

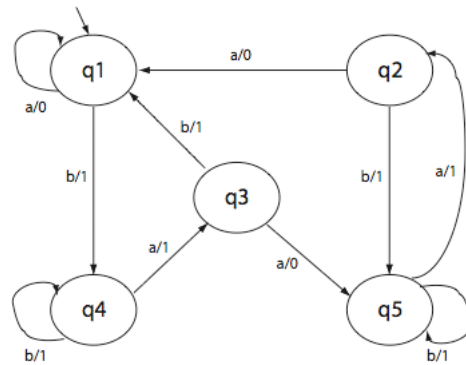
Let $M = (X, Y, Q, q_1, \delta, O)$ be a minimal and complete FSM.

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 q_i and q_j in Q , W contains a string s such that:

$$O(q_i, s) \neq O(q_j, s)$$

Example of W



$W = \{baaa, aa, aaa\}$

$O(baaa, q1) = 1101$

$O(baaa, q2) = 1100$

Thus **baaa** distinguishes state $q1$ from $q2$ as $O(baaa, q1) \neq O(baaa, q2)$

© Aditya P. Mathur 2007

27

Steps in the construction of W

Step 1: Construct a sequence of k -equivalence partitions of Q denoted as P_1, P_2, \dots, P_m , $m > 0$.

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

© Aditya P. Mathur 2007

28



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 $\Sigma_{k1}, \Sigma_{k2} \dots \Sigma_{kn}$ such that

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

States in Σ_{ki} are k-equivalent.

If state u is in Σ_{ki} and v in Σ_{kj} for $i \neq j$, 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 $\Sigma_1 = \{q1, q2, q3\}$ and $\Sigma_2 = \{q4, q5\}$.

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

© Aditya P. Mathur 2007

31



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

© Aditya P. Mathur 2007

32



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

© Aditya P. Mathur 2007

33



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.

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

P_3 Table

© Aditya P. Mathur 2007

34



Construct 4-equivalence partition: Construct P_4 table

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

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

P_4 Table



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 q_1 and q_2 .

Find tables P_i and P_{i+1} such that (q_1, q_2) are in the same group in P_i and different groups in P_{i+1} . We get P_3 and P_4 .

Initialize $z = \epsilon$. Find the input symbol that distinguishes q_1 and q_2 in table P_3 . This symbol is b . We update z to $z.b$. Hence z now becomes b .

© Aditya P. Mathur 2007

37



Finding the distinguishing sequences: Example (contd.)

The next states for q_1 and q_2 on b are, respectively, q_4 and q_5 .

We move to the P_2 table and find the input symbol that distinguishes q_4 and q_5 .

Let us select a as the distinguishing symbol. Update z which now becomes ba .

The next states for states q_4 and q_5 on symbol a are, respectively, q_3 and q_2 . These two states are distinguished in P_1 by a and b . Let us select a . We update z to baa .

© Aditya P. Mathur 2007

38



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 $o(q1,baaa) \neq o(q2,baaa)$.

© Aditya P. Mathur 2007

39



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.

$$W = \{a, aa, aaa, baaa\}$$

© Aditya P. Mathur 2007

40



Chow's method: where are we?

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

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. ← Next (a)

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

Step 5: Desired **test set=P.Z**

© Aditya P. Mathur 2007

41



Step 3: (a) Construct the **testing tree** for M

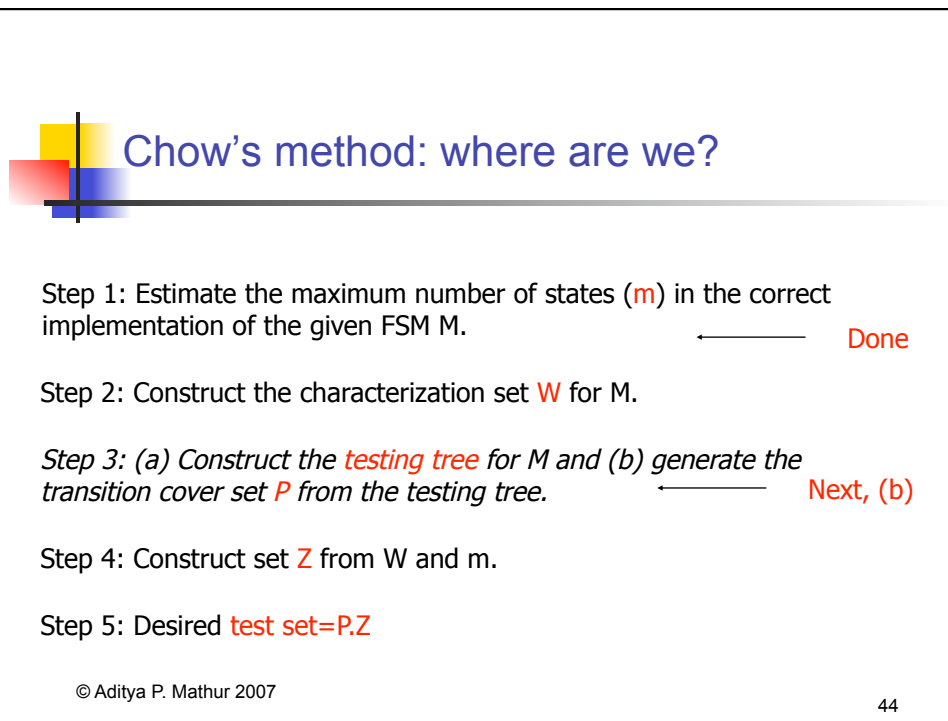
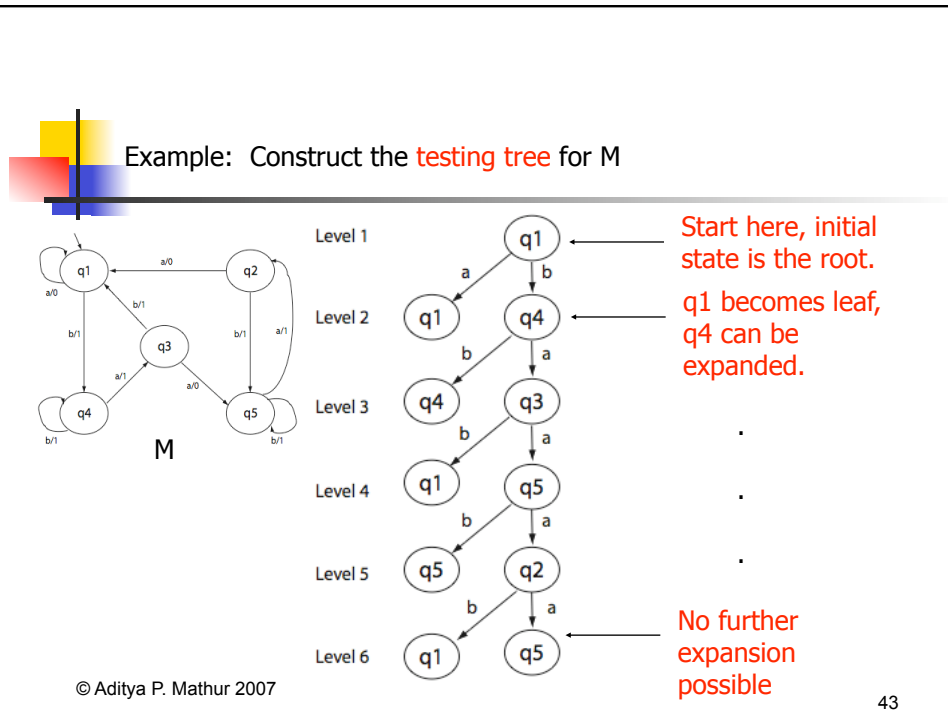
A **testing tree** of an FSM is a tree rooted at the initial state. It contains at least one path from the initial state to the remaining states in the FSM. Here is how we construct the testing tree.

State q_0 , the initial state, is the **root** of the testing tree. Suppose 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 $\delta(n, x)=m$ for each x in X . This branch is labeled as x . This step is repeated for all nodes at level k .

© Aditya P. Mathur 2007

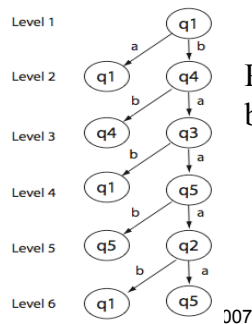
42





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

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



$P = \{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\}$

007

45



Chow's method: where are we?

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

← Done

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.

← Done

Step 4: Construct set Z from W and m .

← Next

Step 5: Desired **test set** $= P.Z$

© Aditya P. Mathur 2007

46



Step 4: Construct set Z from W and m

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

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

For $m=n$, we get

$$Z = X^0.W = W$$

For $X=\{a, b\}$, $W=\{a, aa, aaa, baaa\}$, $m=6$

$$\begin{aligned} Z &= W \cup X^1.W = \{a, aa, aaa, baaa\} \cup \{a, b\} \cdot \{a, aa, aaa, baaa\} \\ &= \{a, aa, aaa, baaa, \underline{aa}, \underline{aaa}, aaaa, abaaa, ba, baa, \underline{baaa}, \\ &\quad \underline{bbaaa}\} \end{aligned}$$

© Aditya P. Mathur 2007

47



Chow's method: where are we?

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

← Done

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.

← Done

Step 4: Construct set Z from W and m .

← Done

Step 5: Desired test set = $P.Z$

← Next

© Aditya P. Mathur 2007

48



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.

© Aditya P. Mathur 2007

49

Example 1: Testing an erroneous application



Error-revealing
test cases

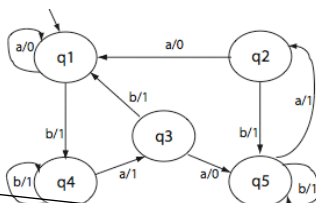
t1=baaaaaa

M1(t1)=110100**1**

© Aditya P. Mathur 2007

M

(a) Specification



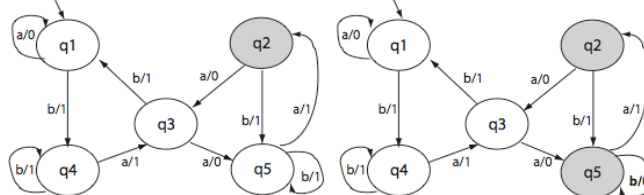
Correct design

M(t1)=1101000

M(t2)=110**1**1

(a) Specification

t2=baaba



(b) Transfer error in state q2.

(c) Transfer error in state q2 and operation error in state q5.

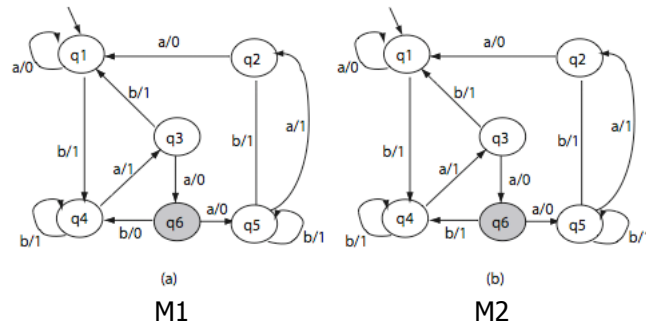
M1

M2

M2(t2)=1100**1**

50

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



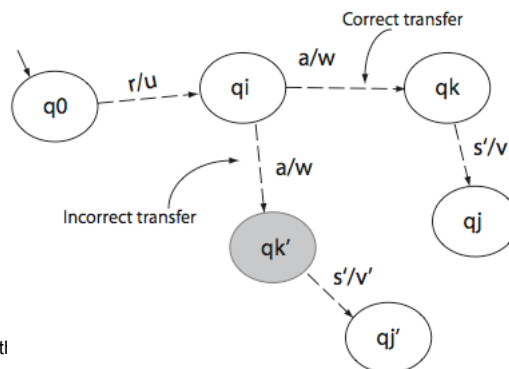
$t_1 = \text{baaba}$ $M(t_1) = 11011$ $M_1(t_1) = 11001$
 $t_2 = \text{baaa}$ $M(t_2) = 1101$ $M_2(t_2) = 1100$

© Aditya P. Mathur 2007

51

Error detection process

Given $m=n$, 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 q_0 to state q_i . Then, $s=as'$ takes it from q_i to state q_j or q_j' .



© Aditya P. Matl

52

The Partial W (Wp) method

© Aditya P. Mathur 2007

53



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?

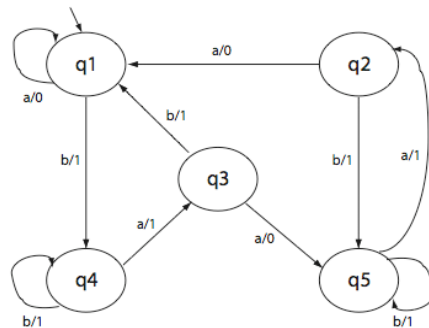
© Aditya P. Mathur 2007

54



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 q_i in Q , there is a string in S that takes M from its initial state to q_i .



$S = \{\epsilon, b, ba, baa, baaa\}$

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

S is not necessarily unique.

55



State identification set

Given an FSM M with Q as the set of states, an identification set W_i for state $q_i \in Q$ has the following properties:

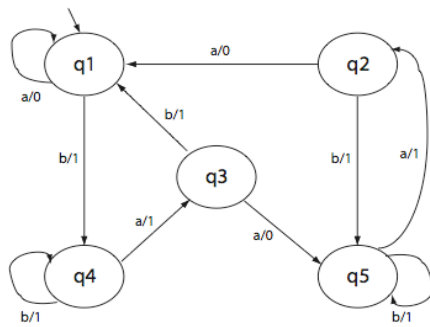
- (a) $W_i \subseteq W$, $1 \leq i \leq n$ [Identification set is a subset of W .]
- (b) $O(q_i, s) \neq O(q_j, s)$, for $1 \leq j \leq n$, $j \neq i$, $s \in W_i$
[For each state other than q_i , there is a string in W_i that distinguishes q_i from q_j .]
- (c) No subset of W_i satisfies property (b). [W_i is minimal.]

© Aditya P. Mathur 2007

56

State identification set: Example

Last element of the output string



$W_1=W_2=\{\text{baaa, aa, a}\}$

$W_3=\{\text{a, aa}\}$ $W_4=W_5=\{\text{a, aaa}\}$

© Aditya P. Mathur 2007

S_i	S_j	x	$O(S_i, x)$	$O(S_j, x)$
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

57

Wp method: Example:

Step 1: Compute S , P , W , W_i , \mathcal{W}

$S=\{\epsilon, b, ba, baa, baaa\}$

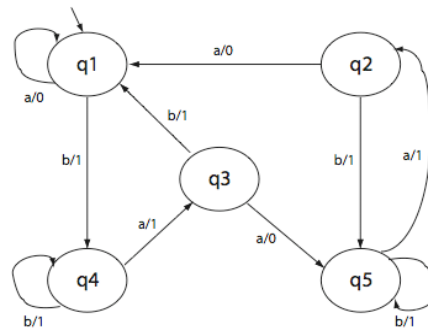
$P=\{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\}$

$W_1=W_2=\{\text{baaa, aa, a}\}$

$W_3=\{\text{a, aa}\}$ $W_4=W_5=\{\text{a, aaa}\}$

$W=\{\text{a, aa, aaa, baaa}\}$

$\mathcal{W}=\{W_1, W_2, W_3, W_4, W_5\}$



© Aditya P. Mathur 2007

58



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

$$T1 = S.W = \{\epsilon, b, ba, baa, baaa\} \cdot \{a, aa, aaa, baaa\}$$

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)

$$\begin{aligned} R = P - S &= \{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\} \\ &\quad - \{\epsilon, b, ba, baa, baaa\} \\ &= \{a, bb, bab, baab, baaab, baaaa\} \end{aligned}$$

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

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



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

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

$\delta(q1, a) = q1$

$\delta(q1, bb) = q4$

$\delta(q1, bab) = q1$

$\delta(q1, baab) = q5$

$\delta(q1, baaab) = q5$

$\delta(q1, baaaa) = q1$

$$\begin{aligned} T2 &= (\{a\} \cdot W_1) \cup (\{bb\} \cdot W_4) \cup (\{bab\} \cdot W_1) \cup (\{baab\} \cdot W_5) \cup \\ &\quad (\{baaab\} \cdot W_5) \cup (\{baaaa\} \cdot W_1) \\ &= \{abaaa, aaa, aa\} \cup \{bba, bbbaa\} \cup \{babbaaa, babaa, baba\} \cup \\ &\quad \{baaba, baabaaa\} \cup \{baaaba, baaabaaa\} \cup \\ &\quad \{baaaabaaa, baaaaaa, baaaaa\} \end{aligned}$$

© Aditya P. Mathur 2007

61



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)

© Aditya P. Mathur 2007

62



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 , $1 \leq i \leq (m-n)$

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



Summary

Behavior of a large variety of applications can be modeled using finite state machines (FSM). GUIs can also be modeled using FSMs

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. *What happens if it is not?*

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