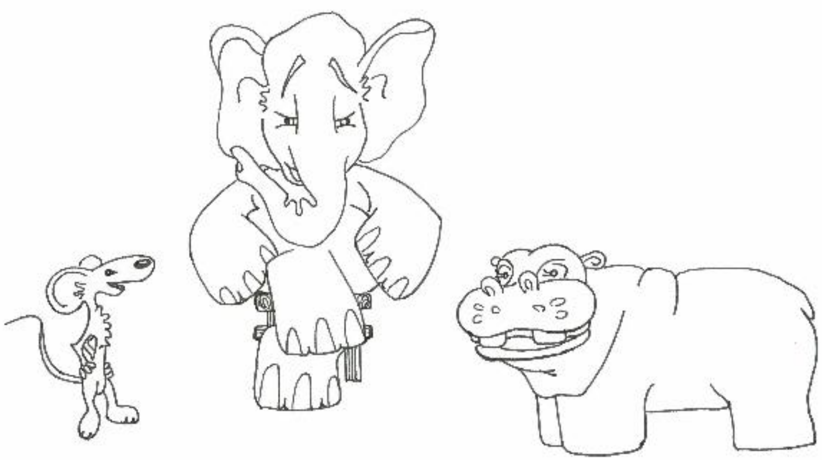
Foundations of Software Testing

Slides based on: Draft V1.0 August 17, 2005

Test Generation: Finite State Models

Aditya P. Mathur

Purdue University



W and Wp methods

Last update: September 14, 2005

Learning Objectives

- What are Finite State Models?
- The W method for test generation
- The Wp method for test generation
- Automata theoretic versus control-flow based test generation

UIO method is not covered in these slides. It is left for the students to read on their own (Section 5.6).

Where are these methods 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.

Warning: It will be a mistake to assume that the test generation methods described here are applicable only to protocol testing!

Strings and languages

Strings

Strings play an important role in testing. A string serves as a test input. Examples: 1011; AaBc; “Hello world”.

A collection of strings also forms a language. For example, a set of all strings consisting of zeros and ones is the language of binary numbers. In this section we provide a brief introduction to strings and languages.

Alphabet

A collection of symbols is known as an alphabet. We use an upper case letter such as X and Y to denote alphabets.

Though alphabets can be infinite, we are concerned only with finite alphabets. For example, X={0, 1} is an alphabet consisting of two symbols 0 and 1. Another alphabet is Y={dog, cat, horse, lion}that consists of four symbols ``dog'', ``cat'', ``horse'', and ``lion''.

Strings over an Alphabet

A string over an alphabet X is any sequence of zero or more symbols that belong to X. For example, 0110 is a string over the alphabet {0, 1}. Also, dog cat dog dog lion is a string over the alphabet {dog, cat, horse, lion}.

We will use lower case letters such as p, q, r to denote strings. The length of a string is the number of symbols in that string. Given a string s, we denote its length by |s|. Thus |1011|=4 and |dog cat dog|=3. A string of length 0, also known as an empty string, is denoted by ε.

Note that ε denotes an empty string and also stands for "element of" when used with sets.

String concatenation

Let s1 and s2 be two strings over alphabet X. We write s1.s2 to denote the concatenation of strings s1 and s2.

For example, given the alphabet X={0, 1}, and two strings 011 and 101 over X, we obtain 011.101=011101. It is easy to see that |s1.s2|=|s1|+|s2|. Also, for any string s, we have s. ε =s and ε.s=s.

Languages

A set L of strings over an alphabet X is known as a language. A language can be finite or infinite.

The following sets are finite languages over the binary alphabet {0, 1}:

∅: The empty set

{ε}: A language consisting only of one string of length zero

{00, 11, 0101}: A language containing three strings

Regular expressions

Given a finite alphabet X, the following are regular expressions over X:

If a belongs to X, then a is a regular expression that denotes the set {a}.

Let r1 and r2 be two regular expressions over the alphabet X that denote, respectively, sets L1 and L2. Then r1.r2 is a regular expression that denotes the set L1.L2.

Regular expressions (contd.)

If r is a regular expression that denotes the set L then r^+ is a regular expression that denotes the set obtained by concatenating L with itself one or more times also written as L^+ Also, r^* known as the Kleene closure of r, is a regular expression. If r denotes the set L then r^* denotes the set $\{\varepsilon\} \cup L^+$.

If r1 and r2 are regular expressions that denote, respectively, sets L1 and L2, then r1r2 is also a regular expression that denotes the set L1 $\cup$ L2.

Review of FSMs

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.

Requirements specification or design specification?

An FSM could serve any of two roles: as a specification of the required behavior and/or as a design artifact according to which an application is to be implemented.

The role assigned to an FSM depends on whether it is a part of the requirements specification or of the design specification.

Note that FSMs are a part of UML 2.0 design notation.

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.

FSM and statcharts

Note that FSMs are different from statecharts. While FSMs can be modeled using statecharts, the reverse is not true.

Techniques for generating tests from FSMs are different from those for generating tests from statecharts.

The term “state diagram” is often used to denote a graphical representation of an FSM or a statechart.

FSM (Mealy machine): Formal 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,

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

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

FSM (Moore machine): Formal definition

An FSM (Moore) is a 7-tuple: $(X, Y, Q, q_{0,} \delta, O, F)$, where:$_{,}$

X , Y, Q, q_0, $_{\text{and}}$ δ are the same as in FSM (Mealy)

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

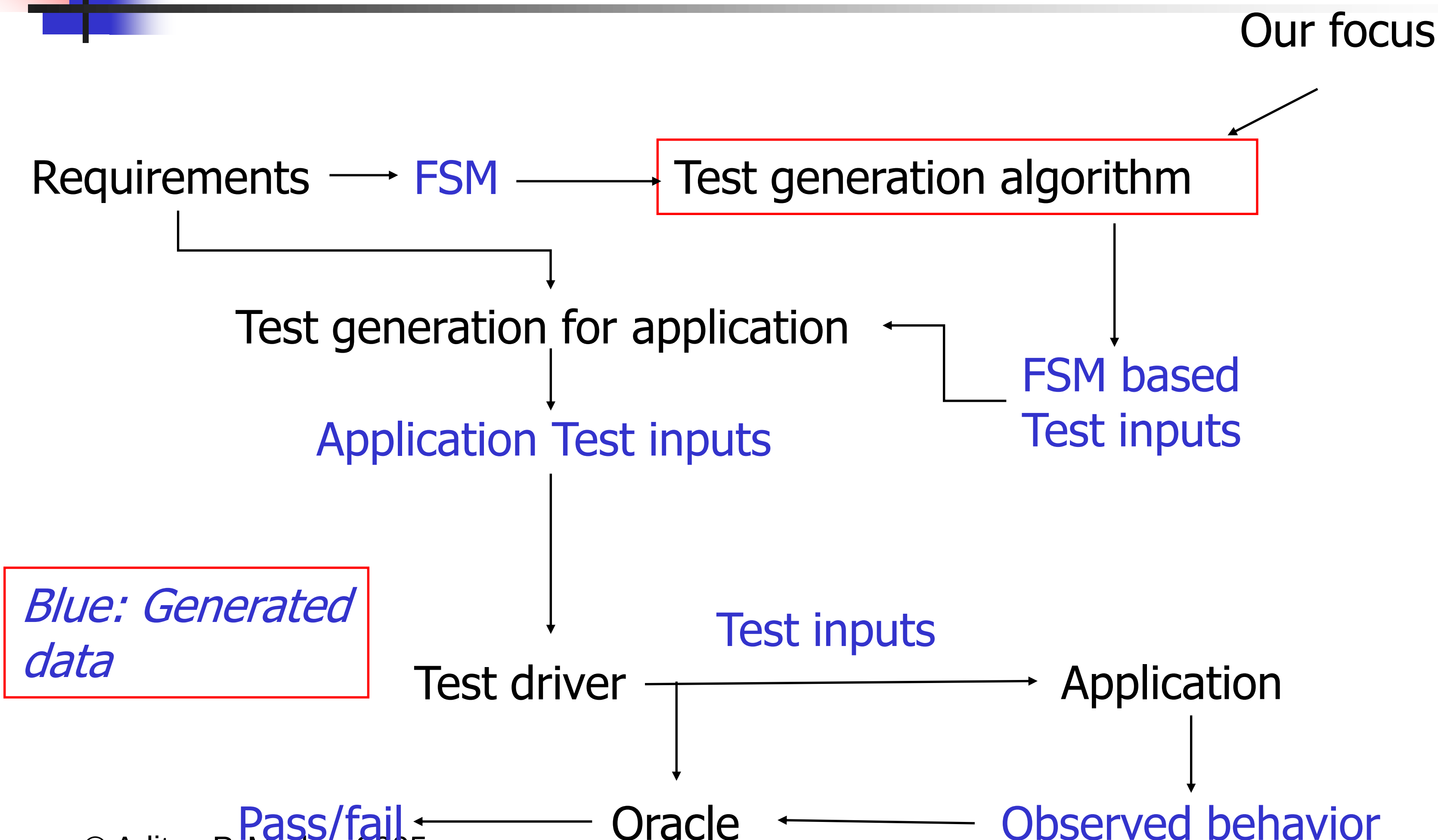
$F \in Q$ is the set of final or accepting or finish states.

FSM: Formal definition (contd.)

Mealy machines are due to G. H. Mealy (1955 publication)

Moore machines are due to E. F. Moore (1956 publication)

Test generation from FSMs



Embedded systems and Finite State Machines (FSMs)

Embedded systems

Many real-life devices have computers embedded in them. For example, an automobile has several embedded computers to perform various tasks, engine control being one example. Another example is a computer inside a toy for processing inputs and generating audible and visual responses.

Such devices are also known as embedded systems. An embedded system can be as simple as a child's musical keyboard or as complex as the flight controller in an aircraft. In any case, an embedded system contains one or more computers for processing inputs.

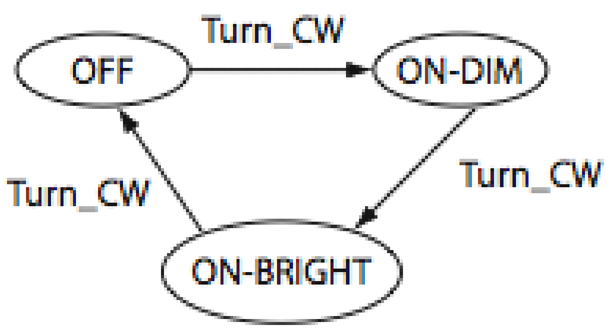
Specifying embedded systems

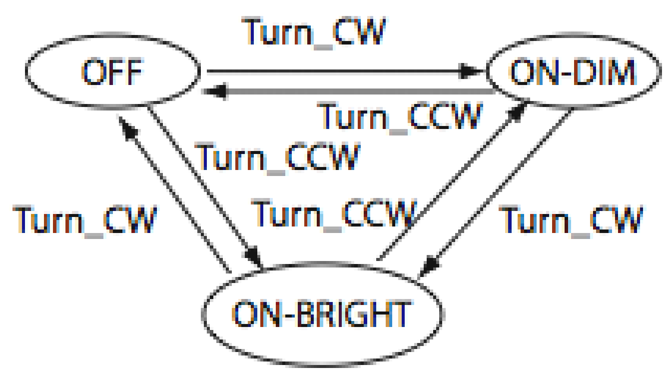
An embedded computer often receives inputs from its environment and responds with appropriate actions. While doing so, it moves from one state to another.

The response of an embedded system to its inputs depends on its current state. It is this behavior of an embedded system in response to inputs that is often modeled by a finite state machine (FSM).

FSM: lamp example

Simple three state lamp behavior:





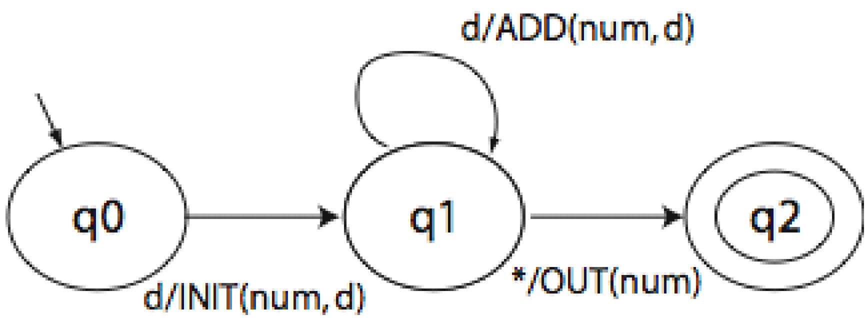
(a) (b)

(a) Lamp switch can be turned clockwise.

(b) Lamp switch can be turned clockwise and counterclockwise..

FSM: Actions with state transitions

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.

FSM: Formal definition

An FSM is a quintuple: $(X, Y, Q, q_{0,} \delta, O)$, 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,

FSM: Formal definition (contd.)

q_0 in Q is the initial state,

δ: Q x X$\rightarrow$ Q is a next-state or state transition function, and

O: Q x X$\rightarrow$ Y is an output function.

In some variants of FSM more than one state could be specified as an initial state. Also, sometimes it is convenient to add $F\subseteq Q$ as a set of final or accepting states while specifying an FSM.

State diagram representation of FSM

A state diagram is a directed graph that contains nodes representing states and edges representing state transitions and output functions.

Each node is labeled with the state it represents. Each directed edge in a state diagram connects two states. Each edge is labeled i/o where i denotes an input symbol that belongs to the input alphabet X and o denotes an output symbol that belongs to the output alphabet O. i is also known as the input portion of the edge and o its output portion.

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.

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				

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_i are considered V-equivalent if $O_1(q_i, s)=O_2(v, q_j)$ 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 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 $q_i \varepsilon Q_1$ and $q_j \varepsilon Q_2$ are considered k-equivalent if, when excited by any input of length k, yield identical output sequences.

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.

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.

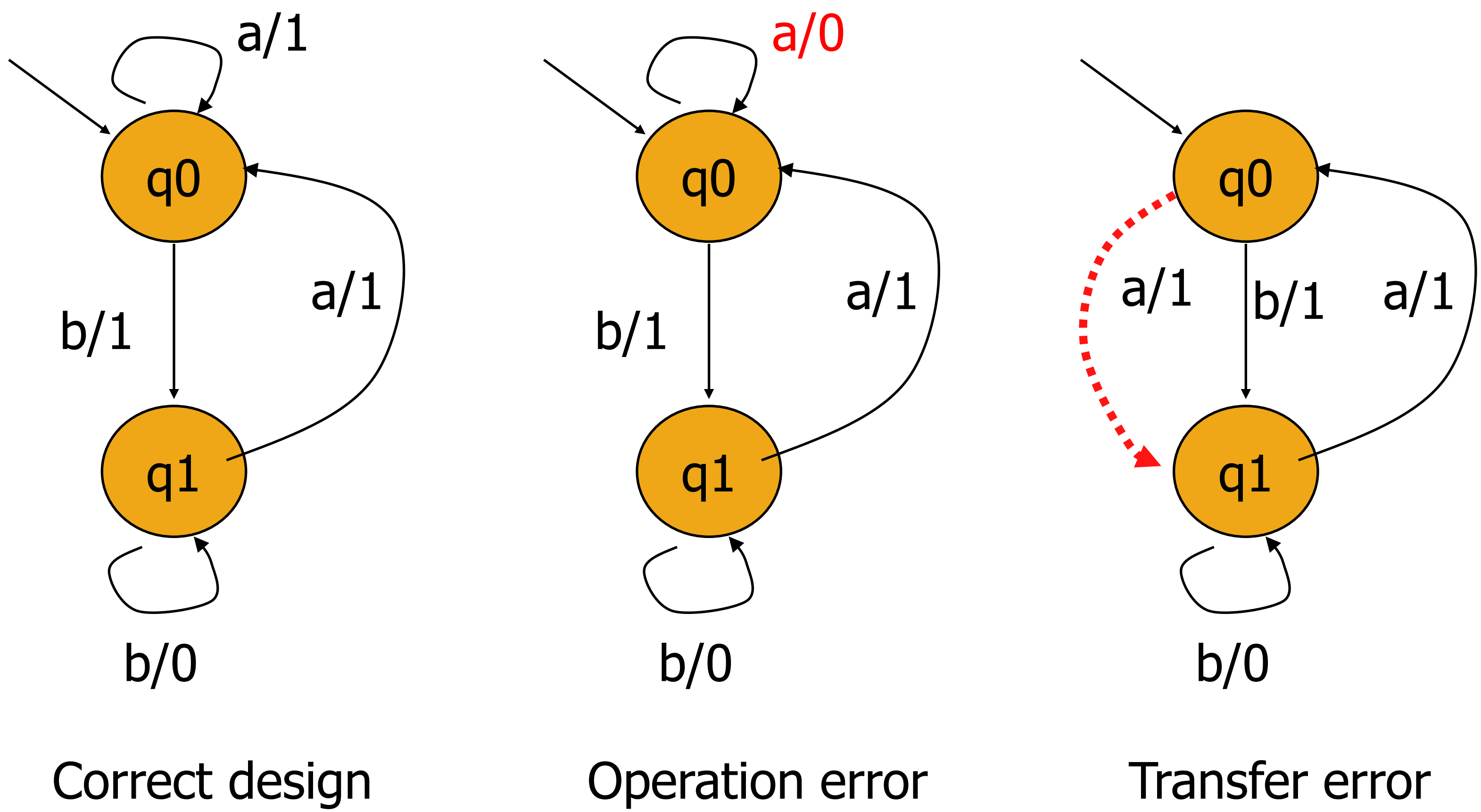
Faults Targeted

Faults in implementation

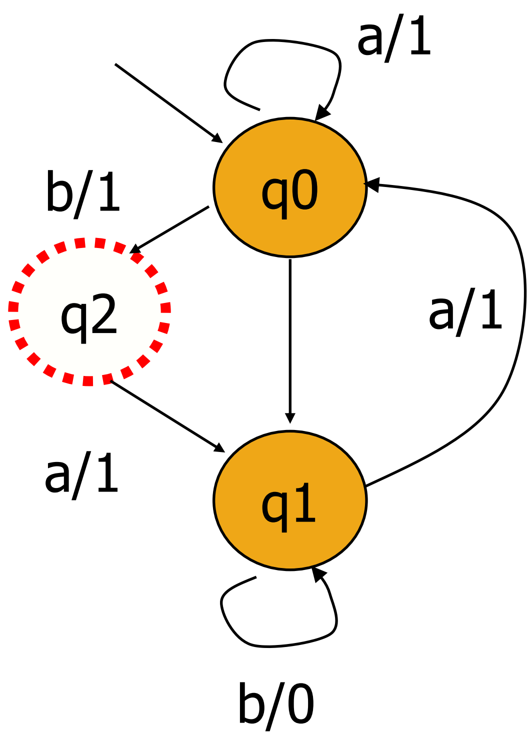
An FSM serves to specify the correct requirement or design of an application. Hence tests generated from an FSM target faults related to the FSM itself.

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

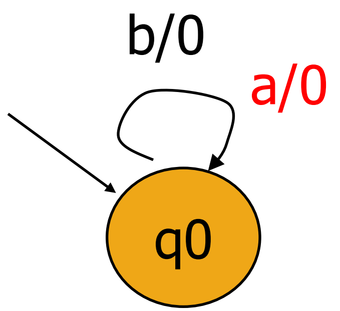
Fault model


a/1
q0
b/1
a/1
q1
b/0
Correct design
a/0
q0
b/1
a/1
q1
b/0
Operation error
q0
a/1
b/1
a/1
q1
b/0
Transfer error

Fault model (contd.)

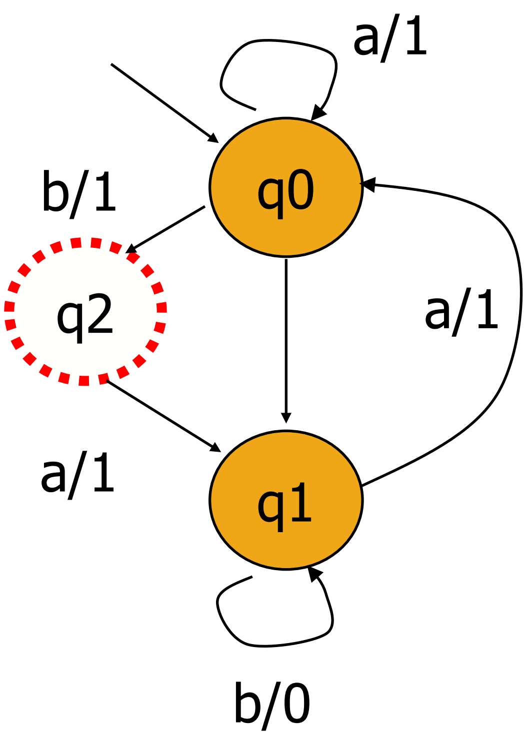


Extra state error

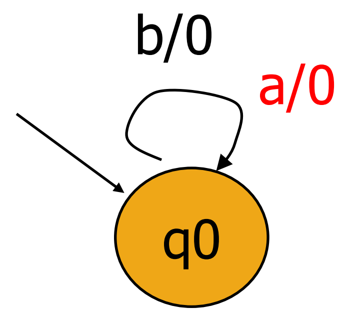


Missing state error

Fault model (contd.)



Extra state error



Missing state error

Test generation using Chow's method

Assumptions for test generation

Minimality: 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.

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.

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 $m=|Q|$.

Step 2: Construction of W. What is W?

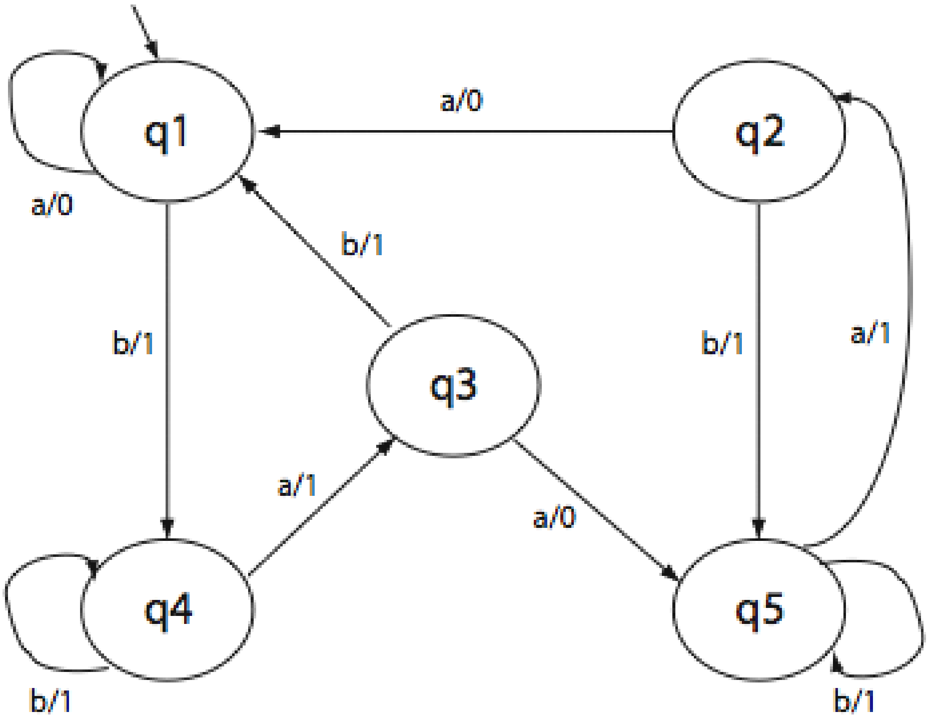
Let $M=(X, Y, Q, q1, \delta, O)$ be a minimal and complete FSM.

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

$O(qi, s) \neq O(qj, s)$

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

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

States in Σ_{ki} are k-equivalent.

If state v 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 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 lies state q_i.

P_1 Table

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

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.

P_2 Table

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

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	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	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 of the process

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

Finding the distinguishing sequences: Example (contd.)

Using the procedure analogous to the one 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}

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

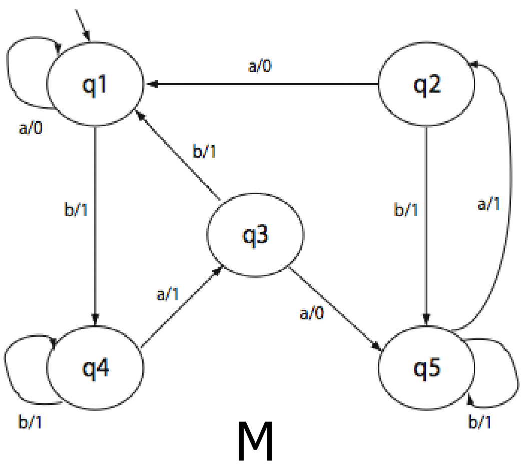
Step 3: (a) Construct the testing tree for M

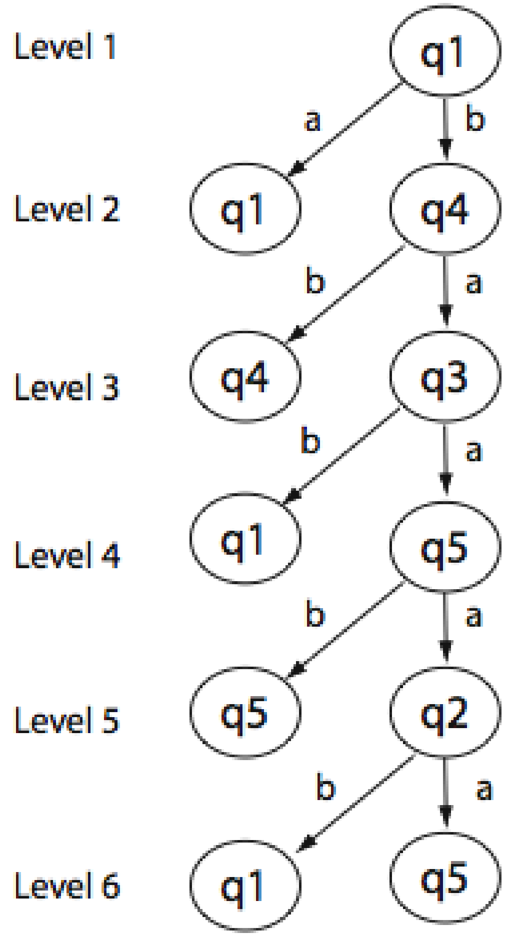
A testing tree of an FSM is a tree rooted at the initial state. It contains at least one path from the initial state to the remaining states in the FSM. Here is how we construct the testing tree.

State q0, 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 , 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 $x\square X$. 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

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.

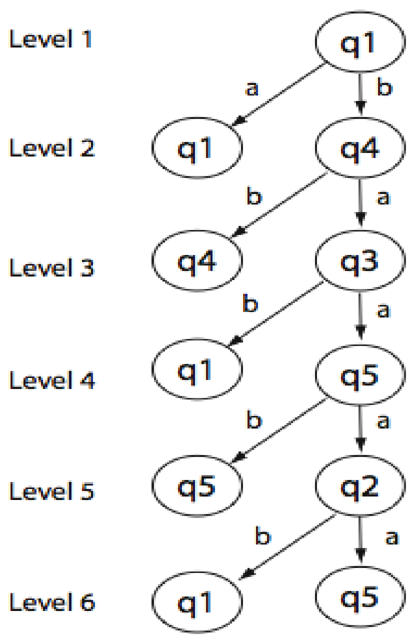
Step 3: (a) Construct the testing tree for M and (b) generate the transition cover set P from the testing tree. ← Next, (b)

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

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

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, we get

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

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

$Z = W \cup X^1.W =$ {a, aa, aaa, baaa} $\cup$ {a, b}.{a, aa, aaa, baaa}
={a, aa, aaa, baaa, aa, aaa, aaaa, baaaa, ba, baa, baaa, bbaaa}

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

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

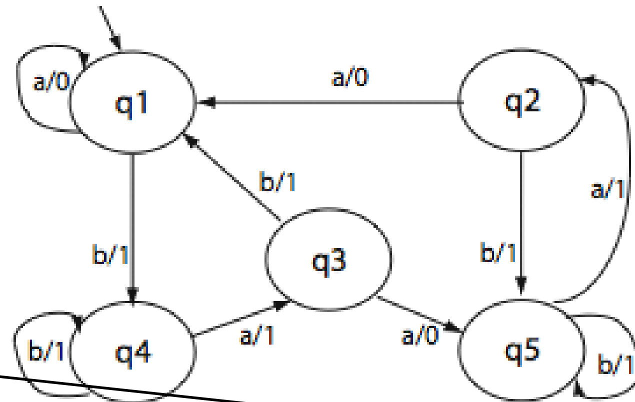
Correct design

M(t1)=1101001

M(t2)=11011

M

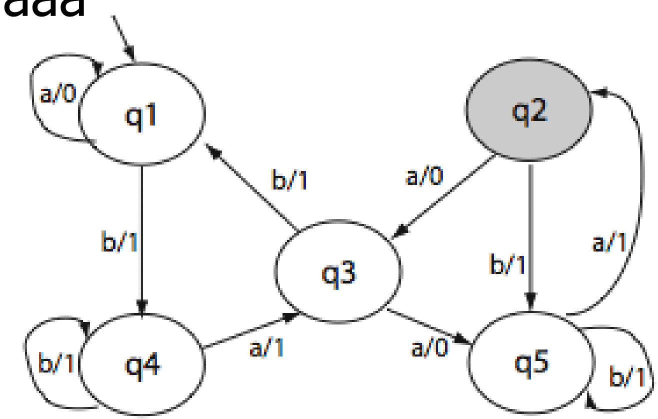
Error revealing test cases

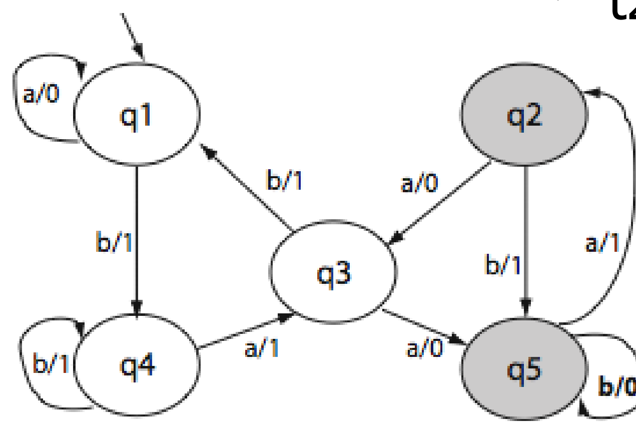


(a) Specification

t1=baaaaaa

t2=baaba





M1(t1)=1101001

(b) Transfer error in state q2.

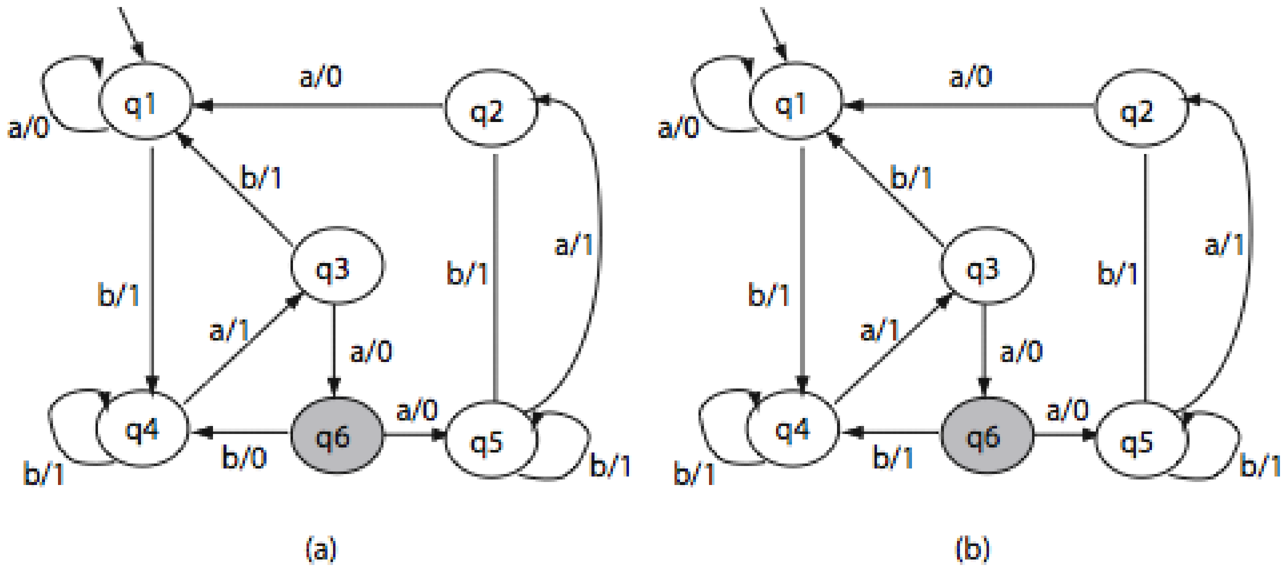
(c) Transfer error in state q2 and operation error in state q5.

M2(t2)=11001

M1

M2

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



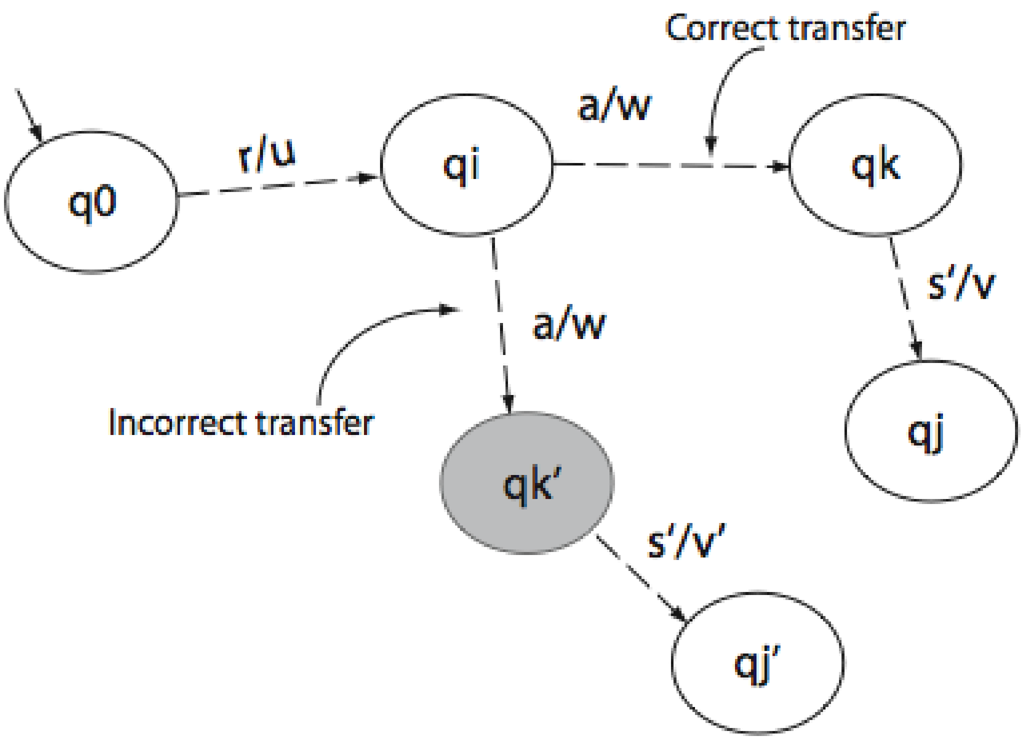
M1 M2

t1=baaba M(t1)=11011 M1(t1)=11001

t2=baaa M(t2)=1101 M2(t2)=1100

Error detection process: in-class discussion

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 q0 to state qj. Then, s=as’ takes it from qi to state qj or qj’.



Automata theoretic versus control theoretic methods for test generation

Automata-theoretic vs. Control theoretic techniques

The W and the Wp methods are considered automata-theoretic methods for test generation.

In contrast, many books on software testing mention control theoretic techniques for test generation. Let us understand the difference between the two types of techniques and their fault detection abilities.

Control theoretic techniques

State cover: A test set T is considered adequate with respect to the state cover criterion for an FSM M if the execution of M against each element of T causes each state in M to be visited at least once.

Transition cover: A test set T is considered adequate with respect to the branch/transition cover criterion for an FSM M if the execution of M against each element of T causes each transition in M to be taken at least once

Control theoretic techniques (contd.)

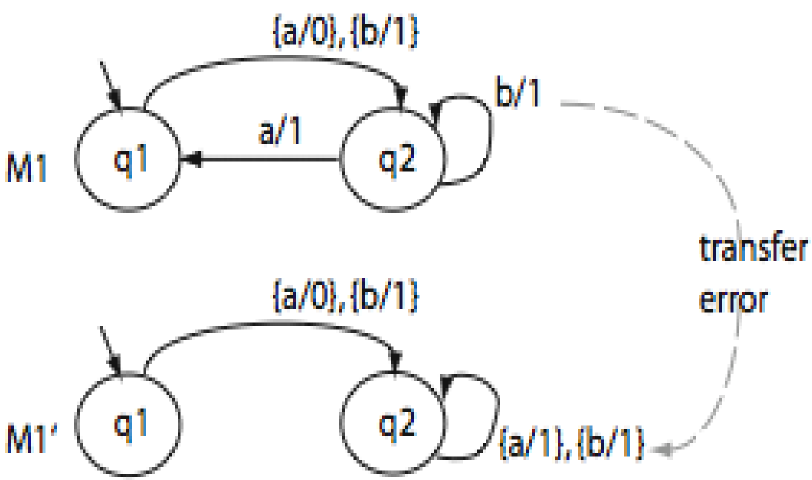
Switch cover: A test set T is considered adequate with respect to the 1-switch cover criterion for an FSM M if the execution of M against each element of T causes each pair of transitions (tr1, tr2) in M to be taken at least once, where for some input substring ab tr1: qi=δ(qj, a) and tr_2: qk= δ(qi, b) and qi, qj, qk are states in M.

Control theoretic techniques (contd.)

Boundary interior cover: A test set T is considered adequate with respect to the boundary-interior cover criterion for an FSM M if the execution of M against each element of T causes each loop (a self-transition) across states to be traversed zero times and at least once. Exiting the loop upon arrival covers the ``boundary'' condition and entering it and traversing the loop at least once covers the ``interior'' condition.

Control theoretic technique: Example 1

Consider the following machines, a correct one (M1) and one with a transfer error (M1’).

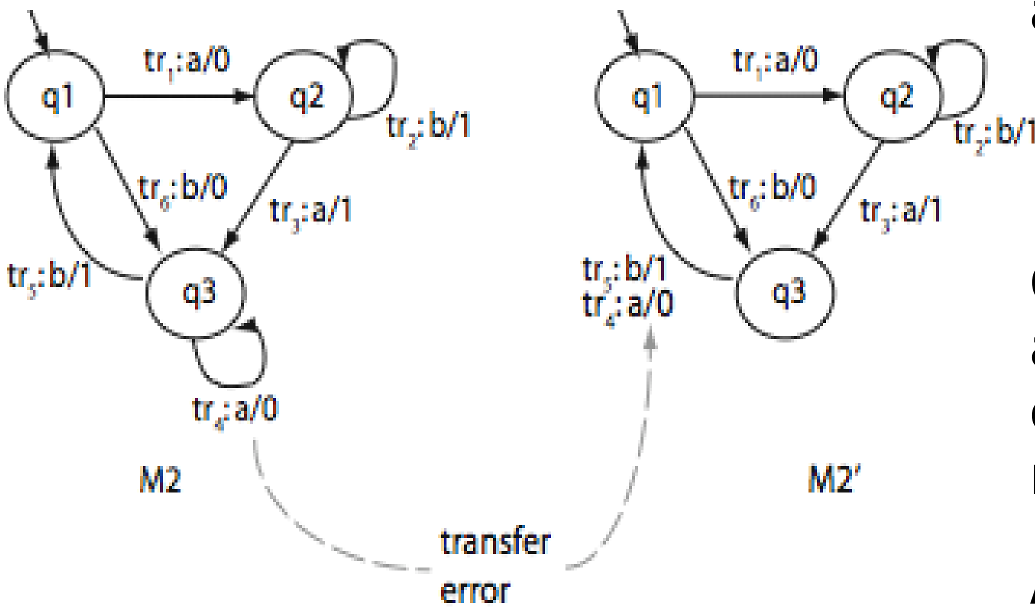


t=abba covers all states but does not not reveal the error. Both machines generate the same output which is 0111.

Will the tests generated by the W method reveal this error? Check it out!

Control theoretic technique: Example 2

Consider the following machines, a correct one (M2) and one with a transfer error (M2').



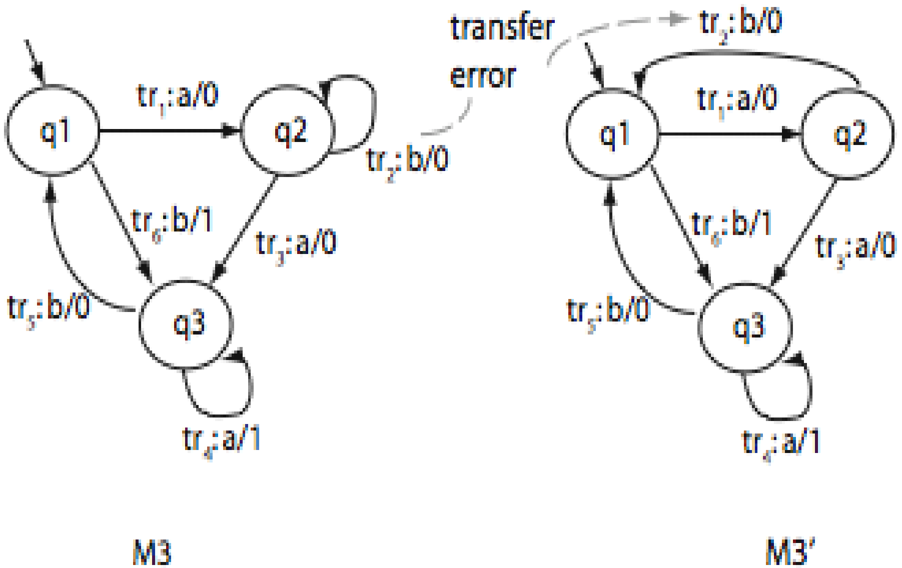
There are 12 branch pairs, such as (tr1, tr2), (tr1, tr3), tr6, tr5).

Consider the test set: {bb, baab, aabb, aaba, abbaab}. Does it cover all branches? Does it reveal the error?

Are the states in M2 1-distinguishable?

Control theoretic technique: Example 3

Consider the following machines, a correct one (M3) and one with a transfer error (M3').



Consider T={t1: aab, t2: abaab}. T1 causes each state to be entered but loop not traversed. T2 causes each loop to be traversed once.

Is the error revealed by T?

The Partial W (Wp) method

The partial W (Wp) method

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

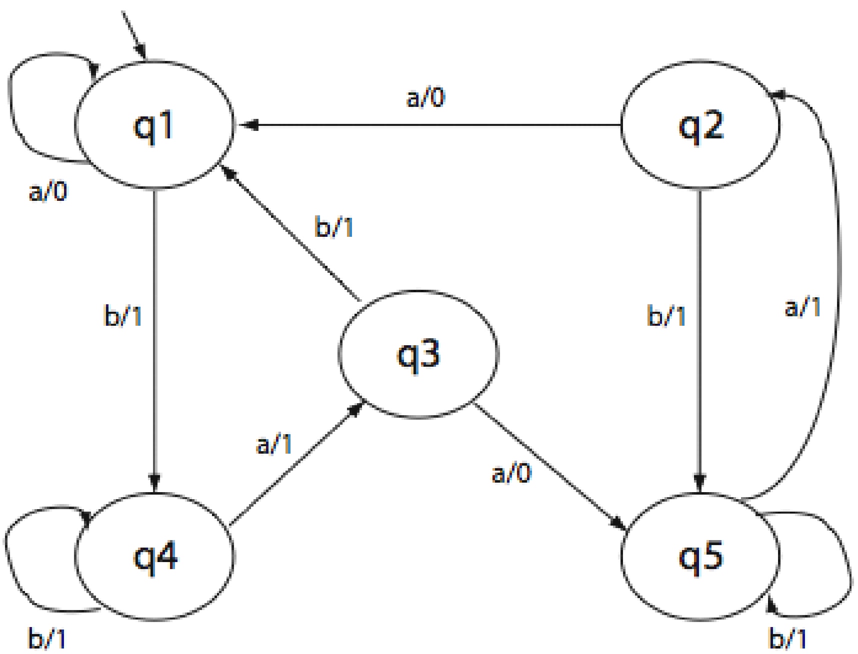
Size of tests generated is generally smaller than that generated using the W-method.

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. Also, S is not necessarily unique.

State identification set

Given an FSM M with Q as the set of states, an identification set for state $qi \in Q$ is denoted by W_i and has the following properties:

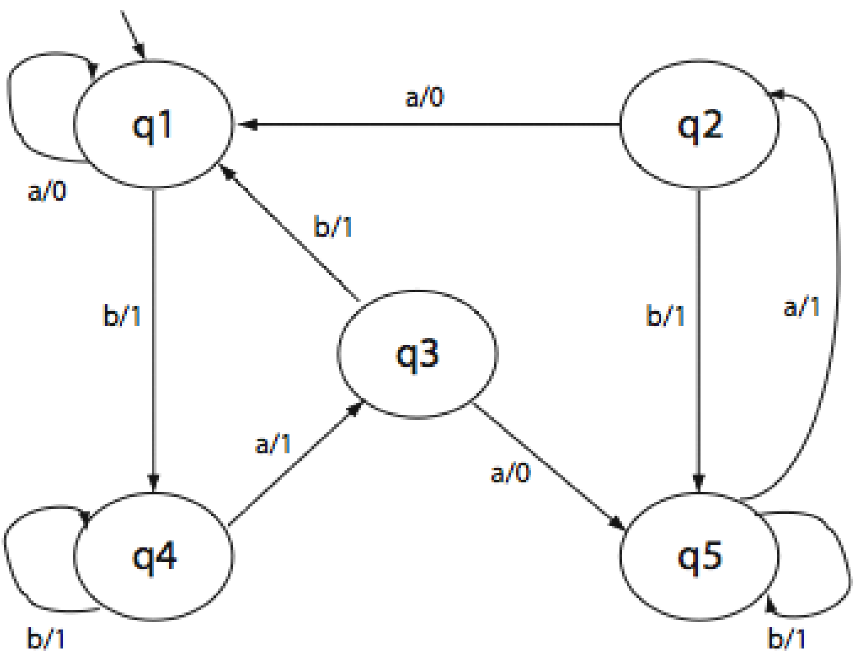
(a) $W_i \subseteq W$, $1 \leq i \leq n$ [Identification set is a subset of W.]

(b) $O(qi, s) \neq O(qj, s)$, 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



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

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

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

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

S={ε, b, ba, baa, baaa}

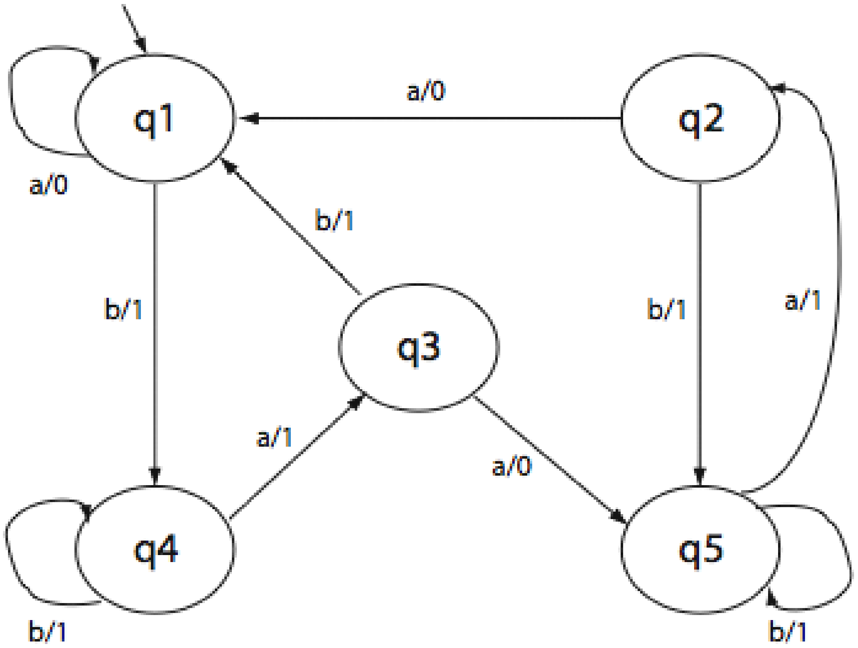
P={ε, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa}

$W_1=W_2$={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 [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},…r_{ik}.

$\delta(r_{ik}, m)=q_{ij}$, where $m \in X$ (the alphabet)

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

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

$\delta(q1, a)=q1$ $\delta(q1, bb)=q4$ $\delta(q1, bab)=q5$

$\delta(q1, baab)=q5$ $\delta(q1, baaab)=q5$ $\delta(q1, baaaa)=q1$

$T2=(\{a\}. \mathcal{W}_1) \cup (\{bb\}. \mathcal{W}_4) \cup (\{bab\}. \mathcal{W}_5) \cup (\{baab\}. \mathcal{W}_5) \cup$
$\{baaab\}. \mathcal{W}_5) \cup (\{baaaa\}. \mathcal{W}_1)$
$=\{abaaa, aaa, aa\} \cup \{bba, bbaaa\} \cup \{baba, babaaa\} \cup$
$\{baaba, baabaaa\} \cup \{baaaba, baaabaaa\} \cup \{baaaabaaa,$
$baaaaaa, baaaaa\}$

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. Also, even when tests from phase cover all transitions, they do not apply the state identification sets and hence not all transfer errors are guaranteed to be revealed by these tests.

Wp method:

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

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

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

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?*

Summary (contd.)

Automata theoretic techniques generate tests superior in their fault detection ability than their control-theoretic counterparts.

Control-theoretic techniques, that are often described in books on software testing, include branch cover, state cover, boundary-interior, and n-switch cover.

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