Test Generation from Finite State Models

Dr. Mark C. Paulk

SE 4367 – Software Testing, Verification, Validation, and Quality Assurance

Topics: Software Testing

Part II: Test Generation

3. Domain Partitioning

4. Predicate Analysis

5. Test Generation from Finite State Models

6. Test Generation from Combinatorial Designs

What Is a Finite State Machine?

An FSM is an abstract representation of behavior exhibited by some systems.

An FSM is derived from application requirements.

- **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, …**

States and Transitions

A state is a description of the status of a system that is waiting to execute a transition.

A transition is a set of actions to be executed when a condition is fulfilled or when an event is received.

- **Identical stimuli trigger different actions depending on the current state.**

In some finite-state machine representations, it is possible to associate actions with a state.

- **entry action: performed when entering the state**
- **exit action: performed when exiting the state**

Where Are FSM Methods Used?

FSMs started with conformance testing of communications protocols.

Testing of any system modeled as a finite state machine – designed using FSMs

- **elevator designs, nuclear plant protection systems, steam boiler control, …**

Finite state machines are widely used in modeling all kinds of systems.

- **generation of tests from FSM specifications assists in testing the conformance of implementations to the corresponding FSM model**

Requirement or Design?

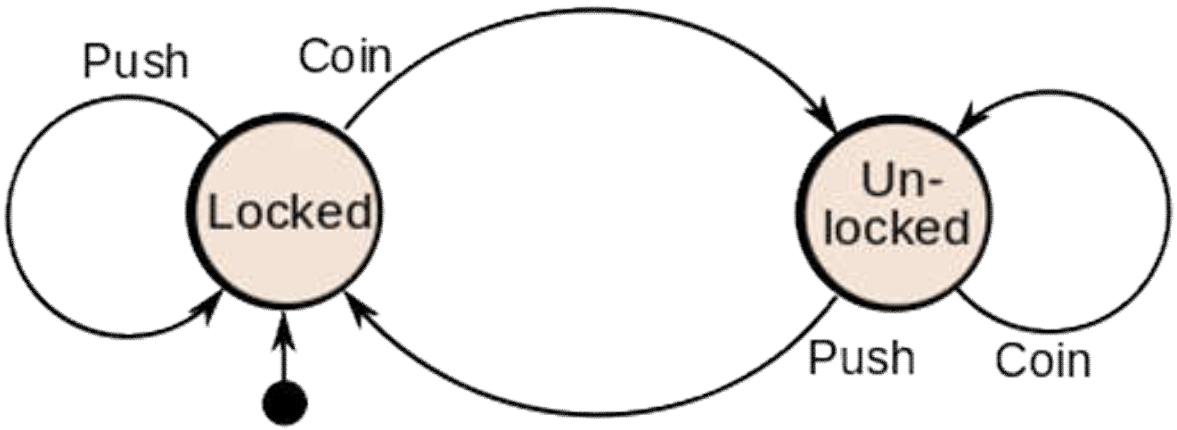
An FSM could serve

- **as a specification of the required behavior**
- **as a design artifact according to which an application is to be implemented**

FSM could be a part of the requirements specification or of the design documentation

FSMs are part of UML 2.0 design notation.

Turnstile FSM



Current State	Input	Next State	Output
Locked	coin	Unlocked	Release turnstile so customer can push through
	push	Locked	None
Unlocked	coin	Unlocked	None
	push	Locked	When customer has pushed through lock turnstile

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

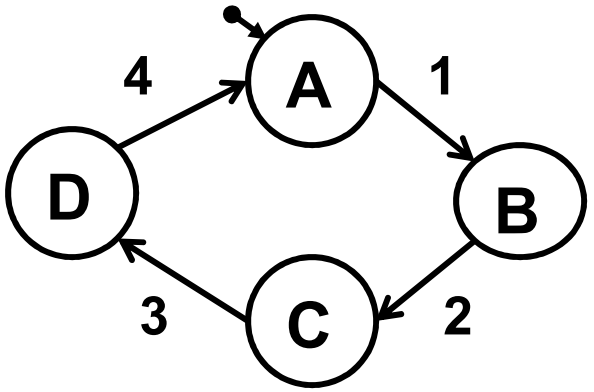
A table is often used as an alternative to the state diagram to represent the state transition function δ and the output function O.

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.**

State Transition Table

Current state → Input ↓	A	B	C	D
1	B	B	C	D
2	A	C	C	D
3	A	B	D	D
4	A	B	C	A

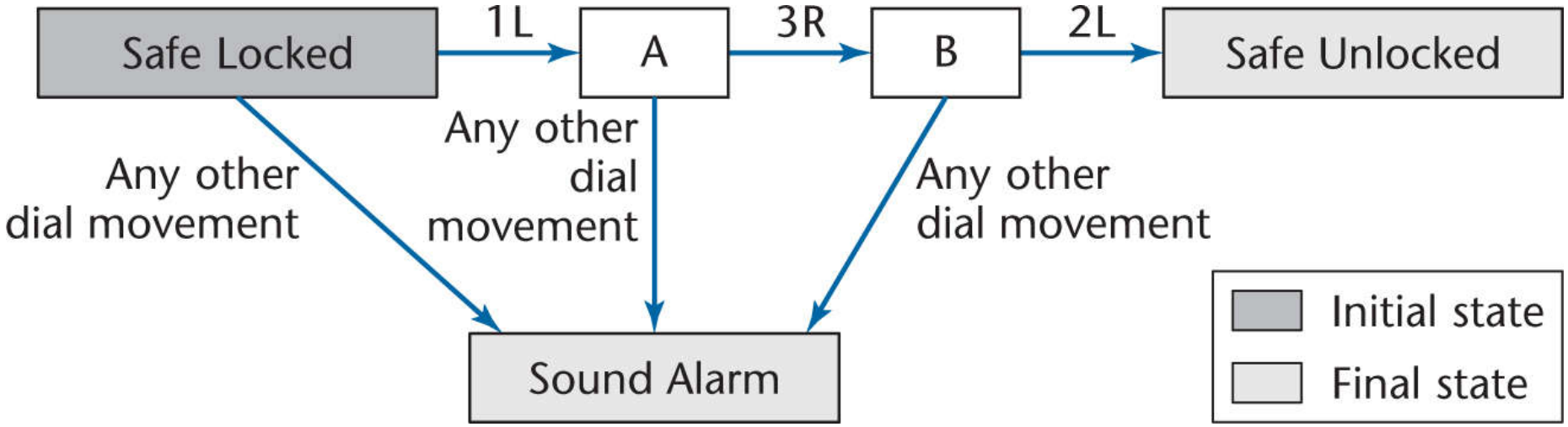


Mathur, Example 5.3

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				

Safe Example of an FSM

A safe has a combination lock that can be in one of three positions, labeled 1, 2, and 3. The dial can be turned left or right (L or R). Thus there are six possible dial movements, namely 1L, 1R, 2L, 2R, 3L, and 3R. The combination to the safe is 1L, 3R, 2L; any other dial movement will cause the alarm to go off



The set of states J is

{Safe Locked, A, B, Safe Unlocked, Sound Alarm}

The set of inputs K is

{1L, 1R, 2L, 2R, 3L, 3R}

The transition function T is on the next slide

The initial state S is Safe Locked

The set of final states F is

{Safe Unlocked, Sound Alarm}

Safe Example: Transition Table

Dial Movement \ Current State	Table of Next States Safe Locked	A	B
1L	A	Sound alarm	Sound alarm
1R	Sound alarm	Sound alarm	Sound alarm
2L	Sound alarm	Sound alarm	Safe unlocked
2R	Sound alarm	Sound alarm	Sound alarm
3L	Sound alarm	Sound alarm	Sound alarm
3R	Sound alarm	B	Sound alarm

Extended Finite State Machines

FSM transition rules have the form

current state and event ⇒ new state

Extend the standard FSM by adding global predicates

Transition rules then take the form

current state and event and predicate ⇒ new state

UML State Machines

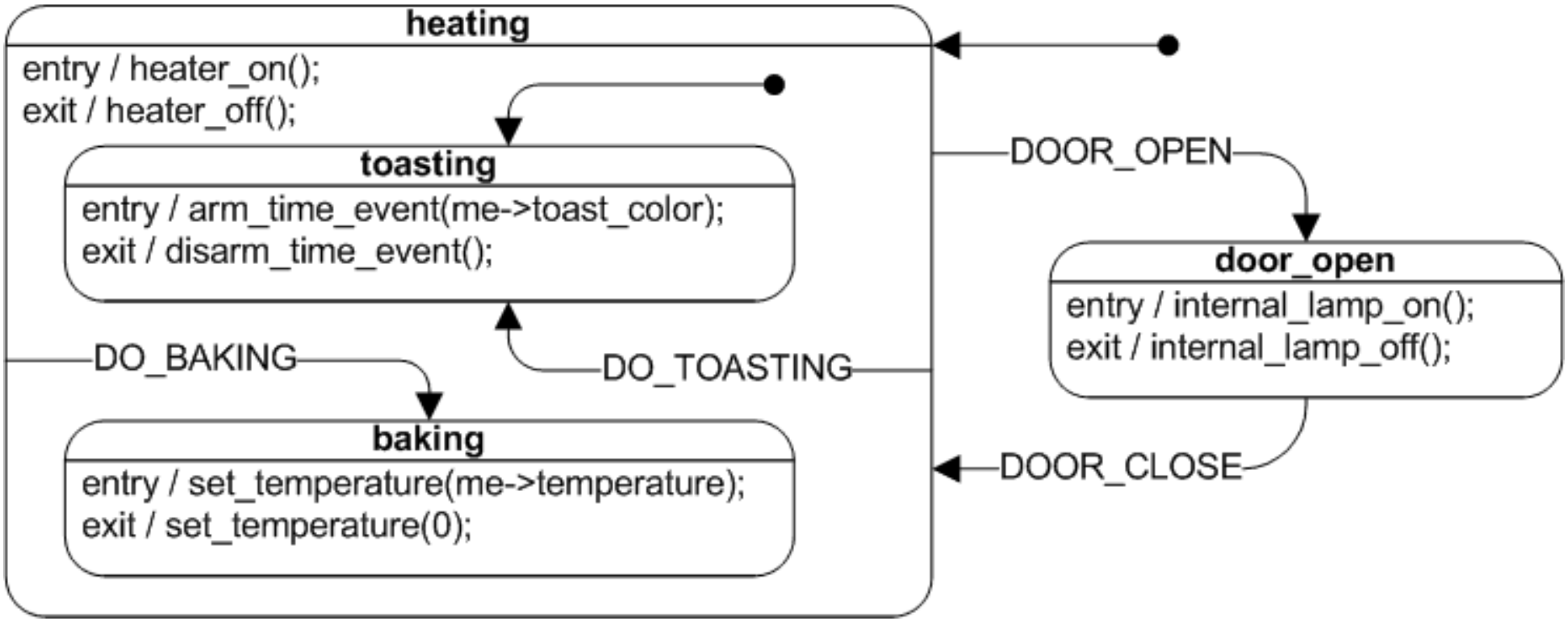
UML state machines introduce the concepts of hierarchically nested states and orthogonal regions, while extending the notion of actions.

- **object-oriented variant of Harel statechart**

Have the characteristics of both Mealy machines and Moore machines.

- **support actions that depend on both the state of the system and the triggering event, as in Mealy machines**
- **support entry and exit actions, which are associated with states rather than transitions, as in Moore machines**

UML Toaster Oven State Machine with Entry and Exit Actions (Wikipedia)



FSM Notations

There are many different kinds of finite state machines.

There are many different notations for describing FSMs.

- **some notations do not include a start or end state**
  - **an initial transition (arrow) starts the FSM**
  - **a double-circled state denotes a final state**
- **if you include the start and end nodes (states) when drawing FSMs**
  - **don't loop back to the start state**
  - **don't exit from the end state**

FSM Tradeoffs

Using an FSM, a specification is

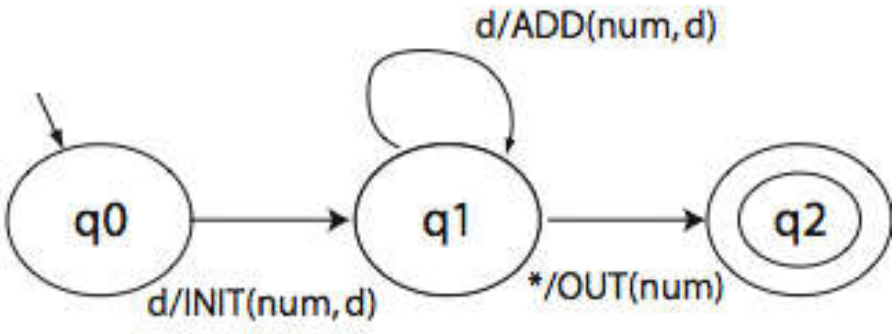
- **easy to write down**
- **easy to validate**
- **easy to convert into a design**
- **easy to convert into code automatically**
- **more precise than graphical methods**
- **almost as easy to understand**
- **easy to maintain**

Timing considerations are not handled.

A finite-state machine is a restricted Turing machine where the head can only perform "read" operations, and always moves from left to right.

Mathur, Example 5.2 (Diagram)

DIGDEC: 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.

Mathur, Example 5.3 (Table)

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

- **d is an abbreviation for the ten digits, 0 to 9**

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

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

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.

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 <u>V-equivalent</u> if $O_1(q_i, s) = O_2(q_j, s)$ for all s in V.

Distinguishability

States q_i and q_j are said to be <u>equivalent</u> if $O_1(q_i, r) = O_2(q_j, r)$ for any set V, i.e., they yield identical output sequences.

If q_i and q_j are not equivalent, then they are said to be <u>distinguishable</u>.

This definition of equivalence also applies to states within a machine.

- **machines M_1 and M_2 could be the same machine**

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.

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

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

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.

Machine Equivalence

Machines M_1 and M_2 are said to be equivalent if

- **for each state σ in M1 there exists a state σ ' in M_2 such that σ and σ ' are equivalent and**
- **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.

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.

Conformance Testing

The term conformance testing is used during the testing of communication protocols.

An implementation of a communication protocol conforms to its specification if it passes a collection of tests derived from its specification.

Protocols can be modeled as FSMs.

Software designs can be modeled as FSMs, statecharts, Petri nets, …

Faults in FSM Implementation

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

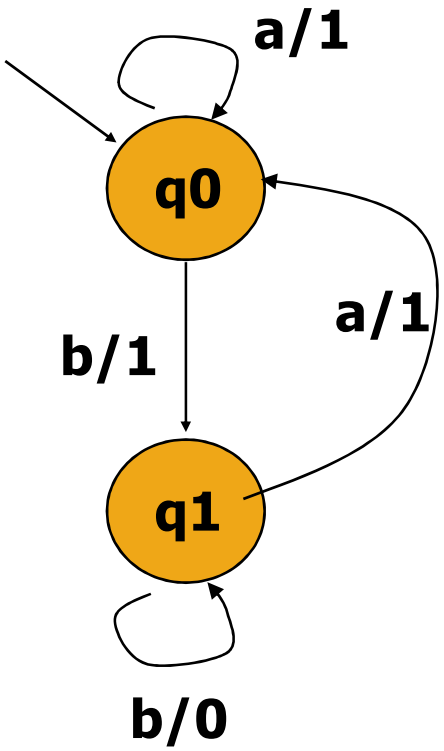
- **the number of possible implementations of a design M_d is infinite**

Tests generated from an FSM target faults related to the FSM itself.

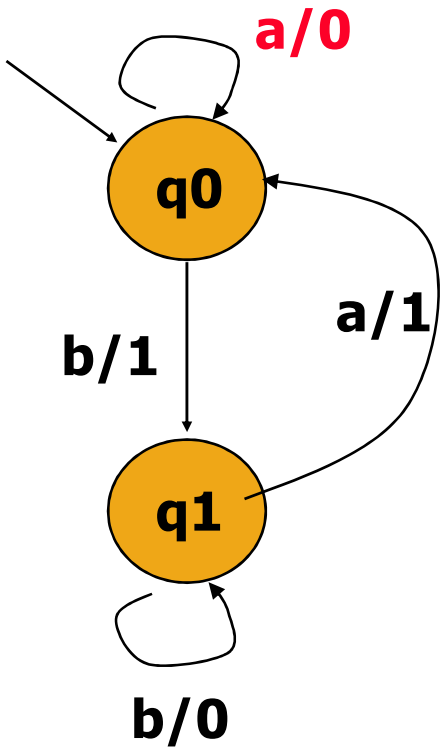
What faults are targeted by the tests generated using an FSM?

FSM Fault Models

Operation Error



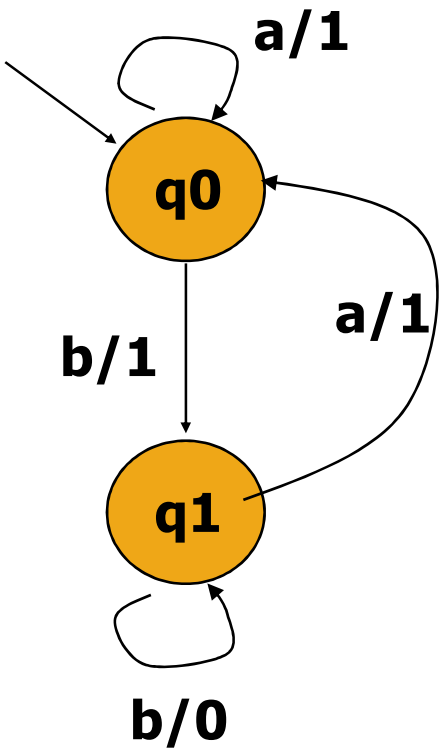
Correct design



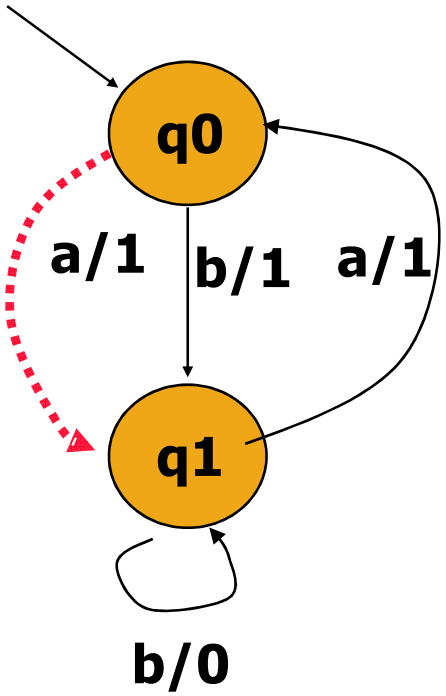
Operation error

FSM Fault Models

Transfer Error



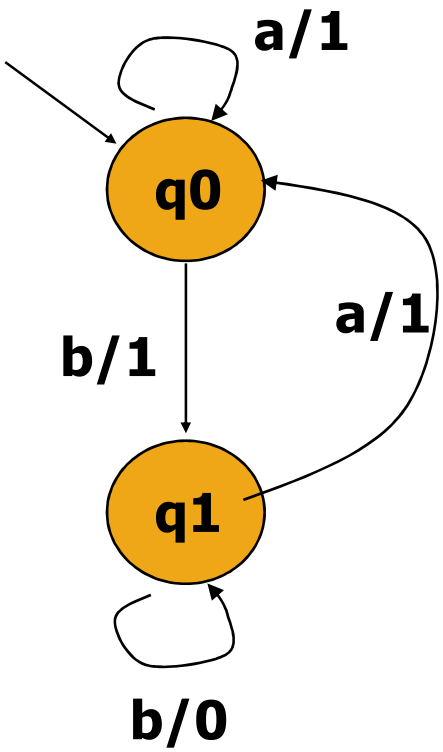
Correct design



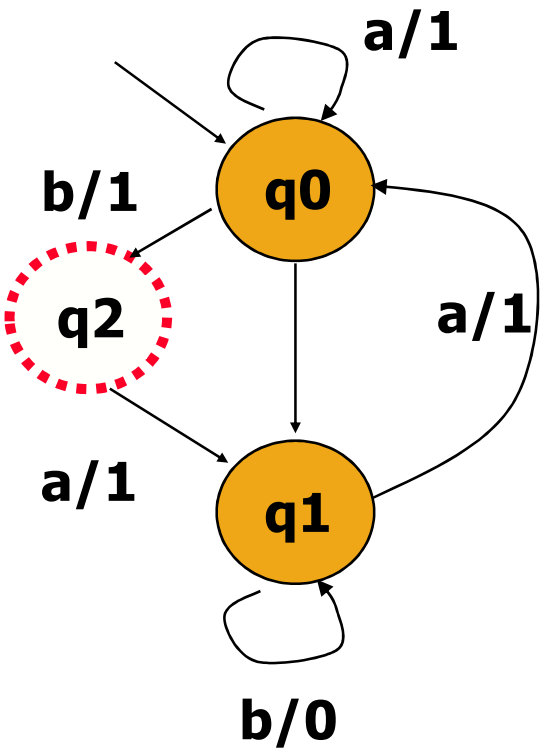
Transfer error

FSM Fault Models

Extra State Error



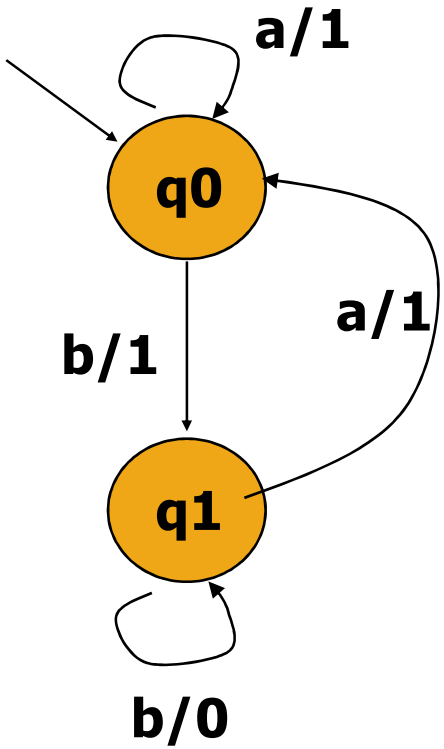
Correct design



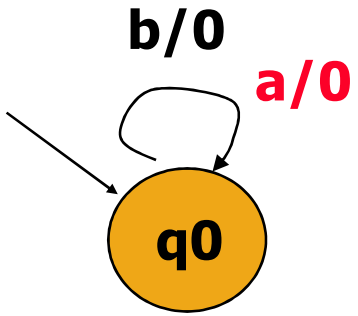
Extra state error

FSM Fault Models

Missing State Error



Correct design



Missing state error

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.**

Characterization Set W

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

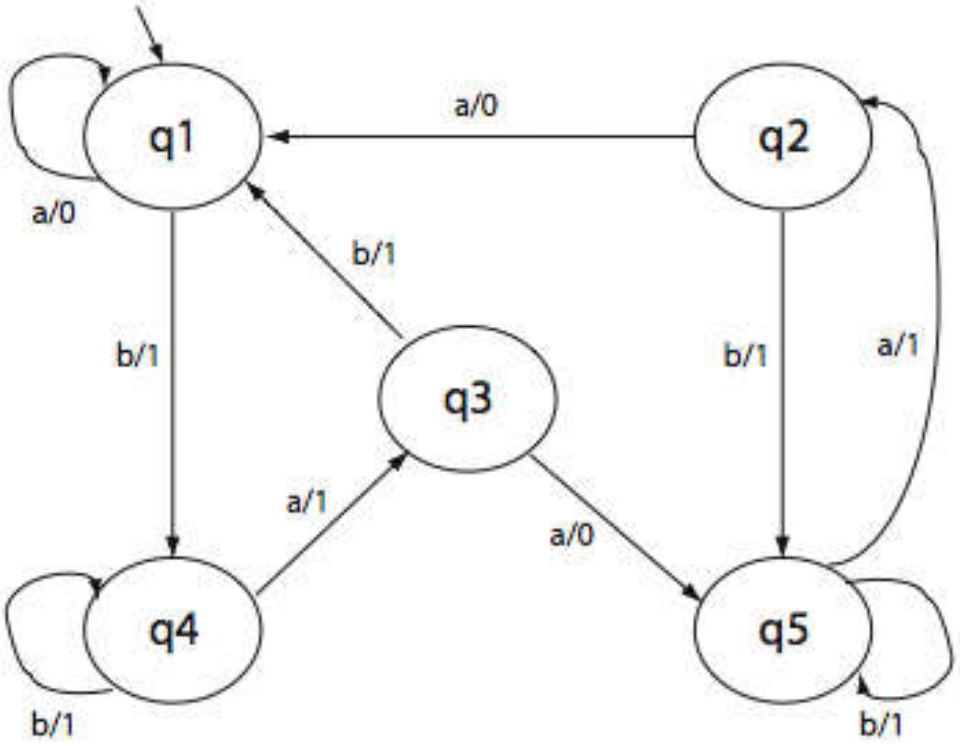
W is a finite set of input sequences that distinguish the behavior of any pair of states in M.

- **each input sequence in W is of finite length**

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

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

Mathur, Example 5.7



A simple FSM

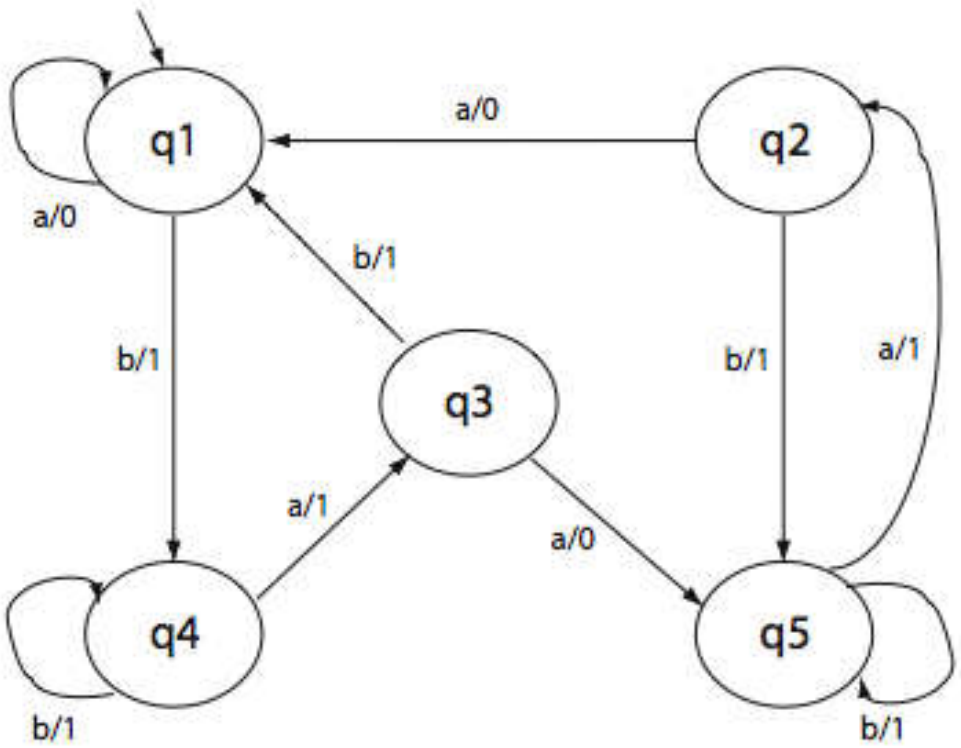
M = {X,Y,Q,q_1,δ,O}

X = {a,b}

Y = {0,1}

Q = {q_1,q_2,q_3,q_4,q_5}

q_1 is the initial state



Given
W = {a,aa,aaa,baaa}

$O(baaa,q_1) = 1101$

$O(baaa,q_2) = 1100$

baaa distinguishes state q_1 from q_2 since $O(baaa,q_1) \neq O(baaa,q_2)$

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 v is in Σ_{ki} and v in Σ_{kj} for $i \neq j$, then u and v are k-distinguishable.

Mathur, Example 5.8

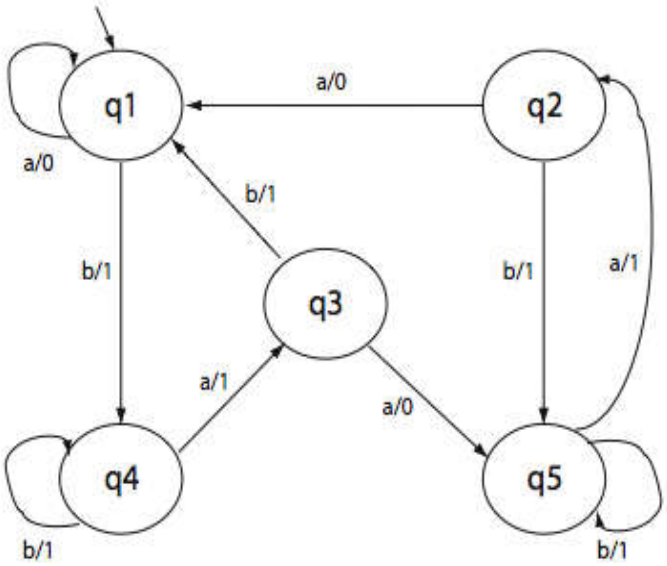
Construct the one-equivalence partition for M from Example 5.7.

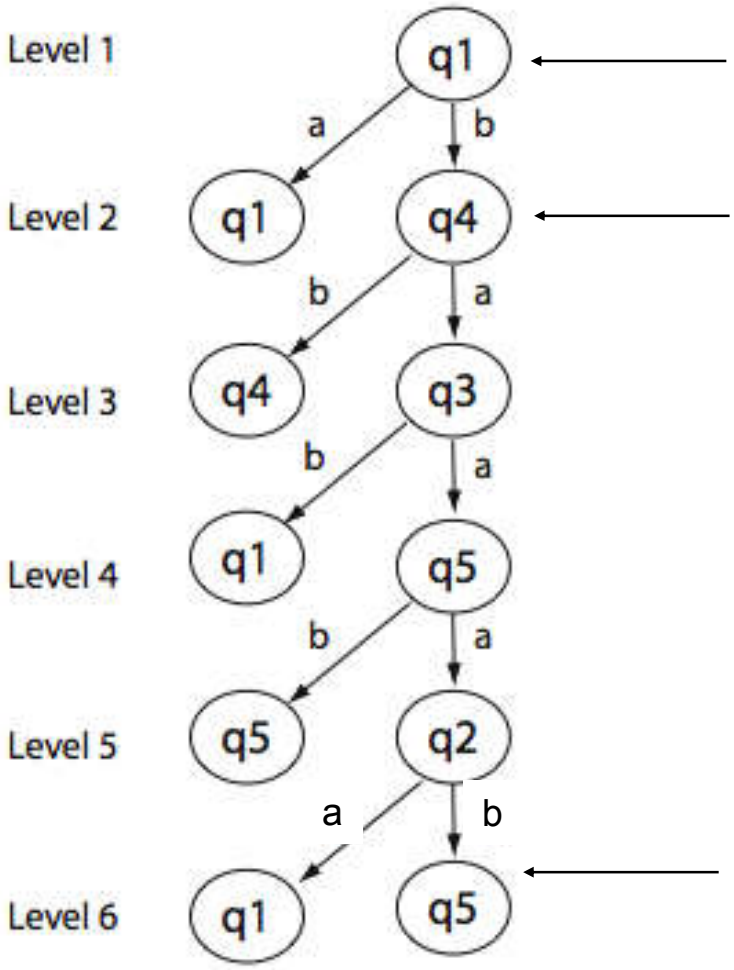
Current state	Output		Next state	
	a	b	a	b
q1	0	1	q1	q4
q2	0	1	q1	q5
q3	0	1	q5	q1
q4	1	1	q3	q4
q5	1	1	q2	q5

$p_1 = \{1, 2\}$
$\Sigma_{11} = \{q_1, q_2, q_3\}$
$\Sigma_{12} = \{q_4, q_5\}$

	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

Mathur, Example 5.11
Testing Tree





Start here, initial state is the root.

q1 becomes leaf, q4 can be expanded.

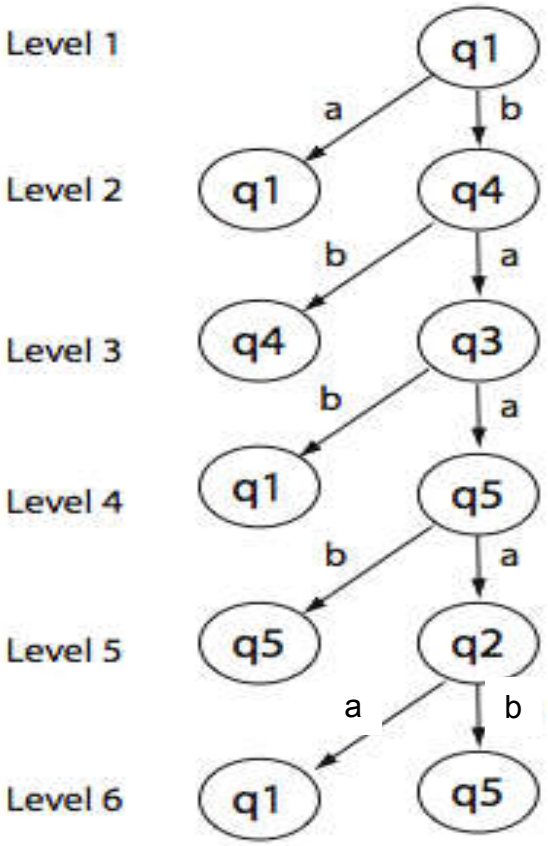
.

.

.

No further expansion possible

Transition Cover Set from the Testing Tree



A <u>transition cover set</u> 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}

Using P and W

Transition cover set P covers all the nodes and all the paths in the FSM starting from the initial state.

P and the characterization set W are used to generate test sets for FSMs…

W, Wp, and UIO

W-method uses the characterization set W of distinguishing sequences as the basis to generate a test set for an IUT (Implementation Under Test).

Partial W-method, Wp, is similar to W-method (tests are generated from a minimal, complete and connected FSM), but the size of the test set is often smaller.

Unique input/output (UIO) sequences finds transfer and operation errors only.

- **UIO sequence of input and output pairs that distinguishes a state of an FSM from the remaining states**

FSM Testing Tree Example

Given a finite state machine with

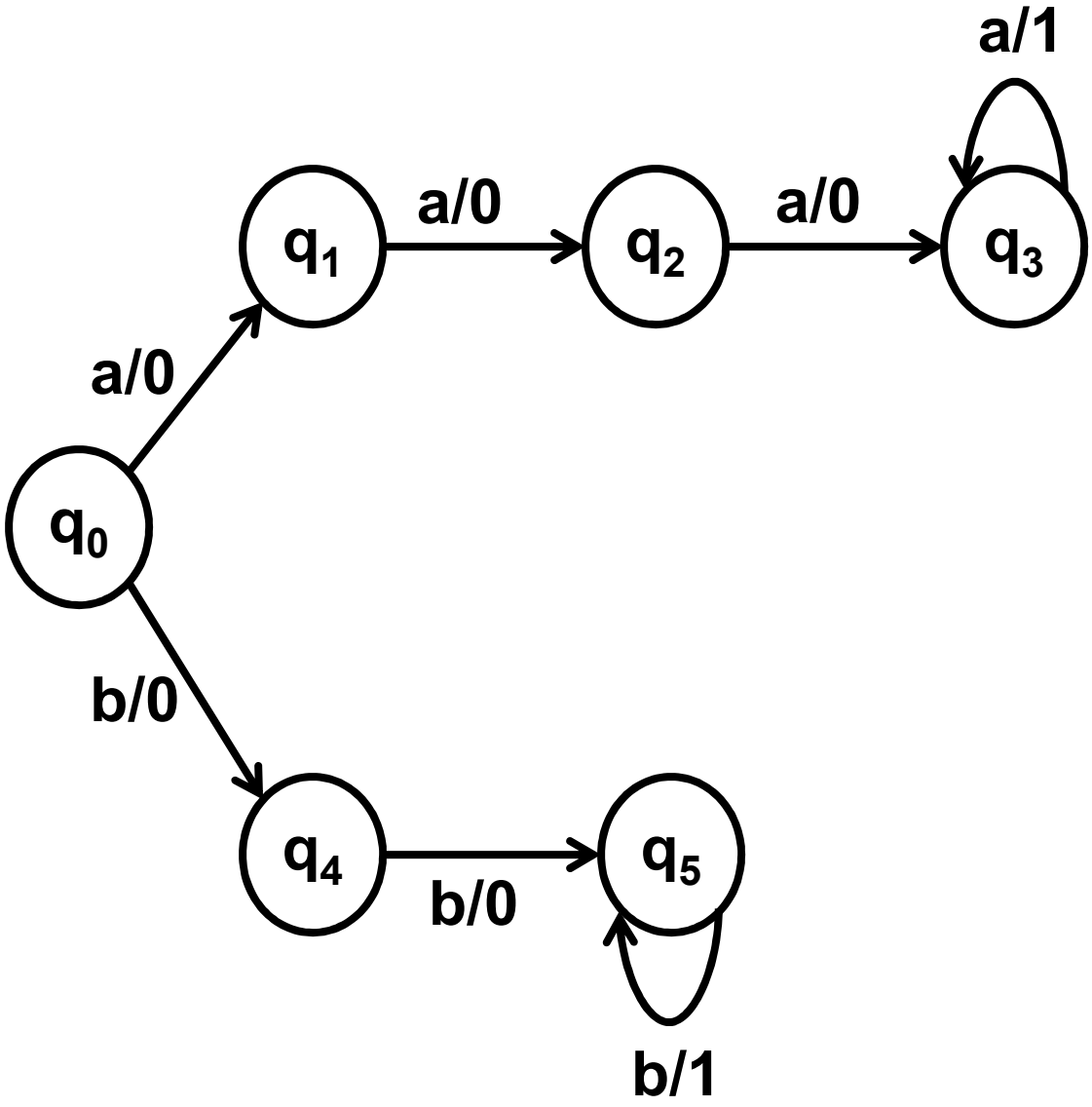
- **input alphabet {a, b}**
- **output alphabet {0,1}**

that will recognize a substring of four consecutive a's (aaaa) or three consecutive b's (bbb), including overlapping substrings.

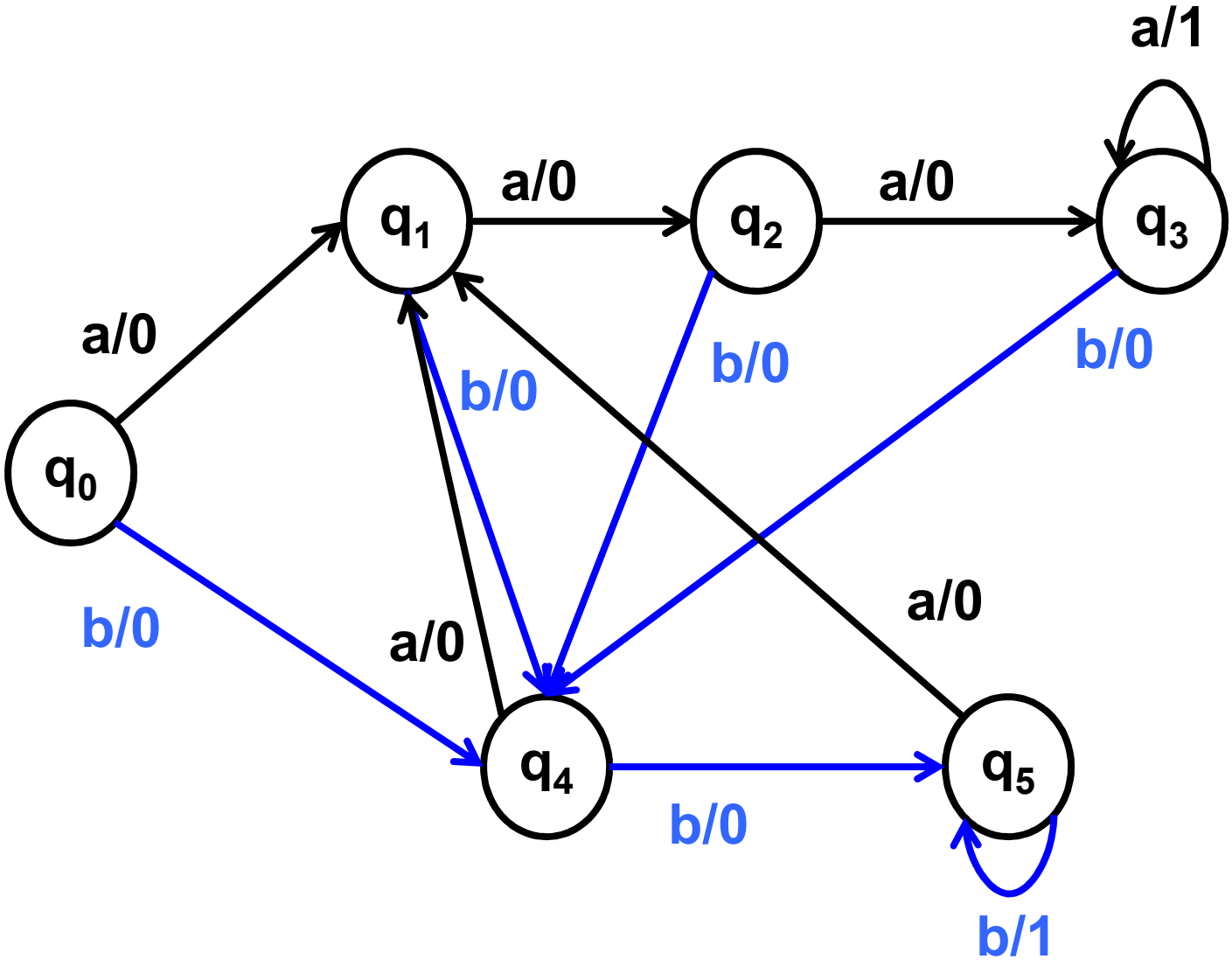
It outputs 0's until recognizing a substring, then outputs a 1.

The FSM does not terminate.

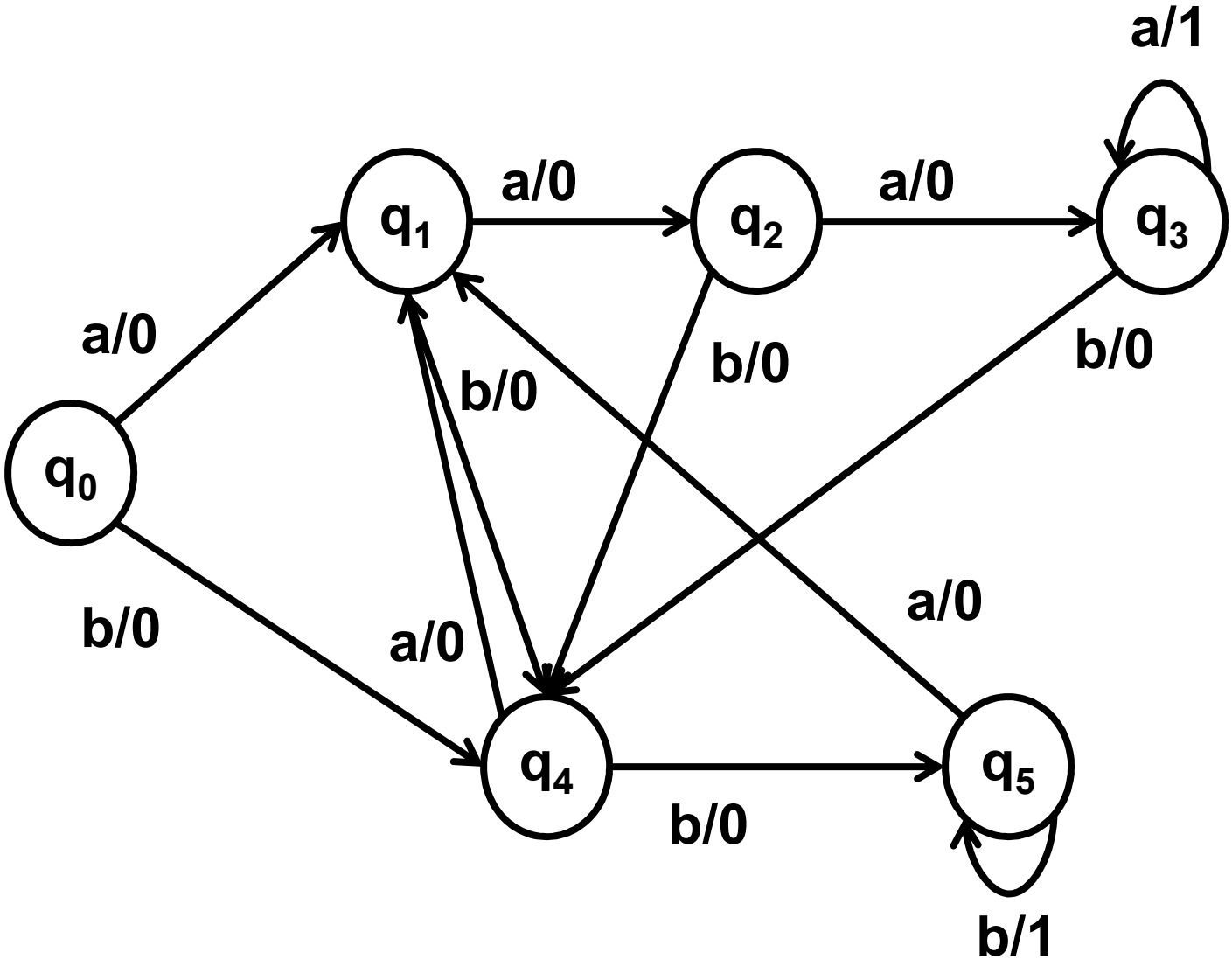
Recognizing 4 a's and 3 b's



Completely Specifying the FSM



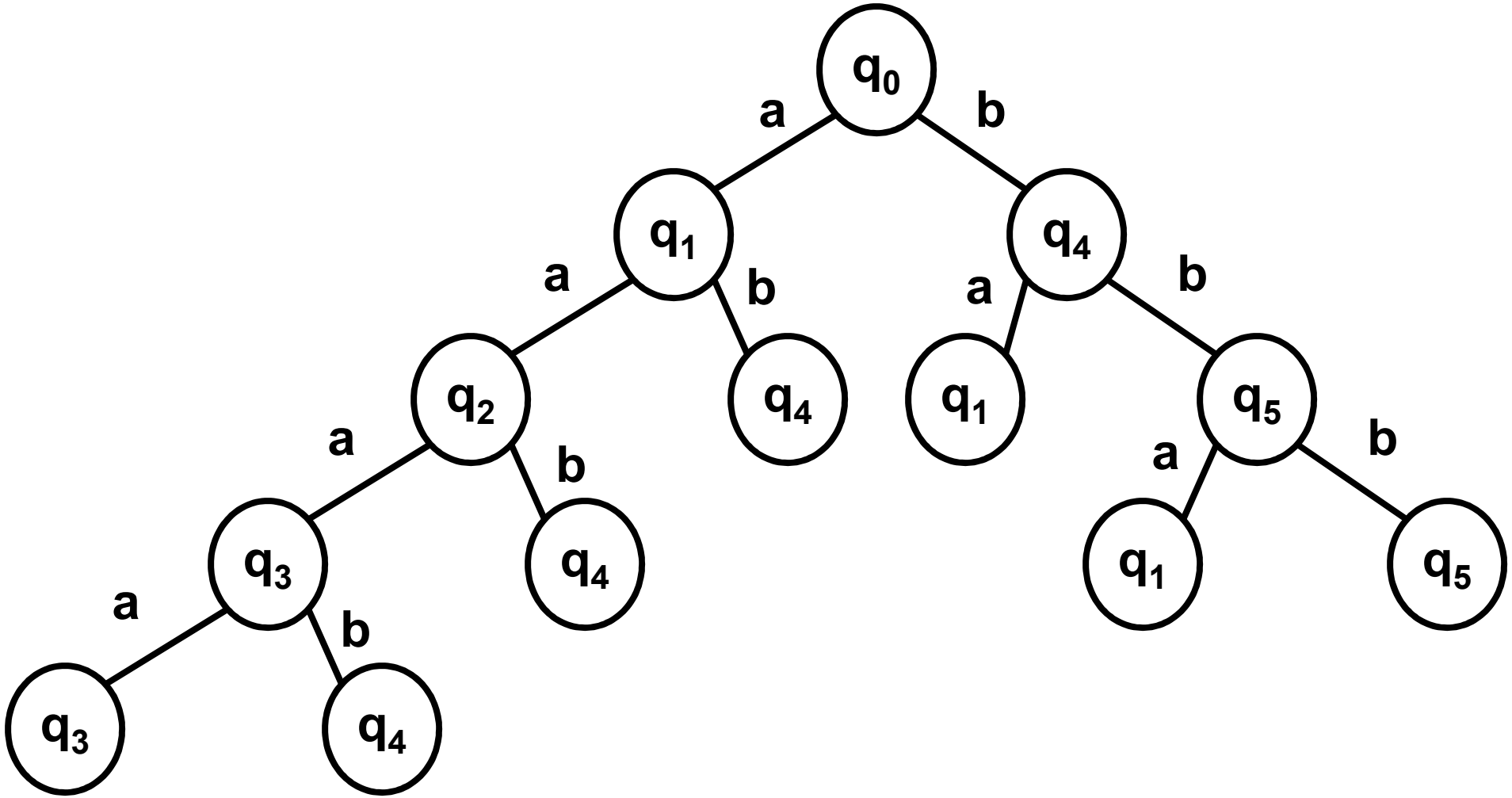
State Diagram



State Transition Table

Current state	Output a	b	Next state a	b
q_0	0	0	q_1	q_4
q_1	0	0	q_2	q_4
q_2	0	0	q_3	q_4
q_3	1	0	q_3	q_4
q_4	0	0	q_1	q_5
q_5	0	1	q_1	q_5

Testing Tree



Transition Cover Set

P = {ε, a, aa, aaa, aaaa, aaab, aab, ab, b, bb, bbb, ba, bba}

The empty input sequence ε is required (and used in generating the test sequence).

Summary

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

- **communication protocols can be modeled using FSMs**
- **GUIs can 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, missing state errors, and extra state errors in the implementation

- **given that the FSM representing the implementation is complete, connected, and minimal**

Automata theoretic techniques generate tests superior in their fault detection ability to 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 is the same.

Summary – Things to Remember

Finite state machines
- **fully specified**
- **strongly connected**

Characterization sets (W)

Testing trees

Transition cover sets

Questions and Answers