Formal black box testing
Test generation from Finite State Models

Comments on the title

- black:
  - poor information about the structure of the implementation under test
- formal:
  - there is a formal specification of the design

  $\Downarrow$

  - test generation is automated
  - successful test implies the implementation is correct*

* As opposite to the famous quote Dijkstra-Burton-Randell (see next slide)

The famous quote Dijkstra-Burton-Randell

- Program testing can be used to show the presence of bugs, but never to show their absence!
  - Source: E. W. Dijkstra: Notes On Structured Programming, T.H. –Report 70-WSK-03, 1970, at the end of section 3, On The Reliability of Mechanisms. EWD
- Testing shows the presence, not the absence of bugs
  - Source: J.N. Buxton and B. Randell, eds, Software Engineering Techniques, April 1970, p. 16. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969. BR

(Possibly the earliest documented use of the famous quote)

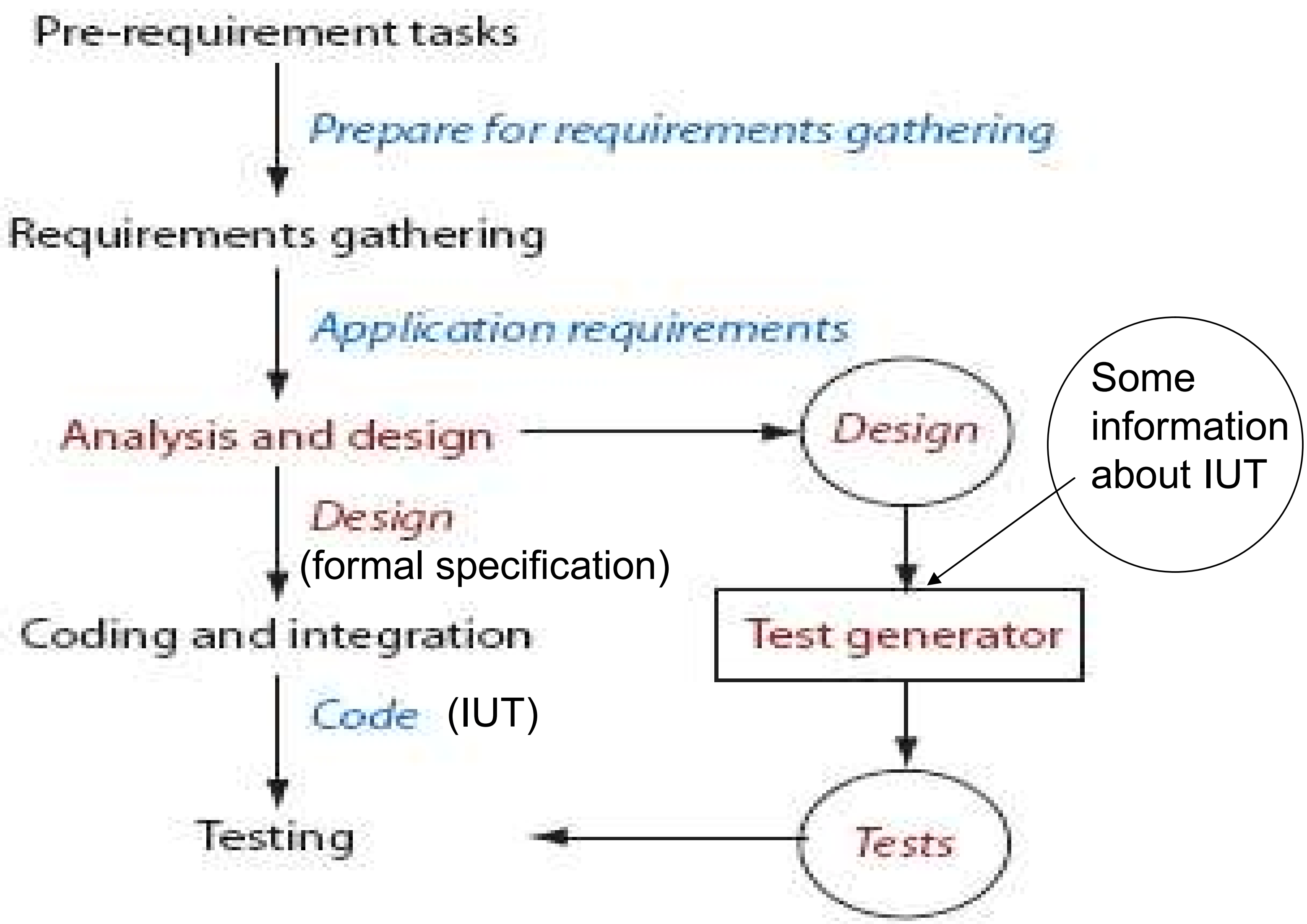
The testing problem

- the design of a system: Finite state machines(FSM), state charts, Petri Nets etc
- IUT: an Implementation(of the design)Under Test
  - a program
  - another design etc.

The testing problem:

to determine if the IUT is *equivalent* to a formal representation of a design

Design and *automated* test generation in a software development process



Test generation procedures to derive tests from FSMs (Finite State Machines)

- algorithms that take a FSM and some attributes of IUT as inputs to generate tests.
- test generation methods can be automated (though only some have been integrated into commercial test tools)

In that presentation:

- the W-method
- the transition tour (TT) method
- the distinguishing sequence (DS) method,
- the unique input/output (UIO) method
- the partial-W (Wp) method.

Bibliographic Notes

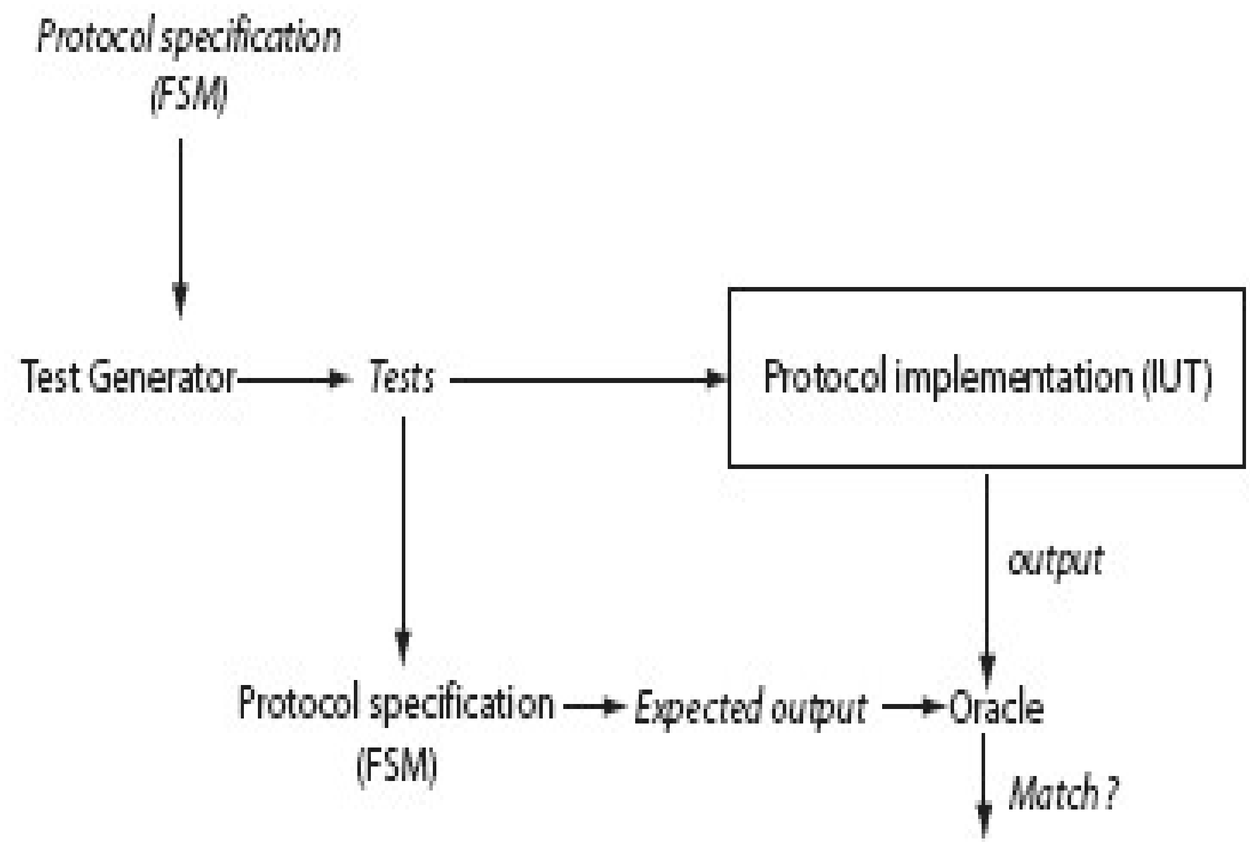
- the W-method
  [Chow78] T.S. Chow, "Testing Design Modelled by Finite-State Machines", IEEE Trans. S.E. 4, 3, 1978.
- the partial-W (Wp) method
  S. Fujiwara, G. Bochman, F. Khendek, M. Amalou, and A. Ghedasmi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591{603, June 1991.
- the unique input/output (UIO) method
  K.K. Sabnani and A.T. Dahbura, "A protocol Testing Procedure", Computer Networks and ISDN Systems, Vol. 15, No. 4, pp. 285-297, 1988.
- the transition tour (TT) method
  S. Naito and M. Tsunoyama, "Fault Detection for Sequential Machines by Transition Tours", Proc. of FTCS (Fault Tolerant Computing Systems), pp.238-243, 1981.
- the distinguishing sequence (DS) method
  G. Gonenc, “A method for the design of fault detection experiments“, IEEE Trans. Computer, Vol. C-19, pp. 551-558, June 1970.

Presentation mainly based on:

Aditya P. Mathur, *Foundations of Software Testing,*

Pearson Education. 2008, 689 pages

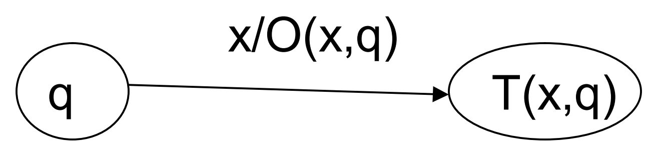
Testing a protocol implementation against an FSM model.



Deterministic FSM (DFSM)

Deterministic FSM:

- $M = (X, Y, Q, q0, T, O)$
  - X : (finite)Set of inputs
  - Y : (finite)Set of outputs
  - Q: (finite)Set of states
  - q0: the initial state,
  - T : Transition function, X x Q --> Q,
  - O : Output function, X x Q --> Y.
- T and O are extended, in a canonical way, to domain X*xQ
  - $T(ax,q)=T(x, T(a,q)), T(\varepsilon,q)=q$,
  - $O(ax,q)=O(a,q)O(x,T(a,q)), O(\varepsilon,q)= \varepsilon$

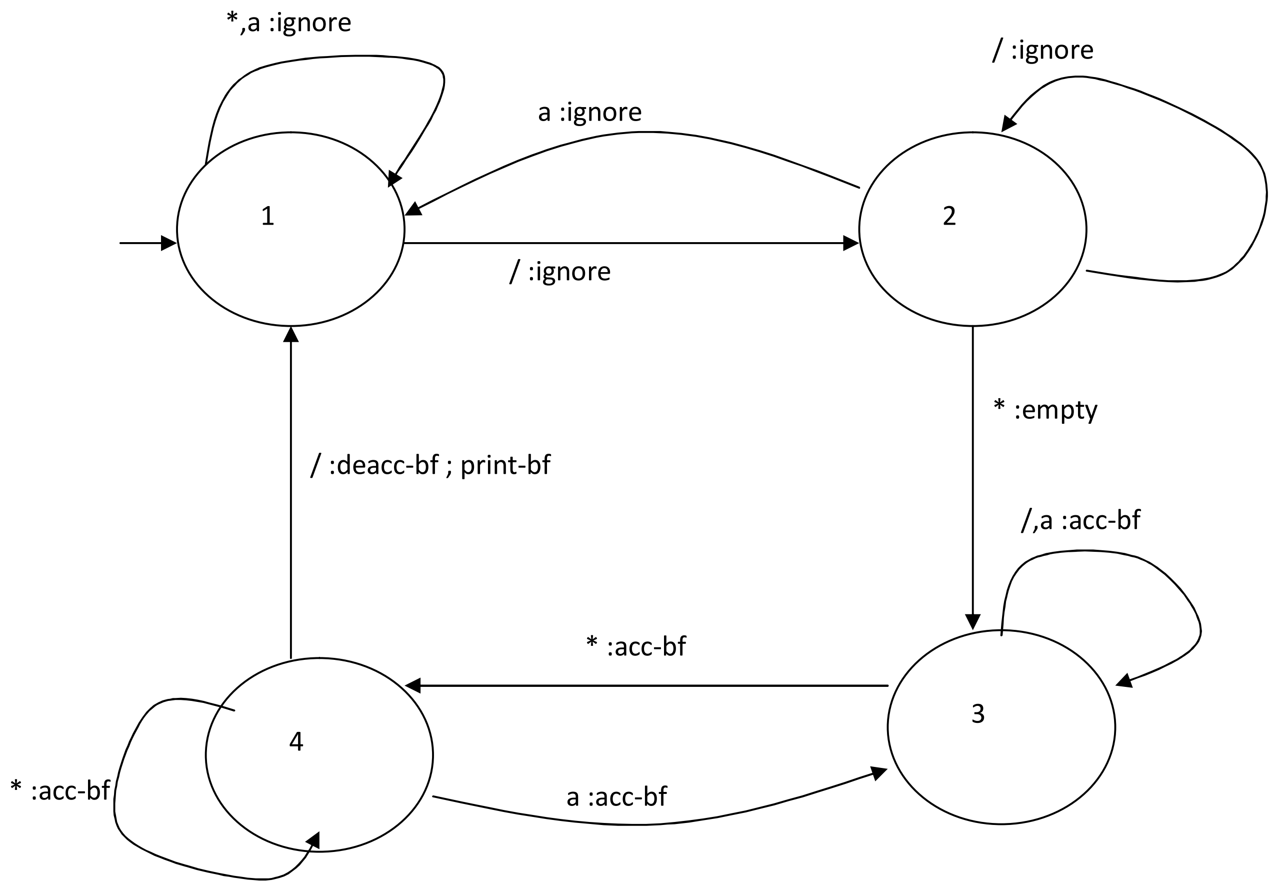


DFSM specification for a “C comments” printing system

(a "white rabbit" in an well known paper of Chow,1978)

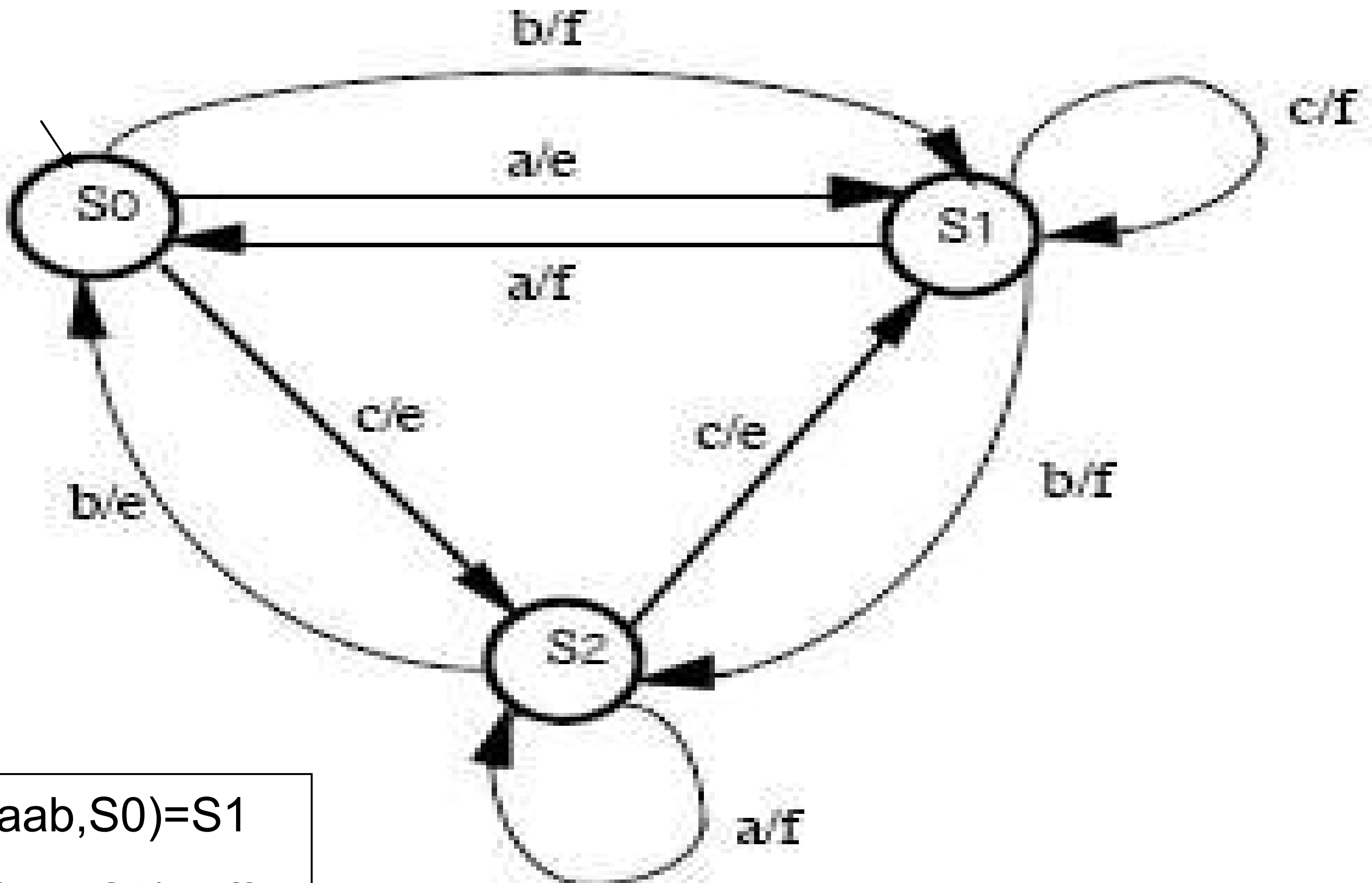
- User Requirements:
  - Input consists of characters *, /, a.
  - Print only comments
  - A comment is an input sequence enclosed by /* on the left and */ on the right (it may contain other /* 's but not */ 's)
- DFSM
  - X={*,/,a}
  - Y= {ignore, empty, acc-bf, deacc-bf, print-bf, deacc&print-bf }
  - T and O, given by the following diagram:

Diagram for “C comments” DFSM



1- waiting for a comment to start
2- a possible comment start
3- accumulating the comment content
4- waiting for comment to end

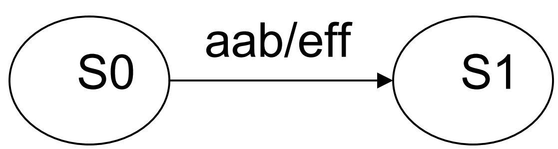
abc-DFSM, another example



T(aab,S0)=S1

O(aab,S0)=eff

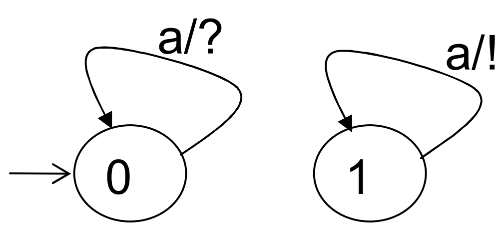
O(aab,S1)=fef



Complete, connected DFSM

- M is **completely** specified, if from each state of M there exists a transition and an output for each input symbol in X.
- M is **strongly connected**, if for each pair of states (q, p), there exists an input sequence which takes M from q to p.
- M is **connected**, if for each state q there exists an input sequence which takes M from the initial state q0 to q.

C comments DFSM is complete and strongly connected

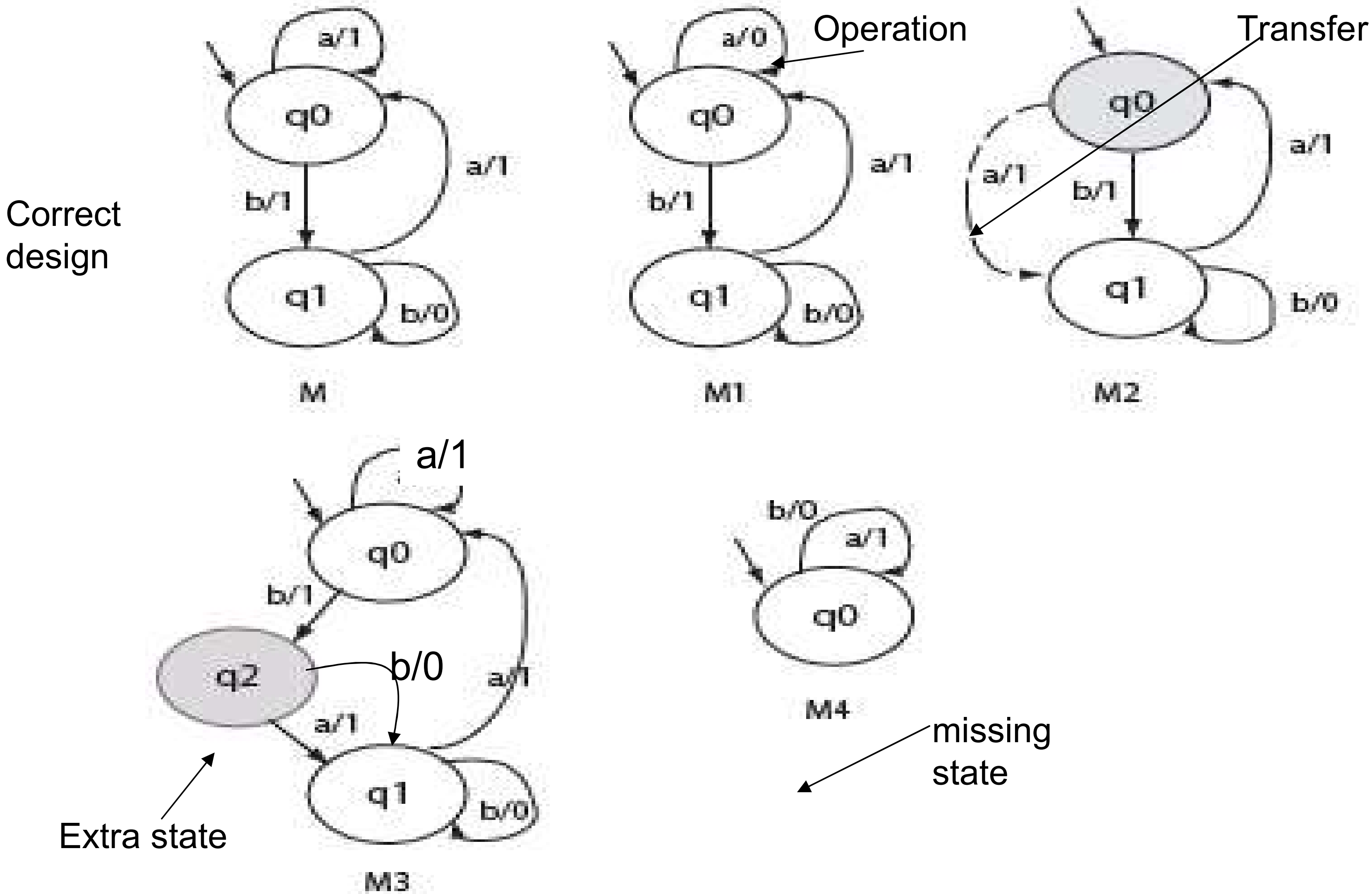


Completely specified,
not connected
(hence, not strongly connected)

(widely used) Fault models for FSM implementations

- *Operation error*
- *Transfer error*
- *Extra state error*
- *Missing state error*:

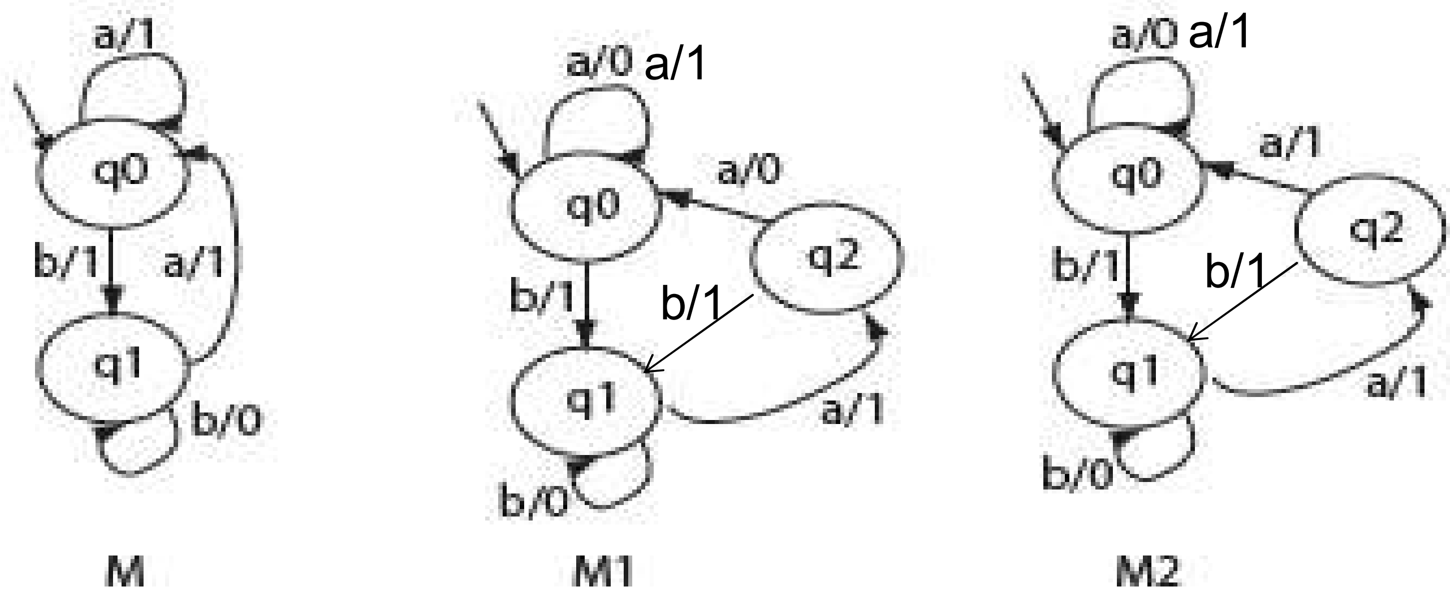
Example: FSM fault models



Equivalence

- Given a set V of input sequences, two states q and p are **V-equivalent** (written as "**q ≡(V) p**"), if q and p respond with identical output sequences to each input sequence in V.
  $\forall$ x(x$\in$V $\Rightarrow$ O(x,q)=O(x,p))
- Note. Q and p are **distinguishable by V** if not equivalent on V
- Two states q and p are **equivalent** (written as "**q ≡ p**), if they are V-equivalent for any set V.
  $\forall$ x(O(x,q)=O(x,p))
- Two FSMs **S and I are equivalent** if their initial states So and Io are equivalent.
- Two states q and p are **k-equivalent, k**≥1 if they are $\mathbf{X^k}$ equivalent

Example: extra state may or may not be an error



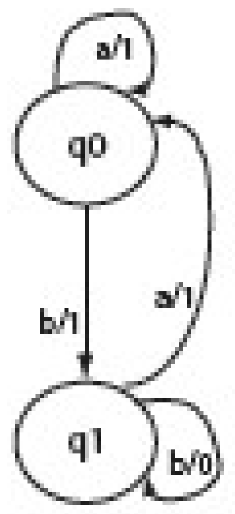
M1 and M2, extra state mutants of M

M1≢ M (non equivalnt FSMs), OM(abaa,q0)≠ OM1(abaa,q0)

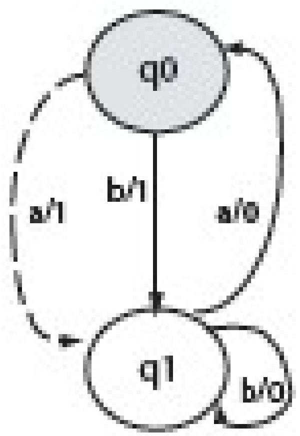
M2 ≡ M (they are equivalent) (How to prove?)

Mutants of a given FSM

- A mutant of a FSM specification is an FSM obtained by introducing one or more errors zero or more times: the errors introduced belong to a given fault model

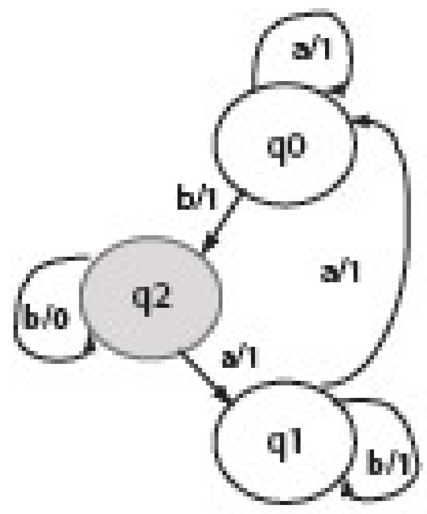


FSM M



Mutant M1

Operation and transfer errors



Mutant M2

Operation and extra state errors

How difficult is to find an error?

Take “C comments” FSM and consider a “transfer error mutant” replacing

T(2,/)=2 by T(2,/)=1

- Validation fails:
- //*a*/
  contains a comment but it is not printed!

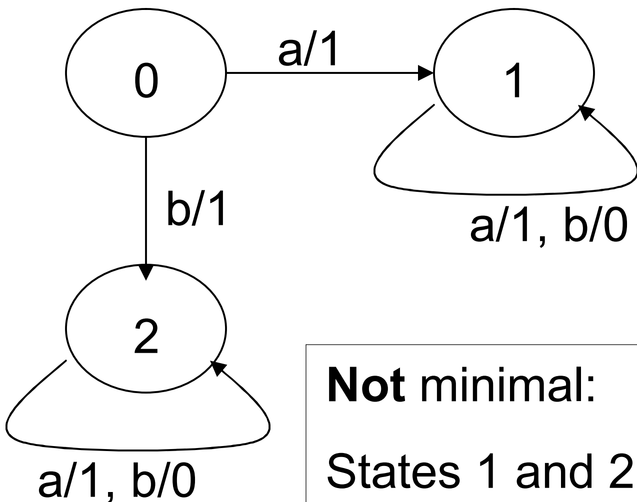
- Remark. The “error” is present in [Chow78]. Only 5% of my students catch the inadequacy!

How difficult id to prove a two FSM's equivalence?

- Prove that M2 ≡ M, for the extra state mutant M2
  - Show by induction on the length n≥0 of the input x:
    forall n(forall x (|x|=n implies
    O(q0,x)=O2(q0,x)=O2(q2,x) and
    O(q1,x)=O2(q1,x)
    ))
- See why the same proof fails considering the extra state mutant M1
- Remarks on proof ?

Minimal FSM

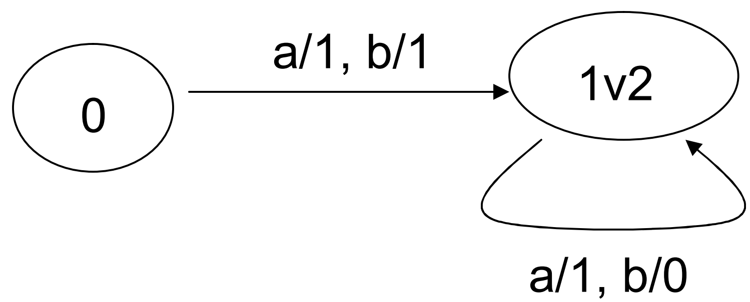
- An FSM M is **minimal** if the number of states in M is less than or equal to the number of states for any machine M' which is equivalent to M.



Not minimal:

States 1 and 2 **are equivalent**,

might be merged



Minimal:

(how to prove?

Note: States 0 and 1v2

are distinguishable (O(b,0)=1, O(b,1)=0)

If one state, then O(bb,q0)=11 or 00)

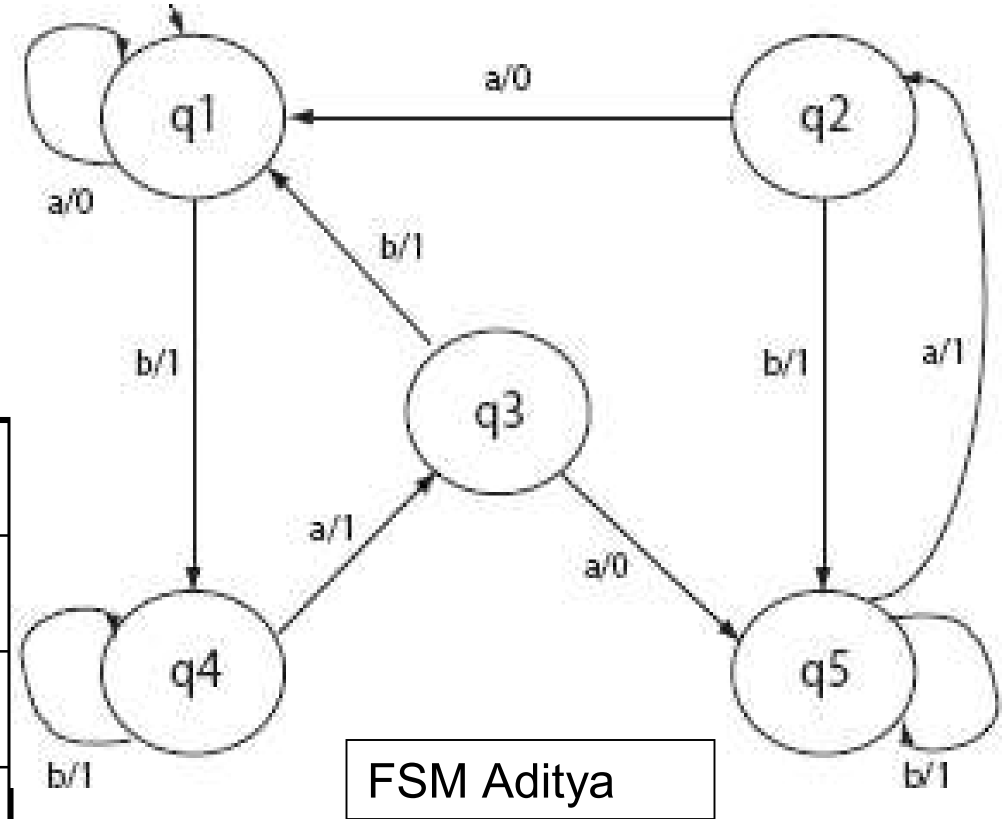
Characterization set

- Most methods for generating tests from finite state machines make use of an important set known as the **characterization set**.
- Let $M = (X, Y, Q, q0, T, O)$ be an FSM that is **minimal and complete**.
- A characterization set of M, denoted as W, is a finite set of input sequences that **distinguish the behavior of any pair of states** in M.
  $\forall p,q\ (p \neq q \Rightarrow \exists w \in W\ O(w,p) \neq O(w,q))$
  or
  $\forall p,q\ (p \neq q \Rightarrow p \not\equiv_W q)$

Example: characterization set

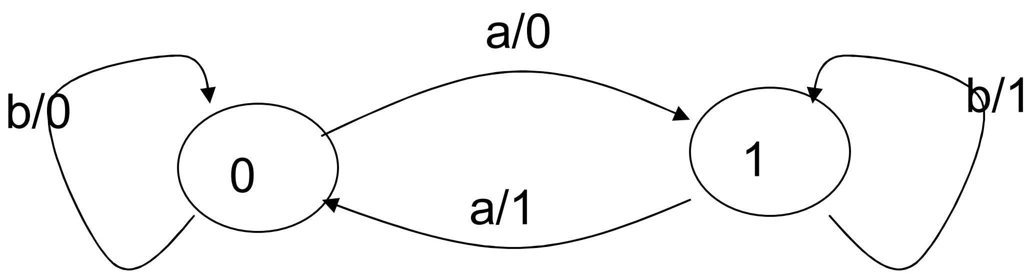
W = {a, aa, aaa, baaa}

	1	2	3	4	5
1	#	baaa	aa	a	a
2	#	#	aa	a	a
3	#	#	#	a	a
4	#	#	#	#	aaa
5	#	#	#	#	#



Note. All pairs are equivalent for shorter strings than those from W. ($1 \equiv (X^{<=3}) 2$ a.s.o.)

Might be more than one characterization set



W1={a}

W2={b}

Constructing a characterization set

The algorithm to construct a characterization set for an FSM M consists of two main steps.

1. The first step is to construct a sequence of *k*-equivalence partitions $P1, P2, \ldots Pk, \ldots Pm$, where $m \geq 1$
   - This iterative step converges in at most *n* steps where *n* is the number of states in M.
2. The W- procedure: in the second step these *k*-equivalence partitions are traversed, in reverse order, to obtain the distinguishing sequences for every pair of states.

k-equivalence partition

- Given an FSM $M = (X, Y, Q, q1, , O)$, a *k-equivalence partition* of Q, denoted by Pk, is a collection of n finite sets of states denoted as $Qk\ 1, Qk\ 2, \ldots, Qk\ n$ such that
- Union of Qkj is Q
- States in Qkj are k-equivalent,
- If $q \in Qki$, $p \in Qkj$ and $i \neq j$, then q and p are k-distinguishable

Construction of the *k*-equivalence partitions

- We illustrate using the FSM Aditya
- Computing P1 (the 1-equivalence partition)
  1. write the transition and output functions for this FSM in a tabular form
  2. regroup the states that are identical in their Output entries
  3. Construct P1 table

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

1. T and O of FSM, in tabular form

2. regroup

P1 has

Q11={q1,q2,q3}

Q12={q4,q5}

P1 table

3. Construct P1 table
   - copy the "Next State" sub-table
   - rename each Next State entry by appending a second subscript which indicates the group to which that state belongs.

Σ	Current state	Next state a	Next state b
1	q_1	$q_{1\,1}$	$q_{4\,2}$
	q_2	$q_{1\,1}$	$q_{5\,2}$
	q_3	$q_{5\,2}$	$q_{1\,1}$
2	q_4	$q_{3\,1}$	$q_{4\,2}$
	q_5	$q_{2\,1}$	$q_{5\,2}$

Important note.

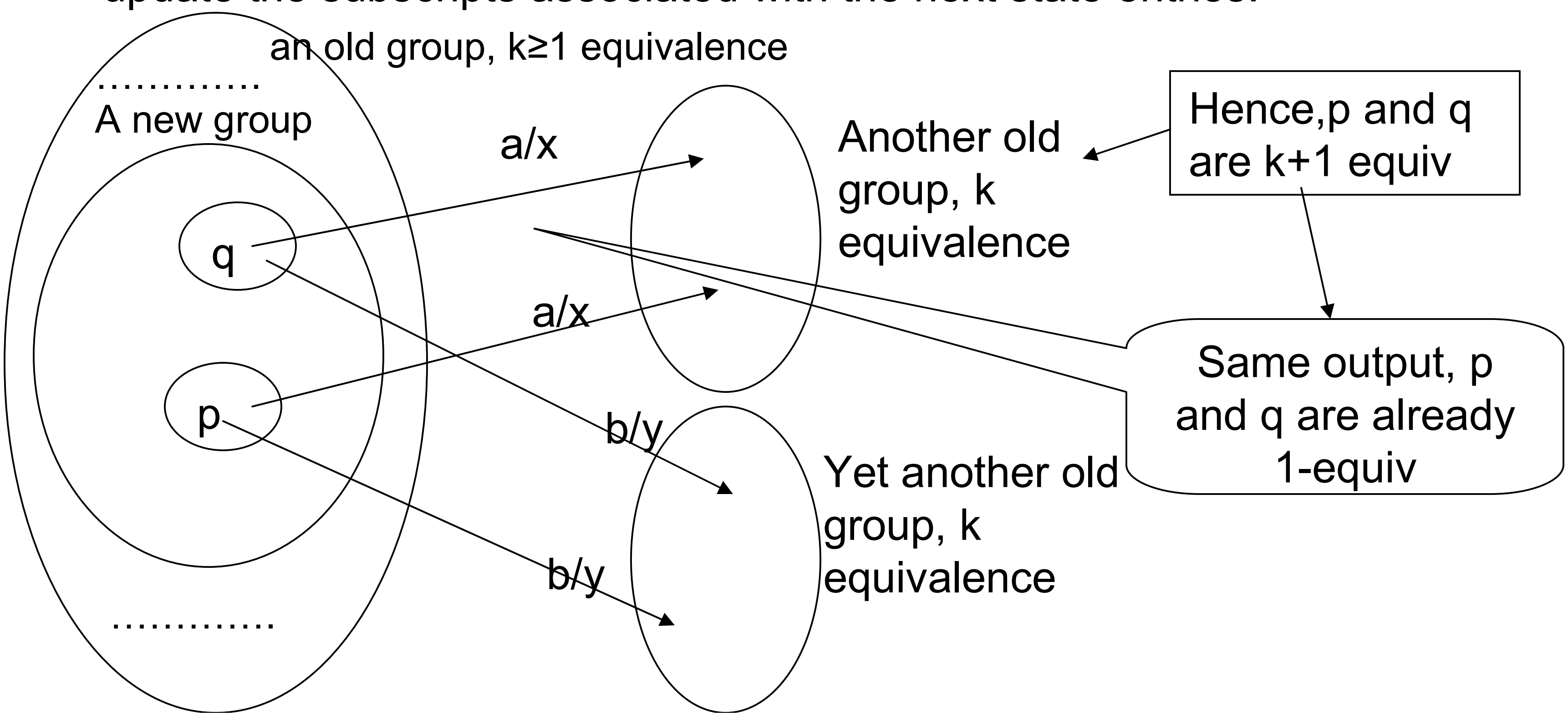
States in the same group

are 1-equivalent

P_1 table.

Construct P(k+1) table from Pk

- In every Pk group, regroup all rows with identical *second* subscripts in its row entries under the Next State column
  - states from the same new group, for every input, lead into states from the same old group (these are k-equivalent)
  - They are already 1-equivalent (were in the same group in P1)
  - Hence, states from the same new group are (k+1) equivalent
- relabel the groups
- update the subscripts associated with the next state entries.



P2 P3 tables

Σ	Current state	Next state	
		a	b
1	q_1	q_1 1	q_4 3
	q_2	q_1 1	q_5 3
2	q_3	q_5 3	q_1 1
3	q_4	q_3 2	q_4 3
	q_5	q_2 1	q_5 3

Σ	Current state	Next state	
		a	b
1	q_1	q_1 1	q_4 3
	q_2	q_1 1	q_5 4
2	q_3	q_5 4	q_1 1
3	q_4	q_3 2	q_4 3
4	q_5	q_2 1	q_5 4

P4 table

Σ	Current state	Next state	
		a	b
1	q_1	q_1 1	q_4 4
2	q_2	q_1 1	q_5 5
3	q_3	q_5 5	q_1 1
4	q_4	q_3 3	q_4 4
5	q_5	q_2 2	q_5 5

There are no distinct
4- equivalent states
There is, any pair of
distinct states can be
distinguished by an
input of length 4

The W- procedure

```
{P1,… Pn is the set of k-equiv partitions}
W=∅;
forall p≠q  do
    if ∃r(1≤r<n ∧ (p,q) equiv in Pr but not equiv in Pr+1) then
        choose such an r; //(p,q) r –equiv but not (r+1)-equiv
        z=ε; p1=p; p2=q;
        for m=r downto 1 do
            choose x  in X such that G(p1,x)≠G(p2,x) in Pm;
            z=zx;
            p1=T(x, p1); p2=T(x, p2)
        end for
        choose x such that O(x,p1) ≠O(x,p2);
        z=zx;
        W=W u {z};
    else {(p,q) not equiv in P1} do
        find x in X such that O(x,q) ≠O(x,p);
        W=W u {x}; // x distinguishes q and p
    end if
end forall
```

G(q , x) denotes the label of the group to which the machine moves when excited using input *x in state q . For example, in the table for P3 G(q2, b) = 4* and *G(q5, a) = 1.*

Example: W procedure

For q1 and q2 we may find baaa, baba etc.

(r=3; part if …then of the W procedure)

For q3 and q4 we find a

(no r; part if…else of the W procedure)

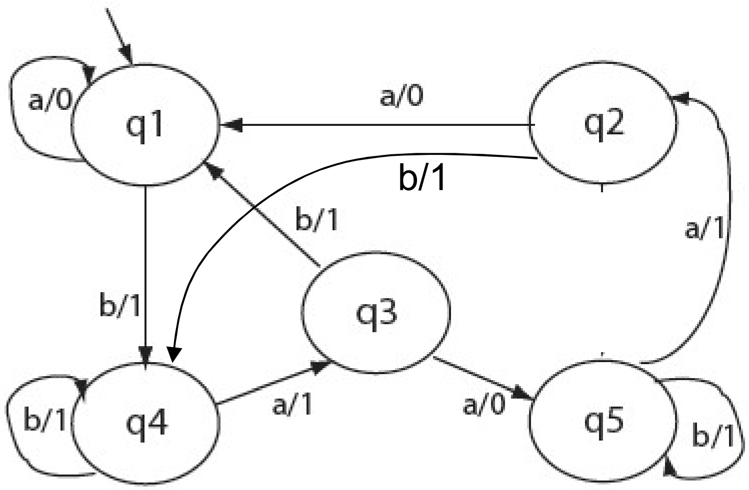
For q4 and q5 we may find aaa or aba.

(r=2; part if …then of the W procedure)

For q2 and q3 we may find aa or ba.

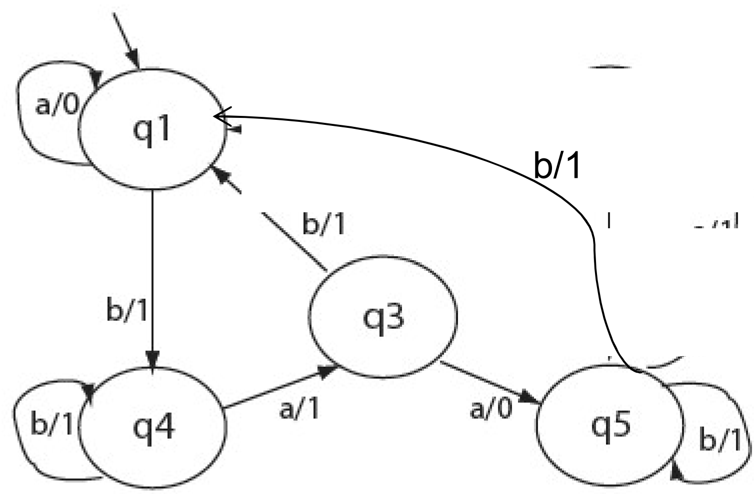
(r=1; part if …then of the W procedure)

Exercise



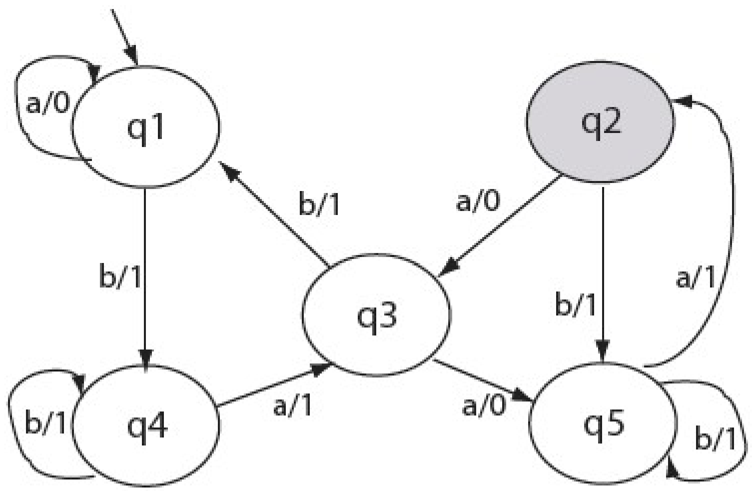
1.Construct Pk equivalence tables
2. Find an equivalent minimal FSM.

A.
1. We find P3=P4 and q1≡q2
2. The minimal FSM is shown bellow.



Exercise

- Construct Pk equivalence classes.
- Find a characterization set

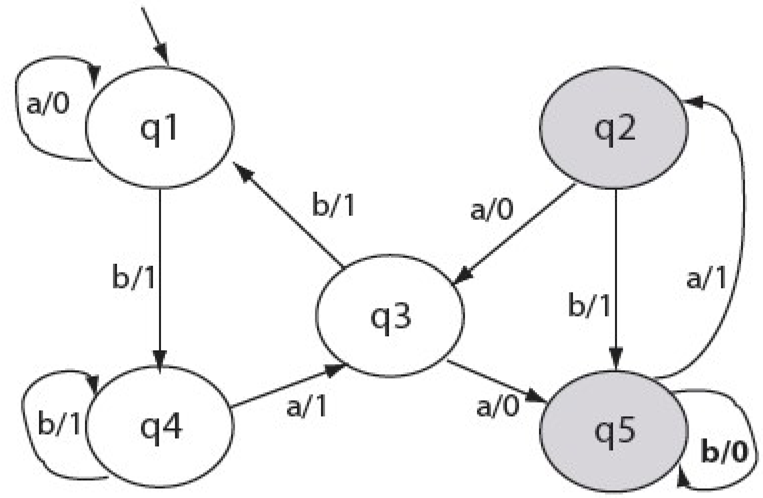


A.

- We find P3=P4, |P3|=5. The FSM is minimal
- W={aaa}

Exercise

- Construct Pk equivalence classes.
- Find a characterization set



A.

- We find P2=P3, |P2|=5. The FSM is minimal
- W={a,b,ba,bb}

Identification sets

- Analogous to the characterization set for M, we associate an *identification* set with each state of M.
- An identification set for state *q* is denoted by *Wq* and has the following properties:
  - Wq $\subseteq$ W
  - Wq is minimal with respect to
    $\forall p \; (p \neq q \Rightarrow \exists s \in Wq \; O(s,p) \neq O(s,q))$

EXAMPLE. Consider the machine FSM Aditya: characterization set and its W shown in the table. From the table we deduce: W1=W2={baaa,aa,a}: W3={a,aa}; W4=W5={a,aaa}

- While the characterization sets are used in the W-method, the *Wi* sets are used in the Wp method.

The W-method (Chow)(1/?)

- The W-method is used for constructing a test set T from a given FSM specification S and knowing some information about the IUT I.
- $S \equiv (T)\ I \Rightarrow S \equiv I$
- The method makes some assumptions about the specification S and the IUT I.

Chow´s W method assuptions

Hypothesis and assumptions:

- S is deterministic and minimal
- I is *assumed* to be deterministic
- S is **completely** specified and I is *assumed* to be as well
- all states in S are reachable and those for I are *assumed* to be as well
- The number of states in I is *assumed* to be bounded by an integer *m*, which may be larger than the number *n* of states in S

The W methods provides a test set **T=P•Z**, where

- P is a transition cover of S
- Z= ({e} ∪ X ∪ ... X(*m-n times*)) •W
- W is a characterization set of S

State cover, transition cover

- Let Q be a set of input sequences. Q is a **state cover** set of S if for each state q of S, there is an input sequence x in Q such that T(x,q0)=q.
  - For the initial state qo, we have T(ε,q0)=qo. The empty input sequence belongs to Q.
  - Note: In many cases, one uses a state cover set that is closed under the operation of "selecting a prefix").
- Let P be a set of input sequences. P is a **transition cover** set of S if:
  - Q⊆P, Q a state cover
  - for each transition p-x/y->q, there are sequences w and wx in P such that T(w,q0)=p and T(x,p)=q.
- The empty sequence is a member of P.
- By definition, each transition cover set P contains a subset

which is also a state cover set.

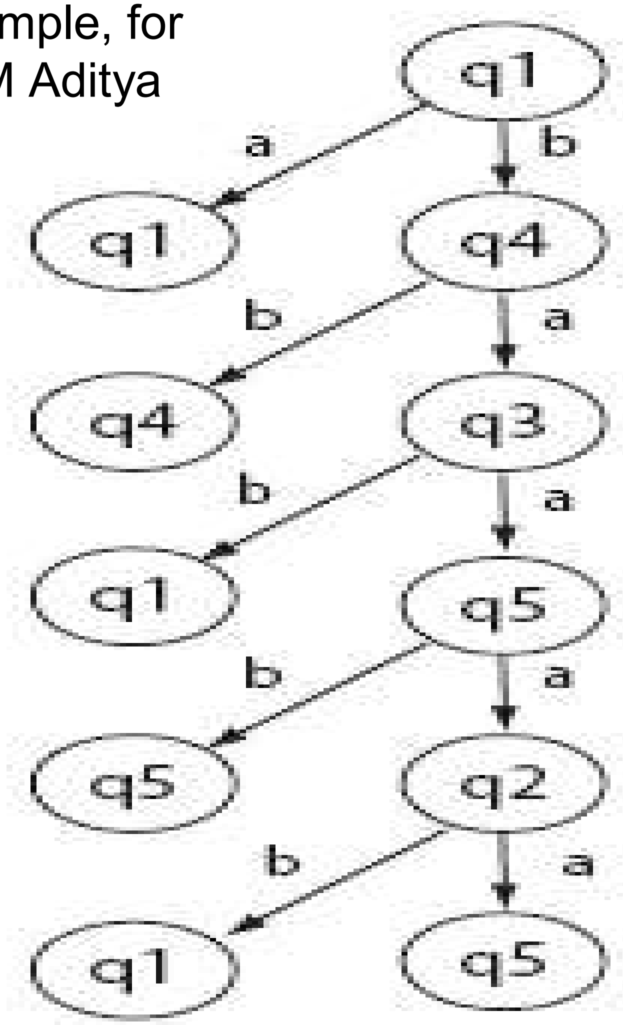
- The set of all partial paths in the testing tree of S, as defined in [Chow 78], is a transition cover set. A procedure for the construction of this set is also given there.

Computation of the transition cover set (1/2)

- We can construct a transition cover set *P* using the "testing tree" of M.
- A testing tree for an FSM is constructed as follows.
  - State *q*0, the initial state, is the root of the testing tree. This is level 1 of the 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 T(*n, x*) = *m* for *x in X*. This branchis labeled as *x* . This step is repeated for all nodes at level *k*.
- Once a testing tree has been constructed, we obtain the transition cover set *P* by concatenating labels of all *partial paths* along the tree.

Testing tree

Example, for
FSM Aditya



P =

{ε,

a, b,

bb, ba,

bab, baa,

baab, baaa,

baaab, baaaa}

Thus exciting an FSM with elements of *P* ensures that
•**all states are reached**, and
•**all transitions have been traversed** at least once.

Constructing *Z*

Suppose that the number of states estimated to be in the IUT is *m* and the number of states in the design specification is *n*, ***m≥n***. Given this information we compute *Z* as:

$$\boldsymbol{Z = X[m - n].W},$$

for ***m* = *n*,** i.e. when the number of states in the IUT is the same as that in the specification.

$$\boldsymbol{Z = \mathrm{X}.W}$$

For ***m* < *n*** we still use
Z* = *X.W.

$$Z = (X^0.W) \cup (X.W) \cup (X^2.W) \ldots \cup (X^{m-1-n}.W) \cup (X^{m-n}.W)$$

The W-method (Chow)(2/?)

Let be S having n states. W-method consists of the following sequence of steps.

- Step 1 Estimate the maximum number m of states in the IUT.
- Step 2 Construct the characterization set *W* for the given machine S.
- Step 3 Construct the testing tree for S and from it determine the transition cover set *P*.
- Step 4 Construct set Z=X[m-n]W
- Step 5 *P.Z* is the desired test set.

Deriving a test set

The test set: ***T =P.Z.***

Example (FSM Aditya).

- For m=n=5,

$Z = X^0.W = \{a, aa, aaa, baaa\}$

$T = P.Z =$

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

{a, *aa*, *aaa*, *aaaa*, *abaaa*, *ba*, *baa*, *baaa*, *baaaa*, *baaaaa*, *baaaaaa*, *baaaaaaa*, *baaaabaaa*, *baaaba*, *baaabaa*, *baaabaaa*, *baaabbaaa*, *baaba*, *baabaa*, *baabaaa*, *baabbaaa*, *baba*, *babaa*, *babaaa*, *babbaaa*, *bba*, *bbaa*, *bbaaa*, *bbbaaa }*

For m=6,

$Z = W \cup X.W =$

{a, aa, aaa, baaa, aa, aaa, aaaa, abaaa, ba, baa, baaa, bbaaa}

Testing using the W-method

To test the given IUT M*i* against its specification S, we do the following for each test input.

1. Find the expected response $S(t)$ to a given test input t. This is done by examining the specification. Alternately, if a tool is available, and the specification is executable, one could determine the expected response automatically.
2. Obtain the response $I(t)$ of the IUT, when excited with t in the initial state.
3. If $S(t) = I(t)$ then no flaw has been detected so far in the IUT.

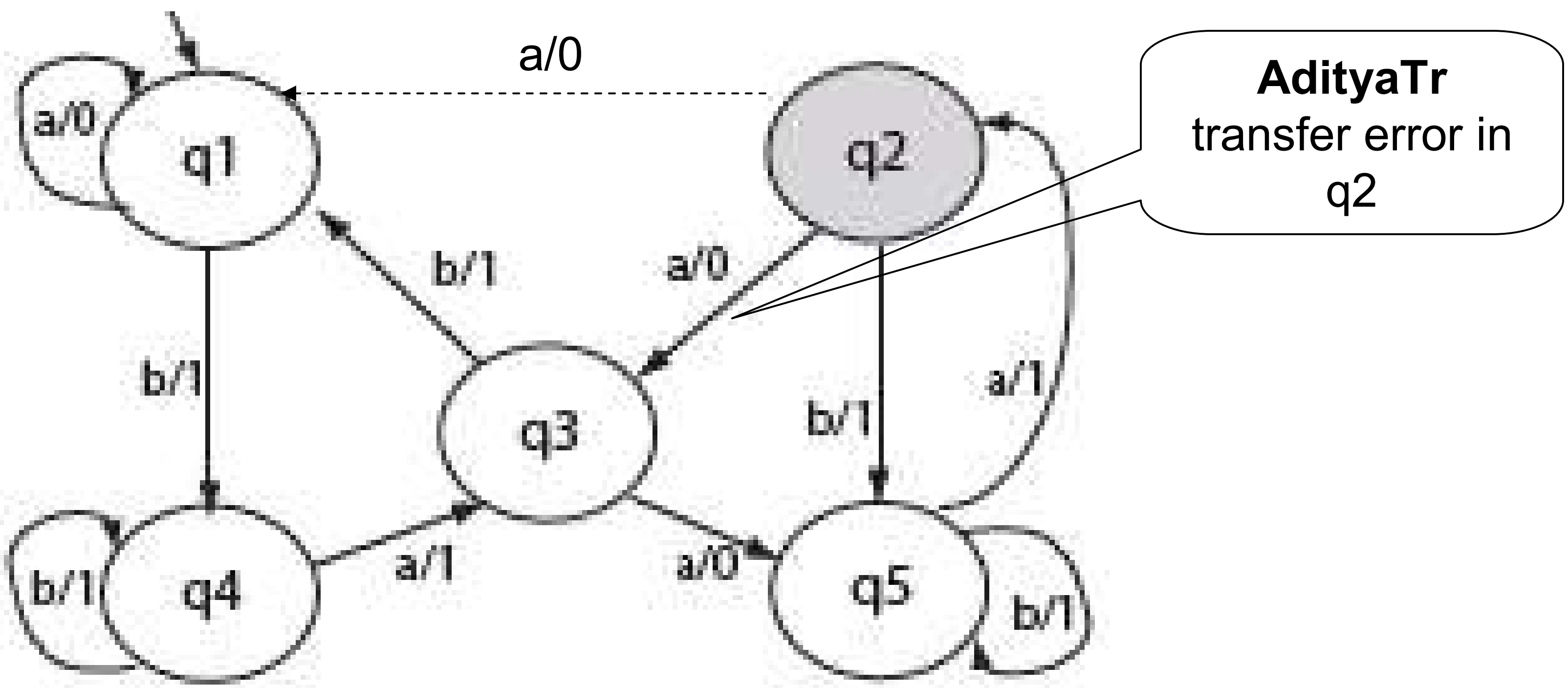
$S(t) \neq I(t)$ implies an error in the design or the IUT under test

Example, testing with W- method

transfer error

See some C implementation here

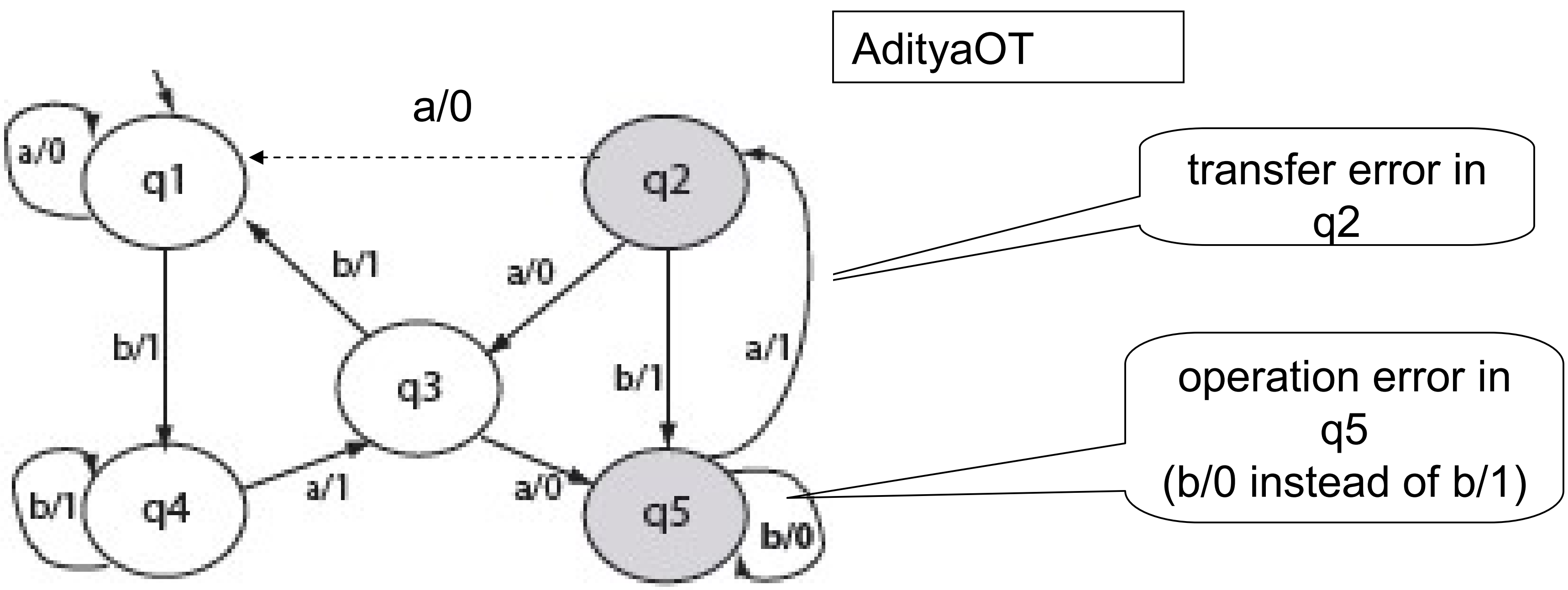
- S is Aditya.
- I is below, one error
  With *t* = *baaaaaa we have*S(*t*) = 1101000 and I(*t*) = 1101001. Thus the input sequence *baaaaaa* has revealed the transfer error in IUT.



Example, testing with W- method

operation + transfer error

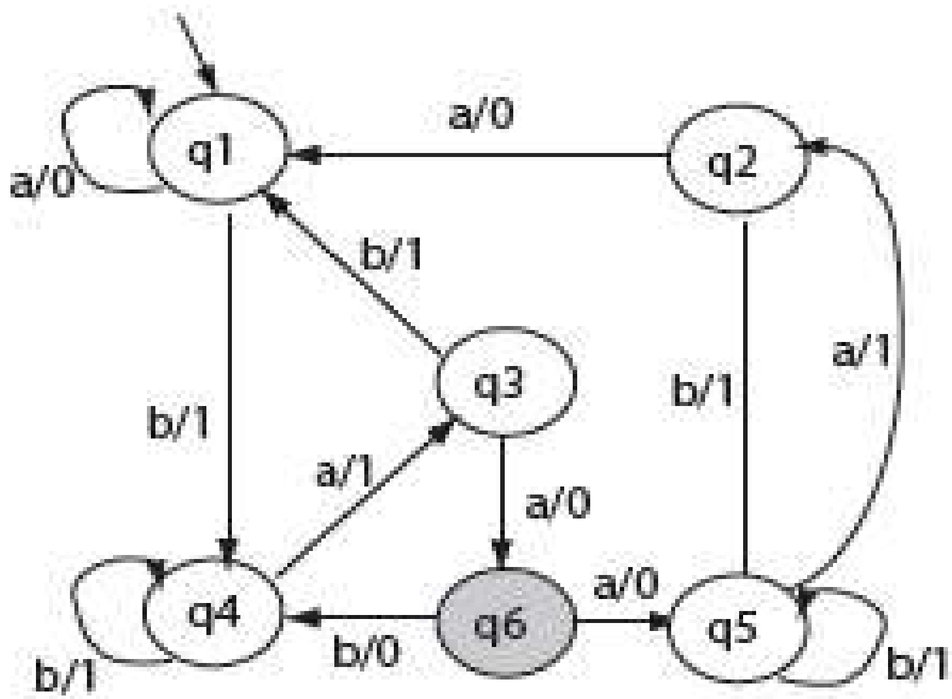
- S is Aditya.
- I is below, two errors.
  With *t* = *baaba we have* S(*t*) = 11011 and I(*t*) = 11001 and operation error is revealed
- With *t* =*baaaaaa, transfer error, already shown.*



Example, testing with W- method

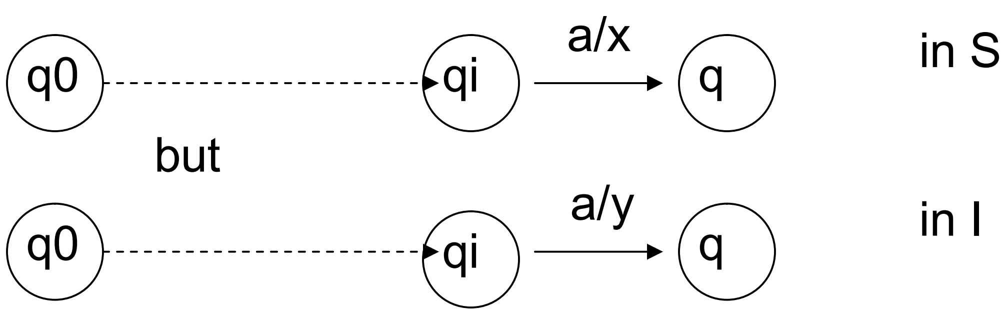
extra state error

- S is Aditya.
- I is below, one error.
  With *t* = *baaba we have* S(*t*) = 11011 and I(*t*) = 11001 and extrastate error is revealed

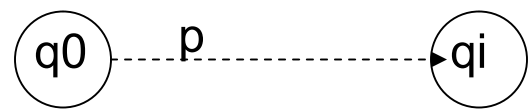


The error detection process (1/2)

- m=n, T=PW
- examine carefully how the test sequences generated by the W-method detect operation and transfer errors.
- Operation error. Let us suppose that:



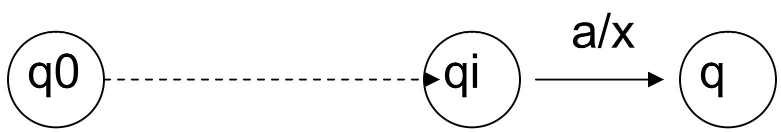
In P (transition cover) there exist p such that



We have also pa in P, by def, thus operation error is detected when testing paw for a w in W.

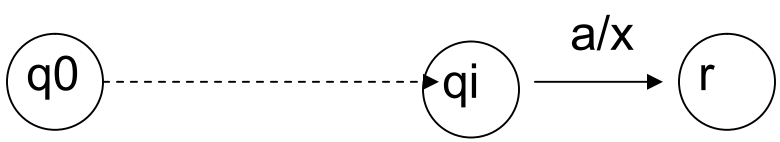
The error detection process (2/2)

- m=n, T=PW
- Transfer error. Let us suppose that:



in S

but



in I, and q≠r

There is w∈W such that O(w,q)≠O(w,r) . Then paw ∈PW and the error is revealed