Testing per FSM

Angelo Gargantini

Linguaggi di Programmazione per la Sicurezza – Crema Unimi

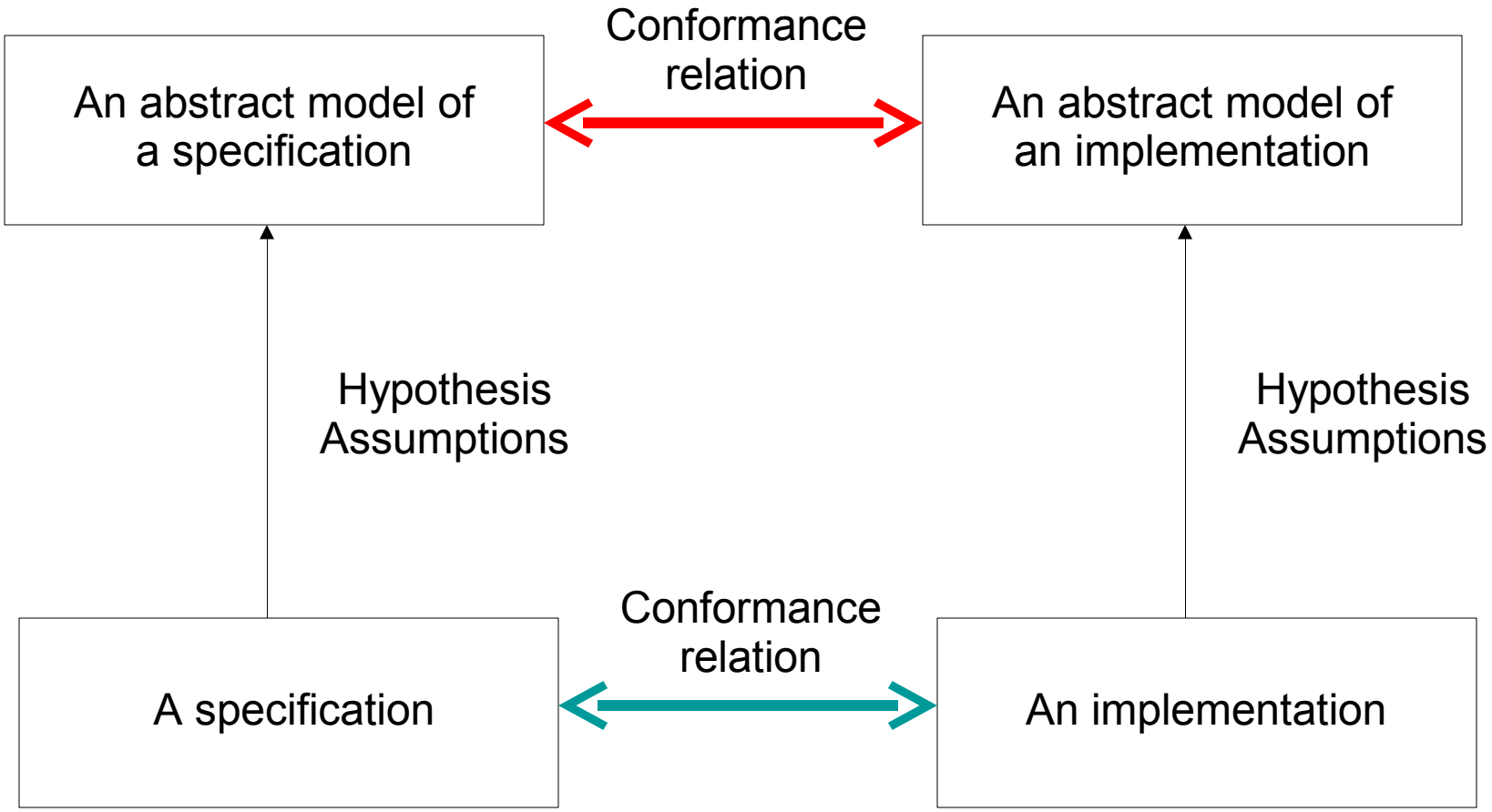
Materiale

- Questi lucidi
- motres, capitolo 4
  - motres4.pdf

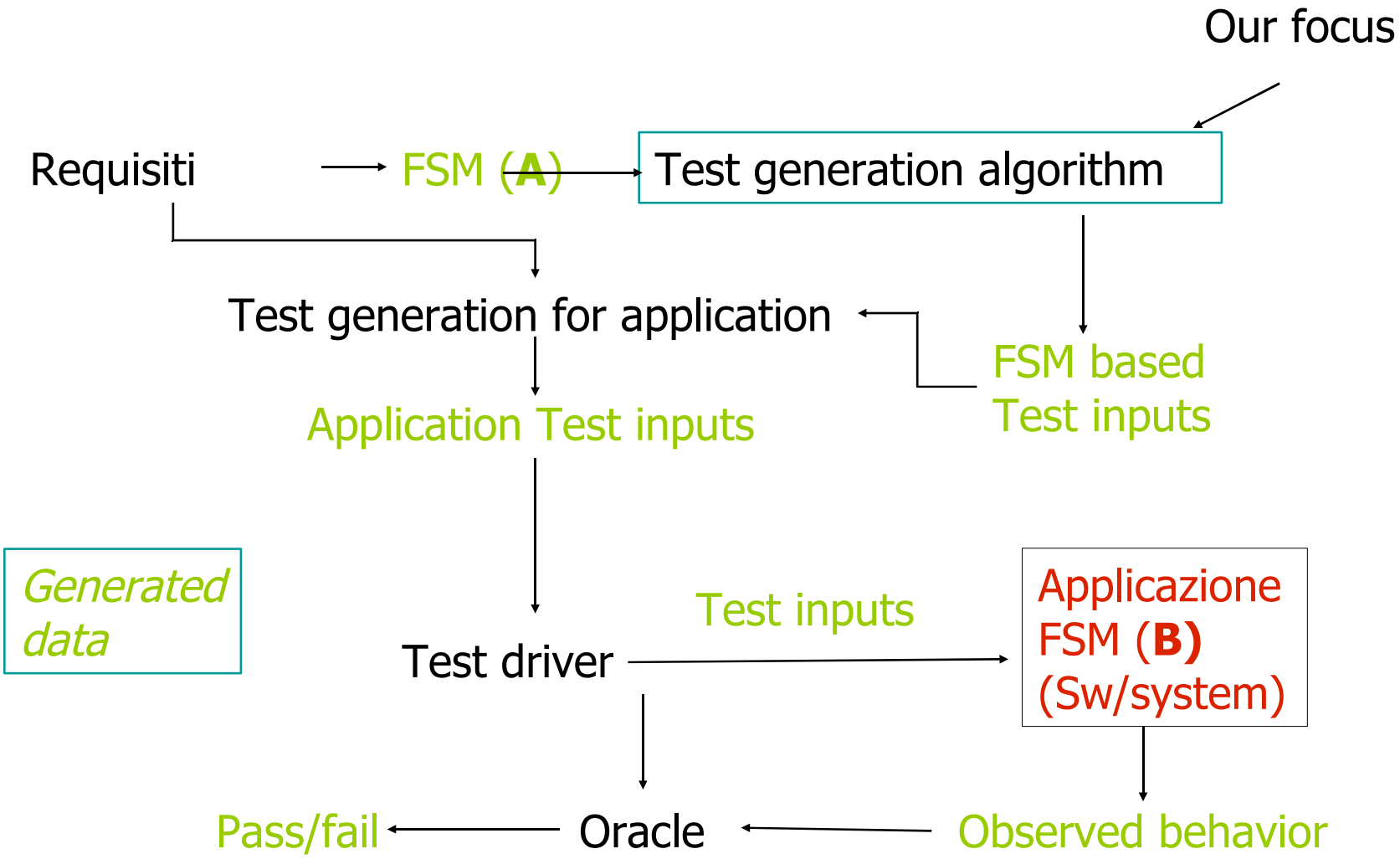
Conformance testing problem

- Given:
  - Complete information of specification machine **A** (states, transition and output function)
  - Implementation machine **B**, black box, only I/O is observable
- Goal:
  - Determine whether B is correct implementation of (conforms to, is equivalent to) A by applying a test sequence to B and observing the output.

Testing Hypothesis/Assumptions



Test generation per FSMs



Conformance testing assumption

- Specification A is strongly connected
  - It must be possible to reach all states
- A is reduced (minimized)
  - We can determine equivalence only to minimized machine, since equivalent states are not distinguishable.
- B does not change during experiment and has the same input alphabet as A
- B has no more states than A
  - Assume, faults do not increase number of states, only:
    - Wrong output on transition
    - Wrong state in transition destination

Assumptions (2)

- Completely-specified:
  - Rispondono ad ogni input in ogni stato
- Deterministic
- Initialized
- What if a FSM is not
  - Minimized?
  - Completely-specified?
  - Deterministic?
  - Strongly-connected?
  - Initialized?

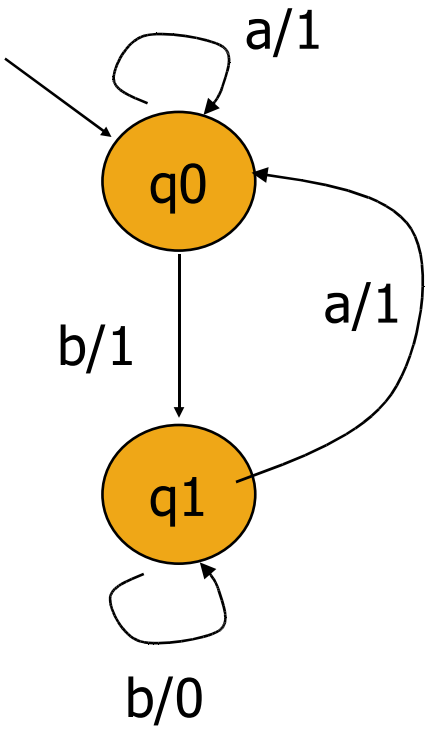
Assumptions (3)

- Assumptions about implementations
  - A fault model is a hypothetical model of what types of faults may occur in an implementation
  - Usually based on mutations
- What if we cannot have fault models?
  - There are an infinite number of faulty implementations

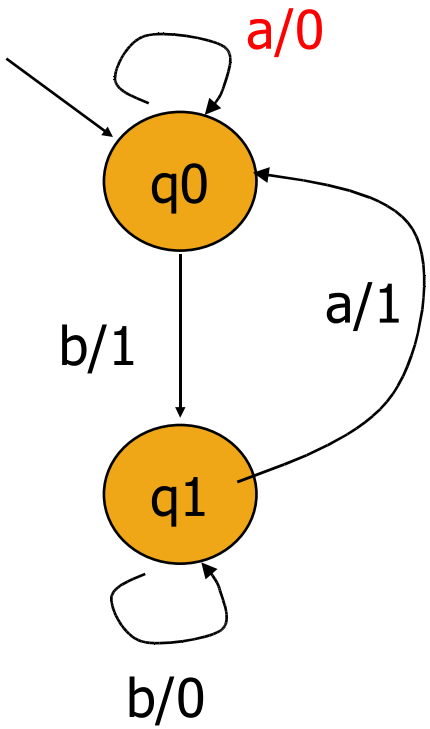
Fault Models (1)

- Fault models for FSM
  - Output faults
  - Transfer faults
  - Additional or missing transitions
  - Other faults (not considered)
    - Additional or missing states
    - Transfer faults with additional states

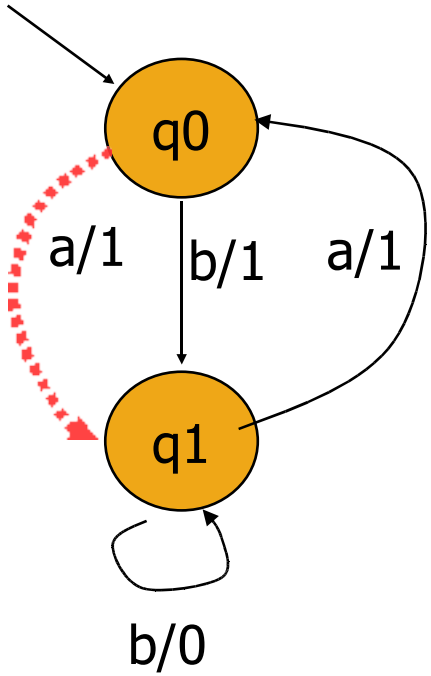
Fault model



Correct design

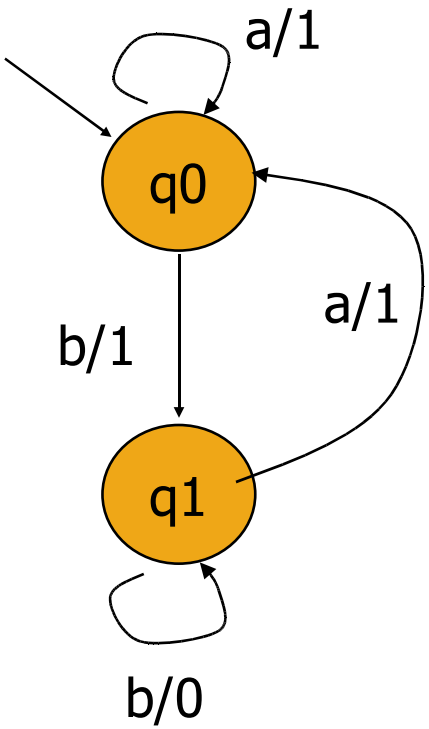


Operation error

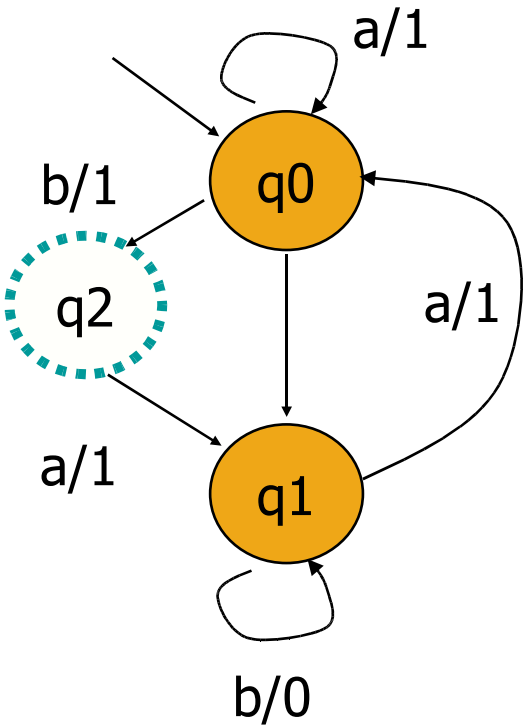


Transfer error

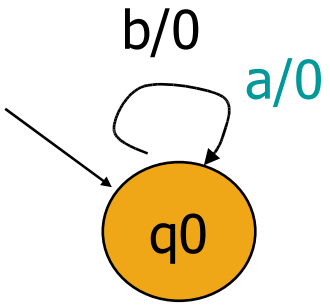
(Fault model)



Correct design



Extra state error

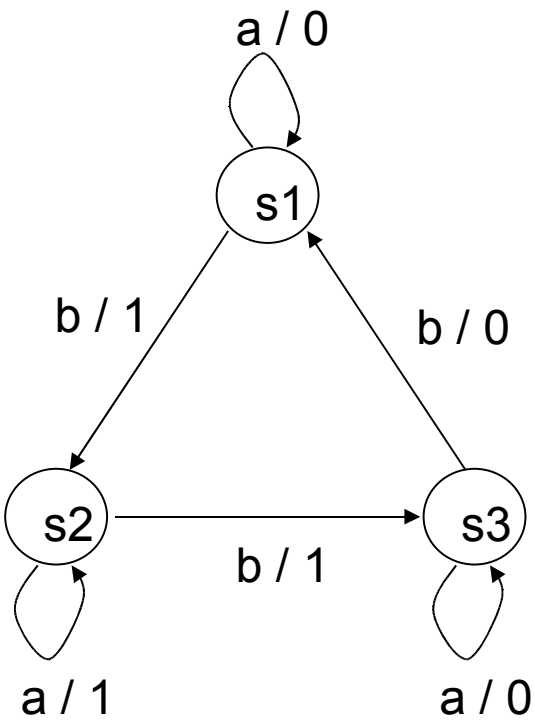


Missing state error

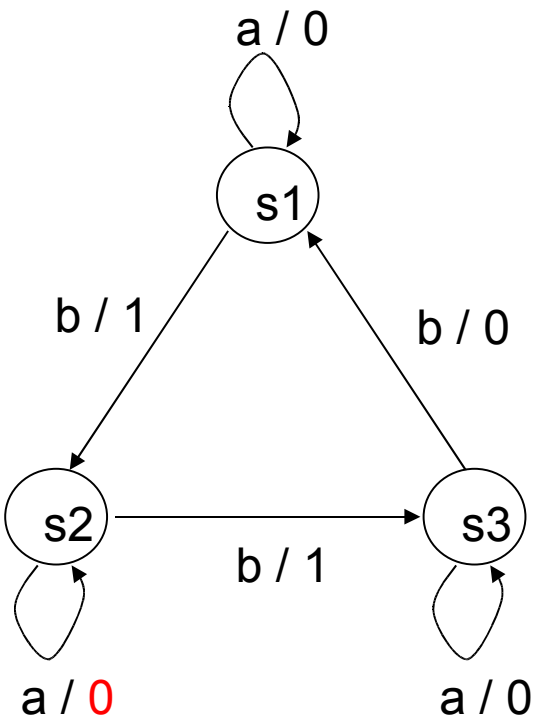
Fault Models sull'intera FSM

- Output faults

< A: specification >



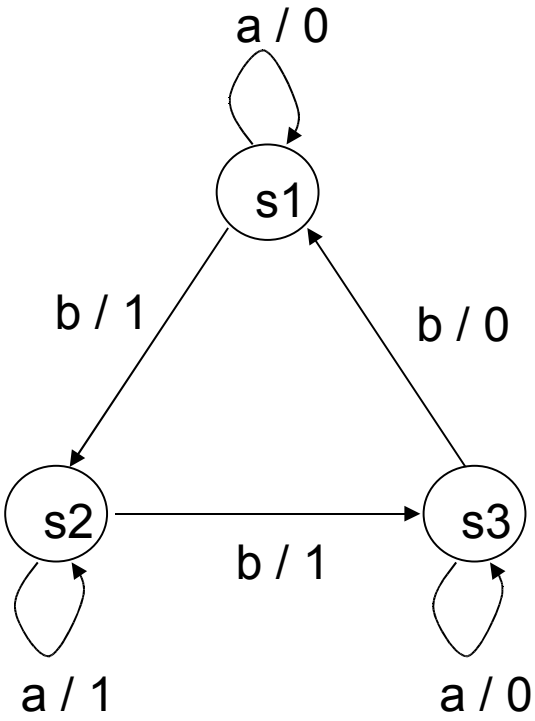
<B: implementation >



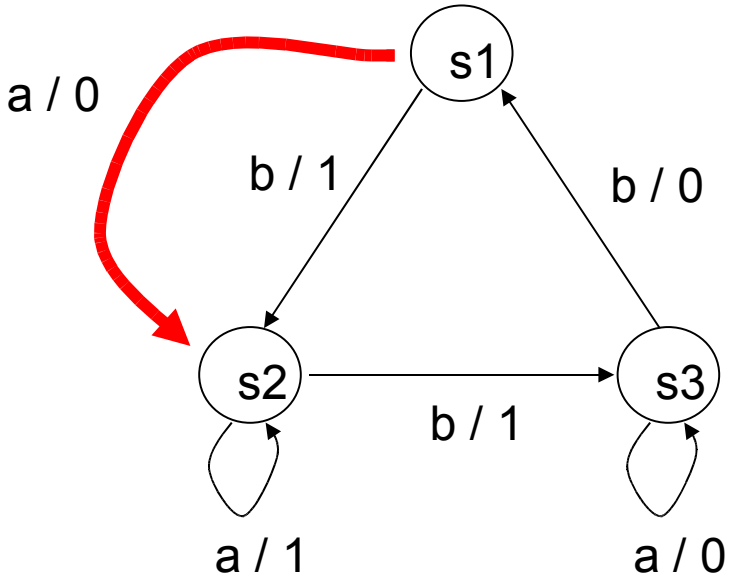
Fault Models (3)

- Transfer faults

< specification >

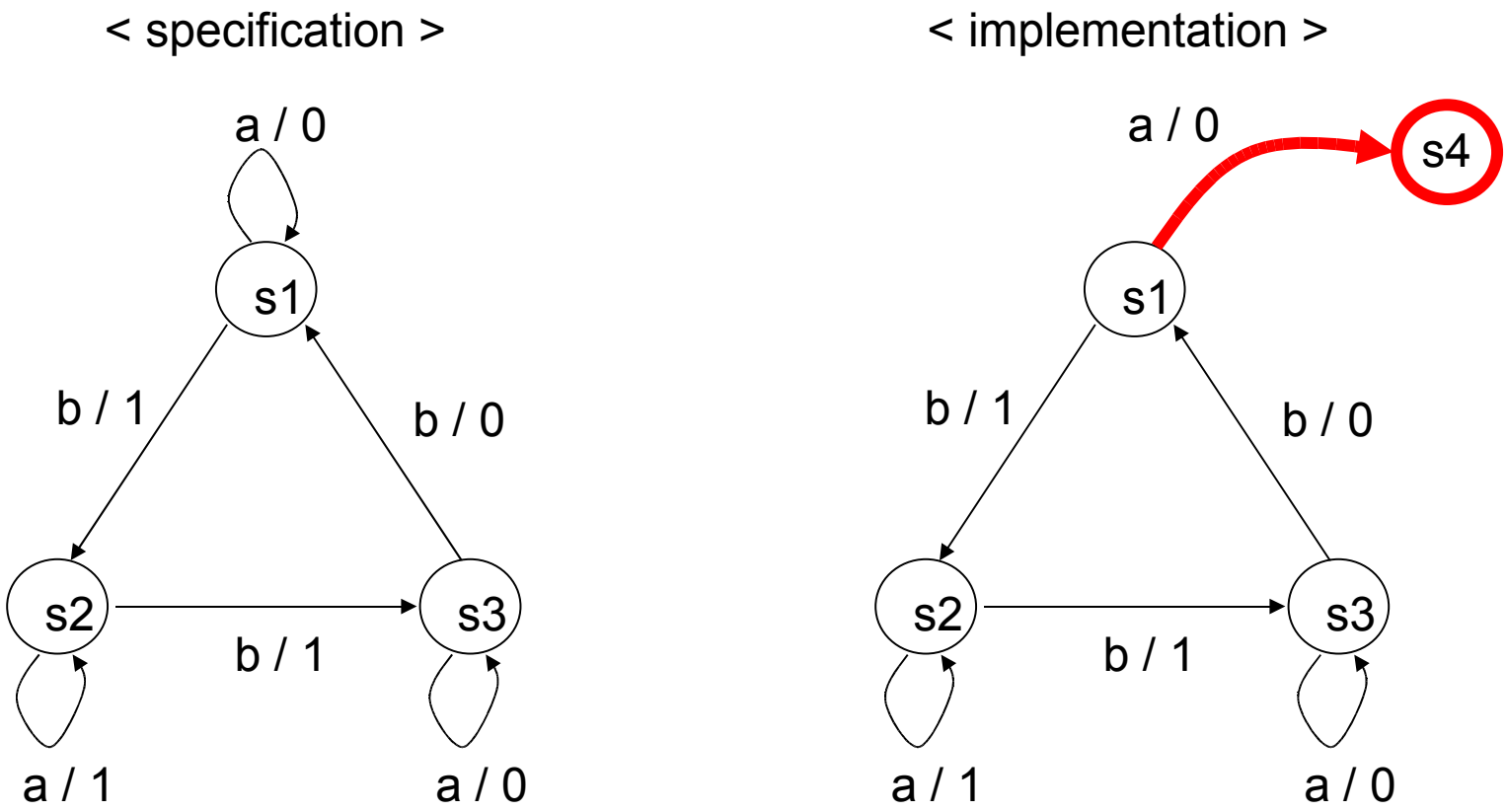


< implementation >



Fault Model (4)

- Transfer faults with additional states

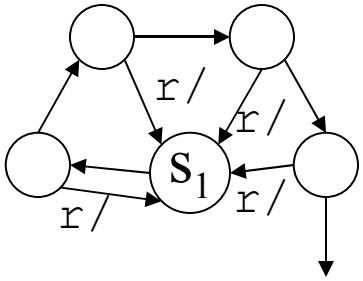


Fault Models (5)

- Fault models for software
  - Sequencing faults
  - Arithmetic and manipulative faults
  - Calling wrong functions
  - Wrong specification of data type
  - Wrong values
  - Wrong number of variables
  - Wrong operators
  - ……

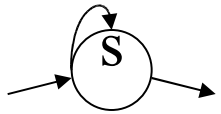
Status messages and Reset

- FSM has a reset capability if special input r takes the machine from any state to initial s1.
- If r input is defined for all states of B then reset is reliable.



- Status message tells the current state of machine without changing it.
- Reliable status message guarantees that state will stay the same as before message.

status/[state]



Metodi e loro capacità

- Con status message
  - State cover method
  - TT-method (transition tour)
    - trova tutti i fault
- Senza status
  - TT-method
    - Garantisce di trovare tutti gli output fault
  - DS-method (distinguishing sequences)
  - W-method (characterizing sets)
  - UIO-method (UIO sequences)

per questo spesso si aggiungono degli status messages in sistemi embedded

Struttura del test di conformità

- Inizializzazione: move to some known state s1:
  - usa reset
- verifica di similarità tra B e A
  - B ha uno stato che “risponde” come quello di A
    - usa status message , DS, UIO o altro
    - fallo per ogni stato di A
- verifica delle transizioni: per ogni transizione verifica $\delta(si, a)=sj$:
  - applica la sequenza che muove la macchina a si
  - applica a (e controlla l'output)
  - verifica lo stato finale che sia sj

State cover

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

Con status esempio

- Considera il CDPlayer
  - Aggiungi l'output
    - Se premi play e non c'è il CD fa beep
    - Se premi play e c'è il CD si illumina una luce
    - ...
  - Assumi di avere uno status
  - Copertura di tutti gli stati
  - Copertura di tutte le transizioni

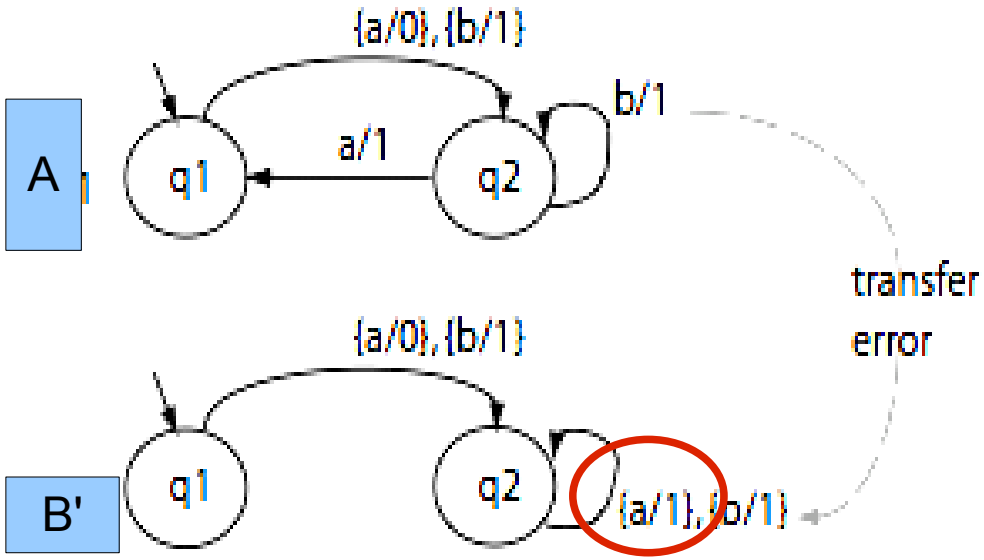
Senza Status message

- Con lo status message:
  - State cover non trova comunque tutti gli errori
    - Se ho una transizione sbagliata e non la copro non me ne accorgo
  - Transition cover scopre ogni errore
- Cosa succede se non c'è status?
    - Se non c'è status devo capire in quale stato mi trovo semplicemente osservando solo gli output

Transition coverage

Esempio di errori non trovati senza status message

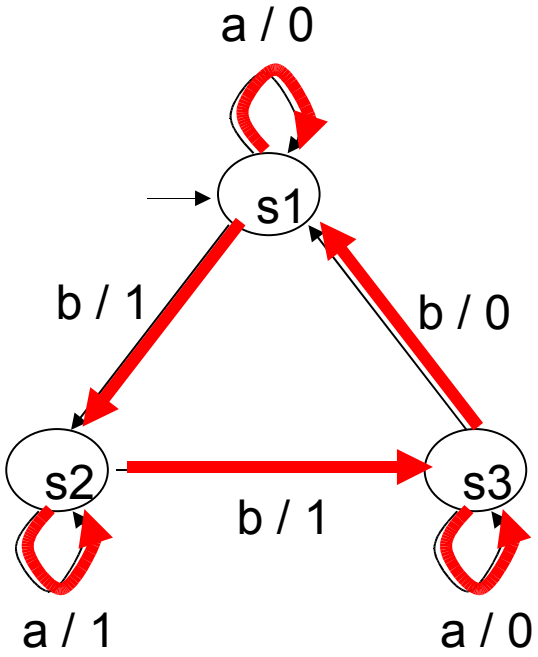
Consider the following machines, a correct one (A) and one with a transfer error (B').



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

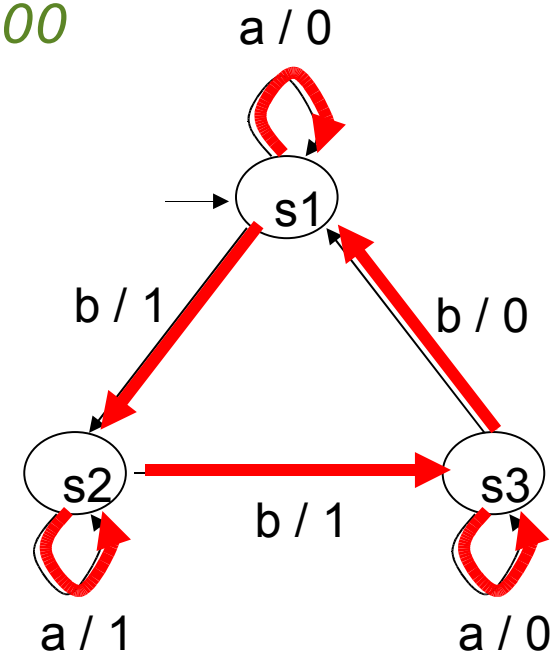
TT-method (1)

- Il metodo che ottiene il transition coverage è il Transition Tour method
- A transition tour of a FSM
  - A path starting at the initial state, traverses every transition at least once, and returns to the initial state



TT-method (2)

- From a transition tour, we identify a test suite consisting of an input sequence and its expected output sequence
  - The input sequence *ababab* and
  - Its expected output sequence *011100*



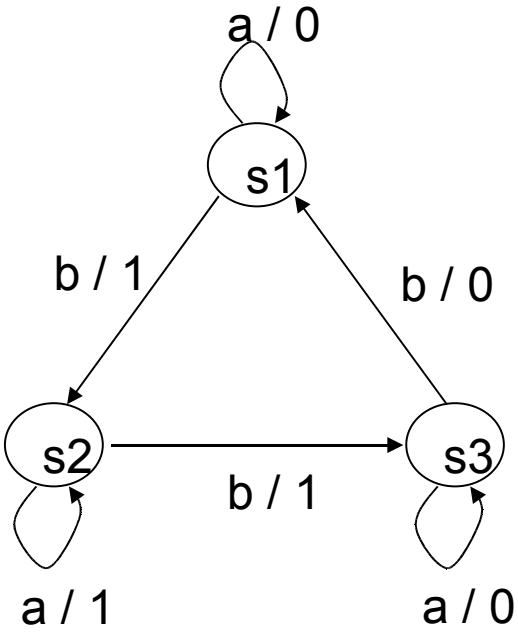
Come calcolare un transition tour

- problema classico dell'attaversamento dei grafi
- il tuor più corto si chiama “Euleriano”
- per macchine simmeriche (tanti archi uscenti quanti entranti è semplice)
  - vedi corso di algoritmi
  - intuitivamente, su ogni nodo percorri le transizioni che finiscono sullo stesso nodo e poi percorri le transizioni in uscita che non hai ancora percorso
- noi: TT anche non euleriano

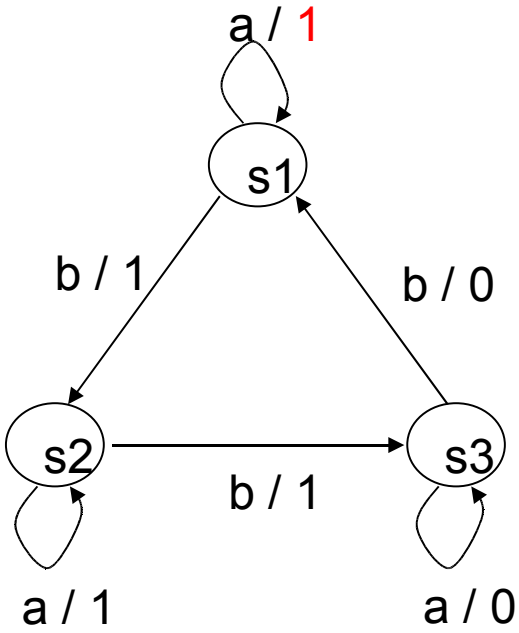
TT-method (3)

- Transition tours can find all output faults

< specification >
Input sequence: *ababab*
Expected output sequence: *011100*



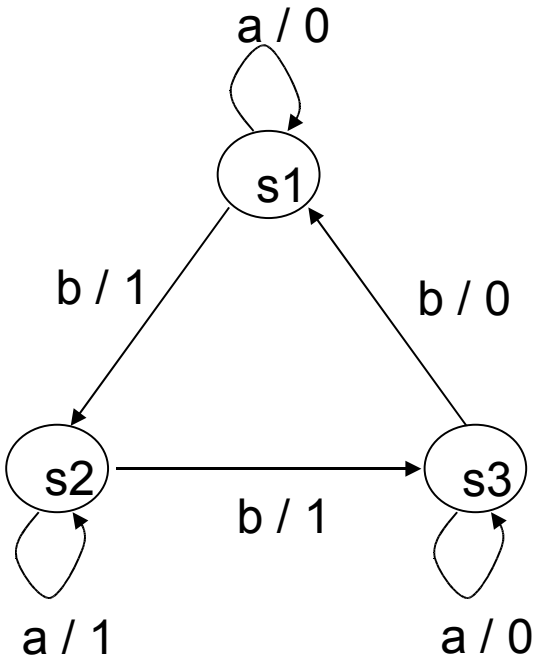
< implementation >
Input sequence: *ababab*
Observed output sequence: *111100*



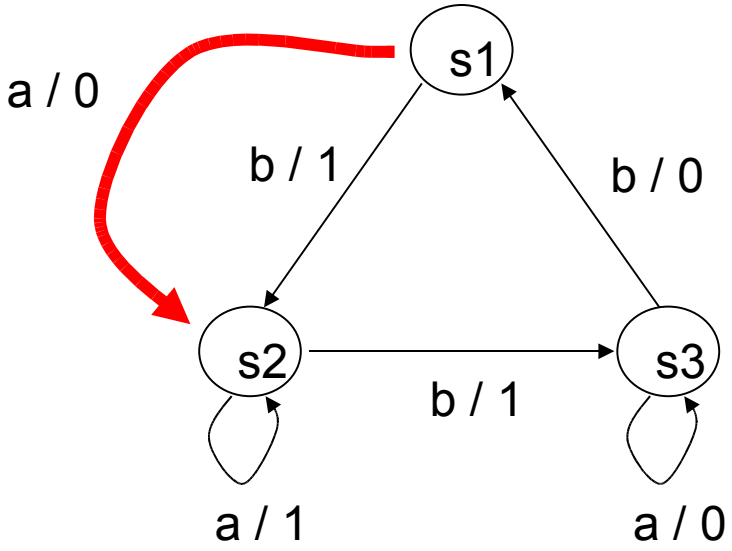
TT-method (5)

- Transition tours trova alcuni difetti:

< specification >
Input sequence: *ababab*
Expected output sequence: *011100*



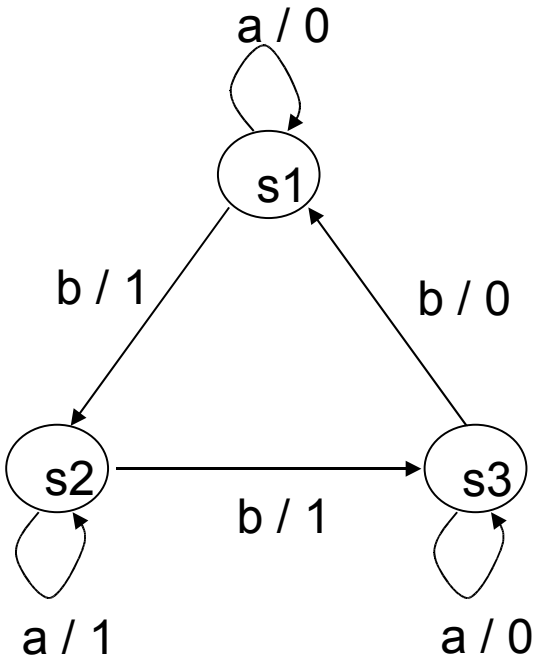
< implementation >
Input sequence: *ababab*
Observed output sequence: *010001*



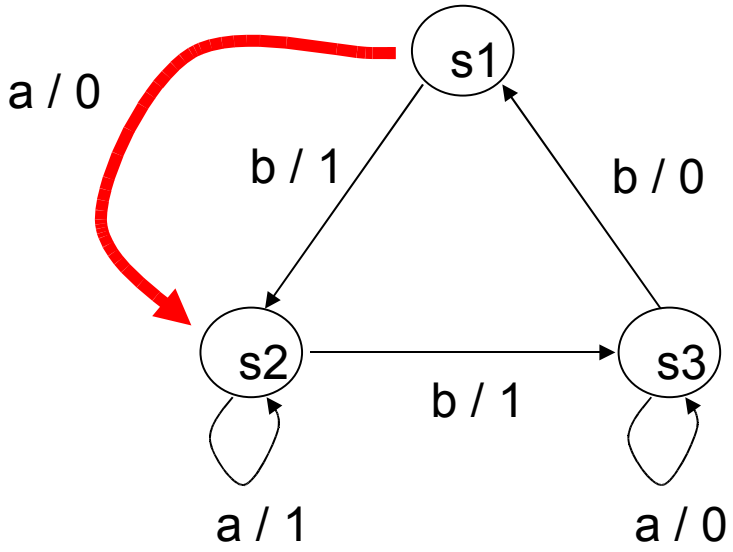
TT-method (4)

- Transition tours potrebbe anche non trovarli:

< specification >
Input sequence: *bababa*
Expected output sequence: *111000*

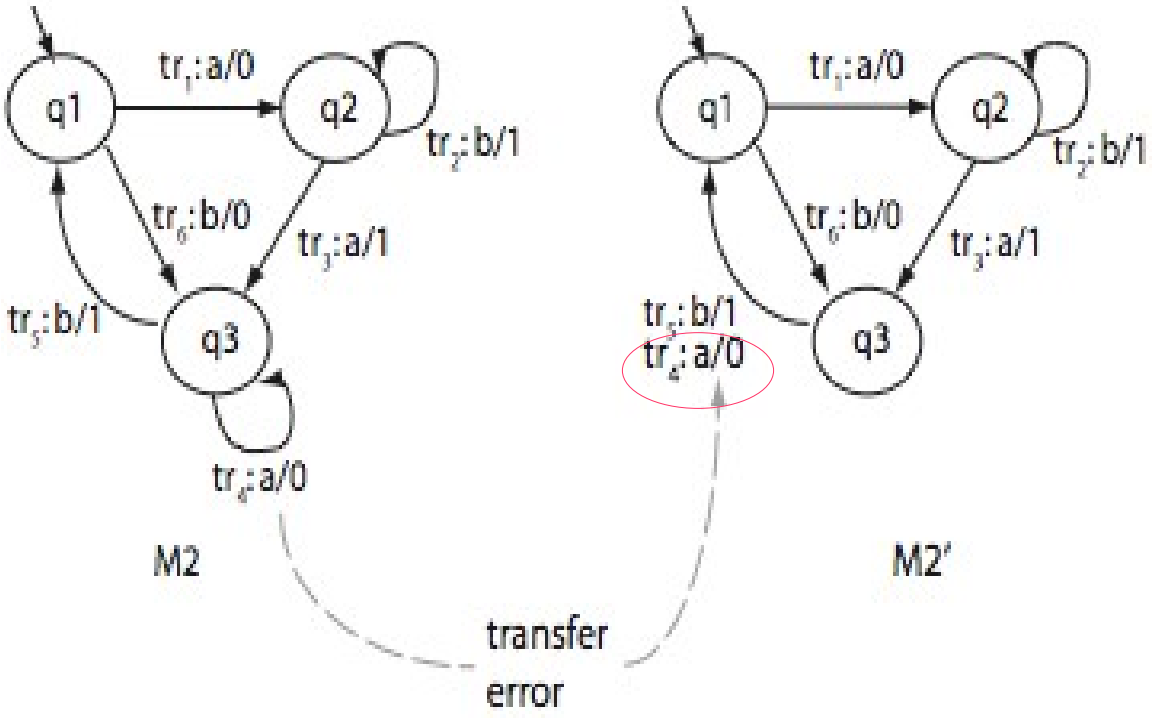


< implementation >
Input sequence: *bababa*
Observed output sequence: *111000*



TT method

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

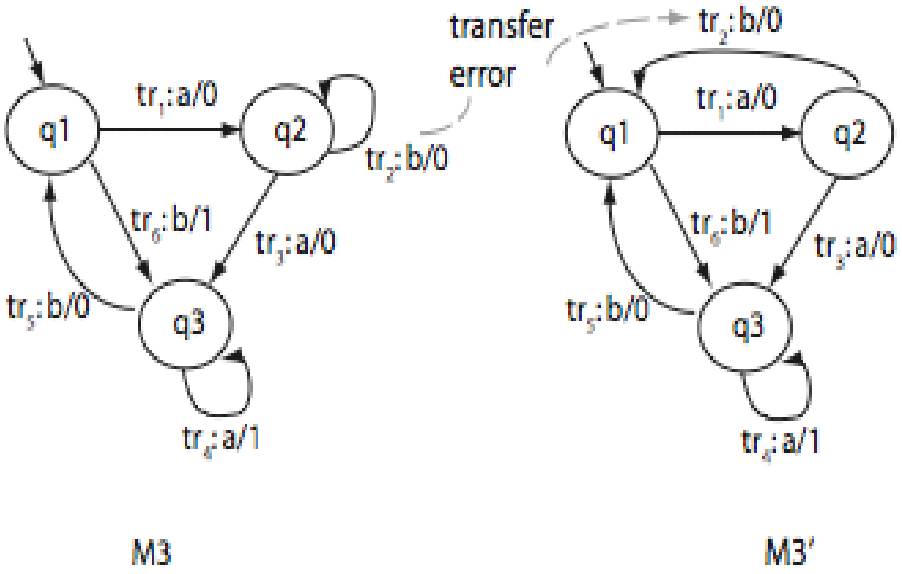


There are 12 branch pairs, such as (tr1, tr2), (tr1, tr3), tr6, tr5).

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

TT e state coverage

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



M3 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?

Detecting Output Faults and Transfer Faults

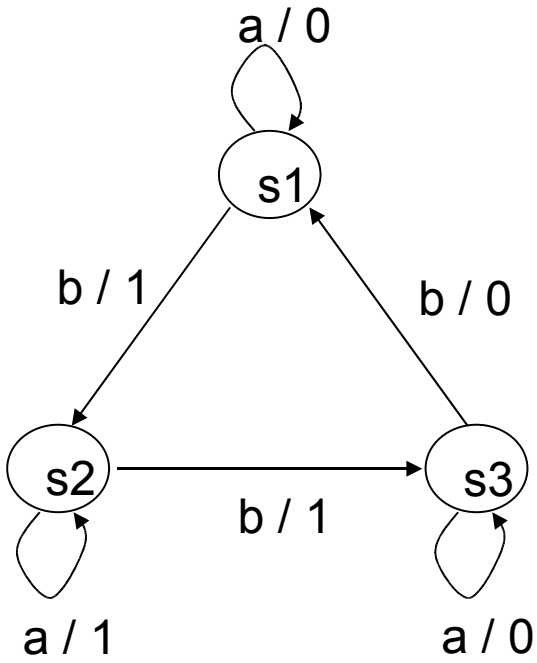
- DS-method, W-method, and UIO-method
- The main idea
  - Generate a test suite such that, for every transition *(s, i, o, s')*,
    - Step 1: Puts the implementation into state s (Setup)
    - Step 2: Applies input *i* and check whether the actual output is *o* (Output fault)
    - Step 3: Determines whether the target state of the implementation is *s'* (Transfer fault)

DS-method (1)

- An input sequence is a **distinguishing sequence** if
  - After applying the input sequence, we can determine the source state by observing the output sequence

DS-method (2)

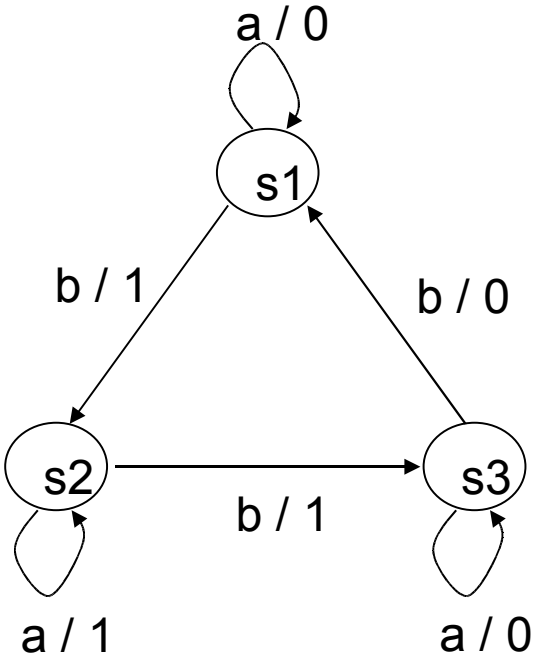
- Show that *a* is not a distinguishing sequence



Initial state	Input seq	Output seq	Final state
S1	a	0	S1
S2	a	1	S2
S3	a	0	S3

DS-method (3)

- Show that *ab* is a distinguishing sequence



Initial state	Input seq	Output seq	Final state
S1	ab	01	S2
S2	ab	11	S3
S3	ab	00	S1

Come calcolare una DS

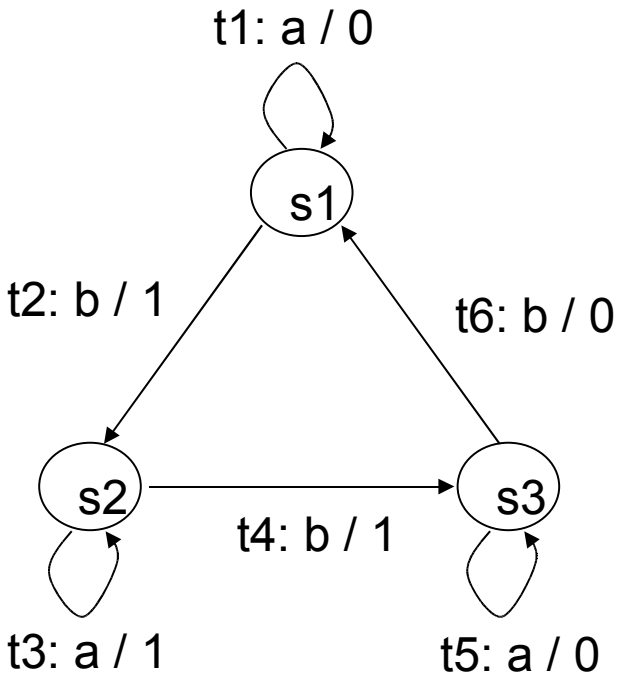
- Un po' troppo complicato ...
  - non vi verrà richiesto: però dovrete dire se una sequenza è una DS oppure no

DS-method (4)

- For every transition *(s, i, o, s')*
  - Step 1: Put the implementation into state s (Setup)
  - Step 2: Apply input *i* and check whether the actual output is *o* (Output fault)
  - Step 3: Determine whether the target state of the implementation is *s'* using a distinguishing sequence (Transfer fault)

DS-method (5)

- A test suite

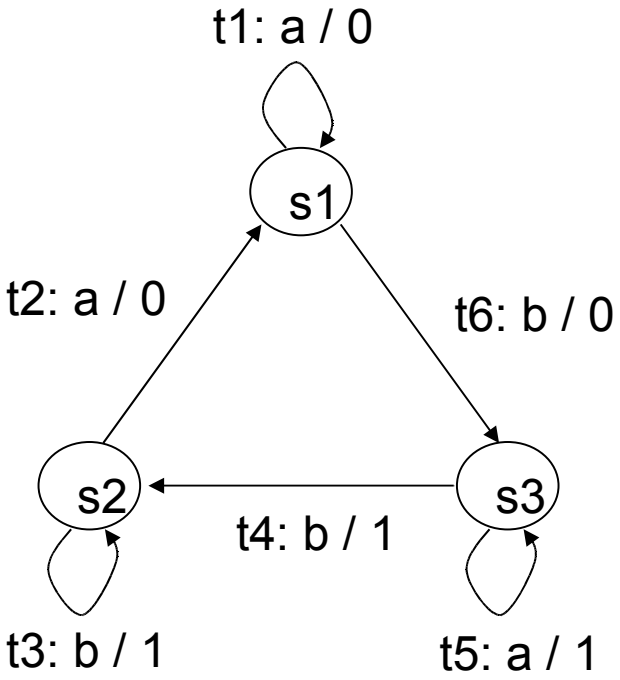


t1: reset/null a/0 a/0 b/1
t2: reset/null b/1 a/1 b/1
t3: reset/null b/1 a/1 a/1 b/1
t4: reset/null b/1 b/1 a/0 b/0
t5: reset/null b/1 b/1 a/0 a/0 b/0
t6: reset/null b/1 b/1 b/0 a/0 b/1

Setup **Output** **Transfer**

DS-method (6)

- Comparisons
  - Very few FSMs possess a distinguishing sequence
  - Even if an FSM has a distinguishing sequence, the sequence is too long
  - Example: there is no DS
    - A DS cannot start with *a* (s1, s2)
    - A DS cannot start with *b* (s2, s3)



Skip da qui in poi !!!

W-method (1)

- A set of input sequences is a **characterizing set** if
  - After applying all input sequences in the set, we can determine the source state by observing the output sequences

Step 2: Construction of W. What is W?

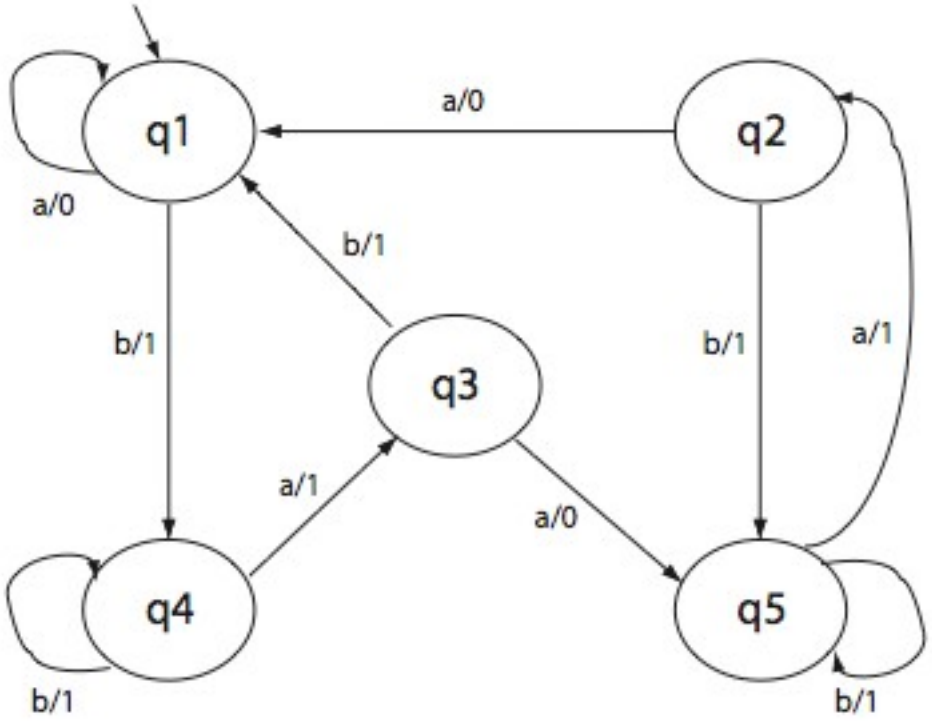
Let $M=(X, Y, Q, q1, \delta, O)$ be a minimal and complete FSM.

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- skip!!

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

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

What is a k-equivalence partition of Q?

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

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

States in Σ_{ki} are k-equivalent.

If state 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.

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

P_1 Table

Group number

Construct 2-equivalence partition: Construct P_2 table

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

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=ε. 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.

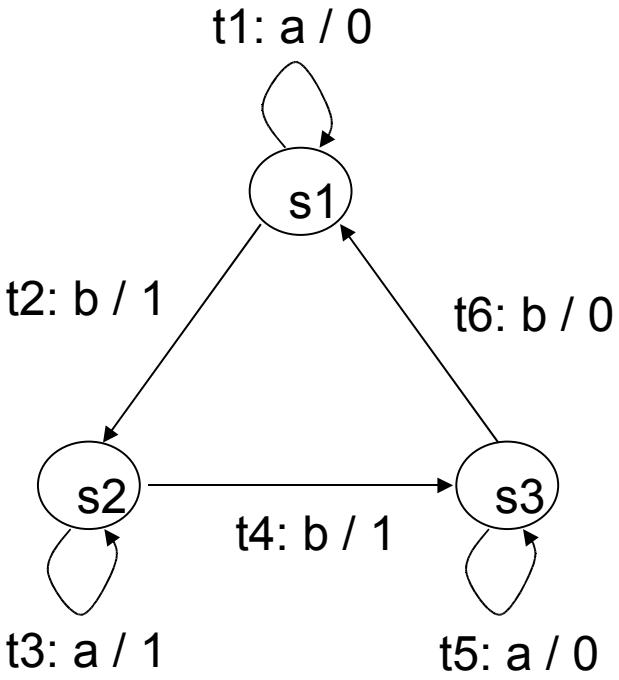
W={a, aa, aaa, baaa}

calcolo di W

- anche in questo caso non vi è richiesto, però è richiesto che sappiate dire se un dato insieme W è corretto o no: vedi prossime slides

W-method (2)

- Show that {*a,b*} is a characterizing set



Initial state	Input seq	Output seq	Final state
S1	a	0	S1
S2	a	1	S2
S3	a	0	S3

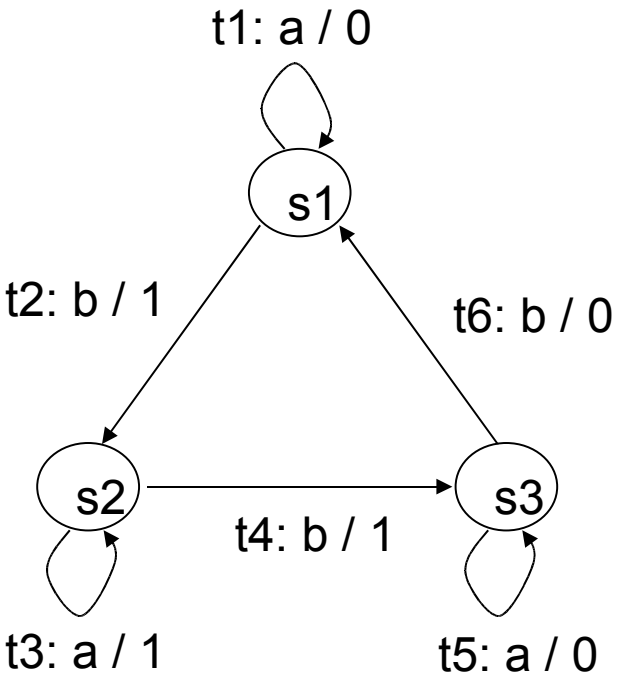
a distingue tra S2 e {S1,S3}

Initial state	Input seq	Output seq	Final state
S1	b	1	S3
S2	b	1	S2
S3	b	0	S2

b distingue tra S3 e {S1,S2}

W-method (3)

- A test suite



t1: reset/null a/0 a/0
t1: reset/null a/0 b/1
t2: reset/null b/1 a/1
t2: reset/null b/1 b/1
t3: reset/null b/1 a/1 a/1
t3: reset/null b/1 a/1 b/1
t4: reset/null b/1 b/1 a/0
t4: reset/null b/1 b/1 b/0
t5: reset/null b/1 b/1 a/0 a/0
t5: reset/null b/1 b/1 a/0 b/0
t6: reset/null b/1 b/1 b/0 a/0
t6: reset/null b/1 b/1 b/0 b/1

W-method (4)

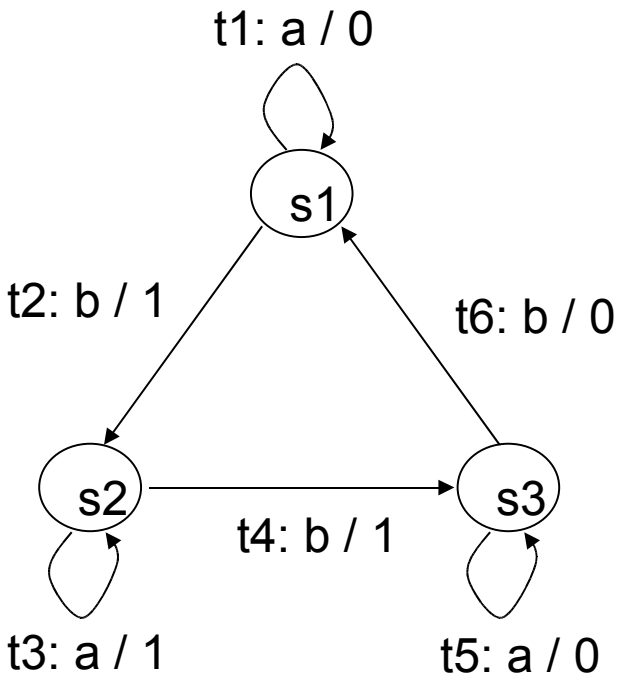
- Comparisons
  - Although every FSM has a characterizing set, the set may have too many elements
  - Both distinguishing sequences and characterizing sets impose too strong requirements
    - We are just interested in determining whether the target state is a specific state
      - State identification versus state verification

UIO-method (1)

- Let *s* be a state.
- An input sequence is a **UIO sequence** for s if
  - After applying the input sequence, we can determine the source state is *s* or not by observing the output sequence

UIO-method (2)

- Show that *a* is a UIO sequence for s2
- Show that *b* is a UIO sequence for s3

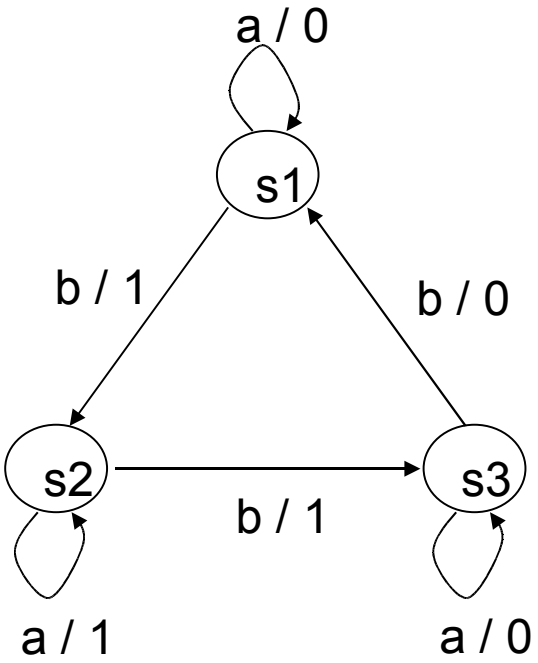


Initial state	Input seq	Output seq	Final state
S1	a	0	S1
S2	a	1	S2
S3	a	0	S3

Initial state	Input seq	Output seq	Final state
S1	b	1	S3
S2	b	1	S2
S3	b	0	S2

UIO-method (3)

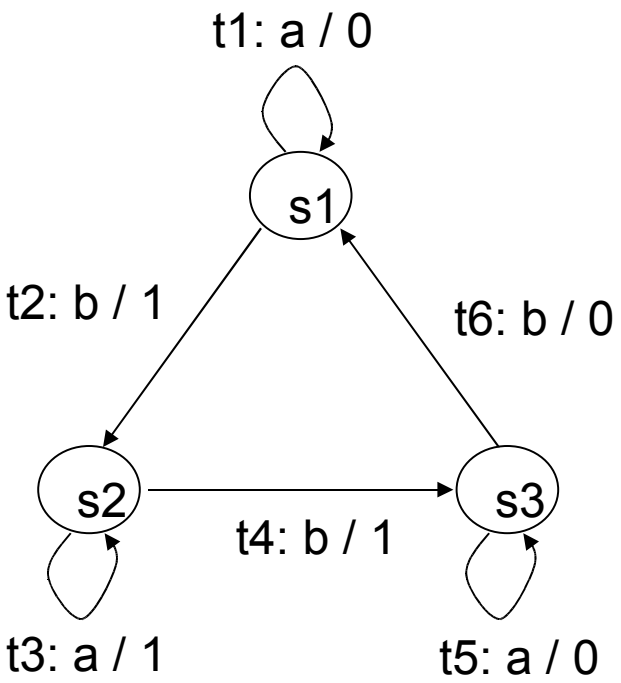
- Show that *ab* is a UIO sequence for s1



Initial state	Input seq	Output seq	Final state
S1	ab	01	S2
S2	ab	11	S3
S3	ab	00	S1

UIO-method (4)

- A test suite



t1: reset/null a/0 a/0 b/1
t2: reset/null b/1 a/1
t3: reset/null b/1 a/1 a/1
t4: reset/null b/1 b/1 b/0
t5: reset/null b/1 b/1 a/0 b/0
t6: reset/null b/1 b/1 b/0 a/0 b/1

UIO-method (5)

- Comparisons
  - Many FSMs have UIO sequences
  - UIO sequences are usually short
  - However, fault detection capability is not powerful