Software Testing and Quality Analysis

Module Supervisor: Mrs. M. Halassa

Chapter 6:
Generation of tests for finite state machine

Plan

- Introduction
- finite state machines.
- Using finite state machines.
- Conformance Testing
- Fault models based on finite state machines.
- Test based on finite state machines.
- Conclusion

Introduction

- The generation of tests from finite state machines (FSMs) facilitates the verification of whether implementations conform to their respective FSM models.
- Testing plays a crucial role in this process by selecting specific tests derived from the specification and executing them on the implementation. This approach aims to uncover any deviations or abnormal behaviors in the system under test

finite state machines

- Finite state machines(FSMs) offer a description formalism that is powerful with a lot of efficient algorithms.
- finite state machines is said to be finite because it has a finite number of distinct states.
- A FSMs forms a labeled oriented graph, whose states are the nodes and transitions are the labeled edges.

Using Finite State Machines

Finite state machines are used in different contexts:

- Data_Compression
- Compilation
- Biology (genomics)
- To describe the morpho-lexical level of natural languages (dictionaries, lexicons).

Using Finite state machines.

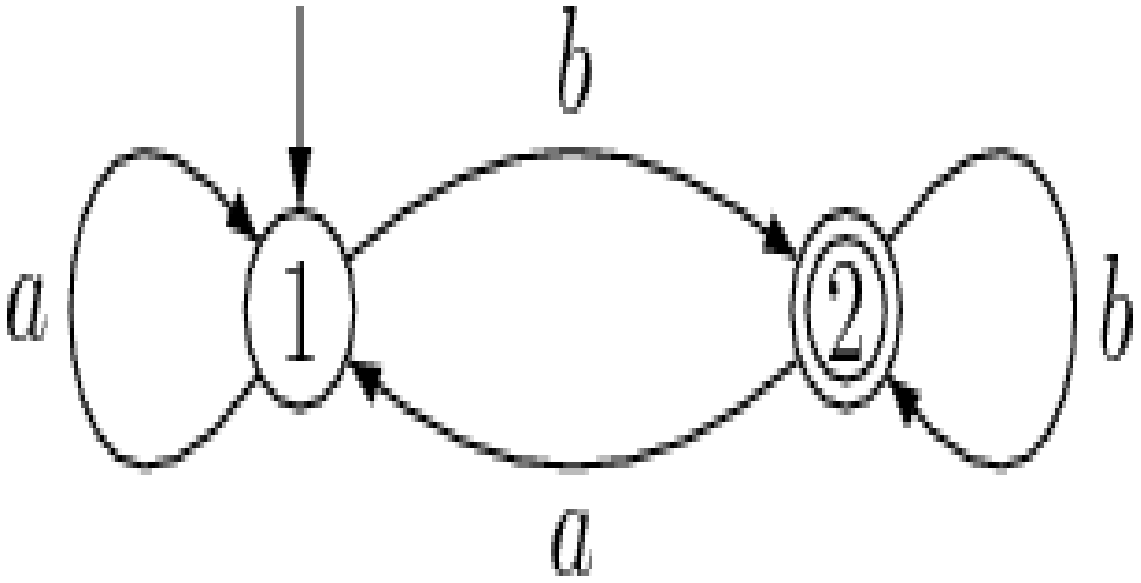
- To describe the dynamic properties of a system. For example in description languages like UML (state-transition diagrams).
- testing of protocols
- In tests based on finite state machines

Definition: FSMs

- A finite state machines is a quintuplet $A = (\Sigma, Q, \delta, i, F)$:
- Σ is a finite set of symbols called an alphabet.
- Q is a finite set whose elements are called states.
- δ is a relationship of $Q \times \Sigma \times Q$ called transitions of A.
- i is a state of Q called initial state.
- F is a subset of Q called the set of end states of A..
- A finite state machines is made of components that are all finite $(\Sigma, Q, \delta, i, F)$, hence the term finite.

Deterministic FSM

- A finite state machines is said to be deterministic, that is to say that from a state and a symbol , a single state can be reached.
- E.g. :



Finite state machines.

- Mealy and Moore FSMs(machines): a category of FSMs where actions can be performed at the outputs while the FSM is running.

- Mealy Machine (due to G. H. Mealy -1955 publication)

  Outputs corresponds transitions between states.

- Moore Machine (due to E. F. Moore -1956 publication)

  Outputs are determined only by the states

Here in after we consider Mealy machines; the latter model finite state systems in a more appropriate way.

- A Mealy machine has a finite number of states and produces outputs after receiving input data

Conformance Testing

- **Definition:** Conformance testing is the process of verifying whether an implementation behaves according to its specification. For systems modeled as finite state machines (FSMs), conformance testing ensures that the implementation under test (IUT) adheres to the specified transitions, outputs, and overall state behavior.

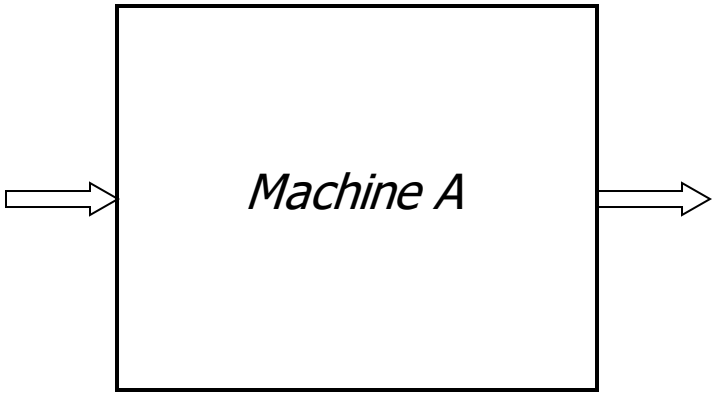
Conformance Testing

- The first class of methods is based on a represented specification based on Mealy Machines.
- The problem of conformance testing based on this model is therefore the following:

- Given a known machine S (the specification) and an unknown machine I (the implementation under test) whose inputs/outputs can only be observed, determine by a test (a finite sequence of inputs/outputs) whether I is equivalent to S.

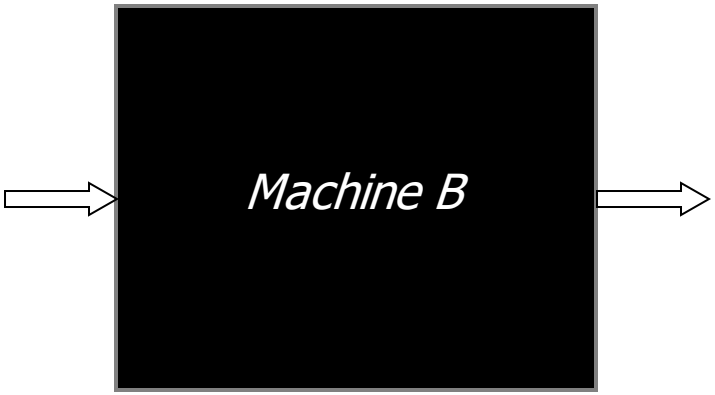
Gaol: Conformance Testing

Specification



Model

Implementation



Black box

Does it comply with A?

Fault Models

- The set of anomalies that can take place is very large, sometimes even the description of these anomalies is very complex.
- The major objective of the test is the detection of the presence of any anomaly, and several anomalies can generate the same error. Which leads us to reason in terms of fault model,
- Thus the test sequences will be selected so that they detect all the faults described in the fault model;

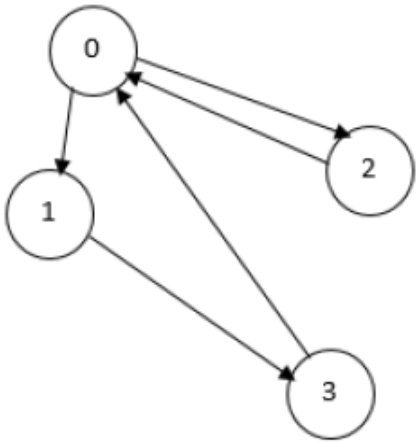
Fault models based on finite state machines

- For this problem to have a solution, it is generally assumed that the specification is minimal and strongly related and that the implementation is also minimal and strongly related,

Reminder

Strongly connected graph:

A graph is strongly connected, if there is a path, in "both directions", for any pair (u,v)

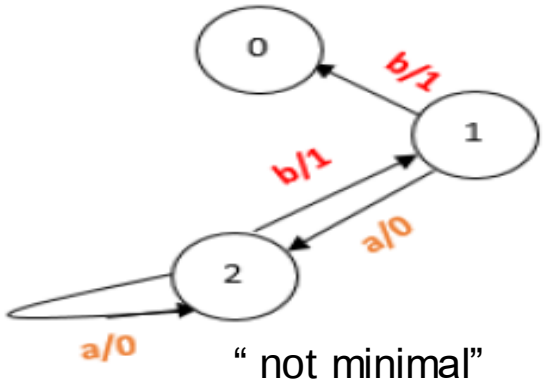


Equivalent states:

We will say that two states q1 and q2 are equivalent if all the inputs applied to them give the same outputs

Minimal Mealy machine:

We will say that a Mealy machine is reduced or minimal if it does not contain equivalent states.



" not minimal"

Fault models based on finite state machines

The types of faults to be considered for a deterministic specification based on finite state machine formalism can be described as follows

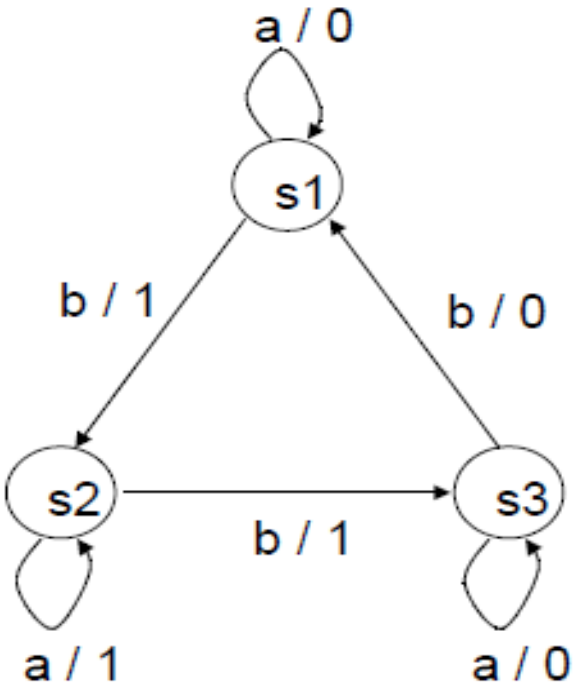
A. output fault: which occurs when the output of an implementation transition differs from this same transition in the specification.

- In the following figure, the transition (s2 s2) in the implantation has an output fault (0 instead of 1 provided by the specification.).

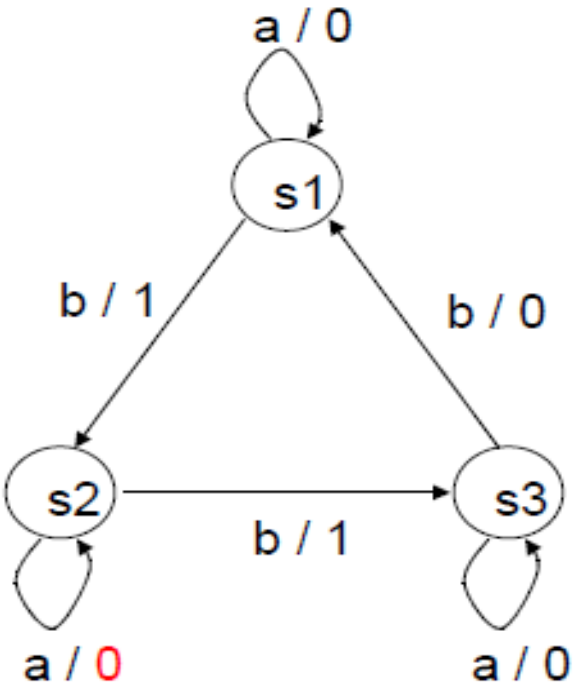
Fault model based on finite state machines

output fault: example

< spécification >



< implémentation >



Fault model based on finite state machines

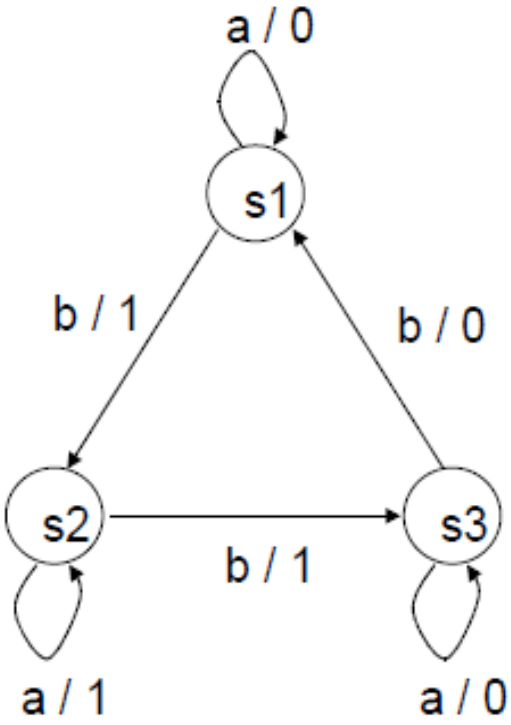
B. transfer fault: which occurs in the case where the terminal state of an implantation transition differs from that of the same transition in the specification.

- an implementation transition has a transfer fault (state S2 instead of S1 provided by the specification.).

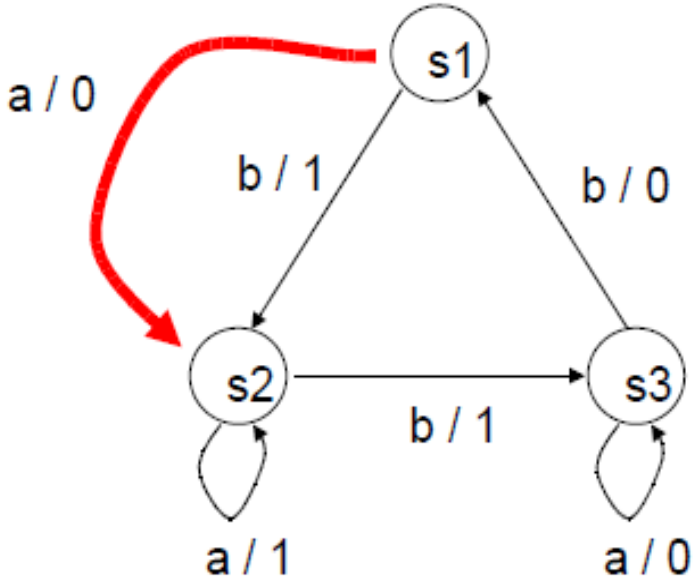
Fault model based on finite state machines

transfer fault: example

< spécification >



< implémentation >



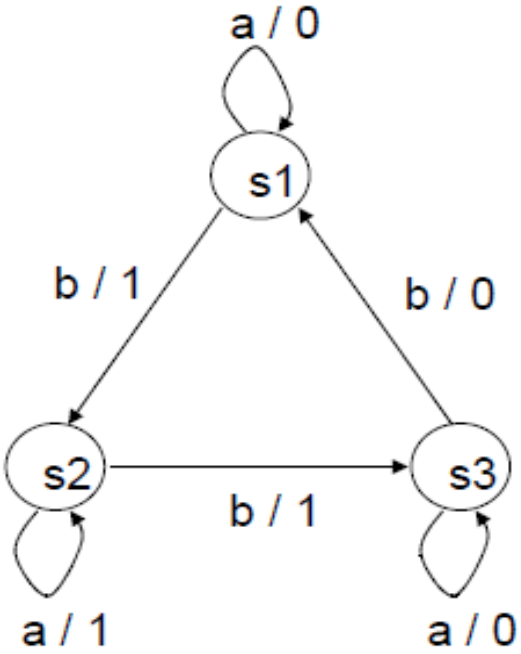
Fault model based on finite state machines

C. transfer fault with additional states: some errors can only be modeled by additional states with transfer faults. This is the case of the transition between S1 and S4 which has a transfer fault in the additional state S4.

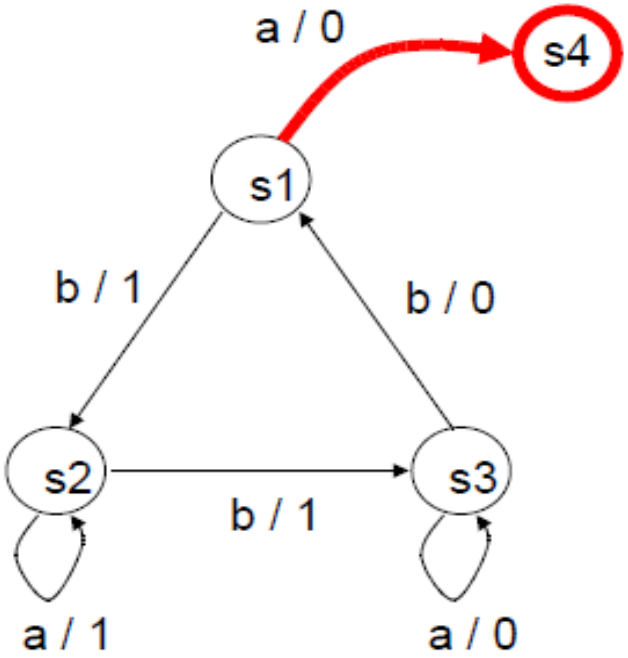
Fault model based on finite state machines

transfer fault with additional states: example

< spécification >



< implémentation >



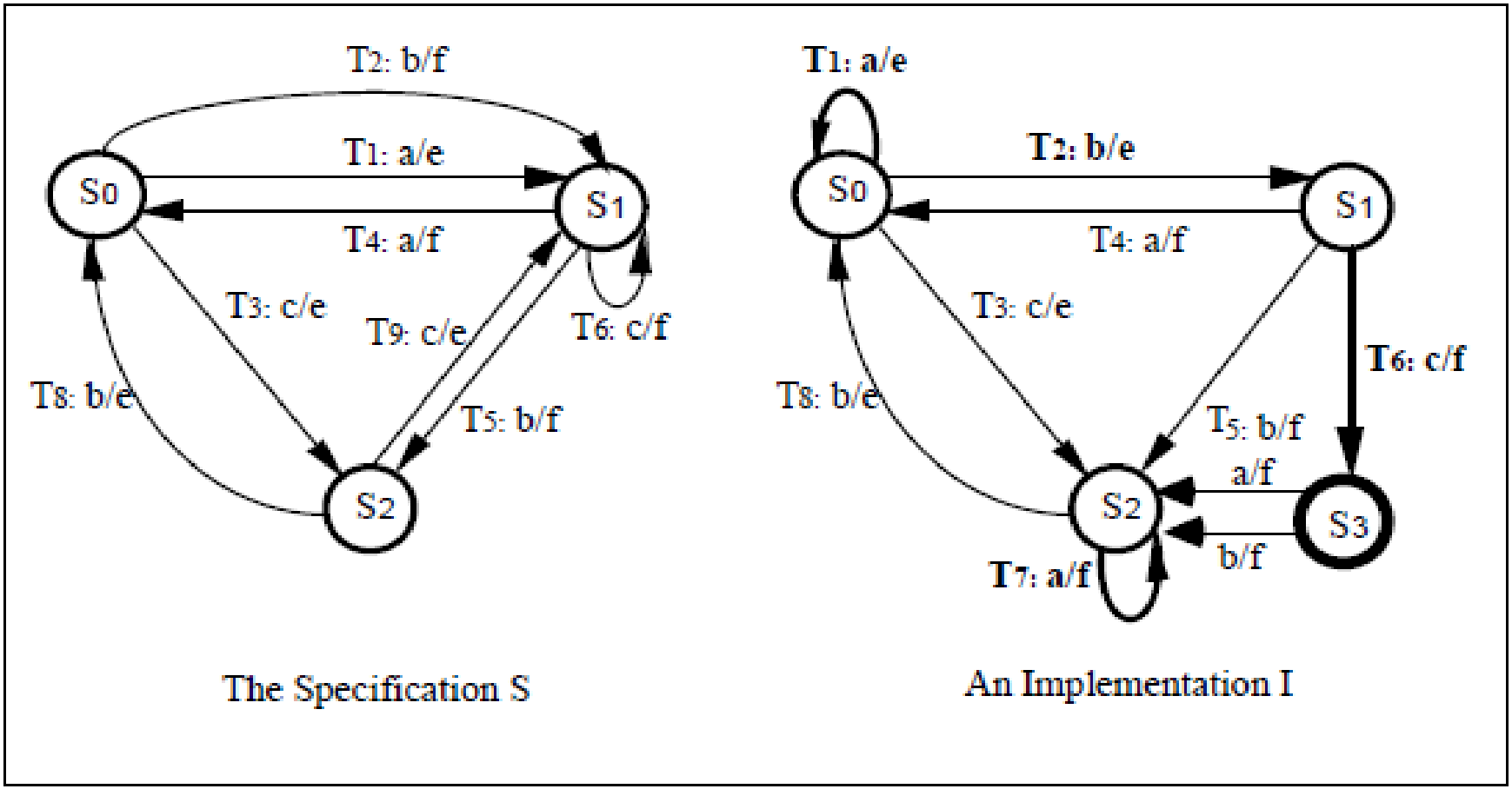
Fault model based on finite state machines

D. for lack of additional or missing transitions:

in the following figure, the T7 transition in the implantation is an additional transition. Transition T9 is a missing transition in implantation.

Fault model based on finite state machines

transfer fault with additional states: example



FSM based testing

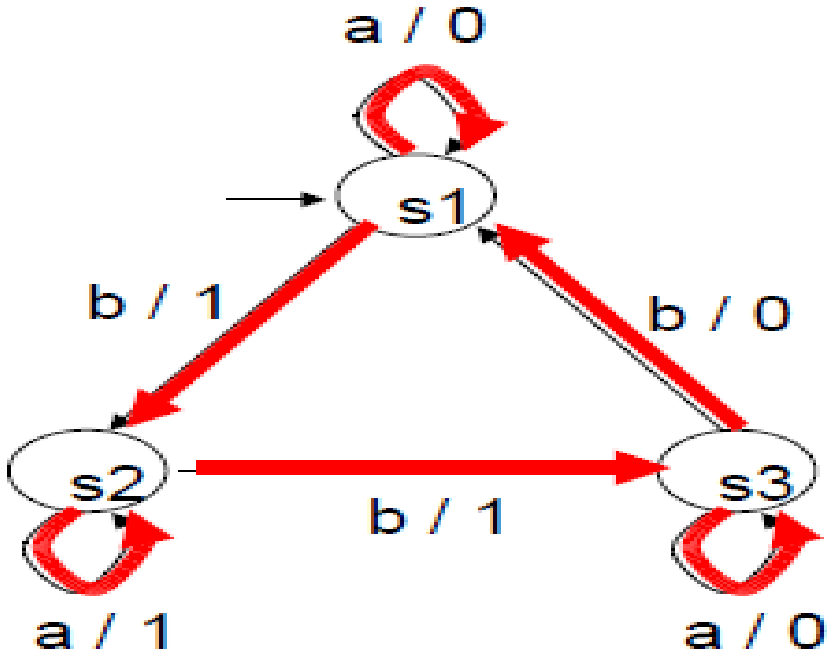
- The test synthesis is based on the search for correction and completeness of the test suite. That is, we are looking for a test suite that rejects only non-conform implementations (correction) and that can reject any non-conform implementation (completeness) of the fault model.
- Completeness implies that each of the transitions of I must be tested according to those of S

FSM based testing

- Output errors
  - TT method (transition round)
- Output errors + transition errors
  - DS (*Distinguishing Sequence*) method
  - Method W (discriminant sets)
  - *Unique Input Output (UIO)* method

TT method (1)

A transition tour of an FSM is a path from the initial state back to the initial state that traverses all transitions at least once



TT method (2)

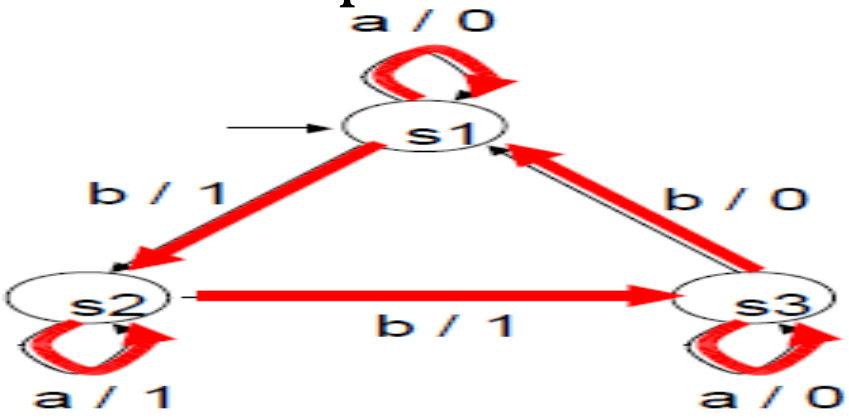
From a transition tour , we draw a test suite formed by a sequence of inputs and the sequence of outputs

Correspondent

– The *ababab* input suite

– The expected sequence of outputs

011100



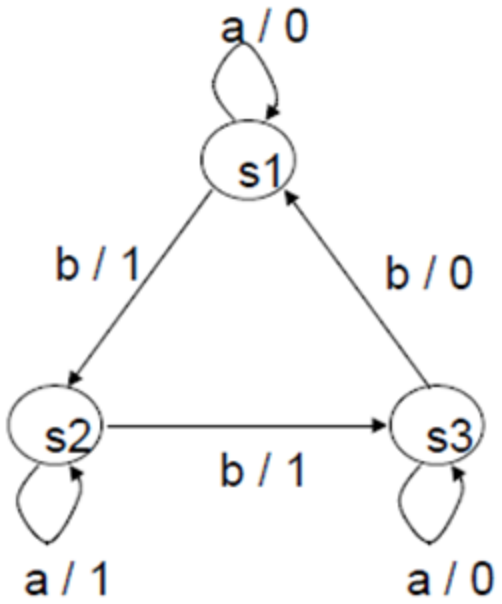
TT method (3)

- Transition tour can detect any output errors

< specification >

Inputs suit : *ababab*

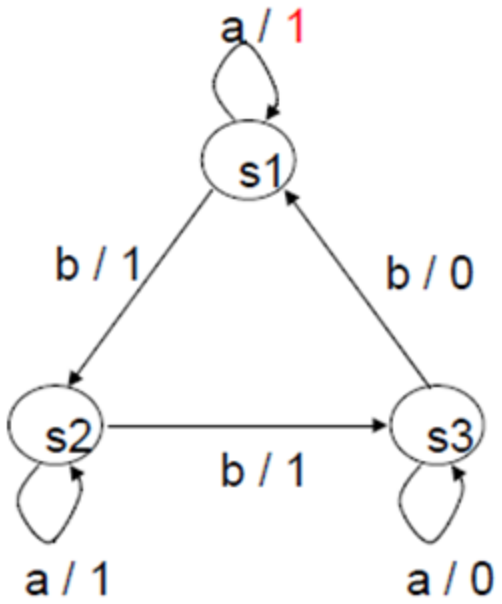
Correct outputs suite : *011100*



< implementation >

Inputs suit *ababab*

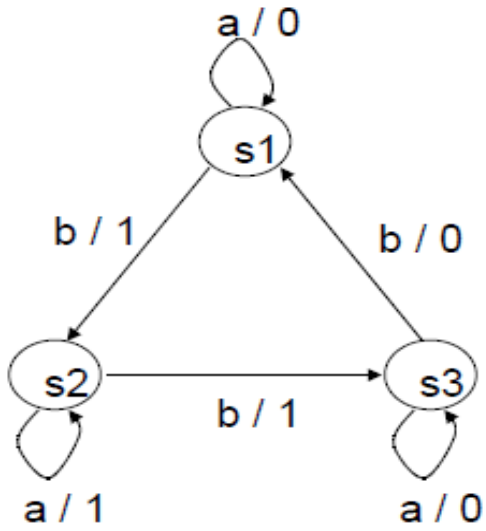
Observed Outputs suite: *111100*



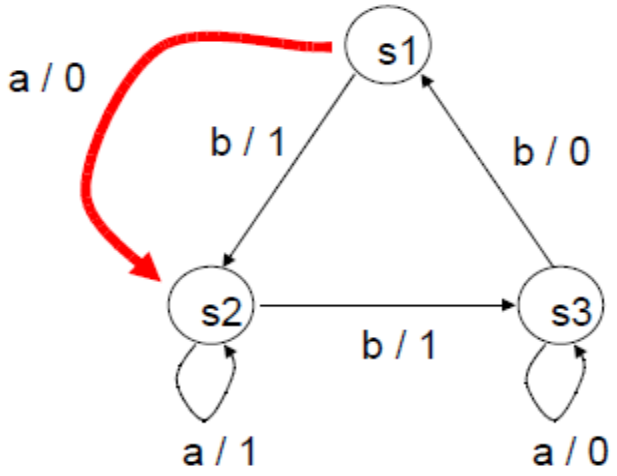
TT method (4)

- Transition tour cannot detect transition errors

< spécification >
Suite d'entrées: *ababab*
Suite de sorties attendue: *011100*



< implémentation >
Suite d'entrée: *ababab*
Suite de sorties observée: 010001



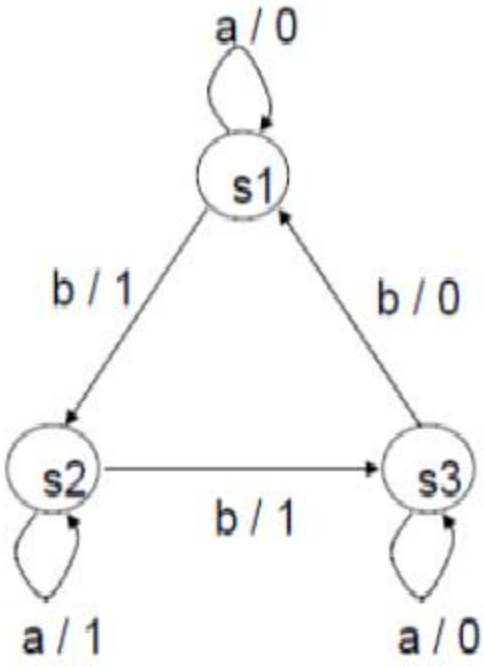
TT method (4)

- Transition tour cannot detect transition errors

< specification >

Inputs suit : *ababab*

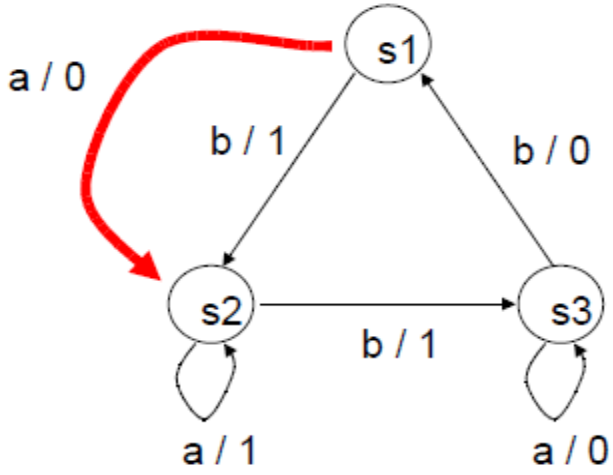
Correct outputs suite : *011100*



< implementation >

Inputs suit *ababab*

Observed Outputs suite: **010001**



Output and transitions -error detection

- **DS, W and UIO methods**
- The main idea: Generate a test suite. For any transition (s, i, o, s'),
- **Step 1:** Put the implantation in state s
- **Step 2:** Apply input i and check if the observed output is o (Output error)
- **Step 3:** Identify the arrival state by a sequence (Transition error)

Set of preambles

- Some of these methods use a set of transition sequences, which are integrated into the test sequences. This is called the set of preambles and is defined below:
- **Set of preambles** Q is the set of *input / output* symbol sequences that make it possible to arrive at any state of S starting from the initial state S0. For the initial state we have:

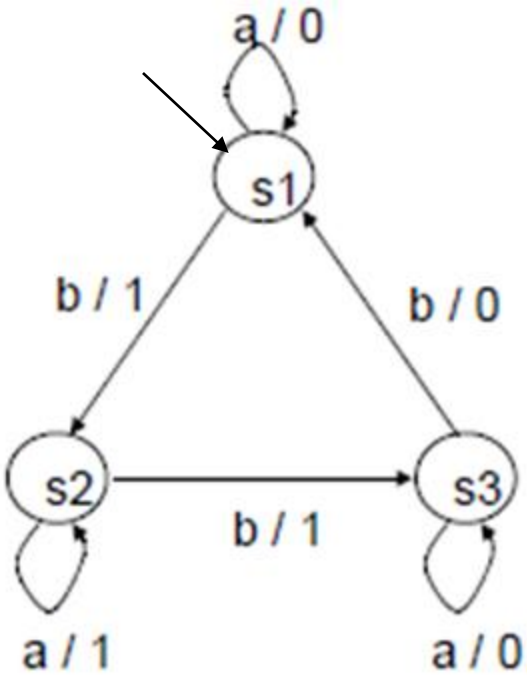
Q ={**reset/null }; reset/null** therefore still belongs to Q**.**

Distinction Sequence Method (DS) (1)

- **Principle:** A **DS** distinction sequence is a sequence of inputs causing a sequence of different outputs for each state of the FSM to which it is applied.

DS method (2)

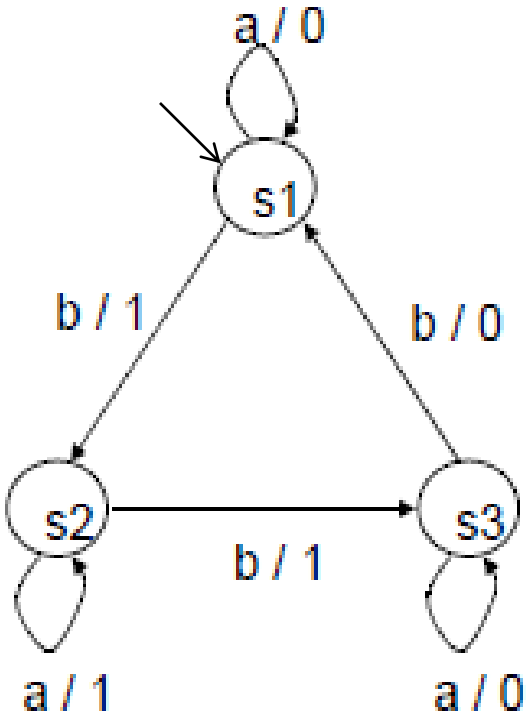
- a is not a distinguishing sequence (or discriminating sequence)



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

DS method (3)

- ab is a distinguishing sequence



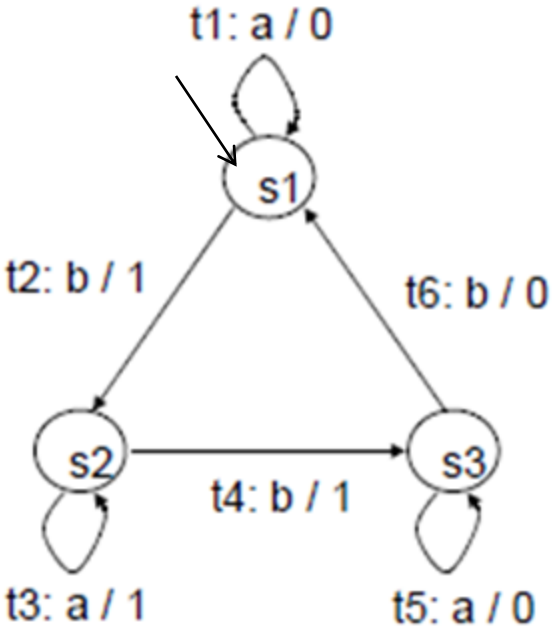
Initial state	Input seq	Output seq	Final state
S1	ab	01	S2
S2	ab	11	S3
S3	ab	00	S1

DS method (4)

- **The main idea:** Generate a test suite. For any transition (s, i, o, s'),
- Step 1: Put the implantation in state s (go to s)
- Step 2: Apply input i and check if the observed output is o (Output error)
- Step 3: Identify the arrival state by a sequence. (Transition Error)

DS method (5)

- Suite of test cases



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

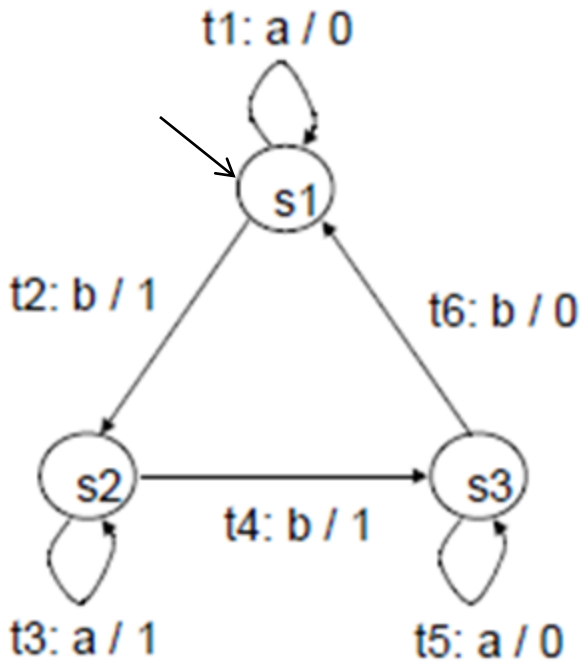
preambles output transition

W Method (1)

- **Principle:** A discriminant set W is a set of input sequences for identifying each state in the specification.
- A sequence of inputs W distinguishes two states if, applied respectively to each of the states, it produces different outputs.

W Method (2)

- {a,b} is a discriminating set

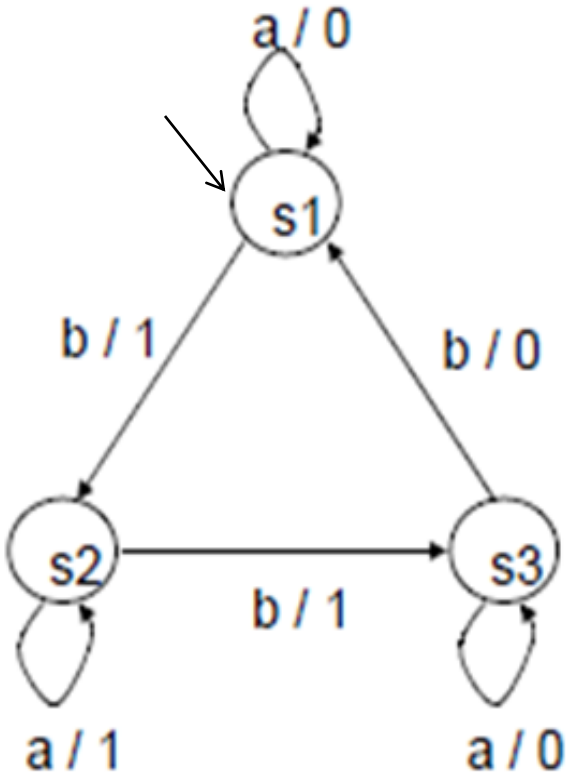


Initial state	Input seq	Output seq	Final state
S1	a	0	S1
S2	a	1	S2
S3	a	0	S3

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

W Method (3)

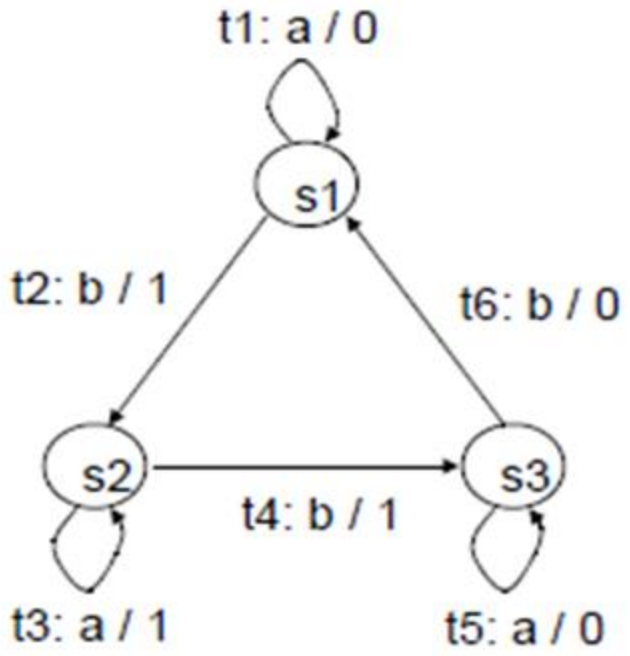
- {a,b} is a discriminating set



Initial state	Input seq	Output seq	Final state
S1	{a,b}	01	S2
S2	{a,b}	11	S3
S3	{a,b}	00	S1

W Method (4)

Suite of test



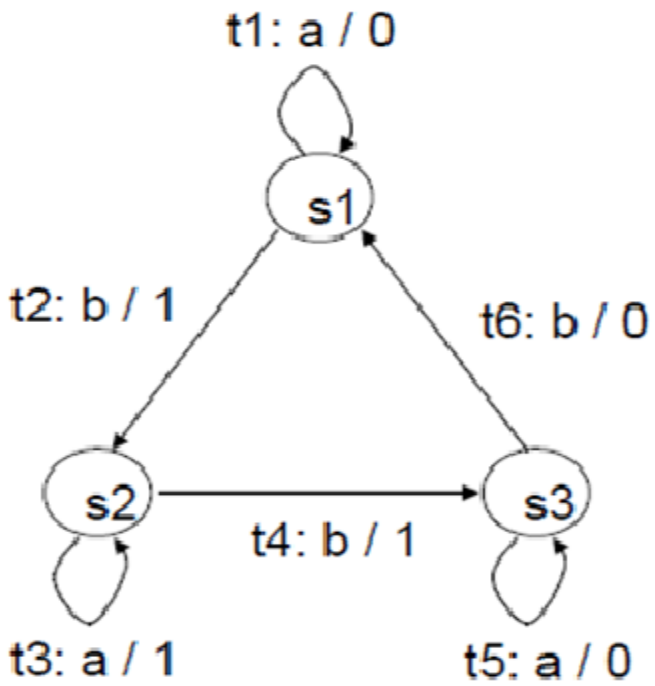
t1: reset/null a/0 a/0
t1: reset/null a/0 b/1
t2: reset/null b/1 a/1
t2: reset/null b/1 b/1
t3: reset/null b/1 a/1 a/1
t3: reset/null b/1 a/1 b/1
t4: reset/null b/1 b/1 a/0
t4: reset/null b/1 b/1 b/0
t5: reset/null b/1 b/1 a/0 a/0
t5: reset/null b/1 b/1 a/0 b/0
t6: reset/null b/1 b/1 b/0 a/0
t6: reset/null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

UIO method (1)

- **Principle:** A unique input output (UIO) is an input and output sequence describing the behavior of the specification in a unique way. Therefore, each state is identifiable through this sequence of inputs and outputs.
- This method detects output and transfer faults. On the other hand, it is applicable only to deterministic, minimal, strongly related FSMs with a UIO sequence for each state.

UIO method (2)

- a is a UIO suite for s2 , b is a UIO suite for s3

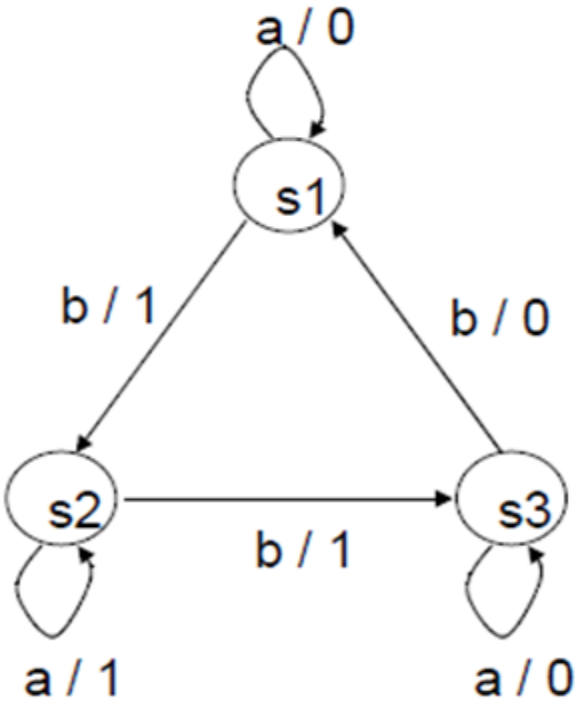


Initial state	Input seq	output seq	Final state
S1	a	0	S1
S2	a	1	S2
S3	a	0	S3

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

UIO method (3)

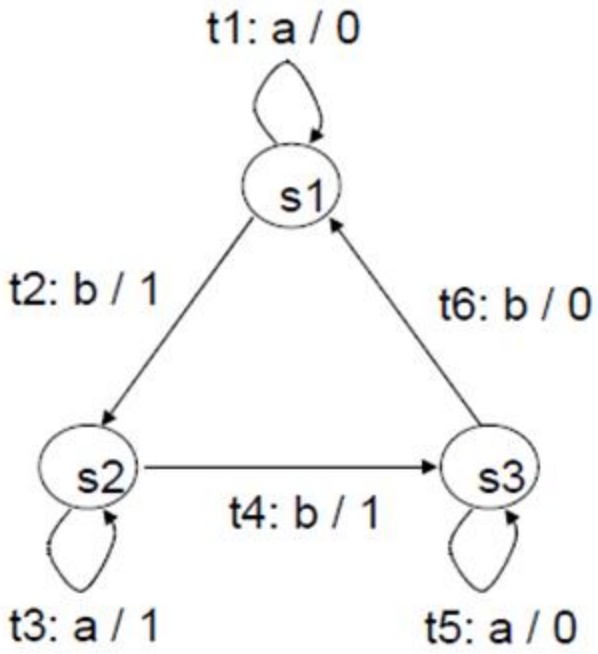
- ab is a UIO suite for s1



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

UIO method (4)

Suite of test



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