Testing Basato su specifiche

Angelo Gargantini

Testing e verifica del software

2024

Testing basato su specifiche

Nel testing basato su specifiche, la specifica dei requisiti svolge due ruoli fondamentali:

1. i test criteria sono definiti a partire dalle specifiche
   - quando fermarsi con il testing di un sw è deciso guardando la sua specifica
   - i dati di test sono derivati dalle specifiche (o dalla documentazione delle funzionalità del sw)
     - functional testing
- la specifica è usata come oracolo: prendo gli stessi input che do al programma e li "applico alla specifica" e vedo se gli output del programma sono gli stessi

Esempio Oracolo

Una funzione foo **che per ogni x mi restituisce il doppio**

- provo con foo(2), foo(1), foo(10), … come posso essere sicuro che l'output ottenuto è quello atteso?

Nel caso di specifiche formali:

$\forall x$ foo(x) = 2 x

- nel caso di specifiche eseguibili posso prendere i casi di test ed "eseguirli" sia sulle specifiche che sui programmi e vedere i risultati:
  - **conformance testing**: testo la conformità del programma alla sua specifica

Notazione della specifica

- **Si possono usare diverse notazioni per specificare il sistema**
- **Esempi**
  - UML: macchini di stati/diagrammi di interazione
  - ASM: ...
  - Simulink: ...
- **Noi useremo le FSM (di Mealy)**

- **Macchine a stati finiti (FSM)**

Macchine a stati finiti con output

Una FSM (S, I, δ) con output è:

- una macchina di Mealy se è una FSM che produce un output per **ciascuna transizione**
- una macchina di Moore se è una FSM che produce un output per **ciascun stato**

Macchina di Mealy

Una macchina di Mealy è una tupla

$(S, I, O, \delta, \lambda)$

- S : insieme finito di stati
- I : insieme finito di eventi di input
- O : insieme finito di eventi di output
- δ : S x I ->S : funzione di transizione
- λ : S x I ->O : funzione di output

Spesso è anche fissato lo stato iniziale $s_0 \in S$

FSM di Mealy: rappresentazione grafica

Nei diagrammi che rappresentano una FSM di Mealy, ogni arco è etichettato da i/o

- **i** denota un simbolo di input ed è anche noto come evento di input
- **o** denota un simbolo di output ed è anche noto come evento di output

FSM di Mealy: rappresentazione grafica

Esempio:

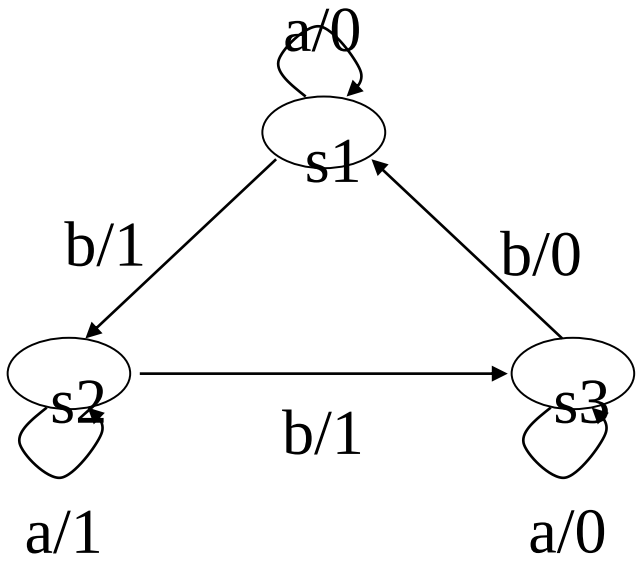
S = {s1, s2, s3}

I = {a, b}

O = {0, 1}

δ = { (s1,a,s1), (s1,b,s2),
(s2,a,s2), (s2,b,s3),
(s3,a,s3), (s3,b,s1)}

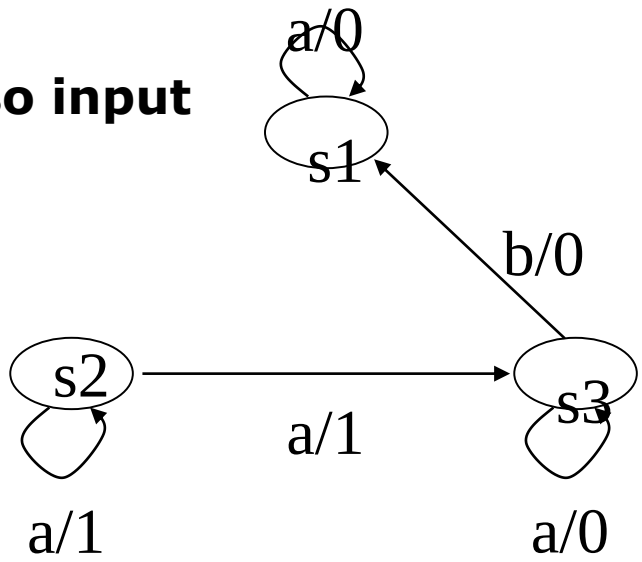
λ = { (s1,a,0), (s1,b,1),
(s2,a,1), (s2,b,1),
(s3,a,0), (s3,b,0)}



Errori nella definizione dalla mia macchina

- **1. non definisco cosa fare quando ho un certo input**
  - Arriva input b in S1

- **2. ho diverse transizioni con lo stesso input**

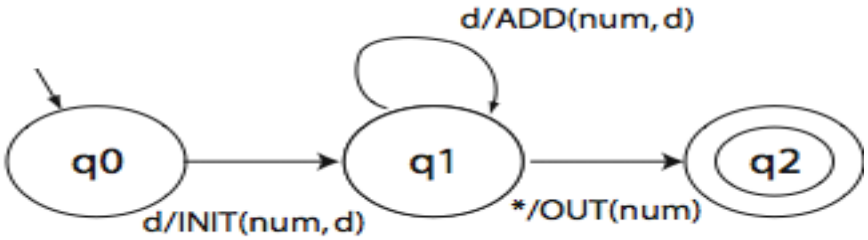


FSM: azioni sulle transizioni

Un evento di output può anche essere una azione della macchina

Esempio: macchina per convertire una sequenza di cifre decimali (d) in un intero (num)

- Azioni:
  - INIT: inizializza "num"
  - ADD: aggiunge la cifra "d" al valore corrente di "num"
  - OUT: output del numero



FSM di Mealy

È utile vedere una FSM di Mealy come la tupla (S, I, O, T)

- S : insieme finito di stati
- I : insieme finito di eventi di input
- O : insieme finito di eventi di output
- T : insieme finito di transizioni
  - Una transizione è una tupla (s, i, o, s')
    - s : stato sorgente
    - i : evento di input
    - o : evento di output
    - s' : stato target

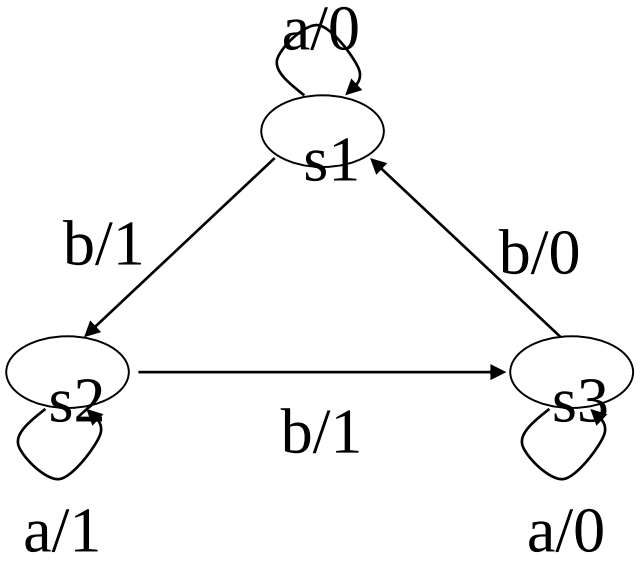
FSM di Mealy: esempio

S = {s1, s2, s3}

I = {a, b}

O = {0, 1}

T = {(s1,a,0,s1),
(s1,b,1,s2),
(s2,a,1,s2),
(s2,b,1,s3),
(s3,a,0,s3),
(s3,b,0,s1)}



Nota importante

Nel seguito per FSM intenderemo macchine di Mealy

Per esse utilizzeremo la definizione come la tupla (S, I, O, T)

Limiti delle FSM

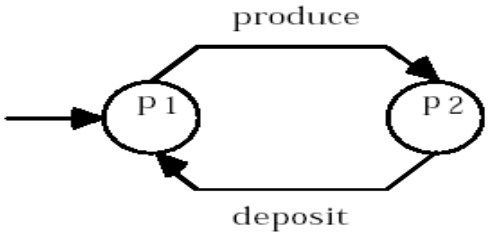
E’ possibile rappresentare solo un numero finito di stati

Esplosione del numero di stati:

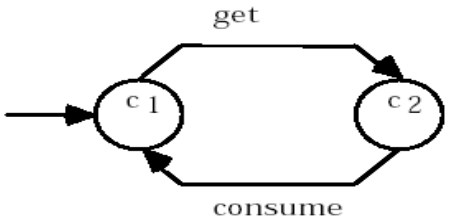
- dato un numero di FSM con k1, k2, … kn stati ciascuna, la loro composizione è una FSM con k1 * k2 *… * kn stati
- tale crescita è esponenziale nel numero di FSM
- ci piacerebbe una crescita lineare, ossia k1 + k2 +… + kn stati

Esplosione degli stati: esempio

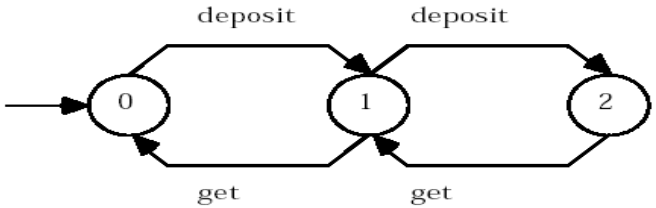
Produttore



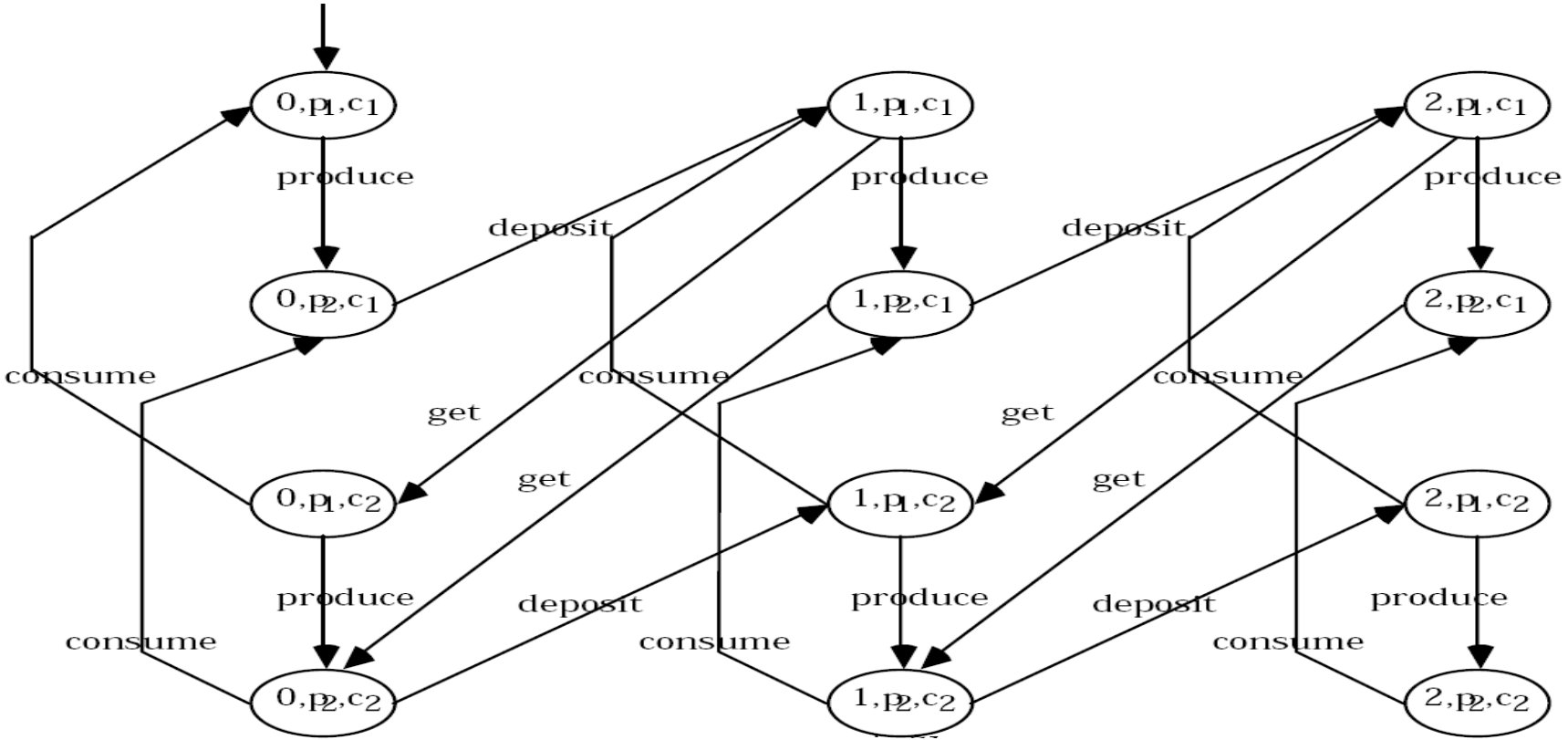
Consumatore



Magazzino



Combinando le FSM



Limiti delle FSM

Per superare i limiti di composizionalità delle FSM, sono state definite opportune estensioni (tra cui le Statecharts di UML) che sono dotate dei concetti di sottomacchina e permettono

- Composizione sequenziale
- Composizione parallela
- Modularità

Esercizio

Modellare con una FSM il comportamento di una sbarra che consente l'accesso/l'uscita di un parcheggio.

•Per accedere al parcheggio, il semaforo deve essere verde. Mentre la sbarra è aperta, il semaforo è rosso.

Quando l'auto è entrata, la sbarra si chiude ed il semaforo ritorna verde.

Per uscire dal parcheggio, l'autista deve inserire il biglietto pre-pagato nel dispositivo che comanda la sbarra.

Quando l'auto è uscita la sbarra si chiude.

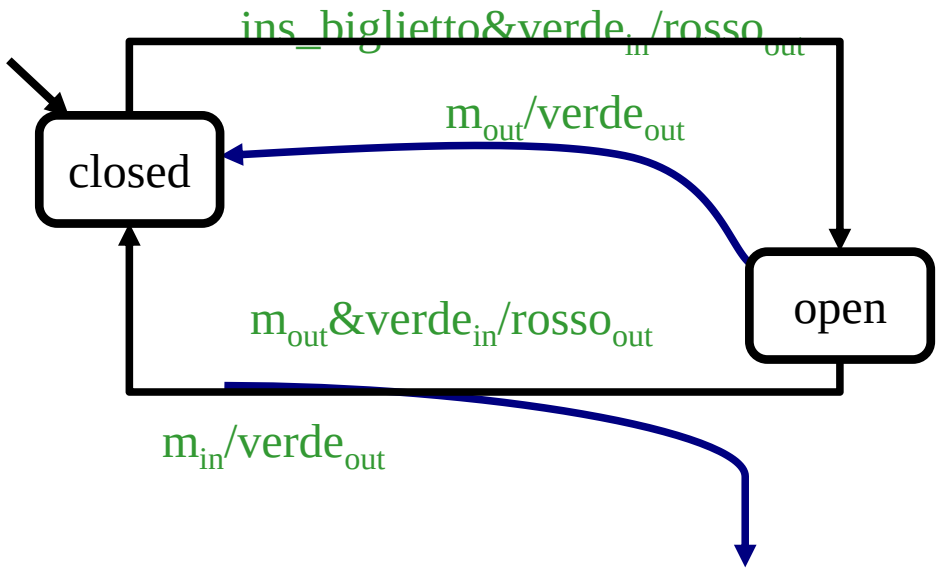
Soluzione esercizio

$verde_{in}$, $verde_{out}$, $rosso_{in}$, $rosso_{out}$: colori semaforo

m_{in} : segnale di presenza dell'auto nel parcheggio

m_{out}: segnale di presenza dell'auto fuori il parcheggio

ins_biglietto: segnale di inserimento biglietto



In sintesi

- Abbiamo visto:
  - le Macchine a Stati finiti sono estese con il concetto di output
  - la differenza tra macchine di Mealy e di Moore
  - i limiti delle macchine a stati finiti dovuti alla non-composizionalità dei modelli
- Ricordate che:
  - l'output può anche essere rappresentato da un'azione della macchina
- Infine:
  - da ora in avanti per FSM intenderemo una macchina di Mealy

Conformance testing con FSM

Supponiamo che

- la specifica sia data come macchina a stati finiti **S** di cui si conosce tutto (stati, transizioni e funzione di output)
- il sistema (programma, protocollo, …) implementato da testare sia una macchina a stati finiti **I**, tipo black box, di cui si può osservare solo l'output (applicando certi input)

Obiettivo:

- determinare se **I** è una implementazione corretta di **S** applicando una sequenza di test e osservando l'output
  - corretta se I si comporta come S per ogni sequenza

Testing con le FSM (1)

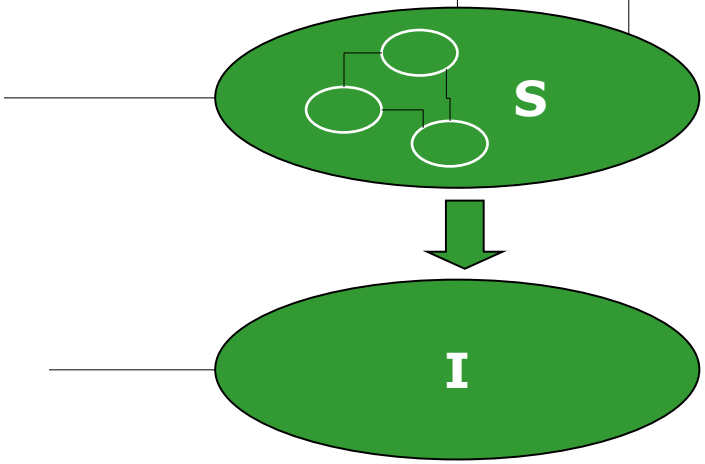
casi di test

2. trova i casi di test

1. analizza la FSM

3. seleziona e applica gli input scelti alla specifica

4. applica gli stessi input al programma

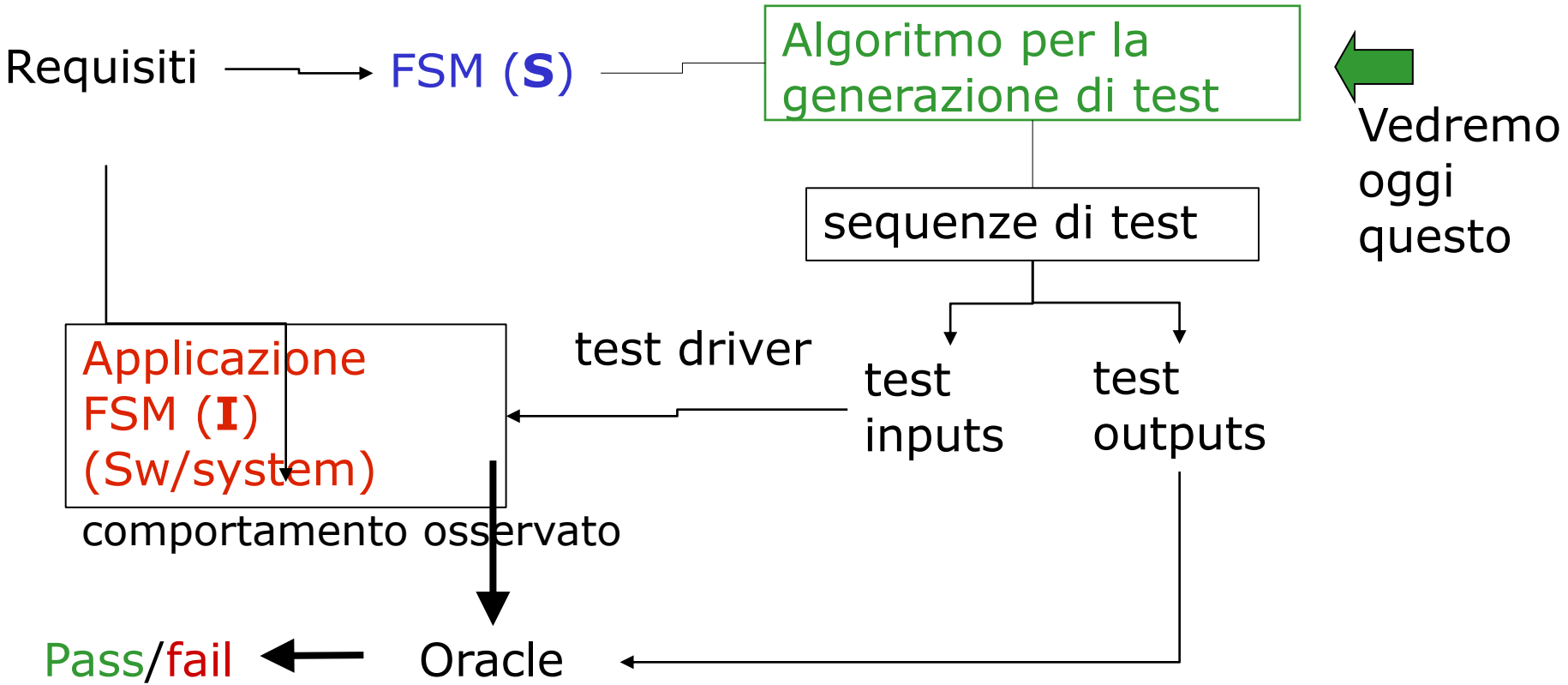


A

B

5. confronta gli output **A** e **B**

Testing con le FSM (2)



Perchè testing con le FSM è utile

Le FSMs sono spesso usate direttamente per specificare protocolli, sistemi di controllo embedded e circuiti digitali

Molte altre notazione sono simili alle FSM: StateCharts-Harel, SDL for communication protocols, UML state diagrams, Mathworks StateFlow

Spesso è possibile **astrarre una parte del sistema** o un suo comportamento o una sua interazione con l'ambiente e modellarlo come FSM e applicare le tecniche di testing su quella parte

Test ideale

- Lo scopo dei metodi di test per presenteremo nelle prossime lezioni è quello di provare l'assenza di difetti (test ideale)

- Solo sotto ipotesi precise e abbastanza forti, altrimenti non c'è garanzia

- I metodi che vedremo però hanno dimostrato di essere efficaci nella pratica anche quando le ipotesi non valgono

Assunzioni (1)

S e I sono deterministiche e inizializzate

S e I sono completamente specificate

- rispondono ad ogni input in ogni stato

La specifica S è fortemente connessa

- deve essere possibile raggiungere tutti gli stati

S è minimizzata (ridotta)

- l'equivalenza può essere stabilita solo tra macchine minimizzate, poiché stati equivalenti sono indistinguibili

I non cambia durante l'esperimento

I ha lo stesso alfabeto di S

Assunzioni (2)

I non ha più stati di S

- assumiamo che i difetti non aumentano il numero di stati
  - output sbagliato su una transizione
  - stato destinazione di una transizione sbagliato

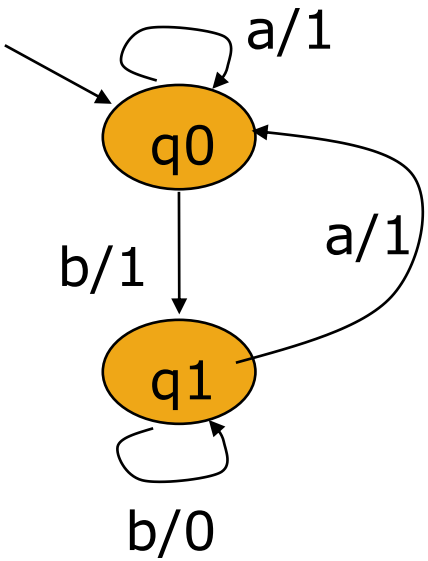
Fault model – modello dei difetti

un fault model è un modello ipotetico su quali tipi di difetti possono accadere in una implementazione

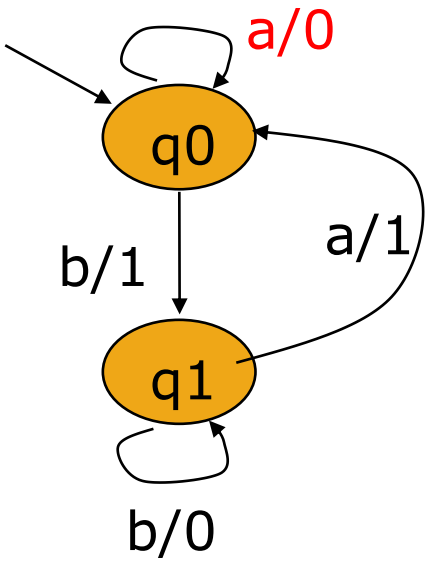
Senza modello dei difetti:

- c'è un numero infinito di implementazioni errate
- il testing non può garantire nulla

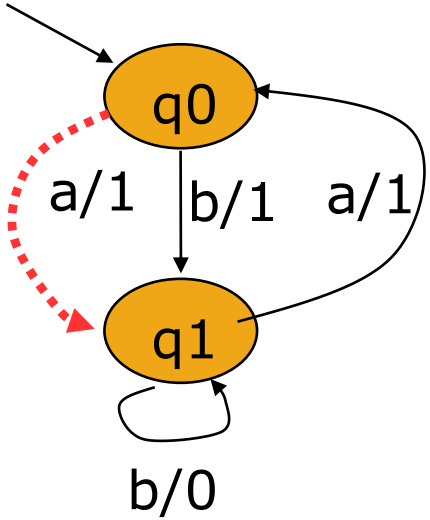
Difetti considerati



macchina corretta

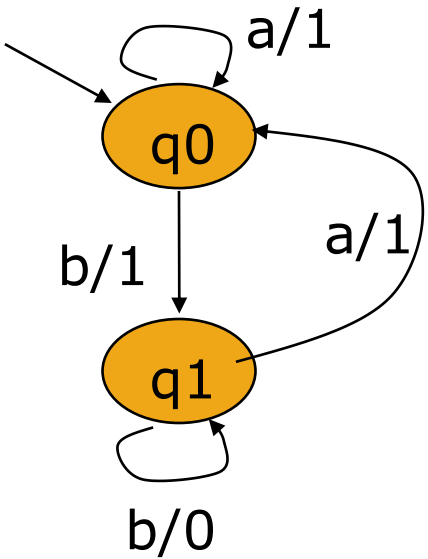


difetto di output

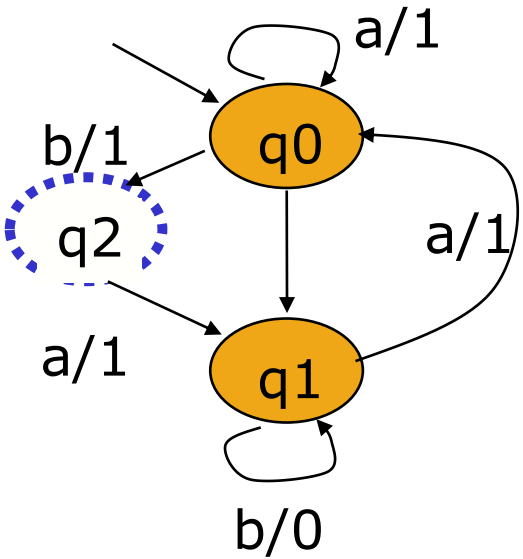


difetto di trasferimento

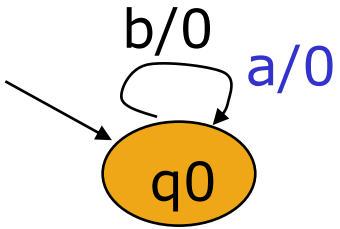
Difetti non considerati



macchina corretta



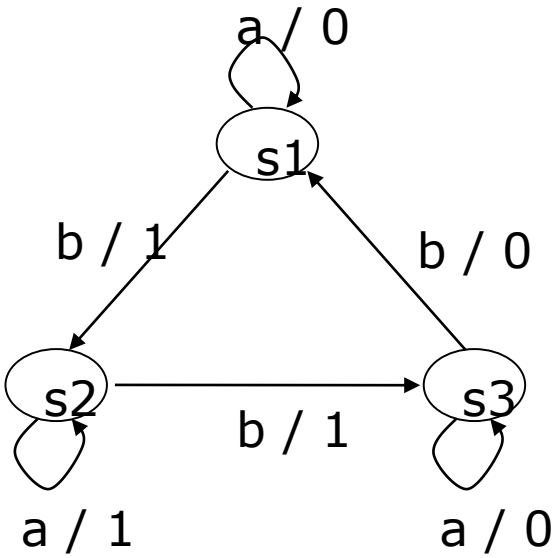
extra state



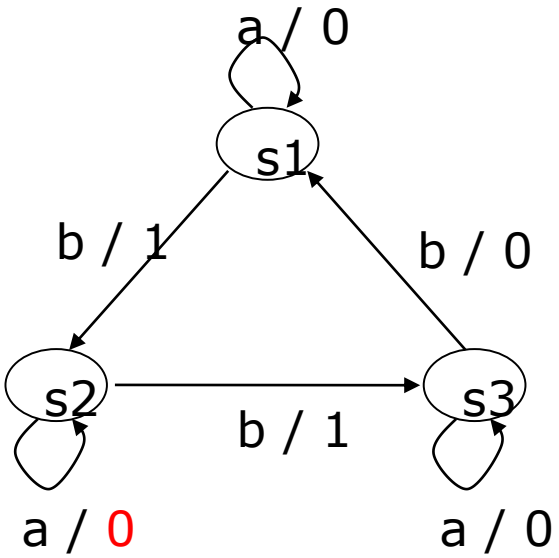
missing state
stato mancante

Difetti di output

< A: specification >



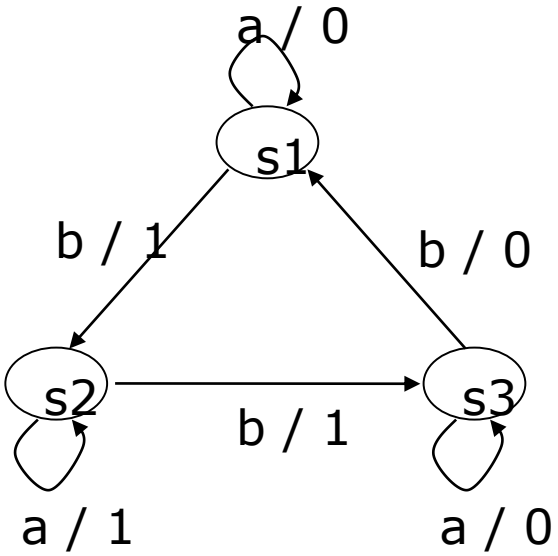
<B: implementation >



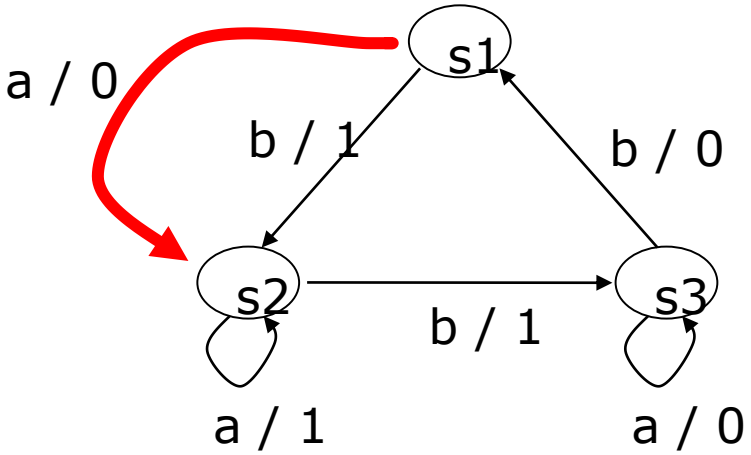
Transfer fault

Difetti di trasferimento

< specification >



< implementation >



Fault Model per SW

Nota che in un SW i modelli di difetti sono molto più numerosi e difficili da scoprire:

- difetti di uso di aritmetica
- chiamata a funzioni sbagliate
- difetto nella specifica del tipi di dato
- difetto di inizializzazione
- difetto nel numero di variabili
- uso di operatori logici errati
- ……

In sintesi

- Abbiamo visto che nel testing basato su specifiche:
  - la specifica è usata per definire criteri di testing
  - essa è usata come oracolo
- Ricordate che nel conformance testing basato sulle macchine stati finiti:
  - i casi di test sono generati dalla macchina S che rappresenta la specifica
  - vengono poi applicati alla macchina I per controllarne la conformità
  - i difetti ammessi in I (fault model) sono di due tipi: transfer faults e output faults

- **Conformance testing con le FSM**

Test di conformità

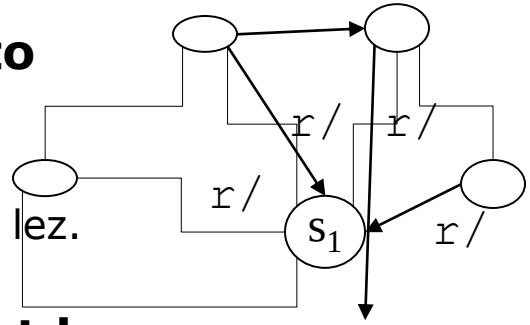
Nel test di conformità si testa che un'implementazione sia conforme alla sua specifica

In questa unità assumiamo che la specifica sia data come FSM ben formata (connessa, completamente specificata, deterministica,...)

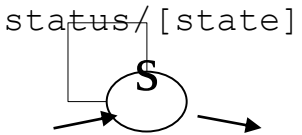
Messaggi di reset e status

Una FSM si può rinizializzare se ha un input particolare r che porta la macchine allo stato iniziale s1 da qualsiasi altro stato. Questo messaggio si chiama reset

- se non c'è reset si utilizza una homing sequence - lez. sulle sequenze particolari delle FSM



Il messaggio di stato, o status mostra in output lo stato corrente della macchina senza cambiarlo



Test sequence

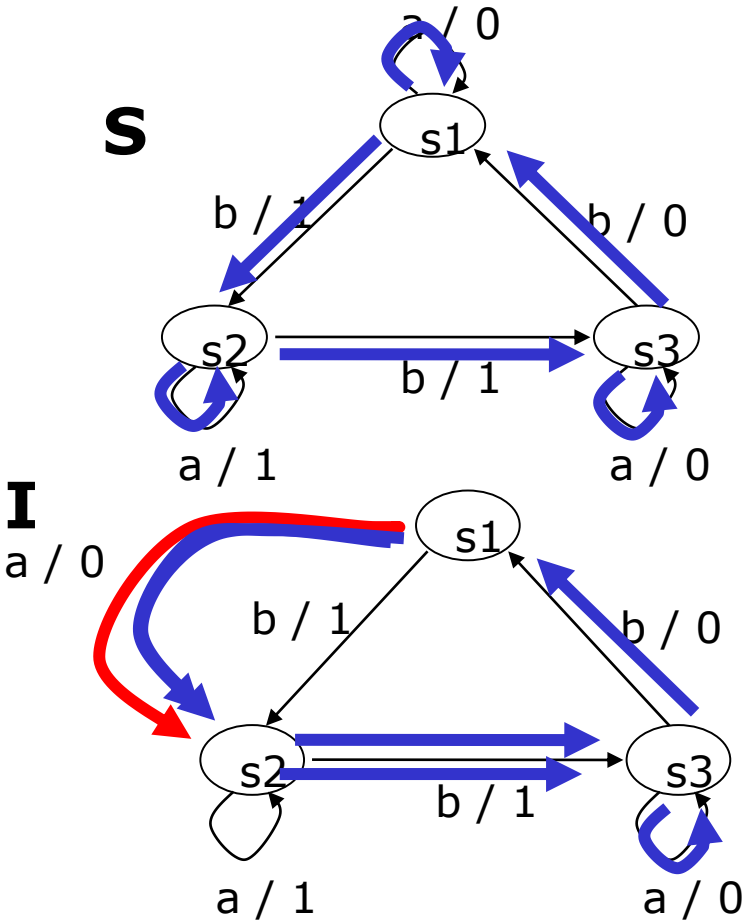
I metodi che vedremo generano un test set che spesso è formato da una sola sequenza di test (test sequence)

- in ogni caso usando reset si può trasformare un test set in un'unica test sequence

Applicando la test sequence a S e a I e osservando l'output si testa la conformità di I ad S

- nel caso la macchina abbia anche lo status message, si può osservare anche lo stato corrente applicando lo status

Esempio



Per S sia il test set T composto da una sola test sequence t:

ababab

Applicando t a S otteniamo l'output

011100

Sia I l'implementazione di S con un transfer error. Applichiamo t a I e otteniamo l'output

010001

I due output differiscono; test di conformità con successo: I non è conforme a S

Metodi

Esistono numerosi metodi:

- copertura degli stati - state cover method
- transition tour (TT) method
- DS-method (distinguishing sequences)
- W-method (characterizing sets)
- UIO-method (UIO sequences)

Differiscono tra loro per

- applicabilità
  - alcuni richiedono reset, status, o altre sequenze
- capacità a trovare i difetti
- lunghezza dei casi di test e complessità di calcolo

Coperture

Copertura degli stati un test set T è adeguato secondo la copertura degli stati di una FSM M se l'esecuzione da parte di M di tutte le sequenze di T causa la visita di ogni stato di M

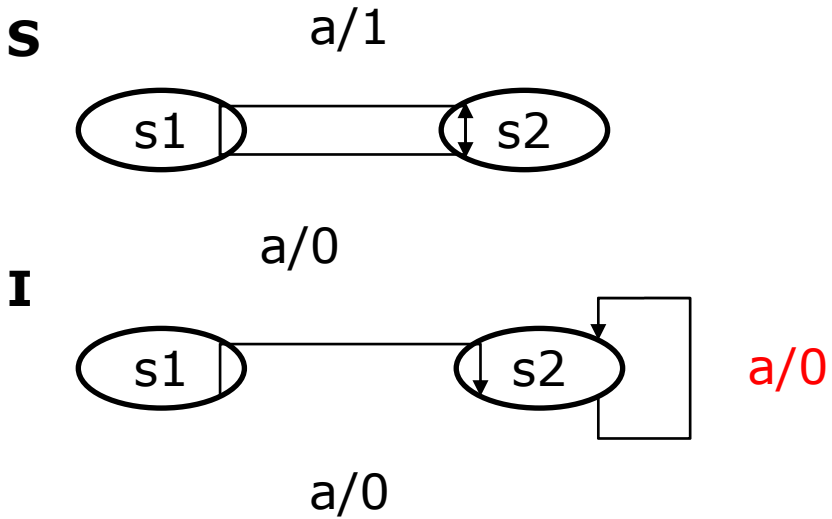
Copertura delle transizioni un test set T è adeguato secondo la copertura degli transizioni di una FSM M se l'esecuzione da parte di M di tutte le sequenze di T causa la visita di ogni transizione di M

Copertura degli stati e status

METODO 1: copertura degli stati e uso dello status message per verificare la correttezza dello stato corrente

Non garantisce la scoperta di alcun difetto

- se ho una transizione sbagliata e non la copro non me ne accorgo



Esempio

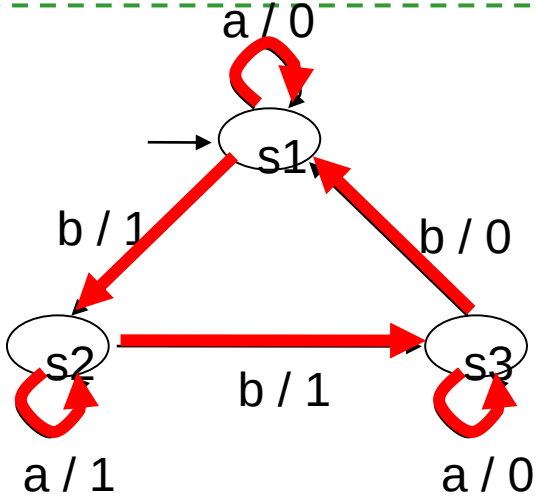
la test sequence:

status a status

copre tutti gli stati di S, applicata a I produce lo stesso output (*s1,0,s2*)

Un transition tour di una FSM M

è una sequenza di input che applicata a M nel suo stato iniziale attraversa tutte le transizioni di M almeno una volta (e ritorna allo stato iniziale)



Esempio:

la sequenza *ababab*

è un TT della macchina M

Come calcolare un transition tour

Problema classico dell'attraversamento dei grafi

il tour più corto si chiama "Euleriano"

Per macchine simmetriche (tanti archi uscenti quanti entranti) è semplice

- tempo lineare - vedi libro 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

Per macchine non simmetriche è complesso:

- Chinese Postman Problem, che può essere risolto in tempo polinomiale

Per il ns testing, TT anche non euleriano va bene

METODO 2: il test set ha una sola sequenza che sia un Transition Tour

Con due varianti:

con status message: applica lo status message dopo ogni input per verificare lo stato

- garantisce la scoperta di tutti i difetti sia di output che di transfer

senza status message

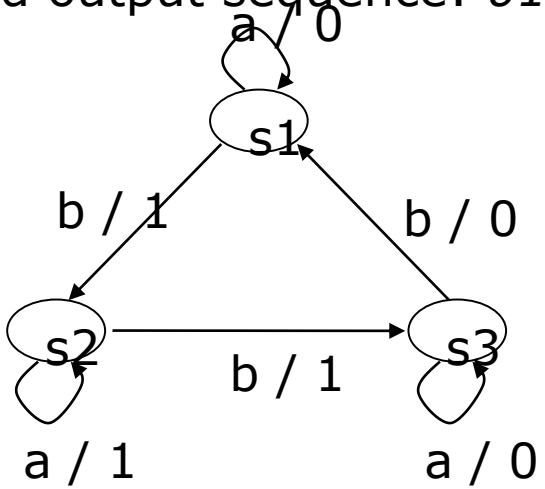
- garantisce la scoperta solo dei difetti di output ma non di transfer

Esempio (1)

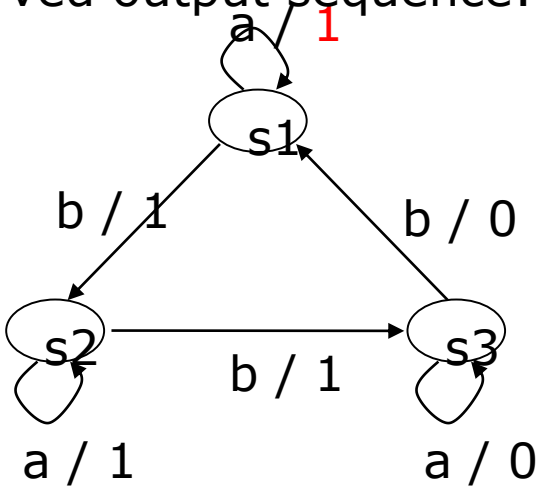
TTs scoprono tutti gli output faults

- ogni transizione viene testata

S
Input sequence: *abababab*
Expected output sequence: *011100*



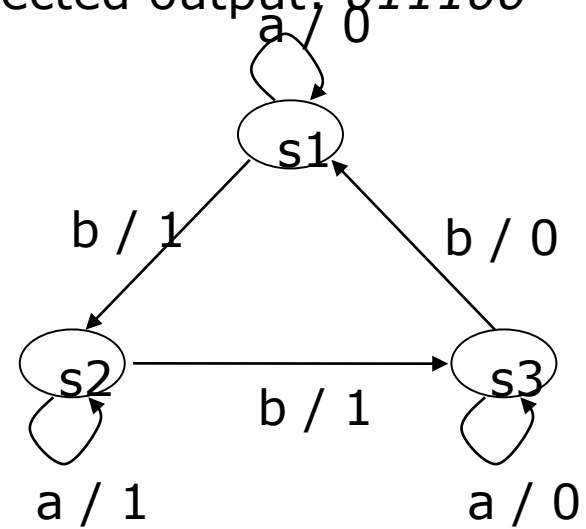
I (con output fault)
Input sequence: *abababab*
Observed output sequence: *111100*



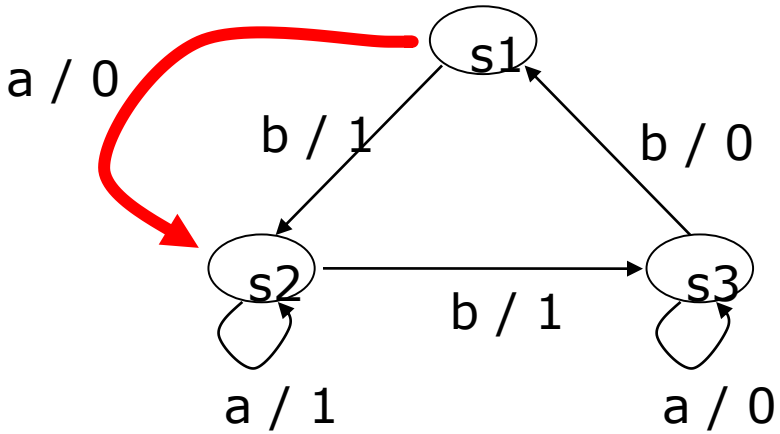
Esempio (2)

TT (senza status) trova alcuni transfer faults

S
Input sequence: *abababab*
Expected output: *011100*



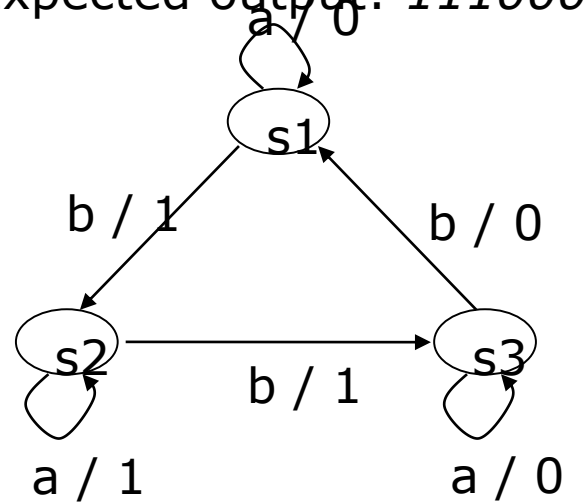
I (con transfer fault)
Input sequence: *abababab*
Observed output sequence: *010001*



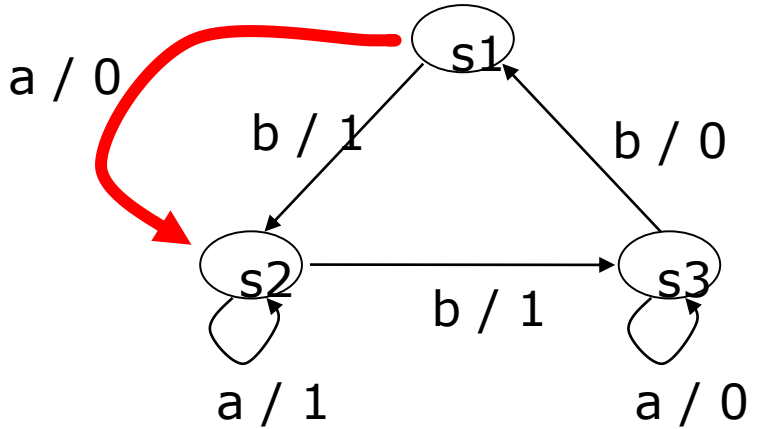
Esempio (3)

TT senza status non trova tutti i transfer faults

S Input (TT): *bababa*
Expected output: *111000*



I Input : *bababa*
Observed output: *111000*



per questo motivo i progettisti inseriscono in sistemi embedded spesso uno status message, cioè un input particolare che produce come output lo stato corrente

Senza Status Message

Con status message

- il metodo TT va bene: scopre tutti i difetti

Senza status message

- non riesco a scoprire se lo stato corrente è quello atteso e il TT non può garantire conformità
- se non c'è status devo capire in quale stato mi trovo applicando qualche input aggiuntivo e osservando solo gli output
  - uso di sequenze particolari al posto dello status
  - separating sequences, Distinguishing sequences, UIO sequences

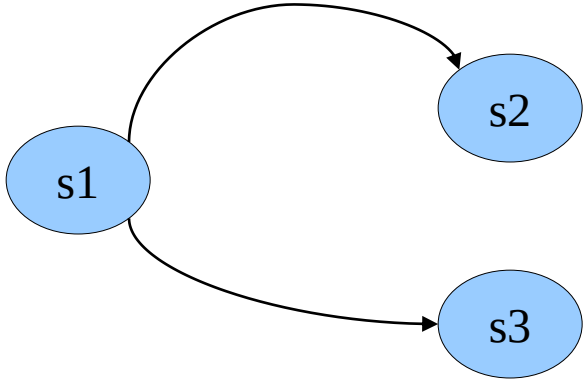
In sintesi

- Abbiamo visto:
  - reset: msg che porta la macchina al suo stato iniziale
  - status: msg che produce come output lo stato corrente
  - test di conformità: applica la test sequence a S e a I e confronta l'output osservato
- Ricorda che:
  - la copertura degli stati non garantisce alcunché
  - la copertura delle transizioni (con un transition tour) garantisce la scoperta di tutti i difetti nel caso ci sia lo status, altrimenti i transfer faults potrebbero non essere scoperti

- **Alcune estensioni**

Non determinismo

- **In caso la macchina sia non determinisca che fare?**



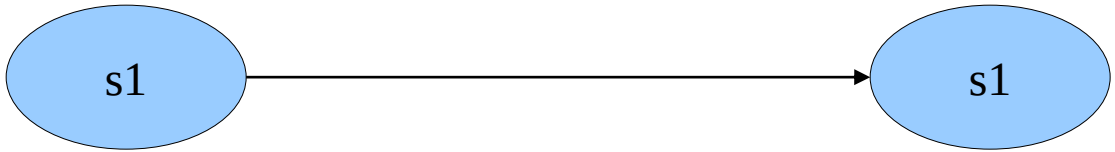
Alcune soluzioni possibili:

- Uso di runtime testing: genero mentre eseguo i casi di test
- Test non come sequenze ma come alberi

Aggiungo variabili

- **Spesso ho macchine con variabili**

- **Agiungo variabili agli stati**
- **+ guardie e assegnamenti**
- **EFSM**



Altri esempi: UML/ Abstract State machines/NuSMV

Esercizio FSM

- **Scrivi una macchina a stati finiti con almeno 4 stati e due input e due output e trovane un transition tuor (euleriano). Introduci un difetto e scopri se il tuo test è in grado di scoprirlo.**