FSM-Based Testing
Part II

Mohammad Mousavi

Eindhoven University of Technology, The Netherlands

Software Testing, 2011

Announcements

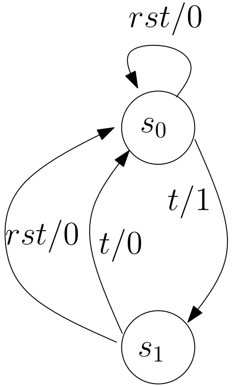
- Comments on the first deliverable are posted.
- Some common issues:
  - Strange moments of choice,
  - Unnamed or plain (neither input nor output) transitions, and
  - Initial state issue.
- A package for the second deliverable is also posted.

Outline

Finite State Machines (recap)

$M = (I, O, S, \delta, \lambda)$ with

- I, O, and S finite and non-empty sets of input symbols, output symbols, and states
- state transition function $\delta : S \times I \rightarrow S$
- output function $\lambda : S \times I \rightarrow O$

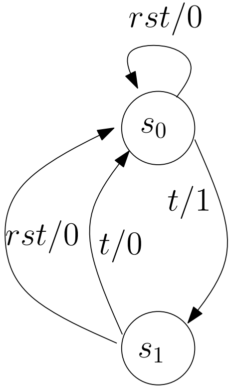


Finite State Machines (recap)

$M = (I, O, S, \delta, \lambda)$ with

- I, O, and S finite and non-empty sets of input symbols, output symbols, and states
- state transition function $\delta : S \times I \to S$
- output function $\lambda : S \times I \to O$

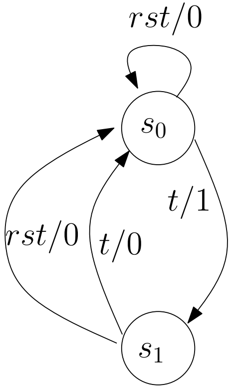
λ and δ are generalized to sets of states and sequences of inputs.



Finite State Machines (recap)

$M = (I, O, S, \delta, \lambda)$ with

- I, O, and S finite and non-empty sets of input symbols, output symbols, and states
- state transition function $\delta : S \times I \to S$
- output function $\lambda : S \times I \to O$



λ and δ are generalized to sets of states and sequences of inputs.

Assumption: FSM's are reduced, deterministic and complete.

Essential Notions (recap)

1. Reduced FSM:
   for all distinct $s, s' \in S$, there exists an $x \in I^*$ such that $\lambda(s, x) \neq \lambda(s', x)$. I.e., x separates s and s'.
2. Completeness:
   For each input a, and state s, there exists a s' such that $\delta(s, a) = s'$.
3. Determinism: For each input a, and state s, there exists at most one s' such that $\delta(s, a) = s'$.

Equivalence (recap)

Let $M = (I, O, S, \delta, \lambda)$ and $M' = (I, O, S', \delta', \lambda')$

1. State equivalence: for $s \in S$ and $s' \in S'$

$$s \approx s' \quad \doteq \quad \forall_{x \in I^*}\ \lambda(s, x) = \lambda'(s', x)$$

2. Machine equivalence

$$M \approx M' \quad \doteq \quad \forall_{s \in S} \exists_{s' \in S'}\ s \approx s' \wedge \forall_{s' \in S'} \exists_{s \in S}\ s' \approx s$$

Outline

Five Fundamental Testing Problems

1. Determine the final state $\surd$
   1.1 homing sequence: a testcase to reveal the final state
   1.2 synchronizing sequence for s: final state (after running the testcase) is s

Five Fundamental Testing Problems

1. Determine the final state $\surd$
   1.1 homing sequence: a testcase to reveal the final state
   1.2 synchronizing sequence for s: final state (after running the testcase) is s
2. State identification (identify the initial state) $\surd$
   2.1 preset distinguishing sequence
   2.2 adaptive distinguishing sequence

Five Fundamental Testing Problems

1. Determine the final state $\surd$
   1.1 homing sequence: a testcase to reveal the final state
   1.2 synchronizing sequence for *s*: final state (after running the testcase) is *s*
2. State identification (identify the initial state) $\surd$
   2.1 preset distinguishing sequence
   2.2 adaptive distinguishing sequence
3. State verification (is machine in state *s*?):
   UIO sequence $\surd$

Five Fundamental Testing Problems

1. Determine the final state $\surd$
   1.1 homing sequence: a testcase to reveal the final state
   1.2 synchronizing sequence for s: final state (after running the testcase) is s
2. State identification (identify the initial state) $\surd$
   2.1 preset distinguishing sequence
   2.2 adaptive distinguishing sequence
3. State verification (is machine in state s?): UIO sequence $\surd$
4. Conformance testing (is blackbox A equivalent to the FSM?)
5. Machine identification (derive the FSM from a blackbox)

Outline

Basic Idea

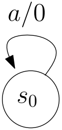
- Specification: FSM A
- Implementation: A blackbox B, only input-output observable
- Problem: given the specification
  determine a testcase (a sequence of inputs)
  to determine whether $A \approx B$
  (Also called: fault detection, machine testing)

Basic Idea

- Specification: FSM A
- Implementation: A blackbox B, only input-output observable
- Problem: given the specification
  determine a testcase (a sequence of inputs)
  to determine whether $A \approx B$
  (Also called: fault detection, machine testing)
- Challenge: for each testcase t, B can always
  behave the same as A up to $\mid t \mid$

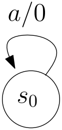
Example

- specification:



Example

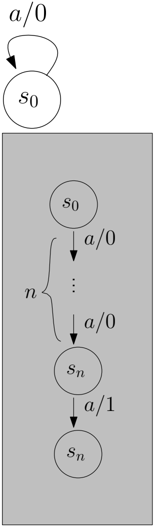
- specification:



- suppose that sequence a^n is the testcase (the answer to the conformance testing problem)

Example

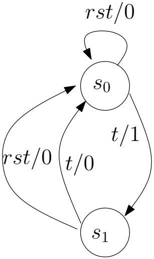
- specification:
- suppose that sequence a^n is the testcase (the answer to the conformance testing problem)
- for any n, some incorrect implementation may pass the test:



Simplifying Assumptions

Assumptions on A

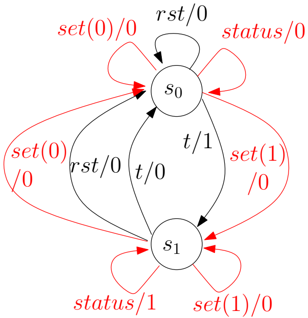
- strongly connected: (testcase long enough $\Rightarrow$ each state visited)
- reduced: (equivalence: interesting / efficient on reduced FSMs)



Simplifying Assumptions

Assumptions on A

- strongly connected: (testcase long enough $\Rightarrow$ each state visited)
- reduced: (equivalence: interesting / efficient on reduced FSMs)
- A has the following transitions
  - $set(j)/0$: a transition from each state to state j,
  - $status/i$: a self-loop indicating the current state



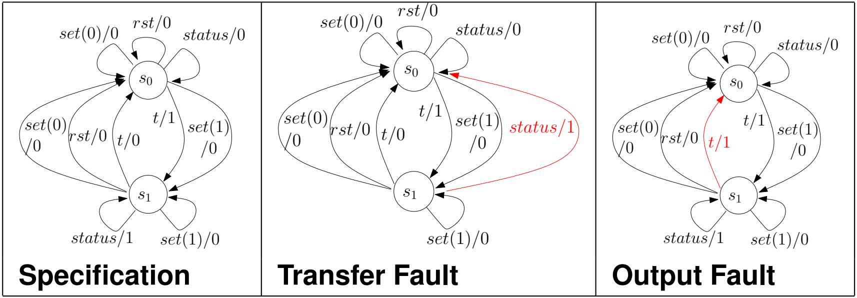
Last assumption: only for convenience; to be dropped later.

Simplifying Assumptions

Assumptions on B

- constant FSM
  (should not change; should be finite)
- at most | S_A | states
  (an upper bound is needed;
  here, only transfer and output faults tested
  no new states due to faults)

Fault Model



$set(0)/0$
$rst/0$
$status/0$
s_0
$set(0)$
$/0$
$rst/0$
$t/0$
$t/1$
$set(1)$
$/0$
s_1
$status/1$
$set(1)/0$
Specification
$set(0)/0$
$rst/0$
$status/0$
s_0
$set(0)$
$/0$
$rst/0$
$t/0$
$t/1$
$set(1)$
$/0$
$status/1$
s_1
$set(1)/0$
Transfer Fault
$set(0)/0$
$rst/0$
$status/0$
s_0
$set(0)$
$/0$
$rst/0$
$t/1$
$t/1$
$set(1)$
$/0$
s_1
$status/1$
$set(1)/0$
Output Fault

Basic Algorithm

For each state s and input a in A:

- set the state to s,
- supply input a and check in B whether
  1. output is $\lambda(s, a)$, and
  2. the target is $\delta(s, a)$.

N.B. all transitions are covered by the algorithm (a transition tour is taken).

Basic Algorithm

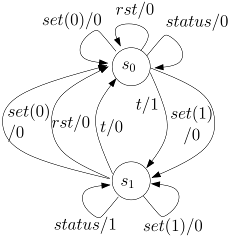
For each state s and input a in A:

- set the state to s, using $set(s)$
- supply input a and check in B whether
  1. output is $\lambda(s, a)$ (observe the output), and
  2. the target is $\delta(s, a)$ (supply *status* and observe the output).

N.B. all transitions are covered by the algorithm (a transition tour is taken).

Example

testcase:
set(0), status, status, status, rst, status, t, status, set(1), status,
status, status, t, status, set(1), rst, status, set(1), set(0), status

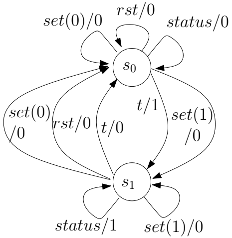


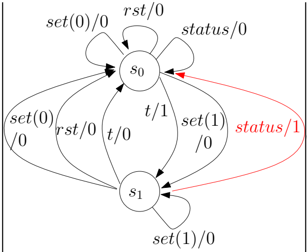
Specification		
0,0, 0,0, 0, 0, 1, 1, 0,1, 1,1, 0, 0, 0,0,0, 0,0,0		

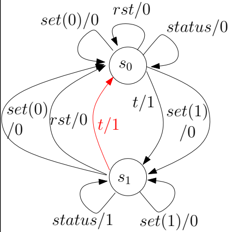
Example

testcase:

set(0), status, status, status, rst, status, t, status, set(1), status, status, status, t, status, set(1), rst, status, set(1), set(0), status





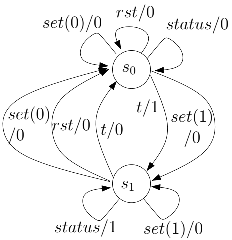


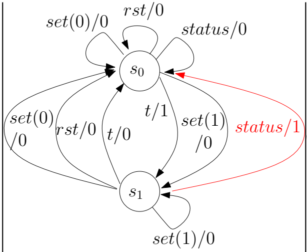
Specification	**Transfer Fault**	**Output Fault**
0,0, 0,0, 0, 0, 1, 1, 0,1, 1,1, 0, 0, 0,0,0, 0,0,0		

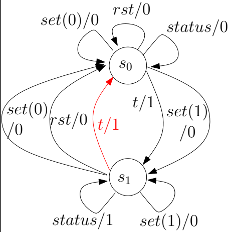
Example

testcase:

set(0), status, status, status, rst, status, t, status, set(1), status,
status, status, t, status, set(1), rst, status, set(1), set(0), status





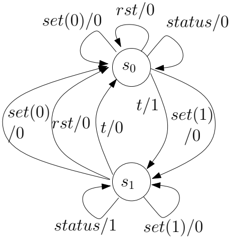


Specification	Transfer Fault	Output Fault
0,0, 0,0, 0, 0, 1, 1, 0,1,	0,0, 0, 0, 0, 0, 1,1, 0,1,	
1,1, 0, 0, 0,0,0, 0,0,0	1, 0, 1,1, 0,0,0 0,0,0	

Example

testcase:

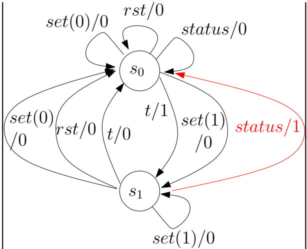
set(0), status, status, status, rst, status, t, status, set(1), status, status, status, t, status, set(1), rst, status, set(1), set(0), status



Specification

0,0, 0,0, 0, 0, 1, 1, 0,1,

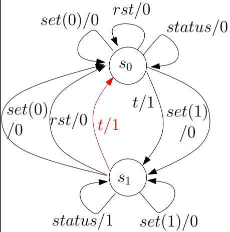
1,1, 0, 0, 0,0,0, 0,0,0



Transfer Fault

0,0, 0, 0, 0, 0, 1,1, 0,1,

1, 0, 1,1, 0,0,0 0,0,0



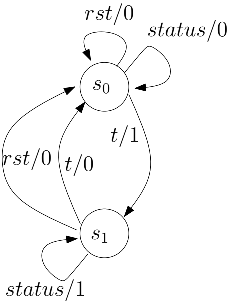
Output Fault

0,0, 0, 0, 0, 0, 1,1, 0,1,

1,1, 1, 0, 0,0,0 0,0,0

Transfer Sequence

- FSM's usually have no *set*(j): use homing sequence and transfer sequences
- A transfer sequence $\tau(i, j)$: $\delta(s_i, \tau(s_i, s_j)) = s_j$
- $\tau(i, j)$ need not be unique; the shortest path can be found efficiently
- Examples:
  $HS = t$
  $\tau(0, 1) = t$,
  $\tau(1, 0) = t$ or rst.



Algorithm with Transfer/Homing Sequences

1. Go to a known final state s'
2. For each state s and input a in A:
   - set the state from s' (the current known state) to s,
   - supply input a and check in B whether
     2.1 output is $\lambda(s, a)$, and
     2.2 the target is $\delta(s, a)$

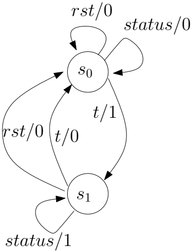
Algorithm with Transfer/Homing Sequences

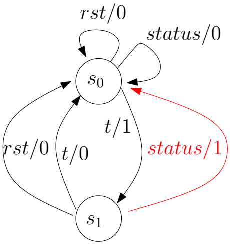
1. Go to a known final state s' (using the homing sequence)
2. For each state s and input a in A:
   - set the state from s' (the current known state) to s, using $\tau(s', s)$
   - supply input a and check in B whether
     2.1 output is $\lambda(s, a)$ (observe the output), and
     2.2 the target is $\delta(s, a)$ (supply *status* and observe the output, if successful let $s' = \delta(s, a)$)

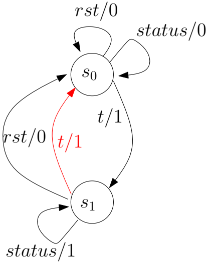
Example

testcase: first supply $HS = t$, if $output = 1$ then supply t, ts, otherwise, ts, where:

ts = status, status, status, rst, status, t, status, status, status, rst, status, t, t, status







Specification	**Transfer Fault**	**Output Fault**
1,0,0, 0,0, 0,0, 1,1, 1,1, 0,0, 1,0,0	1,0,0, 0,0, 0, 0, 1,1, 0,0, , 0,0, 1,0,0	1,1,0, 0,0, 0,0, 1,1, 1,1, 0,0, 1,1,0

“Status”, Realistic?

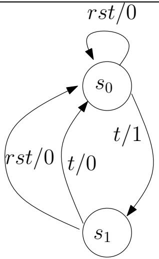
- Sometimes present: registers in hardware, state dumps (logs) in protocols
- Usually not: let’s do without them and apply state identification (e.g., preset or adaptive distinguishing sequence)
- Challenge: distinguishing sequences change the state

Algorithm with Nuts and Bolts

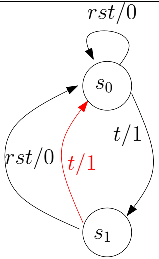
1. Go to a known final state s_0 (homing sequence)
2. Take a sequence of all states $s_0, \ldots, s_{n+1}$, where $s_0 = s_{n+1}$. (state tour or state cover seq.) and let $i = 0$:
   2.1 supply the DS and check if FSM is in s_i,
   2.2 FSM is in t_i (due to DS), supply $\tau(t_i, s_{i+1})$ and repeat for $i := i + 1$, until $i = n + 1$
3. For each state s_i and input label a, (assuming that current state is s'_j)
   3.1 supply $\tau(s'_j, s_{i-1})$, DS, $\tau(t_{i-1}, s_i)$, take yourself back to the safe path,
   3.2 supply the input a, DS and check whether the output is $\lambda(s_i, a)$ and FSM is in $\delta(s_i, a)$, the current state is s'_{j+1} (due to a, DS)

   N.B. In step 3.1, one may apply $\tau(t_{i-1}, s_i)$, if $s'_j = t_{i-1}$ and skip the step if $s'_j = s_i$.

Example



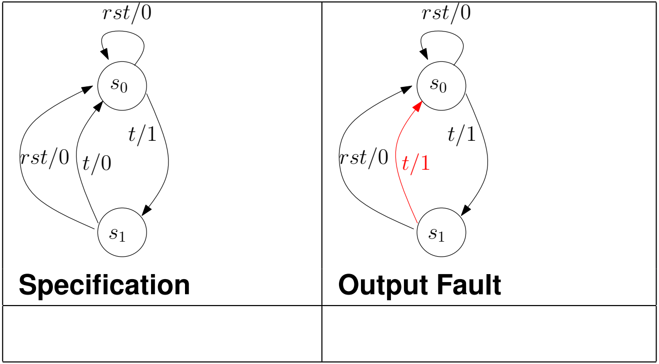
Specification



Output Fault

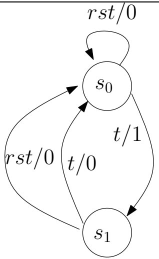
Example

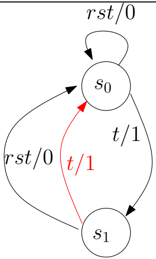
testcase: first supply $HS = t, rst$, $output = 1, 0$ (no other choice!), then supply $t, rst, t, rst, t, t, rst, t$



Example

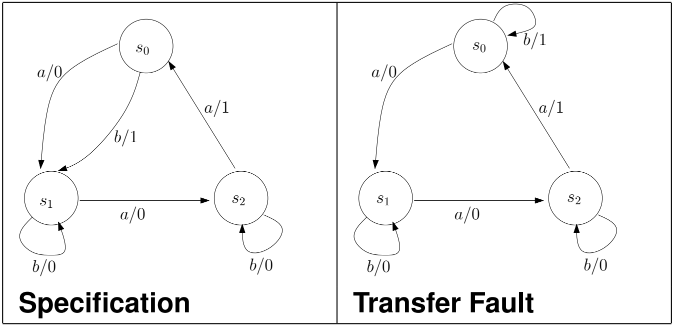
testcase: first supply $HS = t, rst$, $output = 1, 0$ (no other choice!), then supply $t, rst, t, rst, t, t, rst, t$





Specification	**Output Fault**
1, 0, 1, 0, 1, 0, 1	1, 1, 1, 0, 1, 1, 0, 1

Example



s_0
$a/0$
$b/1$
$a/1$
s_1
s_2
$a/0$
$b/0$
$b/0$
Specification
s_0
$b/1$
$a/0$
$a/1$
s_1
s_2
$a/0$
$b/0$
$b/0$
Transfer Fault

Solution

- Homing sequence: *ba*

observed output	00	01	10
target state	s_2	s_0	s_2

- Adaptive distinguishing sequences:
  $DS(s_0) = aa,\ \lambda(s_0, aa) = 00$
  $DS(s_1) = aa,\ \lambda(s_1, aa) = 01$
  $DS(s_2) = a,\ \lambda(s_2, a) = 1$

Outline

Basic Idea

- Implementation: A blackbox B, only input-output observable
- Problem: by applying a testcase (a sequence of inputs) determine an FSM A such that $A = B$
- Same challenges as in conformance checking: B should be strongly connected, finite and constant. But that is not all...

Simple solution

- Construct all different reduced and strongly connected FSM's with n states and I_B and O_B as inputs and outputs,
- Use conformance testing: find the one conforming to B!

Example

- State-space explosion:
  with n states, p inputs and q outputs, $(nq)^{np}/n!$ machines
  2 states, 2 inputs, 2 outputs, 256 machines!
- Possible Solutions:
  - Run all possible machines simultaneously (construct “direct sum” machines)
  - Use machine learning: have an oracle which provides a test-case for each failure

Study Guide

- Only sections II, III, IV.A, IV.B, IV.C and VII.A of Lee and Yananakis's paper were treated.
- There will be a study-guide on the web page, to guide you through the available material.