5. Test Generation – Finite State Models

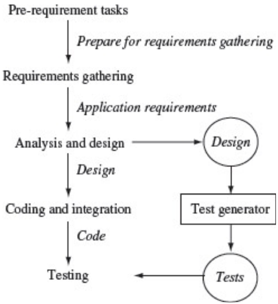
Barbara Re

Advanced Topics on Software Engineering – Software Testing
MSc in Computer Science
University of Camerino

Models in the Design Phase

Design Phase

- Between the requirements phase and the implementation phase "*The last you start the first you finish*"
- Produce models in order to clarify requirements and to better formalize them
- Models can be the source of test set derivation strategies
- Tests can be generated directly from formal expressions of software designs, such tests can be used to test an implementation against its design



Simplified Software Development Process

Various modeling notations for behavioral specification of a software system have been proposed, which to use depends on the system you are developing, and the aspects you would like to highlight:

- Finite State Machines
- Petri Nets
- Statecharts
- Message sequence charts

Embedded Computer

Many devices used in daily life contain embedded computers

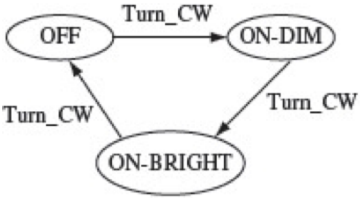
- Child's musical keyboard
- Computer inside a toy for processing inputs and generating audible and visual responses
- Engine control of an automotive
- Flight controller in an aircraft

Embedded Computer

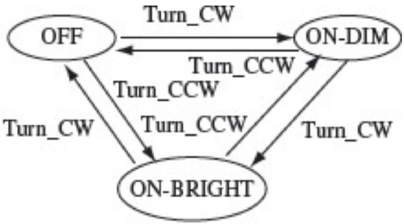
An embedded computer often receives inputs from its environment and responds with appropriate actions. While doing so, it moves from one state to another. The response of an embedded system to its inputs depends on its current state.

The behavior of an embedded system in response to inputs that is often modeled by an Finite State Machines (FSM)

Change of lamp state

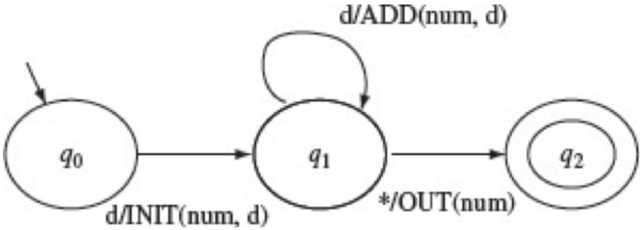


(a)



(b)

State diagram of the DIGDEC machine



Moore and Mealy

- FSMs that do not associate any action with a transition are known as Moore machines (actions depend on the current state)
- FSMs that do associate actions with each state transition are known as Mealy machines. A Mealy machine has a finite set of outputs.

We are concerned with Mealy machines!!!!

Finite State Machines

FSM

A finite state machine is a six-tuple $<\mathscr{X}, \mathscr{Y}, \mathscr{Q}, q_0, \delta, \mathscr{O}>$ where:

- $\mathscr{X}$: finite set of input symbols
- $\mathscr{Y}$: finite set of output symbols
- $\mathscr{Q}$: finite set of states
- $q_0 \in \mathscr{Q}$: initial state
- δ: transition function ($\mathscr{Q} \times \mathscr{X} \to \mathscr{Q}$)
- $\mathscr{O}$: output function ($\mathscr{Q} \times \mathscr{X} \to \mathscr{Y}$)

Many possible extensions:

- Transition and output functions can consider strings
- Definiton of the set of final or accepting states $\mathscr{F} \subseteq \mathscr{Q}$ (mainly used as an automaton to recognize a language)
- The transition function implies that for any state $q_i \in \mathscr{Q}$ there is at most one next state - Non determinism

Properties of FSM

Useful properties/concepts for test generation

- Completely specified (input enabled)
  - An FSM M is said to be completely specified if from each state in M there exists a transition for each input symbol
  - $\forall(q_i \in \mathcal{Q}, a \in \mathcal{X}).\exists q_j \in \mathcal{Q}.\delta(q_i, a) = q_j$
- Strongly connected
  - An FSM is strongly connected if for every pair of states, say (q_1, q_2), there exists an input sequence that takes it from q_1 to q_2
  - $\forall(q_i, q_j) \in \mathcal{Q} \times \mathcal{Q}.\exists s \in X^*.\delta^*(q_i, s) = q_j$
  - in a strongly connected FSM, every state is reachable from the initial state

Properties of FSM....cntd

Useful properties/concepts for test generation

- V-equivalence (distinguishable)
  - Let M_1 and M_2 two FSMs. Let $\mathscr{V}$ denote a set of non-empty string on the input alphabet $\mathscr{X}$, and $q_i \in \mathscr{Q}_1$ and $q_j \in \mathscr{Q}_2$. q_i and q_j are considered *$\mathscr{V} - equivalent$* if $\mathscr{O}_1(q_i, s) = \mathscr{O}_2(q_j, s)$. If q_i and q_j are $\mathscr{V} - equivalent$ given any set $\mathscr{V} \subseteq \mathscr{X}^+$ than they are said to be *equivalent* ($q_i \equiv q_j$). If states are not equivalent they are said to be *distinguishable*.
  - This definition of equivalence also applies to states within a machine. Thus, machines M_1 and M_2 could be the same machine.

Properties of FSM....cntd

Useful properties/concepts for test generation...cntd

- Machine equivalence
  - M_1 and M_2 are said to be *equivalent* if $\forall q_i \in \mathcal{Q}_1 . \exists q_j \in \mathcal{Q}_2 . q_i \equiv q_j$ and viceversa.
- k-equivalence
  - Two FSMs are k-equivalent if no string of length k or less over its input alphabet can distinguish them
  - Let M_1 and M_2 two FSMs and $q_i \in \mathcal{Q}_1$ and $q_j \in \mathcal{Q}_1$ and $k \in \mathbb{N}$. q_i and q_j are said to be $\mathscr{K}$ *− equivalent* if they are $\mathscr{V}$ *− equivalent* for $\mathscr{V} = \{s \in X^+ | \mid s \mid \leq k\}$
- Minimal machine
  - an FSM is considered *minimal* if the number of its states is less than or equal to any other *equivalent* FSM

Conformance Testing

Conformance Testing

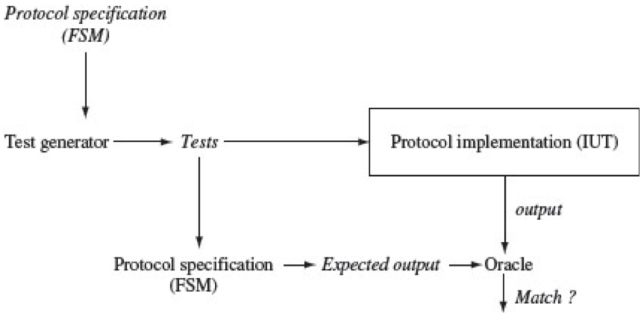
Relates to testing of communication protocols. It aims at assessing that an implementation of a protocol conform to its specification. Protocols implementation generally specify:

- Control rules (modelled by FSM)
- Data rules (modelled by program segments)

Testing an implementation of a protocol involves testing both the control and data portions (we concentrate on control)

Note that the term conformance testing applies equally well to the testing of any implementation that corresponds to its specification, regardless of whether or not the implementation is that of a communication protocol.

A simplified procedure for testing a protocol implementation against an FSM model



Reset inputs

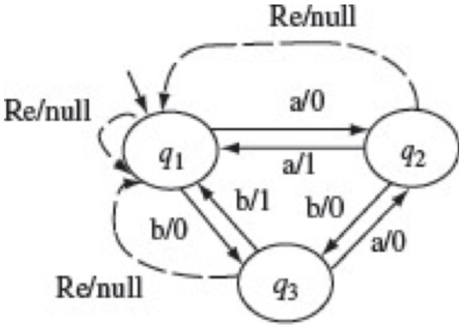
Thus, given a set of test cases $T = \{t_1, t_2, ..., t_n\}$, a test proceeds as follows:

1. Bring the IUT to its start state. Repeat the following steps for each test in T.
2. Select the next test case from T and apply it to the IUT. Observe the behavior of the IUT and compare it against the expected behavior. The IUT is said to have failed if it generates an output different from the expected output.
3. Bring the IUT back to its start state by applying the reset input and repeat the above step but with the next test input.

It is usually assumed that the application of the reset input generates a null output. Thus, for the purpose of testing an IUT against its control specification FSM the input and output alphabets are augmented as follows ($\mathscr{X} = \mathscr{X} \cup \{Re\}$, and $\mathscr{Y} = \mathscr{Y} \cup \{null\}$) where *Re* denotes the reset input and *null* the corresponding output.

Transitions corresponding to reset (Re) inputs

The transitions corresponding to the reset input are generally not shown in the state diagram. In a sense, these are hidden transitions that are allowed during the execution of the implementation.

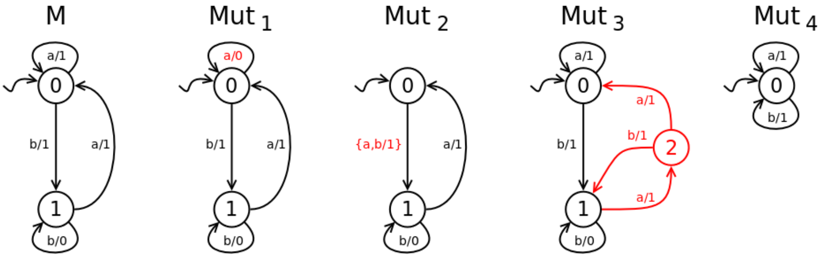


The Testing Problem

FSM and Testing

- Testing based on requirements checks if the implementation conforms to the machine on a given requirement.
- The testing problem is reconducted to an equivalence (nevertheless finite experiments). Is the SUT (IUT) equivalent to the machine defined during design?
- Fault model for FSM – given a fault model the challenge is to generate a test set T from a design M_d where any fault in M_i of the type in the fault model is guaranteed to be revealed when tested against T
  - Operation error (refers to issues with $\mathscr{O}$)
  - Transfer error (refers to issues with δ)
  - Extra-state error (refers to issues with $\mathscr{Q}$ and δ)
  - Missing-state error (refers to issues with $\mathscr{Q}$ and δ)

Fault Model

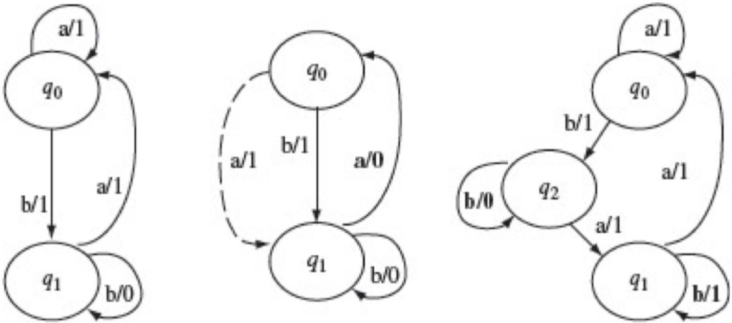

M
a/1
0
b/1
a/1
1
b/0
Mut 1
a/0
0
b/1
a/1
1
b/0
Mut 2
0
{a,b/1}
a/1
1
b/0
Mut 3
a/1
0
a/1
b/1
b/1
2
a/1
1
b/0
Mut 4
a/1
0
b/1

Mutation of FSMs

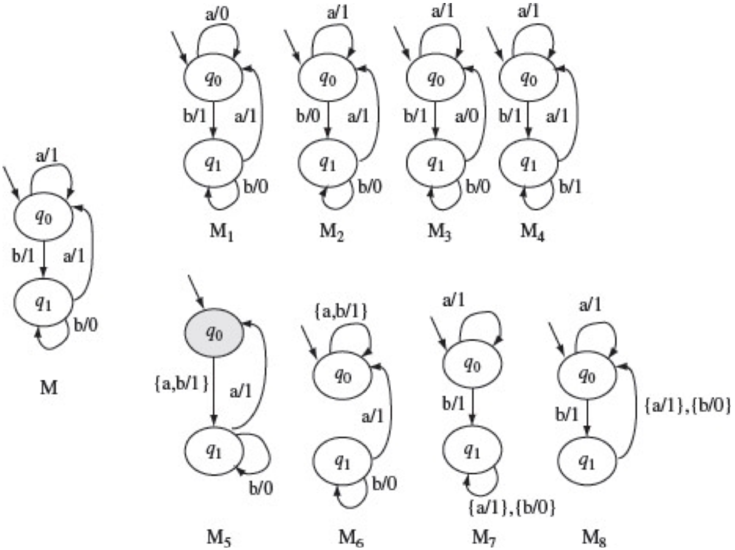
Mutant

A mutant of an FMS M_d is an FSM obtained by introducing one one or more errors one or more times.

- Equivalent mutants: mutants that could not be distinguishable from the originating machine



Mutation of FSMs: Examples



The Testing Problem - Fault coverage

- Techniques to measure the goodness of a test set in relation to the number of errors that it reveals in a given implementation M_i
- Methods for the generation of test sets are often evaluated based on their fault coverage. The fault coverage of a test set is measured as a fraction between 0 and 1 and with respect to a given design specification.

The Testing Problem

Fault coverage

- N_t: total number of first order mutants of the machine M used for generating tests.
- N_e: Number of mutants that are equivalent to M
- N_f: Number of mutants that are distinguished by test set T generated using some test generation method
- N_l: Number of mutants that are not distinguished by T

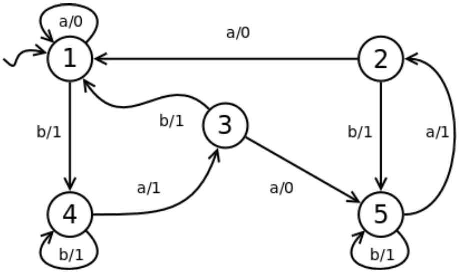
The fault coverage of a test suite T with respect to a design M is denoted by $FC(T, M)$ and computed as follows:

$$\begin{aligned} FC(T, M) &= \text{Number of mutants not distinguished by T /} \\ &\quad\ \text{Number of mutants that are not equivalent to M} \\ &= (N_t - N_e - N_f)/(N_t - N_e) \end{aligned}$$

Characterization Set

Characterization Set useful in various methods for generating tests from FSMs

Let $M = <\mathscr{X}, \mathscr{Y}, \mathscr{Q}, q_1, \delta, \mathscr{O}>$ an FSM that is minimal and complete. A characterization set for M, denoted as $\mathscr{W}$, is a finite set of input sequences that distinguish the behaviour of any pair of states in M.



Characterization Set

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

- The first step is to construct a sequence of k-equivalence partitions $P_1, P_2, ..., P_m$ where $m > 0$. This iterative step converges in at most n steps where n is the number of states in M
- 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 partitions

The notion of $\mathscr{K}-$ *equivalence* leads to the notion of $\mathscr{K}-$ *equivalence partitions*

Given an FSM a $\mathscr{K}-$ *equivalence partition* of $\mathscr{Q}$, denoted by $\mathscr{P}_k$, is a collection of n finite sets of states denoted as $\Sigma_{k_1}, \Sigma_{k_2}, ..., \Sigma_{k_n}$ such that:

- $\cup_{i=1...n}\Sigma_{K_i} = \mathscr{Q}$
- States in Σ_{k_j}, for $1 \leq j \leq n$ are $\mathscr{K}-$ *equivalent*
- if $q_l \in \Sigma_{k_i}$ and $q_m \in \Sigma_{k_j}$, for $i \neq j$, then q_l and q_m must be $\mathscr{K}-$ *distinguishable*

$\mathscr{K}-$ *equivalence* partitions can be derived using an iterative approach for increasing number of $\mathscr{K}$

K-equivalence partitions

The notion of $\mathscr{K}$ − *equivalence* leads to the notion of $\mathscr{K}$ − *equivalence partitions*

Given an FSM a $\mathscr{K}$ − *equivalence partition* of $\mathscr{Q}$, denoted by $\mathscr{P}_k$, is a collection of n finite sets of states denoted as $\Sigma_{k_1}, \Sigma_{k_2}, ..., \Sigma_{k_n}$ such that:

- $\cup_{i=1...n}\Sigma_{K_i} = \mathscr{Q}$
- States in Σ_{k_j}, for $1 \leq j \leq n$ are $\mathscr{K}$ − *equivalent*
- if $q_l \in \Sigma_{k_i}$ and $q_m \in \Sigma_{k_j}$, for $i \neq j$, then q_l and q_m must be $\mathscr{K}$ − *distinguishable*

$\mathscr{K}$ − *equivalence* partitions can be derived using an iterative approach for increasing number of $\mathscr{K}$

Let's use the intuition

Let's build K-equivalnce partitions for the previous FSM

Computing 1-equivalence partition

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

State transition and output table for M.

Computing 1-equivalence partition

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

State transition and output table for M with grouping indicated.

The construction of P1 is now complete

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

State transition and output table for M with grouping indicated.

The construction of P1 is now complete

The groups separated by the horizontal line constitute a 1-equivalence partition. We have labeled these groups as 1 and 2. Thus, we get the 1-equivalence partition as

$P_1 = \{1, 2\}$
$Group1 = \Sigma_{11} = \{q_1, q_2, q_3\}$
$Group2 = \Sigma_{12} = \{q_4, q_5\}$

In preparation to begin the construction of the 2-equivalence partition

Σ	Current state	Next state	
		a	b
1	q_1	q_{11}	q_{42}
	q_2	q_{11}	q_{52}
	q_3	q_{52}	q_{11}
2	q_4	q_{31}	q_{42}
	q_5	q_{21}	q_{52}

P_1 table.

P2 table

Σ	Current state	Next state	
		a	b
1	q_1	q_{11}	q_{43}
	q_2	q_{11}	q_{53}
2	q_3	q_{53}	q_{11}
3	q_4	q_{32}	q_{43}
	q_5	q_{21}	q_{53}

P_2 table.

P3 table

Σ	Current state	Next state	
		a	b
1	q_1	q_{11}	q_{43}
	q_2	q_{11}	q_{54}
2	q_3	q_{54}	q_{11}
3	q_4	q_{32}	q_{43}
4	q_5	q_{21}	q_{54}

P_3 table.

P4 table

Σ	Current state	Next state	
		a	b
1	q_1	q_{11}	q_{44}
2	q_2	q_{11}	q_{55}
3	q_3	q_{55}	q_{11}
4	q_4	q_{33}	q_{44}
5	q_5	q_{22}	q_{55}

P_4 table.

P4 table

Note that no further partitioning is possible using the scheme described earlier. We have completed the construction of k-equivalence partitions, k = 1,2,3,4, for machine M.

How to derive $\mathscr{W}$ from K-equivalence partitions

1. Let M an FSM for which $P = \{P_1, P_2, ..., P_n\}$ is the set of k-equivalence partition. $\mathscr{W} = \emptyset$
2. Repeat the steps (a) through (d) given below for each pair of states (q_i, q_j), $i \neq j$, in M
   (a) Find r ($1 \leq r < n$ such that the states in pair (q_i, q_j) belong to the same group in P_r but not in P_{r+1}. If such an r is found then move to step (b) otherwise we find an $\eta \in \mathscr{X}$ such that $\mathscr{O}(q_i, \eta) \neq \mathscr{O}(q_j, \eta)$, set $\mathscr{W} = \mathscr{W} \cup \{\eta\}$ and continue with the next available pair of states. The length of the minimal distinguishing sequence for (q_i, q_j) is $r + 1$.
   (b) Initialize $z = \epsilon$. Let $p_1 = q_i$ and $p_2 = q_j$ be the current pair of states. Execute steps (i) through (iii) given below for $m = r, r - 1, ..., 1$
      (i) Find an input symbol η in P_m such that $\mathscr{G}(p_1, \eta) \neq \mathscr{G}(p_2, \eta)$. In case there is more than one symbol that satisfy the condition in this step, then select one arbitrarily.
      (ii) set $z = z\eta$
      (iii) set $p_1 = \delta(p_1, \eta)$ and $p_2 = \delta(p_2, \eta)$
   (c) Find an $\eta \in \mathscr{X}$ such that $\mathscr{O}(p_1, \eta) \neq \mathscr{O}(p_2, \eta)$. Set $z = z\eta$
   (d) The distinguishing sequence for the pair (q_i, q_j) is the sequence z. Set $\mathscr{W} = \mathscr{W} \cup \{z\}$

Example

- Termination of the *$\mathscr{W}$ – procedure* guarantees the generation of distinguishing sequence for each pair.

S_i	S_i	x	$\mathscr{O}(S_i, x)$	$\mathscr{O}(S_j, x)$
1	2	baaa	1	0
1	3	aa	0	1
1	4	a	0	1
1	5	a	0	1
2	3	aa	0	1
2	4	a	0	1
2	5	a	0	1
3	4	a	0	1
3	5	a	0	1
4	5	aaa	1	0

Example

- Termination of the $\mathscr{W}$ – *procedure* guarantees the generation of distinguishing sequence for each pair.

S_i	S_i	x	$\mathscr{O}(S_i, x)$	$\mathscr{O}(S_j, x)$
1	2	baaa	1	0
1	3	aa	0	1
1	4	a	0	1
1	5	a	0	1
2	3	aa	0	1
2	4	a	0	1
2	5	a	0	1
3	4	a	0	1
3	5	a	0	1
4	5	aaa	1	0

The W-Method

The W-Method aims at deriving a test set to check the implementation (Implementation Under Test - IUT) of an FSM model

Assumptions

- M is completely specified, minimal, connected, and deterministic
- M starts in a fixed initial states
- M can be reset to the initial state. A `null` output is generated by the reset
- M and IUT have the same input alphabet

The W-Method

The W-Method aims at deriving a test set to check the implementation (Implementation Under Test - IUT) of an FSM model

Assumptions

- M is completely specified, minimal, connected, and deterministic
- M starts in a fixed initial states
- M can be reset to the initial state. A `null` output is generated by the reset
- M and IUT have the same input alphabet

W-Method steps

Given an FSM $\mathscr{M} =< \mathscr{X}, \mathscr{Y}, \mathscr{Q}, q_0, \delta, \mathscr{O} >$ the W-method consists of the following steps:

1. Estimate the maximum number of states in the correct design
2. Construct the characterization set $\mathscr{W}$ for the given machine $\mathscr{M}$
3. Construct the testing tree for $\mathscr{M}$ and determine the transition cover set $\mathscr{P}$
4. Construct set $\mathscr{Z}$
5. $\mathscr{P} \cdot \mathscr{Z}$ is the desired test set

Computation of the transition cover set

$\mathscr{P}$ - transition cover set

Let q_i and $q_j, i \neq j$ be two states of $\mathscr{M}$. $\mathscr{P}$ consists of sequences $s \cdot x$ s.t. $\delta(q_0, s) = q_i \wedge \delta(q_i, x) = q_j$ for $s \in \mathscr{X}^* \wedge x \in \mathscr{X}$. The set can be constructed using the testing tree for $\mathscr{M}$.

Testing tree

The testing tree for an FSM $\mathscr{M}$ can be constructed as follows:

1. State q_0 is the root of the tree
2. Suppose that the testing tree has been constructed till 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 to $k-1$ then n is a leaf node. Otherwise expand it by adding branch from node n to a new node m if $\delta(n, x) = m$ for $x \in \mathscr{X}$. This branch is labeled as x.

Computation of the transition cover set

$\mathscr{P}$ - transition cover set

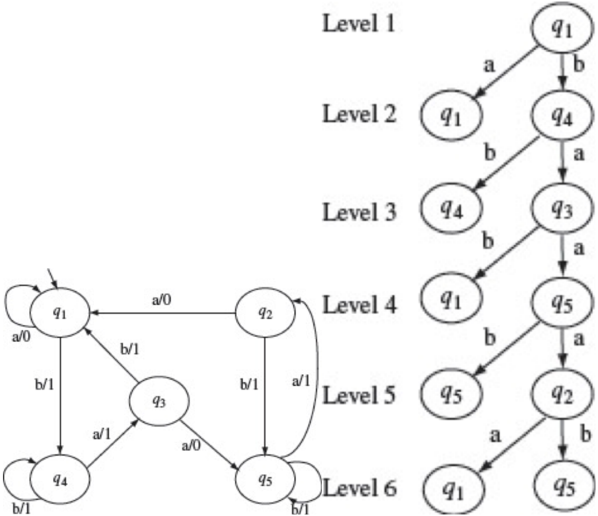
Let q_i and $q_j, i \neq j$ be two states of $\mathscr{M}$. $\mathscr{P}$ consists of sequences $s \cdot x$ s.t. $\delta(q_0, s) = q_i \wedge \delta(q_i, x) = q_j$ for $s \in \mathscr{X}^* \wedge x \in \mathscr{X}$. The set can be constructed using the testing tree for $\mathscr{M}$.

Testing tree

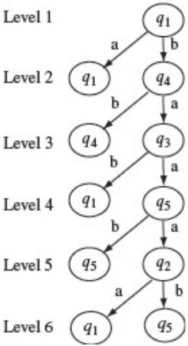
The testing tree for an FSM $\mathscr{M}$ can be constructed as follows:

1. State q_0 is the root of the tree
2. Suppose that the testing tree has been constructed till 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 to $k-1$ then n is a leaf node. Otherwise expand it by adding branch from node n to a new node m if $\delta(n, x) = m$ for $x \in \mathscr{X}$. This branch is labeled as x.

Testing tree



Testing tree



The transition cover set *P* is obtained *P* by concatenating labels of all partial paths along the tree.

$$P = \{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\}$$

Constructing $\mathscr{Z}$

The set $\mathscr{Z}$

Suppose number of states estimates to be *m* for the IUT, and *n* in the specification $m > n$.
We compute $\mathscr{Z}$ as:
$\mathscr{Z} = (\mathscr{X}^0 \cdot \mathscr{W}) \cup (\mathscr{X} \cdot \mathscr{W}) \cup (\mathscr{X}^1 \cdot \mathscr{W}) \cdots \cup (\mathscr{X}^{m-1-n} \cdot \mathscr{W}) \cup (\mathscr{X}^{m-n} \cdot \mathscr{W})$

Deriving a test set

Having constructed $\mathscr{P}$ and $\mathscr{Z}$, we can easily obtain a test set $\mathscr{T}$ as $\mathscr{P} \cdot \mathscr{Z}$

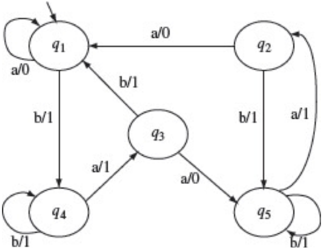
$$
\begin{aligned}
T = P \cdot Z = \{&\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\} \cdot \{a, aa,\\
&aaa, baaa\}\\
= \{&a, aa, aaa, baaa,\\
&aa, aaa, aaaa, abaaa,\\
&ba, baa, baaa, bbaaa,\\
&bba, bbaa, bbaaa, bbbaaa,\\
&baa, baaa, baaaa, babaaa,\\
&baba, babaa, babaaa, babbaaa,\\
&baaa, baaaa, baaaaa, baabaaa,\\
&baaba, baabaa, baabaaa, baabbaaa,\\
&baaaa, baaaaa, baaaaaa, baaabaaa\\
&baaaba, baaabaa, baaabaaa, baaabbaaa\\
&baaaaa, baaaaaa, baaaaaaa, baaaabaaa\}
\end{aligned}
$$

Deriving a test set - II

$$Z = X^0 \cdot W \cup X^1 \cdot W = \{a, aa, aaa, baaa, aa, aaa, aaaa, abaaa,$$
$$ba, baa, baaa, bbaaa\}$$

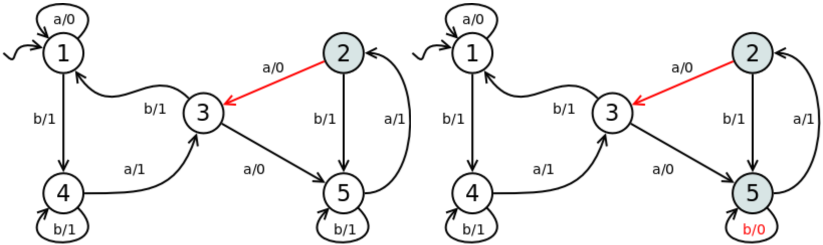
$$T = P \cdot Z = \{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\} \cdot$$
$$\{a, aa, aaa, baaa, aa, aaa, aaaa, abaaa, ba, baa, baaa, bbaaa\}$$

Testing using the $\mathscr{W}$-method



We assume that the specification is also given in terms of an FSM which we refer to as the correct design

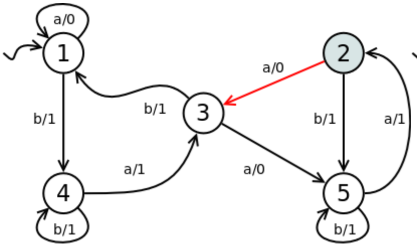
Deriving a test set – $\mathscr{P} \cdot \mathscr{Z}$

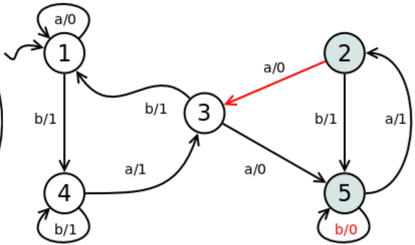


Try sequences:

- *ba*
- *baaaaaa*
- *baaba*

Deriving a test set – $\mathscr{P} \cdot \mathscr{Z}$

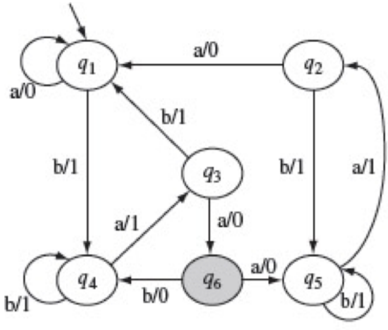




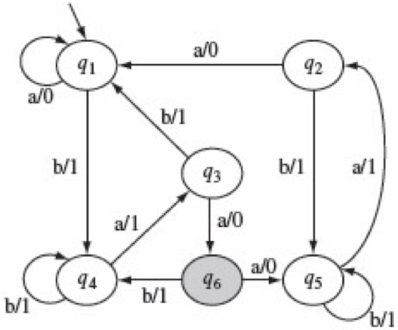
Try sequences:

- *ba*
- *baaaaaa*
- *baaba*

Example - Extra State



(a)

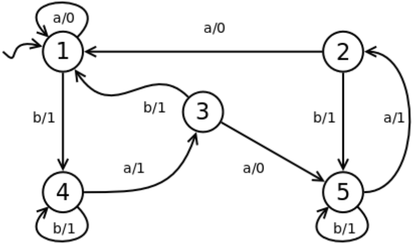


(b)

$\mathscr{W}$-method fault detection rationale

- A test case generated by the $\mathscr{W}$ − *method* is of the form $r \cdot s$ where $r \in \mathscr{P}$ and $s \in \mathscr{W}$
  - Why can we detect operation errors?
  - Why can we detect transfer errors?

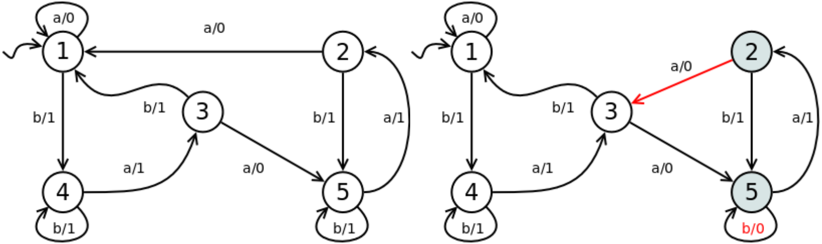
$$\mathscr{P} = \{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\}$$
$$\mathscr{W} = \{a, aa, aaa, baaa\}$$



$\mathscr{W}$-method fault detection rationale

- A test case generated by the $\mathscr{W}$ − *method* is of the form $r \cdot s$ where $r \in \mathscr{P}$ and $s \in \mathscr{W}$
  - Why can we detect operation errors?
  - Why can we detect transfer errors?

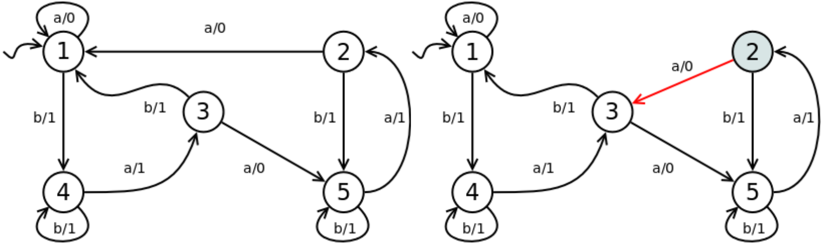
$$\mathscr{P} = \{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\}$$
$$\mathscr{W} = \{a, aa, aaa, baaa\}$$



$\mathscr{W}$-method fault detection rationale

- A test case generated by the $\mathscr{W}$ − *method* is of the form $r \cdot s$ where $r \in \mathscr{P}$ and $s \in \mathscr{W}$
  - Why can we detect operation errors?
  - Why can we detect transfer errors?

$$\mathscr{P} = \{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\}$$
$$\mathscr{W} = \{a, aa, aaa, baaa\}$$



The partial $\mathscr{W}$ – *method* (aka *Wp* – *method*)

Wp – *method*

Main characteristics:

- It considers minimal, complete and connected FSM
- is inspired by the $\mathscr{W}$ – *method* it generates smaller test sets
- uses a derivation phase split in two phases that make use of state identification sets $\mathscr{W}_i$ instead of characterization set $\mathscr{W}$
- uses the state cover set ($\mathscr{S}$) to derive the test set.

Identification Set and State Cover Set

Identification Set

The Identification Set is associated to each state $q \in \mathcal{Q}$ of an FSM.

An Identification set for state $q_i \in \mathcal{Q}$, where $|\mathcal{Q}| = n$, is denoted by $\mathcal{W}_i$ and has the following properties:

1. $\mathcal{W}_i \subseteq \mathcal{W}$ per $1 < i \leq n$
2. $\exists j, s.1 \leq j \leq n \wedge s \in \mathcal{W}_i \wedge \mathcal{O}(q_i, s) \neq \mathcal{O}(q_j, s)$
3. No subset of $\mathcal{W}_i$ satisfies property 2.

State Cover Set

The state cover set is a nonempty set of sequences ($\mathcal{S} \subseteq \mathcal{X}^*$ s.t.:

- $\forall q_i \in \mathcal{Q}\ \exists r \in \mathcal{S} s.t. \delta(q_0, r) = q_i$

From the definition it is evident that $\mathcal{S} \subseteq \mathcal{P}$

Identification Set and State Cover Set

Identification Set

The Identification Set is associated to each state $q \in \mathcal{Q}$ of an FSM.

An Identification set for state $q_i \in \mathcal{Q}$, where $|\mathcal{Q}| = n$, is denoted by $\mathscr{W}_i$ and has the following properties:

1. $\mathscr{W}_i \subseteq \mathscr{W}$ per $1 < i \leq n$
2. $\exists j, s.1 \leq j \leq n \wedge s \in \mathscr{W}_i \wedge \mathscr{O}(q_i, s) \neq \mathscr{O}(q_j, s)$
3. No subset of $\mathscr{W}_i$ satisfies property 2.

State Cover Set

The state cover set is a nonempty set of sequences ($\mathscr{S} \subseteq \mathscr{X}^*$ s.t.:

- $\forall q_i \in \mathcal{Q}\ \exists r \in \mathscr{S} s.t. \delta(q_0, r) = q_i$

From the definition it is evident that $\mathscr{S} \subseteq \mathscr{P}$

The $\mathscr{W}p$ procedure (assuming $m = n$)

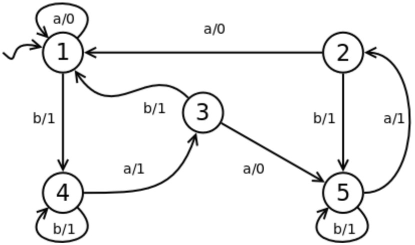
The test set derived using the $\mathscr{W}p - method$ is given by the union to two test sets $\mathscr{T}_1$, $\mathscr{T}_2$ calculated according to the following procedure:

1. Compute sets $\mathscr{P}$, $\mathscr{S}$, $\mathscr{W}$, and $\mathscr{W}_i$
2. $\mathscr{T}_1 = \mathscr{S} \cdot \mathscr{W}$
3. Let $\mathcal{W} = \{\mathscr{W}_1, \mathscr{W}_2, \ldots, \mathscr{W}_n\}$
4. Let $\mathcal{R} = \{r_1, r_2, \ldots, r_k\}$ where $\mathcal{R} = \mathscr{P} - \mathscr{S}$ and $r_j \in \mathcal{R}$ is s.t. $\delta(q_0, r_j) = q_i$
5. $\mathscr{T}_2 = \mathcal{R} \otimes \mathcal{W} = \cup_{j=1}^{K}(\{r_j\} \cdot \mathscr{W}_i)$ where $\mathscr{W}_i \in \mathcal{W}$ is the state identification set for state q_i ($\otimes$ is the partial string concatenation operator)

$\mathscr{W}p$ – *method* rationale

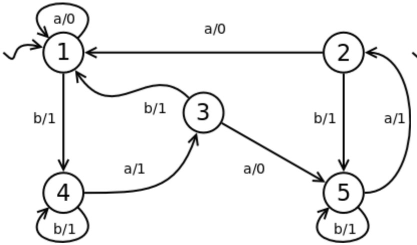
- Phase 1: test are of the form *uv* where $u \in \mathscr{S}$ and $v \in \mathscr{W}$. Reach each state than check if it is distinguishable from another one
- Phase 2: test covers all the missing transitions and then check if the reached state is different from the one specified in the model

$\mathscr{W}p - method$ in practice



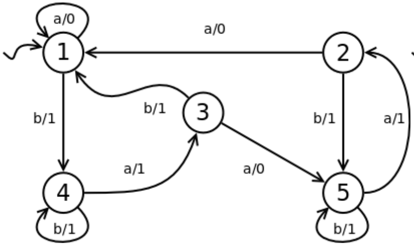
$\mathscr{W} = \{a, aa, aaa, baaa\}$
$\mathscr{P} = \{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\}$
$\mathscr{S} = \{\epsilon, b, ba, baa, baaa\}$
$\mathscr{W}_1 = \{baaa, aa, a\}, \mathscr{W}_2 = \{baaa, aa, a\}, \mathscr{W}_3 = \{aa, a\}$
$\mathscr{W}_4 = \{aaa, a\}, \mathscr{W}_5 = \{aaa, a\}$

$\mathscr{W}p - method$ in practice



$\mathscr{W} = \{a, aa, aaa, baaa\}$
$\mathscr{P} = \{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\}$
$\mathscr{S} = \{\epsilon, b, ba, baa, baaa\}$
$\mathscr{W}_1 = \{baaa, aa, a\}, \mathscr{W}_2 = \{baaa, aa, a\}, \mathscr{W}_3 = \{aa, a\}$
$\mathscr{W}_4 = \{aaa, a\}, \mathscr{W}_5 = \{aaa, a\}$

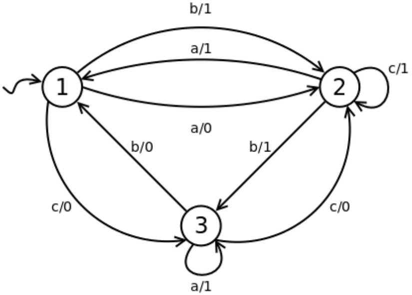
$\mathscr{W}p - method$ in practice



$\mathscr{W} = \{a, aa, aaa, baaa\}$
$\mathscr{P} = \{\epsilon, a, b, bb, ba, bab, baa, baab, baaa, baaab, baaaa\}$
$\mathscr{S} = \{\epsilon, b, ba, baa, baaa\}$
$\mathscr{W}_1 = \{baaa, aa, a\}, \mathscr{W}_2 = \{baaa, aa, a\}, \mathscr{W}_3 = \{aa, a\}$
$\mathscr{W}_4 = \{aaa, a\}, \mathscr{W}_5 = \{aaa, a\}$

Is it phase 2 needed?

Let's consider the following FSM:



Now introduce an operation error or a transfer error on a "*c*" transition

The $\mathscr{W}p$ procedure (assuming $m > n$)

Modify the derivation of the two sets as follows:

- $\mathscr{T}_1 = \mathscr{S} \cdot \mathscr{Z}$ where $\mathscr{Z} = \mathscr{X}[m-n] \cdot \mathscr{W}$
- $\mathscr{T}_2 = (\mathcal{R} \cdot \mathscr{X}[m-n]) \otimes \mathcal{W}$
  - Let $\mathcal{S} = \mathcal{R} \cdot \mathscr{X}[m-n] = \{s | s = r \cdot u \text{ s.t. } r \in \mathcal{R} \wedge u \in \mathscr{X}[m-n]\}$ then $\mathscr{T}_2 = \mathcal{S} \otimes \mathscr{W} = \cup_{s \in \mathscr{S}}(s \cdot \mathscr{W}_l)$ where $\delta(q_0, s) = \delta(\delta(q_0, r), u) = q_l$

Possible alternatives to W-method

- W-method high effectiveness in bugs identification
- High number of generated tests

To solve this issue alternative solutions have been proposed possibly reducing effectiveness:

- UIO-sequence method
- Distinguishing signatures

UIO-Sequence Method

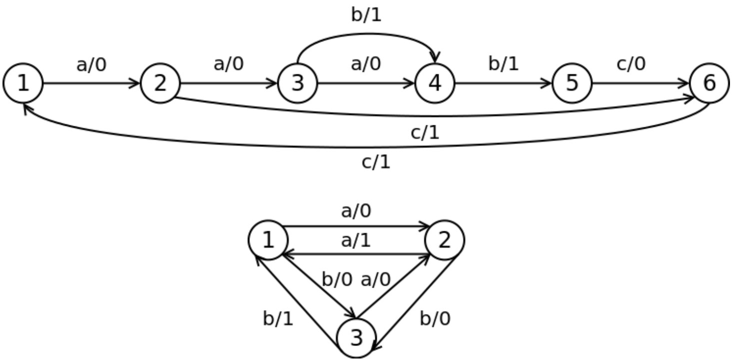
Assumptions

- M is completely specified, minimal, connected, and deterministic
- M starts in a fixed initial states
- M can be reset to the initial state. A `null` output is generated by the reset
- M and IUT have the same input alphabet
- M and IUT have the same number of states

UIO-Sequence

A UIO sequence is a sequence of input and output pairs that distinguish a state q of an *FSM* from the remaining states.
$UIO(s) = i_1/o_1, i_2/o_2, \ldots, i_n/o_n$ s.t. $\forall q' \in \mathcal{Q} s.t. q' \neq q. \exists j \in [1 \ldots n]. \mathcal{O}(\delta(q', i_1 i_2 \ldots i_{j-1}), i_j) \neq \mathcal{O}(\delta(q, i_1 i_2 \ldots i_{j-1}), i_j)$

UIO-Sequence examples



The UIO sequence does not always exist

Distinguishing Signatures

Distinguishing Signature or Sequence (DS)

Sequence of input/output labels that is unique to a state s

Minimal transfer sequence

A minimal transfer sequence is a sequence of input/output that brings the machine from state j to state i along the shortest path $\mathcal{P}_i(j)$

Given a state i a DS can be built using the identification set and minimal transfer sequences for each state j with $j \neq i$. In particular for an *FSM M* with k states a DS is given by the following concatenation: $DS(q_i) = W(q_i, q_1) \cdot P_i(t_1) \cdot W(q_i, q_2) \cdots P_i(t_{k-1}) W(q_i, q_k)$

Distinguishing Signatures

Distinguishing Signature or Sequence (DS)

Sequence of input/output labels that is unique to a state *s*

Minimal transfer sequence

A minimal transfer sequence is a sequence of input/output that brings the machine from state *j* to state *i* along the shortest path $\mathcal{P}_i(j)$

Given a state *i* a DS can be built using the identification set and minimal transfer sequences for each state *j* with $j \neq i$. In particular for an *FSM M* with *k* states a DS is given by the following concatenation:
$DS(q_i) = W(q_i, q_1) \cdot P_i(t_1) \cdot W(q_i, q_2) \cdots P_i(t_{k-1}) W(q_i, q_k)$

Test generation

Let $M = \langle \mathcal{Q}, \mathcal{X}, \mathcal{Y}, q_1, \delta, \mathcal{O} \rangle$ an FSM and $\mathcal{E} = \{\langle q_i, x, y, q_j \rangle | q_i, q_j \in \mathcal{Q} \wedge x \in \mathcal{X} \wedge y \in \mathcal{Y} \wedge \delta(q_i, x) = q_j \wedge \mathcal{O}(q_i, x) = y\}$ the set of edges of M

1. Find the UIO for each state in M
2. Find the shortest path from the initial state to each of the remaining states.
3. For each edge $e = \langle q_i, x, y, q_j \rangle \in \mathcal{E}$, build $\mathcal{TE}(e) = \mathcal{P}_{head(e)}(1) \cdot label(e) \cdot UIO(tail(e))$ where $head(e) = q_i$, $tail(e) = q_j$, $label(e) = x/y$
4. Optionally a unique sequence can be derived using reset actions.

Assessment of automata theoretic strategies

Control Flow based techniques are typically assessed according to different criteria:

State coverage

A test set T is considered adequate with respect to the state cover criterion for an FSM M if the execution of M agianst each element of T causes eash state in M to be visited at least once

Transition coverage

A test set T is considered adequate with respect to the branch, or 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

Assessment of automata theoretic strategies

Switch coverage (n-switch coverage)

A test set T is considered adequate with respect to the 1-switch cover criterion for an FSM M if the execution of M against each element of T causes each pair of transition (tr_1, tr_2) in M to be taken at least once, where for some input substring $ab \in X^*$, $tr_1 : q_i = \delta(q_j, a) \wedge tr_2 : q_k = \delta(q_i, b)$ and q_i, q_j, q_k are states of M

Boundary-interior coverage

A test set T is considered adequate with respect to the boundary-interior cover criterion for an FSM M if the execution of M against each element of T causes each loop body to be traversed zero times and at least once. Exiting the loop upon arrival covers the "boundary" condition and entering it and traversing the body at least once covers the "interior" condition.