

# Model Based Testing

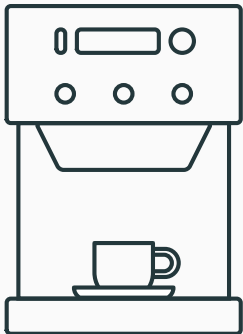
---

Dr Michael Foster

*Based on material from Professor Rob Hierons*

- Model-based Testing
- Formal models of systems (FSMs)
- Testing from finite state machines
- Identifying states
  - Distinguishing sequences
  - Unique I/O sequences (UIOs)
  - The  $W$  method

# Motivating Example - Simple Drinks Machine



- Select a drink
- Insert coins
- Press “vend” to dispense drink

# How do we test this system?

## Unit testing

- Generate tests according to code components
- Aim to achieve some level of code coverage

What if we don't have the source code?

## Model based testing

- Generate tests according to a formal model
- Model is a specification or description of a property of interest, often an abstraction and relatively understandable
- Aim to achieve some level of model coverage
- Usually black-box and complements white-box testing
- Major benefits realised if the model has a formal semantics – potential for automation!

# Formal Models of Systems

There are lots of different modelling notations (Z, B, state machines)

We will introduce MBT with state machines

An FSM (Mealy machine) is a sextuple  $(S, s_0, X, Y, \delta, \lambda)$  where

- $S$  is a finite set of states
  - $s_0 \in S$  is the initial state
  - $X$  is the input alphabet
  - $\delta : (S, X) \rightarrow S$  is the state transition function
  - $Y$  is the output alphabet
  - $\lambda : (S, X) \rightarrow Y$  is the output function
- } Just like a DFA<sup>1</sup>

We can extend  $\delta$  and  $\lambda$  to take sequences giving  $\delta^*$  and  $\lambda^*$

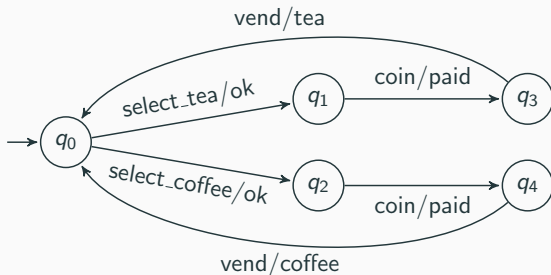
---

<sup>1</sup>DFA: Deterministic Finite Automaton

What types of faults can we test for with FSMs?

- A transition produces the wrong output (output faults) – often the easiest to find by simply executing the faulty transition to observe the erroneous output
- A transition has the wrong ending state (state transfer faults) – requires executing the transition and then showing that an erroneous state has been reached; methods that identify or separate states are therefore required
- There are extra states; can only happen if there are also state transfer faults

# FSM of the Drinks Machine



- We give the system inputs and it changes state and gives us outputs
- Transition  $x/y$  represents giving input  $x$  and getting output  $y$ , e.g.
  - inputs `select_tea`, `coin`, `vend`, gives us outputs `ok`, `paid`, `tea`
  - Machine follows path  $q_0 \rightarrow q_1 \rightarrow q_3 \rightarrow q_0$

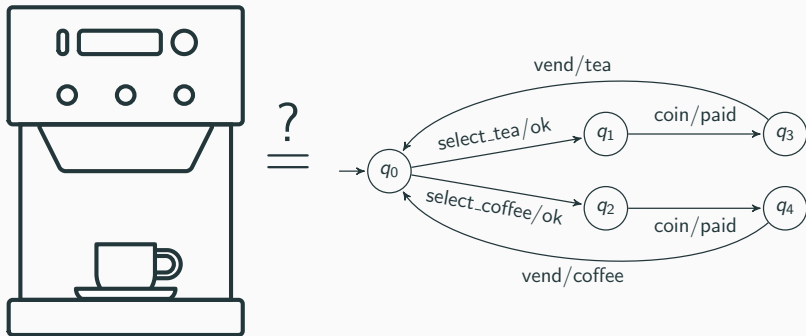
# Terminology

We will focus on FSM testing with *deterministic*, *minimal*, and *completely specified* models

- An FSM is *deterministic* if there is at **most** one transition from each state for each input (implicit since  $\delta$  and  $\lambda$  are functions)
- An FSM is *completely specified* if there is at **least** one transition from each state for each input
  - We can make any FSM complete with an “error state” or “null actions”
- An FSM is *minimal* if there is no trace-equivalent FSM with fewer states, i.e., if no smaller FSM in terms of number of states defines the same regular language
  - Note: we can convert any FSM into an equivalent minimal FSM



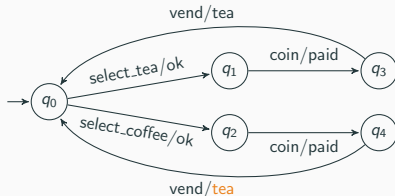
# Testing from an FSM



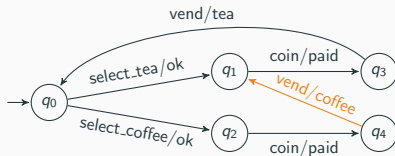
- Assume the software behaves like an FSM model
- Submit inputs to the FSM and software in parallel
- Observe and compare the outputs

# Example Faults for Drinks Machine

**Output faults:** A transition produces the wrong output (tea)



**State transfer faults:** A transition goes to the wrong state ( $q_1$ )

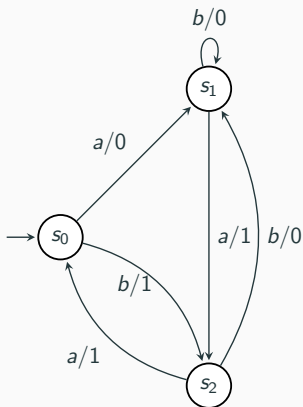


# The Transition Tour Method

- Given an FSM  $M$ , the transition tour method involves:
  - Finding a path (sequence of transitions)  $\rho$  from the initial state of  $M$  that includes **all** transitions of  $M$ .
  - The test sequence is the label of  $\rho$ : the corresponding input/output sequence.
- The transition tour method is guaranteed to find **output faults** as long as there are no state transfer faults.

# The Transition Tour Method: Example

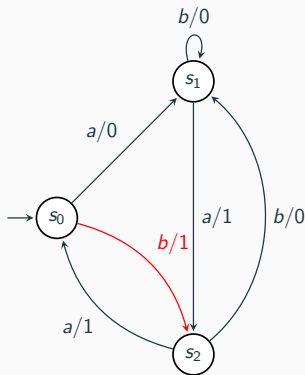
Consider a simple example



There is more than one possible transition tour

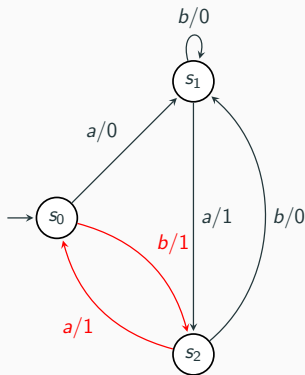
## Transition Tour Method: Example

We could start with transition  $(s_0, s_2, b/1)$ .



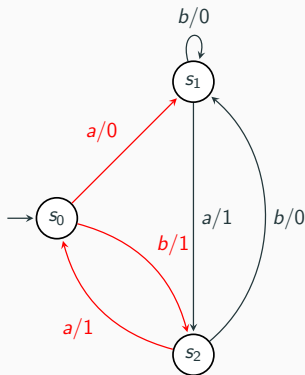
## Transition Tour Method: Example

Then follow  $(s_0, s_2, b/1)$  by  $(s_2, s_0, a/1)$ .



## Transition Tour Method: Example

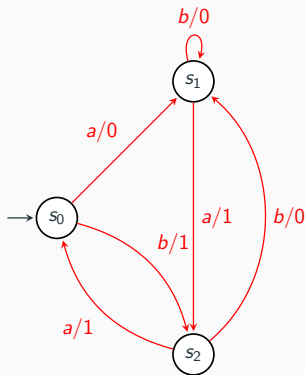
Now maybe follow  $(s_0, s_1, a/0)$ , giving path  
 $(s_0, s_2, b/1)(s_2, s_0, a/1)(s_0, s_1, a/0)$ .



# Transition Tour Method: Example

Completing this, we could choose:

- $(s_0, s_2, b/1)(s_2, s_0, a/1)(s_0, s_1, a/0)$  followed by
- $(s_1, s_1, b/0)(s_1, s_2, a/1)(s_2, s_1, b/0)$

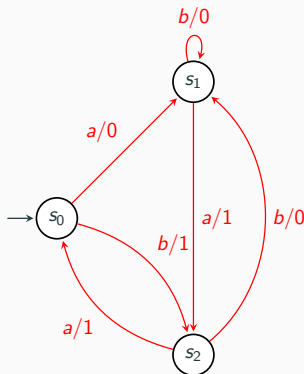




## Transition Tour Method: Example

This gives us the following test sequence:  $b/1, a/1, a/0, b/0, a/1, b/0$ .

- In testing we apply input sequence  $baabab$  and expect to observe output sequence  $110010$
- The test sequence can also be represented by  $baabab/110010$ .



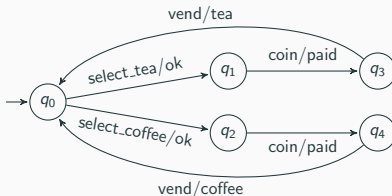
# Transition Tours: Generating a Transition Tour

- One could apply a simple heuristic such as:
  - In the current state  $s$ , if there is a transition  $t$  from  $s$  that has not yet been included then add  $t$  to the current path and update the current state.
  - Otherwise, follow the shortest path from  $s$  that moves  $M$  to a state with at least one uncovered transition.
- Alternatively, we can use graph algorithms (we are looking for a path that includes all edges).
  - These can return an optimal (shortest) transition tour in polynomial time.

- A transition tour may not find state transfer faults.
- To find state transfer faults we need to be able to *check states*.
- In black-box testing, this requires finding appropriate input sequences.
- We might follow each transition by sequences that distinguish between possible states.

# Transition Tour for Drinks Machine

Assume no state transfer faults, then we can test by just executing every transition



The input sequence *select\_tea*, *coin*, *vend*, *select\_coffee*, *coin*, *vend* will do that for us.

We validate that the output sequence is *ok*, *paid*, *tea*, *ok*, *paid*, *coffee*.

# That's Not Good Enough!

- If there are state transfer faults, a transition tour may not find them
- Output faults may also be masked
- We want to explicitly check for state transfer faults

# Identifying States

To test from an FSM, we want to check every transition  $(q_i, q_j, x/y)$

- Get the FSM to state  $q_i$
- Submit input  $x$
- Do we get output  $y$ ?
- **Do we end up in state  $q_j$ ?**

## Challenges

**Controllability:** How do we get the FSM to  $q_i$ ?

**Observability:** How do we know the FSM is in  $q_j$ ?

## Controllability

Find a sequence that gets the *specification* to the desired state.

## Observability

Characterise states in terms of the I/O actions they can perform:

- Distinguishing sequences
- Unique I/O sequences (UIOs)
- Characterising set

# Distinguishing States

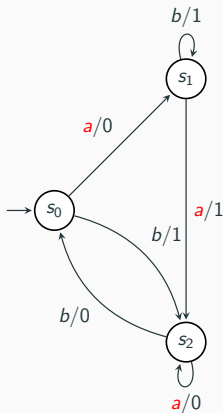
Separating two states:

- Two states  $s$  and  $s'$  of FSM  $M$  are *separated* by input sequence  $x$  if: the response of  $M$  to  $x$  is different in states  $s$  and  $s'$   
 $(\lambda^*(s, x) \neq \lambda^*(s', x))$
- If there is such an input sequence  $x$  then  $s$  and  $s'$  are said to be *separable*.
- States  $s$  and  $s'$  are said to be *equivalent* if they are *not separable*.



# Distinguishing States: Example 1

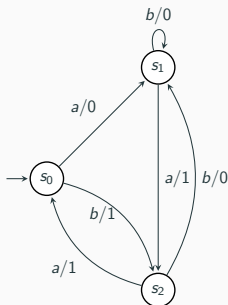
In the following FSM, input  $a$  separates states  $s_0$  and  $s_1$ , but *does not* separate states  $s_0$  and  $s_2$ .



## Distinguishing States: Example 2

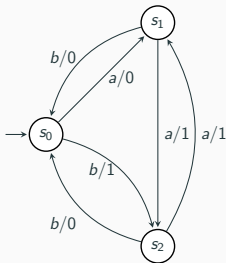
Inputs sequences of length greater than 1 are sometimes needed.

In the following FSM, no input of length 1 can separate states  $s_1$  and  $s_2$ , but  $aa$  does ( $\lambda^*(s_1, aa) = 11$ ,  $\lambda^*(s_2, aa) = 10$ )



## Distinguishing States: Example of Equivalence

In the following FSM, no input can separate states  $s_1$  and  $s_2$ , therefore they are *equivalent*.



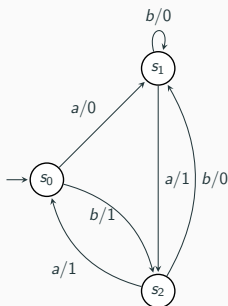
# Distinguishing Sequences

A distinguishing sequence  $D$  is an input sequence that produces a different output **for each state**.

- For every pair of states  $s$  and  $s'$  of FSM  $M$  such that  $s \neq s'$ , we have that  $\lambda^*(s, D) \neq \lambda^*(s', D)$
- This means that the output produced in response to  $D$  **identifies** the state of  $M$ .

# Distinguishing Sequences: Example

- For the example FSM below,  $aa$  forms a Distinguishing Sequence since:
  - From state  $s_0$  the output sequence is 01
  - From state  $s_1$  the output sequence is 11
  - From state  $s_2$  the output sequence is 10

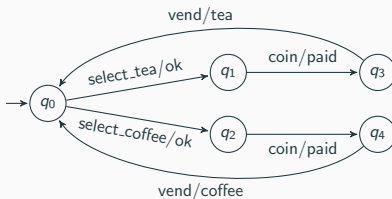


# Distinguishing Sequences for Drinks Machine

Not every FSM has a distinguishing sequence! Does the drinks machine have one?

Yes! (assuming we know when an input has been refused)

$[select\_tea, coin, vend]$



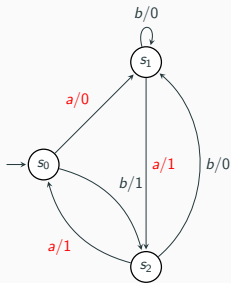
- From  $q_0$  we get *ok, paid, tea*
- From  $q_1$  we get *refuse, paid, tea*
- From  $q_2$  we get *refuse, paid, coffee*
- From  $q_3$  we get *refuse, refuse, tea*
- From  $q_4$  we get *refuse, refuse, coffee*

# Unique I/O Sequences (UIOs)

- A unique I/O sequence *separates* **one state** from **all the other states**.
- To see if an FSM is in state  $s$ , we can execute the sequence for  $s$
- Formally,  $\lambda^*(s, \bar{x}) = \bar{y} \implies (\forall s'. s' \neq s \implies \lambda^*(s', \bar{x}) \neq \bar{y})$ 
  - $\bar{x}$  represents a *sequence* of inputs
  - $\bar{y}$  represents a *sequence* of outputs
- This means that input sequence  $\bar{x}$  identifies state  $s$  since: if  $\bar{y}$  is produced in response to  $\bar{x}$ , then we *must* have been in state  $s$ , otherwise we must have been in a different state.
- Note: Distinguishing Sequences identify all of the states but a UIO might only identify *one* state.
- Not all FSMs have these either!

## Unique I/O Sequences (UIOs): Example

- In the example,  $a/0$  forms a UIO for state  $s_0$ :
  - From state  $s_0$  the output sequence is 0
  - From state  $s_1$  the output sequence is 1
  - From state  $s_2$  the output sequence is 1
- Note that  $a$  is not a Distinguishing Sequence.

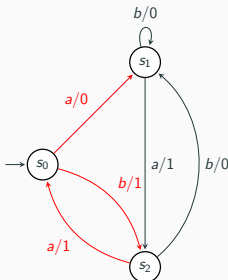




# Unique I/O Sequences (UIOs): Example

Using UIOs to test transitions...

- In order to test the transition  $t = (s_2, s_0, a/1)$  we can build a test sequence as follows:
  - A preamble  $b/1$  that reaches the initial state of  $t$  ( $s_2$ );
  - The input/output pair  $a/1$  of the transition  $t$ .
  - Finally, the chosen UIO for the **end state** of the transition ( $a/0$ ).
- This gives the test sequence  $b/1, a/1, a/0$ .



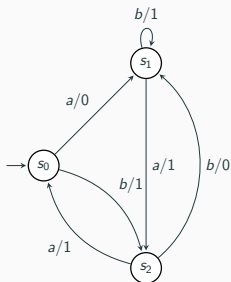
# Characterising Sets

- A *characterising set*  $W$  is a set of input sequences that distinguishes each **pair of states**.
- For every pair of states  $s, s'$  of FSM  $M$  such that  $s \neq s'$ , we have some  $\bar{x} \in W$  such that  $\lambda^*(s, \bar{x}) \neq \lambda^*(s', \bar{x})$  (i.e. the input sequence  $\bar{x}$  distinguishes these states).
- More formally,  $\forall s \neq s' \in S. \exists \bar{x} \in W. \lambda^*(s, \bar{x}) \neq \lambda^*(s', \bar{x})$
- Every minimal FSM with  $n$  states has a characterising set  $W$  with at most  $n - 1$  sequences of length at most  $n - 1$
- There are polynomial time algorithms to generate  $W$  sets
- We can minimise every FSM

# Characterising Sets: Usage

- If we know the output triggered by each input sequence from  $W$ , then we can identify the state.
- To check transition  $t = (s_i, s_j, x/y)$  we can separately follow  $t$  by each element of  $W$ .
- Thus, using a characterising set leads to multiple tests for a transition.

# Characterising Sets: Example



For the above FSM,  $\{a, b\}$  is a characterising set since for every pair of states, there is an element in the set that distinguishes them:

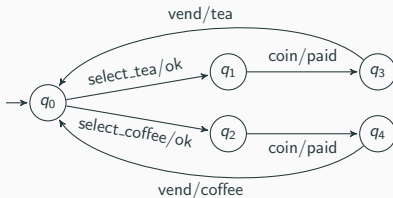
State	Response to $a$	Response to $b$
$s_0$	0	1
$s_1$	1	1
$s_2$	1	0

# Characterising Sets: Implications

- For this example, if we are checking the final state of a transition  $t$  we separately follow it by  $a$  and  $b$ .
- So, we use two tests for a transition.
- If the response to  $a$  after  $t$  is 1 and the response to  $b$  after  $t$  is 0 the transition must have taken the implementation to a state corresponding to  $s_2$ .

# Characterising Set for Drinks Machine

Again, assuming we know when an input has been refused (e.g., a loop in each state that outputs “refused”):  $W = \{[select\_tea, coin, vend]\}$



State	Response to $[select\_tea, coin, vend]$
$q_0$	ok, paid, tea
$q_1$	refused, paid, tea
$q_2$	refused, paid, coffee
$q_3$	refused, refused, tea
$q_4$	refused, refused, coffee

# The $W$ Method

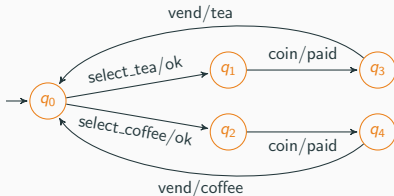
The  $W$  method produces a set of input sequences to test correctness, assuming:

- Implementation behaves like some unknown FSM with no more than  $n$  states.
- The system under test has a reliable reset function.
  - A reset (or reset function / operation) is an input that takes the FSM to the initial state irrespective of the current state.
  - A reliable reset is a reset that is known to take the system under test to its initial state (i.e., it has been correctly implemented)<sup>1</sup>
- $V$  is a *state cover*: A set of input sequences such that each state of the FSM is reached from  $s_0$  by a (unique) sequence from  $V$ .  $V$  must **include the empty input sequence** (i.e., reach the initial state).
- $W$  is a characterising set.

---

<sup>1</sup>Note: Many real systems have a reset and sometimes this simply means switching the system off and then on again, but it may be a more complex operation.

# The $W$ Method for the Drinks Machine



$$W = \{[select\_tea, coin, vend]\}$$

$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

We start with the empty sequence in  $V$ , by definition. Then visit every state from the initial state;  $[select\_tea]$  gets us to  $q_1$ . Next,  $[select\_tea, coin]$  gets us to  $q_3$ . Next,  $[select\_coffee]$  gets us to  $q_2$ . Finally,  $[select\_coffee, coin]$  gets us to  $q_4$ .



# Applying the $W$ Method

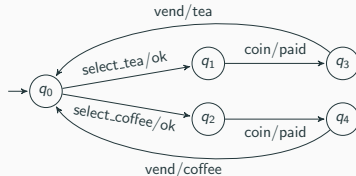
For FSM with  $n + 1$  states, the  $W$  method produces the test set  $VW \cup VXW$  (Remember  $X$  is the input alphabet)

- $VW$  checks that each input sequence  $\bar{x} \in V$  goes to the right state. The set  $VW$  contains each element of the state cover  $V$  followed by each element of the characterising set  $W$ .
- $VXW$  checks that each transition from each state goes to the right state. Formally,  $VXW = \{vxw \mid v \in V, x \in X, w \in W\}$

More generally, for  $n + m$  states, we get  $VW \cup VXW \cup \dots \cup VX^m W$

Make sure you reset before each sequence!

# Applying the $W$ Method to the Drinks Machine



$$V = \{\epsilon, [\text{select\_tea}], [\text{select\_tea, coin}], [\text{select\_coffee}], [\text{select\_coffee, coin}]\}$$

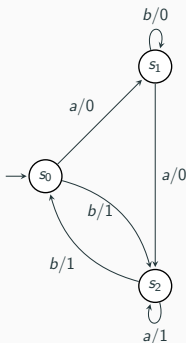
$$X = \{[\text{select\_tea}], [\text{select\_coffee}], [\text{coin}], [\text{vend}]\}$$

$$W = \{[\text{select\_tea, coin, vend}]\}$$

$$VW = \{[\text{select\_tea, coin, vend}], [\text{select\_tea, select\_tea, coin, vend}], \\ [\text{select\_tea, coin, select\_tea, coin, vend}], \dots\}$$

$$VXW = \{[\text{select\_tea, select\_tea, coin, vend}], \\ [\text{select\_tea, select\_tea, select\_tea, coin, vend}], \\ [\text{select\_tea, coin, select\_tea, select\_tea, coin, vend}], \dots\}$$

## A Smaller Example



$$V = \{\epsilon, a, b\}, X = \{a, b\}, \text{ and } W = \{a, b\}$$

$$VW = \{\epsilon, a, b\}\{a, b\} = \{a, b, aa, ab, ba, bb\}$$

$$VXW = \{\epsilon, a, b\}\{a, b\}\{a, b\}$$

$$= \{aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb\}$$

# Things to Think About

- Where do models come from? Have you ever (voluntarily) drawn a model for your software?
  - Model inference!
- What if we can't reliably reset?
  - Use transfer

- Sometimes we do not have access to the source code of a system
- In these cases, we need to apply *black-box* testing techniques
- Model-based testing is one such technique
- We can test that the system matches an FSM specification using the *W* method.
  - The *W* method is guaranteed to find a fault if the implementation is faulty and satisfies the assumption (at most one extra state).