Model Based Testing

Dr Michael Foster
Based on material from Professor Rob Hierons

Outline

- Formal models of systems (FSMs)

Outline

- Formal models of systems (FSMs)
- Testing from finite state machines

Outline

- Formal models of systems (FSMs)
- Testing from finite state machines
- Identifying states

Outline

- Formal models of systems (FSMs)
- Testing from finite state machines
- Identifying states
    - Distinguishing sequences

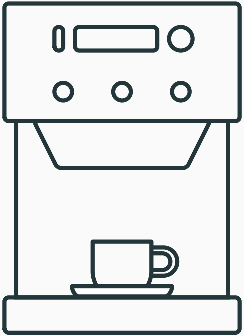
Outline

- Formal models of systems (FSMs)
- Testing from finite state machines
- Identifying states
    - Distinguishing sequences
    - Unique I/O sequences

Outline

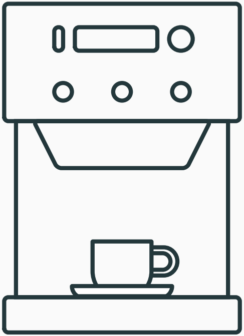
- Formal models of systems (FSMs)
- Testing from finite state machines
- Identifying states
    - Distinguishing sequences
    - Unique I/O sequences
    - The W method

Motivating Example - Simple Drinks Machine



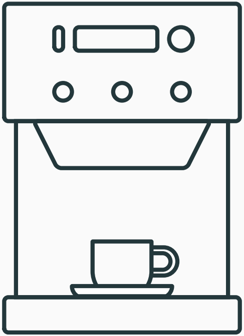
- Select a drink

Motivating Example - Simple Drinks Machine



- Select a drink
- Insert coins

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

How do we test this system?

Unit testing

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

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?

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

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

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
- Aim to achieve some level of model coverage

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 six tuple $(S, s_0, X, Y, \delta, \lambda)$ where

- S is a finite set of states

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 six tuple $(S, s_0, X, Y, \delta, \lambda)$ where

- S is a finite set of states
- $s_0 \in S$ is the initial state

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 six tuple $(S, s_0, X, Y, \delta, \lambda)$ where

- S is a finite set of states
- $s_0 \in S$ is the initial state
- X is the input alphabet

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 six tuple $(S, s_0, X, Y, \delta, \lambda)$ where

- S is a finite set of states
- $s_0 \in S$ is the initial state
- X is the input alphabet
- $\delta : (S, X) \to S$ is the state transition function

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 six tuple $(S, s_0, X, Y, \delta, \lambda)$ 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

Just like a DFA

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 six tuple $(S, s_0, X, Y, \delta, \lambda)$ 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

Just like a DFA

- Y is the output alphabet

Formal Models of Systems

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

We will introduce MBT with state machines

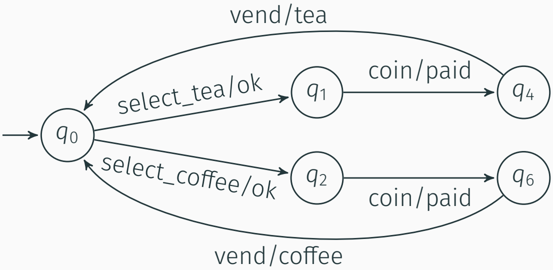
An FSM (Mealy machine) is a six tuple $(S, s_0, X, Y, \delta, \lambda)$ where

- 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

Just like a DFA

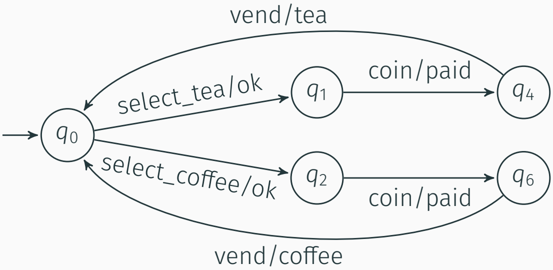
- Y is the output alphabet
- $\lambda : (S, X) \rightarrow Y$ is the output function

FSM of the Drinks Machine



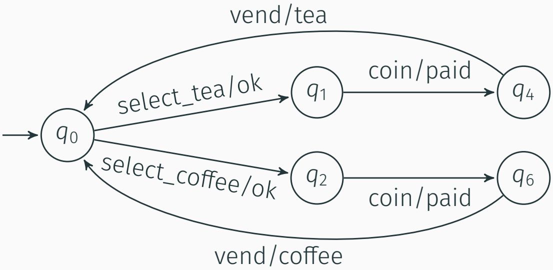
- We give the system inputs and it changes state and gives us outputs

FSM of the Drinks Machine



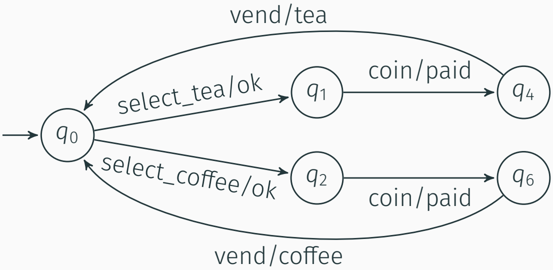
- We give the system inputs and it changes state and gives us outputs
- Transition x/y represents giving input x and getting output y, e.g.

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*

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 $q_0 \rightarrow q_1 \rightarrow q_4 \rightarrow q_0$

Terminology (sorry!)

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 δ and λ are functions)

Terminology (sorry!)

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 δ and λ are functions)
- An FSM is *completely specified* if there is at **least** one transition from each state for each input

Terminology (sorry!)

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 δ and λ 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”

Terminology (sorry!)

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

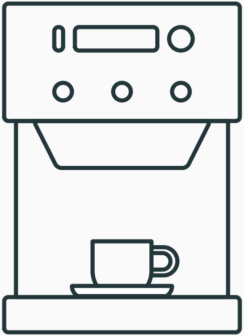
- An FSM is *deterministic* if there is at **most** one transition from each state for each input (implicit since δ and λ are functions)
- An FSM is *completely specified* if there is at **least** one transition from each state for each input
    - We can make any FSM complete with an “error state” or “null actions”
- An FSM is *minimal* if there is no trace equivalent FSM with fewer states

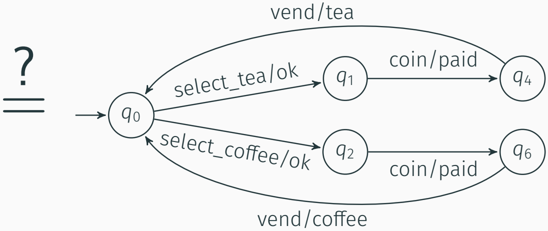
Terminology (sorry!)

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 δ and λ 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
    - We can minimise any FSM by following the algorithm

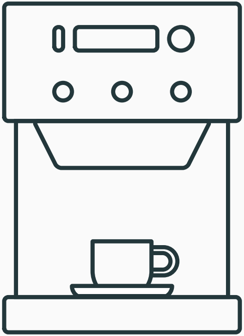
Testing from an FSM

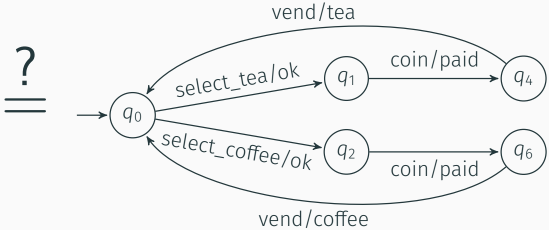




- Assume the software behaves like an FSM model

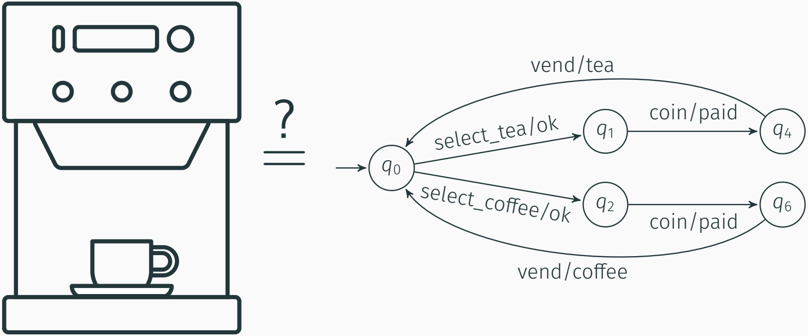
Testing from an FSM





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

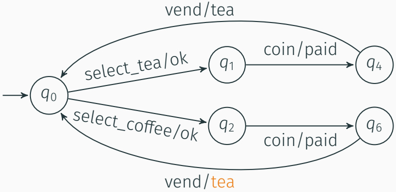
Testing from an FSM



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

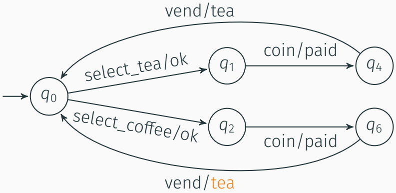
Faults

Output faults: A transition produces the wrong output

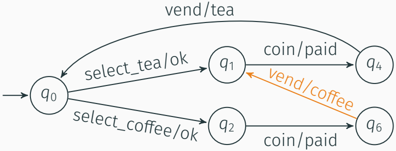


Faults

Output faults: A transition produces the wrong output

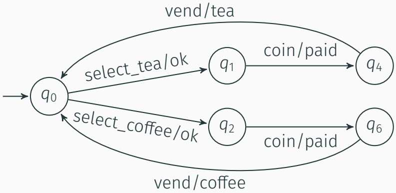


State transfer faults: A transition goes to the wrong state



Transition Tour

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

That's Not Good Enough!

- If there are state transfer faults, a transition tour may not find them
- Output faults may also be masked

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

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

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?

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?**

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

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?

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?

Solutions

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:

Solutions

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

Solutions

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

Solutions

Controllability

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

Observability

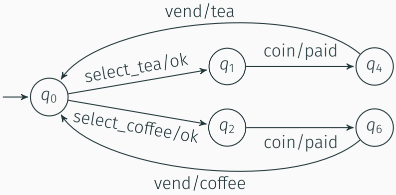
Characterise states in terms of the I/O actions they can perform:

- Distinguishing sequences
- Unique I/O sequences
- Characterising set

Distinguishing Sequences

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

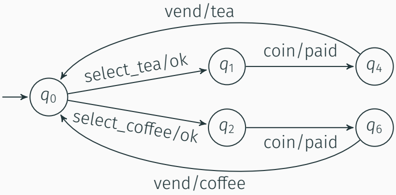
Not every FSM has a distinguishing sequence! Does the drinks machine have a distinguishing sequence?



Distinguishing Sequences

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

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



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

select_tea, coin, vend

Unique I/O Sequences

- A unique I/O sequence separates *one state* from all the other states.

Unique I/O Sequences

- 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

Unique I/O Sequences

- 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, $\lambda^*(s, \bar{x}) = \bar{y} \implies (\forall s'.\, s' \neq s \implies \lambda^*(s, \bar{x}) \neq \bar{y})$

Unique I/O Sequences

- 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

Unique I/O Sequences

- 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

Unique I/O Sequences

- 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
- Not all FSMs have these either!

Characterising Sets

- A *W* set is a *set* of input sequences that distinguishes each pair of states

Characterising Sets

- A *W* set is a *set* of input sequences that distinguishes each pair of states
- Formally, $\forall s \neq s' \in S.\, \exists \overline{x} \in W.\, \lambda^*(s, \overline{x}) \neq \lambda^*(s', \overline{x})$

Characterising Sets

- A *W* set is a *set* of input sequences that distinguishes each pair of states
- Formally, $\forall s \neq s' \in S.\, \exists \overline{x} \in W.\, \lambda^*(s, \overline{x}) \neq \lambda^*(s', \overline{x})$
- Every minimal FSM with n states has a W set with at most $n-1$ sequences of length at most $n-1$

Characterising Sets

- A W set is a *set* of input sequences that distinguishes each pair of states
- Formally, $\forall s \neq s' \in S.\, \exists \overline{x} \in W.\, \lambda^*(s, \overline{x}) \neq \lambda^*(s', \overline{x})$
- Every minimal FSM with n states has a W set with at most $n - 1$ sequences of length at most $n - 1$
- There are polynomial time algorithms to generate W sets

Characterising Sets

- A *W* set is a *set* of input sequences that distinguishes each pair of states
- Formally, $\forall s \neq s' \in S.\, \exists \overline{x} \in W.\, \lambda^*(s, \overline{x}) \neq \lambda^*(s', \overline{x})$
- Every minimal FSM with n states has a W set 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

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 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

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
- W is a characterising set

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
- W is a characterising set
- V is a *state cover*

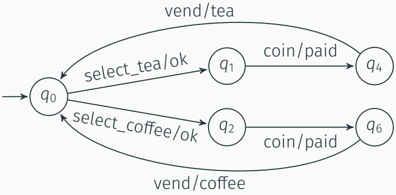
The *W* Method

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
- *W* is a characterising set
- *V* is a *state cover*

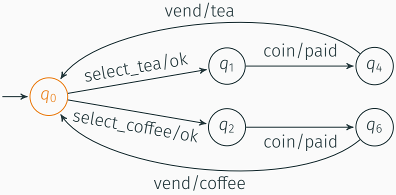
A *state cover V* is a set of input sequences such that each state of an FSM *M* is reached from s_0 by a sequence from *V*, and *V* also contains the empty input sequence.

The W Method for Drinks



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

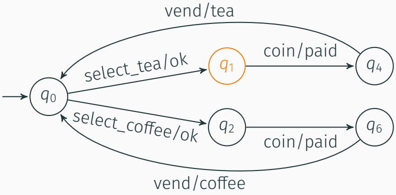
The W Method for Drinks



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

$$V = \{\epsilon\}$$

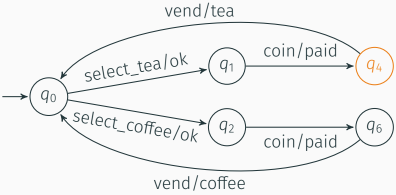
The W Method for Drinks



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

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

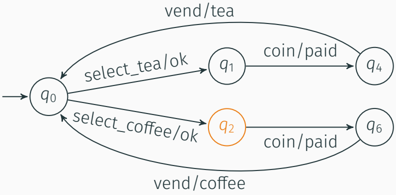
The W Method for Drinks



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

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

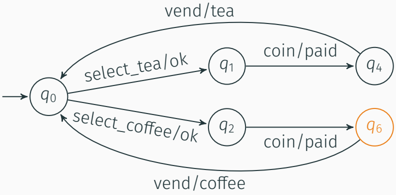
The W Method for Drinks



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

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

The W Method for Drinks



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

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

Applying the W Method

For FSM with $n + 1$ states, the W method produces the test set $VW \cup VXW$

- VW checks each $\bar{x}$ in V goes to the right state

Applying the W Method

For FSM with $n+1$ states, the W method produces the test set $VW \cup VXW$

- VW checks each $\bar{x}$ in V goes to the right state
- VXW checks that each transition from each state goes to the right state

Applying the W Method

For FSM with $n + 1$ states, the W method produces the test set $VW \cup VXW$

- VW checks each $\bar{x}$ in V goes to the right state
- VXW checks that each transition from each state goes to the right state

For $n + m$ states, we get $VW \cup VXW \cup \ldots \cup VX^m W$

Applying the W Method

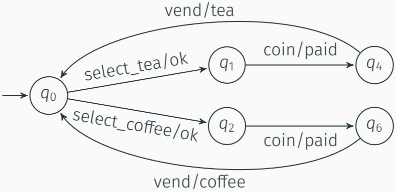
For FSM with $n+1$ states, the W method produces the test set $VW \cup VXW$

- VW checks each $\bar{x}$ in V goes to the right state
- VXW checks that each transition from each state goes to the right state

For $n+m$ states, we get $VW \cup VXW \cup \ldots \cup VX^m W$

Make sure you reset before each sequence!

Applying the W Method to Drinks

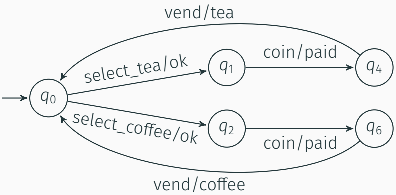


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

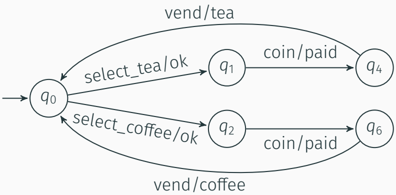


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

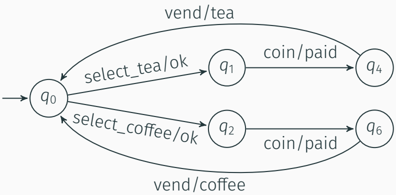


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

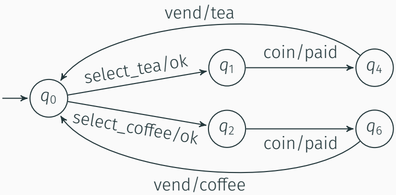


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

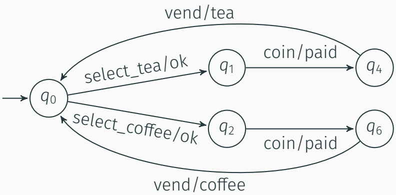


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

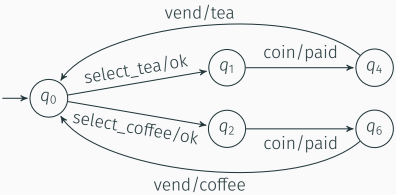


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

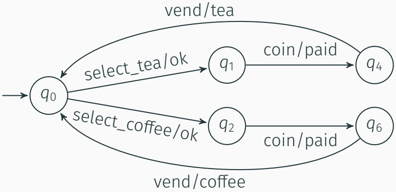


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

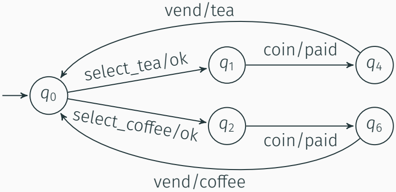


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

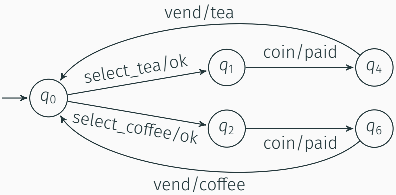


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

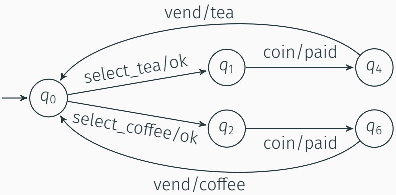


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

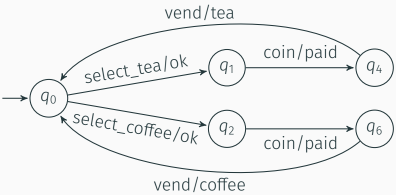


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

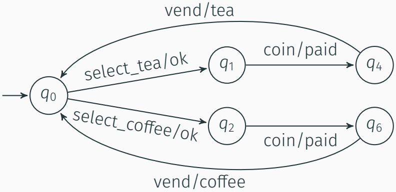


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

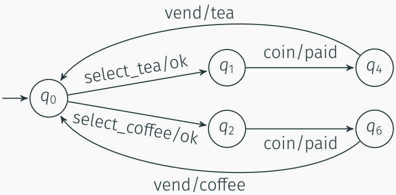


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

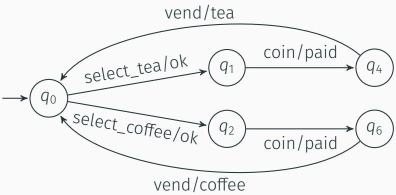


$$V = \{\epsilon, [select\_tea], [select\_tea,\ coin], [select\_coffee], [select\_coffee,\ coin]\}$$

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

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

Applying the W Method to Drinks

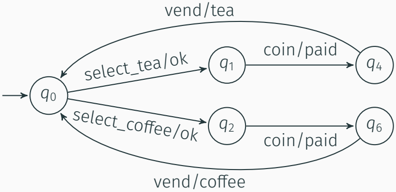


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

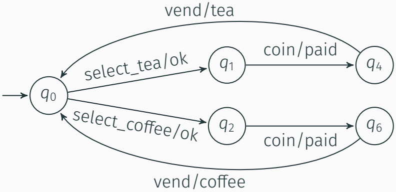


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

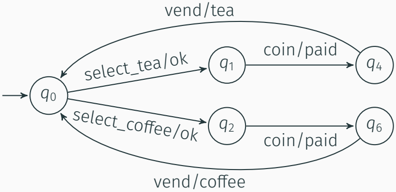


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

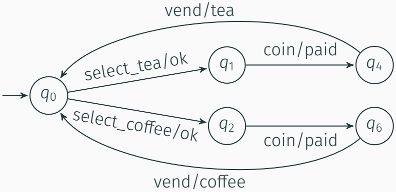


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

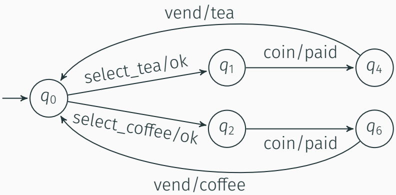


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

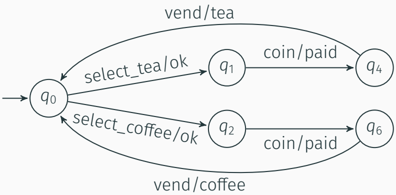


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

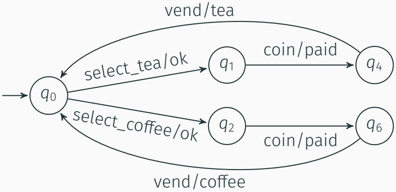


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

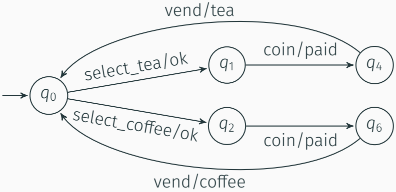


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

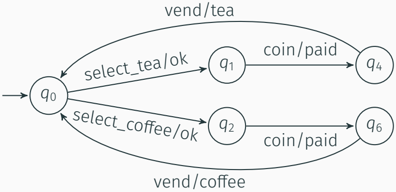


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks

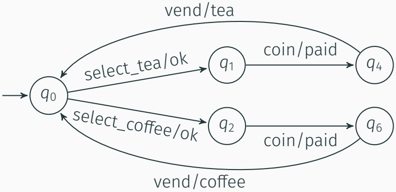


$$V = \{\epsilon, [select\_tea], [select\_tea, coin], [select\_coffee], [select\_coffee, coin]\}$$

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

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

Applying the W Method to Drinks



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

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

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

Things to Think About

- Where do models come from? Have you ever (voluntarily) drawn a model for your software?

Things to Think About

- Where do models come from? Have you ever (voluntarily) drawn a model for your software?
  - Model inference!

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?

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

Summary

- Sometimes we do not have access to the source code of a system

Summary

- Sometimes we do not have access to the source code of a system
- In these cases, we need to apply *black-box* testing techniques

Summary

- 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

Summary

- 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