Black Box Testing Techniques (part 2)

Dr. John H Robb
UTA Computer Science and Engineering

(c) JRCS 2016

Black Box Testing Techniques

- Typical “black-box” test analysis and design techniques include:
  - Equivalence partitioning
  - Boundary value analysis
  - Decision table testing
  - State transition analysis
  - Sequence enumeration
  - Use Case testing
  - Decision logic and Karnaugh maps

- Black box is a bit of a misnomer – we don’t and can’t test strictly black box

- So these are correctly called Specification based test analysis and design techniques, but I use the term “black box” because it is so commonly used

Finite State Machines

- Why do we care about testing finite state machines?
  - Abstraction: designs can often be best understood as finite-state machines
    - String processing/searching
    - Protocols – communication, cache coherence, etc.
    - Control component of any discrete system
    - Embedded Systems/Real-time Simulations
  - Automatic abstraction:
    - Tools that take systems and produce (coarse) finite state abstractions
    - These can be modeled as FSMs
  - UML uses FSMs as a specification technique

State Machines: Definition of Terms

From IEEE 24765 except as noted

- Finite State Machine (FSM)
  - A computational model consisting of a finite number of states and transitions between those states, possibly with accompanying actions.
- State Diagram
  - A diagram that depicts the states that a component or system can assume, and shows the events or circumstances that cause and/or result from a change from one state to another.
- State
  - A condition or mode of existence that a system, component, or simulation may be in.
- State Transition
  - A transition between two states of a component or system (ISTQB)
- State Table
  - A grid showing the resulting transitions for each state combined with each possible event, showing both valid and invalid transitions. (ISTQB)

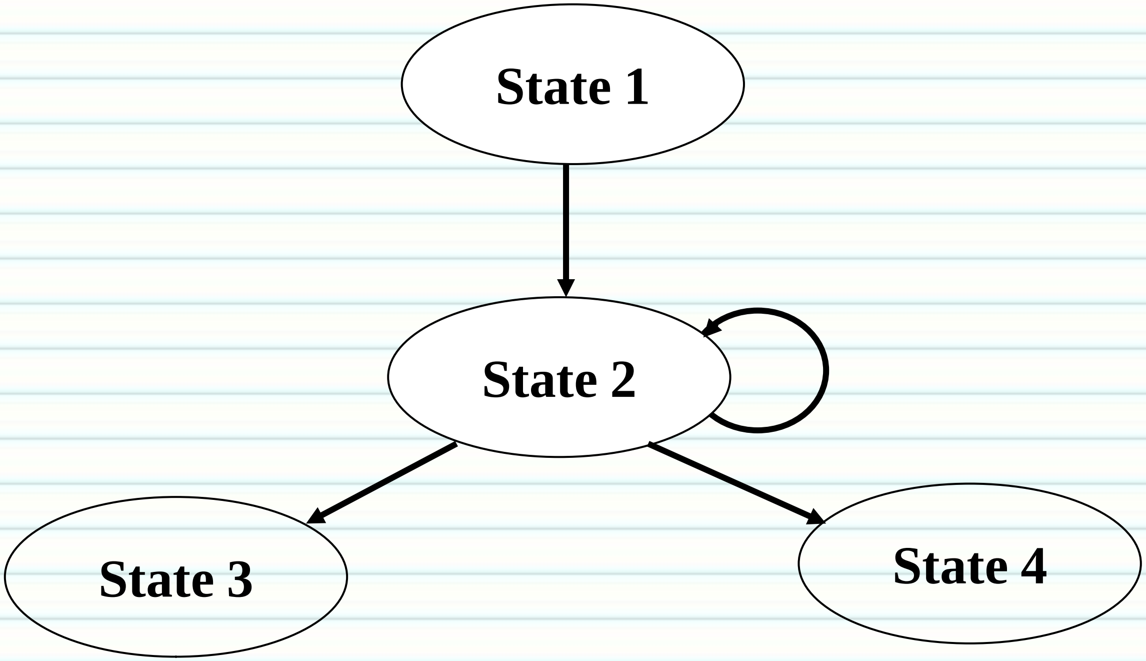
State Machines: Definition of Terms

From IEEE 24765 except as noted

- State Transition Testing
  - –A black box test design technique in which test cases are designed to execute valid and invalid state transitions. See also N-switch testing
- N-switch testing
  - –A form of state transition testing in which test cases are designed to execute all valid sequences of N+1 transitions. [Chow]

Present State and Next State

- On a well-drawn state diagram, all possible transitions will be visible, including loops back to the same state. From this diagram it can be deduced that if the present state is State 2, then the previous state was either State 1 or 2 and the next state must be either 2, 3, or 4.



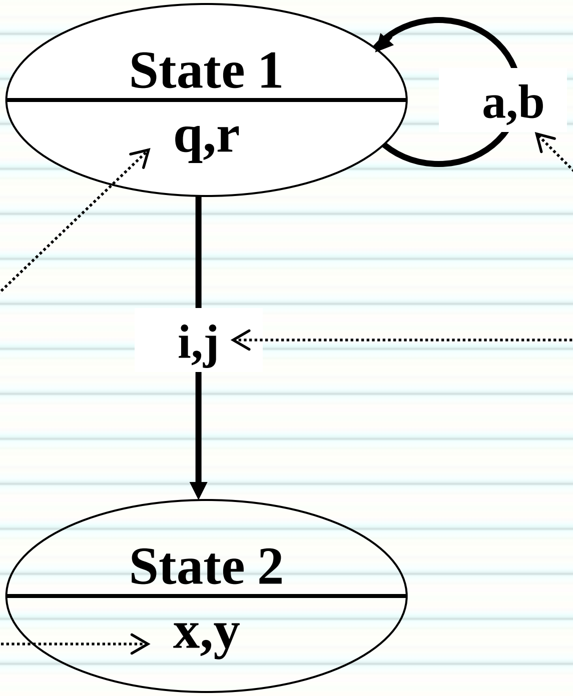
(c) JRCS 2016

Moore and Mealy Machines

- There are two types of graphical depictions of state machines, they differ in the way that outputs are produced.
- **Moore Machine**:
  - Outputs are independent of the inputs-outputs are effectively produced from within the state of the state machine.
- **Mealy Machine**:
  - Outputs can be determined by the present state alone, or by the present state and the present inputs-outputs are produced as the machine makes a transition from one state to another.

Moore Machine Diagrams

The Moore State Machine output is shown inside the state bubble, because the output remains the same as long as the state machine remains in that state.
The output can be arbitrarily complex but must be the same every time the machine enters that state.



Input condition that must exist in order to execute these transitions from State 1

Output condition that results from being in a particular present state

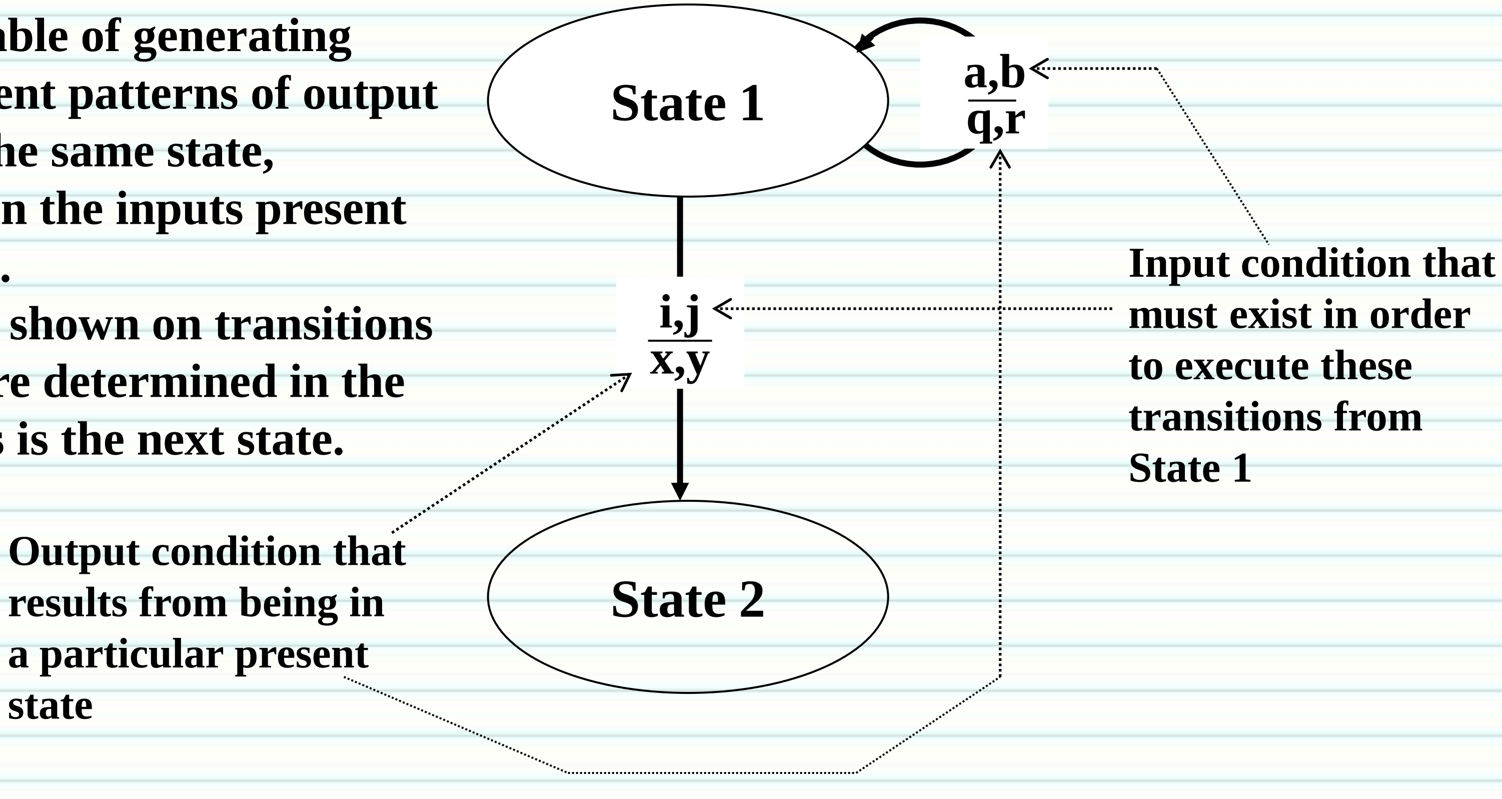
Mealy Machine Diagrams

The Mealy State Machine generates outputs based on:

- The Present State, and
- The Inputs to the M/c.

So, it is capable of generating many different patterns of output signals for the same state, depending on the inputs present on the event.

Outputs are shown on transitions since they are determined in the same way as is the next state.



Input condition that must exist in order to execute these transitions from State 1

Output condition that results from being in a particular present state

(c) JRCS 2016

Use in Software

- For Software Engineering State Machines are primarily used in Specification of Requirements, e.g., UML
- UML state machines have offered some flexibility in an attempt to overcome the limitations of traditional finite state machines while retaining most of their benefits.
- UML state machines (statechart) allow for hierarchically nested states and orthogonal regions. Nested states can at times be difficult to navigate especially when they are extended (many levels and scope).
- They also extend the notion of actions. UML state machines have the characteristics of both Mealy and Moore machines.
- They support actions that depend on both the state of the system and the triggering event (Mealy) and also entry and exit actions, which are associated with states rather than transitions (Moore).

(c) JRCS 2016

UML Statechart

- Like state transition diagrams, UML state diagrams are directed graphs - nodes denote states and connectors transitions
- **Event** – is a type not a specific occurrence
  - Token is an event for the turnstile, but each specific token is an instance of the Token event. For the Push event, if the Push event occurred at a specific time it would be an instance of the Push event.

  An event instance outlives the occurrence that caused it. A specific event instance can goes through three stages. The event instance is

  1. **Received** (on the event queue).
  2. **Dispatched** to the state machine, at which point it becomes the current event.
  3. **Consumed** when the state machine finishes processing the event instance.
- A **state** is much like the previous but captures the specific aspects of system behavior in a more focused manner (a single state variable)

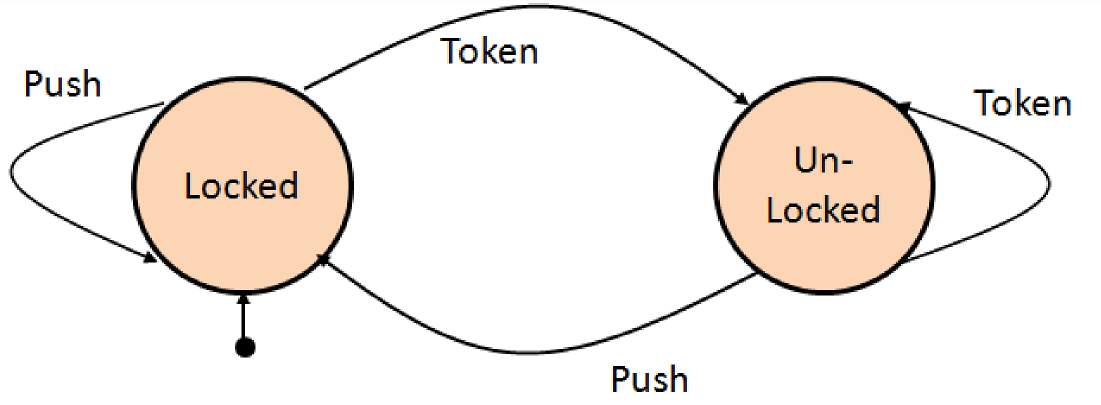
UML Statechart (cont.)

- **Guards -** Boolean expressions evaluated dynamically based on the value of event parameters. They affect the behavior only when they evaluate to TRUE and disable them when evaluated to FALSE.
- **Actions** – the dispatching of an event causes the state machine to performing actions.
- **Transitions** - switching from one state to another. The event that caused the transition is called the trigger.
- **Run to completion** - Incoming events cannot interrupt the processing of the current event and must be stored until the state machine can process them. This approach avoids any concurrency issues within a single state machine.

- In practice there are a wide variety of state machines used.
- So, these are important concepts to know. However, in this course we will concentrate on the use of simple state transition diagrams.

Simple System

- Turnstile system
- A turnstile is used to control access to subways, is a gate with rotating arms (typically 3) near waist height, one across the entryway.
- Initially the arms are locked, preventing entry.
- Depositing a token in a slot on the turnstile unlocks the arms, allowing a single customer to enter. After the customer enters by rotating through the arms, the arms are locked again until another token is inserted.



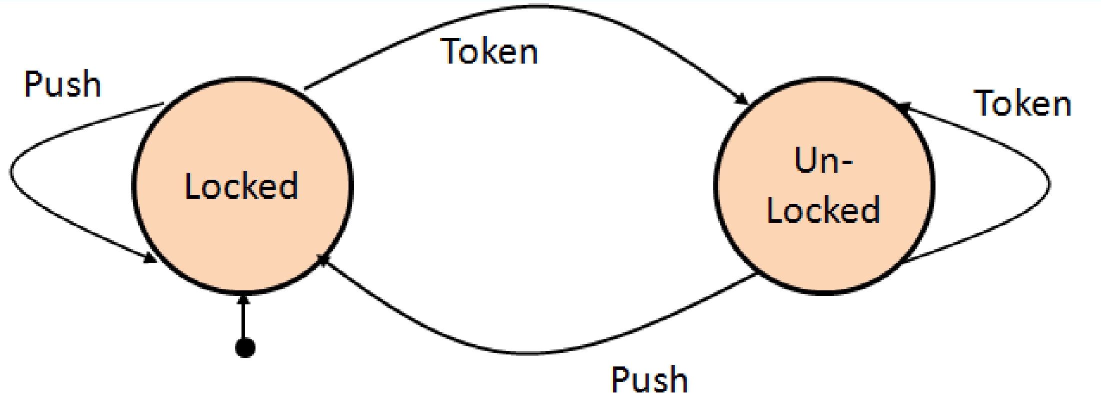
A state machine must have a start state!

(c) JRCS 2016

Simple System

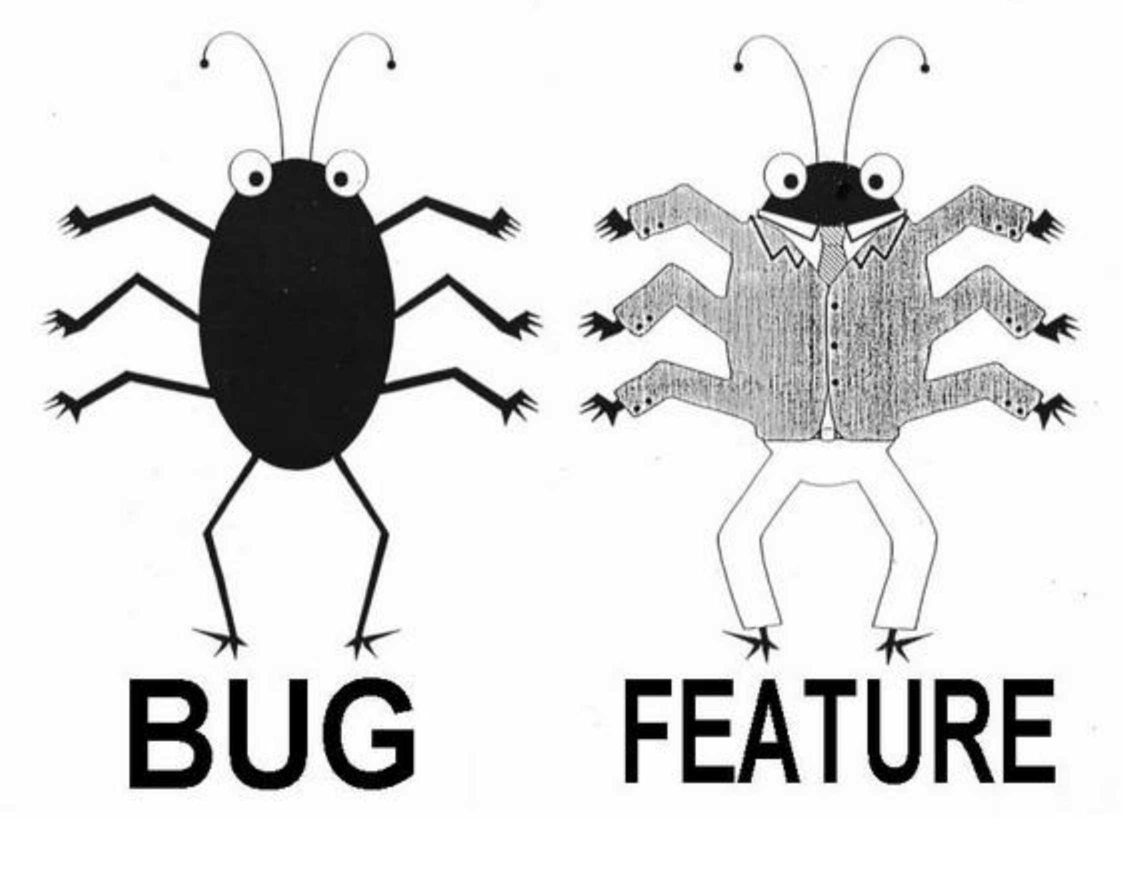
- A state table can be used to represent the state machine as well

Current State	Input	Next State	Output
Locked (start)	token	Unlocked	Unlock command
	push	Locked	None
Unlocked	token	Unlocked	None
	push	Locked	Lock command



A state machine must have a start state!

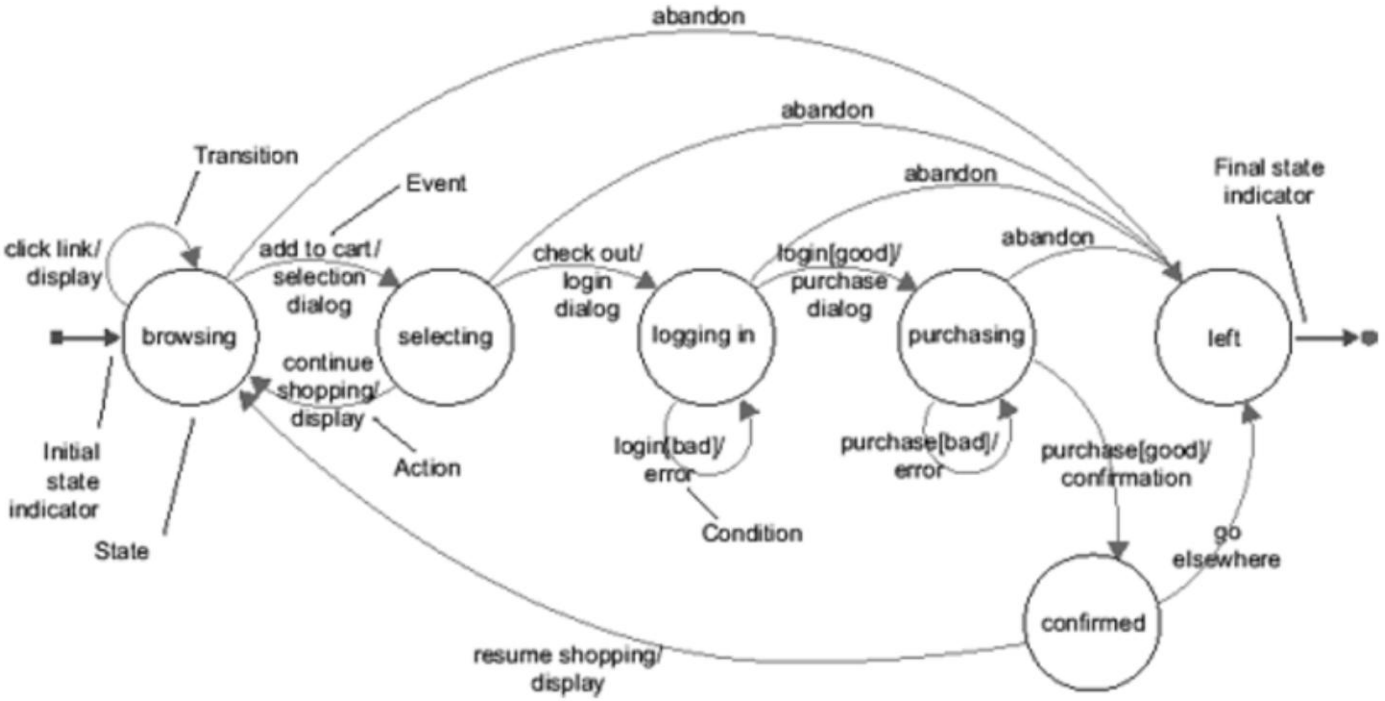
What is the Difference Between a…



What is State Transition Testing?

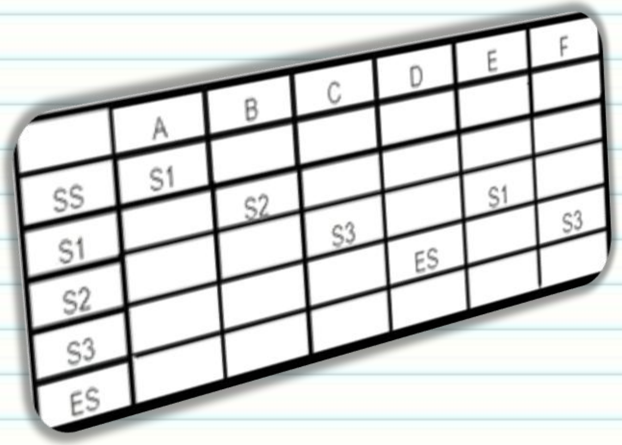
- State transition testing was defined as a black-box test design technique in which test cases are designed to execute valid and invalid state transitions
- When can we use State-based testing?
  - When we have sequences of events that occur and conditions that apply to those events
  - When the proper handling of a particular event/condition situation depends on the events and conditions that have occurred in the past
- What is the term “bug hypothesis” in state-based testing? (ISTQB)
  - We're looking for situations where the wrong action or the wrong new state occurs in response to a particular event under a given set of conditions based on the history of event/condition combinations so far

The Web Browser

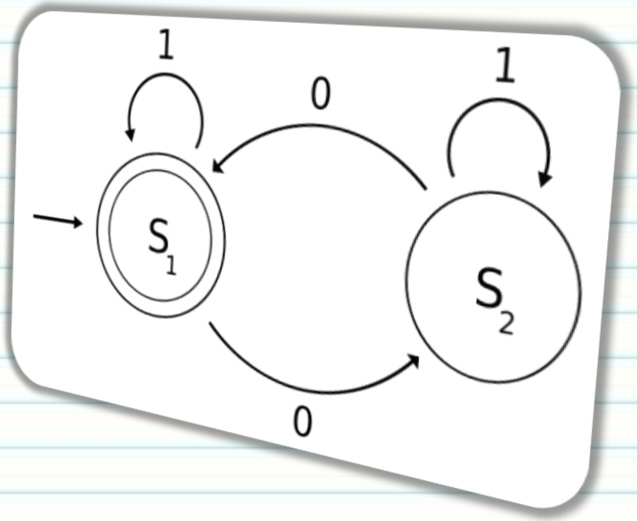


Transition Diagrams and Tables

- The underlying model for State-based testing is a state transition diagram or table
  - The diagram or table connects beginning states, events, and conditions with resulting states and actions



	A	B	C	D	E	F
SS	S1					
S1		S2				
S2			S3		S1	
S3				ES		S3
ES						



States / Events / Actions

- We will use these definitions for our testing class
  - State
    - Persists until something external happens, usually triggering a transition
    - A state can persist for an indefinite period
  - Event
    - Occurs, either instantly or in a limited, finite period
    - It is the something that happens
    - The external occurrence that triggers the transition
  - Action
    - The response of the system during the transition
    - An action, like an event, is either instantaneous or requires a limited, finite period

Coverage Criterion

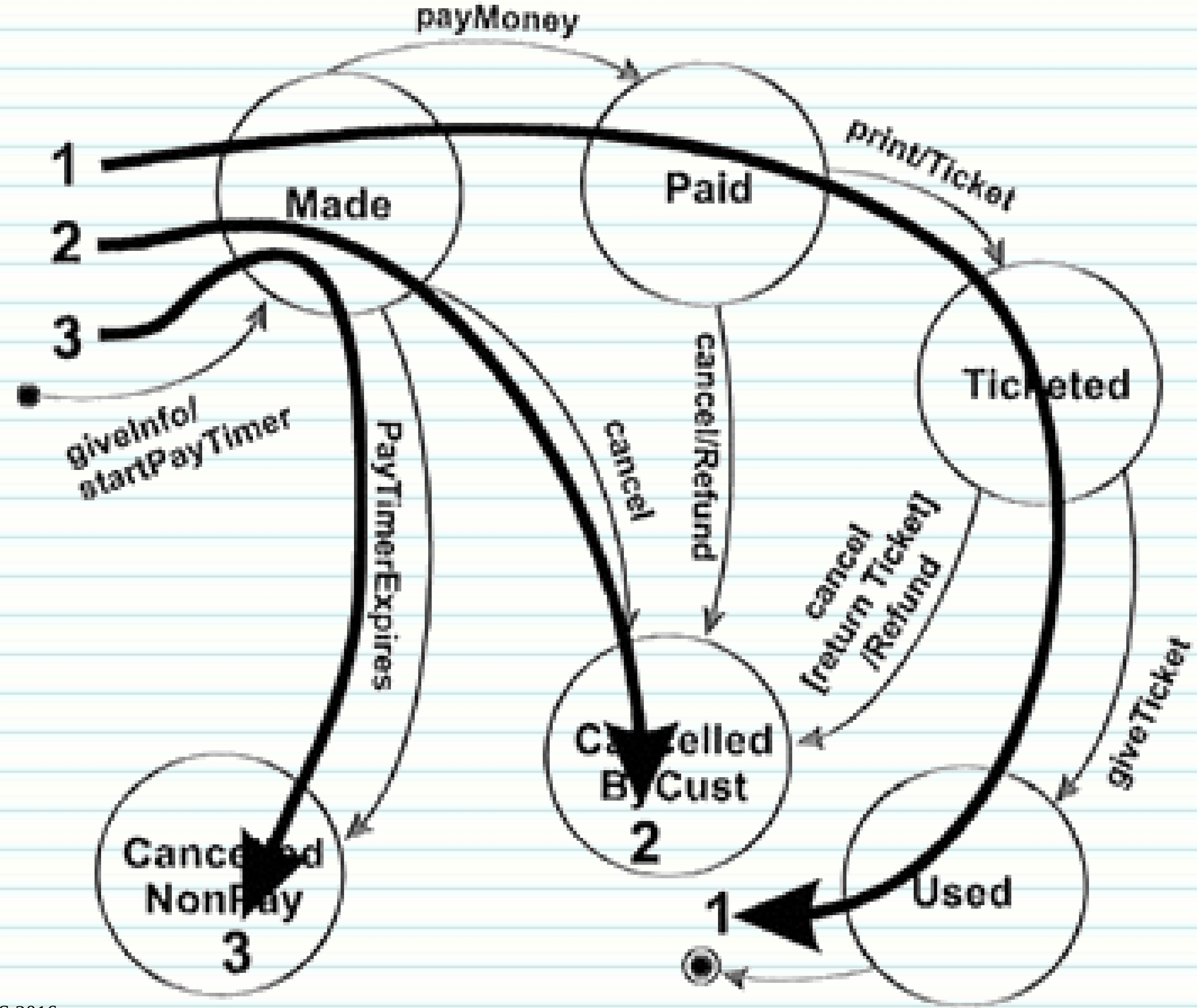
- Various coverage criteria apply for state-based testing:

a. A set of tests that trigger all unique transitions at least once
   - The weakest criterion (see next slide)

b. Visiting every state and traverse every transition (not just unique)
   - A weak criterion (see subsequent slide)

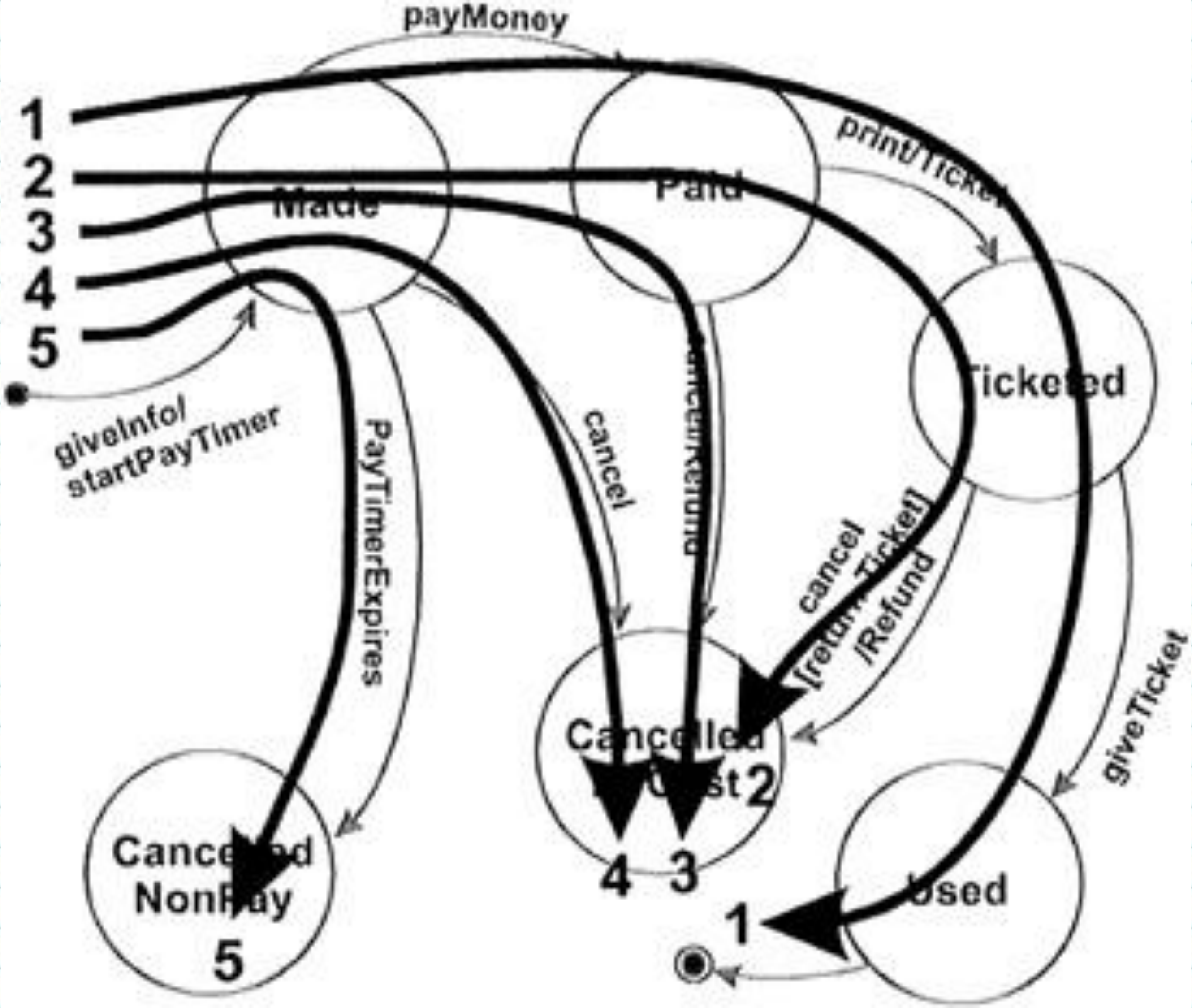
c. At least one test covers every row in a state transition table
   - High coverage criterion - achieves "every state and transition" coverage and covers **combinations not presented in diagrams**
   - **This kind of coverage is required for safety or security critical systems! It can be very expensive**

A set of test cases that trigger all transitions at least once



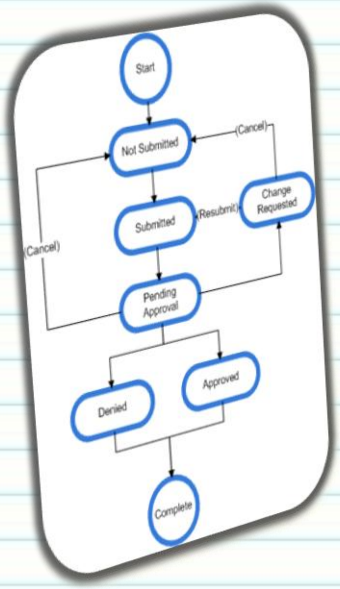
(c) JRCS 2016

A set of test cases that trigger all transitions and states at least once



Deriving Test Cases

- Procedure for Deriving State-based Tests



Procedure for Deriving Tests From State Diagrams

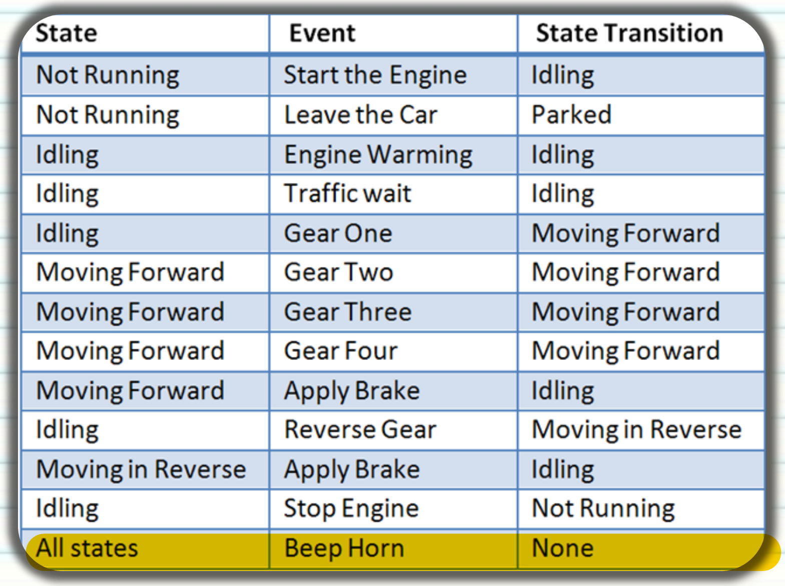
- State-diagram based testing provides a formal procedure for deriving tests

1. Setting a rule for where a test procedure or test step must start and where it may or must end
   - E.g., a test step may start in an initial state and may only end in a final state (which can be the same state)
   - Sequences of states and transitions that pass through the initial state more than once can be allowed
2. Defining a sequence of event/condition combinations that leads to an allowed test ending state
   - For each transition that will occur, the expected action that the system should take is captured (the expected result)
3. Each visited state and traversed transition should be marked as covered (use a marker)
4. Steps 2 and 3 should be repeated until all states have been visited and all transitions traversed
   - I.e. every node and arrow has been marked with the marker

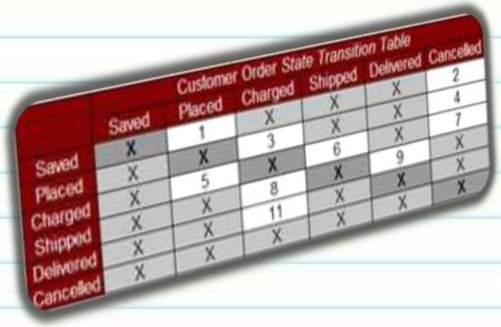
Creating the Test Inputs and Outputs

- The procedure presented before will generate logical test cases
  - For specific test cases to be created, input values and the expected output values have to be generated
- When deriving case-based tests a check of coverage completeness achieved has to be done
  - Generating tests is not completed until every state and every transition has been highlighted

State Transition Tables

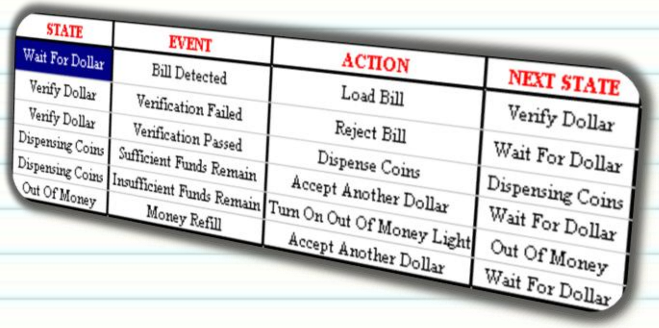


State	Event	State Transition
Not Running	Start the Engine	Idling
Not Running	Leave the Car	Parked
Idling	Engine Warming	Idling
Idling	Traffic wait	Idling
Idling	Gear One	Moving Forward
Moving Forward	Gear Two	Moving Forward
Moving Forward	Gear Three	Moving Forward
Moving Forward	Gear Four	Moving Forward
Moving Forward	Apply Brake	Idling
Idling	Reverse Gear	Moving in Reverse
Moving in Reverse	Apply Brake	Idling
Idling	Stop Engine	Not Running
All states	Beep Horn	None



Customer Order State Transition Table

	Saved	Placed	Charged	Shipped	Delivered	Cancelled
Saved	X	1	X	X	X	2
Placed	X	X	3	X	X	4
Charged	X	5	X	6	X	7
Shipped	X	X	8	X	9	X
Delivered	X	X	11	X	X	X
Cancelled	X	X	X	X	X	X

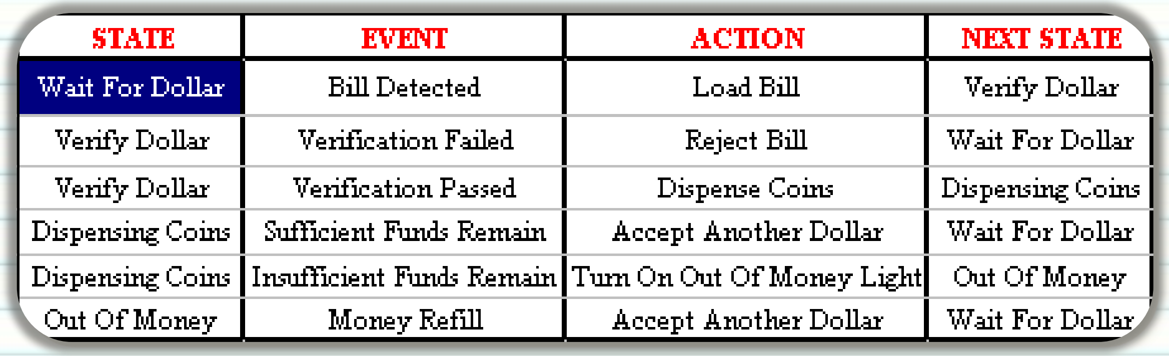


STATE	EVENT	ACTION	NEXT STATE
Wait For Dollar	Bill Detected	Load Bill	Verify Dollar
Verify Dollar	Verification Failed	Reject Bill	Wait For Dollar
Verify Dollar	Verification Passed	Dispense Coins	Dispensing Coins
Dispensing Coins	Sufficient Funds Remain	Accept Another Dollar	Wait For Dollar
Dispensing Coins	Insufficient Funds Remain	Turn On Out Of Money Light	Out Of Money
Out Of Money	Money Refill	Accept Another Dollar	Wait For Dollar

(c) JRCS 2016

Constructing State Transition Tables

- Constructing state transition tables follows the scheme:
  - List all the states from the state transition diagram
  - List all the event/condition combinations shown on the state transition diagram
  - Create a table that has a row for each state with every event/condition combination
- Each row in a state transition table has four fields:
  - Current state
  - Event/condition
  - Action
  - Next state



STATE	EVENT	ACTION	NEXT STATE
Wait For Dollar	Bill Detected	Load Bill	Verify Dollar
Verify Dollar	Verification Failed	Reject Bill	Wait For Dollar
Verify Dollar	Verification Passed	Dispense Coins	Dispensing Coins
Dispensing Coins	Sufficient Funds Remain	Accept Another Dollar	Wait For Dollar
Dispensing Coins	Insufficient Funds Remain	Turn On Out Of Money Light	Out Of Money
Out Of Money	Money Refill	Accept Another Dollar	Wait For Dollar

Why State Transition Tables?

- Why State Transition Tables?
  - They force us to consider combinations of states with event/condition combinations that we might have forgotten
- Deriving state transition tables can reveal undefined situations
  - Forgotten by the business analysts
  - Considered to be impossible
    - The test analyst has the task to find the way a barely possible situation may occur

Deriving Table-based Tests

- Deriving tests covering a state transition table can be based on the following steps:

1. Start with a set of tests derived from a state transition diagram
   - Including the starting and stopping state rule
2. Construct the state transition table and confirm that the tests cover all the defined rows
   - If they do not, then there is a problem with the existing set of tests, the table generated or the state transition diagram
3. Select a test that visits a state for which one or more undefined rows exists in the table
   - Modify that test to attempt to introduce the undefined event/condition combination for that state
   - Notice that the action in this case is undefined
4. Mark covered rows (e.g., a marker)
5. Repeat steps 3 and 4 until all rows have been covered

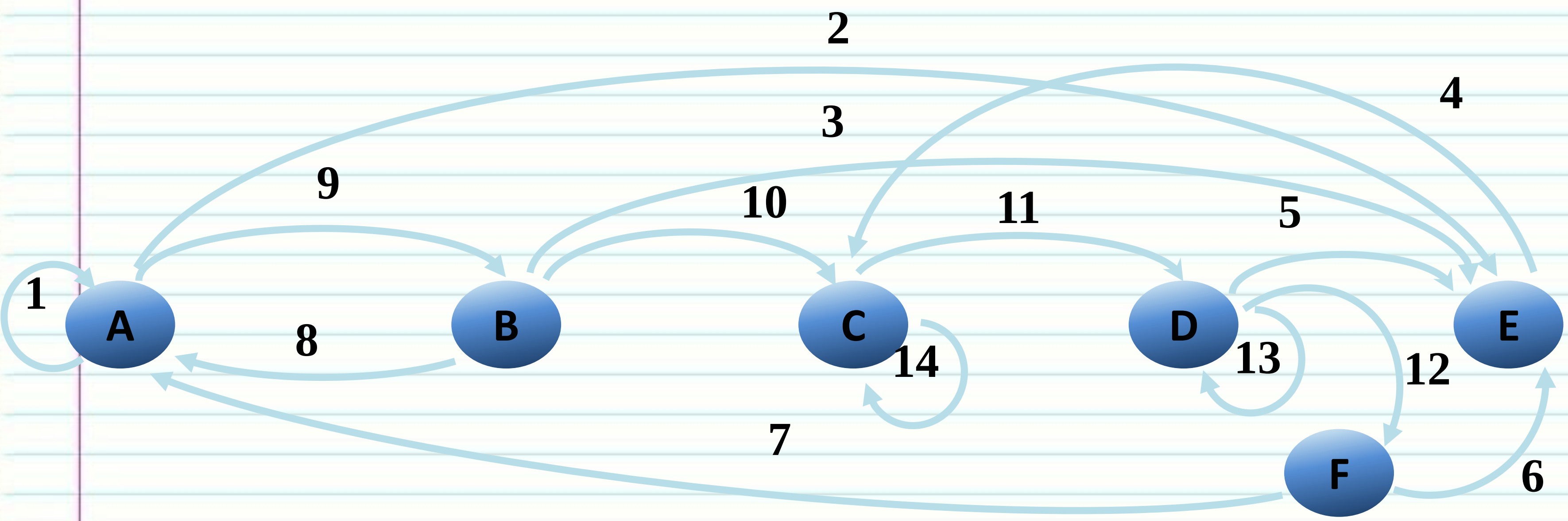
One Undefined Combination per Step

- Each test step should include a single undefined event/condition combination
  - Two undefined actions should not be combined in a single test step
  - We can't be sure that the system will remain testable after the first invalid
- What is the ideal system behavior under undefined conditions?
  - Undefined event/condition combination should be ignored or rejected with an intelligent error message
  - Processing continues normally from that point

What is Switch Coverage?

- Switch Coverage is a technique for generating sequences of transitions
  - State labels are replaced in the diagram with letters and the transition labels with numbers
  - A state/transition pair can be specified in a table as a letter followed by a number

Switch Coverage Example

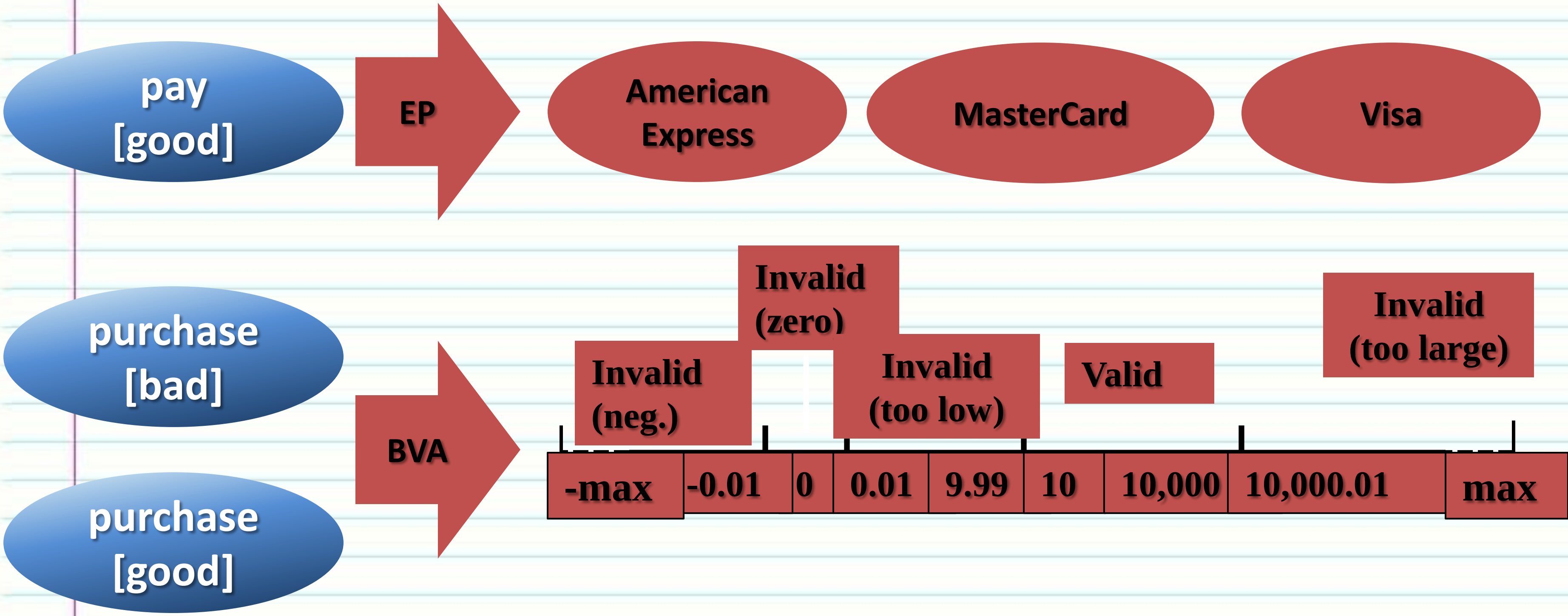


0-switch			1-switch								
A1	A2	A9	A1A1	A1A2	A1A9						

Coverage of all individual transitions is also known as 0-switch coverage, coverage of transition pairs is l-switch coverage, coverage of transition triples is 2-switch coverage, etc.

State Testing with Other Techniques

- State-based testing can be well combined with equivalence partitioning and boundary value analysis

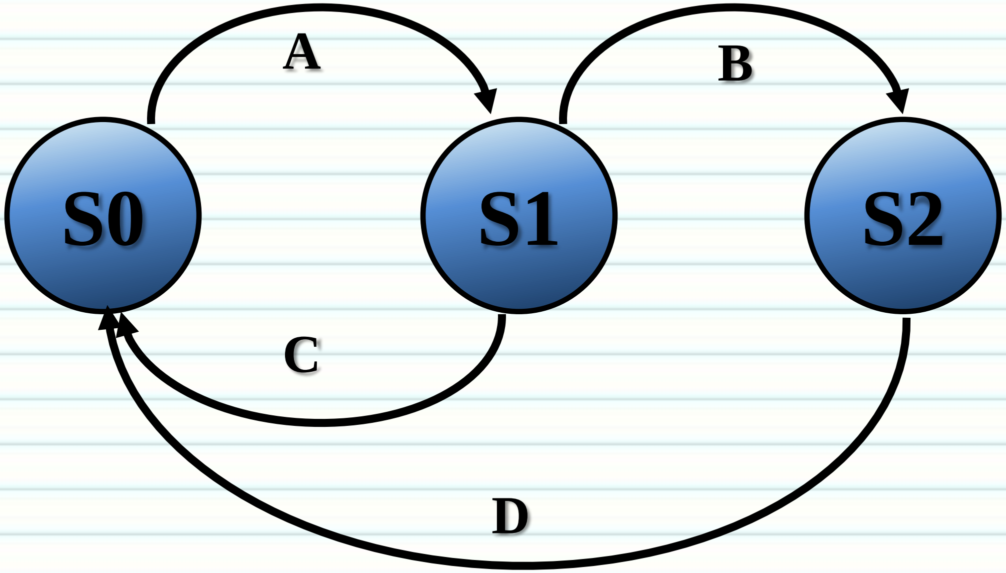


(c) JRCS 2016

Exercises (1)

1. What is missing from this state diagram? Simple answer!

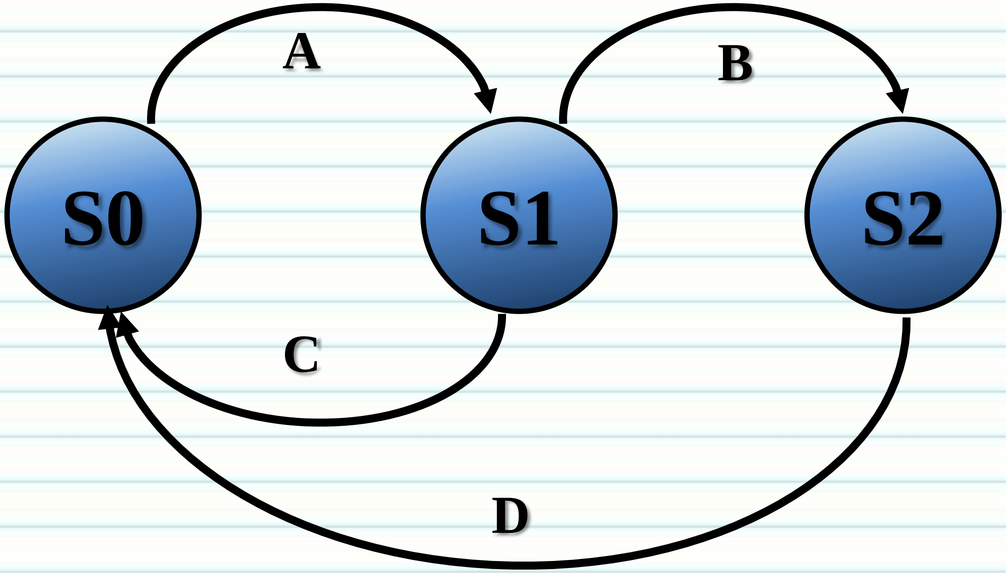
2. Given the following state transition diagram – which of the test cases below will cover the following series of state transitions? S1 S0 S1 S2 S0
   1. C, A, B, D
   2. C, A, C, D
   3. C, A, D, C
   4. C, A, C, D



(c) JRCS 2016

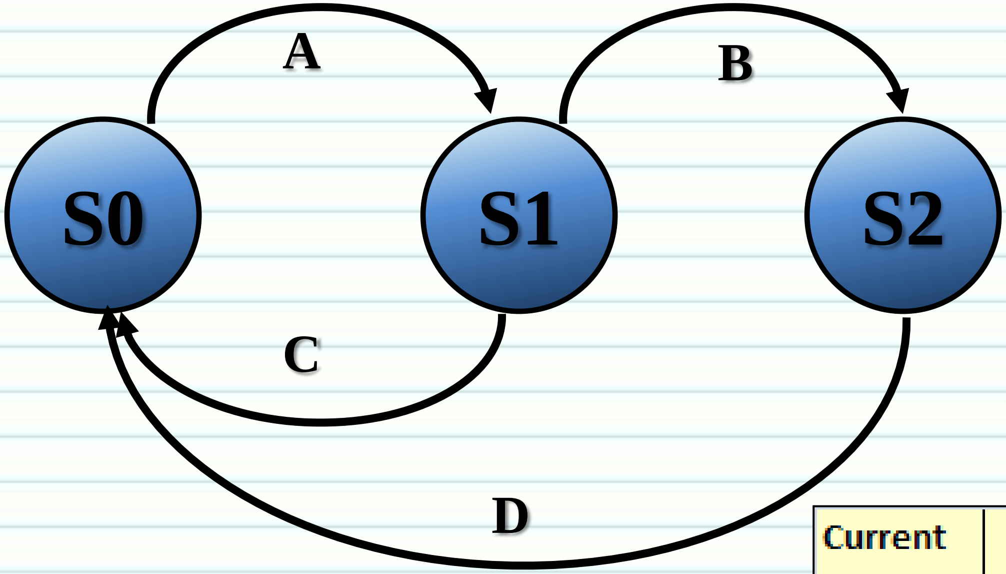
Exercises (2)

- Develop the state transition table for the following state diagram



Exercises (2)

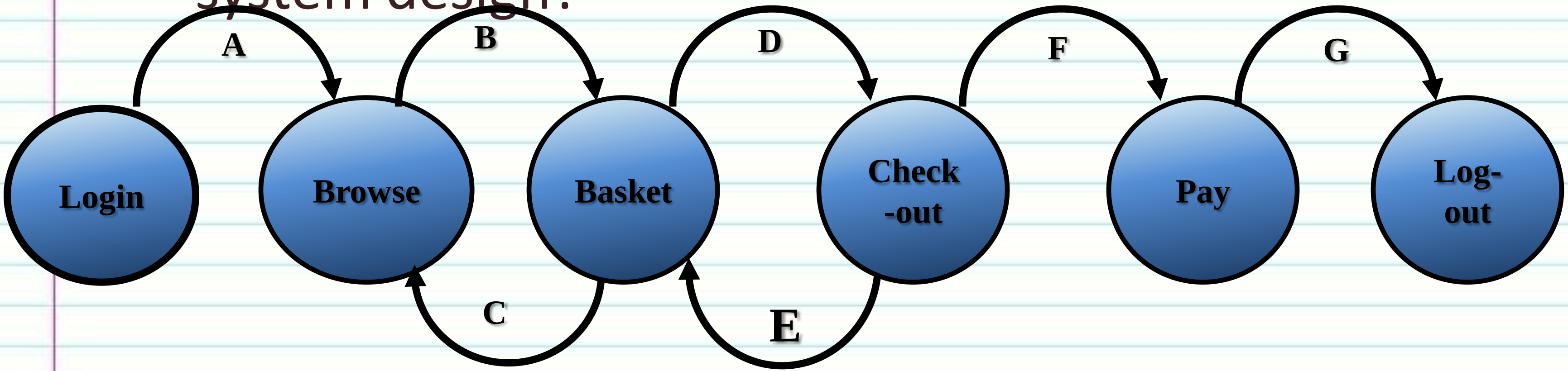
- Develop the state transition table for the following state diagram



Current State	Event	Action	Next State
Start		-	S1
S0	A	-	S1
S1	C	-	S0
	B	-	S2
S2	D	-	S1

Exercises (3)

2. Given the following state transition diagram which of the following series of state transitions contains an INVALID transition which may indicate a fault in the system design?



A. Login Browse Basket Checkout Basket Checkout Pay Logout

B. Login Browse Basket Checkout Pay Logout

C. Login Browse Basket Checkout Basket Logout

D. Login Browse Basket Browse Basket Checkout Pay Logout

(c) JRCS 2016

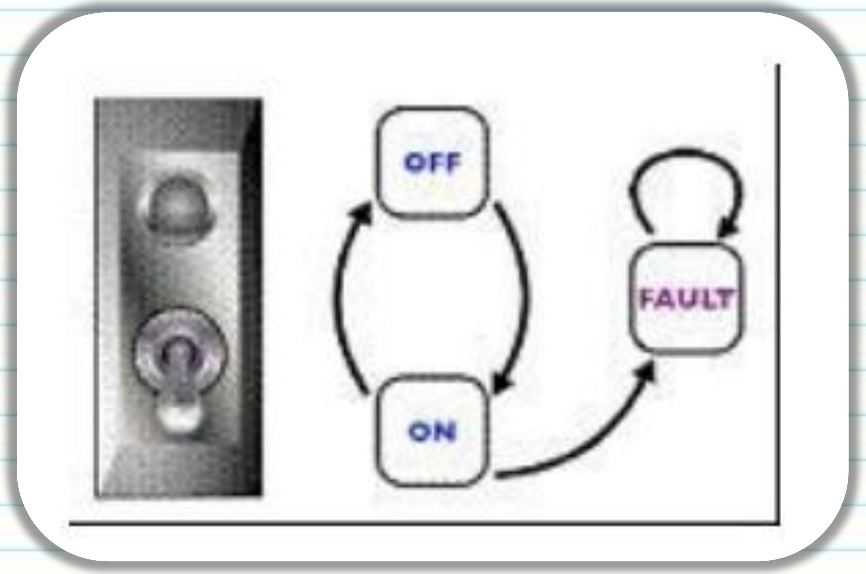
Exercises (4)

3. Consider the following state transition diagram of a switch. Which of the following represents an invalid state transition?

a) OFF, ON, Fault

b) ON,OFF, Fault

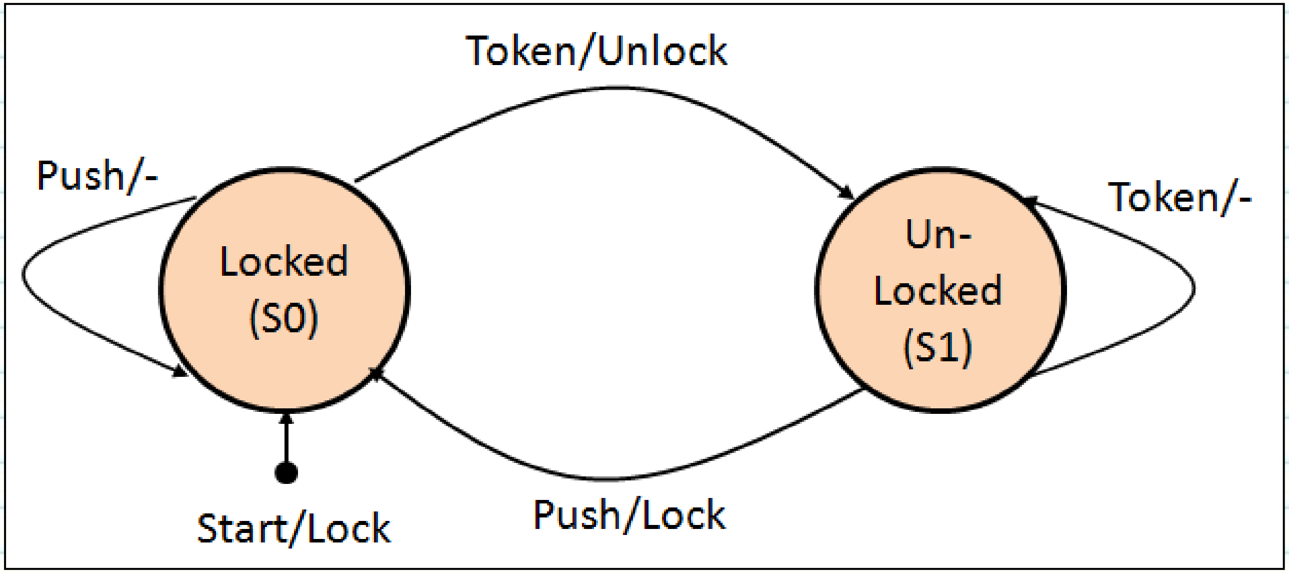
c) Fault, Fault, On



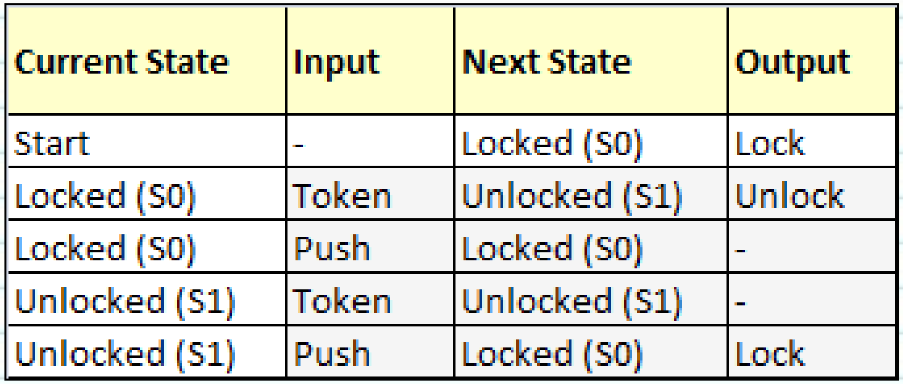
(c) JRCS 2016

Exercises (5)

4. For the examples on the next slides perform the following:
   - – Draw a state transition diagram
   - – Make a state transition table from the diagram
   - – Define logical test cases



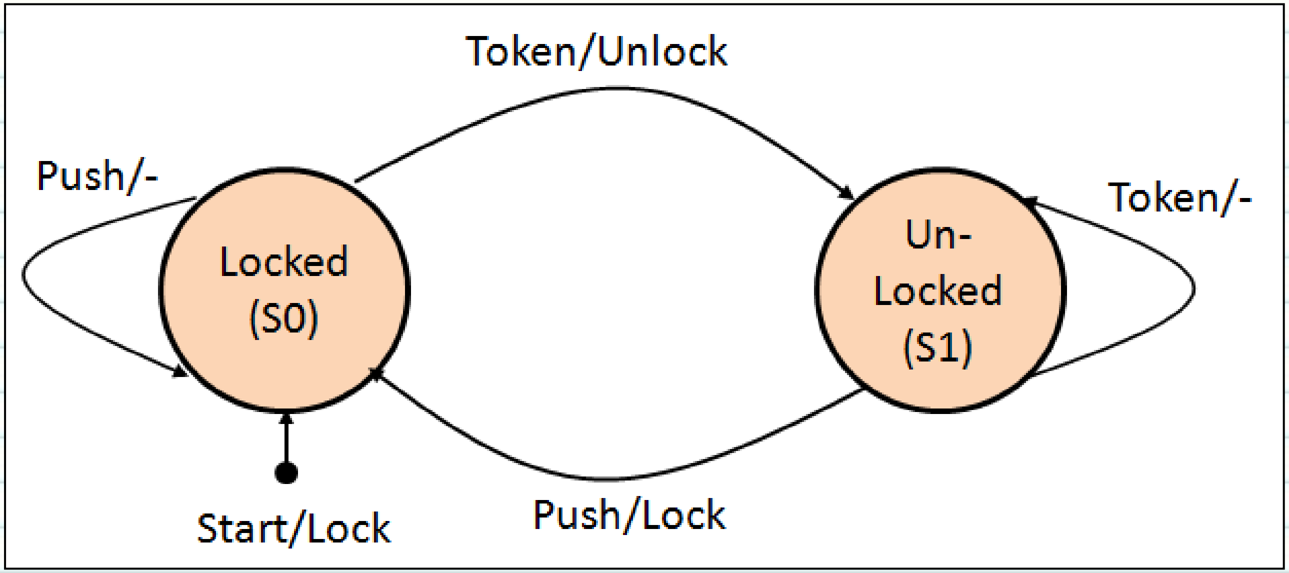
Exercises (5)



Current State	Input	Next State	Output
Start	-	Locked (S0)	Lock
Locked (S0)	Token	Unlocked (S1)	Unlock
Locked (S0)	Push	Locked (S0)	-
Unlocked (S1)	Token	Unlocked (S1)	-
Unlocked (S1)	Push	Locked (S0)	Lock

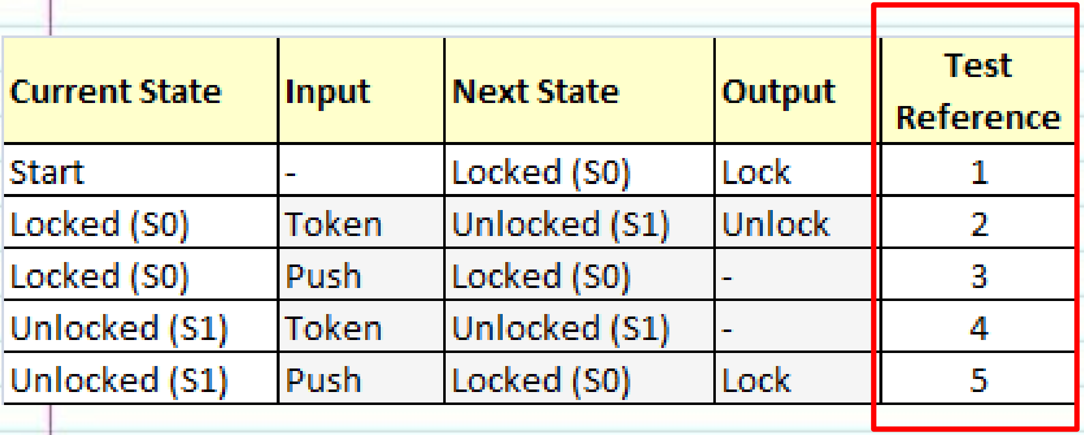
What is unusual about our turnstile's behavior?

How would I fix the state table to address this?



What would I do to test undefined events?

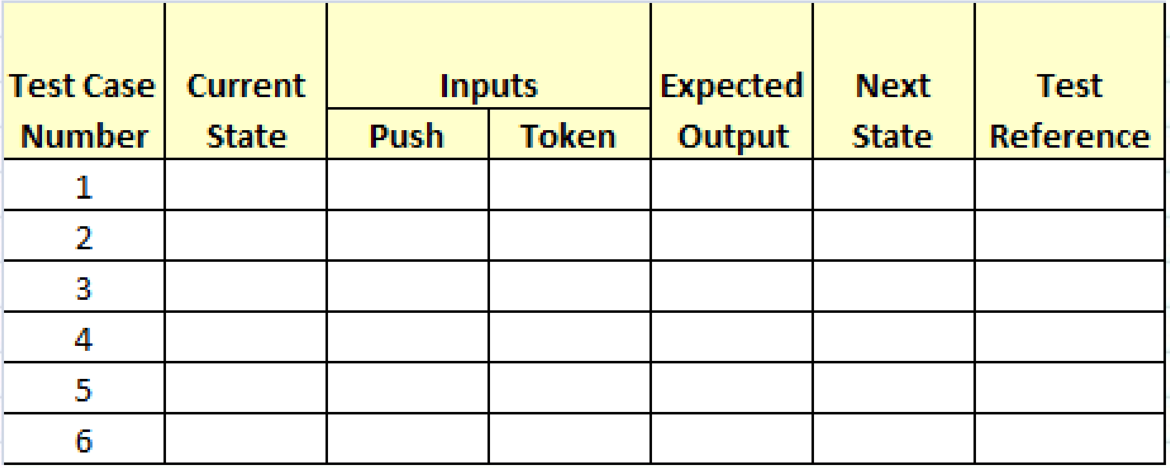
Exercises (5)



Current State	Input	Next State	Output	Test Reference
Start	-	Locked (S0)	Lock	1
Locked (S0)	Token	Unlocked (S1)	Unlock	2
Locked (S0)	Push	Locked (S0)	-	3
Unlocked (S1)	Token	Unlocked (S1)	-	4
Unlocked (S1)	Push	Locked (S0)	Lock	5

- I added a reference number to ease test tracing

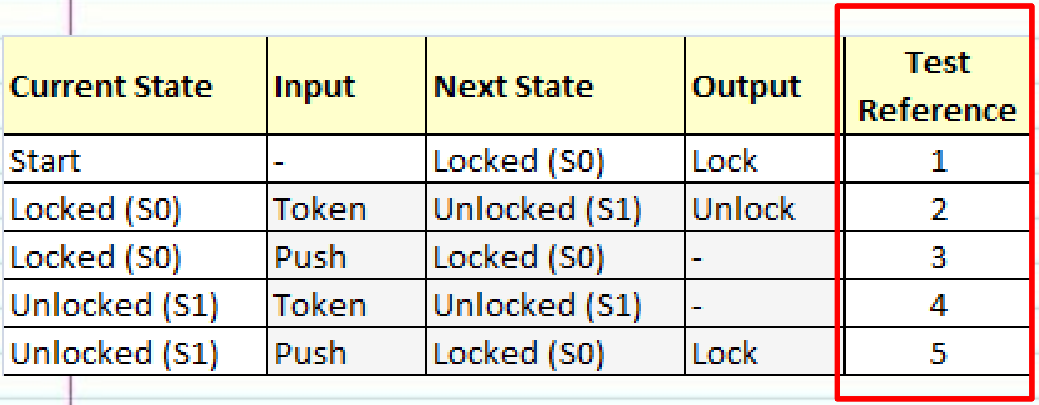
- For the given State transition table develop the test cases using our procedure



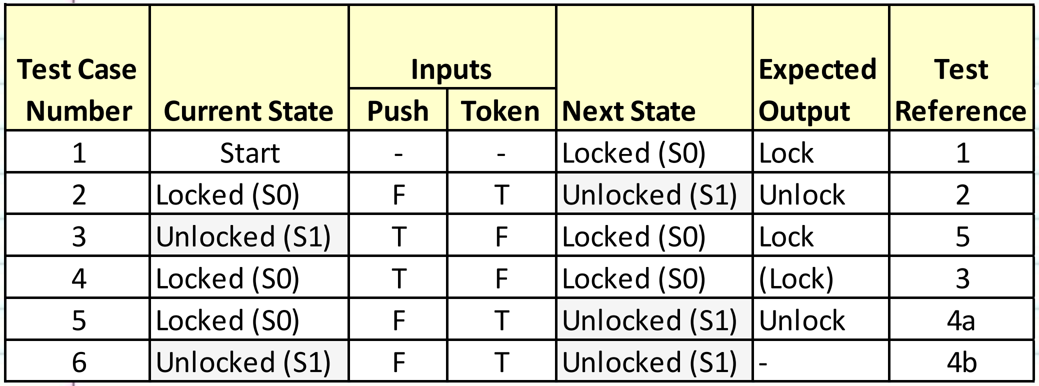
Test Case Number	Current State	Inputs		Expected Output	Next State	Test Reference
		Push	Token			
1						
2						
3						
4						
5						
6						

- Use this as a template

Exercises (5)



Current State	Input	Next State	Output	Test Reference
Start	-	Locked (S0)	Lock	1
Locked (S0)	Token	Unlocked (S1)	Unlock	2
Locked (S0)	Push	Locked (S0)	-	3
Unlocked (S1)	Token	Unlocked (S1)	-	4
Unlocked (S1)	Push	Locked (S0)	Lock	5

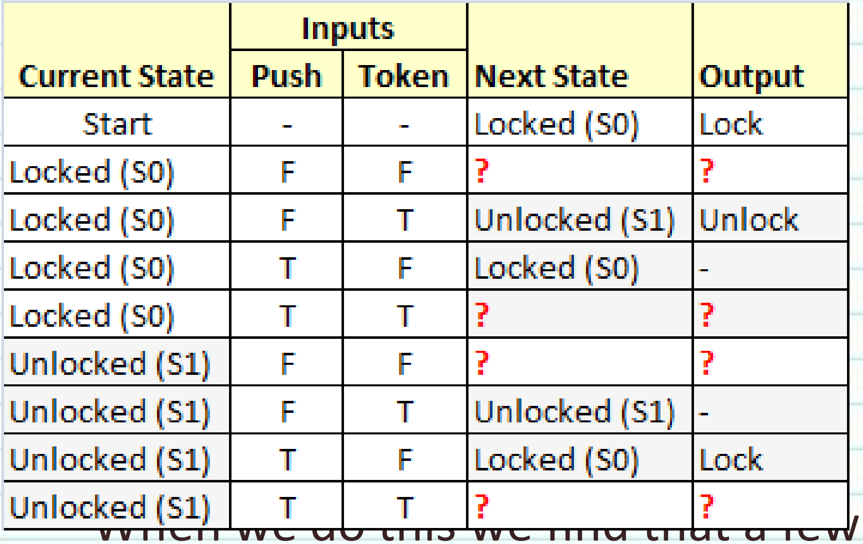


Test Case Number	Current State	Inputs		Next State	Expected Output	Test Reference
		Push	Token			
1	Start	-	-	Locked (S0)	Lock	1
2	Locked (S0)	F	T	Unlocked (S1)	Unlock	2
3	Unlocked (S1)	T	F	Locked (S0)	Lock	5
4	Locked (S0)	T	F	Locked (S0)	(Lock)	3
5	Locked (S0)	F	T	Unlocked (S1)	Unlock	4a
6	Unlocked (S1)	F	T	Unlocked (S1)	-	4b

- Questions:

1. What am I assuming at the Start state? What is my input?
2. Can I make an assumption about the order of inputs Push and Token?
3. What assumption am I making about my ability to set the Lock/unlock command under test?
4. What assumption am I making about being able to set the current state? I am assuming that I must use the last Next State as the Current State.What if I could? Then I could use a single test case for 4

Completing the State Transition Table



Current State	Inputs		Next State	Output
	Push	Token		
Start	-	-	Locked (S0)	Lock
Locked (S0)	F	F	?	?
Locked (S0)	F	T	Unlocked (S1)	Unlock
Locked (S0)	T	F	Locked (S0)	-
Locked (S0)	T	T	?	?
Unlocked (S1)	F	F	?	?
Unlocked (S1)	F	T	Unlocked (S1)	-
Unlocked (S1)	T	F	Locked (S0)	Lock
Unlocked (S1)	T	T	?	?

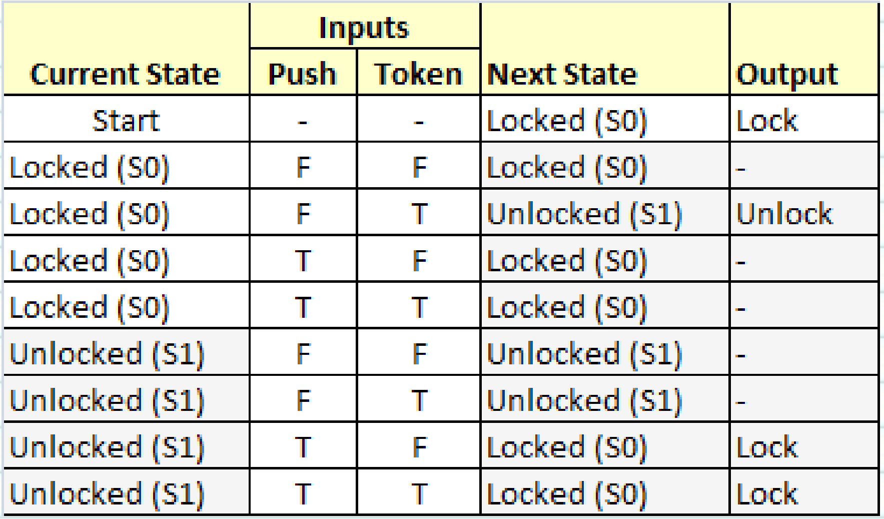
Now we’re going to examine all possible inputs of Push and Token for each state.

- When we do this we find that a few conditions have not been defined
- For the cases where we are in the Locked state or the Unlocked state and both Push and Token are False this is pretty obvious, but what do we do when they are both True?
- It turns out that we need to specify this behavior because it will make a financial difference to our customer. How? Should I keep the turnstile Locked or Unlocked when both Push and Token are true?

(c) JRCS 2016

Completing the State Transition Table (cont.)

- When we specify the appropriate behavior we end up with the following:



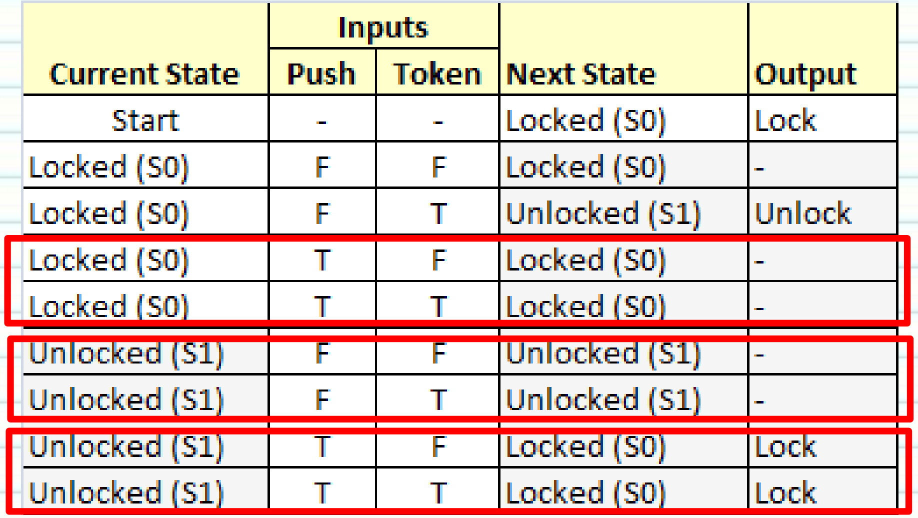
Current State	Inputs		Next State	Output
	Push	Token		
Start	-	-	Locked (S0)	Lock
Locked (S0)	F	F	Locked (S0)	-
Locked (S0)	F	T	Unlocked (S1)	Unlock
Locked (S0)	T	F	Locked (S0)	-
Locked (S0)	T	T	Locked (S0)	-
Unlocked (S1)	F	F	Unlocked (S1)	-
Unlocked (S1)	F	T	Unlocked (S1)	-
Unlocked (S1)	T	F	Locked (S0)	Lock
Unlocked (S1)	T	T	Locked (S0)	Lock

- We can reduce this down because we have several states where both the True and False input combinations produce the same output

- We can see this on the next slide Note that the customer has decided to keep the State in Locked when Push and Token are both T.

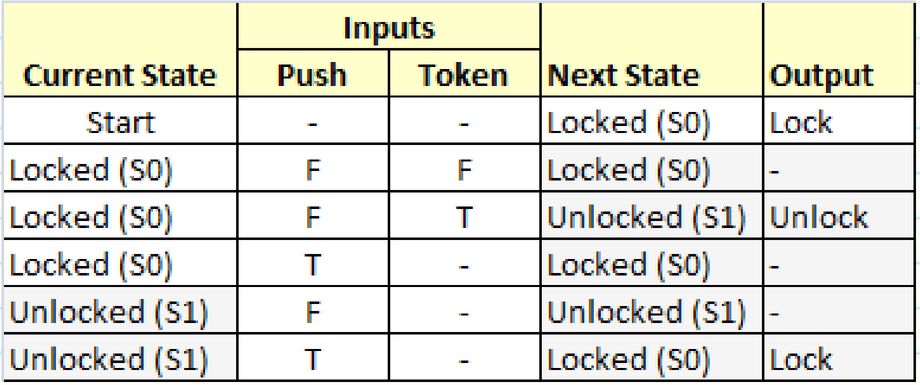
Completing the State Transition Table (cont.)

- We can reduce this down because we have several states where both the True and False input combinations produce the same output



Current State	Inputs		Next State	Output
	Push	Token		
Start	-	-	Locked (S0)	Lock
Locked (S0)	F	F	Locked (S0)	-
Locked (S0)	F	T	Unlocked (S1)	Unlock
Locked (S0)	T	F	Locked (S0)	-
Locked (S0)	T	T	Locked (S0)	-
Unlocked (S1)	F	F	Unlocked (S1)	-
Unlocked (S1)	F	T	Unlocked (S1)	-
Unlocked (S1)	T	F	Locked (S0)	Lock
Unlocked (S1)	T	T	Locked (S0)	Lock

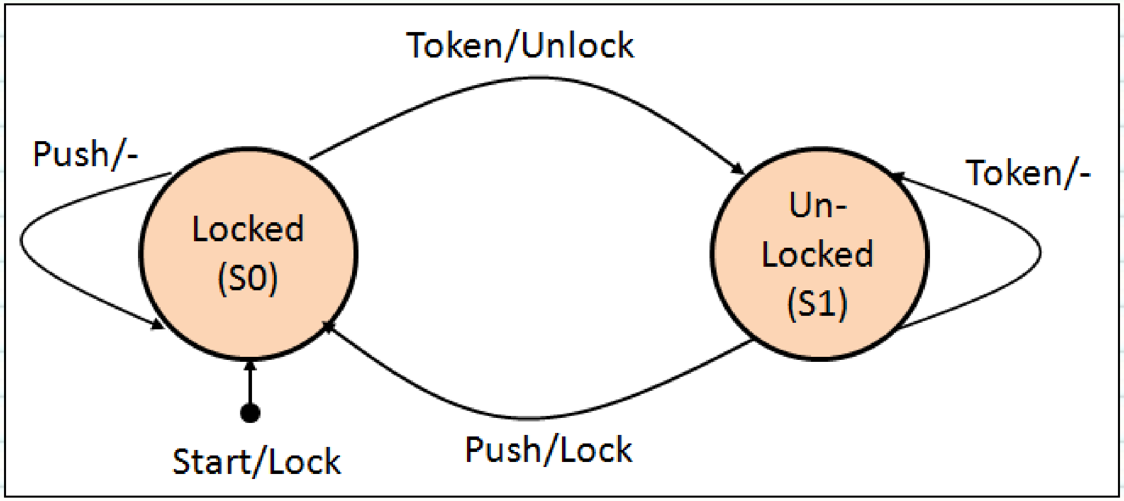
- When we do this we get:



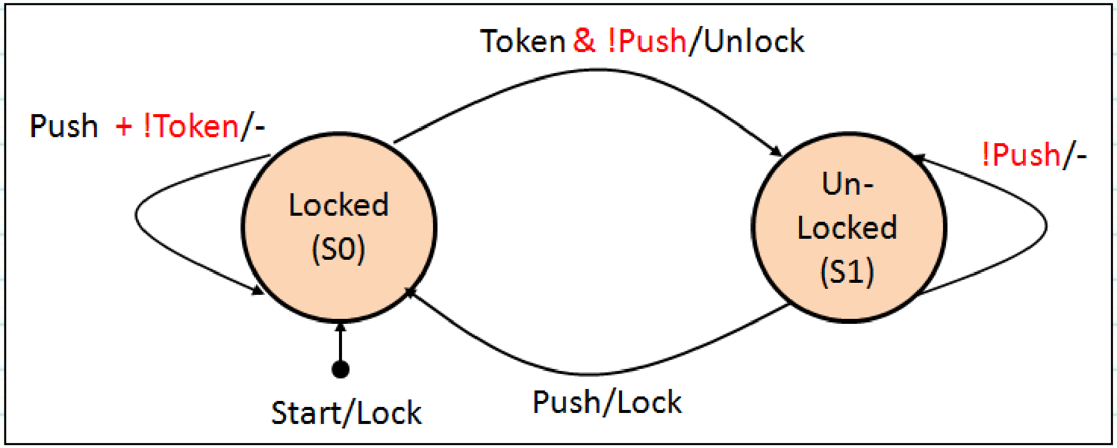
Current State	Inputs		Next State	Output
	Push	Token		
Start	-	-	Locked (S0)	Lock
Locked (S0)	F	F	Locked (S0)	-
Locked (S0)	F	T	Unlocked (S1)	Unlock
Locked (S0)	T	-	Locked (S0)	-
Unlocked (S1)	F	-	Unlocked (S1)	-
Unlocked (S1)	T	-	Locked (S0)	Lock

Updated State Diagram

- With our fully specified and reduced table the state diagram goes from

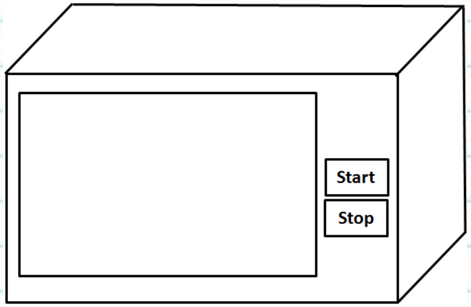


- To the following



Student Exercise - State Machine

- Our company has developed a low-cost microwave oven. It doesn’t have all the features of a more robust model, but due to its simplicity it is expected to sell for a low price and many models are expected to be sold.



- When power is initially applied to the unit, the unit does not “cook”. In order to cook food, the door must be closed and the start button must be pressed. Once cooking, the microwave continues to cook food until either the door is opened or the Stop button is pressed.
- Since this is a quick-to-market release, there may be some areas of uncertainty in the requirements. Safety of a microwave oven is always a paramount concern. Above all else, the microwave must not cook when the door is open. It must not cook when the Stop button is pressed.
- Draw a state diagram of the microwave. Develop the state table showing the Current State, all input combinations, the Expected Output and the Next State. You need to simplify the state table using “-“ for cases where T and F can be used as an input.

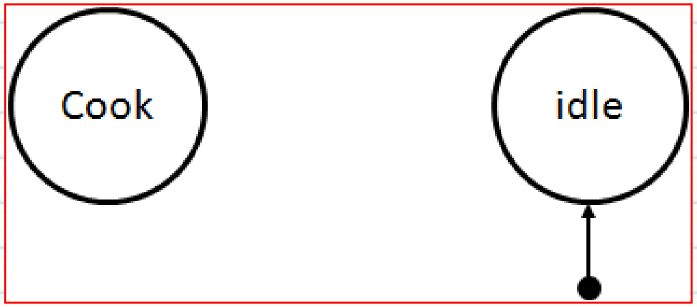
Student Exercise - State Machine

- Update the state diagram using the information gained from the state table where the requirements may have been incomplete.
- From the state table – what are the minimum number of test cases needed to test all unique actions? Is this a reasonable set? How many test cases would you use and why?

(c) JRCS 2016

Answers

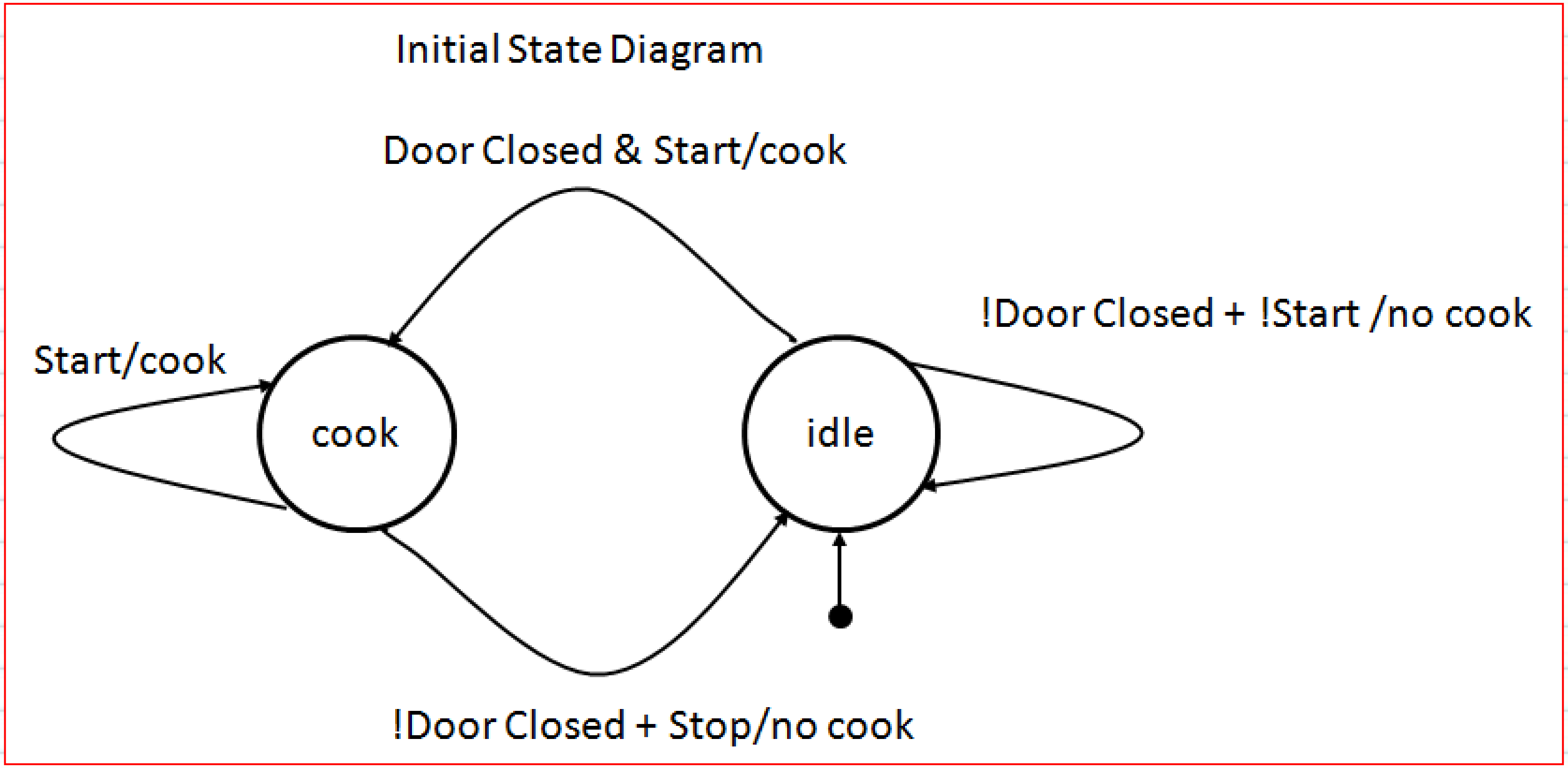
- We can initially specify this using two states which is a natural place to start since the microwave (in its simplest form) is either cooking or idle. Technically, cooking would be setting the output to power the Magnetron tube and idle would be setting the output to not power the tube. For this step we start with the two states and then consider the inputs and outputs.



- We have three inputs: Start, Stop, and Door Closed.
- Start (represents the Start button being pushed) – {T,F} T: button pushed, F: button not pushed
- Stop (represents the Stop button being pushed) – {T,F} T: button pushed, F: button not pushed
- Door closed (represents the door being closed) – {T,F} T:door closed, F: door open
- We have one output: Cook
- Cook (represents turning on or off the Magnetron tube) – {T,F} T:tube on (cooking), F: tube off (idle)

(c) JRCS 2016

Answers (cont.)



We see that we start in the idle state and that we stay in the idle state until the Door is closed and the Start button is pressed. We only issue the Cook command when going to the cook state or transitioning back again. When Cook all Start button pushes cause no real change in cooking state. When in Cook if the door is opened or the Stop button is pressed we stop cooking.

Answers (cont.)

The Start and Stop buttons can be NOT pressed (both F) at the same time ().

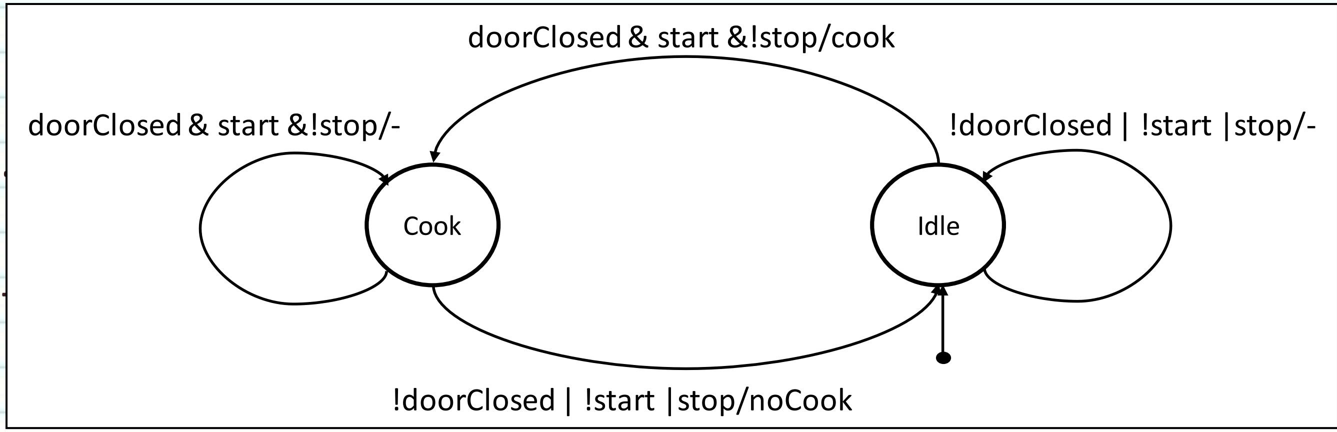
This would be a simple case of no user action during that input cycle, so we would take no action – in essence, we ignore this as an input.

Current	Input			Exp Out	
State	doorClosed	start	stop	cook	Next State
start	-	-	-	NoCook	idle
idle	F	F	F	NoCook	idle
idle	F	F	T	NoCook	idle
idle	F	T	F	NoCook	idle
idle	F	T	T	NoCook	idle
idle	T	F	F	NoCook	idle
idle	T	F	T	NoCook	idle
idle	T	T	F	Cook	cook
idle	T	T	T	NoCook	idle
cook	F	F	F	NoCook	idle
cook	F	F	T	NoCook	idle
cook	F	T	F	NoCook	idle
cook	F	T	T	NoCook	idle
cook	T	F	F	NoCook	idle
cook	T	F	T	NoCook	idle
cook	T	T	F	Cook	cook
cook	T	T	T	NoCook	idle

The bigger dilemma is when we have the Start and Stop buttons pressed (both T) at the same time – our requirements didn't say what to do – or did they…? We have the overriding requirement "It must not cook when the Stop button is pressed", so what is the correct action to do here? We must ignore the Start button and meet the requirement that says the oven must not cook when the Stop button is pressed.

Answers (cont.)

- So the minimum answer would be 9 test cases.
- A better answer is 8 tes cases with inspections required of the code for the equivalence cases (don’t cares).

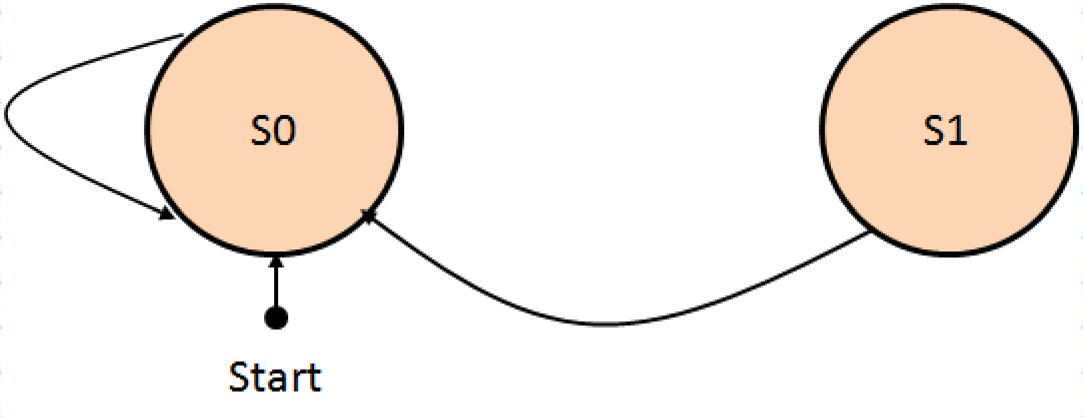

doorClosed & start &!stop/cook
doorClosed & start &!stop/-
!doorClosed | !start |stop/-
Cook
Idle
!doorClosed | !start |stop/noCook

Current	Input			Exp Out	Next State
State	doorClosed	start	stop	cook	
start	-	-	-	NoCook	idle
idle	F	-	-	NoCook	idle
idle	T	F	-	NoCook	idle
idle	T	T	F	Cook	cook
idle	T	T	T	NoCook	idle
cook	F	-	-	NoCook	idle
cook	T	F	-	NoCook	idle
cook	T	T	F	Cook	cook
cook	T	T	T	NoCook	idle

But for safety critical devices all combinations of inputs are required, so the full answer would be 17 test cases as depicted in the un-minimized table.

Important Definitions

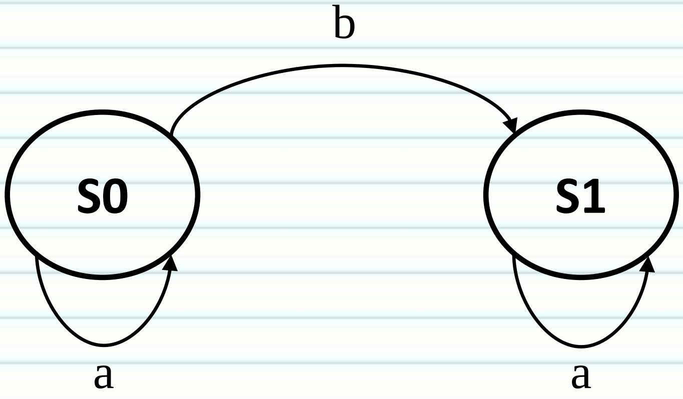
- dead state - a trapped state - where there is no ability to leave that state (the state has no outgoing transitions, but has incoming transitions) - S0



- unreachable state - a state that cannot be reached (has no incoming transitions) - can be true for either a state diagram or table - S1 (the arrow from S1 to S0 would be erroneous here)

Common Problems with State Machines (cont.)

- No transition of input b when in state S1



Example

There is a soda machine. It has a quarter slot (Q), a select soda button (S) - only one kind of soda, a return change button (R). It has the following outputs/actions: Dispense soda (D), Return change (C)), and a LED display (M). Soda's are 75 cents.

Pressing the S button causes the soda to be dispensed (D) when 75 cents credit or more is received, otherwise ignored. Pressing the (R) returns one deposited quarter (Q). D dispenses a soda, and indicates the credit is less 75 cents.

Initial state - (I) - the machine has no credit, has unlimited sodas, displays "Welcome", and is on.

Inputs

quarter slot (Q)

select soda button (S)

return change button (R)

Outputs/actions

Dispense soda (D)

Return change (C)

Message (M)

Condition	Message
No quarters	Welcome
1 quarter deposited	25 cents credit
2 quarter deposited	50 cents credit
3 quarter deposited	75 cents credit

(c) JRCS 2016

Example (cont)

Rules for state diagram and sequence enumeration (here and **homework**):

- Each state MUST address all inputs.
- Inputs
  1. show inputs as a single letter
  2. only show positive (true) values for inputs (unless and interlock must be added)
  3. Inputs cannot occur simultaneously - unless stated otherwise - this means for each state only one T (true) input at a time
- Outputs
  1. Show all outputs with each sequence e.g., !D, !C, M="Welcome"

Rules for this soda machine

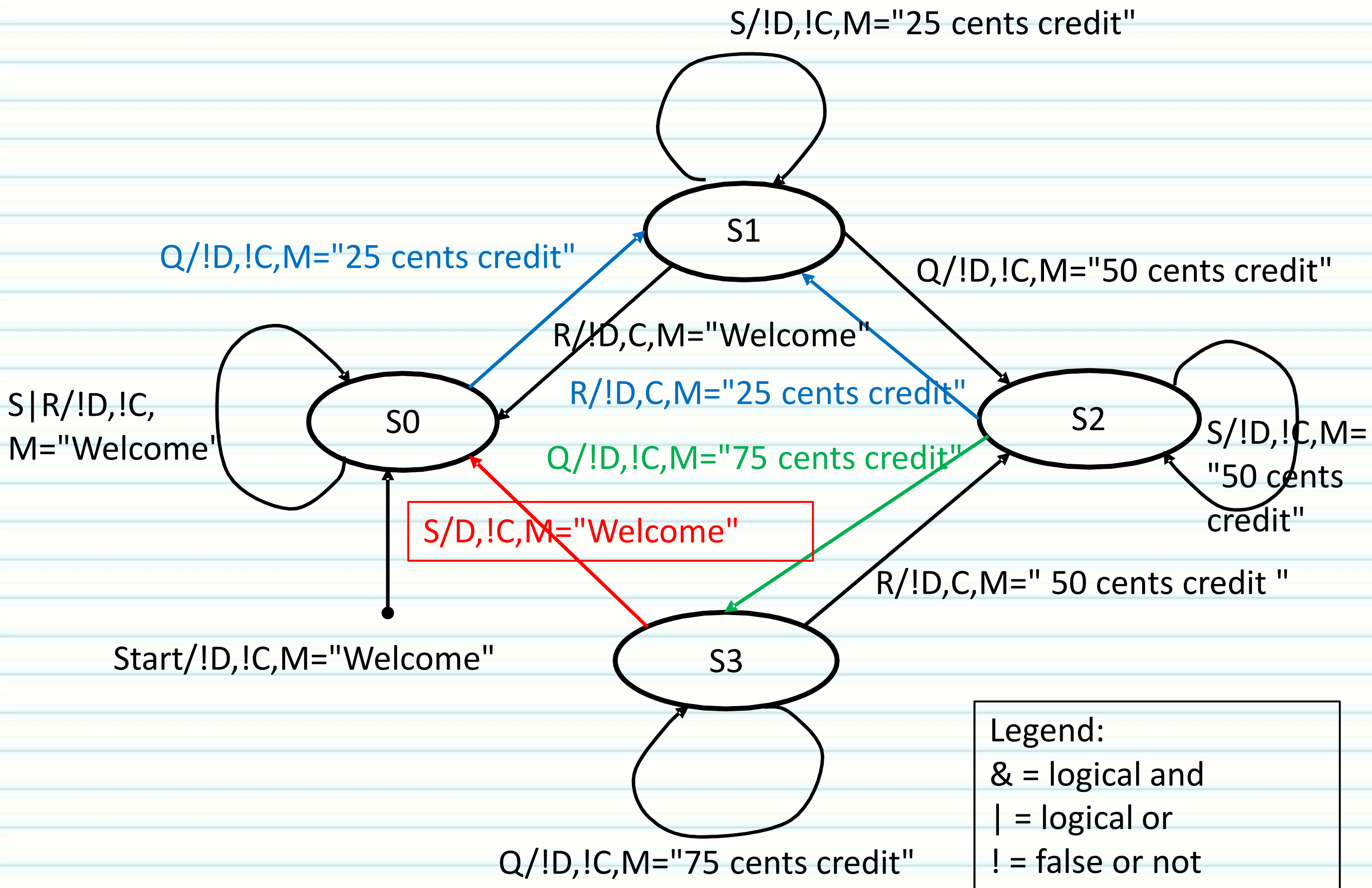
- Assume that to dispense change the output C must be set true - depict this as C. No dispense is !C
- Assume to dispense soda the output D must be set true, you can depict this as D. No dispense is !D
- See problem description for other behavior rules

Example (cont)

Easiest way to solve this as a state diagram is the following

1. Begin with the Start event - this takes us to State S0
2. For each state examine all inputs and develop the responses to each
3. Once finished with all states examine the states from a semantic standpoint
   a. Do they make sense?
   b. Is anything left out?

Example (cont)


S/!D,!C,M="25 cents credit"
S1
Q/!D,!C,M="25 cents credit"
Q/!D,!C,M="50 cents credit"
R/!D,C,M="Welcome"
R/!D,C,M="25 cents credit"
S|R/!D,!C,
M="Welcome"
S0
S2
S/!D,!C,M=
"50 cents
credit"
Q/!D,!C,M="75 cents credit"
S/D,!C,M="Welcome"
R/!D,C,M=" 50 cents credit "
Start/!D,!C,M="Welcome"
S3
Q/!D,!C,M="75 cents credit"

Legend:
& = logical and
| = logical or
! = false or not
- = no action or output