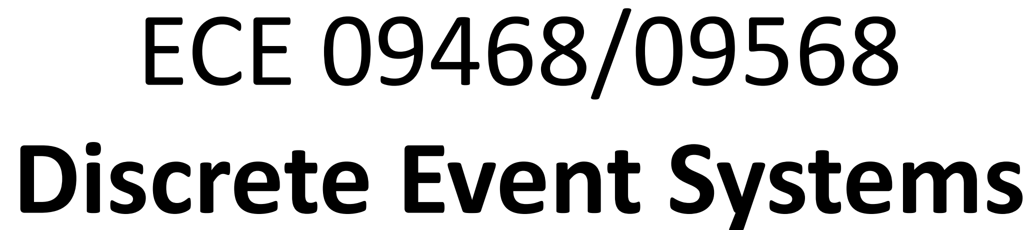


Lecture 4: **Finite State Machine**

Dr. Ying (Gina) Tang
Department of Electrical and Computer Engineering
Rowan University

What is a Finite State Machine?

- A finite state machine (FSM) is an abstract model of a system.
- It represents a system that can only be in a limited (finite) number of states
- It represents a system by defining:
  - » all the possible states a system could have
  - » all the events that could cause the system to change a state
- The main design tool for FSM is the State Transition Diagram (State Diagram).

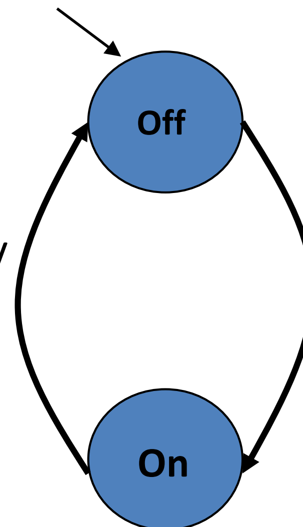
Rowan University

State Diagram

- A state diagram is a graphical representation of a state machine.
- It depicts the relationships between the system states and the events that cause the system to move from one state to another.
- The states are represented by circles; Events are represented by arrows between the states

e.g. Light Switch

Switch down/
open circuit



Switch up/
Close circuit

Name of event that causes transition

Action performed when transition occurs

Basic Terms of FMS

- **Event**: Something that happens to the object instantaneously. It is often implemented as an operation on an object.
- **State:** Defines behavior and attribute values held by an object.
- **State transition:** A change of state (in response to an event).
- **Finite automaton:** Same as finite state machine. (It has a *finite* number of states.)

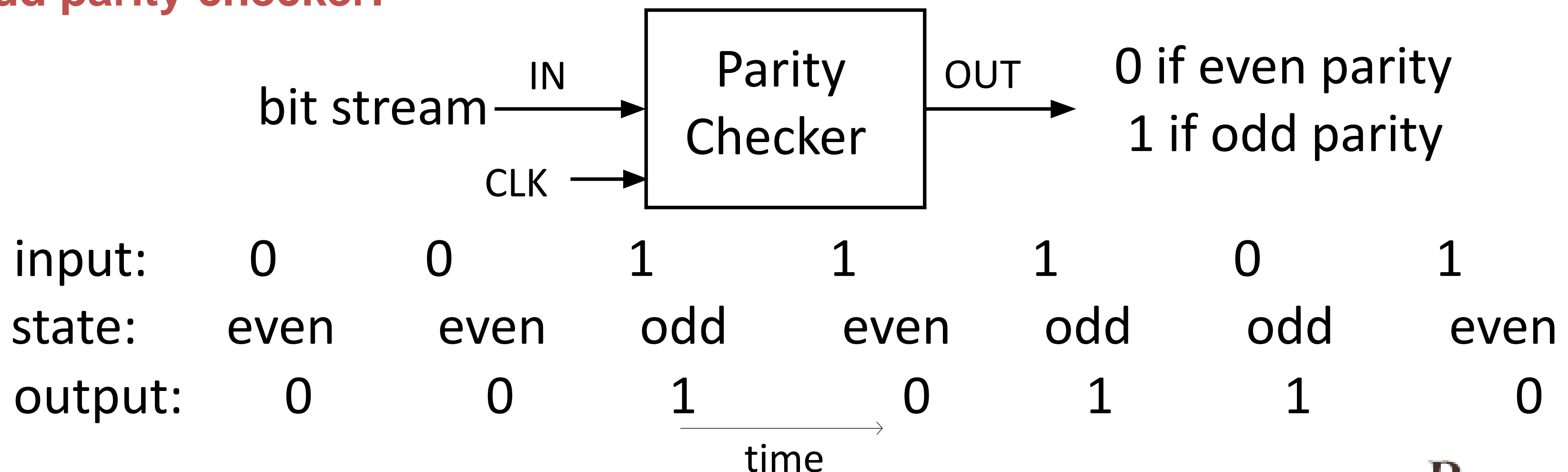
Rowan University

Applications of FMS

- Primarily used for designing
  - Logic circuits: Reference Book: Contemporary Logic Design by Randy H. Katz
  - Computer programs
  - Electronic control systems
- Also used for
  - Pattern recognition
  - Artificial intelligence
  - test and formal verification of systems

FSM for Even/Odd Parity Check

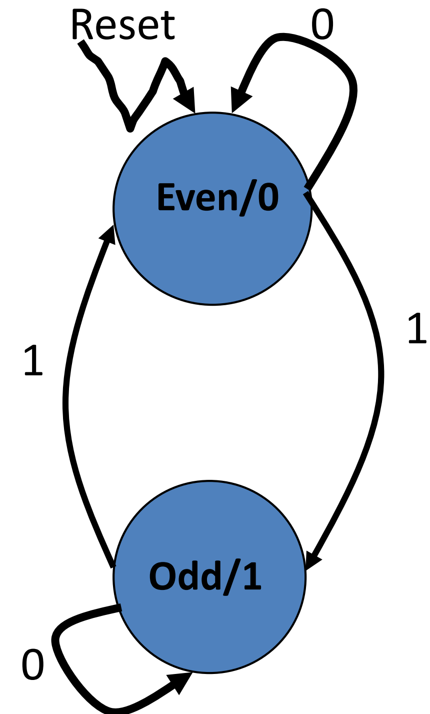
Parity checker counts the number of 1's in a bit-serial input stream. If the checker asserts its output when the input stream contains an odd number of 1's, it is called an **odd parity checker**. If it asserts its output when it has seen an even number of 1's, it is an **even parity checker**.



Rowan University

FSM for Even/Odd Parity Check

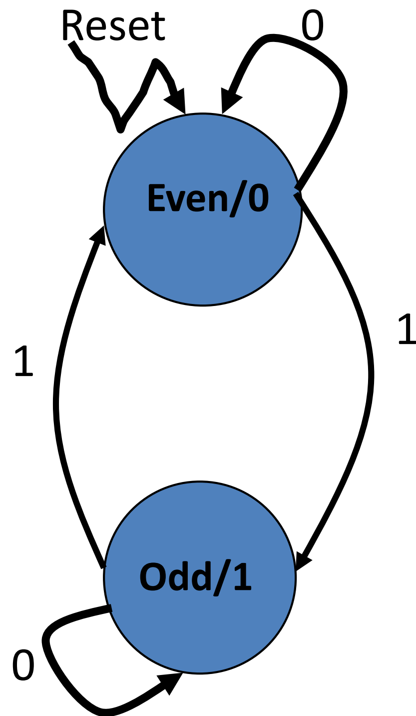
- **State Diagram**
  - States: parity checker is in one of two states (even or odd).
  - Initial state: even.
  - Inputs: 0 or 1. Inputs cause state transitions.
  - Outputs: depends on which state the parity checker is in. 0 for even and 1 for odd. (Moore machine: output is associate with state)



Rowan University

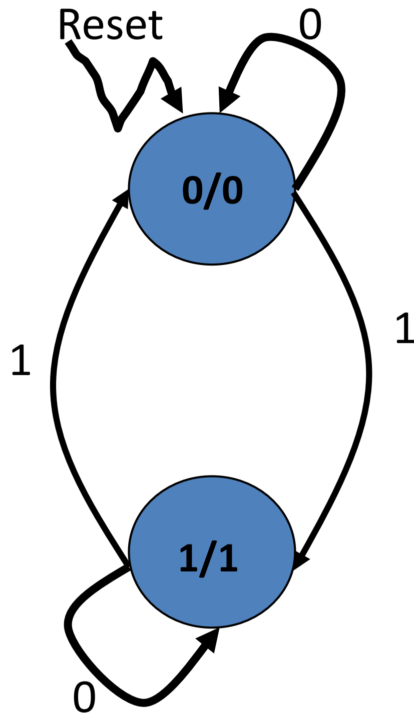
FSM for Even/Odd Parity Check

FSM may be presented by state transition table.



Present State	Input	Next State	Output
Even	0	Even	0
Even	1	Odd	0
Odd	0	Odd	1
Odd	1	Even	1

FSM for Even/Odd Parity Check

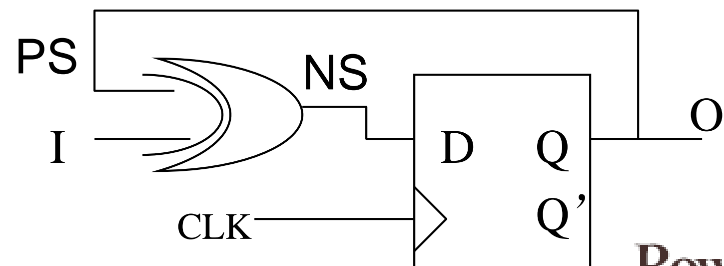


Binary coding: Even (0); Odd (1)

Present State (PS)	Input (I)	Next State (NS)	Output (O)
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

NS = PS XOR I
(exclusive disjunction of Present State and Input)

Implement the parity checker using D-flip flop:

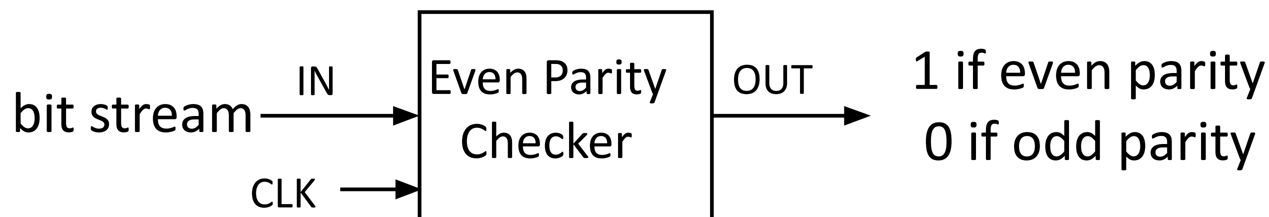


Rowan University

FSM for Even/Odd Parity Check

Your turn…

- Redesign the odd parity checker FSM (Moore machine) to make it check for even parity (that is, assert the output when the input contains an even number of 1's). Show your state diagram



Rowan University

More FSM Applications

Man, Wolf, Goat and Cabbage problem:

A man, a wolf, a goat, and a cabbage are all on one bank of a wide river. The man wishes to take himself and the three others across to the opposite side. How to get them to the other side of the river?

The constraints:

- There is only room in the boat for the man and at most one of the three others.
- The wolf and goat can't be left alone.
- The goat and the cabbage can't be left alone.

More FSM Applications

Man, Wolf, Goat and Cabbage problem:

- **Notation:**

  M – man; W – wolf; G – goat; C : cabbage

- **States: (A, B)**

  Where A and B are subsets of {M,W,G,C}. A represents which entities are on the initial side of the river and the second subset B, the entities on the opposite side.

  - For example, ({MGC},{W} means the man, goat and cabbage are on the initial side and the wolf is on the opposite side -- we simplify the representation as MGC-W

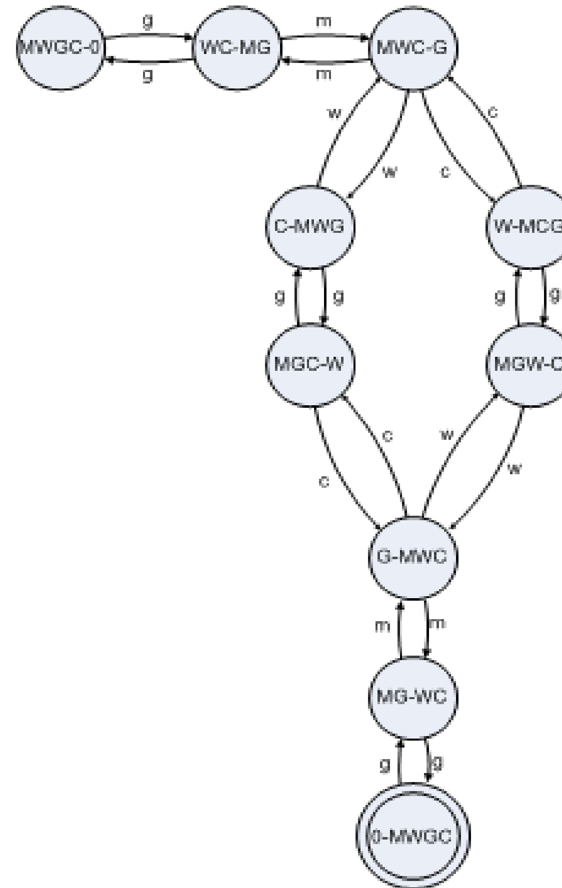
Rowan University

More FSM Applications

Man, Wolf, Goat and Cabbage problem:

- **Exhaust 16 possible states and 6 violate constraints:**

❑ **MWGC-0**	❑ GC-MW
❑ MGC-W	❑ WC-MG
❑ MWC-G	❑ WG-MC
❑ MGC-W	❑ W-MGC
❑ WGC-M	❑ G-MWC
❑ MW-GC	❑ W-MGC
❑ MG-WC	❑ M-WGC
❑ MC-WG	❑ **0-MWGC**



Rowan University

FMS with Outputs

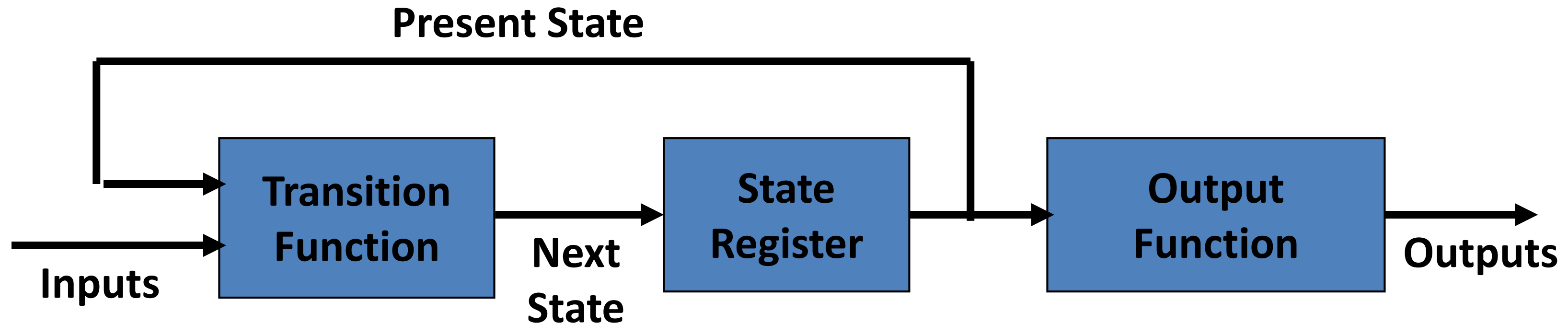
- FSM may generate outputs.
- There are two main methods for handling where to generate the outputs for a FSM.
- They are called
  - a Moore Machine and
  - a Mealy Machine.

  (named after their respective authors.)
- Moore Machine generates an output for each state.
- Mealy Machine generates an output for each transition.

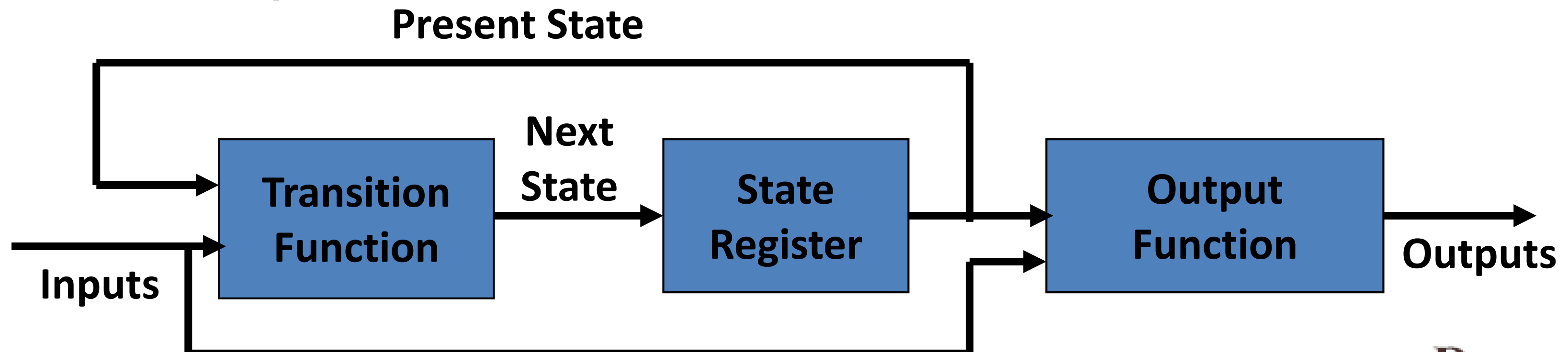
Rowan University

FMS with Outputs

Moore Machine: the output function is a function of a state - it depends only on a state.



Mealy Machine: the output function is a function of both state and input.

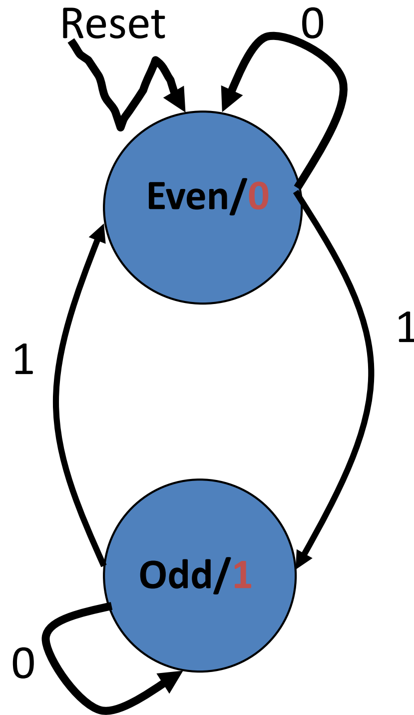


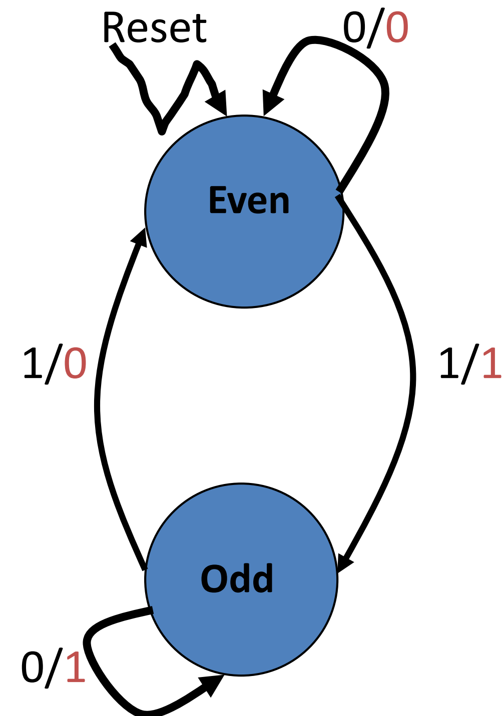
Rowan University

FMS with Outputs

Moore Machine: Output is associated with the state and hence appears after the state transition takes place.

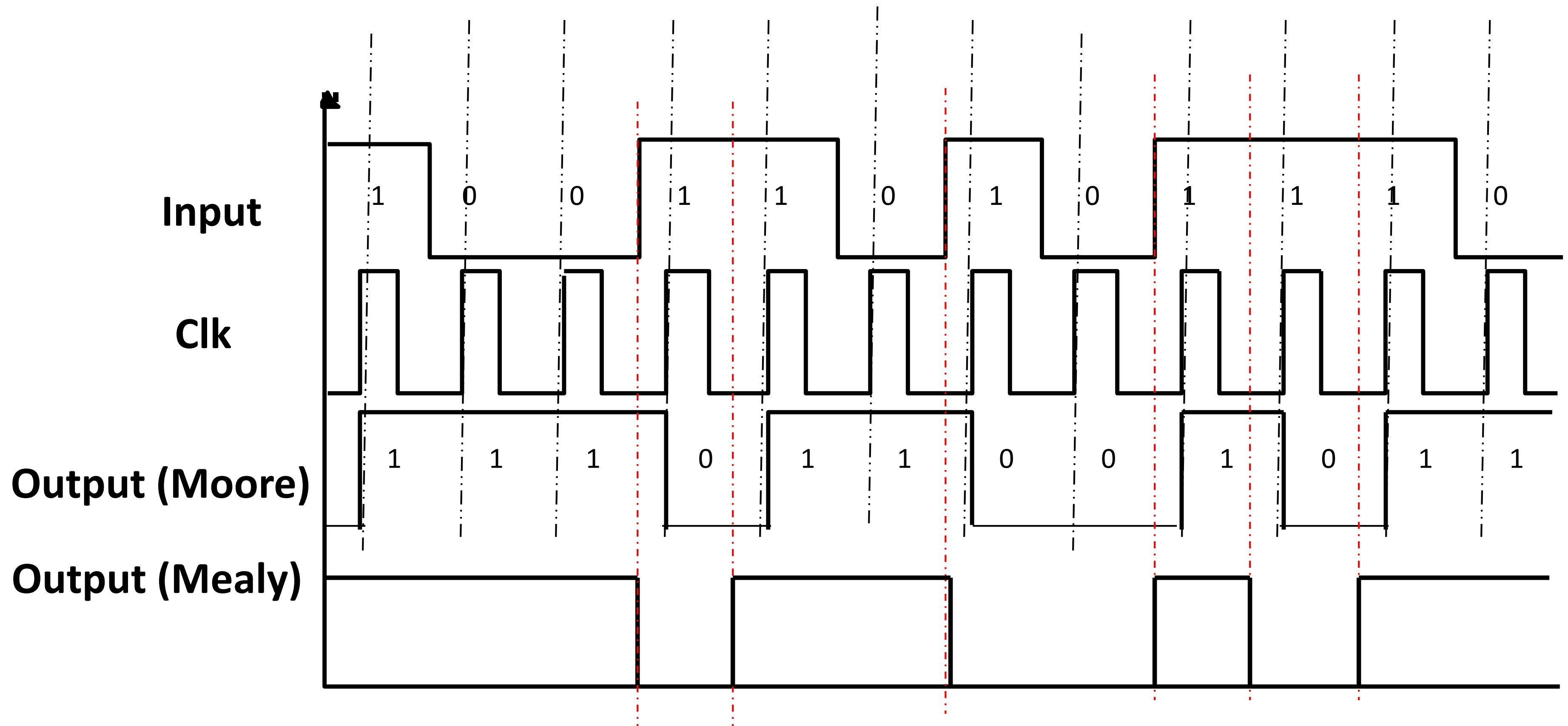
Mealy Machine: Output is associated with the state transition, and appears before the state transition is completed (by the next clock pulse).





Rowan University

FMS with Outputs



Moore: the output change is synchronous with the enabling clock edge.
Mealy: the output changes asynchronously with the enabling clock edge.

Rowan University

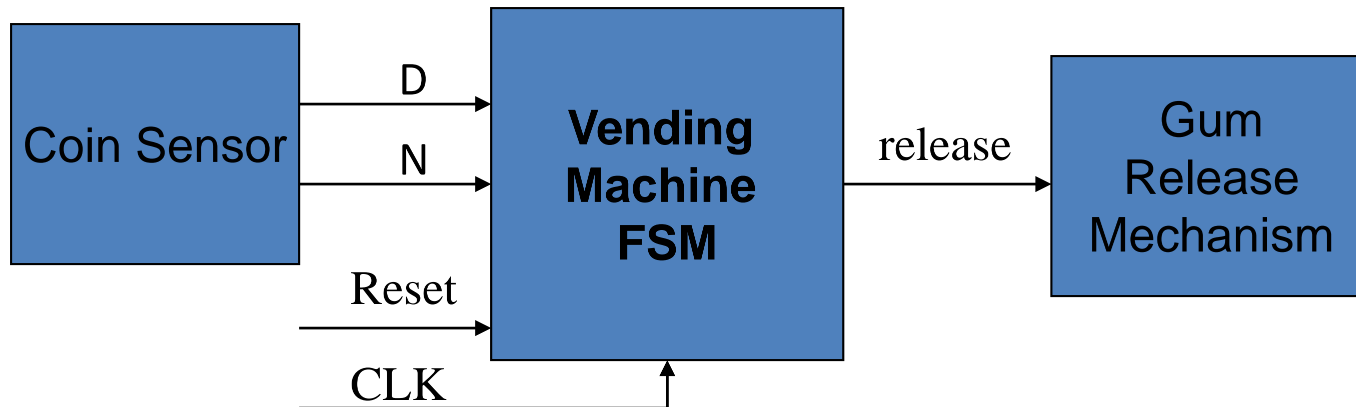
FMS with Outputs

- **Moore Machines are safer to use:**
  - Outputs change at clock edge (synchronous).
  - In Mealy machines, input change can cause output change as soon as logic is done (asynchronous) – a big problem when two machines are interconnected.
- **Mealy Machines react faster to inputs**
  - React in same cycle – don't need to wait for clock
- **Mealy Machines tend to have less states:**
  - Different outputs on arcs (n^2) rather than states (n).
- **Moore and Mealy may be mixed**

Rowan University

Simple Vending Machine

- Deliver package of gum after 15 cents deposited
- Single coin slot for dimes, nickels
- No change
- Once the gum has been delivered, some external circuitry will generate a reset signal to put the control back into its initial state.

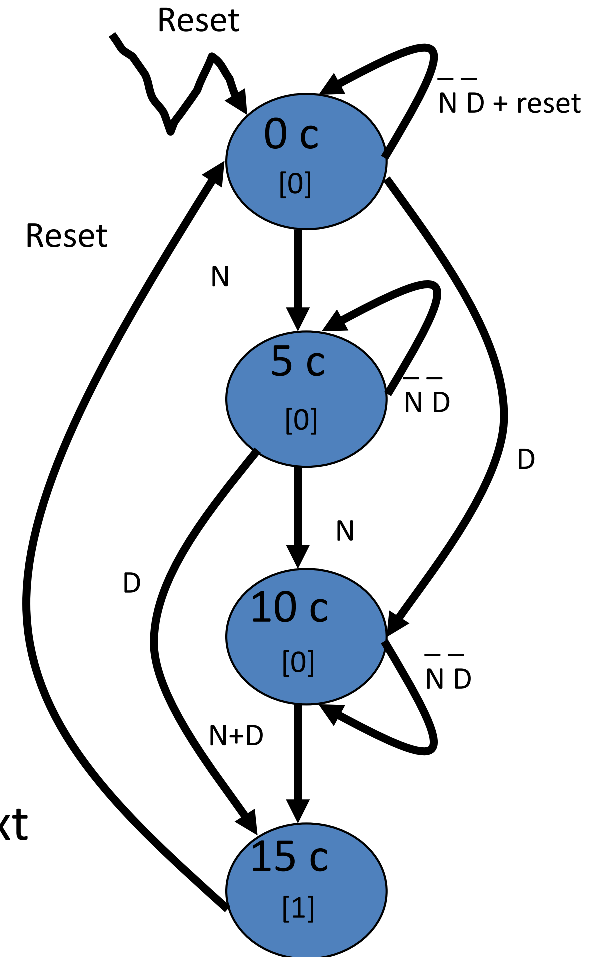


- States: 0 c; 5 c; 10 c; 15 c.
- Inputs/events: insert a dime (D); insert a nickel (N).
- Output: not_release (0); release (1)

Simple Vending Machine

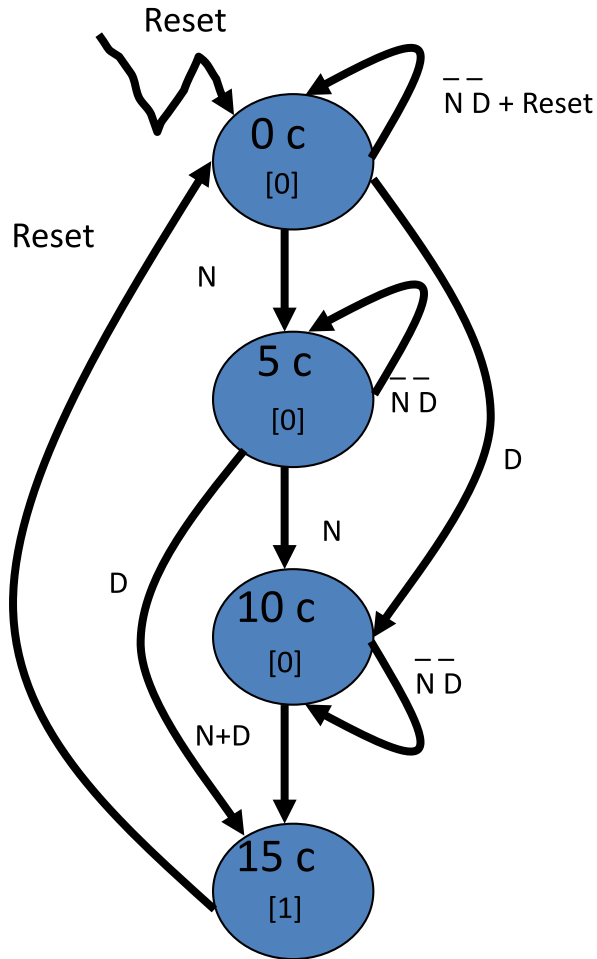
State Diagram:

- Identify States: 0 c; 5 c; 10 c; 15 c.
- Identify Inputs/events:
  - insert a dime (D)
  - insert a nickel (N)
  - Reset
- Identify Outputs:
  - not_release (0)
  - release (1)
- Draw circle for each state.
- For the each state, identify all the possible events and how the state changes when the events happen. Draw the arcs from present state and next state for all the possible events.
- Specify outputs on states (for Moore) or on transitions (for Mealy).



Rowan University

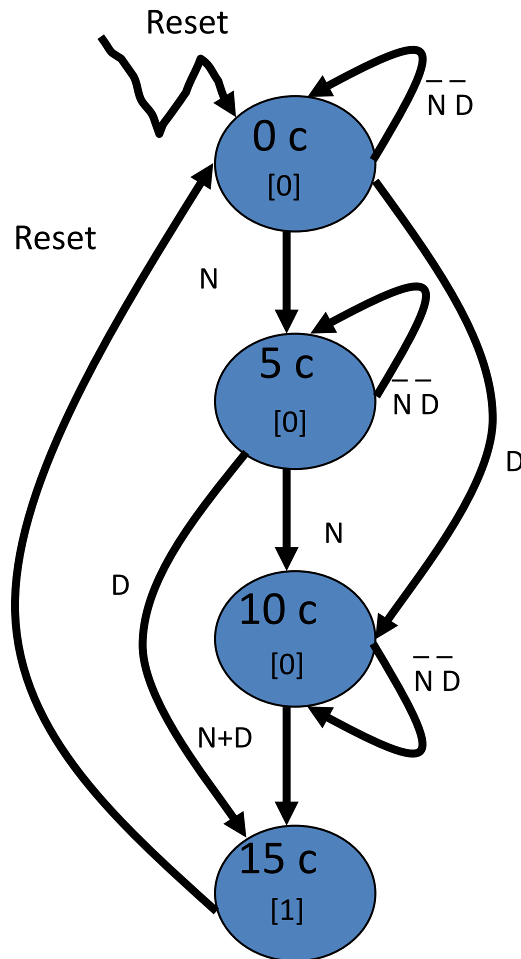
Simple Vending Machine



State table:

Present State	Inputs D	Inputs N	Next State	Output release
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	X	X
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	X	X
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	X	X
15¢	X	X	15¢	1

Simple Vending Machine

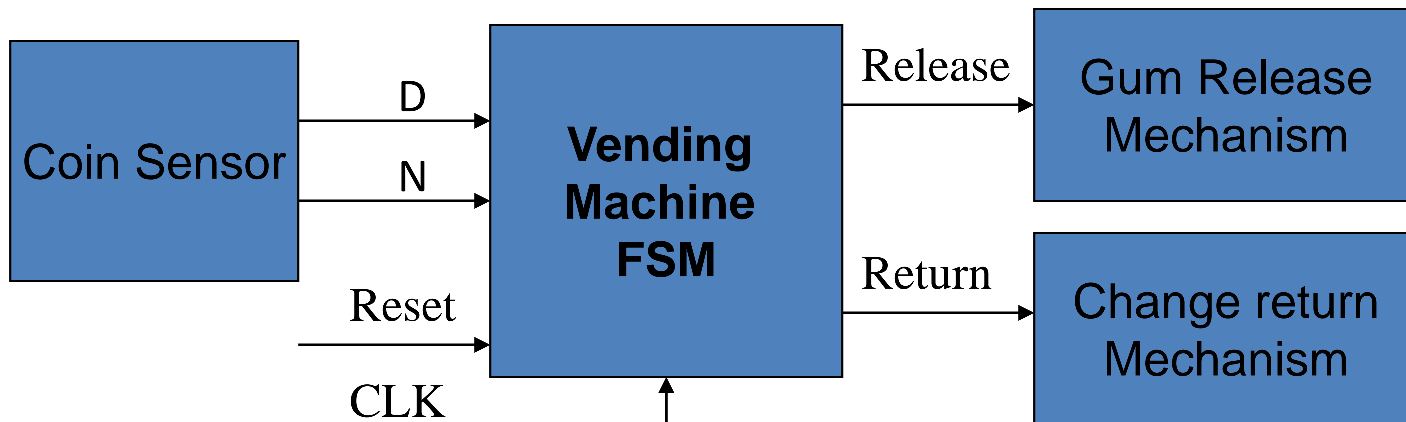


Binary Coding: 2 bits represent 4 states

Present State	Inputs D	N	Next State	Output Release
0¢ (00)	0	0	0¢ (00)	0
	0	1	5¢ (01)	0
	1	0	10¢ (10)	0
	1	1	X	X
5¢ (01)	0	0	5¢ (01)	0
	0	1	10¢ (10)	0
	1	0	15¢ (11)	0
	1	1	X	X
10¢ (10)	0	0	10¢ (10)	0
	0	1	15¢ (11)	0
	1	0	15¢ (11)	0
	1	1	X	X
15¢ (11)	X	X	15¢ (11)	1

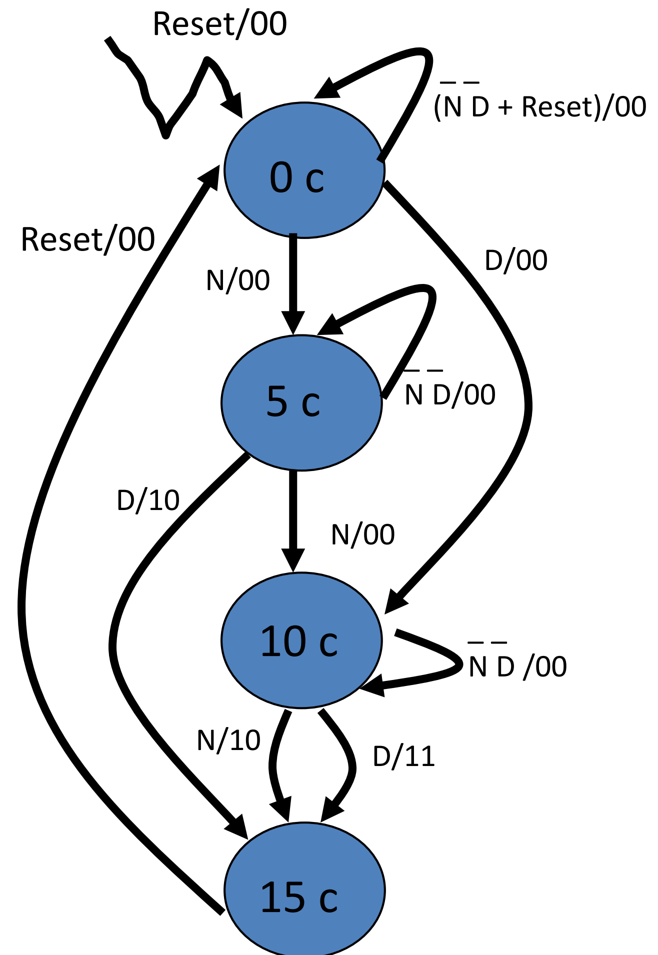
Vending Machine with Returns

- **What if it return changes? i.e.**
  - **Deliver package of gum after 15 cents deposited**
  - **Single coin slot for dimes, nickels**
  - **If more than 15 cents are inserted, the change will be returned. (dime is inserted when 10 cents are in the machine already)**



Vending Machine with Returns

- **States: 0 c; 5 c; 10 c; 15 c.**
- **Inputs/events:**
  - insert a dime (D)
  - insert a nickel (N)
  - Reset
- **Output:**
  - No release (00)
  - Release and no return (10)
  - Release and return a nickel (11)

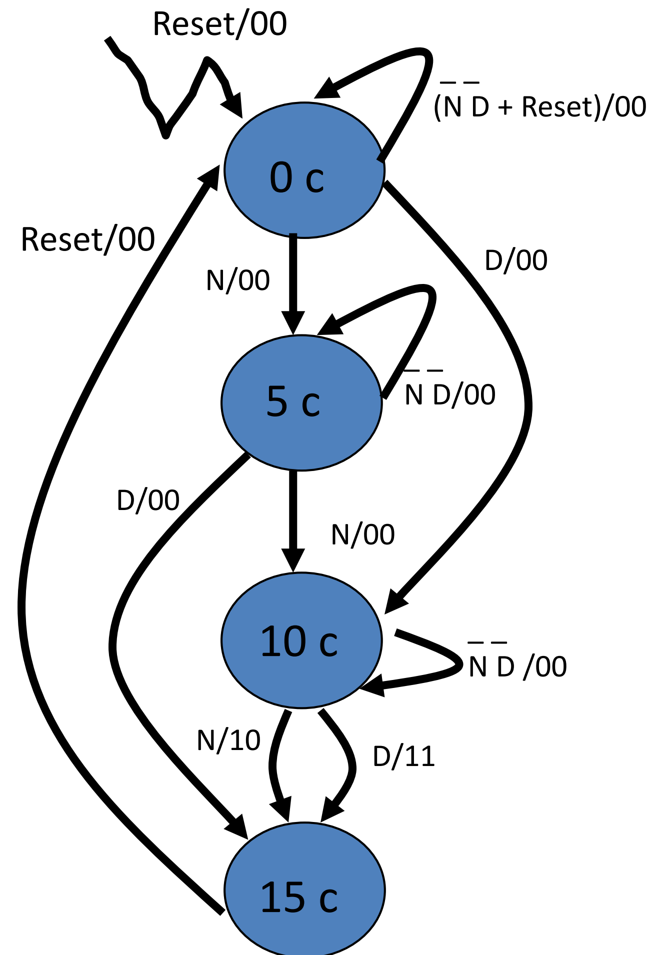


Rowan University

Vending Machine with Returns

- **Exercise:**
  - Design Moore Machine of the vending machine with change return.

  Hint: outputs are associate with states. More states may be needed to show different outputs.



Moor Machine?

Mealy Machine

Rowan University

Another Vending Machine

- **Design a finite state machine for a vending machine controller that accepts nickels (5 cents each), dimes (10 cents each), and quarters (25 cents each). When the value of the money inserted equals or exceeds twenty cents, the machine releases the item and returns change if any (only return up to 15 cents), and waits for next transaction.**

States: 0c (A), 5c (B), 10c (C), 15c (D), 20c/25c/0c (A)

Inputs: Nickel (N), Dime (D), Quarter (Q)

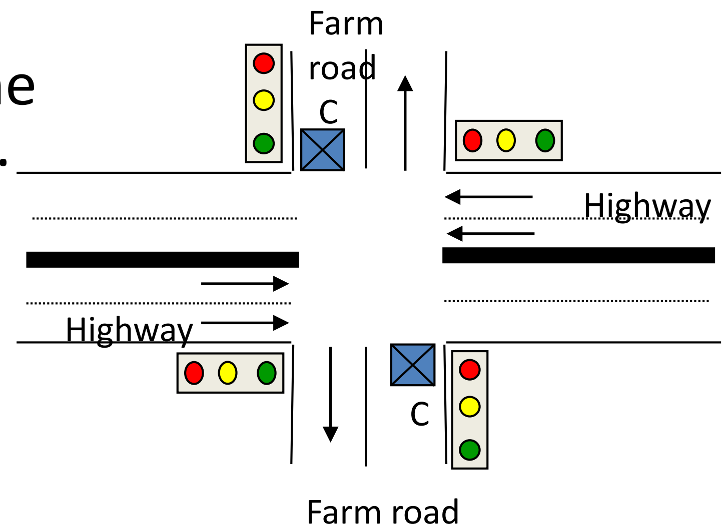
Outputs: No release (000); Release /wo returning change (100); Release and return 5c (110); Release and return 10c (101); Release and return 5c & 10c (111)

Rowan University

Traffic Light Controller

A busy highway intersected by a little used farm-road.

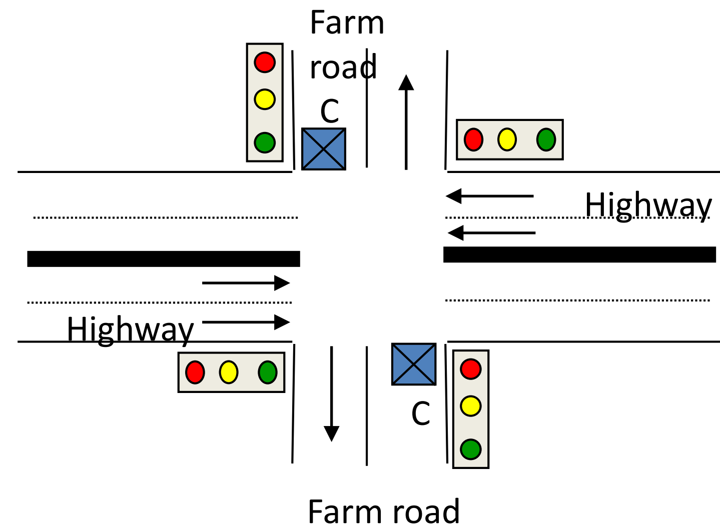
- Detectors are placed along the farm road to raise the signal C as long as a vehicle is waiting to cross the highway. The traffic light controller should operate as follows:
  - If a vehicle is detected on the farm road (C), the highway lights should change from yellow (Y) to red (R), allowing the farm road lights to become green (G).
  - The farm road lights stay G only as long as a vehicle is detected on the farm road (C) and never longer than a set interval (TL) to allow the traffic to flow along the highway.



Rowan University

Traffic Light Controller

- If TL expires or no car detected on farm-road ($\overline{C}$), the farm road lights change from G to Y to R, allowing the highway lights to return to G .
- Even if vehicles are waiting to cross the highway (C), the highway should remain green for a set of interval (TL).



-- Assume there is an external timer that, once set via the control signal ST (set timer), will assert the signal TS after a short time interval has expired. (used for timing yellow lights) and TL after a long time interval (for green lights). The timer is automatically reset when ST is asserted.

Traffic Light Controller

Assume you have an interval timer that generates:

- a short time pulse (TS) and
- a long time pulse (TL),
- A control signal - start timer (ST).
- TS is to be used for timing yellow lights and TL for green lights

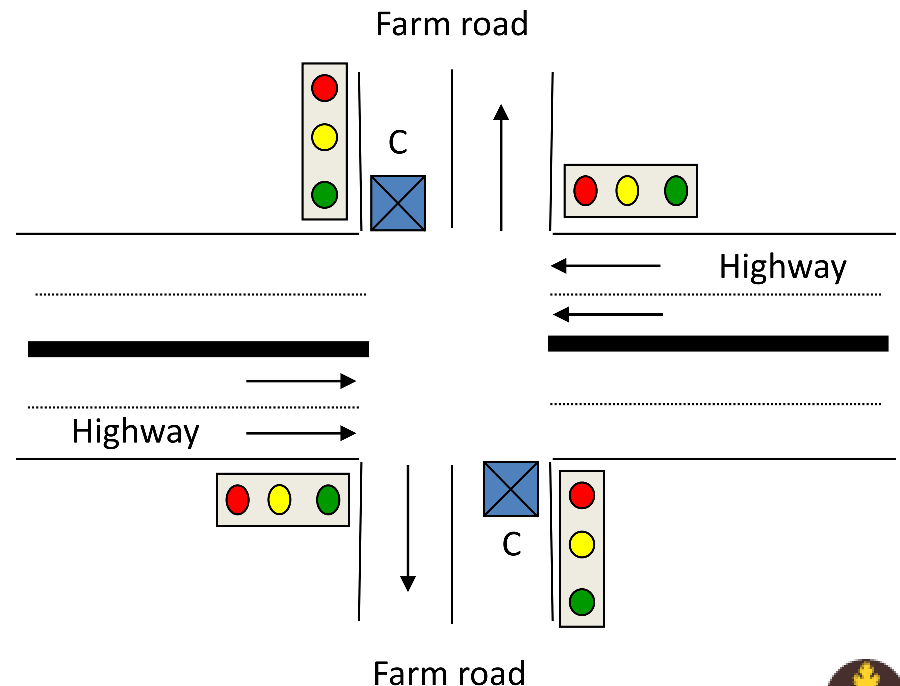
1. Identify States

S_o: Highway green (farm road red)

S_1: Highway yellow (farm road red)

S_2: Farm road green (highway red)

S_3: Farm road yellow (highway red)



Rowan University

Traffic Light Controller

2. Identify the inputs

Reset:

Place controller in initial state

C :

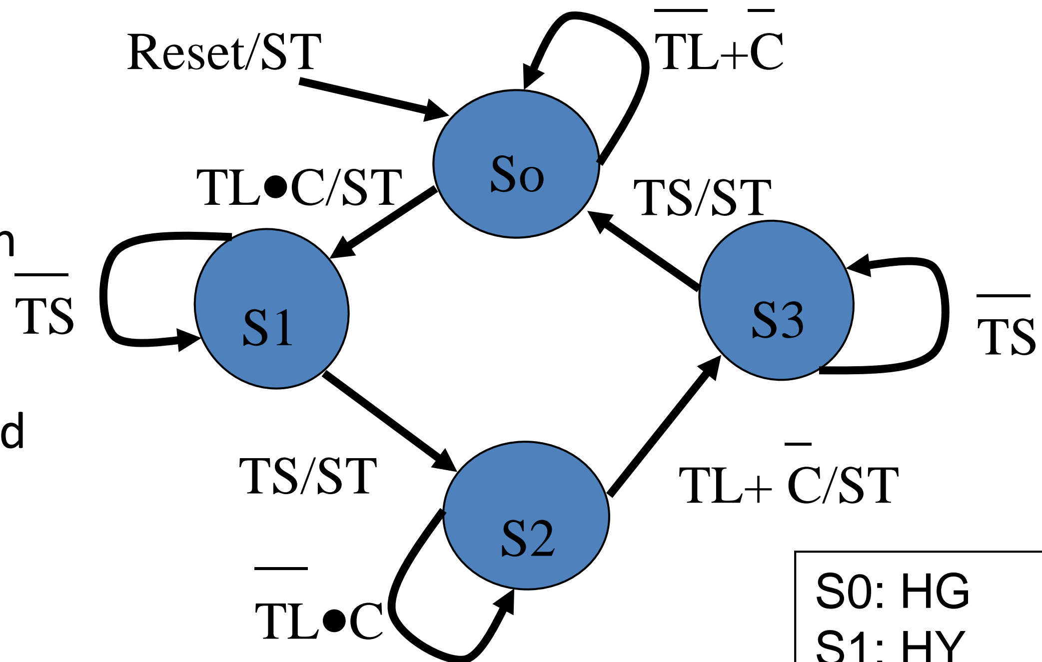
Detects vehicle on farm road in either direction

TS :

Short timer interval has expired

TL :

Long time interval has expired



3. Identify the outputs

HG, HY, HR -Highway Green, Yellow, and Red lights (Moore outputs)

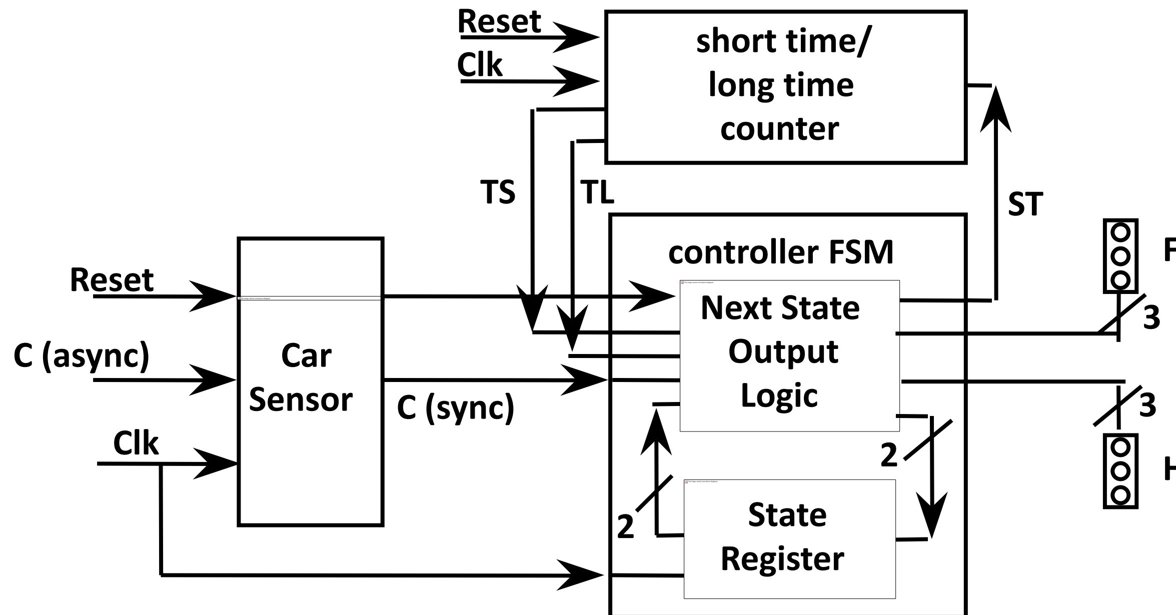
FG, FY, FR - Farmroad Green, Yellow, and Red lights (Moore outputs)

ST - start timing a long and short interval (Mealy output)

Rowan University

Traffic Light Controller

Traffic light system interface



We will use ROM to implement the traffic light controller FSM

Traffic Light Controller

The steps of ROM implementation of FSM:

- Based on the number of <u>states and inputs</u> in FSM, decide of <u>address</u> <u>bits</u>: Reset C TS TL Q1 Q0
- Based on the number of <u>states and outputs</u>, decide <u>word bits</u>: HG HY HR FG FY FR ST P1 P0
- Fill ROM contents starting from address 0

Address						Word (content)								
Reset	C	TS	TL	Q_1	Q_0	HG	HY	HR	FG	FY	FR	ST	P_1	P_0
0	0	0	0	0	0	?	?	?	?	?	?	?	?	?
0	0	0	0	0	1	?	?	?	?	?	?	?	?	?
……						……								

Each address indicates inputs and current state.
Each word (content) indicates outputs and next state.

Rowan University

Traffic Light Controller

input				PS		Output							NS	
Reset	C	TL	TS	Q1	Q0	HG	HY	HR	FG	FY	FR	ST	P1	P0
0	0	-	-	0	0	1	0	0	0	0	1	0	0	0
0	-	0	-	0	0	1	0	0	0	0	1	0	0	0
0	1	1	-	0	0	1	0	0	0	0	1	1	0	1
0	-	-	0	0	1	0	1	0	0	0	1	0	0	1
0	-	-	1	0	1	0	1	0	0	0	1	0	1	0
0	1	0	-	1	0	0	0	1	1	0	0	0	1	0
0	0	-	-	1	0	0	0	1	0	1	0	0	1	1
… …														

Address	Word (content)
Reset C TS TL Q_1 Q_0	HG HY HR FG FY FR ST P_1 P_0
0 0 0 0 0 0	1 0 0 0 0 1 0 0 0
0 0 0 0 0 1	0 1 0 0 0 1 0 1 0
……	……

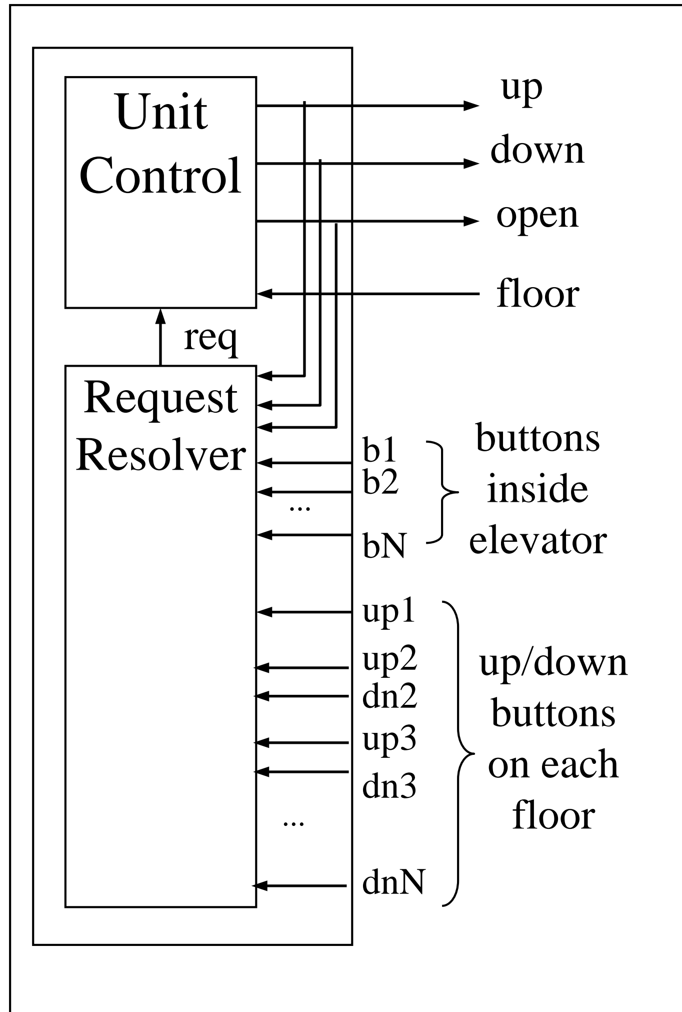
Rowan University

An Elevator Example

Software implementation example of FSM

- An elevator example:
  - Move the elevator either up or down to reach the requested floor. Once at the requested floor, open the door for at least 10 seconds, and keep it open until the requested floor changes. Ensure the door is never open while moving. Do not change directions unless there are no higher requests when moving up or no lower requests when moving down.

An Elevator Example



States:

Idle, GoingUp, GoingDn, DoorOpen

Events/Transitions:

req > floor, req = floor, req < floor

Inputs:

req, floor

Actions that occur in each state (outputs):

Up (u), down (d), door_open (o), start_timer (t)

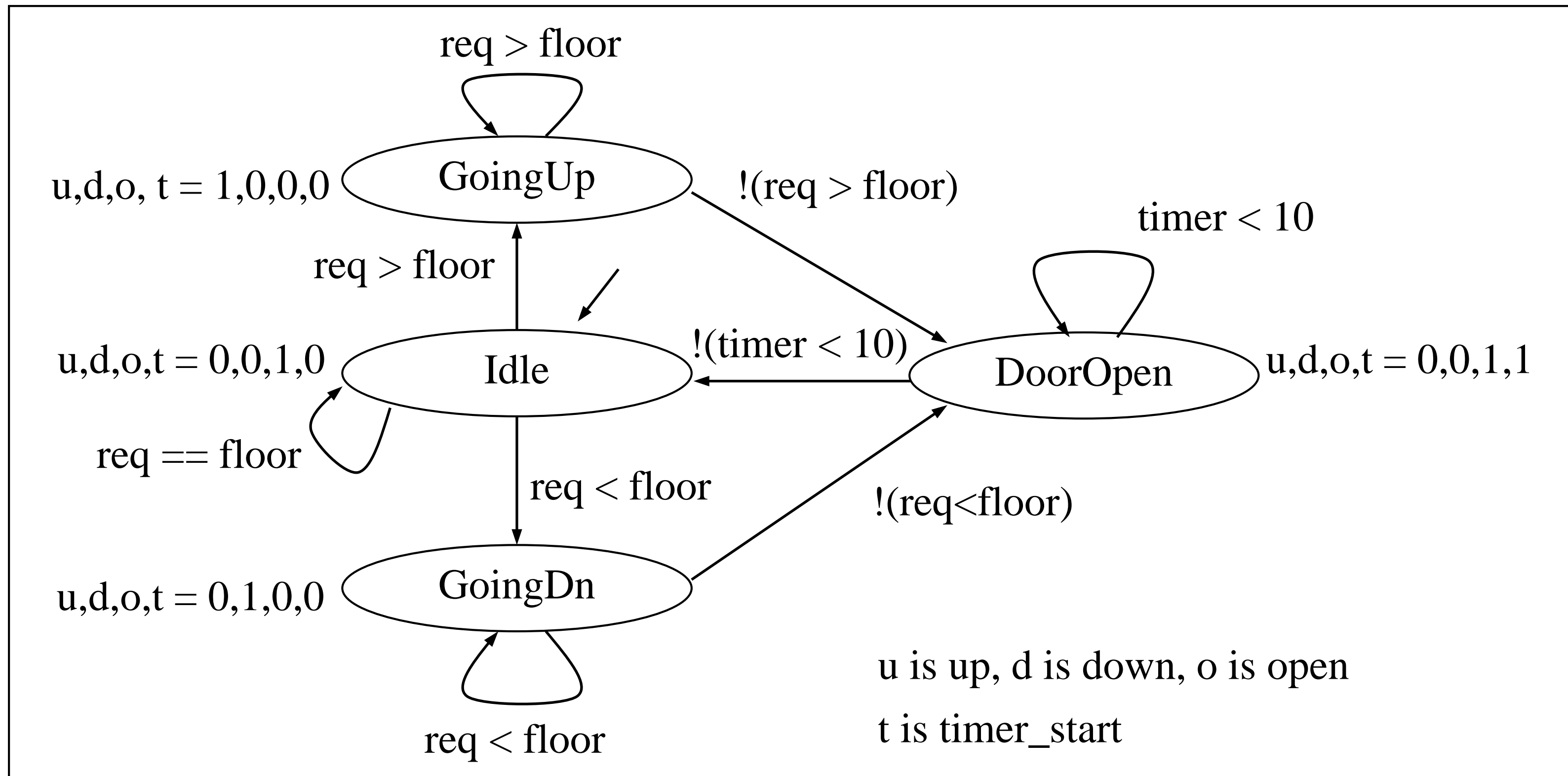
E.g., In the *GoingUp* state, u,d,o,t = 1,0,0,0.

In door_open state, u,d,o,t=0,0,1,1.

- Simple elevator controller
  - Request Resolver resolves various floor requests into single requested floor
  - Unit Control moves elevator to this requested floor

An Elevator Example

***UnitControl* process using a state machine**



Rowan University

An Elevator Example

Capturing *UnitControl* state machine in C

- Enumerate all states (#define)
- Declare state variable initialized to initial state (IDLE)
- Single switch statement branches to current state's case
- Each case has actions
  - up, down, open, timer_start
- Each case checks transition conditions to determine next state
  - if(...) {state = ...;}

```
#define IDLE 0
#define GOINGUP 1
#define GOINGDN 2
#define DOOROPEN 3
void UnitControl() {
  int state = IDLE;
  while (1) {
    switch (state) {
      IDLE: up=0; down=0; open=1; timer_start=0;
        if   (req==floor) {state = IDLE;}
        if   (req > floor) {state = GOINGUP;}
        if   (req < floor) {state = GOINGDN;}
        break;
      GOINGUP: up=1; down=0; open=0; timer_start=0;
        if   (req > floor) {state = GOINGUP;}
        if   (!(req>floor)) {state = DOOROPEN;}
        break;
      GOINGDN: up=0; down=1; open=0; timer_start=0;
        if   (req < floor) {state = GOINGDN;}
        if   (!(req<floor)) {state = DOOROPEN;}
        break;
      DOOROPEN: up=0; down=0; open=1; timer_start=1;
        if (timer < 10) {state = DOOROPEN;}
        if (!(timer<10)){state = IDLE;}
        break;
    }
  }
}
```


Rowan University

General sequential programming template for FSM

```
#define S0   0
#define S1   1
...
#define SN   N
void StateMachine() {
  int state = S0; // or whatever is the initial state.
  while (1) {
    switch (state) {
      S0: outputs=…; //for Moor outputs. Put Mealy outputs to individual transition condition below
        if( T0' s condition is true ) //whatever happens when the transition is true
               {state = T0' s next state; actions=…; }
        if( T1' s condition is true ) {state = T1' s next state; actions=…; }
        ...
        if( Tm' s condition is true ) {state = Tm' s next state; actions=…; }
        break;
      S1:
        …
        break;
      ...
      SN:
        …;
        break;
    }
  }
}
```


Rowan University

Steps to Design FSM

1. Understand the system
2. Identify the states & initial state
3. Identify the events/transitions/inputs
4. Identify outputs
5. Draw system interface (optional)
6. Draw state diagram/state table

Finite Automata

- A <u>state diagram</u> is a graphical representation of a <u>state machine</u>.
- The origin of FSMs is <u>finite automata</u>.
- Finite automata are primarily used in parsing for recognizing languages.
- A difference between finite automata and FSMs is that actions may be related to FSMs. The role of actions is to generate output.
- Original Finite Automata definition do not have output function. If an automaton generates output, output function can be added.

Rowan University

Formal Definition of FSM

A finite automaton (FA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- $Q = \{q_1, q_2, \ldots, q_m\}$ is a non-empty finite set of states;
- $\Sigma = \{e_1, e_2, \ldots, e_n\}$ is a non-empty finite set of events/alphabet;
- $\delta: Q \times \Sigma \rightarrow Q$ is a state transition partial function in which "partial function" means that the function may not be defined for all pairs in $Q \times \Sigma$;
- q_0 is an initial (or start) state;
- F is a set of final states. F could be empty set.

Rowan University

Math Foundation of DES

<u>Function:</u> it is a mapping from a set to another.

Examples (continuous variables):

- e.g., f: R→R can define a function (from real number to real number)

  $f(x)=y$; $x \in R$ and $y \in R$

- f: [0, 24]→[0, 24] defined as follows:

$$f(t)= \begin{cases} t-3, & 24 \geq t \geq 3 \\ 24-t, & 3 \geq t \geq 0 \end{cases}$$

Math Foundation of DES

Function - Discrete cases:

Assume $D=\{d_1, d_2, d_3\}$ and $E=\{e_1, e_2\}$.

- $f1{:}D \rightarrow N$ can define a mapping from D to natural number set, or a natural number vector.

  e.g., defining $f1(d_1)=1$, $f1(d_2)=3$, and $f1(d_3)=2$, we have

$$f1= \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix} = (1\ 3\ 2)^T$$

- $f2{:}D \rightarrow E$ and define $f2(d_1)=e_1$, $f2(d_2)=e_1$, and $f2(d_3)=e_2$, i.e.,

$$f2= \begin{pmatrix} e_1 \\ e_1 \\ e_2 \end{pmatrix} = (e_1\ e_1\ e_2)^T$$

Math Foundation of DES

Function - Discrete cases:

Assume $Q=\{q_1, q_2, q_3\}$ and $E=\{e_1, e_2\}$.

- $f: Q\times E\rightarrow N$ can define a mapping from $Q\times E$ to natural number set, or a natural number matrix.

  e.g., defining $f(q_1,e_1)=1$, $f(q_1,e_2)=0$, $f(q_2,e_1)=2$,

  $f(q_2,e_2)=0$, $f(q_3,e_1)=0$, and $f(q_3,e_2)=2$, we have

$$f = \begin{array}{c} \\ q_1 \\ q_2 \\ q_3 \end{array} \begin{array}{c} \begin{array}{cc} e_1 & e_2 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 2 & 0 \\ 0 & 2 \end{pmatrix} \end{array} \quad \text{Or} \quad f = \begin{pmatrix} 1 & 0 \\ 2 & 0 \\ 0 & 2 \end{pmatrix}$$

The dimension of the function is 3×2 which is (dimension of the 1st set) $\times$ (dimension of 2nd set)

Rowan University

Math Foundation of DES

Function - Discrete cases:

Assume $Q=\{q_1, q_2, q_3\}$ and $E=\{e_1, e_2\}$.

- $\delta: Q\times E \rightarrow Q$ can define a mapping from $Q\times E$ to Q

  - this basically defines a table.

  e.g., defining $\delta(q_1,e_1)=q_2$, $\delta(q_1,e_2)=q_3$, $\delta(q_2,e_1)=q_1$, and $\delta(q_3,e_2)=q_2$ and others are not defined, we have

$\delta =$

	e_1	e_2
q_1	q_2	q_3
q_2	q_1	-
q_3	-	q_2

Or $\delta =$

q_2	q_3
q_1	-
-	q_2

If Q is a set of states of a FSM and E is a set of events, $\delta: Q\times E \rightarrow Q$ is the *state transition function*.

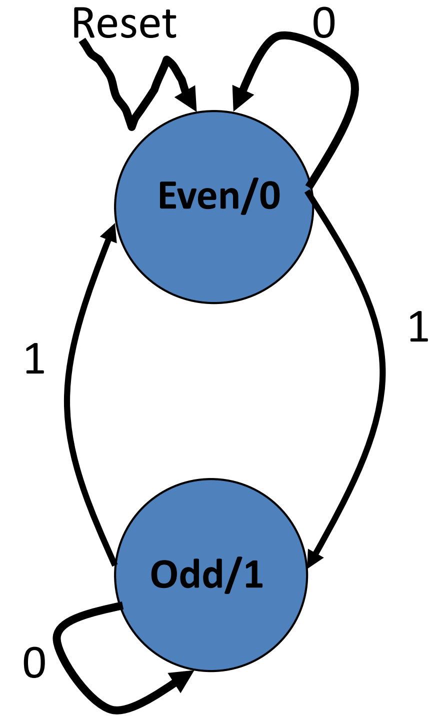
Rowan University

Formal Definition - Examples

Parity Checker

A=(Q, Σ, q0, δ, F) where

- *Q*={Even, Odd}, two-state set,
- Σ={0, 1}, two-event set,
- q0 = Even,
- δ(Even, 0)=Even, δ(Even, 1)=Odd, δ(Odd, 0)=Odd, δ(Odd, 1)=Even
- *F* = Φ, the empty set.



Note:

$$\delta = \begin{pmatrix} \text{Even} & \text{Odd} \\ \text{Odd} & \text{Even} \end{pmatrix}$$ or

	0	1
Even	Even	Odd
odd	Odd	Even

Rowan University

Formal Definition - Examples

Traffic controller:

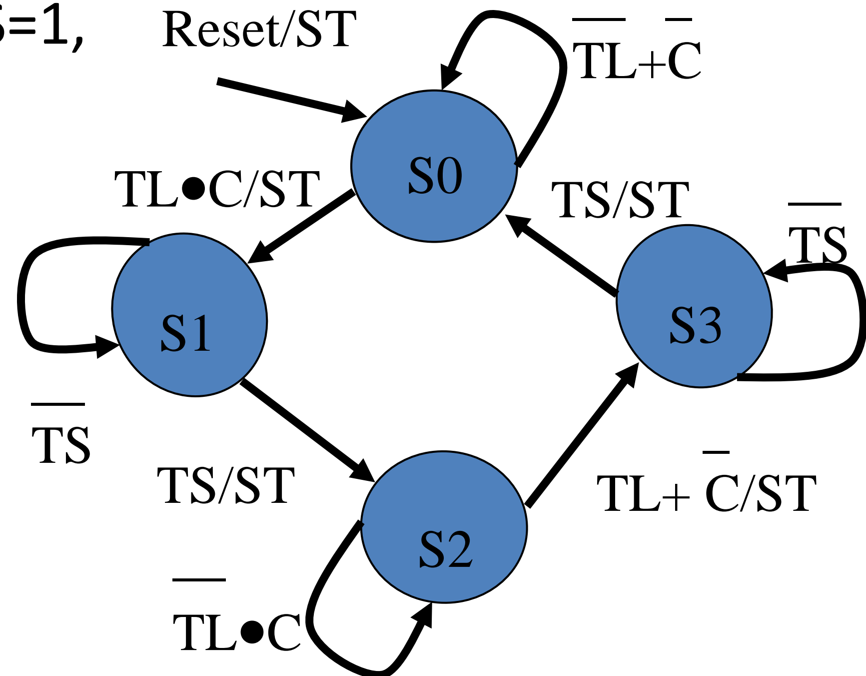
A=(Q, Σ, q0, δ, F) where

- *Q*={S0, S1, S2, S3}
- Σ={e1, e2, e3} where

  e1 occurs if TL∧C=1, e2 if TS=1, and e3 if TL∨C' =1
- q0 = S0,
- *F* = Φ

$\delta =$

	e_1	e_2	e_3
S_0	S_1	-	-
S_1	-	S_2	-
S_2	-	-	S_3
S_3	-	S_0	-



Rowan University

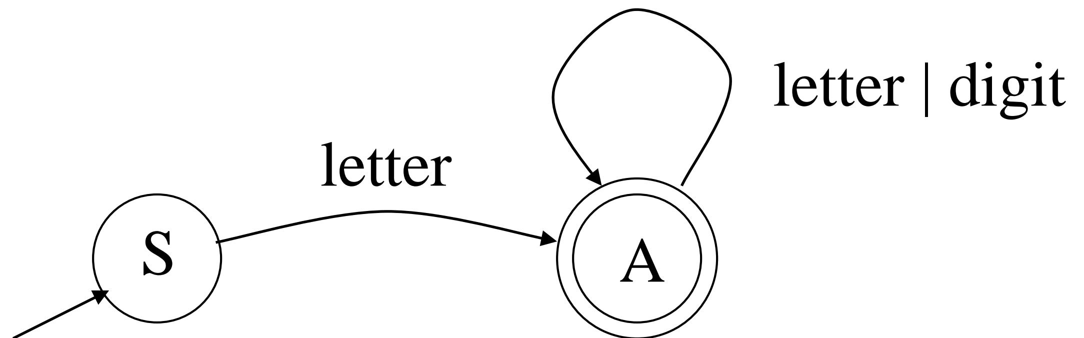
Finite Automation

- Finite automata are primarily used in parsing for recognizing languages.
- Finite automata can be used to describe formal language.
- A **formal language** is a set of *words*, i.e. finite strings of letters, symbols, or token. The set from which these letters are taken is called *alphabet* over which the language is defined.
- Input strings belonging to a given language should turn an automaton to final states and all other input strings should turn this automaton to states that are not final.
- Automata play a major role in *compiler* design and parsing.

Rowan University

Example of FSM for Compiler

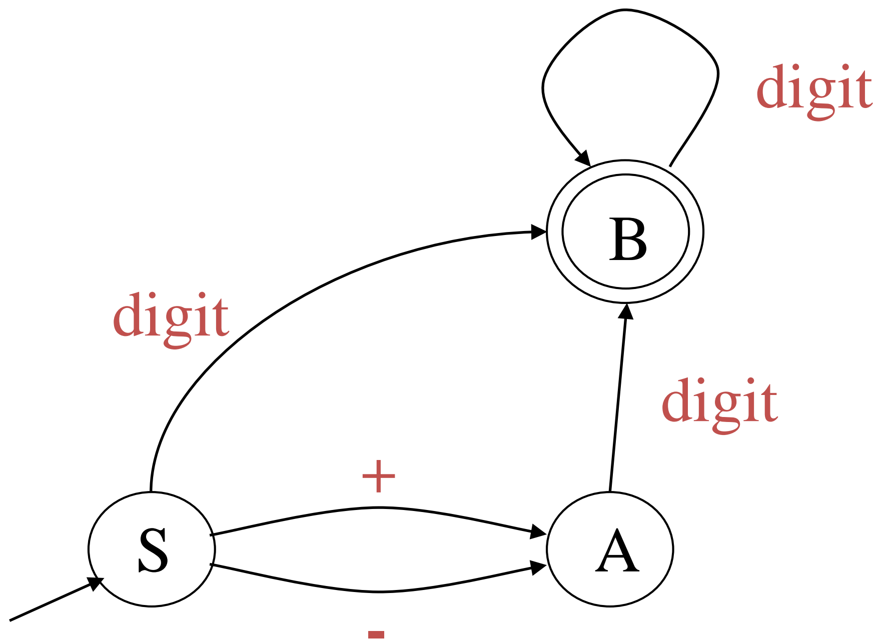
- A FSM is similar to a compiler in that:
  - A compiler recognizes legal *programs* in some (source) language.
  - A finite-state machine defines legal *strings*
- Example: Pascal Identifiers
  - sequences of one or more letters or digits, starting with a letter:



Valid identifiers: x, MyID, mp3, X007…
Invalid identifiers: 123, a?, 5books…

Example: Integer Literals

- **FSM that accepts integer literals with an optional + or - sign:**



A string is accepted by a FSM if there exists a sequence of state transitions starting in the start state, ending in a final state, that consumes the entire string.

e.g., accepted: 15; +17; -1

Unaccepted: 17.5; a; 2b

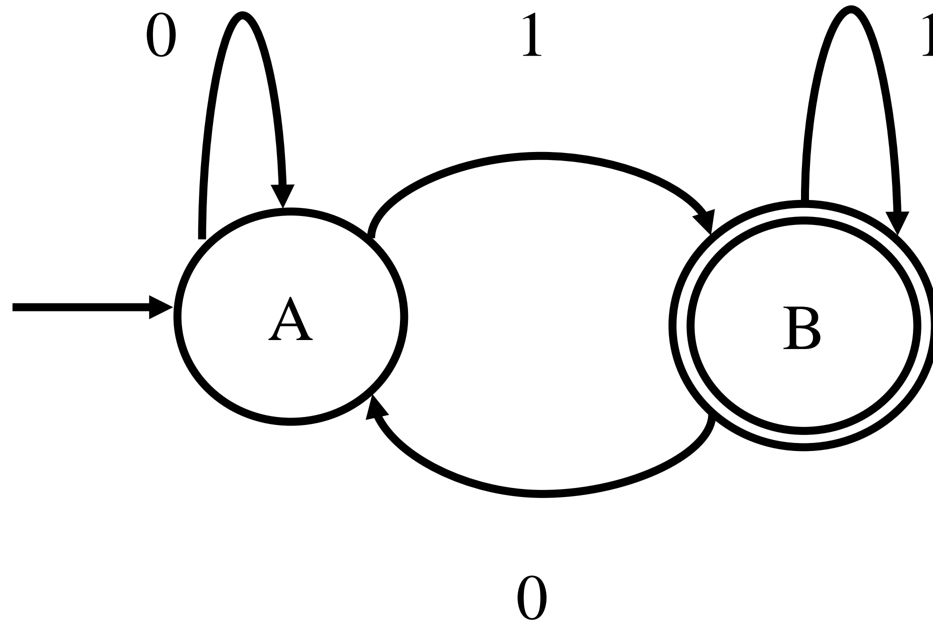
First symbol should be digit/+/-;

Everything going to B (final state) must be a digit.

Rowan University

FMS Example

Finite State Machine that accepts strings over alphabet {0,1} that end in 1:



The string could start from 0 or 1 and must end in 1 (everything going to final state must be 1)

Rowan University

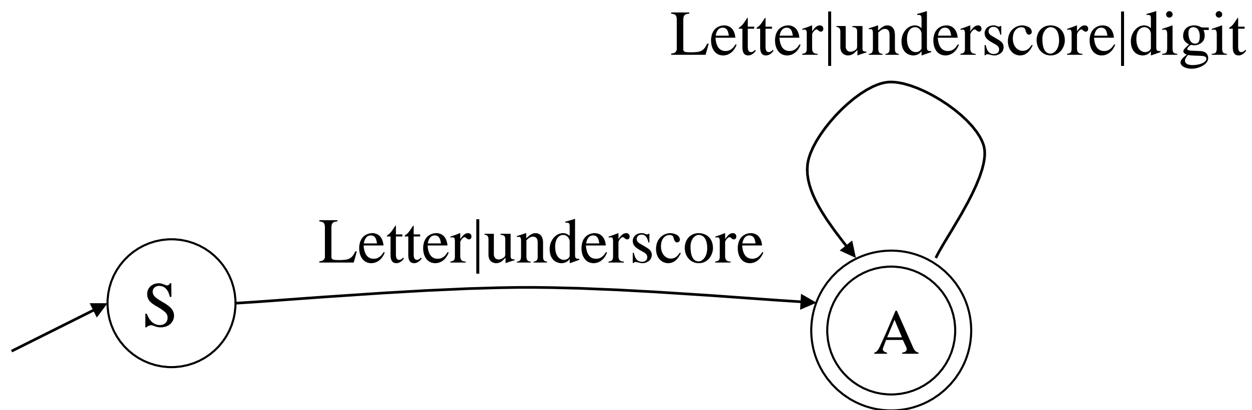
Exercise

- **Exercise 1:** Draw a finite-state machine that accepts the identifiers
  - one or more letters, digits, or underscores, starting only with a letter or an underscore.

Rowan University

Exercise

- **Exercise 1:** Draw a finite-state machine that accepts the identifiers
  - one or more letters, digits, or underscores, starting only with a letter or an underscore.



Rowan University

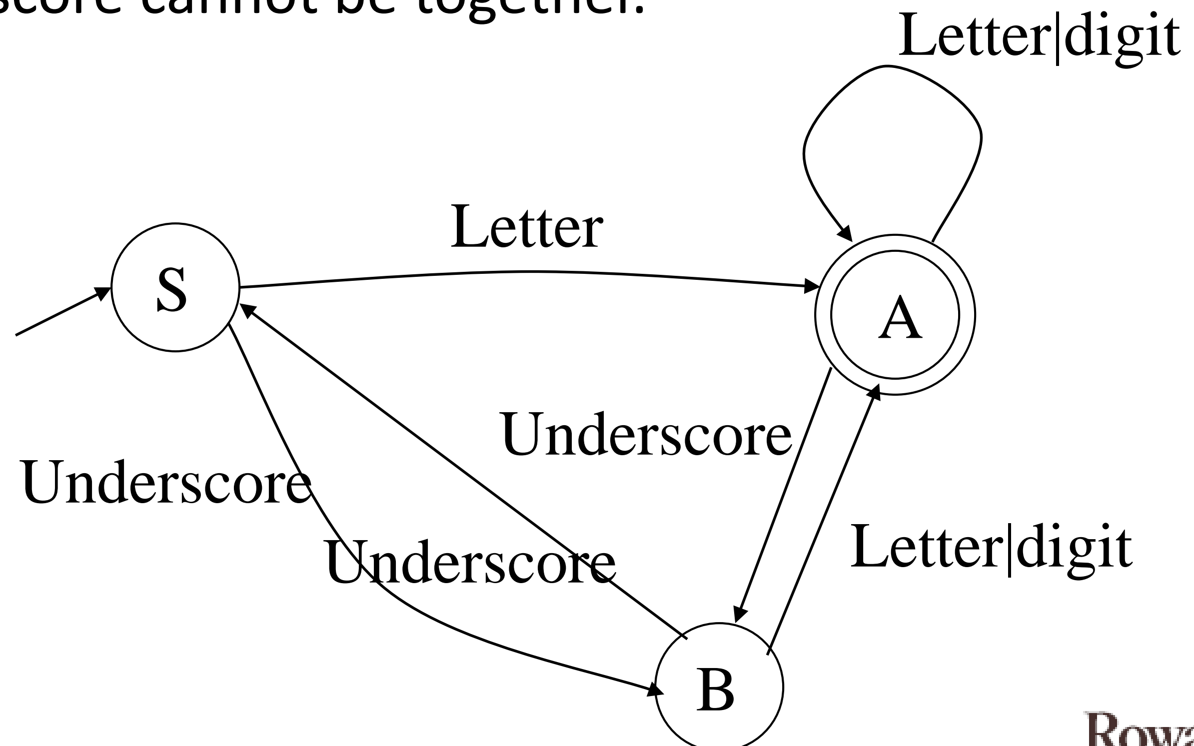
Exercise

- **Exercise 2:** Draw a finite-state machine that accepts the identifiers:
  - one or more letters, digits, or underscores, starting only with a letter or an underscore.
  - do not end with an underscore.
  - Two underscores cannot be together.

Rowan University

Exercise

- **Exercise 2:** Draw a finite-state machine that accepts the identifiers:
  - one or more letters, digits, or underscores, starting only with a letter or an underscore.
  - do not end with an underscore.
  - Two underscore cannot be together.



Rowan University

State Path & Event Path

Given an automaton, $q_0 q_{i1} \ldots q_{ik}$ is a state path if there is an event path $e_{i1} \ldots e_{i1}$ such that all the below equations hold:

$\delta(q_0, e_{i1}) = q_{i1}$

$\delta(q_{i1}, e_{i2}) = q_{i2}$

…

$\delta(q_{i,k-1}, e_{ik}) = q_{ik}$

Language

1. Formal language L is the set of all possible event paths in an automaton $A=(\Sigma, Q, q_0, \delta, F)$.
2. A is called a generator of L over the alphabet Σ.
3. Marked language $L_m \subset L$ is a set of all possible event paths whose corresponding state paths' last element is in F.

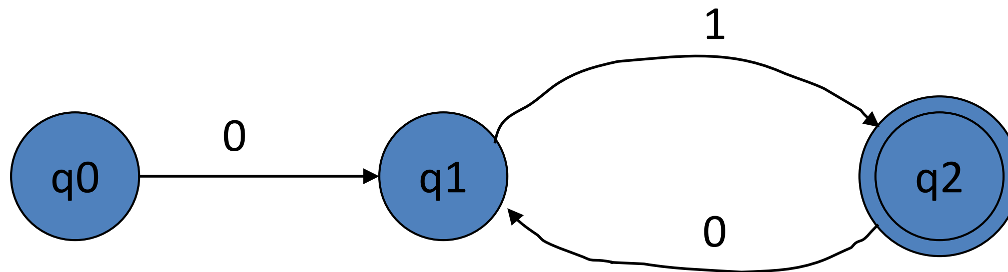
Rowan University

Languages: Example

$$\Sigma = \{0,1\} \qquad \delta(q_0,0) = q_1 \qquad F = \{q_2\}$$

$$Q = \{q_0, q_1, q_2\} \qquad \delta(q_1,1) = q_2$$

$$\delta(q_2,0) = q_1$$



$L = \{\varepsilon, 0, 01, 010, 0101, 01010, 010101, \ldots\}$

$L_m = \{01, 0101, 010101, \ldots\} = \{(01)^i, i=0, 1, 2, \ldots\}$

Rowan University

Formal Language Representation of a DES

- The behavior of a DES is constituted from events. The set of possible events is known as the *alphabet*. In any given time line, the DES executes a sequence of events.
- DES is represented by a set of strings, referred to as the language (denoted L). Some strings in L are marked and these constitute a marked language Lm. A marked string represents a properly completed (desired) behavior, i.e., one that fulfills control objectives.