CS221: Digital Design

http://jatinga.iitg.ernet.in/~asahu/cs221

FSM: Examples, Optimization and State Encoding

A. Sahu

Dept of Comp. Sc. & Engg.

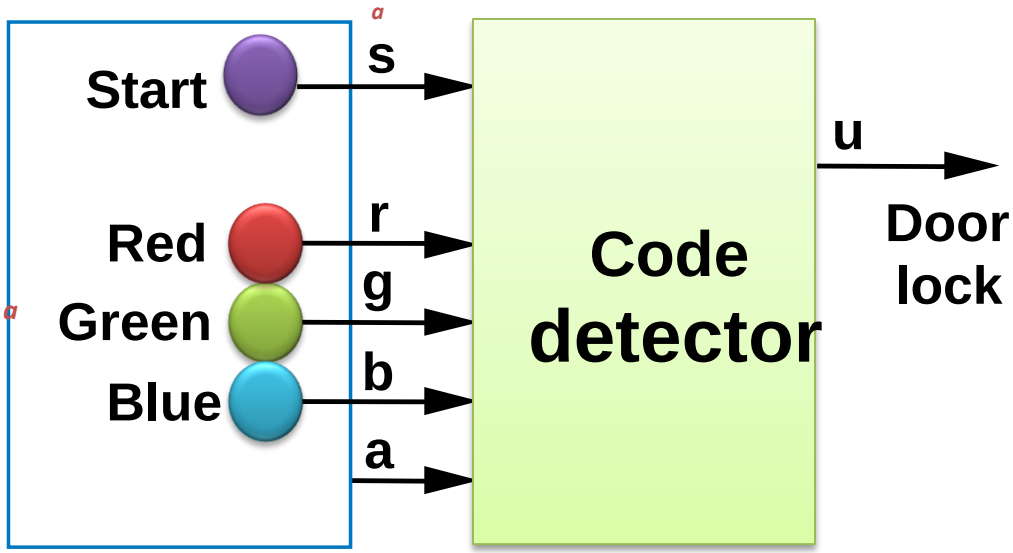
Indian Institute of Technology Guwahati

Outline

- FSM Examples
- Moore Vs Mealy Machines
- FSM State Optimization : RM and IC
- FSM State Encoding
- FSM + Data Path
- ASM

FSM Example 5 : Code Detector

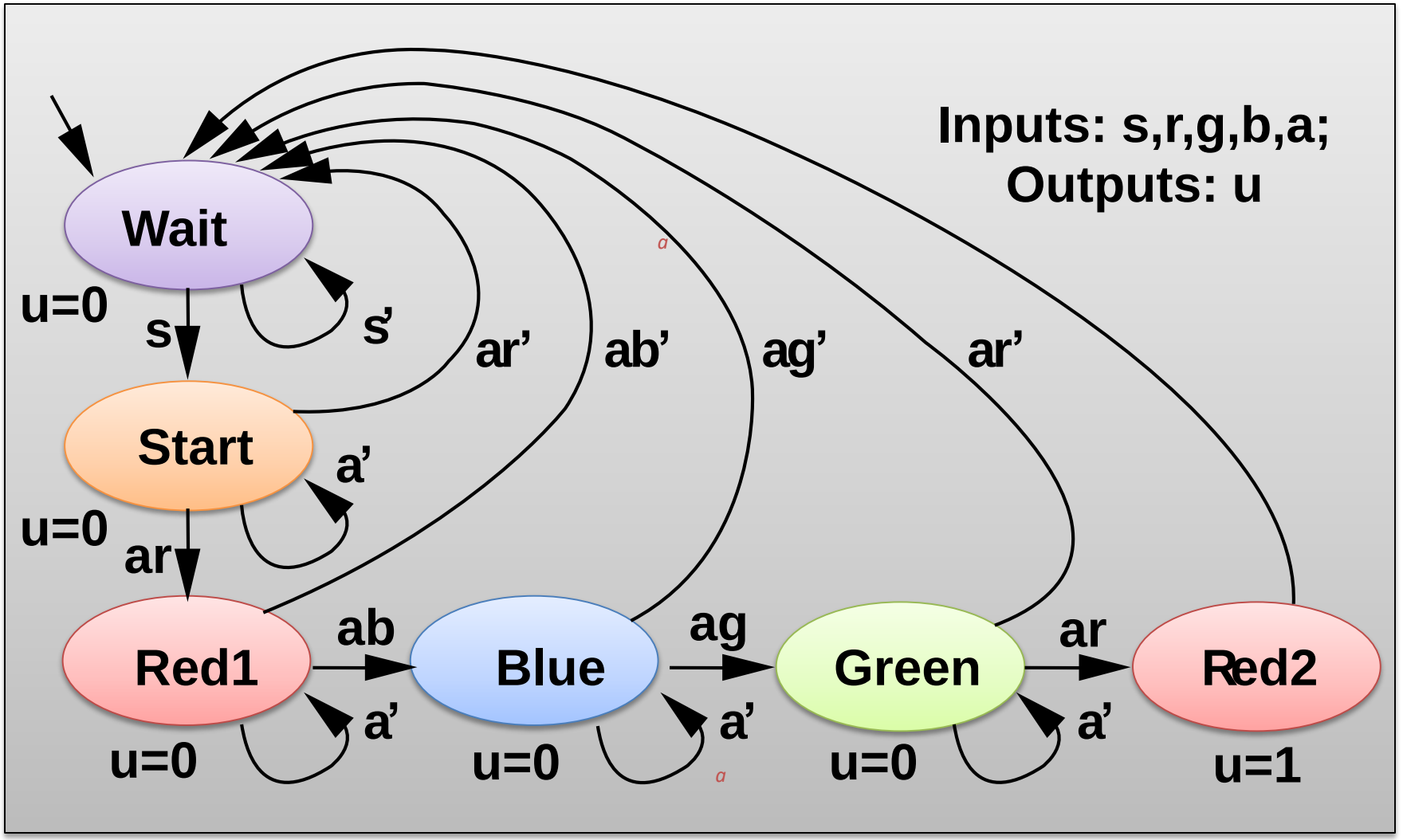
- Unlock door (u=1) only when buttons pressed in sequence:
  - **start, then red, blue, green, red**
- Input from each button: *s, r, g, b*
  - Also, output *a* indicates that some colored button pressed



FSM Example 5: Code Detector

- Wait for start (s=1) in “Wait”,
- **Once started (“Start”)**
  - If see red, go to “Red1”
  - Then, if see blue, go to “Blue”, Then, if see green, go to “Green”, Then, if see red, go to “Red2”
  - In that state, open the door (u=1)
  - Wrong button at any step, return to “Wait”

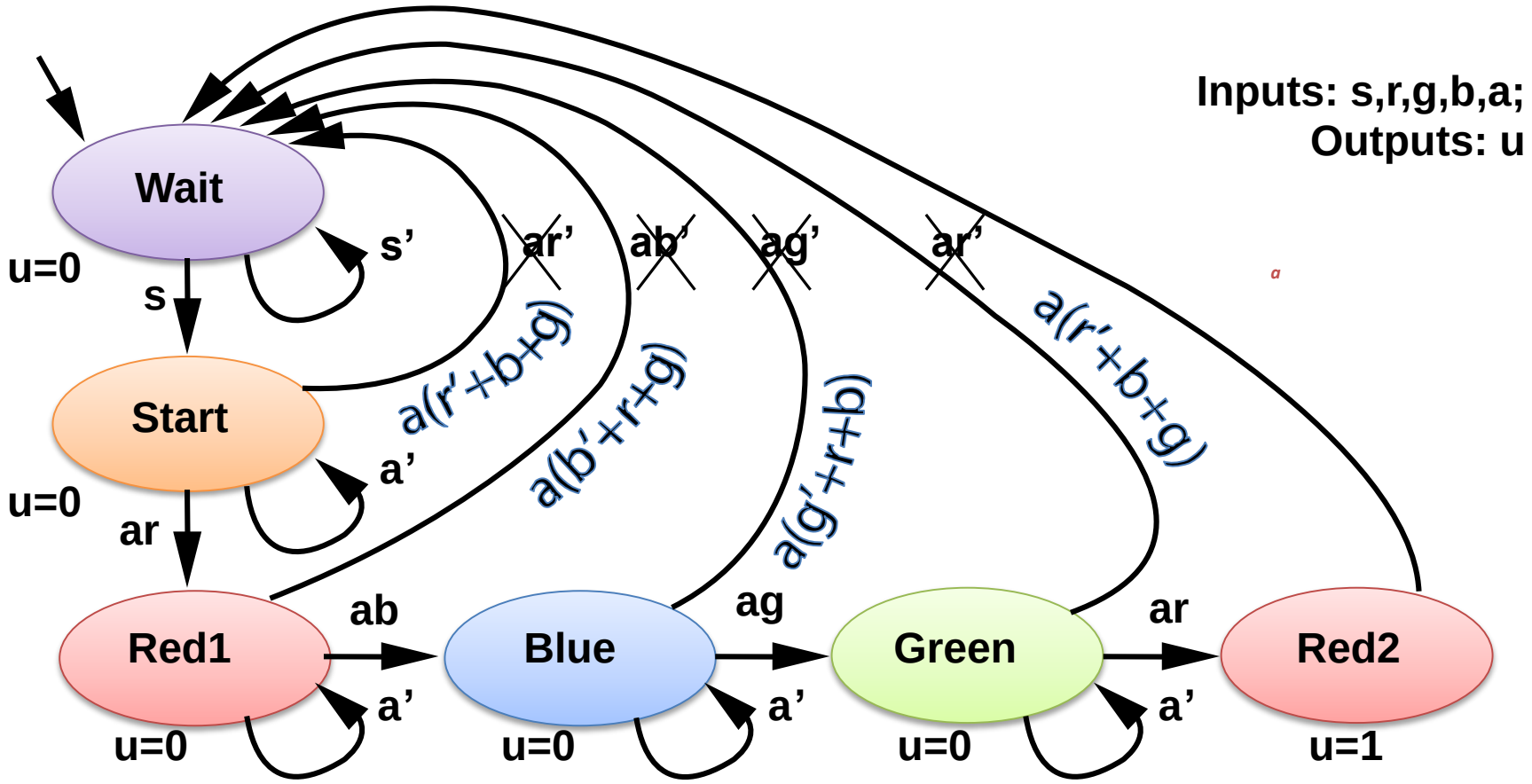
FSM Example 5 : Code Detector



Q: Can you trick this FSM to open the door, without knowing the code?

A: Yes, hold all buttons simultaneously

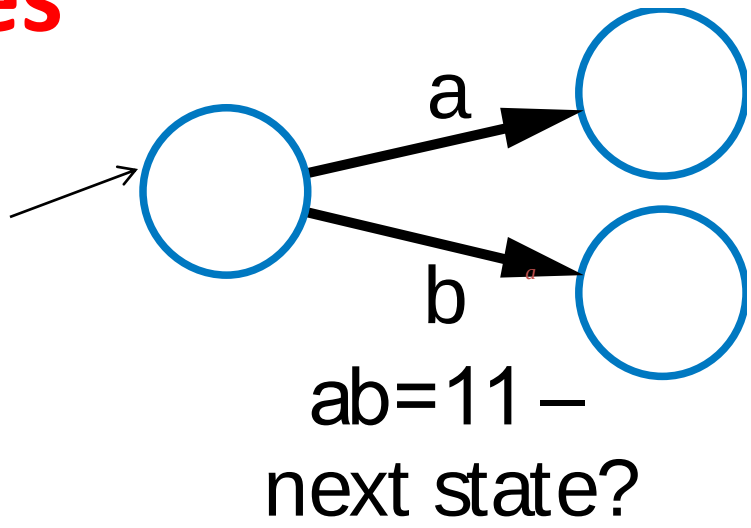
Improve FSM for Code Detector

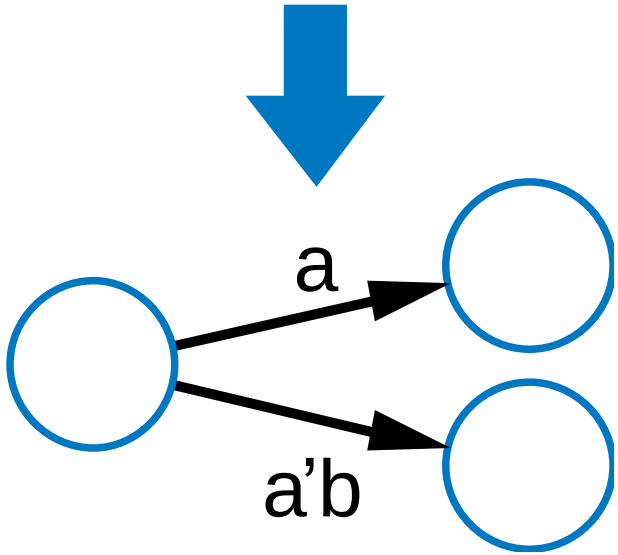


- New transition conditions detect if wrong button pressed, returns to "Wait"
- FSM provides formal, concrete means to accurately define desired behavior

Common Pitfalls Regarding Transition Properties

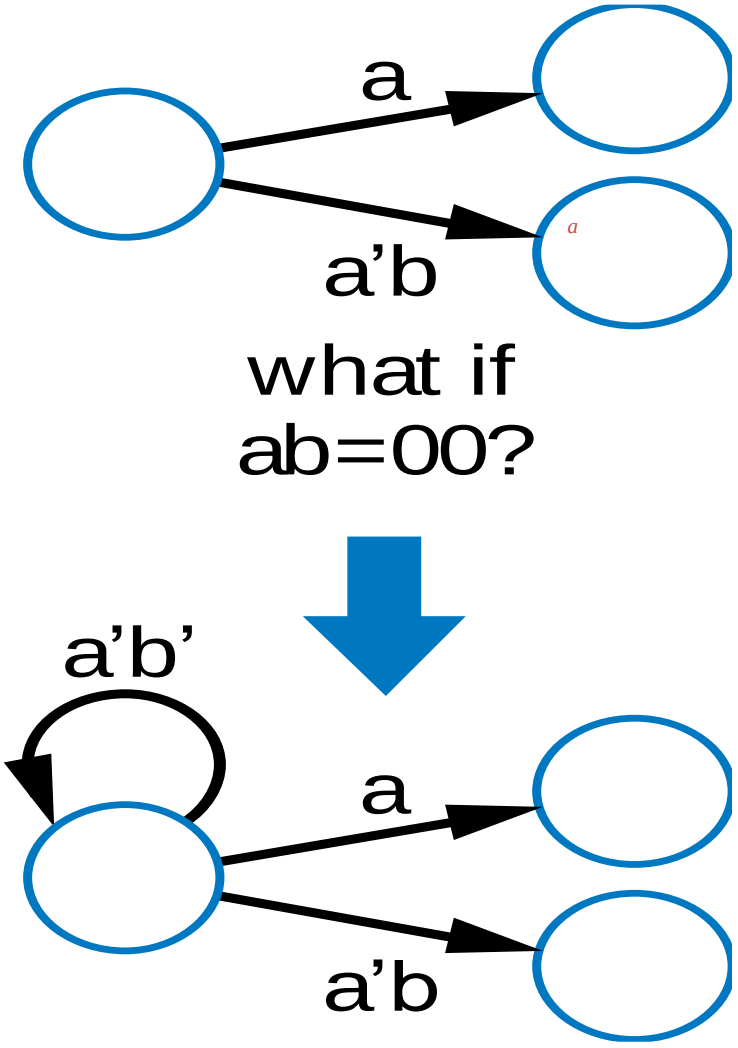
- *Only* one condition should be true
  - For all transitions leaving a state
  - Else, which one?





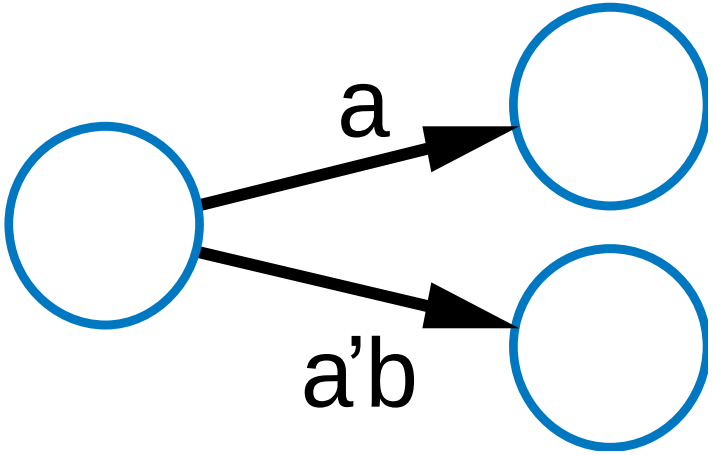
Common Pitfalls Regarding Transition Properties

- *One* condition must be true
  - For all transitions leaving a state
  - Else, where go?



Verifying Correct Transition Properties

- Can verify using Boolean algebra
  - Only one condition true: AND of each condition pair (for transitions leaving a state) should equal 0
  - → proves pair can never simultaneously be true
  - Example

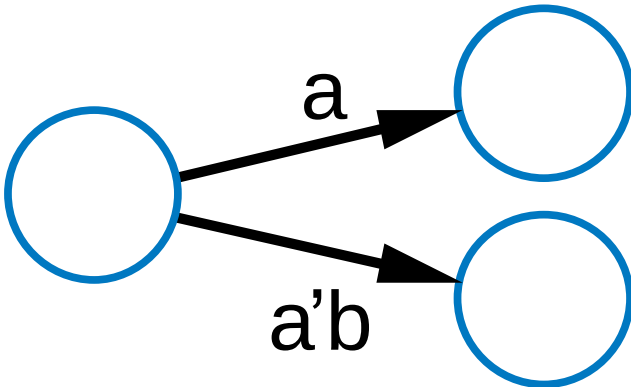


Answer:

a * a'b
= (a * a') * b
= 0 * b
= 0
OK!

Verifying Correct Transition Properties

- Can verify using Boolean algebra
  - One condition true: OR of all conditions of transitions leaving a state) should equal 1
  - → proves at least one condition must be true
  - Example



a + a'b
= a*(1+b) + a'b
= a + ab + a'b
= a + (a+a')b
= a + b

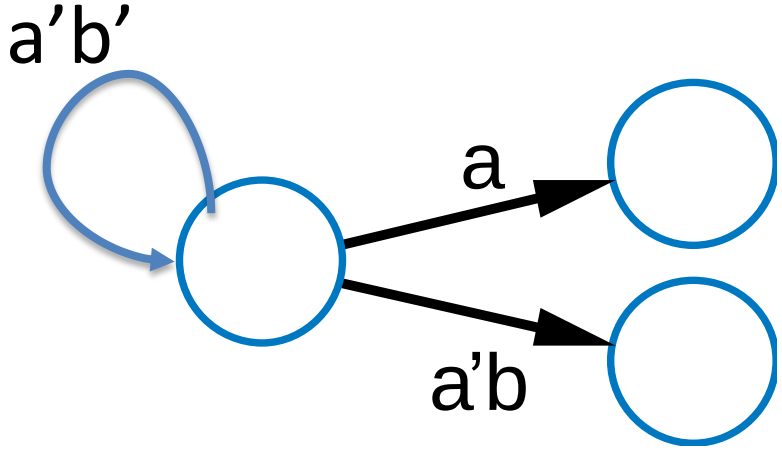
Fails! Might not be 1 (i.e., a=0, b=0)

Verifying Correct Transition Properties

- Can verify using Boolean algebra
  - Only one condition true
  - One condition true

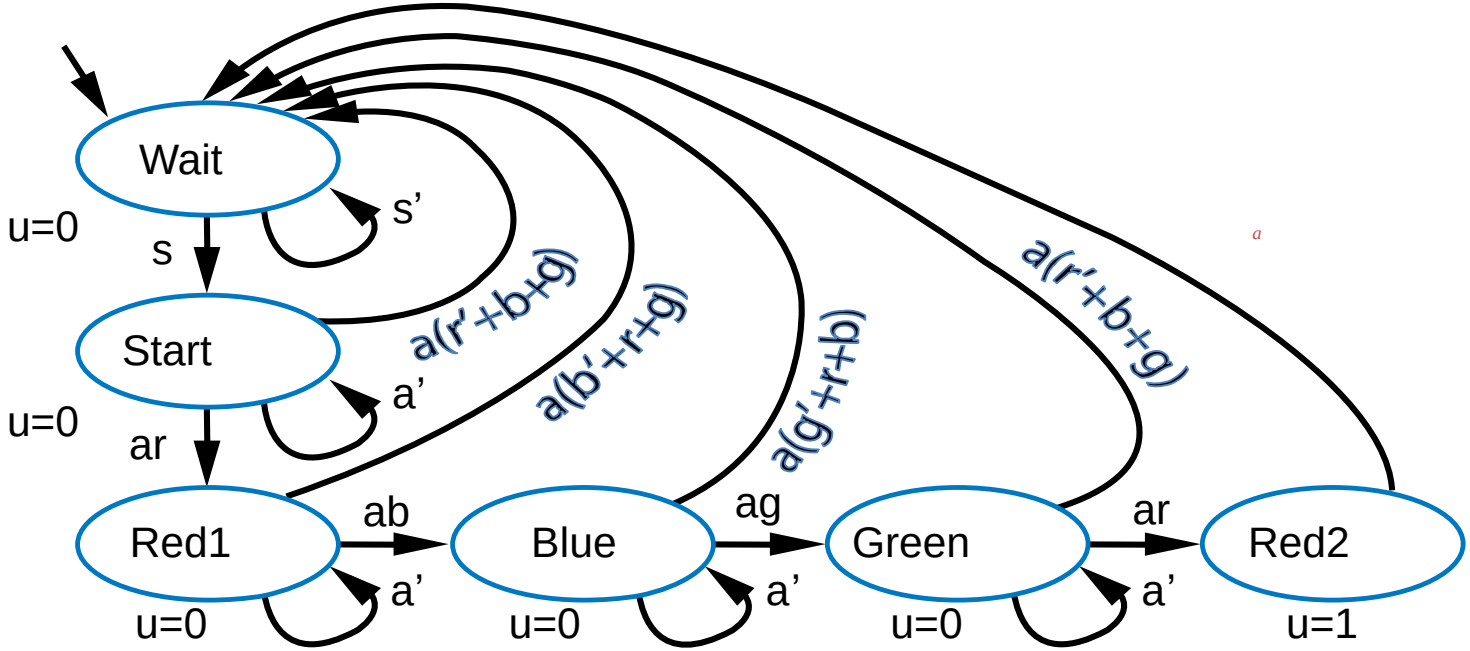
Q: For shown transitions, prove whether:

* Only one condition true (AND of each pair is always 0)
* One condition true (OR of all transitions is always 1)



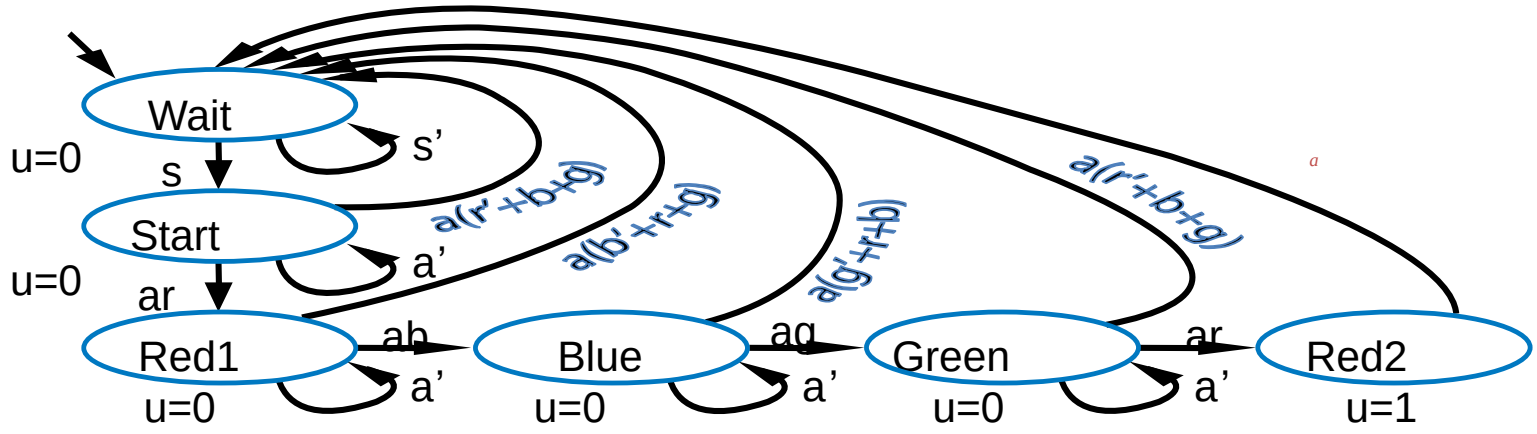
$$a + a'b + a'b' = 1$$

Evidence that Pitfall is Common



- Recall code detector FSM
  - We "fixed" a problem with the transition conditions
  - Do the transitions obey the two required transition properties?
    - Consider transitions of state *Start*, and the "only one true" property

Evidence that Pitfall is Common

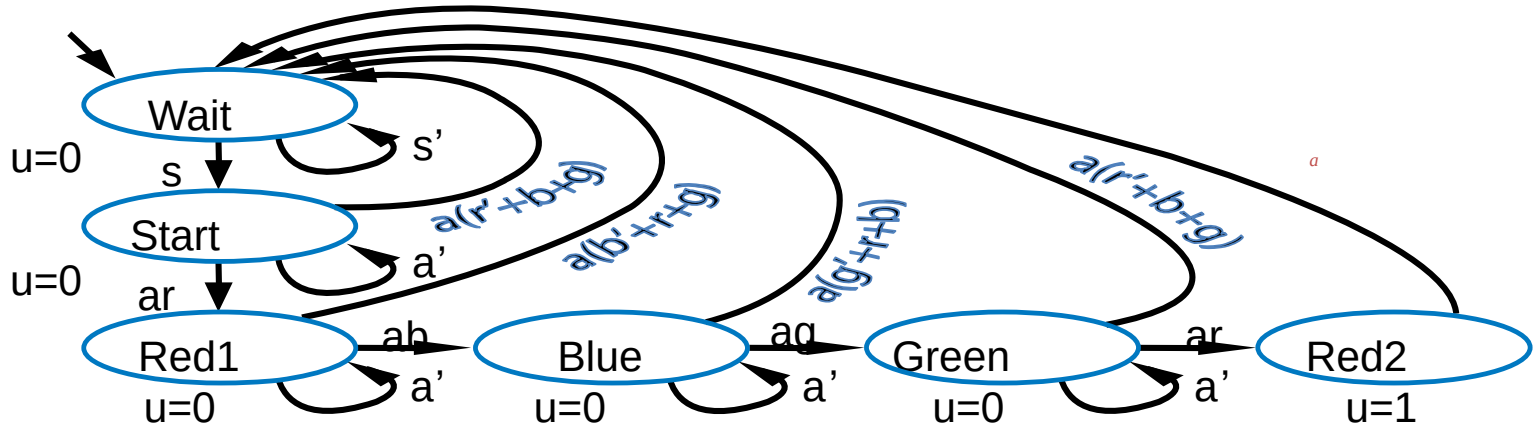


Consider transitions of state *Start*, and "only one true" property

ar * a'	a' * a(r'+b+g)	ar * a(r'+b+g)
= (a*a')r	= 0*r	= (a'*a)*(r'+b+g) = 0*(r'+b+g)
		= (a*a)*r*(r'+b+g) = *r*(r'+b+g)
= 0	**= 0**	= arr'+arb+arg
		= 0 + arb+arg
		= arb + arg
		= ar(b+g) ***// not ZERO***

Fails! Means that two of Start's transitions could be true

Evidence that Pitfall is Common



Consider transitions of state *Start*, and "only one true" property

Intuitively: press red and blue buttons at same time: conditions ar, and a(r'+b+g) will both be true. Which one should be taken?

Q: How to solve?

A: ar should be arb'g' (likewise for ab, ag, ar)

arb'g' * a(r'+b+g) = 0

FSM Types

- Two main types of FSMs
  - Moore FSM : output is only function of state
  - Mealy FSM: output is function of state and inputs

Set Theoretic Description

Moore Machine is an ordered quintuple

$$Moore = (\mathbf{S}, \mathbf{I}, \mathbf{O}, \delta, \lambda)$$

where

$\mathbf{S}$ = Finite set of states $\neq \Phi, \{s_1, s_2, \cdots, s_n\}$

$\mathbf{I}$ = Finite set of inputs $\neq \Phi, \{i_1, i_2, \cdots, i_m\}$

$\mathbf{O}$ = Finite set of outputs $\neq \Phi, \{o_1, o_2, \cdots, o_l\}$

δ = Next state function which maps $\mathbf{S} \times \mathbf{I} \rightarrow \mathbf{S}$

λ = Output function which maps $\mathbf{S} \rightarrow \mathbf{O}$

Set Theoretic Description

Mealy Machine is an ordered quintuple

$$\text{Mealy} = (\mathbf{S}, \mathbf{I}, \mathbf{O}, \delta, \beta)$$

where

$\mathbf{S} = \text{Finite set of states} \neq \Phi, \{s_1, s_2, \cdots, s_n\}$

$\mathbf{I} = \text{Finite set of inputs} \neq \Phi, \{i_1, i_2, \cdots, i_m\}$

$\mathbf{O} = \text{Finite set of outputs} \neq \Phi, \{o_1, o_2, \cdots, o_l\}$

$\delta = \text{Next state function which maps} \quad \mathbf{S} \times \mathbf{I} \rightarrow \mathbf{S}$

$\beta = \text{Output function which maps} \quad \mathbf{S} \times \mathbf{I} \rightarrow \mathbf{O}$

Clocked synchronous FSM

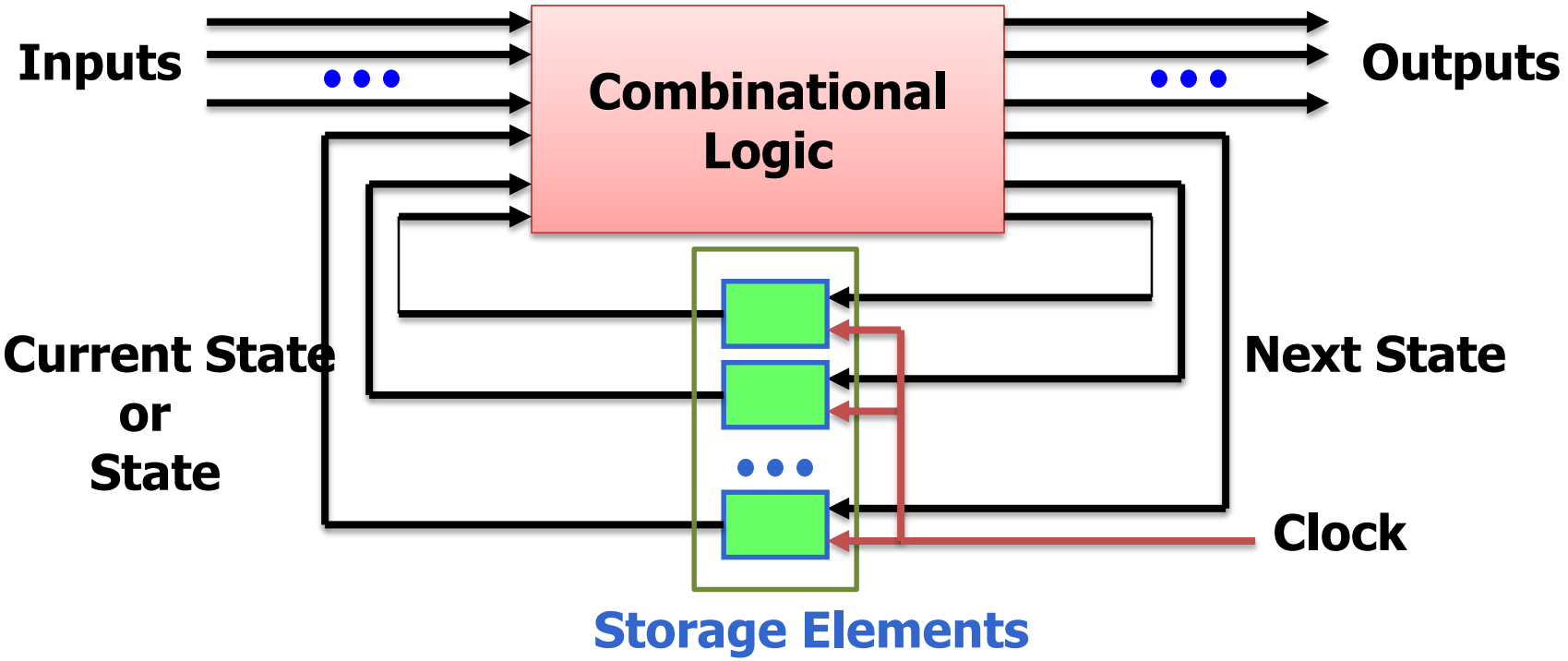
- **Clocked**
  - All storage elements employ a clock input (i.e. all storage elements are flip-flops)
- **Synchronous**
  - All of the flip flops use the same clock signal

Clocked synchronous FSM

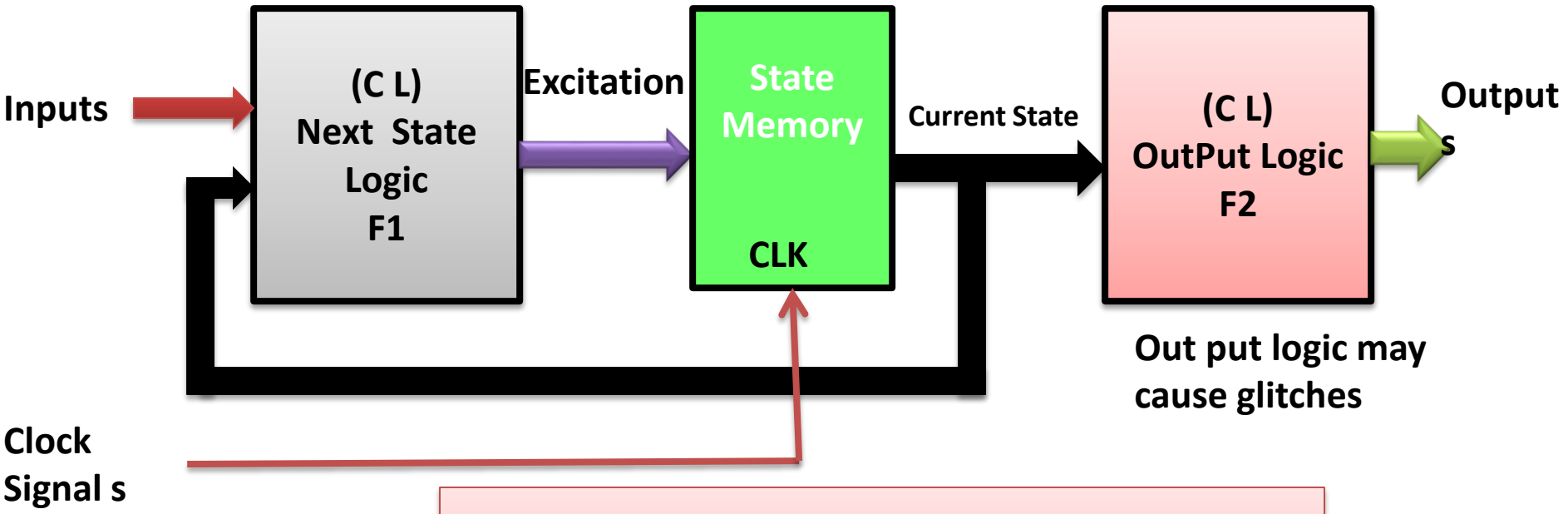
- **FSM**
  - State machine is simply another name for sequential circuits. Finite refers to the fact that the number of states the circuit can assume if finite
- **Async FSM:** A synchronous clocked FSM changes state only when a triggering edge (or tick) occurs on the clock signal

Clocked synchronous FSM structure

- **States**: determined by possible values in sequential storage elements
- **Transitions**: change of state
- **Clock**: controls when state can change by controlling storage elements



Moore machine

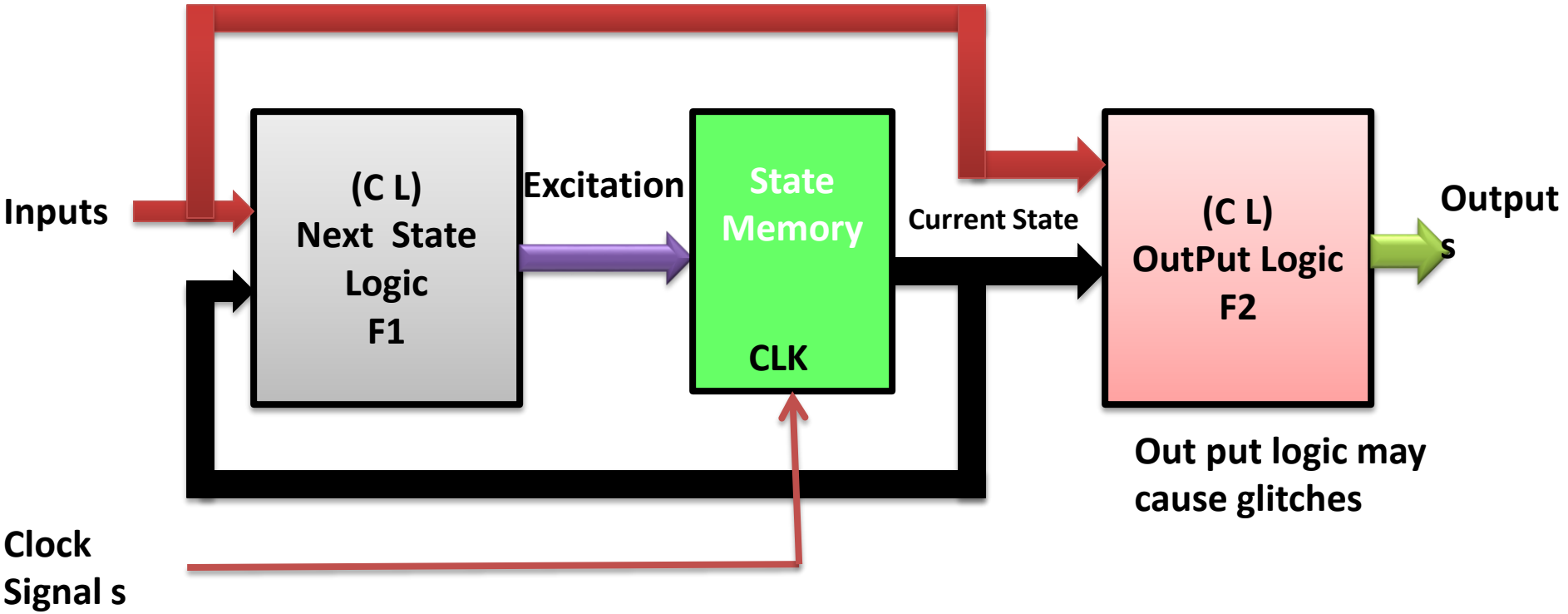


$$\text{Next state} = F_2(\text{current state, inputs})$$

$$\text{Output} = G_2(\text{current state})$$

Mealy machine

- A Much better name of State “Memory” is State Storage



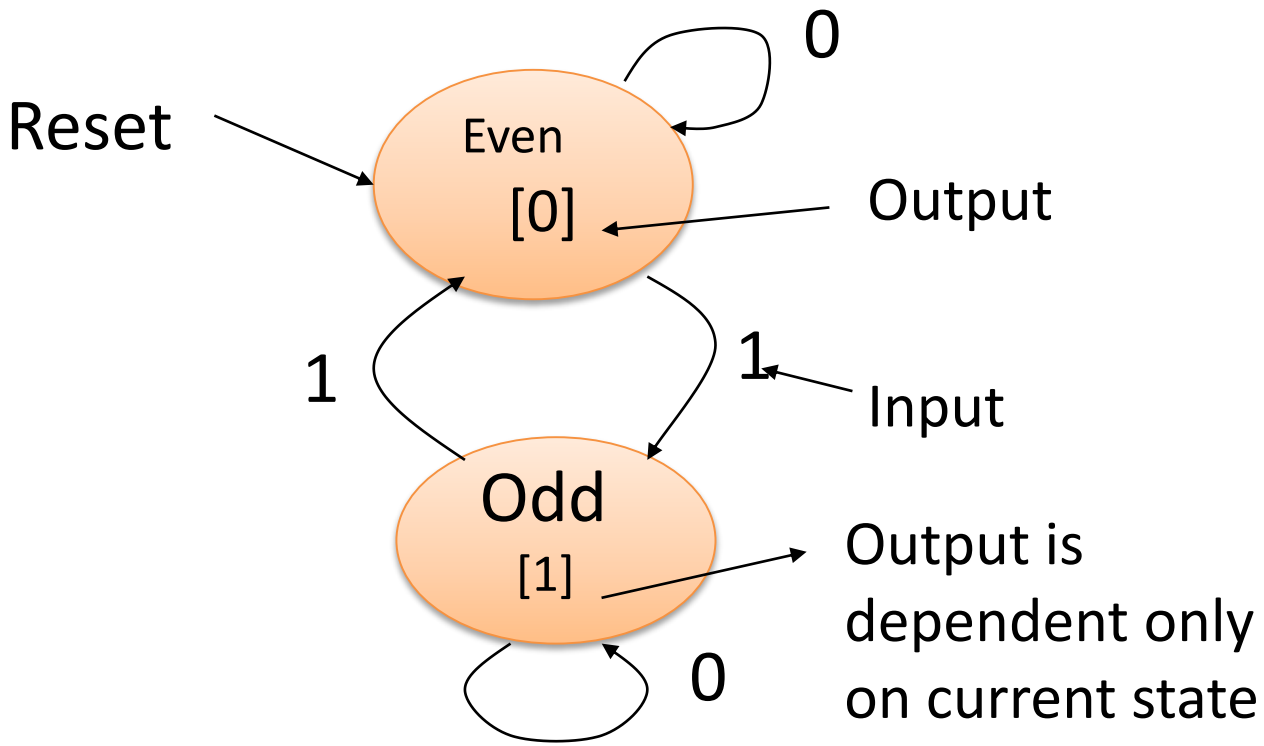
State Storage= Set of n FFs
2^n State can be stored

Next state = F_1(current state, inputs)

Output = F_2(current state, inputs)

Example of Moore & Melay Machine

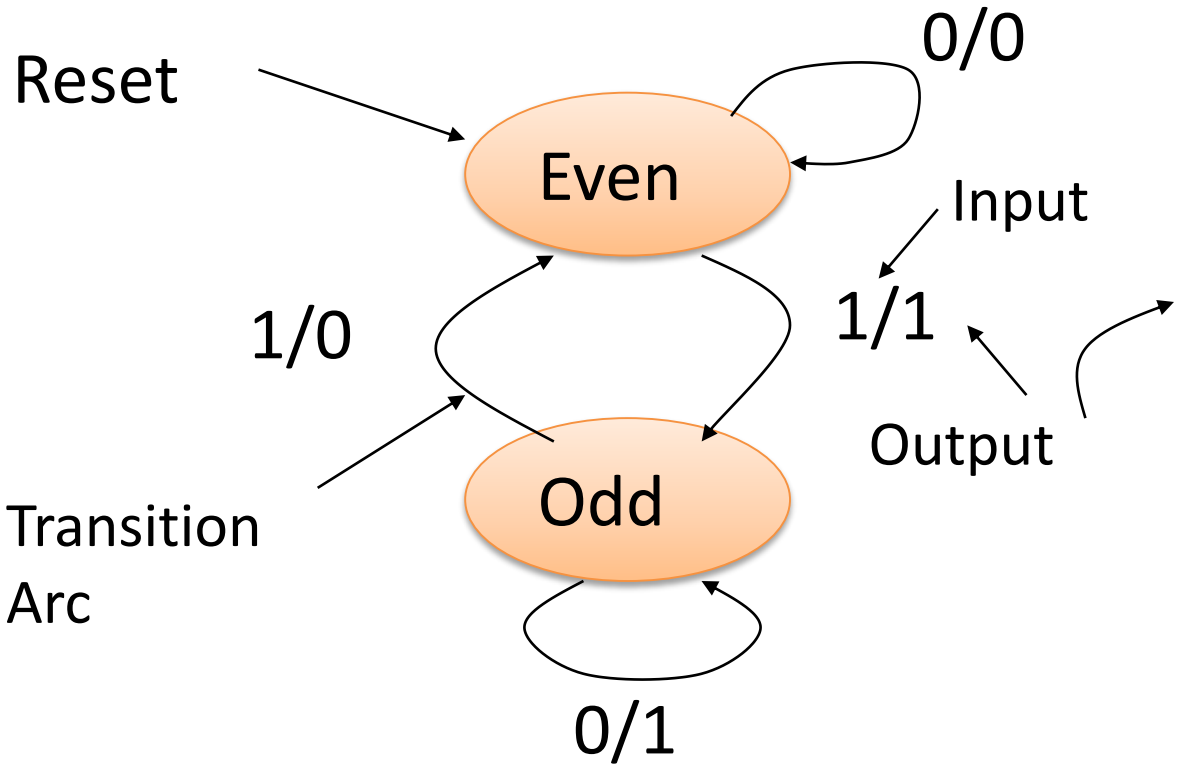
- Parity Checker Solution : (Moore)



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

Example of Moore & Mealy Machine

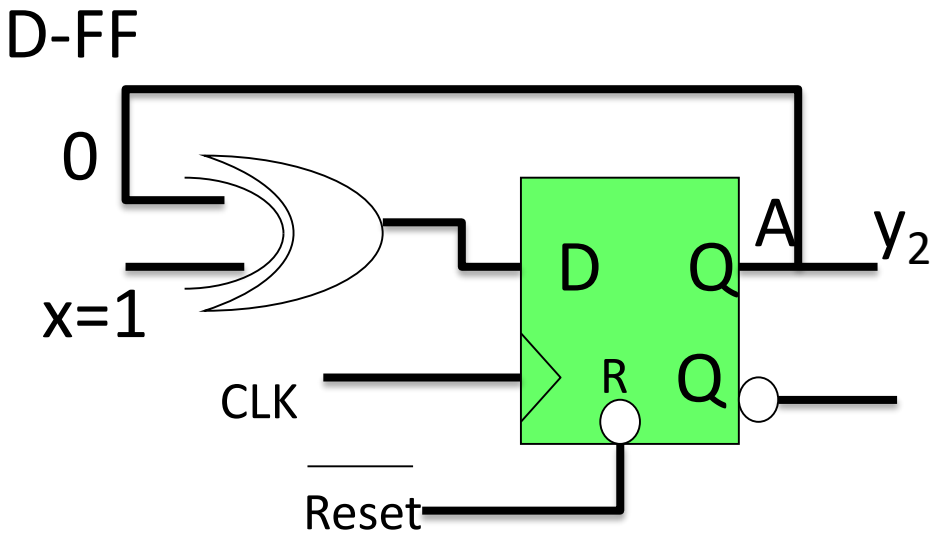
- Parity Checker Solution : (Mealy)

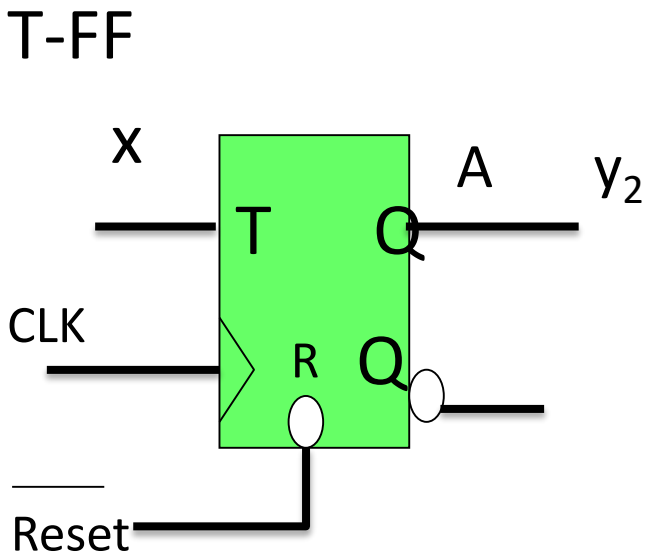


O/P is dependent on current state and input in Mealy

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

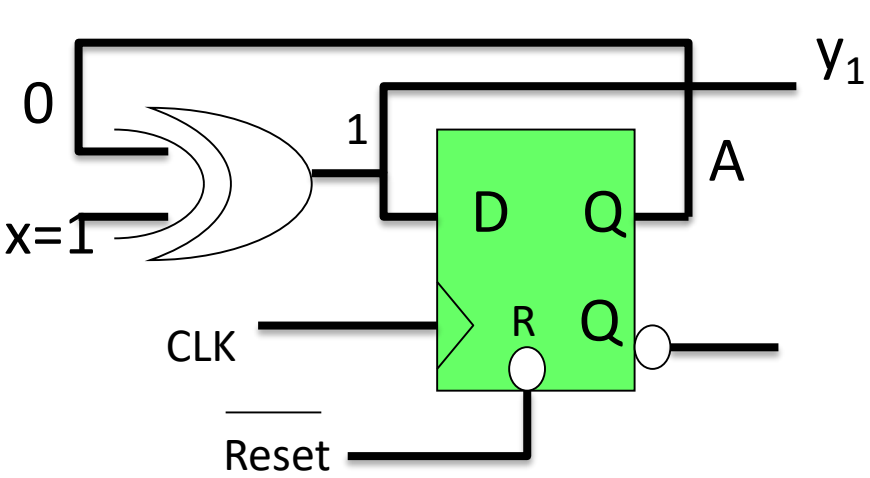
Parity Checker: Moore M/C Implementation



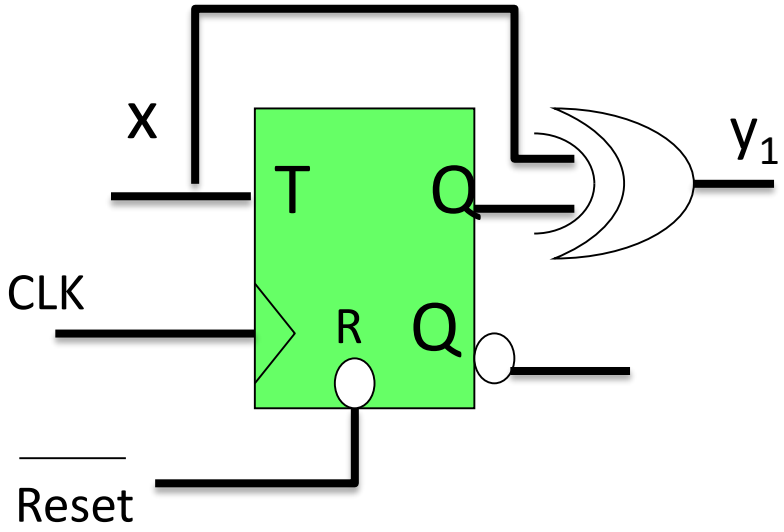


Moore O/P is synchronized with clock.

Parity Checker: Mealy M/C Implementation



D-FF



T-FF

Mealy O/P is not synchronized with clock.

Comparison : Mealy FSM Vs Moore FSM

- **Mealy** machines have **less states**
  - Outputs are on transitions (n^2) rather than states (n)
- **Moore** machines are **safer** to use
  - Outputs change at clock edge (always one cycle later)
- Mealy machines: input change can cause output change as soon as logic is done
  - A **big problem** when two machines are interconnected
  - Asynchronous feedback may occur if one isn't careful

Comparison : Mealy FSM Vs Moore FSM

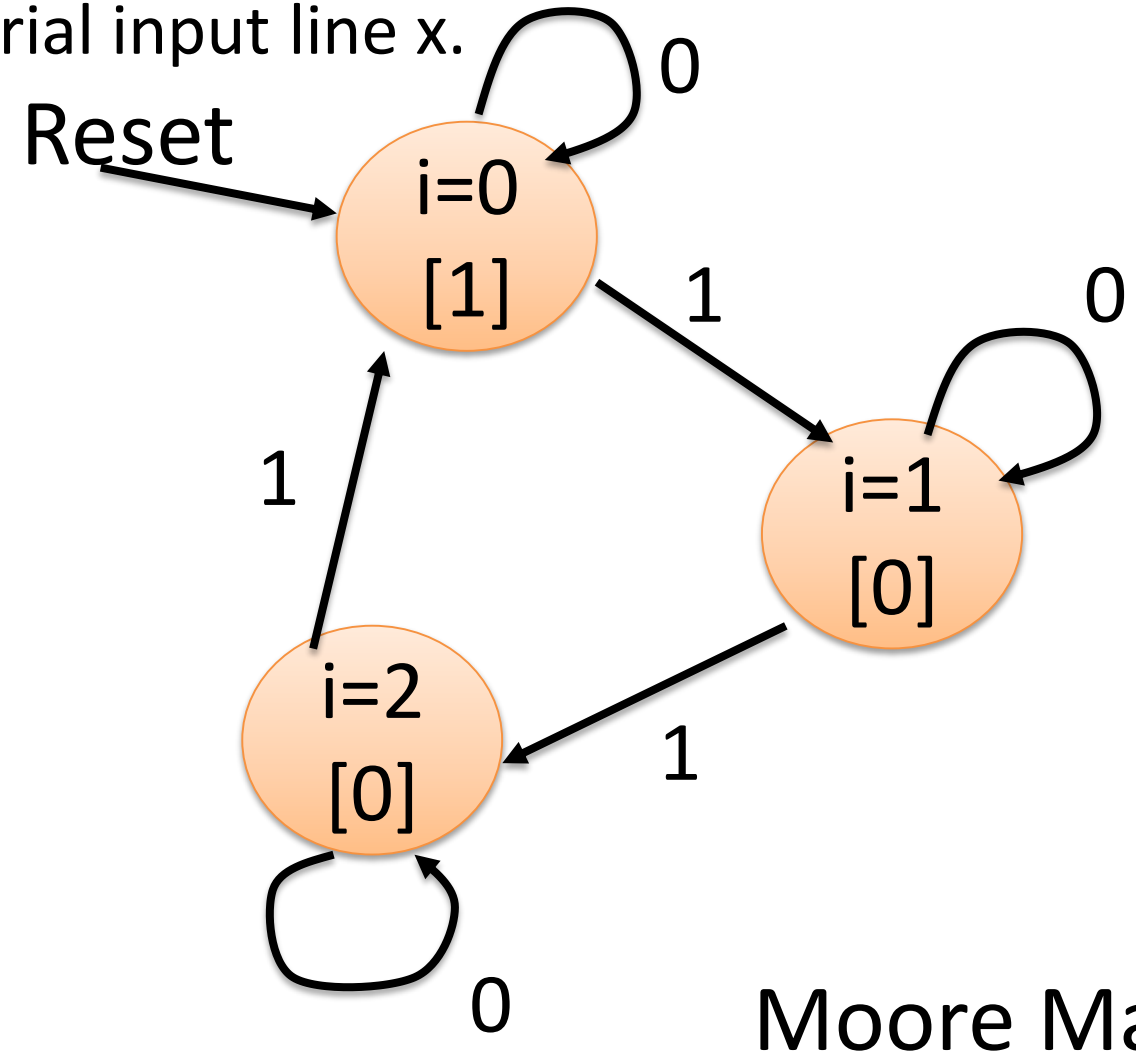
- **Mealy** machines **react faster** to inputs
  - react in same cycle – don't need to wait for clock
  - outputs may be considerably shorter than the clock cycle
  - in Moore machines, more logic may be necessary to decode state into outputs – there may be more gate delays after clock edge

FSM Example : Mealy vs Moore

- Design a system that outputs a ‘1’ whenever it receives a multiple of 3 # of 1’s (i.e., 0, 3, 6, 9, etc. # of 1’s) on a serial input line x.

FSM Example: Moore Machine

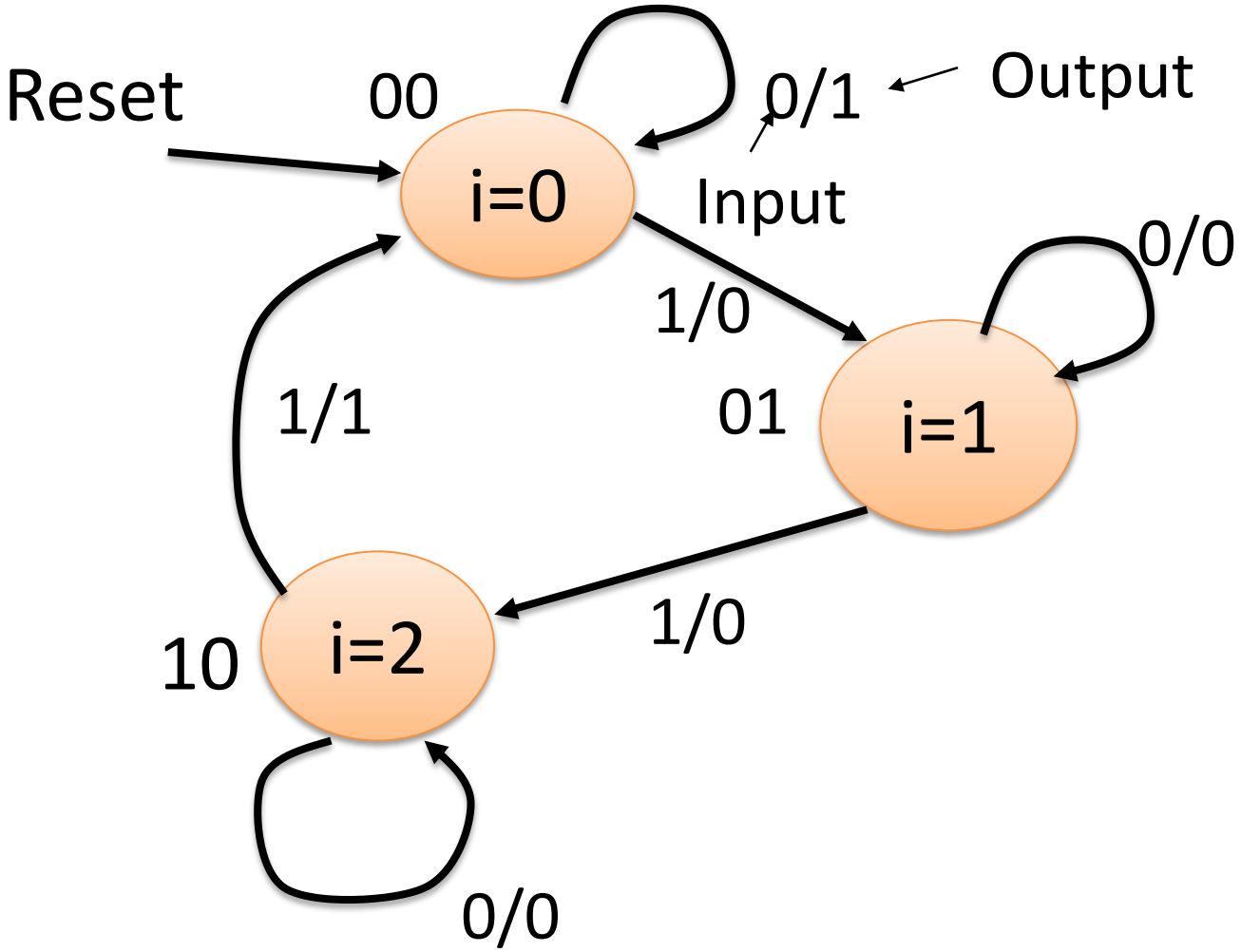
- Design a system that outputs a '1' whenever it receives a multiple of 3 # of 1's (i.e., 0, 3, 6, 9, etc. # of 1's) on a serial input line x.



Moore Machine

FSM Example: Moore Machine

- Design a system that outputs a '1' whenever it receives a multiple of 3 # of 1's (i.e., 0, 3, 6, 9, etc. # of 1's) on a serial input line x.



Mealy Machine

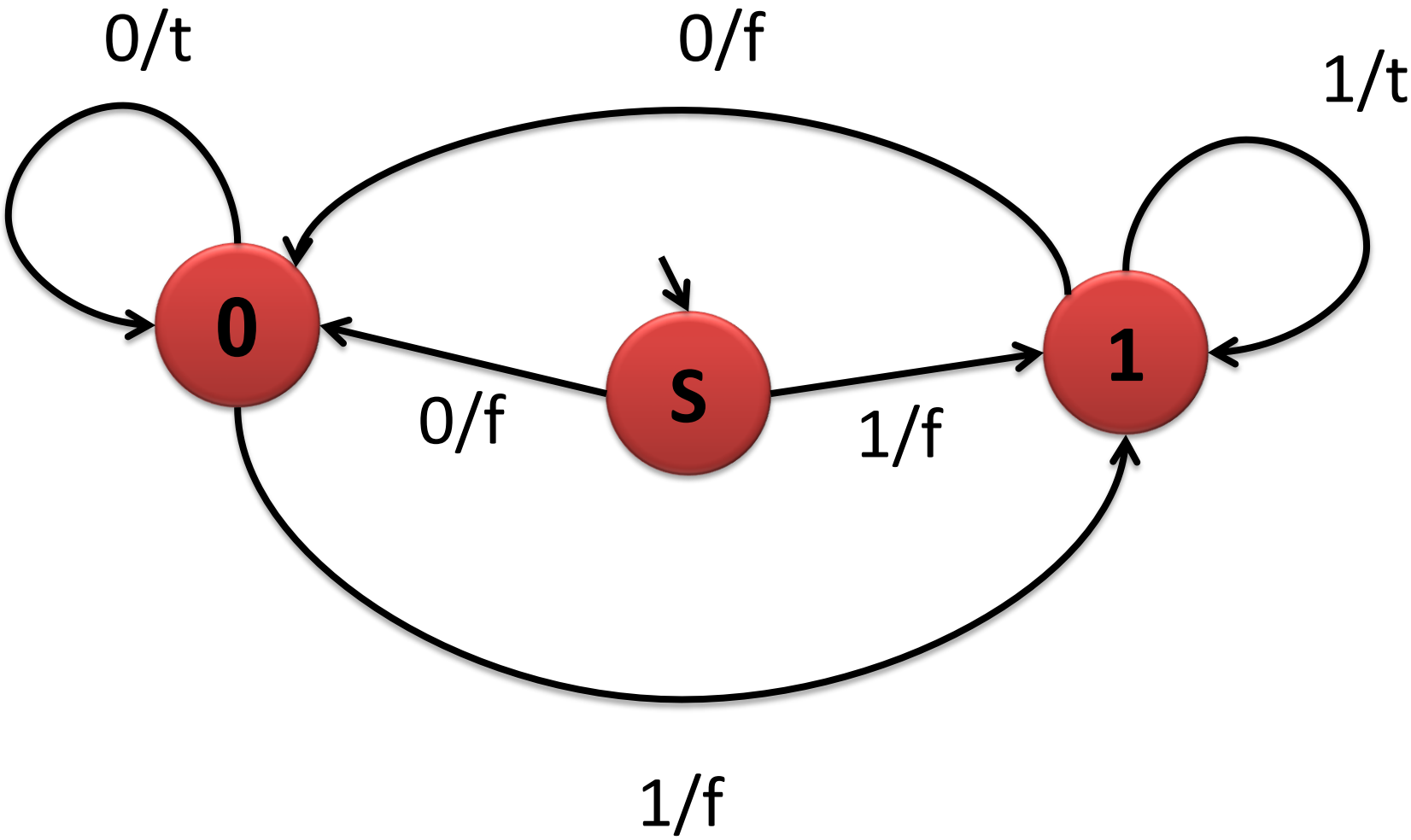
Mealy Vs Moore : Problem

- $\Sigma_O = \{t, f\}$ $\Sigma_I = \{0, 1\}$
- Outputs t $\Leftrightarrow$ at least 2 last input digits are same

Solutions to Problem

- Mealy implementation
  - –Needs only 3 states
- Moore implementation
  - –Needs 2 more almost duplicated states just in order to the output value t there.

FSM: Mealy Machine



FSM: Moore Machine

Outputs **true** if at least 2 last input digits are same

Is it possible to design Moore FSM of the same problem with 3 state ?

You can try now..

FSM: Moore Machine