Sequential Logic Design

Dr. Warisa Sritriratanarak

2023/2

Combinational logic

- Everything you've seen until now...
- ...is called **Combinational Logic**

Combinational logic

Combinational Logic

- Circuits built out of logic gates (AND, OR...)
- Without flip-flops

Combinational logic

Combinational Logic

- Can build any circuit (or program) where:
- Outputs depend on **current** inputs

Combinational logic

Combinational Logic

- **Cannot** build a circuit (or program) where:
- Outputs depend on **past** inputs
- In other words, there's no **history**

Combinational logic

History

- Many solutions need history
- E.g., detecting a 4-digit pin is correct

Sequential logic

Sequential logic

- Solves this problem
- It allows us to keep track of what's happened before

Sequential logic

Sequential logic

- Outputs depend on current and past inputs
- In digital systems, this is achieved through **flip-flops**

Pause to think

What happens here?

- Output of a NOT gate connected to its input?

Pause to think

What happens here?

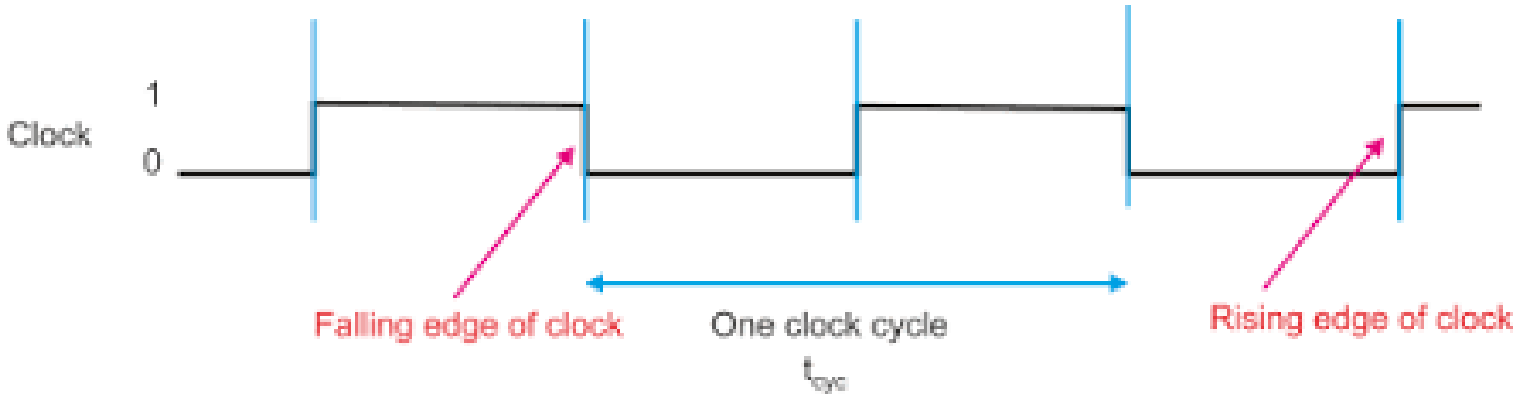
- Neither 0 not 1
- We broke binary

Sequential logic

Clocks

- Before introducing flip-flops, let's see what a clock is

Sequential logic



Clock signal

Sequential logic

Clocks

- A square-wave that oscillates between 0 and 1
- With a given frequency in Hz (and corresponding period in seconds)

Sequential logic

Clocks

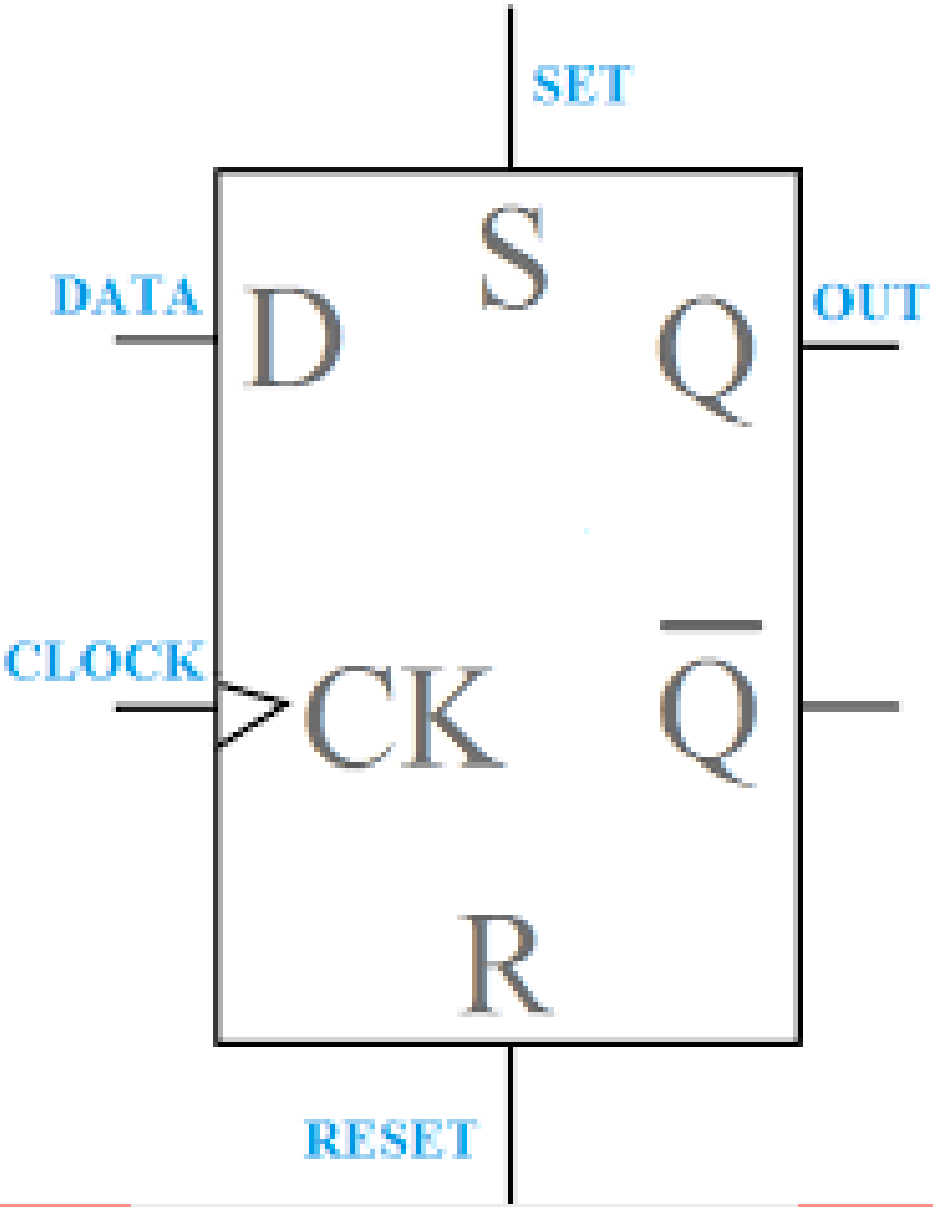
- 1Hz = 1 cycle per second (period is 1 second)
- 10Hz = 10 cycles per second (period is 0.1 seconds)

Sequential logic

Clocks

- Clock signal is a special input
- (Reset too)

Sequential logic



Truth table

$Q\prime$ is the output on the **next** clock transition

D	Q	$Q\prime$
0	0	0
0	1	0
1	0	1
1	1	1

Sequential logic

You can feed the (current) output of a flip-flop

- Back into its input function for (next) output
- This gives us **state**

Sequential logic

With 1 flip-flop (1 bit)

- We can encode 2 states (0 and 1)

Sequential logic

With 2 flip-flops (2 bits)

- We can encode 4 states (00, 01, 10 and 11)

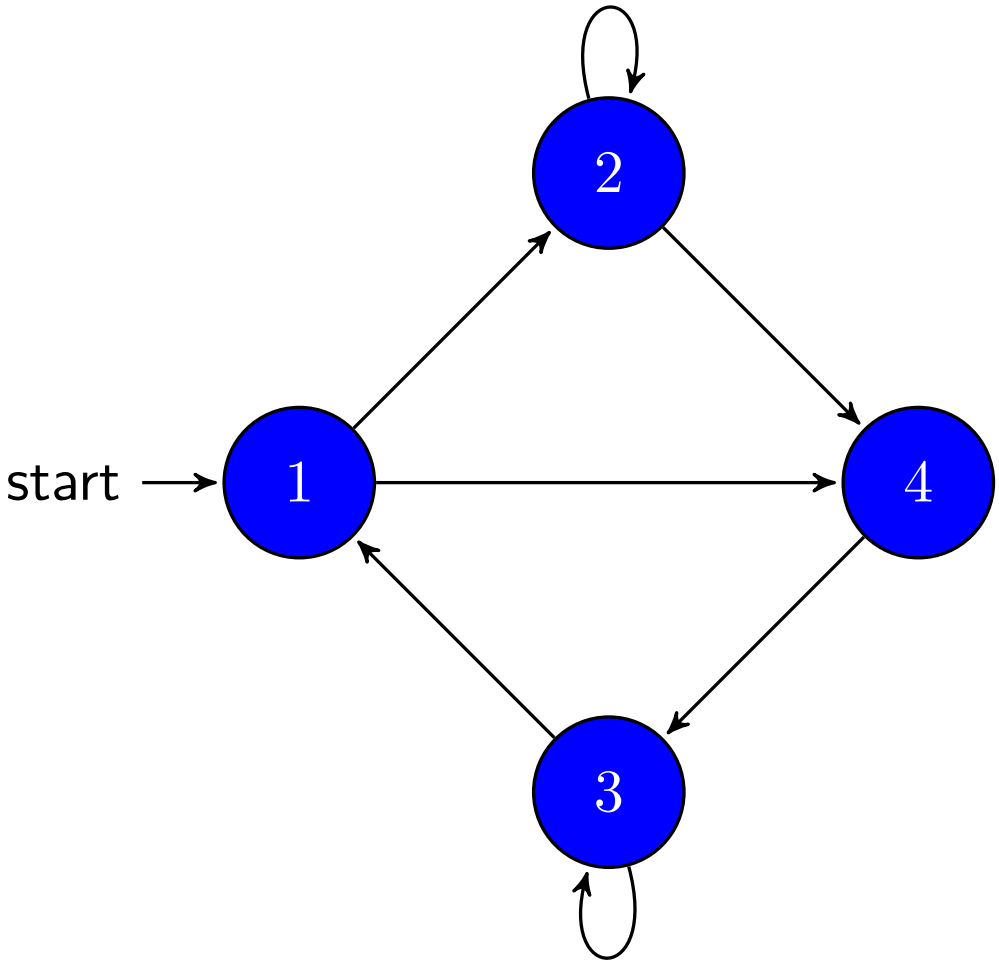
Sequential logic

With N flip-flops (N bits)

- We can encode 2^N states

Sequential logic

State machine



State machines

A state machine is:

- A set of states (in our case, flip-flop values)
- Rules for transitions between states (flip-flop logic functions)
- Rules for output in function of state

State machines

A state machine is:

- A solution to a sequential problem
- I.e., where the order of events matters

State machines

Imagine a system with 1 input "A"

- "A" changes synchronously to a clock

State machines

We want one output to be "1"

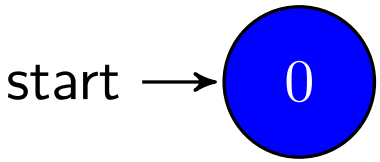
- Whenever "A" is 1 for 3 clock cycles in a row

State machines

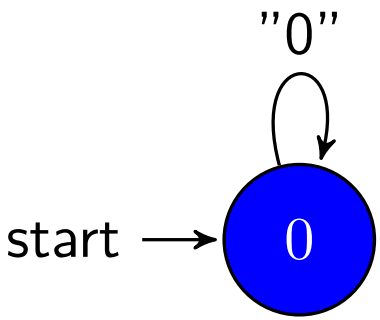
When the output is "1"

- It remains "1", as long as "A" is 1
- (goes back to 0 when "A" is 0)

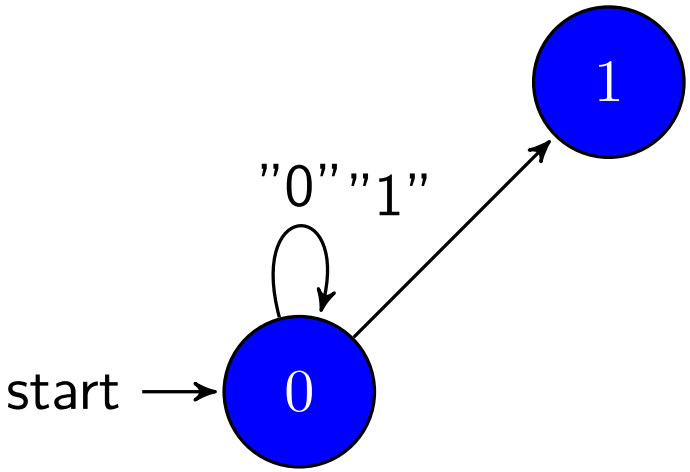
State machines



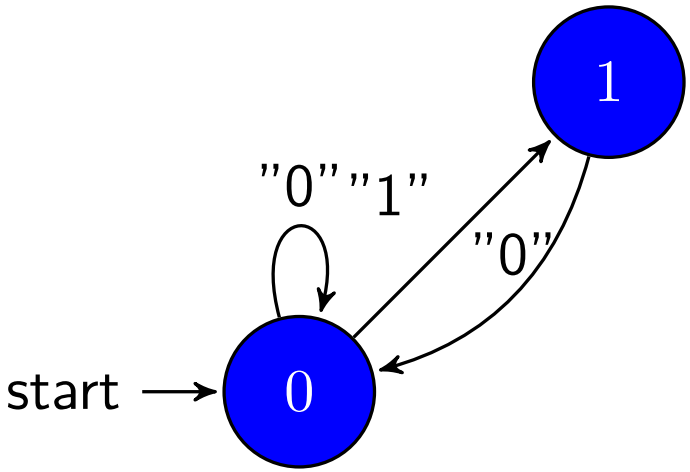
State machines



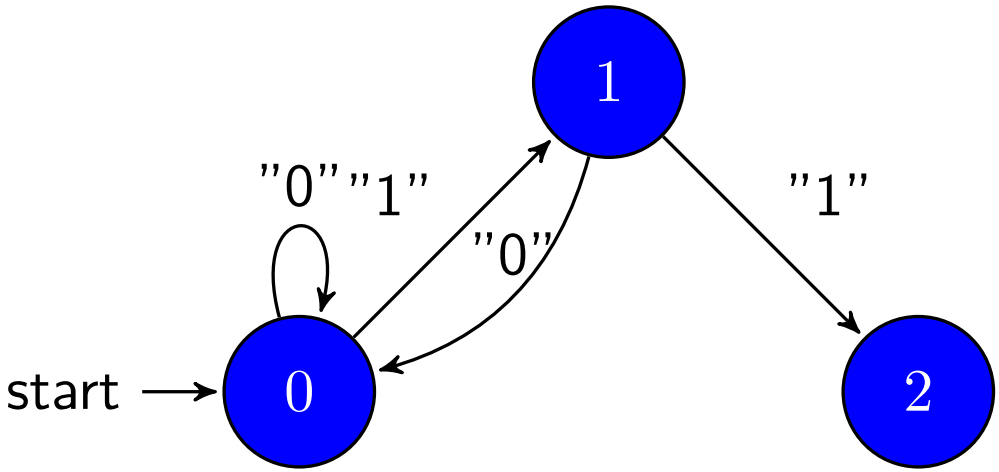
State machines



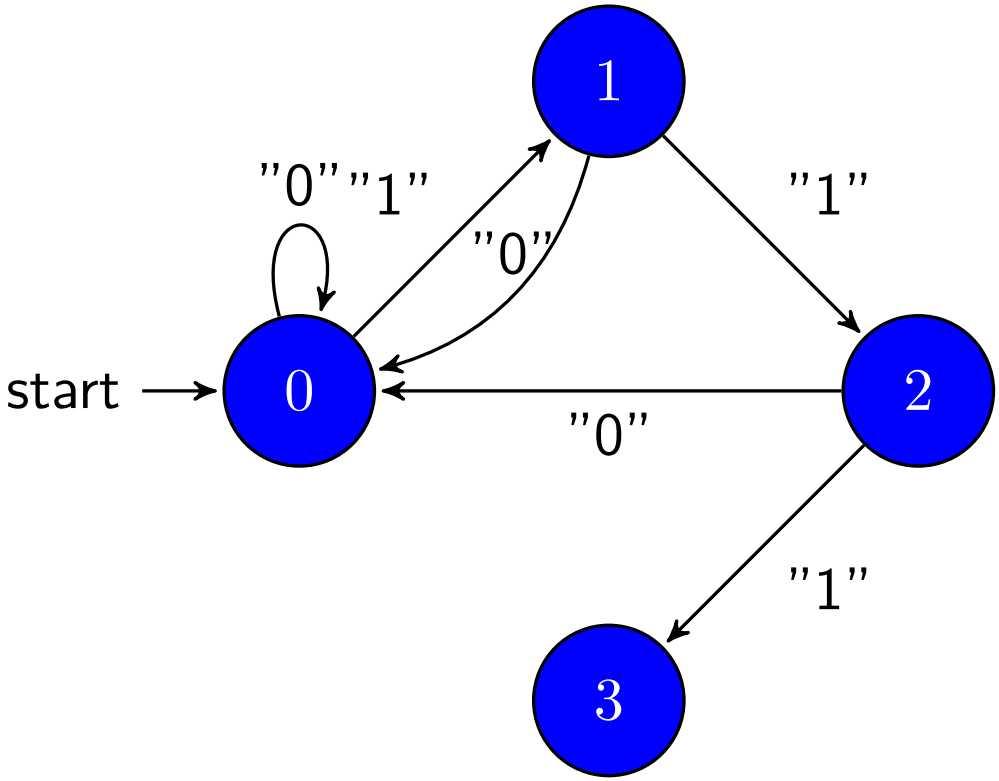
State machines



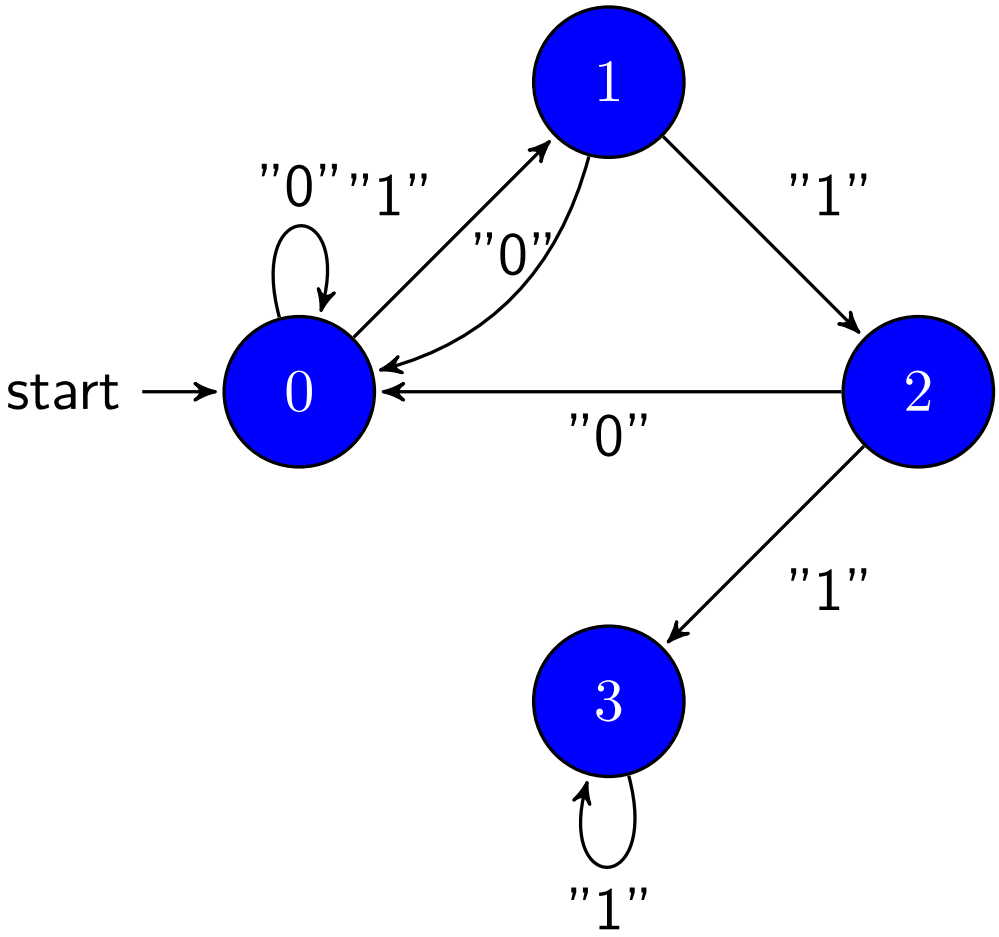
State machines



State machines

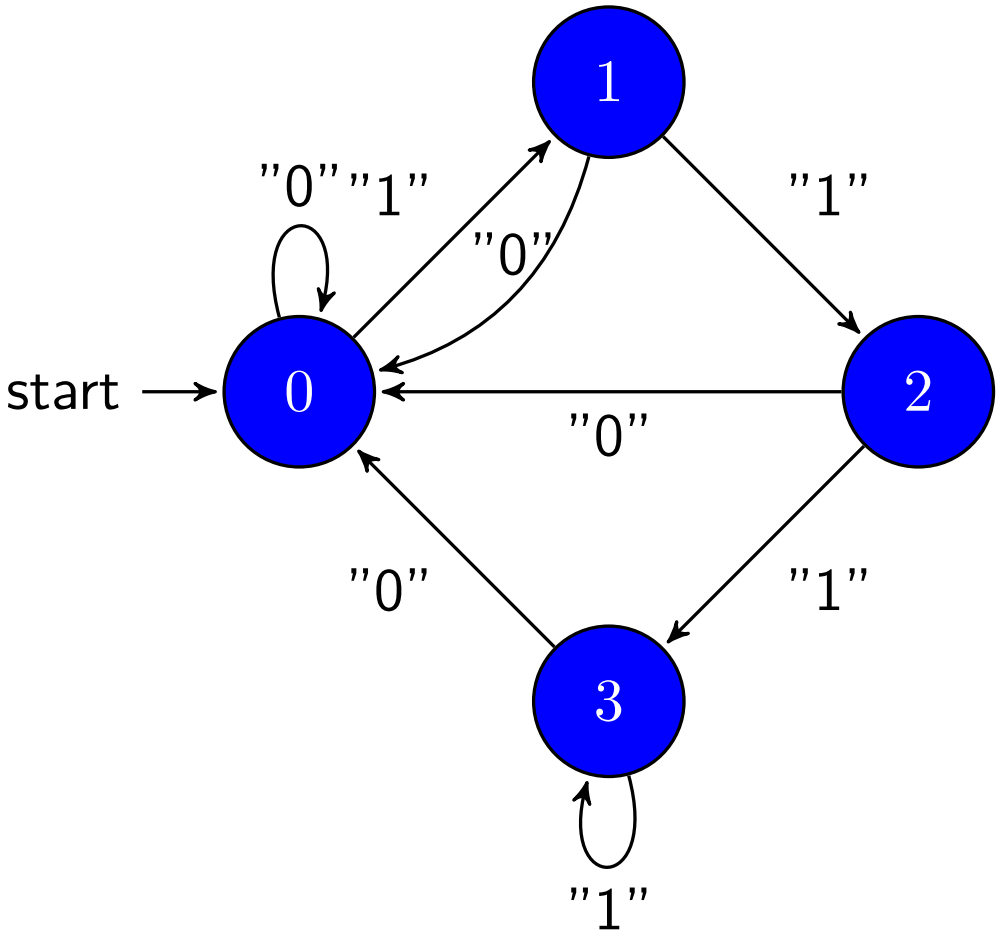


State machines



Output is "1" in state 3

State machines



Output is "1" in state 3

State machines

A big, big part of computer engineering

- (hardware or software)
- is being able to design state machines

State machines

How to implement them?

- The usual way
- (It's always truth tables)

State machines

Number of flip-flops N:

- 2^N states

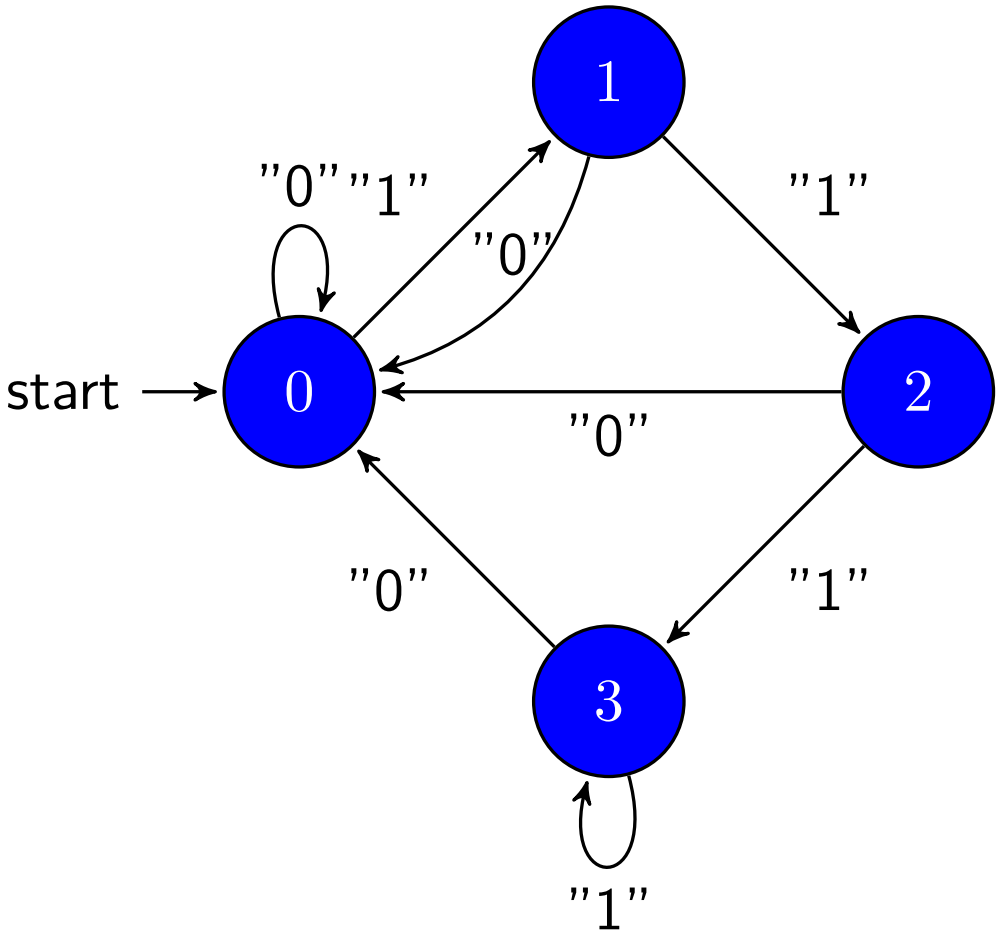
State machines

Q_1	Q_0	A	D_1	D_0	O

State machines

Q_1	Q_0	A	D_1	D_0	O
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

State machines

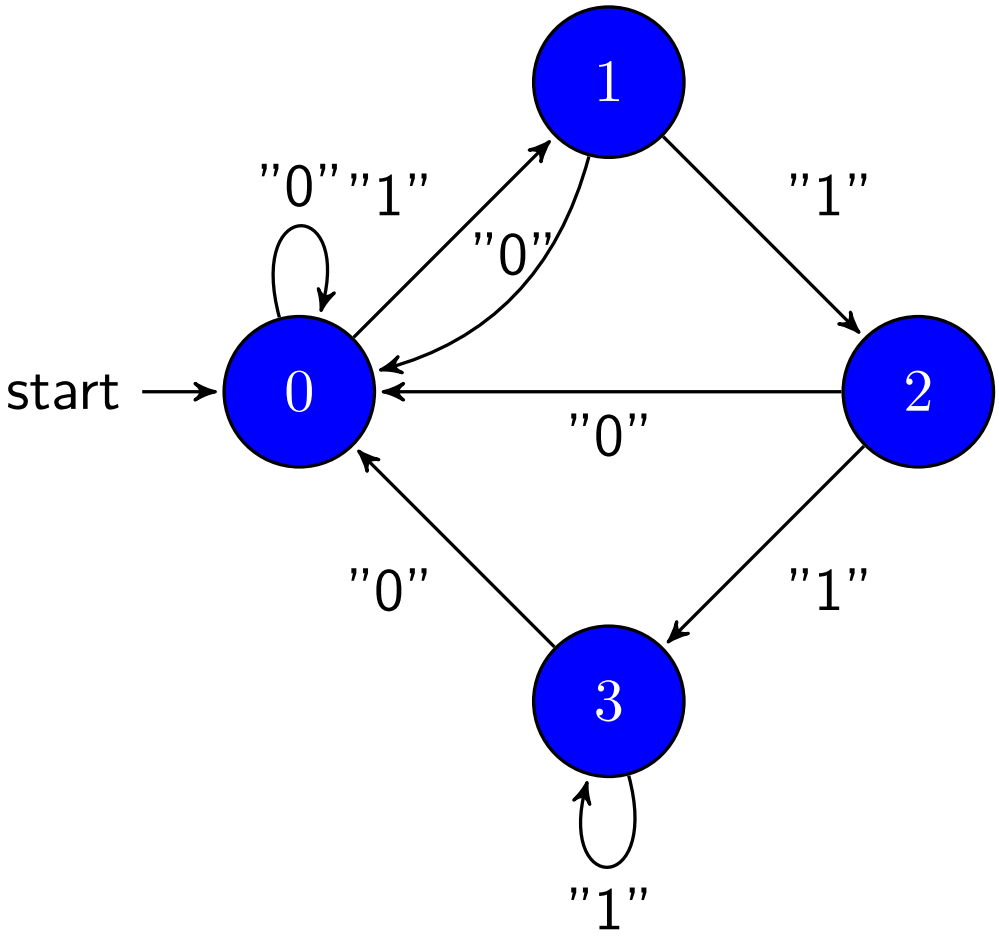


Output is "1" in state 3

State machines

Q_1	Q_0	A	D_1	D_0	O
0	0	0	0	0	0
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

State machines

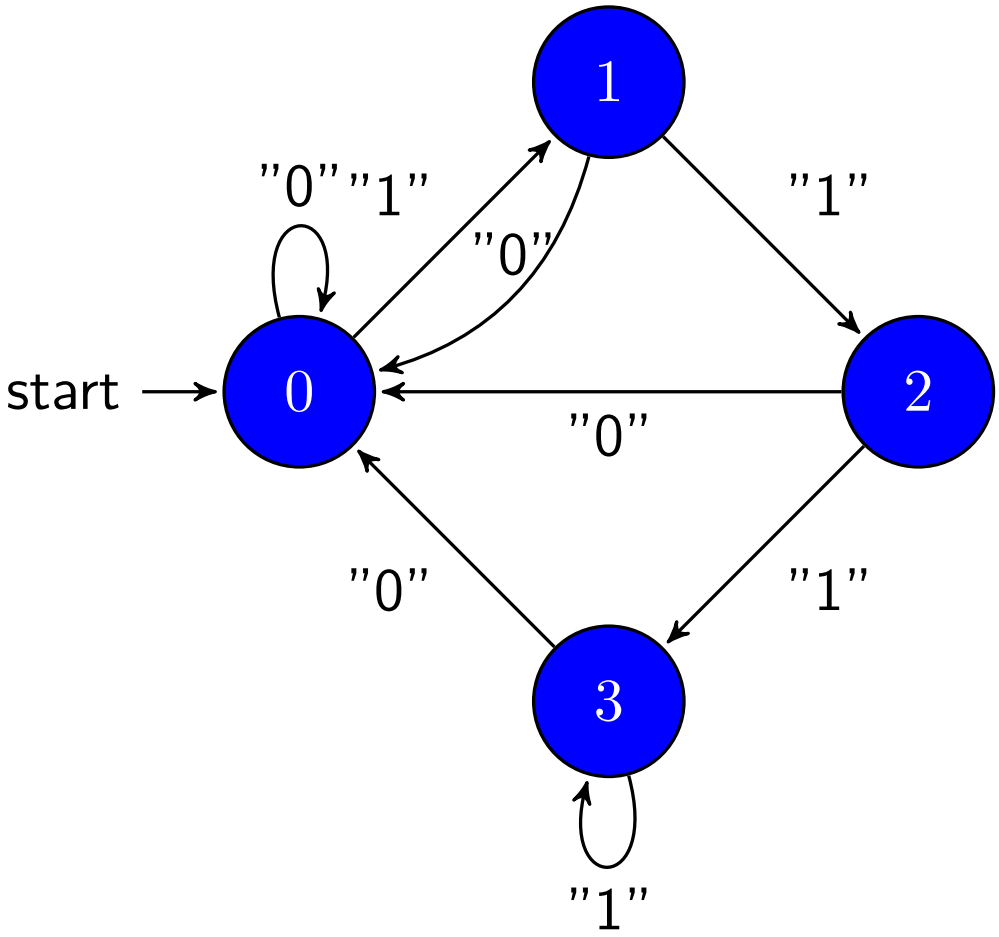


Output is "1" in state 3

State machines

Q_1	Q_0	A	D_1	D_0	O
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

State machines

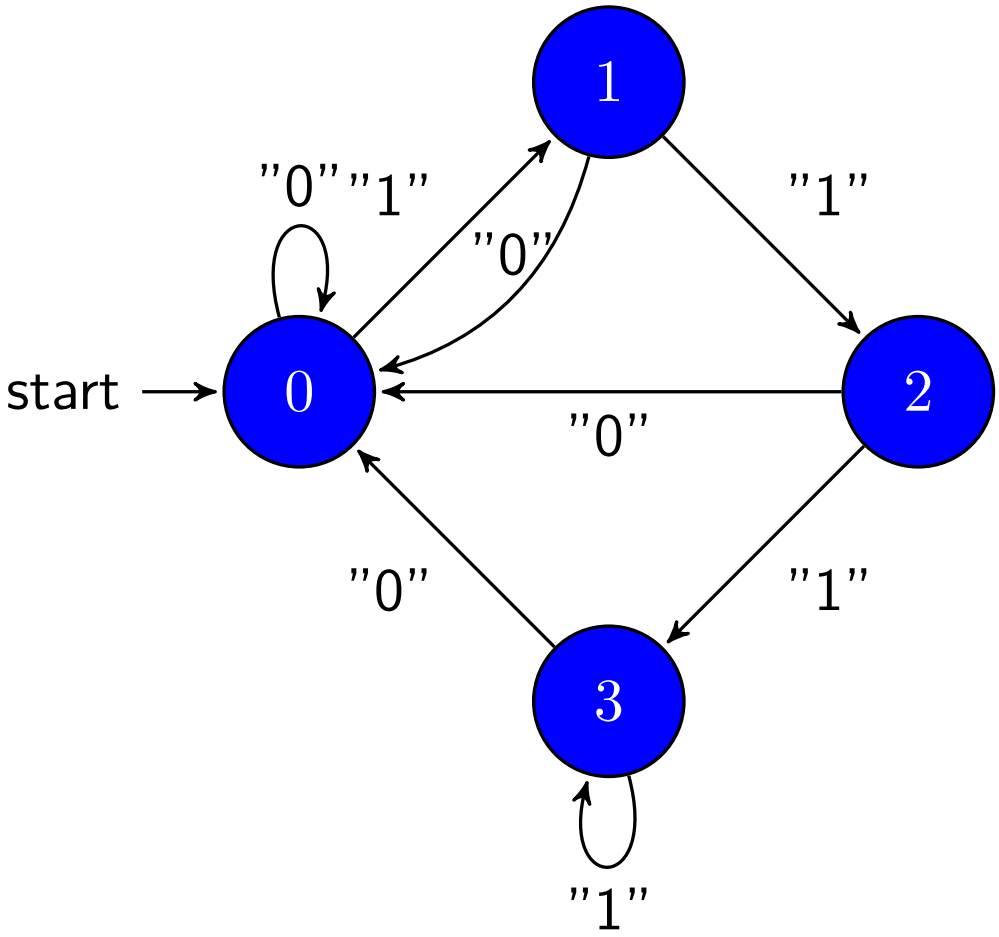


Output is "1" in state 3

State machines

Q_1	Q_0	A	D_1	D_0	O
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

State machines

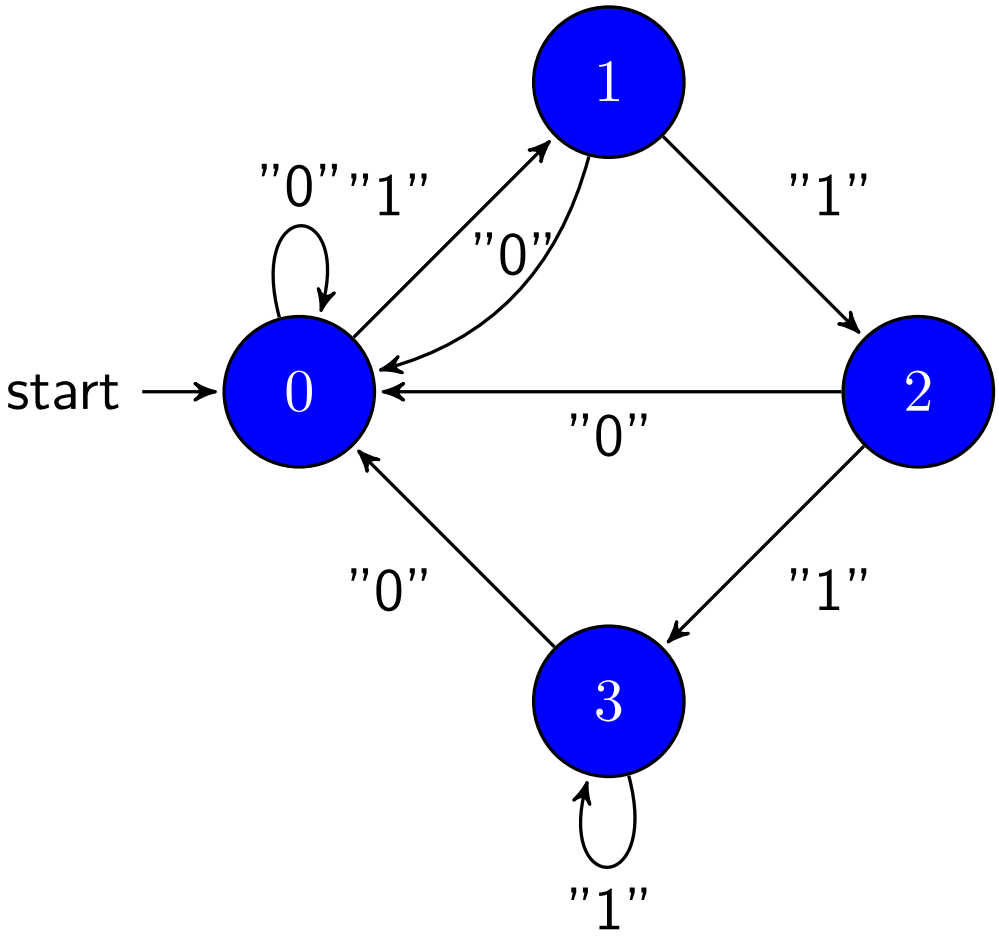


Output is "1" in state 3

State machines

Q_1	Q_0	A	D_1	D_0	O
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0			
1	0	1			
1	1	0			
1	1	1			

State machines

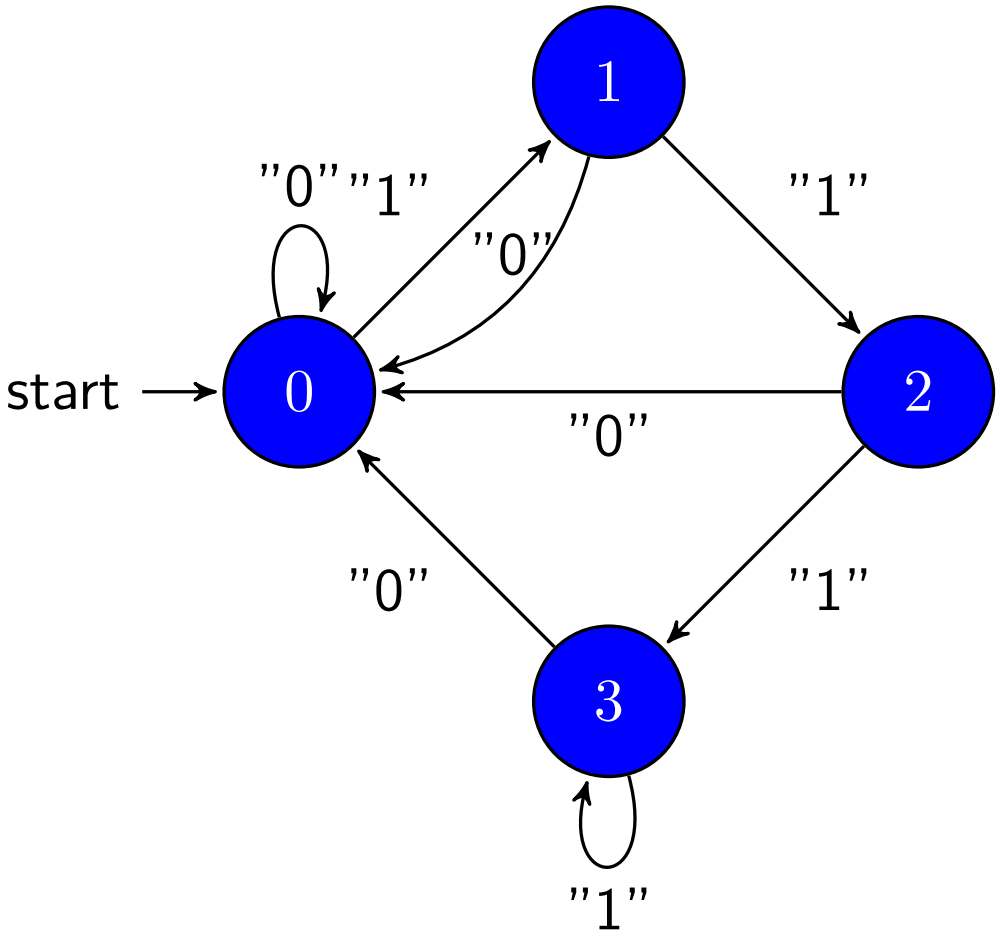


Output is "1" in state 3

State machines

Q_1	Q_0	A	D_1	D_0	O
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1			
1	1	0			
1	1	1			

State machines

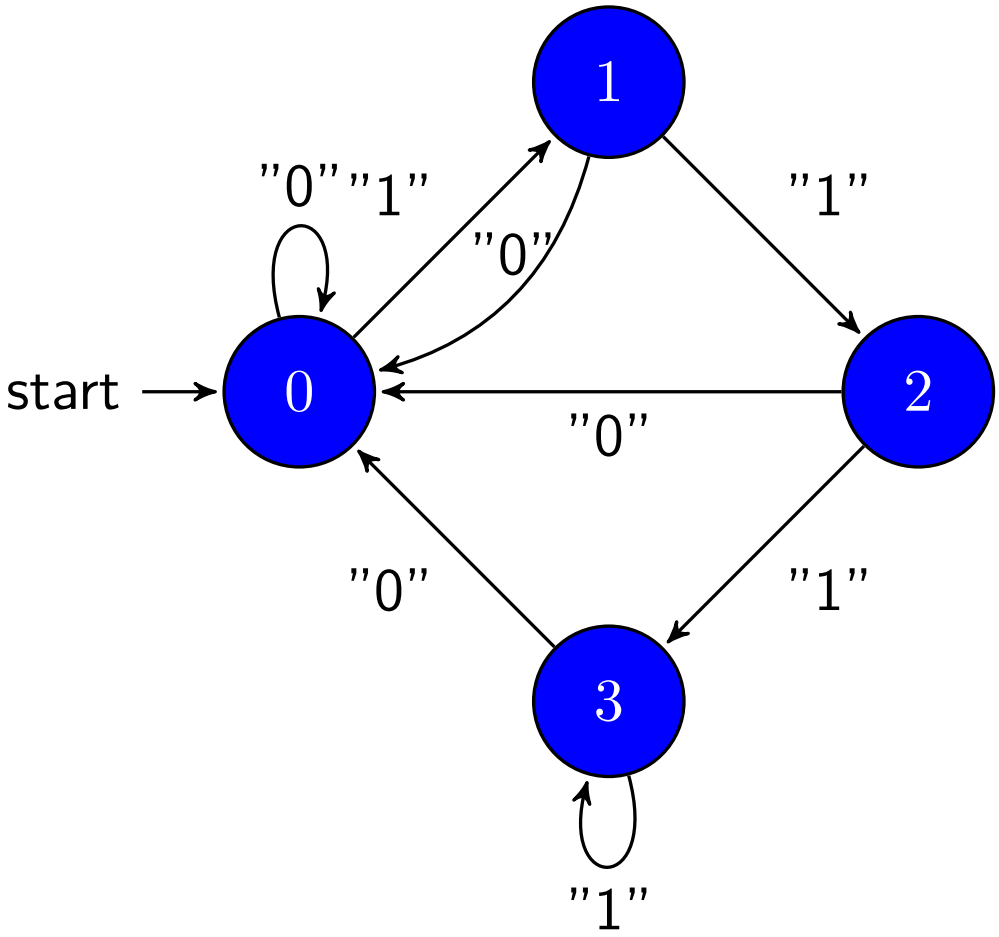


Output is "1" in state 3

State machines

Q_1	Q_0	A	D_1	D_0	O
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0			
1	1	1			

State machines

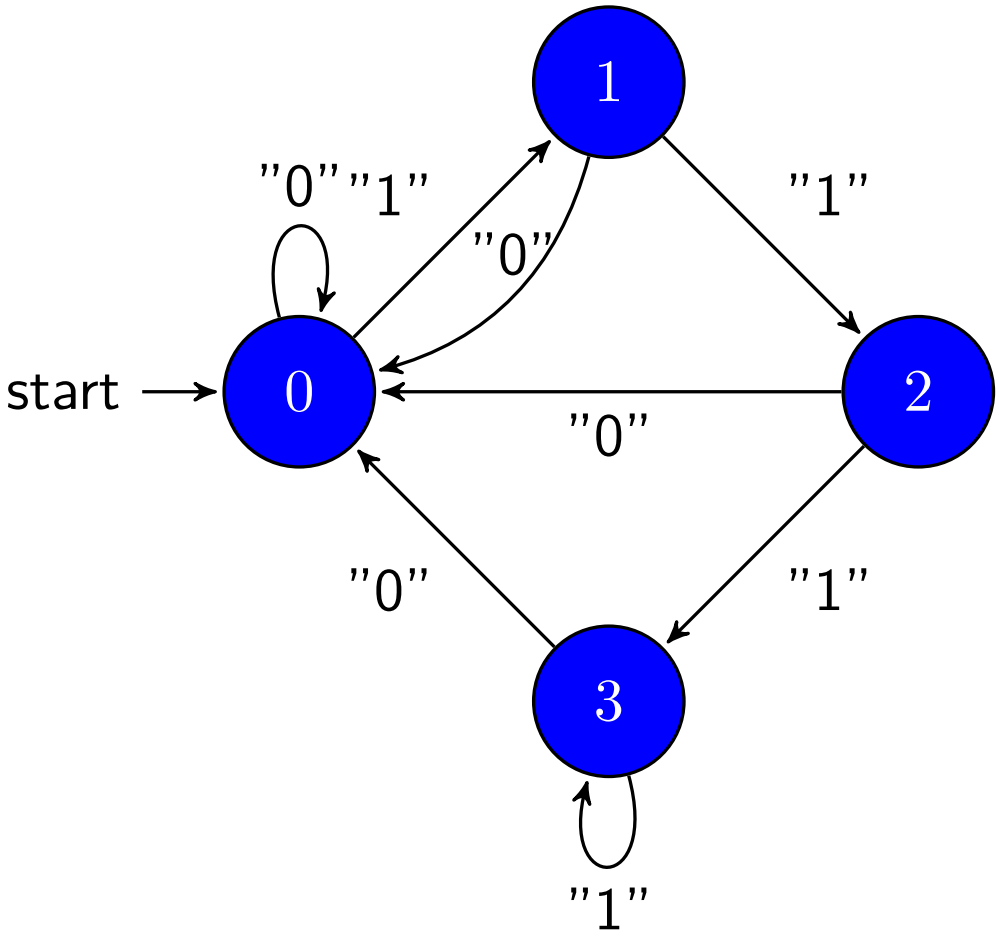


Output is "1" in state 3

State machines

Q_1	Q_0	A	D_1	D_0	O
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1			

State machines



Output is "1" in state 3

State machines

Q_1	Q_0	A	D_1	D_0	O
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

State machines

- $\mathsf{O} = Q_1.Q_0$
- $D_1 = \mathsf{A}.(Q_1 + Q_0)$
- $D_0 = \mathsf{A}.(!Q_0 + Q_1.Q_0)$

State machines

Problem: Simple vending machine

- Two sensors:
  - A: "1" if the coin is 5 baht
  - B: "1" if the coin is 10 baht

State machines

Problem: Simple vending machine

- Design a system that sells drinks for 15baht
- One output to dispense drink
- (assume person never inserts more than 15baht)