Finite State Machines

Lecture 4

Recall a Language is Regular if

- L is empty
- L contains a single string (could be the empty string)
- If L_1, L_2 are regular, then $L = L_1 \cup L_2$ is regular
- If L_1, L_2 are regular, then $L = L_1 L_2$ is regular
- If L is regular, then L^* is regular

Unbounded vs. Infinite

- Why do we need bullet 5?
- Why can't we say that L^* is the infinite union of $\{\varepsilon\} \cup L \cup LL \cup LLL \cup \ldots$
- Recursive definitions: at every branch of recursion we need to reach a base case in **finite number** of steps.
- We can invoke the union rule for any integer n number of steps
- infinity is not a number! I can only produce infinite sets by an operation like the *.

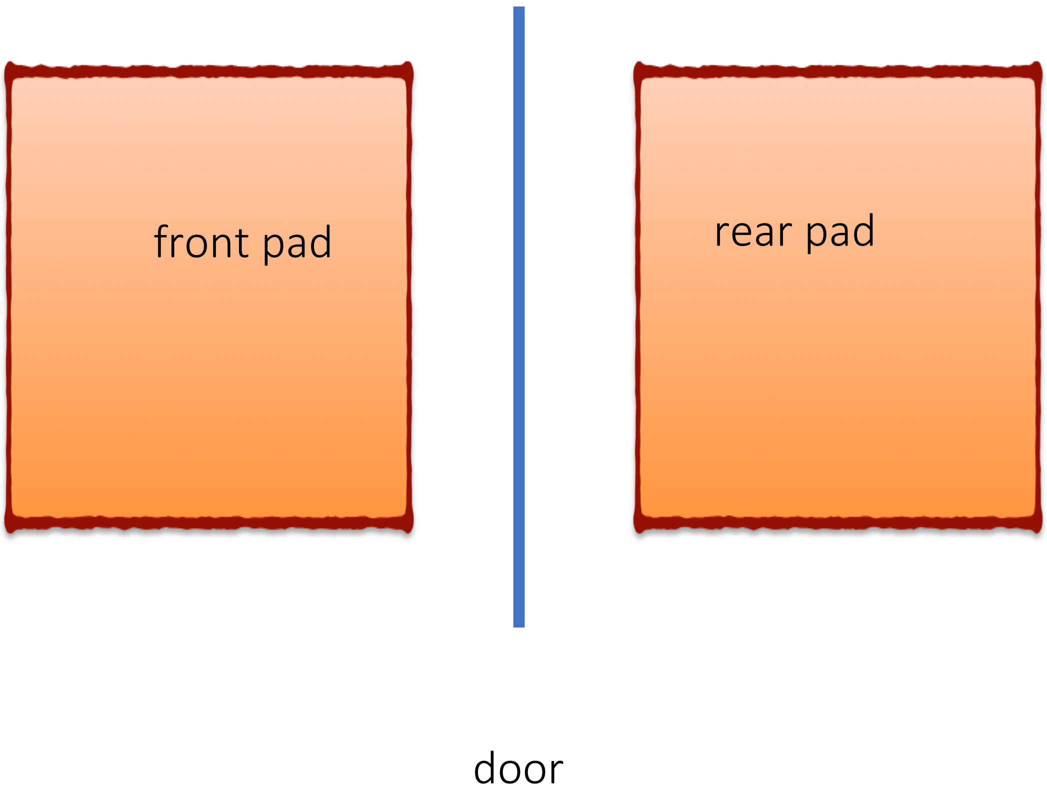
Complexity of Languages

- Central Question: How complex an algorithm is needed to compute (aka decide) a language? How much memory do I need?

- Today: a simple class of algorithms, that are fast and can be implemented using minimal hardware
  - Finite State Machines -Deterministic Finite Automata (FSM-DFA)
  - DFAs around us: Vending machines, Elevators, Digital watch logic, Calculators, Lexical analyzers (part of program compilation), …

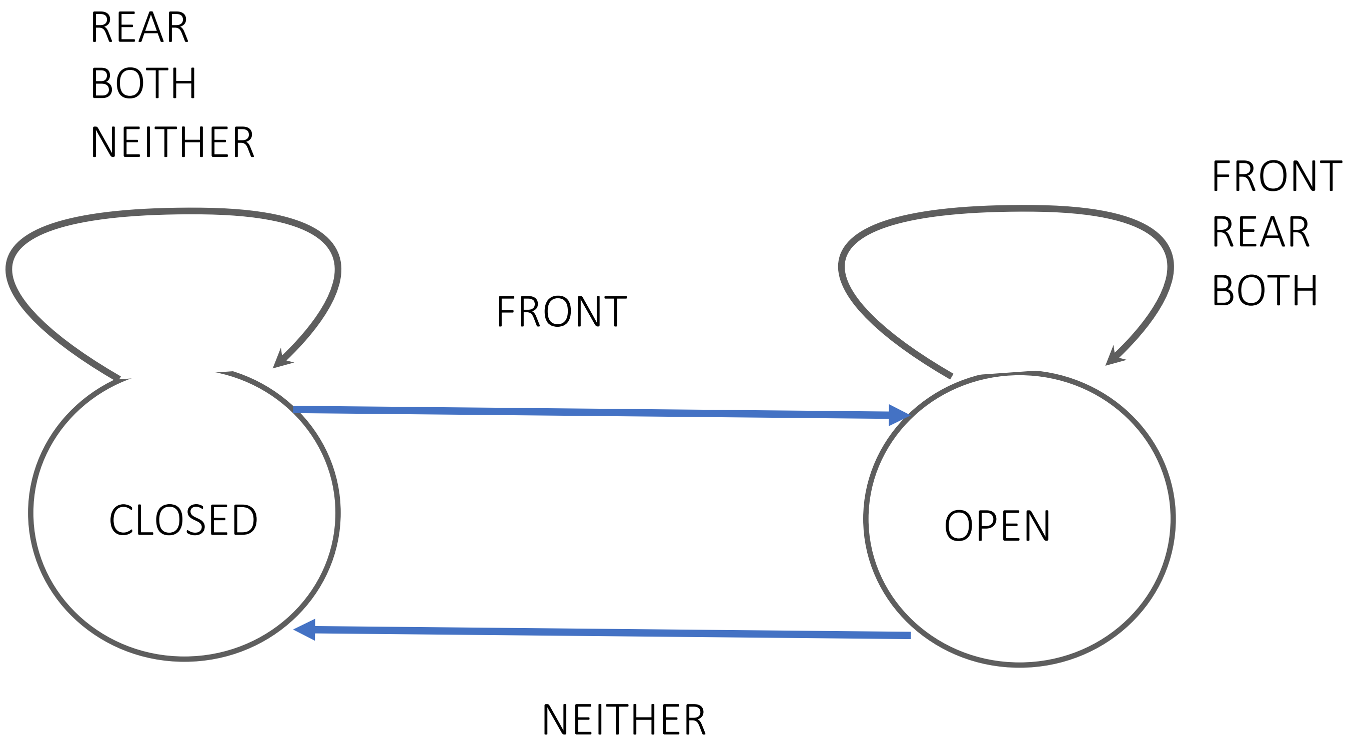
DFA (a.k.a. FSM)

- Finite: cannot use more memory to work on longer inputs
- Eg. Automatic door



DFA (a.k.a. FSM)

- Finite: cannot use more memory to work on longer inputs
- Eg. Automatic door



DFA (a.k.a. FSM)

- Finite: cannot use more memory to work on longer inputs
- Eg. Automatic door

Input signal

State

	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

Multiple of 5

```
MULTIPLEOF5(w[1..n]):
    rem ← 0
    for i ← 1 to n
        rem ← (2 · rem + w[i]) mod 5
    if rem = 0
        return TRUE
    else
        return FALSE
```

- Could do long division, keep the intermediate results in an array but I don't want to spend that much memory!

- Only one variable, rem, which represents the remainder of the part of the string I read so far when I divided by 5.

Multiple of 5

$\textsc{MultipleOf5}(w[1..n])$:

$rem \leftarrow 0$
for $i \leftarrow 1$ to n
    $rem \leftarrow (2 \cdot rem + w[i]) \bmod 5$
if $rem = 0$
    return $\textsc{True}$
else
    return $\textsc{False}$

m0=2m if I see “0” next
m1=2m+1 if I see “1” next

- If I know the remainder for m mod 5, and I read one more bit then line 3 tells me what the new remainder is (either m0 or m1)

Multiple of 5

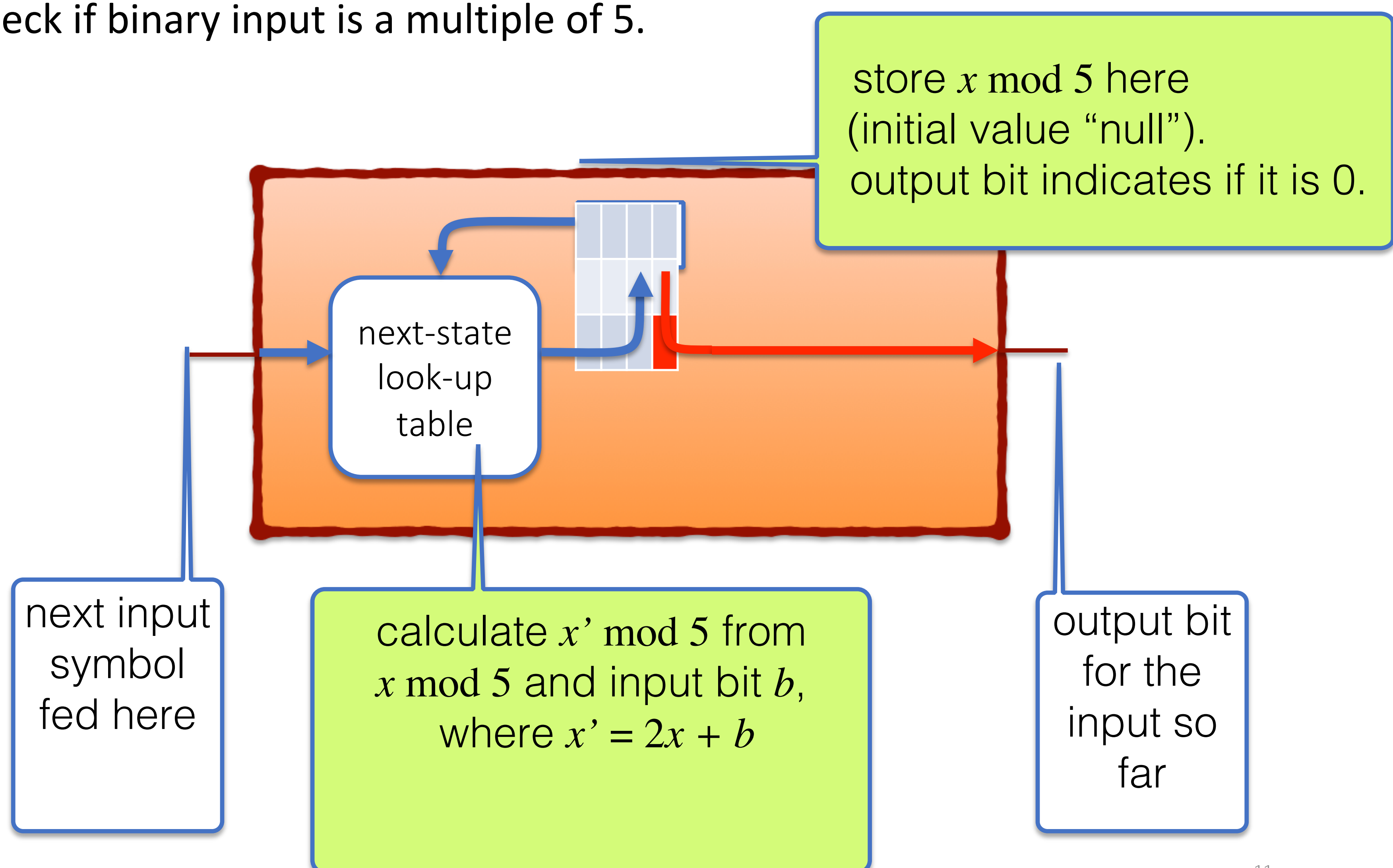
- Important feature of algorithm: Aside from variable i which counts the input bits and is necessary to read input, I only have one variable rem, which takes only a small (5) number of values.

- Streaming algorithm : Data flies by! Once w[i] is gone, it is gone forever.

- Variable has a very small number of states, which I am able to specify at compile time. Very small amount of memory!

DFA (a.k.a. FSM)

- check if binary input is a multiple of 5.



"Lookup" table

```
DoSomethingCool(w[1..n]):
    q ← 0
    for i ← 1 to n
        q ← δ[q, w[i]]
    return A[q]
```

- q encapsulates the state of the algorithm

- Takes a small amount of values, which I know up front (e.g. q is a number between 1 and 4). Unbounded, not infinite!

- Depending on the character I read at position i, I change my state with function called delta (δ).

- I have a hardcoded array A and based on what the state is when I finish reading the string, I output the value of the array.

“Lookup” table

```
DoSomethingCool(w[1..n]):
    q ← 0
    for i ← 1 to n
        q ← δ[q, w[i]]
    return A[q]
```

If we want to use our new DoSomethingCool algorithm to implement MultipleOf5, we simply give the arrays δ and A the following hard-coded values:

q	$\delta[q, 0]$	$\delta[q, 1]$	$A[q]$
0	0	1	True
1	2	3	False
2	4	0	False
3	1	2	False
4	3	4	False

only one accepting state!

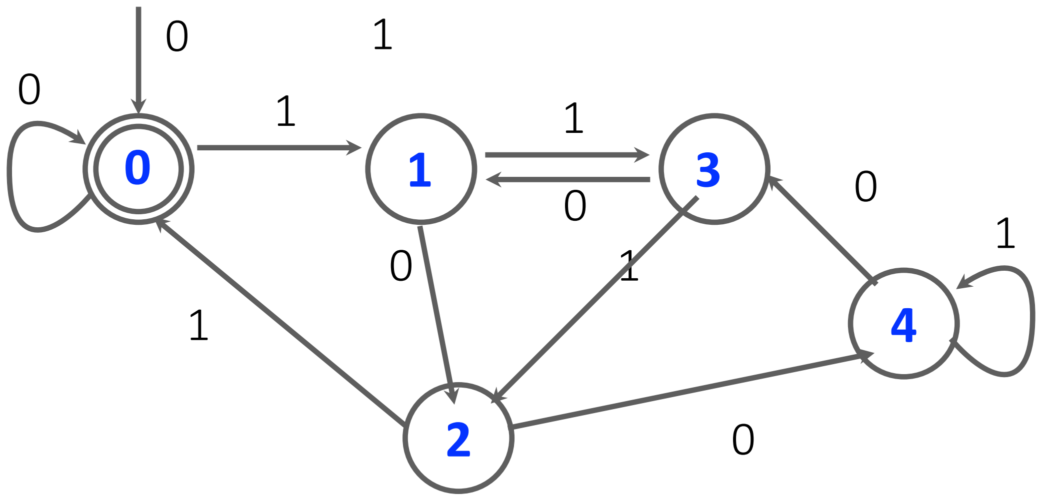
Instead of doing arithmetic at all, I could just hard code this lookup table into the code and simply do a lookup

DFA (a.k.a. FSM)

- Algorithm or Machine? Algorithm is a Machine!!
- Once you program the machine, you don't have to monitor it. It runs AUTOMATICALLY (Automaton…)

DFA (a.k.a. FSM)

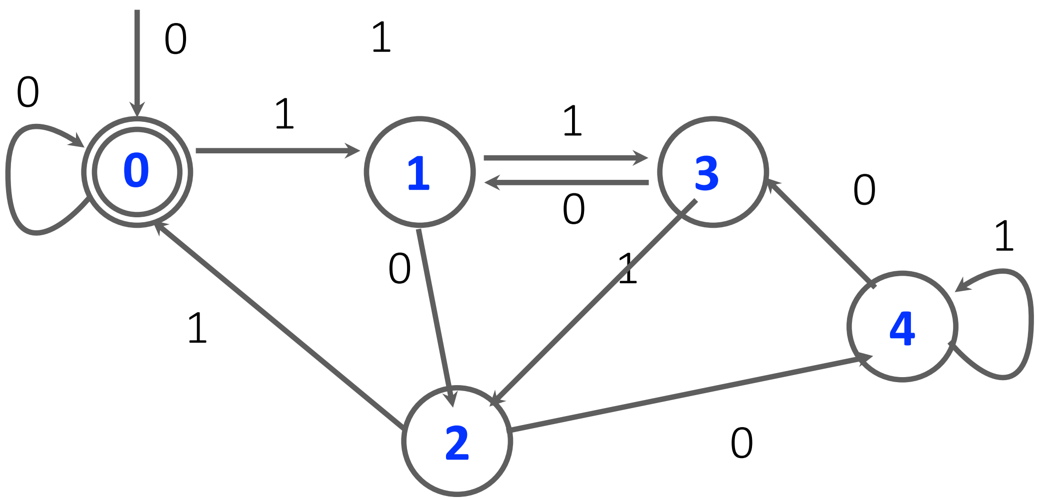
- Equivalent view as a graph!



DFA (a.k.a. FSM)

- Example: check if input 01010101 is a multiple of 5

input bit	current state	next state
0	**0**	**0**
1	**0**	**1**
0	**0**	**2**
1	**2**	**0**
0	**0**	**0**
1	**0**	**1**
0	**1**	**2**
1	**2**	**0**



DFA (a.k.a. FSM)

- check if input (MSB first) is a multiple of 5

input bit	current state	next state
0	0	0
1	0	1
0	0	2
1	2	0
0	0	0
1	0	1
0	1	2
1	2	0

How to fully specify a DFA (syntax):

FINITE Alphabet: Σ

FINITE Set of States: Q

Start state: $s \in Q$

Set of Accepting states: $A \subseteq Q$

Transition Function: $\delta : Q \times \Sigma \rightarrow Q$

$$\boxed{\delta(q, a) = (2q + a) mod 5}$$

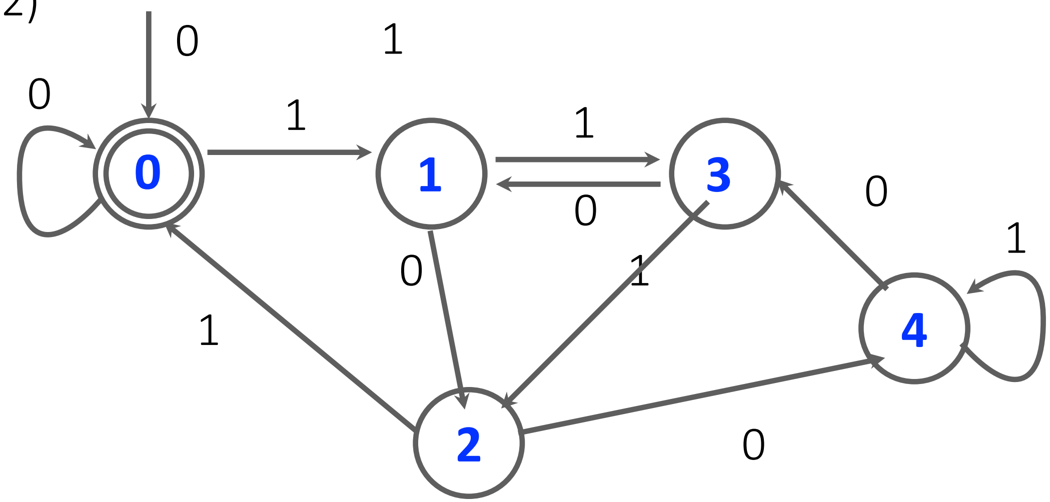
DFA (a.k.a. FSM)

- 3 equivalent ways to specify a FSM:

1)

q	$\delta[q,0]$	$\delta[q,1]$	$A[q]$
0	0	1	TRUE
1	2	3	FALSE
2	4	0	FALSE
3	1	2	FALSE
4	3	4	FALSE

2)



3)

$$\delta(q,a) = (2q+a)mod5$$

Together with a description of what are the states and what are the accepting states

How to interpret these functions?

- $M = (\Sigma, Q, \delta, s, A)$
- $\delta^*(q,w)$ be the state M reaches starting from a state $q \in Q$, on input $w \in \Sigma^*$
- *Recursive definition?*
- *What are the cases going to be?*

Behavior of a DFA on an input

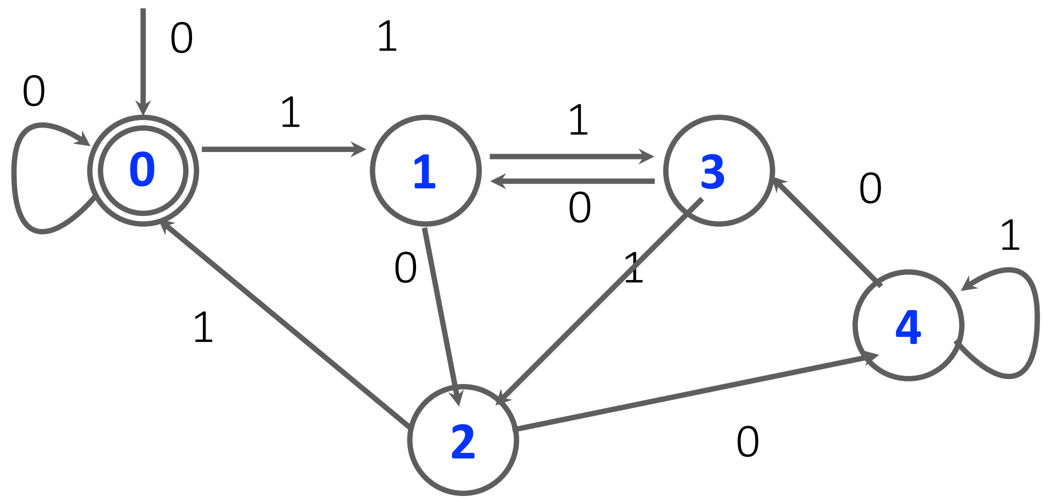
- $M = (\Sigma, Q, \delta, s, A)$
- $\delta^*(q,w)$ be the state M reaches starting from a state $q \in Q$, on input $w \in \Sigma^*$
- Formally,

- $\delta^*(q,w) = q \;\; \text{if } w{=}\varepsilon$
- $\delta^*(q,w) = \delta^*(\delta(q,a), \mathrm{x}) \;\; \text{if w=ax}$

recursion!

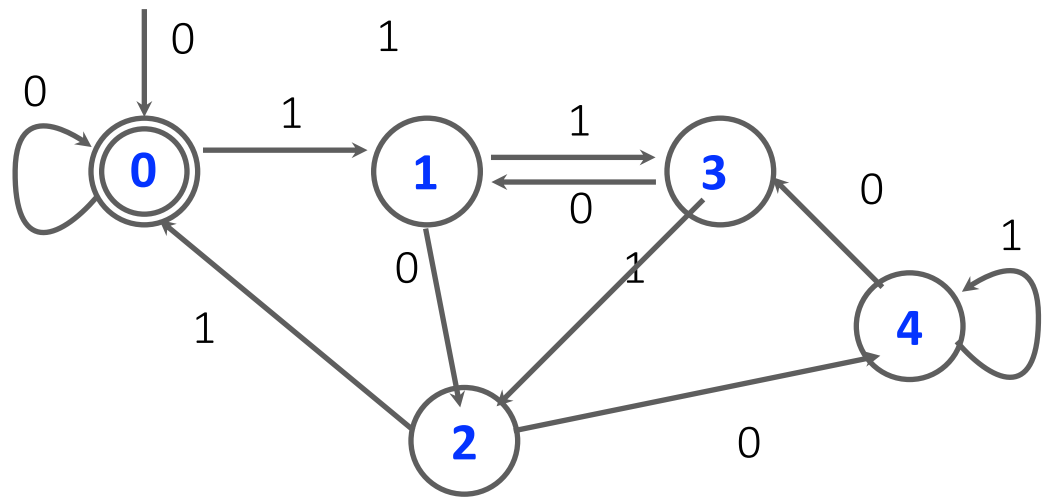
Behavior of a DFA on an input

- $\delta^*(0{,}01001)$ = ? 4
- $\delta^*(0,\varepsilon)$ = ? 0
- $\delta^*(0{,}010)$ = ? 2
- $\delta^*(2{,}01)$ = ? 4

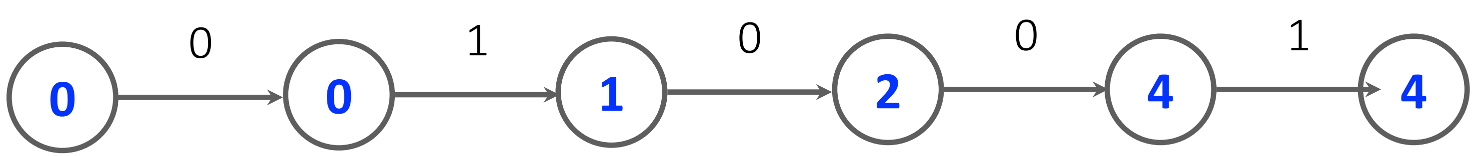


Behavior of a DFA on an input

- $\delta^*(0,01001) = 4$



- Specify a walk in the graph
- Best represented as



Example: What strings does this machine accept?

Alphabet: $\Sigma = \{0,1\}$

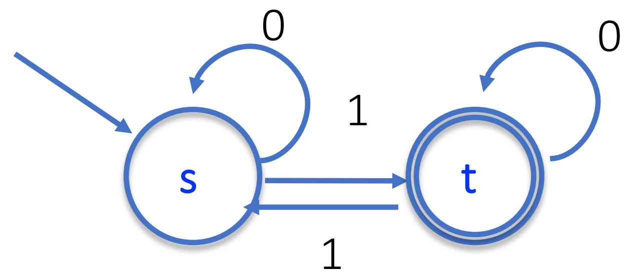
Set of States: $Q = \{s,t\}$

Start state: $s \in Q$

Accepting state: $t \in Q$

Transition Function: $\delta : Q \times \Sigma \rightarrow Q$

$\delta(s,0)=s$, $\delta(s,1)=t$, $\delta(t,0)=t$, $\delta(t,1)=s$



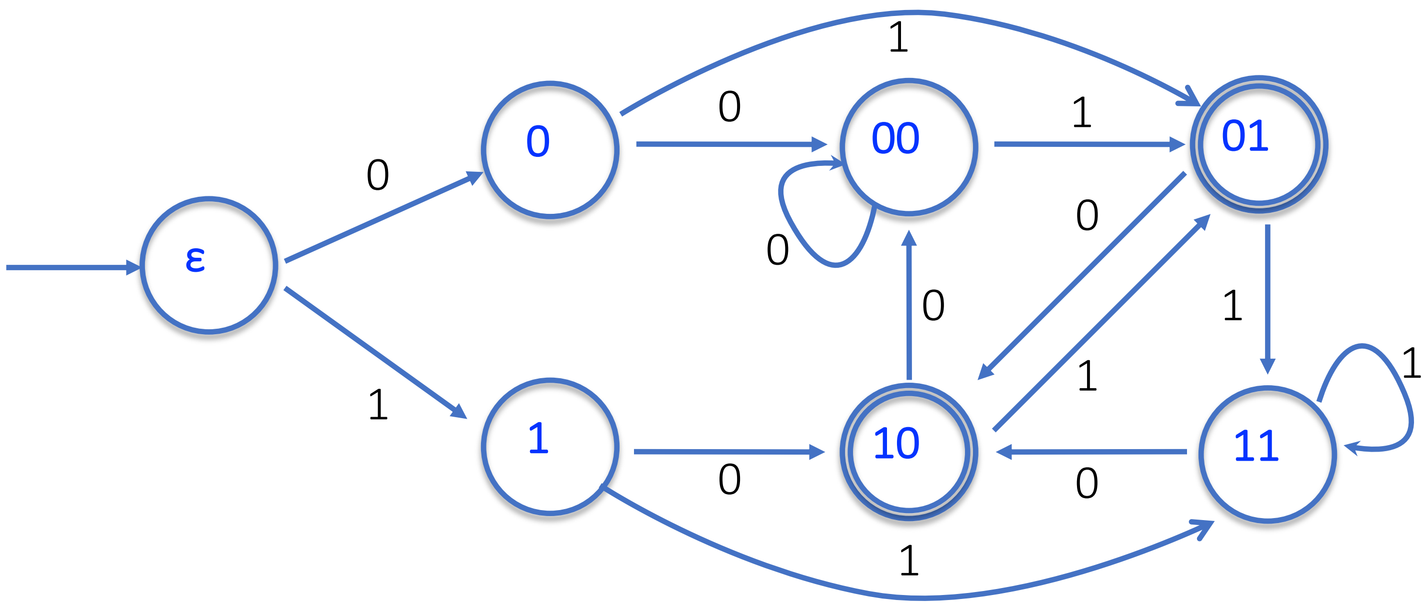
Question: what is L(M)?

Answer: strings with odd number of ones!

Construction Exercise

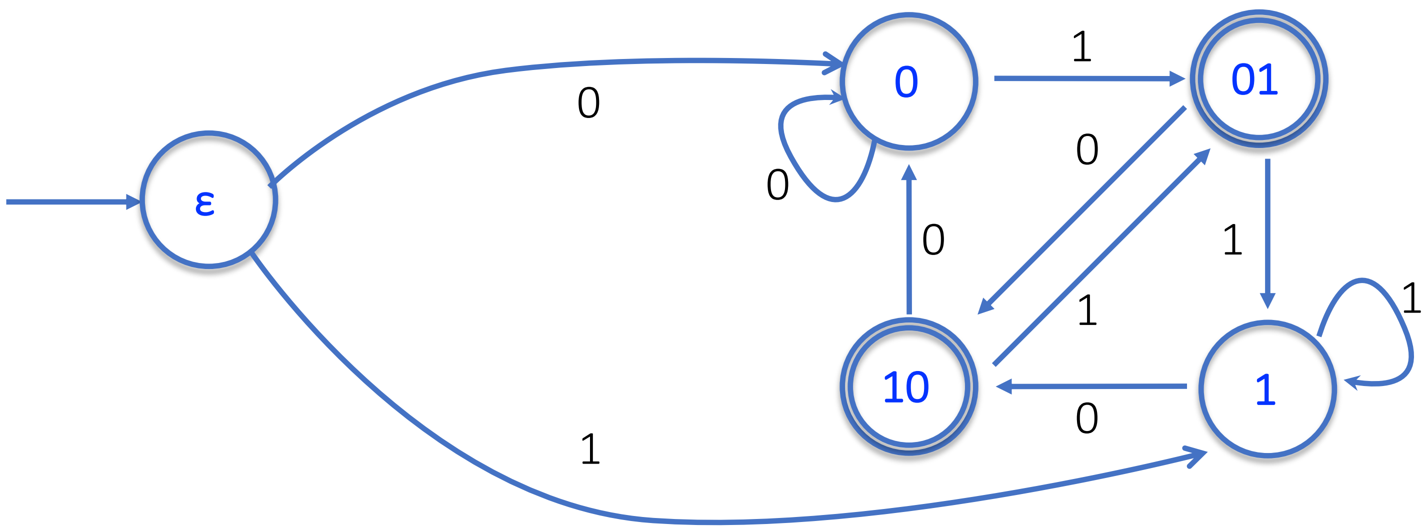
- $L(M) = \{w \mid w$ ends in 01 or 10 $\}$
- Is it regular??
- What should be in the memory?
- Last two bits seen.
  Possible values: ε, 0, 1, 00, 01, 10, 11

(0+1)*01+(0+1)*10



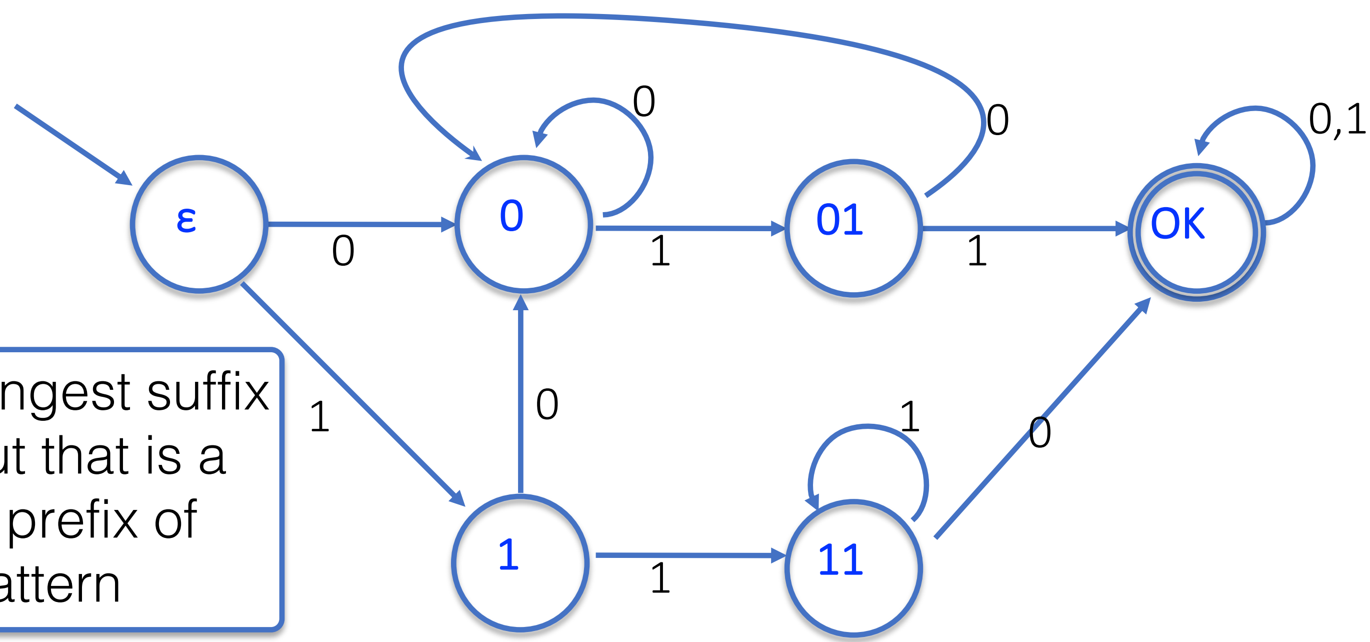
Construction Exercise

- $L(M) = \{w \mid w$ ends in 01 or 10 $\}$
- Is it regular??
- What should be in the memory? Last two bits seen. Possible values: ε, (0+00), (1+11), 01, 10



Construction Exercise

- $L(M) = \{w \mid w$ contains 011 or 110 $\}$
- Brute force: Enough to remember last 3 symbols (8+4+2+1=15 states). Stay at accepting states if reached.
- "Clever" construction: Enough to remember valid prefixes. States: ε, 0, 1, 01, 11, OK (can forget everything else)



State: longest suffix of input that is a valid prefix of pattern