

# CSC 236 Lecture 10: Formal Language Theory 2

Harry Sha

July 26, 2023

# Today

Review

Equivalence

Regex

Equivalence of NFAs and Regular Expressions (!)

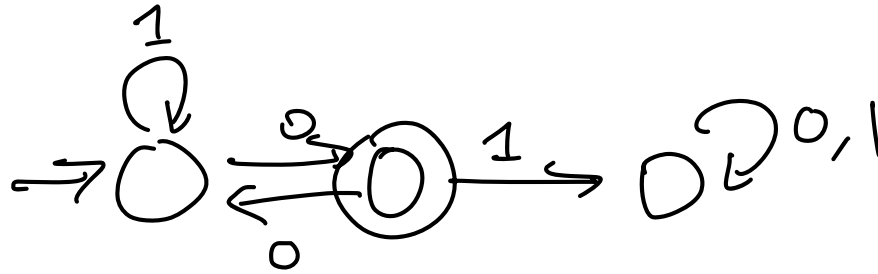
Review

Equivalence

Regex

Equivalence of NFAs and Regular Expressions (!)

# Review



- DFAs
- A language  $A$  is **regular** iff there is a DFA  $M$  such that  $L(M) = A$

# Closure

Suppose  $A$  and  $B$  are regular languages, then

- $\overline{A}$  ,
- $A \cup B$  ,
- $A \cap B$  ,
- $AB$  ,
- $A^n$ , and
- $A^*$ ,

are also regular.

# Closure

$$\bar{A} = \Sigma^* \setminus A$$

Suppose  $A$  and  $B$  are regular languages, then

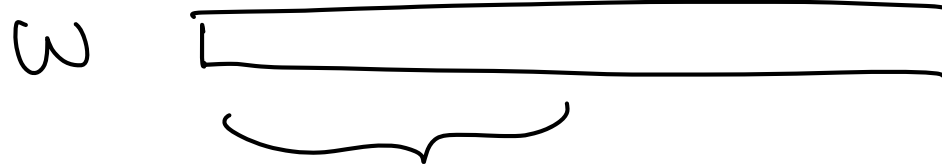
- $\bar{A}$  (by flipping states),  $accept \rightarrow reject$ ,  $reject \rightarrow accept$ .
- $A \cup B$  (by running two DFAs in parallel),
- $A \cap B$  (hw),
- $AB$  (today!),
- $A^n$  (today!), and
- $A^*$  (today!),

are also regular.

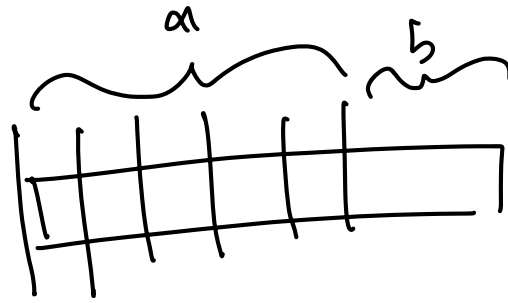
$AB$

$$w \in AB \iff w = ab \text{ for some } a \in A, b \in B.$$

The same trick of running the DFAs in parallel doesn't work.



AB



$$AB = \{ab : a \in A, b \in B\}.$$

The same trick of running the DFAs in parallel doesn't work.  
How would you write code to solve this?

is in  $AB(w)$ :

```
for i = 0 ... n :  
  | if is in A(w[:i]) and is in B(w[i:]):  
  |   | return true.
```

```
return false.
```



# Review: Nondeterminism

“If any sequence of **choices** leads to an accept state, accept”

“Reject only if every sequence of choices leads to reject”

# Choices

- If there are multiple arrows marked with the read character, choose which arrow to take.
- If there is an  $\epsilon$ -transition, choose whether or not to take it!

Review

Equivalence

Regex

Equivalence of NFAs and Regular Expressions (!)

# Equivalence of DFAs and NFAs

## Theorem

*Let  $A$  be any language. There is a DFA  $M$  such that  $A = L(M)$  if and only if there exists a NFA  $N$  such that  $A = L(N)$ .*

# Equivalence of DFAs and NFAs

## Theorem

*Let  $A$  be any language. There is a DFA  $M$  such that  $A = L(M)$  if and only if there exists a NFA  $N$  such that  $A = L(N)$ .*

The forward direction is true since every DFA is already a NFA (it just doesn't use the extra features). The backwards direction might be surprising to you!

# Equivalence of DFAs and NFAs

## Theorem

*Let  $A$  be any language. There is a DFA  $M$  such that  $A = L(M)$  if and only if there exists a NFA  $N$  such that  $A = L(N)$ .*

The forward direction is true since every DFA is already a NFA (it just doesn't use the extra features). The backwards direction might be surprising to you!

Essentially it says all the extra features we give NFAs don't in fact give them more "power". *If I can solve it with an NFA, I can also solve it with a DFA.*

# NFA $\Rightarrow$ DFA

Subset construction.

more details:

check out Sipser

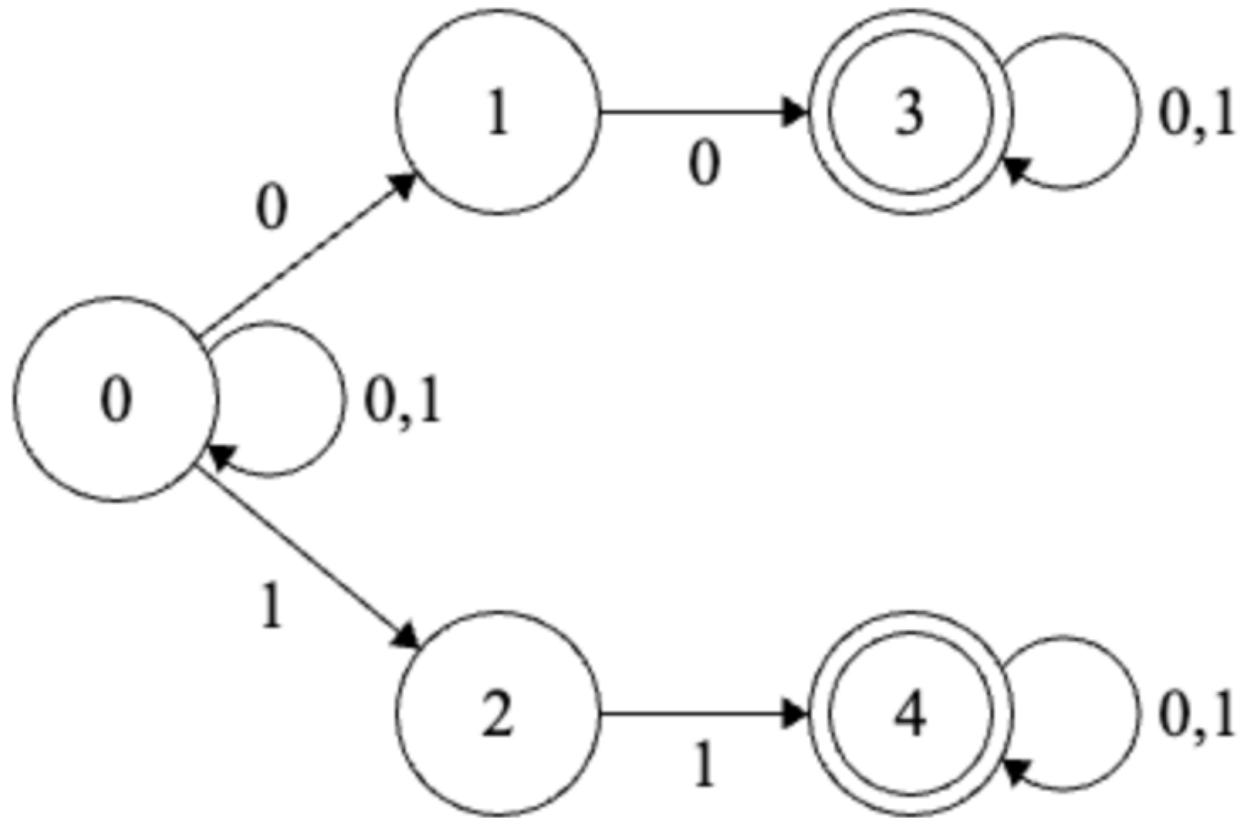
We need to simulate a NFA  $N$  with a DFA  $M$

High level idea:

- Have a state in  $M$  for every subset of states in  $N$ .
- Let  $S, T$  be two states in  $M$ . Then  $S$  transitions to  $T$  reading  $\sigma$  if some choice allows me to go from a state in  $S$  to a state in  $T$ . The choice includes an unlimited number of  $\epsilon$  transitions.
- The accepts states are the subsets that contain accept states.
- The start state is the subset containing the start state in  $N$  and all states reachable from the start state using an  $\epsilon$  transition

Intuitively, if I'm at state  $S$  after having read  $w$ .  $S$  should contain all the possible states I could have been in  $N$ .

## Example

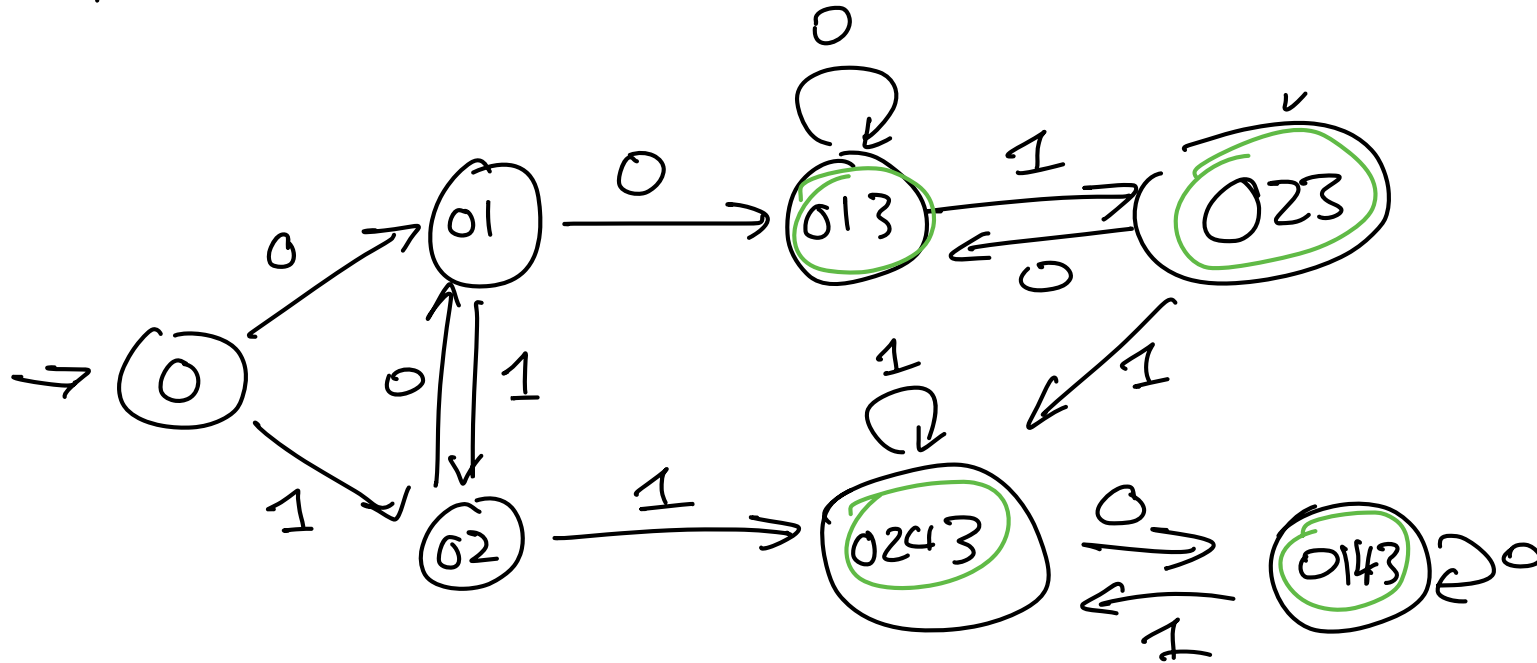
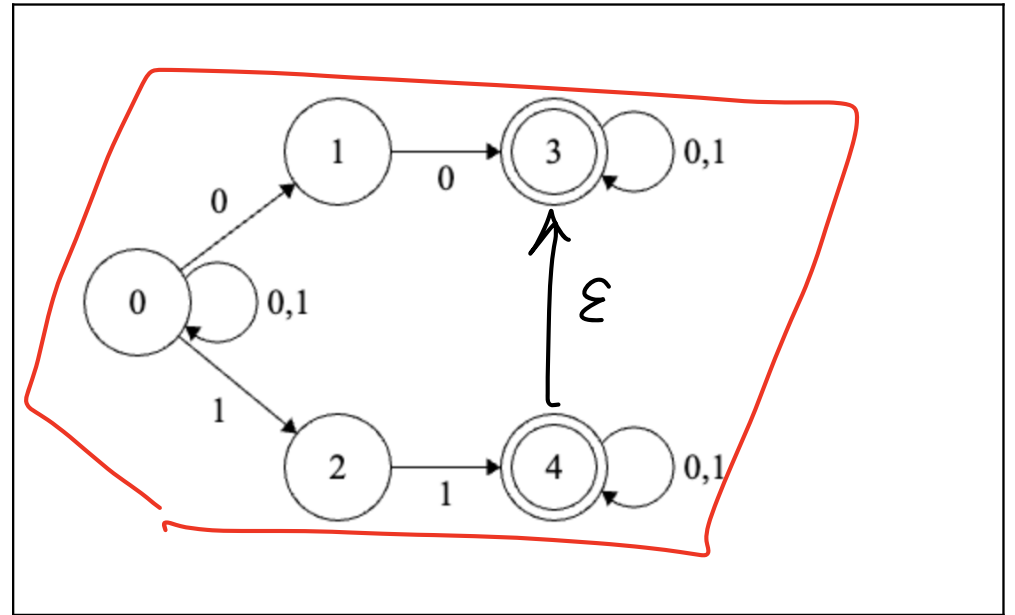




# Example

represent  $\{0,1,2\}$  by 012

	0	1
0 :	01	02
1 :	3	1
2 :	1	4,3
3 :	3	3
4 :	4,3	43



# Takeaway

For every NFA  $N$ , there exists a (potentially huge) DFA  $M$  such that  $L(N) = L(M)$ .

# Regular Languages (again)

The following are equivalent

- $A$  is regular
- There is a DFA  $M$  such that  $L(M) = A$
- There is a NFA  $N$  such that  $L(N) = A$

# Equivalent models of computation



The more complex a model of computation, the easier it is to use. I.e. NFAs are more complex so finding NFAs for a language is easier than finding DFAs for language.

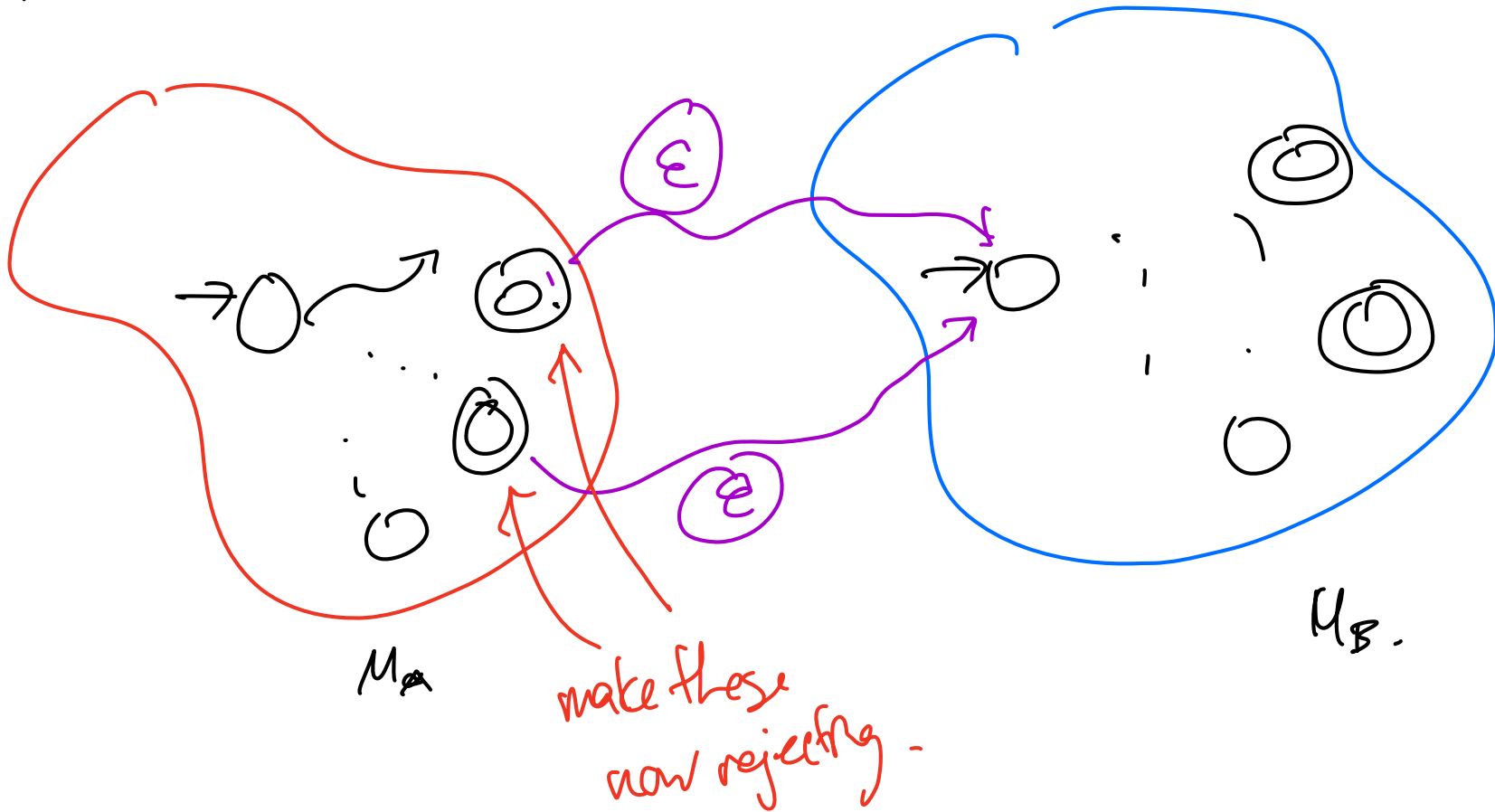
The more limited a model of computation, the easier it is to prove things about. Since DFAs are so restrictive, we can prove a lot of nice properties about them!

Since these two models of computation are equivalent, we can pick the right model for the right situation!

$AB$  is regular

suppose I have an DFA  $M_A$  st  $L(M_A) = A$   
 $M_B$  st  $L(M_B) = B$ .

$M$



$\subseteq :$

Claim  $L(M) = AB$ .

$AB$

This NFA **guesses** when the string in  $B$  should start. It starts in  $M$  and can only move to  $N$  from accepts states in  $M$  since we need the first part of the string to be part of  $A$ .

$A^n$

Suppose  $A$  was regular, how would you show that  $\forall n \in \mathbb{N}. A^n$  is regular?

By induction!

$$A^* = A^0 \cup A^1 \cup A^2 \dots$$

if  $A$  is regular,  $A^*$  is regular.

$M_A$  is a DFA for  $A$



$a_1 a_2 a_3 \dots a_k$   
 where each  $a_i \in A$



Review

Equivalence

Regex

Equivalence of NFAs and Regular Expressions (!)

# Regular Expressions

Regular expressions are all about matching patterns in strings.

Examples:

- Working with the terminal (live)
- valid email checking

# Regular Expressions - Formally

$$\Sigma = \{a, b\}$$

Let  $\Sigma$  be an alphabet. Define the set of regular expressions  $\mathcal{R}_\Sigma$  recursively as follows.

$a^* | baa$

$\mathcal{R}_\Sigma$  is the smallest set such that

- $\emptyset \in \mathcal{R}_\Sigma$
- $\epsilon \in \mathcal{R}_\Sigma$
- $a \in \mathcal{R}_\Sigma$  for each  $a \in \Sigma$

} Base cases

$\bigcirc R \in \mathcal{R}_\Sigma \implies (R)^* \in \mathcal{R}_\Sigma$

$\bigcirc R_1, R_2 \in \mathcal{R}_\Sigma \implies (R_1 R_2) \in \mathcal{R}_\Sigma$

$\bigcirc R_1, R_2 \in \mathcal{R}_\Sigma \implies (R_1 | R_2) \in \mathcal{R}_\Sigma$

} "Inductive steps".

Note that  $\mathcal{R}_\Sigma$  is defined inductively.

# Operator Precedence

\* comes before concatenation which comes before |.

# Operator Precedence

\* comes before concatenation which comes before |.

For example,  $a^*b|bc$  means  $((a^*)b)|(bc)$ .

# Language of a Regular Expression

↪ the set of strings the expression matches.

The language of a regular expression  $R$ , denoted  $L(R)$  is the set of strings that  $R$  matches. Formally,

- $L(\emptyset) = \emptyset$
- $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\}$  for  $a \in \Sigma$
- $L(R^*) = L(R)^*$  for  $R \in \mathcal{R}_\Sigma^*$
- $L(R_1 R_2) = L(R_1) L(R_2)$  for  $R_1, R_2 \in \mathcal{R}_\Sigma$
- $L(R_1 | R_2) = L(R_1) \cup L(R_2)$  for  $R_1, R_2 \in \mathcal{R}_\Sigma$

# Examples

What are the languages of the following regular expressions?

(Assume the alphabet is  $\{0, 1\}$ )

- $((0|1)(0|1)(0|1))^*$

$$\begin{aligned} \mathcal{L}((0|1)(0|1)(0|1))^* &= \mathcal{L}((0|1)(0|1)(0|1))^* \\ &= (\{0,1\} \{0,1\} \{0,1\})^* \\ &= (\text{any string of length 3})^* \end{aligned}$$

# Examples

~~ends with 00 and~~

# of 0s in the string is even.  
and it ends with 0.

What are the languages of the following regular expressions?  
(Assume the alphabet is  $\{0, 1\}$ )

- $((0|1)(0|1)(0|1))^*$

1010

- $(1^*010)^*$

111001110 X

100110011 X

001001001 X

00000 ✓

ε ✓

→ 101101 ✓

1111111 X



# Examples

What are the languages of the following regular expressions?  
(Assume the alphabet is  $\{0, 1\}$ )

- $((0|1)(0|1)(0|1))^*$

- $(1^*01^*0)^*$

- $(0|1)^*1(0|1)(0|1)$

3rd last chr is a 1.

110110 ✓

01000 ✗

00 ✗

$\epsilon$  ✗

01111 ✓

0001011 ✗

# Examples

What are the languages of the following regular expressions?  
(Assume the alphabet is  $\{0, 1\}$ )

- $((0|1)(0|1)(0|1))^*$
- $(1^*01^*0)^*$
- $(0|1)^*1(0|1)(0|1)$
- $(0|1)^*010(0|1)^*$

010 ✓

111 ✗

0111010000 ✓

$\epsilon$  ✗

contains 010.

# Examples

What are the languages of the following regular expressions?  
(Assume the alphabet is  $\{0, 1\}$ )

- $((0|1)(0|1)(0|1))^*$
- $(1^*01^*0)^*$
- $(0|1)^*1(0|1)(0|1)$
- $(0|1)^*010(0|1)^*$
- $(0|\epsilon)1^*$

# Examples

What are the languages of the following regular expressions?  
(Assume the alphabet is  $\{0, 1\}$ )

- $((0|1)(0|1)(0|1))^*$
- $(1^*01^*0)^*$
- $(0|1)^*1(0|1)(0|1)$
- $(0|1)^*010(0|1)^*$
- $(0|\epsilon)1^*$

# Shorthand

$RRR \dots R$   
~

If  $R$  is a regex, and  $m \in \mathbb{N}$ , then

- $R^m$  means  $m$  copies of  $R$ .
- $R^+ = RR^*$ , i.e. at least one copy of  $R$ .
- $R^{m+} = R^m R^*$  means at least  $m$  copies of  $R$ .

If  $S = \{s_1, s_2, \dots, s_n\}$  is a <sup>finite</sup> set of strings, then  $S$  is shorthand for  $s_1 | s_2 | \dots | s_n$ . A common one is to use the alphabet,  $\Sigma$ .

# Examples

Write regular expressions for the following languages

- Starts with 010.

$010(011)^*$

$010\Sigma^*$

# Examples

$$\Sigma = \{0, 1\}.$$

Write regular expressions for the following languages

- Starts with 010.
- The second and the second last letter of  $w$  are the same.

$$(012) \left( \underline{0(012)^*0} \mid \underline{1(012)^*1} \right) (01)^*$$

$$01 = \bar{z}$$

$$\Sigma \bar{0} \bar{z}^* \bar{0} \bar{z} \mid \bar{z} \bar{1} \bar{z}^* \bar{1} \bar{z}$$

# Examples

Write regular expressions for the following languages

- Starts with 010.
- The second and the second last letter of  $w$  are the same.
- Contains 110.

$\Sigma^* 110 \Sigma^*$



# Examples

Write regular expressions for the following languages

- Starts with 010.
- The second and the second last letter of  $w$  are the same.
- Contains 110.
- 1s always occur in pairs.

$(0111)^*$

# Examples

Write regular expressions for the following languages

- Starts with 010. ~
- The second and the second last letter of w are the same.
- Contains 110.
- 1s always occur in pairs.
- Doesn't contain more than four 1s in a row.

}

)

Review

Equivalence


Regex

Equivalence of NFAs and Regular Expressions (!)

# Equivalence of NFAs and Regex

For every regular expression  $R$ , there exists a NFA  $N$  such that  $L(R) = L(N)$ .

# Regex $\rightarrow$ NFA

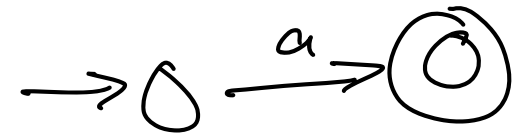
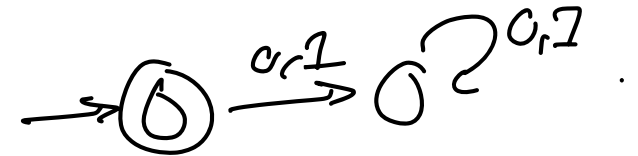
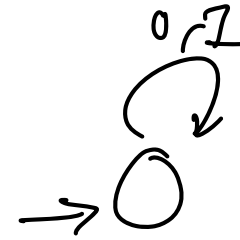
structural  
induction  


We want to show  $\forall R \in \mathcal{R}_\Sigma$ , there is an equivalent NFA. How do we do this?

# Regex $\rightarrow$ NFA

Base cases:

$$\begin{aligned}\emptyset &: L(\emptyset) = \emptyset \\ \epsilon &= \{\epsilon\} \\ a &= \{a\}.\end{aligned}$$



## Regex $\rightarrow$ NFA

Inductive step, let  $R_1, R_2 \in \mathcal{R}_\Sigma$ , and assume they have equiv. NFA  $N_1, N_2$ .

WTS:  $R_1 R_2, R_1^*, R_1 \mid R_2$  have equivalent NFAs

$$L(R_1 R_2) = L(R_1) L(R_2) \quad \checkmark \quad \text{b/c reg. lang. are closed under concatenation.}$$

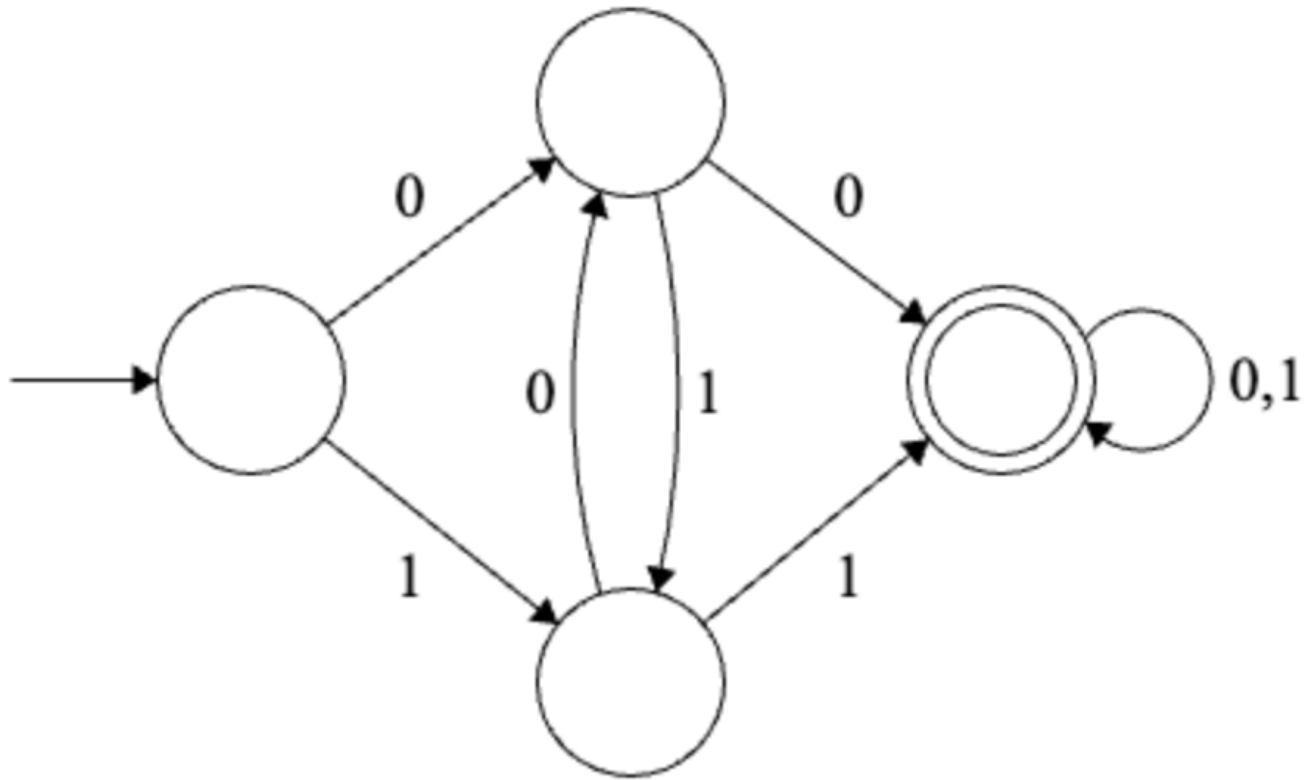
$$L(R_1 \mid R_2) = L(R_1) \cup L(R_2).$$

$$L(R_1^*) = L(R_1)^*$$

NFA  $\rightarrow$  Regex



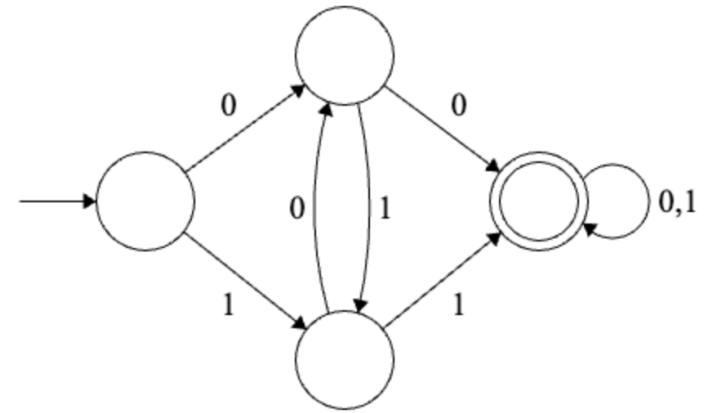
## Example <sup>1</sup>



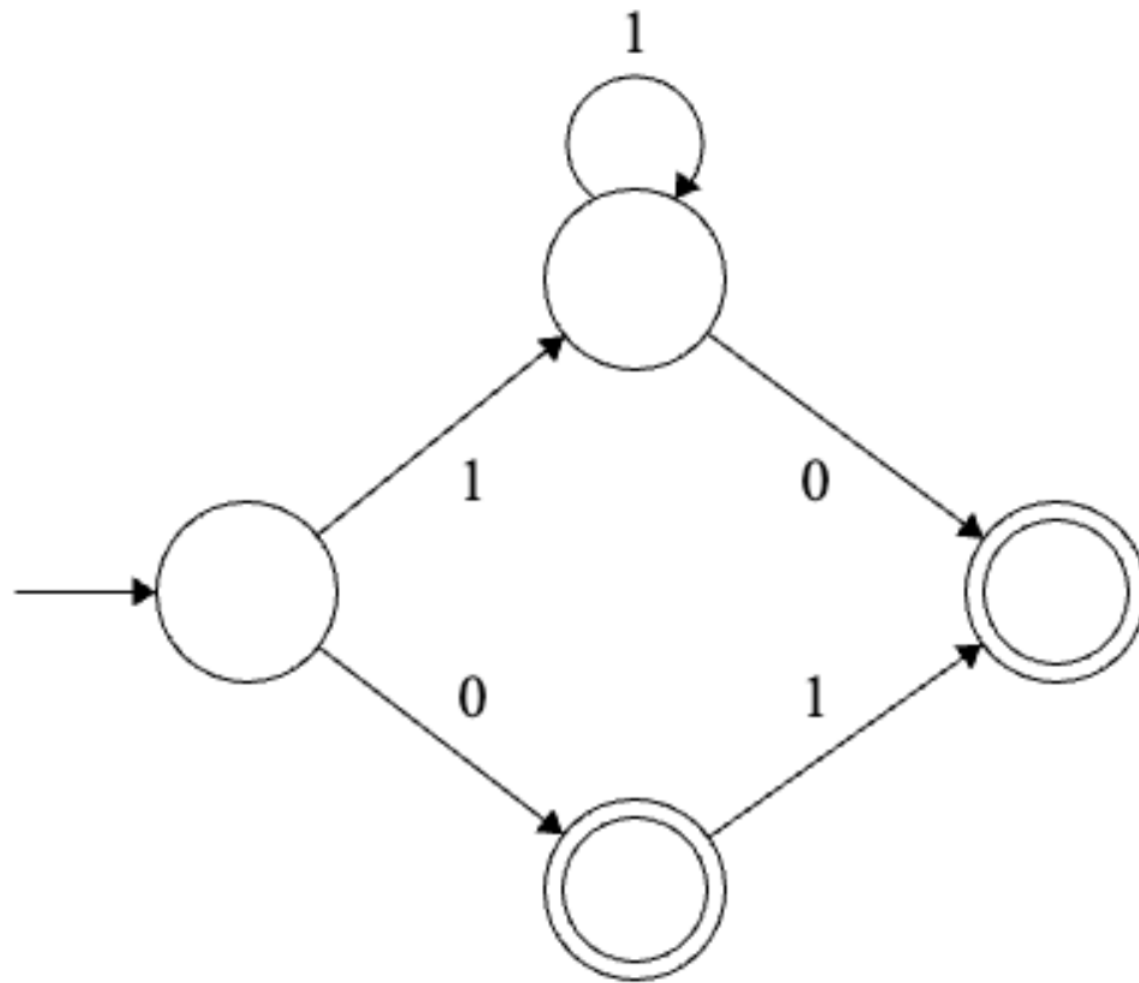
---

<sup>1</sup>Reference: CSC236 2022 Fall

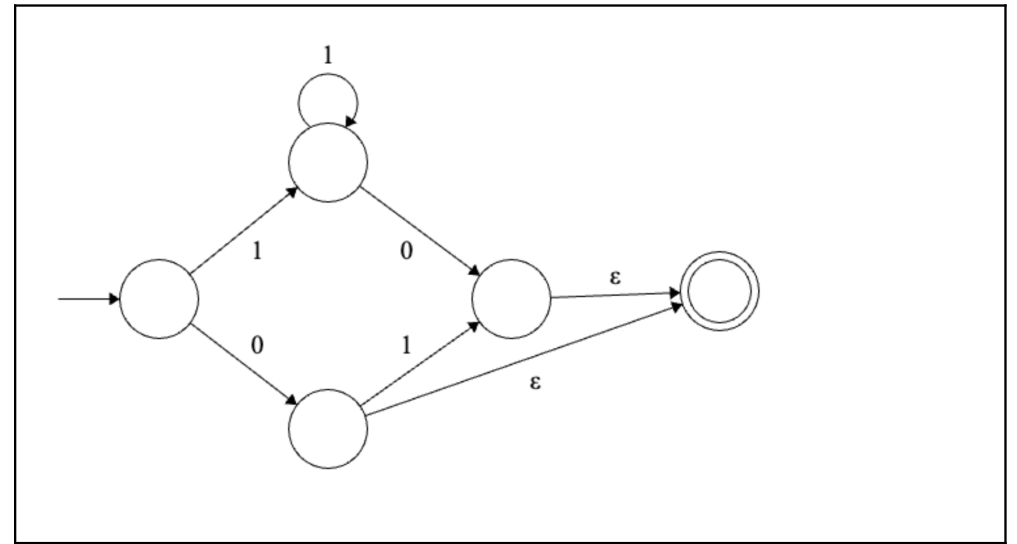
# Example



## Example 2

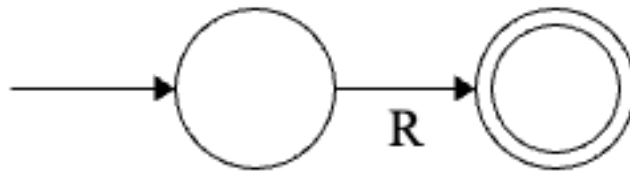


## Example 2



# Sketch

- Alter the NFA so that there's just one accept state (using  $\epsilon$  transitions).
- Iteratively rip out states, replacing transitions with regular expressions until you have something that looks like



$R$  is the equivalent regular expression.

## “Ripping” out states

For two states  $q_1, q_2$  with a transition between them, let  $f(q_1, q_2)$  be the regular expression labelling the transition.

Here are the steps to rip out a state  $q$ .

1. Remove the loop: If there is a self loop on state  $q$ , for each state  $s$  with a transition into  $q$ , update the transition  $f(s, q) = f(s, q)f(q, q)^*$ .
2. Bypass  $q$ : for each path  $(s, q, t)$  of length 2 through  $q$ , update  $f(s, t) = f(s, t) \mid f(s, q)f(q, t)$ . Note that it is possible that  $s = t$ , in which case this step adds a loop.
3. Remove  $q$ .

# Regular Languages

The following are equivalent

- $A$  is regular
- There is a DFA  $M$  such that  $L(M) = A$
- There is a NFA  $N$  such that  $L(N) = A$
- There is a regular expression  $R$  such that  $L(R) = A$

# Consequences

If I ask you to show me a language  $A$  is regular, you can choose to give me either a DFA, NFA or a regular expression!

↑  
another: decompose as the union/concateration,  
\*, or  $\cap$  intersection etc. of  
other smaller regular languages and then  
apply closure.



# How to choose

- I typically use regular expressions for languages that seem to require some form of 'matching'. For example *contains 121* as a substring, or *ends with 11*. Regular expressions are typically faster to find and write out in an exam setting.
- I'll use NFAs when I can't easily figure out a regular expression for something. These are usually languages for which memory seems to be useful like the Dogwalk example from hw.
- Stuff involving negations also seems easier to do with NFAs than with regular expressions. For example, *contains the substring 011* is easy with regular expression, but *doesn't contain the substring 011* is a bit more complicated.