CMSC 330

Finite Automata

Administrative

Administrative

Quiz 2 Today

Midterm next Thursday: PL Concepts, Ruby, OCaml, NFA/DFA/CFG

Project 3 Released

Finite Automata

What Are Finite Automata?

Automata are data structures that accept/reject strings

Automata can be used to implement regular expressions

That’s what you’ll be doing in the project

Formal Definition - What does the example look like?

An automaton is a 5-tuple

Q - A finite set of states

Σ - A set of symbols, the alphabet

□ - A transition function

q_0 - The initial state

F - A set of final states

Example

Q - {0, 1, 2}

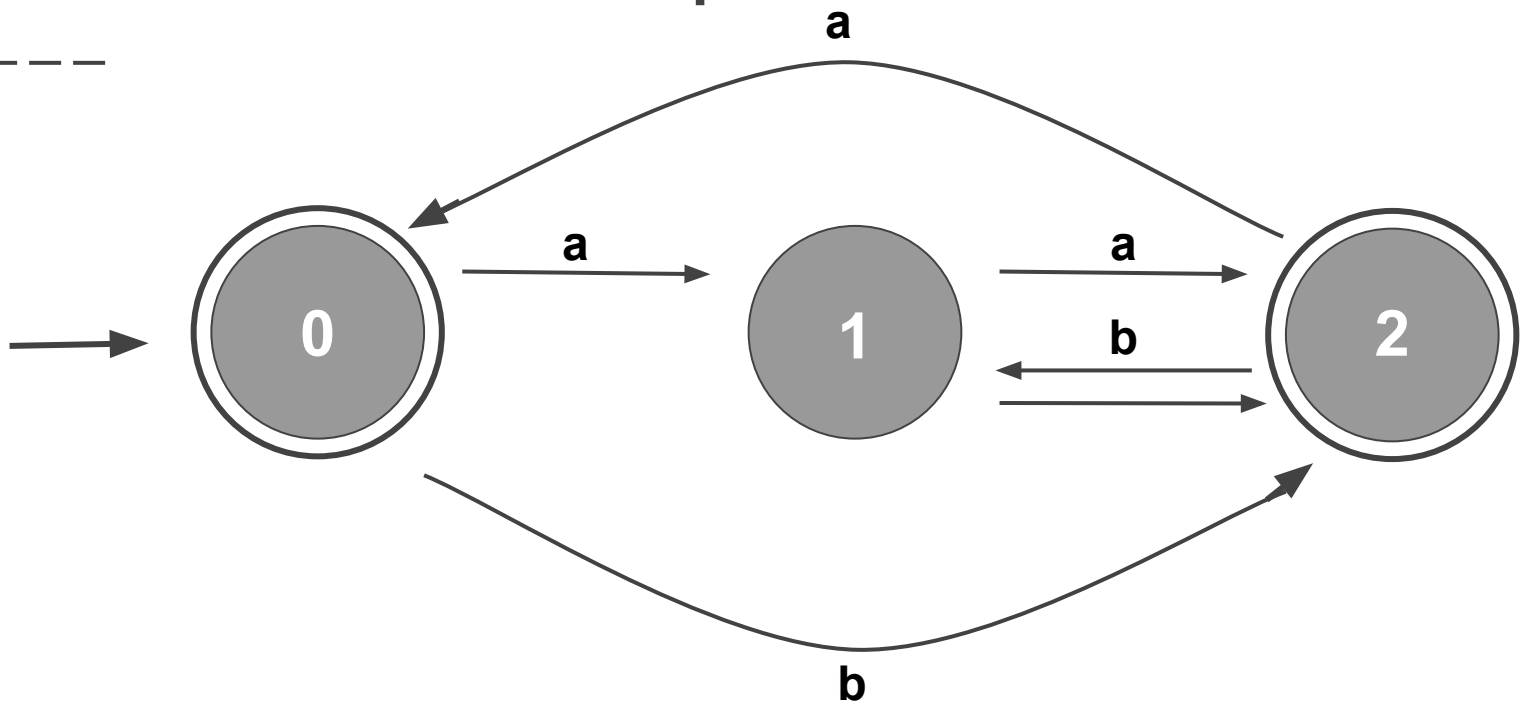
Σ - {a, b}

□ - {(0,a,1); (1,a,2); (2,a,0); (0,b,2); (1,b,2); (2,b,1)}

q_0 - 0

F - {0,2}

Formal Definition - Example



Formal Definition - Notes

Q - {0, 1, 2}

Σ - {a, b}

 - {(0,a,1); (1,a,2); (2,a,0); (0,b,2); (1,b,2); (2,b,1)}

q_0 - 0

F - {0,2}

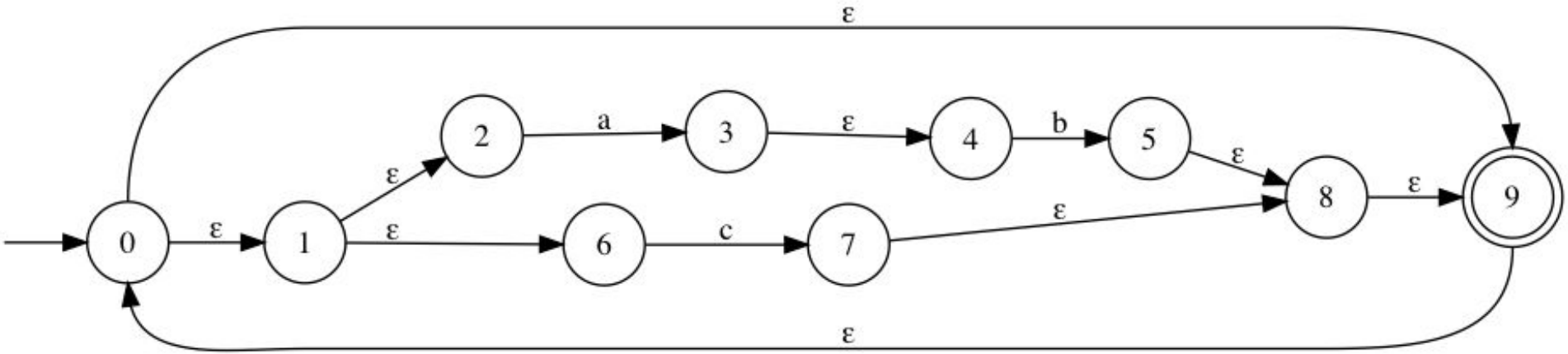
There can only be **one start state**, in this case 0, do not forget to label your start state on quizzes/exams!

There can be **multiple final states**, do not forget to label your final state(s) on quizzes/exams!

A string is **accepted** by the automata if it **ends in a final state.**

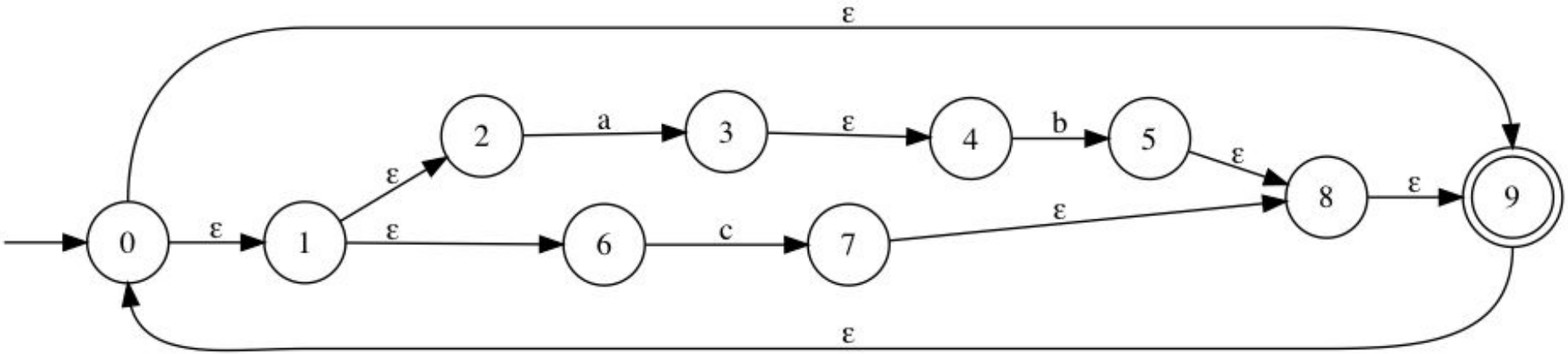
Example

Example



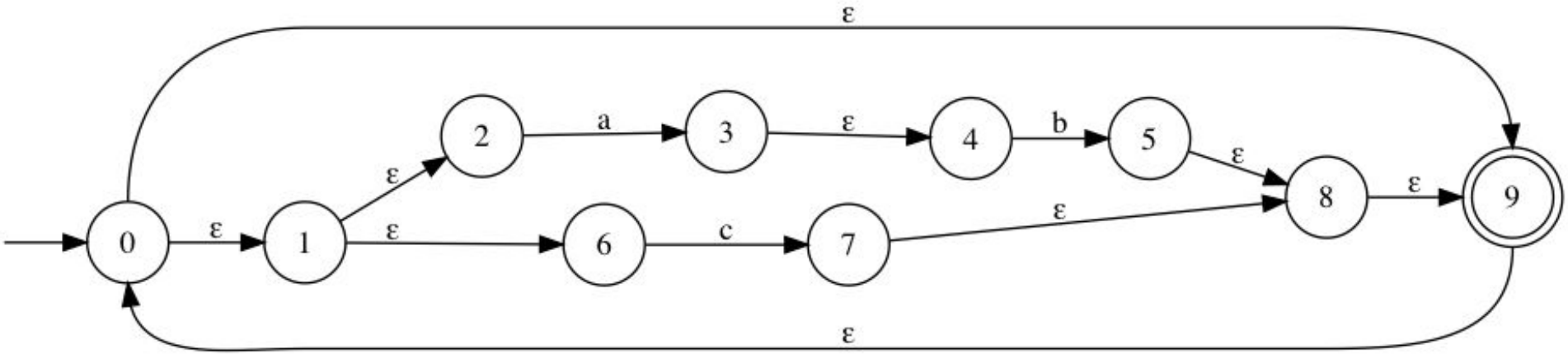
Accepted? “” (Empty String)

Example



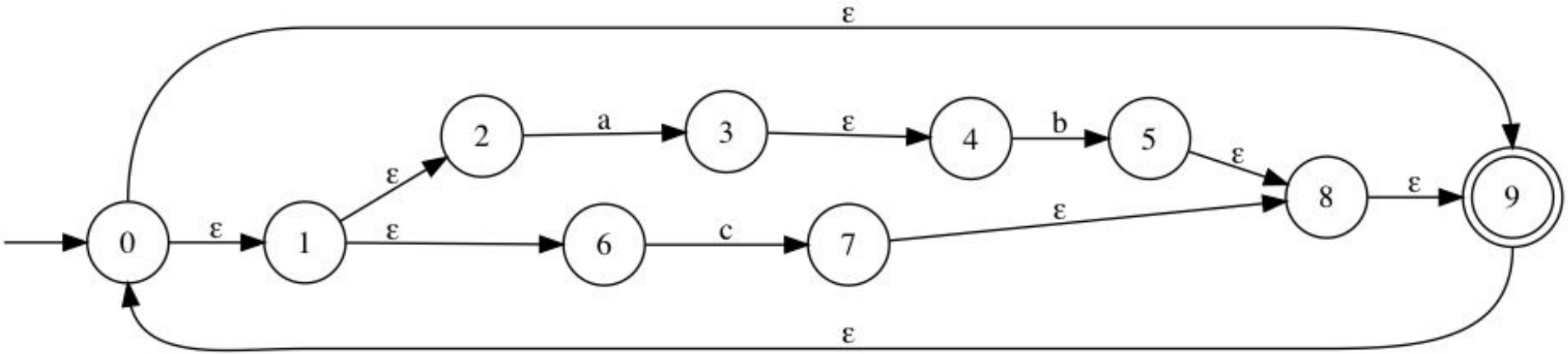
Accepted? ab

Example



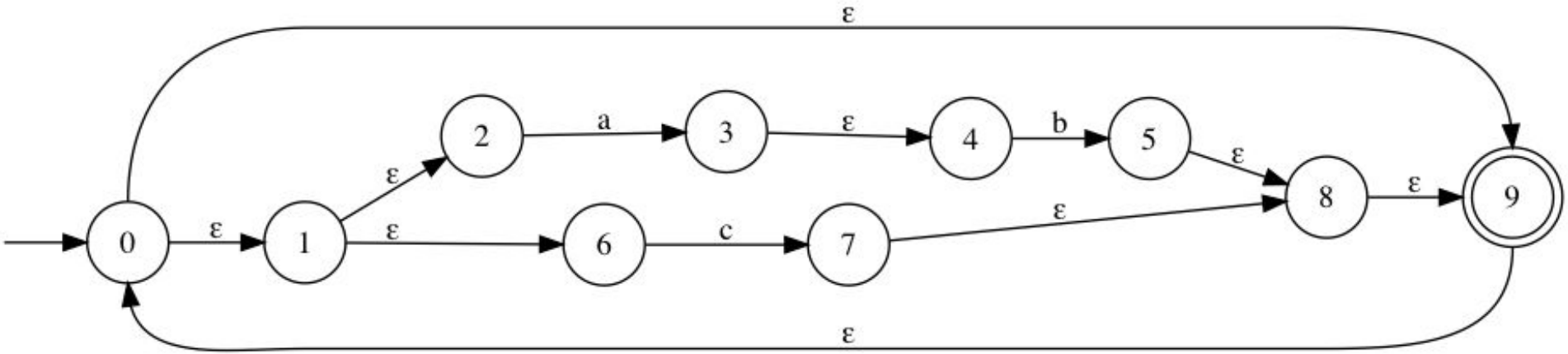
Accepted? ababab

Example



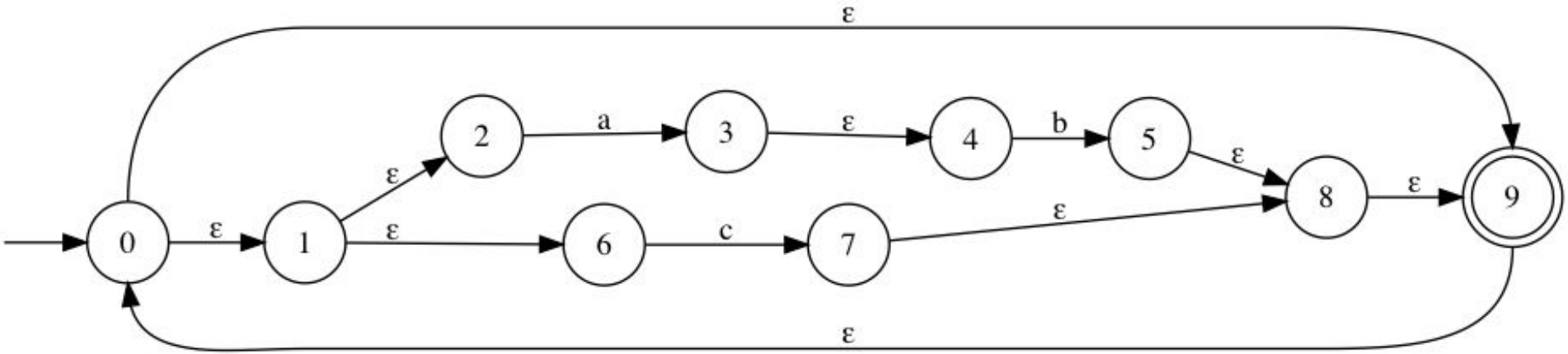
Accepted? abc

Example



What regular expression matches this automata?

Example



What regular expression matches this automata? (ab|c)*

Types of Finite Automata

Types of Finite Automata

Deterministic Finite Automata (DFA)

- Accepts if the string ends on a final state
- A special type of NFA

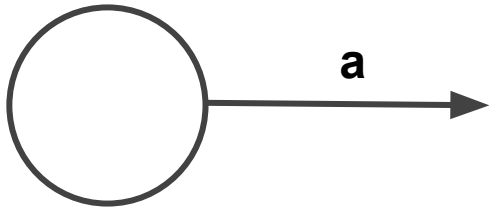
Nondeterministic Finite Automata (NFA)

- 0 or more steps for each character in the string
- Accepts if any valid path ends on a final state

Difference #1: Number of Transitions

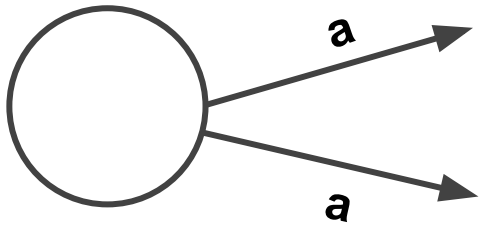
DFA

One transition per symbol



NFA

More than one transition per symbol



Difference #2: Types of Transitions

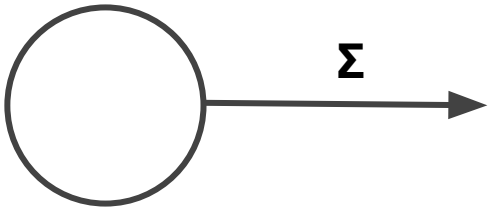
DFA

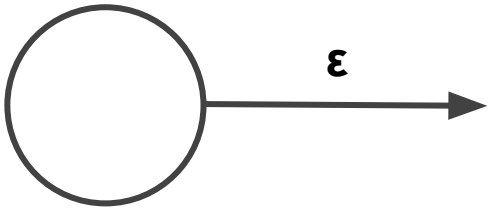
No transitions on empty string

(DFAs can match empty strings, but cannot transition on e)

NFA

May transition on empty string label





Difference #3: Accepting Strings

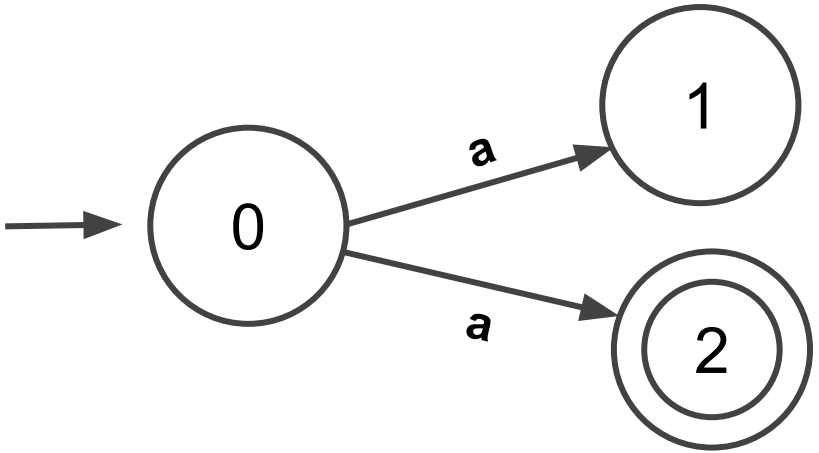
DFA

Accepts if the string ends on a final state

NFA

Accepts if at least one path ends on a final state

“a” would be accept because 0,2



Interlude: Project Talk

Project Talk

In your project, you will implement regular expressions

To do this, you will implement an NFA

(Note: Regex, NFA, DFA accept the same languages)

Project Talk - OCaml NFA

First you make an NFA, recall

Q - A finite set of states

Σ - A set of symbols, the alphabet

δ - A transition function

q_0 - The initial state

F - A set of final states

You will be given

q_0 - The initial state

F - A set of final states

δ - A transition function

How you define the NFA type is up to you, try to make something easy to match on!

Project Talk - E-Closure Function

Remembers that NFAs can transition on the empty string?

This is called an “ε-transition”

The ε-closure(S1) is the set of all states reachable from S1 using only ε-transitions

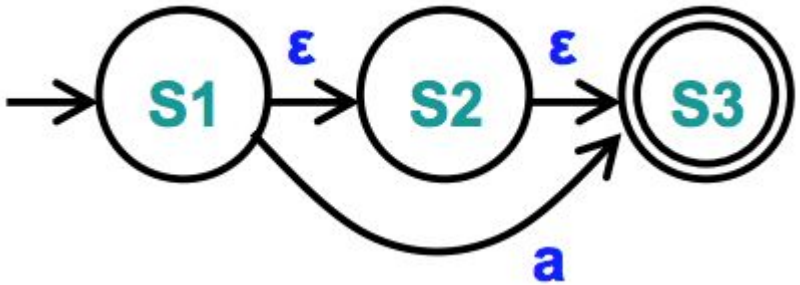
Note: ε-closure(S1) always includes S1

Project Talk - E-Closure 1

ε-closure(S1) =

ε-closure(S2) =

ε-closure(S3) =

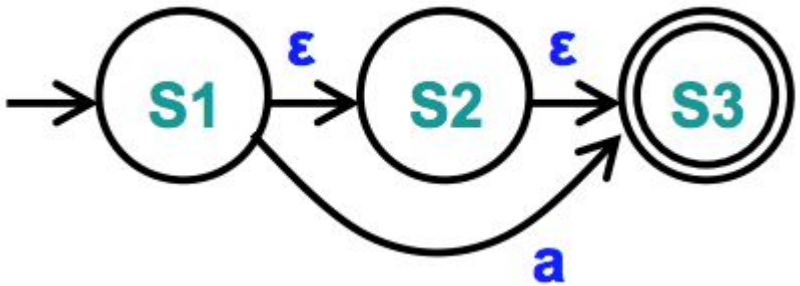


Project Talk - E-Closure 1

ε-closure(S1) = {S1, S2, S3}

ε-closure(S2) = {S2, S3}

ε-closure(S3) = {S3}



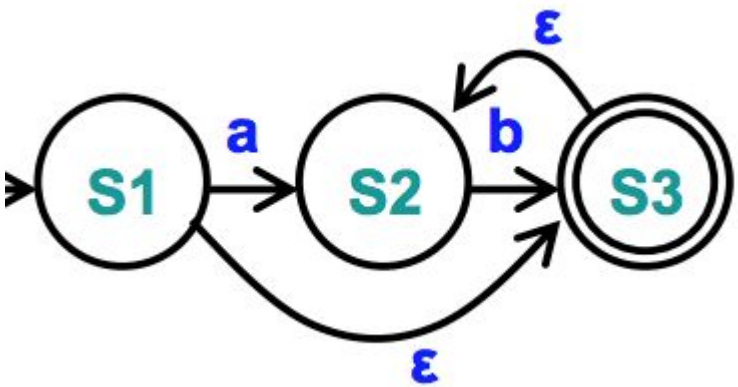
Project Talk - E-Closure 2

ε-closure(S1) =

ε-closure(S2) =

ε-closure(S3) =

ε-closure({S2,S3}) =



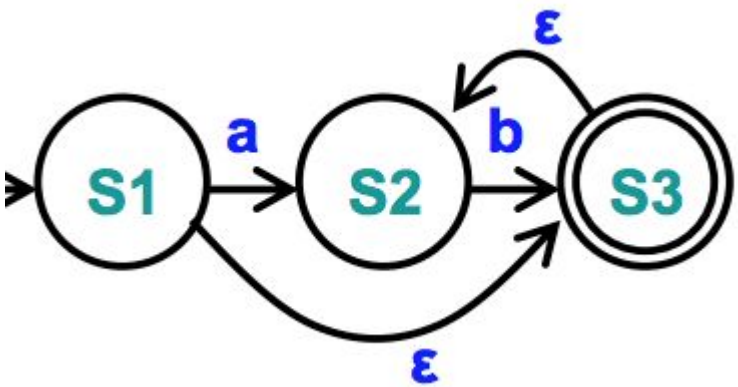
Project Talk - E-Closure 2

ε-closure(S1) = {S1, S2, S3}

ε-closure(S2) = {S2}

ε-closure(S3) = {S2, S3}

ε-closure({S2,S3}) = {S2,S3}



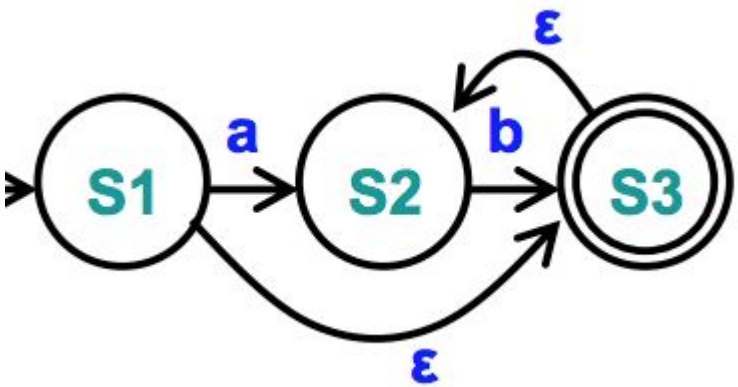
Project Talk - Move

---

move(S1,symbol)

Set of states reachable from S1 using exactly one transition on the symbol.

Note: move(S1,a) only includes S1 if S1 has a transition to itself on a.



Project Talk - Move

---

move(S1,a) =

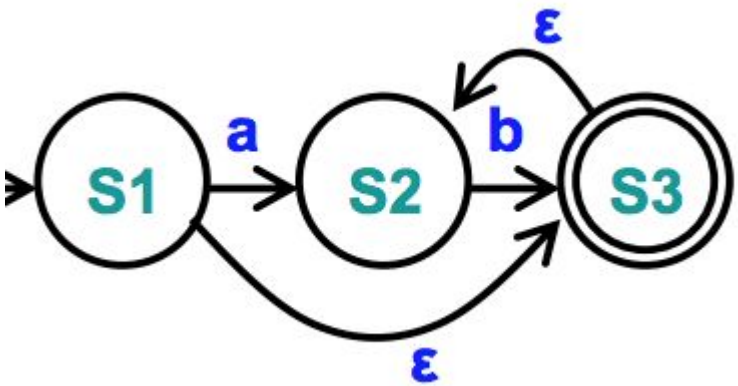
move(S1,b) =

move(S2,a) =

move(S2,b) =

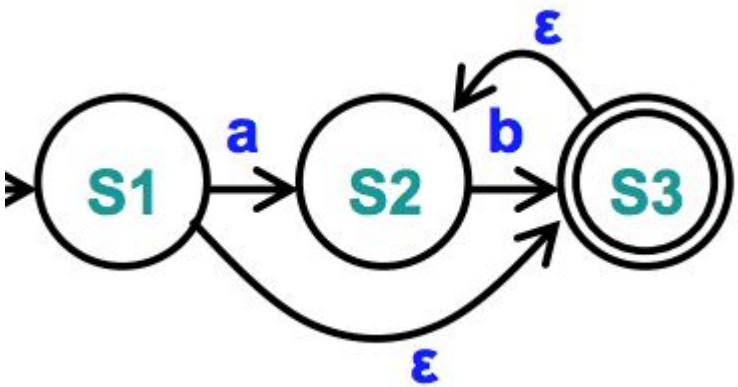
move(S3,a) =

move(S3,b) =



Project Talk - Move

```
move(S1,a) = {S2}

move(S1,b) = {}

move(S2,a) = {}

move(S2,b) = {S3}

move(S3,a) = {}

move(S3,b) = {}
```



Reductions

Introduction

There are a few operations we care about

- Regex to NFA
- NFA to DFA
- Minimizing a DFA
- Also: DFA to Regex, DFA Complement

Your project will ask you to turn a regex to an NFA

Regex to NFA

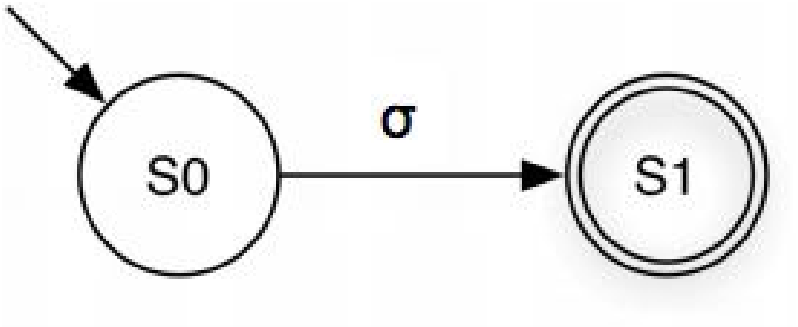
Regular Expressions to NFAs

Regular expressions are defined recursively

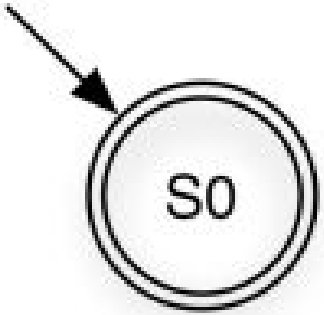
Know a number of base cases and inductive cases

This tells us how to build an NFA given a regex

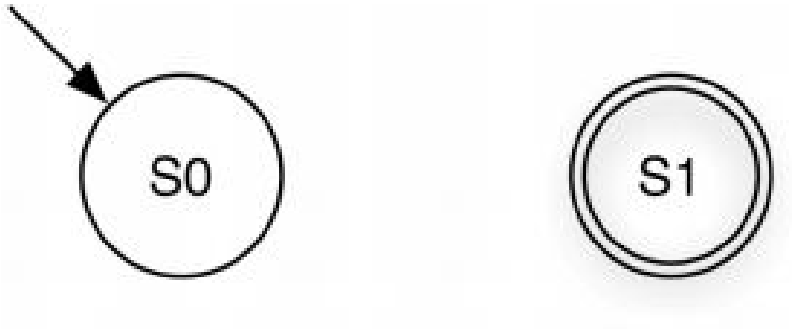
Base Case: A Symbol in the Language (σ in Σ)



Base Case: ε

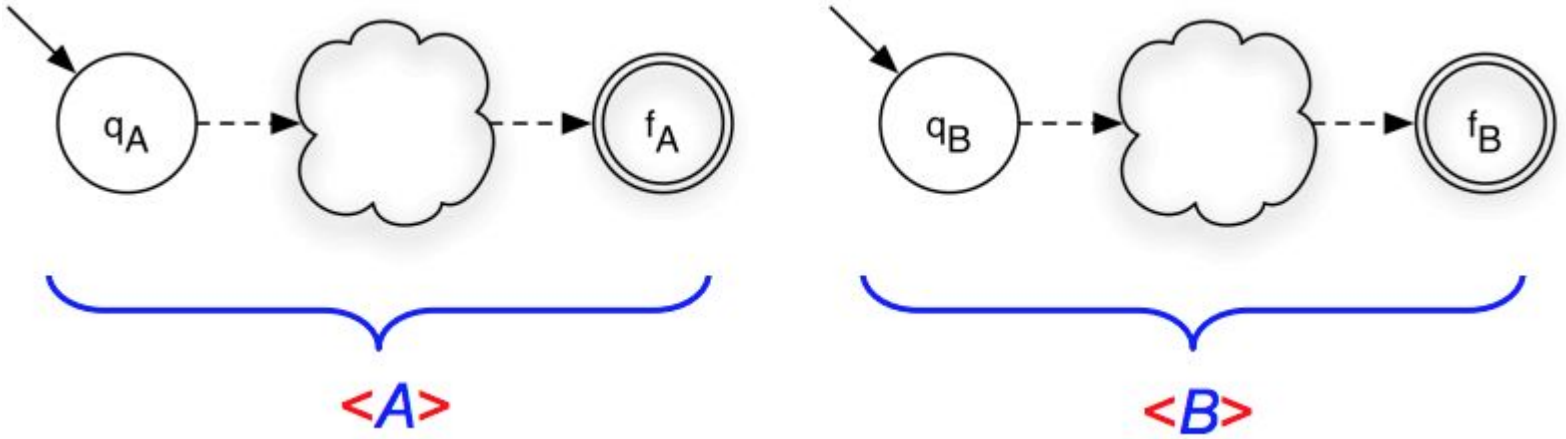


Base Case: ø



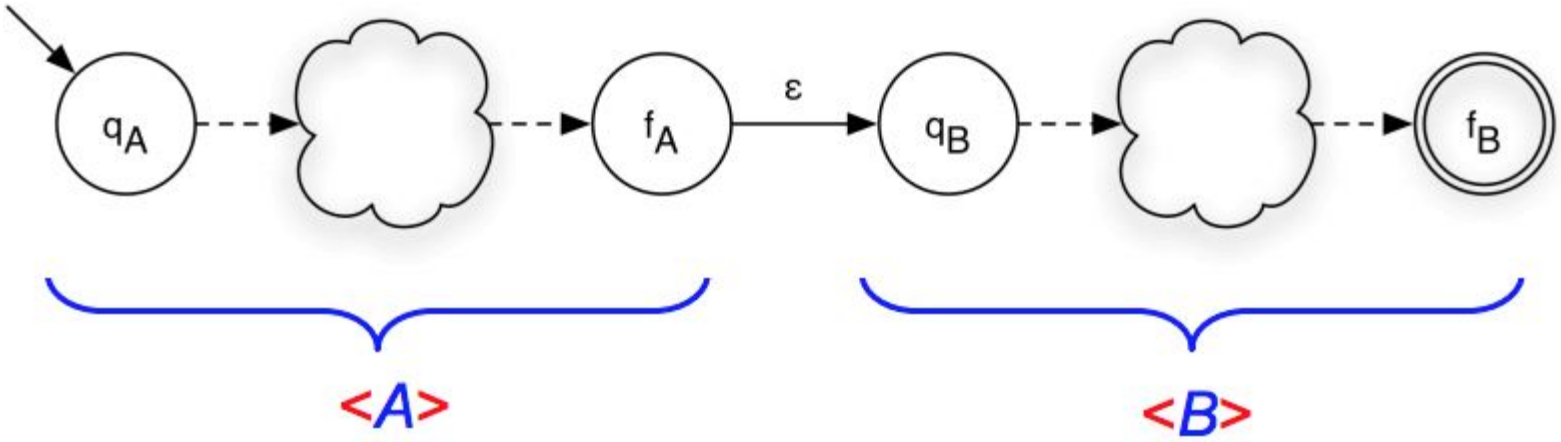
Concatenation: AB

A and B are each represented by an NFA



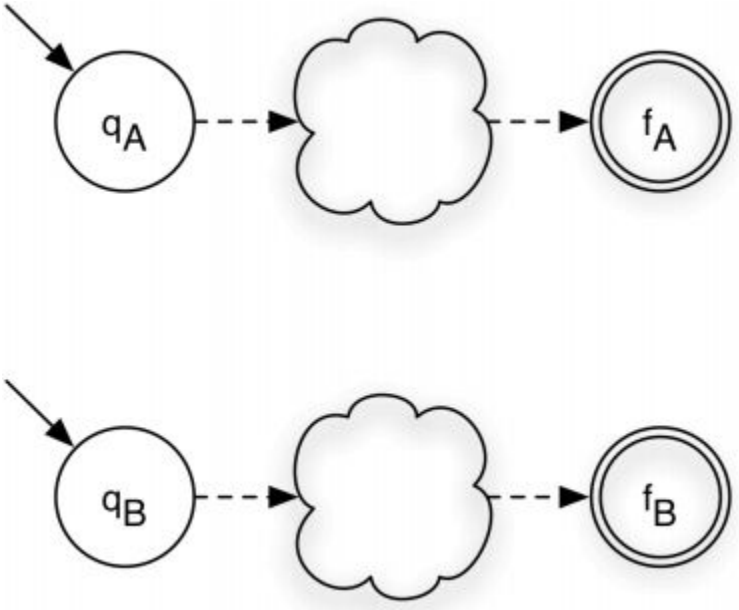
Concatenation: AB

To concatenate A and B, create a new NFA using the start of A, the final states of B, add an ε-transition from the final states of A to the start state of B.



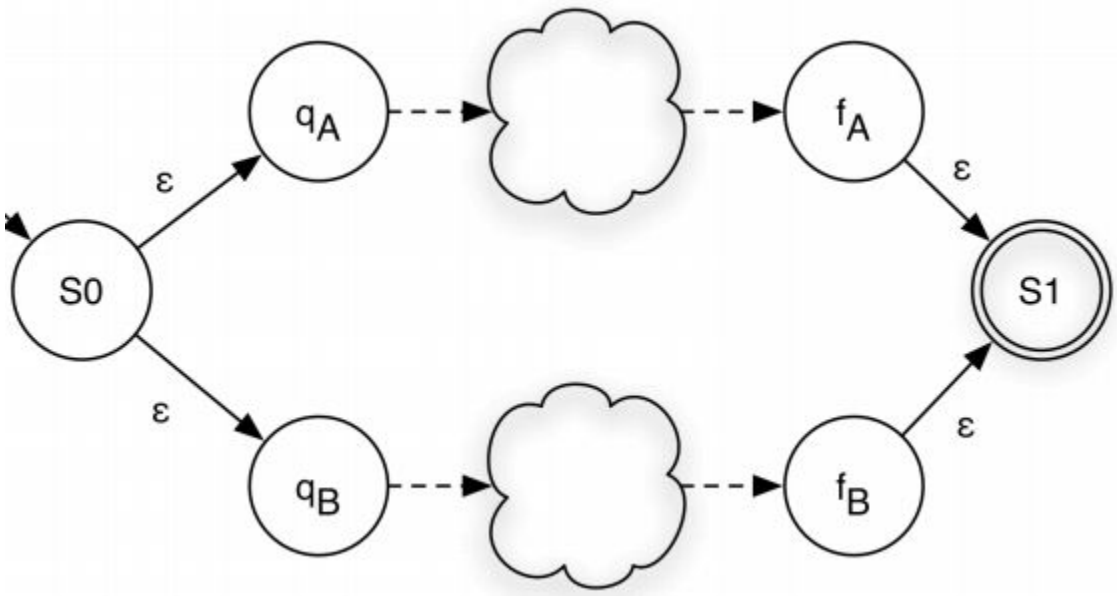
Union: A|B

A and B are each represented by an NFA



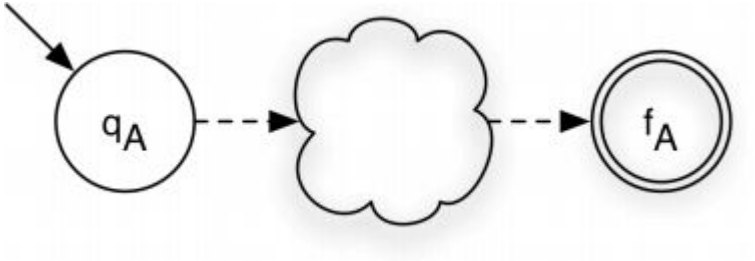
Union: A|B

Add a new start and end, add ε-transitions from new start to old starts, add ε-transitions from old finals to new finals.



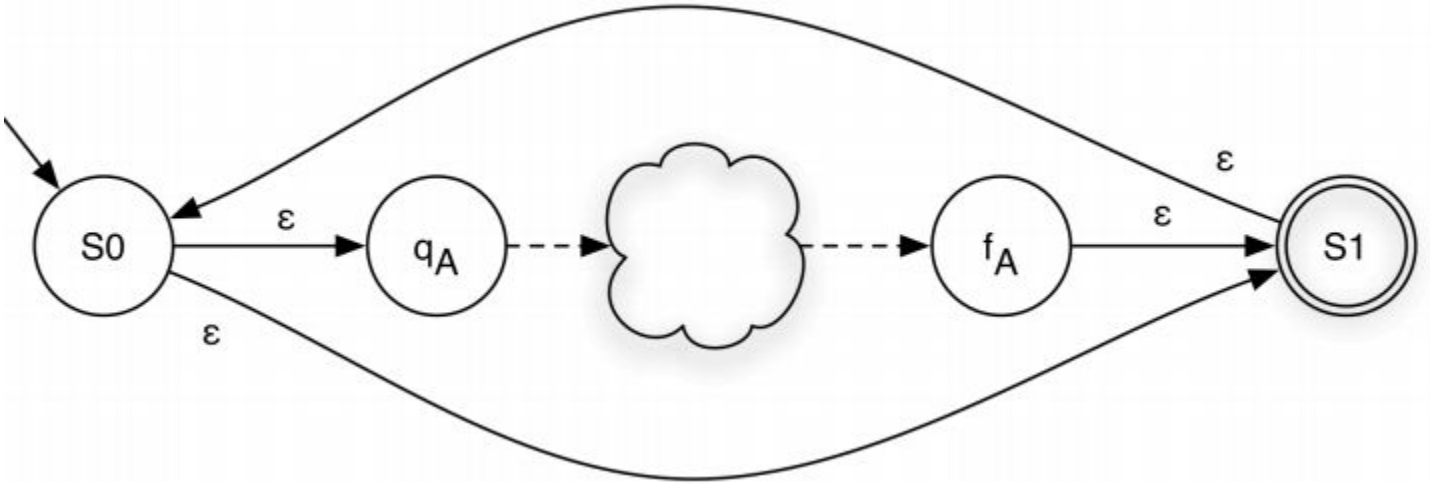
Closure: A*

A is represented by an NFA



Closure: A*

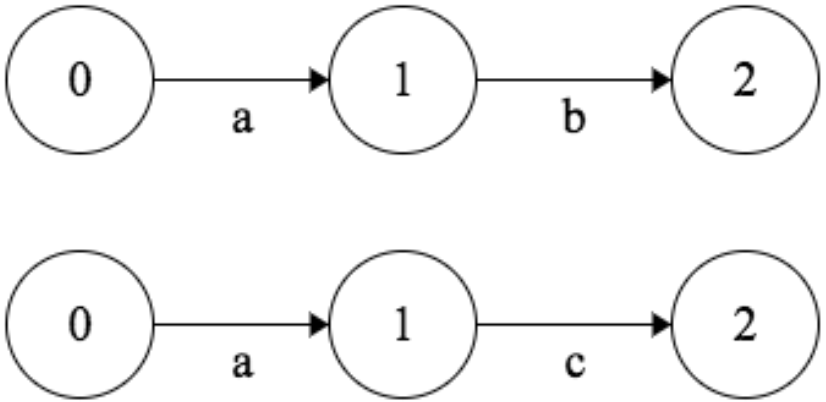
Add a new start and end state. Add ε-transition from new start to old start, old finals to new finals, and from new start to new final, and from new final to new start.



Draw an NFA for (ab|ac)*

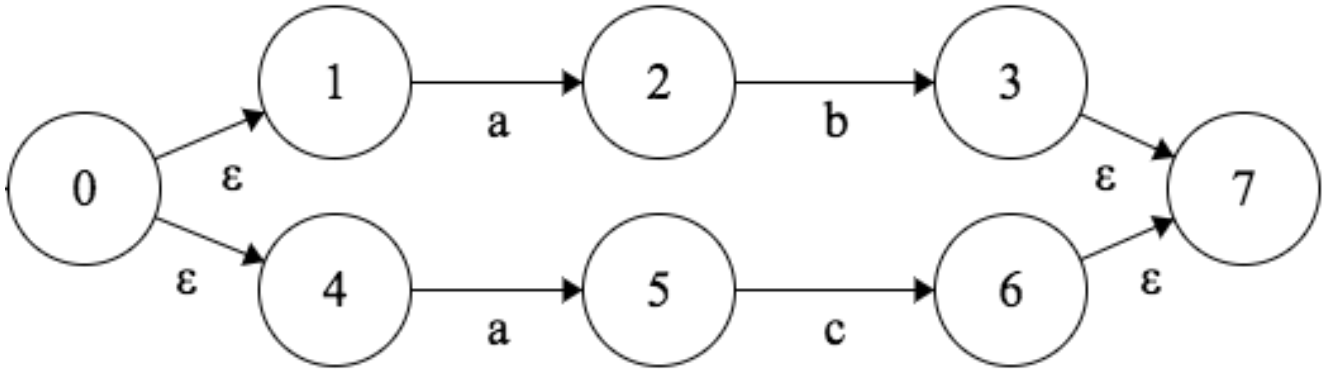
Draw an NFA for (ab|ac)*

Begin by creating an NFA for “ab” and “ac”



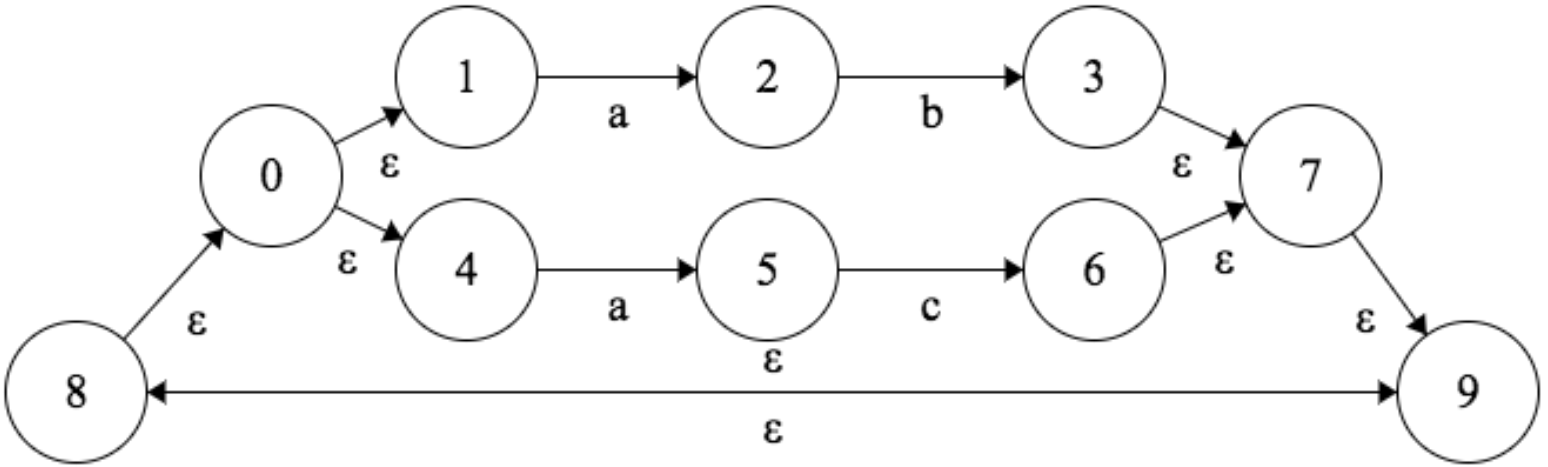
Draw an NFA for (ab|ac)*

Next, find the union of “ab” and “ac”



Draw an NFA for (ab|ac)*

Next, find the closure of (ab|ac).



Regex to NFA Complexity?

If a regular expression “A” has size n, where...

n = # of symbols + # of operations

Then how many states does the NFA of “A” have?

Each union and closure operation adds just two states

O(n), overall, which is pretty good!

Regex to NFA Practice

---

1. a
2. a*
3. ab
4. (ab)*
5. (a|b)
6. (a|b)*
7. (a|b)*a
8. Binary strings that start and end with 1

Bonus: Consider e_closure and move for each solution

Reducing NFA to DFA

Reducing DFA to RE

Minimizing DFA

Complement of DFA

Resources

Resources

Finite State Machine Designer: http://madebyevan.com/fsm/