FIT2014 Theory of Computation

Lecture 8

Kleene's Theorem. I.

Regexp $\rightarrow$ NFA $\rightarrow$ FA

slides by Graham Farr

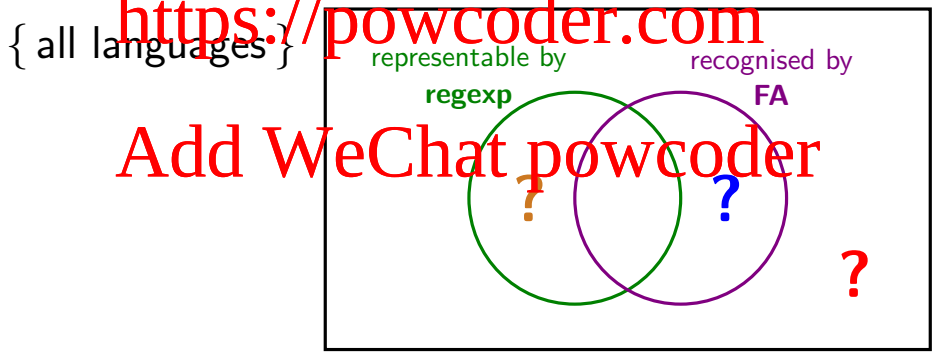
Overview

- Questions
- Kleene's Theorem
- Convert Regular Expressions to NFA
- Convert NFA to FA
- *Next lecture:*
  Convert FA to Regular Expression

Stephen Cole Kleene (1909–1994)
https://mathshistory.st-andrews.ac.uk/Biographies/Kleene/

Questions

- Can every language which is represented by a **regular expression** be described by a **finite automaton**?
- Can every language which is described by a **finite automaton** be represented by a **regular expression**?
- Can every language be represented by a **regular expression** or a **finite automaton**?



Kleene's Theorem

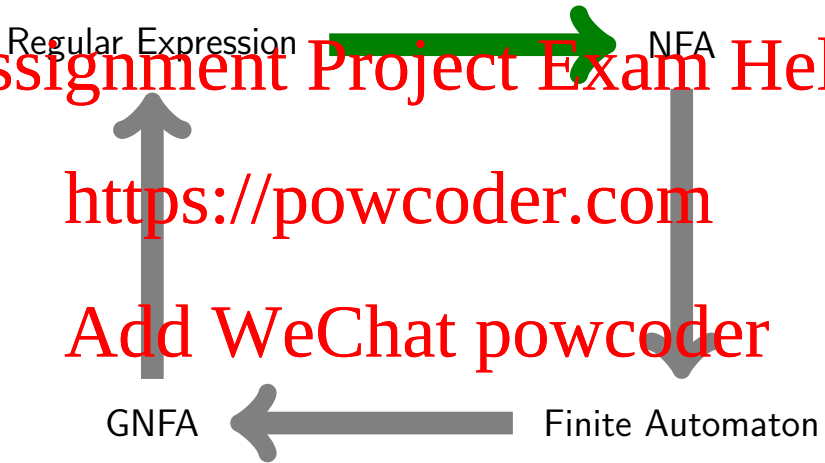
Theorem

Any language which can be defined by

- Regular Expressions
- Finite Automata
- Nondeterministic Finite Automata (NFA)
- Generalized Nondeterministic Finite Automata (GNFA)

can be defined by any of the other methods.

Kleene's Theorem

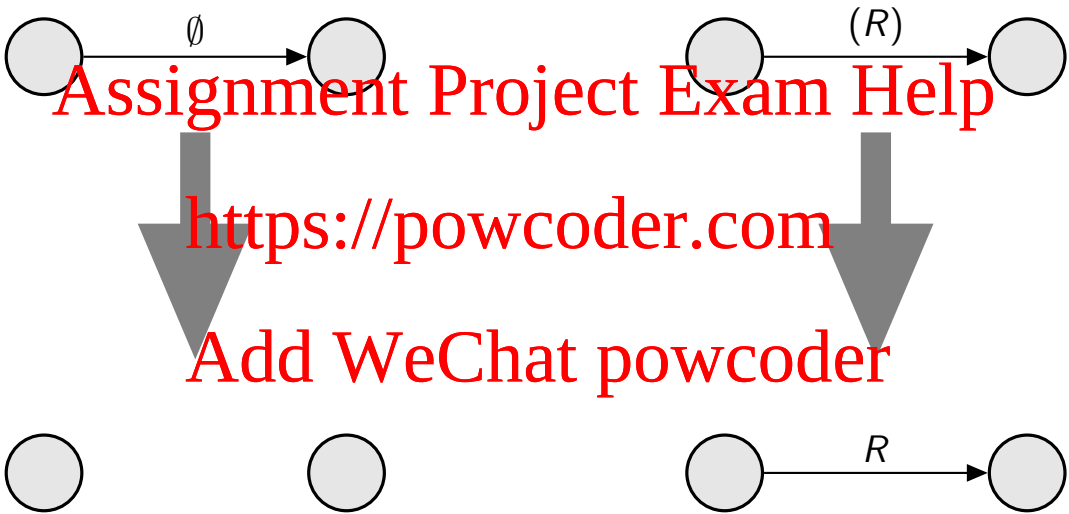


Converting Regular Expression to NFA

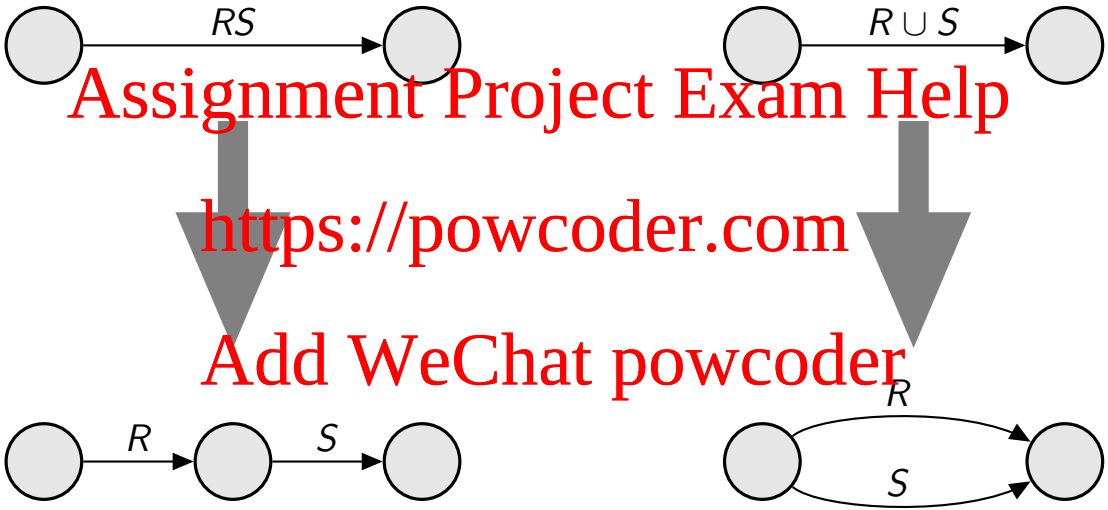
Start with:

Apply the following rules until all edges are labelled with a letter or ε:

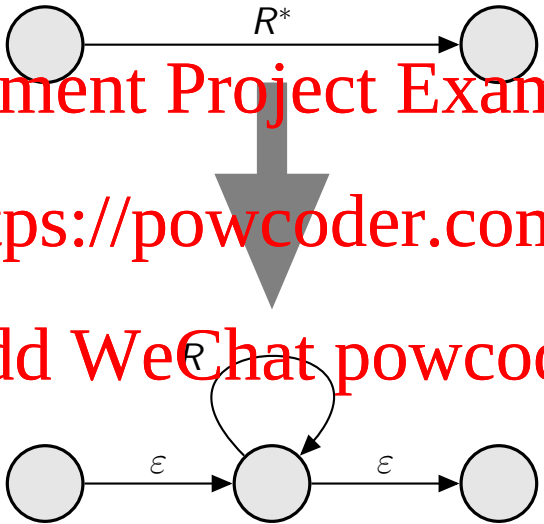
Converting Regular Expression to NFA



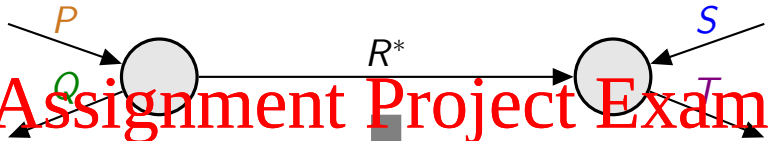
Converting Regular Expression to NFA



Converting Regular Expression to NFA

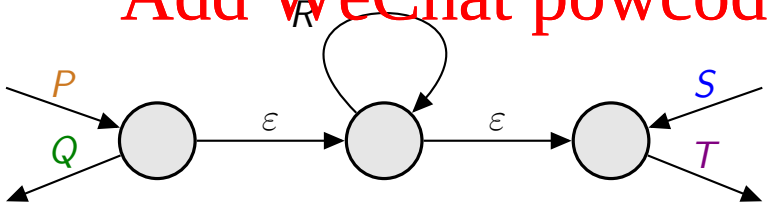


Converting Regular Expression to NFA



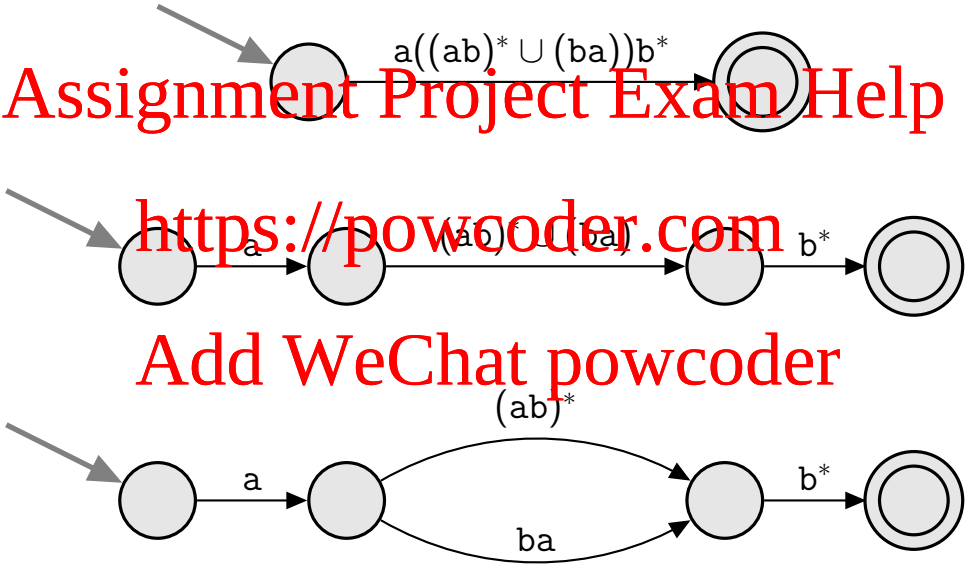
Here, you can't match PR^*Q or SR^*T, in general.

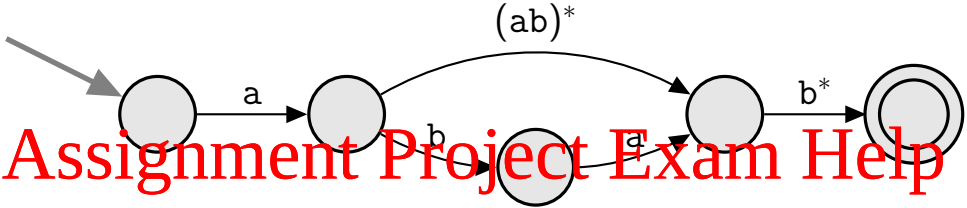
The two ε transitions are necessary, in general.

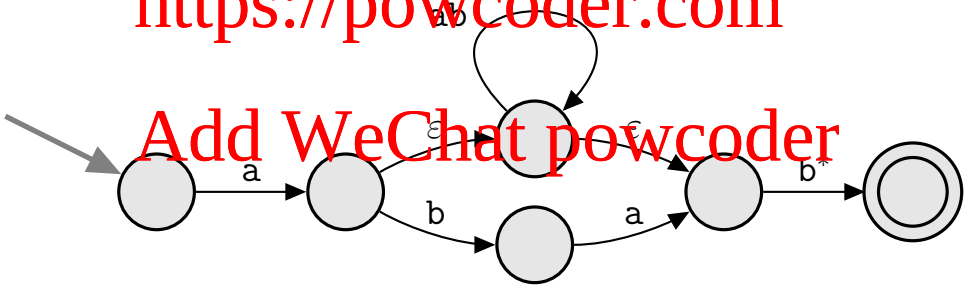


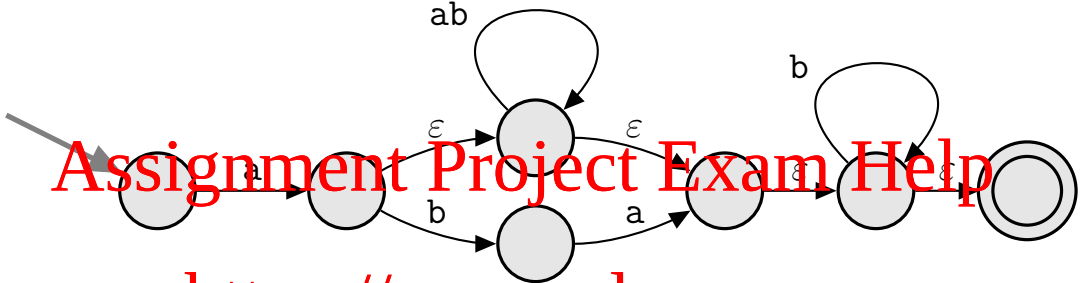
So the R loop cannot be at left node or right node.

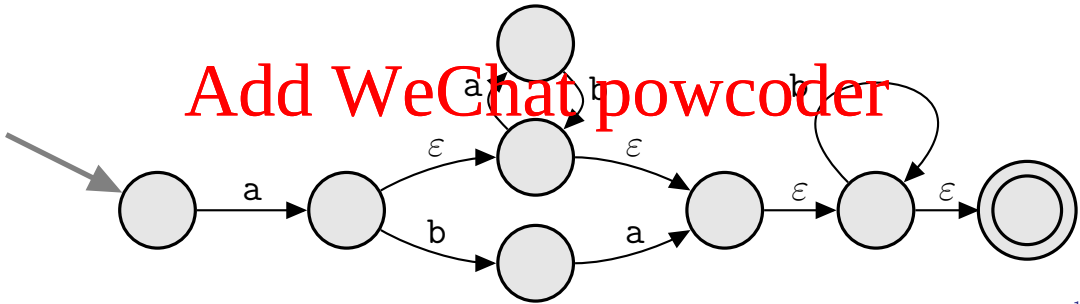
Converting Regular Expression to NFA. Example: $a((ab)^* \cup (ba))b^*$







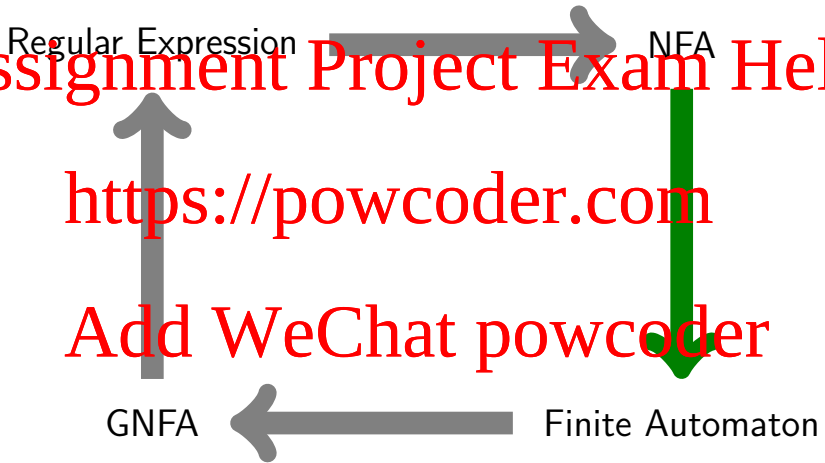




Complexity?

How reversible is this construction?

Kleene's Theorem



Converting a NFA to a FA

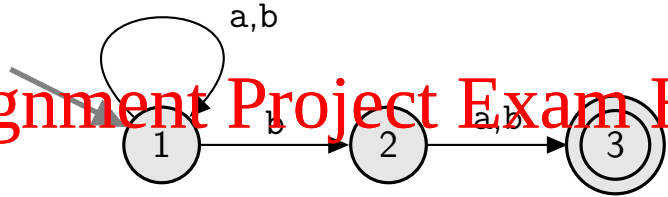
In a FA:

- Any string w traces a unique path, starting from the Start State and ending at some unique state, which we'll call endState(w).
- The string w is accepted if endState(w) is a Final State, otherwise it is rejected.
- endState(ε) $=$ Start State.

In a NFA:

- Any string w traces a set of paths, starting from the Start State and ending at some set of states, which we'll call endStates(w).
- The set might have zero, one or more members.
- The string w is accepted if endStates(w) contains a Final State, otherwise it is rejected.
- endStates(ε) $=$ $\{$ Start State $\}$ if there are no ε transitions.

Converting a NFA to a FA



$$\text{endStates}(\texttt{ab}) = \{1, 2\}$$
$$\text{endStates}(\texttt{aba}) = \{1, 3\}$$

In general, if w is a string and x is a single letter, then

$$\text{endStates}(wx) = \{q : \text{for some state } p \in \text{endStates}(w), \text{there is a transition } p \xrightarrow{x} q\}$$

. . . *provided* there are no *empty* transitions.

This suggests part of a method for constructing endStates(w) for all strings w.

Converting a NFA to a FA

Idea:

sets of states in the NFA $\longrightarrow$ states in the FA.

Informally (and *assuming no empty transitions* for the time being):

Start with the one-element set { Start State }.

- This is endStates(ε).
- It's the set of NFA states we can possibly be in at the very start.

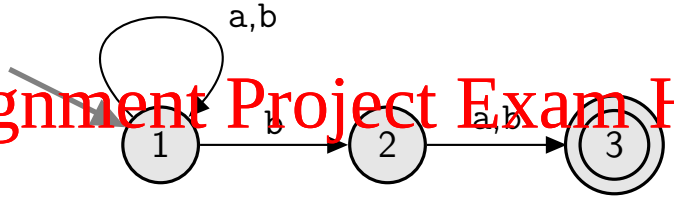
Construct endStates(a), the set of all states we could then get to by reading a single a.
Construct endStates(b), the set of all states we could then get to by reading a single b.

For each set of states, X, that we construct:

- find the set of states we can get to from X, by reading a single a.
- find the set of states we can get to from X, by reading a single b.

Keep doing this, until we no longer get any new sets of states.

Converting a NFA to a FA



state	a	b
Start {1}		

→

state	a	b
Start {1}	{1}	{1,2}
{1,2}		

→

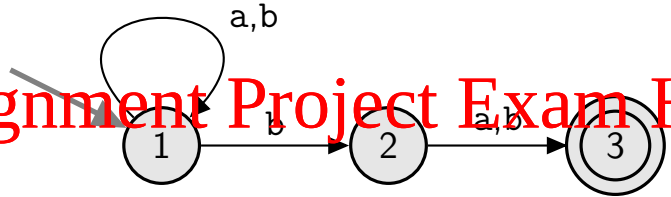
state	a	b
Start {1}	{1}	{1,2}
{1,2}	{1,3}	{1,2,3}
{1,3}		
{1,2,3}		

... →

	state	a	b
Start	{1}	{1}	{1,2}
	{1,2}	{1,3}	{1,2,3}
	{1,3}	{1}	{1,2}
	{1,2,3}		

.....

Converting a NFA to a FA

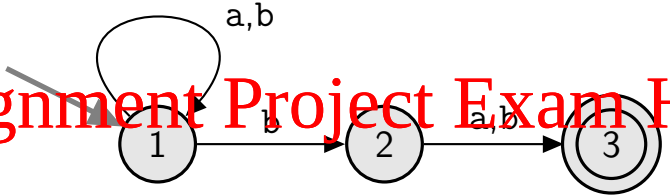


...

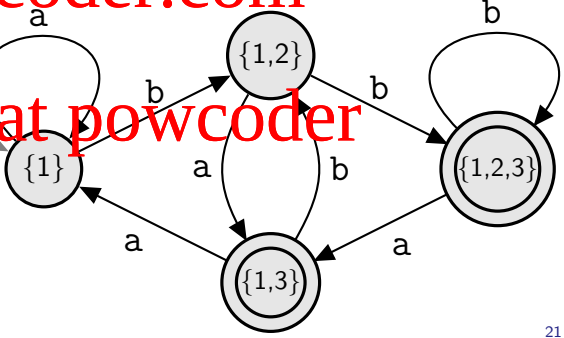
	state	a	b
Start	{1}	{1}	{1,2}
	{1,2}	{1,3}	{1,2,3}
	{1,3}	{1}	{1,2}
	{1,2,3}	{1,3}	{1,2,3}

	state	a	b
Start	{1}	{1}	{1,2}
	{1,2}	{1,3}	{1,2,3}
Final	{1,3}	{1}	{1,2}
Final	{1,2,3}	{1,3}	{1,2,3}

Converting a NFA to a FA



state		a	b
Start	{1}	{1}	{1,2}
	{1,2}	{1,3}	{1,2,3}
Final	{1,3}	{1}	{1,2}
Final	{1,2,3}	{1,3}	{1,2,3}



Algorithm: Conversion of NFA *without empty transitions* to FA

Input: a NFA

NextSetOfStatesOfNFA := { Start State of NFA }.

Create new incomplete row in FA table, for Start State called **NextSetOfStatesOfNFA**.

while *the FA table still has at least one incomplete row* **do**

- **CurrentStateInFA** := the state for the first incomplete row of the FA.
- **for** *each letter* x *in the alphabet* **do**
  - **NextSetOfStatesOfNFA** := $\{q$: for some NFA-state p in **CurrentStateInFA**, $\exists$ transition $p \xrightarrow{x} q\}$
  - Write **NextSetOfStatesOfNFA** in table entry for row **CurrentStateInFA**, column x.
  - **if** **NextSetOfStatesOfNFA** *is new* **then**
    - Create new incomplete row in table, using *set* **NextSetOfStatesOfNFA** as state.

Any FA state which (as a *set*) contains an NFA Final State is labelled Final.

Output: the FA

Converting a NFA to a FA

Now suppose that the NFA might have empty transitions, $q_1 \xrightarrow{\varepsilon} q_2$.

These allow change of state without reading any letter of the input string.

Every time we include a new state q in **NextSetOfStatesOfNFA**, we also need to include any state we can reach from it along empty transitions.

Look at all paths from q that just use ε transitions ...

$$q \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_2 \xrightarrow{\varepsilon} \cdots\cdots \xrightarrow{\varepsilon} q_i$$

... and include all states on such paths.

Modify earlier algorithm, for constructing the sets of NFA states, to take account of empty transitions.

Algorithm: Conversion of NFA to FA

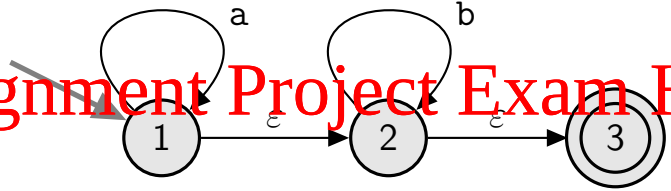
Input: a NFA

NextSetOfStatesOfNFA := { Start State of NFA }.
for *each* $q \in$ **NextSetOfStatesOfNFA** **do**
  Add, to **NextSetOfStatesOfNFA**, all states reachable from q along ε-transitions.
Create new incomplete row in FA table, for Start State called **NextSetOfStatesOfNFA**.
while *the FA table still has at least one incomplete row* **do**
  CurrentStateInFA := the state for the first incomplete row of the FA.
  for *each letter* x *in the alphabet* **do**
    NextSetOfStatesOfNFA :=
      $\{q$: for some NFA-state p in **CurrentStateInFA**, $\exists$ transition $p \xrightarrow{x} q\}$
    for *each* $q \in$ **NextSetOfStatesOfNFA** **do**
      Add, to **NextSetOfStatesOfNFA**, all states reachable from q along ε-transitions.
    Write **NextSetOfStatesOfNFA** in table entry for row **CurrentStateInFA**, column x.
    if **NextSetOfStatesOfNFA** *is new* **then**
      Create new incomplete row in table, using *set* **NextSetOfStatesOfNFA** as state.

Any FA state which (as a *set*) contains an NFA Final State is labelled Final.

Output: the FA

Converting a NFA to a FA



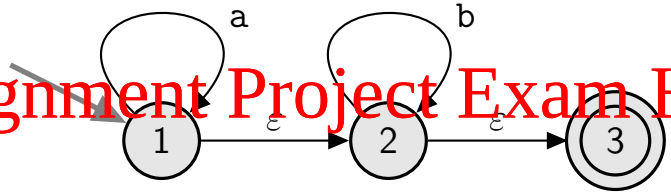
state	a	b
Start {1,2,3}		

state	a	b
Start {1,2,3}	{1,2,3}	{2,3}
{2,3}		

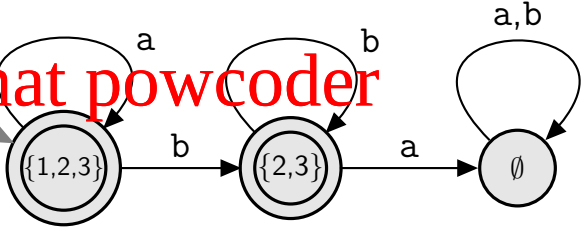
state	a	b
Start {1,2,3}	{1,2,3}	{2,3}
{2,3}	∅	{2,3}
∅		

state	a	b
Start {1,2,3}	{1,2,3}	{2,3}
{2,3}	∅	{2,3}
∅	∅	∅

Converting a NFA to a FA



state	a	b
Start {1,2,3}	{1,2,3}	{2,3}
{2,3}	∅	{2,3}
∅	∅	∅



Converting a NFA to a FA

Complexity?

- Think about how many states the FA may have as a function of the number of states in the NFA.

Revision

Today:

- Understand Kleene's Theorem
- Be able to convert Regular Expression $\longrightarrow$ NFA
- Be able to convert NFA $\longrightarrow$ Finite Automaton

Next lecture:

- Be able to convert FA $\longrightarrow$ Regular Expression

Reading:

Sipser, Ch 1, especially pp. 54–58, 66–69.