EQUIVALENCE WITH FA

COMP 4200 – Formal Language

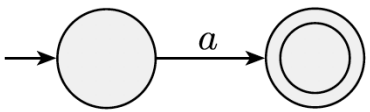
EQUIVALENCE WITH FINITE AUTOMATA

- A language is regular if and only if some regular expression describes it.
- This theorem has two directions.
  - If a language is described by a regular expression, then it is regular.
  - If a language is regular, then it is described by a regular expression.

IF A LANGUAGE IS DESCRIBED BY A REGULAR EXPRESSION, THEN IT IS REGULAR.

- We have a RE, R describing some language A. We show how to convert R into an NFA recognizing A.
- First, convert R into NFA N. We consider 6 cases from formal definition.

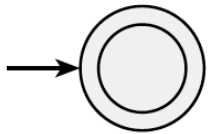
1. R = a for some a ∈ Σ. Then L(R) = {a}, the NFA recognizes L(R).



- Formally, N = {q1, q2}, Σ, δ, q1, {q2}, Accepts only a.

where we describe δ by saying that δ(q1, a) = {q2} and that δ(r, b) = ∅ for r ≠ q1 or b ≠ a.

2. $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$, and the following NFA recognizes L(R).

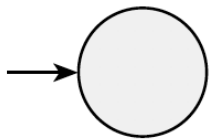


ε

- Formally, $N = \{q1\}, \Sigma, \delta, q1, \{q1\}$, where $\delta(r, b) = \emptyset$ for any r and b.

3. $R = \emptyset$. Then $L(R) = \emptyset$, and the following NFA recognizes L(R).

no final state for ∅



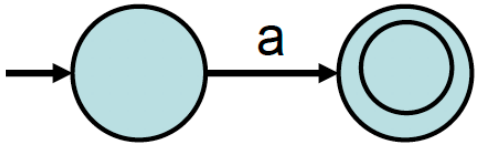
- Formally, $N = \{\{q\}, \Sigma, \delta, q, \emptyset,\}$ where $\delta(r, b) = \emptyset$ for any r and b.

4. $R = R_1 \cup R_2$.
5. $R = R_1 \circ R_2$.
6. $R = R^*$

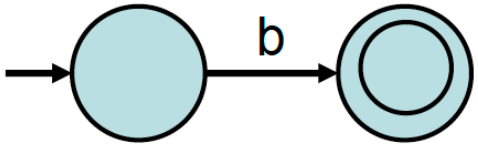
BUILDING AN NFA FROM THE REGULAR EXPRESSION

BASE CASES

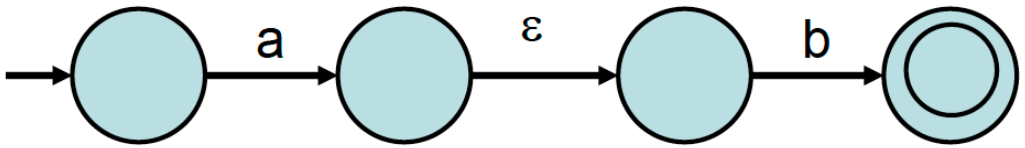
- a,

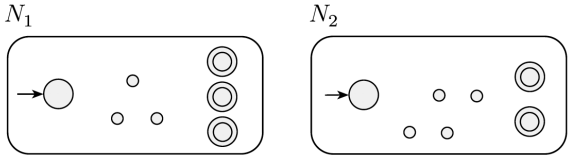


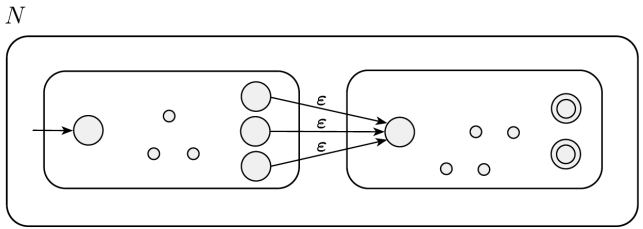
- b,

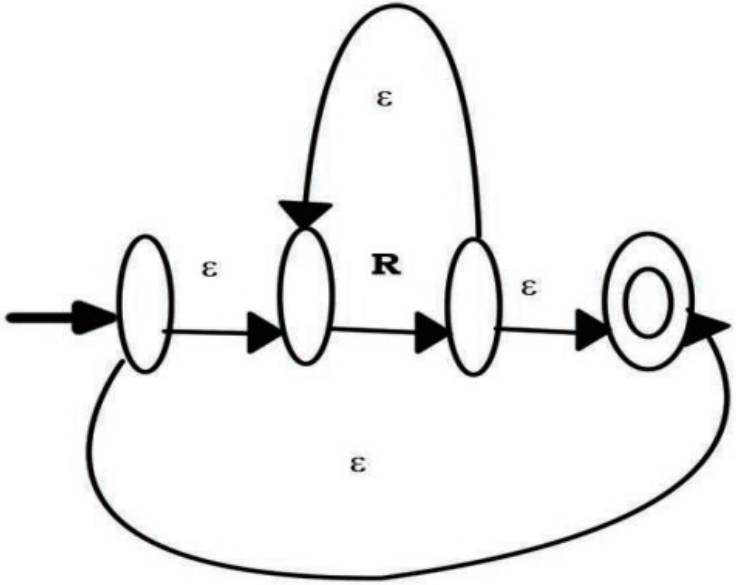


- ab,

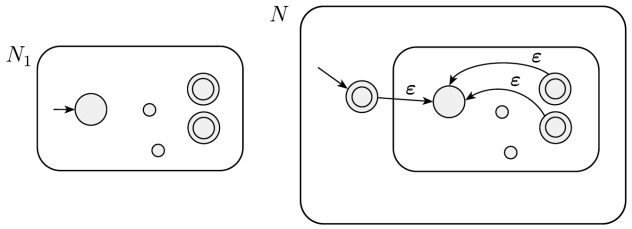


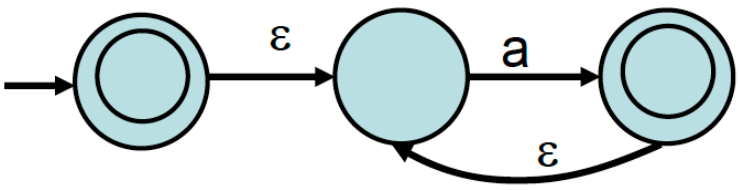




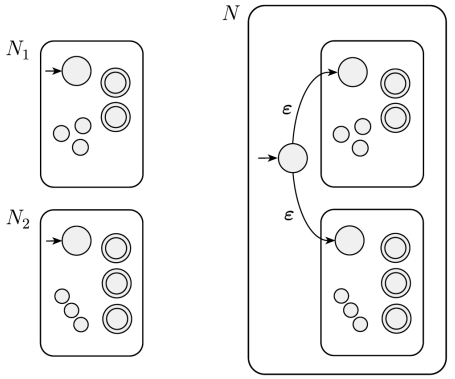


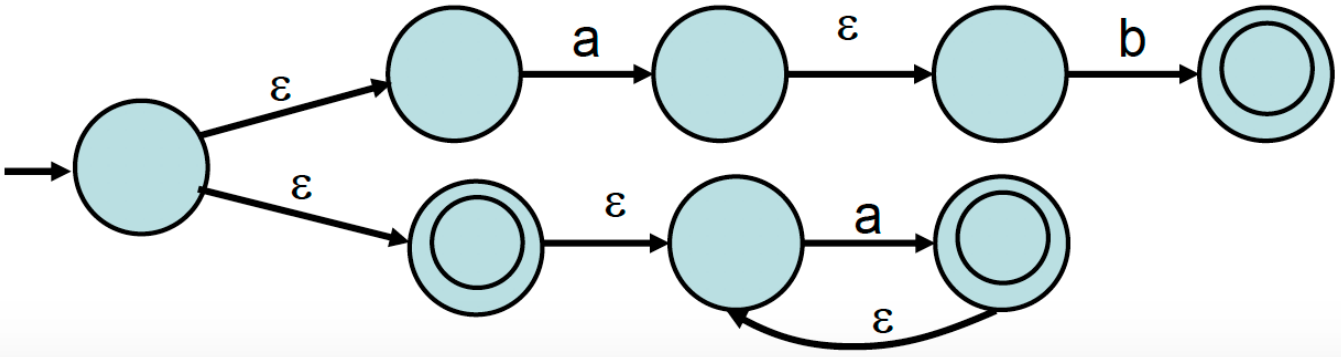
r=R*





L = AB ∪ A*

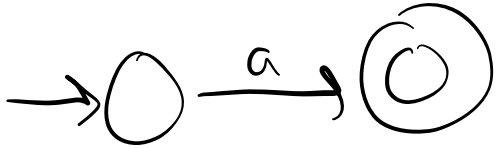


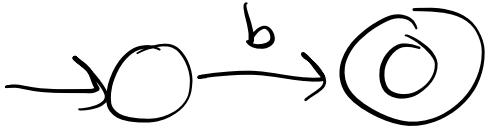


a

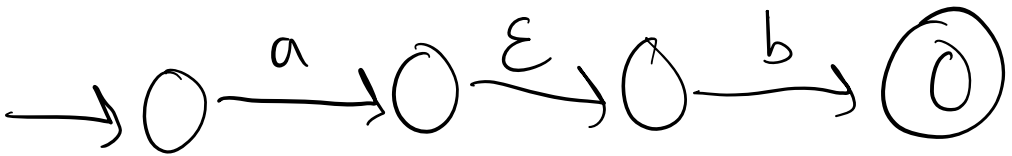
(AB ∪ A)* b

(ab ∪ a)*

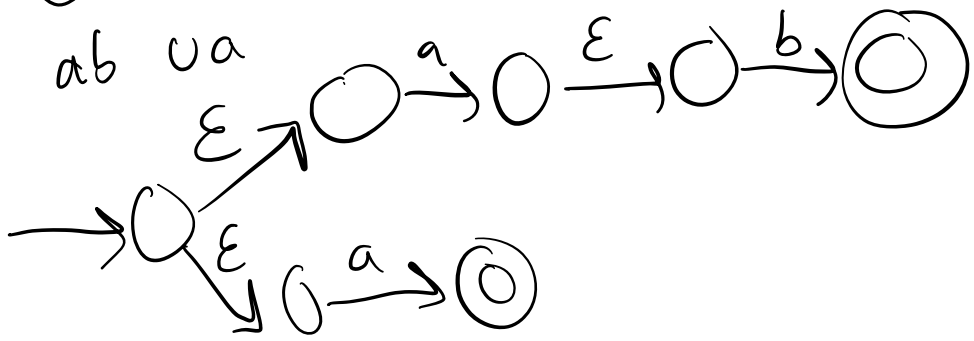




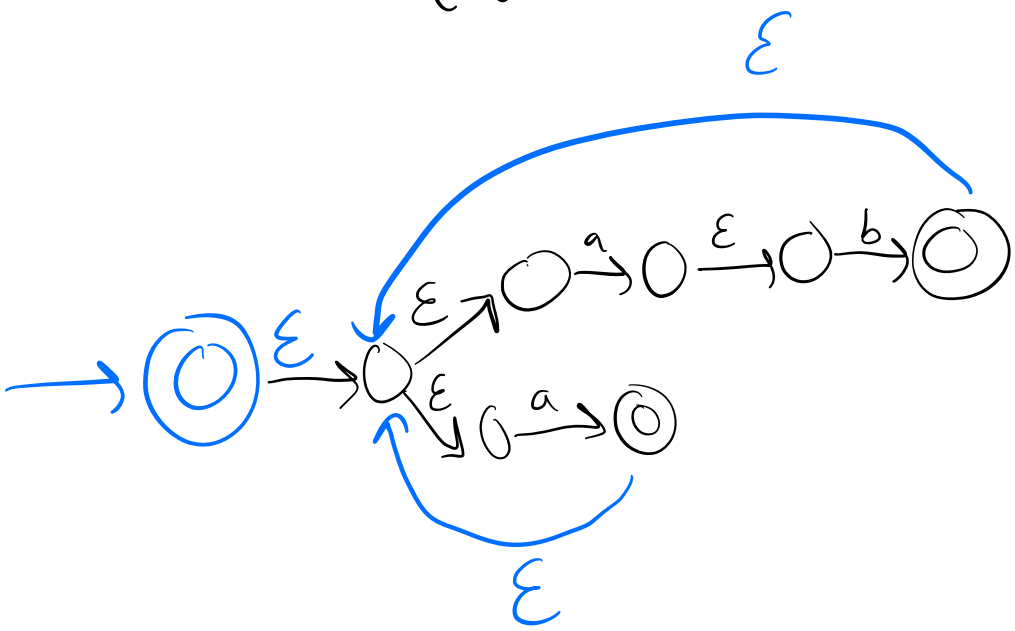
ab



ab ∪a

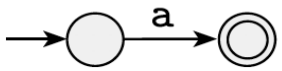


$(ab \cup a)^*$

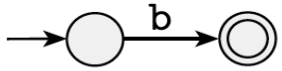


(AB ∪ A)*

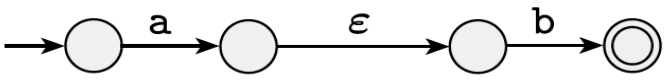
a



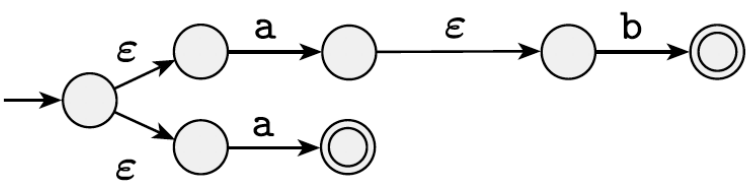
b



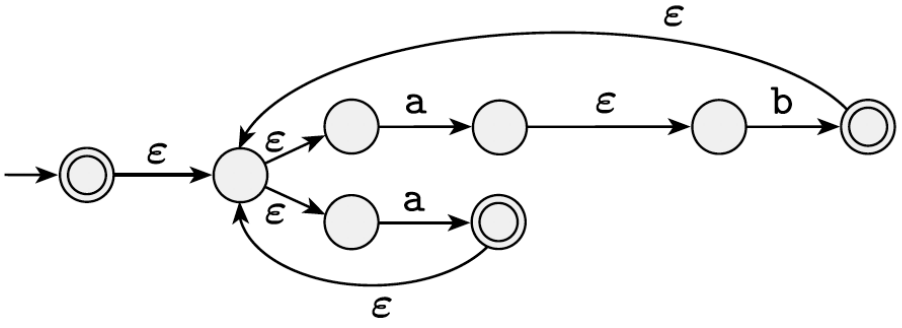
ab



ab ∪ a



(ab ∪ a)*

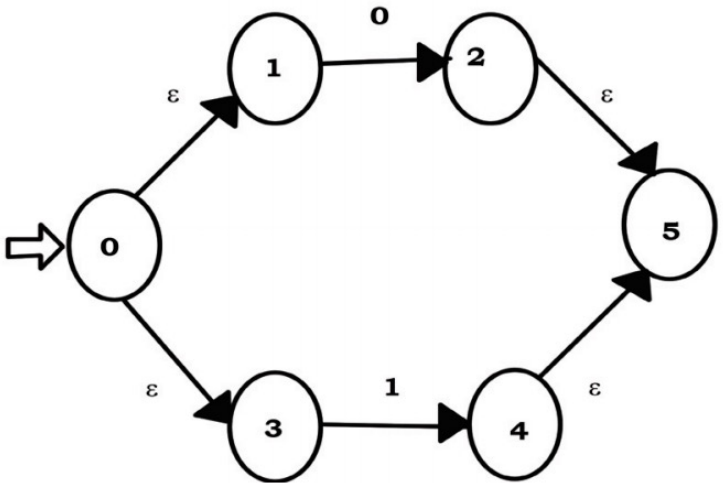


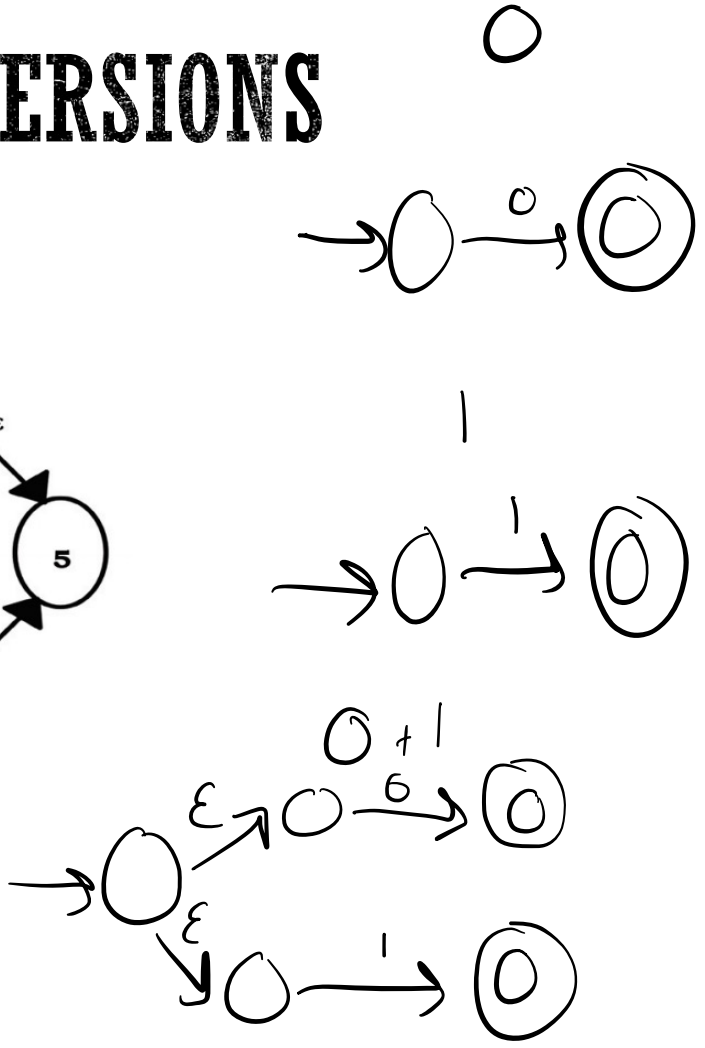
WHY?

- Many software tools work by matching regular expressions against text.
- One possible algorithm for doing so:
  - Convert the regular expression to an NFA.
  - (Optionally) Convert the NFA to a DFA using the subset construction.
  - Run the text through the finite automaton and look for matches.
- Runs extremely quickly!

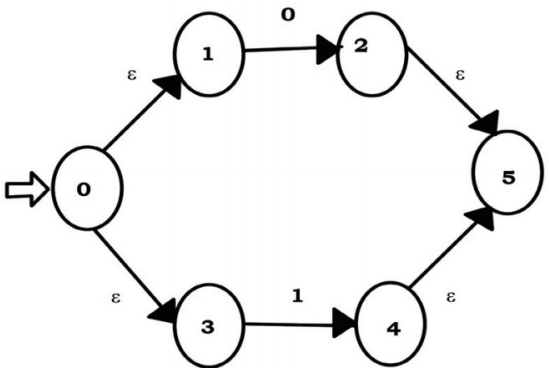
BASIC CONVERSIONS

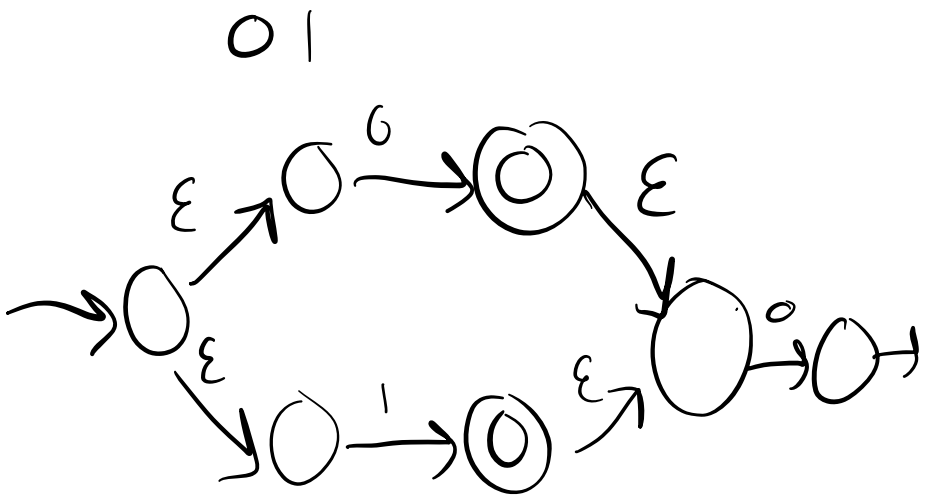
- 0 + 1



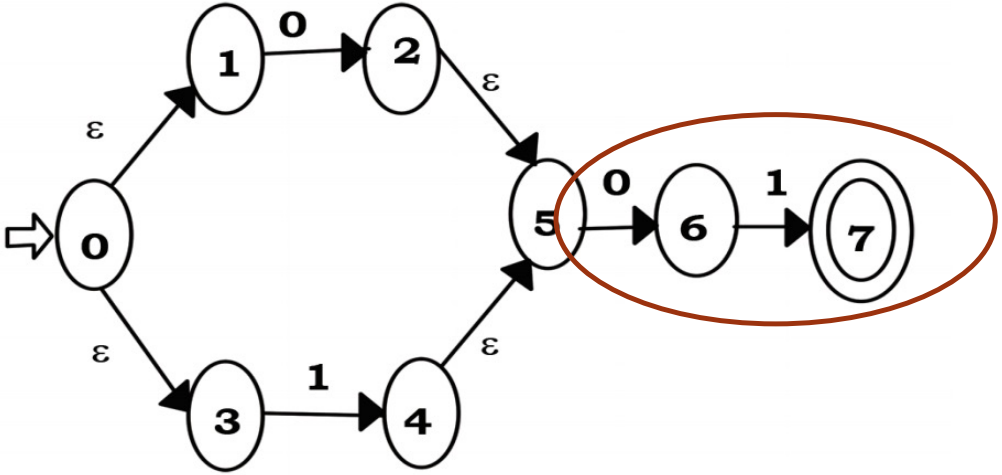


- (0+1)01
  - (0+1)

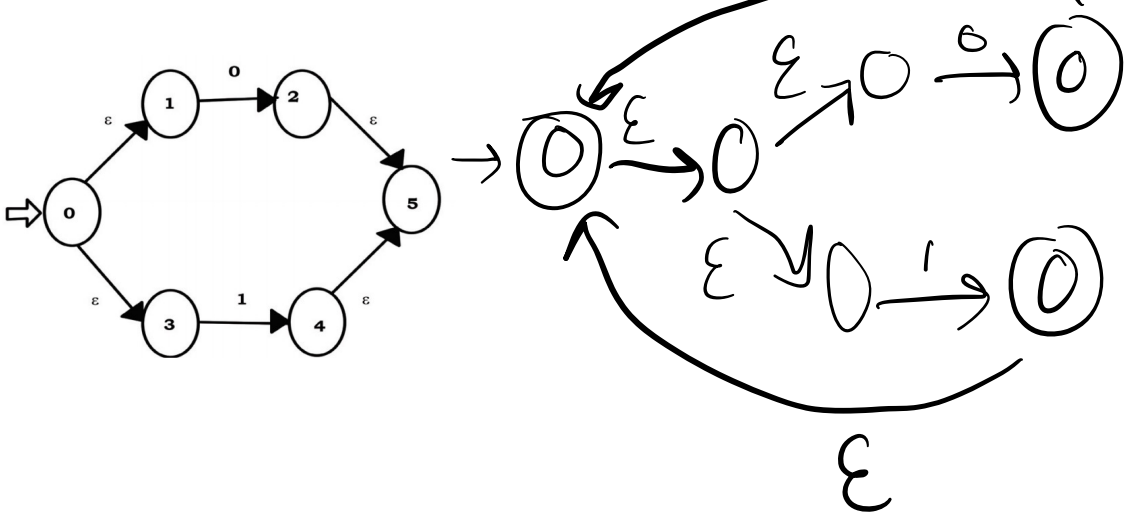




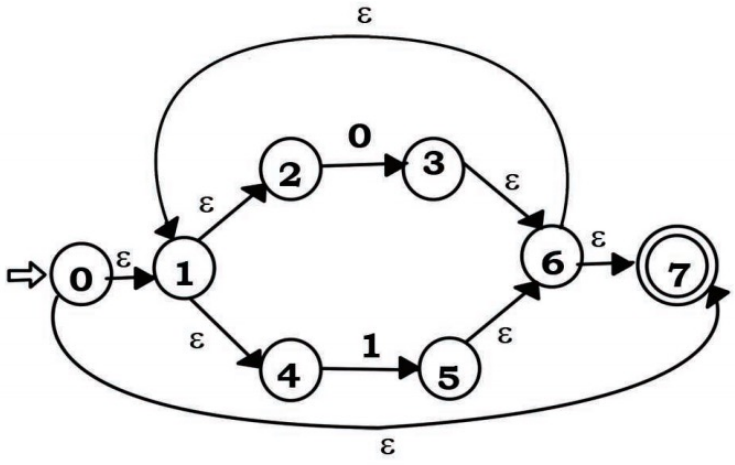
- (0+1)01



- (0+1)*
  - (0+1)



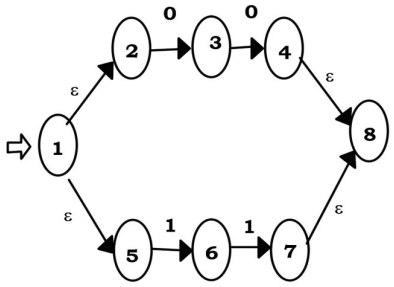
  - (0+1)*



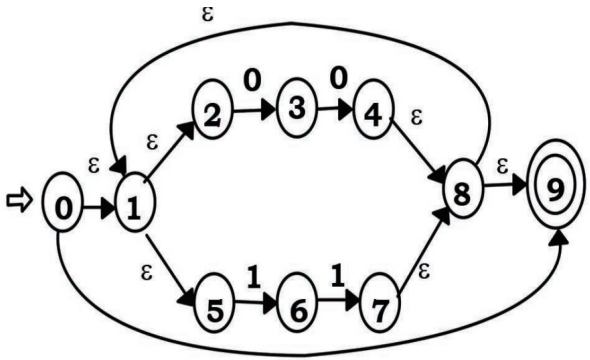
$(00+11)^*$

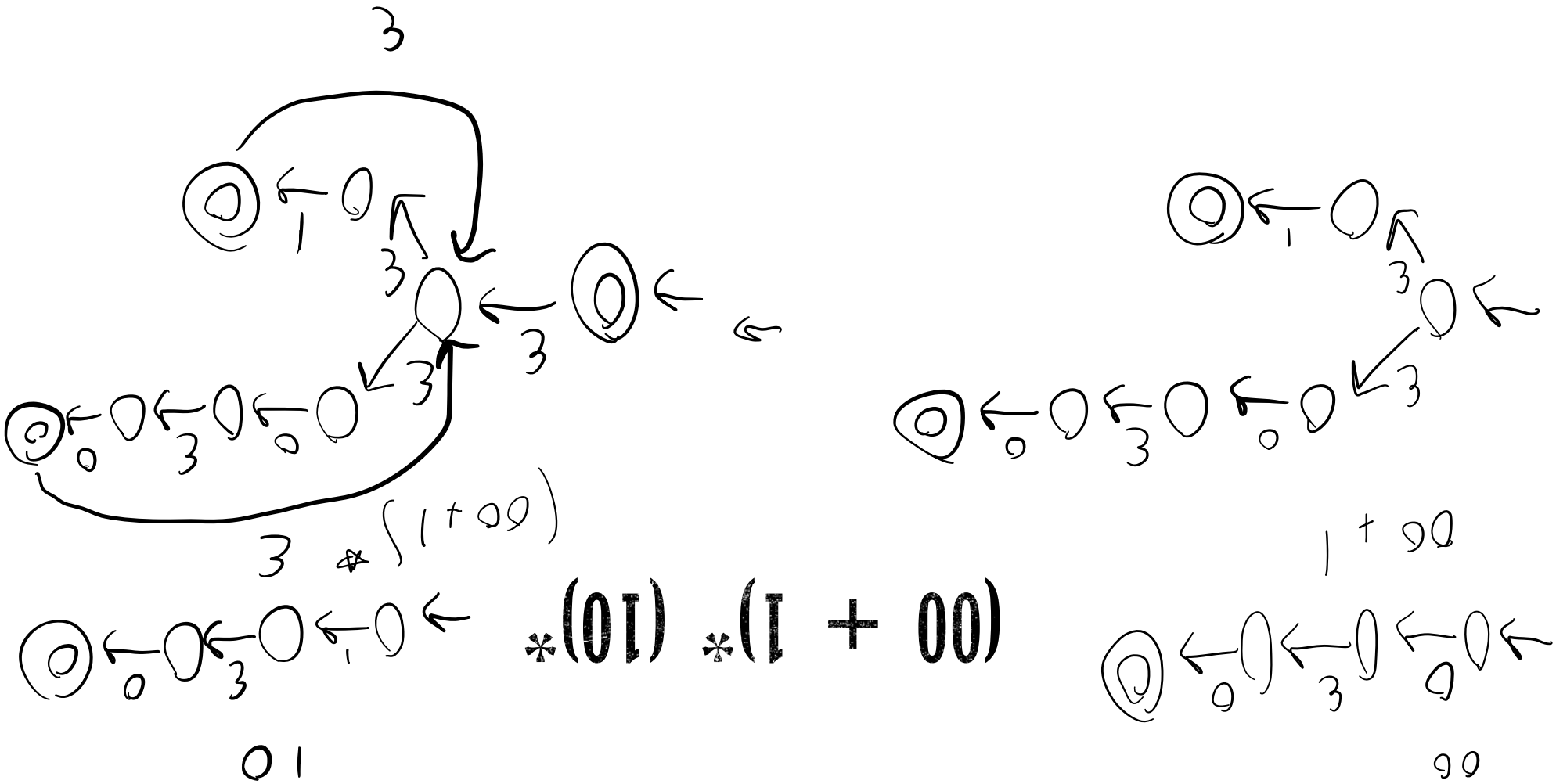
$(00+11)^*$

- Step 1: (00+11)



- Step 2: $(00+11)^*$



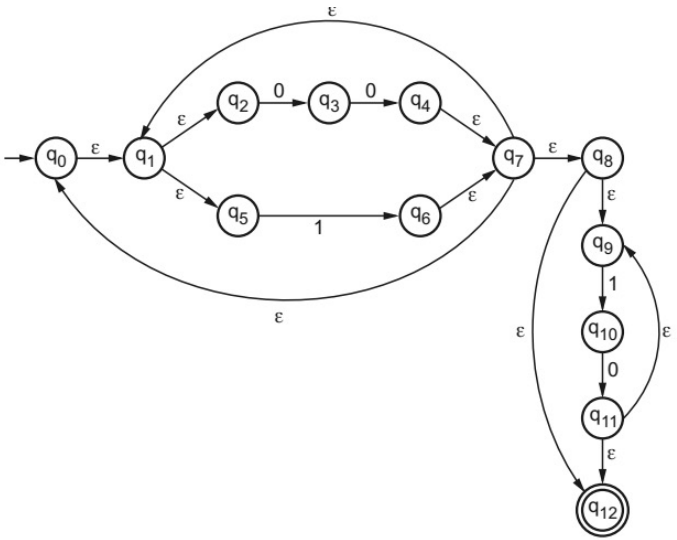

(00 + 1)* (10)* (01)*

$(01)^*$

ε

$(00+1)^*$ $(10)^*$

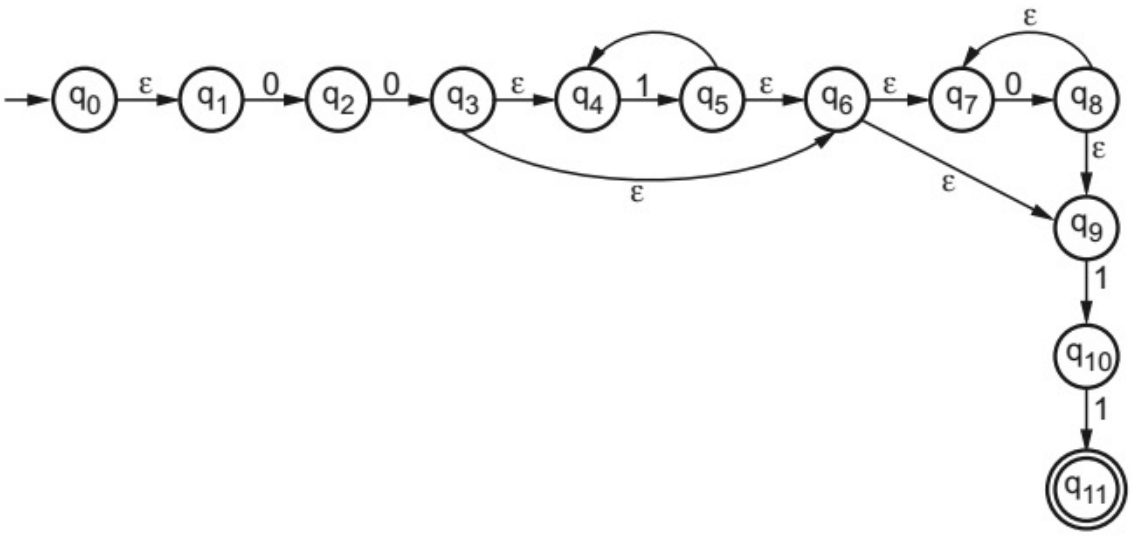
(00 + 1)* (10)*



00
ε
0*
ε
1*
ε
001*0*11
ε
100*
ε
001*0*100
ε
ε
ε
ε

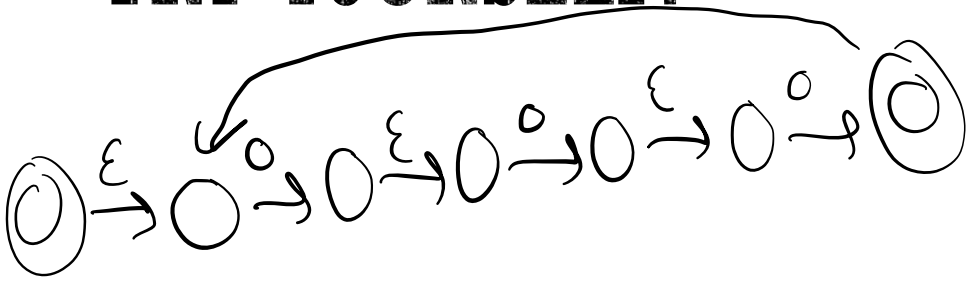
11* 0* 1 00
0
1
q
ε

001*0*11



TRY YOURSELF!

- (a ∪ b)*aba
- (000)* 1 U (00)*1
- 00(0 U 1)*
- (ab U c)*b
- (0+1)* (00+11) (0+1)*
- (ab+c*)*b
- a(abb)* U b
- a(bc)*



$00(0 \cup 1)^*$

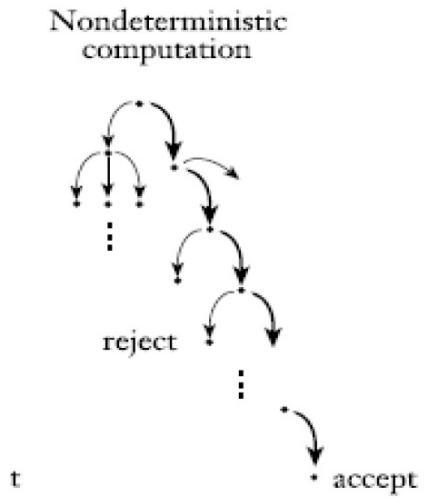
$a(abb)^* \cup b$

FA TO RE

IF A LANGUAGE IS REGULAR, THEN IT IS DESCRIBED BY A REGULAR EXPRESSION.

- We need to show that if a language A is regular, a regular expression describes it.
  - Convert DFA to RE
- Two parts, using a new type of finite automaton called a generalized nondeterministic finite automaton, GNFA.
  - Convert DFAs into GNFAs, and
  - GNFAs into regular expressions
- A GNFA is nondeterministic and so may have several

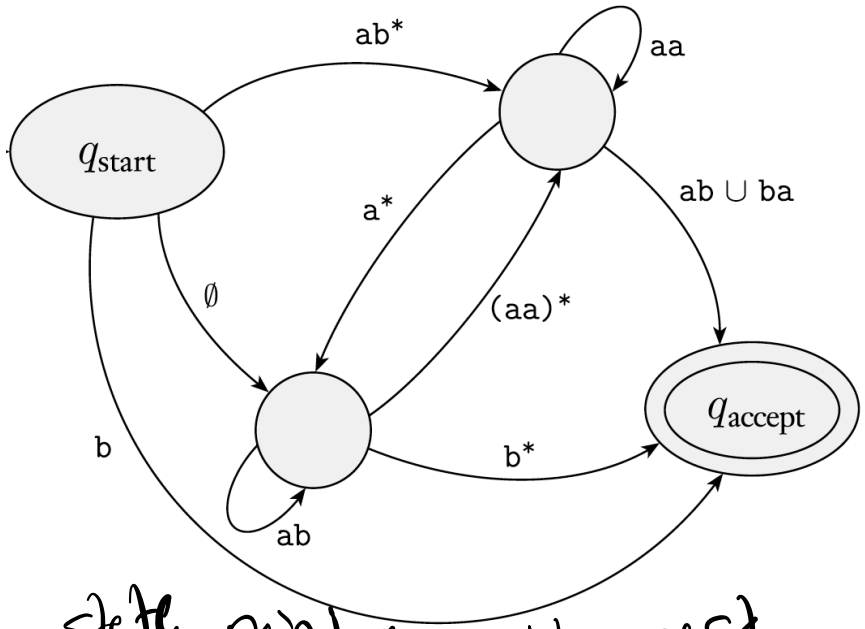
different ways to process the same input string



GENERALIZED NONDETERMINISTIC FINITE AUTOMATON

start & accept can't be the same

no incoming arrows for start state



every state points to the rest

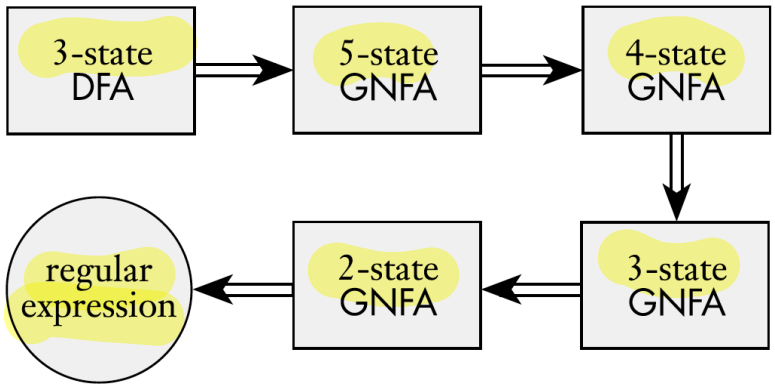
- GNFA, is a FA with transition having RE.
- We require that GNFAs always have a special form that meets the following conditions.
  - The start state has transition arrows going to **every other state** but no incoming arrows - arrows coming in from any other state.
  - There is only a *single accept state*, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
  - Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

1 accept state

CONVERT DFA TO GNFA

- Add new start and accept state.
- ε arrow to the old start state and a ε arrows from the old accept states.
- If any arrows have multiple labels, we replace each with a single arrow whose label is the union of the previous labels.
- If no arrows between states add ∅, which has no cost.
- Now GNFA is in special form

- Say that the GNFA has k states. Then, because a GNFA must have a start and an accept state and they must be different from each other, we know that $k \geq 2$.
  - if $k > 2$, construct an equivalent GNFA with $k - 1$ states. This step can be repeated on the new GNFA until it is reduced to two states.
  - If $k = 2$, the GNFA has a single arrow that goes from the start state to the accept state. The label of this arrow is the equivalent regular expression.

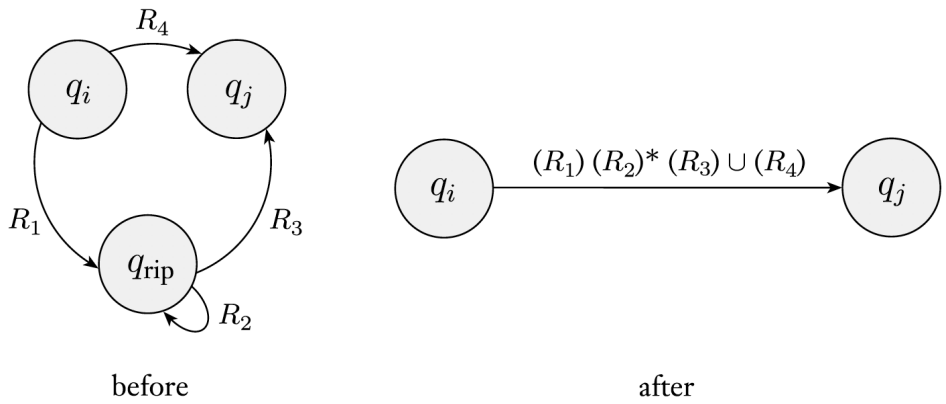


Typical stages in converting a DFA to a regular expression

CONSTRUCTING AN EQUIVALENT GNFA WITH ONE FEWER STATE

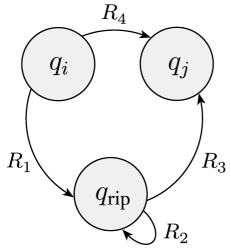
- Select a state
- Rip it out of the machine
- Repair the remainder so that the same language is still recognized, call the removed state q_{rip}.
- Take the machine from q_i to q_i either directly or via q_{rip}.

if parallel then union

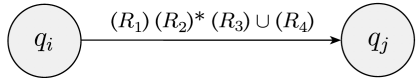


In the old machine, if

- q_i goes to q_{rip} with an arrow labeled R_1,
- q_{rip} goes to itself with an arrow labeled R_2,
- q_{rip} goes to q_j with an arrow labeled R_3, and
- q_i goes to q_j with an arrow labeled R_4,
- Then in the new machine, the arrow from q_i to q_j gets the label →(R_1) (R_2) * (R_3) U (R_4)



before



after

FORMAL DEFINITION

- A GNFA is similar to a NFA except for the transition function, which has the form

$$\delta : \left(Q - \{q_{\text{accept}}\}\right) \times \left(Q - \{q_{\text{start}}\}\right) \longrightarrow \mathcal{R}.$$

A generalized nondeterministic finite automaton is a 5-tuple, $(Q, \Sigma, \delta, q_{start}, q_{accept})$, where

1. Q is the finite set of states,
2. Σ is the input alphabet,
3. $\delta: \left(Q - \{q_{accept}\}\right) \times \left(Q - \{q_{start}\}\right) \rightarrow \mathcal{R}$ is the transition function,
4. q_{start} is the start state, and
5. q_{accept} is the accept state.

- Consider M be the DFA for language A. Then we convert M to a GNFA G by adding a new start state and a new accept state and additional transition arrows as necessary.
- CONVERT(G), which takes a GNFA and returns an equivalent regular expression.
- This procedure uses recursion, which means that it calls itself.

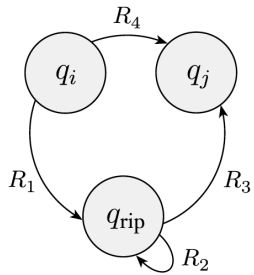
- CONVERT(G):
  - Let k be the number of states of G.
  - If k = 2, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
  - If k > 2, we select any state $q_{rip} \in Q$ different from q_{start} and q_{accept} and let G′ be the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$, where

$$Q' = Q - \{q_{rip}\},$$

and for any $q_i \in Q' - \{q_{accept}\}$ and any $q_j \in Q' - \{q_{start}\}$, let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

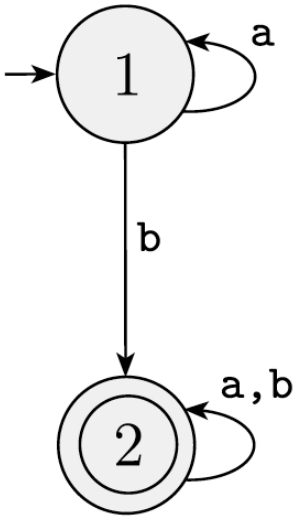
for $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, $R_3 = \delta(q_{rip}, q_j)$, and $R4 = \delta(q_i, q_j)$.



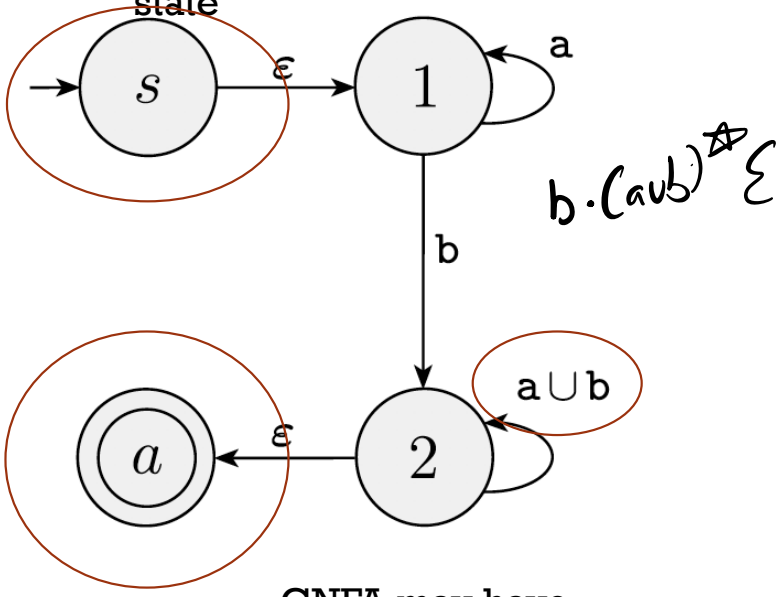
  - Compute CONVERT(G′) and return this value.

EXAMPLE DFA TO GNFA

Given DFA



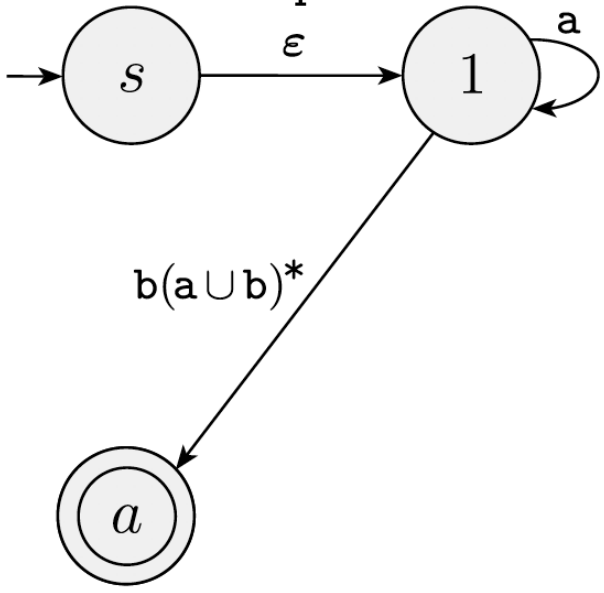
Adding new start state and accept state



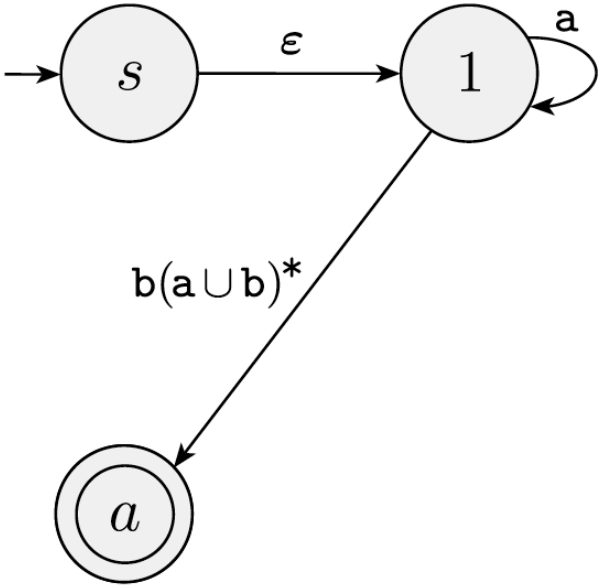
$b \cdot (a \cup b)^* \cdot \varepsilon$

GNFA may have only single transition going from 2 to itself

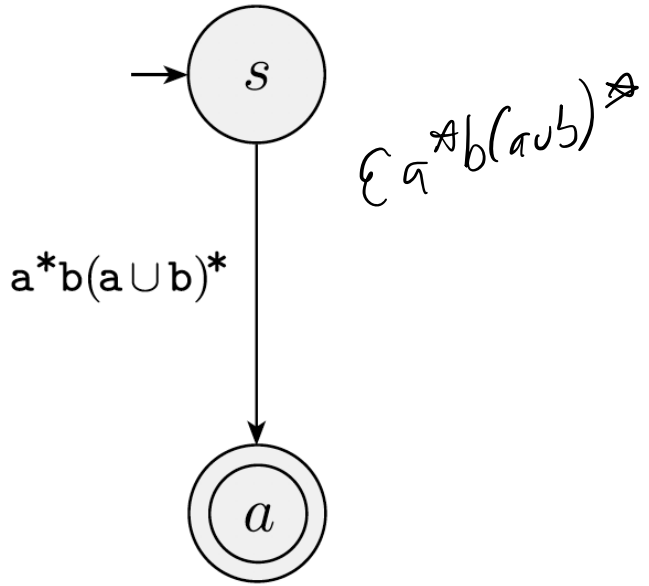
remove state 2 And update



State q_i is state 1, state q_j is a, and q_{rip} is 2, so R_1 = b, R_2 = a ∪ b, R_3 = ε, and R_4 = ∅. New label is (b)(a ∪ b)*(ε) ∪ ∅ → b(a ∪ b)*.



State q_i is state 1, state q_j is a, and q_{rip} is 2,
so R_1 = b, R_2 = a ∪ b, R_3 = ε, and R_4 = ∅.
New label is (b)(a ∪ b)*(ε) ∪ ∅ → b(a ∪ b)*.

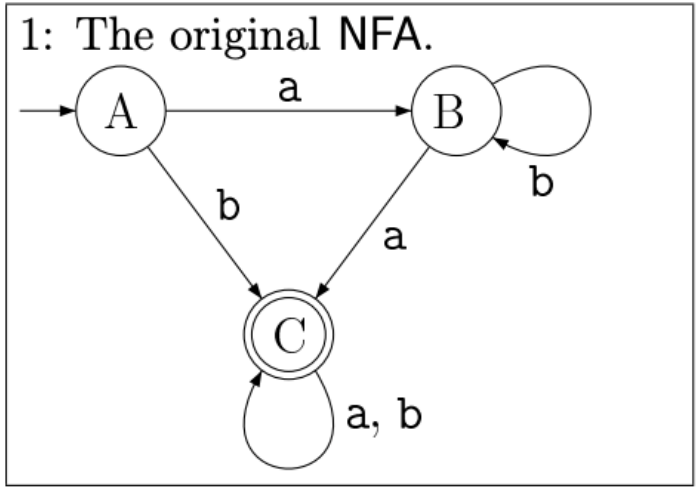


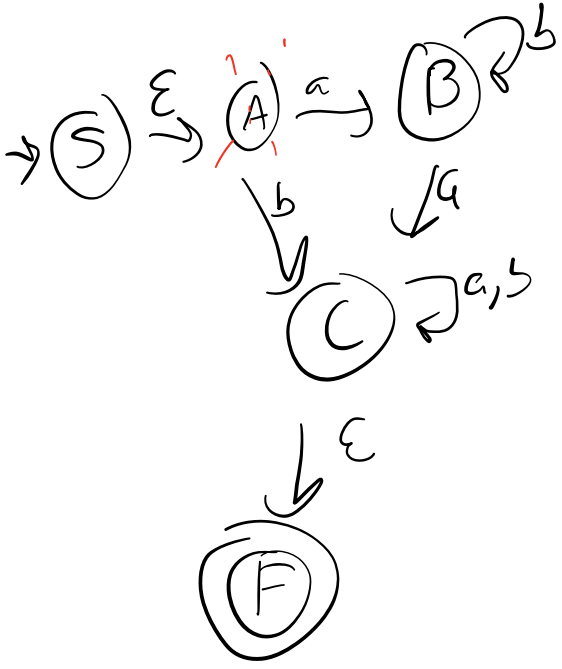
Only start and accept state remain

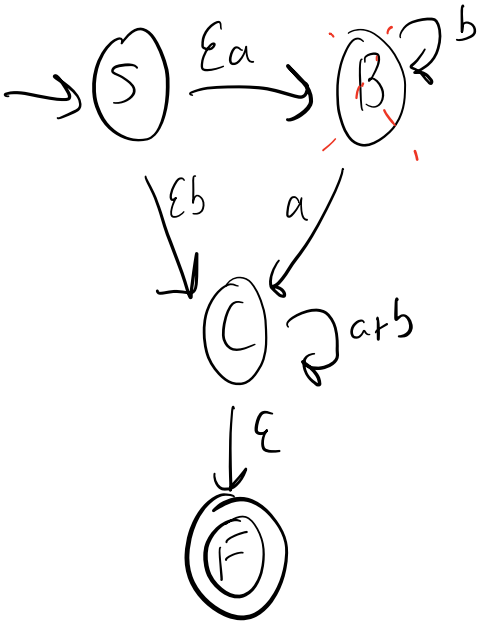
HINTS

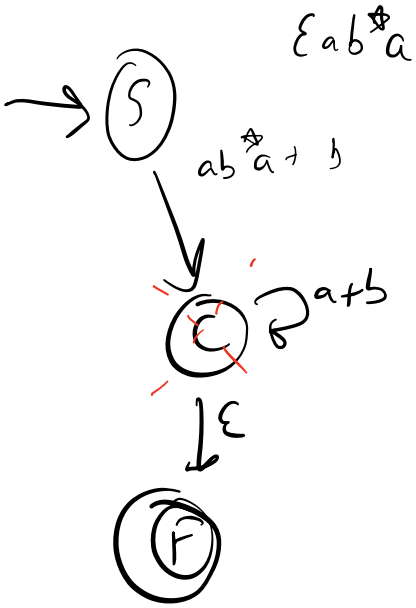
- The initial state of the DFA must not have any incoming edge.
- There must exist only one final state in the DFA.
- The final state of the DFA must not have any outgoing edge.
- Eliminate all the intermediate states one by one.
- These states may be eliminated in any order.
- Only an initial state going to the final state will be left.
- The transition is the required regular expression.
- Same direction and loops between two states → concatenation
- Opposite transitions and parallel transition → union
- Self loop → closure

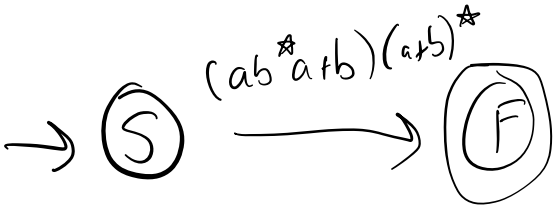
EXAMPLE – 2



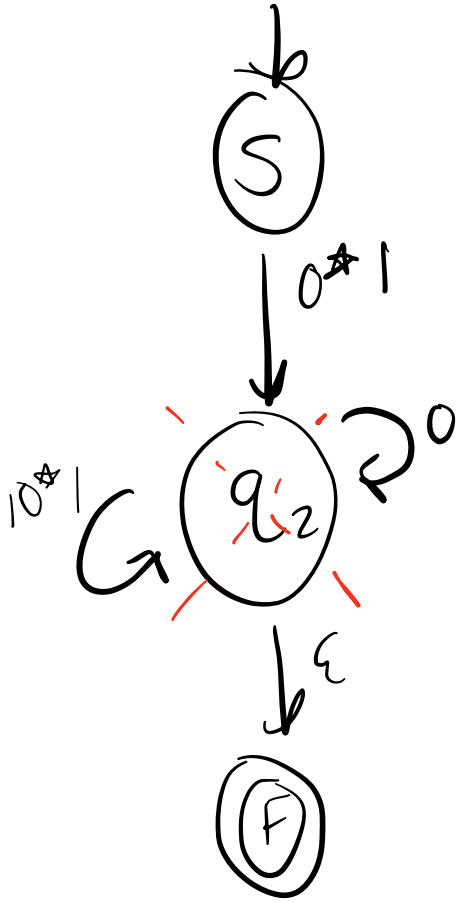


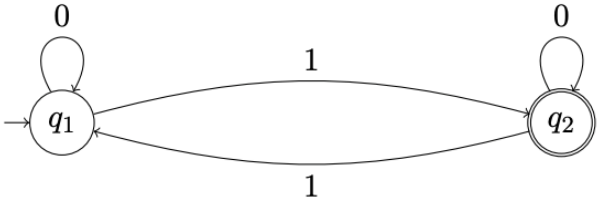

S
εa
B
b
εb
a
C
a+b
ε
F

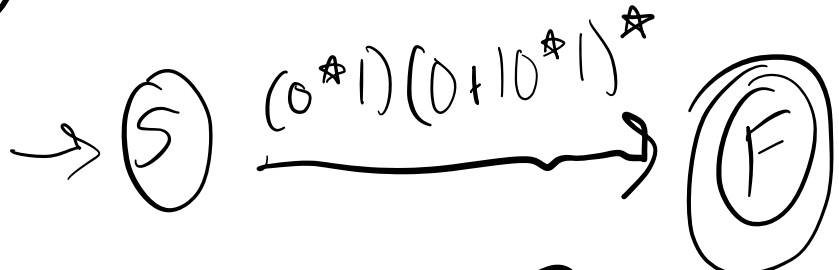

εab*a
S
ab*a + b
C
a+b
ε
F

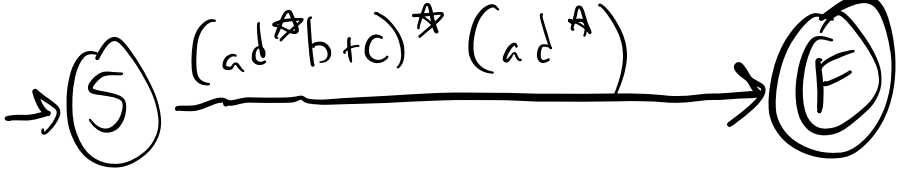

S
(ab*a+b)(a+b)*
F

EXAMPLE – 3

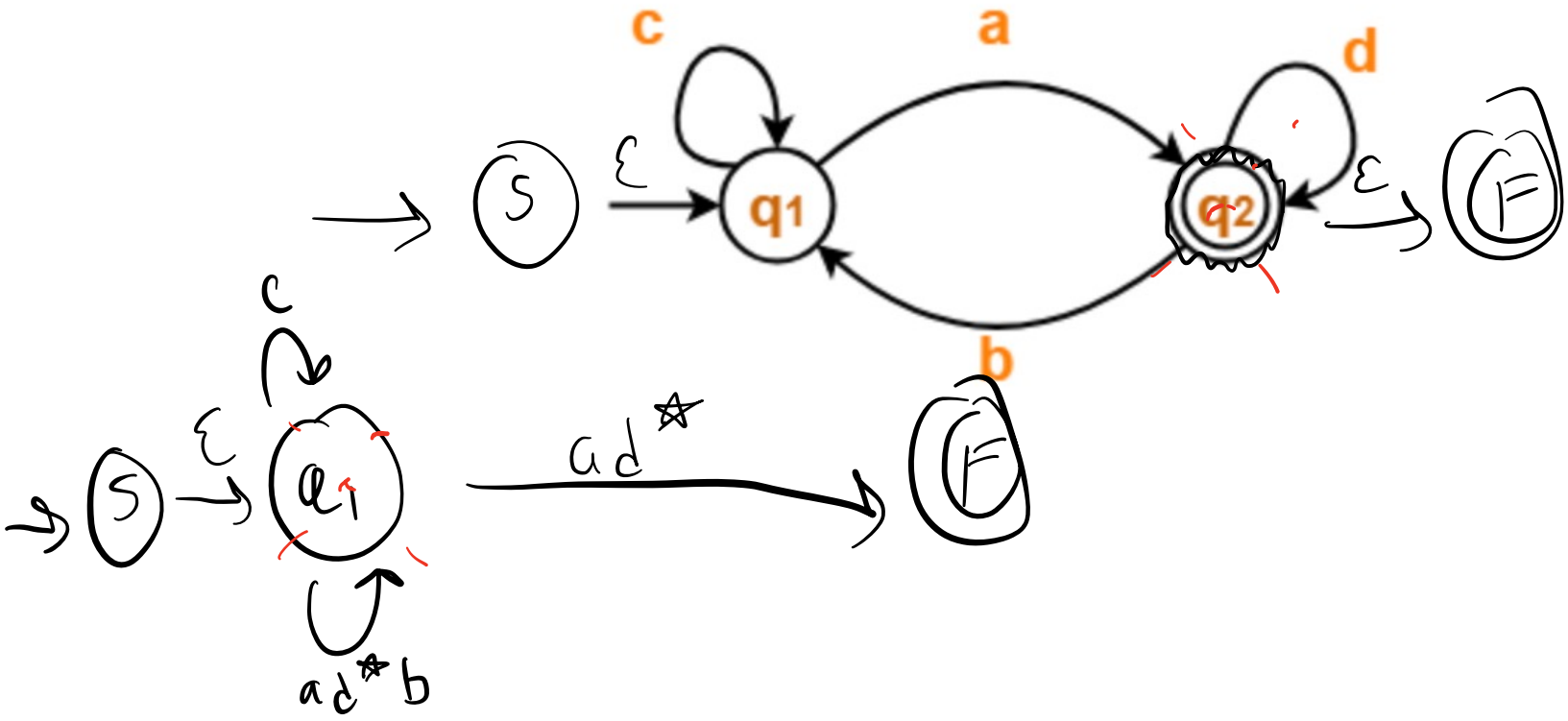




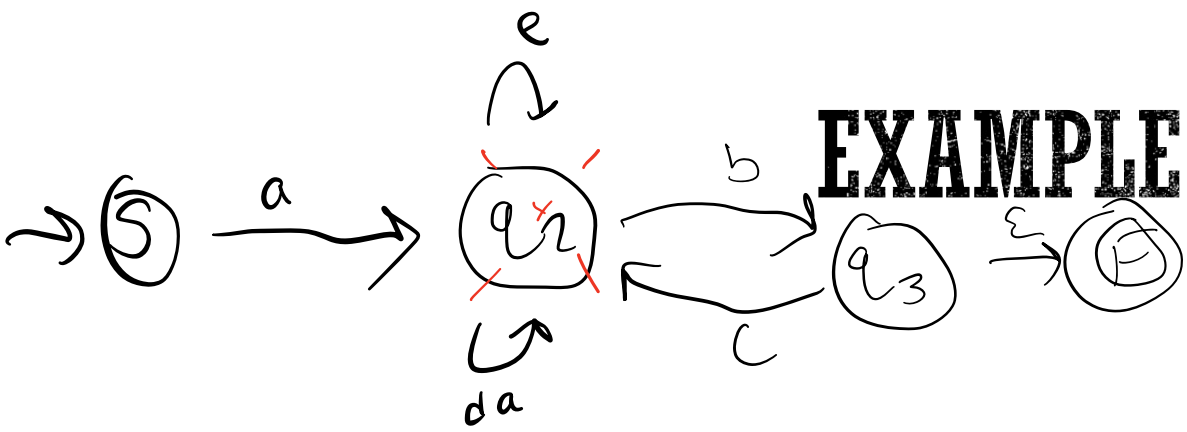


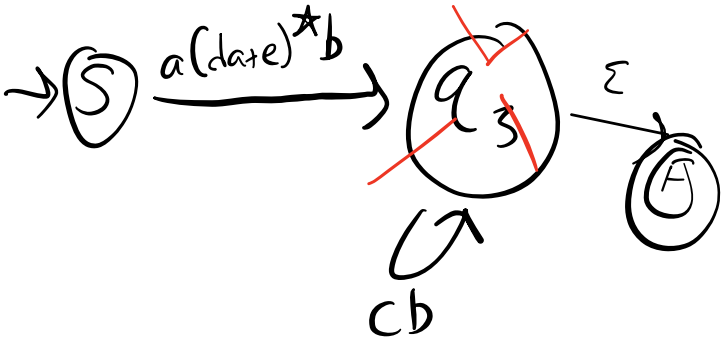


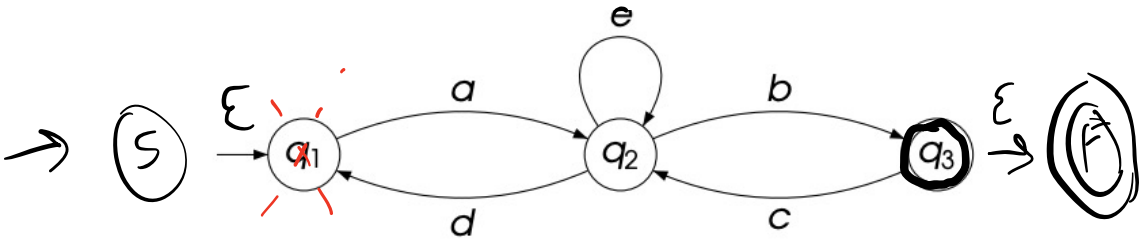
EXAMPLE – 4

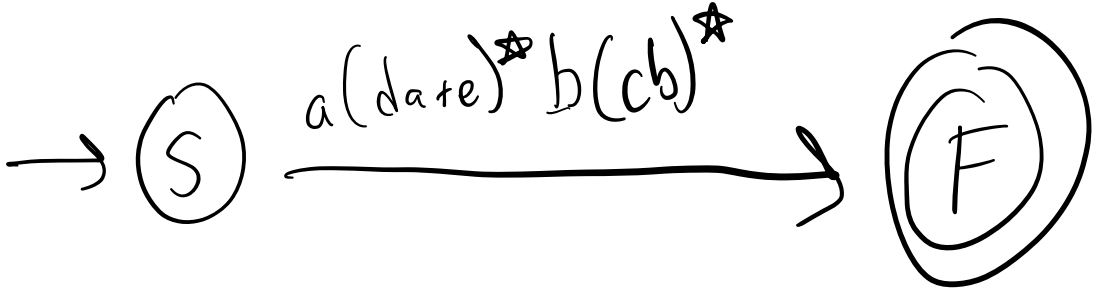


EXAMPLE

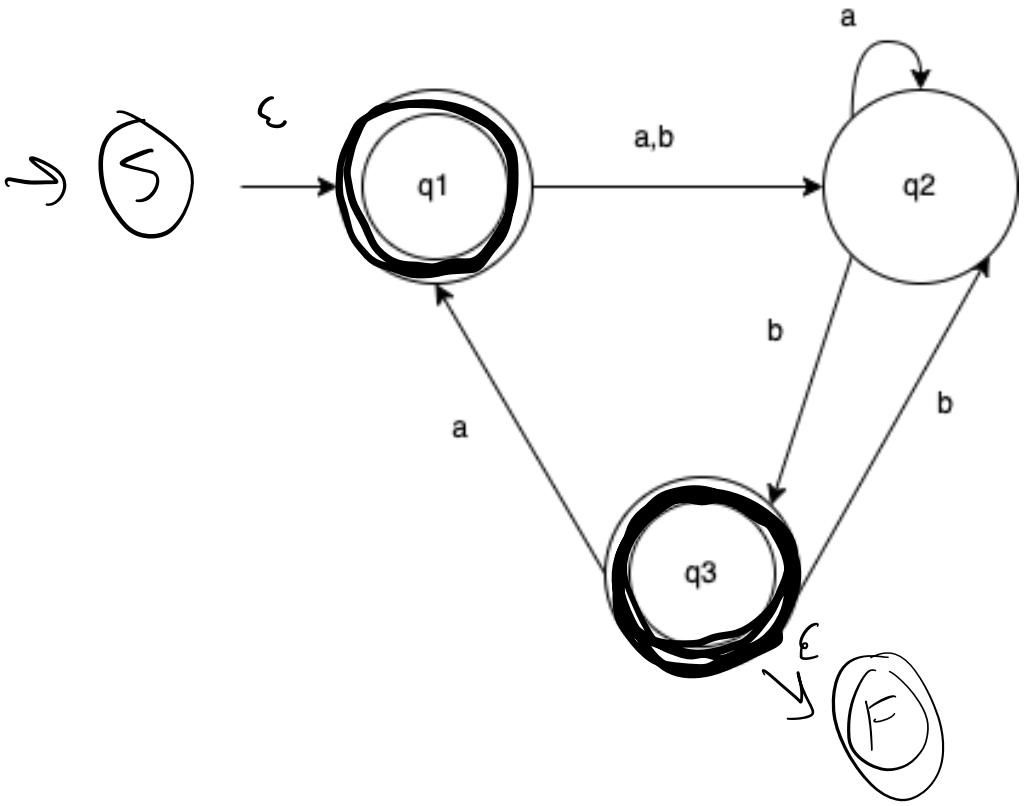




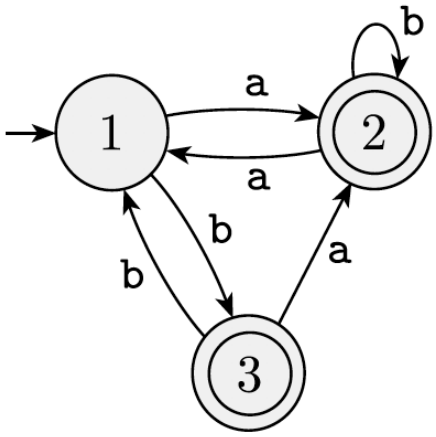




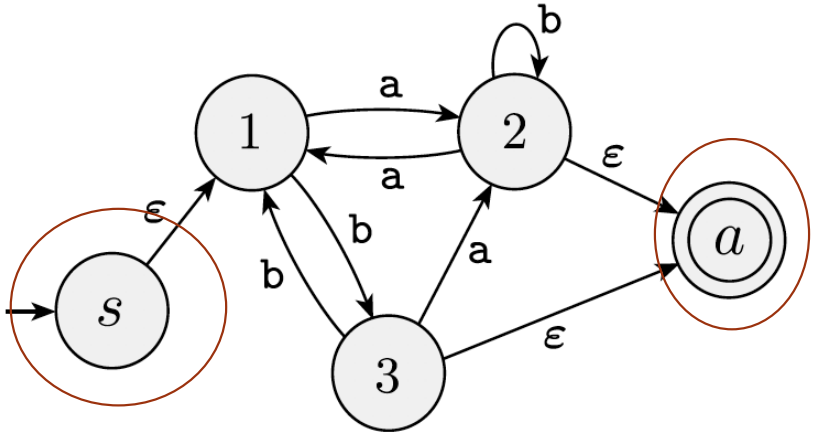
EXAMPLE

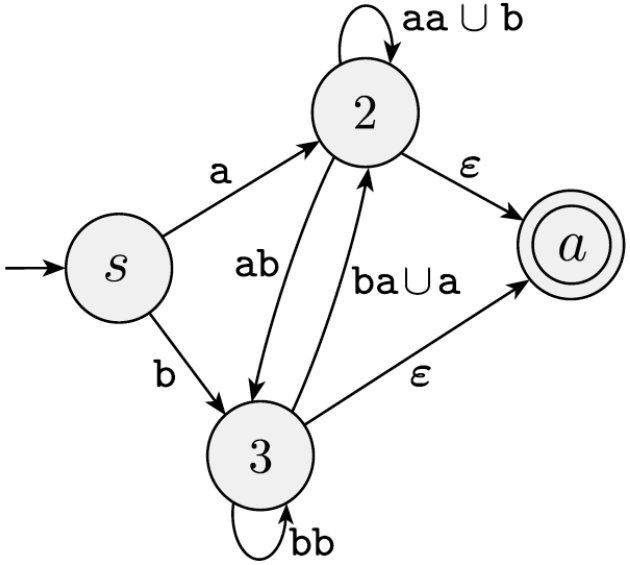


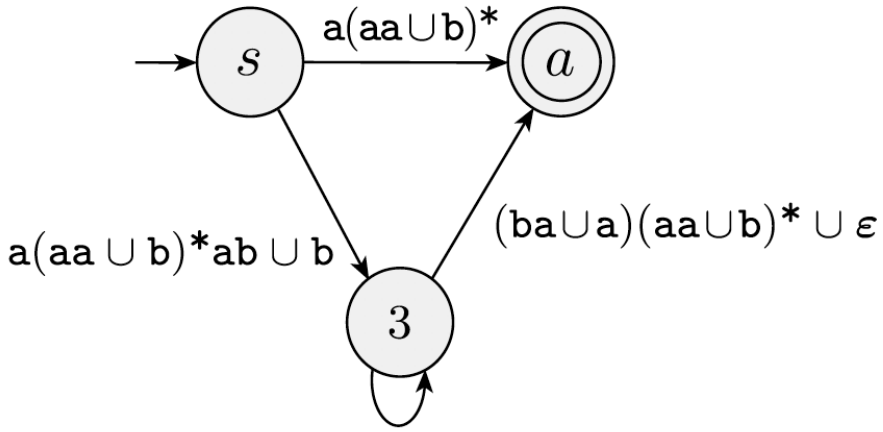
EXAMPLE – 5

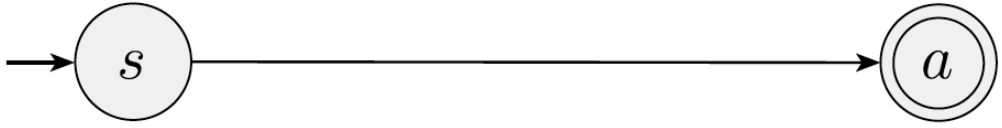


Step 1



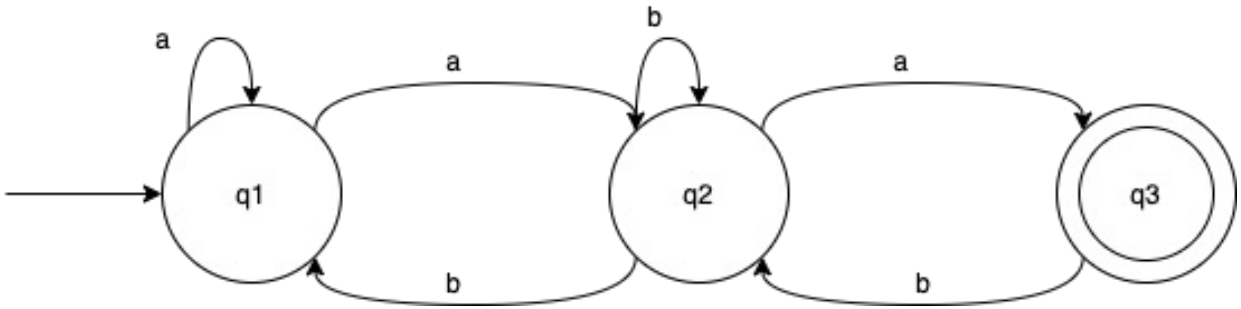






$$(a(aa \cup b)^*ab \cup b)((ba \cup a)(aa \cup b)^*ab \cup bb)^*((ba \cup a)(aa \cup b)^* \cup \varepsilon) \cup a(aa \cup b)^*$$

EXAMPLE – 6



HINTS

- The initial state of the DFA must not have any incoming edge.
- There must exist only one final state in the DFA.
- The final state of the DFA must not have any outgoing edge.
- Eliminate all the intermediate states one by one.
- These states may be eliminated in any order.
- Only an initial state going to the final state will be left.
- The transition is the required regular expression.
- Same direction and loops between two states → concatenation
- Opposite transitions and parallel transition → union
- Self loop → closure

TRY YOURSELF!