Regular Operation

Ref: prof chih-jen lin

Closure Under Regular Operation

In a deterministic finite automaton, the set of states that can be reached from a given state by following a specific input symbol is called the closure of that state.

If this set of reachable states is the same as the set of states in the automaton, then the automaton is said to be closed under that input symbol.

Similarly, in a nondeterministic finite automaton, the set of states that can be reached from a given state by following a specific input symbol is called the closure of that state. If this set of reachable states is a subset of the set of states in the automaton, then the automaton is said to be closed under that input symbol.

In both cases, the term "closed" is used to describe the property of the automaton being able to reach all of its states from a given state using the specified input symbol.

Closure Under Regular Operation

- Recall we defined three operations:

$$\cup, \circ, *$$

- For

$$A_1 \cup A_2,$$

  we proved that it's regular by constructing a new DFA

Closure Under Regular Operation

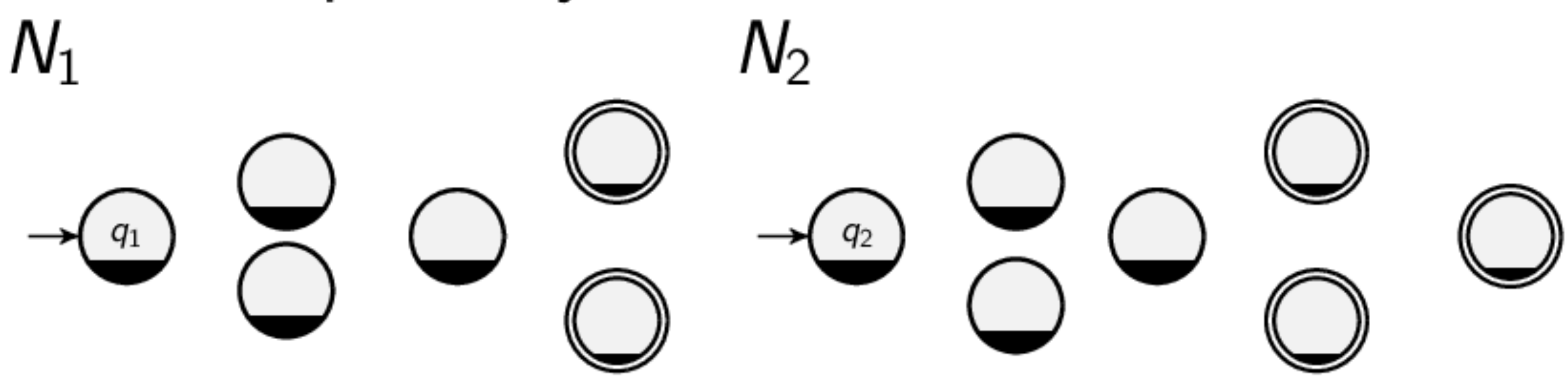
- But we had difficulties to prove that

$$A_1 \circ A_2$$

  is regular
- We will see that by using NFA, the proof is easier

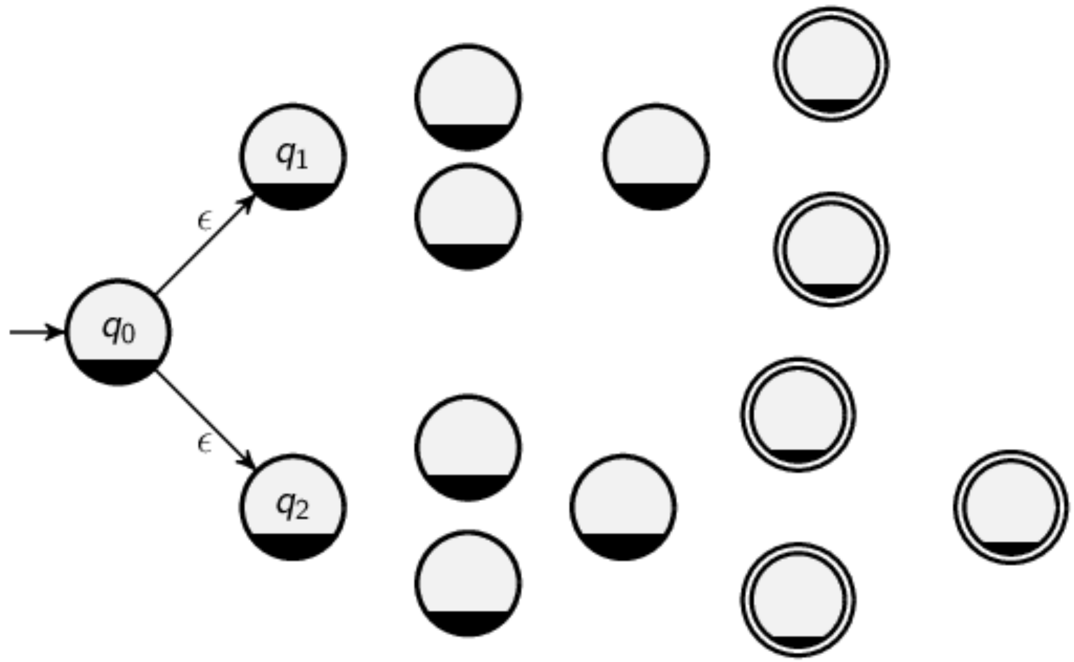
Union

- Given two regular languages A_1, A_2 under the **same** Σ
  Is $A_1 \cup A_2$ regular?
- To prove that a language is regular, by definition, it should be accepted by one DFA (or an NFA)
  We will construct an NFA for $A_1 \cup A_2$
- Assume A_1 and A_2 are recognized by two NFAs N_1 and N_2, respectively



Union

- We construct the following machine



- Formal definition

Union

Two NFAs:

$$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$
$$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

Note for NFA, $\epsilon \notin \Sigma$

Union

- New NFA

$$Q = Q_1 \cup Q_2 \cup \{q_0\}$$
$$q = q_0$$
$$F = F_1 \cup F_2$$

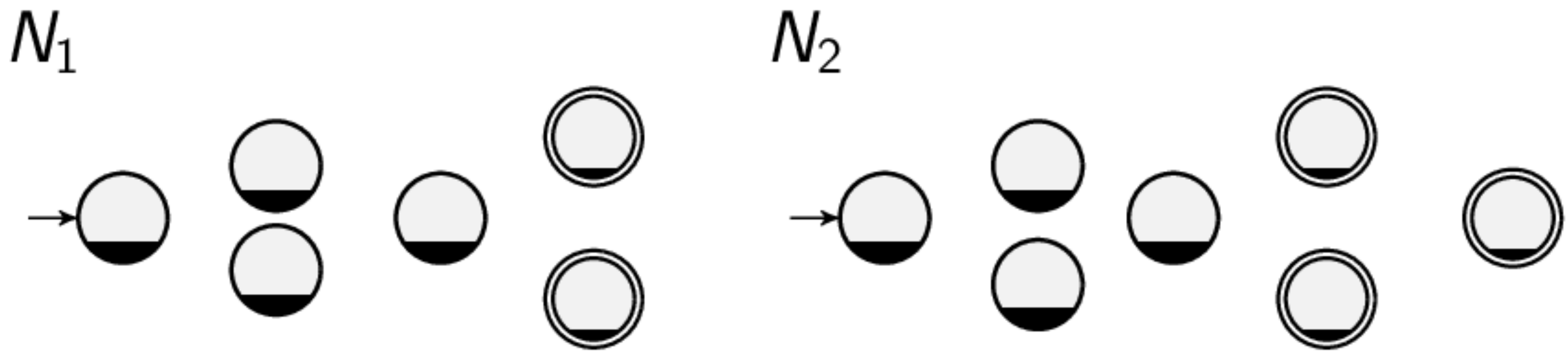
$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \epsilon \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon \end{cases}$$

Union

- The last case of δ is easily neglected

Closed under Concatenation

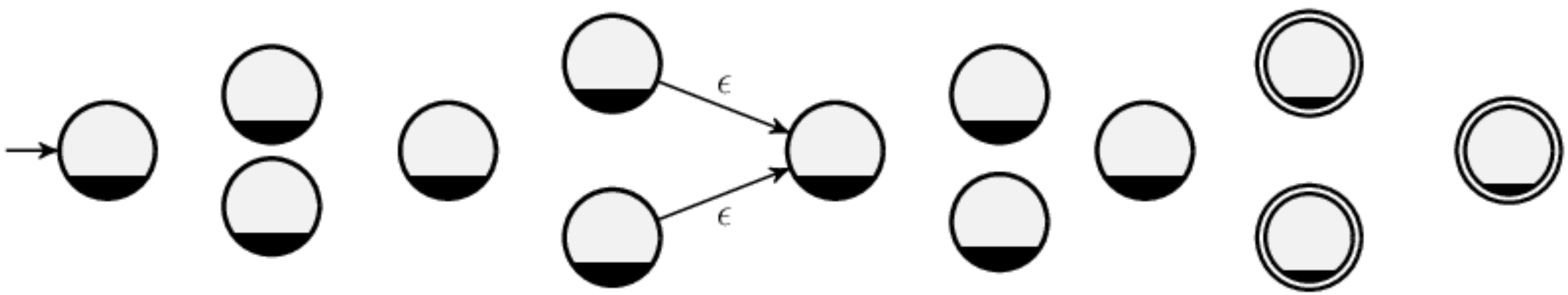
Given two NFAs



- Idea: from any accept state of N_1, add an ϵ link to q_2 (start state of N_2)
- Earlier in using DFA, the difficulty was that we didn't know where to cut the string to two parts

Closed under Concatenation

- Now we non-deterministically switch from the first to the second machine
- The new machine:



- Accept states of N_1 are no longer accept states in the new machine

Closed under Concatenation

- Formal definition. Given two automata

$$(Q_1, \Sigma, \delta_1, q_1, F_1)$$
$$(Q_2, \Sigma, \delta_2, q_2, F_2)$$

- New machine

$$Q = Q_1 \cup Q_2$$
$$q_0 = q_1$$
$$F = F_2$$

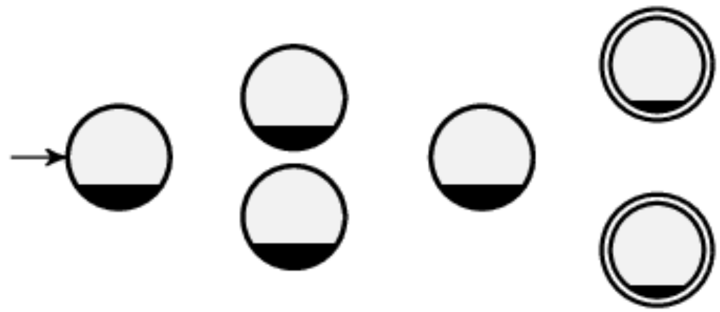
Closed under Concatenation

δ function:

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \setminus F_1 \\ \delta_2(q,a) & q \in Q_2 \\ \delta_1(q,\epsilon) \cup \{q_2\} & q \in F_1, a = \epsilon \\ \delta_1(q,a) & q \in F_1, a \neq \epsilon \end{cases}$$

Closed under star

- Given the following machine

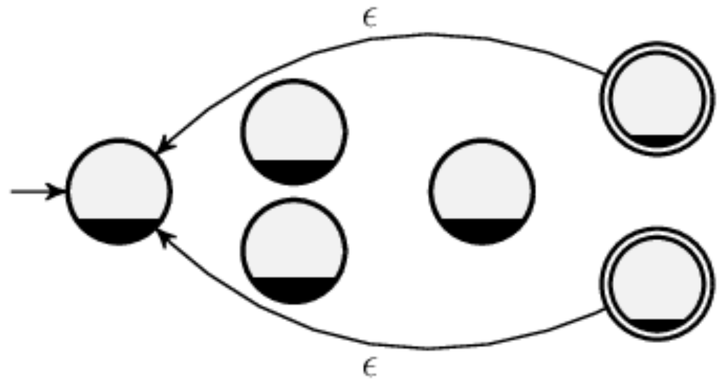


- Recall the star operation is defined as follows

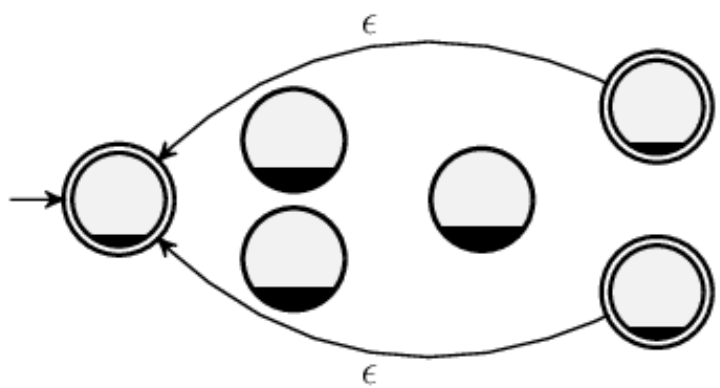
$$A^* = \{x_1 \cdots x_k \mid k \geq 0, x_i \in A\}$$

- The situation is related to $A_1 \circ A_2$, but we now work on the same machine A
- How about the following diagram

Closed under star

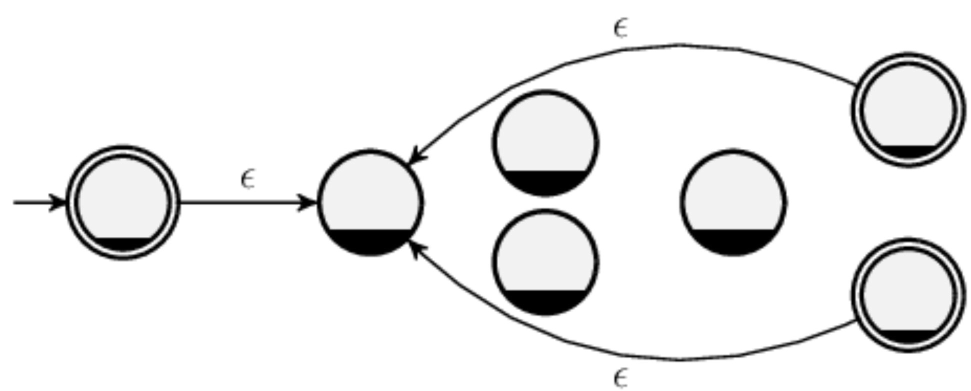


- The problem is that ϵ may not be accepted
- How about making the start state an accepting one



Closed under star

- This may make the machine to accept strings not in A
- Some strings reaching the start state in the end were rejected. But now may be accepted
- A correct setting



- Formal definition

Closed under star

- Given the machine

$$(Q_1, \Sigma, \delta_1, q_1, F_1)$$

- New machine:

$$Q = Q_1 \cup \{q_0\}$$
$$q_0 : \text{ new start state}$$
$$F = F_1 \cup \{q_0\}$$

Closed under star

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \setminus F_1 \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1, a = \epsilon \\ \delta_1(q,a) & q \in F_1, a \neq \epsilon \\ \{q_1\} & q = q_0, a = \epsilon \\ \emptyset & q = q_0, a \neq \epsilon \end{cases}$$

Regular Languages

Regular Languages

In computer science, a regular language is a language that can be described using a regular expression or a finite automaton.

Regular languages are a subset of the context-free languages, which in turn are a subset of the set of all possible languages that can be defined by a formal grammar.

Regular Languages

Regular languages are characterized by a set of rules, called regular expressions, that describe the pattern of the language.

Regular expressions use a combination of characters and special symbols to define a pattern that matches a specific set of strings.

For example, the regular expression a* matches any string that contains zero or more occurrences of the letter "a", while the regular expression a+b+ matches any string that contains one or more occurrences of the letter "a", followed by one or more occurrences of the letter "b".

Regular Languages

Regular languages have a number of useful properties, including the fact that they can be recognized and parsed efficiently using finite automata.

This makes them useful in a wide range of applications, such as text processing, compiler design, and natural language processing.

Empty Set and Empty String

In the context of formal languages, the terms "empty string" and "empty set" refer to two distinct concepts:

Empty string: The empty string is a string of length 0. It is denoted by the symbol ε or sometimes by λ. It represents a string with no characters or symbols. It is a valid string, but it contains no information.

Empty set: The empty set is a set with no elements. It is denoted by the symbol Ø. It represents a collection of objects with no members. It is also known as the null set.

Empty Set and Empty String

The difference between the empty string and the empty set is that the empty string is a string with no characters, while the empty set is a set with no elements.

The empty string is a member of the set of all strings, while the empty set is a subset of all sets.

In some cases, the empty string and the empty set may be used interchangeably, but it's important to understand their distinct meanings in formal language theory and in other areas of mathematics and computer science.

Regular Expression

Ref: prof chih-jen lin

Regular Expression

A regular expression (regex) is a sequence of characters that forms a search pattern, which is used to match and manipulate strings of text.

Regular expressions are a powerful tool for text processing and pattern matching, and are widely used in programming, text editors, and other applications that work with text.

A regular expression consists of a combination of characters and special symbols, which can be used to define a pattern of text that should be matched.

For example, the regular expression hello would match the string "hello", while the regular expression w.*d would match any string that starts with "w" and ends with "d", with any number of characters in between.

Regular Expression

Regular expressions are supported by most programming languages, including Java, Python, and Perl, as well as text editors like Vim and Emacs.

They can be used for a variety of tasks, such as searching and replacing text, validating input data, and parsing structured text formats like XML and JSON.

Regular Expression and Regular Languages

Regular expressions and regular languages are related concepts in computer science, but they refer to different things.

A regular expression is a pattern of characters that is used to describe a set of strings. It is a compact way of specifying a regular language.
Regular expressions are used in many programming languages and tools for string manipulation and searching, and they allow for powerful and flexible matching of text patterns.

Regular Expression and Regular Languages

Regular expressions and regular languages are related concepts in computer science, but they refer to different things.

A regular language, on the other hand, is a set of strings that can be described by a regular expression or recognized by a finite automaton.
Regular languages are a fundamental concept in formal language theory and are widely used in programming and computer science.
They include simple languages such as the set of all strings of 0's and 1's with an even number of 0's, as well as more complex languages such as the set of all valid email addresses.

Regular Expression and Regular Languages

Regular expression is a compact notation for specifying a regular language, while a regular language is a set of strings that can be recognized by a finite automaton or described by a regular expression.

Regular Expression

- Example

$$(0 \cup 1)0^*$$

- This is a simplification of

$$(\{0\} \cup \{1\}) \circ \{0\}^*$$

- Regular expressions are practically useful

  Example: finding lines in a file containing "a" or "b"

  `egrep '(a|b)' file`

Regular Expression

- Formally this means we consider the language

$$\Sigma^* a \Sigma^* \cup \Sigma^* b \Sigma^*$$

  Σ^*: all strings over Σ

- Example

$$(0 \cup 1)^*$$

  all strings by 0 and 1

- Example:

$$(0\Sigma^*) \cup (\Sigma^* 1)$$

  all strings start with 0 or end with 1

Regular Expression - Definition

- R is a regular expression if it is one of the following expressions
  1. a, where $a \in \Sigma$
  2. ϵ ($\epsilon \notin \Sigma$)
  3. $\emptyset$
  4. $R_1 \cup R_2$, where R_1, R_2 are regular expressions
  5. $R_1 \circ R_2$, where R_1, R_2 are regular expressions
  6. R_1^*, where R_1 is a regular expression
- $\emptyset$ and ϵ

  ϵ: empty string

Regular Expression - Definition

$\emptyset$: empty language (language without any string)

$$(0 \cup \epsilon)1^* = 01^* \cup 1^*$$
$$(0 \cup \emptyset)1^* = 01^*$$
$$\emptyset 1^* = 1^* \emptyset = \emptyset$$

Concatenating 1^* with nothing $\Rightarrow$ nothing

- We have an inductive definition

An expression is constructed from smaller strings

Regular Expression Example

- 0^*10^*: strings with exactly one 1
- $(\Sigma\Sigma)^*$: strings with even length
- Assume $\Sigma = \{0, 1\}$

$$0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$$

  Strings that start and end with the same symbol
- $\emptyset^* = \{\epsilon\}$
- $R \cup \emptyset = R$
- $R \circ \epsilon = R$

Regular Expression Example

$$(+ \cup - \cup \epsilon)(DD^* \cup DD^*.D^* \cup D^*.DD^*),$$

where

$$D = \{0, \ldots, 9\}$$

- 72, 2.1, 7., -.01

$$72 \in DD^*$$
$$2.1 \in DD^*.D^*$$
$$7. \in DD^*.D^*$$
$$.01 \in D^*.DD^*$$

Regular Expression Floating Number

- Why not $D^*.D^*$

  . is not allowed

Finite Automata

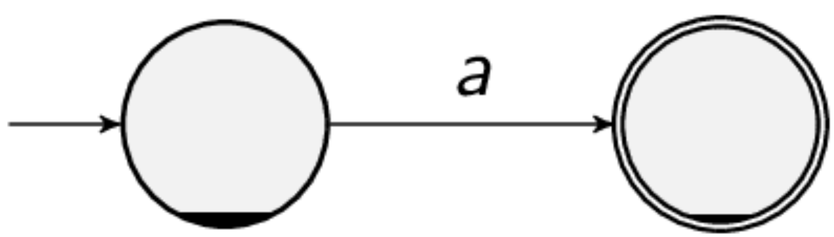
- They have equivalent descriptive power
- language regular $\Leftrightarrow$ described by regular expression

Regular Expression – 1.55

- Language by a regular expression
  $\Rightarrow$ regular (described by an automaton)
- The proof is by induction. We go through all cases in the definition
- $R = a \in \Sigma$

  Can such a language be recognized by an NFA?

  This language has only one string and can be recognized by



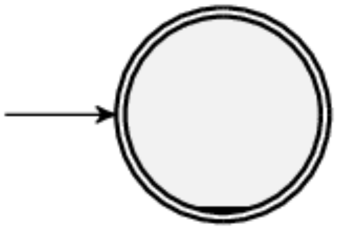
Regular Expression – 1.55

Note that this is an NFA.

Formal definition:

$$N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$$
$$\delta(q_1, a) = \{q_2\}$$
$$\delta(r, b) = \emptyset, r \neq q_1 \text{ or } b \neq a$$

- $R = \epsilon$

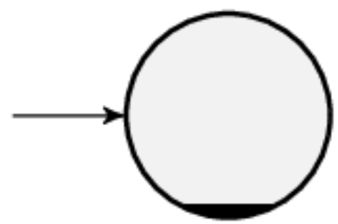


Regular Expression – 1.55

Formal definition

$$N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$$
$$\delta(q_1, a) = \emptyset, \forall a$$

- $R = \emptyset$



Regular Expression – 1.55

Formal definition

$$N = (\{q\}, \Sigma, \delta, q, \emptyset)$$
$$\delta(r, a) = \emptyset, \forall r, a$$

Note: earlier we only say $F \subset Q$, so F can be $\emptyset$

- For the other three situations

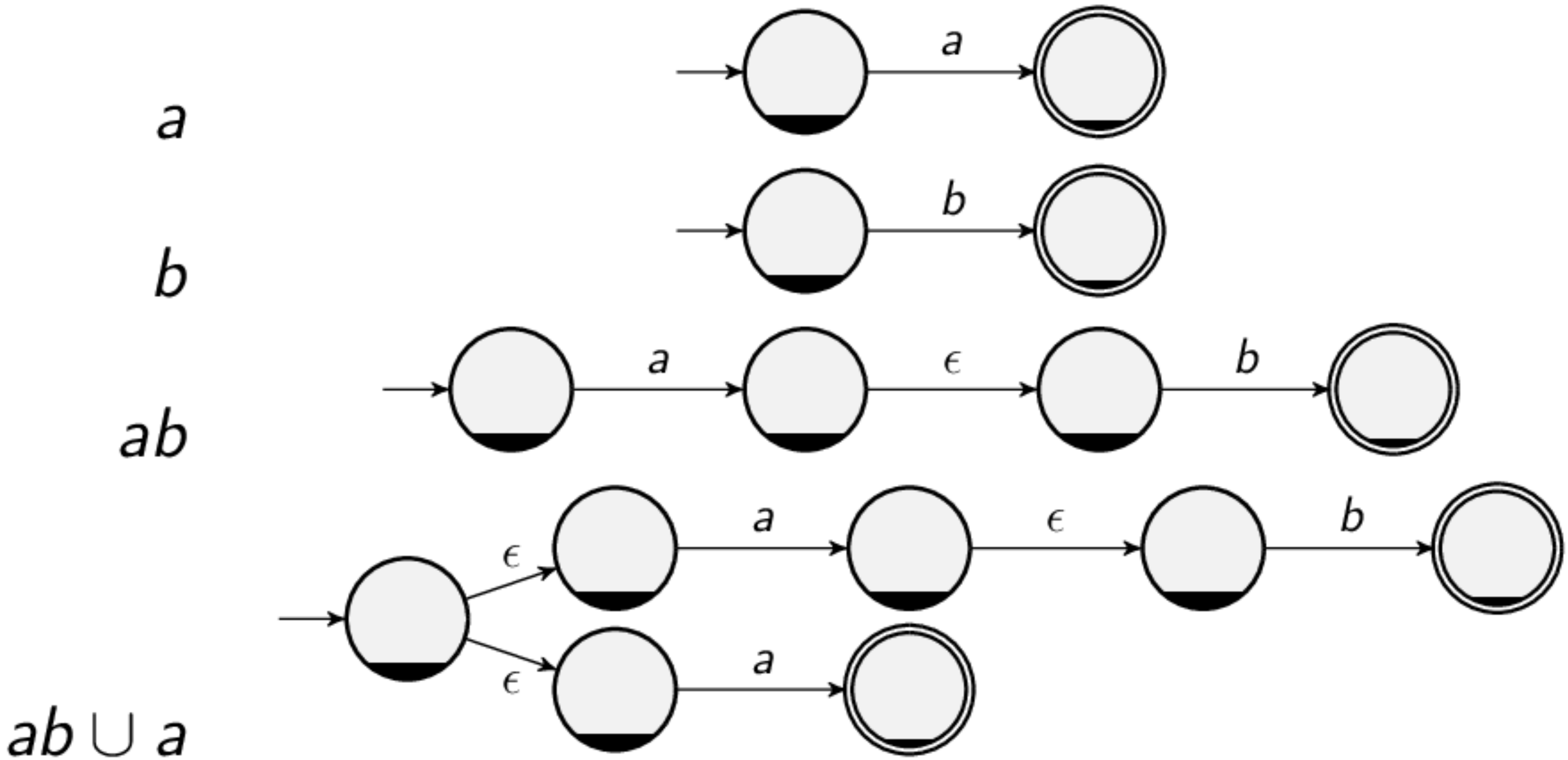
$$R = R_1 \cup R_2$$
$$R = R_1 \circ R_2$$
$$R = R_1^*$$

we use the proof in NFAs

Regular Expression – 1.55

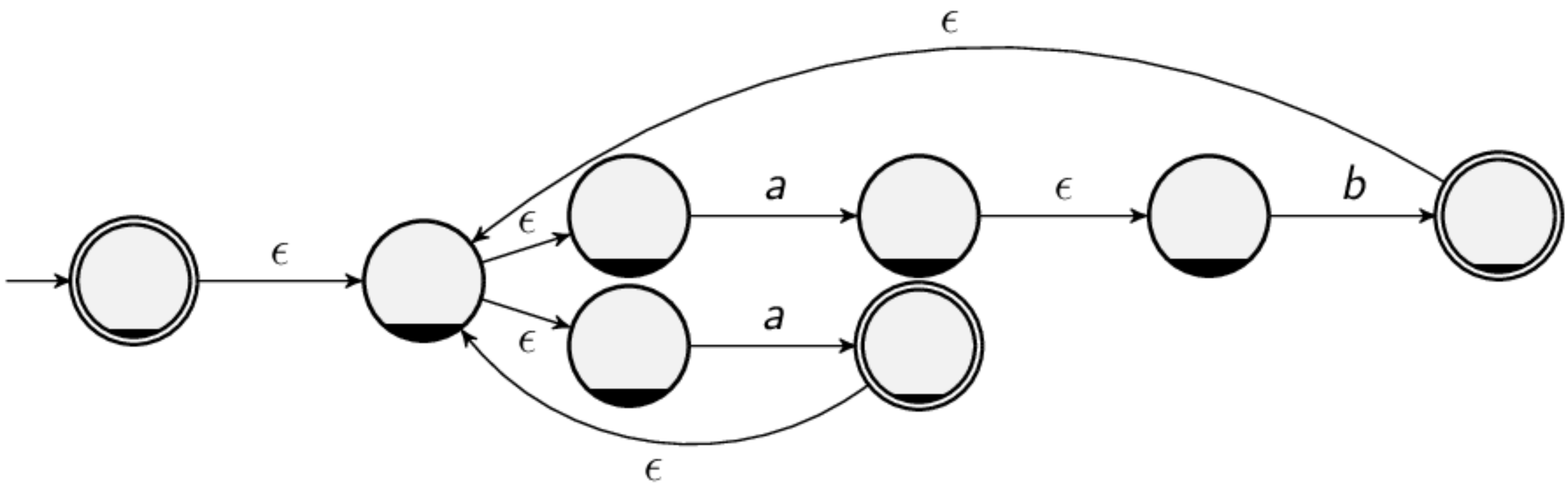
- We will see details by an example

Regular Expression – 1.57 (ab U a)*



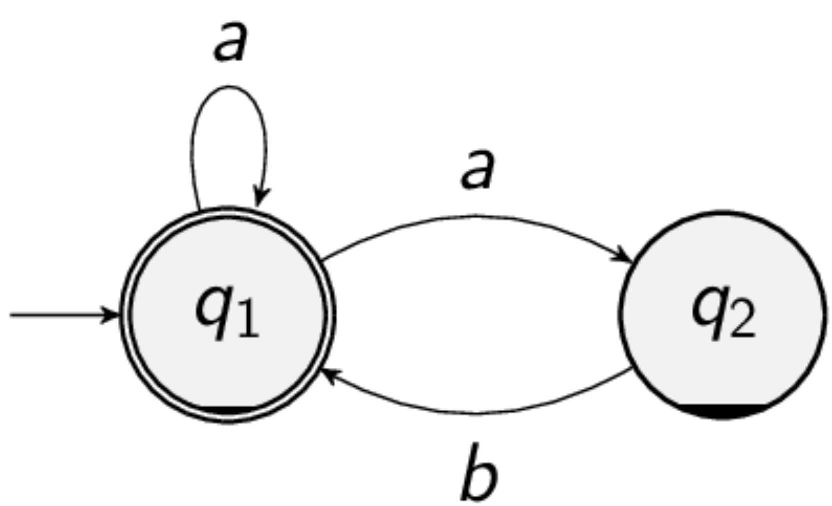
Regular Expression – 1.57 (ab U a)*

$(ab \cup a)^*$



- Such a construction usually does not give an NFA with the smallest number of states
- This example $\Rightarrow$ can be by 2 states

Regular Expression – 1.57 (ab U a)*

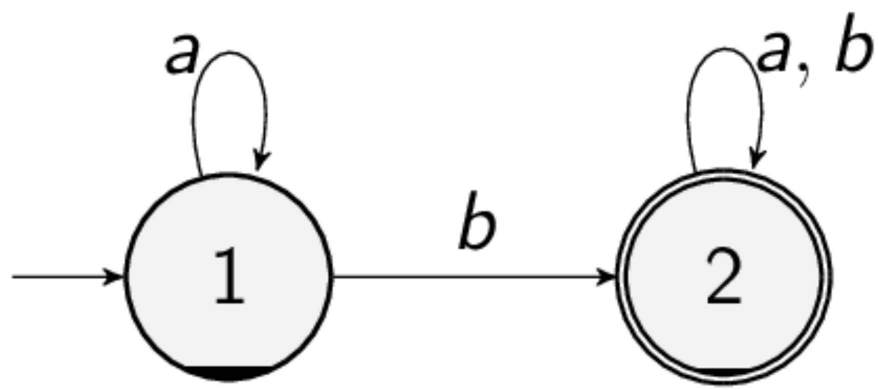


$$(\{q_1, q_2\}, \{a, b\}, \delta, q_1, \{q_1\})$$

	a	b	ϵ
q_1	$\{q_1, q_2\}$	$\emptyset$	$\emptyset$
q_2	$\emptyset$	$\{q_1\}$	$\emptyset$

Regular Expression – Simple Example

- Given an NFA, how can we convert it to a regular expression?
- Example:



- Quickly we see that this corresponds to

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

Regular Expression – Simple Example

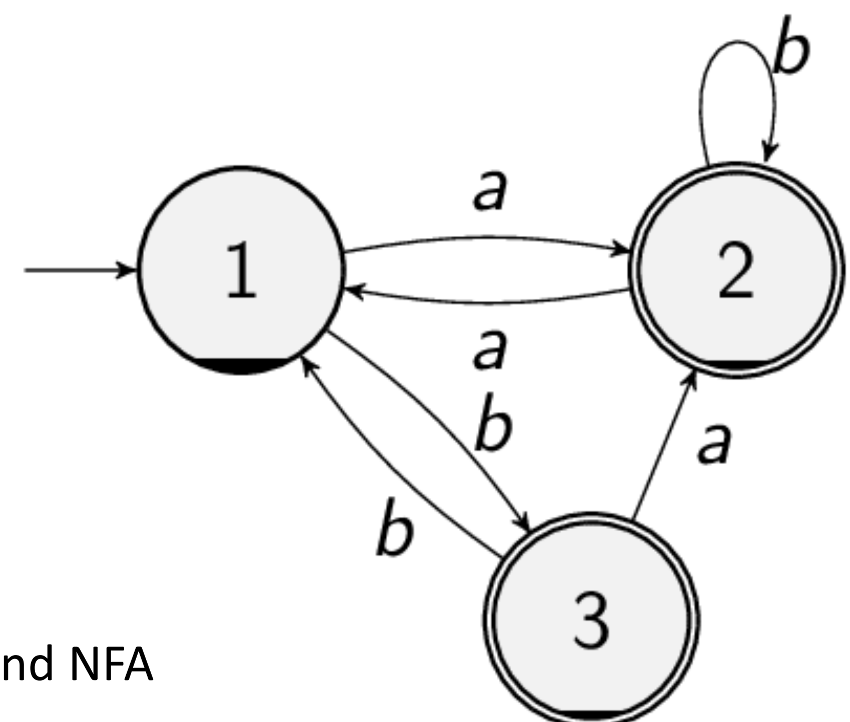
- However, in other situations we may not easily see what the regular expression is

Regular Expression

Fix the NFA (new start state and new acceptance state)
Pick States repeatedly with few transitions
Repeat

- Example 1.68

  Consider the following DFA

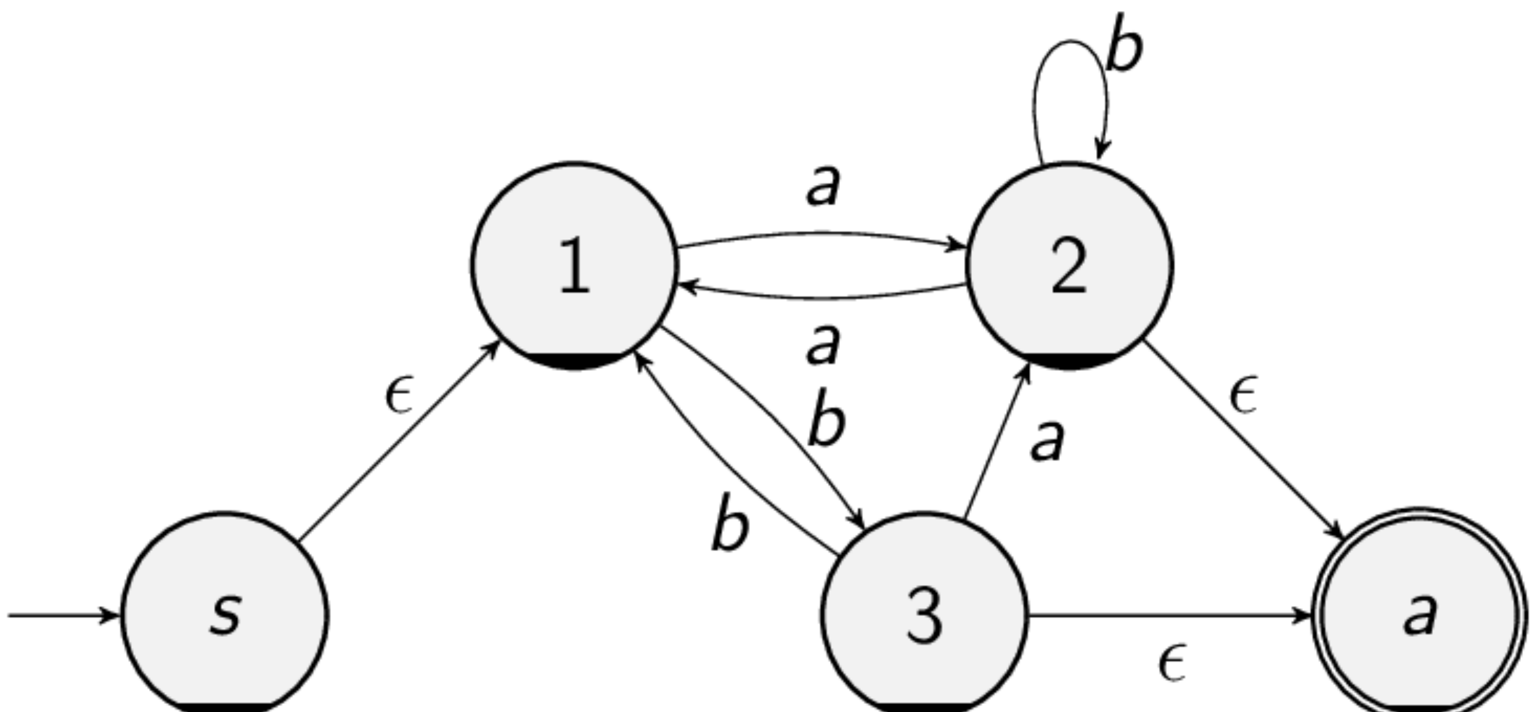


This can be done in a DFA and NFA

In DFA there is no epsilon link

Regular Expression

- It is not that easy to directly see what the regular expression is
- We need a procedure shown below
- First, add a start and an accept states



Regular Expression

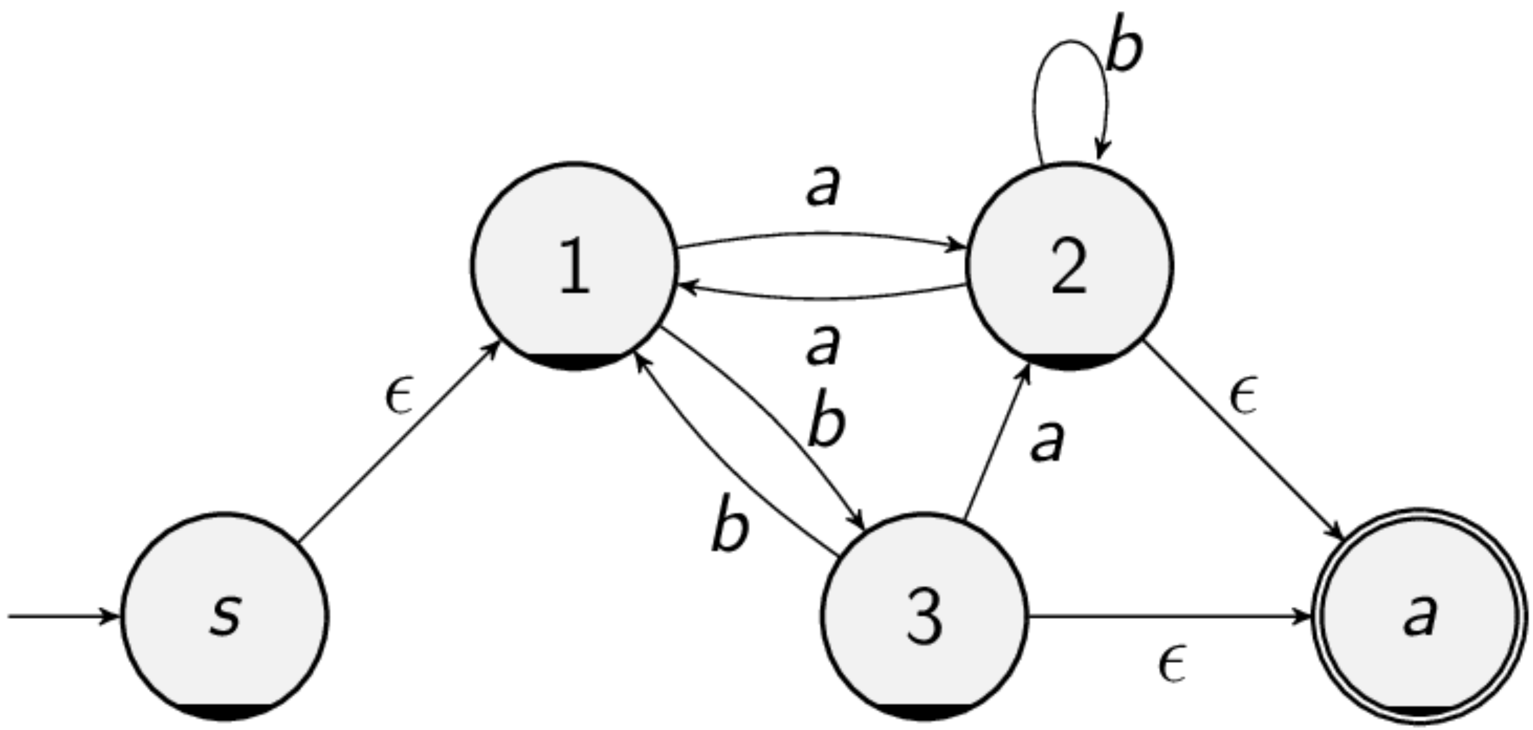
- This generates a generalized NFA (GNFA)
- Our procedure is

$$\text{DFA} \rightarrow \text{GNFA} \rightarrow \text{regular expression}$$

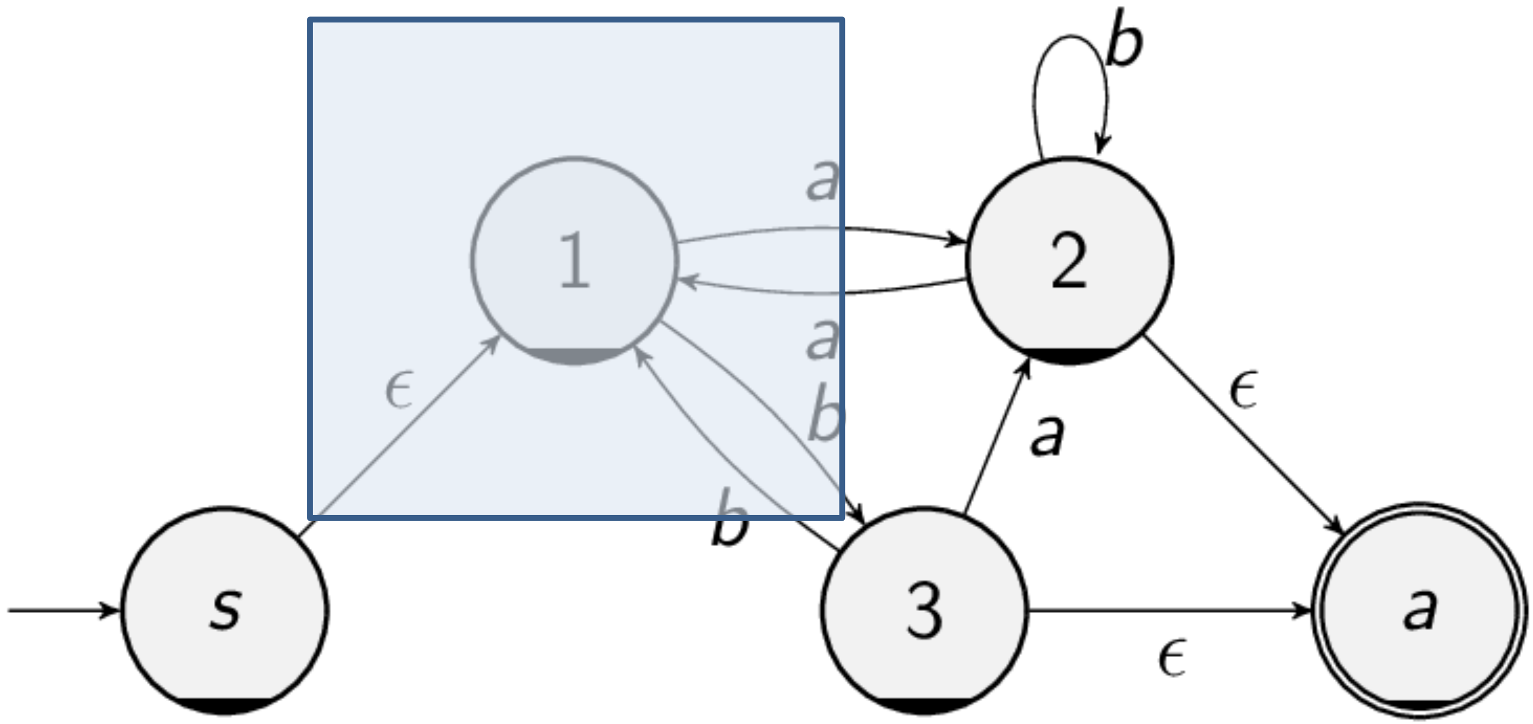
- Remove state 1

Regular Expression

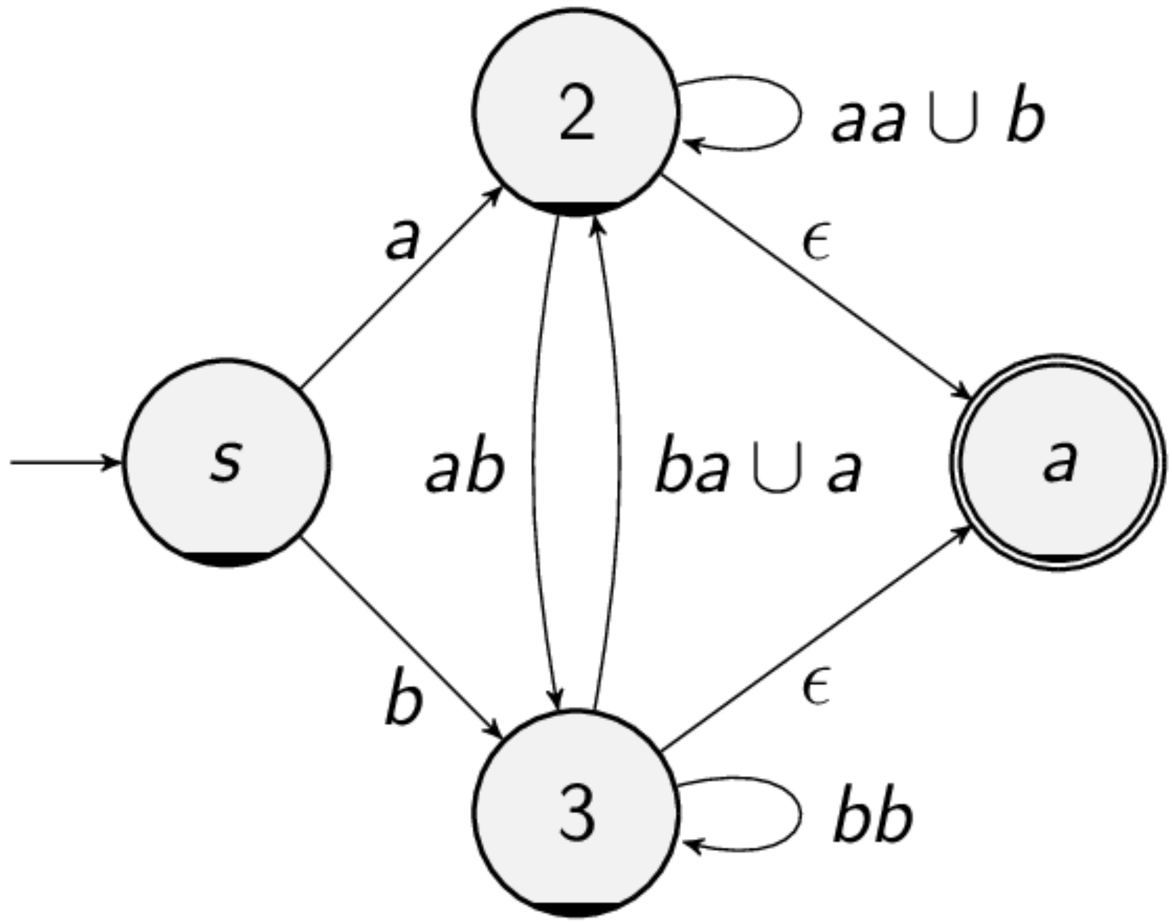
Fix the NFA (have one start state and one end state)
Pick States repeatedly with few transitions
Repeat
Result one start to one end state and transition in between



Regular Expression



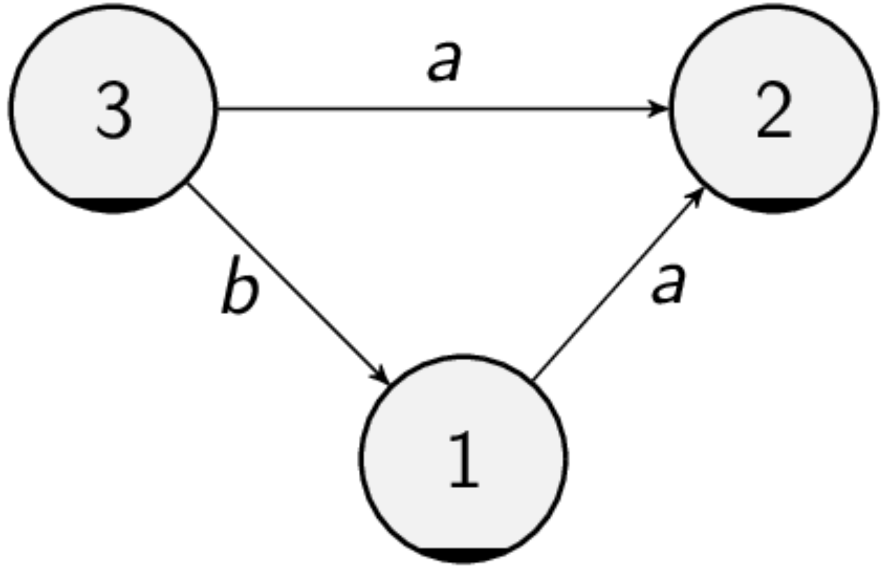
Regular Expression



- Example: the link

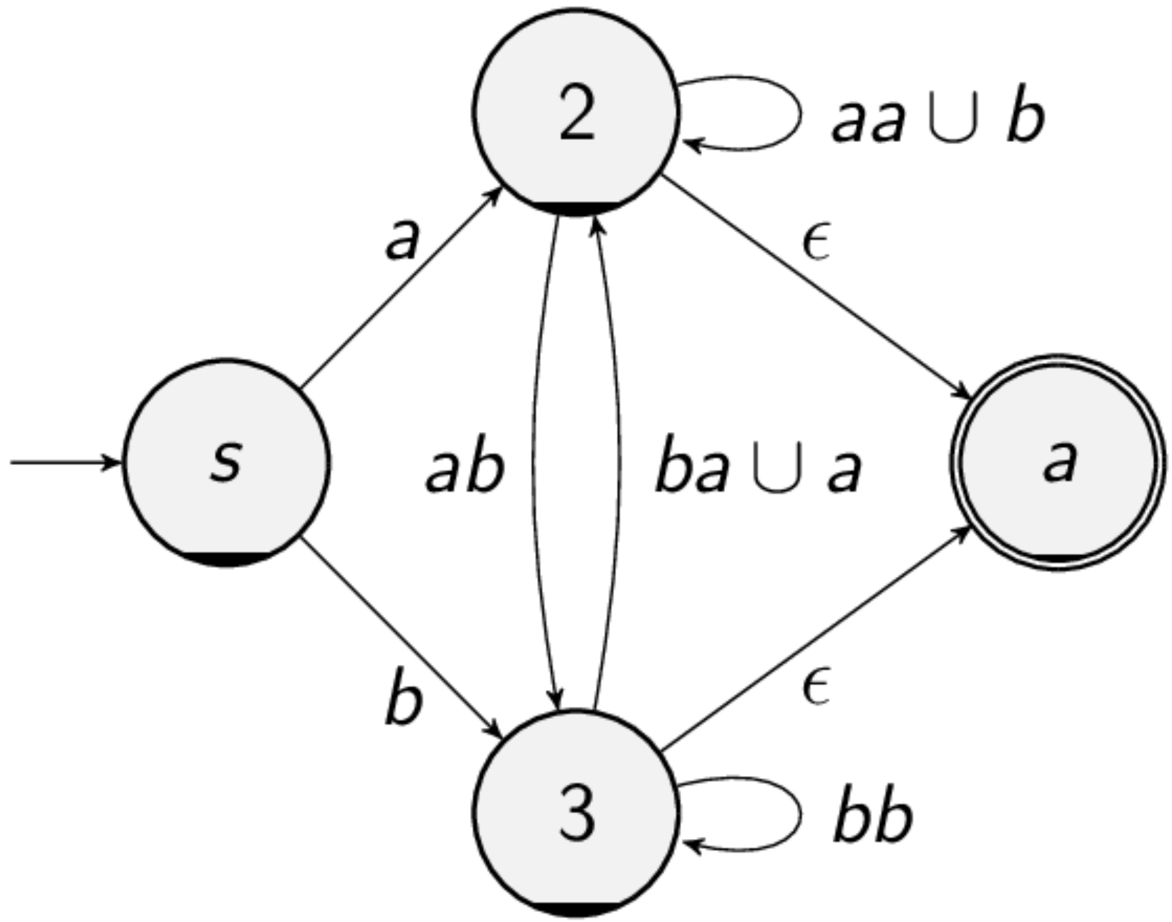
$$3 \to 2$$

Regular Expression



- Thus $ba \cup a$
- Idea: now 1 is removed. Need to check how we can go from 3 to 2 via state 1
- Need to check all pairs of states
- Remove state 2

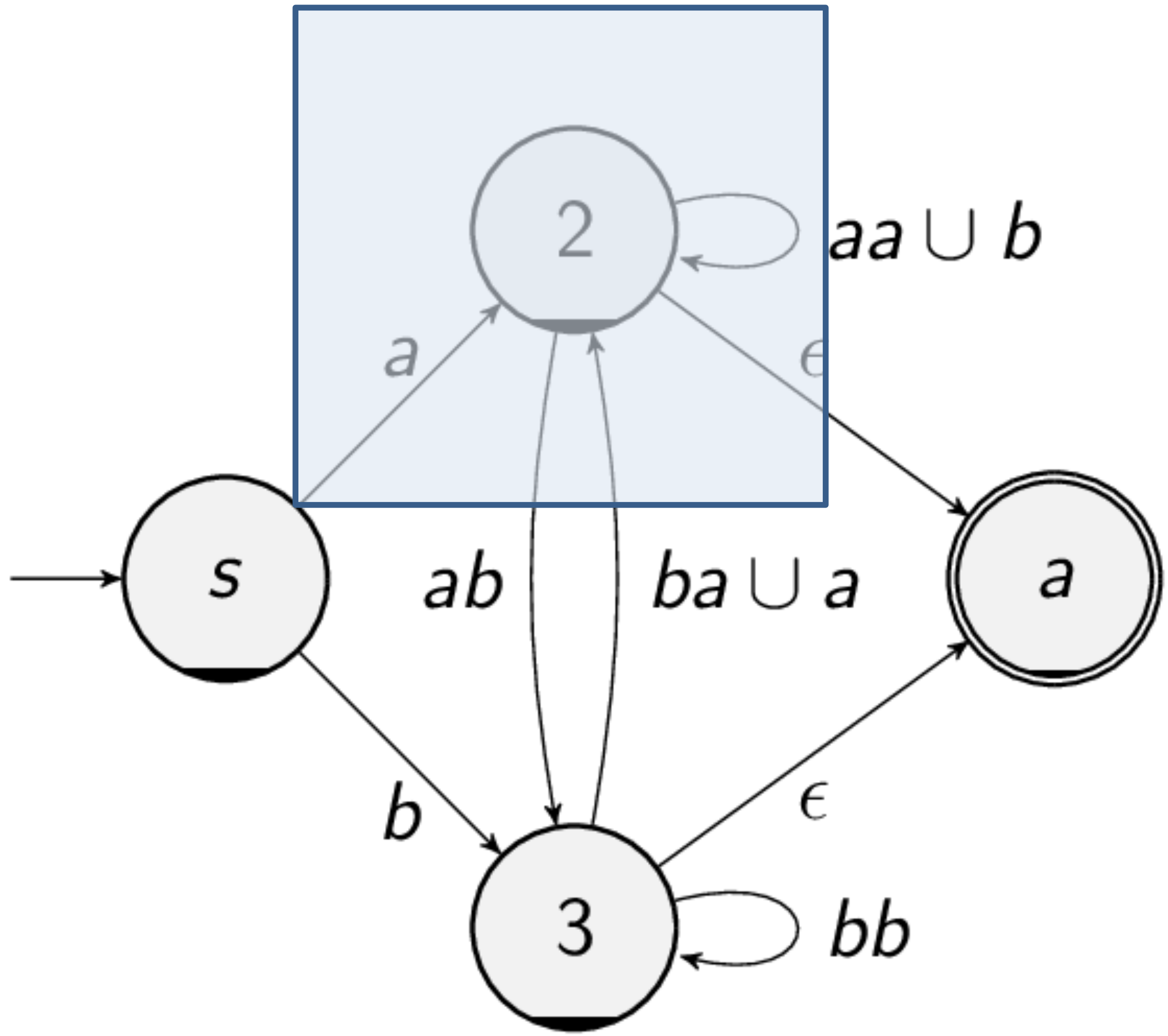
Regular Expression



- Example: the link

$$3 \rightarrow 2$$

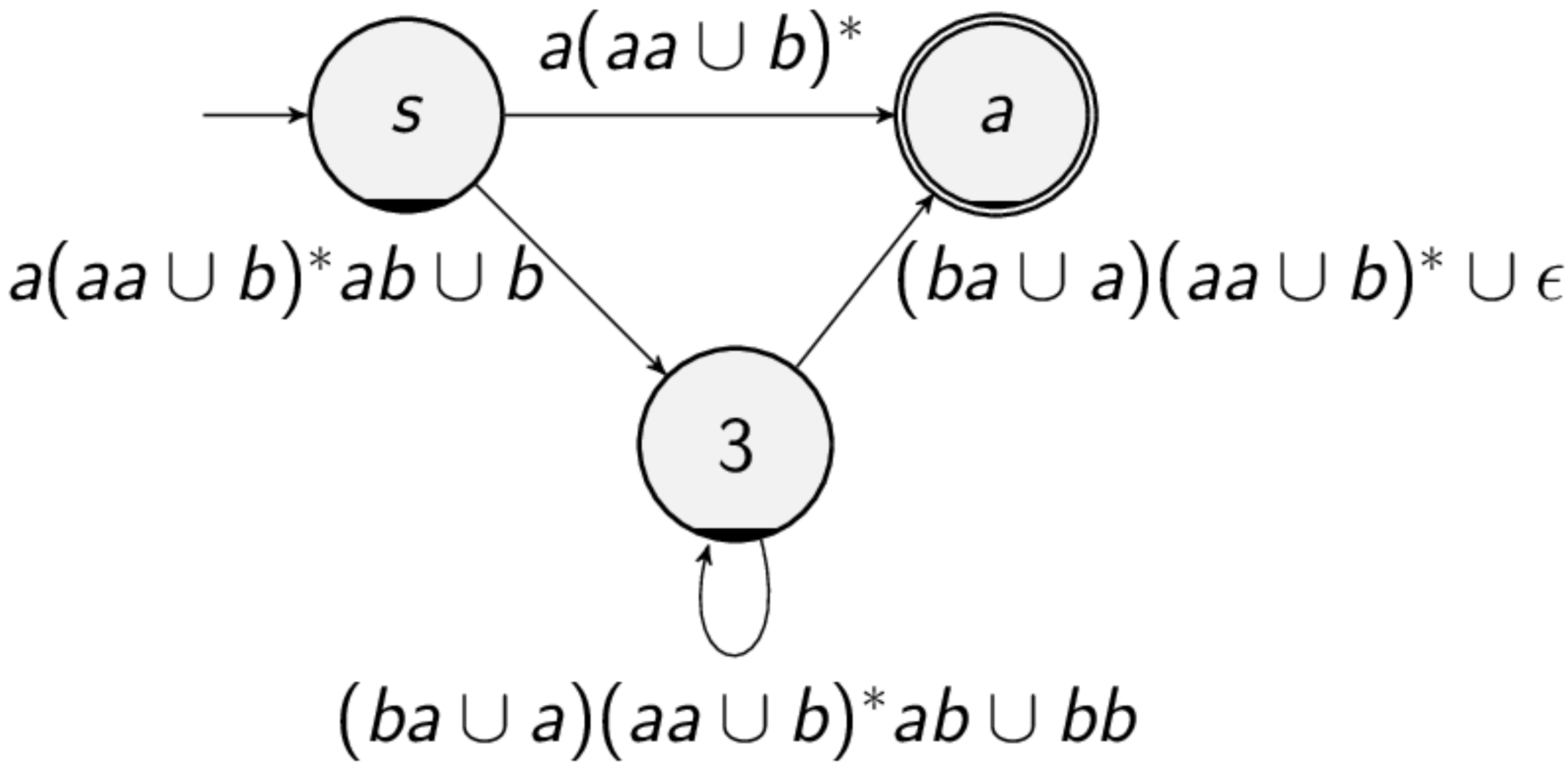
Regular Expression



- Example: the link

$$3 \to 2$$

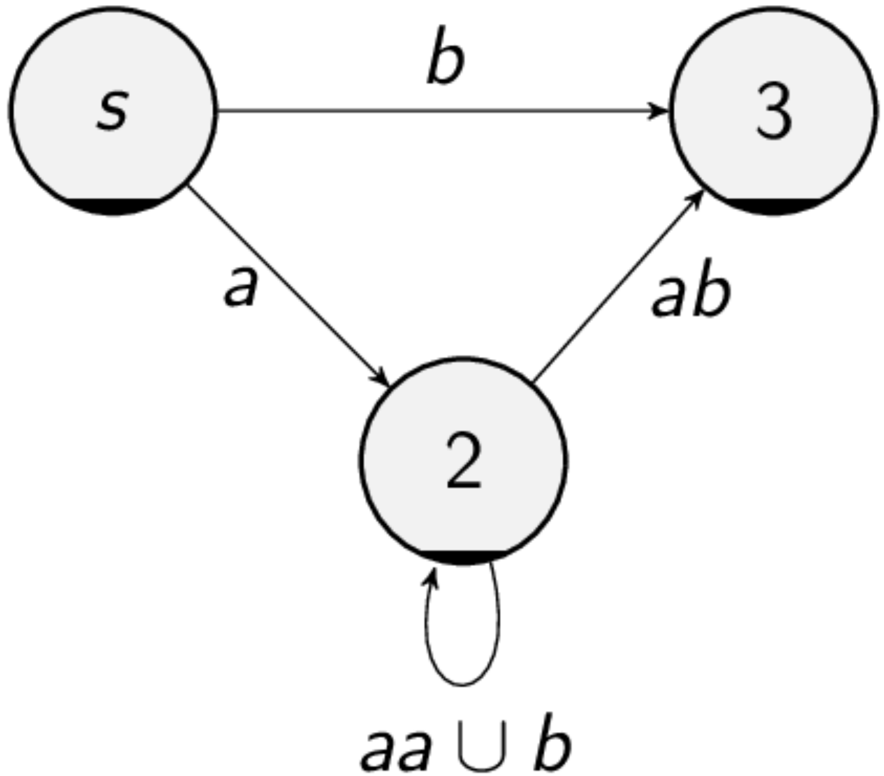
Regular Expression



- Example:

$$s \to 3$$

Regular Expression



- Thus $a(aa \cup b)^* ab \cup b$

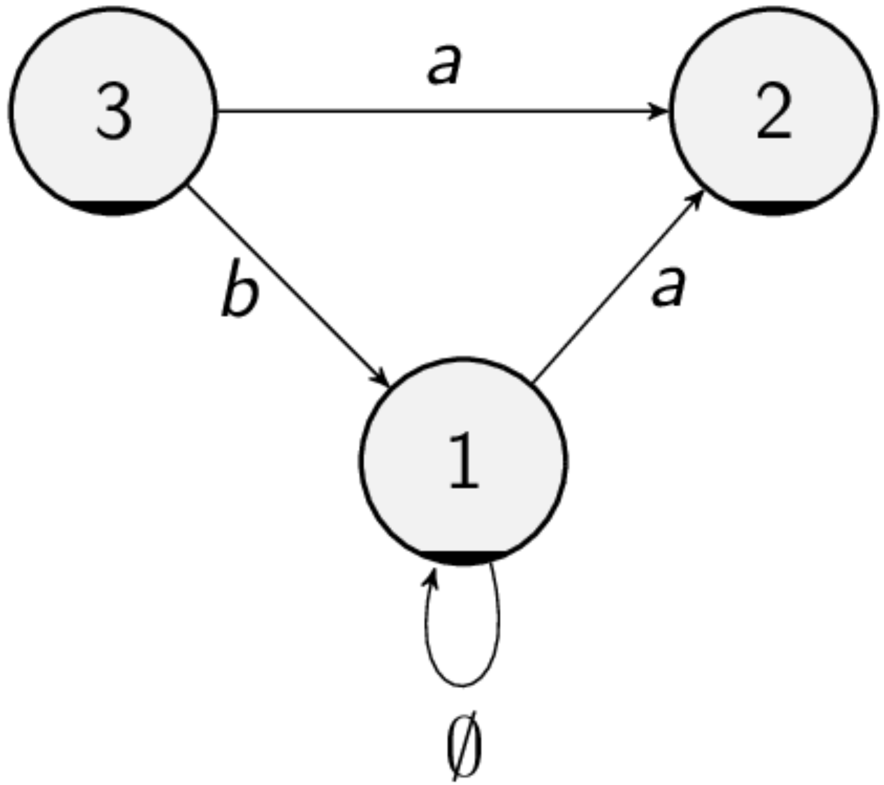
Regular Expression

- Here we need to handle

$$2 \xrightarrow{aa \cup b} 2$$

- Thus in the early example of removing state 1, we actually have

Regular Expression



and

$$b\emptyset^* a \cup a = b\epsilon a \cup a = ba \cup a$$

- Remove state 3

Regular Expression – Simple Example

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

- We will formally explain the procedure

GNFA

- Here we give the formal definition of generalized NFA
- Between any two states: a regular expression
- $(Q, \Sigma, \delta, q_{start}, q_{accept})$
- Single accept state. No longer a set F
- The δ function:

$$(Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to R$$

  R: all regular expressions over Σ
- DFA $\to$ GNFA

GNFA

Two new states: q_{start}, q_{accept}

$$q_{start} \rightarrow q_0 \text{ with } \epsilon$$
$$\text{any } q \in F \rightarrow q_{accept} \text{ with } \epsilon$$

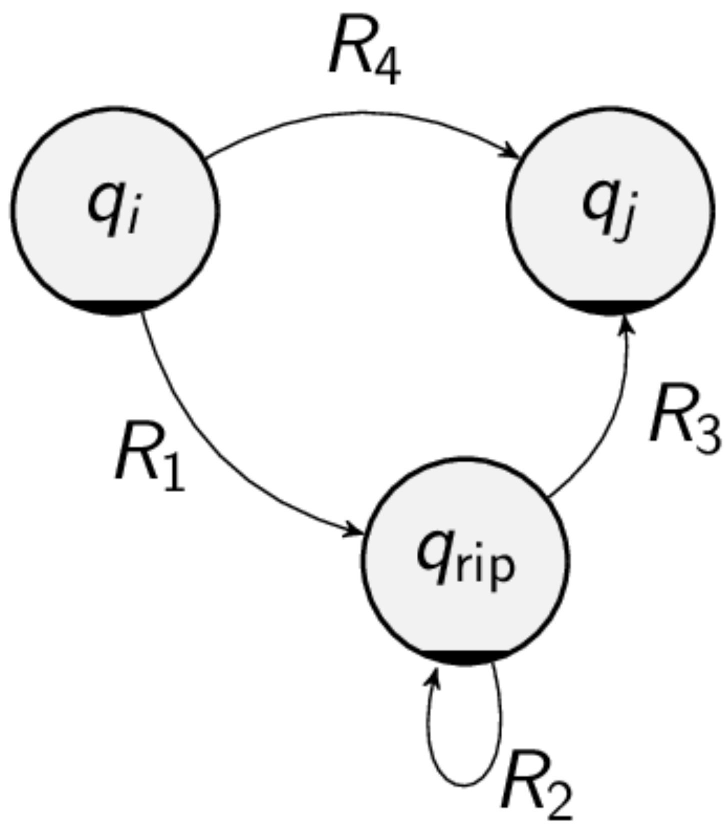
- In the definition, between any two states there is an expression

  But what if in the graph two states are not connected ?

  $\emptyset \in R$ so if no connection, we simply consider $\emptyset$ as the expression between two states

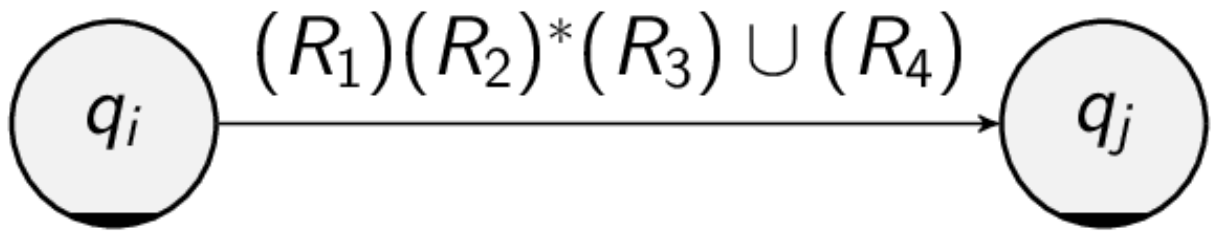
GNFA –> Regular Expression

- q_{rip} is the state being removed



- The new regular expression between q_i and q_j is

GNFA –> Regular Expression



- In our example
  3-state DFA $\rightarrow$ 5-state GNFA $\rightarrow$ 4-state $\cdots \rightarrow$
  2-state GNFA $\rightarrow$ regular expression
- In the procedure any any (i, j) related to q_{rip} considered
- Algorithm: convert(G)
  1. k: $\#$ of G
  2. If $k = 2$

GNFA –> Regular Expression

$$\text{return } R \text{ between } q_s \text{ and } q_a$$

3. If $k > 2$, choose any $q_{\text{rip}} \in Q \setminus \{q_s, q_a\}$ for removal

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

$$\forall q_i \in Q' - \{q_{accept}\}, q_j \in Q' - \{q_{start}\}$$

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

where

$$R_1 = \delta(q_i, q_{\text{rip}}), R_2 = \delta(q_{\text{rip}}, q_{\text{rip}}),$$
$$R_3 = \delta(q_{\text{rip}}, q_j), R_4 = \delta(q_i, q_j)$$

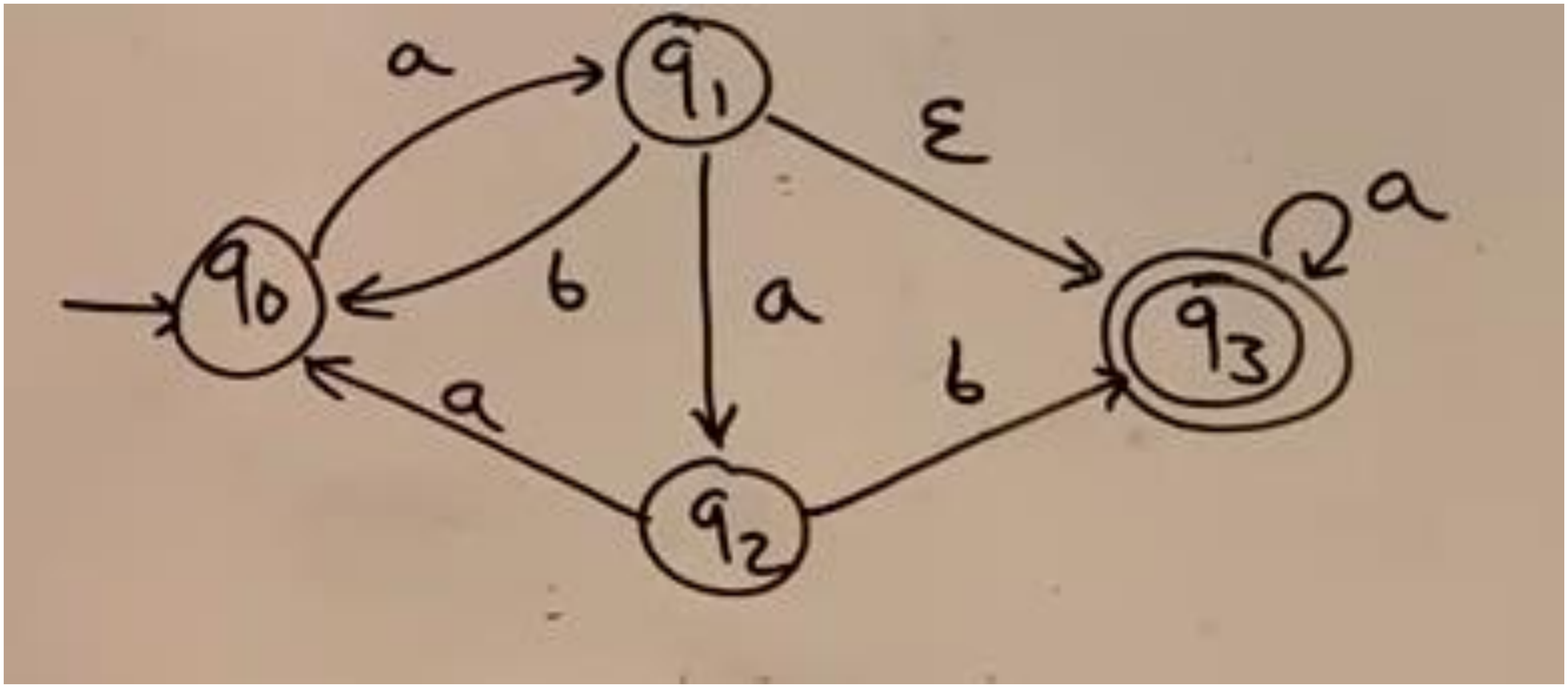
GNFA –> Regular Expression

4. Run convert(G'), where

$$G' = (Q', \Sigma, \delta', q_s, q_a)$$

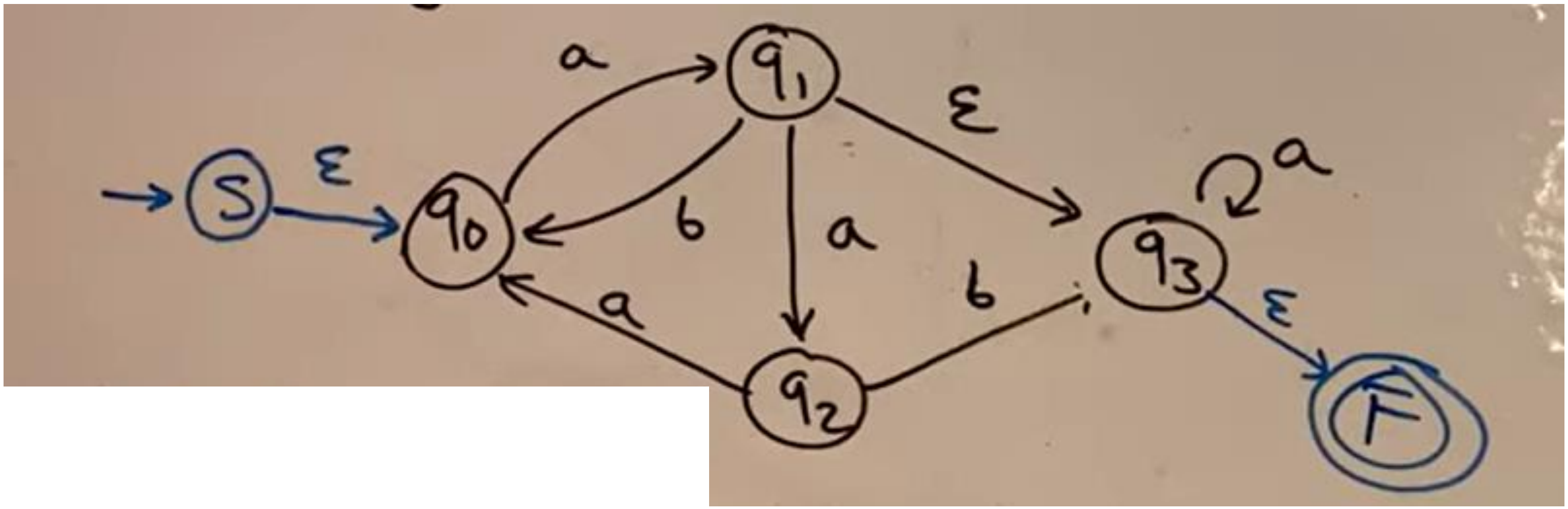
- You can see we have a recursive setting
  The process stops when $k = 2$
- Why in the textbook we modify DFA to GNFA?
  Is it ok to use NFA?
  Seems ok??

GNFA –> Regular Expression



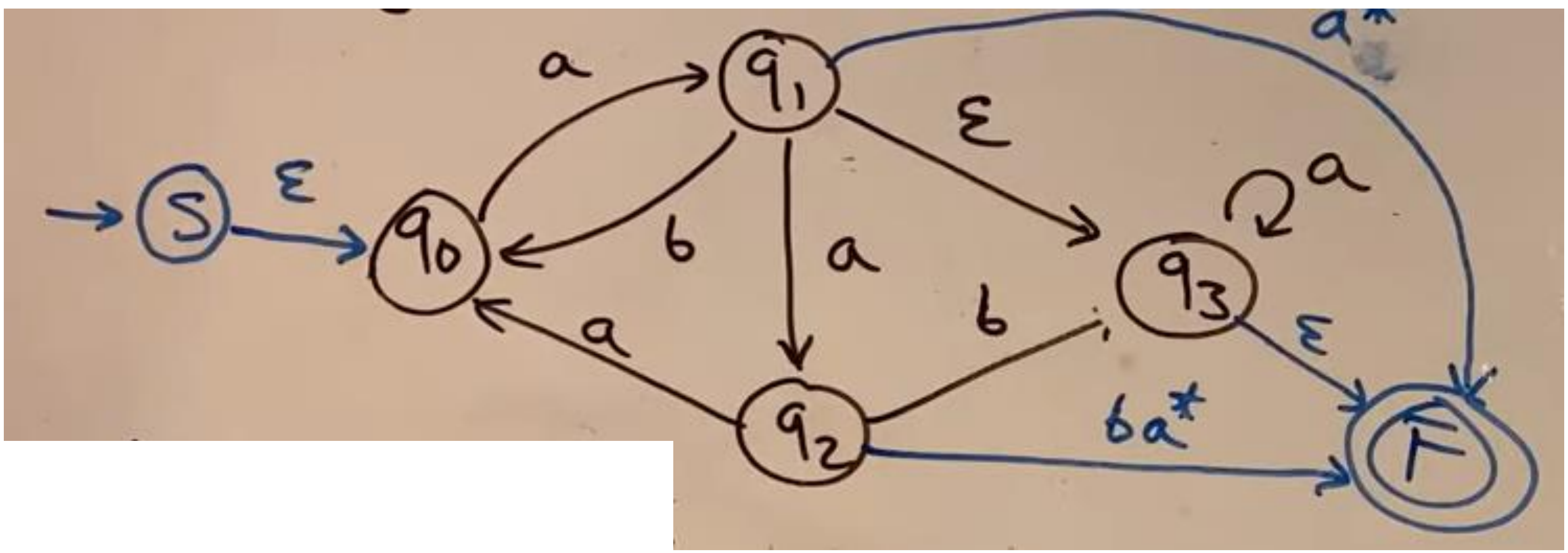
Fix the NFA (have one start state and one end state)
Pick States repeatedly with few transitions
Repeat
Result one start to one end state and transition in between

GNFA –> Regular Expression



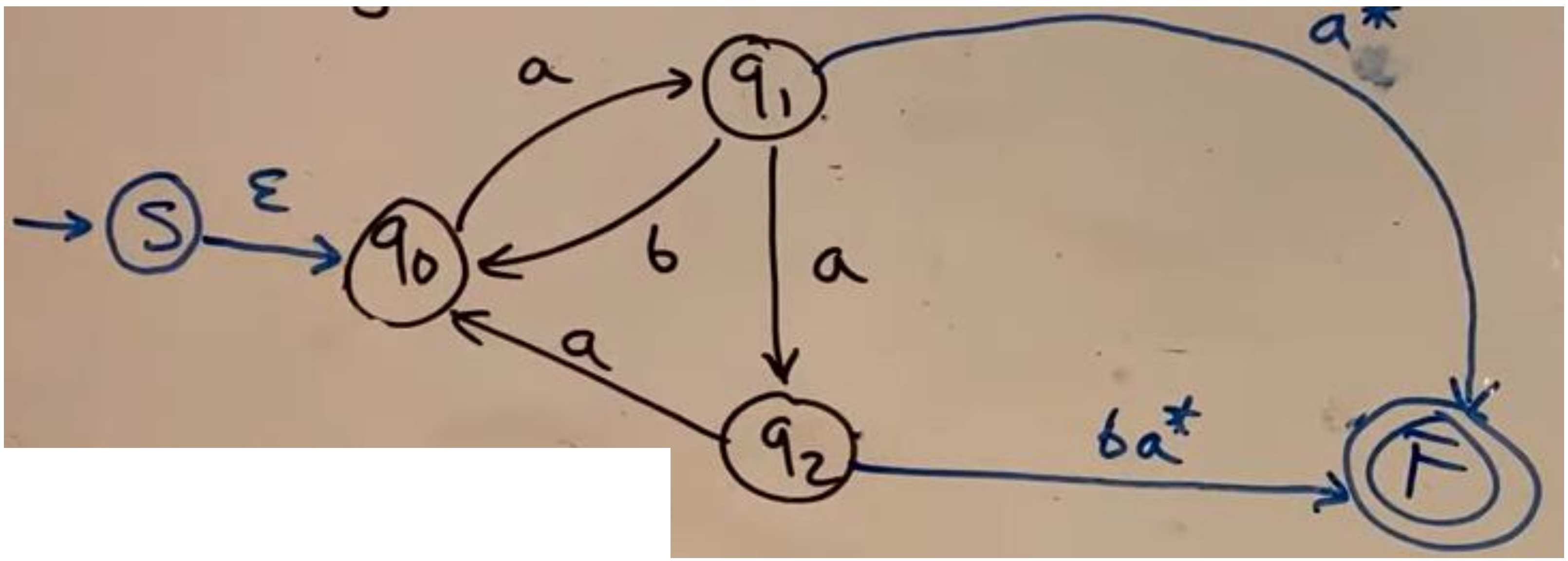
Q3 - In	Out
q1	F
q2	

GNFA –> Regular Expression

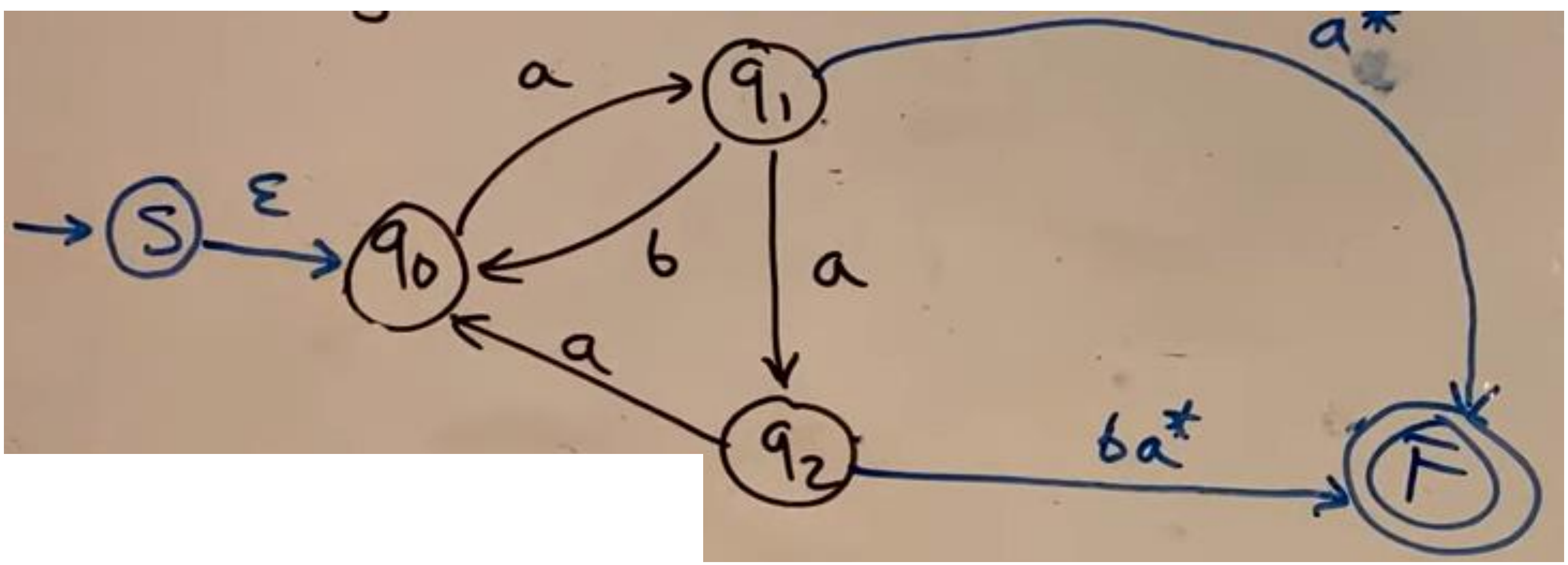


Q3 - In	Out
q1	F
q2	

GNFA –> Regular Expression

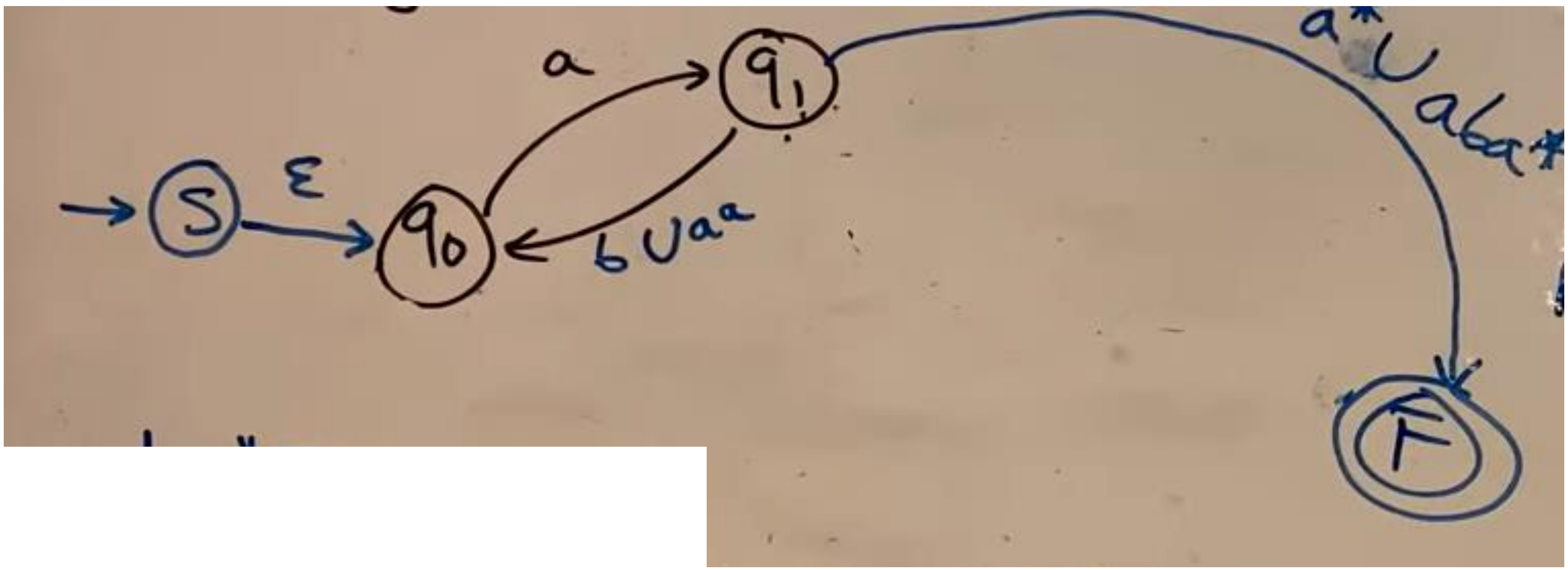


GNFA –> Regular Expression



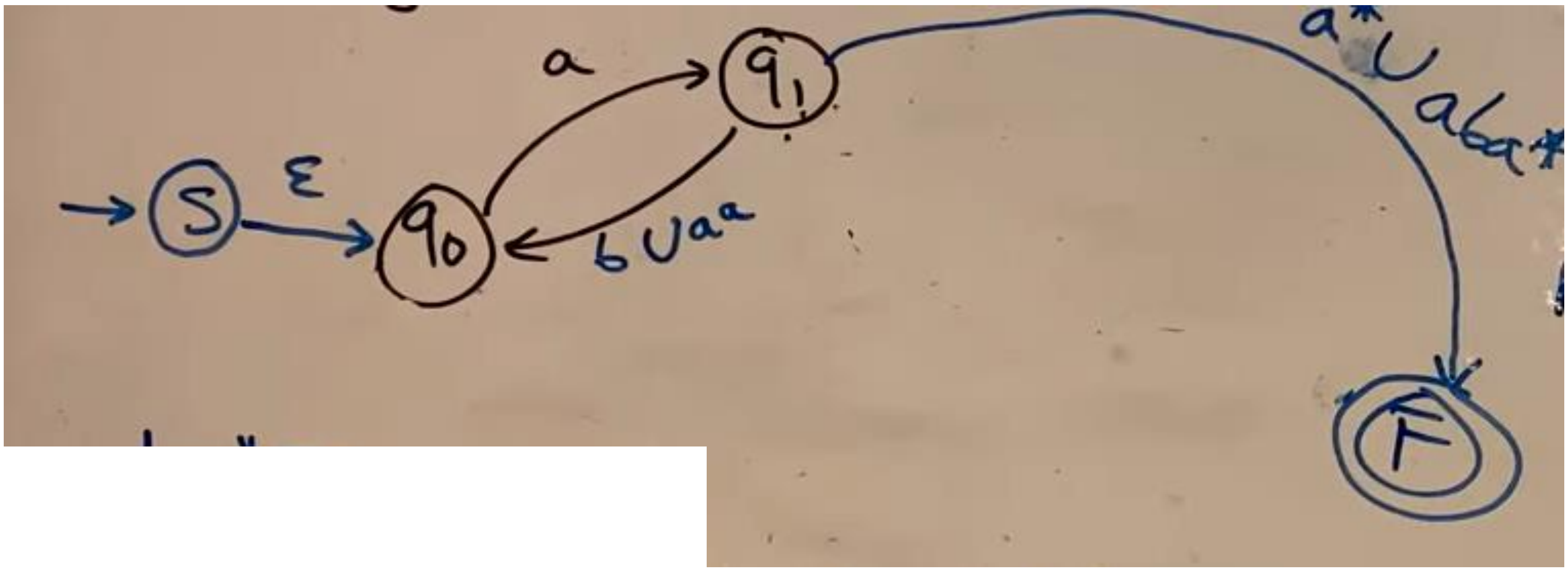
Q2 - In	Out
q1	q0
	F

GNFA –> Regular Expression



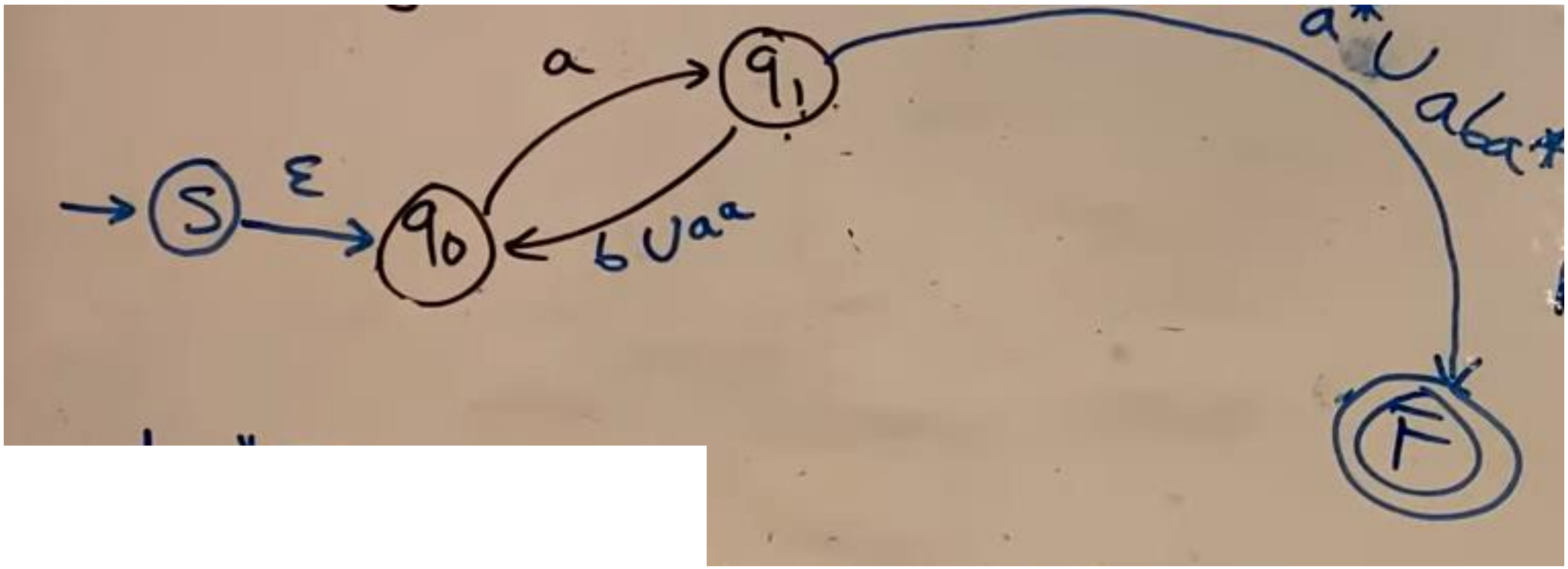
Q2 - In	Out
q1	q0
	F

GNFA –> Regular Expression



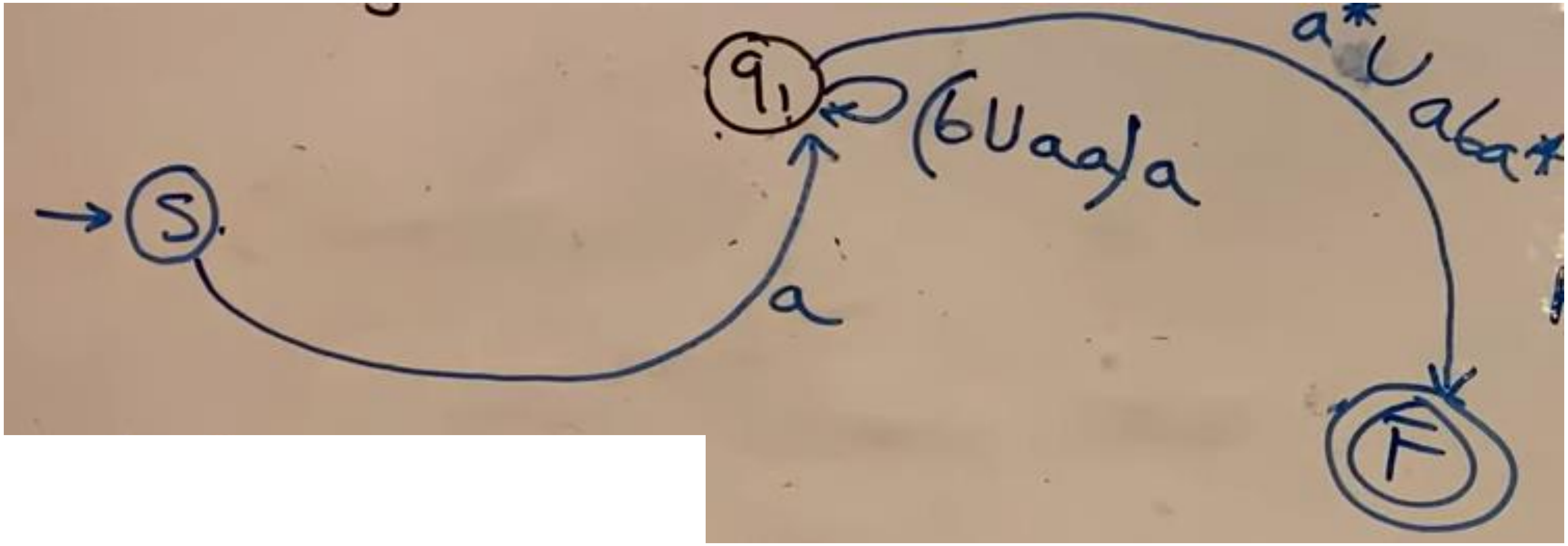
Q0 - In	Out
S	q1
q1	

GNFA –> Regular Expression



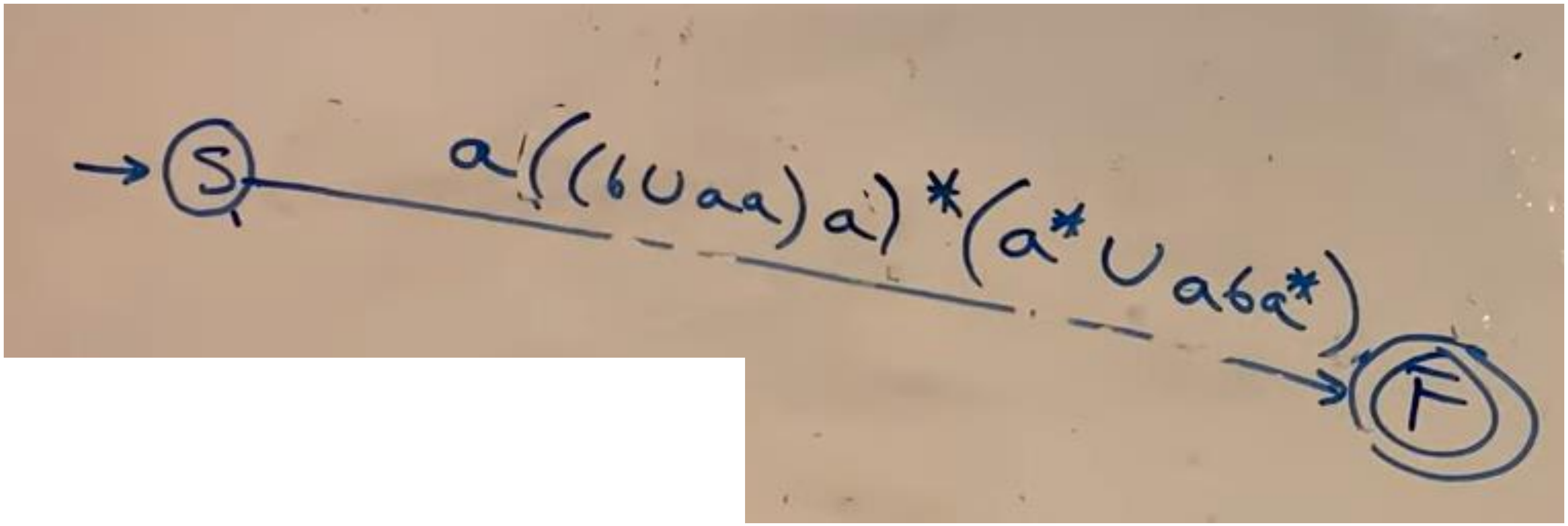
Q0 - In	Out
S	q1
q1	

GNFA –> Regular Expression



Q0 - In	Out
S	q1
q1	

GNFA –> Regular Expression

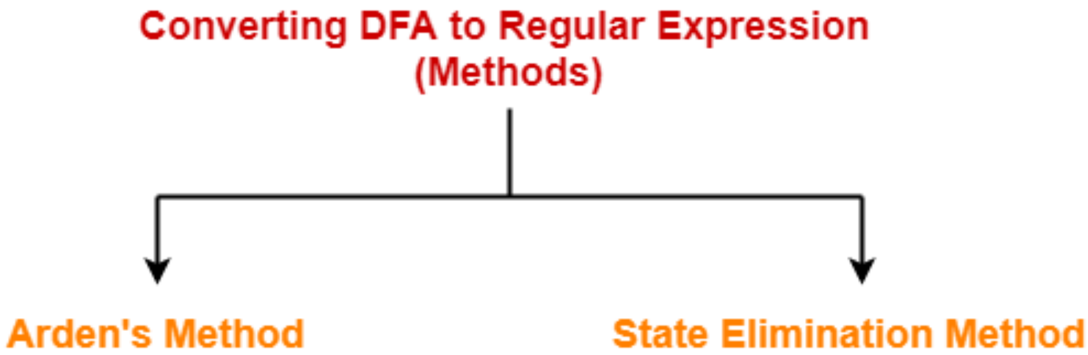


Q1 - In	Out
S	F

The regex is:

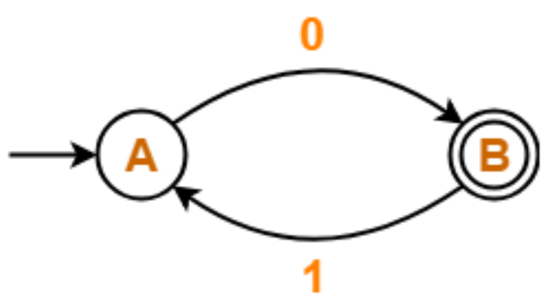
$a((b \cup aa)a)^*(a^* \cup aba^*)$

GNFA –> Regular Expression

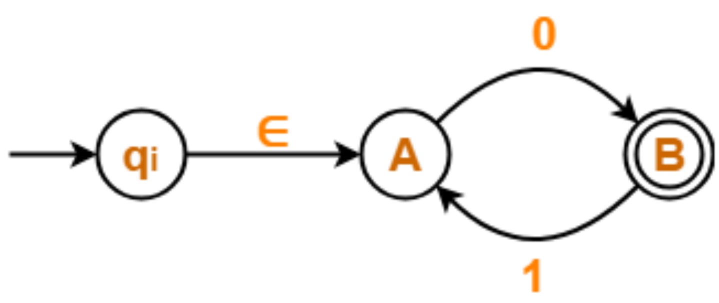


GNFA –> Regular Expression

Example 1

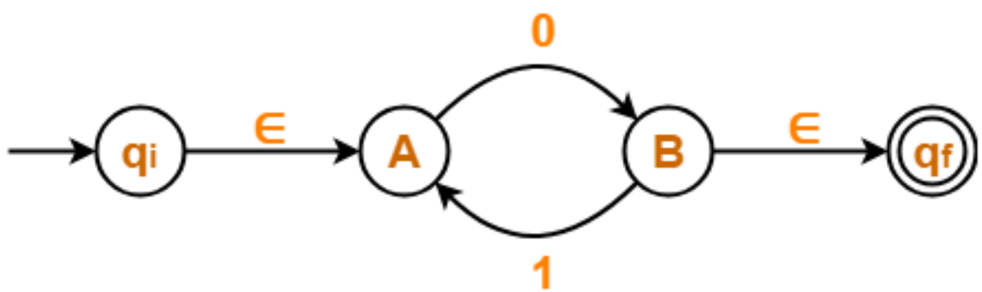


GNFA –> Regular Expression



GNFA –> Regular Expression

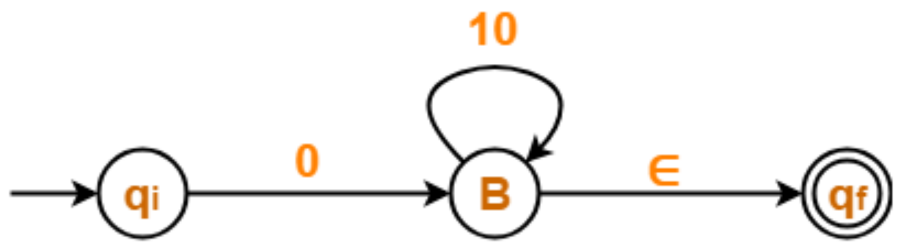
Now, we start eliminating the intermediate states with A



A - In	Out
qi	B
B	

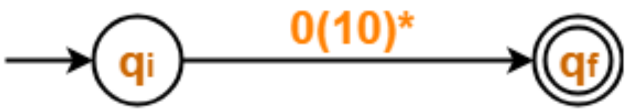
GNFA –> Regular Expression

Now, we start eliminating the intermediate states with B



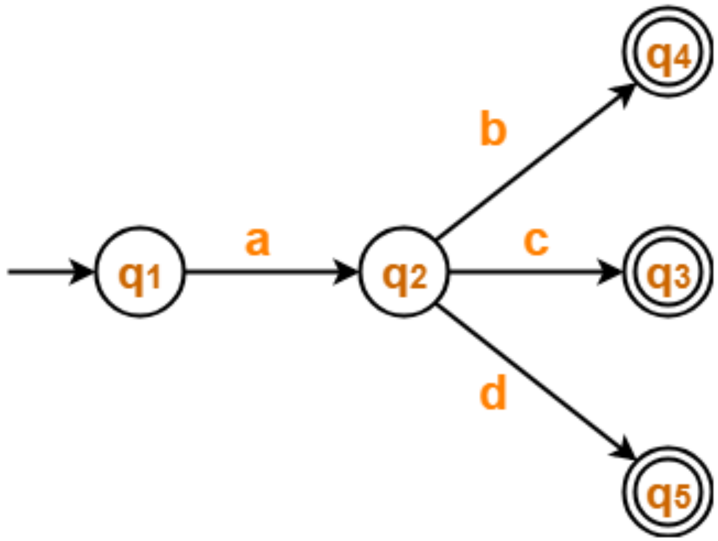
B - In	Out
qi	qf

GNFA –> Regular Expression

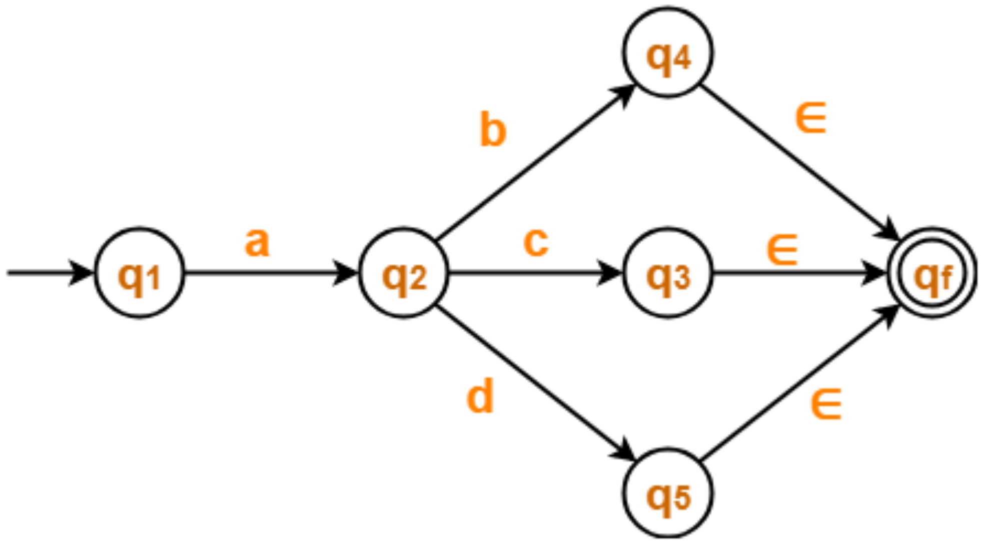


GNFA –> Regular Expression

Example 2



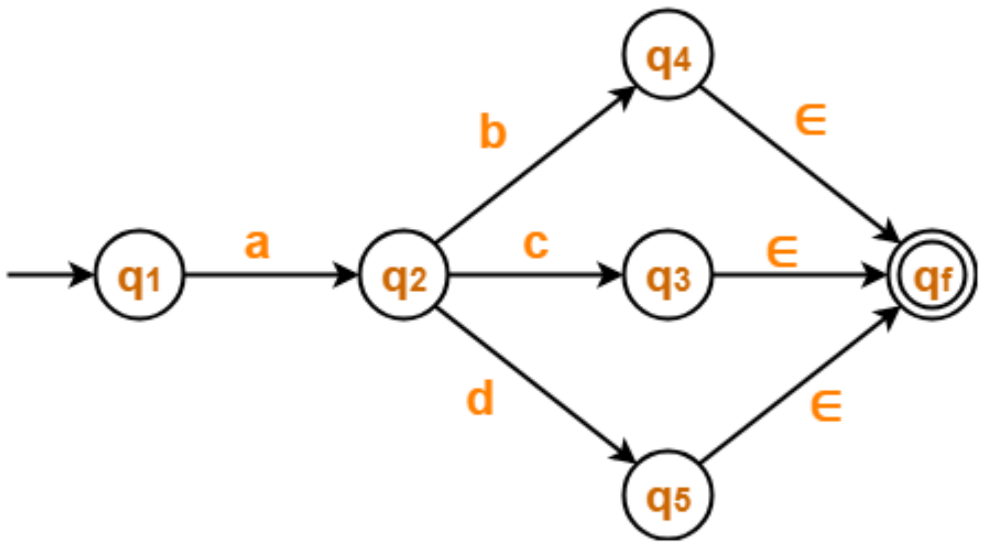
GNFA –> Regular Expression



There exist multiple final states.
So, we convert them into a single final state.

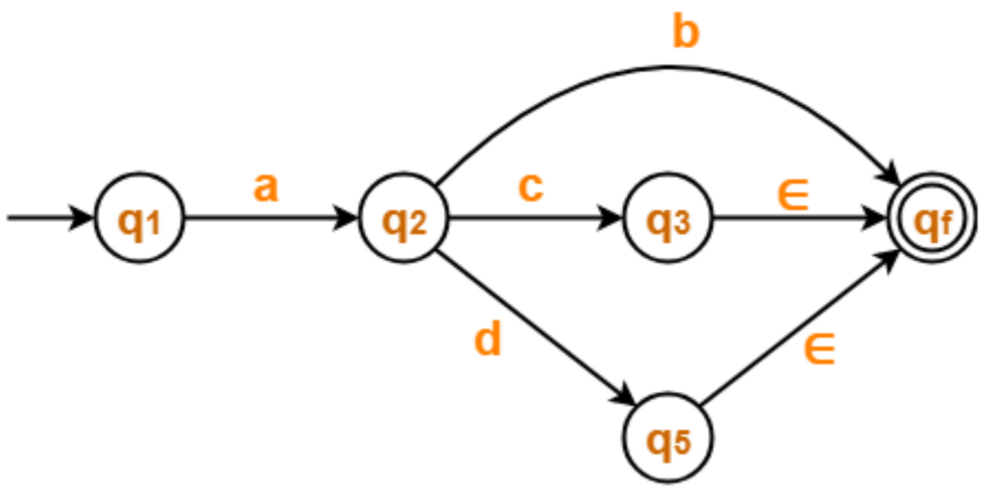
GNFA –> Regular Expression

Now, we start eliminating the intermediate states with q4



q4 - In	Out
q2	qf

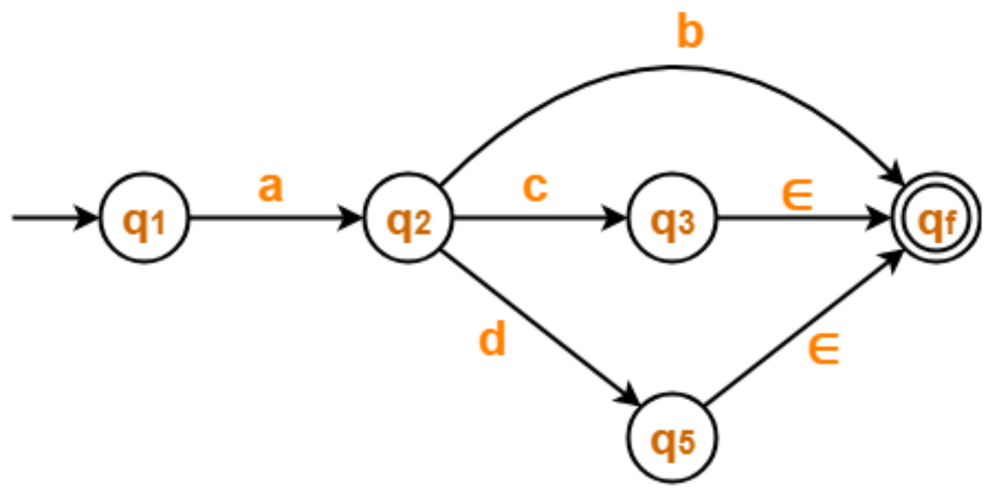
GNFA –> Regular Expression



q4 - In	Out
q2	qf

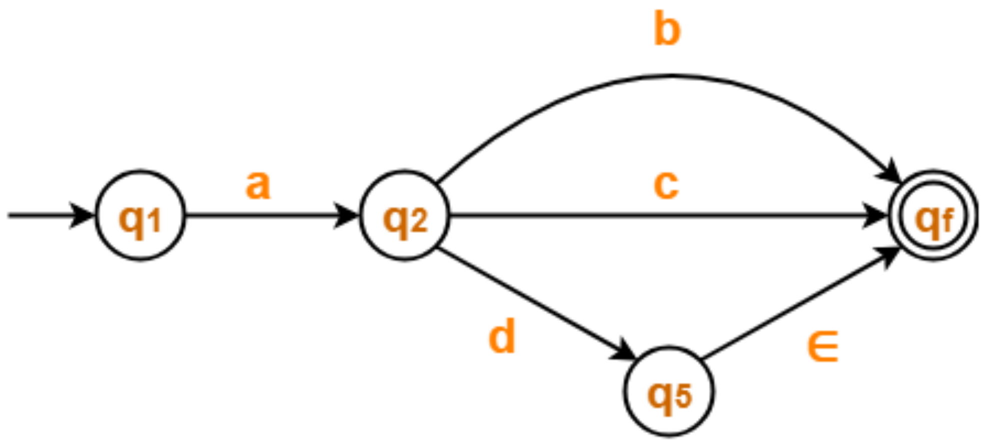
GNFA –> Regular Expression

Now, we start eliminating the intermediate states with q3



q3 - In	Out
q2	qf

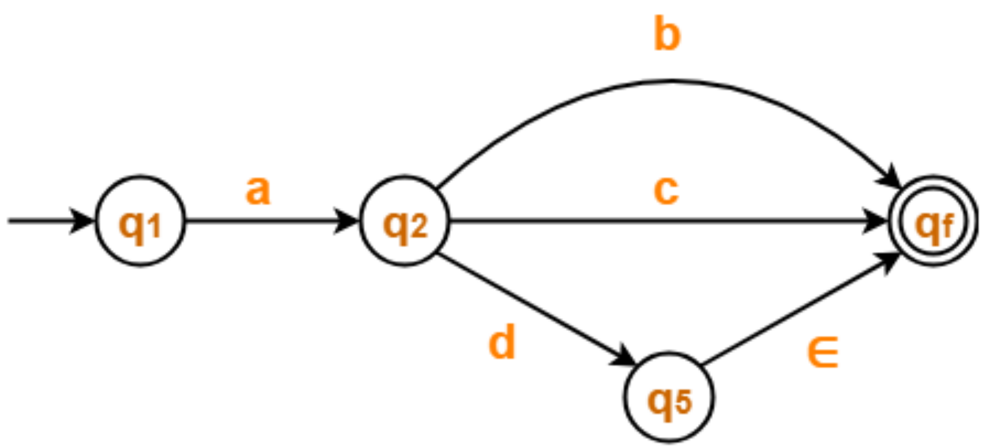
GNFA –> Regular Expression



Q3 - In	Out
q2	qf

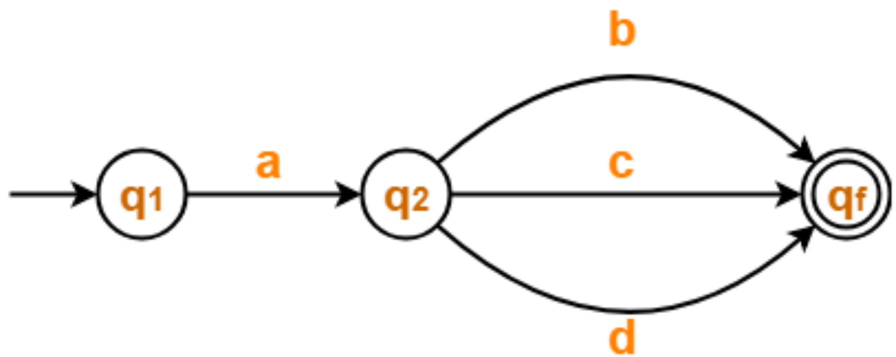
GNFA –> Regular Expression

Now, we start eliminating the intermediate states with q5



q5 - In	Out
q2	qf

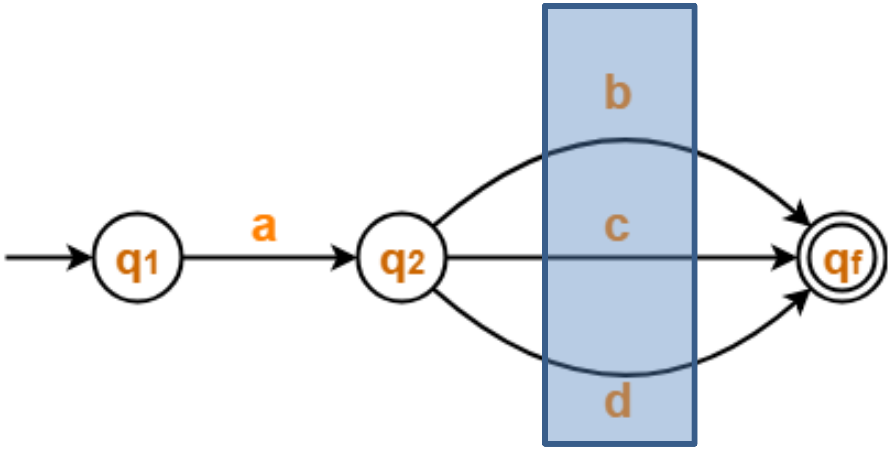
GNFA –> Regular Expression



Q5 - In	Out
q2	qf

GNFA –> Regular Expression

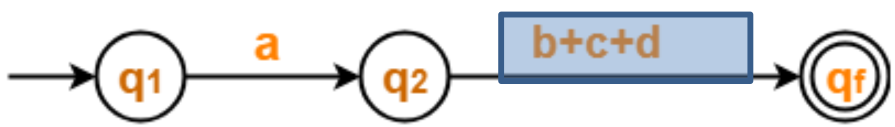
Now, we start eliminating the intermediate states with q2



q2 - In	Out
q1	qf

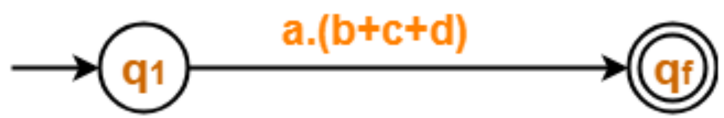
GNFA –> Regular Expression

Union and + are the same.



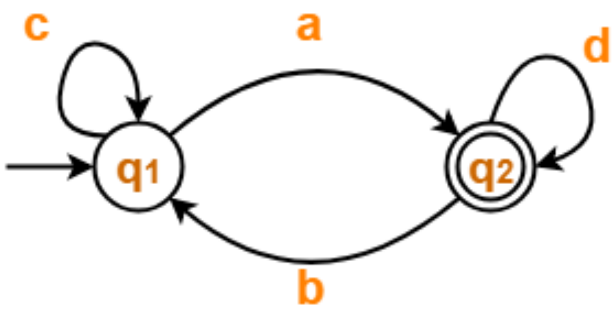
q2 - In	Out
q1	qf

GNFA –> Regular Expression

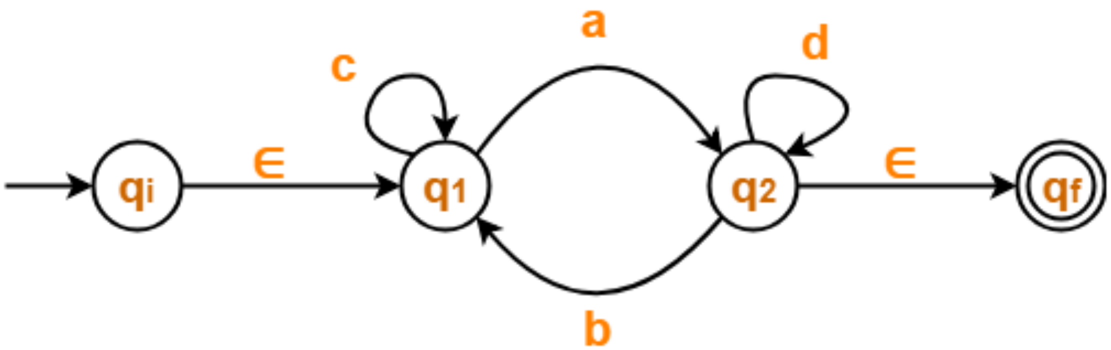


GNFA –> Regular Expression

Example 3



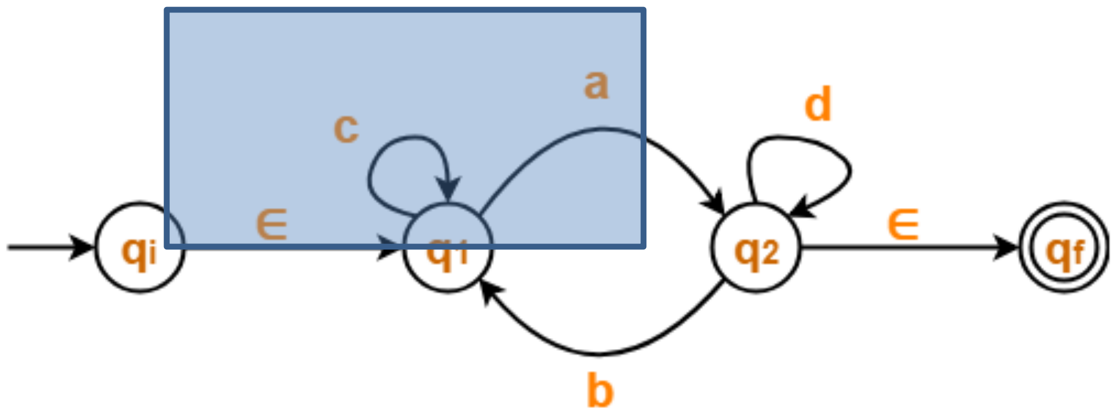
GNFA –> Regular Expression



Creating Incoming and Outgoing Edges

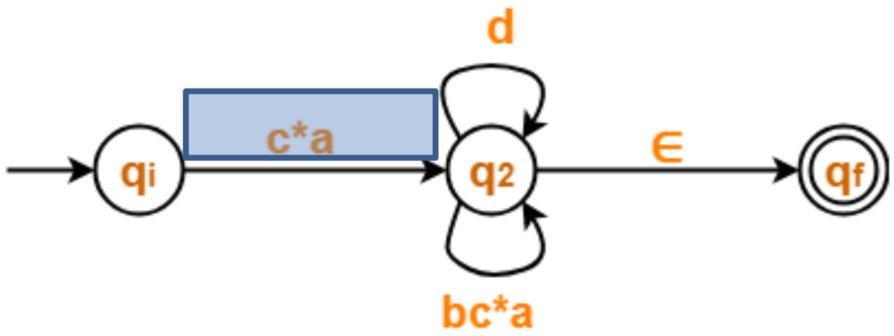
GNFA –> Regular Expression

Now, we start eliminating the intermediate states with q1



q1 - In	Out
qi	q2
q2	

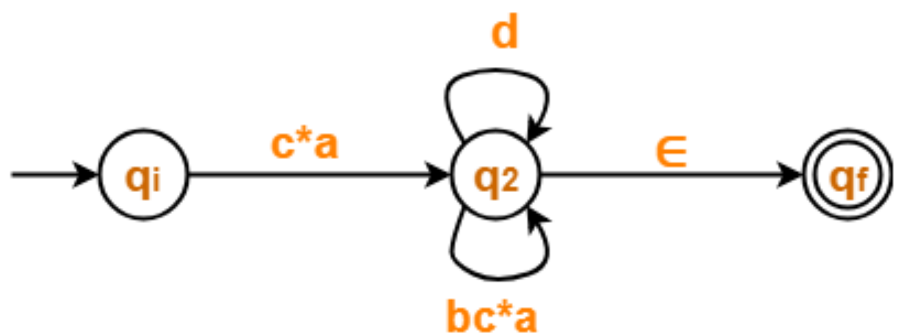
GNFA –> Regular Expression



Q1 - In	Out
qi	q2
q1	

GNFA –> Regular Expression

Now, we start eliminating the intermediate states with q2



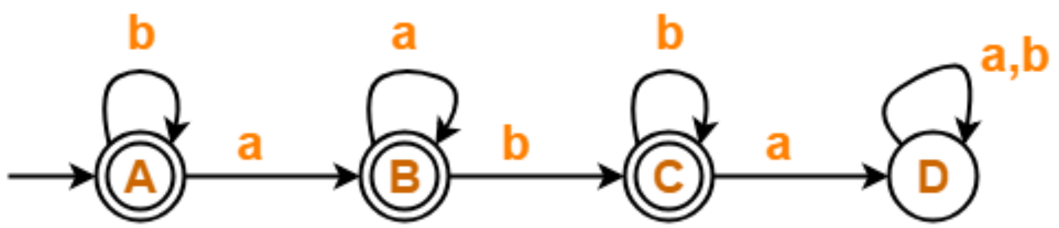
Q2 - In	Out
qi	qf
q2	

GNFA –> Regular Expression

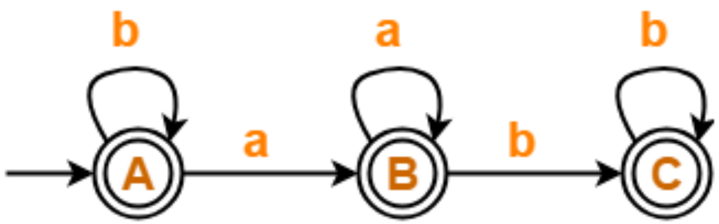
q2 - In	Out
qi	qf
q2	

GNFA –> Regular Expression

Example 4

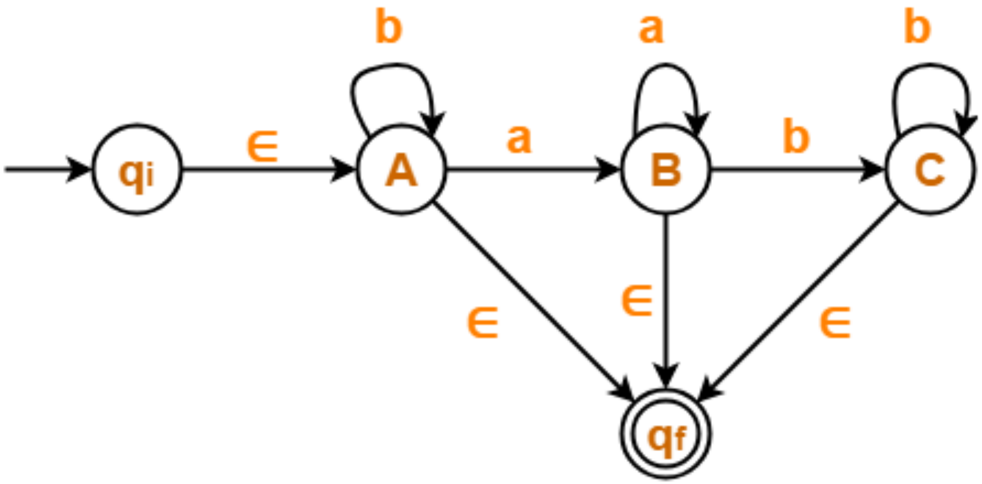


GNFA –> Regular Expression



State D is a dead state as it does not reach to any final state.
So, we eliminate state D and its associated edges.

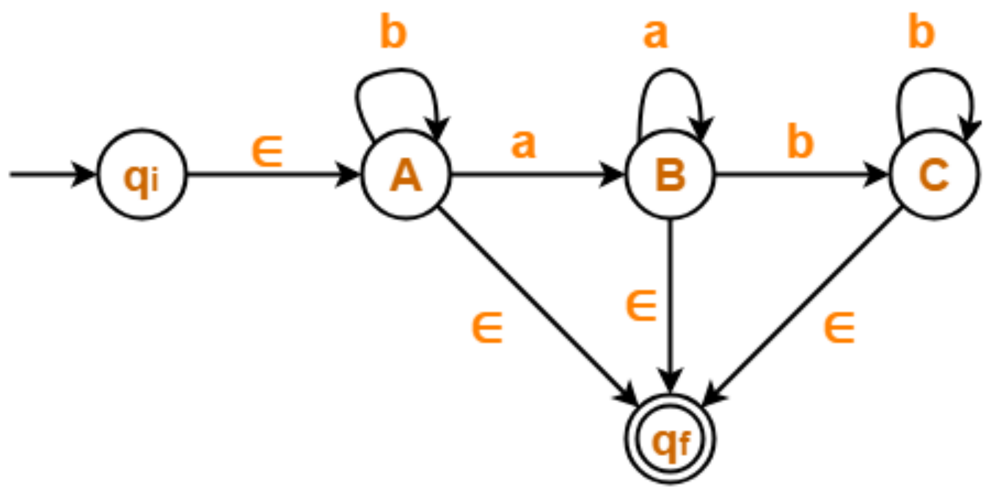
GNFA –> Regular Expression



There exist multiple final states.
So, we convert them into a single final state.

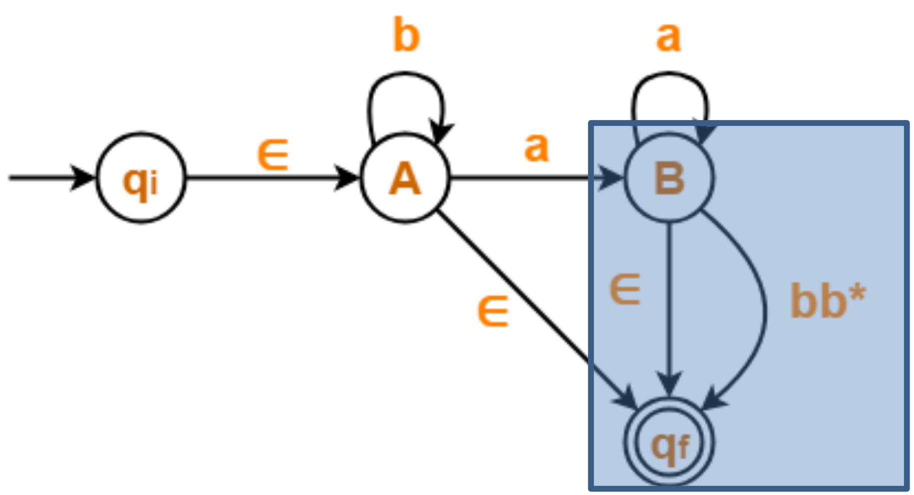
GNFA –> Regular Expression

Now, we start eliminating the intermediate states with C.



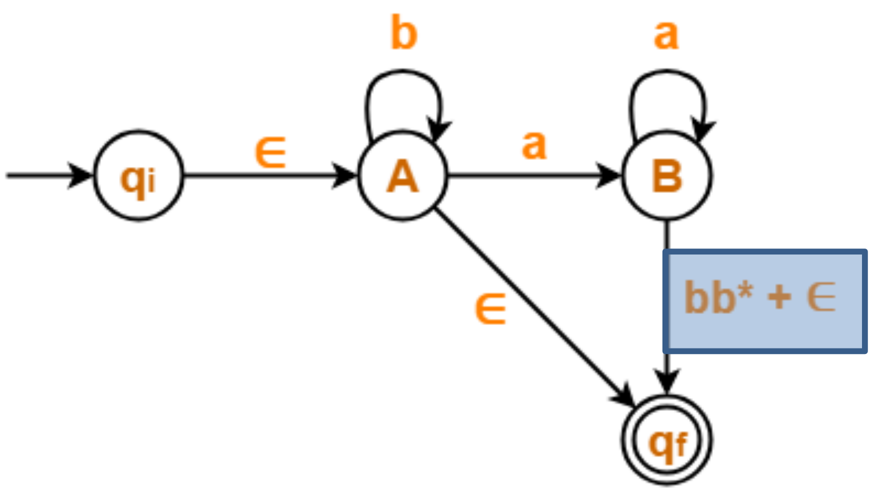
C - In	Out
B	qf
C	

GNFA –> Regular Expression



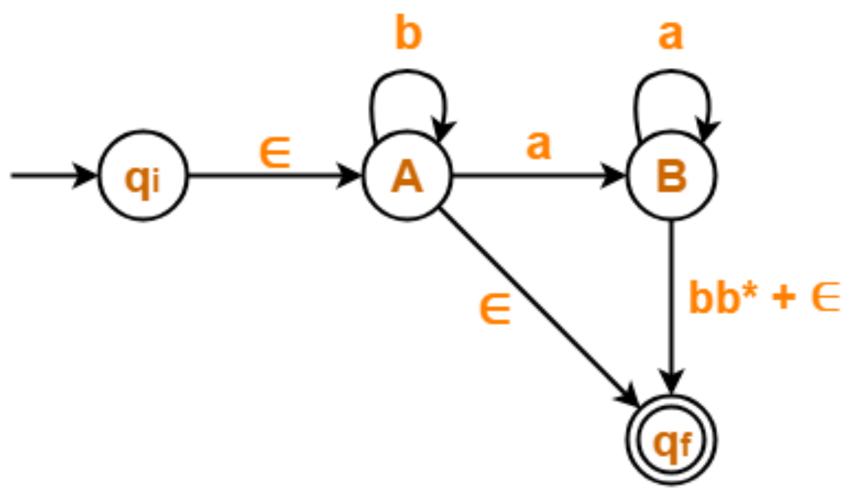
Now, we can combine the two parallels.

GNFA –> Regular Expression



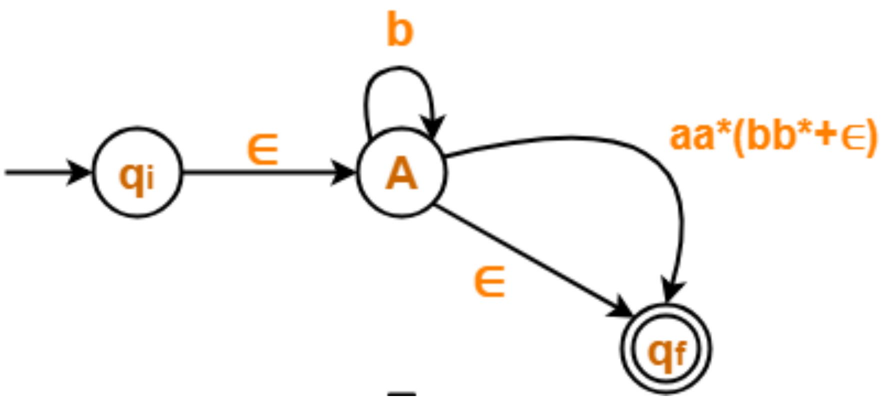
Now, we can combine the two with union. + and union denote same operation

GNFA –> Regular Expression

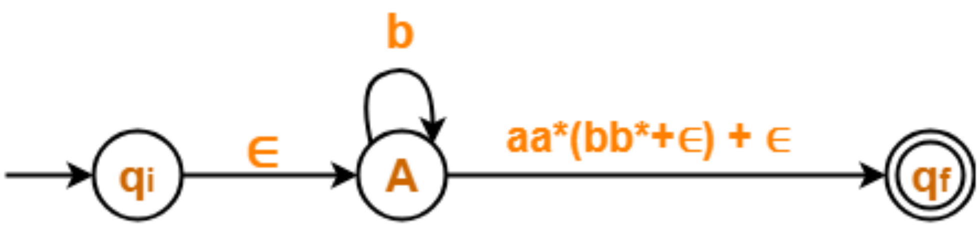


B - In	Out
A	qf
B	

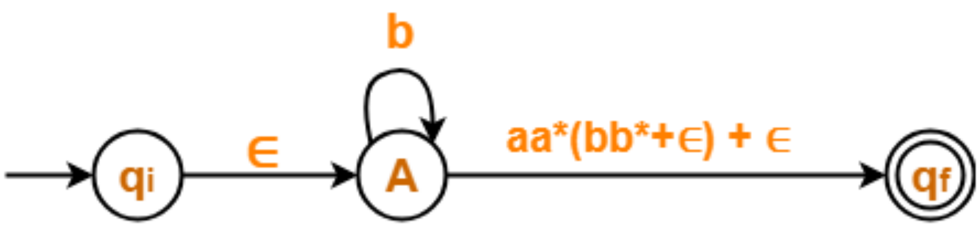
GNFA –> Regular Expression



GNFA –> Regular Expression

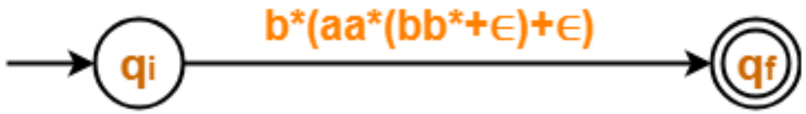


GNFA –> Regular Expression



A - In	Out
qi	qf

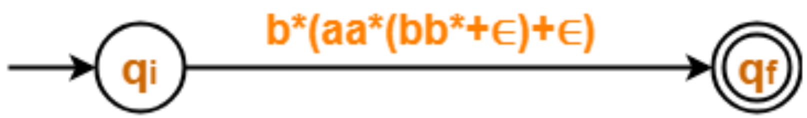
GNFA –> Regular Expression



Regular Expression = b*(aa*(bb*+∈)+∈)

GNFA –> Regular Expression

Expression can be simplified

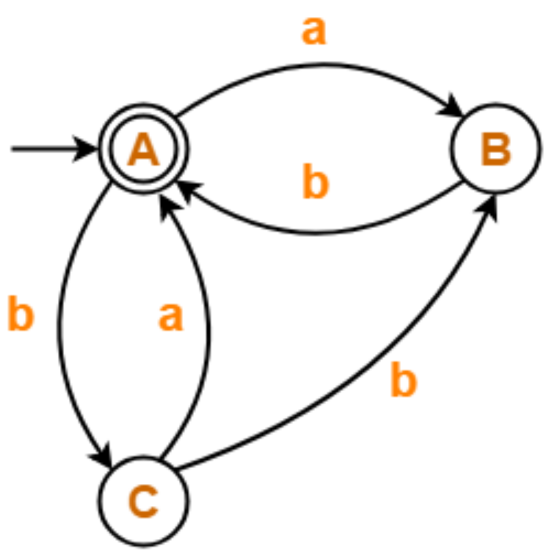


$bb^* + \in = b^*$

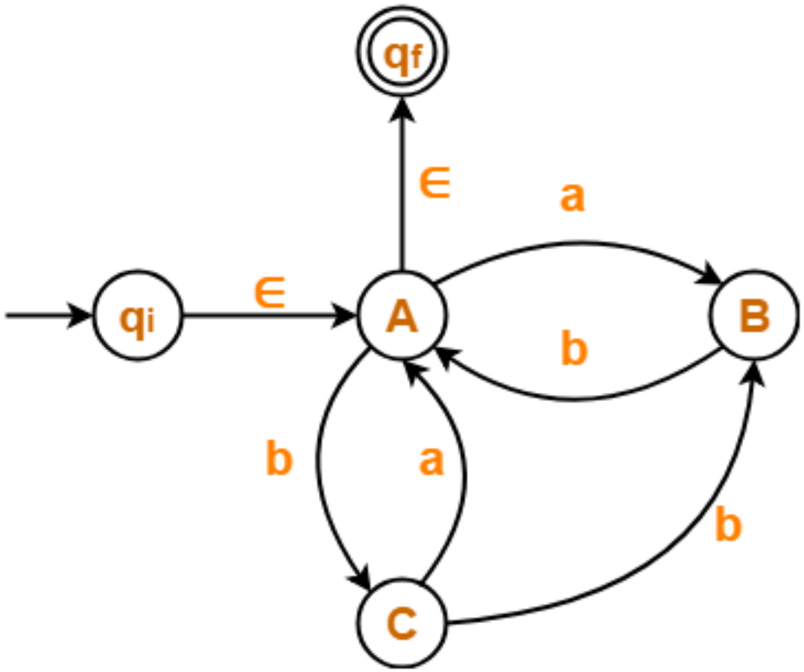
Regular Expression = b*(aa*b*+∈)

GNFA –> Regular Expression

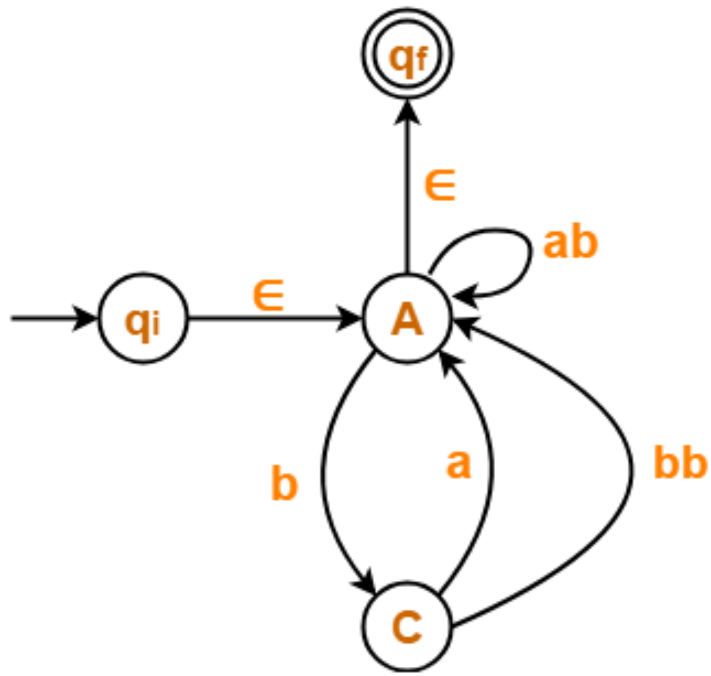
Example 5



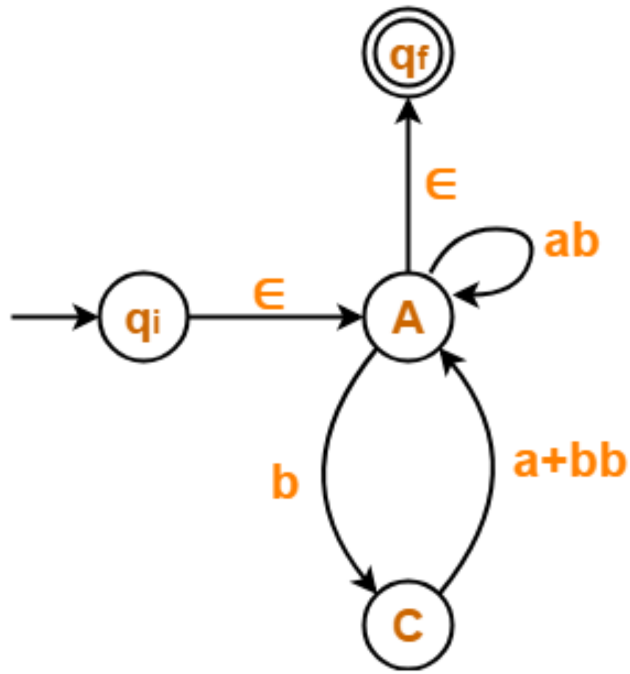
GNFA –> Regular Expression



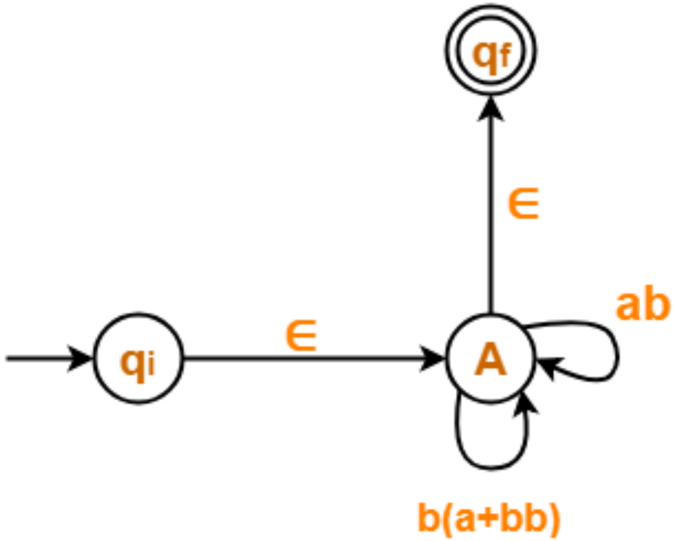
GNFA –> Regular Expression



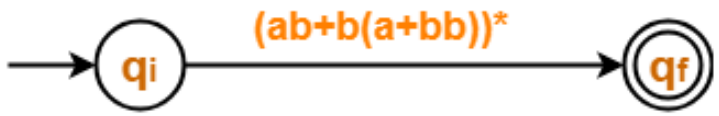
GNFA –> Regular Expression



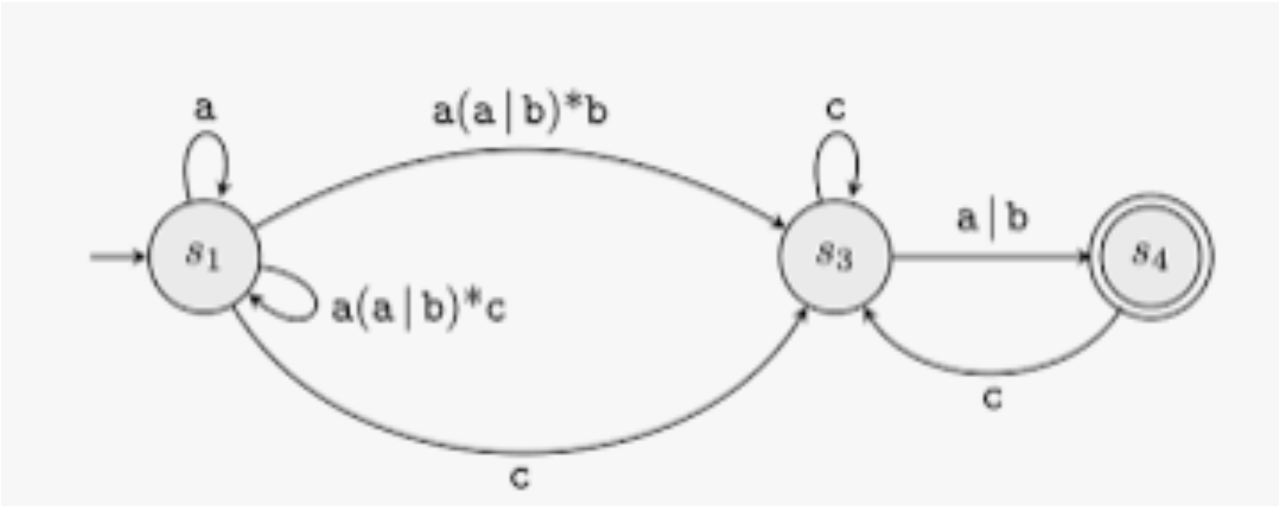
GNFA –> Regular Expression



GNFA –> Regular Expression

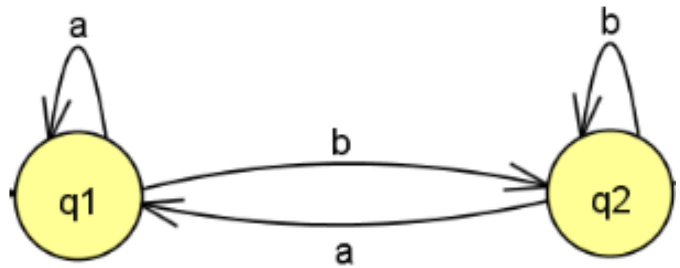


GNFA –> Regular Expression

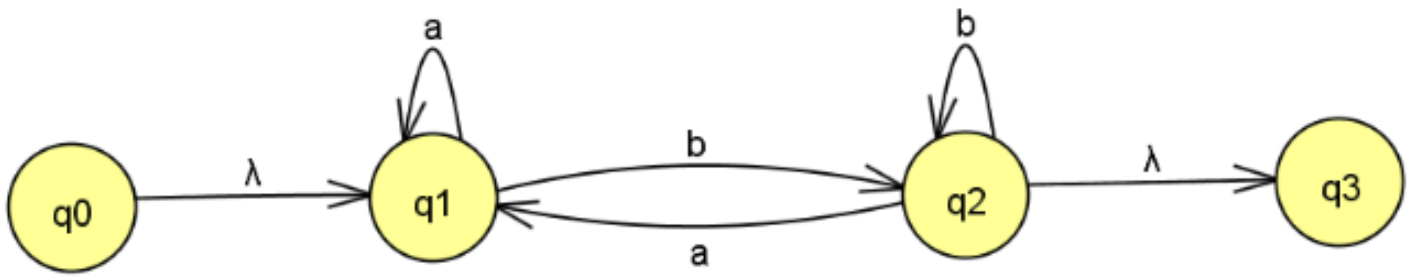


Example of another or(|)/union/+ notation

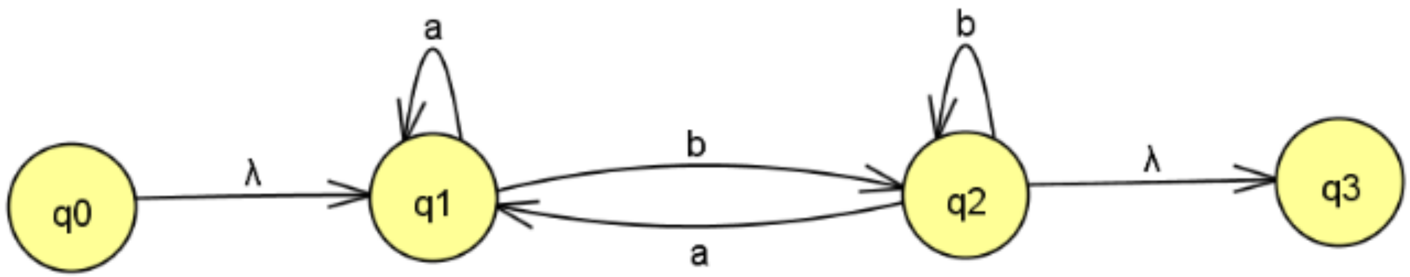
GNFA –> Regular Expression



GNFA –> Regular Expression



GNFA –> Regular Expression



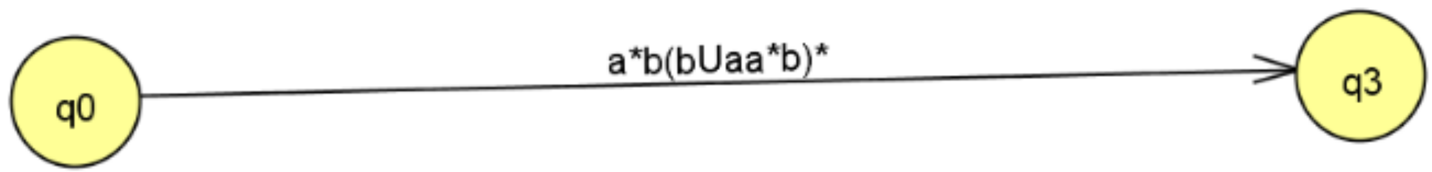
q1 - In	Out
q0	q2
q2	
q1	

GNFA –> Regular Expression

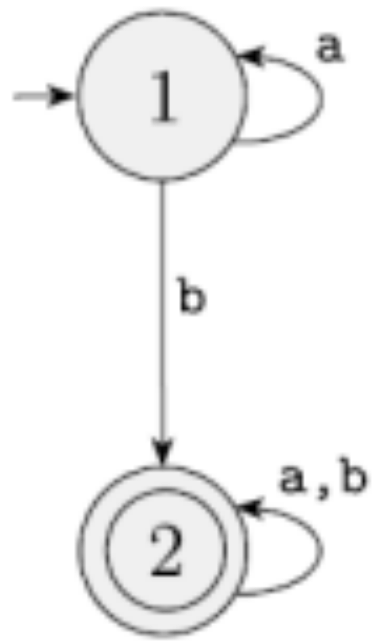
GNFA –> Regular Expression

Q2 - In	Out
q0	q3
q2	

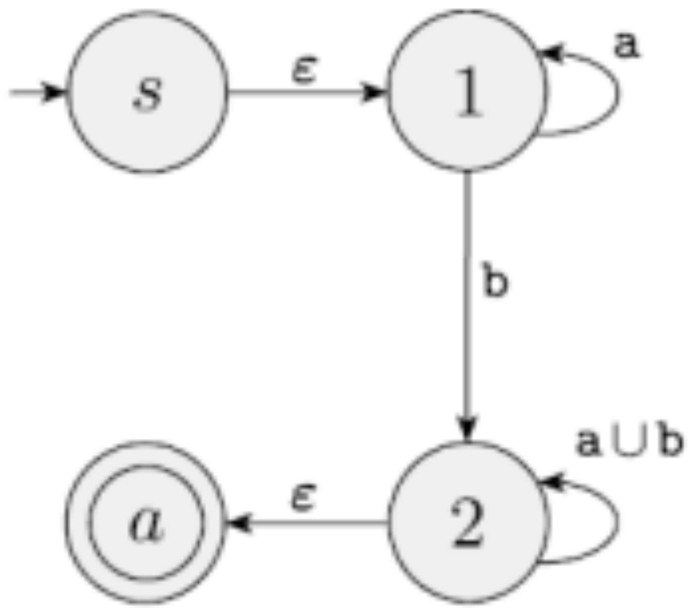
GNFA –> Regular Expression



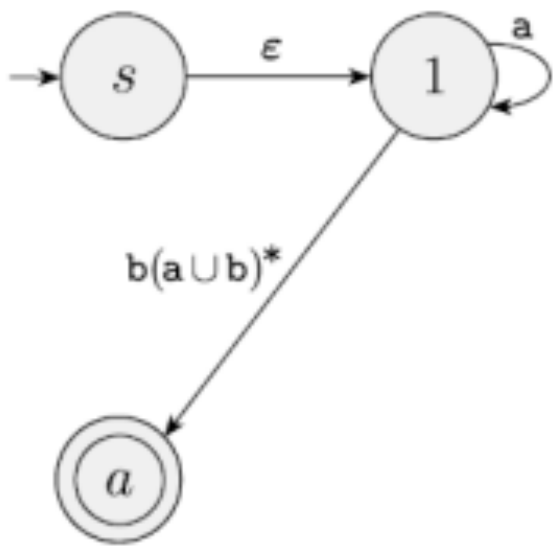
GNFA –> Regular Expression



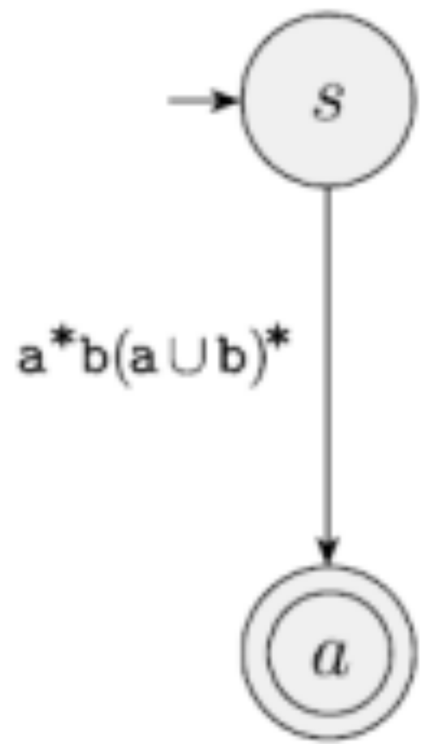
GNFA –> Regular Expression



GNFA –> Regular Expression



GNFA –> Regular Expression



GNFA –> Regular Expression