

02 | 07 | 2025

What did we learn so far?

DFA - Deterministic finite Automata

- Limited memory
- Deterministic behaviour

NFA - Non deterministic finite Automata

- Limited memory
- Non deterministic behaviour.

- However  $NFA = DFA$
- Any language that is recognized by a DFA/NFA is called a Regular language

# Regular Expressions

- We can use regular operations to build up expressions that describes languages.

- regular operations

- Union ( $\cup$ )

- Concatenation ( $\circ$ )

- Kleene Star  $^*$   $(\{0\} \cup \{1\})^*$

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

→ This expression describes a language

It basically describes strings in the language where starting with either 0 or 1 followed by zero or more 0s

$$(0 \cup 1)^* = (\{0\} \cup \{1\})^* \xrightarrow{\text{star}}$$

↑ union      ↑ concatenation

If  $\Sigma$  is an alphabet, the regular expression  $\Sigma^*$  describes the language consisting of all length 1 strings over the alphabet  $\Sigma$ .

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

alphabet.

$\Sigma^*$  ← using  $\Sigma$  as a regular expression

$$\underline{\Sigma} \equiv (0 \cup 1)^* = \{0, 1\}^*$$

$\Sigma^*$  describes the language consisting of all strings over the alphabet.

$$\Sigma = \{0, 1, 2\}^*$$

$$\Sigma = (0 \cup 1 \cup 2)^* = \{\varepsilon, 0, 1, 2, 01, 000, \dots\}$$

In regular expressions Star operations are done first, followed by concatenation and finally union, unless parenthesis change the usual order.

1. Parenthesis
2. Star
3. Concatenation
4. Union

$$5 + \underline{3 \times 2} + 6$$

$$5 + 6 + 6$$

$$11 + 6$$

$$17$$

# ~~Def~~ Regular Expressions

$R$  is a regular expression,  
if  $R$  is:

1.  $a$ , for some  $a \in \Sigma$
  2.  $\epsilon \in \{\epsilon\}$
  3.  $\phi \in \{\}$
  4.  $R_1 \cup R_2$ , where  $R_1$  &  $R_2$  are regular expressions.
  5.  $R_1 \cap R_2$ , where  $R_1$  &  $R_2$  are regular expressions.
  6.  $R_1^*$ , where  $R_1$  is a regular expression.
- $\left. \begin{matrix} \{a\} \\ \{\epsilon\} \\ \{\} \end{matrix} \right\}$  these are not equal

$$\text{Ex: } R \cup \emptyset = R$$

$$\text{e.g. } R = \{\epsilon\}$$

$$L(R) = \{\epsilon\}$$

$$\underline{L(R \cup \emptyset)} = L(R) = \{\epsilon\}$$

$$R \cup \{\epsilon\} \neq R$$

$$\text{e.g. } R = \{\epsilon\}$$

$$L(R) = \{\epsilon\}$$

$$L(R \cup \{\epsilon\}) =$$

{ε, ϵ}

remember regular expression  
describes a language

$$0^*10^* = \{w \mid w \text{ contains a single}\}$$

simple exercise: let  $\Sigma = \{0, 1\}$   
what are the languages described  
by following regular expressions?

$$1. (01)^* = \{\underline{01}\}^*$$

$$4. (01)^* \cup \epsilon = \{\epsilon, \underline{01}^*\}$$

$$2. 1^*0\epsilon = 1^*$$

$$5. (0 \cup \epsilon)1^* = \underline{01}^* \cup \underline{1}^*$$

$$3. 0^*0\phi = \{0\}^*0\phi = \{\underline{0}\}^*0\{0\} = \phi$$

$01^* \neq (01)^*$

Regular Expressions = Regular Languages.

Any regular language (a language that is recognized by a DFA/NFA) can be expressed using a regular expression.

Theorem 1.54

A language is regular iff some regular expression describes it.

"If part"

Lemma 1.55 If a Language is described by a regular expression, then it is regular.

Proof: Let  $R$  be a regular expression then we will convert it into an equivalent NFA.

$$1. R = a, a \in \Sigma, L(R) = \{a\}$$



$$2. R = \epsilon, L(R) = \{\epsilon\}$$

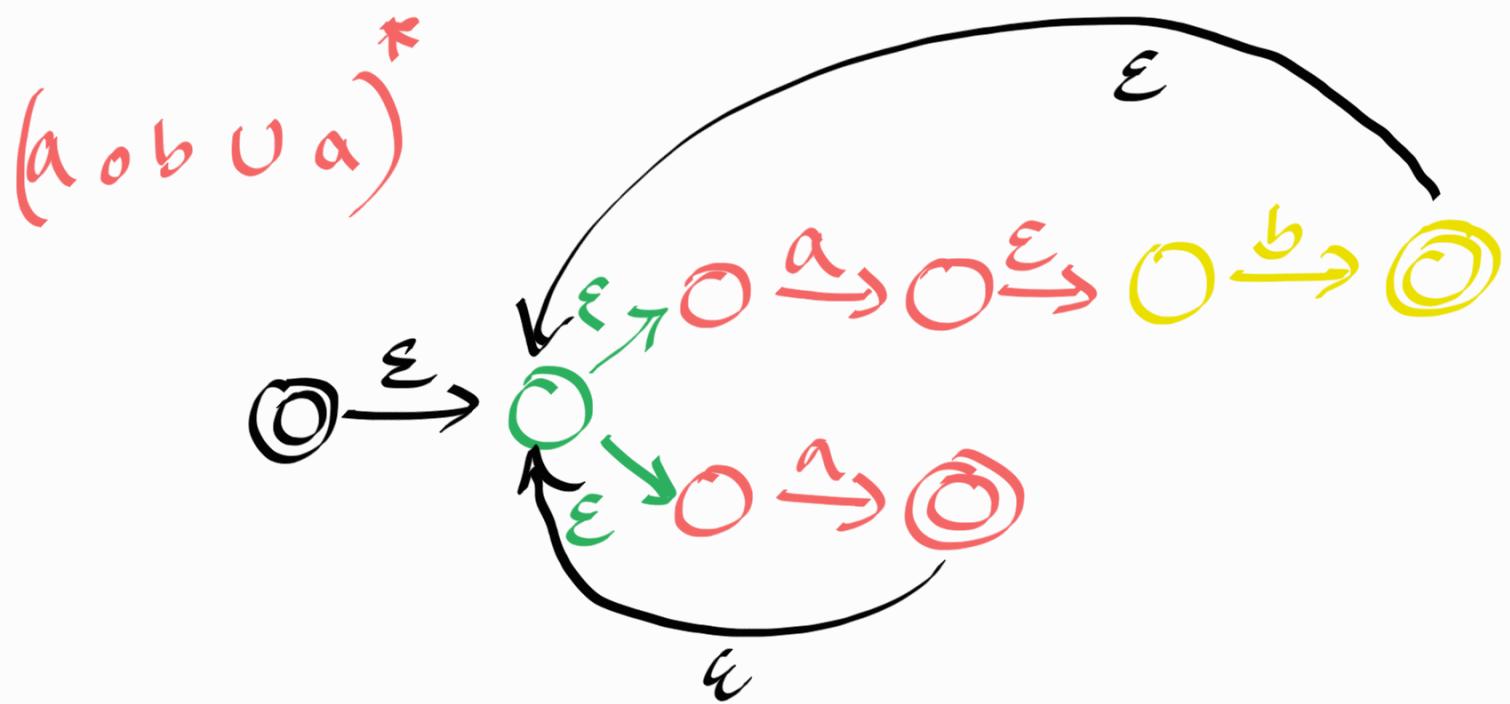
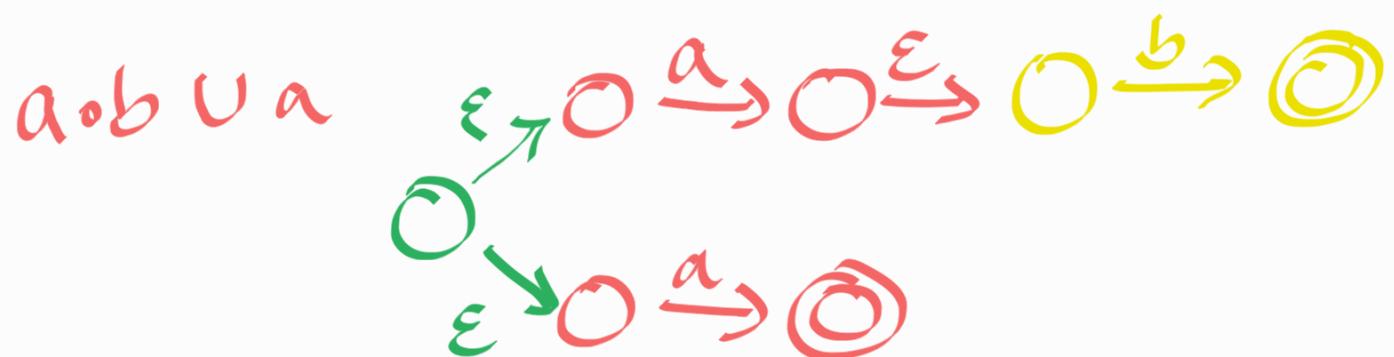


$$3. R = \phi, L(R) = \{\}$$



4.  $R = R_1 \cup R_2 \quad \} \quad \text{use the already created}$   
 5.  $R = R_1 \cdot R_2 \quad \} \quad \text{NFA's of } R_1, R_2 \text{ then}$   
 6.  $R = R_1^*$  create larger NFA's.  
follow Thm 1.45, 1.47,  
1.49

Ex:  $(a \circ b \cup a)^*$



"Only if" part

Lemma 1.60

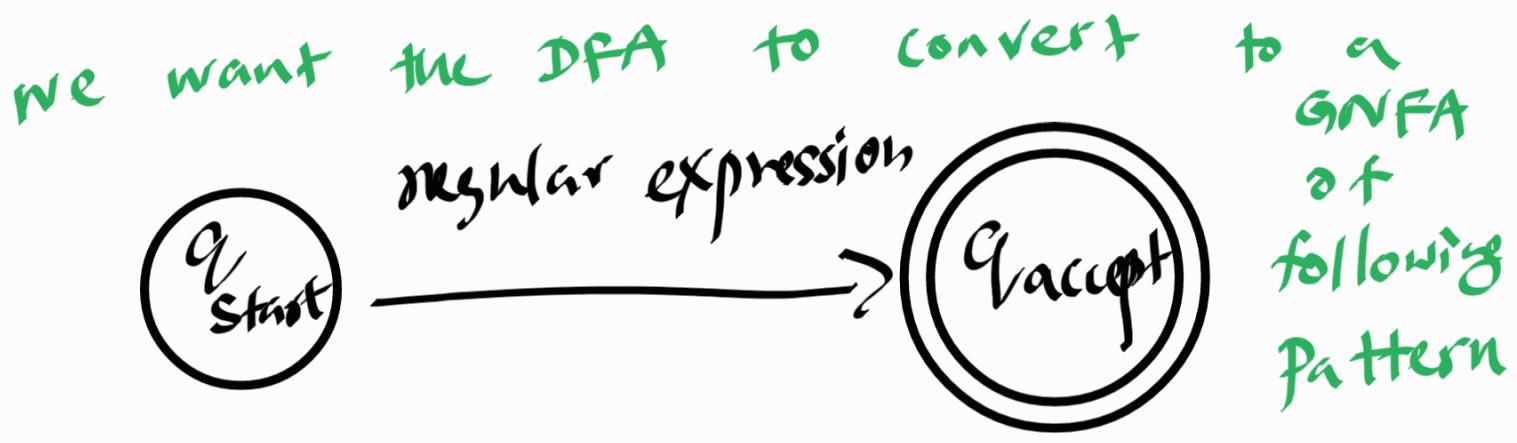
If a language is regular,  
then it is described by a  
regular expression.

Idea!

step 1: DFA  $\rightarrow$  GNFA

step 2: GNFA  $\rightarrow$  regular expression

Generalized NFA are simply  
Nondeterministic finite automata wherein  
the transition arrows may have any  
regular expressions as labels, instead  
of only one symbol or  $\epsilon$ .

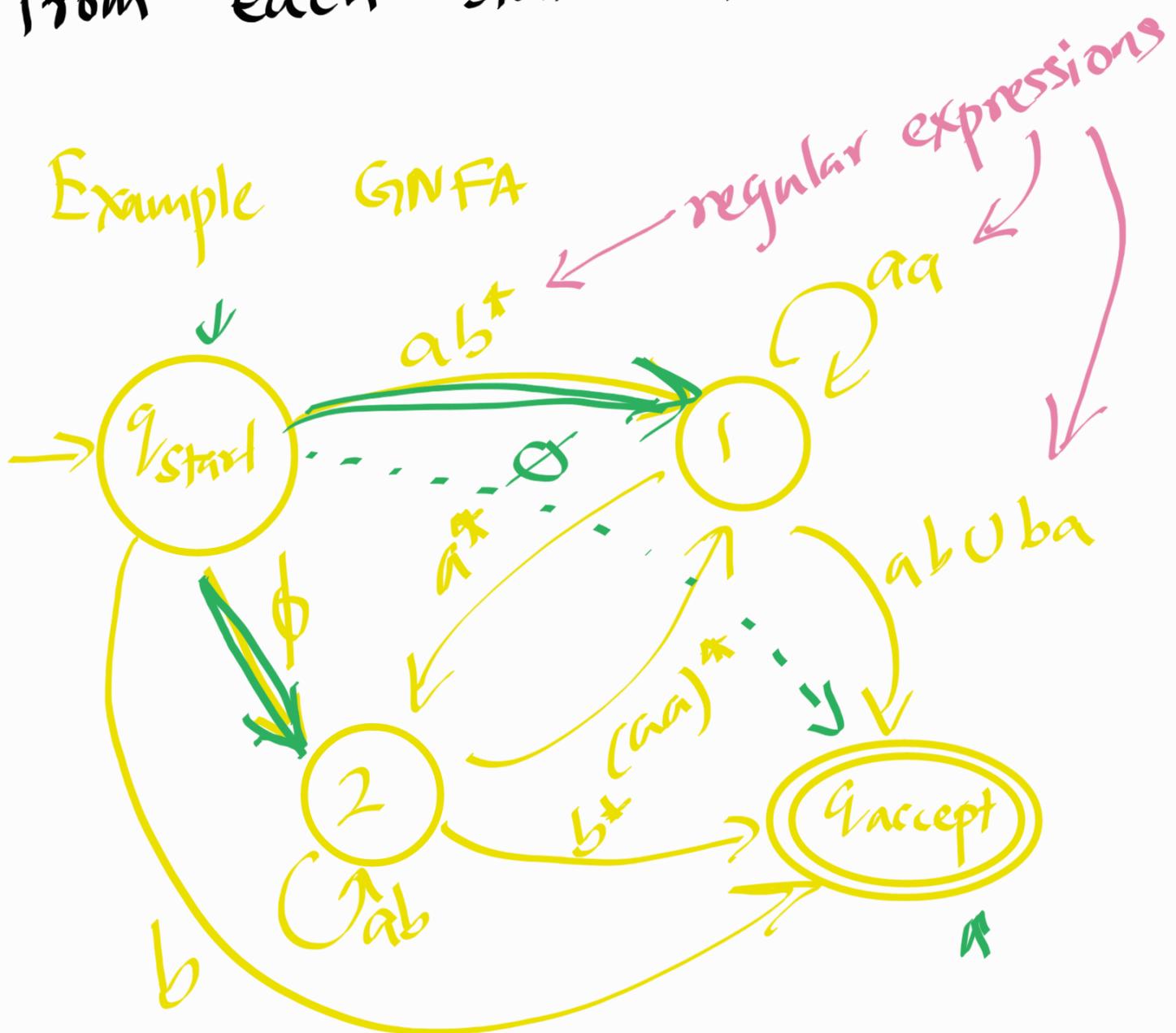


GNFA follows a special set of conditions.

following conditions needs to be met if we want to call a machine GNFA.

- start state has transition arrows going to every other state but no arrows coming in from any 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.  
Accept State cannot be the start state.

- Except for start and accept state, one arrow goes from every state to every other state and also from each state to itself.



Note about transitions:

$\emptyset$  transitions mean that there is no transition

Let's look at how we can transform a DFA to a GNFA

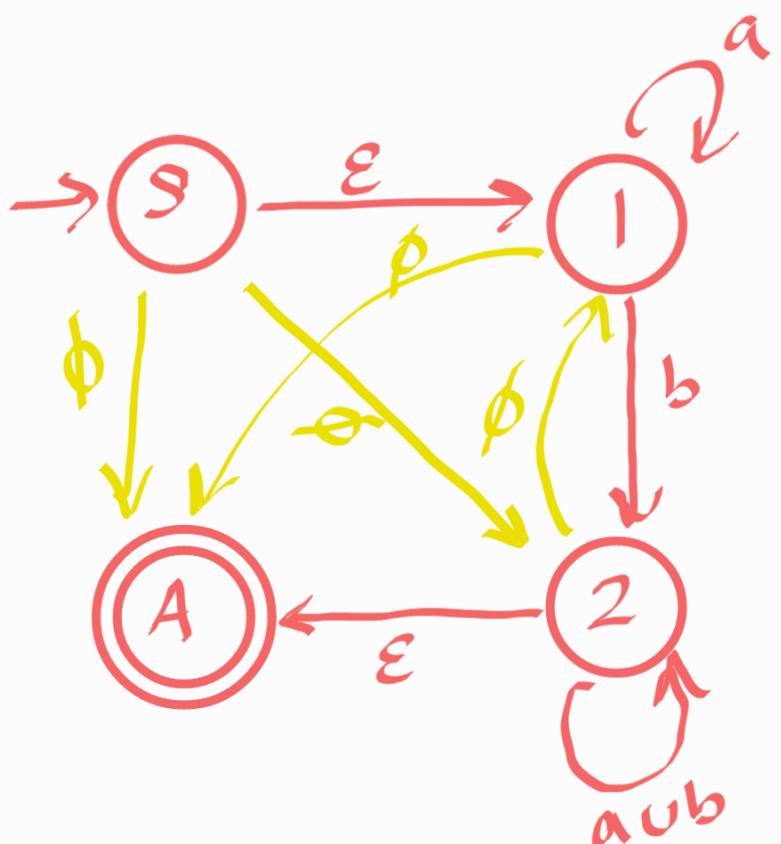
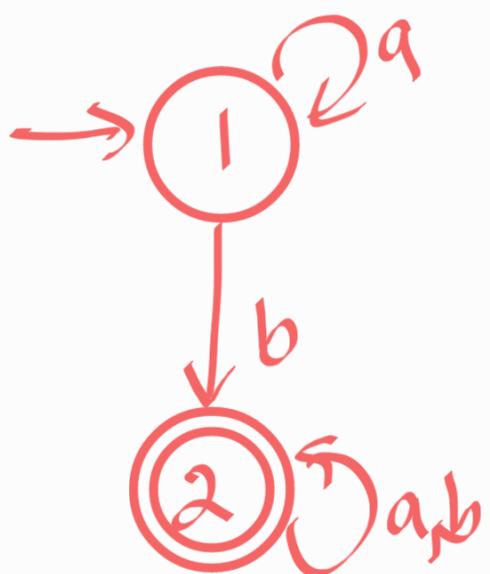
1. Add a new start state and add a  $\epsilon$  transition to the old start state
2. Add a new accept state and put  $\epsilon$ -transitions to new accept state from all old accept states.
3. If any arrows have multiple labels (or if there are multiple arrows going between two states in the same direction), we replace each with a single arrow whose label is the union of the previous labels.



Converting a DFA to a GNFA is easy.

Let's look at a simple DFA and convert it to a

GNFA



Now we need to convert the  
GNFA into a regular expression.

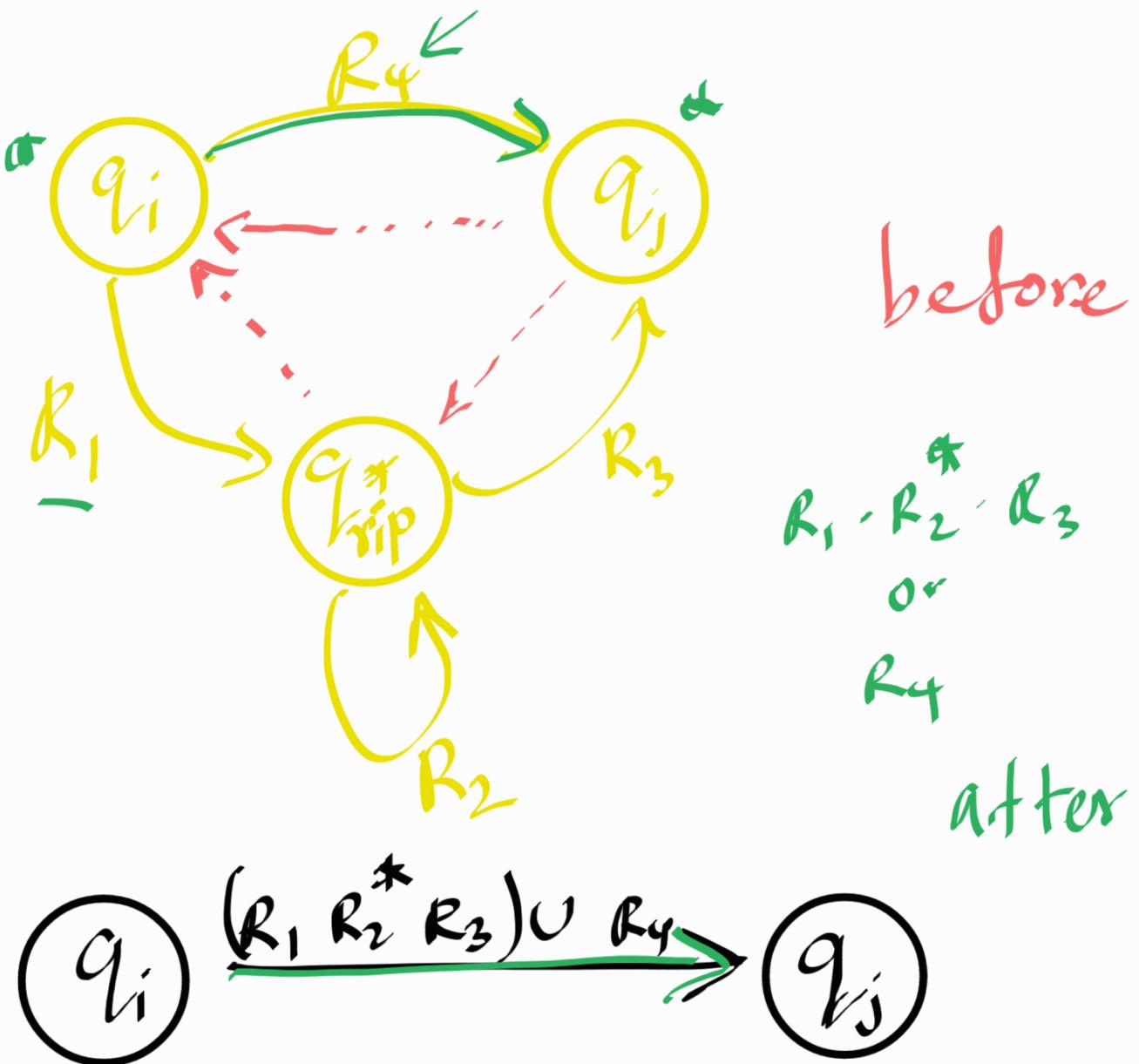
We systematically reduce the number  
of states of the given GNFA  
one by one until it has 2  
states.

- Starting state.
- Accept state.
- arrow between these 2 states  
with the final regular expression.

\* The key idea about reducing  
the number of states is called  
ripping.

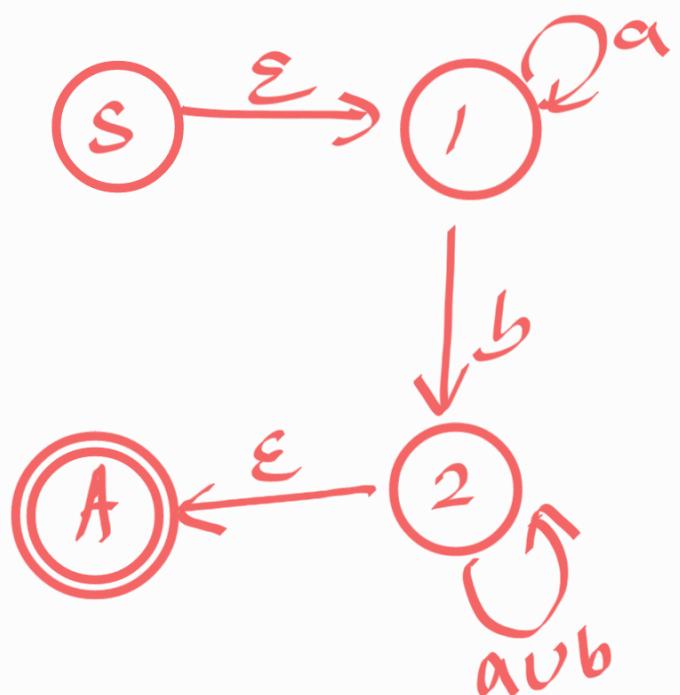
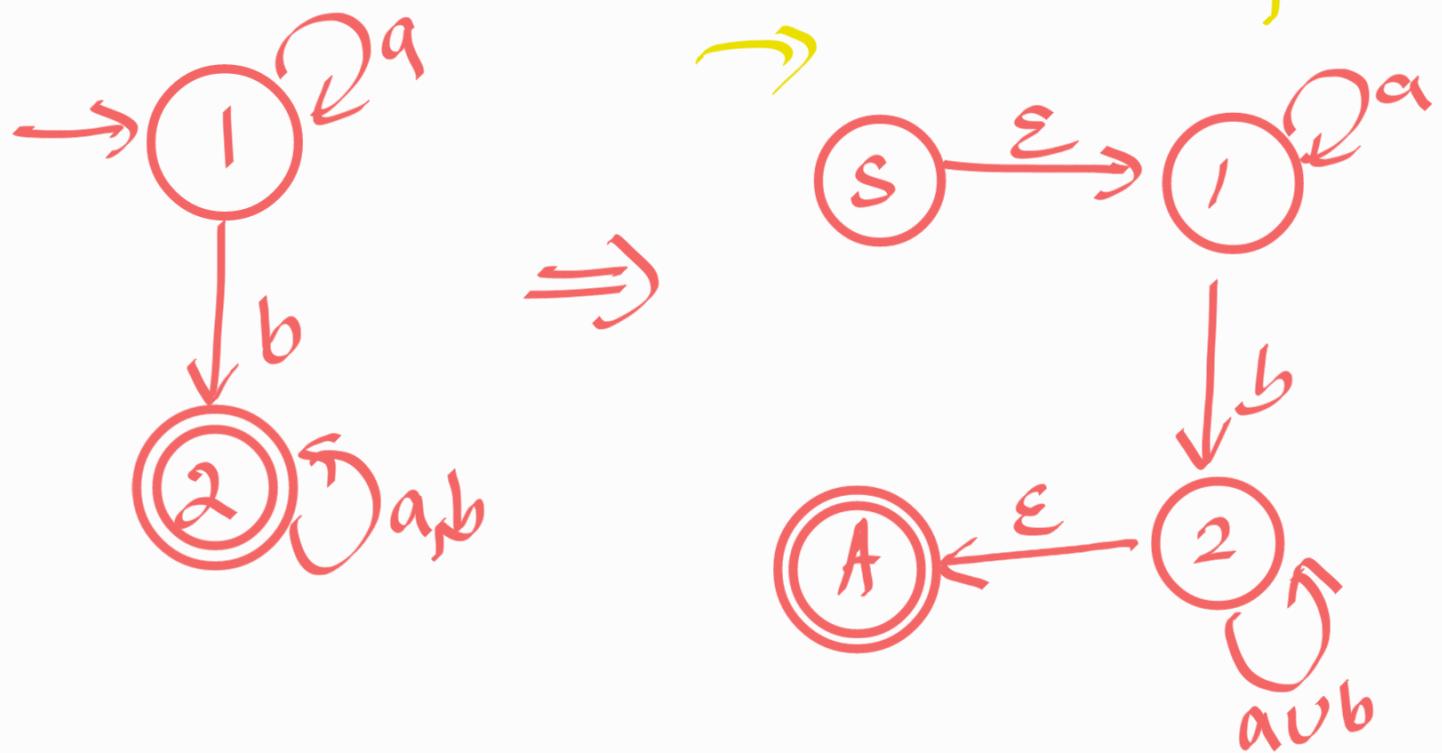
We basically rip out a state and  
repair the damage, so that the  
same language is recognized.

We can rip any state except start and accept state.

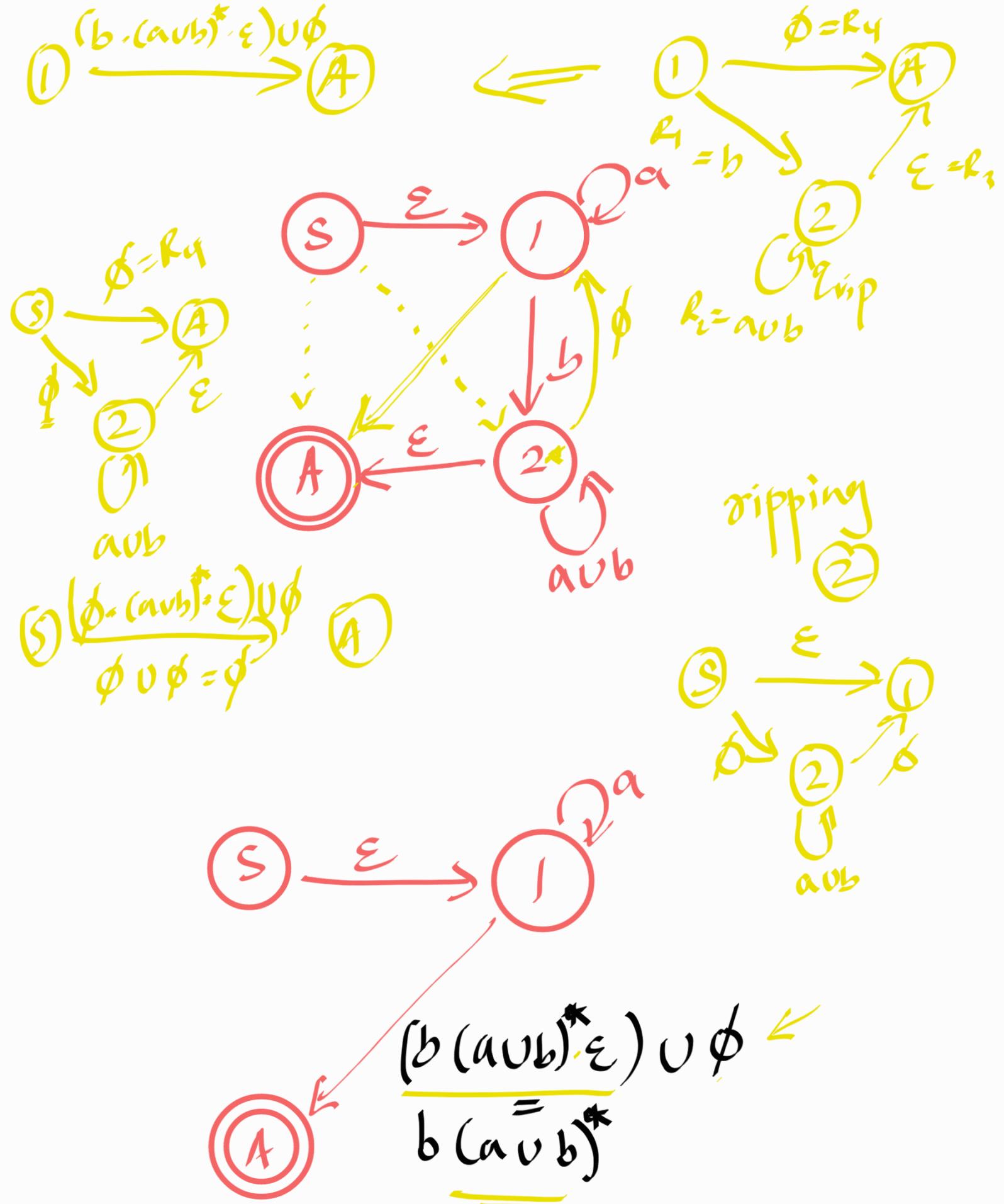


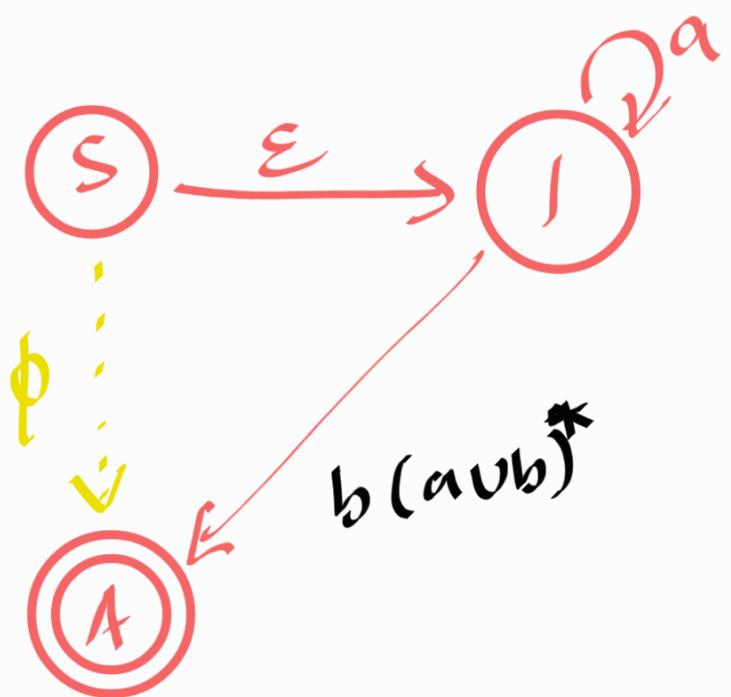
We make this change for each arrow going from any state  $q_i$  to any state  $q_j$  including the case where  $q_i = q_j$ .

Example : DFA  $\rightarrow$  GNFA  $\rightarrow$  Regular Expression.

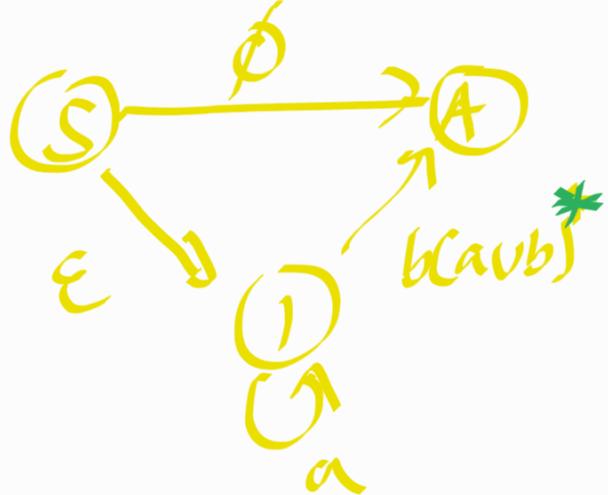


Sometimes we do not include unnecessary  $\phi$  transitions, since it clutters the graph.



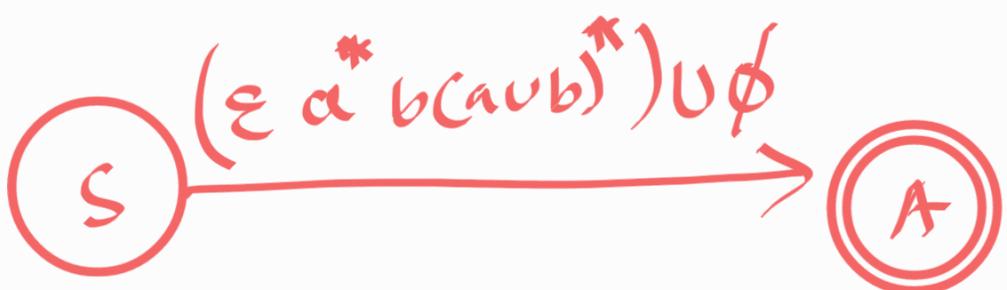


dipping ①



$$S \xrightarrow[\alpha^* \cdot b(a \cup b)^*]{\epsilon \cdot a^* \cdot b(a \cup b)^* \cup \phi} A$$

$$R \cdot \phi - \phi$$



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

This is the final regular expression.