Chapter 1

1.3 Regular Expressions

Regular Expression Definition (inductive definition)

- We can use the regular operations to
  - build up expressions describing languages,
  - which are called regular expressions.

- The value of a regular expression is a language.

- **Example**: $(0 \cup 1)0^*$
  - **value :** the language consisting of all strings starting with a 0 or a 1 followed by any number of 0s.

.

Regular Expression Definition (inductive definition)

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

.

Shorthand for the sets {0} and {1}.

Regular Expression Definition (inductive definition)

Chapter 1

Shorthand for $(\{0\} \cup \{1\}) = \{0,1\}$

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

.

Shorthand for the sets {0} and {1}.

Regular Expression Definition (inductive definition)

Shorthand for $(\{0\} \cup \{1\}) = \{0,1\}$

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

$\{0\}^*$

.

Shorthand for the sets $\{0\}$ and $\{1\}$.

Regular Expression Definition (inductive definition)

Shorthand for $(\{0\} \cup \{1\}) = \{0,1\}$

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

$\{0\}^*$

is shorthand for $(0 \cup 1) \circ 0^*$

Regular Expression Definition (inductive definition)

- R is a regular expression if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

Chapter 1

Regular Expression Definition (inductive definition)

- R is a regular expression if R is

1. a for some a in the alphabet Σ, Language: $\{a\}$

2. ε, Language: $\{\varepsilon\}$

3. $\emptyset$, Language: { }

4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,

5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or

6. (R_1^*), where R_1 is a regular expression.

Regular Expression Precedence

- Star → concatenation →union.
- unless parentheses change the usual order.

Chapter 1

Regular Expression Precedence

- Star → concatenation →union.
- unless parentheses change the usual order.

- Parentheses in an expression may be **omitted**.
  - If the evaluation is done in the precedence order
  - **Example:** $0^*1 \cup 0$

$$0^*10^* \cup 0^* = (0^*1) \cup 0^* \neq 0^*(1 \cup 0^*)$$

Regular Expression Definition (inductive definition)

- For convenience, we let $\mathbf{R^+}$ be shorthand for $\mathbf{RR^*}$.
  - R^* : **0 or more** concatenations of strings from R
  - R^+ : **1 or more** concatenations of strings from R
  - $R^+ \cup \varepsilon = R^*$.
  - R^k be shorthand for the **concatenation of k R's** with each other.

Regular Expression Definition (inductive definition)

- For convenience, we let $\mathbf{R^+}$ be shorthand for $\mathbf{RR^*}$.
  - R^* : **0 or more** concatenations of strings from R
  - R^+ : **1 or more** concatenations of strings from R
  - $R^+ \cup \varepsilon = R^*$.
  - R^k be shorthand for the **concatenation of k R's** with each other.

  - The language that describes **regular expression R** denoted by L(R).

Regular Expression : Examples

0^*10^* =?

$\Sigma^*1\Sigma^*$ =?

$\Sigma^*001\Sigma^*$ =?

$1^*(01^+)^*$ =?

$(\Sigma\Sigma)^*$ =?

$(\Sigma\Sigma\Sigma)^*$ =?

$01 \cup 10$ =?

Regular Expression : Example ($\Sigma = \{0,1\}$)

$0^*10^* = \{w \mid w \text{ contains a single } 1\}$.

$\Sigma^*1\Sigma^*$ =?

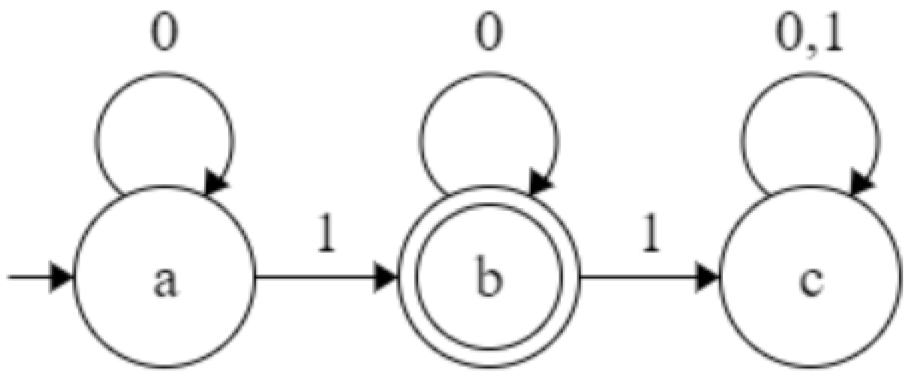
$\Sigma^*001\Sigma^*$ =?

$1^*(01^+)^*$ =?

$(\Sigma\Sigma)^*$ =?

$(\Sigma\Sigma\Sigma)^*$ =?

$01 \cup 10$ =?



Regular Expression : Example ($\Sigma = \{0,1\}$)

$0^*10^* = \{w \mid w \text{ contains a single } 1\}$.

$\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.

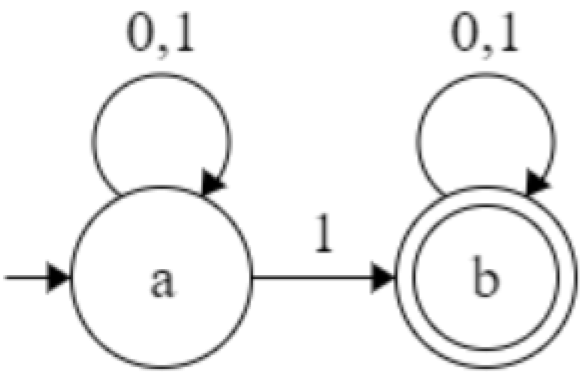
$\Sigma^*001\Sigma^*$ =?

$1^*(01^+)^*$ =?

$(\Sigma\Sigma)^*$ =?

$(\Sigma\Sigma\Sigma)^*$ =?

$01 \cup 10$ =?



Regular Expression : Example ($\Sigma = \{0,1\}$)

$0^*10^* = \{w \mid w \text{ contains a single } 1\}$.

$\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.

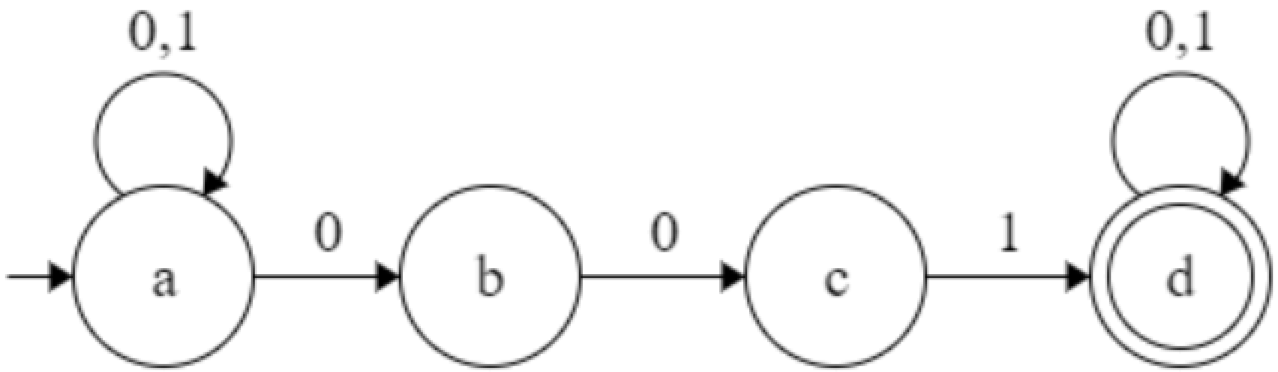
$\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.

$1^*(01^+)^*$ =?

$(\Sigma\Sigma)^*$ =?

$(\Sigma\Sigma\Sigma)^*$ =?

$01 \cup 10$ =?



Chapter 1

Regular Expression : Example ($\Sigma = \{0,1\}$)

$0^*10^* = \{w \mid w \text{ contains a single } 1\}$.

$\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.

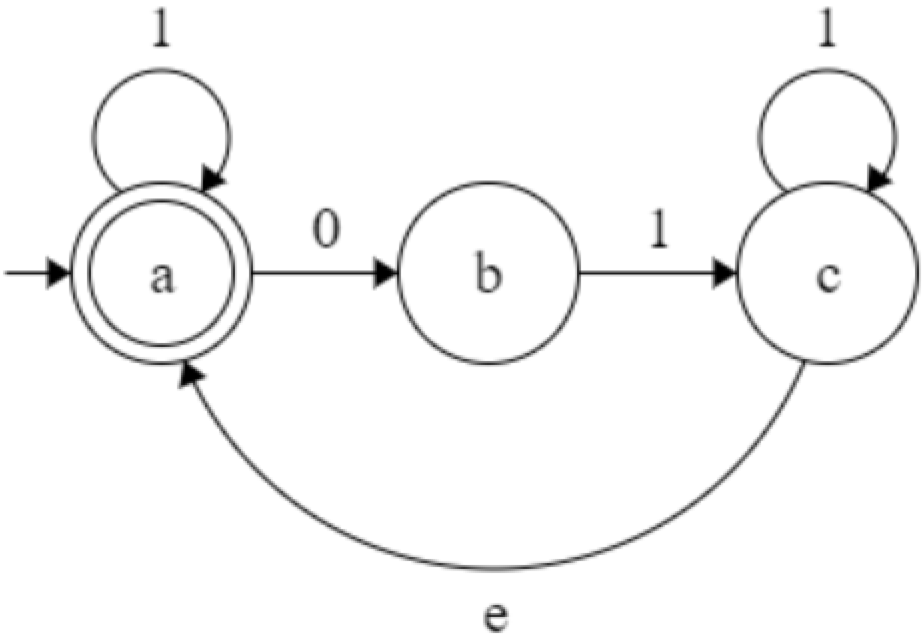
$\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.

$1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.

$(\Sigma\Sigma)^*$ =?

$(\Sigma\Sigma\Sigma)^*$ =?

$01 \cup 10$ =?



Regular Expression : Example ($\Sigma = \{0,1\}$)

$0^*10^* = \{w \mid w \text{ contains a single } 1\}$.

$\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.

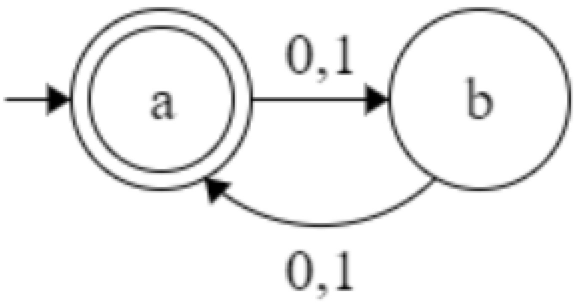
$\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.

$1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.

$(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.

$(\Sigma\Sigma\Sigma)^*$ =?

$01 \cup 10$ =?



* The length of a string is the number of symbols that it contains.

Regular Expression : Example ($\Sigma = \{0,1\}$)

$0^*10^* = \{w \mid w \text{ contains a single } 1\}$.

$\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.

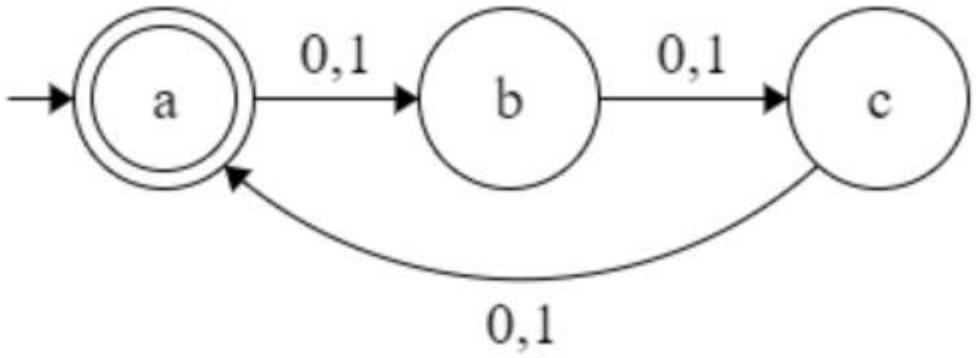
$\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.

$1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.

$(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]

$(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$.

$01 \cup 10$ =?



Regular Expression : Example ($\Sigma = \{0,1\}$)

$0^*10^* = \{w \mid w \text{ contains a single } 1\}$.

$\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.

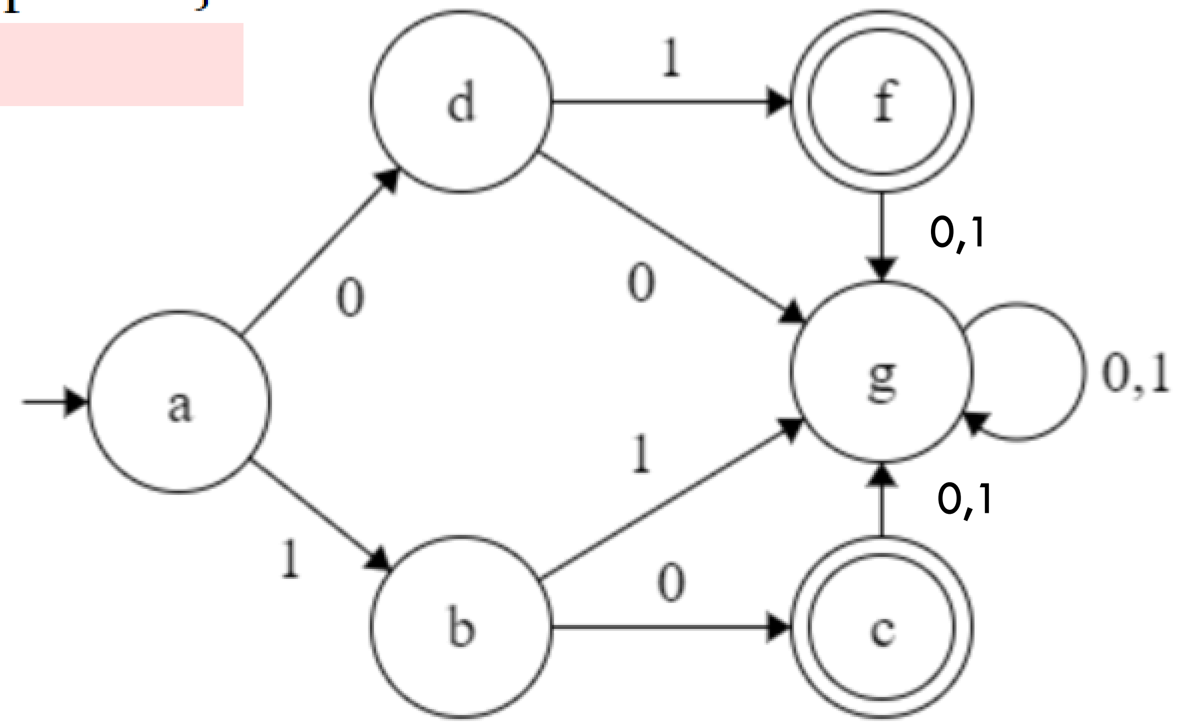
$\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.

$1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.

$(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]

$(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$.

$01 \cup 10 = \{01, 10\}$.



Chapter 1

Regular Expression Definition

- If we let **R** be any **regular expression**, we have the following identities.
  - $R \cup \emptyset = R$
  - $R \circ \varepsilon = R$

Regular Expression Definition

- If we let **R** be any **regular expression**, we have the following identities.
  - $R \cup \emptyset = R$
  - $R \circ \varepsilon = R$

- But
  - $R \cup \varepsilon$ may not equal R : $R=0 \rightarrow L(R)=\{0\}$ but $L(R\cup\varepsilon)=\{0,\varepsilon\}$.
  - $R \circ \emptyset$ may not equal R : $R=0 \rightarrow L(R)=\{0\}$ but $L(R\circ\emptyset)=\emptyset$

EQUIVALENCE WITH FINITE AUTOMATA

Theorem : A language is regular if and only if some regular expression describes it.

- This theorem has two directions.
  - Part 1: A language is described by a regular expression, then it is regular.
  - Part 2: A language is regular, then some regular expression describes it.

Chapter 1

EQUIVALENCE WITH FINITE AUTOMATA

Theorem : A language is regular if and only if some regular expression describes it.

- This theorem has two directions.
  - **Part 1:** A language is described by a regular expression, then it is regular.
  - Part 2: A language is regular, then some regular expression describes it.

- **Proof Idea :** (proof by construction)

EQUIVALENCE WITH FINITE AUTOMATA

Theorem : A language is regular if and only if some regular expression describes it.

- This theorem has two directions.
  - Part 1: A language is described by a regular expression, then it is regular.
  - Part 2: A language is regular, then some regular expression describes it.

- Proof Idea : (proof by construction)
  - A regular expression R describing some language A.
  - We show how to convert R into an NFA recognizing A.
    - By using this corollary : if an NFA recognizes A then A is regular.

EQUIVALENCE WITH FINITE AUTOMATA

Theorem : A language is regular if and only if some regular expression describes it.

- This theorem has two directions.
  - Part 1: A language is described by a regular expression, then it is regular.
  - Part 2: A language is regular, then some regular expression describes it.

- Proof :
  - Let's convert R into an NFA N.
  - We consider the six cases in the formal definition of regular expressions.

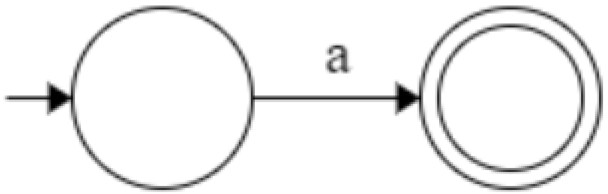
1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

EQUIVALENCE WITH FINITE AUTOMATA

- Part 1: A language is described by a regular expression, then it is regular.

- Proof (cont.) :

1. R = a for some a∈Σ.
   - Then L(R)={a}, and the following NF A recognizes L(R).

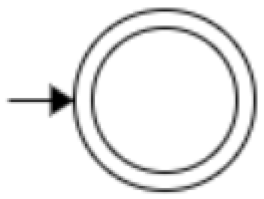


EQUIVALENCE WITH FINITE AUTOMATA

- Part 1: A language is described by a regular expression, then it is regular.

- Proof (cont.) :

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



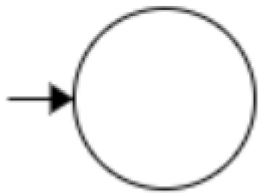
Chapter 1

EQUIVALENCE WITH FINITE AUTOMATA

- Part 1: A language is described by a regular expression, then it is regular.

- Proof (cont.) :

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

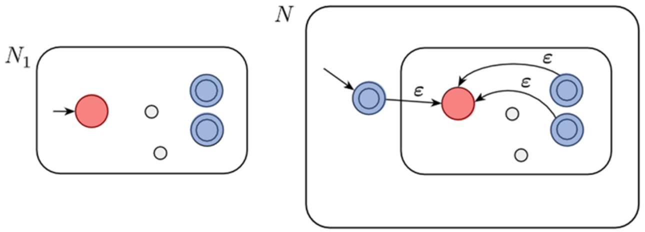


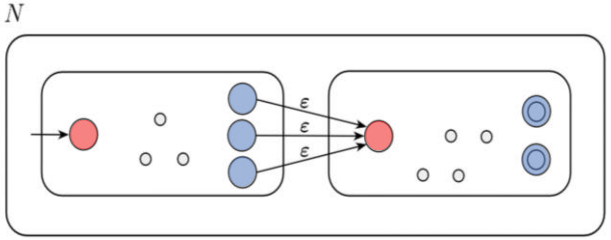
EQUIVALENCE WITH FINITE AUTOMATA

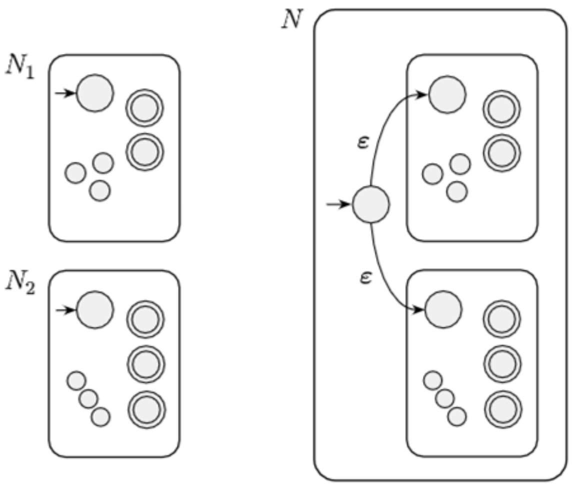
- Part 1: A language is described by a regular expression, then it is regular.

- Proof (cont.) :

4. $R = R_1 \cup R_2$
5. $R = R_1 \circ R_2$
6. $R = R_1{}^*$.
   - we use the constructions given in the proofs that the class of regular languages is closed under the regular operations.







EQUIVALENCE WITH FINITE AUTOMATA

- Example: convert the regular expression $(ab \cup a)^*$ to an NFA.

EQUIVALENCE WITH FINITE AUTOMATA

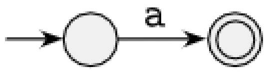
- **Example:** convert the regular expression $(ab \cup a)^*$ to an NFA.

a

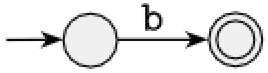
EQUIVALENCE WITH FINITE AUTOMATA

- **Example:** convert the regular expression $(ab \cup a)^*$ to an NFA.

a



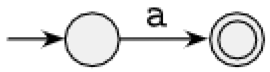
b



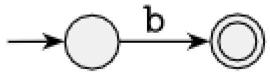
EQUIVALENCE WITH FINITE AUTOMATA

- **Example:** convert the regular expression $(ab \cup a)^*$ to an NFA.

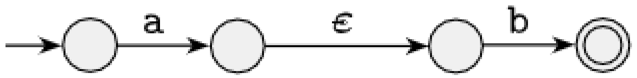
a

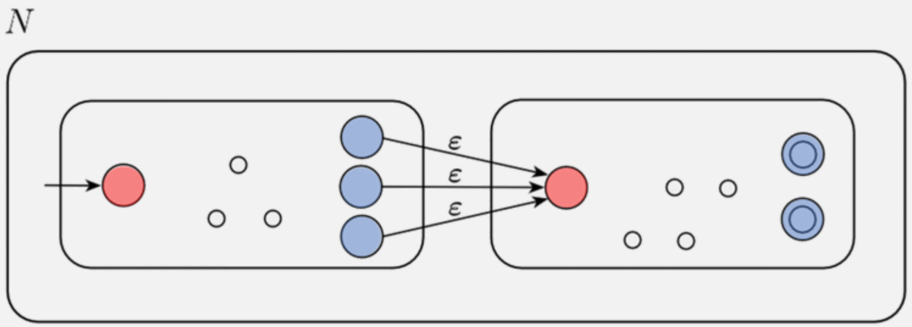


b



ab



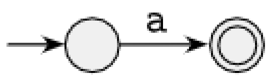


NFA Used for concertation of two regular languages.

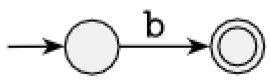
EQUIVALENCE WITH FINITE AUTOMATA

- Example: convert the regular expression $(ab \cup a)^*$ to an NFA.

a

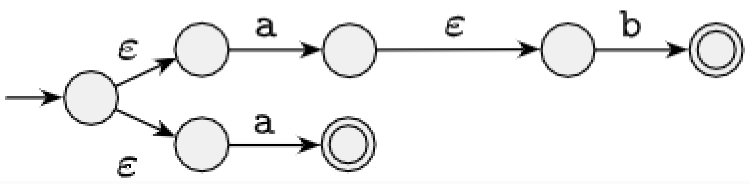


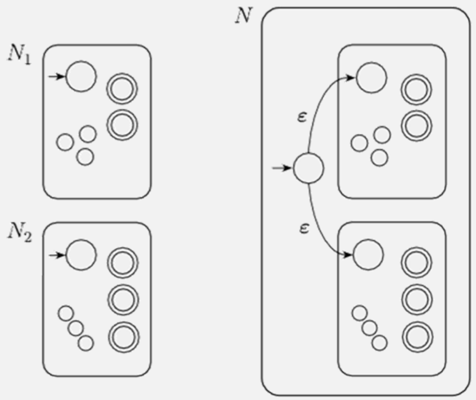
b



ab

$ab \cup a$

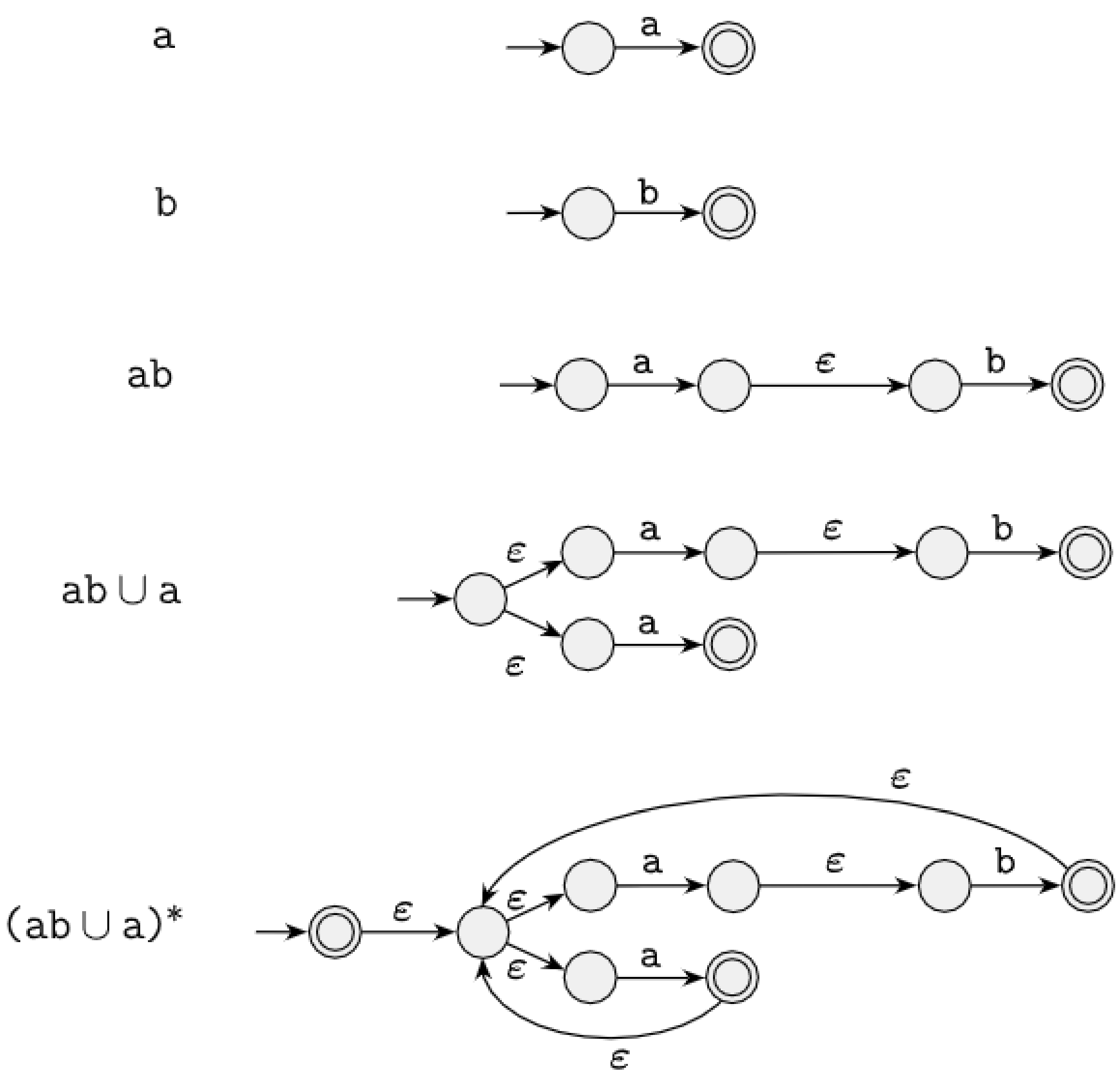




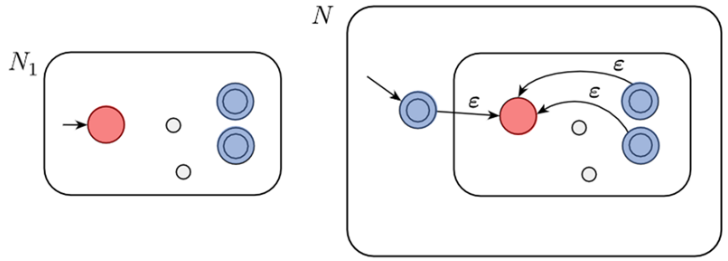
NFA Used for union of two regular languages.

EQUIVALENCE WITH FINITE AUTOMATA

- **Example:** convert the regular expression $(ab \cup a)^*$ to an NFA.



NFA Used for star of two regular languages.



EQUIVALENCE WITH FINITE AUTOMATA

- **Example:** convert the regular expression $(a \cup b)^*aba$ to an NFA.

EQUIVALENCE WITH FINITE AUTOMATA

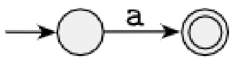
- **Example:** convert the regular expression $(a \cup b)^*aba$ to an NFA.

a

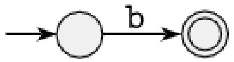
EQUIVALENCE WITH FINITE AUTOMATA

- **Example:** convert the regular expression $(a \cup b)^*aba$ to an NFA.

a

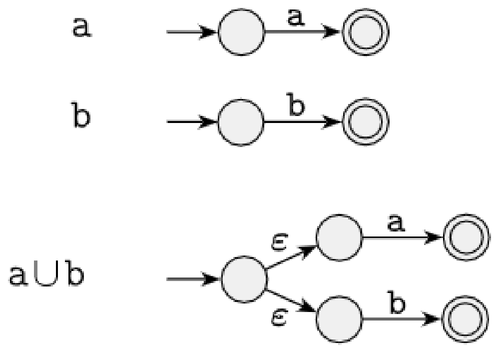


b



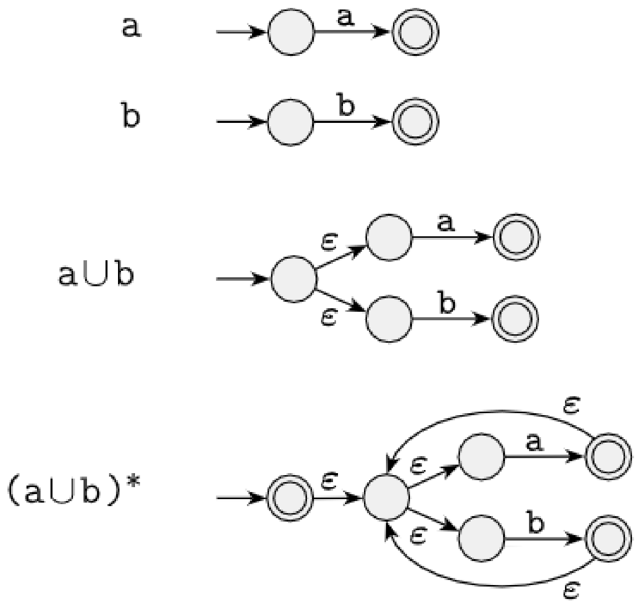
EQUIVALENCE WITH FINITE AUTOMATA

- **Example:** convert the regular expression $(a \cup b)^* aba$ to an NFA.



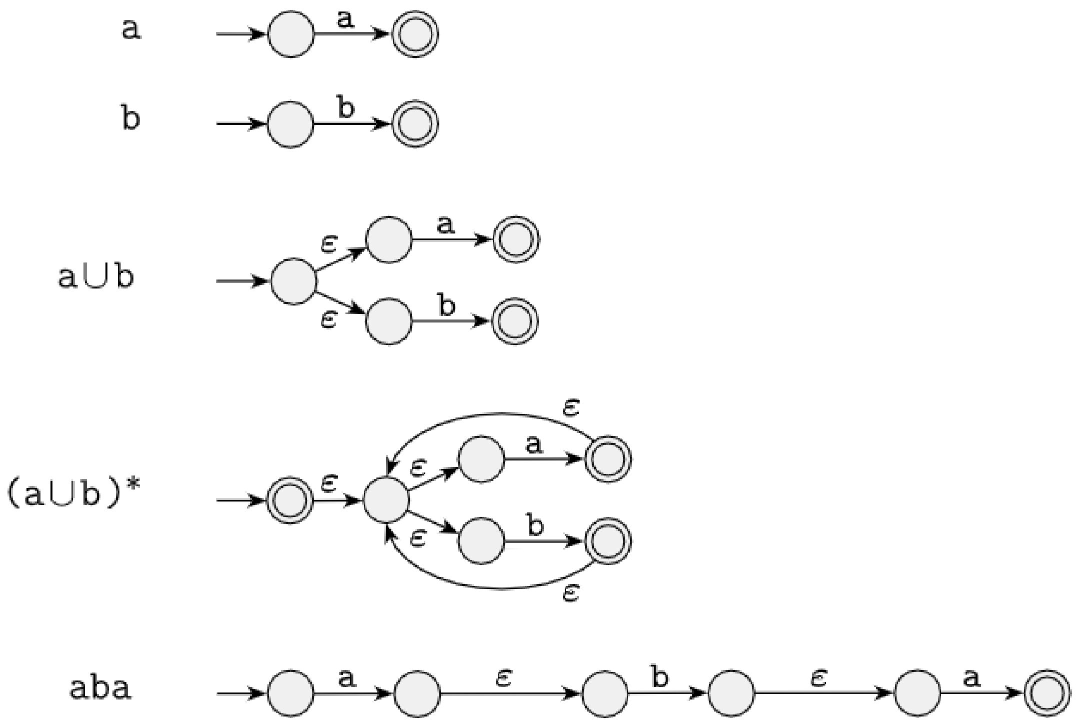
EQUIVALENCE WITH FINITE AUTOMATA

- **Example:** convert the regular expression $(a \cup b)^{*}aba$ to an NFA.



EQUIVALENCE WITH FINITE AUTOMATA

- Example: convert the regular expression $(a\cup b)^*aba$ to an NFA.

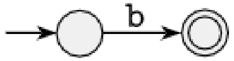


EQUIVALENCE WITH FINITE AUTOMATA

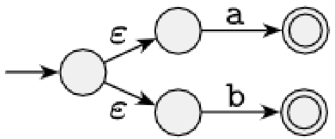
- **Example:** convert the regular expression $(a \cup b)^* aba$ to an NFA.

a

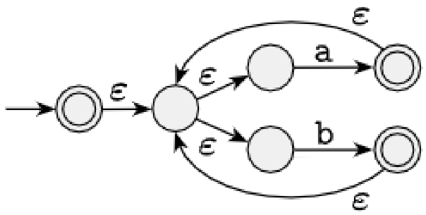
b



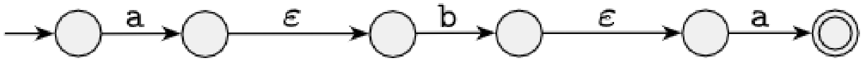
a∪b



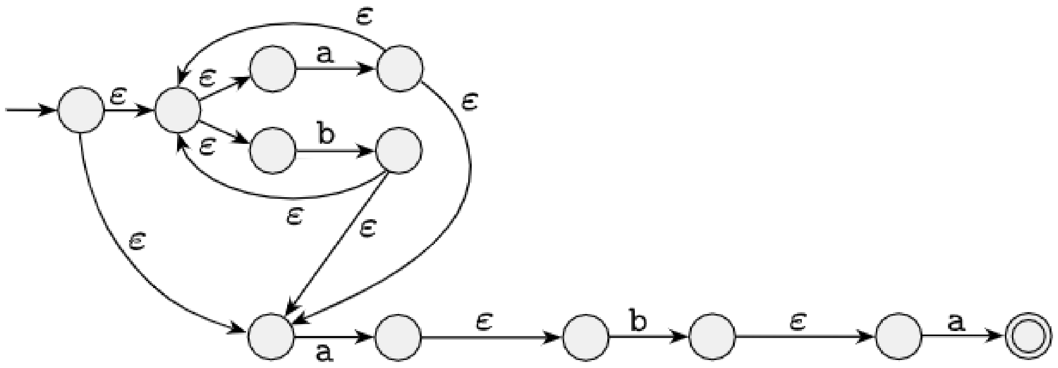
(a∪b)*

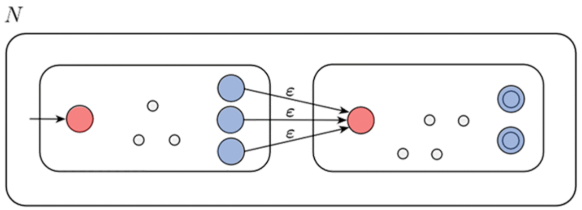


aba



(a∪b)*aba





EQUIVALENCE WITH FINITE AUTOMATA

Theorem : A language is regular if and only if some regular expression describes it.

- This theorem has two directions.
  - Part 1: A language is described by a regular expression, then it is regular.
  - Part 2: A language is regular, then some regular expression describes it.

EQUIVALENCE WITH FINITE AUTOMATA

Part 2: A language is regular, then some regular expression describes it.

- Proof Idea:
  - Because A is regular, it is **accepted by a DFA**.
  - We describe a procedure for converting DFAs into equivalent **regular expressions**.

EQUIVALENCE WITH FINITE AUTOMATA

Part 2: A language is regular, then some regular expression describes it.

- Proof Idea:
  - Because A is regular, it is **accepted by a DFA.**
  - We describe a procedure for converting DFAs into equivalent **regular expressions.**
  - We break this procedure into **two parts**,
    - using a new type of finite automaton called a generalized nondeterministic finite automaton, GNFA.

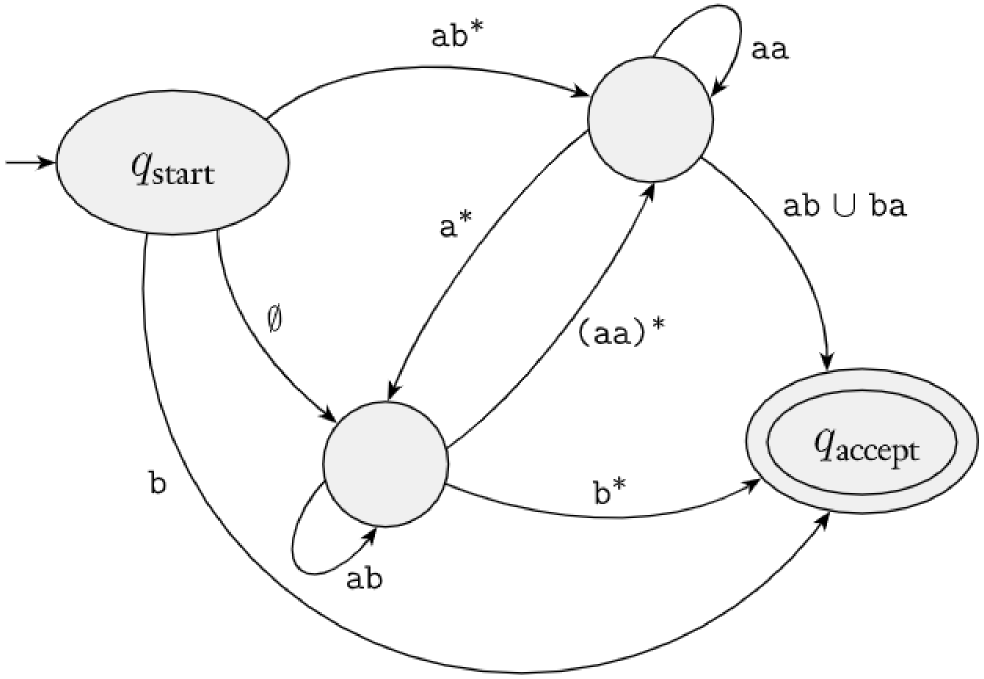
EQUIVALENCE WITH FINITE AUTOMATA

Part 2: A language is regular, then some regular expression describes it.

- Proof Idea:
  - Because A is regular, it is **accepted by a DFA.**
  - We describe a procedure for converting DFAs into equivalent **regular expressions.**
  - We break this procedure into **two parts**,
    - using a new type of finite automaton called a generalized nondeterministic finite automaton, GNFA.
    - **First** we show how to **convert DFAs into GNFAs**, and
    - **then GNFAs into regular expressions.**

Generalized nondeterministic finite automata (GNFA)

- is a **nondeterministic finite automata** wherein **the transition arrows may have any regular expressions as labels**, instead of only members of the alphabet or ε.

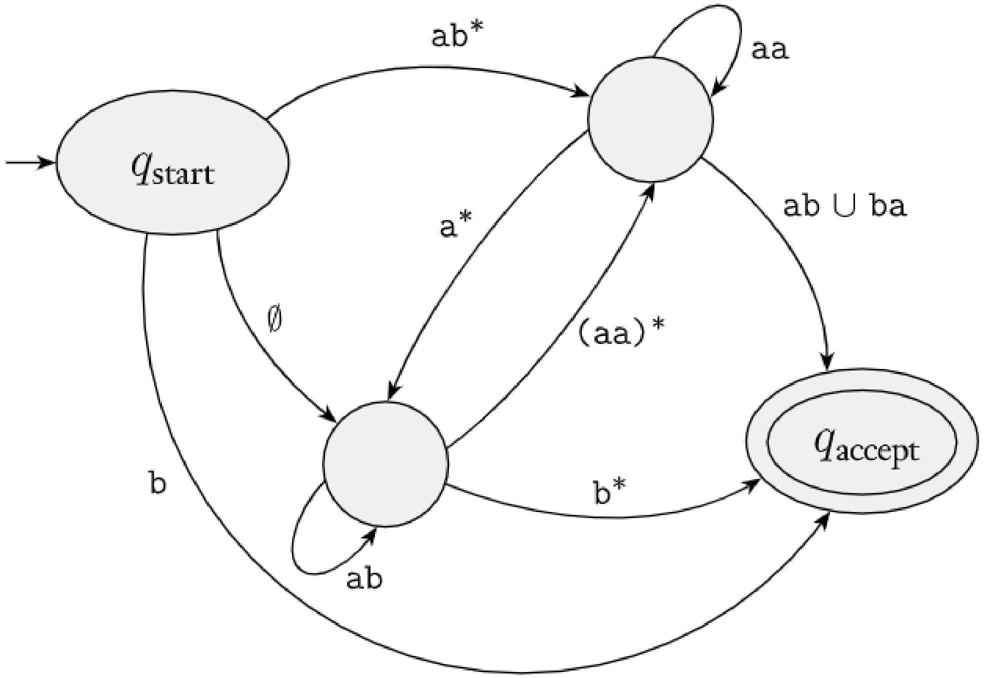


Chapter 1

Generalized nondeterministic finite automata (GNFA)

- is a nondeterministic finite automata wherein **the transition arrows may have any regular expressions as labels**, instead of only members of the alphabet or ε.

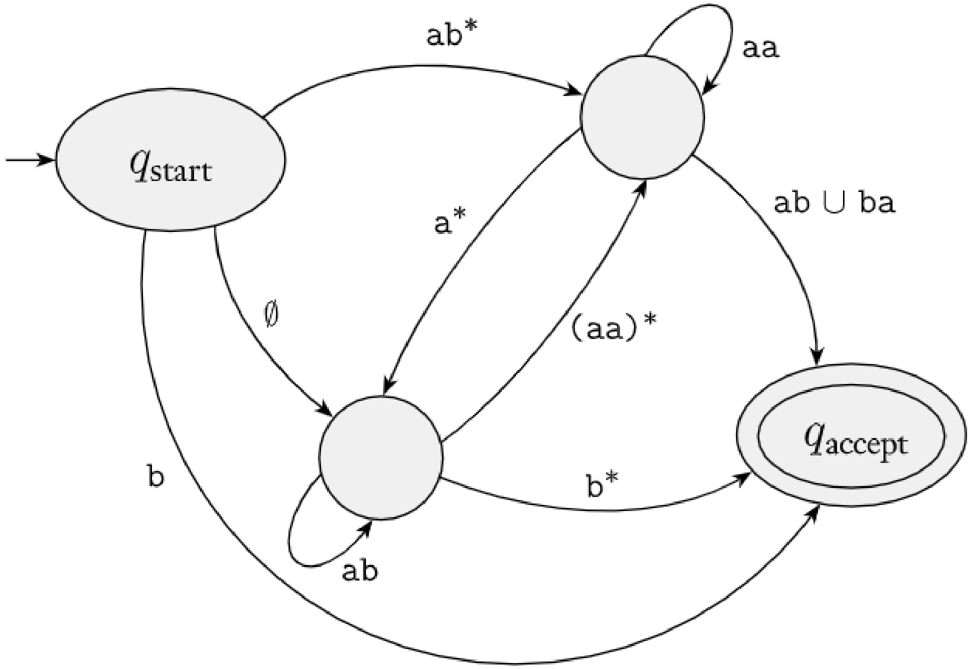
- reads blocks of symbols from the input,
  - **not** necessarily **just one symbol at a time** as in an ordinary NFA.



Chapter 1

Generalized nondeterministic finite automata (GNFA)

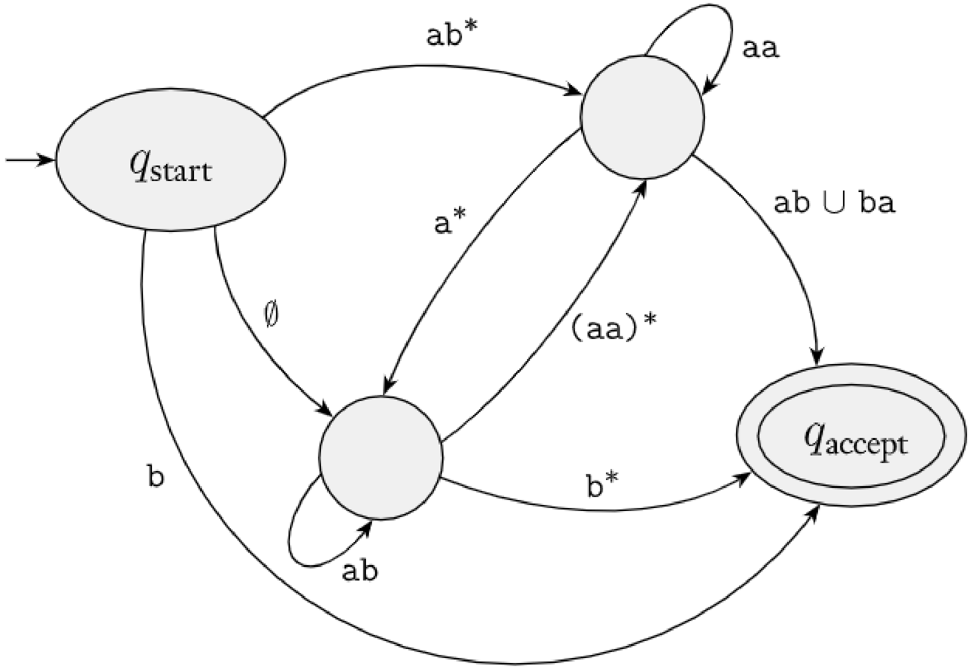
- moves along a transition arrow connecting two states
  - by reading a block of symbols from the input,
    - a string described by the regular expression on that arrow.



Generalized nondeterministic finite automata (GNFA)

- moves along a transition arrow connecting two states
  - by reading a block of symbols from the input,
    - a string described by the regular expression on that arrow.

- accepts its **input** if its processing can cause the GNFA to be in an accept state at the end of the input.



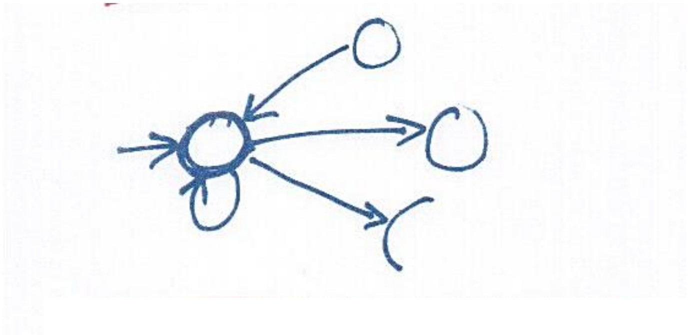
Generalized nondeterministic finite automata (GNFA)

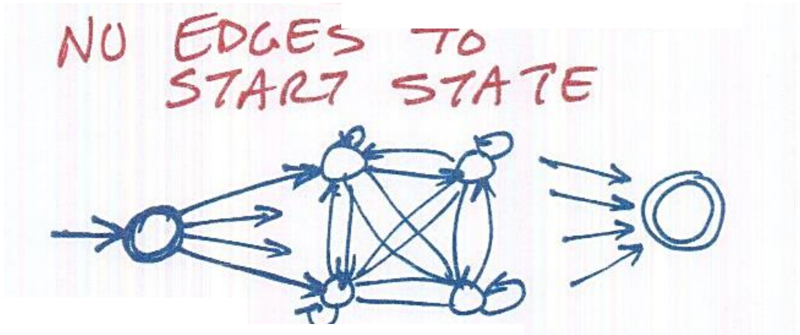
- For convenience, we require that GNFAs always have a special form that meets the following conditions.

Generalized nondeterministic finite automata (GNFA)

- For convenience, we require that GNFAs always have a special form that meets the following conditions.

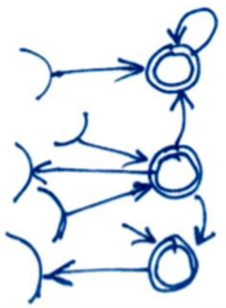
  - The start state has transition arrows **going to every other state**
    - but no arrows **coming in from any other state**.





Generalized nondeterministic finite automata (GNFA)

- For convenience, 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 arrows **coming in from any other state**.
  - There is only a single accept state,
    - and it has arrows **coming in from every**
    - but no arrows **going to any other** state.
  - the **accept state** is not the same as the **start state**.



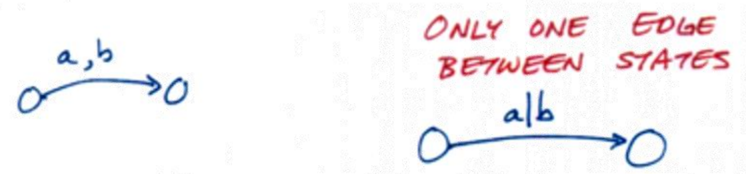
Generalized nondeterministic finite automata (GNFA)

- For convenience, 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 arrows **coming in from any other state**.

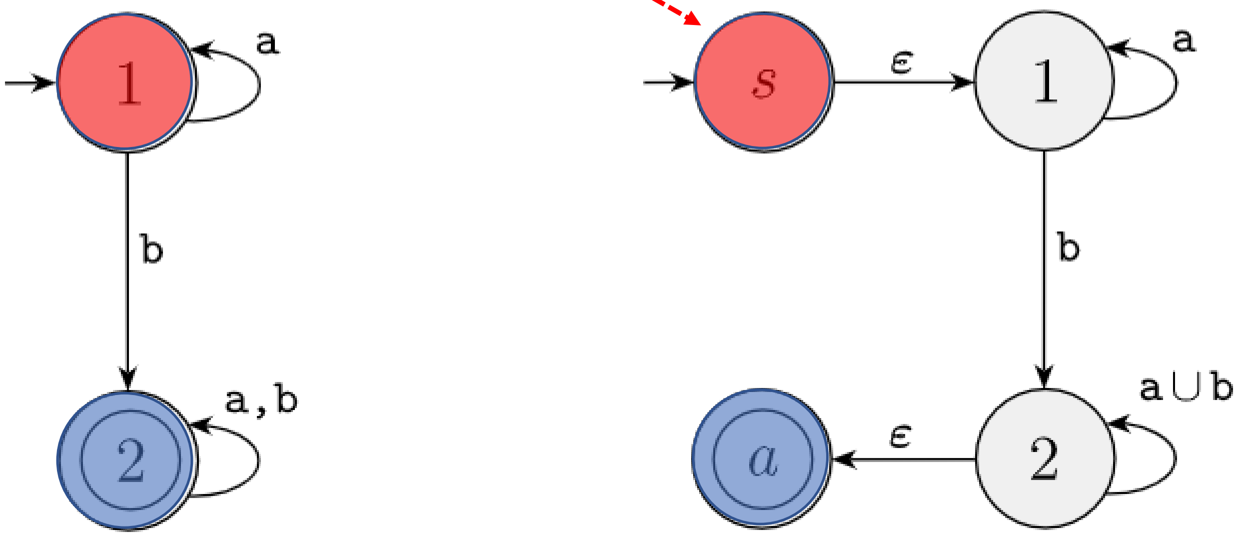
  - There is only a single accept state,
    - and it has arrows **coming in from every** other state
    - but no arrows **going to any other** state.
  - the **accept state** is not the same as the **start state**.

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



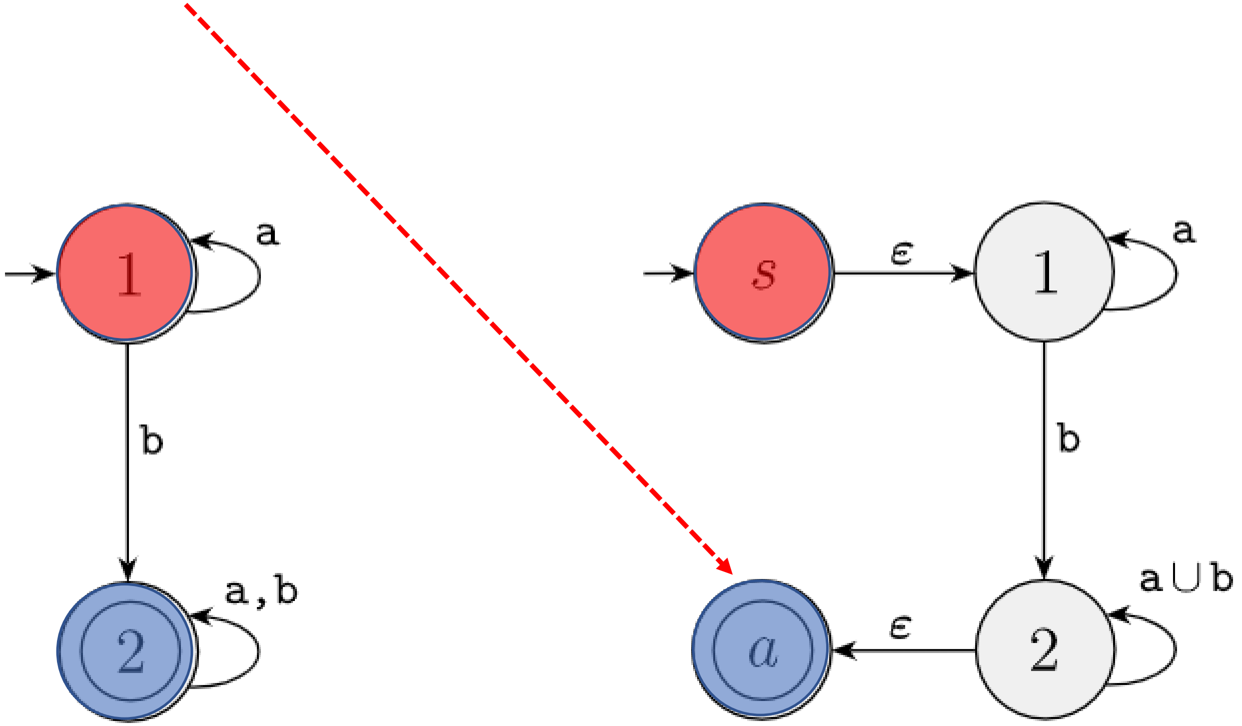
Converting a DFA into a GNFA

1. We simply add
   - a new start state with an ε arrow to the **old start state** and



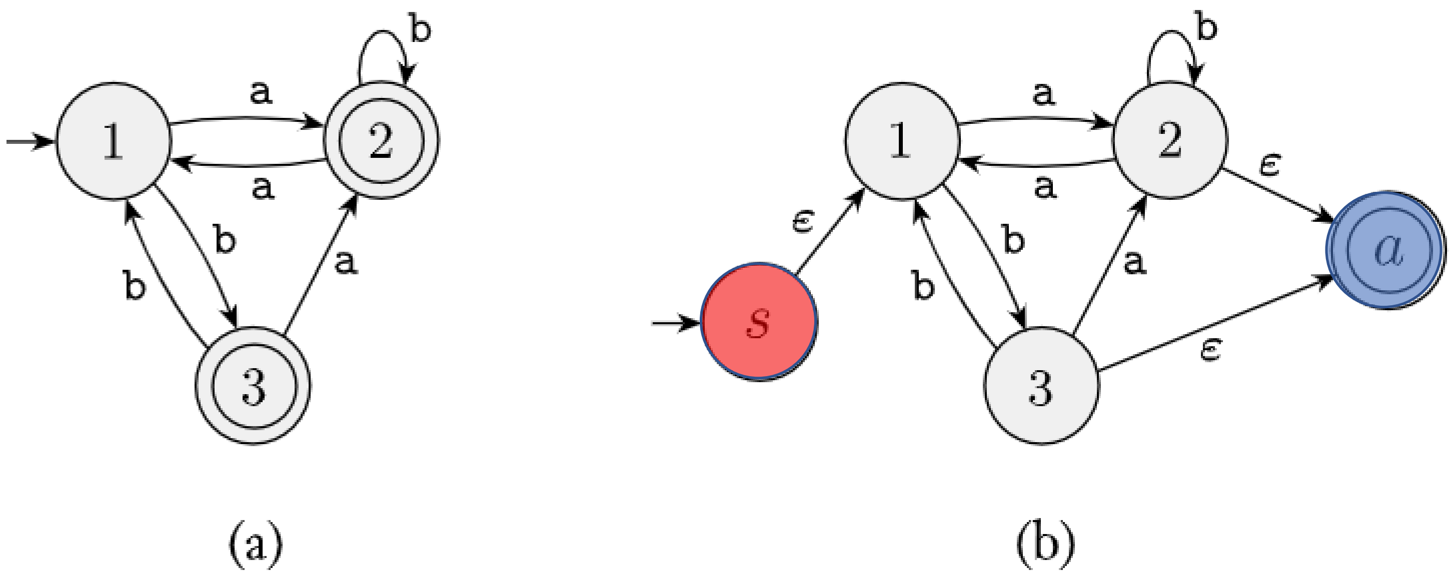
Converting a DFA into a GNFA

1. We simply add
   - a new start state with an ε arrow to the **old start state** and
   - a new accept state with ε arrows from the **old accept states**.



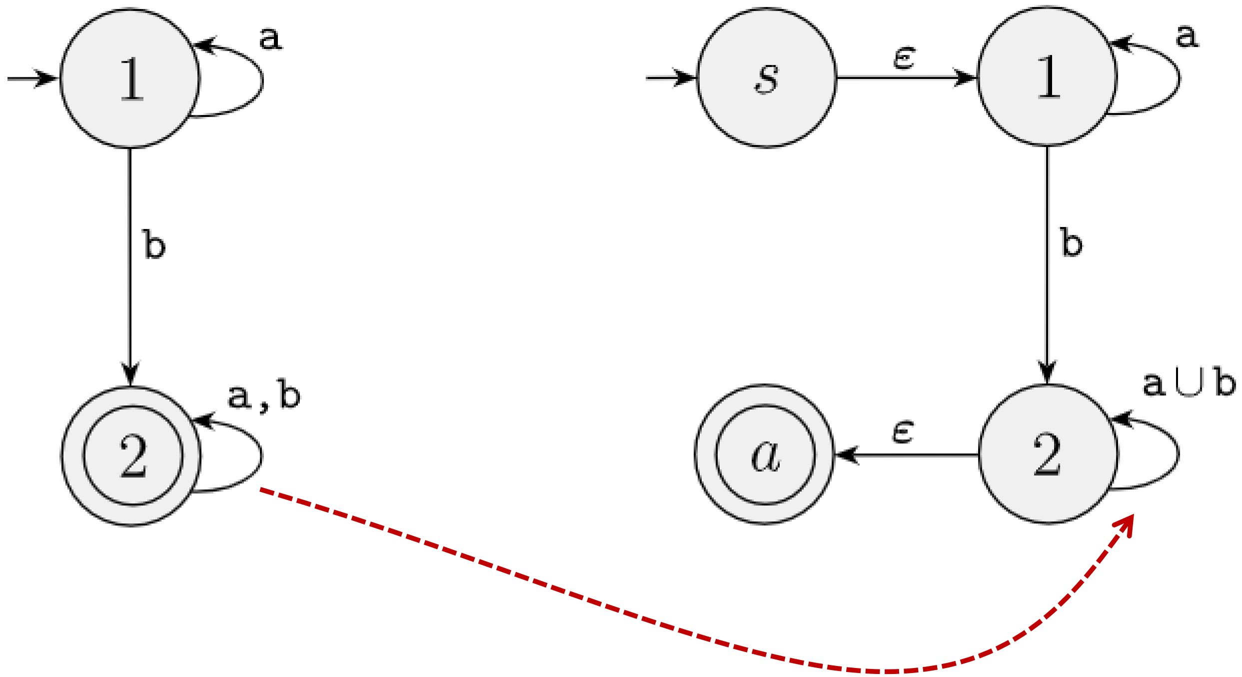
Converting a DFA into a GNFA

1. We simply add
   - a new start state with an ε arrow to the **old start state** and
   - a new accept state with ε arrows from the **old accept states**.



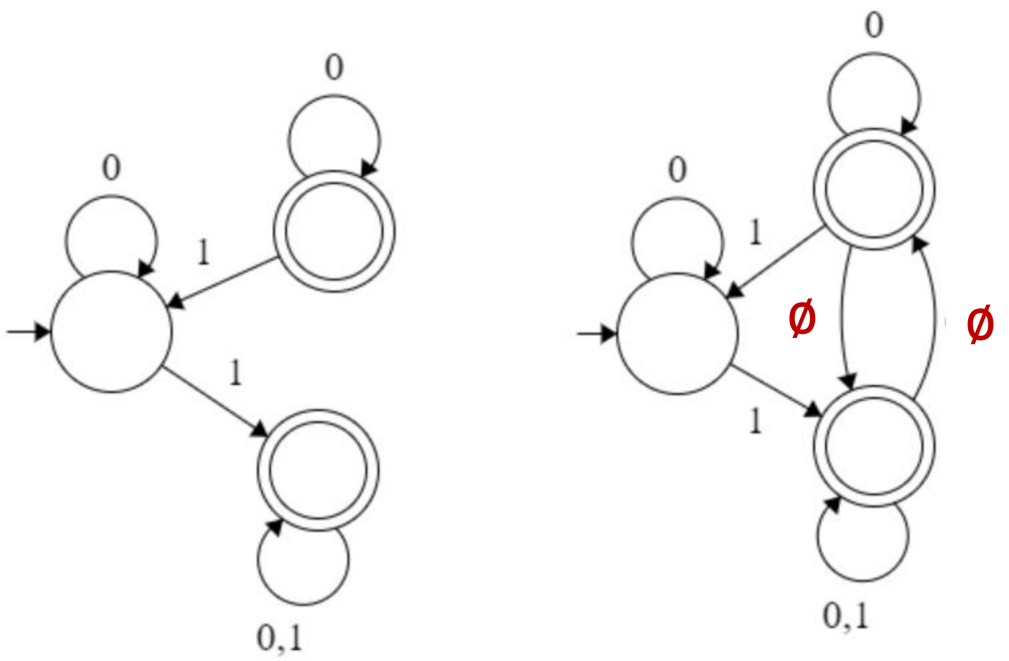
Converting a DFA into a GNFA

2. If any arrows have multiple labels (or multiple arrows going between the same 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 into a GNFA

3. Finally, we add **arrows labeled ∅**
   - between states that had **no arrows**.



Converting a DFA into a GNFA

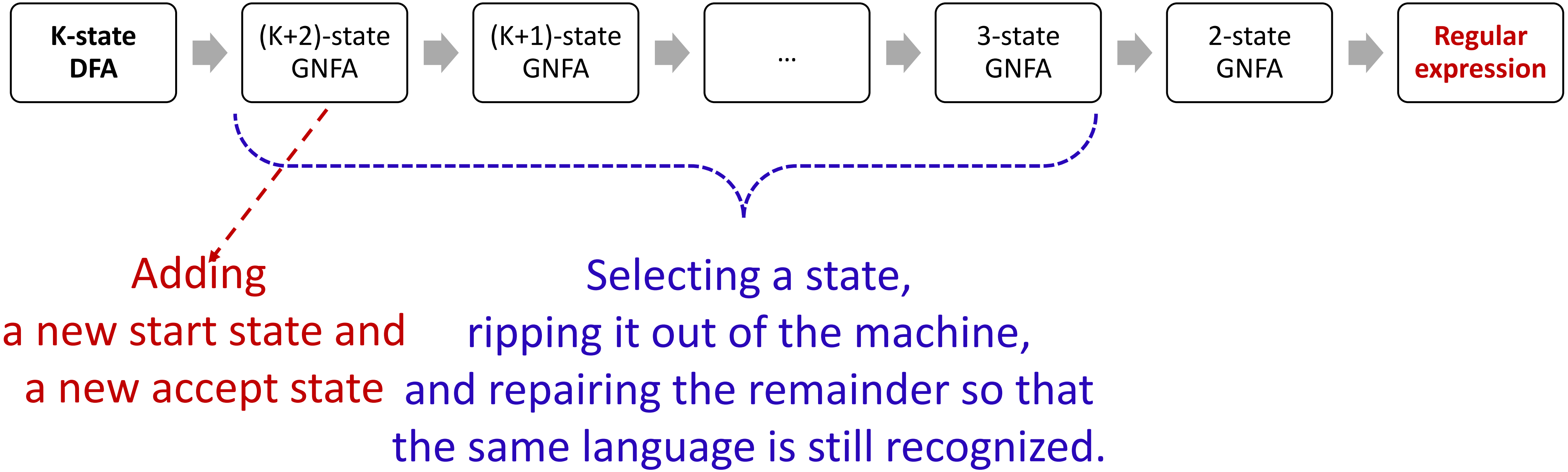
The preceding algorithm to convert a DFA into a regular expression.

K-state DFA → (K+2)-state GNFA → (K+1)-state GNFA → … → 3-state GNFA → 2-state GNFA → **Regular expression**

Adding
a new start state and
a new accept state

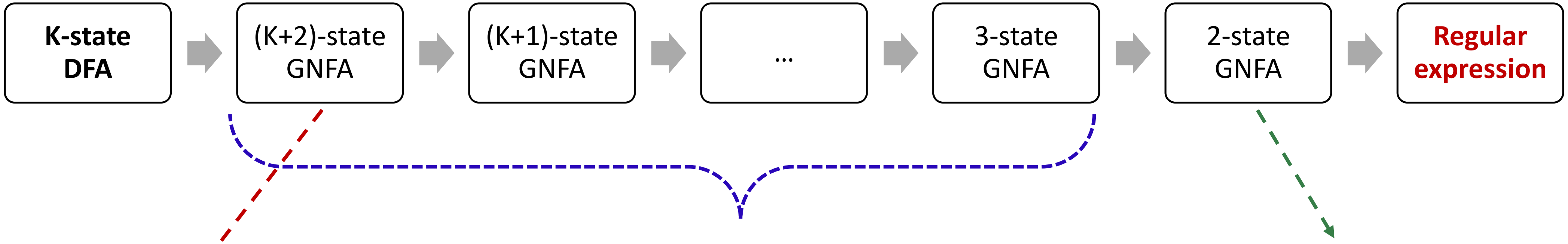
Converting a DFA into a GNFA

The preceding algorithm to convert a DFA into a regular expression.



Converting a DFA into a GNFA

The preceding algorithm to convert a DFA into a regular expression.



Adding
a new start state and
a new accept state

Selecting a state,
ripping it out of the machine,
and repairing the remainder so that
the same language is still recognized.

The label of the arrow
that connects start state
and accept state is the
equivalent RE.

Converting a DFA into a GNFA

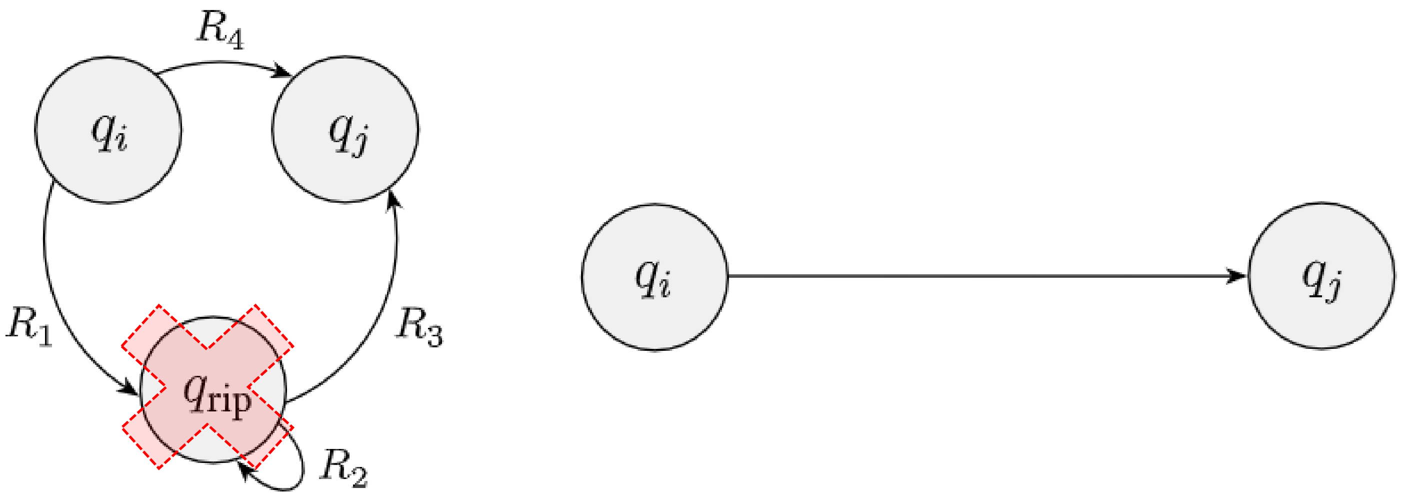
The preceding algorithm to convert a DFA into a regular expression.

K-state DFA → (K+2)-state GNFA → (K+1)-state GNFA → … → 3-state GNFA → 2-state GNFA → **Regular expression**

3-state DFA → 5-state GNFA → 4-state GNFA → 3-state GNFA → 2-state GNFA → **Regular expression**

Converting a DFA into a GNFA

- How can we rip a state out of the machine and repairing the remainder so that
- the same language is still recognized ?



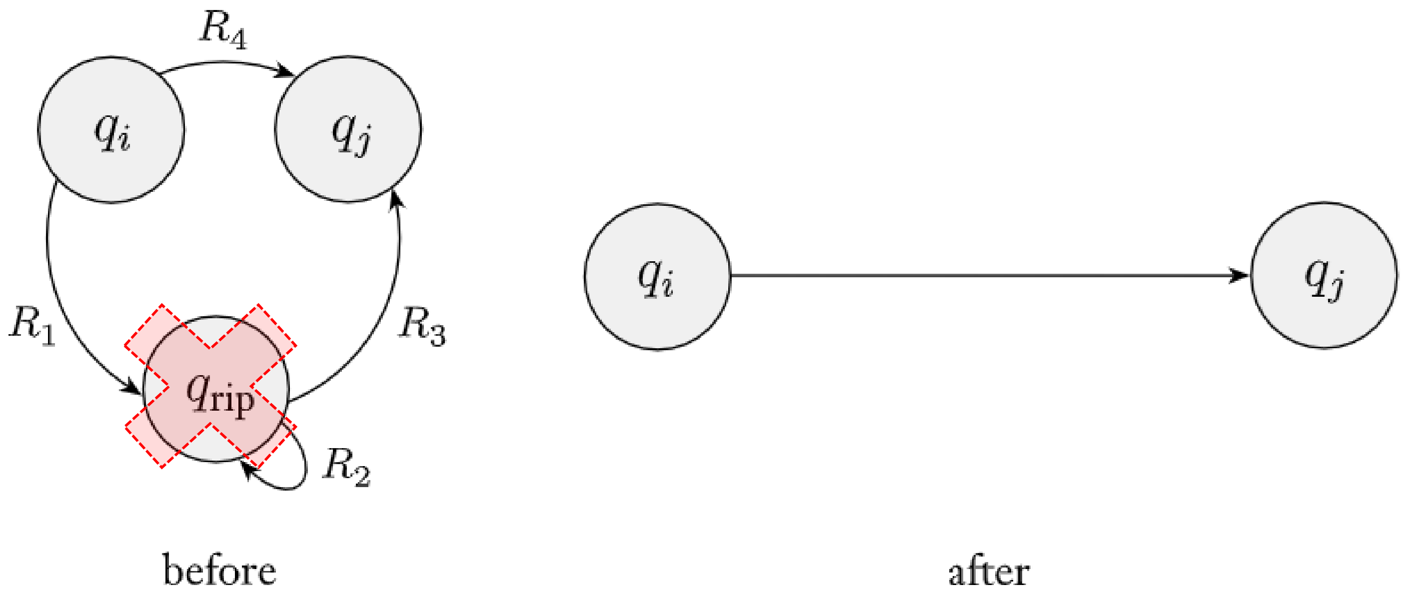
before

after

Converting a DFA into a GNFA

- How can we rip a state out of the machine and repairing the remainder so that
- the same language is still recognized ?

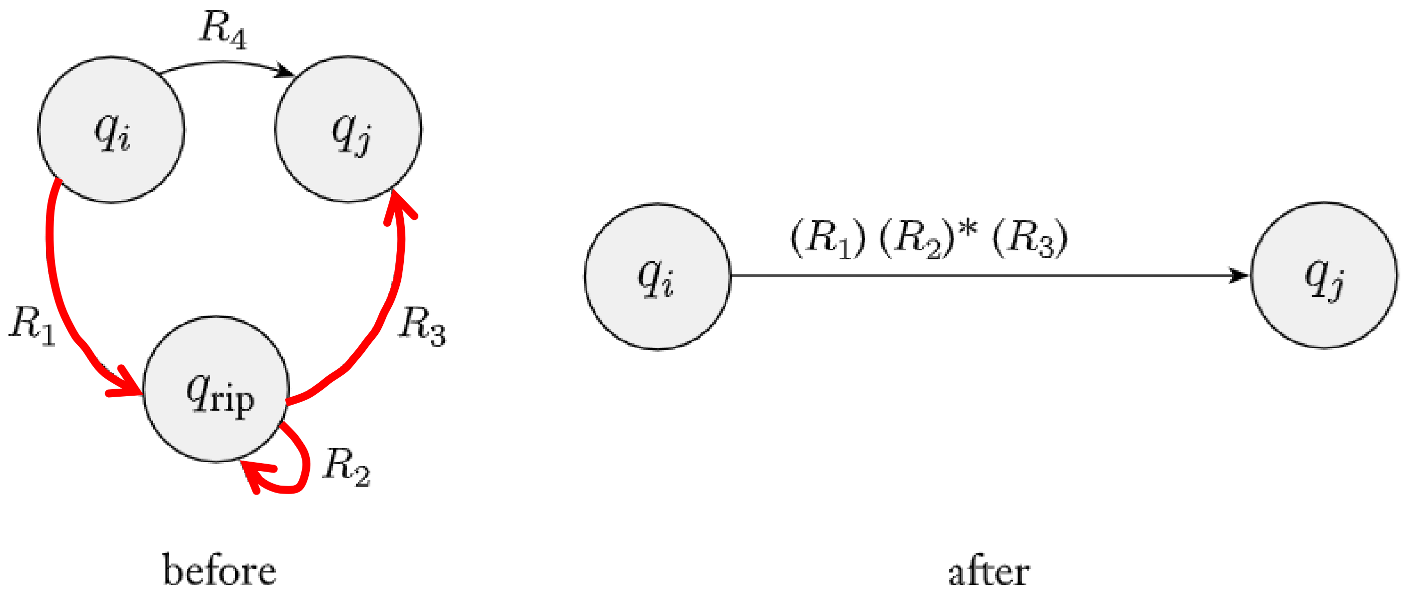
- The **new label** going from a state q_i to a state q_j is a regular expression
  - that describes **all strings** that would take the machine from q_i to q_j
    - either directly or via q_{rip}.
- In the following DFA, R_1, R_2,R_3, and R_4 are regular expressions.



Converting a DFA into a GNFA

- How can we rip a state out of the machine and repairing the remainder so that
- the same language is still recognized ?

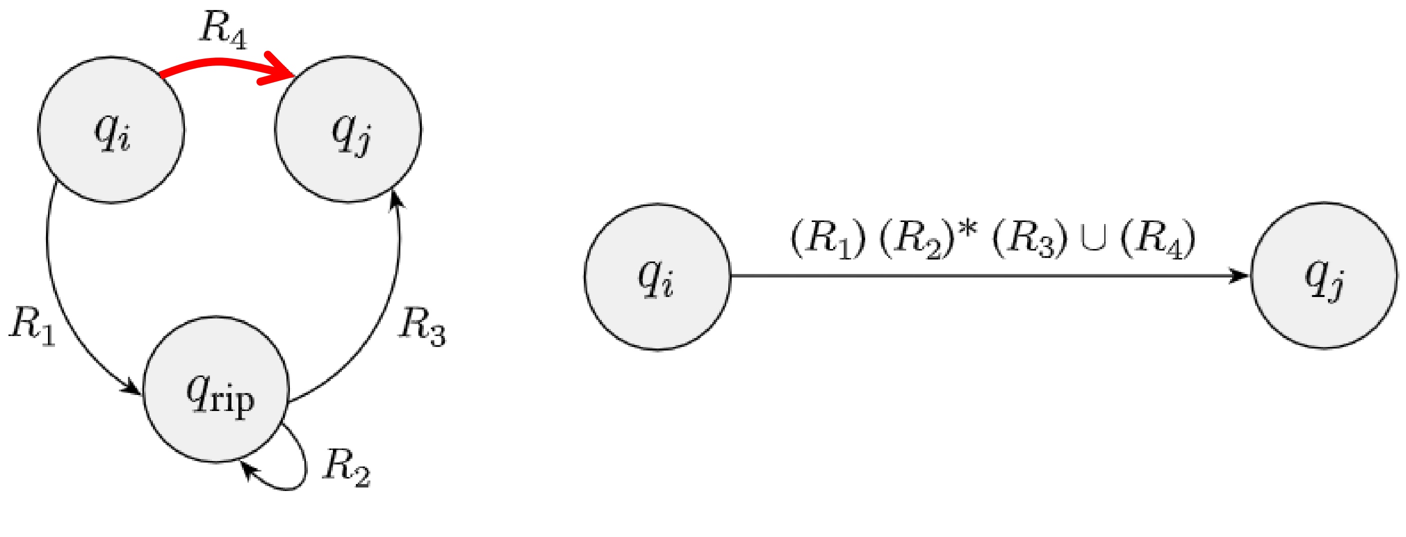
- There are two paths from q_i to q_j .
  - **Path 1 :** q_i, q_{rip}, and q_j.



Converting a DFA into a GNFA

- How can we rip a state out of the machine and repairing the remainder so that
- the same language is still recognized ?

- There are two path from q_i to q_j .
  - **Path 1 :** q_i, q_{rip}, and q_j.
  - **Path 2 :** q_i, and q_j.



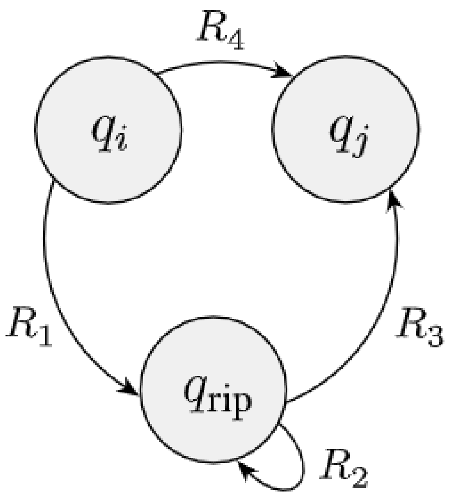
before
after

Chapter 1

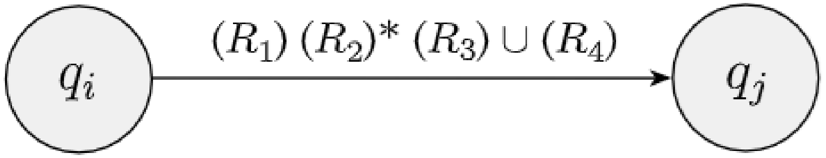
Converting a DFA into a GNFA

- How can we rip a state out of the machine and repairing the remainder so that
- the same language is still recognized ?

- We make this change for each arrow going from any state q_i to any state q_j, including the case where $q_i = q_j$.
- The new machine recognizes the original language.



before



after

Chapter 1

EQUIVALENCE WITH FINITE AUTOMATA

- Part 2: A language is regular, then some regular expression describes it.

- Proof:
  - Definition of GNFA

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

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

The symbol R is the collection of all regular expressions over the alphabet Σ

EQUIVALENCE WITH FINITE AUTOMATA

- Part 2: A language is regular, then some regular expression describes it.

- Proof (cont.):
  - A GNFA accepts a string w in Σ^*
    - If $w = w_1 w_2 \cdots w_k$, where each w_i is in Σ^* and a sequence of states $q_0, q_1, \ldots, q_k$ exists such that

1. $q_0 = q_{\text{start}}$ is the start state,
2. $q_k = q_{\text{accept}}$ is the accept state, and
3. for each i, we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$; in other words, R_i is the expression on the arrow from q_{i-1} to q_i.

EQUIVALENCE WITH FINITE AUTOMATA

- Part 2: A language is regular, then some regular expression describes it.

- Proof (cont.):
  - we let **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**.

  - We use the procedure CONVERT(G), which takes a GNFA and returns an equivalent regular expression.

Chapter 1

EQUIVALENCE WITH FINITE AUTOMATA

- Part 2: A language is regular, then some regular expression describes it.

- Proof (cont.):

CONVERT(G):

1. Let k be the number of states of G.

EQUIVALENCE WITH FINITE AUTOMATA

- Part 2: A language is regular, then some regular expression describes it.

- Proof (cont.):

CONVERT(G):

1. Let k be the number of states of G.
2. 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.

EQUIVALENCE WITH FINITE AUTOMATA

- Part 2: A language is regular, then some regular expression describes it.

- Proof (cont.):

CONVERT(G):

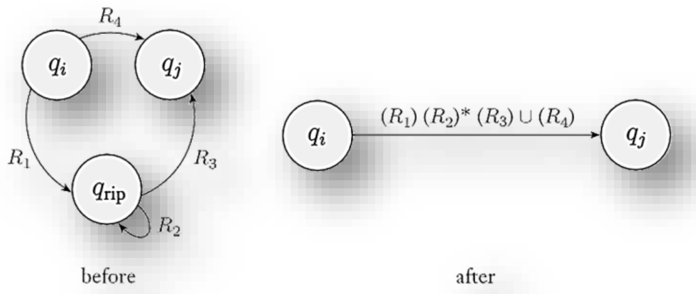
1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R.
   Return the expression R.
3. If $k > 2$, we select any state $q_{\text{rip}} \in Q$ different from q_{start} and q_{accept}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

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

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

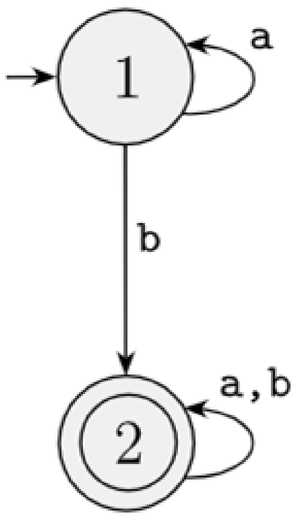
$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

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



Converting a DFA into a GNFA

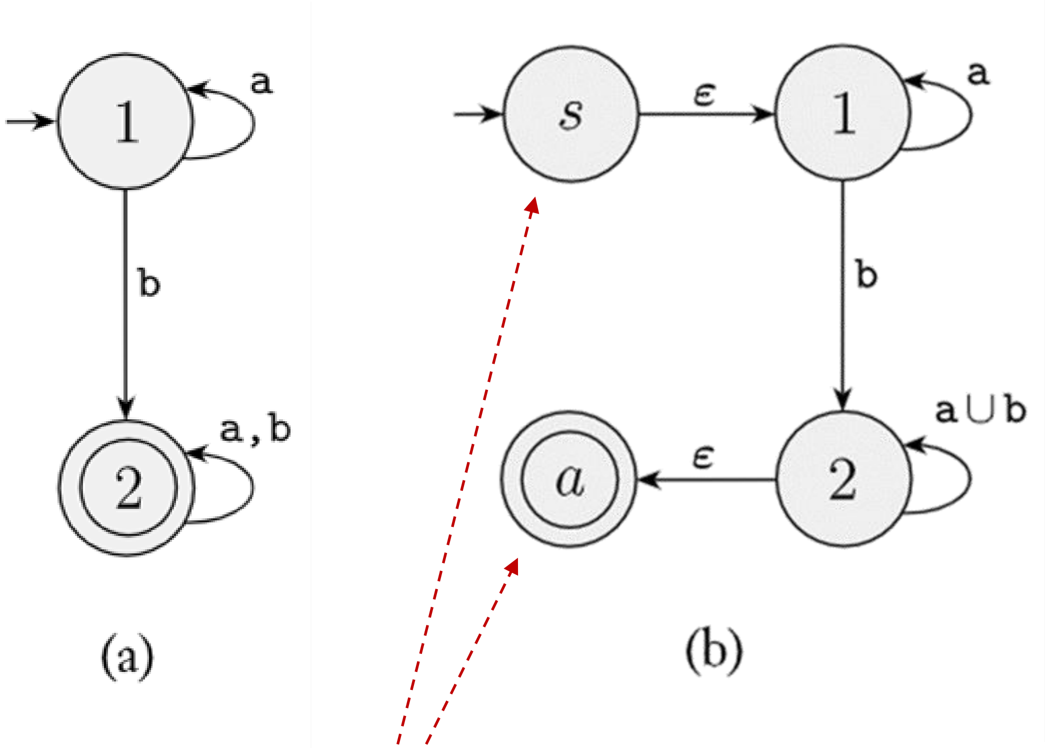
- Example : Convert the following DFA to a regular expression.



(a)

Converting a DFA into a GNFA

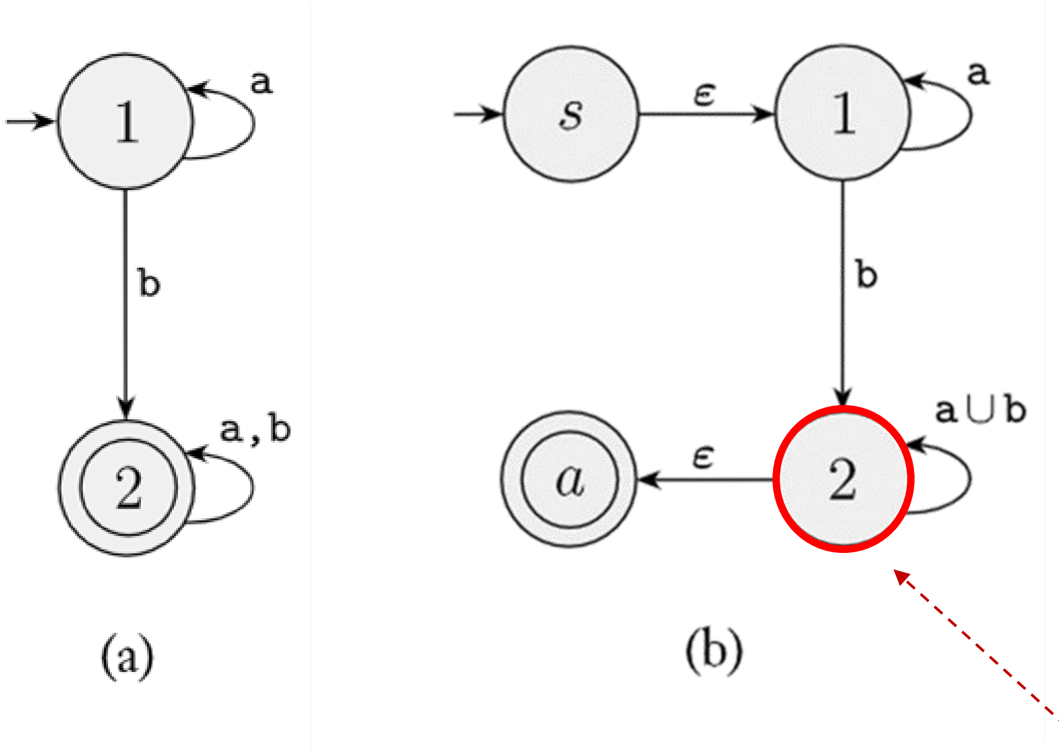
- Example : Convert the following DFA to a regular expression.



Adding two states

Converting a DFA into a GNFA

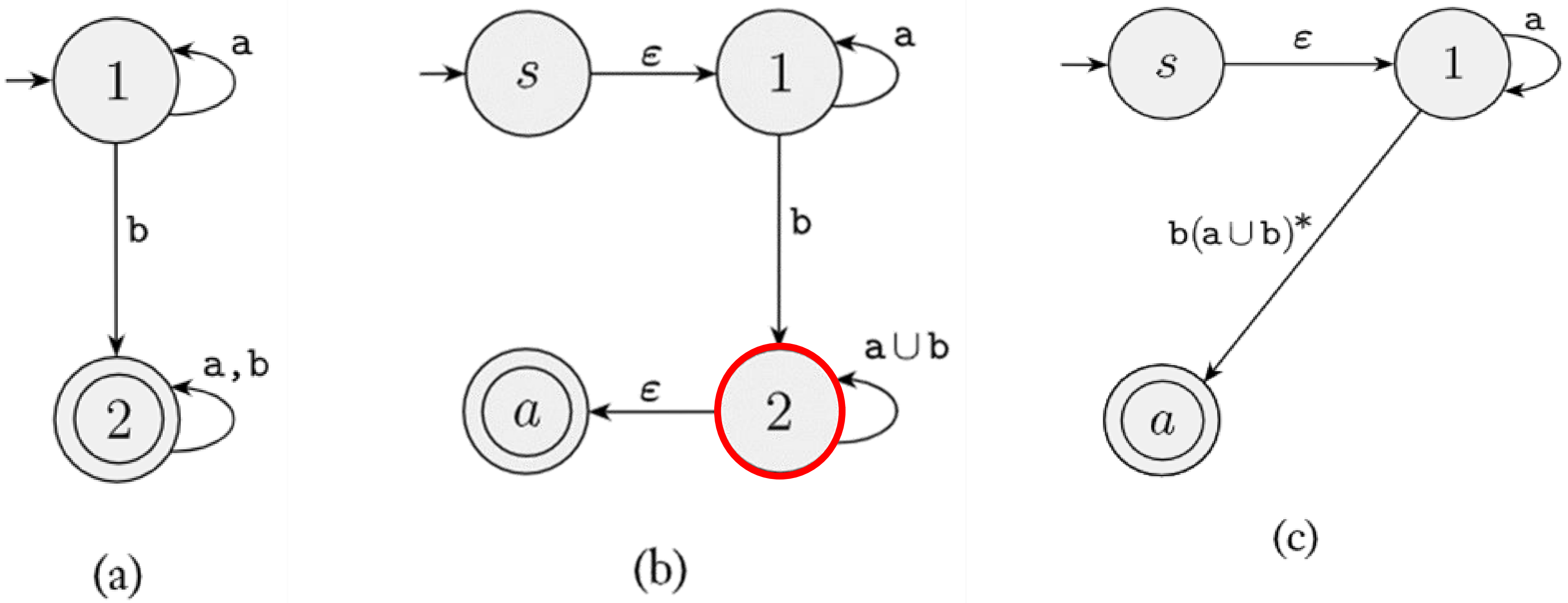
- Example : Convert the following DFA to a regular expression.



we remove state2 and
update the remaining arrow labels

Converting a DFA into a GNFA

- Example : Convert the following DFA to a regular expression.



Converting a DFA into a GNFA

- Example : Convert the following DFA to a regular expression.