Regular Expressions

- Formal Definition
- Equivalence with Finite Automaton
- Generalized nondeterministic finite automaton

Fatama Binta Rafiq

Lecturer

Software Engineering Department

Daffodil International University

Regular Expression

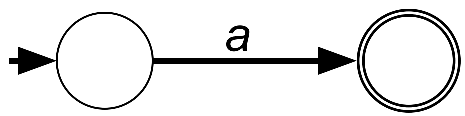
- Regular expression describes languages.
- Regular expression can be build up using regular operations.
- Precedence order: $* \cdot \cup$
- Example:
  - $(0 \cup 1)0^* = (\{0\} \cup \{1\}) \cdot \{0\}^* = \{0,1\} \cdot \{0\}^*$
    $A = \{w \mid w$ is a string starting with a 0 or a 1 followed by zero or more 0's$\}$
  - $(0 \cup 1)^* = (\{0\} \cup \{1\})^* = \{0,1\}^*$
    $A = \{$all possible string with 0s and/or 1s$\}$.

Formal Definition of Regular Expression

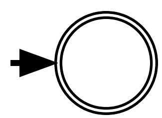
- ***R*** is a regular expression if ***R*** is –
  - a for some $a \in \Sigma$, represents the language $\{a\}$.
  - ε, represents the language $\{\varepsilon\}$ containing a single string, namely, the empty string.
  - φ, represents the empty language that doesn't contain any string. $L(\varphi^*) = \{\varepsilon\}$.
  - $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
    - $R \cup \varphi = R$, but $R \cup \varepsilon$ may not be equal to R.
  - $(R_1 \cdot R_2)$, where R_1 and R_2 are regular expressions,
    - $R \cdot \varepsilon = R$, but $R \cdot \varphi$ may not be equal to R.
  - $(R_1{}^*)$, where R_1 is a regular expressions,

Equivalence with finite automata

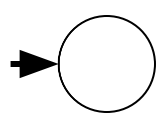
- Let convert regular language *R* into an NFA considering the six cases in the formal definition of regular language.
  - $R = a$, $a \in \Sigma$. Then $L(R)=\{a\}$, and the NFA that recognizes $L(R)$ is –



  - $R = \varepsilon$. Then $L(R)=\{\varepsilon\}$, and the NFA that recognizes $L(R)$ is –

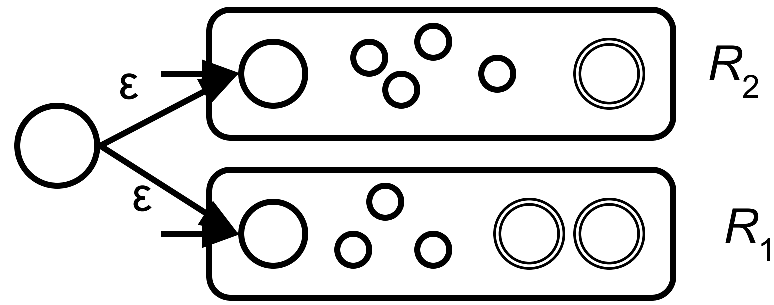


  - $R = \varphi$. Then $L(R)= \varphi$, and the NFA that recognizes $L(R)$ is –

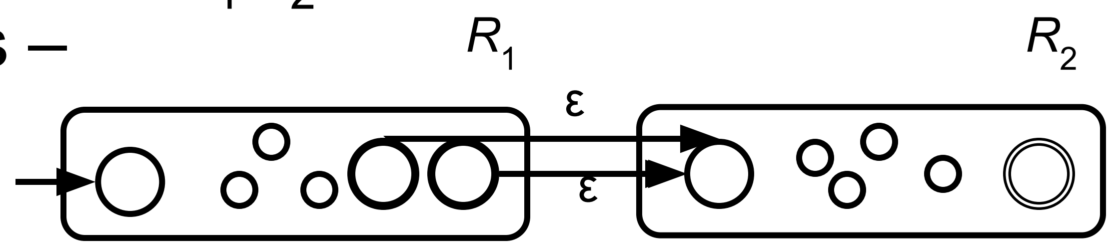


Equivalence with finite automata

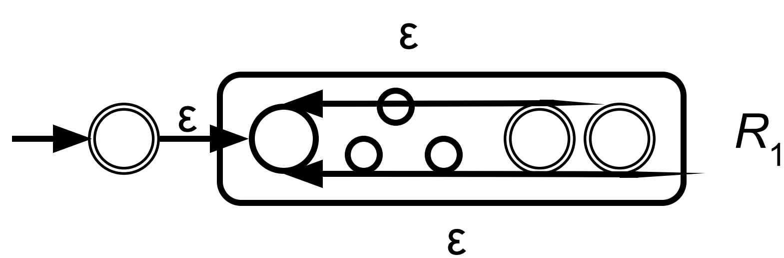
- $R = R_1 \cup R_2$. Then $L(R)=\{R_1,R_2\}$, and the NFA that recognizes $L(R)$ is –



- $R = R_1 \cdot R_2$. Then $L(R)=\{R_1 R_2\}$, and the NFA that recognizes $L(R)$ is –

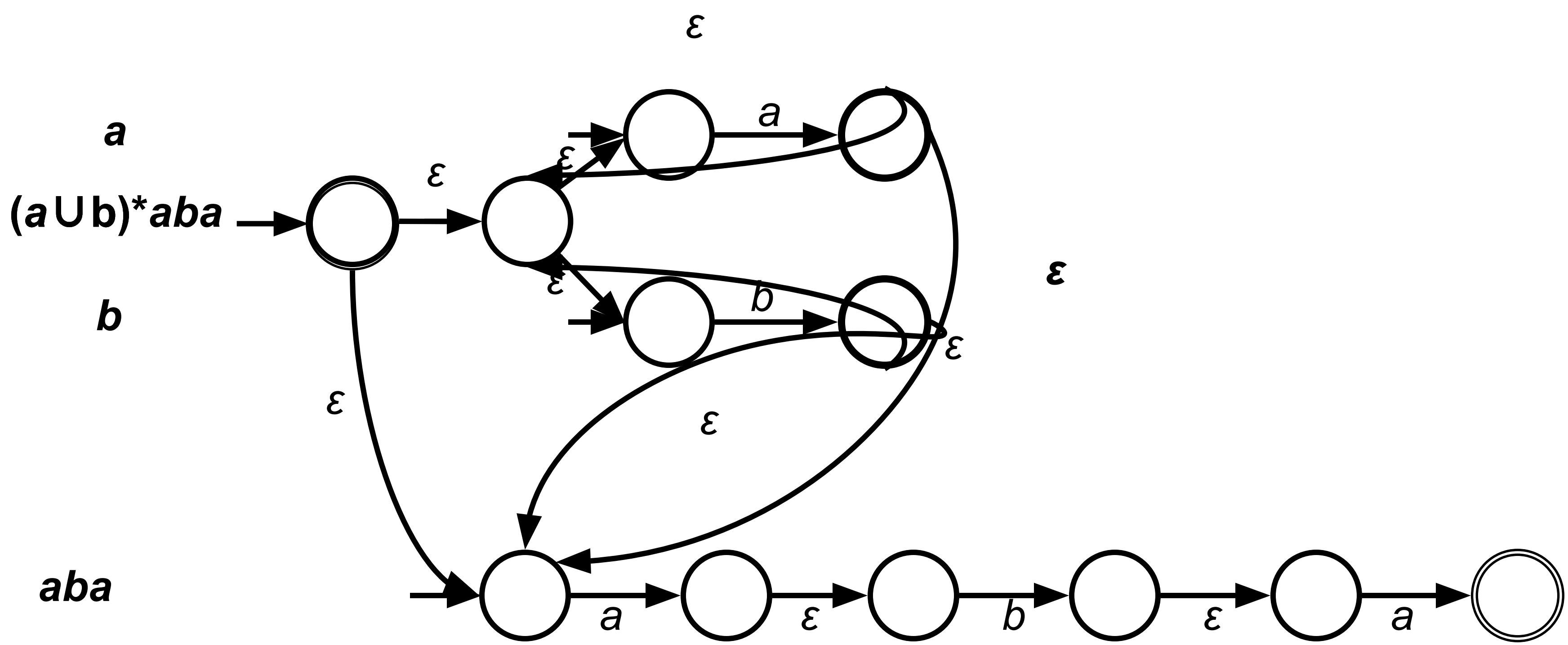


- $R = R_1{}^*$. Then $L(R)=\{R_1\}^*$, and the NFA that recognizes $L(R)$ is –



.

Converting a regular expression to an NFA

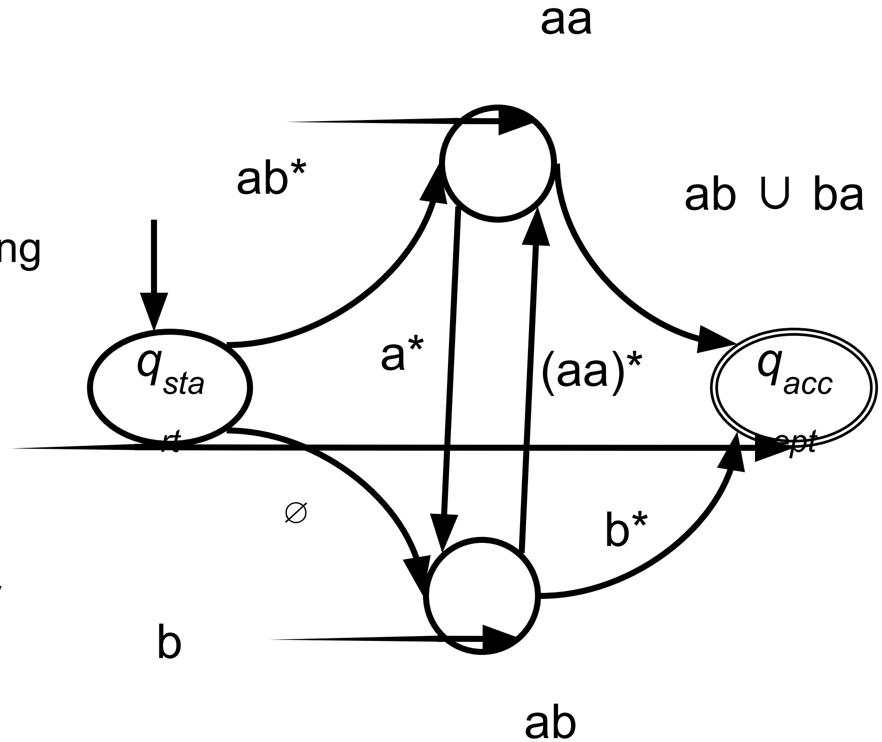


Building an NFA from regular expression: $(a \cup b)^*aba$

Converting a DFA to a regular expression

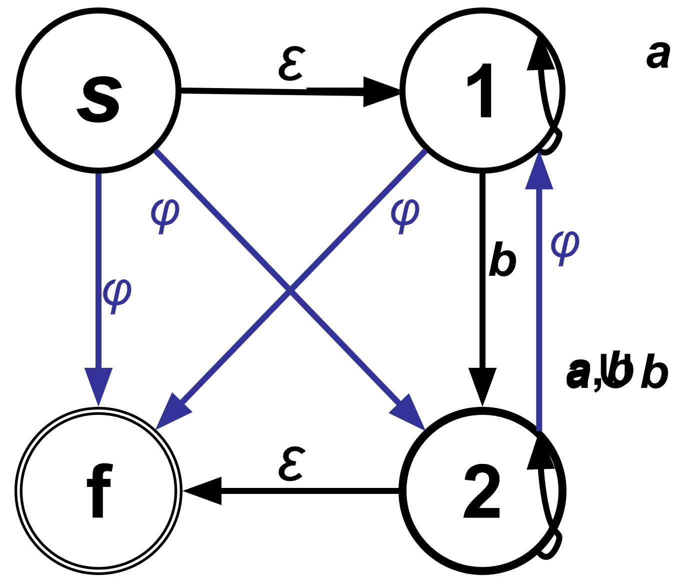
- This can be done in two parts. For this we introduce a new type of finite automata called ***generalized nondeterministic automaton***, GNFA.
  - First we will convert a DFA to GNFA, and
  - then GNFA to regular expression.

- GNFA has the following special form –
  - Transition labels might be in regular expression form.
  - The start state doesn't have any incoming arrow from any other state.
  - There is only one accept state, and it doesn't have any outgoing arrow to any other state.
  - Start state is never the same as accept state.
  - There is only one outgoing arrow to any other state and to itself, except the start and accept states. We will consider φ labeled outgoing arrows, if no transition exists between any two states.



Converting a DFA to GNFA

- Add a new start state with an ε arrow to the old start state.
- Add new accept state with ε arrows from the old accept states.
- If any arrows have multiple labels, union the previous labels into one label.
- Add arrows with φ label between states where there are no arrows. This won't change the language as φ label arrows can never be used.
  - Even we might ignore adding such arrows, as these are arrows which can be assumed to be there with no use.



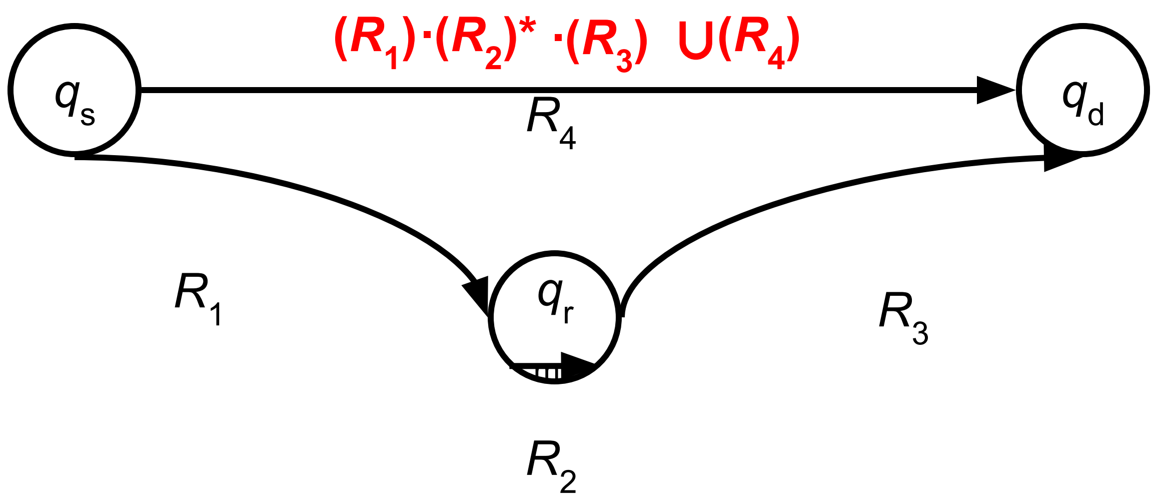
Converting a GNFA to a regular expression

- Let consider the GNFA to be with k states.
- We will continuously remove one state from the GNFA until $k = 2$. These last two states are actually the start and the accept states.
- We do so by selecting a state, ripping it out of the machine, and ***repairing*** the remainder so that the same language is still recognized.
- Any state will do, provided that the state is not the start or the accept states.

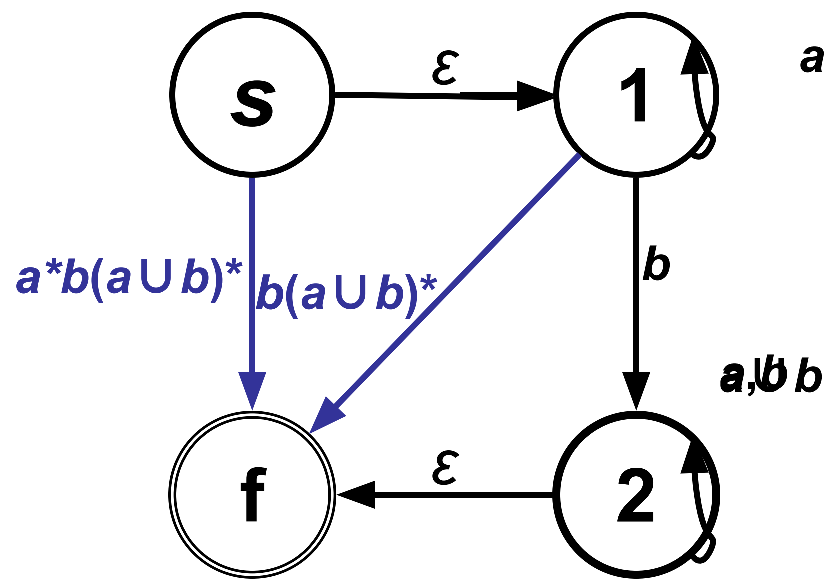
Repairing after removing a state

- Let call the removed state q_{rmv}.
- Repair the machine by altering the regular expressions that label each of the remaining arrows. This change is done for each arrow going from any state q_s to q_d, including the case where $q_s = q_d$.
- The new labels compensate for the absence of q_{rmv} by adding back the lost computations. i.e., The new label going from a state q_s to state q_d is a regular expression that describes all strings that would take the machine from q_s to q_d either directly or via q_{rmv}.

.

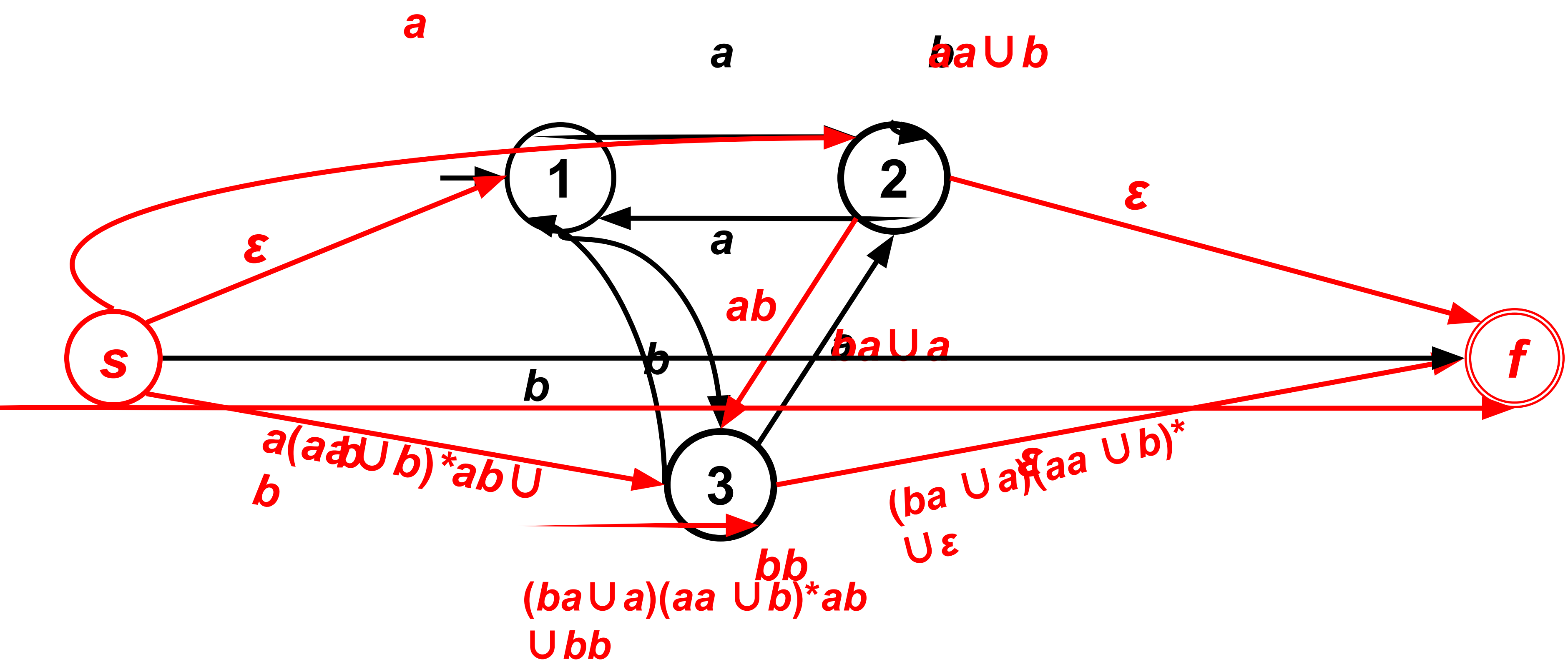


Example



Converting a two state DFA to an equivalent regular expression

Example



(a(aa∪b)*ab∪b)((ba∪a)(aa ∪b)*ab ∪bb)*((ba ∪a)(aa ∪b)* ∪ε) ∪ (a(aa∪b)*)

Converting a three state DFA to an equivalent regular expression

Algorithm Convert(G)

Formally: Add q_{start} and q_{accept} to create G

Where $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$

Run CONVERT(G):

If #states = 2

return the labeled expression from q_{start} to q_{accept}

Else If #states > 2

select $q_{rip} \in Q$ different from q_{start} and q_{accept}

Let G' be the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$, where

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

δ': for any $q_i \in Q' - \{q_{accept}\}$ and $q_j \in Q' - \{q_{accept}\}$

$\delta'(q_i, q_j) = \delta(q_i, q_{rip})\delta(q_{rip}, q_{rip})^*\delta(q_{rip}, q_j) \cup \delta(q_i, q_j)$

return CONVERT(G')

REGULAR LANGUAGES ARE COLSED UNDER REGULAR OPERATIONS

- **Union:** $A \cup B = \{ w \mid w \in A \text{ or } w \in B \}$
- **Intersection:** $A \cap B = \{ w \mid w \in A \text{ and } w \in B \}$
- **Reverse:** $A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \}$
- **Negation:** $\neg A = \{ w \mid w \notin A \}$
- **Concatenation:** $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$
- **Star:** $A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \}$