Regular Expressions

Mohammad Hasan

CSE, CUET

Introduction

- *Regular expressions* are an algebraic way to describe languages.
- They describe exactly the regular languages.
- If E is a regular expression, then L(E) is the language it defines.

- **Application**: text-search, compiler design, Utilities (AWK, GREP in UNIX), modern programming languages (PERL), and text editors all provide mechanisms for the description of patterns using RE.

Introduction

- **(5+3) x 4 [arithmetic expression]**
- **(0 U 1) * 1 [Regular expression]**
- **RE offer something that automata do not:**

- **A declarative way to express the strings we want to accept. Thus, RE serve as the input language for many systems that process strings.**

Examples

1. Search commands such as the UNIX *grep* or equivalent commands for finding strings that one sees in Web browsers or text-formatting systems.

- These systems use a RE like notation for describing patterns that the user wants to find in a file.

2. Lexical-analyzer generators, such as **Lex/Flex**.

- A generator accepts descriptions of the forms of tokens, which are essentially REs, and produces a DFA that recognizes which token appears next on the input

RE: Definition

- R is a **regular expression** if R is

1. a for some *a* in the alphabet Σ
2. $\in$
3. $\varnothing$
4. **$(R_1 \text{ U } R_2)$**, where R_1 & R_2 are RE
5. **$(R_1 \text{ o } R_2)$,** where R_1 & R_2 are RE
6. **$(R_1{*})$**, where R_1 is RE

RE: Definition

- Basis 1: If a is any symbol, then **a** is a RE, and L(**a**) = {a}.
  - Note: {a} is the language containing one string, and that string is of length 1.
- Basis 2: ϵ is a RE, and L(ϵ) = {ϵ}.
- Basis 3: ∅ is a RE, and L(∅) = ∅.

RE: Definition

- Induction 1: If $\mathbf{E_1}$ and $\mathbf{E_2}$ are REs, then $\mathbf{E_1+E_2}$ is a RE, and $\mathbf{L(E_1+E_2) = L(E_1) \cup L(E_2)}$.

- Induction 2: If $\mathbf{E_1}$ and $\mathbf{E_2}$ are REs, then $\mathbf{E_1E_2}$ is a RE, and $\mathbf{L(E_1E_2) = L(E_1)L(E_2)}$.

Concatenation **: the set of strings wx such that w is in $\mathbf{L(E_1)}$ and x is in $\mathbf{L(E_2)}$.**

RE: Definition

- Induction 3: If **E** is a RE, then **E*** is a RE, and **L(E*) = (L(E))***.

Closure, or "Kleene closure" = set of strings $w_1w_2 \ldots w_n$, for some $n \geq 0$, where each w_i is in L(E).
Note: when n=0, the string is ϵ.

Precedence of Operators

- **Parentheses** may be used wherever needed to influence the grouping of operators.
- Order of precedence is *** (highest)**, then concatenation, then **+ (lowest)**.

Examples: RE's

- L(**01**) = {01}.
- L(**01**+**0**) = {01, 0}.
- L(**0**(**1**+**0**)) = {01, 00}.
  - Note order of precedence of operators.
- L(**0***) = {ϵ, 0, 00, 000,... }.
- L((**0**+**10**)*(ϵ+**1**)) = all strings of 0's and 1's without two consecutive 1's.

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

1. $0^*10^* = \{w|w$ has exactly a single $1\}$
2. **$\Sigma^*1\Sigma^* = \{w|w$ has at least one $1\}$**
3. **$\Sigma^*001\Sigma^* = \{w|w$ contains the strings 001 as a substring$\}$**
4. **$(\Sigma\Sigma)^* = \{w|w$ is a string of even length$\}$**

-the length of a string is the number of symbols that it contains

5. $(\Sigma\Sigma\Sigma)^* = \{w|$ the length of w is a multiple of three$\}$

6. $01 \cup 10 = \{01, 10\}$

7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w|w$ starts & ends with the same symbol$\}$

8. $(0 \cup \epsilon)1^* = 01^* \cup 1^*$ the expression $0 \cup \epsilon$ describes the language $\{0, \epsilon\}$, so the concatenation operation adds either 0 or ϵ before every string in 1^*

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

9. $(0 \cup \in)(1 \cup \in) = \{\in, 0, 1, 01\}$
10. $1 ^* \varnothing = \varnothing$ Concatenating the empty set to any set yields the empty set
11. $\varnothing^* = \{\in\}$

The Star operation puts together any number of strings from the language to get a string in the result. If the language is empty, the star operation can put together 0 strings, giving only the empty string.

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

- **$R \cup \varnothing = R$**: adding the empty language to any other language will not change it
- **$R \circ \in = R$**: adding the empty string to any string will not change it
- **$R \cup \in \neq R$**

If $R = 0$, then $L(R) = \{0\}$ but $L(R \cup \in) = \{0, \in\}$

- **$R \circ \varnothing \neq R$**

If $R = 0$, the $L(R) = \{0\}$ but $L(R \circ \varnothing) = \varnothing$

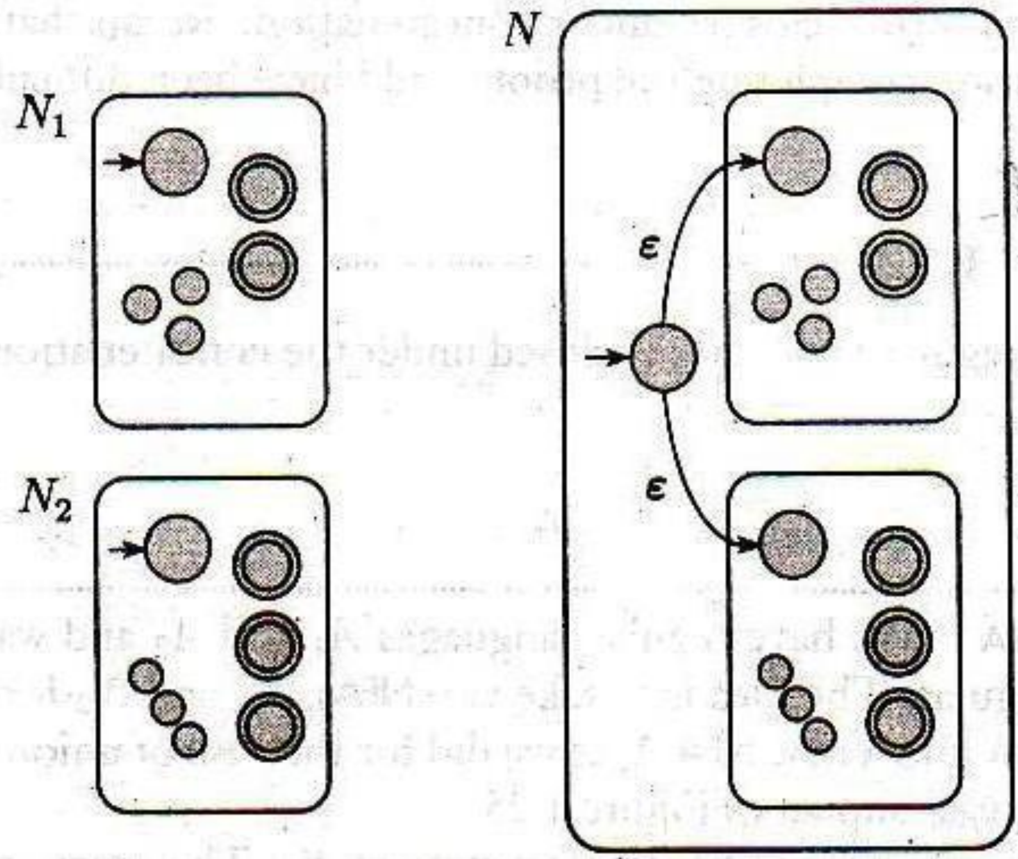
Importance

- REs are useful tools in the design of compilers for programming languages.
- Elemental objects in a programming language, called *tokens*, such as the variable names and constants, may be described with RE.
- A numerical constant that may include a fractional part and/or a sign may be described as a member of the language
- **{+, -, ∈} {D D* . U D D* . D* U D* . D D*}**
- Where, D = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- **Expressions**: **72, 3.14159, + 7. , and -.01**
- **Once the syntax of the tokens have been described with the REs, automatic systems can generate the lexical analyzer, the part of a compiler that initially processes the input program.**

Theorem 1: The class of regular languages is closed under the union operation

- **Proof Idea:**
- Regular languages A_1 and A_2
- Prove that A_1 U A_2 is regular
- Take two NFAs N_1 & N_2 for A_1 & A_2 and combined them into one new NFA, N
- Machine N must accept its input if either N_1 or N_2 accepts this input
- The new machine has a new state that branches to the start states of the old machines with $\in$arrows

Construction of NFA N to recognize $A_1 \cup A_2$



Proof

- Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1,
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2,

Construct $N = (Q, \Sigma, \delta, q_0, F)$ recognize $\mathbf{A_1 \cup A_2}$

1. $\mathbf{Q = \{q_0\} \cup Q_1 \cup Q_2}$

The states of N are all the states of N_1 & N_2, with the addition of a new start state q_0.

2. The state q_0 is the start state of N.

Proof

3. The accept states $F = F_1 \cup F_2$.

The accept states of N are all the accept states of N_1 & N_2. That way N accepts if either N_1 accepts or N_2 accepts.

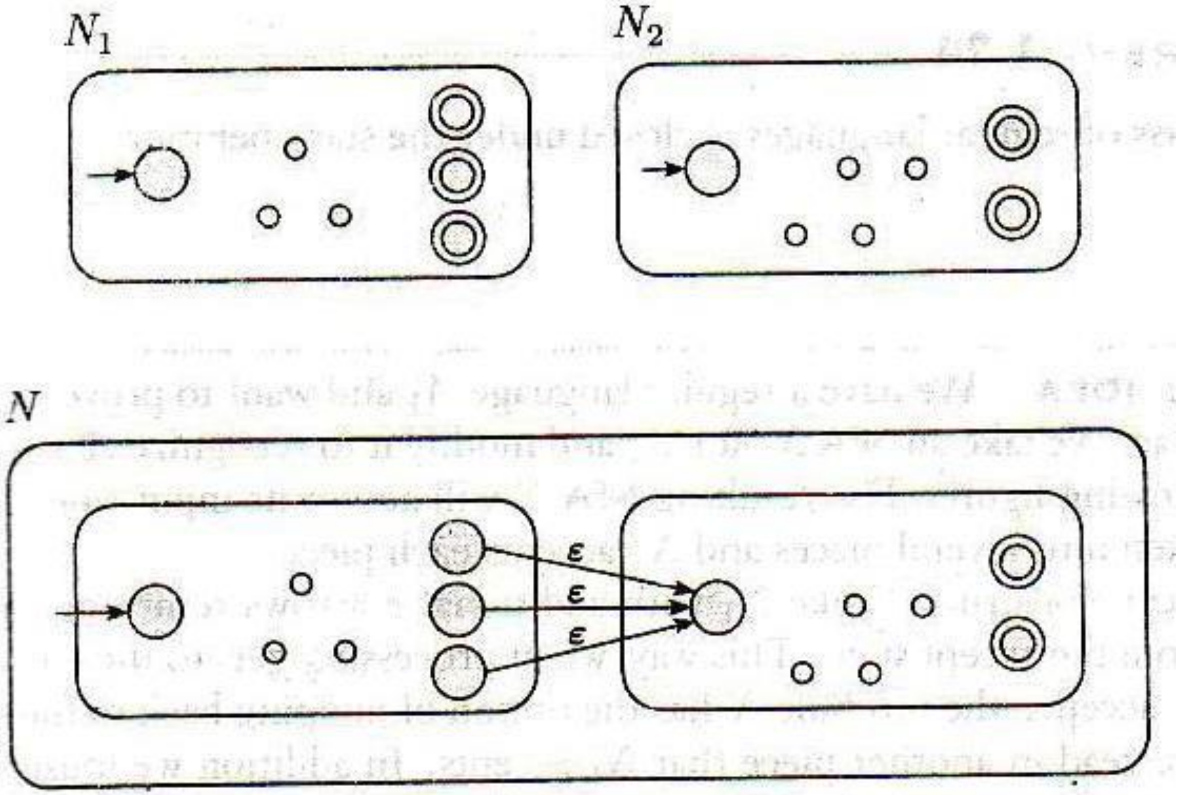
4. Define δ so that for any $q \in Q$ & any $a \in \Sigma_\varepsilon$

$$\boxed{\delta(q,a) = \begin{cases} \delta_1(q,a) & \mathrm{q} \in \mathrm{Q}_1 \\ \delta_2(q,a) & \mathrm{q} \in \mathrm{Q}_2 \\ \{q_1, q_2\} & \mathrm{q} = \mathrm{q}_0 \,\mathrm{and}\, \mathrm{a} = \varepsilon \\ \phi & \mathrm{q} = \mathrm{q}_0 \,\mathrm{and}\, \mathrm{a} \neq \varepsilon \end{cases}}$$

Theorem 2: The class of regular languages is closed under the concatenation operation

- Proof Idea:
- Regular languages A_1 and A_2
- Prove that A_1 o A_2 is regular
- Take two NFAs, N_1 & N_2 for A_1 & A_2 and combined them into a new NFA, N
- Assign N's start state to be the state of N_1
- The accept states of N_1 have additional ε arrows that allow branching to N_2 whenever N_1 is in an accept state, signifying that it has found an initial piece of the input that constitutes a string in A_1.

- **The accept states of N are the accept states of N_2 only.**
- **Therefore, it accepts when the input can be split into two parts, the first accepted by N_1 and the second by N_2.**



Proof

- Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1,
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2,

Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ recognize $\mathbf{A_1 o A_2}$

1. $Q = Q_1 \cup Q_2$

The states of N are all the states of N_1 & N_2,

2. The state q_1 is the same as the start state of N_1.

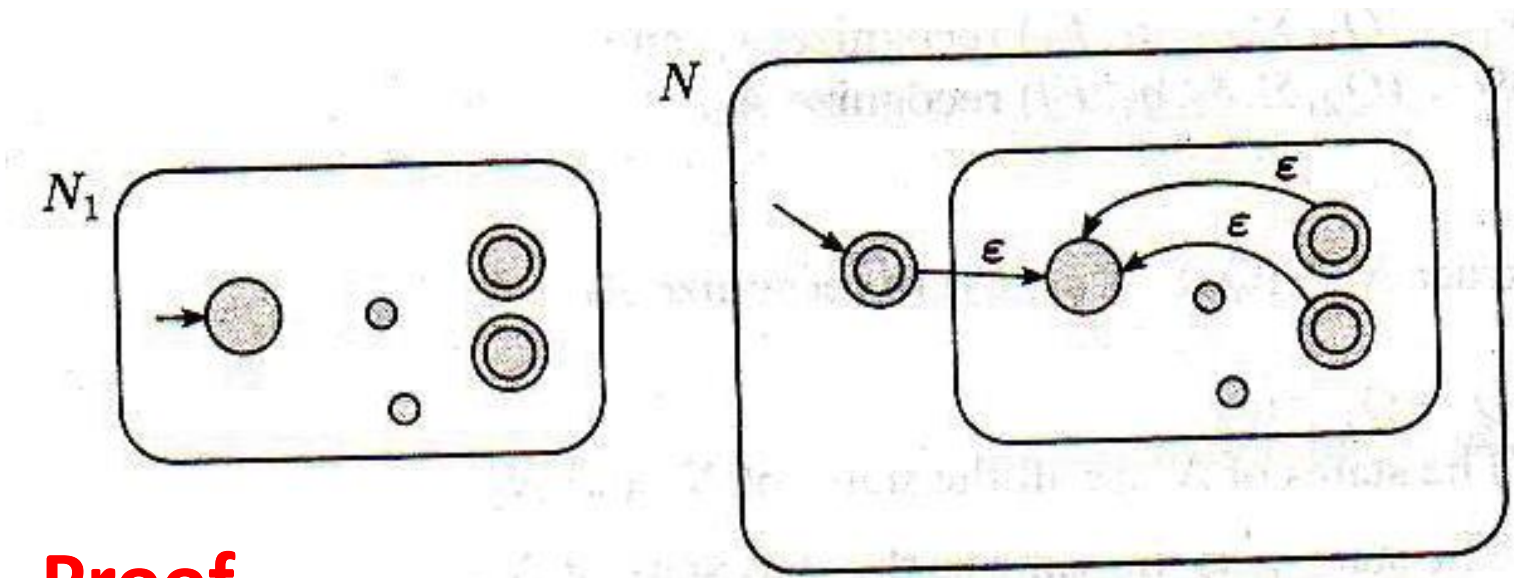
3. The accept states F_2 are the same as the accept state of N_2.

4. Define δ so that for any $q \in Q$ & any $a \in \Sigma_\varepsilon$

$$\boxed{\delta(q,a) = \begin{cases} \delta_1(q,a) & \mathrm{q} \in \mathrm{Q}_1 \text{ and } \mathrm{q} \notin \mathrm{F}_1 \\ \delta_1(q,a) & \mathrm{q} \in \mathrm{F}_1 \text{ and } \mathrm{a} \neq \varepsilon \\ \delta_1(q,a) \mathrm{U} \{q_2\} & \mathrm{q} \in \mathrm{F}_1 \text{ and } \mathrm{a} = \varepsilon \\ \delta_2(q,a) & \mathrm{q} \in \mathrm{Q}_2 \end{cases}}$$

Theorem 3: The class of regular languages is closed under the star operation

- **Proof Idea**:
- Regular languages A_1
- Prove that A_1^* also is regular
- Take an NFA, N for A_1 and modify it to recognize A_1^*
- Resulting NFA N will accept its input whenever it can be broken into several pieces & N_1 accepts each piece.



Proof:

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1,

Construct $N = (Q, \Sigma, \delta, q_0, F)$ recognize $\mathbf{A_1}^*$

1. $Q = \{q_0\} \cup Q_1$

The states of N are the states of N_1 + a new state

2. The state q_0 is the new start state

3. $F = \{q_0\} \cup F_1$.

The accept states are the old accept states + the new start state

4. Define δ so that for any $q \in Q$ & any $a \in \Sigma_\varepsilon$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \mathrm{q} \in \mathrm{Q}_1 \text{ and } \mathrm{q} \notin \mathrm{F}_1 \\ \delta_1(q,a) & \mathrm{q} \in \mathrm{F}_1 \text{ and } \mathrm{a} \neq \varepsilon \\ \delta_1(q,a) \mathrm{U} \{q_1\} & \mathrm{q} \in \mathrm{F}_1 \text{ and } \mathrm{a} = \varepsilon \\ \{q_1\} & \mathrm{q} = q_0 \text{ and } \mathrm{a} = \varepsilon \\ \phi & \mathrm{q} = q_0 \text{ and } \mathrm{a} \neq \varepsilon \end{cases}$$

Equivalent with Finite Automata

- RE and FA are equivalent in their descriptive power.
- This fact is rather remarkable, because FA & RE superficially appear to be rather different.
- However, any RE can be converted into a FA that recognizes the language it describes, & vice-versa.
- Recall that a Regular language is one that is recognize by some FA

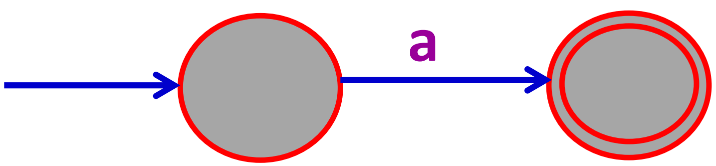
Theorem: A language is regular if and only if some regular expression describe it

- Two directions: 02 lemmas
- **Lemma 1**: if a language is described by a RE, then it is regular
- **Proof Idea**: Say that we have a RE R describing some language A.
- We show how to convert R into an NFA recognizing A
- *If an NFA recognizes A then A is regular.*

Proof

- Let's convert R into NFA N.
- **<u>Six cases:</u>**

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

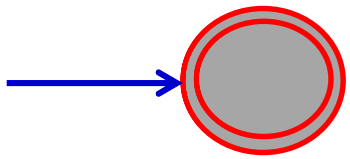


Note: this machine fits the definition of an NFA but not that of a DFA because it has some states with no exiting arrow for each possible input symbol.

Formally, **$N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, where we describe δ by saying that $\delta(q_1, a) = \{q_2\}$,**

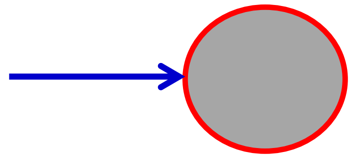
$\delta\ (r, b) = \varnothing$ for $r \neq q_1$ or $b \neq a$

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



Formally, **$N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$, where $\delta(r, b) = \varnothing$ for any r and b**

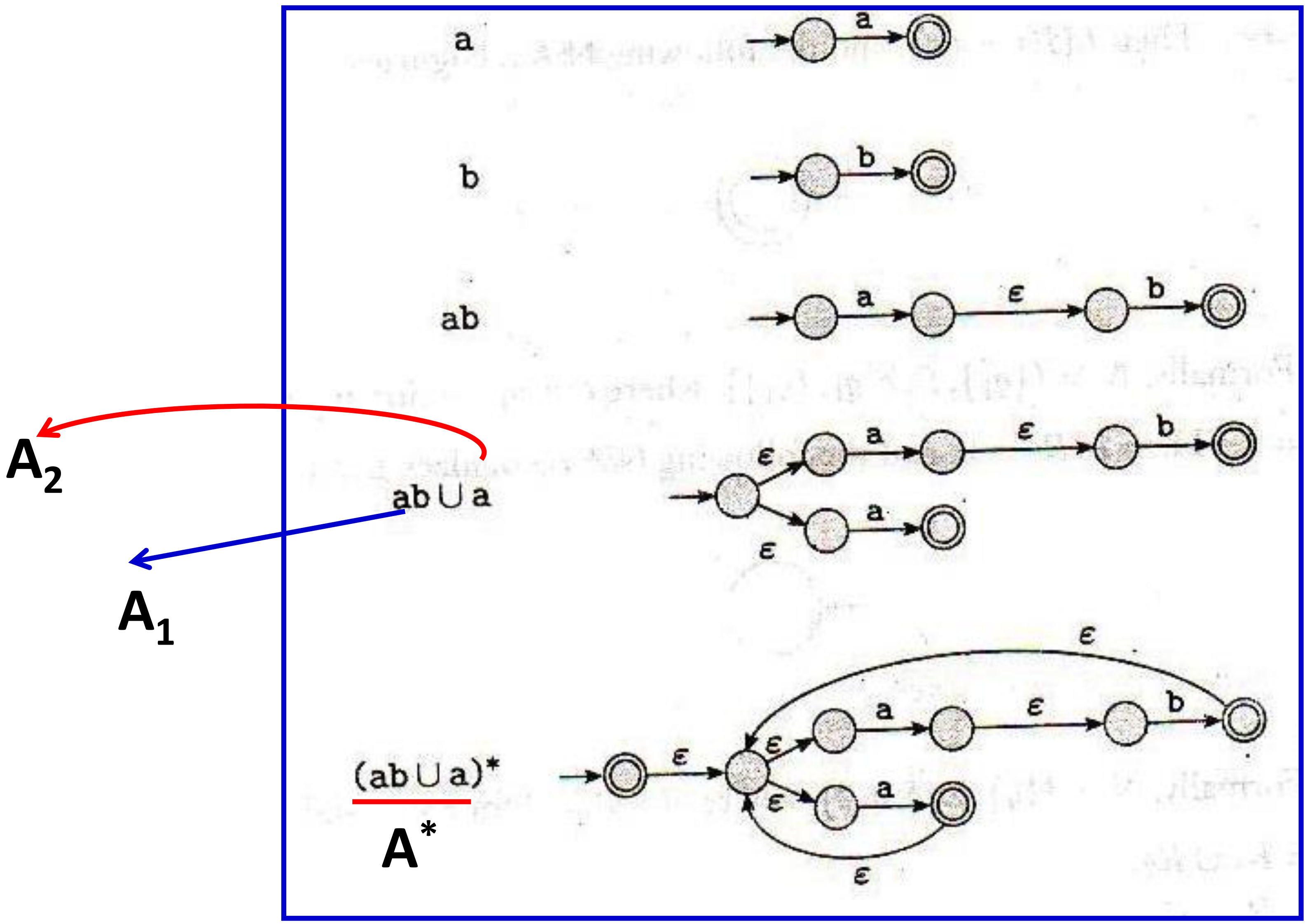
3. $R = \varnothing$. **Then the following NFA recognizes L (R)**



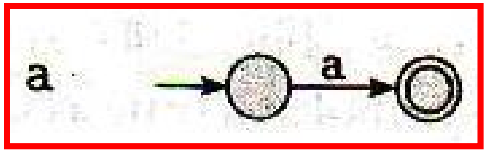
Formally, $N = (\{q\}, \Sigma, \delta, q, \{\varnothing\})$, for any r and b

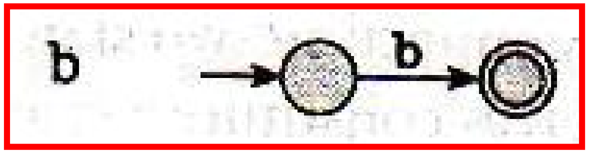
4. $R = R_1 \cup R_2$
5. $R = R_1 \circ R_2$
6. $R = R_1^*$.

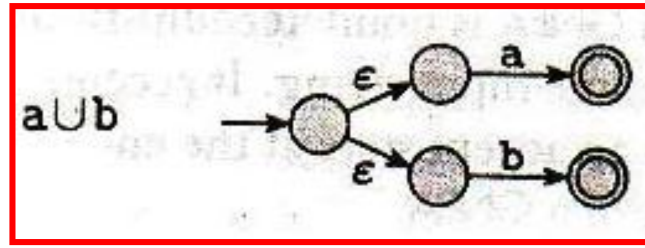
Convert RE (ab U a)* to NFA

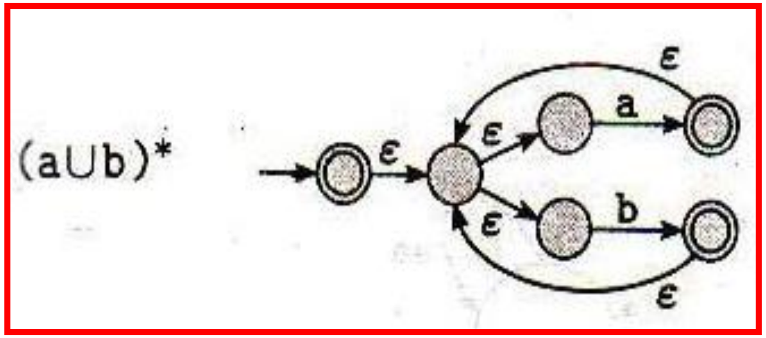


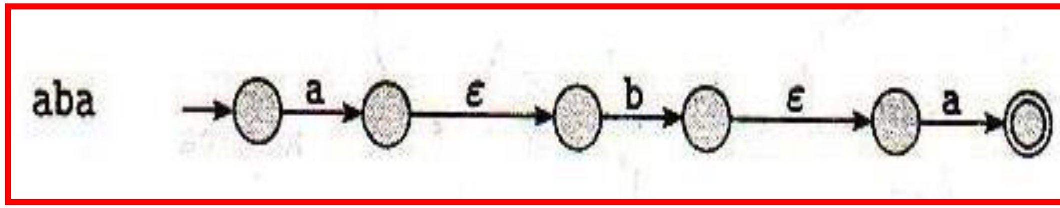
Convert **(a U b)*aba** to NFA

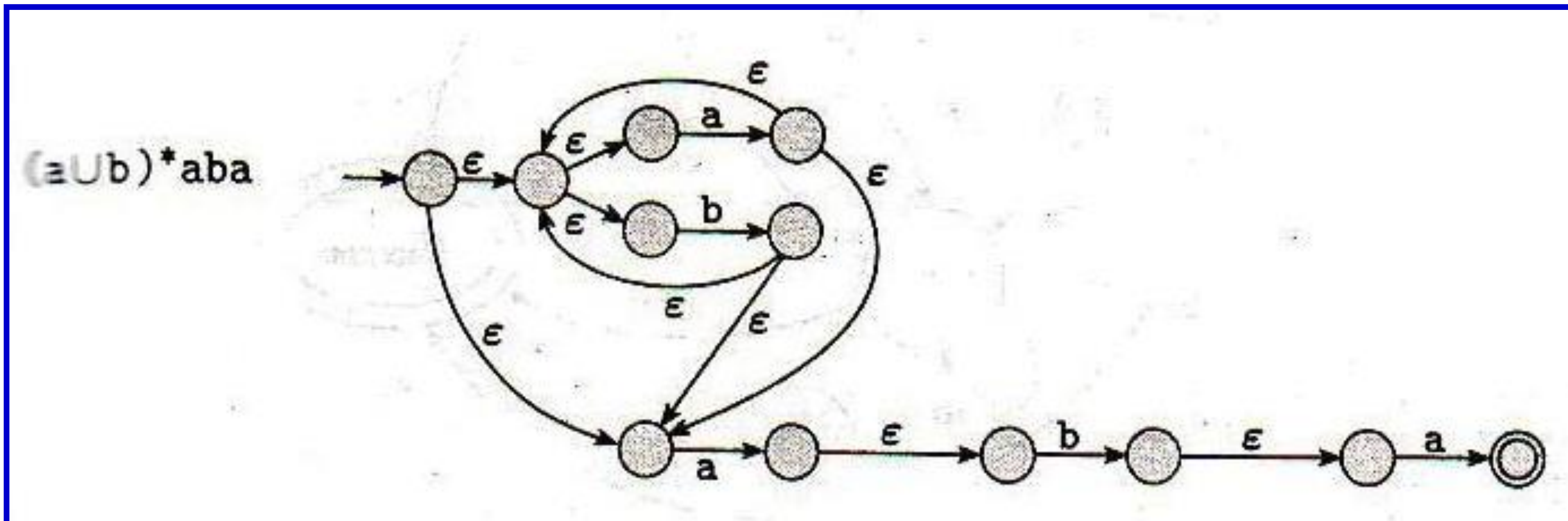




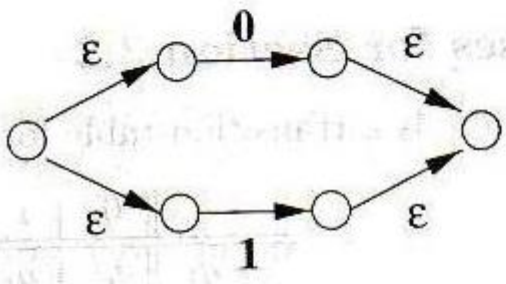




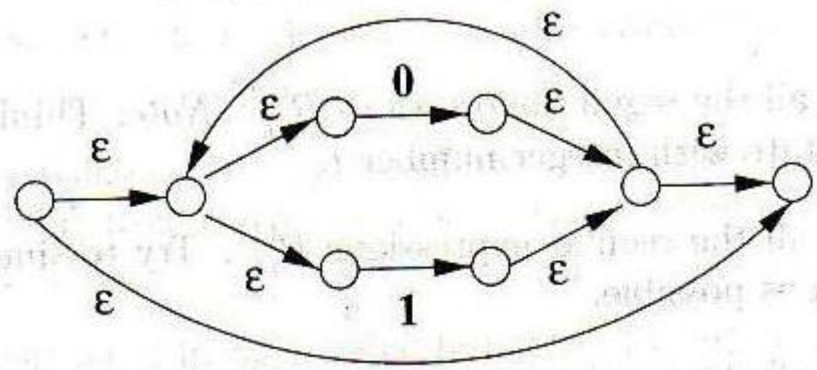




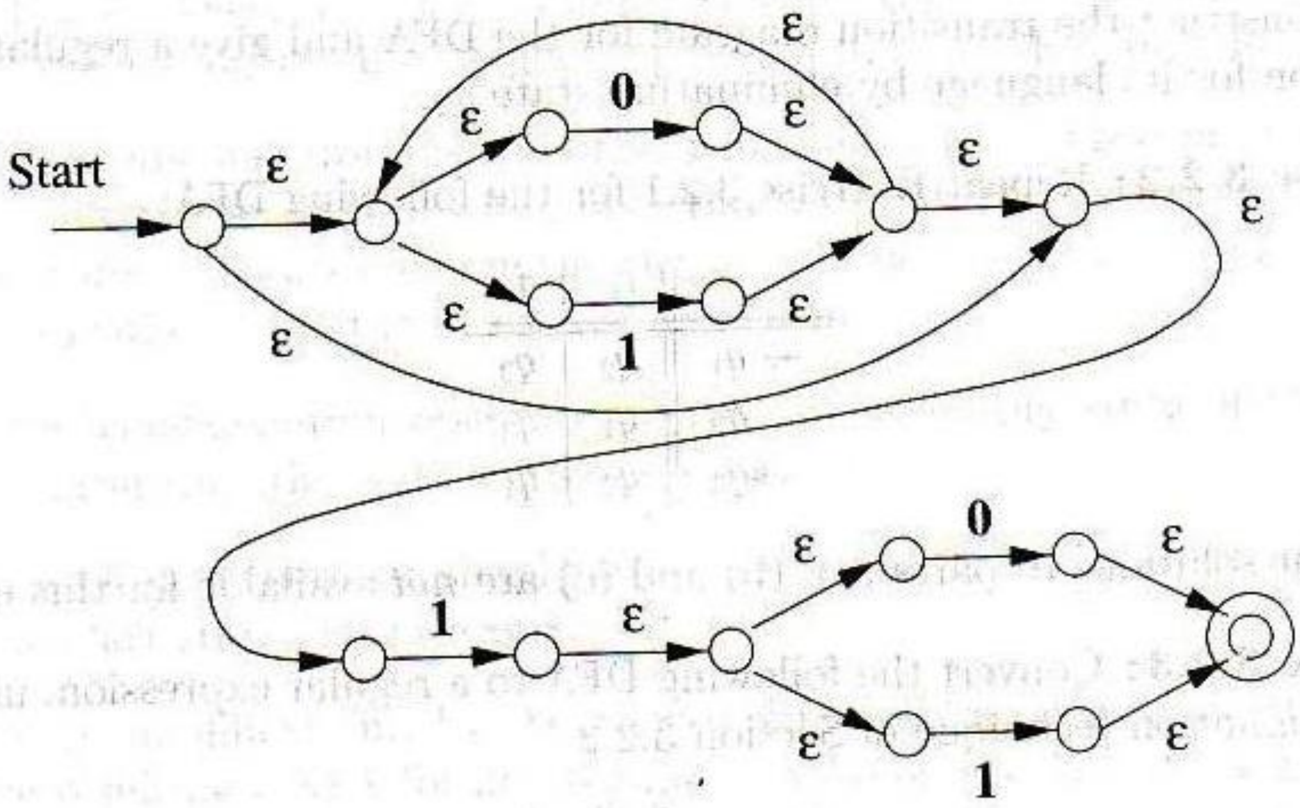
Convert **(0+1)* 1 (0+1)** to an ε-NFA



(a)



(b)



(c)

Assignments

- Convert the Following to NFA
- **1. $(0 \cup 1)^* \, 000 \, (0 \cup 1)^*$**
- **2. $a^* \cup b^*$**
- **3. aba U bab**
- **4. $a \, (ba)^* \, b$**
- **5. $(\varepsilon \cup a) \, b$**