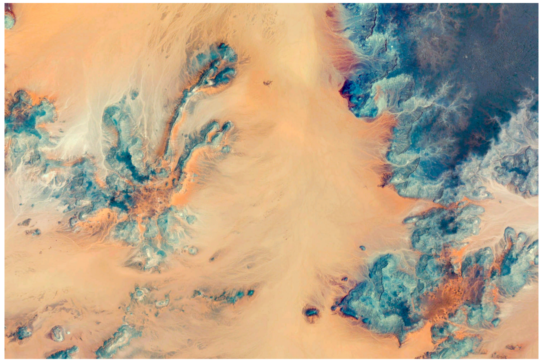
Chapter 2

# AUTOMATA THEORY-

## *For Mathematical Modeling*



# AUTOMATA THEORY from Mathematical Modeling View

We study the following topics in this chapter.

A. BASIC TERMS

Motivation and Key concepts

Determining a language - Regular Expressions

B. AUTOMATA

- Finite Automata, see Section 2.4
- Finite Automata- Basic Problems, in Section 2.5

B. C. Equivalence of DFA and NFA

- Equivalence of DFA and NFA (Nondeterministic automata)
- Determine a regular expression given an automaton, in Section 2.7.

**Knowledge blocks** include automata theory, formal languages and grammars;

bringing the theoretical foundation of computer science (CS).

**COURTESY**:

Some material of chapter 1 (Logic Theory) and 2 (Automat Theory) are borrowed from texts

1. REF. 1A: MICHAEL HUTH and MARK RYAN. LOGIC IN COMPUTER SCIENCE - Modelling and Reasoning about Systems, **Chapter 1,2,4** [2nd ed., Cambridge University Press 2004]

   REF. 1B: MICHAEL HUTH and MARK RYAN. Solutions Manual - LOGIC IN COMPUTER SCIENCE [2nd ed., Cambridge University Press 2004]

2. REF. 2: Peter Linz. *An Introduction to Formal Languages and Automata*, 6th Ed., Jones & Bartlett Learning (2017)
3. REF. 3: Mordechai Ben-Ari. *Mathematical Logic for Computer Science*, Chapter 15 Third Edition, Springer-Verlag (2012).

## 2.1 Overview - Motivation - Key concepts

1. **Computer science** a very diverse (with many important topics), but in spite of this diversity, there are some common underlying principles. To study these basic principles, we construct *abstract models* of computers and computation.
2. **Why study automata theory?** Automata theory provides such abstract model, its ideas have some important applications, both software (programming languages, and compilers ...), midleware (operating systems) to hardware (digital design).
3. An **automaton**[1]: a construct that possesses all the features of a digital computer, or any process. It

   accepts *input*, produces *output*, may have some *storage*, and

   can make decisions in transforming the input into the output, see Figure 2.1, in which we use notations below.

- ○ with label: describe states of a process/agent
- $\rightarrow$: transitions from a state to another one
- □: action of an agent (or multiple agents)

**Chapter's aims:** We will study various automata, mostly

abstract and mathematical concepts, to see how they are related to

(formal) *languages* and *grammars*.

### From Alphabets to Languages- KEY TERMINOLOGIES

We need basic definitions and their relations.

1. An **alphabet** $\Sigma$ is a finite and non-empty set of symbols (or characters).

[1] single **automaton**, plural **automata**

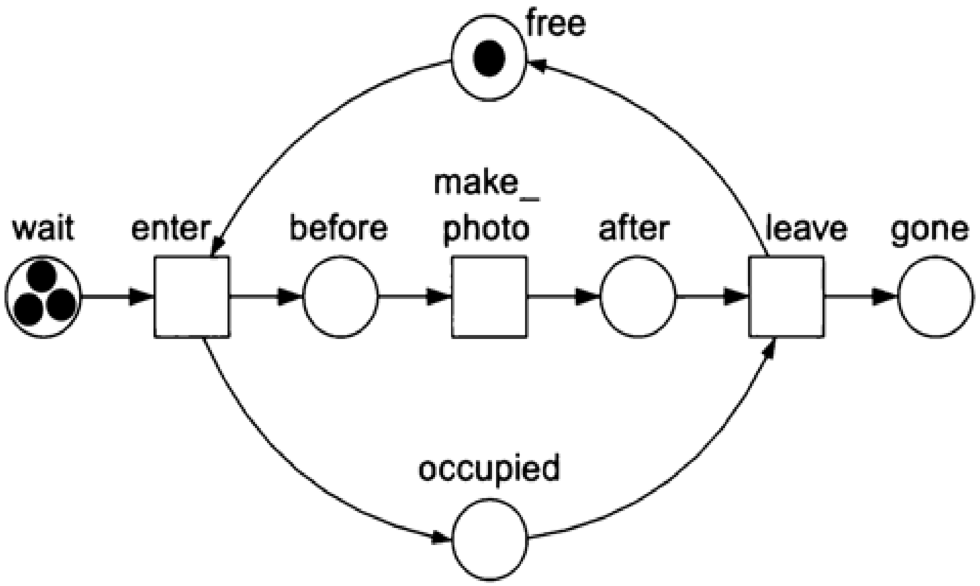


**Figure 2.1:** *A automaton for the process of an X-ray machine*

For example: $\Sigma = \{a, b, c\}$, the binary $\Sigma = \{0, 1\}$,

- The set of all lower-case letters: $\Sigma = \{a, b, c, \dots, z\}$ ... And other sets.

$\Sigma$ consists of almost all available characters (lowercase and capital letters, numbers, symbols and special characters as space or Greek characters $\alpha, \beta ...$).

2. A **string (word)** $u$ over $\Sigma$ is a finite sequence of symbols from the alphabet $\Sigma$.
   - A empty string is denoted by $\varepsilon$ (with no symbols at all).

- The **length** of a string $u$, denoted by $|u|$, is the number of characters. Obviously

  $$|\varepsilon| = 0, \text{ empty string has length of } 0, \text{ and } \quad \varepsilon u = u\varepsilon = u.$$

  If $u = u_1 u_2 \cdots u_n$, clearly $|u| = n$. Let $\Sigma^*$ be the set of **all** strings over $\Sigma$.

- **String concatenation** is map from $\Sigma^* \times \Sigma^*$ to $\Sigma^*$, with $(u, v) \mapsto u.v = uv \ =: w.$

  The substrings $u$ and $v$ are said to be a *prefix* and a *suffix* of $w$.

3. A **language** $L$ over $\Sigma$ is a sub-set of $\Sigma^*$.

   Denote $\emptyset$ the *empty language*. $\Sigma^*$ itself is called the *universal language*.

   **Question**: How about $\{\varepsilon\}, \ \{0, 00, 001\}$, are they languages over $\Sigma = \{0, 1\}$ ?

4. A **formal language** : an abstraction of the general characteristics of *programming languages*.

   A formal language consists of

   (i) a set of *symbols* and

   (ii) formation rules by which these symbols are combined into entities called *sentences*.

5. **Operations on languages**:

   Because languages are *sets*, hence set operators as the *union, intersection, difference* ... of two languages are immediately defined.

   - Union of languages $L_1, L_2$ is set-like union $L_1 \bigcup L_2$.
   - *Product* (or multiplication) $L_1 L_2 = \{u\ v|\ u \in L_1,\ v \in L_2\}$.

     This is based on concatenation of languages $L_1, L_2$
   - *Power* $L^0 = \{\varepsilon\}, \qquad L^i = L^{i-1}\ L, \quad$ if power $\ i \geq 1.$

- *Iteration* (star operation) $L^*$ is a special union of powers

$$L^* = \bigcup_{i=0}^{\infty} L^i = \{\varepsilon\} \bigcup L^+$$

where $L^+ = \bigcup_{i=1}^{\infty} L^i$ is named *proper iteration*.

The union, product and iteration are called regular operations.

- The *complement* $\overline{L} = \Sigma^* \setminus L$ of a language $L$ [is defined with respect to $\Sigma^*$].
- The *reverse* of a language is the set of all string reversals, that is,

$$L^R = \{w^R \ : \quad w \in L.\}$$

### ♦ EXAMPLE 2.1.

Let $\Sigma = \{a, b, c\}$, $L_1 = \{ab, aa, b\}$, $L_2 = \{b, ca, bac\}$

1. $L_1 \cup L_2 =?$ $L_1 \cup L_2 = \{ab, aa, b, ca, bac\}$,
2. $L_1 \cap L_2 =?$ $L_1 \setminus L_2 =?$
3. $L_1 L_2 =?$ $L_1 L_2 = \{abb, aab, bb, abca, aaca, bca, abbac, aabac, bbac\}$,
4. $L_2 L_1 =?$ Now newly define $L = \{ab, aa, b, ca, bac\}$, find $L^2 =?$

### PRACTICE 2.1.

1. Prove by induction that: **If** $u$ and $v$ are strings, **then** the length of their concatenation is the sum of the individual lengths, that is $|u.v| = |u| + |v|$.
2. Let alphabet $\Sigma = \{a, b, c\}$.

   i) Write at least 10 elements of $\Sigma^*$.

   Is $L = \{a, aa, aab\}$ a finite language over $\Sigma$?

ii) Is $L_3 = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \ldots\}$ a language? finite?

iii) Is $L_4 = \{a^n b^n : \quad n = 0, 1, 2, \ldots\}$ also a language over $\Sigma$? Is it $L_3$?

Is the string $u = abb$ in $L_4$? How about $v = aaaaabbbbb$?

**PRACTICE 2.2.**

Let $\Sigma = \{a, b, c\}$. **Give at least 5 strings for each of the following languages**

1. $L_1$: all strings with exactly one '$\underline{a}$'.
2. $L_2$: all strings of even length.
3. $L_3$: all strings which the number of appearances of '$\underline{b}$' is divisible by $\underline{3}$.
4. $L_4$: all strings ending with '$\underline{a}$'.
5. $L_5$: all non-empty strings not ending with '$\underline{a}$'.
6. $L_6$: all strings including at least one '$\underline{a}$' and

   whose the first appearance of '$\underline{a}$' is not followed by a '$\underline{c}$'.

## 2.2 Determining a language

A language can be specified in several ways:

a) **Enumeration** (listing) of its words, for example: $L_1 = \{\varepsilon, 0, 1\}$.

b) Using a **property**, such that all words of the language have this property but other words have not.

c) Using its **grammar**, [defined yet?] for example:

$L(G) = \{a^n b^n | \ n \geq 1\}$ defined by a grammar $G = (V, T, P, S)$, as defined explicitly next.

♦ **EXAMPLE 2.2** (Using a **property**).

Fix $\Sigma = \{a, b\}$, then $L_4 = \{a^n \ b^n : \quad n = 0, 1, 2, \ldots\}$ is a language

(it is in fact $L_3 = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$ in PRACTICE 2.1 above by enumeration).

Also using property on string length gives $L_6 = \{u \in \Sigma^* | \quad n_a(u) = n_b(u)\}$, where $n_c(u)$ denotes the number of letter 'c' in word $u$. Is $L_4 \subset L_6$? ■

**Definition 2.1 (Grammar of a language)** ♣

A **grammar** is defined as a quadruple $G = (V, T, P, S)$ where

$V$ is a finite set of objects called **variables**,

$T$ is a finite set of objects called **terminal symbols**, $T \cap V = \emptyset$,

$P$ is a finite set of **productions** (production rules), and

$S \in V$ is a special symbol called the **start** variable.

We will assume that all production rules are of the form

$$x \to y, \qquad x \in (V \cup T)^+ \text{ and } y \in (V \cup T)^*.$$

- **If** string $w$ consists of $x$, say $\underline{w = axb}$, string $z$ consists of $y$, say $\underline{z = ayb}$, and $x \to y$,

  **then** we define direct derivation, written as $w \Rightarrow z$.

We say that string $w$ derives string $z$ (or that $z$ is derived from $w$).

- If there is a finite *derivation* sequence $w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$ [unspecified steps]

  then we reduce it as $w_1 \overset{*}{\Rightarrow} w_n$. Relation $\overset{*}{\Rightarrow}$ is called a **derivation**.

ELUCIDATION

1. Briefly, the **production rules** of grammar $G$ specify how the grammar transforms one string into another, and they define a language associated with the grammar.
2. Successive strings are generally derived by applying such productions of the grammar $G$ in arbitrary order, and as often as desired.
3. **Derivation** are based on productions.

**Definition 2.2 (Language generated by a grammar)** ♣

A grammar $G = (V, T, P, S)$ can normally generate many strings.

The language $L = L(G)$ defined or generated by the grammar $G$ is

$$L = L(G) = \{w \in T^* : \ S \overset{*}{\Rightarrow} w\} \tag{2.1}$$

(the set of all such *terminal strings*). We obtain the derivation

$$[S \overset{*}{\Rightarrow} w] \ \equiv \ [S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w],$$

in which strings $S$ and $w_i$ are called **sentential forms** of the derivation $\overset{*}{\Rightarrow}$. [a]

[a] Sentential forms are strings that contain both variables and terminals.

### ♦ EXAMPLE 2.3.

Let define grammar $G = (V, T, P, S)$ where

$V = \{S\}$ [only one variable],

$T = \{a, b\}$ [terminal symbols], and

$P = \{S \to aSb,\ S \to ab,\ S \to \varepsilon\}$ with three production rules.

We see $S \Rightarrow aSb \Rightarrow aaSbb = a^2Sb^2 \Rightarrow a^2b^2 \ldots$ then $S \overset{*}{\Rightarrow} aabb = a^2b^2$.

Also $a^2Sb^2 \Rightarrow a^3b^3 \ldots$ and hence

recursively we get the following derivation $S \overset{*}{\Rightarrow} a^nb^n$, i.e.

$$S \Rightarrow aSb \Rightarrow a^2Sb^2 \Rightarrow \ldots \Rightarrow a^nSb^n.$$

Therefore the language defined by $G$ is

$$L(G) = \{w \in T^* :\ S \overset{*}{\Rightarrow} w\} = \{a^n\, b^n |\ n \geq 1\}\ \blacksquare$$

### ♦ EXAMPLE 2.4.

Consider a language $L = \{a^n\, b^{n+1} \mid n \geq 0\}$. Find a grammar that generates $L$.

Should the grammar $G = (V, T, P, S)$ be defined as before?

**No**, we need to modify it to generate an extra $b$ in $L$.

Explicitly we extend a new 'stronger variable' $S$ to incorporate $b$ to the sentences of $L$, hence if redefine $V = \{S, M\}$, $T = \{a, b\}$ then $M$ replaces the old $S$'s role, and so the set of productions becomes $P = \{S \to Mb,\ M \to aMb,\ M \to \varepsilon\}$.

Can you derive specific sentences to convince yourself, like $ab^2 \in L$ by checking $[S \overset{*}{\Rightarrow} ab^2]$?

Or $S \Rightarrow Mb \Rightarrow aMbb \Rightarrow aaMbbb?$ ■

## Is a language generated by a certain grammar $G$?

Often it is not so easy to find a grammar for a language described in an informal way, or to give an intuitive characterization of the language defined by a grammar. To show mathematically that a given language $L$ is indeed generated by a certain grammar $G$, we must be able to show

(a) that every word $w \in L$ can be derived from the start variable $S$ using $G$ and

(b) that every string so derived is in $L$.

**Theorem** (REF. 3, p 17): [a] Not all languages can be generated by grammars.

[a] Antal Iványi. Algorithms of Informatics, AnTonCom Budapest (2010).

HW: Readers should try the following.

**PRACTICE 2.3.** *Denote $n_c(u)$ the number of letter 'c' in any word $u$.*

Take now $\Sigma = \{a, b\}$, and consider a language

$$L_6 = \{u \in \Sigma^* | \quad n_a(u) = n_b(u)\}.$$

Prove that the grammar $G = (V, T, P, S)$ with $T = \Sigma$, $V = \{S, M\}$, and productions

$P = \{S \to \varepsilon, S \to aSb,\ M \to bSa,\ S \to SS\}$ generates the language $L$.

**Hint**: check both $L(G) \subseteq L$ and $L \subseteq L(G)$.

**PRACTICE 2.4.**

Solve examples in page 4.12, and Exercises in page 13, 14 of REF 1 .

## 2.3 Regular Expressions

**Definition 2.3**

***Regular expressions*** *over some given alphabet* $\Sigma$ *involve a combination of* ♣

- strings of symbols from $\Sigma$, parentheses $()$, and
- the operators $+$ (union), $\cdot$ (product or concatenation), and $*$ (transitive closure).

A regular expression over the alphabet $\Sigma$ is 'recursively' determined by 3 conditions

1. $\emptyset$, $\varepsilon$ and $a \in \Sigma$ are all regular expressions, called **primitive** regular expressions.
2. If $r$ and $s$ are regular expressions, so are $(r)$, $r + s$, $r \cdot s$, and $r^*$.
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in (2). ■

**EXAMPLE 2.5.** *On the alphabet* $\Sigma = \{a, b, c\}$

- the string $(a + b \cdot c)^* \cdot (c + \emptyset)$ is a regular expression, explain?
- But $(a + b +)$ is **not** a regular expression, since there is no way it can be constructed from the primitive regular expressions.

In fact, regular expressions can be used to describe some simple languages.

**EXAMPLE 2.6.** *On the alphabet* $\Sigma = \{a, b\}$

- The regular expression $(a+b)^*$ represents all the strings, written $L$
- $a^*(ba^*)^*$ represent the same language $L$
- $(a+b)^*aab$ represent all strings ending with $aab$? or $L \cdot aab$

## *Languages associated with regular expressions*

**Definition 2.4 (Kleene)** ♣

Language $L \subseteq \Sigma^*$ is **regular** if and only if there exists a regular expression over $\Sigma$ representing language $L$.

**Knowledge box 1.** ***Regular Expressions*** *denote languages in the following ways.*

♦ Consider a regular expression (string) $r \in \Sigma^* = \{\emptyset, \varepsilon, a, b, \ldots\}$.

The language $L(r) \subset \Sigma^*$ **associated** with expression $r$ is defined by the following rules.

1. $\emptyset$ is a regular expression, representing the empty language,
2. $\varepsilon$ is a regular expression, representing language $\{\varepsilon\}$,
3. For every $a \in \Sigma$, $a$ is a regular expression, representing the language $\{a\}$.

   **What if $r = a \cdot b = ab$, then the language $L(r) = ?$**

♦ Consider regular expressions $r$ and $s$, representing languages $X = L(r)$, $Y = L(s)$ respectively. **Then** $(r+s)$, $(r \cdot s)$, $r^*$ are regular expressions, (so they) respectively representing languages $X \bigcup Y$, $X\,Y$, and $X^*$. Explicitly,

4. $L(r+s) \equiv L(r) + L(s) = X \bigcup Y$
5. $L(r \cdot s) \equiv L(r)\, L(s) = X\,Y$

6. $L((r)) \equiv L(r) = X$
7. $L(r^*) \equiv \big(L(r)\big)^* = X^*$.

♦ **EXAMPLE 2.7.** *On the alphabet* $\Sigma = \{a, b\}$

- Take $r = a$, $L(r^*) \equiv \big(L(a)\big)^* = \{\varepsilon, a, aa, aaa, \ldots\}$.
- Take $r = a^* \cdot b$, it give the language, $L(r) = L(a^* \cdot b) = L(a^*)L(b) = \{b, ab, a^2b, a^3b, \ldots, aa, aaa, \ldots\}$.
- The language $L(a^* \cdot (a+b)) = L(a^*)\ L(a+b) = ...? = \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}$
- $r = (a+b)^*$ represents all the (regular) strings of of $a$'s and $b$'s, any order.

  So language $L(r) = \big(L(a+b)\big)^* = \Sigma^*$? ■

  We also employ the following properties, applied for regular expressions $x, r$ and $s$.

$$\begin{aligned}
(x+r)+s &\equiv x+(r+s);\\
(xr)s &\equiv x(rs)\\
(x+r)^* \equiv (x^*+r)^* &\equiv (x+r^*)^* \equiv (x^*+r^*)^*\\
(x+r)^* &\equiv (x^*r^*)^*\\
(r^*)^* &\equiv r^*\\
x^*x &\equiv xx^*;\\
xx^*+\varepsilon &\equiv x^*
\end{aligned}$$

**PRACTICE 2.5.** *On the alphabet* $\Sigma = \{a, b\}$,

I) $s = (aa)^*(bb)^*b$ represents all the (regular) strings of language $L(s) =?$

II) Is $E = (a+b)^*(a+bb)$ regular? Find the language $L(E) =?$

Hint: $L(a+bb)$ consists of word ending by $a$ or a double $b$. ■

## PROBLEM 2.1.

On the alphabet $\Sigma = \{a, b, c\}$, give at least 5 words for each language represented by the following regular expressions.

a) $E_1 = a^* + b^*$,

b) Elucidate why $a^*b = \{b, ab, a^2b, a^3b, \ldots, aaa\ldots ab\}$?

Then find the language of $E_2 = a^*b + b^*a$.

c) $E = \underline{\text{b(ca + ac)}\ \ \text{(aa)}} + \ a^* \cdot (a+b) = \underline{\text{r}} + s \implies L_E = ?$

Hint for c): You will prove firstly $L_2 = L(s) = L(a^* \cdot (a+b)) = \{a^{n+1},\ a^m b \mid n, m \geq 0\}$.

Next find $L_0 = L(b(ca+ac)) = L(bca) \cup L(bac)$,

then finally get $L_1 = L(r) = L_0\ \{aa\} = L(r)$. ■

## PROBLEM 2.2.

Simplify each of the following regular expressions

1. $E_1 = \varepsilon + ab + abab(ab)^*$,
2. $E_2 = aa(b^* + a) + a(ab^* + aa)$,
3. $E_3 = a(a+b)^* + aa(a+b)^* + aaa(a+b)^*$.

# PART B. AUTOMATA

AIM of studying Automata: toward a representation of a process or system.

## 2.4 Finite Automata

An **automaton** (a machine) can be represented by a graph in which

the vertices give the *initial states* and one or several *final/accepting states*, and the edges means *transitions or events*.

**Definition 2.5 (Finite automaton (FA))** ♣

A finite automaton $M$ is a 5-tuples $(Q, \Sigma, q, \delta, F)$ or $(Q, \Sigma, I, \delta, F)$ where

1. $Q$ is a finite set, called the **states**,
2. $\Sigma$ is a finite set, called **alphabet**,
3. $q \in Q$ is the initial or **start state**,
4. $\delta : Q \times \Sigma \to Q$ is the **transition function**,
5. $F \subseteq Q$ is the set of **accepting states** of the automaton.

NOTE: the map $\delta : Q \times \Sigma \to Q$ is *surjective*, but not necessarily *injective*.

**Definition 2.6 (Accepted string and Language of a machine (finite automaton))** ♣

Fix a finite automaton $M = (Q, \Sigma, q, \delta, F)$.

A string $w = a_1 a_2 \cdots a_n \in \Sigma^*$ is accepted by an automaton $M$ if symbolically

$q \overset{w}{\rightsquigarrow} q_w \in F$, meaning applying $w$ on the start state $q$ gives a final (accept) state $q_w$ in $F$.

- If $A$ is the set of all strings $w$ that machine $M$ accepts, we say that

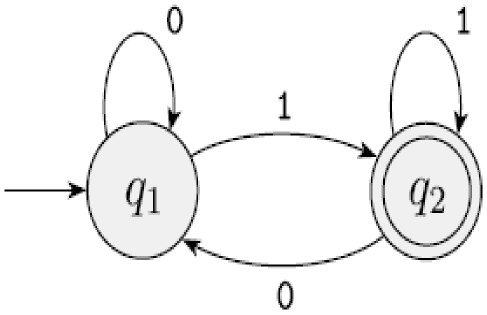
  $A$ is **the language of machine** $M$ and write $L(M) = A = \{w : |M \text{ accepts string } \; w\}$.

  We say that machine $M$ recognizes (or accepts) the language $A$.

NOTE: If use automaton $M = (Q, \Sigma, I, \delta, F)$ where $I$ is the set of a few start states ($I \subset Q$) we just say $w = a_1 a_2 \cdots a_n \in \Sigma^*$ accepted by $M$ when apply $w$ on some $q \in I$.

♦ **EXAMPLE 2.8** (Finite automaton to a regular expression )**.**

This state diagram shows the two-state finite automaton $M = \big(\{q_1, q_2\}, \{0, 1\}, q_1, \delta, \{q_2\}\big)$,



A finite automaton with two states

where $F = \{q_2\}$ (unique accepting state), the transition function $\delta$ is given by table below, that is $\delta(q_1, 0) = q_1$, $\delta(q_1, 1) = q_2$ and so on. Final states are drawn with a double circle.

| | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_1$ | $q_2$ |

Let string $r = 1101 \in \{0, 1\}^*$, will automaton $M$ accept it? Write the language $L(M)$? $L(r)$?

- We apply $r = 1101$ on the start state $q_1$, letter by letter to find out $r \in L(M)$?

  Use $r = 1101$ we finally get state $q_2 \in F$ (accepting states).

  The string $r$ so is **accepted** because $q_2 \in F$, hence $r \in L(M)$.

  But string $s = 110$ leaves $M$ in state $q_1$, so it is **rejected**, and hence $s \notin L(M)$.

- You would see that with this specific automaton $M$,

  $M$ accepts all strings that end in a **1**. Thus the language of $M$, $L(M) = \{w | \ w$ ends in a **1** $\}$

  [Check directly all string $w = u \cdot 1$ acting from $q_1$, for all string $u \in \{0, 1\}^*$]

  So can we write $\underline{L(M) = \{0, 1\}^* \ 1}$, expressing the complete information of $M$?

- What is $L = L(r)?$ Moreover the regular expression

$$r = 1101 \in 1^* \ 01 = \underline{1^* \cdot (01) =: L} \subset L(M)$$

$L$ partially expresses information of $M$, so $L \neq L(M)$.

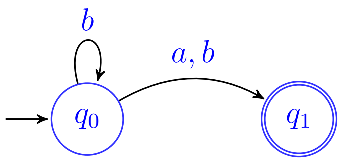
Abstractly, if we would replace letter $a = 1, \ b = 0$ on the machine $M$

then we get the same conclusion, for $r = aaba \in a^*\, ba = L$.

- Conlude that $r = 1101 \in 1^*(0 + 1) = 1^*\, 0 + 1^*\, 1$ is incorrect! Why?

■

♦ **EXAMPLE 2.9** (Finite automaton to a regular expression ).



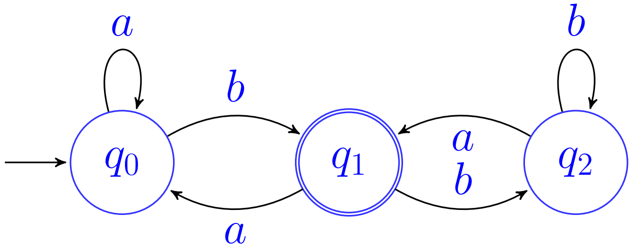
**Regular expression** is $r = b^*(a + b)?$ But what is the transition function $\delta$?

**PROBLEM 2.3.**

I) Let $\Sigma = \{a, b\}$. **Which of the strings**

1. $a^3b$,
2. $aba^2b$,
3. $a^4b^2ab^3a$,
4. $a^4ba^4$,
5. $ab^4a^9b$,

**are accepted by the following finite automaton?**



II) **Do Exercises on page 4.2x, for** $x = 4, 5, 6$ **of REF. 1.**

### 2.4.1 Nondeterministic finite automata

**Definition 2.7 (Nondeterministic finite automaton (NFA))**

*A non-deterministic finite automaton $M$ is represented by a 5-tuples $(Q, \Sigma, q_0, \delta, F)$ or $(Q, \Sigma, I, \delta, F)$ where* ♣

1. $Q$ a finite, nonempty set of **states**,
2. $\Sigma$ is the **input alphabet**,
3. $q_0$ is an initial state, [$q_0$ can be extended to $I \subset Q$, the set of initial states],
4. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is a transition function,
5. $F \subseteq Q$ is the set of final/accepting states of the automaton.

- Generally the $\delta$ (in Item 4) is *surjective*. It can be extended to a set $E$ of many transition functions.
- **Nondeterminism/ Nondeterministic property** means a choice of moves for automaton $M$.

  Rather than prescribing a unique move in each situation, we allow a set of possible moves.

  Formally, we achieve this by defining the transition function $\delta$ so that $\mathrm{Range}(\delta) \subset Q$

  or $\mathrm{Range}(\delta) \in 2^Q$ [its range is a set of possible states in $Q$].
- An NFA is a *directed, labeled graph*, named the transition graph $T(M) = (V, E)$ of automaton $M$, whose node set $V \equiv Q$ and $E = \{(p, a, q)\}$,

$$p \xrightarrow{a} q$$

  of transitions - **directed edges [labeled with letter** $a \in \Sigma$**]** - from (state) node $p$ to node $q$.
- Among nodes some are initial and some final states.

  *Initial states* are marked by a small arrow entering the corresponding vertex, while the **final states are marked with double circles** [as $q_4$ in the next figure].

- A sequence $(q_i, a_{i+1}, q_{i+1})$, where $q_i \in Q$ for $i = 0, 1, \ldots, n-1$ of edges of graph $T(M)$

  is a walk $w$ in $T(M)$ with the label $a_1, a_2, \ldots, a_n$.

  We write $w = a_1, a_2, \ldots, a_n$ or shortly $q_0 \xrightarrow{w} q_n$.

♦ **EXAMPLE 2.10.** *Fix* $Q = \{q_1, q_2, q_3, q_4\}$ *and* $\Sigma = \{0, 1, \ \varepsilon\}$.

* Transition function $\delta$ gives the $\text{Range}(\delta) \in 2^Q$:

**If**, as in Figure 2.2, the current state is $q_1$, select actions $w = 1\ 0\ 1$

then the symbol $a = 1$ is read, and $\delta(q_1, a) = \{q_1, q_2\}$,

**hence** either $q_1$ or $q_2$ could be the next state of the NFA.

* The transition graph $T(M)$ of $M$ has edges $(q_1, q_1)$ and $(q_1, q_2)$ with label $a = 1$.

* In Figure 2.2, string of actions $w = 1\ 0\ 1$ acts on $q_1$ and returns $q_4$.

The states in a walk are **not** necessary distinct. ■

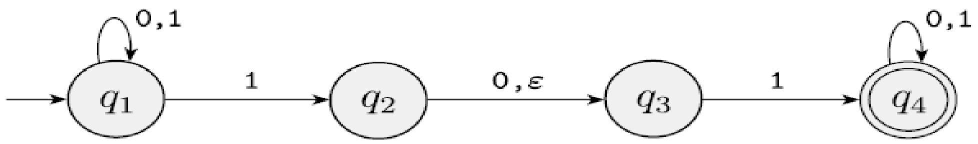
**Definition 2.8 (Extended transition function in NFA)** ♣

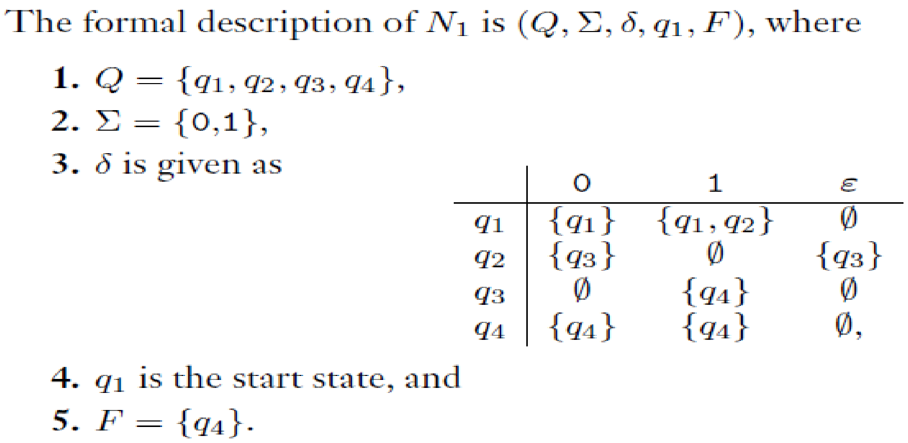
For an NFA, the **extended transition function** $\delta^*$ is defined

so that $\delta^*(q_i, w)$ contains $q_j$ if and only if

there is a walk in the transition graph from $q_i$ to $q_j$ labeled $w$.

This holds for all $q_i, q_j \in Q$, and $w \in \Sigma^*$.




The formal description of $N_1$ is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. $\delta$ is given as

|  | 0 | 1 | $\varepsilon$ |
|---|---|---|---|
| $q_1$ | $\{q_1\}$ | $\{q_1, q_2\}$ | $\emptyset$ |
| $q_2$ | $\{q_3\}$ | $\emptyset$ | $\{q_3\}$ |
| $q_3$ | $\emptyset$ | $\{q_4\}$ | $\emptyset$ |
| $q_4$ | $\{q_4\}$ | $\{q_4\}$ | $\emptyset$, |

4. $q_1$ is the start state, and
5. $F = \{q_4\}$.

An nfa with 4 states and 1 accepting state.

**Figure 2.2:** *NFA with total 4 states*

We need two more concepts for the next definition (language accepted by NFA).

- **Productive walk**: A walk $w = a_1, a_2, \ldots, a_n \equiv q_0 \xrightarrow{w} q_n$

  is *productive* if its start-vertex is an initial state

  and its end-vertex is a final state, i.e. $q_0 \in I$ and $q_n \in F$.

- **Word and language accepted by an** NFA $M$ or its transition graph $T(M)$:

We say an NFA $T(M)$ accepts or recognizes a word $w = a_1, a_2, \ldots, a_n$ [or $w$ accepted by $T(M)$]

if this word is the label of a productive walk (that is, there is some sequence of possible moves that will put the machine in a final state $q_n$ at the end of the string.)

The set of words accepted by an NFA is called the `language accepted` by this NFA.

**Definition 2.9 (Language accepted by `NFA`)**

♣

The language being **accepted** by an `NFA` $M = (Q, \Sigma, I, \delta, F)$ is determined by

$$L(M) = \{w \in \Sigma^* \ : \ \delta^*(q, w) \cap F \neq \emptyset, \ \exists q \in I\}. \tag{2.2}$$

It is the language consists of all strings $w = a_1, a_2, \ldots, a_n$ labeling of a **productive walk**.

Explicitly this means for each string $w \in L(M)$ there is a walk labeled $q \xrightarrow{w} p$ from an

initial vertex $q \in I$ of the transition graph to some final vertex $p \in F$.

So instead of using

$$L(M) = \{w \in \Sigma^* \ : \ \delta^*(q, w) \cap F \neq \emptyset, \ \exists q \in I\},$$

we also use the equivalent

$$L(M) = \{w \in \Sigma^* \ : \ \exists q \in I, \exists p \in F, \ \text{and there exists the path } \ q \xrightarrow{w} p\}. \tag{2.3}$$

**Knowledge box 2 (Regular Expressions denote Regular Languages).**

- The language accepted by an NFA so is defined by the extended transition $\delta^*(q, w)$ [Def. 2.8].
- A language $L$ is **regular** if there exists a *regular expression* generating $L$ [Def. 2.4 (Kleene)].

  Let $r$ be a regular expression. Then there exists some nondeterministic finite automaton (NFA) that accepts the language $L(r)$.

  Consequently, $L(r)$ is a regular language.

We discuss more on the equivalence connection between regular languages and regular expressions later in Section 2.7, including two smaller problems

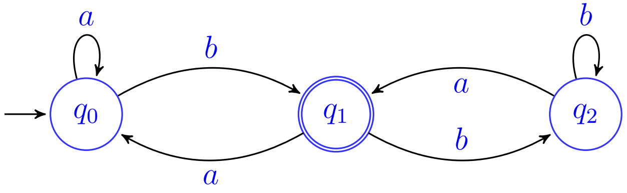
1. Regular Expressions define regular languages: how construct an NFA $M$ from a given regular expression $r$, so that $M$ accepts $L(r)$.
2. Regular Languages give regular expressions: from a regular language $L$, find a regular expression that generates $L$.

**PROBLEM 2.4.**

(I) Consider the set of strings on $\Sigma = \{a, b\}$ in which every $aa$ is followed immediately by $b$. For example $aab, aaba, aabaabbaab$ are in the language, but $aaab$ and $aabaa$ are not.

Construct an accepting NFA.

(II) Give regular expression for the following finite automaton



**PROBLEM 2.5.** *Do Exercise on page **4.3y**, for $y = 1, 2, 3$ of REF. 1.*

## 2.4.2 Deterministic finite automata

**Definition 2.10 (Deterministic finite automaton (DFA))** ♣

A deterministic finite automaton or DFA $M$ is defined by a 5-tuples

$$M = (Q, \Sigma, q_0, \delta, F)$$

with

1. $Q$ a finite, nonempty set of **states**,
2. $\Sigma$ is the **input alphabet**,
3. $q_0 \in Q$ is the initial state,
4. $\delta : Q \times \Sigma \to Q$ is a transition function,
5. $F \subseteq Q$ is the set of final/accepting states of the automaton, $|F| \geq 1$.

### ELUCIDATION

- A DFA fulfills that $|I| = 1$ and moreover

$$|\delta(q, a)| \leq 1, \ \forall q \in Q, \ \forall a \in \Sigma$$

(extened from Definition 2.7 where $\delta(q, a)$ was a map.)

For letter $a \in \Sigma$, **if** $|\delta(q, a)| = 1$ [usual map]

**then** $\delta(q, a) = p$, an edge in the transition graph $T(M)$.

- The fact $|\delta(q, a)| = 0$ says $\delta(q, a) = \emptyset$ (**nonfinal trap state** in the DFA):

  this represents an *impossible* move for the automaton and, therefore,

  means **nonacceptance** of the letter $a$, and nonacceptance of any string $r$ holding $a$.

- Given a walk $r = abc \ldots s$ (series of **accepted** arrows/ letters/ actions), and a state $q$,

  the automaton $M$ starts with the transition function, called an application $\delta(q, a) = p$.

$$q \xrightarrow{a} p \xrightarrow{b} \cdots \xrightarrow{s} p_n$$

  The input mechanism then advances one position to the right, as a result

  each move consumes one input symbol of $r$.

- CONCLUSION:

  The transition graph $T(M)$ of finite automaton $M$ has an edge $(q, p)$ with label $a$...

  When **the end letter** $s$ of the string $r = a\ b\ c \ldots\ s$ is reached,

  the string is **accepted** if the automaton is in state $p_n \in F$ (one of its final states):

$$q \xrightarrow{a} p \xrightarrow{b} \cdots \xrightarrow{\mathbf{s}} \mathbf{p_n}.$$

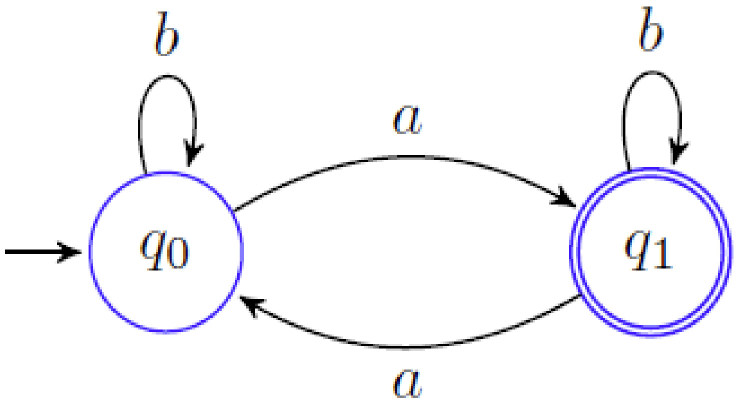
  Otherwise the string is **rejected**. ♦

♦ **EXAMPLE 2.11.**

The figure shows the transition graph $T(M)$ of a DFA $= M = (Q, \Sigma, q_0, \delta, F)$ with $F = \{q_1\}$.

i/ Determine the transition $\delta$.

ii/ Check $M$ recognizes the **set** $A$ of strings which contain an odd number of $a$, that is $L(M) = A$.



A DFA with two states and only one accepting state

iii/ Determine a new machine $M_B$ accepts the **set** $B$ of strings which contain an even number of $a$, that is $B = L(M_B)$.

HINT: Obviously $Q = \{q_0, q_1\}$, the alphabet $\Sigma = \{a, b\}$, and the set $F = \{q_1\}$ of accepting states.

**i**/ The transition $\delta$ fulfills $\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_0 \ldots$ and given in table

| | a | b |
|---|---|---|
| $\to q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_0$ | $q_1$ |

**ii**/: Need to write the pattern of strings $r \in A$ which contain an odd number of $a$, as

$$r = b^* a^n b^*, \ n = 2k + 1?$$

Then check the equality $\delta(q_0, r) = q_1 \in F$ if and only if $r$ has the above pattern only.

**iii**/: Perhaps we redefine the new defining string

$$s = b^* a^m b^*, \ m = 2k \Longrightarrow B = L(M_B)$$

where $M_B = (Q, \Sigma, q_1, \delta, F)$, $F = \{q_0\}$? ■

**Proposition 2.1**

*Transition graphs provide convenient ways for working with FA.* ♠

Precisely, given a DFA $M = (Q, \Sigma, q_0, \delta, F)$, let its transition graph be $T(M)$.

For all $q_i, q_j \in Q$, and $w \in \Sigma^+$, then the extended transition $\delta^*(q_i, w) = q_j$

if and only if there is a walk with label $w$ (in $T(M)$) from $q_i$ to $q_j$.

♦ **EXAMPLE 2.12.** *Set alphabet* $\Sigma = \{0, 1\}$

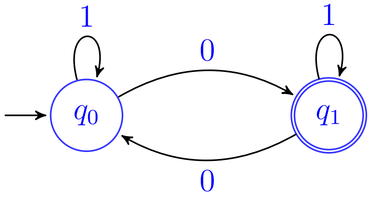
- Give a DFA that accepts all words that contain a number of 0 multiple of 3.
- Give an execution of this automaton on word w= 1101010.
- Any other way to define DFA?

**GUIDANCE for solving.**

Define the states $Q$, the transition $\delta$ and set $F$ of accepting states properly.

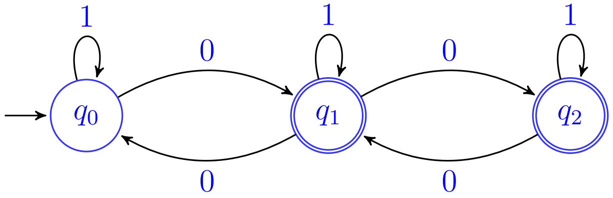
i) Call DFA $M = (Q, \Sigma, q_0, \delta, F)$ with $Q = \{q_0,\ q_1\}$, and $F = \{q_1\}?$.

Then define a transition $\delta$ such that $\delta(q_o, w) \in F$ for **any word** $w$ satisfying that $\underline{n_0(w) \mod 3 = 0}$ [word $w$ contains a number of 0 multiple of 3].



ii) Here $w = 1101010$ fulfills $n_0(w) \mod 3 = 0$, and $\delta(q_o, w) = q_1 \in F$.

iii) What if we redefine states $Q = \{q_0,\ q_1,\ q_2\}$, and $F = \{q_1, q_2\}$? Can we redraw the graph as



**PROBLEM 2.6.** *Do Exercise on page 4.37, REF.1.*

WHAT NEXT?

We next study four basic problems in Finite Automata theory:

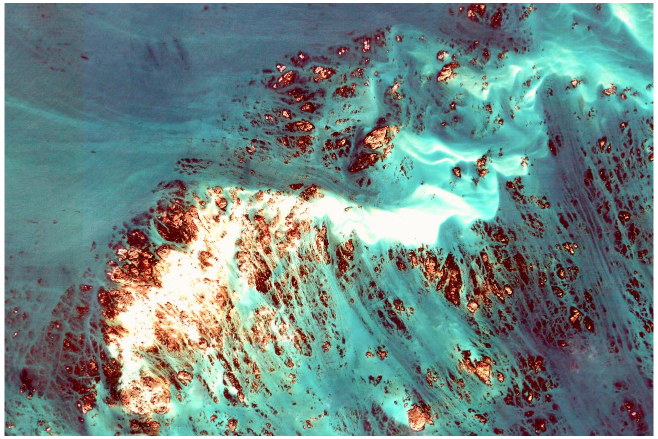
1. Construction of a DFA and NFA from a given language $L$, or a given regular expression $E$, Section 2.5.1 and 2.5.2.
2. Equivalence of automata: Build a DFA from a given NFA, see Section 2.6.1;

   **Why?** For an NFA, for a single input given to a single state, the machine can go to more than one state. So, for an NFA, it is *hard to decide* for a string to be accepted by the NFA or not. This type of problem will not arise for DFA.

3. Minimization of a DFA, given in Section 2.6.2; **Why?** To save storage in computer.
4. Inverse of Problem 1: Determining a regular expression given an automaton (NFA- DFA), studied in Section 2.7.

   Problems 1 and 4 show a **mutual relationship** between languages and automata.

# Finite Automata- Basic Problems



- Problem 1: finding automata (DFA and NFA) given a regular expression
- Problem 2: studying equivalence of DFA and NFA (transform NFA to DFA)
- Problem 3: minimizing a DFA
- Problem 4: finding a regular expression given an automaton

# 2.5 Finite Automata- Basic Problems

## 2.5.1 Recognized languages and Construct a DFA

**Definition 2.11 (Language recognized (accepted) by DFA)** ♣

I) A language $L \subset \Sigma^*$ (over an alphabet $\Sigma$) is **accepted** by an automaton

if there exists a DFA$M$ accepting all strings of $L$. Then we write $L = L(M)$.

Precisely, the language $L(M)$ being accepted by a DFA $M = (Q, \Sigma, q_0, \delta, F)$ is determined by **productive words** $w$:

$$L(M) = \{w \in \Sigma^* \ : \ \delta^*(q_0, w) \in F\}. \tag{2.4}$$

Nonacceptance means that the DFA stops in a nonfinal state, so that

$$\overline{L(M)} = \{w \in \Sigma^* \ : \ \delta^*(q_0, w) \notin F\}.$$

II) A language is said to be **regular** if it is accepted by some DFA $M$.

**Proposition 2.2**

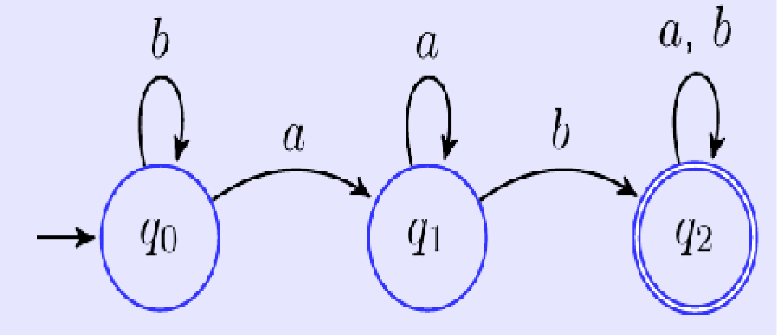
*Given two **accepted** (recognized) languages $X, Y$ by certain* DFA $M$

*then $X \bigcup Y$, $X \bigcap Y$, $X \cdot Y$, and $X^*$ are also recognized.* ♠

♦ **EXAMPLE 2.13** (From a language to a DFA).

Construct a DFA given in above figure with 3 states $Q = \{q_0, q_1, q_2\}$,

the alphabet $\Sigma = \{a, b\}$, and such that it



recognizes the language $L$ containing the sub-string $ab$.

**GUIDANCE for solving.**

Denote the DFA by $M = (Q, \Sigma, q_0, \delta, F)$, where $q_0 \notin F$. Need to find function $\delta$ and set $F$.

- KEY FACTS:

(1) Any string $S \in \Sigma^*$ built up from the alphabet $\Sigma = \{a, b\}$ can have any fixed length, and written as $S = (a + b)^*$.

(2) The fact of $L(M) = L \ni ab$ [ the language $L$ contains the sub-string $ab$]

implies that $L$ consists of **regular expression** of the form $S_l \ ab \ S_r = (a + b)^* \ ab \ (a + b)^*$.

Indeed, first $q_0 \notin F$, and $|F| \geq 1$.

- The substring $ab$ acts on the initial $q_0$ in 2 steps to give the final state $q_2$ via $q_1$ (w.l.g), hence state $q_2 \in F$. The initial $q_0 \notin F$, also

$q_1 \notin F$, hence $|F| = 1$, we get $q_0 \overset{ab}{\to} q_2 \in F$.

This results in $F = \{q_2\}$. As a result, any word $w_{right} = ab \ (a + b)^* \in L$.

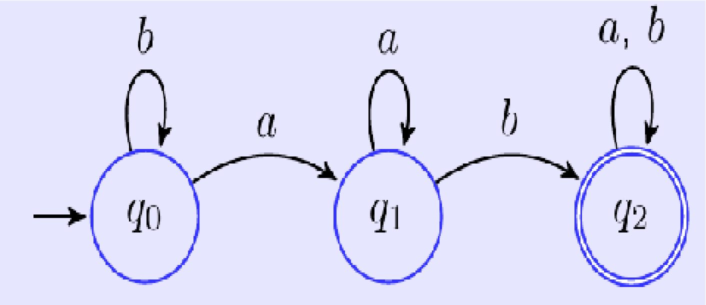
- Second since $M$ is deterministic, $w_{left} = (a+b)^*\, ab \in L$ and so the **concatenated word**

$$w = w_{left} \cdot w_{right} = (a+b)^*\, ab\, (a+b)^* \in L = L(M).$$

We get $M = \big(\{q_0, q_1, q_2\}, \{a, b\}, q_0, \delta, \{q_2\}\big)$, where the transition $\delta$ is given by the table

| | $a$ | $b$ |
|---|---|---|
| $\rightarrow q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_1$ | $q_2$ |
| $q_2*$ | $q_2$ | $q_2$ |

Here $^*$ means final state(s). Can you draw the FA or transition graph $T(M)$?



This example shows a language $L$, accepted by the above DFA $M$, so $L$ is regular. $L$ is precisely determined by a **regular expression** $ab$:

$$L = L(ab) = \{r, \quad r = (a+b)^*\, ab\, (a+b)^*\}. \blacksquare$$

**NOTE**: $(a+b)^* = \big(\{a\} \cup \{b\}\big)^* = \Sigma^*$

is the set of all string made by $a, b$ in any order and length.

And writing $c\, d \equiv c \cdot d$ = concatenation of two strings $c, d$.

**PRACTICE 2.6.**

Determine a DFA that recognizes the language over the alphabet $\{a, b\}$ with an even number of $a$ and an even number $b$.

**Automaton:**

Shall we provide a deterministic machine $M$ so that each productive word in its language $L(M)$ has an even number of $a$ and an even number $b$?

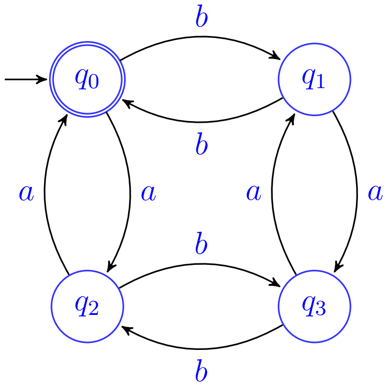
The machine is defined by table below in which we denote

$\rightarrow$: start state, and

$^*$: final state(s)

| | $a$ | $b$ |
|---|---|---|
| $\rightarrow q_0^*$ | $q_2$ | $q_1$ |
| $q_1$ | $q_3$ | $q_0$ |
| $q_2$ | $q_0$ | $q_3$ |
| $q_3$ | $q_1$ | $q_2$ |

and diagram



### 2.5.2 Regular languages and Construct an NFA

Definition 2.11.II) says that a **language** $L$ **is regular** if it is accepted by some DFA. We will see that NFA and DFA are equivalent, [see Section 2.6], so $L$ is also regular if it is accepted by some NFA.

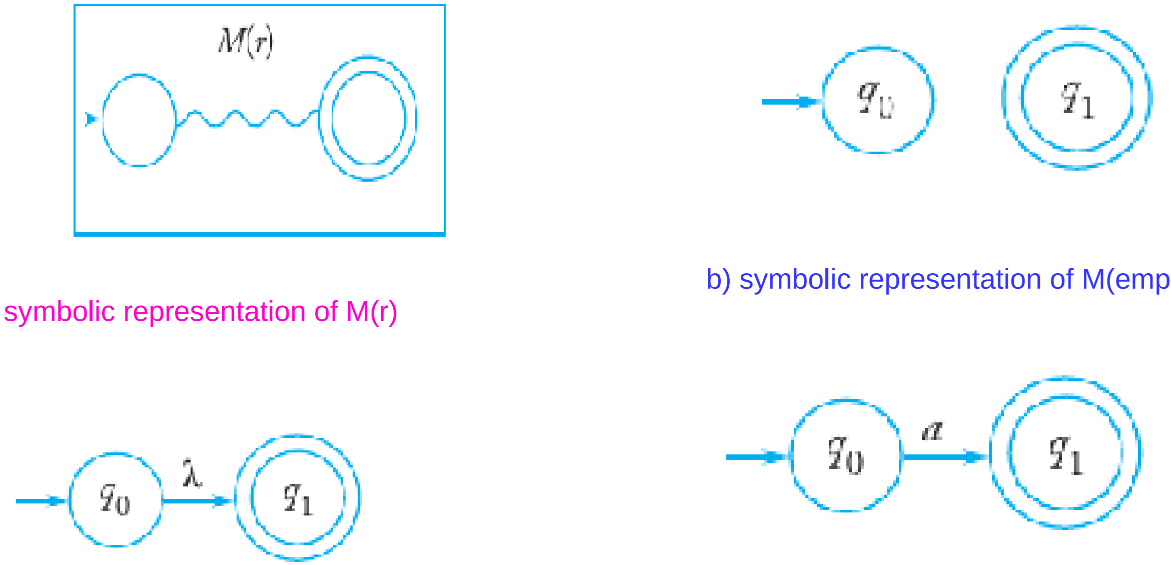
**Theorem 2.1 (THEOREM 3.1 of REF. 2)**

*(I) If we have any regular expression $r$, we can construct an nondeterministic finite automaton (NFA) $M$ that accepts the language $L(r)$.*

*(II) For every NFA $M_0$ there is an equivalent one $M_1$ with a **single** final state.* ♡

**Fact 2.1.**

1. Given a regular expression $r$, then there exists an NFA $M(r)$ accepting the regular language $L(r)$, and the **symbolic (or schematic) representation** of $M(r)$ is shown in Figure 2.3.a).



a) symbolic representation of M(r)

b) symbolic representation of M(empty)

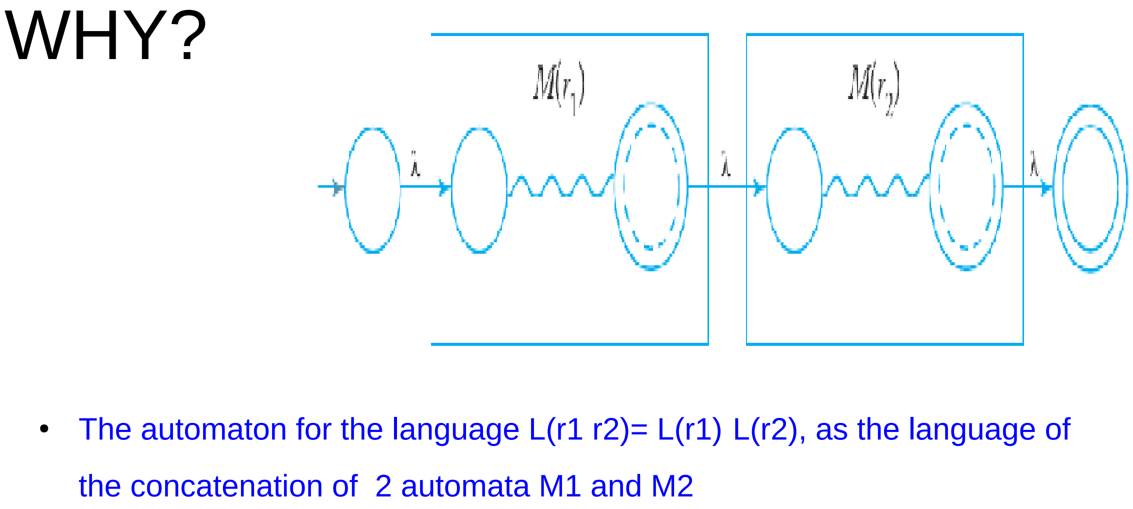
c) symbolic representation of M(λ)

d) symbolic representation of M(a)

**Figure 2.3:** *Symbolic representation of automata based on (primitive) regular expressions*

2. The simple (primitive) regular expressions $\emptyset, \varepsilon(\lambda)$, and $a \in \Sigma$ determine the automata (and their accepted languages), respectively shown in Figure 2.3.b) to d).

3. The automaton $M(r_1 \cdot r_2)$ of product of two regular expressions $r_1, r_2$ is given Figure 2.4.



WHY?

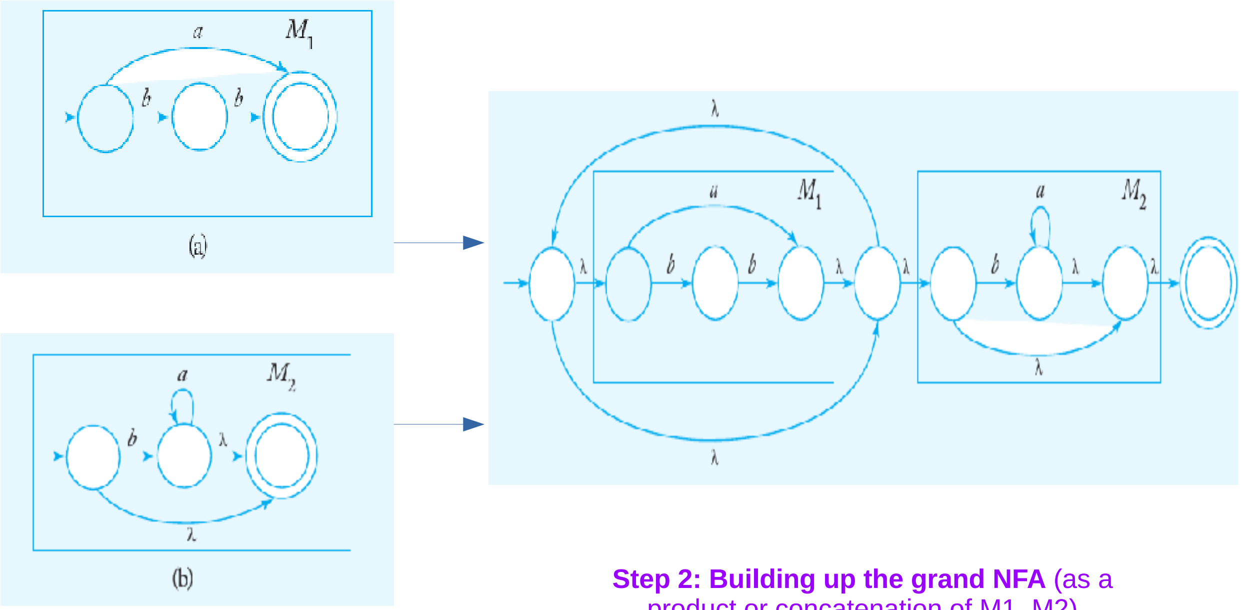
- The automaton for the language L(r1 r2)= L(r1) L(r2), as the language of the concatenation of 2 automata M1 and M2

**Figure 2.4:** *Automaton for L (r1 r2)*

♦ **EXAMPLE 2.14** (From a language to a NFA).

Construct an NFA with 3 states $Q = \{q_0, q_1, q_2\}$, and the alphabet $\Sigma = \{a, b\}$, that recognizes the given language $L(r)$ where $r = (a + bb)^*\ (ba^* + \varepsilon)$.

HINT: see Figure 2.5. The left top figure shows machine $M_1$ accepting language



**Step 1: Construct NFA of compnents,** M1 accepts L1= L(a+bb)...

**Step 2: Building up the grand NFA** (as a product or concatenation of M1, M2)

**Figure 2.5:** *Construct an* NFA *with 3 states*

$L_1 = L(r_1) = L((a+bb)^*) = L(a+bb)$, the left bottom figure shows $M_2$ accepting $L_2 = L(r_2) = L((ba^* + \varepsilon)) = L((ba^*\,\varepsilon + \varepsilon))$. The result machine $M = M(r_1\, r_2) = M_1 \cdot M_2$.

**PROBLEM 2.7.** *Consider the set of strings* $\Sigma^*$ *on* $\Sigma = \{a, b, c\}$.

Construct DFAs (automata) that recognize the languages represented by the following regular expressions

- $E_1 = a^* + b^*a$, $E_2 = b^* + a^*\, ab\, a^*$, and $E_2 = aab + cab^*\, ac$.

DIY: See Figures 3.3 till 3.5 in REF.2. for automata $M(r_1 + r_2)$ and $M(r_1^*)$, given regular expressions $r_1, r_2$.

PROLOGUE - WHAT NEXT ?

We study Equivalence of DFA and NFA.

# PART C. Equivalence of DFA and NFA

AIM: Study Equivalence of DFA and NFA

## 2.6 Equivalence of DFA and NFA

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. The observable pros and cons of NFA and DFA, as well as their relationship are briefed as follows.

♣ **OBSERVATION 4.**

1. A DFA (deterministic automaton) is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. Generally,

   deterministic and nondeterministic finite automata **recognize** the same class of languages.

2. Although the classes of dfa's and nfa's are in fact equally powerful, as seen by the following claim: For every language $L_D$ accepted by some DFA

   there is a NFA $M$ that accepts the same language, $L(M) = L_D$.

   But the converse (from NFA to DFA) is **not so obvious**.

**How to convert any NFA into an equivalent DFA?**

We need the concept of equivalence between finite automata $M_1$ and $M_2$:

$M_1$ and $M_2$, are said to be equivalent if they both accept the same language,

$$L(M_1) = L(M_2).$$

### ♦ EXAMPLE 2.15.

The DFA shown in top-left of Figure 2.6 is equivalent to the NFA (non-deterministic one) in its bottom-right, since they both accept the language $L = \{(10)^n \ : n \geq 0\}$.

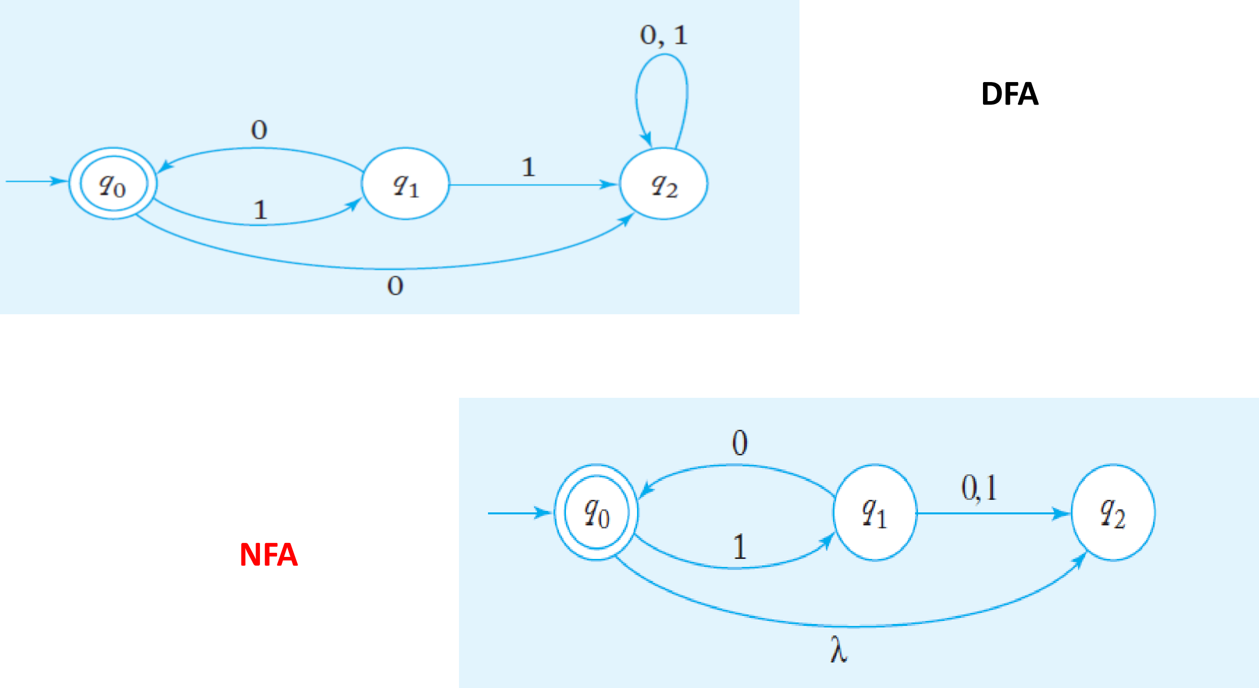


**Figure 2.6:** *Equivalence of two small automata*

### 2.6.1 From NFA (non-deterministic automaton) to DFA

**Theorem 2.2 (From NFA to DFA- Theorem 1.10 of REF.3)**

***For any NFA one may construct an equivalent DFA.*** ♡

An algorithm for transforming a NFAto an equivalent DFA must have:

**Input**: A NFA $M_N = (Q_N, \Sigma, I \ni q_0, \delta_N, F_N)$,

and let $L$ be the language accepted by $M_N$.

**Output: A** DFA $M_D = (Q_D, \Sigma, \{q_0\}, \delta_D, F_D)$ such that $L = L(M_D)$.

HOW to transform from non-deterministic machine $M_N$ to deterministic machine $M_D$?

1. In the **transition graph** $T_D = T(M_D)$ of DFA $M_D$,

   every node must have exactly $|\Sigma|$ outgoing edges,

   each labeled with a different element of $\Sigma$ (by definition).

   The initial state set $I$ (of NFA) becomes the initial node of $T_D$.

2. After an NFA has read a string $w$, we may **not** know exactly what state it will be in, but, by definition, we can say that it must belong to a set of possible states $S \subset Q$:

$$q_0 \xrightarrow{w} u_j \in S$$

An equivalent DFA after reading the same string $w$ must be in some definite set $S_n$,

see **ALGORITHM 2.1** [THEOREM 2.2, REF.2.]

**Algorithm 2.1 NFA to an equivalent DFA** $\big(M_N = (Q_N, \Sigma, I \ni q_0, \delta_N, F_N)\big)$

**1) Initializing $T_D$:** Create $T_D$ vertex $I \ni q_0$, identify it as the initial vertex. Set $S = I$.

**2) Enlarging $T_D$:** At loop $n \geq 1$, repeat next steps until no more edges are missing.

i) - Take any vertex $S = \{u_j\}$ of $T_D$ that has no outgoing edge for some symbol $a \in \Sigma$.

ii) - Compute values of the extended $A_j = \delta^*_N(u_j, a)$, for all $u_j \in S$.

iii) - Make a set $S_n = \bigcup_{u_j \in S} A_j = \bigcup_j \delta^*_N(u_j, a)$ by merging sets $A_j$ [use the above $a$].

iv) - Create node $S_n \in T_D$ if it does not exist, and

Add to $T_D$ an edge with label $a$ from $S$ to $S_n$: $S \xrightarrow{a} S_n$.

**3) Expanding set $F_D$:** Any $S_n$ whose entries contains any $q_f \in F_N$ (of the NFA $M_N$) is identified

as a final vertex in the DFA $M_D$. Update $S = S_n$, go back to Step 2.

**Stopping condition:** No more edges are missing.

If $M_N$ accepts empty string $\varepsilon$, the vertices $I$ in $T_D$ is also made a final vertex.

Briefly, in constructing DFA [Step 2)] we can use the corresponding transition $\delta_D$:

$$\delta_D(q_D, a) = \Big\{ S(q_D, a) = \bigcup_{q \in q_D} \delta^*_N(q, a) \Big\}, \quad \forall q_D \in Q_D,\ \forall a \in \Sigma.\ [q_D = S \text{ in Step 2i) }]. \qquad (2.5)$$

NOTES:

1. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.
2. An important conclusion we can draw from ALGORITHM 2.1 is that every language accepted by an NFA is regular.

♦ **EXAMPLE 2.16.**

Given a NFA $M_N = (Q_N, \Sigma, I \ni q_0, \delta_N, F_N)$ [top-left in next figure]

where $Q_N = \{0, 1\}$, $\Sigma = \{a, b\}$, the start $I = \{0\}$, the final $F_N = \{1\}$,

and the transition function $\delta_N$ is given by table below,

| $\delta_N$ | a | b |
|---|---|---|
| $\to$ 0 | 1 | $\{0, 1\}$ |
| 1 | 0 | 1 |
| $1^*$ | 0 | 1 |

Initialize $S_0 := \{0\}$, [since NFA has 1 start state], graph $T_D$ has initial vertex $S_0$.

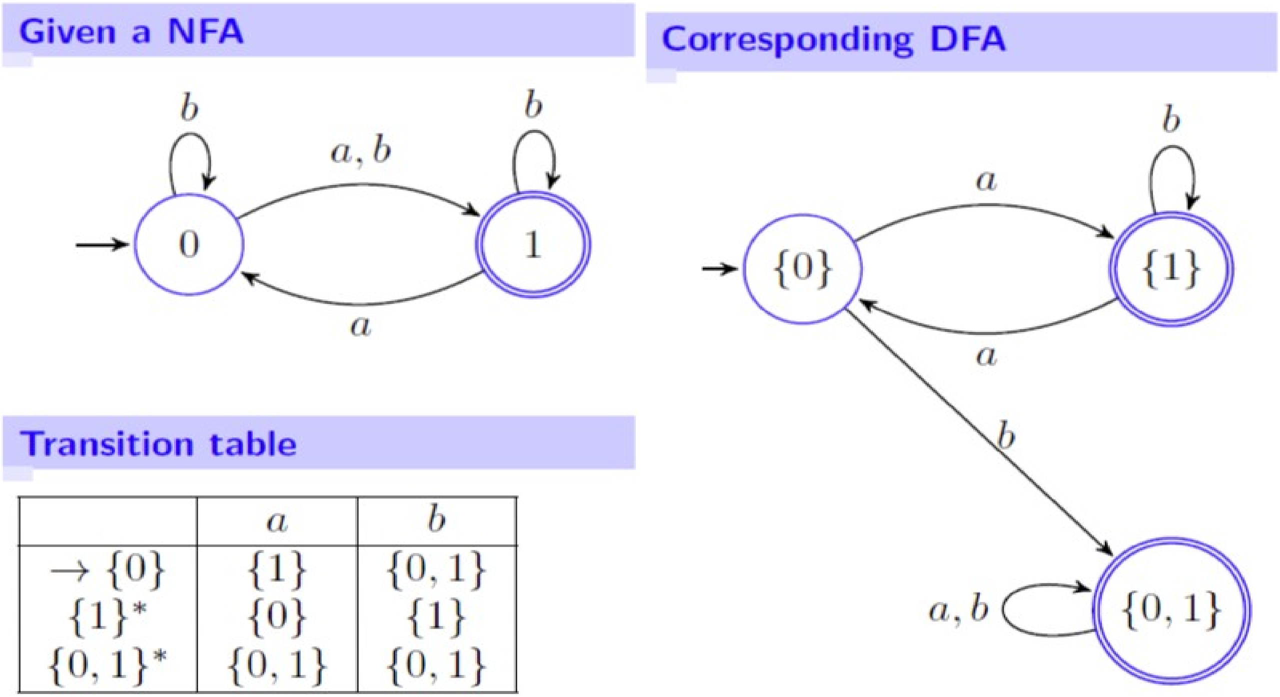
Loop $n = 1$: $S1 := \{0\}$, [use $b$], next use $a$ on $0$ shows $S2 := \{1\} = F_N$, but $S2$ becomes final in $T_D$;

Loop $n = 2$: $\delta_N^*(1, a) = \{0\}$ and $\delta_N^*(1, b) = \{1\}$, gives us $S3 := \{0, 1\}$, now becomes final in $T_D$;

Repeat steps ... till no more edges are generated.

The result DFA $M_D$ has the states $Q_D = \{S_0, S_2, S_3\} = \{\{0\}, \{1\}, \{0, 1\}, \}$,

the final states $F_D = \{\{1\}, \{0, 1\}\}$, the transition $\delta_D$ is in the left bottom of figure below.



Transition table

| | $a$ | $b$ |
|---|---|---|
| $\rightarrow \{0\}$ | $\{1\}$ | $\{0,1\}$ |
| $\{1\}^*$ | $\{0\}$ | $\{1\}$ |
| $\{0,1\}^*$ | $\{0,1\}$ | $\{0,1\}$ |

**Definition 2.12 (*Accessible states* )**

*A state $p$ is **accessible** in a* DFA *$M = (Q, \Sigma, \{q_0\}, \delta, F)$ if it is on a walk which starts by an initial state: $q_0 \to \cdots p \to \cdots$ .*

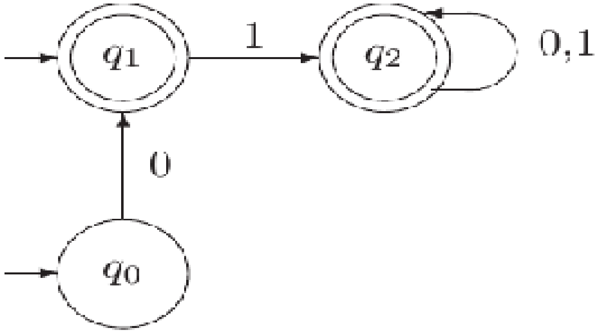
*RULE: The **inaccessible** states [not on any walk which starts by an initial state] of $M$ can be eliminated without changing accepted language.* ♣

- Reminder: accessible and productive states are different.

  A state $p$ is **productive** if it is on a walk which ends in a final state: $\cdots \to p \to q_n \in F$.

**PRACTICE 2.7.** *(I) Remake EXAMPLE 2.15.*

*(II) Use ALGORITHM 2.1 to transform the NFA in the figure below to a DFA $M_D$.*



A nondeterministic finite automaton with 3 states

**GUIDANCE for solving.**

Build sets $S_n$ as the states of the DFA, note that $q_1$ is both initial and final node.

Initialize $S_0 := \{q0, q1\},$ [since NFA has 2 start states], graph $T_D$ has initial vertex $S_0$.

Loop $n = 1$: $S1 := \{q0\}$, [use $\varepsilon$], $S2 := \{q1\}$, [use 0 on $q_0$, $S2$ become final in $T_D$];

Loop $n = 2$: $\delta^*_N(q_1, 1) = q_2$ gives $S3 := \{q2\},$ [become final in $T_D$];

Similarly $n \geq 3$: $S4 := \{q0, q2\},\ S5 := \{q1, q2\},\ S6 := \{q0, q1, q2\},$ where $S_0$ is the initial state.

Make the transition function $\delta_D$ with Equation 2.5

$$\delta_D(q_D, a) = \left\{ \bigcup_{q \in q_D} \delta_N^*(q, a) \right\}, \ \forall q_D \in Q_D, \ \forall a \in \Sigma. \ [q_D = S \text{ in Step 2i}) \ ]$$

we get the transition table

| $\delta_D$ | 0 | 1 |
|---|---|---|
| $\rightarrow \ S_0$ | $\{S_2\}$ | $\{S_3\}$ |
| $S_1$ | $\{S_2\}$ | $\emptyset$ |
| $S_2$ | $\emptyset$ | $\{S_3\}$ |
| $S_3$ | $\{S_3\}$ | $\{S_3\}$ |
| $S_4$ | $\{S_5\}$ | $\{S_3\}$ |
| $S_5$ | $\{S_3\}$ | $\{S_3\}$ |
| $S_6$ | $\{S_5\}$ | $\{S_3\}$ |

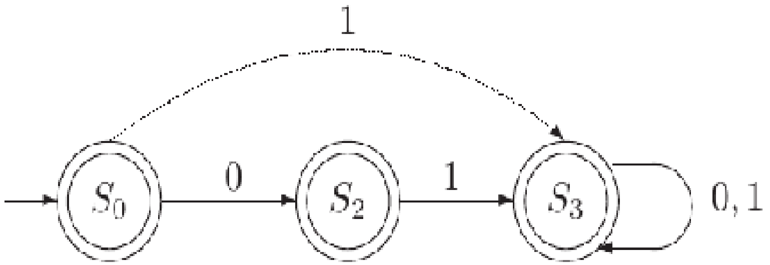
This automaton contains many inaccessible states, $S_1, S_4, S_5, S_6$, and

deleting them determines the accessible states of DFA:

$U_0 = \{S_0\}, \ U_1 = \{S_0, S_2, S_3\}, \ U_2 = \{S_0, S_2, S_3\} = U_1 = U.$

The transition table of the resulted DFA $M_D$ is

| $\delta_D$ | 0 | 1 |
|---|---|---|
| $\rightarrow\ S_0$ | $\{S_2\}$ | $\{S_3\}$ |
| $S_2$ | $\emptyset$ | $\{S_3\}$ |
| $S_3$ | $\{S_3\}$ | $\{S_3\}$ |



The equivalent DFA with NFA above

**Figure 2.7:** *The output DFA from the above input NFA*

The corresponding transition graph $T(M_D)$ is in Fig 2.7. ■

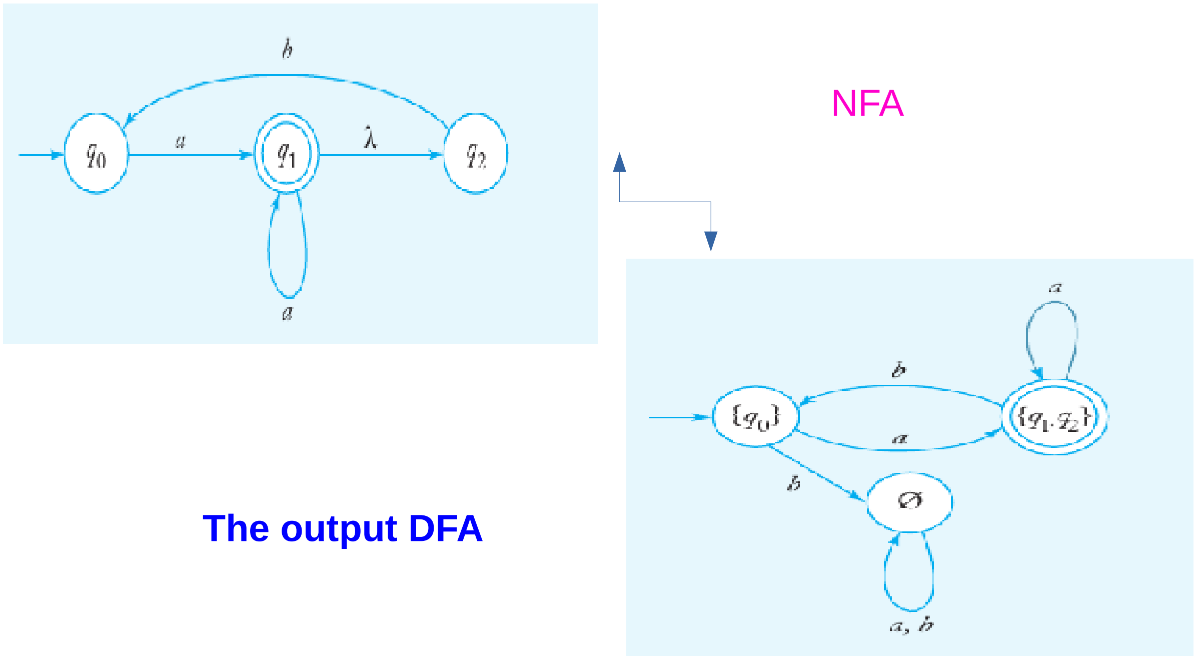
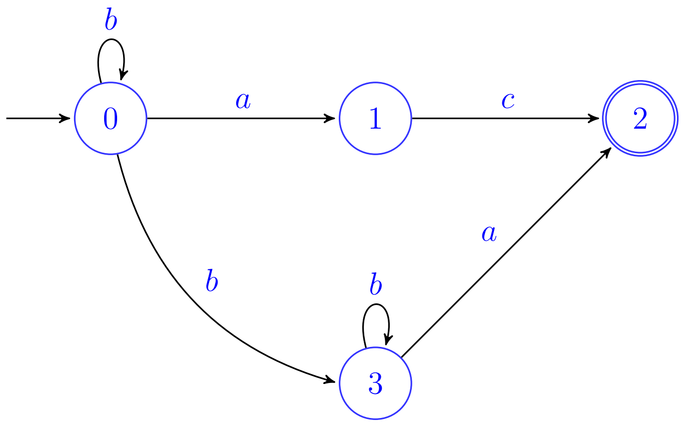


**Figure 2.8:** *NFA-to-DFA*

**PRACTICE 2.8.**

A/ Use ALGORITHM 2.1 to transform the NFA in Figure 2.8 to a DFA $M_D$.

B/ Let $\Sigma = \{a, b, c\}$. Determine DFAs which corresponds to the following NFA.



C/ Do Exercises on page 4.4x, $x = 6, 7, 8$ in REF.1.

### 2.6.2 Minimization of a DFA

♣ **OBSERVATION 5.**

- Any DFA defines a unique language, but the converse is not true.

  For a language $L$, there are many DFA's $M$ that accept it: $L = L(M)$.

  There may be a considerable difference in **the number of states** of such equivalent automata.
- Representation of an automaton for computation requires space proportional to the number of states, it is desirable to reduce the number of states as far as possible, a **storage minimization problem**.
- Reducing the states of a DFA is based on finding and combining *indistinguishable* states.

**Definition 2.13**

*Fix a machine $M = (Q, \Sigma, q_0, \delta, F)$, and $\delta^*$ the extended map of $\delta$.* ♣

**Distinguishablity:** States $p$ and $q$ of $M$ are called *distinguishable*

$DIS$ : if there exists some string $w \in \Sigma^*$ such that

$$\delta^*(p, w) \in F \quad \text{and} \quad \delta^*(q, w) \notin F.$$

`[I go to final, and You not]`

**Indistinguishablity:**

Two states $p$ and $q$ of $M$ are called *indistinguishable* if $\overline{DIS}$ happens:

$$\forall w \in \Sigma^* : \ \delta^*(p, w) \in F \Leftrightarrow \delta^*(q, w) \in F \quad \texttt{[we are in the same class!]}$$

#### ELUCIDATION

- Hence, **indistinguishablity defines a 'binary relation'** on the set $Q$.

At the original automaton, the set of final states $F$ makes a 'equivalence class', and the complement $F^c = Q \setminus F$ makes another 'equivalence class'.

- Under actions of symbols in $\Sigma$ and words in $\Sigma^*$ the states in $Q$ change their equivalence classes, until a stopping status occurs.
- How to define the `stopping condition/status`? The process must terminate and it determines all pairs of distinguishable states, **we get a minimum DFA**.

REMINDER

**Proposition 2.3**

- *An equivalence relation $R \subset K \times K$ satisfies 3 properties: reflexive, symmetric and transitive.*

  *Given an equivalence relation $R$, the equivalence class of state $p \in K$ is*

$$R_p = \{q \in K : \ q \, R \, p\} = \{q \in K : \ (p, q) \in R\}.$$

- *On any set $K$, if we could define an equivalence relation $R$*

  *then we can build up equivalence classes $\{R_k\}$ and*

  *they give a partition of $K$: $K = \bigcup_{k \in K} R_k$.* ♠

## Multi-steps approach for CONSTRUCTION of Equivalent automata

- **ALGORITHM 2.1**: Transform an NFA $M_0$ to a new DFA $M$; done.

Then $\mathrm{min}$ (DFA) Problem means Minimization of $M$ to a minimal DFA $M_1$.

(Minimal in the sense of minimum number of states).

Minimization of automaton $M$ includes

- **ALGORITHM 2.2**: Find all equivalence classes of $M$

Recall that :

(I) States $p, q \in K$ are **indistinguishable** when they are in one *equivalence class*.]

(II) The **inaccessible states** [not on any walk which starts by an ini tial state] of $M$ can be eliminated without changing accepted language

- **ALGORITHM 2.3**: Construct the minimal DFA $M_1$.

**Algorithm 2.2 Equivalence-Classes-DFA** $(Q, \Sigma, \{q_0\}, \delta, F)$ - Procedure **mark**, REF.2.

*Input*: A DFA $M = (Q, \Sigma, \{q_0\}, \delta, F)$.

*Output:* all equivalence classes of nodes in the graph $T(M)$ of $M$.
REMARK: Indistinguishability classes of machine $M$ are equivalence classes of nodes in graph $T(M)$.

**1) Avoid redundancy:** Remove all *inaccessible* states, by

enumerating all *simple paths* of $T(M)$ starting at the initial state $q_0$. Any state **not** part of some path is inaccessible.

**2) Loop** $n = 1$**: Create equivalence classes:** Consider all pairs of states $(p, q)$ in $T(M)$.

If $p \in F$ and $q \notin F$ or vice versa, we mark the pair $(p, q)$ as distinguishable (they are in distinct equivalence classes). So there are at least two equivalence classes.

**3) Loop $n \geq 2$ to find all equivalence classes of states:** Repeat the following step:

- For all pairs $(p, q)$ and all symbol $a \in \Sigma$, compute

  $\delta(p, a) = p_a$ and $\delta(q, a) = q_a$.

- If the new pair $(p_a, q_a)$ is marked as *distinguishable*, we mark $(p, q)$ as *distinguishable*.

**Until Stopping condition valid:** No previously unmarked pairs are marked. In other words, denote $S_n$ and $S_{n+1}$ be the sets of equivalent classes at loop $n$,

if $S_n = S_{n+1}$, then stop and return $S_n$ as an equivalent partition.

♦ **EXAMPLE 2.17** (Use ALGORITHM 2.2 to find indistinguishability classes)**.**

Consider the top-left automaton $M$ as follows.

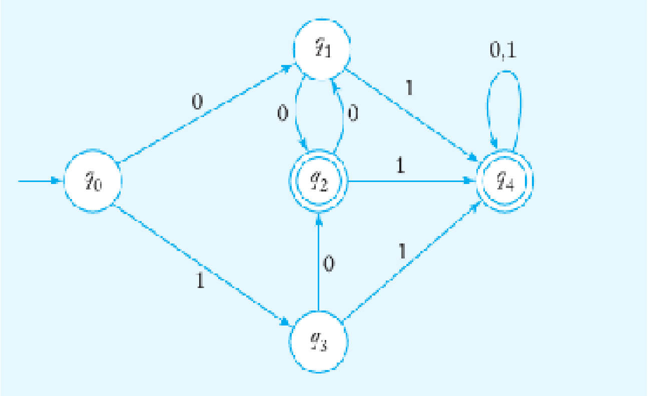
Indistinguishability classes of machine $M$ are equivalence classes of nodes in graph $T(M)$.

In Step 2 we get two equivalence classes $I = \{q_2, q_4\} = F$ and $II = \{q_0, q_1, q_3\} = F^c$.

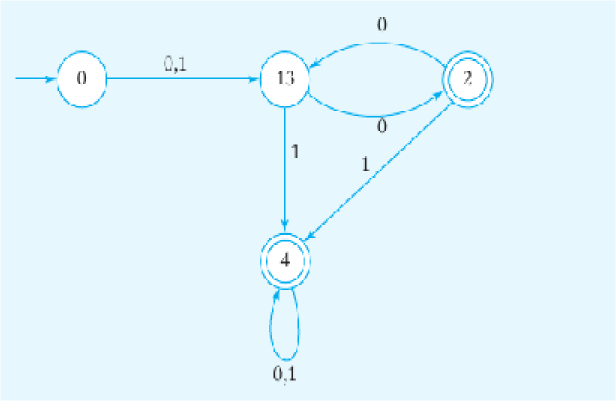
In Step 3, use action $a = 0$ on states $q_0, q_1$:

$\delta(q_0, 0) = q_1$ and $\delta(q_1, 0) = q_2$, the outputs are **distinguishable**,

so we must split the old $II = \{q_0, q_1, q_3\}$ into new $II = \{q_0\}$ and $III = \{q_1, q_3\}$.



Given a deterministic automaton M



We find a minimum DFA M1 of M →

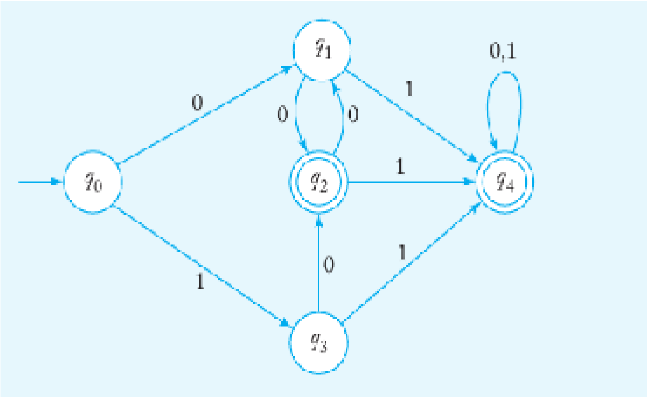
Next on states $q_2, q_4$:

$\delta(q_2, 0) = q_1$ and $\delta(q_4, 0) = q_4$, the outputs are **again distinguishable**,

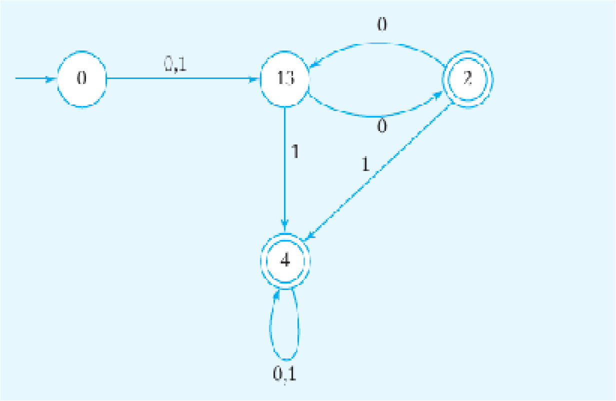
so we must split class $I = \{q_2, q_4\}$ into new $I = \{q_2\}$ and $IV = \{q_4\}$.

No further splitting is needed, obtain 4 nodes in total.

We could identify (label) $I$ as number 2, node $II$ as 0, $III$ as series $1\ 3\ldots$ ■



Given a deterministic automaton M



We find a minimum DFA M1 of M

**ALGORITHM 2.3** [Construction of the minimal DFA- Procedure **reduce**, REF.2.]

**Algorithm 2.3 Construction of the minimal** DFA$(Q, \Sigma, \{q_0\}, \delta, F)$)

*Input*: A DFA $M = (Q, \Sigma, \{q_0\}, \delta, F)$.

*Output:* A reduced DFA $M_1 = (Q_1, \Sigma, q_*, \delta_1, F_1)$ such that $|Q_1| \leq |Q|$.

## 1) Find distinguishable classes:

Use ALGORITHM 2.2 to generate all distinct equivalence classes, typically written as $C_k =$

$\{q_i, \dots, q_k\}$

**2) Create a state in $M_1$:**

For each set $C_k = \{q_i, q_j, \dots, q_k\}$ [states $q$ are indistinguishable], we make a state labeled $i\ j\ \cdots\ k \in Q_1$ of $M_1$.

**3) Define the transition rule $\delta_1$ of $M_1$:**

For each transition in $M$ of the form $\delta(q, a) = p$ (letter $a \in \Sigma$) we find the sets to which $q$ and $p$ belong in.

If $q \in C_k = \{q_i, q_j, \dots, q_k\}$ and $p \in C_n = \{q_l, q_m, \dots, q_n\}$

where class $C_n$ disjoint class $C_k$ we then symbolically set

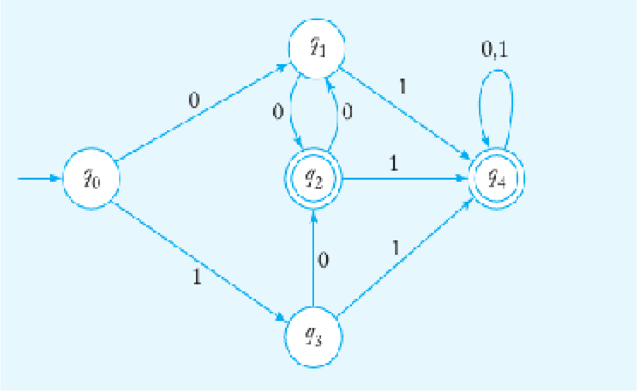
rule $\delta_1$ as $\delta_1(C_k, a) = C_n$, or $\delta_1(i\ j\ \cdots\ , a) = l\ m \cdots\ n.$

**4) Locate the new initial state $q_*$:**

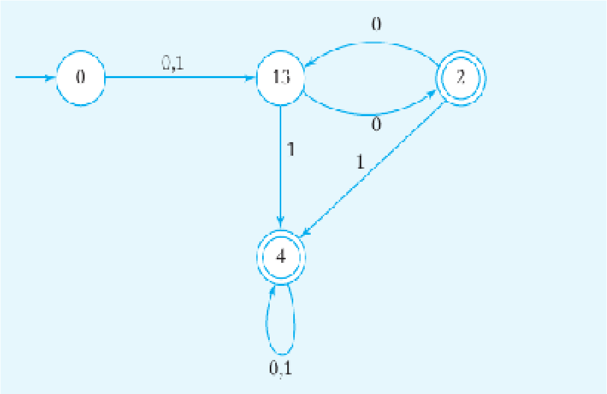
New initial state $q_*$ of $M_1$ is one of state $C_k$ including the 0.

**5) Build the new final set $F_1$:** $F_1$ of $M_1$ is the set of all the states whose label contains $i$ such that $q_i \in F$.

♦ **EXAMPLE 2.18** (Use ALGORITHM 2.3 to find min DFA).



Given a deterministic automaton M



We find a minimum DFA M1 of M →

From EXAMPLE 2.17, steps 1) and 2) are done by ALGORITHM 2.2.

In step 3) we define the transition rule for the bottom-right automaton $M_1$.

For example with $a = 0$, choose $C_k = III = \{q_1, q_3\}$,

then $\delta(q_1, 0) = q_2 \in I$, and class $I \cap C_k = \emptyset$.

So there is an edge labeled 0 from state $III \equiv\ 1\,3$ to state $I \equiv 2$.

The remaining transitions are similarly found, giving the function $\delta_1$ of the reduced $M_1$.

How about the node of initial state $q_*$ and $F_1$?

Step 4): the class $II = \{q_0\}$ in $T(M)$ give node initial state $q_* = 0$ in $M_1$.

Step 5): $F = \{q_2, q_4\}$ so the new final set of $M_1$ is

$$F_1 = \{C_n : i \in C_n \ \wedge q_i \in F\} = \{2, 4\}.$$

We obtain fully the reduced DFA $M_1 = (Q_1, \Sigma, q_*, \delta_1, F_1)$.

See slides **4.50 and 4.51 in REF.1** for more example.

---

## 2.7 Determine regular expression given automata

**Recall Problem 4: finding a regular expression given an automaton**

Definition 2.11 suggests that **the connection**

between *regular languages* and *regular expressions* [as in Definition 2.3]

is a mutually close one. The two concepts are essentially the same:

(i) for every regular language there is a regular expression, and

(ii) for every regular expression there is a regular language.

### 2.7.1 Regular Expressions denote Regular Languages

Because of the equivalence of NFA's and DFA's, a language is also regular if it is accepted by some NFA.

Theorem 2.1 claims the existence of a NFA$M(r)$, given $r$:

"If we have any regular expression $r$, we can construct an nondeterministic finite automaton (NFA) that accepts the language $L(r)$."

♦ **EXAMPLE 2.19.** *Given 3 states* $Q = \{q_0, q_1, q_2\}$, *and the alphabet* $\Sigma = \{a, b\}$,

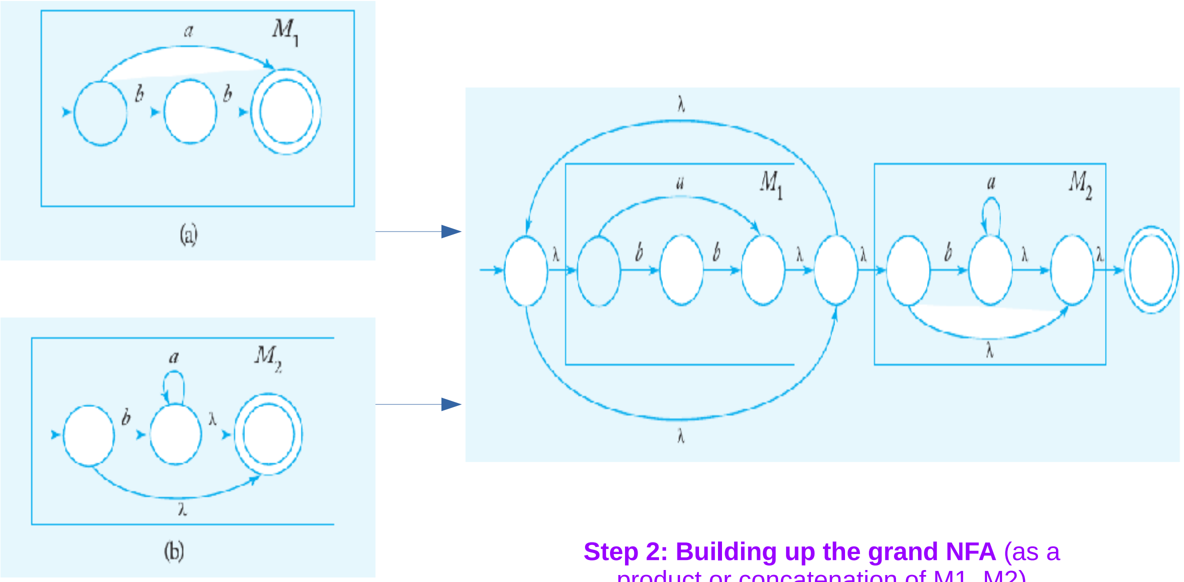
we construct an NFA that recognizes the language $L(r)$ where the regular

$$r = (a + bb)^* \, (ba^* + \varepsilon).$$

We can use this theorem to build $M = M_1 \cdot M_2$, **the concatenation of** $M_1, M_2$. Indeed, $M_1$ [fig. (a)] accepts the language $L_1 = L((a + bb)^*)$,

$M_2$ [fig. (b)] accepts the language $L_2 = L(ba^* + \varepsilon)$.

The product language $L = L_1 \, L_2$ is accepted by the NFA $M$.



**Step 1: Construct NFA of compnents,** M1 accepts L1= L(a+bb)...

**Step 2: Building up the grand NFA** (as a product or concatenation of M1, M2)

The next part is the converse of Theorem 2.1 [THEOREM 3.2 of REF. 2] .

### 2.7.2 Regular Languages gives Regular Expressions

**Theorem 2.3**

*For every regular language $L$, there* ***exists a corresponding regular expression*** *$r$ such that $L = L(r)$.* ♡

- FACT: Any regular language $L$ has an associated NFA $M$, and hence, a transition graph $T(M)$ is fully determined by $L$.

  Therefore, given $L$, we need **to find a regular expression** $R$ capable of generating the labels of all the walks (in the graph $T(M)$) from $q_0$ to any final state.
- COMPLICATION: by the existence of cycles in $T(M)$ that can be traversed arbitrarily.
- IDEAS: use generalized transition graphs (GTG).

  The graph $T(M)$) of any NFA $M$ can be considered a new *generalized transition graph* $GT(M)$ if the edge labels are interpreted properly.

  An edge labeled with a single symbol $a$ is interpreted as an edge labeled with the expression $a$, while an edge labeled with *multiple* symbols $a, b, \ldots$ is interpreted as

  an edge labeled with the expression $a + b + \ldots$.

**Definition 2.14 (Generalized Transition Graph (GTG))** ♣

(I) A generalized transition graph $GT(M)$ of an NFA $M$ is a transition graph whose **edges are labeled with regular expressions**.

- The label of any walk from the initial state to a final state is the concatenation of several regular expressions, and hence itself a regular expression, see Equation 2.6.
- The strings denoted by such regular expressions are a subset of the language accepted by the generalized transition graph $GT(M)$, with the full language being the union of all such generated subsets.

(II) A **complete GTG** is a graph in which all edges are present.

**Property 2.1.**

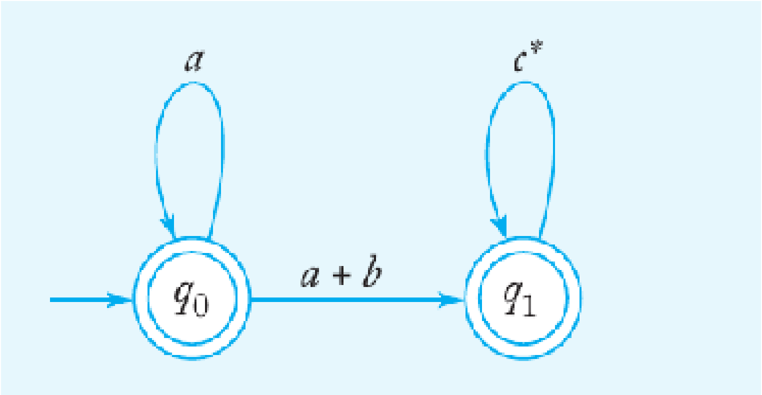
1. If a GTG $GT(M)$, after conversion from $M$, has some edges missing, we put them in and label them with $\emptyset$. A complete GTG with $|V|$ vertices has exactly $|V|^2$ edges.
2. When a GTG has more than two states, we can find an equivalent graph by removing one state at a time, until only two states are left.

♦ **EXAMPLE 2.20.** *Figure 2.9 represents a generalized transition graph* $GT(M)$.

The language accepted by $GT(M)$ is $L = L(R) = L(a^* \ + a^* \ (a+b) \ c^*)$,

as clearly seen from an inspection of the graph.

The edge $(q_0, q_0)$ labeled $a$ is a cycle that can generate any number of $a$'s, it represents $L(a^*)$.



A simple a generalized transition graph of language L, giving the regular expression R

**Figure 2.9:** *A simple GTG, language $L$ and its regular expression*

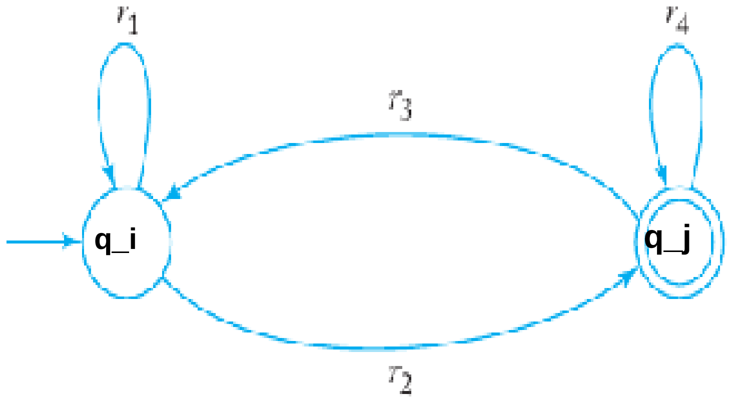
We could have labeled this edge $a^*$ without changing the language accepted by the graph. (since the label of every walk in a GTG is a regular expression). ■

**PRACTICE 2.9.** *Now we have the simple two-state complete GTG shown in Figure 2.10.*

Can we check that the regular expression $R$ from this GTG is

$$R = r_1^* \, r_2 \, (r_3 \, r_1^* \, r_2 + \; r_4)^* ? \tag{2.6}$$

Using double indexes for edges, in this complete GTG with 4 arcs, we can replace



This automaton M returns a regular expression?

**Figure 2.10:** *A simple* NFA *and its regular expression?*

regular expression $r_1^*$ by $r^*_{i\;i}$ , $r_2$ by $r_{i\;j}$ , $r_3$ by $r_{j\;i}$ , and finally $r_4$ by $r_{j\;j}$. ■

**ALGORITHM 2.4** [Construct regular expression from NFA- Procedure **nfa-to-rex**, REF.2.]

*Input*: An NFA $M = (Q = \{q_0, q_1, \ldots, q_n\}, \Sigma, \{q_0\}, \delta, F)$, $|F| = 1$, or a language $L$

*Output:* A regular expression $R$ such that $M = M(R)$ or $L = L(R)$

[ IDEA: Perform the process $M \longrightarrow GT(M) \longrightarrow R$]

**(1) Conversion:** Convert $M$ into a *complete GTG* $GT(M) = (V, A)$.

Put label $r_{i\ j}$ to the edge from $q_i \in V$ to $q_j \in V$.

**(2) Checking the number of nodes and making decision:**

**2a**): If the GTG has only two states, with $q_i$ as its initial state and $q_j$ its final state, its associated regular expression [extended from Equation 2.6] is

$$R = r_{i\ i}^{*}\, r_{i\ j} \left(r_{j\ i}\, r_{i\ i}^{*}\, r_{i\ j} + r_{j\ j}\right)^{*} \tag{2.7}$$

**2b)** If the GTG has three states, with initial state $q_i$, final state $q_j$, and third state $q_k$, we create new edges (between $q_i, q_j$), labeled

$$r_{p\ k}\, r_{k\ k}^{*}\, r_{k\ q} + r_{p\ q} \equiv r_{p\ q} + r_{p\ k}\, r_{k\ k}^{*}\, r_{k\ q}, \qquad p = i,\ j, \quad q = i,\ j; \tag{2.8}$$

and remove vertex $q_k$ and its associated edges.

**2c**) If the GTG has at least 4 states, pick a state $q_k$ to be removed.

Apply **rule 2b)** for all pairs of states $(q_i, q_j)$, $i \neq k$, and $j \neq k$. At each step apply the simplifying rules [wherever possible, when this is done, remove state $q_k$]:

$$\begin{cases} r + \emptyset = r, \\ r \cdot \emptyset = r, \\ \emptyset^* = \varepsilon, \end{cases}$$

**(3) Loop and Stop:**

Repeat Step 2 until the correct regular expression is obtained.

Therefore, for arbitrary GTGs we remove one state at a time

until only two states are left.

Then we apply Equation (2.6) to get the final regular expression.

♦ **EXAMPLE 2.21** (Illustration for Step **2b**). *Figure 2.11 represents a reduction of a complete generalized transition graph $GT(M)$.*

To remove $q_2$ , we first introduce some new edges, exploit Equation 2.8, $i = 1$, $j = 3$:

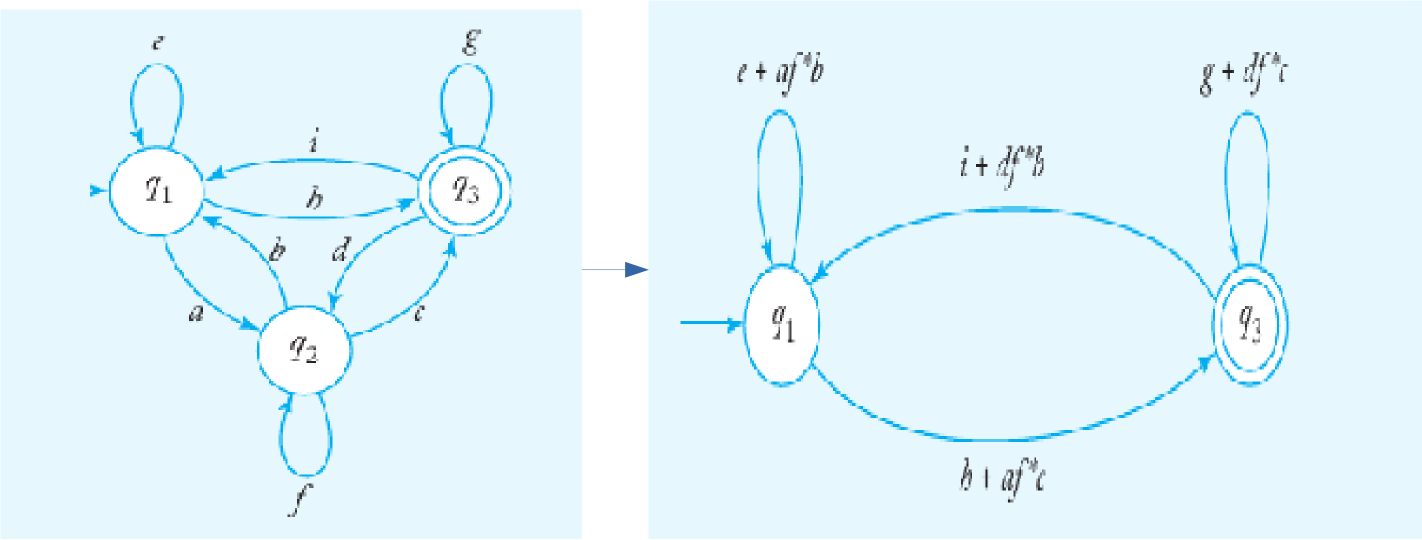
create loop at $q_1$ : $e+\ a\ f^*b,$ then create loop at $q_3$ : ?

edge from $q_1$ to $q_3$ : $h+\ a\ f^*c,$

edge from $q_3$ to $q_1$ : $i+?$

When this is done, we remove $q_2$ and all associated edges,

and get new GTG $G_1$ in Figure 2.11.II. ■



(I) A complete GTG G0

**(II) The new and equivalent GTG G1, obtained from G0, by removing q2**

**Figure 2.11:** *Reduce node of a complete GTG*

**REMINDER**:

1) ALGORITHM 2.4 (NFA to regular expression) essentially includes

Step **1**: Converting FA $M$ or language $L$ to a complete GTG $GT(L)$

Step **2**: Reducing complete GTG and making regular expression $R$.

2) Notation $n_x(w) =$ the number of letter $x$ in string $w$.

♦ **EXAMPLE 2.22.** *Find a regular expression $R$ for the language*

$$L = \{w \in \{a, b\}^* : \ n_a(w) \text{ is even, and } \ n_b(w) \text{ is odd}\}.$$

**Define the states (nodes) $V$ and the arcs $A$ of the graph $GT(L) = (V,\ A)$?**

1. Step **1**: Converting a language to a complete GTG

   In the graph $GT(L) = (V,\ A)$, $V$ has 4 nodes

$$\{q_1 = EE, q_2 = OE, q_3 = OO, q_4 = EO)\}$$

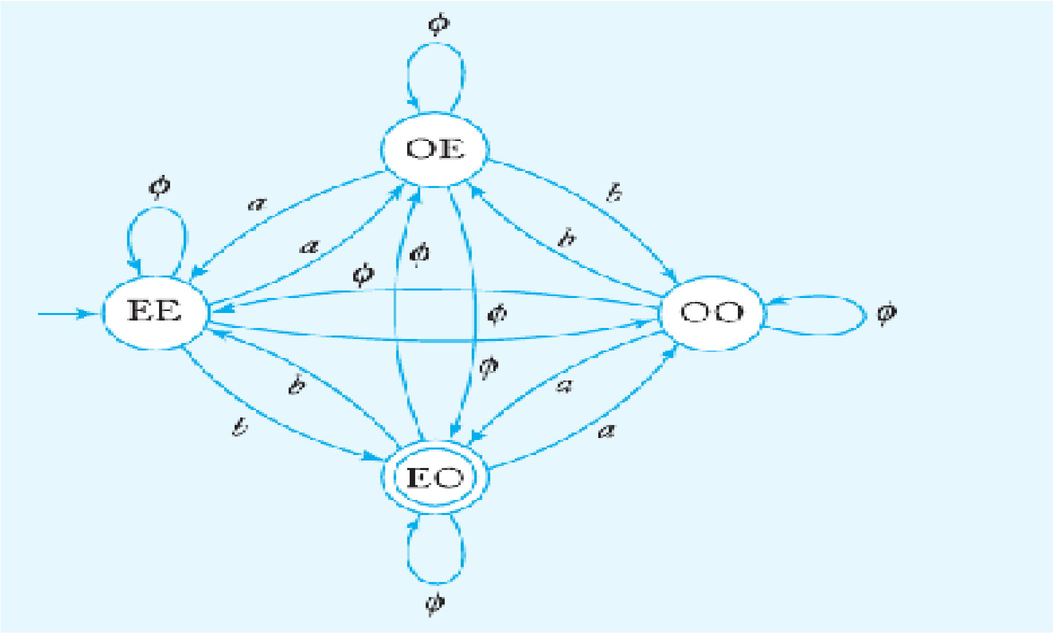
where

$EE =$ the node of strings $w$ with an even number of symbol $a$'s and $b$'s,

$OE =$ the node of strings $w$ with an odd number of $a$'s, and even number of $b$'s,

$OO =$ the node of strings $w$ with an odd number of symbol $a$'s, and $b$'s,

$EO$ denote the node of strings $w$ with an even number of $a$'s, and odd number of $b$'s.

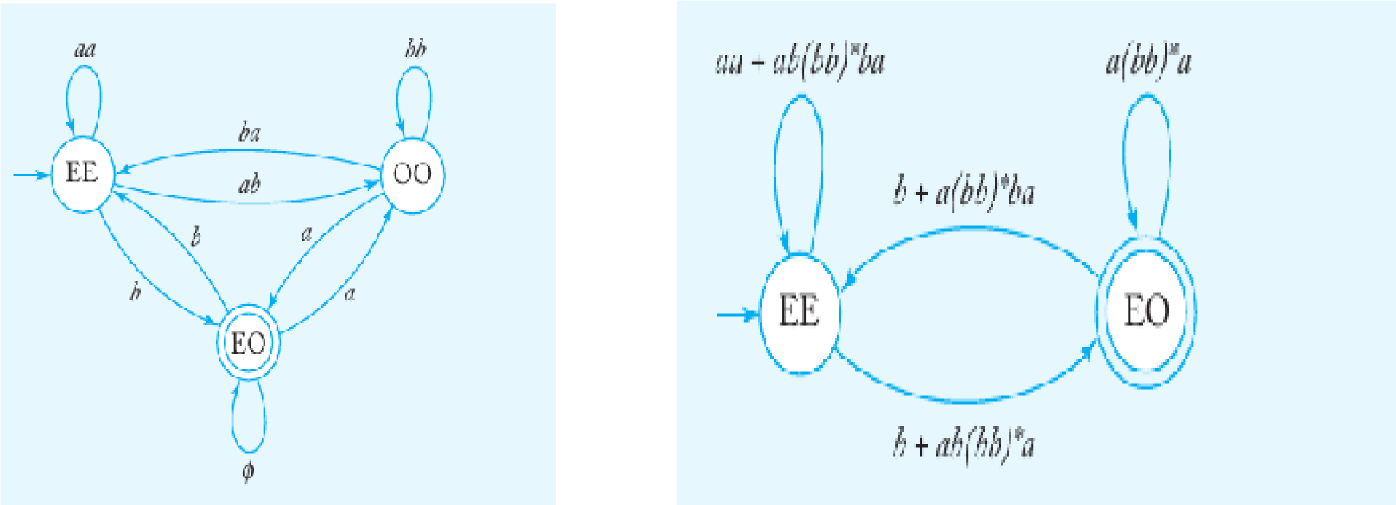


**The complete GTG of L has 4 nodes, and 4^2 = 16 arcs**

**Figure 2.12:** *A complete GTG with 4 nodes*

**Define the arcs $A$?**

The graph $GT(L) = (V,\ A) =: G_0$ is complete, so $|A| = 16$ arcs, as in Figure 2.12. We represented a conversion of $L$ to a complete generalized transition graph $GT(L)$.



G2 = G_1 \ node OO

G1 = G_0 \ node OE

**Figure 2.13:** *Reduced GTG with 3 and 2 nodes, with output of regular expressions*

2. Step **2**: Reducing complete GTG and making regular expression $R$

GTG has 4 states, pick a state $q_2 = OE$ to be removed, use Step **2c**).

The loop at $q_1$ (edge between $q_1 = EE$ and itself) in the reduced graph, will have the label

$$r_{EE} = \emptyset + \ a\ \emptyset^*\ a = aa$$

[by using Eqn. 2.8 : $r_{p\ q} + \ r_{p\ k}\ r^*_{k\ k}\ r_{k\ q},\ p = i,\ j;\ q = i,\ j.$]

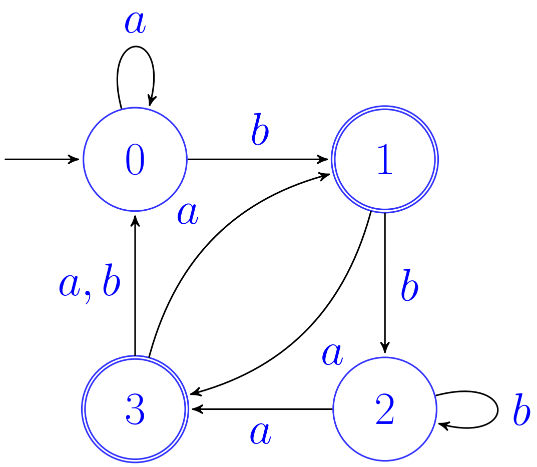
Similarly [DIY] we can find new edges of the reduced graph, after removing $q_2 = OE$ we get a new graph with 3 nodes (left) and then 2 nodes (the right) of Figure 2.13.

Equation 2.7 provides the correct regular expression $R = b + \ ab\ (bb)^*\ a$.

**PRACTICE 2.10.**

Find a regular expression corresponding to a given finite automaton $M$.

## The transition graph $T(M)$



HINT: Find the GTG $GT(M)$ from $T(M)$, then apply Step 2 of ALGORITHM 4.

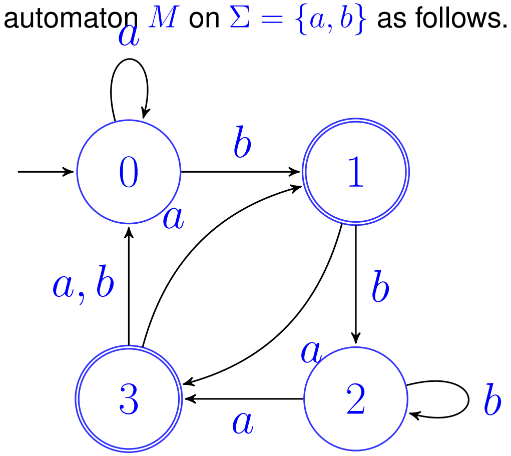
**Answer**:

With $X = a^*b + a^*b(a + bb^*a)a$; $Y = (a + bb^*a)$ ;

then $R = X(Y(a + b)X)^* + XY((a + b)XY)^*$.

## 2.8 Reviewed Problems

1. Consider the language $L$ determined by an automaton $M$ on $\Sigma = \{a, b\}$ as follows.



a/ **Choose** a better conclusion between statements:

i) This automaton is **not** a DFA since the number of states is not finite.

ii) This automata is a NFA since it is not deterministic.

b/ **Is this** automaton a NFA or DFA? Why?

c/ **Which of** the following strings is valid (accepted): $aabb$, $aababbab$, or $aabba$?

d/ **Which string** is not valid? $ababab$, $aabbbaabbab$, or $aabbbbaaa$?

e/ **Which regular expression** Z corresponds to the considering finite automaton

(i) $X = a^*b$; $Y = X(a + bb^*a)$ ; $Z = X(Y(a + b)X)^*$

(ii) $X = a^*b + a^*b(a + bb^*a)a$; $Y = (a + bb^*a)$ ; $Z = X(Y(a + b)X)^* + XY((a + b)XY)^*$

f/ When using determinisation algorithm [ALGORITHM 1] to convert NFA into DFA, **how many** states are there in the new DFA?

$6,$ $7,$ $10,$ or `None of the others.`

g/ **How many** states are there in the minimized/optimized DFA (which is equivalent to the above NFA)? $6,$ $7,$ $10,$

or `None of the others.`

2. **Find** the correct statement

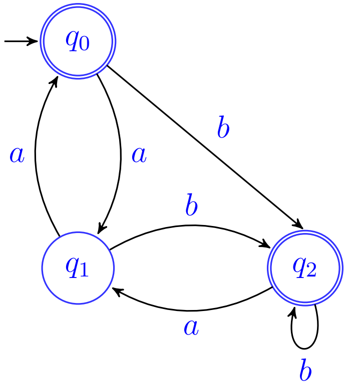
a) When occuring an event from a state, the NFA does not determine the next state.

b) NFA has not finite number of states but DFA has a finite number of states .

c) The number of states is always reduced when determinisation from NFA to DFA.

d) NFA does not determine surely the next state in order to simplify the graph.

3. Do the following automaton $M$



and **regular expression** $E = ((aa)^* + bb^*a(aa)^*b(ab)^*)^*$ **present the same language** $L$? If not, give a counter-example.

Your choice:

(a) They are equivalent.

(b) They are not equivalent.

The counter-example is ____________________$baa$?.

4. Are two regular expressions $E_1 = (a + b)^*$ and $E_2 = (aa + ab + ba + bb)^*$ are equivalent? If not, give a counter-example.

   Your choice:

   (a) `They present the same language`

   (b) $E_1 \subseteq E_2$

   (c) `They are not equivalent, the counter-example is` ______________$a$?.

5. Let's consider $\Sigma = \{a, b, c\}$ and language $L = \{a, aab, bbc, ba\}$.

   Which string does **not** belong to $L^4$?

   i) $aababbc$; ii) $baaaaab$

   iii) $abaaabba$ ; iv) $abbcaab$