Regular Languages and Regular Grammars

Syllabus – Module 2

Module -2	
REGULAR LANGUAGES, REGULAR GRAMMARS AND PROPERTIES OF REGULAR LANGUAGES: Regular Expressions, Connection between Regular Expressions and Regular Languages, Regular Grammars, Closure Properties of Regular Languages, Identifying Non-regular Languages. **Text Book 1:** Chapter 3: 3.1 -3.3, Chapter 4: 4.1,4.3	**07 Hours**

Module 2

Introduction:

- One way of describing regular languages is via the notation of regular expressions.
- This notation involves

→a combination of strings of symbols from some alphabet Σ,

→parentheses,

→operators +,., and *.

Examples:
1. RE for the language L = {a} is a.
2. RE for the language L = {a, b, c} is a+b+c
3. RE for the language L = {λ, a, bc, aa, abc, bca, bcbc, aaa, aabc,… } is (a + bc)*

- The expression (a + (b·c))* stands for the star-closure of {a} ∪ {bc}, that is, the language {λ, *a*, *bc*, *aa*, *abc*, *bca*, *bcbc*, *aaa*, *aabc*,…}.

→+ to denote union, · for concatenation and * for star-closure.

Every regular language can be described by some DFA or some NFA. So, a language is said to be regular, if there exist a finite accepter for it.

Formal Definition of a Regular Expression

- Let Σ be a given alphabet, Then

1. Ø, λ and $a \in \Sigma$ are all regular expressions. These are called **primitive regular expressions.**

2 If $r1$ and $r2$ are regular expressions, so are $r1+ r2$, $r1.r2$, r1*, and ($r1$).

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).

- c + Ø and (c + Ø) are regular expressions.
- On the other hand, (a + b +) is not a regular expression.

Languages Associated with Regular Expressions

- The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. Ø is a regular expression denoting the empty set,

2. λ is a regular expression denoting {λ}.

3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

If $r1$ and $r2$are regular expressions, then

4. $L(r1 + r2) = L(r1) \cup L(r2)$,

5. $L(r1 \cdot r2) = L(r1)\, L(r2)$,

6. $L((r1)) = L(r1)$,

7. $L(\mathrm{r1}^*) = (L(r1))^*$.

Languages Associated with Regular Expressions

- Consider, for example, the regular expression $a \cdot b + c$.
- We can consider this as

  $r1 = a \cdot b$ and $r2 = c$

  In this case, we find $L(a \cdot b + c) = \{ab, c\}$.

- By taking $r1 = a$ and $r2 = b + c$, We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$.

So, which one is correct?

- To avoid such confusions, we establish a set of precedence rules for evaluation in which

  star-closure precedes concatenation and concatenation precedes union so $\{ab, c\}$ is correct

Now, Consider, for example, the regular expression (a · (b + c))*

→ Parentheses has highest precedence

→ Inner most parentheses (b + c) → {b, c}

→Next Parentheses (a . (b+c)) →(a . {b,c})

→ star applies to the entire set → {ab, ac}*

Operator Precedence in Regular Expressions

1. **Parentheses** `()` → Highest precedence (explicit grouping).
2. **Kleene star** `*` → Applies to the preceding expression.
3. **Concatenation** `·` (implicit, no explicit symbol).
4. **Union** `+` **(Alternation/Choice)** → Lowest precedence.

Regular Expressions

Regular operations. Let A, B be languages:

- <u>Union:</u> $A \cup B = \{w \mid w \in A \text{ or } w \in B\}$
- <u>Concatenation:</u> $A \circ B = \{xy \mid x \in A \text{ and } y \in B\} = AB$
- <u>Star:</u> $A^* = \{x_1 \dots x_k \mid \text{each } x_i \in A \text{ for } k \geq 0\}$

  Note: $\varepsilon \in A^*$ always

Example. Let $A = \{\text{good, bad}\}$ and $B = \{\text{boy, girl}\}$.

- $A \cup B = \{\text{good, bad, boy, girl}\}$
- $A \circ B = AB = \{\text{goodboy, goodgirl, badboy, badgirl}\}$
- $A^* = \{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, } \dots\}$

Example 1 - Exhibit the Language in set notation….

$$
\begin{aligned}
L(a^* \cdot (a+b)) &= L(a^*)L(a+b) \\
&= (L(a))^* (L(a) \cup L(b)) \\
&= \{\lambda, a, aa, aaa, ...\}\{a, b\} \\
&= \{a, aa, aaa, ..., b, ab, aab, ...\}.
\end{aligned}
$$

Example 2

For $\Sigma = \{a,b\}$, the expression

$$r=(a+b)^*(a+bb)$$

is regular. It denotes the language

$$L(r)= \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

$L(r)$ is the set of all strings on $\{a, b\}$, terminated by either an a or a bb.

Try….

- Find all strings in L((a + bb)*) of length 5.

- Find all strings in L((ab + b)* b(a + ab)*)

Example 3

The expression

$$r = (aa)^* (bb)^* b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n}b^{2m+1} : n \geq 0, m \geq 0\}$$

Example 4

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$L(r) = \{w \in \Sigma^*: w$ has at least one pair of consecutive zeros$\}$ → $r = (0 + 1)^* 00(0 + 1)^*$.

$L(r) = \{w \in \Sigma^*: w$ has no pair of consecutive zeros$\}$ → $r = (1 + 01)^* (0 + \lambda)$ or

$r = (1 + 01)^* 0 + (1 + 01)^*$

Example 5

- Language consisting of all strings having single 0 followed by any number of 1's or single 1 followed by any number of 0's. 01* + 10*

Example 6

- Different ways of generating alternate a's and b's .

  1. (ab)* 2. b (ab)* 3. (ba)* 4. a(ba)*

Example 7

- Obtain a regular expression to accept a language with a's and b's of even length

  (aa + ab + ba + bb)*

Example 8

- Obtain a regular expression to accept a language with a's and b's of odd length

  (aa + ab + ba + bb)* (a+b)

Example 9

- Regular expression L(R) = { w| w ϵ (0,1) }* with at least 3 consecutive zeroes

(0+1) * 000 (0 +1)*

Example 10

- Obtain RE to accept a’s and b’s ending with b and has no substring aa

(b + ab)* (b + ab)

Example 11

- Obtain a regular expression to accept a language starting with **a** and ending with **b**

a (a+b)* b

Example 12

- Obtain a regular expression to accept a language with a’s and b’s of either even or multiples of 3 or both

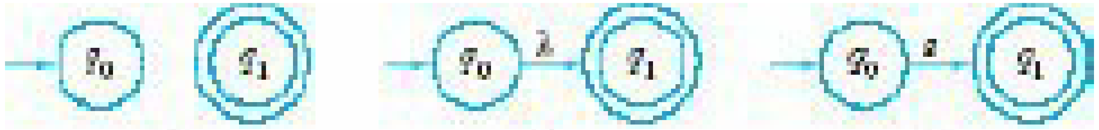
((a+b)(a+b))* + ((a+b)(a+b) (a+b))*

<u>Connection between Regular Expressions and Regular Languages</u>

- For every **regular language** there is a **regular expression**, and for every **regular expression** there is a **regular language**.

- Our definition says that a **language is regular** if it is **accepted** by some **DFA**.
- Because of the **equivalence of nfa's and dfa's**, a language is **also regular** if it is **accepted** by some **NFA**.

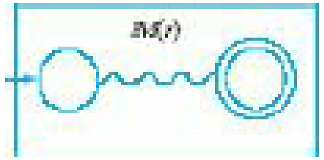
Let *r* be a regular expression. Then there exists some nondeterministic finite accepter that accepts *L*(*r*).



(a) (b) (c)

(a) nfa accepts Ø.
(b) nfa accepts $\{\lambda\}$.
(c) nfa accepts $\{a\}$.

NFA that accepts Regular Languages



Schematic representation of an NFA accepting $L(r)$.

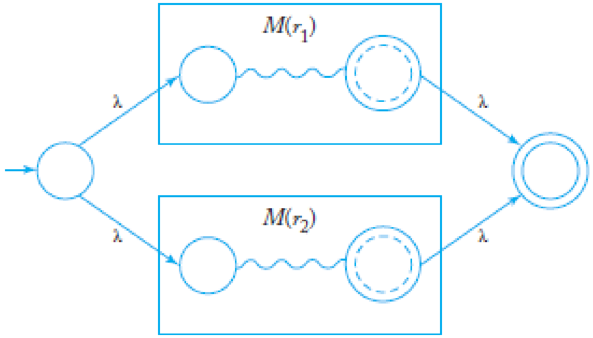
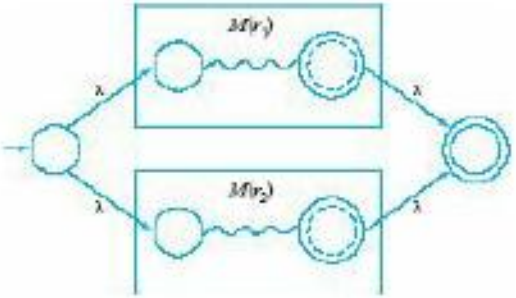


FIGURE 3.3 Automaton for $L(r_1 + r_2)$.



Automaton for $L(r_1 + r_2)$.

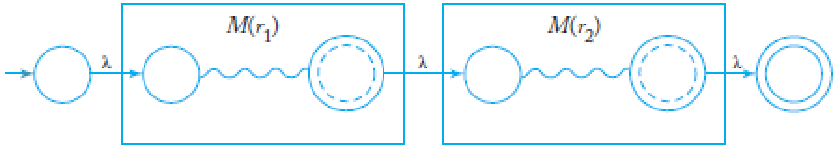
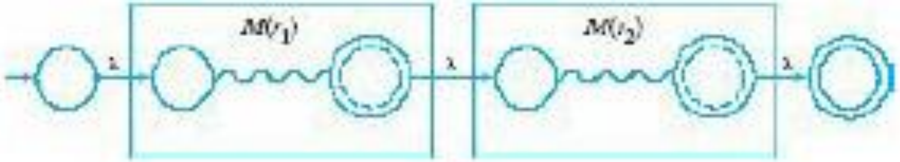


FIGURE 3.4 Automaton for $L(r_1 r_2)$.



Automaton for $L(r_1 r_2)$.

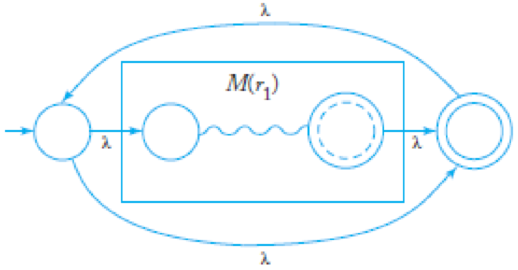
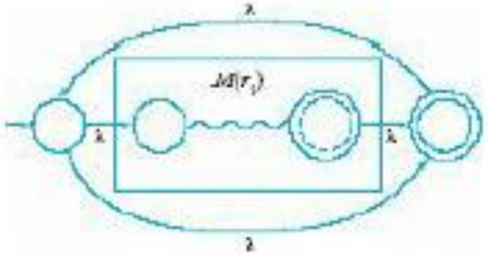


FIGURE 3.5 Automaton for $L(r_1^*)$.



Automaton for $L($ r1*).

• Find an nfa that accepts $L(r)$, where $r=(a + bb)^* (ba^* + \lambda)$

EXAMPLE 3.7

Find an nfa that accepts $L(r)$, where

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

Automata for $(a + bb)$ and $(ba^* + \lambda)$, constructed directly from first principles, are given in Figure 3.6. Putting these together using the construction in Theorem 3.1, we get the solution in Figure 3.7.

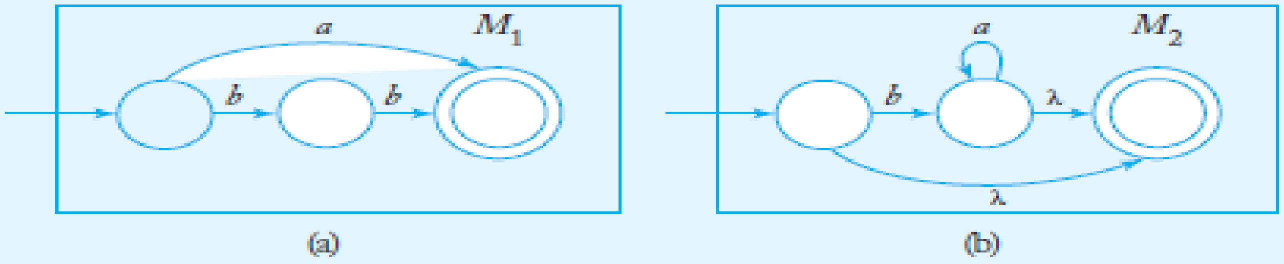


FIGURE 3.6 (a) M_1 accepts $L(a + bb)$. (b) M_2 accepts $L(ba^* + \lambda)$.

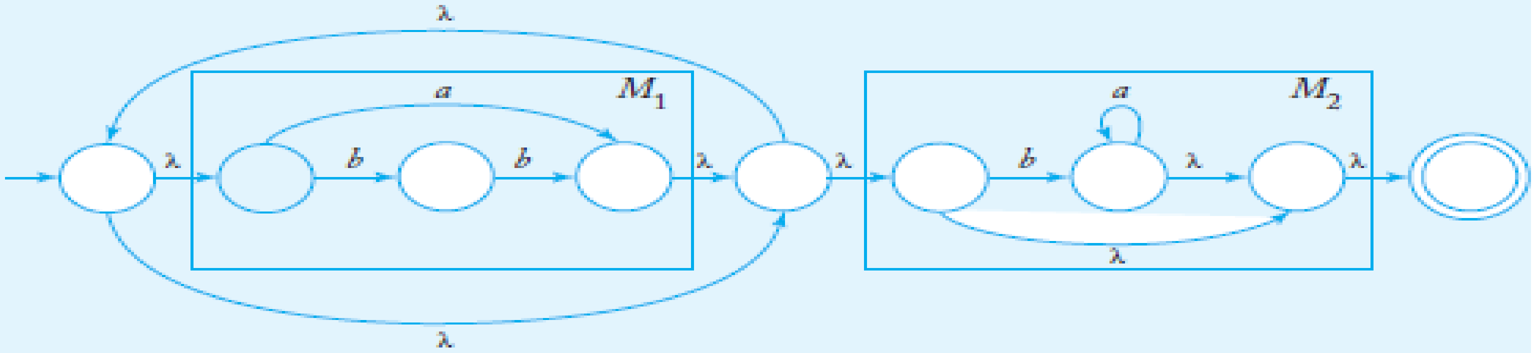
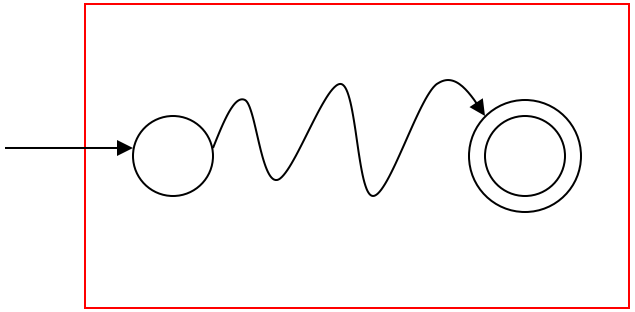


FIGURE 3.7 Automaton accepts $L((a + bb)^* (ba^* + \lambda))$.

Regular language L_1

$$L(M_1) = L_1$$

NFA M_1

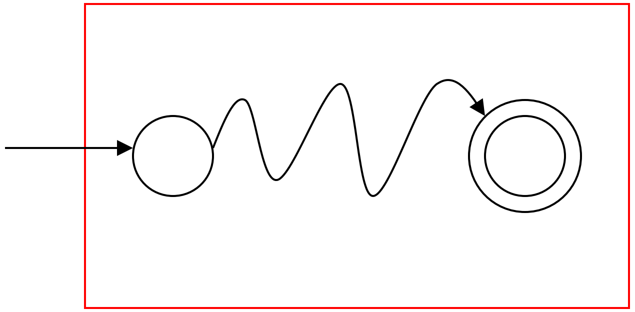


Single accepting state

Regular language L_2

$$L(M_2) = L_2$$

NFA M_2



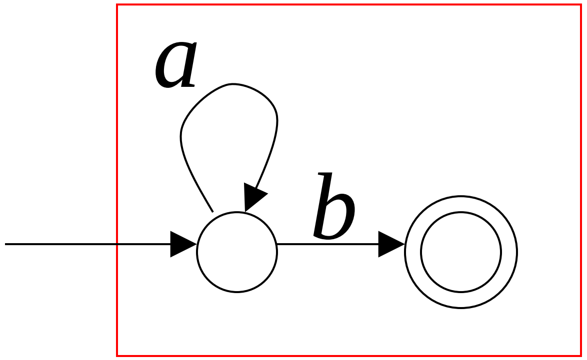
Single accepting state

Example

$n \geq 0$

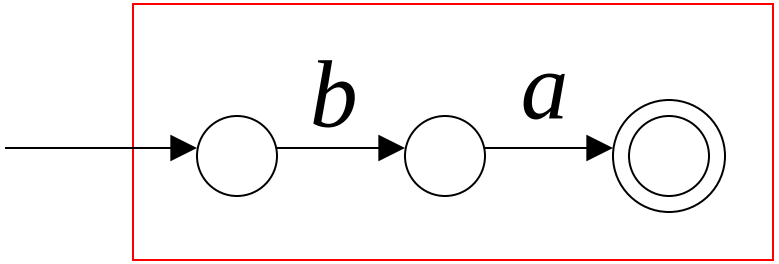
$L_1 = \{a^n b\}$

M_1



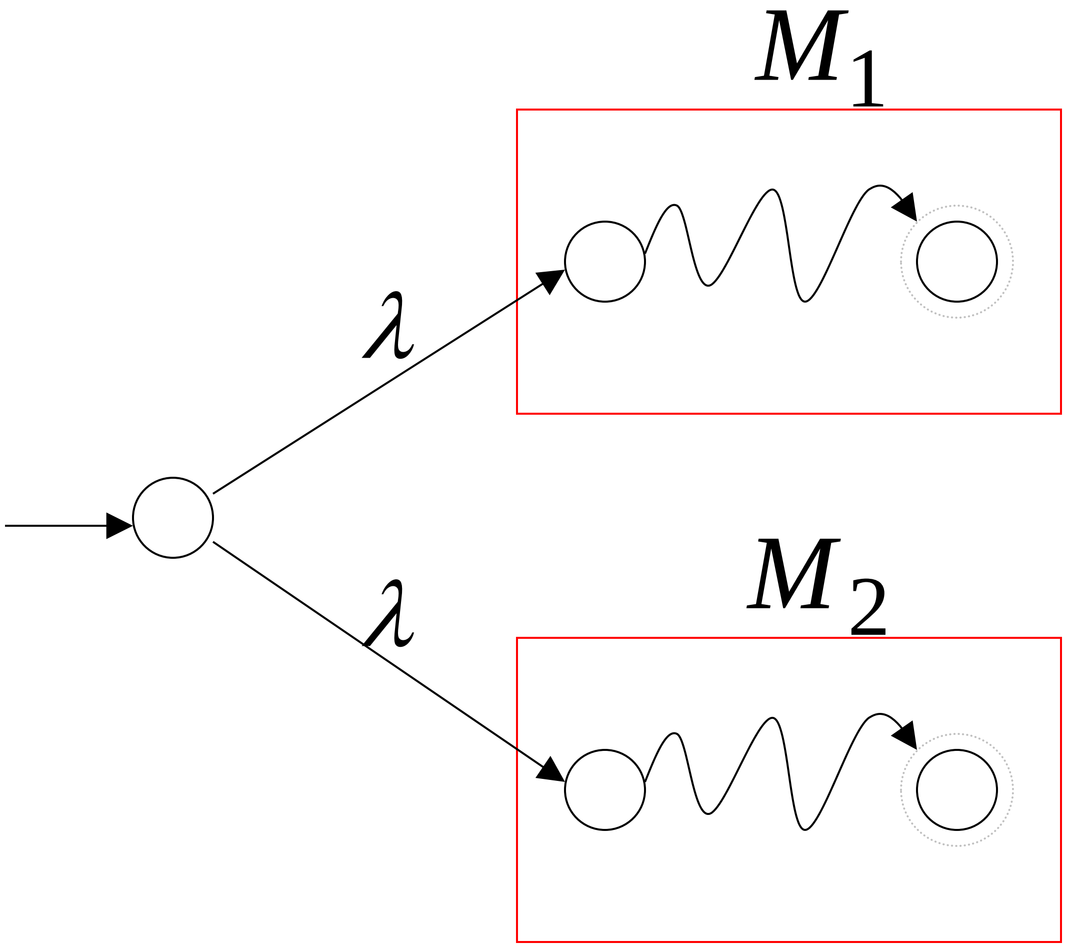
$L_2 = \{ba\}$

M_2

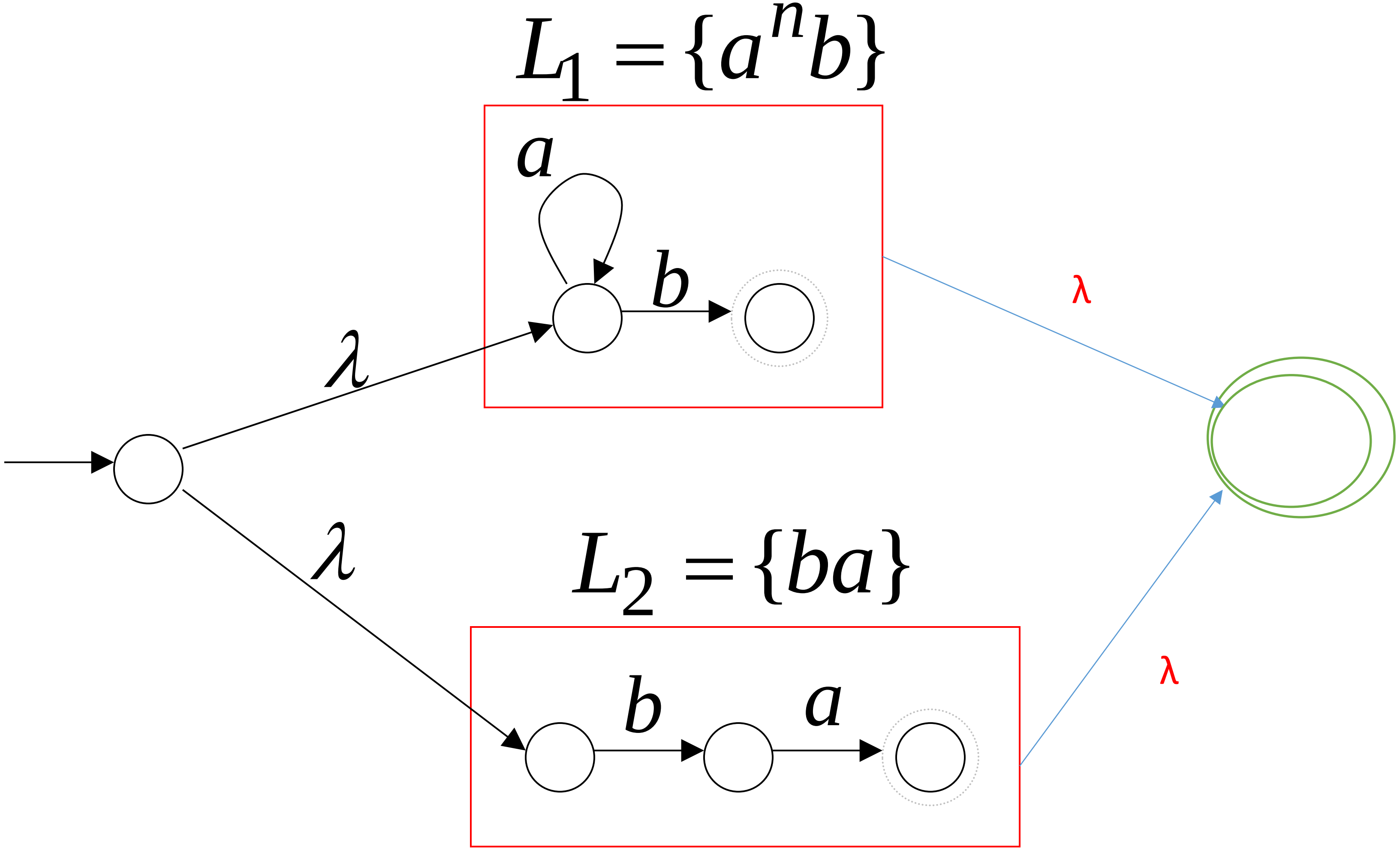


Union $L_1 \cup L_2$

NFA for



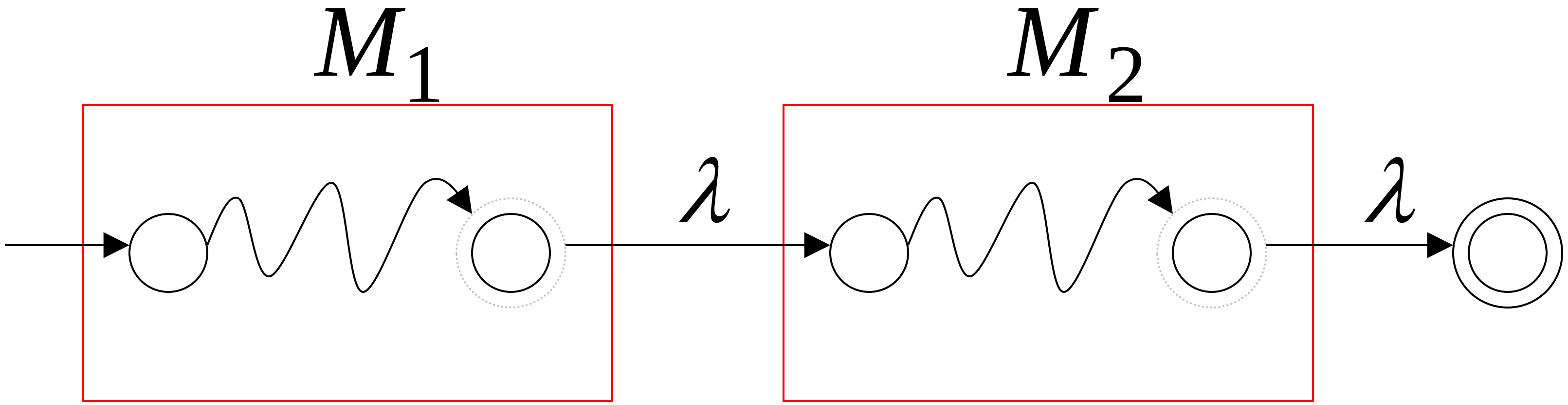
Example NFA for $L_1 \cup L_2 = \{a^n b\} \cup \{ba\}$



Concatenation

$$L_1 L_2$$

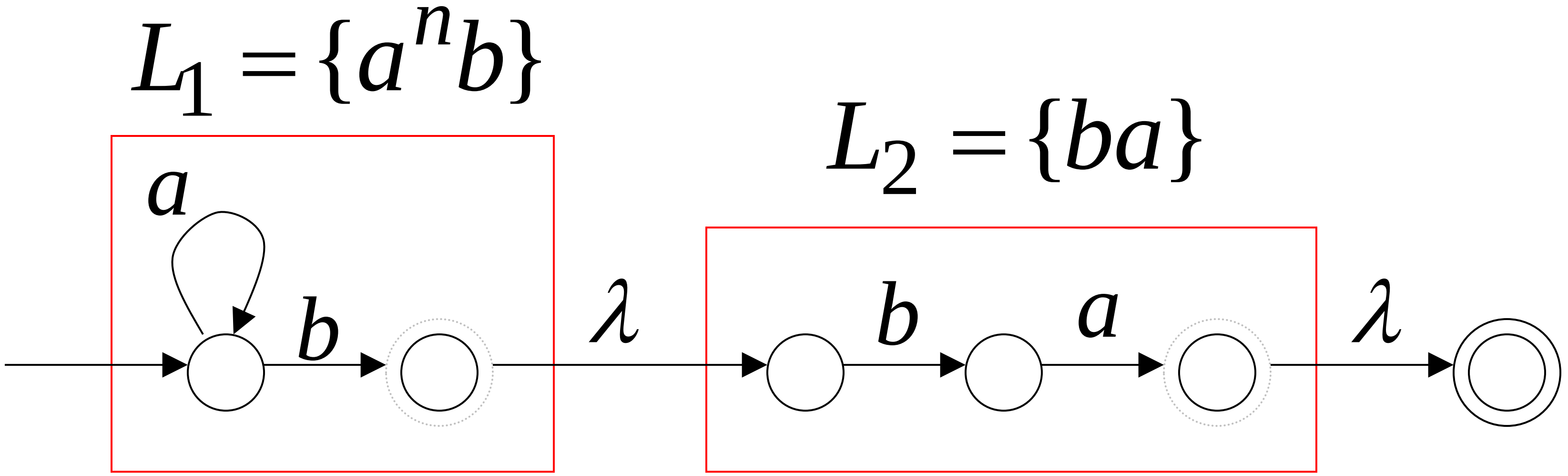
NFA for



Example

$$L_1 L_2 = \{a^n b\}\{ba\} = \{a^n bba\}$$

NFA for



How do we construct automata for the remaining operations?

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1{}^*$

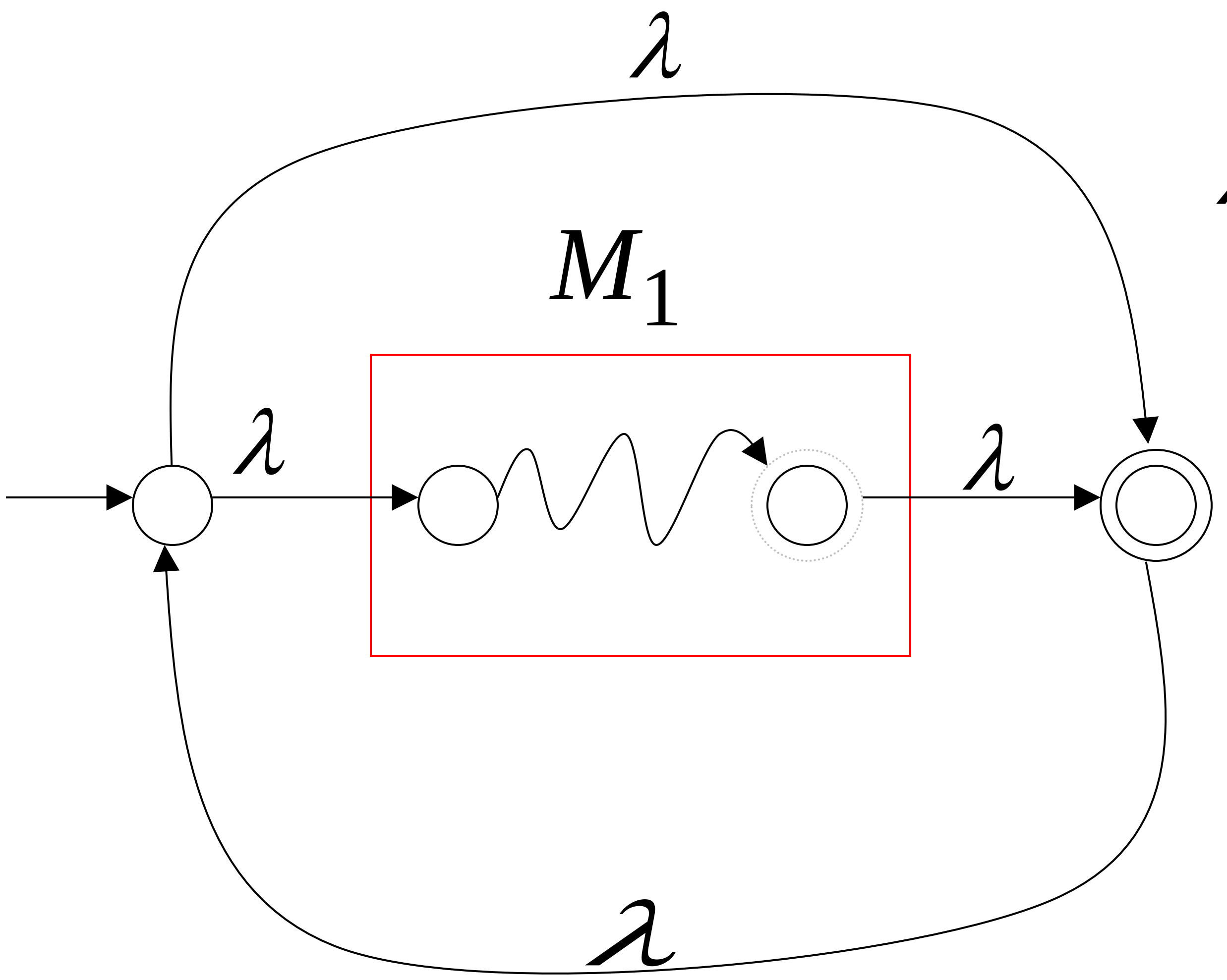
Reversal: $L_1{}^R$

Complement: $\overline{L_1}$

Intersection: $L_1 \cap L_2$

Star Operation L_1*

NFA for



$\lambda \in L_1*$

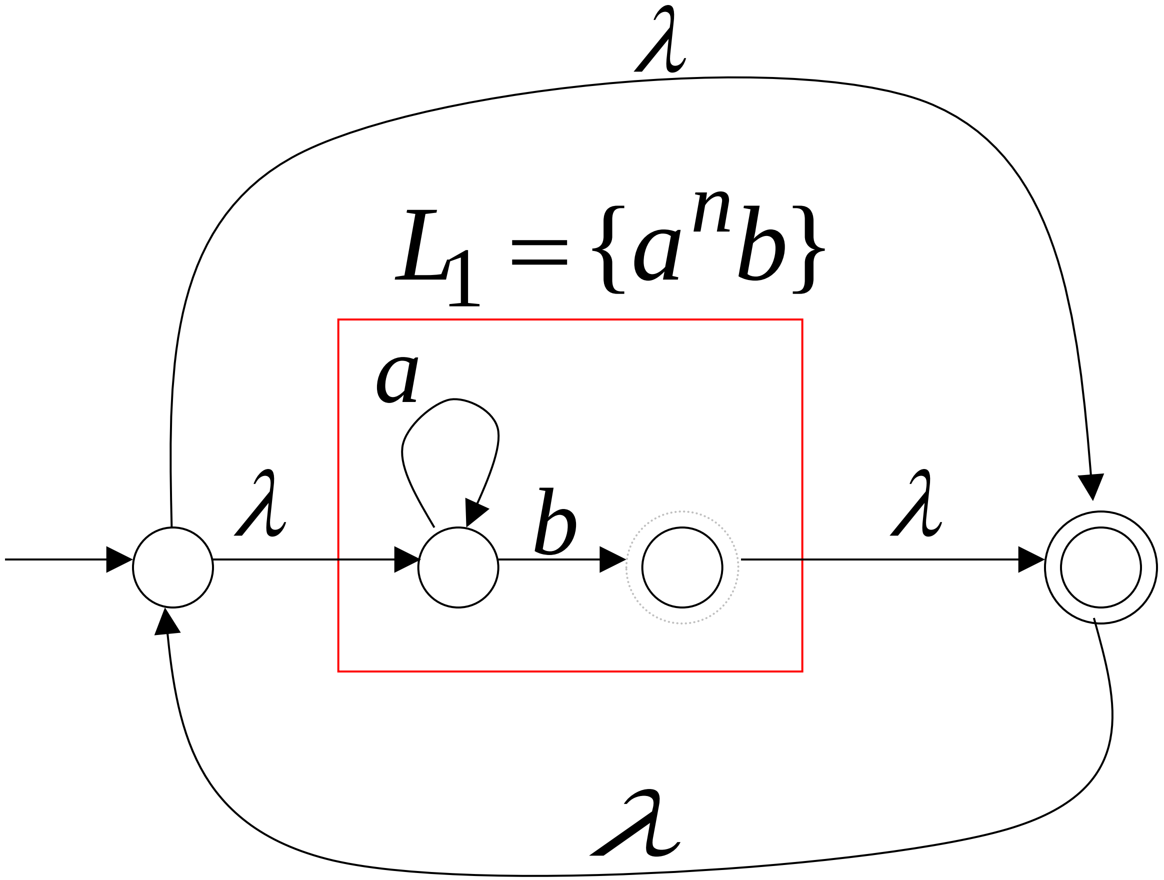
Example

$$L_1{}^* = \{a^n b\}^*$$

$$w = w_1 w_2 \cdots w_k$$

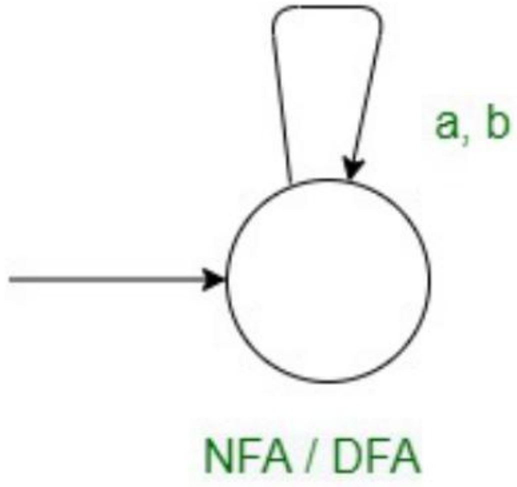
$$w_i \in L_1$$

NFA for



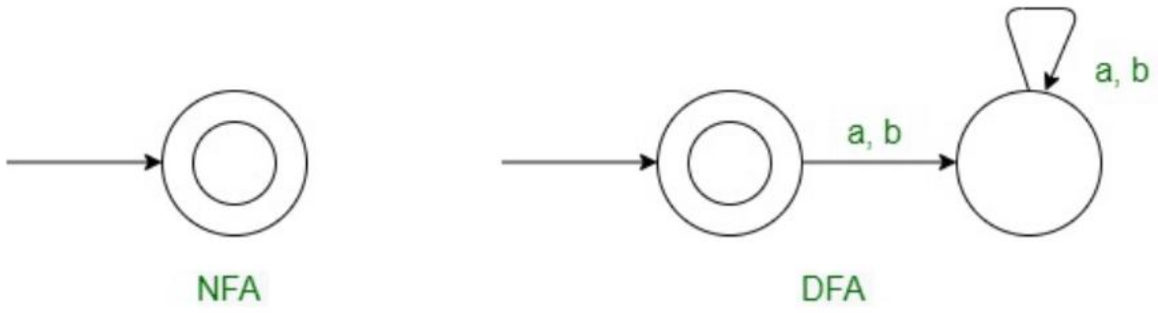
Case-1 :

When r = Φ, then FA will be as follows.



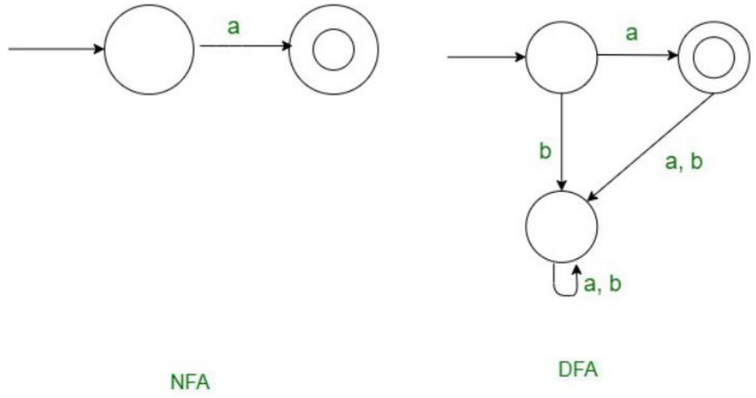
Case-2 :

When **r = ε**, then FA will be as follows.



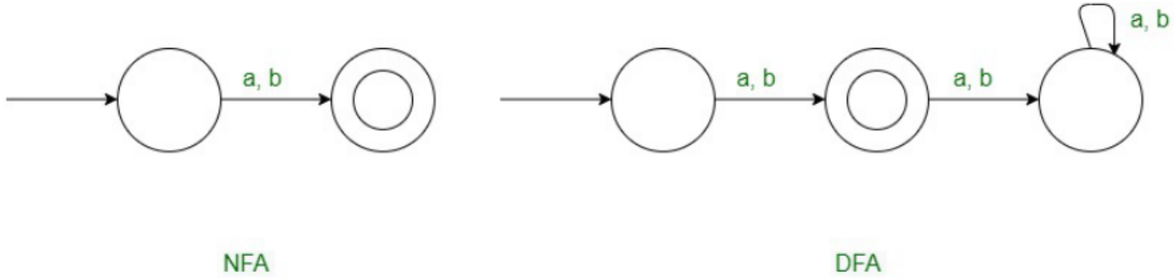
Case-3 :

When **r = a**, then FA will be as follows.



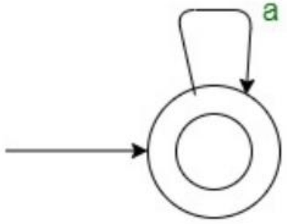
Case-4 :

When **r = a+b** , then FA will be as follows.

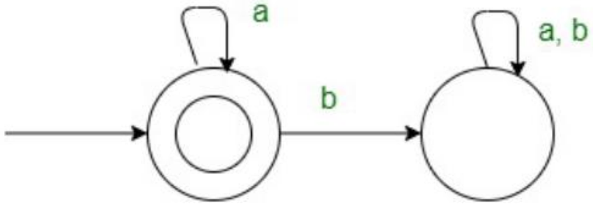


Case-5 :

When **r = r = a*** , then FA will be as follows.



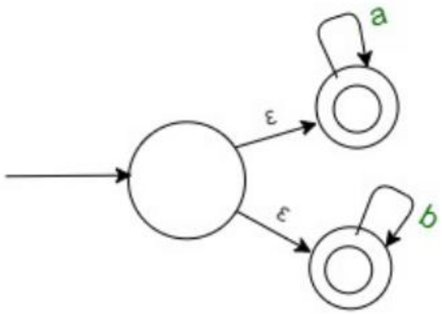
NFA



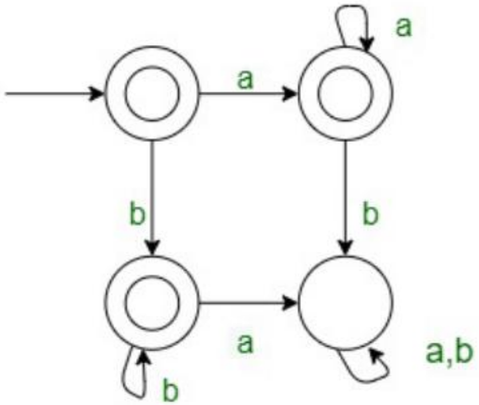
DFA

Case-6 :

When **r = a* + b***, then FA will be as follows.



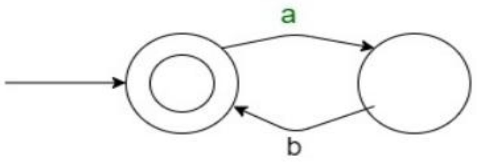
NFA



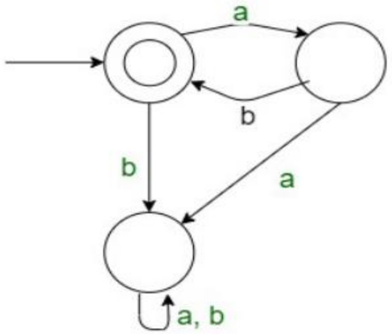
DFA

Case-7 :

When **r = (ab)***, then FA will be as follows.



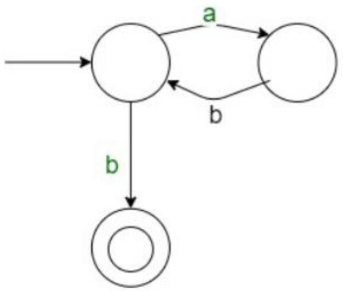
NFA



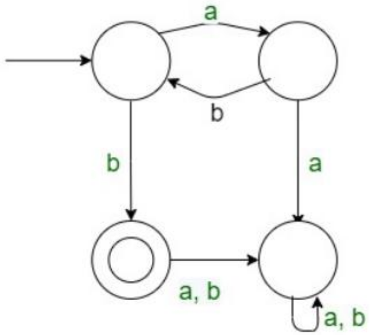
DFA

Case-8 :

When **r = (ab)*b**, then FA will be as follows.



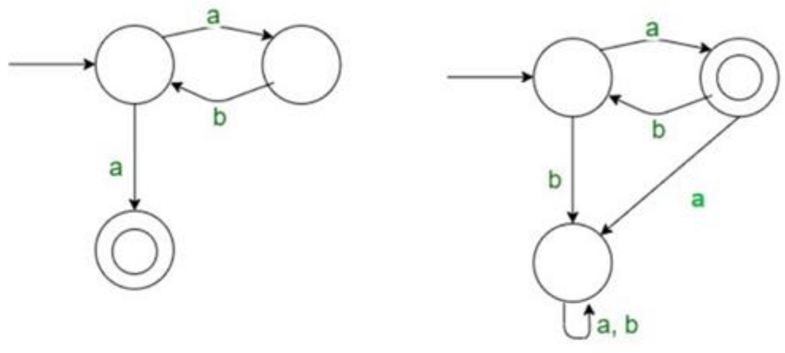
NFA



DFA

Case-9 :

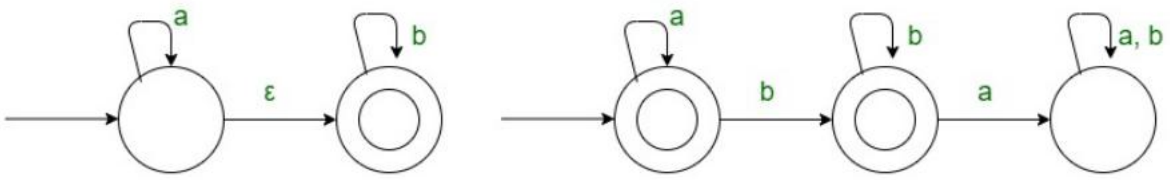
When **r = (ab)*a**, then FA will be as follows.



NFA DFA

Case-10 :

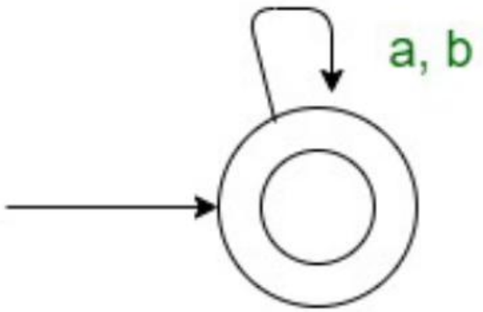
When **r = a*b***, then FA will be as follows.



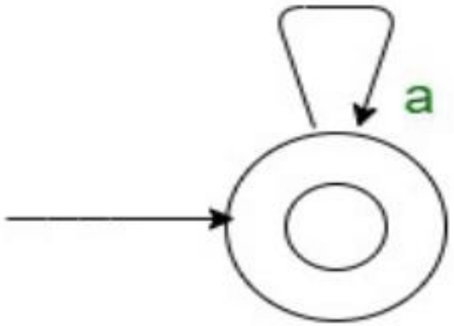
NFA DFA

Case-11 :

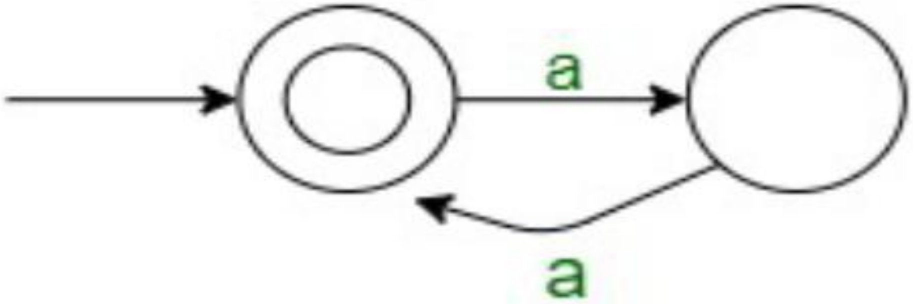
When **r = (a+b)***, then FA will be as follows.



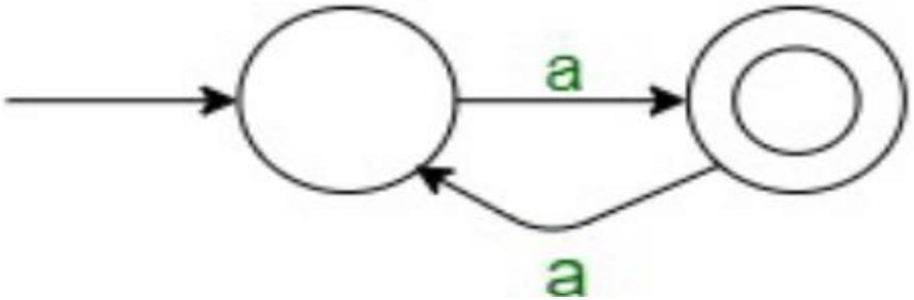
Case-1 : r = a*



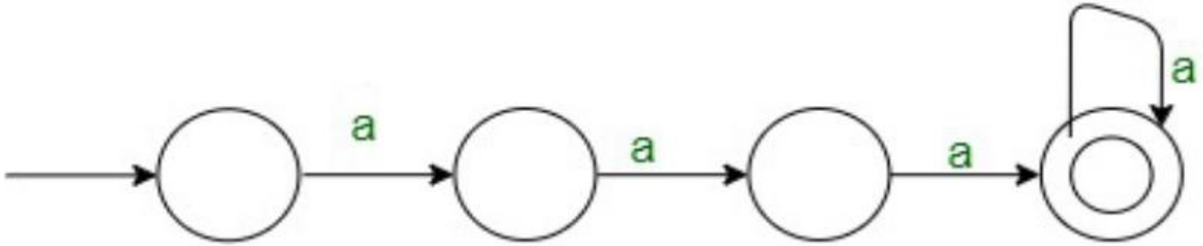
Case-2 : r = (aa)*



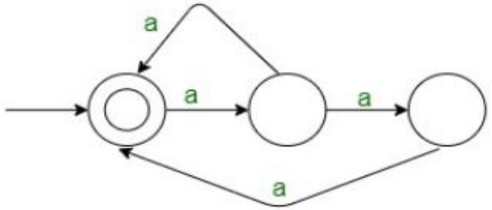
Case-3 : r = (aa)*a



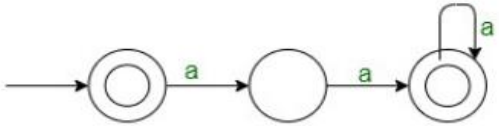
Case-4 : r = aaaa*



Case-5 : r= (aa + aaa)*

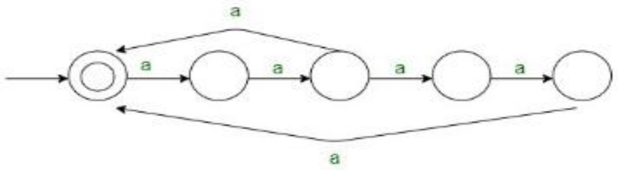


NFA

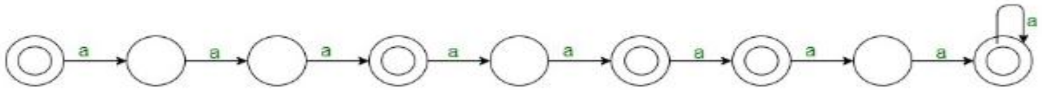


DFA

Case-6 : r=(aaa+aaaaa)*



NFA



Assignments

- Obtain NFA which accepts strings **a’s** and **b’s** starting with **a** and **b**
- Obtain NFA for regular expression **(a*+b*+c*)**
- Obtain NFA for regular expression **(a+b)* aa (a+b)***

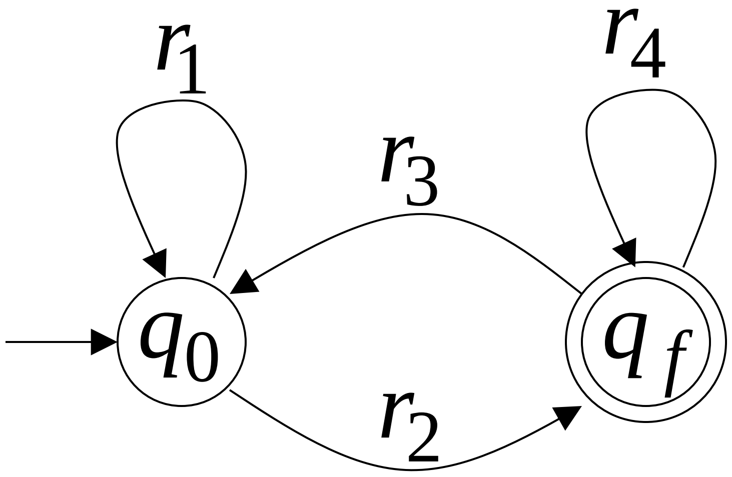
Generalized Transition Graph (GTG)

- A generalized transition graph 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,

Generalized Transition Graph (GTG)

The final transition graph:



The resulting regular expression:

$$r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^*$$

$$L(r) = L(M) = L$$

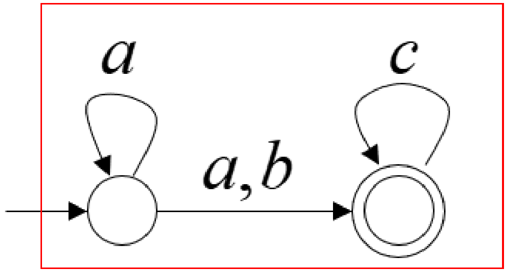
From M construct the equivalent

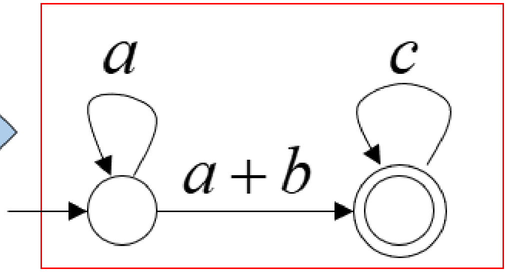
Generalized Transition Graph

in which transition labels are regular expressions

Example:

M





$$r = r_1 * r_2 (r_4 + r_3 r_1 * r_2) *$$

So, r = a* (a+b) (c)*

Procedure nfa-rex

1. Start with an nfa with states q0,q1, ...qn and a single final state , distinct from its initial state

Procedure nfa-rex

1. Start with an nfa with states q0,q1, ...qn and a single final state , distinct from its initial state
2. Convert the nfa into a complete generalized transition graph.

$$r_{ii}^* \, r_{ij} \, (r_{jj} + r_{ji} \, r_{ii}^* \, r_{ij})^*$$

3. If the generalized transition graph(GTG) has only 2 states with qi as initial and qj as final, as its associated regular expression is

Let r_{ij} stand for the label of the edge from qi to qj

4. If GTG has 3 states with the initial state qi and final state qj and the third state qk, introduce new edge labelled

$r_{pq} + r_{pk} r_{kk}^{*} r_{kq}$ for p =i, j and q=i, j. When this is done the remove the vertex q_k and its associated edges.

5 . If GTG has 4 or more edges , pick a state q_k to be removed. Apply rule 4 for all pairs of states (q_i , q_j). $i \neq k$, $j \neq k$. At each step apply the simplifying rules

$r+\Phi = r$

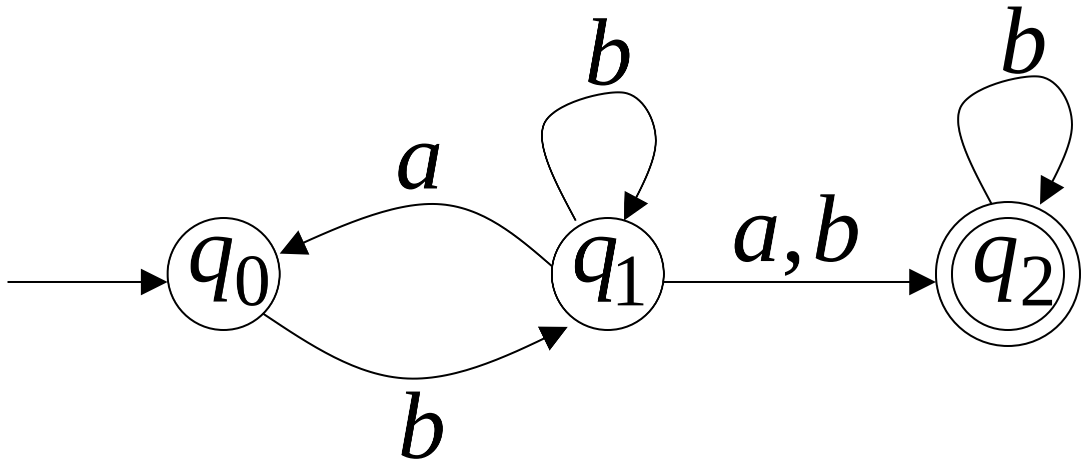
$r\Phi = \Phi$

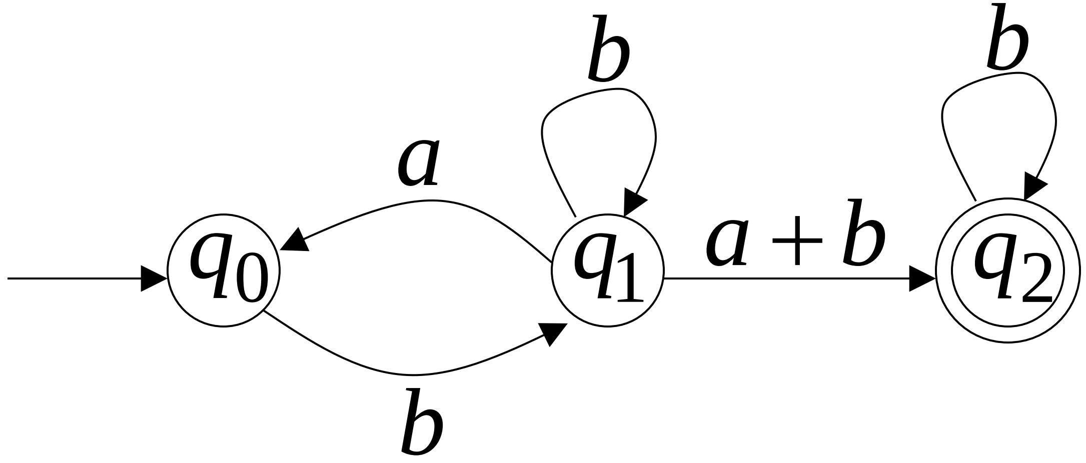
$\Phi^* = \lambda$

Wherever possible. When this is done , remove q_k

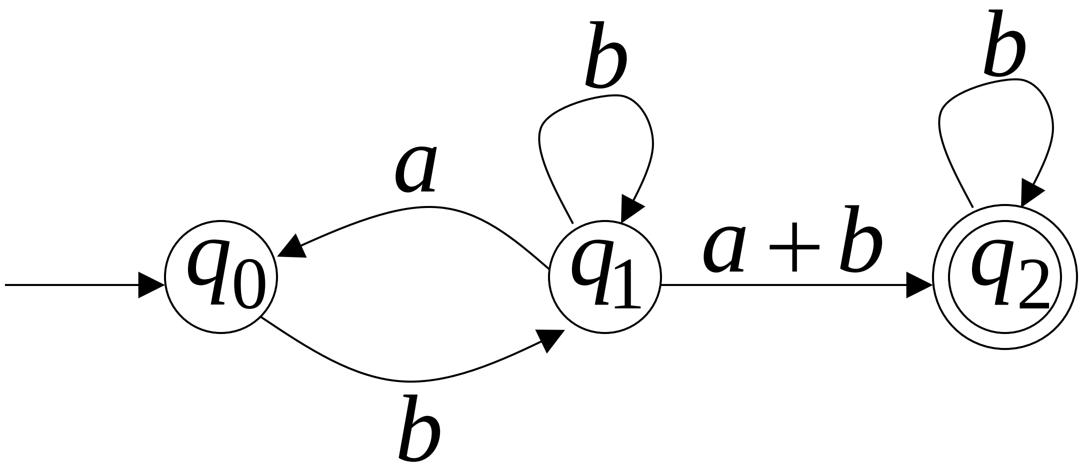
6. Repeat step 3 to 5 until the correct regular expression is obtained.

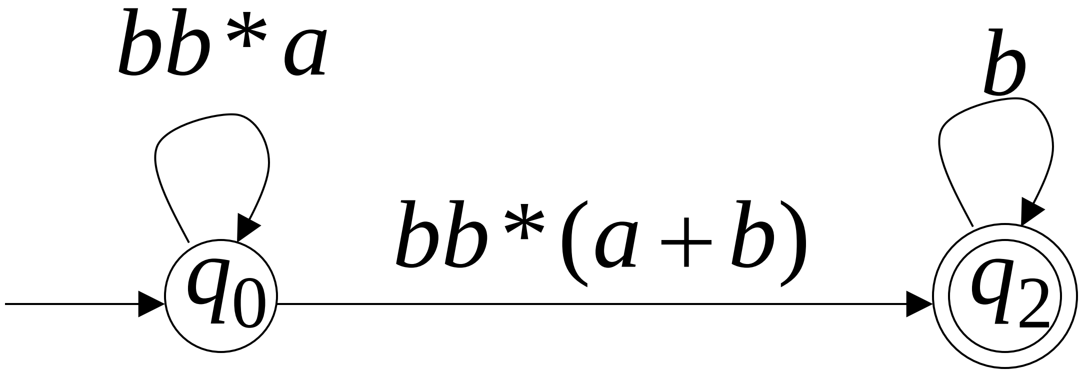
Another Example:



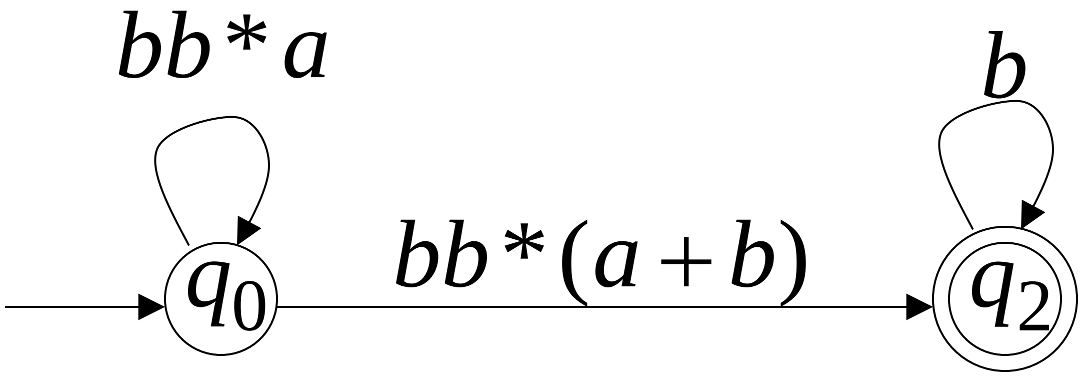


Reducing the states:





Resulting Regular Expression:

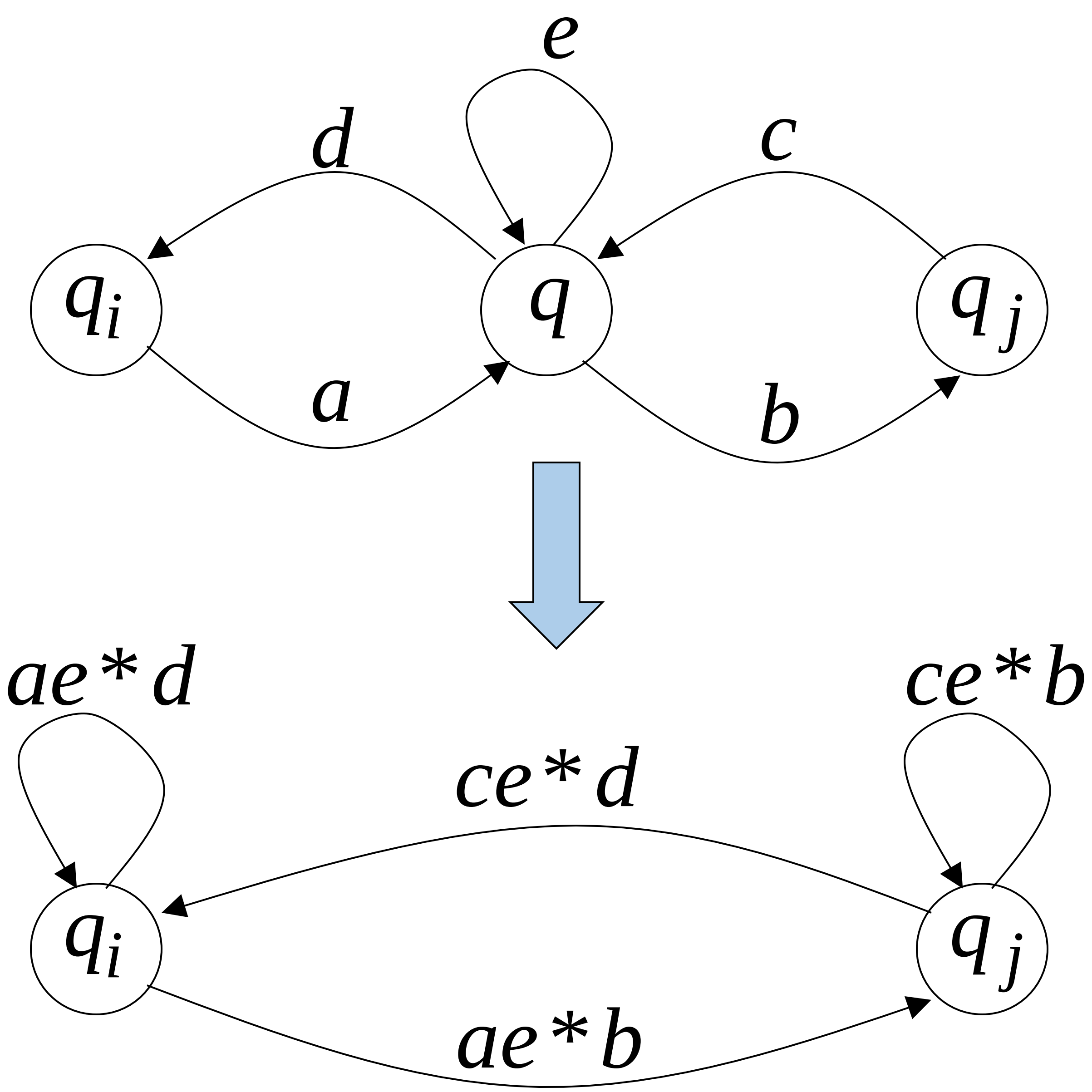


$$r = (bb^*a)^*bb^*(a+b)b^*$$

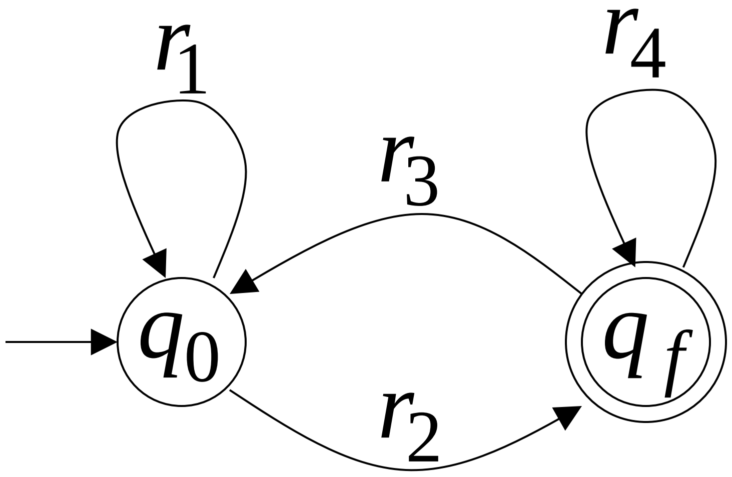
$$L(r) = L(M) = L$$

In General

Removing states:



The final transition graph:

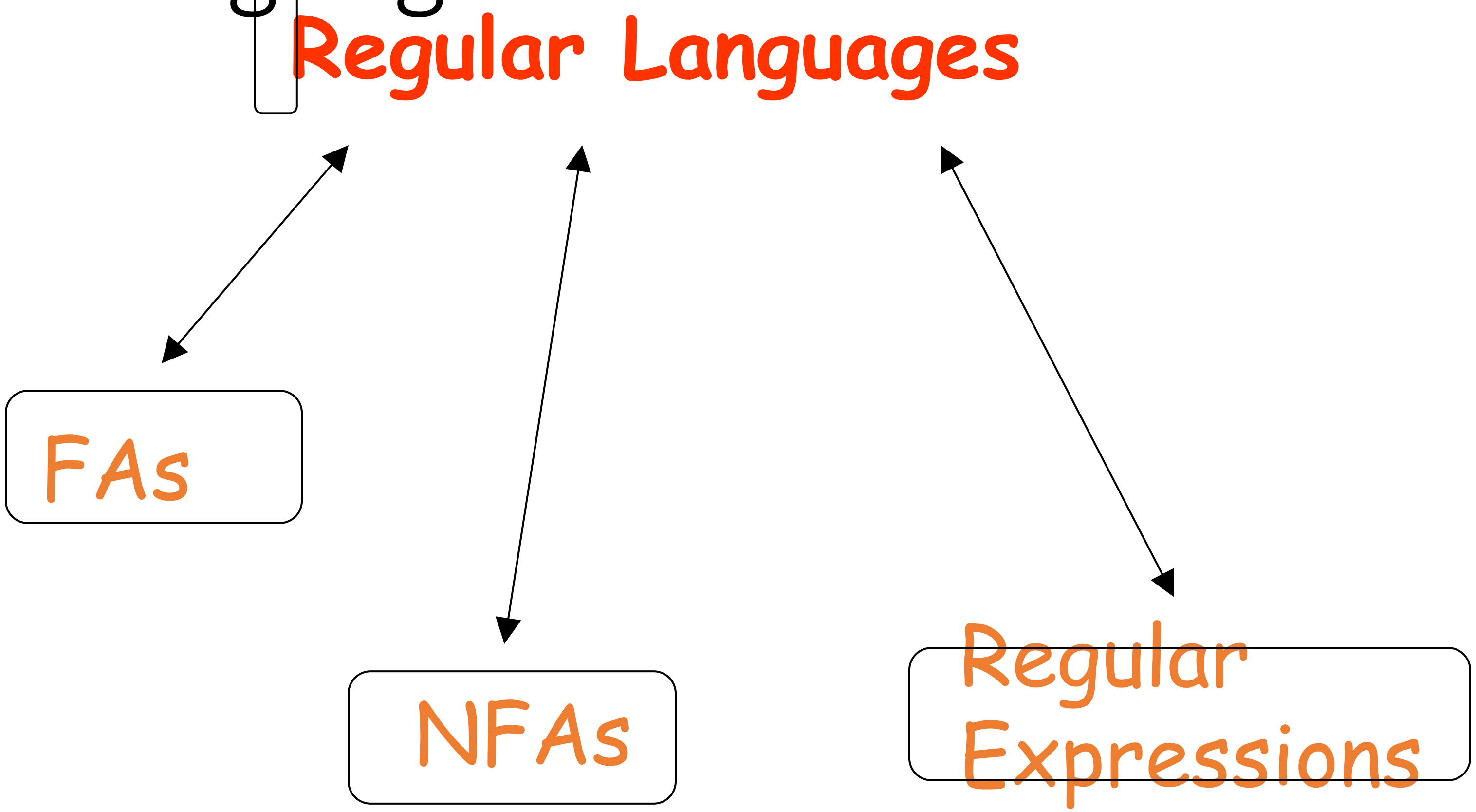


The resulting regular expression:

$$r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^*$$

$$L(r) = L(M) = L$$

Standard Representations of Regular Languages



When we say: We are given a Regular Language L

We mean: Language L is in a standard representation

Elementary Questions

about

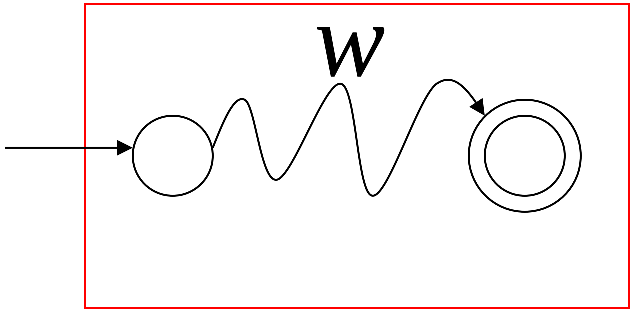
Regular Languages

Question: Given regular language L
and string w
how can we check if $w \in L$?

Question: Given regular language L
and string w
how can we check if $w \in L$?

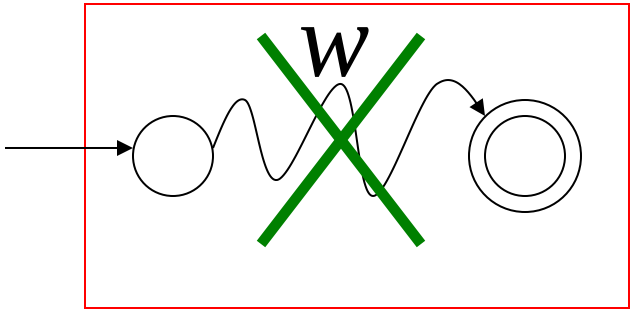
Answer: Take the DFA that accepts L
and check if w is accepted

DFA



$$w \in L$$

DFA



$$w \notin L$$

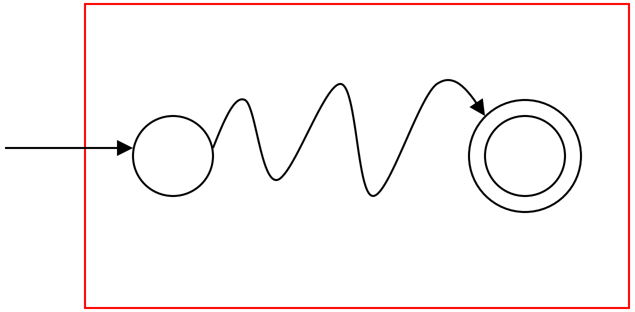
Question: Given regular language L how can we check if L is empty: $(L = \varnothing)$?

Question: Given regular language L
how can we check
if L is empty: $(L = \emptyset)$?

Answer: Take the DFA that accepts L

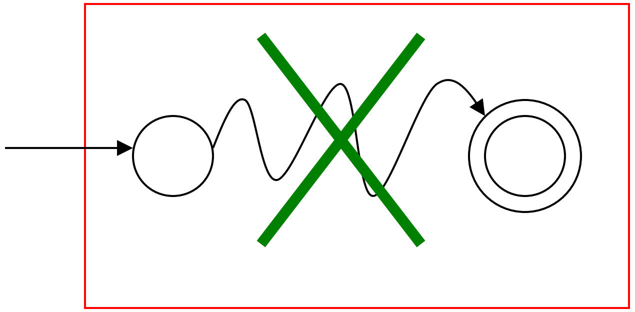
Check if there is any path from
the initial state to a final state

DFA



$$L \neq \emptyset$$

DFA



$$L = \emptyset$$

Question: Given regular language L how can we check if L is infinite?

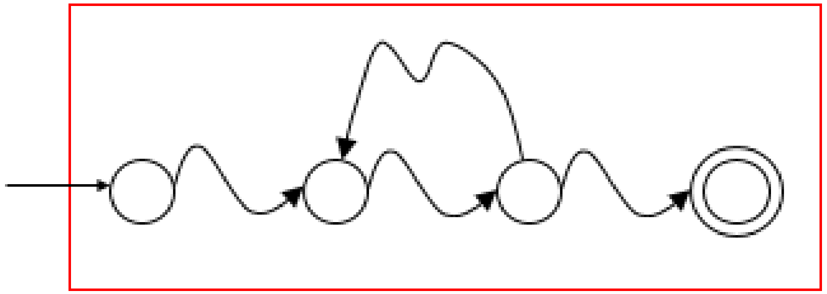
if the DFA has a loop that includes an accepting state, then the language is infinite; otherwise, if all accepting paths have a bounded length, the language is finite.

Question: Given regular language L
how can we check
if L is infinite?

Answer: Take the DFA that accepts L

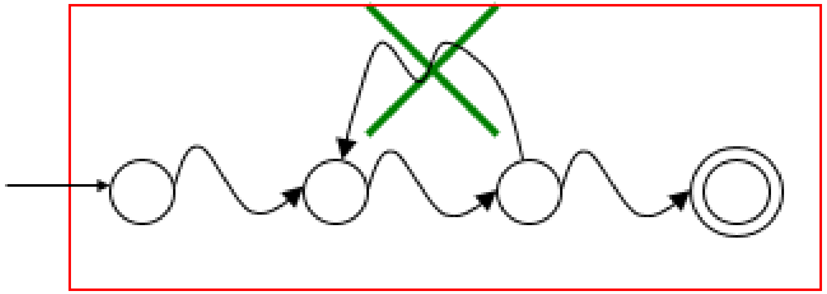
Check if there is a walk with cycle
from the initial state to a final state

DFA



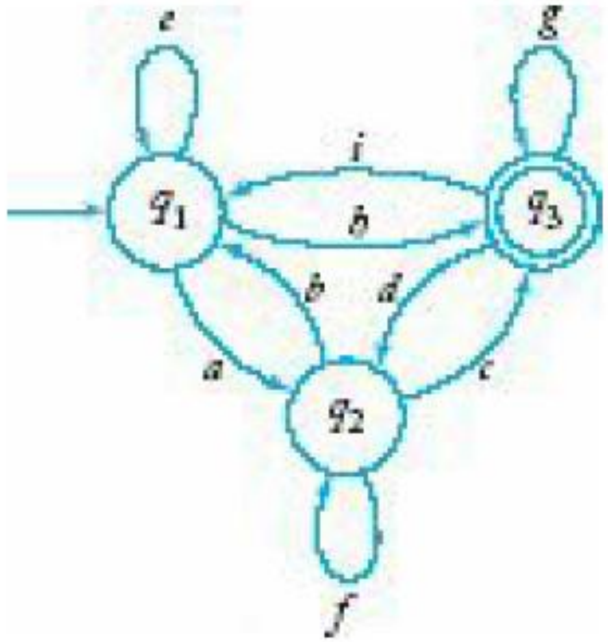
L is infinite

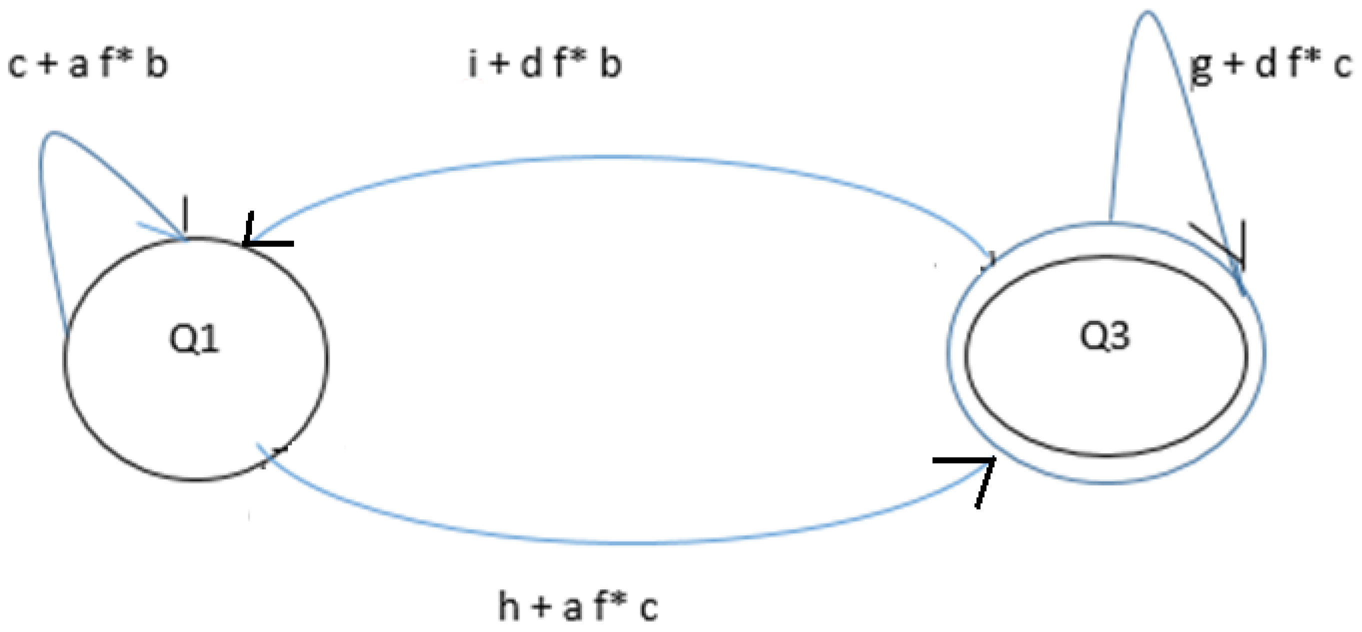
DFA



L is finite

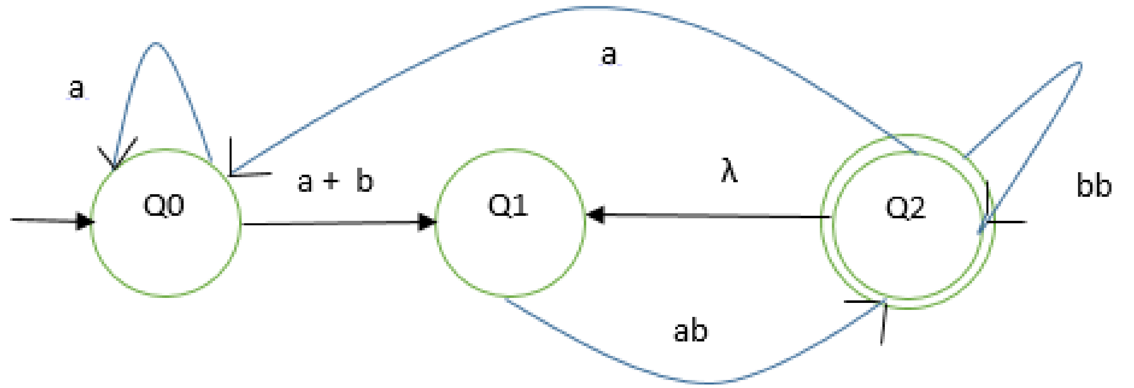
Find GTG for the following





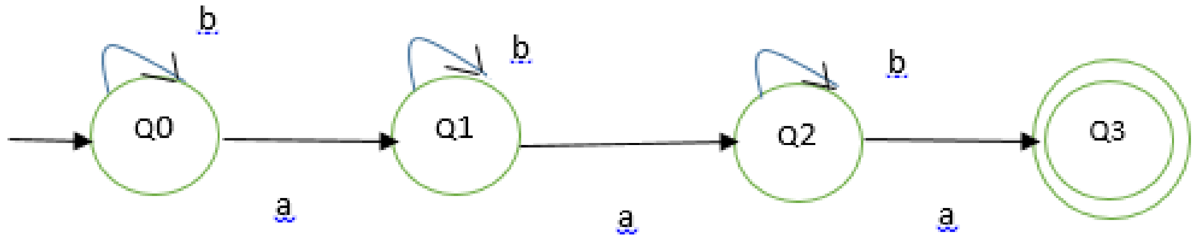
Find GTG for the following $r = r_1{}^* r_2 (r_4 + r_3 r_1{}^* r_2)^*$

1



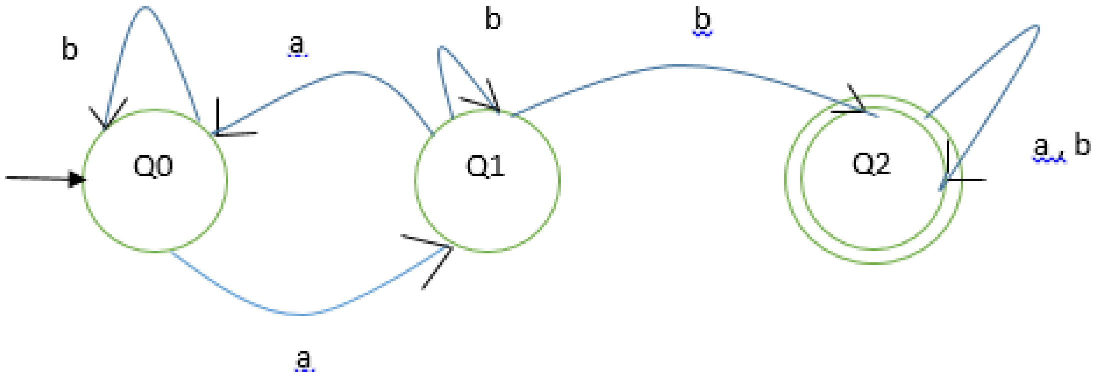
a*(a+b)ab((bb + ab) + aa* (a+b) ab)*

2



b * a b* ab* a (Φ)*

3



(b + ab* a) * ab*b (a + b)*

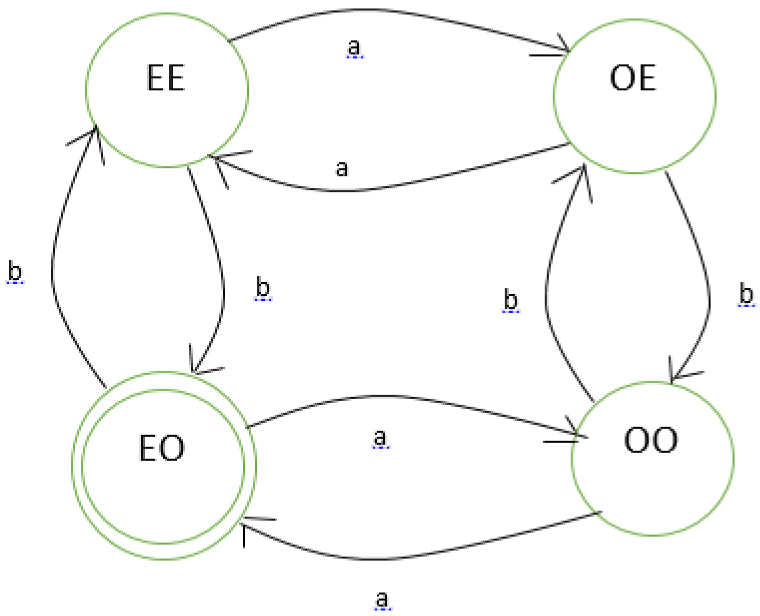
4 L = { w | even number of **a's** and odd number of **b's**}

$R1 = aa + ab(bb)^* ba$

$R2 = b + ab(bb)^*a$

$R3 = b + a(bb)^*ba$

$R4 = a (bb)^* a$



Substitute in $R = r1^* r2(r4 + r3r1^*r2)^*$

Regular Grammars

Right- and Left-Linear Grammars

A grammar $G = (V, T, S, P)$ is said to be **right-linear** if all productions are of the form

$$A \to xB,$$

$$A \to x,$$

where $A, B \in V$, and $x \in T^*$. A grammar is said to be **left-linear** if all productions are of the form

$$A \to Bx,$$

or

$$A \to x.$$

A regular grammar is one that is either right-linear or left-linear.

Describing Regular Languages via

1. Reg Expressions
2. If it is accepted by DFAs or NFAs
3. Certain grammars

In a **regular grammar**, **at most one variable appears on the right side of any production**. Furthermore, that variable must consistently be either the rightmost or leftmost symbol of the right side of any production.

The grammar $G_1 = (\{S\}, \{a,b\}, S, P_1)$, with P_1 given as

$$S \rightarrow abS|a$$

is right-linear. The grammar $G_2 = (\{S, S_1, S_2\}, \{a, b\}, S, P_2)$, with productions

$$S \rightarrow S_1ab,$$

$$S_1 \rightarrow S_1ab|S_2,$$

$$S_2 \rightarrow a,$$

is left-linear. Both *G1* and *G2* are regular grammars.

The sequence

$$S \Rightarrow abS \Rightarrow ababS \Rightarrow ababa$$

$L(G_1)$ is the language denoted by the regular expression $r = (ab)^* a$.

In a similar way, we can see that $L(G_2)$ is the regular language $L(aab(ab)^*)$.

Linear Grammars

The grammar $G = (\{S, A, B\}, \{a, b\}, S, P)$ with productions

$$S \to A$$

$$A \to aB|\lambda,$$

$$B \to Ab,$$

is not regular. Although every production is either in right-linear or left-linear form, the grammar itself is neither right-linear nor left-linear, and therefore is not regular. The grammar is an example of a **linear grammar**.

A linear grammar is a grammar in which **at most one variable can occur on the right side** of any production, **without restriction on the position of this variable**.

Regular grammar is always linear but not all linear grammars are regular.

Right-Linear Grammars Generate Regular Languages

Let $G = (V, T, S, P)$ be a right-linear grammar. Then $L(G)$ is a regular language.

Proof: We assume that $V = \{V_0, V_1, \ldots\}$, that $S = V_0$, and that we have productions of the form $V_0 \to v_1 V_i, V_i \to v_2 V_j, \ldots$ or $V_n \to v_l, \ldots$. If w is a string in $L(G)$, then because of the form of the productions

$V_0 \to v_1 V_i$
$V_i \to v_2 V_j$
…
$V_n \to v_l$

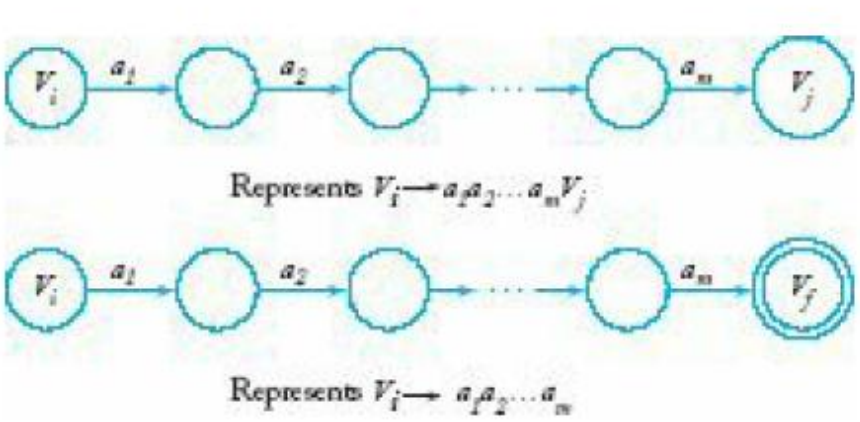
$$\begin{aligned} V_0 &\Rightarrow v_1 V_i \\ &\Rightarrow v_1 v_2 V_j \\ &\overset{*}{\Rightarrow} v_1 v_2 \cdots v_k V_n \\ &\Rightarrow v_1 v_2 \cdots v_k v_l = w. \end{aligned}$$

The automaton to be constructed will reproduce the derivation by consuming each of these v's in turn. The initial state of the automaton will be labeled V_0, and for each variable V_i there will be a nonfinal state labeled V_i. For each production

$$V_i \to a_1 a_2 \cdots a_m V_j,$$

the automaton will have transitions to connect V_i and V_j that is, δ will be defined so that

$$\delta^* (V_i, a_1 a_2 \cdots a_m) = V_j.$$



For each production

$$V_i \rightarrow a_1a_2 \cdots a_m,$$

the corresponding transition of the automaton will be

$$\delta^* (V_i, a_1a_2 \dots a_m) = V_f,$$

where V_f is a final state.

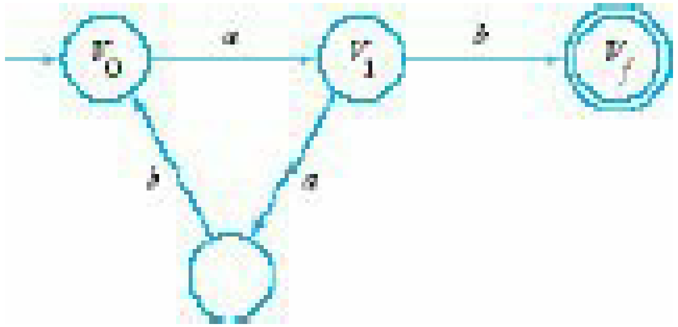
Construct a finite automaton that accepts the language generated by the grammar

$$V_0 \rightarrow aV_1,$$

$$V_1 \rightarrow abV_0 | b,$$

where V_0 is the start variable. We start the transition graph with vertices V_0, V_1, and V_f. The first production rule creates an edge labeled a between V_0 and V_1. For the second rule, we need to introduce an additional vertex so that there is a path labeled ab between $V1$ and V_0.

The language generated by the grammar and accepted by the automaton is the regular language
***L* ((*aab*) * *ab*.**



Right-Linear Grammars for Regular Languages

If L is a regular language on the alphabet Σ, then there exists a right-linear grammar $G = (V, \Sigma, S, P)$ such that $L = L(G)$.

Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a dfa that accepts L. We assume that $Q = \{q_0, q_1, \ldots, qn\}$ and $\Sigma = \{a_1, a_2, \ldots, a_m\}$. Construct the right-linear grammar $G = (V, \Sigma, S, P)$ with

$$V = \{q_0, q_1, \ldots, q_n\}$$

and $S = q_0$. For each transition

Take example aab*a

$$\delta(q_i, a_j) = q_k$$

of M, we put in P the production

$$q_i \to a_j q_k. \tag{3.5}$$

In addition, if q_k is in F, we add to P the production

$$q_k \to \lambda. \tag{3.6}$$

We first show that G defined in this way can generate every string in L. Consider $w \in L$ of the form

$$w = a_i a_j \ldots a_k a_l.$$

For M to accept this string it must make moves via

$$\begin{aligned} \delta(q_0, a_i) &= q_p, \\ \delta(q_p, a_j) &= q_r, \\ &\vdots \\ \delta(q_s, a_k) &= q_t, \\ \delta(q_t, a_l) &= q_f \in F. \end{aligned}$$

δ(q0, ai) = qp
δ(qp, aj) = qr
:
:
δ(qs, ak) = qt
δ(qt, al) = qf
qf ∊ F

By construction, the grammar will have one production for each of these δ's. Therefore, we can make the derivation

$$\begin{aligned} q_0 &\Rightarrow a_i q_p \Rightarrow a_i a_j q_r \stackrel{*}{\Rightarrow} a_i a_j \cdots a_k q_t \\ &\Rightarrow a_i a_j \cdots a_k a_l q_f \Rightarrow a_i a_j \cdots a_k a_l, \end{aligned} \tag{3.7}$$

with the grammar G, and $w \in L(G)$.

Conversely, if $w \in L(G)$, then its derivation must have the form (3.7). But this implies that

$$\delta^* (q_0, a_i a_j \ldots a_k a_l) = q_f,$$

completing the proof. ■

Construct a right-linear grammar for *L (aab*a)*.
The string *aaba* can be derived with the constructed grammar by

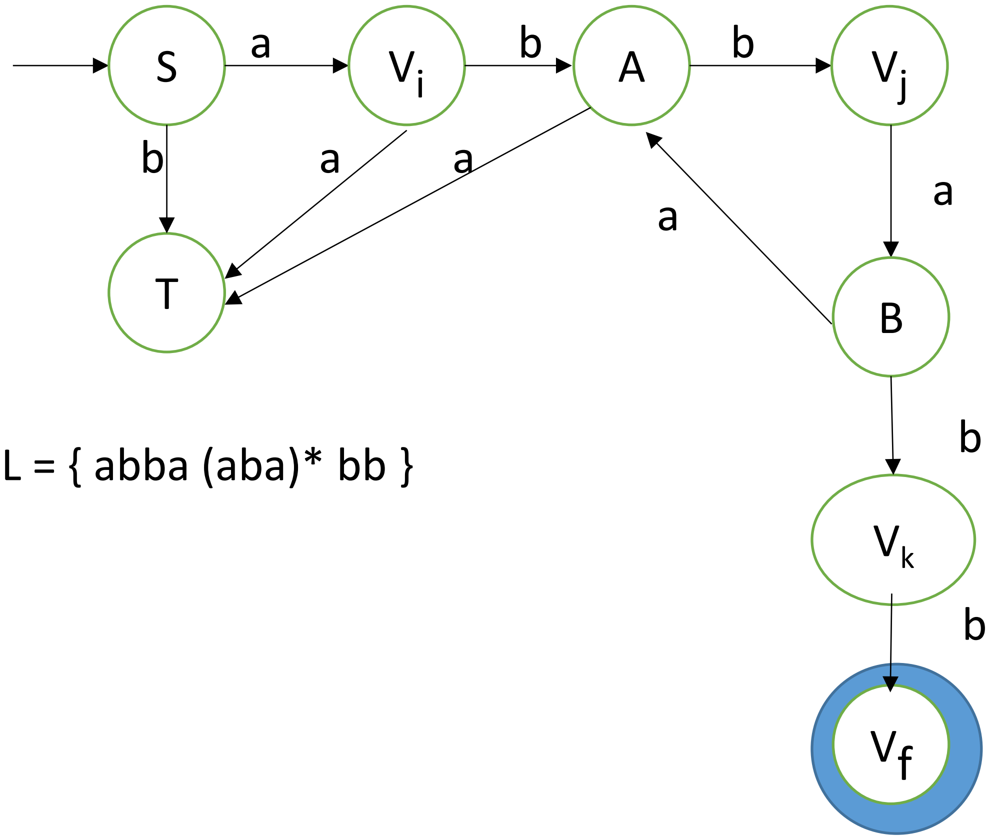
$q_0 \Rightarrow aq_1 \Rightarrow aaq_2 \Rightarrow aabq_2 \Rightarrow aabaq_f \Rightarrow aaba.$

$\delta(q_0, a) = \{q_1\}$	$q_0 \rightarrow aq_1$
$\delta(q_1, a) = \{q_2\}$	$q_1 \rightarrow aq_2$
$\delta(q_2, b) = \{q_2\}$	$q_2 \rightarrow bq_2$
$\delta(q_2, a) = \{q_f\}$	$q_2 \rightarrow aq_f$
$q_f \in F$	$q_f \rightarrow \lambda$

δ(q0, a) = q1	q0 → aq1
δ(q1, a) = q2	q1→ aq2
δ(q2, b) = q2	q2→ bq2
δ(q2, a) = qf	q2→ aqf
qf ϵ F	qf ϵ F

1. Construct DFA that accepts the language generated by the grammar

$S \rightarrow abA$
$A \rightarrow baB$
$B \rightarrow aA \mid bb$



L = { abba (aba)* bb }

2. Find regular grammar that generate L (aa*(ab + a)*)

$G = (V, T, S, P)$, where
$V = \{S, A, B\}$,
$T = \{a, b\}$,
$P = \{S \to aA,\ A \to aA|aB|\lambda,\ B \to bA\}|$

The derivation of a string $aaaababa$:

$$S \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow aaaaB \Rightarrow aaaabA \Rightarrow aaaabaB$$
$$\Rightarrow aaaababA \Rightarrow aaaababaA \Rightarrow aaaababa.$$

Q0 → a Q1
Q1 → a Q1 | aQ2 | λ
Q2 → b Q1

3. Construct right linear and left linear grammar for the language

L = {$a^n b^m$: n >= 2, m >=3}

<u>Right Linear:</u>

S → aaA
A → aA | B
B → bbbC
C → bC | λ

or

S → aaA
A→ aA | bbbB
B → bB | λ

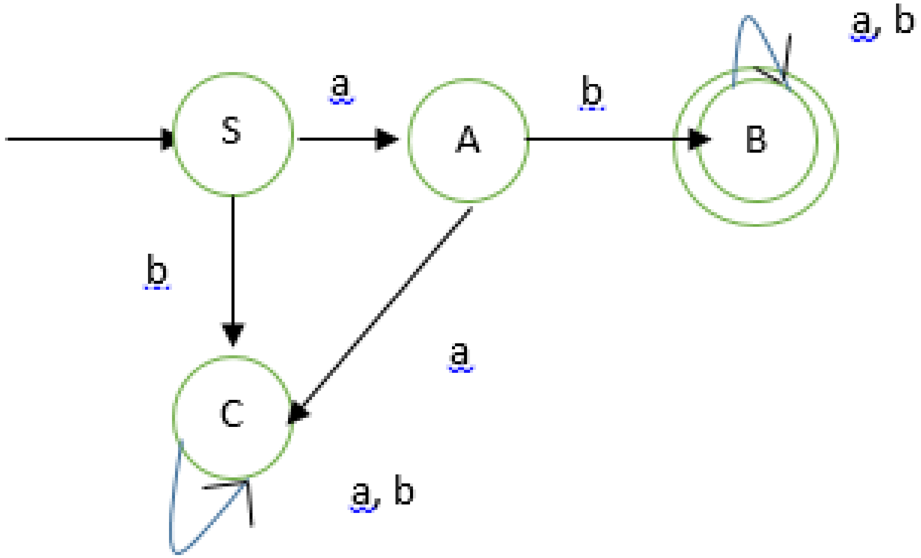
<u>Left Linear:</u>

S → Abbb
A → Ab | B
B → Caa
C → Ca | λ

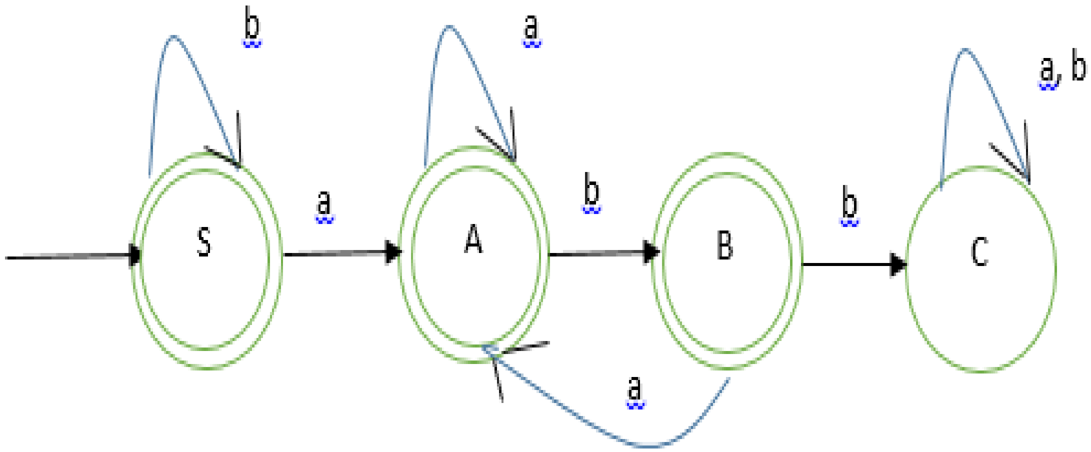
1) Draw DFA for the following

S → 01 A
A → 10B
B → 0A | 11

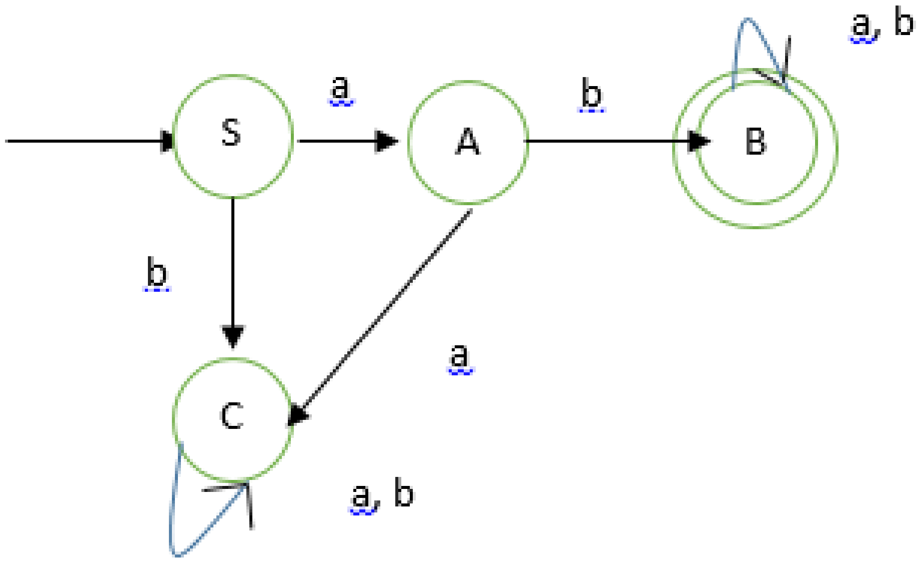
a) Construct regular grammar for the following



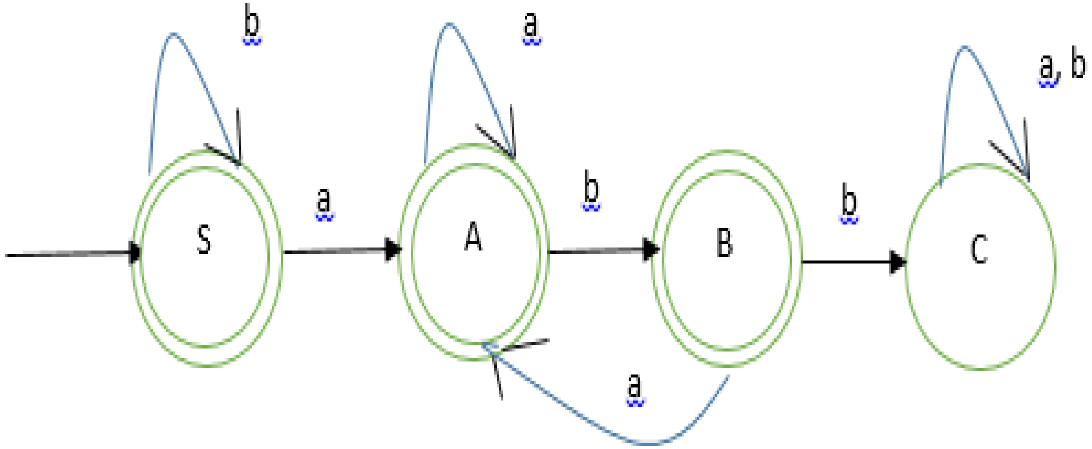
b) Construct regular grammar for the following



a) Construct regular grammar for the following



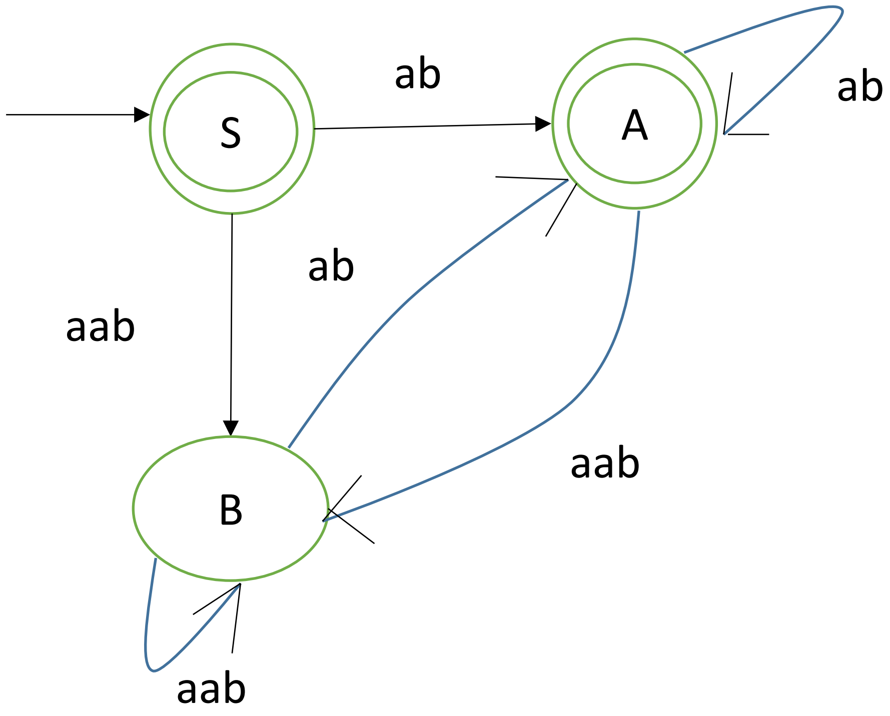
b) Construct regular grammar for the following



S → aA | bC
A → aC | bB
B → aB | bB | λ
C → aC | bC

S → aA | bS | λ
A → aA | bB | λ
B → aA | bC | λ
C → aC | bC

a) Obtain Right Linear Grammar for the following:



S → abA | aabB | λ
A → abA | aabB | λ
B → aabB | abA