10/9

# Module-3

## # Regular language (RL)

* Regular language are formal languages that regular expressions can describe and can also be recognised, by finite automata
* They can be used to define sets of strings such as ~~words~~, sequence of characters or words that follow specific pattern. They are important in computer science & theoritical concepts of computer science because they form a foundation for understanding the theory of computation & the design of compilers and other software tools.

→ Formal Definition - A RL can be defined as the collection of strings that are recognized by the finite automata. An FA is a 5 tuple $(Q, \Sigma, \delta, q_0, F)$. FA's & RE specify patterns or rules that define a language such as sequence of characters that must or must not appear in the strings.

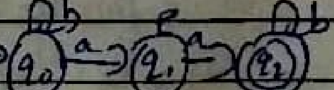
* The words in a regular language must follow the rules specified by FA or RE to be a part of the language.
* Examples of ~~RL~~ RE:

(i) Binary string that represents even no's

(ii) Set of string that contains exactly 2 a's

(iii) Set of all the binary no's that are divisa-ble by 3.

(iv) Set of all the strings that contain the sub-string 01

→ (ii) RE = $b^* aab^*$ , FA = 

RL = {baab, bbaabb, baabb, bbaab, ---}

(iv) RE = $(0+1)^* 01 (0+1)^*$

RL = {010100, 100110, 110111, 000100, ---}

# Properties of RL:

* RL are closed under union, concatination & ~~Kleen~~ kleen closure (kleenestar), this means that if 2 RL's are combined using one of the operations, the resulting language will also be regular

(i) Union: Let $L_1$ & $L_2$ be a regular language then $L_1 \cup L_2$ is also a regular language
ex:- $L_1 = a$, $L_2 = b$ then $L_1 \cup L_2 = a + b$ (RL) is also

(ii) Concatination: Let $L_1$ & $L_2$ be a regular language then $L_1 \cdot L_2$ is als a regular language.
ex:- $L_1 = a$, $L_2 = b$ then $L_1 \cdot L_2 = ab$ (RL)

(iii) Kleene star - Let '$L$' be a regular language the $L^*$ is also a regular language.
ex:- $L = a$ then $L^* = a^*$ (RL).

# Pumping Lemma for RL:

Pumping lemma puts forward that for a RL 'L', there exists a constant pumplength such that any string in the language can be decomposed into 3 parts and these parts can be repeated any no of times (by pumping the middle part) while still being in a language. This can be stated mathematically as follows, let 'L' be a RL an 'P' be constant pump length specified by the pumping lemma. Then for any string 'W' in 'L' such that $|W| \geq P$ if condition satisfied ~~if then~~ it can be decomposed into 3 parts $(x, y, z)$, where the following conditions

(i) $|xy| \leq P$

(ii) $|y| \geq 1$

(iii) For any non-negative integer $k$, $xy^kz$ is in L

(if contra to prove sub $k=0, k=1$ not a

# $WW^R \Rightarrow$ then take W of any string of no length)

. $W = aabb$, $W^R$ = (reverse) $bbaa$

worked for aaabbb

1Q) Using pumping lemma prove that the Language $L = \{a^n b^n / n \geq 1\}$ is not regular.

Sol: Step 1: write language for given $a^n b^n$ such that n=1,2,...

$L = \{ab, aabb, aaabbb, aaaabbbb, \ldots\}$

Step 2: Pick one string among it

$w = aabb$, let pumplength $= 4 \ (\geq 4)$

$|w| = 4$

$|w| \geq P$

$|4| >= 4 \Rightarrow$ true

(condition satisfied so divide into 3 parts)

Step 3: $W = \underset{x}{a}\,\underset{y}{ab}\,\underset{z}{b}$ (or) $\underset{x}{aa}\,\underset{y}{b}\,\underset{z}{b}$ (or) $\underset{x}{a}\,\underset{y}{a}\,\underset{z}{bb}$

Now considering $W = \underset{x}{a}\,\underset{y}{ab}\,\underset{z}{b}$

Step 4: (i) $|xy| <= P$

$|(1)(2)| \leq P \Rightarrow |3| <= 4 \Rightarrow$ true

(ii) $|y| \geq 1$

$\Rightarrow |2| \geq 1 \Rightarrow$ true.

(iii) $xy^k z$, $k \geq 0$

for $k=0 \Rightarrow a(ab)^0 b \Rightarrow ab \in L \{ab, aabb, \ldots\}$

for $k=1 \Rightarrow a(ab)^1 b \Rightarrow aabb \in L$

for $k=2 \Rightarrow a(ab)^2 b \Rightarrow aababb \notin L$.

Therefore, given Language is not a regular lang.

$\Rightarrow$ Step 2: Picking $w = aaabbb$, let pumplength $= 6$

$|w| = 6$

$|w| \geq P \Rightarrow 6 \geq 6 \Rightarrow$ true

should satisfy $|xy| \leq P$ condition

Step 3: $W = \underset{x}{aa}\,\underset{y}{ab}\,\underset{z}{bb}$

Step 4: (i) $|xy| \leq P$ → xyz should not be having 0

$|4| \leq 6 \Rightarrow$ true

(ii) $|y| \geq 1 \Rightarrow 2 \geq 1 \Rightarrow$ true.

(iii) $xy^k z$ ; $k \geq 0$

for $k=0 \Rightarrow aa(ab)^0 bb \Rightarrow aabb \in L$

for $k=1 \Rightarrow aa(ab)^1 bb \Rightarrow aaabbb \in L$

$k=2 \Rightarrow aa(ab)^2 bb \Rightarrow aaababbb \notin L$.

∴ given L is not a RL.

(i) Kleene star closure:

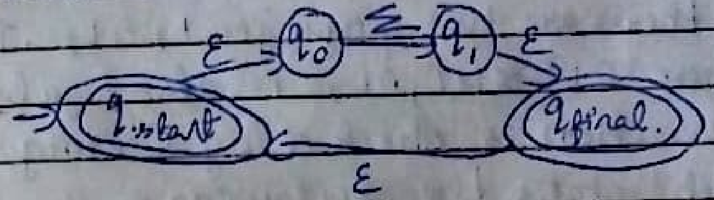
* If a language $L_1$ is a regular language(RL) then its clean star closure $L_1^*$ will also be regular.
* How to make $L_1^*$ also regular:

(i) Make a finite automata for the language

(ii) Create a new start state. Join to a original state ~~to~~ with a null transition.

(iii) Create a new final state. Join the original final state to new final state with a null transition. The original final state will not be a final state anymore.

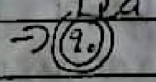
(iv) Make a null transition from the new final state to the new start state.

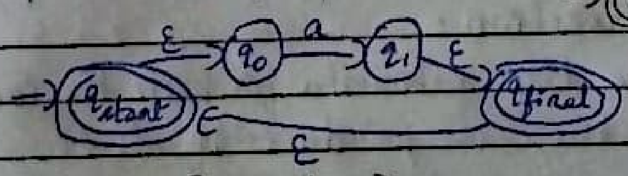



for ex: $L_1 = a$ (Regular LF, Regular exp)

then $L_1^* = a^*$ will also be regular.

to make it closure.



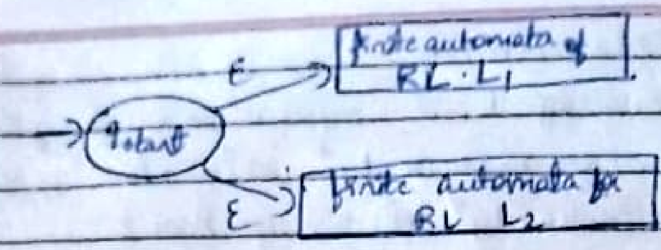

(ii) Union $(L_1 \cup L_2)$

* Let us assume there are two languages $L_1$ & $L_2$ the union of these two ~~no~~ RL is $(L_1 \cup L_2)$, will also be regular.
* The process for union:

(i) Make the FA for both $L_1$ & $L_2$ seperately

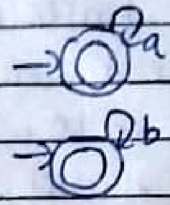
(ii) Add a new start state, join the new start state to the automata's original start state by null transitions. The final state

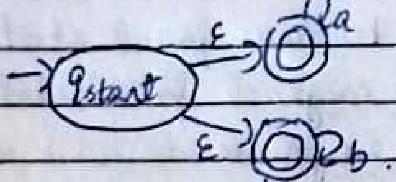
~~(iii)~~ remains the same as per ~~for~~ the two automatas.




ex:- $L_1 = a^*$ $\Rightarrow$

$L_2 = b^*$ $\Rightarrow$




(iii) Intersection ($L_1 \cap L_2$)

* If there are two languages $L_1$ & $L_2$. The intersection of these RL i.e. $L_1 \cap L_2$ is also regular. The intersection is checked by Demorgan's law which states the following:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

* The process for the intersection is slightly different from other properties. Steps to implement the intersection:

(i) Make the automata for both the languages

(ii) Take the cross product of all the states from both the automations

(iii) The final state is the one that has the final state of first automata & second automata

Ex:- Let $L_1 = \{a, a^2, a^3, ...\}$

$L_2 = \{a^2, a^4, a^6, ...\}$

$L_1 \cap L_2 = \{a^2, a^4, a^6, ...\}$

② $L_1 = \{10, 110, 101, 100, 1110, 1011, 1010, ...\}$

R.E $= 1^*0(0+1)^*$

$L_2 = \{01, 001, 010, 011, 0001, 0101, ...\}$

RE $= 0^*1\underline{(0+1)^*}$ this will remain (final same state)

$\Rightarrow L_1 \cap L_2$ is regular.

(iv) Concatination:

Let use assume two language $L_1$ & $L_2$. The ~~intersection~~ concatination of these R.L i.e $L_1 . L_2$ is also regular

Steps to implement concatination:

(i) Make the FA for both the languages $L_1$ & $L_2$ separately

(ii) Suppose the order of concatination is $L_1 . L_2$. Make a null transition from the final state of $L_1$ to the start state of $L_2$. Make the final state of $L_1$ as non-final. The final state of $L_2$ remains same

ex:- $L_1 = ab$ , $L_2 = ba$

then $L_1 . L_2$

$L_1$ = 

$L_2$ = 

$\rightarrow L_1 . L_2 \Rightarrow$ 

make $L_1$ final state to non-final & ε transition to $L_2$

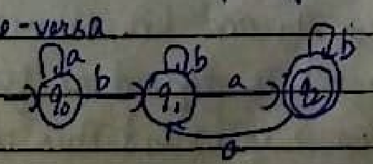
(v) Complement:

$L_1'$ represents the complement of the regular language $L_1$. lets assume an alphabet, A, where $A^*$ contains the language $L_1$. The complement is defined as $A^* - L_1$, which is also regular.

The process for the complement

There is one step for the complement process,

1. Make the accepting states non-accepting & vice-versa

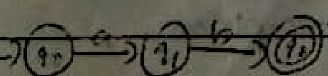


L is a regular & it is FA

FS $\rightarrow$ Non FS

NFS $\rightarrow$ FS

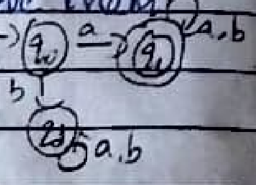
ab

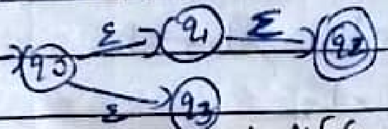
 $\rightarrow$ complement of this is

 (change final state to non-final & vice versa)

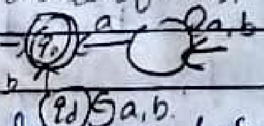
(vi) <u>Reversal</u>
Lets assume a regular language $L_1$. The reverse of $L_1$ is represented as $L_R$. The reversed regular language consist of the reverse of all strings present in $L_1$.
The process of reversal to reverse a language, we ~~to rev~~ perform the following steps:
① Make a definite finite automation for the regular lang
② Make the final state the new initial state and the initial state the new final state.
③ Reverse the direction of the transitions of M to make M'
If there are dangling states, we can remove them from the automation.

→ ex:- $a(a+b)^*$




reversal of this.




change F.S to N.F.S & arrow's change

(vii) <u>Difference</u> - Consider the two languages $L_1$ & $L_2$. The difference between these RL's is shown by $L_1 - L_2$. This means that the strings produced in $L_1$ are regular & not the ones in $L_2$. This difference is $L_1 \cap \bar{L_2}$ which means the intersection of $L_1$ & the complement of $L_2$.

$L_1 - L_2 = L_1 \cap \bar{L_2}$

The <u>process for difference</u> The process to take the difference between the languages is as follows:
① Make the automations for the regular languages $L_1$ & $L_2$
② Take the complement of the second language (the lang^e that is to be subtracted $L_2$ in this case)
The steps for complement are explained.

(viii) <u>HomoMorphism</u> - is allowing a string

For a regular language, $L_1$, and the Homomorphism, H, the language's alphabet is defined as follows.
$H(L_1) = \{H(S) \mid$ where S is in $L_1\}$. This is also a RL
The process for Homomorphism.

$H(L) = \{h(w) \mid w \in L\}$ where $h$ is [illegible] homomorphism

[illegible]

- Homomorphism is basically [illegible] substitution.

Homomorphism is closed under R.L's. The algorithm is defined for the regular expression the language.
- Define the homomorphism as the operation on the regular expression of a regular language $L_1$.
- Prove that $L_1(H(R)) = H(L_1(R))$.
- Assume $R$ is such that $L_1 = L_1(R)$. let $R' = H(R)$ & resultantly $H(L_1) = L_1(R')$.

Ex: Suppose that $H(0) = ac$ & $H(1) = bc$.
Let a regular expression of a R.L be $L_1 = 00H$. Then $H(L_1)$ is the language of $acacbc + ac^t \ldots 0^*$. By equating, we see that $H(L_1)$ is also a regular language.

(x) Inverse Homomorphism: Consider $H$ as the Homomorphism & the R.L $L_1$. The alphabet of $L_1$ is the output of the language of $H$. Therefore, $H^{-1}(L_1) = \{H(s) \mid$ where $s$ is in $L_1\}$ is also a R.L.

How → # $h^{-1}(L)$.

Eg: Let $h(0) = a$, $h(1) = b$, $h(2) = ab$

$\Sigma = \{0, 1, 2\}$

$\Gamma = \{a, b\}$

$L = \{abab\}$

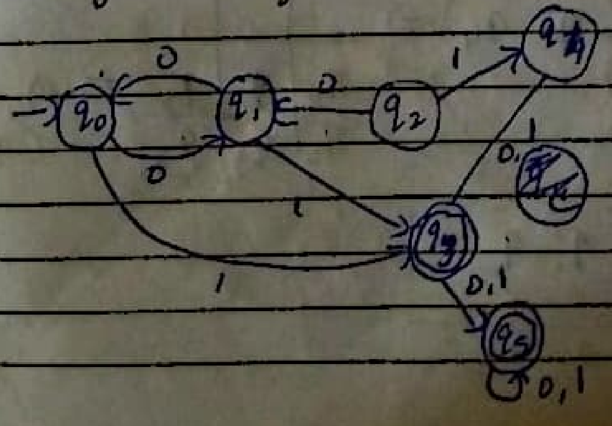
$h^{-1}(L) = \{\underset{abab}{0101}, \underset{ab\,ab}{22}, \underset{a\,b\,ab}{012}, \underset{ab\,a\,b}{201}\}$

$h(h^{-1}(L)) = \{abab, abab, abab, abab\}$

22/9/25

# ⇒ Minimization of the DFA

Q)



if ( states not there then no change)
→ $(q_0, q_1)$
$(q_3, q_3)$

## Transition table

| | 0 | 1 |
|---|---|---|
| $q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_0$ | $q_3$ |
| $q_2$ | $q_1$ | $q_4$ |
| $q_3$ | $q_5$ | $q_5$ |
| $q_4$ | $q_3$ | $q_3$ |
| $q_5$ | $q_5$ | $q_5$ |

✓ → (FS, FS)
(indistinguishable) (NFS, NFS)

× → (FS, NFS)
(distinguishable) (NFS, FS)

$N.F.S = \{q_0, q_1, q_2, q_4\}$
$F.S. = \{q_3, q_5\}$

vertical
removing
start state

| | | | | | |
|---|---|---|---|---|---|
| $q_1$ | ✓ | | | | |
| $q_2$ | ✓ | ✓ | | | |
| $q_3$ | × | × | × | | |
| $q_4$ | ✓ | ✓ | ✓ | × | |
| $q_5$ | × | × | × | ✓ | × |
| | $q_0$ | $q_1$ | $q_2$ | $q_3$ | $q_4$ |

horizontal start to previous final state
removing final state

$(q_{..}, q_{..}), (q_3, q_5), q_2, q_4$

→ Pair $(q_0, q_1)$ ⇒

| | 0 | 1 |
|---|---|---|
| $q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_0$ | $q_3$ |

$(q_1, q_0)$ $(q_3, q_3)$
↳ not there so
× , × ⇒ ×

Pair $(q_0, q_2)$ ⇒

| | 0 | 1 |
|---|---|---|
| $q_0$ | $q_1$ | $q_3$ |
| $q_2$ | $q_1$ | $q_4$ |

$(q_0, q_1)$ $(q_3, q_4)$
✓ , × ⇒ ×

Pair $(q_0, q_4)$ ⇒

| | 0 | 1 |
|---|---|---|
| $q_0$ | $q_1$ | $q_3$ |
| $q_4$ | $q_3$ | $q_3$ |

= ×

$(q_1, q_3)$ $(q_3, q_3)$
× ↳ not there so ×

Pair $(q_1, q_2)$ ⇒

| | 0 | 1 |
|---|---|---|
| $q_1$ | $q_0$ | $q_3$ |
| $q_2$ | $q_1$ | $q_4$ |

$(q_0, q_1)$ $(q_3, q_4)$
✓ × ⇒ ×.

Pair $(q_1, q_4)$ ⇒

| | 0 | 1 |
|---|---|---|
| $q_1$ | $q_0$ | $q_3$ |
| $q_4$ | $q_3$ | $q_3$ |

$(q_0, q_3)$ ⇒ ×
$(q_3, q_3)$ ⇒ Not pair ⇒ ×.
⇒ ×

× $(q_0, q_4)$

Pair $(q_2, q_4)$ ⇒

| | 0 | 1 |
|---|---|---|
| $q_2$ | $q_1$ | $q_4$ |
| $q_4$ | $q_3$ | $q_3$ |

$(q_1, q_3)$ ⇒ ×
$(q_4, q_3)$ ⇒ Not pair
⇒ ×

Pair $(q_3, q_5)$ ⇒

| | 0 | 1 |
|---|---|---|
| $q_3$ | $q_5$ | $q_5$ |
| $q_5$ | $q_5$ | $q_5$ |

$(q_5, q_5)$ = Not
$(q_5, q_5)$ = Not
⇒ ×.

New table

| | | | | | |
|---|---|---|---|---|---|
| $q_1$ | ✓ | | | | |
| $q_2$ | × | × | | | |
| $q_3$ | × | × | × | | |
| $q_4$ | × | × | × | × | |
| $q_5$ | × | × | × | ✓ | × |
| | $q_0$ | $q_1$ | $q_2$ | $q_3$ | $q_4$ |

Transition table:

$(q_0, q_1), (q_3, q_5), \underbrace{q_2, q_4}$ - - - - write all remaining states

| | 0 | 1 |
|---|---|---|
| $\rightarrow (q_0, q_1)$ | $(q_1, q_0)$ | $(q_3, q_5)$ |
| $* (q_3, q_5)$ | $(q_3, q_5)$ | $(q_3, q_5)$ |
| $q_2$ | $(q_1, q_0)$ | $(q_4)$ |
| $q_4$ | $(q_3, q_5)$ | $(q_3, q_5)$ |

→ will get $\{q_3, q_5\}$ ⇒ $\{q_5\}$ but see if it is in pair like it is in pair $\{q_3, q_5\}$ so write $\{q_3, q_5\}$

Transition Diagram:

9/25