**Unit -2**

**Regular Expression**

A language (set of strings/words) can be represented by following manner.

Set theory listing method

Set theory property method

Natural language like English

**Example**: Set of all possible combination of a is represented by

L= {$˄$, a, aa, aaa, aaaa, aaaaa,……..}

L= {$a^{n}|n\geq 0\} $

But in finite automata we use regular expression because of accepted by finite automata

**r =** $a^{\*}$

In natural language $a^{\*}$ means it is a language which consist of all the possible combination of a.

**Definition of Regular expression:** It is the way of algebraic representation/description of regular language using the set of operators $+, ∙, \*$ .

The language accepted by finite automata is called Regular language and easily described by regular expression using recursive definition.

**Operator used in regular expression:**

|  |  |  |  |
| --- | --- | --- | --- |
|  Symbol |  In R.E. | Meaning | Example |
| $$∪$$ | $$+$$ | Either /or(Parallels’) | a + b |
| $$∙$$ | $$∙$$ | Series | a.b |
| $$\*$$ | $$\*$$ | Iteration (Repeat ion of 0 or more time) | $$ a^{\*}$$ |

|  |
| --- |
|  **( )** |
| $$\*$$ |
| $$∙$$ |
| $$+$$ |

**Precedence of operator:**

 **Higher**

 **Lower**

**Recursive definition of regular expression:**

**Basis:**

Rule1.Empty set$ ∅$, empty language $˄$ and any literal character constant a$ \in ∑$ is a regular expression.

**Recursive:** if$ r\_{1} and r\_{2}$ be two regular expression then using operators $+, ∙, \*$ .

Rule2. Union of two regular expression ($r\_{1}+r\_{2}$) is also regular expression.

Rule3. Concatenation of two regular expression ($r\_{1}∙r\_{2}$) is also regular expression.

Rule4. Closure of a regular expression ($r^{\*}$) is also regular expression.

Rule5. ($r$) is also regular expression.

Rule6. $r^{+}$ is also regular expression.

**Algebraic laws for regular expression:**

1. $∅+R=R$ = $∅+R$
2. $∅∙R=R.∅=∅$
3. $Λ∙R=R∙Λ=R$
4. $Λ^{\*}=Λ \& ϕ^{\*}=Λ$
5. $R+R=R$
6. $R^{\*}∙ R^{\*}= R^{\*}$
7. $R∙R^{\*}=R^{\*}R=R^{+}$
8. $(R^{\*})^{\*}= R^{\*}$
9. $Λ+R∙R^{\*}= R^{\*}$
10. $\left(PQ\right)^{\*}P=P \left(QP\right)^{\*}$
11. $\left(P+Q\right)^{\*}=\left(P^{\*}+ Q^{\*}\right)^{\*}=\left(P^{\*}∙ Q^{\*}\right)^{\*} $
12. $\left(P+Q\right)R=PR+QR $

 $R\left(P+Q\right)=RP+RQ$

Note: $\left(P^{\*}∙ Q^{\*}\right)= Λ+(P+Q)^{\*}Q$

**Transition Graph:** It is collection of 3 things.

1. Finite set of state in which at least one of them is start state & some (may be none) as final state.

Initial & final

1. Alphabet$∑$
2. A finite set of transition (edge label) for how to go from some state to some other.

So it is labeled directed graph of state. It is denoted by symbol$ δ$ .

 $δ\left(q\_{i}, a\right)=q\_{j}$



**Generalized/Extended Transition Graph:** It is just like Transition graph but directed edge connecting some pair of state and given by r.e. It is denoted by$\hat{δ}$.

$$ \hat{δ}\left(q\_{i}, re\right)=q\_{j}$$



**Kleen’S theorem:**

The set of regular languages, the set of NFA-recognizable languages, and the set of DFA-recognizable languages are all the same

Let L be a language over an alphabet$∑$. Then L is regular if and only if it is the language accepted by some finite automaton with alphabet$∑$.

Finally for each regular expression there must be finite automation as $Λ$- NFA.

So it states that any regular language can be accepted by finite automata $(Λ$- NFA, DFA, and NFA).

Its also states that the language accepted by any finite automata is always regular.

**Regular expression to Finite automaton:** According to Kleen’s theorem let r be a regular expression then there exist a $Λ$- NFA.



1. For **r.e.= a+ b**

It is the case of parallel path so join it as follows:

First make the finite automata for a & b.

Now take two extra states as a new initial and new final state. The new initial state is joined with both initial state of finite automata transition diagram via $Λ$- transition and both the final state is joined with new final state of finite automata transition diagram through $Λ$- transition.

 2. For **r.e.= a**$⋅$ **b**

It is the case of series but no need of extra state:

 First make the finite automata for a & b. So initial state is the initial state of the first machine and final state is the final of the second machine.

Now the final state of the first machine is joined via $Λ$- transition to the initial state of the second finite automata machine.

3. For **r.e.=** $a^{\*}$

It’s a case of looping and similarly parallel path :

First make the finite automata for only a.

Now take two extra states as a new initial and new final state. The new initial state is joined with initial state of finite automata transition diagram via $Λ$- transition and final state is joined with new final state of finite automata transition diagram through $Λ$- transition.

And also join the previous final state with previous initial then join the newly initial state with previous newly final state.

**Finite automaton to Regular expression:** The regular expression can also be given by its equivalent DFA. For the language of DFA transition diagram in terms of Regular expression use Arden’s theorem.

**Arden’s theorem:** Let P, Q and R be regular expression on$∑$. Then if P does not contain null$(P\ne Λ)$, the equation $R=Q+RP $ has a unique solution given by$R=QP^{\*}$.

Proof: The equation $R=Q+RP $ , has a solution $R=QP.^{\*}$ so the value of R satisfied the given equation. Put $R=QP^{\*}$ in the right hand side of the equation $R=Q+RP $

 $ R=Q+QP^{\*}P $

 $R=Q+QPP^{\*} $

 $R=Q(Λ+QPP^{\*}) $

 $R=QP^{\*}$

Note: 1.Arden theorem is applied only in case of DFA so no null transition.

 2. There is only one initial state.

**Pumping Lemma for Regular language:**

It is property of regular language given by Y. Bar Hillel, Micha A Perles & Eli Shamir in 1961.

It is a useful tool for disproving the regularity of a given language.

Pumping lemma for regular language signifies the fact that if a regular language is infinite, then it must contain string of that form.

Pumping means generate input string by pushing a character and lemma means parts of theorem so called pumping lemma.

**Formal statement Pumping Lemma for Regular language:** “ Let L be a regular set, then there exists a positive integer constant m such that , if ω is any word in L such that the length of ω is at least m that is $|ω|\geq m$ and we can rewrite ω=xyz in such a way that

1. $|xy|\leq m$
2. $\left|y\right|\geq 1 , y\ne ˄, (y is pumped)$
3. $For all i\geq 0, xy^{i}z\in L.$

**Application of Pumping Lemma**: It is used to check whether a given language is regular language or not.

**Step:**

1. Assume that given language L is regular language. Let m be the constant of pumping lemma as $m=\left|w\right|+1.$
2. Take a language string ω from L such that $|ω|\geq m$ & write ω =xyz. It means ω can be broken into three parts as x, y and z.
3. Find a suitable $i\geq 0,$ $xy^{i}z\in L.$

**Note:**

In this lemma proof is done by contradiction by pigeonhole principle.

Pumping lemma should not used to proof that a given language to be regular.

Which string ω to select is very important and it is the key to solution using pumping lemma so always take **larger string**.

**Properties o f Regular language:** There are two types of properties as closure properties and decision properties.

Closure properties means the operation on any two elements of the set produces another element of the same set.

Decision properties are the properties in which the solution is only yes/no.

**Closure properties of Regular language**:

1. If $L\_{1}$ and $L\_{2 }$ be two regular language expressed by regular expression, then

$L\_{1}$ $∪$ $L\_{2 }$,$ L\_{1}$ $.L\_{2 }$and $L^{\*}$ be also Regular language. So we say that if $L\_{1}$ and $L\_{2 }$ are set two regular language class then$L\_{1}$ $∪$ $L\_{2 }$,$ L\_{1}$ $.L\_{2 }$and $L^{\*}$ are also belongs to set of regular language class.

2. If L is regular then compliment of the regular language $L^{c} $is always regular language.

3. If L is regular then reversal of the regular language $L^{r} $is also regular language.

4. If $L\_{1}$ and $L\_{2 }$ be two regular language then intersection of both $L\_{1}$ $∩$ $L\_{2 }$is also regular language.

5. If $L\_{1}$ and $L\_{2 }$ be two regular language then set difference of both $L\_{1}$ $-L\_{2 }$is also regular language.

|  |  |
| --- | --- |
| Operation | Regular Language |
| Union | Yes |
| Catenation | Yes |
| iteration | Yes |
| Reversal  | Yes |
| intersection | Yes |
| Subtraction | Yes |
| complementation | Yes |
| Homomorphism | Yes |

**Decision property of Regular Language:** A finite automaton is like a computer .It receives a input and produces output Yes or No as accept or reject. All the property which has solution yes or no is called decision property.

**Membership:**

1. For RE: Given a string $ω$ & R.E.

Is $ω$ an element of RL?

It means $ω\in L or not?$

1. For FA: Given a string $ω$ & FA M.

Is $ω$ an element of L(M)?

Is $ω$ accepted by FA M?

Algorithm: If there is path from initial state to final state then any string $ω$ is always accepted by FA.

Example:

Finiteness

Here L (M) = $a^{\*}(a+b)c^{\*}$

1. **Emptiness:** Is language of FA Machine L (M) =$ ϕ$?

The above FA transition is not empty.

 4. **Finiteness:** Is L (M) finite?

Example1.



Yes this DFA is finite because it accepts only aa,ab,ba,bb.

L(M) ={aa,ab,ba,bb}.

5. **Equivalent of two r.e**.: Two regular expression $r\_{1 }$and $r\_{2}$ is equivalent iff both have the same set of string as language.

$ r\_{1 }and r\_{2}$ are equal iff

$$L\left(r\_{1 }\right)=L\left(r\_{2}\right)$$

Example: Let $r\_{1 }$=$\left(P+Q\right)^{\*}$ and $r\_{2}=\left(P^{\*}+ Q^{\*}\right)^{\*}$ are equivalent because both have same set of word.

6. **Equivalent of two Finite Automata**: Two finite automata $M\_{1} \& M\_{2} $are equivalent iff they accept same language as L ($M\_{1}$) = L$(M\_{2})$.

For this we use of comparison method where pair of state either belongs to final or non final state.

**Regular language and finite automata :( Application of Regular expression & finite automata)** A variety of software application from different area can be simplified using convert RE into Computer implementation of its FA. So Application of RE and FA are divide into two category as system software like language compiler, Operating system utility , program development tool etc. and Application program like text editor ,recognizer.

1.**Lexical analyzer in compiler** : The job of the compiler is to translate high level language like (C, java , C++ ) into low level language (machine level language).The first phase of compiler scans the source file character by character and identify token using lex command. It also recognizes token from text file (source program ) using regular expression then make DFA to detect valid word from source file.

Ex. RE in C language,

 re = $l\left(l+d\right)^{\*}$ where $l=A to Z \& a to z$

 $d=0+1+2+3+4+5+6+7+8+9$

2. **Text editor**: As find and replace are used for processing the text.

In unix/linux line editor command (ed command ) **grep** and **egrep** for generating regular expression.

grep$\rightarrow $ global regular expression print

Syntax: grep [poption ] pattern [file..]

egrep$\rightarrow $extended global regular expression print

3. Many application

$\rightarrow $Many forms of pattern – recognition (WWW search engine use)

$\rightarrow $Sequentila circuit design like counter and adder etc

$\rightarrow $Communication protocol

$\rightarrow $Software specification and design

**Myhill –Narode Theorem:** The Myhill- Narode theorem was proposed by John Myhill and Anil Narode for proving the given language is non regular.

Given a language L, two strings x and y are said to be in the same class if for all possible strings z either xz or yz are in L or both are not.

The myhill- Nerode theorem states

1. A language L divides the set of all possible strings into mutually exclusive classes.
2. If L is regular, the number of classes created by L is regular.
3. If the number of classes l creates is finite ,then L is regular.