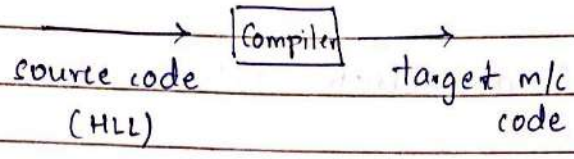
05-08-19

# Introduction to Compiler Design.

→ **Compiler.**

Compiler is a translator which converts source code into target machine code.




HLL eg: C, Java, C++ etc

compiler eg: Turbo C, gcc.

* Compiler is distinguished as

| (Analyser) frontend of compiler | (Synthesiser) back-end of compiler. |
|---|---|
| → dependent of source program | → independent of source program |
| → independent of m/c architecture | → dependent on machine architecture |

- Language Processing System.

Skeletal Source Program
↓
Preprossor
↓ Pre-prossed source program / Modified
Compiler
↓ Target Assembly code
Assembler
↓ Relocatable object code
Linker ← Libraries and Relocatable Obj Files.

Loader

↓

~~Absolute~~ Machine Code

Preprosscessor - System software which includes macros in the code

Ex: #define..

Linker - System Software which includes the header files from libraries.

Loader - loads the machine code into Primary memory.

→ If addresses are fixed then it is called "Absolute" code.

→ Fortran was the first HLL & first compiler was developed

→ Parser generators

such as Yacc ⇒ Syntax Analysis.

→ IDE (Integractive Development Environment)

contains editors, linkers, debuggers etc

→ Fortran compiler was written in binary (machine language)

Pascal compiler was written in Fortran. and so on.

→ Parser = Syntax Analyser.

→ Lexical Analyser = Scanner.

Compilation in two steps.

Analysis determines the operations implied by the source program which are recorded in a tree structure.

Synthesis takes the tree structure and translates it to target program

- Compiler passes:

  A collection of phases is done once or multiple times. (single pass or multi-pass).

Scanner

Lexical analysis - collects sequences of characters into meaningful units called tokens.

Ex: a[index] = 4+2.

It may enter literals (string consts) into literal table.

Parser.

Syntax Analysis - determines the structure of program

Result → Parse tree

leaf nodes in Parse tree is the statement.

Semantic Analyser

determines its running time behaviour prior to executions.

declaration & datatype checking → Static semantics.

→ Extra information will be computed into Syntax tree to give Annotated Syntax tree.

i.e, name of tokens will be added.

07-08-19

## Source Code Optimizer.

Constant folding (optimization) is performed on annotated tree.

Ex: Before a[i] = 4+2 ; After a[i] = 6

→ It assigns a temp variable for optimization

## Code Generator.

takes Intermediate code or IR and generate code for target machine

* address of a[i] = base or start address + i x sizeof(datatype)

a[i] = 6 ⇒

```
Ex:  MOV R0, index     ⇒ move value of index in R0 (Resistor)
     MUL R0, 2         ⇒ Multiply 2 to value in R0, store in R0
     MOV R1, &a        ⇒ move value of &a in R1
     ADD R1, R0        ⇒ add address of a and 2*i, store in R1
     MOV *R1, 6        ⇒ value of (*R1) R1 should be 6
```

## Target Code optimizer

Improves the target code

→ Efficient address modes are assigned

→ Efficient operations replace less efficient ones.

```
Ex:  MOV R0, index
     SHL R0
     MOV &a[R1], 6
```

SHL ⇒ shift left by one bit.

Ex: 0000 0100 = 4 i.e. 4 x 2 = 8

0000 1000 = 8. one bit shift = mul by 2

Q a=b+c. Write o/p for each phase
Q Find e-num data-type.
Q. address form for 2D array.

classmate Date Page

* Data Structures used among different phases.

• Symbol Table

- keeps info associated with identifiers: datatypes, const, func etc
- Interacts with almost every phase of compiler.
- Access opperation need to be constant-time
- One or several hash tables are used

Hash table

Keys are inserted in each bucket (slot) of hash table using an efficient hash function.
Ex: H[i] = key % Primeno.

When multiple keys ~~are~~ end up with same hash address or slot, it is called collusions.
Collusions are solved by either placing in next empty bucket or by creating linked lists and inserts.
!!! Basic operations performed are Insertion, Deletion & Searching.

A strong hash function is used to avoid collusions and uniform insertion.

• Literal Table

- Stores string constants hence reduces size of program

13-08-19

## Function of a Scanner.

→ Reading characters from source code and form them into logical units called tokens.

→ Tokens are logical entities defined as an enumerated datatype.

→ Relation b/w Tokens and its Strings.

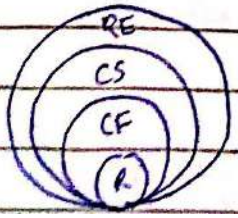
- The string is called stringvalue or Lexeme of token.
- Any value associated to a token is called attributes of a token.
- Token can be viewed as collection of all its attributes

→ Scanner treats everything as characters and reads one by one.

## Chomsky Hierarchy

(Languages ↓)

| | | |
|---|---|---|
| Recursively Enumerable | - type 0 | (Unrestricted grammar) Turing machine |
| Context-Sensitive | - type 1 | (Context-Sensitive grammar) Linear-bounded automaton |
| Context-free | - type 2 | (Context-free grammar) Pushdown automaton |
| Regular | - type 3 | (Regular language) Finite state automaton |

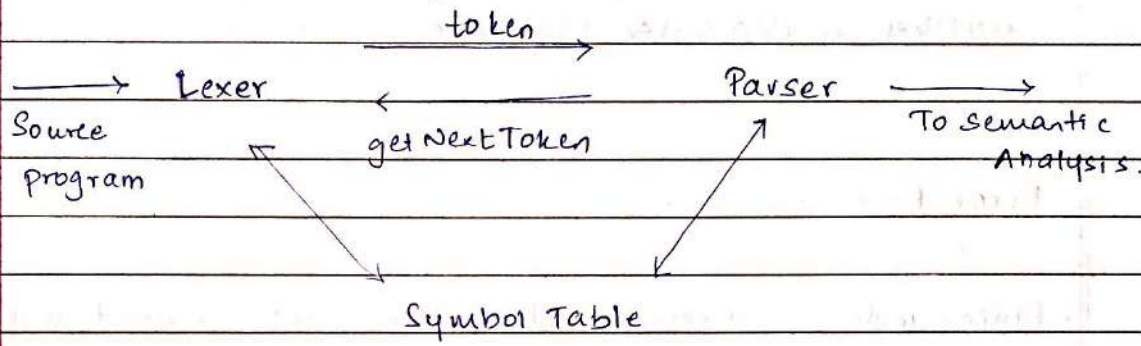


## Regular Expression.

→ represents patterns of strings of characters.
→ A regular expression r
- completely defined by the set of strings it matches
- The set is called language of r written as L(r).

14-08-19

## Role of Lexical Analyser.




## Functions of lexical analyzer.

- Primary fun$^n$ - Recognize the tokens.
- Secondary fun$^n$.
  - Eliminates comments and whitespaces (newline, tab, blank)
  - Correlating error message generated by compiler with source program
  - Keep track of newline characters, so that it can associate line number with each error message.

| | Token | Informal description | Sample lexemes. |
|---|---|---|---|
| ① | id | p, r, i, n, t, f | printf |

printf("total = %d\n", score)

| | | | |
|---|---|---|---|
| ② | else | e, l, s, e | else. |
| ③ | comparison | <, >, <=, >= etc | <, >, <= etc |

Lexical Errors.

- Some errors cannot be recognized by lexer.
  Ex: fi(a == f(x))

- Recognizes errors like d = 2r.
  (not a valid identifier)

→ Such errors are recognised when no pattern for tokens matches a character sequence

Error Recovery.

- Panic mode - successive characters are ignored until we reach to a well formed token
- Delete one character from remaining input
- Insert a missing character into the remaining input
- Replace a character by another character.
- Transpose two adjacent characters.

- Regular Expression.

~~set~~ set of valid strings is language.

→ may contain special characters called meta-characters or meta-symbols.

→ An escape character can be used to turn off (over-write) the special meaning of meta-character. Ex: *
  → backslash and quotes Ex: * or "*"

- Single characters itself are written as Regular exp as follows

→ a matches the character a by writing L(a) = {a}

→ ε denotes empty string by L(ε) = {ε}

→ {} or ϕ matches no string at all by L(ϕ) = {}

Regullau Expression operations.

- Choice among alternatives, indicated by meta-character |
- Concatenation, indicated by juxtaposition.
- ~~Repition~~ Repetition or "closure" indicated by *

Ex: ① a* (zero or more occurance)

$L(a*) = \{\epsilon, a, aa \ldots\}$

② a|b

$\therefore L(a|b) = \{a, b\}$

③ $L(ab) = \{ab\}$ (concatenation)

1. Choice among Alternatives.

→ language of r|s is union of language of r and language of s i.e, $L(r|s) = L(r) \cup L(s)$.

2. Concatenation

→ Concatenation of set of string S1S2 is set of strings of S1 appended by set of strings of S2.

→ (a|b)c matches ac and bc

Precedence of Operation

Standaud Convention → 1. Repetition *
2. Concatenation
3. Alternatives |

Ex: ① $a|bc^*$ is interpreted as $a|(b(c^*))$

• Parantheses is used to indicate different precedence

$L(a|bc^*) = \{ a, b, bc, bcc \ldots \}$

r (Regular Exp)

Ex ② $L((a|b)c^*) = \{ a, b, ac, bc, acc, bcc \ldots \}$

r

• NOTE:

decimals (alphabets 0-9) Ex: 999, 01, 02 ... etc

hexadecimal (0-9 and A to F) Ex: 0x9F, 0x3ed etc

octal (0-7) Ex: 01, 0671 etc

Name for Regular Expression.

- digit = $0|1|2|\ldots|9$
- $(0|1|2|\ldots|9)(0|1|2|\ldots|9)^*$

⇒ digit digit* (accepts n no. of digits)

Examples -

1. $\Sigma = \{a, b, c\}$

$(a|c)^* b (a|c)^*$ ⇒ r

$L(r) = \{ b, ab, cb, ba, bc, aba, abc, cbc, abaa, ccbaa \ldots \}$

→ set of all strings over this alphabet that contain exactly one b.

2. $\Sigma = \{a, b, c\}$

$(a|c)^* | (a|c)^* b (a|c)^*$ or $(a|c)^* (b|\varepsilon)^* (a|c)^*$

$L(r) = \{ a, aba, c, aaba, cabc, ab, cb \ldots\ldots \}$

→ Strings in Set contains atmost one b.

16-08-19 Tutorial

hw Implement in C

- Search for a char inside a string, if found display the pos.
- (i) Search a given substring in a string. Display suitable messages (start and end indexes). (ii) Find all occurances of substrings

cw Write output of each phases of compiler for following statements.

1. area = length * breadth
2. result = (4+6) * d

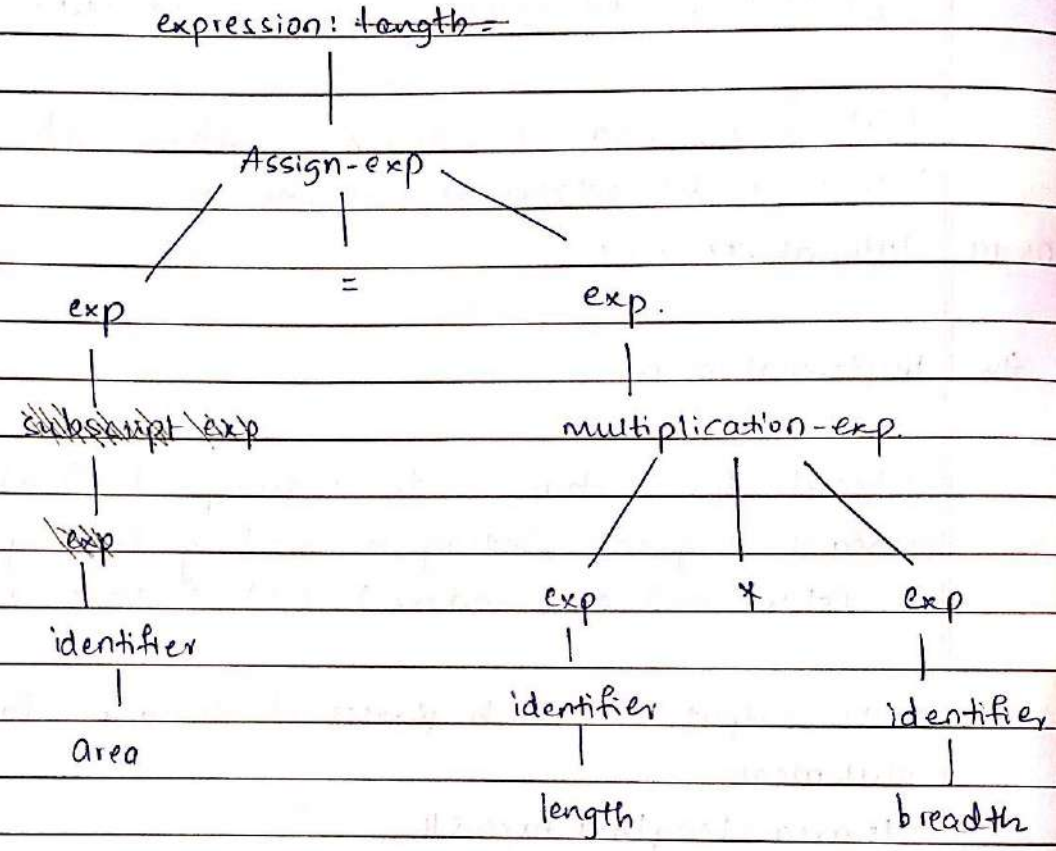
1. area = length * breadth

Sconner.

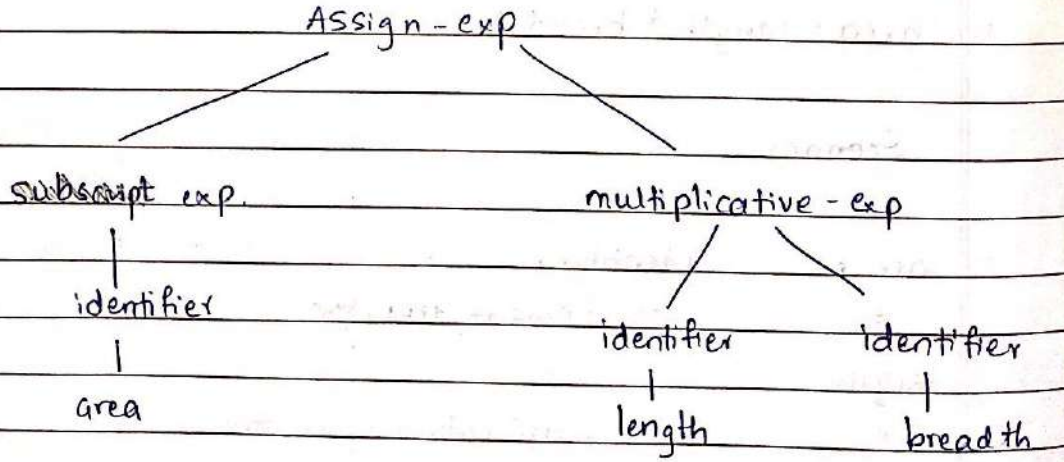
| | |
|---|---|
| area | identifier. |
| = | assignment operator |
| length | identifier |
| * | multiplication operator |
| breadth | identifier. |

Parser

Parse Tree

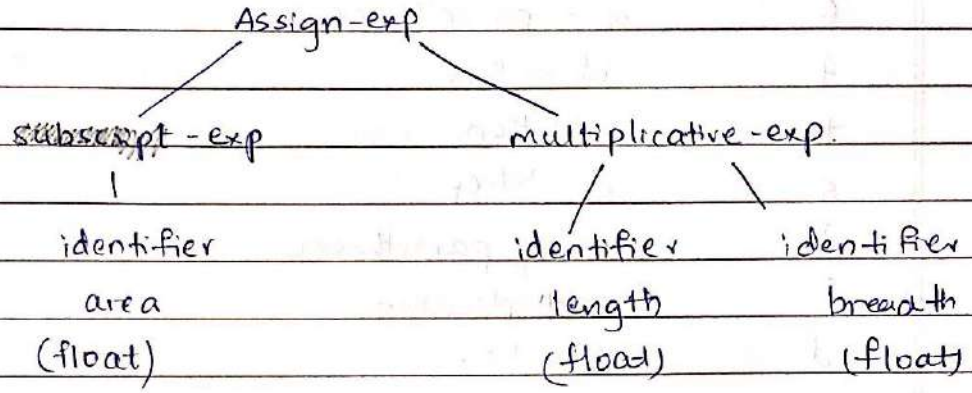



Syntax Tree



Semantic Analyser.

## Annotated Syntax tree




## Optimization on Intermediate code

t = length * breadth
area = t

area = length * breadth

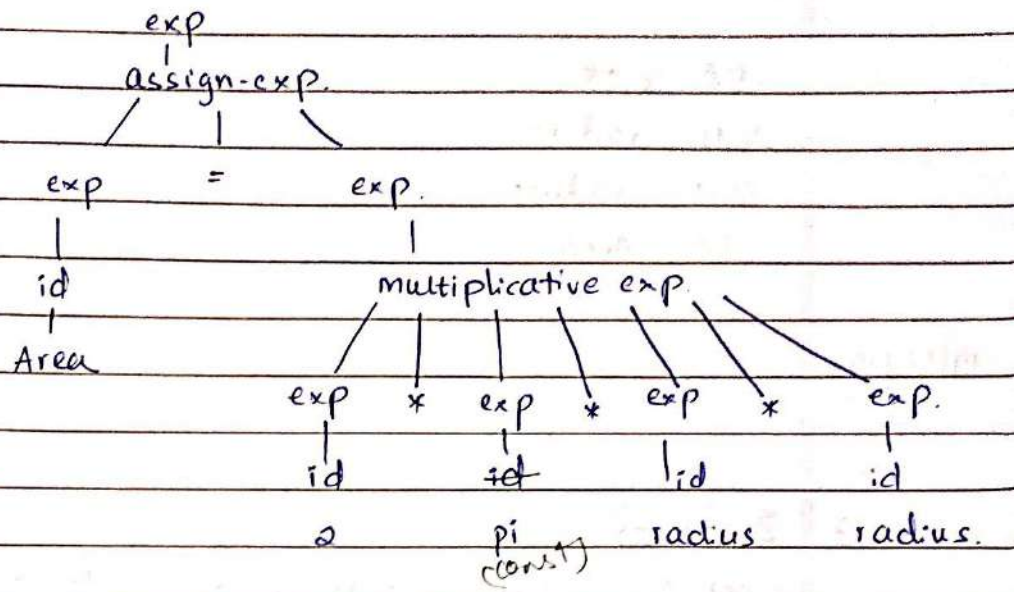
## Possible Assembly code

```
MOV  R0, length        LDA load into accu.
MUL  R0, breadth
MOV  A, R0             A is accumulator Resistor
STA  area              Store into accumulator (STA)
```

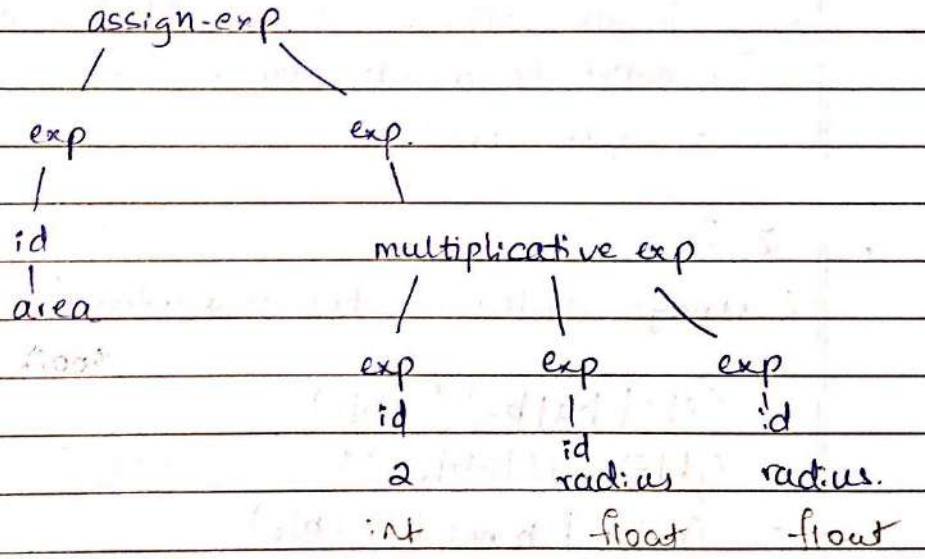
3. Area = 2 * pi * radius * radius

| | |
|---|---|
| Area | Identifier |
| = | assignment |
| 2 | identifier |
| * | multiplication |
| pi | identifier |
| * | multi |
| radius | identifier |
| * | multi |
| radius | identifier. |

Parse Tree.




Syntax Tree




Optimization

$t_1 = 2 * pi$
$t_2 = t_1 * radius$
$t_3 = t_2 * radius.$

$\Rightarrow$

$t_1 = 2 * 3.14$
$t_2 = 6.28 * radius$
$t_3 = t_2 * radius$

$result = t_3$

$result = t_3$

$0x(dig|ab)^*$
$0x(dig|alb)(dig)^*$

Possible Assembly code.

```
LDA   6.28
MUL   radius
MUL   radius.
STA   Area
```

19/08/19.
class.

Ex:

3. $\Sigma = \{a, b\}$
   - set of strings consists of single b surrounded by same no. of a's.
   - $S = \{b, aba, aabaa, \ldots\} = \{a^n b a^n \mid n > 0\}$
   - Set cannot be decribed by a regular exp because regular exp cannot count.

4. $\Sigma = \{a, b, c\}$
   - strings contain no two consecutive b's
   - $(a|c|ba|bc)^*(b|\varepsilon)$
   - $(b|\varepsilon)(a|c|ab|cb)^*$
   - $(\text{not } b \mid b \text{ not } b)^*(b|\varepsilon)$

5. $\Sigma = \{a, b, c\}$
   - $(b|c)^* a (b|c)^* a)^* (b|c)^*$
   - strings contain even no. of a's

   $L(r) = \{\epsilon, b, c, aa, baa, caa, aca, aba \ldots\}$

Q. Why not 00 is written in octal
Q. ~~00~~ signed octal and signed hex (HW)
Q. Limitations of Reg Exp (HW)

Write the regular expressions

1. unsigned int.
   Ex: 0, 1, 111, 99, 456 ...

   digit ← 0|1|2|3|4....|9

   unsignedint ← digit digit*

2. signed int. [0-7]
   Ex: -15, 0, 1, 2, +99, -456, -80, +444

   digit ← 0|1|2|3...|9 0
   sign ← +|-|ϵ

   signedint ← sign digit digit*

3. unsigned octal
   Ex: ~~00~~, 0777 ~~etc~~, 0, 0762, 01, 046 ... etc.

   digit ← 0|1|2...|7
   ~~unsignoctal ← 0digit digit*~~ 0digit*

4. unsigned hex.
   Ex: 0xAB6, 0x762, 0xCE55 etc.

   dig ← 0|1|2|....|7|8|9
   alp ← (A|B|C|D|...|F)|(a|b|c|...|f)
   begin ← 0(x|x)

   unsignhex ← begin (dig|alp)(dig|alp)*

20-08-19

## Extended Operators.

1. r+ 1 or more repitions
   r* 0 or more repitions.

2. . matches any character except \n (0 or more occurance)

3. [ ] Class of characters Ex: [aeiou] vowels.
   [a-zA-Z]

4. ~(a|b|c) any character not in given set
   Ex: ~(a|e|i|o|u) ⇒ consonants.
   or [^aeiou]

5. Optional Sub expression
   r? 0 or 1 occurance
   Ex: (\+|\-)? L(r) = {+, -, ∈}

## Examples

1. unsigned int
   dig ← [0-9]
   unsignint ← dig+

2. signed int
   dig ← [0-9]
   sign ← (\+|\-)?
   signint ← sigdig+

3. unsigned octal
   dig ← [0-9]
   octal ← 0dig+

4. unsigned hexadecimal
   dig ← [0-9A-Za-z]
   beg ← 0(x|x)
   unsignhex ← begdig+

5. Whitespaces.

ws ← [ \t\n]

exp ← ws+

6. Single line comment

s comm ← "//".*

7. Multiline comment

mcomm ← "/*" ~\n "*/"

8. Regular Expression for the following

1. PAN Card

Ex: ESPPM6789M

XXX Y Z

field1

field1 ← [A-Z]

field2 ← A|B|C|F|G|H|L|J|T|K|P

field3 ← [0001-9999] [0-9]

pan ← field1{3}field2 field1 field3 field1{4}

2. USN.

Ex: 01FE18BCS237

year ← [00-99] [0-9]{2}

dept ← [A-Z]{2}

no. ← [0-9]{3}

usn ← 01FEyearBdeptno.

21-06-19

## Lexical Regular expressions.

| | expression | Matches | Example |
|---|---|---|---|
| 1. | c | one non-operator char c | a if c is a |
| 2. | \c | character c (literal) | *, \+. |
| 3. | "s" | string s literally | "+", "-", "**". |
| 4. | . | matches any char except newline | a.b L(r) = {a$b} (one of the valid string) a.*b ⇒ {ab, a2b, a$3b ...etc} |
| 5. | ^ | takes it as beginning of a line | ^abc (string begins with abc) |
| 6. | $ | end of a line | abc$ (string ends with abc) |
| 7. | [s] | any one of characters in string s | [0-9], [abc] either a, b, c |
| 8. | [^s] | any one character NOT in string s. | [^aeiou] (consonants) |
| 9. | r* | 0 or more strings matching r | a* {ϵ, a, aa, aaa...} |
| 10. | r+ | 1 or more strings matching r | a+ {a, aa, aaa...} |

| | | | |
|---|---|---|---|
| 11. | r? | zero or one occurance of r | (\+\|\-)? ⇒{+,-,ϵ} a? {ϵ,a} |
| 12. | r{m,n} | between m and n occurances of r | a{1,4} ⇒{a,aa,aaa, aaaa} |
| 13. | r1r2 | concatenation (r1 followed by r2) | ab ⇒ {ab} |
| 14. | r1\|r2 | either r1 or r2 | +\|- ⇒ {+,-} |
| 15. | (r) | same as r | (a) ⇒ {a} |
| 16. | r1/r2 | r1 when followed by r2 | abc/123 |

Q. Write Regular expressions for following

1. Unsigned Real No.
2. Signed Real No in decimal notation
3. Signed Real No in exponent form.

① Unsigned Real no.

Ex: 2.34, 0.03, 69.69, 420.420, ....

dig ← [0-9]

unsignReal ← ~~dig\.~~ dig+ \. dig+

Q. PAN card reg exp.

2. Signed Real no. in decimal notation

Ex: 0.03, +8.74, -76.35 ....

dig ← [0-9]

sign ← (\+|\-)?

signedReal ← sign dig+ \. dig+

3. Signed Real no. in exponent form

(+|-) (E|e) (+|-)

mantissa (real no.) — exp (int)

Ex: 2.38 e-15 , 3.75 E7 ...

dig ← [0-9]

sign ← (\+|\-)?

signexp ← sign dig+ \. dig+ (E|e) sign dig+

~~Q~~ Aadhar no.

Ex: 5895 6189 7135.

no. ← [0-9]

aadhar ← no.{12}

23-08-19 Tutorial.

Q Place the following tokens (keys) in the symbol table (Hash table) using hash function

Hf(key) = sumof(ci) % 13 where ci is ordering of each char in string

1) num1, num2, avg, mean, median.

| sumof(ci) \ ci | num1 | num2 | avg | mean | median |
|---|---|---|---|---|---|
| hf(key) | 50%13 = 11 | 51%13 = 12 | 30%13 = 4 | 33%13 = 7 | 46%13 = 7 |

| | | |
|---|---|---|
| H[0] | | |
| H[1] | | |
| H[4] | avg | |
| H[7] | | mean → median → NULL |
| H[11] | num1 | |
| H[12] | num2 | |

2) 1234, 5678, 78936, 78923, 298790 using $hf_2(key) = key \% 17$.

| ci | 1234 | 5678 | 78936 | 298790 | 78923 |
|---|---|---|---|---|---|
| hf2(key) | ~~14%17~~ ~~=0~~ 10 | ~~30%17~~ ~~=13~~ 0 | ~~38%17~~ ~~=4~~ 5 | ~~41%17~~ ~~=7~~ ~~7~~ 15 | ~~34%17~~ ~~=0~~ ~~15~~ 9 |

Q. where ci is the ASCII numbering (a ⇒ 97 0 ⇒ 48)

1)

| | num1 | num2 | avg | mean | median |
|---|---|---|---|---|---|
| hf(key) | 385 % 13 | 386 % 13 | 318 % 13 | 417 % 13 | 622 % 13 |
| | 8 | 9 | 6 | 1 | 11 |

Q. Write regular expressions for

?? 1. identifiers

Ex: x , _x , _123 , num1 , num_2 etc

alp ← [A-Z a-z] dig ← [0-9]

~~iden ← (_|ϵ) alp+~~ iden ← alp* (_|ϵ) (alp+dig+) (_|ϵ)

2. keywords ( branching and looping statements)

key ← while | do | if | for | elseif | switch | break | exit | continue

1. identifiers

letter ← [A-Z a-z _]

dig ← [0-9]

iden ← (letter | dig)*

2. Relational operators

rel ← <= | >= | > | < | == | !=

4. USN (undergrad)

format → 01F E dd B CS ddd

field1 f2 f3 f4 f5 f6

dig ← [0-9] branch ← CS|EC|EE|BT|ME|CV|AR|AT

usn ← 01F(E|A)dig2Bbranch num

dig2 ← 1(5|6|7|8|9)|[2-9]dig

num ← 00[1-9] | 0[1-9][0-9] | [1-9][0-9][0-9]

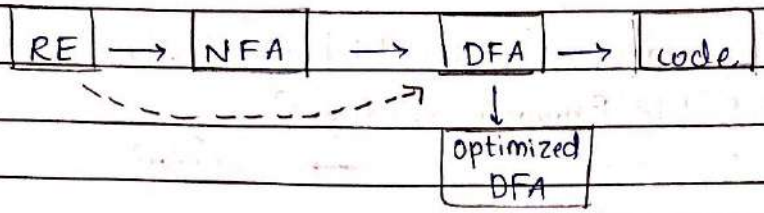
a 97
b 98
c 99
d 100
e 101
f 102
g 103
h 104
i 105
j 106
k 107
l 109
m 109
n 110
o 111
p 112
q 113
r 114
s 115
t 116
u 117
v 118

5. PAN card

26-08-19

Lexical Analyser.

→ Uses Regular exp to arrive at code.




NFA - non deterministic Finite Automata

DFA - Deterministic Finite Automata

Fig. Scanning Process.

~~NFA~~ RE → NFA : Thompson Alg , NFA → DFA : Subset Alg

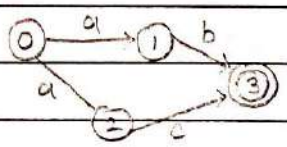
DFA → opt DFA : Mark alg

NFA (non-deterministic Finite Automata) (Plural)

→ Mathematical way of describing an algorithm which recognises if an input is valid or not (accepts or rejects) is called Automaton (vague description)

Page-self

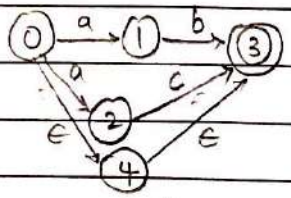
→ set of stages ~~de~~ recognizing character from input string.




for L(ab|ac) = {ab, ac}

It is not determined if after stage-0, which stage will follow.

→ Non-deterministic can have ∈ transitions



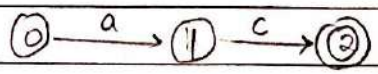

δ(0,a) = {1, 2}

δ(0, ∈) = {4}

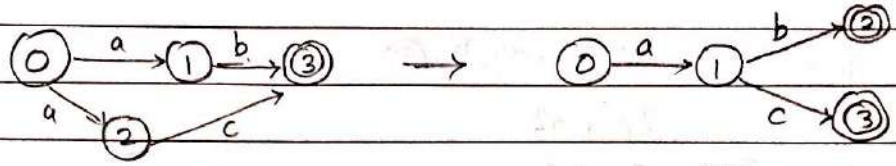
→ NFA can have more than one transitions from same input string.

→ DFA will have determined stages and same input string will have one transition.

Ex ①




Ex: ②




NFA DFA

## Finite Automata.

Finite Automata are recognisers which simply say Yes or No about each possible input string. Finite Automata are the mathematical representations that describe particular kind of algorithm.

Finite Automata comes in two stages ① NFA ② DFA

① NFA has no restrictions on the labels of their edges. A symbol can label several stages out of same state and $\epsilon$ (empty string) which can be a possible label.

② DFA have for each state and for each symbol of its input alphabet exactly one edge with that symbol leaving that string.

### Formal Def$^n$

→ An NFA consists of ① Finite set of stages, $S$.

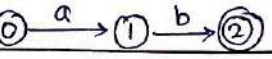
② Set of input symbols $\Sigma$, the input alphabet. We Assume that $\epsilon$ which stands for empty string is never a member of alphabet $\Sigma$.

③ A transition function that gives for each state and for each symbol in $\Sigma \cup \{\epsilon\}$ as set of next states.

④ A state $s_0$ from $S$ that is designated as start state

or initial state

⑤ A set of states F which is subset of S that is designated as accepting states or final state.

Ex: 


① $S = \{0, 1, 2\}$

② $\Sigma = \{a, b\}$

③ $\delta(0, a) = 1$

④ $s_0 = 0 \in S$

⑤ $F = \{2\}$ $F \subseteq S$

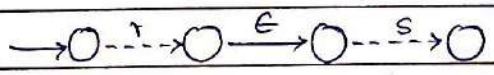
also Ex for DFA

→ A DFA is a special case of NFA where

① There are no moves on input symbols $\epsilon$ (no $\epsilon$ transitions)

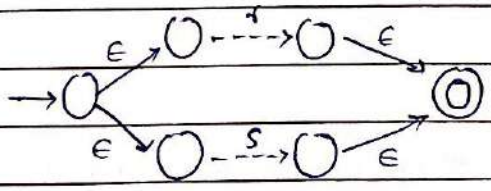
② For each state 's' an input symbol 'a' there is exactly one edge out of 's' labelled 'a'

1) Regular exp rs (r concatenated with s)

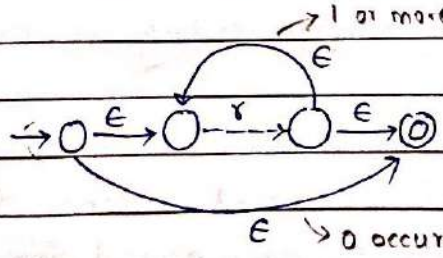



Genral NFA

2) r|s



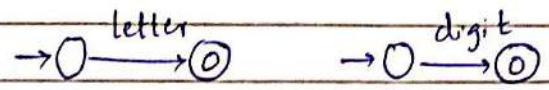

3) r*

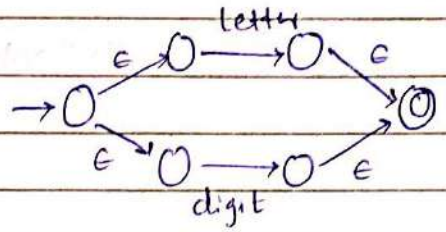


Construct NFA for following regular exp using Thompson's method
(RE → NFA)

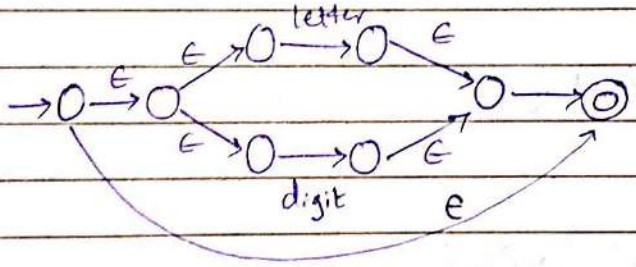
1. identifiers.

iden ← letter (letter | digit)*

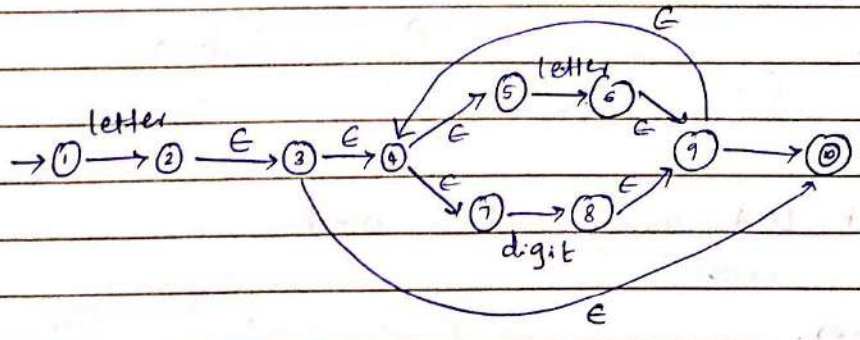


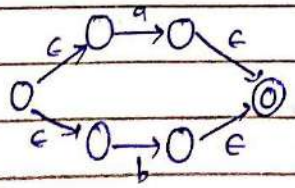

letter | digit



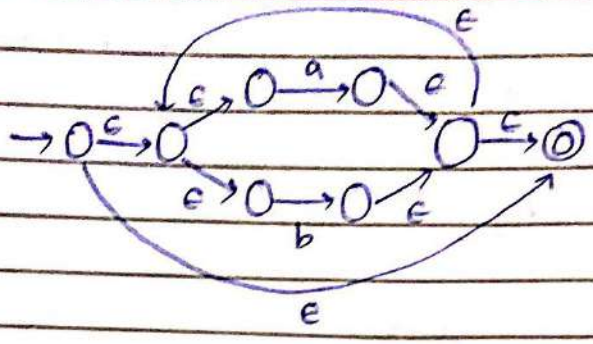

(letter | digit)*

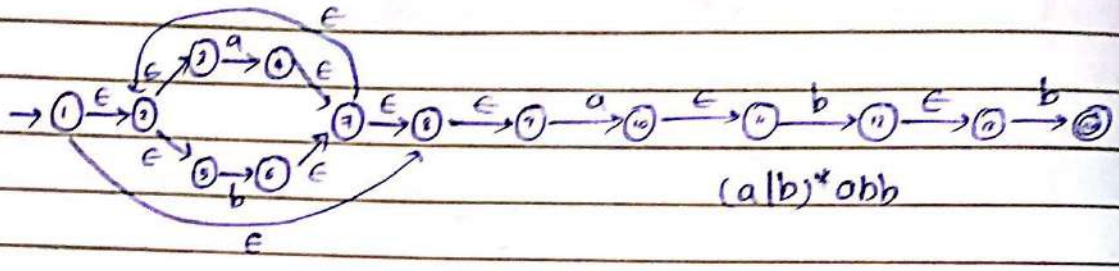



2. (a|b)* abb

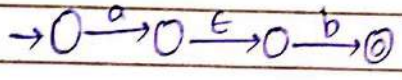



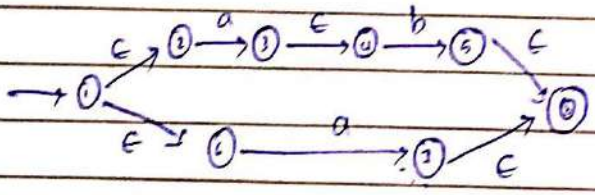

a | b



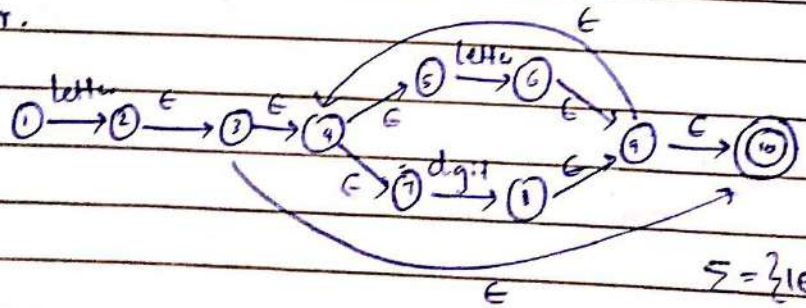

3. ab|a




Construct DFA using subset method

1. identifier.




start state $\overline{\{1\}} = \{1\} = A$

closure

$\overline{A_{letter}} = \{\overline{2}\}$

$= \{2, 3, 4, 5, 7, 10\}$

$= B.$

$\overline{A_{digit}} = \phi$

$\overline{B_{letter}} = \{\overline{6}\}$

$= \{6, 9, 10, 4, 5, 7\}$

$= \{4, 5, 6, 7, 9, 10\}$

$= C$

$\overline{B_{digit}} = \{\overline{8}\}$

$= \{8, 9, 10, 4, 5, 7\}$

$= \{4, 5, 7, 8, 9, 10\}$

$= D$

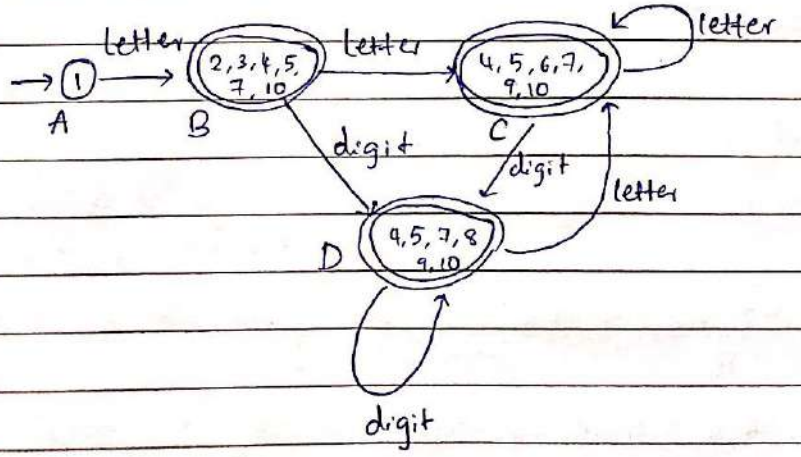
$\overline{C_{letter}} = \{\overline{6}\} = C$ (C on letter is going to itself)

$\overline{C_{digit}} = \{\overline{8}\} = D$ (C on digit is going to D)

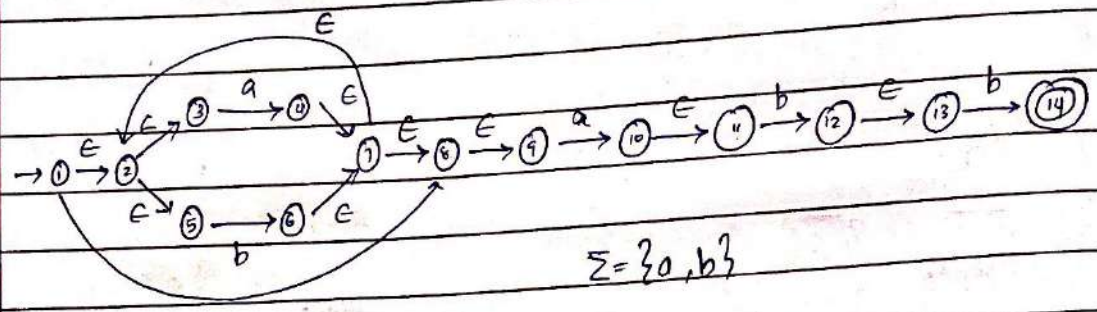
i.e, no new subsets are formed. so stop.

$\overline{D_{letter}} = \{\overline{6}\} = C$

$\overline{D_{digit}} = \{\overline{8}\} = D$



2. $(a|b)^* abb$




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

$\overline{\{1\}} = \{1, 2, 3, 5, 8, 9\} = A$

$\overline{A_a} = \overline{\{4, 10\}} = \bar{4} \cup \bar{10}$

$= \{4, 7, 2, 3, 5, 8, 9\} \cup \{10, 11\}$

$= \{2, 3, 4, 5, 7, 8, 9, 10, 11\}$

$= B$

$\overline{A_b} = \{\overline{6}\}$

$= \{6, 7, 2, 3, 5, 8, 9\}$

$= \{2, 3, 5, 6, 7, 8, 9\}$

$= C$

$\overline{B_a} = \{\overline{4, 10}\} = \bar{4} \cup \bar{10} = B$

$\overline{B_b} = \{\overline{6, 12}\} = \bar{6} \cup \bar{12}$

$= \bar{6} \cup \{12, 13\}$

$= \{2, 3, 5, 6, 7, 8, 9, 12, 13\}$

$= D$

$\overline{C_a} = \{\overline{4, 10}\} = B$ $\quad \overline{C_b} = \bar{6} = C$

$\overline{D_a} = \{\overline{4, 10}\} = B$ $\quad \overline{D_b} = \{6, 14\} = \bar{6} \cup \bar{14}$

$= C \cup \{14\}$

$= \{2, 3, 5, 6, 7, 8, 9, 14\} = E$

$\overline{E_a} = \{4, 10, 14\}$

$= \{\overline{4, 10}\} \cup \bar{14}$

$= \{2, 3, 4, 5, 7, 8, 9, 10, 11, 14\} = F$

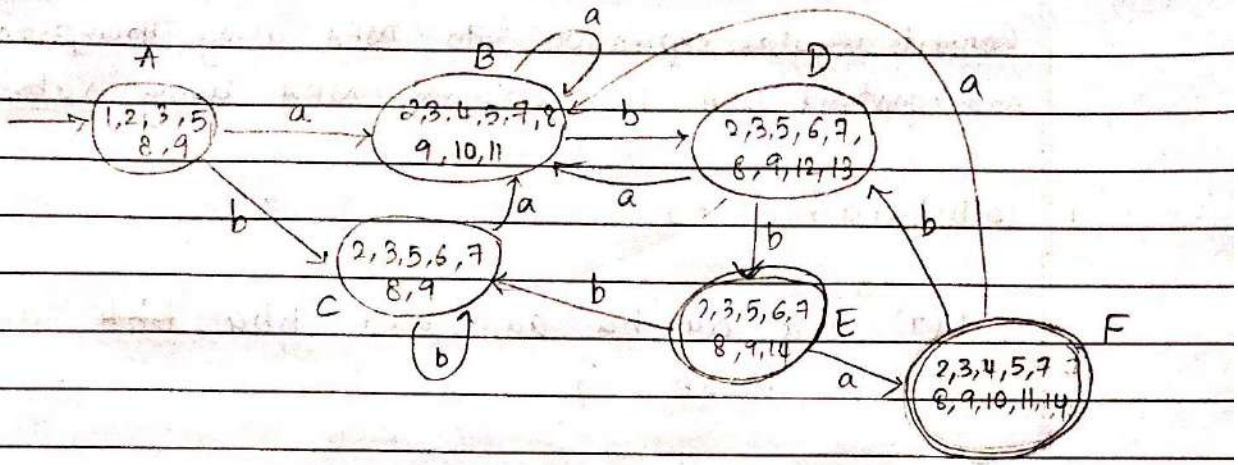
$\overline{E_b} = \{\overline{6}\} = C$

$\overline{F_a} = \{\overline{4, 10}\} = B$

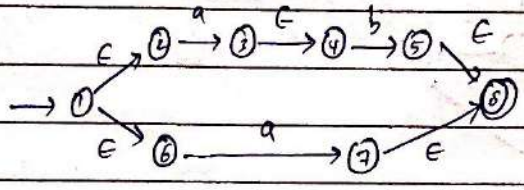
$\overline{F_b} = \{\overline{6, 12, 14}\}$

$= \{2, 3, 5, 6, 7, 8, 9, 12, 13, 14\}$

$= D$




5. ab|a = r




$L(r) = \{a, ab\}$

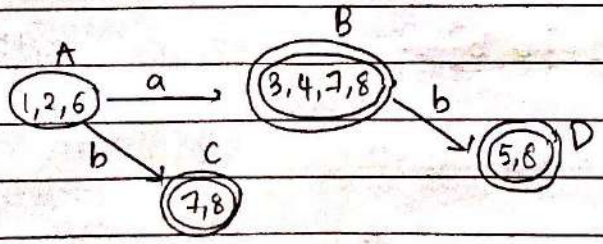
$T = \{1,2,6\} = A$ $\quad \Sigma = \{a, b\}$

$\overline{A_a} = \{\overline{3,7}\} = \{3,4,7,8\} = B$

$\overline{A_b} = \{\overline{7}\} = \{7,8\} = C$

$\overline{B_a} = \phi \quad \overline{B_b} = \{\overline{5}\} = \{5,8\} = D$

$\overline{C_a} = \phi \quad \overline{C_b} = \phi \quad \overline{D_a} = \phi \quad \overline{D_b} = \phi$



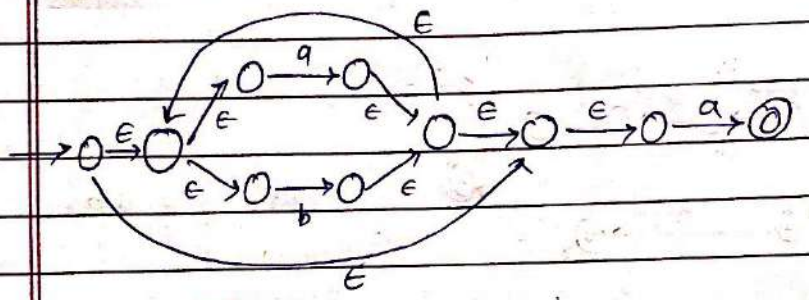

⇒ Unoptimized

30-08-19 Tutorial

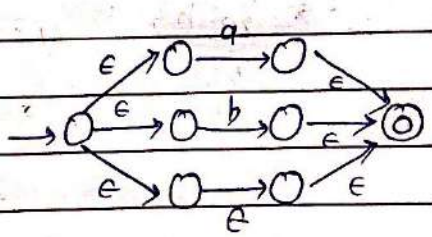
Convert regular expression into NFA using Thompson's method and construct DFA from obtained NFA using Subset const.

1. $(a|b)^*a(a|b|\epsilon) = r$

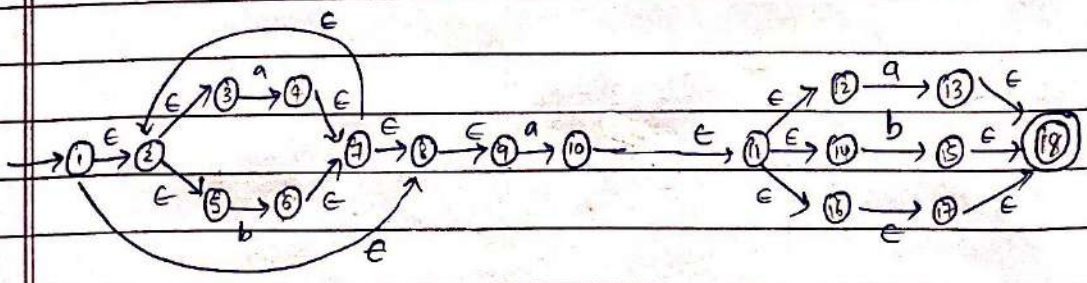
$L(r) = \{ a, aa, ba, aaa, aab, bba, \cancel{bbb}, aaab, bbab \ldots \}$




$(a|b)^*a$




$a|b|\epsilon$




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

$(a|b)^*a(a|b|\epsilon)$

$\bar{I} = \{1, 2, 3, 5, 8, 9\} = A$

$\bar{A}_a = \{\overline{4, 10}\} = \{2, 3, 4, 5, 7, 8, 9, 11, 12, 14, 16, 17, 18\} = B$

$\bar{A}_b = \{\overline{6}\} = \{2, 3, 5, 6, 7, 8, 9\} = C$

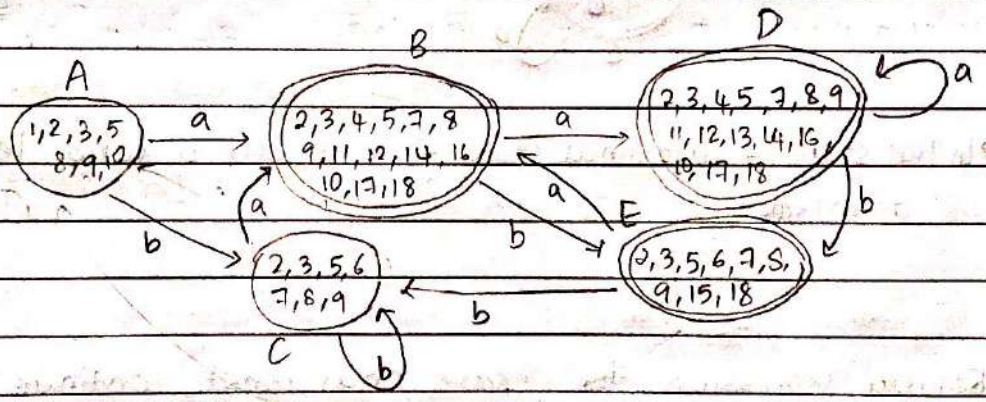
$\bar{E}_a = \{a, \overline{10, 13}\} = \{2,3,4,5,7,8,9,11,12,13,14,16,17,18\} = D$

$\bar{B}_b = \{\overline{6, 15}\} = \{2,3,5,6,7,8,9,15,18\} = E$

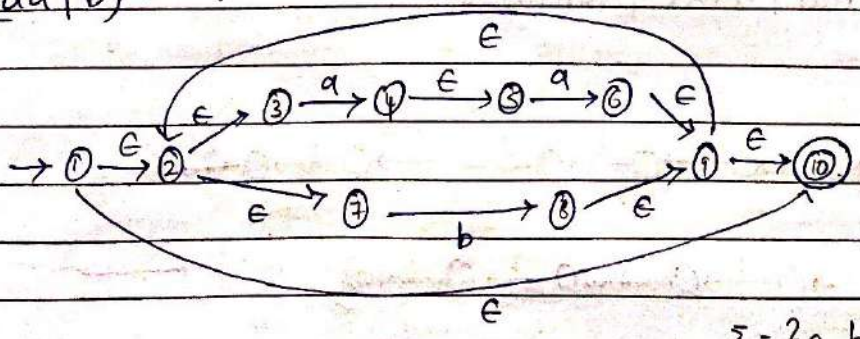
$\bar{C}_a = \{\overline{4, 10}\} = B$ $\quad$ $\bar{C}_b = \bar{6} = C$

$\bar{D}_a = \{4, \overline{10, 13}\} = D$ $\quad$ $\bar{D}_b = \{\overline{6, 15}\} = E$

$\bar{E}_a = \{\overline{4, 10}\} = B$ $\quad$ $\bar{E}_b = \{\overline{6}\} = C$




2. $(aa|b)^* = r$ $\quad L(r) = \{\epsilon, aa, b, aaaa, bb, aaaaaa, bbb \ldots, aab, aaaab\}$




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

$\bar{1} = \{1,2,3,7,10\} = A$

$\bar{A}_a = \{\bar{4}\} = \{4,5\} = B$ $\quad$ $\bar{A}_b = \{\bar{8}\} = \{8,9,10,2,3,7\} = C$

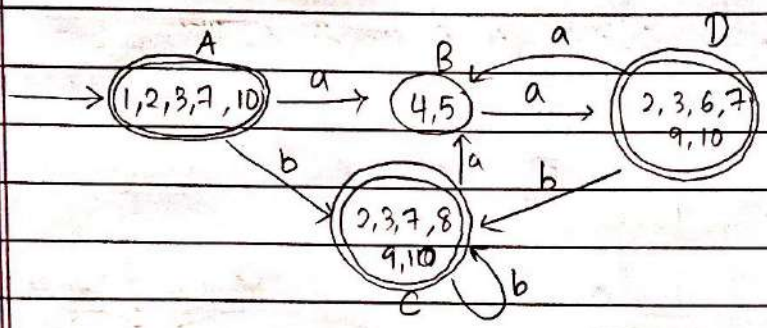
$\bar{B}_a = \{\bar{6}\} = \{6,9,10,2,3,7\} = \{2,3,6,7,9,10\} = D$

$B_b = \phi$

$\bar{C}_a = \{\bar{4}\} = B$

$\bar{C}_b = \{\bar{8}\} = C$

$\bar{D}_a = \{\bar{4}\} = B$ $\quad$ $\bar{D}_b = \{\bar{8}\} = C$



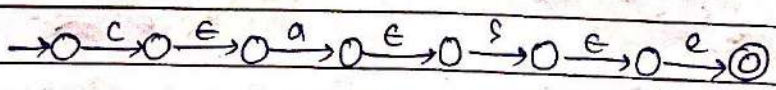

→ Initial state is also final state ⇒ ε is also a valid lexeme, It is accepted

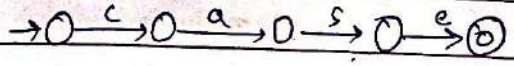
03-09-19

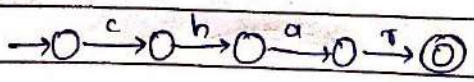
Q Regular expression for case, char, const, continue

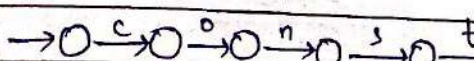
case | char | const | continue

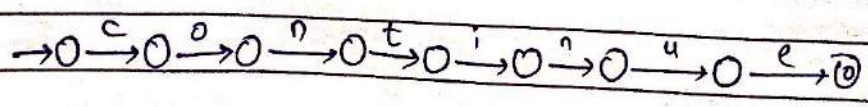
NFA →



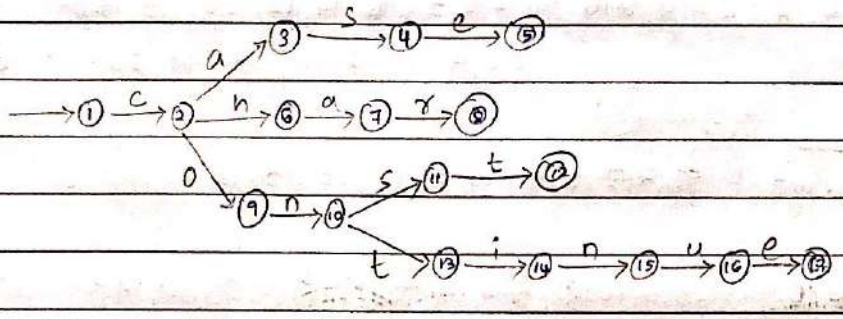

DFA → 

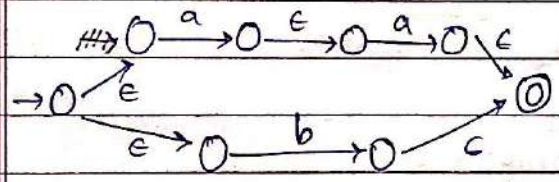



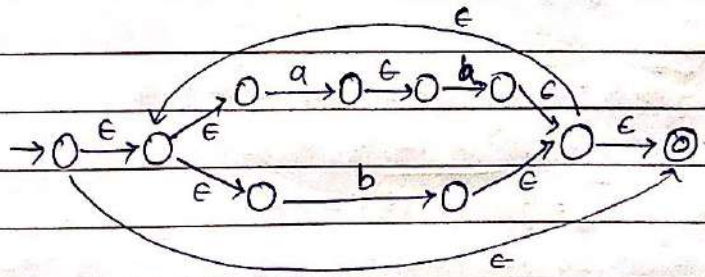
Combined DFA :



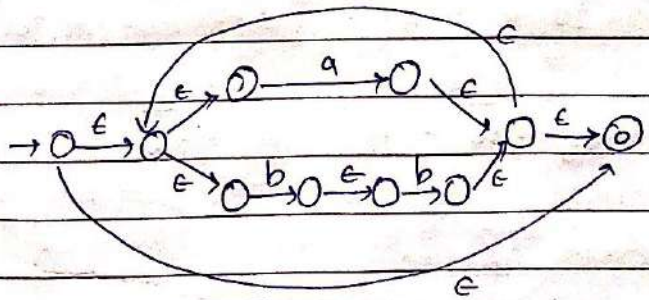
Q. $(aa|b)^*(a|bb)^*$ Construct NFA and then DFA



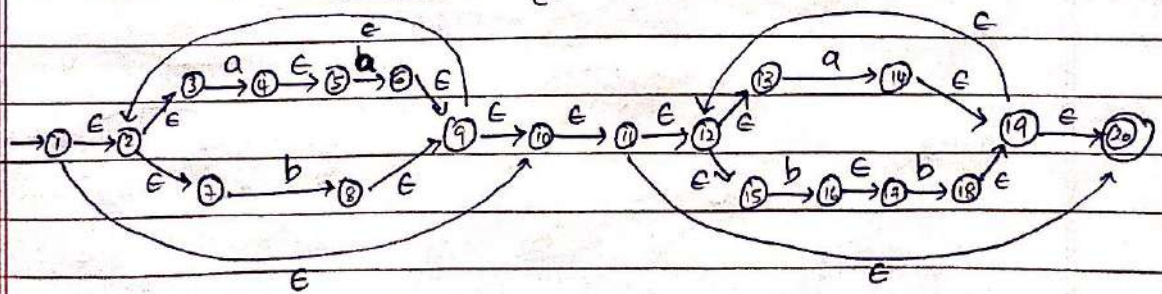
aa|b



$(aa|b)^*$



$(a|bb)^*$



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

$\overline{\{17\}} = \{1,2,3,7,10,11,12,13,15,20\} = A$

$\overline{A}_a = \{\overline{4,14}\} = \overline{4} \cup \overline{14} = \{4,5,14,19,12,20,13,15\}$

$= \{4,5,12,13,14,15,19,20\} = B$

$\overline{A}_b = \{\overline{8,16}\} = \{2,3,7,8,9,10,11,12,13,15,16,17,20\} = C$

$\overline{B}_a = \{\overline{6,14}\} = \{6,9,10,2,3,7,11,12,13,15,20,14,19\}$

$= \{2,3,6,7,9,10,11,12,13,14,15,19,20\} = D$

$\overline{B}_b = \{\overline{16}\} = \{16,17\} = E$

$\overline{C}_a = \{\overline{4,14}\} = B$ $\quad \overline{C}_b = \{\overline{8,16,18}\} = \{2,3,7,8,9,10,11,12,13,15,16,17,18,19,20\} = F$

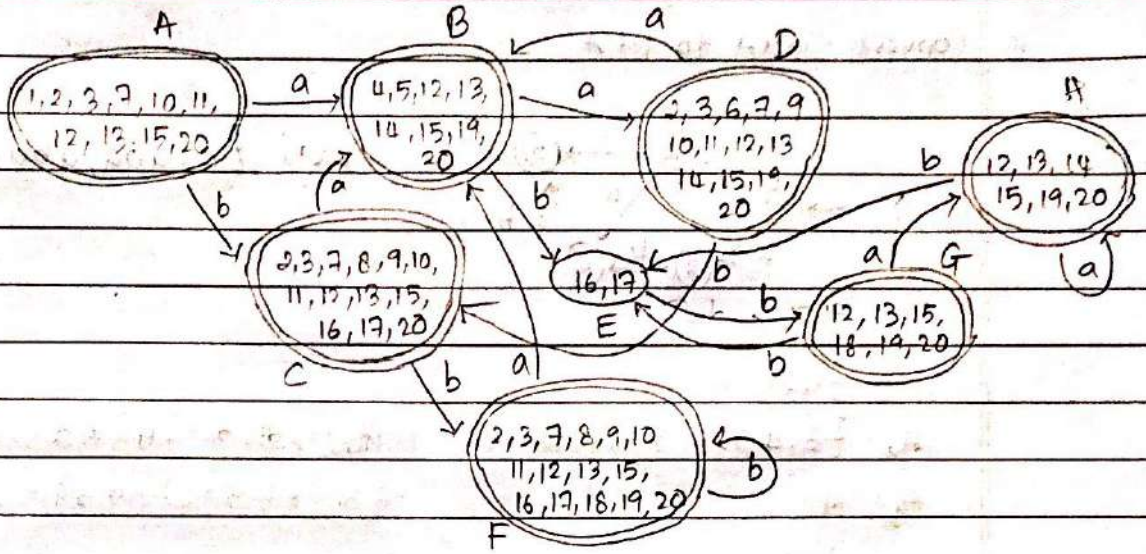
$\overline{D}_a = \{\overline{4,14}\} = B$ $\quad \overline{D}_b = \{\overline{8,16}\} = C$

$\overline{E}_a = \phi$ $\quad \overline{E}_b = \{\overline{18}\} = \{12,13,15,18,19,20\} = G$

$\overline{F}_a = \{\overline{4,14}\} = B$ $\quad \overline{F}_b = \{\overline{8,16,18}\} = F$

$\overline{G}_a = \{\overline{14}\} = \{14,19,20,12,13,15\} = \{12,13,14,15,19,20\} = H$

$\overline{G}_b = \{\overline{16}\} = E$ $\quad \overline{H}_a = \{\overline{14}\} = H$ $\quad \overline{H}_b = \{\overline{16}\} = E$

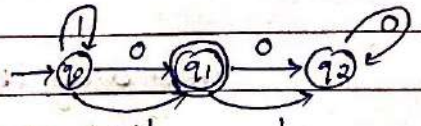



Q. State transition table is given. Convert NFA to DFA

| δ | 0 | 1 |
|---|---|---|
| → $q_0$ | $q_1$ | $\{q_0, q_1\}$ |
| $q_1$* | $q_2$ | $q_2$ |
| $q_2$ | $q_2$ | $\phi$ |

* - Final state

→ NFA:




$q_2$ - inaccesible.

$\{\overline{q_0}\} = \{q_0\}$ — A

$\overline{A_1} = \{\overline{q_0, q_1}\} = \{q_0, q_1\} = B$

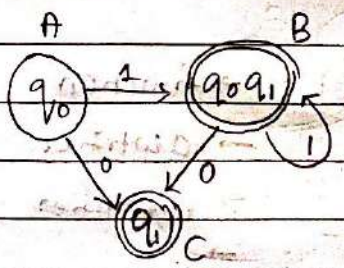
$\overline{A_0} = \{\overline{q_1}\} = \{q_1\} = C$

$\overline{B_1} = \{\overline{q_0, q_1}\} = B$

$\overline{B_0} = \{\overline{q_1}\} = C$

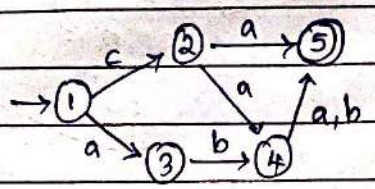
$\overline{C_1} = \phi$

$\overline{C_0} = \phi$




$L(r) = \{1110, 110, 10, 0, 1, 111, 11, \ldots\}$

Q. Convert NFA to DFA



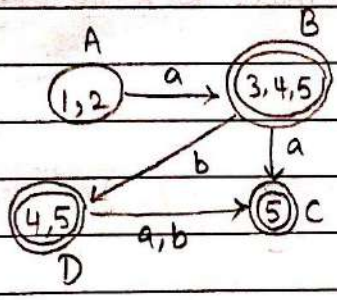

$L(r) = \{a, aba, abb, aa, ab \ldots\}$

$\{\overline{1}\} = \{1,2\} = A$

$\overline{A_a} = \{\overline{3,4,5}\} = \{3,4,5\} = B$ $\quad \overline{B_a} = \{\overline{5}\} = \{5\} = C$

$\overline{A_b} = \phi$ $\quad \overline{B_b} = \{\overline{4,5}\} = \{4,5\} = D$

$\overline{C_a} = \phi \quad \overline{C_b} = \phi \quad \overline{D_a} = \{\overline{5}\} = C \quad \overline{D_b} = \{\overline{5}\} = C$




ISA-1 QP.

1b. int main()

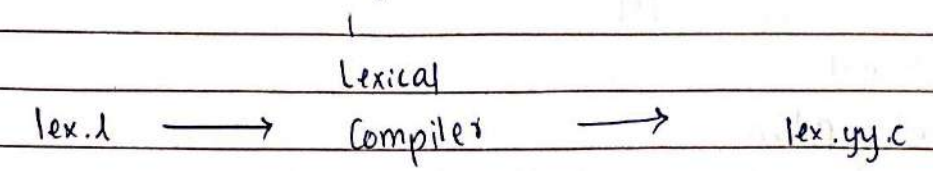
neat diag lexical analysis (Role of lexer) (14-08)

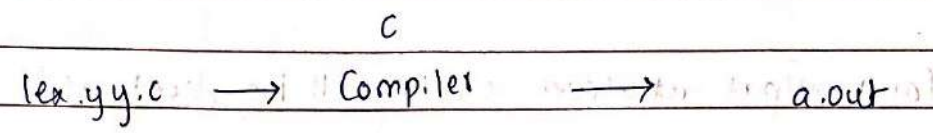
main() → function
main → identifier
int a → identifier
b → "
; → spl char
return → keyword
0 → const

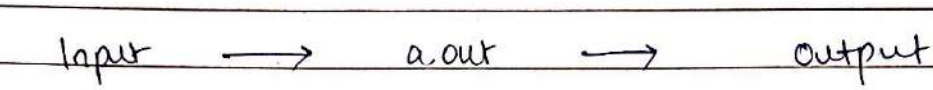
3b in ESA OP HW

06-09-19 Tutorial

# Lexical Analyser - Lex Tool




- Structure of Lex programs

declarations
%.%.
translation rules followed by → Pattern {Action}
(typically C statements)
%.%.
auxiliary functions
(main() or other functions)

- yytext is pointer points to current lexeme it has matched. (pre-defined)
- yyin is input file pointer
- yyout is output file pointer
- yylex() - pre-defined function is called so that it reads input char by char and matches with the rules (Regular exp). Reads until eof is reached

Commands -

```
$ lex vowel.l
$ cc lex.yy.c -lfl      (-lfl command used to link)
$ ./a.out
$ cat input.c
        displays content of file input.c
$ ./a.out
```

- default output - whatever is read will be displayed

Ex:

(1)

```
%{
#include<stdio.h>
int count=0;
%}

%%
[aeiouAEIOU]  {printf("%c", *yytext); count++;}
%%
main()
{
yyin = fopen("input.c", "r");
yylex();
printf("\n No. of ws %d \n", count);
}

int yywrap()
{
  return 1;
}
```

Ex: // Reads whitespaces from file

(2)

```
%{
#include<stdio.h>
%}
%%
[\n\t' '] {};
%%
main()
{
yyin = fopen("input.txt", "r");
yylex();
}
int yywrap()
{
 return 1;
}
```

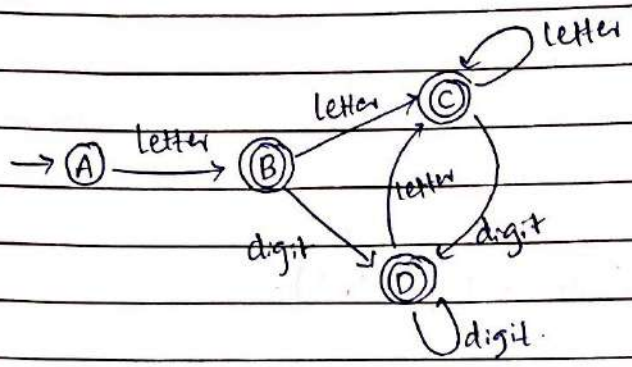
(3)

09-09-19

## Optimization of DFA (Mark algorithm).

Q identifier

RE ← letter(letter|digit)*




$\Sigma = \{letter, digit\}$

Two sets, one of distinguishable states and another of similar states are found.

1. Distinguishable states

(p,q) iff p ∈ non-accepting states
q ∈ accepting states.
or vice versa.

2. Similar states

(x,y) iff x ∈ accepting states.
y ∈ accepting states.
OR
x ∈ non-accepting states
y ∈ non-accepting states.

In DFA of identifier.

1. Distinguishable

(A,B)✓, (A,C)✓, (A,D)✓

2. Similar

(B,C), (C,D), (D,B)

we check if any similar states can be combined

(B, C)

$\delta(B, letter) = C = p'$ $\quad \delta(B, digit) = D$

$\delta(C, letter) = C = q'$ $\quad \delta(C, digit) = D$.

check if $(p', q')$ is already marked as distinguishable

(C, D)

$\delta(C, letter) = C$ $\quad \delta(C, digit) = D$

$\delta(D, letter) = C$ $\quad \delta(D, digit) = D$

(B, D)

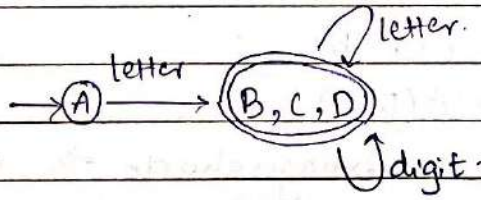
$\delta(B, letter) = C$ $\quad \delta(B, digit) = D$

$\delta(D, letter) = C$ $\quad \delta(D, digit) = D$

None of the above obtained pair can be marked as distinguishable ∴ All remain as ~~distinguishable~~ similar states.

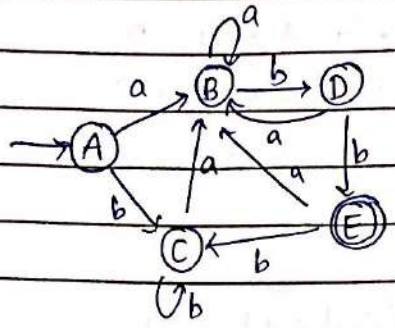
∴ They can be combined

$\Rightarrow (B, C) \cup (C, D) \cup (B, D) = (B, C, D)$




Hence DFA is optimised.

Q. $(a|b)^* abb$




1. distinguishable
(A,E), (B,E), (C,E), (D,E)

2. Similar states
(A,B), (A,C), (A,D)
(B,C), (B,D)
(C,D)

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

Consider
(A,B)

$\delta(A, a) = B$ $\quad$ $\delta(A, b) = C$
$\delta(B, a) = B$ $\quad$ $\delta(B, b) = D$

(A,C)
$\delta(A, a) = B$ $\quad$ $\delta(A, b) = C$
$\delta(C, a) = B$ $\quad$ $\delta(C, b) = C$

(A,D)
$\delta(A, a) = B$ $\quad$ $\delta(A, b) = C$
$\delta(D, a) = B$ $\quad$ $\delta(D, b) = E$

(C,E) is marked as distinguishable $\Rightarrow$ ~~A+B~~ (A,D)

(B,C)
$\delta(B, a) = B$ $\quad$ $\delta(B, b) = B$ D.
$\delta(C, a) = B$ $\quad$ $\delta(C, b) = C$

(B,D) ✓

$\delta(B,a) = B$ $\quad \delta(B,b) = D$

$\delta(D,a) = B$ $\quad \delta(D,b) = E$

(D,E) ✓ ⇒ (B,D) ✓

(C,D) ✓

$\delta(C,a) = B$ $\quad \delta(C,b) = C$

$\delta(D,a) = B$ $\quad \delta(D,b) = E$

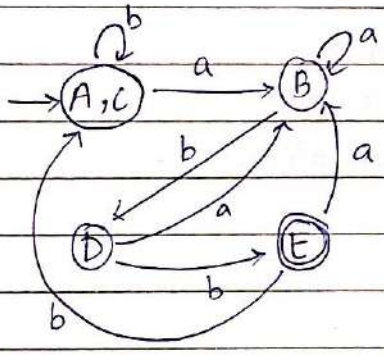
(C,E) ✓ ⇒ (C,D) ✓

In 2nd iteration, as (C,D) ✓ ⇒ (A,B) ✓

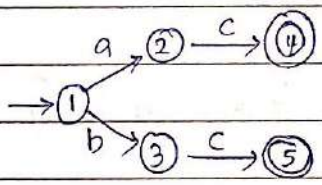
as (D,C) ✓ ⇒ (B,C) ✓

Except (A,C) all pairs are marked as distinguishable

∴ A & C are combined




3Q.




$L1(1) = \{ac, bc\}$

$\Sigma = \{a, b, c\}$

1. Distinguishable

(1,4) ✓, (2,4) ✓, (1,5) ✓, (3,5) ✓, (3,4) ✓, (2,5) ✓

2. Similar

(1,2), (2,3), (1,3), (4,5)

(1,2)

$\delta(1,a) = 2$ $\quad \delta(1,b) = 3$ $\quad \delta(1,c) = \phi$

$\delta(2,a) = \phi$ $\quad \delta(2,b) = \phi$ $\quad \delta(2,c) = 4$

(2,3)

$\delta(2,a) = \phi$ $\quad \delta(2,b) = \phi$ $\quad \delta(2,c) = 4$

$\delta(3,a) = \phi$ $\quad \delta(3,b) = \phi$ $\quad \delta(3,c) = 5$

(1,3)

$\delta(1,a) = 2$ $\quad \delta(1,b) = 3$ $\quad \delta(1,c) = \phi$

$\delta(3,a) = \phi$ $\quad \delta(3,b) = \phi$ $\quad \delta(3,c) = 5$

(4,5)

$\delta(4,a) = \phi$ $\quad \delta(4,b) = \phi$ $\quad \delta(4,c) = \phi$

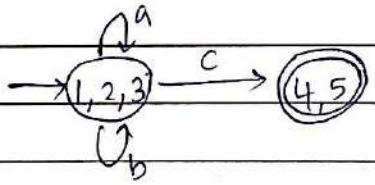
$\delta(5,a) = \phi$ $\quad \delta(5,b) = \phi$ $\quad \delta(5,c) = \phi$

none are marked as distinguishable

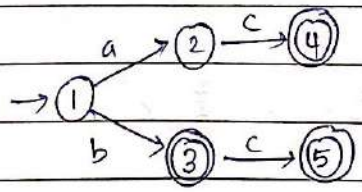
$\therefore (1,2) \cup (2,3) \cup (1,3) = \{1,2,3\}$

and $\{4,5\}$

Doubt.
min 3 states should be obtained




4Q




$L2(1) = \{ac, b, bc\}$

$\Sigma = \{a, b, c\}$

Distinguishable (1,3) ✓ (1,5) ✓ (1,4) ✓, (2,3) ✓, (2,4) ✓, (2,5) ✓

Similar (1,2) (3,4) (4,5) (5,3).

(1,2)

$\delta(1,a) = 2$ $\quad \delta(1,b) = 3$ $\quad \delta(1,c) = \phi$

$\delta(2,a) = \phi$ $\quad \delta(2,b) = \phi$ $\quad \delta(2,c) = 4$

(3,4)

$\delta(3,a) = \phi$ $\quad \delta(3,b) = \phi$ $\quad \delta(3,c) = 5$

$\delta(4,a) = \phi$ $\quad \delta(4,b) = \phi$ $\quad \delta(4,c) = \phi$

(4,5)

$\delta(4,a) = \phi$ $\quad \delta(4,b) = \phi$ $\quad \delta(4,c) = \phi$

$\delta(5,a) = \phi$ $\quad \delta(5,b) = \phi$ $\quad \delta(5,c) = \phi$

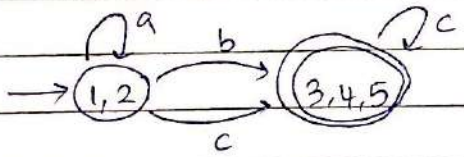
(5,3)

$\delta(5,a) = \phi$ $\quad \delta(5,b) = \phi$ $\quad \delta(5,c) = \phi$

$\delta(3,a) = \phi$ $\quad \delta(3,b) = \phi$ $\quad \delta(3,c) = 5$.

None are marked as distinguishable

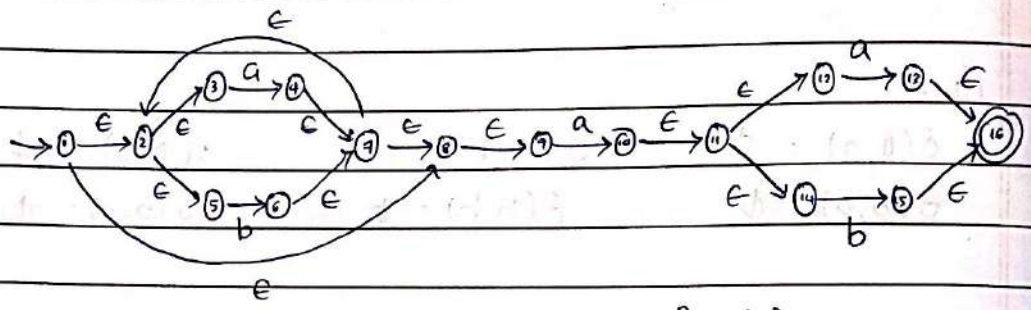
$\therefore (3,4) \cup (4,5) \cup (3,5) = \{3,4,5\}$ and $\{1,2\}$



15-09-19 Tut practise

1. Construct NFA, DFA and minimize the states of DFA. for the following

a) $(a|b)^* a(a|b)$ $L(1) = \{aa, ab, aaa, bbab, abab...\}$




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

$\bar{T} = \{1, 2, 3, 5, 8, 9\} = A$

$\bar{A}_a = \{\overline{4, 10}\} = \bar{4} \cup \bar{10} = \{4, 7, 8, 9, 2, 3, 5\} \cup \{10, 11, 12, 14\}$

$= \{2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 14\} = B$

$\bar{A}_b = \{\bar{6}\} = \{6, 7, 2, 3, 5, 8, 9\} = \{2, 3, 5, 6, 7, 8, 9\} = C$

$\bar{B}_a = \{\overline{4, 10, 13}\} = \bar{4} \cup \bar{10} \cup \bar{13} = B \cup \{13, 16\}$

$= \{2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 14, 13, 16\} = D$

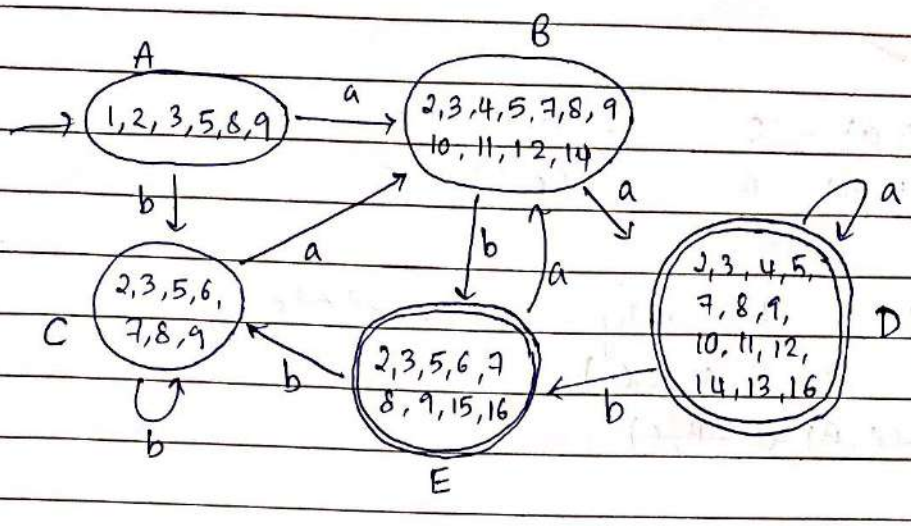
$\bar{B}_b = \{\overline{6, 15}\} = \bar{6} \cup \bar{15} = C \cup \{15, 16\}$

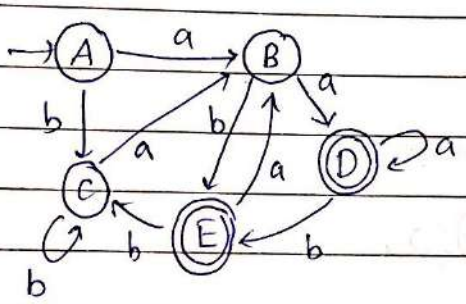
$= \{2, 3, 5, 6, 7, 8, 9, 15, 16\} = E$

$\bar{C}_a = \{\overline{4, 10}\} = B$ $\bar{C}_b = \{\bar{6}\} = C$

$\bar{D}_a = \{\overline{4, 10, 13}\} = D$ $\quad \bar{D}_b = \{\overline{6, 15}\} = E$

$\bar{E}_a = \{\overline{4, 10}\} = B$ $\quad \bar{E}_b = \{\overline{6}\} = C$




1. Distinguishable states

$(A,E)^{\checkmark}$, $(B,E)^{\checkmark}$, $(C,E)^{\checkmark}$, $(A,D)^{\checkmark}$, $(B,D)^{\checkmark}$, $(C,D)^{\checkmark}$

2. Similar States

$(A,B)^{\checkmark}$, $(B,C)^{\checkmark}$, $(C,A)$, $(D,E)^{\checkmark}$

Consider $(A,B)^{\checkmark}$

$\delta(A,a) = B$ $\quad \delta(A,b) = C$

$\delta(B,a) = D$ $\quad \delta(B,b) = E$

$(B,C)^{\checkmark}$

$\delta(B,a) = D$ $\quad \delta(B,b) = E$

$\delta(C,a) = B$ $\quad \delta(C,b) = C$

unmarked — combine

(C,A)

$\delta(C,a) = B$ $\quad \delta(C,b) = C$

$\delta(A,a) = B$ $\quad \delta(A,b) = C$

(D,E) ✓

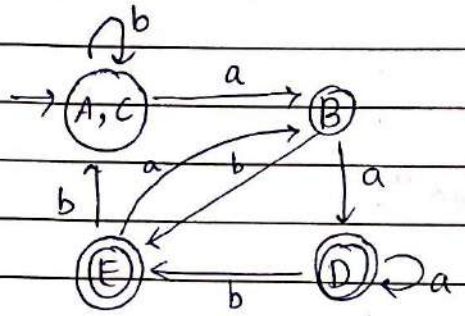
$\delta(D,a) = D$ $\quad \delta(D,b) = E$

$\delta(E,a) = B$ $\quad \delta(E,b) = C$

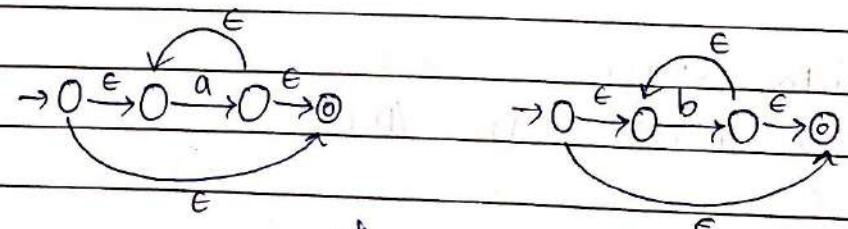
∴ (C,A) is ~~only~~ only similar state

and (so A ∪ C)

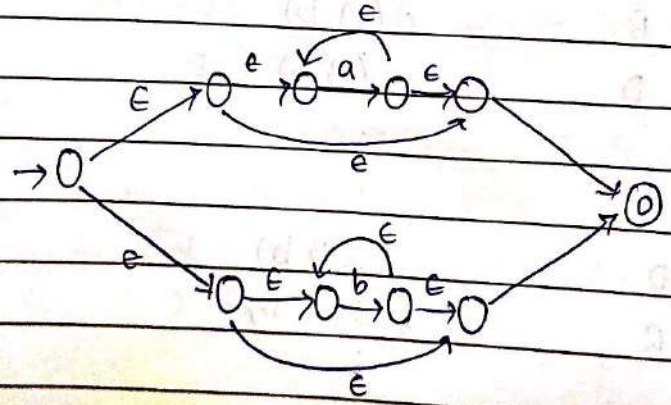
~~(A,B) ∪ (B,C)~~



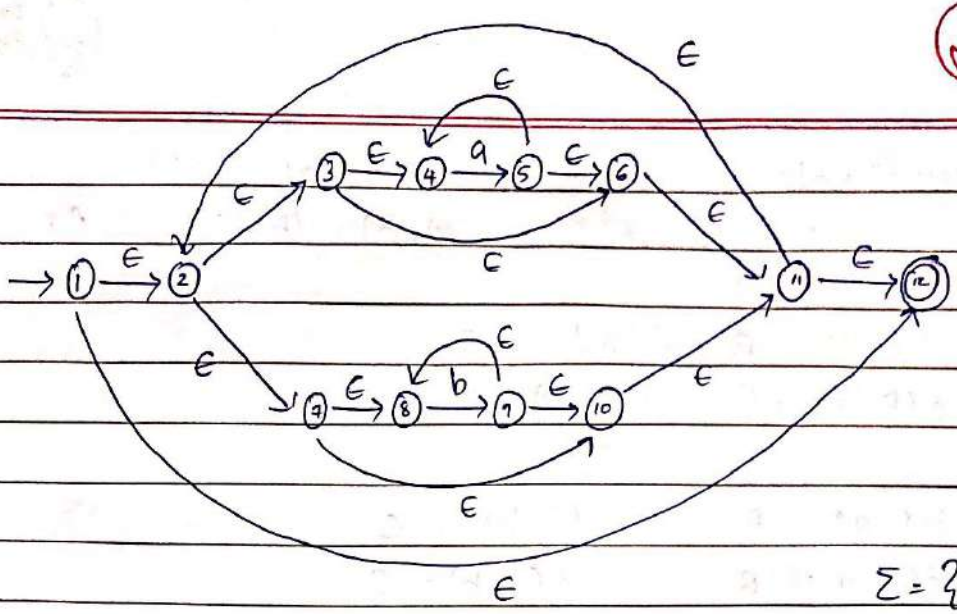

b) $(a^* \mid b^*)^*$ $\quad L(r) = \{\epsilon, a, b, aa, bb, aaa, bbb \ldots\}$




$a^*$ $\qquad$ $b^*$




$a^* \mid b^*$




$(a^*|b^*)^*$

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

$T = \{1, 2, 3, 4, 6, 11, 12, 7, 8, 10\}$

$= \{1, 2, 3, 4, 6, 7, 8, 10, 11, 12\} = A$

$\bar{A}_a = \{\bar{5}\} = \{5, 4, 6, 11, 2, 3, 7, 8, 12, 10\}$

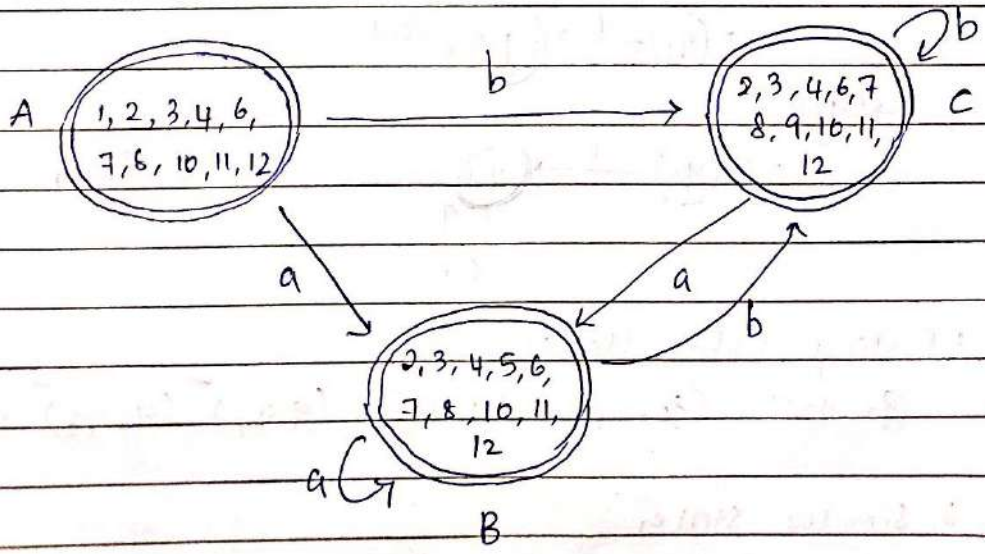
$= \{2, 3, 4, 5, 6, 7, 8, 10, 11, 12\} = B$

$\bar{A}_b = \{\bar{9}\} = \{8, 9, 10, 11, 12, 2, 3, 4, 7, 6\}$

$= \{2, 3, 4, 6, 7, 8, 9, 10, 11, 12\} = C$

$\bar{B}_a = \{\bar{5}\} = B \qquad \bar{B}_b = \{\bar{9}\} = C$

$\bar{C}_a = \{\bar{5}\} = B \qquad \bar{C}_b = \{\bar{9}\} = C$



1. Distinguishable
none

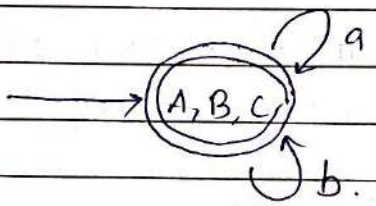
2. Similar states
(A, B) (B, C) (C, A)

$\delta(A, a) = B \quad \delta(A, b) = C$
$\delta(B, a) = B \quad \delta(B, b) = C$

$\delta(B, a) = B \quad \delta(B, b) = C$
$\delta(C, a) = B \quad \delta(C, b) = C$

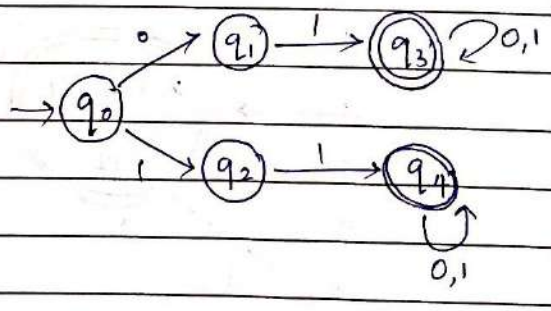
$\delta(C, a) = B \quad \delta(C, b) = C$
$\delta(A, a) = B \quad \delta(A, b) = C$

None are marked as distinguishable
$\therefore (A, B) \cup (B, C) \cup (C, A)$




2. Minimize of states of DFA for the given DFA below.

(1)




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

1. Distinguishable states
$(q_0, q_3)$, $(q_0, q_4)$, $(q_1, q_3)$ $(q_1, q_4)$ $(q_2 q_3)$ $(q_2 q_4)$

2. Similar states
$(q_0 q_1)$ $(q_1 q_2)$ $(q_0 q_2)$ $(q_3 q_4)$

$(q_0, q_1)$ ✓

$\delta(q_0, 0) = q_1$ $\quad$ $\delta(q_0, 1) = q_2$

$\delta(q_1, 0) = \phi$ $\quad$ $\delta(q_1, 1) = q_3$

$(q_1, q_2)$

$\delta(q_1, 0) = \phi$ $\quad$ $\delta(q_1, 1) = q_3$

$\delta(q_2, 0) = \phi$ $\quad$ $\delta(q_2, 1) = q_4$

$(q_0\ q_2)$ ✓

$\delta(q_0, 0) = q_1$ $\quad$ $\delta(q_0, 1) = q_2$

$\delta(q_2, 0) = \phi$ $\quad$ $\delta(q_2, 1) = q_4$

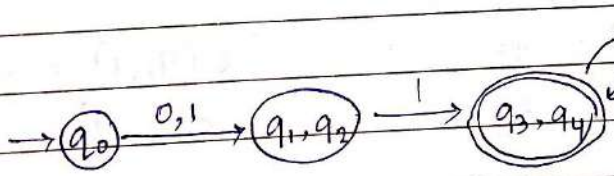
$(q_3\ q_4)$

$\delta(q_3, 0) = q_3$ $\quad$ $\delta(q_3, 1) = q_3$

$\delta(q_4, 0) = q_4$ $\quad$ $\delta(q_4, 1) = q_4$

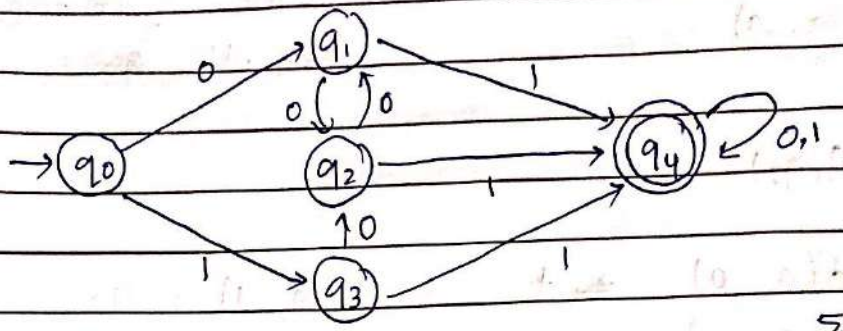
$(q_1 q_2)$ and $(q_3 q_4)$ remain as similar states.

$\therefore q_1 \cup q_2$ and $q_3 \cup q_4$




(21)

(2)




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

Distinguishable

~~$(q_0\ q_4)$~~ ✓ ~~$(q_1 q_4)$~~ ✓ ~~$(q_2 q_4)$~~ ✓ ~~$(q_3, q_4)$~~ ✓

Similar

~~$(q_0\ q_1)$~~ ✓ ~~$(q_1 q_2)$~~ ~~$(q_1 q_3)$~~

~~$(q_0 q_2)$~~ ✓ ~~$(q_2 q_3)$~~

~~$(q_0 q_3)$~~ ✓

$(q_0\ q_1)$ ✓

$\delta(q_0, 0) = q_1$ $\quad$ $\delta(q_0, 1) = q_3$

$\delta(q_1, 0) = q_2$ $\quad$ $\delta(q_1, 1) = q_4$

$(q_1, q_2)$

$\delta(q_1, 0) = q_2$ $\quad$ $\delta(q_1, 1) = q_4$

$\delta(q_2, 0) = q_1$ $\quad$ $\delta(q_2, 1) = q_4$

$(q_1, q_3)$

$\delta(q_1, 0) = q_2$ $\quad$ $\delta(q_1, 1) = q_4$

$\delta(q_3, 0) = q_2$ $\quad$ $\delta(q_3, 1) = q_4$

$(q_0\ q_2)$ ✓

$\delta(q_0, 0) = q_1$ $\quad$ $\delta(q_0, 1) = q_3$

$\delta(q_2, 0) = q_1$ $\quad$ $\delta(q_2, 1) = q_4$

$\lambda \equiv \in$
+ for |

$(q_2 q_3)$

$\delta(q_2, 0) = q_1$ $\quad$ $\delta(q_2, 1) = q_4$

$\delta(q_3, 0) = q_2$ $\quad$ $\delta(q_3, 1) = q_4$

$(q_0 q_3)$ ✓

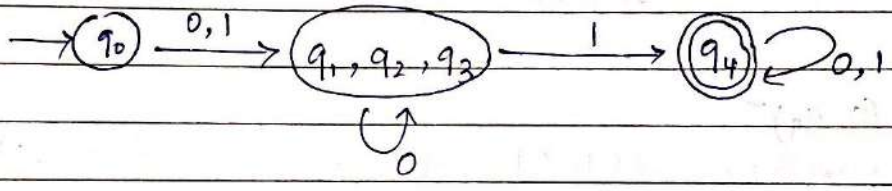
$\delta(q_0, 0) = q_1$ $\quad$ $\delta(q_0, 1) = q_3$

$\delta(q_3, 0) = q_2$ $\quad$ $\delta(q_3, 1) = q_4$

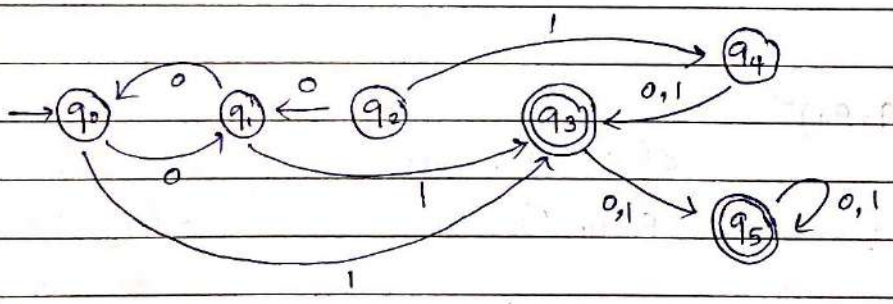
~~In second iteration, As $(q_0, q_1)$ ✓ $(q_0, q_2)$ ✓ $(q_0, q_3)$ ✓~~

⇒ As $(q_1 q_2)$, $(q_1 q_3)$, $(q_2 q_3)$ are unmarked they will be combined

$(q_1 q_2) \cup (q_1 q_3) \cup (q_2 q_3) = (q_1 q_2 q_3)$




(3)




Distinguishable

$(q_0, q_3)$ ✓ $(q_0, q_5)$ ✓

$(q_1, q_3)$ ✓ $(q_1, q_5)$ ✓

$(q_2, q_3)$ ✓ $(q_2, q_5)$ ✓

$(q_4, q_3)$ ✓ $(q_4, q_5)$ ✓

Similar

$(q_0 q_1)$ $(q_0 q_2)$ $(q_0 q_4)$

$(q_1 q_2)$ $(q_1 q_4)$

$(q_2 q_4)$ $(q_3 q_5)$

$(q_0, q_1)$

$\delta(q_0, 0) = q_1$ $\quad$ $\delta(q_0, 1) = q_3$

$\delta(q_1, 0) = q_0$ $\quad$ $\delta(q_1, 1) = q_3$

$(q_0, q_2)$ ✓

$\delta(q_0, 0) = q_1$ $\quad$ $\delta(q_0, 1) = q_3$

$\delta(q_2, 0) = q_1$ $\quad$ $\delta(q_2, 1) = q_4$

$(q_0, q_4)$ ✓

$\delta(q_0, 0) = q_1$ $\quad$ $\delta(q_0, 1) = q_3$

$\delta(q_4, 0) = q_3$ $\quad$ $\delta(q_4, 1) = q_3$

$(q_1, q_2)$ ✓

$\delta(q_1, 0) = q_0$ $\quad$ $\delta(q_1, 1) = q_3$

$\delta(q_2, 0) = q_1$ $\quad$ $\delta(q_2, 1) = q_4$

$(q_1, q_4)$ ✓

$\delta(q_1, 0) = q_0$ $\quad$ $\delta(q_1, 1) = q_3$

$\delta(q_4, 0) = q_3$ $\quad$ $\delta(q_4, 1) = q_3$

$(q_2, q_4)$ ✓

$\delta(q_2, 0) = q_1$ $\quad$ $\delta(q_2, 1) = q_4$

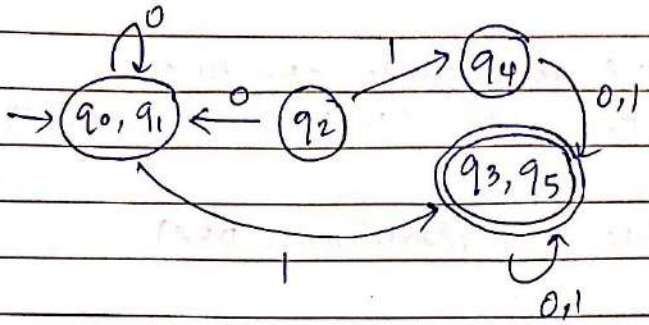
$\delta(q_4, 0) = q_3$ $\quad$ $\delta(q_4, 1) = q_3$

$(q_3, q_5)$

$\delta(q_3, 0) = q_5$ $\quad$ $\delta(q_3, 1) = q_5$

$\delta(q_5, 0) = q_5$ $\quad$ $\delta(q_5, 0) = q_5$

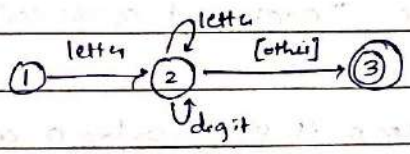
$q_0 \cdot \cup q_1$ and $q_3 \cup q_5$




Implementation of Finite Automata in Code.

Code is written in pseudocode using a language ~~constraints~~ constructs.

Ex: identifiers




[] indicates read and unread.

other is used to recognise (,) which differentiates b/w two identifiers.

Code to accept identifiers:

{starting in state 1}
if the next character is a letter then
advance the input;
{state 2}
while the next character is a letter or digit do
advance the input {stay in state 2}
end while;
{go to state 3 without advancing}
else
{errors or other cases};
end if;

Generic code:

Express DFA as a data structure.

Use of transition table. (table driven method).

Transition table of identifier (DFA):

| Inputstate / state | letter | digit | other | Accepting |
|---|---|---|---|---|
| 1 | 2 | | | No |
| 2 | 2 | 2 | [3] | no |
| 3 | | | | Yes. |

Brackets indicate "noninput-consuming" transitions.

Blank - error.

& Accepting column is a Boolean array

code scheme:

```
State := 1;
    ch := next input character;
    while not Accept [state] and not error (state) do
        newstate := T[state, ch];
        if Advance [state, ch] then ch := next input char;
            state := newstate;
    end while;
    if Accept[state] then accept;
```