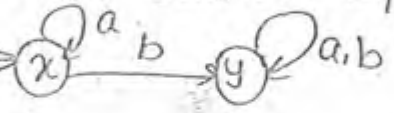
Dated 25.Oct.10

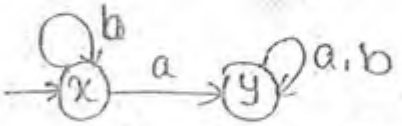
Day-1

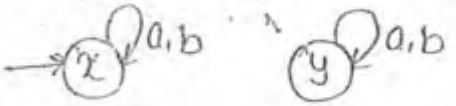
COMPILER

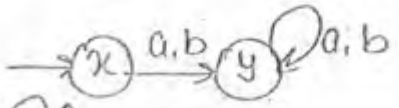
Q. How many possible finite automata's are there, where x is always initial state over the alphabet x and y or a and b.

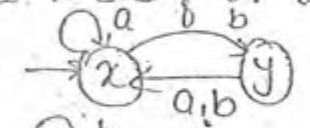
Sol"

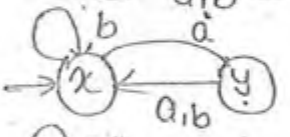
1. 

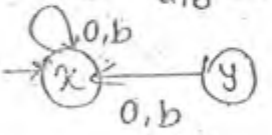
2. 

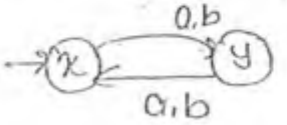
3. 

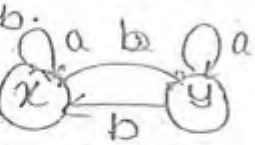
4. 

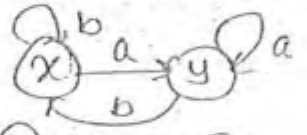
5. 

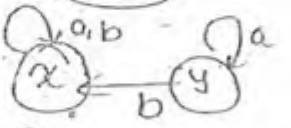
6. 

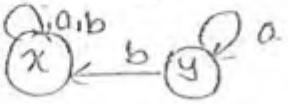
7. 

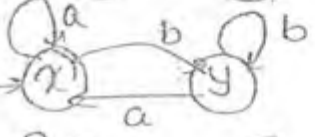
8. 

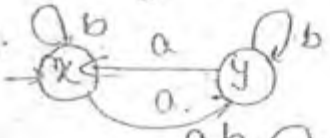
9. 

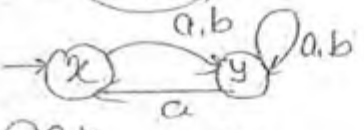
10. 

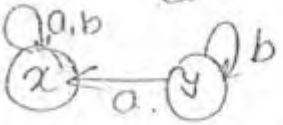
11. 

12. 

13. 

14. 

15. 

16. 

find - On the basis of No of final states

final 2 (x & y) = 16

1 (x (or) y) = 16 / 16 = 32

0 (no final) = 16

64 Ans

→ Consideration of No of possible finite automata's can be done as:-

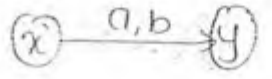
		a	b
2	x	2	2 → 4
2	y	2	2 → 4
4			16

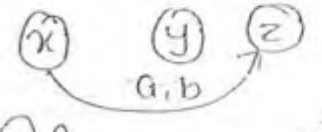
16 × 4 = 64 Ans

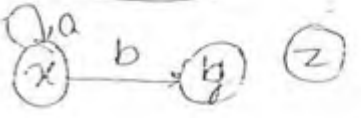
Q. How many possible finite automata's with three states x, y, z where x is always initial states over the alphabet a and b.

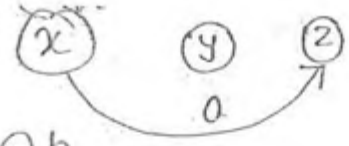
Sol"

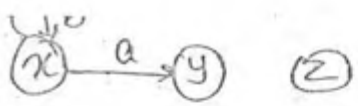
1. 

2. 

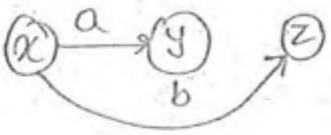
3. 

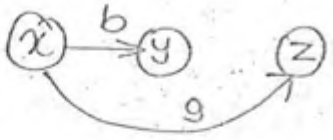
4. 

5. 

6. 

7. 

8. 

9. 

		a	b	
2	x	3	3 ⇒ 9	
2	y	3	3 ⇒ 9	729
2	z	3	3 ⇒ 9	
8				→ 5832 Ans

final
0 → 1
2 → 3
1 → 3
3 → 3

QN How many possible finite automata's are there states x, y & z over the alphabet a and b

Sol" x is initial → 5832
y is initial → 5832
z is initial → 5832
} = 3×5832 = 17496

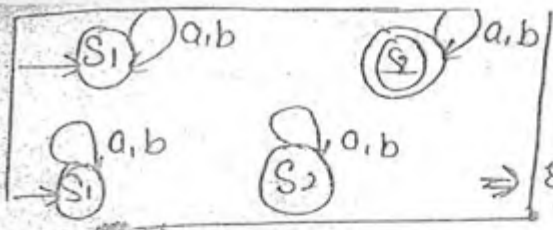
QN How many possible finite automatas are there with three states x y and z, over the alphabet a, b and c, where x is both initial and final.

	a	b	c	
x	3	3	3	3^3 ⇒ 19,683 Ans
y	3	3	3	
z	3	3	3	

QN How many possible finite automatas are there with state x any, where x is always initial state over the alphabet a, b that accepts empty language.

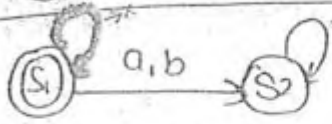
Sol"

		a	b	
2	x	2	2	2^4 ⇒ 16
2	y	2	2	
4				16×4 = 64



⇒ Empty language. (final state is not reachable)

⇒ Empty language (not any final state)



⇒ 1, not empty language.

possibilities of finding the automatas which accepts empty language.

→ No final state = 16 (Basic and easy)

→ final state are not reachable = 4

20 possibilities

If there are two states then-

2 — 16 (both are final) ✗

1 → 16 (x is initial) ✗

16 (4) ✓

0 → 16 (Empty language) ✓

QN How many possible finite automates are there with two states x and y, where x is always initial state with alphabet a and b, that accepts everything.

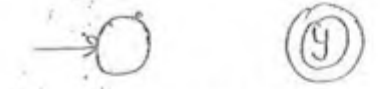
Soln 2 → 16 ✓

1 → x → 16 (4) ✓ ⇒ 20

y → 16 ✗

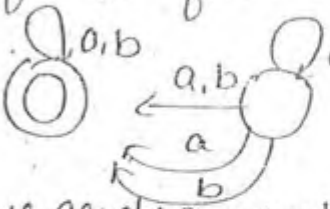
0 → 16 ✗

If y is final state ⇒



(y not accepting ∈). ✗

If x is final then-

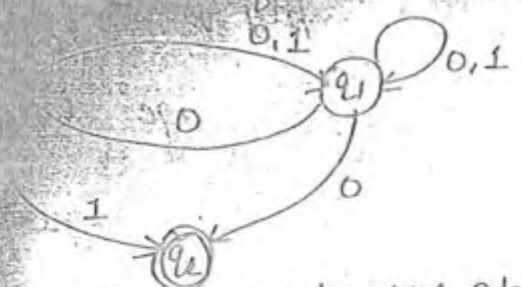


⇒ 4

Total possibilities = 20

Note:- If 20 DFA's are there which are accepting empty language then 20 DFA's should be there which will accepts everything, because of complementation rule.

finite automata →



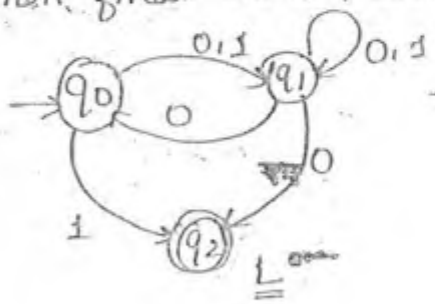
age accepted by the above finite automata and
ge accepted by the above finite automata by
al and non final states, then which one of the
tent :-

1)* = L

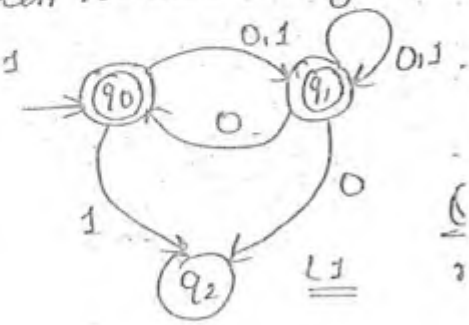
= (0+1)*

$L_1 \subseteq (0+1)^*$

$L_1 = L$



L



L_1

If it is DFA then $\Rightarrow (0+1)^* - L$ (Remove L from Σ).

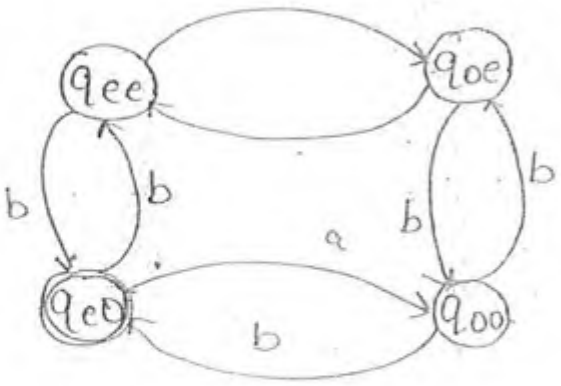
Note:- In the case of DFA we will always get complimented finite automata, but in the case of NFA, we will not always get complimented finite automata (manual checking is only solution).

Q.N Construct finite automata that accepts all strings of a's and b's where no of a's in given string is even and no of b's in the give string is odd.

Soln

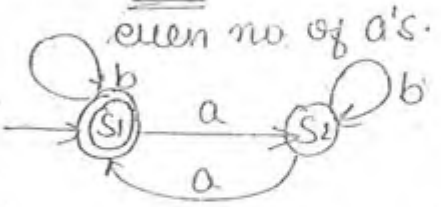
S

S_1 even a's S_2 odd b's

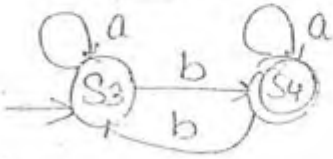


P1

even no of a's.



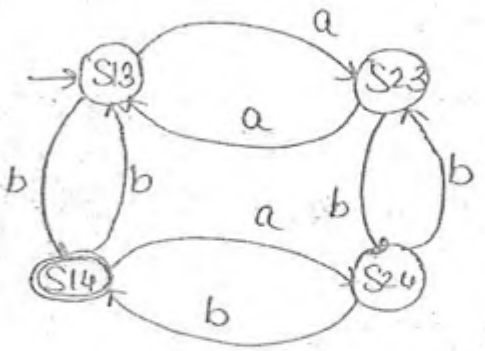
P2 (odd no of b's)



By P1 & P2

$S_{13} \xrightarrow{a} (S_1, a) \cup (S_3, a) = S_2 S_3$

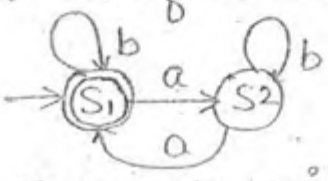
$S_{13} \xrightarrow{b} (S_1, b) \cup (S_3, b) = S_2 S_4$



final state ⇒ where both the finals are there.

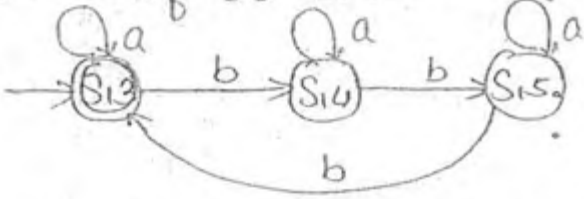
Q.1 Construct FA which accepts all strings of a's and b's in which no of a's are divisible by 2 and no of b's are divisible by 3.

P_1: no of a's divisible by 2.



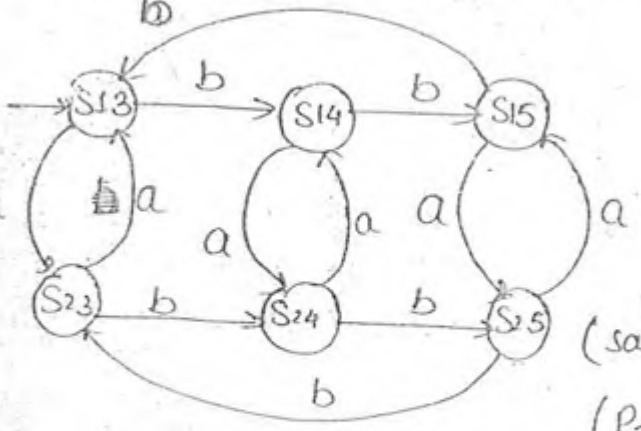
How many states
= 2×3 = 6 states.

P_2: no of b's divisible by 3.



$S_{13} \xrightarrow{a} S_{23}$ $S_{15} \xrightarrow{a} S_{25}$

$S_{13} \xrightarrow{b} S_{14}$ $S_{15} \xrightarrow{b} S_{13}$



final states

$P_1 \wedge P_2 = S_{13}$

$P_1 \cup P_2 = S_{13}, S_{14}, S_{15}, S_{23}$

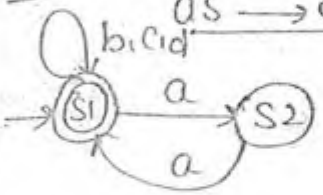
$P_1 - P_2 = S_{14}, S_{15}$
(satisfying 1st but not second one)

$(P_2 - P_1) = S_{23}, S_{24}, S_{25}$

Q14 Find the minimum no of state required to construct minimum dfa which will accept all strings of a's, b's and c's, d's where the no of a's divisible by 2, No of b's not divisible by 3, no of cs not divisible by 5, No of ds not divisible by 7. How many minimum no of states required.

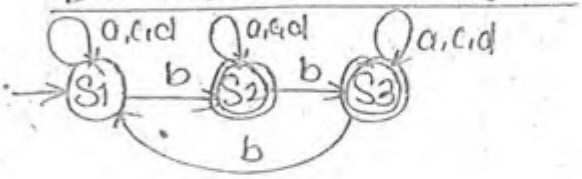
Soln P1

a's → divisible → 2



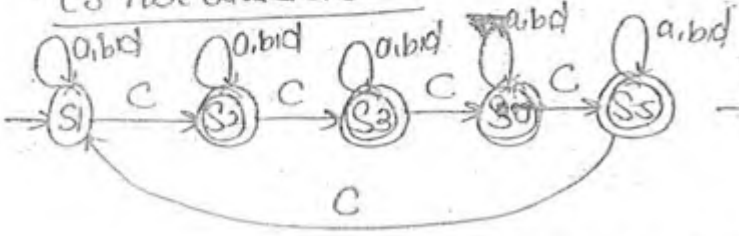
P2

b's → not divisible by 3



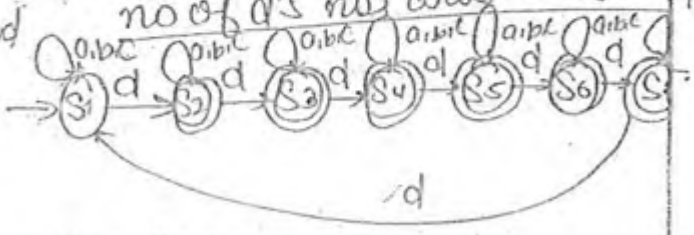
P3

C's not divisible - 5



P4

no of d's not divisible by 7

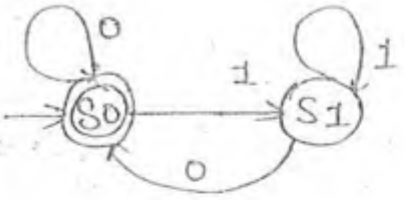


Minimum No of states = 2 * 3 * 5 * 7 = 210 Ans

Q15 Construct finite automata that accepts all binary number which are divisible by 2.

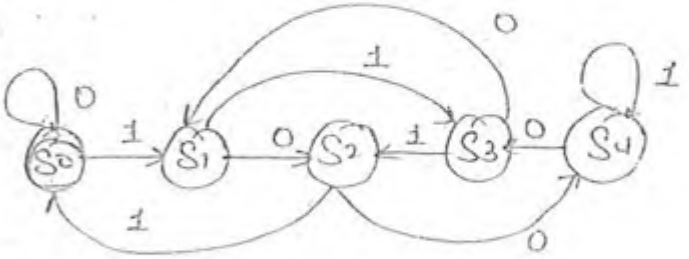
Soln: If initial and final states are same then this method can be utilized ⇒

	0	1
→ S_0^*	S_0	S_1
S_1	S_0	S_1



⇒ all binary no divisible by 5 - (Remainders = 0, 1, 2, 3, 4)

	0	1
→ S_0^*	S_0	S_1
S_1	S_2	S_3
S_2	S_4	S_0
S_3	S_1	S_2
S_4	S_3	S_4



→ all ternary no divisible by 7 -

	0	1	2
→*	S0	S1	S2
S1	S3	S4	S5
S2	S6	S0	S1
S3	S2	S3	S4
S4	S6	S6	S0
S5	S1	S2	S3
S6	S4	S5	S6

Binary or ternary no matter

always contain 7 states

Note:- The minimum no of states required to construct finite automata that accepts all base m numbers, which are divisible by n, Contain n-states.

→ all binary no's divisible by 5 and starting with 0.

	0	1
→ S	S0	D ⇒ start with 0
* S0	S0	S1
S1	S2	S3
S2	S4	S0
S3	S1	S2
S4	S3	S4
D	D	D ⇒ start with 1

divisible by 5

⇒ 7 states

→ all binary no divisible by 9 and starts with 1

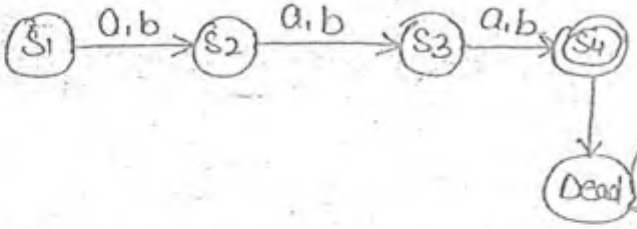
	0	1
→ S	D	S1 ⇒ starting with 1
* S0	S0	S1
S1	S2	S3
S2	S4	S5
S3	S6	S7
S4	S8	S0
S5	S10	S2
S6	S3	S4
S7	S5	S6
S8	S7	S8
D	D	D ⇒ starting with 0

Divisible by 9

⇒ 9+2 = 11 states

QN Construct a FA that accepts all strings of a's and b's, where the length of the string of exactly 3.

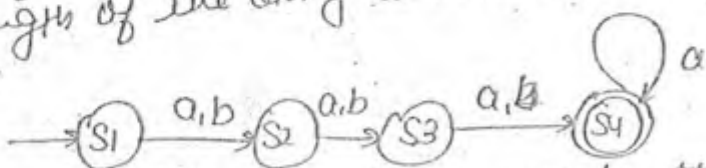
Soln



⇒ 3+2 = 5 states

Note:- The minimal finite automata that accepts all strings of a's and b's, where the length of string is 'n' exactly contains (n+2) states.

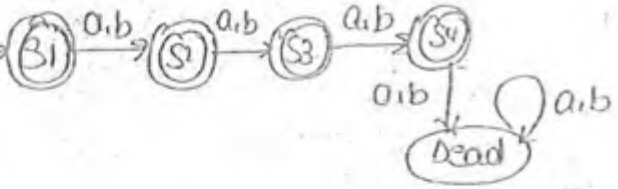
QN Construct minimal FA that accepts all strings of a's and b's where the length of the string is atleast 3:



Note:- The minimal finite automata that accepts all string's of a's and b's, where the length of string atleast n contains (n+1) states.

QN FA, accepts all strings of a's and b's where the length of string is almost three.

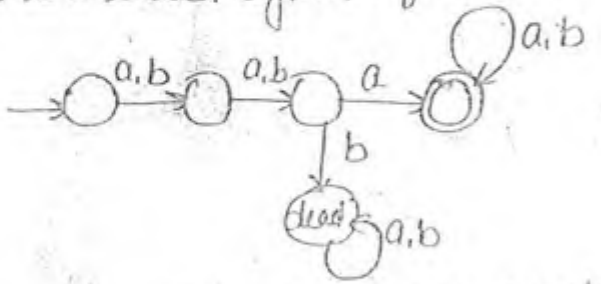
Soln



= 3+2 = 5 states

Note:- where the length of strings are almost n, contain n+2 state

QN FA, that accepts all strings of a's and b's, where each string conta 'a' as the third symbol from LHS.



⇒ 3+2 = 5 states

Note:- The minimal FA that accepts all strings of a's and b's where n'th symbol from L.H.S. contain n+2 states.

Note:- The minimum FA which accepts all the strings of a's and b's, where nth symbol from the right hand side is b contains 2^n states.

Lec-2

Dated 5.Oct.10

$L_1 = \{a^n \mid n \geq 1\}$ → Regular

$L_2 = \{a^n b^n \mid n \geq 1\}$ → CFL (Regular + 1 stack)

$L_3 = \{a^n b^n c^n \mid n \geq 1\}$ → CSL

$L_4 = \{a^m b^n \mid m \neq n,\ m, n \geq 1\}$ → CFL

$L_5 = \{a^l b^m c^n \mid l \neq m \text{ or } m \neq n\}$ → CFL

How can we say that what the language is given as:- ($l, m, n \geq 1$)

If we want to check that given language is regular or not) ⇒ Language

- Finite language (Regular language)
- Infinite language
  - Memory required (non-regular) (any comparision)
  - Memory not required
    - not in AP (not regular)
    - in AP (Regular)

Examples

$L = \{a^m b^m \mid m \leq 10,000\}$

Regular language (just because of finiteness)

$L = \{a^n \mid n \geq 1\}$ (Regular)

Here we are using the concept of generation of series.

$L = \{a^{n!} \mid n \geq 1\}$

Theorem:- For a given problem if you can construct the CFG, then it can surely we solved by T.M.

Non-regular (not in AP)

$L = \{a^{2^n} \mid n \geq 1\}$ = not regular (not in AP)

④ $L = \{a^{2n} \mid n \geq 1\}$ ⇒ even no of a's.

Regular language (in AP)

⑤ $L = \{a^n \mid n \geq 1\}$

⇒ not regular (not in AP)

⑥ $L = \{a^n b^n \mid n \geq 1\}$

⇒ not regular (memory required)

⑦ $L = \{w w^R \mid n \geq 1\}$

⇒ non regular (memory required)

⑧ $L = \{a^i b^j \mid \gcd(i,j) = 1\}$

⇒ non regular (memory required).

ϕ **Regular ∧ CFL ⇒ CFL** (Intersection of lower and higher language will always go to high language).

$(a+b)^* \wedge a^n b^n = a^n b^n$

ϕ **CFL ∧ CFL = not CFL (CSL)** → (need not be)

$a^l b^l c^m \wedge a^m b^n c^n$

$a^l b^l c^l$ (not CFL)

ϕ $a^n b^n c^n$ = CSL ⇒ Compliment of CSL is need not be CSL.

$\Downarrow$

$(a^n b^n c^n)^l$ ⇒ Compliment of CFL, need not be CFL, it can be CSL. (CFL's are not closed under compliment ation).

$\Downarrow$

CFL

Note:- ① Insersection → CFL ⇒ need not be CFL or (CSL)

② Compliment of CFL ⇒ need not be CFL or (CSL)

③ Intersection of Regular and CFL is always CFL.

Q. Give the regular expression that derives all strings of a's, where each string begin with a and end with b.

$R = \{a.(a+b)^*.b\}$ Ans ⇒ Concatenation order is important

$a+b = b+a$ (order can be changed)

Q. Regular expression, where the first and last symbols are different

⇒ $a(a+b)^*b + b(a+b)^*a$

Q. Regular expression, that derives all string's of 0's and 1's, where each string starting and ending symbols are same.

⇒ $a(a+b)^*a + b(a+b)^*b + \lambda + a + b$

Q. Give the Regular expression that derives all strings of a's and b's, where all strings contains abb as substring.

⇒ $(a+b)^*\, a\, bb\, (a+b)^*$

Q. Regular expression, where the length of string is exactly three.

⇒ $(a+b)(a+b)(a+b) = (a+b)^3 \Rightarrow (a+b)^n$

Q. Regular expression, where the length of string is atleast 3.

⇒ $(a+b)(a+b)(a+b)(a+b)^*$ ⇒ more effecient

⇓ (or)

$(a+b)^*(a+b)^3$ ⇒ more effecient

⇓ (or)

$(a+b)^*(a+b)^3(a+b)^*$ ⇒ more correct but not effecient

Q. Regular expression, where the length of string is atmost 3.

$= \lambda + (a+b) + (a+b)(a+b) + (a+b)(a+b)(a+b) = (a+b+\lambda)^3$

Q. Regular expression, where the length of string is even.

$= ((a+b)^2)^*$

Q. Odd length-

$(a+b)((a+b)^2)^*$ or $((a+b)^2)^*(a+b)$

Q. Regular expression, that where each string starts with a and not having two consecutive b's.

$= (a + ab)^+$ (or) $a(a+ba)^*(\lambda+b)$

(a+ab)*

QN Regular expression, where each string does not contain 2 consecutive a's and b's

Soln = $(b+\epsilon)(a+b)^*(a+\epsilon)$ or $(a+\epsilon)(ba)^*(b+\epsilon)$

QN Regular expression, where each string contain exactly 2 a's.

$= b^* a b^* a b^*$

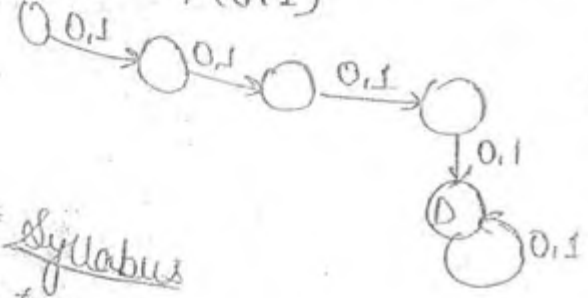
QN Regular expression, where each string contain atmost two a's.

$= b^* + b^* a b^* + b^* a b^* a b^*$

$= b^*(a+\epsilon) b^* (a+\epsilon) b^*$

QN find the minimal states of dfa that accepts described by the R.E. = (0+1)(0+1)(0+1)(0+1) ..... n-times

(0+1)(0+1)(0+1)



If n=3, ⇒ 5 states ⇒ (3+2)

If n=n then (n+1) states

* Syllabus

① Lexical analyser
② Parsing
③ Syntax directed Translation
④ Intermediate code generation
⑤ Code Optimization

} ⇒ Imp

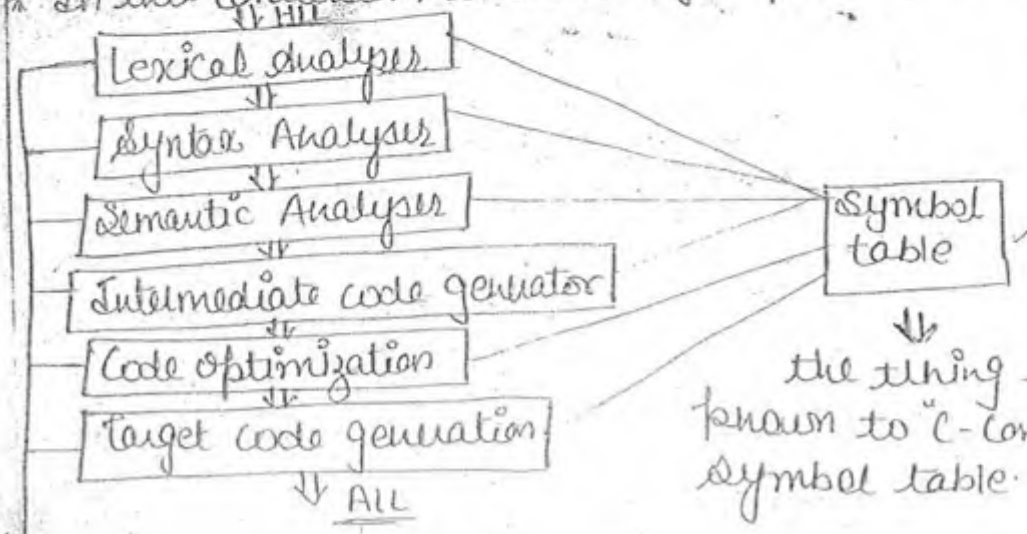
References

→ Compiler Technique & Aho Ullmen & Ravisethi

INTRODUCTION

Compiler
⇓ HLL
Assembly level language

* Compiler is a convertor, that can convert High level language into Assembly level language.
* In this conversion, we are using 6-phases, which are as follows:-



* C-Compiler knows everything about C-language.

⇓
the thing that are not known to C-Compiler, is stored in symbol table.

⇒ used to handle all the errors made by user.

Exp:- $x = a + b * 60.5$

⇓
Lexical Analyser
⇓
$(id_1, 1) = (id_2, 2) + (id_3, 5) * 60.5$

7 tokens

$id_1 = id_2 + id_3 * 60.5$

⇓
Syntax Analyser

Symbol table

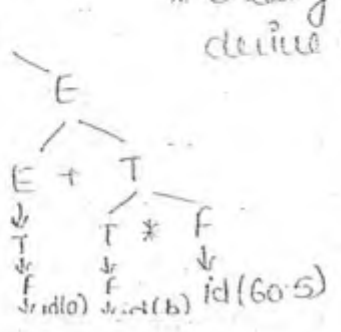
1	int	x
2	int	a
3	int	b

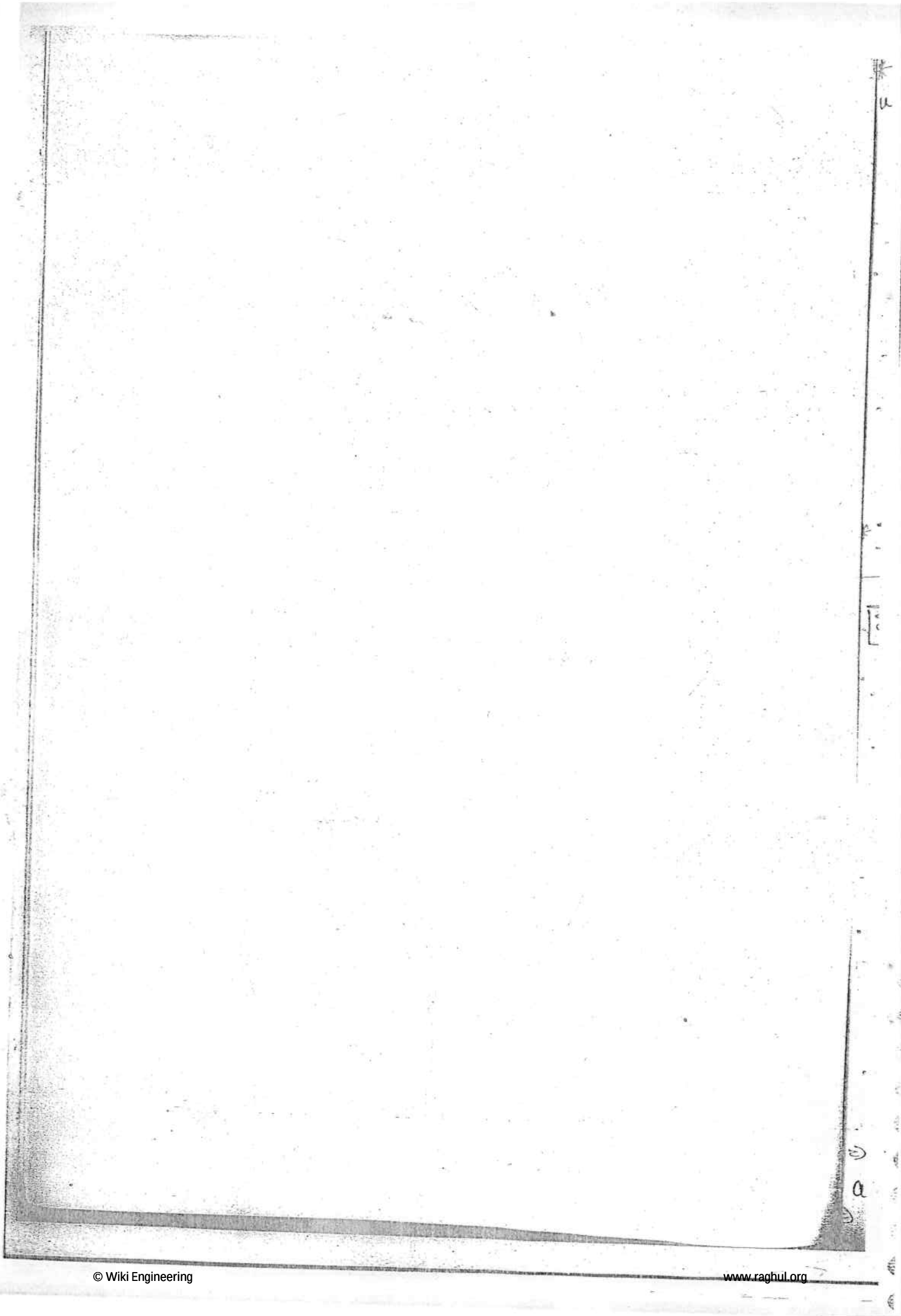
x, a, b = identifiers

* C language compiler, uses CFG, to define all the arithmetic expression.

$a + b = c$ = Syntax error
⇓
CFG can't generate this

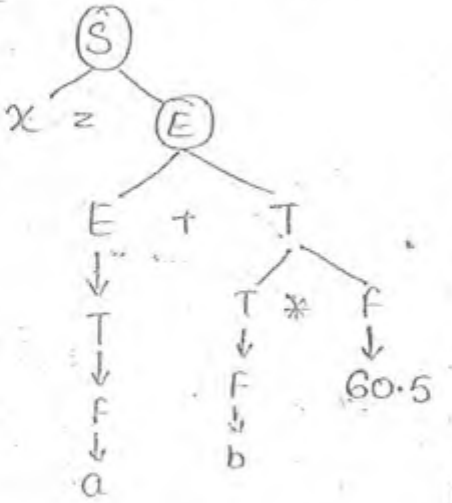
$S \rightarrow id = E$
$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow id$





* Type checking will never be take by the syntax and will done by semantic analyser.

Semantic Analysis (Type checking)



- How can we multiply int to floating point number?
- for this semantic analyser uses the Implicit conversion by float to int (60.5) = 60

* Type mismatch type errors are given by semantic analyser.

$x = a + b * 60$ (CFG take care of priorities itself)

Intermediate Operations or code

$t_1 = b * 60$ (t_1, t_2) = temporary variables

$t_2 = a + t_1$

$x = t_2$

⇓

Code Optimization

⇓

$t_1 = b * 60$

$x = a + t_1$

⇓

Target Code

⇓

MOV b, R1 (b to R1).
MUL R1, 60 (R1 = R1 * 60)
MOV a, R2 (a to R2)
ADD R1, R2 (R1 = R1 + R2)
MOV R1, x (R1 to x)

L.A.
Syntax
Semantic
I.C.G.
} front end

Code Optimizn. ⇒ optional phase

Target code ⇒ Back end

Front end = Depends upon source language

Back end = Depends on processor

In order to achieve the portability, we seprate the phases of analysis

for the same source code, we can generate different assembly language

sets; based upon type of processors.

Types of Compiler

1. Single pass Compiler
2. Multi pass Compiler

(1) Single pass Compiler:- all the 6-modules at a time in memory

Disadvantage :-
① Wastage of space.

Advantage
① No Divide and Conquer (less time required).

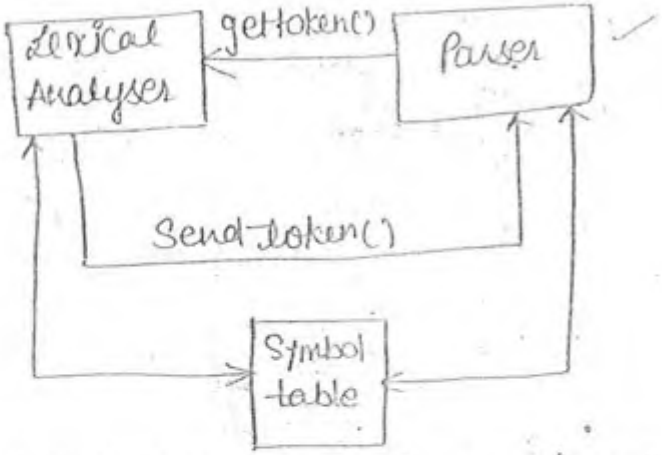
(2) Multi pass Compiler:- front end and back end are placing seprately in memory.

Disadvantage
① More time required (Divide and conquer)

Advantage
① Less space required.

Chapter No 1

Lexical Analysis



* Initially the I/P is given to parser. parser calls lexical analyser for tokens

Lexical Analyser :- ① It will read the given I/P strings and divides the string into some meaningful groups or words which are called as tokens
② It will remove the comment lines in the given C pgm.

3) It will eliminate white space characters, in the source c
White Space Charaters → blank(Space), tab, newline characters

4) It will help to provide error messages. It scan each and eve
line of source code. The line number is also provided by the
lexical analyser.

Q) Find the no of tokens in the following C pgm:-

```
int max(i,j)
 int i, j;
 /* return max of i & j */
 {
  return i>j ? i : j;
 }
```

Sol^n

```
int | max | ( | i | , | j | ) |
int | i | , | j | ; |
{ |
return | i | > | j | ? | i | : | j | ; |
} |
```

= 23 tokens Ans

Q) By not seeing the next symbol, we can say that is a token:-

a) ++
b) > ⇒ = may be, then (>=) ⇒ token
c) main ⇒ may be main(or some user defined variable
d) < ⇒ = may be, then (<=) ⇒ token

Q) find the no of tokens,

```
printf ("Hai x = %d", i);
```

Sol^n printf | (| " Hai x = %d " | , | i |) | ;
(1) (2) (3) (4) (5) (6) (7)

⇒ Inside " " not need to enter = 7 tokens Ans

Q) Find the no of tokens

```
printf ("i = %d, &i = %x", i, &i);
```

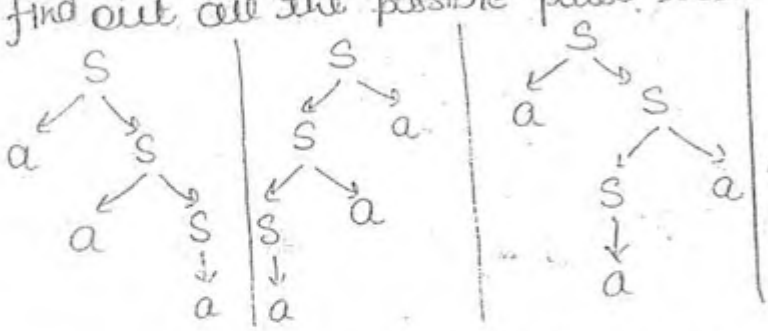
Sol^n printf | (| " i = %d, &i = %x " | , | i | , | &i |) | ; = 10 tokens Ans

ϕ Grammar

Exp:- $S \to aS \mid Sa \mid a$

$w = aaa$

find out all the possible parse trees.



$S \to aS \mid Sa \mid a$

$\Downarrow \quad \Rightarrow a^+$

$S \to a \mid aS \Downarrow$

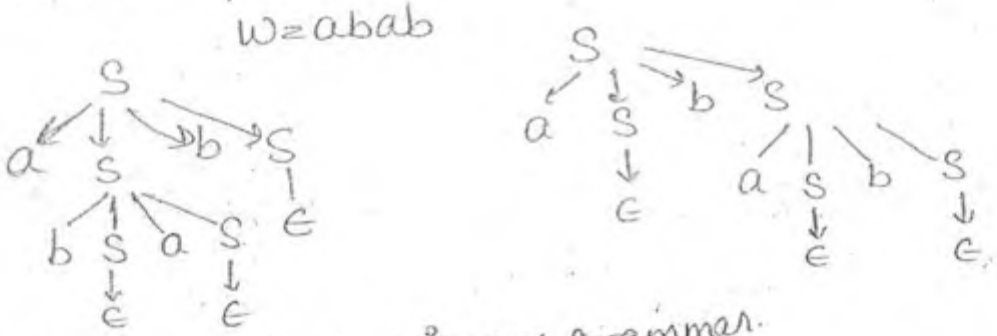
$\Rightarrow a^+$

The above grammar is ambiguous grammar, because more than 1-parse tree is available to derive a string $w = aaa$

ON check the following grammar is ambiguous or not.

$S \to aSbS \mid bSaS \mid \epsilon$

$w = abab$

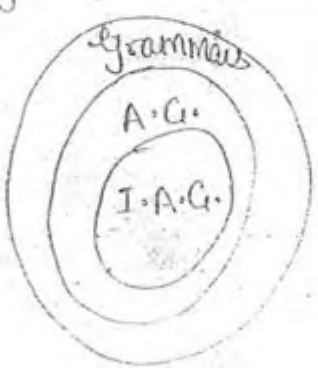


Given grammar is ambiguous grammar.

Note:- ① There is no algorithm to check whether the given grammar is ambiguous grammar or not so it is an undecidable problem.

② There is no algorithm, to convert the given ambiguous grammar into unambiguous grammar. It is also undecidable problem.

③ Those ambiguous grammars, from which we can not eliminate ambiguity, is called as inherently ambiguous grammars.



Grammar

CFG

- Left Recursive Grammar
  - $A \to Aa \mid a$
  - $A \to A\alpha \mid \beta$
  - $\alpha, \beta \in (T \neq V)^*$
  - A = Single variable
- Right Recursive Grammar
  - $A \to aA \mid a$
  - $A \to \alpha A \mid \beta$
  - $\alpha, \beta \in (V+T)^*$
  - A = Single variable

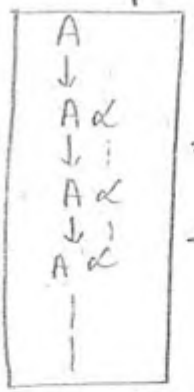
Note ① For every Left Most Derivation Tree, Right Most Derivation Tree is also possible.

② If the given grammar is unambiguous grammar, LMDT, RMDT both are same.

③ Top Down Parser will always keep the LMRecursive grammar into infinite loop

Exp:- $A \to A\alpha \mid \beta$

$w = \beta\alpha^{100}\beta$



⇒ Infinite loop

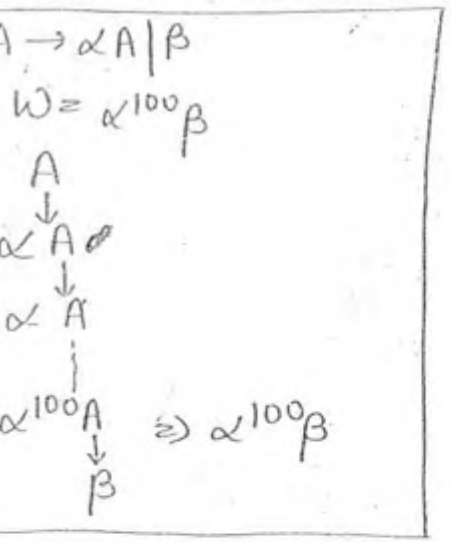
repeated again & again

Exp:- Right Recursion (Top Down Parser)

→ No problem will occur.

$A \to \alpha A \mid \beta$

$w = \alpha^{100}\beta$



Elimination of Left Recursion

Extended Backneous Normal form (EBNF)

Exp:- $E \to E+T \mid T$

$T \to T * F \mid F$

$F \to id$

Sol^{n} $E \to E+T \mid T$

$T(+T)^0$
$T(+T)^1$
$T(+T)^2$
$T(+T+T)^3$
$T+T+T+T+T+T+T$---

$E \to T\{+T\}$

$T \to T*F \mid F$

$F(*F)^0$
$F(*F)^1$
$F(*F)^2$
$F(*F*F)^3$
$F*F*F*F*$----

$T \to F\{*F\}$

$\Rightarrow$ $E \to T\{+T\}$
$T \to F\{*F\}$
$F \to id$

Dm

Drawback:- Equivalent grammar is not CFG.

Exp:- $E \to E+T \mid T$
$T \to T*F \mid F$
$F \to id$

$\Rightarrow$ Eliminate left recursion:

Recursion should be there but not left recursion.

Sol^{n}

$E \to E+T \mid T$ | $T \to T*F \mid F$ | $F \to id$
$E \to TE'$ | $T \to FT'$ |
$E' \to +TE' \mid \epsilon$ | $T' \to \epsilon \mid *FT'$ |

Dm

Exp:- Eliminate left recursion-

$S \to aBDh$
$B \to Bb \mid h$
$D \to EF$
$E \to g \mid \epsilon$
$F \to f \mid \epsilon$

Sol^{n}

$B \to Bb \mid h$ | $S \to aBDh$
$B \to hB'$ | $D \to EF$
$B' \to \epsilon \mid bB'$ | $E \to g \mid \epsilon$
| $F \to f \mid \epsilon$

Dm

minate left recursion

(L) | a
L, S | S

S → (L) | a
L → SL'
L' → ∈ | , SL'

Eliminate left recursion

→ Aa | b
→ AC | Sd | ∈

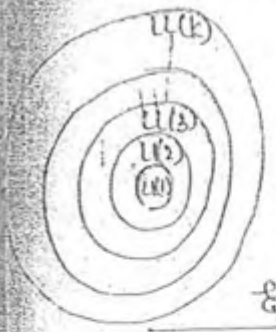
S → Aa | b
A → Ac | Aad | ∈ | bd

⇓

S → Aa | b
A → bd A' | A'
A' → c | cA' | adA'

Left factoring

Parser sees one symbol at a time, from left to right.



$S \to a\alpha_1 \mid a\alpha_2 \mid a\alpha_3$

$w = a\alpha_3$

⇒ Parser is confused to choose out of these, bcoz all are giving 'a'. This is known as left factoring.

⇓ Elimination

$S \to a\beta$
$\beta \to \alpha_1 \mid \alpha_2 \mid \alpha_3$

QN Eliminate left factoring from the following grammar :-

S → iEtS | iEtSeS | a
E → b

Sol' S → iEtSS' | a
S' → ∈ | eS
E → b

QN Eliminate left recursion

S → A
A → Ad | Ae | aB | aC
B → bBC | f

Sol' S → A
A → aBA' | aCA'
A' → ∈ | dA' | eA'
B → bBC | f

QH Eliminate left factoring-

$S \to aAd \mid aB$

$A \to a \mid ab$

$B \to ccd \mid ddc$

Sol^n $S \to aS'$

$S' \to Ad \mid B$

$A \to aA'$

$A' \to \epsilon \mid b$

$B \to ccd \mid ddc$

QH $S \to abc \mid abd \mid aef$

Sol^n $S \to aS'$

$S' \to bc \mid bd \mid ef$

$S' \to bB' \mid ef$

$B' \to c \mid d$

$\Rightarrow$ all having the same priorities

ϕ $E \to E+E \mid E*E \mid E/E \mid E-E \mid id$

$E \to E+T \mid T$ ⇒ lower priority

$T \to T*F \mid F$ ⇒ higher priority

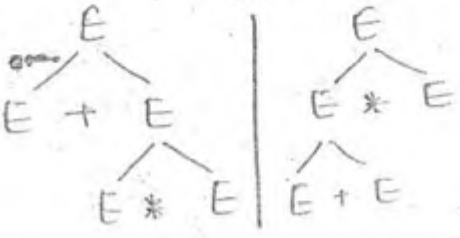
$F \to id$

$E \to E+T \mid E-T \mid T$

$T \to T*F \mid T/F \mid F$

$F \to id$

$w = a+b*c$



ambiguous grammar

* In CFG, the thing that will be on lower level, will be evaluated first
* The operator which have lower priority, make that root

QH Convert the following ambigious into unambiguous grammar.

$R \to R+R \mid R \cdot R \mid R^* \mid a \mid b$

Sol^n $R \to R+T \mid T$

$T \to T \cdot F \mid F$

$F \to (G)^* \mid G$

$G \to a \mid b$

QH ambigious → unambigious

$Bexpr \to Bexp \text{ or } Bexp \mid Bexp \text{ and } Bexp \mid \text{not } Bexp \mid 0 \mid 1$

Sol^n $Bexpr \to Bexp \text{ or } T \mid T$

$T \to T \text{ and } F \mid F$

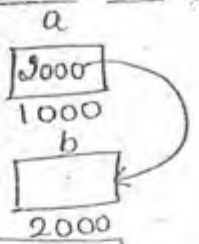
$F \to \text{not } G \mid G$

$G \to 0 \mid 1$

End

Some extra and important points (D.S.)

① int *a;
② int b;
③ a = &b



a = variable pointer

① int (*fp)()
② int fun();
③ fp = fun;
(*fp)();

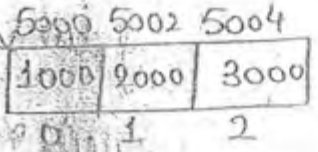
fp = functional pointer pointing to function that returns integer

1000
aaa()
{ printf("aaa"); }

2000
bbb()
{ printf("bbb"); }

3000
ccc()
{ printf("ccc"); }

Exp main()
{
int (*ptr[3])();
ptr[0] = aaa;
ptr[1] = bbb;
ptr[2] = ccc;
(*ptr[2])();
}



O/P = ccc

CHAPTER NO. 3 (Parsing)

DH.1 int (*f)(int, int)

Meaning:- A pointer to a function that takes two integers as I/P and returns O/P as integer.

DH. What does the following 'C' statement declare:-

char *(*(*a[N])())()

int (*)() ⇒ pointer to a function that returns integer.

⇒ Array of N pointers, pointing to function, that returns pointer to a function returning pointing to character or character pointer.

DH What does the following C-statement will do-

void (*abc)(int, void (*def)())

Sol"

abc is a pointer to a function, which will return void and which will take two parameter i-

(1) integer parameter

(2) A pointer def to a function, which will return void.