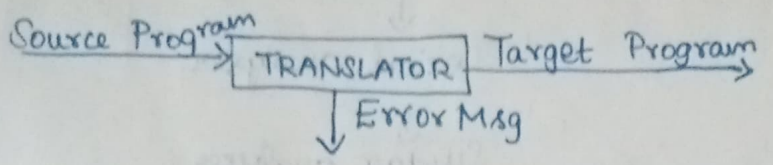
23.01.2024

UNIT-1

Translator:



Types:

i) Assembler. (Translate Assembly lang. code to HC)

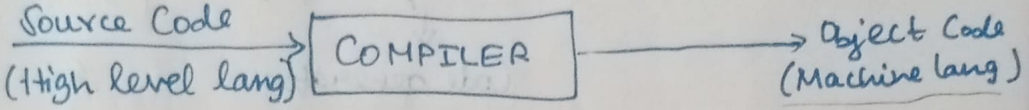
ii) Compiler.

iii) Interpreter.

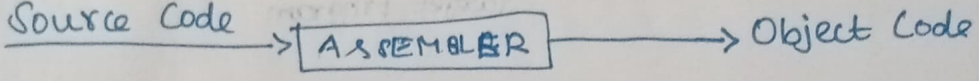
#i include<stdio.h>
Preprocessor
Macro/ Directive

Relocatable Machine Code.
Eg: a.c specific only for a.exe.

Compiler:



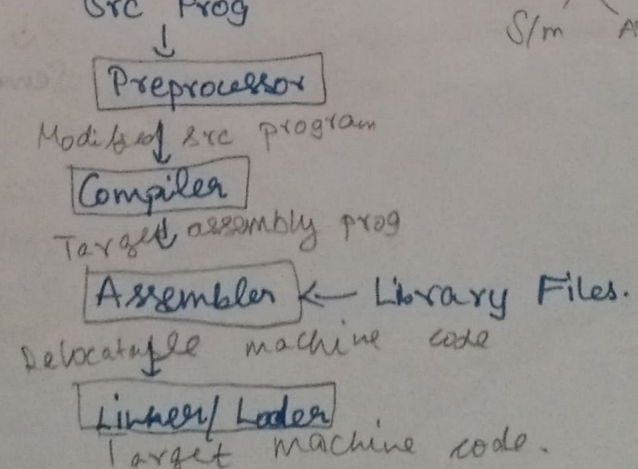
Assembler:



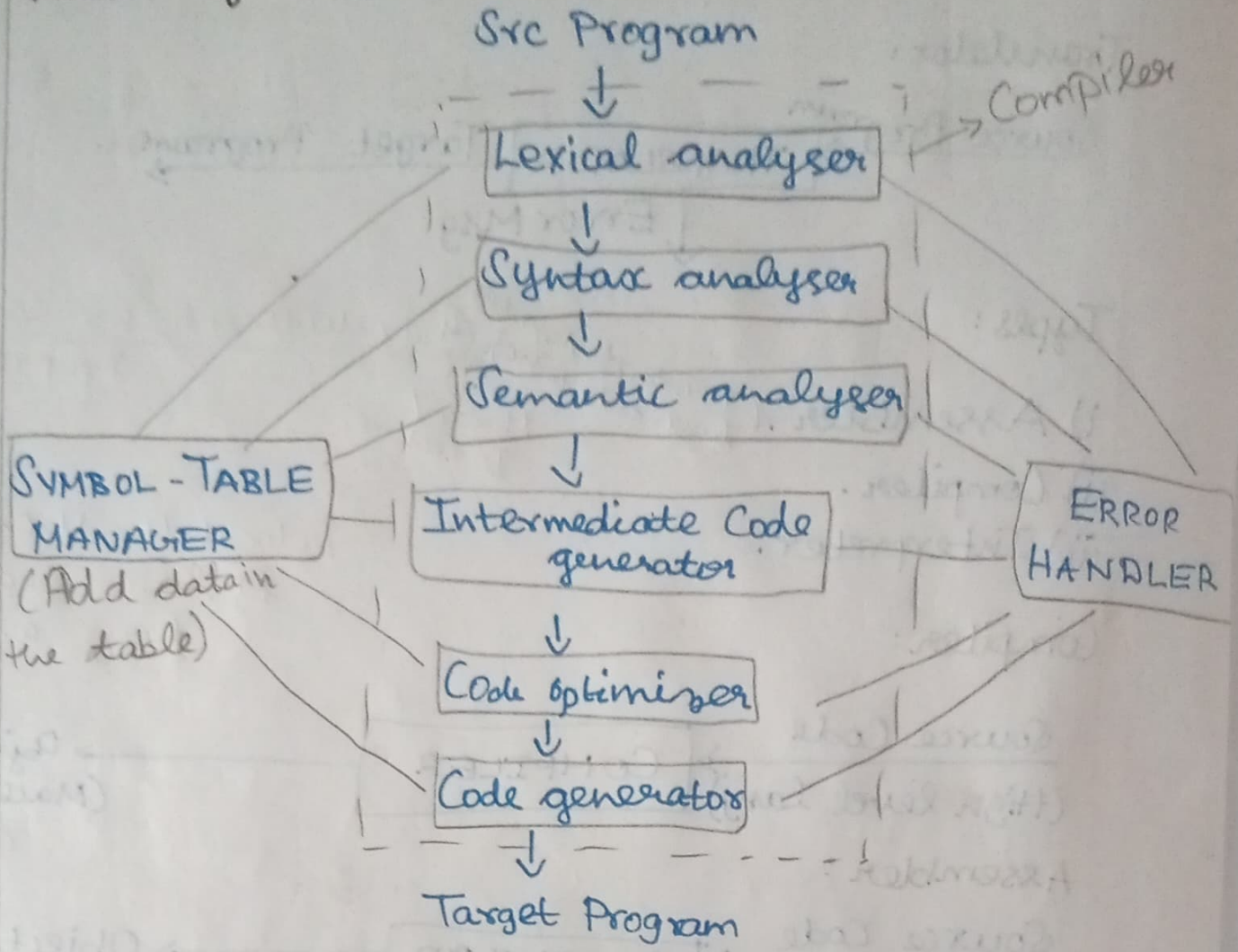
Compiler	Interpreter
→ Translate the compiler src program in a single line.	→ Translate the source program line by line.
→ Faster.	→ Slower.
→ Less time.	→ More time.
→ More efficient	→ Less efficient.
→ Larger in size	→ Smaller than compilers.

Language Processor:

S/W
S/m Appln



Phases of Compiler:



Phases of Compiler

result = Im + em * 50

↓

Lexical Analyser Scanning Opn.

TOKENS: Group of lexemes

id1 = id2 + id3 * 50

id1 <reop1> id2 <aop1> id3 <aop3> 50

O/P - TOKENS — Id, OP, CONST, keywords

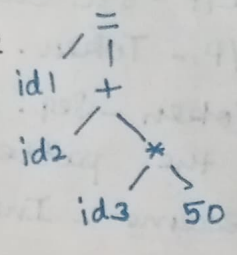
↓

Ex: JS is compiled by Browser - Parsing

Syntax Analyser (Parser).

↓

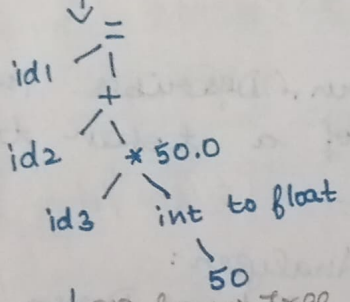
O/P: Syntax Tree



↓

Semantic Analyser (Typecasting)

↓



↓ O/P: Syntax Tree

Intermediate Code Generator (Code Readability)

METHODOLOGIES:

- Three Addr Code
- Postfix notation
- Syntax Tree.

$t_1 = int\ to\ float(50)$ $id1 = t_3$

$t_2 = id3 * t1$

$t_3 = id2 + t2$

↓ o/p: Three Addr Code.

Code Optimization (Reducing the lines of code) (To run faster)

METHODOLOGIES:

- Removing Common Sub-Expression
- Removing Duplicate
- Dead Code Elimination
- Reduce temp variables
- Loop unrolling (Reduce iteration)
- Remove temp. variable

$t1 = id3 * int\ to\ float(50)$

$id1 = id2 + t1$

↓ O/P: Three Addr Code.

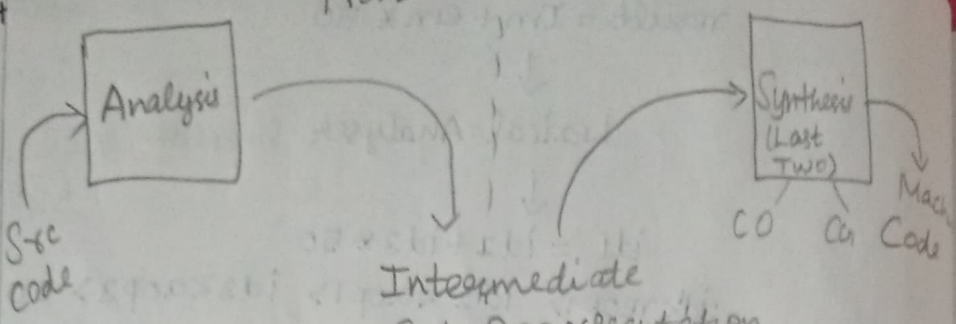
Code Generation (Contains Assembler)

↓

```
LDF   R1, 50.0   (Load Floating pt)
MOV   R2, id3
MULF  R1, R2
MOV   R2, id2
ADDF  R1, R2
STAF  id1, R1
```

30.01.2024

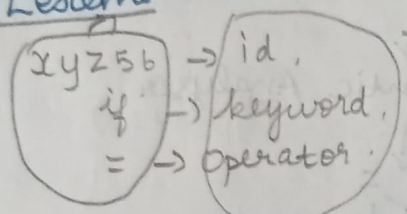
Phases of Compiler:



Lexical Analyser:

* I/P - Src Code, Stream of characters.
* O/P - Token.
* Token - Seq. of characters which matches with the pattern.
* Lexeme - Instance of a token. (ie) Grp of characters forming a token

Eg:



* Pattern - Describes the rule that the lexemes of a token taken.

Eg: (letter | letter/digit)* —) Identifier

Syntax Analyser:

* Also called as Parser.
* I/P - Tokens.
* O/P - Parse Tree.

Syntax Tree: Root node operators

Semantic Analyser:

* Meaning
* Declarations and Type Checking.

Symbol Table:

* Stores all the information about identifiers
* A DS containing a record for each identifier.
* When identifier detected, it is stored in the symbol.

Error Handler:

* Lexical error - Misspelled / Incorrect name.
* Syntactical error - Missing semi-colon / Unbalanced Parenthesis.

Error-Recovery Strategies:

* Panic mode.
* Statement Level.
*
*

Semantical Error:
Type Mismatch, Multiple declaration of variable, Argument mismatch.

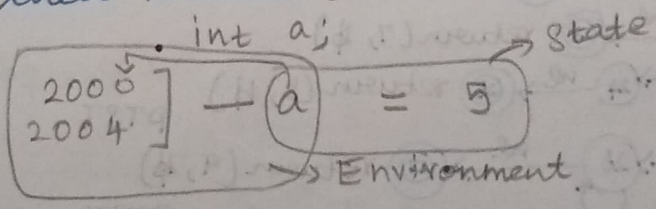
Compiler Construction Tools

- Parser generators (Syntax Analyser) — I/P: Grammar of prog. Lang; O/P: Parse Tree
- Scanner generators — I/P: Regular expr describing tokens; O/P: (Lexical Analyser)
- Syntax-directed translation engines (Semantic Analyzer) — O/P: Intermediate Code
- Automatic Code Generators — Intermediate language; O/P: Machine language
- Data Flow Analysis Engines (Code Optimization)

Programming Language Basics

Static & Dynamic Distinction:
→ Static (fixed) - Event occur at compile time.
→ Dynamic (change) - Event occur at run time.

Environment & States:
Eg: int a; ← Environment is mapping from names to locations in the store.
(ie) Mapping from names to variables.
State is mapping from locations in store to their values. (changes)



Call by Ref, Call by Value
Local Var; Global Var:

31.01.2024

Lexical Analyzer:
Need and Role:

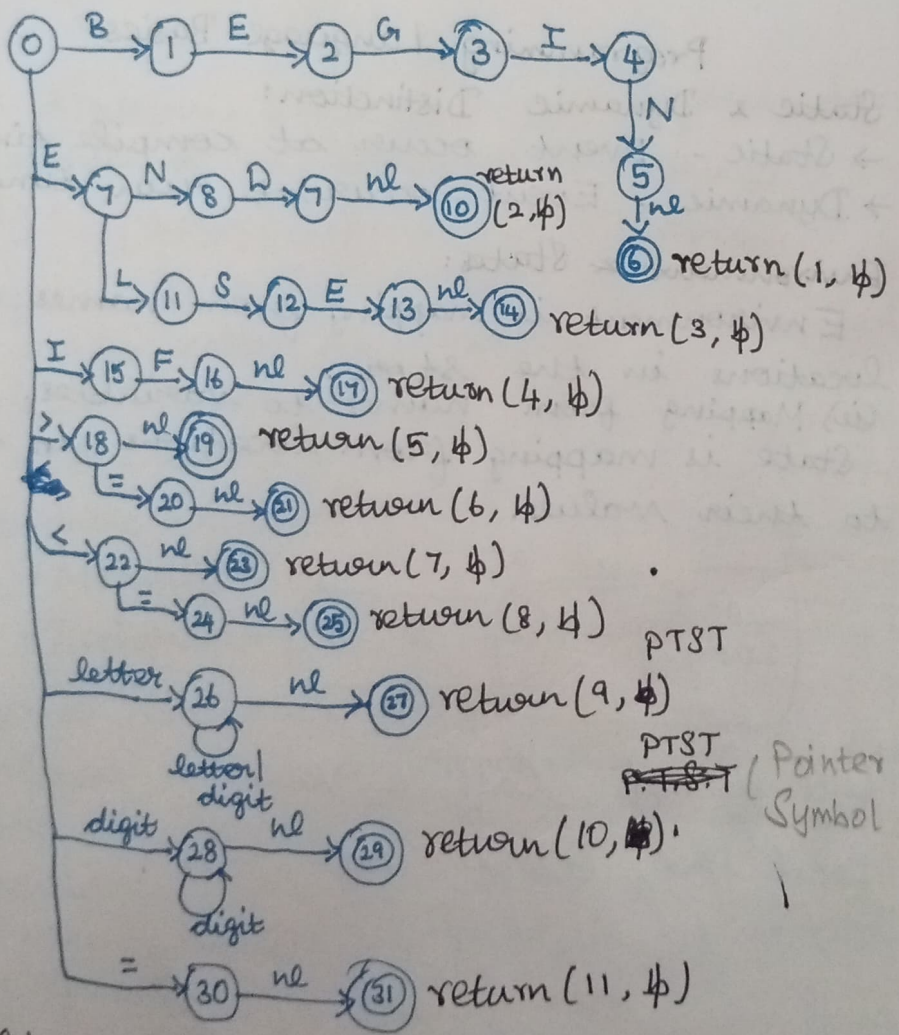
1. Token formation.
2. Stripping comments / whitespaces.
3. Error messages.

Tokens:

Token id	Value

Keywords	Operators	Id	Constant
BEGIN	<	letter(	digit+
END	<=	letter|	
ELSE	>	digit)*	
IF	>=		
	=		

Transition Diagram:



PTST (Pointer To Symbol Table)

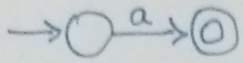
Eg: a then
Else End

Finite Automata

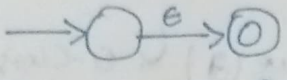
NFA → DFA → Minimized DFA → Recognizing of Token

NFA

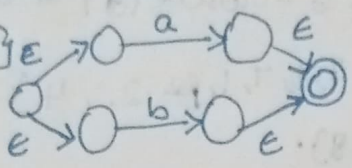
1. a



2. ϵ

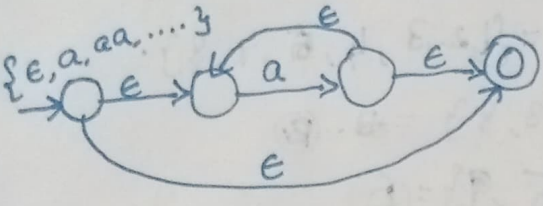


3. a|b = {a,b}



a = {ϵ, a, ϵ} = {a}.
b = {ϵ, b, ϵ} = {b}.

4. a* = {ϵ, a, aa, ...}



06.02.2024

1. (a|b)* abb = {abb, aabb, babb, aaabb, bbabb, ...}.

NFA:

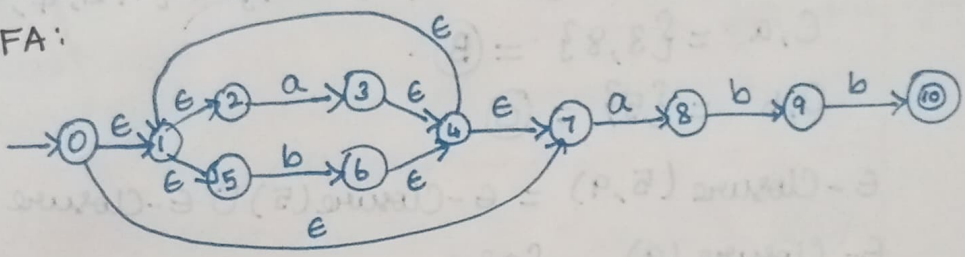
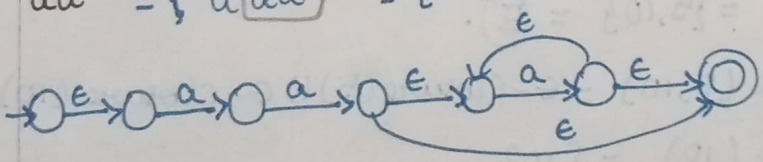


abb = 0-7-8-9-10

bbabb = 0-1-5-6-4-1-5-6-4-7-8-9-10

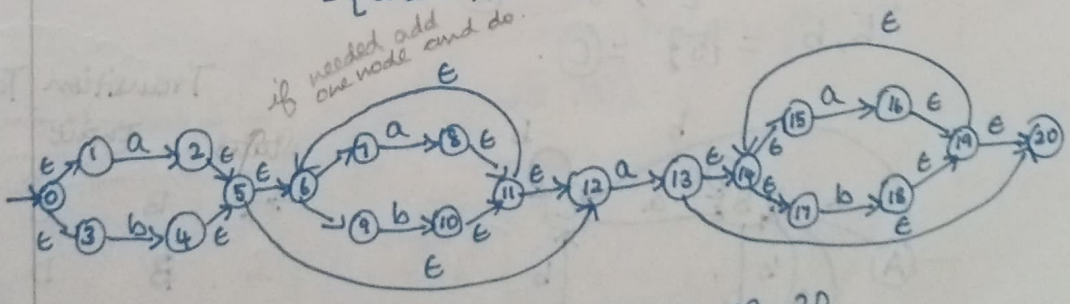
2. aa^+ = { a $\overbrace{aa^*}^{a^+}$ } = {aa, aaa, ...}.



3. $(a|b)^+ a (a|b)^*$ = (a|b)(a|b)* a (a|b)*

= {aaa, aa, ba, bab, ...}

if needed add one node and do



aaa = 0-1-2-5-12-13-14-15-16-19-20

DFA:

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

ϵ-Closure (0) = {0, 1, 2, 4, 7}. — (A).

A, a = {3, 8}. — (B).

A, b = {5}. — (C).

ϵ-Closure (3, 8) = ϵ-Closure (3) $\cup$ ϵ-Closure (8)

ϵ-Closure (3) = {3, 6, 7, 1, 2, 4}.

ϵ-Closure (8) = {8}.

ϵ-Closure (3, 8) = {1, 2, 3, 4, 6, 7, 8}.

B, a = {3, 8} = (B).

B, b = {5, 9} = (D).

ϵ-Closure (5) = {5, 6, 1, 7, 2, 4} = {1, 2, 4, 5, 6, 7}.

C, a = {3, 8} = (B).

C, b = {5} = (C).

ϵ-Closure (5, 9) = ϵ-Closure (5) $\cup$ ϵ-Closure (9).

ϵ-Closure (9) = {9}.

ϵ-Closure (5, 9) = {1, 2, 4, 5, 6, 7, 9}.

D, a = {3, 8} = (B).

D, b = {5, 10} = (E).

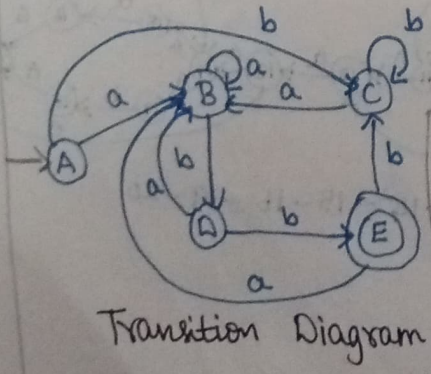
ϵ-Closure (5, 10) = ϵ-Closure (5) $\cup$ ϵ-Closure (10).

ϵ-Closure (10) = {10}.

ϵ-Closure (5, 10) = {1, 2, 4, 5, 6, 7, 10}.

E, a = {3, 8} = (B).

E, b = {5} = (C).



Transition Diagram

Eg.

abb

A-B-D-E

Transition Table

States	Input a	Input b
→A	B	C
B	B	D
C	B	C
D	B	E
*E	B	C

07.02.2024

Minimized DFA

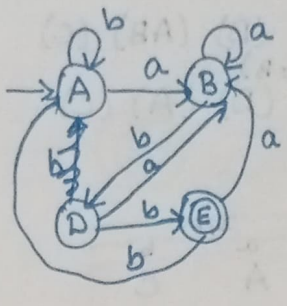
P_0 = (E) (ABCD). // Final & Non-final

P_1 = (E) (AC) (BD).

P_2 = (E) (AC) (B) (D).

P_3 = (E) (B) (D) (A). (∵ A = C)

States	I/P a	I/P b
→A	B	A
B	B	D
D	B	AE
*E	B	A



ABCD
⇓a
BBBB

ABCD
⇓b
CACE

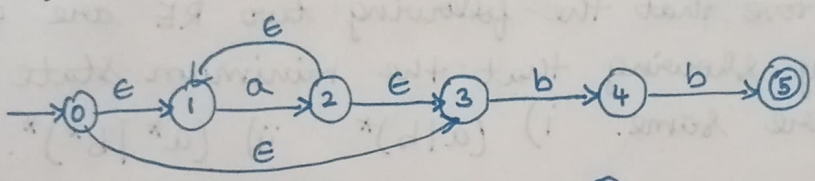
BD
⇓a
BB

BD
⇓b
DE

AC
⇓a
BB

AC
⇓b
CC
=
Same so A = C.

Construct a minimized DFA for the given Regular Expression a* bb.



ϵ-Closure (0) = {0, 1, 3}. — (A).

A, a = {2} — (B).

A, b = {4, 5} — (C).

ϵ-Closure (2) = {2, 1, 3} = {1, 2, 3}. — (B).

B, a = {2} — (B).

B, b = {4, 5} — (C).

~~ϵ-Closure (4, 5) = ϵ-Closure (4) ∪ ϵ-Closure (5).~~

ϵ-Closure (4) = {4}. — (E). (C).

ϵ-Closure (5) = {5} — (D)

~~ϵ-Closure (4, 5) = {4, 5}~~

C, a = { }

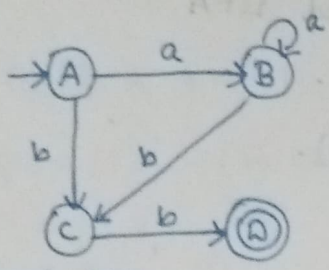
C, b = {5} — (D).

D, a = { }.

D, b = { }.

States	I/P a	I/P b
A	B	C
B	B	C
C	–	D
D	–	–

DFA:



Minimized DFA:

$P_0 = (Final)(Non-Final)$

$P_0 = (D)\ (ABC)$

$P_1 = (D)\ (AB)\ (C)$

$(\because A = B)$

$P_2 = (D)\ (A)\ (C)$

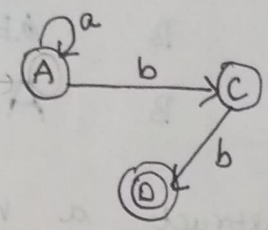
ABC
⇓a
BBϕ

ABC
⇓b
CCD

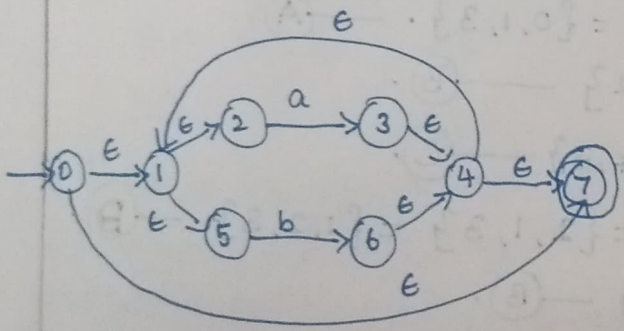
AB
⇓a
BB

AB
⇓b
CC

States	I/P a	I/P b
→A	A	C
C	-	D
*D	-	-



Prove that the following two RE are equivalent by showing that the minimum state DFA's are same. i) $(a|b)^*$ ii) $(a^*|b^*)^*$.



ϵ-Closure (0) = {0, 1, 2, 5, 7} — (A).

A, a = {3} — (B).

A, b = {6} — (C).

ϵ-Closure (3) = {3, 4, 1, 7, 2, 5} = {1, 2, 3, 4, 5, 7}
— (B).

B, a = {6} — (D).

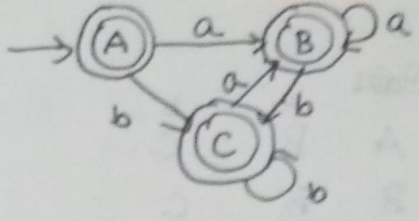
B, b = {6} — (C).

ϵ-Closure (6) = {6, 4, 7, 1, 2, 5} = {1, 2, 4, 5, 6, 7} — (C).

C, a = {3} — (B).

C, b = {6} — (C).

States	a	b
A	B	C
B	B	C
C	B	C



Minimized DFA:

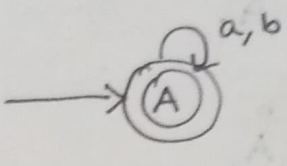
$P_0 = (ABC)$

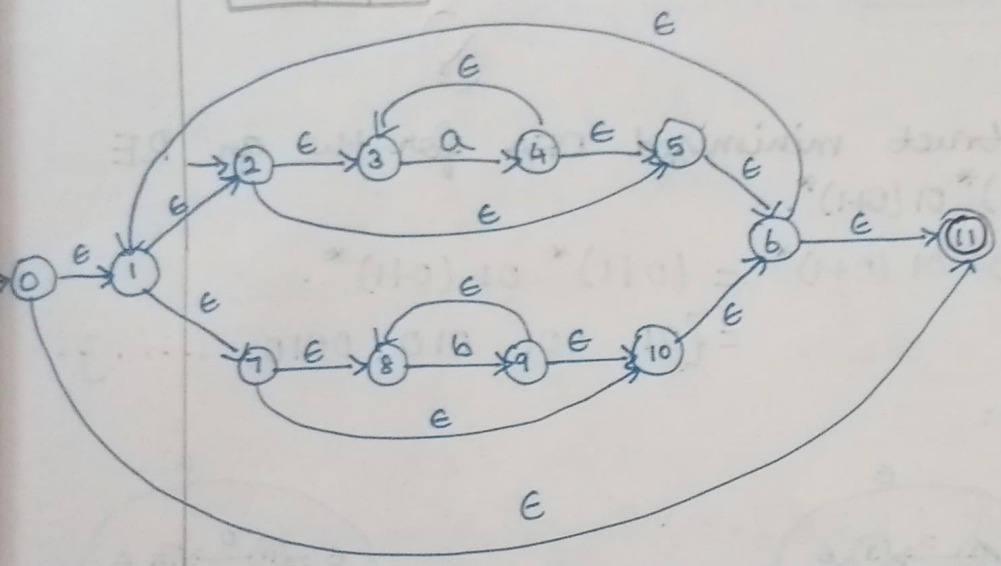
$(\because A = B, A = C)$.

Replace B, C by A.

$P_1 = A$.

States	a	b
*A	A	A





ϵ-Closure (0) = {0, 1, 2, 7, 3, 8, 10, 11, 5}.

= {0, 1, 2, 3, 5, 7, 8, 10, 11} — (A).

A, a = {4} — (B).

A, b = {9} — (C).

ϵ-Closure (4) = {4, 5, 3, 6, 1, 11, 2, 7, 8, 10, 5}.

= {1, 2, 3, 4, 5, 6, 7, 8, 10, 11} — (B).

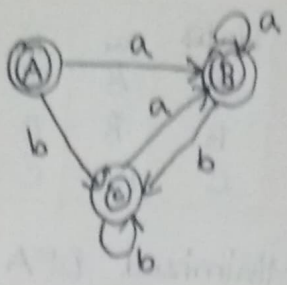
B, a = {4} — (B), B, b = {9} — (C).

ϵ-Closure (9) = {9, 8, 10, 6, 11, 1, 2, 7, 3, 5}.

= {1, 2, 3, 5, 6, 7, 8, 9, 10, 11} — (C).

C, a = {4} — (B), C, b = {9} — (C).

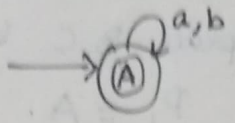
States	a	b
A	B	C
B	B	C
C	B	C



Minimized DFA:

$P_0 = (ABC)$

$P_1 = A.$



State	a	b
→*A	A	A

(N)

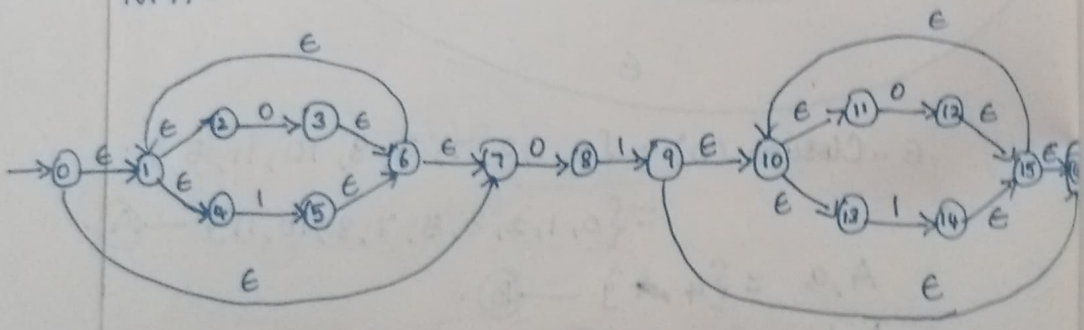
13.02.2024

1. Construct minimized DFA for the gn RE $(0+1)^* 01 (0+1)^*$.

$(0+1)^* 01 (0+1)^* = (011)^* 01 (011)^*.$

take '1' $= \{01, 001, 010, 0010, \ldots\}.$

NFA:



ϵ-Closure (0) = {0, 1, 7, 2, 4} = {0, 1, 2, 4, 7} — (A).

A, a 0 = {3, 8} — (B).

A, 1 = {5} — (C).

ϵ-Closure (3, 8) = ϵ-Closure (3) ∪ ϵ-Closure (8).

ϵ-Closure (3) = {3, 6, 1, 2, 4, 7}

ϵ-Closure (8) = {8}

ϵ-Closure (3, 8) = {1, 2, 3, 4, 6, 7, 8} — (B).

B, 0 = {3, 8} — (B).

B, 1 = {5, 9} — (D).

ϵ-Closure (5) = {5, 6, 1, 2, 4, 7} — (C).

C, 0 = {3, 8} — (B).

C, 1 = {5} — (C).

ϵ-Closure (5,9) = ϵ-Closure(5) $\cup$ ϵ-Closure(9).

ϵ-Closure (9) = {9, 10, 11, 13, 16}. — (D).

ϵ-Closure (5,9) = {1, 2, 4, 5, 6, 7, 9, 10, 11, 13, 16}.
— (D).

D, 0 = {3, 8, 12} — (E).

D, 1 = {5, 14} — (F).

ϵ-Closure (3,8,12) = {1, 2, 3, 4, 6, 7, 8, 12, 15, 16, 10, 11, 13}
— (E).

E, 0 = {3, 8, 12} — (E).

E, 1 = {5, 9, 14} — (F) (G).

ϵ-Closure (5,14) = {1, 2, 4, 5, 6, 7, 14, 15, 10, 11, 13, 16} — (F).

F, 0 = {3, 8, 12} — (E).

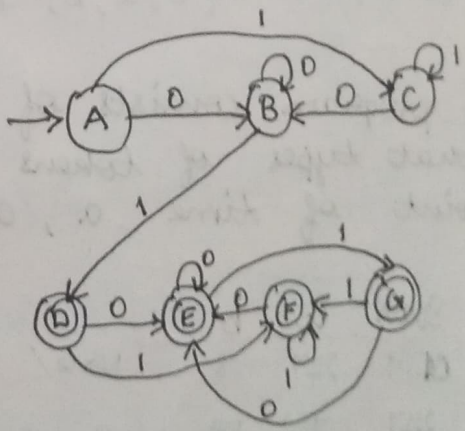
F, 1 = {5, 14} — (F).

ϵ-Closure (5,9,14) = {1, 2, 4, 5, 6, 7, 9, 10, 11, 13, 14, 15, 16}. — (G).

G, 0 = {3, 8, 12} — (E).

G, 1 = {5, 14} — (F).

States	0	1
→A	B	C
B	B	D
C	B	C
*D	E	F
*E	E	G
*F	E	F
*G	E	F



Minimized DFA:

P_0 = (Final) (Non Final)
= (DEFG) (ABC)

DEFG	DEFG	ABC	ABC
⇓0	⇓1	⇓0	⇓1
EEEE	FGFF	BBB	CDC

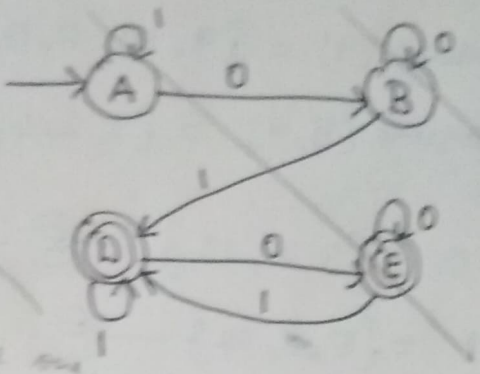
P_1 = (DFG) (E) (AC) (B)

$P_1 = (DFG)(E)(AC)(B)$

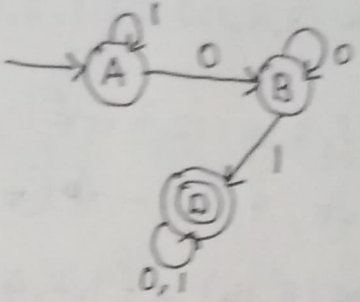
$P_2 = (D)(E)(A)(B)$

$P_3 = (D)(A)(B)$

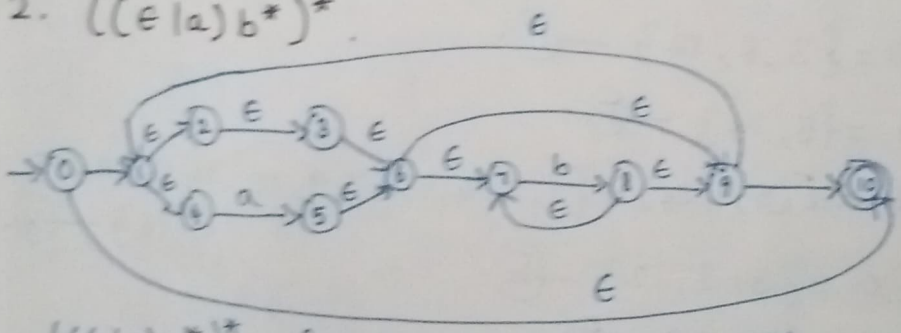
State	0	1
→A	B	A
B	B	D
*D	E	D
*E	E	D



State	0	1
→A	B	A
B	B	D
*D	D	D



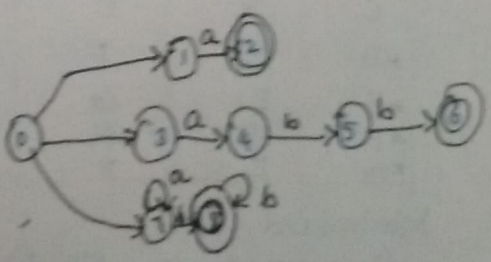
2. $((\epsilon \mid a)b^*)^*$



$((\epsilon \mid a)b^*)^* = \{\epsilon, a, b, ab, abab, abb, \ldots\}$.

A program consists of 3 RE. Check whether what types of tokens are generated at each point of time a, abb, a^*b^+.

State	a	b	
0 1 7	247	8	None
247	7	58	a
* 8	–	8	a^*b^+
7	7	8	None
* 58	–	68	a^*bb^*
68	–	8	abb, a^*b^+



→ abb has higher priority. So neglect a^*bb^*.

14/2/24

Lex Tool:

* It is used to constructing LA → Lexical analyzer
* Automated construction of LA → o/p: Tokens
* lex.l → extension
  → filename

Source program (lex.l) → [Lex compiler] → lex.yy.c (Transition diagram)

lex.yy.c → [C compiler] → object file (a.out)

Stream of characters → [a.out] → sequence of tokens.

Lex Compiler: .. ..

*. It deals with the structure of lex program
* The three components are.

(i) declarations

%. %.

(ii) Translation Rules

%%

(iii) Auxiliary functions (subroutines)

Declarations

* Used for variables, constants and Regular Expression.

Eg:
```
%{
    int a,b;
    int c=5;
%}
```

Eg:
```
%{
    R.E.
%}
```

Eg: For letter & digit

letter [a-z A-Z]

digit [0-9]

id ⇒ {letter}{(letter / digit)*}

numbers ⇒ [(digit)+]

delimiter [\n \t]

Translation Rules:

Pattern 1 { Action 1 }

. . . . . .

Pattern n { Action n }

always return a string token value

Eg:
```
%%
{id}        { printf("%s, is a identifier", yytext); }
if          { printf("%s, is a keyword", yytext); }
else        "   "   "   "   "   "   "   "
"<"         { printf("%s is a operator", yytext); }
number      { printf("%s is a number", yytext); }

{id}        { INSTALL() }
```

Auxiliary Function → user defined functions.

```
int INSTALL(){

}
```

number of words in a

Write a Lex program to count the no. of words in the gn program.

Declaration:

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

Translation Rule.

```
%%
([a-z A-Z 0-9])*   {i++}
"\n"               {printf("%d ",i);i=0}
%%
```

Auxiliary Function:

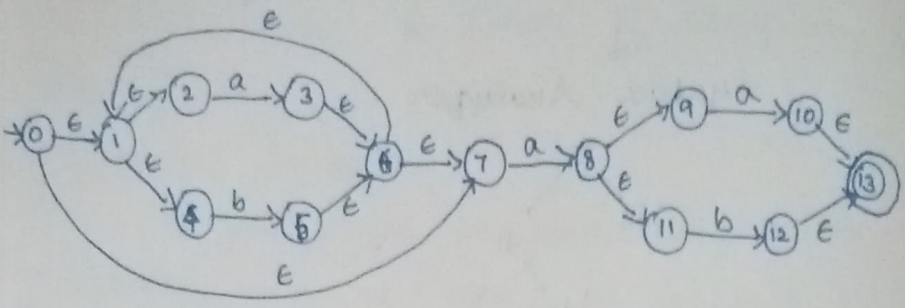
```
int yywrap(){}
int main()
{
  printf("Enter String");
  yylex();
  return 0;
}
```

yylex() returns a value indicating the type of token.

To count no. of vowels in the given sentence.

Minimized DFA:

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



ε-Closure {0} = {0, 1, 2, 4, 7} — (A).

A, a = {3, 8}. — (B).

A, b = {5}. — (C).

ε-Closure {3,8} = ε-Closure {3} ∪ ε-Closure {8}.

ε-Closure {3} = {3, 6, 1, 2, 4, 7}.

ε-Closure {8} = {8, 9, 11}.

ε-Closure {3,8} = {1, 2, 3, 4, 6, 7, 8, 9, 11}. — (B).

B, a = {3, 8, 10}. — (D).

B, b = {5, 12}. — (E).

ε-Closure {5} = {5, 6, 1, 2, 4, 7}. — (C).

C, a = {3, 8}. — (B).

C, b = {5}. — (C).

ε-Closure {3,8,10} = {1,2,3,4,6,7,8,9,11} ∪ {10,13}.

= {1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13}. — (D).

D, a = {3, 8, 10}. — (D).

D, b = {5, 12}. — (E).

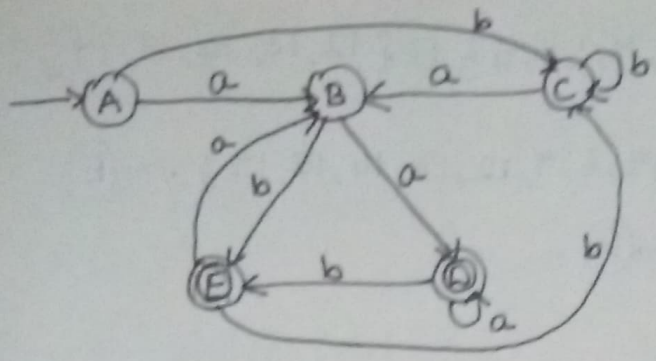
ε-Closure {5,12} = {5, 6, 1, 2, 4, 7} ∪ {12, 13}.

= {1, 2, 4, 5, 6, 7, 12, 13}. — (E).

E, a = {3, 8} — (B).

E, b = {5}. — (C).

States	a	b
→A	B	C
B	D	E
C	B	C
*D	D	E



Minimized DFA:

$P_0 = (ABC)\ (DE)$.

ABC — a → BDB; ABC — b → CEC; AC — a → BB; AC — b → CC

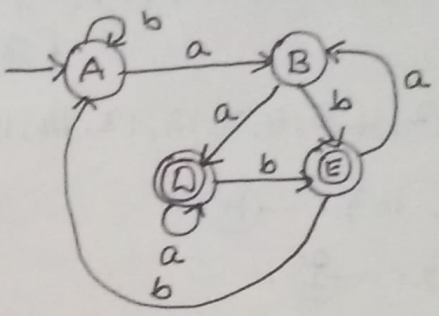
$P_1 = (AC)\ (B)\ (DE)$.

$(\because A = C)$.

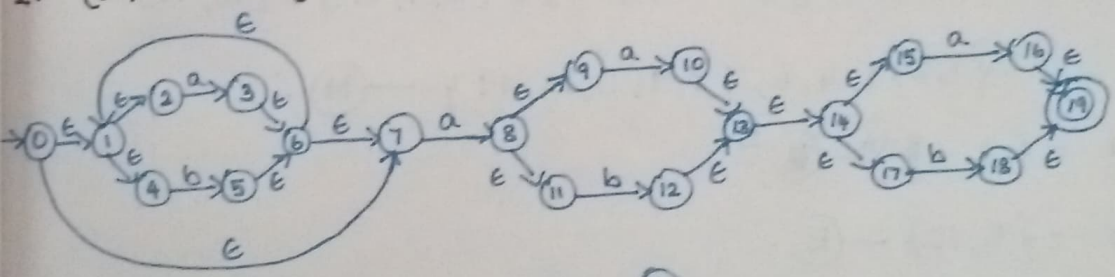
DE — a → DB; DE — b → EC

$P_2 = (A)\ (B)\ (DE)$.

States	a	b
→A	B	A
B	D	E
*D	D	E
*E	B	A



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



ε-Closure {0} = {0, 1, 2, 4, 7} — (A).

A, a = {3, 8}. — (B).

A, b = {5}. — (C).

ε-Closure {3, 8} = {1, 2, 3, 4, 6, 7, 8, 9, 11}. — (B).

B, a = {3, 8, 10}. — (D).

B, b = {5, 12}. — (E).

ε-Closure {5} = {1, 2, 4, 5, 6, 7}. — (C).

C, a = {3, 8}. — (B).

C, b = {5}. — (C).

ε-Closure {3, 8, 10} = {1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17}. — (D)

D, a = {3, 8, 10, 16}. — (F)

D, b = {5, 12, 18}. — (G)

ϵ-Closure $\{5,12\} = \{5,6,7,1,2,4\} \cup \{12,13,14,15,17\}$.

$= \{1,2,4,5,6,7,12,13,14,15,17\}$. — (E).

E, a $= \{3,8,16\}$. — (H).

E, b $= \{5,18\}$. — (I).

ϵ-Closure $\{3,8,10,16\} = \{1,2,3,4,6,7,8,9,10,11,13,14,15,16,19\}$. — (F).

F, a $= \{3,8,10,16\}$. — (F).

F, b $= \{5,12,18\}$. — (G).

ϵ-Closure $\{5,12,18\} = \{5,6,1,2,4,7\} \cup \{12,13,14,15,17\} \cup \{18,19\}$.

$= \{1,2,4,5,6,7,12,13,14,15,17,18,19\}$. — (G).

G, a $= \{3,8,16\}$. — (H).

G, b $= \{5,18\}$. — (I).

ϵ-Closure $\{3,8,16\} = \{3,6,1,2,4,7\} \cup \{8,9,11\} \cup \{16,19\}$.

$= \{1,2,3,4,6,7,8,9,11,16,19\}$. — (H).

H, a $= \{3,8,10\}$ — (B).

H, b $= \{5,12\}$ — (E).

ϵ-Closure $\{5,18\} = \{5,6,1,2,4,7\} \cup \{18,19\}$.

$= \{1,2,4,5,6,7,18,19\}$ — (I).

I, a $= \{3,8\}$ — (B).

I, b $= \{5\}$. — (C).

States	a	b
→A	B	C
B	D	E
C	B	C
D	F	G
E	H	I
*F	F	G
*G	H	I
*H	D	E
*I	B	C

Minimized DFA:

$P_0 = (ABCDE)\ (FGHI)$.

ABCDE	ABCDE	FGHI	FGHI
⇓a	⇓b	⇓a	⇓b
BDBFH	CECGI	FHDB	GIEC

$P_1 = (AC)\ (BDE)\ (FGHI)$.

$(\because A == C)$

BDE	BDE
⇓a	⇓b
DFH	EGI

$P_2 = (A)\ (B)(D)\ (E)\ (F)(G)(H)(I)$

States	a	b
→A	B	A
B	D	E
D	F	G
E	H	I
F	F	G
G	H	I
H	D	E
I	B	A