CS 419 Compiler

Project Form

Project Idea:

……Project 2……..…….

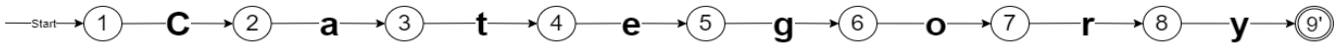
Team Members NO#: ……..

ID	Name	Level& Department	Section(Day -from-to)	Role (Lead/Member)	Grade
201900548	فادى ملاك عطيه	3 CS	Wednesday From 4 to 6	Lead	
201900603	مارينا رومانى نصر	3 CS	Wednesday From 4 to 6	Member	
201900881	مينا فوزى فايز	3 CS	Wednesday From 4 to 6	Member	
201900972	يوستينا اشرف خليل	3 CS		Member	
201900509	عمر خالد حسن محمد	3 CS	Thursday from 8 to 10	Member	
201900199	اياد ايمن محمد	3 CS		Member	
201900179	أمير حنا ثابت فهيم	3 CS		Member	

Regular Expression, Finite automata and Conversion from RegX to NFA, NFA to DFA

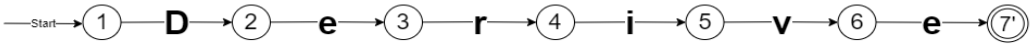
RE: Category

NFA & DFA



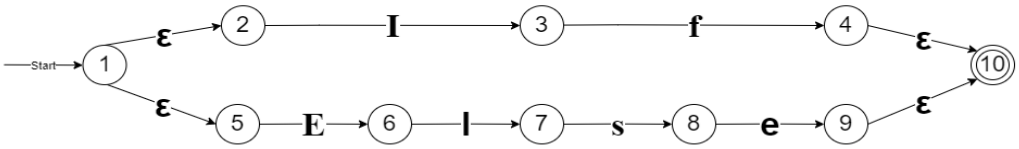
RE: Derive

NFA & DFA

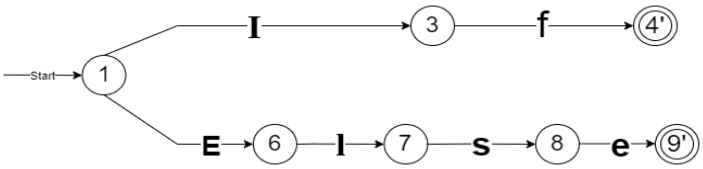


RE: (If | Else)

NFA

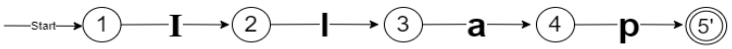


DFA



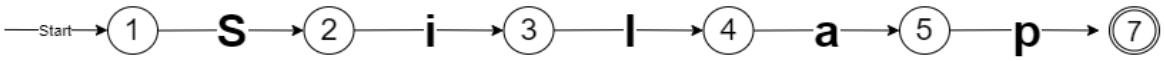
RE: ILap

NFA & DFA



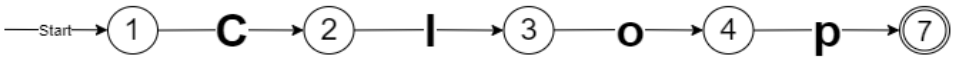
RE: Silap

NFA & DFA



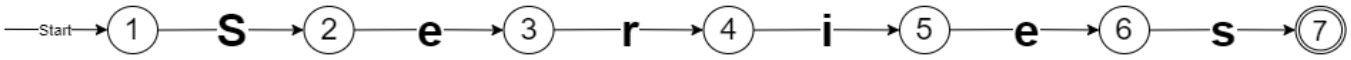
RE: Clop

NFA & DFA



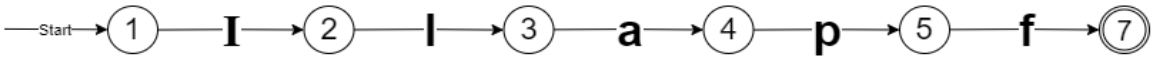
RE: Series

NFA & DFA



RE: Ilapf

NFA & DFA

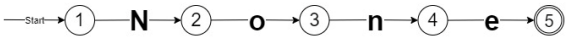


RE: Silapf

NFA & DFA

RE: None

NFA & DFA



RE: Logical

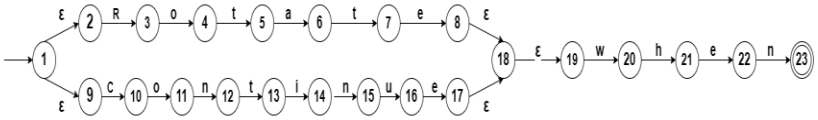
NFA & DFA

RE: terminatethis

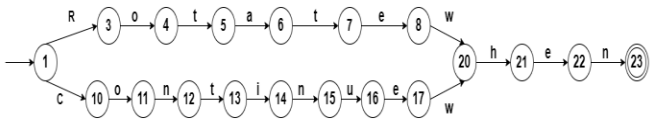
NFA & DFA

Keyword: Rotatewhen/Continuewhen

NFA

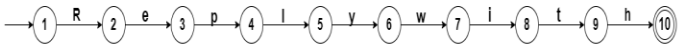


DFA



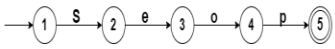
Keyword: Replywith

NFA & DFA



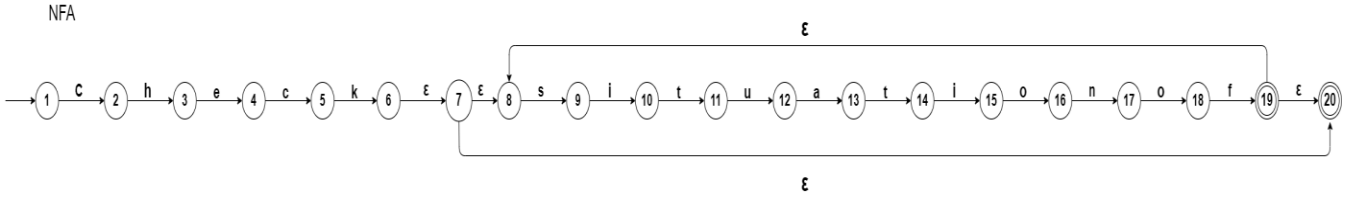
Keyword: Seop

NFA & DFA

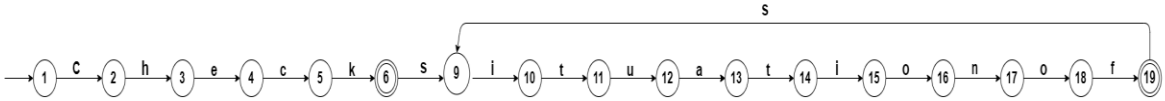


Keyword: Check –situationof

NFA

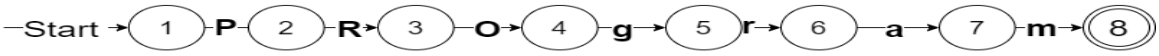


DFA



RE : Program

NFA & DFA :

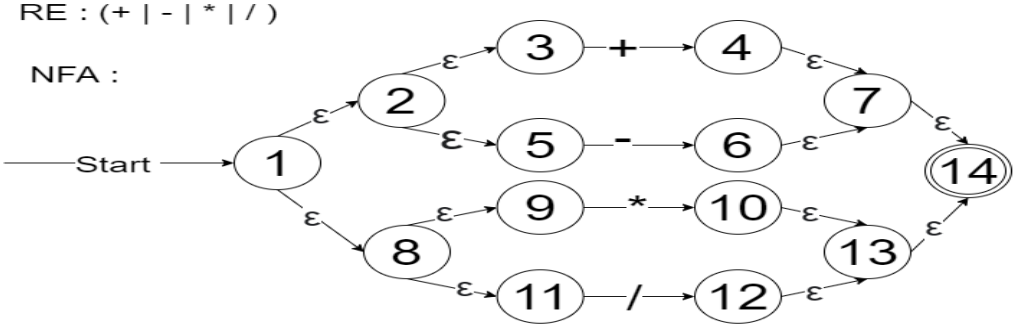


RE : End

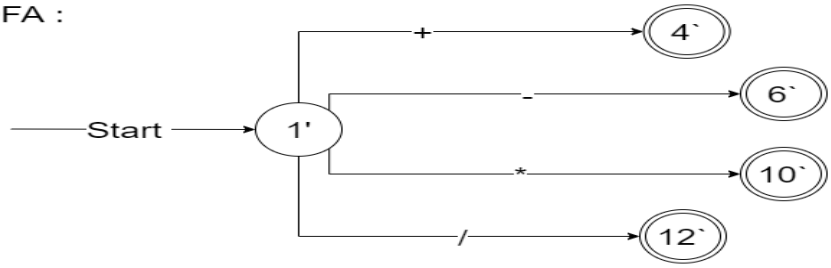
NFA & DFA :

RE : (+ | - | * | /)

NFA :

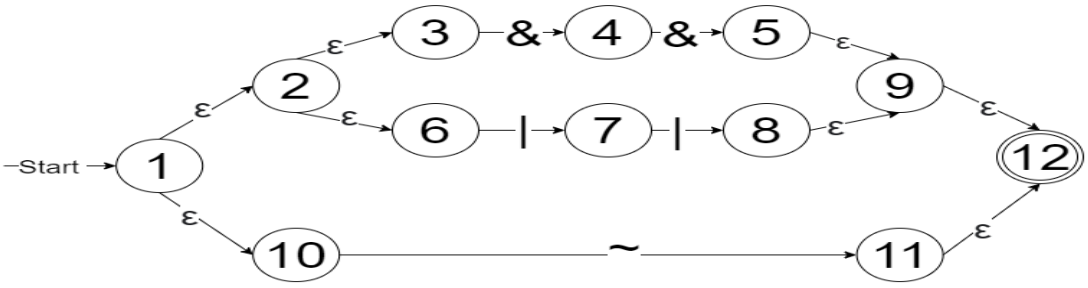


DFA :

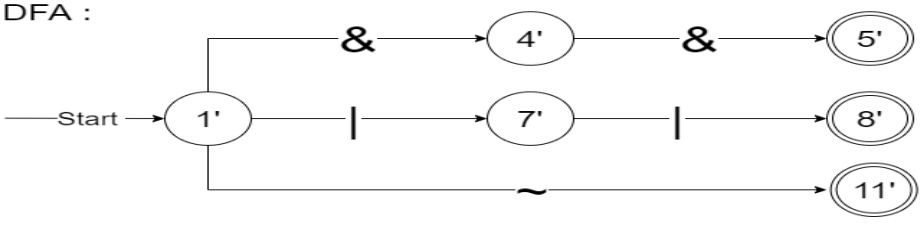


RE : (&& | || | ~)

NFA :



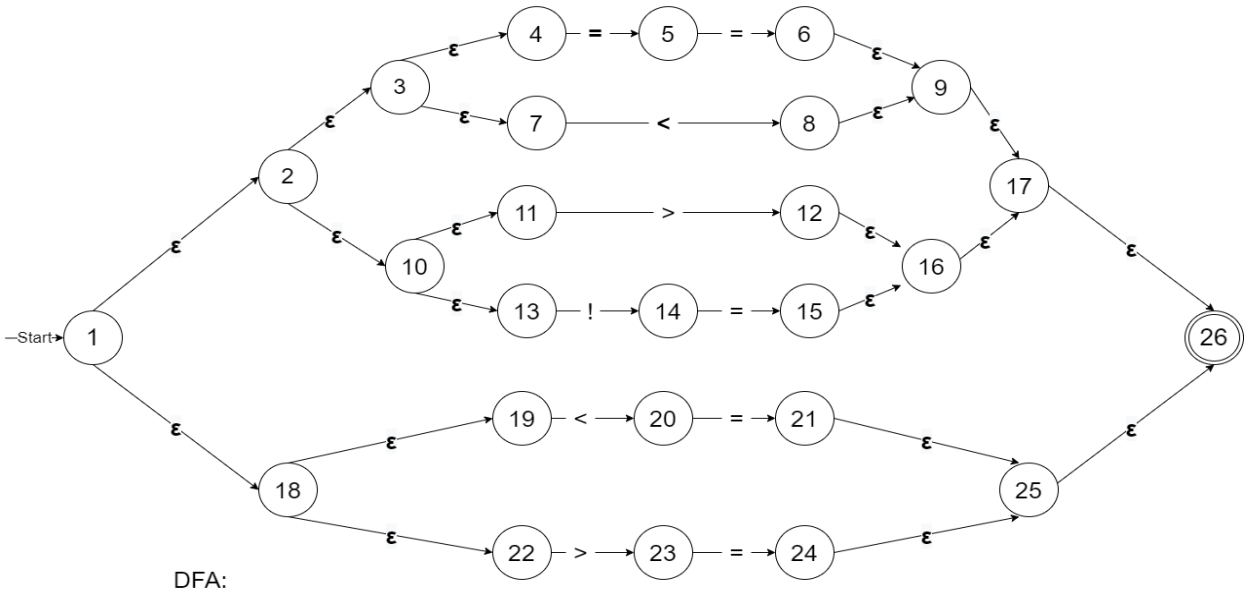
DFA :



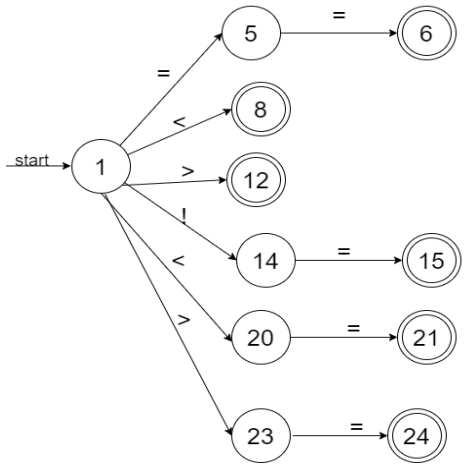
1) Keyword: (==,<,>,!=,<=,>=)

Regular Expression: (== | < | > | != | <= | >=)

NFA:



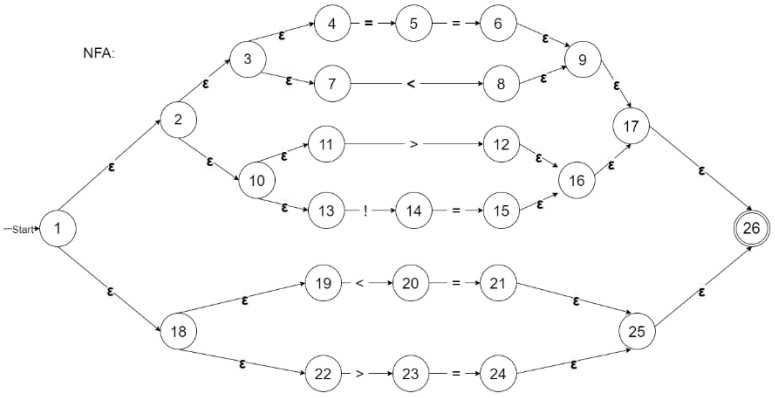
DFA:



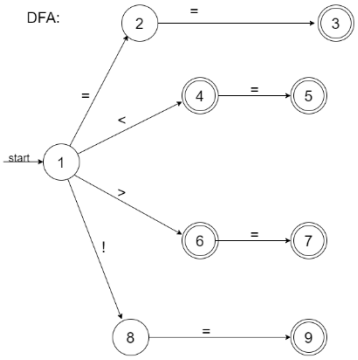
1) Keyword: (==,<,>,!=,<=,>=)

Regular Expression: (== | < | > | != | <= | >=)

NFA:



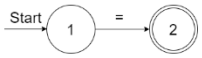
DFA:



Keyword: =

Regular Expression: =

NFA and DFA:



Keyword: .

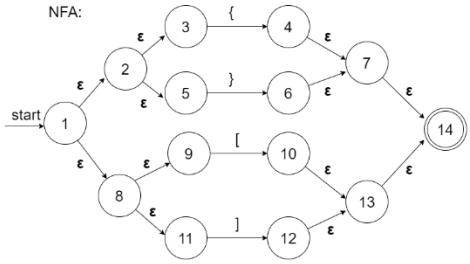
Regular Expression: .

NFA and DFA:

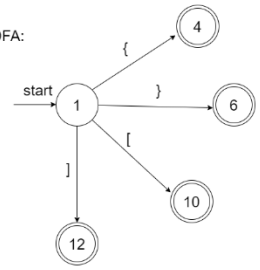
Keyword: {,},[,]

Regular Expression: ({ | } | [|])

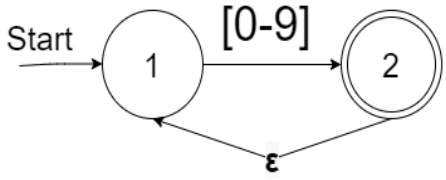
NFA:



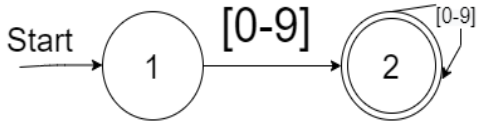
DFA:



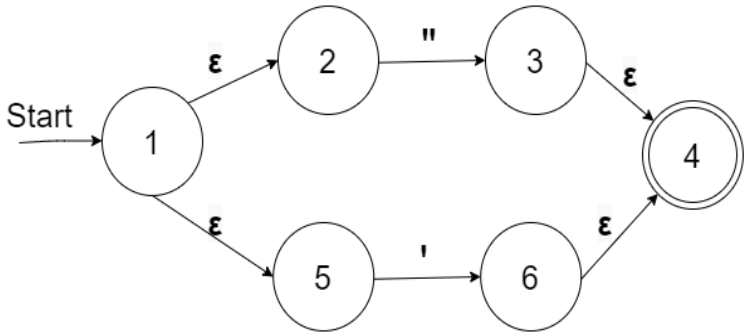
NFA



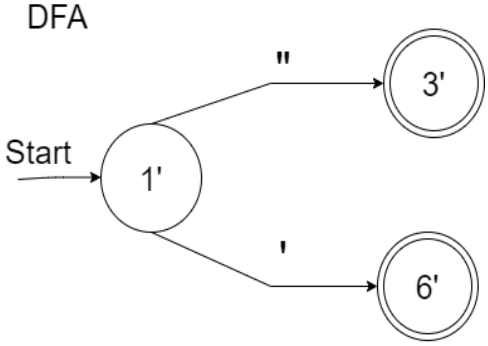
DFA



NFA

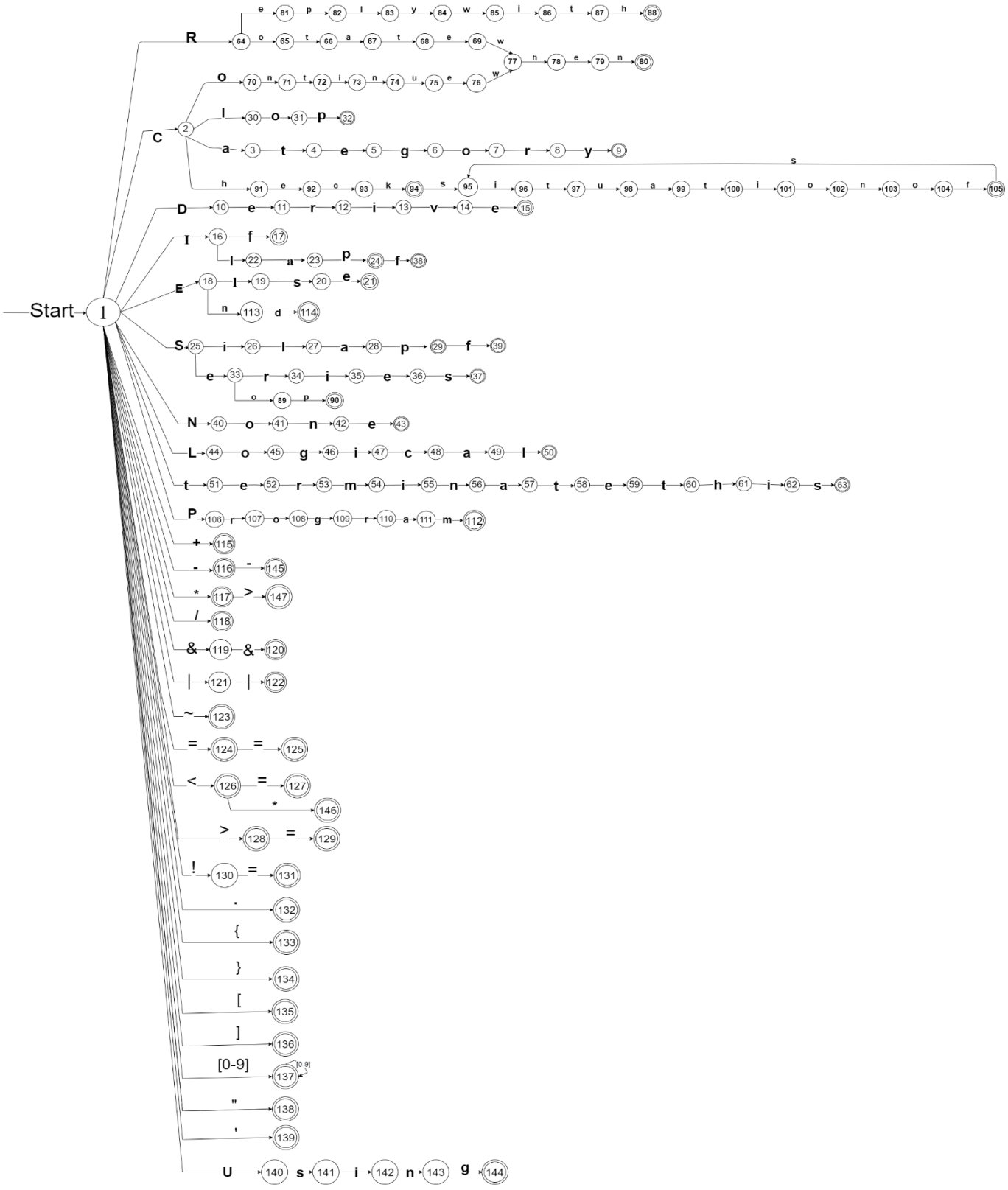


DFA



NFA&DFA

Total DFA :


Start
1
R
64 o 65 t 66 a 67 t 68 e 69 w 77 h 78 e 79 n 80
e 81 p 82 l 83 y 84 w 85 l 86 t 87 h 88
o 70 n 71 t 72 i 73 n 74 u 75 e 76 w
C
2
l 30 o 31 p 32
a 3 t 4 e 5 g 6 o 7 r 8 y 9
h 91 e 92 c 93 k 94 s 95 i 96 t 97 u 98 a 99 t 100 i 101 o 102 n 103 o 104 f 105 s
D 10 e 11 r 12 i 13 v 14 e 15
I 16 f 17
l 22 a 23 p 24 f 38
E 18 l 19 s 20 e 21
n 113 d 114
S 25 i 26 l 27 a 28 p 29 f 39
e 33 r 34 i 35 e 36 s 37
o 89 p 90
N 40 o 41 n 42 e 43
L 44 o 45 g 46 i 47 c 48 a 49 l 50
t 51 e 52 r 53 m 54 i 55 n 56 a 57 t 58 e 59 t 60 h 61 i 62 s 63
P 106 r 107 o 108 g 109 r 110 a 111 m 112
+ 115
- 116 - 145
* 117 > 147
/ 118
& 119 & 120
| 121 | 122
~ 123
= 124 = 125
< 126 = 127
* 146
> 128 = 129
! 130 = 131
. 132
{ 133
} 134
[135
] 136
[0-9] 137 [0-9]
" 138
' 139
U 140 s 141 i 142 n 143 g 144

1.

Program → Program ClassDeclaration   End

Remove left-recursion

Program → Program`

Program` → ClassDeclaration   End Program` | **ε**

First(Program) →{Category , **ε** }

First(Program`) →{Category , **ε** }

2.

ClassDeclaration→ Category ID{ Class_Implementation} |
Category ID Derive { Class_Implementation}

First(ClassDeclaration) →{Category}

3.

Class_Implementation→ VarDeclaration
Class_Implementation| MethodDeclaration
Class_Implementation | Comment Class_Implementation |
using_command Class_Implementation| Func _Call
Class_Implementation |empty

First(Class_Implementation) →{ First(VarDeclaration),
First(MethodDeclaration), First(Comment)
First(using_command) , First(Func _Call), **ε**}

First(Class_Implementation) →

{ Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , **< , - , using , ID ,ε** }

4.

MethodDeclaration→ Func Decl ;| Func Decl { VarDeclaration Statements }

First(MethodDeclaration) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical}

5.

Func Decl →Type ID (ParameterList)

First(Func Decl) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical}

6.

Type → Ilap | Silap | Clop | Series | Ilapf | Silapf | None | Logical

First(Type) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical}

7.

ParameterList →empty | None | Non-Empty List

First(ParameterList) → {None , **ε** , Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical }

8.

Non-Empty List→ Type ID | Non-Empty List , Type ID

Remove left-recursion

Non-Empty List→ Type ID Non-Empty List`

Non-Empty List` →, Type ID Non-Empty List` | **ε**

First(Non-Empty List) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical }

First(Non-Empty List`) → { , , **ε** }

9.

VarDeclaration→ empty | Type ID_List ; VarDeclaration

First(VarDeclaration) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , **ε** }

10.

ID_List →ID | ID_List , ID

Remove left-recursion

ID_List →ID ID_List`

ID_List` →, ID ID_List` | **ε**

First(ID_List) → {ID}

First(ID_List`) → {, , **ε** }

11.

Statements→empty | Statement Statements

First(Statements) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write ,**ε** }

12.

Statement→Assignment | If _Statement | Rotatewhen _Statement | Continuewhen_Statement | Replywith _ Statement | terminatethis _Statement|read (ID); | write (Expression); |

First(Statement)→ {Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write , **ε** }

13.

Assignment→ VarDeclaration = Expression;

First(Assignment)→{ Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , **ε** }

14.

Func _Call → ID (Argument_List) ;

First(Func _Call) → {ID}

15.

Argument_List →empty | NonEmpty_Argument_List

First(Argument_List) → {**ε** , ID , Number}

16.

NonEmpty_Argument_List →Expression |
NonEmpty_Argument_List , Expression

Remove left-recursion

NonEmpty_Argument_List → Expression NonEmpty_Argument_List`

NonEmpty_Argument_List` → , Expression NonEmpty_Argument_List` | **ε**

First(NonEmpty_Argument_List) → {First(Expression)}

= First(NonEmpty_Argument_List) → {ID , Number}

First(NonEmpty_Argument_List`) → {, , ε}

17.

Block Statements→{ statements }

First(Block Statements)→{ { }

18.

If _Statement→ if (Condition _Expression) Block Statements

First(If _Statement) → {if}

19.

Condition _Expression→ Condition |Condition Condition _Op Condition

First(Condition _Expression) → {First(Condition)}

= First(Condition _Expression) → { ID , Number}

20.

Condition _Op → and | or

First(Condition _Op) → {and , or}

21.

Condition→ Expression Comparison _Op Expression //ambiguous grammar

First(Condition) → {First(Expression)}

= First(Condition) → {ID , Number}

22.

Comparison _Op → == | != | > | >= | < | <=

First(Comparison _Op) → { =, ! , > , < }

23.

Rotate _Statement → Rotate when(Condition _Expression) Block Statements

First(Rotate _Statement) → {Rotate}

24.

Continuewhen _Statement →Continuewhen (expression ; expression ; expression) Block Statements

First(Continuewhen _Statement) →{Continuewhen}

25.

Replywith _Statement→ Replywith Expression ; | returnID ;

First(Replywith _Statement)→ {Replywith, returnID}

26.

terminatethis _Statement→ terminatethis;

First(terminatethis _Statement)→ {terminatethis}

27.

Expression → Term |Expression Add_Op Term

Remove left-recursion

Expression → Term Expression`

Expression` → Add_Op Term Expression` | **ε**

First(Expression) → {First(Term)}

= First(Expression) → {First(Factor)}

= First(Expression) → {ID , Number}

First(Expression`) → {+,-, **ε**}

28.

Add_Op → + | -

First(Add_Op) → {+,-}

29.

Term→Factor| Term Mul_Op Factor

Remove left-recursion

Term →Factor Term`

Term` → Mul_Op Factor Term` | **ε**

First(Term) → {First(Factor)}

= First(Term) → {ID , Number}

First(Term`) → {* , / , **ε**}

30.

Mul_Op→* | /

First(Mul_Op) → {*, /}

31.

Factor→ ID| Number

First(Factor)→ {ID , Number}

32.

Comment →<* STR *> | -- STR

First(Comment) →{ < ,- }

33.

using_command →using(F_name.txt);

First(using_command) →{using}

34.

F_name →STR

First(F_name) →{STR}

1.

Follow(Program) → {&)

Follow(Program`) → {&)

2.

Follow(ClassDeclaration) → {End)

3.

Follow(Class_Implementation) → {}}

4.

Follow(MethodDeclaration) → {Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , | , using , ID , } }

5.

Follow(Func Decl)→ {; , { }

6.

Follow(Type)→ {ID}

7.

Follow(ParameterList)→{) }

8.

Follow(Non-Empty List) → {) }

Follow(Non-Empty List`) → {) }

9.

Follow(VarDeclaration)→ { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write , } }

10.

Follow(ID_List) → { ; }

Follow(ID_List`) → { ; }

11.

Follow(Statements) → { } }

12.

Follow(Statement) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write , } }

13.

Follow(Assignment)→ { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write , } }

14.

Follow(Func _Call)→ { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , | , using , ID ,}}

15.

Follow(Argument_List)→ {) }

16.

Follow(NonEmpty_Argument_List)→ {) }

Follow(NonEmpty_Argument_List`)→ {) }

17.

Follow(Block Statements)→ { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write , }}

18.

Follow(If _Statement)→ { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write , } }

19.

Follow(Condition _Expression)→ {) }

20.

Follow(Condition _Op)→ {ID , Number}

21.

Follow(Condition)→ {) , and , or}

22.

Follow(Comparison _Op) →{ID , Number}

23.

Follow(Rotate _Statement) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write , } }

24.

Follow(Continuewhen _Statement) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write , } }

25.

Follow(Replywith _Statement) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write , } }

26.

Follow(terminatethis _Statement) →{ Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , if, Rotate, Continuewhen, Replywith, Terminatethis, read, write , } }

27.

Follow(Expression)→{) , ; , , , =, ! , > , < ,) , and , or, + , - }

Follow(Expression`) → {) , ; , , , =, ! , > , < ,) , and , or, + , - }

28.

Follow(Add_Op) → {ID , Number}

29.

Follow(Term) → {) , ; , , , =, ! , > , < ,) , and , or, + , - }

Follow(Term`) → {) , ; , , , =, ! , > , < ,) , and , or, + , - }

30.

Follow(Mul_Op) → {ID , Number}

31.

Follow(Factor) →{* , / ,) , ; , , , =, ! , > , < ,) , and , or, + , - }

32.

Follow(Comment) → { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , < , - , using , ID ,} }

33.

Follow(using_command) →{ Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , < , - , using , ID ,} }

34.

Follow(F_name) → {.txt}