CS 419 Compiler

Project Form

Project Idea:

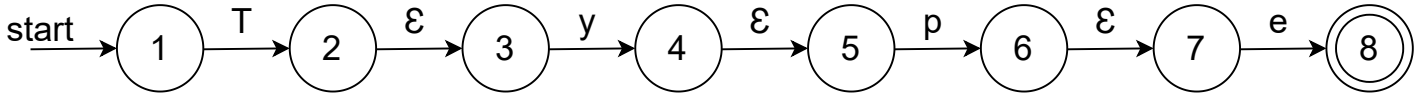
PROJECT#3 is a case sensitive objects oriented Language.

Team Members NO#: (7)

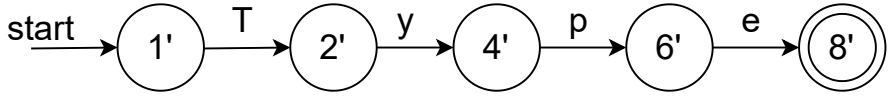
ID	Name	Level& Department	Section(Day -from-to)	Role (Lead/Member)	Grade
201900950	وليد شعبان عبد الحليم رضوان	Level 3 (CS)	الاربع من 2 ل 4	Leader	
201900062	احمد عزت نصر محمد	Level 3 (CS)	الاربع من 10 ل 12	Member	
201900079	احمد ماهر عبد القادر سيد	Level 3 (CS)	الاربع من 10 ل 12	Member	
201900279	خالد رزق علي فرج	Level 3 (CS)	الاربع من 12 ل 2	Member	
201900040	أحمد سمير حشمت حافظ	Level 3 (CS)	الاربع من 10 ل 12	Member	
201900806	مريم يحى صلاح مرزوق	Level 3 (CS)	الاربع من 2 ل 4	Member	
201900356	سهيلة حسن حسني احمد	Level 3 (CS)	الاربع من 12 ل 2	Member	

Regex : Type

NFA



DFA

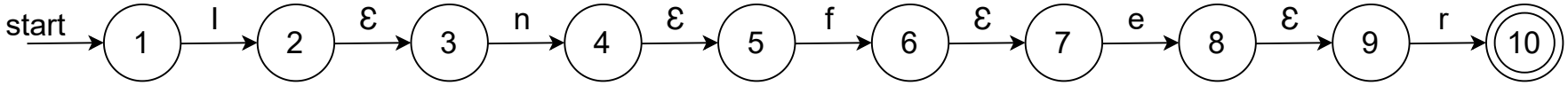


Transition Table

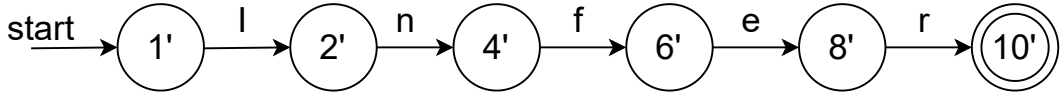
	T	y	p	e
1'	2'			
2'		4'		
4'			6'	
6'				8'
8'				

Regex : Infer

NFA



DFA

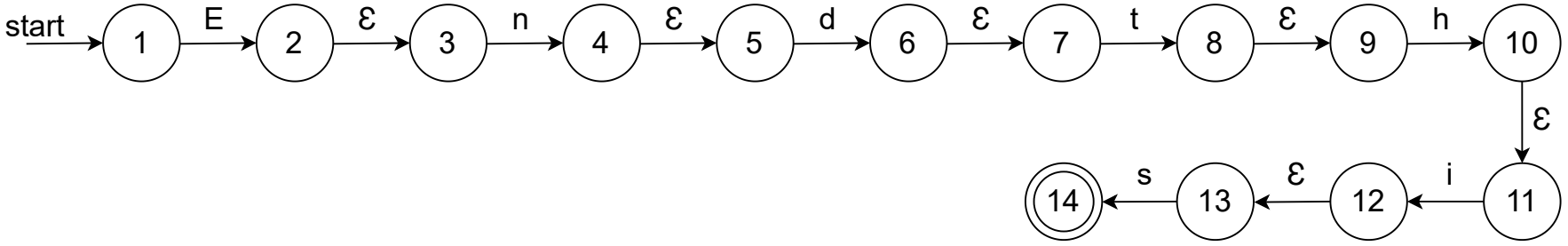


Transition Table

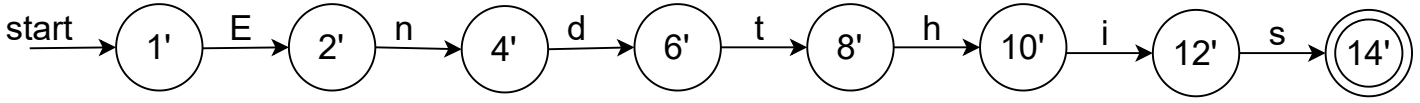
	l	n	f	e	r
1'	2'				
2'		4'			
4'			6'		
6'				8'	
8'					10'
10'					

Regex : Endthis

NFA



DFA

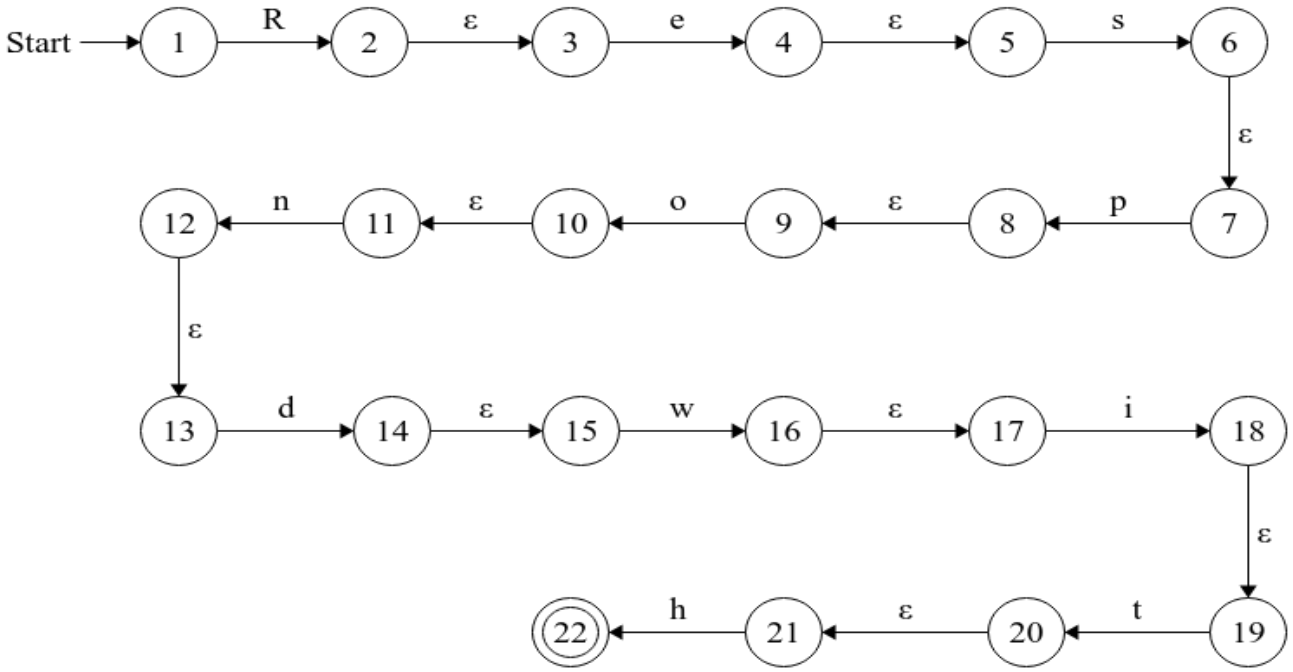


Transition Table

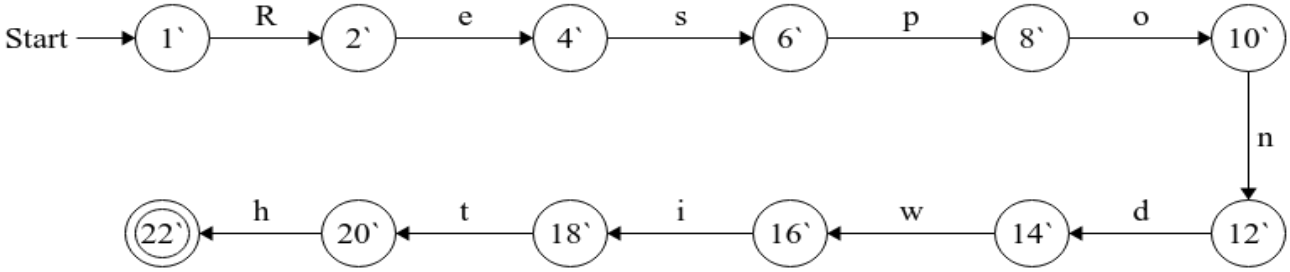
	E	n	d	t	h	i	s
1'	2'						
2'		4'					
4'			6'				
6'				8'			
8'					10'		
10'						12'	
12'							14'
14'							

1Regex: Respondwith

NFA:



DFA:

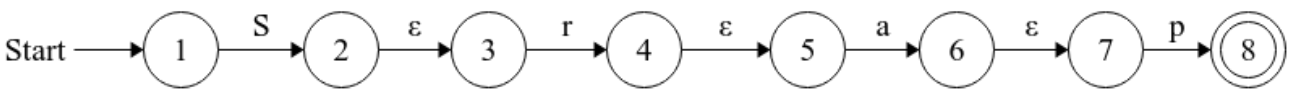


Transition Table:

	R	e	s	p	o	n	d	w	i	t	h
1`	2`										
2`		4`									
4`			6`								
6`				8`							
8`					10`						
10`						12`					
12`							14`				
14`								16`			
16`									18`		
18`										20`	
20`											22`
22`											

Regex: Srap

NFA:



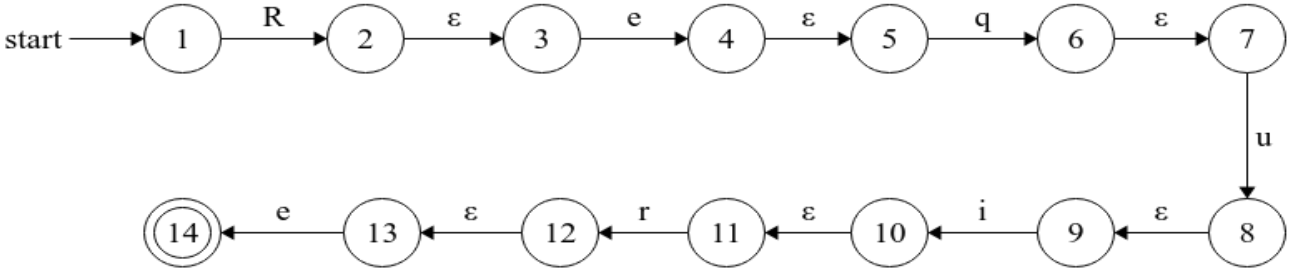
DFA:

Transition Table:

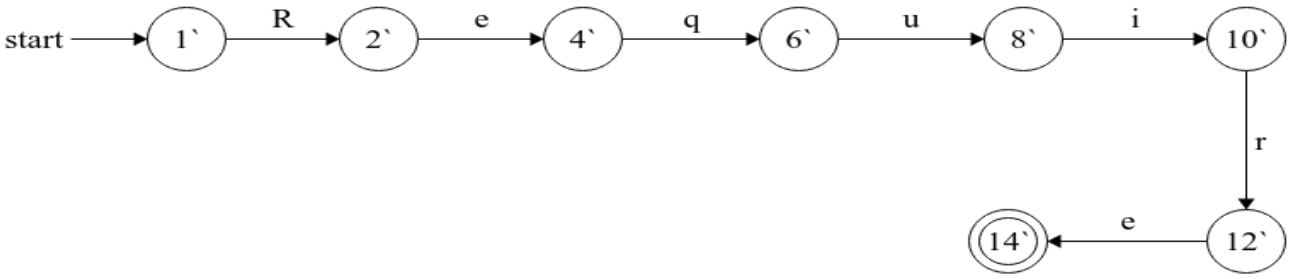
	S	r	a	p
1`	2`			
2`		4`		
4`			6`	
6`				8`
8`				

Regex: Require

NFA:



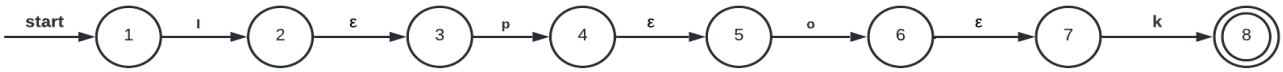
DFA:



Transition Table:

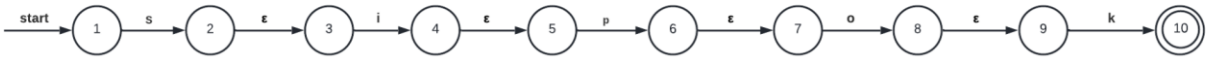
	R	e	q	u	i	r	e
1`	2`						
2`		4`					
4`			6`				
6`				8`			
8`					10`		
10`						12`	
12`							14`
14`							

Regex: Ipok



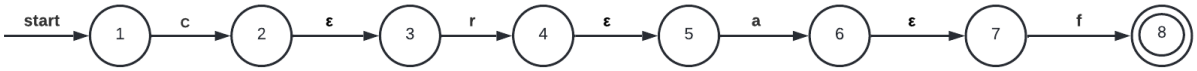
	I	p	o	k
1`	2`			
2`		4`		
4`			6`	
6`				8`
8`				

Regex: Sipok



	S	i	p	o	k
1`	2`				
2`		4`			
4`			6`		
6`				8`	
8`					10`

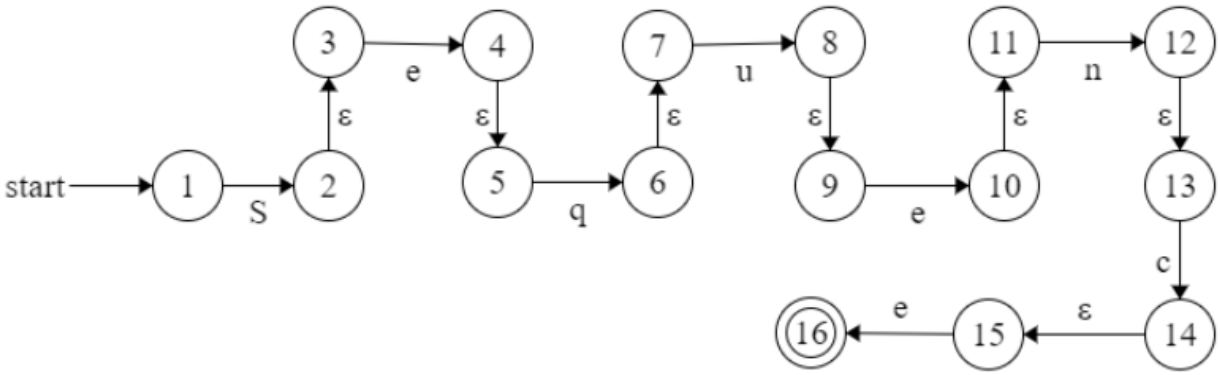
Regex: Craf



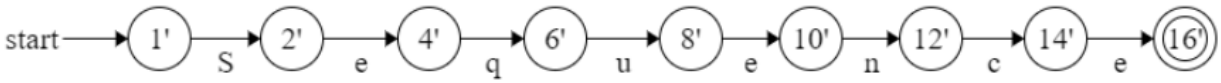
	C	r	a	f
1`	2`			
2`		4`		
4`			6`	
6`				8`
8`				

Regex: Sequence

NFA:



DFA:

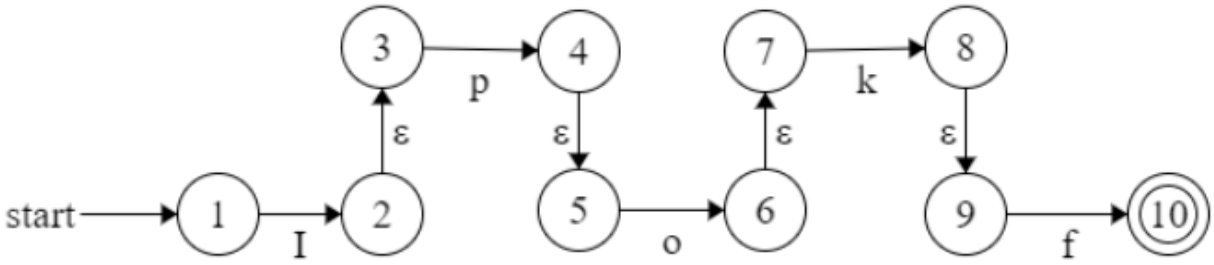


Transition table:

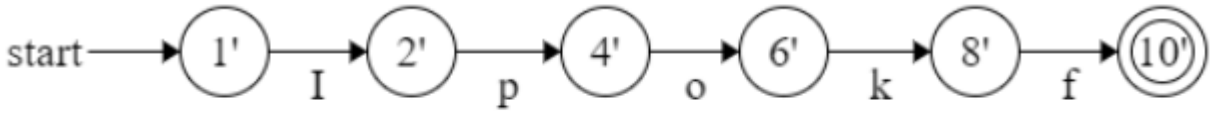
	S	e	q	u	e	n	c	e
1`	2`							
2`		4`						
4`			6`					
6`				8`				
8`					10`			
10`						12`		
12`							14`	
14`								16`
16`								

Regex: Ipokf

NFA:



DFA:

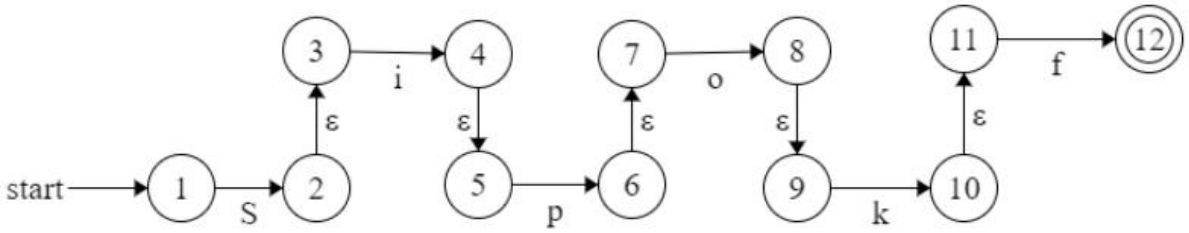


Transition table:

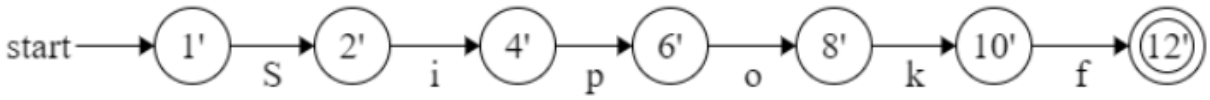
	I	p	o	k	f
1`	2`				
2`		4`			
4`			6`		
6`				8`	
8`					10`

Regex: Sipokf

NFA:



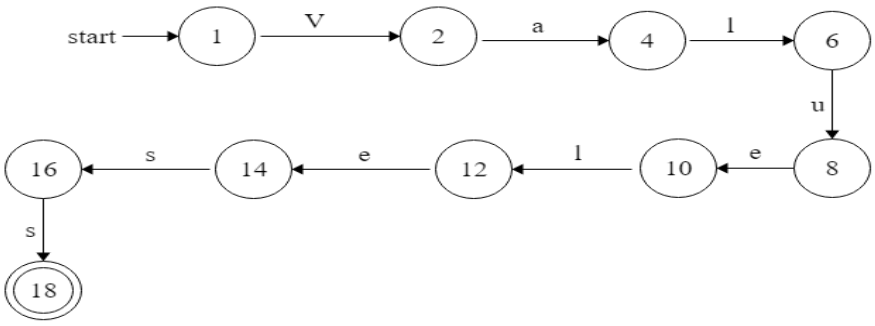
DFA:



Transition table:s

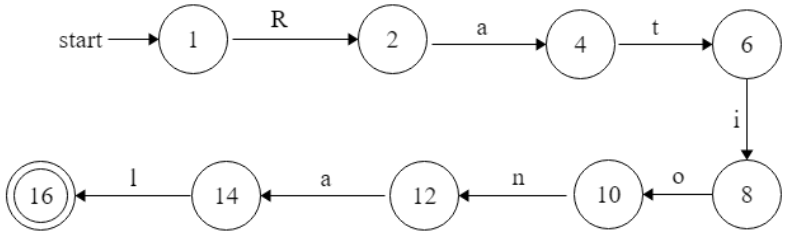
	S	i	p	o	k	f
1`	2`					
2`		4`				
4`			6`			
6`				8`		
8`					10`	
10`						12`
12`						

Valueless



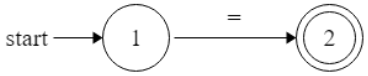
	V	a	l	u	e	l	e	s	s
1	2								
2		4							
4			6						
6				8					
8					10				
10						12			
12							14		
14								16	
16									18

Rational



	R	a	t	i	o	n	a	l
1	2							
2		4						
4			6					
6				8				
8					10			
10						12		
12							14	
14								16

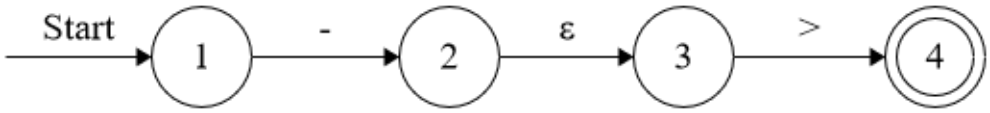
Regex: =



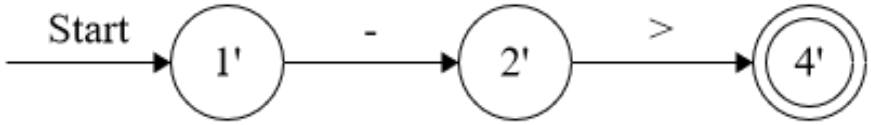
	=
1	**2**

Regex: ->

NFA:



DFA:

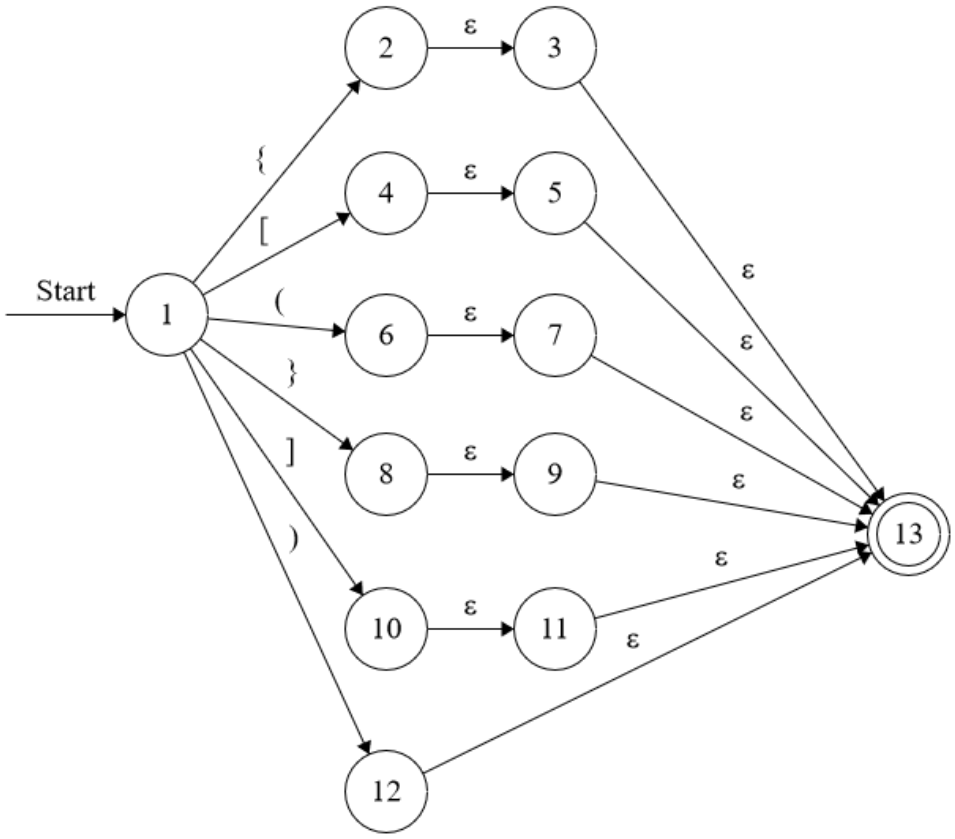


Transition Table:

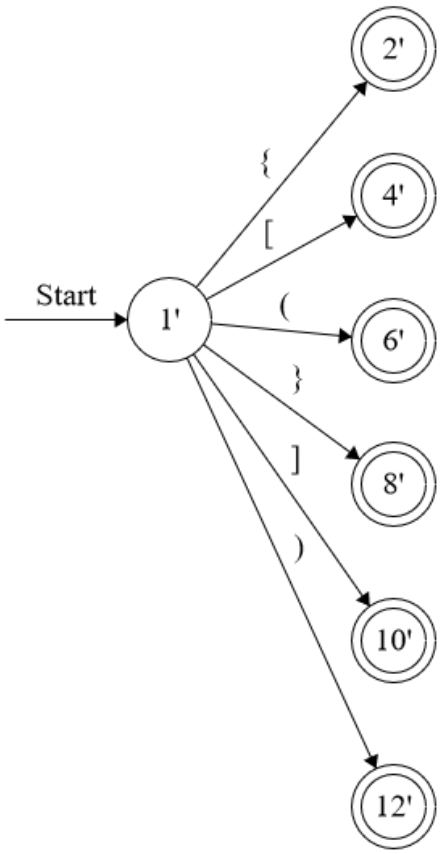
	-	>
1’	2’	
2’		4’
4’		

Regex: {} | [] | ()

NFA:



DFA:

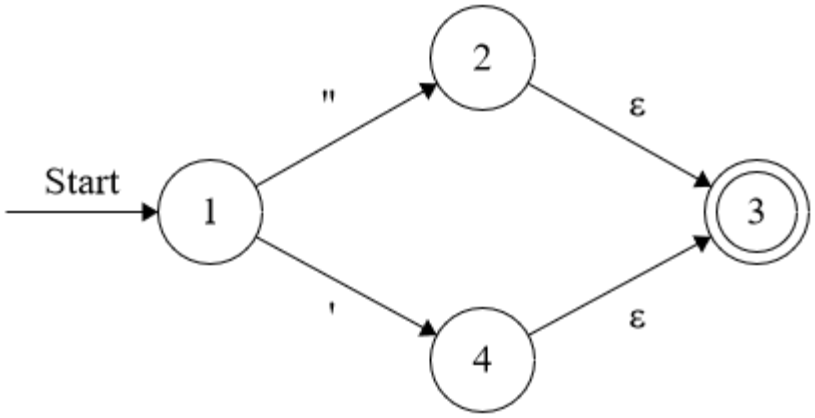


Transition Table:

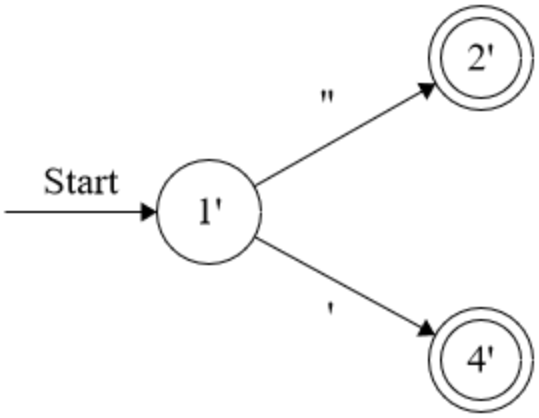
	{	[	(	}	]	)
1’	2’	4’	6’	8’	10’	12
2’						
4’						
6’						
8’						
10’						
12’						

Regex: "|'

NFA:



DFA:

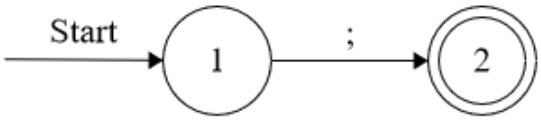


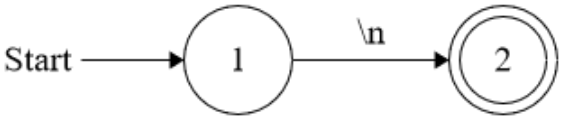
Transition Table:

	‘	“
1’	2’	4’
2’		
4’		

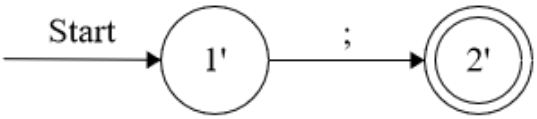
Regex: ; | \n

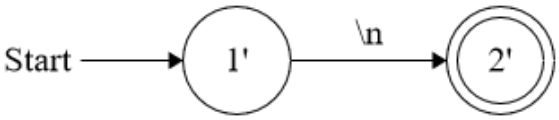
NFA:





DFA:





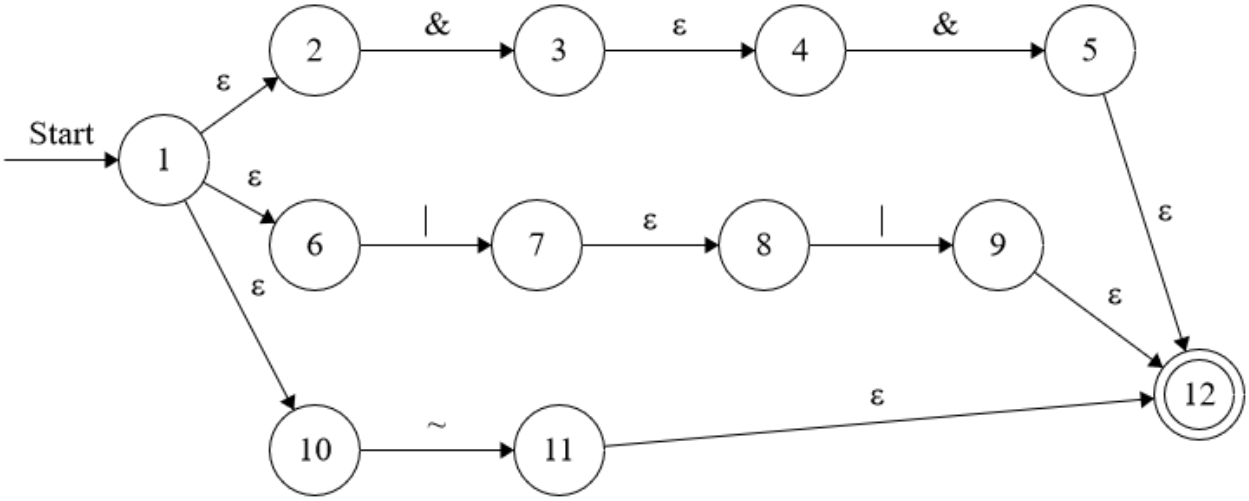
Transition Table:

	;
1’	2’
2’	

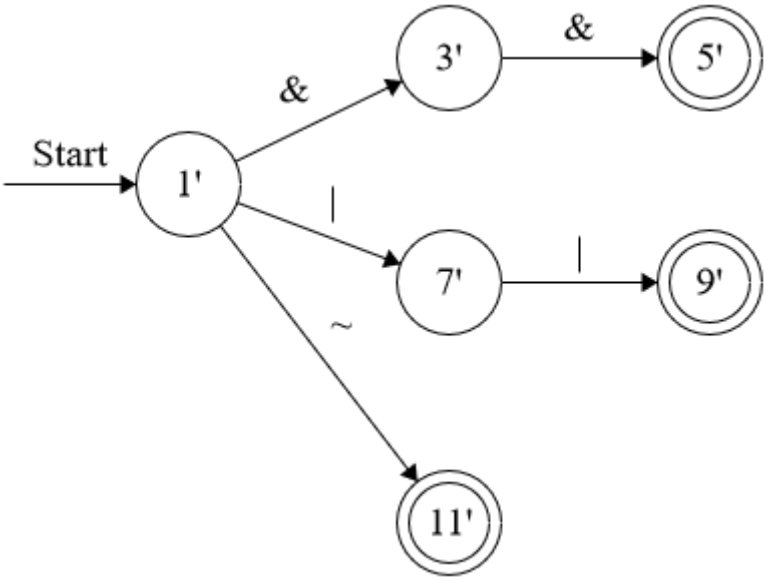
	/n
1’	2’
2’	

Regex: && , || , ~

NFA:



DFA:

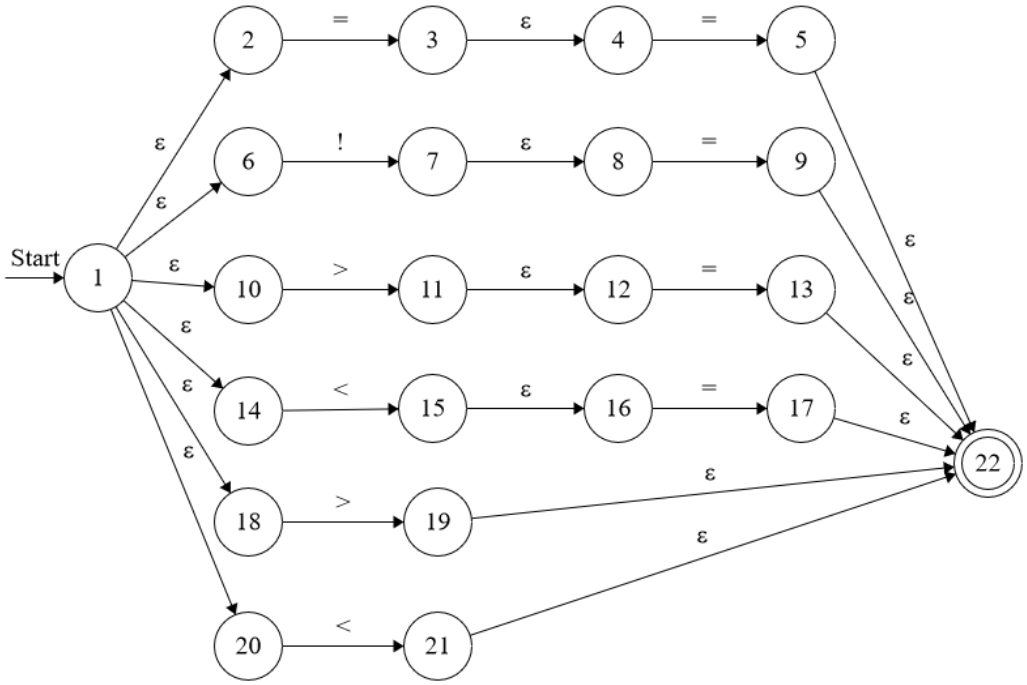


Transition Table:

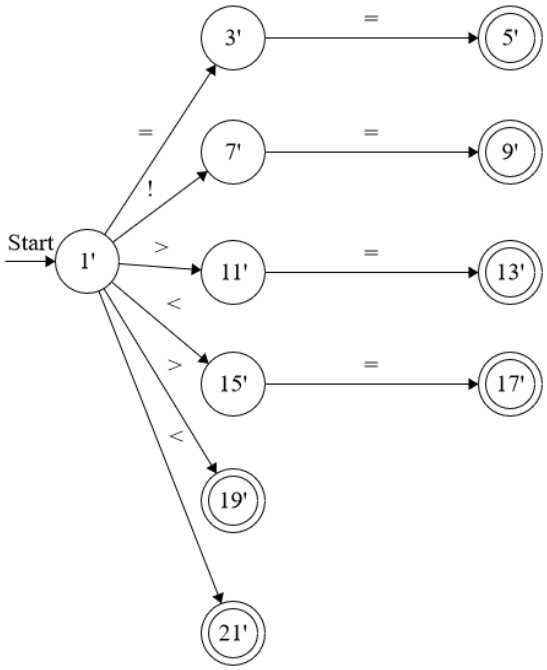
	&	&	|	|	~		
1’	3’		7’		11’		
3’		5’					
5’							
7’				9’			
9’							
11’							

Regex: ==|<|>|!=|<=|>=

NFA:



DFA:

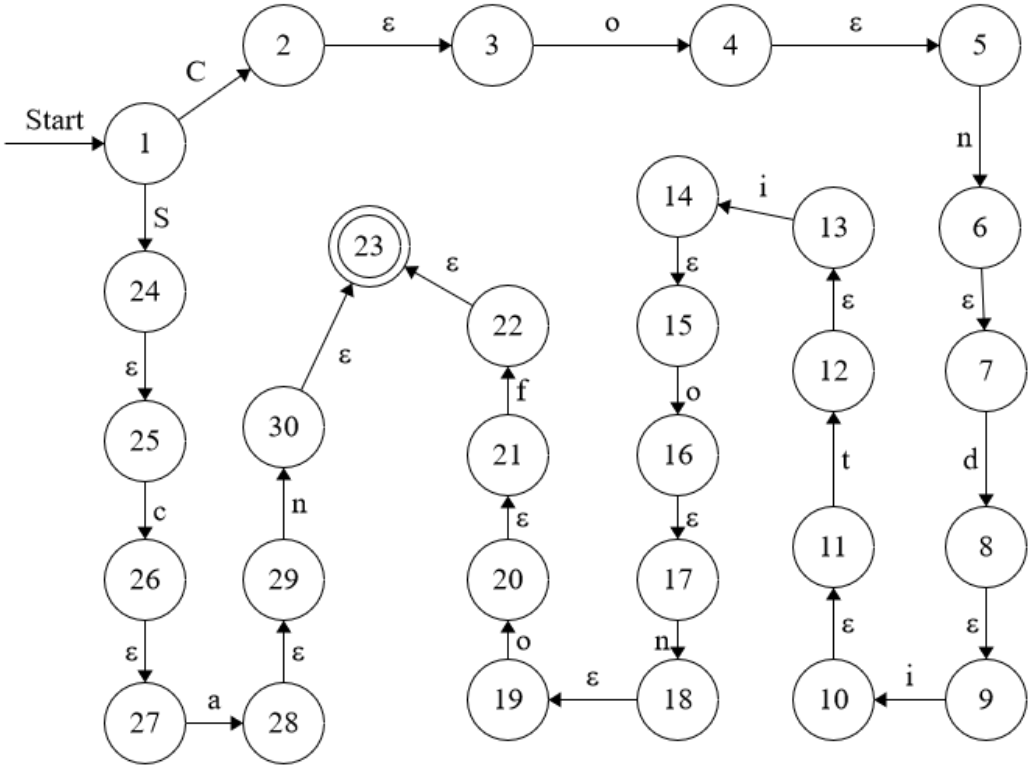


Transition Table:

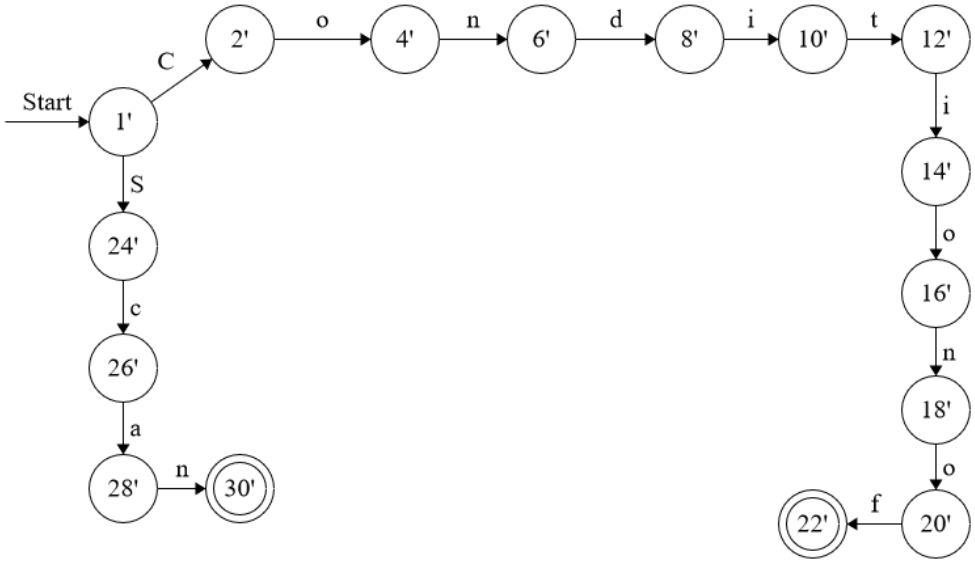
	=	=	!	=	>	=	<	=	>	<
1’	3’		7’		11’		15’		19’	21’
3’		5’								
5’										
7’				9’						
9’										
11’						13’				
13’										
15’								17’		
17’										
19’										
21’										

Regex: Scan | Conditionof

NFA:



DFA:

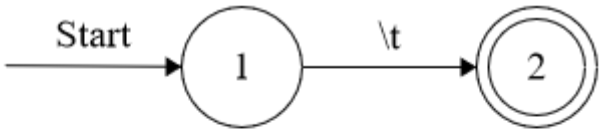


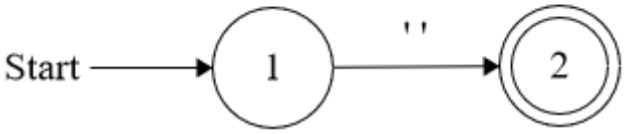
Transition Table:

	C	o	n	d	i	t	i	o	n	o	f	S	c	a	n
1’	2’											24’			
2’		4’													
4’			6’												
6’				8’											
8’					10’										
10’						12’									
12’							14’								
14’								16’							
16’									18’						
18’										20’					
20’											22’				
22’															
24’													26’		
26’														28’	
28’															30’
30’															

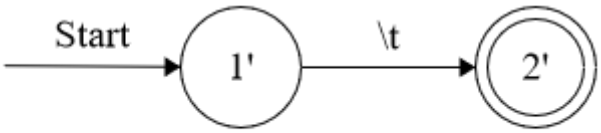
Regex: \t|

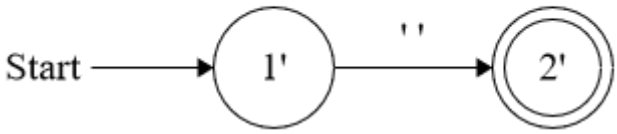
NFA:





DFA:





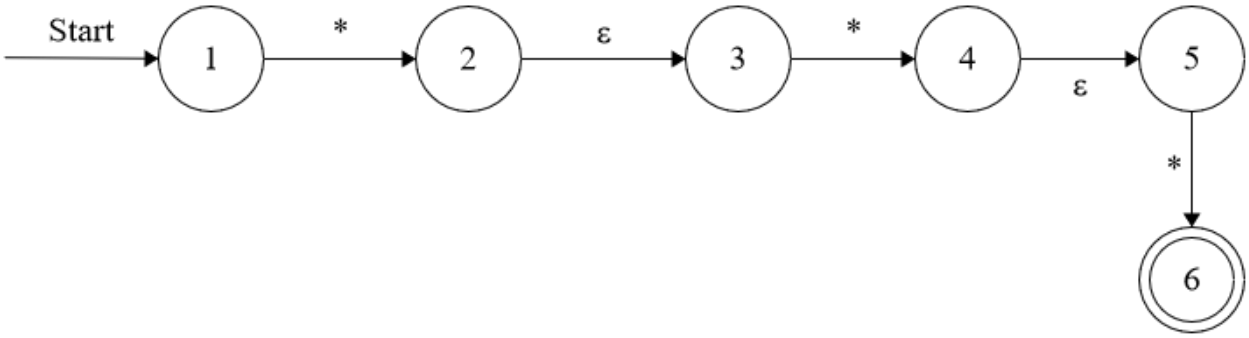
Transition Table:

	/t
1’	2’
2’	

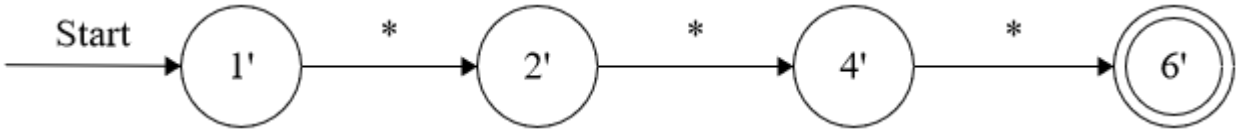
	space
1’	2’
2’	

Regex: ***

NFA:



DFA:

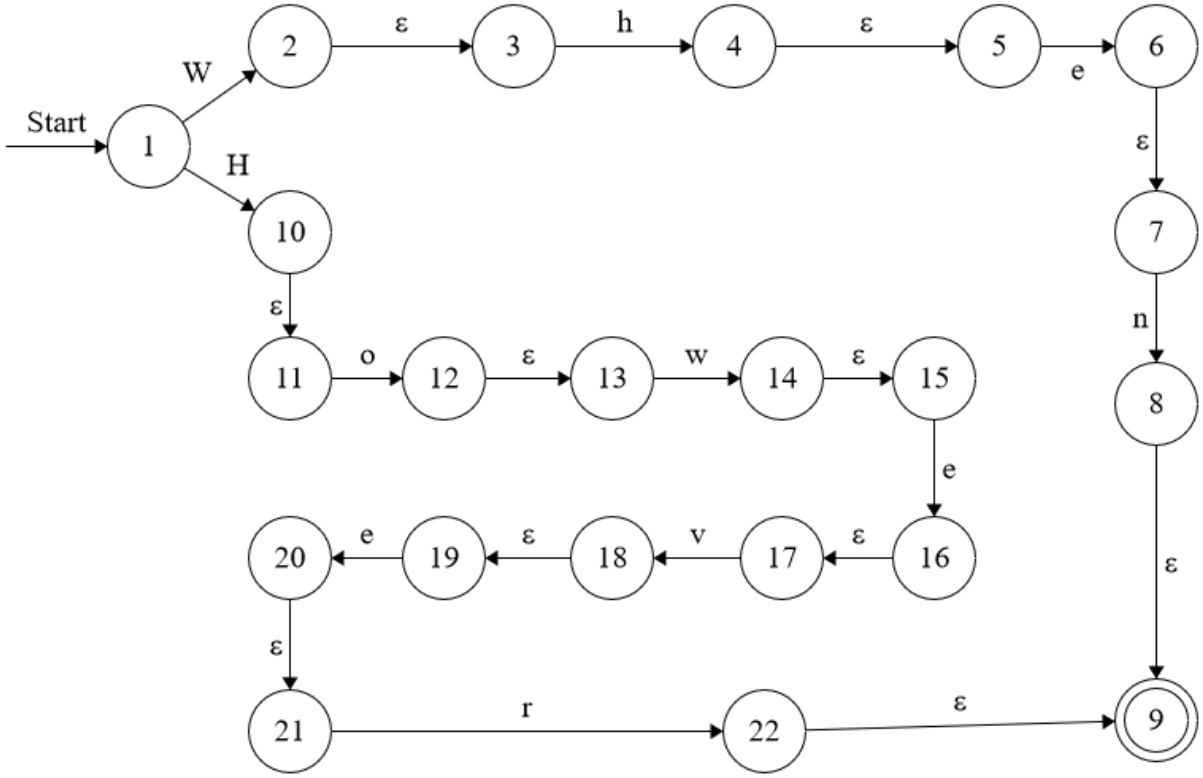


Transition Table:

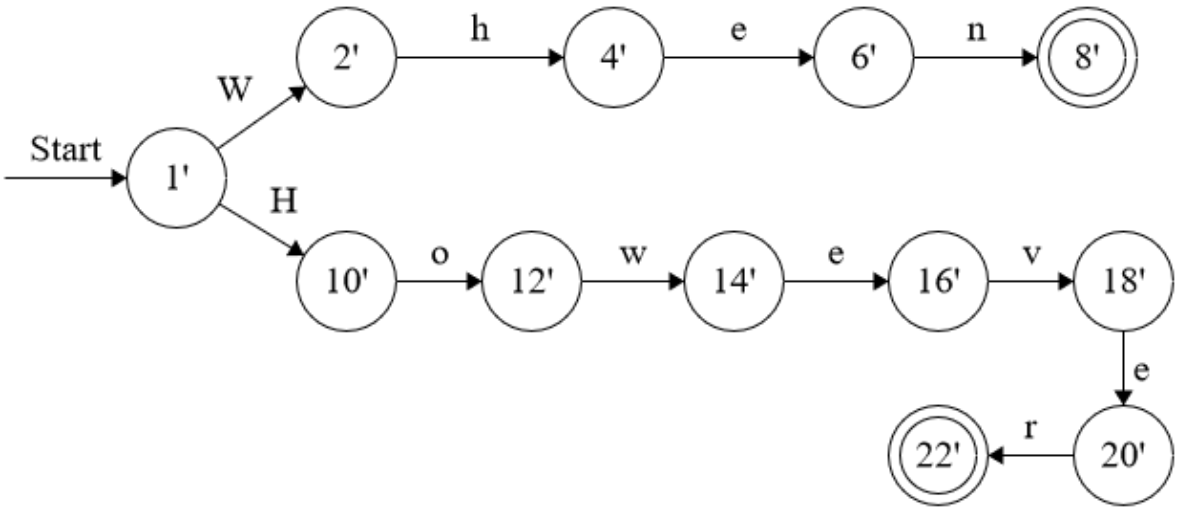
	*	*	*
1’	2’		
2’		4’	
4’			6’
6’			

Regex: However | When

NFA:



DFA:

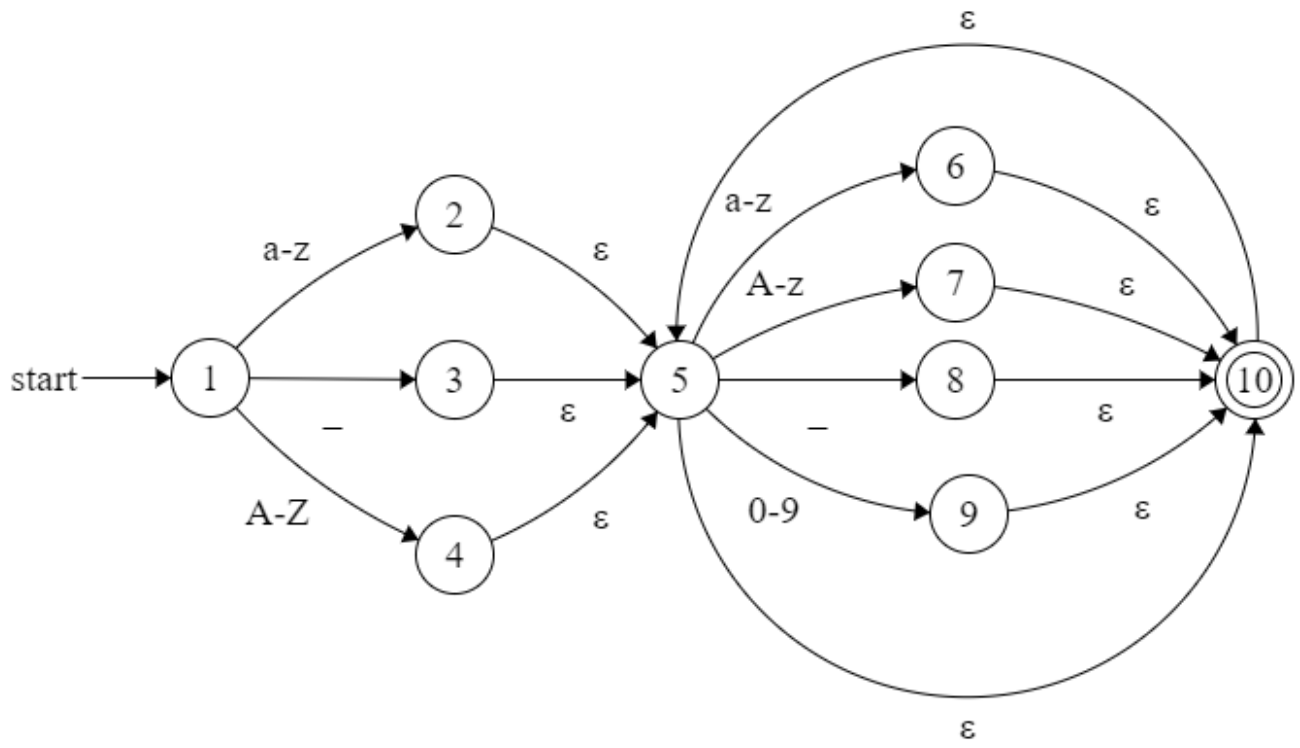


Transition Table:

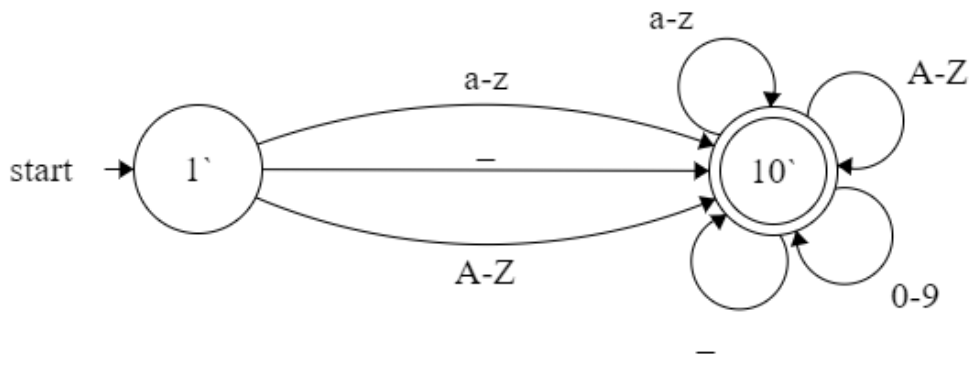
	W	h	e	n	H	o	w	e	v	e	r
1’	2’				10’						
2’		4’									
4’			6’								
6				8’							
8’											
10’						12’					
12’							14’				
14’								16’			
16’									18’		
18’										20’	
20’											22’
22’											

Regex: [a-zA-z_][a-zA-Z0-9_]*

NFA:



DFA:



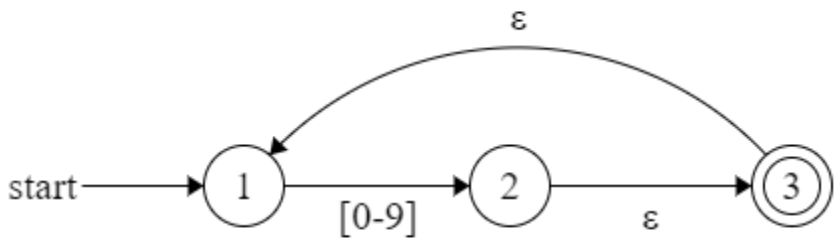
0

Transition table:

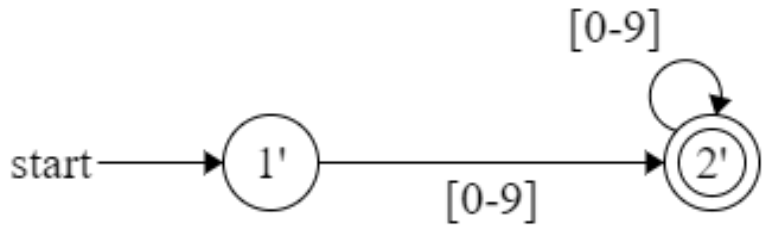
	a-z	A-Z	_	0-9
1`	10`	10`	10`	10`
10`	10`	10`	10`	10`

Regex: [0-9]+

NFA:



DFA:



Transition table:

	0-9
1`	2`
2`	2`

Program -> StartSymbols ClassDeclaration EndSymbol

StartSymbols -> @ | ^

EndSymbols -> $ | #

ClassDeclaration -> TypeID ClassDeclaration'

ClassDeclaration' -> { ClassImplementation } | Infer { ClassImplementation }

ClassImplementation -> VariableDecl ClassImplementation
| MethodDecl ClassImplementation
| Comment ClassImplementation
| RequireCommand ClassImplementation
| FuncCall ClassImplementation
| ϵ

MethodDecl -> FuncDecl MethodDecl'

MethodDecl' -> ; | { VariableDecl Statements }

FuncDecl -> TypeID (ParameterList)

Type -> Ipok
| Sipok
| Craf
| Sequence
| Ipokf
| Sipokf
| Valueless
| Rationa

ParameterList -> None

| NonEmptyList

| ϵ

NonEmptyList -> TypeID NonEmptyList'

NonEmptyList' -> , TypeID NonEmptyList' | ϵ

VariableDecl -> TypeIDList VariableDecl' | ϵ

VariableDecl' -> ; VariableDecl | [ID] ; VariableDecl

IDList -> ID IDList'

IDList' -> , ID IDList' | ϵ

Statements -> Statement Statements

Statement -> Assignment

| IfStatement

| HoweverStatement

| whenStatement

| RespondwithStatement

| EndthisStatement

| Scanvalur (ID) ;

| Print (Expression) ;

| ϵ

Assignment -> TypeIDList VariableDecl' = Expression

FuncCall -> ID (ArgumentList) ;

ArgumentList -> NonEmptyArgumentList | ϵ

NonEmptyArgumentList -> Expression NonEmptyArgumentList'

NonEmptyArgumentList' -> , Expression NonEmptyArgumentList' | ϵ

BlockStatements -> { statements }

IfStatement -> if (ConditionExpression) Block Statements IfStatement'

IfStatement' -> else Block Statements | ϵ

ConditionExpression -> Condition ConditionExpression'

ConditionExpression' -> && Condition | || Condition | ϵ

Condition -> Expression ComparisonOp Expression

ComparisonOp -> ==
| !=
| >
| >=
| <
| <=

HoweverStatement -> However (ConditionExpression) Block Statements

whenStatement -> when (expression ; expression ; expression) Block Statements

RespondwithStatement -> Respondwith Expression ; | return ID ;

EndthisStatement -> Endthis ;

Expression -> Term Expression'
Expression' -> AddOp Term Expression' | ϵ

AddOp -> + | -

Term -> Factor Term'
Term' -> MulOp Factor Term' | ϵ

MulOp -> * | /

Factor -> ID | Number

Comment -> </ /> | ***

RequireCommand -> Require (F_name .txt) ;

F_name -> STR

Nonterminal	Nullable?	First	Follow
S	✕	@	
Program	✕	@	$
StartSymbols	✕	@	TypeID
EndSymbols	✕	$	
ClassDeclaration	✕	TypeID	EndSymbol
ClassDeclaration'	✕	{	EndSymbol
ClassImplementation	✕	TypeIDList	}
MethodDecl	✕	TypeID	
MethodDecl'	✕	;	
FuncDecl	✕	TypeID	;
Type	✕	Ipok	
ParameterList	✕	None	)
NonEmptyList	✕	TypeID	
NonEmptyList'	✕	,	|
VariableDecl	✕	TypeIDList	TypeIDList, |, =
VariableDecl'	✕	;	|, =
IDList	✕	ID	
IDList'	✕	,	|
Statements	✕	TypeIDList	}, else, |
Statement	✕	TypeIDList	TypeIDList
Assignment	✕	TypeIDList	TypeIDList
FuncCall	✕	ID	
ArgumentList	✕	ID	)
NonEmptyArgumentList	✕	ID	|

Nonterminal	Nullable?	First	Follow
NonEmptyArgumentList'	✕	,	|
BlockStatements	✕	{	
IfStatement	✕	if	
IfStatement'	✕	else	
ConditionExpression	✕	ID	)
ConditionExpression'	✕	&&	)
Condition	✕	ID	&&, |
ComparisonOp	✕	==	ID
HoweverStatement	✕	However	
whenStatement	✕	when	
RespondwithStatement	✕	Respondwith	
EndthisStatement	✕	Endthis	
Expression	✕	ID	TypeIDList, ,, &&, |, ==, ;
Expression'	✕	+	TypeIDList, ,, &&, |, ==, ;
AddOp	✕	+	ID
Term	✕	ID	+
Term'	✕	*	+, |
MulOp	✕	*	ID
Factor	✕	ID	*
Comment	✕		
RequireCommand	✕	Require	
F_name	✕	STR	.txt

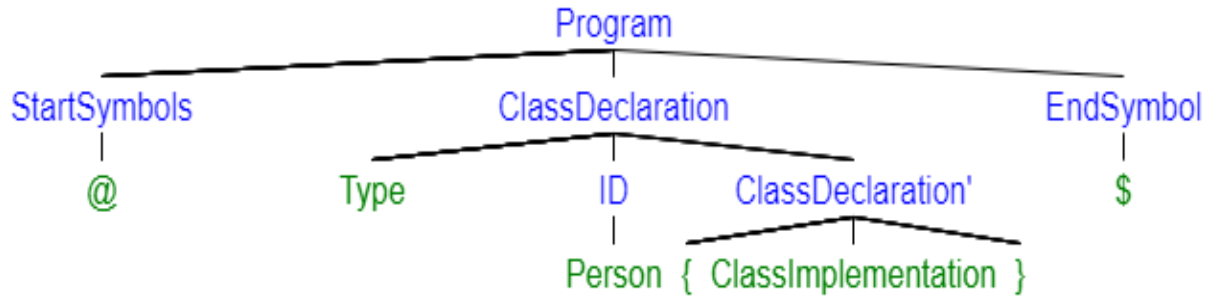
	$	EndSymbol	@	|	^	#	TypeID	{	}	Infer	;	(	)	Ipok	None	,	ε	TypeIDList	[	ID	]	=	statements	if	Block	else	&&	||	==	However	when	expression	Respondwith	return	Endthis	+	-	*	/	Number		/>	***	Require	.txt	STR
S			S ::= Program $																																											
Program			Program ::= StartSymbols ClassDeclaration EndSymbol																																											
StartSymbols			StartSymbols ::= @ | ^																																											
EndSymbols	EndSymbols ::= $ | #																																													
ClassDeclaration							ClassDeclaration ::= TypeID ClassDeclaration'																																							
ClassDeclaration'								ClassDeclaration' ::= { ClassImplementation } | Infer { ClassImplementation }																																						
ClassImplementation																	ClassImplementation ::= VariableDecl ClassImplementation																												
MethodDecl							MethodDecl ::= FuncDecl MethodDecl'																																							
MethodDecl'											MethodDecl' ::= ; | { VariableDecl Statements }																																			
FuncDecl							FuncDecl ::= TypeID (ParameterList)																																							
Type														Type ::= Ipok																																
ParameterList															ParameterList ::= None																															
NonEmptyList							NonEmptyList ::= TypeID NonEmptyList'																																							
NonEmptyList'																NonEmptyList' ::= , TypeID NonEmptyList' | ε																														
VariableDecl																		VariableDecl ::= TypeIDList VariableDecl' | ε																												
VariableDecl'											VariableDecl' ::= ; VariableDecl | [ID] ; VariableDecl																																			
IDList																				IDList ::= ID IDList'																										
IDList'																IDList' ::= , ID IDList' | ε																														
Statements																		Statements ::= Statement Statements																												
Statement																		Statement ::= Assignment																												
Assignment																		Assignment ::= TypeIDList VariableDecl' = Expression																												
FuncCall																				FuncCall ::= ID (ArgumentList) ;																										
ArgumentList																				ArgumentList ::= NonEmptyArgumentList | ε																										
NonEmptyArgumentList																				NonEmptyArgumentList ::= Expression NonEmptyArgumentList'																										
NonEmptyArgumentList'																NonEmptyArgumentList' ::= , Expression NonEmptyArgumentList' | ε																														
BlockStatements								BlockStatements ::= { statements }																																						
IfStatement																								IfStatement ::= if (ConditionExpression) Block Statements IfStatement'																						
IfStatement'																										IfStatement' ::= else Block Statements | ε																				
ConditionExpression																				ConditionExpression ::= Condition ConditionExpression'																										
ConditionExpression'																											ConditionExpression' ::= && Condition | || Condition | ε																			
Condition																				Condition ::= Expression ComparisonOp Expression																										
ComparisonOp																													ComparisonOp ::= ==																	
HoweverStatement																														HoweverStatement ::= However (ConditionExpression) Block Statements																
whenStatement																															whenStatement ::= when (expression ; expression ; expression) Block Statements															
RespondwithStatement																																	RespondwithStatement ::= Respondwith Expression ; | return ID ;													
EndthisStatement																																			EndthisStatement ::= Endthis ;											
Expression																				Expression ::= Term Expression'																										
Expression'																																				Expression' ::= AddOp Term Expression' | ε										
AddOp																																				AddOp ::= + | -										
Term																				Term ::= Factor Term'																										
Term'																																						Term' ::= MulOp Factor Term' | ε								
MulOp																																						MulOp ::= * | /								
Factor																				Factor ::= ID | Number																										
Comment																																									Comment ::= | ***					
RequireCommand																																												RequireCommand ::= Require (F_name .txt) ;		
F_name																																														F_name ::= STR

(Program)

@Type Person{

ClassImplementation

}$



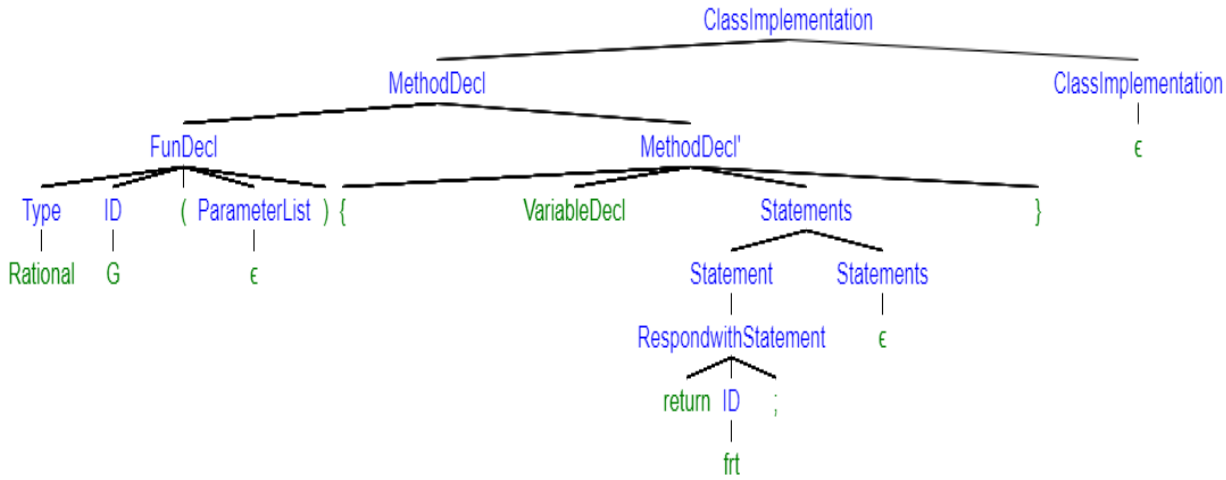
(ClassImplementation)

Rational G () {

VariableDecl

return frt;

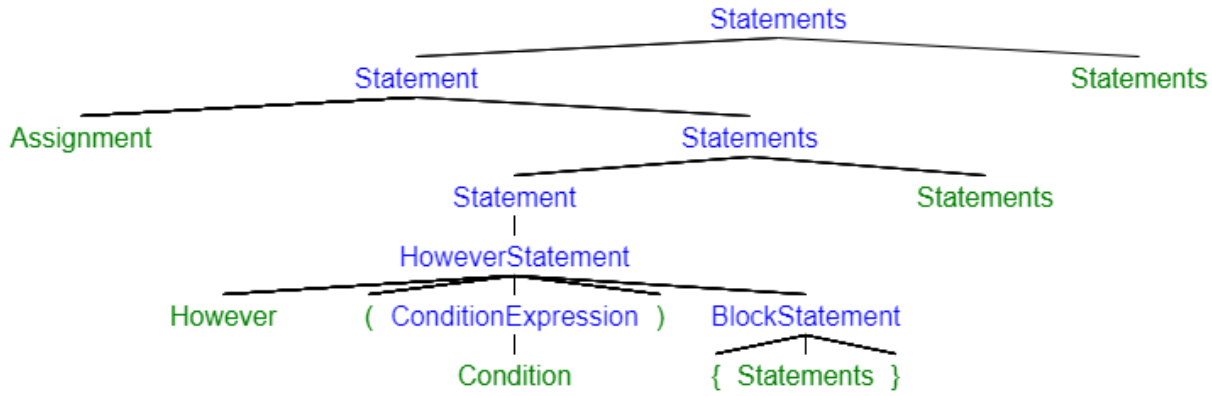
}



(Statements)

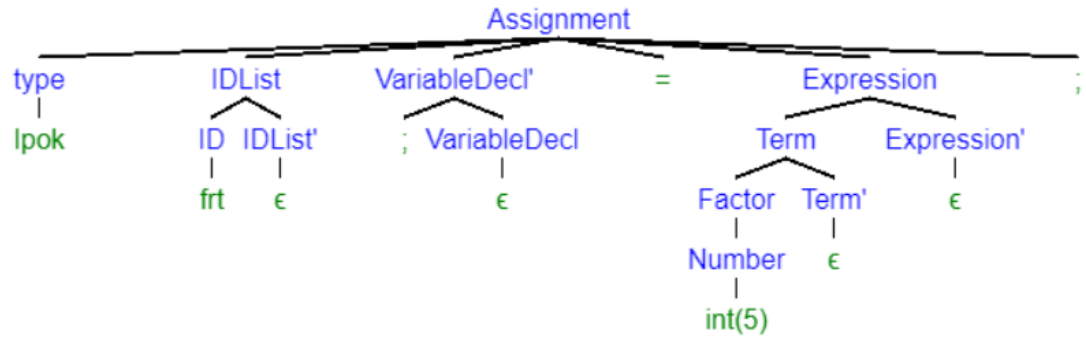
Assignment

However (Condition) {Statements}



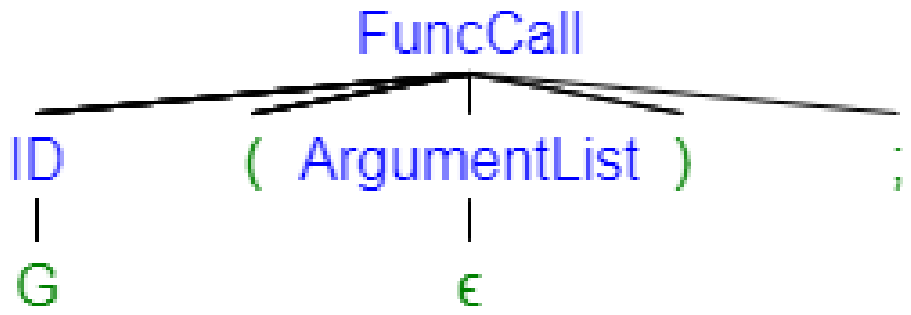
(Assignment)

Ipok frt; = 5;



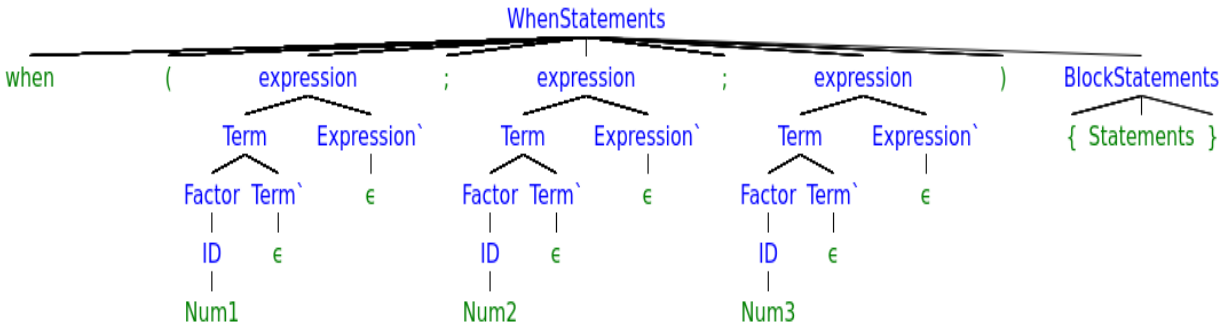
(FuncCall)

G();



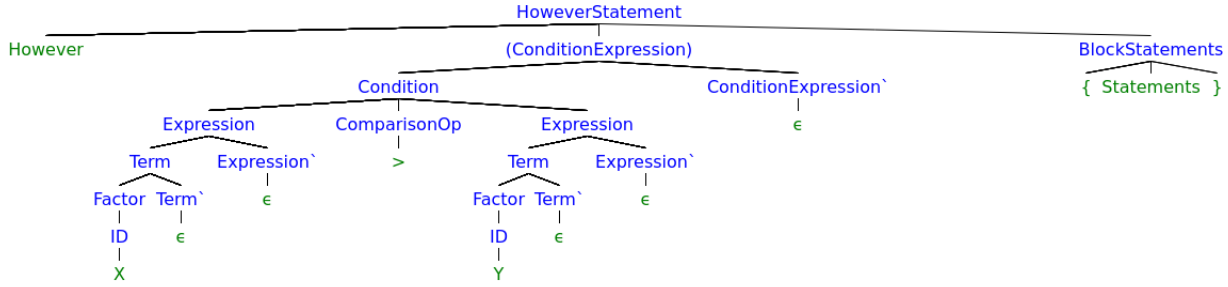
WhenStatements

whenStatement -> when (expression ; expression ; expression) Block Statements

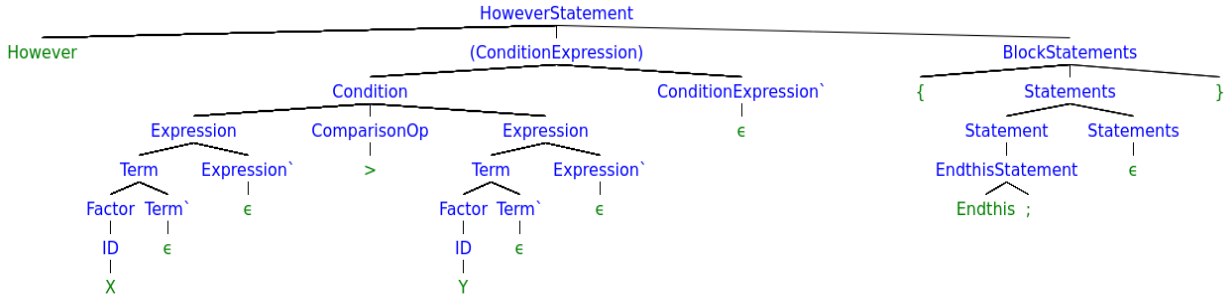


HoweverStatement

HoweverStatement -> However (ConditionExpression) Block Statements

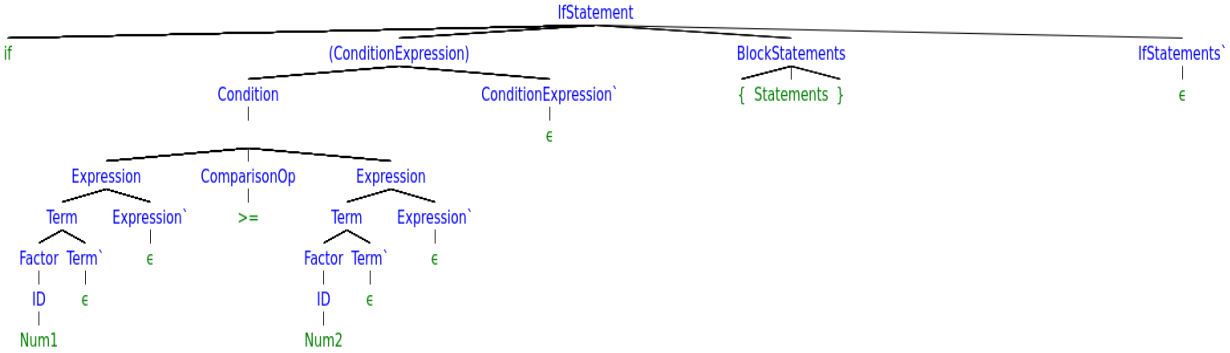


However with EndthisStatement

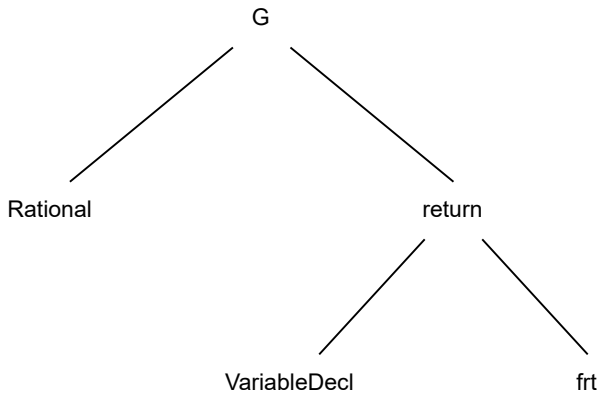


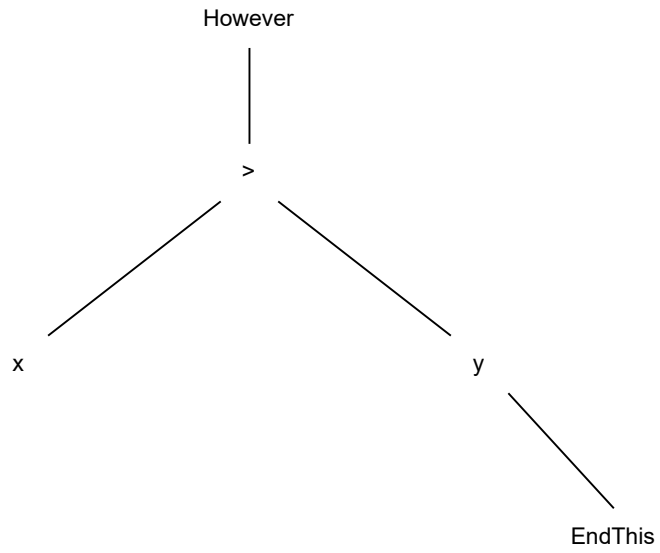
IfStatement

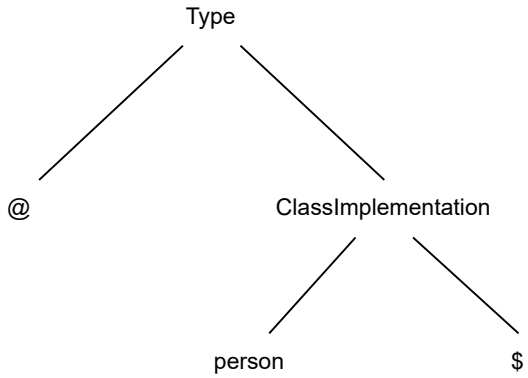
IfStatement -> if (ConditionExpression) Block Statements IfStatement'

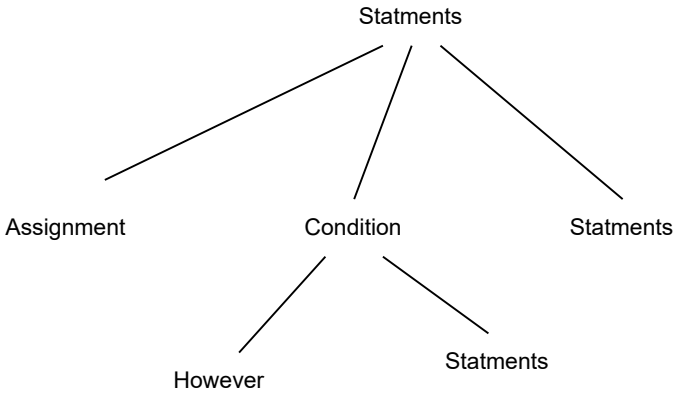


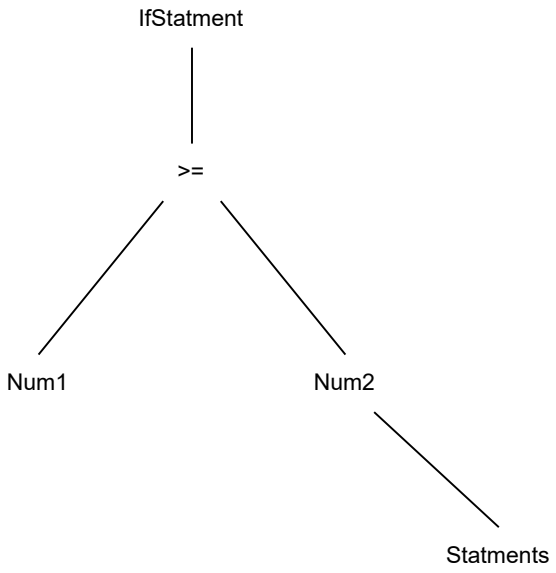
EndthisStatement -> Endthis ;

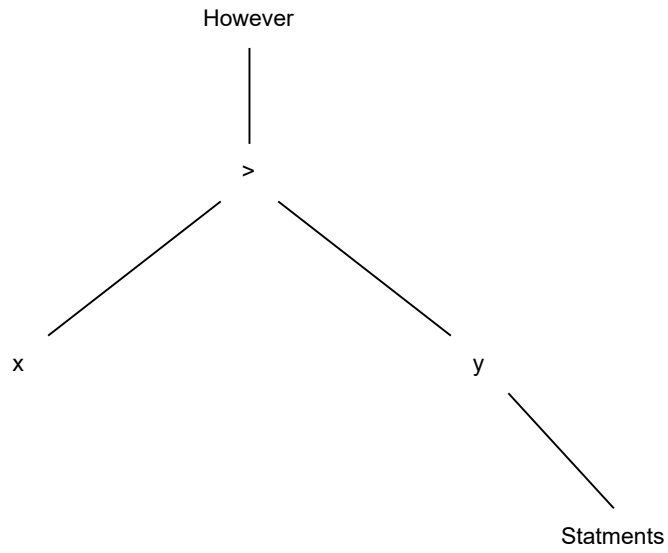

G
Rational
return
VariableDecl
frt

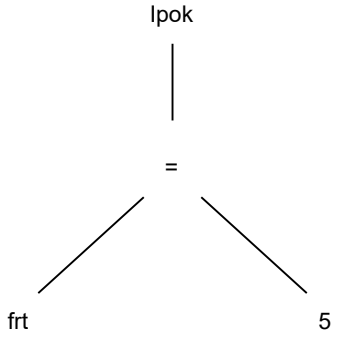

However
>
x
y
EndThis

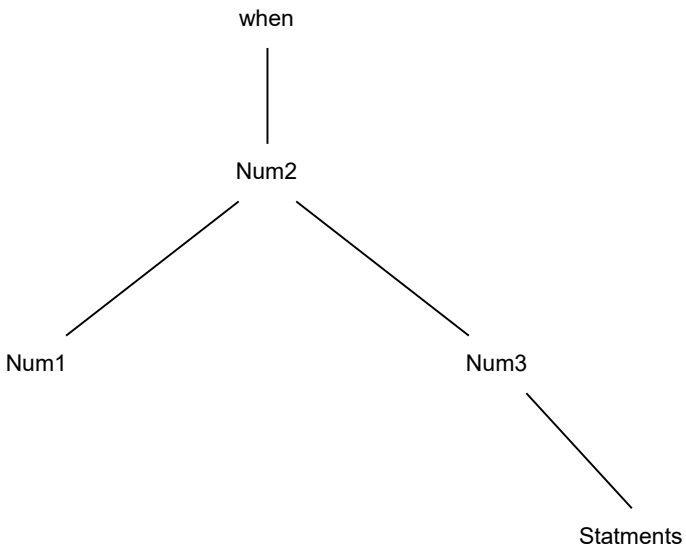

Type
@
ClassImplementation
person
$


Statments
Assignment
Condition
Statments
However
Statments


IfStatment
>=
Num1
Num2
Statments


However
>
x
y
Statments


Ipok
=
frt
5


when
Num2
Num1
Num3
Statments