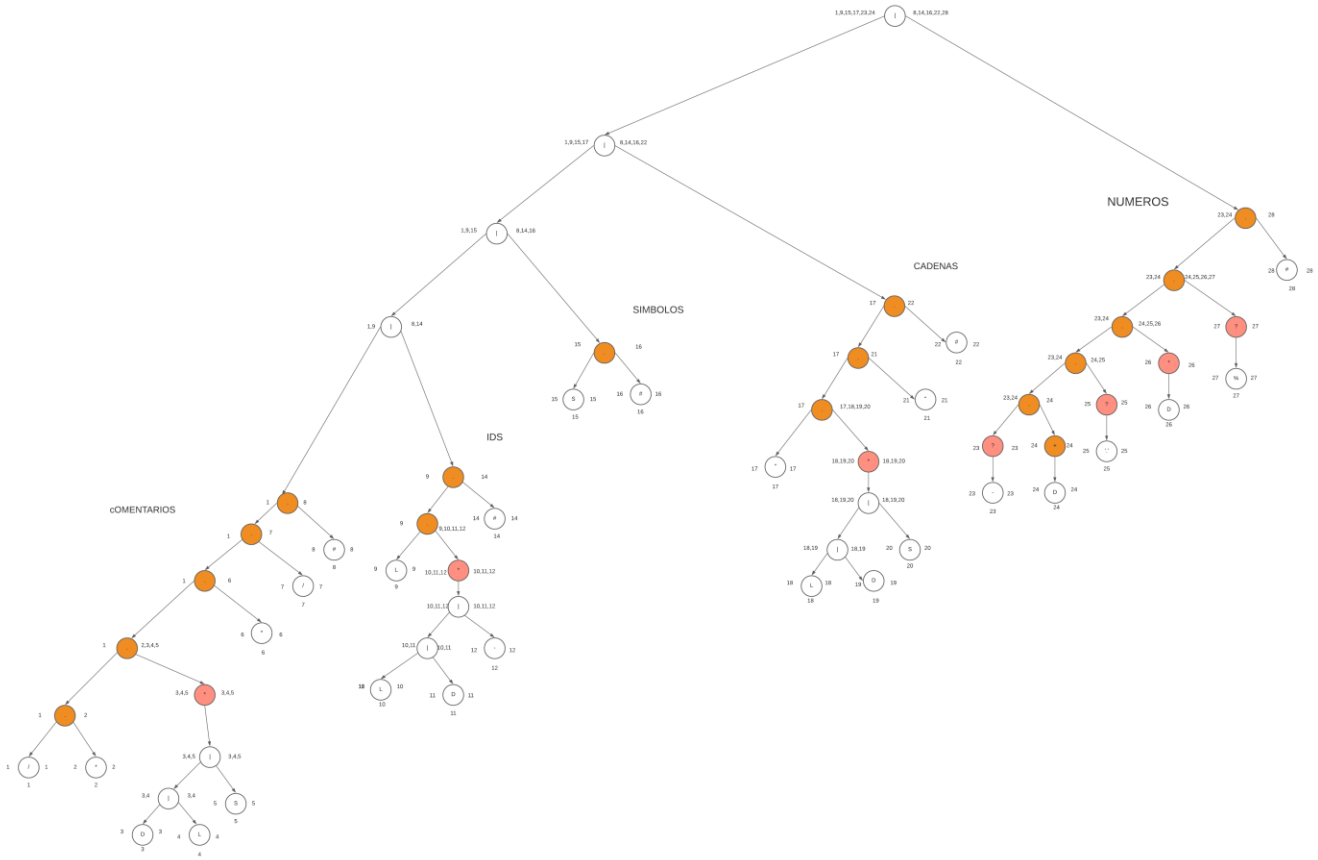
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1
Estudiante: Vernik Carlos Alexander Yaxon Ortiz
Carne: 201712057

Manual Técnico

Para los analizadores léxicos, se hizo uso del método del árbol, para generar los autómatas finitos deterministas y así poder codificarlos de una manera más sencilla y accesible.

Archivos Css

Para el análisis de los archivos css se realizó el método del árbol comenzando con el árbol de las expresiones regulares para el lenguaje de css.



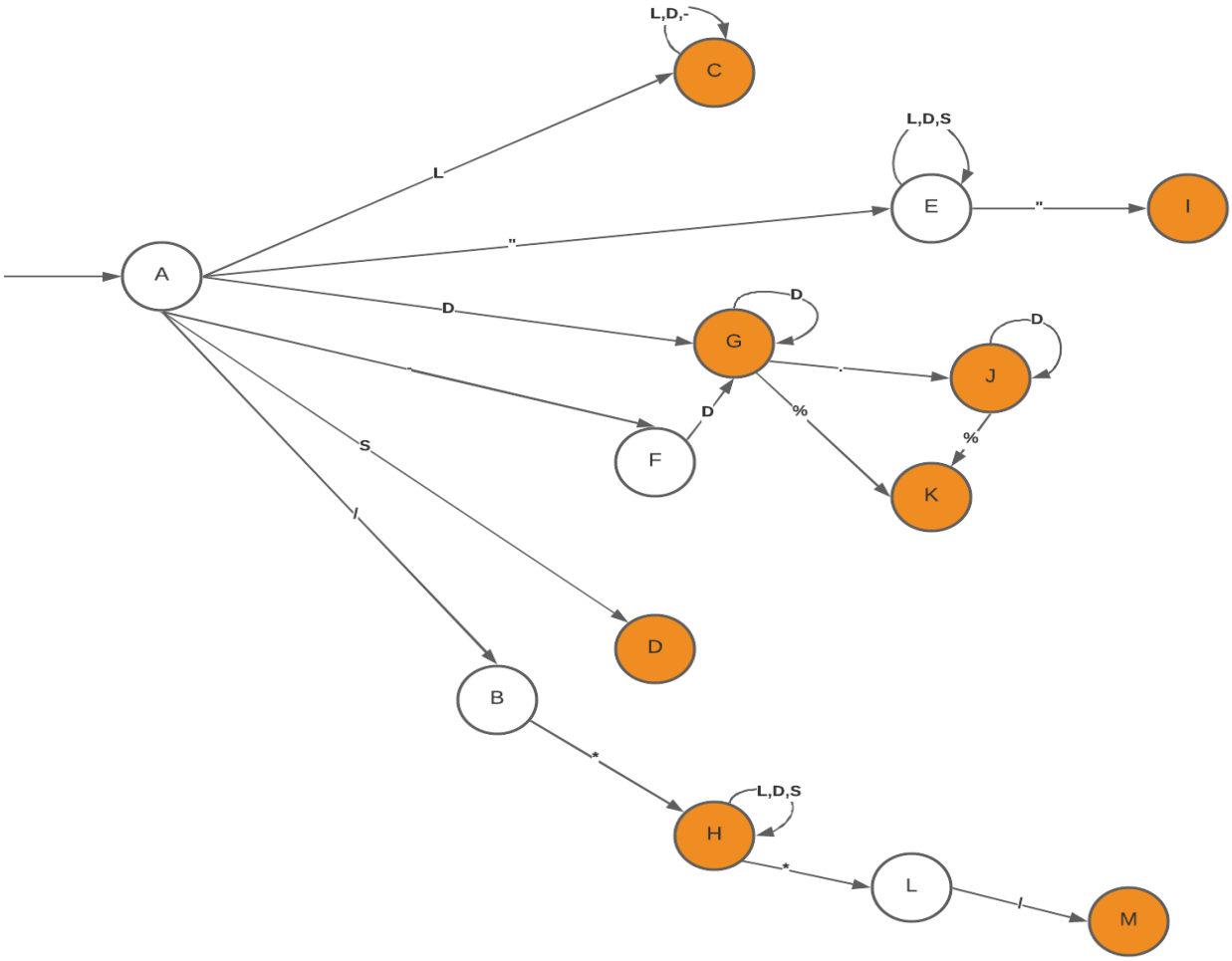
Después de haber realizado el árbol de las expresiones regulares procedemos a realizar la tabla de siguientes, respetando las reglas del método del árbol, las cuales nos dicen que solo se calculan los siguientes de las concatenaciones, las cerraduras positivas y las cerraduras de Kleen.

ALFABETO	NODO	FOLOW
*	1	2
"/"	2	3,4,5,6
D	3	3,4,5,6
L	4	3,4,5,6
S	5	3,4,5,6
*	6	7
"/"	7	8
#	8	-
L	9	10,11,12,14
L	10	10,11,12,14
D	11	10,11,12,14
-	12	10,11,12,14
#	14	-
S	15	16
#	16	-
"	17	18,19,20,21
L	18	18,19,20,21
D	19	18,19,20,21
S	20	18,19,20,21
"	21	22
#	22	-
-	23	24
D	24	24,25,26,27,28
.	25	26,27,28
D	26	26,27,28
%	27	28
#	28	-

Luego de calcular la tabla de siguientes analizamos la raíz del árbol que generamos anterior mente y obtenemos la siguiente tabla de transiciones

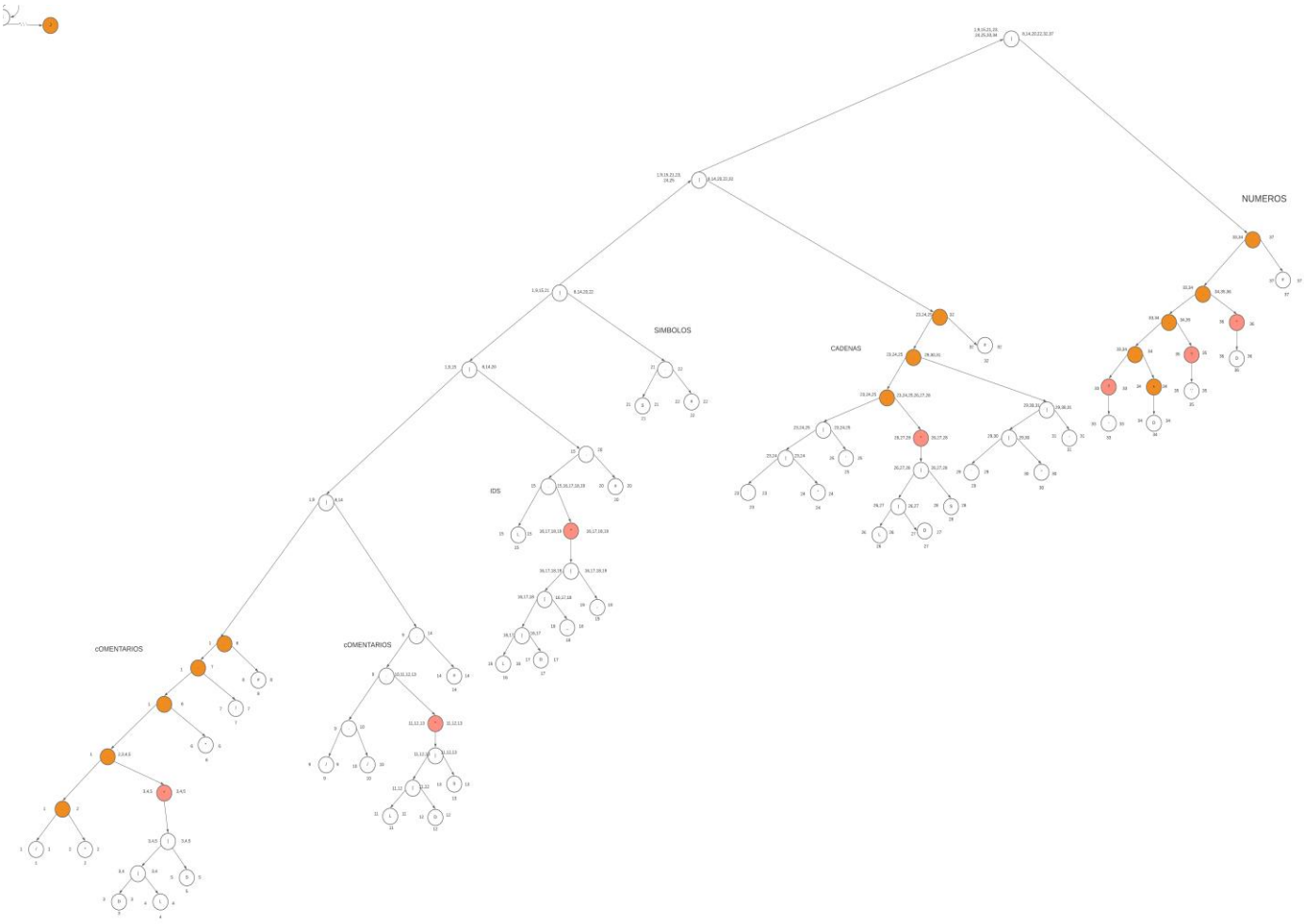
	ESTADOS	L	D	S	*	"/"	"	%	-	.
1,9,15,17,23,24	A	C= 10,11,12,14	G= 24,25,26,27,28	D= 16	B= 2		E =18,19,20,21		F= 24	
2	B					H= 3,4,5,6				
10,11,12,14	*C	C= 10,11,12,14	C= 10,11,12,14						C= 10,11,12,14	
16	*D									
18,19,20,21	E	E= 18,19,20,21	E= 18,19,20,21	E= 18,19,20,21			I= 22			
24	F		G= 24,25,26,27,28							
24,25,26,27,28	*G		G= 24,25,26,27,28					K= 28		J= 26,27,28
3,4,5,6	*H	H= 3,4,5,6	H= 3,4,5,6	H= 3,4,5,6	L=7					
22	*I									
26,27,28	*J		J= 26,27,28					K= 28		
28	*K									
7	L					M= 8				
8	*M									

Con esta tabla de transiciones final, obtenemos el autómata finito determinista que utilizamos para analizar el lenguaje css.



Archivos Js

Para el análisis de los archivos js se realizó el método del árbol comenzando con el árbol de las expresiones regulares para el lenguaje de js.



Después de haber realizado el árbol de las expresiones regulares procedemos a realizar la tabla de siguientes, respetando las reglas del método del árbol, las cuales nos dicen que solo se calculan los siguientes de las concatenaciones, las cerraduras positivas y las cerraduras de Kleen.

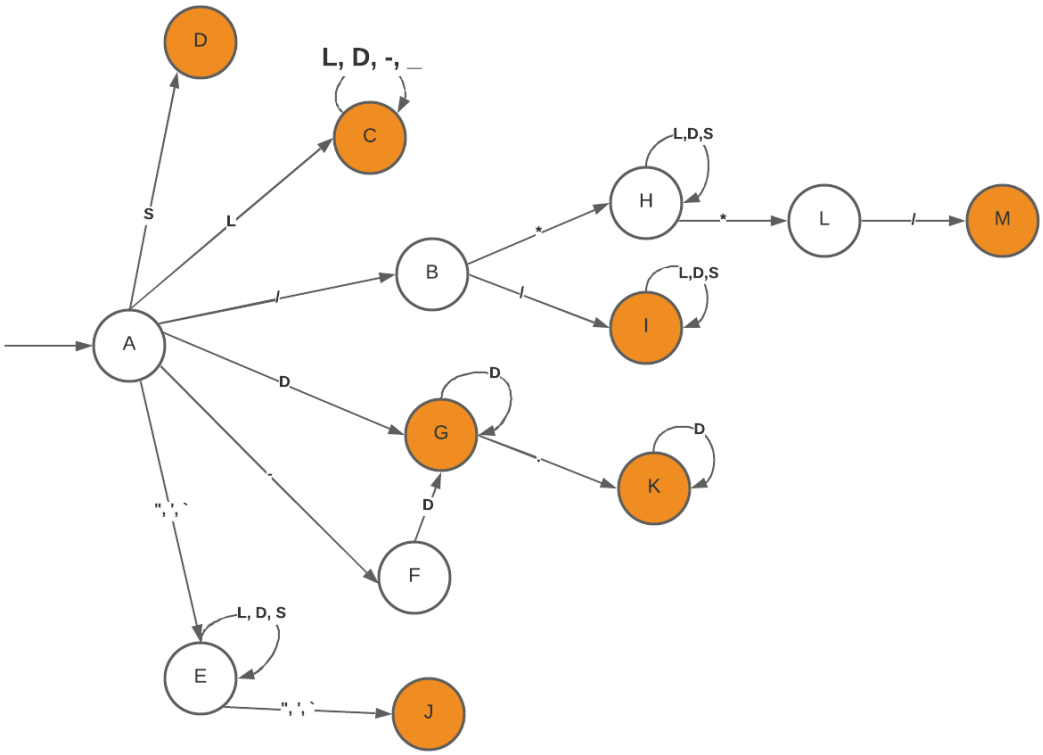
ALFABETO	NODO	FOLOW
"/"	1	2
*	2	3,4,5,6
D	3	3,4,5,6
L	4	3,4,5,6
S	5	3,4,5,6
*	6	7
"/"	7	8

#	8	-
"/"	9	10
"/"	10	11,12,13,14
L	11	11,12,13,14
D	12	11,12,13,14
S	13	11,12,13,14
#	14	-
L	15	16,17,18,19,20
L	16	16,17,18,19,20
D	17	16,17,18,19,20
_	18	16,17,18,19,20
-	19	16,17,18,19,20
#	20	-
S	21	22
#	22	-
`	23	26,27,28,29,30,31
"	24	26,27,28,29,30,31
'	25	26,27,28,29,30,31
L	26	26,27,28,29,30,31
D	27	26,27,28,29,30,31
S	28	26,27,28,29,30,31
`	29	32
"	30	32
'	31	32
#	32	-
-	33	34
D	34	34,35,36,37
.	35	36,37
D	36	36,37
#	37	-

Luego de calcular la tabla de siguientes analizamos la raíz del árbol que generamos anterior mente y obtenemos la siguiente tabla de transiciones

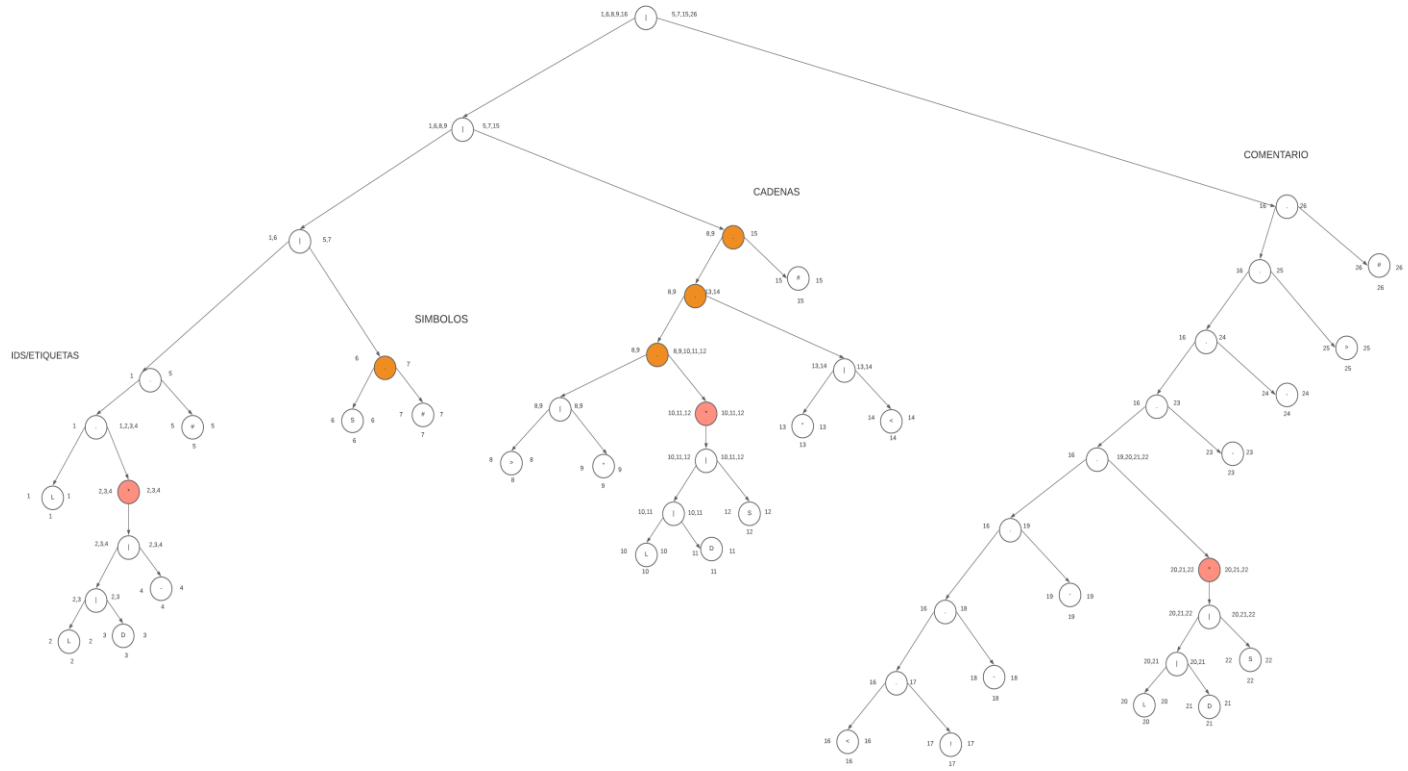
	ESTADOS	L	D	S	*	"/"	"	'	`	-	.	_
1,9,15,21,23, 24,25,33,34	A	C= 16,17,18,19,20	G= 34,35,36,37	D= 22		B= 2,10	E= 26,27,28,29,30,31	E= 26,27,28,29,30,31	E= 26,27,28,29,30,31	F= 34		
2,10	B				H= 3,4,5,6	I= 11,12,13,14						
16,17,18,19,20	*C	C= 16,17,18,19,20	C= 16,17,18,19,20							C= 16,17,18,19,20		C= 16,17,18,19,20
22	*D											
26,27,28,29,30,31	E	E= 26,27,28,29,30,31	E= 26,27,28,29,30,31	E= 26,27,28,29,30,31			J= 32	J= 32	J= 32			
34	F		G= 34,35,36,37									
34,35,36,37	*G		G= 34,35,36,37								K= 36,37	
3,4,5,6	H	H= 3,4,5,6	H= 3,4,5,6	H= 3,4,5,6	L= 7							
11,12,13,14	*I	I= 11,12,13,14	I= 11,12,13,14	I= 11,12,13,14								
32	*J											
36,37	*K		K= 36,37									
7	L					M= 8						
8	*M											

Con esta tabla de transiciones final, obtenemos el autómata finito determinista que utilizamos para analizar el lenguaje js.



Archivos Html

Para el análisis de los archivos html se realizó el método del árbol comenzando con el árbol de las expresiones regulares para el lenguaje de html.



Después de haber realizado el árbol de las expresiones regulares procedemos a realizar la tabla de siguientes, respetando las reglas del método del árbol, las cuales nos dicen que solo se calculan los siguientes de las concatenaciones, las cerraduras positivas y las cerraduras de Kleen.

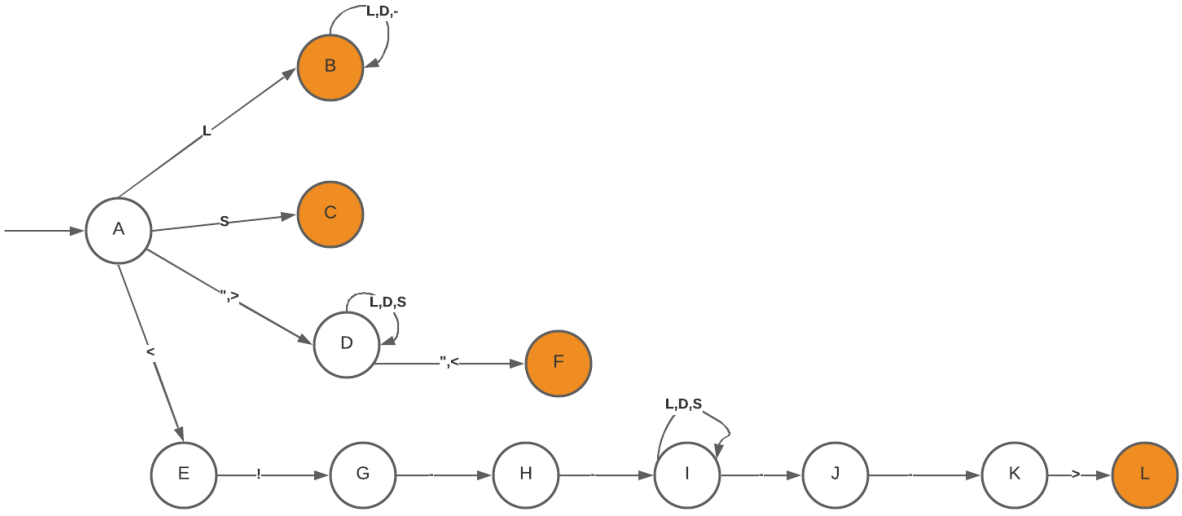
ALFABETO	NODO	FOLOW
L	1	2,3,4,5
L	2	2,3,4,5
D	3	2,3,4,5
-	4	2,3,4,5
#	5	-
S	6	7
#	7	-
>	8	10,11,12,13,14
"	9	10,11,12,13,14
L	10	10,11,12,13,14
D	11	10,11,12,13,14
S	12	10,11,12,13,14
"	13	15
<	14	15
#	15	-

<	16	17
!	17	18
-	18	19
-	19	20,21,22,23
L	20	20,21,22,23
D	21	20,21,22,23
S	22	20,21,22,23
-	23	24
-	24	25
>	25	26
#	26	-

Luego de calcular la tabla de siguientes analizamos la raíz del árbol que generamos anterior mente y obtenemos la siguiente tabla de transiciones

	ESTADOS	L	D	S	"	-	!	<	>
1,6,8,9,16	A	B= 2,3,4,5		C= 7	D= 10,11,12,13,14			E= 17	D= 10,11,12,13,14
2,3,4,5	*B	B= 2,3,4,5	B= 2,3,4,5			B= 2,3,4,5			
7	*C								
10,11,12,13,14	D	D= 10,11,12,13,14	D= 10,11,12,13,14	D= 10,11,12,13,14	F=15			F=15	
17	E						G= 18		
15	*F								
18	G					H=19			
19	H					I= 20,21,22,23			
20,21,22,23	I	I= 20,21,22,23	I= 20,21,22,23	I= 20,21,22,23		J= 24			
24	J					K=25			
25	K								L=26
26	*L								

Con esta tabla de transiciones final, obtenemos el autómata finito determinista que utilizamos para analizar el lenguaje html.



Programación de Estados

Para la programación de estados, en este proyecto se utilizaron métodos en el lenguaje de programación Python, también en algunos estados que se requieran se utilizan sentencias while para generar un bucle mientras se va pasando de carácter en carácter la cadena de entrada e ir analizando cada carácter, hasta llegar a un estado de aceptación y agregar el token a la lista de tokens aceptados, o hasta llegar a un estado de error y también agregarlo a una lista especial para errores, mostrando la fila y columna del error para luego corregirlo.

```
def estadoA(self,entrada, consola):
    self.cadena = entrada + "$"
    self.caracterActual = ""
    self.borrarTokenYErrores()


    while self.posicionCar < len(self.cadena):
        self.caracterActual = self.cadena[self.posicionCar]
        self.bitacora +=" ->estadoA"

        #estado A a estado B para comentarios
        if self.caracterActual == "/":
            # self.addToken(Tipo.DIAGONAL, "/")  #mandar a estado B por comentario
            self.estadoB(self.posicionCar, consola)
        elif self.caracterActual == "{":
            self.bitacora +=" ->estadoD"
```

En este caso pondremos de ejemplo el estado a del analizador léxico para el lenguaje Css; declaramos el estado a con una entrada que sería la cadena de entrada del archivo el cual vamos a analizar, también

solicitamos un parámetro consola para poder imprimir en consola los errores léxicos que vallamos encontrando a lo largo del análisis; después, concatenamos a la cadena un símbolo que nos indicara el fin del archivo. Declaramos una variable con la cual manejaremos el carácter en el cual nos encontramos situados, sería una especia de apuntador. Luego encontramos la sentencia while que como se explicó antes nos ayuda a ir analizando carácter a carácter hasta encontrar el fin del archivo.

En la primer línea luego de declarado el while es el método por el cual iremos avanzando en la cadena de entrada el cual nos indica que iremos cambiando de carácter por la posición del carácter que ira aumentando conforme vamos analizando.

Luego las sentencias if que nos ayudan a analizar qué tipos de caracteres son permitidos.

```
#estado A a estado G (Numeros)
elif self.caracterActual.isnumeric():
    sizeLexema = self.getSizeLexema(self.posicionCar)
    self.estadoG(self.posicionCar,self.posicionCar+sizeLexema , consola)
    self.posicionCar = self.posicionCar + sizeLexema

#estado A a estado C (Reservadas e IDs)
elif self.caracterActual.isalpha():
    sizeLexema = self.getSizeLexema(self.posicionCar)
    self.estadoC(self.posicionCar,self.posicionCar+sizeLexema, consola)
    #self.reservadas(posicionCar, posicionCar+sizeLexema)
    self.posicionCar = self.posicionCar + sizeLexema
```

Para verificar si es un numero utilizamos el método isnumeric que nos devuelve un booleano y este nos ayuda a verificar si es o no un número, lo mismo pasa con el método isalpha que nos devuelve un booleano y este nos ayuda a verificar si en efecto es una letra o no.

```
    else:
        self.addError(self.columna,self.fila, self.caracterActual)
        print("Error Lexico: ", self.caracterActual)
        consola.insert('1.0', "Error Lexico: "+self.caracterActual+"\n")
        for i in range(0,len(self.listaErrores)):
            valor = self.listaErrores[i].getValor()
            valor += str(self.listaErrores[i].getColumna())
            valor += str(self.listaErrores[i].getFila())
            print(valor)
self.posicionCar +=1 #incremento contador while
```

Para el manejo de errores tenemos al final del while luego de pasar por todas las sentencias y no acceder a ellas exitosamente entra a este estado sumidero en el cual manejamos los errores para luego seguir analizando el archivo y así poder devolver si hubo o no errores léxicos. Por ultimo tenemos la variable de control para el while la cual va aumentando con forme vamos analizando.

```
if len(self.listaErrores)>0:
    reporte = reporteHtml()
    reporte.reporteEnHtml(self.listaErrores,self.rutaDestino1)
    reporte.vistaTokens(self.listaTokens,self.rutaDestino1)
    return "La entrada que ingresaste fue: Exiten Errores Lexicos"
else:
    reporte.vistaTokens(self.listaTokens,self.rutaDestino1)
    return "La entrada que ingresaste fue:" + self.cadena + "\n Analisis Exitoso"
```

Al final del análisis del archivo tenemos unas sentencias if para comprobar si hay errores léxicos, si los hubiera se generan la tabla de errores en un archivo html que luego en la clase del grafico podremos acceder y ver en el buscador predeterminado. Si no hubiera errores léxicos solamente se generan la tabla de tokens que de igual forma podremos acceder en la interfaz gráfica.

```
def addToken(self, tipo, valor):

    nuevo = Token(tipo, valor)
    self.listaTokens.append(nuevo)
    self.caracterActual = ""
    self.estado = 0
    self.lexema = ""

def addError(self, columna, fila, valor):

    nuevo = Error(columna, fila, valor)
    self.listaErrores.append(nuevo)
    #puede que tenga que agregar algo
```

Para agregar un token o un error a la lista tenemos estos métodos los cuales añaden los tokens o errores a sus respectivas listas haciendo uso del método append()

Clase Token

```
class Tipo(Enum):
    #Simbolos
    LLAVEIZQ = 1
    LLAVEDER = 2
    DPUNTOS = 3
    PCOMA = 4
    COMA = 5
    PUNTO = 6
    NUMERAL = 7
    PARENTESISIZQ = 8
    PARENTESISDER = 9
    COMILLAS = 10
    ASTERISCO = 11
    GUION = 12
```

Para la clase token de cada lenguaje tenemos una clase tipo la cual utilizamos para agregar todos los tokens que son validos en el lenguaje.

```
class Token:
    tipoToken = Tipo.NINGUNO
    valorToken = ""
    def __init__(self, tipo, valor ):
        self.tipoToken = tipo
        self.valorToken = valor

    def getTipo(self):
        return self.tipoToken

    def getValor(self):
        return self.valorToken

class Error:
    columna = 0
    fila = 0
    valor = ""
    def __init__(self,columna, fila,valor):
        self.columna = columna
        self.fila= fila
        self.valor = valor

    def getColumna(self):
        return self.columna
    def getFila(self):
        return self.fila
    def getValor(self):
        return self.valor
```

y tenemos las clases token y las clases errores las cuales nos encapsulan los métodos que el token tiene y que los errores tienen.

Grafico

```
class Grafico:
    bitacora=""
    filas=1
    rutaDestino=""
    grafoCadena=""
    grafoComentario=""
    grafoNumero=""
    def __init__(self):
        repBitacora=""
        rutaDestino=""
        grafoCadena=""
        grafoComentario=""
        grafoNumero=""
        self.ventana = Tk()
        self.ventana.geometry("1000x600")
        self.ventana.title(" [OLC1] Proyecto 1" )
        #self.ventana.configure(bg = '#73AB85')
        self.ventana.configure(bg = '#F5A903')

        self.menu = Menu(self.ventana)

        self.archivo_item = Menu(self.ventana)
        self.archivo_item.add_command(label="Nuevo", command=self.abrir)
```

Para el apartado grafico de la aplicación se utilizo la librería Tkinter con todos sus métodos que son muy útiles.

```
def analisar(self):
    valor= self.combo.get()
    self.txtConsola.delete('1.0',END)
    entrada = self.txtEntrada.get('1.0', END)
    if valor.lower() =="css":
        scaner = ScannerCss()
        retorno = scaner.estadoA(entrada, self.txtConsola)
        self.bitacora = scaner.reporteBitacora()
        self.filas= scaner.getFilas()
        self.llenarFilas()
        self.rutaDestino=scaner.rutaDestino()
        self.pathD(self.rutaDestino)
        self.txtConsola.insert(END,retorno)
        messagebox.showinfo('Proyecto-1', 'Analisis Finalizado')
    elif valor.lower() =="js":
        scaner = ScannerJs()
        retorno = scaner.estadoA(entrada, self.txtConsola)
        fila= scaner.getFilas()
        self.rutaDestino=scaner.rutaDestino()
        self.grafoCadena= scaner.getPrimerCadena()
        self.grafoComentario= scaner.getPrimerComentario()
        self.grafoNumero= scaner.getPrimerNumero()
        self.pathD(self.rutaDestino)
        self.txtConsola.insert(END,retorno)
        messagebox.showinfo('Proyecto-1', 'Analisis Finalizado')
    elif valor.lower() =="html":
        scaner = ScannerHtml()
        retorno = scaner.estadoA(entrada, self.txtConsola)
        fila= scaner.getFilas()
        self.rutaDestino=scaner.rutaDestino()
        self.pathD(self.rutaDestino)
```

para el selector de lenguaje a analizar se utilizó un combo box el cual nos ayuda a definir cual lenguaje vamos a analizar, o que lenguaje desea analizar el cliente.

Reporte Html

```
class reporteHtml:
    columna= 0
    fila = 0
    listaErrores= list()
    directorio= ""

    def __init__(self):
        self.columna=0
        self.fila=0
        self.listaErrores= list()
        directorio= ""

    def pathD(self,ruta):
        print(str(ruta))
        if not os.path.isdir(str(ruta)):
            os.makedirs(str(ruta))

    def reporteEnHtml(self, lista,ruta):
        self.pathD(ruta)
        self.listaErrores = lista
        f = open(ruta+"/reporteErrores.html","w+")

        mensaje = """<html>

<head>
```

Para el reporte en html tenemos una clase que nos crea un archivo html y luego lo rellena con los tokens obtenidos, y los errores obtenidos del análisis que se realizó anteriormente. También se tienen manejos de errores por si no se ha hecho ningún análisis anteriormente.

Reporte Js

```
class Rparbol:
    variable=""
    def __init__(self):
        self.variable=""

    def comando(self,ruta,grafo,nombre):
        self.graficar(ruta,grafo,nombre)
        subprocess.call(['cd',ruta, '&&'  'dot', '-Tpng',  nombre+'.txt', '-o', nombre+'.png'],shell=True)

    def graficar(self,ruta,grafo,nombre):
        f = open(ruta+"/"+nombre+".txt","w+")
        mensaje= """ digraph G{"""
        mensaje+=str(grafo)

        mensaje+="""}"""
        f.write(mensaje)
        f.close()
```

Para el análisis del lenguaje Js tenemos una clase que nos genera una grafica, a travez de graphviz, la cual vine de ir concatenando los estados por los cuales pasa, las cadenas, los números y los comentarios.

Reporte Css

```
self.bitacora += "->estadoB"
```

Para el reporte del lenguaje css solamente se va concatenando los estados por los cuales va pasando cada uno de los caracteres del lenguaje mientras se va analizando hasta el final del análisis y luego podemos acceder a la cadena y mostrarla en consola.