PROYECTO 2 LFP

Manual Técnico

USAC – Facultad de ingeniería
Damián Ignacio Peña Afre

CONTENIDO

ENCABEZADO

- Nombre del sistema: Compilador a HTML
- Desarrollador: Damián Ignacio Peña Afre
- Lenguaje utilizado: Python
- Tipo de programa: Desktop
- Cuenta con interfaz gráfica

TECNICAS O PARADIGMAS

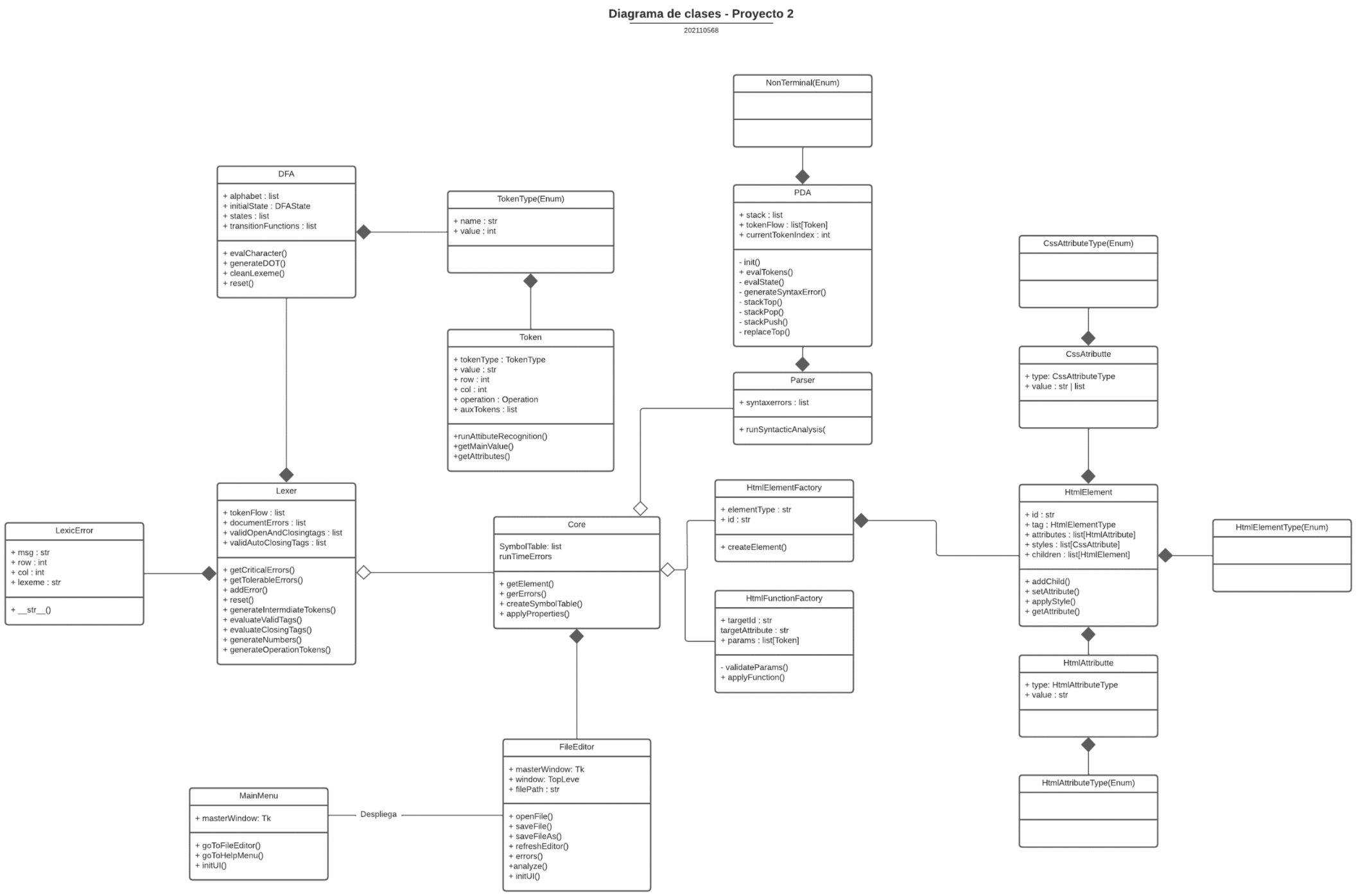
Se utilizo el paradigma de la programación orientada a objetos junto con la separación de los archivos en carpetas en función del tipo de tarea que realizan, de esta manera el proyecto esta segmentado de una forma modular.

Se separó la lógica del aspecto visual, de modo que obtiene una estructura más limpia de código.

CONVENCIONES DE NOMENCLATURA

Se optó por utilizar '*camelcase*' para nombrar los archivos y variables. Tanto el nombre de la clase como el archivo que lo contiene empiezan con letra mayúscula. Los métodos/funciones y archivos que contienen únicamente métodos/funciones empiezan por letra minúscula.

DIAGRAMA DE CLASES



METODOS PRINCIPALES

1. *getNextToken y evalCharacter*

Dentro de la clase que define al autómata encargado del analisis léxico (DFA) se encuentra estos métodos que se encargan de determinar si existe alguna transición con el carácter recibido en la cadena de entrada o programa fuente, llevar un conteo de las filas y columnas, omitir las líneas con comentarios, así como de generar los tokens de los elementos reconocidos.

```
def getNextToken(self):

        while self.currentIndex < len(self.stringFlow):
            token = None
            currentCharacter = self.stringFlow[self.currentIndex]

            # Comments

            # * Skips line
            if self.skipLine:
                self.currentIndex += 1
                self.col += 1

                if currentCharacter == '\n':
                    self.row += 1
                    self.col = 1
                    self.skipLine = False

                continue

            # * Skips line breaks
            if self.skipLineBreaks and currentCharacter == '\n':
                token = self._evalAcceptanceState()
                self.row += 1
                self.col = 1
                self.currentIndex += 1

            if token == None:
                token = self._evalCharacter()

            if token == True:
                self.col += 1
                self.currentIndex += 1
                continue
            if isinstance(token, Token):

                if token.tokenType == TokenType.ONELINE_COMMENT_OPENING:
                    self.skipLine = True
                    # self.col += 1
                    # self.currentIndex += 1
                    continue

                if token.tokenType == TokenType.MULTILINE_COMMENT_OPENING:
                    self.skipToken = True
                    # self.col += 1
                    # self.currentIndex += 1
                    continue

                if token.tokenType == TokenType.MULTILINE_COMMENT_CLOSING:
                    self.skipToken = False
                    # self.col += 1
                    # self.currentIndex += 1
                    continue

                if self.skipToken:
                    # self.col += 1
                    # self.currentIndex += 1
```

```
                continue

            return token

        if isinstance(token, LexicError):
            return token

    if self.lexeme != '':
        return self._evalAcceptanceState()

    return None

# True -> valid state/character
# False -> invalid state/character
# Token -> valid token/lexeme
# LexicError -> invalid token/lexeme
def _evalCharacter(self):
    currentCharacter = self.stringFlow[self.currentIndex]

    if currentCharacter == ' ':

        if self.skipBlankSpaces:
            token = self._evalAcceptanceState()
            self.col += 1
            self.currentIndex += 1
            return token
        else:
            self.lexeme += currentCharacter
            return True

    #  ...
```

2. evalTokens y evalState

Luego de haber pasado por el analizar léxico, el PDA o autómata de pila es el encargado de revisar el orden de los tokens dentro del flujo de tokens. Para ello almacena una serie de símbolos no terminales y terminales dentro de una pila, de manera tal, que al vaciar completamente esta pila y no haber mas tokens validar el flujo.

```
def evalTokens(self):
        while self._stackTop() != EOF_TOKEN:
            resp = self._evalState()

            if isinstance(resp, SyntaxError):
                return resp

        # POP EOF
        self._stackPop()

        if self.currentTokenIndex < len(self.tokenFlow):
            # !
            return SyntaxError(self.tokenFlow[self.currentTokenIndex], [EOF_TOKEN])

        return None

    def _evalState(self):
        evaluatedToken = EMPTY_TOKEN
        stackTop = self._stackTop()

        try:
            evaluatedToken = self.tokenFlow[self.currentTokenIndex]
        except IndexError:
            if not isinstance(stackTop, NonTerminal):
                evaluatedToken.row = self.tokenFlow[self.currentTokenIndex-1].row
                evaluatedToken.col = self.tokenFlow[self.currentTokenIndex-1].col
                return SyntaxError(evaluatedToken, [stackTop])
```

```
        # * NonTerminal replacement rules

        # S -> 4 11 A 11 5 4 11 B 11 5 4 11 C 11 5
        if stackTop == NonTerminal.S:

            replace = [
                OPEN_SCOPE_TOKEN,
                CONTROL_SCOPE_TOKEN,
                NonTerminal.A,
                CONTROL_SCOPE_TOKEN,
                CLOSE_SCOPE_TOKEN,
                OPEN_SCOPE_TOKEN,
                PROPERTIES_SCOPE_TOKEN,
                NonTerminal.B,
                PROPERTIES_SCOPE_TOKEN,
                CLOSE_SCOPE_TOKEN,
                OPEN_SCOPE_TOKEN,
                LOCATION_SCOPE_TOKEN,
                NonTerminal.B,
                LOCATION_SCOPE_TOKEN,
                CLOSE_SCOPE_TOKEN,
            ]
            self._replaceTop(replace)
            return True

        # A -> 11 11 7 A | epsilon
        if stackTop == NonTerminal.A:
            # ! Preanalysis
            if evaluatedToken.tokenType == TokenType.ID and evaluatedToken.lexeme !=
CONTROL_SCOPE_TOKEN.lexeme and evaluatedToken.lexeme != PROPERTIES_SCOPE_TOKEN.lexeme and
evaluatedToken.lexeme != LOCATION_SCOPE_TOKEN.lexeme:
                replace = [
                    VALID_CONTROL_TOKEN,
                    VALID_VARIABLE_TOKEN,
                    SEMICOLON_TOKEN,
                    NonTerminal.A
                ]
                self._replaceTop(replace)
                return True

            # | epsilon
            self._stackPop()
            return True

    # * Terminal replacement rules

        # 4 -> <!--
        if stackTop == OPEN_SCOPE_TOKEN and evaluatedToken.tokenType ==
TokenType.DEFINITION_SCOPE_OPENING:
            self._stackPop()
            self.currentTokenIndex += 1
            return True

        # 5 -> -->
        if stackTop == CLOSE_SCOPE_TOKEN and evaluatedToken.tokenType ==
TokenType.DEFINITION_SCOPE_CLOSING:
            self._stackPop()
            self.currentTokenIndex += 1
            return True
```

3. createSymbolTable y applyProperties

Luego de procesar sintácticamente el flujo de tokens se construye la tabla de símbolos, donde cada token de tipo variable es inicializado como un elemento HTML gracias a la clase HtmlElementFactory. Luego de que se haya inicializado cada uno de estos objetos se procede a leer los 2 bloques restantes para aplicar o modificar sus propiedades.

```
def createSymbolTable(self):

        # this element is the root of the html tree
        Core.SymbolTable.append(HtmlElement("this", HtmlElementType.BODY))

        # * analyze the controls scope

        index = 0
        while index < len(Lexer.tokenFlow):
            token = Lexer.tokenFlow[index]

            if token.idType == idType.CONTROL:
                nextToken = Lexer.tokenFlow[index + 1]
                element = HtmlElementFactory(token.lexeme, nextToken.lexeme)
                Core.SymbolTable.append(element.createElement())

            index += 1

    def applyProperties(self):

        # Skip definition scope
        index = 0
        while index < len(Lexer.tokenFlow):
            token = Lexer.tokenFlow[index]

            if token.tokenType == TokenType.DEFINITION_SCOPE_CLOSING:
                index += 1
                break
            index += 1

        while index < len(Lexer.tokenFlow):
            tokenFlow = Lexer.tokenFlow

            if tokenFlow[index].idType == idType.VARIABLE:
                targetId = tokenFlow[index].lexeme
                targetAttribute = tokenFlow[index + 2].lexeme
                params = []

                # parameters
                for j in range(index+4, len(tokenFlow)):

                    if tokenFlow[j].tokenType == TokenType.COMMA:
                        continue

                    if tokenFlow[j].tokenType == TokenType.PARENTHESIS_CLOSING:
                        index = j
                        break

                    params.append(tokenFlow[j])

                # Execute function
                function = HtmlFunctionFactory(targetId, targetAttribute, params)
                res = function.applyFunction()
                if res != True:
                    Core.runTimeErrors.append({
                        'type': 'runtime',
                        'row': str(tokenFlow[index].row),
                        'column':  str(tokenFlow[index].col),
                        'lexema': "",
                        'expected': "",
                        'description': res
                    })

            index += 1
```

REQUERIMIENTOS

- Contar con un menú de opciones
- Cargar un archivo con extensión GPW
- Editar el archivo fuente
- Analizar el archivo fuente
- Mostrar los errores del archivo fuente

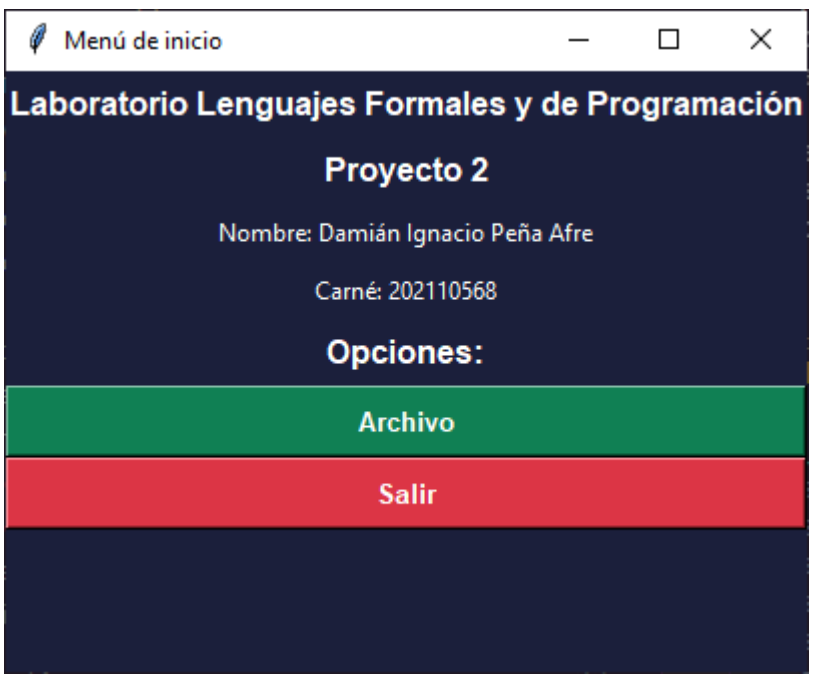
DESCRIPCIÓN

Para la realización del proyecto se utilizaron las siguientes herramientas:

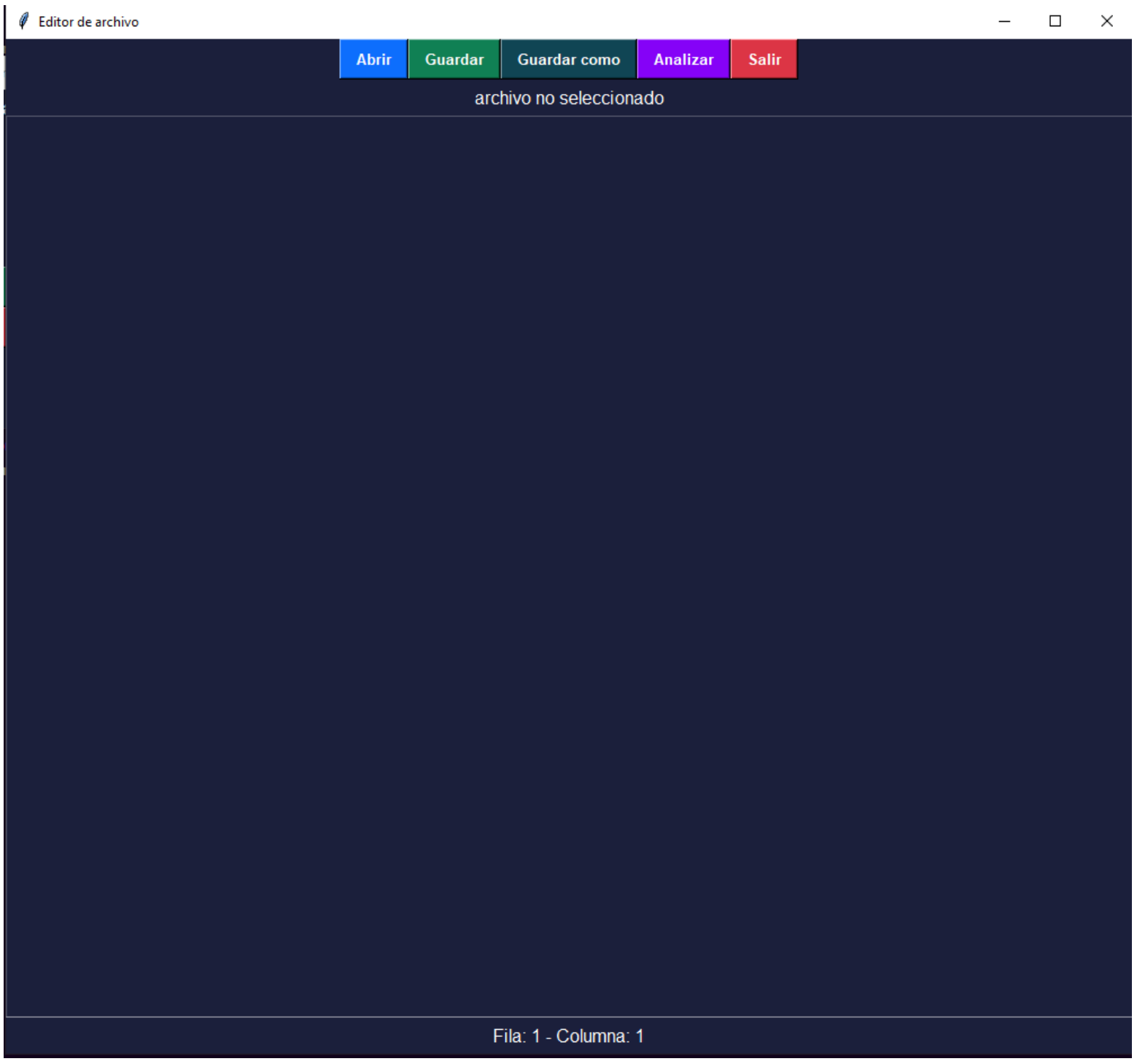
1. Visual Studio Code: Editor de código ligero.
2. Git: Programa para llevar un control de las versiones del proyecto.
3. Github: Repositorio remoto para llevar un control de las versiones del proyecto en la nube.

NTERFACES PRINCIPALES

1. Menu principal



2. Editor de archivo



Se toma el contenido dentro del área de texto para sobrescribir o crear un archivo. Es posible también analizar el contenido de dicho archivo con el botón de analizar, en dado caso se presenten errores, es posible también, presionar el botón de errores para generar dicho reporte.

PLANIFICACIÓN

Día	Tarea	Tiempo empleado (Horas)
Domingo 16	Diagramación y diseño de autómatas	3
Lunes 17	Analizador léxico	3
Martes 18	Corrección de errores analizador léxico	2
Miercoles 19	Analizador sintáctico	3
Jueves 20	Corrección de errores analizador sintáctico	1
Viernes 21	Generación de elementos HTML	3
Sabado 22	Generación de reportes a partir de elementos Html	2
Jueves 27	Ultimas correcciones, ajustes y documentación	2
Viernes 28	Entrega	-

ANEXOS

Tokens

L = Set of letters
N = Set of numbers
P = parameters = {ID, string, boolean, number}
e = épsilon

No	REGEX	Name	Aditional Props
1	//	oneline_comment_opening	
2	/*	multiline_comment_opening	
3	*/	multiline_comment_closing	
4	<!--	definition_scope_opening	Scope: Controles[1] | propiedades[2] | Colocacion[3]
5	-->	definition_scope_closing	Scope: Controles[1] | propiedades[2] | Colocacion[3]
6	.	point	
7	;	semicolon	
8	true | false	boolean	
9	N+	number	
10	“(L|N)*”	String	
11	L(N|L)*	id	Type: *Control*[1] | variable[2] | this[3] | *function*[4] *| scope*[5]
12	(	parenthesis_opening	
13	)	parenthesis_closing	
14	,	Comma	

Gramatica

e → épsilon

Initial state

So → $4^{1}11^{5}$ A $11^{5}5^{1}$ $4^{2}11^{5}$ B $11^{5}5^{2}$ $4^{3}11^{5}$ C $11^{5}5^{3}$

Control scope

A → $11^{\rightarrow 1}11^{\rightarrow 2}$ 7 A

| e

Properties Scope

B → $11^{2\,|\,3}$ 6 $11^{\rightarrow 4}$ 12 D 13 7 B

| e

D → 8 E

| 9 E

| 10 E

| 11 E

E → 14 D

| e

Location Scope

C → $11^{2\,|\,3}$ 6 $11^{\rightarrow 4}$ 12 D 13 7 C

| e

ε, E, ε

ε, E, 14 D

ε, D, 11 E

ε, D, 10 E

ε, D, 9 E

ε, D, 8 E

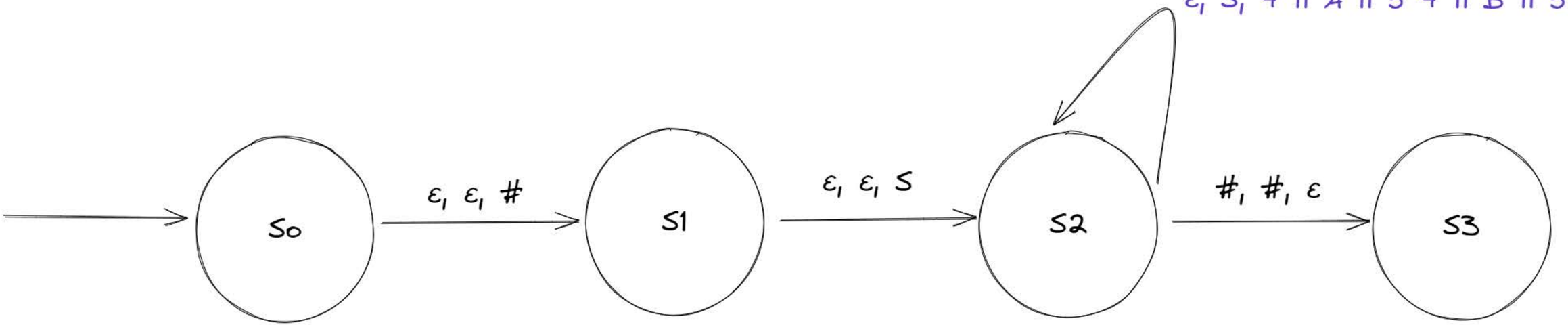
ε, B, ε

ε, B, 11 6 11 12 D 13 7 B

ε, A, ε

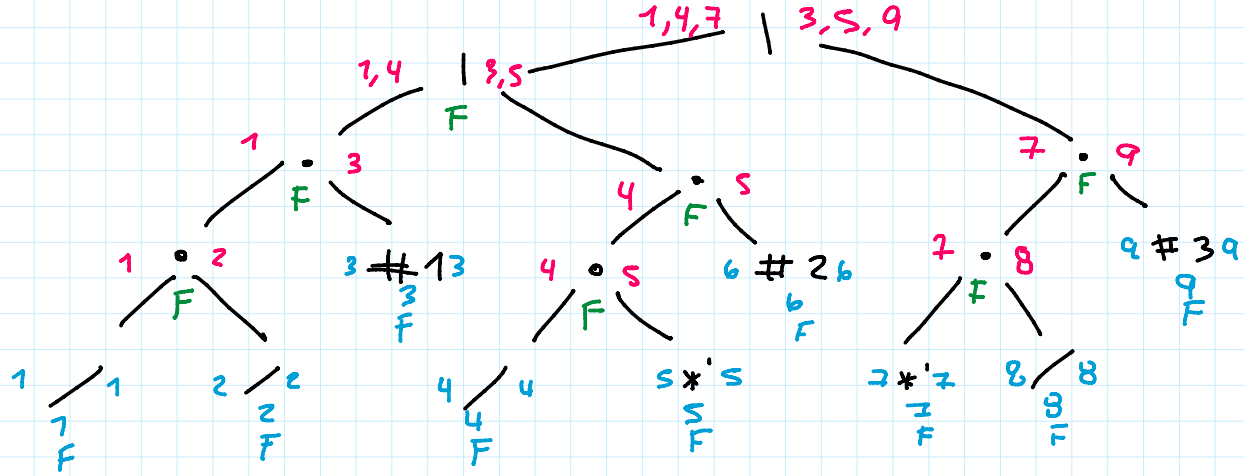
ε, A, 11 11 7 A

ε, S, 4 11 A 11 5 4 11 B 11 5 4 11 B 11 5



* Comentarios

(// #1) | (/ * #2) | (* / #3)
1 2 3 4 5 6 7 8 9



i	Sim	Sig
1	/	2
2	/	3
3	#1	—
4	/	5
5	*	6
6	#2	—
7	*	8
8	/	9
9	#3	

→ $S_0 = \underset{/}{1},\underset{/}{4},\underset{*}{7}$

$T(S_0, /) = sig(1) \cup sig(4)$

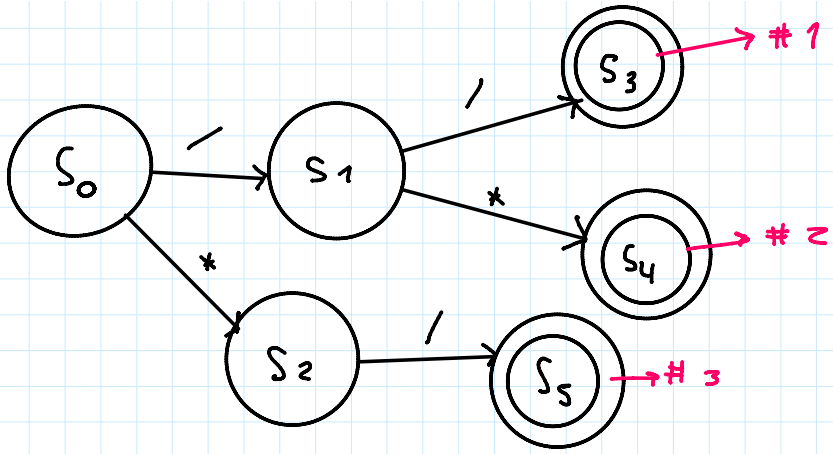
$= \underset{/}{2},\underset{*}{5} = S_1$

$T(S_0, *) = sig(7) = \underset{/}{8} = S_2$

$T(S_1, /) = sig(2) = \underset{\#1}{3} = S_3$

$T(S_1, *) = sig(5) = \underset{\#2}{6} = S_4$

$T(S_2, /) = sig(8) = \underset{\#3}{9} = S_5$

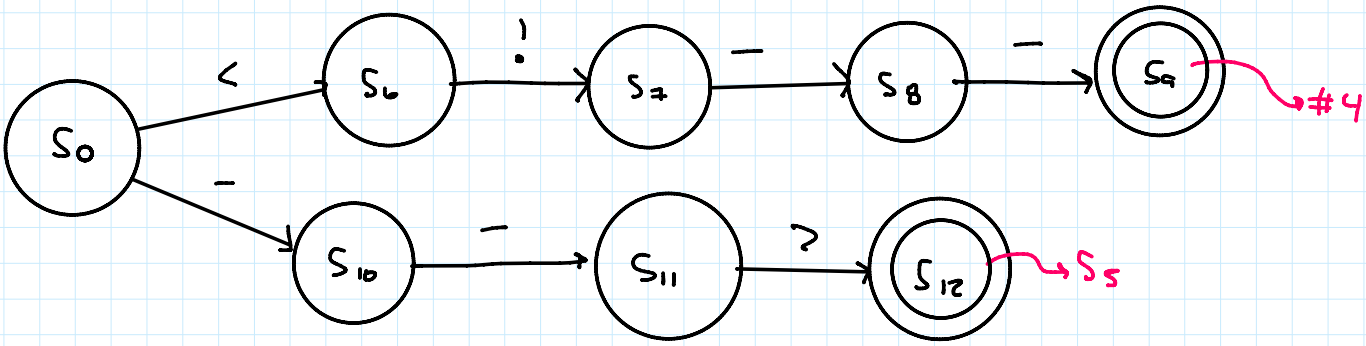


* Etiquetas de cierre

↳ al no tener simbolos en comun con alguna otra regex se puede hacer de forma lineal

(<!-- #4) | (--> #5)

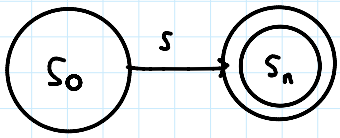
Continuando con la numeracion



* Simbolos adicionales

. ; ( ) ,

Se realizan directamente con un transición desde S_0



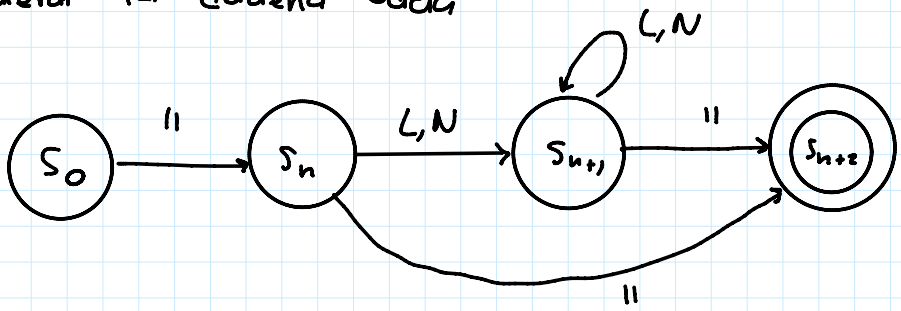
Palabras reservadas

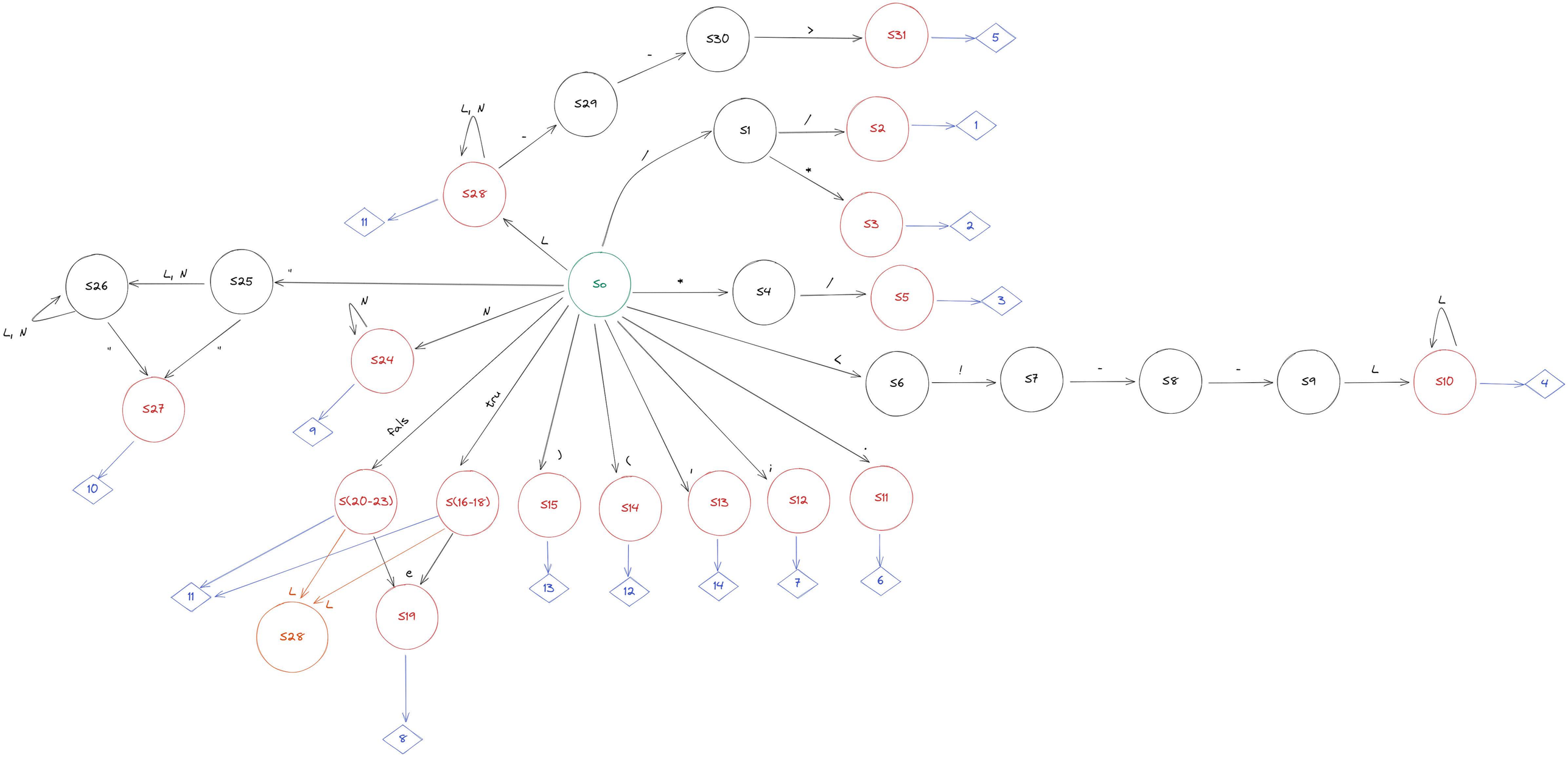
* true | false

↳ al tener simbolos que son parte de L es necesario agregar una transicion para el estado que maneja a #11

String

Considerar la cadena vacía




S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
S16-18
S20-23
S24
S25
S26
S27
S28
S29
S30
S31
S19
S28
1
2
3
4
5
6
7
8
9
10
11
12
13
14
L
N
/
*
-
>
!
<
"
(
)
,
;
.
e
tru
fals