LENGUAJES FORMALES Y
DE PROGRAMACIÓN

MARZO 2022

Manual Técnico

Proyecto 1

UNIVERSIDAD SAN CARLOS DE GUATEMALA

- **Facultad de Ingeniería**
- **Escuela de Ciencias y Sistemas**

ANGELA GABRIELA PINELO FLORES

Carnet: 202002536

ÍNDICE

INTRODUCCIÓN

El siguiente proyecto demuestra en menor medida cómo funciona un analizador Léxico, la implementación de autómatas para el mismo analizador y generamos reportes y formularios en HTML, todo esto es a través de una aplicación generada con python

OBJETIVOS

- Implementar una solución de software implementando los conceptos vistos en clase y laboratorio.
- Implementar un analizador léxico utilizando los conceptos de alfabeto, tokens y sus propiedades.
- Desarrollar una interfaz gráfica utilizando el lenguaje de programación Python.
- Familiarizar la comprensión de HTML y su estructura

REQUERIMIENTOS

- Conocimientos en los siguientes lenguajes: Pyhton, JS, HTML y CSS
- IDE de preferencia Visual Studio Code (Puede no ser necesario)
- Sistema Operativo de preferencia
- Tener instalado Python 3 o posterior
- Tener instaladas las siguientes librerías de Python: TkInter, PrettyTable

ALCANCES DEL PROYECTO

ELa intención del proyecto es que el encargado del desarrollo de esta aplicación aprenda a reconocer caracteres válidos para un lenguaje en específico, por medio de un analizador léxico y posteriormente de su análisis, generar reportes de los tokens válidos, reportes de errores léxicos, y lo principal, un formulario dinámico para que el usuario final pueda llenarlo. Todo esto por medio de la programación, implementando el lenguaje de Python para el análisis léxico, y su salida de formularios aplicando teoría del desarrollo web. Con esto se busca que el desarrollador realice formularios dinámicos, aplicando los conceptos de lenguajes formales.

LÓGICA DEL PROGRAMA

A continuación se presenta una breve descripción de las clases, métodos, funciones y módulos que fueron implementadas para la realización del programa :

- Frame (módulo dónde inicia el programa)
- Analizador(Clase dónde se encuentra todo el análisis)
- Token (Clase de tipo objeto que representa nuestros Tokens)
- Error (Clase de tipo objeto que representa nuestros Errores)

FRAME

Es en esté módulo donde se inicializa nuestro programa con la interfaz gráfica.

Primero las importaciones de librerías y de otros módulos:

```
from tkinter import *
from tkinter import filedialog
from tkinter.filedialog import askopenfilename
from tkinter.filedialog import asksaveasfile
from tkinter import messagebox
from Analisis import Analizador
import sys
```

Luego se inicializa la interfaz utilizando la librería TkInter (botones, cuadro de texto, etc)

```
root=Tk()
root.title("Creador de Formularios")
root.iconbitmap("corazon.ico")
root.config(bg="sky blue")

miFrame=Frame(root, width=500, height=400)
#miFrame.pack()
miFrame.config(bg="light pink")
```

```
botonCargar=Button(root,text="Cargar archivo",command = lambda: leerArchivo())
botonCargar.grid(row=0, column=0,padx=1,pady=1)
botonAnalizar=Button(root,text="Analizar archivo",command = lambda: analizar())
botonAnalizar.grid(row=1, column=0,padx=1,pady=2)
botonSalir=Button(root,text="Salir",command = lambda: salir())
botonSalir.grid(row=3,column=0,padx=1,pady=2)


analizarTexto=Text(root,width=70, height=30)
analizarTexto.grid(row=3, column=8)
scrollTexto=Scrollbar(root, command=analizarTexto.yview)
scrollTexto.grid(row=3,column=9, sticky="nsew")


reportesMenu=Menubutton(root, text="Reportes", relief=RAISED)
reportesMenu.grid(row=2, column=0,padx=1,pady=2)
reportesMenu.menu =  Menu ( reportesMenu, tearoff = 0 )
reportesMenu["menu"] =  reportesMenu.menu
```

```
repTokens = IntVar()
repErrores = IntVar()

reportesMenu.menu.add_checkbutton ( label="Tokens",
                                    variable=repTokens, command=lambda: reportarTokens())
reportesMenu.menu.add_checkbutton ( label="Errores",
                                    variable=repErrores, command=lambda: reportarErrores())
```

Ahora las funciones que son llamadas cuándo un botón es presionado:

- LeerArchivo: Esta función es llamada cuándo se presiona el botón de "Cargar archivo" y permite cargar el archivo .form, leerlo y mostrarlo en el cuadro de texto

```
def leerArchivo():
    #en realidad elimino lo que va después de ellos
    delimitadores=""
    Tk().withdraw()
    entrada = filedialog.askopenfilename(initialdir="c:/LFP", title="Escoge un archivo", filetypes= (("form
    with open(entrada, encoding='utf-8') as archivo:
    #archivo = open(entrada, 'r')
        global contenido
        contenido = archivo.read().strip()
        analizarTexto.insert(1.0,contenido)
    contenido=contenido.lower()
    print(contenido)
```

- modifica_texto: Esta función nos permite modificar la información dentro del cuadro de texto procediente de nuestro archivo .form

```
def modifica_texto():
    f = open ('fichero_prueba.txt','a')
    f.write(analizarTexto.get(1.0,END))
    f.close()

root=Tk()
root.title("Creador de Formularios")
root.iconbitmap("corazon.ico")
root.config(bg="sky blue")

miFrame=Frame(root, width=500, height=400)
miFrame.config(bg="light pink")
```

- analizar: Esta función es llamada al presionar el botón de "Analizar" y a través de ella se llama a nuestra clase Analizador explicada posteriormente
y a funciones dentro de ella que nos permiten obtener los Tokens y Errores y analizar el contenido del documente
- reportarTokens: Esta función es llamada al acceder a reportes --> Tokens y a través de la clase Analizador de igual manera nos permite generar nuestro reporte de Tokens en HTML.
- reportarErrores: Esta función es llamada al acceder a reportes --> Errores y a través de la clase Analizador de igual manera nos permite generar nuestro reporte de Errores en HTML.

```
def analizar ():
    scanner = Analizador()
    scanner.AnalisisLexico(contenido)
    scanner.imprimirDatos()
    scanner.impTokens()
    scanner.impErrores()

def reportarTokens():
    fabricador=Analizador()
    fabricador.AnalisisLexico(contenido)
    fabricador.reporteTokens()

def reportarErrores():
    fabricador=Analizador()
    fabricador.AnalisisLexico(contenido)
    fabricador.reporteErrores()
def salir():
    sys.exit()
root.mainloop()
```

ANALIZADOR

Dentro del módulo Analisis se encuentra esta clase de tipo objeto, la cuál es la encargada de realizar todo el análisis de nuestro programa a través de las diferentes funciones dentro de ella

Empezamos por definir sus atributos:

```
class Analizador():
    def __init__(self):
        self.listaTokens = []
        self.listaErrores = []
        self.linea = 1
        self.columna = 0
        self.buffer = ''
```

- listaTokens = una lista donde añadiremos Tokens
- listaErrores = una lista donde añadiremos los Errores
- linea = acá guardaremos la línea en dónde nos encontramos
- columna =acá guardaremos la columna en dónde nos encontramos
- buffer = se guarda el caracter o caracteres que vamos leyendo

FUNCIONES

```
def imprimirDatos(self):
    print ('**************Lista Tokens************')
    for token in self.listaTokens:
        token.getTok()
    for error in self.listaErrores:
        error.getError()
#tengo que crear mi objeto Token first
def agregar_token(self,caracter,token,linea,columna):
    self.listaTokens.append(Token(caracter,token,linea,columna))
    self.buffer = ''

#tengo que crear mi objeto Error first
def agregar_error(self,caracter,descripcion,linea,columna):
    self.listaErrores.append(Error( caracter, descripcion, linea, columna))
    self.buffer=''
```

```python
def AnalisisLexico(self,cadena):
    text = cadena + '$'
    estado = 0
    option = True
    for caracter in text:
        option = True
        while option:
            if estado == 0:
                option=False
                if caracter == '~':
                    self.buffer+=caracter
                    self.columna+=1
                    #acá ya agrego a la lista de tokens y a los tokens, ya en agregar token se re
                    self.agregar_token(self.buffer, 'Virgulilla', self.linea, self.columna)
                elif caracter == '\n':
                    self.linea += 1
                    self.columna = 1
                elif caracter == '\r':
                    self.linea += 1
                    self.columna = 1
                elif caracter in ['\t',' ']:
                    self.columna += 1
                elif caracter == '>':
                    self.buffer+= caracter
                    self.columna+=1
                elif caracter == '>':
                    self.buffer+= caracter
                    self.columna+=1
                    self.agregar_token(self.buffer, 'Mayor que', self.linea, self.columna)
                elif caracter == '<':
                    self.buffer+= caracter
                    self.columna+=1
                    self.agregar_token(self.buffer, 'Menor que', self.linea, self.columna)
                elif caracter == ',':
                    self.buffer+= caracter
                    self.columna+=1
                    self.agregar_token(self.buffer, 'Coma', self.linea, self.columna)
                elif caracter == ':':
                    self.buffer+= caracter
                    self.columna+=1
                    self.agregar_token(self.buffer, 'Dos Puntos', self.linea, self.colum
                elif caracter == '[':
                    self.buffer+= caracter
                    self.columna+=1
                    estado = 3
                elif caracter =='$':
                    print("Archivo leído con éxito")
                elif caracter.isalpha():
                    self.buffer+= caracter
                    self.columna+=1
                    estado = 1
                elif caracter=='"':
                    self.buffer+= caracter
                    self.columna+=1
                    estado = 2
```

por motivos estéticos el resto de código dentro de esta función no será agregado en la imagen, puede encontrarse en el siguiente repositorio: https://github.com/AngelaPinelo/LFP_202002536.git

```python
def impTokens(self):
    print("TABLA TOKENS")
    x = PrettyTable()
    x.field_names = ["Lexema", "Token", "Fila", "Columna"]
    for i in self.listaTokens:
        x.add_row(i.enviarDataTok())
    print(x)

def impErrores(self):
    print("TABLA ERRORES")
    x = PrettyTable()
    x.field_names = ["lexema","Descripcion", "Fila", "Columna"]
    if len(self.listaErrores)==0:
        print('No hay errores')
    else:
        for i in self.listaErrores:
            x.add_row(i.enviarDataError())
        print(x)
```

```python
def reporteTokens(self):···


def reporteErrores(self):···
```

- imprimirDatos : Está función únicamente nos muestra en consola los Tokens y Errores solo para la verificación del funcionamiento correcto del programa
- agregar_token: utilizando mi objeto Token que será definido posteriormente, esta función permite agregar a mi lista de tokens todos los que deseemos agregar
- agregar_error: utilizando mi objeto Error que será definido posteriormente, esta función permite agregar a mi lista de errores todos los que deseemos agregar
- AnalisisLexico: Es en esta función dónde se lleva a cabo el análisis del programa, simulando un autómata con 4 estados distintos (0,1,2 y 3), es en cada uno de estos estados nos permite identificar y almacenar los distintos Tokens y Errores.
- impTokens: Esta función me permite imprimir en consola la lista de Tokens
- impErrores:Esta función me permite imprimir en consola la lista de Errores utilizando Prettytable
- reporteTokens: Crea el reporte en HTML de los Tokens obtenidos.
- reporteErrores: Crea el reporte en HTML de los }Errores obtenidos.

TE

Dentro del módulo TE se crean los objetos de Token y Error con sus respectivas funciones que nos permiten mandarles datos y obtenerlos (get y set)

```python
class Token:
    def __init__ (self,lexema,tipo,linea,columna):
        self.lexema = lexema
        self.linea = linea
        self.columna = columna
        self.tipo = tipo

    def enviarDataTok(self):
        return [self.lexema, self.tipo, self.linea, self.columna]

    def getTok(self):

        print('\n *****************')
        print('Tipo:', self.tipo)
        print('Lexena:', self.lexema)
        print('Linea:', self.linea)
        print('Columna:', self.columna)
```

```python
class Error :
    def __init__ (self,tipo,descripcion, linea,columna):
        self.descripcion = descripcion
        self.linea = linea
        self.columna = columna
        self.tipo = tipo

    def enviarDataError(self):
        return [self.descripcion, self.tipo, self.linea, self.columna]

    def getError(self):

        print('\n *****************')
        print('Tipo:', self.tipo)
        print('Descripcion:', self.descripcion)
        print('Linea:', self.linea)
        print('Columna:', self.columna)
```

ER Y AFD

Se mostrará cada expresión regular con su AFD de cada uno de los tokens válidos para el lenguaje, primero se expresará el lexema, luego el tipo de Token, después la expresión regular y por último el AFD. Para una mejor comprensión de lo expresado se resumirá en mayor medida las ER de cada token.

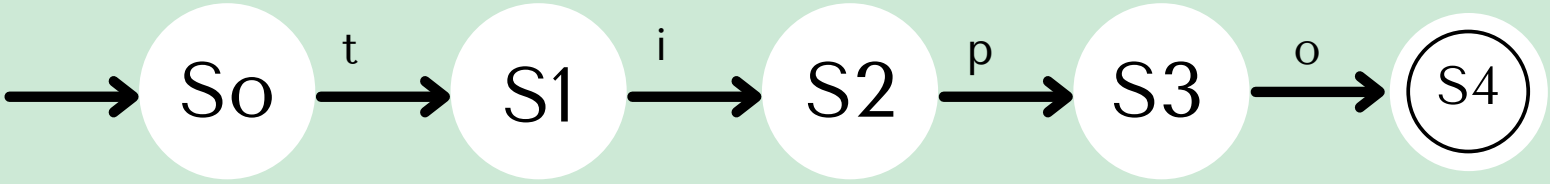
Sabemos que dentro de nuestro "Lenguaje" poseemos distintas palabras reservadas:

- formulario
- tipo
- valor
- fondo
- valores
- evento

Como podemos observar las palabras reservadas es en sí un conjunto de letras (se decidió hacer el lenguaje de tipo case insensitive, lo que significa que no distinguimos entre mayúsculas y minúsculas por lo cual no tomaremos en cuenta eso), debido a esto utilizamos la siguiente ER:
L = conjunto de letras de la a-z/A-Z o bien ([a-zA-Z])
*= El asterisco significa que puede venir 0 o más veces

LL*

Esta ER realmente funciona para cualquier palabra reservada dentro de nuestro lenguaje y se para poder representarla con un AFD pondremos de ejemplo la palabrra reservada "tipo" de la siguiente forma:

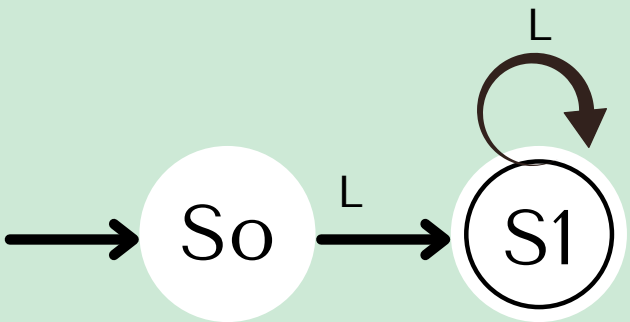


Es de esta forma que se representarían todas las palabras reservadas en un AFD, tomando en cuenta su último estado S4 como el de aceptación (es por esto que contiene doble círculo), a su vez estas palabras son aceptadas por el autómata simplificado en el código en el estado 1.

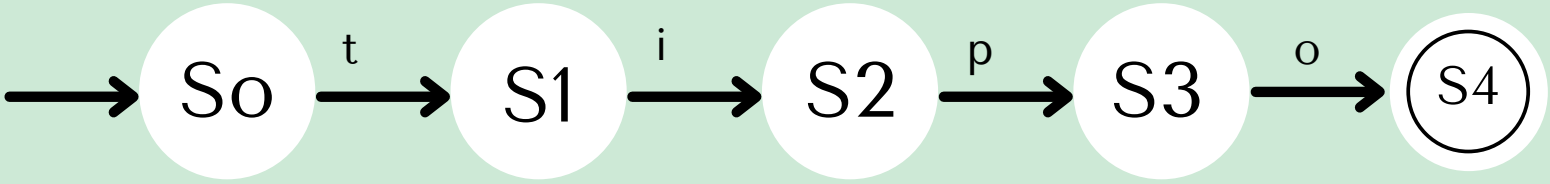
Cómo se puede mostrar en este ejemplo todas las letras de la palabra reservada "tipo" pertenecen al alfabeto y es por eso que la ER es:

$$LL^*$$

Y su autómata puede representarse de la siguiente forma:



Esta ER realmente funciona para cualquier palabra reservada dentro de nuestro lenguaje y se para poder representarla con un AFD pondremos de ejemplo la palabrra reservada "tipo" de la siguiente forma:

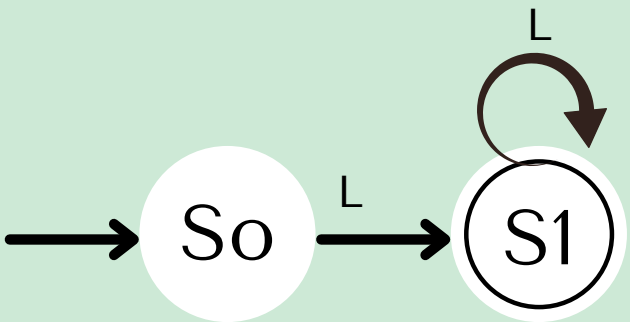


Es de esta forma que se representarían todas las palabras reservadas en un AFD, tomando en cuenta su último estado S4 como el de aceptación (es por esto que contiene doble círculo), a su vez estas palabras son aceptadas por el autómata simplificado en el código en el estado 1.

Cómo se puede mostrar en este ejemplo todas las letras de la palabra reservada "tipo" pertenecen al alfabeto y es por eso que la ER es:

$$LL^*$$

Y su autómata puede representarse de la siguiente forma:



Para los signos dentro de nuestro lenguaje como lo son:

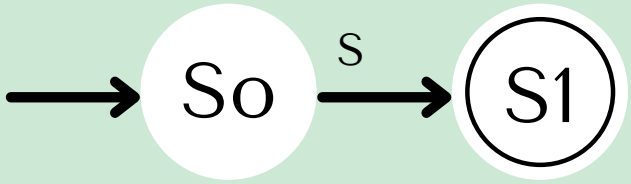
- Virgulilla '~'
- Dos puntos ':'
- Coma ','
- Espacio ' '
- Menor que '<'
- Mayor que '>'

Poseen un estado propio que es catalogado en general como el estado 0, su expresión regular es la siguiente:

S=Signo dentro del lenguaje

S

Y su autómata puede representarse de la siguiente forma:



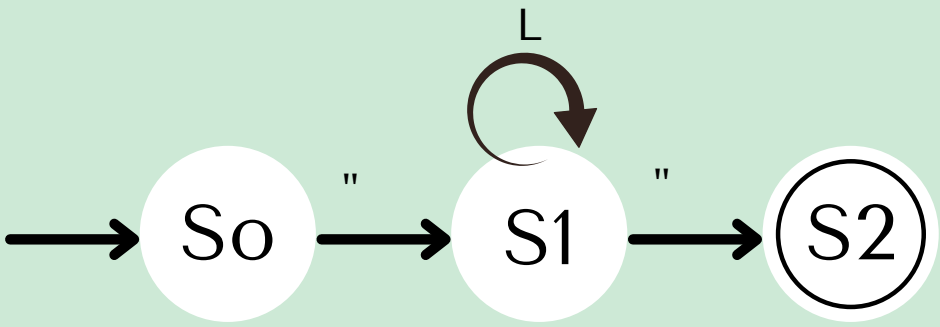
Ahora bien, las comillas no entran dentro de los signos ya que estas vienen seguidas de lo que catalogamos como "instrucción" dentro de nuestro proyecto, lo que quiere decir que nuestro analizador al reconocer una comilla la manda a nuestro estado 2, almacenando la cadena que viene dentro de ella ya que posee una instrucción

Su ER está dada por:

L= [a-zA-Z]

"L*"

Y su autómata puede representarse de la siguiente forma:



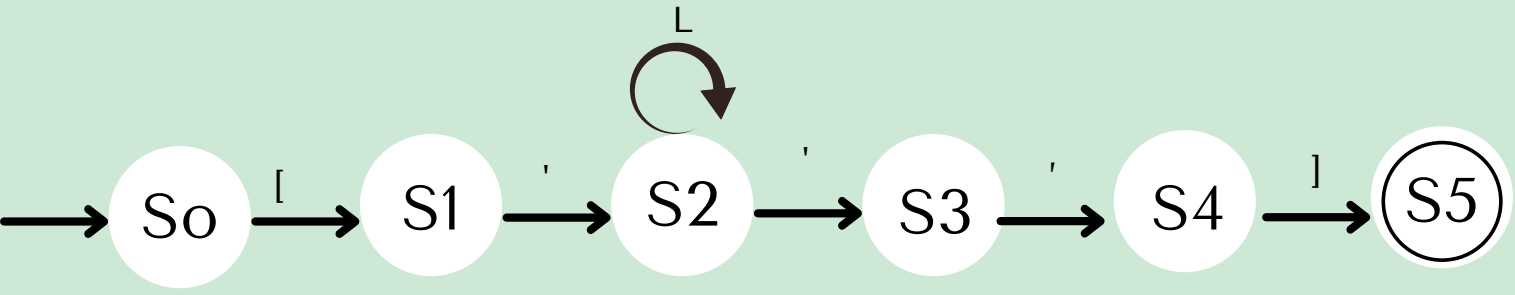
Tampoco los corchetes ([]) están incluidos dentro de nuestros signos directamente ya que dentro de estos vienen incluídos los grupos que debemos de almacenar como opciones para la creación de ciertos parámetros en nuestro formulario y debemos de almacenar todo lo que viene dentro de ellos. Esto está representado dentro de el estado 3

Su ER está dada por:

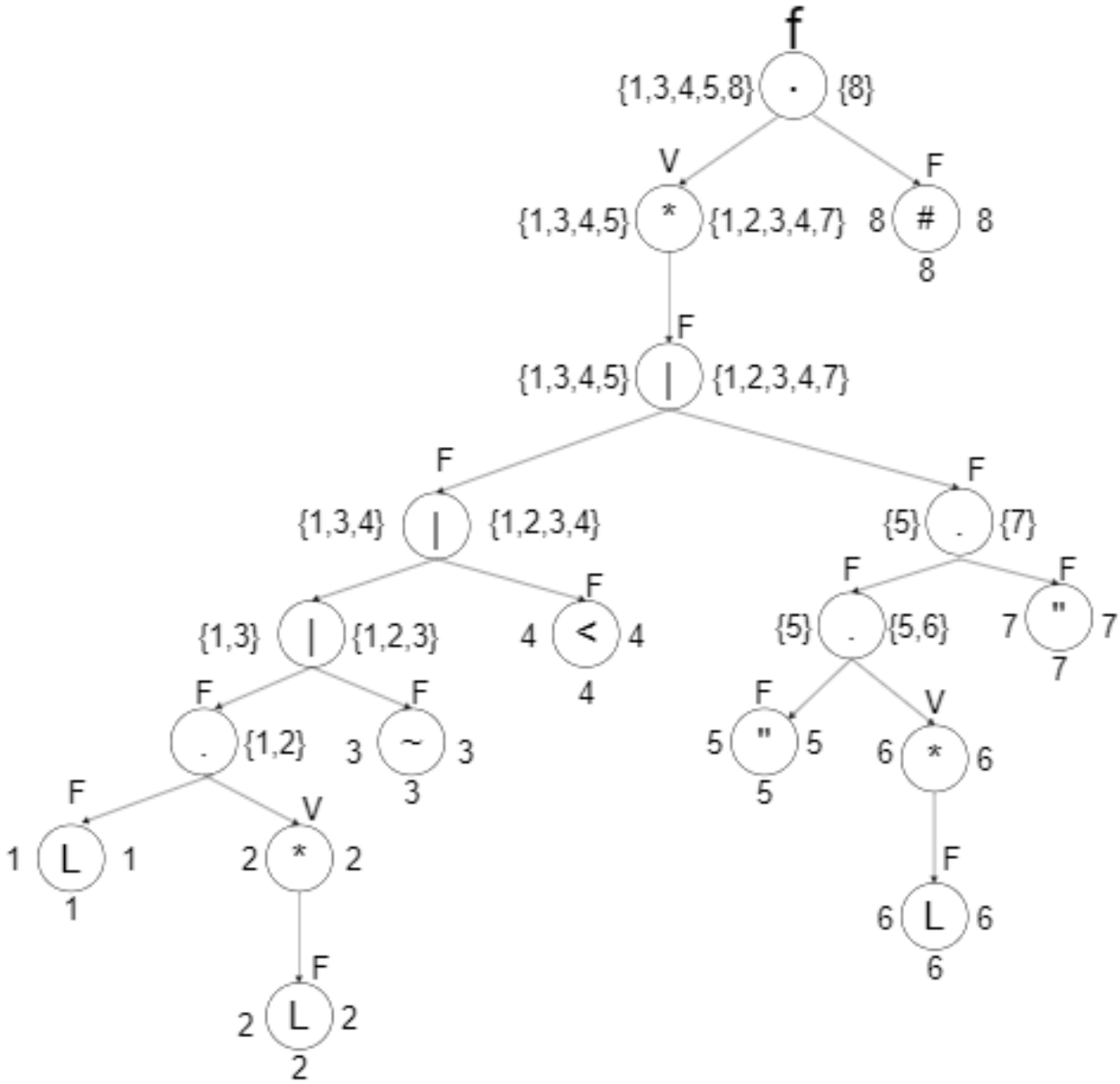
L= [a-zA-Z]

['L*',]

Y su autómata puede representarse de la siguiente forma:



ÁRBOL


f
{1,3,4,5,8} · {8}
V
{1,3,4,5} * {1,2,3,4,7}
F
8 # 8
8
F
{1,3,4,5} | {1,2,3,4,7}
F
{1,3,4} | {1,2,3,4}
F
{5} . {7}
F
{1,3} | {1,2,3}
F
4 < 4
4
F
{5} . {5,6}
F
7 " 7
7
F
. {1,2}
F
3 ~ 3
3
F
5 " 5
5
V
6 * 6
F
1 L 1
1
V
2 * 2
F
6 L 6
6
F
2 L 2
2