



INTEGRANTES:

- Ariel Humberto Valle Escoto
- Eduardo Quetzal Delgado Pimentel
- Ricardo David López Arellano

MATERIA: Traductores de lenguajes II

2023b

INTRODUCCIÓN

Un analizador sintáctico descendente predictivo es un tipo de analizador sintáctico utilizado en el campo de la informática y la teoría de la compilación. Su objetivo principal es analizar la estructura gramatical de un programa fuente para determinar si cumple con la gramática del lenguaje de programación en cuestión. En otras palabras, verifica si el código fuente está escrito correctamente en términos de la sintaxis del lenguaje de programación.

Los analizadores sintácticos descendentes predictivos utilizan un conjunto de funciones de predicción para determinar qué regla de producción aplicar en función del símbolo actual de entrada y, a veces, de algunos símbolos de contexto. Estas funciones de predicción se construyen a partir de la gramática del lenguaje y permiten al analizador predecir qué producción debe aplicarse sin tener que retroceder en el análisis.

Los analizadores sintácticos descendentes predictivos son más fáciles de entender e implementar en comparación con otros tipos de analizadores sintácticos, como los analizadores sintácticos ascendentes o los analizadores sintácticos LR. Sin embargo, tienen limitaciones y no pueden manejar todas las gramáticas, especialmente aquellas que son ambiguas o recursivas izquierdas.

CAPTURAS

PRACTICA 4

INGRESA LA EXPRESIÓN:
A+9-1*8*N

ID	VALOR	OTRO VALOR
1	A	NONE
2	N	NONE

ID	VALOR	OTRO VALOR
1	ID	A
2	NUM	9
3	+	1
4	NUM	1
5	NUM	8
6	*	4
7	ID	N
8	*	6
9	-	3

HACER GRAFO

PRACTICA 4

INGRESA LA EXPRESIÓN:

1+9-22*A

TABLA DE SÍMBOLOS:			TABLA DE GRAFO:			
1	A	NONE	1	NUM	1	
			2	NUM	9	
			3	+	1	2
			4	NUM	22	
			5	ID	A	
			6	*	4	5
			7	-	3	6

HACER GRAFO

PRACTICA 4

INGRESA LA EXPRESIÓN:

2+2*1-A/L

TABLA DE SÍMBOLOS:			TABLA DE GRAFO:			
1	A	NONE	1	NUM	2	
2	L	NONE	2	NUM	2	
			3	NUM	1	
			4	*	2	3
			5	+	1	4
			6	ID	A	
			7	ID	L	
			8	/	6	7
			9	-	5	8

HACER GRAFO

PRACTICA 4

INGRESA LA EXPRESIÓN:

2+5/3

TABLA DE SÍMBOLOS:			TABLA DE GRAFO:			
			1	NUM	2	
			2	NUM	5	
			3	NUM	3	
			4	/	2	3
			5	+	1	4

HACER GRAFO

CÓDIGO

```
import re
import tkinter as tk
from tkinter import font
from PIL import Image, ImageTk
import networkx as nx
import matplotlib.pyplot as plt

class Nodo:
    def __init__(self, operador, izquierda, derecha):
        self.operador = operador
        self.izquierda = izquierda
        self.derecha = derecha

class Hoja:
    def __init__(self, tipo, valor):
        self.tipo = tipo
        self.valor = valor

class TablaSimbolos:
    def __init__(self):
        self.tabla = {}

    def agregar(self, identificador, valor):
        self.tabla[identificador] = valor

    def obtener(self, identificador):
        return self.tabla.get(identificador, None)

def analizador_sintactico(expresion):
    tokens = re.findall(r'\d+|[-+*/()]\w+', expresion)
    idx = 0

    def match(expected_token):
        nonlocal idx
        if idx < len(tokens) and tokens[idx] == expected_token:
            idx += 1
        else:
            raise Exception(f"Error de sintaxis: Se esperaba '{expected_token}'")

    def factor():
        if tokens[idx] == '(':
            match('(')
```

```

        sub_arbol = expresion()
        match(')')
        return sub_arbol
    elif tokens[idx].isdigit():
        num = Hoja('num', tokens[idx])
        match(tokens[idx])
        return num
    elif tokens[idx].isalpha():
        id = Hoja('id', tokens[idx])
        match(tokens[idx])
        return id
    else:
        raise Exception("Error")

def termino():
    left = factor()
    while idx < len(tokens) and tokens[idx] in ['*', '/']:
        operador = tokens[idx]
        match(operador)
        right = factor()
        left = Nodo(operador, left, right)
    return left

def expresion():
    left = termino()
    while idx < len(tokens) and tokens[idx] in ['+', '-']:
        operador = tokens[idx]
        match(operador)
        right = termino()
        left = Nodo(operador, left, right)
    return left

tabla_simbolos = TablaSimbolos()
raiz = expresion()

def construir_tabla_simbolos(nodo):
    if isinstance(nodo, Nodo):
        construir_tabla_simbolos(nodo.izquierda)
        construir_tabla_simbolos(nodo.derecha)
    elif isinstance(nodo, Hoja) and nodo.tipo == 'id':
        tabla_simbolos.agregar(nodo.valor, None)

construir_tabla_simbolos(raiz)

return raiz, tabla_simbolos

```

```

tSimbolos = "" # Declarar tSimbolos como una variable global

def mostrar_tabla_simbolos(tabla_simbolos):
    global tSimbolos
    i = 0
    tSimbolos = "Tabla de símbolos:\n"
    for identificador in tabla_simbolos.tabla:
        i += 1
        valor = tabla_simbolos.obtener(identificador)
        tSimbolos += f"{i}\t{identificador}\t{valor}\n"
    tablaSimbolos.config(text=tSimbolos) # Actualiza la etiqueta de la tabla de
    símbolos

def mostrar_grafo(raiz):
    global tGrafo
    tGrafo = "Tabla de grafo:\n"
    grafo = []

    def dfs(nodo):
        if isinstance(nodo, Nodo):
            left_id = dfs(nodo.izquierda)
            right_id = dfs(nodo.derecha)
            grafo.append((nodo.operador, left_id, right_id))
            return len(grafo)
        elif isinstance(nodo, Hoja):
            grafo.append((nodo.tipo, nodo.valor, None))
            return len(grafo)

    dfs(raiz)
    i = 0
    for item in grafo:
        i += 1
        if item[2] is not None:
            tGrafo += f"{i}\t{item[0]}\t{item[1]}\t{item[2]}\n"
        else:
            tGrafo += f"{i}\t{item[0]}\t{item[1]}\n"

def main():
    global tGrafo, tSimbolos
    expresion = entrada.get()
    raiz, tabla_simbolos = analizador_sintactico(expresion)
    mostrar_tabla_simbolos(tabla_simbolos)
    mostrar_grafo(raiz)

```

```
tablaSimbolos.config(text=tSimbolos, anchor='w', justify='center')
tablaGrafo.config(text=tGrafo, anchor='w', justify='left')

ventana = tk.Tk()
ventana.title("PRACTICA 4")

# Cambia el tamaño de la ventana
ventana.geometry("600x500") # Cambia el ancho y alto

# Cambia el color de fondo de la ventana
ventana.configure(bg="gray") # Cambia el color de fondo

# Crear una fuente personalizada después de crear la ventana
fuente_personalizada = font.Font(family="Algerian", size=12) # Cambia "Arial" al
tipo de fuente que desees y el tamaño

# Aplica la fuente personalizada a la etiqueta
etiqueta = tk.Label(ventana, text="Ingresa la expresión:",
font=fuente_personalizada)
etiqueta.pack()

entrada = tk.Entry(ventana, font=fuente_personalizada)
entrada.pack()

# Crear un marco para contener las tablas
marco_tablas = tk.Frame(ventana)
marco_tablas.pack()

# Tabla de símbolos
tablaSimbolos = tk.Label(marco_tablas, text="", anchor='w', justify='left',
font=fuente_personalizada)
tablaSimbolos.pack(side="left", padx=10, pady=10)

# Tabla de grafo
tablaGrafo = tk.Label(marco_tablas, text="", anchor="w", justify='left',
font=fuente_personalizada)
tablaGrafo.pack(side="right", padx=10, pady=10)

# Cambia el tamaño y la forma de los botones a "ridge"
calcular_button = tk.Button(ventana, text="Hacer Grafo", command=main,
font=fuente_personalizada, width=20, height=2, bg="black", fg="white")
calcular_button.pack()

ventana.mainloop()
```

CONCLUSIONES

Lo que podemos concluir como equipo al terminar la práctica 4 es que comprendimos de mejor manera el tema de los analizadores sintácticos descendentes ya que con este conocimiento podremos realizar nuestro proyecto final, ya después de varias practicas podemos ver como poco a poco logramos entender o juntar las piezas de lo que realizaremos en un futuro ya que es fundamental entender bien desde el principio para así poder seguir desarrollando las piezas de este compilador ya que si lo hiciéramos sin ninguna estructura, estamos seguros que no funcionaria, pero con estos avances logramos observar una mejoría en las habilidades que tenemos para poder desarrollar estos programas.

En conclusión, un analizador sintáctico descendente predictivo es una herramienta esencial en la teoría de compiladores y la informática. Su función es verificar si un programa fuente sigue la sintaxis del lenguaje de programación, permitiendo detectar errores gramaticales en el código. Estos analizadores utilizan funciones de predicción basadas en la gramática del lenguaje para determinar las reglas de producción, lo que los hace más fáciles de entender y implementar en comparación con otros tipos de analizadores. A pesar de su simplicidad, los analizadores sintácticos descendentes predictivos tienen limitaciones y no pueden manejar todas las gramáticas, especialmente las ambiguas o recursivas izquierdas, lo que requiere técnicas más avanzadas en esos casos.