TRABAJO PRÁCTICO Nº 3

ANALIZADOR SINTACTICO PREDICTIVO NO RECURSIVO

Contenido Conceptual

Construcción de un árbol de análisis sintáctico. Derivaciones. Gramáticas LL(1). Análisis sintáctico predictivo no recursivo.

Objetivos

- Implementar a través de Python analizador sintáctico predictivo no recursivo.

EJERCICIOS

Ejercicio 1:

Implemente a través de Python analizador sintáctico predictivo no recursivo para la gramática: E→ E + E | E – E | E % E | (E) | id, y que funcione como calculadora, es decir, si la entrada es: 10+5-2, que muestre el resultado 13. % es la operación módulo o resto, por ejemplo 5%2 = 1, el resto de la división entre 5 y 2 es 1.

Ejemplo:

Para la gramática de expresiones:

E→ E + E | E ∗ E | (E) | id

Se elimina la recursividad inmediata por la izquierda de la gramática de expresiones y se obtiene la siguiente gramática:

E → T E'

E' → + T E'| ε

T → F T'

T' → * F T'| ε

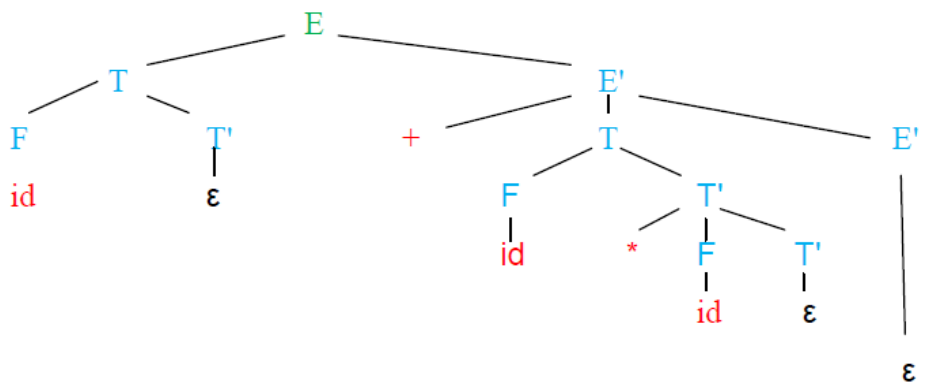
F → (E) | id

La tabla de análisis sintáctico es la siguiente:

No terminal	Símbolo de entrada					
	id	+	*	(	)	$
E	E → T E'			E → T E'		
E'		E' → + T E'			E' → ε	E' → ε
T	T → F T'			T → F T'		
T'		T' → ε	T' → * F T'		T' → ε	T' → ε
F	F → id			F → (E)		

Para la entrada id+id*id, obtenemos el siguiente árbol sintáctico:

Pila	Entrada	Salida
$E	id + id * id$	
$E' T	id + id * id$	E→ T E'
$E' T' F	id + id * id$	T→ F T'
$E' T' id	id + id * id$	F→ id
$E' T'	+ id * id$	
$E'	+ id * id$	T'→ ε
$E' T +	+ id * id$	E'→ + TE'
$E' T	id * id$	
$E' T' F	id * id$	F→ FT'
$E' T' id	id * id$	F→ id
$E' T'	* id$	
$E' T' F *	*id$	T'→ *FT'
$E' T' F	id$	
$E' T' id	id$	F→ id
$E' T'	$	
$E'	$	T'→ ε
$	$	E'→ ε



Implementación con Python:

```
#Analizador sintactico predictivo para operaciones aritméticas + * y ()
#Ejemplo de entrada id*id+id
#Gramatica
#E → T E'
#E' → + T E'| ε
#T → F T'
#T' → * F T'| ε
#F → ( E ) | i
def obtener_col(simbolo_entrada): #obtiene columna tabla de analisis
    if(simbolo_entrada == 'i'):
        return 0
    else:
        if(simbolo_entrada == '+'):
            return 1
        else:
            if(simbolo_entrada == '*'):
                return 2
            else:
                if(simbolo_entrada == '('):
                    return 3
                else:
                    if(simbolo_entrada == ')'):
                        return 4
                    else:
                        if(simbolo_entrada == '$'):
```

```
                                        return 5
                                    else:
                                        return 6


def obtener_fila(no_terminal): #obtiene fila tabla de analisis
    if(no_terminal == 'E'):
        return 0
    else:
        if(no_terminal == 'E\''):
            return 1
        else:
            if(no_terminal == 'T'):
                return 2
            else:
                if(no_terminal == 'T\''):
                    return 3
                else:
                    if(no_terminal == 'F'):
                        return 4
                    else:
                        return 5



class Pila:
    def __init__(self):
        self.items = []
```

```
    def estaVacia(self): #verificar si la pila está vacía
        return self.items == []

    def insertar(self, item): #inserta elemento en la pila (cima)
        self.items.append(item)

    def extraer(self): #extrae elemento de la pila (cima)
        return self.items.pop()

    def inspeccionar(self): #devuelve el elemento de la cima de la pila
        return self.items[len(self.items)-1]

    def tamano(self): #devuelve el tamaño de la pila
        return len(self.items)

    def contenido(self): #devuelve el tamaño de la pila
        return (self.items)

tabla=[["E->TE'","","","E->TE'","","",""],["","E'->+TE'","","","E'->e","E'->e"],["T->FT'","","","T->FT'","","",""],["","T'->e","T'->*FT'","","T'->e","T'->e"],["F->i","","","F->(E)","","",""]]
p=Pila()
p.insertar('$')
p.insertar('E')
simbolo_entrada = 'i'
entrada = ['i','+','i','*','i','$']
```

```
entrada_2 = ['i','+','i','*','i','$']
salida = ''
print ('PILA \t\t\t\t ENTRADA \t\t\t\t SALIDA')
print(str(p.contenido()) + '\t\t\t' + str(entrada_2) + '\t\t\t' + str(salida))
for simbolo_entrada in entrada:
        cima_pila = p.inspeccionar()
        while(cima_pila != simbolo_entrada):
                col = obtener_col(simbolo_entrada)
                fil = obtener_fila(cima_pila)
                salida = tabla[fil][col]
                if(salida != ''):
                        p.extraer()
                        posicion = salida.find('>')
                        produccion = salida[posicion+1:len(salida)]
                        produccion_pila = []
                        for  simbolo in produccion:
                                if(simbolo != '\''):
                                        posicion_2 = produccion.find(simbolo)
                                        if(produccion[posicion_2+1:posicion_2+2] == '\''):
                                                produccion_pila.append(simbolo + '\'')

                                        else:
                                                produccion_pila.append(simbolo)

                        for simbolo in reversed(produccion_pila):
                                if(simbolo != 'e'):
                                        p.insertar (simbolo)
```

```
            print(str(p.contenido()) + '\t\t\t' + str(entrada_2) + '\t\t\t' + str(salida))
            cima_pila = p.inspeccionar()
    if(simbolo_entrada == '$' and p.inspeccionar() == '$'):
            print("Arbol sintáctico construido!")
    else:
            p.extraer()
            entrada_2.pop(0)
            print(str(p.contenido()) + '\t\t\t' + str(entrada_2) + '\t\t\t')
```