Universidad Católica San Pablo

Compiladores

Práctica 6 – Analizadores sintáctico descendente recursivo predictivo (3ra parte)

Objetivo:

- Construir un analizador sintáctico descendente predictivo.
- Escribir una tabla de análisis sintáctico
- Escribir el algoritmo de reconocimiento de una cadena utilizando el analizador.

Requisito:

- Práctica 5 (sobre getPrimero y getSiguiente)

Indicaciones:

1. **Crear la tabla de análisis sintáctico**

   Hacemos una evaluación NoTerminal vs Terminales y buscamos en la gramática, para cada NoTerminal, qué producción originó el elemento no terminal.

```
def crearTabla(self, … ):
    tas =  # AQUI UTILIZAR SU ESTRUCTURA CREADA

    for nodoNt in NoTerminal:
        for nodoT in getPrimero(nodoNt):
            if nodoT   !=  lambda
                 tas[ nodoNt ][ nodoT ] =
                            buscarProduccion( nodoNt, nodoT
)
            else:
                for nodoT2 in getSiguiente(nodoNt):
                    tas[nodoNt][nodoT2]  = lambda

            # tas[?][?] = ?   depende de su implementación
```

   La función **buscarProducción** ubica la producción desde la cual se originó el elemento nodoT.

   Verificar el resultado con el obtenido de la creación estática (práctica anterior)

2. Implementar algoritmo de validación de cadena

- Para validar una entrada, esta debe ser tokenizada y colocada dentro de una cola.
- Adicionar el símbolo de dólar a dicha cola.
- Tener una pila para ver los nodos que se van expandiendo en la validación.
- La idea es ir haciendo match entre los elementos que apuntan la pila y la cola en conformidad con la tabla de análisis sintáctico.

A seguir, implementar el siguiente algoritmo de validación:

```
analiza( cadena ) {
    entrada <- cola(cadena)
    pila <- iniciar una pila vacía

    pila.push( DOLAR )
    pila.push( gramatica->estadoInicial )
    entrada.push( DOLAR )

    mientras( !vacia(entrada) Y !vacia(pila) ) {
        si( entrada.top() = pila.top() ) {
            entrada.pop();
            pila.pop();
        } sino {
            tmp <- pila.pop();
            para  x  en tas[tmp][entrada.top()].reverse()
                si x != "lambda"
                    pila.push(x)
        }
    }

    retorna vacia(entrada) Y vacia(pila)
}
```

3. Verifique las entradas para:

num + num + num + num

(num + num) + (num + num)

num * (num / num)

(num *) num

Universidad Católica
San Pablo