UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

Asignatura: Compiladores (0434)

Proyecto: Analizador Sintáctico Ascendente

Profesor: Ing. Manuel Castañeda Castañeda

Integrantes del equipo:

- *Bautista Pérez Brian Jassiel*
- *González Ramírez David*
- *Guillén Castro Jorge Luis*
- *Meneses Gómez Iassiel Navih*

Grupo: 05
Semestre: 2022-2
Fecha de entrega: 25 de mayo de 2022

Índice

1. Introducción

El compilador es una herramienta que permite el traducir un lenguaje de programación a otro para su futura interpretación, el cuál maneja un análisis del código origen, para efectos de este proyecto se hizo enfásis en el análizador sintáctico.
El analizador sintáctico, también conocido como parser, representa la segunda fase de compilación, se encarga de realizar un análisis mediante las gramáticas para verificar si la sintaxis de la instrucción es adecuada para el programa, una vez realizado esto se genera un árbol sintáctico, el cual puede ser ascendente o descendente.
Para este proyecto, se implementó un árbol sintáctico ascendente, la construcción se realiza desde las hojas más externas del árbol y van poco a poco subiendo a sus raíces, esto se puede entender como una entrada, la cual va generando pequeñas derivaciones hasta llegar al axioma inicial.
Para la correcta realización del proyecto se va a implementar un analizador sintáctico ascendente de acuerdo a una gramática dada, donde el analizador léxico tendra que reconocer los números reales, identificadores, operadores y paréntesis, por lo cual se podrá indicar si la cadena es válida o no.

2. Desarrollo

Se debe de elaborar un analizador sintáctico ascendente que revise las sentencias de acuerdo a la siguiente gramática:
$E \to T$
$E \to E + T$
$E \to E - T$
$T \to i$
$E \to (E)$
i - Será cualquier número real o identificador.
Se utilizó el lenguaje Java para la implementación del código de nuestro analizador sintáctico.
Analizador sintáctico ascendente
A lo largo del curso se vio que un analizador léxico es aquel que conforma la primera fase de la compilación y está destinado a leer caracteres del archivo de entrada, donde se encuentra la cadena a analizar, reconocer subcadenas que correspondan a símbolos del lenguaje y retomar los tokens correspondientes y sus atributos. Además procesa los lexemas que le va a suministrar el analizador sintáctico y verificará que estén bien ordenados, si no lo están se generarán errores.
A continuación, el código:

```
import java.io.*;


public class bottomUpParser{
  static int contador=0;
  static
  int flag=0;

  public static void main(String[] args) throws IOException, Exception{

    analizador();
  }

  //Definicion de la gramatica
  public static void regla1(String cadena){
    if(cadena.length()==2){
      System.out.println("i -> T");
  //regla 2
      String cambio = cadena.replaceFirst("i","T");
      System.out.println("Cadena: "+cambio);
      regla5(cambio);
    }
    else if(cadena.length()>=3){
      System.out.println("i->T");
```

```
      String cambio = cadena.replaceFirst("i","T");
      System.out.println("Cadena: "+cambio);
      if(cambio.equals(cadena)){
        System.out.println("\n******La cadena si fue aceptada******\n");
      }
      else{
        switch(cambio){
          case "E+T;" -> regla4(cambio);
          case "E-T;" -> regla3(cambio);
          case "(E+T);" -> regla4(cambio);
          case "(E-T); " -> regla3(cambio);
          default -> regla5(cambio);
        }
      }
    }
  }

  //Regla 1
  public static void regla2(String cadena){
    System.out.println("E -> T");
    String cambio = cadena.replace("(E)", "T");
    System.out.println("Cadena: "+cambio);
    regla5(cambio);
  }

  //Regla 3
  public static void regla3(String cadena){
    System.out.println("E-T -> E");
    String cambio = cadena.replace("E-T","E");
    System.out.println("Cadena: "+cambio);
    if(cambio.length()!=2){
      regla2(cambio);
    }
  }

  //Regla4
  public static void regla4(String cadena){
    System.out.println("E+T -> E");
    String cambio = cadena.replace("E+T", "E");
    System.out.println("cadena: "+cambio);
    if(cambio.length()!=2){
      regla2(cambio);
    }
  }

  //Regla5
  public static void regla5(String cadena){
    if(cadena.length() == 2){
      System.out.println("T->E");
      String cambio = cadena.replaceFirst("T", "E");
      System.out.println("Cadena: "+cambio);
    }
    else if(cadena.length() >= 3){
      contador++;
      System.out.println("T -> E");
      String cambio = cadena.replaceFirst("T","E");
      System.out.println("Cadena: "+cambio);
      if(cambio.contains("(") && flag != 0){
        regla2(cambio);
      }
      else{
        regla1(cambio);
      }
    }
  }

```

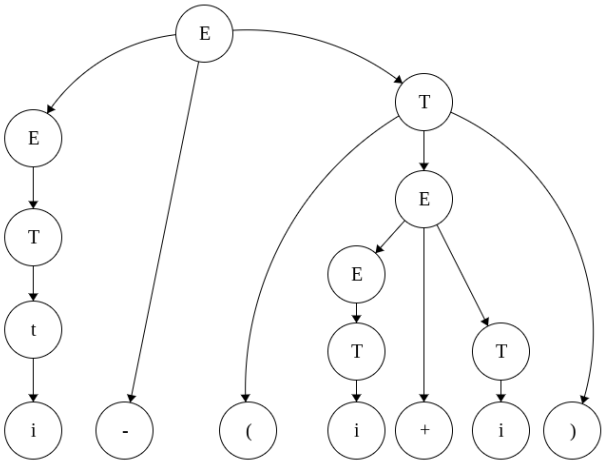
```
91    public static void analizador() throws NoSuchFieldException, IOException{
92      String cadena;
93      int linea = 0, parentesis;
94      FileReader prueba = new FileReader("./prueba.txt");
95      BufferedReader b = new BufferedReader(prueba);
96      while((cadena = b.readLine()) !=null){
97        linea++;
98        //esta linea de codigo delimita la expresion regular del ejemplos solicitado
99        if(cadena.matches("i[+|-]?[(]*i[+|-]i[)]+[;]")){ //i-(i+i)
100       //aqui mostrara la cadena solicitada
101         System.out.println("\nEjemplo: "+cadena);
102         if(cadena.contains("()")){ //buscara la presencia de parentesis
103           parentesis = CP(cadena);
104           if(parentesis != 0){
105               System.out.print("Error en la linea: "+linea+". La cadena es: "+cadena+"\n");
106           }
107           else{
108               flag = 0;
109               regla1(cadena);
110           }
111         }
112         else{
113           regla1(cadena);
114         }
115       }
116       else{
117         System.out.println("Error en la lina: "+linea+". La cadena es: "+cadena+"\n");
118       }
119     }
120     b.close();
121   }
122
123   //Estado para los parentesis. Verifica que los parentesis
124   //esten balanceados
125   public static int CP(String cadena){
126     int i = 0, f = 0;
127     while(cadena.length()-1 != f){
128       switch (cadena.charAt(f)){
129         case '(' -> {
130           i++;
131           f++;
132         }
133         case ')' -> {
134           i--;
135           f++;
136         }
137         default -> f++;
138       }
139     }
140     return i;
141   }
142 }
```

3. Análisis de resultados

A continuación el árbol sintáctico hecho de manera manual junto con su análisis:
Gramática.
1)$E \to T$
2)$E \to E + T$
3)$E \to E - T$
4)$T \to i$
5)$E \to (E)$

Cadena de Entrada. $i-(i+3{,}4)$
$4)T \to i-(i+T)$
$4)T \to i-(T+T)$
$1)E \to i-(E+T)$
$2)E \to i-(E)$
$5)T \to i-i+T$
$4)T \to T-T$
$1)E \to E-T$
$3)E \to E$

Cadena transformada con la gramática para realizar el analizador sintáctico ascendente: $i-(i+i)$
$3)E \to E-T$
$1)E \to T-T$
$4)T \to i-T$
$5)T \to i-(E)$
$2)E \to i-(E+T)$
$1)E \to i-(T+T)$
$4)T \to i-(i+T)$
$4)T \to i-(i+i)$



Árbol Sintáctico Ascendente.

4. Conclusiones

Aquí van las Conclusiones