



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

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	2
2. Desarrollo	2
3. Análisis de resultados	5
4. Conclusiones	6

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 énfasis en el analizador 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 tendrá 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:

$E \rightarrow T$

$E \rightarrow E + T$

$E \rightarrow E - T$

$T \rightarrow i$

$E \rightarrow (E)$

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:

```
1
2
3
4
5 import java.io.*;
6
7
8 public class bottomUpParser{
9     static int contador=0;
10    static
11    int flag=0;
12
13    public static void main(String[] args) throws IOException, Exception{
14
15        analizador();
16    }
17
18    //Definicion de la gramatica
19    public static void regla1(String cadena){
20        if(cadena.length()==2){
21            System.out.println("T -> i");
22            String cambio = cadena.replaceFirst("i","T");
23            System.out.println("Cadena: "+cambio);
24            regla5(cambio);
```

```

25     }
26     else if (cadena.length() >= 3) {
27         System.out.println("T-> i");
28         String cambio = cadena.replace("i+i", "i+T");
29         System.out.println("Cadena: " + cambio);
30         if (cambio.equals(cadena)) {
31             System.out.println("\n*****La cadena si fue aceptada*****\n");
32         }
33         else {
34             switch (cambio) {
35                 case "i-i+T;" -> regla7(cambio);
36                 case "i-i-T;" -> regla6(cambio);
37                 case "E+T;" -> regla4(cambio);
38                 case "E-T;" -> regla3(cambio);
39                 case "i-(i+T);" -> regla7(cambio);
40                 case "i-(i-T);" -> regla6(cambio);
41                 case "(E+T);" -> regla4(cambio);
42                 case "(E-T);" -> regla3(cambio);
43                 default -> regla5(cambio);
44             }
45         }
46     }
47 }
48
49 //Regla 2
50 public static void regla2(String cadena) {
51     System.out.println("T -> E");
52     String cambio = cadena.replace("(E)", "T");
53     System.out.println("Cadena: " + cambio);
54     regla6(cambio);
55 }
56
57 //Regla 3
58 public static void regla3(String cadena) {
59     System.out.println("E -> E-T");
60     String cambio = cadena.replace("E-T", "E");
61     System.out.println("Cadena: " + cambio);
62     if (cambio.length() != 2) {
63         regla2(cambio);
64     } else if (cambio.length() == 2) {
65         regla11(cambio);
66     }
67 }
68
69 //Regla4
70 public static void regla4(String cadena) {
71     System.out.println("E -> E+T");
72     String cambio = cadena.replace("E+T", "E");
73     System.out.println("Cadena: " + cambio);
74     if (cambio.length() != 2) {
75         regla2(cambio);
76     }
77 }
78
79 //Regla5
80 public static void regla5(String cadena) {
81     if (cadena.length() == 2) {
82         System.out.println("E-> T");
83         String cambio = cadena.replaceFirst("T", "E");
84         System.out.println("Cadena: " + cambio);
85     }
86     else if (cadena.length() >= 3) {
87         contador++;
88         System.out.println("E -> T");
89         String cambio = cadena.replaceFirst("T", "E");
90         System.out.println("Cadena: " + cambio);

```

```

91         if(cambio.contains("(") && flag != 0){
92             regla2(cambio);
93         }
94         else{
95             regla10(cambio);
96         }
97     }
98 }
99
100 //Regla6
101 public static void regla6(String cadena){
102     System.out.println("T-> i");
103     String cambio = cadena.replace("i-T", "T-T");
104     System.out.println("cadena: "+cambio);
105     if(cambio.length()!=2){
106         regla8(cambio);
107     }
108 }
109
110 //Regla7
111 public static void regla7(String cadena){
112     System.out.println("T-> i");
113     String cambio = cadena.replace("i+T", "T+T");
114     System.out.println("cadena: "+cambio);
115     if(cambio.length()!=2){
116         regla9(cambio);
117     }
118 }
119
120 //Regla8
121 public static void regla8(String cadena){
122     System.out.println("E-> T");
123     String cambio = cadena.replace("T-T", "E-T");
124     System.out.println("cadena: "+cambio);
125     if(cambio.length()!=2){
126         regla3(cambio);
127     }
128 }
129
130 //Regla9
131 public static void regla9(String cadena){
132     System.out.println("E-> T");
133     String cambio = cadena.replace("T+T", "E+T");
134     System.out.println("cadena: "+cambio);
135     if(cambio.length()!=2){
136         regla4(cambio);
137     }
138 }
139
140 //Regla10
141 public static void regla10(String cadena){
142     System.out.println("T-> i");
143     String cambio = cadena.replace("i", "T");
144     System.out.println("cadena: "+cambio);
145     if(cambio.length()!=2){
146         regla8(cambio);
147     }
148 }
149
150 //Regla11
151 public static void regla11(String cadena){
152     System.out.println("\n*****La cadena si fue aceptada*****\n");
153 }
154
155
156

```

```

157
158
159 public static void analizador() throws NoSuchFieldException, IOException{
160     String cadena;
161     int linea = 0, parentesis;
162     FileReader prueba = new FileReader("./prueba.txt");
163     BufferedReader b = new BufferedReader(prueba);
164     while((cadena = b.readLine()) != null){
165         linea++;
166         //esta linea de codigo delimita la expresion regular del ejemplos solicitado
167         if(cadena.matches("i[+|-]?[(\\*i[+|-]i[])+[;]")){ //i-(i+i)
168             //aqui mostrara la cadena solicitada
169             System.out.println("\\nEjemplo: "+cadena);
170             if(cadena.contains("(")){ //buscara la presencia de parentesis
171                 parentesis = CP(cadena);
172                 if(parentesis != 0){
173                     System.out.print("Error en la linea: "+linea+". La cadena es: "+cadena+"\\n");
174                 }
175                 else{
176                     flag = 0;
177                     regla1(cadena);
178                 }
179             }
180             else{
181                 regla1(cadena);
182             }
183         }
184         else{
185             System.out.println("Error en la lina: "+linea+". La cadena es: "+cadena+"\\n");
186         }
187     }
188     b.close();
189 }
190
191 //Estado para los parentesis. Verifica que los parentesis
192 //esten balanceados
193 public static int CP(String cadena){
194     int i = 0, f = 0;
195     while(cadena.length()-1 != f){
196         switch (cadena.charAt(f)){
197             case '(' -> {
198                 i++;
199                 f++;
200             }
201             case ')' -> {
202                 i--;
203                 f++;
204             }
205             default -> f++;
206         }
207     }
208     return i;
209 }
210 }

```

3. Análisis de resultados

A continuación el árbol sintáctico hecho de manera manual junto con su análisis:

Gramática.

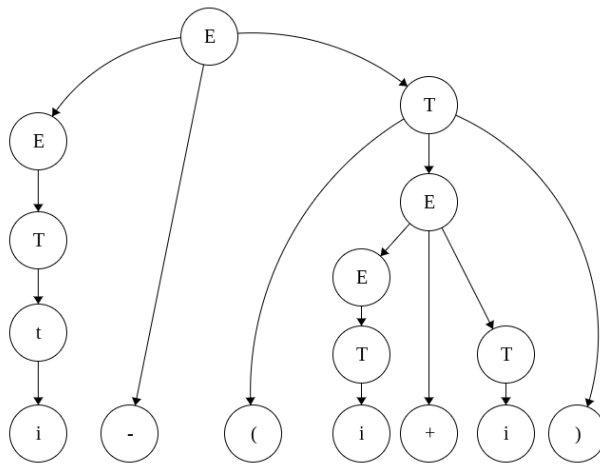
- 1) $E \rightarrow T$
- 2) $E \rightarrow E + T$
- 3) $E \rightarrow E - T$

$$5) E \rightarrow (E)$$

Cadena de Entrada.

$$i - (i + \mathbf{3}, \mathbf{4})$$
$$4) T \rightarrow i - (i + T)$$
$$4) T \rightarrow i - (T + T)$$
$$1) E \rightarrow i - (E + T)$$
$$2)E \rightarrow i - (E)$$
$$5) T \rightarrow i - i + T$$
$$4) T \rightarrow T - T$$
$$1) E \rightarrow E - T$$
$$3) E \rightarrow E$$

Cadena transformada con la gramática para realizar el analizador sintáctico ascendente:

$$i = (i + i)$$
$$3) E \rightarrow E - T$$
$$1) E \rightarrow T - T$$
$$4) T \rightarrow i - T$$
$$5) T \rightarrow i - (E)$$
$$2) E \rightarrow i - (E + T)$$
$$1)E \rightarrow i - (T + T)$$
$$4) T \rightarrow i - (i + T)$$
$$4)T \rightarrow i - (i + i)$$


Árbol Sintáctico Ascendente.

4. Conclusiones

En este proyecto se pusieron a prueba los conceptos vistos en clase y se pudo comprender de una manera práctica las fases que lleva a cabo el compilador para realizar la transición entre código fuente a código máquina, que es un proceso que a simple vista no se distingue pero que conociendo estos conceptos de analizador léxico, sintáctico y semántico podemos ver que es un proceso no tan sencillo.

Bibliografía

- [1] AHO, ALFRED V, (2008) *Compiladores. Principios, técnicas y herramientas.*, 2da ed. México *Person Education*