



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1

MANUAL TÉCNICO

Henry Ronely Mendoza Aguilar
Carné: 202004810
PROYECTO 1

Objetivos y Alcances del Sistema

Objetivo Principal

- Brindar la solución optima para la construcción de autómatas finitos deterministas y no deterministas por medio del método de Thompson y Árbol.

Objetivos Específicos

- Diseñar una plataforma agradable e intuitiva por medio del lenguaje de programación Java.
- Aprovechar de mejor manera los recursos brindados por el entorno de trabajo NetBeans
- Programar un sistema funcional para cualquier usuario y hardware en el que se utilice.

Requisitos del Hardware

- 512 MB de RAM
- 20MB de disco duro para almacenamiento del software (ya que puede aumentar con la generación de imágenes).
- Procesador Pentium 2 a 266Mhz

Requisitos del Software

- Java
- Graphviz
- Windows 10 (64-bit), Linux o Mac OS

Librerías Empleadas

Graphviz:

Graphviz (abreviatura de Graph Visualization Software) es un paquete de herramientas de código abierto iniciado por AT&T Labs Research para dibujar gráficos especificados en scripts de lenguaje DOT que tienen la extensión de nombre de archivo "gv". También proporciona bibliotecas para que las aplicaciones de software utilicen las herramientas. Graphviz es un software gratuito con licencia de Eclipse Public License.

Formato JSON:

JSON (JavaScript Object Notation) es un formato de archivo estándar abierto y un formato de intercambio de datos que utiliza texto legible por humanos para almacenar y transmitir objetos de datos que consisten en pares atributo-valor y arreglos (u otros valores serializables). Es un formato de datos común con diversos usos en el intercambio electrónico de datos, incluido el de aplicaciones web con servidores.

JSON es un formato de datos independiente del idioma. Se derivó de JavaScript, pero muchos lenguajes de programación modernos incluyen código para generar y analizar datos en formato JSON. Los nombres de archivo JSON usan la extensión. json.

JFLEX:

JFlex es una herramienta para la generación de analizadores léxicos escritos en java. A partir de un fichero de especificación que describe las características léxicas de un lenguaje, JFlex genera un código fuente compilable que puede ser utilizado como analizador léxico.

CUP:

Es un parser-generator. Es un analizador sintáctico que construye un parser para gramáticas tipo LALR(1), con código de producción y asociación de fragmentos de código JAVA. Cuando una producción en particular es reconocida, se genera un archivo fuente Java, parser.java que contiene una clase parser, con un método Symbol parser().

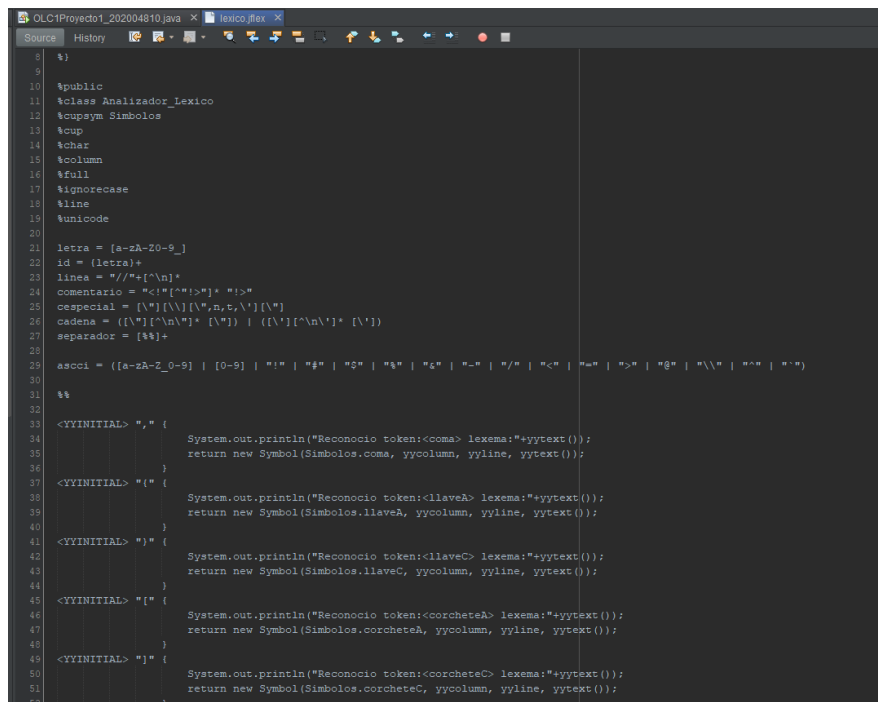
Descripción de Métodos

Para el desarrollo de la aplicación se emplearon los métodos de lectura de archivos de texto plano que posee Java las cuales son las pertenecientes al paquete IO. Por medio de un JFileChooser se escogen los archivos para una interfaz gráfica.

```
JFileChooser fc = new JFileChooser();
int op = fc.showOpenDialog(this);
if (op == JFileChooser.APPROVE_OPTION) {
    ruta=fc.getSelectedFile().getAbsolutePath();
    archivo = fc.getSelectedFile();
    String nombre = fc.getSelectedFile().getName();
    filename = nombre.replaceAll(".exp", "");
}

fr = new FileReader(archivo);
br = new BufferedReader(fr);
String linea;
while ((linea = br.readLine()) != null) {
    file += "\n"+linea;
}
this.jTextArea2.setText(file);
```

Anlizador Léxico: Empleando la librería de JFlex se realizó un archivo con extensión jflex en la cual se definieron los tokens para la validación del análisis léxico.



```
1 8
9
10 %public
11 %class Analizador_Lexico
12 %cupsym Simbolos
13 %cup
14 %char
15 %column
16 %full
17 %ignorecase
18 %line
19 %unicode
20
21 letra = [a-zA-Z0-9_]
22 id = (letra)+
23 linea = "//"+[">"]* ">"
24 comentario = "<"+[">"]* ">"
25 especial = [{"\\"}{"\\n","\\t","\\r"}{"\\"}
26 cadena = ({\\"}{"\\n\\"}* {"\\n"}) | ({\\"}{"\\n\\"}* {"\\n"})
27 separador = [{"\\s"}]
28
29 asoci = ({a-zA-Z_0-9} | [0-9] | "." | "&" | "$" | "&" | "&" | "-" | "/" | "<" | "=" | ">" | "@" | "\\n" | "\\r" | "\\t")
30
31 %%
32
33 <YINITIAL> "," {
34     System.out.println("Reconocio token:<coma> lexema:"+yytext());
35     return new Symbol(Simbolos.coma, yycolumn, yyline, yytext());
36 }
37 <YINITIAL> "{" {
38     System.out.println("Reconocio token:<llaveA> lexema:"+yytext());
39     return new Symbol(Simbolos.llaveA, yycolumn, yyline, yytext());
40 }
41 <YINITIAL> "}" {
42     System.out.println("Reconocio token:<llaveC> lexema:"+yytext());
43     return new Symbol(Simbolos.llaveC, yycolumn, yyline, yytext());
44 }
45 <YINITIAL> "[" {
46     System.out.println("Reconocio token:<corcheteA> lexema:"+yytext());
47     return new Symbol(Simbolos.corcheteA, yycolumn, yyline, yytext());
48 }
49 <YINITIAL> "]" {
50     System.out.println("Reconocio token:<corcheteC> lexema:"+yytext());
51     return new Symbol(Simbolos.corcheteC, yycolumn, yyline, yytext());
52 }
```

Analizador Sintáctico: Empleando la librería de CUP se desarrollo en un archivo con extensión cup la gramática para la ejecución del análisis sintáctico.

```
terminal id, coma, conj, cadena, llaveA, llaveC, rango, or, klinee, positiva, interrogacion, concat, dospuntos, puntoycoma, sep
non terminal INICIO,E,E_2, SENTENCIA, CONJUNTOS, EXPRESION, VALIDACION, ER, ER_2, VALOR, RCONJ, RCONJ_2, VCONJ;

start with INICIO:

INICIO::= llaveA:c E (: Token t1 = new Token("llaveA",String.valueOf(c),cright,cleft);
                        tokens.add(t1);
                        System.out.println("Fin de analisis de entrada");
                        :)
;

E::= SENTENCIA E_2
;

E_2::= SENTENCIA E_2
| llaveC:c (
                        Token t1 = new Token("llaveC",String.valueOf(c),cright,cleft);
                        tokens.add(t1);
                        )
;

SENTENCIA::= CONJUNTOS
| EXPRESION
| VALIDACION
| separador:c (
                        Token t1 = new Token("separador",String.valueOf(c),cright,cleft);
                        tokens.add(t1);
                        )
;

CONJUNTOS::= conj:c dospuntos:d idi:i flecha:f RCONJ (
                        Token t1 = new Token("conj",String.valueOf(c),cright,cleft);
                        Token t2 = new Token("dospuntos",String.valueOf(d),dright,dleft);
                        Token t3 = new Token("id",String.valueOf(i),iright,ileft);
                        Token t4 = new Token("flecha",String.valueOf(f),fright,fleft);
```

Método del árbol:

Para el método del árbol se utilizó un algoritmo basado en la detección de los caracteres que incluye la expresión polaca que ingresa dentro del archivo exp así mismo con la repetición continua de este método por medio de ciclos se asignaron los valores de siguientes para el armado del grafo.

```
public static Hoja[] setTree(){
    Hoja lnodes[]=hojas.clone();
    Hoja aux[]=lnodos.clone();
    int i = 0;
    while (aux.length!=0){
        while (i<aux.length){
            if((" "+aux[i].getValor())!=" "+aux[i].getValor()) && (!Operador(aux[i+1].getValor())) && (!Operador(aux[i+2].getValor())){
                if("V".equals(aux[i+1].getAnulable()) && "V".equals(aux[i+2].getAnulable())){
                    for (int j = 0; j < lnodes.length; j++) {
                        if (aux[i].getId()==lnodos[j].getId()){
                            aux[i].setAnulable("V");
                            lnodes[i].setAnulable("V");
                        }
                    }
                }
            }
            else{
                for (int j = 0; j < lnodes.length; j++) {
                    if (aux[i].getId()==lnodos[j].getId()){
                        aux[i].setAnulable("F");
                        lnodes[i].setAnulable("F");
                    }
                }
            }
            aux[i].setValor(aux[i].getValor()+" "+aux[i+1].getValor());
            aux.removeElement(aux, (i+1));
            aux.removeElement(aux, (i+1));
            break;
        }
        else if((" "+aux[i].getValor())!=" "+aux[i].getValor()) && "V".equals(aux[i].getValor()) && (!Operador(aux[i+1].getValor())){
            if("V".equals(aux[i+1].getAnulable()) && "V".equals(aux[i+2].getValor())){
                for (int j = 0; j < lnodes.length; j++) {
                    if (aux[i].getId()==lnodos[j].getId()){
                        aux[i].setAnulable("V");
                        lnodes[i].setAnulable("V");
                    }
                }
            }
            else if("F".equals(aux[i+1].getAnulable()) && "V".equals(aux[i+2].getValor())){
                for (int j = 0; j < lnodes.length; j++) {
                    if (aux[i].getId()==lnodos[j].getId()){
                        aux[i].setAnulable("F");
                        lnodes[i].setAnulable("F");
                    }
                }
            }
        }
        aux[i].setValor(aux[i].getValor()+" "+aux[i+1].getValor());
        aux.removeElement(aux, (i+1));
        break;
    }
}
```

Para las transiciones fue necesario la implementación del algoritmo anterior junto a una matriz la cual permitía guardar las transiciones en el orden adecuado, esta misma matriz fue empleada para la construcción del AFD ya que no se deben modificar las transiciones.

```
public static void Transiciones(Siguientes[] sig, String nombre){

    List<Transiciones> trans = new ArrayList<>();
    String t[]=hojas[0].getPrimer().split(",");
    String term = "";
    for (int i = 0; i < t.length; i++) {
        for (int j = 0; j < hojas.length; j++) {
            if(t[i].equals(hojas[j].getVh())){
                term+=hojas[j].getValor()+",";
            }
        }
    }
    Transiciones S0 = new Transiciones("S0",t,term.split(","));
    trans.add(S0);

    for (int i = 0; i < sig.length; i++) {
        String temp[]=sig[i].getSiguientes();
        String terminales = "";
        for (int j = 0; j < temp.length; j++) {
            for (int k = 0; k < hojas.length; k++) {
                String x = temp[j].replaceAll(" ", "");
                if(x.equals(hojas[k].getVh()) && !"".equals(x) && !" ".equals(x)){
                    terminales+=hojas[k].getValor()+",";
                }
            }
        }
        Transiciones S = new Transiciones("S"+(i+1),temp,terminales.split(","));
        trans.add(S);
    }
    trans=duplicado(trans);
}
```

```

    }
    for (int i = 1; i < matrix.length; i++) {
        if (matrix[i][0].equals(trans.get(i-1).getEstado())) {
            String movs [] = trans.get(i-1).getMovimiento();
            for (int j = 0; j < movs.length; j++) {
                for (int k = 0; k < hojas.length; k++) {
                    String x = movs[j].replaceAll(" ", "");
                    if (x.equals(hojas[k].getVh())) {
                        String colum = hojas[k].getValor();
                        for (int l = 1; l < matrix[0].length; l++) {
                            if(colum.equals(matrix[0][l])){
                                int max = Integer.parseInt(hojas[k].getVh());
                                if(max>=trans.size()){
                                    matrix[i][l]="S"+String.valueOf(trans.size()-1);
                                }else{
                                    matrix[i][l]="S"+hojas[k].getVh();
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
}
```

Uso de Graphviz: Para el uso de graphviz se utilizó la escritura de archivos y posterior mente la ejecución de consola, para la generación de los árboles, AFD, AFND y tablas de transición y siguientes.

```
FileWriter reportel = null;
PrintWriter pw = null;
try{

    reportel = new FileWriter("estructuras.dot");
    pw = new PrintWriter(reportel);

    pw.println("digraph G (");
    pw.println("label = \"Estado de Memoria de las Estructuras\"");
    pw.println("fontsize=\"20\"");

    pw.println("subgraph cluster_0 (");
    pw.println("node [style=filled];");
    Cola.Nodo aux_1 = clientes.first;
    while(aux_1!=null){
        Cliente c = (Cliente) aux_1.data;
        pw.println("C"+c.id+"[label = \"Cliente "+c.id+"\n"+c.name+"\"");
        aux_1=aux_1.next;
    }
    aux_1=clientes.first;
```

```
    aux_5=c_espera.first;
    while(aux_5!=null){
        Cliente c = aux_5.cliente;
        if(aux_5.next!=null){
            Cliente c2=aux_5.next.cliente;
            pw.println("rank=same(Es"+c.id+"->Es"+c2.id+"");
        }
        aux_5=aux_5.next;
    }

    pw.println("label = \"Lista Clientes en Espera\"");
    pw.println("color=blue");

    pw.println(")");

    }catch(Exception e){
        e.printStackTrace();
    }finally{
        try{
            if(null != reportel){
                reportel.close();
                ProcessBuilder buil = new ProcessBuilder("dot","-Tpng","-o","estructuras.png","estructuras.dot");
                buil.redirectErrorStream(true);
                buil.start();
            }
        }catch(Exception e2){
            e2.printStackTrace();
        }
    }
}
```

Análisis de cadenas: Para el análisis de cadenas se implementaron buscadores y el algoritmo empleado en el Método del árbol para convertir la expresión polaca en una expresión regular, así mismo se modificaron caracteres de entrada para la detección de la cadena por medio de las librerías implementadas en java para la lectura de expresiones regulares.

```

for (validacion t : val) {
    for (expresion ex : er) {
        if(ex.getNombre().equals(t.ER)){
            String copy[] = ex.getPolaca().clone();
            String arr [] = ex.getPolaca();
            for (int j = 0; j < arr.length; j++) {
                if(arr[j].equals("\"\\\"\\\\'\"")){
                    arr[j]="\\\"\\\\'";
                }else if(arr[j].equals("\"\\\"\\\\\\\"\"")){
                    arr[j]="\\\"\\\\\\\"";
                }else if(arr[j].equals("\"\\\"\\\\n\"")){
                    arr[j]="\\\"\\\\n";
                }else if(arr[j].equals("\"\\\"\\\\.\"")){
                    arr[j]="\\\"\\\\.";
                }else{
                    arr[j]=arr[j].replaceAll("\\\"", "");
                }
            }
            String convertir[]=ex.estructura(arr);
            String temp = convertir[0];
            temp = temp.replaceAll("\\\"\\\\.", ".");
            for (conjuntos c : conj) {
                if(temp.contains(c.getNombre())){
                    temp = temp.replace("(" + c.getNombre() + ")", c.getRango());
                }
            }

            String entrada = t.getCadena().replaceAll("\\\"", "");
            Pattern pattern = Pattern.compile(temp);
            Matcher matcher = pattern.matcher(entrada);
            boolean match = matcher.matches();

            if (match) {
                salida+="Cadena: "+entrada+" con ER: "+t.getER()+" es: VALIDA\n";
                json+="";
                json+="\"Valor\": \""+entrada+"\", \n";
                json+="\"ExpresionRegular\": \""+t.getER()+"\", \n";
                json+="\"Resultado\": \"Cadena Válida\" \n";
            }else{
                salida+="Cadena: "+entrada+" con ER: "+t.getER()+" es: INVALIDA\n";
                json+="";
                json+="\"Valor\": \""+entrada+"\", \n";
            }
        }
    }
}

```

Almacenamiento de datos: Para el almacenamiento de datos se utilizó la programación orientada a objetos en el cual se creó una clase llamada Conjuntos para la información de los conjuntos planteados en el archivo de entrada, Expresión para la información de las expresiones del archivo de entrada, Validación para los datos de validación en el archivo de entrada.


```

public class conjuntos {

    String nombre, rango;

    public conjuntos(String nombre, String rango) {
        this.nombre = nombre;
        this.rango = rango;
    }

    public String info() { ...6 lines }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

```

```

public class expression {

    String nombre;
    String [] polaca;

    public expression(String nombre, String [] polaca) {
        this.nombre = nombre;
        this.polaca = polaca;
    }

    public String[] estructura(String[] arr) { ...30 lines }

    public void graph() { ...151 lines }

    public void Thompson() { ...97 lines }

    public static String[] removeElement( String[] arr, int index ) { ...5 lines }

    public static boolean Operador(String op) { ...14 lines }

    public String info() { ...15 lines }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String[] getPolaca() {
        return polaca;
    }

    public void setPolaca(String[] polaca) {
        this.polaca = polaca;
    }

}

```

```

public class validation {

    String ER, cadena;

    public validation(String ER, String cadena) {
        this.ER = ER;
        this.cadena = cadena;
    }

    public String info() { ...6 lines }

    public String getER() {
        return ER;
    }

    public void setER(String ER) {
        this.ER = ER;
    }

    public String getCadena() {
        return cadena;
    }

    public void setCadena(String cadena) {
        this.cadena = cadena;
    }

}

```

Archivo html y json: Para la escritura de los archivos de salida se emplearon las librerías de java además de la implementación adecuada de la estructura de los correspondientes archivos.

```
FileWriter reportel = null;
PrintWriter pw = null;
try{

    reportel = new FileWriter("ERRORES_202004810\\errores"+filename+".html");
    pw = new PrintWriter(reportel);

    String html = "<!DOCTYPE html>\n" +
        <html lang="en">\n" +
        <head>\n" +
        <meta charset="UTF-8">\n" +
        <meta http-equiv="X-UA-Compatible" content="IE=edge">\n" +
        <meta name="viewport" content="width=device-width, initial-scale=1.0">\n" +
        <link rel="stylesheet" href="style.css">\n" +
        <title>Reporte</title>\n" +
        </head>\n" +
        <body>\n" +
        <div class="container-table">\n" +
        <div class="table__title1">\n" +
        Reporte de Errores\n" +
        </div></div>";

    html+="
```