Manual Técnico

Proyecto 1

Kevin Steve Martinez Lemus
[OLC1] Universidad De San Carlos De Guatemala

Índice

I. Introducción

El presente documento describe los aspectos técnicos informáticos del sistema de información. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

Objetivo

Instruir el uso adecuado del Sistema de Información, para el acceso oportuno y adecuado en la modificación de este, mostrando la descripción de los archivos relevantes del sistema los cuales nos orienten en la configuración y soporte de este.

Requerimientos

- Tener Java (versión 8 o superiores) instalado.
- Se recomienda tener un editor de código (NetBeans 8.2 u otro) para ver y editar el código de este programa.

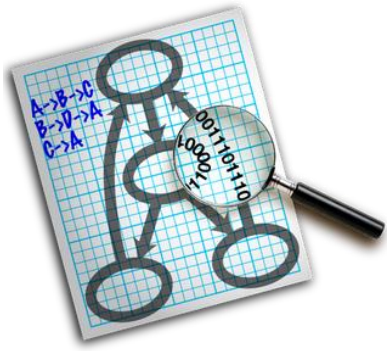
II. Descripción Del Sistema

1. Descripción del Contenido del Sistema

Este es un sistema en el cual el usuario interactúa directamente con una interfaz gráfica, este programa permite la lectura de un código escrito por el usuario, lo analiza y crea autómatas con las expresiones regulares descritas, verifica entradas con las expresiones regulares, abre y guarda archivos .exp.

2. Lenguajes Utilizados

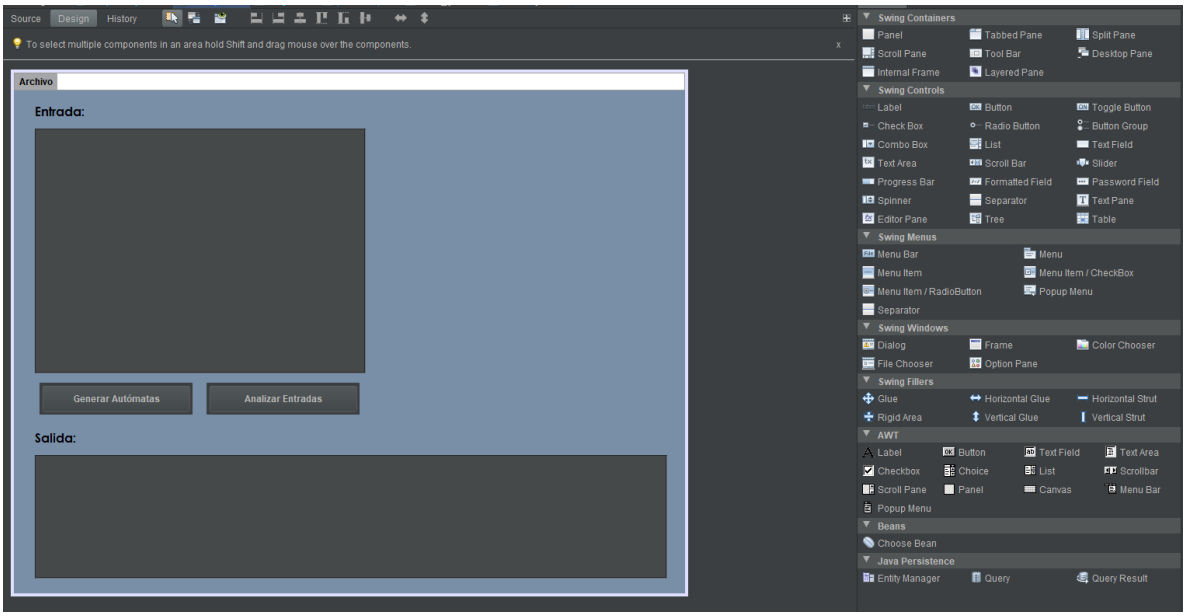
Para la creación de este sistema todas las funcionalidades se realizaron en lenguaje de JAVA y también se utilizó Graphviz para la creación de reportes. Utilizando un enfoque al paradigma de programación POO (Programación Orientada a Objetos). Creando las diferentes estructuras de datos para almacenar la información.



Todo realizado en el IDE NetBeans 8.2:

3. Interfaz Gráfica

Para la realización de la interfaz gráfica se utilizo el entorno de Netbeans, creando un nuevo JFrame Form y con drag and drop de los competentes que se utilizaron:



4. Analizadores

4.1 Analizador Léxico

Para el análisis léxico se utilizó la librería de jflex-full-1.8.2 en el cual se describe los caracteres y cadenas de texto que el lenguaje puede reconocer.

4.2 Analizador Sintáctico

Para el análisis sintáctico se utilizo la librería java-cup-11b en el cual se describe la gramática correspondiente al lenguaje que se desea leer, utilizando los tokens leídos por el analizador léxico.

Para crear los analizadores se utilizó en siguiente código:

```
try {

    String ruta = "src/analizadores/";
    String opcFlex[] = {ruta+"lex.jflex","-d",ruta};
    jflex.Main.generate(opcFlex);

    String opcCup[]={"-destdir",ruta, "-parser","Parser",ruta+"parser.cup"};
    java_cup.Main.main(opcCup);

} catch (Exception e) {
}
```

Y se utilizo una clase AnalizadorLenguaje para que administrara la instancia de los analizadores, el análisis del texto mediante estos, y guardar los errores que se produjeron.

```
public class AnalizadorLenguaje {
    public static AnalizadorLenguaje analizador;
    public static LinkedListError errores;
    public static LinkedListExpresion expresiones;

    public void analize(String text){
        try {
            System.out.println("Inicio de analisis");
            errores = new LinkedListError();
            expresiones = new LinkedListExpresion();
            Scanner scanner = new Scanner(new BufferedReader(new StringReader(text)));
            Parser parser = new Parser(scanner);
            parser.parse();
            System.out.println("Fin de analisis");
        } catch (Exception e) {
        }
    }

    public static AnalizadorLenguaje getInstance(){
        if(analizador == null){
            errores = new LinkedListError();
            analizador = new AnalizadorLenguaje();
            expresiones = new LinkedListExpresion();
        }
        return analizador;
    }
```

5. Gráficas

Para la graficar las estructuras correspondientes se utilizaron 2 métodos principales, el escribir y el graficar, en donde el método escribir, escribe el txt en la carpeta correcta, que luego el método de graficar, grafica por medio de graphviz. Para el método de escribir se utilizó:

```
public static void escribirDot(String title, String resultado, String type){
    try {
        String ruta;
        switch(type){
            case "afnd":
                ruta = System.getProperty("user.dir") + "\\AFND_202004816\\"+title+".txt";
                break;
            case "arbol":
                ruta = System.getProperty("user.dir") + "\\ARBOLES_202004816\\"+title+".txt";
                break;
            case "s":
                ruta = System.getProperty("user.dir") + "\\SIGUIENTES_202004816\\"+title+".txt";
                break;
            case "t":
                ruta = System.getProperty("user.dir") + "\\TRANSICIONES_202004816\\"+title+".txt";
                break;
            case "afd":
                ruta = System.getProperty("user.dir") + "\\AFD_202004816\\"+title+".txt";
                break;
            case "json":
                ruta = System.getProperty("user.dir")+"\\SALIDAS_202004816\\"+title+".json";
                break;
            case "html":
                ruta = System.getProperty("user.dir")+"\\ERRORES_202004816\\"+title+".html";
                break;
            default:
                ruta = System.getProperty("user.dir") + "\\"+title+".txt";
                break;
        }

        File file = new File(ruta);

        FileWriter fw = new FileWriter(file);
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write(resultado);
        bw.close();

    } catch (Exception e) {
        e.printStackTrace();
    }
```

En donde se especifica, dependiendo de los parámetros, la ubicación y nombre del archivo que se está guardando para luego graficar. Para el método de graficar sucede los mismo, se especifica que tipo de grafica se va a estructurar y con eso se elige que archivo es el que se va a graficar:

```java
public static void graficarImagen(String title, String type){
    try {

        String dotPath = "C:\\Program Files\\Graphviz\\bin\\dot.exe";
        String fileInputPath;
        String fileOutputPath;

        switch(type){
            case "afnd":
                fileInputPath = System.getProperty("user.dir") + "\\AFND_202004816\\"+title+".txt";
                fileOutputPath = System.getProperty("user.dir") + "\\AFND_202004816\\"+title+".jpg";
                break;
            case "arbol":
                fileInputPath = System.getProperty("user.dir") + "\\ARBOLES_202004816\\"+title+".txt";
                fileOutputPath = System.getProperty("user.dir") + "\\ARBOLES_202004816\\"+title+".jpg";
                break;
            case "s":
                fileInputPath = System.getProperty("user.dir") + "\\SIGUIENTES_202004816\\"+title+".txt";
                fileOutputPath = System.getProperty("user.dir") + "\\SIGUIENTES_202004816\\"+title+".jpg";
                break;
            case "t":
                fileInputPath = System.getProperty("user.dir") + "\\TRANSICIONES_202004816\\"+title+".txt";
                fileOutputPath = System.getProperty("user.dir") + "\\TRANSICIONES_202004816\\"+title+".jpg";
                break;
            case "afd":
                fileInputPath = System.getProperty("user.dir") + "\\AFD_202004816\\"+title+".txt";
                fileOutputPath = System.getProperty("user.dir") + "\\AFD_202004816\\"+title+".jpg";
                break;
            default:
                fileInputPath = System.getProperty("user.dir") + "\\"+title+".txt";
                fileOutputPath = System.getProperty("user.dir") + "\\"+title+".jpg";
                break;
         }

        String tParam = "-Tjpg";
        String tOParam = "-o";

        String[] cmd = new String[5];
        cmd[0] = dotPath;
        cmd[1] = tParam;
```

6. Expresiones Regulares

Para el guardado de las expresiones regulares se utilizó una clase Expresión estructurada de la siguiente manera:

```java
public class Expresion {
    String operador;
    Object primero;
    Object siguiente;

    public Expresion(String operador, Object primero, Object siguiente) {
        this.operador = operador;
        this.primero = primero;
        this.siguiente = siguiente;
    }

    public Expresion(String operador, Object primero) {
        this.operador = operador;
        this.primero = primero;
        this.siguiente = null;
    }
```

Este se comporta como un árbol binario en donde la información del nodo es el operador, mientras el primero y siguiente pueden tomar el valor de otra expresión, así como de datos.

7. Archivos De Salida

Estos de dividen en dos: El reporte de errores y de verificación de entradas. Estos utilizan el mismo método de escribir que se utilizo para graficar, con diferencia que el reporte de errores se grafica con una tabla en html y la verificación de entradas se escribe en un json. Para el html:

```
public void escribirErrores(ArrayList<Error_> errores){
    String resultado = "<!DOCTYPE>\n<html>\n    <head>\n <title style=>Errores</title>\n   "
            + "<!-- CSS only -->\n" +
            "<link href=\"https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css\" rel=\"stylesheet\" integrity=\"sha384-1BmE4kWBq78iYhFldvF
            + "</head>\n   <body style=\"width:80%;margin:auto;\">";

    resultado += "\n    <br><center><h1>Lista De Errores</h1>\n <br>    <table class=\"table table-striped\">\n";
    resultado += "<thead class=\"table-dark\"><td>#</td><td>Tipo</td><td>Descripción</td><td>Linea</td><td>Columna</td></thead>\n<tbody>";
    for(int i = 0; i< errores.size(); i++){
        resultado += "<tr><td>"+(i+1)+"</td><td>"+errores.get(i).getType()+"</td><td>"+errores.get(i).getMessage()+"</td><td>"+errores.get(i).getFila()+"</td
    }

    resultado += "</tbody></table></body></html>";

    Proyecto1.escribirDot("Errores", resultado, "html");
}
```

Y para el json:

```
public static void verificarEntradas(){
    String salidajson = "[";
    for(int i = 0 ; i<entradas.size();i++){
        for(int j = 0; j<expresiones.size();j++){
            if(entradas.get(i).getExp().equals(expresiones.get(j).getId())){
                //System.out.println("Entrada: " + entradas.get(i).getLexema() + " - Con Expresion: " + expresiones.get(j).g
                String[] lex = entradas.get(i).getLexema().split("\"");
                if(verificar(expresiones.get(j).getExpresion(),lex[1])){
                    //System.out.println("Entrada: " + lex[1] + " Válida Con Expresión: " + entradas.get(i).getExp());
                    salida += ">>>  Entrada: " + lex[1] + " Válida Con Expresión: " + entradas.get(i).getExp() + ".\n";
                    salidajson += "\n   {\n     \"Valor\":"+entradas.get(i).getLexema()+",\n"
                            + "     \"ExpresionRegular\":\""+entradas.get(i).getExp()+"\",\n"
                            + "     \"Resultado\":\"Cadena Válida\"\n   },";
                }else{
                    //System.out.println("Entrada: " + lex[1] + " NO Válida Con Expresión: " + entradas.get(i).getExp());
                    salida += ">>>  Entrada: " + lex[1] + " NO Válida Con Expresión: " + entradas.get(i).getExp() + ".\n";
                    salidajson += "\n   {\n     \"Valor\":"+entradas.get(i).getLexema()+",\n"
                            + "     \"ExpresionRegular\":\""+entradas.get(i).getExp()+"\",\n"
                            + "     \"Resultado\":\"Cadena No Válida\"\n   },";
                }
            }
        }
    }

    salidajson = salidajson.substring(0,salidajson.length()-1);
    salidajson += "\n]";
    escribirDot("Salida", salidajson, "json");
}
```