Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Ingeniería en Ciencias y Sistemas
Lab. Organización de lenguajes y compiladores

MANUAL TECNICO

Justin Josue Aguirre Román
202004734

INTRODUCCION

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, árboles y grafo de autómata utilizado para su desarrollo.

OBJETIVO DE ESTE

El objetivo primordial de este Manual es ayudar y guiar al técnico a informarse y utilizar herramientas utilizadas en el software auxiliar para el área de compiladores de ingeniería en ciencias y sistemas, para de esa manera poder hacer uso de la información deseada para poder despejar todas las dudas existentes y para poder comprender:

• Guía para gestión de herramientas para poner en funcionamiento el sistema de manejo para impresiones.

• Conocer cómo utilizar el sistema, mediante una descripción detallada e ilustrada de las opciones.

• Conocer el alcance de toda la información por medio de una explicación detallada e ilustrada de cada una de las páginas que lo conforman el manual técnico.

REQUERIMIENTOS

El sistema pude ser instalado en cualquier sistema operativo que cumpla con los siguientes requerimientos:

a. Sistema Operativo: Cualquiera con una fecha de salida del 2014 en adelante.

a. Windows

i. Windows 10 (8u51 y superiores)

ii. Windows 8.x (escritorio)

iii. Windows 7 SP1

iv. Windows Vista SP2

v. Windows Server 2008 R2 SP1 (64 bits)

vi. Windows Server 2012 y 2012 R2 (64 bits)

vii. RAM: 128 MB

viii. Espacio en disco: 124 MB para JRE; 2 MB para Java Update

ix. Procesador: Mínimo Pentium 2 a 266 MHz

x. Exploradores: Internet Explorer 9 y superior, Firefox

b. Linux

i. Oracle Linux 5.5+1

ii. Oracle Linux 6.x (32 bits), 6.x (64 bits)2

iii. Oracle Linux 7.x (64 bits)2 (8u20 y superiores)

iv. Red Hat Enterprise Linux 5.5+1 6.x (32 bits), 6.x (64 bits)2

v. Red Hat Enterprise Linux 7.x (64 bits)2 (8u20 y superiores)

vi. Suse Linux Enterprise Server 10 SP2+, 11.x

vii. Suse Linux Enterprise Server 12.x (64 bits)2 (8u31 y superiores)

viii. Ubuntu Linux 12.04 LTS, 13.x

ix. Ubuntu Linux 14.x (8u25 y superiores)

x. Ubuntu Linux 15.04 (8u45 y superiores)

xi. Ubuntu Linux 15.10 (8u65 y superiores)

METODOS UTILIZADOS

El software que requiere la Facultad de ingeniería conlleva distintos métodos principales que hacen posible su funcionamiento en sus distintas etapas de desarrollo y ejecución.

Abrir Archivo

Tras hacer la validación del botón seleccionado se hará uso de JFileChooser para seleccionar la ruta y un FileReader para obtener los datos dentro del archivo, activando los analizadores sintáctico y léxico que formaran y almacenaran en Listas dinámicas, los tokens necesarios para el resto de la ejecución del programa.

```
public void Analizar() {
    try {
        lexico = new Analizador_Lexico(new BufferedReader(new FileReader(archivo)));
        sintactico = new Analizador_Sintactico(lexico);
        sintactico.parse();

        if ( sintactico.errores.size() > 0) {
            contenido = "";
            JOptionPane.showMessageDialog(this, "GENERANDO REPORTE DE ERRORES", "ERROR ENCONTRADO", WARNING_MESSAGE);
            ReporteErrores();
            Errores = true;
        } else {
            ReporteErrores();
            Errores = false;
            ErroresLex = lexico.errores;
            ErroresSintact = sintactico.errores;
            CONJUNTOS = sintactico.CONJUNTOS;
            EXPRESIONES = sintactico.EXPRESIONES;
            PRUEBAS = sintactico.PRUEBAS;
            GuardarPolaca();

            JOptionPane.showMessageDialog(this, "ANALISIS COMPLETADO", "SIN ERRORES ENCONTRADOS", WARNING_MESSAGE);
            GenAutomata.setEnabled(true);
        }
    } catch (Exception e) {
    }
}
```

ANALIZAR Y TRNAFORMAR

Para el manejo de los archivos y tokens se desarrolla el guardado de tokens especiales para cada tipo de dato: para los conjuntos únicamente se guardan en una lista con Tokens que guardan el tipo que ocupa, lexema y si se trata de un operador. Para las pruebas se guardan de la misma forma que los conjuntos.

Para las expresiones se utiliza un tipo de Token Expresión que guarda el nombre y

el lexema (expresión regular) tal y como viene en el archivo de entrada. Sin embargo, con ellas se lleva a cabo el proceso de **TRANSFORMAR** que pasa de notación polaca a una notación normal las expresiones regulares.

```
public void GuardarPolaca() {
    String Nombre = "";
    int c = 0, i = 0;
    ArrayList<Token> Polaca = new ArrayList<Token>();

    while (c < EXPRESIONES.size()) {

        Token aux = EXPRESIONES.get(c);

        if (aux.getTipo() == "id") {
            Nombre = aux.getLexema();
        }
        i = 0;
        if (aux.getSize() > 0) {
            while (aux.getSize() > i) {
                Polaca.add(0, aux.getCaracter(i));
                i++;
            }
            Expresiones nuevoPol = new Expresiones(Nombre, Polaca);
            Polaca = new ArrayList<Token>();
            ExpPolacas.add(nuevoPol);

        }
        System.out.println(c);
        c++;
    }

}
```

El algoritmo utilizado para ello se compone de dos validaciones que verificaran que haya una secuencia: operador-elemento-elemento, en la cual elemento son los no operadores y los operadores pueden ser concatenación o Or. También puede ser una aparición operador-elemento en el cual el operador puede ser cualquier tipo de cerradura (“*”, “+”, “?”).

Luego de encontrar alguna de estas secuencias los elementos son retirados de la lista, los operadores cambian a elemento y se concatenan en un String según su naturaleza.

```
if (tk0.Caracter.getOperador() && !tk1.Caracter.getOperador() && !tk2.Caracter.getOperador() && (!tk0.Caracter.getLexema().equals("?") && !tk0.Caracter.ge
    System.out.println("");
    System.out.println("APLICO CASO 1");

    int AuxX = x;
    int conta = Expresion.get(AuxX).Elem + 1;
    TokenCaracterCambio tkConta = Expresion.get(AuxX);
    do {
        tkConta.cambio = true;
        AuxX++;
        tkConta = Expresion.get(AuxX);
        tkConta.SetElem(Expresion.get(AuxX).Elem - 1);
    } while (tkConta.cambio && conta == Expresion.get(AuxX).Elem);
    Expresion.remove(x);
    tk0.cambio = true;
    Expresion.add(AuxX, tk0);
    tk1.cambio = true;
    tk2.cambio = true;
    Expresion.get(AuxX - 1).Caracter.setLexema("( "+Expresion.get(AuxX - 1).Caracter.getLexema() + " " + Expresion.get(AuxX).Caracter.getLexema());
    Expresion.remove(AuxX);
    Expresion.get(AuxX - 1).Caracter.setLexema(Expresion.get(AuxX - 1).Caracter.getLexema() + " " + Expresion.get(AuxX).Caracter.getLexema()+" )");
    Expresion.remove(AuxX);
    AuxX++;
    while (AuxX < Expresion.size()) {
        Expresion.get(AuxX).SetElem(Expresion.get(AuxX).Elem - 2);
        AuxX++;
    }
```

En caso de no encontrar ninguna las 3 variables auxiliares de búsqueda se moverán un espacio a la derecha en la expresión hasta encontrar concordancia o bien el auxiliar mas a la derecha exceda el tamaño de la expresión.

GENERAR ARBOL Y SIGUIENTES

Primero crea objetos de tipo Nodo los cuales se dividirán en operadores y no operadores, además de guardar el lexema de cada carácter, id o cadena de la expresión regular; en caso de ser un no operador almacenara su número de hoja, y creara su id a partir de que tipo de token se valida y el numero de apariciones de este tipo, por ejemplo: en caso de que el token recibido sea una cadena y sea la 4ta cadena que hace su aparición se guardara con el id “SCadena4”. Estos Nodos los guardara en una lista enlazada como una propiedad de la expresión validada.

Luego utiliza el mismo algoritmo de **TRANSFORMAR.**

Con la diferencia que al encontrar coincidecias escribe código grapvhiz que luego se ejecutara

```
Nodos nuevo = null;
if (op) {
    nuevo = new Nodos(tipo, lex, op, anulable);

} else {
    nuevo = new Nodos(tipo, lex, op, anulable, Hoja);
    nuevo.Primeros.add(Hoja);
    nuevo.Ultimos.add(Hoja);
}

ListaNodos.add(nuevo);
ListaNodos2.add(nuevo);
i++;
```

```
Nodos nuevo = new Nodos("EOF", "$", false, false, Hoja);
nuevo.Primeros.add(Hoja);
nuevo.Ultimos.add(Hoja);
ListaNodos2.add(nuevo);

String ult = "[" + Hoja + "]";
String prim = "[" + Hoja + "]";

contenido += "EOF[label=\"" + prim + "|{false|$|" + nuevo.Hoja + "}|" + ult + "\"];\n";

nuevo = new Nodos("SPuntoFinal", ".", true, false);
nuevo.Ultimos.add(Hoja);

Nodos tk = null;
for (int a = 0; a < ListaNodos2.size(); a++) {
    tk = ListaNodos2.get(a);
    if (ultimo.equals(tk.Nodo)) {
        break;
    }
}
```

Para la parte de siguientes únicamente al encontrar concatenación, cerradura Kleene o cerradura positiva se calcularan al nodo hoja sus siguientes escribiendo contenido en graphviz por aparte del método del árbol.

```
    if (tk0.lexema.equals("+") || tk0.lexema.equals("*")) {
        for (int a = 0; a < tkl.Ultimos.size(); a++) {
            Nodos tkplus = null;
            for (int c = 0; c < ListaNodos2.size(); c++) {
                if ((int) tkl.Ultimos.get(a) == ListaNodos2.get(c).Hoja) {
                    tkplus = ListaNodos2.get(c);
                    break;
                }
            }
            for (int b = 0; b < tkl.Primeros.size(); b++) {
                tkplus.Siguientes.add(tkl.Primeros.get(b));
            }
        }
    }

    x = 0;
    y = 1;
    z = 2;
```

GENERAR TABLA DE TRANSICIONES Y AUTOMATA AFN

```
public void Transiciones() {
    ArrayList<String> Terminales = new ArrayList<String>();
    String afd = "digraph G {\n";
    afd += "label = \"AFD DE " + this.nombre + "\"\n";

    String contenido = "digraph G {\n";
    contenido += "label = \"TABLA TRANSICIONES PARA " + this.nombre + "\"\n";
    contenido += "a0 [label=<\n <TABLE cellspacing=\"0\" cellpadding=\"10\">\n"
            + "<TR>\n"
            + "<TD>ESTADO</TD>\n";
    int estado = 0;
    for (int a = 0; a < Nodos.size(); a++) {
        if (!Nodos.get(a).operador) {
            Nodos aux = Nodos.get(a);
            boolean Visto = false;
            for (int b = 0; b < Terminales.size(); b++) {
                if (("" + aux.getLexema()).equals("" + Terminales.get(b))) {
                    Visto = true;
                    break;
                }
            }
            if (!Visto) {
                Terminales.add(aux.getLexema());
            }
        }
    }
    Terminales.remove((Terminales.size() - 1));
    for (int a = 0; a < Terminales.size(); a++) {
        contenido += "<TD>" + Terminales.get(a) + "</TD>\n";
```

Se valua en caso encontrar un estado generado con el mismo ai este se guarda de manera provicional pero no global, de ser un nuevo estado se guarda de manera global. En ambos casos se lanzara un link que una ambos estados o el estado consigo mismo de ser necesario.

```
for (int a = 0; a < ListaEstados.size(); a++) {

    ArrayList<Estados> EstadosAux = new ArrayList<Estados>();
    Estados aux = ListaEstados.get(a);
    contenido += "<TR><TD>" + aux.estado + "</TD>";

    for (int b = 0; b < aux.hojas.size(); b++) {
        String ult = "[ ";
        Nodos hoja = aux.hojas.get(b);

        for (int c = 0; c < hoja.Siguientes.size(); c++) {
            ult += hoja.Siguientes.get(c) + " ,";
        }
        ult = ult.substring(0, ult.length() - 1);
        ult += "]";
        boolean encontrado = false;
        System.out.println("\nHoja: " + hoja.Hoja + "\n");
        for (int c = 0; c < ListaEstados.size(); c++) {
            System.out.println(ult + "              " + ListaEstados.get(c).id);
            if (("" + ListaEstados.get(c).id).equals(ult)) {
                encontrado = true;
                System.out.println("Encontre");
            }
        }
        System.out.println("Sali For");
```

GENERAR DOT

Para ello se tiene el método que recibe todo el código grapvhiz además de la ruta en donde se desea guardar y el nombre de archivo.

```
public void Dot(String contenido, String Clave) {
    ProcessBuilder pbuilder;
    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter("C:\\Users\\justin\\Desktop\\USAC\\2022\\primerSemestre\\COMPI1\\OLC-Pro
        bw.write(contenido);
        bw.close();
        pbuilder = new ProcessBuilder("dot", "-Tpng", "-o", "C:\\Users\\justin\\Desktop\\USAC\\2022\\primerSemestre\\COMPI1\\OLC-Proye
        pbuilder.redirectErrorStream(true);
        pbuilder.start();
    } catch (IOException ex) {
        Logger.getLogger(Interfaz.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```