UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS Y SISTEMAS

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

SECCIÓN “C”

PRIMER SEMESTRE 2023

MANUAL TÉCNICO

Diego Andrés Huite Alvarez

202003585

18/03/2023

Índice

INTRODUCCIÓN

Bienvenido al manual técnico de "Exregan", un software diseñado para generar autómatas finitos deterministas y no deterministas a partir de expresiones regulares en notación polaca y usando el método del árbol. Con Exregan, podrá diseñar y analizar expresiones regulares y generar sus respectivos AFD Y AFND.

REQUERIMIENTOS

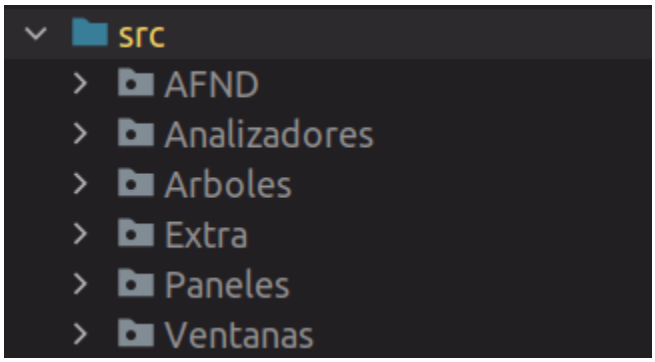
Software:

- Sistema operativo windows, linux o macOS
- Java jdk 8
- Visor de imágenes
- Graphviz instalado
- Jflex y Cup

Hardware:

- Mouse
- Teclado
- Monitor
- 2gb de ram

ESTRUCTURA DEL SOFTWARE



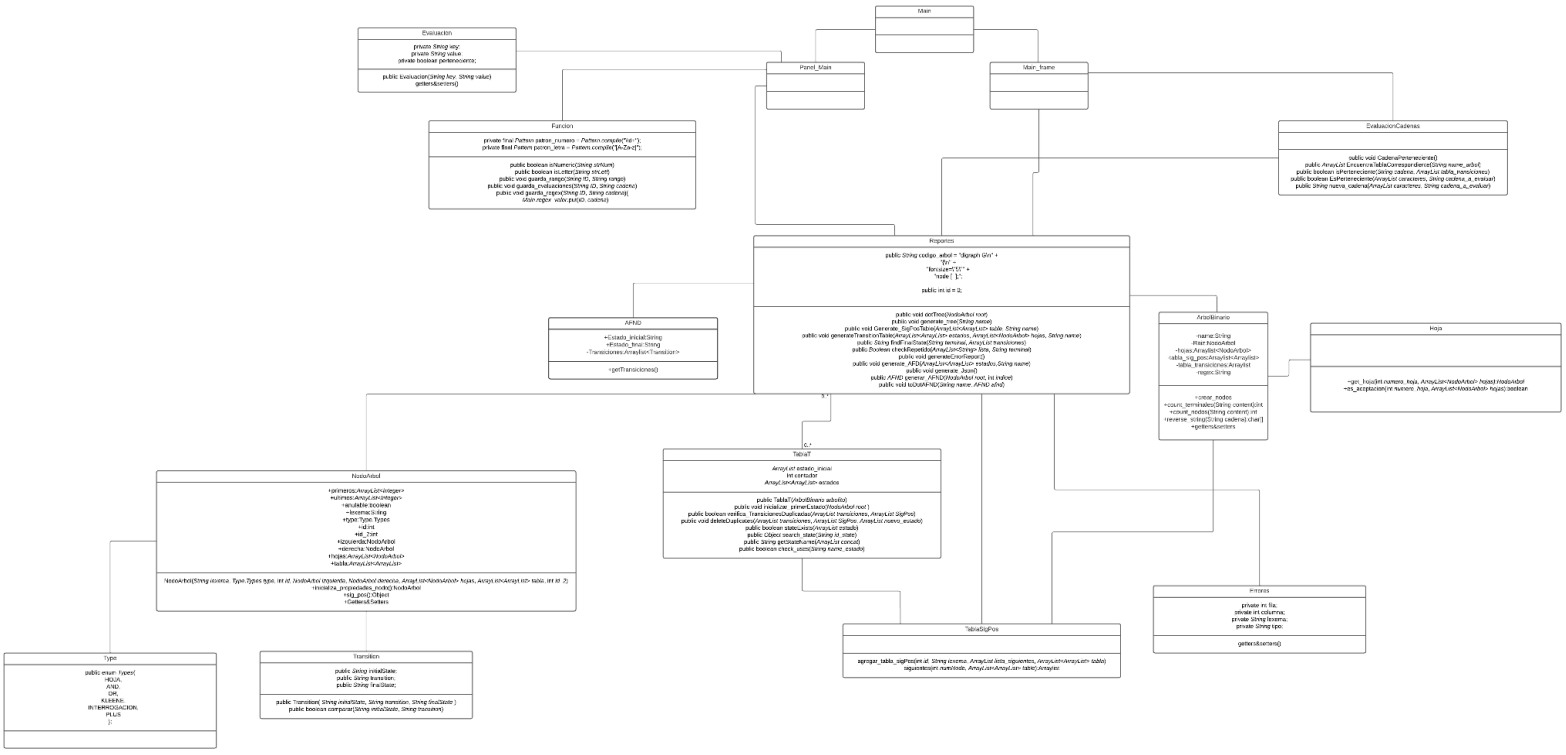
PAQUETES

El software está dividido en los paquetes

- ANFD: paquete que contiene la lógica necesaria para generar los reportes de los autómatas finitos no deterministas.
- ANALIZADORES: Contiene los .java encargados de hacer el análisis léxico y sintáctico posible, así como también se incluyen los .jar usados para generar las clases para los análisis.
- ÁRBOLES: Este paquete contiene todas las clases que son necesarias para hacer el método del árbol, tabla de siguientes y la tabla de transiciones para posteriormente generar los AFD.
- EXTRA: Paquete que contiene la lógica para poder evaluar si una cadena hace match una expresión regular, una clase Error para los errores léxicos y sintácticos, y para poder guardar la información del archivo de entrada como las expresiones regulares, conjuntos y demás.

- PANELES: Este paquete contiene meramente clases que heredan de Jpanel de java swing. Las clases contenidas son enfocadas principalmente a la interfaz gráfica y al comportamiento de esta.
- VENTANAS: Este paquete es el que se encarga de generar la ventana principal del software, en este paquet se incluye el método main.

CLASES



LÓGICA DEL PROGRAMA

El programa recibe un input como este:

```
{
CONJ: letra -> a~z;
CONJ: separados -> 2,a,e,R,d,0,9;
CONJ: mayus-> G~O;
CONJ: nums-> 1~5;
CONJ: symbols-> #~;;

EXPreg0 -> .| "a" "b" \";
EXPreg1 -> . |{letra} "2" * {nums};
EXPreg2 -> .. |{letra} "2" * {nums} . | * |{separados} {mayus} "x" {separados};
EXPreg3 -> .....? \" ||.."x""y""z""w". "z" "y"*||.. "x" "y" "z" "w" . "z" "y" "a" "b" "b"  ;

%%
// pruebas
<! comentario multilinea
asdasdasdasd
asdsad!>

EXPreg1: "a12344445"; // si pertenece
EXPreg1: "a12344445c"; // no pertenece
EXPreg2: "a55559GJJJ0"; // si pertenece
EXPreg0: "b\""; // si pertenece
EXPreg0: "a\""; // si pertenece
EXPreg3: "\"xyzwwwwabb"; // si pertenece
}
```

Declaraciones de conjuntos y expresiones regulares

%%

Evaluaciones de cadenas con expresiones regulares

Para poder analizar el siguiente lenguaje se hizo uso de las siguientes herramientas:

- Análisis léxico: jflex
- Análisis sintáctico: cup

ANÁLISIS LÉXICO

Para poder extraer los tokens del lenguaje anteriormente descrito, se usó java jflex, que a partir de expresiones regulares, genera una lista de tokens al analizar dicha entrada. Algunas de las expresiones regulares para poder tokenizar el lenguaje son las siguientes:

```
LETRA = [A-Za-z]
NUMERO = \d+
ESPECIALES = \\n|\\\'|\\\"
SYMBOL = [!-$]|[&-)]|\/|-|[=->]|@|[\[-\`]
IDENTIFICADOR = [a-zA-Z][a-zA-Z0-9_]*
FLECHA = -\s*>
```

```
<YYINITIAL> {SYMBOL} {return new Symbol(sym.symbol, yyline, yycolumn, yytext());}
```

Todas las expresiones regulares que se encuentran en el archivo .jflex por lo que por cuestiones de limpieza del documento, queda a discreción del lector el poder revisarlo en el repositorio.

La clase de java que permite esto fue creada con Jflex, se llama “Lexico.java” y se encuentra en el java package de “Analizadores”

ANÁLISIS SINTÁCTICO

Para la elaboración del análisis sintáctico se hizo uso de java cup. La gramática para poder hacer el proyecto fue la siguiente:

```
INICIO ::= INSTRUCCIONES;

INSTRUCCIONES ::= "{"
                  LISTA_SENTENCIAS
                  "%" "%"
                  LISTA_SENTENCIAS_EVAL
                  "}"


// NO TERMINALES PARA REALIZAR UNA EVALUACION ----------------------------------------------------------------------------

EVALUACION ::= id ":" cadena ";"

LISTA_SENTENCIAS::= SENTENCIA LISTA_SENTENCIAS| SENTENCIA;
LISTA_SENTENCIAS_EVAL ::= EVALUACION LISTA_SENTENCIAS_EVAL|EVALUACION;

SENTENCIA ::= CONJUNTOS|REGEX ";"


// NO TERMINALES DE CONJUNTOS--------------------------------------------------------------------------------------------
CONJUNTOS ::= "CONJ" ":" id "->" RANGO




RANGO ::= CARACTER CARACTER2



CARACTER::= letra
           |numero
           |symbol
           |"{"
           |"}"
           |":"
           |";"
           |"%"
           |"."
           |","
           |"*"


CARACTER2::= "~" CARACTER ";"

LISTA ::= "," CARACTER LISTA



// NO TERMINALES DE REGEX------------------------------------------------------------------------------------------------

REGEX ::= id "->" REG_EXP

REG_EXP::= "." REG_EXP REG_EXP
         | "|" REG_EXP REG_EXP
         | "*" REG_EXP
         | "+" REG_EXP
         | "?" REG_EXP
         | ELEMENTO_REGEX
         ;


ELEMENTO_REGEX ::= "{" id "}"
```

En donde se toma en consideración la notación polaca de las expresiones regulares.

FLUJO DEL PROGRAMA

El programa cuando el botón “analizar” es presionado, va hacia las clases del paquete Árboles para generar el árbol, tabla de siguientes y tabla de transiciones, para luego poder elaborar los autómatas y por último las evaluaciones de cadenas.