Universidad de San Carlos de Guatemala
Laboratorio de OLC1
Sección B
Kenneth Haroldo López López
201906570

REGEXIVE

MANUAL TÉCNICO

Guatemala, 10 de marzo de 2021

ÍNDICE

Contenido

ESPECIFICACIONES DEL ENTORNO DE DESARROLLO

Lenguaje Utilizado:

- Java

IDE Utilizado:

- Apache NetBeans IDE 11.3

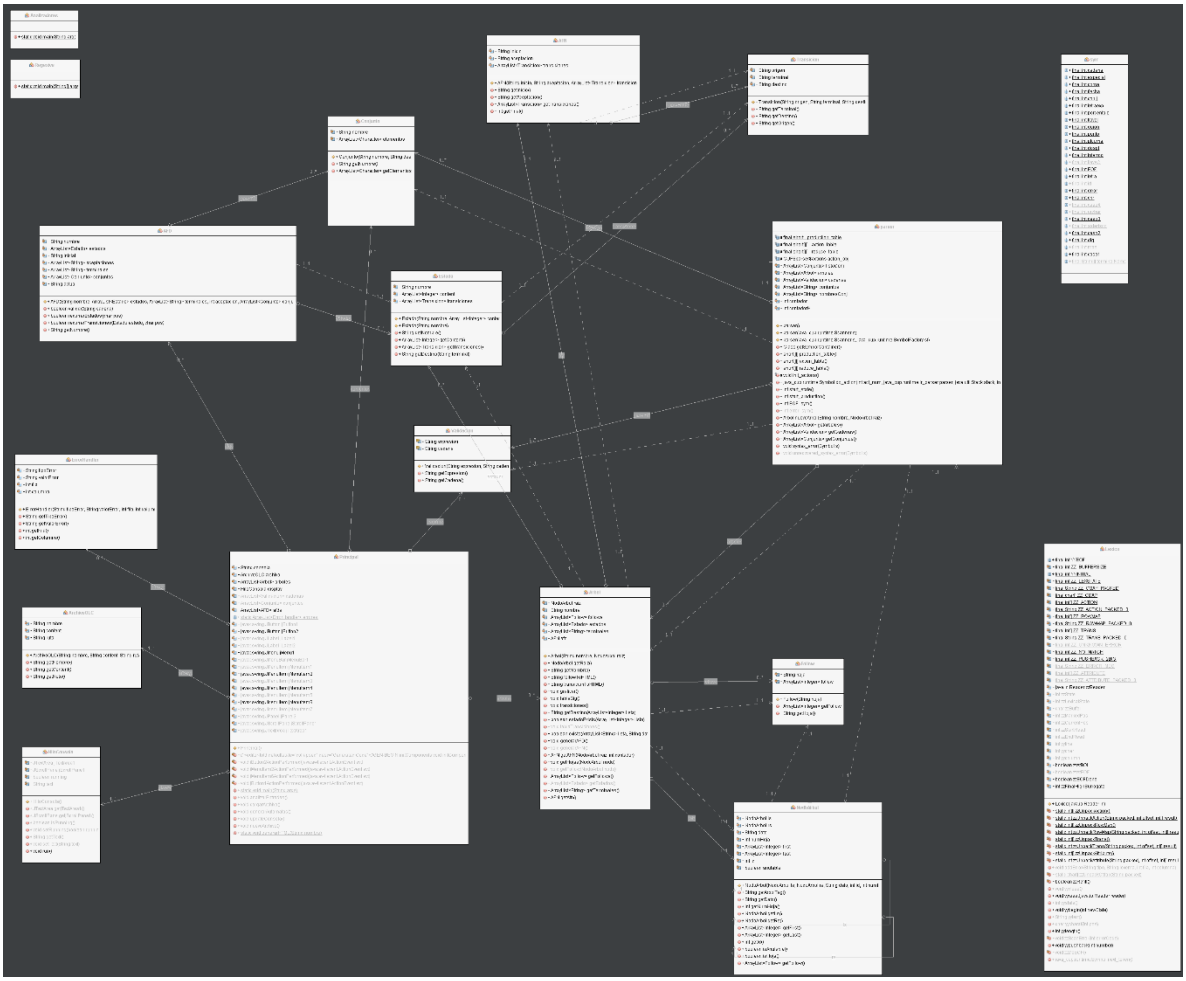
Sistema Operativo:

- Windows 10 Home

Librerías Externas:

- Java CUP 11b
- Java Jflex 1.7.0

DIAGRAMA DE CLASES



CARGA DE ARCHIVOS

Estructura del archivo olc

La extensión única aceptada para la carga de archivos es olc. Este será un archivo de texto plano que contará con la información necesaria para el análisis de cadenas y generación de reportes de salida.

```
{

CONJ: mayus - > A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z;
CONJ: minus -     > a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z;
CONJ: letra -  > a~z;
CONJ: digito -  > 0~9;
CONJ: posibles -   > P, C, O;

EXP5  -   > . \' . + | | | | \n {minus} {mayus} {digito} " " \';
P1 -   >  . {posibles} . ? "-"  . {digito} . {digito} . {digito}  . {letra}  . {letra} {letra};
%%
%%
EXP5 : "\'cadena entre comilla simple\'";
P1 : "P-385fde"; // bueno
P1 : "P388fdc"; // bueno
P1 : "P38df8c"; // malo
P1 : "P-38df8k";  // malo
P1 : "K-385ffk";  // malo
}
```

Consta de 2 secciones, la sección de definiciones y la sección de validaciones. En la sección de definiciones se encontrará la información acerca de los conjuntos y expresiones a utilizar en el programa. Y la sección de validaciones, en donde se encontrarán las cadenas a evaluar en el programa, utilizando las expresiones y conjuntos previamente definidos.

GENERACIÓN DE ARCHIVOS

Por cada expresión regular definida en el archivo de entrada se deberá crear una serie de reportes en forma de imágenes. Estos son:

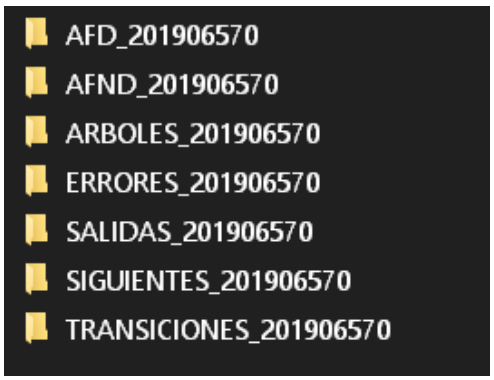
- Árbol
- Tabla de siguientes
- Tabla de transiciones
- AFD
- AFND

Cada reporte es generado desde que el usuario elige la opción de generar autómatas de la vista principal. El contenido del archivo será procesado mediante las librerías de Jflex y CUP; las cuales se encargarán de verificar que la estructura del archivo de entrada sea la correcta y así evitar errores durante el tiempo de ejecución.

Si el archivo contiene errores se deberá crear un archivo HTML con la tabla de errores correspondiente.

Una vez el usuario haya generado los autómatas correspondientes al archivo de entrada, se podrá utilizar la opción de analizar entradas. En la cual se utilizarán los autómatas previamente generados para analizar la validez de las cadenas. Los resultados de las validaciones serán mostrados en pantalla mediante la consola de salida del GUI, así como en un archivo json.

Cada reporte debe ser almacenado en el sistema en un directorio específico para cada tipo de reporte.



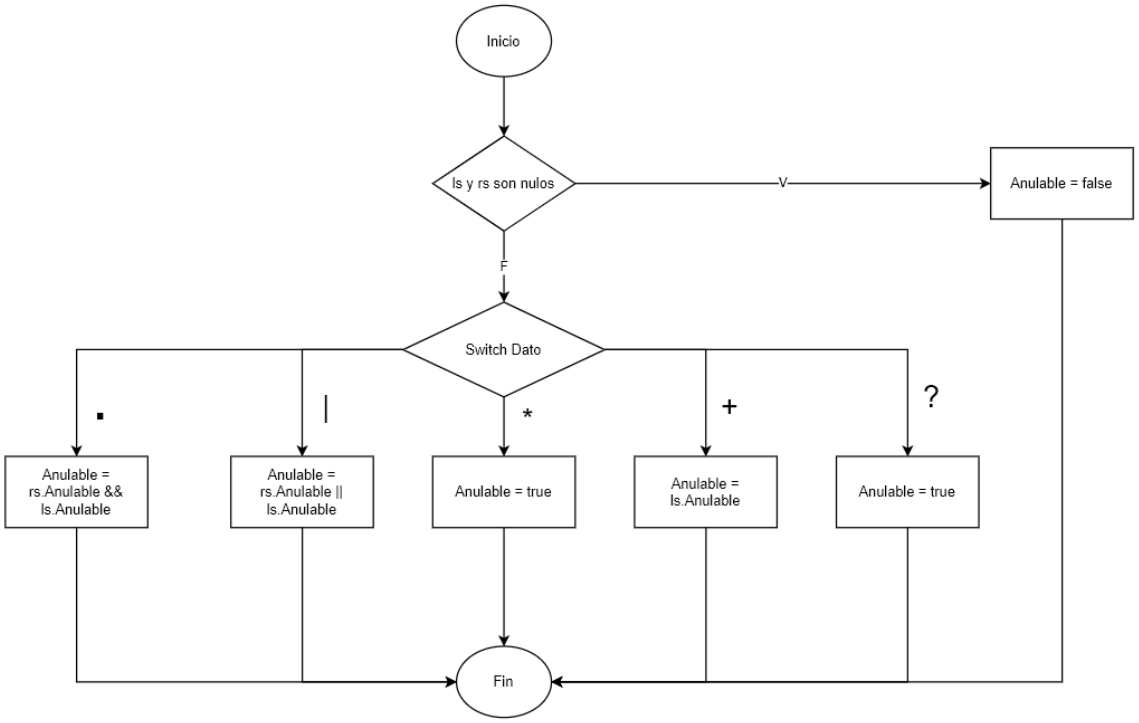
Generación de árbol asociado

La creación del árbol se realiza mediante una función recursiva apoyándose del archivo de CUP, en este las derivaciones provocadas por las expresiones generarán un nuevo nodo que será insertado en el árbol. El árbol será generado de “abajo

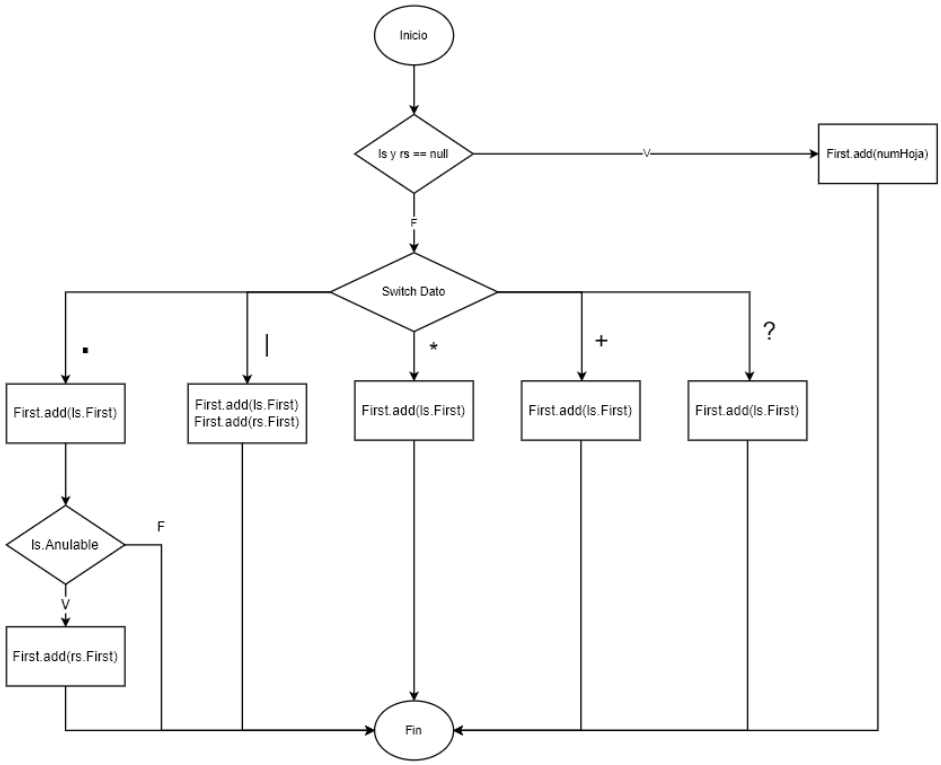
hacia arriba” creando los nodos hoja primero y ascendiendo en la profundidad del árbol hasta la raíz. Cada nodo será un objeto perteneciente a la clase “NodoArbol”.

Donde ls y rs pertenecerán al resultado de una llamada recursiva del método de generación de árbol (de tratarse de nodos hoja este será nulo); dato será el valor de String que ese nodo albergará en el árbol; id será el número de nodo creado (se deberá mantener un contador externo que crezca a medida que se creen más nodos); numHoja será el número de hoja del nodo (si el nodo no es una hoja no se utilizará este atributo; se necesita de un contador externo para mantener el control del número de hojas creadas); first será un ArrayList de enteros que almacenará los first del nodo siguiendo las reglas del método del árbol. Igual es el caso para deducir los last y si el nodo es anulable o no.

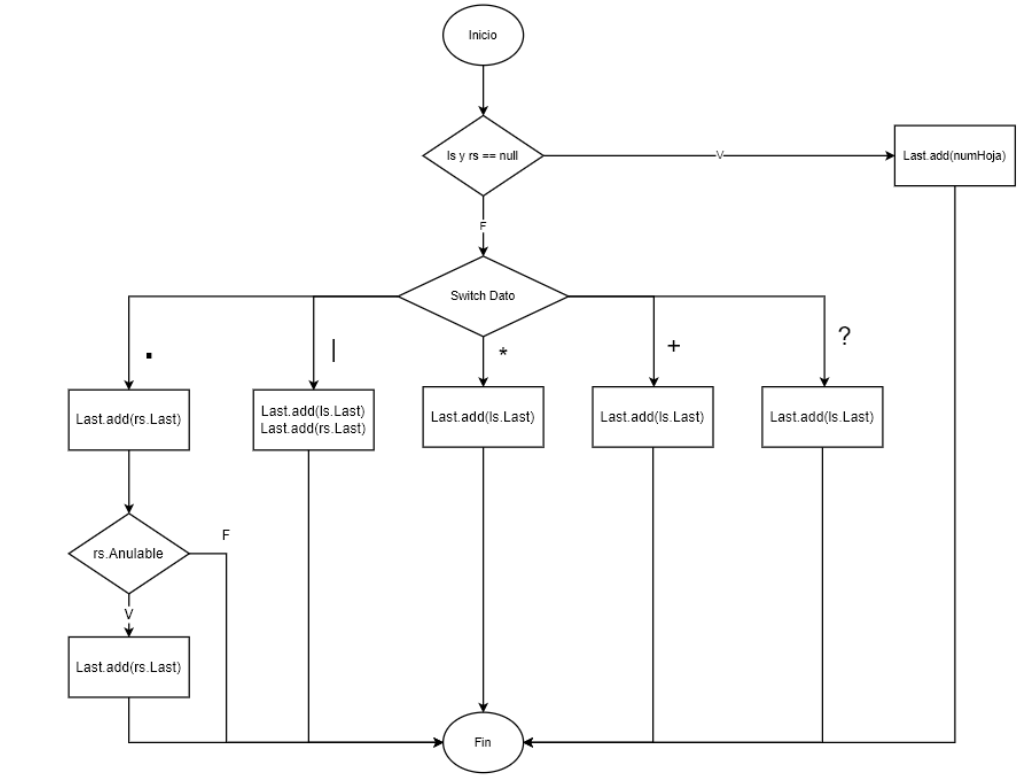
Determinación de atributo anulable:



Determinación de First:



Determinación de Last:



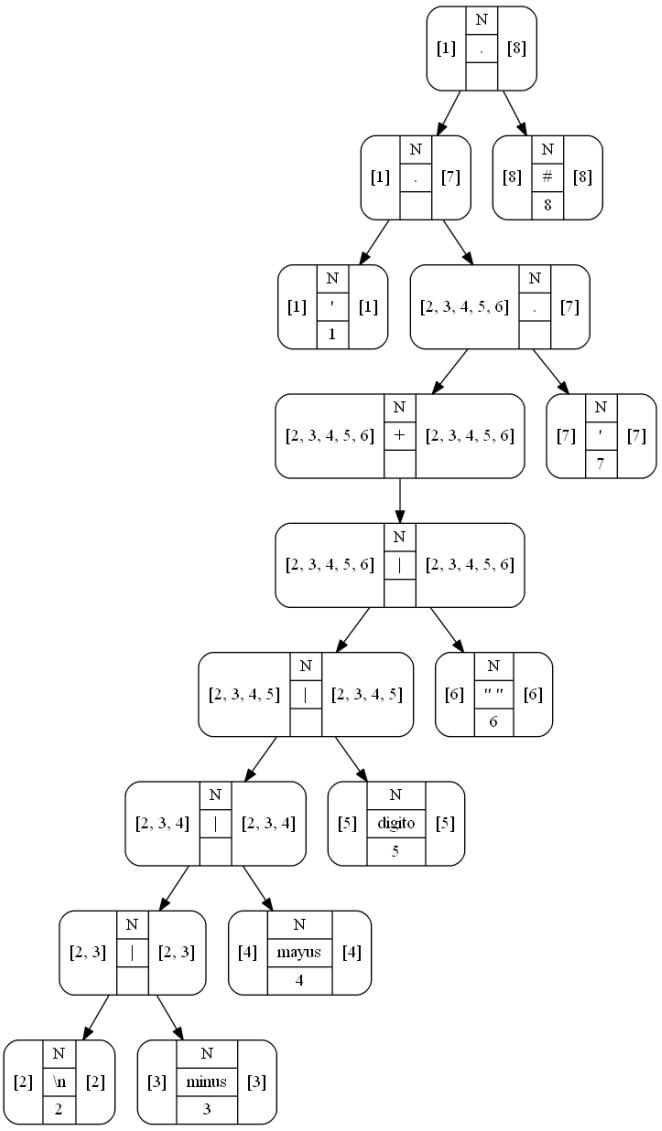
Graficar el árbol:

El método getDotTag() de la clase NodoArbol retorna un String que contiene la estructura necesaria para mostrar los datos del nodo de manera gráfica en el lenguaje de graphviz. Este tendría la siguiente estructura:

```
Nodo( ID ) [ label = " { FIRST } | { ANULABLE | DATO | NUMHOJA } | { LAST } " ] ;
```

En caso de que el nodo no se trate de una hoja (ambos hijos son nulos) se omite la parte de numHoja. El método dotTag() es recursivo y genera las conexiones necesarias para unir nodos padre con hijos; además es recursivo, por lo que basta con realizar una llamada desde el nodo raíz para obtener el String completo del archivo en dot que al compilarse generará la vista del árbol.

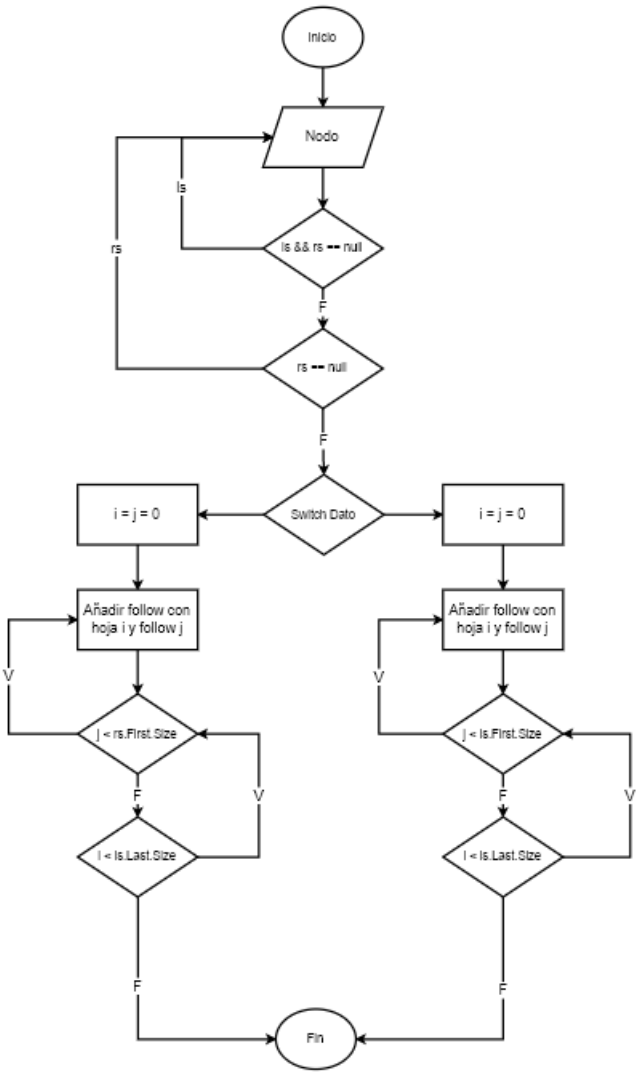
Ejemplo:



Generar tabla de siguientes

Se hace uso de la clase Follow para almacenar los datos de los siguientes del árbol, en el cual el atributo “hoja” corresponde al dato de la hoja a la que se le analiza el siguiente, mientras el ArrayList de int almacenará la lista de siguientes calculados.

Cálculo de siguientes:



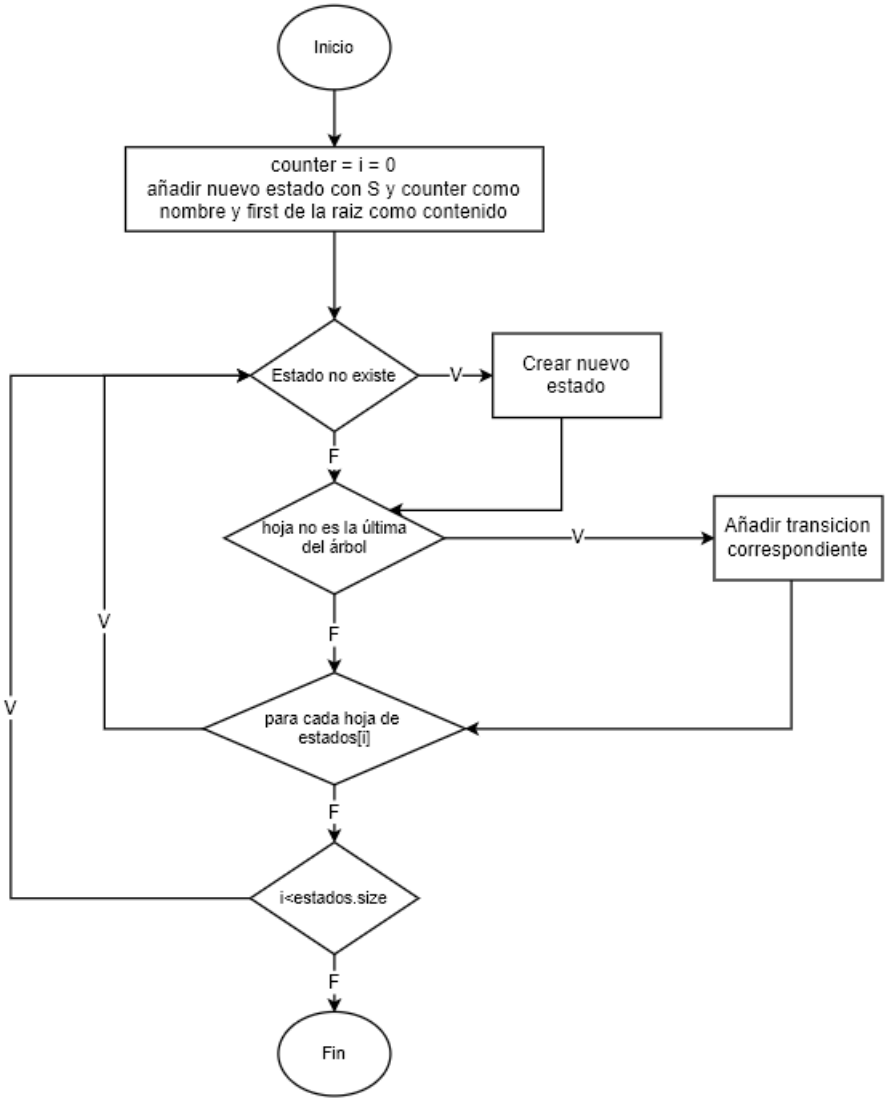
Posteriormente se genera con graphviz una tabla con la información de cada Follow. Los follows se registran en orden por lo que cada posición de la lista también es el número de hoja del dato.

Ejemplo:

Hoja		Siguientes
posibles	1	[2, 3]
-	2	[3]
digito	3	[4]
digito	4	[5]
digito	5	[6]
letra	6	[7]
letra	7	[8]
letra	8	[9]
#	9	[]

Generar tabla de transiciones:

Se hará uso de la tabla de siguientes para generar objetos Estado, estos cuentan con un nombre, una lista de las hojas que lo conforman y de sus transiciones (con objetos Transición, siendo el destino el nuevo Estado creado si existe o crear uno nuevo), generando transiciones para con cada hoja si esta se encuentra en el contenido del estado. Se empieza creando un estado inicial con contenido igual a la lista de first del nodo raíz. Se utiliza un contador para nombrar cada estado nuevo creado.



Con la información de los estados en la lista de estados se crea la tabla de transiciones.

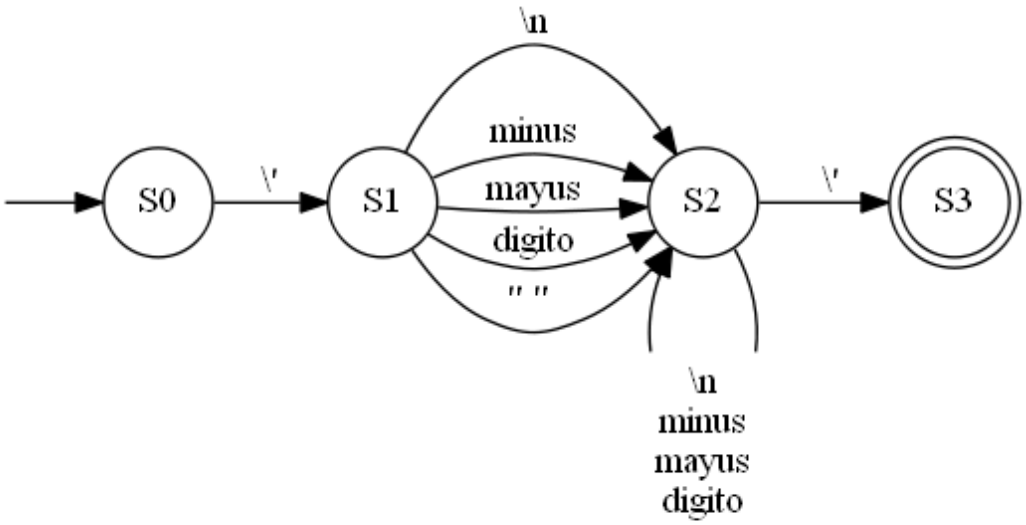
Ejemplo:

Estado	Terminales					
	\'	\n	minus	mayus	digito	" "
S0[1]	S1					
S1[2, 3, 4, 5, 6]		S2	S2	S2	S2	S2
S2[2, 3, 4, 5, 6, 7]	S3	S2	S2	S2	S2	S2
S3[8]{ACEPTACION}						

Generación del AFD

Con la tabla de transiciones generada se procede a llamar a la función de generarAFD() del árbol, la cual se encarga de crear en un archivo dot todos los nodos pertenecientes a los estados y realizar las conexiones necesarias dependiendo de sus transiciones y los caracteres de transición. Posteriormente compila el archivo generado en una imagen.

Ejemplo:



Generación de AFND

Utilizando la función recursiva de getAFN se le enviará como parámetro inicial el hijo izquierdo de la raíz y un número 0 (este servirá como contador cada vez que se cree un estado nuevo), pues en este caso no nos interesa la hoja de aceptación del árbol ya que el AFND creado solo poseerá un estado de aceptación. El método se encargará de crear objetos AFN y retornarlos para que puedan ser usados por la llamada anterior al método. Siguiendo las reglas del método de Thomson.

Un objeto AFN cuenta con un listado de transiciones de tipo Transición, un estado final y un estado inicial. Cada vez que se llame el método de manera recursiva enviará el nodo que se quiere analizar y el número de estado siguiente en la creación del AFN actual.

Luego se procede a llamar la función generarAFN para que proceda a compilar una imagen del AFND creado como se hizo con el AFD.

Ejemplo:

Manejo de Errores

Si existe algún error en la entrada, la generación de reportes y autómatas es cancelada y se procede a adjuntar los errores encontrados en una tabla generada en HTML.

Ejemplo:

Reporte de Errores

TIPO	DESCRIPCION	FILA	COLUMNA
Error Sintáctico (Recuperado)	No se esperaba este componente: -.	10	4

Validación de Cadenas

Todas las cadenas para validar provenientes del segundo bloque del archivo de entrada son almacenadas en objetos de la clase Validación, en donde se almacena la cadena a validar y la expresión regular a utilizar para validarla. La validación se activa cuando el usuario hace click en el botón de la función y ya se han generado los autómatas correspondientes.

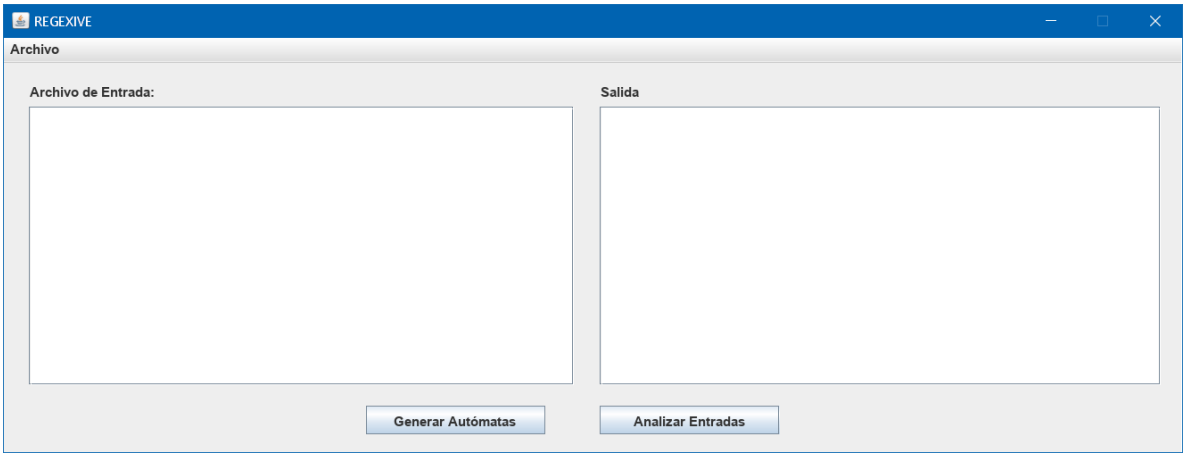
Se hace uso del autómata previamente generado accediendo a la lista de árboles generados en la pestaña Principal, se utiliza el árbol con el nombre que coincida con la gramática del objeto Validacion. Se procede a leer carácter por carácter de la cadena a validar y se realiza un cambio de estado si existe una transición con el carácter analizado, de no existir ninguna transición en el autómata; no validará la cadena. Y si el estado del autómata al final del análisis no es un estado de aceptación (no contiene la última hoja del árbol), también tomará como no válida la cadena.

Se deberá generar un archivo json con un único arreglo que posea los datos de todas las cadenas analizadas y su conclusión después de su análisis.

Ejemplo:

```
[
  {
    "Valor":"\\'cadena entre comilla simple\\'",
    "ExpresionRegular":"EXP5",
    "Resultado":"Cadena Valida"
  },
  {
    "Valor":"P-385fde",
    "ExpresionRegular":"P1",
    "Resultado":"Cadena Valida"
  },
  {
    "Valor":"P388fdc",
    "ExpresionRegular":"P1",
    "Resultado":"Cadena Valida"
  },
  {
    "Valor":"P38df8c",
    "ExpresionRegular":"P1",
    "Resultado":"Cadena No Valida"
  },
  {
    "Valor":"P-38df8k",
    "ExpresionRegular":"P1",
    "Resultado":"Cadena No Valida"
  },
  {
    "Valor":"K-385ffk",
    "ExpresionRegular":"P1",
    "Resultado":"Cadena No Valida"
  }
]
```

INTERFAZ GRÁFICA UTILIZADA



En el espacio de archivo de entrada el usuario podrá escribir el archivo manualmente o se mostrará el contenido de un archivo que este desee cargar desde la respectiva opción del menú archivo.

La salida es un espacio no accesible para el usuario en el cual se le notificará de las operaciones realizadas por el sistema. Los botones activarán sus respectivas operaciones. Generar autómatas, analizará el archivo y creará imágenes mientras Analizar Entradas generará los json de salida y verificará la validez de las cadenas del archivo de entrada (Ver generación de archivos pág. 6).