Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Primer Semestre 2023
Catedráticos
Ing. Mario Bautista
Ing.Manuel Castillo
Ing.Kevin Lajpop
Tutores Académicos
Mynor Ruiz
Josè Pérez
Kevin López
Maynor Piló Tuy

EXREGAN

Proyecto 1

Contenido

1. Objetivos

1.1 Objetivos Generales

Aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para la construcción de una solución de software que permita generar análisis por medio del método del árbol.

1.2 Objetivos Específicos

- Reforzar el concepto del método de Árbol de expresiones regulares en Autómatas Finitos Deterministas (AFD).
- Reforzar el concepto del método de Thompson de expresiones regulares en Autómatas Finitos No Deterministas (AFND).
- Identificar y programar el proceso de reconocimiento de lexemas mediante el uso de Autómatas Finitos Determinista.

2. Descripción General

El curso de Organización de Lenguajes y Compiladores 1, perteneciente a la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, requiere que usted, como conocedor en la construcción de analizadores Léxico y Sintáctico, cree un sistema que sea capaz de realizar el Método del Árbol y el Método de Thompson, para que los estudiantes puedan verificar que las respuestas de las tareas y exámenes del curso son correctas. Este aplicativo, requiere de las funciones que se abordan en la **Sección 4,** y requiere que las expresiones regulares sean ingresadas en notación polaca o prefija.

Cabe destacar que el sistema debe contar con las siguientes funcionalidades principales:

- Intérprete de Expresiones Regulares Permitidas: Esta funcionalidad será la encargada de analizar un archivo de expresiones regulares que definirán el patrón que se utilizará dentro del sistema para que reciba los lexemas, se entiende que con la información de la expresión regular se tendrá lo necesario para la validación de los lexemas.

3. Funcionalidades

Se solicita un sistema que permita la edición de archivos fuente a través de un área de texto. Además, que permita la visualización de los resultados y reportes generados por la fase de análisis.

3.1. Menú Archivo

- Nuevo Archivo: Se podrá crear un nuevo archivo con extensión [.olc].
- Abrir archivo: Nos permitirá abrir un archivo que esté guardado en disco. Los archivos a abrir tendrán extensión [.olc].
- Guardar: Nos permitirá guardar el archivo actual.
- Guardar como: Nos permitirá guardar el archivo con otro nombre.
- Generar Autómatas: Analizará el archivo en base al área de expresiones regulares, generando sus respectivos reportes.
- Analizar cadenas y generar JSON de salida: Analizará el archivo en base a las cadenas a evaluar con las expresiones regulares creadas anteriormente, indicando si es correcta o incorrecta su evaluación y guardará el archivo en una ruta definida por el usuario.

3.2. Comentarios

El lenguaje OLC permitirá el uso de comentarios de una o varias líneas. Para comentarios multilínea se utilizará el símbolo **<!** para indicar el inicio del comentario y el símbolo **!>** para indicar el final del comentario. Para comentarios de una sola línea se utilizará la doble barra ***//***.

```
<! Este es un comentario
   en nuestro programa
!>
// Este es un comentario en nuestro programa
```

4. Intérprete de Expresiones Regulares

Esta función permitirá realizar un análisis léxico y sintáctico para los archivos con lenguaje **OLC** (.olc) dentro del cual se definen todas las expresiones regulares que se van a lograr reconocer en el lenguaje fuente**.**

4.1 Definición del Lenguaje

El archivo deberá de contener la definición de las expresiones regulares se compone de una sección en la que cada sentencia define el token (identificador) con que el analizador debe reconocer los lexemas ingresados, seguido de la definición de la expresión regular. Seguido de esto dos símbolos de porcentaje (%%). Por último, se define un identificador y un lexema de entrada, el cual se deberá comparar con la definición que se encuentra en la parte superior.

Cada sentencia se delimita utilizando punto y coma.

```
{
////// CONJUNTOS
CONJ: nombre_conjunto -> notacion; //la notación se define en la sección 4.1.2
CONJ: nombre_conjunto -> notacion;
tId -> Expresión_regular_en_prefijo;
tId -> Expresión_regular_en_prefijo;
// Mas sentencias
%%
%%
tId: "Lexema de entrada";
tId : "Otro Lexema";
// Mas sentencias
}
```

4.1.1 Expresiones regulares

Las expresiones regulares establecen el patrón que se debe cumplir para representar un token, estas se reconocerán en notación polaca o prefija. A continuación, se muestra la notación a utilizar.

Notación	Definición
. a b	Concatenación entre a y b
| a b	Disyunción entre a y b
* a	0 o más veces
+ a	1 o más veces
? a	0 o una vez

* **a** & **b** pueden ser caracteres, conjuntos designados con anterioridad o caracteres especiales.

4.1.2 Conjuntos

Para la definición de conjuntos se utiliza la palabra reservada "CONJ". Un conjunto puede utilizarse dentro de una expresión regular, pero no en la definición de otro conjunto.

A continuación, la notación a utilizar para la definición de conjuntos:

Notación	Definición
a~c	Conjunto {a, b, c}.
a~z	Conjunto de la a hasta la z en minúsculas.
A~Z	Conjunto de la A hasta la Z en mayúsculas.
0~7	Conjunto del 0 al 7.
0,2,4,6,8	Conjunto {0, 2, 4, 6, 8}
A,b,C,d	Conjunto {A, b, C, d}
!~&	Conjunto de signos entre ! (33 en código ascii) y & (38 en código ascii). Nota: el rango valido será desde el ascii 32 hasta 125 omitiendo los ascii de las letras y dígitos.

Los conjuntos vistos anteriormente son ejemplos de las diferentes variantes que éstos pueden tomar (Ej: también puede existir a~d o 0~9).

4.1.3 Caracteres Especiales

Dentro del lenguaje pueden utilizarse estos caracteres especiales:

Notación	Definición
\n	Salto de línea
\’	Comilla Simple
\”	Comilla Doble

4.1.4 Ejemplo

Notas:

- La definición de conjuntos CONJ puede existir en cualquier parte del archivo.
- El uso de conjuntos se verá delimitado por **{** llaves **}**

```
{
////// CONJUNTOS

CONJ: letra -> a~z;
CONJ: digito -> 0~9;

/////// EXPRESIONES REGULARES

ExpReg1 -> . {letra} * | "_" | {letra} {digito};
ExpresionReg2 -> . {digito} . "." + {digito};
RegEx3 -> . {digito} * | "_" | {letra} {digito};

%%
%%

ExpReg1 : “primerLexemaCokoa”;
ExpresionReg2 : “34.44”;
}
```

En este caso ***“primerLexemaCokoa”*** se debe comparar con: {***letra} * | "_" | {letra} {dígito}*** para determinar si la entrada cumple con la expresión regular, de ser correcta se escribirá en el archivo JSON

4.1.5 Generación de JSON de Salida

Una vez que se analicen cada una de las líneas debajo del doble porcentaje (%%) se deberá generar un archivo JSON cuya ruta será definida más adelante, el cual tendrá las siguientes características:

```
[
   {
      "Valor":"primerLexemaCokoa",
      "ExpresionRegular":"ExpReg1",
      "Resultado": "Cadena Válida"
   },
   {
      "Valor":"34.44",
      "ExpresionRegular":"ExpReg2",
      "Resultado": "Cadena Válida"
   }
]
```

Archivo de Salida esperada

5. Flujo del programa

5.1 Generación de AFD

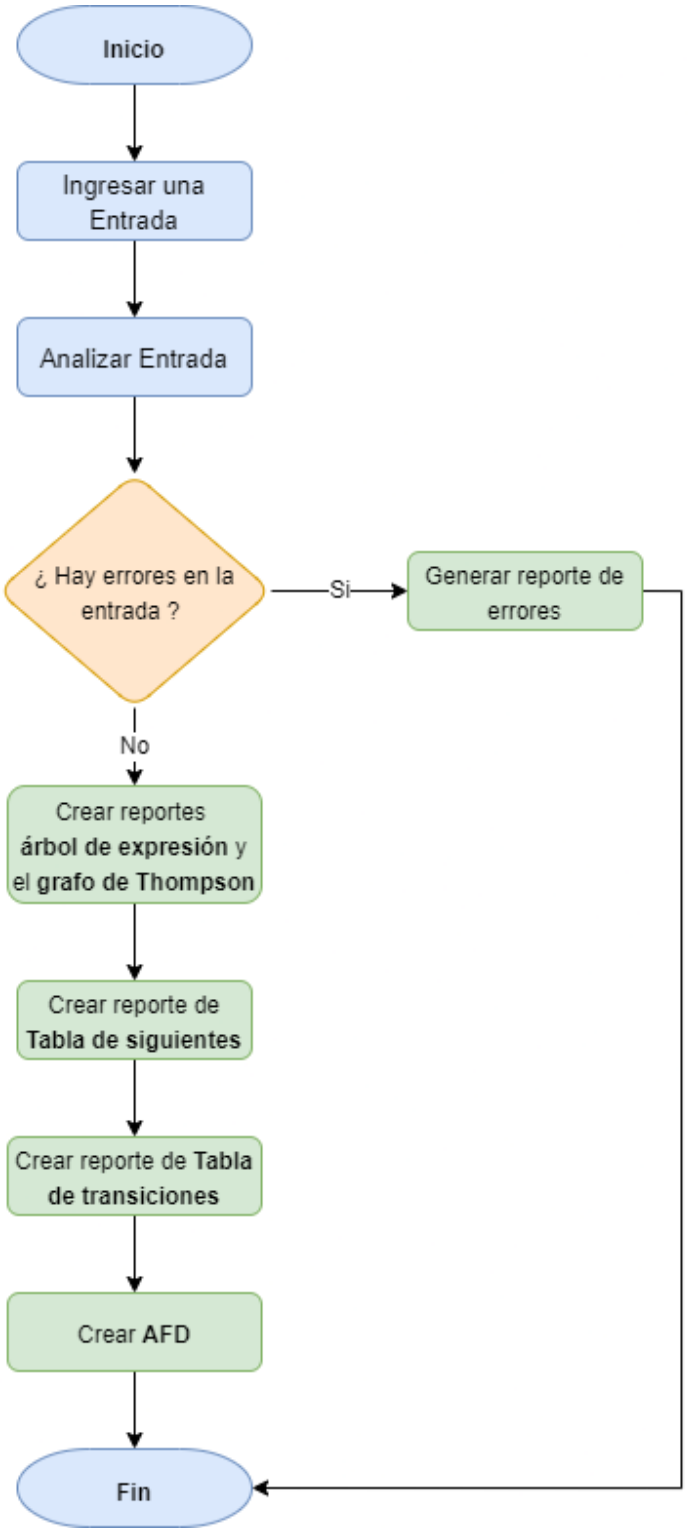


Diagrama de flujo para la creación de autómatas finitos determinista

5.2 Validación de cadenas

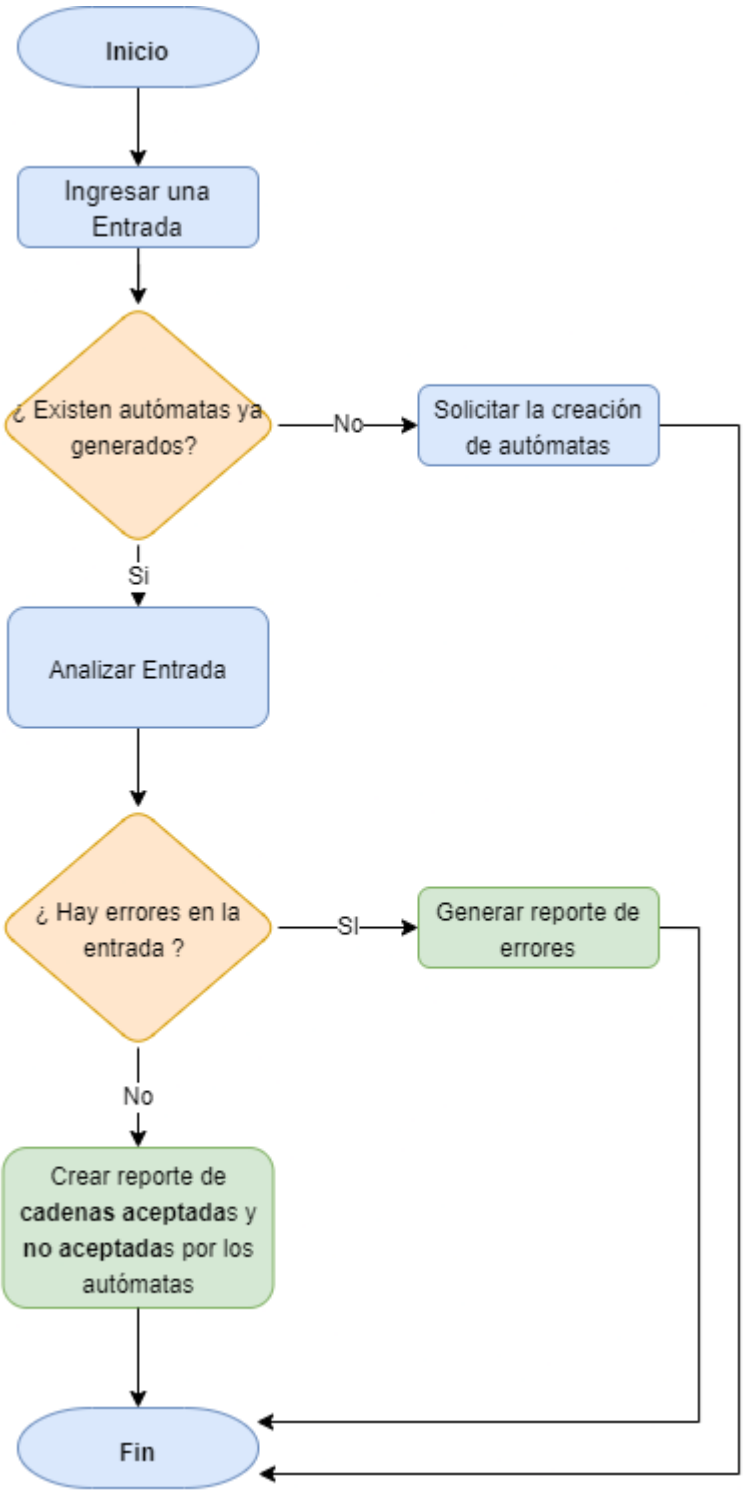
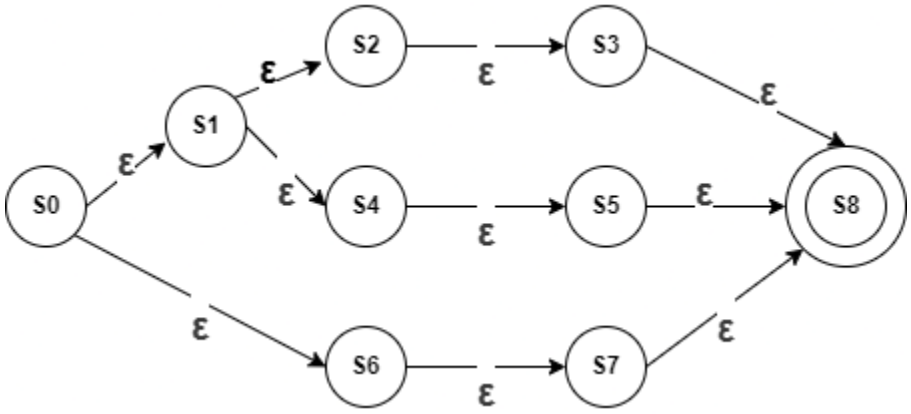


Diagrama de flujo para validación de cadenas

6. Método de Thompson

6.1 Autómata Finito No Determinista

Es un autómata finito, que a diferencia del AFD, este contiene transiciones vacías, esto quiere decir que para un símbolo hay más de una transición posible.



Ejemplo de un AFND creado con el método de Thompson

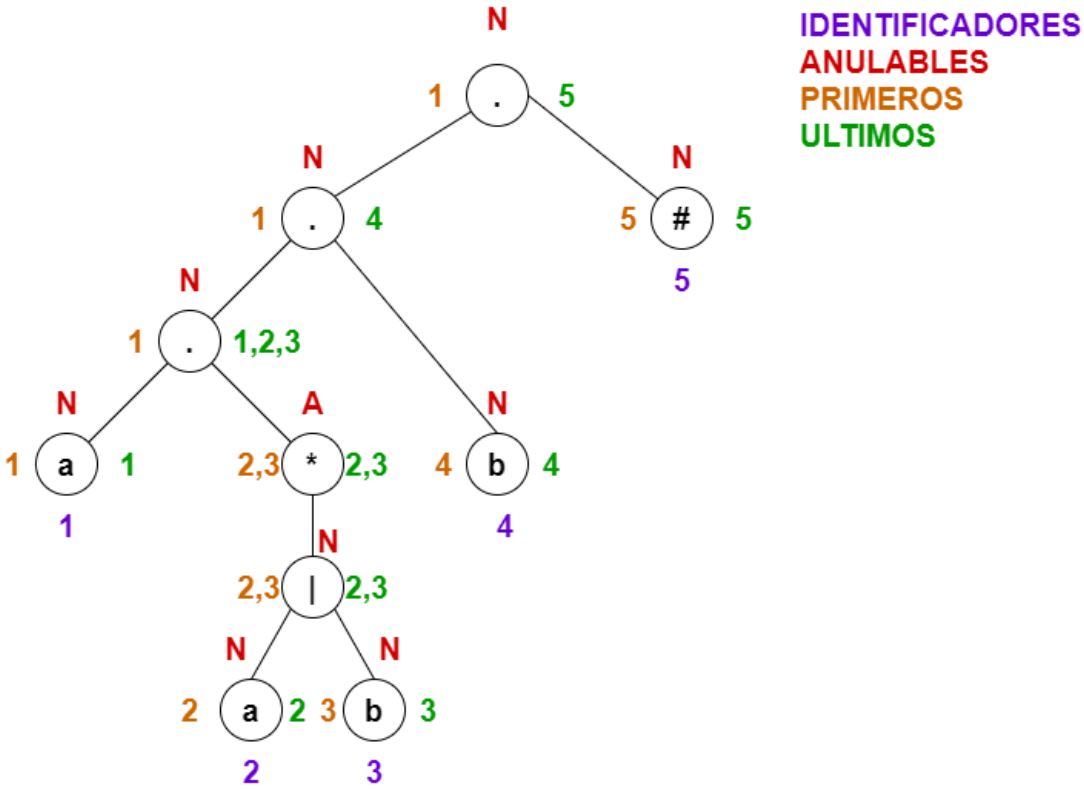
Para aplicar el método de Thompson, se tienen un conjunto de reglas para cada tipo de transición. **Ver Anexo**

7. Método del árbol

7.1 Árbol de expresión

Cada expresión regular descrita con anterioridad en el archivo de entrada deberá generar un árbol de expresión igual al que se muestra a continuación. Cada hoja del árbol tendrá los siguientes atributos dentro de la gráfica.

- **Identificador**
- **Anulable | No Anulable**
- **Primeros**
- **Últimos**



Árbol de expresión

7.2 Tabla de transición y tabla de siguientes

Con el árbol de expresión generado con anterioridad se deberá generar una tabla de transición y una tabla de siguientes correspondientes.

Tabla de siguientes

	Hoja	Siguientes
A	1	2,3,4
A	2	2,3,4
B	3	2,3,4
B	4	5
#	5	--

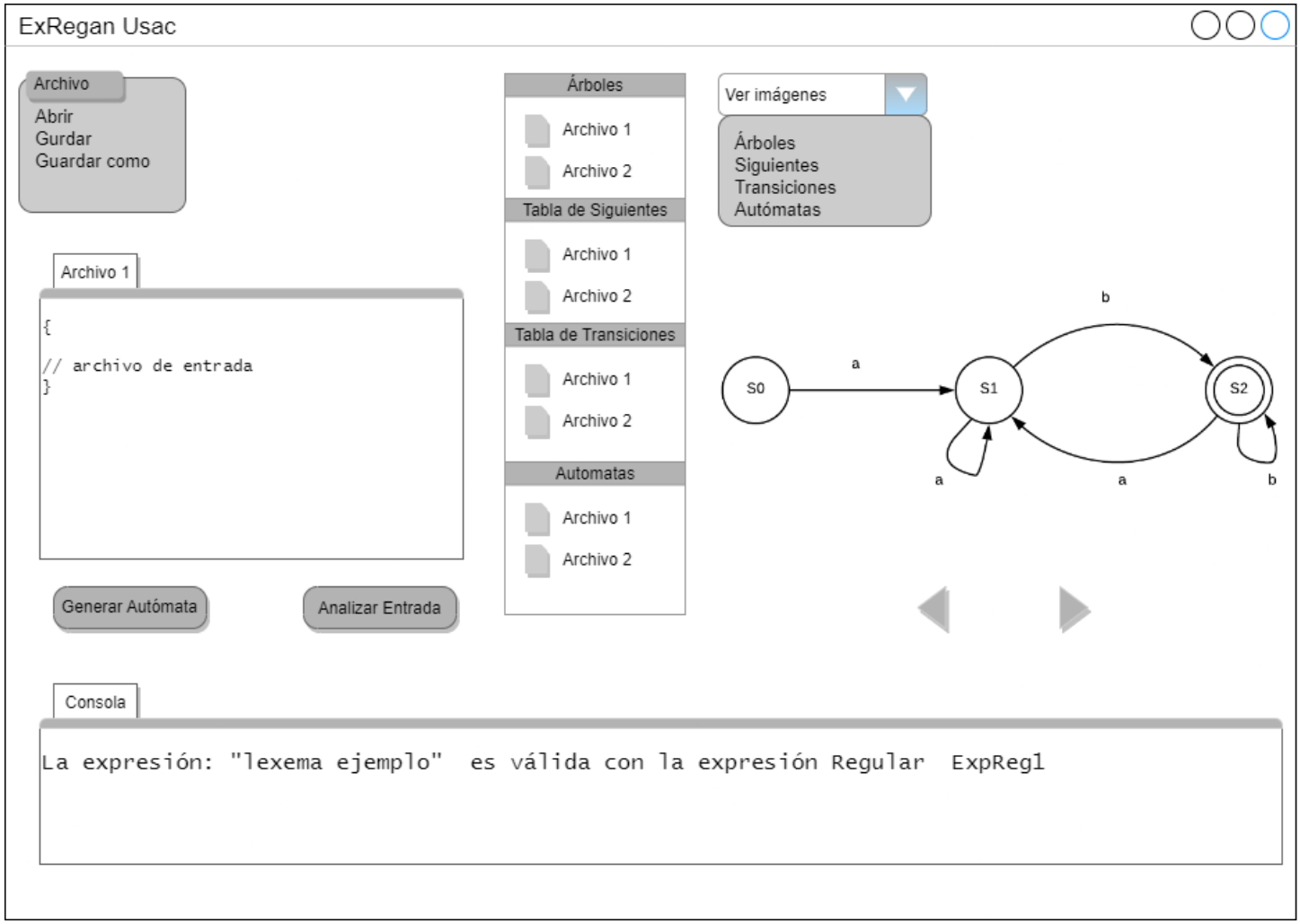
Tabla de Transiciones

Estado	**a Terminales b**	
S0 {1}	S1	--
S1 {2,3,4}	S1	S2
S2 {2,3,4,5}	S1	S2

8. Interfaz Sugerida

Consola: la consola deberá mostrar el siguiente mensaje:

La expresión: ***<lexema de entrada>*** *es válida con la expresión Regular* ***<Nombre de la Expresión Regular>***



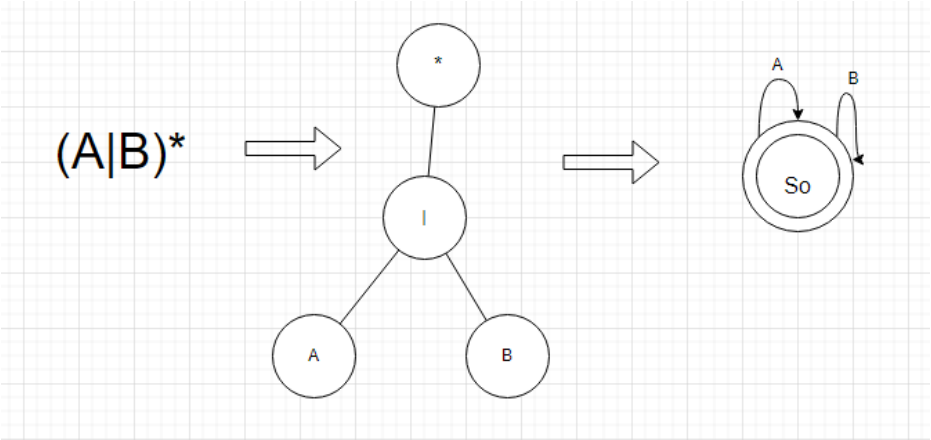
9. Anexos

9.1 Explicación sobre Método del Árbol

Sirve para encontrar un AFD mínimo dada una expresión regular. Las características de un AFD son:

- No tiene transiciones Epsilon
- No tiene dos o más transiciones con el mismo símbolo a un mismo estado

En el ejemplo se puede observar la entrada, y la salida esperada del método del árbol.

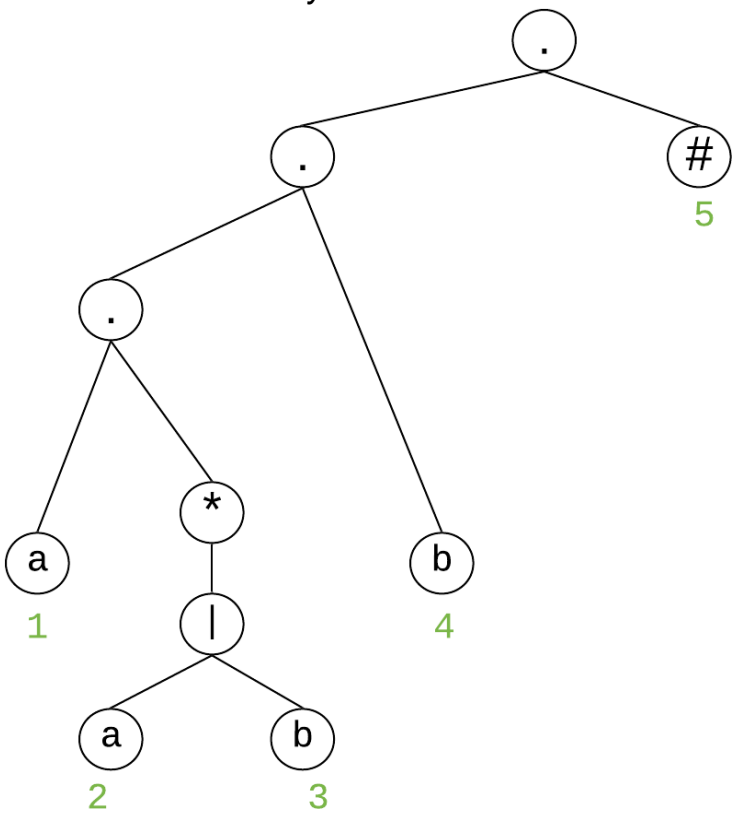


8.1.1 Pasos a seguir para realizar el método del árbol

8.1.1.1 Agregar al final de la expresión regular el símbolo #

a(a|b)*b → a(a|b)*b#

8.1.1.2 Construir el árbol de Sintaxis y dar un identificador único a cada hoja

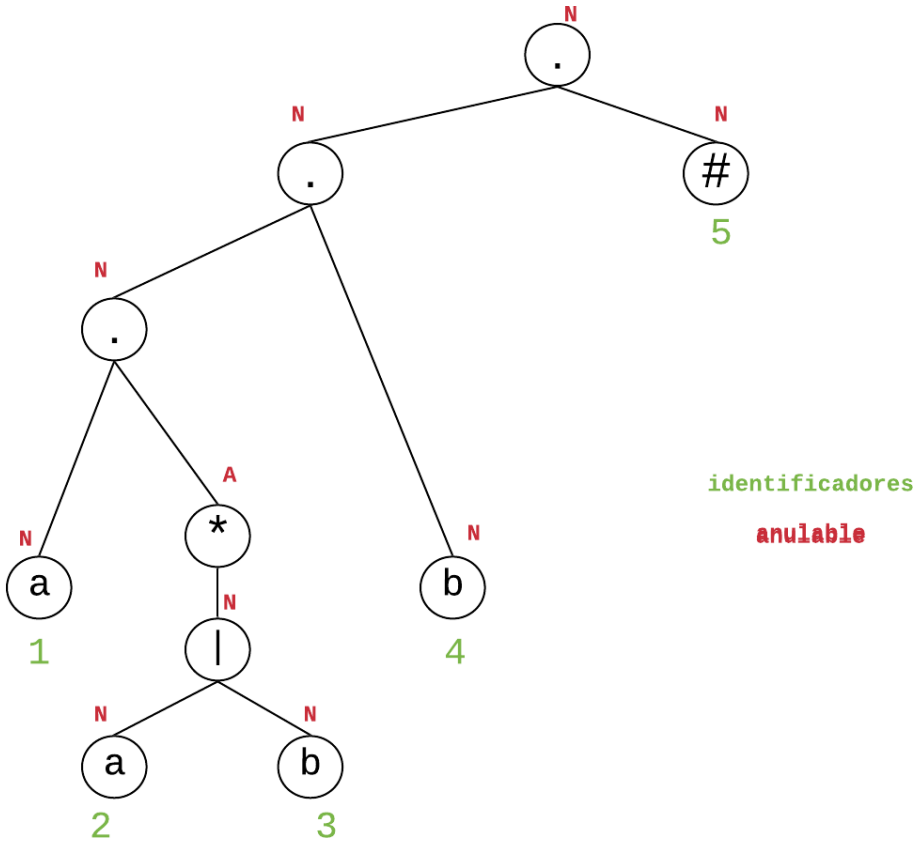


8.1.1.3 Determinar los nodos anulables

Se debe basar utilizando las siguientes reglas:

Terminal	No anulable
C_1*	Anulable
C_1+	No Anulable (Si C_1 es no anulable)
C_1?	Anulable
C_1|C_2	(Anulable(C_1)||Anulable(C_2))?Anulable:No Anulable
C_1C_2	(Anulable(C_1)&&Anulable(C_2))?Anulable:No Anulable

La salida esperada, utilizando las reglas anteriores es:

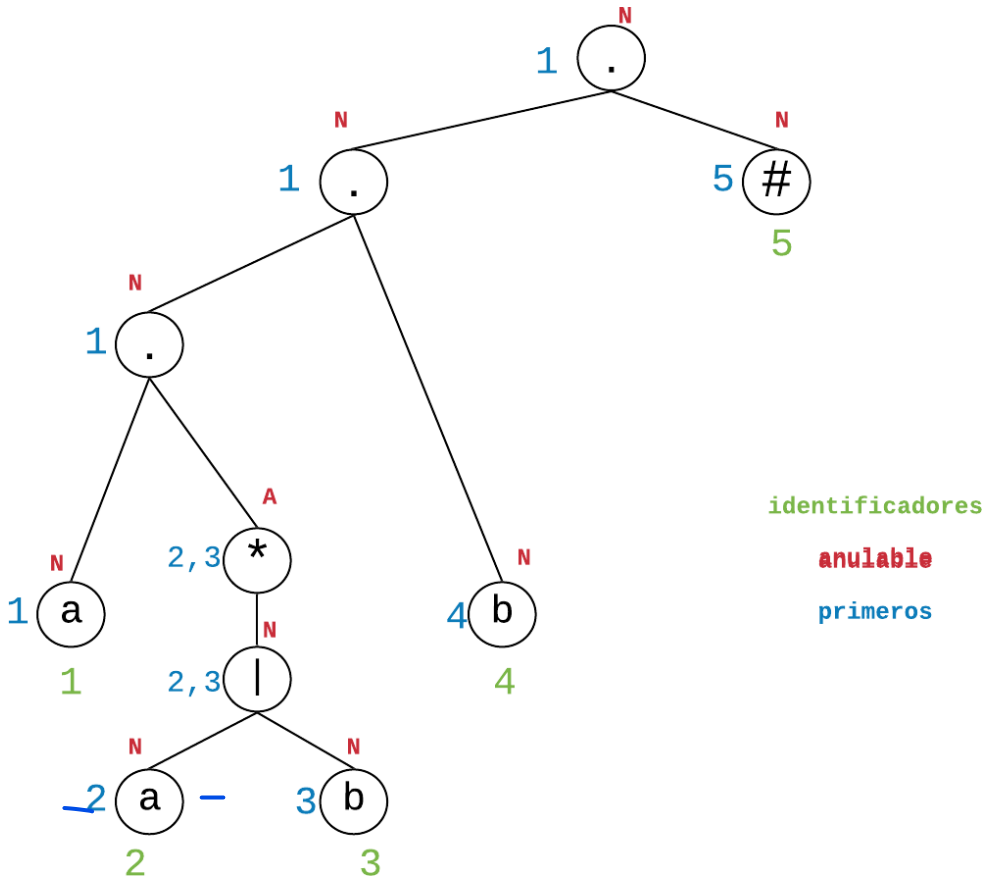


8.1.1.4 Determinar los nodos primeros

Se debe basar utilizando las siguientes reglas:

terminal	Terminal
C_1*	Primeros(C_1)
C_1+	Primeros(C_1)
C_1?	Primeros(C_1)
C_1|C_2	Primeros(C_1) + Primeros(C_2)
C_1C_2	(Anulable(C_1))?(Primeros(C_1) + Primeros(C_2)):Primeros(C_1)

La salida esperada, utilizando las reglas anteriores es:

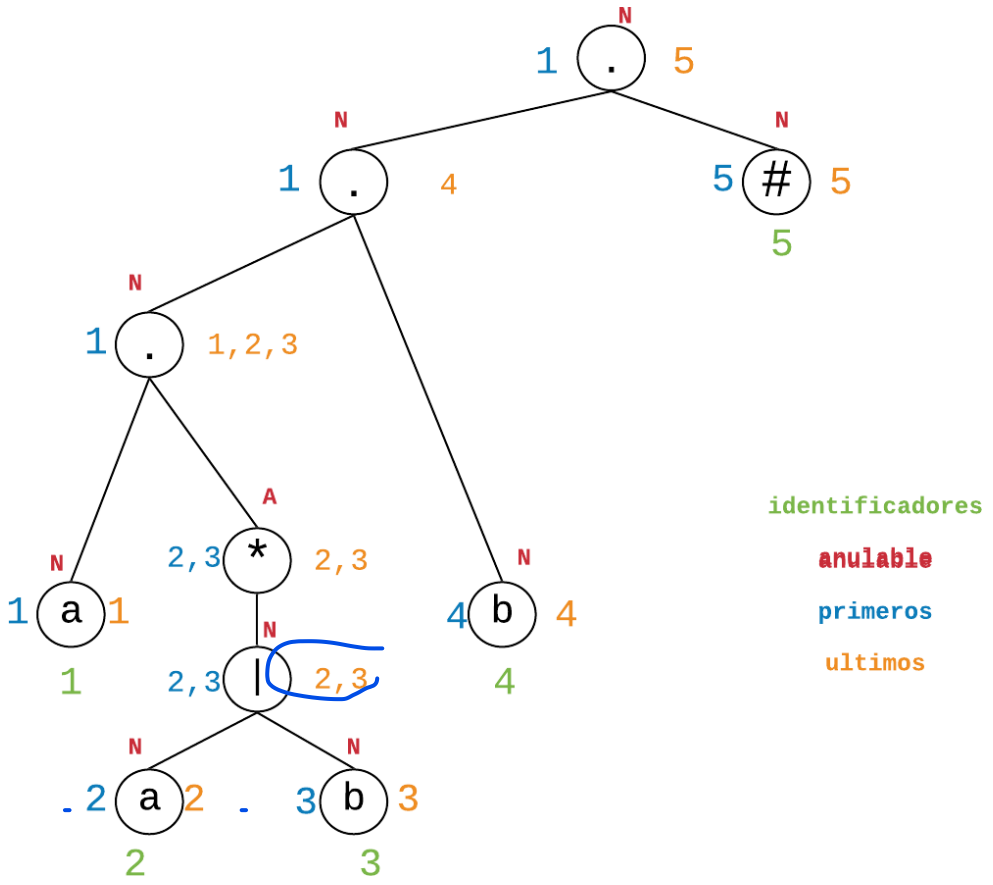


8.1.1.5 Determinar los nodos últimos

Se debe basar utilizando las siguientes reglas:

terminal	Terminal
C_1*	ultimo(C_1)
C_1+	ultimo(C_1)
C_1?	ultimo(C_1)
C_1|C_2	ultimo(C_1) + ultimo(C_2)
C_1C_2	(anulable(C_2))?(ultimo(C_1) + ultimo(C_2)):ultimo(C_2)

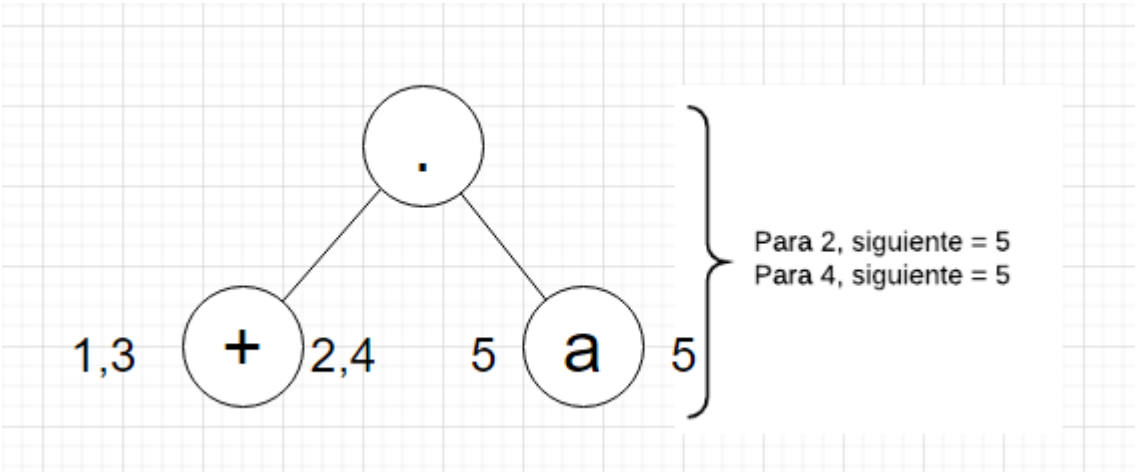
La salida esperada, utilizando las reglas anteriores es:



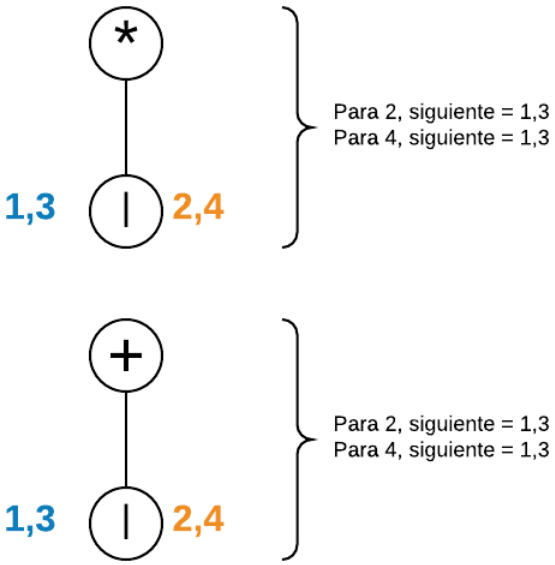
8.1.1.6 Crear Tabla de Siguientes

Solamente Concatenación (·), Cero o Muchas veces (*) y Una o muchas veces (+) tienen Siguiente

En la concatenación los Siguientes para los últimos de C_1 son los primeros del Nodo C_2



Para * y + los siguientes para los últimos de C_1 son los primeros de C_1



La salida esperada, utilizando las reglas anteriores es:

Hoja		Siguientes
a	1	2,3,4
a	2	2,3,4
b	3	2,3,4
b	4	5
#	5	--

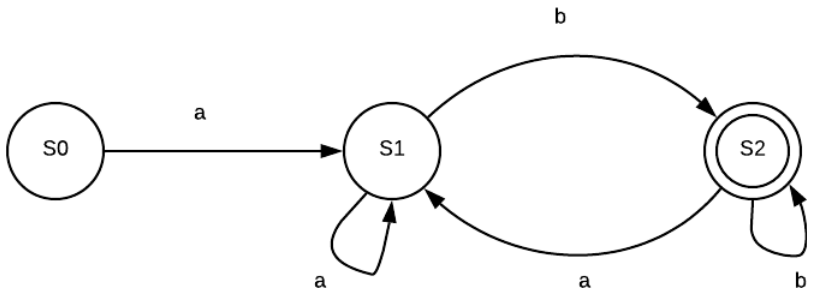
8.1.1.7 Construir la tabla de Transiciones

Estado	**Terminales**	
	a	**b**
S0 {1}	S1	--
S1 {2,3,4}	S1	S2
S2 {2,3,4,5}	S1	S2

El estado inicial son los identificadores que estén en los primeros del nodo raíz

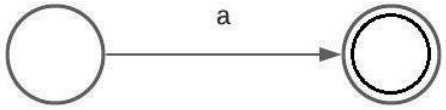
Un estado de aceptación es todo aquel que tenga el identificador del símbolo # agregado inicialmente.

8.1.1.8 Ilustración del AFD construido utilizando la tabla de Transiciones

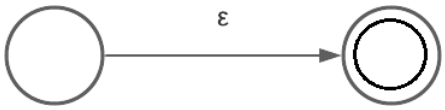


9.2 Reglas del Método de Thompson

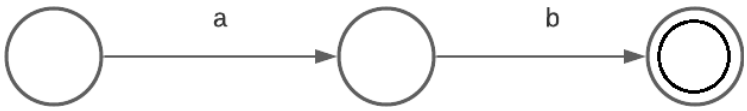
9.2.1 Transición “a”



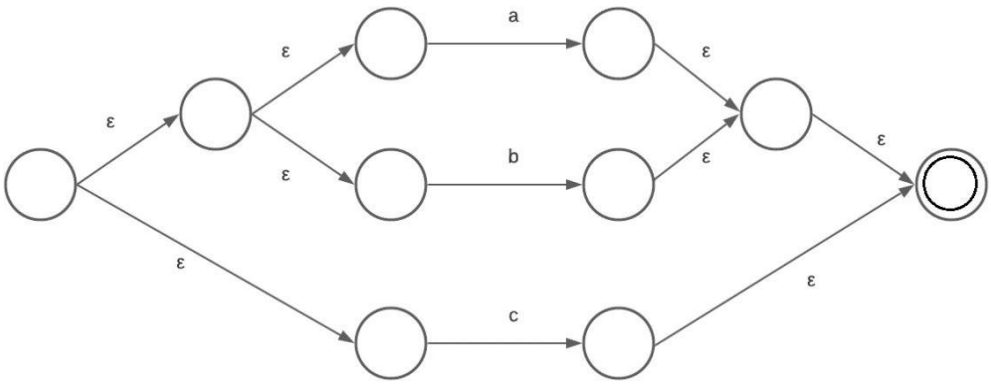
9.2.2 Transición epsilon “ε”



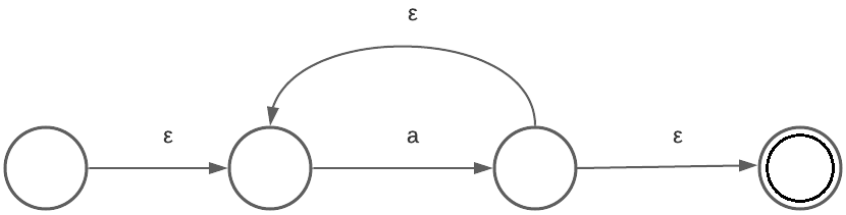
9.2.3 Transición “ab”



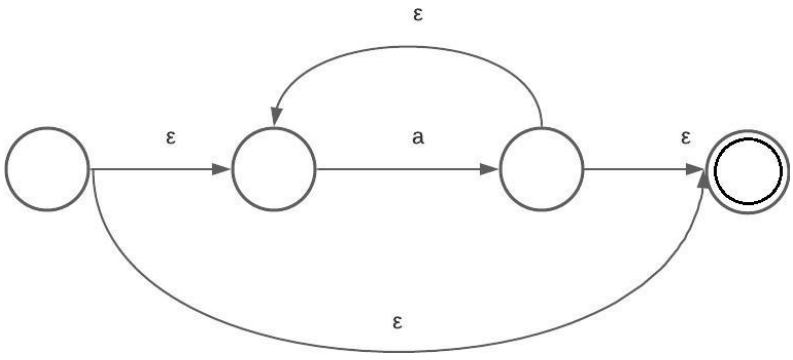
9.2.4 Transición “a|b|c”



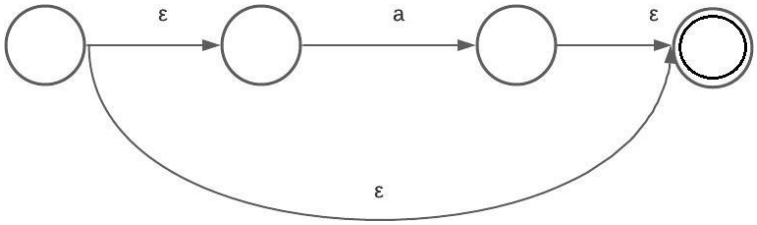
9.2.5 Transición “a^{+}”



9.2.6 Transición “a^{*}”



9.2.7 Transición “a?”

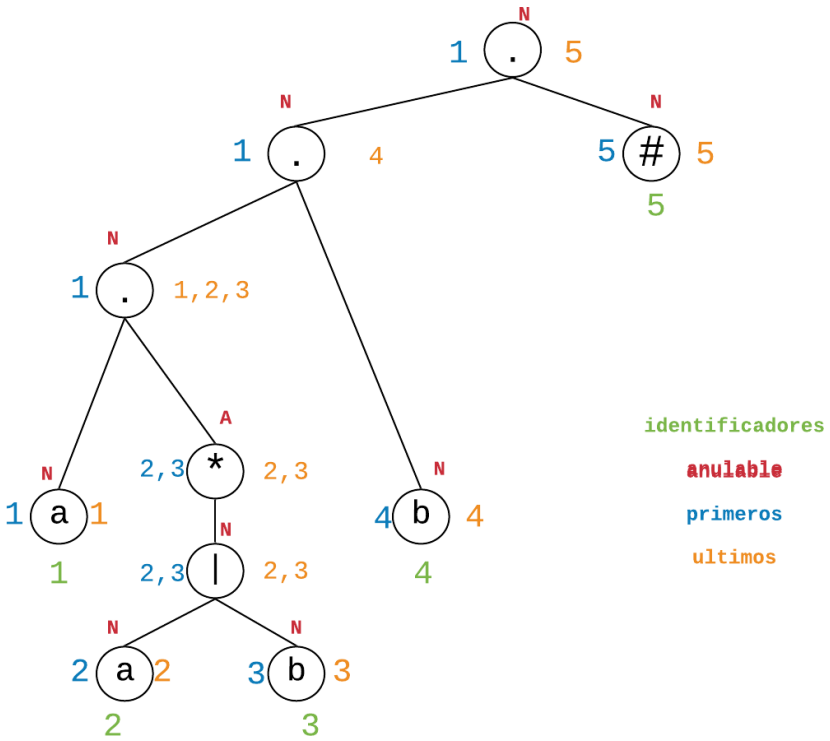


10. Reportes

Los reportes son una parte fundamental de la aplicación “EXREGAN”, ya que muestra de forma visual el resultado del Método del Árbol, Método de Thompson y los errores que se producen al momento de analizar los archivos de entrada (.olc).

A continuación, se muestran los ejemplos de estos reportes. (El diseño de los reportes queda a discreción del estudiante, únicamente se pide que sean totalmente legibles)

10.1 Árbol



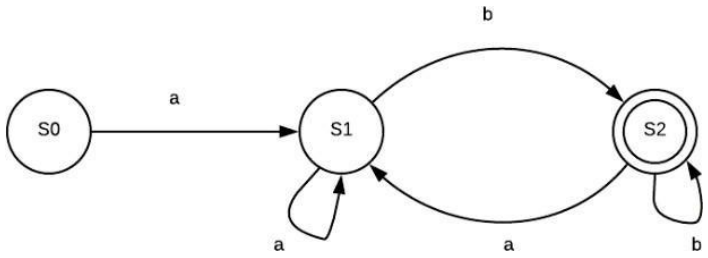
10.2 Siguientes

Hoja		Siguientes
a	1	2,3,4
a	2	2,3,4
b	3	2,3,4
b	4	5
#	5	--

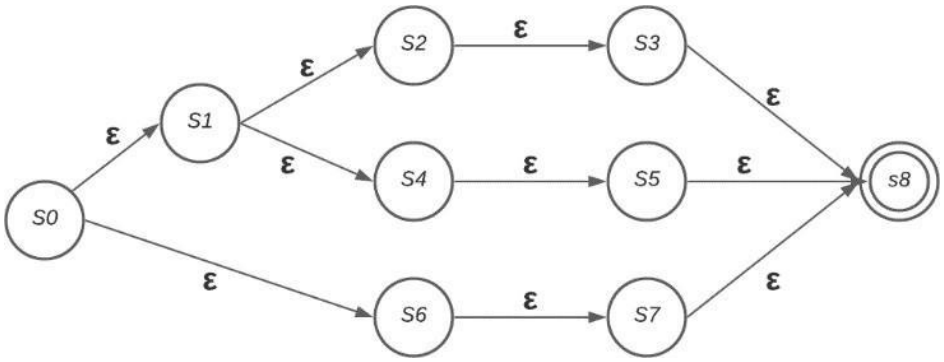
10.4 Transiciones

Estado	Terminales	
	a	b
S0 {1}	S1	--
S1 {2,3,4}	S1	S2
S2 {2,3,4,5}	S1	S2

10.5 AFD



10.5 AFND



10.6 Errores

El reporte de errores debe contener la información suficiente para detectar y corregir errores en el código fuente. Para mostrar este reporte, se requiere que lo muestre como una página HTML.

#	Tipo de Error	Descripción	Línea	Columna
1	Léxico	El carácter “$” no pertenece al lenguaje.	5	32

10.7 Salidas

Las salidas son archivos JSON generados tras un análisis exitoso de la entrada.

Nota: Se debe crear las siguientes carpetas para almacenar los reportes generados durante la ejecución del programa:

- **ARBOLES_#CARNET**: En esta carpeta se almacenarán los reportes de los árboles de expresión.
- **AFD_#CARNET**: En esta carpeta se almacenará los reportes de los grafos de los autómatas finitos deterministas
- **AFND_#CARNET**: En esta carpeta se almacenará los reportes de los autómatas finitos no deterministas.
- **SIGUIENTES_#CARNET:** En esta carpeta se almacenarán los reportes de las tablas de siguientes generados.
- **TRANSICIONES_#CARNET:** En esta carpeta se almacenarán los reportes de las tablas de transiciones.
- **ERRORES_#CARNET:** En esta carpeta se almacenarán los reportes de los errores en los archivos de entrada..
- **SALIDAS_#CARNET:** En esta carpeta se almacenarán los archivos JSON generados como salidas exitosas.

11. Requerimientos Mínimos

Para que el estudiante tenga derecho a calificación, deberá cumplir con la generación completa del Método del Árbol, esto implica lo siguiente:

1. Generar Árbol de Expresiones.
2. Generar Tabla de Siguientes.
3. Generar Tabla de Transiciones.
4. Poseer con al menos 4 commits realizados en el repositorio donde se estuvo trabajando el proyecto.
5. Interfaz gráfica funcional.

Por último, se deberá entregar el archivo de gramática, para verificar que el estudiante trabajó de forma individual.

SI EL ESTUDIANTE NO CUMPLE CON ESTOS REQUERIMIENTOS, NO TENDRÁ DERECHO A CALIFICACIÓN.

12. Entregables

- Código fuente del proyecto.
- Manual de Usuario.
- Manual Técnico.
- Archivo de gramática para la solución (El archivo debe de ser limpio, entendible y no debe ser una copia del archivo de CUP).
- Link al repositorio privado de Github en donde se encuentra su proyecto (Se debe de agregar como colaborador al auxiliar que le califique).

13. Restricciones

1. Lenguajes de programación a usar: **Java**
2. Herramientas a de análisis léxico y sintáctico: **JFlex/CUP**
3. El Proyecto es de manera individual.
4. Para graficar se puede utilizar cualquier librería (Se recomienda Graphviz)
5. Copias completas o parciales de: código, gramática, etc. tendrán una nota de 0 puntos, los responsables serán reportados al catedrático de la sección y a la escuela de Ciencias y Sistemas.
6. La calificación tendrá una duración de 30 minutos, acorde al programa del laboratorio.
7. Se debe poder analizar la cadena de entrada y comprobar el resultado a través del AFD, de lo contrario no se calificará.

14. **Fecha de Entrega**

Sábado 18 de marzo de 2023.

La entrega será por medio de la plataforma UEDI, si existieran inconvenientes con la plataforma se utilizará Classroom PREVIAMENTE INDICADO CON SU RESPECTIVO AUXILIAR.

Entregas fuera de la fecha indicada, no se calificarán.

SE LE CALIFICARA DEL COMMIT REALIZADO HASTA ESTA FECHA.