Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Primer Semestre 2017
Catedrático:
Ing. Mario Bautista
Tutor académico:
Miguel Ruano

J-LEX
Primera Práctica

1. Objetivos

1.1 Objetivo General

Aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para construcción de una solución de software que permita generar transacciones bancarias.

1.2 Objetivos Específicos

- Reforzar el concepto del método de Árbol de expresiones regulares en Autómatas Finitos Deterministas (AFD).
- Identificar y programar el proceso de reconocimiento de lexemas mediante el uso de Autómatas Finitos Determinista.

2. Descripción General

La empresa *Ultimate* es la encargada de realizar diferentes transacciones, las cuales permiten realizar operaciones básicas como definir expresiones regulares para crear autómatas finitos deterministas que se usarán para validar lexemas. Para ello se proponen validar los lexemas en base a un lenguaje entendible únicamente por empleados autorizados, para evitar que este lenguaje sea conocido por empleados que ya no cuentan con los permisos necesarios se realizan cambios constantemente en las expresiones regulares permitidas en el lenguaje.

El sistema debe contar con las siguientes funcionalidades principales:

- Intérprete de Expresiones Regulares Permitidas: Esta funcionalidad analizará un archivo de expresiones regulares que serán las que se permiten dentro del lenguaje que recibe el lenguaje entendible con la información de las Tiendas.

3. Funcionalidades

Se requiere un programa que permita la edición de programas fuente a través de un área de texto. Además, que permita la visualización de los resultados y reportes de análisis.

Menú Archivo

- Nuevo Archivo: Se podrá crear un nuevo archivo con extensión .der.
- Abrir archivo: Nos permitirá abrir un archivo que esté guardado en disco. Los archivos a abrir tendrán extensión [.der].
- Guardar: Nos permitirá guardar el archivo actual.

- Guardar como: Nos permitirá guardar el archivo actual con otro nombre.
- Generar XML de salida: Guardará el archivo en una ruta definida por el usuario.

Comentarios

El lenguaje DER permitirá el uso de comentarios de una o varias líneas. Para comentarios multilínea se utilizará la palabra <! para indicar el inicio del comentario y la palabra !> 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. Interprete de Expresiones Regulares

Este realiza un análisis léxico y sintáctico de un archivo en Lenguaje **DER** (.der) 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 para 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 regulares. 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 (;)

```
{
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.

Notacion	Definicion
. a b	Concatenación entre a y b
| a b	Disyunción entre a y b
*** a**	a 0 o más veces
+ a	a 1 o más veces
? a	A 0 o una vez

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:

Notacion	Definicion
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.

4.1.3 Caracteres Especiales

Dentro del lenguaje pueden utilizarse estos caracteres especiales:

Notacion	Definicion
\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 { corchetes }

```
{
////// 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} {digito}*** para determinar si la entrada cumple con la expresión regular, de ser correcta se escribirá en el archivo XML.

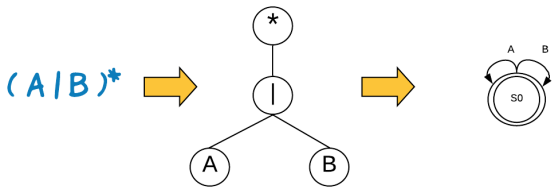
4.1.5 Generación de XML de Salida

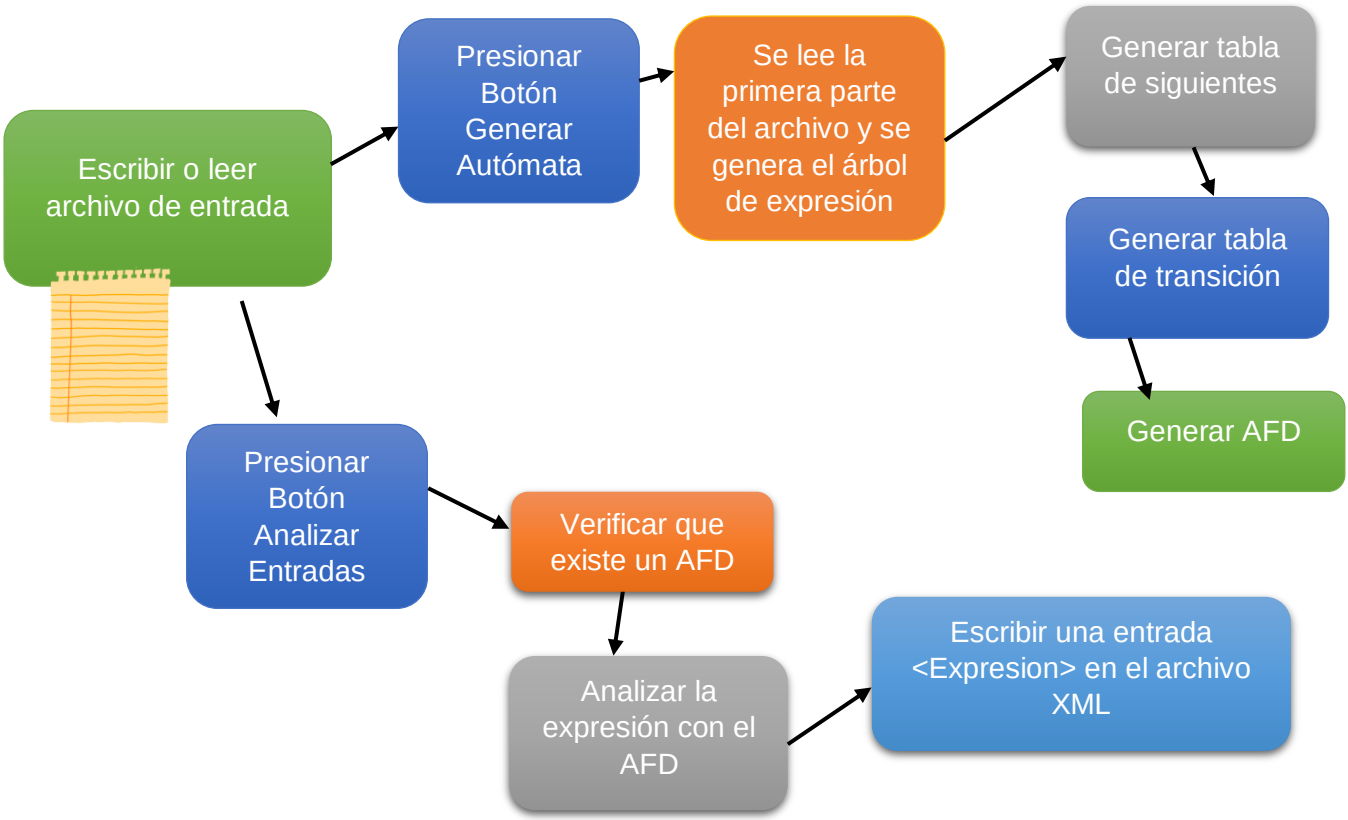
Una vez que se analicen cada una de las líneas debajo del doble porcentaje (%%) se deberá generar un archivo XML cuya ruta será definida por el programador, el cual tendrá las siguientes características:

Archivo de Salida esperada

```
<Salida>
	<Expresion>
		<Valor>primerLexema<Valor/>
		<ER>ExpReg1 </ER>
		<Resultado>Cadena Valida<Resultado/>
	</Expresion>
	<Expresion>
		<Valor>34.44<Valor/>
		<ER>ExpresionReg1 </ER>
		<Resultado>Cadena Valida<Resultado/>
	</Expresion>
</Salida>
```

5. Flujo del programa





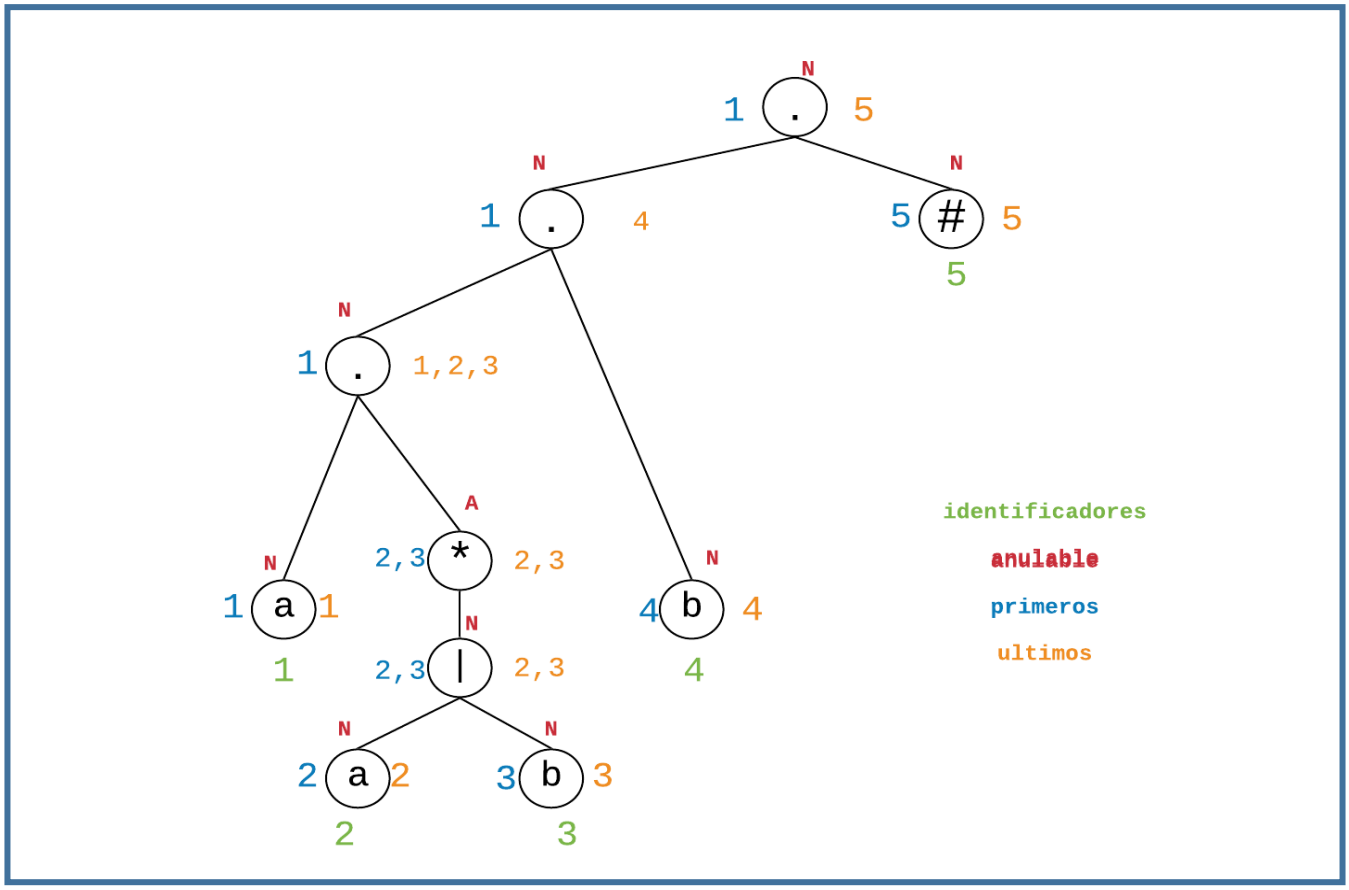
Nota: las gráficas del árbol de expresión se generarán en una carpeta llamada Arboles, el programa debe ser capaz de mostrar todos los archivos de árbol generados por la cadena de entrada (realizar una galería).
También se deben hacer galerías para la tabla de siguientes, la tabla de transición y la gráfica del AFD.

6. Generación de AFD

6.1.1 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**
- **Ultimos**



Gráfica del Árbol de expresión

6.1.2 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.

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

Tabla de siguientes

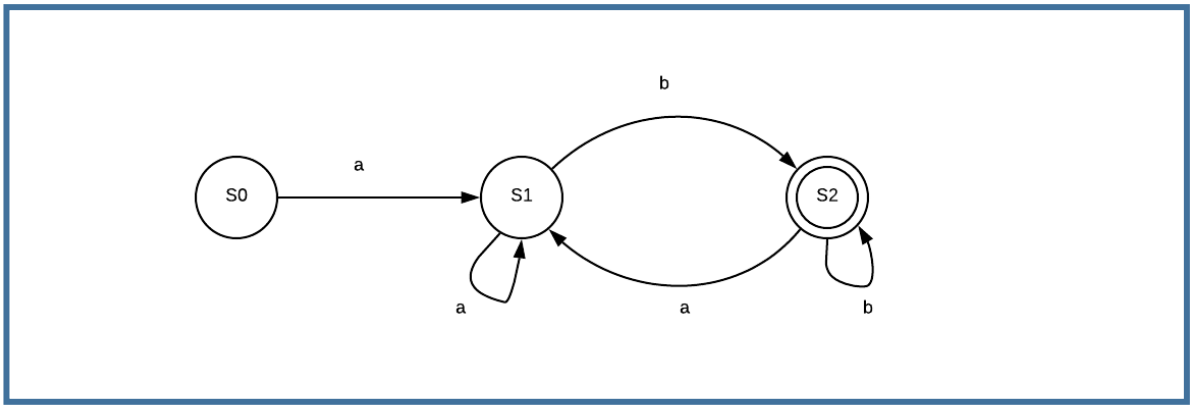
Estado	**Terminales**	
	a	**b**
S0 {1}	S1	--
S1 {2,3,4}	S1	S2
S2 {2,3,4,5}	S1	S2

Tabla de transiciones

Estado de aceptación.

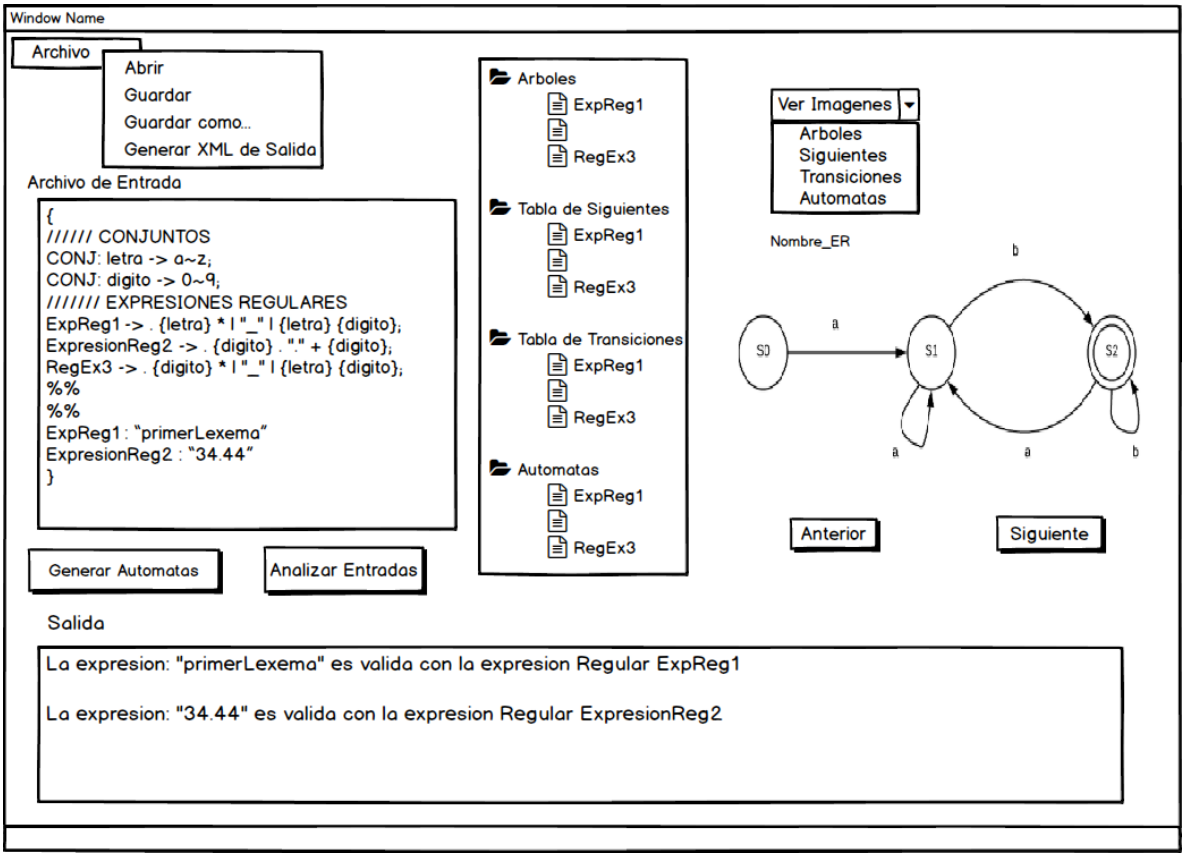
6.1.3 Autómata finito determinístico

Cada tabla de transición deberá usarse para generar el autómata finito determinístico siguiente:



7. 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>**



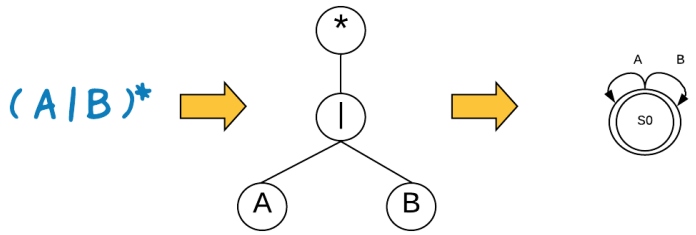
8. Anexos

8.1 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.

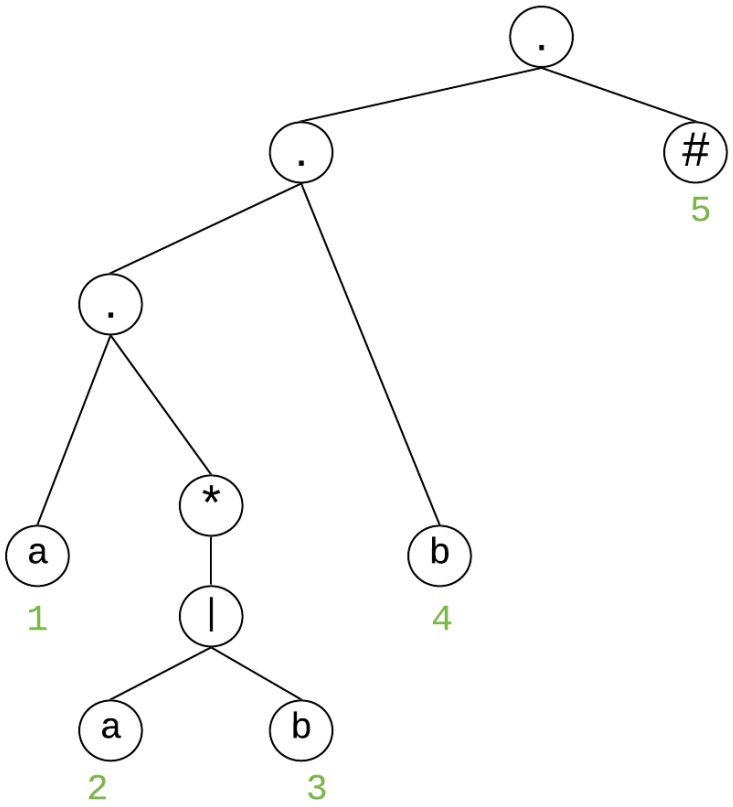


8.1.1 Pasos a seguir para realizar el método del árbol

8.1.1.1 Agregar al final de la expresión regular el símbolo #

$$a(a|b)^*b \rightarrow a(a|b)^*b\#$$

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

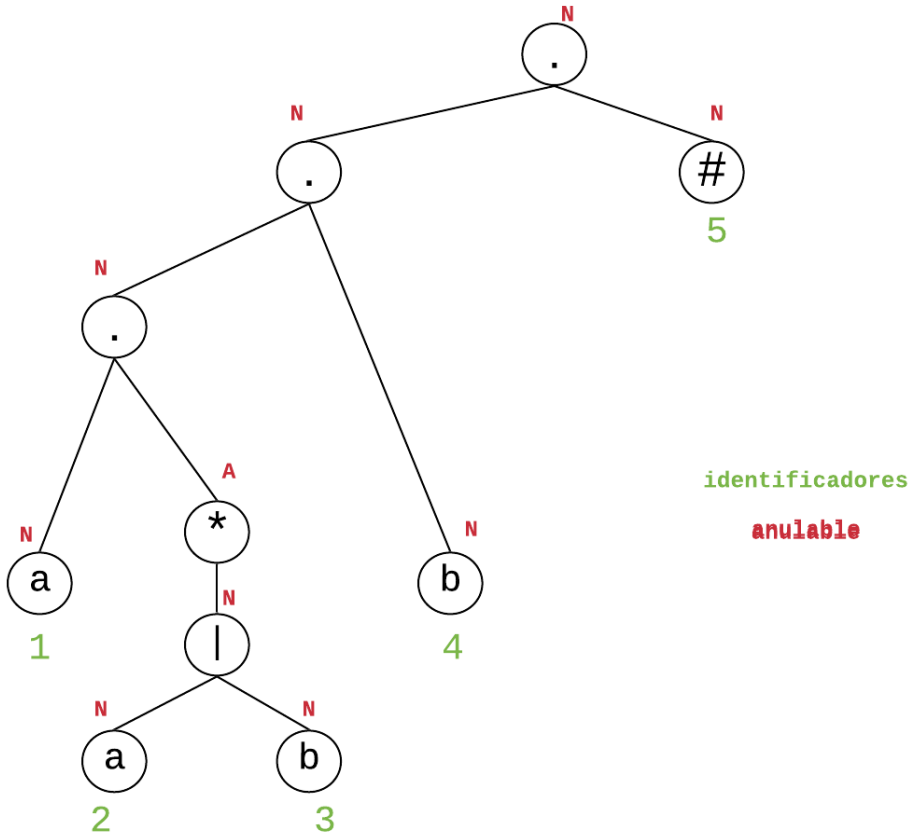


8.1.1.3 Determinar los nodos anulables

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:

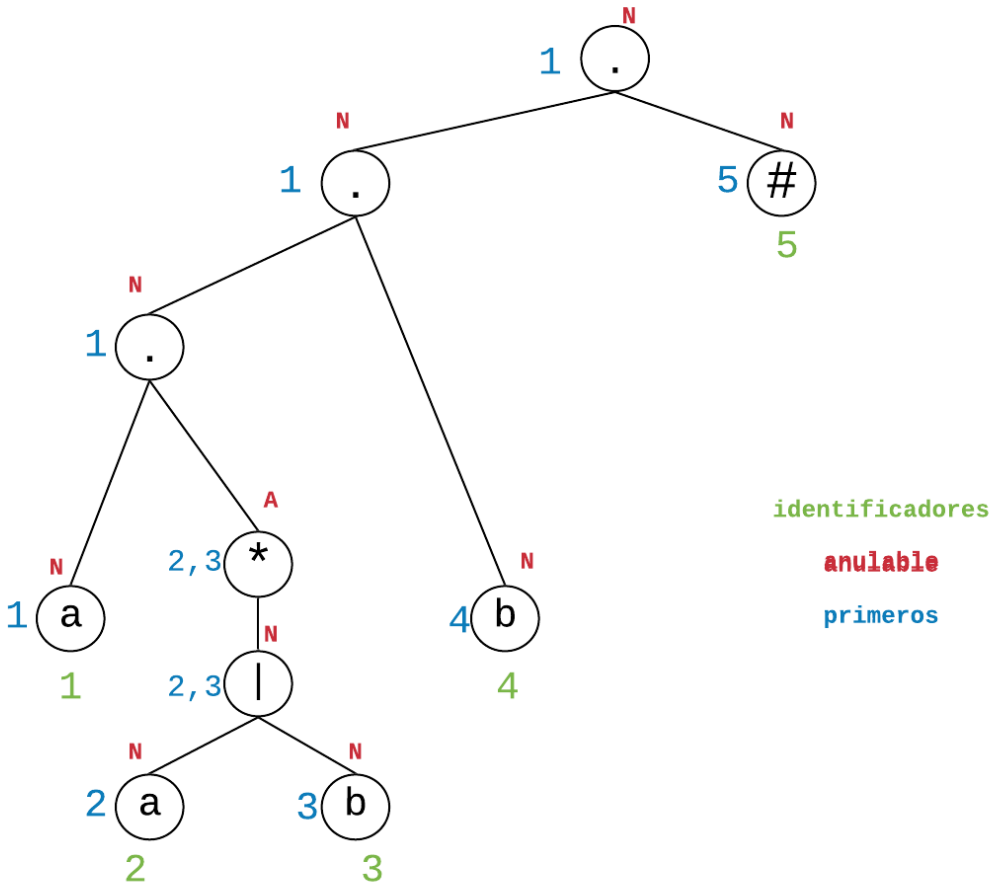


8.1.1.4 *Determinar los nodos primeros*

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:

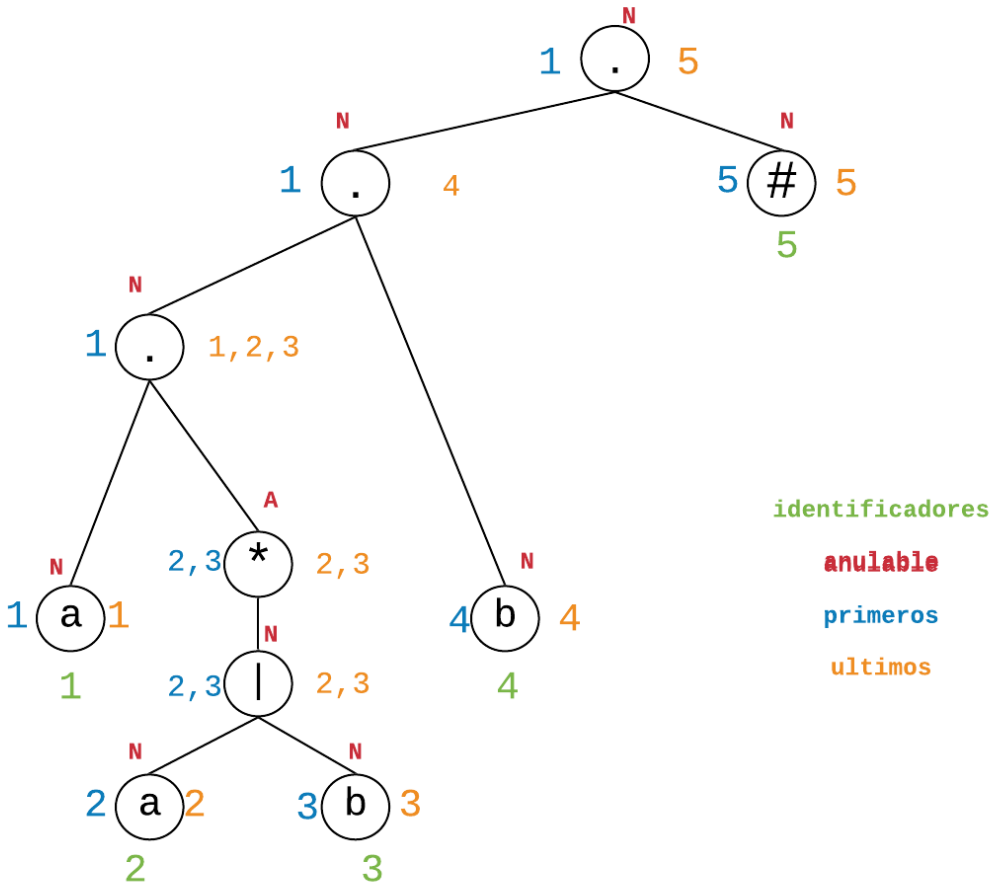


8.1.1.5 *Determinar los nodos últimos*

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)

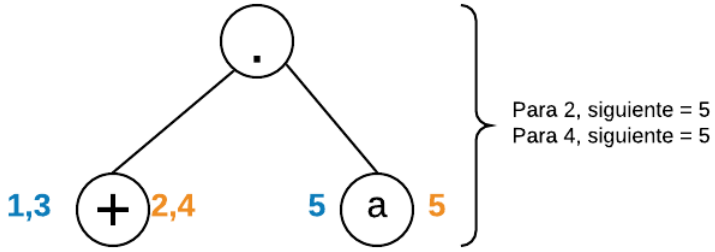
La salida esperada, utilizando las reglas anteriores es:



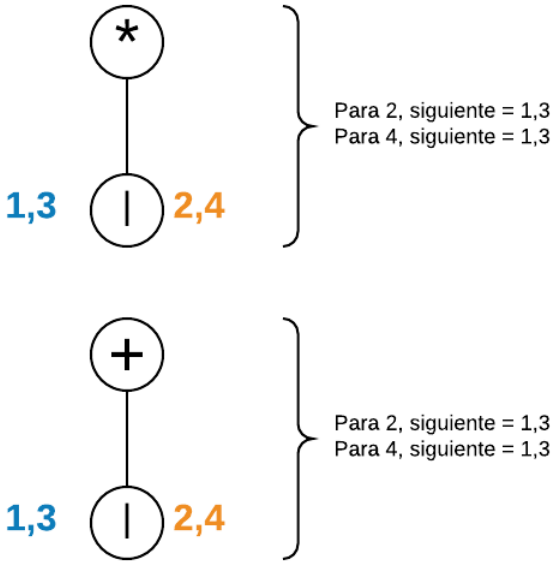
8.1.1.6 Crear Tabla de Siguientes

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	--

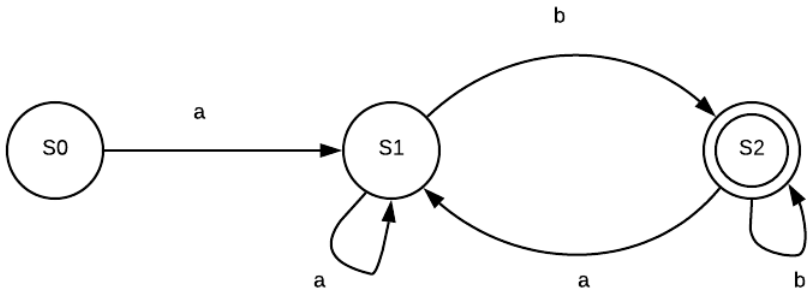
8.1.1.7 Construir la tabla de Transiciones

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



Entregables

Código Fuente, manuales de usuario y técnico, archivo de gramáticas.

Restricciones

Lenguajes de programación a usar: Java
Herramientas de análisis léxico y sintáctico: JFlex/CUP
Para graficar se puede utilizar cualquier librería
La Práctica es Individual
Copias completas/parciales de: código, gramáticas, etc. serán merecedoras de una nota de 0 puntos, los responsables serán reportados al catedrático de la sección y a la Escuela de Ciencias y Sistemas.
Se debe poder analizar la cadena de entrada a través del AFD, de lo contrario no se calificará

Fecha de Entrega

Domingo 09 de Junio de 2019 (se estará indicando el medio y forma de entrega)