Universidad Rafael Landívar
Facultad de Ingeniería
Ingeniería en Informática y Sistemas
Catedrático: Moises Alonso

PROYECTO PRÁCTICO FASE II

Generador de Scanner

Daniel Fernando Cabrera Reyes
1117121

José Mario Marroquín Roldán
1234621

Diego Andres Bautista Cruz
1181821

Guatemala, 27 de febrero del 2023

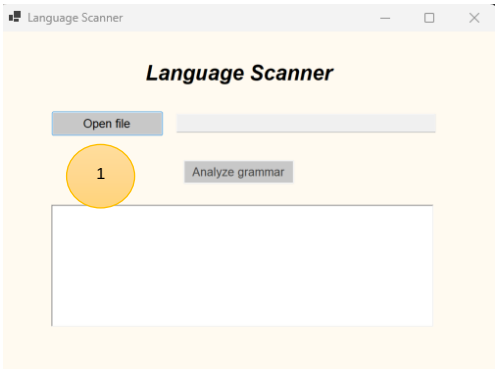
I. Descripción del Proyecto

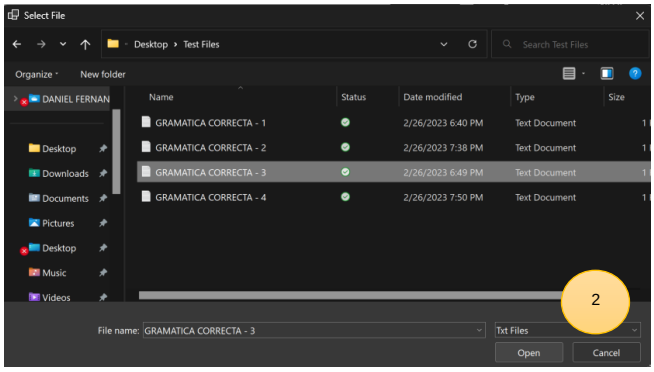
El proyecto presentado consiste en una aplicación generadora de Scanner, la cual tiene como objetivo analizar la estructura léxica de la definición de una gramática contenida en un archivo de texto. El análisis consiste en la validación y comprobación del formato específico para los elementos que conforman a la gramática, los cuales son: SETS, TOKENS, ACTIONS y ERROR. Cada una de las secciones cuenta con su propio objetivo, formato y elementos, requerido en conjunto para una correcta gramática. La correcta verificación del formato de la gramática a utilizar permitirá en un futuro la correcta construcción del lenguaje que describe esta gramática.

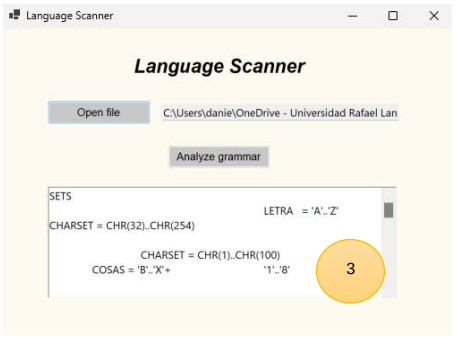
II. Manual de Usuario

1. Abrir archivo de texto

Al pulsar el botón “Open file”, se abrirá una ventana de diálogo, donde se podrá buscar el archivo a analizar. Al encontrar el archivo, hacer clic sobre él y presionar “Abrir” en la esquina inferior derecha de la ventana de diálogo. Al finalizar este procedimiento, se mostrará el texto del archivo en la caja de texto del programa.







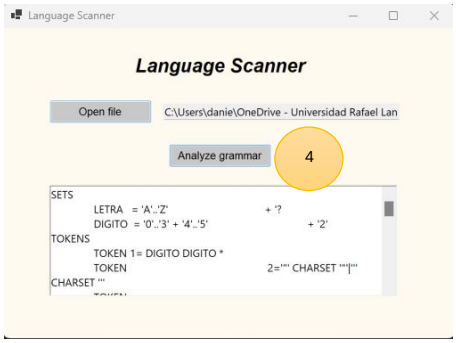
En el repositorio de GitHub existe una carpeta “Inputs” el cual contiene dos carpetas, una para los textos con los lenguajes con sintaxis correcta y el otro con sintaxis incorrecta.

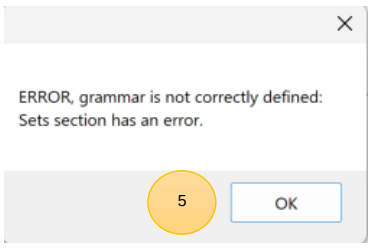
Repositorio de GitHub: https://github.com/danieelfcr/LanguageScanner.git

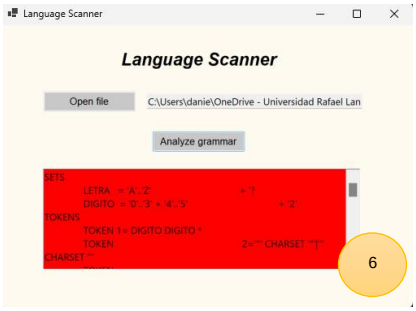
2. Analizar gramática

Al presionar el botón “Analyze grammar”, se comenzará a analizar la gramática del texto incluido en el archivo. Al finalizar el análisis, se observará una ventana de diálogo la cuál mencionará si la gramática es correcta o incorrecta. En caso de ser incorrecta, se mostrará la sección en la cuál la gramática es errónea.

Al cerrar las ventanas emergentes, la caja de texto cambiará de color dependiendo si la gramática del texto analizado es correcto (verde) o si es incorrecto (rojo).



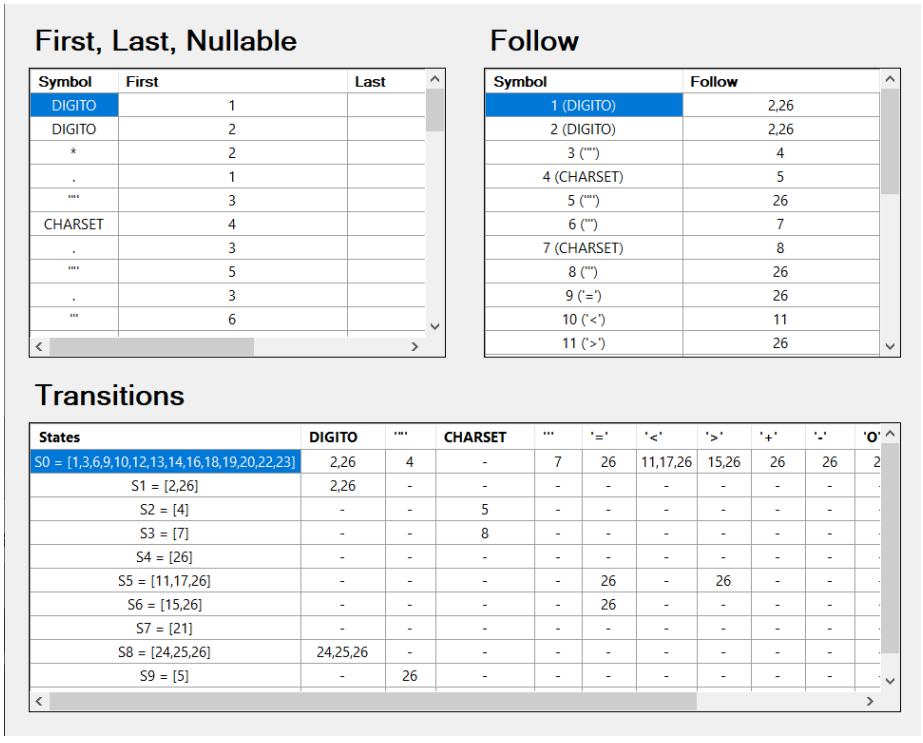




3. Generación del árbol de expresiones y el autómata finito determinista

Después de analizar la gramática y determinar que fue ingresada correctamente, el sistema se encarga de generar de forma automática un árbol de expresiones con base a la expresión regular que forman los tokens proporcionados. El objetivo principal de este árbol es servir como una estructura auxiliar para obtener los componentes básicos que posteriormente formarán al autómata finito determinista. Después de la creación del árbol de expresiones se mostrarán en pantalla tres cuadros que presentan la información completa del autómata generado por la expresión regular.

La primera, ubicada en la esquina superior izquierda presenta las listas First y Last y el indicador Nullable de cada símbolo. La segunda, a la derecha de la primera, muestra las listas Follow para cada símbolo terminal o no terminal que fue encontrado en la expresión. La última tabla, debajo de las dos mencionadas, proporciona la información más relevante del autómata creado: sus estados, con los cuales es posible definir cadenas aceptadas o no aceptadas por la gramática proporcionada.



First, Last, Nullable

Symbol	First	Last
DIGITO	1	
DIGITO	2	
*	2	
.	1	
""	3	
CHARSET	4	
.	3	
""	5	
.	3	
""	6	

Follow

Symbol	Follow
1 (DIGITO)	2,26
2 (DIGITO)	2,26
3 ("")	4
4 (CHARSET)	5
5 ("")	26
6 ("")	7
7 (CHARSET)	8
8 ("")	26
9 ('=')	26
10 ('<')	11
11 ('>')	26

Transitions

States	DIGITO	""	CHARSET	""	'='	'<'	'>'	'+'	'-'	'O'
S0 = [1,3,6,9,10,12,13,14,16,18,19,20,22,23]	2,26	4	-	7	26	11,17,26	15,26	26	26	2
S1 = [2,26]	2,26	-	-	-	-	-	-	-	-	
S2 = [4]	-	-	5	-	-	-	-	-	-	
S3 = [7]	-	-	8	-	-	-	-	-	-	
S4 = [26]	-	-	-	-	-	-	-	-	-	
S5 = [11,17,26]	-	-	-	-	26	-	26	-	-	
S6 = [15,26]	-	-	-	-	26	-	-	-	-	
S7 = [21]	-	-	-	-	-	-	-	-	-	
S8 = [24,25,26]	24,25,26	-	-	-	-	-	-	-	-	
S9 = [5]	-	26	-	-	-	-	-	-	-	

III. Manual Técnico

a) Especificaciones

- **Lenguajes de programación utilizado:** C#
- **Versión:** .NET 6.0
- **Sistema operativo:** Windows
- **Versión sistema operativo:** 7.0 en adelante
- **Recursos utilizados:** Archivo de texto (.txt)

b) Características de la Gramática

a. SETS

La sección de SETS corresponde a la definición de un conjunto de símbolos no terminales a los que se les asigna un conjunto de símbolos terminales, los cuáles pueden ser caracteres o dígitos. Entre las características de esta sección se encuentran:

- ✓ Esta sección puede no venir dentro del archivo de texto.
- ✓ Si esta sección viene, debe poseer al menos un SET.
- ✓ La palabra SETS debe de venir en mayúscula.
- ✓ Se puede utilizar la función CHR(Número del 0 al 255).
- ✓ Puede haber muchos espacios entre el identificador, el símbolo “=” y la definición.
- ✓ Puede haber muchos saltos de línea.
- ✓ Para identificar que se desea de un carácter a otro (ya sea con la función CHR() o con el carácter en sí), se debe de colocar el no terminal seguido de dos puntos (..) y el otro terminal sin espacios.
- ✓ Los sets pueden estar concatenados con el signo “+”. En este caso sí pueden existir espacios.

b. TOKENS

La sección TOKENS consiste en un conjunto de definiciones para una expresión regular, que se utilizará en versiones posteriores para un análisis sintáctico de la gramática. Es una sección de gran importancia, pues combina algunas características de las demás secciones. Sus principales características son:

- ✓ Debe ser una sección obligatoria.
- ✓ Tiene que comenzar con la palabra TOKENS.
- ✓ Cada elemento dentro de la sección debe comenzar con la palabra TOKEN, seguido de un número de identificación y un signo igual.
- ✓ Al menos un TOKEN debe existir.
- ✓ Pueden haber indefinidos saltos de línea y espacios o tabulaciones entre dos TOKENS
- ✓ Cada TOKEN puede contener uno o más símbolos.
- ✓ Los símbolos permitidos son no terminales (en mayúsculas) o terminales (un solo símbolo que puede ser o no una letra mayúscula y está encerrado entre comillas simples).
- ✓ Cada símbolo puede ser acompañado o no de uno de los tres operadores básicos de una expresión regular (+, *,?).
- ✓ Pueden existir paréntesis para agrupar símbolos, pero cada apertura debe contar con su respectivo cierre.
- ✓ La disyunción entre dos símbolos se representa con el operador (|), el cual debe tener obligatoriamente un símbolo a cada lado para ser válido.
- ✓ Se pueden agregar llamadas a funciones definidas en la sección ACTIONS, siempre y cuando estén dentro de una apertura y cierre de los símbolos de llaves ({ , }).
- ✓ Cada función debe escribirse en mayúsculas y finalizar con una apertura y cierre de paréntesis.

c. ACTIONS

La sección de ACTIONS corresponde a un conjunto de funciones que definen las palabras reservadas del lenguaje. Estas definiciones contenidas en diversas funciones serán útiles para brindarle un significado implícito a palabras escritas de forma específica. Algunas de las características son:

- ✓ Las funciones contenidas poseer: un identificador, paréntesis abierto y cerrado y tokens,
- ✓ Los tokens se forman de un identificador numérico, un signo igual y la definición de la palabra contenida en apóstrofes,
- ✓ El identificador de los tokens debe ser únicamente numérico y en la definición de la palabra solo puede utilizarse letras,
- ✓ La función “RESERVADAS()” siempre debe estar contenida en la sección una sola vez

d. ERROR

La sección de ERROR se refiere al contenido de todos los posibles mensajes de error que se utilizarán para el lenguaje. En la gramática no cuenta con ningún encabezado, ya que son una lista secuencial de errores con las siguientes características:

- ✓ Cada error está conformado por: un identificador, un signo de igualdad y un número
- ✓ El identificador puede ser conformado únicamente por letras y este debe tener por los menos la palabra “ERROR” como sufijo
- ✓ Del lado derecho de la igualdad solamente se definen valores numéricos

c) Expresiones regulares utilizadas

Para la validación del archivo de texto que contiene la gramática mediante el scanner se utilizaron expresiones regulares. Las cuales validan exclusivamente la construcción léxica de la gramática; esto quiere decir que se verifica el cumplimiento de los formatos específicos de cada sección.

Para la construcción de las expresiones regulares en el lenguaje de programación se importó una librería que contiene la definición de una clase para la escritura y revisión de expresiones regulares, propia de C#, “Regex”. Esto conlleva al uso de términos específicos del lenguaje, tales como:

- ‘\\S’: cualquier carácter que no se un dígito.
- ‘\\t’: tabulación
- ‘\\n’: salto de línea
- ‘\\s’: cualquier tipo de espacio en blanco (\\n, \\t, \\r, \\b)
- ‘[X-X]’: la definición de rangos de números y letras
- ‘*’: cerradura estrella
- ‘+’: cerradura positiva
- ‘|’: alternación, OR
- ‘()’: signos de agrupación

A continuación, se presentará la expresión regular utilizada para cada sección, así como una breve explicación del funcionamiento de cada una:

✓ Expresión regular: “SETS”

```
(\\s*SETS(\\s)+((\\s)*([A-Z])+( |\\t)*=( |\\t)*('([a-z])'\\.\\.'([a-z])'|'([A-
Z])'\\.\\.'([A-Z])'|'.'|'([0-9])'\\.\\.'([0-9])'|CHR\\((([0-9])|([0-9])([0-9])|0([0-
9])([0-9])|1([0-9])([0-9])|2([0-5])([0-5]))\\)(\\.\\.CHR\\((([0-9])|([0-9])([0-
9])|0([0-9])|0([0-9])([0-9])|1([0-9])([0-9])|2([0-5])([0-5]))\\))?)(( |\\t)*\\+(
|\\t)*('([a-z])'\\.\\.'([a-z])'|'([A-Z])'\\.\\.'([A-Z])'|'.'|'([0-9])'\\.\\.'([0-
9])'|CHR\\((([0-9])|([0-9])([0-9])|0([0-9])([0-9])|1([0-9])([0-9])|2([0-5])([0-
5]))\\)(\\.\\.CHR\\((([0-9])|([0-9])([0-9])|0([0-9])|0([0-9])([0-9])|1([0-9])([0-
9])|2([0-5])([0-5]))\\))?))*)+\\s*)?
```

Esta sección, según la expresión regular presentada, puede o no venir dentro del archivo de texto. Si la sección viene, debe de contener obligatoriamente la palabra “SETS” y un set. Luego de la palabra “SETS” debe de venir un salto de línea. Luego, pueden venir muchos saltos de línea, espacios o tabulaciones para encontrarse con el primer símbolo no terminal. Los símbolos no terminales deben estar escritos en mayúscula obligatoriamente. Pueden venir muchos espacios entre el no terminal y el símbolo “=”, al igual que entre el símbolo “=” y el primer símbolo terminal. Los símbolos terminales pueden ser definidos como un solo carácter entre comillas simples, puede definirse un rango entre caracteres, por ejemplo ‘A’..’Z’, en donde los terminales deben de estar escritos entre comillas y deben de separarse por dos puntos “..” sin espacios. También puede definirse un carácter con ASCII con la función CHR(), tomando en cuenta que los caracteres ASCII van desde 0 hasta 255. También se puede definir un rango con las funciones CHR() utilizando las mismas reglas descritas anteriormente. Los símbolos terminales pueden estar concatenados utilizando el símbolo “+”, siguiendo las mismas reglas, pudiendo haber muchos espacios entre los símbolos concatenados.

✓ Expresión regular de la sección “TOKENS”

```
\\s*TOKENS\\s*\\n(\\s*TOKEN( |\\t)*([1-9][0-9]*)( |\\t)*=( |\\t)*(((( |\\t)*{(
|\\t)*([A-Z]+( |\\t)*\\(( |\\t)*\\)( |\\t)*)+( |\\t)*}( |\\t)*)|(([A-Z]+(
|\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?))|(( |\\t)*(([A-Z]+(
|\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?))+( |\\t)*\\|( |\\t)*(([A-Z]+(
|\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?))( |\\t)*(\\|( |\\t)*(([A-Z]+(
|\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?)( |\\t)*)+)*)|(\\(( |\\t)*((((A-
Z]+( |\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?))( |\\t)*)|(( |\\t)*(([A-
Z]+( |\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?))+( |\\t)*\\|( |\\t)*(([A-
Z]+( |\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?))( |\\t)*(\\|( |\\t)*(([A-
Z]+( |\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?)( |\\t)*)+)*)|(\\((
|\\t)*((((A-Z]+( |\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?))( |\\t)*)|((
|\\t)*(([A-Z]+( |\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?))+( |\\t)*\\|(
|\\t)*(([A-Z]+( |\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?))( |\\t)*(\\|(
|\\t)*(([A-Z]+( |\\t)*(\\*|\\+|\\?)?)|('\\S'( |\\t)*(\\*|\\+|\\?)?)(
|\\t)*)+)*))+\\)( |\\t)*(\\*|\\+|\\?)?))+\\)( |\\t)*(\\*|\\+|\\?)?))( |\\t)*)+)+\\s*
```

TOKENS es una sección obligatoria que tiene el objetivo de representar expresiones regulares siguiendo un formato riguroso. La primera de sus características es que debe de contar con la palabra específica “TOKENS”, seguida de caracteres de espacio y un salto de línea obligatorio. Cada TOKEN es acompañado de un número de identificación que puede repetirse y no tiene limitantes, seguido del signo “=”. A partir de dicho formato pueden ocurrir cuatro casos distintos que ayudarán a formar una expresión regular válida, distinta a las expresiones usadas para validar el formato de la gramática. El primer caso es para determinar símbolos terminales y no terminales, siendo los primeros siempre encerrados entre comillas simples, ambos tipos de símbolos pueden ser acompañados, o no, de uno de los tres operadores básicos de una expresión regular (*, +,?).

El segundo caso permite utilizar el símbolo de la disyunción para expresiones regulares (|) y puede generar cadenas de varias disyunciones, obligando a que el formato de entrada respete que ambos lados de la disyunción deben de existir. El tercer caso se enfoca en los símbolos de agrupación o paréntesis, con los cuales son válidos tanto el primer caso como el segundo, además, es posible contener una pareja de paréntesis dentro de otra. La expresión regular mostrada se encarga de validar que una apertura de paréntesis siempre esté acompañada de su respectivo cierre, evitando errores de formato. El cuarto caso es el encargado de permitir el uso de funciones dentro de un TOKEN, para lo cual siempre deberá abrirse y cerrarse una pareja de llaves ({,}) en donde dentro pueden existir varias llamadas a funciones acompañadas de una pareja abierta y cerrada de paréntesis.

✓ Expresión regular: “ACTIONS”

```
ACTIONS\\s*RESERVADAS\\s*\\(\\s*\\)\\s*{\\s*([0-9]|([1-9][0-9]*))\\s*=\\s*'([A-Z]|[a-z])+'\\s*)+}\\s*([A-Z]|[a-z])+\\s*\\(\\s*\\)\\s*{\\s*([0-9]|([1-9][0-9]*))\\s*=\\s*'([A-Z]|[a-z])+'\\s*)+}\\s*)*
```

La sección de ACTIONS debe ser definida en la gramática. Al comienzo de esta debe tener la palabra ‘ACTIONS’, a la cual puede seguir luego muchos o ningún espacio en blanco, que ahora en adelante se referirá a estos como “espacios”.

Luego comienza la definición de funciones con su identificados con letras mayúsculas, espacios, paréntesis abierto, espacios, paréntesis cerrar espacios y se abren llaves, lo que se refiere al contenido de las funciones que son tokens propios de esta sección. En cuanto a la definición de funciones, la primer en ser declarada es la función “RESERVADAS()” que debe existir obligatoriamente en la gramática, después pueden venir o no más funciones que cumplan el formato.

En cuanto al contenido de las funciones, los tokens de la función son definidos mediante 3 elementos: un identificador numérico, un signo de igualdad y una palabra contenida en apóstrofes. La forma de evaluación es así: espacios, números como identificador, espacios, signo igual, espacios, palabra contenida en apóstrofes y espacios; así en cada token. La función debe tener por lo menos un token y puede tener varios.

✓ Expresión regular: “ERROR”

```
([A-Z]*ERROR\\s*=\\s*[1-9][0-9]*\\s*)+
```

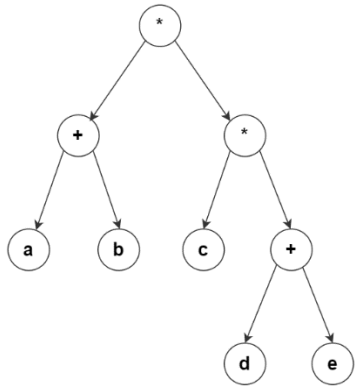
La sección de ERROR se refiere a la definición de errores utilizados por el lenguaje generado por la gramática analizada correctamente. Esta sección comienza inmediatamente después de la sección de ACTIONS, luego de la última llave cerrada de la última función. En cualquier gramática siempre debe haber definido al menos un error y puden definirse muchos. La estructura de los errores es de la forma: el identificador del error, un signo de igualdad y un número. El identificador del error se compone únicamente de letras y al menos la palabra “ERROR” al final, o bien solo esta. La forma de evaluarlo es así: espacios, identificador de error, espacios, signo de igualdad, espacios, número, espacios.

d) Autómata Finito Determinista

En la generación de código, tras realizar el análisis léxico de la gramática, ahora se realiza el análisis sintáctico, para posterior evaluación del correcto ingreso de diversas entradas. Para esto se utilizó la herramienta de una máquina de estados/transiciones por medio de un autómata finito determinista. Este autómata finito determinista permite el análisis de cadenas por medio de algunos elementos: una gramática, árboles de expresiones, funciones first, last y nullable, función follow y una estructura de transiciones de estados.

✓ Árbol de expresiones

Ilustración 1
Representación gráfica de un árbol de expresiones



En la construcción de la gramática se comienza con una expresión regular dada por un alfabeto y lenguaje. Dicha expresión tiene un producto único debido al orden de las operaciones que se realizan en esta, una jerarquía. Para analizar dinámicamente esta jerarquía se utiliza un árbol de expresiones; así como también es de utilidad para la creación del autómata finito determinista más adelante.

Básicamente el árbol describe una expresión regular, donde todos los valores de las hojas son símbolos terminales o no terminales. Los nodos intermedios y raíz son las operaciones para realizar. En este caso, los operandos válidos que trabajaremos son +, *, ?, |. Las operaciones se realizan de forma similar a como se interpreta un arbol binario, el nodo intermedio es la operación sobre las símbolos de hijo izquierdo y derecho de este. Las operaciones que se encuentran en los niveles más bajos del árbol son aquellas con mayor prioridad o que deben realizarse antes.

Sin embargo, hay una anotación en particular sobre las operaciones, ya que estas pueden ser de tipo unario o binario. Las operaciones binarias se trabajan de la forma antes mencionada; mientras que, para los operadores unarios solo se cuenta con un símbolo como hijo, así que la operación va sobre este único hijo.

✓ Funciones First-Last-Nullable

Las funciones de first, last y nullable se realizan para cada nodo de el árbol de expresiones. Cada nodo contiene dos listas y una variable tipo booleano que permite asignar los valores correspondientes. Para dichas funciones, se realiza un recorrido post-order del árbol. Para la función 'nullable' se comprueba si alguno de sus hijos es anulable y se le asigna 'True' o 'False' según el símbolo que se encuentre en el nodo. Las funciones "First" y "Last" se realizan en un mismo método, en donde según sea el símbolo '.', '|', '*', '+' o '?', se realizará una serie de validaciones por medio de operaciones if-else, para saber sus correspondientes first y last.

✓ Función Follow

La tabla de Follows es el último componente necesario para la generación del autómata. Los Follows son listas de números que únicamente pueden pertenecerle a los símbolos terminales o no terminales de la expresión regular, sin embargo, se forman evaluando las hojas internas del árbol (operadores). Por definición del algoritmo, los operadores '|' y '?' son ignorados durante el proceso, dejando únicamente a la concatenación y las dos cerraduras ('*' y '+'). El proceso consiste en recorrer el árbol en postorder y por cada nodo que almacena un operador verificar su símbolo.

Si dicho símbolo operador es una concatenación, se combinan todos los elementos de la lista Last del hijo izquierdo con los elementos de la lista First del hijo derecho, tomando como raíz el nodo donde se encuentra el símbolo. Por otro lado, si el símbolo es cualquiera de las dos cerraduras, la combinación se hace sobre la lista Last y First del hijo izquierdo, el cual es hijo único por tratarse de operaciones unarias. Durante el procedimiento se ignoran números que ya estén repetidos en la lista Follow de cada nodo hoja, además, se ordenan las listas para una mayor facilidad de análisis.

Symbol	Follow
1 (DIGITO)	2,26
2 (DIGITO)	2,26
3 ("")	4
4 (CHARSET)	5
5 ("")	26
6 ("")	7
7 (CHARSET)	8
8 ("")	26
9 ('=')	26
10 ('<')	11
11 ('>')	26

Ilustración 1 *tabla de follows por cada símbolo terminal y no terminal de la expresión regular*

✓ Transiciones de Estados

Las transiciones de estados es la información almacenada en una estructura de las transiciones, con el símbolo especificado, que se pueden realizar de un estado hacia otro. Específicamente se describen todos los estados hacia los que se pueden acceder desde un estado determinado. Esta información es la necesaria el análisis sintáctico de una entrada definida por la gramática inicial.

Para la construcción de las transiciones de estados se requirieron los elementos antes descritos: Árbol de expresiones, First, Last, Nullable y Follow; en especial se hace uso de la información brindada del follow para establecer las transiciones por símbolo y estado. Una transición está conformada por el estado y un listado de símbolos que hacen transición por cada símbolo no terminal disponible en la gramática, de la siguiente forma:

States	DIGITO	'"'	CHARSET	'''
S0 = [1,3,6,9,10,12,13,14,16,18,19,20,22,23]	2,26	4	-	7

El primer estado disponible, y el estado inicial del autómata, corresponde al conjunto de símbolos del resultado de la función first de la raíz del árbol de expresiones. Con el estado inicial se asigna a los símbolos terminales y no terminales de la gramática el follow del símbolo que lo representa, este último símbolo también asignado por el árbol de expresiones y follow.

Tras finalizar esta asignación con todos los símbolos que conforman el estado y la lista de transición de los símbolos terminales/no terminales se analiza todos los símbolos del estado actual en busca de aquellos que cuentan con un listado de transiciones, si dicho listado de transiciones del listado no tiene los mismos símbolos que alguno de los listados de símbolos de los estados quiere decir que se ha generado un nuevo estado. De un estado se pueden generar n estados diferentes. En cambio, si dicho estado ya existe no se realiza ningún otro proceso.

Tras realizar dicho proceso con todos los estados se contará con una tabla de transiciones completas con un número determinado de estados que según un símbolo pueden realizar una transición hacia otro estado. Se considerará que todos aquellos estados que contengan entre sus símbolos el símbolo extendido se dice que son estados de aceptación; lo que significa que haber llegado a este punto significa que la entrada analizada es correcta.

Ilustración 2 *Ejemplificación de una tabla de transiciones descrita de forma gráfica*

States	'a'	'b'	'c'
S0 = [1,3]	1,2	3,4	-
S1 = [1,2]	1,2	6	-
S2 = [3,4]	5,6	3,4	-
S3 = [6]	-	-	-
S4 = [5,6]	-	-	5,6