Universidad Rafael Landívar  
Faculta de Ingeniería  
Ingeniería en Informática y Sistemas  
Lenguajes Formales y Autómatas  
Catedrático: Ing. Moisés Alonso

Primer Entrega Proyecto LFA  
**“Analizador Léxico de Gramática”**

Sergio Daniel Lara Vásquez  
Carne 1044418

Ciudad de Guatemala, 04 de marzo de 2020.

**INTRODUCCIÓN**

En la realización del proyecto se plantearon diferentes expresiones regulares para cada TOKEN en la gramática, gracias al algoritmo de Árbol de Expresión Regular a partir de una expresión regular del Ingeniero Moisés Alonso, decidí crear 4 árboles de expresión, donde cada uno contiene el árbol para los TOKENS, SETS, ACTIONS y ERROR.

Un problema que surgió durante la realización del proyecto fue como puedo recorrer el árbol de expresión teniendo en cuenta las reglas de las expresiones regulares como los operadores unarios y no unarios, el problema fue solucionado gracias al recorrido InOrder por su versatilidad de moverse entre los objetos de un árbol binario así mismo una variable que es mandada como parámetro por referencia la cual llame **“Aceptación”**.

¿Por qué crear una variable por referencia?

Bueno, al necesitar evaluar el nodo actual y poder determinar qué acción realizar teniendo en cuenta también el nodo anterior y no conocer su padre, una variable por referencia es la opción adecuada ya que la pude modificar y consultar en cada nodo utilizando su valor del recorrido un paso antes.

La aplicación fue trabajada en Windows Forms en C# para tener una interfaz más Fácil de usar.

**DESARROLLO**

**Análisis**

1. Entradas:
   1. Archivo .txt con la Gramática a utilizar.
2. Salidas:
   1. Errores en la gramática.
   2. Retroalimentación al usuario.
3. Restricciones:
   1. SETS puede o no venir.
   2. La declaración de cada SET puede ser concatenada con un signo +, \* o dos puntos seguidos o así mismo.
   3. SETS puede poseer la palabra reservada CHAR para indicar que es contenido de un SET
   4. TOKENS siempre debe de existir.
   5. Un token esta conformado por la palabra TOKEN mas un Numero + el signo igual + una Expresión Regular.
   6. ACTIONS debe de existir siempre.
   7. Todas las ACTIONS son funciones las cuales están conformadas por un Identificador más paréntesis luego del identificador, seguido por llaves que indiquen el inicio y fin de las palabras reservadas a utilizar, así mismo estas palabras reservadas poseen una sintaxis conformada por un número, el signo igual y por último la palabra reservada entre comillas simples.
   8. Siempre debe de existir al menos un Error declarado.
   9. Los Errores están conformados por la palabra ERROR mas el signo igual, e igualados a un número entero.
4. Procesos:
   1. Analizar las Expresiones Regulares (ER) ingresadas para luego construir el árbol de expresión.
   2. Construir un árbol de expresión para cada ER.
   3. Recorrer el árbol de expresión para validar que cada carácter de la gramática ingresada cumpla con la ER.

Para el análisis de la expresión regular se tiene un método llamado AnálisisER el cual devuelve una lista de strings los cuales son ingresados y analizados en el árbol de expresión. Para este proyecto cree 5 palabras reservadas las cuales son:

**<id>** Indica que debe de venir un identificador en mayúsculas.

**<PR>** Indica que debe de existir una palabra reservada en mayúsculas sin signos intermedios.

**<t>** Indica que se aproxima un Enter.

**<D>** Indica que la cadena ingresada debe de ser un digito con una longitud menor o igual 3 y mayor o igual a 1

**<C>** Indica que debe de venir en el texto ingresado un carácter, solo uno.

Luego de definir estas palabras reservadas también se analiza si lo ingresado posee operadores unarios en la ER para agregarles de ser necesario el slash invertido si no se necesita su significado en la ER.

Para la construcción del árbol se sigue el pseudo código del Ingeniero Moisés donde cada “token” descrito es su algoritmo representa un elemento de la lista generada por AnalisisER.

Expresiones regulares utilizadas:

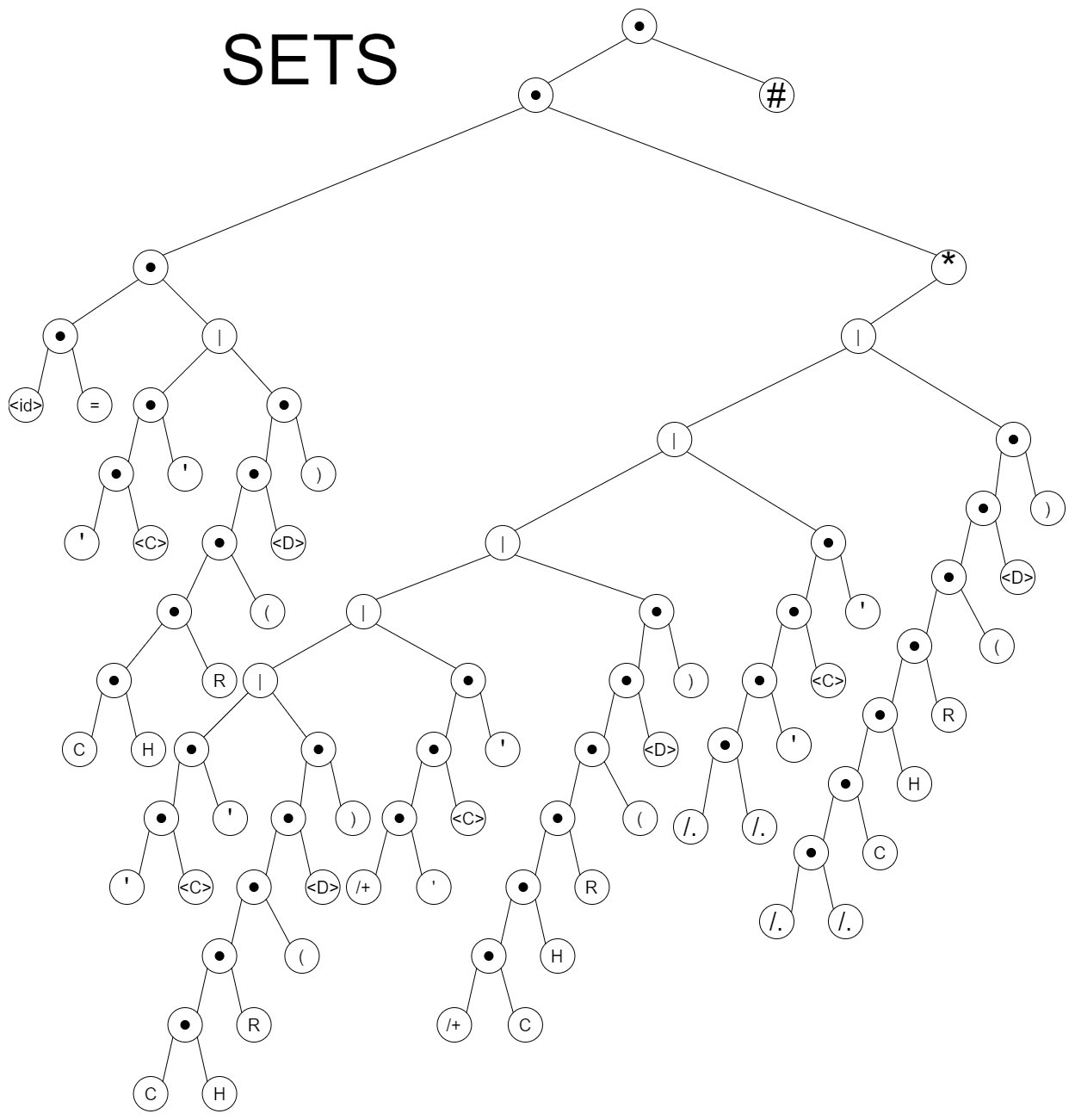
**SETS:** (<id>.=.(('.<C>.')|(C.H.R./(.<D>./))).(('.<C>.')|(C.H.R./(.<D>./))|(/+.'.<C>.')|(/+.C.H.R./(.<D>./))|(/../..'.<C>.')|(/../..C.H.R./(.<D>./)))\*).#

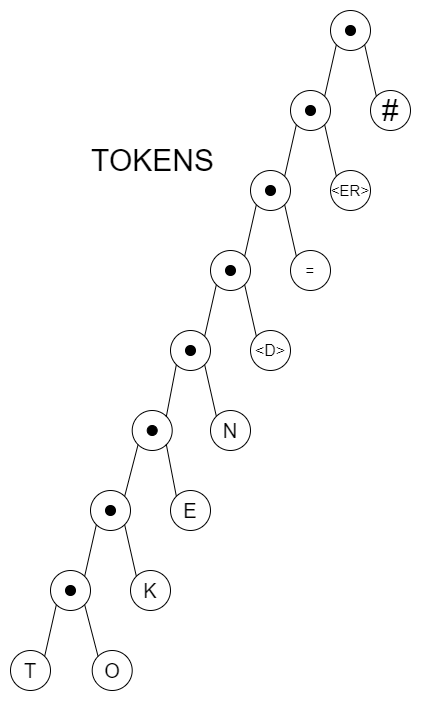
**TOKENS:**   
(T.O.K.E.N.<D>.=.<ER>).#

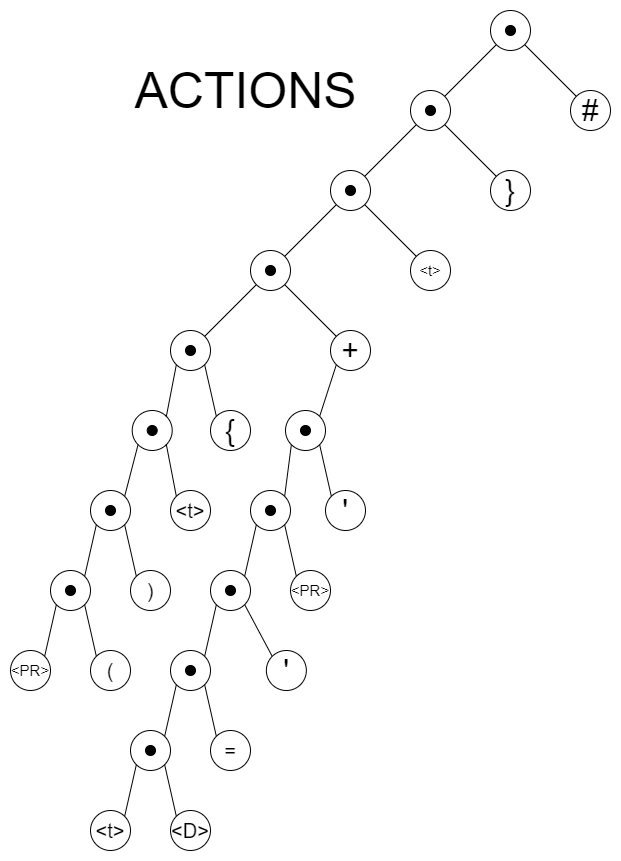
**ACTIONS:**   
(<PR>./(./).<t>.{.(<t>.<D>.=.'.<PR>.')+.<t>.}).#

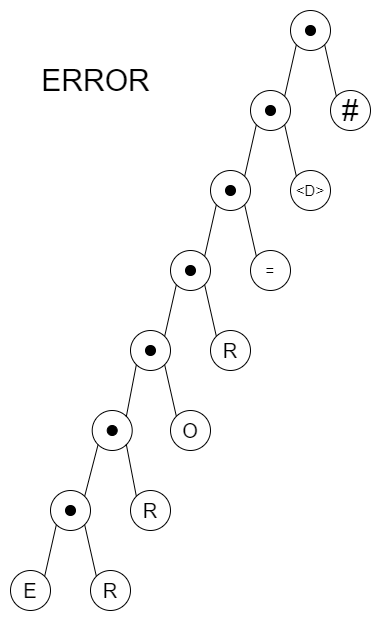
**ERROR:**   
(E.R.R.O.R.=.<D>).#

Arboles generados a partir de las ER:









Luego de tener cada árbol armado tenemos que recorrerlo, pero ¿Cómo? Si lo recorremos estrictamente por InOrder si estamos en un operador | y ya una de las ramas nos dio la respuesta de que es la correcta ya no debemos visitar la otra rama del | o con el \* tenemos que visitar a sus nodos hoja varias veces para esto cree la variable **“Aceptación”** la cual actúa de la siguiente manera sobre los siguientes operadores:

**Nodos hoja:** Cuanto estamos sobre un nodo hoja solo verificamos que el contenido del nodo sea cumplido en el primer char del string que estamos mandando a verificar, por lo tanto, cuando esto se cumple la variable Aceptación toma un valor verdadero, en caso contrario false.

**[ . ]** : Cuando encontramos una concatenación y Aceptación es verdadera significa que el nodo hoja anterior se cumplió las reglas que marcaba su contenido por lo tanto para seguir verificando que toda la cadena concatenada se cumpla verificamos que Aceptación sea verdadera para visitar el nodo derecho, en caso contrario como no se cumplió con una parte de la ER ya no es necesario visitar los demás nodos ya que ya encontramos el primero con un error.

**[ | ]** : Al solo necesitar saber si tenemos que visitar el nodo derecho en caso de que los nodos hoja anteriores no hallan sido cumplidos por la cadena entrante, verificamos con Aceptación, si Aceptación es positiva significa que los subárboles izquierdos cumplieron con la condición de la ER por lo tanto no hay necesidad de visitar los hijos a la derecha de |. Pero si en caso Aceptación fuera negativa significa que ningún subárbol a la izquierda fue cumplido, por lo tanto, entramos al hijo derecho de |.

**[ \* ] y [ + ]** : Ya que el árbol este recorrido en InOrder, siempre que estemos ejecutando el programa visitaremos los hijos izquierdos primero sin importar que, teniendo en cuenta esto todos los subárboles izquierdos se checan al menos una vez, cuando llegamos al [ \* ] o a [ + ] si aceptación es positiva y la longitud del string ingresado aun no ha sido recorrido todo volvemos a recorrer desde el hijo izquierdo del nodo actual. Si no se cumple esto significa que ningún subárbol contiene lo que esta en el string con la cadena a evaluar. En el caso del [ + ] solo verificamos que se halla cumplido Aceptación verdadera al menos una vez.