Universidad Rafael Landívar

Facultad de ingeniería

Ingeniería en Informática y Sistemas

Lenguajes formales y autómatas

Catedrático: Mgtr. Moisés Alonso

# DISEÑO DE APLICACIÓN – GENERADOR DE SCANNER

Nombre y carné:

Marcos Andrés Calderón Morales (1060918)

Guatemala, 06 de mayo del 2020

# Introducción

La programación es el proceso para solucionar problemas a través de la definición de un programa. Esta cuenta con paradigmas, que representan enfoques diferentes para la construcción de soluciones a problemas. También cuenta con metodologías, las cuales son todas las técnicas y conocimientos necesarios para el funcionamiento de un programa informático.

Este proyecto consistió en desarrollar la primera fase de un compilador, consistente en un programa que recibe como entrada un archivo de texto y produce una salida compuesta de tokens (componentes léxicos) o símbolos. Específicamente, en esta tercer fase se procedió a la generación de programas que a su vez, estos programas sean capaces de reconocer lenguajes formales. Esto se realizó mediante la generación de una máquina aceptadora, también conocida como “autómata”, el cual lee si una cadena es aceptada por un lenguaje o no.

En base a ello, se ha desarrollado una simulación del analizador, en el que cada archivo que analiza está formado de:

1. SETS
   1. La palabra “SETS” debe estar en mayúscula.
   2. Pueden estar concatenados a través del signo “+”.
   3. Se puede utilizar la función CHR
   4. Contiene la definición abreviada de un conjunto de símbolos terminales.
   5. Puede haber muchos espacios en blanco entre el identificador, el símbolo “=” y la definición
   6. Esta parte puede o no venir.
      1. si aparece, debe poseer al menos un SET.
2. TOKENS
   1. Representan los símbolos terminales y no terminales de la gramática.
   2. La palabra TOKENS debe existir y estar en mayúscula.
   3. Esta sección debe existir.
   4. Cada token debe poseer la palabra: TOKEN y un número, seguido del signo igual “=”.

Para la fase II, la cual es la creación del analizador sintáctico, tiene como objetivo el análisis de la sección de tokens. Una vez ya revisada correctamente la sección de tokens, se procede a:

* Obtener toda la expresión de tokens como una sola cadena
* Esta cadena alimenta nuestro algoritmo para transformarla en un árbol de expresión.
* Seguido, se procede a la construcción de la tabla de first, last y anulable. Por lo que se realiza un recorrido post orden para determinar los valores de cada nodo. Ya tenida la primer tabla, se procede a generar la tabla de Follow, en la cual (dependiendo del operador que se encuentre en el nodo), se le asigna sus follow. Por último, se procede a generar la tabla de transiciones, en donde, partimos con los first de la raíz como estado inicial, y verificamos las transiciones con todos los símbolos terminales con los que encuentra la expresión. Al generar las transiciones, verificamos cada una, y miramos que exista previamente en el conjunto de estados, para así, de esta manera, agregarla y repetir el mismo proceso hasta que ya no encuentren nuevos estados. De esta manera, generamos nuestro autómata finito determinista por medio del método directo, en donde, a partir de una expresión regular, se forma el AFD.

Seguido de lo realizado en la fase III, se procedió a realizar el siguiente procedimiento:

* Se procedió a conectar las transiciones con sus respectivos estados, los cuales fueron obtenidos en la fase II.
* Seguidamente, se procedió a generar un algoritmo, que basado el archivo de entrada ingresado por el usuario, este genera otro programa donde va definido las reglas gramaticales para el reconocedor de lenguaje creado. En el algoritmo, se definen la sección de SETS mediante condiciones, los cuales permitirán utilizarse como transiciones para moverse de estados.
* La finalidad de esta fase es admitir o no la cadena de entrada, la cual si es permitida, debe mostrar como Salida todos los componentes léxicos

1. ACTIONS
   1. Contiene definición de funciones, en este caso específico las palabras reservadas del lenguaje.
   2. Reservadas() siempre debe existir.
   3. Puede haber otras funciones.
2. ERROR
   1. Debe venir al menos uno,
   2. ERROR debe tener asignado un número.
   3. El identificador debe tener como sufijo la palabra ERROR en mayúscula

Si alguna de estas características no las trae consigo el archivo de texto, el resultado deberá ser indicándole al usuario que existe un error y la posición del primer error.

De lo contrario, salida será un indicador mostrando que el archivo no contiene ningún error de gramática.

# OBJETIVOS

## GENERAL

Generar un programa que sea capaz de reconocer un lenguaje mediante un archivo de entrada.

## ESPECIFICOS

1. Generar el autómata finito mediante la tabla de estados y transiciones (generadas en la FASE II)
2. Crear una solución que tenga definida una gramática, mediante un archivo de entrada.
3. Indicar si la cadena ingresada al programa es aceptada por el lenguaje regular.
4. Mostrar componentes léxicos, basado en la sección de TOKENS.

# Expresiones regulares

SETS

Probada en Regex101: \*([A-Za-z])+ \*= \*('[A-Za-z]'|CHR\([0-9]+\))((..|\+)('[A-Za-z]'|CHR\([0-9]+\)))\*

En el programa: ( \*.L+. \*.=. \*.(('.S.')|(C.H.R.\\(.N+.\\))).(((\\..\\.)|\\+).(('.S.')|(C.H.R.\\(.N+.\\))). \*)\*)

TOKENS

ANALIZADOR LÉXICO:

* Probada en Regex101: \*TOKEN +[0-9]+ \*= \*(('?.'?|[A-Z]+) \*)+
* En el programa: ( \*.T.O.K.E.N.Z\*.N+.Z\*.=.Z\*.((('.S.')|S+). \*)+)

ANALIZADOR SINTÁCTICO (basado en el archivo “GRAMATICA.txt”):

* Inicialmente: DIGITO DIGITO \* | '"' CHARSET '"' | ''' CHARSET ''' | '=' | '<' '>' | '<' | '>' | '>' '=' | '<' '=' | '+' | '-' | 'O' 'R' | '\*' | 'A' 'N' 'D' | 'M' 'O' 'D' | 'D' 'I' 'V' | 'N' 'O' 'T' | '(' '\*' | '\*' ')' | ';' | '.' | '{' | '}' | '(' | ')' | '[' | ']' | '.' '.' | ':' | ',' | ':' '=' | LETRA ( LETRA | DIGITO ) \*
* En el programa: ((DIGITO.DIGITO\*)|(\".CHARSET.\")|('.CHARSET.')|(=)|(<.>)|(<)|(>)|(>.=)|(<.=)|(\\+)|(-)|(O.R)|(\\\*)|(A.N.D)|(M.O.D)|(D.I.V)|(N.O.T)|(\\(.\\\*)|(\\\*.\\))|(;)|(\\.)|({)|(})|(\\()|(\\))|([)|(])|(\\..\\.)|(:)|(,)|(:.=)|(LETRA.(LETRA|DIGITO)\*).#)

ACTIONS

Probada en Regex101: \*TOKEN +[0-9]+ \*= \*(('?.'?|[A-Z]+) \*)+

En el programa: ( \*.N+. \*.=. \*.'.L+.')

ERROR

Probada en Regex101: \*TOKEN +[0-9]+ \*= \*(('?.'?|[A-Z]+) \*)+

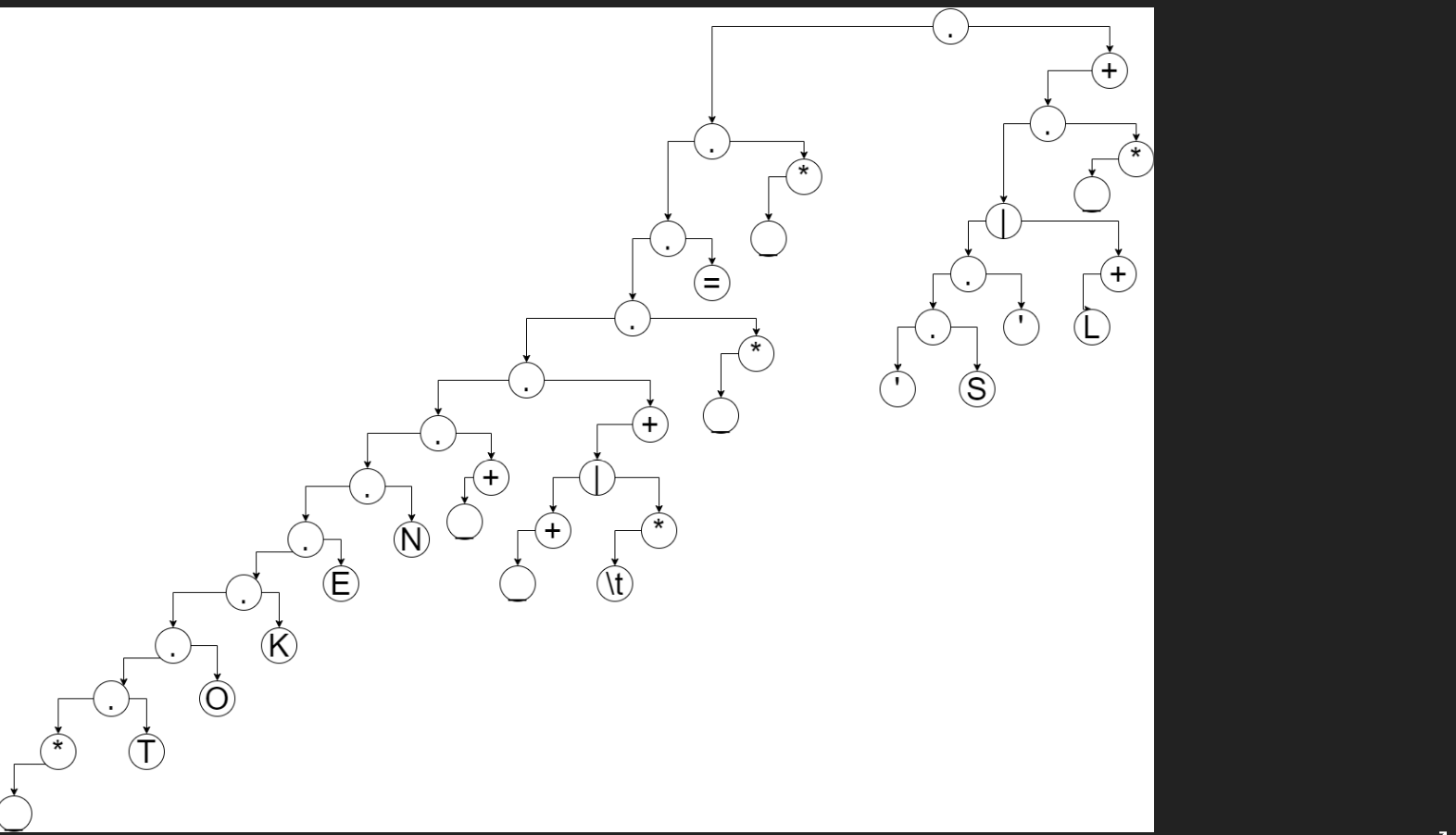
En el programa: ( \*.E.R.R.O.R. \*.=. \*.N+)

# Árboles de expresión

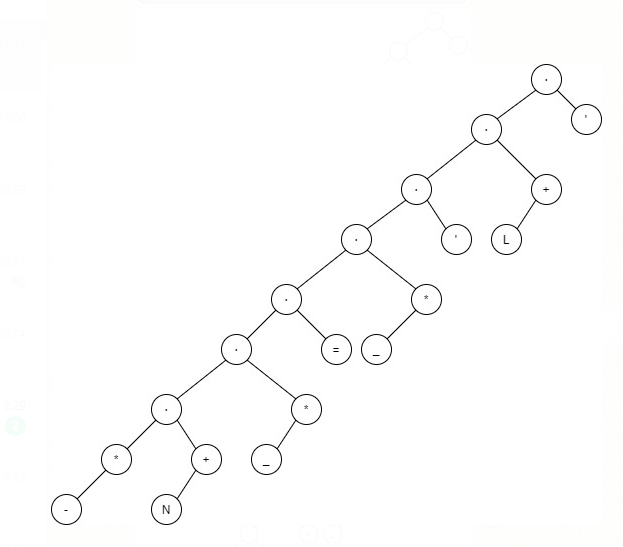
SETS

# 

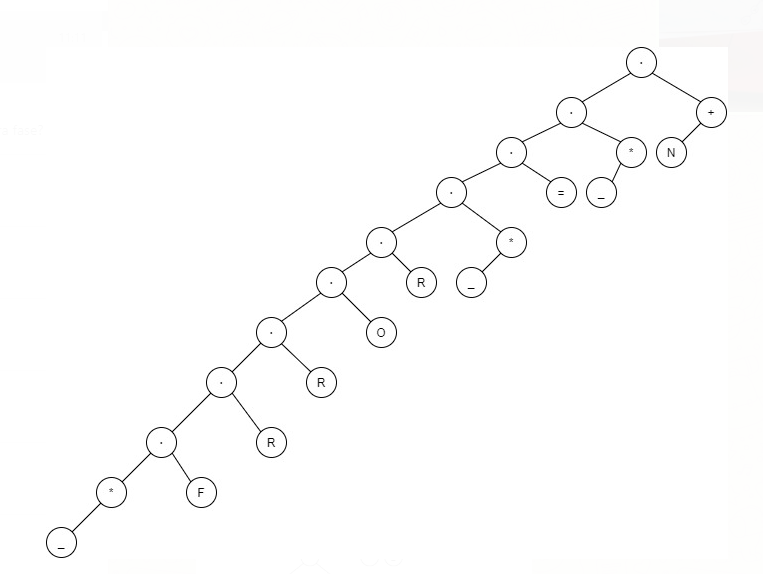
TOKENS



ACTIONS



ERROR



# TABLAS

TABLA No.1: FIRST, LAST Y ANULABLE

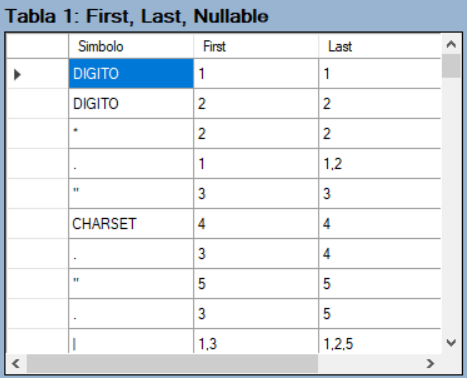


TABLA NO.2: FOLLOWS

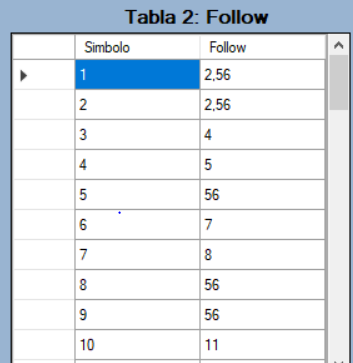
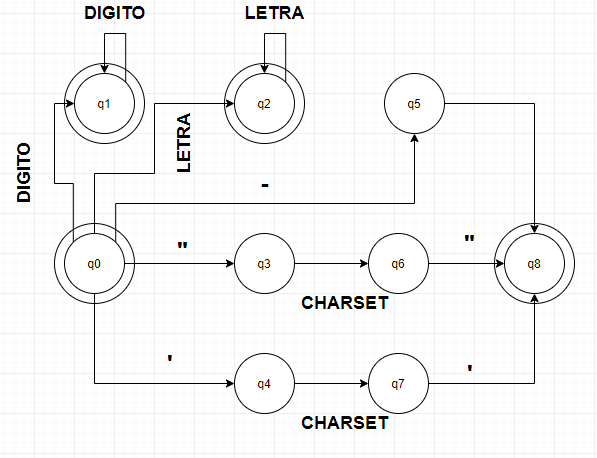


TABLA NO.3: ESTADOS Y TRANSICIONES



RAZONAMIENTO UTILIZADO



Automata basado en el archivo de entrada “Gramática – prueba 4.txt” en el cual fue utilizado de prueba para aplicar el razonamiento sobre el desarrollo de la fase III. Dando como resultados:

