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

Tercera Entrega Proyecto LFA

“Generador de Scanner”

Sergio Daniel Lara Vásquez
Carne 1044418

Ciudad de Guatemala, 05 de mayo de 2020.

DESARROLLO

Análisis

a) Entradas:
   a. Archivo .txt con la Gramática a utilizar.

b) Salidas:
   a. Solución compilada en C# con un autómata basado en los Tokens de la gramática ingresada.

c) Restricciones:
   a. No se debe generar la solución con la gramática errónea

d) Procesos:
   a. Luego de la verificación de la gramática se procede a crear un objeto Estado.
   b. A cada objeto estado esta compuesto por un diccionario de tipo <string, string> donde la llave carácter que genera una transición en ese estado y el valor es el Estado a ir nuevamente, también un diccionario tipo <string, int> el cual utiliza como llave un símbolo char y en su valor el numero de token al que pertenece el estado actual dependiendo del char ingresado de la cadena a evaluar.
   c. Luego de determinar a que Estado se debe mover el programa este identifica sobre que tokens trabaja el Estado actual y le muestra al usuario este numero junto con la cadena ingresada.

Tomando en cuenta todo el procedimiento realizado en las 2 entregas posteriores, en esta se genera un autómata complemente, esto escribiendo un código en texto plano hacia un archivo .cs en otra solución el cual genera el autómata en otra solución.

¿Pero como funciona el auto mata?

Ha este necesitamos la definición de todos los SET y ACTIONS, así mismo como los objetos nuevos creados que es Estado, ha estado le añadimos dos funciones de tipo string la cual nos devuelve las transiciones y token con un respectivo char en cada estado.

Todo el autómata se basa en la función Analizar (), la cual contiene un parámetro A de tipo string, otro string E, otro string Tk y una variable booleana llamada W. La variable A nos indica la cadena de entrada que nos ingresa el usuario, luego E es el estado sobre el que estamos trabajando, Tk el cual nos indica sobre que token estaba trabajando el estado anterior al que estamos y W que es una variable utilizada como verificación de un proceso interno.

```
public bool Analizador (ref string A, string E, string Tk, bool W)
{
	Estado T;
	if (E == null) { return false; }
	bool B = Estados.TryGetValue(E, out T);
	string ENuevo = "";
	if (Tk.Length <= 0) { ENuevo = null; }
	else
	{
		ENuevo = T.getTrancicion(Tk);
		string A2 = A;
		string AB2 = "";
		bool Work = false;
		try { A2 = A2.Substring(1, A2.Length - 1); } catch { }
		if (A2.Length > 0)
		{
			Work = Analizador(ref A2, ENuevo, A2[0].ToString(), true);
		}
		else
		{
			Work = Analizador(ref A2, ENuevo, "", true); }
			if (ENuevo == null || !Work)
			{
				foreach (var item in ListaSets)
				{
					if (item.Pertenece(Tk[0]) && Tk.Length == 1)
					{
						Tk = "<" + item.ID + ">";
						ENuevo = T.getTrancicion(Tk);
						break;
					}
				}
			}
		}
		if (ENuevo != null)
		{
			Token = T.getTokenTransicion(Tk);
			return Analizador(ref A, ENuevo, "", W); }
		}
		else
		{
			if (EAceptacion.Contains(E))
			{
				return true;
			}
			else
			{
				return false;
			}
		}
	}
```

En base al código anterior, se explica que es una función recursiva de tipo bool, la cual en base al estado que se le es mandada realiza operaciones. Primero utiliza el primer carácter de la cadena ingresada y busca las tranciones y token pertenecienes del estado con ese carácter para generar el nombre del siguiente estado, si no logra generar nada produce null, cuando logra generar un nuevo estado verifica que con el siguiente token la cadena valla a terminar en un estado de aceptación o sino pasa a revisar si en los SETs se encuentra algún no valido con el cual mandar ahora la función con un nuevo Tk en ves del primer carácter.

Cuando el programa ya determino si se puede o no utilizar un Estado analiza si la cadena del estado es nula, si lo es verifica si se encuentra en un estado de aceptación y devuelve verdadero, si no falso.

SET DE PRUEBAS

Archivos utilizados

ARCHIVO1:

```
SETS
	LETRA='A'..'Z'+'a'..'z'+'_'
	DIGITO='0'..'9'
TOKENS
	TOKEN 1=DIGITO DIGITO *
	TOKEN 2='='
	TOKEN 3=':''='
	TOKEN 5='A''N''D'
	TOKEN 4=LETRA(LETRA|DIGITO)* {RESERVADAS()}
ACTIONS
RESERVADAS()
{
	5='PROGRAM'
	6='INCLUDE'
	7='CONST'
	8='TYPE'
}
ERROR=9
```

ARCHIVO2:

```
SETS
	LETRA   = 'A'..'Z'+'a'..'z'+'_'
TOKENS
	TOKEN  5  = '<''>'
	TOKEN  6  = '<'
	TOKEN  7  = '>'
	TOKEN  8  = '>''='
	TOKEN  9  = '<''='
	TOKEN  43  = '.'
	TOKEN  50  = '.''.'
	TOKEN  51  = ':'
	TOKEN  53  = ':''='
	TOKEN 3= LETRA*  { RESERVADAS() }
ACTIONS
RESERVADAS()
{
	18 = 'PROGRAM'
	19 = 'INCLUDE'
}
ERROR = 5
```

Expresiones regulares utilizadas generadas a partir de los tokens

ARCHIVO1:

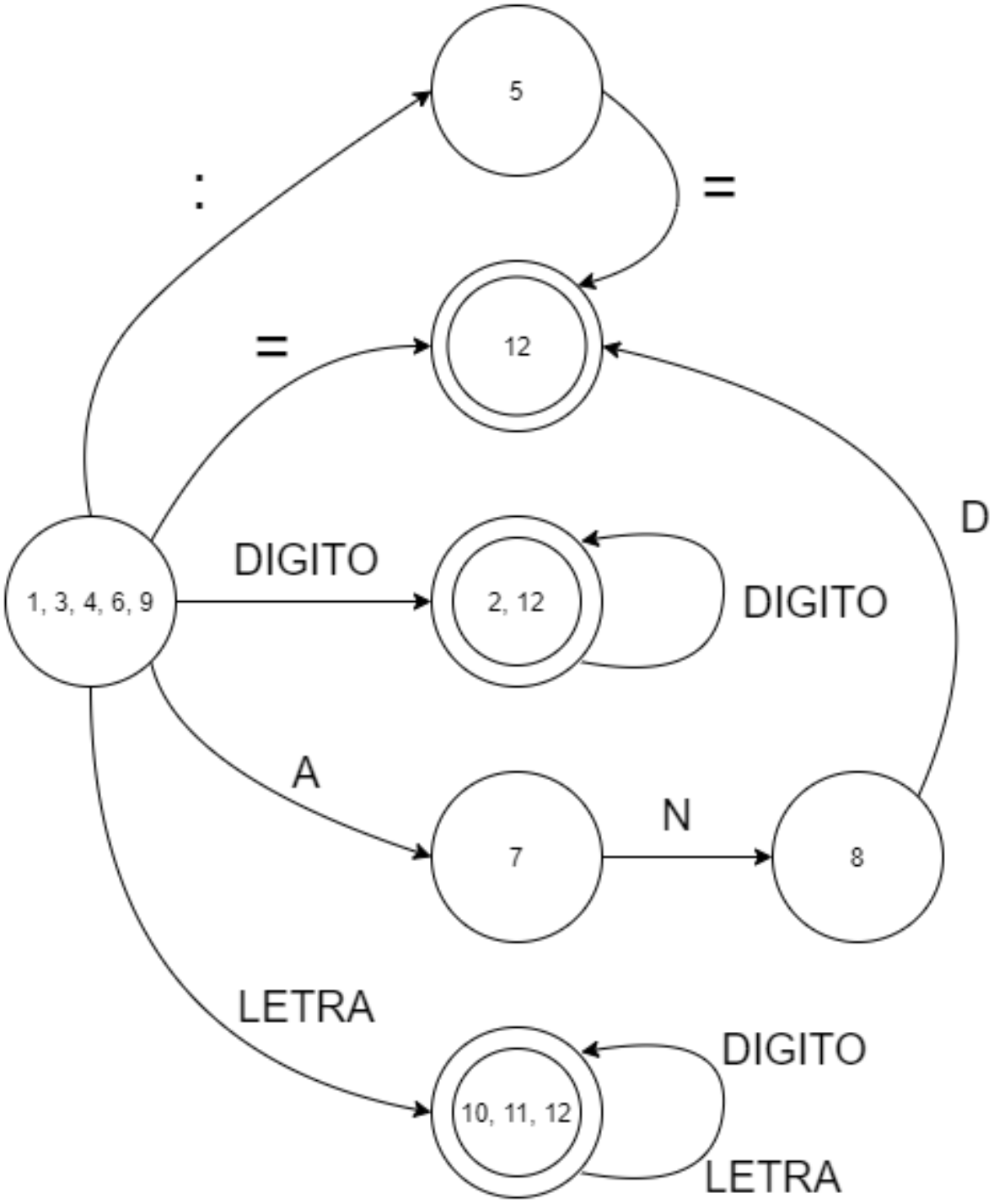
```
((DIGITO DIGITO)*|=|:=|AND|LETRA(LETRA|DIGITO)*)#
```

ARCHIVO2:

```
(<>|<|>|>=|<=|/.|/./.|:|:=|LETRA*)#
```

Autómatas generados

ARCHIVO1:



ARCHIVO2: