Uso del código generado en SP-PS1 para una aplicación Windows C# que efectúe un análisis léxico.

Ing. Francisco Ríos Acosta
Instituto Tecnológico de la Laguna
Torreón,Coah; a 17 de diciembre del 2007.

I N D I C E.

1. Aplicación Windows C#, analizador léxico para reconocer los tokens id y opasig.

La idea es construir una aplicación Windows C#, que utilice el código generado en C# por el programa SP-PS1.
Esta aplicación consiste de una interfase gráfica de usuario semejante a la vista de la simulación del analizador léxico usando SP-PS1. La aplicación *Form1* en ejecución es mostrada en la fig.#1.1.

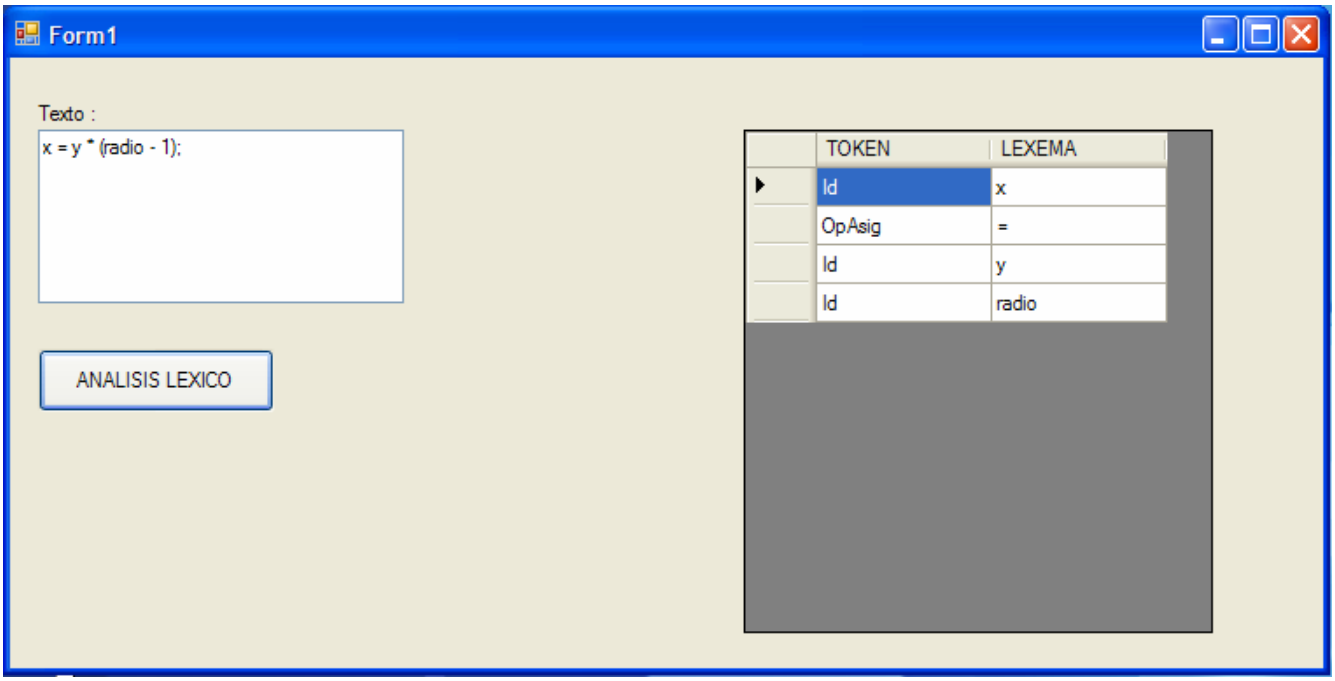

Fig. No. 1.1 Aplicación Windows C# que analiza léxicamente la entrada *x=y*(radio – 1);*.

En la figura 1.1 sólo son reconocidos los tokens *Id* y *OpAsig*. Es decir, el analizador léxico sólo reconoce a estos 2 tipos de tokens.
Cuando el usuario teclea un texto de entrada y hace click sobre el botón con leyenda "ANALISIS LEXICO", la aplicación muestra a las parejas TOKEN-LEXEMA en el componente *DataGridView* que fueron reconocidas durante el análisis léxico.

> Veamos y sigamos cada uno de los pasos para construir la aplicación Windows C# visto en la figura #1.1.

PASO 1.- Creación del directorio que contiene al programa *SP-PS1*.

Usaremos el programa *SP-PS1* para generar los AFD óptimos o reducidos que reconocen a los tokens Id y OpAsig.

Lo primero que debemos hacer es crear un directorio de trabajo que albergue al programa *SP-PS1*. El directorio de trabajo llámalo como gustes. En él deberás copiar los archivos :

- ps1.exe
- Codigo1.txt
- Codigo2.txt
- Codigo3.txt
- Codigo4.txt
- Codigo5.txt
- Codigo6.txt

Luego ejeucta el programa *ps1.exe* que te responde con la aplicación mostrada en la figura 1.2.

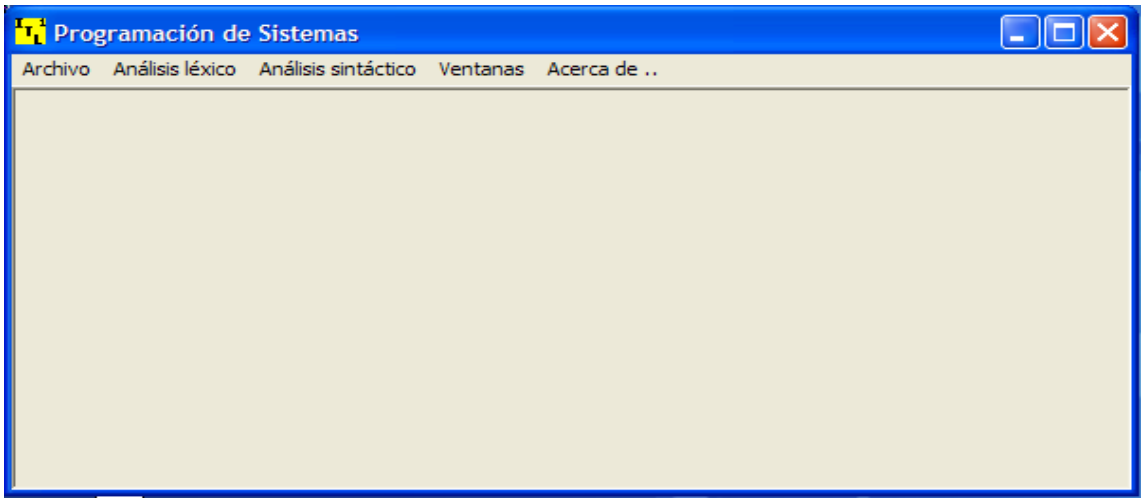

Fig. No. 1.2 Interfase gráfica del programa *SP-PS1*.

PASO 2.- Generación del AFND usando las reglas de Thompson, que reconoce al token *Id*.

- Entremos al menú *Archivo* | *Nuevo* e ingresemos la expresión regular vista en la figura #1.3.

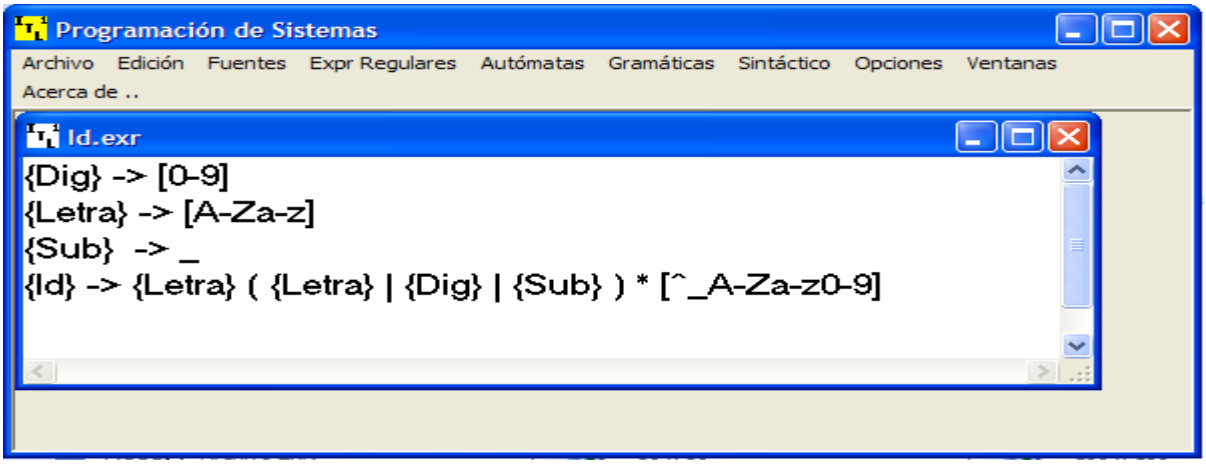

Fig. No. 1.3 Edición de la expresión regular *Id.exr.*

- Luego salvemos la expresión regular usando *Archivo* | *Salvar Como* con el nombre "*Id.exr*".

- Usemos *Expr Regulares* | *Compilar* para saber si hemos incurrido en errores. Cuando no tenemos errores en nuestra expresión regular, la aplicación muestra una caja de mesaje donde nos los hace saber, figura #1.4.

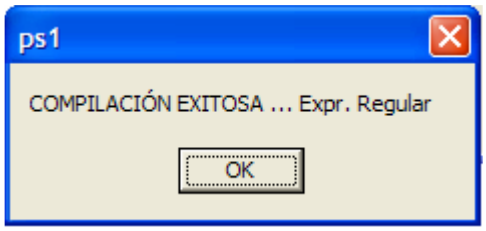

Fig. No. 1.4 Expresión regular *Id.exr*, compilada de forma exitosa.

- Ahora aplicamos las reglas de *Thompson* a la expresión regular que acabamos de compilar. Usemos la trayectoria del menú de SP-PS1 *Automatas* | *Thompson (AFND)* y observemos el AFND construido a partir de la expresión regular *Id.exr*, fig. #1.5.

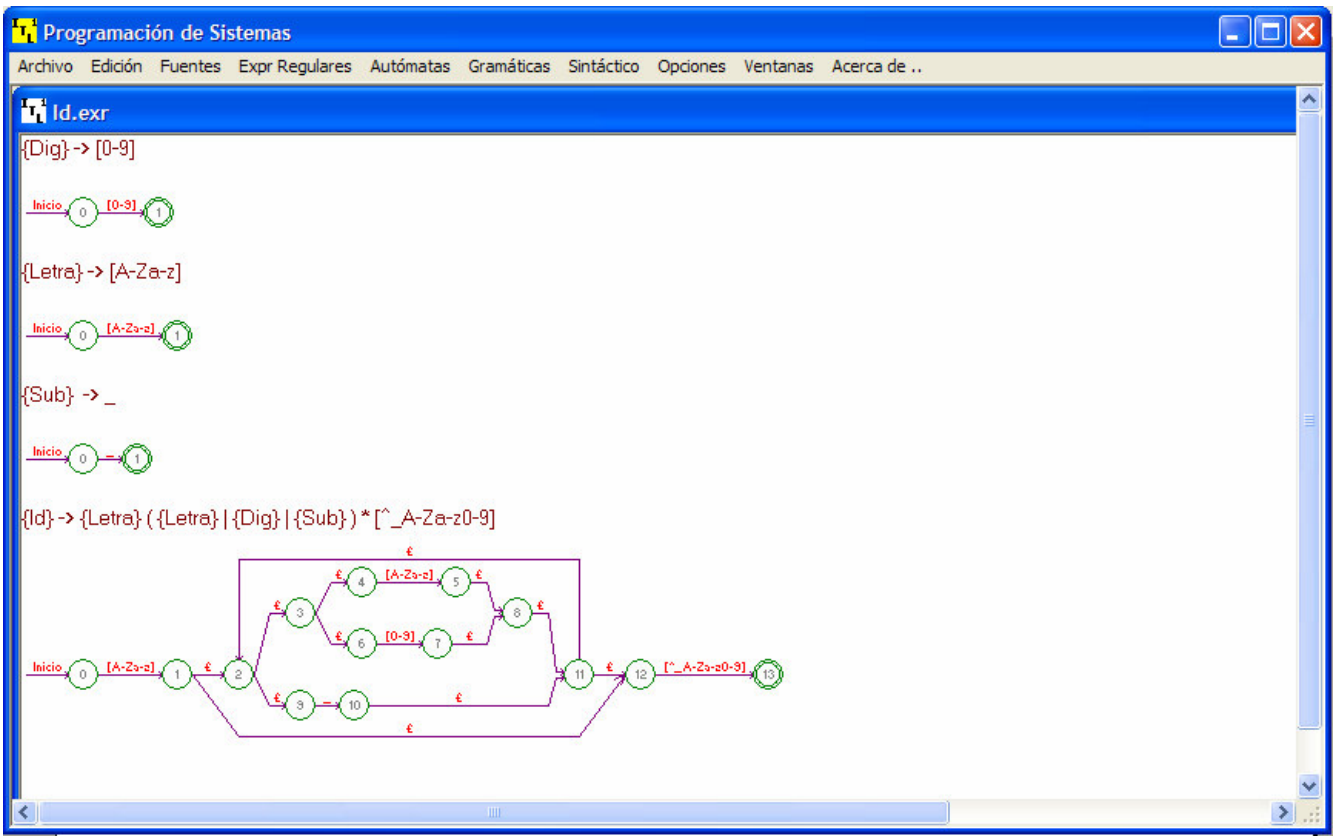

Fig. No. 1.5 AFND correspondiente a la expresión regular *Id.exr*.

Observemos en la fig#1.5, que el AFND reconoce leyendo un caracter que no forma parte del lexema encontrado (transición del estado 12 al estado 13 de aceptación).

PASO 3.- Generación del *AFD* –algoritmo de construcción de subgrupos- a partir del *AFND* obtenido en el paso 2.

- La opción del menú para construir el AFD que reconoce al token *Id* es : *Automatas* | *Subgrupos (AFD)*.
- *SP-PS1* responde con una nueva ventana, fig#1.6.

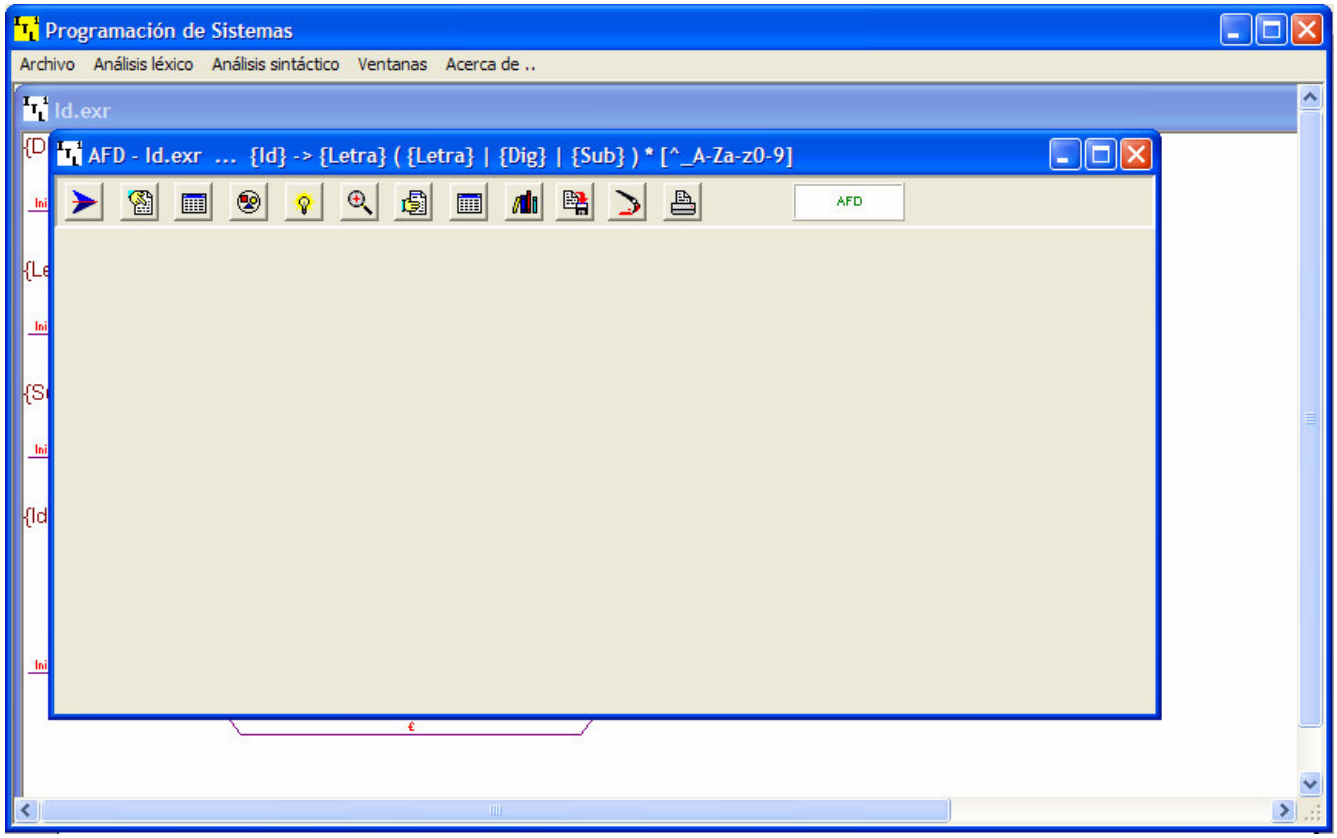

Fig. No. 1.6 Interfase de usuario para la obtención del AFD según el algoritmo de construcción de subgrupos.

- El primer botón con la imagen de la punta de una flecha, es el que sirve para visualizar al autómata finito determinístico AFD construido a partir del AFND obtenido al aplicar las reglas de Thompson.
- Este AFD puede que no sea el óptimo, por lo que debemos de aplicarle un algoritmo llamado de *particiones*.
- Antes de obtener el AFD óptimo o reducido, observemos la fig#1.7 que muestra el AFD obtenido con el algoritmo de construcción de subgrupos.

PASO 4.- Generación del AFD óptimo para el token *Id*.

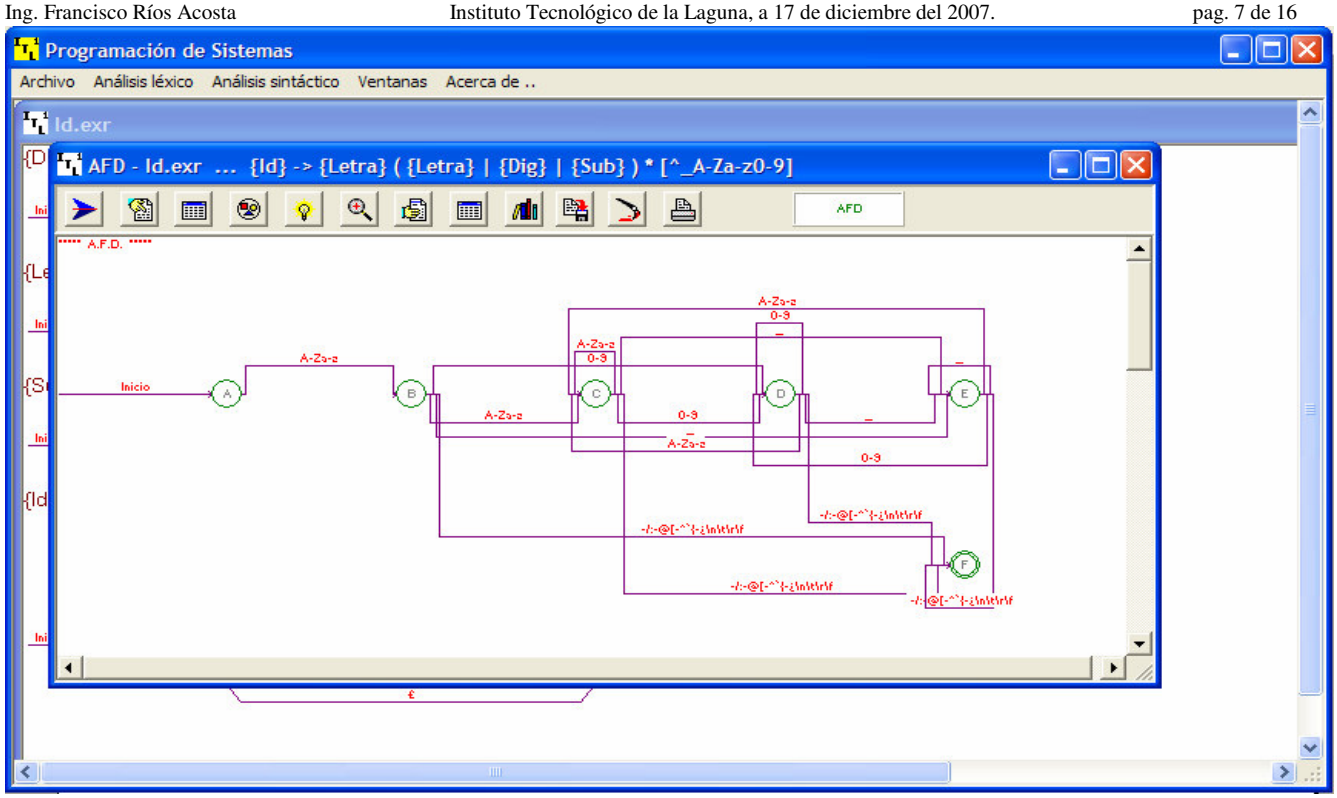

Fig. No. 1.7 AFD que reconoce al token *Id*.

- En esta etapa haremos 2 cosas : primero obtenemos el AFD óptimo o reducido, para luego salvar toda su información.
- Hagamos un click sobre el botón con la imagen del zoom. El AFD óptimo es visualizado por el programa SP-PS1, fig#1.8.

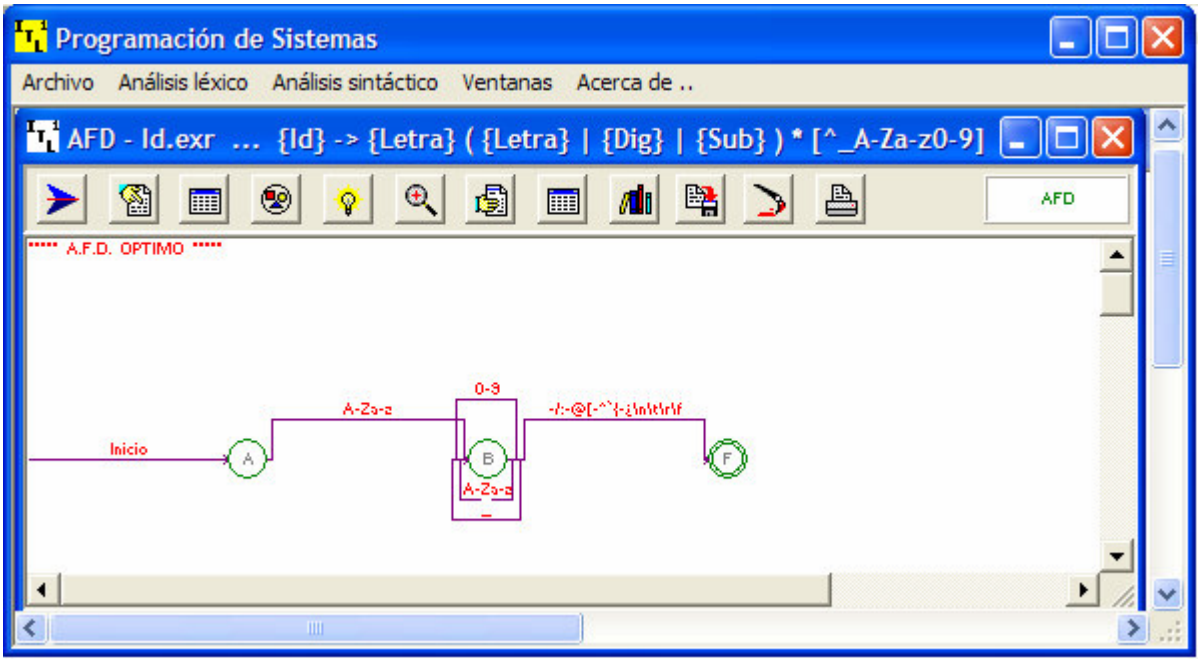

Fig. No. 1.8 AFD óptimo que reconoce al token *Id*.

- Con el fin de poder generar el analizador léxico en etapas posteriores, debemos salvar la información de este AFD reducido. Hagamos un click sobre el botón con una imagen de un diskette. A la pregunta de SI SALVAMOS EL AFD REDUCIDO contestemos que sí, fig#1.9.

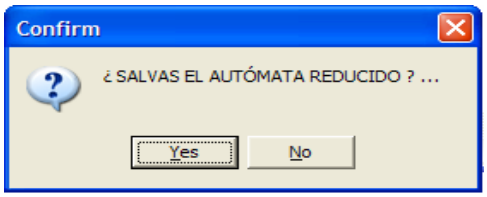

Fig. No. 1.9 Almacenando la información del AFD óptimo.

PASO 5.- Generación del AFD óptimo para el token *OpAsig*.

- Seguimos los mismos pasos utilizados para obtener el AFD óptimo del token *Id* (pasos 2,3 y 4).

PASO 6.- Selección de autómatas y ensamble del analizador léxico que reconoce a los tokens *Id* y *OpAsig*.

- Cierra las ventanas abiertas menos la principal de *SP-PS1*. Selecciona el menú *Analisis lexico* | *Construccion de analizadores lexicos*, y verás la interfase de la fig#1.10.

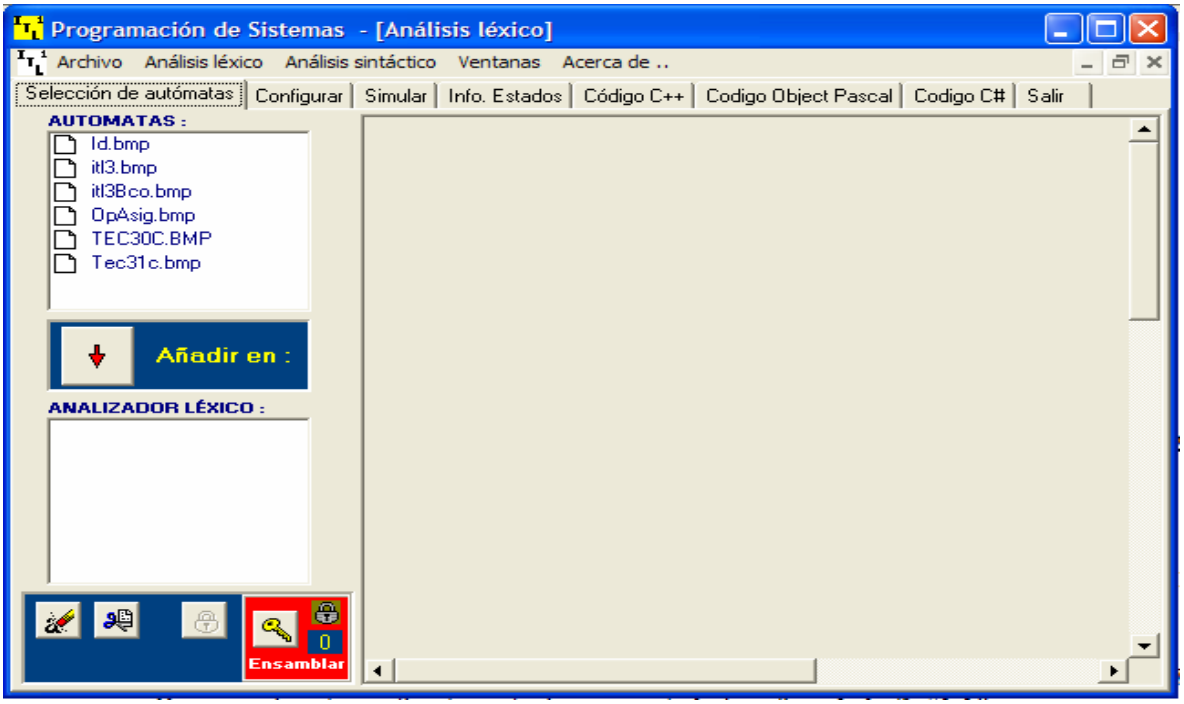

Fig. No. 1.10 Selección de autómatas para el analizador léxico a construir.

- De la ventana AUTOMATAS selecciona los autómatas *Id.bmp* y *OpAsig.bmp*. Agregalos con el botón "*Añadir en :*" al analizador léxico.
- La ventana ANALIZADOR LEXICO contiene ahora a los dos autómatas sleccionados, fig#1.11.

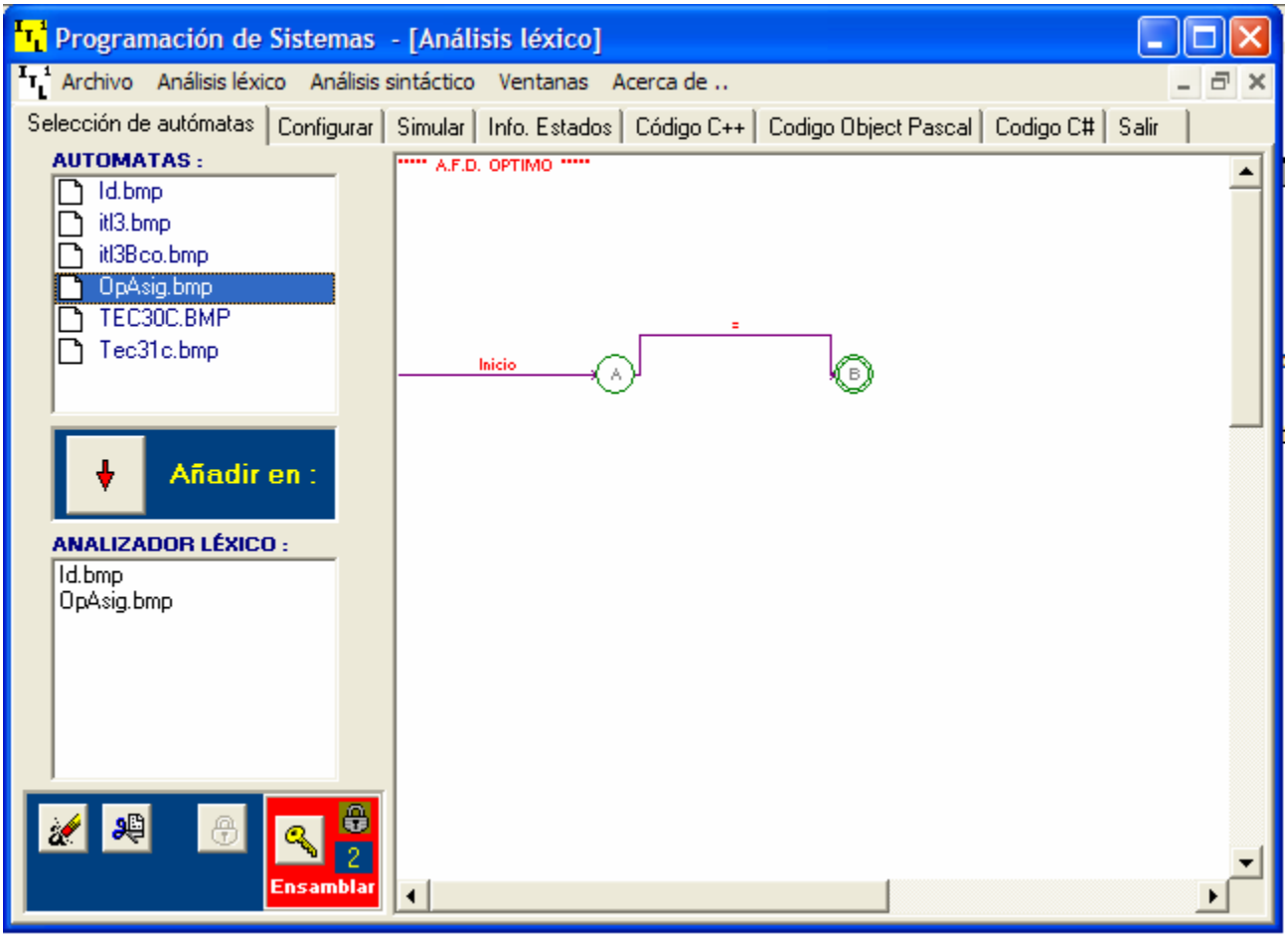

Fig. No. 1.11 Autómatas *Id* y *OpAsig* listos para el registro del ensamble del analizador léxico.

- A este instante, estamos listos para ensamblar el analizador léxico. Por ensamble, entendemos que los autómatas seleccionados y su orden, formarán un analizador léxico que reconocerá sólo al lenguaje de los tokens añadidos.
- Haz click en el botón con la imagen de una llave amarilla y contesta que sí, a la ventana que pregunta por la confirmación del ensamble, fig#1.12.

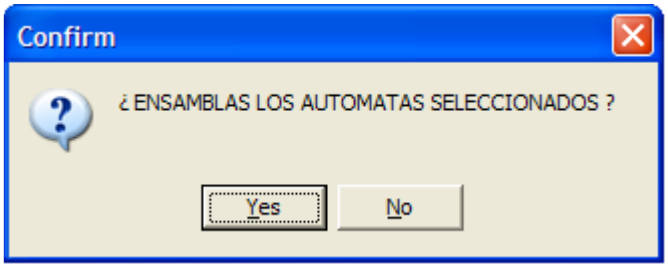

Fig. No. 1.12 Pregunta por la confirmación del ensamble de autómatas seleccionados.

PASO 7.- Configuración del *Retraer()* en AFD's.

- En el caso del autómata para el token *Id*, se requiere de retraer el puntero al caracter i-ésimo menos 1. Esto sucede cuando es necesario leer otro caracter que no forma parte del lexema respecto al token que se reconoce.
- *SP-PS1* permite configurar el caso cuando un autómata requiera de la función *Retraer()*. Seleccionemos la opción *Configurar* y hagamos doble click para configurar el *Retraer()* para el autómata *Id*, fig#1.13.

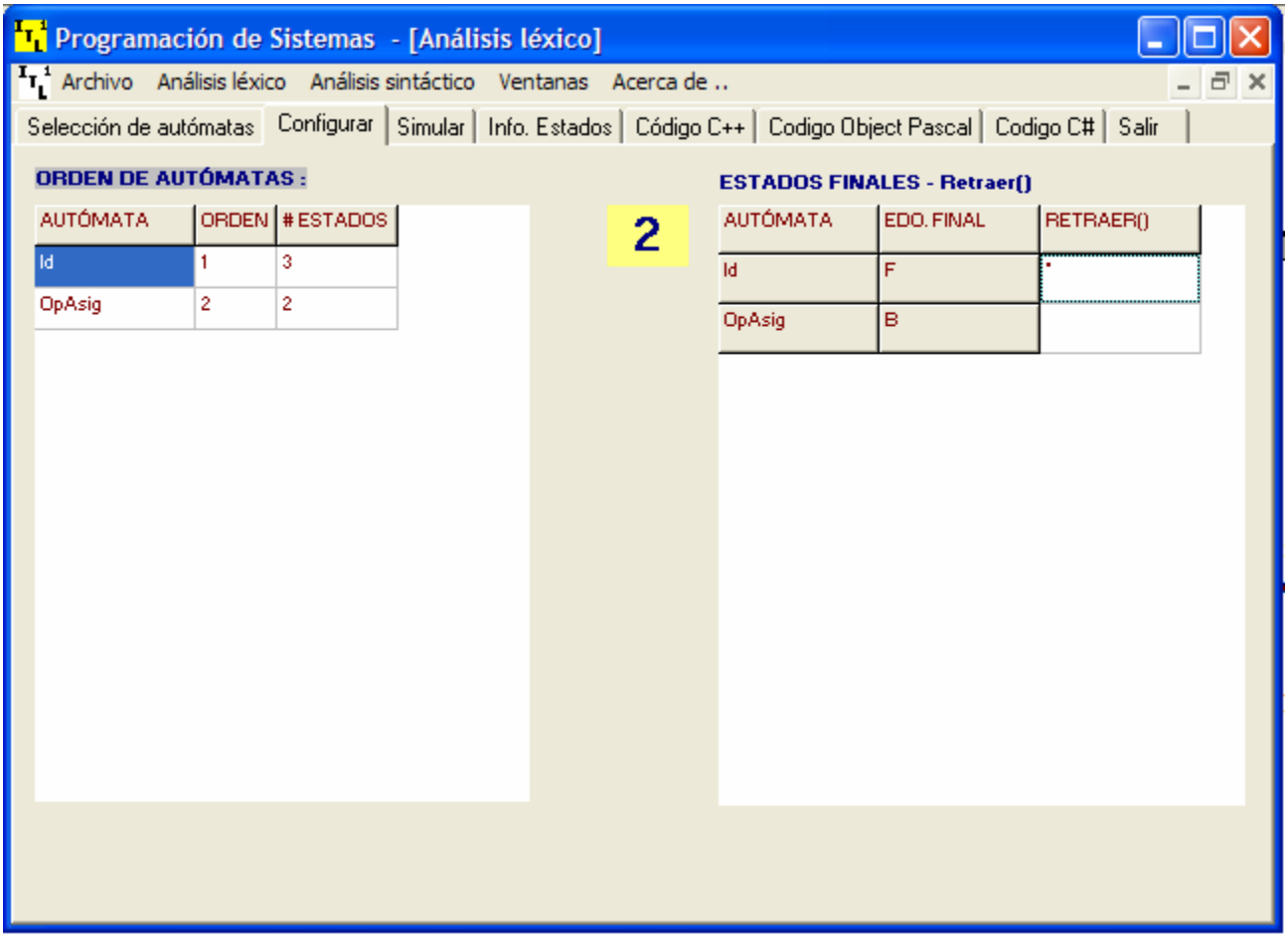

Fig. No. 1.13 Configuración del *Retraer()* para un cierto autómata.

PASO 8.- Generación de código C# para el analizador léxico que reconoce los tokens *Id* y *OpAsig*.

- Una vez que hicimos las inicializaciones de rigor, podemos obtener el código C# para las 2 clases que propongo. El libro del drágón contiene toda la teoría acerca de un análisis léxico. Mi propuesta de clases se soporta sobre lo escrito en el libro del dragón acerca del tema.

- Precisamente la generación de código C# por parte del programa SP-PS1, consiste de 2 clases : class *Automata* y class *Lexico*.
- Hagamos un click sobre la pestaña Codigo C# y luego otro click pero ahora sobre el botón que genera el código –imagen de lápiz-. La salida es el código en las 2 ventanas : una para la clase *Automata* y otra para la clase *Lexico,* figura #1.14.

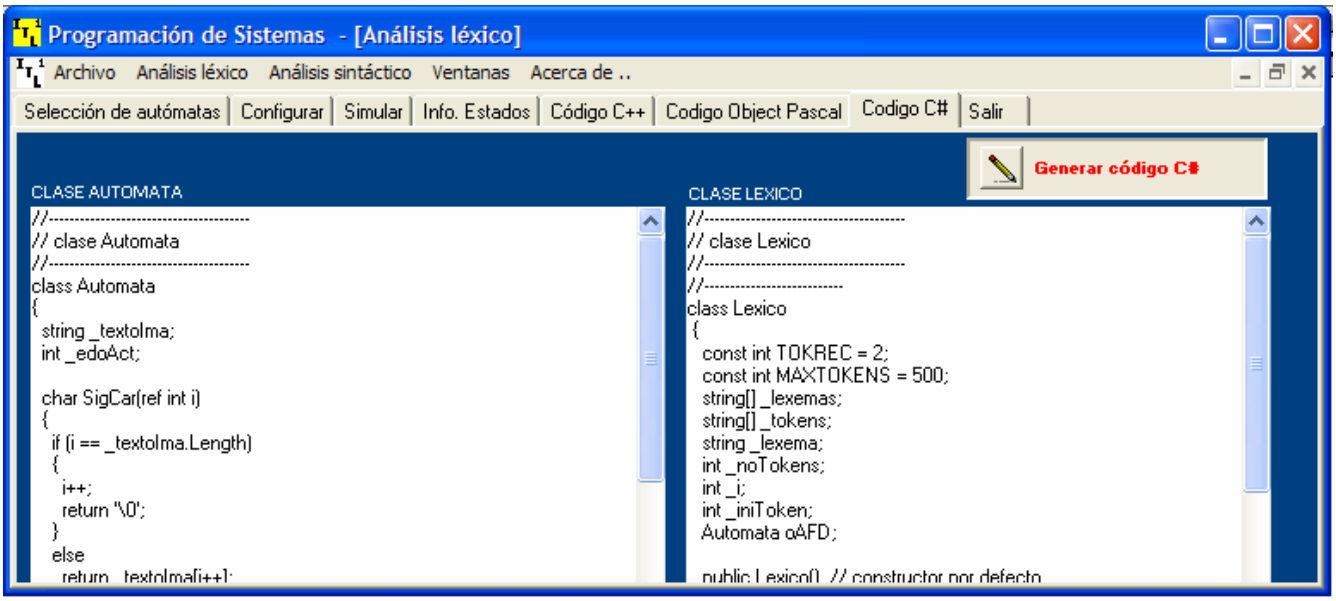

Fig. No. 1.14 Generación del código C# para las clases *Automata* y *Lexico*, referente al analizador léxico para reconocer los tokens *Id* y *OpAsig*.

CLASE AUTOMATA

```
//---------------------------------------
// clase Automata
//---------------------------------------
class Automata
{
  string _textoIma;
  int _edoAct;

  char SigCar(ref int i)
  {
    if (i == _textoIma.Length)
    {
      i++;
      return '\0';
    }
    else
      return _textoIma[i++];
  }

  public bool Reconoce(string texto,int iniToken,ref int i,int noAuto)
  {
     char c;
     _textoIma = texto;
     string lenguaje;
     switch (noAuto)
     {
       //--------------  Automata  Id--------------
      case 0 : _edoAct = 0;
                  break;
       //--------------  Automata  OpAsig--------------
      case 1 : _edoAct = 3;
                  break;
     }
     while(i<=_textoIma.Length)
```

```
       switch (_edoAct)
       {
       //--------------  Automata  Id--------------
      case 0 : c=SigCar(ref i);
                    if
((lenguaje="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz").IndexOf(c)>=0)  _edoAct=1;  else
                     { i=iniToken;
                          return false; }
                    break;
      case 1 : c=SigCar(ref i);
                    if
((lenguaje="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz").IndexOf(c)>=0)  _edoAct=1;  else
                    if ((lenguaje="0123456789").IndexOf(c)>=0)  _edoAct=1;  else
                    if ((lenguaje="_").IndexOf(c)>=0)  _edoAct=1;  else
                    if ((lenguaje=" !\"#$%&\'()*+,-./:;<=>?@[\\]^`{|}~□€□,ƒ„…†‡ˆ‰Š‹Œ□Ž□□‘’“”•––
˜™š›œ□žŸ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿\n\t\r\f").IndexOf(c)>=0) _edoAct=2;  else
                     { i=iniToken;
                          return false; }
                    break;
      case 2 : i--;
                    return true;
                    break;
       //--------------  Automata  OpAsig--------------
      case 3 : c=SigCar(ref i);
                    if ((lenguaje="=").IndexOf(c)>=0)  _edoAct=4;  else
                     { i=iniToken;
                          return false; }
                    break;
      case 4 : return true;
                    break;
       }
     switch (_edoAct)
     {
            case 2 :      // Autómata  Id
                           --i;
                           return true;
     }
     return false;
  }

} // fin de la clase Automata
```

CLASE LEXICO.

```
//---------------------------------------
// clase Lexico
//---------------------------------------
//---------------------------
class Lexico
 {
   const int TOKREC = 2;
   const int MAXTOKENS = 500;
   string[] _lexemas;
   string[] _tokens;
   string _lexema;
   int _noTokens;
   int _i;
   int _iniToken;
   Automata oAFD;

   public Lexico()  // constructor por defecto
   {
     _lexemas = new string[MAXTOKENS];
     _tokens = new string[MAXTOKENS];
     oAFD = new Automata();
     _i = 0;
     _iniToken = 0;
     _noTokens = 0;
   }

   public void Inicia()
   {
     _i = 0;
```

```
      _iniToken = 0;
      _noTokens = 0;
    }

    public void Analiza(string texto)
    {
      bool recAuto;
      int noAuto;
      while (_i < texto.Length)
      {
        recAuto=false;
        noAuto=0;
        for(;noAuto<TOKREC&&!recAuto;)
          if(oAFD.Reconoce(texto,_iniToken,ref _i,noAuto))
            recAuto=true;
          else
            noAuto++;
        if (recAuto)
        {
          _lexema = texto.Substring(_iniToken, _i - _iniToken);
          switch (noAuto)
          {
            //--------------  Automata  Id--------------
           case 0 : _tokens[_noTokens] = "Id";
                    break;
            //--------------  Automata  OpAsig--------------
           case 1 : _tokens[_noTokens] = "OpAsig";
                    break;
          }
          _lexemas[_noTokens++] = _lexema;
        }
        else
          _i++;
        _iniToken = _i;
      }

    }

  }  // fin de la clase Lexico
  //-----------------------------
```

PASO 9.- Creación de la aplicación Windows C# - análisis léxico para tokens *Id* y *OpAsig*.

- Ya generamos el código de las clases *Automata* y *Lexico* usando SP-PS1. Lo que sigue es ejecutar Visual Studio y crear una nueva solución del tipo Windows para C#.
- Agreguemos los componentes : *label1*, *textBox1* multilinea, un botón *button1* y un *dataGridView1*. Asigna las propiedades de los respectivos componentes según se muestra en la fig#1.15.
- Vamos al código en *Form1.cs* y agregamos la definición del objeto *oAnaLex*.

```
public partial class Form1 : Form
    {
        Lexico oAnaLex = new Lexico();
        public Form1()
        {
            InitializeComponent();
        }
```

- Debemos ahora añadir una clase *Automata* al proyecto. Selecciona en Visual Studio la trayectoria del menú : *Project* | *Add Class*. Visual Studio te contesta con una caja de diálogo donde te pregunta por el nombre de la clase, fig#1.16.

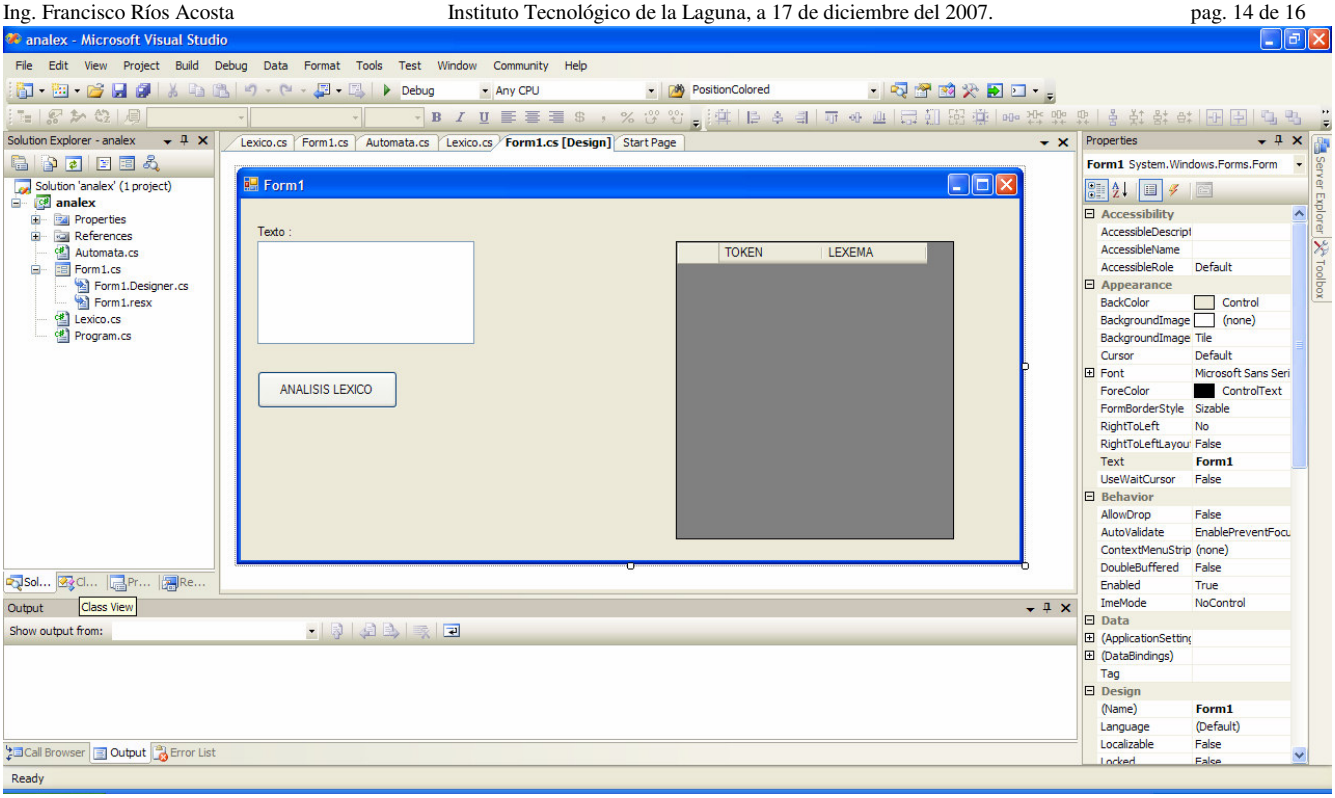

Fig. No. 1.15 Aplicación Windows C# para un analizador léxico que usa el código generado por *SP-PS1*.

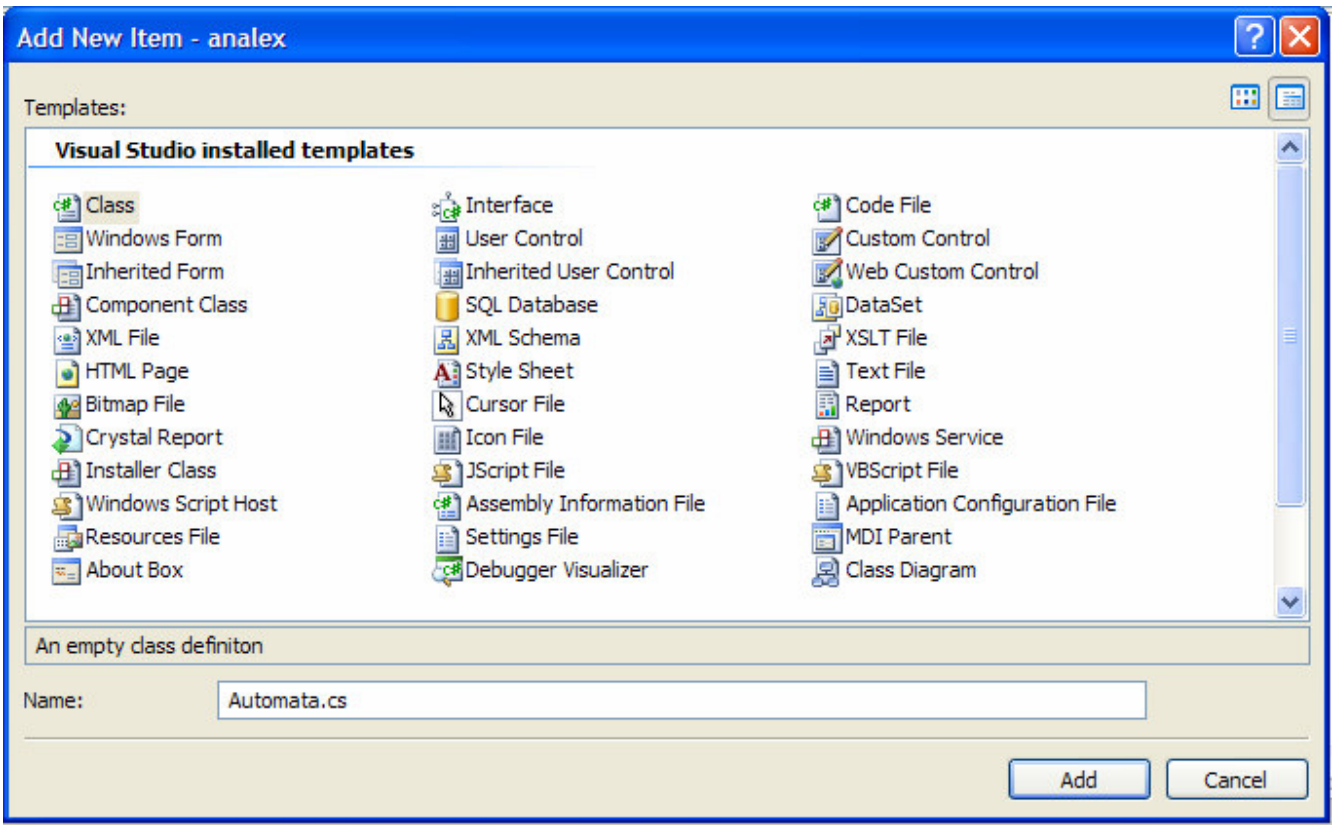

Fig. No. 1.16 Adición de la clase *Automata* al proyecto.

- La clase nueva de nombre *Automata*, estará vacía en su cuerpo. Debemos llenarla con el código que genera *SP-PS1* y que se encuentra en la ventana con leyenda CLASE AUTOMATA. La clase *Automata.cs* tendrá ahora el código que generamos anteriormente, fig#1.17.

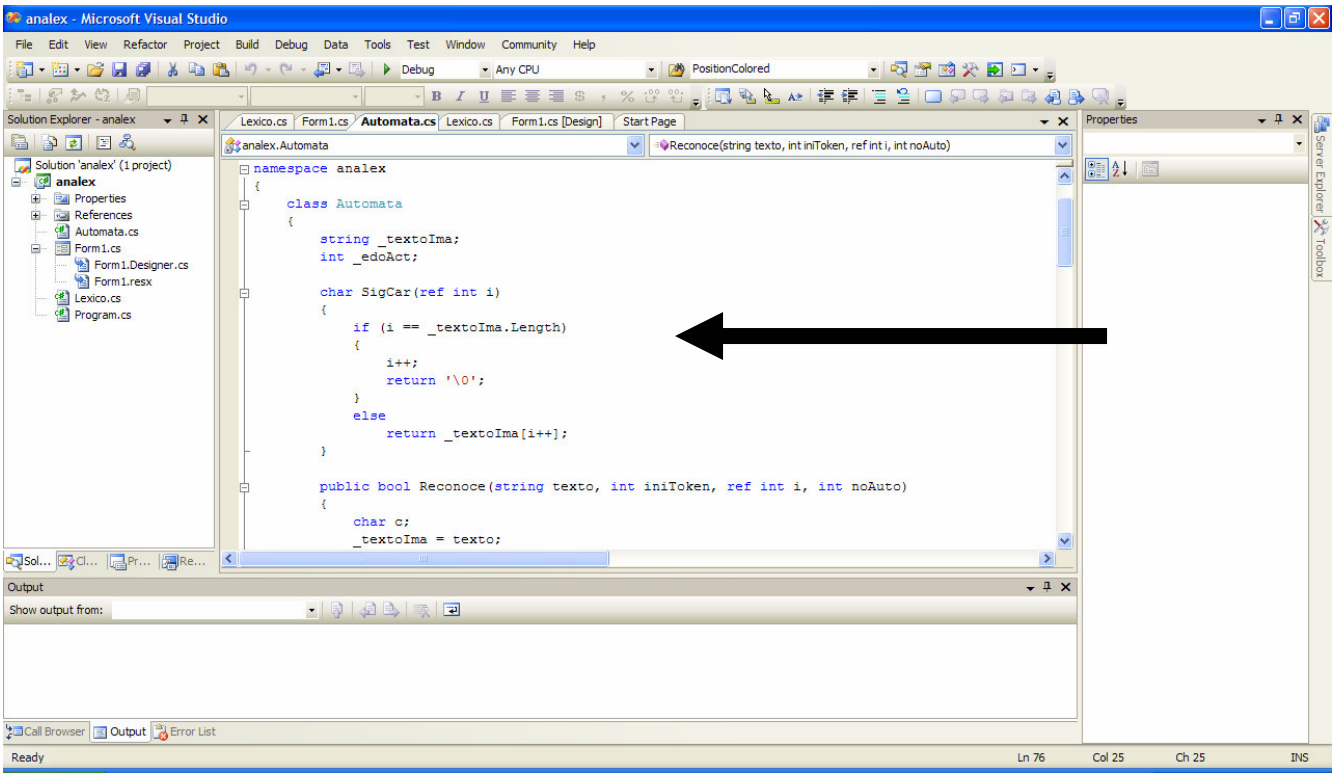

Fig. No. 1.17 Clase *Automata* con el código generado incrustado.

- Hagamos lo mismo para la clase *Lexico*. Agreguemos la clase *Lexico* desde Visual Studio accediendo al menu : *Project* | *Add Class* y demosle el nombre de *Lexico.cs*.
- El cuerpo de la clase Lexico está vacío. Debemos tomar el código generado por SP-PS1 para esta clase y agregarlo dentro de su cuerpo, fig#1.18.
- Sólo falta agregar el código que se ejecuta cuando hacemos un click sobre el botón ANALISIS LEXICO. Añade el código que se muestra a continuación :

```
private void button1_Click(object sender, EventArgs e)
        {
            oAnaLex.Inicia();
            oAnaLex.Analiza(textBox1.Text);
            dataGridView1.Rows.Clear();
            dataGridView1.Rows.Add(oAnaLex.NoTokens);
            for (int i = 0; i < oAnaLex.NoTokens; i++)
            {
                dataGridView1.Rows[i].Cells[0].Value = oAnaLex.Token[i];
                dataGridView1.Rows[i].Cells[1].Value = oAnaLex.Lexema[i];
            }
        }
```

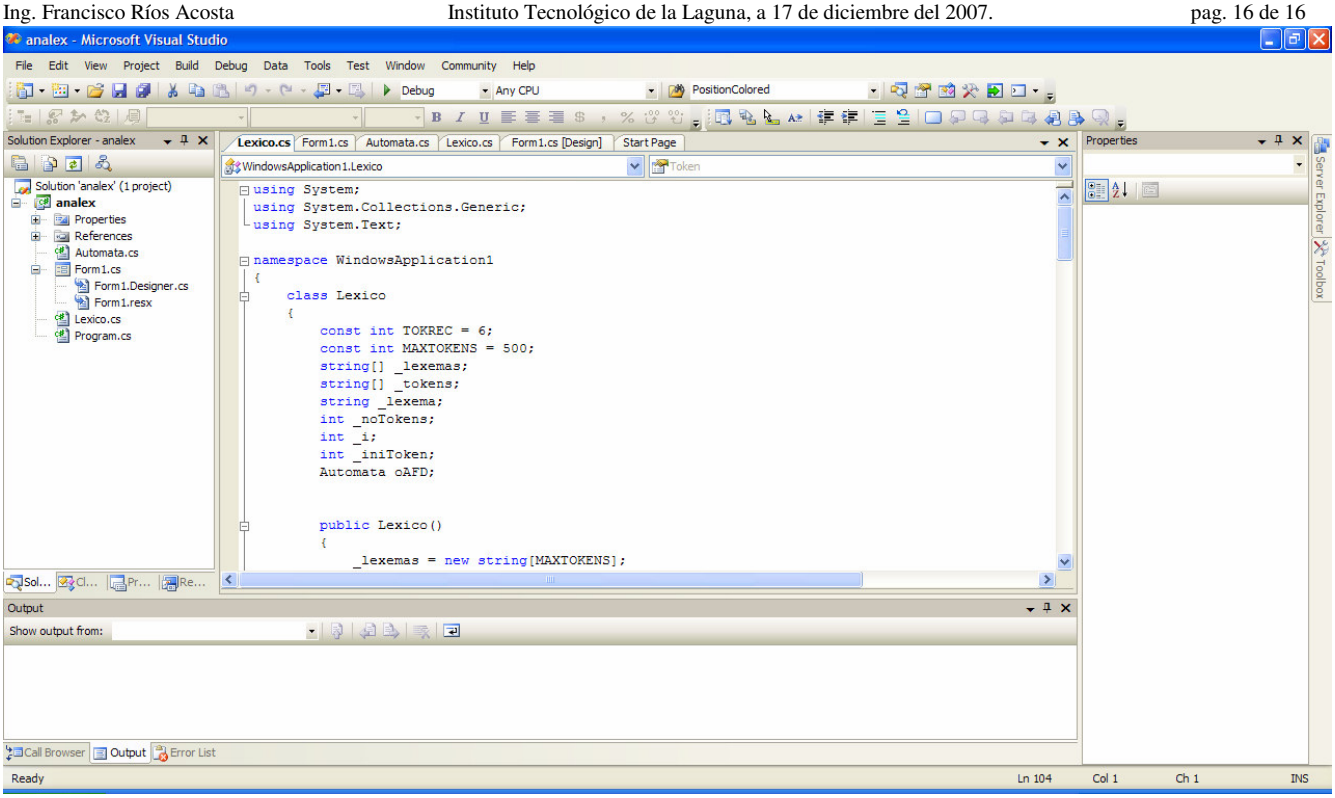

Fig. No. 1.18 Clase *Lexico* con el código generado incrustado.

- Observemos que existen 3 propiedades de la clase *Lexico* que debemos definir dentro de ella : *NoTokens, Token y Lexema*. El código siguiente tiene las definiciones de estas propiedades. Agregalas dentro de la clase *Lexico*.

```
public int NoTokens
        {
            get { return _noTokens; }
        }

        public string[] Token
        {
            get { return _tokens; }
        }

        public string[] Lexema
        {
            get { return _lexemas; }
        }
```

- Sólo falta ejecutar la aplicación. Escribe en la ventana de texto cualquier cadena que contenga identificadores y operadores de asignación. La aplicación deberá reconocerlos y visualizarlos en el *dataGridView1*.