GraphS

Graphs **S**pecification Language

Analizador Léxico

Procesadores de Lenguajes

Curso 2008 – 2009

Jose Domingo López López
josed.lopez1@alu.uclm.es

Ángel Escribano Santamarina
angel.escribano1@alu.uclm.es

Escuela Superior de Informática
Universidad de Castilla-La Mancha

Contenidos

Parte léxica del lenguaje

La parte léxica del lenguaje dado en el EBNF de la Tabla 1, es dado por medio de la tabla de tokens (ver tabla 2), en la que se muestran los tokens, sus patrones y algunos lexemas.

Cabe destacar que dada la gran cantidad de palabras reservadas que puede llegar a tener el lenguaje, hemos optado por utilizar una tabla de palabras reservadas (ver tabla 3), la cual se consultará cada vez que reconozcamos el token *ident* (identificador). Esto se ha hecho para reducir el tamaño del autómata que reconoce los tokens del lenguaje.

Tabla 1: EBNF del lenguaje GraphS

```
SYNTAX ::= DECL GRAPHS_SET
GRAPHS_SET ::= GRAPH {GRAPH}
GRAPH ::= graph ID '{' BODY '}' ';'
BODY ::= (DECL ARCS | OPS_GEN) [OPS]
DECL ::= {DECL_NODES | DECL_EDGES}
DECL_NODES ::= node ID {',' ID} ';'
DECL_EDGES ::= edge ID ['(' INT ')'] {',' ID ['(' INT ')']} ';'
ARCS ::= ARC {ARC}
ARC ::= ID '=' ID CONNECTOR ID ';'
CONNECTOR ::= '-' | '->'
OPS_GEN ::= OP_GEN
OP_GEN::= OP5
OPS ::= {OP}
OP ::= (OP1 | OP2) ';'
OP1 ::= OPN1 '(' ID ')'
OPN1 ::= minimumSpanningTree
OP2 ::= OPN2 '(' ID ',' ID ')'
OPN2 ::= shortestPath
OP5 ::= OPN5 '(' ID ',' ID ',' ID ',' ID ',' ID ')'
OPN5 ::= union
ID ::= (MAYUS | MINUS) {MAYUS | MINUS | DIGITO}
INT ::= 0 | DIG {DIGITO}
MAYUS ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N
| O | P | Q | R | S | T | U | V | W | X | Y | Z
MINUS ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n
| o | p | q | r | s | t | u | v | w | x | y | z
DIG ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
DIGITO ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
COMMENT ::= '/''/' {^LINETERMINATOR} LINETERMINATOR
COMMENT_M ::= '/''*' {ANY_STRING | LINETERMINATOR} '*''/'
LINETERMINATOR ::= \r | \n | \r\n
```

Tabla 2: Tokens del lenguaje GraphS

Token	Patrón	Lexema	Estado AFD	Id
graph	graph	graph	q13	1
node	node	node	q13	2
edge	edge	edge	q13	3
operadorUnario	minimumSpanningTree	minimumSpanningTree	q13	4
operadorBinario	shortestPath	shortestPath	q13	5
operadorQuinario	union	union	q13	6
l_bracket	{	{	q2	7
r_bracket	}	}	q3	8
l_paren	(	(	q4	9
r_paren	)	)	q5	10
semicolon	;	;	q6	11
comma	,	,	q7	12
equal	=	=	q8	13
connector	- | ->	-, ->	q9	14
ident	[a-zA-Z][[a-zA-Z0-9]*	A, arista1, nodoA	q13	15
number	0 | [1-9][0-9]*	0, 107, 991	q11 y q12	16

Tabla 3: Palabras reservadas del lenguaje GraphS

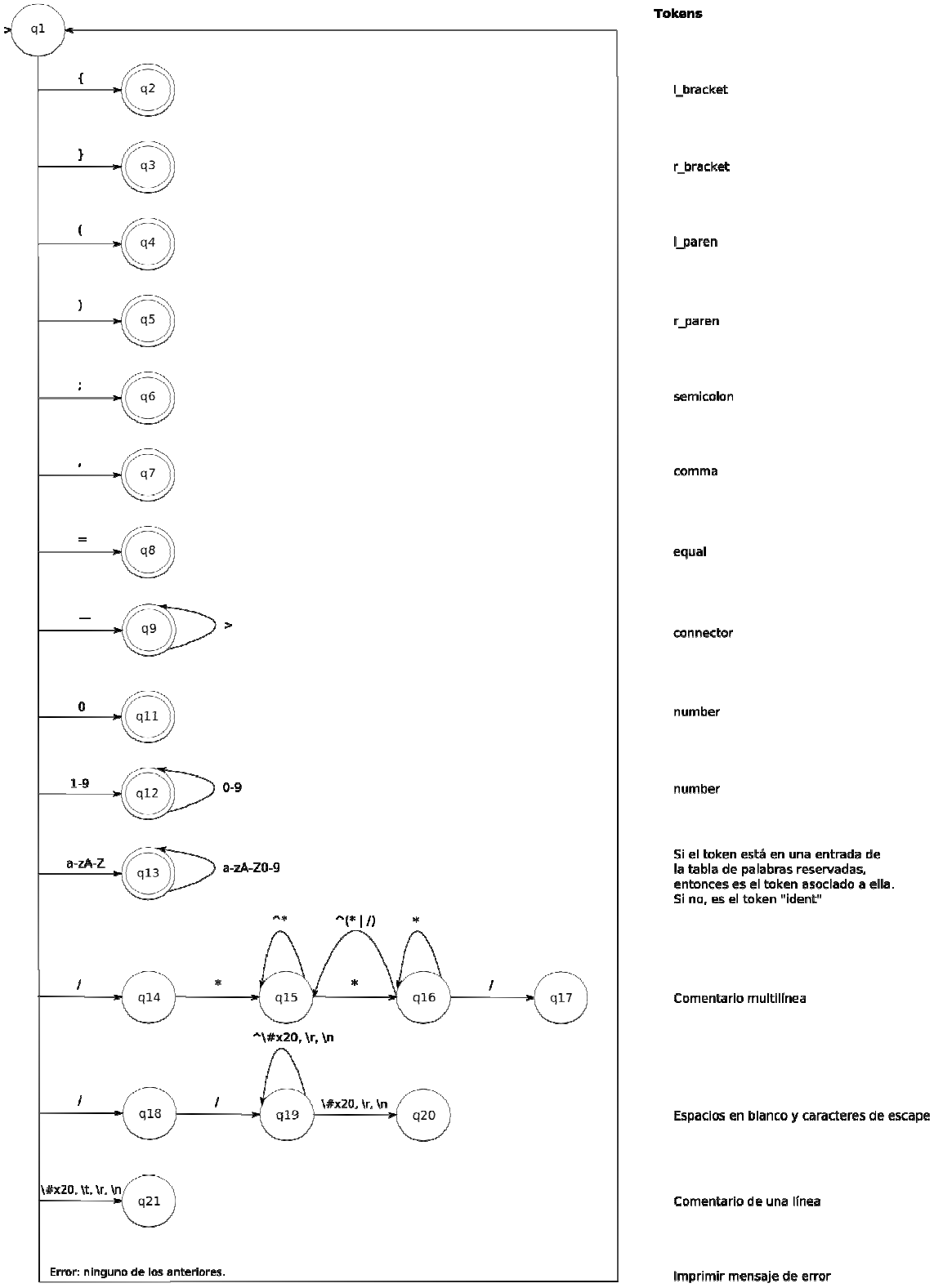
Palabra reservada	Token
graph	graph
node	node
edge	edge
minimumSpanningTree	operadorBinario
shortestPath	operadorBinario
union	operadorQuinario

Autómata finito determinista

En esta sección se muestra el autómata finito determinista (AFD) que reconoce los tokens del lenguaje GraphS y la asociación con cada token es mostrada en la tabla 2.

El estado q13 representa cualquier secuencia de letras en minúscula y mayúscula. Cuando se llega a este estado, se debe comprobar si la secuencia leída corresponde con una de las palabras reservadas, almacenada en la **tabla de palabras reservadas** (ver Tabla 3). Si es así, se devolverá el token correspondiente con la palabra reservada y el estado se considera final. Si no se encuentra en esa tabla, la secuencia que nos llevó a ese estado se considera un identificador (ident).

Los estados q17, q20 y q21 no son finales ya que se limitan a reconocer comentarios, espacios en blanco y caracteres de escape que no generan token y, por tanto, no se pasarán al analizador sintáctico.


Tokens
q1
{
q2
l_bracket
}
q3
r_bracket
(
q4
l_paren
)
q5
r_paren
;
q6
semicolon
,
q7
comma
=
q8
equal
_
q9
>
connector
0
q11
number
1-9
q12
0-9
number
a-zA-Z
q13
a-zA-Z0-9
Si el token está en una entrada de la tabla de palabras reservadas, entonces es el token asociado a ella. Si no, es el token "ident"
^*
^(* | /)
*
/
q14
*
q15
*
q16
/
q17
Comentario multilínea
^\#x20, \r, \n
/
q18
/
q19
\#x20, \r, \n
q20
Espacios en blanco y caracteres de escape
\#x20, \t, \r, \n
q21
Comentario de una línea
Error: ninguno de los anteriores.
Imprimir mensaje de error

Analizador léxico

El analizador léxico (AnaLex.java) se ha generado con la herramienta JFlex y el siguiente fichero de entrada *analex.flex*:

```
/* Codigo de usuario */

class Utility {
      public static final String Keywords[] =
{"graph","node","edge","minimumSpanningTree","shortestPath","union"};
      public static final String errorMsg[] = {
            "Identificador no valido",
            "Error de sintaxis"
      };

      public static boolean isKeyword (String cadena) {
            boolean found = false;
            for (int i = 0; i < Keywords.length && !found; i++) {
                  if (cadena.equals(Keywords[i])) found = true;
            }
            return found;
      }

      public static String Keyword (String cadena) {
            String token = "";

            if (cadena.equals("graph")) token = "graph";
            else if (cadena.equals("node")) token = "node";
            else if (cadena.equals("edge")) token = "edge";
            else if (cadena.equals("minimumSpanningTree")) token =
"operadorUnario";
            else if (cadena.equals("shortestPath")) token =
"operadorBinario";
            else if (cadena.equals("union")) token =
"operadorQuinario";
            else System.out.println (":: ERROR. El lexema no se
encuentra asociado a ningun token.");

            return token;

      }

      public static String errorMsg (int error) {
            return errorMsg[error];
      }

}


class Main {
      public static void main (String argv[]) {
            if (argv.length == 0 || argv.length > 1) {
                  System.out.println("Uso : java AnaLex <inputfile>");
            }
            else {
                  AnaLex scanner = null;
                  try {
```

```
                        scanner = new AnaLex( new
java.io.FileReader(argv[0]));
                        System.out.println (":: Analizando fichero
"+argv[0]+"\n");
                        scanner.yylex();
                  }
                        catch (java.io.FileNotFoundException e) {
                        System.out.println("File not found :
\""+argv[0]+"\"");
                  }
                  catch (java.io.IOException e) {
                        System.out.println("IO error scanning file
\""+argv[0]+"\"");
                        System.out.println(e);
                  }
                  catch (Exception e) {
                        System.out.println("Unexpected exception:");
                        e.printStackTrace();
                  }
            }
      }
}


%%

/* Seccion de opciones y declaraciones */

%class AnaLex
/* %standalone */
%type Object
%line
%column
%switch
%states COMMENT, COMMENTM

%eof{
      if(zzLexicalState==COMMENTM)
            System.out.println(Utility.errorMsg(ERROR_SYNTAX)+".
Apertura de comentario sin cierre. Linea: "+(yyline+1)+" Columna:
"+(yycolumn+1));
%eof}

%{
public static final int ERROR_MSG_IDENT = 0;
public static final int ERROR_SYNTAX  = 1;



%}

/* Macros */

LineTerminator = \r | \n | \r\n
WhiteSpace = {LineTerminator} | [\t\f] | " "



%%
```

```
<YYINITIAL> {
      "//"  { yybegin(COMMENT); }
      "/*"  { yybegin(COMMENTM); }
      [a-zA-Z][a-zA-Z0-9]*    {
                              if (Utility.isKeyword(yytext())) {
                                    String Keyword = new String();
                                    Keyword =
Utility.Keyword(yytext());
                                    if (Keyword.equals(yytext())) {
                                          System.out.println ("Token
reconocido: "+Keyword+" id(13) Linea: " +(yyline+1)+" Columna:
"+(yycolumn+1));
                                    } else {
                                          System.out.println ("Token
reconocido: "+Keyword+" id(13) Lexema: "+yytext()+" Linea: "
+(yyline+1)+" Columna: "+(yycolumn+1));
                                    }
                              } else {
                                    System.out.println("Token
reconocido: ident id(13) Lexema: "+yytext()+" Linea: " +(yyline+1)+"
Columna: "+(yycolumn+1));
                              }
                        }
      "{"               { System.out.println("Token reconocido: {
id(2) Linea: " +(yyline+1)+" Columna: "+(yycolumn+1)); }
      "}"               { System.out.println("Token reconocido: }
id(3) Linea: " +(yyline+1)+" Columna: "+(yycolumn+1)); }
      "("               { System.out.println("Token reconocido: (
id(4) Linea: " +(yyline+1)+" Columna: "+(yycolumn+1)); }
      ")"               { System.out.println("Token reconocido: )
id(5) Linea: " +(yyline+1)+" Columna: "+(yycolumn+1)); }
      ";"               { System.out.println("Token reconocido: ;
id(6) Linea: " +(yyline+1)+" Columna: "+(yycolumn+1)); }
      ","               { System.out.println("Token reconocido: ,
id(7) Linea: " +(yyline+1)+" Columna: "+(yycolumn+1)); }
      "="               { System.out.println("Token reconocido: =
id(8) Linea: " +(yyline+1)+" Columna: "+(yycolumn+1)); }
      "-" | "->"        { System.out.println("Token reconocido: -
id(9) Linea: " +(yyline+1)+" Columna: "+(yycolumn+1)); }
      0                 { System.out.println("Token reconocido: number
id(11) Lexema: "+yytext()+" Linea: " +(yyline+1)+" Columna:
"+(yycolumn+1)); }
      [1-9][0-9]*       { System.out.println("Token reconocido: number
id(12) Lexema: "+yytext()+" Linea: " +(yyline+1)+" Columna:
"+(yycolumn+1)); }
      "*/"              {
System.out.println(Utility.errorMsg(ERROR_SYNTAX)+". Fin de comentario
sin apertura. Linea: "+(yyline+1)+" Columna: "+(yycolumn+1)); }
      [0-9][a-zA-Z0-9]* {
System.out.println(Utility.errorMsg(ERROR_MSG_IDENT)+" <"+yytext()+">
Linea: "+(yyline+1)+" Columna: "+(yycolumn+1)); }
      {WhiteSpace}            { }
      .                 { System.out.println("Expresion ilegal
<"+yytext()+"> Linea: "+(yyline+1)+" Columna: "+(yycolumn+1)); }

}



<COMMENT> {
      {LineTerminator}  { yybegin(YYINITIAL); }
```

```
      .     { }
}


<COMMENTM> {
      "*/"  { yybegin(YYINITIAL); }
      {LineTerminator}  { }
      .     { }
}
```

Cabe destacar las siguientes características del analizador léxico:

- En la Sección de código de usuario se ha implementado una clase **Utility** en la cual mantendremos la tabla de palabras reservadas y una tabla de errores, así como las funciones correspondientes a la comprobación de una cadena en la tabla de palabras reservadas, devolver el token de una palabra reservada, e imprimir mensajes de error.

- En el Sección de opciones y declaraciones se observa que:
  - El nombre de la clase que se genera es AnaLex.
  - El analizador léxico es autónomo ya que hemos creado nuestro propio método *main*, por eso se ha comentado la propiedad *standalone* y se ha añadido una sentencia *type Object*.
  - Está implementado por medio de bloques *switch*.
  - Tiene dos estados adicionales para el tratamiento de estados *mono-línea* y *multi-línea*.

- En la Sección de reglas léxicas se puede observar la definición de los estados declarados en la sección anterior así como la definición del estado YYINITIAL. Es importante observar también la inclusión de reglas que reconocen cadenas **no** válidas, así como cierres de comentario sin apertura, e identificadores que comienzan por un dígito. El tratamiento apertura de comentarios sin cierre se hace en la sección de opciones y declaraciones en el bloque *eof*.

Ejemplo de cadena perteneciente al lenguaje

```
/* Ejemplo de un grafo no dirigido y no valorado */
graph Grafo1 {
     node A, B;
     edge X;

     X = A - B;

     shortestPath (A, E);
     minimumSpanningTree (D);
};
```

```
Token reconocido: graph id(14) Linea: 2 Columna: 1
Token reconocido: ident id(12) Lexema: Grafo1 Linea: 2 Columna:
7
Token reconocido: { id(2) Linea: 2 Columna: 14
Token reconocido: node id(14) Linea: 3 Columna: 2
Token reconocido: ident id(12) Lexema: A Linea: 3 Columna: 7
Token reconocido: , id(7) Linea: 3 Columna: 8
Token reconocido: ident id(12) Lexema: B Linea: 3 Columna: 10
Token reconocido: ; id(6) Linea: 3 Columna: 11
Token reconocido: edge id(14) Linea: 4 Columna: 2
Token reconocido: ident id(12) Lexema: X Linea: 4 Columna: 7
Token reconocido: ; id(6) Linea: 4 Columna: 8
Token reconocido: ident id(12) Lexema: X Linea: 6 Columna: 2
Token reconocido: = id(8) Linea: 6 Columna: 4
Token reconocido: ident id(12) Lexema: A Linea: 6 Columna: 6
Token reconocido: - id(9) Linea: 6 Columna: 8
Token reconocido: ident id(12) Lexema: B Linea: 6 Columna: 10
Token reconocido: ; id(6) Linea: 6 Columna: 11
Token reconocido: operadorBinario id(14) Lexema: shortestPath
Linea: 8 Columna: 5
Token reconocido: ( id(4) Linea: 8 Columna: 18
Token reconocido: ident id(12) Lexema: A Linea: 8 Columna: 19
Token reconocido: , id(7) Linea: 8 Columna: 20
Token reconocido: ident id(12) Lexema: E Linea: 8 Columna: 22
Token reconocido: ) id(5) Linea: 8 Columna: 23
Token reconocido: ; id(6) Linea: 8 Columna: 24
Token reconocido: operadorBinario id(14) Lexema:
minimumSpanningTree Linea: 9 Columna: 5
Token reconocido: ( id(4) Linea: 9 Columna: 25
Token reconocido: ident id(12) Lexema: D Linea: 9 Columna: 26
Token reconocido: ) id(5) Linea: 9 Columna: 27
Token reconocido: ; id(6) Linea: 9 Columna: 28
Token reconocido: } id(3) Linea: 10 Columna: 1
Token reconocido: ; id(6) Linea: 10 Columna: 2
```

Ejemplo de cadena no perteneciente al lenguaje

```
/* Ejemplo de un grafo dirigido y valorado */
graph 999Grafo2 {
     node 0A, B0;
     edge X(7);

     X += A - B;

     shortestPath (A, F);
     minimumSpanningTree (E);
};
```

```
Token reconocido: graph id(14) Linea: 2 Columna: 1
Identificador no valido <999Grafo2> Linea: 2 Columna: 7
Token reconocido: { id(2) Linea: 2 Columna: 17
Token reconocido: node id(14) Linea: 3 Columna: 2
Identificador no valido <0A> Linea: 3 Columna: 7
Token reconocido: , id(7) Linea: 3 Columna: 9
Token reconocido: ident id(12) Lexema: B0 Linea: 3 Columna: 11
Token reconocido: ; id(6) Linea: 3 Columna: 13
Token reconocido: edge id(14) Linea: 4 Columna: 2
Token reconocido: ident id(12) Lexema: X Linea: 4 Columna: 7
Token reconocido: ( id(4) Linea: 4 Columna: 8
Token reconocido: int id(13) Lexema: 7 Linea: 4 Columna: 9
Token reconocido: ) id(5) Linea: 4 Columna: 10
Token reconocido: ; id(6) Linea: 4 Columna: 11
Token reconocido: ident id(12) Lexema: X Linea: 6 Columna: 2
Expresion ilegal <+> Linea: 6 Columna: 4
Token reconocido: = id(8) Linea: 6 Columna: 5
Token reconocido: ident id(12) Lexema: A Linea: 6 Columna: 7
Token reconocido: - id(9) Linea: 6 Columna: 9
Token reconocido: ident id(12) Lexema: B Linea: 6 Columna: 11
Token reconocido: ; id(6) Linea: 6 Columna: 12
Token reconocido: operadorBinario id(14) Lexema: shortestPath
Linea: 8 Columna: 5
Token reconocido: ( id(4) Linea: 8 Columna: 18
Token reconocido: ident id(12) Lexema: A Linea: 8 Columna: 19
Token reconocido: , id(7) Linea: 8 Columna: 20
Token reconocido: ident id(12) Lexema: F Linea: 8 Columna: 22
Token reconocido: ) id(5) Linea: 8 Columna: 23
Token reconocido: ; id(6) Linea: 8 Columna: 24
Token reconocido: operadorBinario id(14) Lexema:
minimumSpanningTree Linea: 9 Columna: 5
Token reconocido: ( id(4) Linea: 9 Columna: 25
Token reconocido: ident id(12) Lexema: E Linea: 9 Columna: 26
Token reconocido: ) id(5) Linea: 9 Columna: 27
Token reconocido: ; id(6) Linea: 9 Columna: 28
Token reconocido: } id(3) Linea: 10 Columna: 1
Token reconocido: ; id(6) Linea: 10 Columna: 2
```