GraphS

Graphs **S**pecification Language

(Analizador Léxico)

Procesadores de Lenguajes
Curso 2008 – 2009

Escuela Superior de Informática
Universidad de Castilla-La Mancha

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

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

Contenidos

Parte léxica del lenguaje (Tabla de tokens)

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 *string*. Esto se ha hecho para reducir el tamaño del autómata que reconoce los tokens del lenguaje.

Tabla 1: EBNF del lenguaje GraphS

```
SYNTAX ::= [DECL_GLOBAL] GRAPHS_SET
DECL_GLOBAL ::= DECL
GRAPHS_SET ::= GRAPH {GRAPH}
GRAPH ::= graph ID '{' BODY '}' ';'
BODY ::= (DECL_LOCAL ARCS | OPS_GEN) [OPS]
DECL_LOCAL ::= DECL
DECL ::= DECL_NODES {DECL_NODES} DECL_EDGES {DECL_EDGES}
DECL_NODES ::= node ID {',' ID} ';'
DECL_EDGES ::= edge ID ['(' INT ')'] {',' ID ['(' INT ')']}
';'
ARCS ::= ARC {ARC}
ARC ::= ID '=' ID ('-' | '->') ID ';'
OPS_GEN ::= OP_GEN
OP_GEN::= OP5
OPS ::= {OP}
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
```

Tabla 2: Tokens del lenguaje GraphS

Token	Patrón	Lexema	Estado AFD	Id
graph	graph	graph	q12	1
node	node	node	q12	2
edge	edge	edge	q12	3
op	op	op	q12	4
minimumSpanningTree	minimumSpanningTree	minimumSpanningTree	q12	5
shortestPath	shortestPath	shortestPath	q12	6
union	union	union	q12	7
l_bracket	{	{	q2	8
r_bracket	}	}	q3	9
l_paren	(	(	q4	10
r_paren	)	)	q5	11
semicolon	;	;	q6	12
comma	,	,	q7	13
equal	=	=	q8	14
dash	-	-	q9	15
arrow	->	->	q11	16
string	[a-zA-Z][[a-zA-Z0-9]*	A, arista1, nodoA	q12	17
int	0 | [1-9][0-9]*	0, 107, 991	q13	18

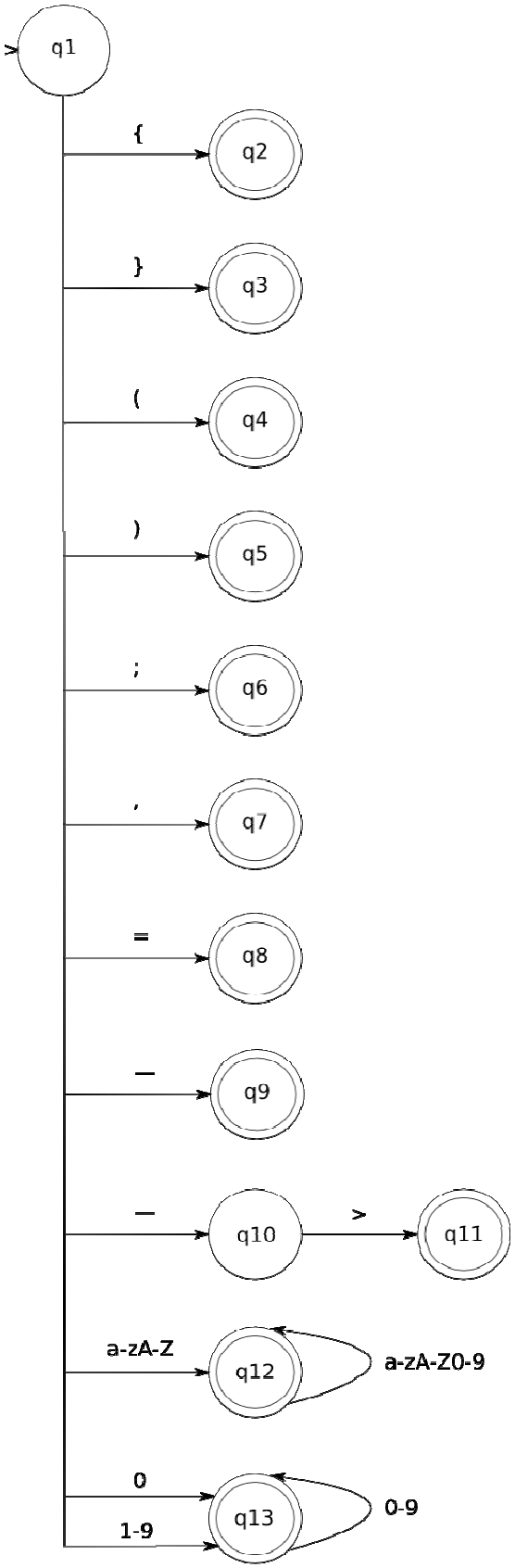
Tabla 3: Palabras reservadas del lenguaje GraphS

Palabra reservada	Token
graph	graph
node	node
edge	edge
op	op
minimumSpanningTree	minimumSpanningTree
shortestPath	shortestPath
union	union

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 q12 representa cualquier secuencia de letras en minúscula, mayúscula y números. 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, el estado se considera normal.



Tokens

{

}

(

)

;

,

=

—

—>

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 "string"

int

Analizador léxico (Fichero analex.flex)

El analizador léxico se ha generado con la herramienta JFlex y el siguiente fichero de entrada:

```
/* Codigo de usuario */

%%

/* Seccion de operaciones y declaraciones */

%class AnaLex
%standalone
%line
%column
%switch
%states COMMENT, COMMENTM

%{
%}

/* Macros */

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

%%

<YYINITIAL> {
     "//" { yybegin(COMMENT); }
     "/*" { yybegin(COMMENTM); }
     "graph"   { System.out.println("Token reconocido:
graph id(1)"); }
     "node"    { System.out.println("Token reconocido:
node id(2)"); }
     "edge"    { System.out.println("Token reconocido:
edge id(3)"); }
     "op" { System.out.println("Token reconocido: op
id(4)"); }
     "minimumSpanningTree"    { System.out.println("Token
reconocido: minimumSpanningTree id(5)"); }
     "shortestPath" { System.out.println("Token
reconocido: shortestPath id(6)"); }
     "union"   { System.out.println("Token reconocido:
union id(7)"); }
     "{"  { System.out.println("Token reconocido: {
id(8)"); }
     "}"  { System.out.println("Token reconocido: }
```

```
id(9)"); }
     "("  { System.out.println("Token reconocido: (
id(10)"); }
     ")"  { System.out.println("Token reconocido: )
id(11)"); }
     ";"  { System.out.println("Token reconocido: ;
id(12)"); }
     ","  { System.out.println("Token reconocido: ,
id(13)"); }
     "="  { System.out.println("Token reconocido: =
id(14)"); }
     "-"  { System.out.println("Token reconocido: -
id(15)"); }
     "->" { System.out.println("Token reconocido: ->
id(16)"); }
     [a-zA-Z][a-zA-Z0-9]*     { System.out.println("Token
reconocido: string id(17) Lexema: "+yytext()); }
     0 | [1-9][0-9]*     { System.out.println("Token
reconocido: int id(18) Lexema: "+yytext()); }
     {WhiteSpace}   { }
     .    { System.out.println("Expresion ilegal
"+yyline+":"+yycolumn+" <" +yytext()+">"); }

}


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


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

A grandes rasgos se observan las siguientes propiedades:

- El nombre de la clase que se genera es Anales.
- El analizador léxico es autónomo (standalone).
- Está implementado por medio de bloques *switch*.
- Tiene dos estados adicionales para el tratamiento de estados *mono-línea* y *multi-línea*.

Ejemplo 1 – Cadena perteneciente al lenguaje

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

     X = A - B;

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

```
Token reconocido: graph id(1)
Token reconocido: string id(17) Lexema: Grafo1
Token reconocido: { id(8)
Token reconocido: node id(2)
Token reconocido: string id(17) Lexema: A
Token reconocido: , id(13)
Token reconocido: string id(17) Lexema: B
Token reconocido: ; id(12)
Token reconocido: edge id(3)
Token reconocido: string id(17) Lexema: X
Token reconocido: ; id(12)
Token reconocido: string id(17) Lexema: X
Token reconocido: = id(14)
Token reconocido: string id(17) Lexema: A
Token reconocido: - id(15)
Token reconocido: string id(17) Lexema: B
Token reconocido: ; id(12)
Token reconocido: op id(4)
Token reconocido: shortestPath id(6)
Token reconocido: ( id(10)
Token reconocido: string id(17) Lexema: A
Token reconocido: , id(13)
Token reconocido: string id(17) Lexema: E
Token reconocido: ) id(11)
Token reconocido: ; id(12)
Token reconocido: op id(4)
Token reconocido: minimumSpanningTree id(5)
Token reconocido: ( id(10)
Token reconocido: string id(17) Lexema: D
Token reconocido: ) id(11)
Token reconocido: ; id(12)
Token reconocido: } id(9)
Token reconocido: ; id(12)
```

Ejemplo 2 – Cadena no perteneciente al lenguaje

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

     X += A - B;

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

```
Token reconocido: graph id(1)
Token reconocido: int id(18) Lexema: 999
Token reconocido: string id(17) Lexema: Grafo2
Token reconocido: { id(8)
Token reconocido: node id(2)
Token reconocido: int id(18) Lexema: 0
Token reconocido: string id(17) Lexema: A
Token reconocido: , id(13)
Token reconocido: string id(17) Lexema: B0
Token reconocido: ; id(12)
Token reconocido: edge id(3)
Token reconocido: string id(17) Lexema: X
Token reconocido: ( id(10)
Token reconocido: int id(18) Lexema: 7
Token reconocido: ) id(11)
Token reconocido: ; id(12)
Token reconocido: string id(17) Lexema: X
Expresion ilegal 5:3 <+>
Token reconocido: = id(14)
Token reconocido: string id(17) Lexema: A
Token reconocido: - id(15)
Token reconocido: string id(17) Lexema: B
Token reconocido: ; id(12)
Token reconocido: op id(4)
Token reconocido: shortestPath id(6)
Token reconocido: ( id(10)
Token reconocido: string id(17) Lexema: A
Token reconocido: , id(13)
Token reconocido: string id(17) Lexema: F
Token reconocido: ) id(11)
Token reconocido: ; id(12)
Token reconocido: op id(4)
Token reconocido: minimumSpanningTree id(5)
Token reconocido: ( id(10)
Token reconocido: string id(17) Lexema: E
```

```
Token reconocido: ) id(11)
Token reconocido: ; id(12)
Token reconocido: } id(9)
Token reconocido: ; id(12)
```