Procesadores de Lenguajes I:

Una introducción a la fase de análisis

J. M. Benedí Ruíz
V. Gisbert Giner
L. Moreno Boronat
E. Vivancos Rubio

Capítulo 2

Análisis léxico

El objetivo de este capítulo es la presentación, descripción e implementación de un Analizador Léxico (AL). Para cubrir este objetivo se hace hincapié en la necesidad de un formalismo matemático para la especificación de las restricciones léxicas, un reconocedor de dicha especificación y unas acciones semánticas para complementar las funciones asignadas a un AL.

A continuación se plantea la construcción de un AL simple con el objetivo de destacar aquellos aspectos más importantes en el marco de un ejemplo concreto, analizando los problemas de definición de tokens y tipos de tokens, acciones semánticas asociadas al proceso de análisis de un cierto símbolo del lenguaje, tratamiento de las palabras reservadas, detección y manipulación de errores léxicos.

Finalmente, se propone el estudio de los generadores automáticos de analizadores léxicos. Este estudio se hace basándose en un generador automático denominado Flex. En el anexo A se presenta con más detalle esta herramienta.

2.1. Descripción de un AL

El módulo del Analizador Léxico (AL) lee el programa fuente carácter a carácter y genera una secuencia de unidades léxicas denominadas “tokens”.

Comenzaremos definiendo y diferenciando una serie de conceptos asociados con el AL que a menudo se confunden (Fig. 2.1). La especificación léxica de un lenguaje determina tanto los símbolos del lenguaje como una definición de los mismos. Un *lexema* es una instanciación (cadena de caracteres) de un símbolo si cumple la definición del mismo. Finalmente, un “*token*” está formado por una codificación del símbolo detectado y por la información (atributos) asociada.

Símbolo	Definición de los símbolos	Lexema	Token *símbolo*	*atributo*
identificador	Cadena de caracteres alfanuméricos empezando por uno alfabético.	X25	ident	ind (X25)
constante entera	Cadena de uno o más dígitos.	127	entero	valor (127)
op. relacional	<, >, =, ≥, ≤, ≠	>	oprel	cod (>)
op. asignación	:=	:=	asig	

Figura 2.1. Conceptos involucrados en el análisis léxico y sus principales diferencias.

2.1.1. Funciones de un AL

Las principales funciones de un AL son:

Detección de los símbolos del lenguaje.

Realización de las acciones asociadas a la detección de un símbolo.

Emisión de los tokens.

Y otras funciones auxiliares como:

- Lectura carácter a carácter del programa fuente.
- Eliminación de cadenas inútiles: comentarios, tabuladores, saltos de línea, etc.
- Detección de los errores léxicos.
- Relación de los mensajes de error con las líneas del programa fuente.
- Manejo eficiente de los ficheros.
- Reconocimiento y ejecución de las directivas de compilación.

2.1.2. AL en el seno de un compilador

Aunque todas las funciones de un AL podrían integrarse en el módulo del Análisis Sintáctico (AS), existen poderosas razones para separar el AL del AS:

- Simplificación del diseño del AS (existe una separación de gramáticas).

- Mejora de la eficacia (se construyen analizadores específicos).

- Portabilidad (sólo el AL depende del alfabeto de la máquina).

El AL puede ser un módulo separado que realiza el análisis léxico completo del programa fuente proporcionando al AS el resultado del mismo. Sin embargo, es mucho más eficiente diseñar el AL como una rutina del AS, que es invocada por éste cada vez que precise un "token" en el proceso de análisis sintáctico, tal y como se ilustra en la Fig. 2.2.

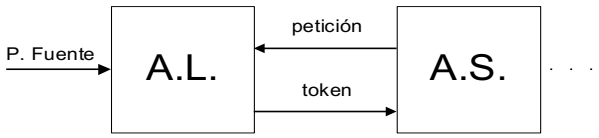

Figura 2.2. Análisis léxico como una rutina del análisis sintáctico.

2.2. Especificación de un AL

Tal y como se indicó en el capítulo anterior, por su adecuada capacidad para describir los símbolos del lenguaje, las *Expresiones Regulares* (ER) son el formalismo de especificación léxica elegido [Hopcroft 95]. Una posible definición recursiva de las ER es:

Dado un alfabeto Σ, se puede definir una Expresión Regular sobre Σ como:

$\varnothing$ es una ER y define el lenguaje $\varnothing$.

ε es una ER y define el lenguaje $\{ \varepsilon \}$.

$a \in \Sigma$ es una ER y define el lenguaje $\{ a \}$.

Sean r y s ER y L_r y L_s sus lenguajes respectivos, entonces se cumple:

$r \mid s$ es una ER y define el lenguaje $L_r \cup L_s$.

$r \cdot s$ es una ER y define el lenguaje $L_r \cdot L_s$.

r^* es una ER y define el lenguaje L_r^*.

En la Fig. 2.3 se puede apreciar un ejemplo de especificación léxica, para un cierto lenguaje de programación, definida por medio de ER.

Análogamente, se consideran los *Autómatas de Estados Finitos* (AEF) [Hopcroft 95] por su capacidad de reconocer los símbolos del lenguaje. Una posible definición de AEF es:

$$AEF = (Q, \Sigma, \delta, q_0, F),$$

Donde Q es el conjunto de estados, Σ es el alfabeto de entrada, $q_0 \in Q$ es el estado inicial, $F \subseteq Q$ es el conjunto de estados finales y $\delta: Q \times \Sigma \rightarrow P(Q)$ es la función de transición, tal que dado el estado y el símbolo actual proporciona el conjunto de posibles estados siguientes. Igualmente se puede definir un AEF determinista, sin más que definir la función de transición como: $\delta: Q \times \Sigma \longrightarrow Q$.

símbolos → identificador | constante_numérica | símbolo_especial

letra → a | ... | z | A | ... | Z

dígito → 0 | ... | 9

identificador → letra (letra | dígito)*

constante_numérica → constante_entera | constante_real

constante_entera → dígito (dígito)*

constante_real → constante_entera . (dígito)*

símbolo_especial → asignación | operador_relacional
| subrango | operador_aritmético
| separador | palabra_reservada

. . .

Figura 2.3. Ejemplo de especificación léxica de un lenguaje de programación.

En la Fig. 2.4 se muestra el AEF asociado a las ER que definían la especificación léxica de la Fig. 2.3.

Como sabemos, existe una serie de teoremas constructivos que permiten relacionar las ER con los AEF [Hopcroft 95]; de tal forma que dado un lenguaje definido por una ER podemos construir un AEF que acepte dicho lenguaje. Este resultado es fundamental ya que posibilita que a partir de la especificación léxica (expresada mediante ER) podamos obtener el AEF que nos permita reconocer los símbolos del lenguaje [Aho 90].

2.3. Construcción de un AL

Aprovechando los resultados anteriores, para construir un AL será necesario especificar (mediante ER) los símbolos del lenguaje y definir las acciones asociadas con la detección de cada símbolo; ya que a partir de las ER se puede obtener los AEF que nos permiten reconocer los símbolos del lenguaje. No obstante, existen algunos aspectos importantes de implementación que es necesario destacar:

Detección de los símbolos del lenguaje.- Este problema puede resultar bastante difícil, dependiendo de las restricciones del lenguaje. Por ejemplo en FORTRAN,

```
DO 10    I = 1, 27
 DO 10    I = 1.27
```

son respectivamente una instrucción repetitiva `DO` y una instrucción de asignación de la constante `1.27` a la variable `DO10I`. Esto significa que la detección de los símbolos debe tener

una capacidad de anticipación ya que con la detección del `DO` no se sabe, hasta que se detecta la "," o el ".", si es una instrucción repetitiva o la parte del nombre de una variable.

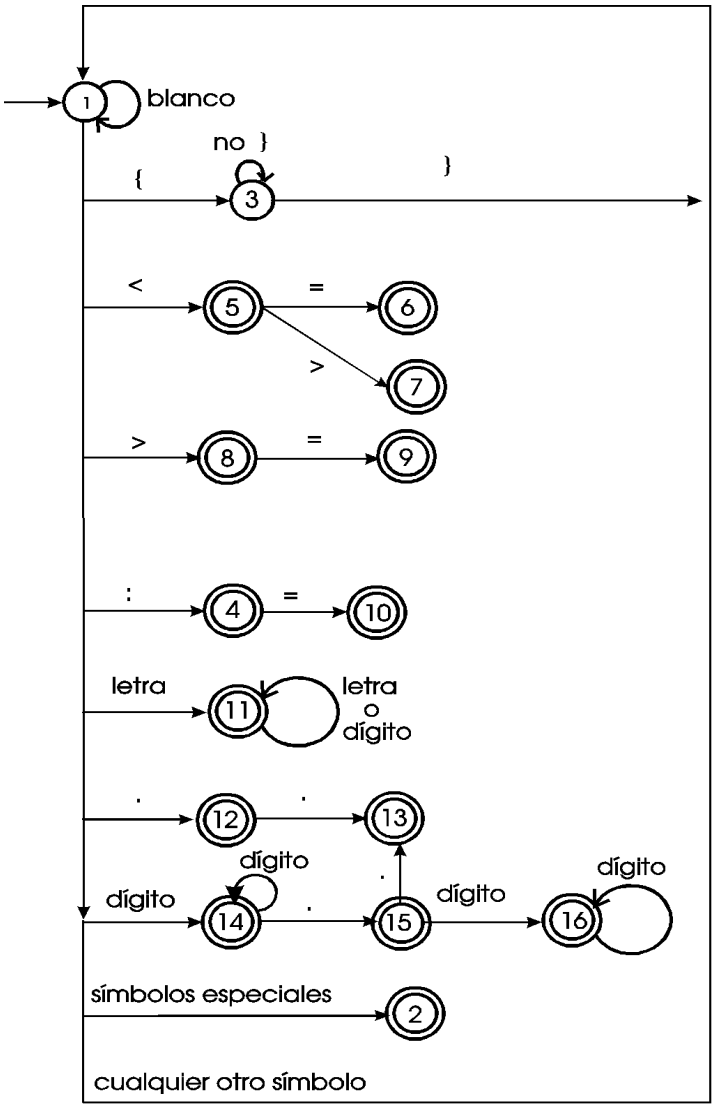

Figura 2.4. Autómata de estados finitos asociado con las expresiones regulares de la figura 2.3.

Análogamente, en PL/I, tampoco existen palabras reservadas con lo que es posible encontrar una instrucción como:

```
If then then then = else; else else = then
```

Como se puede apreciar, la detección de los símbolos resulta bastante complicada, y en PL/I está guiada por el AS.

Tratamiento de las palabras reservadas. Un problema específico es la detección de las palabras reservadas. Una primera solución puede ser considerarlas como cadenas individuales y

por tanto asignarles caminos específicos en el AEF. Esta solución coincide plenamente con lo que se ha visto hasta ahora; sin embargo el AL se hace muy complejo.

Una segunda solución es considerar a las palabras reservadas como casos particulares de los identificadores; de hecho, cumplen la definición de los mismos. El modo de trabajo de esta segunda solución es: Cuando se detecta un identificador, se compara con una tabla de palabras reservadas. Si la comparación tiene éxito se devuelve el "token" asociado a la palabra reservada detectada. Si no, se realizan las acciones relativas a la detección del "token" asociado a los identificadores.

Acciones asociadas a la detección de los símbolos.- Una vez detectado los símbolos es necesario realizar acciones semánticas asociadas con la detección de los mismos. En la Fig. 2.5 se muestra el código asociado al análisis léxico de los símbolos identificadores, palabras reservadas y constantes numéricas (definidos en las Fig. 2.3 y 2.4).

```
{obtch:Obtiene el sig. carácter del programa fuente y lo deposita en ch    }
{ch   Carácter actual                                                      }
{Longident  Longitud máxima establecida para los identificadores.          }
{sim        Símbolo detectado.                                             }
{ind        Indice de un identificador en la tabla de nombres.             }
{num        Valor de la constante numérica detectada.                      }
{tiponum    Tipo de la constante numérica detectada.                       }
{busca_nombre  Comprueba si el identificador está en la tabla de nombres.  }
{palabra_reservada Comprueba si la cadena detectada es un palabra reservada}

'A'..'Z':   begin       { identificador o palabra reservada }
      k:=0;
      repeat
            k:=k+1;
            if k <= LongIdent then ident[k] := ch;
            obtch;
      until not (ch in ['A'..'Z', '0'..'9']);
      if k > Longident then menerror(1)
      k := palabra_reservada(ident);
      if k>0 then sim:=simpr[k]
      else  sim:=identificador;     ind := busca_nombre (ident);
      end;

'0'..'9':   begin       {Constantes numéricas, enteras o reales}
      sim:=numero;      num:=0;
      repeat            {La constante es almacenada como un numero real}
            num:=10*num+(ord(ch)-ord('0'));     obtch;
      until not (ch in ['0'..'9']);
      if ch = '.' then begin
            obtch;      frac:=1;    decimal:=0; tiponum:=realsim;
            while ch in ['0'..'9'] do begin
                  decimal:=decimal*10+(ord(ch)-ord('0'));
                  frac:=frac*10; obtch;
                  end;
            num:=num+decimal/frac;
            end
      else  tiponum:=integersim;
      end
```

Figura 2.5. Código para el análisis léxico de identificadores, palabras reservadas y constantes numéricas.

Además de estas acciones es necesario hacer hincapié en algunos aspectos importantes:

- **Manipulación de la tabla de nombres**. Uno de los símbolos más frecuentes en los lenguajes de programación son los identificadores. Cuando el AL detecta un identificador es necesario almacenar el lexema correspondiente en la tabla de símbolos para su posterior uso. La gestión eficiente de este proceso se realiza en la denominada *tabla de nombres*.

- **Lectura del fichero de entrada**. La lectura del fichero de entrada carácter a carácter ocupa una gran parte del tiempo de compilación; por tanto, es necesario realizar esta tarea eficientemente.

- **Tratamiento de errores léxicos**. Los errores lexicográficos se producen cuando el AL no es capaz de generar un "token" tras aceptar una secuencia de caracteres. Sin embargo el número de errores detectables por un AL es muy escaso, reduciéndose a detectar caracteres ilegales y algún otro como el control sobre la talla de los identificadores, o el rango de las constantes numéricas.

La acción asociada con la detección del error es notificarlo y continuar con el proceso de análisis. Existen técnicas de recuperación de errores léxicos basadas en la modelización del error por medio de transformaciones de borrado, sustitución e inserción de caracteres. Sin embargo, estas técnicas no son usadas en compilación por su elevado coste computacional.

2.4. Generadores automáticos de analizadores léxicos

La implementación de un AL, tanto la detección de símbolos como la codificación de las acciones asociadas puede realizarse en algún lenguaje de programación, utilizando para ello las facilidades de entrada-salida que posea (Fig. 2.5). O bien, utilizar algún generador automático de analizadores léxicos como por ejemplo Flex. (Ver anexo A)

Flex acepta la especificación léxica de un lenguaje definida en términos de expresiones regulares, que definen los símbolos del lenguaje y rutinas C que codifican las acciones semánticas asociadas. El resultado es un módulo C (lex.yy.c) que contiene el AL correspondiente. En el anexo A puede encontrarse una introducción más extensa a Flex.

El lenguaje de especificación Flex consta de tres partes:

Declaraciones auxiliares
%%
Reglas
%%
Procedimientos auxiliares

En la zona de **declaraciones auxiliares**, se pueden definir expresiones regulares auxiliares. Las **reglas** son de la forma:

expresión regular {acción(es) en C}

La zona de **procedimientos auxiliares** pueden incluir funciones en C. En la Fig. 2.6, se puede observar un ejemplo de programa Flex para la especificación léxica de la Fig. 2.3.

```
%{/* definición de constantes empleadas MEN, MEI, IGU, DIF, MAY, MAI*/
%}

                  /* definiciones auxiliares */
delim       [ \t\n]
eb          {delim}+
letra       [A-Za-z]
dígito      [0-9]

%%

{eb}                                  {/* no hay acción ni se devuelve nada */}
{letra} ({letra} | {dígito})*         {yylval = instala_id(); return(ID);}
{dígito}+(\.{dígito}*)?               {yylval = instala_núm; return(NUMERO);}
"<"                                   {yylval = MEN; return(OPREL);}
"<="                                  {yylval = MEI; return(OPREL);}
"="                                   {yylval = IGU; return(OPREL);}
"<>"                                  {yylval = DIF; return(OPREL);}
">"                                   {yylval = MAY; return(OPREL);}
">="                                  {yylval = MAI; return(OPREL);}
":="                                  {return(ASIG);}

%%

instala_id() {
/* procedimiento para instalar el lexema, cuyo primer carácter está apuntado
por yytexto y cuya longitud es yylong, dentro de la tabla de símbolos y devuelve
un apuntador a él */
}

instala_núm() {
/* procedimiento similar para instalar un lexema que es un número */
}
```

Figura 2.6. Ejemplo de especificación léxica en Flex.