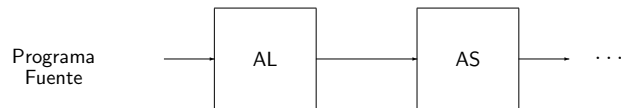



## 2. Análisis Léxico

- Introducción al problema del Análisis Léxico
- Formalismo de especificación léxica de los Lenguajes de Programación
- Construcción de un AL
- Generación automática de un AL



## Especificación léxica de un Lenguaje Natural (Castellano)

Lema	Definición	Instanciación
árbol	(Del lat. <i>arbor</i> , - <i>ōris</i> ) <b>1. m.</b> Planta perenne, de tronco leñoso y elevado, que se ramifica a cierta altura del suelo.	

## Especificación léxica de un Lenguaje de Programación

- **Lema** ⇒ Categoría sintáctica (o símbolo) del LP
- **Definición** ⇒ Patrones (definición) de los símbolos del LP
- **Lexema** ⇒ Cadena de caracteres que es una posible instanciación de un símbolo o categoría sintáctica

## Ejemplo de especificación léxica de un LP

Símbolos	Patrones de los símbolos	Lexemas
identificador	Cadena alfanumérica, con el primer carácter alfabético.	x25
constante entera	Cadena de uno o más dígitos	127
operador relacional	<, <=, >, >=, ==, !=	>
operador de asignación	=	=
...	...	...

## Objetivo del AL

Detectar e identificar las componentes léxicas presentes en la cadena (programa fuente) de entrada

### Componente léxica (o "token")

- Código del símbolo (o categoría sintáctica)
- Atributo(s) del símbolo (o categoría semántica)

### Ejemplo

$m = x > 2 \Rightarrow (\text{id}, \text{ind}_{(m)}) (\text{opasig}) (\text{id}, \text{ind}_{(x)}) (\text{oprel}, \text{cod}_{(>)}) (\text{cte}, \text{val}_{(2)})$

## Expresiones Regulares (Especificación)

- Dado un alfabeto  $\Sigma$ , una Expresión Regular (ER) sobre  $\Sigma$  es:
  - $\emptyset$  es una ER y define el lenguaje  $\emptyset$ ,
  - $\epsilon$  es una ER y define el lenguaje  $\{\epsilon\}$ ,
  - $a \in \Sigma$  es una ER y define el lenguaje  $\{a\}$ .
- Si  $r$  y  $s$  son ER, siendo  $L_r$  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  $L_r^*$ .
  - $r^+$  es una ER y define el lenguaje  $L_r^+$ .

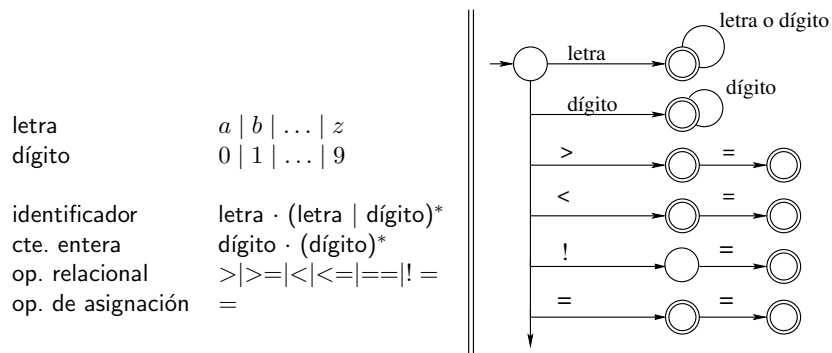
## Autómatas de Estados Finitos (Análisis)

$$AEF = (Q, \Sigma, \delta, q_0, F) \quad \text{Donde: } F \subseteq Q; \quad q_0 \in Q; \quad \delta : Q \times \Sigma \rightarrow \wp(Q)$$

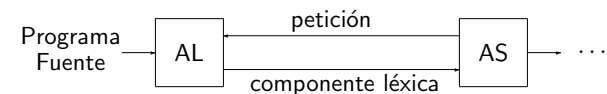
## Equivalencia ER $\Leftrightarrow$ AEF

ER  $\Leftrightarrow$  AEFND con transiciones  $\epsilon \Leftrightarrow$  AEFND  $\Leftrightarrow$  AEFD  $\Leftrightarrow$  AEFD mínimo

## Ejemplo



### 1. AL como una rutina del AS



### 2. Detección de los símbolos del lenguaje

#### 2.1 No es un problema tan fácil

- Ejemplo FORTRAN.- No existen palabras reservadas y el espacio en blanco no es un separador:

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

- Ejemplo PL/I.- No existen palabras reservadas:

if then then then = else; else else = then

## CONSTRUCCIÓN DE UN AL

### 2. Detección de los símbolos del lenguaje

#### 2.2 Detección de las palabras reservadas

- Palabras reservadas como expresiones regulares.
- Palabras reservadas como identificadores especiales (tabla de palabras reservadas).

### 3. Acciones asociadas a la detección de un símbolo

#### 3.1 Emitir las componentes léxicas ("tokens") detectadas

#### 3.2 Tratamiento de errores léxicos

#### 3.3 Manipulación de la Tabla de Símbolos

- Gestión de la Tabla de Símbolos (Lenguajes sencillos, p.ej. FORTRAN)
- Gestión de la Tabla de Nombres (Lenguajes complejos con estructura de bloques)

## CONSTRUCCIÓN DE UN AL

### 4. Otras funciones del AL

- eliminación de cadenas inútiles: comentarios, tabuladores, saltos de línea, etc.,
- lectura eficiente del fichero de entrada,
- relación de los mensajes de error con las líneas del programa fuente.
- reconocimiento y ejecución de las directivas de compilación.

### 5. Estrategias de implementación de un AL: de AEFD a programas

- Implementación manual directa del programa
- Implementación manual basada en la matriz de transiciones del AEFD
- Implementación basada en un generador automático de AL

## GENERADORES AUTOMÁTICOS DE AL: FLEX

**FLEX** es una herramienta de código abierto que convierte una descripción léxica de un LP, basada en expresiones regulares, en un programa C (o C++). El énfasis se pone en el rendimiento.

```
< declaraciones >
%%
< reglas de traducción >
%%
< procedimientos auxiliares >
```

- **Declaraciones.**- Definiciones de expresiones regulares auxiliares.
- **Reglas de traducción.**- Tienen la siguiente forma:

$$p_i \{ \text{acción}_i \} \quad i : 1 \dots n.$$

Donde  $p_i$  es la expresión regular que define un símbolo y  $\text{acción}_i$  es el segmento de programa con las acciones asociadas a la detección del símbolo.

- **procedimientos auxiliares.**- Segmentos de programa auxiliares.

## GENERADORES AUTOMÁTICOS DE AL: LEX

```
letra      [a-zA-Z]
digito     [0-9]
identificador letra(letra|digito)*
cteEntera  digito(digito)*
%%
"<"        { return(MENOR_); }
"<="       { return(MENORIG_); }
">"        { return(MAYOR_); }
">="       { return(MAYORIG_); }
"=="       { return(IGUAL_); }
"!="       { return(DISTINTO_); }
"="        { return(ASIG_); }
{identificador} { return(ID_); ind(); }
{cteEntera}    { return(CTE_); val(); }
%%
void ind()
/* Busca un nombre en la Tabla de Nombres, si no lo encuentra
   lo crea. Devuelve la posición en la Tabla de Nombres. */
...
void val()
/* Devuelve el valor numérico asociado al lexema. */
...
```