



Palabras clave : expresiones regulares, definiciones regulares, cerradura, alternancia, concatenación, alfabeto, lenguajes, autómatas.

# I

## INTRODUCCIÓN.

<b>1.1   ¿ POR QUÉ LENGUAJES ?   .....</b>	<b>2</b>
<b>1.2   COMPILACIÓN   .....</b>	<b>5</b>
<b>1.3   ¿ POR QUÉ AUTÓMATAS ?   .....</b>	<b>6</b>
<b>1.4   ALFABETO, CADENAS Y LENGUAJES   .....</b>	<b>10</b>
<b>1.5   REPRESENTACIÓN FINITA DEL LENGUAJE   .....</b>	<b>14</b>
<b>1.6   EJERCICIOS PROPUESTOS   .....</b>	<b>28</b>



Palabras clave : expresiones regulares, definiciones regulares, cerradura, alternancia, concatenación, alfabeto, lenguajes, autómatas.

## 1.1 ¿ PORQUÉ LENGUAJES ?.

Seguramente has escuchado, leído, o bien comentado acerca del “procesamiento electrónico de datos”, término que se utiliza para denotar al hecho de efectuar ciertas operaciones, tareas, desde simples hasta complicadas, teniendo como medio o herramienta una computadora, Fig. 1.1.



(a)



(b)

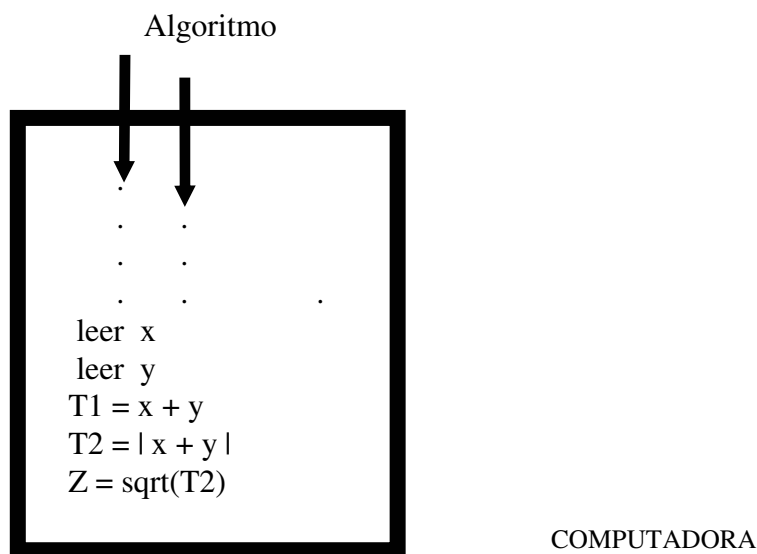


(c)

**Fig. 1.1** (a) Proceso electrónico de datos (b) Proceso electrónico de datos, relativamente simple (c) Proceso electrónico de datos, con operaciones mayores en cantidad y dificultad.

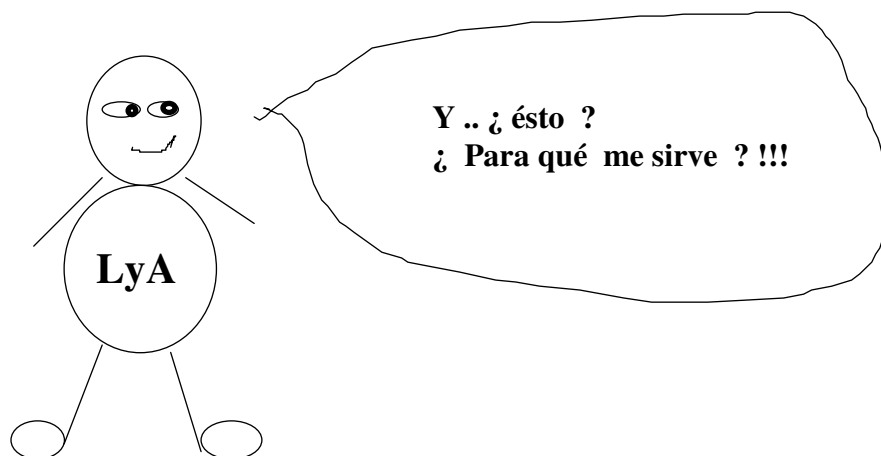


Hagamos la siguiente pregunta: ¿Cómo realiza la computadora, las operaciones citadas en la Fig. 1 (b) y (c) ?. Obviamente, tanto la obtención de la raíz cuadrada del valor absoluto de dos datos de entrada X y Y, como del reporte estadístico, representan un problema y éste es enfrentado, aplicando un algoritmo adecuado para su solución. Los algoritmos son introducidos a la computadora en forma de programas, Fig. 1.2.



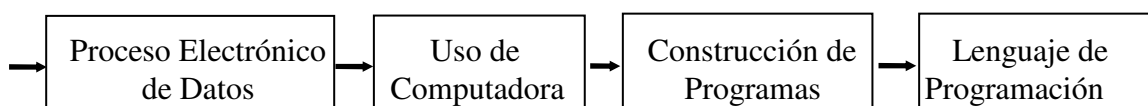
**Fig. 1.2** Programas.

Asimismo , un programa está formado de un conjunto de instrucciones escritas (codificadas) en un cierto ***lenguaje de programación***. Estas instrucciones manipulan los datos de entrada, (los reciben, los procesan), para convertirlos en información -“datos de salida”-.

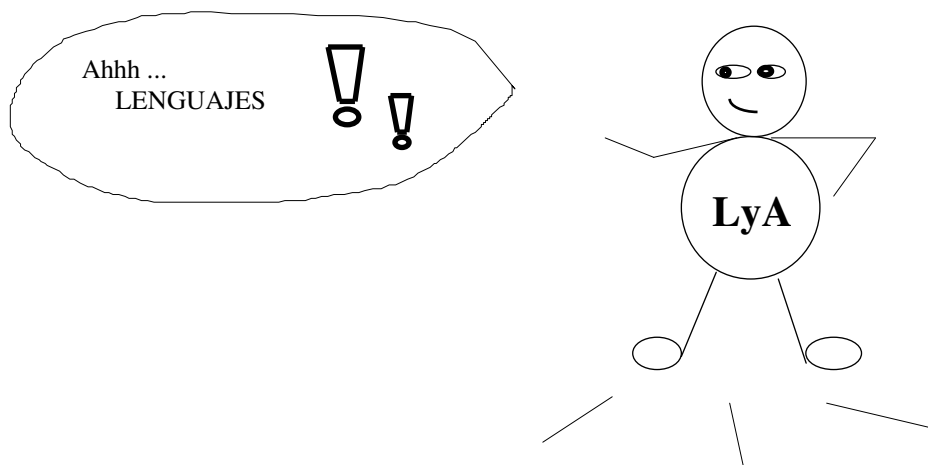




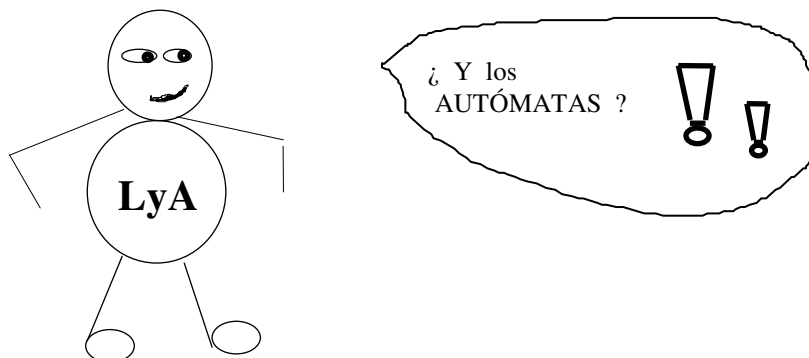
Así, el proceso electrónico de datos conlleva a la utilización de un computadora como recurso. La computadora requiere de programas y éstos a su vez, son entidades compuestas de instrucciones y datos, que se codifican en un determinado *lenguaje de programación*, Fig. 1.3.



**Fig. 1.3 .** La computadora como recurso en el P.E.D.



Nuestro estudio comprenderá, la teoría formal para especificar, representar, definir y reconocer lenguajes de cualesquier naturaleza. Especialmente, nuestro interés serán los **lenguajes de programación**.



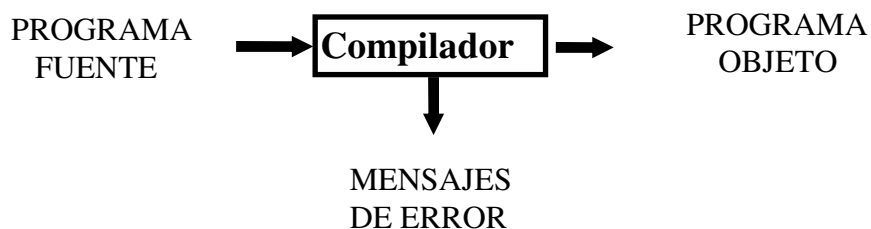


## 1.2 COMPILACIÓN.

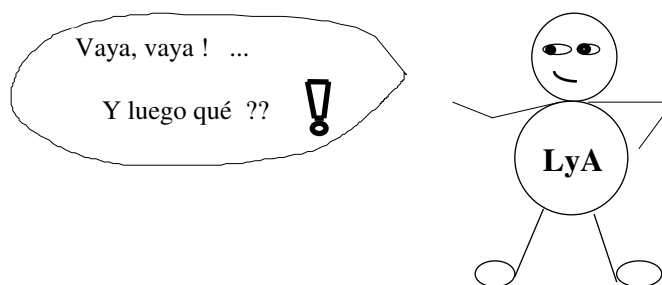
Supongamos que utilizamos un cierto lenguaje para expresarnos. Cualquier lenguaje en la construcción de las sentencias (instrucciones), requiere de observar ciertas reglas, denominadas *reglas de sintáxis*.

Si no respetamos las reglas de sintáxis del lenguaje, nuestras sentencias pueden no ser entendidas, es decir, estamos cometiendo errores al hacer uso de ese lenguaje.

Los *compiladores* son programas que se encargan de la tarea de revisar si un programa (programa fuente) de computadora, codificado (escrito) en un determinado lenguaje de programación, está libre de errores. Si así es, el compilador a su salida, nos proporciona un programa objeto, que generalmente es la traducción del programa fuente a código ensamblador o bien, un código ejecutable; de lo contrario nos informa con mensajes, de los errores encontrados, fig. 1.4.



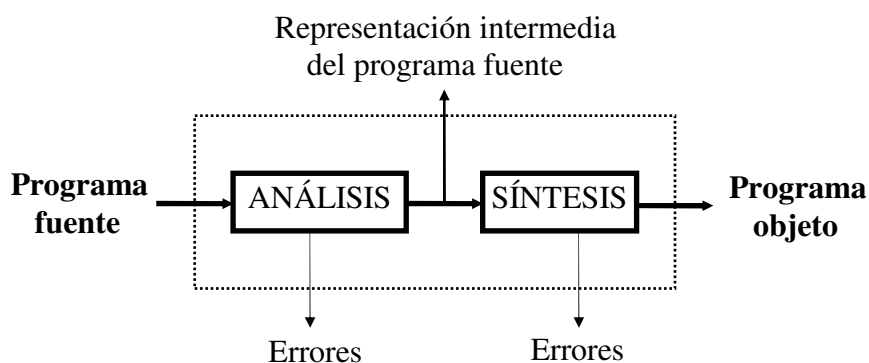
**Fig. 1.4** Función del compilador.





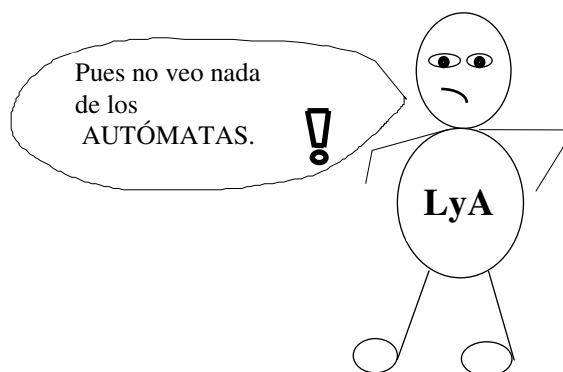
## 1.3 ¿ PORQUÉ AUTÓMATAS ?

El proceso de compilación de un programa se efectúa en dos fases: análisis y síntesis, fig. 1.5.



**Fig. 1.5** Fases de un compilador.

La fase de análisis tiene como entrada, el programa fuente. En base a este programa fuente, la fase de análisis construye una representación intermedia, ¿de quién?... pues del programa fuente, precisamente. En la fase de síntesis se toma de entrada a esta representación intermedia, para la generación del código que constituye al programa objeto.

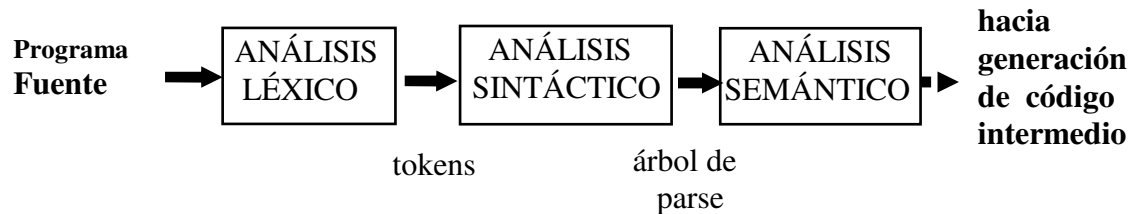


Ocupémonos de la fase de análisis. La fase de síntesis no tiene relevancia para efectos de este curso.

El análisis que toma al programa fuente como entrada, consiste a su vez de tres fases:

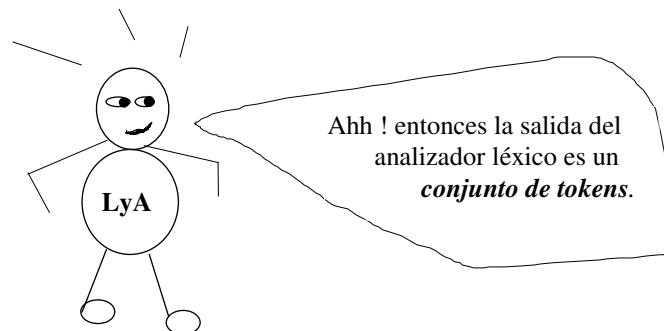
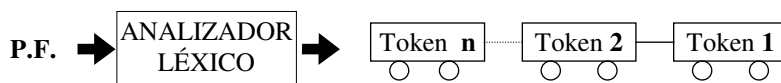


1. *Análisis léxico.*
2. *Análisis sintáctico.*
3. *Análisis semántico.*



**Fig. 1.6** Fases de análisis de un compilador.

La tarea principal del *análisis léxico* consiste en leer de izquierda a derecha, el programa fuente. El programa fuente es la entrada al analizador léxico y generalmente, reside en un archivo texto (los intérpretes de comandos reciben una o varias cadenas como entrada, desde el teclado). Este monitoreo de la entrada (programa fuente) lo efectúa el analizador léxico con el fin de identificar *tokens*, los cuales son cadenas o secuencias de caracteres que tienen un cierto significado.



En un lenguaje de programación, tenemos varias clases de tokens: *Palabras reservadas*, *Identificadores*, *Operadores aritméticos*, *Operadores relacionales*, *Operadores lógicos*,



*Constantes Literales (String), Números, Separadores, Operadores de asignación, Delimitadores, etc..*

Es responsabilidad del diseñador del lenguaje, definir cuántos y cuáles tokens formarán precisamente al lenguaje de programación en cuestión.

En la fig. 1.7 se muestra un fragmento de código en C y la descomposición de cada instrucción en los tokens que la forman.

INSTRUCCIÓN	TOKEN	LEXEMA	PATRÓN
iSuma = iSuma + 2 * x ;	id	iSuma	Letra seguida de cualesquier # de letras, dig. ó subrayado
	OpAsig	=	=
	id	iSuma	
	OpArit	+	+ ó - ó * ó / ó %
	num	2	dig dig ... dig
	OpArit	*	+ ó - ó * ó / ó %
	id	x	idem Suma
	TermInstr	;	;
printf("La suma es = %d\n", iSuma);	PalRes	printf	Palabra reservada o un archivo previamente definido
	Sep	(	( ó ) ó ,
	CteLit	"La suma es= %d \n"	Cualquier # de caracteres ASCII menos las " encerradas entre comillas
	Sep	,	( ó ) ó ,
	id	iSuma	
	Sep	)	( ó ) ó ,
	TermInst	;	;

**Fig. 1.7.** Token, Lexema y Patrón.

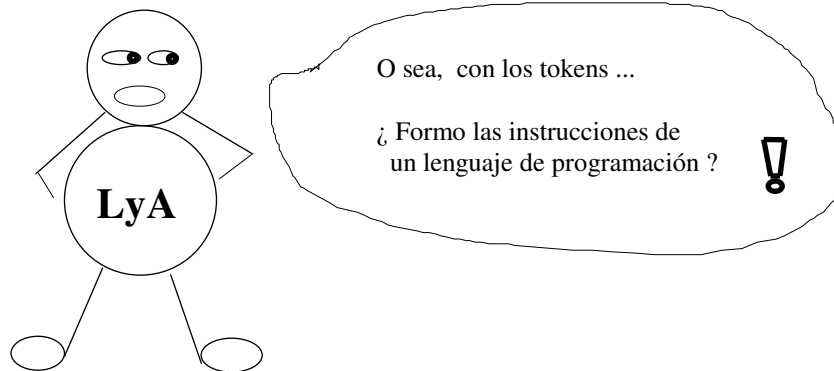
Los términos *token*, *lexema* y *patrón* que aparecen en la tabla de la fig. 1.7 tienen cada uno un significado especial, cuando hablamos de un análisis léxico. En general, existen un conjunto de cadenas en la entrada (programa fuente) para los cuales el mismo token es producido como salida (Ejemplo las cadenas, iSuma, x, producen el token *id*). A estas cadenas se les conoce como *lexemas* para un cierto token. Asimismo este conjunto de



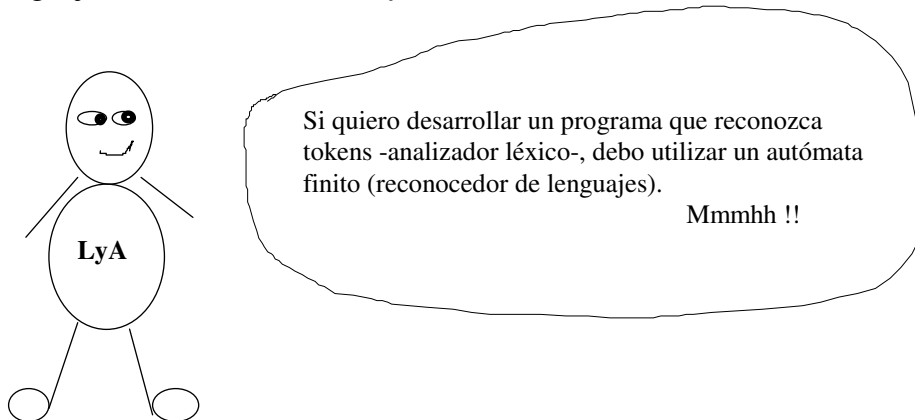


cadenas (lexemas) son descritas por una regla denominada *patrón*, el cual esta asociado con el *token*.

Si analizamos detalladamente las instrucciones de la fig. 1.7, encontraremos que éstas, en su totalidad, están “armadas” por tokens, es decir, cada instrucción es un conjunto de *tokens concatenados*.



Los tokens son especificados formalmente por medio de *expresiones regulares*. El reconocimiento de los tokens es hecho por el analizador léxico utilizando reconocedores de lenguajes, llamados *autómatas finitos*.



El *análisis sintáctico* recibe los tokens que le envía el analizador léxico y con ellos construye una estructura jerárquica, denominada *arbol de reconocimiento*. El analizador sintáctico verifica que una instrucción esté bien construida, es decir, que se hayan observado las reglas de sintáxis para cada instrucción, fig. 1.8.

a) `scanf ("%d",&iNum);`    b) `scanf ("%d"&iNum);`    c) `scanf ("Esp.Form.",ListaParam);`

**Fig.1.8.** a) Instrucción bien construida. b) Instrucción mal construída. c) Sintáxis del `scanf`.

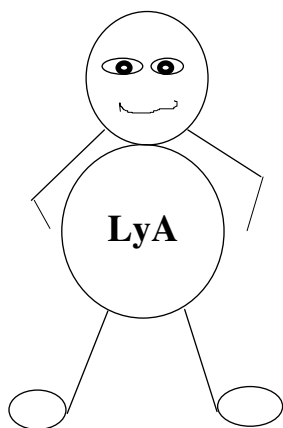


La instrucción en la fig. 1.8 (b) tiene error de sintáxis, ya que el token separador coma “,” no se encuentra separando la constante literal de la dirección del identificador.

La especificación formal de la sintáxis de una instrucción de un lenguaje, es posible realizarla mediante el uso de *gramáticas*. Las gramáticas de *contexto libre* son una clase de gramáticas que nos permiten especificar las reglas de sintáxis para casi todas las instrucciones de un lenguaje de programación.

En este curso de *lenguajes y autómatas* los temas principales a tratar son :

1. Representación finita de un lenguaje.
2. Expresiones regulares.
3. Gramáticas .
4. Autómatas.
5. Máquinas de Turing.



Caray !! ahora si parece que me quedó claro el *porqué* de este curso de :

**LENGUAJES y AUTÓMATAS.**

## 1.4 ALFABETO, CADENAS Y LENGUAJES.

El término **alfabeto** -también llamado vocabulario o clase caracter- denota a cualesquier conjunto finito de símbolos. En una computadora estos símbolos podrían ser los encontrados en el código ASCII, es decir el conjunto de caracteres ASCII es un alfabeto. Otros ejemplos:

*alfabeto binario*  $B = \{ 0,1 \}$

*alfabeto octal*  $O = \{ 0,1,2,3,4,5,6,7 \}$



Una **cadena** es una secuencia finita de símbolos tomados de un determinado alfabeto. En el estudio de los lenguajes, los términos *sentencia* y *palabra* son frecuentemente usados como sinónimos del término *cadena*.

Para una cadena se definen básicamente dos operaciones:

longitud    y  
concatenación.

La longitud de una cadena *s* usualmente escrita como  $|s|$ , es el número de ocurrencias de símbolos en *s*. En la tabla de la fig. 1.9 se muestran algunos ejemplos.

Cadena	Longitud
Hola	4
Autómatas	9
Peras	5
2	1

**Fig 1.9** Longitud de cadenas

La concatenación es una operación binaria cuyos dos operandos son cadenas. Sean  $\text{cad1} = \text{Hola}$     y     $\text{cad2} = \text{Todos}$ , la concatenación de  $\text{cad1}$  y  $\text{cad2}$  se escribe :

$\text{cad1 cad2} = \text{HolaTodos}$   
y su longitud es :  $|\text{cad1 cad2}| = 9$ .

La concatenación de  $\text{cad2}$  y  $\text{cad1}$  es :

$\text{cad2 cad1} = \text{TodosHola}$     y su longitud es  $|\text{cad2 cad1}| = 9$

Se observa que la concatenación no es conmutativa ya que :

$$\text{cad1cad2} \neq \text{cad2cad1}$$

Existe una cadena especial, denominada *cadena vacía* y se denota con el símbolo  $\epsilon$  y su característica consiste en que su longitud es 0.

$$|\epsilon| = 0$$



La cadena vacía  $\epsilon$  es el elemento identidad bajo la operación de concatenación. Lo anterior significa que si cad es una cadena entonces:

$$\text{cad} \epsilon = \epsilon \text{cad} = \text{cad}$$

**Ejemplo 1.1** Sean  $\text{cad1} = \text{Hola}$ ,  $\text{cad2} = \text{Todos}$ ,  $\text{cad3} = !$ . Obtener :

$$(a) \text{cad1 cad3 cad2} = \text{Hola!Todos}$$

$$\begin{aligned} (b) \epsilon \text{cad1} \epsilon \text{cad2} \epsilon &= \epsilon \text{Hola} \epsilon \text{Todos} \epsilon \\ &= \text{Hola} \epsilon \text{Todos} \epsilon \\ &= \text{HolaTodos} \epsilon \\ &= \text{HolaTodos} \end{aligned}$$

$$(c) \epsilon \epsilon \epsilon = \epsilon$$

$$\begin{aligned} (d) \epsilon \epsilon \text{cad3} \epsilon \epsilon \epsilon \text{cad1} &= \epsilon \epsilon ! \epsilon \epsilon \epsilon \text{Hola} \\ &= ! \epsilon \epsilon \epsilon \text{Hola} \\ &= \text{!Hola} \end{aligned}$$

El término **lenguaje** denota cualesquier conjunto de cadenas formadas con símbolos de un cierto alfabeto. Este término suele ser abstracto, como por ejemplo el lenguaje  $\{\epsilon\}$ , que es el conjunto cuyo único elemento es la cadena vacía. En la tabla de la fig. 1.10 se muestran las diferentes operaciones sobre lenguajes, algunas fundamentales y otras compuestas.

OPERACIÓN	DEFINICIÓN
Unión $L \cup M$	$L \cup M = \{s \mid s \text{ está en } L \text{ ó } s \text{ está en } M\}$
Concatenación $LM$	$LM = \{st \mid s \text{ está en } L \text{ y } t \text{ está en } M\}$
Potenciación $L^i$	$L^i = LL^2L^3 \dots L^{i-1}L^i$
Cerradura $L^*$	$L^* = \bigcup_{i=0}^{\infty} L^i$
Cerradura positiva $L^+$	$L^+ = \bigcup_{i=1}^{\infty} L^i$
$L?$	$L? = \bigcup_{i=0}^1 L^i$

$L^*$  denota “0 ó más concatenaciones de  $L$ ”

$L^+$  denota “1 ó más concatenaciones de  $L$ ”

$L?$  denota “0 ó una concatenación de  $L$ ”

**Fig 1.10** Operaciones sobre lenguajes



**Ejemplo 1.2** Sean los lenguajes  $A = \{0,1\}$ ,  $B = \{a,b,c\}$ ,  $C = \{1,2\}$ , obtener :

(a)  $A \cup B$

$$= \{0,1\} \cup \{a,b,c\} = \{0,1,a,b,c\}$$

(b)  $(BC) \cup A$

$$\begin{aligned} &= \{a,b,c\} \{1,2\} \cup \{0,1\} \\ &= \{a1,a2,b1,b2,c1,c2\} \cup \{0,1\} \\ &= \{a1,a2,b1,b2,c1,c2,0,1\} \end{aligned}$$

(c)  $A^*$

$$\begin{aligned} &= \{0,1\}^* = \{0,1\}^0 \cup \{0,1\}^1 \cup \{0,1\}^2 \cup \{0,1\}^3 \cup \dots \\ &= \{\epsilon\} \cup \{0,1\} \cup \{0,1\}\{0,1\} \cup \{0,1\}\{0,1\}\{0,1\} \cup \dots \\ &= \{\epsilon, 0,1\} \cup \{00,01,10,11\} \cup \{000,001,010,011,100,101,110,111\} \cup \dots \\ &= \{\epsilon, 0,1,00,01,10,11,000,001,010,011,100,101,110,111, \dots\} \end{aligned}$$

(d)  $(B^+ \cup C)^0$

$$= \{\epsilon\}$$

(e)  $(C^? A)^?$

$$\begin{aligned} &= (\{1,2\}^? \{0,1\})^? \\ &= ((\{1,2\}^0 \cup \{1,2\}^1) \{0,1\})^? \\ &= ((\{\epsilon\} \cup \{1,2\}) \{0,1\})^? \\ &= (\{\epsilon, 1,2\} \{0,1\})^? \\ &= \{0,1,10,11,20,21\}^? \\ &= \{0,1,10,11,20,21\}^0 \cup \{0,1,10,11,20,21\}^1 \\ &= \{\epsilon\} \cup \{0,1,10,11,20,21\} \\ &= \{\epsilon, 0,1,10,11,20,21\} \end{aligned}$$

(f)  $(C \cup A)^+$

$$\begin{aligned} &= (\{1,2\} \cup \{0,1\})^+ \\ &= \{1,2,0\}^+ = \{0,1,2\}^+ \\ &= \{0,1,2\}^1 \cup \{0,1,2\}^2 \cup \{0,1,2\}^3 \cup \dots \\ &= \{0,1,2\} \cup \{0,1,2\}\{0,1,2\} \cup \{0,1,2\}\{0,1,2\}\{0,1,2\} \cup \dots \\ &= \{0,1,2\} \cup \{00,01,02,10,11,12,20,21,22\} \cup \dots \\ &= \{0,1,2,00,01,02,10,11,12,20,21,22,000,001,002, \dots, 210,221,222, \dots\} \end{aligned}$$

$(C \cup A)^1$

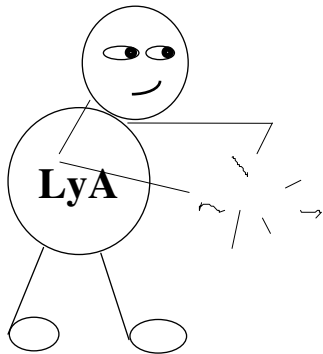
$(C \cup A)^2$

$(C \cup A)^3$



(g)  $(A \cup B?) \{ \in \}$

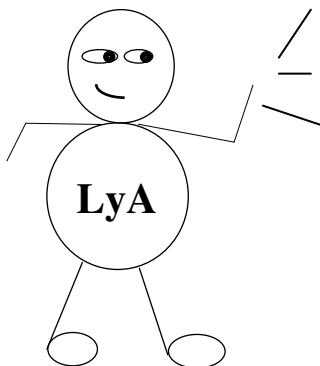
$$\begin{aligned} &= (\{0,1\} \cup \{a,b,c\}?) \{ \in \} \\ &= (\{0,1\} \cup (\{a,b,c\}^0 \cup \{a,b,c\}^1)) \{ \in \} \\ &= (\{0,1\} \cup (\{ \in \} \cup \{a,b,c\})) \{ \in \} \\ &= (\{0,1\} \cup \{ \in, a,b,c \}) \{ \in \} \\ &= \{0,1, \in, a,b,c\} \{ \in \} \\ &= \{ \in, 0,1,a,b,c \} \end{aligned}$$



**Uso de términos alfabeto,  
cadena, lenguajes y sus  
operaciones**

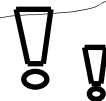
## 1.5 REPRESENTACIÓN FINITA DEL LENGUAJE.

En la sección 1.3 hemos revisado el concepto de token. Los tokens son cadenas de caracteres que tienen un cierto significado. Por ejemplo el Token Id (identificadores de variable y constantes en pascal) puede tener un sin fin de lexemas: iCont, X, asNumCon, sNomAlu, iCalif, iNum\_Elem, A\_\_T, Y11\_ZAZ, etc. En esencia, un token representa a un conjunto finito de cadenas, es decir, *un token representa a un lenguaje*. Las cadenas que forman parte de este lenguaje, cumplen con ciertas reglas. Estas reglas se les denomina patrón. El patrón que reglamenta a un token, puede especificarse utilizando *expresiones regulares*. Las *expresiones regulares* son una notación, que nos permite definir de manera precisa, al conjunto de cadenas que forman el lenguaje representado por un token. Una expresión regular es construida a partir de expresiones regulares simples, usando un conjunto de reglas bien definidas.



Un token representa un lenguaje.  
Una expresión regular se utiliza para definir a un token.  
Entonces :

**¿ Una expresión regular denota a un lenguaje ?**





Una expresión regular  $\underline{r}$  denota a un lenguaje  $L(r)$ .

Las reglas de definición para una expresión regular especifican clara y precisamente, como el lenguaje  $L(r)$  es formado o construido, por una combinación de lenguajes denotados a su vez, por subexpresiones regulares de  $\underline{r}$ .

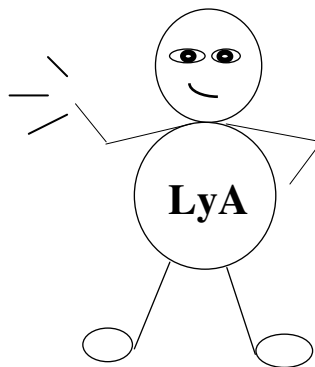
Reglas para expresiones regulares válidas sobre un alfabeto  $\Sigma$ .

1.  $\epsilon$  es una expresión regular que denota al lenguaje  $\{\epsilon\}$ , o sea, el conjunto que contiene solamente a la cadena vacía.
2. Si  $\underline{a}$  es un símbolo perteneciente al alfabeto  $\Sigma$ , entonces  $\underline{a}$  es una expresión regular que denota al lenguaje  $\{a\}$ , lenguaje que sólo contiene a la cadena  $\underline{a}$ .
3. Operaciones para expresiones regulares. Sean  $\underline{r}$  y  $\underline{s}$  dos expresiones regulares que denotan al lenguaje  $L(r)$  y  $L(s)$ , respectivamente. Entonces:
  - (a)  $(r) \mid (s)$  es una expresión regular que denota al lenguaje  $L(r) \cup L(s)$ . A esta operación se le conoce como *alternancia*.
  - (b)  $(r)(s)$  es una expresión regular que denota al lenguaje  $L(r)L(s)$ . Operación *concatenación*.
  - (c)  $(r)^*$  es una expresión regular que denota al lenguaje  $(L(r))^*$ . Operación *cerradura*.
  - (d)  $(r)$  es una expresión regular que denota al lenguaje  $L(r)$ .

Además, tenemos dos operaciones derivadas de las anteriores :

- (e)  $(r)^+$  es una expresión regular que denota al lenguaje  $(L(r))^+$ . Puede expresarse como :  $r^+ = r^* r$ .
- (f)  $(r)?$  es una expresión regular que denota al lenguaje  $(L(r))^0 \cup (L(r))^1$ . Puede expresarse como :  $r? = r \mid \epsilon$ .

- Un lenguaje denotado por una expresión regular, es un conjunto regular.





- Podemos en expresiones regulares seguir ciertas convenciones al evaluar o indicar las operaciones entre ellas. Nosotros adoptaremos las siguientes:

- 1 El operador  $*$  tiene la más alta precedencia y es asociativo a la derecha.
- 2 La concatenación tiene la segunda más alta precedencia y es asociativa a la izquierda.
- 3 La alternancia tiene la más baja precedencia y es asociativa a la izquierda.

Los paréntesis innecesarios pueden eliminarse siguiendo estas convenciones, como por ejemplo :

$(a) | ((b)^* (c))$  es equivalente a :       $a | b^*c$ .

En la práctica, un token (un lenguaje) a menudo, es especificado no solamente por una expresión regular, sino por un conjunto de expresiones regulares. A este conjunto de expresiones regulares que definen a un token, se le denomina **definición regular**. Si  $\Sigma$  es un alfabeto de símbolos básicos, entonces una definición regular es una secuencia de definiciones de la forma:

$d_1 \Rightarrow r_1$   
 $d_2 \Rightarrow r_2$   
....  
 $d_n \Rightarrow r_n$

donde cada  $d_i$  tiene un identificador o nombre distinto a las demás, y cada  $r_i$  es una expresión regular sobre los símbolos en :

$\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$ .

Hagamos algunos ejemplos, donde especificaremos tokens (lenguajes) que cumplen con ciertas características.

**Ejemplo 1.3.**    *Obtener la definición regular para el token con las siguientes características:*

- Es un número entero (sin signo) par.
- El cero es considerado par.

*Lexemas:* 8, 0, 16, 7772, 14444, 222, 418, 1000, 000, 04, ...

$Dig \Rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$

$Par \Rightarrow 0 | 2 | 4 | 6 | 8$

$NumPar \Rightarrow ( Dig^* ) ( Par )$

El lenguaje representado por *NumPar* es la concatenación de  $Dig^*$  con *Par*.

**“Nota” :** Emplearemos la notación

$Dig \Rightarrow 0 | 1 | \dots | 9$

que es menos verbosa.





$$\text{NumPar} \Rightarrow \text{Dig}^* \text{Par}$$

Aplicamos la regla 2 y 3(a) para obtener el lenguaje de la expresión regular *Dig* :

$$\text{Dig} = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$$

Usaremos la notación 0-9 para indicar 0, 1, ... , 9.

Calculamos ahora  $\text{Dig}^*$  - regla 3(c) - :

$$\begin{aligned} \text{Dig}^* &= \{ 0-9 \}^* = \{ 0-9 \}^0 \cup \{ 0-9 \}^1 \cup \{ 0-9 \}^2 \cup \{ 0-9 \}^3 \cup \dots \\ &= \{ \epsilon \} \cup \{ 0-9 \}^1 \cup \{ 0-9 \} \{ 0-9 \} \cup \{ 0-9 \} \{ 0-9 \} \{ 0-9 \} \cup \dots \\ &= \{ \epsilon, 0-9 \} \cup \{ 00-99 \} \cup \{ 000-999 \} \cup \dots \\ &= \{ \epsilon, 0-9, 00-99, 000-999, \dots \} \end{aligned}$$

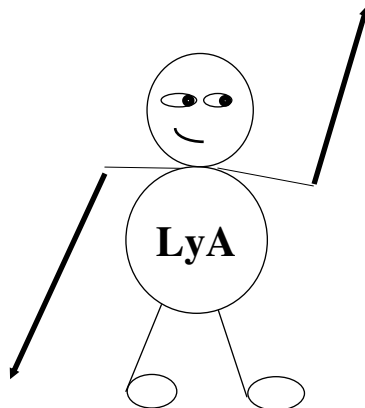
Por último, efectuamos la concatenación - regla 3(b) - :

$$\text{NumPar} \Rightarrow \text{Dig}^* \text{Par}$$

$$\text{Dig}^* \text{Par} = \{ \epsilon, 0-9, 00-99, 000-999, \dots \} \{ 0, 2, 4, 6, 8 \}$$

Concatenamos:

Todos contra todos



Aplicamos la regla 2  
y 3 (a)

$$\text{Dig}^* \text{Par} = \{ 0, 2, 4, 6, 8, 00-98, 000-998, 0000-9998, \dots \}$$

O sea, *NumPar* denota al lenguaje formado por las cadenas de dígitos que terminan en Par, 0, 2, 4, 6, 8.



**Ejemplo 1.4.** Repite la especificación del token *NumPar*, pero ahora eliminando los ceros (0) a la izquierda. El cero no es par.

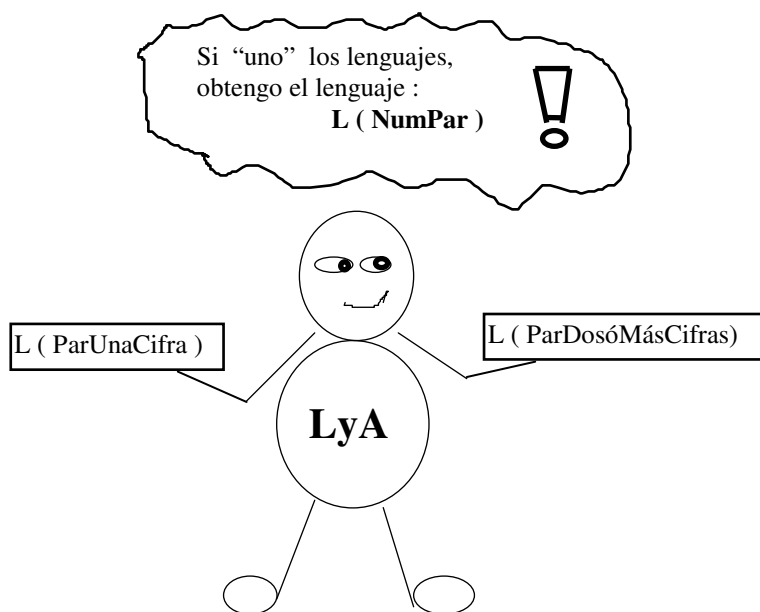
*Lexemas:* 4, 82, 10, 330006, 276, 4564, ...

La solución se obtiene con una definición regular que separe: 1) El lenguaje de los pares de una sola cifra, de donde excluirémos el cero { 2, 4, 6, 8 } y, 2) El lenguaje de los pares con dos o más cifras, y éstos, si pueden terminar en cero (0) además del 2, 4, 6, 8. ¡Hagámoslo!

$$\text{NumPar} \Rightarrow \text{ParUnaCifra} \mid \text{ParDosóMásCifras} \quad (1.4.1)$$

El lenguaje denotado por la expresión regular *NumPar*, es la unión (alternancia) de los lenguajes producidos por las expresiones regulares : *ParUnaCifra* y *ParDosóMásCifras*, es decir :

$$L(\text{NumPar}) = L(\text{ParUnaCifra}) \cup L(\text{ParDosóMásCifras})$$



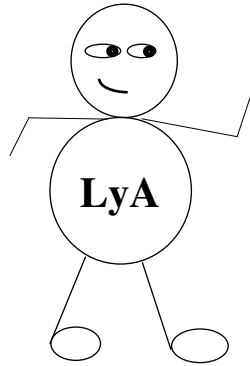
Para que la definición regular 1.4.1 esté completa, es necesario definir : *ParUnaCifra* y *ParDosóMásCifras*. La definición de la primera expresión regular es simple :

$$\text{ParUnaCifra} \Rightarrow 2 \mid 4 \mid 6 \mid 8$$

Y aplicando la regla 3(a):



$$L(\text{ParUnaCifra}) = \{ 2, 4, 6, 8 \}$$



Ya tengo ...

**L(ParUnaCifra)**

La definición regular para *ParDosóMásCifras* consiste de 3 subexpresiones regulares :

$$\text{ParDosóMásCifras} \Rightarrow (\text{Dig SinCero}) (\text{Dig})^* (\text{Par})$$

donde :

$$\text{DigSinCero} \Rightarrow 1|2|\dots|9 \quad // \text{ Todos los dígitos menos el cero.}$$

$$\text{Dig} \Rightarrow 0|1|\dots|9 \quad // \text{ Todos los dígitos se incluye al cero.}$$

$$\text{Par} \Rightarrow 0|2|4|6|8 \quad // \text{ Dígitos que dan la terminación par.}$$

Ahora, podemos obtener el lenguaje denotado por la expresión regular *ParDosóMásCifras*, el cual es la concatenación de los lenguajes denotados por :

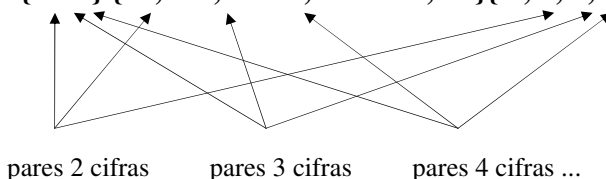
$$\text{ParDosóMásCifras} \Rightarrow (\text{DigSinCero}) (\text{Dig})^* (\text{Par})$$

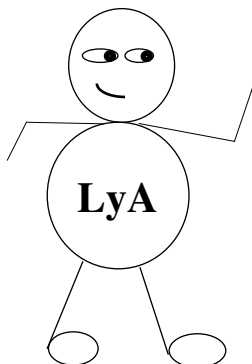
$$L(\text{ParDosóMásCifras}) = \{ 1, 2, \dots, 9 \} \{ 0, 1, 2, \dots, 9 \}^* \{ 0, 2, 4, 6, 8 \}$$

$$= \{ 1-9 \} ( \{ 0-9 \}^0 \cup \{ 0-9 \}^1 \cup \{ 0-9 \}^2 \cup \dots ) \{ 0, 2, 4, 6, 8 \}$$

$$= \{ 1-9 \} ( \{ \epsilon \} \cup \{ 0-9 \}^1 \cup \{ 0-9 \} \{ 0-9 \} \cup \dots ) \{ 0, 2, 4, 6, 8 \}$$

$$= \{ 1-9 \} \{ \epsilon, 0-9, 00-99, 000-999, \dots \} \{ 0, 2, 4, 6, 8 \}$$



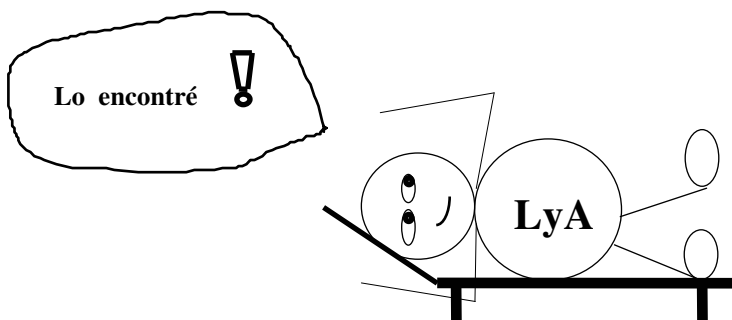


La concatenación obliga a  
2 cifras al menos .... !!!!!,

Ahora, uniré los dos lenguajes



$$\begin{aligned}
 L(\text{NumPar}) &= L(\text{ParUnaCifra}) \cup L(\text{ParDosóMásCifras}) \\
 &= \{2, 4, 6, 8\} \cup \{10-98, 100-998, 1000-9998, \dots\} \\
 &= \{2, 4, 6, 8, 10-98, 100-998, 1000-9998, \dots\}
 \end{aligned}$$



La definición regular pedida es :

<b>Dig</b>	$\Rightarrow 0   1   \dots   9$
<b>DigSinCero</b>	$\Rightarrow 1   2   \dots   9$
<b>Par</b>	$\Rightarrow 0   2   4   6   8$
<b>ParDosóMásCifras</b>	$\Rightarrow (\text{Dig SinCero}) (\text{Dig})^* (\text{Par})$
<b>ParUnaCifra</b>	$\Rightarrow 2   4   6   8$
<b>NumPar</b>	$\Rightarrow \text{ParUnaCifra}   \text{ParDosóMásCifras}$



**Ejemplo 1.5.** Encontrar una definición regular para el token que representa al conjunto de cadenas : Operadores relacionales en el lenguaje C.

Lexemas:

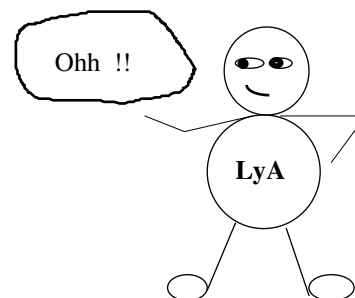
CADENA	LONGITUD DE CADENA
<	1
<=	2 Símbolos < y = concatenados
>	1
>=	2 Símbolos > y = concatenados
!=	2 Símbolos ! y = concatenados
==	2 Símbolos = y = concatenados

En este caso la solución es sencilla; apliquemos la regla 3(a) alternancia y la 3(b) concatenación:

**OpRel**  $\Rightarrow$  < | <= | > | >= | != | ==

Para comprobar la anterior definición regular, obtenemos el lenguaje denotado por la alternancia de cada expresión regular.

$$\begin{aligned}
 L(\text{OpRel}) &= L(<) \cup L(<=) \cup L(>) \cup L(>=) \cup L(!=) \cup L(==) \\
 &= \{<\} \cup \{<=\} \cup \{>\} \cup \{>=\} \cup \{!=\} \cup \{==\} \\
 &= \{<, <=, >, >=, !=, ==\}
 \end{aligned}$$



**Ejemplo 1.6.** Encuentra la definición regular para el token con las siguientes características :

- Las cadenas empiezan con al menos un dígito, y terminan en letra.
- El último dígito debe ser par y la primera letra debe ser vocal ( El cero se considera par ).

Lexema: 764a, 6E, 596432izlkam, 11118M, 0Abcxyz, ...

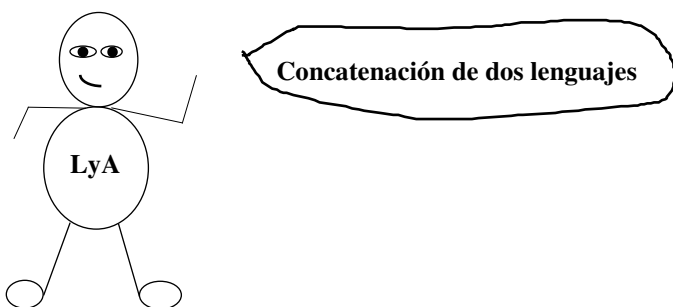


Analizando los lexemas, observamos que por medio de una concatenación es posible llegar a la solución. Dicha concatenación tiene como operandos : la secuencia de dígitos y la secuencia de letras.

$$\text{Token1.6} \Rightarrow (\text{SecDígitos}) (\text{SecLetras}) \quad (1.6.1)$$

y el lenguaje denotado por *token1.6* es :

$$L(\text{Token1.6}) = L(\text{SecDígitos}) L(\text{SecLetras})$$



Desglosemos *SecDígitos*. Esta secuencia de dígitos termina en par, lo que nos obliga a una concatenación :

$$\text{SecDígitos} \Rightarrow \text{Dig}^* \text{Par}$$

donde :

$$\text{Dig} \Rightarrow 0 \mid 1 \mid \dots \mid 9 \quad \text{y}$$

$$\text{Par} \Rightarrow 0 \mid 2 \mid 4 \mid 6 \mid 8$$

Luego, el lenguaje denotado por *SecDígitos* es :

$$\begin{aligned} L(\text{SecDígitos}) &= (L(\text{Dig}))^* L(\text{Par}) \\ &= \{0-9\}^* \{0,2,4,6,8\} \\ &= \{ \epsilon, 0-9, 00-99, 000-999, \dots \} \{0,2,4,6,8\} \quad // \text{ Todos los pares de una} \\ &\quad \text{o más cifras,} \\ &\quad \text{empezando o no, con} \\ &\quad \text{cero.} \end{aligned}$$

La secuencia de letras, obliga a una concatenación de una vocal con cualesquier número de letras, inclusive ninguna.

$$\text{SecLetras} \Rightarrow (\text{Vocal}) (\text{Letras})^*$$



donde :

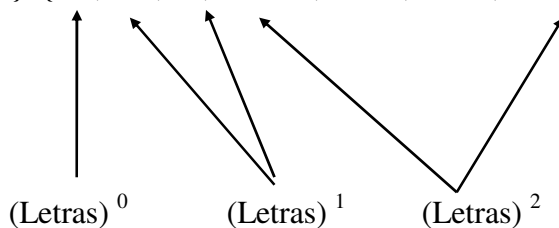
Vocal  $\Rightarrow A|a|E|e|I|i|O|o|U|u$

Letras  $\Rightarrow A|B|\dots|Z|a|b|\dots|z$

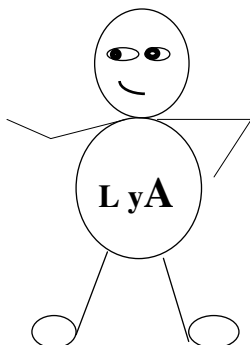
y el lenguaje denotado es :

$L(\text{SecLetras}) = L(\text{Vocal}) (L(\text{Letras}))^*$

$= \{A,a,E,e,I,i,O,o,U,u\} \{A-Z, a-z\}^*$   
 $= \{A,a,E,e,I,i,O,o,U,u\} (\{A-Z, a-z\}^0 \cup \{A-Z, a-z\}^1 \cup \{A-Z, a-z\}^2 \cup \dots)$   
 $= \{A,a,E,e,I,i,O,o,U,u\} (\{\epsilon\} \cup \{A-Z, a-z\} \cup \{A-Z, a-z\}\{A-Z, a-z\} \cup \dots)$   
 $= \{A,a,E,e,I,i,O,o,U,u\} \{\epsilon, A-Z, a-z, AA-ZZ, Aa-Zz, aA-zZ, aa-zz, \dots\}$



La concatenación  
obliga al inicio con  
vocal !!!



Concluimos con la definición regular que corresponde a la solución :

**Dig**  $\Rightarrow 0|1|\dots|9$   
**Par**  $\Rightarrow 0|2|4|6|8$   
**Vocal**  $\Rightarrow A|a|E|e|I|i|O|o|U|u$   
**Letras**  $\Rightarrow A|B|\dots|Z|a|b|\dots|z$   
**SecDígitos**  $\Rightarrow \text{Dig}^* \text{Par}$   
**SecLetras**  $\Rightarrow (\text{Vocal}) (\text{Letras})^*$   
**Token1.6**  $\Rightarrow (\text{SecDígitos}) (\text{SecLetras})$



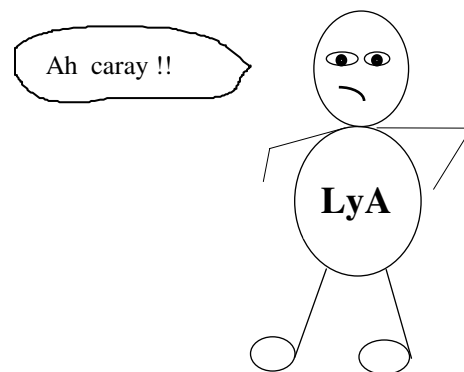
**Ejemplo 1.7** Encontrar la definición regular para el token con las siguientes características :

- *Números enteros con al menos una cifra.*

*Lexemas : 10, 0, 5, 476, 11111, 8965, ...*

La solución es :

**Dig**  $\Rightarrow$  **0 | 1 | ... | 9**      y  
**NumEnt**  $\Rightarrow$  **Dig<sup>+</sup>**



Para comprobar, obtenemos :

$$\begin{aligned}
 L(\text{NumEnt}) &= (L(\text{Dig}))^+ \\
 &= \{0,1,\dots,9\}^+ = \{0-9\}^+ \\
 &= \{0-9\}^1 \cup \{0-9\}^2 \cup \{0-9\}^3 \cup \dots \\
 &= \{0-9\} \cup \{0-9\}\{0-9\} \cup \{0-9\}\{0-9\}\{0-9\} \cup \dots \\
 &= \{0-9, 00-99, 000-999, \dots\} \quad // \text{ cadenas de dígitos con al menos una cifra.}
 \end{aligned}$$

$\uparrow$                        $\nwarrow$                        $\nwarrow$   
 (Dig)<sup>1</sup>                      (Dig)<sup>2</sup>                      (Dig)<sup>3</sup>

Observa lo que sucede si en lugar de la cerradura positiva  $\text{Dig}^+$  utilizamos :

**NumEnt**  $\Rightarrow$  **Dig<sup>\*</sup>**

El lenguaje denotado ahora por la expresión regular *NumEnt* es :

$$\begin{aligned}
 L(\text{NumEnt}) &= (L(\text{Dig}))^* \\
 &= \{0-9\}^* \\
 &= \{0-9\}^0 \cup \{0-9\}^1 \cup \{0-9\}^2 \cup \{0-9\}^3 \cup \dots \\
 &= \{\epsilon\} \cup \{0-9\} \cup \{0-9\}\{0-9\} \cup \{0-9\}\{0-9\}\{0-9\} \cup \dots
 \end{aligned}$$

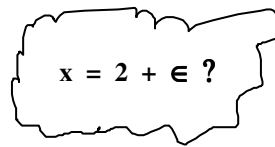


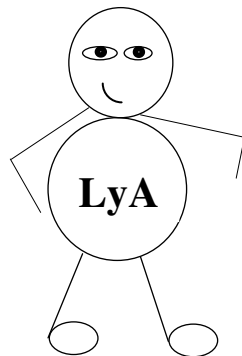


$$= \{ \epsilon, 0-9, 00-99, 000-999, \dots \}$$



¿!! Existe un entero sin dígitos o sea, la cadena vacía .. ?!!!


$$x = 2 + \epsilon ?$$



**Ejemplo 1.8** Supongamos que hemos obtenido un conjunto de cadenas de un texto. Algunos de los lexemas representativos de este lenguaje son los que a continuación se listan.

*Lexemas :* Al953bcz880,    kep17731fgxtp4,    opaMKss6204,    zz9975ThL02286, ...

(1)

(2)

(3)

(4)

*Encontrar una definición regular que represente a este lenguaje.*

Observamos en los lexemas dados de ejemplo, 4 secuencias concatenadas :

- a) Secuencia de letras ( al menos una letra ).
- b) Secuencia de dígitos noes. Esta secuencia puede o no presentarse en algunos lexemas (lexema 3).
- c) Secuencia de letras consonantes, al menos una.
- d) Secuencia de dígitos pares, al menos uno.



De lo anterior, el token es especificado con la siguiente definición regular :

**Letras**      $\Rightarrow$  [A-Za-z]  
**Cons**        $\Rightarrow$  [B-DF-HJ-NP-TV-Zb-df-hj-np-tv-z]    // donde : B-D = B | C | D  
**Non**         $\Rightarrow$  1 | 3 | 5 | 7 | 9  
**Par**          $\Rightarrow$  0 | 2 | 4 | 6 | 8  
**Token1.8**    $\Rightarrow$  ( Letras )<sup>+</sup> ( Non )<sup>\*</sup> ( Cons )<sup>+</sup> ( Par )<sup>+</sup>

La comprobación del lenguaje denotado por Token1.8, L ( Token1.8 ), se deja de ejercicio al alumno.

**Ejemplo 1.9** *Encontrar la definición regular para especificar el token cuyas características son las siguientes :*

- *Inicia con al menos una letra y termina con un caracter que puede ser una vocal o bien, un dígito par.El cero se considera par.*
- *La letra que termina la secuencia inicial de letras, es una vocal.*
- *Entre la secuencia inicial de letras y el último caracter (vocal o par), puede o no existir una secuencia de al menos un dígito seguido del caracter : (dos puntos). Si existe la secuencia intermedia, el último dígito de ésta, debe ser non.*

*Lexemas : A4, abcd9961:i, xyEa, tsqaei7:8, U67103:I, ...*

(1)            (2)            (3)            (4)            (5)

La definición regular para este caso, consta de 3 subexpresiones regulares concatenadas :

**Token1.9**    $\Rightarrow$    ( SecLetras ) ( SecIntermedia ) ? ( CaracterTerminación )

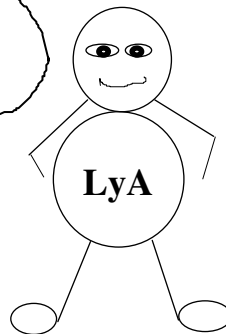
Dado que el lenguaje producido por la expresión regular *SecIntermedia* puede o no existir, se utiliza la siguiente operación :

*SecIntermedia* ?    $\Rightarrow$    *SecIntermedia* |  $\epsilon$

Recordemos :

**r?** = r |  $\epsilon$

Veamos la subexpresión regular *SecLetras* :





Letras  $\Rightarrow A|B|\dots|Z|a|b|\dots|z$

Vocal  $\Rightarrow A|a|E|e|I|i|O|o|U|u$

SecLetras  $\Rightarrow (\text{Letra})^* (\text{Vocal})$

La concatenación obliga a la terminación en vocal para las cadenas del lenguaje denotado por *SecLetras*,  $L(\text{SecLetras})$ .

La expresión regular *CaracterTerminación* es simple, y a continuación se muestra.

CaracterTerminación  $\Rightarrow \text{Vocal}|\text{Par}$

donde :

Par  $\Rightarrow 0|2|4|6|8$

El lenguaje denotado es :

$$\begin{aligned} L(\text{CaracterTerminación}) &= L(\text{Vocal}) \cup L(\text{Par}) \\ &= \{A,a,E,e,I,i,O,o,U,u\} \cup \{0,2,4,6,8\} \\ &= \{A,a,E,e,I,i,O,o,U,u,0,2,4,6,8\} \end{aligned}$$

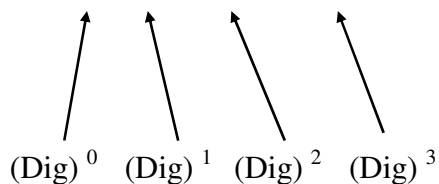
Sólo queda por obtener la expresión regular *SecIntermedia*. Sabemos que al menos hay un dígito y el último debe ser non, por lo que la condición obliga a una concatenación :

SecIntermedia  $\Rightarrow \text{Dig}^* \text{Non}$  :

donde : Non  $\Rightarrow 1|3|5|7|9$  ,

y el lenguaje denotado es :

$$\begin{aligned} L(\text{SecIntermedia}) &= (L(\text{Dig}))^* L(\text{Non}) L(:) \\ &= \{0-9\}^* \{1,3,5,7,9\} \{:\} \\ &= \{ \epsilon, 0-9, 00-99, 000-999, \dots \} \{1,3,5,7,9\} \{:\} \end{aligned}$$





Uniendo a las expresiones regulares anteriores, tenemos la definición que resuelve el problema :

<b>Letras</b>	$\Rightarrow A B \dots Z a b \dots z$
<b>Vocal</b>	$\Rightarrow A a E e I i O o U u$
<b>Par</b>	$\Rightarrow 0 2 4 6 8$
<b>Dig</b>	$\Rightarrow 0 1 \dots 9$
<b>Non</b>	$\Rightarrow 1 3 5 7 9$
<b>SecLetras</b>	$\Rightarrow (Letra)^*(Vocal)$
<b>CaracterTerminación</b>	$\Rightarrow Vocal Par$
<b>SecIntermedia</b>	$\Rightarrow Dig^*Non :$
<b>Token1.9</b>	$\Rightarrow (SecLetras)(SecIntermedia)^?(CaracterTerminación)$

## 1.6 EJERCICIOS PROPUESTOS.

1. *Identifica los tokens, lexema y patrón en los siguientes segmentos de código :*

(a) `function iSuma ( x : real ) : integer;`

(b) `Begin`

`x := y * ( 5 + x );`

`writeln ( 'x = ', x );`

`End;`

(c) `puts ( "Hola" );`

(d) `if ( iNum != 0 )`

`iNum = iNum * x;`

2. *Sean Cad1 = Hola, Cad2 = Todos, Cad3 = !, obtener :*

(a)  $(Cad1) \in (Cad2) \in (Cad3)$

(b)  $(Cad3) \in \in (Cad1) \in$

(c)  $\in \in \in$

(d) *¿Cuál es la longitud de las cadenas que resultan de las anteriores concatenaciones?*



**3.** Sean los lenguajes  $A = \{0,1\}$ ,  $B = \{a,b,c\}$ ,  $C = \{a,1\}$ , obtener :

- (a)  $A \cup B^0 \cup C$
- (b)  $A^2 \cup C$
- (c)  $AC^? \cup B^?$
- (d)  $(A^*)^0$
- (e)  $(B \cup C)^? A^?$
- (f)  $(A^+ \cup \{\epsilon\})$
- (g)  $(C^+)^?$

**4.** Sean los lenguajes  $K = \{x,y\}$ ,  $L = \{x,2,3\}$ ,  $M = \{\epsilon\}$ , obtener :

- (a)  $M^*(K \cup L)^?$
- (b)  $(L^? M^+)^?$
- (c)  $(K \cup M)^? (LM)$
- (d)  $(K^? M)^+$

**5.** Encuentra la expresión regular para el token con las siguientes características :

- Números con punto decimal, con al menos un dígito tanto a la izquierda como a la derecha del punto decimal.
- La cifra a la izquierda del punto debe ser non, y la cifra a la derecha del punto debe ser par. ( El cero es par ).

Ejemplos :

5.2, 1137.01, 22689.87771, ...



**6. Encuentra la definición regular para el token con las siguientes características :**

- Cadenas de dígitos ( solamente dígitos ), que empiezan con par y terminan con non.

Ejemplos :

81, 476543, 69999927, ...

**7. Encuentra la definición regular para el token con las siguientes características :**

- Empiezan con cualesquier número de letras ( puede que ninguna ).
- Siguen de las letras, cualesquier número de dígitos, pero siempre los últimos dos son el 2 y el 1.

Ejemplos :

AB21, 999921, klmxab111121, ...

**8. Encuentra la definición regular para el token con las siguientes características :**

- Empieza con letra consonante y termina en vocal, o bien empieza con vocal y termina en consonante.
- Entre las dos letras de inicio y fin puede o no existir una cadena de dígitos que empiezan con 2 o 3 y terminan en 6 o 9.

Ejemplos :

ka, E21118766h, z333445969u, is, ...

**9. Encuentra una definición regular para el token con las siguientes características :**

- Inicia con @ (arroba) seguido de comillas “, y termina con vocal. O bien, inicia con apóstrofe ‘ y termina con consonante.
- En ambos casos puede o no existir una secuencia intermedia que puede o no iniciar con dígitos seguidos de al menos una letra y terminando siempre en un dígito non.



Ejemplos :

@”A, @”765abc9E, ‘xyz5K, ‘6a7m, @”kl9A, ...

**10.** *Encuentra la definición regular para el token con las siguientes características :*

- Inicia con vocal o con dígito par, y le sigue siempre un 6.
- Termina en un caracter que puede ser las comillas “ o el caracter @.
- Entre el inicio ( después del 6 ) y el final ( antes de “ o @ ) se intercala - a veces no existe una secuencia de dígitos que empieza con non y termina con non. Esta secuencia puede tener uno o más dígitos.

Ejemplos :

A63”, 46”, 8634765@, E6155687”, u6@, i61111”, ...