



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

**ENERO JUNIO 2017**

CARRERA: ING. EN SISTEMAS COMPUTACIONALES

**MATERIA Y SERIE:  
LENGUAJES Y AUTÓMATAS I  
SCD-1015 SC6A**

**GRUPO  
A**

**TÍTULO: ANALIZADOR LÉXICO**

**UNIDADES:  
II, III, IV**

**TEMAS:**  
AUTÓMATAS FINITOS, EXPRESIONES REGULARES, ANALIZADOR LÉXICO

**NOMBRE DE MAESTRO: ESTRADA PEÑA ERASMO**

AYALA SANDOVAL JESUS EDUARDO #14211403

FECHA DE ENTREGA: 28/ABRIL/2017

## **Introducción**

La primera fase para realizar un compilador, es el escaneo de tokens dentro del archivo fuente. Cada token es una palabra reservada dentro del lenguaje, formado por un patrón de caracteres; cada uno de estos tokens pertenece a alguna categoría específica que lo representa, ya sea a él o a un grupo con tokens de las mismas características.

En este documento se da paso a explicar los temas relacionados a la formación de un analizador léxico, tales como los autómatas y expresiones regulares. De igual forma se anexa un analizador léxico escrito en ruby, utilizando expresiones regulares para el escaneo de los lexemas, ubicados en un archivo de texto que contiene las palabras también anexadas.

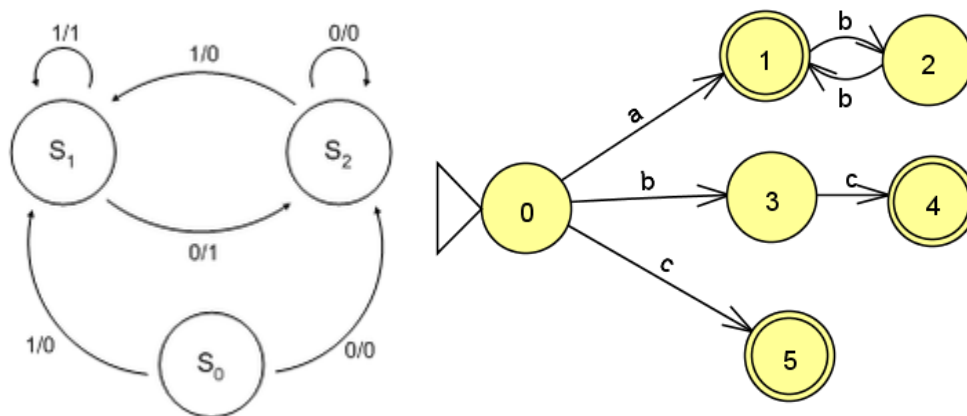
## Autómatas Finitos

Una máquina de estado finito (o autómata finito) es un modelo computacional que consiste en un conjunto de estados, un estado de inicio, un alfabeto de entrada y una función de transición a un siguiente estado, a partir del símbolo de entrada y el estado actual.

Los autómatas finitos pueden describir el proceso de reconocimiento de patrones en cadenas de entrada.

Este modelo está conformado por un alfabeto, un conjunto de estados finito, una función de transición, un estado inicial y un conjunto de estados finales. Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de caracteres pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.

El sistema recibe una cadena de símbolos de un alfabeto y determina si esa cadena pertenece al lenguaje que ese autómata reconoce. De esta manera se pueden construir analizadores léxicos, construyendo programas de computadora que realicen las operaciones de un autómata.



## Autómatas Finitos No Determinísticos

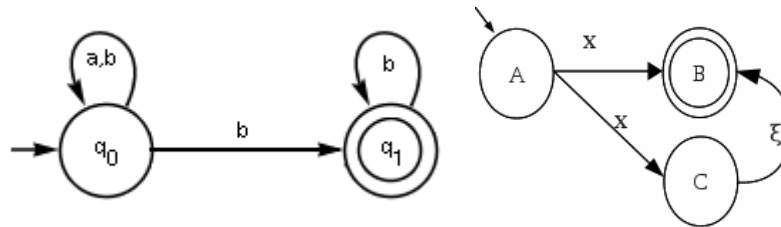
Es el autómata finito que tiene transiciones vacías o que por cada símbolo desde un estado de origen se llega a más de un estado destino.

Un Autómata finito no determinista queda definido por una quintupla, al igual que en los AFD, y que su diferencia fundamental se encuentra, en cómo se definirá a la función de transición:

$$\text{AFND} = (\Sigma, Q, q_0, F, f)$$

- $\Sigma$  Alfabeto de símbolos de entrada.
- $Q$  Conjunto finito de estados
- $q_0 \in Q$  – estado inicial previsto
- $F \subseteq Q$  - es el conjunto de estado finales de aceptación
- $f: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$

Para cada par estado entrada, el autómata puede tener la posibilidad de transitar a más de un estado posible. Puede realizar transiciones de un estado a otro sin leer símbolo alguno de la entrada. A este tipo particular de transiciones se las denomina transiciones. Para algún par de estado entrada, el autómata puede no tener definido ninguna transición. Lo que significa que podrá realizar transición alguna.



### Autómatas Finitos Determinísticos

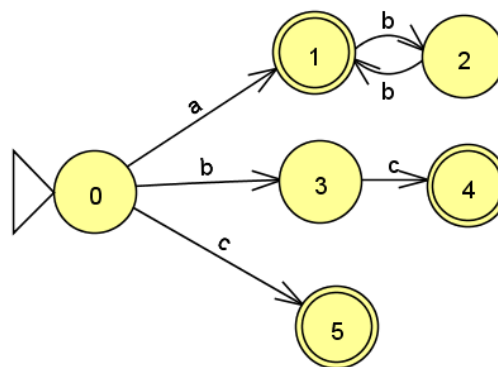
Un autómata finito determinista (DFA por sus siglas en inglés), es un modelo donde el siguiente estado de la transición está dado particularmente por el estado actual y el carácter de entrada actual. Si ningún estado de transición es especificado, la cadena entrante es rechazada.

Tiene todas sus transiciones no vacías y que por cada símbolo desde un estado de origen se llega a un único estado destino.

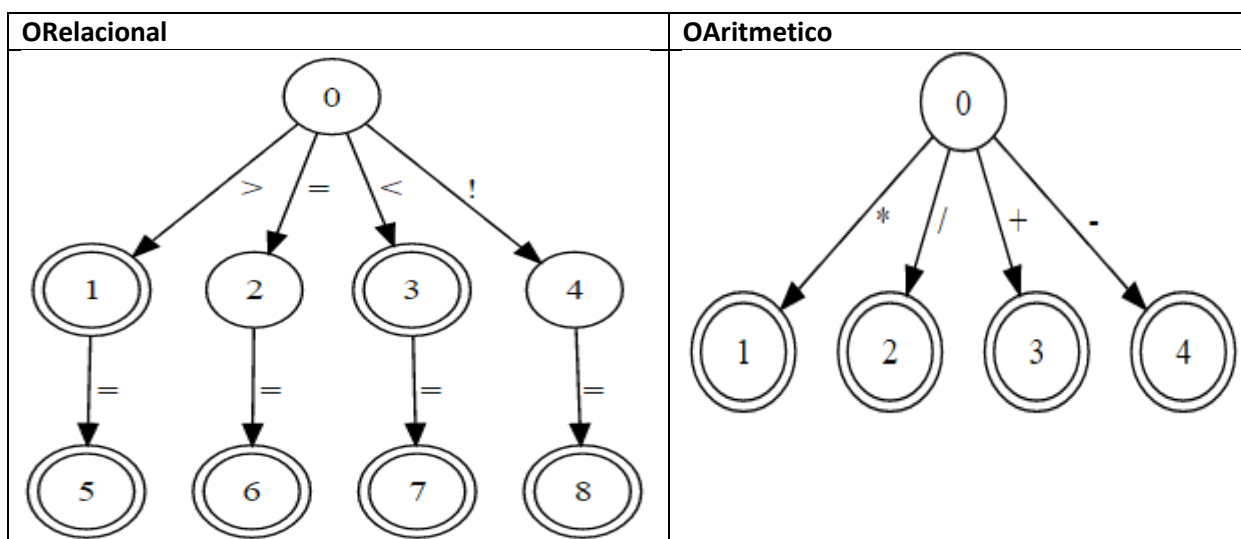
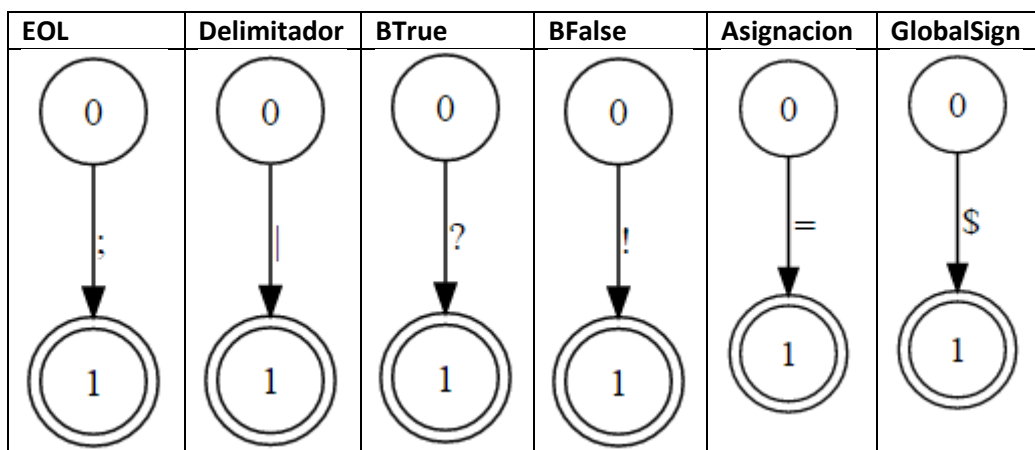
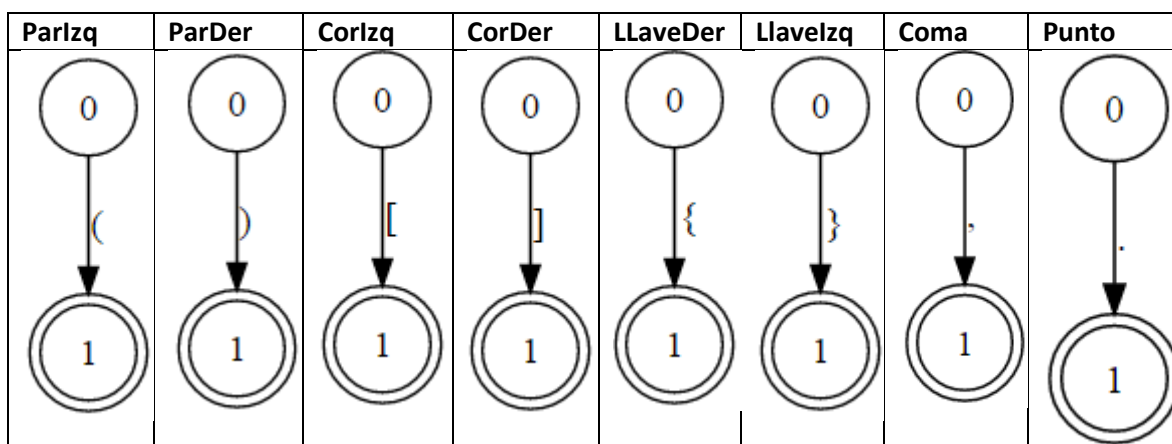
La siguiente definición formalmente introduce un DFA,  $(S, \Sigma, T, s, A)$  donde:

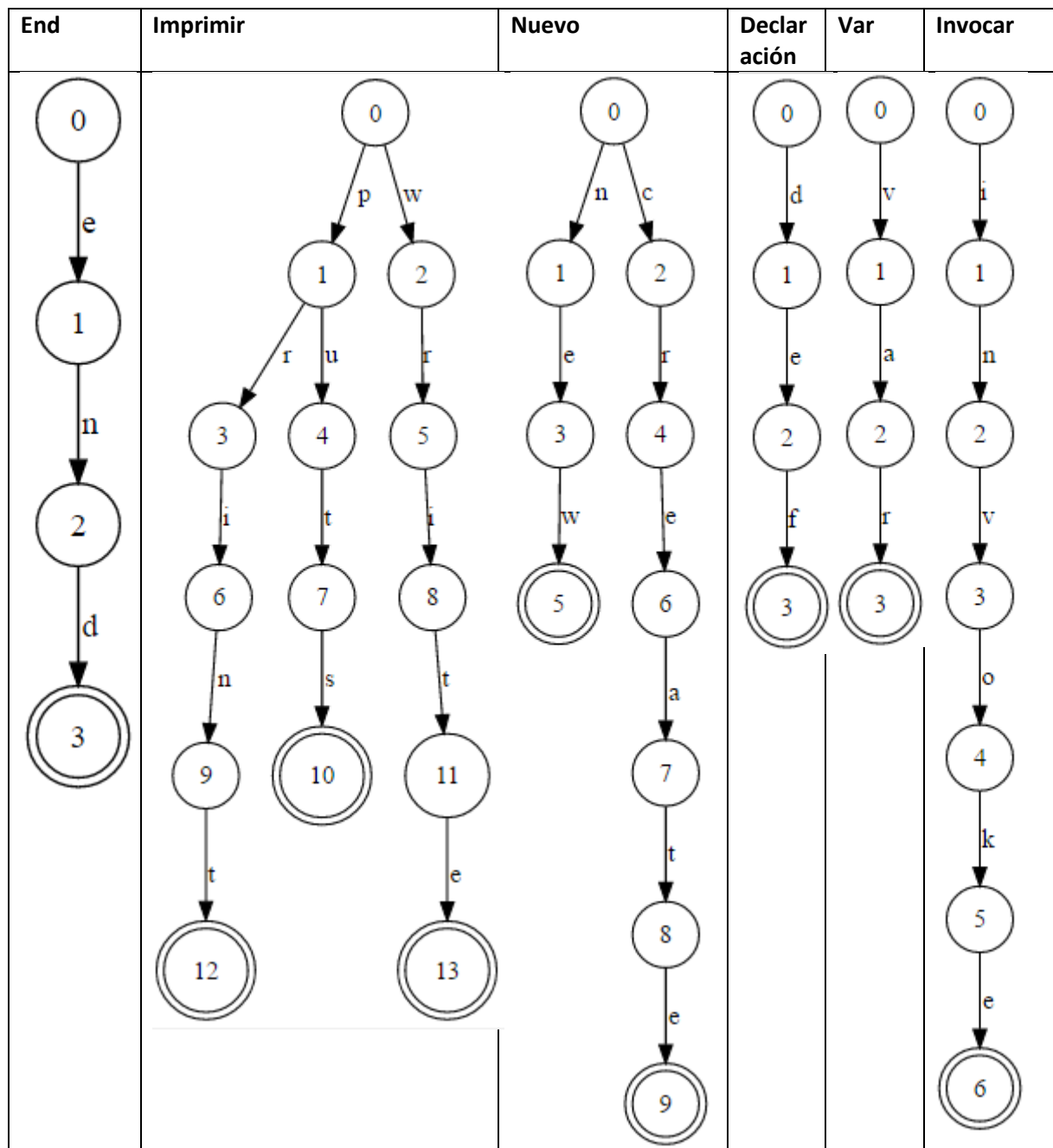
- $S$  conjunto finito no vacío de elementos llamado estados
- $\Sigma$  alfabeto de entrada
- $T$  es una función de transición de  $S \times \Sigma$  en  $S$
- $s \in S$  estado inicial
- $A \subseteq S$  conjunto no vacío de estados finales

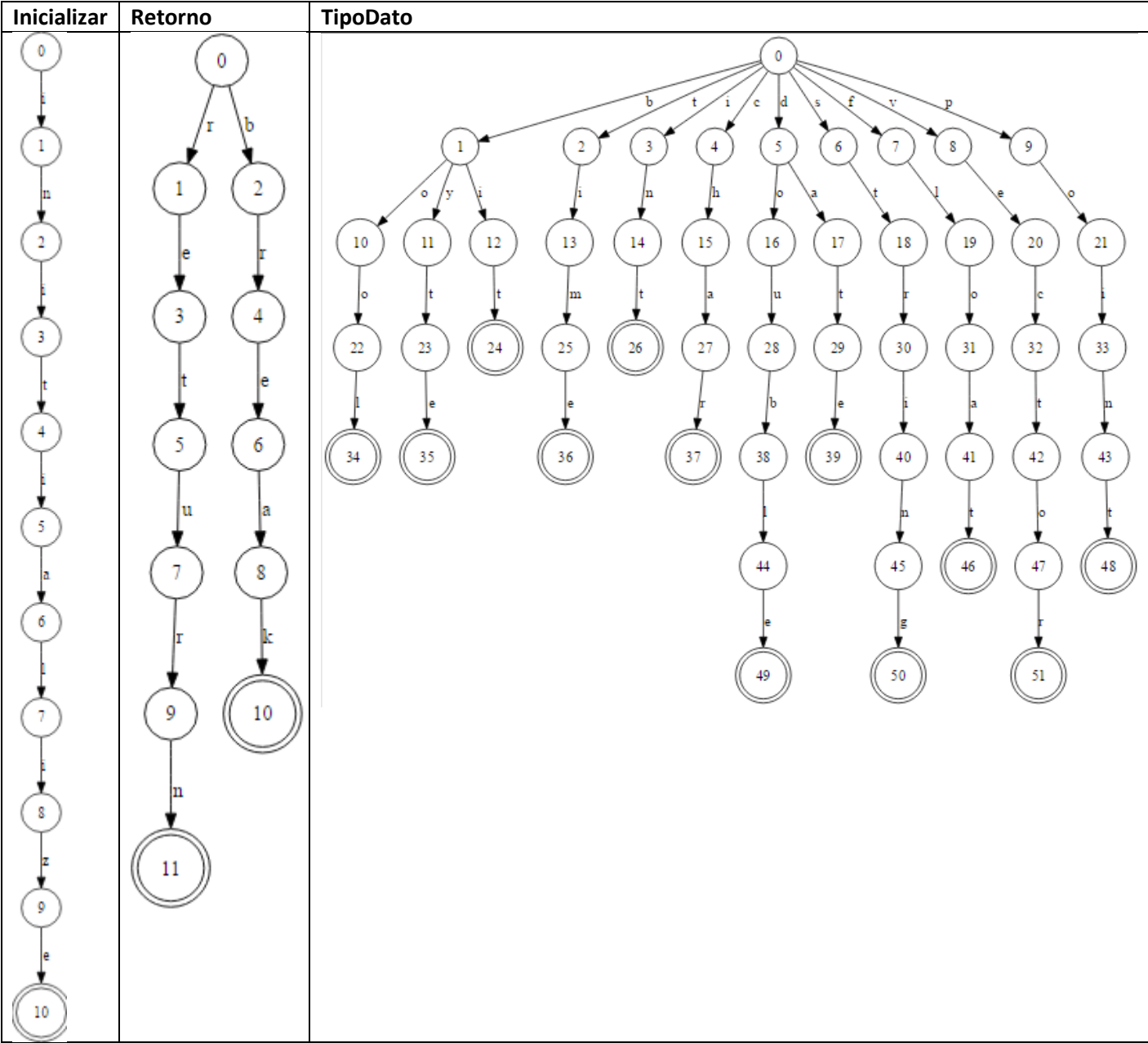
El ADF se encuentra en un estado inicial y lee una entrada de cadena de caracteres de izquierda a derecha. La función de transición  $T$  define los estados de transición y es denotada por  $T(s_i, c) = s_{i+1}$ , donde  $s_i$  y  $s_{i+1}$  son estados de  $S$ , y  $c$  es un carácter del alfabeto de entrada. La función de transición indica que cuando un autómata está en un estado  $s_i$  y recibe el siguiente símbolo de entrada  $c$ , el autómata deberá cambiar al estado  $s_{i+1}$ . El carácter de entrada no causará transición en caso de ser un carácter vacío.



## Autómatas de las expresiones regulares

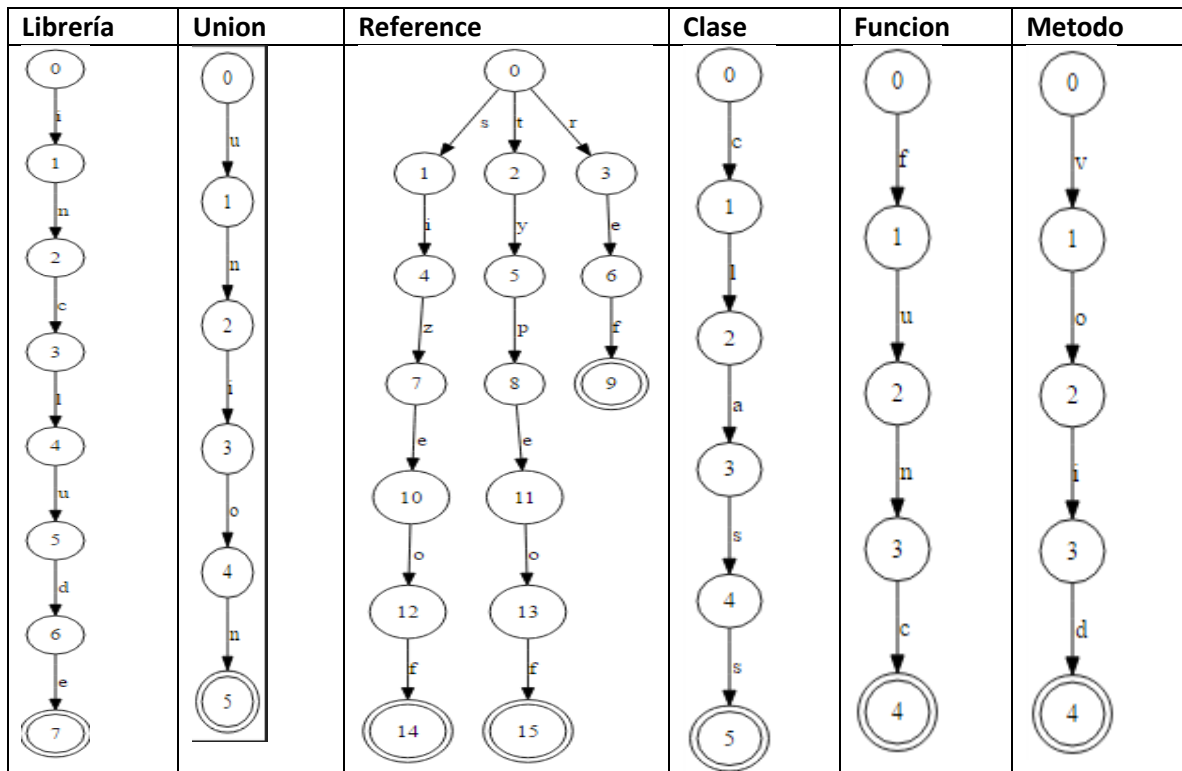
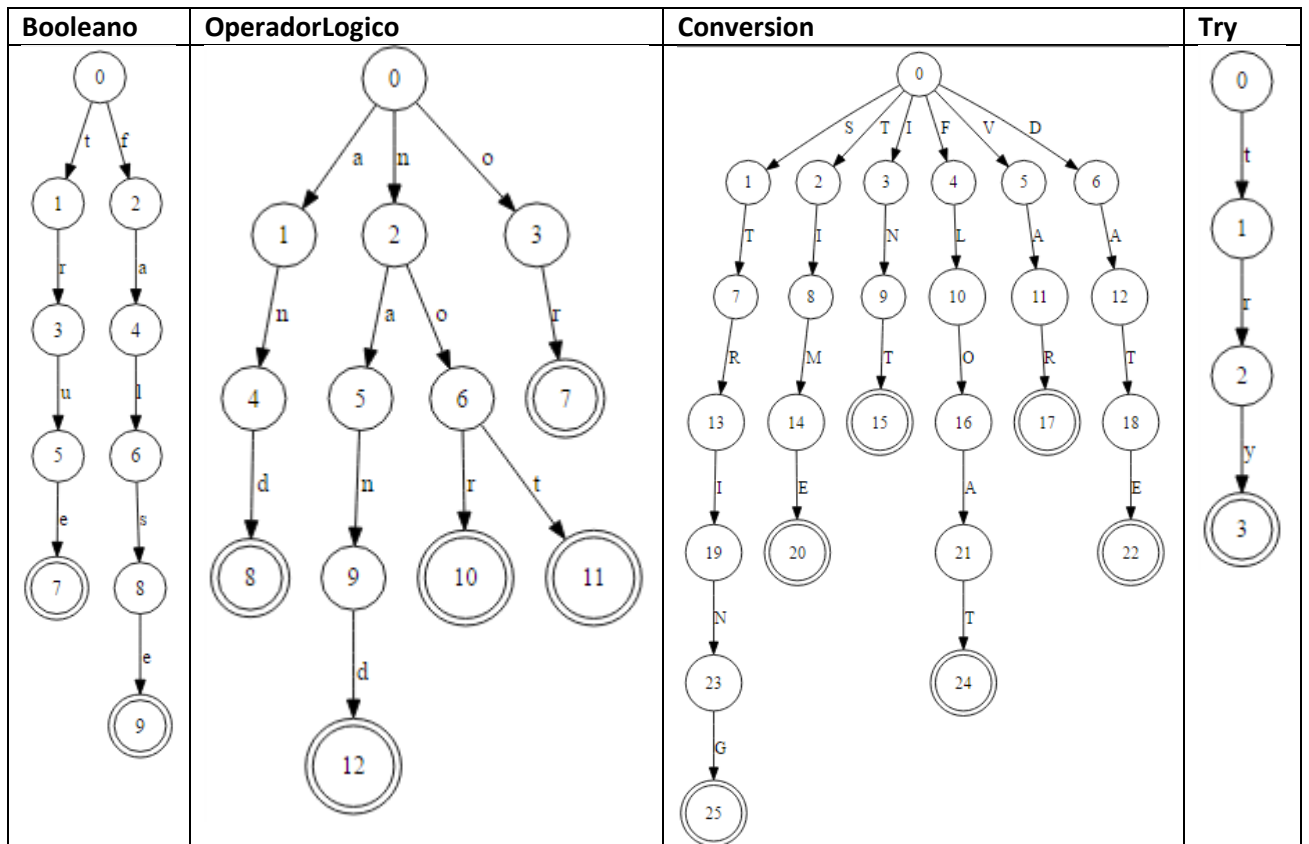


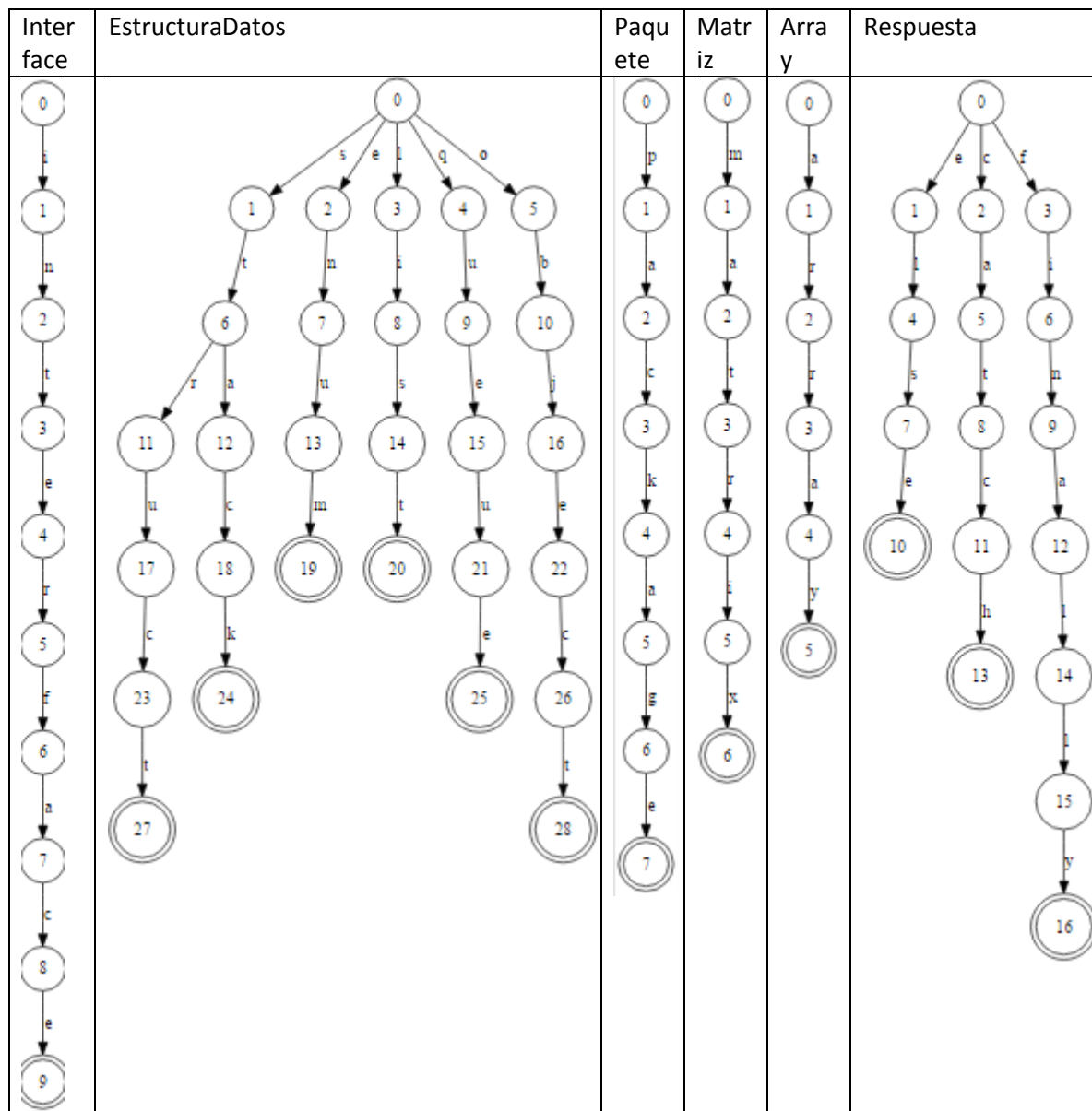


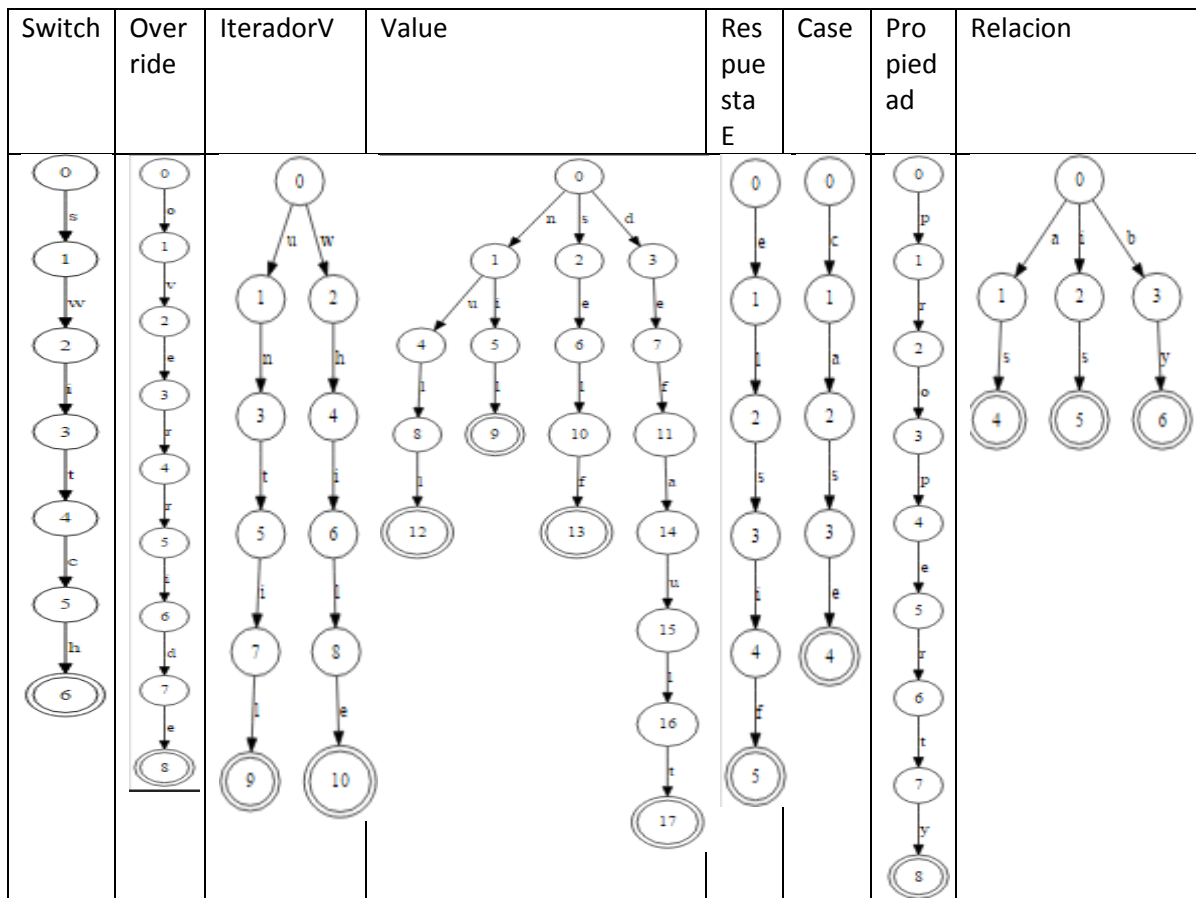
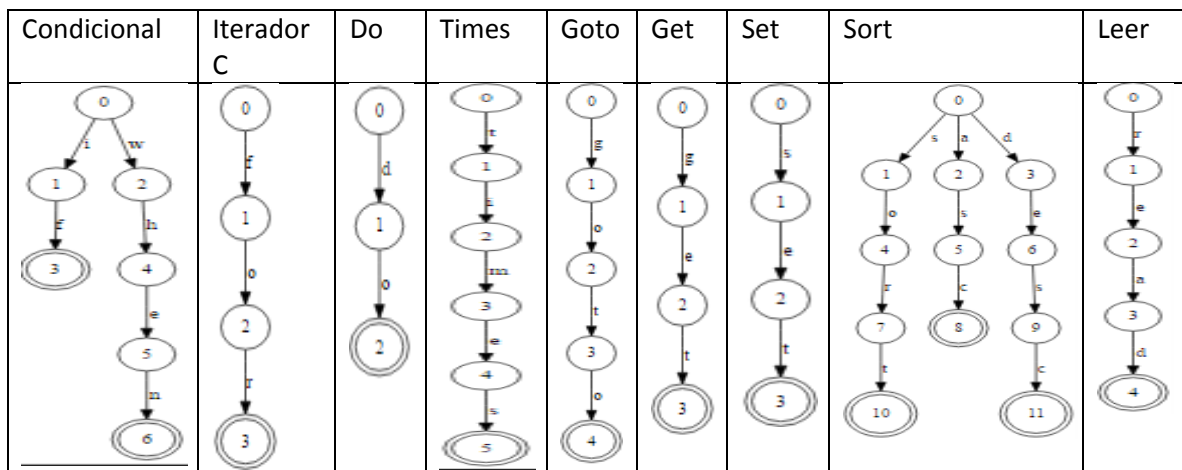


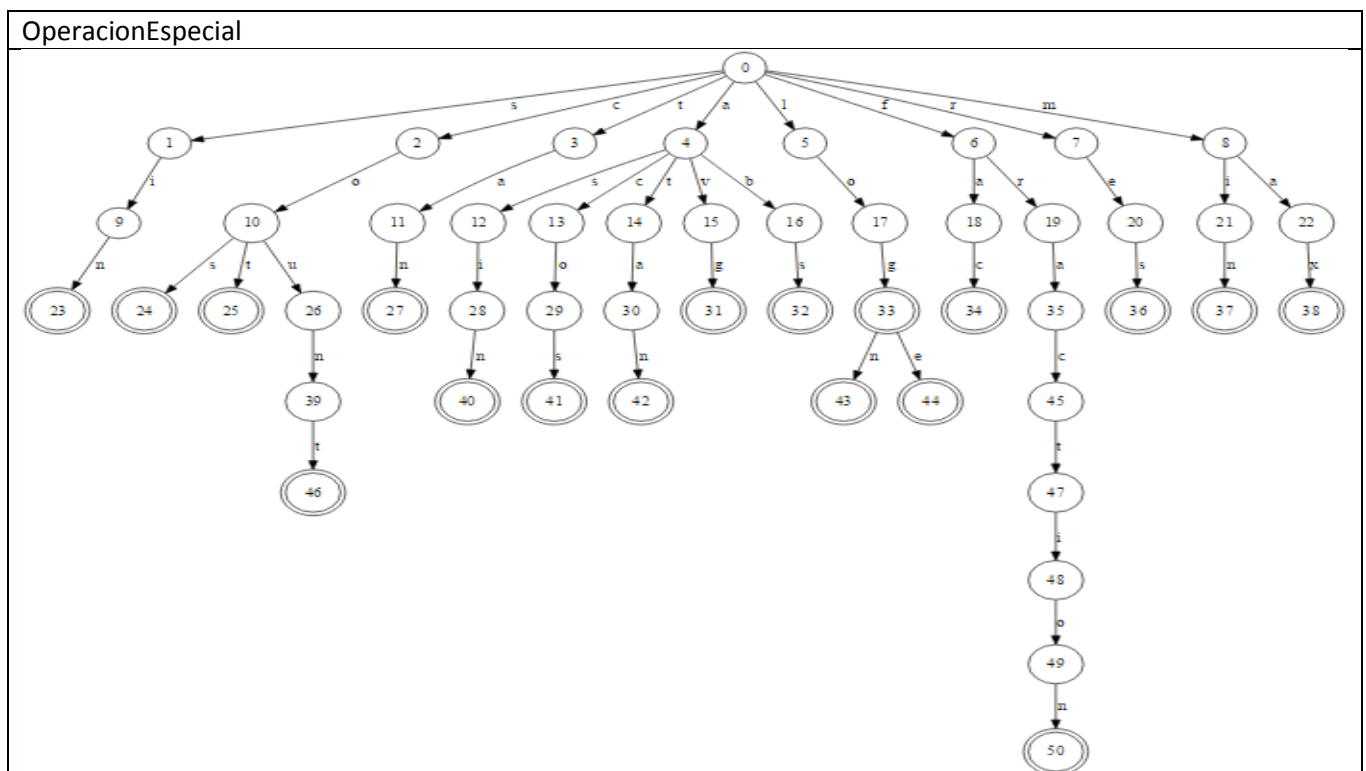
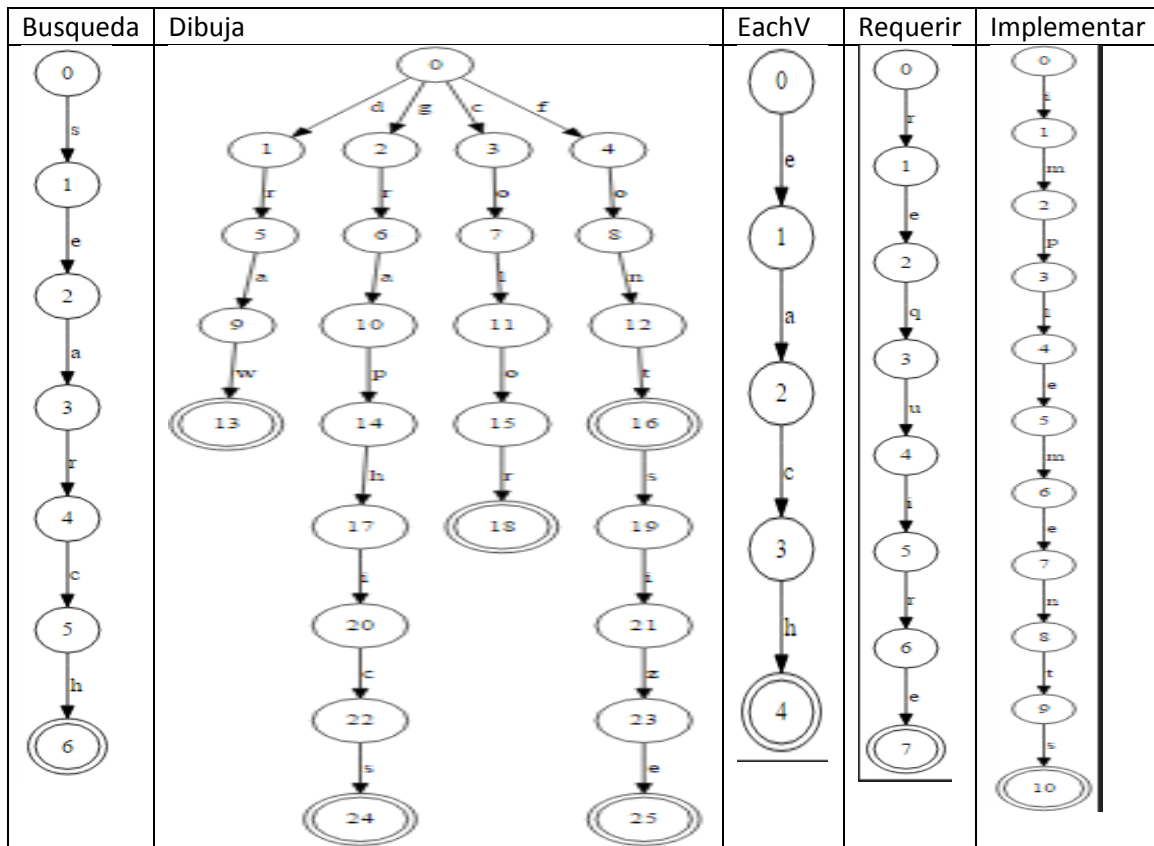




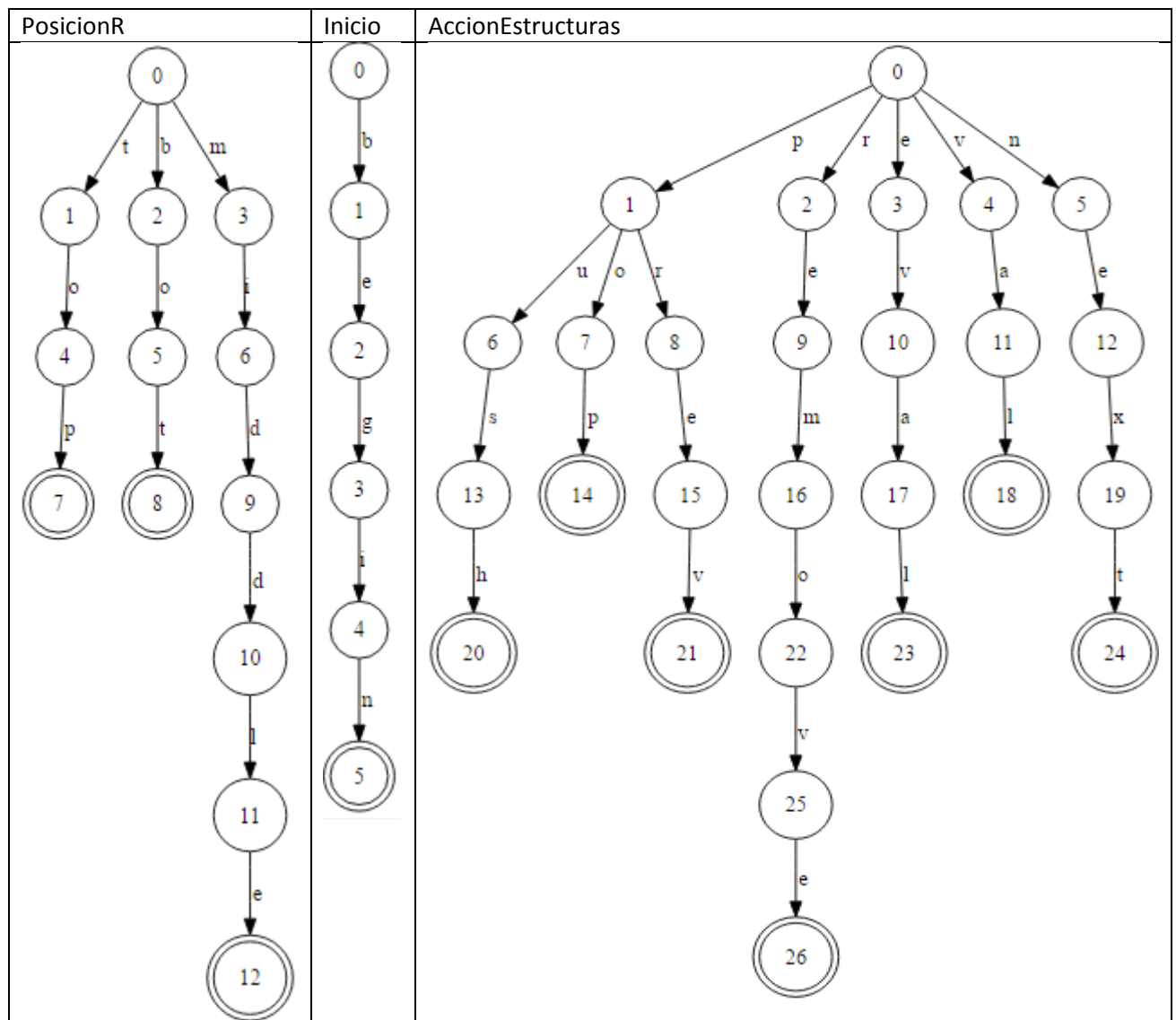


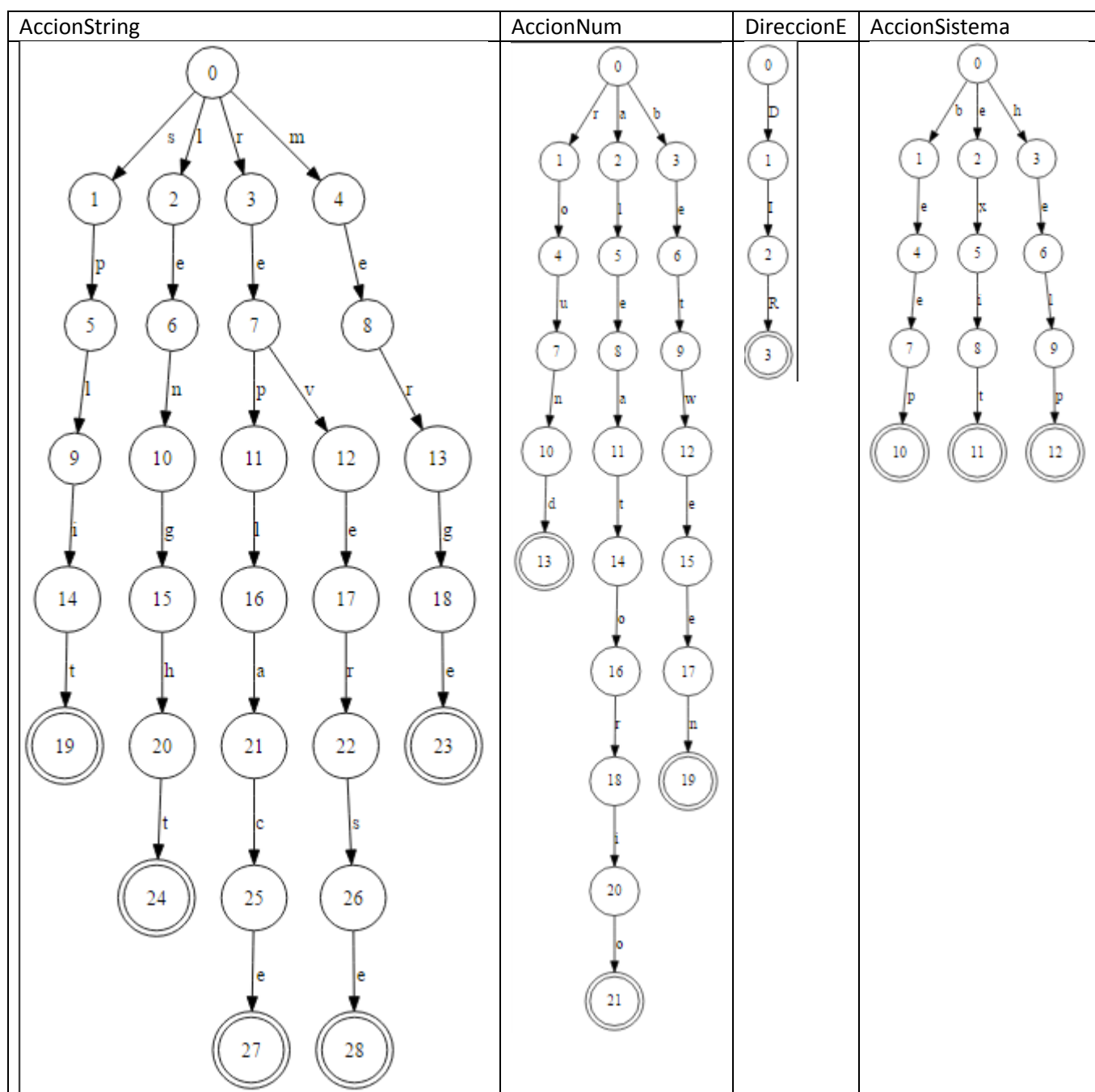


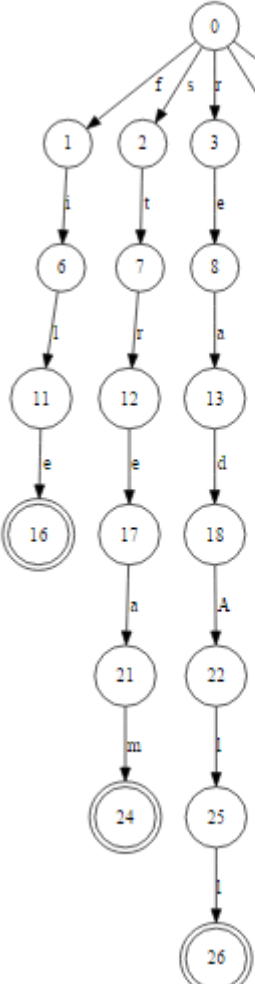
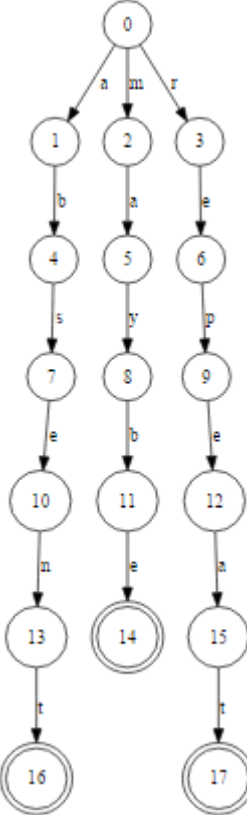
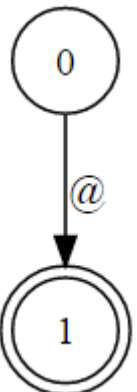
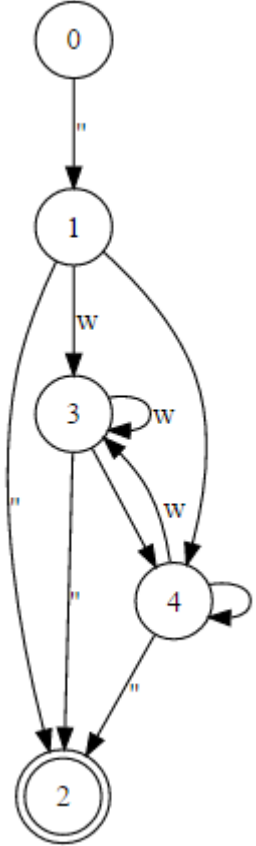




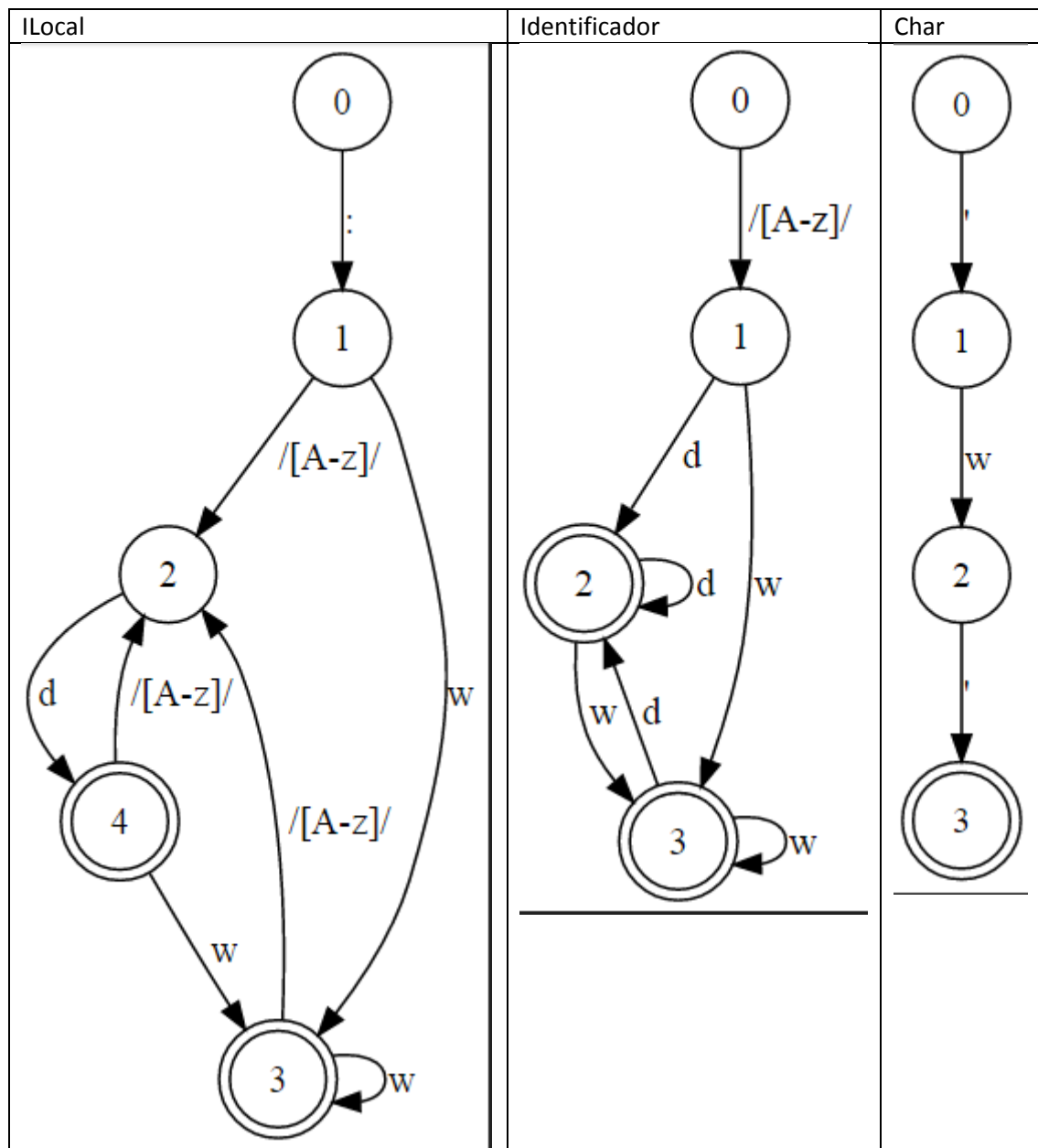
Extender	Longitud	Ajustar	Inspección	Dirección	BusquedaL	BusquedaE	Condición	Regex
<pre> graph TD     0((0)) -- e --&gt; 1((1))     1 -- x --&gt; 2((2))     2 -- t --&gt; 3((3))     3 -- e --&gt; 4((4))     4 -- n --&gt; 5((5))     5 -- a --&gt; 6((6))     6 -- s --&gt; 7(((7))) </pre>	<pre> graph TD     0((0)) -- s --&gt; 1((1))     1 -- i --&gt; 2((2))     2 -- z --&gt; 3((3))     3 -- e --&gt; 4(((4))) </pre>	<pre> graph TD     0((0)) -- r --&gt; 1((1))     1 -- e --&gt; 2((2))     2 -- s --&gt; 3((3))     3 -- i --&gt; 4((4))     4 -- z --&gt; 5((5))     5 -- e --&gt; 6(((6))) </pre>	<pre> graph TD     0((0)) -- i --&gt; 1((1))     1 -- n --&gt; 2((2))     2 -- s --&gt; 3((3))     3 -- p --&gt; 4((4))     4 -- e --&gt; 5((5))     5 -- c --&gt; 6((6))     6 -- t --&gt; 7(((7))) </pre>	<pre> graph TD     0((0)) -- d --&gt; 1((1))     1 -- i --&gt; 2((2))     2 -- r --&gt; 3(((3))) </pre>	<pre> graph TD     0((0)) -- l --&gt; 1((1))     1 -- i --&gt; 2((2))     2 -- k --&gt; 3((3))     3 -- e --&gt; 4(((4))) </pre>	<pre> graph TD     0((0)) -- e --&gt; 1((1))     1 -- q --&gt; 2((2))     2 -- u --&gt; 3((3))     3 -- a --&gt; 4((4))     4 -- l --&gt; 5((5))     5 -- s --&gt; 6(((6))) </pre>	<pre> graph TD     0((0)) -- w --&gt; 1((1))     1 -- h --&gt; 2((2))     2 -- e --&gt; 3((3))     3 -- r --&gt; 4((4))     4 -- e --&gt; 5(((5))) </pre>	<pre> graph TD     0((0)) -- r --&gt; 1((1))     1 -- e --&gt; 2((2))     2 -- g --&gt; 3((3))     3 -- e --&gt; 4((4))     4 -- x --&gt; 5(((5))) </pre>

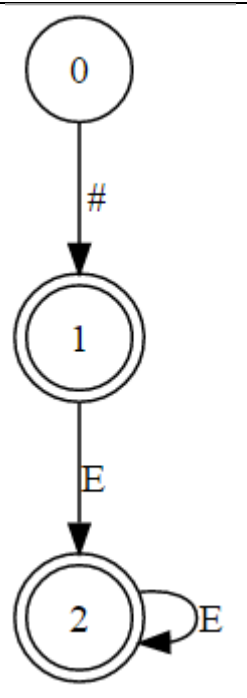
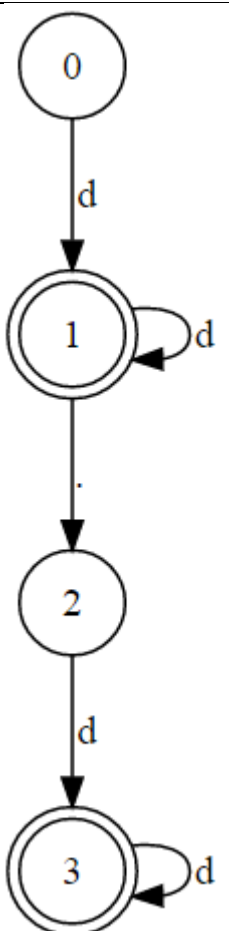
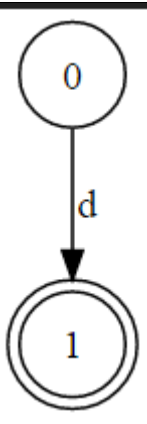
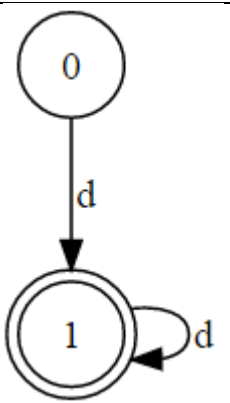




AccionArchivos	Repeticiones	Sobreescribir	String
			





Comentario	Numero	Digito	Entero
			

## Expresiones Regulares

A las expresiones regulares frecuentemente se les llaman patrones, ya que son expresiones que describen a un conjunto de cadenas. Frecuentemente son usadas para dar una descripción concisa de un conjunto, sin tener que listar todos sus elementos.

Las expresiones regulares pueden ser expresadas en términos de la teoría de lenguajes formales. Consisten de constantes y operadores que denotan el conjunto de cadenas y operaciones sobre estos conjuntos, respectivamente. Dado un alfabeto  $\Sigma$  las siguientes constantes son definidas:

- Conjunto vacío:  $L(\emptyset)$  denota el conjunto  $\{\}$
- Cadena vacía:  $L(\epsilon)$  denota el conjunto  $\{\epsilon\}$
- Carácter del alfabeto:  $L(x)$ ,  $x$  elemento de  $\Sigma$  denota el conjunto  $\{x\}$

### Operaciones básicas en expresiones regulares

#### Unión

Si  $r$  y  $s$  son expresiones regulares, entonces  $r \mid s$  es una expresión regular que define cualquier cadena que concuerda con  $r$  o con  $s$ .

- $L(a \mid b \mid c \mid d) = \{a, b, c, d\}$

#### Concatenación

La concatenación de dos expresiones regulares  $r$  y  $s$  se escribe como  $rs$  (yuxtaposición) y corresponde a cualquier cadena que sea la concatenación de dos cadenas, con la primera de ellas correspondiendo a  $r$  y la segunda a  $s$ .

- $L(a \mid b) L(c) = \{a, b\} L(c) = \{ac, bc\}$

#### Cerradura de Kleene

Se escribe  $r^*$ , donde  $r$  corresponde a la expresión regular. La expresión regular  $r^*$  corresponde a cualquier concatenación finita de cadenas, cada una de las cuales corresponde a  $r$ .

- $L(a \mid b^*) = \{a, b^*\} = \{\epsilon, a, b, bb, bbb, \dots\}$

#### Cerradura Positiva

Se escribe  $r^+$ , donde  $r$  corresponde a la expresión regular. La expresión regular  $r^+$  corresponde a cualquier concatenación finita de cadenas, donde  $r$  debe aparecer al menos una vez

- $L(ab^+) = \{ab^+\} = \{ab, abb, abb, abbb, \dots\}$

**Tabla de Expresiones Regulares**

Token	Expresión Regular	Lexema
ParIzq	((	(
ParDer	(	)
CorIzq	([	[
CorDer	(]	]
Llavelzq	({	{
LlaveDer	(}	}
Coma	(,)	,
Punto	(.)	.
EOL	(;)	;
Delim	( )	
BTrue	(?)	?
BFalse	(!)	!
Asignacion	(=)	=
GlobalSign	(\$)	\$
OAritmetico	(\* \+ \-)	+ o / o + o -
ORelacional	((\> =) (\>=) (\<=) (\!=) (\<) (\>))	>= o == o <= o != o < o >
End	^(end)\s?\$	End
Imprimir	^(print puts write)\s?\$	print o puts o write
Nuevo	^(new create)\s?\$	new o créate
Declaracion	^(def)\s?\$	Def
Var	^(var)\s?\$	Var
Invocar	^(invoke)\s?\$	Invoke
Inicializar	^(initialize)\s?\$	Initialize
Retorno	^(return break)\s?\$	return o break
TipoDato	^(bool byte bit char double int string float vect or time date point)\s?\$	bool o byte o bit o char o doublé o int o string o float o vector o time o date o point
Visibilidad	^(private protected public global)\s?\$	private o protected o public o global
Modificador	^(static abstract constant final virtual)\s?\$	static o abstract o constant o final o virtual
AsignacionEs pecial	^(add div mul sub pot sqrt copy del porcent)\s ?\$	add o div o mul o sub o pot o sqrt o copy o del o porcent
Booleano	^(true false)\s?\$	true o false
OperadorLo gico	^(and or not nand nor)\s?\$	and o or o not o nand o nor
Conversion	^(STRING INT FLOAT VAR TIME DATE)\s?\$	STRING o INT o FLOAT o VAR o TIME o DATE
Try	^(try)\s?\$	try
Librería	^(include)\s?\$	include
Union	^(union)\s?\$	union
Reference	^(sizeof typeof ref)\s?\$	sizeof o typeof o ref

Clase	^(class)\s?\$	class
Funcion	^(func)\s?\$	func
Metodo	^(void)\s?\$	void
Interface	^(interface)\s?\$	interface
EstructuraDa tos	^(struct enum list stack queue object)\s?\$	struct o enum o list o stack o queue o object
Paquete	(package)\s?\$	package
Matriz	^(matrix)\s?\$	matrix
Array	^(array)\s?\$	array
Respuesta	^(else catch finally)\s?\$	else o catch o finally
Condicional	^(if when)\s?\$	if o when
IteradorC	^(for)\s?\$	for
Do	^(do)\s?\$	do
Times	^(times)\s?\$	times
Goto	^(goto)\s?\$	goto
Get	^(get)\s?\$	get
Set	^(set)\s?\$	set
Sort	^(sort asc desc)\s?\$	sort o asc o desc
Leer	^(read)\s?\$	read
Switch	^(switch)\s?\$	switch
IteradorV	^(until while)\s?\$	until o while
Value	^(null self nil default)\s?\$	null o self o nil o default
RespuestaEs pecial	^(elsif)\s?\$	elsif
Case	^(case)\s?\$	case
Propiedad	^(property)\s?\$	property
Relacion	^(as is by)\s?\$	as o is o by
OperacionEs pecial	^(sin cos cot tan asin acos atan log loge logn f ac avg abs fraction max min count res)\s?\$	sin o cos o cot o tan o asin o acos o atan o log o loge o logn o fac o avg o abs o fraction o max o min o ocunt o res
Bsuqueda	^(search)\s?\$	search
Dibuja	^(draw graphics color font fontsize)\s?\$	draw o graphics o color o font o fontsize
EachV	^(each)\s?\$	each
Requerir	^(require)\s?\$	require
Implementar	^(implements)\s?\$	implements
Extender	^(extends)\s?\$	extends
Longitud	^(size)\s?\$	size
Ajustar	^(resize)\s?\$	resize
Inspeccion	^(inspect)\s?\$	insect
Direccion	^(dir)\s?\$	dir
BusquedaL	^(like)\s?\$	like
BusquedaE	^(equals)\s?\$	equals
Condicion	^(where)\s?\$	where

Regex	^(regex)\s?\$	regex
PosicionR	^(top bot middle)\s?\$	top o bot o middle
Inicio	^(begin)\s?\$	begin
AccionEstructuras	^(push pop remove next prev val eval)\s?\$	push o pop o remove o next o prev o val o eval
AccionString	^(split replace length merge reverse)\s?\$	split o replace o length o merge o reverse
AccionNum	^(round aleatorio between)\s?\$	round o aleatorio o between
DireccionE	^(DIR)\s?\$	DIR
AccionSistema	^(beep help exit)\s?\$	beep o help o exit
AccionArchivos	^(file stream open close readAll)\s?\$	dile o stream o open o close o readAll
Repeticiones	^(absent maybe repeat)\s?\$	absent o maybe o repeat
Sobreescribir	(@)	@
Override	^(Override)\s?\$	Override
String	(\"(\w  )*\")	“Soy un string”, “string”
ILocal	(\:)([A-z](\d \w)+?)	:Sotlocal,:varlocal
Identificador	([A-z](\d \w)+?)	Jesus,ayala,sandoval
Char	'(\w)\'	Caracter entre comilla simple 'a','s'
Comentario	(\#)(.)*	# Seguido de cualquier serie de caracteres #Hola soy un comentario
Numero	(\d+)(\.\d+)?	Cualquier numro, puede tener punto y signo 12332,12321,-123,-123.213
Entero	(\d+)	[0-9] al menos una vez 123 34
Digito	(\d)	[0-9] 3,5,2

## **Analizador Léxico**

Su principal función consiste en leer la secuencia de caracteres del programa fuente, carácter a carácter, y elaborar como salida la secuencia de componentes léxicos que utiliza el analizador sintáctico. El analizador sintáctico emite la orden al analizador léxico para que agrupe los caracteres y forme unidades con significado propio llamados componentes léxicos (tokens). Tales como:

- Palabras reservadas
- Identificadores
- Operadores
- Símbolos especiales
- Constantes numéricas
- Constantes de caracteres

El analizador léxico opera bajo petición del analizador sintáctico devolviendo un componente léxico conforme el analizador sintáctico lo va necesitando para avanzar en la gramática.

El analizador léxico es responsable de:

- Manejo de apertura y cierre de archivo, lectura de caracteres y gestión de posibles errores de apertura.
- Eliminar comentarios, espacios en blanco, tabuladores y saltos de línea.
- Inclusión de archivos y macros.
- Contabilizar número de líneas y columnas para emitir mensajes de error.

## Código Léxico

```
class Identificador
  attr_accessor :pos,:id
  def initialize(_pos,_id)
    @pos=_pos
    @id=_id
  end
end
```

```
class Token
  attr_accessor :lexema, :categoria
  def initialize(lexema,categoria)
    @lexema=lexema
    @categoria=categoria
  end
end
```

```
#Analizador Lexico
#file          =          File.read          'C:\Users\Jesus\Desktop\Nanna
Language\Compilador\program.na'
```

```
          file          =          File.read          'C:\Users\Jesus\Desktop\Nanna
Language\Compilador\Palabras.txt'
```

```
eRegular = {

  "ParIzq"=>/\(/,
  "ParDer"=>/\)/,
  "CorDer"=>/\]/,
  "CorIzq"=>/\[/,
  "LlaveDer"=>/\}/,
  "LlaveIzq"=>/\{/,
  "coma"=>/\,/ ,
  "Punto"=>/\./,
  "EOL"=>/(\;)/,

  "Delim"=>/\|/,
  "BFalse"=>/\!/,
  "Asignacion"=>/\=/,
  "BTrue"=>/\?/,
  "GlobalSign"=>/\$/,
  "OAritmetico"=>/(\*|\+|\-)/,
  "ORelacional"=>/((\>=)|(\>=)|(\<=)|(\!=)|(\<)|(\>))/,
```



```

"End"=>/^(end)\s?$/,
"Imprimir"=>/^(print|puts|write)\s?$/,
"Nuevo"=>/^(new|create)\s?$/,
"Declaracion"=>/^(def)\s?$/,
"Var"=>/^(var)\s?$/,
"Invocar"=>/^(invoke)\s?$/,
"Inicializar"=>/^(initialize)\s?$/,
"Retorno"=>/^(return|break)\s?$/,

"TipoDato"=>/^(bool|byte|bit|char|double|int|string|float|vector|
time|date|point)\s?$/,

"Visibilidad"=>/^(private|protected|public|global)\s?$/,

"Modificador"=>/^(static|abstract|constant|final|virtual)\s?$/,
"AsignacionEspecial"=>/^(add|div|mul|sub|pot|sqrt|copy|del|porcent\
s?$/,
"Booleano"=>/^(true|false)\s?$/,
"OperadorLogico"=>/^(and|or|not|nand|nor)\s?$/,

"Conversion"=>/^(STRING|INT|FLOAT|VAR|TIME|DATE)\s?$/,
"Try"=>/^(try)\s?$/,
"Libreria"=>/^(include)\s?$/,
"Union"=>/^(union)\s?$/,
"Reference"=>/^(sizeof|typeof|ref)\s?$/,
"Clase"=>/^(class)\s?$/,
"Funcion"=>/^(func)\s?$/,
"Metodo"=>/^(void)\s?$/,
"Interface"=>/^(interface)\s?$/,

"EstructuraDatos"=>/^(struct|enum|list|stack|queue|object)\s?$/,
"Paquete"=>/^(package)\s?$/,
"Matriz"=>/^(matrix)\s?$/,
"Array"=>/^(array)\s?$/,
"Respuesta"=>/^(else|catch|finally)\s?$/,
#EspaciosOpcionales abajo
"Condicional"=>/^(if|when)\s?$/,
"IteradorC"=>/^(for)\s?$/,
"Do"=>/^(do)\s?$/,
"Times"=>/^(times)\s?$/,
"goto"=>/^(goto)\s?$/,
"Get"=>/^(get)\s?$/,
"Set"=>/^(set)\s?$/,
"Sort"=>/^(sort|asc|desc)\s?$/,
"Leer"=>/^(read)\s?$/,
"Switch"=>/^(switch)\s?$/,
"IteradorV"=>/^(until|while)\s?$/,
"Value"=>/^(null|self|nil|default)\s?$/,
"RespuestaEspecial"=>/^(elsif)\s?$/,
"Case"=>/^(case)\s?$/,

```

```

"Propiedad"=>/^(property)\s?$/,
"Relacion"=>/^(as|is|by)\s?$/,

"OperacionEspecial"=>/^(sin|cos|cot|tan|asin|acos|atan|log|loge|log
n|fac|avg|abs|fraction|max|min|count|res)\s?$/,
"Busqueda"=>/^(search)\s?$/,
"Dibuja"=>/^(draw|graphics|color|font|fontsize)\s?$/,
"EachV"=>/^(each)\s?$/,
"Requerir"=>/^(require)\s?$/,
"Implementar"=>/^(implements)\s?$/,
"Extender"=>/^(extends)\s?$/,
"Longitud"=>/^(size)\s?$/,
"Ajustar"=>/^(resize)\s?$/,
"Inspeccion"=>/^(inspect)\s?$/,
"Direccion"=>/^(dir)\s?$/,
"BusquedaL"=>/^(like)\s?$/,
"BusquedaE"=>/^(equals)\s?$/,
"Condicion"=>/^(where)\s?$/,
"Regex"=>/^(regex)\s?$/,
"PosicionR"=>/^(top|bot|middle)\s?$/,
"Inicio"=>/^(begin)\s?$/,

"AccionEstructuras"=>/^(push|pop|remove|next|prev|val|eval)\s?$/,
"AccionString"=>/^(split|replace|lenght|merge|reverse)\s?$/,
"AccionNum"=>/^(round|aleatorio|between)\s?$/,
"DireccionE"=>/^(DIR)\s?$/,
"AccionSistema"=>/^(beep|help|exit)\s?$/,
"AccionArchivos"=>/^(file|stream|open|close|readAll)\s?$/,
"Repeticiones"=>/^(absent|maybe|repeat)\s?$/,
"Override"=>/^(Override)\s?$/,

#Especiales

"Sobreescribir"=>/(\@)/,
"String"=>/(\\"(\w| )*\\")/,
"Char"=>/\'(\w)\'/,
"ILocal"=>/(\:)([A-z]((\d|\w)+)?)/,
"Comentario"=>/(\#)(\.)* ?/,
"Identificador"=>/([A-z]((\d|\w)+)?)/,

"Numero"=>/(\d+)(\.\d+)?/,
#"Entero"=>/(\d)(\d)?/,
#"Digito"=>/(\d)/,

}

```

```

todas=Regexp.new /\n/
  $pila = []
  $tabla_simbolos=[]
  $tokens=[]

  eRegular.each_value do |valor|
    expresion=Regexp.new valor
    todas=Regexp.union(expresion,todas)
  end
  def CheckForId(id)
    pos=0
    $tabla_simbolos.each do |found|
      if found.id==id
        $pila.push(id+"(id)->" + pos.to_s)
        return true
      end
      pos+=1
    end
    return false
  end
  file.gsub(todas).map{Regexp.last_match}.each do |_match|
    eRegular.each do |categoria,expresion|
      case _match.to_s
      when Regexp.new(expresion);
        a=_match.to_s
        x= a.strip
        $tokens.push(Token.new(x,categoria))
        If categoria=="Identificador" or categoria=="ILocal"
          a=x
          if($tabla_simbolos.size==0)
            $pila.push(x+"(id)"+"se guardo en "+$tabla_simbolos.size.to_s)

            $tabla_simbolos.push(Identificador.new($tabla_simbolos.size,x))
          else
            if CheckForId(a)
            else
              $pila.push(x+"(id)"+"se guardo en "+$tabla_simbolos.size.to_s)

              $tabla_simbolos.push(Identificador.new($tabla_simbolos.size,x))
            end
          end
        else
          $pila.push(x + "|->("+categoria+")")
        end
      end
    end
  end
end
end
end

```

```

#Imprime Lexico
    pos=0
    $pila.each do |match|
        p (pos+=1).to_s+" "+match
    end

#Imprime Tabla de Simbolos
    p "_____"
    p "Tabla de Simbolos"
    p "pos|Identificador"
    $tabla_simbolos.each do |identificador|
        p " "+identificador.pos.to_s + " |" + " "+identificador.id
    end

```

## TXT Prueba

'! - " ( ) \$ , . / :local ; ? @ [] | < + <= == => >= \* != add and array bool break byte case catch char class  
copy def del div do double else elsif end false finally float for func if include initialize int invoke mul  
nand nor new not null or package pot print private  
sin cos tan asin acos atan log loge logn fac avg abs fraction read enum list stack queue property as  
is get set sort by asc desc switch object virtual abstract constant final interface STRING INT FLOAT  
VAR goto { } struct union sizeof typeof replace max min count each matrix require implements  
extends package property res draw vector ref search size resize inspect dir like equals where regex  
top bot middle cot times begin percent write push pop remove split lenght merge reverse round  
aleatorio between next prev bit DIR beep graphics color font fontsize nil default close time date  
TIME DATE help create global val eval file stream open readAll exit absent maybe repeat point  
protected public puts return self sqrt static string sub true try until var void when while @ Override  
"Soy un string"  
#Soy un comentario  
'C'  
abc var1 var2 var2 var5  
  
var3 var1 var1 var5var2 x y z x y z a a a

## Corrida de Escritorio

```
C:\WINDOWS\system32\cmd.exe
C:\Users\Jesus\Desktop\Nanna Language\Compilador>ruby compilador.rb
"1" -> <BFalse>
"2" 1 -> <Ohrifuncion>
"3" 1 -> <ParIzq>
"4" 1 -> <ParDer>
"5" 1 -> <GlobalSign>
"6" 1 -> <coma>
"7" 1 -> <Punto>
"8" 1 -> <Ohritmetico>
"9" local(id)se guardo en 0"
"10" 1 -> <EOL>
"11" 1 -> <BTrue>
"12" 0 -> <Sobreescrihir>
"13" 1 -> <CorIzq>
"14" 1 -> <CorDer>
"15" 1 -> <DefLin>
"16" 1 -> <Operacion>
"17" 1 -> <Ohritmetico>
"18" 1 -> <Asignacion>
"19" 1 -> <Asignacion>
"20" 1 -> <Asignacion>
"21" 1 -> <Operacion>
"22" 1 -> <Asignacion>
"23" 1 -> <Ohritmetico>
"24" 1 -> <BFalse>
"25" add1 -> <AsignacionEspecial>
"26" and1 -> <OperadorLogico>
"27" array1 -> <Array>
"28" bool1 -> <TipoDato>
"29" break1 -> <Retorno>
"30" byte1 -> <TipoDato>
"31" case1 -> <Case>
"32" catch1 -> <Respuesta>
"33" char1 -> <TipoDato>
"34" class1 -> <Clase>
"35" copy1 -> <AsignacionEspecial>
"36" def1 -> <Definicion>
"37" del1 -> <AsignacionEspecial>
"38" div1 -> <AsignacionEspecial>
"39" do1 -> <Do>
"40" double1 -> <TipoDato>
"41" else1 -> <Respuesta>
"42" elsif1 -> <RespuestaEspecial>
"43" end1 -> <End>
"44" false1 -> <BFalse>
"45" finally1 -> <Respuesta>
"46" float1 -> <TipoDato>
"47" for1 -> <Iterador>
"48" func1 -> <Funcion>
"49" if1 -> <Condicional>
"50" include1 -> <Libreria>
"51" initialize1 -> <Inicializar>
"52" int1 -> <TipoDato>
"53" invoke1 -> <Invoke>
"54" null1 -> <AsignacionEspecial>
"55" nand1 -> <OperadorLogico>
"56" nor1 -> <OperadorLogico>
"57" new1 -> <Nuevo>
"58" not1 -> <OperadorLogico>
"59" null1 -> <Value>
"60" or1 -> <OperadorLogico>
"61" package1 -> <Paquete>
"62" pot1 -> <AsignacionEspecial>
"63" print1 -> <Imprimir>
"64" puts1 -> <Oisibilidad>
"65" sin1 -> <OperacionEspecial>
"66" cos1 -> <OperacionEspecial>
"67" tan1 -> <OperacionEspecial>
"68" asin1 -> <OperacionEspecial>
"69" acos1 -> <OperacionEspecial>
"70" atan1 -> <OperacionEspecial>
"71" log1 -> <OperacionEspecial>
"72" log1 -> <OperacionEspecial>
"73" logn1 -> <OperacionEspecial>
"74" fac1 -> <OperacionEspecial>
"75" avg1 -> <OperacionEspecial>
"76" abs1 -> <OperacionEspecial>
"77" fraction1 -> <OperacionEspecial>
"78" read1 -> <Leer>
"79" enum1 -> <EstructuraDato>
"80" list1 -> <EstructuraDato>
"81" stack1 -> <EstructuraDato>
"82" queue1 -> <EstructuraDato>
"83" property1 -> <Propiedad>
"84" as1 -> <Relacion>
"85" is1 -> <Relacion>
"86" get1 -> <Get>
"87" set1 -> <Set>
"88" sort1 -> <Sort>
"89" by1 -> <Definicion>
"90" asc1 -> <Sort>
"91" desc1 -> <Sort>
"92" switch1 -> <Switch>
"93" object1 -> <EstructuraDato>
"94" virtual1 -> <Modificador>
"95" abstract1 -> <Modificador>
"96" constant1 -> <Modificador>
"97" final1 -> <Modificador>
"98" interface1 -> <Interface>
"99" SIRMGI1 -> <Conversion>
"100" INT1 -> <Conversion>
"101" FLOAT1 -> <Conversion>
"102" URI1 -> <Conversion>
"103" goto1 -> <Goto>
"104" 1 -> <ClaveDer>
"105" 1 -> <ClaveDer>
"106" struct1 -> <EstructuraDato>
"107" union1 -> <Union>
"108" sizeof1 -> <Reference>
"109" sizeof1 -> <Reference>
"110" replace1 -> <AccionString>
"111" max1 -> <OperacionEspecial>
"112" min1 -> <OperacionEspecial>
"113" count1 -> <OperacionEspecial>
"114" each1 -> <EachU>
"115" nand1 -> <ParIzq>
"116" require1 -> <Requerir>
"117" implements1 -> <Implementar>
"118" extends1 -> <Extender>
"119" package1 -> <Paquete>
"120" property1 -> <Propiedad>
"121" res1 -> <OperacionEspecial>
"122" draw1 -> <Dibujar>
"123" vector1 -> <TipoDato>
"124" ref1 -> <Reference>
"125" search1 -> <Busqueda>
"126" size1 -> <Longitud>
"127" west1 -> <Gustar>
"128" inspect1 -> <Inspeccion>
"129" dir1 -> <Direccion>
"130" like1 -> <Busqueda>
"131" equals1 -> <Busqueda>
"132" where1 -> <Condicional>
"133" request1 -> <Requerir>
"134" top1 -> <PosicionR>
"135" bot1 -> <PosicionR>
"136" middle1 -> <PosicionR>
"137" cot1 -> <OperacionEspecial>
"138" time1 -> <Tiempo>
"139" begin1 -> <Inicio>
"140" percent1 -> <AsignacionEspecial>
"141" write1 -> <Imprimir>
"142" push1 -> <AccionEstructuras>
"143" pop1 -> <AccionEstructuras>
"144" remove1 -> <AccionEstructuras>
"145" split1 -> <AccionString>
"146" lenght1 -> <AccionString>
"147" merge1 -> <AccionString>
"148" reverse1 -> <AccionString>
"149" round1 -> <AccionNum>
"150" aleatorio1 -> <AccionNum>
"151" between1 -> <AccionNum>
"152" next1 -> <AccionEstructuras>
"153" prev1 -> <AccionEstructuras>
"154" hit1 -> <TipoDato>
"155" DIR1 -> <Direccion>
"156" heap1 -> <AccionSistema>
"157" graphics1 -> <Dibujar>
"158" color1 -> <Dibujar>
"159" font1 -> <Dibujar>
"160" fontsize1 -> <Dibujar>
"161" nil1 -> <Value>
"162" default1 -> <Value>
"163" close1 -> <AccionArchivos>
"164" time1 -> <TipoDato>
"165" date1 -> <TipoDato>
"166" TIME1 -> <Conversion>
"167" DATE1 -> <Conversion>
"168" help1 -> <AccionSistema>
"169" create1 -> <Nuevo>
"170" TIME1 -> <Conversion>
"171" DATE1 -> <Conversion>
"172" help1 -> <AccionSistema>
"173" create1 -> <Nuevo>
"174" global1 -> <AccionEstructuras>
"175" wall1 -> <AccionEstructuras>
"176" file1 -> <AccionArchivos>
"177" stream1 -> <AccionArchivos>
"178" open1 -> <AccionArchivos>
"179" readall1 -> <AccionArchivos>
"180" exit1 -> <AccionSistema>
"181" absent1 -> <Repeticiones>
"182" maybe1 -> <Repeticiones>
"183" repeat1 -> <Repeticiones>
"184" point1 -> <TipoDato>
"185" protected1 -> <Oisibilidad>
"186" public1 -> <Oisibilidad>
"187" puts1 -> <Imprimir>
"188" return1 -> <Retorno>
"189" self1 -> <Value>
"190" sqrt1 -> <AsignacionEspecial>
"191" static1 -> <Modificador>
"192" string1 -> <TipoDato>
"193" sub1 -> <AsignacionEspecial>
"194" true1 -> <BTrue>
"195" try1 -> <Try>
"196" until1 -> <IteradorU>
"197" var1 -> <Var>
"198" void1 -> <Metodo>
"199" when1 -> <Condicional>
"200" while1 -> <IteradorU>
"201" 0 -> <Sobreescribir>
"202" Override1 -> <Override>
"203" " Soy un string" -> <String>
"204" $oy un comentario -> <Comentario>
"205" 0 -> <Char>
"206" abc(id)se guardo en 1"
"207" var1(id)se guardo en 2"
"208" var2(id)se guardo en 3"
"209" var3(id) -> 2"
"210" var4(id)se guardo en 4"
"211" var5(id)se guardo en 5"
"212" var1(id) -> 2"
"213" var2(id) -> 3"
"214" x(id)se guardo en 6"
"215" y(id)se guardo en 7"
"216" z(id)se guardo en 8"
"217" x(id) -> 6"
"218" y(id) -> 7"
"219" a(id)se guardo en 9"
"220" a(id) -> 9"
"221" a(id) -> 9"
"
"Tabla de Simbolos"
"posIdentificador"
"0" : local
"1" : abc
"2" : var1
"3" : var2
"4" : var5
"5" : var3
"6" : x
"7" : y
"8" : z
"9" : a
C:\Users\Jesus\Desktop\Nanna Language\Compilador>
```