

Tecnológico Nacional de México

Instituto Tecnológico de Tijuana

Subdirección Académica

Departamento de Sistemas y Computación

Ingeniería en Sistemas Computacionales

AGOSTO - DICIEMBRE 2016

Lenguajes y Autómatas 1

6SC6A

4:00 p.m. a 6:00 p.m.

Documentación Léxico

Erasmó Estrada Peña

Pasillas Luis Miguel Angel - #14210423

Tijuana, B.C. del 22 de noviembre del 2016

Índice

II – Expresiones Regulares.....	1
III – Autómatas	3
IV – Analizador Léxico.....	16

II – Expresiones Regulares

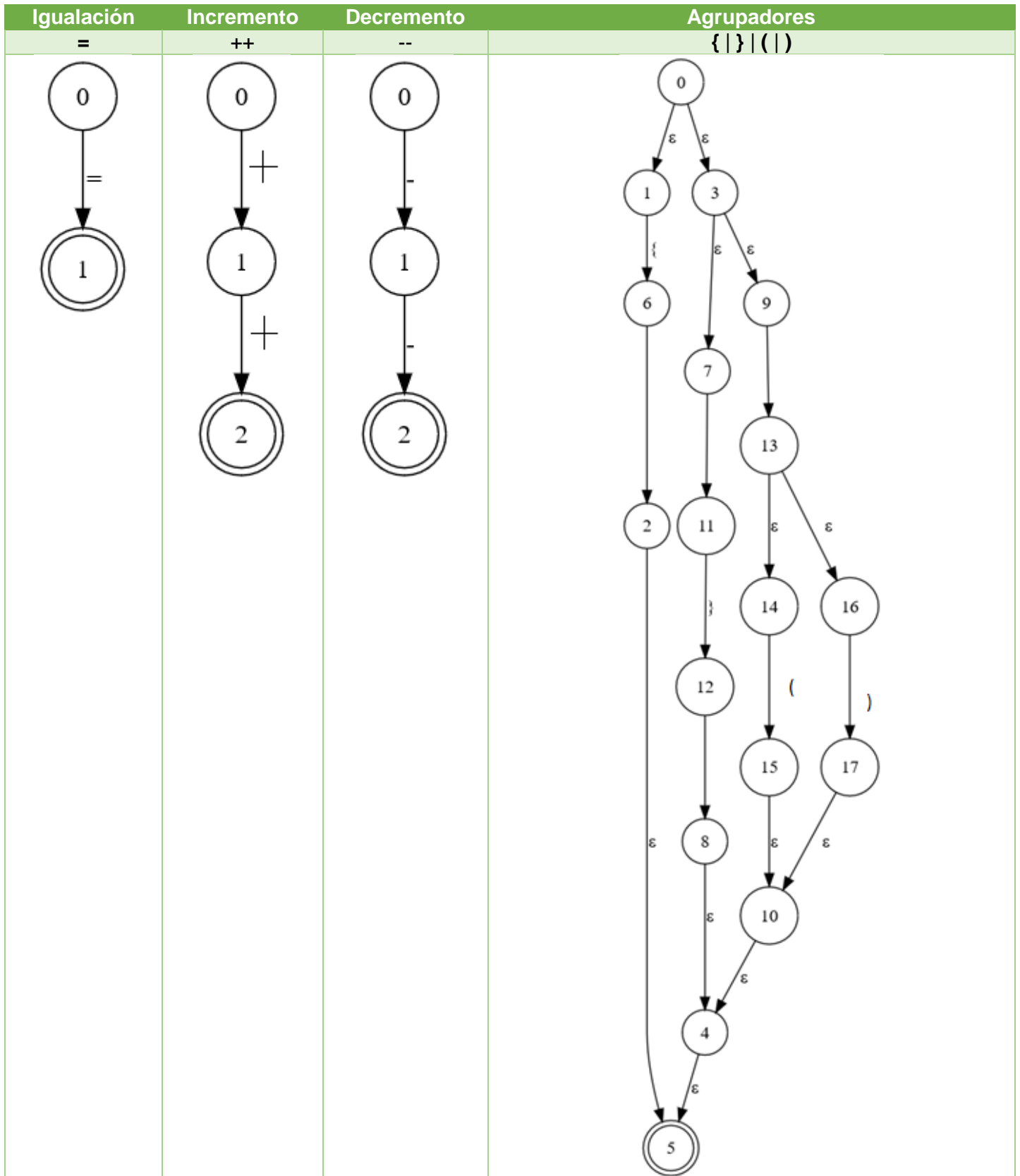
El análisis léxico de un compilador tiene el deber de leer un programa fuente como un archivo de caracteres y dividirlo en *Tokens*. Cada *Token* es una secuencia de caracteres que representan unidad de información en el programa fuente. Como la tarea que realiza el analizador léxico es un caso especial de coincidencias de patrones (*expresiones regulares*), es una regla que genera una secuencia de caracteres que puede representar a un determinado componente léxico. Lexema es una cadena de caracteres que concuerda con un patrón que describe un componen léxico.

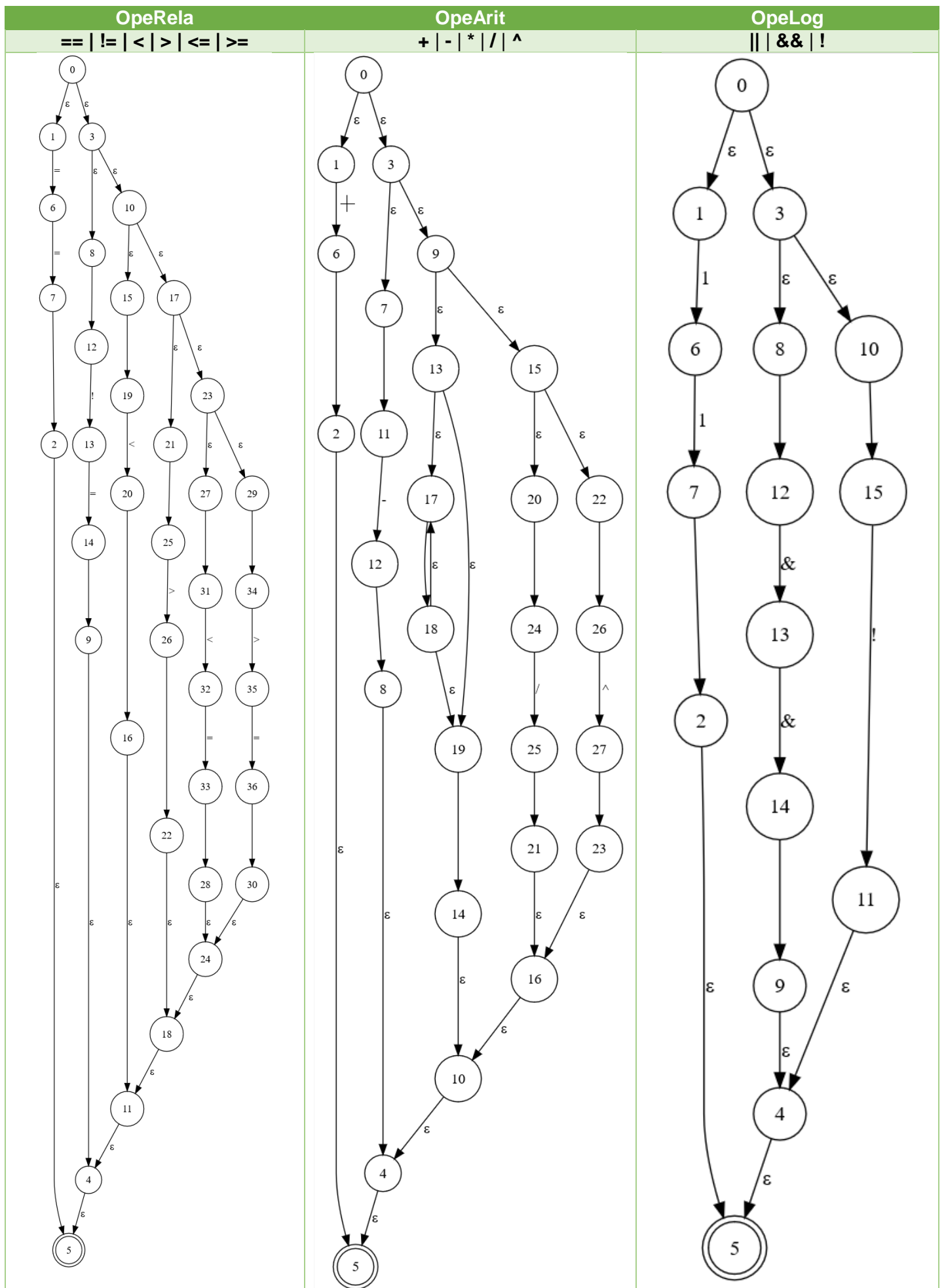
No.	Token	Expresión Regular	Lexema
1	Igualación	<code>^(=)\$</code>	<code>=</code>
2	Incremento	<code>^(++)\$</code>	<code>++</code>
3	Decremento	<code>^(--)\$</code>	<code>--</code>
4	Agrupadores	<code>^(\{ \} ())\$</code>	<code>{ } ()</code>
5	OpeRela	<code>^(== != < > <= >=)\$</code>	<code>== != < > <= >=</code>
6	OpeArit	<code>^(+ - * / ^)\$</code>	<code>+ - * / ^</code>
7	OpeLog	<code>^(&& !)\$</code>	<code> && !</code>
8	TipoBool	<code>^(verdadero falso)\$</code>	<code>verdadero falso</code>
9	TipoDato	<code>^(caracxp cadenaxp entxp bytexp cortoxp boolxp flotanxp flotan64xp uentxp ubytexp ucortoxp ulargoxp decimalxp varxp enumxp structxp)\$</code>	<code>caracxp cadenaxp entxp bytexp cortoxp boolxp flotanxp flotan64xp uentxp ubytexp ucortoxp ulargoxp decimalxp varxp enumxp structxp</code>
10	Instrucción Iteración	<code>^(paraxv paracadaxv mientrasxv hacervx enxv)\$</code>	<code>paraxv paracadaxv mientrasxv hacervx enxv</code>
11	Instrucción Selección	<code>^(sixv sinoxv noxv cambiarxv casoxv)\$</code>	<code>sixv sinoxv noxv cambiarxv casoxv</code>
12	Instrucción Salto	<code>^(descansarxv continuarxv retornarxv defectoxv rendimiendoxv)\$</code>	<code>descansarxv continuarxv retornarxv defectoxv rendimiendoxv</code>
13	Atrapa Errores	<code>^(intentarxv atraparxv finalxv lanzarxv)\$</code>	<code>intentarxv atraparxv finalxv lanzarxv</code>
14	Control Acceso	<code>^(publico privado interno protegido)\$</code>	<code>publico privado interno protegido</code>
15	Referencia	<code>^(clase interface delegado dinamico)\$</code>	<code>clase interface delegado dinamico</code>
16	Modificadores	<code>^(abstractom asincronom constantem eventom externom nuevom anularm parcialm lecturasolom selladom estaticom noguardarm virtualm volatilm)\$</code>	<code>abstractom asincronom constantem eventom externom nuevom anularm parcialm lecturasolom selladom estaticom noguardarm virtualm volatilm</code>
17	Checardor	<code>^(chechar nochechar)\$</code>	<code>chechar nochechar</code>
18	Parámetros	<code>^(paramet referen salida)\$</code>	<code>paramet referen salida</code>
19	Revisión	<code>^(areglado bloquear)\$</code>	<code>areglado bloquear</code>
20	Espacio Nombre	<code>^(espacionombre usando externo)\$</code>	<code>espacionombre usando externo</code>
21	Clave Operadores	<code>^(como esperar es tamañode tipode stackloco explicitoc implicitoc operadorc)\$</code>	<code>como esperar es tamañode tipode stackloco explicitoc implicitoc operadorc</code>
22	Clave Acceso	<code>^(base esta nulo)\$</code>	<code>base esta nulo</code>

23	Clave Contextual	^(obtener global parcial remover poner evaluar donde anadir)\$	obtener global parcial remover poner evaluar donde anadir
24	Clave Consultas	^(desde donde seleccionar grupo dentro ordenarpor unirse dejar en sobre igual por acendente decendente)\$	desde donde seleccionar grupo dentro ordenarpor unirse dejar en sobre igual por acendente decendente
25	Indicadores de Formato	^(omitir izquierda derecha interno deca octa hexa mostrarbase mostrarpunto letramayus mostrarposi cientifico arreglado enterobuf justarcampo basecampo flotantecampo)\$	omitir izquierda derecha interno deca octa hexa mostrarbase mostrarpunto letramayus mostrarposi cientifico arreglado enterobuf justarcampo basecampo flotantecampo
26	Impresión	^(Efsanf Efsanln Impri Imprif Impriln Escan Escanf Escanln ESprint ESprintf ESprintln Eescan EEscanf)\$	Efsanf Efsanln Impri Imprif Impriln Escan Escanf Escanln ESprint ESprintf ESprintln Eescan EEscanf
27	Funciones Cadenas	^(Contar Igualdoble Campos Funcampo tenerprefijo Tenersufijo Repetir Reemplazar Cortar)\$	Contar Igualdoble Campos Funcampo tenerprefijo Tenersufijo Repetir Reemplazar Cortar
28	Función Principal	^(vacio inicio funcion)\$	vacio inicio funcion
29	Función Matemáticas	^(abstr facos fasin fatan fcos fcosh fexp flog fmax fmin fpow fsign ftan ftanh ftruncate)\$	abstr facos fasin fatan fcos fcosh fexp flog fmax fmin fpow fsign ftan ftanh ftruncate
30	Palabras Reservadas	^(factivate fdouble floctors fadd fdrop flock faster fdssize flockmax falias fdynamic fall feach flong)\$	factivate fdouble floctors fadd fdrop flock faster fdssize flockmax falias fdynamic fall feach flong

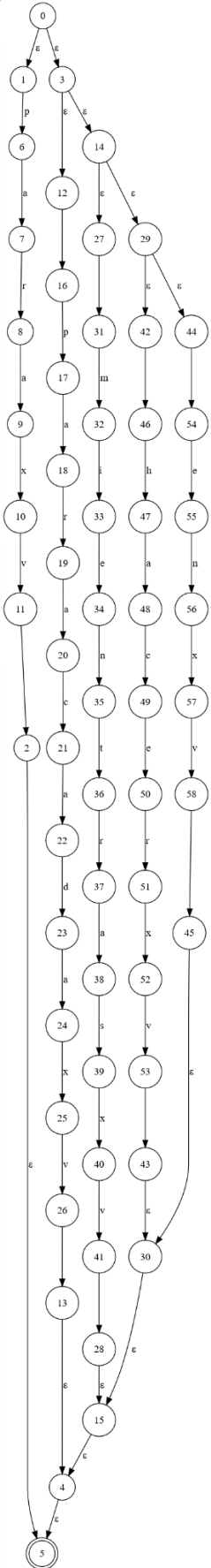
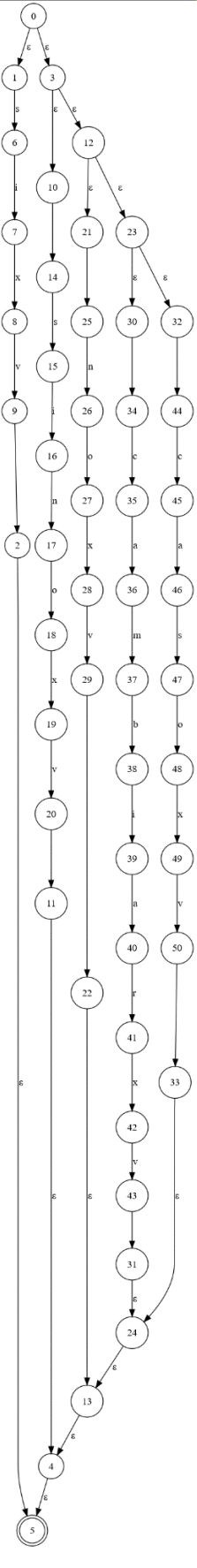
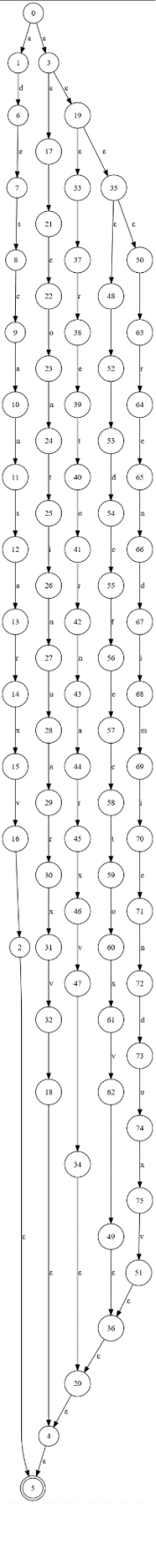
III – Autómatas

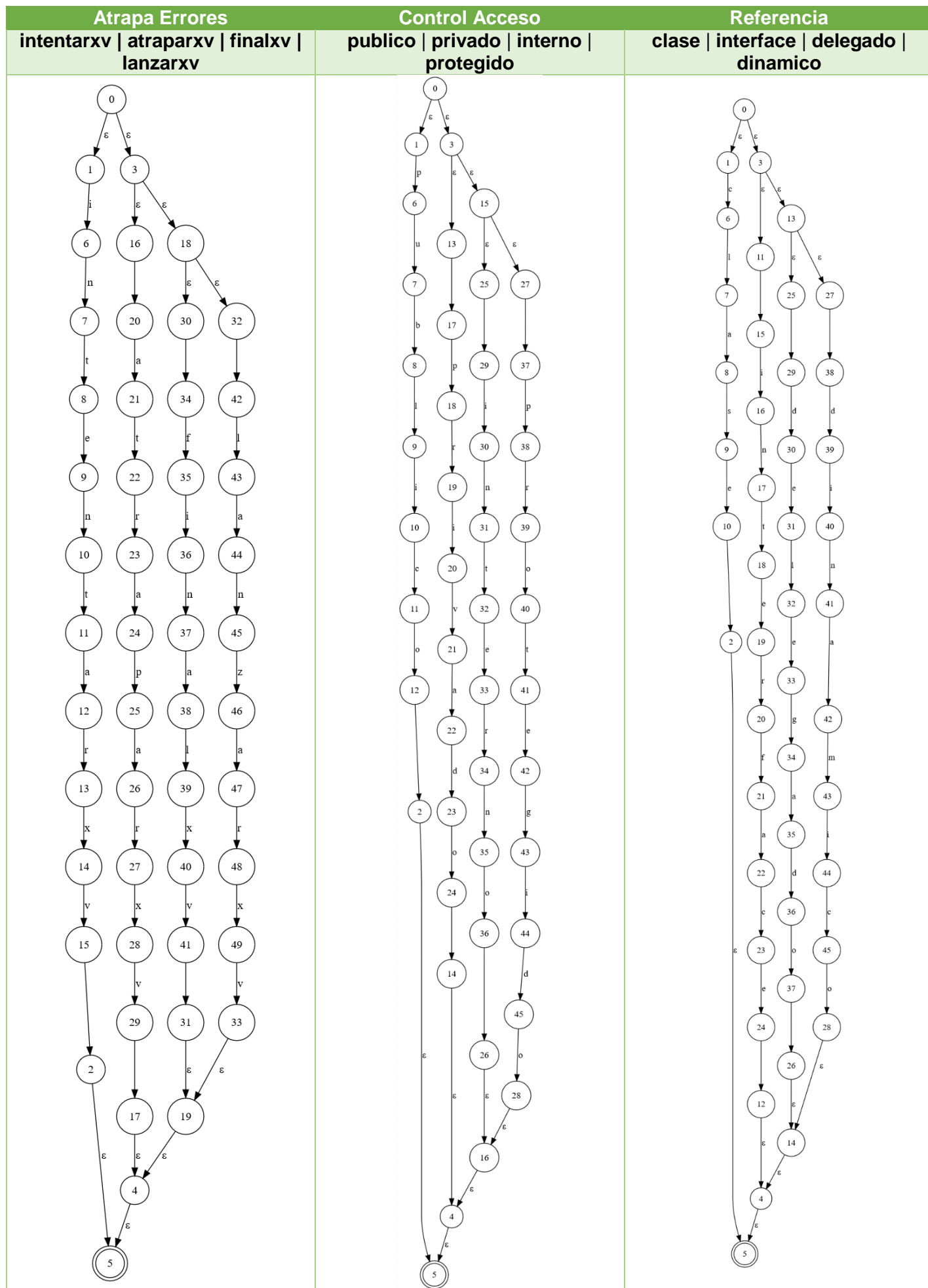
Un Autómata es una construcción lógica que recibe como entrada una cadena de símbolos y produce una salida indicando si la salida es una cadena que pertenece a un determinado lenguaje. Autómata Finito No Determinista: Si se permite que desde un estado se realicen cero, una o más transiciones mediante el mismo símbolo de entrada, se dice que el autómata finito es no determinista.

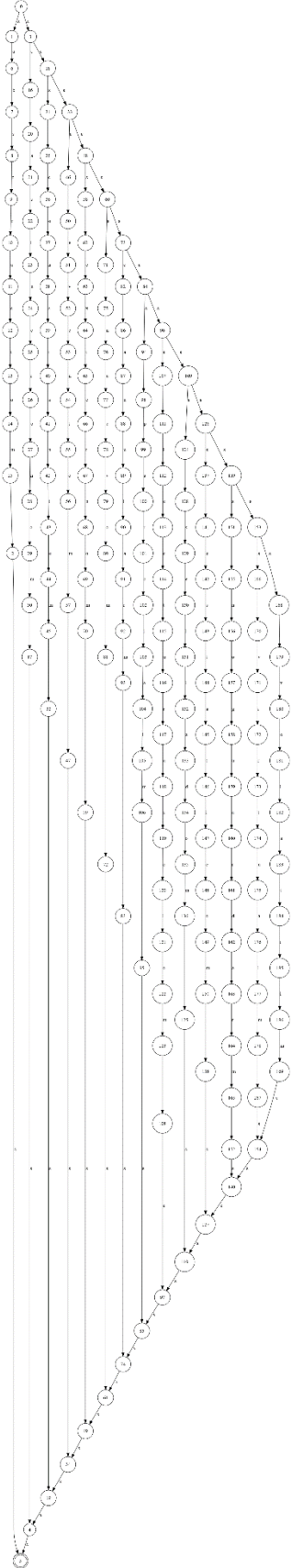
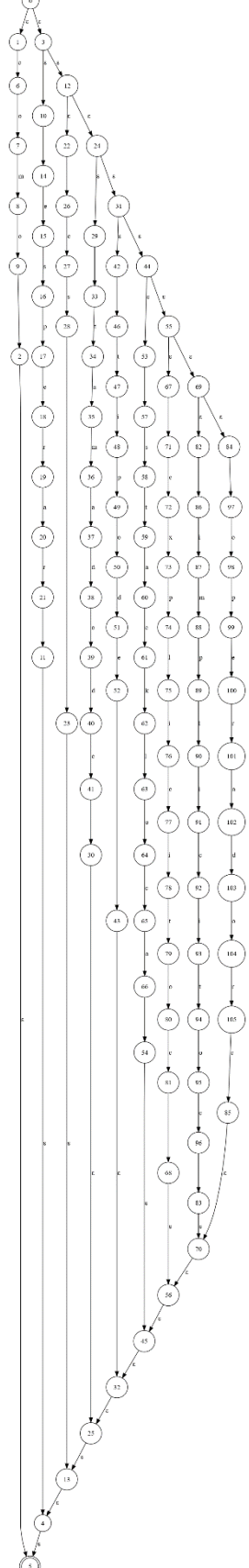
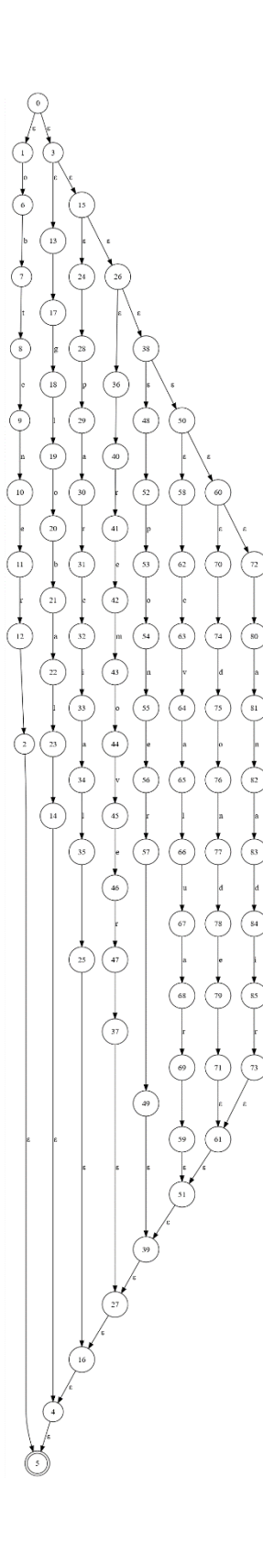




TipoBool	Tipo Dato
verdadero falso	caracxp cadenaxp entxp bytexp cortoxp boolxp flotanxp flotan64xp uentxp ubytexp ucortoxp ulargoxp decimalxp varxp enumxp structxp

<div>Instrucción Iteración</div> <div>paraxv paracadaxv mientrasxv hacerxv enxv</div>	<div>Instrucción Selección</div> <div>sixv sinoxv noxv cambiarxv casoxv</div>	<div>Instrucción Salto</div> <div>descansarxv continuarxv retornarxv defectoxv rendimiendoxv</div>
		



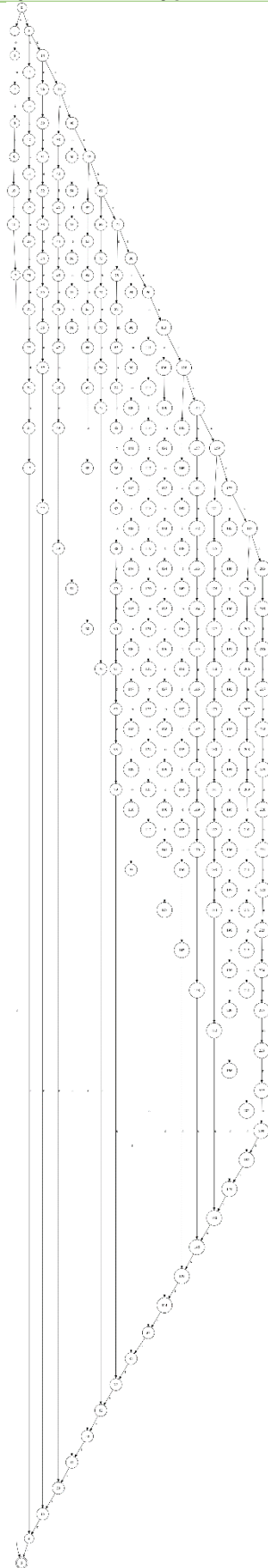
Modificadores abstractom asincronom constantem eventom externom nuevom anularm parcialm lecturasolom selladom estaticom noguardarm virtualm volatilm	Clave Operadores como esperar es tamañode tipode stackloco explicitoc implicitoc operadorc	Clave Contextual obtener global parcial remove poner evaluar donde anadir
		

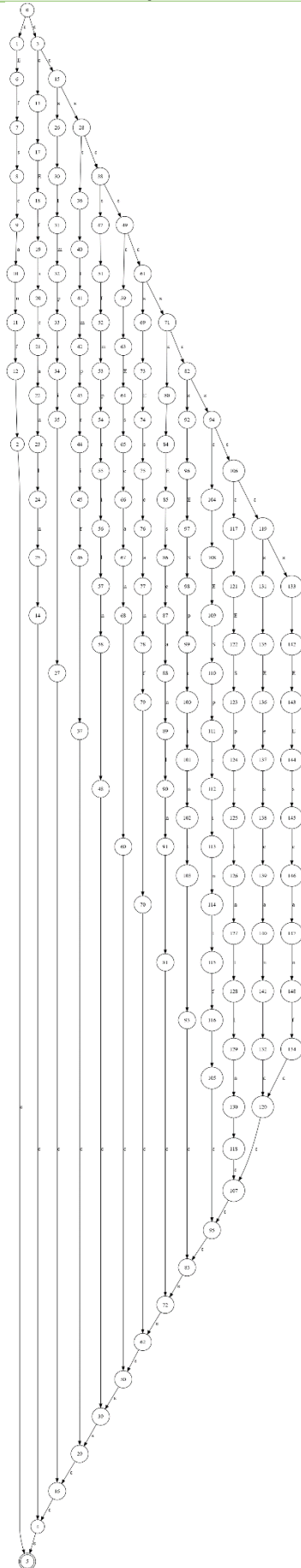
Checadores		Parametros			Revisión	
chechar nochechar		paramet	referen	salida	areglado bloquear	

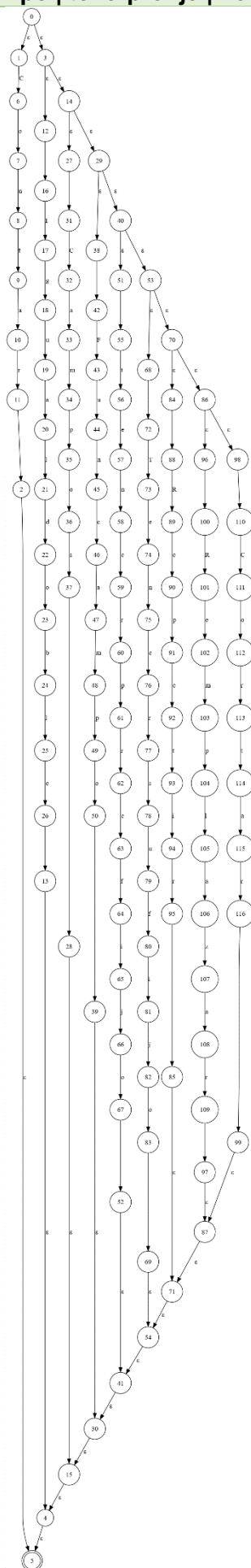
<div>Espacio Nombre</div> <div>espacionombre usando externo</div>	<div>Clave Acceso</div> <div>base esta nulo</div>	<div>Funcion Principal</div> <div>vacio inicio funcion</div>

Indicadores de Formato

omitir | izquierda | derecha | interno | deca | octa | hexa | mostrarbase | mostrarpunto | letramayus |
mostrarposi | científico | arreglado | enterobuf | justarcampo | basecampo | flotantecampo

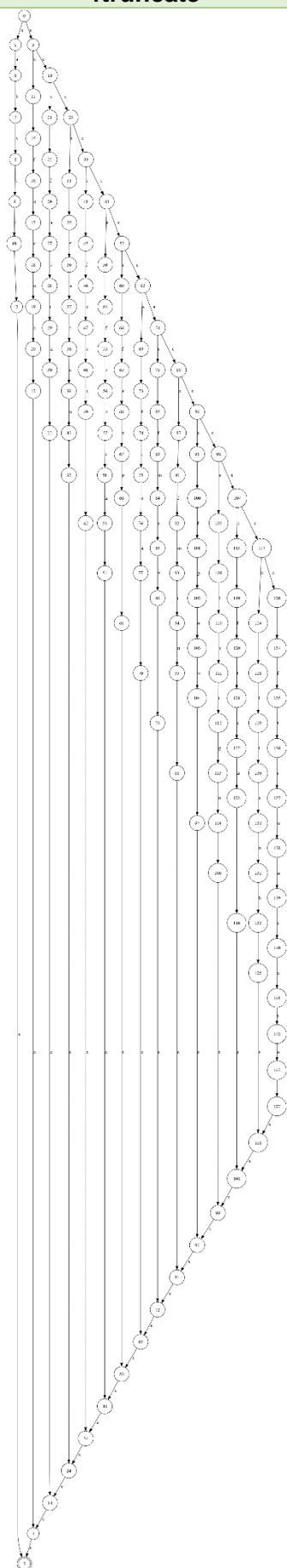






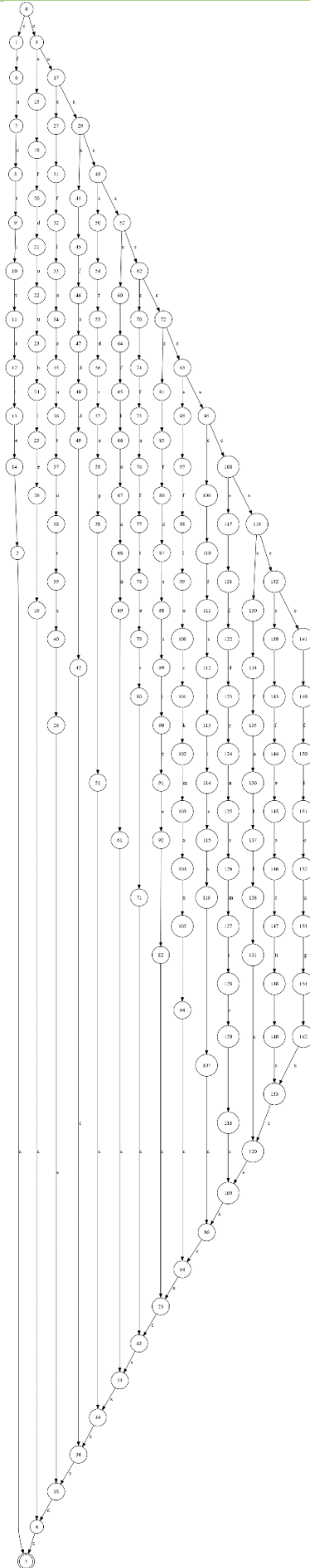
Funcion Matematicas

abstr | facos | fasin | fatan | fcos | fcosh | fexp | flog | fmax | fmin | fpow | fsign | ftan | ftanh | ftruncate



Palabras Reservadas

**factivate | fdouble | floctors | fadd | fdrop | flock | faster | fdssize | flockmax | falias | fdynamic | fall
| feach | flong**



IV – Analizador Léxico

El análisis léxico es la primera fase de un compilador. Toma el código fuente del lenguaje que se escriben en forma de frases, El analizador léxico rompe esta sintaxis en una serie de “tokens”, eliminando cualquier espacio en blanco o comentarios del código fuente. Si el analizador léxico encuentra un token inválido, se genera un error. El analizador léxico trabaja en estrecha colaboración con el analizador de sintaxis. Lee flujos de caracteres del código fuente, identifica tokens válidos y pasa los datos al analizador de sintaxis.

Código del Programa

Muestra el código que se usó para la generar el Compilador Léxico.

```
package main

import (
    "fmt"           //Instrucciones basicas
    "io/ioutil"     //Permite abrir archivos de texto
    "regexp"        //Permite
    "strings"       //Permite la conversion de strings (byte a string)
)

func check(e error) {
    if e != nil {
        panic(e)
    }
}

func main() {

    var (
        //contador = 0
        total = 0
    )
    //var id [160]string

    fmt.Println("\t\nEl archivo contiene: \n")

    //Abriendo el archivo

    //"/home/pasillas/work/usr/haa.txt"
    //"/home/pasillas/Escritorio/ha.txt"
    info, errr := ioutil.ReadFile("C:\\Users\\Angel Pasillas\\Desktop\\Lenguajes y
Automatas 1\\archivoPalabras.txt")
    check(errr)

    fmt.Println("-----")
    //Imprimiendo cadena en consola desde el archivo original
    fmt.Print(string(info))
    fmt.Println("-----")

    s := string(info[:]) //Transformar de Byte a String

    fmt.Println() //Salto de linea
    //=<> Separador de simbolos
    r := strings.NewReplacer("{", " { ", "}", " } ", ";", " ; ", "++", " ++ ", "(", " ( ",
    ")", " ) ", "--", " -- ", "+", " + ", "-", " - ", "*", " * ", "/", " / ", "%", " % ",
    "[", " [ ", "]", " ] ", "==" , " == ", "<=", " <= ", ">=", " >= ", "!=" , " != ", "&&", "
&& ", "||", " || ", "!", " ! ")
    z := r.Replace(s)

    valores := strings.Fields(z) //Separa solo las palabras de los espacios

    fmt.Println(`-----`)
    fmt.Println(`/      N      |      TOKEN      |      LEXEMA      \`)
```

```

fmt.Println(`          | ----- |`)

for i := range valores {

//Tokens comunes
asignacion, _ := regexp.MatchString("^=$", valores[i])
//Asignacion
operadoresArit, _ := regexp.MatchString("^[\\+|\\-|\\*|\\/|\\%|\\^]$", valores[i])
//Operadores aritmeticos
operadoresRela, _ :=
regexp.MatchString("((^\\<$)|(^\\>$)|(^\\=\\=$)|(^\\!\\=\\=$)|(^\\>\\=$)|(^\\<\\=$))",
valores[i]) //Operadores Relacionales
operadoresLogicos, _ := regexp.MatchString("((^\\&\\&$)|(^\\||\\||$)|(^!$))", valores[i])
numerico, _ := regexp.MatchString("^([0-9]*$", valores[i])
//Numerico
finInstruccion, _ := regexp.MatchString("^(\\;)$", valores[i])
//Final de instruccion
agrupadores, _ := regexp.MatchString("^(\\[|\\]|\\(|\\)|\\{|\\})]$",
valores[i]) //Agrupador
incremento, _ := regexp.MatchString("^(\\+\\+)$", valores[i])
//Incremento
decremento, _ := regexp.MatchString("^(\\-\\-)$", valores[i])
//Decremettno
literal, _ := regexp.MatchString("^\"([a-zA-Z]|[0-9]|\\s)*\"$", valores[i])
//Literal
tipoDato, _ :=
regexp.MatchString("((^caracxp$)|(^cadenaxp$)|(^entxp$)|(^bytexp$)|(^cortoxp$)|(^largox
p$)|(^boolxp$)|(^flot anxp$)|(^flotan64xp$)|(^uentxp$)|(^ubypexp$)|(^ucortoxp$)|(^ulargo
xp$)|(^decimalxp$)|(^varxp$)|(^enumxp$)|(^structxp$))", valores[i])
tipoRefere, _ :=
regexp.MatchString("((^claseq$)|(^interfaceq$)|(^delegadoq$)|(^dinamicoq$)|(^objetoq$))
", valores[i])
tipoBooleano, _ := regexp.MatchString("((^verdad$)|(^falso$))", valores[i])
instruccionIteracion, _ :=
regexp.MatchString("((^parala$)|(^paracadala$)|(^mientrasla$)|(^hacerla$)|(^enla$))",
valores[i])
instruccionSeleccion, _ :=
regexp.MatchString("((^sixv$)|(^sinovsixv$)|(^sinov$)|(^cambiarxv$)|(^casov$))",
valores[i])
instruccionSalto, _ :=
regexp.MatchString("((^descansarw$)|(^continuarw$)|(^retornarw$)|(^defectow$)|(^rendimi
entow$))", valores[i])
atraparErrores, _ :=
regexp.MatchString("((^intentarxv$)|(^atraparxv$)|(^finalxv$)|(^lanzarxv$))",
valores[i])
controlAcceso, _ :=
regexp.MatchString("((^publicz$)|(^privadoz$)|(^internoz$)|(^protegidoz$))",
valores[i])
modificadores, _ :=
regexp.MatchString("(^abstractom$)|(^asincronom$)|(^eventom$)|(^externom$)|(^nuevom$)|(^
anularm$)|(^parcialm$)|(^lecturasolom$)|(^selladom$)|(^estaticom$)|(^noguardarm$)|(^vi
rtualm$)|(^volatilm$)|(^constantem$)", valores[i])
parametrosMetodos, _ := regexp.MatchString("((^paramet$)|(^referen$)|(^salida$))",
valores[i])
checks, _ := regexp.MatchString("((^checar$)|(^nochecar$))", valores[i])
revisión, _ := regexp.MatchString("((^arreglado$)|(^bloquear$))", valores[i])
nombresEspacio, _ := regexp.MatchString("((^espacionombre$)|(^usando$)|(^externo$))",
valores[i])
claveConversiones, _ :=
regexp.MatchString("((^explicitoc$)|(^implicitoc$)|(^operadorc$))", valores[i])
claveAcceso, _ := regexp.MatchString("((^base$)|(^esta$)|(^nulo$))", valores[i])
claveContextuales, _ :=
regexp.MatchString("((^obtener$)|(^global$)|(^parcial$)|(^remove$)|(^poner$)|(^evaluar
$)|(^donde$)|(^anadir$))", valores[i])

```

```

claveOperadores, _ :=
regexp.MatchString("((^como$)|(^esperar$)|(^es$)|(^tamañode$)|(^tipode$)|(^stackloco$))", valores[i])
claveConsulta, _ :=
regexp.MatchString("((^desde$)|(^donde$)|(^seleccionar$)|(^grupo$)|(^dentro$)|(^ordenar por$)|(^unirse$)|(^dejar$)|(^en$)|(^sobre$)|(^igual$)|(^por$)|(^acendente$)|(^decendent e$))", valores[i])
impresion, _ :=
regexp.MatchString("((^Efscanf$)|(^Efscanln$)|(^Impri$)|(^Imprif$)|(^Impriln$)|(^Escan$)|(^Escanf$)|(^Escanln$)|(^ESprint$)|(^ESprintf$)|(^ESprintln$)|(^Escan$)|(^EEscanf$))", valores[i])
indicadoresFormato, _ :=
regexp.MatchString("( (^omitir$)|(^izquierda$)|(^derecha$)|(^interno$)|(^deca$)|(^octa$)|(^hexa$)|(^mostrarbase$)|(^mostrarpunto$)|(^letramayus$)|(^mostrarposi$)|(^cientifico$)|(^arreglado$)|(^enterobuf$)|(^justarcampo$)|(^basecampo$)|(^flotantecampo$))", valores[i])
funcPrincipal, _ := regexp.MatchString("((^vaciov$)|(^iniciov$)|(^funcionv$))", valores[i])
palabrasFunciones, _ :=
regexp.MatchString("( (^Contar$)|(^Igualdoble$)|(^Campos$)|(^Funcampo$)|(^tenerprefijo$)|(^Tenersufijo$)|(^Repetir$)|(^Remplazar$)|(^Cortar$))", valores[i])
funcMatematicas, _ :=
regexp.MatchString("((^abstr$)|(^facos$)|(^fasin$)|(^fatan$)|(^fcos$)|(^fcosh$)|(^fexp$)|(^flog$)|(^fmax$)|(^fmin$)|(^fpow$)|(^fsign$)|(^ftan$)|(^ftanh$)|(^ftruncate$))", valores[i])

palabrasBaja, _ :=
regexp.MatchString("((^factivate$)|(^fdouble$)|(^flocators$)|(^frollback$)|(^fadd$)|(^f drop$)|(^flock$)|(^froundceiling$)|(^fafter$)|(^fdssize$)|(^flockmax$)|(^frounddown$)|(^falias$)|(^fdynamic$)|(^froundfloor$)|(^fall$)|(^feach$)|(^flong$)|(^froundhalfdown$)|(^feditproc$)|(^floop$)|(^fallow$)|(^fmaintained$)|(^fasensitive$)|(^frows$)|(^frownumb er$)|(^froutine$)|(^froundup$)|(^fmaterialized$)|(^fencryption$)|(^fassociate$)|(^fmaxv alued$)|(^fdefinition$)|(^flanguage$)|(^frelease$)|(^fvolumes$)|(^frename$)|(^freset$)|(^fleve$)|(^fctype$))", valores[i])

identificador, _ := regexp.MatchString("^([a-zA-Z]([a-zA-Z]|[0-9])*)", valores[i])
//Identificadores

if tipoDato {
fmt.Println(`      |`, total, `      |      TIPO DATO      |      `+valores[i]+`      |`)
} else if tipoBooleano {
fmt.Println(`      |`, total, `      |      TIPO BOOL      |      `+valores[i]+`      |`)
} else if instruccionSalto {
fmt.Println(`      |`, total, `      |      INSTR SALTO      |      `+valores[i]+`      |`)
} else if atraparErrores {
fmt.Println(`      |`, total, `      |      ATRAP ERROR      |      `+valores[i]+`      |`)
} else if controlAcceso {
fmt.Println(`      |`, total, `      |      CONT ACCESO      |      `+valores[i]+`      |`)
} else if asignacion {
fmt.Println(`      |`, total, `      |      ASIGNACION      |      =      |`)
} else if operadoresArit {
fmt.Println(`      |`, total, `      |      OPERA ARIT      |      `+valores[i]+`      |`)
} else if operadoresRela {
fmt.Println(`      |`, total, `      |      OPERA RELA      |      `+valores[i]+`      |`)
} else if numerico {
fmt.Println(`      |`, total, `      |      NUMERICO      |      `+valores[i]+`      |`)
} else if finInstruccion {
fmt.Println(`      |`, total, `      |      FIN INSTRUC      |      ;      |`)
} else if agrupadores {
fmt.Println(`      |`, total, `      |      AGRUPADORES      |      `+valores[i]+`      |`)
} else if incremento {
fmt.Println(`      |`, total, `      |      INCREMENTO      |      ++      |`)
} else if decremento {
fmt.Println(`      |`, total, `      |      DECREMENTO      |      --      |`)
} else if literal {
fmt.Println(`      |`, total, `      |      LITERAL      |      `+valores[i]+`      |`)
}

```

```

} else if modificadores {
fmt.Println(`      |`, total, `      |      MODIFICA      |      `+valores[i]+`      |`)
} else if tipoRefere {
fmt.Println(`      |`, total, `      |      T REFERENCIA      |      `+valores[i]+`      |`)
} else if instriccionSeleccion {
fmt.Println(`      |`, total, `      |      INSTR SELEC      |      `+valores[i]+`      |`)
} else if instruccionIteracion {
fmt.Println(`      |`, total, `      |      INSTR ITERA      |      `+valores[i]+`      |`)
} else if parametrosMetodos {
fmt.Println(`      |`, total, `      |      PARA METOD      |      `+valores[i]+`      |`)
} else if checks {
fmt.Println(`      |`, total, `      |      CHECKS      |      `+valores[i]+`      |`)
} else if revision {
fmt.Println(`      |`, total, `      |      REVISION      |      `+valores[i]+`      |`)
} else if nombresEspacio {
fmt.Println(`      |`, total, `      |      NOMB ESPAC      |      `+valores[i]+`      |`)
} else if claveOperadores {
fmt.Println(`      |`, total, `      |      CLAVE OPERA      |      `+valores[i]+`      |`)
} else if claveConversiones {
fmt.Println(`      |`, total, `      |      CLAVE CONVER      |      `+valores[i]+`      |`)
} else if claveAcceso {
fmt.Println(`      |`, total, `      |      CLAVE ACCESO      |      `+valores[i]+`      |`)
} else if claveContextuales {
fmt.Println(`      |`, total, `      |      CLAVE CONTEXTUA      |      `+valores[i]+`      |`)
} else if claveConsulta {
fmt.Println(`      |`, total, `      |      CLAVE CONSULTA      |      `+valores[i]+`      |`)
} else if indicadoresFormato {
fmt.Println(`      |`, total, `      |      INDICA FORMATO      |      `+valores[i]+`      |`)
} else if impresion {
fmt.Println(`      |`, total, `      |      IMPRESION      |      `+valores[i]+`      |`)
} else if palabrasFunciones {
fmt.Println(`      |`, total, `      |      PALABRAS FUNC      |      `+valores[i]+`      |`)
} else if operadoresLogicos {
fmt.Println(`      |`, total, `      |      OPERA LOGICOS      |      `+valores[i]+`      |`)
}
} else if funcPrincipal {
fmt.Println(`      |`, total, `      |      FUNC PRINCIPAL      |      `+valores[i]+`      |`)
} else if funcMatematicas {
fmt.Println(`      |`, total, `      |      FUNC MATEMAT      |      `+valores[i]+`      |`)
} else if palabrasBaja {
fmt.Println(`      |`, total, `      |      PALABRAS FUNC      |      `+valores[i]+`      |`)
} else if identificador {
fmt.Println(`      |`, total, `      |      ID      |      `+valores[i]+`      |`)
//id[contador] = valores[i]
//contador++
} else {
fmt.Println(`      |`, total, `      |      ERROR      |      `+valores[i]+`      |`)
}

total++ //Aumentador el contador
}

fmt.Println(`      \ ----- /`)
fmt.Println()
}

```

Archivo de las Palabras Reservadas

Texto que lee el programa.

= == != < > <= >= ++ -- + - * / ^ && || ! verdad falso caracxp cadenaxp entxp bytexp cortoxp largoxp boolxp
flotaxp flotan64xp uentxp ubytexp ucortoxp decimalxp varxp enumxp structxp parala paracadala
mientrasla hacerla enla sixv sinovsixv sinovxv cambiarxv casovxv descansarw continuarw retornarw
defectow rendimientow intentarxv atraparxv finalxv lanzarxv publicz privadoz internoz protegidoz publicz
privadoz internoz protegidoz claseq interfaceq delegadoq dinamicoq objetoq abstractom asincronom
constantem eventom externom nuevomanularm parcialm lecturasolom selladom estaticom noguardarm
virtualm volatilism checar nocheckar paramet referen salida arreglado bloquear espacionombre usando
externo como esperar es tamañode tipode stackloco explicitoc implicitoc operadorc base esta nulo obtener
global parcial remover poner evaluar donde anadir desde donde seleccionar grupo dentro ordenarpor
unirse dejar en sobre igual por acendente decendente omitir izquierda derecha interno deca octa hexa
mostrarbase mostrarpunto letramayus mostrarposi cientifico arreglado enterobuf justarcampo basecampo
flotantecampo Efscanf Efscanfln Impri Imprif Impriln Escan Escanf Escanln ESprint ESprintf ESprintln
Eescan EEscanf Contar Igualdoble Campos Funcampo tenerprefijo Tenersufijo Repetir Remplazar Cortar
vaciov iniciov funcionv abstr facos fasin fatan fcos fcosh fexp flog fmax fmin fpow fsign ftan ftanh ftruncate
factivate fdouble flocators frollback fadd fdrop flock froundceiling fafter fdssize flockmax frounddown
falias fdynamic froundfloor fall feach flong froundhalfdown feditproc floop fallow fmaintained fasensitive
frows frownnumber froutine froundup fmaterialized fencryption fassociate fmaxvalued fdefinition flanguage
frelease fvolumes frename freset fleve fctype { } ()

Programa Ejecutado

MINGW64:/c/Users/Angel Pasillas/Desktop/Lenguajes y Automatas 1

/	N	TOKEN	LEXEMA	\
	0	ASIGNACION	=	
	1	OPERA RELA	==	
	2	OPERA RELA	!=	
	3	OPERA RELA	<	
	4	OPERA RELA	>	
	5	OPERA RELA	<=	
	6	OPERA RELA	>=	
	7	INCREMENTO	++	
	8	DECREMENTO	--	
	9	OPERA ARIT	+	
	10	OPERA ARIT	-	
	11	OPERA ARIT	*	
	12	OPERA ARIT	/	
	13	OPERA ARIT	^	
	14	OPERA LOGICOS	&&	
	15	OPERA LOGICOS		
	16	OPERA LOGICOS	!	
	17	TIPO BOOL	verdad	
	18	TIPO BOOL	falso	
	19	TIPO DATO	caracxp	
	20	TIPO DATO	cadenaxp	
	21	TIPO DATO	entxp	
	22	TIPO DATO	bytexp	
	23	TIPO DATO	cortoxp	
	24	TIPO DATO	largoxp	
	25	TIPO DATO	boolxp	
	26	TIPO DATO	flotanxp	
	27	TIPO DATO	flotan64xp	
	28	TIPO DATO	uentxp	
	29	TIPO DATO	ubytexp	
	30	TIPO DATO	ucortoxp	
	31	TIPO DATO	decimalxp	
	32	TIPO DATO	varxp	
	33	TIPO DATO	enumxp	
	34	TIPO DATO	structxp	
	35	INSTR ITERA	para la	
	36	INSTR ITERA	paracada la	