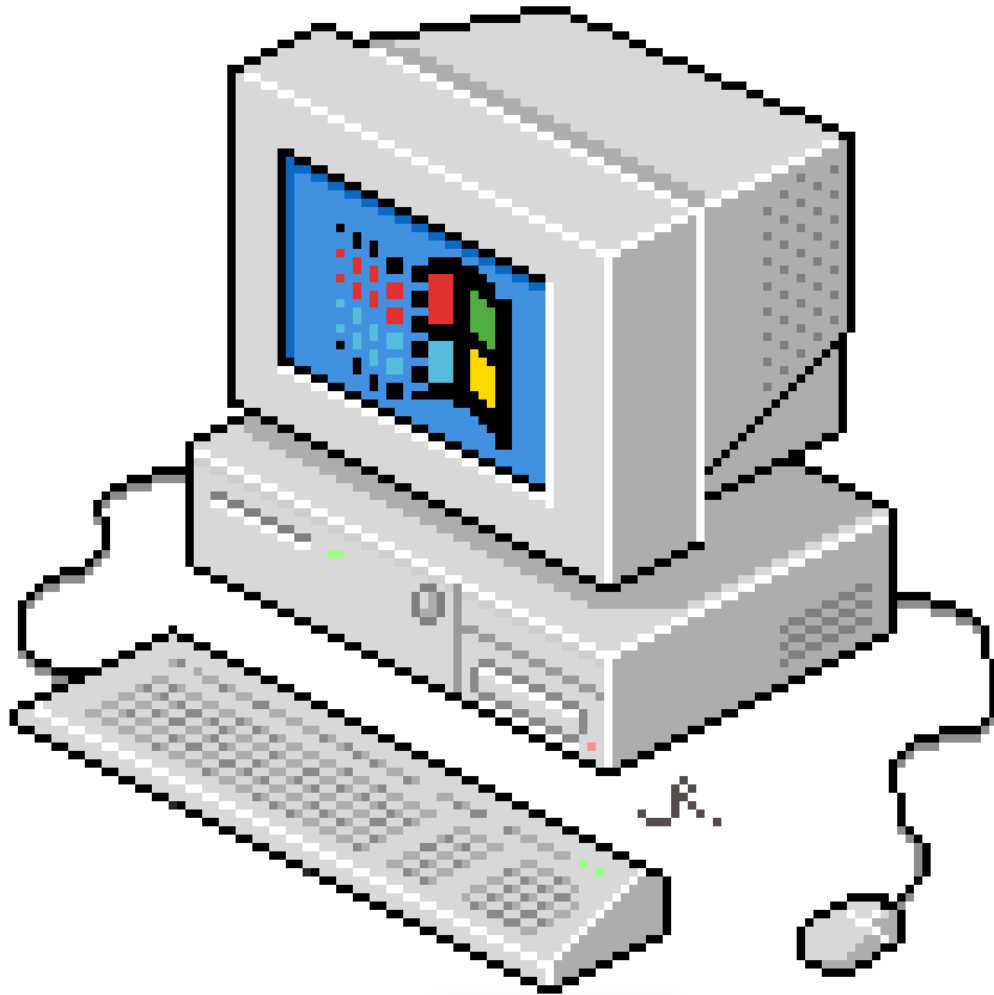


MANUAL TECNICO

PIXE ART



Erick Daniel Ajche Hernandez

201701043

Proyecto 1

Contenido

Introducción.....	3
Objetivos	4
General	4
Específicos	4
Descripción de la Solución	5
IDE	6
Requerimientos:.....	7
Sistema Operativo	7
Librerías Utilizadas.....	7
Otras especificaciones:.....	7
Carpetas:	8
Archivo(.pxla):	8
Analizador Léxico:.....	8
Expresión Regular	11
Expression Regular	11
Autómata Finito Determinista.....	21

Introducción

El fin de este documento es explicar el funcionamiento del programa, como manipula los datos, los asignación de los mismo, dando detalle lógico de porque el programa actual de tal manera. Explicando paso a paso como el programa actúa en ejecución.

Lo importante de la aplicación, es leer ciertos archivos de texto, reportando cada uno de los caracteres, identificando cada uno de sus símbolos o caracteres, además de realizar dibujo estilo pixelart, en los cuales se traducirá el archivo de texto.

Objetivos

General

Da indicaciones del funcionamiento de la aplicación, como de sus distintos métodos para realizar todas las acciones que el programa puede hacer, así como la creación de los reportes, dibujos, HTML y CSS.

Específicos

- Explicación concreta del reconocimiento de tokens con los distintos lexemas que el archivo de entrada incluye.
- Dar a entender el reconocimiento de errores y como el programa se recupera de alguno de ellos.
- La creación de los distintos archivos para crear los dibujos que nos indica cada archivo de entrada.

Descripción de la Solución

El proyecto se desarrollo en base a las necesidades del negocio de PixelArt, ya que necesitaba una aplicación que además de dibujar, tiene que reconocer los distintos archivos de entrada para verificar que no existan errores y si los hay evitar ese archivo.

Para la ejecución correcta del programas como paso inicial contar con un archivo de entrada que sea adecuado a lo que pide el programa, este debe de contar con estos datos:

Archivo de entrada: Debe de contener uno o varios dibujos, estos contendrán distintas características.

Tipo de archivo: Este archivo debe de ser de extensión (.pxla).

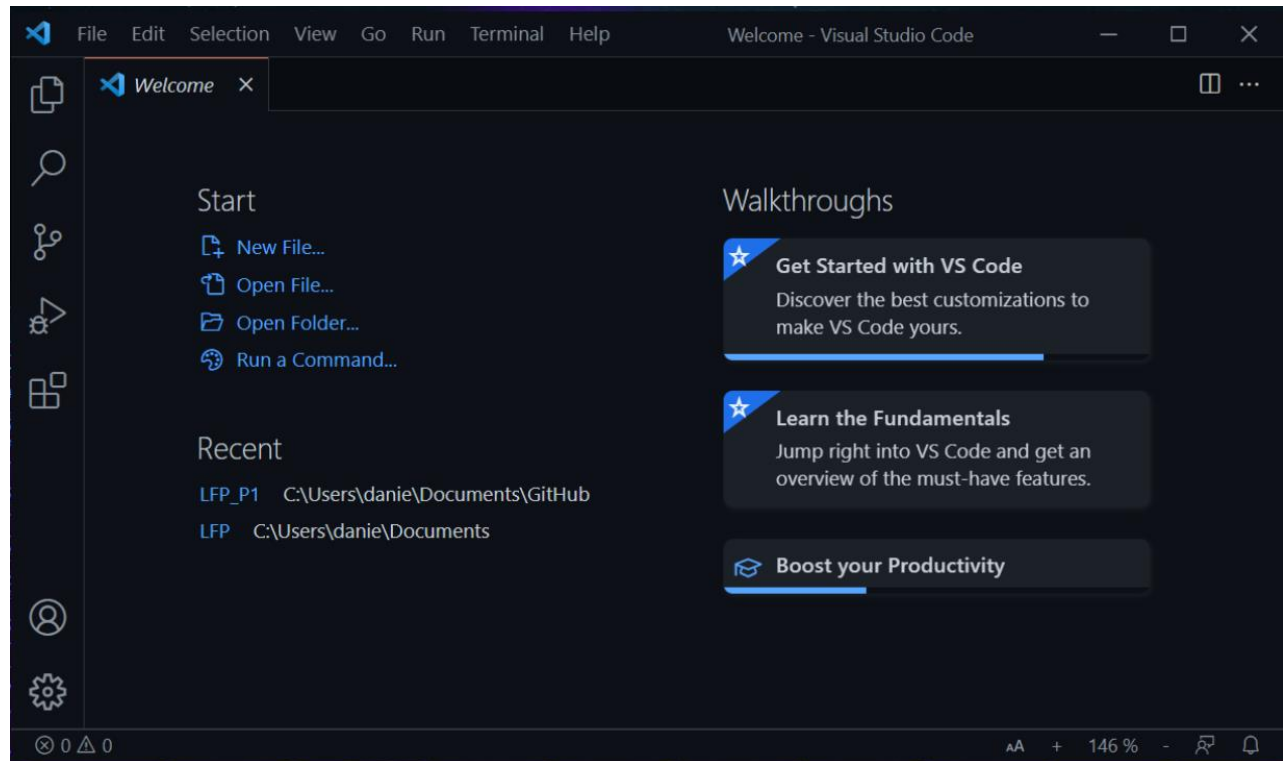
Reportes: Para esto debe de haber ingresado algún archivo antes y haberlo analizado.

Dibujar: Para esto el archivo ingresado debe de haber sido ya analizado.

Contenido

IDE

El programa fue desarrollado en Visual Studio Code, en el lenguaje de programación Python 3.9.6, se usaron estas herramientas por la fácil detección de errores y su versátil manejo de distintos datos, para ser más eficiente el programa, la interfaz grafica fue hecha con la librería tkinter.



Requerimientos:

Es un programa siempre por lo que cualquier tipo de sistema operativo de los últimos 15 años podría correr el programa fácilmente. mínimo 500Mb de RAM.

Sistema Operativo

Fue desarrollado en Windows 10 de 64bits.

Edición de Windows

Windows 10 Pro

© Microsoft Corporation. Todos los derechos reservados.



Sistema

Procesador:	Intel(R) Core(TM) i7-3632QM CPU @ 2.20GHz 2.20 GHz	Información de soporte técnico
Memoria instalada (RAM):	8.00 GB (7.88 GB utilizable)	
Tipo de sistema:	Sistema operativo de 64 bits, procesador x64	
Lápiz y entrada táctil:	La entrada táctil o manuscrita no está disponible para esta pantalla	

Librerías Utilizadas

1. import os
2. import tkinter
3. import filedialog
4. import PIL
5. import imgkit
6. import enum




Otras especificaciones:

El estilo de la tabla es de <https://dev.to/dcodeyt/creating-beautiful-html-tables-with-css-428l>.

Funcionamiento de Código

Carpetas:

Contamos con 3 carpetas, una contiene el es estilo de la tabla para realizar los reportes, otro contiene la interfaz grafica del programa y el otro el manejo de datos y funcionamiento lógico.

 interface.py	M
 main.py	M
 style.css	

Archivo(.pxla):

Este debe de contener distintos caracteres, e información para poder realizar el dibujo.

```
TITULO="Pokebola";
ANCHO=300;
ALTO=300;
FILAS=12;
COLUMNAS=12;
CELDAS = {
    [0,0,FALSE,#000000],
    [0,1,FALSE,#000000],
```

Analizador Léxico:

Realizamos un de carácter por carácter, hasta determinar el tipo de token que era, o si era un error. Formando cadenas de texto, o secuencias de números o reconoces símbolos.

```
aux = Letras()
if aux == "FALSE" or aux == "TRUE":
    celdtemp.setPaint(aux)
    NToken = Token(tipo.BOOL,aux,Fila,Columna)
    TokensDraw.append(NToken)
    nceld == 4
elif aux == "MIRRORX" or aux == "MIRROR" or aux == "DOUBLEMIRROR":
    actualD.addFilter(aux)
    NToken = Token(tipo.FILTRO,aux,Fila,Columna)
    TokensDraw.append(NToken)
else:
    if aux == "TITULO":
        NToken = Token(tipo.RESERVADA,aux,Fila,Columna)
        TokensDraw.append(NToken)
        actualD.addlast(aux)
    elif aux == "ANCHO":
        NToken = Token(tipo.RESERVADA,aux,Fila,Columna)
        TokensDraw.append(NToken)
        actualD.addlast(aux)
    elif aux == "ALTO":
        NToken = Token(tipo.RESERVADA,aux,Fila,Columna)
        TokensDraw.append(NToken)
        actualD.addlast(aux)
    elif aux == "FILAS":
        NToken = Token(tipo.RESERVADA,aux,Fila,Columna)
        TokensDraw.append(NToken)
        actualD.addlast(aux)
```


Mostrar Imagen:

Para este apartado, realizamos el dibujo desde que analizamos el texto, así poder mostrarlo directamente.

Para el dibujo creamos un archivo CSS, el cual le da el color a los pixeles, el tamaño y cuantas filas y columnas, después el HTML, y dibuja los pixeles, y ya creado el HTML, tenemos nuestro dibujo hecho, lo convertimos a un archivo de imagen JPG con ayuda de la librería imgkit, ya en la interfaz grafica solo mostramos en un label la imagen.

```
file = open(dir_path+"\\STYLE_2_"+named+".css", "w")
if Filt == 3:
    file = open(dir_path+"\\STYLE_3_"+named+".css", "w")
file.write("""body {
display: flex;
}""")
file.write(".canvas {\n")
file.write("width: "+pixX+"px;\n")    /* Ancho del lienzo, se asocia al ANCHO de la entrada */
file.write("height: "+pixY+"px;\n") /* Alto del lienzo, se asocia al ALTO de la entrada */
file.write(".pixel {\n")
file.write("width: "+str(int(TamX))+"px;\n")    /* Ancho de cada pixel, se obtiene al operar ANCHO/COLUMNA */
file.write("height: "+str(int(TamY))+"px;\n")    /* Alto de cada pixel, se obtiene al operar ALTO/FILAS (al) */
file.write("float: left;\n")
file.write("box-shadow: 0px 0px 1px #fff;}\n") /*Si lo comentan les quita la cuadrícula de fondo */
if Filt == 0:
    ForCelds0(Drawing)
if Filt == 1:
    ForCeldsMx(Drawing)
if Filt == 2:
    ForCeldsMy(Drawing)
if Filt == 3:
    ForCeldsMxy(Drawing)
lsceld = Drawing.celdas
for cel in lsceld:
    if cel.paint:
        file.write(".pixel:nth-child("+str(cel.nceld)+"){background: "+str(cel.color)+";}\n")
```

Para los filtros dependiendo de cual asignamos los colores a los pixeles de distinta forma, el HTML y el CSS se realiza uno para cada filtro de cada imagen.

```
def ForCelds0(Drawing):
    listac=Drawing.celdas
    fil=int(Drawing.nFil)
    col=int(Drawing.nCol)
    n=1
    for i in range(0,fil,1):
        for j in range(0,col,1):
            for cel in listac:
                if int(cel.coorx) == j and int(cel.coory)==i:
                    cel.nceld = n
                    n=n+1
```

Reportes

Para los reportes desde que analizamos el archivo vamos anotando cada token, lexema o error que el archivo contenga, así y en una tabla mostramos los todos los lexemas identificados, y en otro diferente los errores. Este se creará donde está ubicada la carpeta del programa.

```
def ReportError(doc):
    named = doc.namedoc
    dir_path = os.path.dirname(os.path.realpath(__file__))
    file = open(dir_path+"\\ReporteErrores_"+named+".html", "w")
    file.write('<!DOCTYPE html><html><head><link rel="stylesheet" href="style.css"></head>')
    file.write("<h1>Errores del documento "+named+".</h1>")
    file.write('<table class="styled-table"><thead><tr><td>No.</td><td>Tipo</td><td>Lex</td></tr></thead><tbody>')
    file.write("<tbody>")
    ListT = doc.errors
    n=1
    for i in ListT:
        file.write('<tr><td>'+str(n)+'</td><td>'+str(i.type).replace("tipoE.", "")+'</td><td>'+str(i.lexema)+'</td></tr>')
        n+=1
    file.write("</tbody></table>")
    file.close()
```

Solo dejara mostrar reportes, de los archivos que ya estén analizados.

```
def ReportToken(doc):
    named = doc.namedoc
    dir_path = os.path.dirname(os.path.realpath(__file__))
    file = open(dir_path+"\\ReporteTokens_"+named+".html", "w")
    file.write('<!DOCTYPE html><html><head><link rel="stylesheet" href="style.css"></head><body>')
    file.write("<h1>Tokens del documento "+named+".</h1>")
    file.write('<table class="styled-table"><thead><tr><td>No.</td><td>Tipo</td><td>Lexema</td><td>Fila</td><td>Columna</td></tr></thead><tbody>')
    file.write("<tbody>")
    ListT = doc.tokens
    n=1
    for i in ListT:
        file.write('<tr><td>'+str(n)+'</td><td>'+str(i.token).replace("tipo.", "")+'</td><td>'+str(i.lexema)+'</td><td>'+str(i.fila)+'</td><td>'+str(i.columna)+'</td></tr>')
        n+=1
    file.write("</tbody></table>")
    file.close()
```

Expresión Regular

L={a-z,A-Z,ñ,Ñ}

N={1,2,3,4,5,6,7,8,9,0}

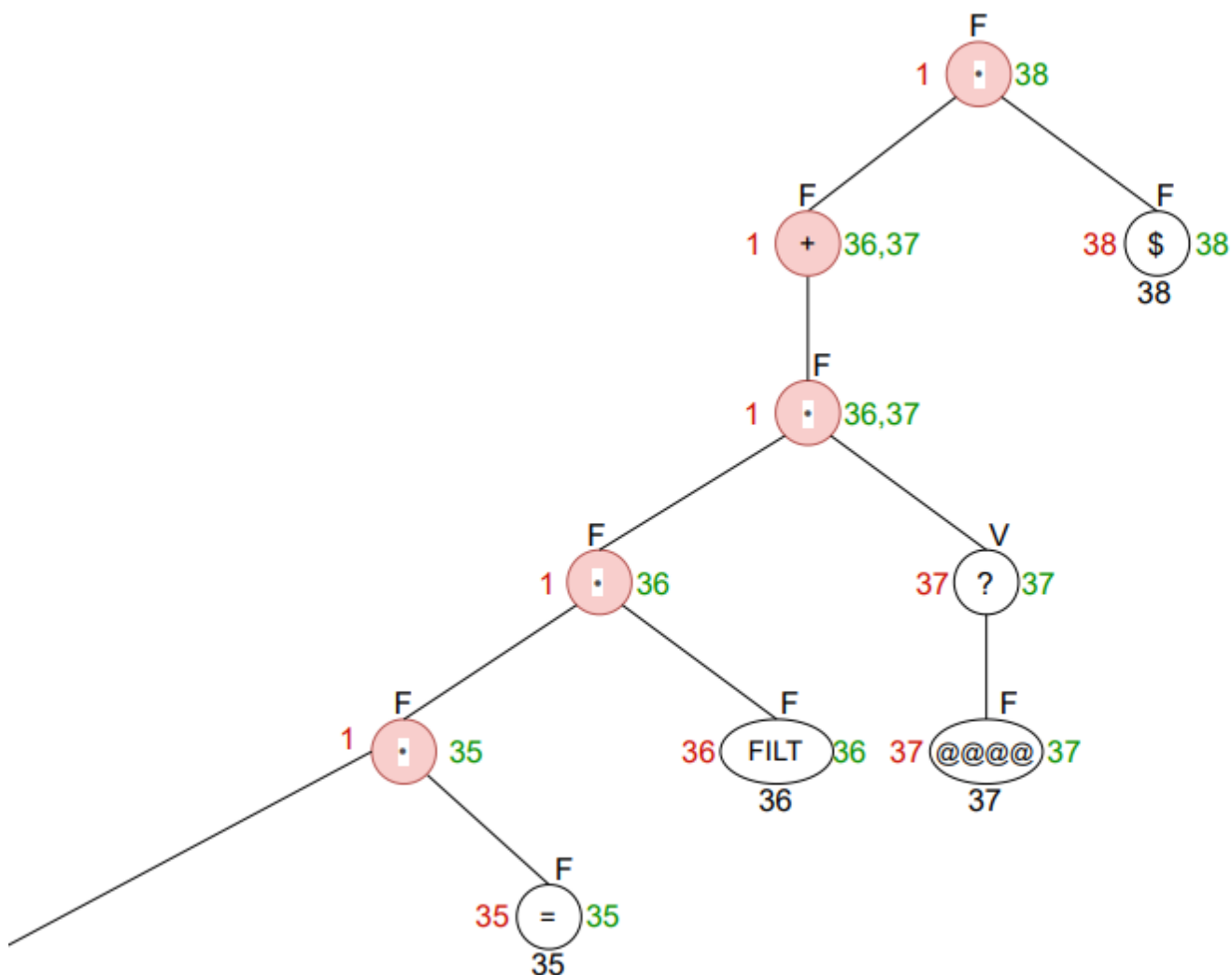
RESV={TITULO,COLUMNA,FILA,ANCHO,ALTO,CELDA,FILTRO}

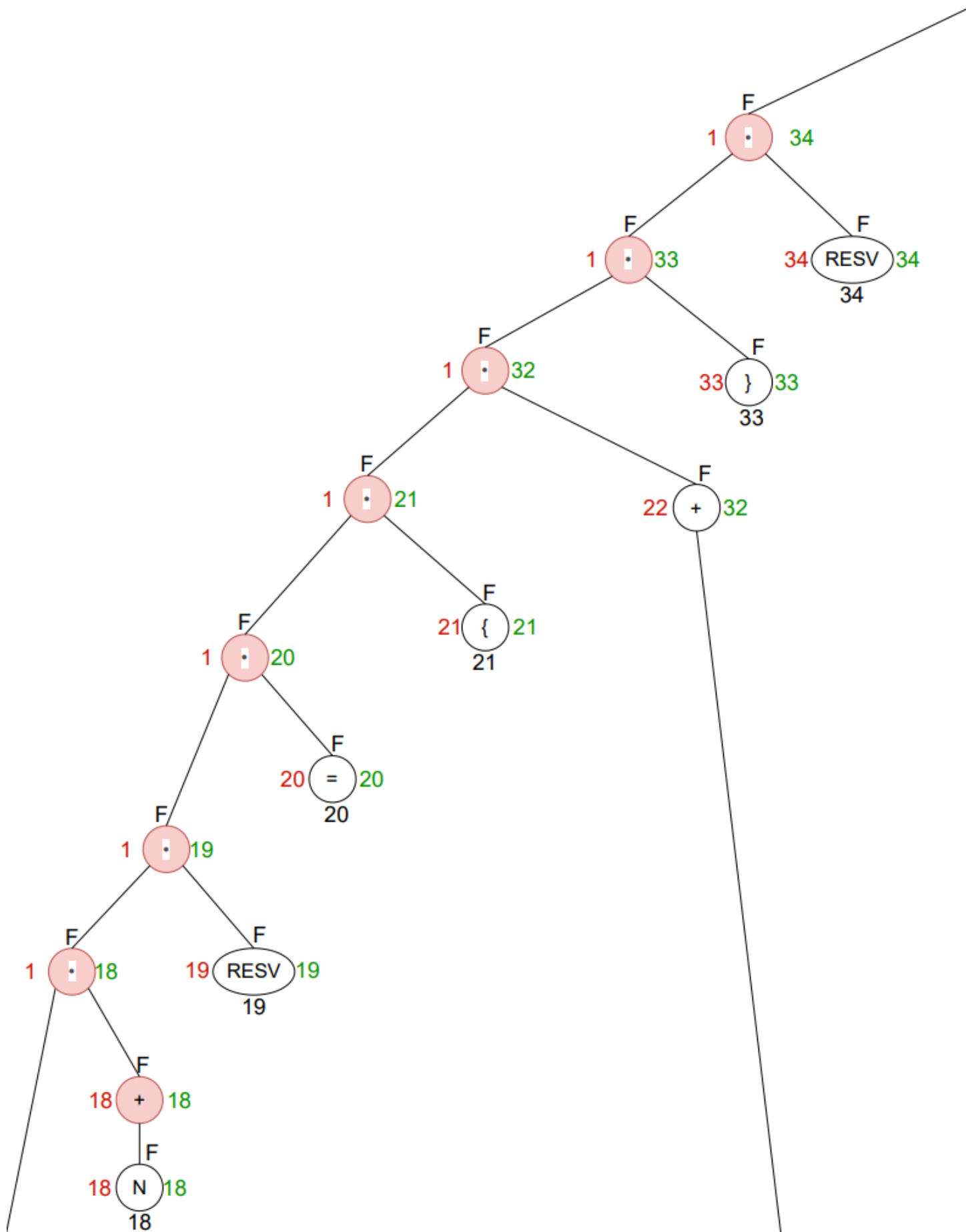
FILT={MIRRORX,MIRROR,DOUBLEMIRROR}

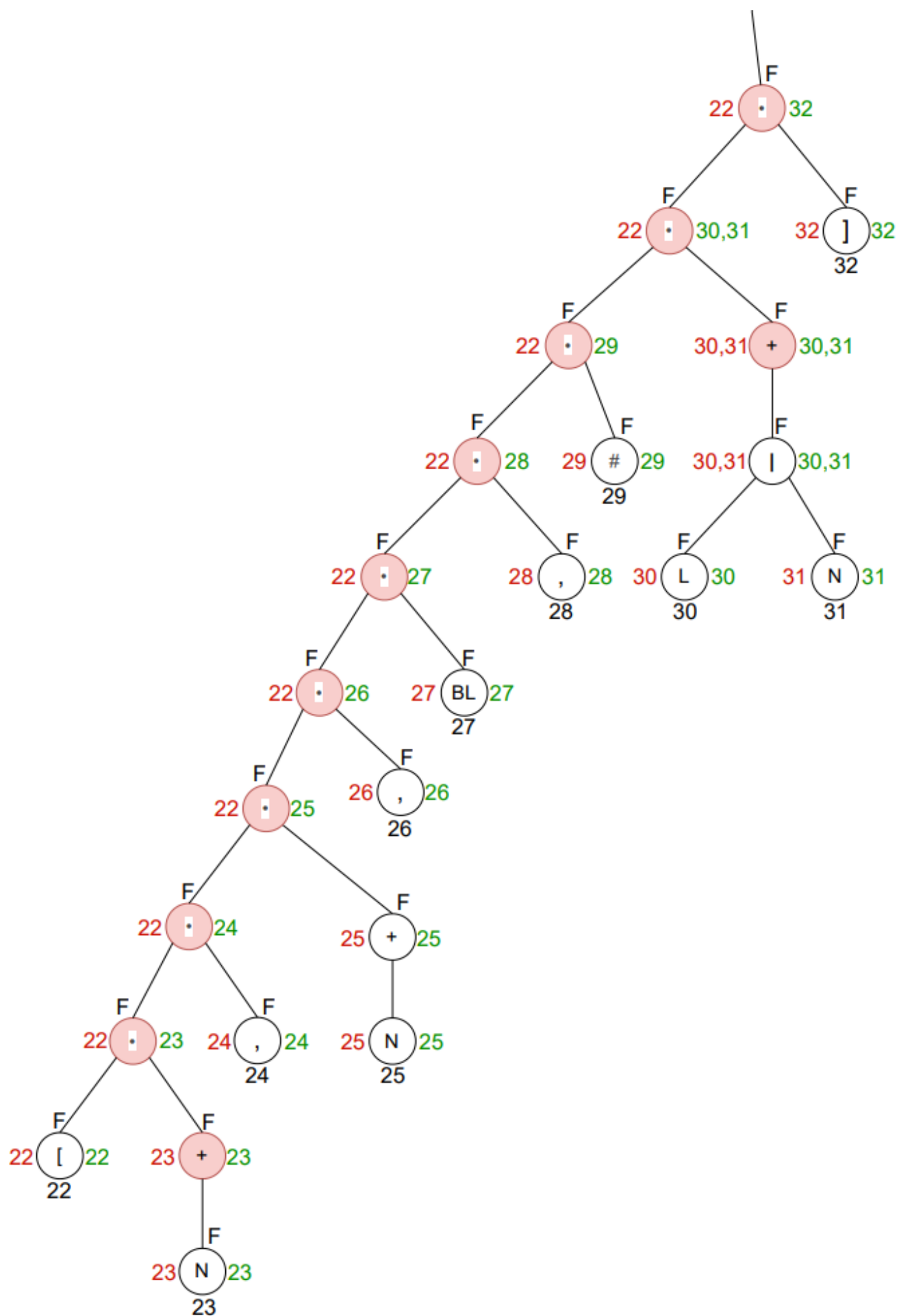
BL={TRUE,FALSE}

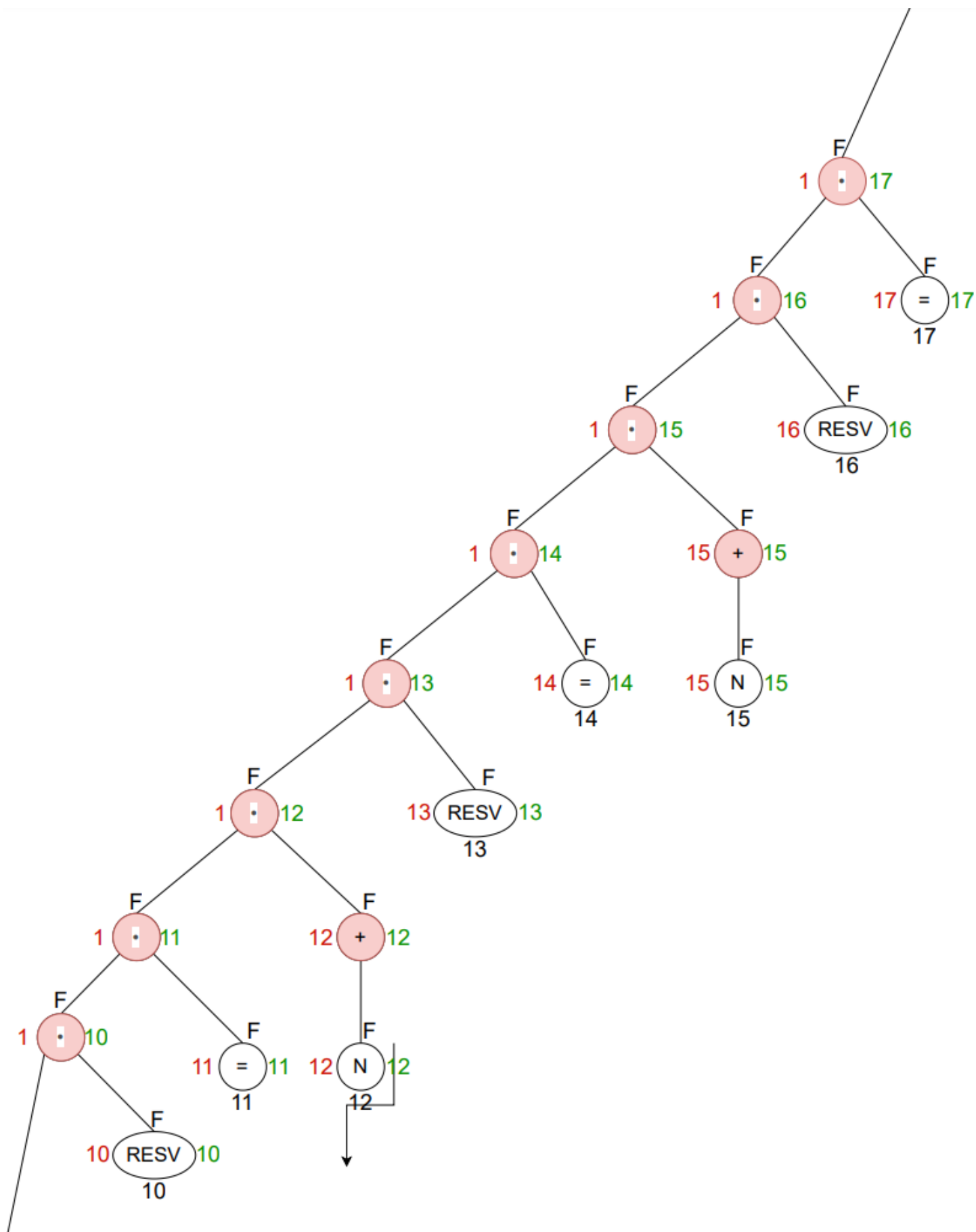
Expression Regular

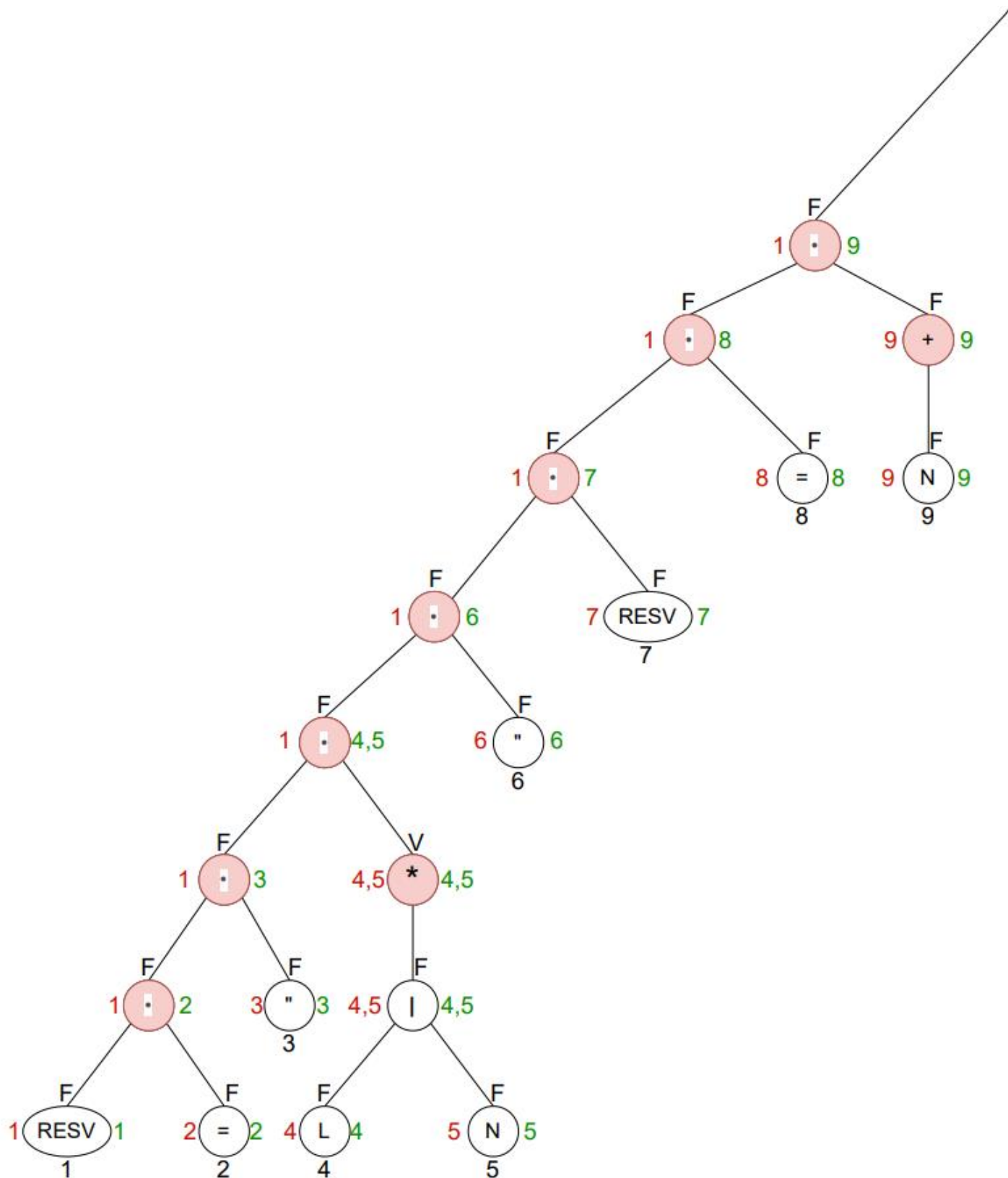
RESV '=' '(' '*' '(' RESV '=' N+ RESV '=' N+ RESV '=' N+ RESV '=' N+ RESV '=' '{' '[' N+ ',' N+ ',' BL ',' '#' (N|L)+ ']' ')' RESV? '='? FILT*











Valor	Hoja	Siguientes
RESV	1	2
=	2	3
"	3	4,5
L	4	4,5,6
N	5	4,5,6
"	6	7
RESV	7	8
=	8	9
N	9	9,10
RESV	10	11
=	11	12
N	12	12,13
RESV	13	14
=	14	15
N	15	15,16
RESV	16	17
=	17	18
N	18	18,19
RESV	19	20
=	20	21
{	21	32
[22	23
N	23	23,24
,	24	25
N	25	25,26
,	26	27
BL	27	28
,	28	29
#	29	30,31
L	30	30,31,32
N	31	30,31,32
]	32	22,33
}	33	34,35,36,37,38
RESV	34	35,
=	35	36
FILT	36	36,37
@@@@	37	38
\$	38	--

	Estado	Valores	Siguientes
Inicio	So	1 RESV	RESV:{1}=S0
	S0	1 RESV	'=: {2}=S1
	S1	2 '=	' " ':{3}=S2
	S2	3 ' " '	L:{4}=S3 N:{5}=S3 ' " ':{6}=S4
	S3	4,5 L,N	L:{4,5}=S3 N:{4,5}=S3 ' " ':{6}=S4
	S4	6 ' " '	RESV:{7}=S5
	S5	7 RESV	'=: {8}=S6
	S6	8 '=	N:{9}=S7
	S7	9 N	N:{9}=S7 RESV:{10}=S8
	S8	10 RESV	'=: {11}=S9
	S9	11 '=	N:{12}=S10
	S10	12 N	N:{12}=S10 RESV:{13}=S11
	S11	13 RESV	'=: {11}=S12
	S12	14 '=	N:{15}=S13
	S13	15 N	N:{15}=S13 RESV:{16}=S14
	S14	16 RESV	'=: {17}=S15
	S15	17 '=	N:{18}=S16

S16	18 N	N:{18}=S16 RESV:{19}=S17
S17	19 RESV	'=: {20}=S18
S18	20 '=	'{':{21}:S19
S19	'{ 21	'{':{22}=S20
S20	22 '[N:{23}=S21
S21	23 N	N:{23}=S21 '':{24}=S22
S22	24 '',	N:{25}=S23
S23	25 N	N:{25}=S23 '':{26}=S24
S24	26 '',	BL:{27}=S25
S25	27 BL	'':{28}=S26
S26	28 '',	'#{':{29}=S27
S27	29 '#'	L:{30}=S27 N:{31}=S27
S28	30,31,32 L,N,'J'	L:{30}=S28 N:{31}=S28 'J':{32}=29
S29	32 'J'	'{':{22}=S19 '{':{33}=S30
S30	33 '}	RESV:{34}=S31
S31	34 RESV	'=: {35}=S32
S32	35 '=	FILT:{36}=S33

	S33	36 FILT	FILT:{36}=S33 '@@@@@':{37}=So \${38}=S34
Aceptación	S34	38 \$	--

[illegible]

Autómata Finito Determinista

