Protocolo

Dueñas Jiménez Cristian Alexis
Teoría de la computación

Octubre 2022

1 Introducción

El problema se nos plantea de la siguiente manera: Debe de verificar si el protocolo está encendido o apagado y ejecutarse nuevamente si está encendido. El programa deberá determinar automáticamente en que momento detenerse. Generar 10^6 cadenas binarias aleatoriamente de longitud 64. Posteriormente validar cada una de las cadenas con el AFD de imparidad.

2 Marco Teórico

Un**autómata finito determinista**, es la implementación de un autómata que sólo puede estar en un único estado después de leer cualquier secuencia de entradas. El término "determinista" hace referencia al hecho de que para cada entrada sólo existe uno y sólo un estado al que el autómata puede hacer la transición a partir de su estado actual.En palabras más simples, en un DFA(o AFD, por sus siglas en inglés y español respectivamente), sólo es posible seguir un camino por condición. Un Autómata Finito Determinista consta de:

1. Un conjunto finito de estados, a menudo designado como Q.
2. Un conjunto finito de símbolos de entrada, a menudo designado como Σ (sigma).
3. Una función de transición que toma como argumentos un estado y un símbolo de entrada y devuelve un estado. La función de transición se designa habitualmente como δ o Δ (delta).
4. Un estado inicial, uno de los estados de Q.
5. Un conjunto de estados finales o de aceptación F. El conjunto F es un subconjunto de Q.

La representación más sucinta de un AFD consiste en un listado de los cinco componentes anteriores. Normalmente, en las demostraciones, definiremos un AFD utilizando la notación de quíntupla siguiente: $A = (Q, \Sigma, \delta, q0, F)$

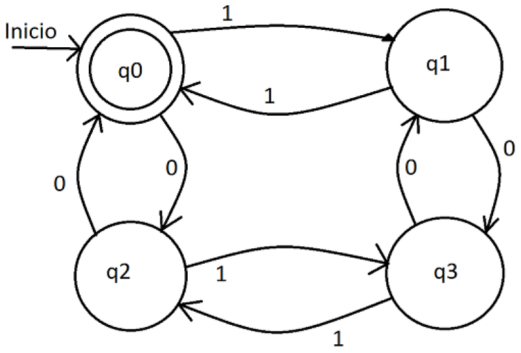


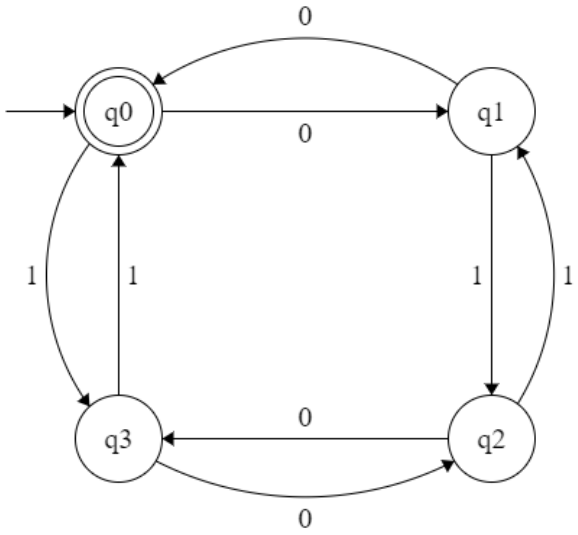
Figure 1: Representación gráfica de un DFA

Un **autómata finito no determinista**(NFA o AFN por sus siglas en inglés y español respectivamente), 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, es decir, es aquel que, a diferencia de los autómatas finitos deterministas, posee al menos un estado , tal que para un símbolo del alfabeto, existe más de una transición posible. En palabras más simples, en un NFA tenemos varios caminos posibles dada una condición. Un Autómata Finito No Determinista consta de:

1. Un conjunto finito de estados, a menudo designado como Q.
2. Un conjunto finito de símbolos de entrada, a menudo designado como Σ (sigma).
3. Una función de transición que toma como argumentos un estado y un símbolo de entrada y devuelve un estado. La función de transición se designa habitualmente como δ o Δ (delta).
4. Un estado inicial, uno de los estados de Q.
5. Un conjunto de estados finales o de aceptación F. El conjunto F es un subconjunto de Q.

3 Desarrollo

En la implementación, primero planteamos un autómata que nos permita verificar si existe paridad en la cadena.



Representación gráfica del automata de paridad.

El programa básicamente sigue a ese autómata, y al llegar al estado final, determina que tiene paridad.

```
import os
import sys
from functions import *

count = 0
iteration = 1
v1 = getRandomNumber(1)

if (iteration == 1) and (v1 == 0):
    print("El valor que se obtuvo fue 0. Protocolo cerrado.")
    sys.exit()

#Creamos los respectivos archivos
f1 = open("../PARCIAL 1/PROGRAMAS/Programa3/Version2/Outputs/
    stringBinary.txt", "w")
f2 = open("../PARCIAL 1/PROGRAMAS/Programa3/Version2/Outputs/
    parityBinary.txt", "w")
f3 = open("../PARCIAL 1/PROGRAMAS/Programa3/Version2/Outputs/
    notParityBinary.txt", "w")
f4 = open("../PARCIAL 1/PROGRAMAS/Programa3/Version2/Outputs/
    history.txt", "w")
```

```
18
19 #Generamos el bucle, que mantendr  activo el protocolo
20 while v1 == 1:
21     print("Protocolo activo: " + os.linesep + "Veces que se ha
       activado: ")
22     print(str(iteration))
23     f1.write("Ciclo " + str(iteration) + os.linesep)
24     f2.write("Cadenas del Ciclo " + str(iteration) + " con paridad:
        " + os.linesep)
25     f3.write("Cadenas del Ciclo " + str(iteration) + " sin paridad:
        " + os.linesep)
26     f4.write("Historial para el Ciclo " + str(iteration) + os.
       linesep)
27     #Mandamos a llamar nuestra funci n que grafica el automata.
28     if((v1 == 1) and (iteration == 1)):
29         graphicMode()
30     iteration+=1    #Incrementamos la iteraci n por cada bucle
31     for x in range (0,1000000):   #Hacemos un for para generar el
       mill n de cadenas binarias.
32         rand = getRandomNumber(500000)   #Establecemos que se cree
       un n mero random
33         binary = bin(rand)[2:]    #Lo convertimos a una cadena
       binaria
34         f1.write(binary + ", ")
35         automata(binary, f2, f3, f4)     #Le mandamos la cadena
       binaria y los archivos para que pueda escribir sobre ellos.
36
37     f1.write(os.linesep)
38     f2.write(os.linesep)
39     f3.write(os.linesep)
40     f4.write(os.linesep)
41     v1 = getRandomNumber(1)
```

Algoritmo 1: Implementación de DFA

Este programa manda a llamar a una libreria, la cual generamos nosotros mismos, en esta librería viene la función que contiene el autómata, y a su vez grafica al mismo.

```
1 import random
2 from graphics import *
3 from draws import *
4
5 def getRandomNumber(lim):
6     randomNumber = random.randint(0,lim)
7     return randomNumber
8
9 def graphicMode():
10     window = GraphWin('Aut mata de Paridad para cadenas Binarias',
        900, 1500)
11
12     message = graphicText(window.getWidth()/2, 20, 'Aut mata -
       Paridad Cadenas Binarias', 24, "black", "courier")
13     message.setStyle("italic")
14     message.draw(window)
15
16     head0 = graphicCircle(300,280,35, "white", "purple", 3)
17     head01 = graphicCircle(300,280,30, "white", "purple", 3)
```

```
    head1 = graphicCircle(450,410,35, "white", "purple", 3)
    head2 = graphicCircle(600,280,35, "white", "purple", 3)
    head3 = graphicCircle(450,150,35, "white", "purple", 3)

    state0 = graphicText(300,280,'q0',16,"black","arial")
    state1 = graphicText(450,410,'q1',16,"black","arial")
    state2 = graphicText(600,280,'q2',16,"black","arial")
    state3 = graphicText(450,150,'q3',16,"black","arial")

    t1 = graphicText(390, 230, '0', 13, "black", "arial")
    t2 = graphicText(350, 195, '0', 13, "black", "arial")
    t3 = graphicText(510, 230, '1', 13, "black", "arial")
    t4 = graphicText(550, 195, '1', 13, "black", "arial")
    t5 = graphicText(390, 330, '1', 13, "black", "arial")
    t6 = graphicText(350, 365, '1', 13, "black", "arial")
    t7 = graphicText(510, 330, '0', 13, "black", "arial")
    t8 = graphicText(550, 365, '0', 13, "black", "arial")

    lineinitial = graphicLine(265,280,200,280)
    line0a3 = graphicLine(415,155,310,245)
    line3a0 = graphicLine(330,265,430,180)
    line1a0 = graphicLine(310,315,415,400)
    line0a1 = graphicLine(435,380,330,295)
    line1a2 = graphicLine(590,315,485,400)
    line2a1 = graphicLine(465,380,570,295)
    line3a2 = graphicLine(570,265,470,180)
    line2a3 = graphicLine(485,155,590,245)

    #Automata Protocolo
    lineinit2 = graphicLine(315,540,235,540)
    head00 = graphicCircle(350, 540, 35, "white", "purple", 3)
    head10 = graphicCircle(550, 540, 35,  "white", "purple", 3)
    text0 = graphicText(275, 530, 'init', 14, "black", "arial")
    text1 = graphicText(350,540,'q0',16,"black","arial")
    text2 = graphicText(550,540,'q1',16,"black","arial")
    text3 = graphicText(450,500,'data',12,"black","arial")
    text4 = graphicText(450,580,'back',12,"black","arial")
    text5 = graphicText(650,540,'timeout',12,"black","arial")
    lineSecond = graphicLine(530,510,370,510)
    lineSecond1 = graphicLine(370,570,530,570)
    lineUlti = graphicLine(570,570,615,540)
    lineUlti2 = graphicLi(570,510,615,540)

    head0.draw(window)
    head01.draw(window)
    head1.draw(window)
    head2.draw(window)
    head3.draw(window)

    state0.draw(window)
    state1.draw(window)
    state2.draw(window)
    state3.draw(window)

    lineinitial.draw(window)
    line0a3.draw(window)
    line3a0.draw(window)
```

```
    line1a0.draw(window)
    line0a1.draw(window)
    line1a2.draw(window)
    line2a1.draw(window)
    line3a2.draw(window)
    line2a3.draw(window)

    t1.draw(window)
    t2.draw(window)
    t3.draw(window)
    t4.draw(window)
    t5.draw(window)
    t6.draw(window)
    t7.draw(window)
    t8.draw(window)

    #Dibujamos el automaa de protocolo
    lineinit2.draw(window)
    lineUlti.draw(window)
    lineUlti2.draw(window)
    head00.draw(window)
    head10.draw(window)
    text0.draw(window)
    text1.draw(window)
    text2.draw(window)
    text3.draw(window)
    text4.draw(window)
    text5.draw(window)
    lineSecond.draw(window)
    lineSecond1.draw(window)

    window.getMouse()
    window.close()

def automata(cadena,aceptar,rechazar,hist):
    estado = 0
    for char in cadena:
        if estado == 0:
            if char == "0":
                estado = 1
            if char == "1":
                estado = 3
            hist.write("(q0,\""+char+"\",q"+str(estado)+")\n")
            continue

        if estado == 1:
            if char == "0":
                estado = 0
            if char == "1":
                estado = 2
            hist.write("(q1,\""+char+"\",q"+str(estado)+")\n")
            continue

        if estado == 2:
            if char == "0":
                estado = 3
            if char == "1":
```

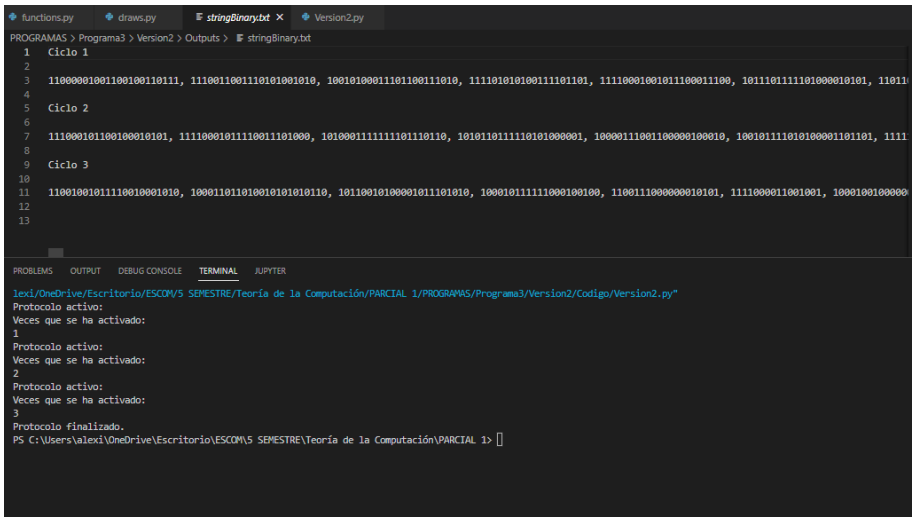
```
                estado = 1
            hist.write("(q2,\""+char+"\",q"+str(estado)+")\n")
            continue

        if estado == 3:
            if char == "0":
                estado = 2
            if char == "1":
                estado = 0
            hist.write("(q3,\""+char+"\",q"+str(estado)+")\n")
            continue
    hist.write("\n")
    if estado == 0:
        aceptar.write(cadena+"  \n")
    else:
        rechazar.write(cadena+"  \n")
```

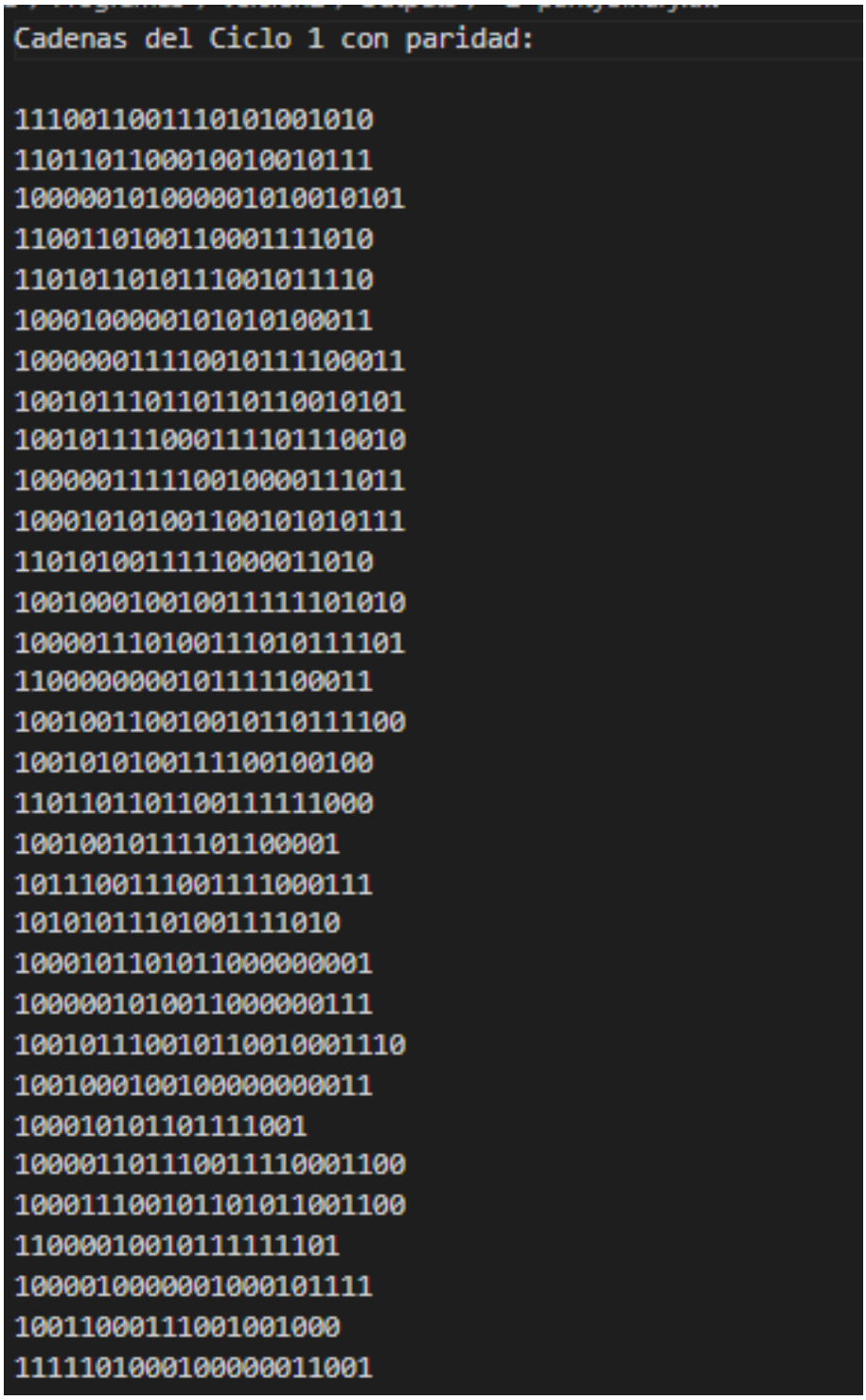
Algoritmo 2: Librería que contiene las funciones de nuestra clase principal

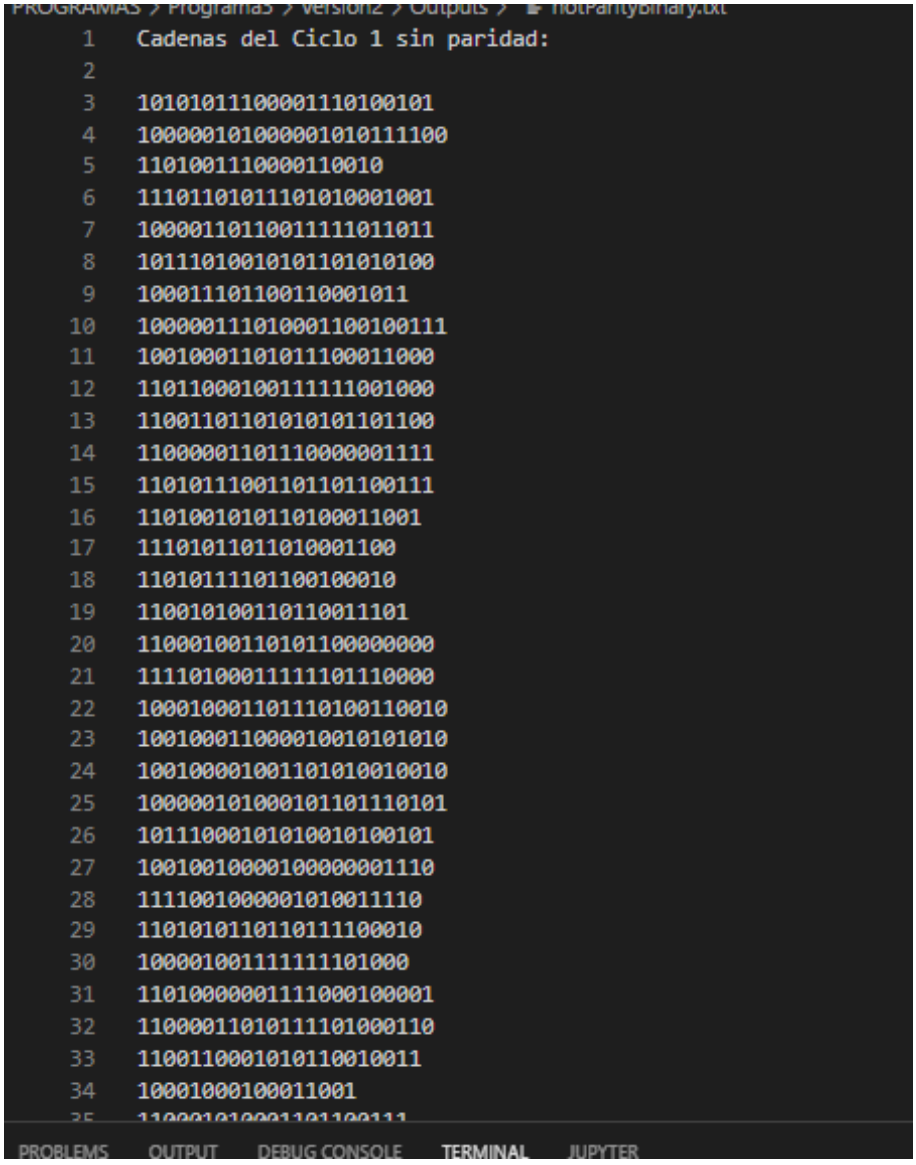
A su vez, este se conecta a otra librería, la cual se encarga de asignarle las propiedades a cada elemento, reduciendo un poco el código.

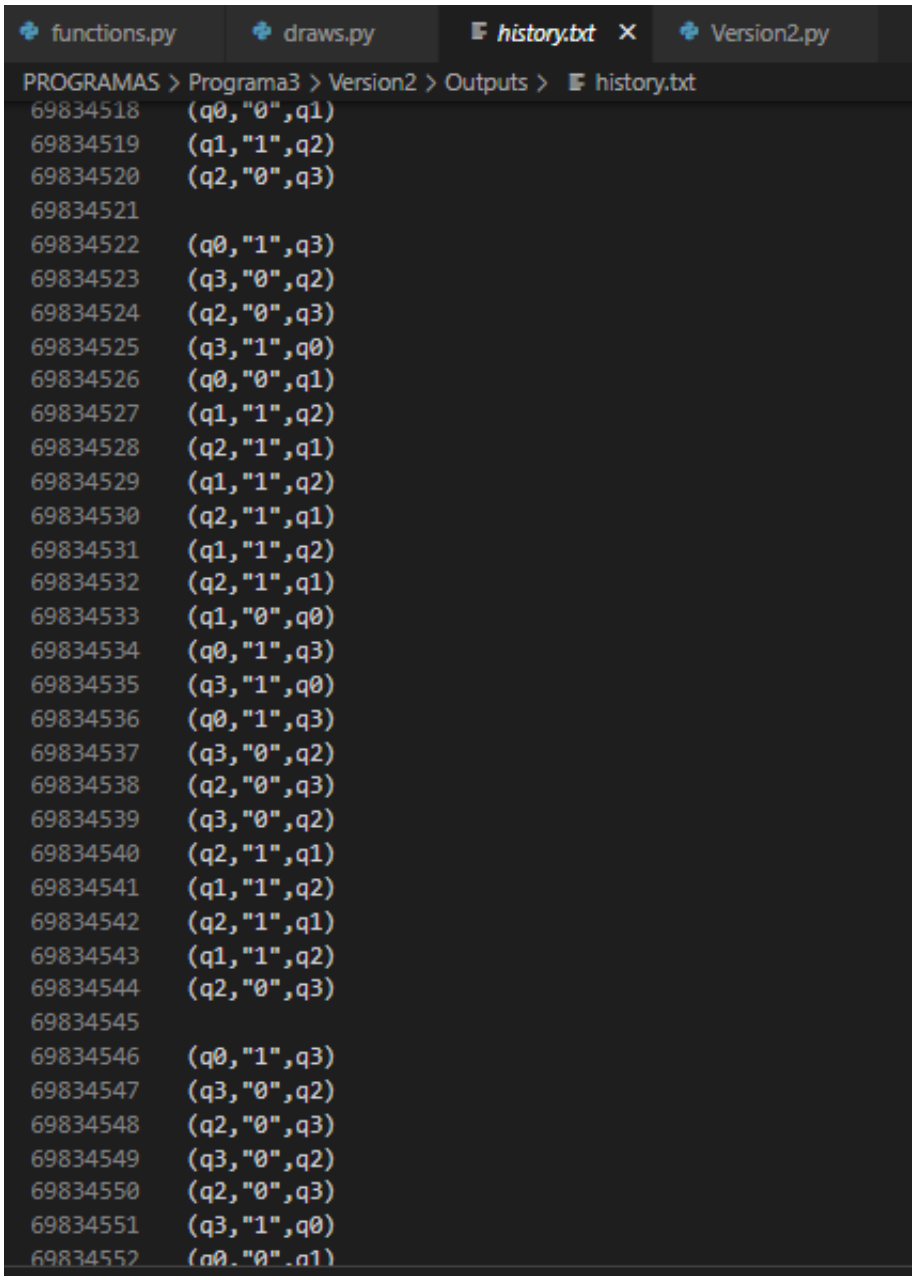
```

from graphics import *

def graphicCircle(x, y, r, color, border, width):
    head = Circle(Point(x,y), r)
    head.setFill(color)
    head.setOutline(border)
    head.setWidth(width)
    return head

def graphicText(x,y,cadena, width, color, font):
    label = Text(Point(x,y), cadena)
    label.setFill(color)
    label.setFace(font)
    label.setSize(width)
    return label

def graphicLine(w,x,y,z):
    line = Line(Point(w,x), Point(y,z))
    line.setWidth(3)
    line.setArrow("first")
    line.setFill("black")
    return line

def graphicLi(w,x,y,z):
    line = Line(Point(w,x), Point(y,z))
    line.setWidth(3)
    line.setFill("black")
    return line
```

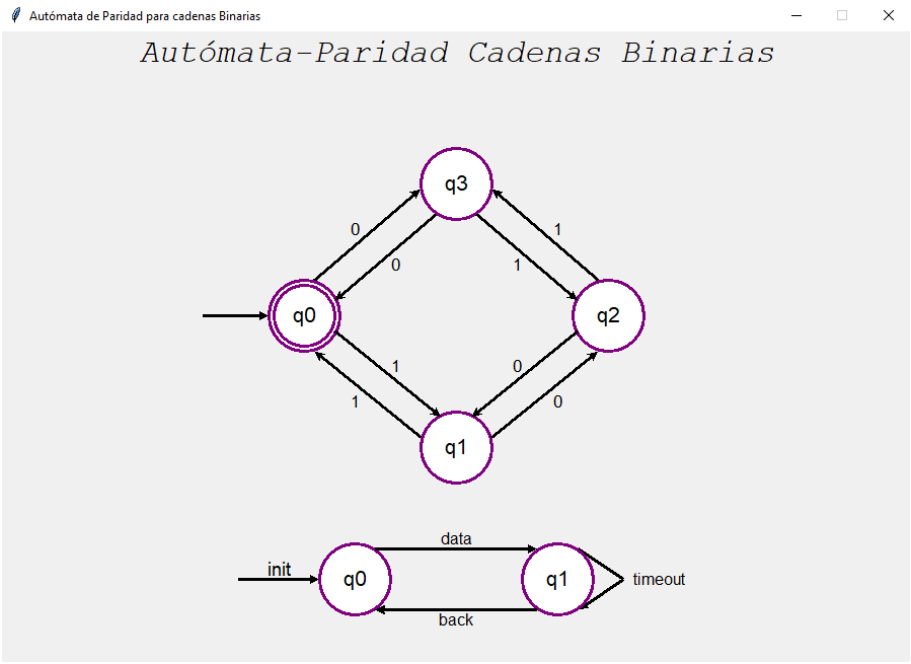
Algoritmo 3: Librería que contiene las propiedades de los elementos.


functions.py
draws.py
stringBinary.txt
Version2.py
PROGRAMAS > Programa3 > Version2 > Outputs > stringBinary.txt
1 Ciclo 1
3 1100008100110010011011, 1110011001110101001010, 1001010001110110011010, 1111010101001111101101, 1111000100101110001100, 1011101111101000010101, 11011
5 Ciclo 2
7 1110001011001000101011, 1111000101111001111010000, 1010001111111101110110, 1010110111110101000001, 1000011100110000100010, 1001011110101000011011011, 1111
9 Ciclo 3
11 1100100101111001000101010, 1000110110100101010101110, 1011001010000101110101010, 1000101111110001001000, 1100111000000010101, 1111000011001001, 1000100100000
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
lexi/OneDrive/Escritorio/ESCOM/5 SEMESTRE/Teoría de la Computación/PARCIAL 1/PROGRAMAS/Programa3/Version2/Codigo/Version2.py"
Protocolo activo:
Veces que se ha activado:
1
Protocolo activo:
Veces que se ha activado:
2
Protocolo activo:
Veces que se ha activado:
3
Protocolo finalizado.
PS C:\Users\alexi\OneDrive\Escritorio\ESCOM\5 SEMESTRE\Teoría de la Computación\PARCIAL 1>


Cadenas del Ciclo 1 con paridad:
11100110011101010010 10
11011011000100100101 11
10000010100000101001 0101
11001101001100011110 10
11010110101110010111 10
10001000001010101000 11
10000001111100101111 00011
10010111011011001001 0101
10010111100011110111 0010
10000011111100100001 11011
10001010100110010101 0111
11010100111110000110 10
10010001001001111110 1010
10000111010011101011 1101
11000000001011111000 11
10010011001001011011 1100
10010101001111001001 00
11011011011001111110 00
10010010111101100001
10111001110011110001 11
10101011101001111010
10001011010110000000 01
10000010100110000001 11
10010111001011001000 1110
10010001001000000000 11
10001010110111100 1
10000110111001111000 11100
10001110010110101100 1100
11000010010111111110 1
10000100000010001011 11
10011000111001001000
11111010001000000110 01


PROGRAMAS > Programa3 > Version2 > Outputs > noParityBinary.txt
Cadenas del Ciclo 1 sin paridad:
10101011100001110100101
10000010100000101011100
1101001110000110010
11101101011101010001001
10000110110011111011011
10111010010101101010100
10001110110011001011
10000011101000110010011
10010001101011100011000
11011000100111111001000
11001101101010101101100
11000001101110000001111
11010111001101101100111
11010010101101000011001
11101011011010001100
11010111101100100010
11001010011011001101
11000100110101100000000
11110100011111101110000
10001000110111010011010
10010001100001001010101
10010000100110101001010
10000010100010110111010
10111000101010010100101
10010010000100000001110
11110010000010100111110
11010101101101111100010
10000100111111111101000
11010000001111000100001
11000011010111101000110
11001100010101100100011
10001000100011001
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER


functions.py
draws.py
history.txt
Version2.py
PROGRAMAS > Programa3 > Version2 > Outputs > history.txt
69834518 (q0,"0",q1)
69834519 (q1,"1",q2)
69834520 (q2,"0",q3)
69834521
69834522 (q0,"1",q3)
69834523 (q3,"0",q2)
69834524 (q2,"0",q3)
69834525 (q3,"1",q0)
69834526 (q0,"0",q1)
69834527 (q1,"1",q2)
69834528 (q2,"1",q1)
69834529 (q1,"1",q2)
69834530 (q2,"1",q1)
69834531 (q1,"1",q2)
69834532 (q2,"1",q1)
69834533 (q1,"0",q0)
69834534 (q0,"1",q3)
69834535 (q3,"1",q0)
69834536 (q0,"1",q3)
69834537 (q3,"0",q2)
69834538 (q2,"0",q3)
69834539 (q3,"0",q2)
69834540 (q2,"1",q1)
69834541 (q1,"1",q2)
69834542 (q2,"1",q1)
69834543 (q1,"1",q2)
69834544 (q2,"0",q3)
69834545
69834546 (q0,"1",q3)
69834547 (q3,"0",q2)
69834548 (q2,"0",q3)
69834549 (q3,"0",q2)
69834550 (q2,"0",q3)
69834551 (q3,"1",q0)
69834552 (q0,"0",q1)


Autómata de Paridad para cadenas Binarias
Autómata-Paridad Cadenas Binarias
q3
q0
q2
q1
0
0
1
1
1
1
0
0
init
data
back
timeout

4 Conclusión

Gracias a este programa, fuimos capaces de mejorar y comprender al completo tanto los DFA's. Me pareció un ejercicio bastante sencillo, así como una buena introducción al tema de autómatas de estados finitos. Sus aplicaciones pueden ir desde lo más básico, como este ejercicio, hasta cosas enormes como redes neuronales. Estos ejercicios nos ayudan a comprender el lenguaje y entendimiento de las computadoras.

References

[1] Como se transforma un AFN en un AFD? (s. f.). Teoría de Automatas y Lenguajes Formales. Recuperado 5 de junio de 2022, de http://repositori.uji.es/xmlui/bitstream/handle/10234/5875/bolAuto1.pdf?sequence=1