Práctica 3. "Protocolo y Autómata para cadenas Pares e Impares""

Carmona Serrano Ian Carlo
Ingeniería en Inteligencia Artificial 5BM1
Teoría de la Computación , ESCOM- IPN

14 de Octubre 2023

1 Introduction

En esta práctica, se ha desarrollado un programa que simula el funcionamiento de un protocolo utilizando un Autómata Finito Determinista (AFD). El objetivo principal del programa es generar y validar un gran conjunto de cadenas binarias aleatorias de longitud 64 mediante el uso de un AFD de paridad.

2 Marco Teórico

2.1 Autómata Finito Determinista (AFD)

Un Autómata Finito Determinista es un modelo matemático ampliamente utilizado en la teoría de autómatas y lenguajes formales. Consiste en un conjunto finito de estados, un alfabeto de entrada, una función de transición que describe cómo el autómata se mueve entre estados en respuesta a símbolos de entrada y un conjunto de estados finales que indican las condiciones para aceptar una cadena de entrada.

2.2 Protocolo

En el contexto informático, un protocolo es un conjunto de reglas y convenciones que rigen la comunicación entre dispositivos o sistemas. Establece cómo se deben formatear, transmitir y recibir los datos, garantizando que las partes involucradas se entiendan mutuamente. Los protocolos son esenciales para la interoperabilidad y la comunicación eficiente en redes y sistemas distribuidos.

2.3 Cadenas Binarias

Las cadenas binarias son secuencias de bits que consisten en ceros (0) y unos (1). Estas cadenas se utilizan ampliamente en la representación de datos y la

comunicación en sistemas digitales y de computadoras.

2.4 AFD de Paridad

Un AFD de paridad es un tipo especial de autómata finito determinista que se utiliza para determinar si una cadena binaria tiene una cantidad par o impar de unos. Este tipo de autómata es útil en la detección de errores en la transmisión de datos, ya que permite verificar la integridad de una cadena binaria.

2.5 Generación Aleatoria de Cadenas

La generación aleatoria de cadenas binarias implica la creación de secuencias de bits sin un patrón predecible. Esto se logra mediante algoritmos que seleccionan aleatoriamente valores binarios (0 o 1) para cada posición en la cadena.

2.6 Graficación de Autómatas

La representación gráfica de autómatas, como AFDs, es una técnica común para visualizar su estructura y comportamiento. Estas representaciones gráficas ayudan a comprender cómo el autómata procesa las cadenas de entrada y cómo se mueve entre sus estados.

En esta práctica, se combina la teoría de autómatas con la programación para simular un protocolo que genera y valida un gran número de cadenas binarias aleatorias utilizando un AFD de paridad. El programa se encarga de verificar el estado del protocolo, generar las cadenas, realizar la validación y almacenar los resultados en archivos de texto.

3 Desarrollo

4 Desarrollo de la Práctica

4.1 Paso 1: Generación de Cadenas Binarias y Almacenamiento

En este primer paso, el programa se encargará de generar un conjunto de un millón de cadenas binarias aleatorias, cada una con una longitud de 64 bits. Estas cadenas binarias se guardarán meticulosamente en un archivo de texto designado, con el propósito de someterlas más adelante a la clasificación a través del autómata correspondiente.

4.2 Código Generador de cadenas

```python
import random

def generar_cadena_binaria(bits):
    cadena = ''.join(random.choice('01') for _ in range(bits))
    return cadena

def generar_cadenas_pares_e_impares(cantidad):
    cadenas = []

    for _ in range(cantidad):
        bits = generar_cadena_binaria(64)
        cadenas.append(bits)
    return cadenas

def guardar_cadenas_en_archivo(cadenas, nombre_archivo):
    with open(nombre_archivo, 'w') as archivo:
        for cadena in cadenas:
            archivo.write(f'{cadena}\n')


cantidad_cadenas = int(input("Ingrese la cantidad de cadenas a generar: "))
nombre_archivo = "cadenas_binarias.txt"

cadenas_generadas = generar_cadenas_pares_e_impares(cantidad_cadenas)
guardar_cadenas_en_archivo(cadenas_generadas, nombre_archivo)

print(f"{cantidad_cadenas} cadenas generadas y guardadas en {nombre_archivo}.")
```

Listing 1: Generador de cadenas binarias

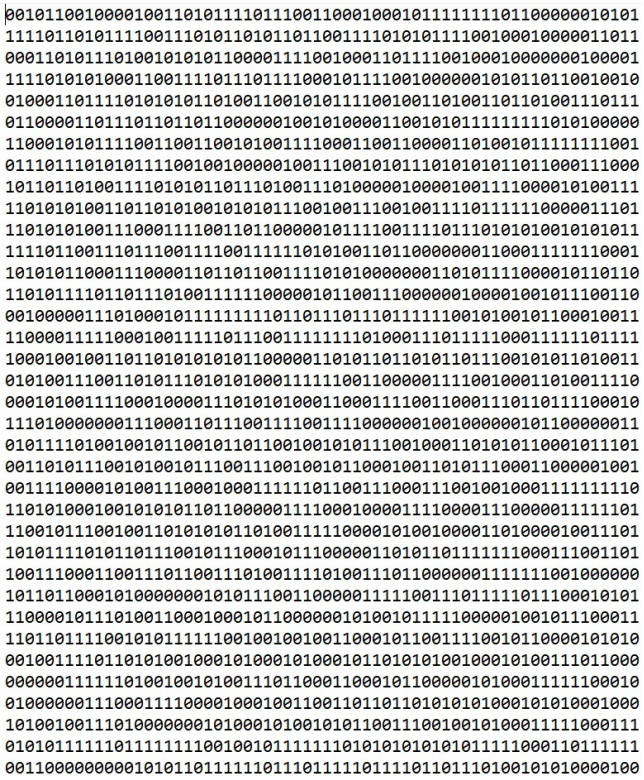

Figure 1: Paso 1

4.3 Paso 2: Inicialización del Protocolo

En este segundo paso, procedemos con la inicialización del programa. El programa se encuentra en uno de dos estados posibles: encendido (1) o apagado (0). En caso de estar en el estado de apagado, la ejecución del programa finaliza. Sin embargo, si se encuentra en el estado 1 (encendido).

```
0
¿Desea continuar?...
Se apago el automata despues de 0 ejecuciones
```

Figure 2: Paso 2 Protocolo Apagado

```
1
¿Desea continuar?...
0
¿Desea continuar?...
Se apago el automata despues de 1 ejecuciones
```

Figure 3: Paso 2 Protocolo Encendido

4.4 Paso 3: Clasificacion de cadenas

El programa continúa su ejecución para llevar a cabo la clasificación de las cadenas binarias generadas previamente. Antes de iniciar la clasificación, el programa realiza una pausa de dos segundos.

```
00101100100001001101011110111001100010001011111111101100000010101
11110110101111001110101101101100111101010111100100010000011011
11110101010001100111101110111000101111001000000101011011001010
01000110111101010101101001100101011110010011010011011010011110111
11010101001101101010010101011100100111001001111011111110000011101
11010101001110001111001101100000101111001111011101010100101010111
11110110011101110011110011111101010011011000000011000111111110001
10101011000111000011011011001111010100000001101011110000101101110
01010011100110101110101010001111110011000001111001000011010011110
00110101110010100101110011100100101100010011010111000110000010001
10101111010110111001011100010111000001101010111111100011100110101
11000010111010011000100010110000001010010111110000010010111000011
11011011110010101111110010010010011000101100111010010110000101010
00100111101101010010001010001010001011010101001000101001111011000
01000000111000111100001000100110011011011010101010001010100010000
01010111111011111111001001011111110101010101010111110001101111111
00110000000001010110111111011101111111011110110111010010101000100
11011011100010010101010011110110011100101111111111110011001110011
00100110010000010010100010010011000111101011010011011110110010111
00000110011100001010011000100111101010001010111100100101011010000
10010101000101010111100001011011111001001010011111111001001000010
11111010010011101000100110100000111000110000111111100110100111101
11100101110111001011100110110001000110011011111101011010110010111
11110010010111101001010001111001011011101100011101110111100011110
10001011001111100010001011000011010110101001101110000111110001101
10010001001011111011000101001010110100100011000110110001111000110
00110001001011011000011111110001010101110101001111100011001100010
00000111110010101010011100001011101101000110111011111100001110111
00001101010100000010010111001011101100011111111111101100011010110
11101110011100011110011101111011100001101100100001100110100010
11111101000111110101110111110000101010001011101010110001001001110
00010011110010100100001000010011111101010011100100001010100011111
```

Figure 4: Paso 3 Clasificación Par

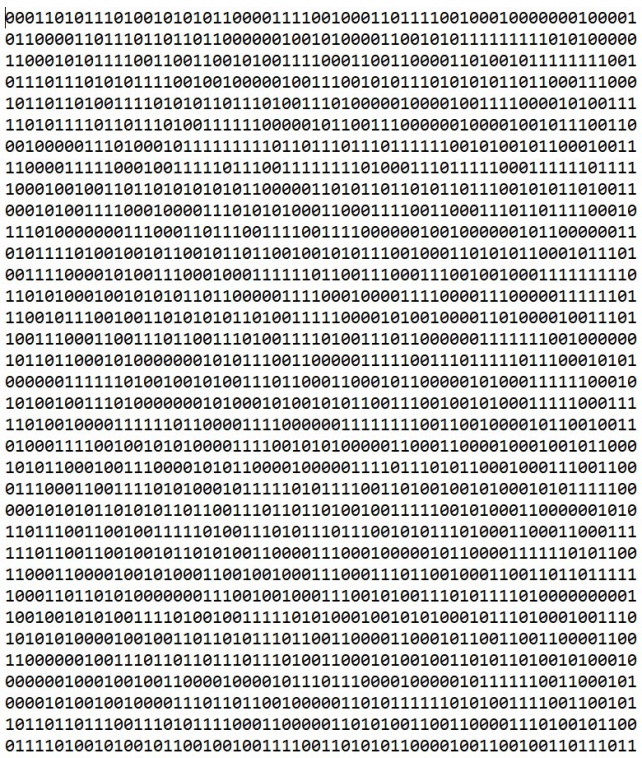

Figure 5: Paso 3 Clasificación Impar

4.5 Código Clasificador

```python
import random
import time

def guardar_cadenas_en_archivo(cadenas, nombre_archivo):
    with open(nombre_archivo, "a+") as archivo:
        archivo.write(f"{cadenas}")


def e1(numero):
    transiciones = {"0": 2, "1": 3}
    return transiciones.get(numero, True)

def e2(numero):
    transiciones = {"0": 1, "1": 4}
    return transiciones.get(numero, False)

def e3(numero):
    transiciones = {"0": 4, "1": 1}
```

```
    return transiciones.get(numero, False)

def e4(numero):
    transiciones = {"0": 3, "1": 2}
    return transiciones.get(numero, False)



def automata_clasificador(cadena, ruta_par, ruta_impar):
    flag = None
    estado = 1

    for i in cadena:
        if estado == 1:
            estado = e1(i)
            flag = estado

        elif estado == 2:
            estado = e2(i)
            flag = estado

        elif estado == 3:
            estado = e3(i)
            flag = estado

        elif estado == 4:
            estado = e4(i)
            flag = estado

    if flag == True:
        guardar_cadenas_en_archivo(cadena, ruta_par)
    else:
        guardar_cadenas_en_archivo(cadena, ruta_impar)


ruta_data = "cadenas_binarias.txt"
ruta_data_impar = "cadenas_binarias_impar.txt"
ruta_data_par = "cadenas_binarias_par.txt"
ejecuciones = 0

on_off = random.choice("10")  # Con un random ve si el
    automata esta prendido o apagado
print(on_off)
input(" Desea  continuar?...")


while on_off == "1":
    ejecuciones = ejecuciones + 1
    with open(ruta_data, "r") as archivo:
        lineas = archivo.readlines()
```

```
68
69      time.sleep(2)
70
71      for linea in lineas:
72          automata_clasificador(linea, ruta_data_par,
        ruta_data_impar)
73
74      on_off = random.choice("10")  # Con un random ve si el
        automata esta prendido o apagado
75
76      print(on_off)
77      input(" Desea  continuar?...")
78
79  print(f"Se apago el automata despues de {ejecuciones}
        ejecuciones")
```

Listing 2: Protocolo Clasificador

4.6 Paso 4: Graficación del AFD Completo (Protocolo y Paridad)

En este cuarto paso, se procede a la representación gráfica del Autómata Finito Determinista (AFD) completo, que incluye tanto el protocolo como el AFD de paridad. Esta visualización se realiza en un único gráfico que permite comprender de manera integral cómo interactúan y se combinan estos dos elementos clave en el proceso de clasificación de las cadenas binarias generadas. La representación gráfica proporciona una perspectiva visual esencial para comprender el funcionamiento conjunto del protocolo y el AFD de paridad en la práctica.

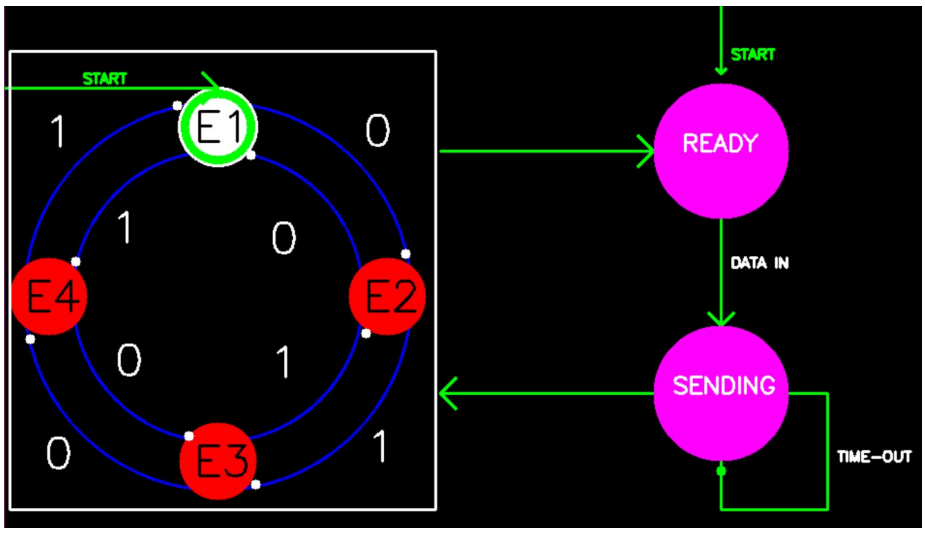

Figure 6: Paso 2 Protocolo Encendido

4.7 Código Grafica

```
import cv2
import numpy as np

# Crear una imagen en blanco
width, height = 1000, 600
image = np.zeros((height, width, 3), dtype=np.uint8)

# Definir el centro y el radio del c rculo
circle_center = (220, 300)
circle_radius = 200

# Definir la fuente y otros par metros de texto
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 1.5
font_color = (0, 0, 0)  # Color del texto en formato BGR
font_thickness = 2



# Dibujar el c rculo en la imagen
cv2.circle(image, circle_center, circle_radius, (255, 0, 0),
    thickness=2)
cv2.circle(image, circle_center, 150, (255, 0, 0), thickness
    =2)
cv2.circle(image, (220, 125), 40, (0, 255, 0), thickness=cv2
    .FILLED)
cv2.circle(image, (220, 125), 30, (255, 255, 255), thickness
    =cv2.FILLED)
cv2.circle(image, (220, 125), 40, (255, 255, 255), thickness
    =2)
cv2.circle(image, (395, 300), 40, (0, 0, 255), thickness=cv2
    .FILLED)
#cv2.circle(image, (675, 300), 30, (255, 255, 255),
    thickness=cv2.FILLED)
cv2.circle(image, (220, 470), 40, (0, 0, 255), thickness=cv2
    .FILLED)
#cv2.circle(image, (500, 470), 30, (255, 255, 255),
    thickness=cv2.FILLED)
cv2.circle(image, (45, 300), 40, (0, 0, 255), thickness=cv2.
    FILLED)
#cv2.circle(image, (325, 300), 30, (255, 255, 255),
    thickness=cv2.FILLED)
cv2.circle(image, (254, 154), 5, (255, 255, 255), thickness=
    cv2.FILLED)
cv2.circle(image, (414, 256), 5, (255, 255, 255), thickness=
    cv2.FILLED)
cv2.circle(image, (178, 103), 5, (255, 255, 255), thickness=
    cv2.FILLED)
```

```
cv2.circle(image, (73, 264), 5, (255, 255, 255), thickness=cv2.FILLED)
cv2.circle(image, (259, 494), 5, (255, 255, 255), thickness=cv2.FILLED)
cv2.circle(image, (26, 344), 5, (255, 255, 255), thickness=cv2.FILLED)
cv2.circle(image, (190, 444), 5, (255, 255, 255), thickness=cv2.FILLED)
cv2.circle(image, (373, 338), 5, (255, 255, 255), thickness=cv2.FILLED)


# Agregar el texto a la imagen
cv2.putText(image, "E1", (195, 140) , font, font_scale, font_color, font_thickness)
cv2.putText(image, "E2", (370, 315) , font, font_scale, font_color, font_thickness)
cv2.putText(image, "E3", (195, 485) , font, font_scale, font_color, font_thickness)
cv2.putText(image, "E4", (20, 315) , font, font_scale, font_color, font_thickness)

cv2.putText(image, "0", (370, 144) , font, font_scale, (255, 255, 255), font_thickness)
cv2.putText(image, "0", (273, 255) , font, font_scale, (255, 255, 255), font_thickness)
cv2.putText(image, "1", (373, 470) , font, font_scale, (255, 255, 255), font_thickness)
cv2.putText(image, "1", (273, 383) , font, font_scale, (255, 255, 255), font_thickness)
cv2.putText(image, "0", (113, 381) , font, font_scale, (255, 255, 255), font_thickness)
cv2.putText(image, "0", (40, 477) , font, font_scale, (255, 255, 255), font_thickness)
cv2.putText(image, "1", (109, 244) , font, font_scale, (255, 255, 255), font_thickness)
cv2.putText(image, "1", (40, 145) , font, font_scale, (255, 255, 255), font_thickness)
cv2.putText(image, "START", (80, 80) , font, 0.5, (0, 255, 0), font_thickness)
cv2.putText(image, "START", (750, 55) , font, 0.5, (0, 255, 0), font_thickness)



# Definir puntos para dibujar la flecha
arrow_start = (0, 85)
arrow_end = (220, 85)

# Dibujar la flecha en la imagen
```

```
cv2.arrowedLine(image, arrow_start, arrow_end, (0, 255, 0),
    thickness=2, tipLength=0.1)
cv2.arrowedLine(image,(740,150) , (740, 330), (0, 255, 0),
    thickness=2, tipLength=0.1)
cv2.arrowedLine(image,(670, 400) , (450, 400), (0, 255, 0),
    thickness=2, tipLength=0.1)
cv2.arrowedLine(image,(450, 150) , (670, 150), (0, 255, 0),
    thickness=2, tipLength=0.1)
cv2.arrowedLine(image,(740,0) , (740,70), (0, 255, 0),
    thickness=2, tipLength=0.1)


cv2.rectangle(image, (5, 47), (445, 520), (255, 255, 255),
    thickness=2)
cv2.rectangle(image, (740, 400), (850, 520), (0, 255, 0),
    thickness=2)
cv2.circle(image, (740, 480), 5, (0, 255, 0), thickness=cv2.
    FILLED)

cv2.circle(image, (740, 150), 70, (255, 0, 255), thickness=
    cv2.FILLED)
cv2.circle(image, (740, 400), 70, (255, 0, 255), thickness=
    cv2.FILLED)

cv2.putText(image, "READY", (700, 150) , font, .8, (255,
    255, 255), font_thickness)
cv2.putText(image, "SENDING", (690, 400) , font, .8, (255,
    255, 255), font_thickness)
cv2.putText(image, "DATA IN", (750, 270) , font, .5, (255,
    255, 255), font_thickness)
cv2.putText(image, "TIME-OUT", (860, 470) , font, .5, (255,
    255, 255), font_thickness)


# Mostrar la imagen resultante
cv2.imshow("C rculo con Flecha", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Listing 3: Grafica

5 Conclusión

En esta práctica, hemos desarrollado un programa que simula el funcionamiento de un protocolo utilizando un Autómata Finito Determinista (AFD). A lo largo de los pasos de desarrollo, hemos abordado varios aspectos fundamentales, desde la generación de cadenas binarias aleatorias hasta la validación de estas cadenas utilizando un AFD de paridad.

Uno de los aspectos destacados de esta práctica es la capacidad de verificar y controlar el estado del protocolo, lo que permite que el programa se ejecute de manera automática y se detenga cuando sea necesario. Además, la generación masiva de cadenas binarias y su clasificación nos ha permitido comprender el funcionamiento de los autómatas en la práctica.

6 Bibliografía

Ullman, J.D. (2009-10). "CS154: Introduction to Automata and Complexity Theory". Sitio web: http://infolab.stanford.edu/ ullman/ialc/spr10/spr10.htmlLECTURE