LENGUAJES DE COMPUTACIÓN

Tarea No.: ***Mini proyecto – Simulación de AFD***	Fecha: 05/10/21
Alumno: Codero Hernández Marco Ricardo	Id alumno: 727272
Palabras clave: Simulación, AFD, Python, Automatización	

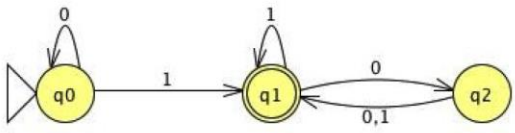
Objetivo general.

Simular un autómata finito determinista(AFD).
Entrada: archivo con la especificación de la cadena a procesar y la entrada

Cadena de entrada
Alfabeto: símbolos separados por punto y coma
Estado inicial
Estados finales
Matriz con la función de transiciones

Salida: mostrar la secuencia de estados visitados para procesar la cadena e indicar si es aceptada o no por el autómata.

M4



L(M4)={w|Conjunto de palabras que tienen al menos un 1 y un número par de 0s siguen al último 1}
Σ={0,1}

Ejemplo para el autómata M4.

Entrada:

0010100
0;1;
0
1;
0;1;
2;1;
1;1;

Salida:

Aceptada
Secuencia de estados: 0/0/0/1/2/1/2/1/

Contenido.

Descripción del algoritmo

Antes de comenzar con la explicación, cabe destacar que todo el código a excepción de las salidas de texto está escrito en inglés, esto pensando que en un futuro pudiera compartirse con entidades externas al curso.

Descrito desde el punto de vista del usuario, es decir, en orden de ejecución, el algoritmo comienza solicitando el nombre del archivo en donde se encuentran las especificaciones del autómata; automáticamente, se hacen una serie de verificaciones para validar el nombre del archivo. Si el archivo es válido, se procederá con el análisis del mismo, de lo contrario, se solicitará un nuevo archivo.

Para el análisis mencionado, se hace uso de clases del paradigma de programación orientado a objetos, esto para otorgarle una coherencia al autómata individual que cuenta con una serie de atributos específicos para ese objeto.

Primero, se "lee" el archivo para separar todas sus líneas y manejarlas individualmente, categorizándolas como se especifica en el objetivo general, de manera que es asignan múltiples variables para cada especificación. Esto se detalla a continuación.

Después, se verifica que la entrada no contenga símbolos que no estén en el alfabeto especificado, caso contrario, se detiene la ejecución del algoritmo.

Posteriormente, se define el estado inicial, los estados finales, la *lista de todos los estados*[1] y un objeto que especifica las *transiciones ligadas a los estados definidos*[2].

[1]Para obtener los estados, se manda a llamar una función que analiza la matriz con la función de transiciones, en donde se busca extraer todos los símbolos de esta misma de manera que no se repita ninguno en la matriz final, la cual se retornará como una lista ordenada de cadenas.

[2]Para obtener las transiciones ligadas con sus estados, se manda a llamar a una función que recorre simultáneamente los estados obtenidos en la función anterior y la matriz de transiciones dada, lo cual retorna un diccionario de diccionarios para las transiciones que se deben de realizar dependiendo de la entrada. El propósito de esta función es de mantener un formato accesible en el código, para contar con un objeto mapeado, no obstante, se puede omitir este formateo.

Una vez que se ha hecho todo lo descrito hasta el momento, finalmente se procede a evaluar la cadena de entrada. La función invocada para hacer esto, descompone la cadena en símbolos individuales, sobre los cuales se itera. Tanto la función como el uso de la clase creada finalizan al mostrar el estado de validez de la cadena ingresada en el autómata y la lista de estados visitados.

Ya que se ha mostrado el resultado, el usuario tiene dos opciones: ingresar otro archivo (puede ser el mismo) o detener la ejecución del algoritmo. Si el usuario desea ingresar más archivos, la ejecución comenzará nuevamente hasta que se desee, caso contrario, la iteración del algoritmo terminará definitivamente para la ejecución iniciada.

Casos de prueba iniciales

0. 0010100 (Caso inicial)

```
1  0010100
2  0;1;
3  0
4  1;
5  0;1;
6  2;1;
7  1;1;
```

```
Introduce el nombre de tu archivo (solo .txt): entrada1

Aceptada
Secuencia de estados: 0/0/0/1/2/1/2/1/
```

1. **00001**

```
00001
0;1;
0
1;
0;1;
2;1;
1;1;
```

```
Introduce el nombre de tu archivo (solo .txt): entrada1

Aceptada
Secuencia de estados: 0/0/0/0/0/1/
```

2. **0100**

```
0100
0;1;
0
1;
0;1;
2;1;
1;1;
```

```
Introduce el nombre de tu archivo (solo .txt): entrada1

Aceptada
Secuencia de estados: 0/0/1/2/1/
```

3. **00100000**

```
00100000
0;1;
0
1;
0;1;
2;1;
1;1;
```

```
Introduce el nombre de tu archivo (solo .txt): entrada1

Rechazada
Secuencia de estados: 0/0/0/1/2/1/2/1/2/
```

4. **010101010**

```
010101010
0;1;
0
1;
0;1;
2;1;
1;1;
```

```
Introduce el nombre de tu archivo (solo .txt): entrada1

Rechazada
Secuencia de estados: 0/0/1/2/1/2/1/2/1/2/
```

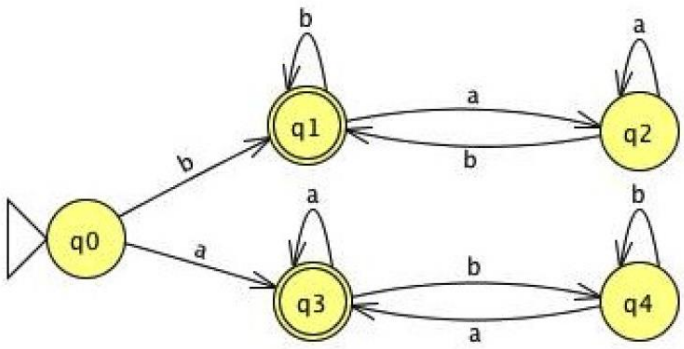
Código.

```
#IS727272 - Cordero Hernández Marco Ricardo - Simulación de AFD
class FSM:
    dump = None

    def __init__(self,fileName):
        with(open(fileName)) as specs:
            self.dump = specs.read().split("\n")

        self.string = self.dump[0] # String to evaluate (string)
        self.symbols = self.dump[1].split(";") # Allowed input symbols (list)
        if(not self.validateString()): # If character in given string not in given symbols
            return print("Error: cadena no válida con alfabeto proporcionado")

        self.initialState = self.dump[2] # Initial state (string)
        self.finalStates = self.dump[3].split(";") # Set of final state(s) (list)
        self.states = self.getStates() # Set of names for available states (list)
        self.transitions = self.getTransitions() # Set of transitions (dictionary)

        self.evaluate() # Evaluation of string

    def validateString(self):
        for elem in self.string:
            if(elem not in self.symbols):
                return False
        else:
            return True

    def getStates(self): # Returns all the states in the FSM
        # Although its specified that the states will be in order, this ensures it
        states = [self.initialState]
        holder = None

        for row in self.dump[4:]:
            holder = row.split(";")

            for elem in holder:
                if(elem not in states and elem != ""):
                    states.append(elem)
        else:
            states.sort()
            return states

    def getTransitions(self): # Handles states and transitions as dictionary
        transitions = {}

        for state, transitionsRow in zip(self.states, self.dump[4:]):
            transitions[state] = {}

            for value, transition in zip(self.symbols, transitionsRow.split(";")):
                if(transition != ""):
                    transitions[state][value] = transition
        else:
            return transitions

    def evaluate(self): # Evaluates given string
        splitString = [i for i in self.string]
        visitedStates = []
        nextState = self.initialState
```

```
        for i in splitString:
            visitedStates.append(nextState)
            nextState = self.transitions[nextState][i]
        else:
            visitedStates.append(nextState)
            print("\n" + ("Aceptada" if(nextState in self.finalStates) else "Rechazada"))
            print("Secuencia de estados: ", end="")
            print(*visitedStates, sep="/", end="")
            print("/")


def veriFile(fileName):
    mode = 0
    try:
        if(fileName[-1:-5:-1][::-1] == ".txt"):
            holder = open(fileName)
            mode = 1
        else:
            holder = open(fileName + ".txt")
            mode = 2

        holder.close()
        return mode
    except:
        return 0


while(1):
    fileName = input("Introduce el nombre de tu archivo (solo .txt): ")

    mode = veriFile(fileName)

    if(mode):
        try:
            FSM(fileName if(mode == 1) else fileName + ".txt")

            dec = input("\n¿Deseas probar con otro archivo? -> ")
            if(dec.upper() not in ("S","SI","SÍ","1","Y","YES")):
                break
        except:
            input("\nUn error inesperado ha ocurrido\n"
                  "Presiona ENTER para continuar...")
            break
    else:
        print("El nombre ingresado no es válido, intenta de nuevo...\n")
```

Para el autómata M5 probar con las cadenas:

M5



Plantilla generada

```
1  // Cadena
2  a;b
3  0
4  1;3;
5  3;1;
6  2;1;
7  2;1;
8  3;4;
9  3;4;
```

1. **aabba**

```
1  aabba
2  a;b
3  0
4  1;3;
5  3;1;
6  2;1;
7  2;1;
8  3;4;
9  3;4;
```

```
Introduce el nombre de tu archivo (solo .txt): entrada2

Aceptada
Secuencia de estados: 0/3/3/4/4/3/
```

2. **bababab**

```
1  bababab
2  a;b
3  0
4  1;3;
5  3;1;
6  2;1;
7  2;1;
8  3;4;
9  3;4;
```

```
Introduce el nombre de tu archivo (solo .txt): entrada2

Aceptada
Secuencia de estados: 0/1/2/1/2/1/2/1/
```

3. **abaaab**

```
1 abaaab
2 a;b
3 0
4 1;3;
5 3;1;
6 2;1;
7 2;1;
8 3;4;
9 3;4;
```

```
Introduce el nombre de tu archivo (solo .txt): entrada2

Rechazada
Secuencia de estados: 0/3/4/3/3/3/4/
```

4. **aababb**

```
1 aababb
2 a;b
3 0
4 1;3;
5 3;1;
6 2;1;
7 2;1;
8 3;4;
9 3;4;
```

```
Introduce el nombre de tu archivo (solo .txt): entrada2

Rechazada
Secuencia de estados: 0/3/3/4/3/4/4/
```

Conclusiones.

Después de realizar la tarea 6, es decir, aquella en donde se "programó" un autómata predefinido y se visibilizó la problemática de elaborar estas máquinas manualmente, en este proyecto se pudo minimizar el tiempo que se debe invertir para automatizar un conjunto de estados sobre los cuales se puede operar. Puede que no sea tan rápido, visual o intuitivo como una interfaz gráfica, pero, al tener el "back-end" de un software de simulación, es posible implementar la parte amigable por el usuario de manera más sencilla.
Los métodos utilizados para extraer los datos del archivo puede que no sean los mejores para autómatas de gran tamaño (aún después de minimizados), ya que, al haber varias estructuras de iteración, el tiempo de ejecución puede crecer de manera constante, sin embargo, estos temas no incumben para el curso.

Bibliografía.

Para este proyecto no se hizo uso de ninguna referencia bibliográfica.