LENGUAJES DE COMPUTACIÓN

|  |  |
| --- | --- |
| Tarea No.: Mini proyecto | Fecha: 12 de octubre de 2021 |
| Alumno: Edgar Vallejo Curti | Id alumno: IS725762 |
| Palabras clave: AFD, autómatas, programación | |

## 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.



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/

Ejemplo para el autómata M4.

## Contenido.

Algoritmo:

Describir de forma general como funciona su algoritmo, como se implemento

Para el autómata M4 mostrar la salida con impresiones de pantalla para las cadenas:

1. **00001**
2. **0100**
3. **00100000**
4. **010101010**

Para el autómata M5 probar con las cadenas:



1. **aabba**
2. **bababab**
3. **abaaab**
4. **aababb**

Para este proyecto, planté al AFD como una clase que opere con el protocolo de Iterator de Python. Esta clase contiene como parámetros los cinco elementos principales de un autómata, que son el estado principal, el conjunto de estados finales, el alfabeto, las transiciones (que son tratadas como un diccionario que contiene como llave los estados, que relacionan a otros diccionarios que tienen como llave el alfabeto, que devuelven el siguiente estado) y la palabra a procesar. Para temas de funcionamiento interno, también definí el índice, el estado actual donde se procese el autómata, el índice de la palabra, y la lista de estados por los que pasa el autómata.

En el constructor de la función, tengo la inicialización de los atributos mencionados arriba. Para seguir con el protocolo Iterator, se define el método \_\_iter\_\_, que lo que hace que cada que sea instanciado en un for, correrá por primera vez otro método llamado reset. Cada iteración del ciclo for llamará al método \_\_next\_\_, que tendrá el siguiente funcionamiento: si se pasó el índice de la palabra, levantará la excepción StopIteration. Si no, verificará si la letra a procesar si es válida en el alfabeto. Una vez pase ese filtro, hará lo siguiente:

El atributo state se asignará con states, cuya llave será el mismo estado. Este estado devolverá otro diccionario, que devolverá el nuevo estado, dependiendo de la letra a procesar. Una vez acabado, se moverá el index de la palabra, y se devolverá el estado actual.

Los últimos métodos son \_\_str\_\_, que imprime con formato si la palabra es aceptada o rechazada por el autómata, y reset, que reinicia el estado, el índice.

La implementación es la siguiente:

Nombre del archivo: AFD.py

*class* AFD():

    \_initial: int  *# The*

    \_final\_states: set

    \_alphabet: set

    \_transitions: dict

    word: str

    actual: int

    idx: int

*def* \_\_init\_\_(*self*, initial=0, final\_states=set(), alphabet=set(), transitions=dict(), word=None) -> None:

*self*.\_initial = initial

*self*.\_final\_states = final\_states

*self*.\_alphabet = alphabet

*self*.\_transitions = transitions

*self*.actual = *self*.\_initial

*self*.idx = 0

*self*.word = word

*def* \_\_str\_\_(*self*) -> str:

        if *self*.idx < len(*self*.word):

            return "The word is still processing"

        status = "Accepted" if *self*.is\_accepted() else "Rejected"

        return *f*'The word {*self*.word} is {status}'

*def* is\_accepted(*self*):

        return *self*.actual in *self*.\_final\_states

*def* reset(*self*):

*self*.actual = *self*.\_initial

*self*.idx = 0

*def* \_\_iter\_\_(*self*):

        if *self*.word is None:

            raise Exception('There\'s not word to process')

*self*.reset()

        return *self*

*def* \_\_next\_\_(*self*):

        past = *self*.actual

        if *self*.idx == len(*self*.word):

*self*.idx += 1

            return past

        if *self*.idx > len(*self*.word):

            raise StopIteration

        if *self*.word[*self*.idx] not in *self*.\_alphabet:

            raise Exception("The word is not in the alphabet")

        letter = *self*.word[*self*.idx]

        transition = *self*.\_transitions[*self*.actual]

        try:

*self*.actual = transition[letter]

*self*.idx += 1

            return past

        except:

            raise Exception('Inconsistent design')

Nombre de archivo: test.py

from AFD import AFD

*def* main():

    M4 = AFD(alphabet={'0', '1'}, final\_states={1}, transitions={

             0: {'0': 0, '1': 1}, 1: {'0': 2, '1': 1}, 2: {'0': 1, '1': 1}})

    words = ('00001', '0100', '00100000', '010101010')

    for word in words:

        states = ""

        M4.word = word

        for idx, state in enumerate(M4):

            if idx == 0:

                states += str(state)

            else:

                states += *f*"/{state}"

        print(M4, end=" on M4\n")

        print(states)

    M5 = AFD(alphabet={'a', 'b'}, final\_states={1, 3}, transitions={

        0: {'a': 3, 'b': 1}, 1: {'a': 2, 'b': 1}, 2: {'a': 2, 'b': 1}, 3: {'a': 3, 'b': 4}, 4: {'a': 3, 'b': 4}

    })

    words = ('aabba', 'bababab', 'abaaab', 'aababb')

    for word in words:

        states = ""

        M5.word = word

        for idx, state in enumerate(M5):

            if idx == 0:

                states += str(state)

            else:

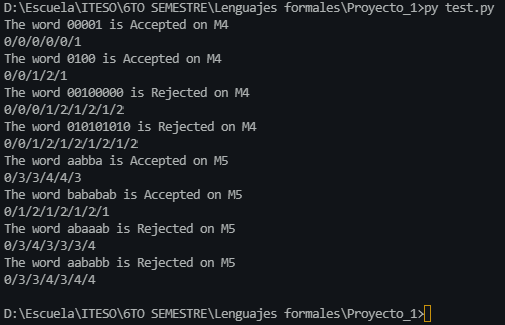
                states += *f*"/{state}"

        print(M5, end=" on M5\n")

        print(states)

if \_\_name\_\_ == '\_\_main\_\_':

    main()



## Conclusiones.

Entre la tarea pasada y este proyecto aprendí a implementar el protocolo Iterator en Python, y consideré que es la mejor manera para hacerlo funcional y estético, a diferencia de la implementación de la tarea pasada, que era algo parecido a una implementación más genérica de otros lenguajes orientado a objetos. De esta manera, podemos interactuar con el autómata de una manera más intuitiva, recibiendo el estado actual en cada iteración, y manejando cada estado por separado, separando las labores de las clases. Claro, le faltaría declarar las excepciones específicas del autómata, como “AlphabetException” o “InconsistentDesignException”, e implementar los comodities que requiere Python, pero creo que este es un buen inicio.

## Bibliografía.

Para las tareas en las que consultan referencias externas a las notas de la clase, inclúyanlas en el documento (libros, notas, páginas web, videos, etc.)