INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

AUTÓMATAS
AFND -> AFD

Osmar Pérez Bautista

INTRODUCCIÓN

Algoritmo AFND -> AFD

OBJETIVO : La construcción de subconjuntos de un AFD, a partir de un AFN.

ENTRADA : Un AFN N.

SALIDA : Un AFD D que acepta el mismo lenguaje que N.

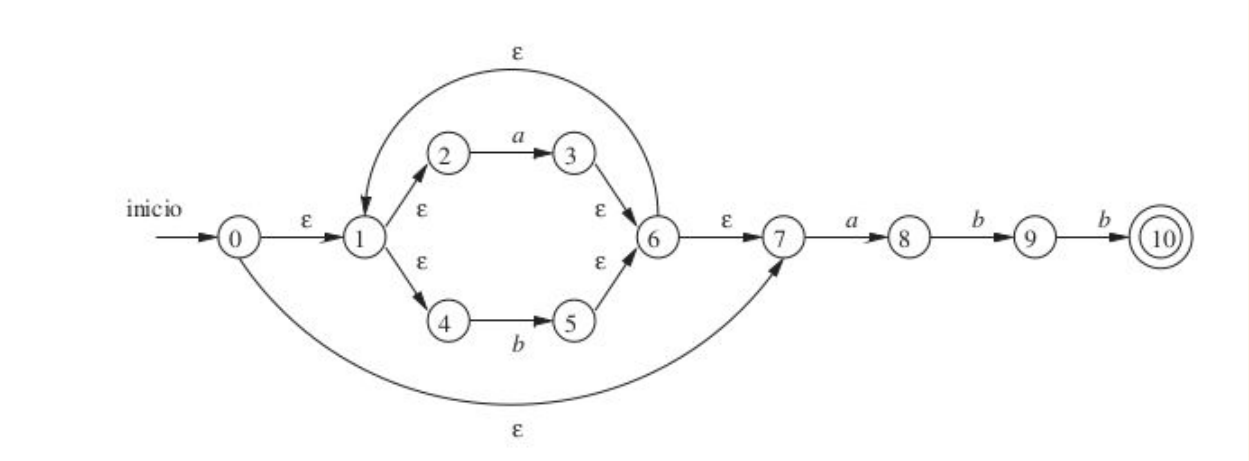
Funciones necesarias en el algoritmo:

OPERACIÓN	DESCRIPCIÓN
ϵ-*cerradura*(s)	Conjunto de estados del AFN a los que se puede llegar desde el estado s del AFN, sólo en las transiciones ϵ.
ϵ-*cerradura*(T)	Conjunto de estados del AFN a los que se puede llegar desde cierto estado s del AFN en el conjunto T, sólo en las transiciones ϵ; $= \cup_{s \text{ en } T}$ ϵ-*cerradura*(s).
mover(T, a)	Conjunto de estados del AFN para los cuales hay una transición sobre el símbolo de entrada a, a partir de cierto estado s en T.

Mi estrategia para algoritmos ya diseñados consiste en planear un caso base ya hecho a mano, donde sea fácil detectar posibles errores para comparar las entradas y salidas de cada funcion.

1.- Procedemos a tener un caso base a|Eb

AFND = {(0, 'E'): [1, 7], (1, 'E'): [2, 4], (2, 'a'): [3], (4, 'b'): [5], (5, 'E'): [6], (3, 'E'): [6], (6, 'E'): [7, 1], (7, 'a'): [8], (8, 'b'): [9], (9, 'b'): [10]}



1.1 .- Pseudo codigo de la función mover

Entrada: Símbolo, Estado

Salida: Lista de estados a los que se puede llegar desde Estado consumiendo un Símbolo

Consideraciones:

1. Dado un estado inicial retornamos todos los estados a los que llega con un símbolo.
2. Si el símbolo es ‘E’ debe retornar su estado actual + los estados a los que puede llegar

Pseudocódigo:

```
mover(estado,símbolo):
    si( (estado,símbolo) no es una transición):
        si(símbolo = ‘E’):
            return [estado]
    return False
    alcanzados = afnd.funcion_transicion[(estado,simbolo)]
    if(estado == afnd.estado_inicial):
        alcanzados.append(estado_inicial)
    return alcanzados
```

Implementación

```
def movereps(self, estado: int):

       """Retorna los elementos alcanzados por un simbolo empezando por un
estado si existe la transicion"""

       res = []

       if (estado, 'E') in self.funcion_transicion.keys():

           res = self.funcion_transicion[(estado, 'E')]

       res.append(estado)

       return res
```

1.2.- Pseudocodigo de la función epsilon-cerradura(estado)

Entrada: Lista de estados

Salida: Un conjunto de estados a los que se puede llegar usando una cadena vacia.

Consideraciones:

1. Se deben evaluar todos los estados que no se han visitado por lo que es una búsqueda en el grafo.
2. Pienso en una función recursiva que retorne la pila de estados alcanzados por la función mover()

Pseudocódigo:

```
epsilon-cerradura(estados, visitados, pila):
    for estado in estados:
        if(estado no esta en visitados): #Si no esta visitado
            visitados.append(estado)        #Lo agregamos a visitados
            alcanzados = mover(estado,'E')          #Vemos a donde llega
            for estado_alcanzado in alcanzados:  #[ ] por elmento en alcanzados
                if(estado_alcanzado not in pila) #Si estado_alcanzado aun no esta
                    pila.append(estado_alcanzado)          #Lo agregamos
                epsilon-cerradura(alcanzados,visitados,pila) #Recorremos
recursivamente los estados que hemos alcanzado
```

```
def epsilon_cerradura(self, estados: list, visitados, pila):
    for estado in estados:
        if estado not in visitados:
            visitados.append(estado)
            alcanzados = self.afnd.movereps(estado)
            for element in alcanzados:
                if(element not in pila):
                    pila.append(element)
            self.epsilon_cerradura(alcanzados, visitados, pila)
    return sorted(pila)
```

1.3.- Pseudocodigo tabla de transiciones

Entrada: Vacío

Salida: Tabla de transiciones diferentes evaluando por cada elemento en el alfabeto del autómata

Consideraciones:

1. Se evaluará cada transición nueva por lo que si es diferente se tomará de manera recursiva.
2. El conjunto inicial es el conjunto A que es la aplicación de epsilon-cerradura(0)
3. Por cada elemento en el conjunto A juntar en una pila los estados a los que llegue con cada una de las transiciones en el alfabeto del autómata
4. Y se aplicara epsilon-cerradura del nuevo conjunto.

Implementación

```
   def dtrans(self, conjunto=[] ,visitados=[]):

       if(conjunto==[]):


conjunto=self.epsilon_cerradura([self.afnd.estado_inicial],[],[])

       if conjunto not in visitados:

           print("Conjunto",conjunto)

           self.conjuntos.append(conjunto)

           visitados.append(conjunto)

           for simbolo in estados_ocupados:

               pila = []

               for estado in conjunto:

                   if(estado,simbolo) in
self.afnd.funcion_transicion.keys(): #Equivalente de mover :D por que tuve
problemas y varios bugs con la pila del programa.


pila.append(self.afnd.funcion_transicion[estado,simbolo])

               flatten = list(chain.from_iterable(pila))

               nuevo_estado = self.epsilon_cerradura(flatten,[],[])

               if(nuevo_estado not in visitados):

                   self.dtrans(nuevo_estado)


self.afd.agregar_transicion(self.conjuntos.index(conjunto),self.conjuntos.
index(nuevo_estado),simbolo)
```

Notas: Termine la practica 2, no me había dado tiempo por que no tuve luz desde el viernes de la semana antepasada hasta el martes de la semana pasada

Tiempo de realización 2:30 horas por un bug en los objetos.