INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

TEORÍA COMPUTACIONAL

PRÁCTICA 07 – AFN-E a AFD

ALUMNO: HERNÁNDEZ CASTELLANOS CÉSAR URIEL

PROFESOR: ROSAS TRIGUEROS JORGE LUIS

FECHA DE REALIZACIÓN: 04 DE OCTUBRE DEL 2017

FECHA DE ENTREGA: 11 DE OCTUBRE DEL 2017

Marco teórico.

¿Qué es un automáta?

Los autómatas son modelos de computadoras y su creador Alan Turing padre de la
computación, desde hace años (30) estudió una maquina abstracta que poseía la misma
capacidad de las computadoras actuales. Su objetivo era determinar la frontera entre lo que
puede y no puede hacer una computadora, y aun cuando sus estudios están basados en
estas maquinas abstractas son aplicables hoy en día a nuestros Pcs.

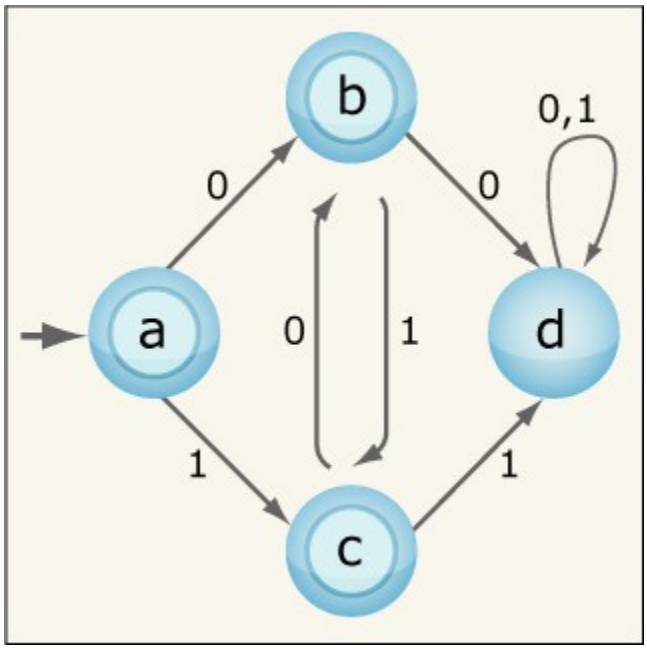

Imagen 1.0 Autómata.

Autómata finito determinista.

Es el autómata finito que tiene todas sus transiciones no vacías y que por cada símbolo desde un estado de origen se llega a un único estado destino.

Los AFD son definiciones ideales dentro de los lenguajes regulares por su cercanía formal hacia la creación de máquinas de reconocimiento fundamentalmente lexicográficas, en tanto sus transiciones son únicas por símbolo, pudiendo a la hora de su implementación en software, matemática y física realizarse con mayor facilidad.

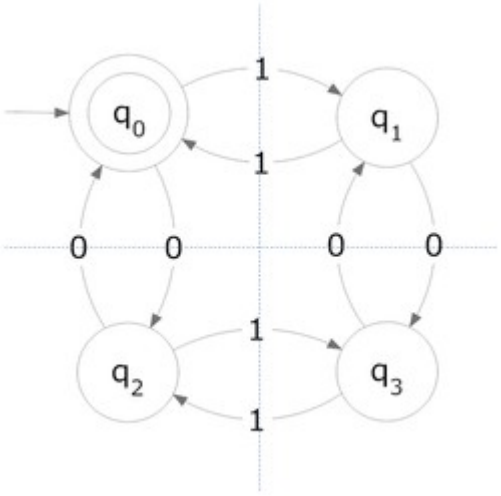

Imagen 1.1 – Autómata finito determinista.

Autómata finito no determinista.

Un autómata finito no determinista (abreviado AFND) es un automáta finito que, a diferencia de los automatas finitos deterministas (AFD), posee al menos un estado $q \in Q$, tal que para un símbolo $a \in \Sigma$ del alfabeto, existe más de una transición $\delta(q,a)$ posible.

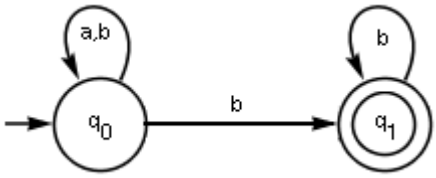

Imagen 1.2 – Autómata finito no determinista.

Algoritmo de conversión de un AFN a un AFD

1. Se calcula la C_ ε del estado inicial del AFN, el resultado será el estado inicial S 0 del AFD. • S 0 será el estado inicial del AFD y el primer Si del AFD.

2. Se calcula para cada Si la operación Ir_A para cada α ϵ Σ, la cual arrojara un estado Si (Pudiendo repetirse).

3. Se realiza la operación 2 con todos los estados hasta que ya no surjan estados diferentes.

El estado inicial del AFD será S0 y los estados finales serán todos aquellos Si que contengan al estado final del AFN original

• La función de transición es el resultado de todas las operaciones Ir_A sobre los Si.

Cerradura épsilon

• Dado un AFN definimos la operación cerradura épsilon de un estado X como:

$$C_\varepsilon(x) = \{x\} \cup \{v \mid v \text{ es alcanzable con transiciones } \varepsilon \text{ a partir de } x\}$$

$C_\varepsilon(q5) = \{q5\} \cup \{q0,q1,q3\}$
$C_\varepsilon(q2) = \{q2\} \cup \{q3,q1\}$
$C_\varepsilon(q1) = \{q1\} \cup \{\lambda\}$
$C_\varepsilon(\{q3,q4\}) = \{q3,q4,q6\}$

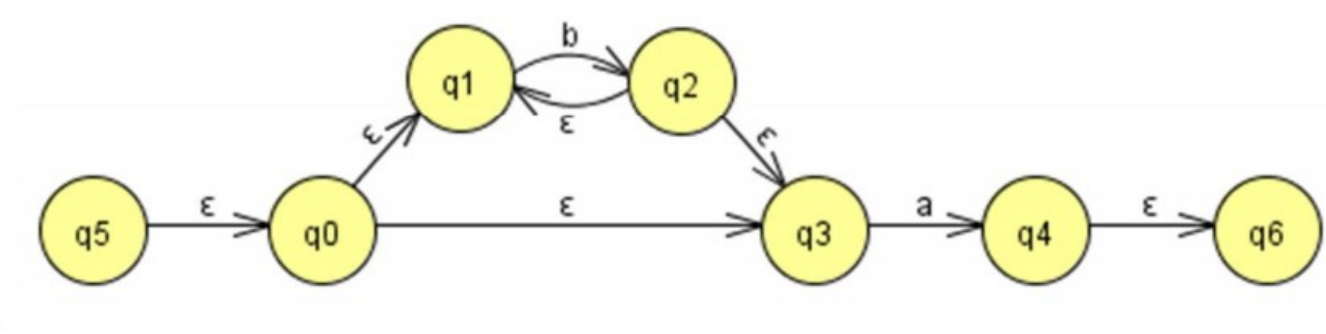

Imagen 1.3 Ejemplo

Operación Ir_A

•Ir_A (C,α) donde C es un conjunto {e1, e2, e3, en} de estados del AFN y α es un símbolo del alfabeto del mismo AFN:

Ir_A (C,α) = C_E(Mover (C ,α)) i.e.

Ir_A (C,α) =C_E(∪ Mover (e_i ,α))

Material y equipo.

El material utilizado en la práctica es el siguiente:

Herramientas de software:

* Oracle VM VirtualBox

* Ubuntu 17.04

* Python 3.6.2

* eric6 Web Browser (QtWebKit)

Herramientas de hardware:

* Computadora personal

Desarrollo de la práctica.

La práctica número siete de teoría computacional, trato en realizar un programa capaz de convertir un AFD-e a un AFN, para lo cual se siguió el siguiente algoritmo:

1. Se calcula la C_ ε del estado inicial del AFN, el resultado será el estado inicial S 0 del AFD. • S 0 será el estado inicial del AFD y el primer Si del AFD.

2. Se calcula para cada Si la operación Ir_A para cada α ϵ Σ, la cual arrojara un estado Si (Pudiendo repetirse).

3. Se realiza la operación 2 con todos los estados hasta que ya no surjan estados diferentes.

El estado inicial del AFD será S0 y los estados finales serán todos aquellos Si que contengan al estado final del AFN original

• La función de transición es el resultado de todas las operaciones Ir_A sobre los Si.

Al inicio se realizó la cerradura epsilon para un único estado, lo cual sirvió como base para realizar la siguiente función que obtiene la cerradura epsilon para mas de una transición, por lo que se tuvo

que hacer uso de la función anterior, a mi parecer fue el mayor problema que se presentó en la práctica.
Diagramas, gráficas y pantallas.

```
# Definición de nuestro automata donde ">" indica el estado inicial, y 'e' las transiciones epsilon.
Atma = AutomataFinitoN(                 [ [ [1], ['a',[1]] , ['b',[1,2]],'>']
                                        , [ [2], ['b',[1] ] ,['a',[3] ]],
                                         [ [3], ['a',[3]] ,['b',[3] ]],
                                          [ [4], ['b',[4]] ,['a',[4] ],'e'],
                                          [ [5], ['a',[5]] ,['b',[4] ],'e']
                                         ] )
```

Imagen 1.4 – Autómata inicial.

```
uriel@Uriel-PC:~/Escritorio$ python prac.py
[**********]  [[1], ['a', [1]], ['b', [1, 2]], '>']
[**********]  [[1, 2], ['a', [1, 3]], ['b', [1, 2]]]
[**********]  [[1, 3], ['a', [1, 3]], ['b', [1, 2, 3]]]
[**********]  [[1, 2, 3], ['a', [1, 3]], ['b', [1, 2, 3]]]
uriel@Uriel-PC:~/Escritorio$
```

Imagen 1.5 – Resultado.

Código

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

class AutomataFinitoN:


    def obtenerCerraduraEpsilon(estado, DELTA):
        miLista=[]
        miLista.append(estado)
        for i in DELTA:
            if (estado == i[0] and i[1]=='e'):
                cadena = str(DELTA.get(i))[2:-2]
                miLista.append(cadena)
                obtenerCerraduraEpsilon(cadena,DELTA)
        return miLista

#def Ecerradura(q,d):
#    aux = []
#    q = list(q)
#    d = dict(d)
#    i =0
#    for x in q:
#        aux.append(q[i])
#        P = []
#        P.append(x)
#        P.append('e')
#        Ptuple = tuple(P)
#        i+=1

#        if d.has_key(Ptuple):
#            aux.append(d[Ptuple][0])

#    return aux
```

```
def convertiraAutomataFinitoDeterminista(argumento):
    argumento.AFD =[]
    estadosTemporales= argumento.obtenerEstadoInicialPrima()
    while len(estadosTemporales) <> 0:
        argumento.AFD.append( estadosTemporales[0])
        for estado in argumento.obtEstado_():
            obtenerTransicionDelAFD=[]
            for mysimbol in argumento.mysimbolsDelLenguaje:
                nuevoEstado=[]
                for e in estado:
                    for t in argumento.obtenerTransicion([e],mysimbol):
                        if t not in nuevoEstado:
                            nuevoEstado.append(t)
                    nuevoEstado.sort()
                obtenerTransicionDelAFD.append([mysimbol,nuevoEstado])
            estadosTemporales.append([estado]+obtenerTransicionDelAFD)
        estadosTemporales.pop(0)
    estasdosDeAceptacion = [l[0][0] for l in filter(lambda s:len(s)>=4 and s[-1]=='*',argumento.AFN)]
    for e in argumento.AFD:
        Aceptacion = False
        for ea in estasdosDeAceptacion :
                    if not Aceptacion:
                        Aceptacion =  ea in e[0]
        if Aceptacion and '*' not in e: e.append('*')
```

```
        if Aceptacion and '*' not in e: e.append('*')
      return argumento.AFD

  def __init__(argumento,AFN):
      argumento.AFN = AFN
      argumento.mysimbolsDelLenguaje =  argumento.getLenguajeDeAFN()
      argumento.AFD = argumento.convertiraAutomataFinitoDeterminista()
  def obtenerTransicion(argumento,e,t):
      return filter(lambda s:s[0]==t,filter(lambda s:s[0]==e,argumento.AFN)[0])[0][1]
  def obtenerEstadoInicialPrima(argumento):
      return filter(lambda s:len(s)>=4 and s[3]=='>',argumento.AFN) # Funci?n encargada de obtener el nuevo estado inicial, es decir el estado inicial prima.
  def getLenguajeDeAFN(argumento):
      return [l[0] for l in  filter(lambda s:len(s)==2,argumento.AFN[-1])];  # Esta funci?n es opcional, ya que se podria definir el lenguaje como usualmente se realiza en clase o bien, definirlo cuando se
  def obtEstado(argumento):
      return [e[0] for e in  argumento.AFD] # Nuevos estados.
  def obtEstado_(argumento):
      return map(lambda e:e[1],filter(lambda s:s[0] in argumento.mysimbolsDelLenguaje and s[1] not in argumento.obtEstado(),argumento.AFD[-1]))
# Tabla de transici?n donde "->" indica el estado inicial de nuestro automata.

# Definición de nuestro automata donde ">" indica el estado inicial, y 'e' las transiciones epsilon.
Atma = AutomataFinitoN(              [ [ [1], ['a',[1]] , ['b',[1,2]],'>']
                                   , [ [2], ['b',[1]] ],['a',[3] ]],
                                    [ [3], ['a',[3]] ,['b',[3] ]],
                                     [ [4], ['b',[4]] ],['a',[4] ],'e'],
                                     [ [5], ['a',[5]] ],['b',[4] ],'e']
                                    ] )
for m in  Atma.AFD:
   print ("[**********]  \033[92m"+str(m))
```

Conclusiones y recomendaciones.

En la presente práctica se realizó el proceso inverso al anterior práctica, por lo que resultó interesante el tener como entrada la salida de del anterior programa, para verificar el correcto funcionamiento, lo que se pudo observar fue que en ciertos casos muy particulares el programa fallaba, pero en la mayoría de los casos arrojaba lo deseado.

En cuanto a las dificultades de la práctica, fue el construir la función que obtiene la delta prima, ya que debemos asociar con diferentes estados como (2,3) o (1,2,3), lo que conllevo a una cierta complicación de la práctica.

De igual forma la función que obtiene la cerradura epsilon, para más de un transición con la cadena vacía causó un poco de problema.

Referencias

[0] E. Martinez, "Web personal de Edgardo Adrián Franco Martínez", *Eafranco.com*, 2017. [Online]. Available: http://www.eafranco.com/. [Accessed: 12- Oct- 2017].

[1]D. Kelley, Automata and formal languages. Upper Saddle River, NJ: Prentice Hall, 2002.

[2]"Autómata finito determinista", Es.wikipedia.org, 2017. [Online]. Available: https://es.wikipedia.org/wiki/Aut%C3%B3mata_finito_determinista. [Accessed: 11- Oct- 2017].

[3] Autómatas finitos deterministas, Matesfacil.com, 2017. [Online]. Available: https://www.matesfacil.com/automatas-lenguajes/automata-finito-y-su-lenguaje.html. [Accessed: 11- Oct - 2017].

[4]A. determinista, "AFD en Python - Automata finito determinista", Pythondiario.com, 2017. [Online]. Available: http://www.pythondiario.com/2015/06/afd-en-python-automata-finito.html. [Accessed: 11- Oct - 2017].

[5]"Alan Turing", Es.wikipedia.org, 2017. [Online]. Available: https://es.wikipedia.org/wiki/Alan_Turing. [Accessed: 11- Oct - 2017]