INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

TEORÍA COMPUTACIONAL

PRÁCTICA 06 – AFN a AFD

ALUMNO: HERNÁNDEZ CASTELLANOS CÉSAR URIEL

PROFESOR: ROSAS TRIGUEROS JORGE LUIS

FECHA DE REALIZACIÓN: 27 DE SEPTIEMBRE DEL 2017

FECHA DE ENTREGA: 07 DE SEPTIEMBRE DEL 2017

¿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 Alan Turing

Autómatas finitos deterministas (AFD)

Un autómata finito determinista (AFD) es una quíntupla

$$M = (\Sigma, Q, \delta, q_0, F)$$

donde

- Σ es un alfabeto (sabemos $\epsilon \notin \Sigma$)
- Q es un conjunto finito no vacío de estados, es decir, $0 < |Q| < \infty$.
- δ es una *función* de transición:

$$\delta : Q \times \Sigma \longrightarrow Q \; ; \; \delta(q, \sigma) = p$$

R.

- $q_0 \in Q$ es el estado inicial.
- $F \subset Q$ es el conjunto de estados finales.

Podemos pensar de un autómata como un dispositivo que lee desde una cinta con símbolos y que realiza cambios de estados internamente:

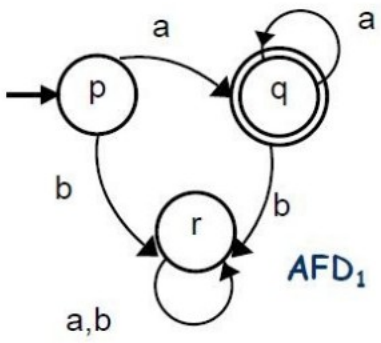

Imagen 1.1 Autómata finito determinista.

Autómata finito no determinista.

Es el autómata finito que tiene transiciones vacías o que por cada símbolo desde un estado de origen se llega a más de un estado destino.

Los AFND son definiciones no tan deseables dentro de los lenguajes regulares porque dificultan su implementación tanto mecánica como informática; aunque en la mayoría de las transformaciones a lo interno de los LR (expresiones regulares a AF, gramáticas regulares a AF) conducen a AFND. Los AFND, por tanto, son imprescindibles en el análisis lexicográfico y el diseño de los lenguajes de programación.

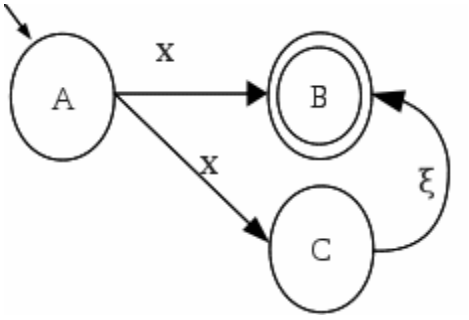

Imagen 1.2 Autómata finito no determinista.

Algoritmo para la conversión de un AFD a un AFN.

Sea un autómata finito estrictamente no determinista (AFND) definido por la 5-tupla A=<Q, T, g, F, q0>, donde Q es el conjunto de estados, T el alfabeto de símbolos terminales, la relación de transiciones Definicion relacion transiciones afnd basica.gif ó Definicion relacion transiciones afnd extendida.gif (léase: del estado qi mediante el terminal x se va a qj), F son los estados finales o de llegada dentro de Q, q0 es el estado inicial o de partida

1. *A* se transforma en $A_{AFD}=<Q_A,T,g_A,F_A,q_{0A}>'$, tal que:
    1. $V_A=P(V)-\{\{\}\}$, con $P(V)$ que es el conjunto potencia de los vértices de *A*.
    2. $F_A=\{x \mid \exists_{f\in F}[f\in x]\}$.
    3. $g_A=\{<r,x,q> \mid \forall_{r\in V_A}\forall_{p\in r}\forall_{x\in T}[\exists_{q\in V}[g(p,x)=q]\Rightarrow\langle\langle r,x,\cup g(p,x)\rangle\in g_A]\}$.
    4. $q_{0A}=\{q_0\}$.
2. Luego se eliminan de A_{AFD} todos los estados y sus correspondientes transiciones inalcanzables desde el estado inicial q_{0A}.

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 numero seis de teoría computacional consistió en implementar un algoritmo que fuera capaz de convertir un AFD a un AFN, al inicio de la sesión de laboratorio el profesor nos mostró dos herramientas las cuales sirven para simular el comportamiento de un autómata y para dibujarlos únicamente, posteriormente el profesor empezó con el algoritmo para convertir un AFD a un AFN, únicamente nos proporcionó la función que obtiene el conjunto potencia de nuestros estados, lo demás se nos quedó de tarea.

Diagramas, gráficas y pantallas.

```
Conjunto potencia

S' = P(S) = [(), ('q0',), ('q1',), ('q0', 'q1')]
```

Imagen 1.3 – Se obtiene el conjunto potencia.

```
¿Qué conjuntos contiene, por lo menos, un estado de F?
==============================Conjunto 1==============================
Vacio
==============================Conjunto 2==============================
q0 No es un estado de aceptación.
==============================Conjunto 3==============================
q1 Es un estado de aceptación, por lo que el conj 3 es parte de F''
==============================Conjunto 4==============================
q0 No es un estado de aceptación.
q1 Es un estado de aceptación, por lo que el conj 4 es parte de F''


Colección de subconjuntos de S que contienen, por lo menos, un estado de F


F' = [('q1',), ('q0', 'q1')]
```

Imagen 1.4 – Conjuntos que contienen al menos un estado de aceptación.

```
F' = [('q1',), ('q0', 'q1')]
```

Imagen 1.5 - Estados de aceptación(F’)

```
Al estado vacio se le dibujan tantos arcos que salen e inciden en el estado, como simbolos del alfabeto haya-

------ a ------
(q0,a)->['q0']
(q1,a)->['q1']

------ b ------
(q0,b)->['q1']
(q1,b)->None

{('q1', 'b'): 'E', ('E', 'b'): 'E', ('E', 'a'): 'E'}
```

Imagen 1.6 – Agregar el estado vacío con sus transiciones.

```
{'q1': 2, 'q0': 1}
{('q0', 'b'): 'E',('q1', 'b'): 'E',('q0', 'a'): 'q0',('q1', 'a'): 'q1'}
```

Imagen 1.7 – Delta estrella.

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

from itertools import chain,combinations
from time import sleep
from colorama import Cursor, init, Fore

def powerset(iterable):
        s=list(iterable)
        return chain.from_iterable(combinations(s,r) for r in range(len(s)+1))
```

Imagen 1.8 - Función que obtiene el conjunto potencia.

```python
def agregarTransicAVacio(Sigma, Q, DELTA):
    Dprima = {}

    for r in range(len(Sigma)):
        Dprima[('E', Sigma[r])] = 'E'
        print("\n------ "+Sigma[r]+" ------")
        for i in range(len(Q)):
            val = DELTA.get((Q[i], Sigma[r]))
            if(val is not None):
                print("("+Q[i]+","+Sigma[r]+")" +"->"+str(val))
            else:
                Dprima[(Q[i], Sigma[r])] = 'E'
                print("("+Q[i]+","+Sigma[r]+")"+ "->"+str(val))
    return Dprima
```

Imagen 1.9 – Agregar transiciones al estado vacio.

```python
def obtDeltaEstrella(Dprima, Q, Fprima, Sigma, DELTA):
    print("\n"+str(Dprima)+"\n")
    miDiccionario = {}
    for j in range(len(Q)):
        miDiccionario[str(Q[j])]=0

    for j in DELTA:
        llave = DELTA[j][0]
        miDiccionario[llave] = miDiccionario[llave] + 1


    for t in range(len(Q)):
        if(miDiccionario.has_key(Q[t])):
            if((miDiccionario[Q[t]]==len(Q)) and 1==1):
                Dprima[(Q[t], Sigma[t])] = Fprima[len(Q)-1]

    print(miDiccionario)
```

Imagen 2.0 – Función encargada de obtener delta estrella.

```
def probadorDeAplicacion():
    F=['q1']
    Q=['q0','q1']
    s='q0'
    DELTA={ ('q0','a'):['q0'],
            ('q0','b'):['q1'],
            ('q1','a'):['q1'] }
    Sigma=['a','b']
    Qprima=list(powerset(Q))
    print("\033[1;33m"+"Conjunto potencia"+'\033[0;m')
    print("\n")
    print("S' = P(S) = "+str(Qprima))
    print("\n")
    Fprima= obtenerConjuntoDeEstadosF(Qprima)
    print("\n")
    print("\033[1;33m"+"Colección de subconjuntos de S que contienen, por lo menos, un estado de F"+'\033[0;m')
    print("\n")
    print ("F' = "+str(Fprima))
    print("\n")
    print("\033[1;33mAl estado vacio se le dibujan tantos arcos que salen e inciden en el estado, como simbolos del alfabeto haya-")
    Dprima = agregarTransicAVacio(Sigma,Q,DELTA)
    obtDeltaEstrella(Dprima,Q,Fprima,Sigma,DELTA)



probadorDeAplicacion()
```

Imagen 2.1 – Función principal.

Conclusiones y recomendaciones.

Este algoritmo conocido como el de construcción de subconjuntos convierte un AFN en un AFD, haciendo uso de solamente tres operaciones sobre los estados de un AFN, lo interesante de todo esto es que ambos autómatas aceptarán el mismo lenguaje, es decir son equivalentes.

En cuanto a la práctica tuve bastantes dificultades, sobre todo al tratar de obtener delta estrella, de ahí en fuera lo demás era relativamente más sencillo, tal vez mi sugerencia sería el dejar algunas listas de ejercicios sobre conversión de AFD a AFN, como lo suelen hacer los profesores de ciencias básicas, siento que se comprendería de mejor manera el concepto.

También seria interesante el llevar a cabo el programa para n estados, seria un buen reto.

Referencias.

[1]"Transformación de autómata finito no determinista a autómata finito determinista - EcuRed", *Ecured.cu*, 2017. [Online]. Available: https://www.ecured.cu/Transformaci%C3%B3n_de_aut%C3%B3mata_finito_no_determinista_a_aut%C3%B3mata_finito_determinista. [Accessed: 04- Oct- 2017].

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

[3] "Autómata finito determinista", Es.wikipedia.org, 2017. [Online].Available: ://es.wikipedia.org/wiki/Aut%C3%B3mata_finito_determinista. [Accessed: 04- Oct- 2017].

[4] Autómatas finitos deterministas, Matesfacil.com, 2017. [Online]. Available: https://www.matesfacil.com/automatas-lenguajes/automata-finito-y-su-lenguaje.html. [Accessed: 04- Oct-2017].

[5] 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: 17- Sep- 2017].

[6] "Alan Turing", Es.wikipedia.org, 2017. [Online]. Available: https://es.wikipedia.org/wiki/Alan_Turing. [Accessed: 04- Oct- 2017].