INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

MATERIA: Teoría Computacional.

PRÁCTICA: Práctica 4. AFN a AFD.

ALUMNO: Meza Vargas Brandon David.

PROFESOR: Jorge Luis Rosas Trigueros.

FECHA PRÁCTICA: 27-nov-2020

FECHA DE ENTREGA: 04-dic-2020

MARCO TEÓRICO

Autómatas Finitos Deterministas (AFD)

Un AFD tiene un conjunto finito de estados y un conjunto finito de símbolos de entrada. El término “determinista” hace referencia al hecho de que para cada entrada sólo existe uno y sólo un estado al que el autómata puede hacer la transición a partir de su estado actual. Un estado se diseña para que sea el estado inicial, y cero o más estados para que sean estados de aceptación. Una función de transición determina cómo cambia el estado cada vez que se procesa un símbolo de entrada.

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, es decir, es aquel que, a diferencia de los autómatas finitos deterministas, posee al menos un estado, tal que, para un símbolo del alfabeto, existe más de una transición posible, un ejemplo de este tipo de autómata lo vemos en la figura 1.

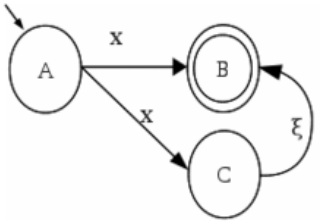


Figura 1. Ejemplo de un AFN

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.

AFND A AFD

Lo que debemos hacer es primar todos los elementos de nuestro AFN, es decir, hacer:

M'
M'=(Q',Σ',s',F',Δ') equivalente a M (L(M)=L(M'))

Veamos esto con un ejemplo:

Tenemos el siguiente AFN;

M=(Q,Σ,s,F,∆)

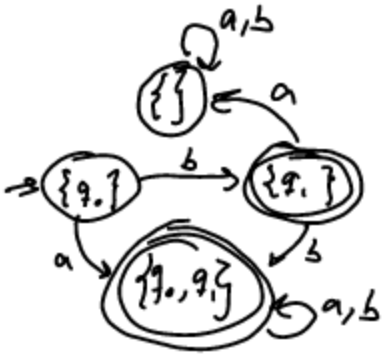
Q={q0,q1}
Σ={a,b}
s=q0
F={q1}

∆	a	b
q0	{q0,q1}	{q1}
q1	{}	{q0,q1}

Construiremos el autómata finito determinista M'
M'=(Q',Σ',s',F',∆') equivalente a M (L(M)=L(M')).

$Q'=2^Q$={ {}, {q0}, {q1}, {q0,q1} }
Σ' = Σ = {a,b}
s'={q0}
F'={ q ϵQ' | ∃ xϵq, xϵF } ={ {q1},{q0,q1} }

∆'(q',σ)= ∆(q',σ)
∆'({}, a)=∆({},a)={}
∆'({}, b)=∆({},b)={},
∆'({q0}, a)=∆({q0},a)={q0,q1}
∆'({q0}, b)=∆({q0},b)={q1}
∆'({q1}, a)=∆({q1},a)={}
∆'({q1}, b)=∆({q1},b)={q0,q1}
∆'({q0,q1}, a)=∆({q0,q1},a)={q0,q1}
∆'({q0,q1}, b)=∆({q0,q1},b)={q0,q1}



Una vez grafiquemos ∆' obtenemos nuestro AFD

MATERIAL Y EQUIPO

Para la realización de esta práctica son necesarias las siguientes herramientas:

- Un equipo de cómputo que cumpla con los requerimientos para el uso del lenguaje de programación Python.
- Tener instalado el lenguaje de programación Python.

Contar con un IDE para programar con Python, cualquiera es útil

DESARROLLO

El desarrollo de esta práctica consistió en pasar a código en el lenguaje de programación Python, la forma en la que se pasa de un AFND a un AFD.

El código debe ser probado con dos AFN, veamos el primero;

M1=({q0,q1}, {a,b}, q0, {q1}, { (q0,a,{q0,q1}), (q0,b,{q1}), (q1,b,{q0,q1}) })

Lo primero que hacemos es crear una función que nos devuelve el conjunto potencia de los estados de nuestro AFND, de esta forma obtenemos los valores de q' cuando mandamos a llamar esta función. (ver figura 2);

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

Figura 2. Función para hacer conjunto potencia de q.

Algo muy importante para que funciones la función de la figura 2, es que debemos de importar chain y combinations de la librería itertools. (ver figura 3).

```
from itertools import chain, combinations
```

Figura 3. Línea necesaria para que funcione la función de la figura 2.

Posteriormente, asignamos los valores de q, sigma, el estado inicia y final de nuestro AFND, para así poder obtener el q', s', sigma prima y F'. (ver figura 4).

```
Q=['q0','q1']
Sigma=['a','b']
s='q0'
F=['q1']

DELTA={ ('q0','a'):['q0','q1'],
        ('q1','b'):['q1'],
        ('q1','b'):['q0','q1']
        }

Qprima=list(powerset(Q))
sprima=(s,) #tupla con q0
Sigmaprima=Sigma
Fprima=[]
```

Figura 4. Símbolos de nuestro AFD y AFND.

El código que se ve en la siguiente figura (figura 5), es un ciclo for anidado con el cual asignamos los estados finales de nuestro AFD, es decir F'.

```
for q in Qprima:
    for x in q:
        if x in F:
            Fprima.append(q)
```

Figura 5. F'

Posteriormente creamos una función para hacer la conversión de AFN a AFD, la cual se explica a continuación:

Primeramente, como vemos en la figura 6, la función recibe de parámetro a Qprima, Sigmaprima, Delta, DELTAprima y una lista e;

```
def conver(Qprima,Sigmaprima, DELTA,DELTAprima, e):
```

Figura 6.

Como se ve en la figura 7, primeramente, contamos con tres fors, el primero recorre cada elemento de Qprima, el segundo los elementos del alfabeto y el tercero va a ir recorriendo de nuevo los elementos que tenemos en Qprima:

```
for x in Qprima:
    for y in Sigmaprima:
        for z in x:
```

Figura 7. Primeros fors de la función.

Dentro del tercer for, establecemos una condición, diciendo que si nuestro DELTA original contiene un elemento de qprima y del alfabeto, procederemos a un nuevo ciclo que recorrerá los elementos que hay en DELTA, es decir las transiciones del AFND, dentro de este último for, existe otra condición, que dice que si el elemento de DELTA no está en los estados del AFD (recordemos que estos estados son la lista e) entonces se irán añadiendo a la lista(ver figura 8).

```
if(DELTA.__contains__((z,y))):
    for w in DELTA[(z,y)]:
        if w not in e:
            e.append(w)
```

Figura 8. Tercer y cuarto for..

En esta parte termina el último for y dentro del segundo for vamos a ir ordenando nuestra lista lexicográficamente con sort() para no tener problemas al evaluar nuestro AFD, de igual forma vamos almacenando los elementos de la lista en nuestro DELTAprima, pasando los elementos de la lista como tupla, finalmente retornamos DELTAprima. (ver figura 9).

```
            e.sort()
            DELTAprima[(x,y)]=tuple(e)
    return DELTAprima;
```

Figura 9. Última parte de la función.

Para ver el resultado de la función, inicializamos DELTAprima como un diccionario y e como una lista, la línea de print hace la llamada a la función para imprimirla. (ver figura 10)

```
DELTAprima ={}
e=[]
print("DELTAprima", conver(Qprima, Sigmaprima, DELTA, DELTAprima, e))
```

Figura 10. Impresión de DELTAprima.

Una vez corremos nuestro código, nos arroja la salida que podemos observar en la figura 11.

```
Qprima [(), ('q0',), ('q1',), ('q0', 'q1')]
Fprima [('q1',), ('q0', 'q1')]
DELTAprima {((), 'a'): (), ((), 'b'): (), (('q0',), 'a'): ('q0', 'q1'), (('q0',)
, 'b'): ('q0', 'q1'), (('q1',), 'a'): ('q0', 'q1'), (('q1',), 'b'): ('q0', 'q1')
, (('q0', 'q1'), 'a'): ('q0', 'q1'), (('q0', 'q1'), 'b'): ('q0', 'q1')}
```

Figura 11. Salida de nuestro código.

Ya que tenemos nuestro código que transforma de AFN a AFD, procedemos a probarlo con algunas cadenas, reutilizando el código de la practica con algunas modificaciones de acuerdo a las variables de nuestro AFD resultante.

En la figura 12, podemos ver algunos ejemplos de cadenas que acepta y que no acepta el AFD.

```
Ejemplos_L=['b', 'bbb', 'aabb', 'aaa']
Ejemplos_Lc=['ba', 'baaa', 'baba', 'bababa']
```

Figura 12. Cadenas que acepta y no acepta el AFD

El código que se encarga de evaluar las cadenas lo vemos en la figura 13.

```
def transicion(estado,sigma):
    #print("estado=",estado,"sigma",sigma)
    estado_siguiente=DELTAprima[(estado,sigma)]
    #print("estado siguiente=",estado_siguiente)
    return estado_siguiente

Ejemplos_L=['b', 'bbb', 'aabb', 'aaa']
Ejemplos_Lc=['ba', 'baaa', 'baba', 'bababa']

for w in Ejemplos_L:
    estado=sprima
    for Sigmaprima in w:
        estado=transicion(estado,Sigmaprima)

    if estado in Fprima:
        print(w, "es aceptada")
    else:
        print(w, "no es aceptada")

for w in Ejemplos_Lc:
    estado=sprima
    for Sigmaprima in w:
        estado=transicion(estado,Sigmaprima)

    if estado in Fprima:
        print(w, "es aceptada")
    else:
        print(w, "no es aceptada")
```

Figura 13. Código que evalúa las cadenas.

La salida de esta parte del código lo vemos en la figura 14.

```
b es aceptada
bbb es aceptada
aabb es aceptada
aaa es aceptada
ba no es aceptada
baaa no es aceptada
baba no es aceptada
bababa no es aceptada
```

Figura 14. Cadenas que acepta y no acepta el AFD.

M2=({q0,q1,q2,q3,q4}, {a,b}, q0, {q2,q4},
{ (q0,a, {q0,q3}), (q0,b,{q0,q1}), (q1,b,{q2}), (q2,a,{q2}),(q2,b,{q2})
(q3,a,{q4}), (q4,a,{q4}), (q4,b,{q4}) }

Ahora, veamos el segundo AFN, en este no se explicará el código, ya que se hizo con el primer AFN, aquí lo único que puede cambiar son los estados, el estado inicial

y los estados finales, además de Qprima, sprima, y Fprima, en la figura 15 podemos ver el código para este AFN;

```
from itertools import chain, combinations
def powerset(iterable):
    s=list(iterable)
    return chain.from_iterable(
            combinations(s,r)
            for r in range(len(s)+1))
Q=['q0','q1','q2','q3','q4']
Sigma=['a','b']
s='q0'
F=['q2','q4']
DELTA={ ('q0','a'):['q0','q3'],
        ('q0','b'):['q0','q1'],
        ('q1','b'):['q2'],
        ('q2','a'):['q2'],
        ('q2','b'):['q2'],
        ('q3','a'):['q4'],
        ('q4','a'):['q4'],
        ('q4','b'):['q4']
        }
print("DELTA", DELTA)
Qprima=list(powerset(Q))
sprima=(s,) #tupla con q0
Sigmaprima=Sigma
Fprima=[]
for q in Qprima:
    for x in q:
        if x in F:
            Fprima.append(q)
print("Qprima",Qprima)
print("Fprima", Fprima)

def conver(Qprima,Sigmaprima, DELTA,DELTAprima, e):
    for x in Qprima:
        for y in Sigmaprima:
            for z in x:
                if(DELTA.__contains__((z,y))):
                    for w in DELTA[(z,y)]:
                        if w not in e:
                            e.append(w)
            e.sort()
            DELTAprima[(x,y)]=tuple(e)
    return DELTAprima;

DELTAprima ={}
e=[]
print("DELTAprima", conver(Qprima, Sigmaprima, DELTA, DELTAprima, e))
```

Figura 15. Código para el segundo AFN.

Como vemos, la función de conversión no cambia.

En la figura 16 podemos ver la salida;

```
Qprima [(), ('q0',), ('q1',), ('q2',), ('q3',), ('q4',), ('q0', 'q1'), ('q0', 'q
2'), ('q0', 'q3'), ('q0', 'q4'), ('q1', 'q2'), ('q1', 'q3'), ('q1', 'q4'), ('q2'
, 'q3'), ('q2', 'q4'), ('q3', 'q4'), ('q0', 'q1', 'q2'), ('q0', 'q1', 'q3'), ('q
0', 'q1', 'q4'), ('q0', 'q2', 'q3'), ('q0', 'q2', 'q4'), ('q0', 'q3', 'q4'), ('q
1', 'q2', 'q3'), ('q1', 'q2', 'q4'), ('q1', 'q3', 'q4'), ('q2', 'q3', 'q4'), ('q
0', 'q1', 'q2', 'q3'), ('q0', 'q1', 'q2', 'q4'), ('q0', 'q1', 'q3', 'q4'), ('q0'
, 'q2', 'q3', 'q4'), ('q1', 'q2', 'q3', 'q4'), ('q0', 'q1', 'q2', 'q3', 'q4')]
Fprima [('q2',), ('q4',), ('q0', 'q2'), ('q0', 'q4'), ('q1', 'q2'), ('q1', 'q4')
, ('q2', 'q3'), ('q2', 'q4'), ('q2', 'q4'), ('q3', 'q4'), ('q0', 'q1', 'q2'), ('
q0', 'q1', 'q4'), ('q0', 'q2', 'q3'), ('q0', 'q2', 'q4'), ('q0', 'q2', 'q4'), ('
q0', 'q3', 'q4'), ('q1', 'q2', 'q3'), ('q1', 'q2', 'q4'), ('q1', 'q2', 'q4'), ('
q1', 'q3', 'q4'), ('q2', 'q3', 'q4'), ('q2', 'q3', 'q4'), ('q0', 'q1', 'q2', 'q3
'), ('q0', 'q1', 'q2', 'q4'), ('q0', 'q1', 'q2', 'q4'), ('q0', 'q1', 'q3', 'q4')
, ('q0', 'q2', 'q3', 'q4'), ('q0', 'q2', 'q3', 'q4'), ('q1', 'q2', 'q3', 'q4'),
('q1', 'q2', 'q3', 'q4'), ('q0', 'q1', 'q2', 'q3', 'q4'), ('q0', 'q1', 'q2', 'q3
', 'q4')]
DELTAprima {((), 'a'): (), ((), 'b'): (), (('q0',), 'a'): ('q0', 'q3'), (('q0',)
, 'b'): ('q0', 'q1', 'q3'), (('q1',), 'a'): ('q0', 'q1', 'q3'), (('q1',), 'b'):
('q0', 'q1', 'q2', 'q3'), (('q2',), 'a'): ('q0', 'q1', 'q2', 'q3'), (('q2',), 'b
'): ('q0', 'q1', 'q2', 'q3'), (('q3',), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), ((
'q3',), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q4',), 'a'): ('q0', 'q1', 'q2',
 'q3', 'q4'), (('q4',), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q1'), 'a'
): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q1'), 'b'): ('q0', 'q1', 'q2', 'q3',
 'q4'), (('q0', 'q2'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q2'), 'b')
: ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q3'), 'a'): ('q0', 'q1', 'q2', 'q3',
'q4'), (('q0', 'q3'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q4'), 'a'):
 ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', '
q4'), (('q1', 'q2'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q1', 'q2'), 'b'):
('q0', 'q1', 'q2', 'q3', 'q4'), (('q1', 'q3'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q
4'), (('q1', 'q3'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q1', 'q4'), 'a'): (
'q0', 'q1', 'q2', 'q3', 'q4'), (('q1', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4
'), (('q2', 'q3'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q2', 'q3'), 'b'): ('
q0', 'q1', 'q2', 'q3', 'q4'), (('q2', 'q4'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'
), (('q2', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q3', 'q4'), 'a'): ('q
0', 'q1', 'q2', 'q3', 'q4'), (('q3', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4')
, (('q0', 'q1', 'q2'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q1', 'q2')
, 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q1', 'q3'), 'a'): ('q0', 'q1',
'q2', 'q3', 'q4'), (('q0', 'q1', 'q3'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), ((
'q0', 'q1', 'q4'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q1', 'q4'), 'b
'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q2', 'q3'), 'a'): ('q0', 'q1', 'q2'
, 'q3', 'q4'), (('q0', 'q2', 'q3'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0'
, 'q2', 'q4'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q2', 'q4'), 'b'):
('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q3', 'q4'), 'a'): ('q0', 'q1', 'q2', 'q
3', 'q4'), (('q0', 'q3', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q1', 'q
2', 'q3'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q1', 'q2', 'q3'), 'b'): ('q0
', 'q1', 'q2', 'q3', 'q4'), (('q1', 'q2', 'q4'), 'a'): ('q0', 'q1', 'q2', 'q3',
'q4'), (('q1', 'q2', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q1', 'q3',
'q4'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q1', 'q3', 'q4'), 'b'): ('q0', '
q1', 'q2', 'q3', 'q4'), (('q2', 'q3', 'q4'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'
), (('q2', 'q3', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q1', 'q2'
, 'q3'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q1', 'q2', 'q3'), 'b'):
('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q1', 'q2', 'q4'), 'a'): ('q0', 'q1', 'q
2', 'q3', 'q4'), (('q0', 'q1', 'q2', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4')
, (('q0', 'q1', 'q3', 'q4'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q1',
 'q3', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q2', 'q3', 'q4'), '
a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q2', 'q3', 'q4'), 'b'): ('q0', 'q1
', 'q2', 'q3', 'q4'), (('q1', 'q2', 'q3', 'q4'), 'a'): ('q0', 'q1', 'q2', 'q3',
'q4'), (('q1', 'q2', 'q3', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0',
'q1', 'q2', 'q3', 'q4'), 'a'): ('q0', 'q1', 'q2', 'q3', 'q4'), (('q0', 'q1', 'q2
', 'q3', 'q4'), 'b'): ('q0', 'q1', 'q2', 'q3', 'q4')}
```

Figura 16. Salida del código.

Al igual que el ejercicio pasado, evaluaremos las cadenas que acepta este AFD, siendo el lenguaje: L={Cadenas que contengan la subcadena aa, bb o ambas}

El código es similar al del ejercicio anterior, cambiando las cadenas que acepta y su complemento. (Ver figura 17).

```
def transicion(estado,sigma):
    #print("estado=",estado,"sigma",sigma)
    estado_siguiente=DELTAprima[(estado,sigma)]
    #print("estado siguiente=",estado_siguiente)
    return estado_siguiente

#L=[Cadenas que contengan la subcadena aa, bb o ambas]
Ejemplos_L=['aa', 'bb', 'aabb', 'abba']
Ejemplos_Lc=['ba', 'abab', 'aba', 'b']

for w in Ejemplos_L:
    estado=sprima
    for Sigmaprima in w:
        estado=transicion(estado,Sigmaprima)

    if estado in Fprima:
        print(w, "es aceptada")
    else:
        print(w, "no es aceptada")

for w in Ejemplos_Lc:
    estado=sprima
    for Sigmaprima in w:
        estado=transicion(estado,Sigmaprima)

    if estado in Fprima:
        print(w, "es aceptada")
    else:
        print(w, "no es aceptada")
```

Figura 17. Código que evalúa las cadenas del lenguaje.

La salida, viendo las cadenas acetadas y las que no, lo podemos ver en la figura 18.

```
aa es aceptada
bb es aceptada
aabb es aceptada
abba es aceptada
ba no es aceptada
abab no es aceptada
aba no es aceptada
b no es aceptada
>>>
```

Figura 18. Cadenas que acepta y no acepta el AFD.

CONCLUSIONES Y RECOMENDACIONES

A partir de los ejercicios realizados en esta práctica, pude comprender de una mejor manera lo revisado en clase sobre como pasar de un autómata finito no determinista a uno determinista, como se vio en la práctica, implementamos un algoritmo en Python para realizar este trabajo, tengo que decir que este problema fue algo difícil de solucionar ya que, si bien hacerlo de forma tradicional a lápiz y papel es más sencillo, pasarlo a un programa es algo completamente distinto.

Puedo concluir que esta práctica será de mucha ayuda, ya que el pasar de un AFND a un AFD de forma tradicional es algo tardado, pues implica muchos estados y transiciones, pero al pasarlo a Python nos ahorraremos mucho tiempo. Además que podemos implementar el código de la practica anterior para evaluar cadenas que acepta y no acepta el AFD resultante.

La práctica se llevó de manera adecuada, además de que el profesor resolvió las dudas que se presentaron y dio el tiempo necesario para realizar esta práctica.

BIBLIOGRAFÍA

- Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2008). *Teoría de autómatas, lenguajes y computación*. Addison Wesley.
- García, P., Pérez, T., Ruiz, J., Segarra, E., Sempere, J. M., & de Parga, M. V. (2001). *Teoría de autómatas y lenguajes formales* (pp. 32-44). Alfaomega.
- Apuntes de la clase de Teoría Computacional por el profesor Jorge Luis Rosas Trigueros.