Instituto Politécnico Nacional
Escuela Superior de Cómputo
Teoria Computacional
Práctica #4 :AFND y AFND a AFD(examen)
Alumno: Machorro Melendez Hugo Andres
Maestro: Rosas Trigueros Jorge Luis
Fecha de realización: 23 - Septiembre - 2019
Fecha de entrega: 02 - Diciembre - 2019

MARCO TEÓRICO.

AFD	AFND
La transición desde un estado puede tener como destino un único estado. Por eso se llama determinista.	La transición desde un estado puede tener multiples destinos. Por eso se le llama no determinista.
No se aceptan transiciones con cadenas vacías.	Permite transiciones con cadenas vacías.
Se permite el uso de backtracking	No siempre se permite el uso de backtracking.
Requiere mas espacio.	Requiere menos espacio.
Una cadena es aceptada si su transición es hacia un estado final.	Una cadena es aceptada si solo una de todas sus posibles transiciones son hacia un estado final.

La definición formal de AFND se basa en la consideración de que a menudo según los algoritmos de transformación de expresiones y gramáticas regulares a AF terminan obteniéndose autómatas con transiciones múltiples para un mismo símbolo o transiciones vacías. Independientemente que sean indeseables, sobre todo para la implementación material, fundamentalmente mecánica, de los autómatas finitos, son imprescindibles durante la modelación de analizadores lexicográficos de los elementos gramaticales de los lenguajes de programación, llamados tókens, como literales numéricos, identificadores, cadenas de texto, operadores, etc.

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.

Haciendo la analogía con los AFDs, en un AFND puede darse cualquiera de estos dos casos:

- Que existan transiciones del tipo $\delta(q,a)=q_1$ y $\delta(q,a)=q_2$ siendo $q_1 \neq q_2$;

- Que existan transiciones del tipo ,δ(q,ε) siendo un estado no-final, o bien un estado final pero con transiciones hacia otros estados.

Cuando se cumple el segundo caso, se dice que el autómata es un autómata finito no determinista con transiciones vacías o transiciones ε (abreviado $AFND_{\varepsilon}$). Estas transiciones permiten al autómata cambiar de estado sin procesar ningún símbolo de entrada.

MATERIAL Y EQUIPO.

1. Sistema operativo Ubuntu.
2. Editor de texto VIM - VI Mejorado.
3. GNU compiler collection.
4. Computadora.

DESARROLLO DE LA PRÁCTICA.

La práctica #4 consistió en hacer un programa que pasará de un autómata finito no determinista a un autómata finito determinista.

1. Debemos pasar de AFND a AFD.

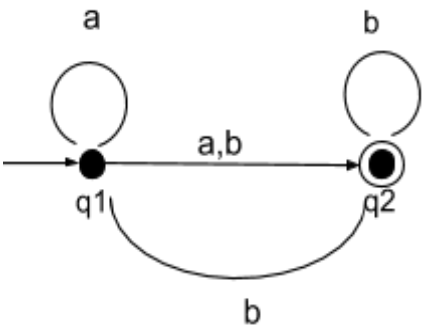
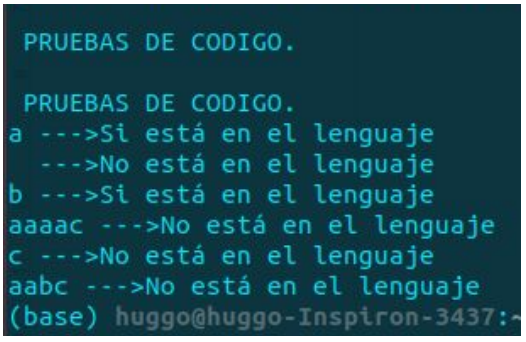


Figura 1. AFND.

```
from itertools import chain, combinations

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

Q=['q0','q1']
s='q0'
F=['q1']
Sigma=['a','b']
DELTA = { ('q0','a'): ['q0','q1'],
          ('q0','b'):['q1'],
          ('q1','a'):[],
          ('q1','b'):['q0','q1'],
          }

Qprima=list(conjunto_potencia(Q))
print("Qprima",Qprima)

Sprima=(s,)
print("Sprima",Sprima)

Sigmaprima=Sigma

print("Construyendo Fprima")
Fprima=[]
for x in Qprima:
    print(x)
    for q in x:
     print(q)
     if q in F:
      Fprima.append(x)
      break
print("Fprima",Fprima)


delta = {}
-- INSERTAR --
```

Figura 2. Implementación.

Explicación: Lo más importante de este programa es saber hacer las transiciones. Primero declaramos el alfabeto, los estados, estado final, estado inicial y las funciones de transición. Después sacamos el conjunto potencia. Por último podemos ver que imprime si la cadena está o no está en el lenguaje.



```
 PRUEBAS DE CODIGO.

 PRUEBAS DE CODIGO.
a --->Si está en el lenguaje
  --->No está en el lenguaje
b --->Si está en el lenguaje
aaaac --->No está en el lenguaje
c --->No está en el lenguaje
aabc --->No está en el lenguaje
(base) huggo@huggo-Inspiron-3437:~
```

Figura 3. Resultados

2. Examen práctico.

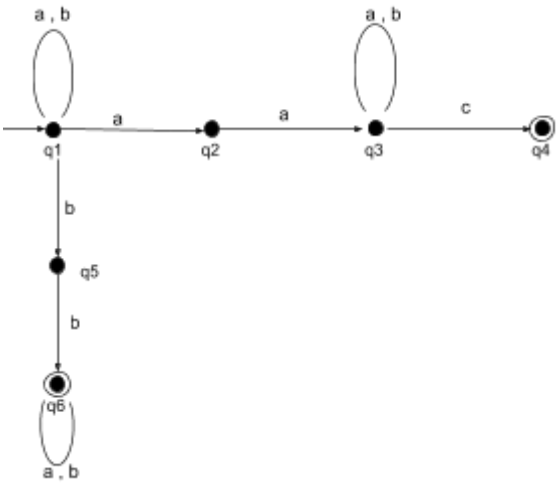


Figura 4. AFND.

```
from itertools import chain, combinations

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

Q=['q0','q1','q2','q3','q4','q5']
s='q0'
F=['q3','q5']
Sigma=['a','b']
DELTA = {('q0','a'): ['q0','q1'],
        ('q0','b'): ['q0','q4'],
        ('q1','a'): ['q2'],
        ('q2','a'): ['q2'],
        ('q2','b'): ['q2'],
        ('q2','c'): ['q3'],
        ('q4','b'): ['q5'],
        ('q5','a'): ['q5'],
        ('q5','b'): ['q5'],
        }

print("Sigma:\n",Sigma)
print("\n")

Sprima=(s,)
print("Sprima:\n",Sprima)
print("\n")

Qprima=list(conjunto_potencia(Q))
print("Qprima:\n",Qprima)
print("\n")

Sigmaprima=Sigma

###print("Construyendo Fprima")
Fprima=[]
for x in Qprima:
   ### print(x)
    for q in x:
     ###print(q)
amen.py" 78L, 1804C
```

Figura 5. Implementación.

Figura 6. Resultado.

Dificultades encontradas: Fue muy complicado encontrar el conjunto potencia para poder pasar de AFND a AFD, todo lo demás viene siendo la misma implementación que un AFD.

CONCLUSIONES Y RECOMENDACIONES.

Los autómatas son muy importantes en la resolución de problemas indefinibles porque estos ayudan a los ingenieros en programación a usarlos como análisis y diseño de la resolución de dicho problema, pueden ser definidos de forma tediosa. A mi como estudiante está muy competitivo este tema.

BIBLIOGRAFÍA.

[1] Manuel Alfonseca Moreno, Justo Sancho Rodríguez, Miguel Martínez Orga. (1987). Teoría de lenguajes, gramáticas y autómatas. Madrid, España: Madrid : Universidad y Cultura, D.L. 1987.

[2] John E. Hopcroft, MOTWANI. 2008. Introducción a la teoría de autómatas lenguajes y computación. Madrid, España.