Reporte: Práctica 1

Barrera Pérez Carlos Tonatihu
Profesor: Saucedo Delgado Rafael Norman
Compiladores
Grupo: 3CM6

1 de septiembre de 2017

Índice

1. Introducción

Los autómatas finitos no determinista y determinista nos permiten identificar si una cadena (secuencia de símbolos) pertenece a un determinado lenguaje regular. Las principales diferencias entre estos dos tipos de autómatas son las siguientes.

- Autómata finito no determinista: Un mismo símbolo puede conducir a diferentes estados desde un mismo estado, además, la cadena vacía es valida.
- Autómata finito determinista: El autómata tiene por cada estado y por cada símbolo sólo una transición hacia otro estado.

La descripción formal de un autómata determinista A es la siguientes.

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

Donde:

- Q es un conjunto de estados finitos.
- Σ es un conjunto finito de símbolos de entrada.
- δ es una función de transición que toma como argumentos un estado y un símbolo y retorna un nuevo estado. En un autómata no determinista la función devuelve un conjunto de estados que pertenecen a Q.
- q_0 es un estado inicial que pertenece al conjunto de estados finitos Q.
- F es un conjunto de estados finales que pertenecen al conjunto Q.

Todas estas características fueron importantes en esta práctica ya que al tenerlas en consideración se pudieron modelar los dos tipos de autómatas mediante el uso de programación orientada a objetos en este caso se utilizo Python. Y con ello poder representar cualquier autómata finito a partir de los elementos que lo componen.

2. Desarrollo

En esta práctica se uso el paradigma orientado a objetos para poder modelar cualquier autómata finito determinista o no determinista con base a sus componentes y con esto poder evaluar cualquier cadena y poder determinar si pertenece a algún determinado lenguaje. Se programaron 3 clases, una para el autómata determinista, otra para el no determinista y una clase extra para poder representar las transiciones que se realizan de una manera más facil.

Primero tenemos el codigo de la clase Transicion.

```
class Transicion:
  def __init__(self, actual, siguiente, caracter):
    self.actual = actual
    self.siguiente = siguiente
    self.caracter = caracter

  # Metodo para poder imprimir de forma legible una instancia de esta clase
  def __str__(self):
    return '{}->{}: {}'.format(self.actual, self.siguiente, self.caracter)
```

Como se puede observar en el codigo nuestra transicion consiste de un estado actual, uno siguiente y un simbolo que realiza esa transición. Esto nos ayudara a modelar las clases de los automatas de una manera más clara.

El código de la clase AFN fue:

```
class AFN:
 def __init__(self):
     self.alfabeto = set()
     self.estados_finales = set()
     self.estado_inicial = 0
     self.estados_actuales = list()
     self.transiciones = list()
     self.estados = set()
     self.estado_error = -1

 def agregar_transicion(self, actual, siguiente, caracter):
     self.transiciones.append(Transicion(actual, siguiente, caracter))

 def agregar_finales(self, estados):
     self.estados_finales = estados

 def agregar_alfabeto(self, alfabeto):
     self.alfabeto = alfabeto
     self.alfabeto.add('e')

 def agregar_inicial(self, estado):
     self.estado_inicial = estado

 def agregar_estado(self, estado):
     if estado in self.estados:
         print("Estado repetido")
     else:
         self.estados.add(estado)

 def evaluar_cadena(self, cadena):
     self.estados_actuales = self.estados_epsilon(self.estado_inicial)
     for caracter in cadena:
         if caracter not in self.alfabeto:
             return False
         siguientes_estados = self.obtener_siguientes(caracter)
         self.estados_actuales = []
         for e in siguientes_estados:
             self.estados_actuales.extend(self.estados_epsilon(e))

     for estado in self.estados_actuales:
         if estado in self.estados_finales:
             return True
     return False
```

```
44
45      def obtener_siguientes(self, caracter):
46          siguientes = list()
47          for estado in self.estados_actuales:
48              for transicion in self.transiciones:
49                  if estado == transicion.actual and transicion.caracter == caracter:
50                      siguientes.append(transicion.siguiente)
51          return siguientes
52
53      def estados_epsilon(self, estados):
54          epsilon = list()
55          epsilon.append(estados)
56          for estado in epsilon:
57              for t in self.transiciones:
58                  if t.caracter == 'e' and t.actual == estado and (t.siguiente not in epsilon):
59                      epsilon.append(t.siguiente)
60          return epsilon
```

El código de la clase AFD hereda muchas propiedades y métodos de la clase AFN modificando el método que se encarga de la evaluación de la cadena.

```
1   class AFD(AFN):
2     def agregar_alfabeto(self, alfabeto):
3       self.alfabeto = alfabeto
4
5     def evaluar_cadena(self, cadena):
6       self.estados_actuales.append(self.estado_inicial)
7       continuar = True
8       for caracter in cadena:
9         if caracter not in self.alfabeto:
10          return False
11        for transicion in self.transiciones:
12          if transicion.actual == self.estados_actuales[0]:
13            if transicion.caracter == caracter:
14              self.estados_actuales[0] = transicion.siguiente
15              continuar = True
16              break
17          else:
18            continuar = False
19        if not continuar:
20         self.estados_actuales[0] = self.estado_error
21
22      return self.estados_actuales[0] in self.estados_finales
```

3. Resultados

hola

4. Conclusiones

Fue indispensable conocer los componentes de este tipo de autómatas ademas de saber el como trabajan ya que sin este conocimiento seria imposible el poder modelarlos en una clase. La parte difícil de esta práctica fue el AFN ya que al poder ir a varios estados desde una mismo con el mismo símbolo por lo que aumenta su complejidad al aumentar el numero de caminos que se pueden tomar. Así que el revisar libros y ejercicios relacionados con este tema fue importante para poder solucionar este problema.

5. Referencias