Práctica 3

Ian Mendoza Jaimes

Compiladores

Profesor: Rafael Norman Saucedo Delgado

Grupo: 3CM6

Índice

1. Introducción

Los autómatas al igual que las expresiones regulares son capaces de modelar lenguajes regulares. Estos lenguajes son particularmente útiles para describir algún lenguaje de programación.

Particularmente, los autómatas y las expresiones regulares son usados en la etapa del analisis léxico de un compilador [2]. Primero se escriben expresiones regulares que describiran a las clases léxicas del lenguaje a traducir. Con estas expresiones, obtendremos sus respectivos autómtas. El problema, es que la construcción de Thompson arroja autómatas no deterministas, los cuales son más faciles de modelar, pero no de programar, además son más tardados de evaluar.

El tiempo de ejecución es muy importante en cualquier algoritmo, y el analizador léxico no es la excepción. Debido a que en esta etapa se evalua todo el código fuente de algún programa a traducir, puede tomar bastante tiempo. De ahi, surge la necesidad de utilizar autómtas deterministas para la obtención de tokens.

El algoritmo de los subconjuntos resuelve este problema [1]. Como entrada recibe un AFN y de salida arroja un AFD. Este método sin duda es uno de los más efectivos para la conversión de un AFN a un AFD, pues nos ahorra la necesidad de crear un AFN intermedio para lidear con las transiciones epsilon.

Antes de comenzar con la programación de este algoritmo, conviene describir las siguientes funciones auxiliares:

- $Cerradura\epsilon(e)$, regresa todos los caminos que podemos seguir con una transición epsilon partiendo de algun estado dado e.
- $Mover(e, s)$, regresa los caminos que podemos seguir partiendo de un estado e evaluando el caracter s.

El algoritmo de los subconjuntos consiste en los siguientes pasos:

1. Agregar Cerradura ϵ (Inicial de AFN) a E
2. Por cada estado $e \in E$
3. Por cada símbolo s
4. Agregar Cerradura ϵ (Mover(e, s))
5. Agregar transición $E \xrightarrow{s}$ Cerradura ϵ (Mover(e, s))
6. El inicial del AFD es Cerradura ϵ (Inicial de AFN)
7. Los finales del AFD contienen finales del AFN

2. Desarrollo

2.1. Descripción de la problema

Se debe implementar un programa que pueda convertir un AFN en un AFD. Para esto, se utilizará el algoritmo de los subconjuntos [1].

2.2. Código

Aqui se muestran los códigos que componen a esta práctica. La clase Afnd y el analizador ya se presentaron en los reportes de las prácticas anteriores y se han omitido para acortar espacio.

main.py

```
from afnd import Afn
from analizador import Analizador
from convertir import Convertidor

class Main:
    def iniciar(self):
        convertidor = Convertidor()
        automata = Afn()

        expresion = input("Ingresa una expresion: ")
        analizador = Analizador()

        analizador.analizar(expresion)
        print(analizador.lista)

        analizador.automata.crearTablaEstados()
        for x in analizador.automata.tablaEstados:
            print(x)

        convertidor.iniciarAfnAuxiliar(analizador.automata)
        nuevoAfd = convertidor.convertir()
        nuevoAfd.crearTablaEstados()

        print("_____")

        for estado in nuevoAfd.tablaEstados:
            print(estado)

        for estado in nuevoAfd.estados:
            print(estado.final)

        nuevoAfd.dibujarAutomata()
        #analizador.automata.dibujarAutomata()

        while True:
            cadena = input('Ingresa una cadena a evaluar:')
```

```
            print(nuevoAfd.evaluarCadena(cadena))
            if input('Quieres otra? s/n   ') != 's':
                break

main = Main()
main.iniciar()
```

convertidor.py

```
from afnd import Afn, Afd

class Convertidor:
    def __init__(self):
        self.afnAuxiliar = None
        self.afdAuxiliar = None
        self.nuevosEstados = None
        self.funcionTransicion = None


    def eliminarTransicionesEpsilon(self):
        if self.afnAuxiliar == None:
            return None

        self.obtenerNuevosEstados()

        nuevoAfn = Afn()
        for nuevo in range(0, len(self.nuevosEstados)):
            nuevoAfn.anadirEstado()

        contador = 1
        for nuevo in  self.nuevosEstados:
            for elemento in nuevo:
                if self.afnAuxiliar.estados[elemento-1].final:
                    if not nuevoAfn.estados[contador-1].final:
                        nuevoAfn.anadirFinal(contador)
                for transicion in self.afnAuxiliar.estados[elemento-1].
                    transiciones:
                    if len(transicion.condiciones) > 0:
                        for siguiente in self.obtenerEstado(transicion.
                            siguiente):
                            nuevoAfn.anadirTransicion(contador,
                                siguiente, transicion.condiciones)
            contador += 1

        return nuevoAfn


    def obtenerEstado(self, estado):
        if type(estado) == set:
            contador = 1
            for nuevoEstado in self.nuevosEstados:
```

```
40                  if estado == nuevoEstado:
41                      return contador
42                  contador += 1
43          else:
44              nuevo = {estado}
45              contador = 1
46              for nuevoEstado in self.nuevosEstados:
47                  if nuevo == nuevoEstado:
48                      return [contador]
49                  contador += 1
50
51              contador = 1
52              auxiliar = list()
53              for nuevoEstado in self.nuevosEstados:
54                  if nuevo <= nuevoEstado:
55                      auxiliar.append(contador)
56                  contador += 1
57
58              return auxiliar
59
60
61      def mover(self, estados, caracter):
62          #print('mover',estados)
63          conjunto = set()
64          for estado in estados:
65              auxiliar = set()
66
67              for transicion in self.afnAuxiliar.estados[estado-1].
                    transiciones:
68                  for condicion in transicion.condiciones:
69                      if condicion == caracter:
70                          auxiliar.add(transicion.siguiente)
71                          break
72              conjunto = conjunto.union(auxiliar)
73          #print('mover-retorno',conjunto)
74          return conjunto
75
76
77      def obtenerNuevosEstados(self):
78          self.nuevosEstados = list()
79          for estado in self.afnAuxiliar.estados:
80              self.nuevosEstados.append(self.cerraduraEpsilon(estado.
                    nombre))
81
82
83      def cerraduraEpsilon(self, estados):
84          #print('epsilon',estados)
85          conjunto = set()
86          for estado in estados:
87              auxiliar = set()
88              arreglo = list()
89              auxiliar.add(estado)
90
91              for transicion in self.afnAuxiliar.estados[estado-1].
```

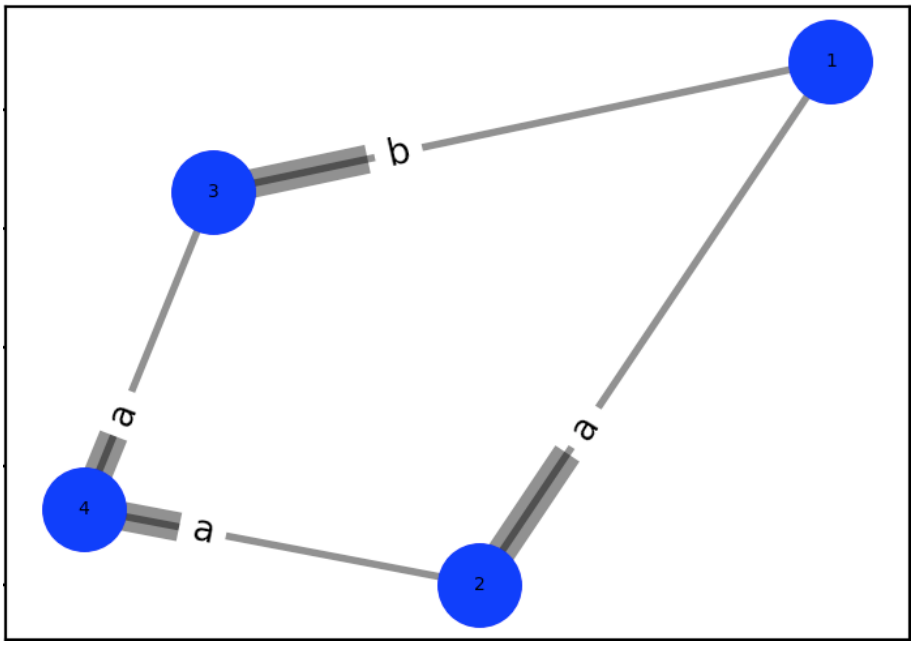
```
                transiciones:
92                  if len(transicion.condiciones) == 0:
93                      #print('EL SIGUIENTE: ', transicion.siguiente)
94                      auxiliar = auxiliar.union(self.cerraduraAuxiliar(
                            transicion.siguiente))
95
96              conjunto = conjunto.union(auxiliar)
97
98          return conjunto
99
100
101     def cerraduraAuxiliar(self, estado):
102         aux = set()
103         aux.add(estado)
104
105         for transicion in self.afnAuxiliar.estados[estado-1].
                transiciones:
106             if len(transicion.condiciones) == 0:
107                 #print('EL SIGUIENTE AUX: ', transicion.siguiente)
108                 aux = aux.union(self.cerraduraAuxiliar(transicion.
                        siguiente))
109
110         return aux
111
112 ##=====================================================
113 ## agarrate, esto se va a poner feo....
114 ##=====================================================
115
116     def convertir(self):
117         self.estadosRevisados = dict()
118         self.nuevosEstados = list()
119         self.funcionTransicion = list()
120
121         self.afdAuxiliar = Afd()
122
123         #print('LEL:',self.cerraduraEpsilon({self.afnAuxiliar.inicial}))
124         self.nuevosEstados.append(self.cerraduraEpsilon({self.
                afnAuxiliar.inicial}))
125         self.afdAuxiliar.anadirEstado()
126
127         con = 0
128         for estado in self.nuevosEstados:
129             con += 1
130             for condicion in self.afnAuxiliar.historialCondiciones:
131                 #print('condicion', condicion)
132                 aux = self.cerraduraEpsilon(self.mover(estado, condicion
                        ))
133                 if len(aux) != 0:
134                     e = self.agregar(aux)
135                     self.afdAuxiliar.anadirTransicion(con, e, condicion)
136
137         return self.afdAuxiliar
138
139
```

```
140     def agregar(self, conjunto):
141         con = 0
142         for estado in self.nuevosEstados:
143             con += 1
144             if estado == conjunto:
145                 return con
146 
147         self.nuevosEstados.append(conjunto)
148         e = self.afdAuxiliar.anadirEstado()
149 
150         con = 0
151         for elemento in conjunto:
152             if self.afnAuxiliar.estados[elemento-1].final:
153                 if con == 0:
154                     self.afdAuxiliar.anadirFinal(e)
155                     con += 1
156 
157         return e
158 
159 
160 
161     def llenarFuncionTransicion(self):
162         i = 0
163         conjuntoAuxiliar = set()
164         while i < len(self.nuevosEstados):
165             self.funcionTransicion.append({})
166             for condicion in self.afnAuxiliar.historialCondiciones:
167                 for elemento in self.nuevosEstados[i]:
168                     conjuntoAuxiliar = self.obtenerSiguiente(condicion,
                            elemento)
169                     if len(conjuntoAuxiliar) > 0:
170                         if condicion in self.funcionTransicion[i]:
171                             self.funcionTransicion[i][condicion].union(
                                    conjuntoAuxiliar)
172                         else:
173                             self.funcionTransicion[i][condicion] =
                                    conjuntoAuxiliar
174 
175                 if condicion in self.funcionTransicion[i]:
176                     incertar = True
177                     for revisado in self.nuevosEstados:
178                         if revisado == self.funcionTransicion[i][
                                condicion]:
179                             incertar = False
180                             break
181                     if incertar:
182                         self.nuevosEstados.append(self.funcionTransicion
                                [i][condicion])
183             i+=1
184 
185 
186     def obtenerSiguiente(self, condicion, estado):
187         conjunto = set()
188         for transicion in self.afnAuxiliar.estados[estado-1].
```

```
            transiciones:
189             for c in transicion.condiciones:
190                 if condicion == c:
191                     conjunto.add(transicion.siguiente)
192         return conjunto
193 
194 
195     def iniciarAfnAuxiliar(self, automata):
196         if type(automata) is not Afn:
197             return None
198 
199         self.afnAuxiliar = automata
200         return 0
201 
202 
203     def iniciarAfdAuxiliar(self, automata):
204         if type(automata) is not Afd:
205             return None
206 
207         self.afdAuxiliar = automata
208         return 0
```

3. Resultados

En esta sección se presentan capturas de pantallas de la ejecución del código presentado anteriormente.



(a) AFD obtenido de un AFN

```
Ingresa una expresion: (a+b)(a)
['a', 'b', '+', 'a', '.']
[['a', [2]]]
[[' ', [7]]]
[['b', [4]]]
[[' ', [7]]]
[[' ', [3, 1]]]
[[' ', [7]]]
[['a', [8]]]
[]
_____________________________________
[['a', [2]], ['b', [3]]]
[['a', [4]]]
[['a', [4]]]
[]
False
False
False
True
SOY EL INICIAL: 1
SOY EL FINAL: 4
Ingresa una cadena a evaluar:aa
True
Quieres otra? s/n   s
Ingresa una cadena a evaluar:ab
False
Quieres otra? s/n   s
Ingresa una cadena a evaluar:ba
True
Quieres otra? s/n   n
```

(b) Varias evaluaciones de cadenas

Figura 1: (a) El autómata generado por la expresión regular: $(a+b)a$. (b) El resultado de varias evaluaciones del autómata.

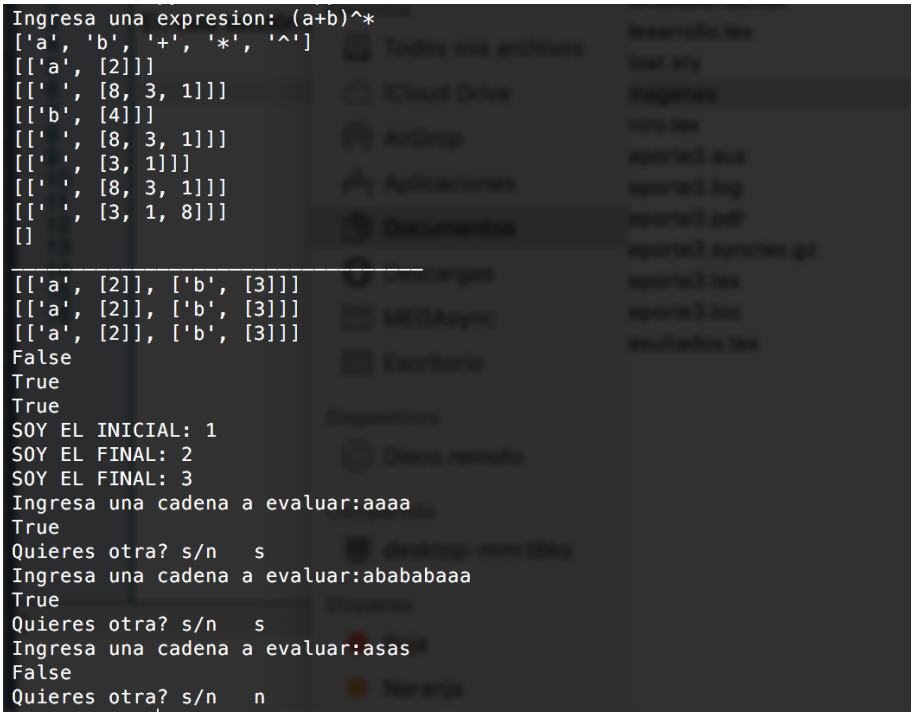


Figura 2: Varias cadenas evaluadas con el autómata generado por la expresión regular: $(a + b)^*$.

4. Conclusiones

Los autómatas al igual que las expresiones regulares, nos ayudan a modelar lenguajes regulares. Ambos tienen una aplicación amplia en el mundo de las ciencias computacionales. En este caso nos hemos enfocado en sus capacidades para modelar la primera etapa de un compilador.

El analizador léxico puede tomar gran parte del tiempo de ejecución de todo el compilador, debido a que es la única etapa en la que se tiene que evaluar todo el código fuente. Por esta razón, es conveniente tener un método eficiente para evaluar los caracteres y obtener los tokens. La mejor manera que tenemos, es utilizar un AFD. Sin embargo, al momento de utilizar la construcción de Thompson para convertir expresiones regulares a autómatas, obtenemos un AFN, que no es tan eficiente como un AFD.

El algoritmo de los subconjuntos esta pensado en solucionar este problema, pues convierte un AFN en un AFD. En esta práctica tuve la oportunidad de implementar este algoritmo y así, poder comparar los tiempos de ejecución de un AFN y un AFD que modelen el mismo lenguaje.

5. Referencias

[1] *Compilers: principles, techniques and tools*. 2001, ch. 2.

[2] *Introduction to Automata Theory, Lenguajes and Computation*. 2001, ch. 2.