Práctica 2

Construcción de Thompson

Oscar Eduardo López Cabagné
2015070715

Índice

Introducción

El Algoritmo de Thompson

El algoritmo Thompson (también conocido como método de Thompson) creado por Ken Thompson y Dennis Ritchie, sirve para obtener autómatas finitos no deterministas con transiciones vacías (AFND-ε) a partir de expresiones regulares (ER).

Dadas las reglas que definen las expresiones regulares se pueden escribir como AFND-e:

- Φ es una expresión regular que describe el lenguaje vacío: en este caso se construye un AFND-e de dos estados, uno inicial y otro final, que no tienen transiciones, por lo cual no están conectados. De esta manera el autómata reconoce el lenguaje vacío.
- ε es una expresión regular que describe el lenguaje {ε}, que es un lenguaje que únicamente contiene la cadena vacía: el autómata que reconoce este lenguaje es aquel que el estado inicial también es final.
- Si "a" esta en el alfabeto, "a" (sin comillas) es una expresión regular que describe el lenguaje {a}: el autómata que reconoce este lenguaje tiene definida una transición desde el estado inicial hacia un estado final.
- Si existen r y s expresiones regulares r es una expresión regular que describe L(r) y s es una expresión regular que describe L(s)
    - r+s describe L(r) U L(s) (lenguaje generado por r union lenguaje generado por s).
    - r.s describe L(r). L(s) (lenguaje generado por r concatenado lenguaje generado por s).
    - r* describe L(r)* (lenguaje generado por r clausura).

Las precedencias de operador son *,., +.

Existen varios programas que realizan este algoritmo y de hecho es habitual también pasar de AFND-e a AFND y de AFND a AFD, también suele ocurrir que el AFD no sea mínimo y se usa otro algoritmo para conseguir el AFD mínimo.

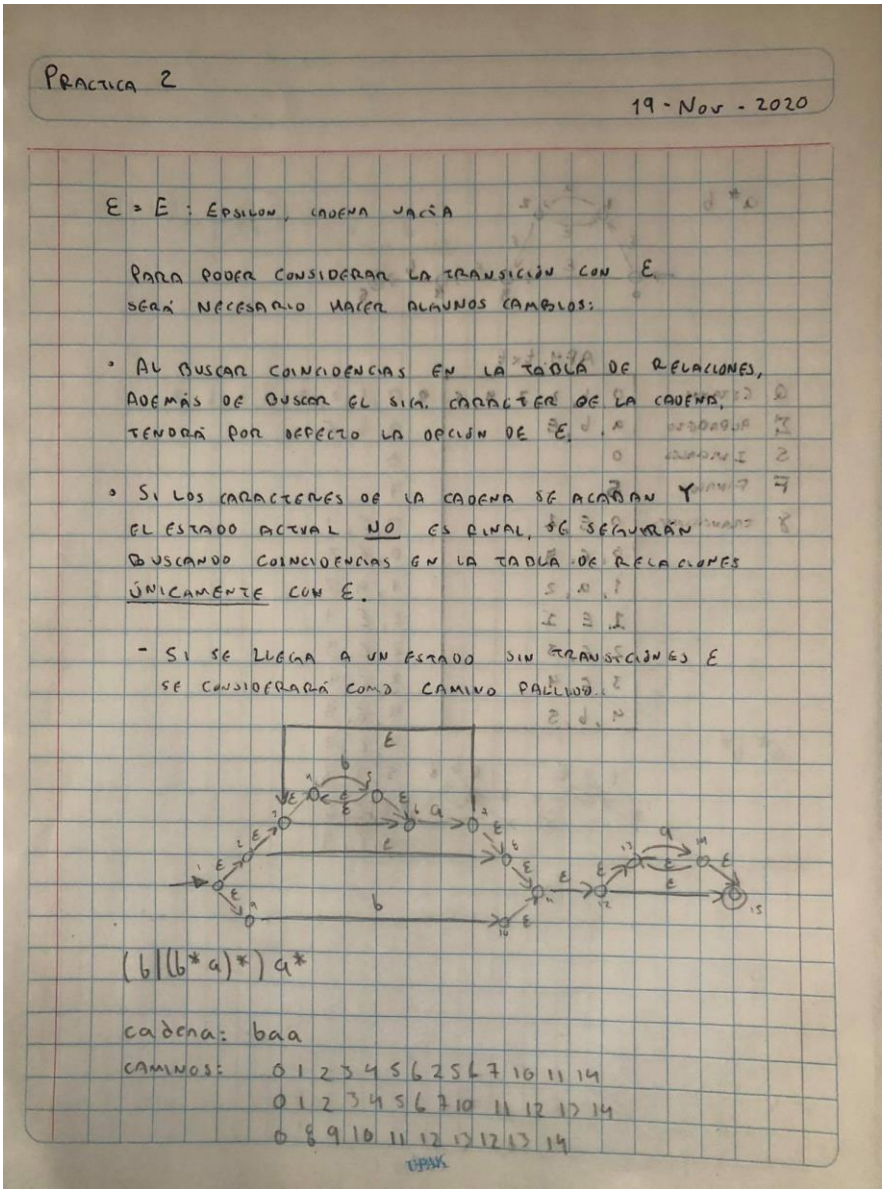
Cualquier ER puede ser reconocida por un AFD ya que los lenguaje regulares de tipo 3 son reconocidos por un AFD como autómata más restrictivo habiendo equivalencia entre no determinismo y determinismo. Generalmente los programas que aplican el algoritmo suelen transformar una ER a AFD mínimo. Algunos programas son:

- Minerva
- MTSolution

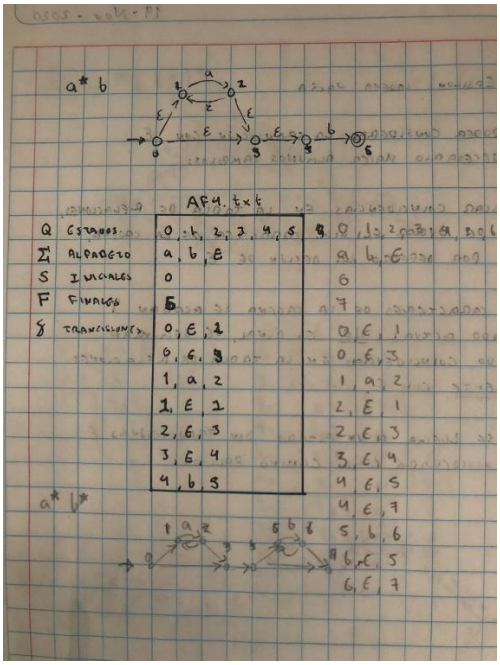
Desarrollo

En esta práctica bastará con realizar algunos cambios y ajustes en nuestro algoritmo utilizado en la práctica pasada.

De manera similar a la anterior, comencé haciendo algunas anotaciones sobre papel, resaltando todos los puntos que consideré importantes, así como algunos de los aspectos a tomar en cuenta para el nuevo algoritmo.



También hice un pequeño autómata con transiciones épsilon para apoyarme durante las pruebas del nuevo programa.



Antes de comenzar la programación, le di un vistazo al código completo de la práctica anterior. Gracias al diseño modular que implementé, los cambios solo deberían realizarse en una única función, la función de Análisis.

```python
def analisis(estado, posicion, cuenta, camino):
    camino += estado
    if cuenta <= len(cadena):
        transicion = cadena[posicion]
        i = 0
        while i <= lineas:
            if relaciones[i][0] == estado:
                if relaciones[i][1] == transicion:
                    analisis(relaciones[i][2], posicion + 1, cuenta + 1, camino)

            i += 1

    else:
        try:
            eFinales.index(estado)
            print(f">: Camino exitoso: {camino}")

        except:
            print("")
            #print(f"Camino fallido: {camino}")

    return 0
```

Función de Análisis Original

El primer cambio que realicé fue que, en el momento en que el programa busca coincidencias dentro de la tabla de relaciones, al encontrar una relación que incluya al estado actual, en lugar de únicamente buscar que contenga una transición coincidente con el siguiente carácter en la cadena, también busca una transición con E.

```
def analisis(estado, posicion, cuenta, camino):
    camino += " " + estado
    #print(camino)
    if cuenta <= len(cadena):
        transicion = cadena[posicion]
        i = 0
        while i <= lineas:
            if relaciones[i][0] == estado:
                if relaciones[i][1] == transicion:
                    analisis(relaciones[i][2], posicion + 1, cuenta + 1, camino)

                if relaciones[i][1] == 'E':
                    analisis(relaciones[i][2], posicion, cuenta, camino)

            i += 1
```

Fragmento de la función análisis (1)

De esta manera, si el estado actual no tiene transiciones con el carácter que buscamos, pero si tiene transiciones E, el programa será capaz de tomar ese camino y pasar al siguiente estado.

Adicional a esto, también será necesario una nueva fase de búsqueda para el caso en que la cadena ya concluyó, pero el estado actual no es el final.

Para realizar esto, el programa realiza una búsqueda similar a la anterior, solo que en esta ocasión únicamente buscará transiciones con E. Si se llega a un estado final a través de esta búsqueda, se tomará como un camino exitoso. Si se llega a un estado no final y sin transiciones E, se termina ese proceso y se continua por otro camino.

```
    else:
        for final in eFinales:
            if estado != final:
                i = 0
                while i <= lineas:
                    if relaciones[i][0] == estado:
                        if relaciones[i][1] == 'E':
                            analisis(relaciones[i][2], posicion, cuenta, camino)

                    i += 1
            else:
                print("Camino Exitoso: " + camino)
    return 0
```

Fragmento de la función análisis (2)

Con esos dos cambios, el nuevo algoritmo estará finalizado. Ahora es capaz de seguir un camino a través de transiciones E para completar la cadena y/o alcanzar un estado final.

La nueva función análisis se ve así:

```
74  def analisis(estado, posicion, cuenta, camino):
75      camino += " " + estado
76      #print(camino)
77      if cuenta <= len(cadena):
78          transicion = cadena[posicion]
79          i = 0
80          while i <= lineas:
81              if relaciones[i][0] == estado:
82                  if relaciones[i][1] == transicion:
83                      analisis(relaciones[i][2], posicion + 1, cuenta + 1, camino)
84
85                  if relaciones[i][1] == 'E':
86                      analisis(relaciones[i][2], posicion, cuenta, camino)
87
88              i += 1
89
90      else:
91          for final in eFinales:
92              if estado != final:
93                  i = 0
94                  while i <= lineas:
95                      if relaciones[i][0] == estado:
96                          if relaciones[i][1] == 'E':
97                              analisis(relaciones[i][2], posicion, cuenta, camino)
98
99                      i += 1
100             else:
101                 print("Camino Exitoso: " + camino)
102     return 0
```

Código de función analisis

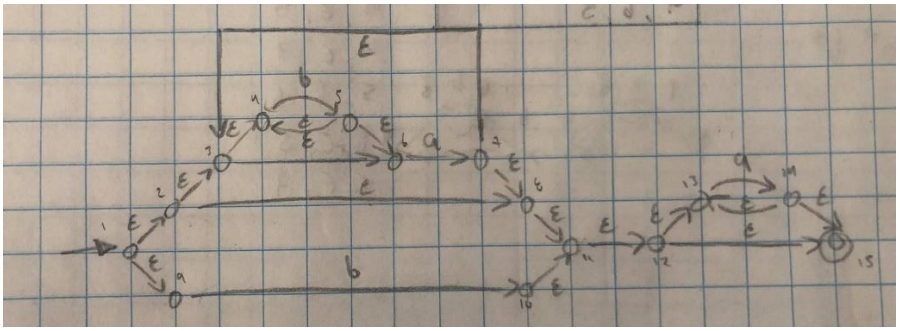
El resto de las funciones (renglones, leer, obtenerCadena e inicio) no requieren ninguna modificación y seguirán funcionando sin ningún problema. Únicamente agregué un comentario en la función leer para recordar que este programa debe contener el carácter ‘E’ como representante de la cadena vacía dentro del renglón del alfabeto.

```
27  def leer(path):
28      global estados
29      global alfabeto            # Ahora el alfabeto contendrá a E : Epsilon (Cadena Vacía)
30      global eIniciales
31      global eFinales
32      global relaciones
33      global lineas
34      i = 0
35      count = 0
36      file = open(path, "r")
```

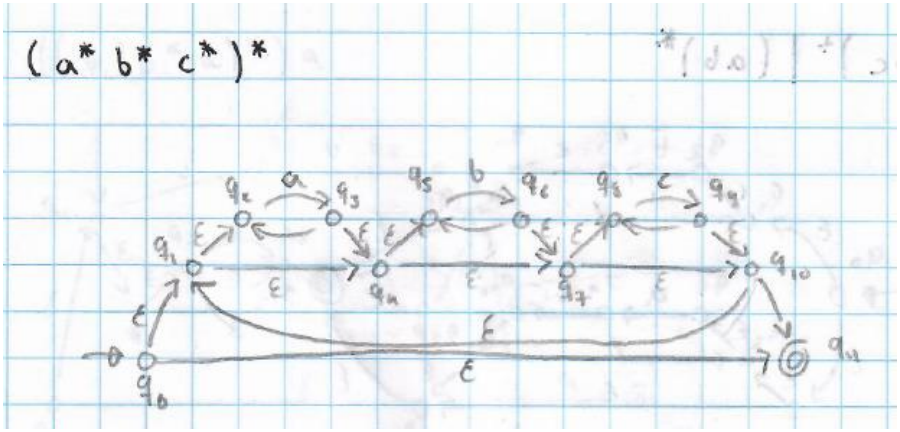
Fragmento de la función Leer

Para generar los txt usé los diagramas siguientes:

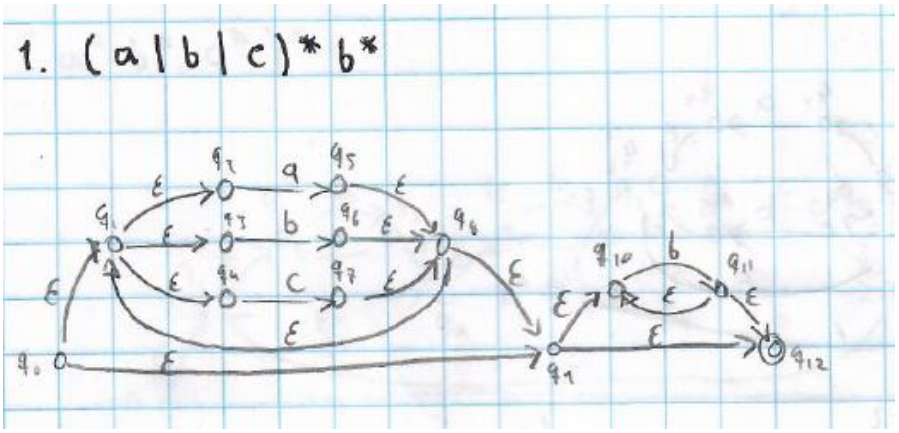
- (b|(b*a)*)a*



- (a*b*c*)*

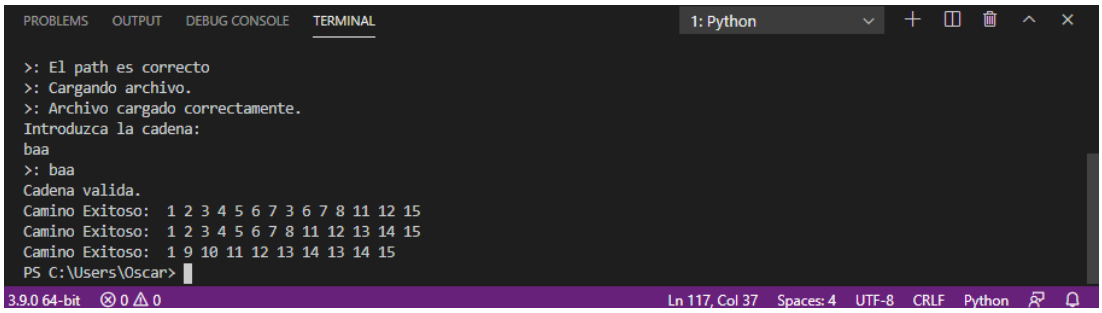


- (a|b|c)*b*

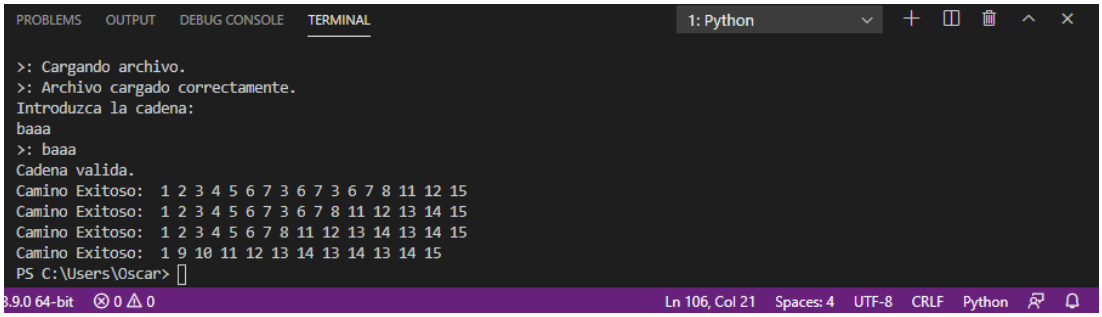


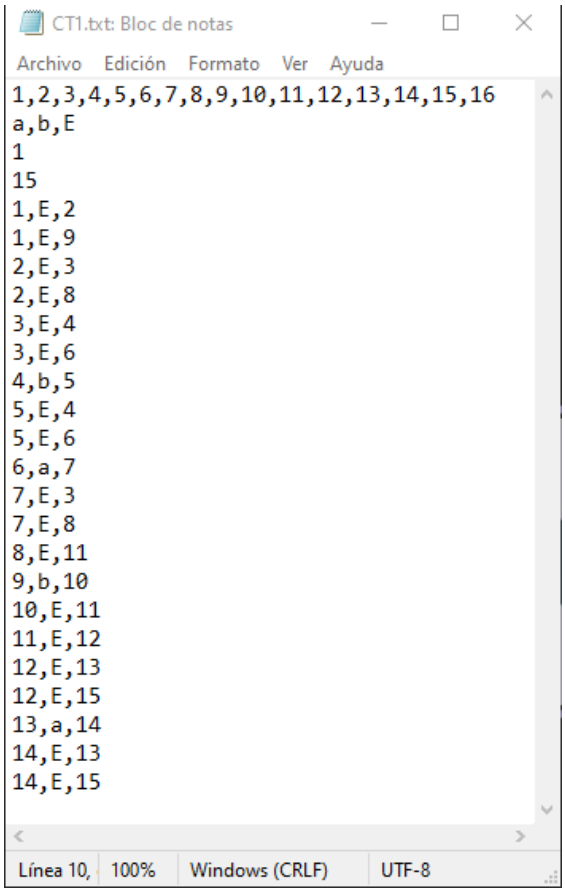
Pruebas

- (b|b*a)*)a*
  - baa



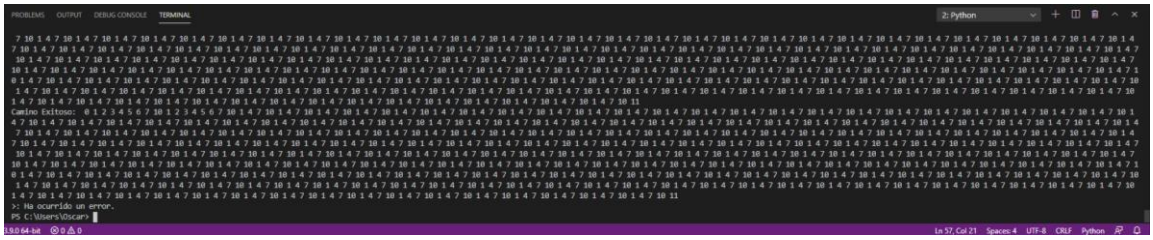
  - baaa

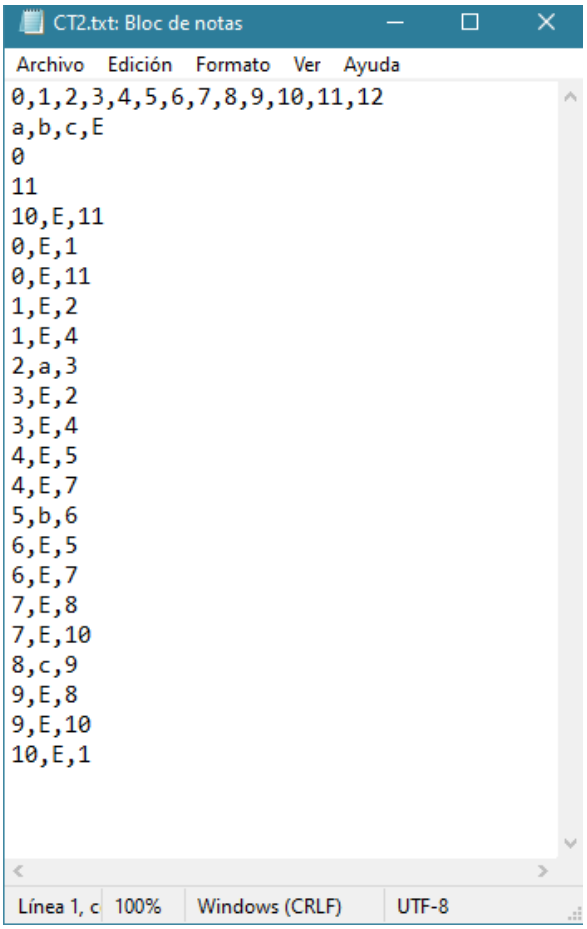




- (a*b*c*)*
    - cabab

    - abab





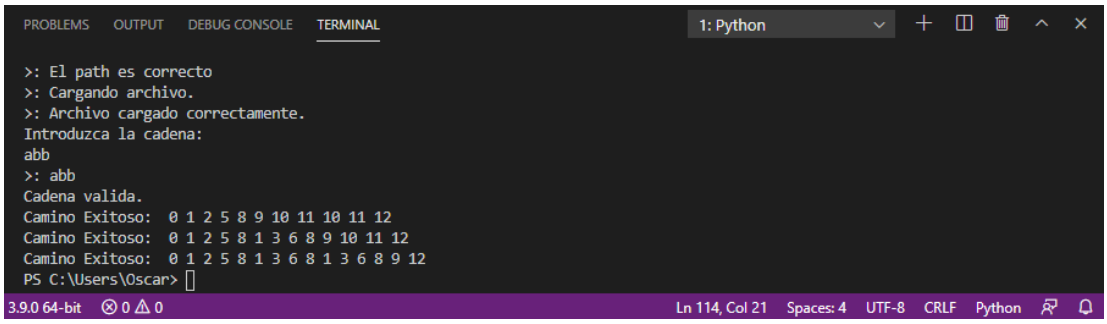
En esta prueba tuve problemas durante ejecución, pues, al terminar de escribir la cadena sobre el autómata, el programa comienza a dar vueltas entre transiciones Épsilon.

Esto lo podemos notar en las capturas fácilmente, pues al concluir la cadena a escribir, en lugar de ir al estado 11 (Final), comienza a seguir el recorrido: 1 4 7 10, una y otra vez. Las variantes de camino exitoso que muestra en realidad son el mismo camino con una vuelta extra por transiciones Épsilon.

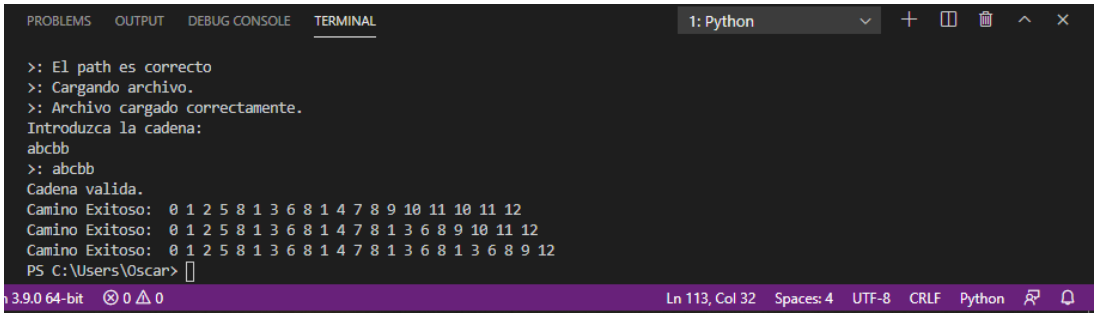
Para solucionarlo se me ocurrió poner la transición al estado final en el principio del arreglo de relaciones, esperando que decidiera seguir ese camino antes que volver al ciclo, sin embargo, nada cambió.

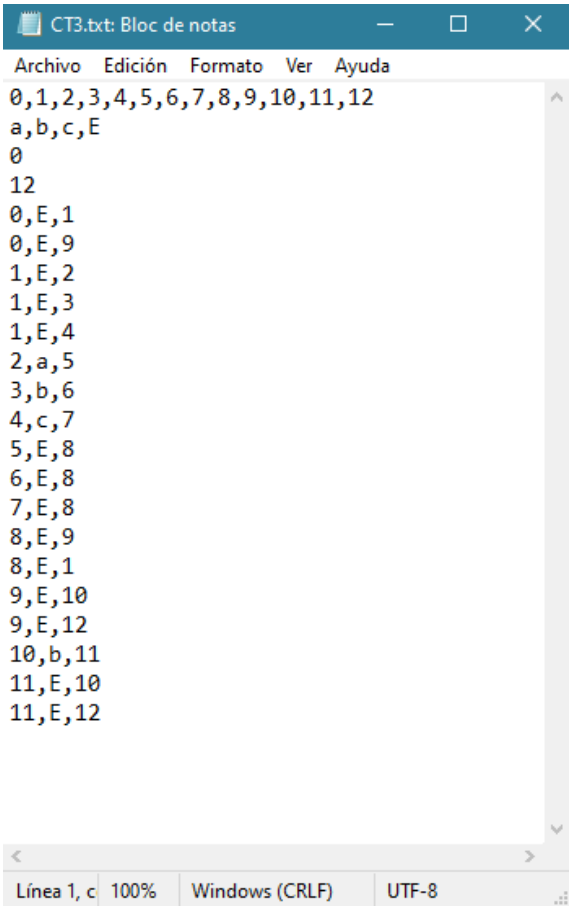
También intenté modificar la preferencia de selección para evitar el ciclo. Una vez más, nada cambió.

- (a|b|c)*b*
  - abb



  - abcbb





Fé de Erratas

Un tiempo después de haber entregado por primera vez este reporte me di cuenta que tuve un error con los archivos txt. Concretamente, el archivo correspondiente a la segunda y tercera prueba se había duplicado y eran iguales.

En esta actualización ese error fue solucionado y las capturas anexadas ya corresponden a sus respectivos archivos.

Conclusión

En esta práctica me quedó mucho más clara la función e importancia de las transiciones Épsilon, además que pude practicar y comprender mejor el proceso de obtención del gráfico de un autómata a partir de su expresión.

Bibliografía

[1]2020. [Online]. Available: http://www.hopelchen.tecnm.mx/principal/sylabus/fpdb/recursos/r94237.PDF. [Accessed: 19- Nov- 2020].

[2]2020. [Online]. Available: http://www.hopelchen.tecnm.mx/principal/sylabus/fpdb/recursos/r94237.PDF. [Accessed: 19- Nov- 2020].

[3]"Metodo Thompson", *prezi.com*, 2020. [Online]. Available: https://prezi.com/p/oksn37ntxmha/metodo-thompson/. [Accessed: 19- Nov- 2020].