Universidad del Valle de Guatemala
Diseño de Lenguajes de Programación
Documentación

Ana Lucia Diaz Leppe
151378

PROYECTO DE COMPIS

Evidencia:

https://www.youtube.com/watch?v=B6prsvF6kts&feature=youtu.be&hd=1

Github: https://github.com/LEPPEDIAZ/Proyecto_01_Compis

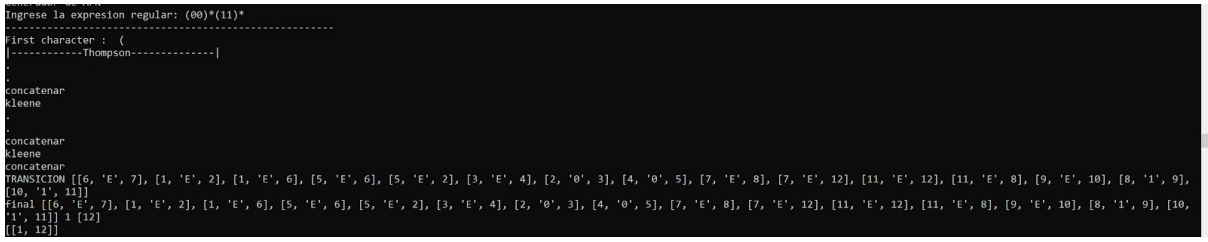
El proyecto se trabajó en python con clases, se comenzará con describir el orden de estas mismas.

1.el primer archivo que es de llamar es el de la expresión regular. Dentro de esta misma se establecieron las siguientes funciones una verificación de de que sea una expresión regular válida, los brackets válidos y el manejo de expresiones con “?”, luego se trabajó con una de las funciones más importantes que es infix a postfix. En donde determinamos que el escáner de la expresión va de izquierda a derecha. Infix es cuando un operador se encuentra entre cada par de operandos. La expresión postfix es cuando se sigue a un operador para cada par de operandos. Por lo tanto se determinaron las diferentes operaciones y variables como (que se puso en orden principal, al igual de su pertenencia al alfabeto y el manejo de los paréntesis. El siguiente paso fue utilizar el evaluador de expresiones para cada una de las diferentes operaciones. Una vez se hubo trabajado esto se inició la construcción de un AFN a partir de una expresión regular por medio del algoritmo de thompson, (es importante notar que se llamó a la clase de transformación en donde se determinó la base del programa siendo de quién es , qué símbolo es y para quien es , es decir :

qo -> a ->q1

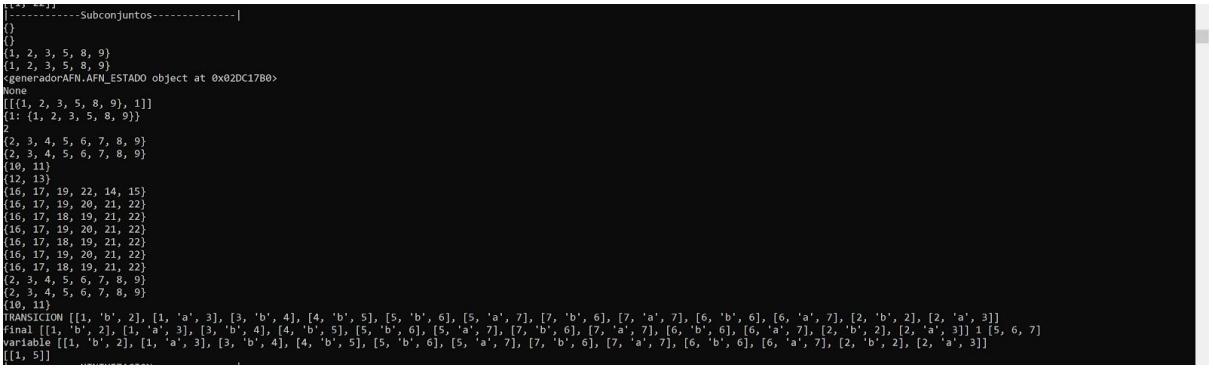
Su Entrada es una expresión regular r, se utilizó por lo tanto al archivo que contiene los diferentes operadores , es importante notar que se trabajó con el archivo generadorAFN dentro de esta misma se encuentran los diferentes operadores siendo concatenar, union, variable kleene, cerradura positiva. Para el funcionamiento de este proyecto se trabajó con los estados iniciales, finales y las transformaciones, las cuales fueron agregadas a las funciones de graficar que recibían un arreglo de los estados iniciales y finales y todas sus transformaciones. Con el objetivo de mantener que N(r) tiene al menos el doble de estados , mantener un estado inicial o múltiples estados de aceptación y su respectiva transición en un símbolo a dos transiciones

salientes. Armando así un Autómata Finito. A continuación se muestra los estados de transición dentro del proyecto:



Obtenido del texto:
AFN
ESTADOS: [6, 'E', 1, 5, 3, 2, '0', 4, 7, 11, 9, 8, '1', 10]
SIMBOLOS: [7, 2, 6, 4, 3, 5, 8, 12, 10, 9, 11]
INICIO: 1
ACEPTACION: 12
TRANSICION: [[6, 'E', 7], [1, 'E', 2], [1, 'E', 6], [5, 'E', 6], [5, 'E', 2], [3, 'E', 4], [2, '0', 3], [4, '0', 5], [7, 'E', 8], [7, 'E', 12], [11, 'E', 12], [11, 'E', 8], [9, 'E', 10], [8, '1', 9], [10, '1', 11]]

2.La siguente parte se encuentra en el archivo AFD_por_AFN.py y AFD.py. AFD recibió las transiciones y sus estados inicial y final . Este se encarga de construir una tabla de transición la cual es la clase DTran, en donde cada para estado D es un conjunto de estados del AFN. Dentro de los cálculos básicos para poder realizar el mismo se tienen 2 diferentes operaciones: las cuales son mover el cual es el conjunto de estados del AFN para los cuales hay una transición sobre el símbolo de entrada a y la cerradura el cual es el conjunto de estados del AGN a los que se puede llegar desde el estado s del AFN.



3 . El AFD Directo Fue la conversión directa de una expresión regular a un AFD en donde se mensajean por textos con el menú principal. Para este mismo se construyó una clase de Árbol Sintáctico y una clase de Nodos en donde se calcula el anulable , la primera posición, la última posición y la siguiente posición. Dentro de la función de Convertir_a_AFD se manejo el conjunto de estados del AFD D y del Dtran , la funcion de transicion para D. Los estados D son estados de posiciones en T. El estado inicial es la primera posición en donde el nodo es la raíz de las transiciones. Los estados de aceptación son los que contienen la posición para el

símbolo de marcador final. Dentro de los operadores de * y + se determinó el anulable como verdadero.

4. Por último, es el algoritmo de minimización este se encuentra en AFD.py. Este recibe el AFD con su conjunto de estados, alfabeto y como siempre su estado inicial y final. Se inició con una partición inicial en dos grupos , manejando los estados de aceptación y no aceptación. Se aplicó un For para cada grupo en donde particionamos en subgrupos de forma que dos estados se encuentren en el mismo subgrupo y sólo si para todos los símbolos de entrada a , los estados s y t tienen transiciones sobre a hacia estados en el mismo grupo . Se se llega a realizar una intersección entonces sigue al siguiente paso si no vuelve a entrar dentro del for. Dentro de esto se elige un estado en cada grupo final como representante para el grupo, marcando así los estados del AFD como el mínimo número de estados.Por lo tanto entra en un while que mire la longitud de los estados sin revisar realizando pop cuando lo encuentre conveniente.