**Manual Técnico**

**INTRODUCCIÓN:**

Autómata es un programa desarrollado en Python haciendo uso de la librería **Tkinter** para la creación de interfaces gráficas, que le permite al usuario ingresar autómatas finitos determinísticos y no determinísticos para su posterior simplificación y conversión de ser necesario, la verificación de secuencias de símbolos de entrada, además de la visualización del autómata en forma de diagrama de burbuja. La estructura del programa es sumamente sencilla, consta de una clase **automata\_finito**, que sirve para representar a los autómatas dentro del programa y que contiene los métodos de simplificación y conversión que serán explicados más a detalle en este manual. La otra clase, que es **programa** sirve para ejecutar la interfaz gráfica de Autómata.

Para la representación en forma de diagrama de burbuja se utiliza una librería de Python que se llama pythomata.

**DESARROLLO:**

Esta aplicación fue desarrollada en Visual Studio Code en el lenguaje de programación Python, para sentar las bases operativas de la misma, se crearon dos clases distintas con diferentes funcionalidades las cuales son **programa** y **automata\_finito**. Para un mejor entendimiento vamos a definir cada una:

* **Archivo programa.py:** es el archivo principal en el que nos encontramos con una pequeña interfaz gráfica. Las funciones que contiene hacen referencia a los comandos de algunos botones y a las acciones que llenan las distintas ventanas que componen la interfaz de autómata. Algunas de las funciones más importantes que componen este archivo son: *iniciar\_programa*, *muestra\_diagrama* y *crea\_tabla\_tr*. Estas funciones se detallarán más adelante en este manual.
* **Clase automata\_finito.py:** esta clase es la representación de un autómata finito determinístico para el programa. Tal y como se espera, se compone de un constructor en el que se indican los atributos que tendrán las instancias de esta clase, estos atributos son los siguientes: estados, símbolos, estado\_inicial, estado\_final (puede ser una lista de estados) y finalmente un diccionario para las transiciones. Además de esto, en la clase automata\_finito se encuentran una variedad de funciones que hacen que el programa pueda simplificar un AFD o convertir un AFND a AFD. Las funciones más importantes de esta clase son las siguientes: *estados\_equivalentes*, *estados\_extraños*, *simplificar\_afd* y *convertir\_afnd.*

Al comenzar la ejecución del programa se solicitan los estados que componen al autómata que se desea tratar. Se pide en el programa que se ingresen los estados separados por coma para más adelante aplicar la función Split con el parámetro coma (,). En la misma ventana, se solicita que se ingresen los símbolos de entrada del autómata siguiendo el mismo patrón de antes. En la siguiente ventana, haciendo uso de la información ingresada, se solicita que se seleccione un estado inicial y los estados finales o de aceptación.

Una vez hecho esto, se abre la ventana principal del programa. En esta ventana se ingresan las transiciones del autómata en una matriz que se genera en base a la información recolectada en las dos ventanas anteriores. Ahora, una vez terminado todo el proceso, el usuario puede darle al botón “CONVERTIR / SIMPLIFICAR”, que tiene el comando iniciar\_programa:



Función iniciar\_programa()

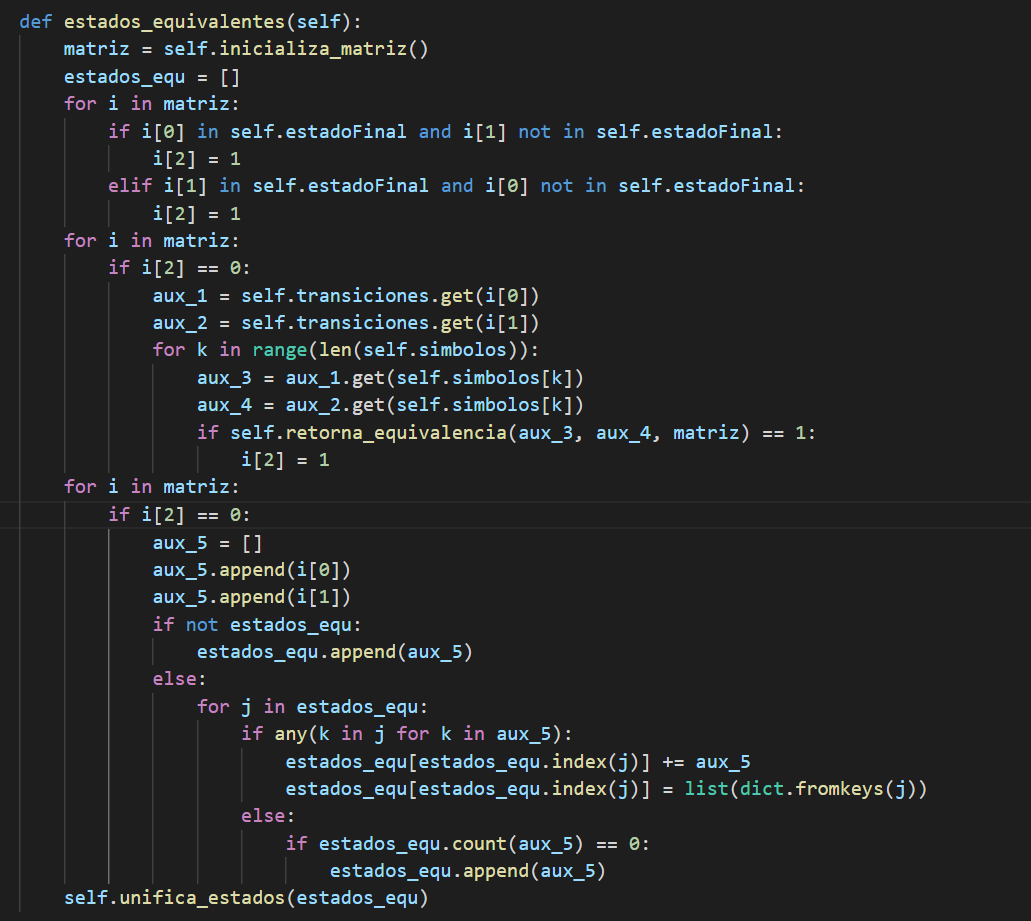
La función iniciar\_programa es la que inicia con toda la lógica de la aplicación. Lo primero que contiene es la llamada a los atributos globales de programa.py o en otras palabras toda la información que el usuario ha ingresado hasta ahora. Luego, se instancia un nuevo autómata finito y se le asignan los atributos. Después de tener conformado el autómata finito se tiene la condición que determina si es determinístico o no y si es necesario convertir o simplificar. Primero hablaremos del proceso de simplificación debido a que es un poco más extenso que el de convertir, ya que internamente debe hacer dos cosas diferentes. En primer lugar, se encuentran los estados extraños utilizando la función que se enseña a continuación:

Texto

Descripción generada automáticamente

El proceso que sigue esta función es bastante sencillo, se parte del estado inicial del autómata y a partir del mismo se va llenando una lista con las transiciones de ese estado únicamente, hasta que ya no se agreguen estados nuevos. Los que quedaron por fuera son inaccesibles por lo tanto se termina el ciclo y se filtran los estados extraños comparando con una lista auxiliar que solo contiene los estados que sí se visitaron.

Ahora bien, una vez se encuentran los estados extraños, es necesario encontrar los estados equivalentes. Un proceso un poco más complejo, que para este caso utiliza la matriz de Myhill-Nerode que sirve para hallar los estados equivalentes de un autómata por medio de una matriz dispersa.



Lo primero que hace la función es inicializar la matriz de Myhill-Nerode que almacena los estados equivalentes como tripletas de la siguiente forma (estado1, estado2, equivalencia). El primer ciclo for de la función determina que los estados que son de aceptación no pueden ser equivalentes con los estados que no lo son y marca la equivalencia con un 1. El segundo ciclo for, lo que hace es buscar en las demás tripletas de la matriz MN aquellas cuya equivalencia sea un 0, se verifican las transiciones de cada estado y si algún par contiene la equivalencia 1, se cambia la equivalencia de la tripleta actual. Para más detalles de como funciona la matriz de Myhill-Nerode para la búsqueda de estados equivalentes en un autómata finito siga el [siguiente enlace](https://www.youtube.com/watch?v=UiXkJUTkp44).

Lo siguiente es enseñar la función de convertir\_afnd, que es bastante similar a la de estados extraños con la diferencia de que esta función debe unificar las transiciones de un estado en caso de que el mismo tenga más de una transición para el mismo símbolo de entrada:

Texto

Descripción generada automáticamente

La función convierte a AFND es una especie híbrido de las dos funciones anteriores. Lo primero que hace es llamar a otra función limpia listas, que se encarga de que en las transiciones del autómata cuando una transición sea una lista e.j. [C, D], se reescriba simplemente como CD. De este modo es más sencillo continuar con le procedimiento. Se establece un estado padre que es el estado actual al que se le evalúan las transiciones, se declara un diccionario vacío que se convertirá en las transiciones resultantes de la conversión, y finalmente se declara una lista vacía que contendrá los nuevos estados del autómata.

Dentro del ciclo **while** hay dos caminos, el primero es el que se toma cuando el estado padre existe en las transiciones originales y el segundo es cuando ese no es el caso y se debe crear un estado nuevo con transiciones nuevas. Para el primer camino, solo basta replicar el estado y las transiciones originales. Pero para el segundo camino es necesario concatenar las transiciones de los estados originales que componen el nuevo estado. La llamada a la función corrige estado es para evitar que se permitan estados anagramas de otros o que un estado se componga del mismo estado varias veces, además de ordenar los estados en orden ascendente. Finalmente, actualiza los estados con una función que se encarga de actualizar los estados de aceptación y el estado inicial en caso de que se hayan visto afectados por todo el proceso.