Librería AUTOM

Juan Esteban Cepeda Baena[1]

[1] Estudiante de Ciencias de la Computación de la Universidad Nacional de Colombia.

jecepedab@unal.edu.co

Abstract. *This paper presents an introduction to the general characteristics of the AUTOM library, which allows the user to build finite automatas and execute various functions with these objects, in particular, deciding whether a string is part of the language accepted by an automaton or not. In addition, the library installation and usage requirements are stated, as well as the source code license.*

Resumo. *En este trabajo se presenta una introducción a las características generales de la librería AUTOM, la cual permite construir autómatas finitos y ejecutar diversas funciones con estos objetos, en particular, decidir si una cadena hacer parte del lenguage aceptado por un autómata o no. Adicionalmente, se enuncian los requisitos de instalación y uso de librería, así como la licencia del código fuente.*

1. Licencia de Uso

MIT.

2. Introducción

La librería Autom consiste en un conjunto de algoritmos que permiten construir autómatas finitos y realizar operaciones con ellos, permitiendo computar todos los procesamientos de una cadena o listas de cadenas, así como sus características, y guardando la información dinámicamente en los archivos requeridos por el usuario.

3. Requisitos previos

Algunos de los requisitos previos para usar la librería son:

a) **Conocimientos del usuario:** Se presume que el usuario está familiarizado con la teoría de autómatas, en particular, con el funcionamineto de los autómatas deterministas, no-deterministas y los no-deterministas con transiciones lambda. Adicionalmente, el usuario debe estar familiarizado con el lenguage Python y el entorno de ejecución *Jupyter Notebook*.

b) **Requisitos técnicos:** El código de AUTOM está escrito en *Jupyter Notebook*, lo que significa que el formato del archivo principal es *.ipynb*. Por lo anterior, es necesario que el usuario tenga instalado el entorno de ejecución *Jupyter Notebook*; opcionalmente, puede utilizarse *Colaboraty*, no obstante, las funciones de guardado y carga de archivos quedan inhabilitadas con esta opción.

4. Instalación y Configuración

Una vez descargada la carpeta de AUTOM, el usuario debe abrirla utilizando el entorno de ejecución *Jupyter Notebook*. Una vez completado el paso anterior, se debe acceder al archivo *AutomLib.ipynb*, en el cual está alojado el código de AUTOM. Una vez dentro del archivo, se deben ejecutar todos los bloques de código. Adicionalmente, omo *Jupyter Notebook* es un entorno de ejecución interactivo, el usuario puede modificar algunos de los autómatas de ejemplo ubicados en los bloques posteriores a la definición de las clases AFD, AFN y AFNLambda.

Las librerías necesarias para utilizar AUTOM son: numbers, math, numpy y string, las cuales vienen instaladas por defecto en Python.

5. Clase Autómata

La clase autómata incorpora la estructura y funcionalidad básica de los autómatas a partir de un conjunto de parámetros dados por el usuario (alfabeto, estado inicial, estados, estados de aceptación, función delta), o a partir de la ruta de un archivo (el formato del archivo se especifica más adenlante). De esta clase, se derivan tres mas:

5.1. Clase AFD

Esta clase incorpora el funcionamiento de un autómata finito determinista. A continuación, se presentan las funciones más importantes de este tipo de autómata:

a) **Constructor(alfabeto, estados, estadoInicial, estadosAceptacion, delta, nombreArchivo)** de la clase para inicializar los atributos. En caso en que el autómata no sea completo (es decir, la función de transición no esté completamente definida) la librería retorna un error. Si el argumento **nombreArchivo** es diferente de vacío, se ignora el resto de parámetros ingresados por el usuario y se intenta inicializar el autómata mediante archivo.
b) **procesarCadena(cadena)**: procesa la cadena y retorna verdadero si es aceptada ó falso si es rechazada por el autómata.
c) **procesarCadenaConDetalles(cadena)**: realiza lo mismo que el método anterior pero aparte imprime los estados que va tomando al procesar cada símbolo.
d) **procesarListaCadenas(listaCadenas,nombreArchivo, imprimirPantalla)**: procesa cada cadenas con detalles pero se impremen en un archivo cuyo nombre es nombreArchivo; si este es inválido se asigna un nombre por defecto. Además, dependiendo del valor del Boleano imprimirPantalla, se imprime por pantalla el resultado anterior.

5.2. Clase AFN

Esta clase incorpora el funcionamiento de un autómata finito no-determinista. A continuación, se presentan las funciones más importantes de este tipo de autómata:

a) **Constructor(alfabeto, estados, estadoInicial, estadosAceptacion, delta, nombreArchivo)** de la clase para inicializar los atributos. Si el argumento **nombreArchivo** es diferente de vacío, se ignora el resto de parámetros ingresados por el usuario y se intenta inicializar el autómata mediante archivo.
b) **procesarCadena(cadena)**: procesa la cadena y retorna verdadero si es aceptada y falso si es rechazada por el autómata.
c) **procesarCadenaConDetalles(cadena):** realiza lo mismo que el método anterior pero aparte imprime los estados que va tomando al procesar cada símbolo de uno de los procesamientos que lleva a la cadena a ser aceptada.
d) **computarTodosLosProcesamientos(cadena, nombreArchivo)**: Imprimir cada uno de los posibles procesamientos de la cadena indicando de qué estado a qué estado pasa al procesar cada símbolo e indicando si al final de cada procesamiento se llega a aceptación o rechazo. Adicionamente, llena una lista de todos procesamientos de aceptación, una lista de todos los procesamientos abortados y una lista de todos los procesamientos de rechazo. Por último, guarda los contenidos de estas listas cada una en un archivo (cuyos nombres son nombreArchivoAceptadas.txt, nombreArchivoRechazadas.txt y nombreArchivoAbortadas.txt) y además las imprime en pantalla juto con el número de procesamientos realizados.
e) **procesarListaCadenas(listaCadenas,nombreArchivo, imprimirPantalla)**: procesa cada cadenas con detalles pero los resultados se imprimen en un archivo cuyo nombre es nombreArchivo; si este es inválido se asigna un nombre por defecto. Adicionalmente, dependiendo del valor del boleano imprimirPantalla, se imprime o no el resultado anterior en pantalla. En caso en que el usuario desee ver los resultados en pantalla, se sigue la siguiente estructura:
    - Cadena
    - Sucesión de parejas (estado, símbolo) de cada paso de un procesamiento de aceptación (si lo hay, si no el más corto de rechazo)
    - Número de posibles procesamientos
    - Número de procesamientos de aceptación
    - Número de procesamientos abortados.número de procesamientos de rechazo
    - Sí o no dependiendo de si la cadena es aceptada o no.

5.3. Clase AFNLambda

Esta clase incorpora el funcionamiento de un autómata finito no-determinista con transiciones lambda. Es importante señalar que el símbolo λ en el marco de esta teoría, se representa con el símbolo $ en Autom. A continuación, se enuncian las funciones más importantes de este tipo de autómata:

a) **Constructor(alfabeto, estados, estadoInicial, estadosAceptacion, delta, nombreArchivo)** de la clase para inicializar los atributos. Si el argumento **nombreArchivo** es diferente de vacío, se ignora el resto de parámetros ingresados por el usuario y se intenta inicializar el autómata mediante archivo.
b) **calcularLambdaClausura(estado)**: calcula y retorna la λ-Clausura de un estado.
c) **calcularLambdaClausura(conjuntoEstados)**: calcula y retorna la λ-Clausura de un conjunto de estados.
d) **procesarCadena(cadena)**: procesa la cadena y retorna verdadero si es aceptada y falso si es rechazada por el autómata.

e) **procesarCadenaConDetalles(cadena):** realiza lo mismo que el método anterior pero aparte imprime los estados que va tomando al procesar cada símbolo de uno de los procesamientos que lleva a la cadena a ser aceptada.

f) **computarTodosLosProcesamientos(cadena, nombreArchivo)**: Imprimir cada uno de los posibles procesamientos de la cadena indicando de qué estado a qué estado pasa al procesar cada símbolo e indicando si al final de cada procesamiento se llega a aceptación o rechazo. Adicionamente, llena una lista de todos procesamientos de aceptación, una lista de todos los procesamientos abortados y una lista de todos los procesamientos de rechazo. Por último, guarda los contenidos de estas listas cada una en un archivo (cuyos nombres son nombreArchivoAceptadas.txt, nombreArchivoRechazadas.txt y nombreArchivoAbortadas.txt) y además las imprime en pantalla juto con el número de procesamientos realizados.

g) **procesarListaCadenas(listaCadenas,nombreArchivo, imprimirPantalla)**: procesa cada cadenas con detalles pero los resultados se imprimen en un archivo cuyo nombre es nombreArchivo; si este es inválido se asigna un nombre por defecto. Adicionalmente, dependiendo del valor del boleano imprimirPantalla, se imprime o no el resultado anterior en pantalla. En caso en que el usuario desee ver los resultados en pantalla, se sigue la siguiente estructura:

   - Cadena
   - Sucesión de parejas (estado, símbolo) de cada paso de un procesamiento de aceptación (si lo hay, si no el más corto de rechazo)
   - Número de posibles procesamientos
   - Número de procesamientos de aceptación
   - Número de procesamientos abortados.número de procesamientos de rechazo
   - Sí o no dependiendo de si la cadena es aceptada o no.

6. Archivos generados por AUTOM

Como se mencionó en la sección anterior, algunas funciones de AUTOM generan archivos con la información requerida por el usuario. Para guardar la información generada por la función **computarTodosLosProcesamientos**, AUTOM incorpora dos carpetas que pueden ser encontradas en la ruta principal de la librería, a saber: **resultadosAFN** y **resultadosAFNLambda** en donde se alojan los archivos requeridos. Por otro lado, la información generada por la función **procesarListaCadenas** se guarda directamente en la ruta principal de la librería, bajo los nombres *resultadosAFN.txt, resultadosAFN.txt* y *resultadosAFNLambda.txt*, en caso en que el usuario no especifique el nombre del archivo, u ocurra un error con el nombre del archivo digitado.

7. Cargar autómatas mediante archivo

Como se mencionó en la sección de autómatas, AUTOM permite cargar autómatas mediante archivos de texto de tipo *.txt*. La estructura de los archivos de texto para inicializar autómatas es la siguiente:

```
#!nfae

#alphabet
a-b
$

#states
Q0
Q1
Q2

#initial
Q0

#accepting
Q2

#transitions
Q0:a>Q0
Q0:b>Q1
Q1:b>Q1
Q1:$>Q2
Q1:a>Q0
Q2:b>Q2
```

A continuación, se explica al detalle las características del archivo.

a) Para indicar el tipo de autómata que se desea inicializar, es importante tener en cuenta lo siguiente:
   - **dfa!**: autómata determinista.
   - **nfa!**: autómata no-determinista.
   - **nfae!**: autómata no-determinísta con transiciones lambda.

b) En la sección **alphabet** se describe cada símbolo del alfabeto del autómata., el rango de caracteres - o caracteres separados por saltos de línea. Un rango de caracteres puede antecer o proceder a un caracter y viceversa.

c) En la sección **states** se denominan cuáles son los posibles estados que puede recorrer el autómata.

d) En la sección **initial** se define cuál será el estado inicial. El estado contenido en esta sección debe estar en la sección de conjunto de estados y no es estricto que sea el primer elemento en esta sección.

e) En la sección **accepting** se define cuáles son los estados de aceptación, cada estado descrito en esta sección debe estar en la sección de conjunto de estados.

f) En la sección **transitions** se definen a transiciones del automata. El estado del que se parte debe estar precedido por : , posteriormente debe estar el símbolo por el cual se hace la transción seguido por un $>$ y finalmente el estado o estados de llegada separados por comas $(,)$. Por ejemplo,

$$state_i : symbol > next_{state_0}; \ldots; next_{state_n}$$

8. Clase Pruebas

Esta clase permite al usuario interactuar con una interfaz gráfica de AUTOM. Entre algunas de sus funciones se encuentran probar inicializar y ejecutar funciones con

autómatas deterministas, no deterministas y no deterministas con transiciones lambda. Adicionalmente, permite cargar autómatas mediante archivo. A continuación, se listan sus funciones principales:

a) **probarAFD(AFD):** Crea autómatas AFD, procesa cadenas con y sin detalles, procesa listas de cadenas, genera archivos.
b) **probarAFN(AFN):** Crea autómatas AFN, procesa cadenas mostrando solo un procesamiento de aceptación, procesa cadenas mostrando todos los procesamientos posibles, consulta los procesamientos de aceptación, abortados y de rechazo, procesar listas de cadenas, generar archivos.
c) **probarAFNLambda(AFNLambda):** Crea autómatas AFN-λ, calcula la λ-clausura de un estado, calcula la λ-clausura de un conjunto de estados, procesa cadenas mostrando solo un procesamiento de aceptación, procesa cadenas mostrando todos los procesamientos posibles, consulta los procesamientos de aceptación, abortados y de rechazo, procesa listas de cadenas, genera archivos.
d) **main():** invoca a los otros para que puedan ser comentados fácilmente y poder escoger cuál se va a probar.

9. Otras clases

9.1. Clase Procesador Archivos

Esta clase permite crear objetos para procesar la información de los archivos que codifican las características de un autómata. En particular, se encarga de determinar el alfabeto, estados, estado inicial, estados de aceptación, y función delta de transiciones entre estados, a partir de la ruta en donde está almacenado el archivo. Finalmente, retorna un objeto de la forma $M = (\Sigma, Q, q_0, F, \delta)$.

9.2. Clase Procesador Cadenas

Esta clase es la encargada de formatear y calcular características de los distintos procesamientos de una cadena por parte de un autómata. Algunas de sus funciones son: verificar si un procesamiento es de aceptación o no, verificar si al menos un procesamiento terminó en un estado de aceptación, formatear la información del procesamiento de una cadena para luego imprimir por pantalla la secuencia de estados e instrucciones, guardar información de los procesamientos en distintos archivos, obtener información de los procesamientos (número de trayectorias totales, de aceptación, rechazo o abortadas, etc), obtener los procesamientos más cortos, entre otras funcionalidades.

10. Más información

Para más información, contactar al correo **jecepedab@unal.edu.co**.