Autómata Finito No Determinista

Martín Alejandro Castro Álvarez

martincastro.10.5@gmail.com

https://github.com/MartinCastroAlvarez/automata-python

Universidad Internacional de Valencia (VIU)

Calle Pintor sorolla, 21 46002, Valencia (España)

Abril 2024

Resúmen. Este trabajo se centra en el diseño e implementación de un Autómata Finito No Determinista (AFN) utilizando Python para el reconocimiento de patrones en direcciones de correo electrónico. Se exploran los principios teóricos de autómatas y lenguajes formales, permitiendo al AFN procesar múltiples transiciones y estados de manera simultánea. La implementación cubre todos los componentes fundamentales de un AFN, desde un alfabeto diverso hasta un conjunto de estados y transiciones bien definidos, demostrando ser una herramienta eficiente en la validación de datos.

Keywords: Autómatas Finitos No Deterministas, Procesamiento de Lenguajes, Python, Validación de Correo Electrónico, Transiciones ε, Metodología de Desarrollo Dirigido por Pruebas.

1. Introducción

1.1. Objetivo

El objetivo principal de este trabajo es diseñar e implementar un Autómata Finito No Determinista (AFN) utilizando Python para el reconocimiento eficiente de patrones específicos en cadenas de texto que representan direcciones de correo electrónico. Este objetivo involucra aplicar los fundamentos teóricos de autómatas y lenguajes formales para simular la capacidad de un AFN de procesar y reconocer patrones estando en múltiples estados simultáneamente.Se espera que el AFN implementado sea capaz de manejar transiciones ε y múltiples transiciones posibles para un mismo símbolo de entrada, proporcionando un enfoque más flexible y poderoso en comparación con los autómatas finitos deterministas.

1.2. Planteamiento del Problema

El procesamiento de textos y lenguajes forma una parte integral de la informática, y la capacidad para identificar y manipular patrones dentro de los datos es una habilidad crítica en este ámbito. El desafío radica en la necesidad de sistemas que puedan manejar la ambigüedad y la multiplicidad de posibles interpretaciones que caracterizan a los lenguajes naturales y de programación. Los métodos tradicionales, que a menudo se basan en autómatas finitos deterministas (AFD), enfrentan limitaciones debido a su incapacidad para abordar varias posibles rutas de procesamiento simultáneamente. Por lo tanto, el problema que abordamos es la creación de un sistema basado en AFN que supere estas limitaciones, demostrando así

la utilidad y la eficiencia de este enfoque en comparación con las técnicas convencionales, y cómo puede ser extendido a problemas más generales y complejos en el campo de la informática.

1.3. Solución Propuesta

El **alfabeto** Σ aceptado por el AFN son todas las letras del alfabeto, números, y símbolos en el estándar UTF-8.

El conjunto de **estados** Q en los que se puede encontrar el AFN son $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, a_8, a_F\}$

El **estado inicial** del AFN se define como q_0.

El único **estado de aceptación** del AFN se define como q_7.

La **tabla de transiciones** se encuentra definida por el grafo de la Fig. 1.

2. Antecedentes Teóricos

2.1. Alfabetos

Un **Alfabeto** Σ es cualquier conjunto finito y no vacío de símbolos $\{c_0, c_1, c_3 \ldots\}$.

La longitud del alfabeto |Σ| corresponde a la cantidad de símbolos distintos del alfabeto Σ.

La **potencia de un alfabeto** Σ^k es el conjunto de todas las palabras de tamaño k conformada por la combinación de símbolos Σ, con $k \geq 0$.

2.2. Palabras

Una **palabra** ω es cualquier subconjunto ordenado de símbolos perteneciente al Σ.

Una **palabra vacía** ε es la palabra que contiene cero símbolos.

La **concatenación** $v\omega$ es la operación de juntar las 2 palabras.

Se llama |ω| a la **longitud** de la palabra ω y corresponde a la cantidad de símbolos que contiene. La longitud de la **palabra vacía** |ε| es 0.

La **potencia de una palabra** ω^k es una palabra que resulta de concatenar k veces la palabra ω, con $k \geq 0$.

2.3. Lenguaje

El conjunto de todas las posibles combinaciones de palabras de cualquier longitud es Σ^*. Se denomina **lenguaje completo.**

El conjunto de todas las posibles combinaciones de palabras de longitudes mayores a 0 es Σ^+.

Un **lenguaje** L es un subconjunto de Σ^*.

El **lenguaje vacío** se denomina Ø.

2.2. Autómata Finito Determinista (AFD)

Un **Autómata Finito Determinista (AFD)** es un quíntuplo ordenado de la forma: $AFD = (Q, \Sigma, \delta, q_0, F)$ tal que Q es un conjunto no vacío de estados, Σ es un alfabeto, $\delta: Q \times \Sigma \rightarrow Q$ es una función de transición de estado cuyo dominio es toda combinación de estado y alfabeto, y cuya imagen es el nuevo estado del autómata, q_0 es el estado inicial y F es el estado de aceptación ó final.

2.3. Autómata Finito No Determinista (AFN)

Un **Autómata Finito No Determinista (AFND)** es aquel AFN en el que la función de transición δ puede conducir a múltiples estados de Q, a ningún estado, y puede contener transiciones ε en las que, de forma aleatoria, el estado del autómata cambia hacia otro.

Para cada AFND, existe algún AFN equivalente, es decir, que reconoce el mismo lenguaje. Sin embargo, podría ocurrir que, si el AFND posee n estados, el AFD posea $2\hat{}n$.

3. Metodología

3.1. Implementación en Python

Utilizamos Python por su simplicidad sintáctica y la eficacia de sus estructuras de datos, lo que facilita la representación de estados y transiciones del AFN. Por otra parte, el autómata se ejecuta mediante el comando (1) en una interfaz de línea de comando.

$$python3\ emailAfn.py" < cadena > " \quad (1)$$

3.2. Estructuras de Datos

3.2.1 Programación Orientada a Objetos

La solución hace uso de un **diseño orientado a objetos** que facilita la representación y manipulación de autómatas finitos no deterministas (AFN). La estructura de datos central para este propósito incluye las clases Alphabet, State, Symbol, y TransitionTable.

La clase **Alphabet** es un conjunto de símbolos permitidos que conforman las direcciones de correo electrónico. Esta clase garantiza que solo se reconozcan los caracteres válidos dentro de los strings evaluados.

Por otra parte, la clase **Symbol** representa un carácter dentro del alfabeto y se utiliza para determinar las transiciones entre estados en el AFN.

La clase **State** representa nodos en el autómata que encapsulan los diferentes estados posibles durante la evaluación de una cadena.

Cada instancia de State está vinculada con otras a través de la clase **TransitionTable**, que dicta cómo se mueve el AFN de un estado a otro basado en el símbolo de entrada actual.

Finalmente, **AFN** es la clase que reúne estos componentes. Inicia en un estado definido, procesa la cadena de entrada símbolo por símbolo mediante la tabla de transiciones y determina si la cadena es aceptada o rechazada basada en los estados de aceptación.

3.3. Transiciones ε

No es necesario utilizar transiciones especiales que permiten al autómata pasar de un estado a otro sin consumir ningún símbolo de entrada. En la práctica, esto significa que, a pesar de que el código en Python lo permita, no es necesario ninguna transición hacia más de un estado simultáneamente.

3.4 Test Driven Development (TDD)

Adoptamos el TDD para garantizar que cada función del AFN se desarrolla a partir de pruebas específicas, asegurando que todos los casos de uso sean considerados y validados. Las pruebas se ejecutan con el script en BASH que se ejecuta mediante (2).

$$./tests/testEmailAfn.sh \quad (2)$$

La Fig. 2 muestra la lista de casos que se espera sean aceptados por el autómata, mientras que la Fig. 3 muestra la lista de casos que se espera que sean rechazados.

4. Resultados

4.1. Ejecución de las Pruebas

La fase de pruebas del Autómata Finito No Determinista (AFN) para el reconocimiento de correos electrónicos demostró su efectividad al aceptar correctamente todas las direcciones válidas, incluyendo aquellas con estructuras y caracteres especiales, y rechazar las inválidas, validando así la precisión del AFN y su capacidad de discriminar formatos incorrectos. Estos resultados (Ver Fig. 4) avalan la utilización del AFN como herramienta fiable para la validación de emails en sistemas informáticos y apoyan su potencial para futuras aplicaciones en análisis léxico y procesamiento de lenguajes naturales.

4.2 Análisis de los Resultados

El análisis de los resultados de las pruebas realizadas con el AFN demuestra una adecuación efectiva entre la implementación y los requisitos teóricos para la validación de direcciones de correo electrónico. La tasa de éxito del 100% en la identificación correcta de las direcciones válidas indica una comprensión profunda y una aplicación precisa de los conceptos de AFNs. Además, la consistencia en el rechazo de las entradas inválidas refuerza la confiabilidad del sistema en contextos reales, donde el filtrado de información incorrecta es tan importante como la aceptación de la correcta.

4.3 Reflexión y Hallazgos

A lo largo del desarrollo de este proyecto, se me ha presentado la valiosa oportunidad de explorar a fondo las teorías que sustentan los autómatas finitos, tanto deterministas (AFD) como no deterministas (AFN). Esto no solo ha solidificado mi entendimiento teórico sino que también ha proporcionado un terreno fértil para la aplicación práctica de estos conceptos en la elaboración de algoritmos y sistemas complejos.

Adicionalmente, la lectura del texto "Reinforcement Learning, second edition: An Introduction" de Sutton y Barto ha sido un complemento iluminador para este estudio: Un autómata que aprende de su entorno puede ser conceptualizado como una entidad que perfecciona su función de transición a través de la retroalimentación obtenida de sus interacciones con el ambiente físico, un proceso análogo al aprendizaje reforzado en sistemas inteligentes. Este paralelismo no sólo ha enriquecido mi perspectiva respecto a la aplicabilidad de la teoría de la computación en aprendizaje automático sino que también ha reforzado mi

convicción en la sinergia que existe entre estos dos dominios, pudiendo ser empleada conjuntamente para afrontar y solucionar desafíos complejos del mundo computacional.

5. Referencias

[1] De Castro Korgi, R. Teoría de la Computación. Lenguajes, autómatas, gramáticas.

[2] Sutton, R. S., & Barto, A. G. Reinforcement Learning, Second Edition: An Introduction (Adaptive Computation and Machine Learning series).

6. Anexo

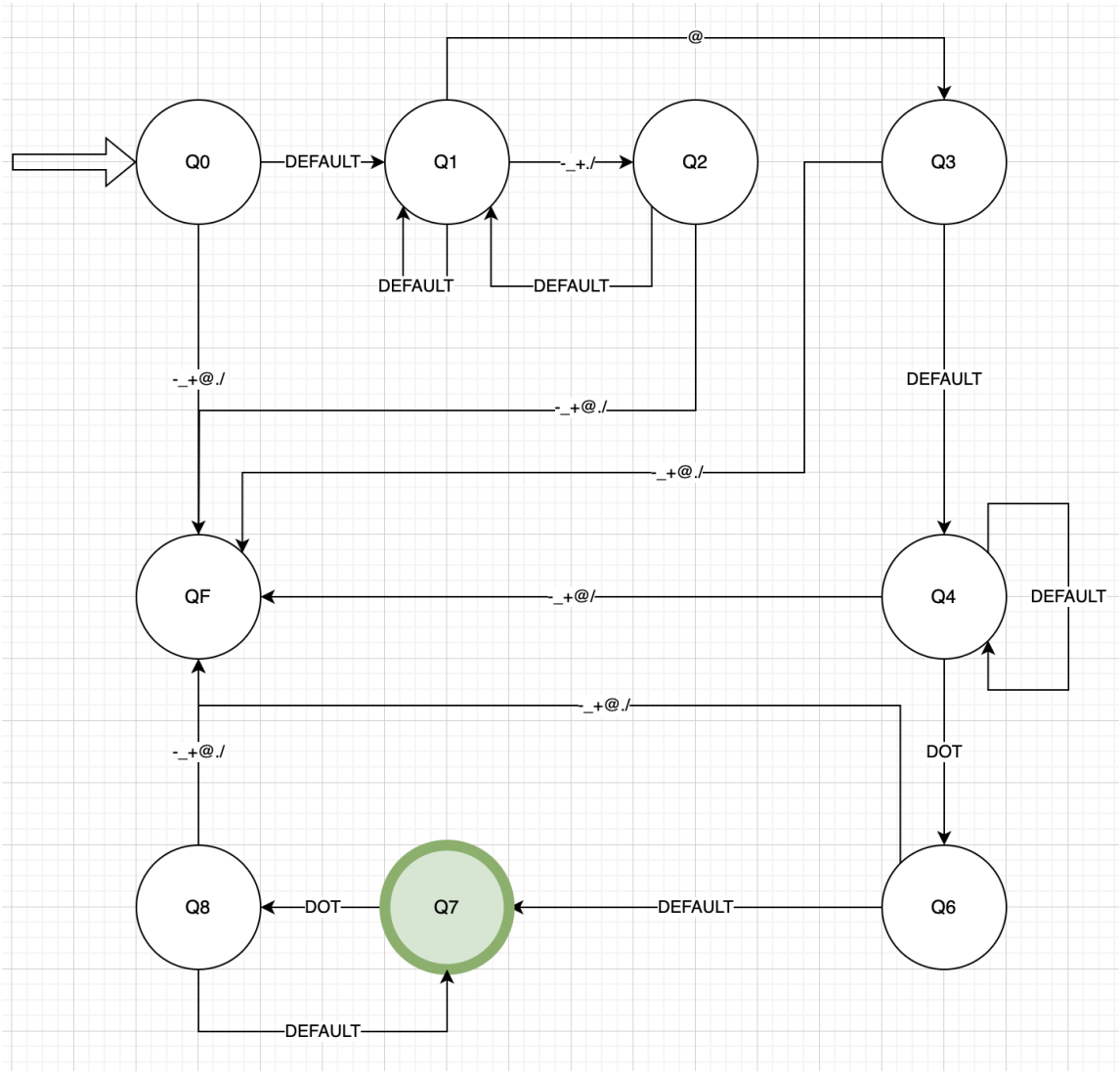

Fig. 1: Diagrama de transiciones del AFN

#1	martincastro@gmail.com
#2	martincastro105@gmail.com
#3	martin-castro.10.5@gmail.com
#4	martin_castro@mail.google.com
#5	martincastro.10.5@mail.google.com
#6	martincastro+1@gmail.com

#7	martincastro.10.5+1@gmail.com
#8	martincastro.10.5+1@gmail.com.es
#9	martin-castro.10.5+1@gmail.com.ar
#10	martincastro@gmail.net
#11	martin.castro@gmail.com
#12	martincastro@Gmail.com

Fig. 2: Tabla de casos de prueba satisfactorios.

#1	martincastro
#2	martincastrogmail.com
#3	martincastro@gmail
#4	martincastro@@gmail.com
#5	martincastro@gmail.com.
#6	.martincastro@gmail.com

#7	martin castro@gmail.com
#8	martin-castro/@gmail.com
#9	martincastro@/gmail.com
#10	martin_castro@gmailcom
#11	martincastro.@gmail.com
#12	martin--castro@gmail.com

Fig. 3: Tabla de casos de prueba no satisfactorios.

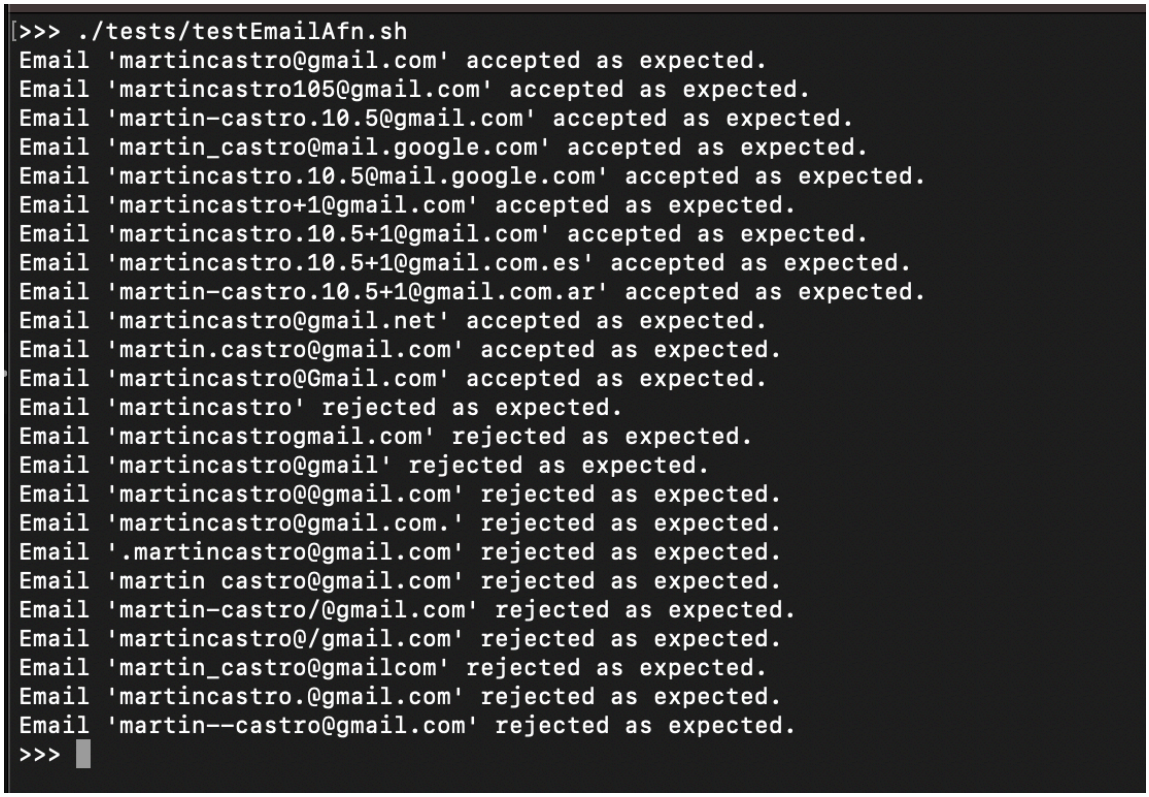

Fig. 4: Diagrama de transiciones del AFN propuesto como solución al problema.