Trabajo Práctico Integrador - Documentación

Facultad de Ingeniería

Cátedra:	Fundamentos Teóricos de la Informática
Alumnos:	Grilli Ignacio, Casteglione Matias
Docentes:	Moreno Leonardo, Morales Leonardo
Fecha:	24 de septiembre de 2025.

Índice

1: Introducción

Este proyecto soluciona la conversión y minimización de autómatas finitos no deterministas a autómatas finitos deterministas y su posterior reducción a autómatas mínimos. El objetivo es proveer una herramienta interactiva y programática para visualizar, analizar y transformar autómatas, facilitando el estudio y la aplicación de la teoría de lenguajes formales.

1.1: Fundamentos teóricos

- **AFND (Autómata Finito No Determinista):** Modelo de autómata donde, para un estado y símbolo de entrada, puede haber múltiples transiciones posibles (incluyendo transiciones épsilon).
- **AFD (Autómata Finito Determinista):** Modelo donde cada estado y símbolo de entrada tiene una única transición posible. No existen transiciones épsilon o lambda.
- **Autómata mínimo:** Es un AFD reducido, con el menor número posible de estados, que reconoce el mismo lenguaje.

2: Metodología

2.1: Estructura de Módulos

- **src/automaton.py:** Define la clase Automaton, que representa cualquier autómata (AFND, AFD, mínimo). Incluye métodos para copiar, obtener nombres legibles de estados y estadísticas.
- **src/parsing.py:** Funciones para leer y escribir autómatas en formato JSON y XML. Permite interoperabilidad y pruebas automáticas.
- **src/conversion.py:** Implementa los algoritmos de conversión AFND→AFD (algoritmo de subconjuntos) y minimización de AFD (Hopcroft).
- **src/visualization.py:** Utiliza matplotlib y networkx para dibujar los autómatas, mostrando estados, transiciones y composición.
- **src/gui.py:** Interfaz gráfica con Tkinter, permite cargar, visualizar, convertir, minimizar y guardar autómatas.
- **src/main.py:** Punto de entrada CLI y GUI, gestiona argumentos y flujo principal.

2.2: Algoritmos implementados

- **Conversión AFND a AFD:** Se utiliza el algoritmo de subconjuntos (powerset construction), que genera estados compuestos en el AFD a partir de conjuntos de estados del AFND, considerando cierres épsilon.
- **Minimización de AFD:** Se implementa el algoritmo de Hopcroft, que particiona los estados en bloques y los refina hasta obtener el conjunto mínimo de estados equivalentes.

2.3: Análisis de cada módulo

automaton.py

- Automaton: Constructor recibe estados, alfabeto, estado inicial, estados de aceptación, transiciones, tipo (DFA/NFA), nombre y composición de estados.
- copy: Devuelve una copia profunda del autómata.
- get_readable_state_name: Si el estado es compuesto, muestra su composición (útil para AFD).
- get_stats: Calcula estadísticas como número de estados, transiciones, transiciones épsilon, etc.

parsing.py

- parse_json_automaton: Lee un archivo JSON y construye un objeto Automaton. Normaliza estados y detecta si es DFA.
- parse_xml_automaton: Similar, pero para XML. Permite interoperabilidad con otras herramientas.
- automaton_to_json_dict / automaton_to_xml_element: Serializan el autómata para guardar o exportar.

conversion.py

- epsilon_closure: Calcula el cierre épsilon de un conjunto de estados (fundamental para AFND).
- move: Calcula el conjunto destino para un símbolo.
- nfa_to_dfa: Algoritmo de subconjuntos. Crea estados compuestos, calcula transiciones y determina estados de aceptación.
- remove_unreachable_states: Elimina estados inaccesibles.
- hopcroft_minimize: Algoritmo de minimización. Divide estados en bloques y construye el AFD mínimo.

visualization.py

- AutomatonVisualizer: Dibuja el autómata en un grafo dirigido, coloreando estados iniciales, de aceptación y mostrando nombres legibles. Las transiciones se agrupan y se muestran con etiquetas.

gui.py

- AutomatonGUI: Ventana principal con pestañas para información y visualización. Permite cargar archivos, convertir, minimizar y guardar resultados. Muestra estadísticas y composición de estados.

main.py

- main: Permite ejecutar el programa por CLI o GUI. Gestiona argumentos para entrada/salida, formatos, minimización y nombre del autómata.

2.4: Pseudocódigo: Conversión AFND a AFD

INICIO

1. Calcular el cierre-ε de los estados iniciales (estados alcanzables con transiciones vacías).

2. Crear el primer estado del DFA a partir de ese conjunto.

3. Mientras haya estados sin procesar en el DFA:

Para cada símbolo del alfabeto:

- Calcular los estados alcanzables (move + cierre-ε).
- Si es un nuevo conjunto de estados, agregarlo al DFA.
- Crear la transición correspondiente.

4. Eliminar estados inalcanzables en el DFA.

5. Minimizar el DFA con el algoritmo de Hopcroft:

Separar estados de aceptación y no aceptación.

- Dividir bloques hasta que no se puedan dividir más.
- Reconstruir el autómata minimizado.

FIN

2.5: Decisiones de Diseño

- Modularidad: Cada módulo tiene una responsabilidad clara, facilitando mantenimiento y pruebas.
- Visualización: Se priorizó la claridad visual, usando colores y nombres legibles.
- Compatibilidad: Soporte para JSON y XML.
- Interfaz amigable: GUI en español, mensajes claros y ayuda visual.

3: Casos de Prueba

Se incluyen ejemplos en la carpeta samples/:

Ejemplo pequeño (sample_nfa.json)

- AFND: 3 estados, transiciones épsilon.
- AFD generado: 4 estados, composición mostrada.
- AFD mínimo: 3 estados.

Paso a paso:

- Carga el AFND.
- Visualiza el grafo y las transiciones.

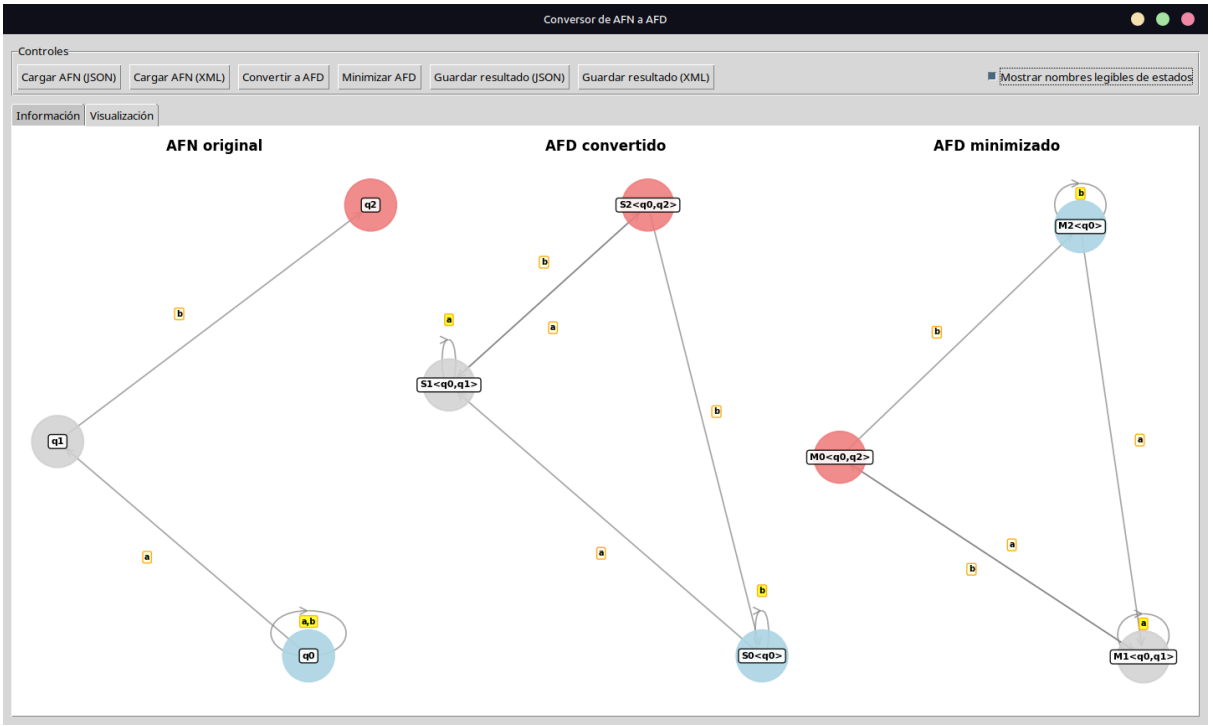
- Convierte a AFD, observa cómo los estados compuestos agrupan varios del AFND.
- Minimiza el AFD, observa la reducción de estados.

Ejemplo mediano (ejemploTP1_9a.json):
- AFND: 5 estados, 10 transiciones.
- AFD: 8 estados.
- AFD mínimo: 5 estados.

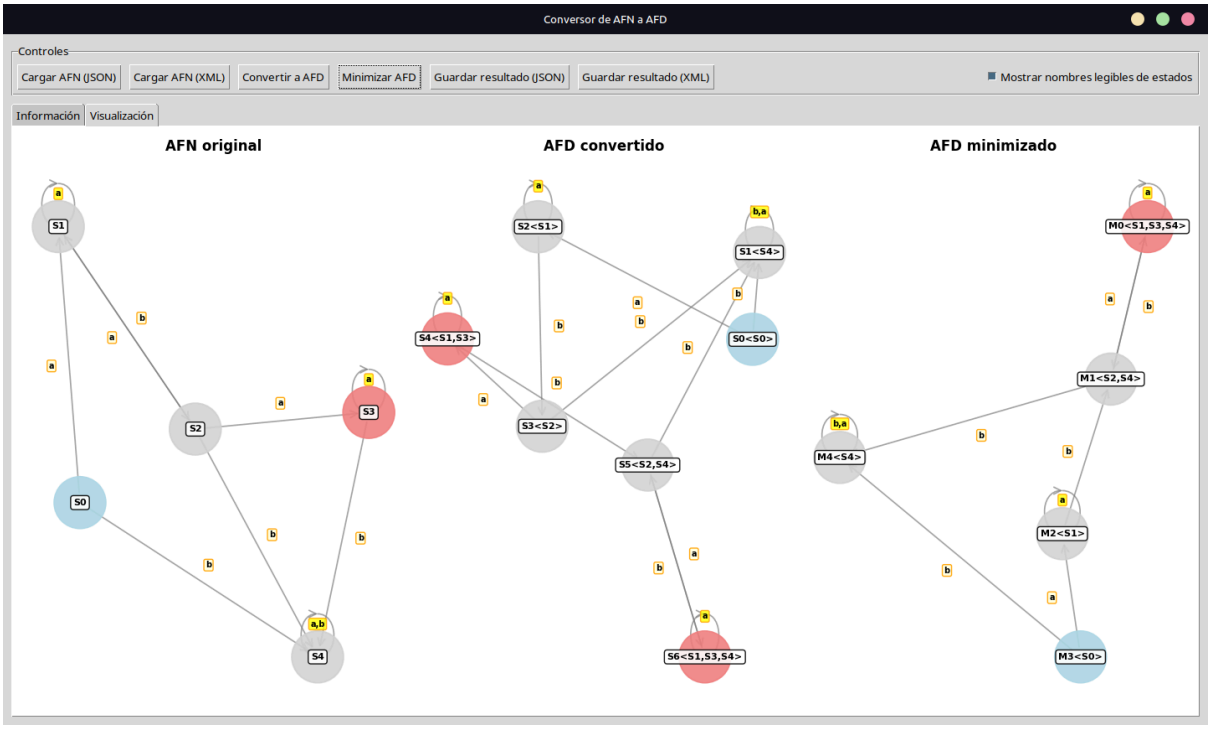
Iteraciones de minimización: El algoritmo de Hopcroft refina los bloques en varias etapas, mostrando cómo se agrupan los estados equivalentes.

3.1: Comparación Visual

La GUI muestra tres diagramas:
1. AFND original.
2. AFD convertido (con composición de estados).
3. AFD mínimo (con agrupación final).



Autómata de samples/sample_nfa.json.



Autómata de samples/ejemploTP1_9a.json.

4: Resultados y Análisis

4.1: Tabla Comparativa

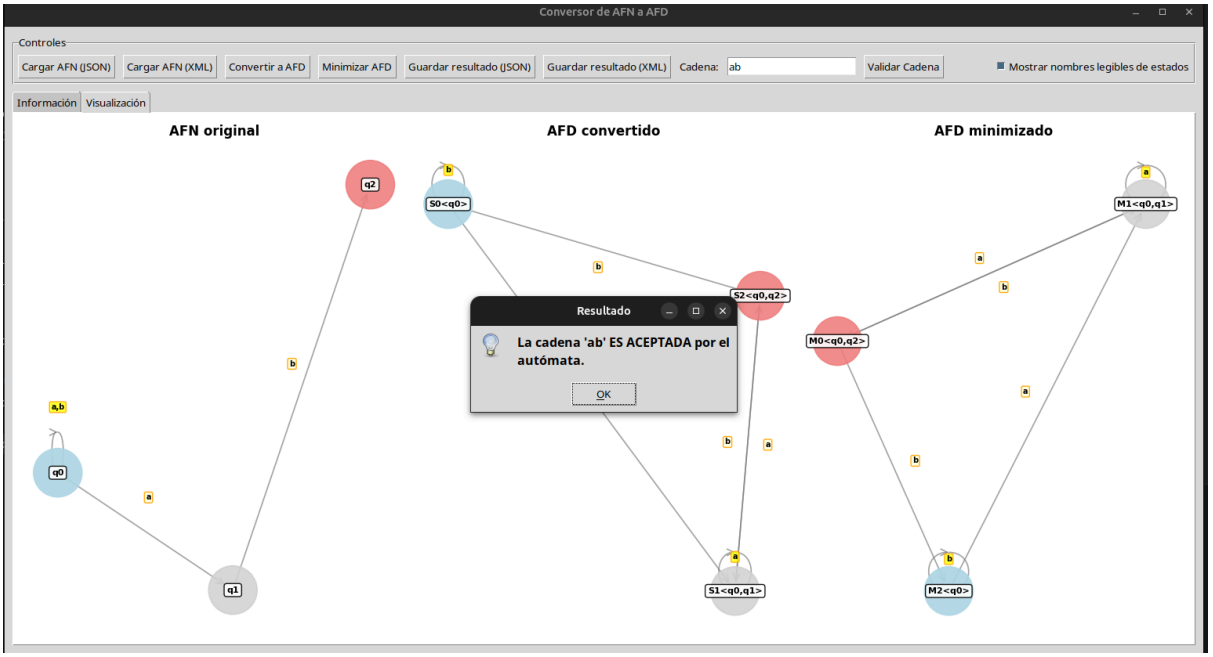
Caso	Estados AFND	Transiciones AFND	Estados AFD	Transiciones AFD	Estados Mínimo	Transiciones Mínimo
sample_nfa.json	3	5	4	8	3	6
ejemploTP1_9a.json	5	10	8	16	5	10

4.2: Verificación de Equivalencia

- Se simulan cadenas de prueba en el AFND, AFD y AFD mínimo.
- Se verifica que todos aceptan/rechazan las mismas cadenas.
- La equivalencia se comprueba por simulación y comparación de lenguajes aceptados.

4.3: Verificación de cadenas

Funcionalidad en la interfaz gráfica que permite **validar cadenas de entrada** en base al autómata cargado, convertido o minimizado. Esta funcionalidad añade un **campo de texto** y un **botón** denominado *"Validar Cadena"* en la sección de controles de la aplicación simulacion realizada en : sample_nfa.json



5: Conclusiones

El algoritmo de conversión y minimización es eficiente para autómatas pequeños y medianos. La modularidad permite escalar y mantener el código fácilmente.
Para autómatas grandes, la visualización puede ser lenta y el número de estados puede crecer exponencialmente en la conversión.

El desarrollo de este proyecto permitió comprender y aplicar los conceptos fundamentales de la teoría de autómatas, específicamente la **conversión de autómatas no deterministas (NFA/ε-NFA) a deterministas (DFA)** y su posterior **minimización**.

La implementación práctica del algoritmo de subconjuntos y del método de Hopcroft no solo facilitó la automatización del proceso, sino que también evidenció la importancia de optimizar los modelos, reduciendo la complejidad sin perder capacidad de reconocimiento.

Mejoras posibles:

- Implementar pruebas automáticas de equivalencia.
- Optimizar la visualización para grandes autómatas.

6: Referencias

- Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2007). *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Pearson Education. Fuente donde se presentan el algoritmo de conversión de NFA a DFA y el algoritmo de minimización de Hopcroft.

- *Libro - Fundamentos de ciencia de la computación* - Juan Carlos Augusto. Fuente proporcionada por la cátedra