Trabajo Práctico Integrador - Documentación

Facultad de Ingeniería

Cátedra: Fundamentos Teóricos de la Informática.

Alumnos: Grilli Ignacio (grillidavidignacio@gmail.com),
Casteglione Matias (matiascasteglione2002@gmail.com).

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

2: Metodología

2.1: Algoritmos implementados

- **Conversión AFND a AFD:** Convierte un autómata no determinista (AFN), que puede tener transiciones épsilon, en un autómata determinista (AFD) equivalente.
    - Se parte del estado inicial del AFN y se calcula su cierre épsilon (todos los estados alcanzables solo con transiciones épsilon).
    - Cada conjunto de estados del AFN se representa como un estado único en el AFD.
    - Para cada símbolo del alfabeto, se calcula el conjunto de estados alcanzables desde el conjunto actual, aplicando primero la transición y luego el cierre épsilon.
    - Se repite el proceso para cada nuevo conjunto de estados generado, hasta que no haya más conjuntos nuevos.
    - Los estados de aceptación del AFD son aquellos conjuntos que contienen al menos un estado de aceptación del AFN.

- **Cierre épsilon:** Determina todos los estados alcanzables desde un estado dado (o conjunto de estados) usando solo transiciones épsilon.
    - Se inicia con el estado (o conjunto) de partida.
    - Se exploran recursivamente todas las transiciones épsilon, agregando los estados alcanzados.
    - El proceso termina cuando no se encuentran nuevos estados por épsilon.

- **Minimización de AFD:** Reduce el número de estados de un AFD, generando un AFD mínimo equivalente.
    - Se parte de una partición inicial: estados de aceptación y no aceptación.
    - Se refina la partición dividiendo los grupos según las transiciones para cada símbolo del alfabeto.
    - Se repite el refinamiento hasta que no se puedan dividir más los grupos.
    - Cada grupo final se convierte en un estado del AFD minimizado.
    - Se reconstruyen las transiciones y los estados de aceptación/iniciales según la nueva partición.

- **Validación de Cadenas:** Determina si una cadena es aceptada por el autómata cargado (AFN, AFD o minimizado).
    - **Para AFD:** se recorre la cadena símbolo por símbolo, siguiendo las transiciones; si se llega a un estado de aceptación al final, la cadena es aceptada.
    - **Para AFN:** se usa el cierre épsilon y búsqueda en anchura (BFS) para explorar todos los caminos posibles; si algún estado de aceptación es alcanzado al consumir toda la cadena, se acepta.

2.2: 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.
- **automaton_to_json_dict**: 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.

2.3: Pseudocódigo: Conversión AFND a AFD

INICIO
- Cerrar épsilon el estado inicial del AFN → S0
- Crear una lista de estados pendientes: [S0]
- Crear el conjunto de estados del AFD: {S0}
- Marcar S0 como estado inicial del AFD

Mientras haya estados pendientes:
- Tomar el siguiente conjunto S de la lista
- Para cada símbolo 'a' del alfabeto (excepto épsilon):
  - Para cada estado s en S:
    - Obtener los destinos por 'a' desde s → D
    - Agregar D al conjunto de destinos
  - Cerrar épsilon el conjunto de destinos → T
  - Si T no está en el conjunto de estados del AFD:
    - Agregar T al conjunto de estados del AFD
    - Agregar T a la lista de pendientes
  - Crear transición en el AFD: S --a--> T

- Todo conjunto S que contenga al menos un estado de aceptación del AFN
- El AFD está definido por los conjuntos generados y las transiciones creadas.

FIN

2.4: Decisiones de Diseño

- **Modularización:** El proyecto se dividió en módulos independientes: modelo, entrada/salida, algoritmos, gráficos, interfaz gráfica y entrada principal.
- **Formato de Datos:** Se eligió JSON como único formato de entrada/salida para los autómatas.

- **Visualización:** Se utilizó Tkinter para la GUI, permitiendo cargar, visualizar, convertir, minimizar y validar cadenas. Se empleó matplotlib y networkx para mostrar los autómatas de forma gráfica, ayudando a la comprensión y depuración.


```json
{
  "name": "tabla_automata_2",
  "states": ["S0", "S1", "S2", "S3", "S4"],
  "alphabet": ["a", "b"],
  "start_state": "S0",
  "accept_states": ["S3"],
  "transitions": {
    "S0": { "a": ["S1"], "b": ["S4"] },
    "S1": { "a": ["S1"], "b": ["S2"] },
    "S2": { "a": ["S1", "S3"], "b": ["S4"] },
    "S3": { "a": ["S3"], "b": ["S4"] },
    "S4": { "a": ["S4"], "b": ["S4"] }
  }
}
```


Formato de entrada para un autómata en json.


```json
{
  "name": "tabla_automata_2__DFA__MIN",
  "states": ["M0", "M1", "M2", "M3", "M4"],
  "alphabet": ["a", "b"],
  "start_state": "M3",
  "accept_states": ["M0"],
  "is_dfa": true,
  "transitions": {
    "M0": { "a": "M0", "b": "M1" },
    "M1": { "a": "M0", "b": "M4" },
    "M2": { "a": "M2", "b": "M1" },
    "M3": { "a": "M2", "b": "M4" },
    "M4": { "a": "M4", "b": "M4" }
  },
  "state_composition": {
    "M0": [ "S1", "S3", "S4" ],
    "M1": [ "S2", "S4" ],
    "M2": [ "S1" ],
    "M3": [ "S0" ],
    "M4": [ "S4" ]
  }
}
```


Formato de salida, después de la conversión y minimización de un autómata en json.

3: Casos de Prueba

Se incluyen ejemplos en la carpeta samples/:

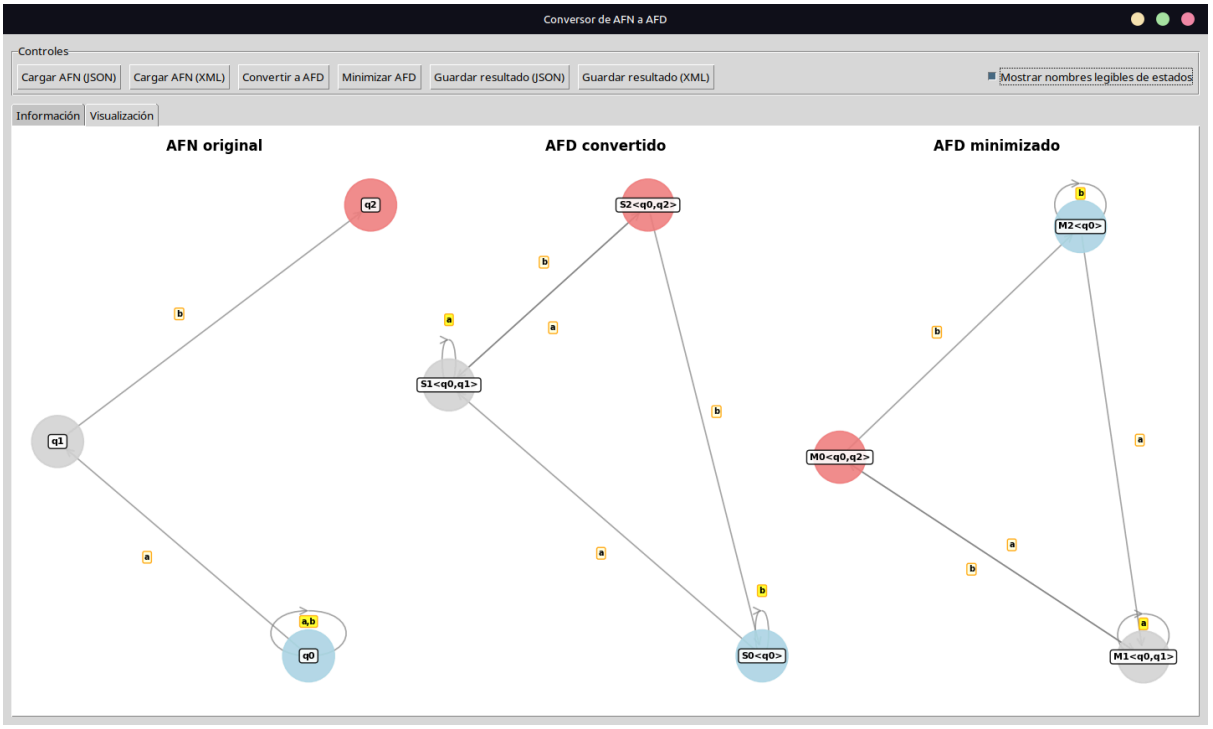
Ejemplo pequeño (sample_nfa.json)
- AFND: 3 estados, transiciones épsilon.
- AFD generado: 4 estados, composición mostrada.
- AFD mínimo: 3 estados.

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.

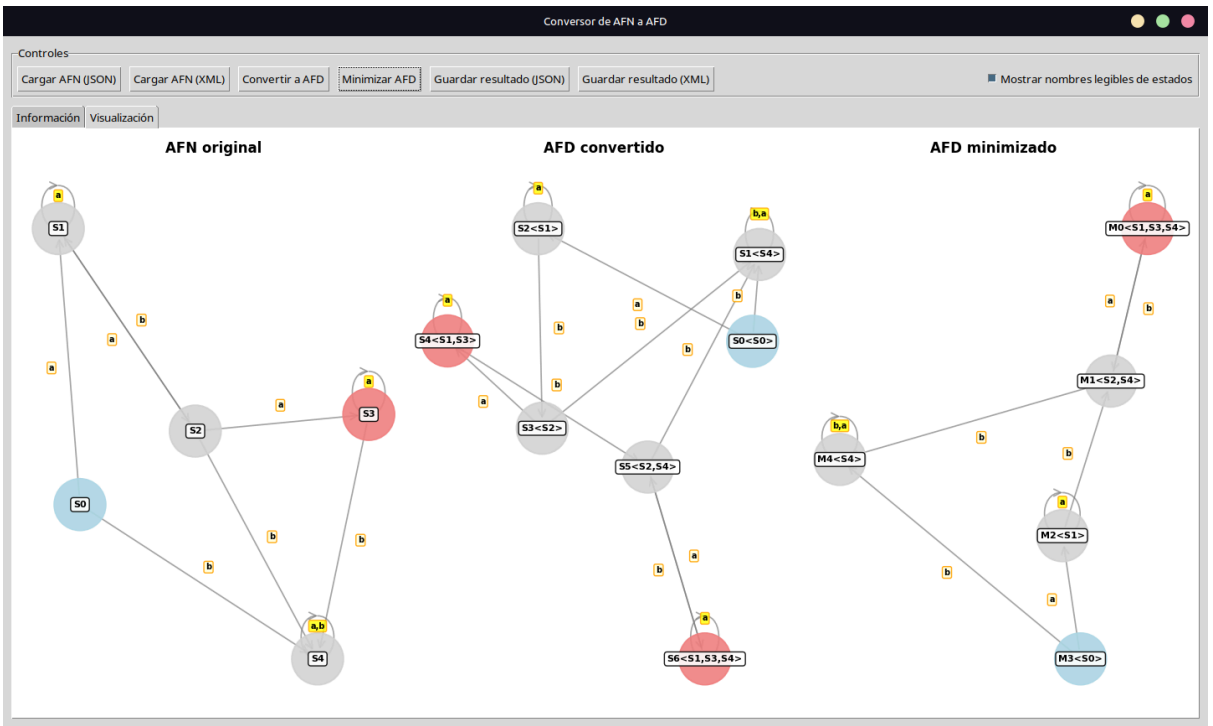
Ejemplo mediano (ejemploTP1_9a.json):
- AFND: 5 estados, 10 transiciones.
- AFD: 8 estados.
- AFD mínimo: 5 estados.

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

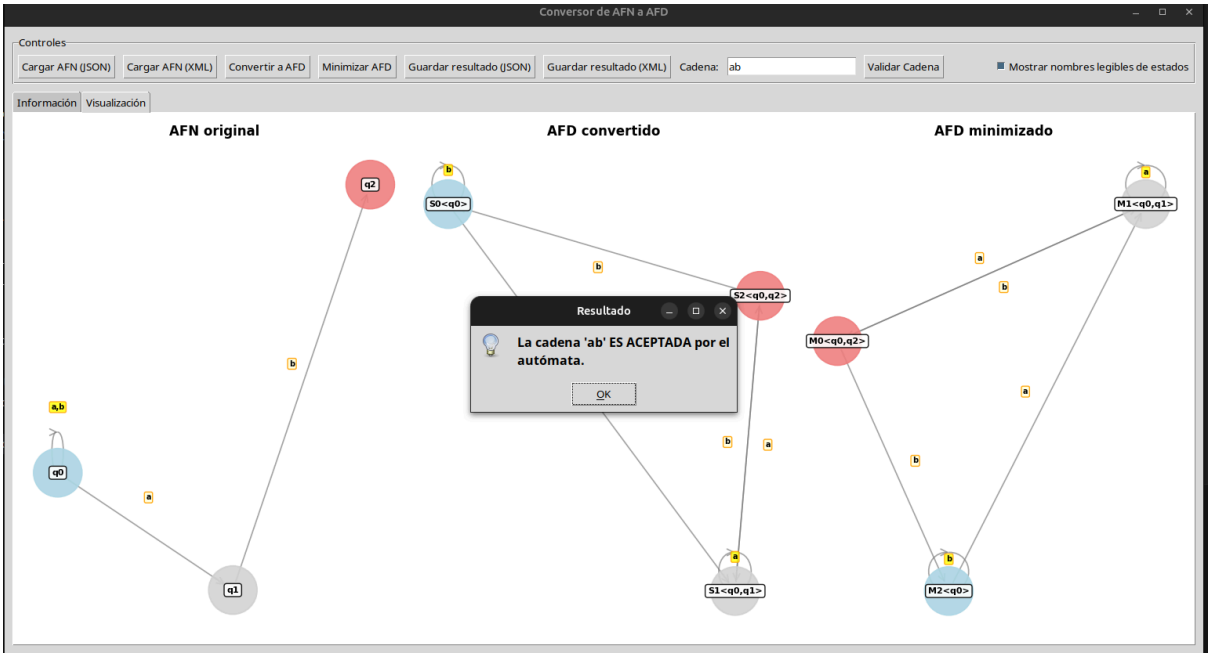
Caso	Estados AFND	Transiciones AFND	Estados AFD	Transiciones AFD	Estados Mínimo	Transiciones Mínimo
sample_nfa.json	3	5	4	8	3	6
ejemploTP1_9a.json	5	10	8	16	5	10

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 evidenció la importancia de optimizar los modelos, reduciendo la complejidad sin perder capacidad de reconocimiento.

Posibles Mejoras:

- 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.)

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