Autómatas finitos

1. ¿Qué es un autómata finito determinista (AFD)?

Un AFD es un **modelo matemático** que reconoce **lenguajes regulares**. Se caracteriza por ser **determinista**, lo que significa que para cada símbolo de entrada y cada estado actual, existe **una sola transición posible al siguiente estado**.

Σ es un alfabeto llamado alfabeto de entrada.

Q es un conjunto finito llamado conjunto de estados.

δ es una función de Q por Σ en Q (d : Q x Σ→ Q) o sea, que toma un estado y un símbolo del alfabeto y **produce otro estado**. A esta función se le llama función de transición.

q0 es un elemento de Q, llamado estado inicial.

F es un subconjunto de Q, llamado conjunto de estados finales.

2. ¿Qué es un autómata finito no determinista (AFND)?

Los autómatas finitos no deterministas (AFND) se diferencian de los AFD en cómo se define la función de transición. En un AFND, para un símbolo del alfabeto, puede existir **más de una transición posible desde un estado** y por tanto no se puede determinar con certeza el siguiente estado conociendo el estado actual y el símbolo a leer.

Σ es un alfabeto llamado alfabeto de entrada.

Q es un conjunto finito llamado conjunto de estados.

δ es una función de Q por Σ unión ε en el conjunto potencia de Q [d : Q x (ΣUε)→ P(Q)] o sea, delta es una función que toma un par ordenado formado por un estado de Q y un símbolo del alfabeto Σ o ε, y devuelve **un subconjunto** de Q. A esta función se le llama función de transición.

q0 es un elemento de Q, llamado estado inicial.

F es un subconjunto de Q, llamado conjunto de estados finales.

Tanto los AFD's como los AFND's son equivalentes en poder computacional (aceptan los mismos lenguajes regulares), pero elegir entre uno y otro depende del problema y la etapa de diseño (teoría vs. implementación).

3. AFD's vs AFND's

Propósito de los AFND

- Modelar situaciones con múltiples caminos simultáneos:
- Permiten transiciones múltiples desde un mismo estado para un símbolo (o incluso sin consumir símbolos, con transiciones ε).
- Capturan la idea de "adivinar" correctamente un camino hacia la aceptación, lo que es útil en análisis teórico.

Simplificar el diseño de autómatas:

- Para ciertos lenguajes, diseñar un AFND es más sencillo que un AFD. Por ejemplo, un AFND con transiciones ε puede unir dos autómatas fácilmente, mientras que un AFD requeriría rediseñar toda la estructura.

Relación con expresiones regulares:

- Los AFND están estrechamente ligados a las expresiones regulares. Convertir una expresión regular a un AFND es directo, mientras que hacerlo a un AFD implica pasos intermedios complejos.

¿Por qué no usar siempre AFD?

Los AFD son más eficientes en ejecución (su simulación es trivial), pero los AFND destacan en:

Menor número de estados:

- Un AFND puede representar ciertos lenguajes con exponencialmente menos estados que su AFD equivalente. Por ejemplo, el lenguaje de cadenas con una "a" en la antepenúltima posición requiere un AFD con al menos 8 estados, pero un AFND lo hace con 4.

Abstracción en diseños complejos:

- Al componer autómatas (uniones, concatenaciones, etc.), los AFND permiten usar transiciones ε para "conectar" componentes sin modificar cada estado individual, algo engorroso en AFD.

Teoría y pruebas:

- En demostraciones matemáticas (como cerradura bajo operaciones regulares), los AFND simplifican argumentos al evitar la explosión de estados de los AFD.

Implementación de herramientas prácticas:

- Algoritmos como el de Thompson (usado en compiladores para convertir regex a autómatas) se basan en AFND por su eficiencia en el diseño, aunque luego se convierten a AFD para su ejecución.

Conclusión

Los AFND no son "mejores" que los AFD, pero son **herramientas complementarias**:

Usar AFND cuando:

- Quieras diseñar autómatas rápidamente.
- Trabajes con operaciones teóricas (uniones, estrellas de Kleene).
- Busques representaciones compactas.

Usar AFD cuando:

- Necesites implementar un reconocedor eficiente.
- Requieras simplicidad en la ejecución (sin ambigüedades).