2. Autómatas finitos

2.1 Autómata Finito Determinista (AFD)

> **Concepto Clave:** "Una máquina sin dudas".

Un AFD es el modelo más estricto. Imagínalo como un tablero de juego donde, dado una casilla (estado) y una carta (símbolo), **solo tienes una única jugada legal**.

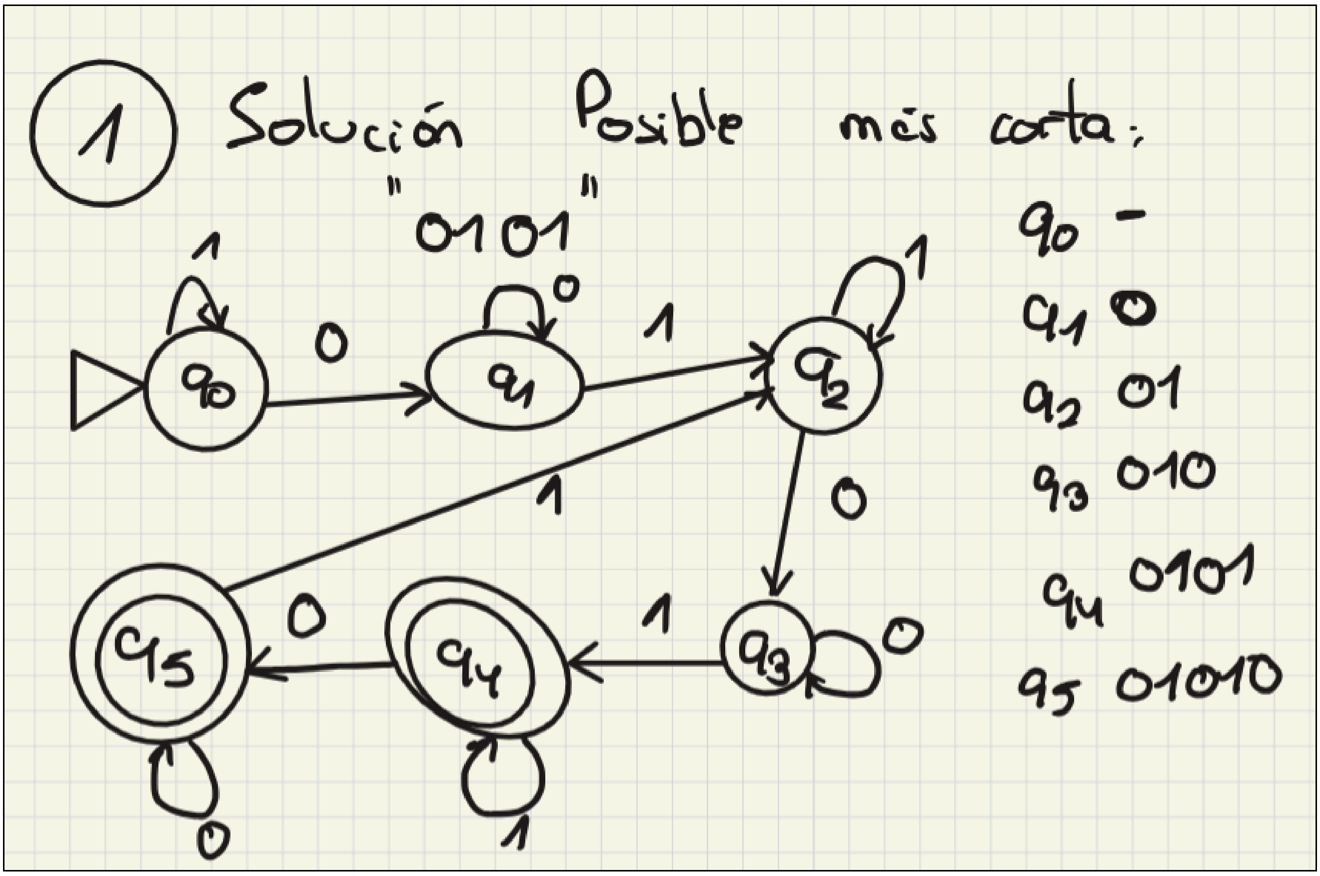
Características para identificar un AFD:

- **Determinismo:** Para cada estado y cada símbolo del alfabeto, existe **exactamente una** flecha de salida. Ni cero, ni dos. Una.
- **Sin magia:** No existen movimientos "gratuitos" (no hay transiciones ε).

Definición Formal (La "5-tupla"):
En ejercicios teóricos, siempre debes definir estas 5 partes:

$$A = (Q, \Sigma, \delta, q_0, F)$$

- Q: Catálogo de todos los estados (los círculos).
- Σ: Alfabeto (las letras o números admitidos, ej: $\{0, 1\}$).
- δ: **El mapa de carreteras**. Es una función $Q \times \Sigma \to Q$. (Entra un estado y un símbolo, sale **un** estado).
- q_0: Donde empieza todo (flecha sin origen).
- F: Donde ganamos (doble círculo).



2.2 Autómata Finito No Determinista (AFN)

> **Concepto Clave:** "Procesamiento en paralelo" o "Multiverso".

El AFN es una máquina teórica más flexible. A diferencia del AFD, aquí la máquina puede "adivinar" el camino correcto.

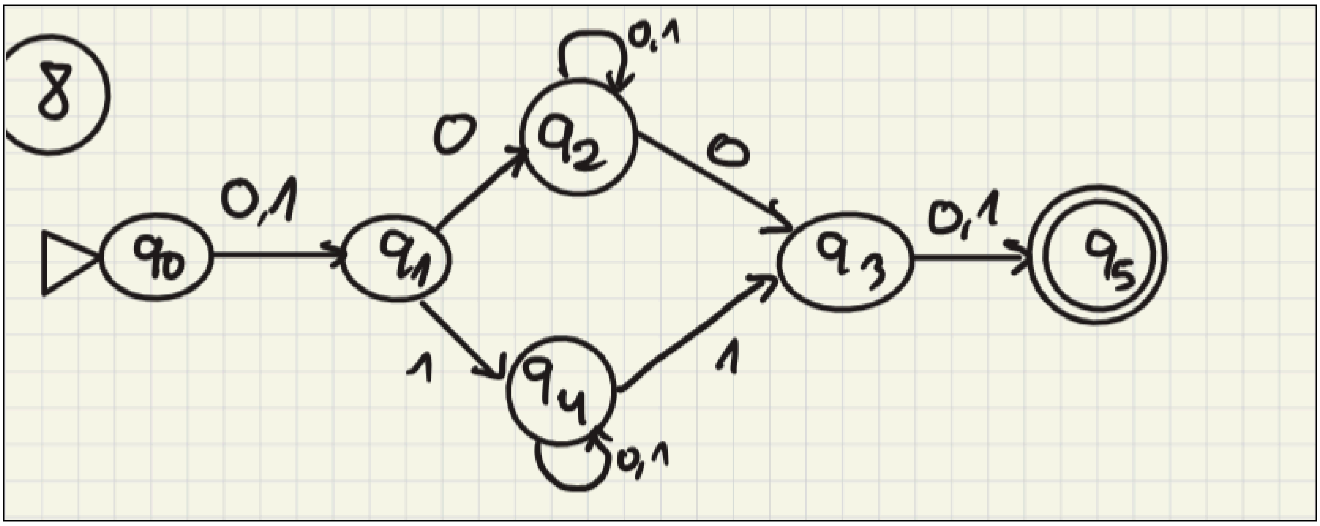
Diferencias prácticas para ejercicios

- **Ambigüedad:** Desde un estado con un símbolo (ej: 'a'), pueden salir **múltiples flechas** o **ninguna**.
- **Multiverso:** Si hay dos flechas con 'a', el autómata se clona y sigue ambos caminos a la vez.
- **Aceptación:** Si **al menos una** de las copias del autómata llega a un estado final al terminar la cadena, la cadena se acepta. Si todas mueren o acaban en no-finales, se rechaza.

Definición Formal
La única diferencia real con el AFD está en la función de transición δ:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$$

- **Traducción:** La función devuelve un **conjunto de estados** (potencia de Q), no un estado único. Puede devolver un conjunto vacío $\emptyset$ (callejón sin salida).



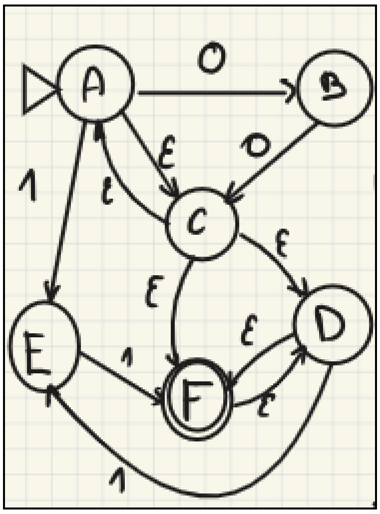
En el AFN, puedes estar en varios estados a la vez y elegir entre múltiples caminos.

2.3 Transiciones ε (Epsilon) y `Clausura-ε`

Una transición ε es un **teletransporte**. Permite al autómata cambiar de estado **sin leer nada** de la cinta de entrada. Te permite estar en varios estados al mismo tiempo

Para resolver ejercicios de conversión, necesitas dominar la **Clausura-ε**.

- **Pregunta:** "¿A dónde puedo llegar desde aquí sin gastar ni una moneda (símbolo)?"
- **Regla:** La `clausura-ε(q)` siempre incluye al propio estado q más cualquier estado alcanzable solo con flechas ε.



La clausura de la imagen anterior sería:

A^0 = cierre-ε(A)
= {A, C, D F}

Algoritmo para sacar la clausura:

1. Sitúate en un estado (ej: q_0).

2. Mete q_0 en tu saco (siempre llegas a ti mismo).
3. Mira si salen flechas con ε. Si sí, sigue esas flechas a los nuevos estados.
4. Desde esos nuevos estados, ¿salen más ε? Síguelas también.
5. Repite hasta que no puedas avanzar más sin leer letras.

> **Ejemplo:** $q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_2$
>
> - $Clausura(q_0) = \{q_0, q_1, q_2\}$
> - $Clausura(q_1) = \{q_1, q_2\}$

2.4 Equivalencia y Conversión: AFN → AFD

Los ordenadores reales no son "adivinos" (no son no-deterministas). Para programar un AFN, primero debemos convertirlo a AFD.

Como el AFN puede estar en varios sitios a la vez, **cada estado del nuevo AFD será un grupo de estados del AFN original**.

Algoritmo:

1. **Inicio:** Calcula la $clausura\text{-}\varepsilon(q_0)$ del AFN. Este conjunto de estados es tu estado inicial del AFD. Llámalo "A".

2. **Iteración:** Para el nuevo estado "A" y cada símbolo del alfabeto (ej: 0 y 1):
   - Mira a dónde van los estados dentro de "A" con ese símbolo.
   - A los destinos, aplícales $clausura\text{-}\varepsilon$.
   - El resultado es un nuevo conjunto. ¿Ya existe? Úsalo. ¿No existe? Bautízalo como "B".

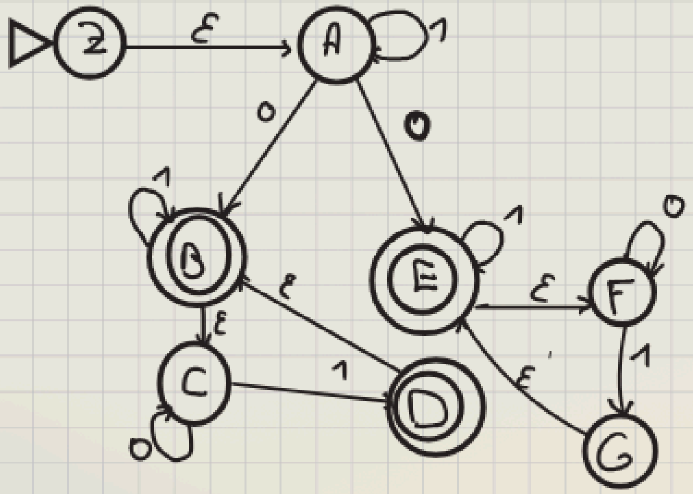
3. **Iteración:** Repite el paso 2 con "B", "C", etc., hasta que no aparezcan conjuntos nuevos.

4. **Estados Finales:** Cualquier estado del AFD (A, B, C...) que contenga **al menos un** estado final del AFN original, se convierte en estado final.

Ejemplo práctico:

2

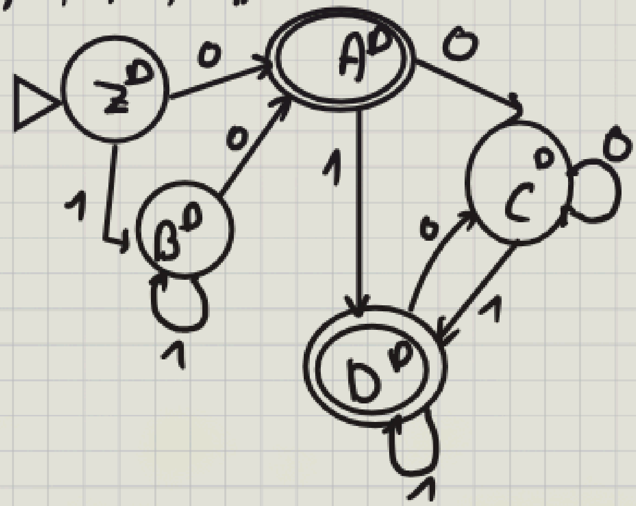
$z^D = \text{cierre-}\varepsilon(z) = \{z, A\}$



$$\begin{cases} \text{mueve}(z^D,0) = \{B,E\} \\ \text{cierre-}\varepsilon(B,E) = \{B,E,C,F\} = A^D \\ \text{mueve}(z^D,1) = \{A\} \\ \text{cierre-}\varepsilon(A) = \{A\} = B^D \end{cases}$$

$$\begin{cases} \text{mueve}(A^D,0) = \{C,F\} \\ \text{cierre-}\varepsilon(C,F) = \{C,F\} = C^D \\ \text{mueve}(A^D,1) = \{B,E,D,G\} \\ \text{cierre-}\varepsilon(B,E,D,G) = \{B,C,E,F,D,G\} = D^D \end{cases}$$

$$\begin{cases} \text{mueve}(B^D,0) = \{B,E\} \\ \text{cierre-}\varepsilon(B,E) = A^D \\ \text{mueve}(B^D,1) = \{A\} \\ \text{cierre-}\varepsilon(A) = B^D \end{cases}$$



$$\begin{cases} \text{mueve}(C^D,0) = \{C,F\} \\ \text{cierre-}\varepsilon(C,F) = C^D \\ \text{mueve}(C^D,1) = \{D,G\} \\ \text{cierre-}\varepsilon(D,G) = \{D,B,C,G,E,F\} = D^D \end{cases}$$

$$\begin{cases} \text{mueve}(D^D,0) = \{C,F\} \\ \text{cierre-}\varepsilon(C,F) = C^D \\ \text{mueve}(D^D,1) = \{B,D,E,G\} \\ \text{cierre-}\varepsilon(B,D,E,G) = D^D \end{cases}$$

2.5 Minimización de AFD

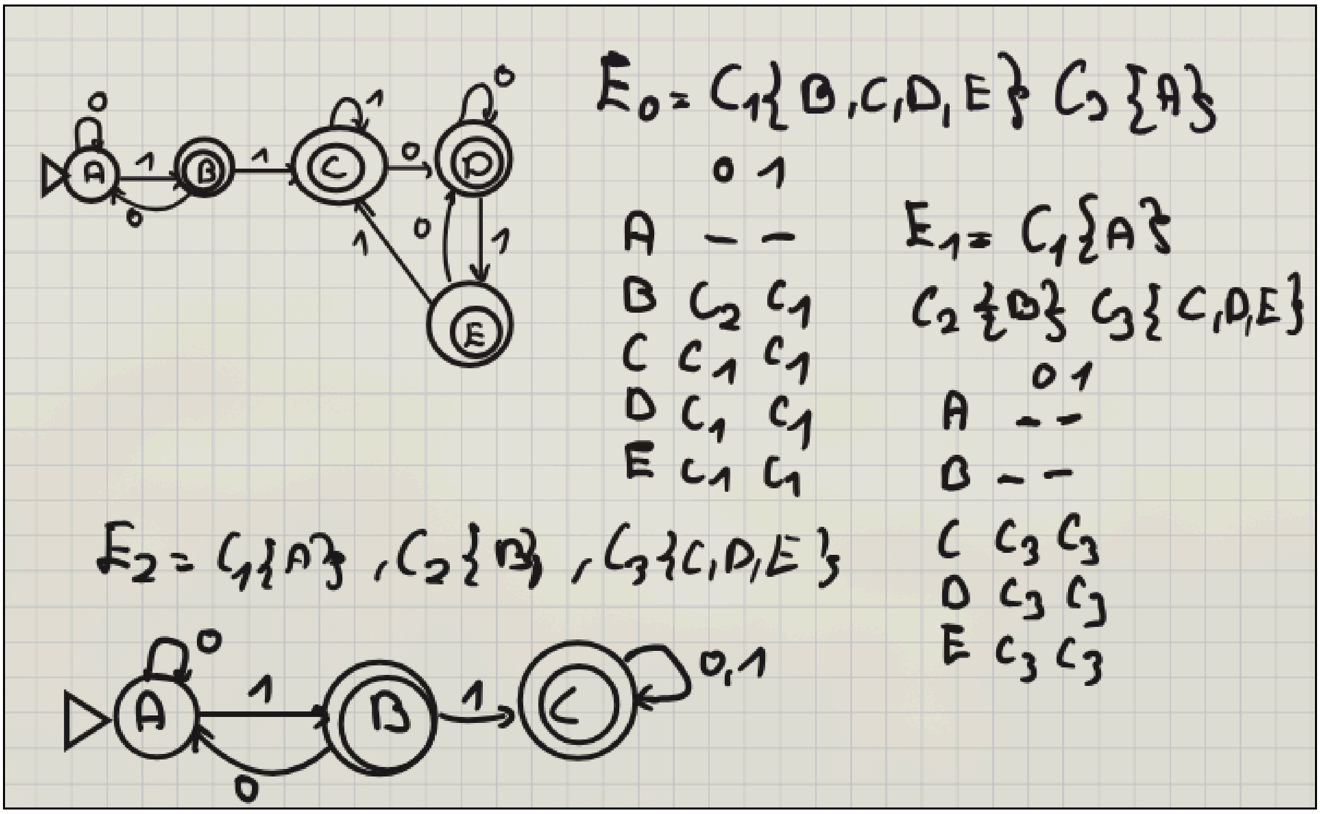
Objetivo: Encontrar el autómata más pequeño posible que haga exactamente lo mismo. Elimina redundancia.

Algoritmo

1. Partición Inicial (E_0): Divide los estados en solo dos grupos (clases):
- **Finales (F):** Todos los estados de aceptación.
- **No Finales ($Q \setminus F$):** El resto.
- *Etiqueta cada grupo como* $C_1, C_2, \ldots$
2. Construcción de la Tabla de Transiciones: Para cada estado, anota a qué Grupo (C_x) viaja con cada símbolo (0, 1...), basándote en la partición anterior.
- *Truco:* No mires el estado destino, mira la **clase** del estado destino.
3. Refinamiento ($E_1, E_2 \ldots$): Analiza los grupos formados:
- Si dentro de un grupo (ej. C_1), todos los estados tienen el **mismo patrón de clases destino**, se quedan juntos.
- Si un estado tiene un patrón diferente, se separa ("rompe" el grupo) y crea una nueva clase para la siguiente iteración.

4. Parada: Repite el proceso hasta que E_n sea idéntica a E_{n-1} (ya no se rompen más grupos).

5. Reconstrucción: Cada grupo final es un único estado en el autómata minimizado.



2.6 Equivalencia entre Estados

Dos estados p y q son equivalentes si son **indistinguibles** para un observador externo.

Imagínate que metemos el estado p en una caja negra y el estado q en otra. Tú no puedes ver el dibujo del autómata. Solo puedes meter cadenas de texto (inputs) y ver si la luz de "Aceptado" se enciende o no.

- Si para **cualquier** cadena infinita de pruebas que se te ocurra, las dos cajas siempre coinciden (o ambas aceptan, o ambas rechazan), entonces los estados son **equivalentes**.
- Si encuentras **una sola** cadena donde una caja dice "Sí" y la otra "No", entonces son **distinguibles** (no equivalentes).

Definición Formal

Sea un autómata (o dos) sobre un alfabeto Σ. Dos estados p y q son equivalentes (se denota $p \equiv q$) si y solo si:

$$\forall w \in \Sigma^* : (\hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F)$$

Traducción:
Para toda palabra w (desde la cadena vacía hasta una palabra de un millón de letras), si partimos de p y leemos w, el resultado final (aceptación o rechazo) debe ser exactamente el mismo que si partimos de q y leemos esa misma w.

Por lo que si nos dicen que si dos lenguajes regulares son equivalentes entre sí si los estados iniciales de sus correspondientes AFD son equivalentes