Tema 3: Autómatas finitos

Autómatas y Lenguajes Formales

Dpto. de Ingeniería de la Información y las Comunicaciones

UNIVERSIDAD
DE MURCIA

Modelo de autómata finito determinista

Un **autómata finito determinista** (*AFD*) es un formalismo matemático para especificación, modelado, diseño o análisis de sistemas que evolucionan cambiando de un estado a otro como reacción a determinados símbolos que se reciben de una fuente de entrada.

Modelo de autómata finito determinista

Un **autómata finito determinista** (*AFD*) es un formalismo matemático para especificación, modelado, diseño o análisis de sistemas que evolucionan cambiando de un estado a otro como reacción a determinados símbolos que se reciben de una fuente de entrada.

Un ejemplo de aplicación con AFD: planteamiento y análisis del problema

- **Un problema de validación de formato:** comprobar si una cadena representa un número decimal sin signo, con parte entera y fraccionaria obligatoria, separadas por un punto.

Modelo de autómata finito determinista

Un **autómata finito determinista** (*AFD*) es un formalismo matemático para especificación, modelado, diseño o análisis de sistemas que evolucionan cambiando de un estado a otro como reacción a determinados símbolos que se reciben de una fuente de entrada.

Un ejemplo de aplicación con AFD: planteamiento y análisis del problema

- **Un problema de validación de formato:** comprobar si una cadena representa un número decimal sin signo, con parte entera y fraccionaria obligatoria, separadas por un punto.
- **Identificación del lenguaje:** los símbolos son los dígitos decimales y el punto y $L_{ef} = \{x.y \mid x, y \in V_{dig}^{+}\}$

Modelo de autómata finito determinista

Un **autómata finito determinista** (*AFD*) es un formalismo matemático para especificación, modelado, diseño o análisis de sistemas que evolucionan cambiando de un estado a otro como reacción a determinados símbolos que se reciben de una fuente de entrada.

Un ejemplo de aplicación con AFD: planteamiento y análisis del problema

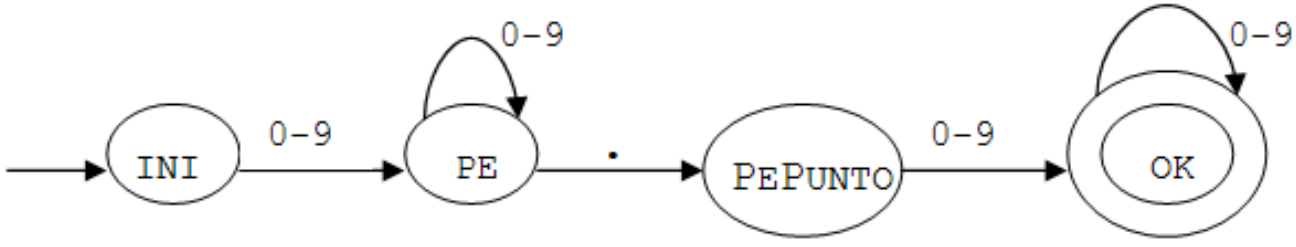
- **Un problema de validación de formato:** **comprobar si una cadena representa un número decimal sin signo, con parte entera y fraccionaria obligatoria**, separadas por un punto.
- **Identificación del lenguaje:** los símbolos son los dígitos decimales y el punto y $L_{ef} = \{x.y \mid x, y \in V_{dig}^{+}\}$
- **Problema teórico equivalente:** **comprobar si una cadena pertenece al lenguaje L_{ef}.**

Un ejemplo de aplicación con AFD: solución-modelado

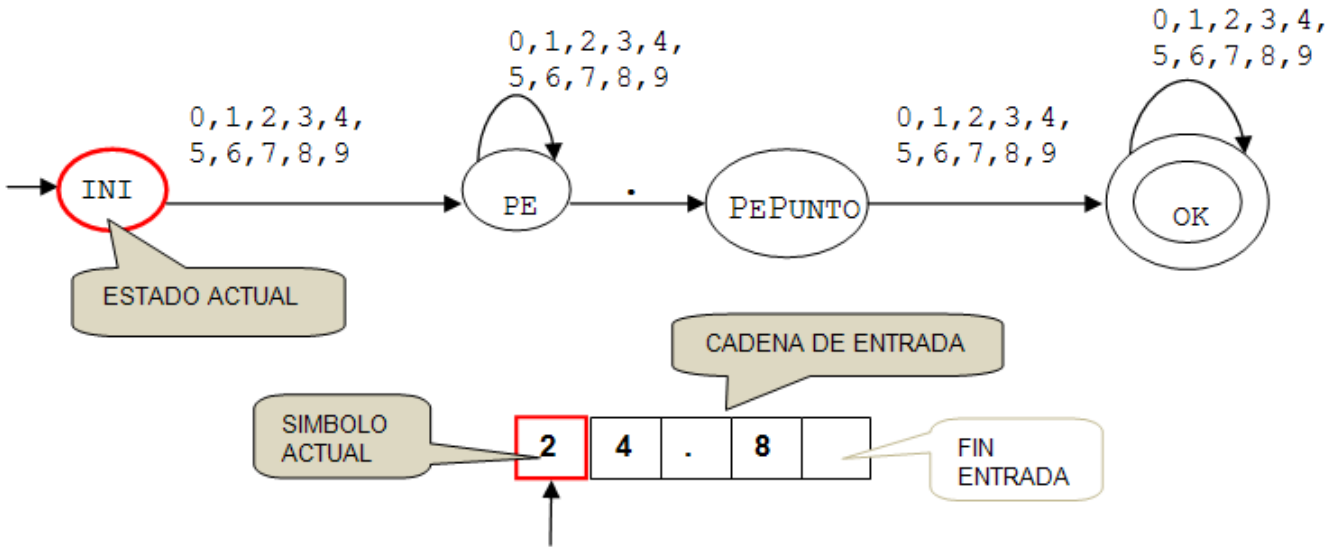
- **Un problema de validación de formato:** comprobar si una cadena representa un números decimal sin signo, con parte entera y fraccionaria obligatoria, separadas por un punto.
- **Idea para resolverlo:** leer la cadena carácter a carácter y distinguir las situaciones (**estados**) en las que nos podemos encontrar en el procesamiento de la cadena y en cómo se pasa de una situación a otra (**transición**) y cuándo podemos asegurar que la cadena es válida o incorrecta.

Un ejemplo de aplicación con AFD: solución-modelado

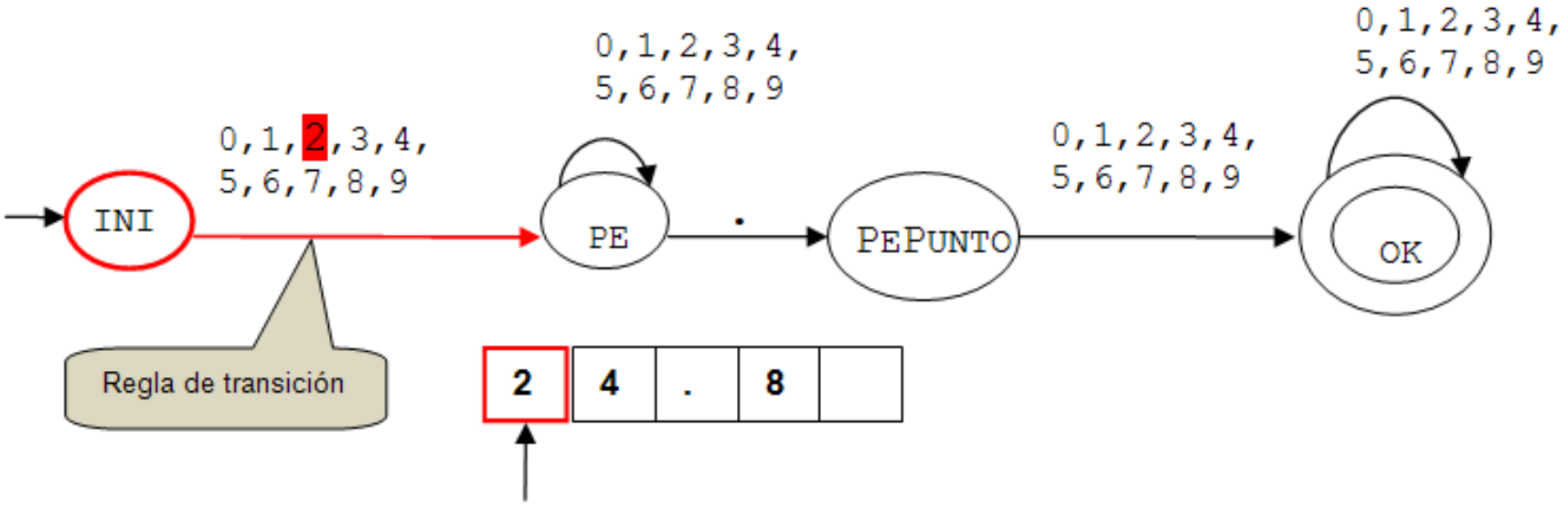
- **Un problema de validación de formato:** comprobar si una cadena representa un números decimal sin signo, con parte entera y fraccionaria obligatoria, separadas por un punto.
- **Idea para resolverlo:** leer la cadena carácter a carácter y distinguir las situaciones (**estados**) en las que nos podemos encontrar en el procesamiento de la cadena y en cómo se pasa de una situación a otra (**transición**) y cuándo podemos asegurar que la cadena es válida o incorrecta.
- La **solución algorítmica** se puede modelar con un *AFD*, que se representa mediante un **diagrama de transición:**



Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



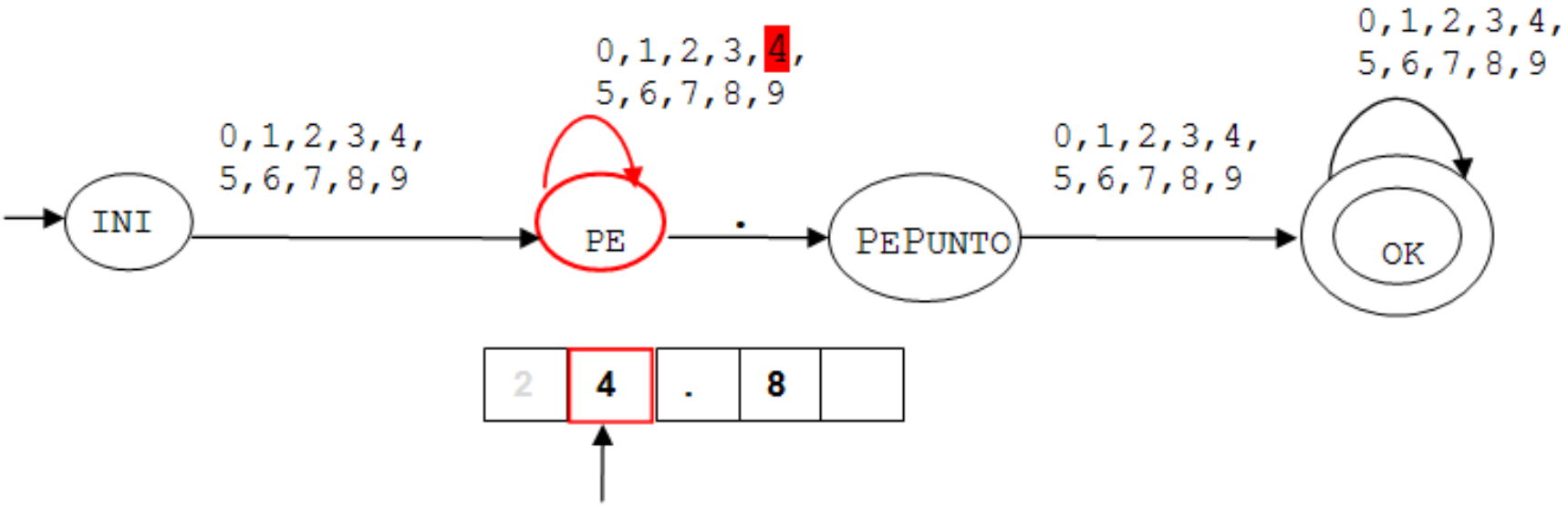
Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2}$$

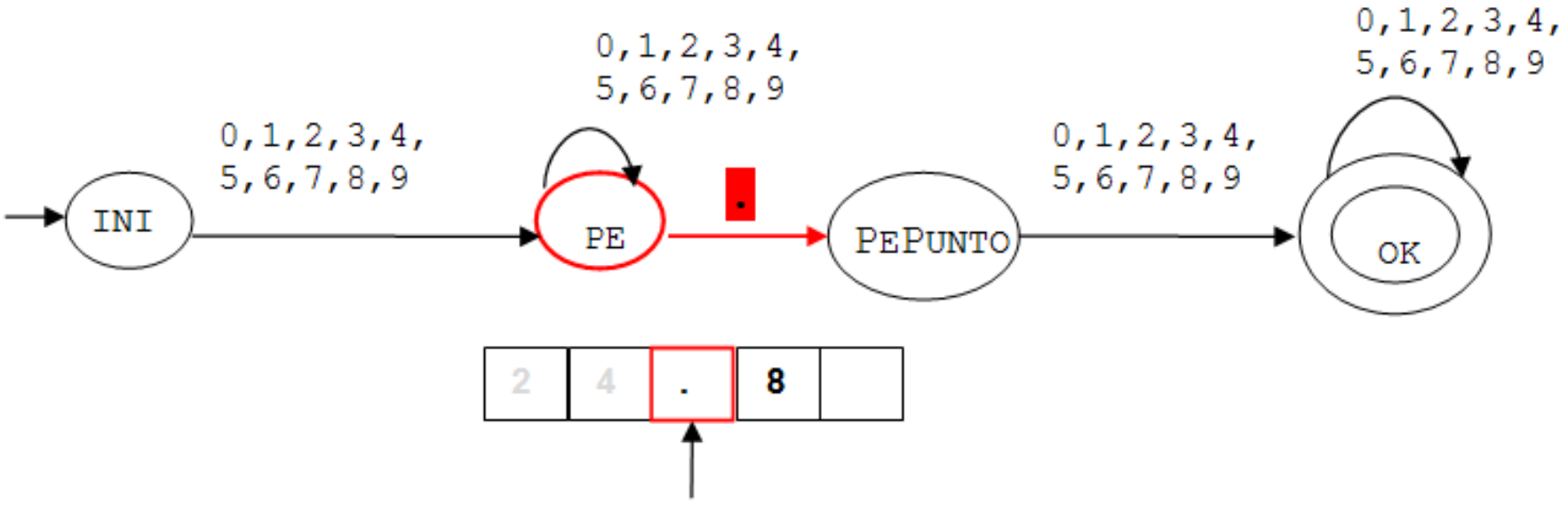
Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4}$$

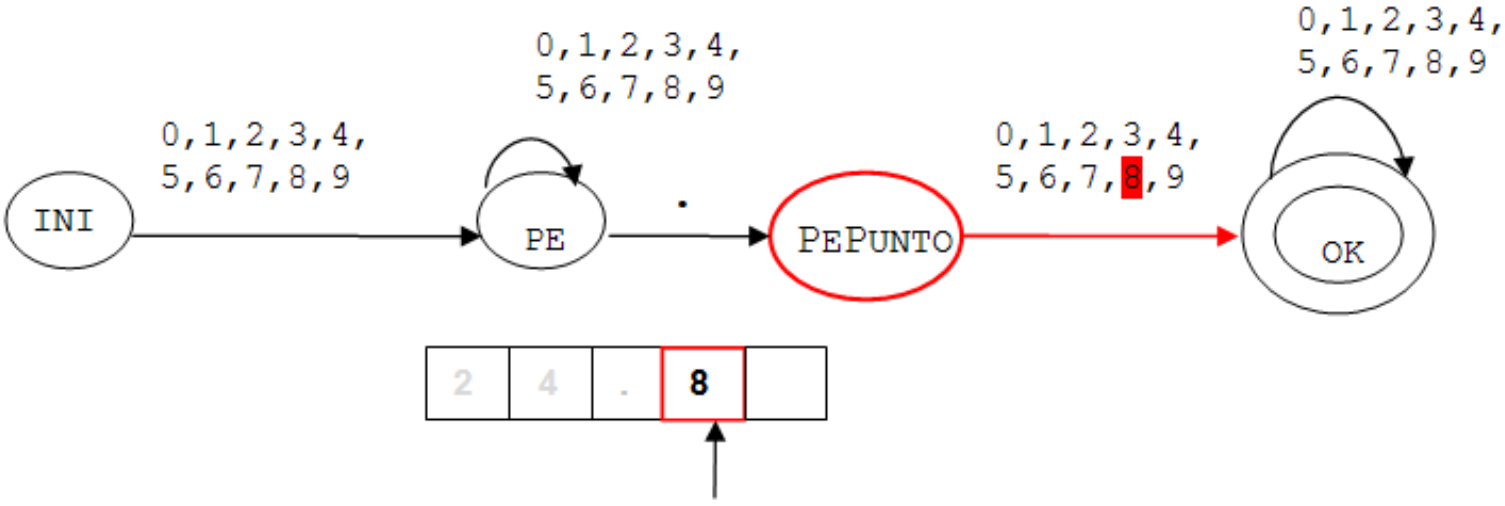
Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{.}$$

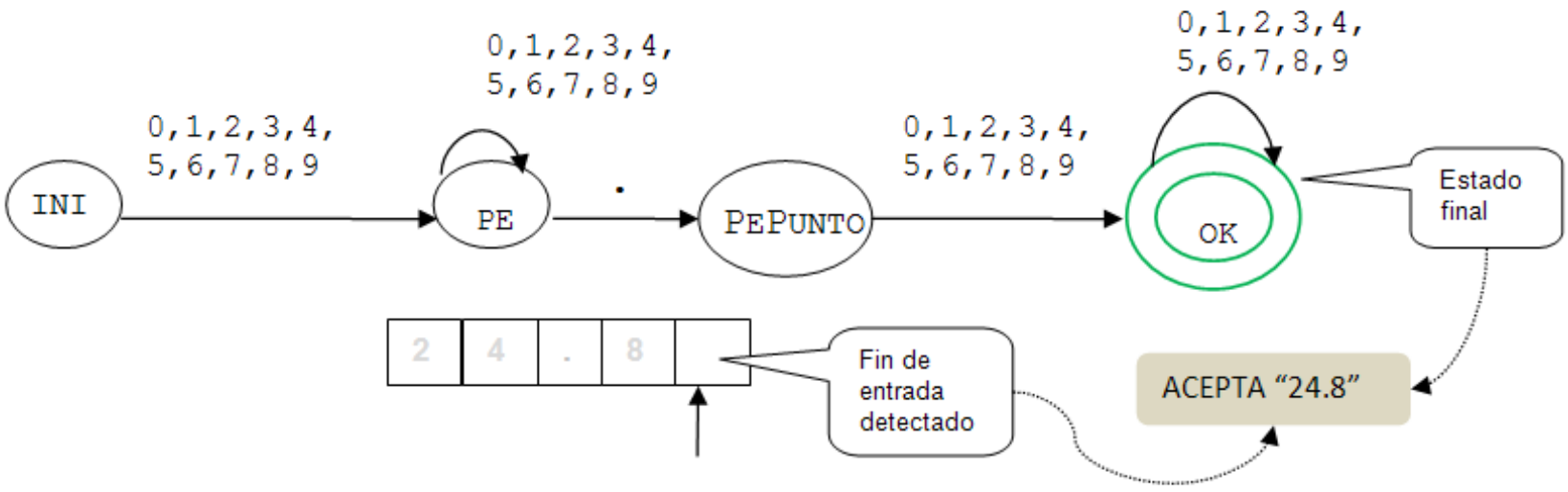
Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{.} \text{PEPUNTO} \xrightarrow{8}$$

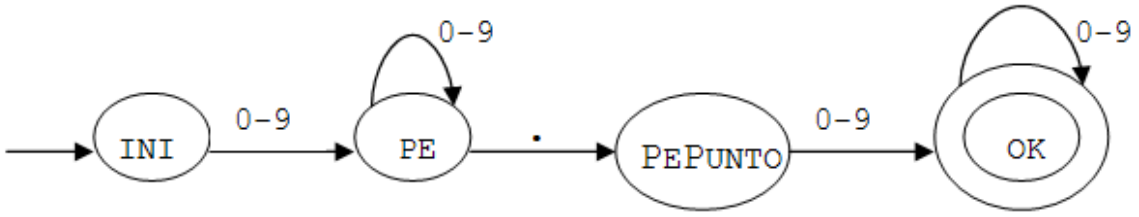
Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena “24.8”:

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{.} \text{PEPUNTO} \xrightarrow{8} \text{OK} \quad [\text{acepta}]$$

Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



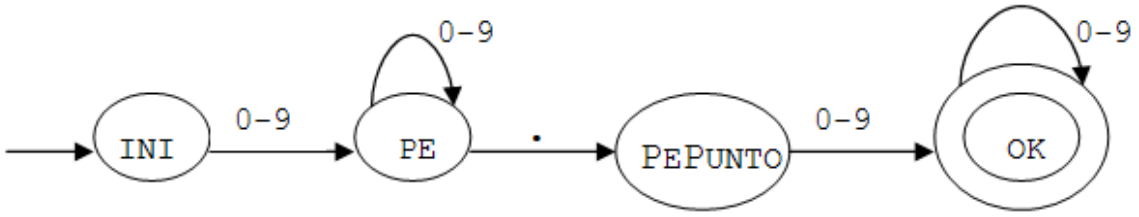
Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{\cdot} \text{PEPUNTO} \xrightarrow{8} \text{OK} \quad [\text{acepta}]$$

'La cadena "24" no es aceptada:

$$\text{INI} \xrightarrow{2}$$

Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



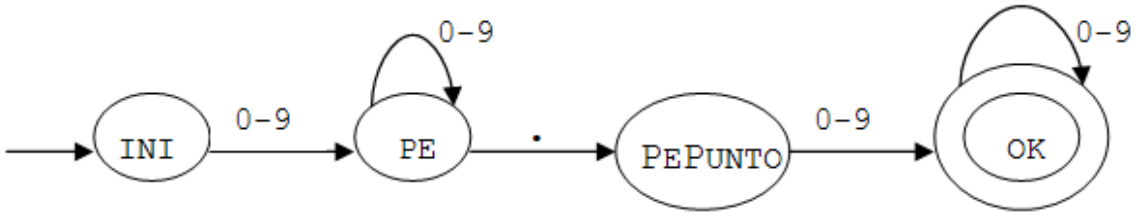
Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{\cdot} \text{PEPUNTO} \xrightarrow{8} \text{OK} \quad [\text{acepta}]$$

'La cadena "24" no es aceptada:

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4}$$

Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



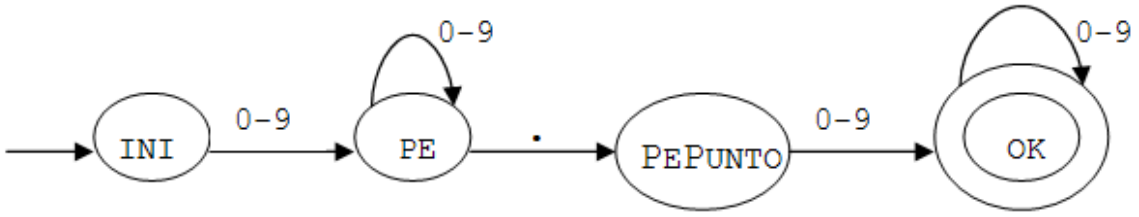
Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{\cdot} \text{PEPUNTO} \xrightarrow{8} \text{OK} \quad [\text{acepta}]$$

'La cadena "24" no es aceptada:

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \quad [\text{rechaza: PE no es estado final}]$$

Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{.} \text{PEPUNTO} \xrightarrow{8} \text{OK} \quad [\text{acepta}]$$

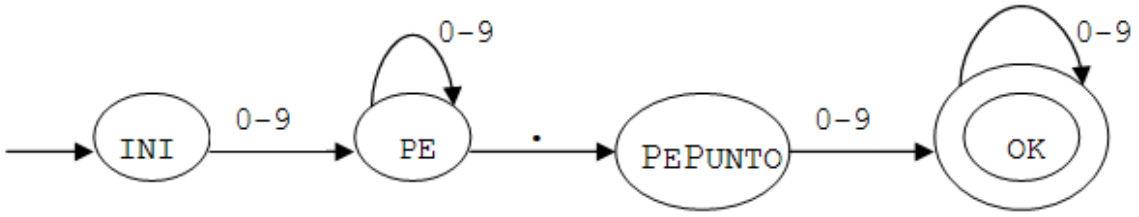
'La cadena "24" no es aceptada:

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \quad [\text{rechaza: PE no es estado final}]$$

'¿Acepta la cadena "24.8."?

$$\text{INI} \xrightarrow{2}$$

Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{\cdot} \text{PEPUNTO} \xrightarrow{8} \text{OK} \quad [\text{acepta}]$$

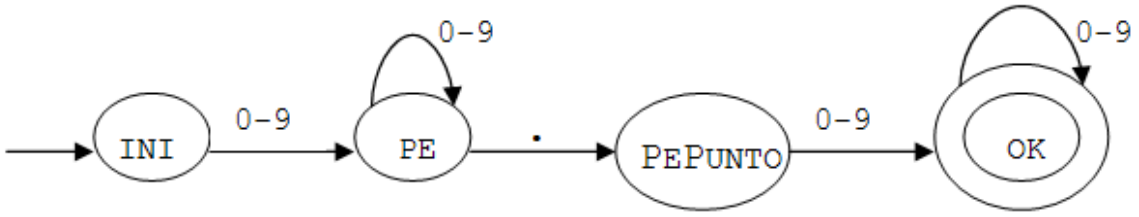
'La cadena "24" no es aceptada:

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \quad [\text{rechaza: PE no es estado final}]$$

'¿Acepta la cadena "24.8."?

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4}$$

Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{.} \text{PEPUNTO} \xrightarrow{8} \text{OK} \quad [\text{acepta}]$$

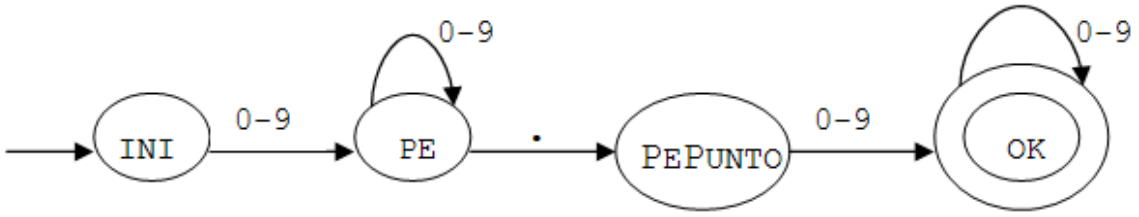
'La cadena "24" no es aceptada:

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \quad [\text{rechaza: PE no es estado final}]$$

'¿Acepta la cadena "24.8."?

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{.}$$

Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{.} \text{PEPUNTO} \xrightarrow{8} \text{OK} \quad [\text{acepta}]$$

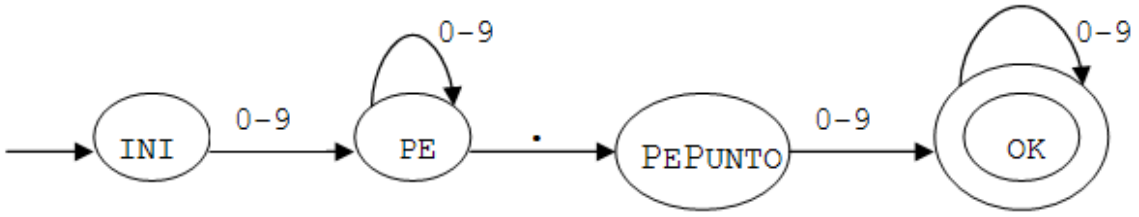
'La cadena "24" no es aceptada:

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \quad [\text{rechaza: PE no es estado final}]$$

'¿Acepta la cadena "24.8."?

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{.} \text{PEPUNTO} \xrightarrow{8}$$

Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{\cdot} \text{PEPUNTO} \xrightarrow{8} \text{OK} \quad [\text{acepta}]$$

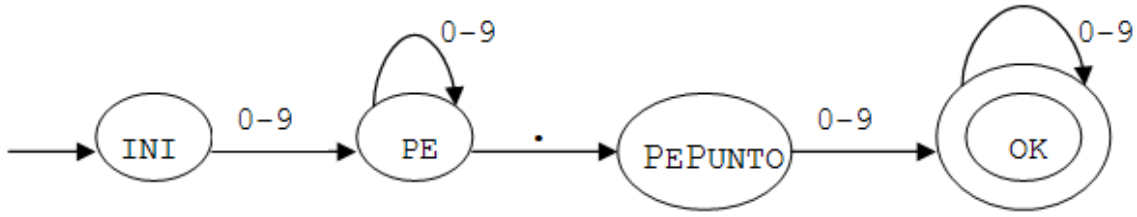
'La cadena "24" no es aceptada:

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \quad [\text{rechaza: PE no es estado final}]$$

'¿Acepta la cadena "24.8."?

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{\cdot} \text{PEPUNTO} \xrightarrow{8} \text{OK}$$

Comprobación mediante el diagrama del AFD: comprobar si existe un camino por el que se lee la cadena y acaba en estado final.



Camino en el diagrama para la cadena "24.8":

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{.} \text{PEPUNTO} \xrightarrow{8} \text{OK} \quad [\text{acepta}]$$

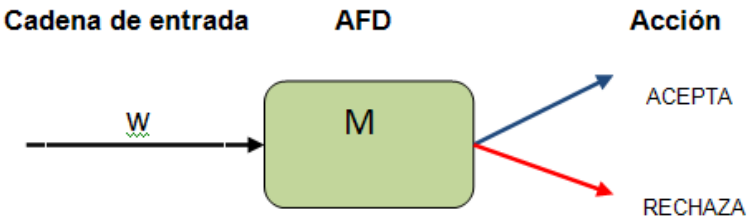
'La cadena "24" no es aceptada:

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \quad [\text{rechaza: PE no es estado final}]$$

'¿Acepta la cadena "24.8."?

$$\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{.} \text{PEPUNTO} \xrightarrow{8} \text{OK} \xrightarrow{.} \quad [\text{rechaza: no termina de leer}]$$

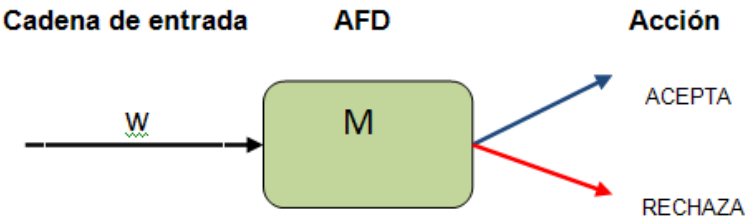
Definición formal de AFD



Un **autómata finito determinista** (AFD) es un modelo de **máquina teórica abstracta** que se representa matemáticamente con una quíntupla $M = (Q, V, \delta, q_0, F)$ donde:

- Q es un **conjunto finito de estados**

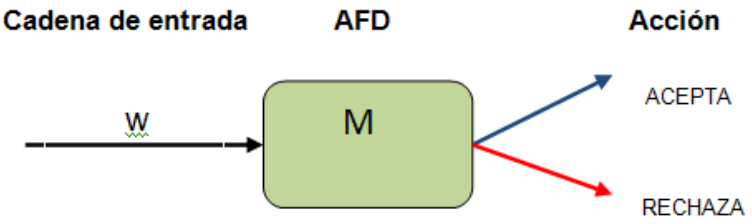
Definición formal de AFD



Un **autómata finito determinista** (AFD) es un modelo de **máquina teórica abstracta** que se representa matemáticamente con una quíntupla $M = (Q, V, \delta, q_0, F)$ donde:

- Q es un **conjunto finito de estados**
- V es el **alfabeto** del autómata

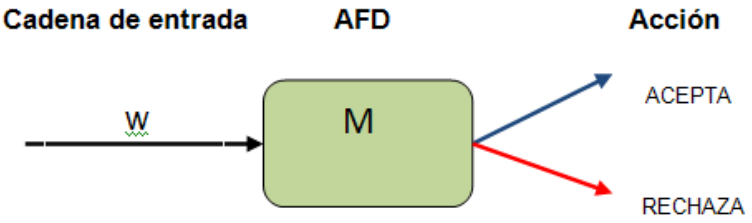
Definición formal de AFD



Un **autómata finito determinista** (AFD) es un modelo de **máquina teórica abstracta** que se representa matemáticamente con una quíntupla $M = (Q, V, \delta, q_0, F)$ donde:

- Q es un **conjunto finito de estados**
- V es el **alfabeto** del autómata
- q_0 es el **estado inicial**

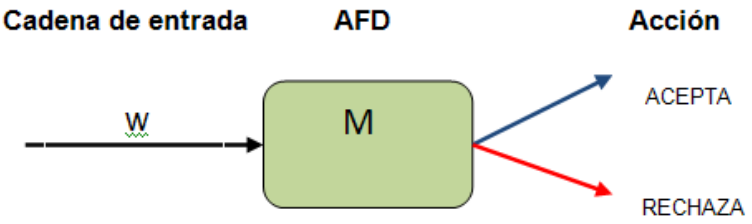
Definición formal de AFD



Un **autómata finito determinista** (AFD) es un modelo de **máquina teórica abstracta** que se representa matemáticamente con una quíntupla $M = (Q, V, \delta, q_0, F)$ donde:

- Q es un **conjunto finito de estados**
- V es el **alfabeto** del autómata
- q_0 es el **estado inicial**
- $F \subseteq Q$ es el **conjunto de estados finales**

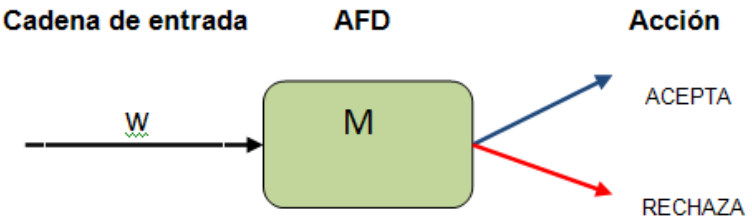
Definición formal de AFD



Un **autómata finito determinista** (AFD) es un modelo de **máquina teórica abstracta** que se representa matemáticamente con una quíntupla $M = (Q, V, \delta, q_0, F)$ donde:

- Q es un **conjunto finito de estados**
- V es el **alfabeto** del autómata
- q_0 es el **estado inicial**
- $F \subseteq Q$ es el **conjunto de estados finales**
- $\delta : Q \times V \longrightarrow Q$ es la **función de transición**.

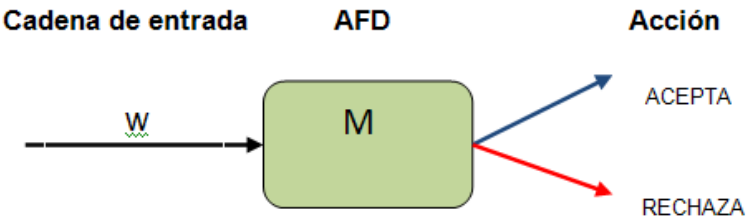
Definición formal de AFD



Un **autómata finito determinista** (AFD) es un modelo de **máquina teórica abstracta** que se representa matemáticamente con una quíntupla $M = (Q, V, \delta, q_0, F)$ donde:

- Q es un **conjunto finito de estados**
- V es el **alfabeto** del autómata
- q_0 es el **estado inicial**
- $F \subseteq Q$ es el **conjunto de estados finales**
- $\delta : Q \times V \longrightarrow Q$ es la **función de transición**.

Definición formal de AFD



Un **autómata finito determinista** (AFD) es un modelo de **máquina teórica abstracta** que se representa matemáticamente con una quíntupla $M = (Q, V, \delta, q_0, F)$ donde:

- Q es un **conjunto finito de estados**
- V es el **alfabeto** del autómata
- q_0 es el **estado inicial**
- $F \subseteq Q$ es el **conjunto de estados finales**
- $\delta : Q \times V \longrightarrow Q$ es la **función de transición**.

Una **regla de transición** (o caso de la función de transición) es del tipo $\delta(\mathbf{q}, \mathbf{a}) = \mathbf{q}'$. Equivale a una instrucción *if-then*:

SI estadoActual==q y simboloActual==a ENTONCES estadoActual=q'

Representación compacta de un AFD

Sea $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ y δ se define matemáticamente:

$$\begin{array}{ll} \delta(q_0, 0) = q_1 & \delta(q_0, 1) = q_0 \\ \delta(q_1, 0) = q_0 & \delta(q_1, 1) = q_1 \end{array}$$

Representación compacta de un AFD

Sea $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ y δ se define matemáticamente:

$$\delta(q_0, 0) = q_1 \qquad \delta(q_0, 1) = q_0$$
$$\delta(q_1, 0) = q_0 \qquad \delta(q_1, 1) = q_1$$

Tabla de transición

δ	0	1
$\rightarrow q_0$	q_1	q_0
$\# \; q_1$	q_0	q_1

Representación compacta de un AFD

Sea $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ y δ se define matemáticamente:

$$\delta(q_0, 0) = q_1 \qquad \delta(q_0, 1) = q_0$$
$$\delta(q_1, 0) = q_0 \qquad \delta(q_1, 1) = q_1$$

Tabla de transición

δ	0	1
$\rightarrow q_0$	q_1	q_0
$\# \; q_1$	q_0	q_1

Representación compacta de un AFD

Sea $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ y δ se define matemáticamente:

$$\begin{array}{ll} \delta(q_0, 0) = q_1 & \delta(q_0, 1) = q_0 \\ \delta(q_1, 0) = q_0 & \delta(q_1, 1) = q_1 \end{array}$$

Tabla de transición

δ	0	1
$\rightarrow q_0$	q_1	q_0
$\# q_1$	q_0	q_1

Representación compacta de un AFD

Sea $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ y δ se define matemáticamente:

$$\delta(q_0, 0) = q_1 \qquad \delta(q_0, 1) = q_0$$
$$\delta(q_1, 0) = q_0 \qquad \delta(q_1, 1) = q_1$$

Tabla de transición

δ	0	1
$\rightarrow q_0$	q_1	q_0
$\# q_1$	q_0	q_1

Representación compacta de un AFD

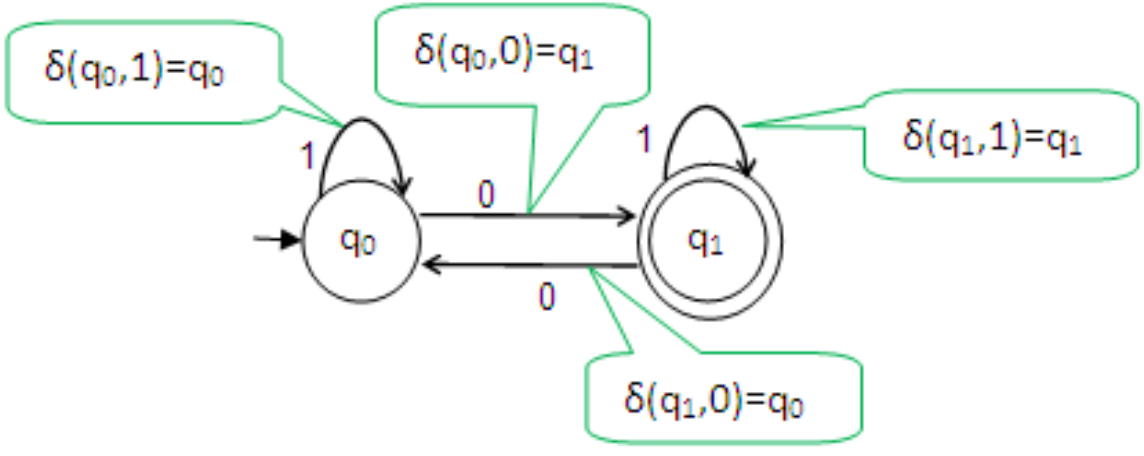
Sea $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ y δ se define matemáticamente:

$$\delta(q_0, 0) = q_1 \qquad \delta(q_0, 1) = q_0$$
$$\delta(q_1, 0) = q_0 \qquad \delta(q_1, 1) = q_1$$

Tabla de transición

δ	0	1
$\rightarrow q_0$	q_1	q_0
$\# q_1$	q_0	q_1

Diagrama de transición



Representación compacta de un AFD

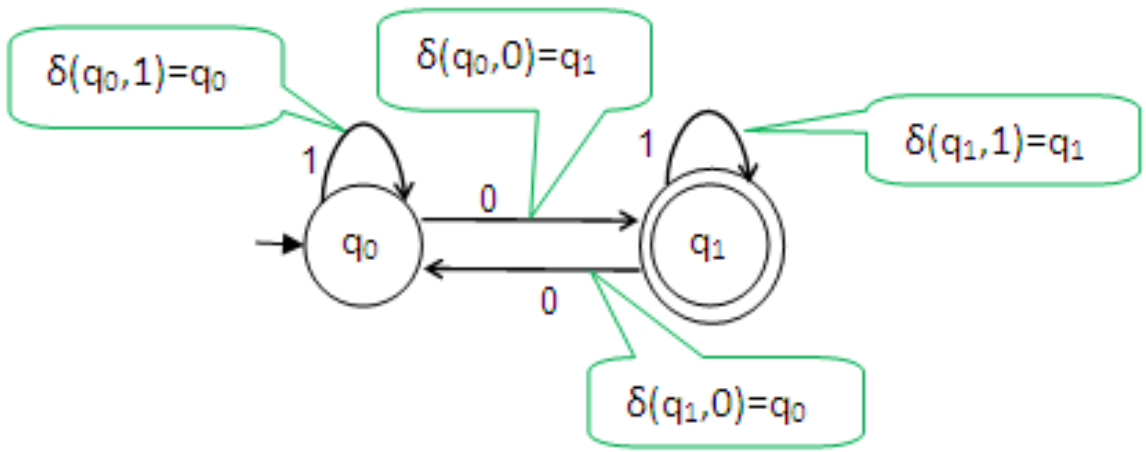
Sea $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ y δ se define matemáticamente:

$$\delta(q_0, 0) = q_1 \qquad \delta(q_0, 1) = q_0$$
$$\delta(q_1, 0) = q_0 \qquad \delta(q_1, 1) = q_1$$

Tabla de transición

δ	0	1
$\rightarrow q_0$	q_1	q_0
$\# \; q_1$	q_0	q_1

Diagrama de transición



Ej.: Acepta **101** por el camino: $q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_1 \quad [q_1 \in F]$

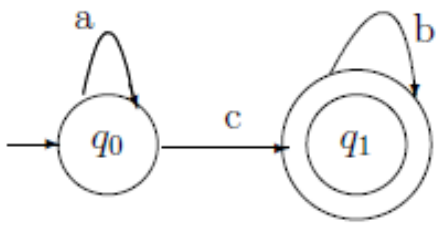
AFDs completos e incompletos

- La función $\delta : Q \times V \longrightarrow Q$ de un *AFD* puede ser parcial (algún caso no definido) o total.
- Se dice que un AFD es **completo** si su función de transición es **total** y es un **incompleto** si δ es **parcial**.
- El algoritmo **completaAFD (M)** (apuntes) obtiene un AFD completo **equivalente** a M.

AFDs completos e incompletos

- La función $\delta : Q \times V \longrightarrow Q$ de un *AFD* puede ser parcial (algún caso no definido) o total.
- Se dice que un AFD es **completo** si su función de transición es **total** y es un **incompleto** si δ es **parcial**.

Ejemplo de transformación de AFD incompleto a completo



AFD incompleto

AFD completo equivalente

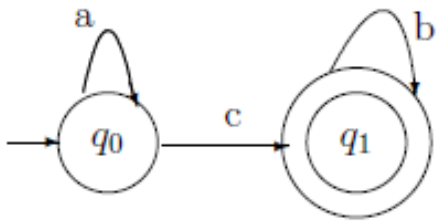
El AFD incompleto rechaza *abc* porque $\delta(q_0, b)$ no está definida:

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b}$$

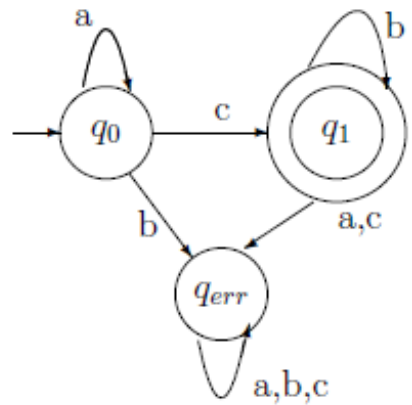
AFDs completos e incompletos

- La función $\delta : Q \times V \longrightarrow Q$ de un *AFD* puede ser parcial (algún caso no definido) o total.
- Se dice que un AFD es **completo** si su función de transición es **total** y es un **incompleto** si δ es **parcial**.

Ejemplo de transformación de AFD incompleto a completo



AFD incompleto



AFD completo equivalente

El AFD completo rechaza *abc* porque q_{err} no es final:

$$q_a \xrightarrow{a} q_0 \xrightarrow{b} q_{err} \xrightarrow{c} q_{err}$$

Funcionamiento general de un AFD

- Al inicio de ejecución el **estado actual** del autómata comienza siendo el inicial y el **símbolo actual** el de más a la izquierda de la ***cadena de entrada***.
- **Realiza un bucle**: cambiar de estado según indica la **regla de transición** correspondiente al estado actual y símbolo actual + leer siguiente símbolo (avanzar apuntador).
- El bucle finaliza cuando no es posible cambiar de estado mediante las reglas de transición.
- Si **se han leído todos los símbolos** (detectado el fin de la entrada) y ***estado actual* es final** entonces ACEPTA la cadena de entrada; en otro caso la RECHAZA.

Restricciones de funcionamiento

- Procesa la cadena símbolo a símbolo y sin retroceder.
- No necesita memoria. Los estados son la memoria de la máquina.

Configuración en un AFD

Una **configuración** de un AFD en cierto instante describe el **estado actual** y la **porción de la cadena de entrada** que le queda por procesar:

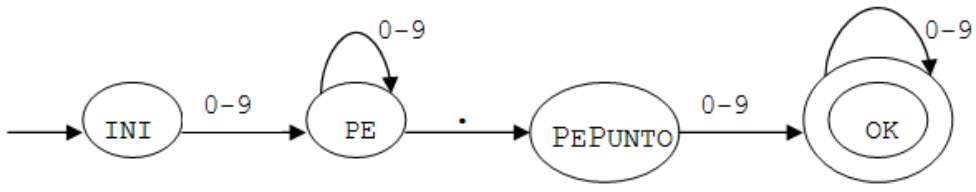
$$(q, x) \text{ con } q \in Q \text{ y } x \in V^*$$

Configuración en un AFD

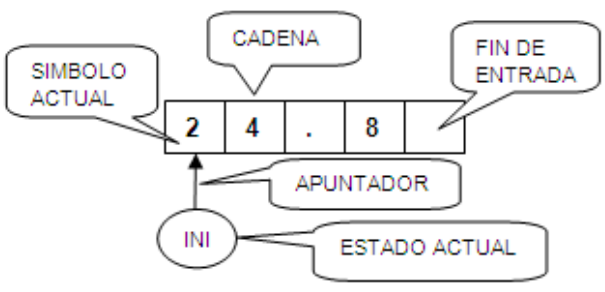
Una **configuración** de un AFD en cierto instante describe el **estado actual** y la **porción de la cadena de entrada** que le queda por procesar:

$$(q, x) \text{ con } q \in Q \text{ y } x \in V^*$$

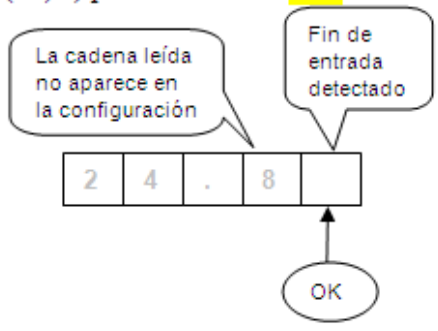
Ejemplo de configuraciones y su representación gráfica



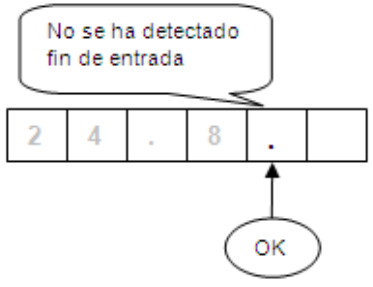
Representación gráfica de la **configuración inicial** (INI, 24.8)



Representación gráfica de la **configuración de aceptación** (OK, λ) para la entrada 24.8



Representación gráfica de la **configuración de parada** (OK, .) no acepta la entrada 24.8.



Paso de cálculo, cálculo y traza de ejecución

- Un **paso de cálculo** consiste en el paso de una configuración a otra por aplicación de una regla de transición.
  Decimos que la configuración (q, az), con $a \in V, z \in V^*$, alcanza en un paso de cálculo la configuración (q', z), y se denota:

$$(q, az) \Rightarrow (q', z)$$

  si y sólo si en la función de transición tenemos la regla $\delta(q, a) = q'$.

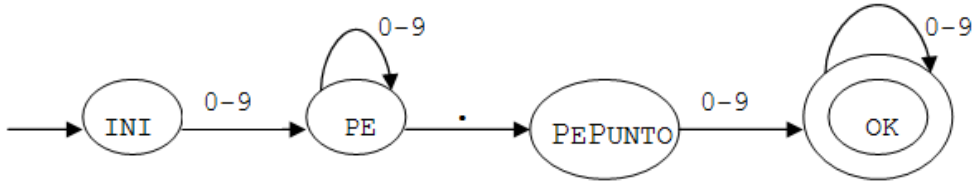
Paso de cálculo, cálculo y traza de ejecución

- Un **paso de cálculo** consiste en el paso de una configuración a otra por aplicación de una regla de transición.
  Decimos que la configuración (q, az), con $a \in V, z \in V^*$, alcanza en un paso de cálculo la configuración (q', z), y se denota:

$$(q, az) \Rightarrow (q', z)$$

  si y sólo si en la función de transición tenemos la regla $\delta(q, a) = q'$.

  Ej.: En el autómata M_{ef}:



  podemos afirmar que: $(\text{INI}, 24.8) \Rightarrow (\text{PE}, 4.8)$, porque tenemos definida la transición $\delta(\text{INI}, 2) = \text{PE}$.

Paso de cálculo, cálculo y traza de ejecución

- Un **paso de cálculo** consiste en el paso de una configuración a otra por aplicación de una regla de transición.
- Un **cálculo** es el proceso de paso de una configuración a otra por aplicación de cero o más reglas de transición. Distinguimos dos casos:
  1. **Cálculo en uno o más pasos.** Se dice que configuración C_0 alcanza la configuración C_n, que se denota $C_0 \Rightarrow^* C_n$, en $n > 0$ pasos si y sólo si existe cálculo consistente en una secuencia de pasos de cálculo del tipo:

     $$C_0 \Rightarrow C_1 \Rightarrow \ldots \Rightarrow C_n$$

     donde cada C_i es una configuración y en el cálculo se aplican $n > 0$ reglas de transición.
  2. **Cálculo en cero pasos.** Por defecto, para cualquier configuración (q, x) se cumple que $(q, x) \Rightarrow^* (q, x)$ (no se aplica ninguna regla).

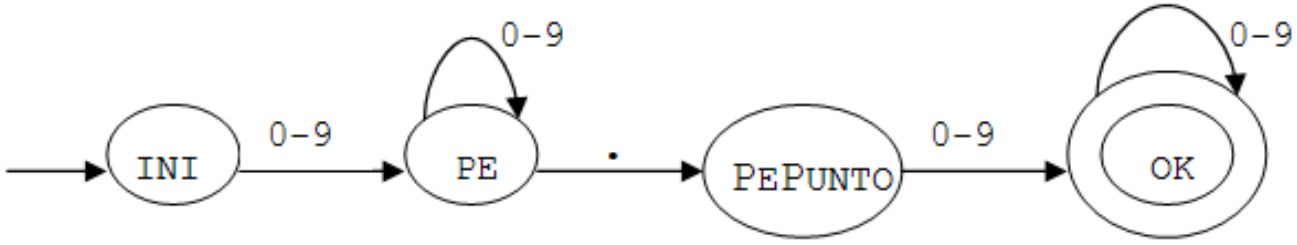
Paso de cálculo, cálculo y traza de ejecución

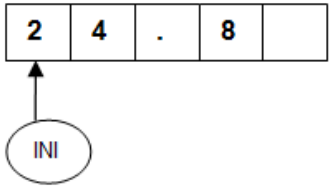
- Un **paso de cálculo** consiste en el paso de una configuración a otra por aplicación de una regla de transición.
- Un **cálculo** es el proceso de paso de una configuración a otra por aplicación de cero o más reglas de transición. Distinguimos dos casos:
  1. **Cálculo en uno o más pasos.** Se dice que configuración C_0 alcanza la configuración C_n, que se denota $C_0 \Rightarrow^* C_n$, en $n > 0$ pasos si y sólo si existe cálculo consistente en una secuencia de pasos de cálculo del tipo:

     $$C_0 \Rightarrow C_1 \Rightarrow \ldots \Rightarrow C_n$$

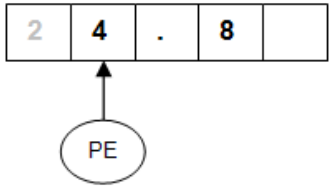
     donde cada C_i es una configuración y en el cálculo se aplican $n > 0$ reglas de transición.
  2. **Cálculo en cero pasos.** Por defecto, para cualquier configuración (q, x) se cumple que $(q, x) \Rightarrow^* (q, x)$ (no se aplica ninguna regla).
- Un **cálculo para una cadena de entrada** w (**o traza de ejecución**) es un cálculo que comienza con la configuración inicial (q_0, w) y acaba en configuración de parada.

Ejemplo de cálculo del autómata M_{ef} con la cadena "24.8"

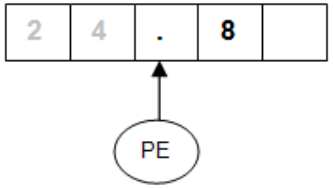




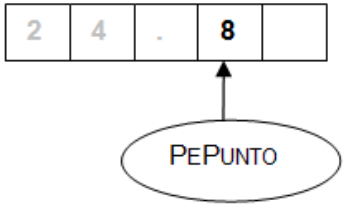
Configur. 0



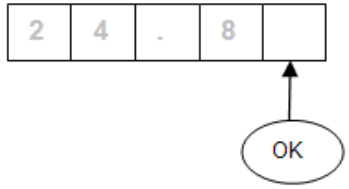
Configur. 1



Configur. 2



Configur. 3



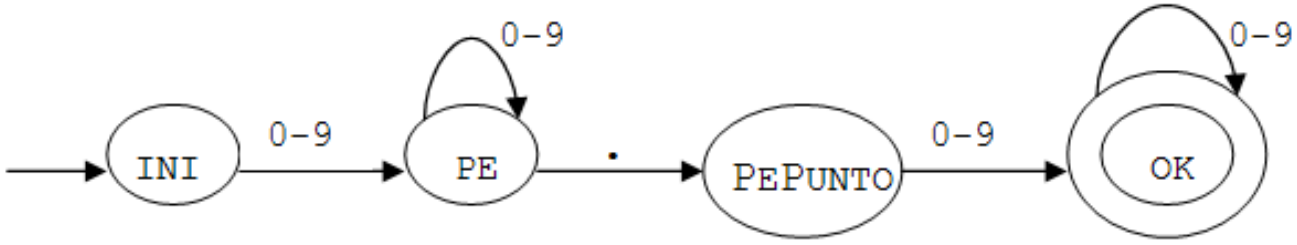
Configur. 4

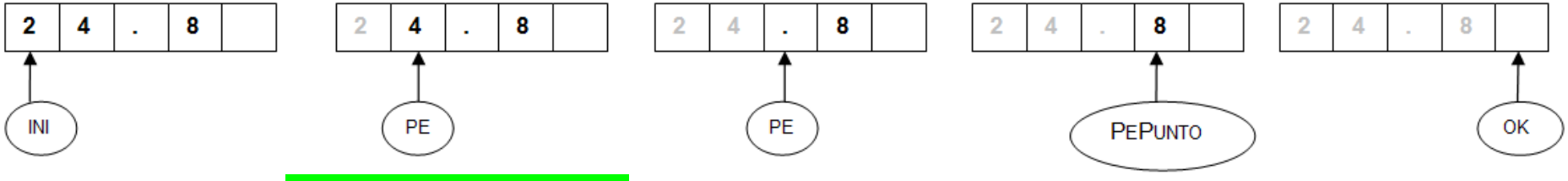
Comprobamos que $(\text{INI}, 24.8) \Rightarrow^* (\text{OK}, \lambda)$ por el siguiente **cálculo**:

paso 1

$$(\text{INI}, 24.8) \overset{\delta(\text{INI}, 2) = \text{PE}}{\Rightarrow}$$

Ejemplo de cálculo del autómata M_{ef} con la cadena "24.8"



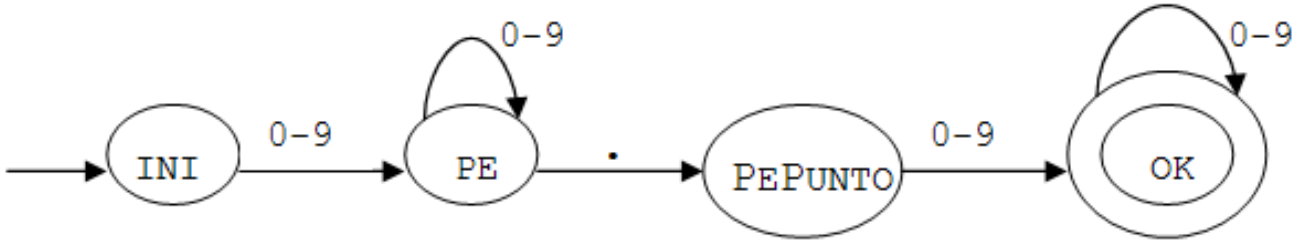


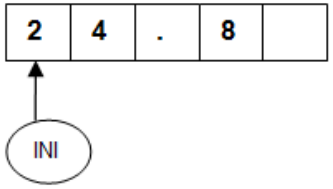
Configur. 0 | Configur. 1 | Configur. 2 | Configur. 3 | Configur. 4

Comprobamos que $(\text{INI}, 24.8) \Rightarrow^* (\text{OK}, \lambda)$ por el siguiente **cálculo**:

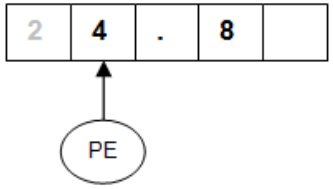
$$(\text{INI}, 24.8) \Rightarrow (\text{PE}, 4.8) \quad \overbrace{\Rightarrow}^{\delta(\text{PE},4)=\text{PE}} \quad \text{paso 2}$$

Ejemplo de cálculo del autómata M_{ef} con la cadena "24.8"

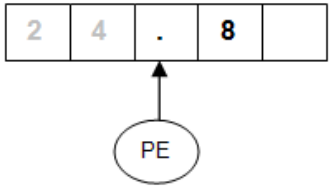




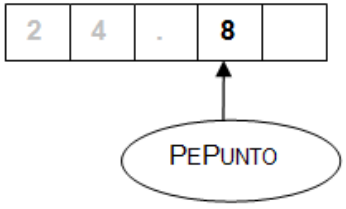
Configur. 0



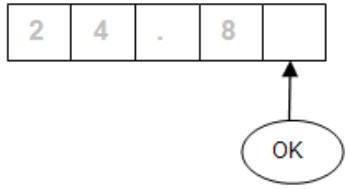
Configur. 1



Configur. 2



Configur. 3



Configur. 4

Comprobamos que $(\text{INI}, 24.8) \Rightarrow^* (\text{OK}, \lambda)$ por el siguiente **cálculo**:

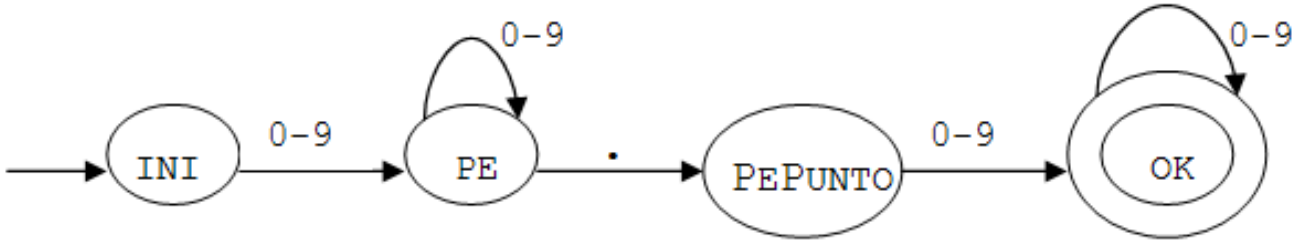
$$(\text{INI}, 24.8) \Rightarrow (\text{PE}, 4.8) \Rightarrow (\text{PE}, .8)$$

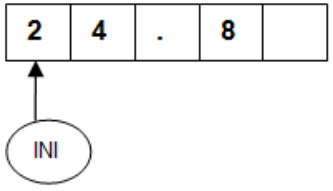
paso 3

$$\delta(\text{PE}, .) = \text{PEPUNTO}$$

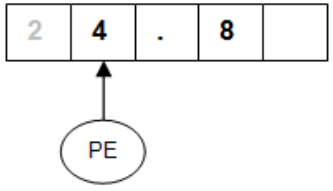
$$\overset{\frown}{\Rightarrow}$$

Ejemplo de cálculo del autómata M_{ef} con la cadena "24.8"

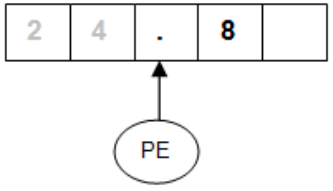




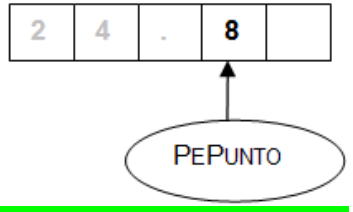
Configur. 0



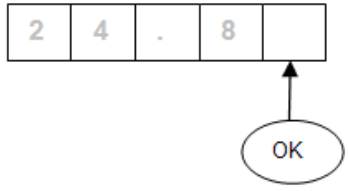
Configur. 1



Configur. 2



Configur. 3

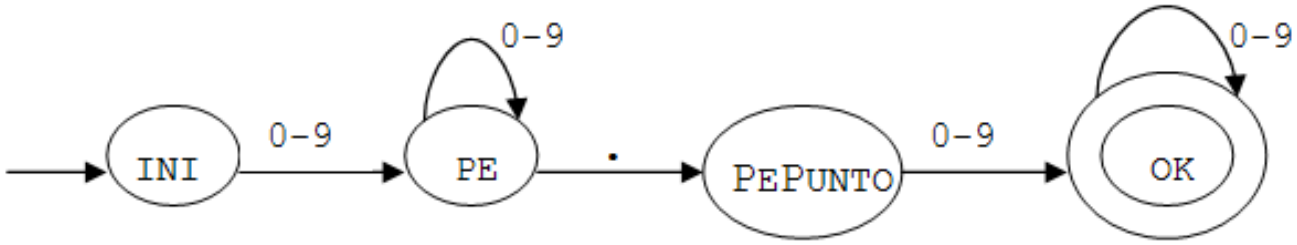


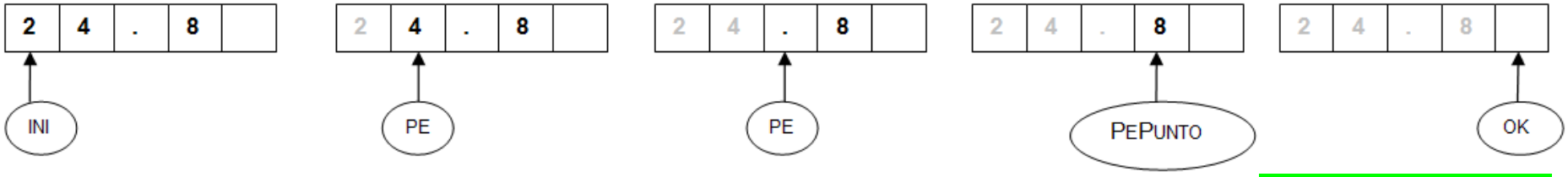
Configur. 4

Comprobamos que $(\text{INI}, 24.8) \Rightarrow^* (\text{OK}, \lambda)$ por el siguiente **cálculo**:

$$(\text{INI}, 24.8) \Rightarrow (\text{PE}, 4.8) \Rightarrow (\text{PE}, .8) \Rightarrow (\text{PEPUNTO}, 8) \overset{\text{paso 4}}{\Rightarrow}$$

Ejemplo de cálculo del autómata M_{ef} con la cadena "24.8"





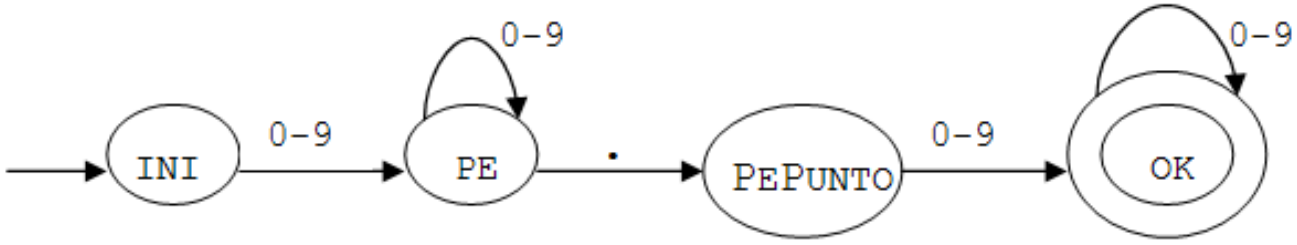
Configur. 0 Configur. 1 Configur. 2 Configur. 3 Configur. 4

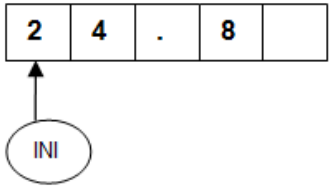
Comprobamos que $(\text{INI}, 24.8) \Rightarrow^* (\text{OK}, \lambda)$ por el siguiente **cálculo**:

$$(\text{INI}, 24.8) \Rightarrow (\text{PE}, 4.8) \Rightarrow (\text{PE}, .8) \Rightarrow (\text{PEPUNTO}, 8) \Rightarrow (\text{OK}, \lambda)$$

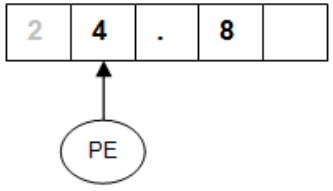
acepta en **tiempo 4 pasos**

Ejemplo de cálculo del autómata M_{ef} con la cadena "24.8"

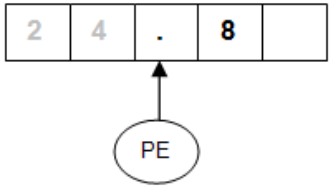




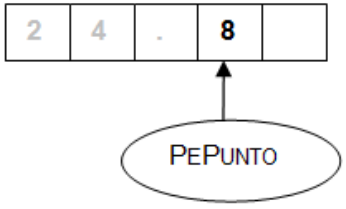
Configur. 0



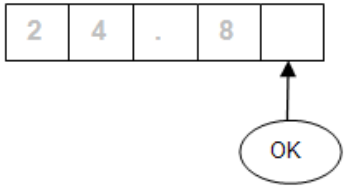
Configur. 1



Configur. 2



Configur. 3



Configur. 4

Comprobamos que $(\text{INI}, 24.8) \Rightarrow^* (\text{OK}, \lambda)$ por el siguiente **cálculo**:

$$(\text{INI}, 24.8) \Rightarrow (\text{PE}, 4.8) \Rightarrow (\text{PE}, .8) \Rightarrow (\text{PEPUNTO}, 8) \Rightarrow (\text{OK}, \lambda)$$

acepta en **tiempo 4 pasos**

Camino en el diagrama: $\text{INI} \xrightarrow{2} \text{PE} \xrightarrow{4} \text{PE} \xrightarrow{\cdot} \text{PEPUNTO} \xrightarrow{8} \text{OK}$

Lenguaje aceptado por un AFD

Dado un autómata finito determinista $M = (Q, V, \delta, q_0, F)$,

- Se dice **formalmente** que una cadena w **es aceptada** por M si y sólo si cumple la condición $(q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F$

Lenguaje aceptado por un AFD

Dado un autómata finito determinista $M = (Q, V, \delta, q_0, F)$,

- Se dice **formalmente** que una cadena w **es aceptada** por M si y sólo si cumple la condición $(q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F$
- El lenguaje aceptado por M, denotado $L(M)$, **se define formalmente**:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

- **Interpretación:** cadenas que pueden leerse siguiendo un camino en el diagrama de transición desde el estado inicial a un estado final.

Lenguaje aceptado por un AFD

Dado un autómata finito determinista $M = (Q, V, \delta, q_0, F)$,

- Se dice **formalmente** que una cadena w **es aceptada** por M si y sólo si cumple la condición $(q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F$
- El lenguaje aceptado por M, denotado $L(M)$, **se define formalmente**:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

- **Interpretación:** cadenas que pueden leerse siguiendo un camino en el diagrama de transición desde el estado inicial a un estado final.
- **La cadena vacía es aceptada** si y sólo si el estado inicial es un estado final, porque $(q_0, \lambda) \Rightarrow^* (q_0, \lambda)$ (cálculo en cero pasos).

Lenguaje aceptado por un AFD

Dado un autómata finito determinista $M = (Q, V, \delta, q_0, F)$,

- Se dice **formalmente** que una cadena w **es aceptada** por M si y sólo si cumple la condición $(q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F$
- El lenguaje aceptado por M, denotado $L(M)$, **se define formalmente**:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

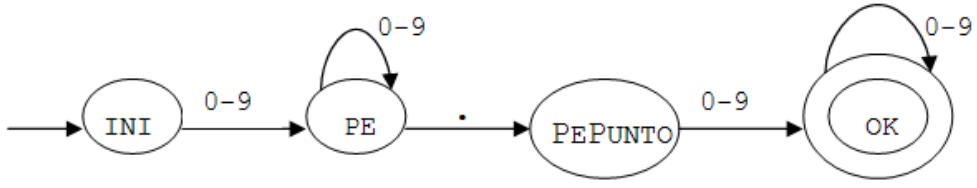
- **Interpretación:** cadenas que pueden leerse siguiendo un camino en el diagrama de transición desde el estado inicial a un estado final.
- **La cadena vacía es aceptada** si y sólo si el estado inicial es un estado final, porque $(q_0, \lambda) \Rightarrow^* (q_0, \lambda)$ (cálculo en cero pasos).

Algoritmo específico de simulación de un AFD

```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.
  estadoActual= INI; simboloActual= nulo; posicion = 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual= leeSimbolo(cadenaEntrada,posicion);
     // Cambio de estado codificando flujo de diagrama
```

Algoritmo específico de simulación de un AFD

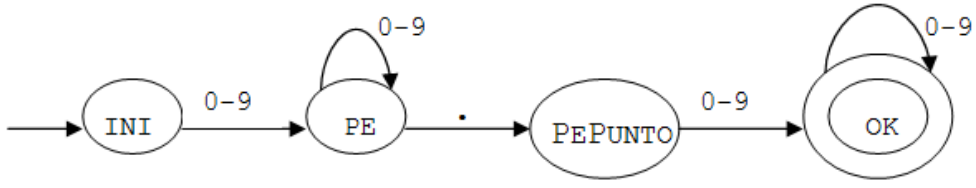
```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.
  estadoActual= INI; simboloActual= nulo; posicion = 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual= leeSimbolo(cadenaEntrada,posicion);
     // Cambio de estado codificando flujo de diagrama
```




```
  SI estadoActual==INI y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
```

Algoritmo específico de simulación de un AFD

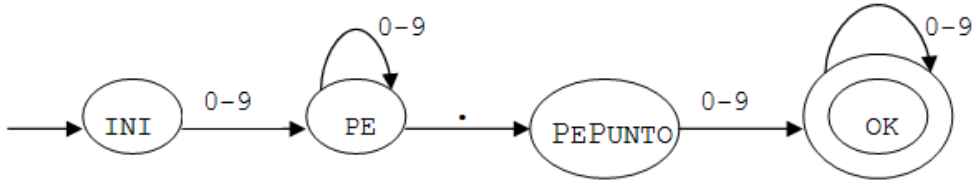
```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.
  estadoActual= INI; simboloActual= nulo; posicion = 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual= leeSimbolo(cadenaEntrada,posicion);
     // Cambio de estado codificando flujo de diagrama
```




```
  SI estadoActual==INI y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
  SI-NO SI estadoActual==PE y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
```

Algoritmo específico de simulación de un AFD

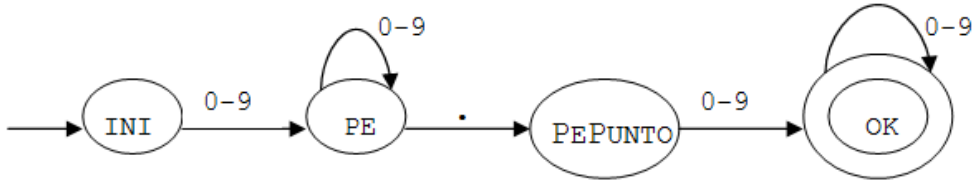
```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.
  estadoActual= INI; simboloActual= nulo; posicion = 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual= leeSimbolo(cadenaEntrada,posicion);
     // Cambio de estado codificando flujo de diagrama
```




```
  SI estadoActual==INI y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
  SI-NO SI estadoActual==PE y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
SI-NO SI estadoActual==PE y simboloActual==‘.’ ENTONCES estadoActual=PEPUNTO;
```

Algoritmo específico de simulación de un AFD

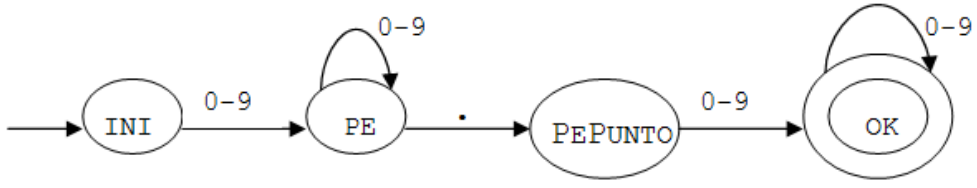
```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.
  estadoActual= INI; simboloActual= nulo; posicion = 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual= leeSimbolo(cadenaEntrada,posicion);
     // Cambio de estado codificando flujo de diagrama
```




```
  SI estadoActual==INI y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
  SI-NO SI estadoActual==PE y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
SI-NO SI estadoActual==PE y simboloActual==‘.’ ENTONCES estadoActual=PEPUNTO;
SI-NO SI estadoActual==PEPUNTO y esDigito(simboloActual) ENTONCES estadoActual=OK;
```

Algoritmo específico de simulación de un AFD

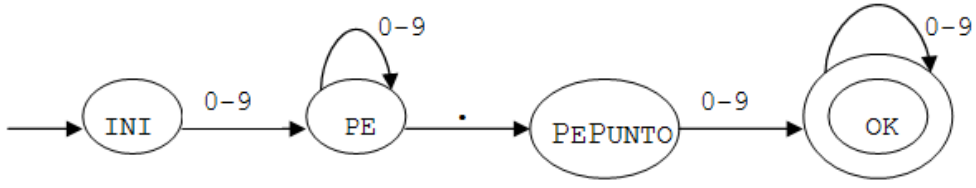
```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.
  estadoActual= INI; simboloActual= nulo; posicion = 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual= leeSimbolo(cadenaEntrada,posicion);
     // Cambio de estado codificando flujo de diagrama
```




```
  SI estadoActual==INI y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
  SI-NO SI estadoActual==PE y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
SI-NO SI estadoActual==PE y simboloActual==‘.’ ENTONCES estadoActual=PEPUNTO;
SI-NO SI estadoActual==PEPUNTO y esDigito(simboloActual) ENTONCES estadoActual=OK;
SI-NO SI estadoActual==OK y esDigito(simboloActual) ENTONCES estado=OK;
```

Algoritmo específico de simulación de un AFD

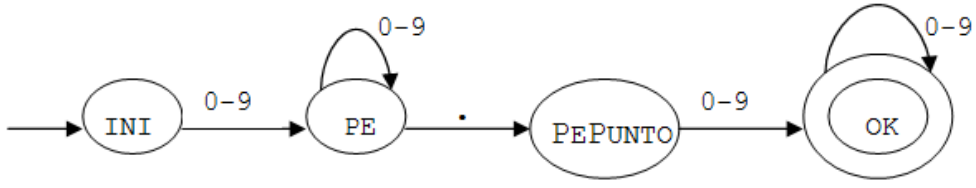
```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.
  estadoActual= INI; simboloActual= nulo; posicion = 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual= leeSimbolo(cadenaEntrada,posicion);
     // Cambio de estado codificando flujo de diagrama
```




```
  SI estadoActual==INI y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
  SI-NO SI estadoActual==PE y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
SI-NO SI estadoActual==PE y simboloActual==‘.’ ENTONCES estadoActual=PEPUNTO;
SI-NO SI estadoActual==PEPUNTO y esDigito(simboloActual) ENTONCES estadoActual=OK;
SI-NO SI estadoActual==OK y esDigito(simboloActual) ENTONCES estado=OK;
  SI-NO DEVUELVE(false); //regla tran. no definida, cadena incorrecta
```

Algoritmo específico de simulación de un AFD

```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.
  estadoActual= INI; simboloActual= nulo; posicion = 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual= leeSimbolo(cadenaEntrada,posicion);
     // Cambio de estado codificando flujo de diagrama
```




```
  SI estadoActual==INI y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
  SI-NO SI estadoActual==PE y esDigito(simboloActual)
  ENTONCES estadoActual=PE;
SI-NO SI estadoActual==PE y simboloActual==‘.’ ENTONCES estadoActual=PEPUNTO;
SI-NO SI estadoActual==PEPUNTO y esDigito(simboloActual) ENTONCES estadoActual=OK;
SI-NO SI estadoActual==OK y esDigito(simboloActual) ENTONCES estado=OK;
  SI-NO DEVUELVE(false); //regla tran. no definida, cadena incorrecta
  posicion=posicion +1;
  FIN-MIENTRAS
```

Algoritmo específico de simulación de un AFD

General

```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.
estadoActual:= INI; simboloActual:= nulo; posicion := 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual= leeSimbolo(cadenaEntrada,posicion);
  SI estadoActual==INI y esDigito(simboloActual) ENTONCES estadoActual:=PE;
  SI-NO SI estadoActual==PE y esDigito(simboloActual) ENTONCES estadoActual:=PE;
  SI-NO SI estadoActual==PE y simboloActual=='.' ENTONCES estadoActual:=PEPUNTO;
  SI-NO SI estadoActual==PEPUNTO y esDigito(simboloActual) ENTONCES
estadoActual=OK;
  SI-NO SI estado==OK y esDigito(simboloActual) ENTONCES estado:=OK;
  SI-NO DEVUELVE(false); //regla tran. no definida, incorrecta
  posicion:=posicion +1;
  FIN-MIENTRAS
 // Comprobación final la cadena se ha leído por completo
 SI esFinal(estadoActual) ENTONCES DEVUELVE(true); //válida
 SI-NO DEVUELVE(false); //cadena incorrecta
```

Orden de complejidad: $O(n)$, siendo $n = |w|$

Algoritmo general de simulación de un AFD

◂ Específico

```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.

  estadoActual= INI; simboloActual:= nulo; posicion := 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual=leeSimbolo(cadenaEntrada,posicion);
//Cambio de estado consultando la tabla de transición
   SI tranDefinida(estadoActual,simboloActual) ENTONCES
     estadoActual:=transicion(estadoActual,simboloActual);
   SI-NO DEVUELVE(false); //cadena incorrecta
   posicion=posicion +1;
  FIN-MIENTRAS
```

Algoritmo general de simulación de un AFD

◂ Específico

```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.

  estadoActual= INI; simboloActual:= nulo; posicion := 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual=leeSimbolo(cadenaEntrada,posicion);
// Cambio de estado consultando la tabla de transición
   SI tranDefinida(estadoActual,simboloActual) ENTONCES
     estadoActual:=transicion(estadoActual,simboloActual);
   SI-NO DEVUELVE(false); //cadena incorrecta
   posicion=posicion +1;
  FIN-MIENTRAS
// Comprobación final la cadena se ha leído por completo
 SI esFinal(estadoActual) ENTONCES DEVUELVE(true); //válida
  SI-NO DEVUELVE(false); //incorrecta
```

Algoritmo general de simulación de un AFD

◂ Específico

```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.

  estadoActual= INI; simboloActual:= nulo; posicion := 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual=leeSimbolo(cadenaEntrada,posicion);
//Cambio de estado consultando la tabla de transición
   SI tranDefinida(estadoActual,simboloActual) ENTONCES
     estadoActual:=transicion(estadoActual,simboloActual);
   SI-NO DEVUELVE(false); //cadena incorrecta
   posicion=posicion +1;
  FIN-MIENTRAS
 //Comprobación final la cadena se ha leído por completo
 SI esFinal(estadoActual) ENTONCES DEVUELVE(true); //válida
  SI-NO DEVUELVE(false); //incorrecta
```

Orden de complejidad: $O(n)$, siendo $n = |w|$

Algoritmo general de simulación de un AFD

◂ Específico

```
FUNCION esValida(cadenaEntrada) DEVUELVE true/false.

  estadoActual= INI; simboloActual:= nulo; posicion := 0;
  MIENTRAS (noFin(cadenaEntrada,posicion))
   simboloActual=leeSimbolo(cadenaEntrada,posicion);
//Cambio de estado consultando la tabla de transición
   SI tranDefinida(estadoActual,simboloActual) ENTONCES
     estadoActual:=transicion(estadoActual,simboloActual);
   SI-NO DEVUELVE(false); //cadena incorrecta
   posicion=posicion +1;
  FIN-MIENTRAS
 //Comprobación final la cadena se ha leído por completo
 SI esFinal(estadoActual) ENTONCES DEVUELVE(true); //válida
  SI-NO DEVUELVE(false); //incorrecta
```

Orden de complejidad: $O(n)$, siendo $n = |w|$

Análisis de AFDs

Objetivo de un problema de análisis de AFD:

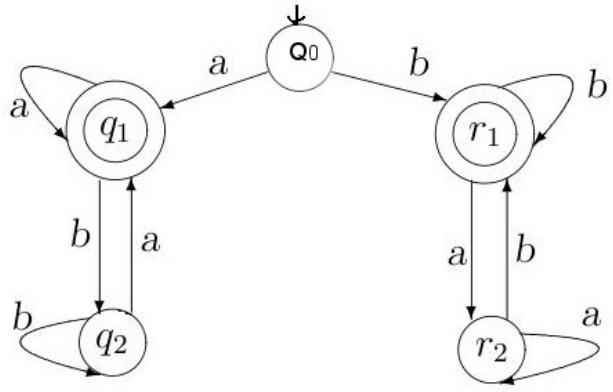
$\Rightarrow$ **Obtener el lenguaje que acepta** un AFD describiéndolo **de forma explícita** con palabras (de manera informal) o matemáticamente por comprensión (formalmente).

Estrategia para resolverlo:

- **Analizar todos los caminos** que llevan desde el estado inicial a un estado final.
- **Establecer hipótesis:** se describe el lenguaje H que supuestamente acepta el autómata M.
- **Comprobar hipótesis:** justificar (o demostrar) que $H = L(M)$, o lo que es lo mismo:
  1. **Que M no es demasiado "estricto"** (acepta todas las cadenas de H) . Formalmente supone probar que $H \subseteq L(M)$.
  2. **Que M no es demasiado "general"** (rechaza cadenas que no pertenecen a H). Formalmente supone probar que $L(M) \subseteq H$.

Ejemplo de análisis de AFD

Sea el autómata M_{etm}:

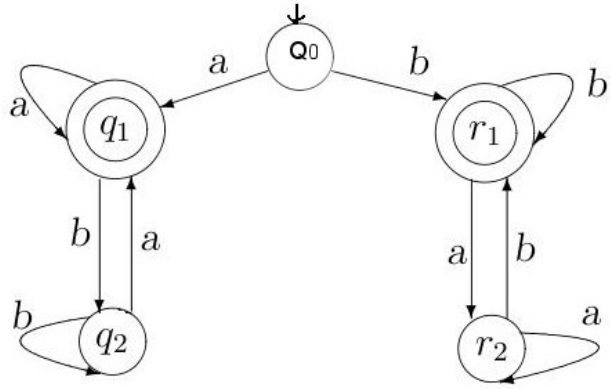


Hipótesis: Deducimos que $L(M) = H$ donde H=?

-
-
-
-

Ejemplo de análisis de AFD

Sea el autómata M_{etm}:

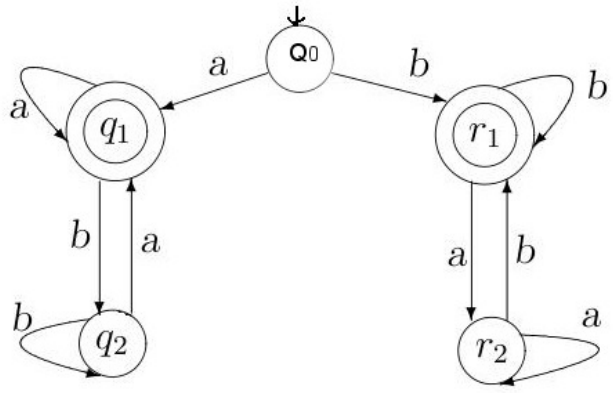


Hipótesis: $H = \{w \in \{a, b\}^* \mid w = a \vee w = b \ldots\}$

- Casos $w = a$ y $w = b$. Son aceptadas:
  $$(Q_0, a) \Rightarrow (q_1, \lambda) \quad \text{y} \quad (Q_0, b) \Rightarrow (r_1, \lambda)$$
-
-
-

Ejemplo de análisis de AFD

Sea el autómata M_{etm}:

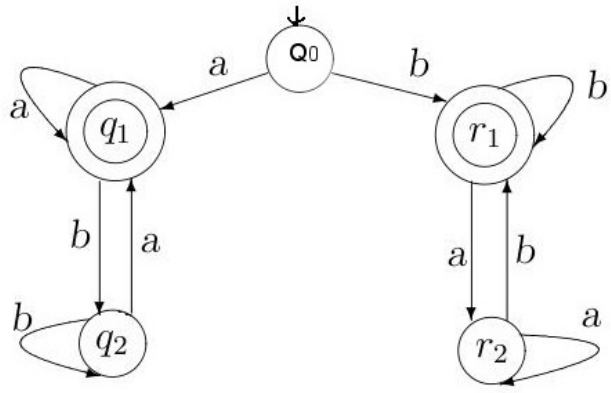


Hipótesis: $H = \{w \in \{a,b\}^* \mid w = a \vee w = b \vee w = aa \vee w = bb \ldots\}$

- Casos $w = a$ y $w = b$. Son aceptadas:
  $$(Q_0, a) \Rightarrow (q_1, \lambda) \quad \text{y} \quad (Q_0, b) \Rightarrow (r_1, \lambda)$$
- Casos $w = aa$ y $w = bb$. Son aceptadas:
  $$(Q_0, aa) \Rightarrow (q_1, a) \Rightarrow (q_1, \lambda) \quad \text{y} \quad (Q_0, bb) \Rightarrow (r_1, b) \Rightarrow (r_1, \lambda)$$
-
-

Ejemplo de análisis de AFD

Sea el autómata M_{etm}:

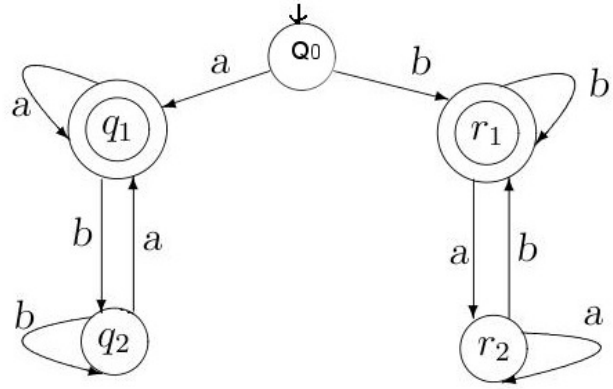


Hipótesis: $L(M) = H = \{w \in \{a,b\}^* \mid w = a \vee w = b \vee w = aza \vee w = bzb, z \in \{a,b\}^*\}$

- Casos $w = a$ y $w = b$. Son aceptadas:
$$(Q_0, a) \Rightarrow (q_1, \lambda) \quad \text{y} \quad (Q_0, b) \Rightarrow (r_1, \lambda)$$
- Casos $w = aa$ y $w = bb$. Son aceptadas:
$$(Q_0, aa) \Rightarrow (q_1, a) \Rightarrow (q_1, \lambda) \quad \text{y} \quad (Q_0, bb) \Rightarrow (r_1, b) \Rightarrow (r_1, \lambda)$$
- Generalizando: cadenas tipo $w = aza$ y $w = bzb \ldots$ son aceptadas.
-

Ejemplo de análisis de AFD

Sea el autómata M_{etm}:



Tesis: $L(M) = H = \{w \in \{a,b\}^* \mid w = a \vee w = b \vee w = aza \vee w = bzb, z \in \{a,b\}^*\}$

Cadenas de a's/b's que **comienzan y terminan con el mismo símbolo.**

- Casos $w = a$ y $w = b$. Son aceptadas:
$$(Q_0, a) \Rightarrow (q_1, \lambda) \quad \text{y} \quad (Q_0, b) \Rightarrow (r_1, \lambda)$$
- Casos $w = aa$ y $w = bb$. Son aceptadas:
$$(Q_0, aa) \Rightarrow (q_1, a) \Rightarrow (q_1, \lambda) \quad \text{y} \quad (Q_0, bb) \Rightarrow (r_1, b) \Rightarrow (r_1, \lambda)$$
- Generalizando: cadenas tipo $w = aza$ y $w = bzb \ldots$ son aceptadas.
- $\Rightarrow$ M no es demasiado estricto: acepta todas las cadenas de H.
- $\Rightarrow$ M no es demasiado general; rechaza cadenas que NO comienzan y terminan con el mismo símbolo.

Diseño de AFDs

Objetivo de un problema de diseño de AFDs:

$\Rightarrow$ **obtener un autómata finito** M que acepte cadenas de cierto lenguaje formal L $\Rightarrow$ que sirva para comprobar si una cadena tiene un formato adecuado o cumple cierta propiedad.

Diseño de AFDs

Objetivo de un problema de diseño de AFDs:

$\Rightarrow$ **obtener un autómata finito** M que acepte cadenas de cierto lenguaje formal L $\Rightarrow$ que sirva para comprobar si una cadena tiene un formato adecuado o cumple cierta propiedad.

Estrategia para resolverlo:

- Descubrir qué estados se necesitan para decidir si una cadena es válida o no, distinguir qué es lo esencial que se debe recordar durante el procesamiento.
- Dibujar diagrama de transición de M.
- Comprobar la corrección del diseño. Supone comprobar:
  1. **Que M no es demasiado estricto:** acepta todas las cadenas que se describen en la especificación del lenguaje L.
  2. **Que M no es demasiado general:** rechaza todas cadenas que NO son del tipo especificado por L.

  $\Rightarrow$ Realizar **test de prueba** que cubra los distintos casos de cadenas que deben ser aceptadas o rechazadas.

Ejemplo de diseño de AFDs

Ej.: CASO: autómata para "conteo" de símbolos.

Ejemplo de diseño de AFDs

Ej.: CASO: autómata para "conteo" de símbolos.

- **Diseñar un afd que acepte el lenguaje:**

$$L_{a3aa} = \{w \in \{a\}^* \mid |w| = 3n + 2, n \geq 0\} \text{ o equivalentemente:}$$

$$L_{a3aa} = \{(aaa)^n aa \mid n \geq 0\}$$

- **Idea:** necesitamos estados que sirvan para recordar el resto de la longitud del prefijo leído entre 3.

Ejemplo de diseño de AFDs

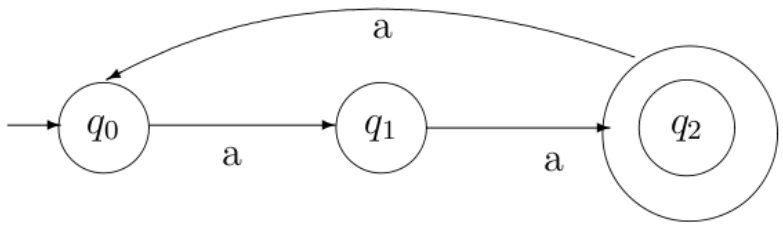
Ej.: CASO: autómata para "conteo" de símbolos.

- **Diseñar un afd que acepte el lenguaje:**

$$L_{a3aa} = \{w \in \{a\}^* \mid |w| = 3n + 2, n \geq 0\} \text{ o equivalentemente:}$$

$$L_{a3aa} = \{(aaa)^n aa \mid n \geq 0\}$$

- **Idea:** necesitamos estados que sirvan para recordar el resto de la longitud del prefijo leído entre 3.
  Estado q_0: resto 0. Recuerda que se han leído cero $a's$, tres $a's \ldots$, y en general un número de $a's$ múltiplo de 3.
  Estado q_1: resto 1. Se han leído una a, cuatro $a's$,... y en general $3n + 1$ $a's$.
  Estado q_2: resto 2. Se han leído dos $a's$, cinco $a's$,... y en general $3n + 2$ $a's$.



- $\rightrightarrows$ **Realizar test de prueba** para comprobar que $L(M) = L_{a3aa}$

Ejemplo de diseño de AFD

Ej.: CASO: autómata para "comprobar si una cadena contiene cierta subcadena".

- Diseñar un AFD que para comprobar si una cadena de a's/b's contiene la subcadena *ab*. El lenguaje es:

$$L_{ab} = \{xaby \mid x, y \in \{a, b\}^*\}$$

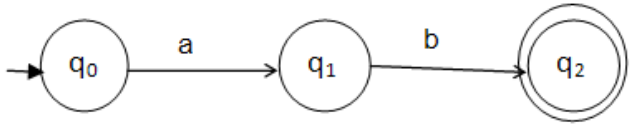
Ejemplo de diseño de AFD

Ej.: CASO: autómata para "comprobar si una cadena contiene cierta subcadena".

- Diseñar un AFD que para comprobar si una cadena de a's/b's contiene la subcadena *ab*. El lenguaje es:

$$L_{ab} = \{xaby \mid x, y \in \{a, b\}^*\}$$

- **Idea:** incluimos los estados y transiciones necesarias para aceptar la cadena válida de menor longitud, que es *ab*.



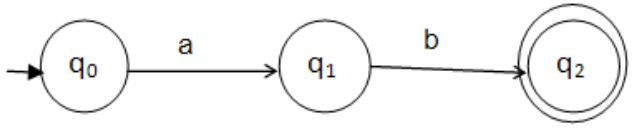
Ejemplo de diseño de AFD

Ej.: CASO: autómata para "comprobar si una cadena contiene cierta subcadena".

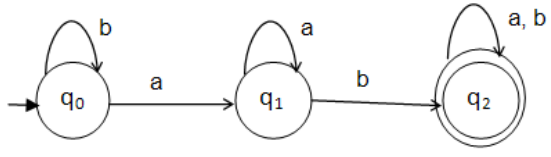
- Diseñar un AFD que para comprobar si una cadena de a's/b's contiene la subcadena *ab*. El lenguaje es:

$$L_{ab} = \{xaby \mid x, y \in \{a, b\}^*\}$$

- **Idea:** incluimos los estados y transiciones necesarias para aceptar la cadena válida de menor longitud, que es *ab*.



Añadimos transiciones para completar el autómata para que se puedan leer cadenas de cualquier longitud:



- **Realizar test de prueba** para comprobar que $L(M) = L_{ab}$

Minimización de AFDs $\rightrightarrows$ reducción del número de estados

- Dos autómatas M y M' son **equivalentes**, y se denota $\mathbf{M \equiv M'}$, si y solo si $\mathbf{L(M) = L(M')}$ (aceptan las mismas cadenas).

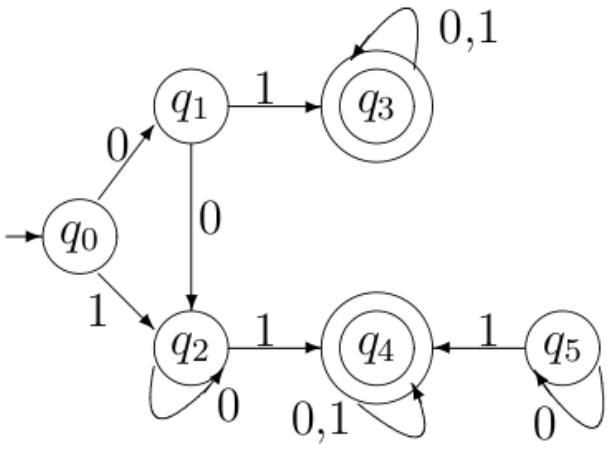
Minimización de AFDs $\rightrightarrows$ reducción del número de estados

- Dos autómatas M y M' son **equivalentes**, y se denota $\mathbf{M \equiv M'}$, si y solo si $\mathbf{L(M) = L(M')}$ (aceptan las mismas cadenas).
- Un *AFD* es **mínimo** si no existe otro *AFD* completo equivalente con menos estados. Debe cumplir dos condiciones:
  1. Todos los estados son **accesibles**.
  2. Todos los estados son **distinguibles**.

Minimización de AFDs $\rightrightarrows$ reducción del número de estados

- Dos autómatas M y M' son **equivalentes**, y se denota $\mathbf{M \equiv M'}$, si y solo si $\mathbf{L(M) = L(M')}$ (aceptan las mismas cadenas).
- Un *AFD* es **mínimo** si no existe otro *AFD* completo equivalente con menos estados. Debe cumplir dos condiciones:
  1. Todos los estados son **accesibles**.
  2. Todos los estados son **distinguibles**.

Ejemplo de AFDs equivalentes, mejor el mínimo

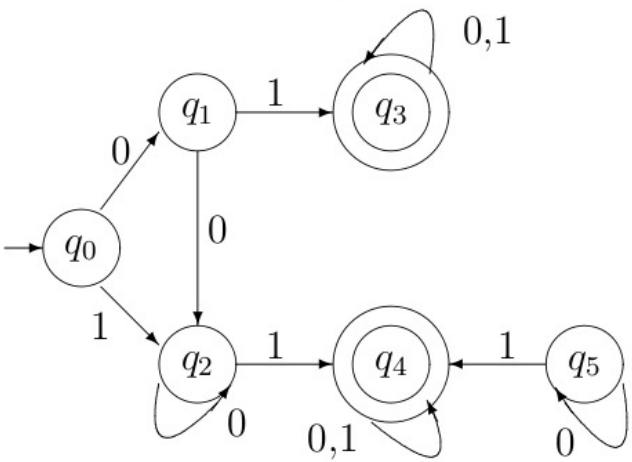


Minimización de AFDs $\rightrightarrows$ reducción del número de estados

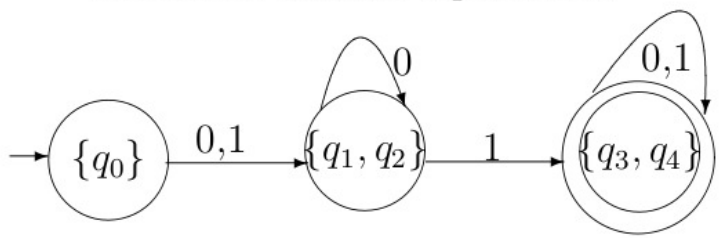
- Dos autómatas M y M' son **equivalentes**, y se denota $\mathbf{M \equiv M'}$, si y solo si $\mathbf{L(M) = L(M')}$ (aceptan las mismas cadenas).
- Un *AFD* es **mínimo** si no existe otro *AFD* completo equivalente con menos estados. Debe cumplir dos condiciones:
  1. Todos los estados son **accesibles**.
  2. Todos los estados son **distinguibles**.

Ejemplo de AFDs equivalentes, mejor el mínimo

Autómata original



Autómata mínimo equivalente



Paso previo: eliminación de estados inaccesibles

Un estado q de un autómata $M = (Q, V, \delta, q_0, F)$ es **accesible** si y sólo si existe una cadena $x \in V^*$ tal que $(q_0, x) \Rightarrow^* (q, \lambda)$. En otro caso el estado es **inaccesible**.

Paso previo: eliminación de estados inaccesibles

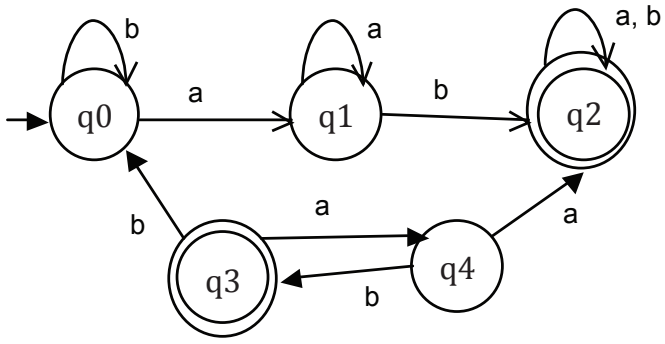
Un estado q de un autómata $M = (Q, V, \delta, q_0, F)$ es **accesible** si y sólo si existe una cadena $x \in V^*$ tal que $(q_0, x) \Rightarrow^* (q, \lambda)$. En otro caso el estado es **inaccesible**.

FUNCIÓN **eliminaInaccesibles(M)**

ENTRADA: un autómata finito M

DEVUELVE: un autómata equivalente sin estados inaccesibles

Ejemplo de eliminación de estados inaccesibles



Paso previo: eliminación de estados inaccesibles

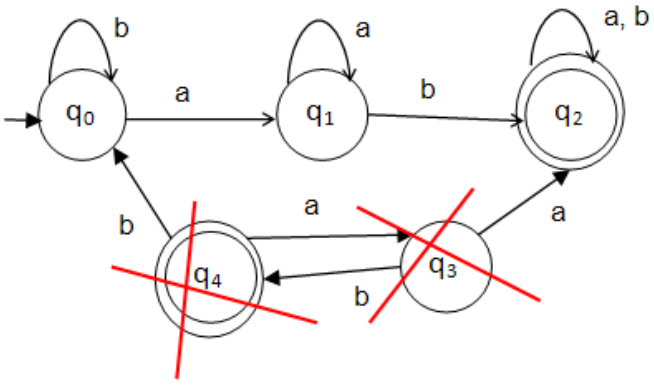
Un estado q de un autómata $M = (Q, V, \delta, q_0, F)$ es **accesible** si y sólo si existe una cadena $x \in V^*$ tal que $(q_0, x) \Rightarrow^* (q, \lambda)$. En otro caso el estado es **inaccesible**.

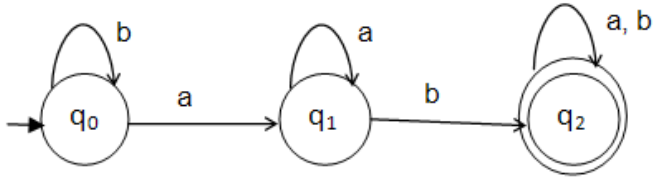
FUNCIÓN **eliminaInaccesibles(M)**

ENTRADA: un autómata finito M

DEVUELVE: un autómata equivalente sin estados inaccesibles

Ejemplo de eliminación de estados inaccesibles





Objetivo: agrupar los estados equivalentes

Dos estados q_i, q_j de un AFD son **equivalentes**, y se denota $q_i \equiv q_j$, sii cumplen la siguiente **condición de equivalencia**:

$$\forall\, x \in V^* \text{ SI } (q_i, x) \Rightarrow^* (q_i', \lambda) \text{ y } (q_j, x) \Rightarrow^* (q_j', \lambda)$$
$$\text{ENTONCES } (q_i' \in F \wedge q_j' \in F) \vee (q_i' \notin F \wedge q_j' \notin F)$$

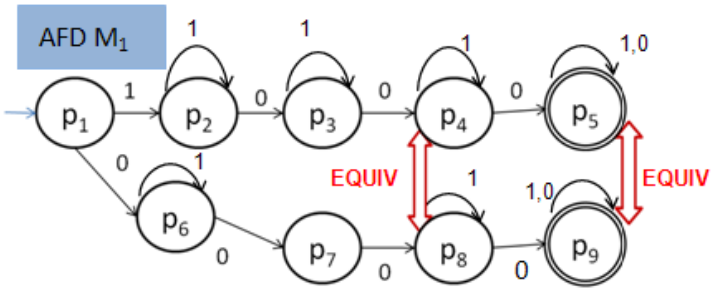
Objetivo: agrupar los estados equivalentes

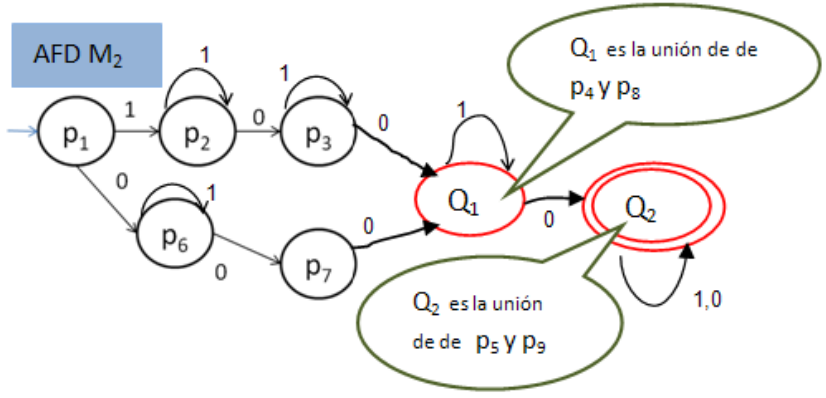
Dos estados q_i, q_j de un AFD son **equivalentes**, y se denota $q_i \equiv q_j$, sii cumplen la siguiente **condición de equivalencia**:

$$\forall\, x \in V^* \text{ SI } (q_i, x) \Rightarrow^* (q_i', \lambda) \text{ y } (q_j, x) \Rightarrow^* (q_j', \lambda)$$
$$\text{ENTONCES } (q_i' \in F \wedge q_j' \in F) \vee (q_i' \notin F \wedge q_j' \notin F)$$

Idea del proceso de minimización: los estados equivalentes representan situaciones semejantes en el procesamiento de una cadena $\rightrightarrows$ **se pueden unir en un único estado** sin que cambie el lenguaje $\rightrightarrows$ **se reduce el número de estados** del autómata.

Ejemplo de estados equivalentes y su agrupación





Deducción de estados distinguibles

Dos estados q_i, q_j de un AFD $M = (Q, V, \delta, q_0, F)$ son **distinguibles** si no son equivalentes, o sea, cumplen la siguiente **condición de distinguibilidad:**

$$\exists\, x \in V^* \text{ tal que } \text{SI } (q_i, x) \Rightarrow^* (q_i', \lambda) \text{ y } (q_j, x) \Rightarrow^* (q_j', \lambda)$$
$$\text{ENTONCES } (q_i' \in F \wedge q_j' \notin F) \vee (q_i' \notin F \wedge q_j' \in F)$$

En ese caso se dice que la cadena x **distingue** a q_i y a q_j.

Deducción de estados distinguibles

Dos estados q_i, q_j de un AFD $M = (Q, V, \delta, q_0, F)$ son **distinguibles** si no son equivalentes, o sea, cumplen la siguiente **condición de distinguibilidad:**

$$\exists\, x \in V^* \text{ tal que } \text{SI } (q_i, x) \Rightarrow^* (q_i', \lambda) \text{ y } (q_j, x) \Rightarrow^* (q_j', \lambda)$$
$$\text{ENTONCES } (q_i' \in F \wedge q_j' \notin F) \vee (q_i' \notin F \wedge q_j' \in F)$$

En ese caso se dice que la cadena x **distingue** a q_i y a q_j.

¿Cómo se averigua si dos estados son distinguibles en un AFD?

1. Partir de un AFD completo.
2. **Regla base:** si en una pareja de estados (q_i, q_j) uno es final y el otro no entonces los estados q_i, q_j **son distinguibles.** (la cadena $x = \lambda$ los distingue).

Deducción de estados distinguibles

Dos estados q_i, q_j de un AFD $M = (Q, V, \delta, q_0, F)$ son **distinguibles** si no son equivalentes, o sea, cumplen la siguiente **condición de distinguibilidad:**

$$\exists\, x \in V^* \text{ tal que } \text{SI } (q_i, x) \Rightarrow^* (q_i', \lambda) \text{ y } (q_j, x) \Rightarrow^* (q_j', \lambda)$$
$$\text{ENTONCES } (q_i' \in F \wedge q_j' \notin F) \vee (q_i' \notin F \wedge q_j' \in F)$$

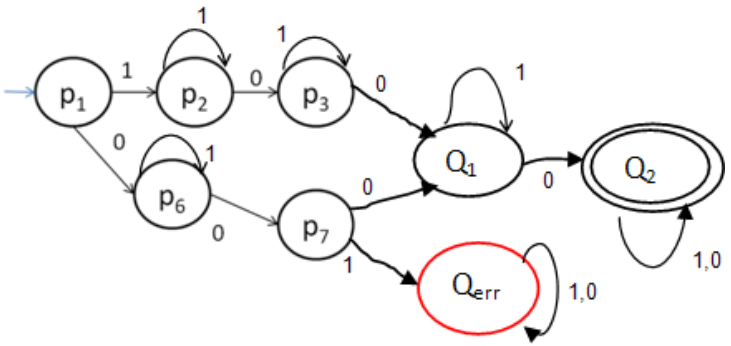
En ese caso se dice que la cadena x **distingue** a q_i y a q_j.

¿Cómo se averigua si dos estados son distinguibles en un AFD?

1. Partir de un AFD completo.
2. **Regla base:** si en una pareja de estados (q_i, q_j) uno es final y el otro no entonces los estados q_i, q_j **son distinguibles.** (la cadena $x = \lambda$ los distingue).
3. **Regla deductiva:** supongamos que $\delta(q_i, a) = q_i'$ y $\delta(q_j, a) = q_j'$, para cierto símbolo $a \in V$, y se sabe que q_i', q_j' son distinguibles, entonces se deduce que q_i y q_j son distinguibles. (**Justificación**: si x que distingue a $q_i', q_j' \Longrightarrow$ la cadena ax distingue a q_i y q_j.)

¿Cómo se averigua si dos estados son distinguibles?

1. Partir de un AFD completo.
2. **Regla base:** si en (q_i, q_j) uno es final y el otro no entonces los estados q_i, q_j **son distinguibles.**
3. **Regla deductiva:** si $\delta(q_i, a) = q_i'$ y $\delta(q_j, a) = q_j'$ y q_i', q_j' son distinguibles, entonces q_i y q_j son distinguibles.

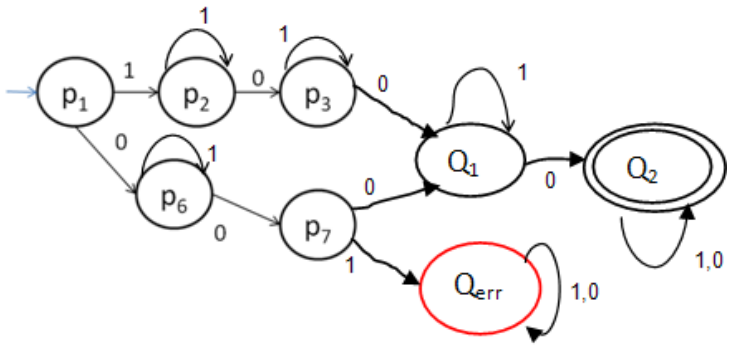


Ej.:

1. Q_2 y Q_{err} son distinguibles (**base**, el AFD debe ser completo).

¿Cómo se averigua si dos estados son distinguibles?

1. Partir de un AFD completo.
2. **Regla base:** si en (q_i, q_j) uno es final y el otro no entonces los estados q_i, q_j **son distinguibles.**
3. **Regla deductiva:** si $\delta(q_i, a) = q_i'$ y $\delta(q_j, a) = q_j'$ y q_i', q_j' son distinguibles, entonces q_i y q_j son distinguibles.

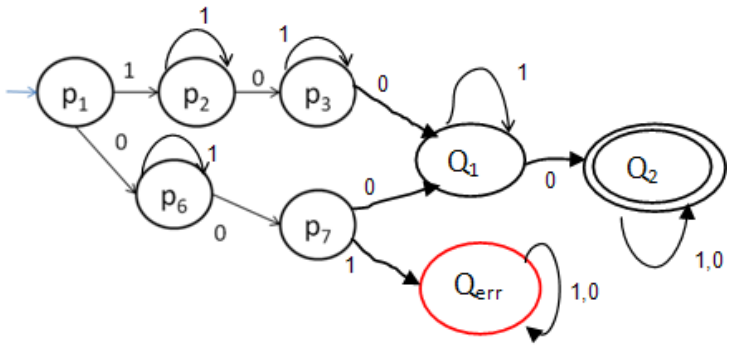


Ej.:

1. Q_2 y Q_{err} son distinguibles (**base**, el AFD debe ser completo).
2. Como $\delta(Q_1, 0) = Q_2$ y $\delta(Q_{err}, 0) = Q_{err}$ y (Q_2, Q_{err}) son distinguibles, entonces Q_1 y Q_{err} son distinguibles.

¿Cómo se averigua si dos estados son distinguibles?

1. Partir de un AFD completo.
2. **Regla base:** si en (q_i, q_j) uno es final y el otro no entonces los estados q_i, q_j **son distinguibles.**
3. **Regla deductiva:** si $\delta(q_i, a) = q_i'$ y $\delta(q_j, a) = q_j'$ y q_i', q_j' son distinguibles, entonces q_i y q_j son distinguibles.

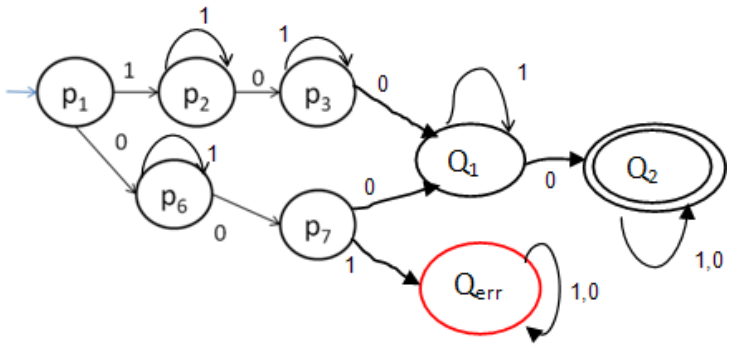


Ej.:

1. Q_2 y Q_{err} son distinguibles (**base**, el AFD debe ser completo).
2. Como $\delta(Q_1, 0) = Q_2$ y $\delta(Q_{err}, 0) = Q_{err}$ y (Q_2, Q_{err}) son distinguibles, entonces Q_1 y Q_{err} son distinguibles.
3. $\delta(p_3, 0) = Q_1$ y $\delta(Q_{err}, 0) = Q_{err}$ y (Q_1, Q_{err}) son distinguibles, entonces p_3 y Q_{err} son distinguibles.

¿Cómo se averigua si dos estados son distinguibles?

1. Partir de un AFD completo.
2. **Regla base:** si en (q_i, q_j) uno es final y el otro no entonces los estados q_i, q_j **son distinguibles.**
3. **Regla deductiva:** si $\delta(q_i, a) = q_i'$ y $\delta(q_j, a) = q_j'$ y q_i', q_j' son distinguibles, entonces q_i y q_j son distinguibles.



Ej.:

1. Q_2 y Q_{err} son distinguibles (**base**, el AFD debe ser completo).
2. Como $\delta(Q_1, 0) = Q_2$ y $\delta(Q_{err}, 0) = Q_{err}$ y (Q_2, Q_{err}) son distinguibles, entonces Q_1 y Q_{err} son distinguibles.
3. $\delta(p_3, 0) = Q_1$ y $\delta(Q_{err}, 0) = Q_{err}$ y (Q_1, Q_{err}) son distinguibles, entonces p_3 y Q_{err} son distinguibles.
4. $\delta(p_3, 1) = p_3$ y $\delta(p_7, 1) = Q_{err}$ y (p_3, Q_{err}) son distinguibles, entonces p_3 y p_7 son distinguibles.

Algoritmo de minimización de un AFD

FUNCIÓN **minimizaAFD (M)**

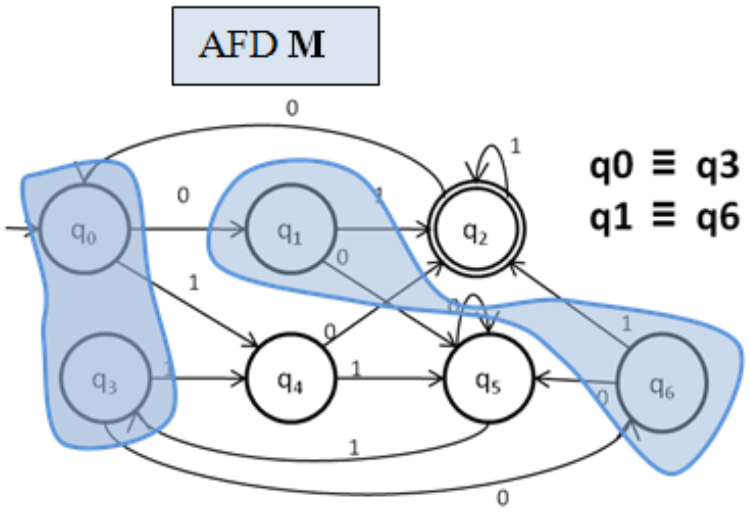
ENTRADA: un autómata finito determinista $M = (Q, V, \delta, q_0, F)$
DEVUELVE: $M_m = (Q_m, V, \delta_m, q_0, F_m)$ tal que M_m es mínimo y equivalente a M.

- Aplicar algoritmo **marcarTablaTriangular(M):**
  - Eliminar estados inaccesibles en M.
  - Completar autómata resultante.
  - Devuelve una tabla T con todas las casillas correspondientes a parejas de estados distinguibles marcadas.
- Obtener el **conjunto de estados del autómata mínimo** a partir de la tabla T, uniendo estados equivalentes entre sí (parejas no marcadas en T son estados equivalentes).
- Obtener el **estado inicial y conjunto de estados finales** del autómata mínimo.
- Obtener el **función de transición** δ_m del autómata mínimo.

Ver ejemplo de aplicación (ppt).

Ej.: Ejemplo de minimización de AFD.

Una vez que se sabe que $q_0 \equiv q_3$ y $q_1 \equiv q_6$ por el algoritmo de marcado de tabla triangular, se obtiene el conjunto cociente, el estado inicial y los estados finales del autómata mínimo:



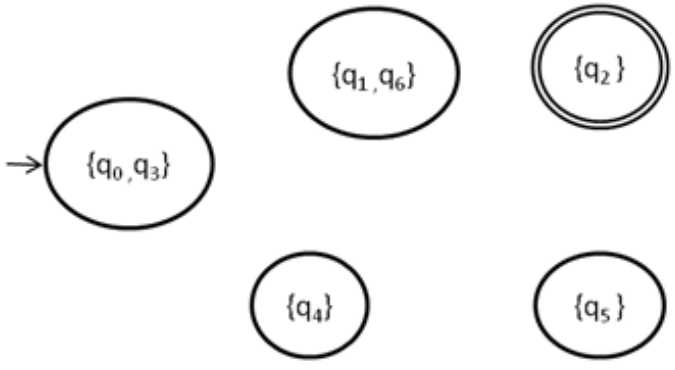
$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$$

Conjunto de estados, estado inicial y conjunto de estados finales del AFD mínimo equivalente a M

$$Q_{\min} = Q/\equiv = \{\{q_0, q_3\}, \{q_1, q_6\}, \{q_2\}, \{q_4\}, \{q_5\}\}$$

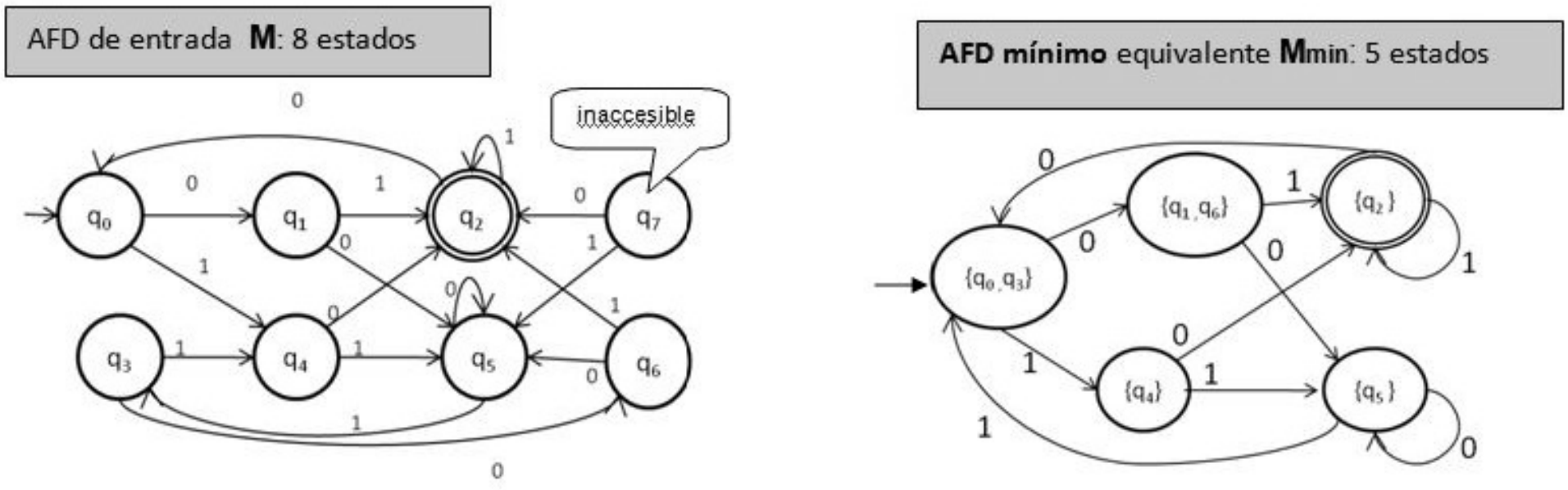
conjunto cociente de estados

$$q_{0\min} = \{q_0, q_3\} \qquad F_{\min} = \{\{q_2\}\}$$



Ej.: Ejemplo de minimización de AFD.

Finalmente se obtiene la función de transición del AFD mínimo a partir de la función de transición del AFD original:



Teorema de unicidad y equivalencia de autómatas

Teorema de unicidad del autómata mínimo

Dado un autómata finito determinista M, existe un **único AFD mínimo completo** equivalente a M y se puede obtener mediante el algoritmo de minimización *minimizaAFD*.

Teorema de unicidad y equivalencia de autómatas

Teorema de unicidad del autómata mínimo

Dado un autómata finito determinista M, existe un **único AFD mínimo completo** equivalente a M y se puede obtener mediante el algoritmo de minimización *minimizaAFD*.

Comprobación de equivalencia de AFDs

FUNCIÓN **equivalentesAFD(M_1, M_2)**

DEVUELVE: *true* si $L(M_1) = L(M_2)$ (los AFDs son equivalentes) y *false* en otro caso.

1. $M_{1min} := minimizaAFD(M_1)$;
2. $M_{2min} := minimizaAFD(M_2)$;
3. SI M_{1min} y M_{2min} son isomorfos (iguales salvo renombramiento de estados)
   ENTONCES DEVOLVER (*true*);
   SI-NO DEVOLVER (*false*);

Autómatas finitos no deterministas (AFNDs)

Un **autómata finito no determinista** (*non deterministic finite automaton NFA*) se define mediante una quíntupla $M = (Q, V, \delta, q_0, F)$ igual que un autómata determinista, excepto que la **función de transición** que se define ahora: $\delta : Q \times (V \cup \{\lambda\}) \longrightarrow \mathcal{P}(Q)$

Autómatas finitos no deterministas (AFNDs)

Un **autómata finito no determinista** (*non deterministic finite automaton NFA*) se define mediante una quíntupla $M = (Q, V, \delta, q_0, F)$ igual que un autómata determinista, excepto que la **función de transición** que se define ahora: $\delta : Q \times (V \cup \{\lambda\}) \longrightarrow \mathcal{P}(Q)$

¿Por qué "**no determinista**"?

- A partir de una configuración es posible que se pueden alcanzar **múltiples configuraciones** (por tanto, **multiples estados**) en el instante siguiente (en un paso de cálculo).
  $\rightrightarrows$ **diferentes formas de procesar una cadena**, debido a:

Autómatas finitos no deterministas (AFNDs)

Un **autómata finito no determinista** (*non deterministic finite automaton NFA*) se define mediante una quíntupla $M = (Q, V, \delta, q_0, F)$ igual que un autómata determinista, excepto que la **función de transición** que se define ahora: $\delta : Q \times (V \cup \{\lambda\}) \longrightarrow \mathcal{P}(Q)$

¿Por qué "**no determinista**"?

- A partir de una configuración es posible que se pueden alcanzar **múltiples configuraciones** (por tanto, **multiples estados**) en el instante siguiente (en un paso de cálculo).
  $\Rightarrow$ **diferentes formas de procesar una cadena**, debido a:
- **Regla de transición no determinista con múltiples opciones**, como $\delta(q, a) = \{q_1, q_2, q_3\} \Rightarrow$ las tres opciones de cambio de estado son aplicables en una configuración del tipo $(\mathbf{q}, \mathbf{az}), z \in V^*$.

Autómatas finitos no deterministas (AFNDs)

Un **autómata finito no determinista** (*non deterministic finite automaton NFA*) se define mediante una quíntupla $M = (Q, V, \delta, q_0, F)$ igual que un autómata determinista, excepto que la **función de transición** que se define ahora: $\delta : Q \times (V \cup \{\lambda\}) \longrightarrow \mathcal{P}(Q)$

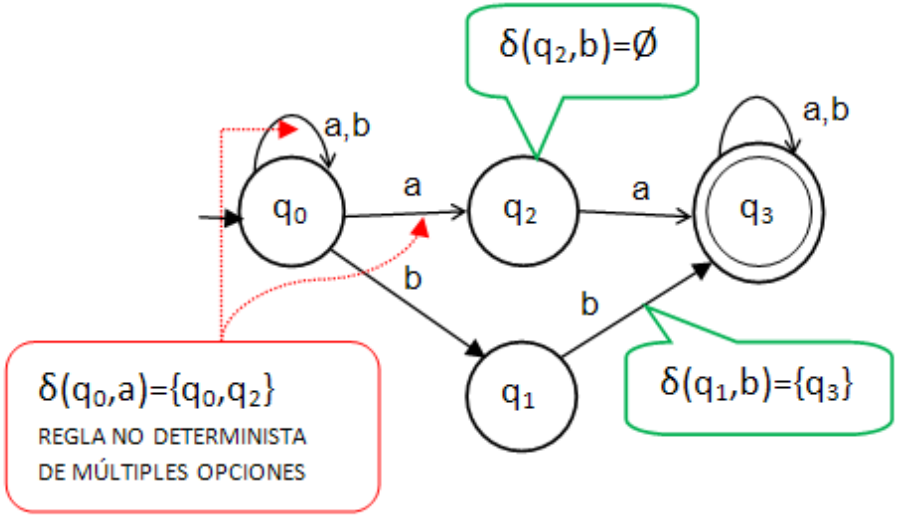
¿Por qué "**no determinista**"?

- A partir de una configuración es posible que se pueden alcanzar **múltiples configuraciones** (por tanto, **multiples estados**) en el instante siguiente (en un paso de cálculo).
  $\rightrightarrows$ **diferentes formas de procesar una cadena**, debido a:
- **Regla de transición no determinista con múltiples opciones**, como $\delta(q, a) = \{q_1, q_2, q_3\} \rightrightarrows$ las tres opciones de cambio de estado son aplicables en una configuración del tipo $(\mathbf{q}, \mathbf{az}), z \in V^*$.
- **Regla no determinista sin lectura de símbolo** o ***λ-transición***, como $\delta(p, \lambda) = \{p'\}$. Si hay otra como $\delta(p, a) = \{p''\}$, entonces las dos reglas son aplicables en una configuración como $(\mathbf{p}, \mathbf{az})$.

Autómatas finitos no deterministas (AFNDs)

Un **autómata finito no determinista** (*non deterministic finite automaton NFA*) se define mediante una quíntupla $M = (Q, V, \delta, q_0, F)$ donde: $\delta : Q \times (V \cup \{\lambda\}) \longrightarrow \mathcal{P}(Q)$

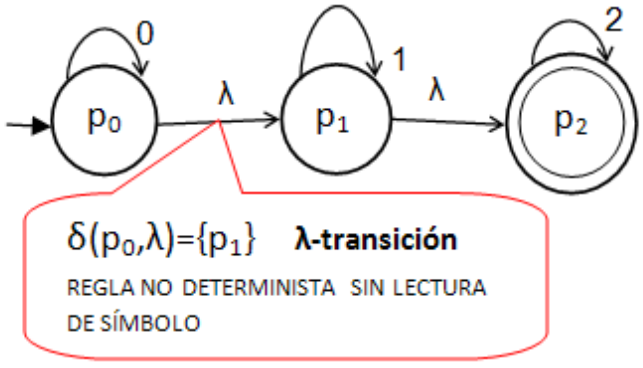
Ejemplos de AFNDs con y sin λ-transiciones



AFND $M_{aa/bb}$

ACEPTA: aa, bb, abb, baa, aaa, bbb, aaba,...

RECHAZA: λ, a, b, ba, bab, aba, abab,...



AFND M_{012}

ACEPTA: λ, 0, 1, 2, 01, 02, 12, 012, 001, ...

RECHAZA: 10, 20, 120, 2201, ...

Aceptación en AFND. Cálculo y tiempo de ejecución

Intuitivamente, $M = (Q, V, \delta, q_0, F)$ **acepta una cadena** w si existe un camino en el diagrama que parte del estado inicial y llega a un estado final **leyendo los símbolos de la cadena, seguidos o pasando por arcos etiquetados con λ antes o después de cada símbolo.**

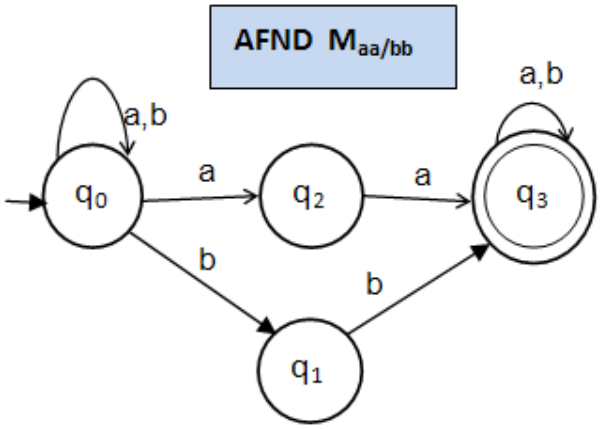
Formalmente, M acepta w si se cumple: $(q_0, w) \Rightarrow^* (q_F, \lambda)$ y $q_F \in F \rightrightarrows$ existe un cálculo que acaba en configuración de aceptación.

Aceptación en AFND. Cálculo y tiempo de ejecución

Intuitivamente, $M = (Q, V, \delta, q_0, F)$ **acepta una cadena** w si existe un camino en el diagrama que parte del estado inicial y llega a un estado final **leyendo los símbolos de la cadena, seguidos o pasando por arcos etiquetados con λ antes o después de cada símbolo.**

Formalmente, M acepta w si se cumple: $(q_0, w) \Rightarrow^* (q_F, \lambda)$ y $q_F \in F \Rightarrow$ existe un cálculo que acaba en configuración de aceptación.

Ej.:



Acepta *aab* por el **cálculo:**

$$(q_0, aab) \overset{q_2 \in \delta(q_0,a)}{\Rightarrow} (q_2, ab) \overset{q_3 \in \delta(q_2,a)}{\Rightarrow}$$

$$(q_3, b) \overset{q_3 \in \delta(q_3,b)}{\Rightarrow} (q_3, \lambda)$$

Luego: $(q_0, aab) \Rightarrow^* (q_3, \lambda)$ y q_3 es final.

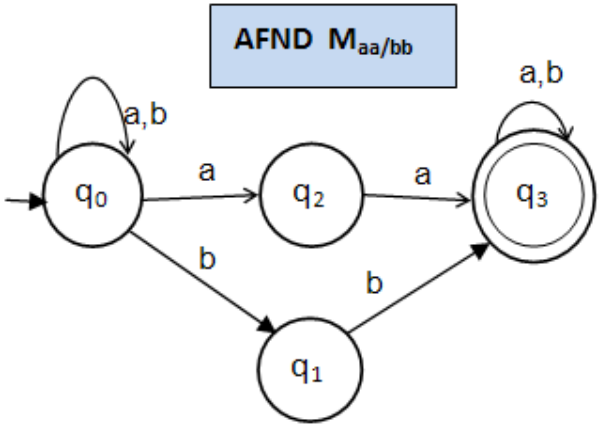
Tiempo de ejecución: 3 pasos (se aplican 3 reglas de transición).

Aceptación en AFND. Cálculo y tiempo de ejecución

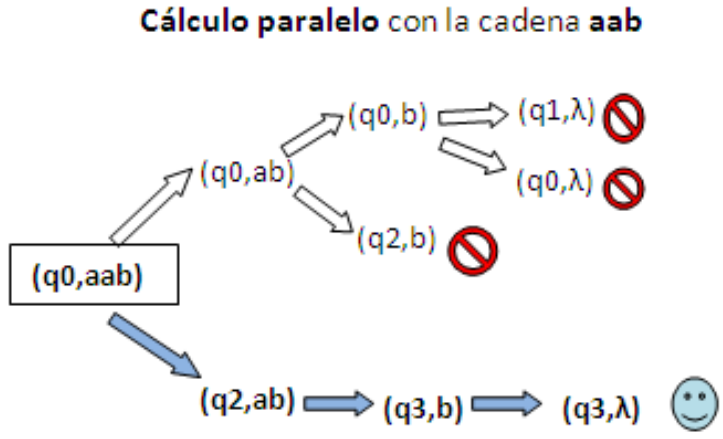
Intuitivamente, $M = (Q, V, \delta, q_0, F)$ **acepta una cadena** w si existe un camino en el diagrama que parte del estado inicial y llega a un estado final **leyendo los símbolos de la cadena, seguidos o pasando por arcos etiquetados con λ antes o después de cada símbolo.**

Formalmente, M acepta w si se cumple: $(q_0, w) \Rightarrow^* (q_F, \lambda)$ y $q_F \in F \rightrightarrows$ existe un cálculo que acaba en configuración de aceptación.

Ej.:



Al ser no determinista hay otros posibles cálculos con *aab*.

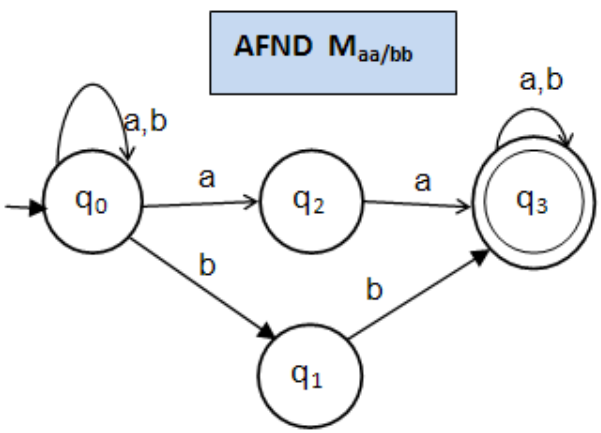


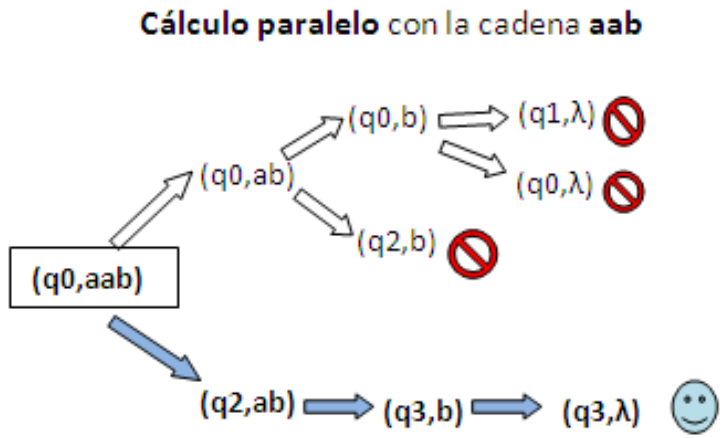
Aceptación en AFND. Cálculo y tiempo de ejecución

Intuitivamente, $M = (Q, V, \delta, q_0, F)$ **acepta una cadena** w si existe un camino en el diagrama que parte del estado inicial y llega a un estado final **leyendo los símbolos de la cadena, seguidos o pasando por arcos etiquetados con λ antes o después de cada símbolo.**

Formalmente, M acepta w si se cumple: $(q_0, w) \Rightarrow^* (q_F, \lambda)$ y $q_F \in F \rightrightarrows$ existe un cálculo que acaba en configuración de aceptación.

Ej.:





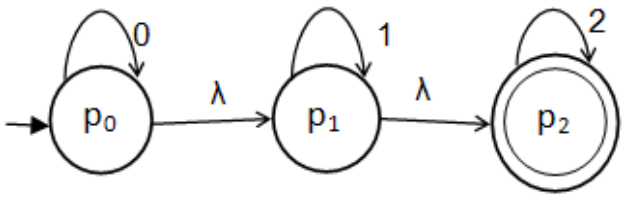
Rechaza *ab*: en 2 pasos, ningún cálculo acaba en estado final.

Aceptación en AFND. Cálculo y tiempo de ejecución

Intuitivamente, $M = (Q, V, \delta, q_0, F)$ **acepta una cadena** si existe un camino en el diagrama que parte del estado inicial y llega a un estado final **leyendo los símbolos de la cadena, seguidos o pasando por arcos etiquetados con λ antes o después de cada símbolo.**

Formalmente, M acepta w si se cumple: $(q_0, w) \Rightarrow^* (q_F, \lambda)$ y $q_F \in F \rightrightarrows$ existe un cálculo ...

Ej.: Autómata M_{012}



Acepta 01 por el **cálculo:**

$$(p_0, 01) \overset{p_0 \in \delta(p_0,0)}{\Rightarrow} (p_0, 1) \overset{p_1 \in \delta(p_0,\lambda)}{\Rightarrow} (p_1, 1) \overset{p_1 \in \delta(p_1,1)}{\Rightarrow} (p_1, \lambda) \overset{p_2 \in \delta(p_1,\lambda)}{\Rightarrow} (p_2, \lambda)$$

Luego: $(p_0, 01) \Rightarrow^* (p_2, \lambda)$ y p_2 es final.

Tiempo de ejecución: acepta en 4 pasos.

Definición formal de lenguaje aceptado por un AFND

Dado un *AFND* $M = (Q, V, \delta, q_0, F)$, el **lenguaje aceptado** por M, denotado **L(M)**, es:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

La cadena vacía es aceptada si y sólo si, según lo anterior, se cumple que $(q_0, \lambda) \Rightarrow^* (q_F, \lambda), q_F \in F$. $\Rightarrow$ A diferencia de lo que ocurre en un AFD, λ puede ser aceptada por un AFND aunque el estado inicial no sea un estado final.

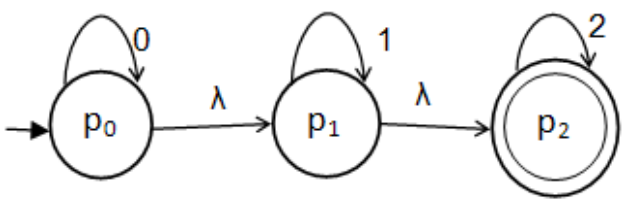
Definición formal de lenguaje aceptado por un AFND

Dado un *AFND* $M = (Q, V, \delta, q_0, F)$, el **lenguaje aceptado** por M, denotado **L(M)**, es:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

La cadena vacía es aceptada si y sólo si, según lo anterior, se cumple que $(q_0, \lambda) \Rightarrow^* (q_F, \lambda), q_F \in F$. $\Longrightarrow$ A diferencia de lo que ocurre en un AFD, λ puede ser aceptada por un AFND aunque el estado inicial no sea un estado final.

Ej.: Autómata M_{012}



M_{012} acepta λ en 2 pasos porque:

$(p_0, \lambda) \Rightarrow (p_1, \lambda) \Rightarrow (p_2, \lambda)$

Luego, $(p_0, \lambda) \Rightarrow^* (p_2, \lambda)$ y $p_2 \in F$, por tanto: $\lambda \in L(M_{012})$.

Definición formal de lenguaje aceptado por un AFND

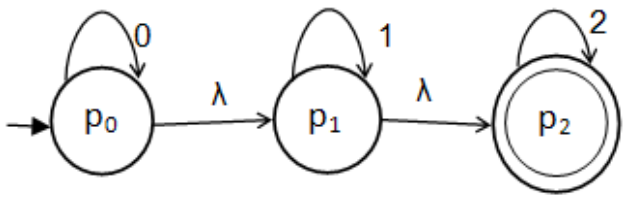
Dado un *AFND* $M = (Q, V, \delta, q_0, F)$, el **lenguaje aceptado** por M, denotado **L(M)**, es:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

Un problema de análisis de un AFND

¿Qué lenguaje acepta M_{012}?

Autómata M_{012}



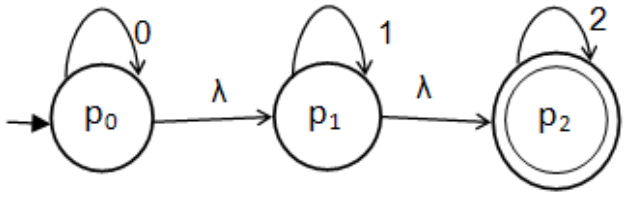
Definición formal de lenguaje aceptado por un AFND

Dado un *AFND* $M = (Q, V, \delta, q_0, F)$, el **lenguaje aceptado** por M, denotado **L(M)**, es:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

Un problema de análisis de un AFND

Autómata M_{012}



¿Qué lenguaje acepta M_{012}?

- El estado final p_2 "recuerda" que se leyó una secuencia opcional de 0's, seguida de una secuencia opcional de 1's y seguida de una secuencia opcional de 2's.

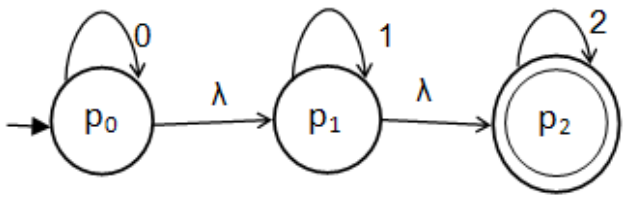
Definición formal de lenguaje aceptado por un AFND

Dado un *AFND* $M = (Q, V, \delta, q_0, F)$, el **lenguaje aceptado** por M, denotado **L(M)**, es:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

Un problema de análisis de un AFND

Autómata M_{012}



¿Qué lenguaje acepta M_{012}?

- El estado final p_2 "recuerda" que se leyó una secuencia opcional de 0's, seguida de una secuencia opcional de 1's y seguida de una secuencia opcional de 2's.
- Acepta cadenas del tipo $0^i1^j2^k$, donde $i, j, k \geq 0$. Luego:
  $L(M_{012}) = \{0^i1^j2^k \mid i, j, k \geq 0\}$

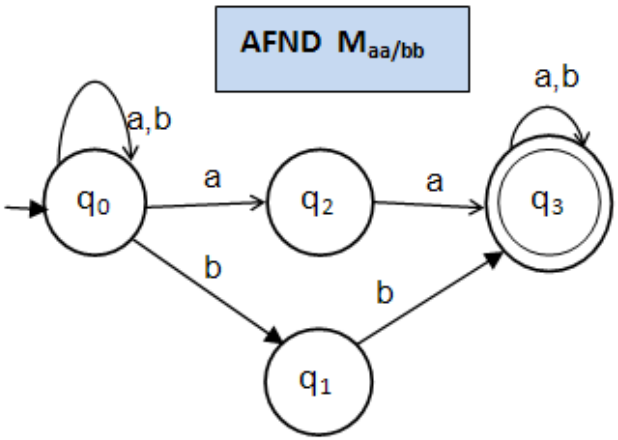
Definición formal de lenguaje aceptado por un AFND

Dado un *AFND* $M = (Q, V, \delta, q_0, F)$, el **lenguaje aceptado** por M, denotado **L(M)**, es:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

Un problema de análisis de un AFND

¿Qué lenguaje acepta $M_{aa/bb}$?

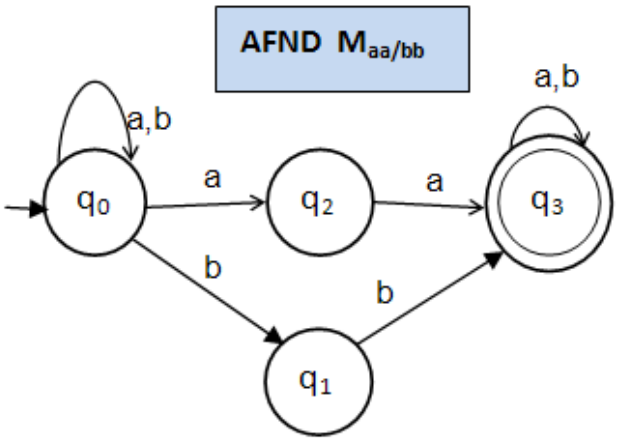


Definición formal de lenguaje aceptado por un AFND

Dado un *AFND* $M = (Q, V, \delta, q_0, F)$, el **lenguaje aceptado** por M, denotado **L(M)**, es:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

Un problema de análisis de un AFND



¿Qué lenguaje acepta $M_{aa/bb}$?

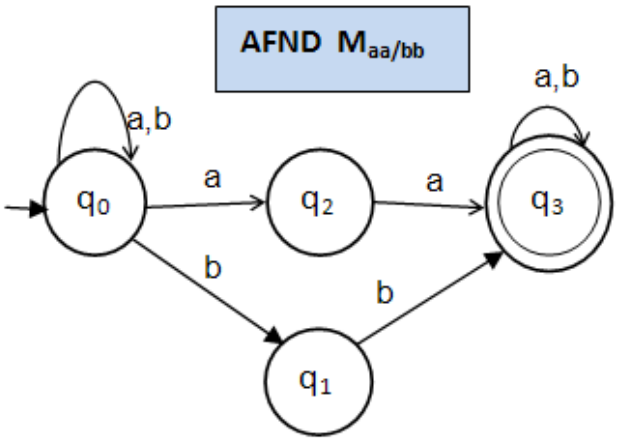
- Para llegar al estado final q_3 hay que leer la subcadena *aa* o la subcadena *bb* y antes o después una secuencia opcional de a's/b's.

Definición formal de lenguaje aceptado por un AFND

Dado un *AFND* $M = (Q, V, \delta, q_0, F)$, el **lenguaje aceptado** por M, denotado **L(M)**, es:

$$L(M) = \{w \in V^* \mid (q_0, w) \Rightarrow^* (q_F, \lambda), q_F \in F\}$$

Un problema de análisis de un AFND



¿Qué lenguaje acepta $M_{aa/bb}$?

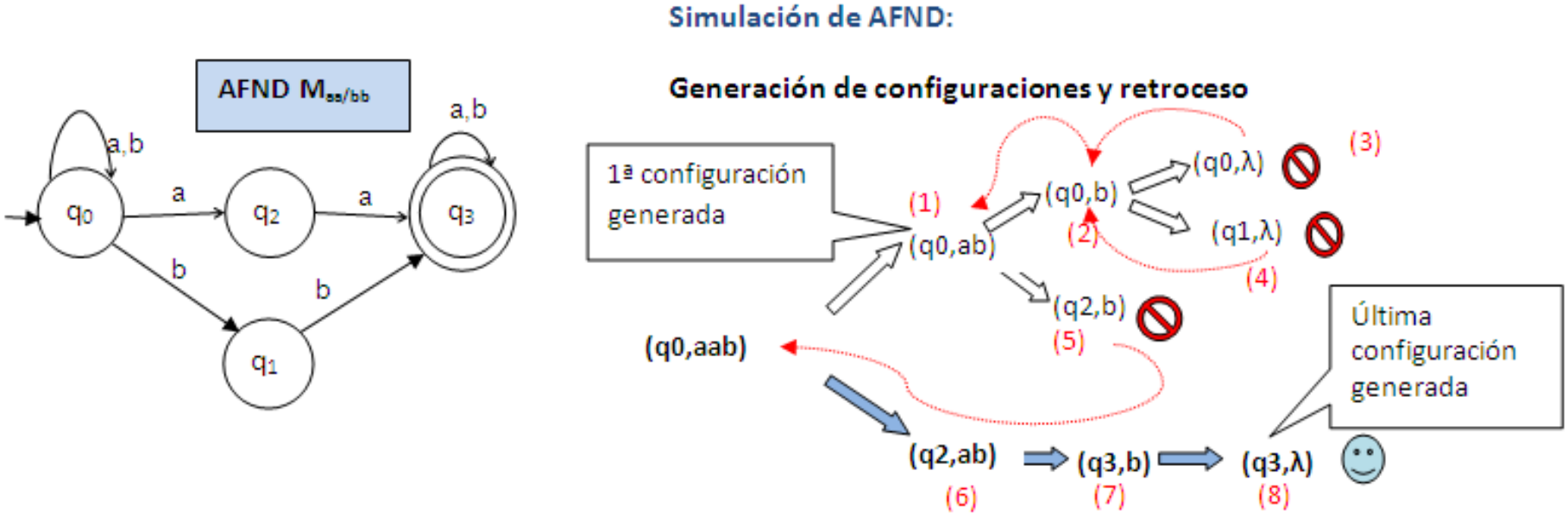
- Para llegar al estado final q_3 hay que leer la subcadena *aa* o la subcadena *bb* y antes o después una secuencia opcional de a's/b's.
- $L(M_{aa/bb}) = \{w \in \{a, b\}^* \mid w = xaay \vee w = xbby,$ **donde** $x, y \in \{a, b\}^*\}$

Simulación de un AFND

- Un AFD puede ser simulado por un algoritmo en tiempo $O(n)$, donde n es la longitud de la cadena de entrada.
- Para simular el funcionamiento de un AFND es necesario explorar las distintas ramas del árbol de cálculo paralelo con una cadena de entrada $\Rightarrow$ **algoritmo con backtracking** de **tiempo exponencial** en el peor de los casos.

Simulación de un AFND

- Un AFD puede ser simulado por un algoritmo en tiempo $O(n)$, donde n es la longitud de la cadena de entrada.
- Para simular el funcionamiento de un AFND es necesario explorar las distintas ramas del árbol de cálculo paralelo con una cadena de entrada $\rightrightarrows$ **algoritmo con backtracking** de **tiempo exponencial** en el peor de los casos.



Solución alternativa a la simulación con backtracking $\rightrightarrows$ convertir el AFND a un AFD equivalente.

El no determinismo facilita el diseño

Ej.: Si queremos diseñar un *AF* que acepte el lenguaje $\{ab, aba\}^*$ es más sencillo obtener un *AFND* que obtener un *AFD* directamente.

- **Test de prueba:**

  Cadenas que deben ser aceptadas: (pertenecen a $\{ab, aba\}^*$):

El no determinismo facilita el diseño

Ej.: Si queremos diseñar un AF que acepte el lenguaje $\{ab, aba\}^*$ es más sencillo obtener un $AFND$ que obtener un AFD directamente.

- **Test de prueba:**

  Cadenas que deben ser aceptadas: (pertenecen a $\{ab, aba\}^*$):
  $\lambda, ab, aba, abab, abaaba, ababa, abaab, \ldots$

El no determinismo facilita el diseño

Ej.: Si queremos diseñar un *AF* que acepte el lenguaje $\{ab, aba\}^*$ es más sencillo obtener un *AFND* que obtener un *AFD* directamente.

- **Test de prueba:**

  Cadenas que deben ser aceptadas: (pertenecen a $\{ab, aba\}^*$): λ, ab, aba, $abab$, $abaaba$, $ababa$, $abaab$, ...

  Cadenas que deben ser rechazadas: a, b, ba, aa, bb, aab, abb, ...

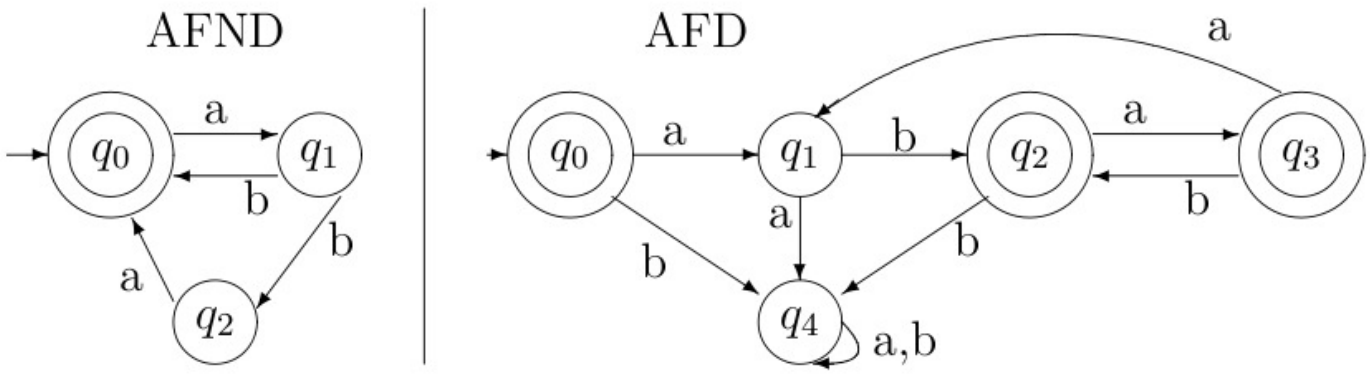
El no determinismo facilita el diseño

Ej.: Si queremos diseñar un *AF* que acepte el lenguaje $\{ab, aba\}^*$ es más sencillo obtener un *AFND* que obtener un *AFD* directamente.

- **Test de prueba:**

  Cadenas que deben ser aceptadas: (pertenecen a $\{ab, aba\}^*$): λ, *ab*, *aba*, *abab*, *abaaba*, *ababa*, *abaab*, ...

  Cadenas que deben ser rechazadas: *a*, *b*, *ba*, *aa*, *bb*, *aab*, *abb*, ...



El no determinismo facilita el diseño

Ej.: Se requiere diseñar un autómata M que sirva para comprobar si una cadena de 0's/1's **contiene la subcadena** 0110 o la subcadena 1000. Equivale a un **problema de búsqueda** de esas subcadenas en secuencias de 0's/1's.

Estrategia para el diseño:

El no determinismo facilita el diseño

Ej.: Se requiere diseñar un autómata M que sirva para comprobar si una cadena de 0's/1's **contiene la subcadena** 0110 o la subcadena 1000. Equivale a un **problema de búsqueda** de esas subcadenas en secuencias de 0's/1's.

Estrategia para el diseño:

1. Obtenemos M_1 que acepte cadenas con subcadena 0110
   $\Rightarrow$ queremos que $L(M_1) = \{x0110z \mid x, z \in \{0,1\}^*\}$
2. Obtenemos M_2 que acepte cadenas con subcadena 1000
   $\Rightarrow$ queremos que $L(M_2) = \{x1000z \mid x, z \in \{0,1\}^*\}$

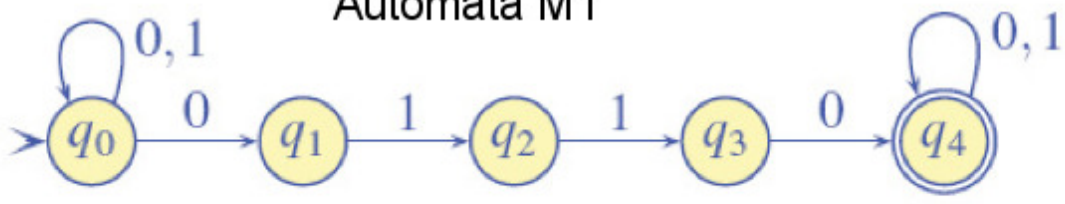
El no determinismo facilita el diseño

Ej.: Se requiere diseñar un autómata M que sirva para comprobar si una cadena de 0's/1's **contiene la subcadena** 0110 o la subcadena 1000. Equivale a un **problema de búsqueda** de esas subcadenas en secuencias de 0's/1's.

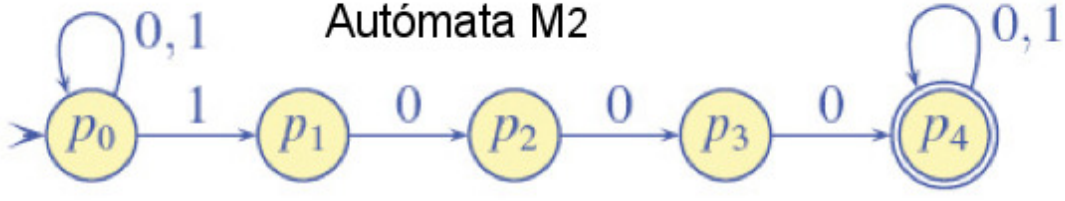
Estrategia para el diseño:

1. Obtenemos M_1 que acepte cadenas con subcadena 0110
   $\Rightarrow$ queremos que $L(M_1) = \{x0110z \mid x, z \in \{0,1\}^*\}$
2. Obtenemos M_2 que acepte cadenas con subcadena 1000
   $\Rightarrow$ queremos que $L(M_2) = \{x1000z \mid x, z \in \{0,1\}^*\}$

Autómata M1



Autómata M2

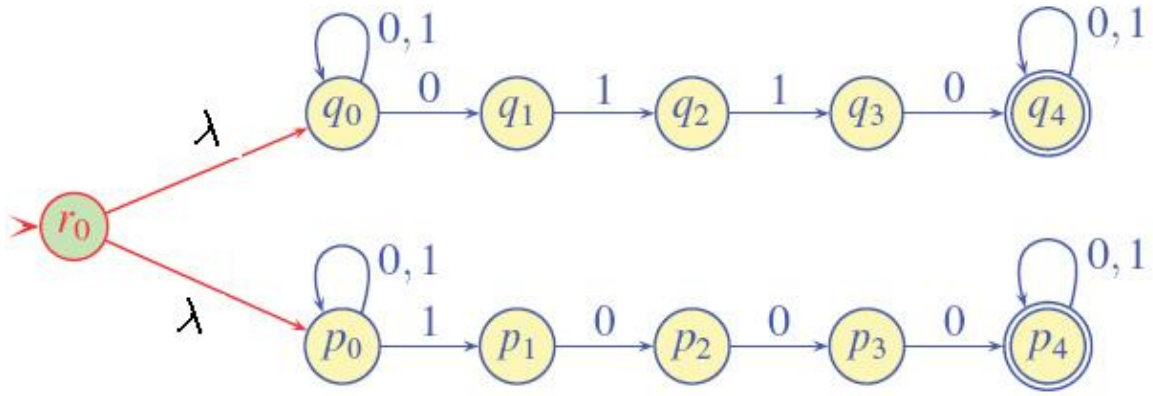


El no determinismo facilita el diseño

Ej.: Se requiere diseñar un autómata M que sirva para comprobar si una cadena de 0's/1's **contiene la subcadena** 0110 o la subcadena 1000. Equivale a un **problema de búsqueda** de esas subcadenas en secuencias de 0's/1's.

Estrategia para el diseño:

1. Obtenemos M_1 que acepte cadenas con subcadena 0110
   $\Rightarrow$ queremos que $L(M_1) = \{x0110z \mid x, z \in \{0,1\}^*\}$
2. Obtenemos M_2 que acepte cadenas con subcadena 1000
   $\Rightarrow$ queremos que $L(M_2) = \{x1000z \mid x, z \in \{0,1\}^*\}$
3. Combinamos M_1 y M_2 mediante λ-transiciones para obtener M tal que $L(M) = L(M_1) \cup L(M_2)$



El no determinismo facilita el diseño

Problema resuelto 6: se requiere diseñar un autómata que acepte cadenas de $a's/b's$ cuyo tercer símbolo desde la derecha es b.

Estrategia para el diseño:

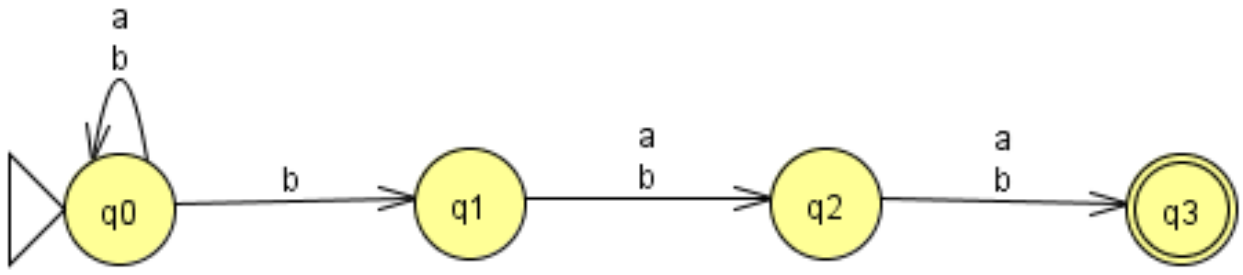
1. Es lo mismo que decir que las cadenas acaban en $baa/bab/bba/bbb$ y antes una secuencia opcional de a's/b's.

El no determinismo facilita el diseño

Problema resuelto 6: se requiere diseñar un autómata que acepte cadenas de $a's/b's$ cuyo tercer símbolo desde la derecha es b.

Estrategia para el diseño:

1. Es lo mismo que decir que las cadenas acaban en $baa/bab/bba/bbb$ y antes una secuencia opcional de a's/b's.
2. El AFND es:



Acepta cadenas que tienen -b- en tercera posición de derecha a izquierda

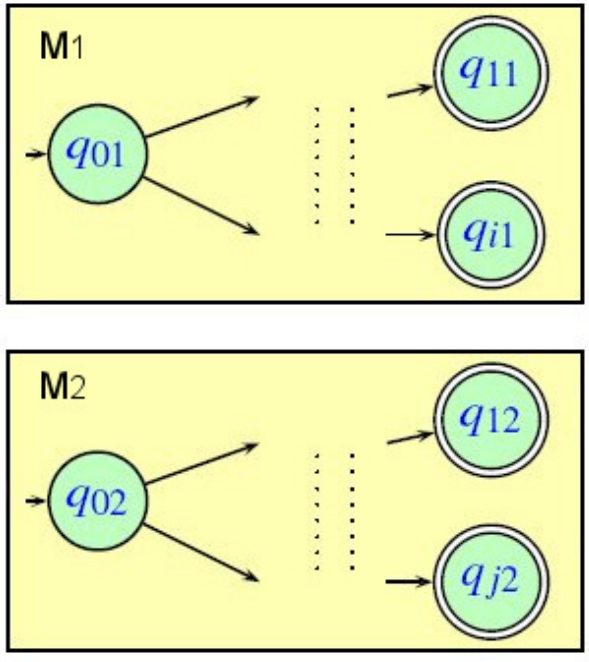
(acaban en baa, bab, bbb, etc....)

Operaciones regulares con AF (I)

Método algorítmico unionAF (M_1, M_2)

DESCRIPCIÓN: combina dos autómatas finitos M_1 y M_2 para obtener otro autómata M_{union} tal que $L(M_{union}) = L(M_1) \cup L(M_2)$

ESQUEMA DE CONSTRUCCIÓN:



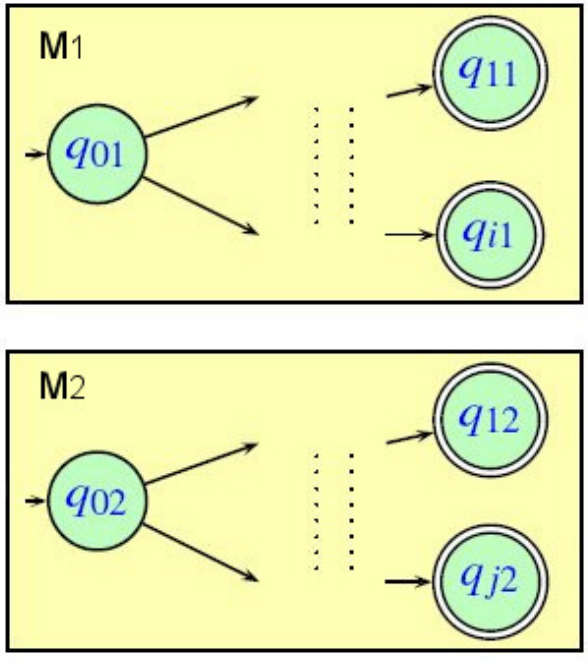
Construcción justificada de $L(M_{union}) = L(M_1) \cup L(M_2)$: ???

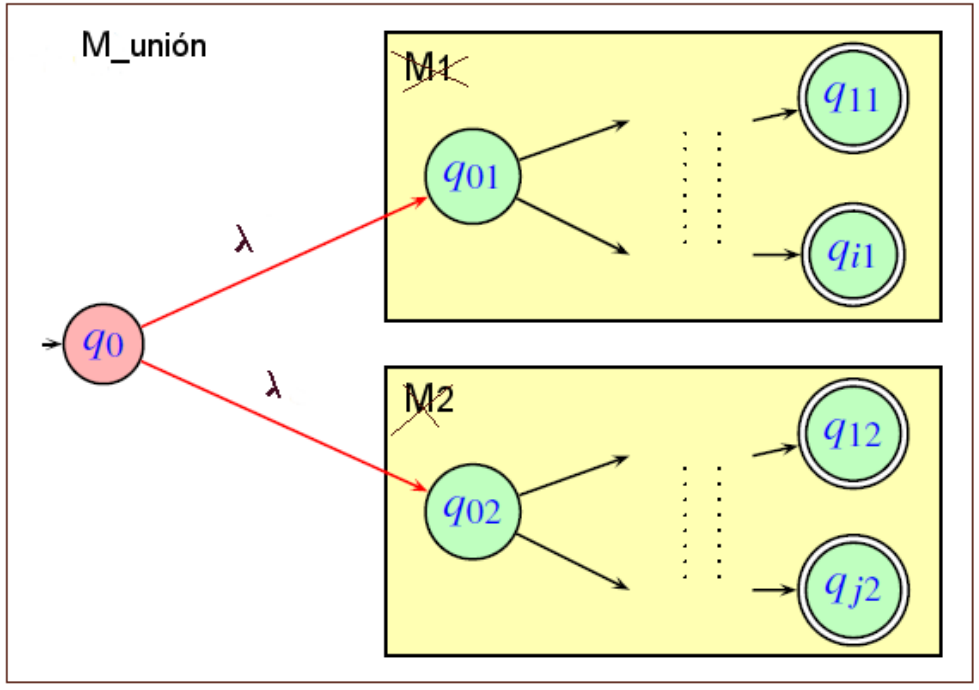
Operaciones regulares con AF (I)

Método algorítmico unionAF(M_1, M_2)

DESCRIPCIÓN: combina dos autómatas finitos M_1 y M_2 para obtener otro autómata M_{union} tal que $L(M_{union}) = L(M_1) \cup L(M_2)$

ESQUEMA DE CONSTRUCCIÓN:

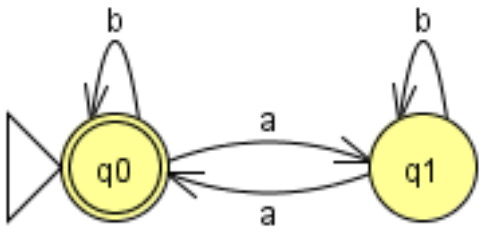


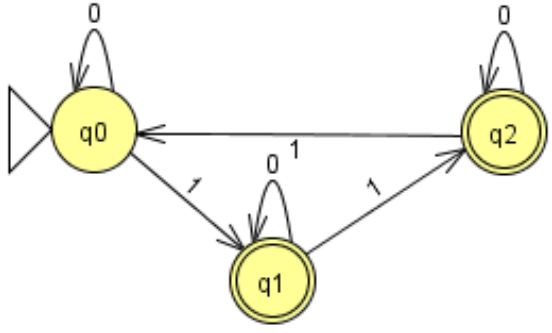


Construcción justificada de $L(M_{union}) = L(M_1) \cup L(M_2)$: ???

Ejemplo de construcción de autómata unión

M1: acepta cadenas con número par de a's



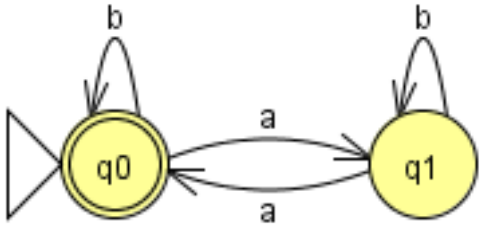


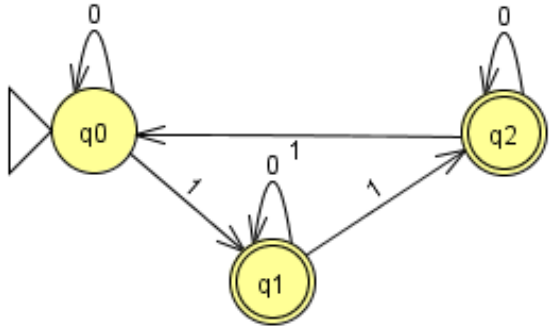
M2: acepta cadenas de número de unos no múltiplo de 3

El autómata unión es: ???

Ejemplo de construcción de autómata unión

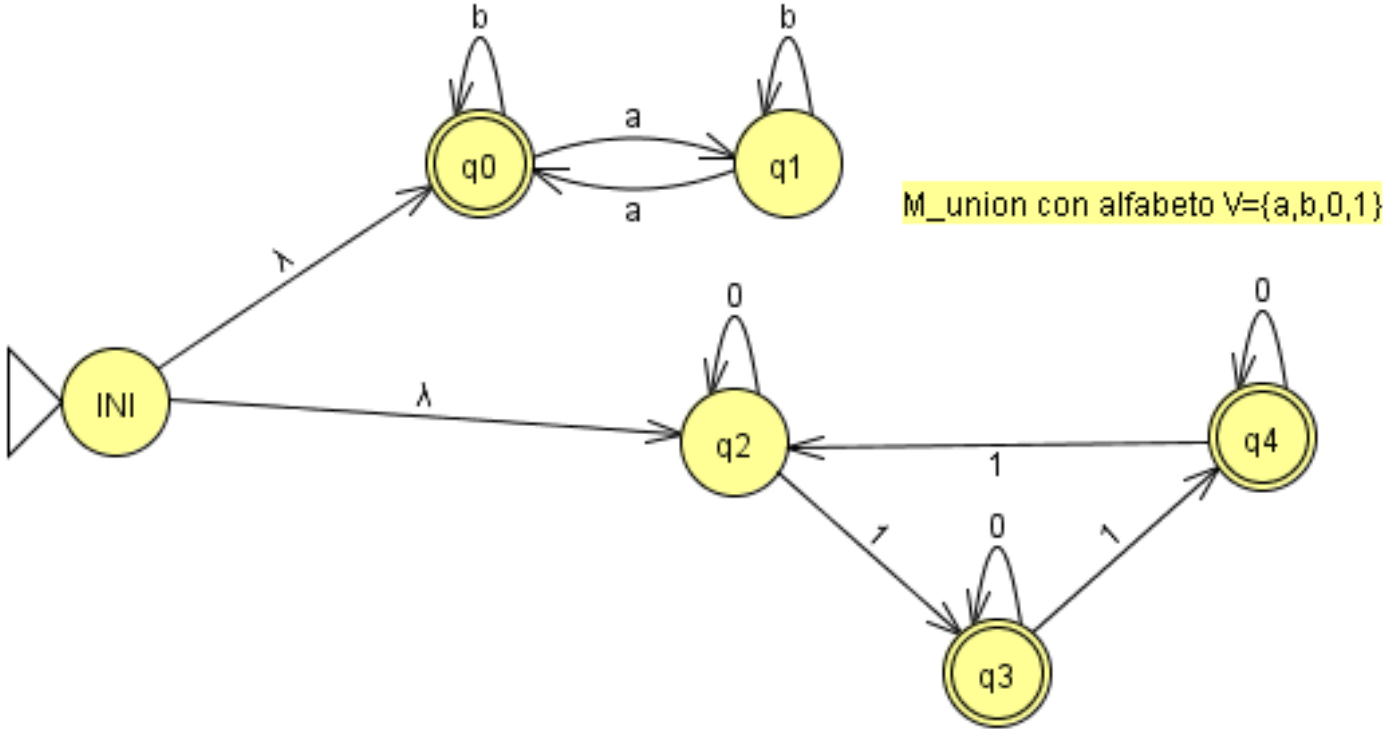
M1: acepta cadenas con número par de a's





M2: acepta cadenas de número de unos no múltiplo de 3

El autómata unión es: ???



M_union con alfabeto V={a,b,0,1}

Operaciones regulares con AF (II)

Método algorítmico complementaAFD (M)

DESCRIPCIÓN: obtiene el autómata complementario. El **autómata complementario de M**, denotado $\overline{M}$, es el que acepta las cadenas que no acepta M, es decir: $L(\overline{M}) = \overline{L(M)} = V^* - L(M)$

FUNCIÓN **complementaAFD(M)** ENTRADA: un AFD $M = (Q, V, \delta, q_0, F)$ DEVUELVE: el AFD complementario de M.

1. $F := Q - F$ // se complementa el conjunto de estados finales
2. DEVUELVE (M) //El autómata resultante es el complementario

Operaciones regulares con AF (II)

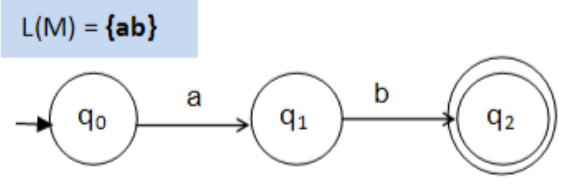
Método algorítmico complementaAFD(M)

DESCRIPCIÓN: obtiene el autómata complementario. El **autómata complementario de M**, denotado $\overline{M}$, es el que acepta las cadenas que no acepta M, es decir: $L(\overline{M}) = \overline{L(M)} = V^* - L(M)$

FUNCIÓN **complementaAFD(M)** ENTRADA: un AFD $M = (Q, V, \delta, q_0, F)$ DEVUELVE: el AFD complementario de M.

1. $M :=$ *completaAFD*(M); //se completa el AFD de entrada
2. $F := Q - F$ // se complementa el conjunto de estados finales
3. DEVUELVE (M) //El autómata resultante es el complementario

Ej.: Obtener el complementario de:



Operaciones regulares con AF (II)

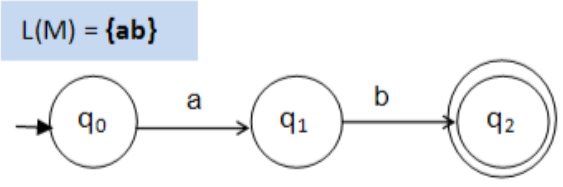
Método algorítmico complementaAFD(M)

DESCRIPCIÓN: obtiene el autómata complementario. El **autómata complementario de M**, denotado $\overline{M}$, es el que acepta las cadenas que no acepta M, es decir: $L(\overline{M}) = \overline{L(M)} = V^* - L(M)$

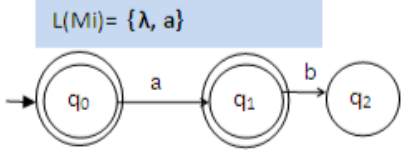
FUNCIÓN **complementaAFD(M)** ENTRADA: un AFD $M = (Q, V, \delta, q_0, F)$ DEVUELVE: el AFD complementario de M.

1. $M :=$ *completaAFD*(M); //se completa el AFD de entrada
2. $F := Q - F$ // se complementa el conjunto de estados finales
3. DEVUELVE (M) //El autómata resultante es el complementario

Ej.: Obtener el complementario de:



Este no es el complementario:



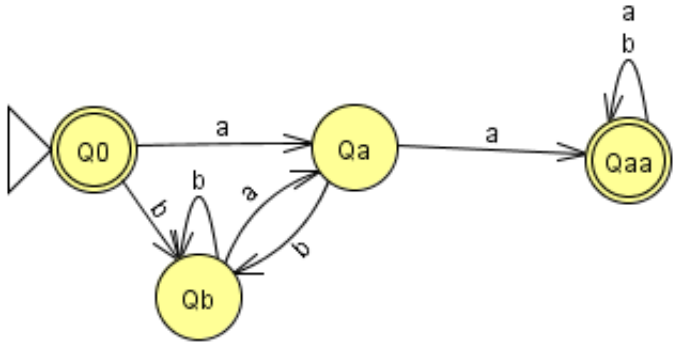
La complementación puede ayudar en el diseño

Ej.: Obtener un AFD que acepte las cadenas de $\{a, b\}^*$ que no son vacías y no contienen la subcadena $aa \Rightarrow L_{naav}$.

La complementación puede ayudar en el diseño

Ej.: Obtener un AFD que acepte las cadenas de $\{a, b\}^*$ que no son vacías y no contienen la subcadena $aa \Rightarrow L_{naav}$.

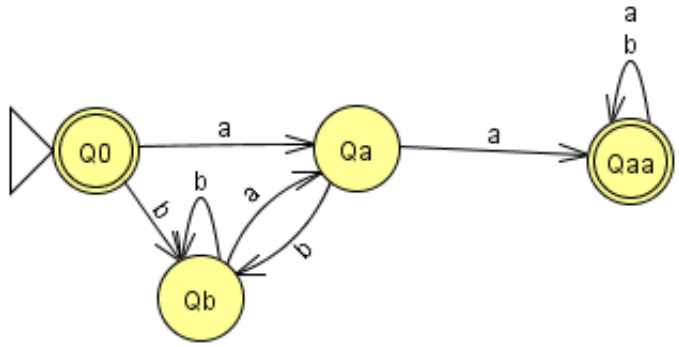
- Podemos pensar en cadenas que son vacías o contienen la subcadena $aa \Rightarrow L_{aav}$ y obtener un autómata M_{aav}:



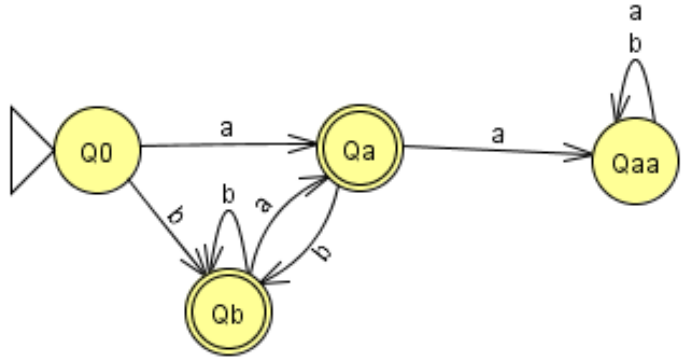
La complementación puede ayudar en el diseño

Ej.: Obtener un AFD que acepte las cadenas de $\{a, b\}^*$ que no son vacías y no contienen la subcadena $aa \Rightarrow L_{naav}$.

- Podemos pensar en cadenas que son vacías o contienen la subcadena $aa \Rightarrow L_{aav}$ y obtener un autómata M_{aav}:



- Complementamos M_{aav} y obtenemos $\overline{M_{aav}}$, que acepta L_{naav} porque $L(\overline{M_{aav}}) = \overline{L(M_{aav})} = \overline{L_{aav}} = L_{naav}$:

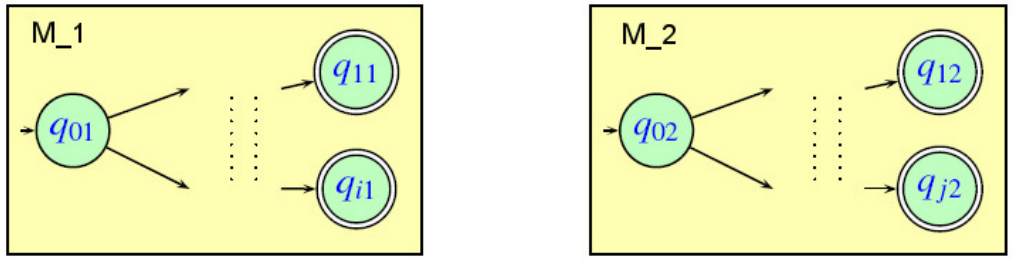


Operaciones regulares con AF (III)

Método algorítmico **concatAF** $(\mathbf{M_1}, \mathbf{M_2})$

DESCRIPCIÓN: combina dos autómatas finitos M_1 y M_2 para obtener otro autómata M_{concat} tal que $L(M_{concat}) = L(M_1) \circ L(M_2)$

ESQUEMA DE CONSTRUCCIÓN:



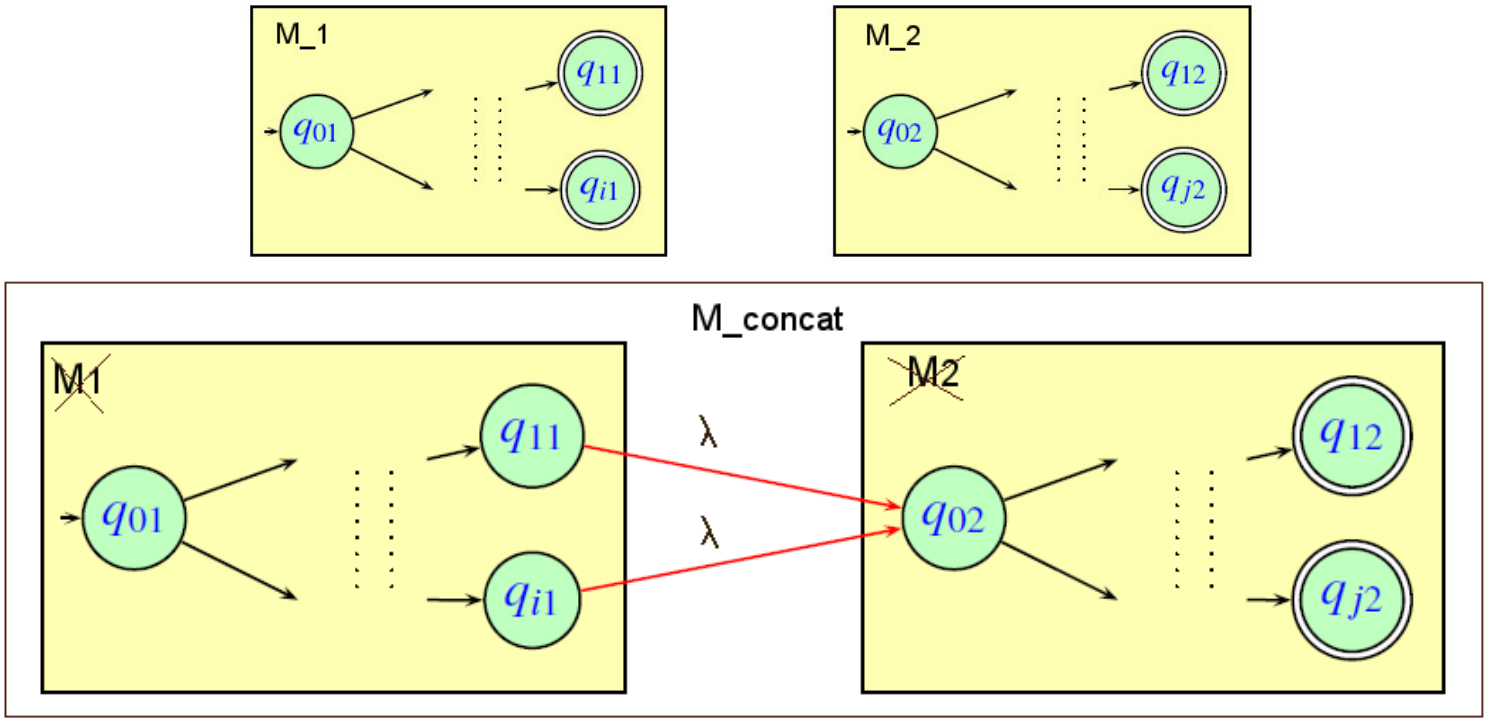
Construcción justificada de $L(M_{concat}) = L(M_1) \circ L(M_2)$: ???

Operaciones regulares con AF (III)

Método algorítmico concatAF (M_1, M_2)

DESCRIPCIÓN: combina dos autómatas finitos M_1 y M_2 para obtener otro autómata M_{concat} tal que $L(M_{concat}) = L(M_1) \circ L(M_2)$

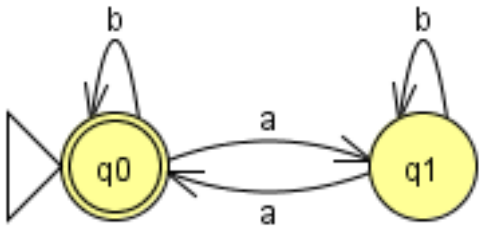
ESQUEMA DE CONSTRUCCIÓN:

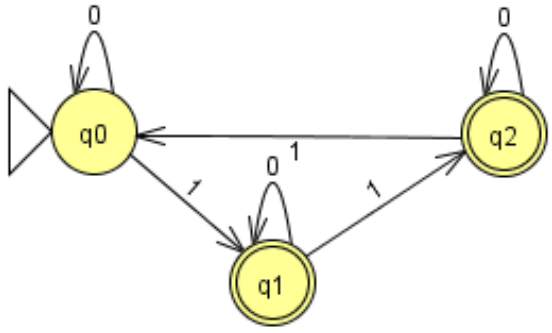


Construcción justificada de $L(M_{concat}) = L(M_1) \circ L(M_2)$: ???

Ejemplo de construcción de autómata concatenado

M1: acepta cadenas con número par de a's



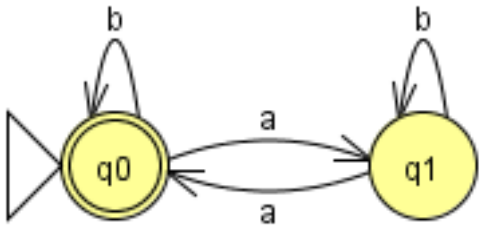


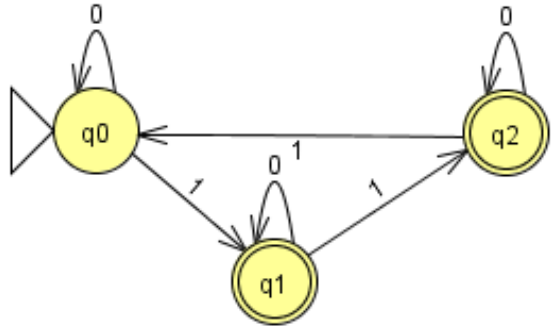
M2: acepta cadenas de número de unos no múltiplo de 3

El autómata concatenado es: ???

Ejemplo de construcción de autómata concatenado

M1: acepta cadenas con número par de a's

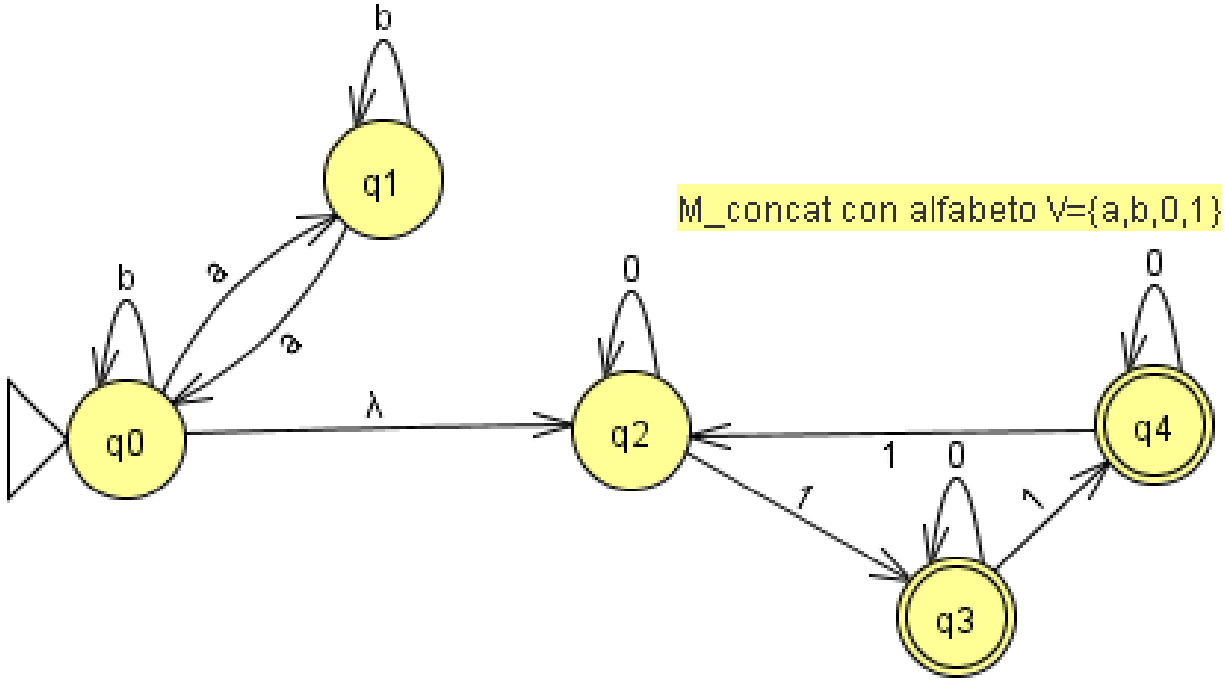




M2: acepta cadenas de número de unos no múltiplo de 3

El autómata concatenado es: ???

M_concat con alfabeto V={a,b,0,1}

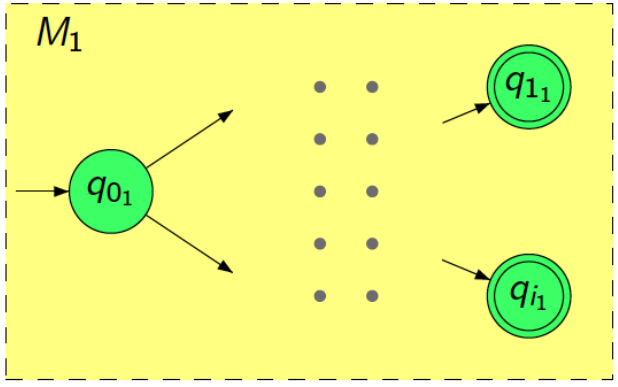


Operaciones regulares con AF (IV)

Método algorítmico clausuraAF(M_1)

DESCRIPCIÓN: a partir de M_1 obtiene otro autómata M_{star} tal que $L(M_{star}) = L(M_1)^*$

ESQUEMA DE CONSTRUCCIÓN:



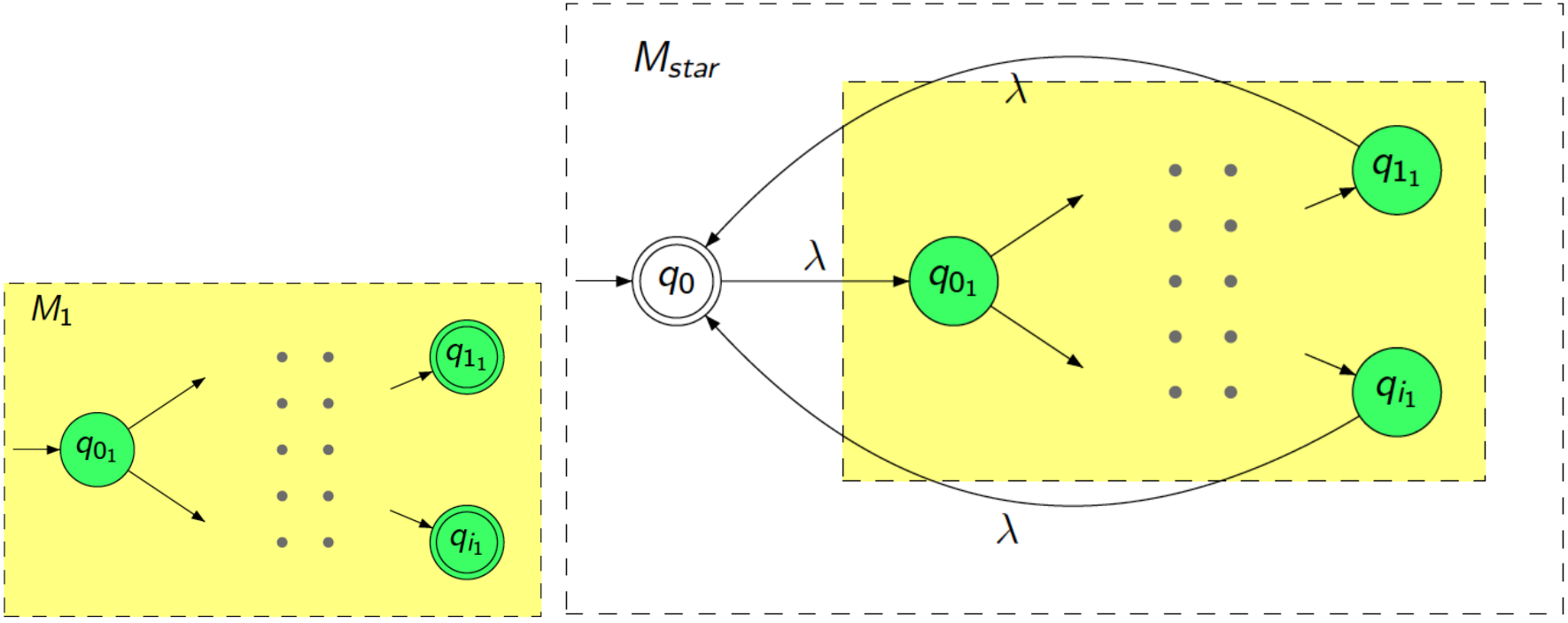
Construcción justificada de $L(M_{star}) = L(M_1)^*$???

Operaciones regulares con AF (IV)

Método algorítmico clausuraAF (M_1)

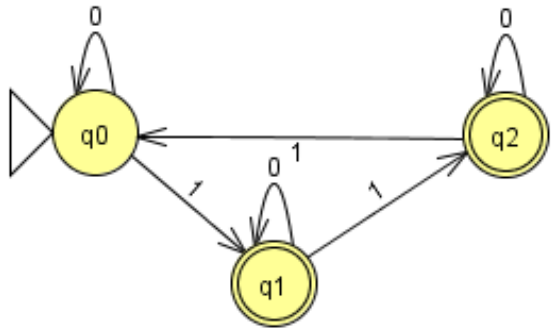
DESCRIPCIÓN: a partir de M_1 obtiene otro autómata M_{star} tal que $L(M_{star}) = L(M_1)^*$

ESQUEMA DE CONSTRUCCIÓN:



Construcción justificada de $L(M_{star}) = L(M_1)^*$???

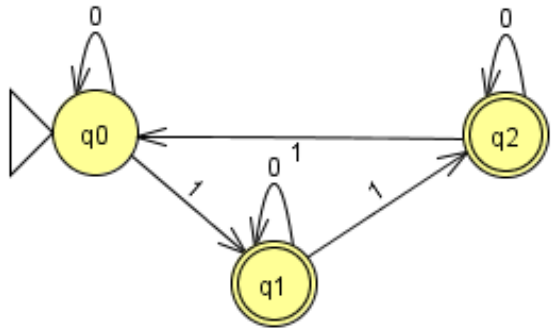
Ejemplo de construcción de autómata clausura



M2: acepta cadenas de número de unos no múltiplo de 3

El autómata clausura de M_2 acepta $L(M_2)^*$: cadenas formadas por concatenación repetida de cadenas de 0's/1's que tienen un número de unos que no es múltiplo de 3: ???

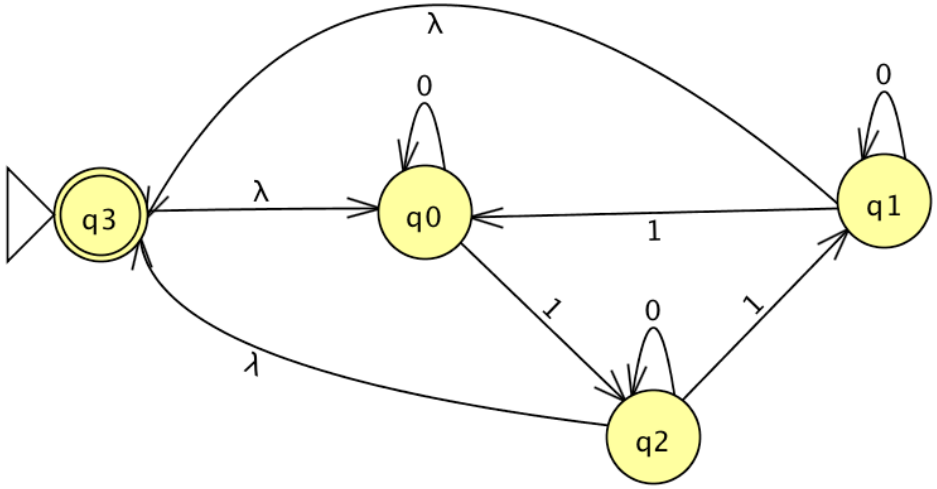
Ejemplo de construcción de autómata clausura



M2: acepta cadenas de número de unos no múltiplo de 3

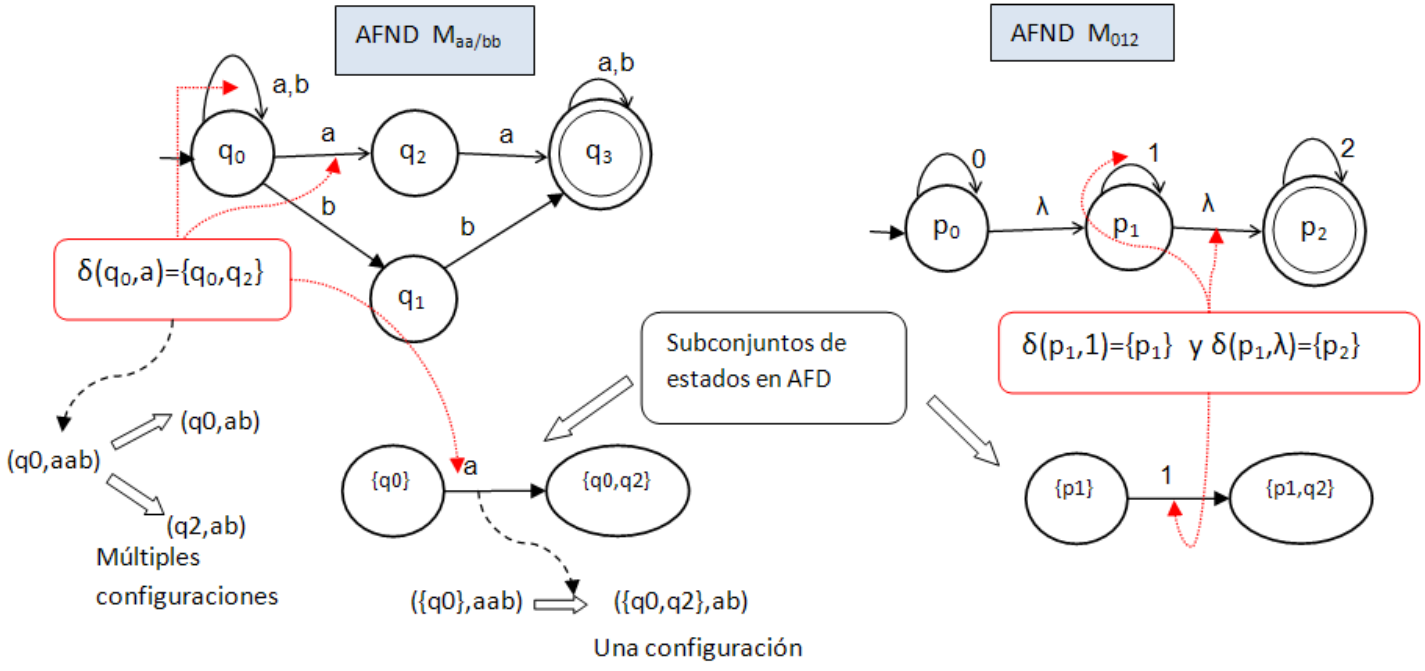
El autómata clausura de M_2 acepta $L(M_2)^*$: cadenas formadas por concatenación repetida de cadenas de 0's/1's que tienen un número de unos que no es múltiplo de 3: ???

M_star: acepta cadenas de L(M_2)*



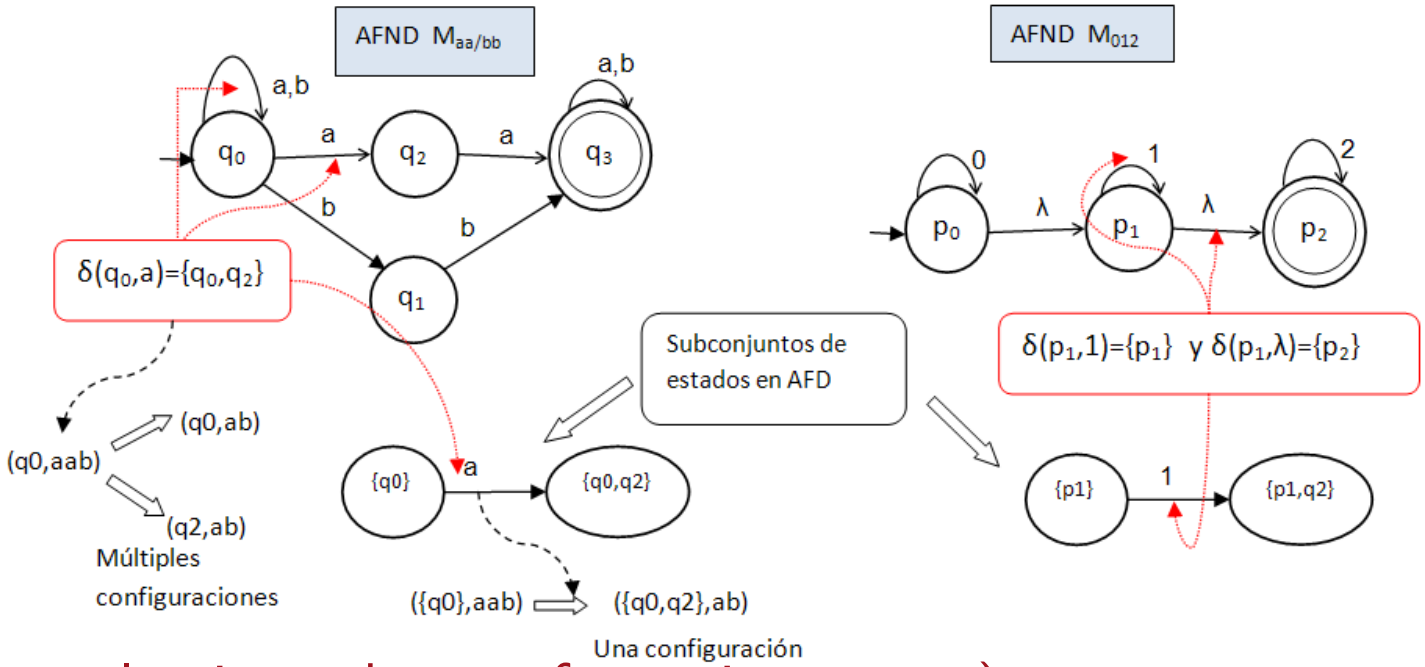
Transformación de AFND a AFD

- **Objetivo:** obtener un autómata determinista M_D que simule el modo de funcionamiento de un autómata no determinista M.
- **Idea:** hacer que **los estados de M_D sean subconjuntos de estados de M** y que la aplicación de las reglas de transición de δ_D en M_D simulen la aplicación de reglas no deterministas de δ en M.



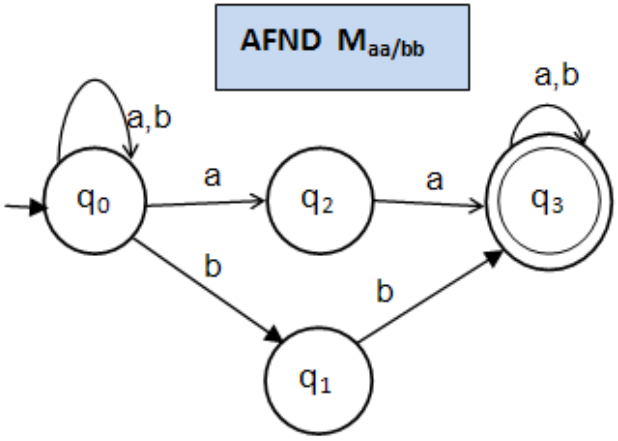
Transformación de AFND a AFD

- **Objetivo:** obtener un autómata determinista M_D que simule el modo de funcionamiento de un autómata no determinista M.
- **Idea:** hacer que **los estados de M_D sean subconjuntos de estados de M** y que la aplicación de las reglas de transición de δ_D en M_D simulen la aplicación de reglas no deterministas de δ en M.



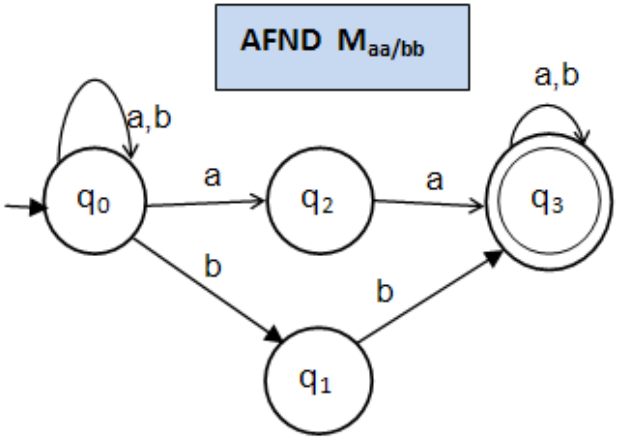
(ver algoritmo de transformación en ppt)

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

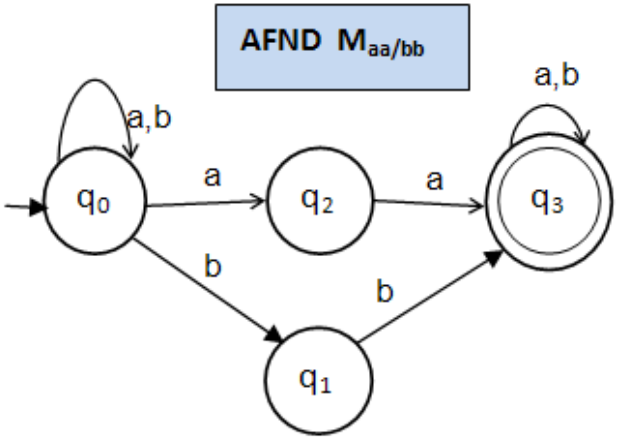
Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones

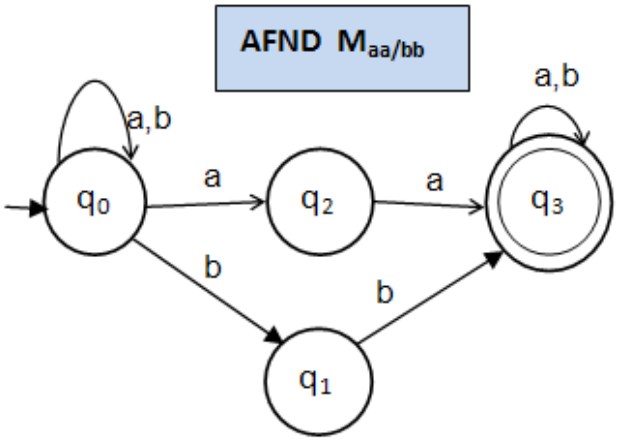


- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

Obtenemos la **función de transición** δ_D para cada estado nuevo y para los símbolos a/b.

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



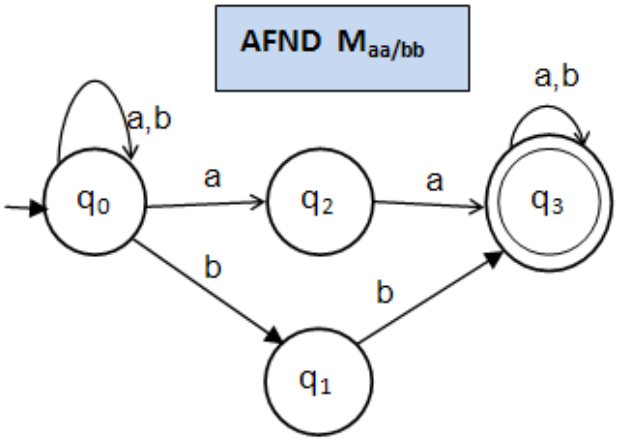
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

Obtenemos la **función de transición** δ_D para cada estado nuevo y para los símbolos a/b.

$$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) =$$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



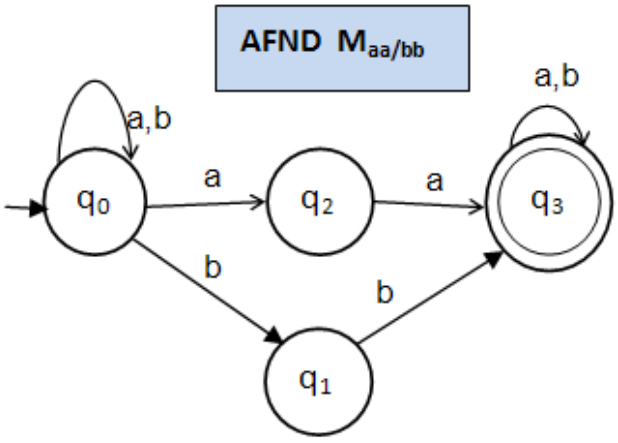
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

Obtenemos la **función de transición** δ_D para cada estado nuevo y para los símbolos a/b.

$$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



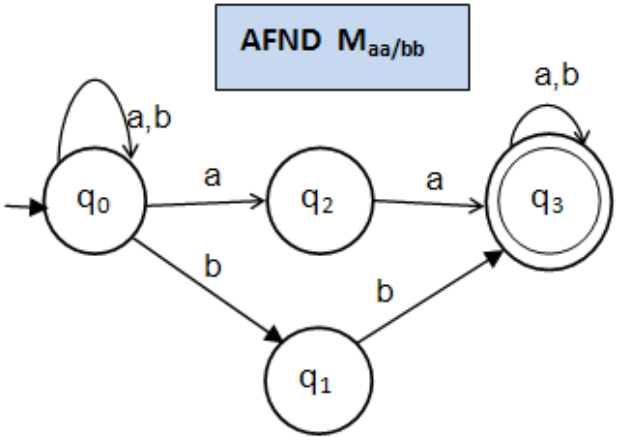
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

Obtenemos la **función de transición** δ_D para cada estado nuevo y para los símbolos a/b.

$$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\} \quad \delta_D(\{q_0\}, b) =$$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones

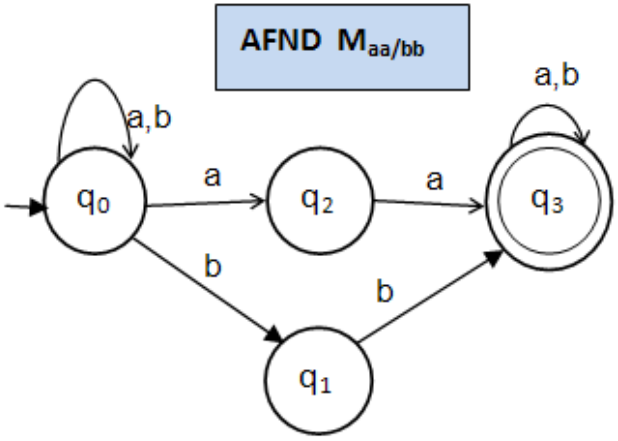


- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$ $\quad \delta_D(\{q_0\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_2\}, a) =$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones

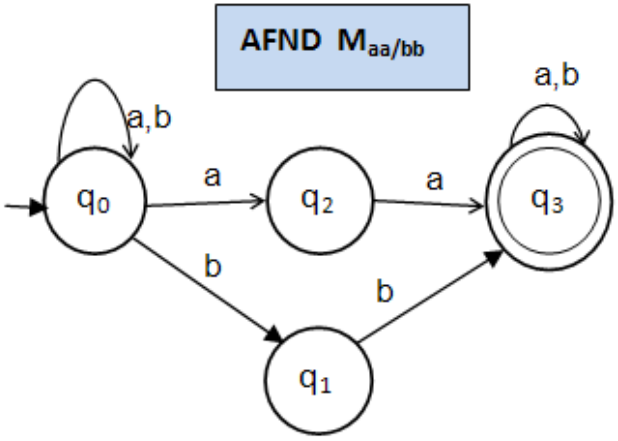


- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$ $\quad \delta_D(\{q_0\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



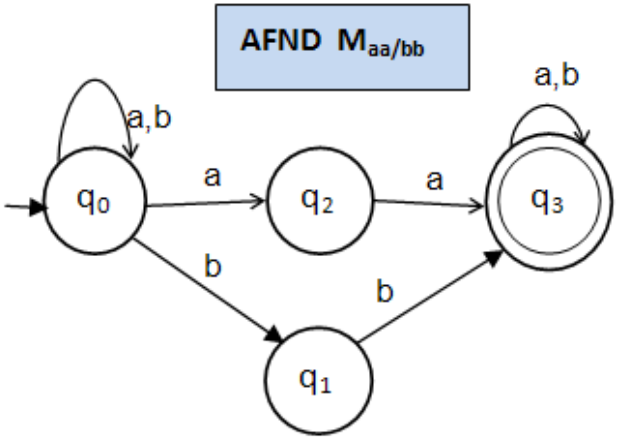
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$ $\quad \delta_D(\{q_0\}, b) = \{q_0, q_1\}$

$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$ $\quad \delta_D(\{q_0, q_2\}, b) =$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



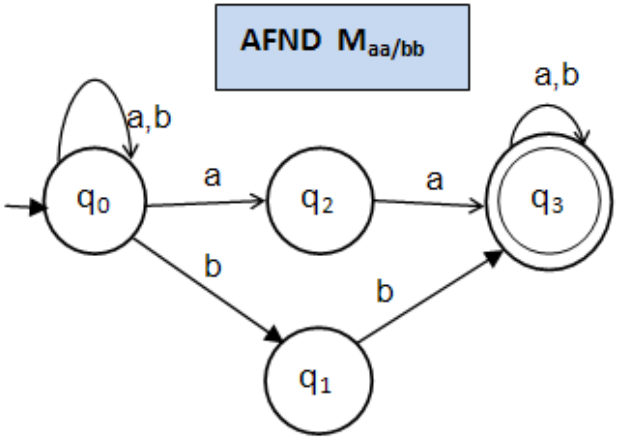
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$ $\quad \delta_D(\{q_0\}, b) = \{q_0, q_1\}$

$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$ $\quad \delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

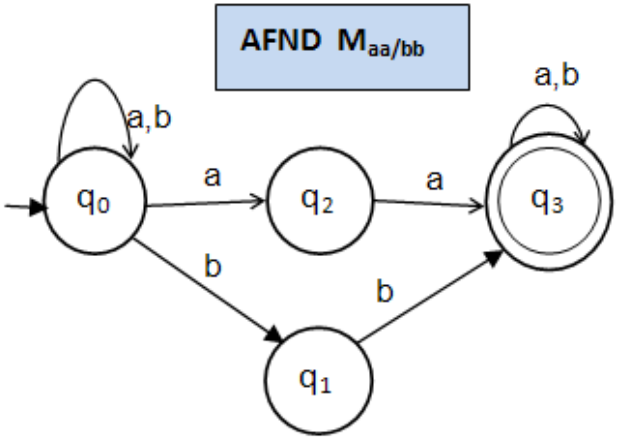
El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$ $\quad \delta_D(\{q_0\}, b) = \{q_0, q_1\}$

$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$ $\quad \delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$

$\delta_D(\{q_0, q_1\}, a) =$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

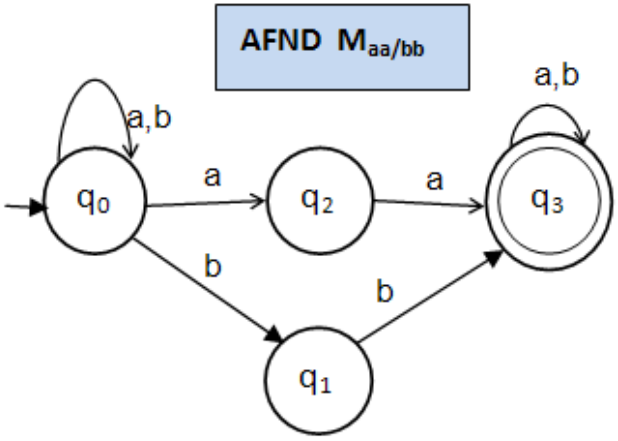
El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$ $\quad \delta_D(\{q_0\}, b) = \{q_0, q_1\}$

$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$ $\quad \delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$

$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\}$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



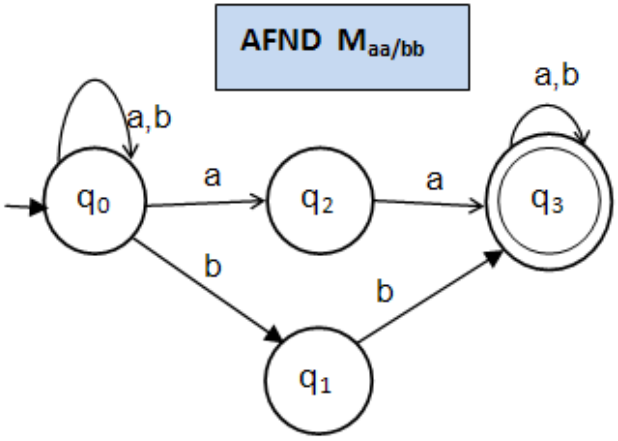
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\}$

$\delta_D(\{q_0\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_1\}, b) =$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



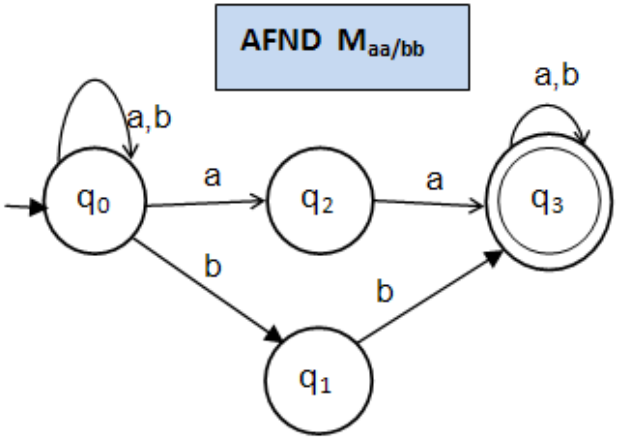
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\}$

$\delta_D(\{q_0\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_1\}, b) = \{q_0, q_1, q_3\}$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



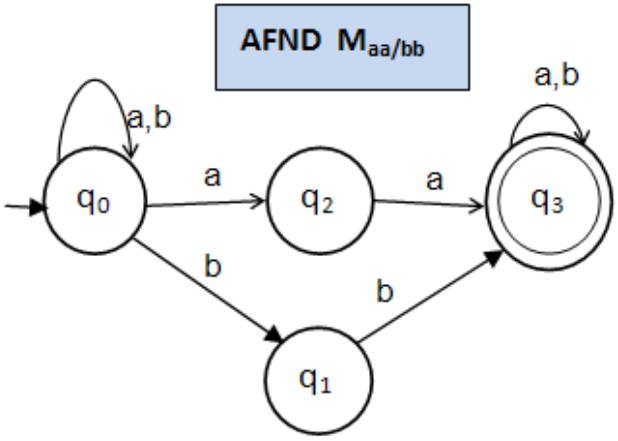
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2, q_3\}, a) =$

$\delta_D(\{q_0\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_1\}, b) = \{q_0, q_1, q_3\}$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



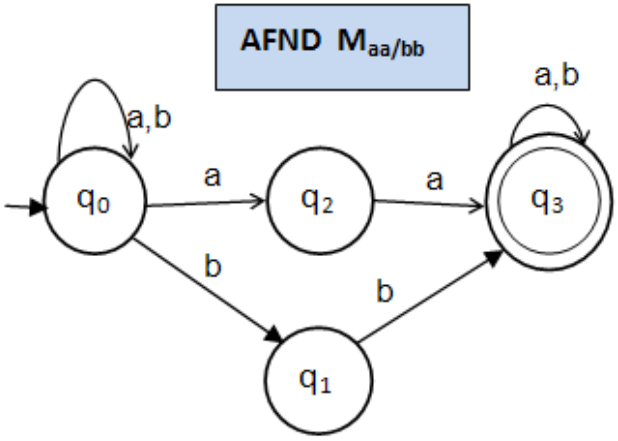
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2, q_3\}, a) = \{q_0, q_2, q_3\}$

$\delta_D(\{q_0\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_1\}, b) = \{q_0, q_1, q_3\}$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



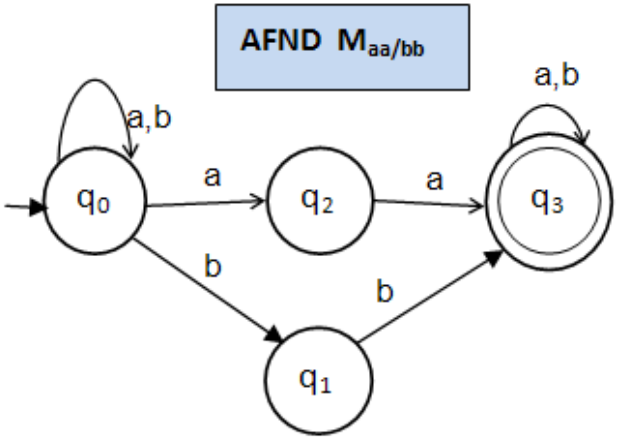
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2, q_3\}, a) = \{q_0, q_2, q_3\}$

$\delta_D(\{q_0\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_1\}, b) = \{q_0, q_1, q_3\}$
$\delta_D(\{q_0, q_2, q_3\}, b) =$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones

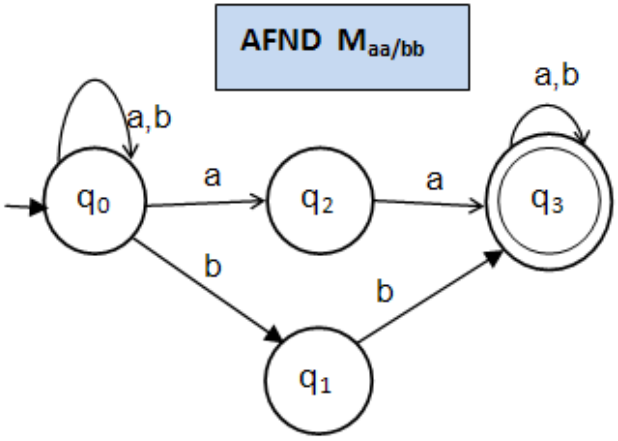


- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\} \qquad \delta_D(\{q_0\}, b) = \{q_0, q_1\}$$
$$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\} \qquad \delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$$
$$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\} \qquad \delta_D(\{q_0, q_1\}, b) = \{q_0, q_1, q_3\}$$
$$\delta_D(\{q_0, q_2, q_3\}, a) = \{q_0, q_2, q_3\} \qquad \delta_D(\{q_0, q_2, q_3\}, b) = \{q_0, q_1, q_3\}$$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



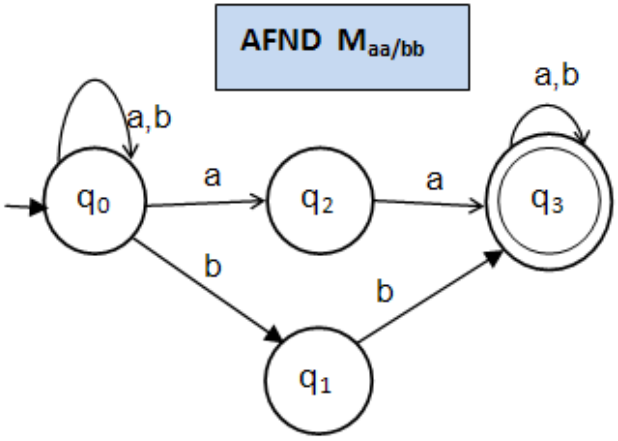
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2, q_3\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1, q_3\}, a) =$

$\delta_D(\{q_0\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_1\}, b) = \{q_0, q_1, q_3\}$
$\delta_D(\{q_0, q_2, q_3\}, b) = \{q_0, q_1, q_3\}$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



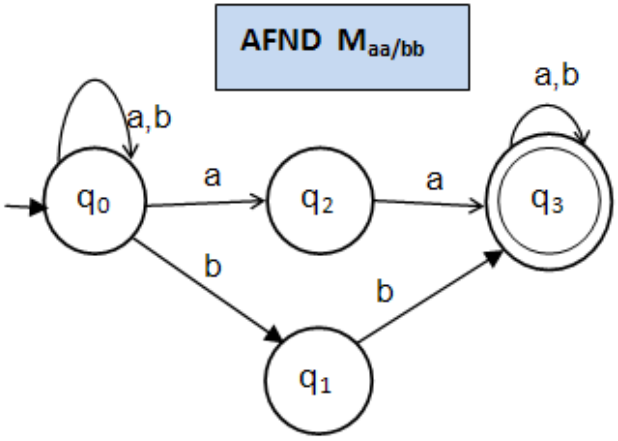
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2, q_3\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1, q_3\}, a) = \{q_0, q_2, q_3\}$

$\delta_D(\{q_0\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_1\}, b) = \{q_0, q_1, q_3\}$
$\delta_D(\{q_0, q_2, q_3\}, b) = \{q_0, q_1, q_3\}$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



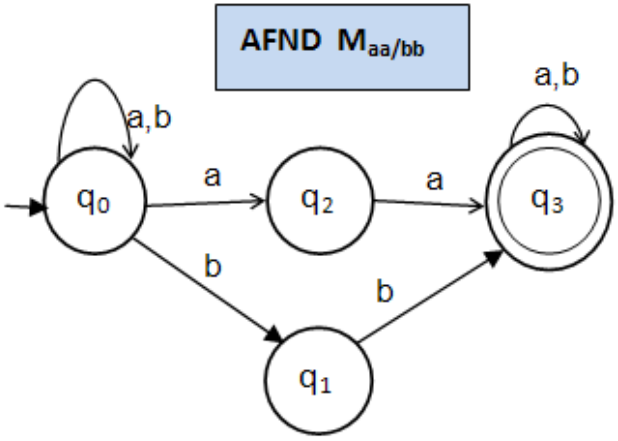
- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\}$
$\delta_D(\{q_0, q_2, q_3\}, a) = \{q_0, q_2, q_3\}$
$\delta_D(\{q_0, q_1, q_3\}, a) = \{q_0, q_2, q_3\}$

$\delta_D(\{q_0\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$
$\delta_D(\{q_0, q_1\}, b) = \{q_0, q_1, q_3\}$
$\delta_D(\{q_0, q_2, q_3\}, b) = \{q_0, q_1, q_3\}$
$\delta_D(\{q_0, q_1, q_3\}, b) =$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones

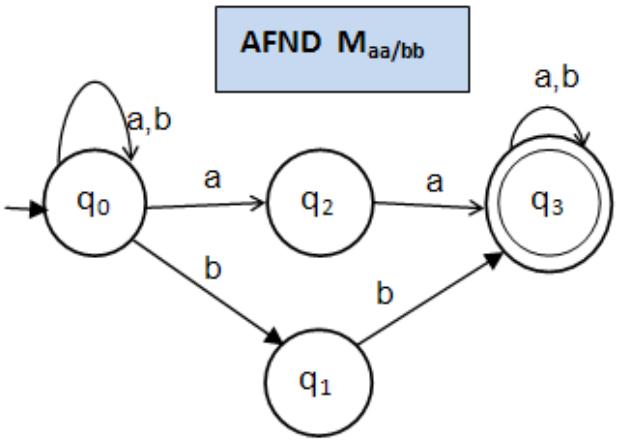


- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\} \qquad \delta_D(\{q_0\}, b) = \{q_0, q_1\}$$
$$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\} \qquad \delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$$
$$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\} \qquad \delta_D(\{q_0, q_1\}, b) = \{q_0, q_1, q_3\}$$
$$\delta_D(\{q_0, q_2, q_3\}, a) = \{q_0, q_2, q_3\} \qquad \delta_D(\{q_0, q_2, q_3\}, b) = \{q_0, q_1, q_3\}$$
$$\delta_D(\{q_0, q_1, q_3\}, a) = \{q_0, q_2, q_3\} \qquad \delta_D(\{q_0, q_1, q_3\}, b) = \{q_0, q_1, q_3\}$$

Aplicación de AFNDtoAFD a un AFND sin λ-transiciones



- Obtener el AFD equivalente al siguiente AFND:
- Al no tener λ-transiciones se puede simplificar el proceso porque $LC(E) = E$ para todo subconjunto de estados E.

El **estado inicial** es q_0 porque $LC(q_0) = \{q_0\}$.

$$\delta_D(\{q_0\}, a) = LC(\{q_0, q_2\}) = \{q_0, q_2\} \qquad \delta_D(\{q_0\}, b) = \{q_0, q_1\}$$
$$\delta_D(\{q_0, q_2\}, a) = \{q_0, q_2, q_3\} \qquad \delta_D(\{q_0, q_2\}, b) = \{q_0, q_1\}$$
$$\delta_D(\{q_0, q_1\}, a) = \{q_0, q_2\} \qquad \delta_D(\{q_0, q_1\}, b) = \{q_0, q_1, q_3\}$$
$$\delta_D(\{q_0, q_2, q_3\}, a) = \{q_0, q_2, q_3\} \qquad \delta_D(\{q_0, q_2, q_3\}, b) = \{q_0, q_1, q_3\}$$
$$\delta_D(\{q_0, q_1, q_3\}, a) = \{q_0, q_2, q_3\} \qquad \delta_D(\{q_0, q_1, q_3\}, b) = \{q_0, q_1, q_3\}$$

Estados finales: $\{q_0, q_1, q_3\}$ y $\{q_0, q_2, q_3\}$.

Los AFDs y los los AFNDs son formalismos igual de potentes

Teorema Un lenguaje es aceptado por un autómata no determinista si y sólo si es aceptado por un autómata determinista.

Los AFDs y los los AFNDs son formalismos igual de potentes

Teorema Un lenguaje es aceptado por un autómata no determinista si y sólo si es aceptado por un autómata determinista.

Dem.: La demostración tiene dos partes:

1. ($\Rightarrow$) Hay que probar que si un lenguaje L_r es aceptado por un AFD M_D entonces existe un AFND M_N tal que $L_r = L(M) = L(M_N)$. Es cierto porque un AFD se puede considerar como un "caso particular" de AFND.

Los AFDs y los los AFNDs son formalismos igual de potentes

Teorema Un lenguaje es aceptado por un autómata no determinista si y sólo si es aceptado por un autómata determinista.

Dem.: La demostración tiene dos partes:

1. ($\Rightarrow$) Hay que probar que si un lenguaje L_r es aceptado por un AFD M_D entonces existe un AFND M_N tal que $L_r = L(M) = L(M_N)$. Es cierto porque un AFD se puede considerar como un "caso particular" de AFND.
2. ($\Leftarrow$) Hay que probar que si un lenguaje L_r es aceptado por un AFND M_N entonces existe un AFD M_D tal que $L_r = L(M_N) = L(M_D)$. Es cierto porque para obtener M_D basta aplicar el algoritmo de transformación de AFND a AFD con M_N como entrada.

Por 1 y 2 se deduce que los autómatas deterministas y no deterministas aceptan los mismos tipos de lenguajes, por tanto son igual de potentes.

Relación entre autómatas finitos y expresiones regulares

Teorema de Kleene: **Un lenguaje puede ser descrito por una ER si y sólo si puede ser aceptado por un AF.**

El teorema completo se suele dividir en dos partes que afirman:

1. **Síntesis**: **si un lenguaje puede ser descrito por una expresión regular entonces también puede ser aceptado un autómata finito.**
2. Análisis: si un lenguaje puede ser aceptado por autómata finito entonces también puede ser descrito por una expresión regular.

Relación entre autómatas finitos y expresiones regulares

Teorema de Kleene: **Un lenguaje puede ser descrito por una ER si y sólo si puede ser aceptado por un AF.**

El teorema completo se suele dividir en dos partes que afirman:

1. **Síntesis**: **si un lenguaje puede ser descrito por una expresión regular entonces también puede ser aceptado un autómata finito.**
2. Análisis: si un lenguaje puede ser aceptado por autómata finito entonces también puede ser descrito por una expresión regular.

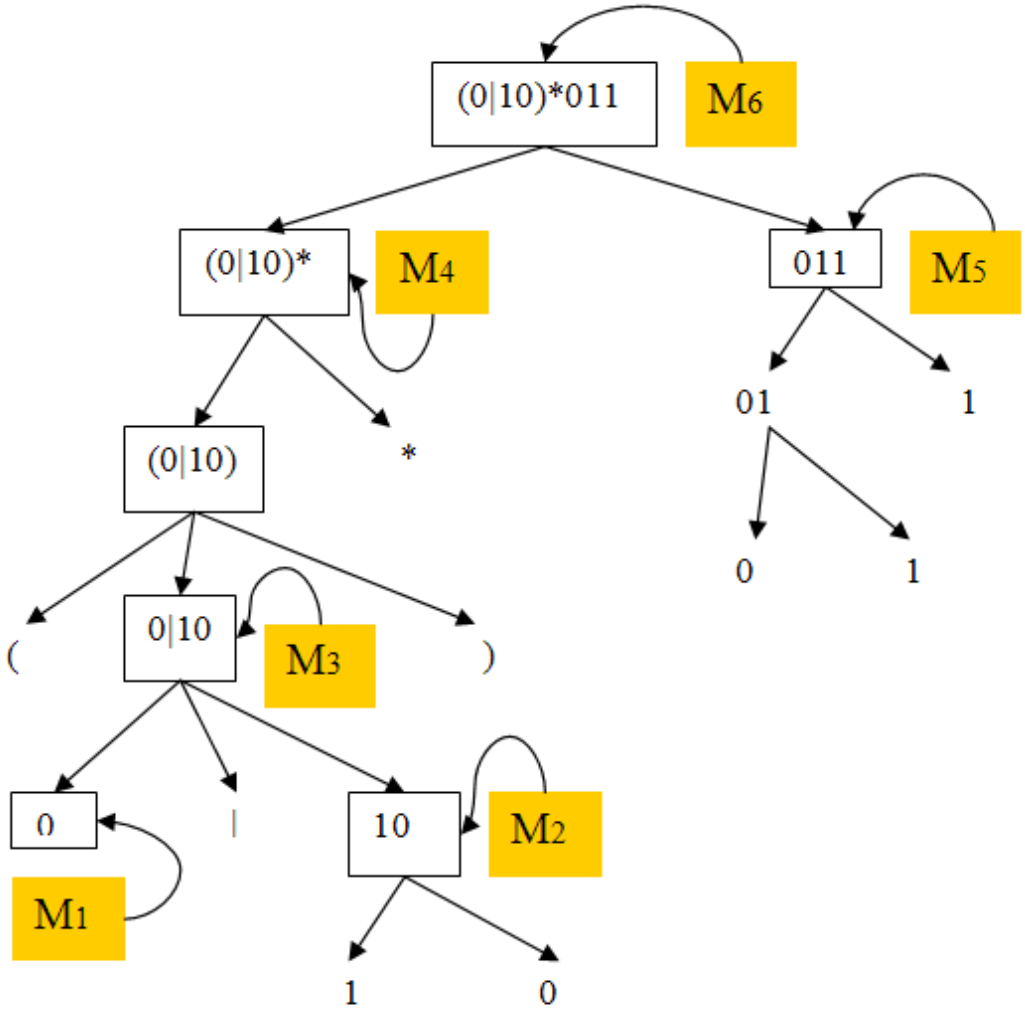
La demostración de la parte de síntesis proporciona un método para **traducir una ER en un AF** $\Rightarrow$ **algoritmo *ERtoAF***.

Este proceso se lleva a cabo en el **motor de ER** (*regex engine*) de ciertas herramientas, que usando la tabla de transición del AF proporciona las funciones para **resolver problemas de procesamiento de cadenas de patrón regular** (*regular pattern matching*).

De las ERs a los AFs

Ejemplo: traducción de ER (0|10)*011 *a AF*

IDEA: trocear la ER en expresiones simples (sin unión ni clausura), obtener un AF para cada una y combinar AFs parciales.



Algoritmo que traduce una ER en un AF (I)

FUNCIÓN **ERtoAF (R)**

ENTRADA: una expresión regular R
SALIDA: un autómata finito M tal que $L(R) = L(M)$

// Casos base de la función recursiva:

1 SI $R = \varnothing$ ENTONCES DEVUELVE (M_{vacio}); //autómata constante

2 SI $R = \lambda$ ENTONCES DEVUELVE (M_{lambda}); //autómata constante

3 SI R es una cadena de símbolos ENTONCES

- $M \leftarrow$ *construyeAFcadena*(R);
  //obtiene un AF que acepta sólo la cadena en R

- DEVUELVE (M);

Algoritmo que traduce una ER en un AF (II)

FUNCIÓN **ERtoAF (R)**

// Casos recursivos

4 SI R es de la forma $R_1 \circ R_2$ ENTONCES

- $M_1 \leftarrow ERtoAF(R_1)$; //llamada recursiva con R_1
- $M_2 \leftarrow ERtoAF(R_2)$; //llamada recursiva con R_2
- $M \leftarrow concatAF(M_1, M_2)$;
- DEVUELVE (M);

5 SI R es de la forma $R_1|R_2$ ENTONCES

- $M_1 \leftarrow ERtoAF(R_1)$;
- $M_2 \leftarrow ERtoAF(R_2)$;
- $M \leftarrow unionAF(M_1, M_2)$;
- DEVUELVE (M);

6 SI R es de la forma R_1^* ENTONCES

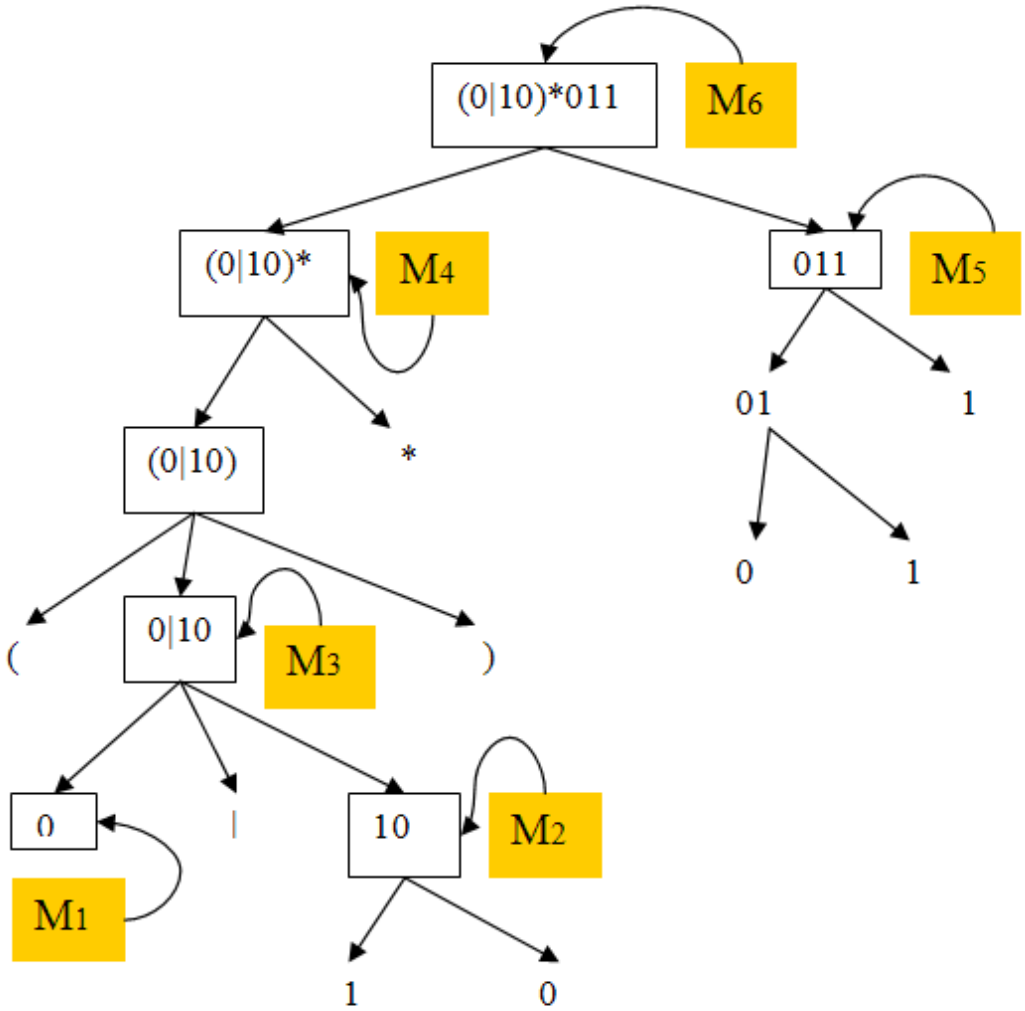
- $M_1 \leftarrow ERtoAF(R_1)$;
- $M \leftarrow clausuraAF(M_1)$;
- DEVUELVE (M);

7 EXIT $(syntaxError)$;

//COMPLEJIDAD: $O(n)$ donde n es el número de caracteres en R

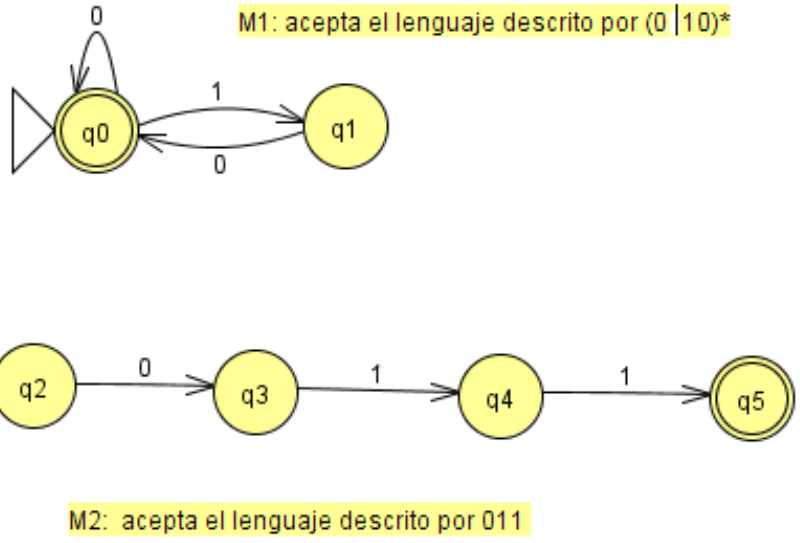
Ejemplo: aplicar algoritmo ERtoAF a la ER (0|10)*011

$\Rightarrow$ Obtener los autómatas para cada ER del árbol sin operador de clausura ni unión y combinar.



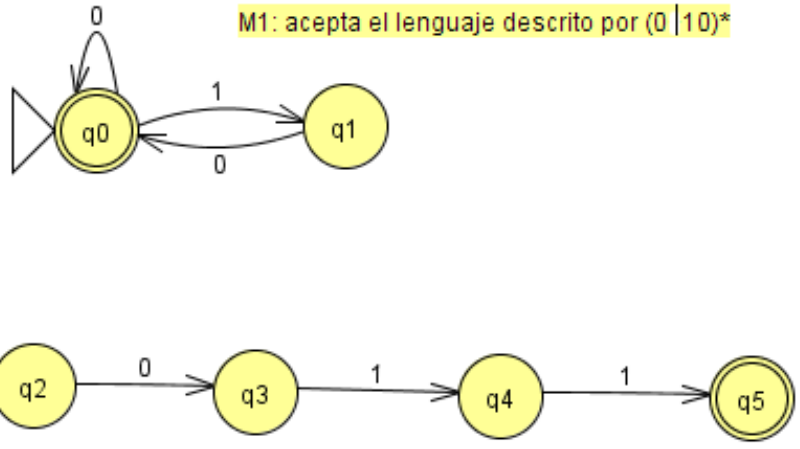
Traducción manual de ER a AF

Podemos **simplificar el proceso de traducción** para $(0|10)^*011$ considerando $(0|10)^*011 = R_1 \circ R_2$, donde $R_1 = (0|10)^*$ y $R_2 = 011$:

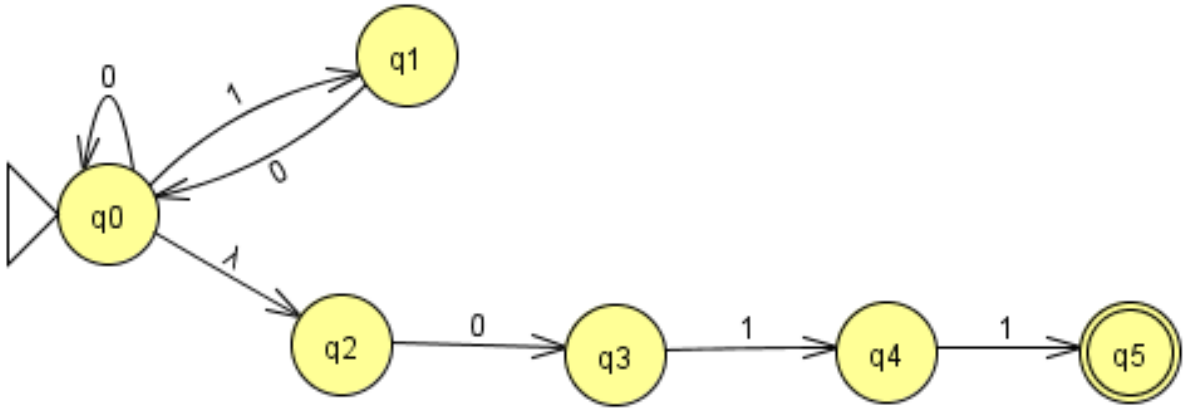


Traducción manual de ER a AF

Podemos **simplificar el proceso de traducción** para $(0|10)^*011$ considerando $(0|10)^*011 = R_1 \circ R_2$, donde $R_1 = (0|10)^*$ y $R_2 = 011$:



Se concatenan los autómatas M_1 y M_2:



Relación entre autómatas finitos y expresiones regulares

Teorema de Kleene: Un lenguaje puede ser descrito por una expresión regular si y sólo si puede ser aceptado un autómata finito.
El teorema completo se suele dividir en dos partes que afirman:

1. Síntesis: si un lenguaje puede ser descrito por una expresión regular entonces también puede ser aceptado un autómata finito.
2. **Análisis**: **si un lenguaje puede ser aceptado por autómata finito entonces también puede ser descrito por una expresión regular.**

- La demostración de la parte de análisis proporciona un método para **traducir un AF en una ER** $\Rightarrow$ **algoritmo *AFtoER***
- **Interés teórico:** termina de comprobarse que los autómatas finitos y las expresiones regulares son **formalismos igual de potentes** $\Rightarrow$ aceptan/describen los mismos tipos de lenguajes, los **lenguajes regulares**.

Sistema de ecuaciones de ER de un AF

Sea un AF $M = (Q, V, \delta, q_0, F)$, con estados $Q = \{q_0, \ldots, q_n\}$ y alfabeto $V = \{a_1, a_2, \ldots, a_k\}$.

Se obtiene **una ecuación de ERs para cada estado** $q_i \in Q$:

- Si q_i no es final entonces su ecuación tiene la siguiente forma:

$$q_i = R_0 q_0 \mid R_1 q_1 \mid \cdots \mid R_i q_i \mid \cdots \mid R_n q_n$$

- Si q_i es final entonces

$$q_i = R_0 q_0 \mid R_1 q_1 \mid \cdots \mid R_i q_i \mid \cdots \mid R_n q_n \mid \lambda$$

Sistema de ecuaciones de ER de un AF

Sea un AF $M = (Q, V, \delta, q_0, F)$, con estados $Q = \{q_0, \ldots, q_n\}$ y alfabeto $V = \{a_1, a_2, \ldots, a_k\}$.

Se obtiene **una ecuación de ERs para cada estado** $q_i \in Q$:

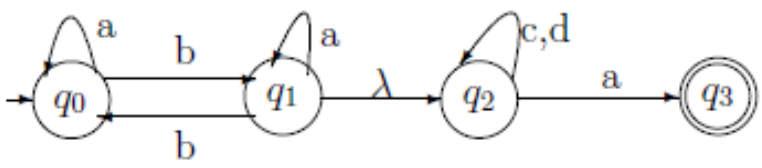
- Si q_i no es final entonces su ecuación tiene la siguiente forma:

$$q_i = R_0 q_0 \mid R_1 q_1 \mid \cdots \mid R_i q_i \mid \cdots \mid R_n q_n$$

- Si q_i es final entonces

$$q_i = R_0 q_0 \mid R_1 q_1 \mid \cdots \mid R_i q_i \mid \cdots \mid R_n q_n \mid \lambda$$

Ejemplo de sistema de ecuaciones de un autómata



$$q_0 = aq_0 \mid bq_1$$

Sistema de ecuaciones de ER de un AF

Sea un AF $M = (Q, V, \delta, q_0, F)$, con estados $Q = \{q_0, \ldots, q_n\}$ y alfabeto $V = \{a_1, a_2, \ldots, a_k\}$.

Se obtiene **una ecuación de ERs para cada estado** $q_i \in Q$:

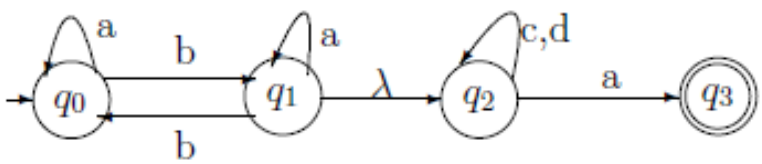
- Si q_i no es final entonces su ecuación tiene la siguiente forma:

$$q_i = R_0 q_0 \mid R_1 q_1 \mid \cdots \mid R_i q_i \mid \cdots \mid R_n q_n$$

- Si q_i es final entonces

$$q_i = R_0 q_0 \mid R_1 q_1 \mid \cdots \mid R_i q_i \mid \cdots \mid R_n q_n \mid \lambda$$

Ejemplo de sistema de ecuaciones de un autómata



$$q_0 = aq_0 \mid bq_1$$

$$q_1 = bq_0 \mid aq_1 \mid \lambda q_2$$

Sistema de ecuaciones de ER de un AF

Sea un AF $M = (Q, V, \delta, q_0, F)$, con estados $Q = \{q_0, \ldots, q_n\}$ y alfabeto $V = \{a_1, a_2, \ldots, a_k\}$.

Se obtiene **una ecuación de ERs para cada estado** $q_i \in Q$:

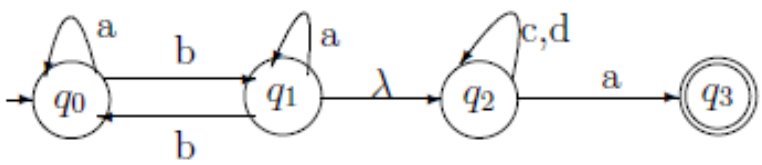
- Si q_i no es final entonces su ecuación tiene la siguiente forma:

$$q_i = R_0 q_0 \mid R_1 q_1 \mid \cdots \mid R_i q_i \mid \cdots \mid R_n q_n$$

- Si q_i es final entonces

$$q_i = R_0 q_0 \mid R_1 q_1 \mid \cdots \mid R_i q_i \mid \cdots \mid R_n q_n \mid \lambda$$

Ejemplo de sistema de ecuaciones de un autómata



$$q_0 = aq_0 \mid bq_1$$
$$q_1 = bq_0 \mid aq_1 \mid \lambda q_2$$
$$q_2 = (c|d)q_2 \mid aq_3$$

Sistema de ecuaciones de ER de un AF

Sea un AF $M = (Q, V, \delta, q_0, F)$, con estados $Q = \{q_0, \ldots, q_n\}$ y alfabeto $V = \{a_1, a_2, \ldots, a_k\}$.

Se obtiene **una ecuación de ERs para cada estado** $q_i \in Q$:

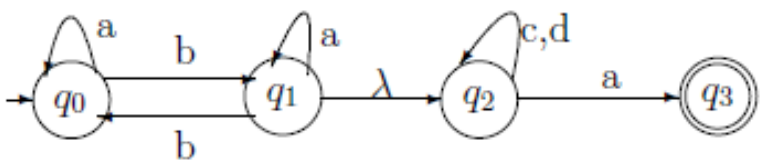
- Si q_i no es final entonces su ecuación tiene la siguiente forma:

$$q_i = R_0 q_0 \mid R_1 q_1 \mid \cdots \mid R_i q_i \mid \cdots \mid R_n q_n$$

- Si q_i es final entonces

$$q_i = R_0 q_0 \mid R_1 q_1 \mid \cdots \mid R_i q_i \mid \cdots \mid R_n q_n \mid \lambda$$

Ejemplo de sistema de ecuaciones de un autómata



$$q_0 = aq_0 \mid bq_1$$
$$q_1 = bq_0 \mid aq_1 \mid \lambda q_2$$
$$q_2 = (c|d)q_2 \mid aq_3$$
$$q_3 = \lambda$$

Algoritmo AFtoER

ENTRADA: $M = (Q, V, \delta, q_0, F)$, con estados $Q = \{q_0, \ldots, q_n\}$.

SALIDA: una expresión regular R_0 tal que $L(R_0) = L(M)$

- Obtener el sistema de ecuaciones del autómata M;

  / Bucle de transformación para eliminar incógnitas*/*
- FOR $i = n$ (último estado) DOWN-TO 0 (estado inicial).
  - Obtener $otrosTerminos[q_i]$:= unión (con $|$) de todos los términos de la parte derecha de la ecuación para q_i, excepto el término en q_i;
    Si sólo aparece el término en q_i entonces $otrosTerminos[q_i] = \varnothing$;
  - Obtener $\boxed{valor[q_i] := coef[q_i]^* \circ otrosTerminos[q_i]}$ donde $coef[q_i]$ es la ER que aparece como coeficiente de q_i;
    Si no hay término en q_i entonces $valor[q_i] := otrosTerminos[q_i]$;
  - Dejar ecuación para q_i expresada como $q_i = valor[q_i]$;

  - FOR $j = i - 1$ DOWN-TO 0 */*Bucle de sustitución */*
    - Sustituir en ecuación para q_j la incógnita q_i por $valor[q_i]$;
    - Transformar ecuación de q_j aplicando propiedad distributiva y agrupar coeficientes del mismo estado con ( ) y uniendo con $|$.
- DEVUELVE $(R_0 := valor[q_0])$; */* ER buscada */*

(Ej.: aplicar al AF anterior)

Algoritmo AFtoER: pasos extra

En lugar de devolver solución para estado inicial entra en bucle que **obtiene ER-solución para el resto de estados**, para los que $valor[q_i]$ aun puede contener incógnitas.

- FOR $i = 0$ TO $n - 1$ /* *sustituye y resuelve* */
  - FOR $j = i + 1$ to n
    sustituir en ecuación para q_j la incógnita q_i por $valor[q_i]$;
  - END-FOR-j
  - $\boxed{valor[q_{i+1}] := coef[q_{i+1}]^* \circ otrosTerminos[q_{i+1}]}$;
- $R_0 := valor[q_0], \ldots, R_n =: valor[q_n]$;
- DEVUELVE $(R_0, \ldots, R_n)$;
  (Ej.: seguir resolviendo sistema anterior)

Algoritmo AFtoER: pasos extra

En lugar de devolver solución para estado inicial entra en bucle que **obtiene ER-solución para el resto de estados**, para los que $valor[q_i]$ aun puede contener incógnitas.

- FOR $i = 0$ TO $n - 1$ /* *sustituye y resuelve* */
  - FOR $j = i + 1$ to n
    sustituir en ecuación para q_j la incógnita q_i por $valor[q_i]$;
  - END-FOR-j
  - $\boxed{valor[q_{i+1}] := coef[q_{i+1}]^* \circ otrosTerminos[q_{i+1}]}$;
- $R_0 := valor[q_0], \ldots, R_n =: valor[q_n]$;
- DEVUELVE $(R_0, \ldots, R_n)$;
  (Ej.: seguir resolviendo sistema anterior)

Propiedad: el método *AFtoER* asegura que: R_i es la ER solución para el estado q_i entonces para cualquier cadena x se cumple que:

$$x \in L(R_i) \Leftrightarrow (q_i, x) \Rightarrow^* (q_F, \lambda)$$

es decir, R_i describe las cadenas que se pueden aceptar desde q_i.

Prop. de cierre de los lenguajes regulares (I)

Teorema: la clase $\mathcal{L}_{reg}$ de lenguajes regulares es **cerrada** bajo las siguientes operaciones:

1. unión
2. concatenación
3. clausura
4. complementación
5. intersección

Prop. de cierre de los lenguajes regulares (I)

Teorema: la clase $\mathcal{L}_{reg}$ de lenguajes regulares es **cerrada** bajo las siguientes operaciones:

1. unión
2. concatenación
3. clausura
4. complementación
5. intersección

Dem.: Sean R_1 y R_2 las expresiones regulares que describen a los lenguajes L_1 y L_2.

1. La ER $R_1 \mid R_2$ muestra que la unión de los dos lenguajes regulares es regular, ya que: $L_1 \cup L_2 = L(R_1) \cup L(R_2) = L(R_1 \mid R_2)$

Prop. de cierre de los lenguajes regulares (I)

Teorema: la clase $\mathcal{L}_{reg}$ de lenguajes regulares es **cerrada** bajo las siguientes operaciones:

1. unión
2. concatenación
3. clausura
4. complementación
5. intersección

Dem.: Sean R_1 y R_2 las expresiones regulares que describen a los lenguajes L_1 y L_2.

1. La ER $R_1 \mid R_2$ muestra que la unión de los dos lenguajes regulares es regular, ya que: $L_1 \cup L_2 = L(R_1) \cup L(R_2) = L(R_1 \mid R_2)$
2. La ER $R_1 \circ R_2$ muestra que $L_1 \circ L_2$ es regular.

Prop. de cierre de los lenguajes regulares (I)

Teorema: la clase $\mathcal{L}_{reg}$ de lenguajes regulares es **cerrada** bajo las siguientes operaciones:

1. unión
2. concatenación
3. clausura
4. complementación
5. intersección

Dem.: Sean R_1 y R_2 las expresiones regulares que describen a los lenguajes L_1 y L_2.

1. La ER $R_1 \mid R_2$ muestra que la unión de los dos lenguajes regulares es regular, ya que: $L_1 \cup L_2 = L(R_1) \cup L(R_2) = L(R_1 \mid R_2)$
2. La ER $R_1 \circ R_2$ muestra que $L_1 \circ L_2$ es regular.
3. La ER R_1^* muestra que L_1^* es regular.

Prop. de cierre de lenguajes regulares (II)

Dem.:

- Cierre bajo complementación: **probamos que si un lenguaje L_r es regular entonces $\overline{L_r}$ es regular.** Al ser L_r regular existe un autómata M_r que lo acepta.
  Puede obtenerse un AF que acepte $\overline{L_r}$ (lo cual prueba que $\overline{L_r}$ es regular) aplicando el siguiente método algorítmico:
  1. $M_D \leftarrow AFNDtoAFD(M_r)$;
  2. $\overline{M_D} \leftarrow complementaAFD(M_D)$; (acepta $\overline{L_r}$)

Prop. de cierre de lenguajes regulares (II)

Dem.:

- Cierre bajo complementación: **probamos que si un lenguaje L_r es regular entonces $\overline{L_r}$ es regular.** Al ser L_r regular existe un autómata M_r que lo acepta.
  Puede obtenerse un AF que acepte $\overline{L_r}$ (lo cual prueba que $\overline{L_r}$ es regular) aplicando el siguiente método algorítmico:
  1. $M_D \leftarrow AFNDtoAFD(M_r)$;
  2. $\overline{M_D} \leftarrow complementaAFD(M_D)$; (acepta $\overline{L_r}$)

Prop. de cierre de lenguajes regulares (II)

Dem.:

- Cierre bajo complementación: **probamos que si un lenguaje L_r es regular entonces $\overline{L_r}$ es regular.** Al ser L_r regular existe un autómata M_r que lo acepta.
  Puede obtenerse un AF que acepte $\overline{L_r}$ (lo cual prueba que $\overline{L_r}$ es regular) aplicando el siguiente método algorítmico:
  1. $M_D \leftarrow AFNDtoAFD(M_r)$;
  2. $\overline{M_D} \leftarrow complementaAFD(M_D)$; (acepta $\overline{L_r}$)
- Cierre bajo intersección: **probamos que si L_1 y L_2 son regulares entonces entonces $L_1 \cap L_2$ es regular.** Al ser L_1 y L_2 regulares existen M_1 y M_2 tal que $L(M_1) = L_1$ y $L(M_2) = L_2$.
  Puede obtenerse un AF que acepte $L_1 \cap L_2$ teniendo en cuenta que:

  $$L_1 \cap L_2 = \overline{\overline{L_1 \cap L_2}} = \overline{\overline{L_1} \cup \overline{L_2}} \qquad \text{(ley de De Morgan)}$$

  1. $M_{1D} \leftarrow AFNDtoAFD(M_1)$; $M_{2D} \leftarrow AFNDtoAFD(M_2)$;
  2. $\overline{M_{1D}} \leftarrow complementaAFD(M_{1D})$; (acepta $\overline{L_1}$)
  3. $\overline{M_{2D}} \leftarrow complementaAFD(M_{2D})$; (acepta $\overline{L_2}$)
  4. $M \leftarrow unionAF(\overline{M_{1D}}, \overline{M_{2D}})$;
  5. $M_i \leftarrow complementaAFD(AFNDtoAFD(M))$; (acepta $L_1 \cap L_2$)

Limitaciones de los AFs y las ERs

- **No todos los lenguajes son regulares:** no pueden ser descritos por expresiones regulares ni aceptados por autómatas finitos.
- **Justificar que un lenguaje no es regular** $\Rightarrow$ argumentar que no puede ser aceptado por un AF o descrito por una ER.

Limitaciones de los AFs y las ERs

- **No todos los lenguajes son regulares:** no pueden ser descritos por expresiones regulares ni aceptados por autómatas finitos.
- **Justificar que un lenguaje no es regular** $\Longrightarrow$ argumentar que no puede ser aceptado por un AF o descrito por una ER.

  Ej.: El lenguaje $L_{pbal}\{(^k)^k \mid k \geq 0\}$ no es regular.

Limitaciones de los AFs y las ERs

- **No todos los lenguajes son regulares:** no pueden ser descritos por expresiones regulares ni aceptados por autómatas finitos.
- **Justificar que un lenguaje no es regular** $\Longrightarrow$ argumentar que no puede ser aceptado por un AF o descrito por una ER.

  Ej.: El lenguaje $L_{pbal}\{(^k)^k \mid k \geq 0\}$ no es regular.

  Ej.: $L_{0i1} = \{w \in \{0,1\}^* \mid ceros(w) = unos(w)\}$ no es regular.

Limitaciones de los AFs y las ERs

- **No todos los lenguajes son regulares:** no pueden ser descritos por expresiones regulares ni aceptados por autómatas finitos.
- **Justificar que un lenguaje no es regular** $\Rightarrow$ argumentar que no puede ser aceptado por un AF o descrito por una ER.

  Ej.: El lenguaje $L_{pbal}\{(^k)^k \mid k \geq 0\}$ no es regular.

  Ej.: $L_{0i1} = \{w \in \{0,1\}^* \mid ceros(w) = unos(w)\}$ no es regular.

  PROBLEMA RESUELTO: demuestra que el lenguaje L_{0i1} no es regular es partiendo del hecho de que el lenguaje $L_s = \{0^k 1^k \mid k \geq 0\}$ no es regular y **usando las propiedades de cierre de los lenguajes regulares**.

Limitaciones de los AFs y las ERs

- **No todos los lenguajes son regulares:** no pueden ser descritos por expresiones regulares ni aceptados por autómatas finitos.
- **Justificar que un lenguaje no es regular** $\Longrightarrow$ argumentar que no puede ser aceptado por un AF o descrito por una ER.

  Ej.: El lenguaje $L_{pbal}\{(^k)^k \mid k \geq 0\}$ no es regular.

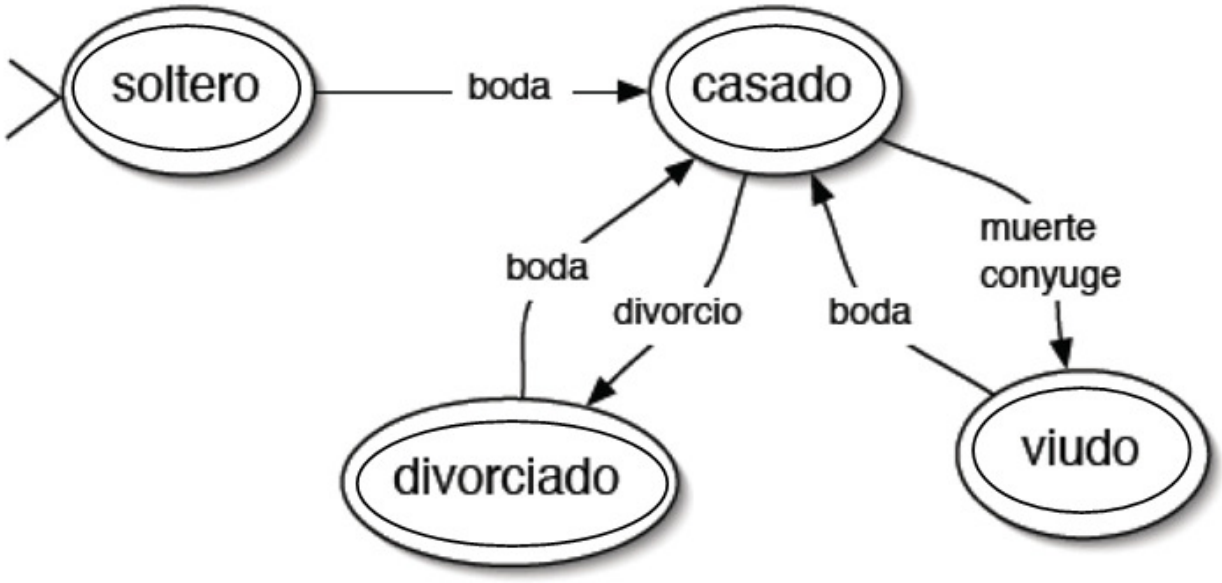
  Ej.: $L_{0i1} = \{w \in \{0,1\}^* \mid ceros(w) = unos(w)\}$ no es regular.

  PROBLEMA RESUELTO: demuestra que el lenguaje L_{0i1} no es regular es partiendo del hecho de que el lenguaje $L_s = \{0^k 1^k \mid k \geq 0\}$ no es regular y **usando las propiedades de cierre de los lenguajes regulares**.
- Para procesar cadenas de lenguajes más complejos se necesita **aumentar la potencia de la máquina teórica**, por ejemplo con:
  - **Autómata con pila**: mecanismo con número finito de estados + memoria tipo pila
  - **Máquina de Turing**: número finito de estados + memoria secuencial (es un modelo simplificado de computador).

Autómatas finitos y sistemas de eventos discretos

En general los autómatas finitos se usan en el modelado, simulación y análisis de sistemas de eventos discretos y se llaman también máquinas de estado finito.

Ejemplo: modelo de evolución de estados civiles



Autómatas finitos y sistemas de eventos discretos

En general los autómatas finitos se usan en el modelado, simulación y análisis de sistemas de eventos discretos y se llaman también máquinas de estado finito.

Autómatas finitos y sistemas de eventos discretos

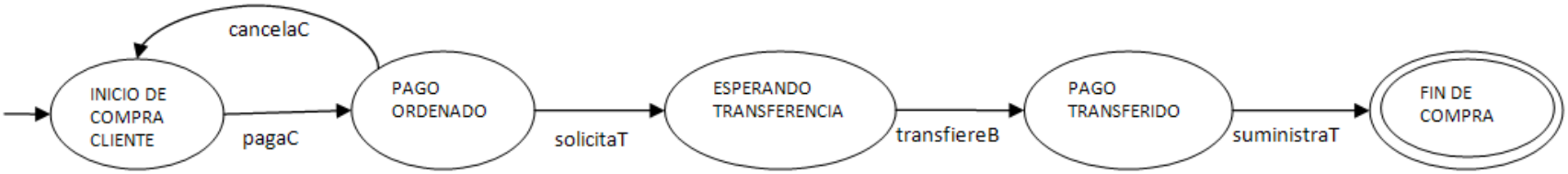
En general los autómatas finitos se usan en el modelado, simulación y análisis de sistemas de eventos discretos y se llaman también máquinas de estado finito.

Ejemplo: eventos y estados en protocolos de comunicación

Para poder utilizar *dinero electrónico* entre un cliente, un banco y una tienda se necesita respetar un **protocolo de comunicación** en la secuencia de eventos de envío y recepción de mensajes.

Ej.: Mostramos un autómata finito para modelar la evolución de estados en la comunicación desde el punto de vista de la tienda.

$$Eventos = \{pagaC, cancelaC, solicitaT, suministraT, transfiereB\}$$



Preguntas de evaluación en apuntes

- Contiene problemas de **razonamiento**, **aplicación o cálculo** y **preguntas tipo test**.
- Son una colección de preguntas de examen, aunque no incluye todo tipo de preguntas posibles de examen.

- **Problemas resueltos:** actividad para auto-evaluación.

- **Problemas propuestos:** actividad para resolver en parte en clase (teoría o prácticas).

- **Preguntas tipo test:** actividad para resolver en parte en clase (teoría o prácticas).