

Modelos básicos de Computación. Máquinas de Turing.  
Autómatas Finitos. Autómatas con Pila.

---

Tema 5(I). Autómatas Finitos.

Algorítmica y Modelos de Computación

# Índice

---

1. Introducción. **Lenguajes, Gramáticas Y Autómatas**
2. Autómatas Finitos.
  - 2.1. Introducción..
  - 2.2. Autómatas finitos deterministas (A.F.D.).
  - 2.3. Minimización de Autómatas finitos.
  - 2.4. Autómatas finitos no deterministas (A.F.N.D.).
  - 2.5. Equivalencia entre A.F.D. y A.F.N.D.
  - 2.6. Autómatas finitos y gramáticas regulares.
  - 2.7. Expresiones regulares.
3. Autómatas con Pila.
  - 3.1. Introducción
  - 3.2. Definición de Autómatas con pila.
  - 3.3. Lenguaje aceptado por un autómata con pila.
  - 3.4. Autómatas con pila y lenguajes libres del contexto.
    - 3.4.1. Reconocimiento descendente. Gramáticas LL(k).
      - 3.4.1.1. Proceso de Análisis Sintáctico Descendente.
      - 3.4.1.2. Analizadores LL y autómatas de pila no deterministas
    - 3.4.2. Reconocimiento ascendente. Gramáticas LR(k).
      - 3.4.2.2. Proceso de Análisis Sintáctico Ascendente.
      - 3.4.2.3. Analizadores LR y autómatas con pila no deterministas
4. Máquinas de Turing.

# 1. Introducción.

---

- Se establece un **isomorfismo** entre la Teoría de Lenguajes Formales y la Teoría de Autómatas, estableciendo una conexión entre la clase de lenguajes generados por ciertos tipos de gramáticas y la clase de lenguajes reconocibles por ciertas máquinas. Se detalla la jerarquía de lenguajes de Chomsky y se establecen las siguientes relaciones :

Gramáticas	Lenguajes	Máquinas
Sin restricciones o de Tipo 0	Sin restricciones o de Tipo 0	Máquina de Turing
Sensible al contexto o de Tipo 1	Sensible al contexto o de Tipo 1	Autómata linealmente acotado
Libre de contexto o de Tipo 2	Libre de contexto o de Tipo 2	Autómata con pila
Regular o de Tipo 3	Regular o de Tipo 3	Autómata Finito

- Cada uno de estos tipos/máquinas añade restricciones al tipo/máquina del nivel superior.

# 2. Autómatas Finitos.

---

## 1. Introducción. Lenguajes, Gramáticas Y Autómatas

## 2. Autómatas Finitos.

### 2.1. Introducción..

### 2.2. Autómatas finitos deterministas (A.F.D.).

### 2.3. Minimización de Autómatas finitos.

### 2.4. Autómatas finitos no deterministas (A.F.N.D.).

### 2.5. Equivalencia entre A.F.D. y A.F.N.D.

### 2.6. Autómatas finitos y gramáticas regulares.

### 2.7. Expresiones regulares.

## 3. Autómatas con Pila.

### 3.1. Introducción

### 3.2. Definición de Autómatas con pila.

### 3.3. Lenguaje aceptado por un autómata con pila.

### 3.4. Autómatas con pila y lenguajes libres del contexto.

#### 3.4.1. Reconocimiento descendente. Gramáticas LL(k).

##### 3.4.1.1. Proceso de Análisis Sintáctico Descendente.

##### 3.4.1.2. Analizadores LL y autómatas de pila no deterministas

#### 3.4.2. Reconocimiento ascendente. Gramáticas LR(k).

##### 3.4.2.2. Proceso de Análisis Sintáctico Ascendente.

##### 3.4.2.3. Analizadores LR y autómatas de pila no deterministas

## 4. Máquinas de Turing.

---

## 2.1. Autómatas Finitos. Introducción.

---

- Autómatas con salidas:
  - Máquina de Moore. Salida asociada al estado.
  - Máquina de Mealy. Salida asociada a la transición.
- Autómatas reconocedores de lenguajes regulares:
  - Autómata Finito Determinista (AFD)
  - Autómata Finito No Determinista (AFND)
    - salidas:
      - reconoce , no reconoce cadenas de un lenguaje regular;
      - 0 o 1

## 2.1. Autómatas Finitos. Introducción.

---

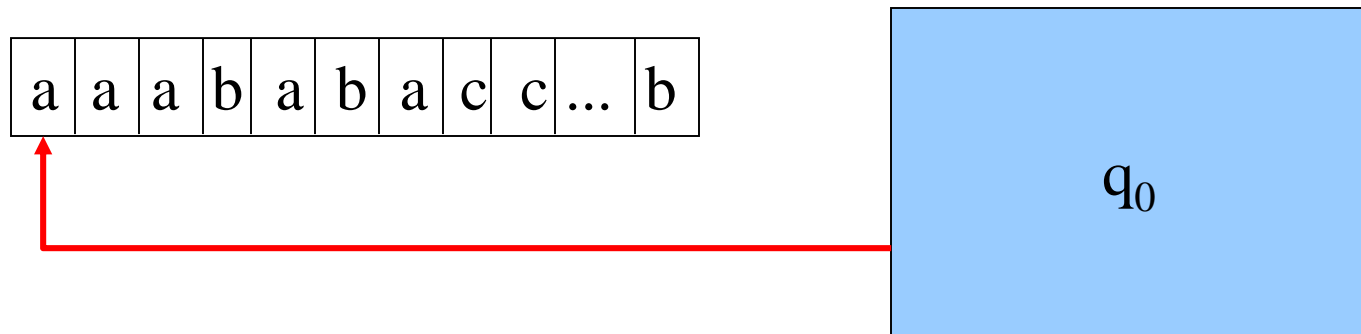
- Un autómata finito es un conjunto de estados y un control que se mueve de un estado a otro en respuesta a entradas externas.
- Los autómatas finitos se pueden clasificar en función del tipo de control como:
  - *Deterministas*, el autómata únicamente puede estar en un estado en un momento determinado.
  - *No Deterministas*, el autómata puede estar en varios estados simultáneamente.
- Ambos definen los mismos lenguajes (regulares), sin embargo los No deterministas permiten describir más eficientemente determinados problemas.

## 2.1. Autómatas Finitos. Introducción.

---

### □ ¿Cómo procesa entradas un AFD?

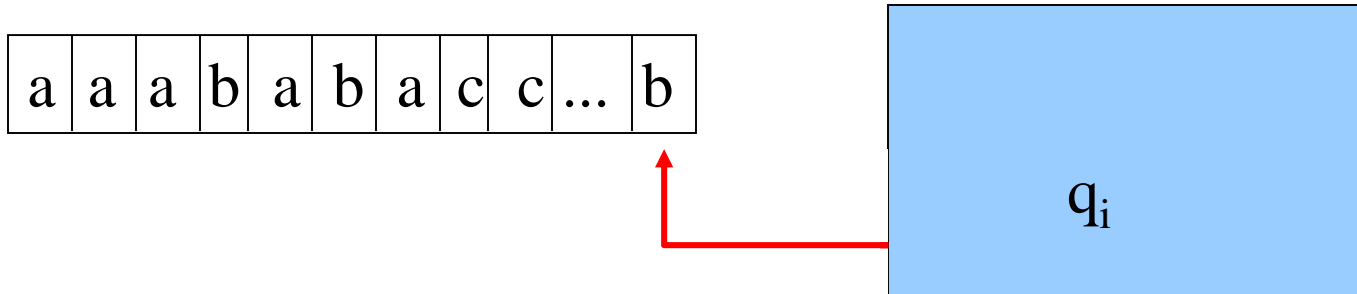
- La entrada a un AF es un conjunto de símbolos tomados del alfabeto de entrada  $\Sigma$ , no hay límite en tamaño de la cadena.
- Existe un “puntero” que en cada momento apunta a una posición de la cadena de entrada.
- El autómata está siempre en un estado de  $Q$ , inicialmente se encuentra en el estado  $q_0$ .



## 2.1. Autómatas Finitos. Introducción.

---

- ¿Cómo procesa entradas un AFD?
  - En cada paso el autómata lee un símbolo de la entrada y según el estado en el que se encuentre, cambia de estado y pasa a leer otro símbolo.



- Así sucesivamente hasta que se terminen de leer todos los símbolos de la cadena de entrada.
- Si en ese momento el AF está en un estado  $q_i$  de  $F$ , se dice que acepta la cadena, en caso contrario la rechaza.



## 2.2. Autómatas finitos deterministas(AFD). Definición.

---

- Un **autómata finito** es una quintupla  $M=(Q,\Sigma,f,q_0,F)$  :
- $Q$  es un conjunto finito llamado *conjunto de estados*.
  - $\Sigma$  es un conjunto finito de símbolos, llamado *alfabeto de entrada*.
  - $f$  es una aplicación llamada *función de transición*
- $$f: Q \times \Sigma \rightarrow Q$$
- $q_0$  es un elemento de  $Q$ , llamado *estado inicial*.
  - $F$  es un subconjunto de  $Q$ , llamado *conjunto de estados finales*.

## 2.2. Representación AFD.

### □ Tablas de transición.

- Es una representación clásica de una función con dos argumentos.
- En las filas se colocarán los estados y en las columnas los símbolos del alfabeto de entrada.
- Cada intersección fila (estado  $q$ ) - columna (carácter  $a$ ) corresponde al estado  $f(q,a)$ .
- El estado inicial se representa con  $\rightarrow$
- Los estados finales con un  $*$

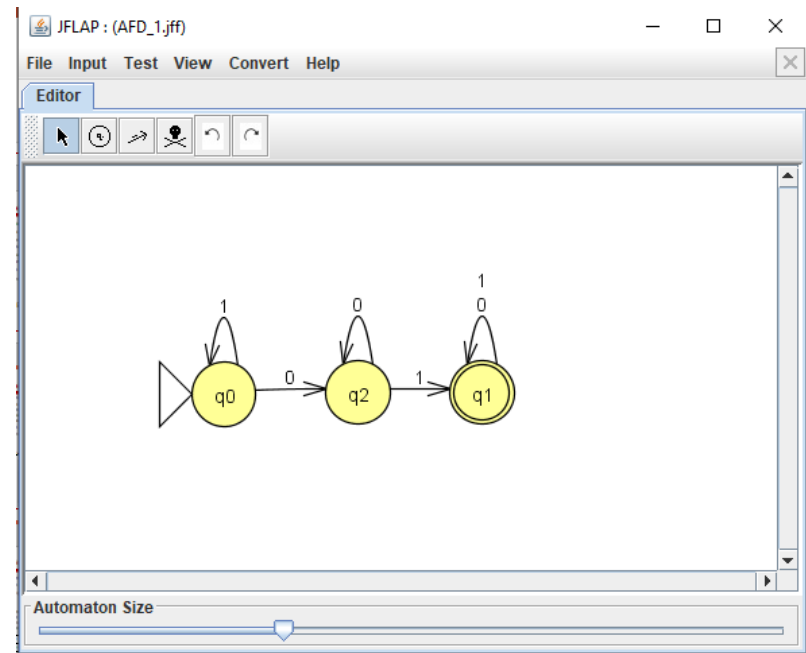
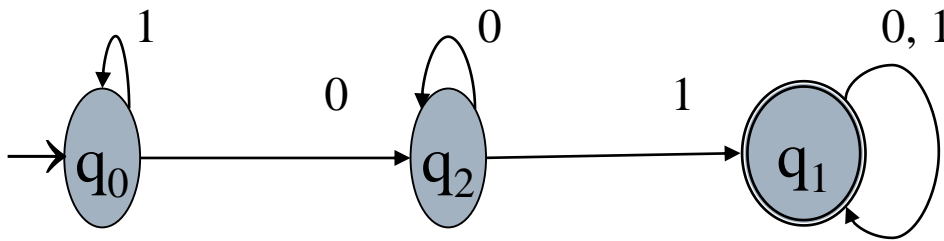
Ejemplo:

	0	1
$\rightarrow q_0$	$q_2$	$q_0$
$*q_1$	$q_1$	$q_1$
$q_2$	$q_2$	$q_1$

## 2.2. Representación AFD.

### □ Diagramas de transición.

- Es un grafo en el que los vértices representan los distintos estados y los arcos las transiciones entre los estados.
- Cada arco va etiquetado con el símbolo que corresponde a dicha transición.
- El estado inicial se representa con  $\rightarrow$
- Los estados finales con un con doble círculo.



## 2.2. Representación AFD.

### □ Determinismo porque:

■ No existen transiciones  $\lambda$

■  $\forall q \in Q, \forall a \in \Sigma \exists$  una única  $f(q,a)$  :

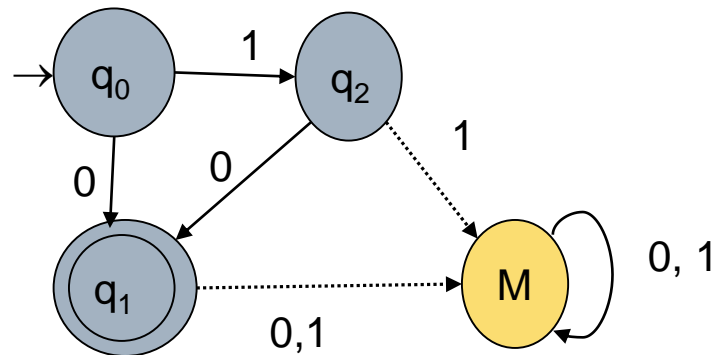
□ una sola arista etiquetada con  $a$  para cada símbolo;

□ Para cada entrada en la tabla un solo estado

□ La indeterminación en el caso que falten transiciones para algunas entradas se resuelve incluyendo un nuevo estado, llamado de **absorción** o **muerto**, al cual llegan todas las transiciones no definidas.

■ Ejemplo:

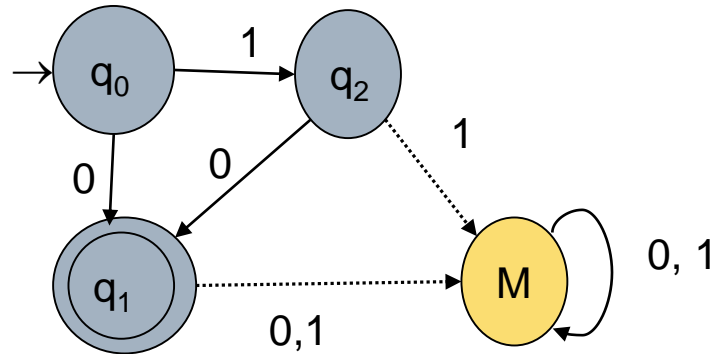
	0	1
→ q <sub>0</sub>	q <sub>1</sub>	q <sub>2</sub>
*q <sub>1</sub>	-	-
q <sub>2</sub>	q <sub>1</sub>	-



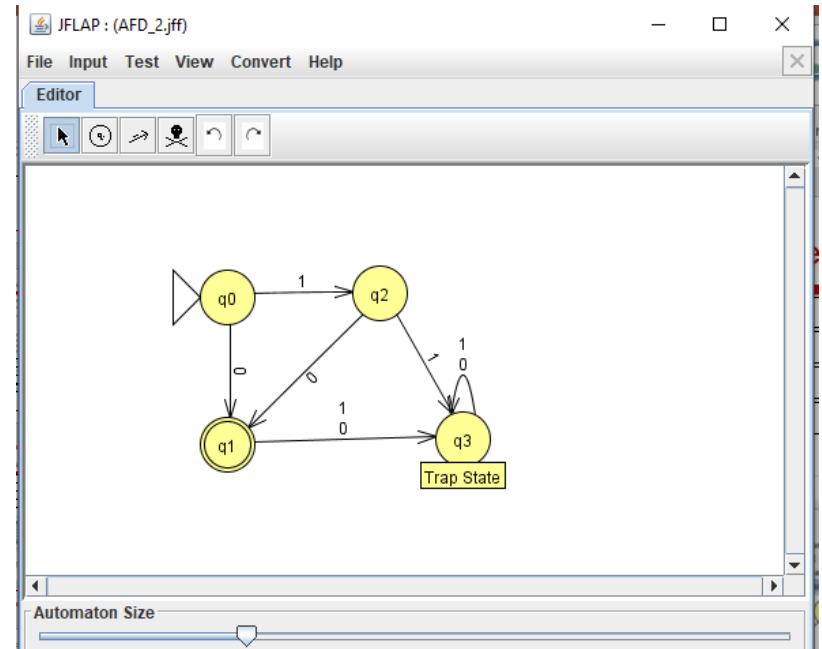
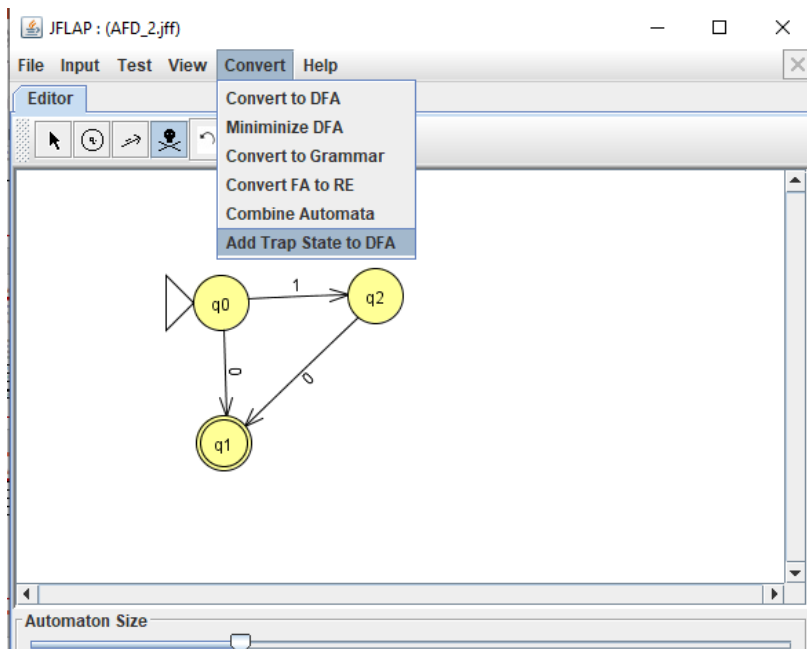
	0	1
→ q <sub>0</sub>	q <sub>1</sub>	q <sub>2</sub>
*q <sub>1</sub>	<b>M</b>	<b>M</b>
q <sub>2</sub>	q <sub>1</sub>	<b>M</b>
<b>M</b>	<b>M</b>	<b>M</b>

## 2.2. Representación AFD.

	0	1
→ q <sub>0</sub>	q <sub>1</sub>	q <sub>2</sub>
*q <sub>1</sub>	-	-
q <sub>2</sub>	q <sub>1</sub>	-



	0	1
→ q <sub>0</sub>	q <sub>1</sub>	q <sub>2</sub>
*q <sub>1</sub>	M	M
q <sub>2</sub>	q <sub>1</sub>	M
M	M	M



## 2.2. AFD. Extensión de f a palabras.

---

- Si  $M=(Q,\Sigma,f,q_0,F)$  es un AFD se define la función de transición asociada a palabras como la función

$$f': Q \times \Sigma^* \rightarrow Q$$

dada por:

- $f'(q,\lambda)=q$
- $f'(q,a)=f(q,a)$
- $f'(q,ax)=f'(f(q,a),x)$

donde  $x \in \Sigma^*$  y  $a \in \Sigma$

## 2.2. Lenguaje aceptado por un AFD.

---

- Una cadena  $x \in \Sigma^*$  es *aceptada* por un autómata  $M=(Q,\Sigma,f,q_0,F)$  si y solo si  $f'(q_0,x) \in F$ . En otro caso la cadena es *rechazada* por el autómata.
- El *lenguaje aceptado* o *reconocido* por un autómata es el conjunto de las palabras de  $\Sigma^*$  que acepta:

$$L(M)=\{ x \in \Sigma^* / f'(q_0,x) \in F \}$$

## 2.2. Simulación algorítmica de un AFD

---

- **Entrada:** cadena de entrada **x** que termina con un carácter fin de cadena o fin de archivo (FDC).
- **Salida:** La respuesta **ACEPTADA** si el autómata reconoce x **NO ACEPTADA** en caso contrario
- **Método:** aplicar f al estado al cual hay una transición desde el estado q a un carácter de entrada c

**función** reconocer()

    q=q<sub>0</sub>

    c= leer\_carácter()

**mientras** c != FDC

        q=f(q,c)

        c= leer\_carácter()

**fmientras**

**si** q ∈ F entonces

        Devolver(ACEPTADA)

**sino**

        Devolver(NO ACEPTADA)

**fsi**

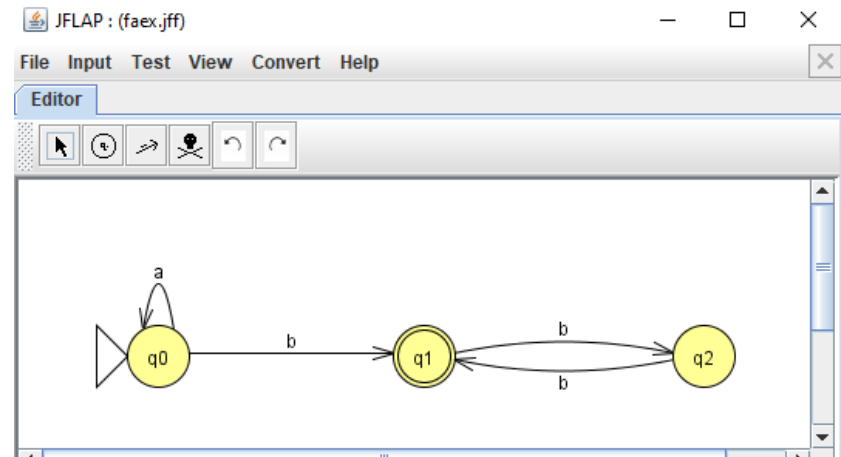
**ffunción**



## 2.2. Simulación algorítmica de un AFD. Ejemplo.

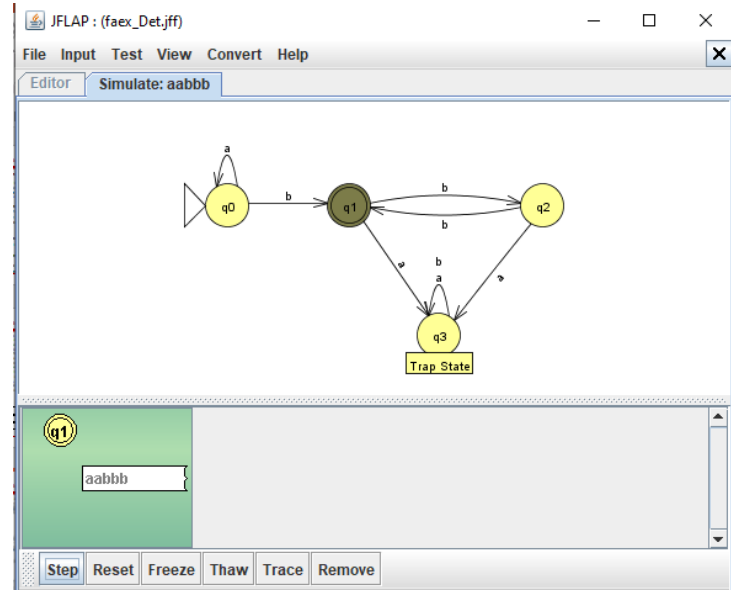
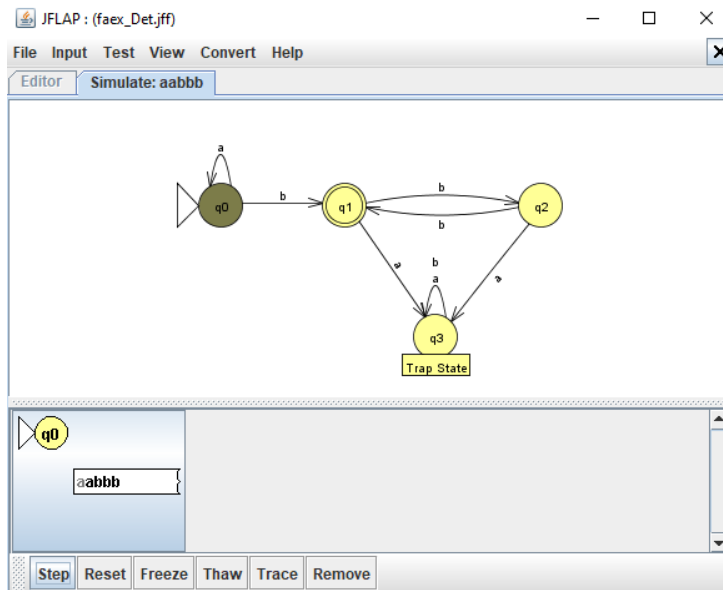
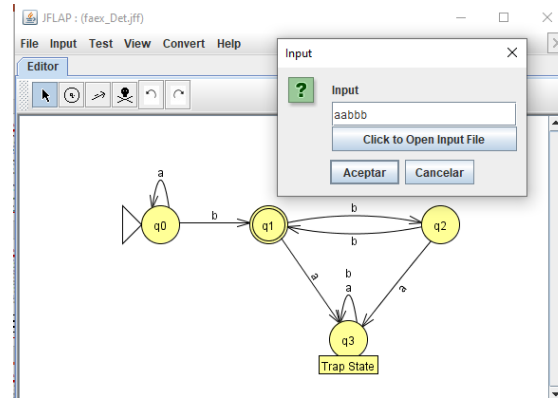
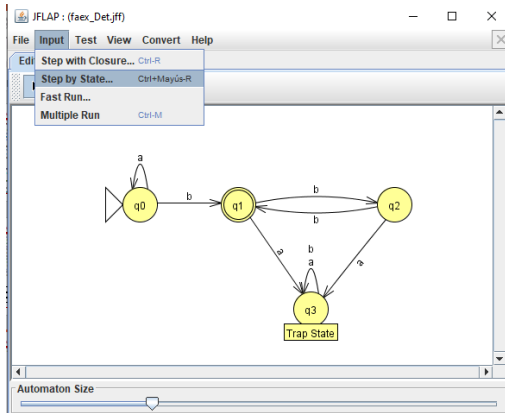
- ❑ Construir un AFD para el lenguaje  $L = \{a^m b^n / m \geq 0, n > 0, n \text{ es impar}\}$ . Es decir, crearemos un AFD que reconozca ese lenguaje de cualquier número de  $a$  seguido de cualquier número impar de  $b$ .
- ❑ Sabemos que las cadenas en nuestro lenguaje pueden comenzar con  $a$ , por lo que el estado inicial debe tener una transición de salida en  $a$ . También sabemos que puede comenzar con cualquier número de  $a$ , lo que significa que el FA debería estar en el mismo estado después de procesar la entrada de cualquier número de  $a$ . Por lo tanto, la transición de salida en  $a$ 
  - ❑ desde  $q_0$  vuelve a sí misma.
- ❑ A continuación, sabemos que las cadenas deben terminar con un número impar de  $b$ . Por lo tanto, sabemos que la transición de salida en  $b$  de  $q_0$  debe ser a un estado final, ya que se debe aceptar una cadena que termina con una  $b$ .
- ❑ Por último, sabemos que solo se deben aceptar las cadenas que terminen con un número impar de  $b$ . Por lo tanto, sabemos que  $q_1$  tiene una transición de salida en  $b$ , que no puede volver a  $q_1$ . Creamos una transición en  $b$  de  $q_1$  a  $q_2$ . Como el AF debería aceptar cadenas que terminen con un número impar de  $b$ , creamos otra transición en  $b$  de  $q_2$  a  $q_1$ .

	a	b
$\rightarrow q_0$	$q_0$	$q_1$
$^* q_1$	-	$q_2$
$q_2$	-	$q_1$



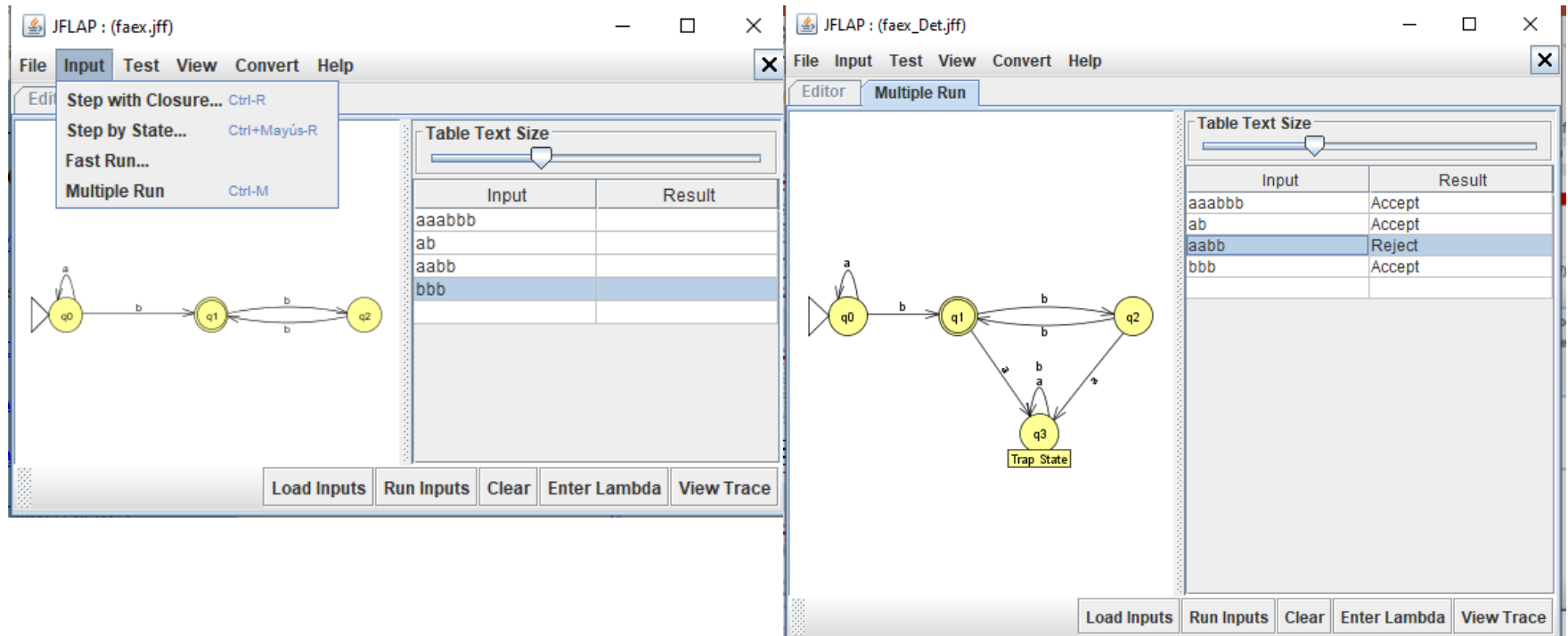
## 2.2. Simulación algorítmica de un AFD. Ejemplo.

❑ Ejecutamos el AFD para la cadena “abbb” del  $L = \{a^m b^n / m \geq 0, n > 0, n \text{ es impar}\}$ .



## 2.2. Simulación algorítmica de un AFD. Ejemplo.

- ❑ Ejecutamos el AFD para múltiples cadenas del  $L = \{a^m b^n / m \geq 0, n > 0, n \text{ es impar}\}$ .



## 2.3. Minimización de Autómatas finitos por el conjunto cociente.

---

### □ Algoritmo para construir el conjunto cociente $Q/E_n$

1.  $Q/E_1 = \{c_1 = q_i \in F, c_2 = q_j \in Q - F\}$

2. Sea  $Q/E_i = \{c_1, c_2, \dots, c_j\}$ .  $Q/E_{i+1}$  se construye:

□  $p$  y  $q$  están en la misma clase si y solo si  $p, q \in c_k$  y  $\forall a \in \Sigma$  se verifica que  $f(p, a)$  y  $f(q, a)$  están en la misma clase  $c_m$  de  $Q/E_i$

3. Si  $Q/E_i = Q/E_{i+1}$  entonces  $Q/E_i = Q/E$ , en caso contrario aplicar el paso **2.** partiendo de  $Q/E_{i+1}$

## 2.3. Minimización de Autómatas finitos por el conjunto cociente.

---

- Dado el AFD  $M=(Q,\Sigma,f,q_0,F)$   
existe un único AFD equivalente mínimo  
(Autómata del conjunto cociente)

$$M_m=(Q_m, \Sigma, f_m, q_{0m}, F_m)$$

■ Donde

$$Q_m=Q/E$$

$$\forall a \in \Sigma, f_m(c_i, a) = c_j \text{ si } \exists p \in c_j, q \in c_i / f(q, a) = p$$

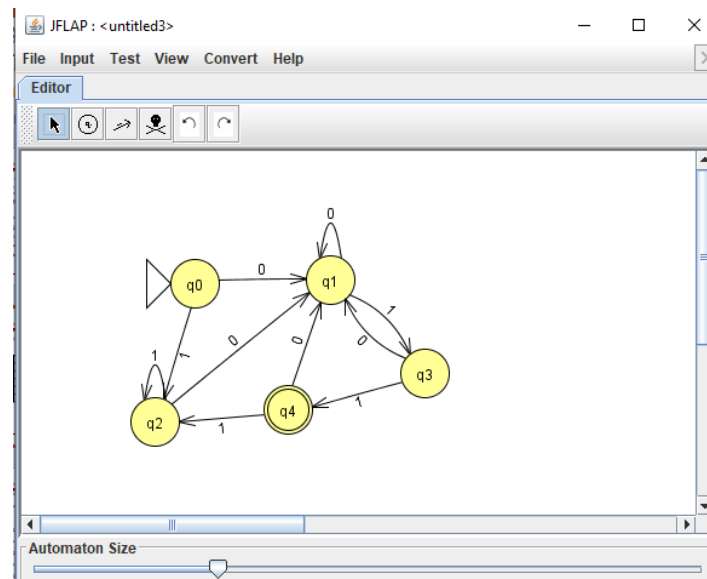
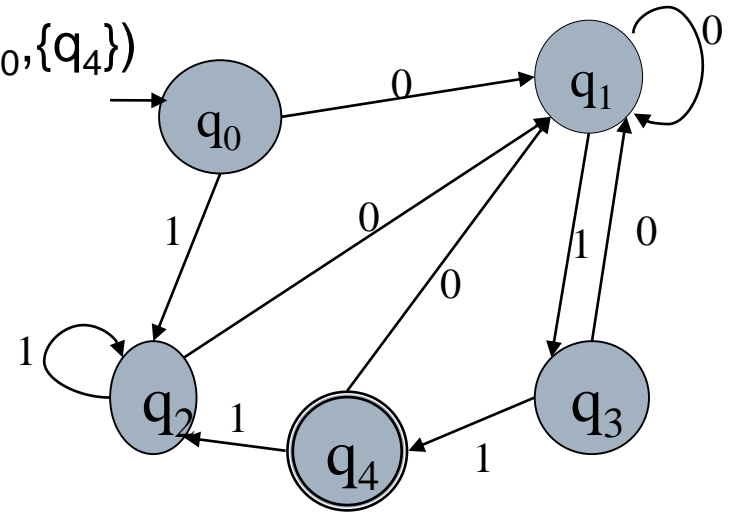
$$q_{0m} = c_0 \text{ si } q_0 \in c_0 \text{ y } c_0 \in Q_m$$

$$F_m = \{c_i / \exists p \in c_i \text{ y } p \in F\}$$

## 2.3. Minimización de Autómatas finitos por el conjunto cociente. Ejemplo

□ Ejemplo:  $M = (\{0,1\}, \{q_0, q_1, q_2, q_3, q_4\}, f, q_0, \{q_4\})$

f	0	1
$\rightarrow q_0$	$q_1$	$q_2$
$q_1$	$q_1$	$q_3$
$q_2$	$q_1$	$q_2$
$q_3$	$q_1$	$q_4$
$*q_4$	$q_1$	$q_2$



## 2.3. Minimización de Autómatas finitos por el conjunto cociente. Ejemplo

### □ Conjunto cociente

1. Conjunto inicial  $Q/E_1 = (\{q_0, q_1, q_2, q_3\}, \{q_4\})$

2.  $Q/E_i$

2.1.  $Q/E_2 = (\{q_0, q_1, q_2\}, \{q_3\}, \{q_4\})$

3.  $Q/E_2 \neq Q/E_1$  paso 2

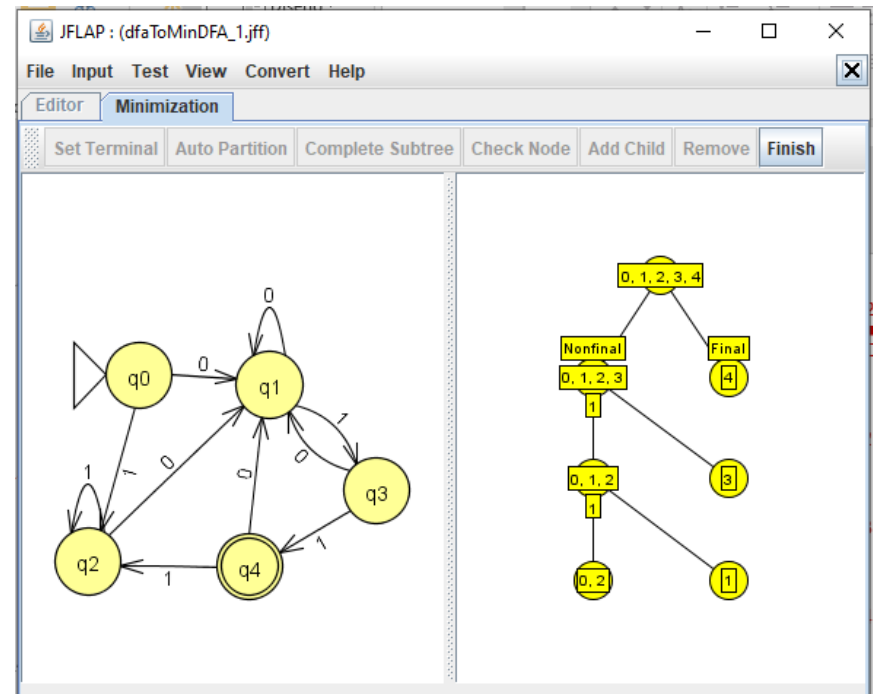
2.2.  $Q/E_3 = (\{q_0, q_2\}, \{q_1\}, \{q_3\}, \{q_4\})$

4.  $Q/E_3 \neq Q/E_2$  paso 2

2.3.  $Q/E_4 = (\{q_0, q_2\}, \{q_1\}, \{q_3\}, \{q_4\})$

5.  $Q/E_4 = Q/E_3 = Q/E$

f	0	1
$\rightarrow q_0$	$q_1$	$q_2$
$q_1$	$q_1$	$q_3$
$q_2$	$q_1$	$q_2$
$q_3$	$q_1$	$q_4$
$*q_4$	$q_1$	$q_2$



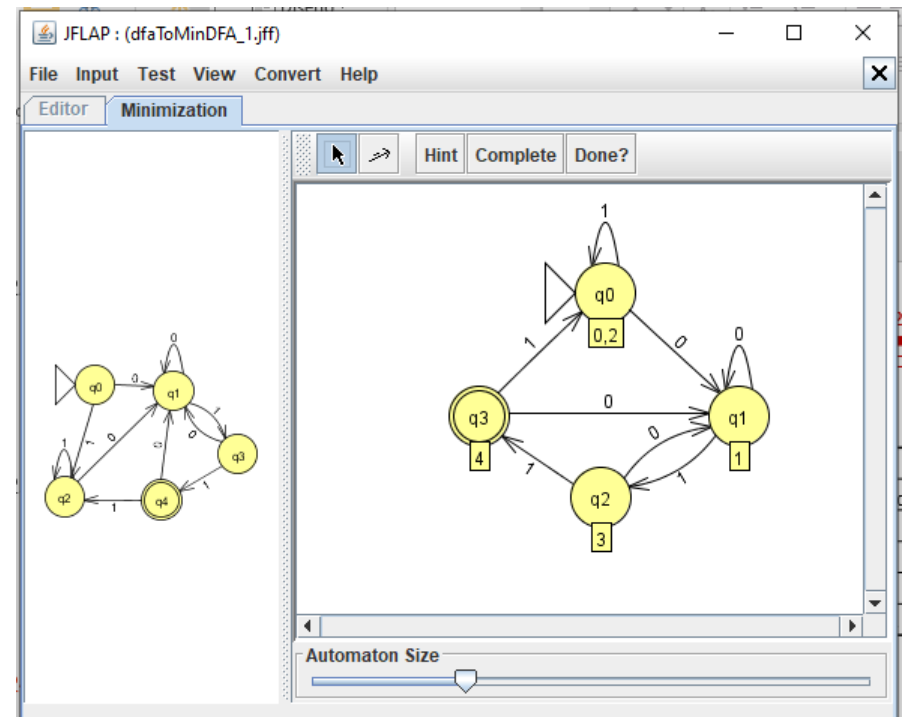
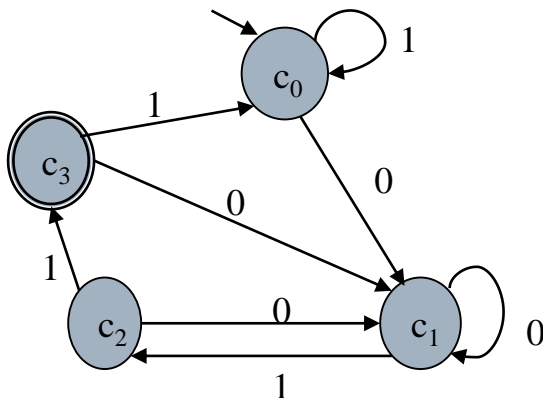
## 2.3. Minimización de Autómatas finitos por el conjunto cociente. Ejemplo

□ Autómata mínimo equivalente:  $M_m = (\{0,1\}, \{c_0, c_1, c_2, c_3\}, f_m, q_{om})$

$$Q_m = (c_0 = \{q_0, q_2\}, c_1 = \{q_1\}, c_2 = \{q_3\}, c_3 = \{q_4\})$$

f	0	1
$\rightarrow q_0$	$q_1$	$q_2$
$q_1$	$q_1$	$q_3$
$q_2$	$q_1$	$q_2$
$q_3$	$q_1$	$q_4$
$*q_4$	$q_1$	$q_2$

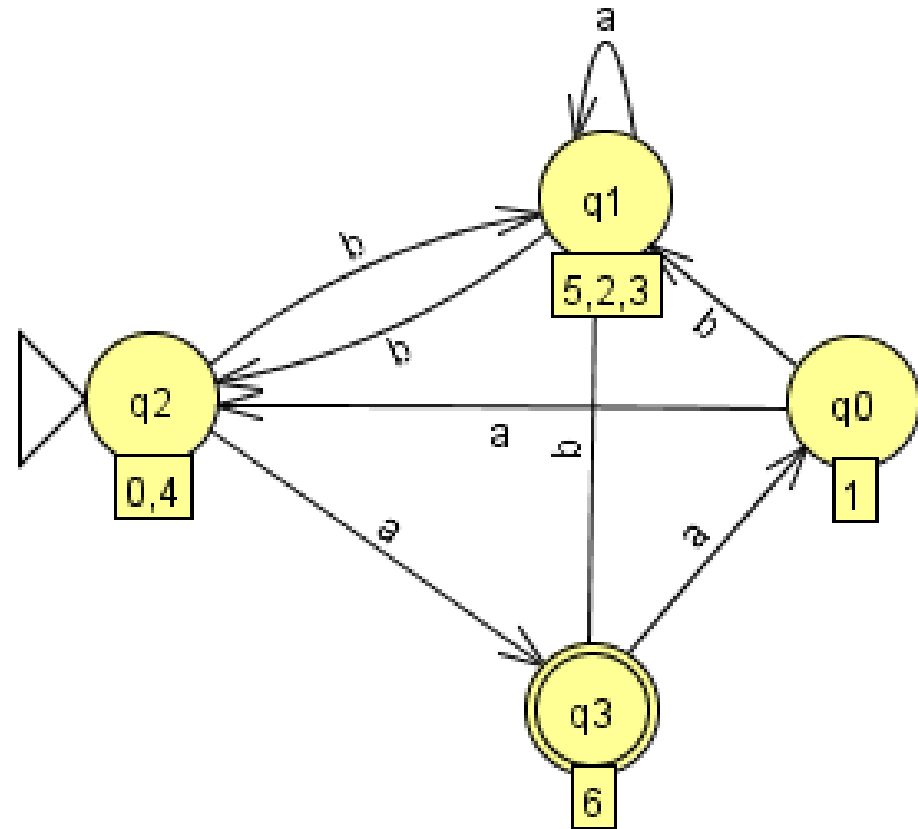
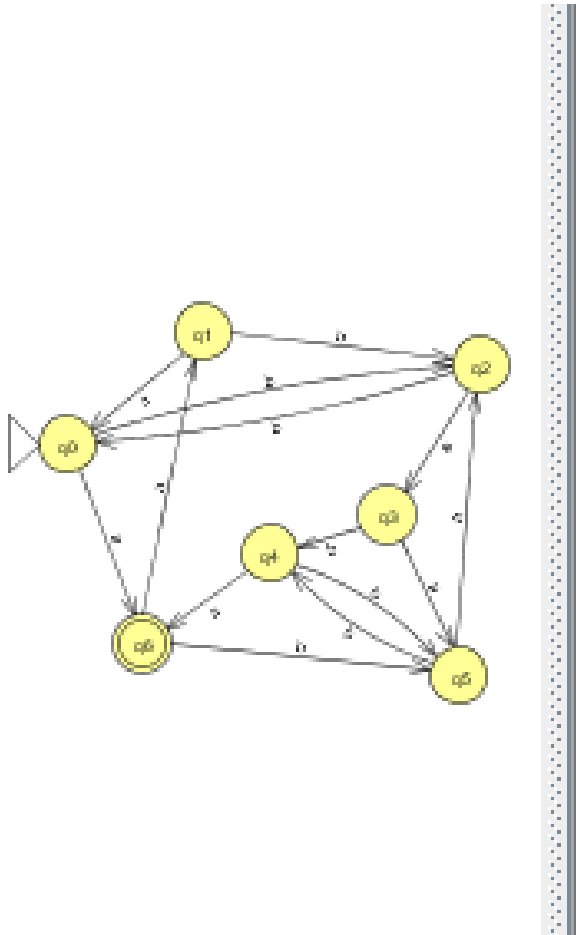
$f_m$	0	1
$\rightarrow c_0$	$c_1$	$c_0$
$c_1$	$c_1$	$c_2$
$c_2$	$c_1$	$c_3$
$*c_3$	$c_1$	$c_0$





## 2.3. Minimización de Autómatas finitos por el conjunto cociente. Ejemplo\_2

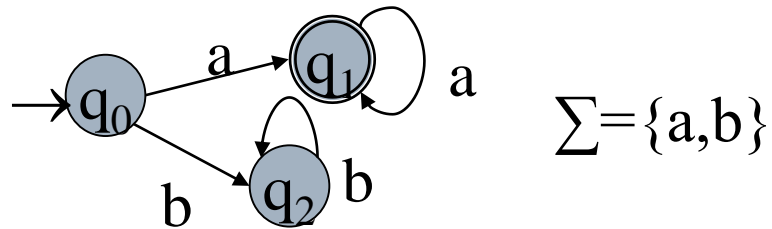
- Minimizar el siguiente AFD:



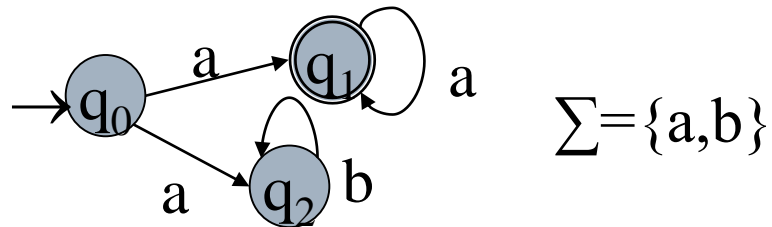
## 2.3. Autómatas Finitos No Deterministas. Introducción.

□ Un autómata finito es no determinista si:

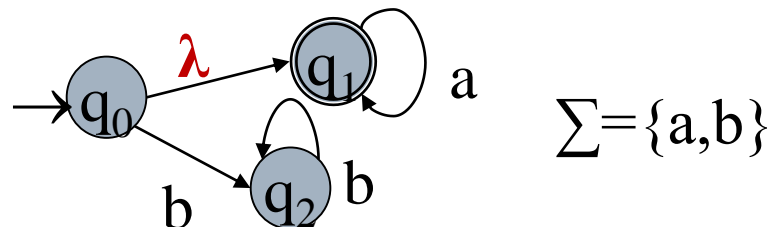
- No  $\exists f(q,a)$  para algún  $a \in \Sigma$  desde algún  $q \in Q$



- $\exists$  mas de una  $f(q,a)$  desde  $q \in Q$  con  $a \in \Sigma$



- $\exists f(q,\lambda)$



## 2.3. Definición AFND.

---

□ Un *autómata finito no determinista* (AFND) es un modelo matemático definido por la quintupla  $M = (Q, \Sigma, f, q_0, F)$  en el que:

- $Q$  es un conjunto finito llamado *conjunto de estados*.
- $\Sigma$  es un conjunto finito de símbolos, llamado *alfabeto de entrada*.
- $f$  es una aplicación llamada *función de transición* definida como:

$$f: Q \times (\Sigma \cup \{\lambda\}) \rightarrow P(Q)$$

donde  $P(Q)$  es el conjunto de las partes de  $Q$ , es decir, conjunto de todos los subconjuntos que se pueden formar con elementos de  $Q$

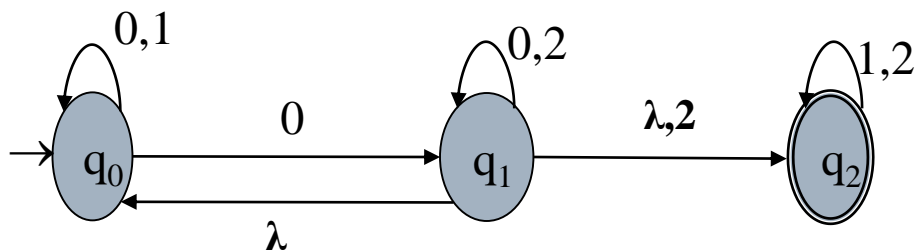
- $q_0$  es un elemento o estado de  $Q$ , llamado *estado inicial*.
- $F$  es un subconjunto de  $Q$ , llamado *conjunto de estados finales*.

## 2.3.. Representación.

### □ Tablas de transición.

	0	1	2	$\lambda$
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$	$\emptyset$	$\emptyset$
$q_1$	$\{q_1\}$	$\emptyset$	$\{q_1, q_2\}$	$\{q_0, q_2\}$
$^*q_2$	$\emptyset$	$\{q_2\}$	$\{q_2\}$	$\emptyset$

### □ Diagramas de transición.



## 2.3. función de transición. Extensión a palabras.

### □ Definición $\lambda$ -clausura:

- Se llama  **$\lambda$ -clausura** de **un estado** al conjunto de estados a los que puede evolucionar sin consumir ninguna entrada, lo denotaremos como  $CL(q)$  o  $\lambda$ -clausura( $q$ ).
- $CL(q)$  se define **recursivamente**:
  - El estado  $q$  pertenece a la  $\lambda$ -clausura de  $q$ ,  $q \in CL(q)$ .
  - Si el estado  $p \in CL(q)$  y hay una transición del estado  $p$  al estado  $r$  etiquetada con una transición nula ( $\lambda$ ), entonces  $r$  también está en  $CL(q)$ .
- Si  $P \subseteq Q$  se llama  $\lambda$ -clausura de un conjunto de estados  $P$  a:

$$CL(P) = \bigcup_{q \in P} CL(q)$$

### □ Algoritmo para el cálculo de $\lambda$ -clausura( $P$ )

**Procedimiento**  $\lambda$ -clausura( $P$ )

Viejos= $\emptyset$

Nuevos= $P$

**mientras** Viejos  $\neq$  Nuevos **hacer**

Nuevos = Viejos  $\cup$   $\{q \mid f(p_i, \lambda) = q, \forall p_i \in \text{Viejos}\}$

**fmientras**

$\lambda$ -clausura( $P$ ) = Nuevos

**fin\_procedimiento**

## 2.3.. función de transición. Extensión a palabras.

---

- Si  $M=(Q,\Sigma,f,q_0,F)$  es un AFND se define la **función de transición** asociada a **palabras** como la función

$f': Q \times \Sigma^* \rightarrow P(Q)$  dada por:

■  $f'(q_0,\lambda)=CL(q_0)$

■  $f'(P,\lambda)=P$

■  $f'(P,ax)=f'(\bigcup_{q \in P} CL(f(q,a)),x)$

donde  $P \subseteq Q$ ,  $x \in \Sigma^*$  y  $a \in \Sigma$

## 2.3. Lenguaje aceptado por un AFND.

---

- Una **cadena**  $x \in \Sigma^*$  es *aceptada* por un AFND  $M=(Q, \Sigma, f, q_0, F)$  si y solo si
$$f'(q_0, x) \cap F \neq \emptyset$$

En otro caso se dice que la cadena es *rechazada* por el autómata.

- Dado un AFND  $M=(Q, \Sigma, f, q_0, F)$  se llama *lenguaje aceptado* o *reconocido* por dicho autómata al conjunto de las palabras de  $\Sigma^*$  que acepta,

$$L(M) = \{ x \in \Sigma^* / f'(q_0, x) \cap F \neq \emptyset \}$$

## 2.3. Simulación algorítmica de un AFND

---

- **Entrada:** cadena de entrada **x** que termina con un carácter fin de cadena o fin de archivo (FDC).
- **Salida:** La respuesta **ACEPTADA** si el autómata reconoce **x** **NO ACEPTADA** en caso contrario
- **Método:** aplicar **f** al estado al cual hay una transición desde el estado **q** a un carácter de entrada **c**

### Función reconocer()

A=CL( $q_0$ )

c= leer\_carácter()

Mientras c != FDC

A=CL ( $\bigcup_{q \in P} f(q,c)$ )

c= leer\_carácter()

fmientras

Si  $A \cap F \neq \emptyset$  entonces

Devolver(ACEPTADA)

Sino

Devolver(NO ACEPTADA)

fsi



## 2.4. Equivalencia entre AFND y AFD

---

- A partir de un AFND con estados  $q_0, \dots, q_m$ , construiremos un AFD equivalente (que acepta el mismo lenguaje) con estados  $Q_0, \dots, Q_n$  donde  $q_0, Q_0$  son los estados iniciales.
- **Premisas:**
  - Se definió inductivamente el conjunto de estados  $C = \lambda\text{-clausura}(s)$ :
    - $s \in C$ .
    - si  $t \in C$  y  $\exists$  una transición vacía de  $t$  a  $u$  entonces  $u \in C$ .
  - Se define el conjunto de estados  $G = f(s, a)$  :
    - si  $\exists$  una transición etiquetada con  $a$  entre  $s$  y  $t$  entonces  $t \in G$ .
  - Estas definiciones se generalizan para conjuntos de estados
$$S = \{ \dots t_i \dots \} :$$
    - Si  $u \in \lambda\text{-clausura}(t_i)$  entonces  $u \in \lambda\text{-clausura}(S)$
    - Si  $u \in f(t_i, a)$  entonces  $u \in f(S, a)$ .

$(f(G, a))$  son los estados  $s$  alcanzables con  $a$  desde algún  $q \in G$

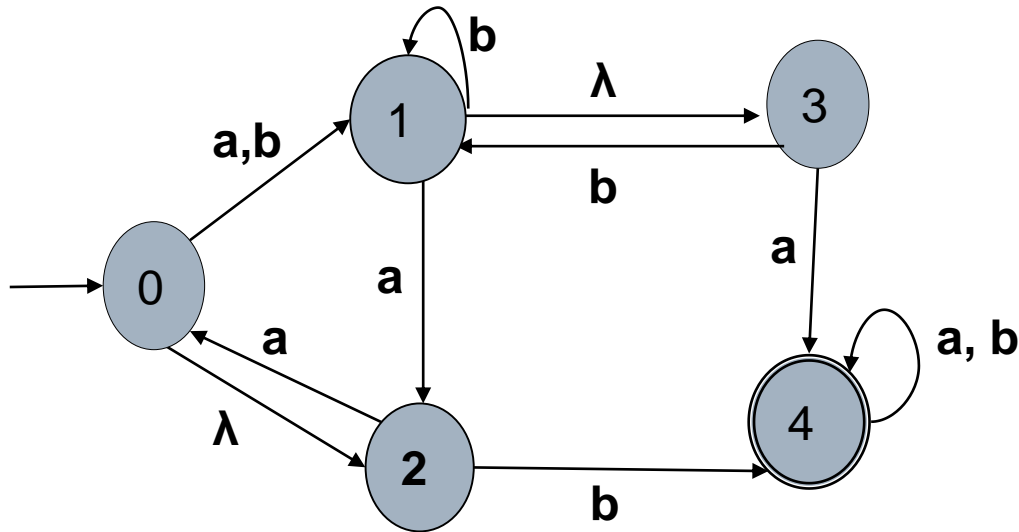
## 2.4. Equivalencia entre AFND y AFD

---

- Algoritmo de construcción de la tabla de transiciones del AFD:
  1. se crea una nueva tabla  $T[\text{estado}, \text{símbolo}]$ , inicialmente vacía.
  2. se calcula  $Q_0 = \lambda\text{-clausura}(q_0)$
  3. se crea una entrada en  $T$  para  $Q_0$ .
  4. para cada casilla vacía  $T[Q, a]$ :
    1. se asigna  $T[Q, a] = \lambda\text{-clausura}(f(Q, a))$
    2. si no existe una entrada en  $T$  para el estado  $T[Q, a]$ , se crea la entrada.
  5. se repite 4 mientras existan casillas vacías.

## 2.4. Equivalencia entre AFND y AFD. Ejemplo.

□ Ejemplo:



	a	b	λ
→0	1	1	2
1	2	1	3
2	0	4	∅
3	4	1	∅
*4	4	4	∅

## 2.4. Equivalencia entre AFND y AFD. Ejemplo.

1. se crea una nueva tabla  $T[\text{estado}, \text{símbolo}]$ , inicialmente vacía.
2. se calcula  $Q_0 = \lambda\text{-clausura}(0) = \{0, 2\}$
3. se crea una entrada en  $T$  para  $Q_0$ .

	a	b
$Q_0$		

4. para cada casilla vacía  $T[Q, a]$ :
  1. se asigna  $T[Q, a] = \lambda\text{-clausura}(f(Q, a))$   
 $\lambda\text{-clausura}(f(Q_0 = \{0, 2\}, a)) = \lambda\text{-clausura}(0, 1) = \{0, 1, 2, 3\} = Q_1$
  2. si no existe una entrada en  $T$  para el estado  $T[Q, a]$ , se crea la entrada.

	a	b
$Q_0$	$Q_1$	
$Q_1$		

5. se repite 4 mientras existan casillas vacías.

	a	b	$\lambda$
0	1	1	2
1	2	1	3
2	0	4	$\emptyset$
3	4	1	$\emptyset$
*4	4	4	$\emptyset$

## 2.4. Equivalencia entre AFND y AFD. Ejemplo

5.  $\lambda\text{-clausura}(f(Q_0=\{0,2\},b))=\lambda\text{-clausura}(1,4)=\{1,3,4\}=Q_2$

6.  $\lambda\text{-clausura}(f(Q_1=\{0,1,2,3\},a))=$   
 $\text{clausura}(0,1,2,4)=\{0,1,2,3,4\}=Q_3$

7.  $\lambda\text{-clausura}(f(Q_1=\{0,1,2,3\},b))=$   
 $\text{clausura}(1,4)=\{1,3,4\}=Q_2$

8.  $\lambda\text{-clausura}(f(Q_2=\{1,3,4\},a))=\lambda\text{-clausura}(2,4)=\{2,4\}=Q_4$

9.  $\lambda\text{-clausura}(f(Q_2=\{1,3,4\},b))=\lambda\text{-clausura}(1,4)=\{1,3,4\}=Q_2$

10.  $\lambda\text{-clausura}(f(Q_3=\{0,1,2,3,4\},a))=\lambda\text{-clausura}(0,1,2,4)=\{0,1,2,3,4\}=Q_3$

11.  $\lambda\text{-clausura}(f(Q_3=\{0,1,2,3,4\},b))=\lambda\text{-clausura}(1,4)=\{1,3,4\}=Q_2$

12.  $\lambda\text{-clausura}(f(Q_4=\{2,4\},a))=\lambda\text{-clausura}(0,4)=\{0,2,4\}=Q_5$

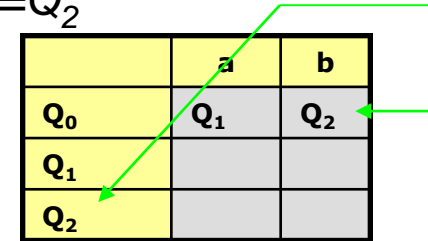
13.  $\lambda\text{-clausura}(f(Q_4=\{2,4\},b))=\lambda\text{-clausura}(4)=\{4\}=Q_6$

14.  $\lambda\text{-clausura}(f(Q_5=\{0,2,4\},a))=\lambda\text{-clausura}(0,2,4)=\{0,2,4\}=Q_5$

15.  $\lambda\text{-clausura}(f(Q_5=\{0,2,4\},b))=\lambda\text{-clausura}(1,4)=\{1,3,4\}=Q_2$

16.  $\lambda\text{-clausura}(f(Q_6=\{4\},a))=\lambda\text{-clausura}(4)=\{4\}=Q_6$

17.  $\lambda\text{-clausura}(f(Q_6=\{4\},b))=\lambda\text{-clausura}(4)=\{4\}=Q_6$



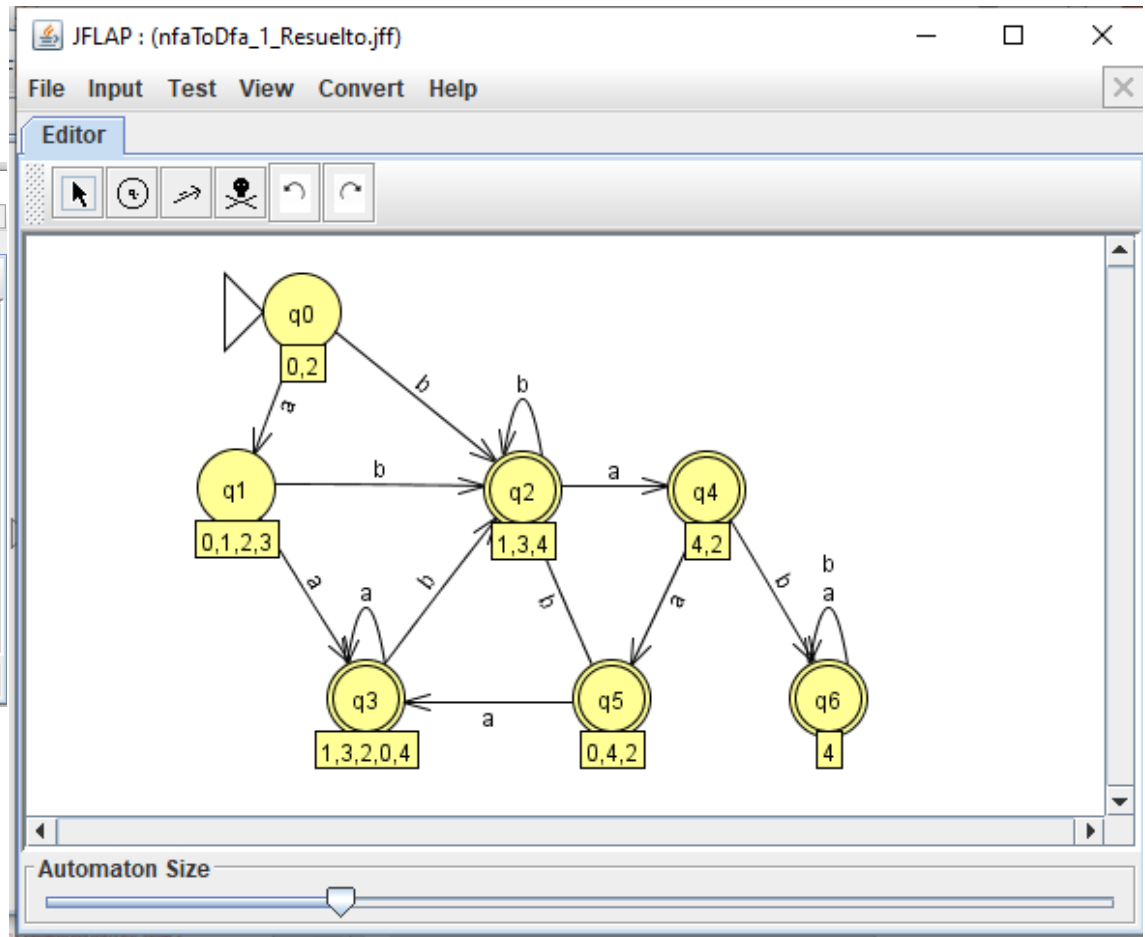
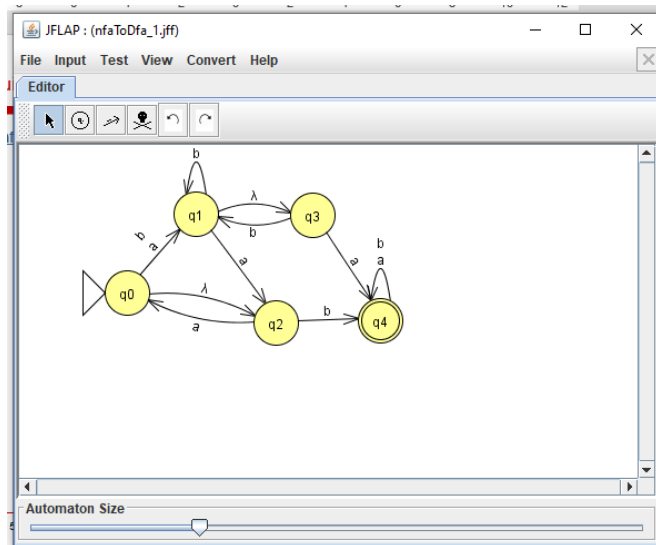
	a	b
Q <sub>0</sub>	Q <sub>1</sub>	Q <sub>2</sub>
Q <sub>1</sub>		
Q <sub>2</sub>		

## 2.4. Equivalencia entre AFND y AFD. Ejemplo

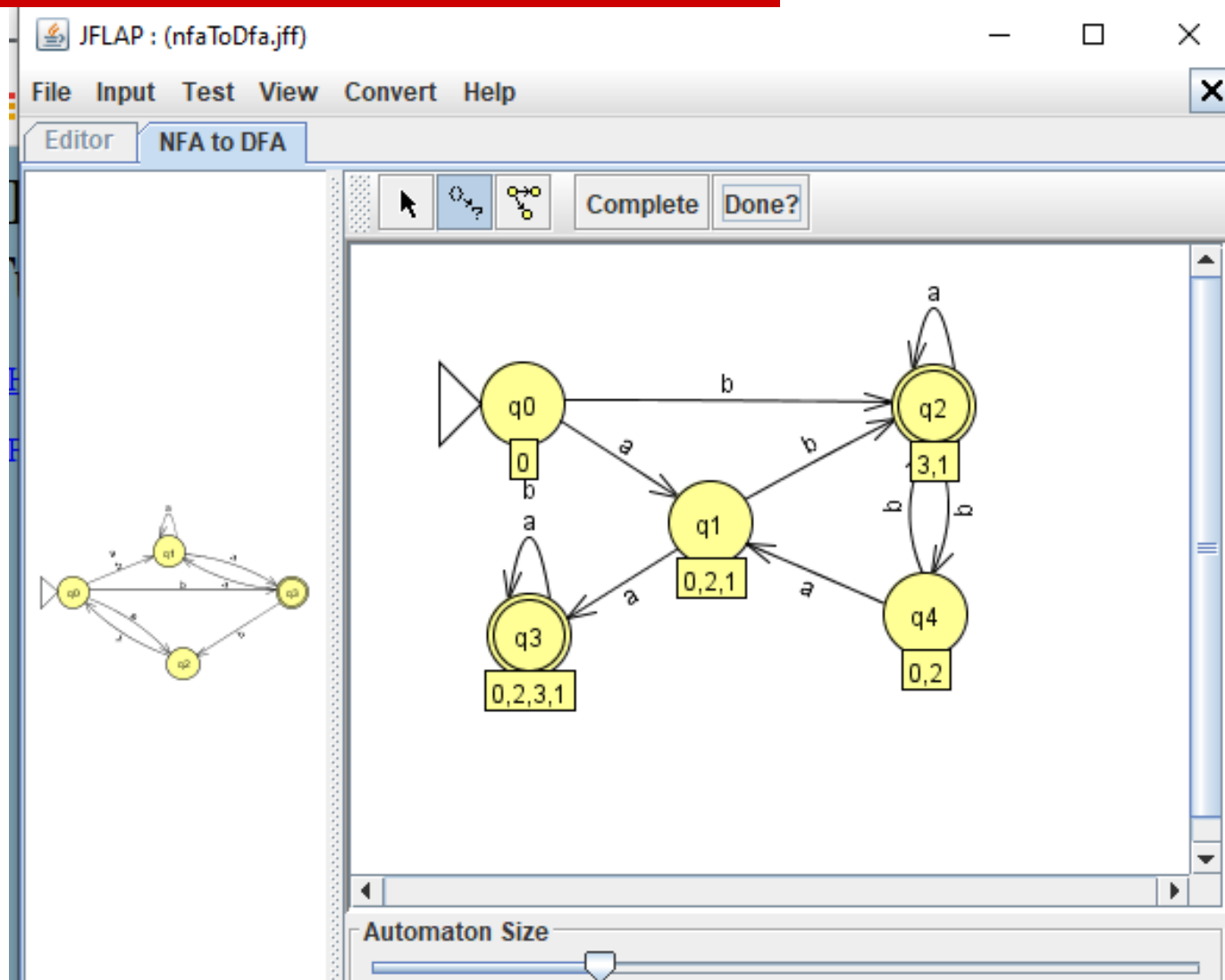
□ El AFD equivalente es:

	a	b
$\rightarrow Q_0$	$Q_1$	$Q_2$
$Q_1$	$Q_3$	$Q_2$
$*Q_2$	$Q_4$	$Q_2$
$*Q_3$	$Q_3$	$Q_2$
$*Q_4$	$Q_5$	$Q_6$
$*Q_5$	$Q_5$	$Q_2$
$*Q_6$	$Q_6$	$Q_6$

## 2.4. Equivalencia entre AFND y AFD. Ejemplo\_1.



## 2.4. Equivalencia entre AFND y AFD. Ejemplo\_2.





## 2.5. Autómatas finitos y gramáticas regulares.

---

### 2.5.1. Gramática regular asociada a un AFD

- Si  $L$  es aceptado por un AFD entonces  $L$  puede generarse mediante una gramática regular.
- ① Sea  $M=(Q, \Sigma, f, q_0, F)$ , la gramática regular equivalente es aquella definida como  $G=(Q, \Sigma, P, q_0)$ , donde  $P$  viene dado por:
  - Si  $f(q,a)=p$  entonces añadir a  $P$  la producción  $q \rightarrow ap$ .
  - Si  $f(q,a)=p$  y  $p \in F$  entonces añadir a  $P$  la producción  $q \rightarrow a$ .
  - Si  $q_0 \in F$  entonces añadir a  $P$  la producción  $q_0 \rightarrow \lambda$

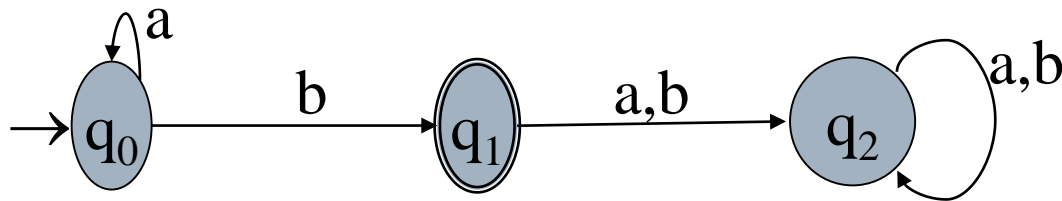
## 2.5. Autómatas finitos y gramáticas regulares.

### □ Ejemplo: De AFD a Gramática regular:

Dado un lenguaje regular  $L$  reconocido por un AFD  $M=(Q, \Sigma, f, q_0, F)$ , se puede obtener una gramática regular  $G=(\Sigma_N, \Sigma_T, S, P)$ , que genere el mismo lenguaje de la siguiente forma:

- $\Sigma_N = Q$
- $\Sigma_T = \Sigma$
- $S = q_0$
- $P = \{(q, ap)/f(q, a) = p\} \cup \{(q_0, \lambda)/p, q_0 \in F\}$

□ Ej.- Considera el siguiente **DFA** que acepta el lenguaje  $a^*b$ :



□ La gramática será:

- $q_0 \rightarrow aq_0|bq_1|b$
- $q_1 \rightarrow aq_2|bq_2$
- $q_2 \rightarrow aq_2|bq_2$

□

[illegible]

## 2.5. Autómatas finitos y gramáticas regulares.

---

### 2.5.2. AFD asociado a una Gramática regular

□ Si  $L$  es un lenguaje generado por una gramática regular, entonces existe un AFD que lo reconoce:

① Sea la gramática regular  $G=(\Sigma_N, \Sigma_T, P, S)$ , se construye el AFND- $\lambda$  equivalente como  $M=(\Sigma_N \cup \{F\}, \Sigma_T, f, S, \{F\})$ , donde  $f$  viene dado por:

- Si  $A \rightarrow aB$  entonces  $f(A, a) = B$
- Si  $A \rightarrow a$  entonces  $f(A, a) = F$
- Si  $S \rightarrow \lambda$  entonces  $f(S, \lambda) = F$
- $F$  es un “nuevo” símbolo no terminal.
- Convertir el AFND-  $\lambda$  obtenido a un AFD.

## 2.5. Autómatas finitos y gramáticas regulares.

### □ Ejemplo: De Gramática regular a AFN:

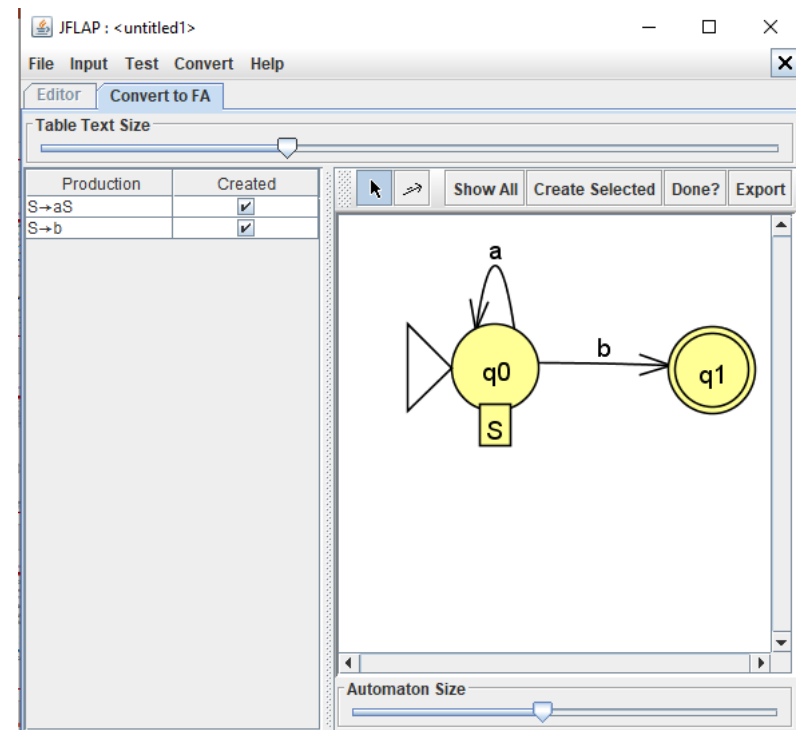
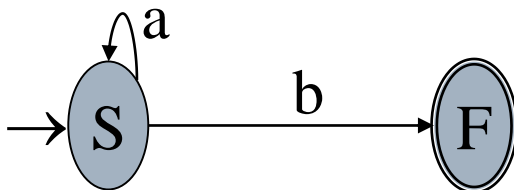
Dado un lenguaje regular  $L$  generado por una gramática regular  $G=(\Sigma_N, \Sigma_T, S, P)$ , se puede obtener un AFD  $M=(Q, \Sigma, f, q_0, F)$  que reconozca el mismo lenguaje de la siguiente forma:

- $Q = \Sigma_N \cup \{F\}$ ,
- $\Sigma = \Sigma_T$
- $q_0 = S$

### □ Ej.- Considera la siguiente gramática:

- $S \rightarrow aS \mid b$  que acepta el lenguaje  $a^*b$ :

### □ El AFD equivalente:



## 2.6. Expresiones regulares.

---

- ❑ Las expresiones regulares son una notación especial que se utiliza habitualmente para describir los lenguajes de tipo regular.
- ❑ La notación más utilizada para especificar patrones son las **expresiones regulares**, sirven como nombres para conjuntos de cadenas.
- ❑ Los usos más habituales de las expresiones regulares son en la creación de analizadores léxicos para compiladores, en la búsqueda de patrones dentro de los editores de texto, en la descripción de redes neuronales y circuitos electrónicos, etcétera.
- ❑ En las expresiones regulares pueden aparecer símbolos de dos tipos:
  - Símbolos base: son los del alfabeto del lenguaje que queremos describir,  $\lambda$  para describir la palabra vacía y  $\emptyset$  para describir un lenguaje vacío.
  - Símbolos de operadores: '+' o unión, '.' o concatenación y '\*' para la clausura. En ocasiones se utiliza para la unión el símbolo '|' y el punto se omite en la concatenación de expresiones regulares.

## 2.6. Lenguaje de las expresiones regulares.

- El lenguaje de las expresiones regulares se construye de forma inductiva a partir de símbolos para expresiones regulares elementales y operadores.

- **Expresiones regulares**

Dado un alfabeto  $\Sigma$  y los símbolos:  $\emptyset$  (lenguaje vacío),  $\lambda$  (palabra vacía),  $\cdot$  (concatenación),  $+$  (unión),  $*$  (clausura), se cumple:

- Los símbolos  $\emptyset$  y  $\lambda$  son expresiones regulares.
- Cualquier símbolo  $a \in \Sigma$  es una expresión regular.
- Si  $u$  y  $v$  son expresiones regulares, entonces,  $u+v$ ,  $uv$ ,  $u^*$  y  $v^*$  son expresiones regulares.

Sólo son expresiones regulares las que se pueden obtener aplicando un número finito de veces las reglas anteriores.

- Se establece la siguiente prioridad en las operaciones:

1. Paréntesis  $()$
2. Clausura  $*$
3. Concatenación  $\cdot$
4. Unión  $+$

- A cada expresión regular le corresponde un **lenguaje regular**:

Si  $\alpha = \emptyset$ ,  $L(\alpha) = \emptyset$

Si  $\alpha = \lambda$ ,  $L(\alpha) = \{\lambda\}$ .

Si  $\alpha = a$ ,  $a \in \Sigma$ ,  $L(\alpha) = \{a\}$

Si  $\alpha$  y  $\beta$  son dos expresiones regulares entonces:

$L(\alpha + \beta) = L(\alpha) \cup L(\beta)$

$L(\alpha\beta) = L(\alpha)L(\beta)$

$L(\alpha^*) = L(\alpha)^*$

## 2.6. Lenguaje de las expresiones regulares.

---

### ■ Definiciones regulares

- Son nombres dados a las expresiones regulares.
- Una definición regular es una secuencia de definiciones de la forma:

$d1 \rightarrow r1$

$d2 \rightarrow r2$

.....

$dn \rightarrow rn$

Donde cada  $d_i$  es un nombre distinto y cada  $r_i$  una expresión regular sobre los símbolos del alfabeto  $\Sigma \cup \{d1, d2, \dots, dn\}$

### ■ Ejemplo: definición regular para identificadores en Pascal:

**letra**  $\rightarrow A|B|\dots|Z|a|b|\dots|z$

**digito**  $\rightarrow 0|1|\dots|9$

**id**  $\rightarrow \text{letra}(\text{letra}|\text{digito})^*$

### ■ Abreviaturas en la notación

- Uno o más casos: operador unitario posfijo +
- Cero o un caso: operador unitario posfijo ?
- Clases de caracteres: [abc]

### ■ Ejemplos:

**digitos**  $\rightarrow \text{digito}^+$

**Exponente\_opcional**  $\rightarrow (E(+|-)?\text{digitos})?$



## 2.6. Lenguaje de las expresiones regulares.

---

- Ejemplo, sobre el alfabeto  $\Sigma = \{a, b\}$ , podemos definir las siguientes expresiones regulares:

Expresión Regular	Lenguaje
$ab$	$\{ab\}$
$a + \emptyset$	$\{a\}$
$(a+b)^*a$	$\Sigma^*\{a\}$
$a + \lambda$	$\{\lambda, a\}$
$b^*ab^*$	$\{b^nab^m/n, m \in \mathbb{N}\}$
$a^*$	$\{\lambda, a, aa, aaa, \dots\}$

## 2.6. Álgebra de las expresiones regulares.

---

### □ Propiedades de la Unión:

- asociativo:  $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
- conmutativo:  $\alpha + \beta = \beta + \alpha$
- elemento neutro la expresión vacía:  $\emptyset + \alpha = \alpha + \emptyset = \alpha$
- idempotente:  $\alpha + \alpha = \alpha$

### □ Propiedades de la concatenación:

- asociativo:  $\alpha (\beta \gamma) = (\alpha \beta) \gamma$
- no es conmutativo:  $\alpha \beta \neq \beta \alpha$
- elemento neutro la expresión lambda:  $\lambda \alpha = \alpha \lambda = \alpha$
- Tiene como elemento anulador la expresión vacía:  $\emptyset \alpha = \alpha \emptyset = \emptyset$
- distributivo respecto al operador de unión:  $\alpha (\beta + \gamma) = \alpha \beta + \alpha \gamma$

### □ Propiedades de la clausura:

- $\lambda^* = \lambda$
- $\emptyset^* = \lambda$
- $\alpha^* = \lambda + \alpha \alpha^*$
- $\alpha \alpha^* = \alpha^* \alpha$
- $(\alpha^* + \beta^*)^* = (\alpha + \beta)^*$
- $(\alpha + \beta)^* = (\alpha^* \beta^*)^*$

## 2.6. Teorema fundamental.

---

- El Teorema servirá para describir mediante una expresión regular el lenguaje reconocido por un autómeta.
- El objetivo del teorema fundamental es resolver sistemas de ecuaciones en el dominio de las expresiones regulares que tienen la forma siguiente:

$$X = \alpha X + \beta$$

- El teorema se divide en dos partes que nos permiten obtener la solución a esta ecuación en los casos en que  $\lambda \in L(\alpha)$  y aquellos en los que  $\lambda \notin L(\alpha)$ , respectivamente.

- **Regla General:**

- **Teorema.** Sean  $\alpha$  y  $\beta$  dos expresiones regulares de forma que  $\lambda \in L(\alpha)$ . En estas condiciones, la solución a la ecuación  $X = \alpha X + \beta$  es de la forma

$$X = \alpha^*(\beta + \gamma)$$
 en donde  $\gamma$  representa cualquier expresión regular.

- **Regla de Inferencia:**

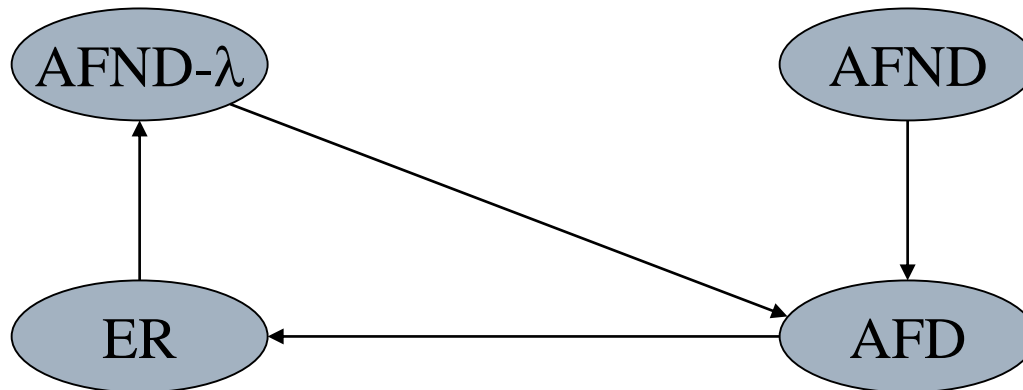
- **Teorema.** Sean  $\alpha$  y  $\beta$  dos expresiones regulares de forma que  $\lambda \notin L(\alpha)$ . En estas condiciones,

$$X = \alpha X + \beta \Leftrightarrow X = \alpha^* \beta$$

## 2.6. Autómatas finitos y expresiones regulares.

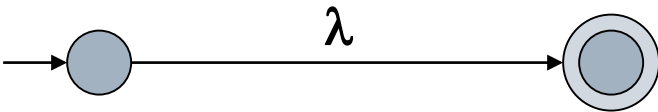

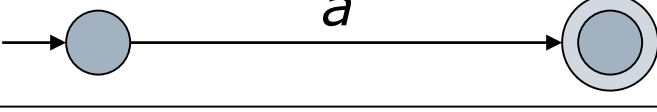
---

- ❑ Todo lenguaje definido por un autómata finito (AFD, AFND) es también definido por una expresión regular.
- ❑ Todo lenguaje definido por una expresión regular es definido por un autómata finito.



## 2.6. AFN asociado a una expresión regular

- Cada lenguaje definido por una expresión regular es también definido por un autómata finito.
- La demostración es inductiva sobre el número de operadores de  $\alpha$  (+, . \*).
- 🔗 **Base.-** (cero operadores)
  - $\alpha$  puede ser  $\emptyset$ ,  $\lambda$ ,  $a$ , donde  $a \in \Sigma_T$

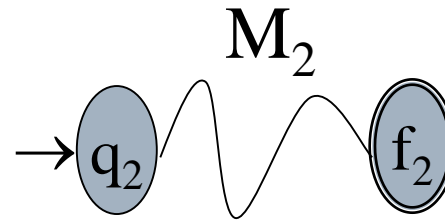
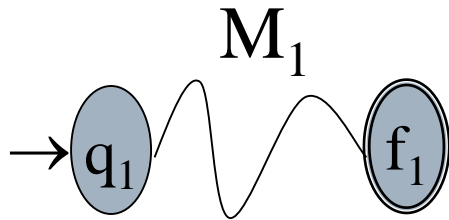
Expresión regular $\alpha$	Autómata
$\lambda$	
$\emptyset$	
$a$	

## 2.6.. AFN asociado a una expresión regular

---

📌 **Inducción.**- (uno o más operadores en  $\alpha$ )

- ❑ Suponemos que se cumple la hipótesis para expresiones regulares de menos de  $n$  operadores.
- ❑ Por hipótesis existen dos AF  $M_1$  y  $M_2$  tal que que acepta el mismo lenguaje  $L(M_1)=L(\alpha_1)$  y  $L(M_2)=L(\alpha_2)$  donde:



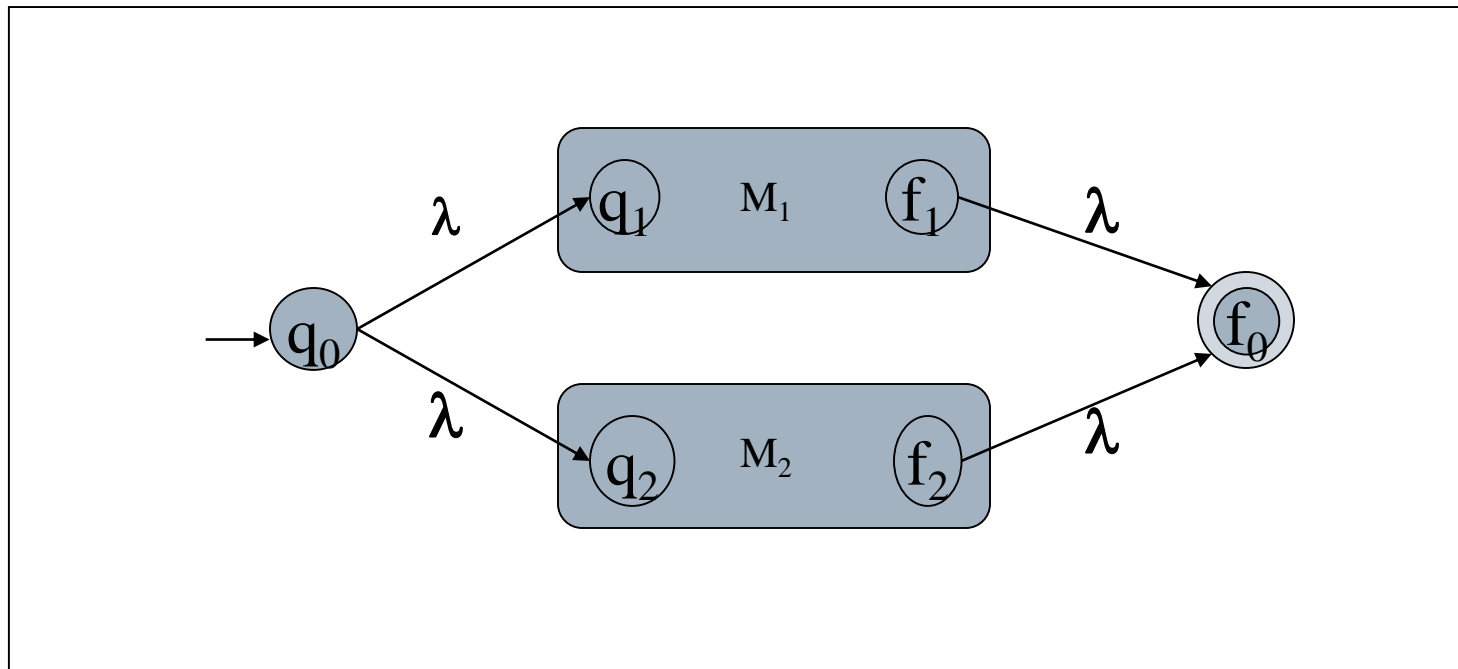
## 2.6. AFN asociado a una expresión regular.

---

- Suponemos que tenemos una expresión regular  $\alpha$  con  $n$  operadores.
- Vamos a construir el autómata  $M$  tal que  $L(M)=L(\alpha)$ .
- Distinguimos tres casos correspondientes a las tres formas posibles de expresar  $\alpha$  en función de otras expresiones regulares con menos de  $n$  operadores :
  1.  $\alpha = \alpha_1 + \alpha_2$  tal que  $op(\alpha_1), op(\alpha_2) < n$
  2.  $\alpha = \alpha_1 \cdot \alpha_2$  tal que  $op(\alpha_1), op(\alpha_2) < n$
  3.  $\alpha = (\alpha_1)^*$  tal que  $op(\alpha_1) = n-1$

## 2.6. AFN asociado a una expresión regular.

1.  $\alpha = \alpha_1 + \alpha_2$  tal que  $op(\alpha_1), op(\alpha_2) < n$
- A partir de  $M_1$  y  $M_2$  construimos otro autómata  $M$  (la unión), el autómata que acepta el mismo lenguaje es:

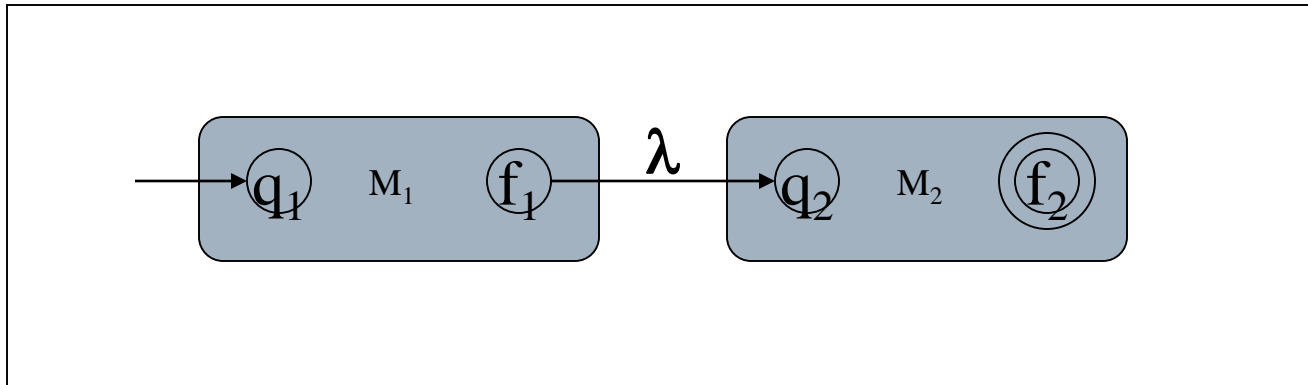




## 2.6. AFN asociado a una expresión regular.

---

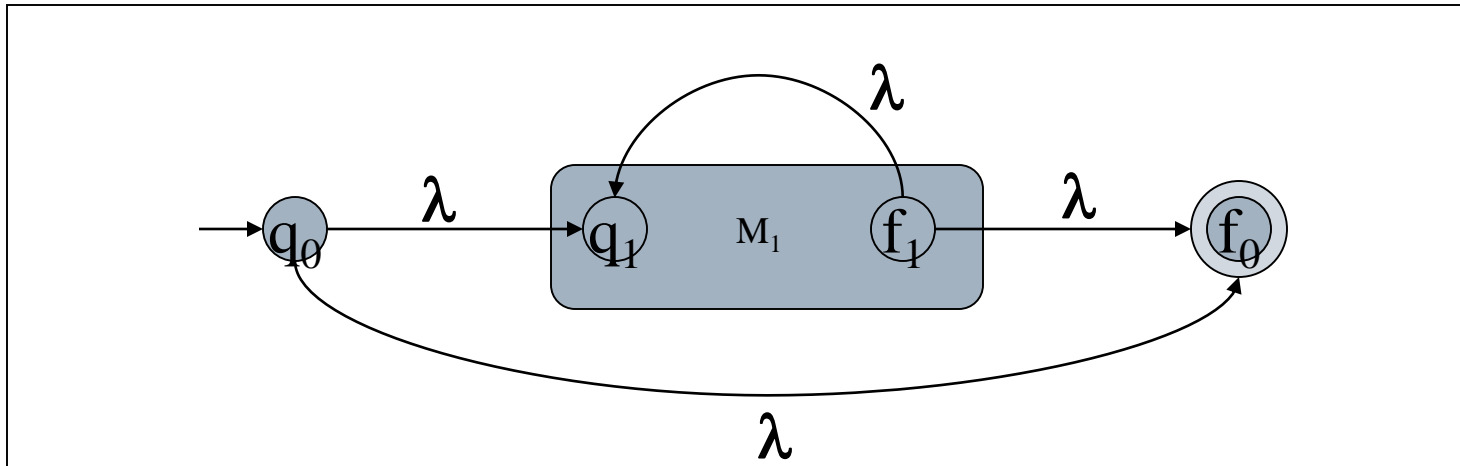
2.  $\alpha = \alpha_1 \cdot \alpha_2$  tal que  $op(\alpha_1), op(\alpha_2) < n$
- A partir de  $M_1$  y  $M_2$  construimos otro autómata  $M$  (la concatenación), el autómata que acepta el mismo lenguaje es:



## 2.6. AFN asociado a una expresión regular

3.  $\alpha = (\alpha_1)^*$  tal que  $op(\alpha_1) = n-1$

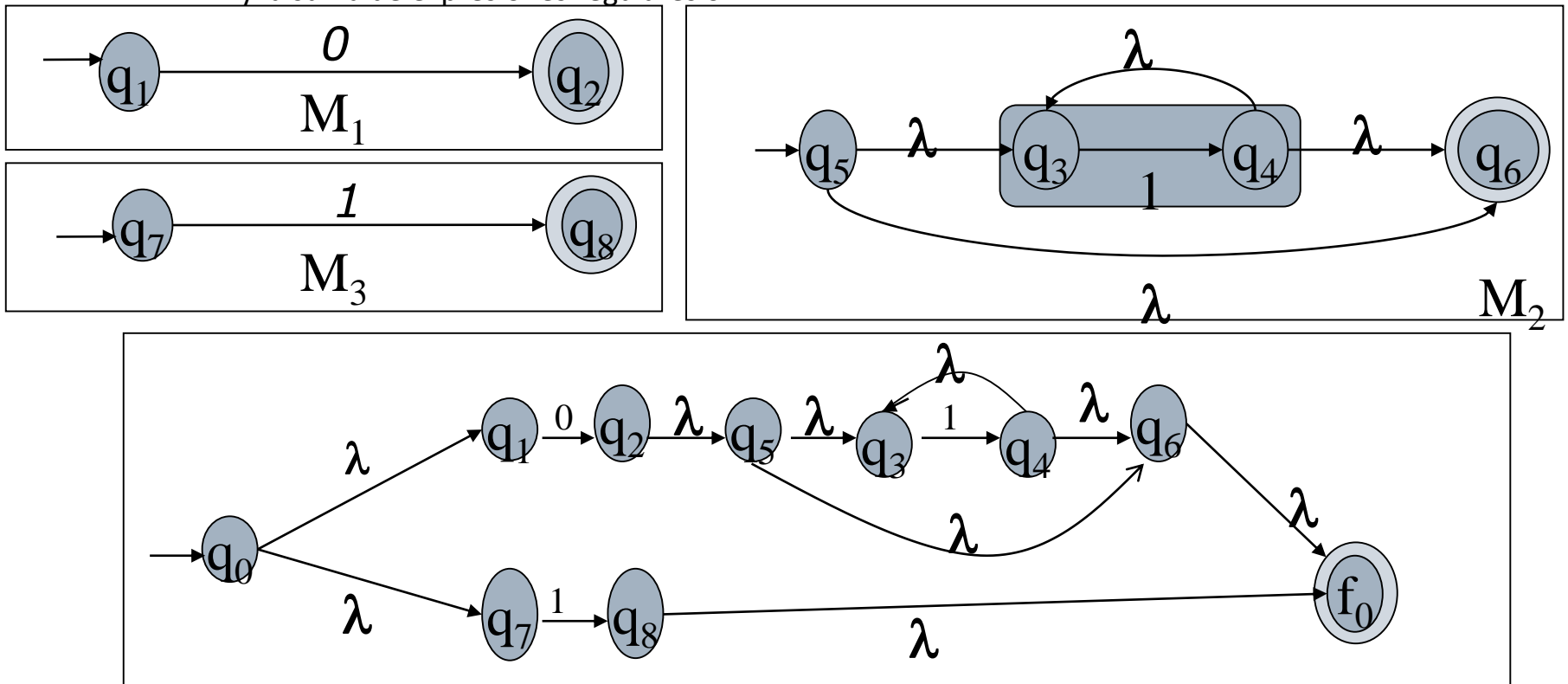
□ A partir de  $M_1$  construimos otro autómata  $M$  (la clausura), el autómata que acepta el mismo lenguaje es:



## 2.6. AFN asociado a una expresión regular

### □ Ejemplo: AF construido para la expresión regular $01^* + 1$ :

- $M_1$  representa el autómata para la expresión regular 0
- $M_2$  representa el autómata para la expresión regular  $1^*$
- $M_3$  representa el autómata para la expresión regular 1
- En el Autómata final se integran simultáneamente los autómatas para la concatenación ( $0$  con  $1^*$ ) y la suma de expresiones regulares  $01^* + 1$



## 2.6. Expresión regular asociada a un AFD

- Si  $L$  es un lenguaje aceptado por un autómata finito  $M$  entonces existe una expresión regular  $\alpha$  tal que  $L = L(M) = L(\alpha)$ .
  - Podemos suponer que el autómata finito  $M$  no tiene  $\lambda$ -transiciones. Si las tuviera, podemos encontrar autómata equivalente sin  $\lambda$ -transiciones
- Sea  $M = (Q, \Sigma, f, q_0, F)$ . A partir de su diagrama de transición podemos obtener un sistema de ecuaciones de expresiones regulares (ecuaciones características del autómata):
  - A cada nodo  $q_i$  le corresponde una ecuación y cada estado se puede considerar como una incógnita  $x_i$  de la ecuación.
  - La ecuación para el estado  $q_i$  tiene en el primer miembro el estado  $q_i$ ,  $x_i$ , y en el segundo miembro una suma de términos, de forma que por cada arco del diagrama de la forma  $q_i \rightarrow q_j$  tenemos un término  $ax_j$ .
  - Si el estado  $q_i$  es final, añadimos el término  $a$  al segundo miembro.
  - Si el estado  $q_0$  es final, añadimos el término  $\lambda$  al segundo miembro para  $x_0$ .
- Cada incógnita para  $q_i$ ,  $x_i$ , representa el conjunto de palabras que llevan de  $q_i$  a un estado final.
- Resolviendo el sistema de las ecuaciones tendremos soluciones de la forma  $x_i = \alpha_i$ , donde  $\alpha_i$  es una expresión regular sobre el alfabeto  $\Sigma$
- El lenguaje descrito por esta expresión regular es:
$$L(\alpha_i) = \{w \in \Sigma^* \mid (q_i, w) \vdash^* (q_F, \lambda), q_F \in F\} \quad (1)$$

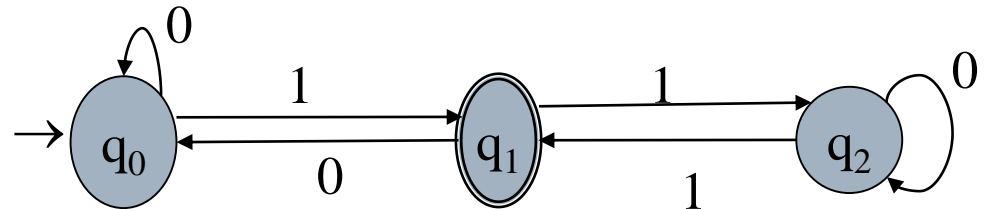
## 2.6. Expresión regular asociada a un AFD

---

- El método para obtener una expresión regular  $\alpha$  a partir de un AF es el siguiente:
  1. Obtener las ecuaciones características del autómata.
  2. Resolver el sistema de ecuaciones
  3.  $\alpha \leftarrow$  solución para el estado inicial.
- Para comprobar que es válido hay que probar que se cumple (1) para toda solución  $\mathbf{x}_i = \alpha_i$  del sistema de ecuaciones, y en particular la solución para el estado inicial es la expresión regular correspondiente al autómata
- No es necesario resolver todas las incógnitas, sólo necesitamos despejar la incógnita correspondiente al estado inicial  $\mathbf{x}_0$ .

## 2.6. Expresión regular asociada a un AFD

□ **Ejemplo.** Sea el AF:



□ Ecuaciones características:

1.  $x_0 = 0x_0 + 1x_1 + 1$

2.  $x_1 = 0x_0 + 1x_2$

3.  $x_2 = 0x_2 + 1x_1 + 1$

□ Resolvemos aplicando la regla de inferencia  $\mathbf{X = aX + \beta \Leftrightarrow X = a^* \beta}$

■ comenzando por la ecuación (3) :  $x_2 = 0^*(1x_1 + 1) = 0^*1x_1 + 0^*1$

■ Sustituyendo en (2):  $x_1 = 0x_0 + 10^*1x_1 + 10^*1 = (10^*1)^*(0x_0 + 10^*1) = (10^*1)^*0x_0 + (10^*1)^*10^*1$

■ Sustituyendo en (1):

$$\begin{aligned}
 \mathbf{x_0} &= 0x_0 + 1[(10^*1)^*0x_0 + (10^*1)^*10^*1] + 1 = 0x_0 + 1(10^*1)^*0x_0 + \\
 &1(10^*1)^*10^*1 + 1 = (0 + 1(10^*1)^*0)x_0 + 1(10^*1)^*10^*1 + 1 = \\
 &(0 + 1(10^*1)^*0)^*(1(10^*1)^*10^*1 + 1) = \\
 &= (0 + 1(10^*1)^*0)^*1[(10^*1)^*(10^*1) + \lambda] = \\
 &= \mathbf{(0 + 1(10^*1)^*0)^*1(10^*1)^*} \text{ expresión regular que describe el lenguaje } L(M).
 \end{aligned}$$

(aplicando las propiedades:  $(ab^*b + a) = a(b^*b + \lambda) = ab^*$ )