

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** (equivalencia) 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:

Gramática	Tipo de Lenguaje	Máquina / Dispositivo reconocedor
G0	Sin restricciones o con estructura de frase	Máquina de Turing
G1	Sensible al contexto	Autómata linealmente acotado
G2	Libre de contexto	Autómata con pila
G3	Regular	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

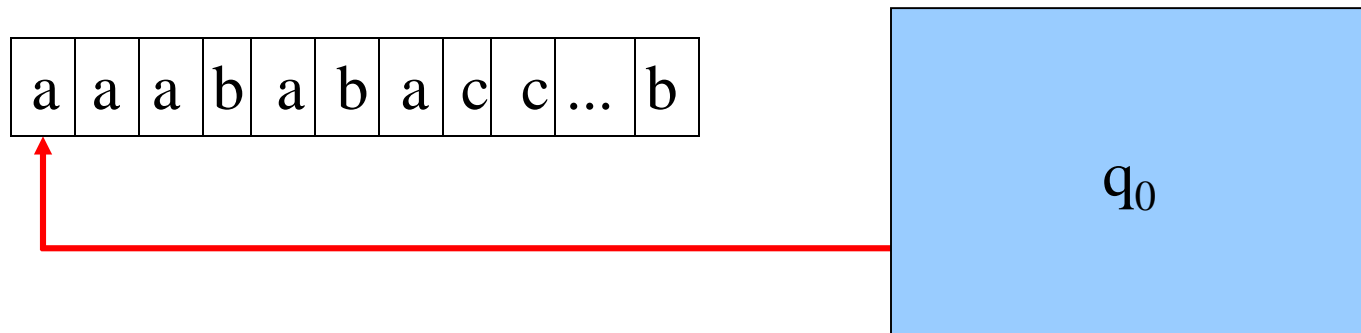
2.1. Autómatas Finitos. Introducción.

- ❑ 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.

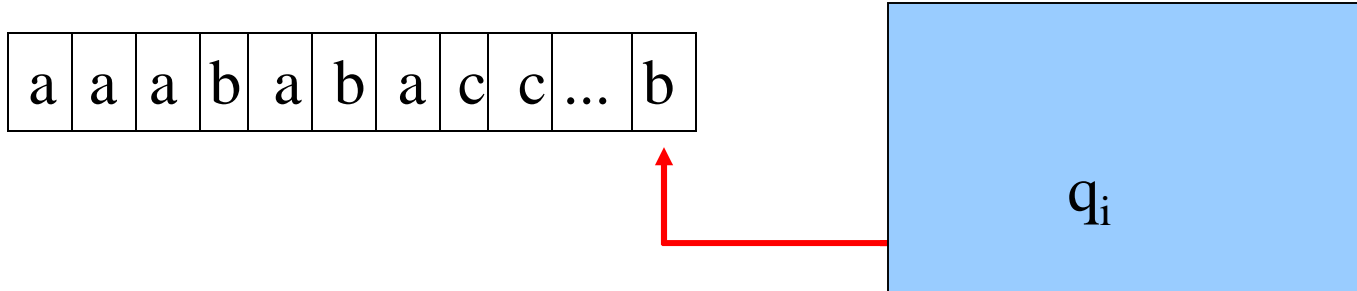
□ ¿Cómo procesa entradas un AFD?

- La entrada a un AF es un conjunto de símbolos tomados del alfabeto de entrada Σ , 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.

2.1. Autómatas Finitos. Simulación algorítmica

- **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 que se transita desde el estado **q** al consumir el símbolo de entrada **c** (hasta que no queden más símbolos por consumir)

función reconocer()

q = q₀

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

```
private boolean esFinal(String estado);  
  
public boolean reconocer(String cadena) {  
    char [] simbolo = cadena.toCharArray( );  
    String estado = 0 ; //El estado inicial es el 0  
    for ( int i=0; i<simbolo.length; i++ ) {  
        estado = transicion( estado, simbolo[i] );  
    }  
    return esFinal(estado);  
}
```

2.2. Autómatas finitos deterministas (AFD). Definición.

- Un **AFD** es una quintupla $M = (Q, \Sigma, f, q_0, F)$
- Q es un conjunto finito llamado *conjunto de estados*
 - Σ es un conjunto finito de símbolos, llamado *alfabeto de entrada*
 - f es una aplicación llamada *función de transición* que representa transiciones entre estados ante un determinado símbolo

$$f: Q \times \Sigma \rightarrow Q$$

- q_0 es un elemento de Q , llamado *estado inicial*.
- F es un subconjunto de Q ($F \subset Q$), llamado *conjunto de estados finales*
- Cada transición debe consumir un símbolo: *no existen transiciones λ*
- Condición de determinismo: *para cada estado y símbolo sólo existe una única transición* (no existen transiciones que partan del mismo estado con el mismo símbolo)

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 $*$

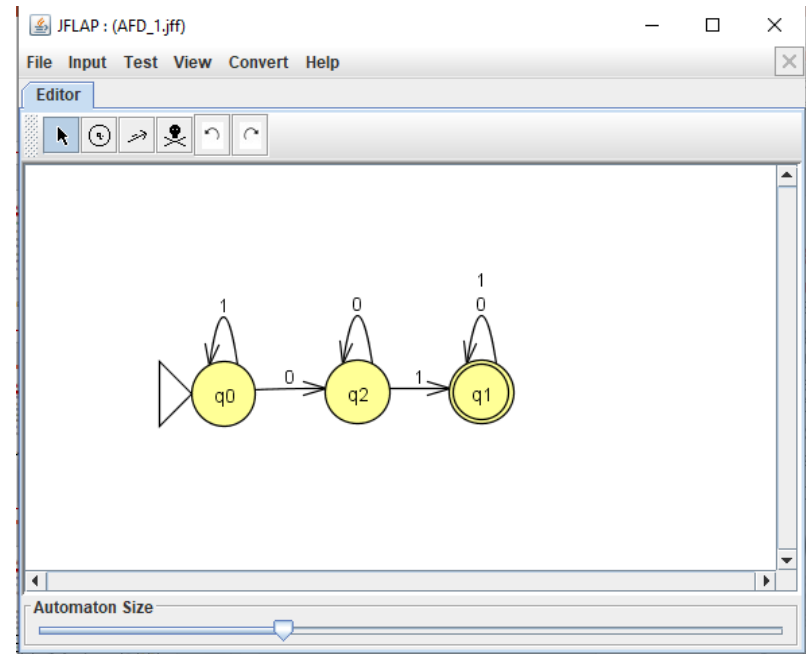
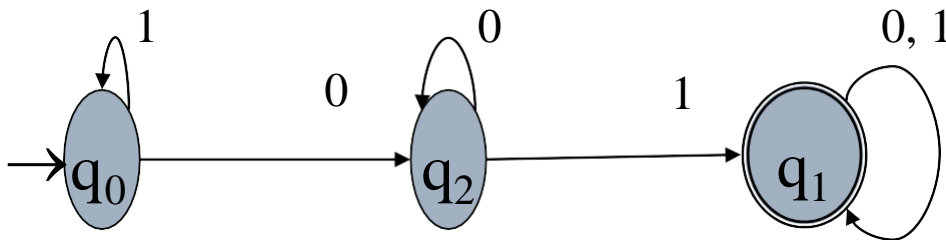
Ejemplo:

	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 λ

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

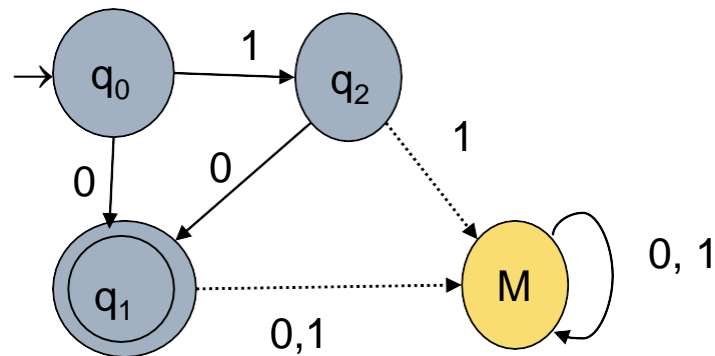
□ Para cada estado y símbolo, una sola arista (transición);

□ 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_0	q_1	q_2
*q_1	-	-
q_2	q_1	-

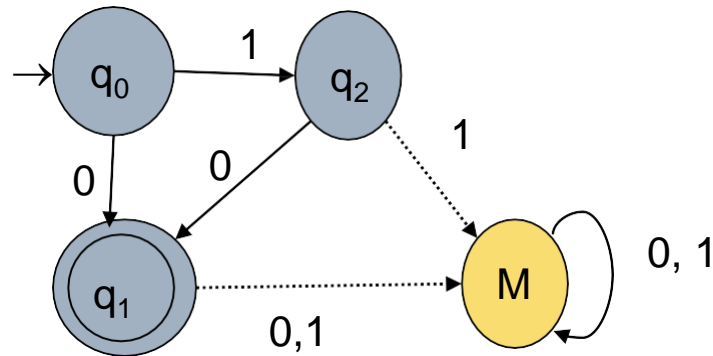


	0	1
q_0	q_1	q_2
*q_1	M	M
q_2	q_1	M
M	M	M

2.2. Representación AFD.

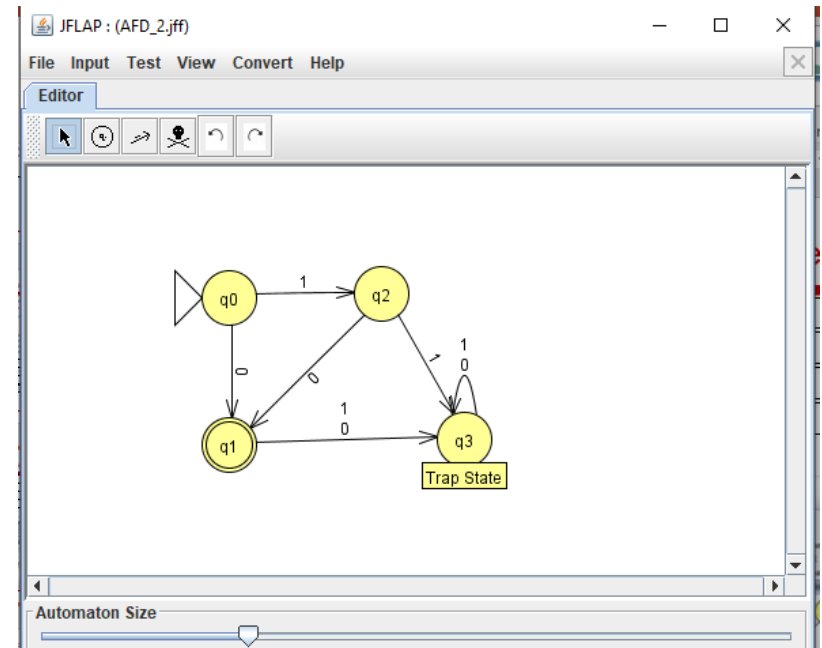
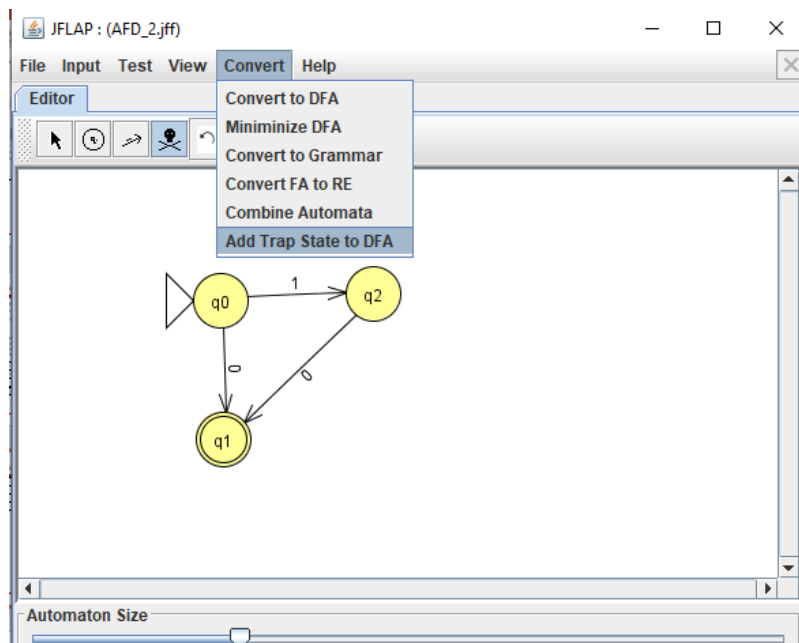
→

	0	1
q ₀	q ₁	q ₂
*q ₁	-	-
q ₂	q ₁	-



→

	0	1
q ₀	q ₁	q ₂
*q ₁	M	M
q ₂	q ₁	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 (desde un estado q) asociada a palabras (conjunto de símbolos) ax / $x \in \Sigma^*$ y $a \in \Sigma$ 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 $x \in \Sigma^*$, $a \in \Sigma$

- Dado un estado q , el resultado de una transición con la palabra ax es el estado q'' resultante de aplicar al estado q' la transición con la palabra x , siendo q' la transición asociada al estado q con el símbolo a

2.2. Lenguaje aceptado por un AFD.

- Una cadena (palabra) $x \in \Sigma^*$ 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 (cadenas) de Σ^* 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 que se transita desde el estado **q** al consumir el símbolo de entrada **c** (hasta que no queden más símbolos por consumir)

función reconocer()

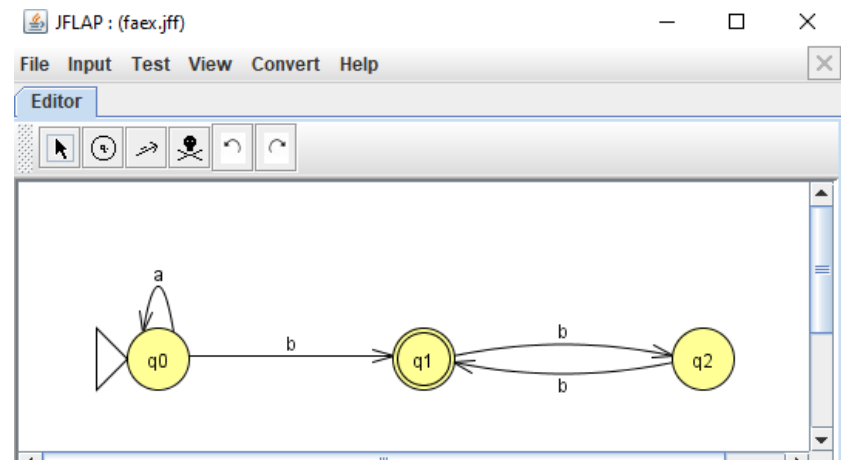
```
q = q0
c = leer_carácter()
mientras c != FDC
    q = f (q, c)
    c = leer_carácter()
fmientras
si q ∈ F entonces
    devolver(ACEPTADA)
sino
    devolver(NO ACEPTADA)
fsi
ffunción
```

```
private boolean esFinal(String estado);
public boolean reconocer(String cadena) {
    char [ ] simbolo = cadena.toCharArray( );
    String estado = 0 ; //El estado inicial es el 0
    for ( int i=0; i<simbolo.length; i++ ) {
        estado = transicion( estado, simbolo[i] );
    }
    return esFinal(estado);
}
```

2.2. Simulación algorítmica de un AFD.

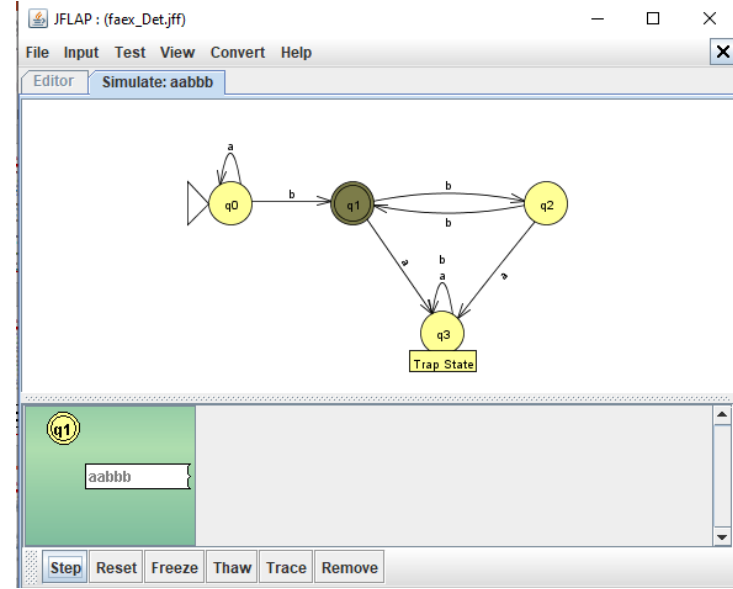
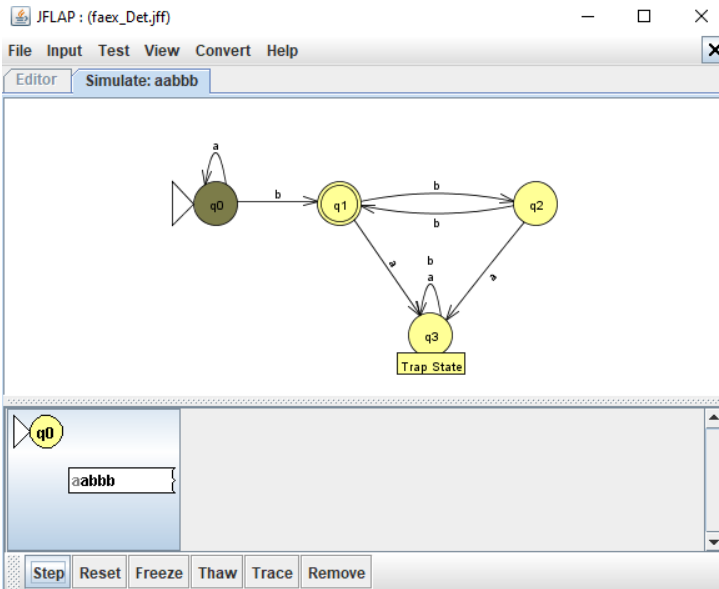
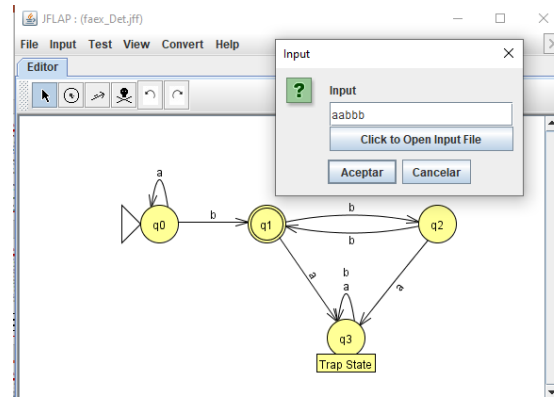
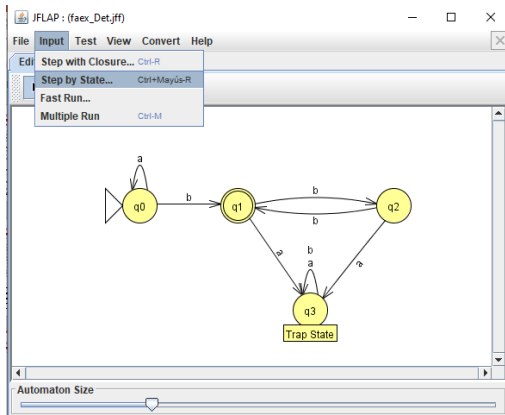
- ❑ Construir un AFD para el lenguaje $L = \{a^m b^n / m \geq 0, n > 0, n \text{ es impar}\}$. Es decir, crearemos un AFD que reconozca ese lenguaje de cualquier número de a seguido de cualquier número impar de b .
- ❑ Sabemos que las cadenas en nuestro lenguaje pueden comenzar con a , por lo que el 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 AF 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



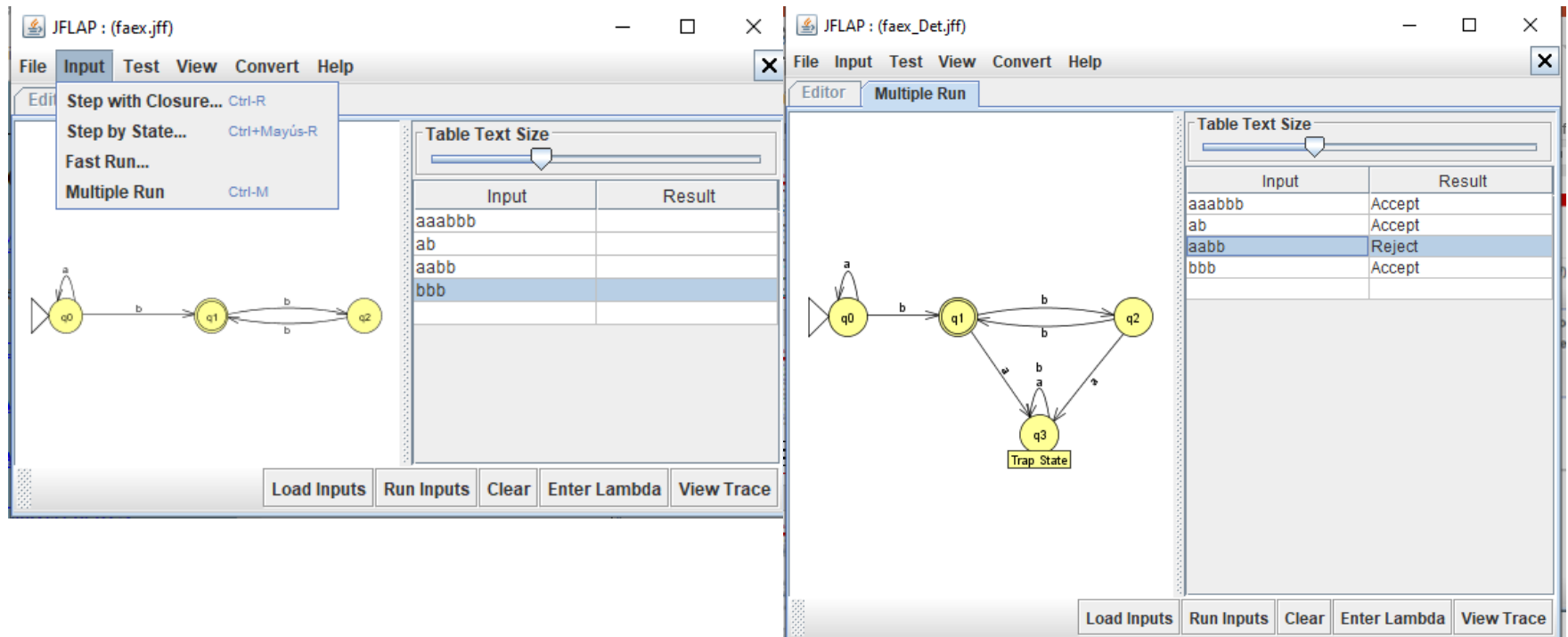
2.2. Simulación algorítmica de un AFD.

- ❑ 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.

- ❑ 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 minimizar AFD (construir el conjunto cociente Q/E_n)

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

crear una partición con dos grupos: los estados finales y los no finales

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

para cada grupo con varios estados, dividir el grupo en subgrupos tales que dos estados q_i, q_j estén en el mismo subgrupo si y solo si para cada símbolo a , la transición desde q_i con el símbolo a debe ir a un estado del mismo grupo que el de q_j con el símbolo a

- q_i y q_j están en la misma clase si y solo si $q_i, q_j \in c_k$ y $\forall a \in \Sigma$ se verifica que $f(q_i, a)$ y $f(q_j, a)$ 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}

repetir el paso 2 hasta que no se dividan más grupos

4. Cada grupo representa un estado en el AFD minimizado
5. Eliminar los estados no alcanzables desde el estado inicial y los que no tengan transiciones que puedan conducir a un estado final

2.3. Minimización de Autómatas finitos por el conjunto cociente.

- Dado el AFD $\mathbf{M} = (\mathbf{Q}, \Sigma, \mathbf{f}, \mathbf{q}_0, \mathbf{F})$
existe un único AFD equivalente mínimo
(Autómata del conjunto cociente)

$$\mathbf{M}_m = (\mathbf{Q}_m, \Sigma, \mathbf{f}_m, \mathbf{q}_{0m}, \mathbf{F}_m)$$

■ Donde

$$\mathbf{Q}_m = \mathbf{Q}/\mathbf{E}$$

$$\forall a \in \Sigma, \mathbf{f}_m(\mathbf{c}_i, a) = \mathbf{c}_j \text{ si } \exists q_i \in \mathbf{c}_i, q_j \in \mathbf{c}_j / \mathbf{f}(q_i, a) = q_j$$

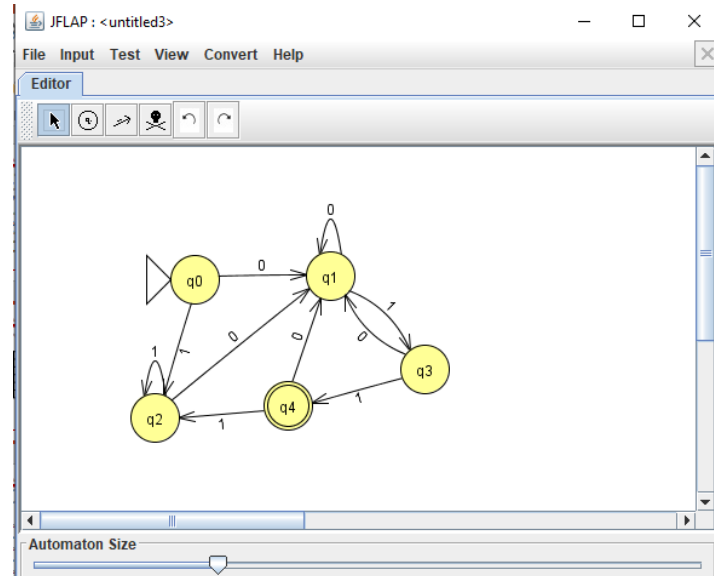
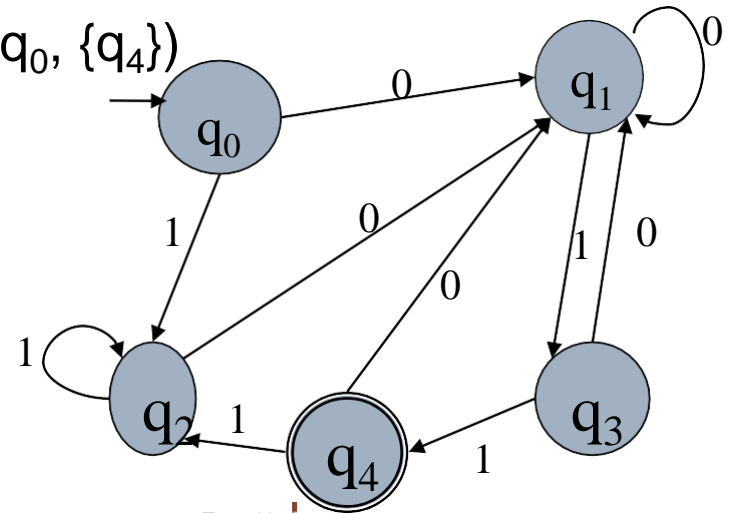
$$\mathbf{q}_{0m} = \mathbf{c}_0 \text{ si } q_0 \in \mathbf{c}_0 \text{ y } \mathbf{c}_0 \in \mathbf{Q}_m$$

$$\mathbf{F}_m = \{\mathbf{c}_i / \exists q_i \in \mathbf{c}_i \text{ y } q_i \in \mathbf{F}\}$$

2.3. Minimización de Autómatas finitos por el conjunto cociente. Ejemplo 1

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

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 1

□ Conjunto cociente

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

para cada grupo con varios estados, dos estados q_i q_j estén en el mismo grupo si y solo si para cada símbolo a , la transición desde q_i con a debe ir a un estado del mismo grupo que el de q_j con a

2. Q/E_1 Dado los grupos $A = \{q_0 \dots q_3\}$, $B = \{q_4\}$, $f(q_0, q_1, q_2) \in A$ y $f(q_3) \in B \rightarrow$ separar q_3 de A

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

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

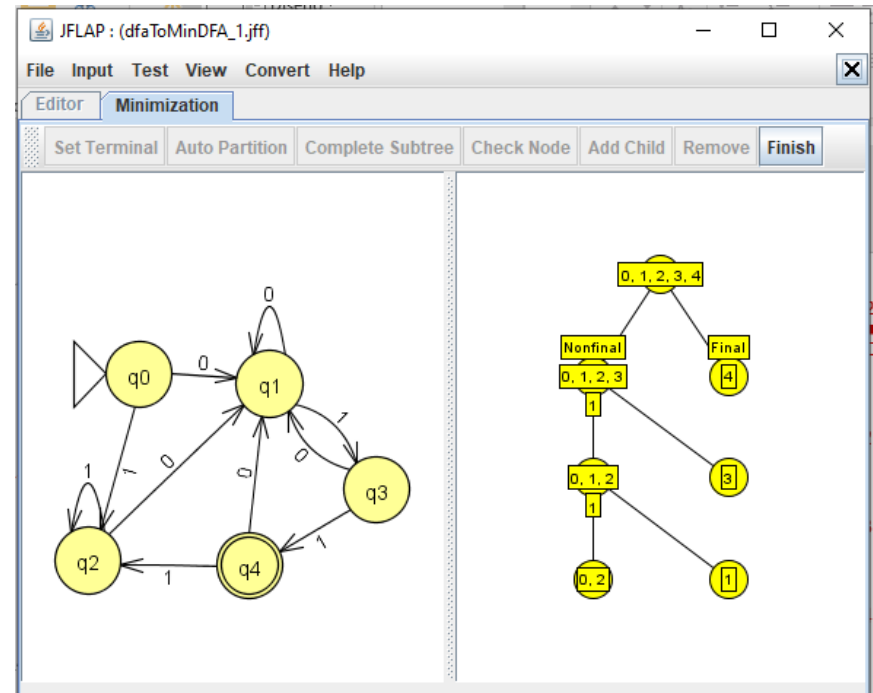
3. $Q/E_2 \neq Q/E_1$ paso 2

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



2.3. Minimización de Autómatas finitos por el conjunto cociente. Ejemplo 1

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

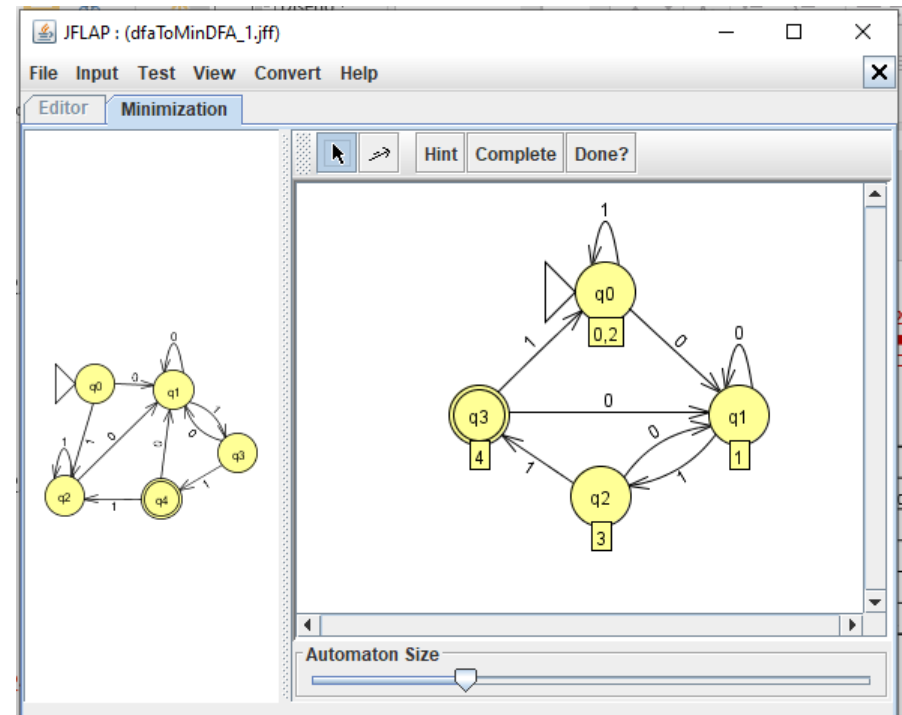
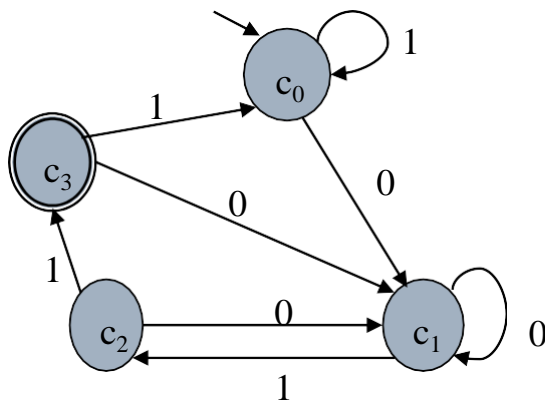
$$Q_m = (c_0 = \{q_0, q_2\}, c_1 = \{q_1\}, c_2 = \{q_3\}, c_3 = \{q_4\})$$

$$q_{om} = c_0$$

$$F_m = c_3$$

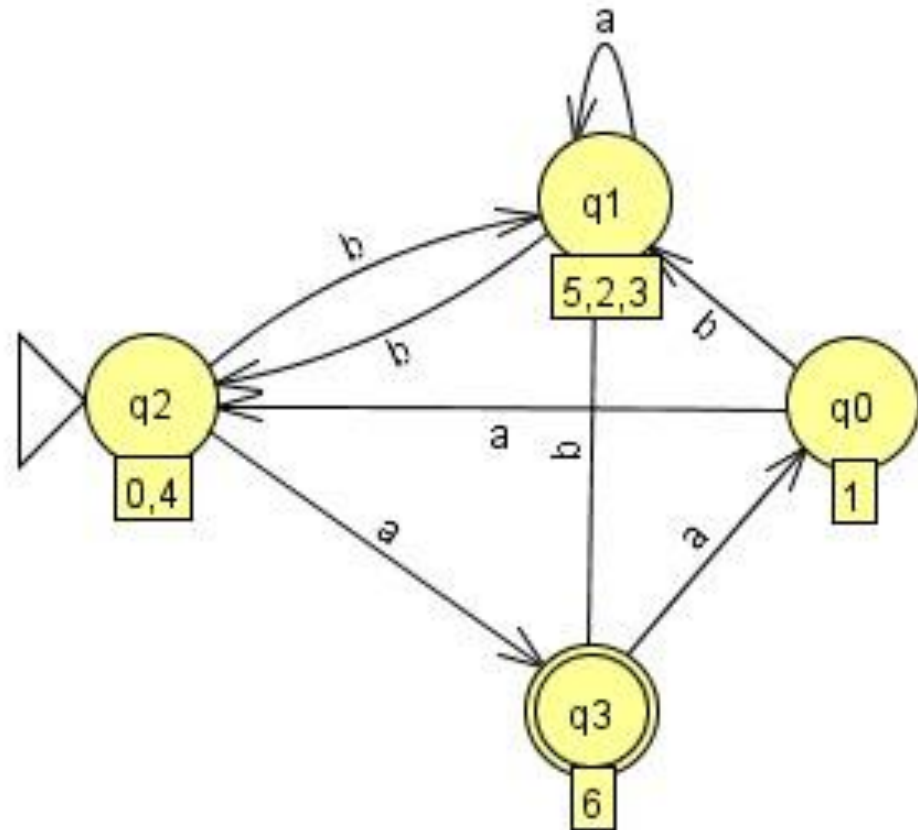
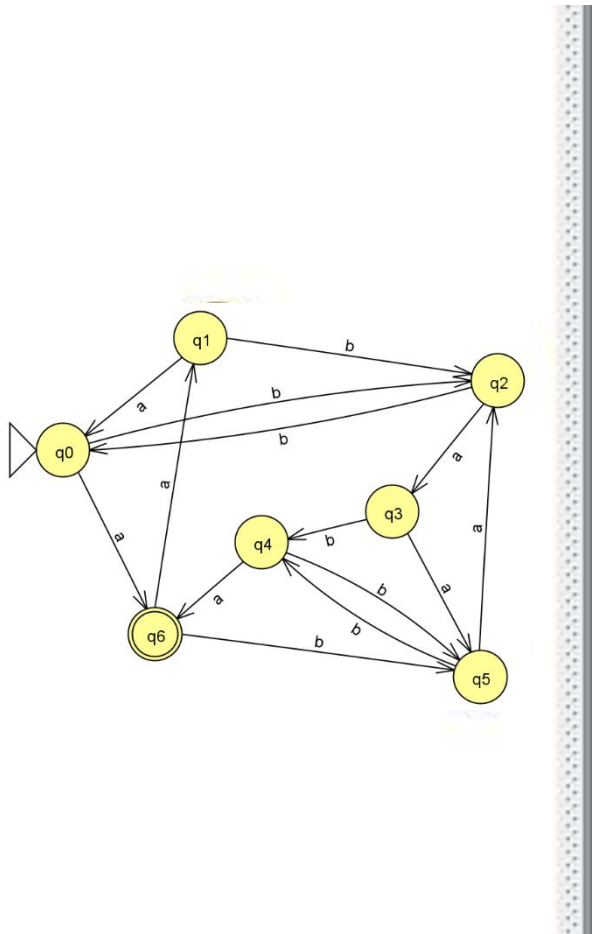
f	0	1
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
$*q_4$	q_1	q_2

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



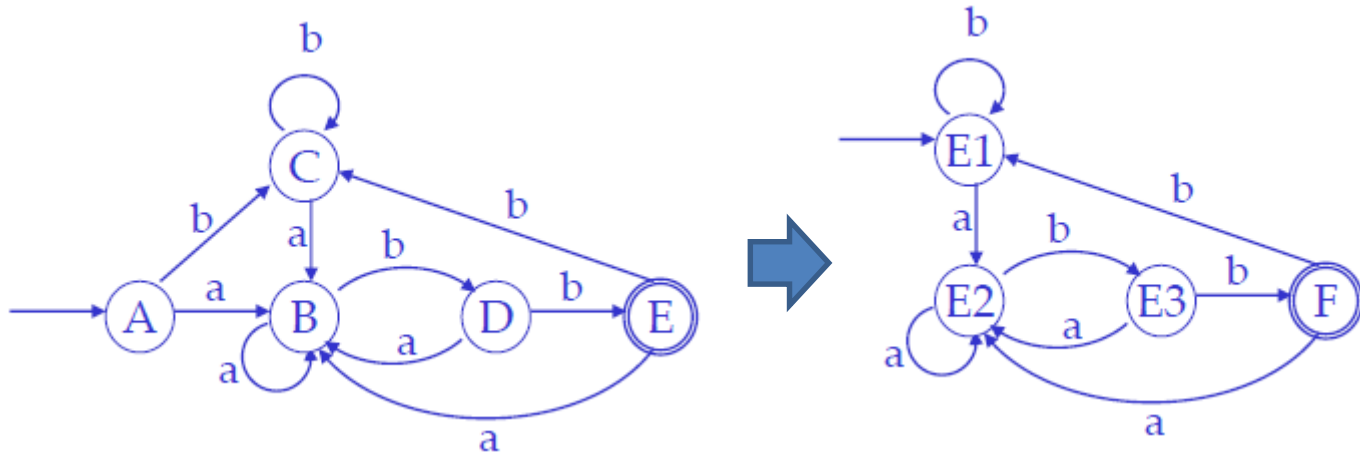
2.3. Minimización de Autómatas finitos por el conjunto cociente. Ejemplo 2

- Minimizar el siguiente AFD:



2.3. Minimización de Autómatas finitos por el conjunto cociente. Ejemplo 3

□ Minimizar el siguiente AFD:



	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

		a	b
E1	A	E1	E1
	B	E1	E1
	C	E1	E1
	D	E1	F
F	E	E1	E1

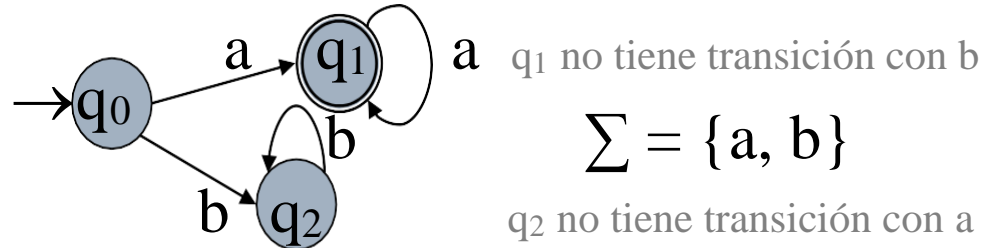
		a	b
E1	A	E1	E1
	B	E1	E2
	C	E1	E1
E2	D	E1	F
F	E	E1	E1

		a	b
E1	A	E2	E1
	C	E2	E1
E2	B	E2	E3
E3	D	E2	F
F	E	E2	E1

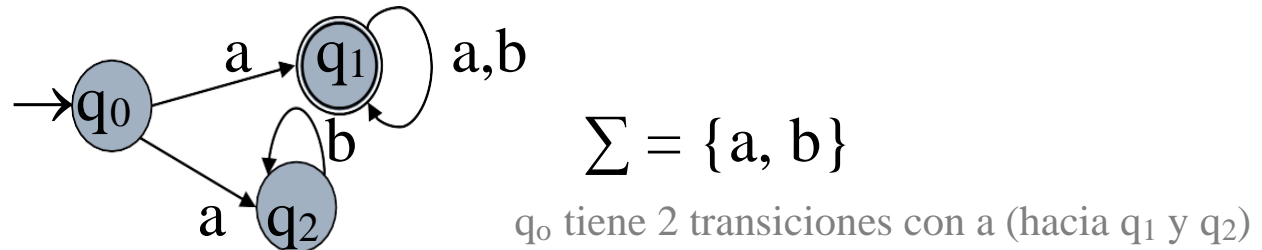
2.4. Autómatas Finitos No Deterministas. Introducción.

□ Un autómata finito es no determinista si:

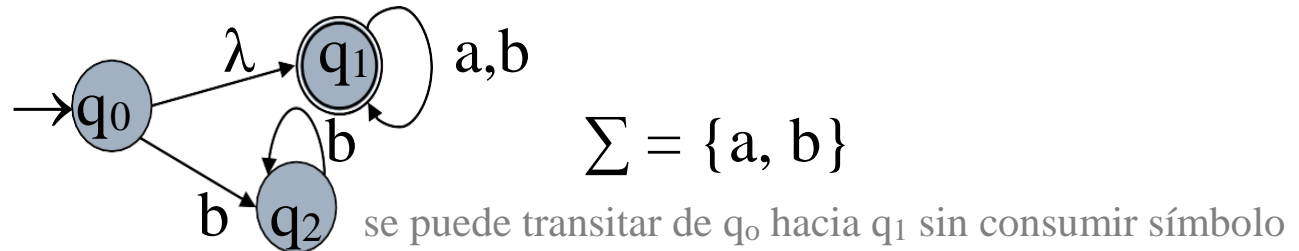
- No $\exists f(q, a)$ para algún $a \in \Sigma$ desde algún $q \in Q$



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



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



2.4. Autómatas Finitos No Deterministas (AFND). Definición

□ 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*.
- Σ 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

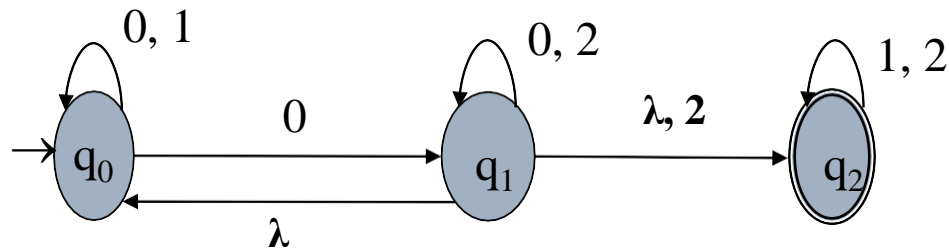
- q_0 es un elemento o estado de Q , llamado *estado inicial*.
- F es un subconjunto de Q ($F \subseteq Q$), llamado *conjunto de estados finales*.

2.4. Representación AFND.

□ Tablas de transición.

	0	1	2	λ
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$	\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.



2.4. AFND. λ -clausura

□ Definición λ -clausura:

- Se llama **λ -clausura** de **un estado** al conjunto de estados (incluido el propio estado) a los que se puede evolucionar sin consumir ninguna entrada, lo denotaremos como $CL(q)$ o λ -clausura(q).
- $CL(q)$ se define **recursivamente**:
 - El estado q pertenece a la λ -clausura de q , $q \in CL(q)$.
 - Si el estado $p \in CL(q)$ tiene una transición del estado p al estado r etiquetada con una transición nula (λ), entonces r también está en $CL(q)$.
- La **λ -clausura** de un estado **q** es ese mismo estado **q** más todos aquellos estados **p** a los que se puede transitar desde **q** mediante transiciones λ (si esos estados **p** contienen a su vez transiciones λ , los estados **p'** a los que se llega desde los estados **p** con transiciones λ , también forman parte de la λ -clausura de **q** y así sucesivamente).

2.4. AFND. λ -clausura

❑ Definición λ -clausura:

- Se llama **λ -clausura** de **un conjunto de estados** $q \in P$ (macroestado P) al conjunto de estados (macroestados) a los que se puede transitar desde cada estado q del conjunto P sin consumir ninguna entrada.
- Si $P \subseteq Q$ se llama λ -clausura de un conjunto de estados $q \in P$:

$$\lambda\text{-clausura}(P) \text{ o } CL(P) = \bigcup_{q \in P} CL(q)$$

λ -clausura del conjunto de estados P es el conjunto de estados formados por todos los estados de P más aquellos a los que se puede acceder desde P mediante transiciones λ

❑ Algoritmo para el cálculo de λ -clausura(P)

Procedimiento λ -clausura(P)

Viejos= \emptyset Nuevos= P

mientras Viejos \neq Nuevos **hacer**

Viejos=Nuevos

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

fmientras

λ -clausura(P)=Nuevos

fin_procedimiento

2.4. AFND. función de transición

- Si $M = (Q, \Sigma, f, q_0, F)$ es un AFND se define la **función de transición** (desde **un conjunto de estados** $q \in P$, $P \subseteq Q$) asociada a un símbolo $a \in \Sigma$ como la función

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

- $f(q_0, \lambda) = CL(q_0)$ el macroestado inicial del AFND es λ -clausura(q_0)
- $f(P, \lambda) = P$
- $f(P, a) = CL(\bigcup_{q \in P} f(q, a))$ donde $P \subseteq Q$, $a \in \Sigma$

- Dado un macroestado P , el resultado de una transición con el símbolo a es un macroestado $P' = \lambda$ -clausura(P_a), siendo P_a todos aquellos estados a los que se puede transitar con el símbolo a desde cada uno de los estados $q \in P$

2.4. AFND. función de transición: Extensión a palabras.

- Si $M = (Q, \Sigma, f, q_0, F)$ es un AFND se define la **función de transición** (desde **un conjunto de estados** $q \in P, P \subseteq Q$) asociada a **palabras** (conjunto de símbolos) $ax / x \in \Sigma^*$ y $a \in \Sigma$ como la función

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

■ $f'(q_0, \lambda) = CL(q_0) = f(q_0, \lambda)$

■ $f'(P, \lambda) = P = f(P, \lambda)$

■ $f'(P, ax) = f'(CL(\bigcup_{q \in P} f(q, a)), x)$ donde $P \subseteq Q, x \in \Sigma^*$ y $a \in \Sigma$

- Dado un macroestado P , el resultado de una transición con la palabra ax es el macroestado P'' resultante de aplicar al macroestado P' la transición con la palabra x , siendo P' la transición asociada al macroestado P con el símbolo a

2.4. Lenguaje aceptado por un AFND.

- ❑ El macroestado asociado a una **cadena (palabra)** $x \in \Sigma^*$ es el conjunto de estados a los que se puede llegar partiendo del estado inicial y realizando las transiciones con los símbolos de la cadena x : $f'(q_0, x)$
- ❑ Una **cadena (palabra)** $x \in \Sigma^*$ es *aceptada* por un AFND $M = (Q, \Sigma, f, q_0, F)$ si y solo si $f'(q_0, x) \cap F \neq \emptyset$. Es decir si su macroestado contiene algún estado final (si alguno de los estados en los que termina al transitar la cadena es estado final)

En un AFND, dada una cadena pueden existir varios caminos. Será aceptada si al menos uno de los caminos conduce a un estado final

- ❑ En otro caso se dice que la cadena es *rechazada* por el autómata
- ❑ El *lenguaje aceptado/reconocido* por un autómata AFND $M = (Q, \Sigma, f, q_0, F)$ es el conjunto de las palabras de Σ^* que acepta:

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

2.4. 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 conjunto de estados a los que se transita desde el estado **q** al consumir el carácter de entrada **c** (hasta que no queden más símbolos por consumir)

función reconocer()

$A = CL(q_0)$

$c = leer_carácter()$

mientras $c \neq FDC$

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

$c = leer_carácter()$

fmientras

si $A \cap F \neq \emptyset$ **entonces**
devolver(ACEPTADA)

sino

devolver(NO
ACEPTADA)

fsi

ffunción

```
private boolean esFinal(String estado);

public boolean esFinal(String [ ] macroestado) {
    for ( int i=0; i<macroestado.length; i++ ) {
        if (esFinal( macroestado[i] )
            return true;
    }
    return false;
}

public boolean reconocer(String cadena) {
    char [ ] simbolo = cadena.toCharArray( );
    String [ ] estado = { 0 }; //El estado inicial es el 0
    String [ ] macroestado =  $\lambda$ _clausura(estado);
    for ( int i=0; i<simbolo.length; i++ ) {
        macroestado = transicion( macroestado, simbolo[i] );
    }
    return esFinal(macroestado);
}
```

2.5. Equivalencia entre AFND y AFD.

- Los AFD tienen la misma capacidad expresiva que los AFND, es decir, dado un AFND existe un AFD capaz de reconocer el mismo lenguaje.

Cada macroestado del AFN corresponde a un estado del AFD.

Potencialmente, para un AFN de N estados existen 2^N macroestados posibles, aunque la inmensa mayoría son estados inalcanzables

- A partir de un AFND $M=(Q, \Sigma, f, q_0, F)$ 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.
- Los estados finales del AFD serán aquellos que correspondan a macroestados que contengan algún estado final del AFND

2.5. Equivalencia entre AFND y AFD. Algoritmo

A partir de un AFND $M=(Q, \Sigma, f, q_0, F)$ con estados q_0, \dots, q_m , construiremos un AFD equivalente con estados Q_0, \dots, Q_n donde q_0, Q_0 , son los estados iniciales.

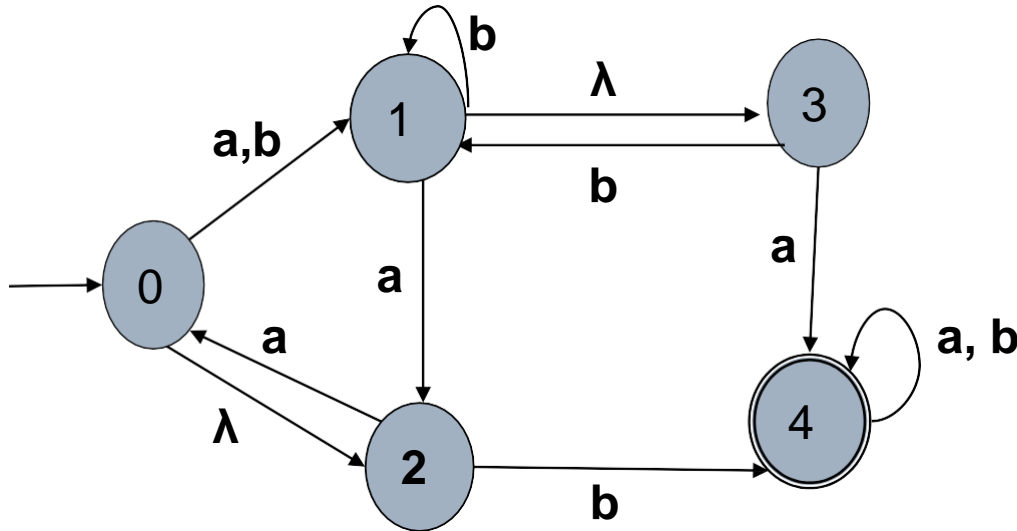
□ Algoritmo para crear un AFD a partir de un AFND:

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 $T[Q_i, a] \quad a \in \Sigma$:
 1. se asigna $T[Q_i, a] = \lambda\text{-clausura}(f(Q_i, a))$
 2. si no existe una entrada en T para el estado $T[Q_i, a]$, se crea.
5. se repite 4 mientras existan casillas vacías.

Los estados finales del AFD serán aquellos que correspondan a macroestados que contengan algún estado final del AFND

2.5. 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	∅

2.5. 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		

	a	b	λ
0	1	1	2
1	2	1	3
2	0	4	\emptyset
3	4	1	\emptyset
*4	4	4	\emptyset

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

	a	b
Q_0	Q_1	
Q_1		

5. se repite 4 mientras existan casillas vacías.

2.5. 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 ₀	Q ₁	Q ₂
Q ₁		
Q ₂		

2.5. 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

2.5. Equivalencia entre AFND y AFD. Algoritmo alternativo

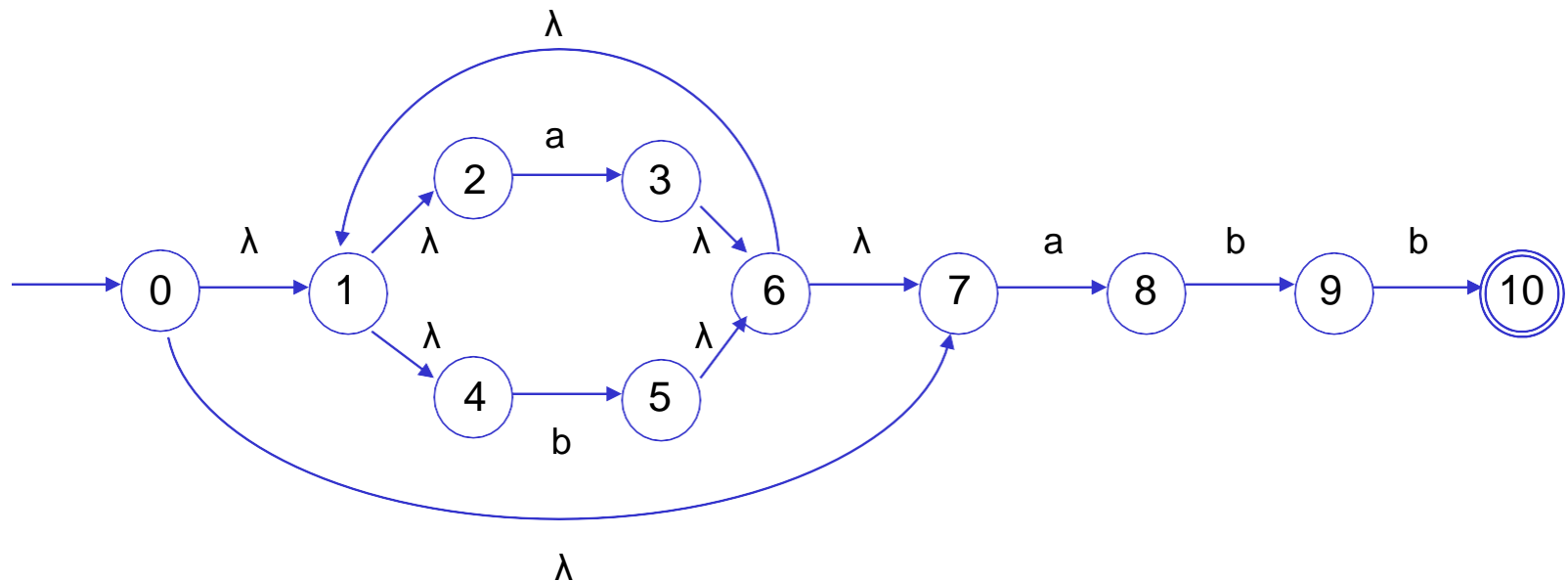
A partir de un AFND $M=(Q, \Sigma, f, q_0, F)$ con estados q_0, \dots, q_m , construiremos un AFD equivalente con estados Q_0, \dots, Q_n donde q_0, Q_0 , son los estados iniciales.

- **Algoritmo alternativo para crear un AFD a partir de un AFND:**
 - Generar el estado inicial del AFD como el macroestado inicial del AFND e incluirlo en una lista de estados por analizar.
 - Analizar el primer estado de la lista por analizar:
 - Extraer el primer estado de la lista e introducirlo en la lista de estados analizados
 - Estudiar las transiciones del estado para cada símbolo del alfabeto
 - Si el macroestado correspondiente a una transición no ha aparecido con anterioridad, crear un nuevo estado del AFD correspondiente a dicho macroestado e incluirlo en la lista de estados por analizar
 - Repetir el paso anterior hasta que no queden estados por analizar.

Los estados finales del AFD serán aquellos que correspondan a macroestados que contengan algún estado final del AFND

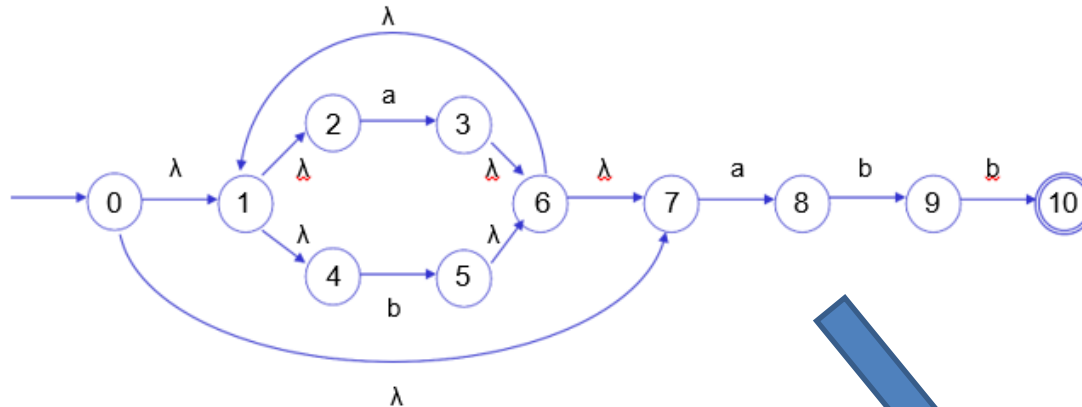
2.5. Equivalencia entre AFND y AFD. Ejemplo

□ Ejemplo: $(a \mid b)^* abb$:

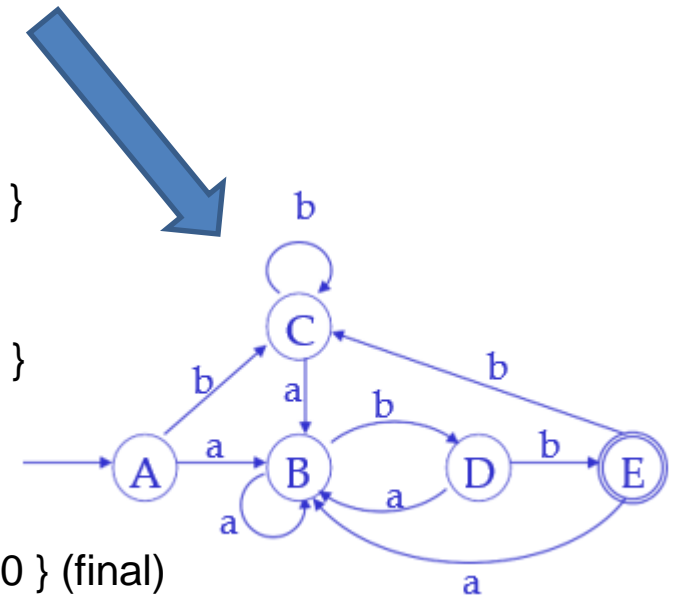


2.5. Equivalencia entre AFND y AFD. Ejemplo

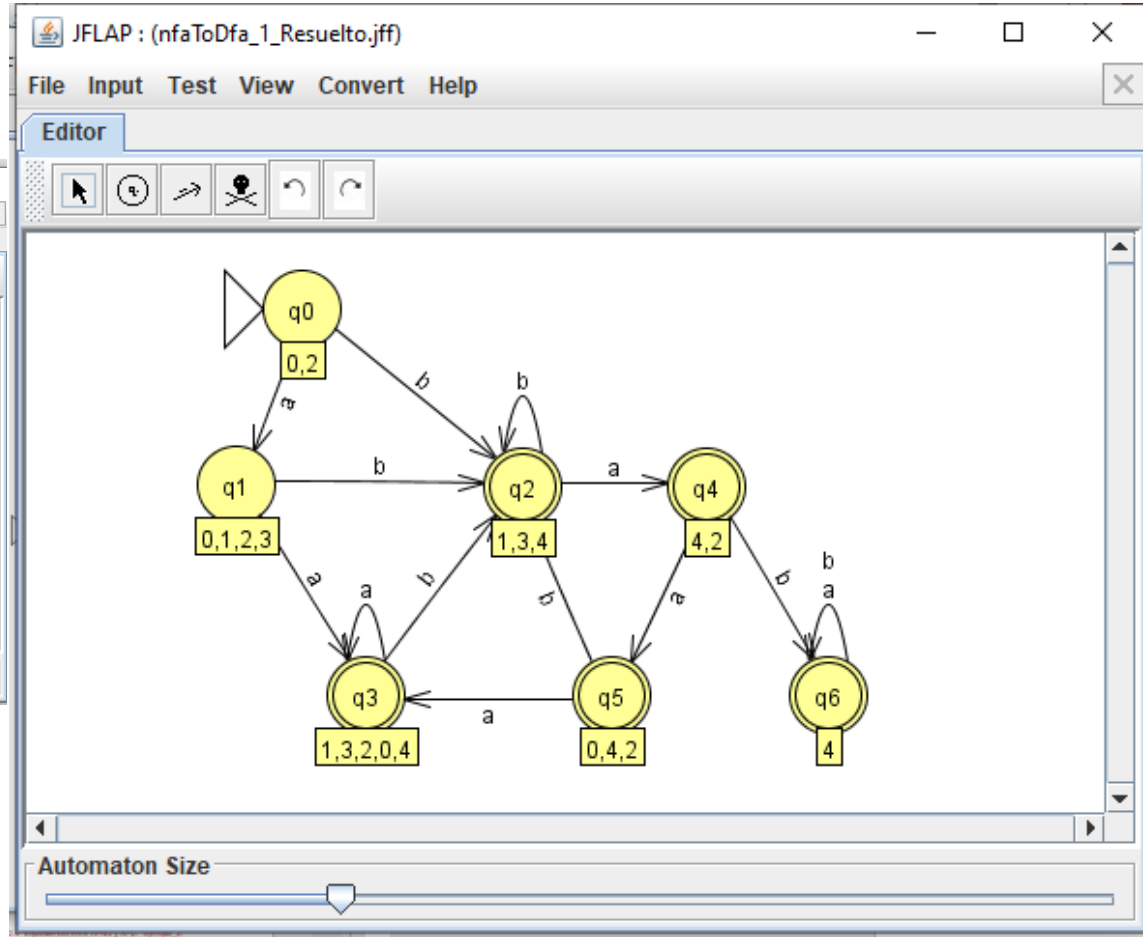
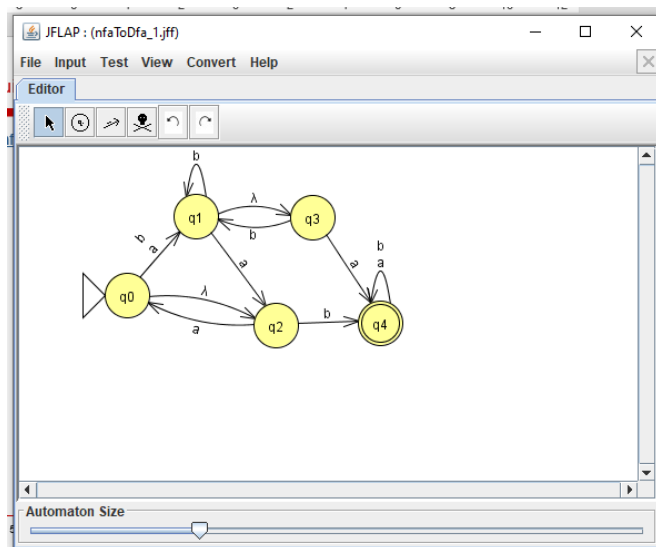
□ El AFD equivalente es:



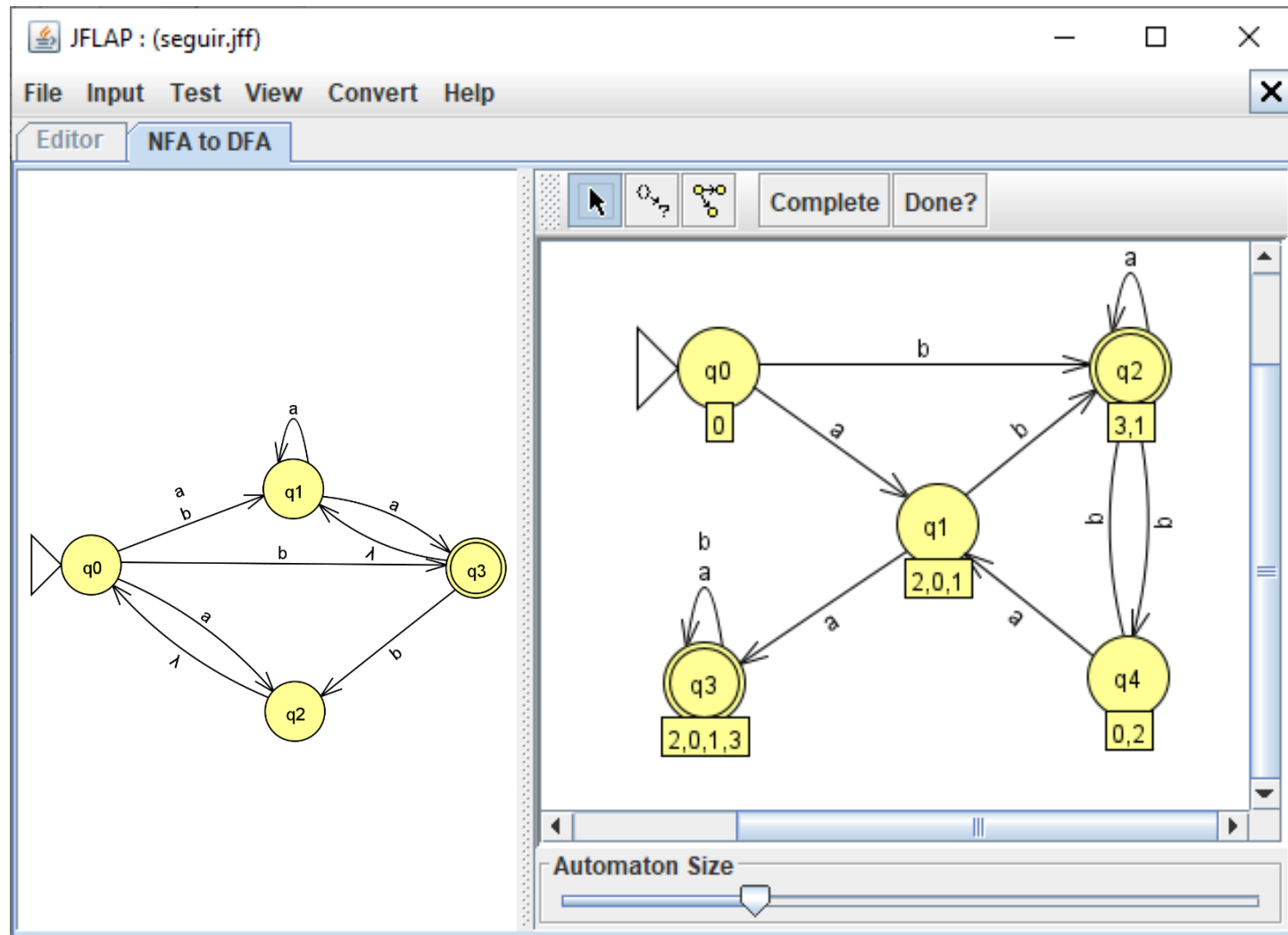
- Estado inicial: $A = \{ 0, 1, 2, 4, 7 \}$
- Transición de A con 'a': $B = \{ 1, 2, 3, 4, 6, 7, 8 \}$
- Transición de A con 'b': $C = \{ 1, 2, 4, 5, 6, 7 \}$
- Transición de B con 'a': B
- Transición de B con 'b': $D = \{ 1, 2, 4, 5, 6, 7, 9 \}$
- Transición de C con 'a': B
- Transición de C con 'b': C
- Transición de D con 'a': B
- Transición de D con 'b': $E = \{ 1, 2, 4, 5, 6, 7, 10 \}$ (final)
- Transición de E con 'a': B
- Transición de E con 'b': C



2.5. Equivalencia entre AFND y AFD. Ejemplo 1.



2.5. Equivalencia entre AFND y AFD. Ejemplo 2.



2.6. 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.
- Sea el AFD $M=(Q, \Sigma, f, q_0, F)$, 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 $q_0 \rightarrow \lambda$

2.6. Autómatas finitos y gramáticas regulares.

□ Ejemplo 1: 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:

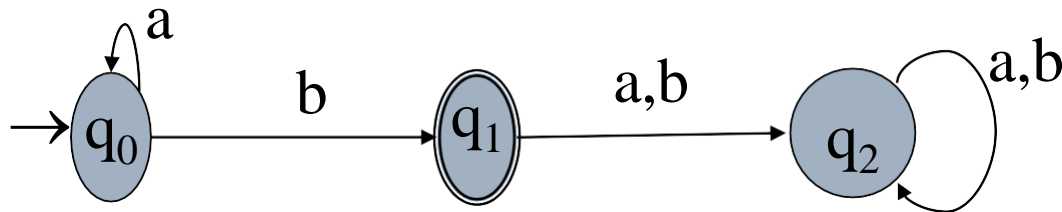
- $\Sigma_N = Q, \quad \Sigma_T = \Sigma, \quad S = q_0$

- $P = \{ (q \rightarrow ap) / f(q, a) = p \} \cup \{ (p \rightarrow \lambda) \mid (q_0 \rightarrow \lambda) / p, q_0 \in F \}$

o bien:

- $P = \{ (q \rightarrow ap) / f(q, a) = p ; \text{ si } p \in F \text{ añadir } (q \rightarrow a) \} \cup \{ (q_0 \rightarrow \lambda) / q_0 \in F \}$

□ Ej.- Considera el siguiente **DFA** que acepta el lenguaje a^*b : reconoce $b, ab, aab, a...ab$
no reconoce $a, ba, aba, babba, \dots$



□ La gramática será:

- $q_0 \rightarrow aq_0 \mid bq_1$

- $q_1 \rightarrow aq_2 \mid bq_2 \mid \lambda$

- $q_2 \rightarrow aq_2 \mid bq_2$

$$q_0 \rightarrow aq_0 \mid bq_1 \mid b$$

$$q_1 \rightarrow aq_2 \mid bq_2$$

$$q_2 \rightarrow aq_2 \mid bq_2$$

2.6. Autómatas finitos y gramáticas regulares.

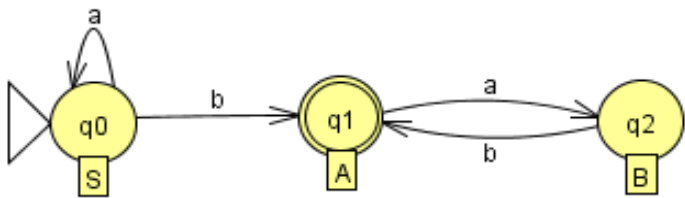
□ Ejemplo 2: De AFD a Gramática regular:

JFLAP : (dfaToRegGrammar.jff)

File Input Test View Convert Help

Editor Convert to Grammar

Hint Show All What's Left? Export



The diagram shows a finite automaton with three states: q0, q1, and q2. q0 is the start state (indicated by a double arrow) and has a self-loop labeled 'a'. q1 is an accepting state (indicated by a double circle) and is reached from q0 by the transition labeled 'b'. q2 is reached from q1 by the transition labeled 'a' and has a transition labeled 'b' back to q1. Each state has a label below it: 'S' for q0, 'A' for q1, and 'B' for q2.

LHS		RHS
S	→	aS
S	→	bA
A	→	aB
B	→	bA
A	→	λ

Table Text Size

LHS	RHS
S	→ aS
S	→ bA
A	→ aB
S	→ b
A	→ aB
B	→ bA
B	→ b

2.6. 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:
- Sea la gramática regular $G=(\Sigma_N, \Sigma_T, S, P)$, se construye el AFND- λ 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- λ obtenido a un AFD.

- Cada símbolo no terminal Σ_N genera un estado $q \in Q$ y se añade un estado final F ($F \in Q$)
- Los símbolos terminales Σ_T es el alfabeto Σ del AF
- El símbolo inicial S genera el estado inicial q_0
- Cada producción de P de tipo $A \rightarrow aB$ genera una transición del estado A al B con el símbolo ‘a’
- Cada producción de P de tipo $A \rightarrow a$ genera una transición del estado A al estado final F con ‘a’
- Si existe la producción $S \rightarrow \lambda$ se genera una transición λ del estado S al estado final F

2.6. Autómatas finitos y gramáticas regulares.

□ Ejemplo: De Gramática regular a AFND

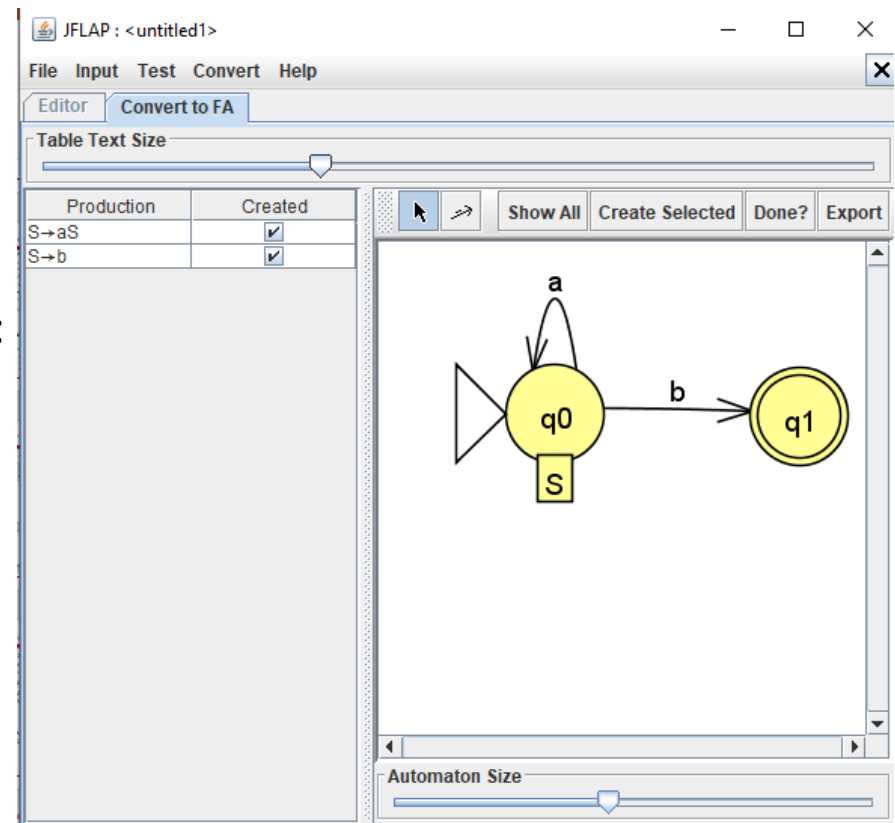
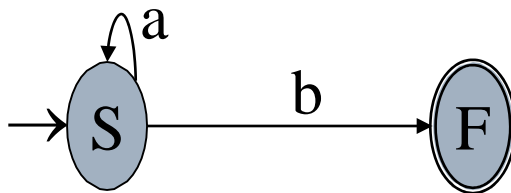
Dado un lenguaje regular L generado por una gramática regular $G=(\Sigma_N, \Sigma_T, S, P)$, se puede obtener un AFD $M=(Q, \Sigma, f, q_0, F)$ que reconozca el mismo lenguaje de la siguiente forma:

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

□ Ej.- Considere la siguiente gramática:

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

□ El AFD equivalente:



2.7. 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, etc.
- En las expresiones regulares pueden aparecer **símbolos** de dos tipos:
 - **Símbolos base**: son los del alfabeto Σ del lenguaje **L** que queremos describir, λ 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.

2.7. 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 Σ y los símbolos: \emptyset (lenguaje vacío), λ (palabra vacía), \cdot (concatenación), $+$ (unión), $*$ (clausura), se cumple:

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

Sólo son **expresiones regulares** las que se pueden obtener aplicando las reglas anteriores.

- Se establece la siguiente prioridad en las operaciones:

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

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

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

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

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

- Si α y β son dos **expresiones regulares** entonces:

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

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

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

2.7. 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:

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

.....

$d_n \rightarrow r_n$

Donde cada d_i es un nombre distinto y cada r_i una **expresión regular** sobre los símbolos del alfabeto $\Sigma \cup \{d_1, d_2, \dots, d_n\}$

□ Ejemplo: definición regular para identificadores en Pascal:

letra $\rightarrow A \mid B \mid \dots \mid Z \mid a \mid b \mid \dots \mid z$

digito $\rightarrow 0 \mid 1 \mid \dots \mid 9$

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

□ Abreviaturas en la notación

- Uno o más casos: operador unitario posfijo $^+$ no confundir con operador $+$ (unión)
- Cero o un caso: operador unitario posfijo $?$
- Clases de caracteres: $[abc]$

□ Ejemplos:

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

exponente_opcional $\rightarrow (E(+|-)?\text{digitos})?$

2.7. Lenguaje de las expresiones regulares.

□ Ejemplo:

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

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

2.7. 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 \cdot :

- 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 $*$:

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

2.7. Teorema fundamental.

- El Teorema servirá para describir mediante una **expresión regular** el **lenguaje** reconocido por un autómata.
- 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 en los que $\lambda \notin L(\alpha)$, respectivamente.

■ Regla General:

- **Teorema.** Sean α y β 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 γ representa cualquier expresión regular.

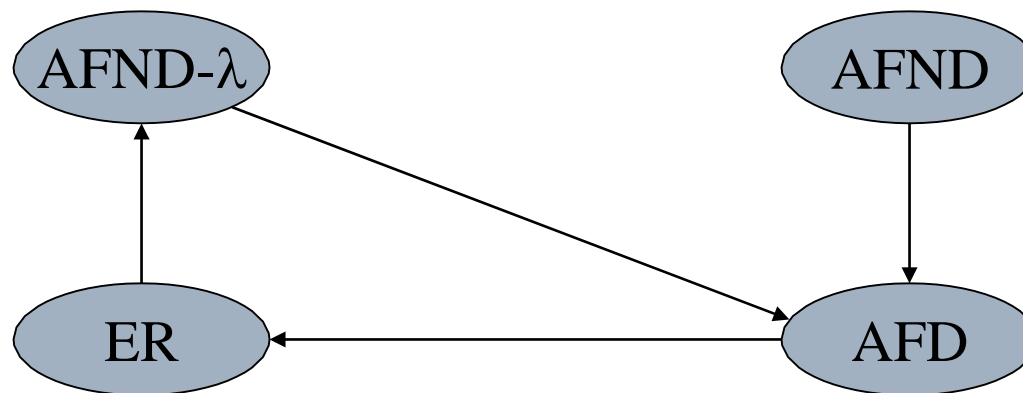
■ Regla de Inferencia:

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

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

2.7. 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 también definido por un **autómata finito** (AFD, AFND).

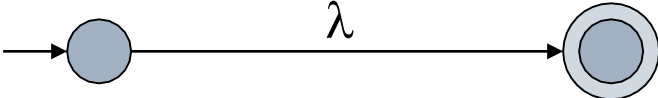

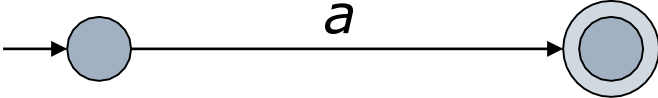


2.7. AFND asociado a una expresión regular

- ❑ Cada lenguaje definido por una expresión regular es también definido por un autómata finito.
- ❑ La demostración es inductiva sobre el número de operadores de α (+, . *).

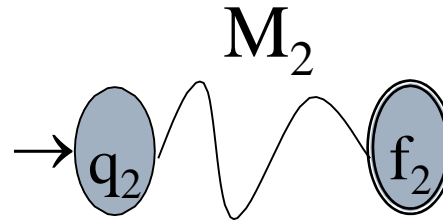
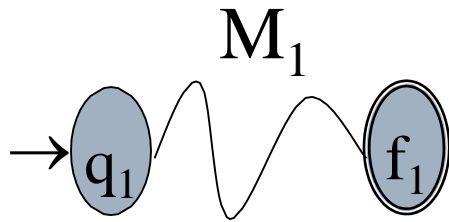
📎 **Base.**- (cero operadores)

- La expresión regular α puede ser \emptyset , λ , a , donde $a \in \Sigma_T$

Expresión regular α	Autómata
λ	
\emptyset	
a	

2.7. AFND asociado a una expresión regular

- 📌 **Inducción.**- (uno o más operadores en la expresión regular α)
- ❑ 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 acepta el mismo lenguaje $L(M_1) = L(\alpha_1)$ y $L(M_2) = L(\alpha_2)$ donde:



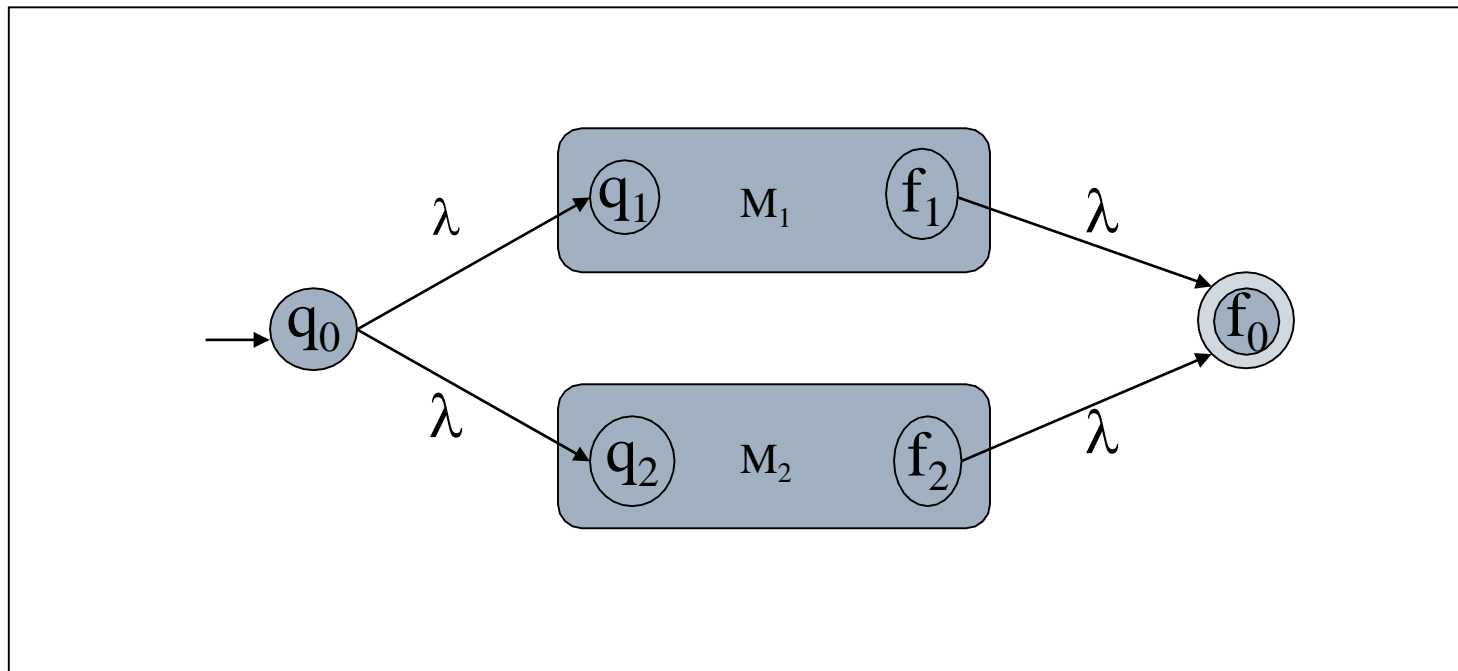
2.7. AFND asociado a una expresión regular

- Suponemos que tenemos una **expresión regular** α 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 α en función de otras **expresiones regulares** con menos de n operadores:

1. $\alpha = \alpha_1 + \alpha_2$ tal que $\text{op}(\alpha_1), \text{op}(\alpha_2) < n$ (unión)
2. $\alpha = \alpha_1 \cdot \alpha_2$ tal que $\text{op}(\alpha_1), \text{op}(\alpha_2) < n$ (concatenación)
3. $\alpha = (\alpha_1)^*$ tal que $\text{op}(\alpha_1) = n-1$ (clausura)

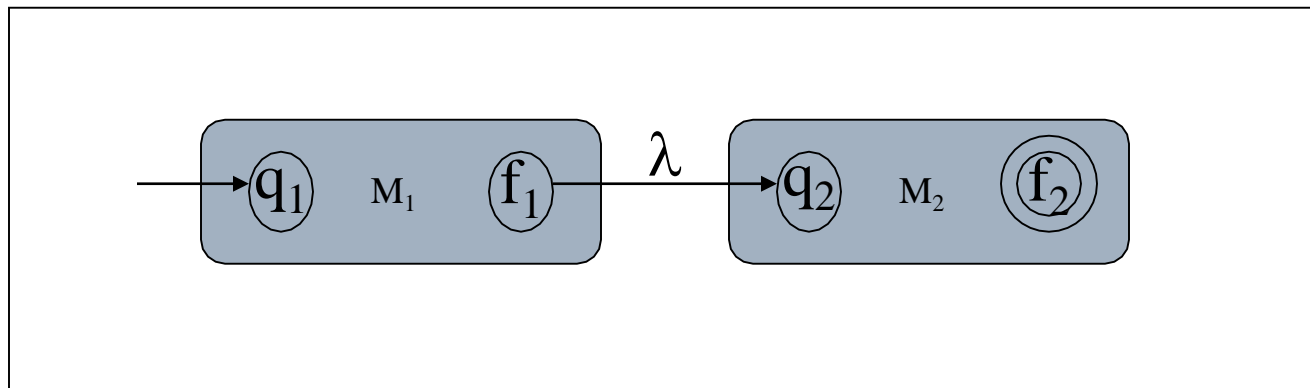
2.7. AFND asociado a una expresión regular

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:



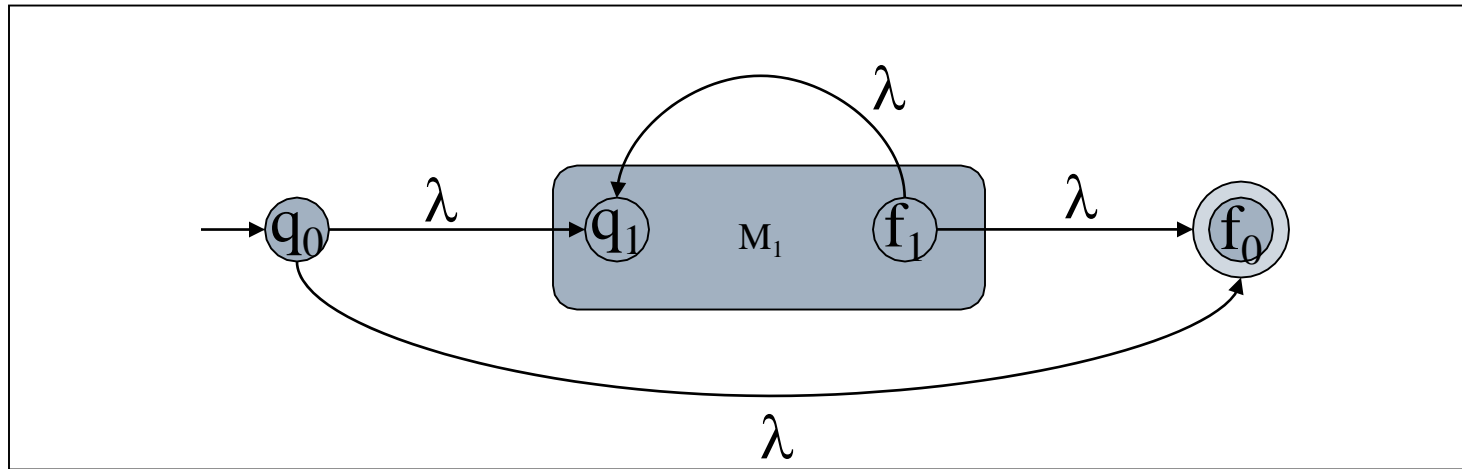
2.7. AFND asociado a una expresión regular

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:



2.7. AFND asociado a una expresión regular

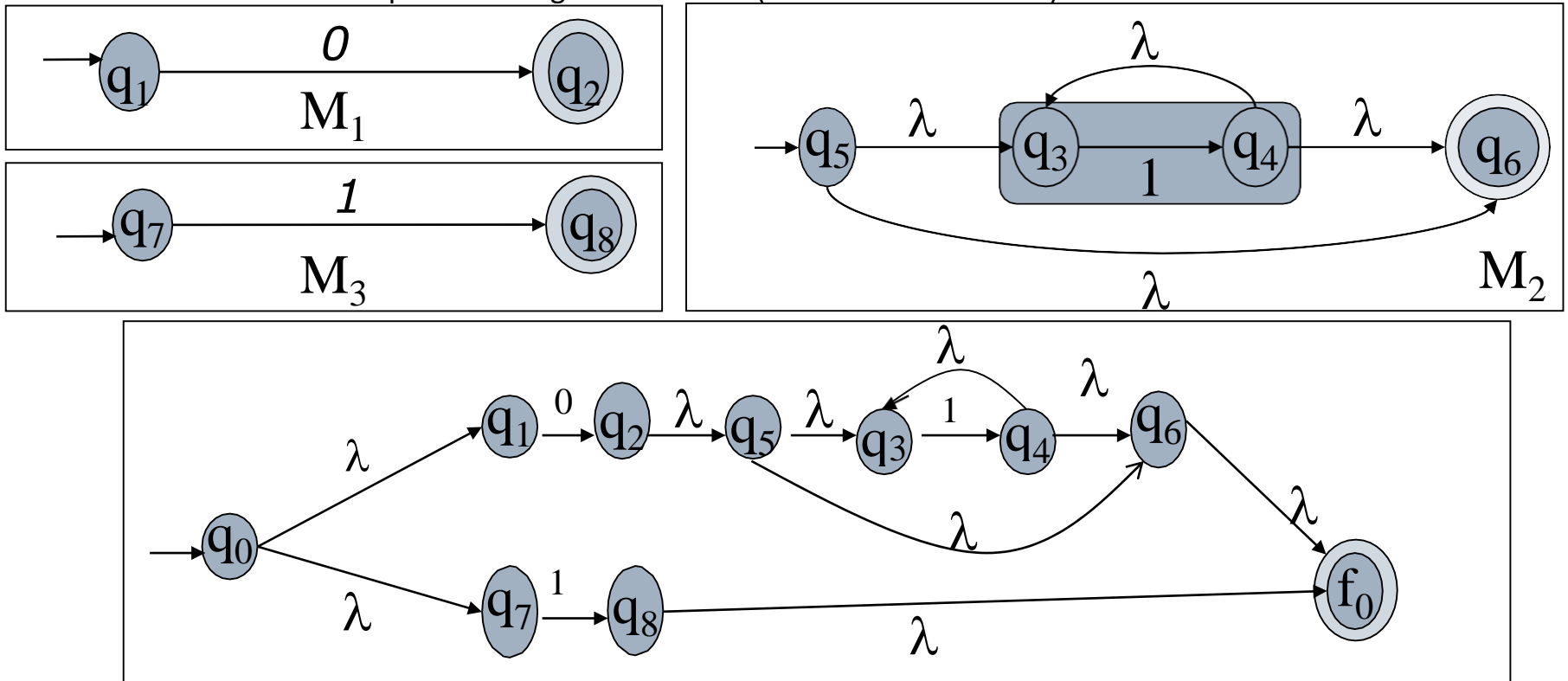
3. $\alpha = (\alpha_1)^*$ tal que $op(\alpha_1) = n-1$
- A partir de M_1 construimos otro autómata M (la clausura), el autómata que acepta el mismo lenguaje es:



2.7. AFND 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 los autómatas para la concatenación (0 con 1^* , M_1 con M_2) y la suma de expresiones regulares $01^* + 1$ (M_1 con M_2 union M_3)



2.7. Expresión regular asociada a un AFD

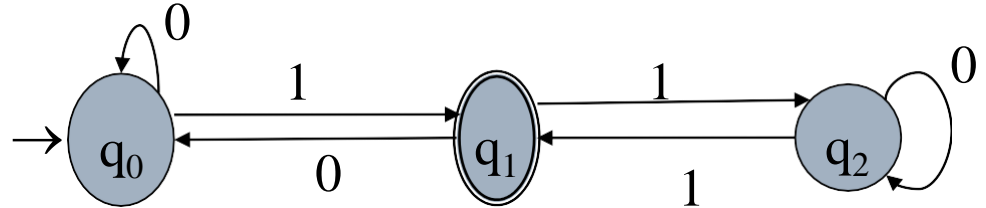
- Si L es un lenguaje aceptado por un autómata finito M entonces existe una expresión regular α tal que $L = L(M) = L(\alpha)$.
 - Podemos suponer que el autómata finito M no tiene λ -transiciones. Si las tuviera, podemos encontrar autómata equivalente sin λ -transiciones (pasar de AFND a AFD)
- Sea el AFD $M = (Q, \Sigma, f, q_0, F)$. 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 del AFD le corresponde una ecuación y cada estado q_i del AFD 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 , denotado como 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 λ 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 α_i es una expresión regular sobre el alfabeto Σ
- El lenguaje descrito por esta expresión regular es:
$$L(\alpha_i) = \{w \in \Sigma^* \mid (q_i, w) \vdash^* (q_F, \lambda), q_F \in F\} \quad (1)$$

2.7. Expresión regular asociada a un AFD

- El método para obtener una expresión regular α 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 .

2.7. Expresión regular asociada a un AFD

□ **Ejemplo.** Sea el AF:



□ Ecuaciones características:

$$(1) \quad x_0 = 0x_0 + 1x_1 + 1$$

$$(2) \quad x_1 = 0x_0 + 1x_2$$

$$(3) \quad x_2 = 0x_2 + 1x_1 + 1$$

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

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

■ Sustituyendo en (2): $x_1 = 0x_0 + 1(0^*1x_1 + 0^*1) = 0x_0 + 10^*1x_1 + 10^*1$
 regla inferencia a $\mathbf{x_1} = 10^*1\mathbf{x_1} + 0x_0 + 10^*1 \Leftrightarrow \mathbf{x_1} = (10^*1)^* (0x_0 + 10^*1) =$
 $(10^*1)^*0x_0 + (10^*1)^*10^*1$

■ Sustituyendo en (1): $\mathbf{x_0} = 0x_0 + 1[(10^*1)^*0x_0 + (10^*1)^*10^*1] + 1 =$
 $0x_0 + 1(10^*1)^*0x_0 + 1(10^*1)^*10^*1 + 1 = (0 + 1(10^*1)^*0)x_0 + 1(10^*1)^*10^*1 + 1$
 regla inferencia a $\mathbf{x_0} = (0 + 1(10^*1)^*0)\mathbf{x_0} + 1(10^*1)^*10^*1 + 1 \Leftrightarrow$
 $\mathbf{x_0} = (0 + 1(10^*1)^*0)^* 1(10^*1)^*10^*1 + 1 = (0 + 1(10^*1)^*0)^* 1[(10^*1)^*(10^*1) + \lambda]$
 $\mathbf{x_0} = (\mathbf{0 + 1(10^*1)^*0})^* \mathbf{1(10^*1)^*}$ expresión regular que describe el lenguaje L(M).

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