

# Resumen Autómatas Finitos

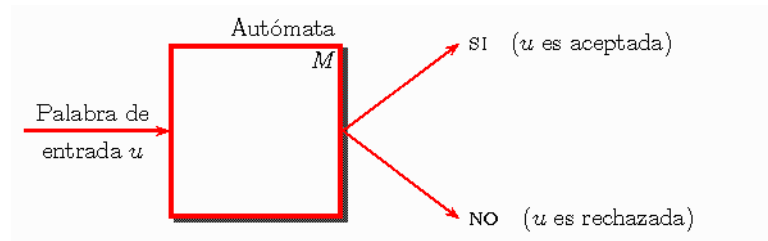
## Facultad de Ciencias UNAM \*

Favio E. Miranda Perea

A. Liliana Reyes Cabello

Lourdes González Huesca

Como hemos comentado, un autómatas es una representación abstracta de una máquina. Sobre todo, un autómatas finito es una máquina abstracta que procesa cadenas aceptándolas o rechazándolas.



Ahora estudiaremos los autómatas más sencillos que están en correspondencia con los lenguajes regulares ya expuestos.

## 1. Autómatas Finitos Deterministas

**Definición 1.1.** Un autómatas finito determinista (AFD) es una quintupla  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  donde

- $Q$  es un conjunto finito de estados.
- $\Sigma$  es el alfabeto de entrada.
- $\delta : Q \times \Sigma \rightarrow Q$  es la función de transición.
- $q_0 \in Q$  es el estado inicial.
- $F \subseteq Q$  es el conjunto de estados finales.

Los autómatas finitos definidos se denominan **autómatas finitos deterministas** ya que para cada estado  $q$  y para cada símbolo  $a \in \Sigma$ , la función de transición  $\delta(q, a)$  *siempre* está definida. Es decir, la función de transición  $\delta$  *determina unívocamente* la acción que el autómatas realiza cuando se encuentra en un estado  $q$  procesando un símbolo  $a$ .

---

\*Material para el curso de Autómatas y Lenguajes Formales, desarrollado bajo el proyecto UNAM-PAPIME PE102117 2017–2018.

**Definición 1.2.** Dado un autómata  $M$ , el lenguaje aceptado o reconocido por  $M$  se denota  $L(M)$  y se define por

$$L(M) := \{w \in \Sigma^* \mid M \text{ se detiene al procesar } w \text{ en un estado } q \in F\}$$

**Ejemplo:** Sean  $\Sigma = \{a, b\}$ ,  $Q = \{q_0, q_1, q_2\}$  donde  $q_0$  es el estado inicial y el conjunto  $F = \{q_0, q_2\}$  son los estados de aceptación. La función de transición  $\delta$  está definida por alguna de las siguientes:

$\delta$	$a$	$b$
$q_0$	$q_0$	$q_1$
$q_1$	$q_1$	$q_2$
$q_2$	$q_1$	$q_1$

$$\begin{aligned} \delta(q_0, a) &= q_0 & \delta(q_0, b) &= q_1 \\ \delta(q_1, a) &= q_1 & \delta(q_1, b) &= q_2 \\ \delta(q_2, a) &= q_1 & \delta(q_2, b) &= q_1 \end{aligned}$$

Del ejemplo anterior se pueden ver dos formas de presentar la función de transición: tabla de estados con símbolos o la definición estándar usada para funciones. También es posible dar una presentación visual del autómata mediante diagramas para representar estados y las transiciones entre ellos.

### 1.1. Diagrama de estados


Un autómata finito se puede representar por medio de un grafo dirigido y etiquetado. Recuerdese que un **grafo** es un conjunto de vértices o nodos unidos por aristas o conectores; si las aristas tienen tanto dirección como etiquetas, el grafo se denomina **grafo dirigido y etiquetado** o **digrafo etiquetado**.

Para el caso de los autómatas finitos, esta representación está modificada de la siguiente forma:

- Los vértices o nodos son los estados del autómata.

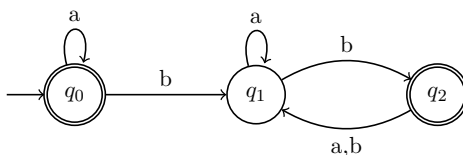
- Un estado  $q$  se representa por: 

- El estado inicial  $q_0$  se representa por: 

- Un estado final  $q$  se representa por: 

- La transición  $\delta = (q, a) = p$  se representa en la forma: 

**Ejemplo:** Veamos la representación gráfica del autómata del ejemplo anterior:



Para autómatas deterministas se adopta una y sólo una de las siguientes convenciones con respecto a los diagramas de estados:

- Como se dijo antes, la función de transición es **total**: cada estado tiene una transición por cada símbolo. Así se puede considerar la existencia de un **estado de error** para las cadenas que serán rechazadas
- Alternativamente se puede asumir que las aristas no dibujadas explícitamente conducen al estado de no-aceptación llamado **estado de error**.  
Es decir, en el diagrama de estados se indican únicamente las aristas que conduzcan a trayectorias de aceptación. Esto permite simplificar considerablemente los diagramas.

## 1.2. Autómatas y lenguajes

Los autómatas son máquinas que nos permiten reconocer lenguajes y la forma de hacerlo es a través de un procesamiento de cadenas que permite “recordar” algunas características de dichas cadenas. El diseño de estas máquinas es un problema importante a abordar:

**Problema:** Dado un lenguaje  $L \subseteq \Sigma^*$  diseñar un AFD  $M$  que acepte exactamente a  $L$ , es decir, tal que  $L(M) = L$ .

Una estrategia de ensayo y error es inconveniente ya que las posibilidades son demasiadas al considerar los símbolos del alfabeto dado y las características del lenguaje a detallar. Además se deben respetar las siguientes características para resolver el problema anterior:

- El autómata debe ser **completo**: debe aceptar todas las palabras de  $L$ , es decir que es necesario que  $L \subseteq L(M)$ .
- El autómata debe ser **correcto**: debe aceptar sólo palabras de  $L$ , es decir que es necesario que  $L(M) \subseteq L$ .

Esta doble contención asegurará que ambos lenguajes sean iguales y que el autómata acepte exactamente las cadenas del lenguaje.

Hay muchos problemas de interés sobre autómatas finitos, nos interesa resolver en este momento los siguientes:

- Dado un autómata  $M$  determinar formalmente el lenguaje aceptado por  $M$ .
- Dado un lenguaje regular  $L$  diseñar un autómata finito determinista  $M$  que acepte o reconozca a  $L$ , es decir, tal que  $L(M) = L$ .

Más adelante se demostrará que estos problemas *siempre* tienen solución. El primer paso para resolverlos es la formalización de la noción de procesamiento, *grosso modo* se debe abstraer la idea de que las cadenas serán “leídas” por el autómata y harán que éste se detenga en un estado. Si en el estado es final, las cadenas serán aceptadas y serán rechazadas en otro caso. Con esto se puede determinar cuál es el lenguaje que acepta el autómata.

## 2. Lenguaje de Aceptación

La idea de diseñar una máquina abstracta para el procesamiento de cadenas, está captado por el funcionamiento de un autómata. De manera informal un autómata  $M$  **reconoce** o acepta una cadena  $w$  si:

1. Se consumen todos los símbolos de  $w$  al comenzar en el estado inicial, generalmente el llamado  $q_0$ , siguiendo las transiciones de acuerdo a la función  $\delta$ .
2. Al terminar, el estado actual de la máquina es final.

El lenguaje aceptado es entonces el mismo que se describió al principio de esta nota mediante una notación por comprensión. Esta definición un tanto informal se puede hacer estricta mediante una extensión de la función de transición para denotar el procesamiento de una cadena en su totalidad.

**Definición 2.1.** Sea  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  un AFD. La función de transición  $\delta$  se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

llamada la función de transición extendida de  $M$  y es definida recursivamente como sigue:

- $\delta^*(q, \epsilon) = q$
- $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$   
O alternativamente  $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$ .

De la definición se sigue que  $\delta^*(q, w)$  devuelve el estado en el que la máquina **termina** al procesar  $w$ . Dado que  $\delta^*$  es una extensión de  $\delta$  es usual *sobrecargar* la operación y escribir  $\delta$  en lugar de  $\delta^*$ .

La definición informal de lenguaje de aceptación de un autómata puede ahora formalizarse haciendo referencia a la función de transición extendida:

**Definición 2.2.** El lenguaje de aceptación se define como:

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

De esta manera es posible verificar formalmente la completud y correctud del diseño de un autómata  $M$  para un lenguaje  $L$ :

- Correctud ( $L(M) \subseteq L$ ): Si  $\delta^*(q_0, w) \in F$  entonces  $w \in L$ .
- Completud ( $L \subseteq L(M)$ ): Si  $w \in L$  entonces  $\delta^*(q_0, w) \in F$ .

Debemos recordar que estas dos propiedades son indispensables para el diseño de una máquina que reconozca un lenguaje.

### 3. Diseño de autómatas finitos deterministas

La noción de un autómata como máquina abstracta con memoria nos proporciona una serie de estrategias a seguir para el diseño de autómatas:

- Se debe determinar explícitamente qué condición “recuerda” cada estado.
- La única memoria que tiene un AFD son los estados.
- Primero se propone un conjunto de estados que “recuerden” condiciones importantes.
- Después se proponen las transiciones para cambiar de un estado a otro.
- Finalmente se determina cuáles estados son finales, observando qué condiciones de estado corresponden con la definición del lenguaje aceptado.

A continuación se detallan algunas estrategias bien establecidas para obtener autómatas.

#### 3.1. Diseño por conjuntos de estados

Muchas veces es preferible generalizar condiciones para definir grupos de estados en lugar de dar condiciones estado por estado. De esta forma se disminuye la posibilidad de error y se facilita la solución de un problema complejo.

Las condiciones para grupos de estados deben ser:

- Excluyentes: un grupo de estados debe cumplir únicamente una condición.
- Comprensivas: los grupos cumplen todos los casos posibles.

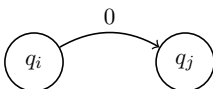
**Ejemplo:** Diseñar un AFD para

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Un análisis informal nos permite distinguir los grupos de estados según las siguientes condiciones y subcasos:

1. La palabra a ser consumida no contiene ni 00 ni 11:

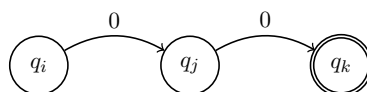
- no se han leído símbolos



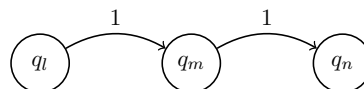
- se leyó un 0 (o se leyó un 1)

2. La palabra contiene 00 pero no 11:

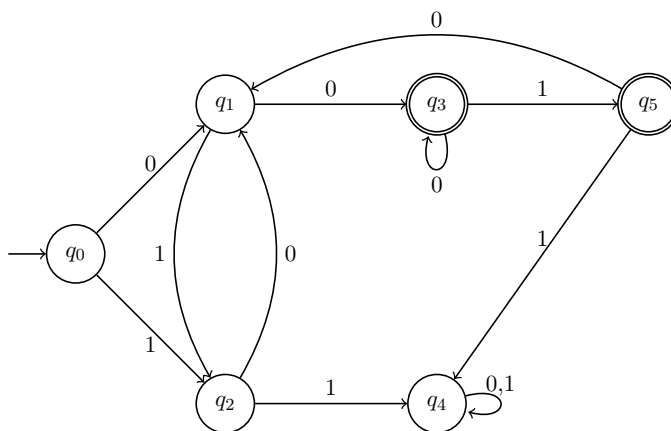
- se leyó otro 0 o se leyó un 1



3. La palabra contiene exactamente la subcadena 11.



4. Finalmente combinamos las partes anteriores y completamos las transiciones:



### 3.2. Diseño por complemento

Dado un lenguaje  $L$  a veces es más fácil diseñar un autómata para el complemento  $\bar{L} = \Sigma^* - L$ :

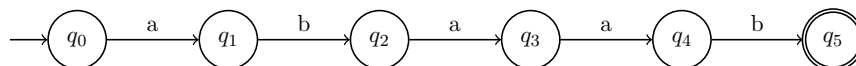
Si  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  reconoce a  $L$ , es decir  $L(M) = L$  entonces  $M^c = \langle Q, \Sigma, \delta, q_0, Q - F \rangle$  reconoce a  $\Sigma^* - L$ , es decir  $L(M^c) = \bar{L}$ .

Este método sólo funciona para autómatas deterministas.

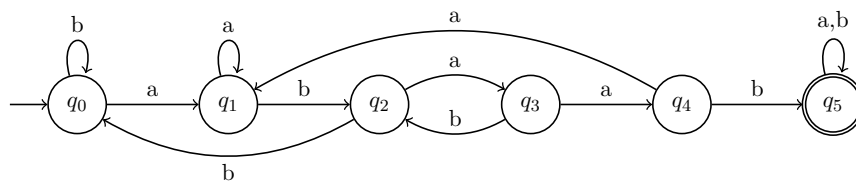
**Ejemplo:** Diseñar un AFD para

$$L = \{w \in \{a, b\}^* \mid w \text{ no contiene } abaab\}$$

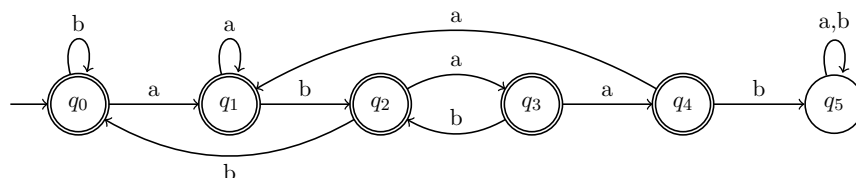
1. Primero se diseña la parte de  $M^c$  que solo acepta la cadena  $abaab$ :



2. Se completa el autómata anterior para reconocer cualquier cadena que contenga a  $abaab$ :



3. Se obtiene  $M$  del anterior en donde el estado que antes era final ahora es el estado de error:



### 3.3. Modularización

Algunas veces la especificación de un lenguaje  $L$  tiene una forma lógica que permite descomponerlo en lenguajes más sencillos. De esta manera el diseño de un AFD se puede modularizar.

- Por ejemplo si  $L = L_1 \cup L_2$  basta construir autómatas para  $L_1$  y  $L_2$  y ejecutarlos en paralelo.
- Lo mismo sucede si  $L = L_1 \cap L_2$  o si  $L = L_1 - L_2$ .

Veamos la definición formal para obtener los anteriores:

**Definición 3.1.** Suponer que  $M_1 = \langle Q_1, \Sigma, \delta_1, q_0, F_1 \rangle$  y  $M_2 = \langle Q_2, \Sigma, \delta_2, p_0, F_2 \rangle$  son dos autómatas que aceptan los lenguajes  $L_1$  y  $L_2$  respectivamente. Sea  $M = \langle Q, \Sigma, \delta, r_0, F \rangle$  un autómata donde  $Q = Q_1 \times Q_2$ , el estado inicial es  $r_0 = (q_0, p_0)$ , la función de transición se define como

$$\delta((q, p), a) = (\delta_1(q, a), \delta_2(p, a))$$

y los estados finales como:

1. Si  $F = \{(q, p) \mid q \in F_1 \text{ o } p \in F_2\}$  entonces  $M$  acepta el lenguaje  $L_1 \cup L_2$ .
2. Si  $F = \{(q, p) \mid q \in F_1 \text{ y } p \in F_2\}$  entonces  $M$  acepta el lenguaje  $L_1 \cap L_2$ .
3. Si  $F = \{(q, p) \mid q \in F_1 \text{ y } p \notin F_2\}$  entonces  $M$  acepta el lenguaje  $L_1 - L_2$ .

Una estrategia a seguir es calcular la tabla de transiciones para el nuevo autómata y renombrar los estados nuevos. Así mismo, si existen estados que no están conectados se pueden eliminar.

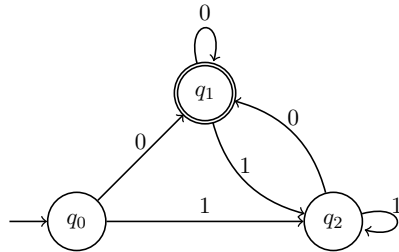
**Ejemplo:** Sea  $L = \{w \in \{0, 1\}^* \mid w \text{ acaba en } 0 \text{ y } 11 \text{ no es subcadena de } w\}$ . Para obtener un autómata que reconozca el lenguaje se descompone como  $L = L_1 - \overline{L_2}$  donde

$$L_1 = \{w \in \{0, 1\}^* \mid w \text{ acaba en } 0\}$$

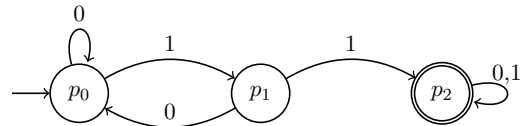
$$L_2 = \{w \in \{0, 1\}^* \mid 11 \text{ es subcadena de } w\}$$

A continuación se construyen los autómatas para reconocer los respectivos lenguajes y después se obtiene  $M$  usando la definición anterior, se proporciona la tabla de transiciones.

$M_1$  :



$M_2$  :



$M :$	$\delta_1 \times \delta_2$	0	1	renombre	
	$(q_0, p_0)$	$(q_1, p_0)$	$(q_2, p_1)$	$r_0$	edo. inicial
	$(q_0, p_1)$	$(q_1, p_0)$	$(q_2, p_2)$	$r_1$	edo. no conectado
	$(q_0, p_2)$	$(q_1, p_2)$	$(q_2, p_2)$	$r_2$	edo. no conectado
	$(q_1, p_0)$	$(q_1, p_0)$	$(q_2, p_1)$	$r_3$	edo. final
	$(q_1, p_1)$	$(q_1, p_0)$	$(q_2, p_2)$	$r_4$	edo. final pero no conectado
	$(q_1, p_2)$	$(q_1, p_2)$	$(q_2, p_2)$	$r_5$	
	$(q_2, p_0)$	$(q_1, p_0)$	$(q_2, p_1)$	$r_6$	edo. no conectado
	$(q_2, p_1)$	$(q_1, p_0)$	$(q_2, p_2)$	$r_7$	
	$(q_2, p_2)$	$(q_1, p_2)$	$(q_2, p_2)$	$r_8$	

## 4. Autómatas No-Deterministas

El determinismo de un autómata, deseable desde el punto de vista teórico, puede provocar complicaciones en la práctica. Veamos las diferencias entre estos conceptos:

- **Determinismo:** dado un estado  $q$  y un símbolo  $a$  existe una única transición  $\delta(q, a) = p$ , es decir  $\delta$  es una función total.
- **No-determinismo:** no hay una transición única al leer un símbolo  $a$  en un estado dado  $q$ .

El no-determinismo se traduce en que hay más de una transición al leer un símbolo, es decir,  $\delta(q, a)$  deja de ser función. O bien no hay transición, es decir,  $\delta(q, a)$  no está definida ( $\delta$  se vuelve función parcial). Sin embargo la máquina funciona únicamente al leer un símbolo y podemos decir que existe el no-determinismo sin lectura de símbolos.

Veamos cómo se puede agregar el no-determinismo a la definición de autómata que vimos anteriormente:

**Definición 4.1.** *Un autómata finito **no** determinista (AFN) es una quintupla  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  donde*

- $Q$  es un conjunto finito de estados.
- $\Sigma$  es el alfabeto de entrada.
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$  es la función de transición.
- $q_0 \in Q$  es el estado inicial.
- $F \subseteq Q$  es el conjunto de estados finales.

Obsérvese que la imagen de  $\delta$  es ahora un elemento de  $\mathcal{P}(Q)$ , es decir es un subconjunto de estados de  $Q$ . Además  $\delta(q, a) = \{q_1, q_2, \dots, q_n\}$  indica que al leer el símbolo  $a$  en el estado  $q$  la máquina puede pasar a cualquiera de los estados  $q_1, \dots, q_n$ . Si  $\delta(q, a) = \emptyset$  entonces no hay transición posible desde el estado  $q$  al leer  $a$ , es decir, la máquina está bloqueada.

Veamos como las nociones vistas para los AFD se modifican:

**Definición 4.2.** *Sea  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  un AFN. La función de transición  $\delta$  se extiende a cadenas mediante una función  $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$  definida recursivamente como sigue:*



- $\delta^*(q, \epsilon) = q$
- $\delta^*(q, wa) = \bigcup_{p \in \delta^*(q, w)} \delta(p, a)$   
 Alternativamente:  $\delta^*(q, aw) = \bigcup_{p \in \delta(q, a)} \delta^*(p, w)$

El lenguaje de aceptación se define mediante  $\delta^*$  como sigue:

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset\}$$

Es decir,  $w \in L(M)$  si y sólo si existe al menos un cómputo o un procesamiento de  $w$  que conduce a un estado final al iniciar la máquina en  $q_0$ .

#### 4.1. Eliminación del no-determinismo

Una pregunta natural es comparar dos autómatas, uno determinista y el otro no. Así también buscar una forma de eliminar el no-determinismo y conservar una máquina que acepte un lenguaje dado.

Todo AFD es a la vez un AFN con la particularidad de que  $\delta(p, a)$  consta de un único estado. La idea para transformar un AFN en un AFD es considerar a cada conjunto de estados  $\delta(p, a)$  del AFN como un único estado del nuevo AFD. A este método se conoce como la construcción de subconjuntos.

**Definición 4.3.** Dado un AFN  $M = \langle Q, \Sigma, \delta_N, q_0, F \rangle$  definimos un AFD  $M^d = \langle Q^d, \Sigma, \delta, q_0^d, F^d \rangle$  como sigue:

- $Q^d = \mathcal{P}(Q)$
- $\delta(S, a) = \delta_N(S, a) = \bigcup_{q \in S} \delta_N(q, a)$
- $q_0^d = \{q_0\}$
- $F^d = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$

Ambos autómatas son equivalentes, es decir,  $L(M) = L(M^d)$ .

### 5. AFN con $\epsilon$ -transiciones

Otra de las máquinas que son útiles para procesar cadenas son las que permiten procesar cadenas vacías, no sólo al tener como estado final a  $q_0$  sino que permiten procesar (sub)cadenas vacías en una parte intermedia. De esta forma se definen los autómatas con transiciones etiquetadas con la cadena  $\epsilon$ :

**Definición 5.1.** Un autómata finito **no** determinista con  $\epsilon$ -transiciones (AFN $\epsilon$ ) es una quintupla  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  donde

- $Q$  es un conjunto finito de estados.
- $\Sigma$  es el alfabeto de entrada.
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$  es la función de transición.

- $q_0 \in Q$  es el estado inicial.
- $F \subseteq Q$  es el conjunto de estados finales.

De la definición de  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$  se observa que la única diferencia está en el dominio de  $\delta$ . Es decir que las transiciones de la forma  $\delta(q, \varepsilon) = a$  están permitidas e indican que la máquina puede cambiar de estado sin leer ningún símbolo. Esto causa también un no-determinismo, que es aun más complicado de modelar matemáticamente pero tiene grandes ventajas:

- Se permiten múltiples cálculos para una cadena de entrada.
- Pueden existir cálculos bloqueados.
- A diferencia de los AFD y AFN simples, pueden existir cálculos infinitos, es decir, surge la **no-terminación**.
- La presencia de  $\varepsilon$ -transiciones permite mayor libertad en el diseño.

Para extender la definición de transiciones a procesamiento de cadenas es necesario introducir un concepto previo que considera los estados a los que la máquina puede llegar al incluir las cadenas vacías.

**Definición 5.2.** Dado un estado  $q$ , definimos la  $\varepsilon$ -cerradura de  $q$  como el conjunto de estados alcanzables desde  $q$  mediante cero o más  $\varepsilon$ -transiciones. Es decir

$$Cl_\varepsilon(q) = \{s \in Q \mid \exists p_1, \dots, p_n \text{ con } p_1 = q, p_n = s, p_i \in \delta(p_{i-1}, \varepsilon)\}$$

Recursivamente:

- $q \in Cl_\varepsilon(q)$
- Si  $r \in Cl_\varepsilon(q)$  y  $\delta(r, \varepsilon) = s$  entonces  $s \in Cl_\varepsilon(q)$

Esta definición se extiende a conjuntos de estados como sigue:

$$Cl_\varepsilon(S) = \bigcup_{q \in S} Cl_\varepsilon(q)$$

Con la  $\varepsilon$ -cerradura de estados se puede definir la extensión de  $\delta$ :

**Definición 5.3.** Sea  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  un AFN $\varepsilon$ . La función de transición  $\delta$  se extiende a cadenas mediante una función

$$\delta^* : Q \times (\Sigma^* \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$$

definida recursivamente como sigue:

- $\delta^*(q, \varepsilon) = Cl_\varepsilon(q)$
- $\delta^*(q, wa) = Cl_\varepsilon\left(\bigcup_{q' \in \delta^*(q, w)} \delta(q', a)\right)$

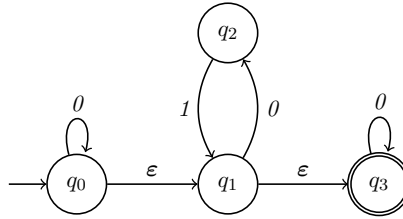
### 5.1. Eliminación de $\varepsilon$ -transiciones

Estudiaremos el proceso de eliminación de transiciones  $\varepsilon$ . Esta eliminación implica que un AFN es también un AFN $\varepsilon$ . Es decir que cualquier AFN $\varepsilon$  es inmediatamente un AFN con la particularidad de que no existen  $\varepsilon$ -transiciones.

**Definición 5.4.** Dado un AFN $\varepsilon$ ,  $M_1 = \langle Q_1, \Sigma, \delta_1, q_0, F_1 \rangle$  existe un AFN equivalente  $M = \langle Q, \Sigma, \delta, p_0, F \rangle$ , definido mediante:

- $Q := Q_1$
- $p_0 := q_0$
- $\delta(q, a) = \delta_1^*(q, a) = Cl_\varepsilon\left(\bigcup_{p \in Cl_\varepsilon(q)} \delta_1(p, a)\right)$
- $F := F_1 \cup \{q_0\}$  si  $Cl_\varepsilon(q_0) \cap F_1 \neq \emptyset$   
 $F := F_1$  en caso contrario.

**Ejemplo:** Consideremos el siguiente autómata:



Para eliminar las  $\varepsilon$ -transiciones calculamos las cerraduras de cada estado:

	$\delta$	0	1	$\varepsilon$	$Cl_\varepsilon$
inicial	$q_0$	$\{q_0\}$	$\emptyset$	$\{q_1\}$	$\{q_0, q_1, q_3\}$
	$q_1$	$\{q_2\}$	$\emptyset$	$\{q_3\}$	$\{q_1, q_3\}$
	$q_2$	$\emptyset$	$\{q_1\}$	$\emptyset$	$\{q_2\}$
final	$q_3$	$\{q_3\}$	$\emptyset$	$\emptyset$	$\{q_3\}$

Y calculamos la nueva función de transición:

$\delta'$	0	1
inicial, final $q_0$	$\delta^*(q_0, 0) = Cl_\varepsilon\left(\bigcup_{p \in Cl_\varepsilon(q_0)} \delta(p, 0)\right) =$ $Cl_\varepsilon\left(\bigcup_{p \in \{q_0, q_1, q_3\}} \delta(p, 0)\right) =$ $Cl_\varepsilon(\{q_0\} \cup \{q_2\} \cup \{q_3\}) =$ $Cl_\varepsilon(\{q_0\}) \cup Cl_\varepsilon(\{q_2\}) \cup Cl_\varepsilon(\{q_3\}) = \{q_0, q_1, q_3, q_2\}$	$\emptyset$
$q_1$	$Cl_\varepsilon\left(\bigcup_{p \in Cl_\varepsilon(q_1)} \delta(p, 0)\right) =$ $Cl_\varepsilon\left(\bigcup_{p \in \{q_1, q_3\}} \delta(p, 0)\right) = Cl_\varepsilon(\{q_2\} \cup \{q_3\}) = \{q_2, q_3\}$	$\emptyset$
$q_2$	$\emptyset$	$\{q_1, q_3\}$
final $q_3$	$\{q_3\}$	$\emptyset$

Ahora  $q_0$  es final dado que  $\{q_3\} \cap Cl_\varepsilon(q_0) = \{q_0, q_1, q_2, q_3\} \neq \emptyset$ , es decir que los finales del autómata original aparecen en la cerradura- $\varepsilon$  del estado  $q_0$  (se acepta a  $\varepsilon$ ).

## 6. Equivalencias

Nuevamente surgen preguntas como la equivalencia entre autómatas deterministas, no deterministas y con transiciones  $\epsilon$ .

En la subsección 4.1 se estudió la forma de eliminar el no-determinismo de un autómata, creando un AFD. Es decir  $AFN \Rightarrow AFD$ .

En la subsección pasada se revisó la forma de eliminar  $\epsilon$ -transiciones y de esta forma obtener un AFN. Así también se observó que un AFN es exactamente un  $AFN\epsilon$  ya que incluye el no-determinismo pero sin transiciones de la cadena vacía.

Esta equivalencia,  $AFN \Leftrightarrow AFN\epsilon$ , cierra el ciclo de equivalencias de autómatas finitos. Cualquier tipo de autómata finito puede convertirse en un AFD y viceversa, es decir:

$$AFD \Leftrightarrow AFN \Leftrightarrow AFN\epsilon$$

Nuestra siguiente meta es probar el Teorema de Kleene, el cual es uno de los resultados más importantes en la Teoría de la Computación pues asegura la equivalencia entre dos de nuestros tres conceptos fundamentales: los autómatas finitos y los lenguajes regulares.

## 7. Teorema de Kleene

Como hemos visto, la relación entre autómatas finitos y lenguajes regulares es muy cercana y las nociones detrás de ambos conceptos estudiadas hasta ahora hacen parecer que esta relación es de equivalencia. En esta nota demostraremos que ese es el caso utilizando el método propuesto por Kleene <sup>1</sup>.

**Teorema 7.1.** *Un lenguaje es regular si y sólo si es aceptado por un autómata finito.*

*Demostración.* La prueba es en dos partes:

- I Síntesis: Dado un lenguaje regular  $L$ , existe un autómata finito  $M$  tal que  $L = L(M)$ .
- II Análisis: Dado un autómata finito  $M$ , existe una expresión regular  $\alpha$  tal que  $L(M) = L(\alpha)$ . Es decir,  $L(M)$  es regular.

□

A continuación se abordarán dos teoremas que demuestran al anterior. Para ello recordemos que las expresiones regulares están en correspondencia con los lenguajes regulares y de esta forma podremos empatar también a las expresiones regulares con los autómatas finitos.

### 7.1. Teorema de Síntesis de Kleene

En esta sección se demostrará una parte de la doble implicación del teorema de Kleene: se pasará de expresiones regulares a autómatas finitos mediante un teorema que se denomina de síntesis ya que se proporcionará una máquina para reconocer un lenguaje regular dado. Es decir, se sintetizará un autómata finito analizando la forma de una expresión regular que genera al lenguaje dado.

---

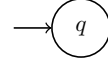
<sup>1</sup>Stephen Cole Kleene fue un matemático estadounidense, alumno de Alonzo Church. También es conocido por iniciar la teoría de la recursión que fue usada para los fundamentos de la Teoría de la Computación como la noción de computabilidad.

**Teorema 7.2.** Dada una expresión regular  $\alpha$  existe un autómata finito  $M$  tal que  $L(\alpha) = L(M)$ .

*Demostración.* La demostración es *constructiva* y se hará mediante inducción sobre las expresiones regulares, es decir proporcionando un autómata que *reconozca* cada caso de una expresión regular.

### Base de la Inducción

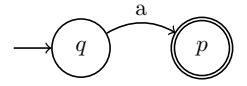
- El caso en que  $\alpha = \emptyset$ , el siguiente autómata reconoce a  $L(\alpha)$ :



- Caso en que  $\alpha = \varepsilon$ . Entonces el siguiente autómata reconoce a  $L(\alpha)$ :



- Para  $\alpha = a$  y  $a \in \Sigma$  se tiene el siguiente autómata que reconoce a  $L(\alpha)$ :



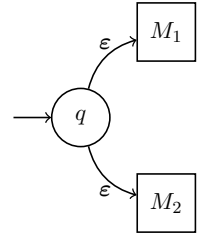
### Hipótesis de inducción

Sean  $M_1, M_2$  dos autómatas que reconocen los lenguajes  $L(\alpha_1)$  y  $L(\alpha_2)$  respectivamente.

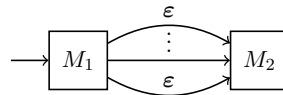
### Paso Inductivo

- Caso en que  $\alpha = \alpha_1 + \alpha_2$ . El siguiente autómata reconoce a  $L(\alpha)$ :

donde  $M_1, M_2$  son autómatas dados por la hipótesis de inducción y las transiciones  $\varepsilon$  van hacia los estados iniciales de cada uno de ellos. Los estados finales de cada autómata se conservan como finales.

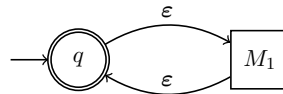


- Para el caso en que  $\alpha = \alpha_1 \alpha_2$  entonces el siguiente autómata reconoce a  $L(\alpha)$ :



donde  $M_1, M_2$  son autómatas que reconocen a  $L(\alpha_1), L(\alpha_2)$  dados por la hipótesis de inducción. El estado inicial es el de  $M_1$  y las transiciones  $\varepsilon$  van de los estados finales de  $M_1$  hacia el inicial en  $M_2$ . Los estados finales sólo son aquellos de  $M_2$ .

- Finalmente el caso  $\alpha = \alpha_1^*$  tiene al siguiente autómata que reconoce a  $L(\alpha)$ :



donde  $M_1$  es un autómata que reconoce a  $L(\alpha_1)$  dado por la hipótesis de inducción y las transiciones  $\varepsilon$  conectan el estado inicial y los finales con el nuevo estado  $q$ . Se pueden dejar los finales de  $M_1$  como finales del nuevo autómata.

□

## 7.2. Minimización de autómatas

El proceso constructivo anterior nos lleva a tener máquinas demasiado grandes, con  $\varepsilon$ -transiciones. En notas anteriores hemos revisado las equivalencias entre autómatas y sólo resta tener una forma de minimizar máquinas. Por lo cual presentamos a continuación los métodos para reducir el tamaño de autómatas. Estudiaremos la forma en que se pueden minimizar los autómatas para obtener máquinas más eficientes.

### 7.2.1. Eliminación de Estados Inaccesibles

Sea  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  un AFD. Decimos que un estado  $q \in Q$  es accesible si y sólo si existe  $w \in \Sigma^*$  tal que  $\delta^*(q_0, w) = q$ . Es decir,  $q$  es accesible si y sólo si el procesamiento de alguna cadena termina en el estado  $q$ .

El conjunto de estados accesibles de un autómata  $M$  se denota  $\text{Acc}(M)$ . Si un estado no es accesible decimos que es inaccesible.

Es claro que el conjunto  $\text{Acc}(M)$  puede construirse de manera algorítmica, por ejemplo como sigue:

```
A_N := {q_0} % estados accesibles
A_V := ∅      % estados verificados
while A_N ≠ A_V do
    A_V := A_N
    A_N := A_N ∪ {q ∈ Q | δ(p, a) = q, a ∈ Σ, p ∈ A_N}
return A_N
```

Los estados inaccesibles en un autómata son inútiles y pueden ser eliminados sin afectar el lenguaje de aceptación como vemos a continuación:

**Proposición 7.1.** Dado  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  un AFN, existe un AFD  $M' = \langle Q', \Sigma, \delta', q_0, F' \rangle$  equivalente a  $M$  que contiene únicamente a los estados accesibles de  $M$ , es decir,  $Q' = \text{Acc}(M)$  y por lo tanto no contiene estados inaccesibles. Para lo anterior basta definir  $M'$  como sigue:

- $Q' = \text{Acc}(M)$
- $\delta' = \delta|_{Q'}$
- $F' = F \cap Q'$

La prueba de la equivalencia  $L(M) = L(M')$  es inmediata y se deja como ejercicio. Debido a este resultado, de ahora en adelante podemos suponer que un autómata no tiene estados inaccesibles.

### 7.2.2. Equivalencia de estados y el Autómata cociente

Puede ser el caso que unas partes de un autómata sean redundantes, es decir que las cadenas que son aceptadas por una parte del autómata también pueden ser procesadas y aceptadas por otra parte. Veamos cómo abstraer y generalizar partes de los autómatas para minimizar el número de estados sin afectar al lenguaje de aceptación.

**Definición 7.1.** Decimos que dos estados  $q, q' \in Q$  de un AFD son equivalentes  $q \equiv q'$  si y sólo si:

$$\forall w \in \Sigma^* (\delta^*(q, w) \in F \Leftrightarrow \delta^*(q', w) \in F)$$

Es decir, si  $\delta^*(q, w), \delta^*(q', w)$  son ambos finales o ambos no finales.

La relación  $\equiv$  entre estados es una relación de equivalencia, es decir cumple lo siguiente:

- Reflexividad:  $q \equiv q$ .
- Simetría: si  $q \equiv q'$  entonces  $q' \equiv q$ .
- Transitividad: si  $q \equiv q'$  y  $q' \equiv q''$  entonces  $q \equiv q''$ .

Adicionalmente la función de transición  $\delta$  es compatible con  $\equiv$ , en el siguiente sentido:

$$\text{Si } q \equiv q' \text{ entonces } \forall a \in \Sigma (\delta(q, a) \equiv \delta(q', a))$$

La relación de equivalencia  $\equiv$  genera una **partición** del conjunto de estados dada por las clases de equivalencia de cada estado definidas como:

$$[q] := \{p \in Q \mid q \equiv p\}$$

Es decir, los conjuntos de estados  $[q]$  cumplen lo siguiente:

- $\forall q \in Q ([q] \neq \emptyset)$  (no hay clases vacías).
- $\forall p, q \in Q ([q] = [p] \text{ ó } [q] \cap [p] = \emptyset)$  (las clases son ajenas).
- $\bigcup_{q \in Q} [q] = Q$  (la unión de todas las clases es el conjunto de estados original).

Al agrupar por clases de equivalencia a los estados de un autómata, se puede calcular otro autómata llamado **autómata cociente** que tiene un número mínimo de estados.

**Definición 7.2.** Dado un AFD  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  existe el autómata cociente  $M/\equiv$  también conocido como  $M^{min}$  que es la minimización de  $M$  y se define como  $M^{min} = \langle Q_m, \Sigma, \delta_m, [q_0], F_m \rangle$  donde:

- $Q_m := \{[q] \mid q \in Q\}$  los estados son las clases de equivalencia
- la clase  $[q_0]$  es el estado inicial.
- $F_m := \{[q] \mid q \in F\}$
- $\delta_m : Q_m \times \Sigma \rightarrow Q_m$  se define como  $\delta_m([q], a) = [\delta(q, a)]$

La definición anterior indica que dado un AFD  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ , el autómata cociente  $M/\equiv$  es el autómata mínimo equivalente a  $M$ . Es decir, aceptan el mismo lenguaje  $L(M) = L(M/\equiv)$  y no existe un autómata equivalente a  $M$  con menos estados que  $M/\equiv$ .

La equivalencia entre  $M$  y  $M^{min}$  se sigue de la siguiente propiedad:

**Lema 7.1.** Sean  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  un AFD y  $M^{min}$  su autómata cociente. Para cualesquiera  $q \in Q, w \in \Sigma^*$  se cumple  $\delta_m^*([q], w) = [\delta^*(q, w)]$

*Demostración.* La prueba es por inducción sobre  $w$ . □

**k-equivalencia** La demostración anterior requiere de una relación de equivalencia que depende de la longitud de una cadena.

Definimos la relación de k-equivalencia para cualquier  $k \in \mathbb{N}$  como sigue:

$$\forall w \in \Sigma^*, |w| \leq k \rightarrow (\delta^*(q, w) \in F \Leftrightarrow \delta^*(q', w) \in F)$$

Es decir, para cualquier cadena  $w$  de longitud menor o igual que  $k$ , los estados  $\delta^*(q, w)$  y  $\delta^*(q', w)$  son ambos finales o ambos no finales.

Así  $\equiv_k$  es una relación de equivalencia cuyas clases se denotan con  $[q]_k$ , es decir

$$[q]_k = \{p \in Q \mid q \equiv_k p\}$$

La relación de k-equivalencia cumple las siguientes propiedades:

**P1**  $q \equiv q'$  si y sólo si  $\forall k \in \mathbb{N} (q \equiv_k q')$ .

**P2**  $q \equiv_0 q'$  si y sólo si  $q, q' \in F$  ó  $q, q' \in Q - F$ .

**P3**  $[q]_0 = F$  si y sólo si  $q \in F$ .

**P4** Si  $q \equiv_k q'$  entonces  $q \equiv_{k-1} q'$ .

**P5**  $[q]_k \subseteq [q]_{k-1}$

**P6** Si  $q \equiv_k q'$  entonces  $\forall a \in \Sigma (\delta(q, a) \equiv_{k-1} \delta(q', a))$

**P7**  $q \equiv_k q'$  si y sólo si  $q \equiv_{k-1} q'$  y  $\forall a \in \Sigma (\delta(q, a) \equiv_{k-1} \delta(q', a))$

**P8** Sea  $P_k = \{[q]_k \mid q \in Q\}$  la partición dada por la relación  $\equiv_k$  para cualquier  $k \in \mathbb{N}$ .

Si  $P_k = P_{k-1}$  para alguna  $k$  entonces  $P_k = P_m$  para toda  $m \geq k$ .

Con las definiciones anteriores podemos construir el autómata mínimo equivalente:

**Definición 7.3.** Dado un AFD  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  el AFD mínimo asociado puede construirse como se describe con el siguiente pseudocódigo:

```

Q := Acc(M)      % estados accesibles

P_0 := {F, Q - F} % construir particion inicial: edos finales y no-finales

k := 0
repeat {
  k := k + 1
  P_k := {q ∈ P_{k-1} | ∀ a ∈ Σ, [δ(q, a)] = [δ(p, a)]}
until P_k = P_{k-1}

return P_k

```

La partición  $P_k$  se construirá a partir de  $P_{k-1}$  manteniendo a dos estados  $q, p$  en la misma clase si y sólo si para toda  $a \in \Sigma$ , los estados  $\delta(q, a)$  y  $\delta(p, a)$  estaban en la misma clase en  $P_{k-1}$ .

Es decir que  $P_k$  es la partición generada por  $\equiv$ :  $P_k = Q / \equiv = \{[q] \mid q \in Q\}$ .

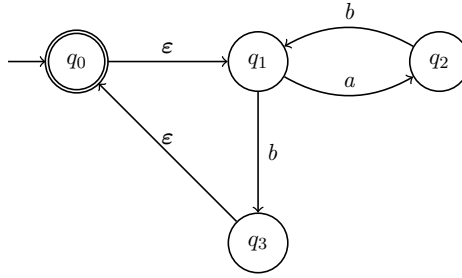


La definición anterior está dada a través de un algoritmo el cual es correcto respecto a la especificación, ya que es consecuencia de la siguiente propiedad:

**Proposición 7.2.** Si  $M$  es un AFD entonces la sucesión de particiones  $P_0, P_1, \dots, P_k$  generadas por las clases de  $k$ -equivalencia de estados se estaciona, es decir existe un  $n \in \mathbb{N}$  tal que para toda  $k \geq n$  se tiene que  $P_k = P_n$ . Más aún  $n \leq |Q|$ , es decir  $n$  es a lo más el número de estados de  $M$ .

## Ejemplo

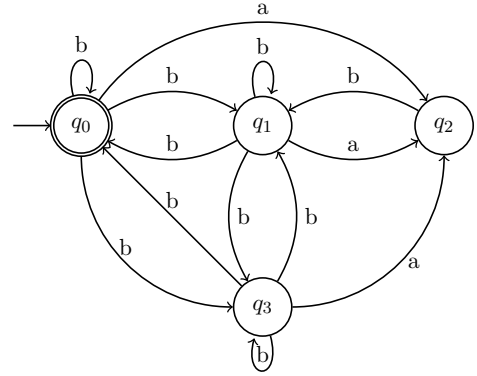
Sea  $M$  el siguiente autómata finito no-determinístico con transiciones  $\varepsilon$ :



Usando los métodos descritos se obtendrá un autómata mínimo determinista de la siguiente forma:

1. Se eliminan las  $\varepsilon$ -transiciones calculando los conjuntos  $Cl_\varepsilon$  de cada estado:

$Q$	$Cl_\varepsilon$	$Q$	$a$	$b$
$q_0$	$\{q_0, q_1\}$	$q_0$	$\{q_2\}$	$\{q_3, q_0, q_1\}$
$q_1$	$\{q_1\}$	$q_1$	$\{q_2\}$	$\{q_3, q_0, q_1\}$
$q_2$	$\{q_2\}$	$q_2$	$\emptyset$	$\{q_1\}$
$q_3$	$\{q_3, q_0, q_1\}$	$q_3$	$\{q_2\}$	$\{q_3, q_0, q_1\}$

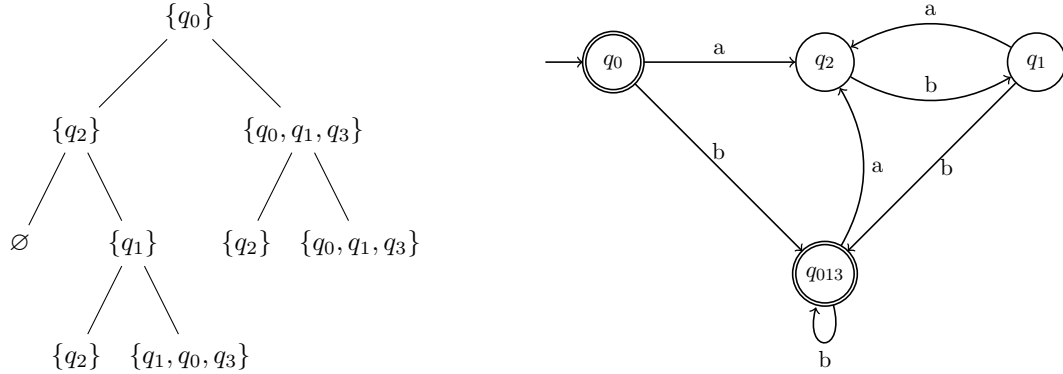


2. Se transforma el autómata anterior en determinista:

3. Finalmente se minimiza.

La primer partición está dada por  $A$  (los estados finales) y  $B$  (los no finales) separando los estados del autómata anterior:

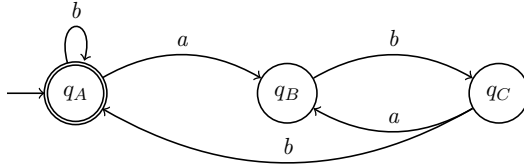
	$A$		$B$	
	$\{q_0\}$	$\{q_0, q_1, q_3\}$	$\{q_2\}$	$\{q_1\}$
$a$	$B$	$B$	$--$	$B$
$b$	$A$	$A$	$B$	$A$



Después se separa la partición  $B$ :

	$A$		$B$	$C$
	$\{q_0\}$	$\{q_0, q_1, q_3\}$	$\{q_2\}$	$\{q_1\}$
$a$	$B$	$B$	$--$	$B$
$b$	$A$	$A$	$C$	$A$

Para obtener:



### 7.3. Teorema de Análisis de Kleene

Ahora demostraremos la segunda parte en donde un autómata finito implica una expresión regular. La noción que está detrás de este teorema es que se analizarán los lenguajes acumulados de cada estado para generar una expresión regular.

**Teorema 7.3.** Dado un autómata finito  $M$  existe una expresión regular  $\alpha$  tal que  $L(M) = L(\alpha)$ . Es decir,  $L(M)$  es regular.

*Demostración.* Existen diversas demostraciones, nosotros usaremos el método de ecuaciones características usando el Lema de Arden que describiremos a continuación.  $\square$

#### 7.3.1. Lema de Arden

Este lema extrae un conjunto de ecuaciones para determinar el lenguaje de aceptación de una máquina. Primero veamos la definición de dichas ecuaciones y después el método para obtener las ecuaciones dado un autómata.

**Definición 7.4.** Sean  $A, B \subseteq \Sigma^*$  y  $X$  una variable:

- Una ecuación lineal derecha para  $X$  es una expresión de la forma:  $X = AX + B$

- Una ecuación lineal izquierda para  $X$  es una expresión de la forma:  $X = XA + B$

donde el símbolo  $+$  denota a la unión de lenguajes.

**Lema 7.2.: Lema de Arden** Sean  $A, B \subseteq \Sigma^*$  dos lenguajes y  $X = AX + B$  una ecuación lineal derecha. Entonces

1.  $A^*B$  es una solución de la ecuación, es decir,  $A^*B = A(A^*B) + B$ .
2. Si  $C$  es otra solución entonces  $A^*B \subseteq C$ , es decir,  $A^*B$  es la solución mínima.
3. Si  $\epsilon \notin A$  entonces  $A^*B$  es la única solución.

Esta parte mostrará que un autómata finito implica una expresión regular, esto se hará por medio de un sistema de ecuaciones a partir de un AFN, para ello se abstraerá la noción del lenguaje aceptado desde un estado particular del autómata y no necesariamente del inicial.

**Definición 7.5.** Dado un AFN  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  tal que  $Q = \{q_0, \dots, q_n\}$ . Definimos los siguientes conjuntos:

- El conjunto de cadenas que se aceptan desde el estado  $q_i$ , para cualquier  $1 \leq i \leq n$ :

$$L_i = \{w \in \Sigma^* \mid \delta^*(q_i, w) \cap F \neq \emptyset\}$$

- $L_0$  es el lenguaje aceptado por  $M$ , es decir,  $L_0 = L(M)$ .

En general no es sencillo calcular directamente los conjuntos  $L_i$ . Para obtener una expresión regular completa respecto al autómata, se obtendrá un sistema de ecuaciones a partir de un AFN. Al resolverlo,  $L_0$  será el lenguaje que reconoce el autómata como se mencionó arriba. El sistema de ecuaciones se define usando:

1. El conjunto de símbolos de  $\Sigma$  tal que existe una transición del estado  $q_i$  al estado  $q_j$ , para cualesquiera  $1 \leq i, j \leq n$ , con  $n$  el total de estados:

$$X_{i,j} = \{a \in \Sigma \mid q_j \in \delta(q_i, a)\}$$

2. El conjunto auxiliar  $Y_i$  que indica si  $\epsilon$  es aceptada desde  $q_i$

$$Y_i = \begin{cases} \{\epsilon\} & \text{si } q_i \in F \\ \emptyset & \text{en otro caso} \end{cases}$$

Por tanto, las ecuaciones de los lenguajes de cada estado están dadas por la siguiente propiedad que es fácil de demostrar para cualquier  $1 \leq i \leq n$ :

$$L_i = \sum_{j=0}^n X_{i,j} L_j + Y_i$$

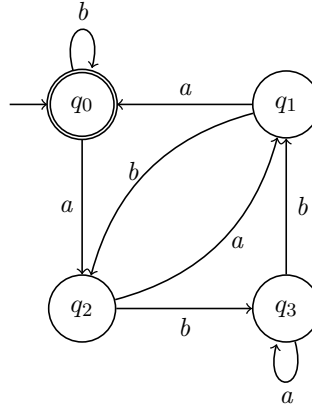
Dicha propiedad genera el llamado sistema de ecuaciones características de un AFN.

Finalmente, el Lema de Arden nos indica cómo calcular las ecuaciones  $L_i$ . Esta será la idea a seguir para demostrar el Teorema de Análisis de Kleene:

1. Dado el autómata  $M$  construir los conjuntos  $X_{i,j}$ ,  $Y_i$  para cada estado  $q_i$  y obtener las ecuaciones correspondientes  $L_i$ .
2. Resolver el sistema de ecuaciones características mediante el Lema de Arden.
3. La solución para  $L_0$  genera una expresión regular para  $L(M)$ .

## Ejemplo

Sea  $M$  el siguiente autómata finito determinista:



Calculemos los conjuntos y ecuaciones:

$i$	$X_{i,j}$	$Y_i$	$L_i$
0	$\{a, b\}$	$\{\epsilon\}$	$aL_2 + bL_0 + \epsilon$
1	$\{a, b\}$	$\emptyset$	$aL_0 + bL_2$
2	$\{a, b\}$	$\emptyset$	$aL_1 + bL_3$
3	$\{a, b\}$	$\emptyset$	$aL_3 + bL_1$

Ahora resolvamos las ecuaciones:

1. Comenzamos por la más sencilla, es decir la que se puede resolver usando el Lema de Arden:

$$L_3 = aL_3 + bL_1 \quad L_3 = a^*bL_1$$

2. Utilizamos el resultado anterior para sustituir en la ecuación de  $L_2$ :

$$L_2 = aL_1 + bL_3 = aL_1 + ba^*bL_1$$

Se puede factorizar  $L_1$  para obtener

$$L_2 = (a + ba^*b)L_1$$

3. Ahora podemos sustituir el resultado anterior en  $L_1$

$$L_1 = aL_0 + bL_2 = aL_0 + b(a + ba^*b)L_1$$

Y resolviendo con el lema de Arden:

$$L_1 = (b(a + ba^*b))^*aL_0$$

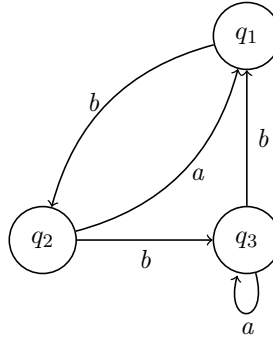
4. Podemos sustituir en  $L_2$  el resultado anterior:

$$\begin{aligned} L_2 &= (a + ba^*b)L_1 \\ &= (a + ba^*b)(b(a + ba^*b))^*aL_0 \end{aligned}$$

5. Finalmente podemos sustituir  $L_2$  en  $L_0$  para calcular la expresión regular que corresponde al autómata:

$$\begin{aligned} L_0 &= aL_2 + bL_0 + \varepsilon \\ &= a(a + ba^*b)(b(a + ba^*b))^*aL_0 + bL_0 + \varepsilon \\ &= \left( a(a + ba^*b)(b(a + ba^*b))^*a + b \right) L_0 + \varepsilon \\ &= \left( a(a + ba^*b)(b(a + ba^*b))^*a + b \right)^* \varepsilon \\ &= \left( a(a + ba^*b)(b(a + ba^*b))^*a + b \right)^* \end{aligned}$$

Las subexpresiones regulares que se obtienen son el análisis de las cadenas aceptadas a partir de cada estado. Por ejemplo la subexpresión  $(a + ba^*b)$  corresponde a la submáquina que involucra al lenguaje  $L_2$  que a su vez depende de  $L_3$  y  $L_1$ :



## Referencias

- [1] J.C. Martin. Introduction to Languages and the Theory of Computation. McGraw-Hill higher education. McGraw-Hill, 2003.