Capítulo 3

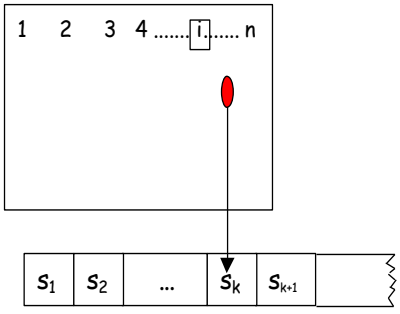
Autómatas de Estados Finitos

Los autómatas finitos son máquinas formales que se usan para reconocer lenguajes regulares. Como se vio en el capítulo anterior, estos son los lenguajes más sencillos, los lenguajes que son generados por gramáticas regulares. En este capítulo se darán las definiciones formales de los autómatas determinísticos y los autómatas no determinísticos. Adicionalmente, se muestra la relación entre estos y las expresiones regulares. Luego, se describen los autómatas con respuestas y cómo estos pueden ser usados para hacer traducciones o cálculos sencillos con las cadenas que pertenecen a los lenguajes regulares. Finalmente, se habla de las limitaciones de este formalismo.

La sección 2.1 describe informalmente lo que es un autómata con aplicaciones en dominios distintos al análisis de lenguajes. La Sección 2.2 da definiciones formales y ejemplos de autómatas de estados finitos. La sección la sección 2.3 muestra la relación entre los autómatas finitos y las expresiones regulares. La Sección 2.4 describe formalmente los autómatas con respuestas y da ejemplos. Finalmente, la Sección 2.5 habla de las limitaciones de los autómatas regulares.

3.1 Qué es un autómata de estados finitos?

Un autómata de estados finitos es una máquina con un número finito de estados que lee símbolos de una cinta de entrada infinita. El comportamiento de la máquina está determinado únicamente por el estado en que se encuentra y el símbolo en la cinta de entrada. Al leer un símbolo de la cinta de entrada cambia de estado y avanza en la cinta de entrada. Cuando ya no quedan símbolos por leer, para. Aún cuando la cita es infinita, la cadena que guía el comportamiento del autómata no lo es. Esta cadena puede ser tan larga como se quiera, pero siempre finita.

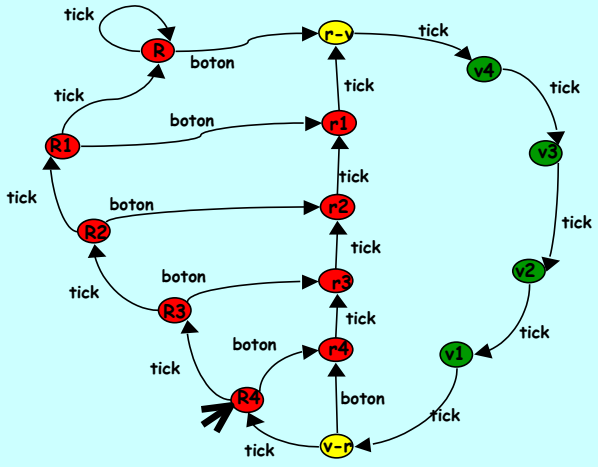


En la máquina de la figura de arriba, en el tiempo k está leyendo s_k y está en el estado i. La cabeza lectora únicamente puede avanzar.

Estas máquinas pueden usarse para reconocer lenguajes. Es decir, para leer cadenas (secuencias de símbolos) y determinar si pertenecen o no a un lenguaje. Los autómatas también sirven para describir el comportamiento de otras máquinas o sistemas. En este caso, no siempre se usa el concepto de cinta de entrada. En lugar de esto el autómata responde a ciertas acciones o estímulos del mundo exterior. Esto no hace que el modelo sea más general ya que podemos pensar en que las acciones se codifican y se ponen en la cinta de entrada que podría inclusive tener una entrada infinita.

Se puede representar un autómata con un multi-grafo[1]. Cada nodo representa un estado, los arcos representan transiciones y están etiquetados con la acción (o con el símbolo leído de la cinta de entrada) que causa la transición. Se debe tener también una convención para indicar los estados donde puede comenzar la ejecución. A continuación se muestran dos ejemplos

Ejemplo 2.1 El autómata de la figura describe el comportamiento de un semáforo peatonal. Inicialmente el semáforo está en rojo para el peatón y se supone que acaba de pasar a rojo. Se quiere garantizar que el semáforo permanezca en rojo al menos cuatro *ticks* de reloj. Esto quiere decir que si el peatón oprime el botón y el semáforo acaba de pasar a rojo debe esperar cuatro *ticks* de reloj para que pase a amarillo (y luego a verde); si han pasado dos *ticks* sólo debe esperar dos *ticks*; y si han pasado 10 cambia inmediatamente. Oprimir el botón más de una vez no afecta el comportamiento. El semáforo permanece en verde por cuatro ticks de reloj.

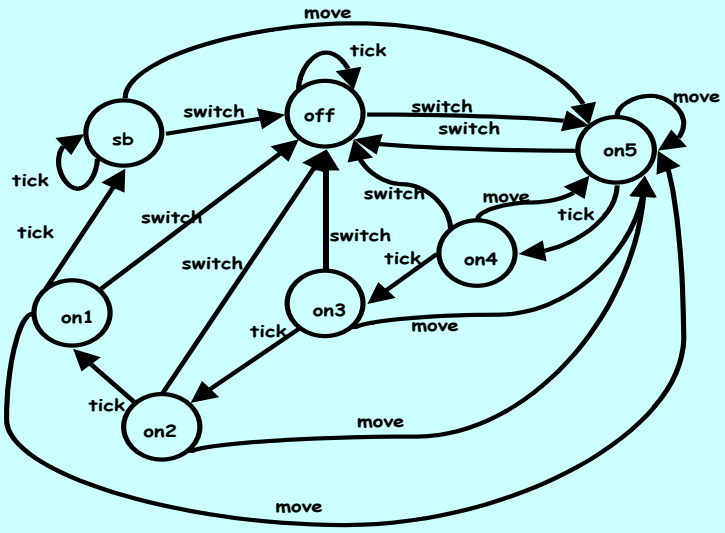


Ejemplo 2.2 El siguiente autómata modela una luz con un temporizador y con sensor de movimiento.

- Al estar apagado (**`off`**), el movimiento no afecta el estado.

[1] Es un multi-grafo por que puede haber más de un arco entre dos nodos.

- Cuando se acciona el interruptor se pasa al estado (**on5**).
- Mientras haya movimiento se queda en (**on5**); si hay un tick de reloj, pasa el estado (**on4**).
- En (**on4**): si percibe movimiento pasa al estado (**on5**); con un tick de reloj, pasa el estado (**on3**).
- En (**on3**): si percibe movimiento pasa al estado (**on5**); con un tick de reloj, pasa el estado (**on2**).
- En (**on2**): si percibe movimiento pasa al estado (**on5**); con un tick de reloj, pasa el estado (**on1**).
- En (**on1**): si percibe movimiento pasa al estado (**on5**); con un tick de reloj, pasa el estado (**sb**).
- En (**sb**): si percibe movimiento pasa al estado (**on5**) de lo contrario con un tick de reloj permanece en este estado.
- En cualquier estado se puede pasar al estado (**off**) accionando el interruptor.



Con estos dos ejemplos, se concluye el tema de autómatas de propósito general. Las siguientes secciones se centran en autómatas para el reconocimiento de lenguajes.

3.2 Autómatas finitos: definiciones formales

En esta sección se presentan las definiciones formales de los autómatas de estados finitos determinísticos y no-determinísticos. Se dan las convenciones para la representación de los autómatas por medio de multi-grafos. Se incluyen ejemplos de reconocedores de lenguajes usando tanto autómatas determinísticos como no-determinísticos. Estos ejemplos se resuelven usando la representación gráfica o también usando un enfoque más formal, lo cual resultará útil para la descripción de autómatas muy grandes o de una alta complejidad. Al final de la sección se discute cómo podría ser la implementación de un autómata en el lenguaje Java y se ilustra cómo podría usarse dentro de una aplicación más grande para ayudar a verificar la entrada del usuario.

3.2.1 Autómatas determinísticos (DFA)

Estos autómatas se denominan determinísticos ya que en cada estado su comportamiento es fijo. Es decir, dado el estado y el símbolo en la cinta de entrada hay un único estado al cual puede pasar.

Definición 2.1. *Un autómata determinístico de estados finitos* **(DFA)**, **M**, *es una quíntupla:* **$(Q, \Sigma, q_I, F, \delta)$**, *donde:*

- **Q** *es un conjunto finito de estados,*
- **Σ** *es un alfabeto finito*
- **$q_I \in Q$** *es el estado inicial*
- **$F \subseteq Q$** *son los estados finales*
- **$\delta : (Q \times \Sigma) \rightarrow Q$** *es la* ***función*** *de transición de estados*

La condición de ser determinístico es debido a que hay un único estado inicial, y las transiciones están descritas por una función total.

Intuitivamente el comportamiento del autómata puede describirse así: el autómata comienza en el estado inicial y lee una secuencia de símbolos: símbolo por símbolo hasta que se acabe la secuencia. En cada instante lee un símbolo **σ**, y dependiendo del símbolo y del estado **s** en el que se encuentra, cambia al estado dado por la función de transición: **$\delta(s, \sigma)$** . Si al terminar de leer toda la secuencia de símbolos está en un estado final, entonces se dice que el autómata acepta la cadena; de lo contrario, se dice que no la acepta.

En este punto vale la pena resaltar el significado de "estado". El hecho que un autómata esté en un estado, quiere decir que el proceso de reconocimiento de la cadena que se está leyendo está en un punto específico que cumple con ciertas condiciones. Es fundamental tener esto presente al diseñar autómatas para reconocer lenguajes.

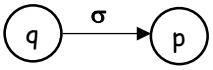
Un autómata puede describirse dando la lista de sus estados, el alfabeto, el estado inicial, los estados finales, y la función transición. Esta función se puede describir usando notación usual para definir funciones o usando una matriz, con una fila por cada estado y una columna por cada símbolo del alfabeto. Si **$\delta(s, \sigma) = s'$** entonces en la matriz se tiene que **$M[s, \sigma] = s'$**. También se puede describir un autómata gráficamente con un grafo generalizado, así:

- Los estados son vértices:

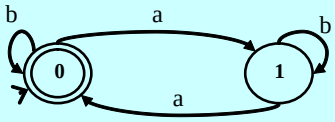
- El estado inicial se marca con una flecha:

- Los estados finales tienen doble borde:

- Si **δ(q,σ)=p** entonces hay un arco etiquetado con **σ** de **q** a **p**:



Ejemplo 2.3 El autómata: M=({0,1},{a,b},{0},{(0,a) →1, (0,b) →0, (1,a) →0, (1,b) →1}} puede representarse gráficamente así:



Note que en este caso el estado inicial también es un estado final.

Como los autómatas se van a usar para reconocer lenguajes, se debe poder determinar cuál es el lenguaje que reconoce un autómata. Esto requiere algunos conceptos adicionales que se definen a continuación.

Definición 2.2. *Dado un autómata* **M=(Q,Σ,q_I,F,δ)** *para* **q** *y* **p** *estados de* **M** *y un símbolo del alfabeto* **σ**, *si* **δ(q,σ)=p** *decimos que* **p** *es* **σ-*sucesor*** *de* **q** *y escribimos* $q \xrightarrow{\sigma} p$.

Se extiende esta definición a cadenas así:

Definición 2.3. *Dado un autómata* M=(Q,**Σ**,q_I,F,**δ**) *para* p *y* q *estados de* M *y* **ω ∈ Σ***, *decimos que* p *es* **ω-*sucesor*** *de* q *y escribimos* $q \xRightarrow{\omega} p$.*Si:*

- **ω=λ** y **p = q**, ó
- **ω=σω'** *y existe un estado* **p'** *tal que* $\mathbf{q} \xrightarrow{\sigma} \mathbf{p'}$ *y* $\mathbf{p'} \xRightarrow{\omega'} \mathbf{p}$

(o equivalentemente: $\boldsymbol{\omega=\sigma_1\sigma_2...\sigma_n}$ *y existen estados* $\mathbf{q_1, q_2, ..., q_n}$. *con* $\mathbf{p = q_1}$ *y* $\mathbf{q = q_n}$ *tales que para todo* **i** *con* $\mathbf{0<i \leq n}$ *se tiene que* $\boldsymbol{\delta(q_{i-1},\sigma_i) = q_i}$*)*

Esta definición puede darse también basándose en la clausura transitiva de la función de transición. La clausura transitiva se obtiene a partir de la función de transición así:

Podemos extender la función de transición usando la clausura transitiva así:

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

- $\delta^*(q, \lambda) = q$
- $\delta^*(q, \sigma\omega) = \delta^*(\delta(q,\sigma),\omega)$

Como δ es una función, entonces para todo estado p y toda cadena $\omega \in \Sigma^*$ existe un único estado q talque ($\delta^*(p,\omega) = q$). Por lo tanto, δ^* también es una función. Note también que $p \stackrel{\omega}{\Longrightarrow} q$ si y solamente si $\delta^*(p,\omega) = q$, dando así otra definición para el concepto de ω-sucesor.

Los conceptos dados ya permiten definir qué quiere decir que un autómata acepta una cadena.

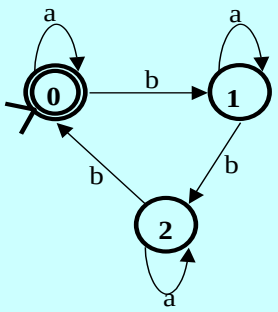
Definición 2.4. *Dado un autómata* $\mathbf{M=(Q,\Sigma,q_I,F,\delta)}$, *decimos que* **M** *acepta una cadena* $\boldsymbol{\omega}$ *si y solo si* $\mathbf{q_I \stackrel{\omega}{\Longrightarrow} q}$ *y* $\mathbf{q \in F}$ (*o equivalentemente si* $\boldsymbol{\delta^*(q_I, \omega) \in F}$).

El lenguaje reconocido por un autómata es el conjunto de todas las cadenas que son aceptadas por un autómata.

Definición 2.5. *Dado un autómata* $\mathbf{M=(Q,\Sigma,q_I,F,\delta)}$, *el lenguaje reconocido por* **M**, $\mathbf{L(M)}$ *es el conjunto de todas las cadenas aceptadas por* **M**. $\mathbf{L(M)=\{\omega: q_I \stackrel{\omega}{\Longrightarrow} q \text{ y } q \in F\}}$, (*i.e.*, $\mathbf{L(M)=\{\omega: \delta^*(q_I, \omega) \in F\}}$).

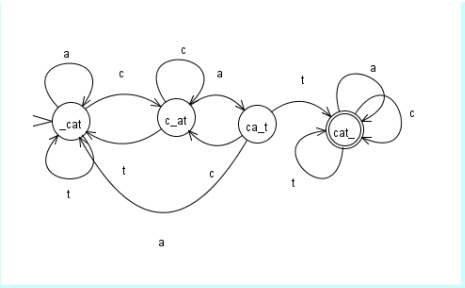
El autómata del Ejemplo 2.3 reconoce las cadenas formadas por a's y b's que tienen un número par de a's.

Ejemplo 2.4 Este autómata reconoce las cadenas formadas por a's y b's tales que el número de b's es múltiplo de tres.



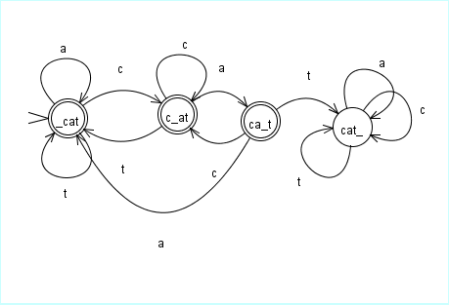
Es conveniente volver a resaltar el significado del nombre de los estados en este ejemplo. Cuando el autómata está en el estado 0, el número de b's módulo tres que han salido hasta el momento es 0, en el estado 1 es uno y en el estado 2 es dos. Se habrían podido nombrar los estados A, B y C. Pero al nombrarlos con números, el nombre del estado tiene significado en relación con las condiciones que se cumplen en el proceso de reconocimiento de la cadena.

Ejemplo 2.5 El siguiente autómata reconoce las cadenas del alfabeto {c, a, t} que contienen la cadena cat como subcadena.



Note que en el estado _cat se está esperando la subcadena **cat**. En el estado 2, si sale la subcadena **at** ya debe aceptar la cadena. En el estado 3 le hace falta sólo una t. Finalmente, en el estado 4, ya recibió la cadena y no importa lo que salga.

Ejemplo 2.6 El siguiente autómata reconoce las cadenas que no contienen la cadena cat como subcadena.



Note que el autómata del Ejemplo 2.6 difiere del autómata del ejemplo 2.5 en los estados finales.

Este ejemplo da las bases para el siguiente teorema:

Teorema 2.1. *Dado un autómata determinístico:* $\mathbf{M=(Q,\Sigma,q_I,F,\delta)}$ *tal que* $\mathbf{L(M)=L}$, *el autómata* $\mathbf{M'=(Q,\Sigma,q_I,Q-F,\delta)}$ *reconoce el complemento de* **L**. *Es decir,* $\mathbf{L(M') = \sim L(M)}$.

Demostración:

$\mathbf{L(M')}$

$=$ ⟨ **Def de L(M)** ⟩

$\mathbf{\{\omega: \delta^*(q_I, \omega) \in Q-F\}}$

$=$ ⟨ **Q es el Universo de los estados** ⟩

$\mathbf{\{\omega: \delta^*(q_I, \omega) \notin F\}}$

$=$ ⟨ $\mathbf{\{x:\neg Q\} = \sim\{x:Q\}}$ ⟩

$\mathbf{\sim\{\omega: \delta^*(q_I, \omega) \in F\}}$

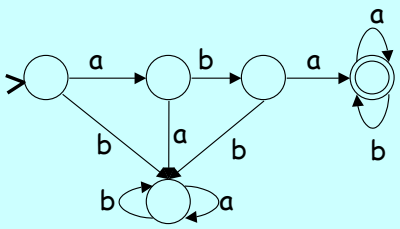
$=$ ⟨ **Def de L(M)** ⟩

$\mathbf{\sim L(M)}$

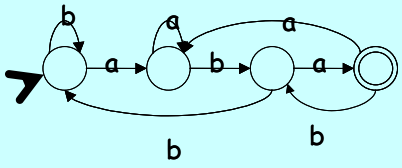
El teorema y su demostración ilustran cómo dado un autómata regular determinístico que reconoce un lenguaje L, se puede construir un autómata que reconoce el complemento del lenguaje.

A continuación se muestran varios ejemplos de reconocedores.

Ejemplo 2.7 El siguiente autómata reconoce las cadenas que comienzan con la subcadena **aba**. Note que si llega a comenzar con algo diferente a **aba** llega a un sumidero del cual no puede salir. Una vez ha recibido la subcadena **aba** ya no importa qué sigue, acepta la cadena.



Ejemplo 2.8 El siguiente autómata reconoce las cadenas que terminan con la subcadena **aba**. Para esto, modificamos el autómata que reconoce las cadenas que contienen **aba** haciendo que el estado final no sea un sumidero.

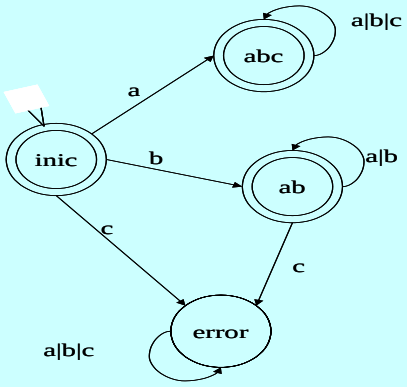


Hay ejemplos más complicados que exigen primero la comprensión de condiciones complejas para poder construir el autómata requerido. En estos casos, también se deben combinar autómatas para llegar a la solución.

Ejemplo 2.9 Se quiere construir un autómata que reconozca las cadenas sobre {a,b,c} donde c sólo puede aparecer si la cadena comienza con a.

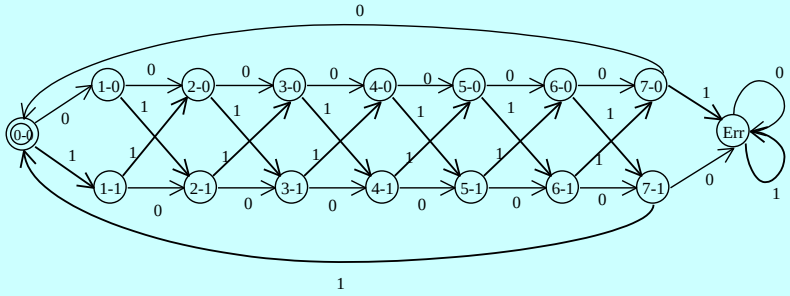
En este caso, se debe resaltar que lo que se dice es que c puede aparecer si comienza con a, no dice que debe aparecer. Dicho en otras palabras, c no puede aparecer si la cadena no comienza por a. Otra característica que se puede deducir es que las cadenas que pertenecen al lenguaje no pueden comenzar con c, pues si comienzan con c no comienza por a y al no comenzar por a no puede contener c. También se puede deducir que el autómata acepta cualquier cadena que no contiene c.

El autómata entonces se compone de dos sub-autómatas, uno que acepta c y otro que no. Ahora si comienza con a, entonces pasa al autómata que acepta las cadenas que contienen c, si comienza con b pasa al que no acepta c. Todos los estados son finales menos el estado de error.



El siguiente ejemplo muestra la utilidad de los autómatas para comprobación de corrección en las comunicaciones.

Ejemplo 2.10 Se quiere verificar que la cadena que se esté leyendo esté formada por secuencias sucesivas de 8 bits en las que el octavo bit indica la paridad de los 7 anteriores.



El nombre del estado (i,j) indica lo siguiente: i indica el número de bits (módulo 8) que ha salido hasta el momento mientras que j indica la paridad de los i bits anteriores.

Si el número de estados es muy grande, puede no resultar práctico dibujar el autómata. En algunos casos se puede definir con una tabla de estados contra símbolos del alfabeto de entrada. Aún así podría resultar complicado. La idea es darle un nombre adecuado a los estados para poder definir la función de transición fácilmente.

Ejemplo 2.10a El autómata del Ejemplo 2.10 lo habríamos podido definir así:

- $Q = \{ (x,y) : x \in [0..7] \ \wedge \ y \in \{0,1\}\} \cup \{err\}$
- $\delta(err,y) = err$ para $y \in \{0,1\}$
- $\delta((x,y),z) = (x+1,(y+z) \bmod 2)$ $x \in [0...6] \wedge y \in \{0,1\} \wedge z \in \{0,1\}$

- $\delta((7,y),z) = (0,0)$ si $y = z$
- $\delta((7,y),z) = err$ si $y \neq z$
- $F = \{(0,0)\}$
- $q_i = (0,0)$

El siguiente ejemplo muestra cómo es más fácil construir un autómata dando la definición de la función que dibujar el autómata, aún cuando la función a primera vista parezca complicada.

Ejemplo 2.11 Un autómata regular que reconozca el lenguaje sobre {0,1,2,3} tal que las cadenas que pertenecen al lenguaje: $\sigma_1 \ldots \sigma_n$ cumplen con la siguientes condiciones:

- $\sigma_1 \in \{0,1,2,3\}$
- $\sigma_2 \in \{0,1,2,3\}$
- $\sigma_{i-1} \leq \sigma_i \leq \sigma_{i+1}$ si $\sigma_i \leq \sigma_{i+1}$
- $\sigma_{i+1} \leq \sigma_i \leq \sigma_{i-1}$ si $\sigma_{i+1} \leq \sigma_i$

Note que lo que está haciendo es definiendo σ_{i+1} a partir de las anteriores. Rescribiendo la definición para definir σ_i, tenemos:

- $\sigma_i \geq \sigma_{i-1} \geq \sigma_{i-2}$ si $\sigma_{i-1} \geq \sigma_{i-2}$
- $\sigma_i \leq \sigma_{i-1} \leq \sigma_{i-2}$ si $\sigma_{i-1} \leq \sigma_{i-2}$

Ahora a construir el autómata

- En primera instancia, lo único que podemos saber del autómata es su alfabeto de entrada: {0,1,2,3}.
- Se debe recordar que los estados modelan, como su nombre lo indica, un estado en la computación (i.e., en este caso, modelan un paso en el proceso de reconocimiento de cadenas). Denotamos los estados (a,b) donde a es el valor que se leyó en tiempo i-2 y b es el valor leído en tiempo i-1. Faltaría decir cómo se comienza. pero en realidad, no hay restricción sobre los dos primeros símbolo de las cadenas. Se debe asegurar que al leer el segundo símbolo se llega al estado correcto (a,b). Entonces, necesitamos 5 nuevos estados:

$$\{inic,0,1,2,3\}.$$

Inic es el estado inicial, cuando no ha leído nada. Cada uno de los estados {0,1,2,3} indica que se ha leído un solo símbolo y cual se ha leído. Se tiene entonces: $Q = \{(x,y) : 0 \leq x,y < 4\} \cup \{0,1,2,3,inic, error\}$

- El estado inicial es inic
- No hay restricción sobre la longitud de la cadena. Mientras no haya llegado al estado de error, la cadena está bien: por lo tanto $F = Q - \{error\}$.

- Ahora para las transiciones: estando en el estado (a,b), si leemos c, se debe cumplir una de las siguientes dos condiciones $a \leq b \leq c$ o $a \geq b \geq c$. Si no se cumple, se debe pasar a un estado de error que es además un sumidero. Si se cumple, se pasa al estado (b,c) pues al leer también avanzamos en el tiempo y estamos en el tiempo (i+1), habiendo leído b en tiempo (i) y b en tiempo (i-2). Esto se expresa de la siguiente forma:
  - Para: $(0 \leq b,a,c < 4)$
    - $\delta((a,b),c) = (b,c)$ para $a \leq b \leq c$
    - $\delta((a,b),c) = (b,c)$ para $a \geq b \geq c$
    - $\delta((a,b),c) =$ error para $a \leq b \wedge c < b$
    - $\delta((a,b),c) =$ error para $a \geq b \wedge c > b$
    - $\delta(error,a) = error$
  - Las Transiciones iniciales: para $0 \leq a,b < 4$:
    - $\delta(inic,a) = a$
    - $\delta(a,b) = (a,b)$

La función de transición también se habría podido describir con con una tabla T de estados x entradas, donde Tabla[x,y] =δ(x,y).

	0	**1**	**2**	**3**
Error	error	Error	error	Error
Inic	0	1	2	3
0	(0,0)	(0,1)	(0,2)	(0,3)
1	(1,0)	(1,1)	(1,2)	(1,3)
2	(2,0)	(2,1)	(2,2)	(2,3)
3	(3,0)	(3,1)	(3,2)	(3,3)
(0,0)	(0,0)	(0,1)	(0,2)	(0,3)
(0,1)	error	(1,1)	(1,2)	(1,3)
(0,2)	error	Error	(2,2)	(2,3)
(0,3)	error	Error	error	(3,3)
(1,0)	(0,0)	Error	error	Error
(1,1)	(1,0)	(1,1)	(1,2)	(1,3)
(1,2)	error	error	(2,2)	(2,3)
(1,3)	error	error	error	(3,3)
(2,0)	(0,0)	error	error	Error
(2,1)	(1,0)	(1,1)	error	Error
(2,2)	(2,0)	(2,1)	(2,2)	(2,3)
(2,3)	error	error	error	(3,3)
(3,0)	(0,0)	error	error	Error
(3,1)	(1,0)	(1,1)	error	Error
(3,2)	(2,0)	(2,1)	(2,2)	Error
(3,3)	(3,0)	(3,1)	(3,2)	(3,3)

EJERCICIOS

1. Describa un autómata determinístico que acepte las cadenas sobre {a,b} en las que el número de a's módulo 2 es igual al número de b's módulo 2.
2. Describa un autómata determinístico que acepte las cadenas sobre {a,b} que no comienzan por a.
3. Describa un autómata que acepte las cadenas sobre {a,b} tales que el número de b's que aparecen en la cadena NO es múltiplo de 3.
4. Describa un autómata determinístico que acepte el cadenas sobre {a,b} tal que el número de a’s módulo 3 más el número de b’s módulo 3 es menor o igual que cuatro.

5. Describa un autómata determinístico que acepte cadenas sobre {a,b} que contienen la subcadena aba si comienzan por la subcadena bbb. Si no comienza por la subcadena bbb no deben tener la subcadena aba. Note que si comienza con el prefijo bbb no tiene que tener la subcadena aba.

6. Describa un autómata determinístico que reconozca el lenguaje sobre [0-9] tal que las cadenas que pertenecen al lenguaje: $\sigma_1 \ldots \sigma_n$ cumplen con la siguientes condiciones:
   - $\sigma_1 \in$ [0-9]
   - $\sigma_2 \in$ [0-9]
   - $\sigma_{i-1} \leq \sigma_i \leq \sigma_{i-2}$ si $\sigma_{i-1} \leq \sigma_{i-2}$ para i>2
   - $\sigma_{i-1} \geq \sigma_i \geq \sigma_{i-2}$ si $\sigma_{i-1} > \sigma_{i-2}$ para i>2

3.2.2 Autómatas No-determinísticos (NDFA)

Podemos definir autómatas finitos donde la transición puede no ser determinística y donde se pueda realizar una transición leyendo cero o más símbolos de la cinta de entrada. Esto le agrega facilidad de expresión, mas no poder computacional.

Definición 2.6. *Un* ***autómata no-determinístico de estados finitos*** *(*NDFA*),* **M**, *es una quintupla:* **(Q,Σ,I,F,Δ)**, *donde:*

- **Q** *es un conjunto finito de estados,*
- **Σ** *es un alfabeto finito*
- **I** $\subseteq$ **Q** *son los estados iniciales*
- **F** $\subseteq$ **Q** *son los estado finales*
- **Δ** $\subseteq$ **(Q** $\times$ **Σ*)** $\times$ **Q** *es una relación de transición de estados*

En los autómatas no-determinísticos, el concepto de sucesor es un poco más complicado que en los autómatas determinísticos.

Definición 2.7. *Dado un autómata* **M=(Q,Σ,I,F,Δ)** *para* **p**$\in$**Q**, **q**$\in$**Q**, **q** *estados de* **M** *y* **ω** $\in$ **Σ***, *decimos que* **p** *es* **ω-*sucesor*** *de* **q** *y escribimos* $\mathbf{q} \overset{\omega}{\Longrightarrow} \mathbf{p}$, *si ocurre cualquiera de las siguientes condiciones:*

- **ω=λ** *y* **p = q**
- **((q, ω), p)**$\in$**Δ**
- *Existen cadenas* **α**$\in$**Σ*** *y* **β**$\in$**Σ*** *tales que* **ω=αβ** *y existe un estado* **q'** *tal que* **((q, α),q')**$\in$**Δ** *y* $\mathbf{q'} \overset{\beta}{\Longrightarrow} \mathbf{p}$.

Note que en un autómata no-determinístico, para algún **ω**, un estado puede tener más de un **ω**-sucesor o ningún **ω**-sucesor.

La clausura transitiva de la relación de transición se define así:

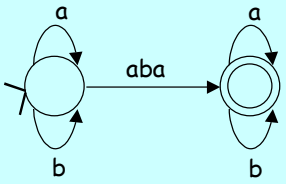
$\Delta^* : (Q \times \Sigma^*) \times Q$

- $((q, \lambda),q) \in \Delta^*$
- si $((q, \omega),p) \in \Delta$ entonces $((q, \omega),p) \in \Delta^*$
- si $\omega \in \Sigma^*$ y existen $\alpha \in \Sigma^*$, $\beta \in \Sigma^*$ y $r \in Q$ tales que $\omega = \alpha\beta$ y $((q, \alpha),r) \in \Delta$ y $((r, \beta),p) \in \Delta^*$ entonces $((q, \omega),p) \in \Delta^*$.

En estos autómatas, la clausura transitiva da origen a una definición alterna al concepto de ω-sucesor ya que p es ω-sucesor de q si y solamente si $((q, \omega),p) \in \Delta^*$.

Los autómatas no determinísticos suelen ser más sencillos que los determinísticos. El siguiente ejemplo muestra cómo se puede simplificar la definición del Ejemplo 2.5.

Ejemplo 2.12 Un autómata que reconoce las cadenas que contienen la subcadena aba



La definición formal de este autómata sería la siguiente:

- **Q = {1,2}**
- **Σ={a,b}**
- **I={1}**
- **F={2}**
- **Δ={((1,a),1),((1,b),1),((1,aba),2),((2,a),2),((2,b),2)}**

A continuación se define el cómo se puede determinar cuál es el lenguaje reconocido por un autómata no determinístico. El concepto de configuraciones puede servir para aclarar el comportamiento de los autómatas. Una configuración indica el estado del proceso de reconocimiento de una cadena en particular. Indica en qué estado está el autómata y qué le falta por leer.

Definición 2.8. *Dado un autómata,* **M=(Q,Σ,I,F,Δ)**, *una configuración sobre* **M** *es una pareja* **⟨q,ω⟩**, *con* **q∈Q** *y con* **ω∈Σ***.

En ejemplo de arriba algunas configuraciones son: ⟨1, ababa⟩, ⟨1,baba⟩, ⟨2, ba⟩, ⟨2, a⟩ y ⟨2,λ⟩.

Podemos pasar de una configuración a otra usando la relación de transición de estados.

Definición 2.9. *Dado un autómata,* **M**, *y* **⟨q,ω⟩** *y* **⟨q',ω'⟩** *configuraciones de* **M** *se dice que* **⟨q',ω'⟩** *es alcanzable en un paso a partir de* **⟨q,ω⟩** *y escribimos* **⟨q,ω⟩ → ⟨q',ω'⟩** *si:*

- **ω = βα**
- **ω' = α**
- **((q, β),q')∈Δ**

La relación de ser alcanzable en un paso puede extenderse con su clausura reflexiva-transitiva para obtener la relación de ser alcanzable en cero o más pasos. Volviendo al Ejemplo 2.12, y a las configuraciones mencionadas arriba se tiene que:

- ⟨1, ababa⟩ → ⟨1,baba⟩,
- ⟨1, ababa⟩ → ⟨2, ba⟩,
- ⟨2, ba⟩ → ⟨2,a⟩
- ⟨2, a⟩ → ⟨2,λ⟩

Definición 2.10. *Dado un autómata,* **M**, *y* **⟨q,ω⟩** *y* **⟨q',ω'⟩** *configuraciones de* **M** *se dice que* **⟨q',ω'⟩** *es alcanzable (en cero más pasos) a partir de* **⟨q,ω⟩** *y escribimos* **⟨q,ω⟩ ⇒ ⟨q',ω'⟩** *si:*

- **⟨q',ω'⟩ = ⟨q,ω⟩**
- **⟨q,ω⟩ → ⟨q',ω'⟩**
- *Existe una configuración* **⟨q'',ω''⟩** *tal que* **⟨q,ω⟩ → ⟨q'',ω''⟩** *y* **⟨q'',ω''⟩ ⇒ ⟨q',ω'⟩**

Un autómata no-determinístico acepta una cadena, **ω** si, comenzando en un estado inicial, con **ω** ***puede*** llegar a uno final leyendo TODA la cadena. Formalmente:

Definición 2.11. *Dado un autómata* **M=(Q,Σ,I,F,Δ)** *decimos que M acepta una cadena* $\omega \in \Sigma^*$ *si existen* $q_I \in I$ *y* $\mathbf{f} \in F$ *tales que:*

- $\mathbf{q_I} \overset{\omega}{\Longrightarrow} \mathbf{f}$, *(o equivalentemente)*
- **⟨q_I,ω⟩ ⇒ ⟨f,λ⟩**

El lenguaje reconocido por un autómata incluye todas las cadenas que son aceptadas por el autómata.

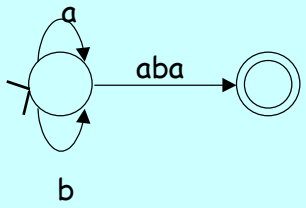
Definición 2.12. **L(M)**, ***el lenguaje reconocido por*** **M=(Q,Σ,I,F,Δ)**, *es el conjunto de todas las cadenas aceptadas por* **M**.

$$L(M) = \{\omega: \omega \in \Sigma^*, \text{ existen } q_I \in I \text{ y } f \in F, \text{ tales que } q_I \overset{\omega}{\Longrightarrow} f\ \}$$

o, usando la notación de configuraciones

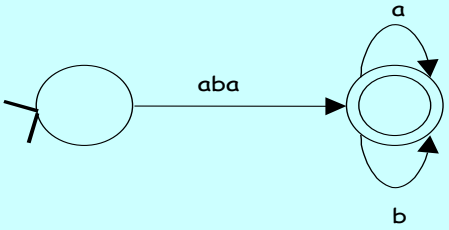
$$L(M) = \{\omega: \omega \in \Sigma^*, \text{ existen } q_I \in I \text{ y } f \in F, \text{ tales que } \langle q_I,\omega\rangle \Rightarrow \langle f,\lambda\rangle\}$$

Ejemplo 2.13 Un autómata que reconoce las cadenas que terminan con la cadena aba



En el ejemplo anterior, al leer la cadena abab, podría llegar al estado final, usando el arco etiquetado con aba, pero la cadena no es aceptada, ya que al llegar al estado final, no ha terminado de leer toda la cadena.

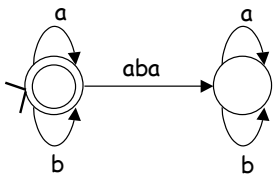
Ejemplo 2.14 Un autómata que reconoce las cadenas que comienzan con la subcadena aba.



Al comparar estos autómatas con los de la sección anterior, se nota que éstos no requieren sumideros ya que en este caso la cadena no terminaría de leerse.

Es importante resaltar que dado un autómata no-determinístico que reconoce L, al cambiar los estados finales por no finales y viceversa no se obtiene autómata que reconozca el complemento de L.

Por ejemplo, al cambiar los estados finales por no finales del autómata descrito arriba, se obtiene el siguiente autómata.



Note que con la cadena aba se puede quedar en el estado inicial y por lo tanto aceptar la cadena.

En la mayoría de

Es muy importante notar que si hay alguna forma de partir de un estado inicial con una cadena y llegar a un estado final, se dice que el autómata acepta la cadena, así haya alguna forma de no aceptarla. Se dice que un autómata no determinístico no acepta una cadena si no existe forma de llegar de un estado inicial a un estado final habiendo leido toda la cadena. Es importante enfatizar este último concepto. Una cadena no es aceptada si no existe forma de aceptarla.

Ejercicios:

1. Defina un autómata de estados finitos que reconozca cadenas sobre el alfabeto {0,1,2} en el que la suma de los dígitos es no es múltiplo de 3.
2. Defina un autómata de estados finitos que reconozca las cadenas sobre el alfabeto {0,1,2,3,4,5,6,7,8,9} que representan número múltiplos de 10. Suponga que el lee el número de izquierda a derecha.
3. Defina autómatas no-determinísticos para los ejercicios de la sección anterior.

3.2.3 Equivalencia entre autómatas determinísticos y no-determinísticos (NDFA)

Se puede demostrar que estos dos tipos de autómatas son equivalentes. Para demostrar esto tenemos que demostrar que siempre que tenemos un autómata determinístico podemos construir otro no determínistico equivalente y viceversa.

La primera parte es fácil, ya que Un autómata determinístico también es un autómata no determinístico donde:

- para todo $((q, \omega),p) \in \Delta$, $|\omega| = 1$
- para todo $(q, \omega) \in (Q \times \Sigma)$, existe un $p \in Q$ tal que $((q, \omega),p) \in \Delta$
- si $((q, \omega),p) \in \Delta$ y $((q, \omega),r) \in \Delta$, entonces $p=r$

La segunda parte de la demostración en la cual se muestra que dado un autómata no-determinístico siempre se puede construir un autómata determinístico equivalente no es tan sencilla. La demostración de este hecho se encuentra en [1].

3.3 Equivalencia de Expresiones Regulares y Autómatas de Estados Finitos

La clase de lenguajes que son reconocidos por un autómata de estados finitos es igual a la clase de lenguajes que pueden ser descritos por una expresión regular.

> ***Teorema 2.2.*** *Un lenguaje L puede representarse usando expresiones regulares si y solo si existe un autómata finito que lo reconoce.*

No se demostrará este teorema formalmente; únicamente se ilustra cómo para todo lenguaje L expresado por medio de una expresión regular podemos construir un autómata no determinístico que lo reconoce. De hecho, así se demuestra, usando inducción estructural, que todo lenguaje que se puede definir por medio de una expresión regular, es reconocido por algún autómata determinístico.

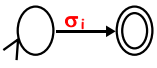
Para lograr esto, se comienza mostrando que para todo lenguaje descrito con una expresión regular básica podemos construir un autómata que lo reconoce. Luego se supone que se tienen autómatas para expresiones regulares y se muestra que se pueden construir autómatas[2] para expresiones regulares compuestas.

Sea $\Sigma=\{\sigma_1,\ldots,\sigma_n\}$.

- El lenguaje representado por $\varnothing$ es reconocido por el autómata no determinístico $M=(Q,\Sigma,I,F,\Delta)$ con $Q=\{1\}$, $I=\{1\}$, $F=\varnothing$, $\Delta=\varnothing$:

- El lenguaje representado por λ es reconocido por el autómata no determinístico $M=(Q,\Sigma,I,F,\Delta)$ con $Q=\{1\}$, $I=\{1\}$, $F=\{1\}$, $\Delta=\varnothing$:

- Para todo σ_i el lenguaje representado por σ_i es reconocido por el autómata no determinístico $M=(Q,\Sigma,I,F,\Delta)$ con $Q=\{1,2\}$, $I=\{1\}$, $F=\{2\}$, $\Delta=\{((1, \sigma i),2)\}$:

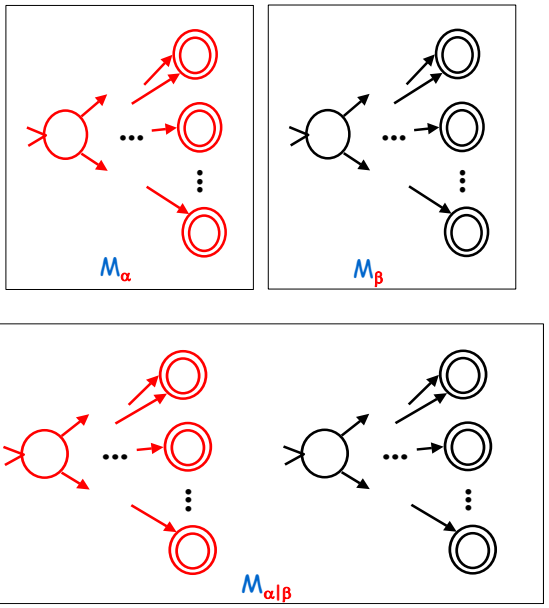


[2] Estos autómatas pueden ser determinísticos o no-determinísticos ya que estos dos formalismos son equivalentes. Un autómata determinístico es un caso particular de autómata no-determinístico y un autómata no-determinístico siempre puede convertirse a uno determinístico equivalente.

- Suponga que $M_\alpha = (Q_\alpha,\Sigma_\alpha,q_{I\alpha}, F_\alpha, \delta_\alpha)$ y $M_\beta=(Q_\beta,\Sigma_\beta,q_{I\beta}, F_\beta, \delta_\beta)$ son autómatas de estados finitos determinísticos que reconocen los lenguajes descritos por **α** y **β** respectivamente. Entonces se deben construir autómatas $M_{\alpha|\beta}$, $M_{\alpha\beta}$ y M_{α^*} tales que reconozcan los lenguajes descritos por **(α|β)**, **(αβ)** y **(α*)** respectivamente. Abajo se muestran estos autómatas. Debe ser evidente para el lector que estos autómatas reconocen los lenguajes requeridos.

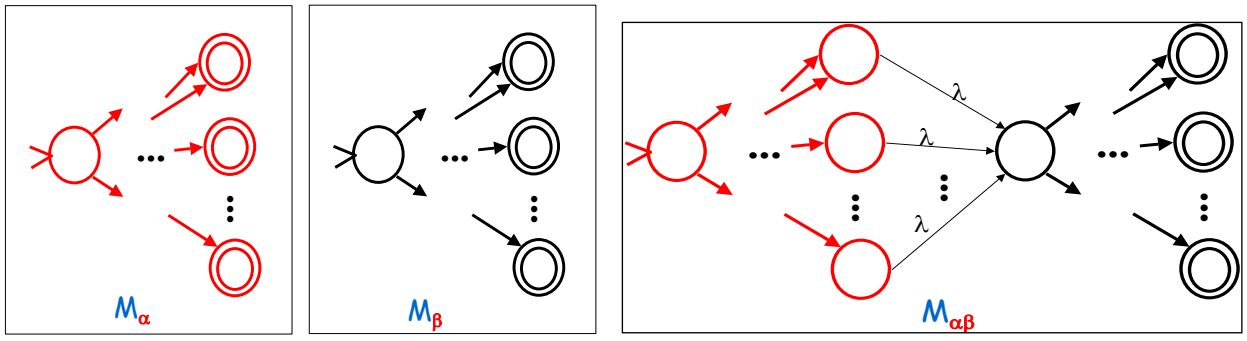
  - $M_{\alpha|\beta} =(Q_\alpha\cup Q_\beta, \Sigma_\alpha\cup\Sigma_\beta, \{q_{I\alpha},q_{I\beta}\},F_\alpha\cup F_\beta,\Delta)$ con

    $\Delta= \{((q,\sigma), \delta_\alpha(q,\sigma)):(q\in Q_\alpha) \wedge (\sigma \in\Sigma_\alpha)\} \cup \{((q,\sigma), \delta_\beta(q,\sigma)):(q\in Q_\beta) \wedge (\sigma \in\Sigma_\beta)\}$



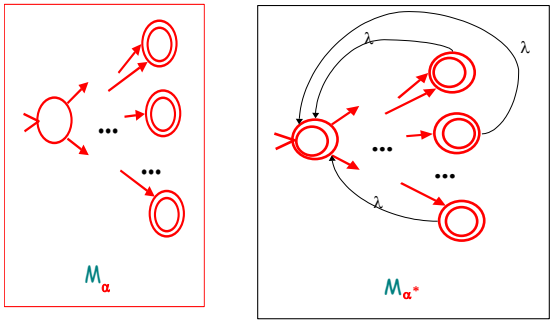
  - $M_{\alpha\beta}=(Q_\alpha\cup Q_\beta, \Sigma_\alpha\cup\Sigma_\beta, \{q_{I\alpha}\},F_\alpha,\Delta)$

    con $\Delta =$ $\{((q,\sigma), \delta_\alpha(q,\sigma)):(q\in Q_\alpha) \wedge (\sigma \in\Sigma_\alpha)\} \cup$
    $\{((q,\sigma), \delta_\beta(q,\sigma)):(q\in Q_\beta) \wedge (\sigma \in\Sigma_\beta)\} \cup$
    $\{((q,\lambda), q_{I\beta}):q\in F_\alpha)\}$



El NDFA: $M_{\alpha^*}=(Q_\alpha \cup \{q_I\}, \Sigma_\alpha, \{q_{I\alpha}\}, F_\alpha \cup \{q_I\}, \Delta)$

con $\Delta=\{((q,\sigma),q'):((q\in Q_\alpha)\wedge(\delta_\alpha(q,\sigma)=q'))\} \cup \{((q,\lambda), q_I):q\in F_\alpha\cup\{q_I\})\}$



3.4 Autómatas con Respuestas

En las secciones anteriores se mostraron los autómatas como una herramienta para reconocer lenguajes. Los autómatas sirven para modelar otro estilo de problemas. Podemos generalizar la definición de autómata para que en cada estado o en cada transición se produzca una respuesta. En este caso, el autómata no solo tendría una cinta de entrada sino también una cinta de salida. En cada instante lee un símbolo de la cinta de entrada y dependiendo de esto escribe algo en la de la salida y cambia de estado. No se puede devolver para revisar lo escrito. Las aplicaciones más naturales de este tipo de autómatas es la traducción de un lenguaje en otro.

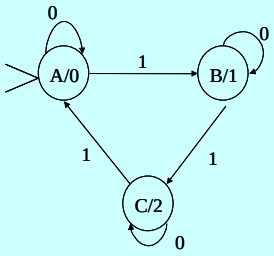
Hay dos tipos de autómatas de salida: unos con respuestas en los estados y otros con respuestas en las transiciones.

Definición 2.13. *Un autómata determinístico con respuestas en los estados, M, es una séxtupla:* $(Q,\Sigma, \Sigma',q_I,\delta,g)$*, donde:*

- Q *es un conjunto finito de estados,*
- Σ *es el alfabeto de entrada, un conjunto finito de símbolos*
- Σ' *es el alfabeto de salida, un conjunto finito de símbolos*
- $q_I \in Q$ *es el estado final*
- $\delta: (Q \times \Sigma) \rightarrow Q$ *es la función de transición de estados*
- $g: Q \rightarrow \Sigma^*$ *es la función de* salida

Al dibujar estos autómatas se escribe el identificador del estado, seguido por el símbolo "/", seguido por la cadena de salida.

Ejemplo 2.15 El siguiente autómata es un contador en módulo 3 del número de 1's que han salido hasta el momento.



Si un autómata con respuestas en los estados recibe la cadena vacía en todo caso contesta algo: la respuesta del estado inicial. El autómata de arriba responde 0 si recibe la cadena vacía. Esto es correcto ya que, en la cadena vacía el número de 1´s es 0 i 0 mod 3 = 0.

Los autómatas con respuestas en las transiciones producen una salida cada vez que se ejecuta una transición.

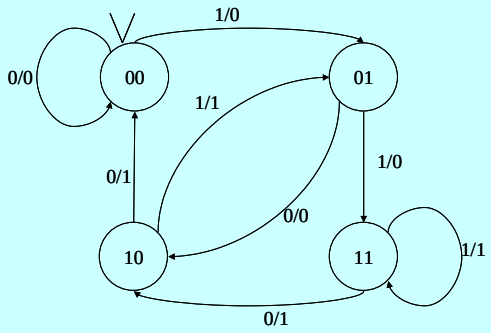
Definición 2.14. Un **autómata determinístico con respuestas en las transiciones, *M***, es una séxtupla: $(\mathbf{Q,\Sigma, \Sigma', q_I,\delta,h})$, donde:

- Q *es un conjunto finito de estados,*
- Σ *es el alfabeto de entrada, un conjunto finito de símbolos*
- Σ' *es el alfabeto de salida, un conjunto finito de símbolos*
- $\boldsymbol{\delta}$: $(Q \times \Sigma) \rightarrow Q$ *es la función de transición*
- **h**: $(Q \times \Sigma) \rightarrow \Sigma'^*$ *es la función de salida*

Algunos autores definen la función de salida, ***h***, como $(Q \times \Sigma) \rightarrow \Sigma$. Es decir sólo se admite un símbolo como respuesta.

Al dibujar estos autómatas en las transiciones se escribe a/w donde a es el símbolo que causa la transición y w w es lo que escribe.

Ejemplo 2.16 El siguiente autómata replica la entrada con un retraso de 2. Por defecto, los primeros dos símbolos que imprime son siempre 0.



El siguiente ejemplo es, tal vez el más complejo que tiene este capítulo. Se muestra el proceso de construcción.

Ejemplo 2.17 Se quiere definir un autómata con respuestas en las transiciones para que lea un número en base 10 de izquierda a derecha y escriba el resultado de dividir el número por 4 seguido por el caracter R seguido por el residuo. Puede suponer que el final del número de entrada se marca con el símbolo #.

Primero se va a mostrar cómo se divide un número por 4. Digamos: 1234#.:

- Divida 01 por 4 escriba 0 y lleve 1
- Divida 12 por 4 escriba 3 y lleve 0
- Divida 03 por 4 escriba 0 y lleve 3
- Divida 34 por 4 escriba 8 y lleve 2
- Finalmente escriba el residuo que es 2

Para poder implementar un proceso se necesitan cuatro estados, donde cada estado indica cuánto se lleva, y uno adicional para terminar.

Si está en el estado i, quiere decir que el residuo que se lleva es i. Veamos qué pasa en cada estado al leer cada símbolo de entrada.

En el estado 0, el estado inicial

Lee	Interpreta	Escribe	Pasa a
0	0	0	0
1	1	0	1
2	2	0	2
3	3	0	3
4	4	1	0
5	5	1	1
6	6	1	2
7	7	1	3
8	8	2	0
9	9	2	1
#		R0	F

En el Estado 1

Lee	Interpreta	Escribe	Pasa a
0	10	2	2
1	11	2	3
2	12	3	0
3	13	3	1
4	14	3	2
5	15	3	3
6	16	4	0
7	17	4	1
8	18	4	2
9	19	4	3
#		R1	F

En el estado 2

Lee	Interpreta	Escribe	Pasa a
0	20	5	0
1	21	5	1
2	22	5	2
3	23	5	3
4	24	6	0
5	25	6	1
6	26	6	2
7	27	6	3
8	28	7	0
9	29	7	1
#		R0	F

En el estado 3

Lee	Interpreta	Escribe	Pasa a
0	30	7	2
1	31	7	3
2	32	8	0
3	33	8	1
4	34	8	2
5	35	8	3
6	36	9	0
7	37	9	1
8	38	9	2
9	39	9	3
#		R1	F

En este caso, es más fácil definir el autómata sin dibujos pero explícitamente así:

Q = {0,1,2,3,f}

Σ = {0,1,2,3,4,5,6,7,8,9,#}

Γ = {0,1,2,3,4,5,6,7,8,9,R}

q_I = 0

δ(i,#) = f

δ(i,d) = (i*10 + d) mod 4

h(i,#) = Ri

h(i,4) = (i*10 + d) div 4

El siguiente autómata modela la suma de dos números en base diez.

Ejemplo 2.18 Queremos definir un autómata con respuestas cuyo alfabeto de entrada es:

$$\{(x,y):\ x \in \{0,1,2,3\},\ y \in \{0,1,2,3\}\}.$$

La cadena de entrada representa 2 números en base 4. Por ejemplo si tenemos los siguientes 2 números: 112113 y 333. Este número se representaría así:

σ6	σ5	σ4	σ3	σ2	σ1
(1,0)	(1,0)	(2,0)	(1,3)	(1,3)	(3,3)

La respuesta del autómata debe ser el número que representa la suma de los dos números representados en la entrada. La respuesta también debe ser dada en base 4.

En este caso la respuesta sería.

ρ6	ρ5	ρ4	ρ3	ρ2	ρ1
1	1	3	1	1	2

Para la solución se requieren 2 estados uno en que se lleva 1 y otro en que no se lleva nada. Los estados se llaman 0 y 1 respectivamente.

$\delta(0,(x,y))$ = if $(x + y) < 4$ then 0 else 1.

$\delta(1,(x,y))$ = if $(x + y + 1) < 4$ then 0 else 1.

$h(1,(x,y)) = (x+y+1) \bmod 4$

$h(0,(x,y)) = (x+y) \bmod 4$

Ahora haría falta marcar el final de cadena con algún símbolo digamos #. Entonces agregamos las siguientes definiciones.

$\delta(0,\#) = 0$

$\delta(1,\#) = 0$

$h(1,\#) = 1$

$h(0,\#) = \lambda$

Ejercicios

1. Diseñe un autómata con respuestas que lea cadenas sobre el alfabeto {a,b,c} y como respuesta refleje la entrada intercalando después de cada símbolo un número en {0,1,2} que representa el número (módulo 3) de veces que ha salido el símbolo de entrada. Por ejemplo, si el autómata lee: abccaabcca, entonces respondería a1b1c1c2a2a0b2c0c1a1.

2. Defina un autómata con respuestas en las transiciones que reciba una cadena de la foma: $s_1 \ldots s_n$ con $s_i \in \{0,1,2\}$. El autómata debe responder con una cadena de la forma: $r_1 \ldots r_n$, donde:

   - $r_1 = s_1 \mod 3$
   - $r_i = (s_i + s_{i-1}) \mod 3$ para $i > 1$

3. Se desea definir un autómata con respuestas en las transiciones que lea cadenas de la forma: $\sigma_1\sigma_2 \ldots \sigma_n$ escriba cadenas de la forma: $\rho_1\rho_2 \ldots \rho_n$. Donde:

   $\rho_i = \sigma_i / 2$ Si tanto i como σ_i son pares

   $\rho_i = ((\sigma_i + \sigma_{i-1}) \mod 5)$ De lo contrario

   Suponga que el alfabeto de entrada es {0,1,2,3,4}. Suponga que $\sigma_o = 0$

   Ejemplo

I	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
σ	1	2	3	4	4	4	2	3	3	4	4	5	4	4	1	2	2	3
ρ	1	1	0	2	3	2	1	0	1	2	3	4	4	2	0	1	4	0

3.5 Implementación

Es relativamente fácil pensar en hacer una clase autómata. Comenzamos definiendo una clase abstracta que define la forma de procesar símbolos y cadenas. Para definir un autómata particular, se tendrían que definir los estados, y las transiciones particulares. Si el autómata tiene respuestas, se definiría también la función de respuestas. En este caso, estaríamos implementando autómatas determinísticos pero no exigimos que la función sea total.

Comenzamos con la definición de estados. Como se vio en las secciones anteriores, a veces es útil definir los estados como tuplas (e.g., el ejemplo de división por cuatro o el de secuencias de 8 bits donde el último bit verifica la paridad de los siete anteriores). Entonces, definimos el estado como una subclase de ArrayList. Se define un método selector para obtener los componentes de la tupla y un método id que se usa cuando el estado es un valor único – no una tupla.

```
public class State extends ArrayList<Object> {

   protected State(Object vals [])
   {
      super();
      int i;
      for (i=0; i < vals.length ; i++) {
         add(vals[i]);
      }
   }

   public Object nTh(int n) {
      return this.get(n);
   }

   public Object id() throws Exception {
      if (size() == 1)
         return this.get(0);
      else throw new Exception("método no válido para estados con id de tupla ");
   }
}
```

Tenemos una fábrica para evitar que se generen copias de estados iguales.

```
public class StateFactory {
   private static final Hashtable <Integer, State>
      statePool = new Hashtable <Integer , State> ();

   public static State s(Object [] a) {

      int code = Arrays.hashCode(a);
      State s = statePool.get(code);
      if(s == null) {
         s = new State(a);
         statePool.put(code, s);
      }
       return s;
   }
}
```

Un autómata se compone de un conjunto finito de estados, un alfabeto finito, unos estados finales (si no es de respuestas), un estado inicial y la función de transición. Para el alfabeto podemos suponer que acepta cadenas sobre los caracteres char de java. Como no exigimos que la función de transición sea total, si no hay una transición para un estado dado, simplemente se detendría la ejecución en ese punto. Para el conjunto de estados usamos una clase States. Esta clase se implementa como un arreglo de estados y nos permite referirnos a un estado tanto como por el estado mismo, como por su posición en el arreglo de estados.

```
public class States  extends ArrayList <State> {

    public States () {
        super ();
    }

    public int storeState(State s)throws Error {
        int pos;

        pos=  indexOf(s);
        if (pos == -1)
        {
            pos = new Integer(size());
            add(pos,s);
         }
        return pos;
    }

    public boolean has(State s) {
        return contains(s);
    }

    public int index(State s) {
        return indexOf(s);
    }

    public State getState(int index) {
        return get(index);
    }

}
```

Finalmente, definimos una clase abstracta autómata como se muestra a continuación. Lo que debe definir cada clase autómata subclase de esta es la inicialización y la función de transición delta. Vale la pena explicar en detalle esta clase. Los atributos de la clase son:

- La cinta de salida
- La cinta de entrada
- El estado actual
- Los estados
- Los estados Finales
- El estado inicial

```
public abstract class Automata {

   String inputTape;
   StringBuffer outputTape = new StringBuffer();
   State currentState;
   States states;
   ArrayList  <State> finalStates ;
   State inic;
   int tapePos;

   protected Automata(){
   }

   protected void initializeAutomata(States myStates,
                                     State myStartState,
                                     ArrayList <State> myFinalStates ) {
      states = myStates;
      inic=myStartState;
      finalStates = myFinalStates;
   }

   public void setInput(String inputString) {
      inputTape = inputString;
   }

   private void processSymbol() throws Exception {
      outputTape.append(transitionOutput(currentState,inputTape.charAt(tapePos)));
      currentState = delta(currentState,inputTape.charAt(tapePos));
      if (!states.has(currentState)) {
         throw new Exception("Estado inexistente: " + currentState);
      }
      outputTape.append(stateOutput(currentState));
      tapePos++;
   }

   public boolean processString() throws Exception
   {
      tapePos = 0;
      currentState = inic;
      outputTape = new StringBuffer();
      outputTape.append(stateOutput(currentState));

      while (tapePos < inputTape.length()) {
         processSymbol();
      }
      return finalStates.contains(currentState) && tapePos==inputTape.length();
    }

   protected  abstract State delta(State s, char c) throws Exception;

   protected  String stateOutput(State s) {
      return new String();
   }

   protected String transitionOutput(State s, char c){
      return new String();
   }

   protected static State s(Object ... nn)
   {
      return  StateFactory.s(nn);
   }

   public String getOutput(){
     return outputTape.toString();
  }

 }
```

El constructor es vacío y más bien se define un inicializador (`initializeAutómata`) para los datos del autómata. Se usa este método para definir cuáles son los estados, el estado inicial y los estados finales. Como el autómata puede correrse con varias entradas, se incluye el método `setInput`. Se provee un método apra obtener lo que se ha escrito en la cinta de salida `getInput`. Al definir una clase autómata subclase de esta clase abstracta, se deben definir funciones para la transición, para dar la respuesta en un estado y para la respuesta en una transición. Estos métodos son:

- `protected abstract State delta(State s, char c) throws Exception;`
- `protected String stateOutput(State s) {`
- `protected String transitionOutput(State s, char c){`

y se usan en los dos métodos definidos en la clase abstracta `processSymbol` y `processString`. Estos métodos son los que realizan el proceso de la cadena. Se realiza la transición correspondiente y se escribe en la cinta de salida.

En la clase autómata también definimos un método "`State s(Oject ... nn)`" para poder tener acceso a los estados a través de la fábrica de una forma más amigable. Para crear el estado representado por una tupla "(x,y)", se puede escribir simplemente `s(x,y)`.

Lo interesante es ver cómo se construyen autómatas nuevos heredando de esta clase abstracta. Mostramos la implementación del contador módulo 3, de las palabras que contienen cat, y del verificador de paridad.

Comenzamos con la clase contador módulo 3 que implementa el autómata cuyo diagrama se muestra en el Ejemplo 2.4. En este caso, definimos tres estados: s(0), s(1) y s(2). Indicamos que el estado inicial es el s(0) y que este es el único estado final.

Luego se define el método delta que define la función de transición de estados. Esta depende del valor del estado dado por st.id(), que en este caso es un entero y del carácter leído.

El otro ejemplo es el del verificador de paridad. Este es un poco más complejo.

```
public class ContadorModulo3 extends Automata {

   public ContadorModulo3()
   {
      super();

      States myStates = new States() ;
      ArrayList <State> myFinalStates = new ArrayList <State> ();
      State myStartState;
      int i;

      for (i=0; i<3; i++)
         myStates.storeState(s(i));
      myFinalStates.add(s(0));
      myStartState = s(0);
      initializeAutomata(myStates,myStartState,myFinalStates);
   }
```

```
   protected State delta(State st, char c) throws Exception{

      State resp;
      int stVal = (Integer) st.id();

      if (c == 'a') {
         resp =  s((stVal+1)%3);
      }
      else if (c == 'b'){
         resp = st;
      }
      else
         throw new Exception("Error: caracter no permitido - "+c);

      return resp;
      }
}
```

Note que sería fácil hacer un contador módulo n. Esto se haría así:

```
public class ContadorModuloN extends Automata{
   private int n;

   public ContadorModuloN(int  n)
   {
      super();
      this.n = n;
      States myStates = new States() ;

      ArrayList <State> myFinalStates = new ArrayList <State> ();

      State myStartState;

      int i;
      for (i=0; i<n; i++){
         myStates.storeState(s(i));
      }
      myFinalStates.add(s(0));
      myStartState = s(0);
      initializeAutomata(myStates,myStartState,myFinalStates);
   }

   protected State delta(State st, char c) throws Exception{

      State resp;
      int stVal = (Integer) st.id();

      if (c == 'a') {
         resp =  s((stVal+1)%n);
      }
      else if (c == 'b'){
         resp = st;
      }
      else
         throw new Exception("Error: caracter no permitido - "+c);

      return resp;
       }
}
```

De esta forma, para definir un contador módulo 3 lo haríamos así:

ContadorModuloN cont3 = new ContadorModuloN(3);

Hemos visto cómo se definen los autómatas pero no cómo se usan. Para esto, se usan los métodos para darle la cadena y para procesarla.

3.6 Limitaciones de los Autómatas Regulares

Los autómatas regulares no sirven para reconocer todos los lenguajes recursivamente enumerables. De hecho hay muchos lenguajes que no pueden ser reconocidos por ningún autómata regular. En esta sección se mostrará cómo se puede demostrar que un lenguaje no es regular usando algunos resultados conocidos.

El siguiente teorema sirve para demostrar por construcción que un lenguaje es regular. Como se verá en uno de los ejemplos, también servirá para demostrar, por contradicción que un lenguaje no es regular.

> ***Teorema 2.3.*** *Teorema: Si* **L** *y* **M** *son lenguajes regulares entonces los siguientes lenguajes también son regulares:*
>
> - **L • M**
> - **L ∪ M**
> - **L ∩ M**
> - **L***
> - **~L**
> - **L - M**

La demostración de este teorema es por construcción. Es decir, dado que se tiene un autómata que reconoce L y otro que reconoce M se construye uno que reconozca el lenguaje requerido. Por el teorema que se demostró en la Sección 2.2, se deduce que los lenguajes son cerrados bajo la complementación. Podemos probar que son cerrados bajo la unión, la concatenación y la clausura basándonos en la construcción de los autómatas que reconocen los lenguajes descritos por expresiones regulares. Para demostrar que los lenguajes regulares son cerrados bajo la intersección es necesario construir un autómata un poco más complejo y esta demostración se sale del objetivo de este texto.

Para demostrar que un lenguaje es regular se puede usar cualquiera de las siguientes técnicas:

- Construir un autómata que lo reconoce. En este caso hay que demostrar que reconoce todas las cadenas en el lenguaje y no reconoce ninguna cadena que no está en el lenguaje.
- Construir una gramática regular que lo genera.

- Construir una expresión regular que lo describe.
- Mostrar que es finito, ya que todo lenguaje finito es regular.

Se quiere encontrar una forma de determinar si un lenguaje es regular. Intuitivamente, un lenguaje que requiera un número potencialmente infinito de estados para ser reconocido, no es regular. El ejemplo clásico de un lenguaje que no es regular es $\{a^n b^n : n >= 0\}$ (i.e., las cadenas formadas por una secuencia de a´s de longitud n, seguida de una secuencia de b's de longitud n). El problema radica en que cuando un autómata comienza al leer las b's debe recordar cuántas a's han salido.

Los siguientes teoremas son expuestos en [3] y se usan para demostrar que un lenguaje no es regular. El segundo teorema es más fuerte que el primero.

Teorema 2.4. *Sea* **L** *un lenguaje regular infinito sobre un alfabeto* Σ. *Entonces existen cadenas* $\alpha \in \Sigma^*$, $\beta \in \Sigma^*$, $\gamma \in \Sigma^*$ *tales que* $\beta \neq \lambda$ *y* $\alpha\beta^n\gamma \in \mathbf{L}$ *para todo* $n \geq 0$.

Dem:

Sea **L** *un lenguaje regular e infinito entonces, existe un autómata determinístico* $\mathbf{M} = (Q, \Sigma, q_I, F, \delta)$ *tal que* $\mathbf{L(M)} = \mathbf{L}$. *Como* **L** *es infinito, existe* $\omega \in \mathbf{L}$ *con* $|\omega| = \mathbf{k}$ *tal que* $\mathbf{k} > \#.\mathbf{Q}$.

Sea $\omega = \sigma_1\sigma_2...\sigma_k$, *existen estados* $q_0, q_1, ..., q_k$ *únicos con* $q_0 = q_I$ *y* $q_k \in \mathbf{F}$ *tales que:*

$q_{t-1} \xrightarrow{\sigma_t} q_t$ *para* $1 \leq t \leq k$. *(i.e.,* $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \quad ... \quad q_{k-1} \xrightarrow{\sigma_k} q_k$*)*

Como $\mathbf{k} > \#.\mathbf{Q}$ *entonces, por el principio del palomar, debe existir al menos un estado por el que pasa más de una vez. Dicho formalmente:*

existen estados q_i *y* q_j *con* $0 \leq i < j \leq k$ *tales que* $q_i = q_j$, **(1)**

Note que: $q_0 \xrightarrow{\sigma_1...\sigma_i} q_i \xrightarrow{\sigma_{i+1}...\sigma_j} q_j \xrightarrow{\sigma_{j+1}...\sigma_k} q_k$ **(2)**

Como $i=j$*:* $q_0 \xrightarrow{\sigma_1...\sigma_i} q_i \xrightarrow{\sigma_{i+1}...\sigma_j} q_i \xrightarrow{\sigma_{j+1}...\sigma_k} q_k$ **(3)**

Sean: $\alpha = \sigma_1...\sigma_i$, $\beta = \sigma_{i+1}...\sigma_j$, $\gamma = \sigma_{j+1}...\sigma_k$ *como* $i < j$, *entonces* $\beta \neq \lambda$, *se puede, entonces, rescribir la ecuación* **(3)** *así:* $q_0 \xRightarrow{\alpha} q_i \xRightarrow{\beta} q_i \xRightarrow{\gamma} q_k$ **(4)**

Entonces, para llegar de q_0 *a* q_1, *puede leer* β *cero o más veces. Es decir:*

$q_0 \xRightarrow{\alpha} q_i \xRightarrow{\gamma} q_k$; $q_0 \xRightarrow{\alpha} q_i \xRightarrow{\beta} q_i \xRightarrow{\gamma} q_k$; $q_0 \xRightarrow{\alpha} q_i \xRightarrow{\beta} q_i \xRightarrow{\beta} q_i \xRightarrow{\gamma} q_k$;…

En general: $q_0 \xRightarrow{\alpha} q_i \xRightarrow{\beta^n} q_i \xRightarrow{\gamma} q_k$ **(5)**

Como $q_0 = q_I$ *y* $q_k \in \mathbf{F}$ *entonces* $\alpha\beta^n\gamma \in \mathbf{L}$.

□

Este teorema no sirve para demostrar que un lenguaje es regular; sirve para demostrar que no es regular. Se demuestra por contradicción: si no existen cadenas con las propiedades enunciadas, entonces el lenguaje no es

regular. En general la demostración se hace así: se supone que el lenguaje es regular y se definen las cadenas α, β y ρ de forma general y se muestra que existe un n tal que $\alpha\beta^n\rho \notin L$, llegando así a una contradicción.

Ejemplo 2.19 **$L=\{a^p$: p es primo$\}$** no es regular.

Dem: Suponga que L es regular. Entonces por el Teorema 2.4, deben existir cadenas $\alpha \in a^*$, $\beta \in a^*$ y $\rho \in a^*$ con $\beta \neq \lambda$ tales que $\alpha\beta^n\gamma \in L$ para cualquier **n**.

Sean: $\alpha=a^p$, $\beta=a^q$, $\gamma=a^r$. Como $\beta \neq \lambda$ entonces se sabe que $q \geq 1$. Se debe cumplir que para todo **n**, $p+nq+r$ es un número primo.

Sea $n=(p+2q+r+2)$ entonces $p+nq+r$ debe ser primo:

$p+nq+r$ es primo
$=$ $\langle n=p+2q+r+2 \rangle$
$p+(p+2q+r+2)q+r$ es primo
$=$ $\langle$aritmética$\rangle$
$p+pq+2q^2+rq+2q+r$ es primo
$=$ $\langle$aritmética$\rangle$
$(q+1)(p+2q+r)$ es primo
$=$ $\langle q \geq 1 \Rightarrow q+1 \geq 2$ y $(p+21+r) \geq 2 \rangle$
false

Por lo tanto, la suposición inicial (L es regular) es falsa.

El siguiente teorema establece un resultado más fuerte:

Teorema 2.5. *Sea* **L** *un lenguaje regular y* $M = (Q, \Sigma, q_I, \delta)$ *un autómata determinístico tal que* $L(M)=L$, *entonces para cualquier cadena* $\omega \in L$ *tal que* $|\omega| \geq |Q|$ *existen cadenas* $\alpha \in \Sigma^*$, $\beta \in \Sigma^*$, $\rho \in \Sigma^*$ *tales que:* $\omega=\alpha\beta\rho$, $|\alpha\beta| \leq \#.Q$, $\beta \neq \lambda$ *y* $\alpha\beta^n\rho \in L$ *para todo* $n \geq 0$.

Dem:

Sean **L** *un lenguaje regular,* $M = (Q,\Sigma,q_I, F, \delta)$ *tal que* $L(M) = L$, *y* $\omega \in L$ *con* $|\omega| \geq |Q|$.

Sean $k=|Q|$, $m= |\omega|$, $\omega = \sigma_1\sigma_2...\sigma_k...\sigma_m$.

Entonces, existen estados $q_0, q_1, ..., q_k ..., q_m$ *con* $q_0 = q_I$ *y* $q_m \in F$ *tales que:* $q_{t-1} \xrightarrow{\sigma_t} q_t$ *para* $1 \leq t \leq k$ *y* $q_k \xrightarrow{\sigma_{k+1}...\sigma_m} q_m$.

(i.e., $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \quad ... \quad q_{k-1} \xrightarrow{\sigma_k} q_k \xrightarrow{\sigma_{k+1}...\sigma_m} q_m$ *)*

Como **k = #.Q** *entonces, por el principio del palomar, debe pasar más de una vez por un estado cuando reconoce los primeros* **k** *símbolos de la cadena. Dicho formalmente: existen estados* $\mathbf{q_i}$ *y* $\mathbf{q_j}$ *con* $\mathbf{0 \le i < j \le k}$ *tales que* $\mathbf{q_i = q_j}$, ***(1)***

$$\mathbf{q_0} \xrightarrow{\sigma_1 \ldots \sigma_i} \mathbf{q_i} \xrightarrow{\sigma_{i+1} \ldots \sigma_j} \mathbf{q_j} \xrightarrow{\sigma_{j+1} \ldots \sigma_k} \mathbf{q_k} \xrightarrow{\sigma_{k+1} \ldots \sigma_m} \mathbf{q_m}$$

$$\mathbf{q_0} \xrightarrow{\sigma_1 \ldots \sigma_i} \mathbf{q_i} \xrightarrow{\sigma_{i+1} \ldots \sigma_j} \mathbf{q_i} \xrightarrow{\sigma_{j+1} \ldots \sigma_k} \mathbf{q_k} \xrightarrow{\sigma_{k+1} \ldots \sigma_m} \mathbf{q_m} \qquad \textbf{(2)}$$

Sean:

$\alpha = \sigma_1 \ldots \sigma_i$,

$\beta = \sigma_{i+1} \ldots \sigma_j$,

$\rho = \sigma_{j+1} \ldots \sigma_k \ldots \sigma_m$, *con* $\beta \neq \lambda$ *ya que* $\mathbf{i < j}$, *y con* $|\alpha\beta| \le \#.\mathbf{Q}$, *ya que* $\mathbf{j \le k}$. ***(3)***

De ***(2)*** *y* ***(3)*** *se deduce que:* $\mathbf{q_0} \xRightarrow{\alpha} \mathbf{q_i}$ ***y*** $\mathbf{q_i} \xRightarrow{\beta} \mathbf{q_i}$ ***y*** $\mathbf{q_i} \xRightarrow{\rho} \mathbf{q_m}$ ***(4)***

Hay que demostrar que para todo **n**, $\alpha\beta^n\rho \in \mathbf{L}$

Dado que $\mathbf{q_0 = q_I}$ *y* $\mathbf{q_k \in F}$ *es lo mismo que demostrar:*

$$\mathbf{q_0} \xRightarrow{\alpha} \mathbf{q_i} \curlywedge \mathbf{q_i} \xRightarrow{\beta^n} \mathbf{q_j} \curlywedge \mathbf{q_j} \xRightarrow{\rho} \mathbf{q_k}.$$

Dado ***(4)****, sólo hay que demostrar:*

$$\mathbf{q_i} \xRightarrow{\beta^n} \mathbf{q_j}.$$

Esta demostración se termina de la misma forma que la demostración del teorema anterior. La diferencia de la demostración radicó en la forma como se escogieron las subcadenas para asegurar que cumplieran con las condiciones requeridas (*ver* (*3*)).

□

Este teorema se usa así: Se supone que el lenguaje es regular y se escoge una cadena en el lenguaje de longitud suficiente para la cual al escoger cadenas que cumplan con las condiciones descritas, exista un n tal que $\alpha\beta^n\rho \notin L$.

Ejemplo 2.20 $L=\{a^n b^n : n \ge 0\}$ no es regular.

Suponga que L es regular y que M es un autómata con k estados que lo reconoce. Sea $\omega = a^k b^k$. Es claro que $\omega \in L$. Entonces por el teorema 2.5 deben existir cadenas: α, β y ρ, tales que $\alpha\beta\rho = \omega$ y $|\alpha\beta| \le k$ y tales

que para todo n, $\alpha\beta^n\rho \in L$. Como $\omega=a^kb^k$ y $|\alpha\beta|\leq k$ entonces α y β están compuestas sólo de a´s. Por lo tanto, podemos decir que:

- $\alpha=a^t$
- $\beta=a^s$
- $\rho=a^{k-(s+t)}b^k$

Donde $1\leq s+t \leq k$ y $s \geq 1$.

Es claro que: $\alpha\beta\rho=\omega$

Decir que para todo n, $\alpha\beta^n\rho \in L$ equivale a decir que para todo **n**: **t+ns+k-(s+t)=k**. Sin embargo, tomando **n=0** tenemos que:

```
   t+ns+k-(s+t)=k
≡    ⟨n=0⟩
   t+k-(s+t)=k
≡    ⟨aritmética⟩
   -s = 0
≡    ⟨s >= 1⟩
   false
```

Por lo tanto L no es regular.

Ejemplo 2.21 $L=\{\omega: \omega \in \{a,b\}^* \wedge \omega=\omega^R\}$ no es regular.

Suponga que L es regular y que M es un autómata con k estados que lo reconoce. Sea $\omega=a^kba^k$. Es claro que $\omega \in L$. Entonces por el teorema 2.5 deben existir cadenas: α, β y ρ, tales que $\alpha\beta\rho=\omega$ y $|\alpha\beta|\leq k$ y que para todo n, $\alpha\beta^n\rho \in L$. Como $\omega=a^kb^k$ y $|\alpha\beta|\leq k$ entonces α y β están compuestas sólo de a´s. Por lo tanto, podemos decir que:

- $\alpha=a^t$
- $\beta=a^s$
- $\rho=a^{k-(s+t)}ba^k$

Donde $1\leq s+t \leq k$ y $s \geq 1$.

Es claro que: $\alpha\beta\rho=\omega$

Como no se hicieron suposiciones adicionales con respecto a las cadenas, decir que para todo n, $\alpha\beta^n\rho \in L$ equivale a decir que para todo n: t+ns+k-(s+t)=k. Sin embargo, tomando n=0 tenemos que:

t+ns+k-(s+t)=k

= ⟨n=0⟩

t+k-(s+t)=k

= ⟨aritmética⟩

-s = 0

= ⟨s >= 1⟩

false

Por lo tanto L no es regular.

El teorema 2.3 puede usarse para demostrar que un lenguaje no es regular. El siguiente ejemplo ilustra este uso.

Ejemplo 2.22 $L=\{a^nb^m: n\geq 0 \wedge m\geq 0 \mid n\neq m\}$ no es regular[3].

Suponga que L es regular, entonces por el teorema 2.3 a*b*-L debe ser regular:

a*b*-L

= ⟨def de L y de a*b*⟩

$\{a^nb^m : n\geq 0 \wedge m\geq 0\}-\{a^nb^m : n\geq 0 \wedge m\geq 0 \mid n\neq m\}$

= ⟨ operaciones sobre conjuntos ⟩

$\{a^nb^m : n\geq 0 \wedge m\geq 0 \mid n=m\}$

= ⟨Leibniz⟩

$\{a^nb^n : n\geq 0\}$

Pero se sabe que este lenguaje no es regular. Por lo tanto se llega a una contradicción, invalidando la suposición inicial: L es regular.

Por lo tanto, L NO es regular

Es interesante notar que, en cambio, $L=\{a^nb^m: n\geq 0 \wedge m\geq 0\}$ sí es regular. Note que no se está diciendo nada acerca de la relación entre n y m. De hecho L=a*b*.

Ejercicios

1. Dados los siguientes lenguajes, si son regulares o no y demuéstrelo.

   a. $L = \{a^nb^m: n\geq 0 \wedge m\geq 0 \mid m < n\}$

   b. $L = \{a^nb^m: n\geq 0 \wedge m\geq 0 \mid m \leq n\}$

[3] En notación de [2] esto sería: $\{n.m:nat \mid n\neq m: a^nb^m\}$

c. $L = \{a^n b^m : n \geq 0 \wedge m \geq 0 \mid m > n\}$

d. $L = \{a^n a^m : n \geq 0 \wedge m \geq 0 \mid m = n\}$

e. $L = \{a^n b^m : n \geq 0 \wedge m \geq 0 \mid m \geq n\}$

f. $L = \{a^n b^m : n \geq 0 \wedge m \geq 0 \mid 5 \geq m \geq n\}$

g. $L = \{a^n b^m : n \geq 0 \wedge m \geq 0 \mid 5 \leq m \leq n\}$

h. $L = \{a^n b^m : n \geq 0 \wedge 10 \geq m \geq 0 \mid m \neq n\}$

i. $L = \{a^n b^m : n \geq 0 \wedge 10 \geq m \geq 0 \mid m \neq n, m \leq 5\}$

2. Demuestre que si un lenguaje, L, es regular entonces el reverso del lenguaje, L^R, también lo es. Se define L^R como $\{\omega : \omega^R \in L\}$.
3. Definimos el lenguaje de los prefijos de L como: $L_{pre} = \{\omega$: existe un β tal que $\omega\beta \in L\}$. Demuestre que si L es regular entonces L_{pre} es regular.
4. Definimos el lenguaje de los sufijos de L como: $L_{suf} = \{\omega$: existe un β tal que $\beta\omega \in L\}$. Demuestre que si L es regular entonces L_{suf} es regular.
5. Demuestre que todo lenguaje finito es regular.
6. Demuestre que si L es infinito y es reconocido por un autómata determinístico de estados finitos con k estados, entonces existe una cadena $\omega \in L$ tal que $|\omega| > k$.