Autómatas y lenguajes formales

Víctor Mijangos de la Cruz

vmijangosc @ ciencias.unam.com

II. Lenguajes regulares

Lenguajes regulares

Operaciones sobre cadenas

Para entender los lenguajes regulares retomaremos las operaciones:

1. Concatenación Si $w_1 = (a_1, ..., a_n)$ y $w_2 = (b_1, ..., b_m)$, entonces $w_1 \cdot w_2 = (a1, ..., a_n, b_1, ...b_m)$
2. Adversación/Unión/Suma: $w_1 + w_2 = w_1 \oplus rw_2$
3. Potencia: Si $w \in \Sigma^*$, entonces: 1) $w^0 = \epsilon$; 2) $w^{n+1} = w^n \cdot w$
4. Estrella Kleene: w^* incluye las potencias de 0 a ∞.
5. Operador más: w^+ incluye las potencias de 1 a ∞.

Expresión regular

Expresión regular

El conjunto de expresiones regulares o regex, E, se define como:

1. $\emptyset \in E$ — Base
2. $\epsilon \in E$
3. $a \in E, \forall a \in \Sigma$
4. Si $r_1, r_2 \in E$, entonces $r_1 + r_2 \in E$ — Recursión
5. Si $r_1, r_2 \in E$, entonces $r_1 \cdot r_2 \in E$
6. Si $r \in E$, entonces $r^* \in E$

Construcción de expresiones regulares

Podemos ver que una expresión de la forma:

$$(a + b) * \cdot c^*$$

es regular.

Prueba.

- Por 3, a, b y c son expresiones regulares.
- Por 4, $a + b = r_1$ es expresión regular.
- Por 6, $(a + b)^* = r_1^* = r_2$ y $c^* = r_3$ son expresiones regulares.
- Por 5, $r_2 \cdot r_3 = (a + b)^* \cdot r_3$ es expresión regular.

□

Denotación de regex

Denotación de una expresión regular

Dada una expresión regular, definimos su denotación de manera recursiva como:

1. $[[\emptyset]] = \emptyset$ — Base
2. $[[\epsilon]] = \{\epsilon\}$
3. $[[a]] = \{a : a \in \Sigma\}$
4. Si r_1 y r_2 son regex, entonces $[[r_1 + r_2]] = [[r_1]] \cup [[r_2]]$ — Recursión
5. Si r_1 y r_2 son regex, entonces $[[r_1 \cdot r_2]] = [[r_1]] \cdot [[r_2]]$
6. Si r es regex, entonces $[[r^*]] = [[r]]^*$

Las operaciones sobre conjuntos son las mismas operaciones que hemos definido sobre lenguajes.

Denotaciones

Supónagse que se tienen las regex 0 y 1 del alfabeto $\Sigma = \{0, 1\}$, entonces:

- $[[0]] = \{0\}$, y $[[1]] = \{1\}$
- $[[0 + 1]] = [[0]] \cup [[1]] = \{0, 1\}$
- $[[0 \cdot 1]] = [[0]] \cdot [[1]] = \{01\}$
- $[[0^*]] = [[0]]^* = \{\epsilon, 0, 00, 000, 0000, ...\}$
- $[[(0 + 1) \cdot 1]] = \{01, 11\}$

Los conjuntos resultantes de las denotaciones de una expresión regular son sub conjuntos de las cadenas de Σ^*, por tanto, son lenguajes.

Equivalencia de regex

Equivalencia de expresiones regulares

Dos expresiones regulares R y S son equivalentes, denotado $R \equiv S$, si y sólo si $[[R]] = [[S]]$.

Ejemplo: $(R+S)^* \equiv (R^* \cdot S^*)^*$

$$
\begin{aligned}
[[R^* \cdot S^*]]^* &= \bigcup_{n=0} \Big([[R^*]] \cdot [[S^*]] \Big)^n = \bigcup_{n=0} \Big(\bigcup_{n=0} [[R]]^n \cdot \bigcup_{n=0} [[S]]^n \Big) \\
&= \bigcup_{n=0} [[R]]^n \cdot [[S]]^n = \{\epsilon\} \cup [[R]] \cup [[S]] \cup [[R]]^2 \cup [[S]]^2 \cup [[RS]] \cup [[SR]] \ldots \\
&= \bigcup_{n=0} \Big([[R]] \cup [[S]] \Big)^n = \bigcup_{n=0} [[R+S]]^n = [[(R+S)]]^* \\
&= [[(R+S)^*]]
\end{aligned}
$$

Lenguaje regular

Lenguaje regular

Un lenguaje regular L es la denotación de una expresión regular R. Esto es $L = [[R]]$. Cuando conocemos la expresión regular lo denotamos como $L(R)$.

Ejemplo: Un lenguaje regular finito es $L = \{01, 11\}$, construido por la expresión regular $(0 + 1) \cdot 1$.
Otro lenguaje regular es $L(R) = \{\epsilon, 0, 1, 01, 11, 00, 001, 111, 000, 1111, 0001...\}$, donde $R = (0 + 1)^* \cdot 1^*$

Gramática de lenguajes regulares

Gramática regular o tipo 3

Sea $G = (\Sigma, \Delta, S, R)$ una gramática y $L = L(G)$ el lenguaje generado por esta gramática. L es regular si las producciones de la gramática son de la forma:

$$X \to a \cdot Y | \epsilon$$

Donde $Y \in \Delta \cup \epsilon$ y $a \in \Sigma$.

Ejemplo: Podemos tomar la siguiente gramática regular $G = (\Sigma, \Delta, S, R)$ con alfabeto $\Sigma = \{0, 1\}$ y las producciones:

$$S \to 0X_1 | 1X_1$$
$$X_1 \to 1X_1 | \epsilon$$

Que genera el lenguaje regular $L(G) = [[(0 + 1) \cdot 1^*]]$

Cerradura de unión

Lema (cerradura de unión)

Los lenguajes regulares son cerrados bajo la unión; i.e., si L_1, L_2 regulares, entonces $L_1 \cup L_2$ es regular.

Prueba.

Sean $L_1 = [[R_1]]$ y $L_2 = [[R_2]]$, entonces

$$L_1 \cup L_2 = [[R_1]] \cup [[R_2]] = [[R_1 + R_2]]$$

Y ya que $R_1 + R_2$ es una expresión regular,

$$L_1 \cup L_2$$

es un lenguaje regular. □

Cerrados bajo operaciones

Teorema (cerradura bajo concatenación)

Los lenguajes regulares son cerrados bajo concatenación. Esto es, la concatenación de dos o más lenguajes regulares es un lenguaje regular.

Teorema (cerradura bajo Kleene)

Los lenguajes regulares son cerrados bajo estrella de Kleene; i.e., si L regular, L^* es regular.

Prueba.

Sea $L = [[R]]$ para una regex R. Claramente $L^* = [[R]]^* = [[R^*]]$ □

Teorema (cerradura bajo inverso)

Los lenguajes regulares son cerrados bajo inverso L^R.

Asociatividad y conmutatividad de regex

Asociatividad

Sean R, S, T expresiones regulares, entonces la unión y la concatenación son asociativas:

- $R + (S + T) = (R + S) + T$
- $R \cdot (S \cdot T) = (R \cdot S) \cdot T$

Conmutatividad

Sean R y S expresiones regulares, entonces la unión entre éstas es conmutativa:

$$R + S = S + R$$

Identidad y nulos

Identidad

La identidad en la unión es $\emptyset$ y en la concatenación ϵ; esto es, se cumple para toda expresión regular R:

1. $\emptyset + R = R + \emptyset = R$
2. $\epsilon \cdot R = R \cdot \epsilon = R$

Nulos

Si R es una expresión regular, $\emptyset$ es el elemento nulo para la concatenación. Esto es:

$$\emptyset \cdot R = R \cdot \emptyset = \emptyset$$

Distributividad

Distributividad

Sean R, S y T expresiones regulares, entonces la concatenación y la unión distribuyen:

$$R \cdot (S + T) = R \cdot S + R \cdot T$$
$$(S + T) \cdot R = S \cdot R + T \cdot R$$

Teorema

Si L, M y N son lenguajes regulares, entonces:

$$L \cdot (M \cup N) = L \cdot M \cup L \cdot N$$

Por ejemplo, $L = [[0]]$, $M = [[\epsilon]]$ y $N = [[1^*]]$, entonces:

$$\begin{aligned} L \cdot (M \cup N) &= [[0]] \cdot \big([[\epsilon]] \cup [[1^*]]\big) = [[0]] \cdot \big([[\epsilon + 1^*]]\big) \\ &= [[0 \cdot (\epsilon + 1^*)]] = [[0 \cdot \epsilon + 0 \cdot 1^*]] = L \cdot (M \cup N) = L \cdot M \cup L \cdot N \end{aligned}$$

Idempotencia y cerraduras

Idempotencia de Kleen

El operador de estrella de Kleene es idempotente; esto es, $(R^*)^* = R^*$

Leyes de cerradura

Tenemos las siguientes leyes y fórmulas para definir operadores:

1. $\emptyset^* = \epsilon$ — Cerradura de $\emptyset$
2. $\epsilon^* = \epsilon$ — Cerradura de ϵ
3. $R^+ = R \cdot R^* = R^* \cdot R$ — Definición $+$
4. $R? = R + \epsilon$ — Definición ?

Equivalencia de regex

Ley de cerradura y unión

Dado dos expresiones regulares R y S, tenemos que:

$$(R + S)^* = (R^* S^*)^*$$

Equivalencia de expresiones regulares

Dos expresiones regulares R y S se dice que son similares, $R \equiv S$, si y sólo si $L(R) = L(S)$.

Ejemplo: $R = a^+ b$ y $S = (\emptyset + a)^*(\emptyset \cdot b + a \cdot b)$ son equivalentes:

$$\begin{aligned} S &= (\emptyset + a)^*(\emptyset \cdot b + a \cdot b) = (a)^*\big((\emptyset + a) \cdot b\big) \\ &= a^*(a \cdot b) = (a^* \cdot a) \cdot b \\ &= a^+ \cdot b \end{aligned}$$

Por tanto $R \equiv S$.

Autómatas finitos

Autómata finito

Los autómatas finitos presentan las siguientes ventajas:

1. Sirven como herramienta esencial en la fase lexicográfica de un compilador.
2. Un simulador de un autómata finito es rápido y eficientemente.
3. Requieren de una cantidad fija de memoria, por lo que facilita problemas de uso y asignación de memoria.
4. Los resultados teóricos nos permiten simplificar los autómatas para hacerlos más eficientes.

Autómata finito determinista

Autómata finito determinista

Un autómata finito determinista (AFD) es un 5-tupla $A = (Q, \Sigma, \delta, q_0, F)$ donde:

1. $Q = \{q_0, q_1, ..., q_n\}$ es un conjunto finito de estados.
2. $\Sigma = \{a_1, a_2, ..., a_m\}$ es un conjunto de símbolos.
3. $\delta : Q \times \Sigma \rightarrow Q$ es la función de transición-
4. $q_0 \in Q$ es el estado inicial.
5. $F \subseteq Q$ es un conjunto de estados finales.

Ejemplo de AFD

Sea el AFD $A = (Q, \Sigma, \delta, q_0, F)$ determinado por los elementos:

1. $Q = \{q_0, q_1, q_2\}$ cuenta con 3 estados. Y un estado final $F = \{q_2\}$.
2. $\Sigma = \{0, 1\}$ alfabeto binario.
3. La función δ está dada de la siguiente forma:

$$\delta(q_0, 0) = q_1$$
$$\delta(q_0, 1) = q_0$$
$$\delta(q_1, 0) = q_1$$
$$\delta(q_1, 1) = q_2$$
$$\delta(q_2, 0) = q_2$$
$$\delta(q_2, 1) = q_2$$

Diagrama de transición

Diagrama de transición: Es una gráfica dirigida $G = (V, E)$ definida como:

1. Cada estado $q \in Q$ se asocia a un nodo $n \in V$
2. Para $q, p \in Q$ y $a \in \Sigma$, si $\delta(q, a) = p$, entonces hay una transición $e_{p,q} \in E$ del nodo asociado a q al asociado a p.
3. El nodo inicial q_0 tiene una flecha sin nodos precedentes.
4. Los nodos terminales $q \in F$ se marcan con doble círculo.

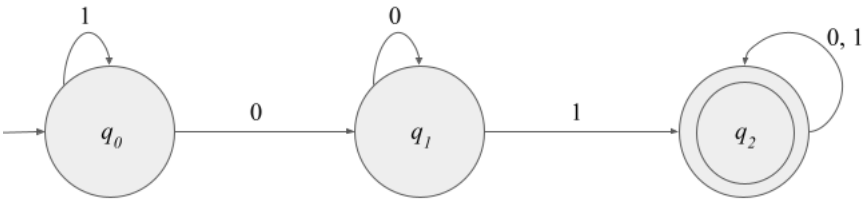


Tabla de estados

Tabla de estados: Representa la función δ de forma tabular T como:

1. Los renglones se asocian a los estados $q \in Q$.
2. Las columnas representan los símbolos $a \in \Sigma$.
3. El estado inicial se marca con $\rightarrow$ y el fina con $*$.
4. Cada entrada de la tabla es de la forma: $T_{i,j} = \delta(q_i, a_j)$

	0	**1**
$\rightarrow \mathbf{q_0}$	q_1	q_0
$\mathbf{q_1}$	q_1	q_2
$*\mathbf{q_2}$	q_2	q_2

Extensión de la función de transición

Extensión de la función de transición

Dada la función de transición $\delta : Q \times \Sigma \to Q$ de un AFD, su extensión, denotada $\hat{\delta} : Q \times \Sigma^* \to Q$ se define como:

- $\forall q \in Q$, $\hat{\delta}(q, \epsilon) = q$ (Relfexividad)
- Si $w = x \cdot a \in \Sigma^*$, $a \in \Sigma$, $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$ (Transitividad)

Ejemplo: Del autómata anterior, podemos ver que la función delta aplicada a la cadena $010 \in \Sigma^*$ es:

$$\hat{\delta}(q_0, \epsilon) = q_0$$
$$\hat{\delta}(q_0, 0) = \delta(\hat{\delta}(q_0, \epsilon), 0) = \delta(q_0, 0) = q_1$$
$$\hat{\delta}(q_0, 01) = \delta(\hat{\delta}(q_0, 0), 1) = \delta(q_1, 1) = q_2$$
$$\hat{\delta}(q_0, 010) = \delta(\hat{\delta}(q_0, 01), 0) = \delta(q_2, 0) = q_2$$

Lenguajes y autómatas

Lenguaje generado por un autómata

Dado un autómata finito determinista $A = (Q, \Sigma, \delta, q_0, F)$ el lenguaje generado por este, denotado $L(A)$, se define como:

$$L(A) = \{w \in \Sigma^* : \hat{\delta}(q_0, w) \in F\}$$

Ejemplo: El autómata anterior genera el lenguaje que contiene las cadenas con al menos un 0 y un 1. De hecho podemos ver que:

$$L(A) = \{w \in \Sigma^* : w = 1^*00^*1(0+1)^*\}$$

Puesto que es claro ver que $\hat{\delta}\big(q_0, 1^*00^*1(0+1)^*\big) = q_2 \in F$.

Autómata finito no determinista

Autómata finito no determinista

Un autómata finito no determinista (AFN) es una 5-tupla $A = (Q, \Sigma, \delta, q_0, F)$ donde:

1. $Q = \{q_0, q_1, ..., q_n\}$ es un conjunto finito de estados.
2. $\Sigma = \{a_1, a_2, ..., a_m\}$ es conjunto de símbolos.
3. $q_0 \in Q$ es el estado inicial.
4. $F \subseteq Q$ es el conjunto de estados finales.
5. La función de transición es $\delta : Q \times \Sigma \to \mathcal{P}(Q)$, tal que $\delta(q, a) \subseteq Q$.

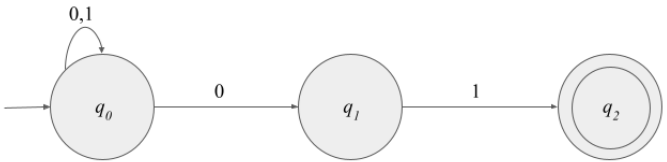
El AFN, al igual que el AFD, toma como entrada un estado y un símbolo, pero la transición no está definida. Así, puede transitar a más de un estado observando un mismo estado y un mismo símbolo.
La salida de la función, es por tanto el conjunto de estados posibles a los que se puede mover.

Ejemplo de autómata no determinista

Considérese el autómata $A = (Q, \Sigma, \delta, q_0, F)$ definido como $\Sigma = \{0, 1\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$ y las transiciones dadas por:

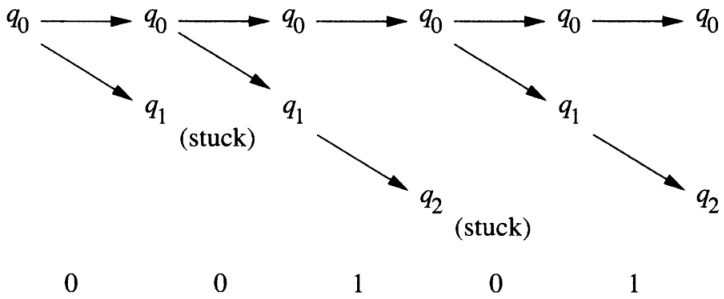
$$\delta(q_0, 0) = \{q_0, q_1\}$$
$$\delta(q_0, 1) = \{q_1\}$$
$$\delta(q_1, 1) = \{q_2\}$$

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



Ciclos en los AFN

Un AFN puede entrar en ciclos, o probar varios caminos. El autómata se atora (*stuck*) cuando no logra hacer una transición, llegar a un estado final o acabar con la cadena.
Por ejemplo, dado el autómata anterior, tenemos que para la cadena 00101 se da el proceso:



Función de transición extendida

Función de transición extendida para AFN

Dado un AFN $A = (Q, \Sigma, \delta, q_0, F)$, la extensión de δ es la función $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$ definida como:

- $\hat{\delta}(q, \epsilon) = \{q\}$
- Si $w = x \cdot a$, $x \in \Sigma^*$, $a \in \Sigma$, dado que $\hat{\delta}(q, x) = \{p_1, p_2, ..., p_k\}$ entonces:

$$\hat{\delta}(q, w) = \bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, ..., r_m\}$$

Del autómata anterior, tenemos que para 001:

$$\hat{\delta}(q_0, \epsilon) = \{\epsilon\}$$
$$\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$$
$$\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$$
$$\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_1\}$$

Lenguaje de un AFN

Lenguaje generado por un AFN

Dado un AFN $A = (Q, \Sigma, \delta, q_0, F)$, el lenguaje definido por A está dado como:

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

El autómata definido anteriormente define el lenguaje:

$$L = [[(0+1)^*01]]$$

Para demostrar esto, se debe mostrar que:

1. $q_1 \in \hat{\delta}(q_0, w)$ si y sólo si w termina en 0.
2. $q_2 \in \hat{\delta}(q_0, w)$ si y sólo si w termina en 01.

Demostración del ejemplo

Lema

En el autómata anterior $q_1 \in \hat{\delta}(q_0, w)$ si y sólo si w termina en 0.

Supóngase $q_1 \in \hat{\delta}(q_0, w)$, si $|w| = 1$ entonces $w = 0$. Si $|w| > 1$ entonces $w = x \cdot a$, pero ya que se pasa a q_1 sólo con 0, entonces $a = 0$ y $w = x \cdot 0$.

Supóngase que w termina en 0. Si $|w| = 1$, entonces $w = 0$ y por tanto, $\delta(q_0, 0) = \{q_0, q_1\}$. Y por tanto $q_1 \in \hat{\delta}(q_0, 0)$. Si $|w| > 1$ entonces $w = x \cdot 0$, y de aquí
$\hat{\delta}(q_0, x \cdot 0) = \{q_0\} \cup \{q_1\} = \{q_0, q_1\}$.

Demostración

Lema

En el autómata anterior $q_2 \in \hat{\delta}(q_0, w)$ si y sólo si w termina en 01.

Supóngase $q_2 \in \hat{\delta}(q_0, w)$, entonces $|w| \geq 2$. Si $|w| = 2$ entonces $w = 01$, pues para llegar a q_2 tienen que consumirse estos símbolos. Si $|w| > 2$, entonces $w = x \cdot 01$, pues sólo se puede pasar a q_2 consumiendo 0 y 1.

Supóngase que $w = x \cdot 01$ termina en 0. Descompóngase en $w = y \cdot 1$, $y = z \cdot 0$. Pero y termina en 0, y por lema anterior, entonces $q_1 \in \hat{\delta}(q_0, y)$ y por definición de la extensión $\hat{\delta}(q_0, y \cdot 1) = \cup_{i=1}^{k} \delta(p_i, 1)$ con $p_i \in \hat{\delta}(q_0, y)$. De aquí tenemos que $q_2 \delta(q_1, 1) \in \hat{\delta}(q_0, y \cdot 1)$.

Corolario

De aquí se obtiene que el lenguaje del autómata es $L = [[(0 + 1)^* 01]]$.

De AFN a AFD

Algorithm Conversión de AFN a AFD

1: **procedure** $\text{To-AFD}(N = (Q_N, \Sigma, \delta_N, q_0, F_N))$
2: $Q_D = \mathcal{P}(Q_N)$, los nuevos estados son conjuntos de estados del AFN
3: $\{q_0\}$ es el inicial.
4: $F_D = \{S \subseteq Q : S \cap F_N \neq \emptyset\}$
5: Para toda $S \subseteq Q$ y $a \in \Sigma$:

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$$

6: Los estados inalcanzables se desechan.
7: **return** $D = \{Q_D, \Sigma, \delta_D, \{q_0\}, F_D\}$
8: **end procedure**

Ejemplo de conversión

Sea el AFN dado por el autómata anterior, con la tabla de transición δ_N:

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

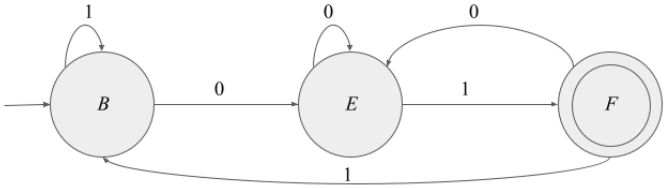
1. $Q_D = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$ y $\{q_0\}$ es inicial.
2. $F_D = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$
3. La tabla de transición para δ_D será construida como:

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

Ejemplo de conversión

4. Renombremos los estados y veamos cuáles son inalcanzables:

	Estado	**0**	**1**
$\emptyset$	A	A	A
$\rightarrow \{q_0\}$	$\rightarrow$B	E	B
$\{q_1\}$	C	A	D
$*\{q_2\}$	*D	A	A
$\{q_0, q_1\}$	E	E	F
$*\{q_0, q_2\}$	*F	E	B
$*\{q_1, q_2\}$	*G	A	D
$*\{q_0, q_1, q_2\}$	* H	E	F



Equivalencia de autómatas

Equivalencia de autómatas

Sean A y B dos autómatas finitos, entonces A es equivalente a B si y sólo si $L(A) = L(B)$.

Teorema

Sea $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ un autómata finito no-determinista y sea $D = (Q_D, \Sigma, \delta, \{q_0\}, F_D)$ el autómata determinista construido por el procedimiento anterior. Entonces N y D son equivalentes.

Para demostrar este teorema, se probará que $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$ para toda cadena $w \in \Sigma^*$. Lo haremos por inducción sobre w.
Si $w = \epsilon$, entonces tenemos que:

$$\hat{\delta}_D(\{q_0\}, \epsilon) = \{q_0\} = \hat{\delta}_N(q_0, \epsilon)$$

Por lo que se cumple la igualdad.

Continuación de demostración

Ahora, sea $w = x \cdot a$, con $a \in \Sigma, x \in \Sigma^*$. La hipótesis de inducción nos dice que:

$$\hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x) = \{p_1, p_2, ..., p_k\}$$

Donde p_i son los estados que resultan de estas transiciones. Por definición de la extensión de la función de transición:

$$\hat{\delta}_N(q_0, w) = \bigcup_{i=1}^{k} \delta_N(p_i, a)$$

Por la construcción del autómata determinista $\delta_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$ pues:

$$\hat{\delta}_D(\{q_0\}, w) = \delta_D\big(\hat{\delta}_D(\{q_0\}, x), a\big) = \delta_D(\{p_1, ..., p_k\}, a)$$
$$= \bigcup_{i=1}^{k} \delta_N(p_i, a) = \hat{\delta}_N(q_0, w)$$

Podemos concluir que $L(N) = L(D)$.

AFDs y AFNs

Corolario

Un lenguaje regular L es aceptado por un Autómata Finito Determinista si y sólo si es aceptado por un Autómata Finito no Determinista.

El teorema anterior muestra que hay si L es aceptado por un AFN, podemos constuir un AFD que acepte el mismo lenguaje.
Ahora, si L es aceptado por un AFD $D = (Q, \Sigma, \delta_D, q_0, F)$, podemos construir un AFN donde: si $\delta_D(q, a) = p$, entonces $\delta_N(q, a) = \{p\}$. Claramente, esta construcción muestra que el AFN acepta el mismo lenguaje que el AFD original.

Trnasiciones ϵ

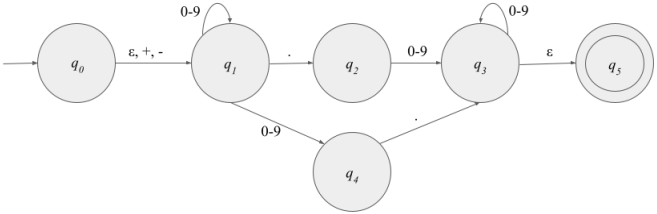
AFN con transiciones ϵ

Un Autómata Finito no Determinista con transiciones ϵ es un AFN $A = (Q, \Sigma, \delta, q_0, F)$ donde la función de transición está definida como:

$$\delta : Q \times \Sigma \cup \{\epsilon\} \to \mathcal{P}(Q)$$

Esto es, puede tomar al símbolo ϵ como argumento.

	ϵ	+,-	.	**0-9**
$\to q_0$	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1, q_4\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_3	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_4	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
$*q_5$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



Cerradura epsilon

Cerradura ϵ

Dado un ϵ-AFN, la cerradura ϵ de los estados, $\text{ECLOSE}()$, se define como:

- $q \in \text{ECLOSE}(q)$
- Si $p \in \text{ECLOSE}(q)$ y $r \in \delta(p, \epsilon)$, entonces $r \in \text{ECLOSE}(q)$.

Ejemplo: Del autómata anterior, tenemos dos transiciones ϵ:

- $q_0 \in \text{ECLOSE}(q_0)$ y además $\delta(q_0, \epsilon) = \{q_1\}$, por lo que $\text{ECLOSE}(q_0) = \{q_0, q_1\}$.
- $q_3 \in \text{ECLOSE}(q_3)$ y como $\delta(q_3, \epsilon) = \{q_5\}$, entonces $\text{ECLOSE}(q_3) = \{q_3, q_5\}$
- Para todos los demás casos, $\text{ECLOSE}(q_i) = \{q_i\}$.

Extensión de función de transición

Extensión de la función de transición de ϵ-AFN

Si $A = (Q, \Sigma, \delta, q_0, F)$ es un ϵ-AFN, la extensión de su función de transición, $\hat{\delta}$, se define como:

- $\hat{\delta}(q, \epsilon) = \text{ECLOSE}(q)$
- Si $w = x \cdot a$ (con $a \neq \epsilon$) y $\hat{\delta}(q, x) = \{p_1, ..., p_k\}$, entonces:
  1. $\{r_1, r_2, ..., r_m\} = \bigcup_{i=1}^{k} \delta(p_i, a)$
  2. $\hat{\delta}(q, w) = \bigcup_{j=1}^{m} \text{ECLOSE}(r_j)$

Ejemplo de extensión de función

Del autómata anterior, podemos ver que $\hat{\delta}(q_0, 5.6)$ es:

- $\hat{\delta}(q_0, \epsilon) = \text{ECLOSE}(q_0) = \{q_0, q_1\}$
- $\delta(q_0, 5) \cup \delta(q_1, 5) = \emptyset \cup \{q_1, q_4\}$ y
  $\hat{\delta}(q_0, 5) = \text{ECLOSE}(q_1) \cup \text{ECLOSE}(q_4) = \{q_1, q_4\}$
- $\delta(q_1, .) \cup \delta(q_4, .) = \{q_2\} \cup \{q_3\} = \{q_2, q_3\}$ y
  $\hat{\delta}(q_1, 5.) = \text{ECLOSE}(q_2) \cup \text{ECLOSE}(q_3) = \{q_2\} \cup \{q_3, q_5\} = \{q_2, q_3, q_5\}$
- $\delta(q_2, 6) \cup \delta(q_3, 6) \cup \delta(q_5, 3) = \{q_3\}$ y
  $\text{ECLOSE}(q_3) = \{q_3, q_5\}$

Por tanto: $\hat{\delta}(q_0, 5.6) = \{q_3, q_5\}$.

Lenguaje aceptado por un ϵ-AFN

Lenguaje aceptado por un ϵ-AFN

Sea $A = (Q, \Sigma, \delta, q_0, F)$ un AFN con transiciones ϵ. Definimos el lenguaje aceptado por este autómata comoa:

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

El autómata anterior acepta secuencias de dígitos con puntos decimales. Como 5.6, 0.9999, -1.5, etc. Vemos que:

$$L(A) = [[('+' + '-')?\Big(\big((0-9)^*.(0-9)^+\big) + \big((0-9)^+.(0-9)^*\big)\Big)]]$$

De ϵ-AFN a AFD

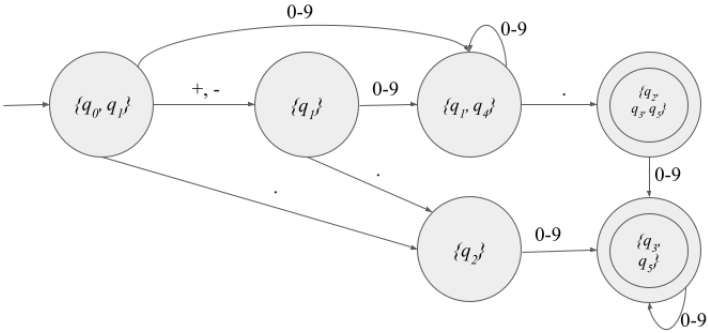
Algorithm Conversión de ϵ-AFN a AFD

procedure $\textsc{To-AFD}(E = (Q_E, \Sigma, \delta_E, q_0, F_E))$
  $Q_D = \{\textsc{Eclose}(S) : S \subseteq \mathcal{P}(Q_N)\}$
  $q_D = \textsc{Eclose}(q_0)$
  $F_D = \{S \subseteq Q_D : S \cap F_E \neq \emptyset\}$
  for $a \in \Sigma$, $S \in Q_D$ **do**
    $\{p_1, p_2, .., p_k\} = S$
    $\{r_1, r_2, ..., r_m\} = \bigcup_{i=1}^{k} \delta_E(p_i, a)$
    $\delta_D(S, a) = \bigcup_{j=1}^{m} \textsc{Eclose}(r_j)$
  end for
  return $D = \{Q_D, \Sigma, \delta_D, q_D, F_D\}$
end procedure

Ejemplo de AFD desde ϵ-AFN

Del autómata de los dígitos, podemos ver que:

- $Q_D = \{\{q_0, q_1\}, \{q_1\}, \{q_2\}, \{q_1, q_4\}, \{q_3, q_5\}, \{q_2, q_3, q_5\}\}$
- $q_D = \text{ECLOSE}(q_0) = \{q_0, q_1\}$ y $F_D = \{\{q_2, q_3, q_5\}\{q_3, q_5\}\}$



Demostración del algoritmo

Teorema

Un lenguaje L es aceptado por un ϵ-AFN si y sólo si L es aceptado por un DFA.

$<$ Sea $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ es un AFD, con $L = L(D)$, si $\delta_D(q, a) = p$, entonces $\delta_E(q, a) = \{p\}$. Además $\delta_E(q, \epsilon) = \emptyset$. Por tanto, tenemos un ϵ-AFN que acepta el mismo lenguaje que D.

$>$ Definamos $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ un ϵ-AFN, tal que $L = L(E)$. Construimos por el método anterior el AFD:

$$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$$

Demostraremos que $\hat{\delta}_E(q_0, w) = \hat{\delta}_D(q_D, w)$, por inducción sobre w.

Base: Si $w = \epsilon$, entonces $\hat{\delta}_E(q_0, \epsilon) = \text{ECLOSE}(q_0) = q_D = \hat{\delta}_D(q_D, \epsilon)$.

Continuación demostración

Inducción: Sea $w = x \cdot a$, $a \in \Sigma$. Por hipótesis de inducción $\hat{\delta}_E(q_0, x) = \hat{\delta}_D(q_D, x) = \{p_1, p_2, ..., p_k\}$.
Por definición de $\hat{\delta}_E$ en ϵ-ANF, tenemos que:

- $\{r_1, r_2, ..., r_m\} = \bigcup_{i=1}^{k} \delta_E(p_i, a)$, y
- $\hat{\delta}_E(q_0, w) = \bigcup_{j=1}^{m} \textsc{Eclose}(r_j)$

Por otro lado, por construcción de D, tenemos que:

$$\hat{\delta}_D(q_D, w) = \delta_D(\{p_1, p_2, ..., p_k\}, a) = \bigcup_{j=1}^{m} \textsc{Eclose}(r_j)$$

Esto es $\hat{\delta}_E(q_0, w) = \hat{\delta}_D(q_D, w)$. Por lo que ambos autómatas aceptan las mismas cadenas.

Autómatas de Moore y Mealy

Autómata de Moore

Una autómata de Moore es una 6-tupla $A = (Q, \Sigma, O, \delta, \lambda, q_0)$ tal que:

- $Q = \{q_0, q_1, ..., q_n\}$ un conjunto finito de estados con estado inicial q_0.
- $\Sigma = \{a_1, a_2, ..., a_m\}$ conjunto de símbolos.
- $O = \{o_1, o_2, ..., o_k\}$ conjunto de símbolos de emisión0.
- $\delta : Q \times \Sigma \to Q$ función de transición.
- $\lambda : Q \to O^*$ función de transducción.

Autómata de Mealy

Autómata de Mealy

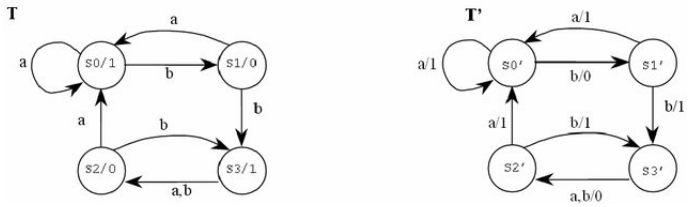
Una autómata de Mealy es una 6-tupla $A = (Q, \Sigma, O, \delta, \lambda, q_0)$ tal que:

- $Q = \{q_0, q_1, ..., q_n\}$ un conjunto finito de estados con estado inicial q_0.
- $\Sigma = \{a_1, a_2, ..., a_m\}$ conjunto de símbolos.
- $O = \{o_1, o_2, ..., o_k\}$ conjunto de símbolos de emisión0.
- $\delta : Q \times \Sigma \to Q$ función de transición.
- $\lambda : Q \times O^* \to Q$ función de transducción.

Equivalencia de autómatas de Moore y Mealy

Un autómata de Moore puede ser transformado en uno de Mealy simplemente con la regla que transforma las funciones de transducción como:

- Si $\delta(q, a) = p$ y $\lambda(p) = o$, entonces $\lambda'(q, o) = p$.
- Todos los demás elementos pertenecen iguales.



Lenguajes regulares y autómatas finitos

Autómatas finitos y lenguajes regulares

Los autómatas finitos están ampliamente ligados a los lenguajes regulares y, por tanto, a las expresiones regulares.
Hasta ahora hemos mostrado la relación entre autómatas finitos:

1. Todo AFN puede ser convertido en un AFD que acepte el mismo lenguaje.
2. Todo ϵ-AFN puede ser convertido en un AFD que acepte el mismo lenguaje.
3. Todo AFD puede convertirse en un AFN o ϵ-AFN que acepte el mismo lenguaje.

Para mostrar que se relacionan con las expresiones regulares, mostraremos lo siguiente:

1. Dado un DFA, podemos encontrar una regex que defina el lenguaje.
2. Dada una regex, podemos encontrar un ϵ-AFN que acepte el lenguaje.

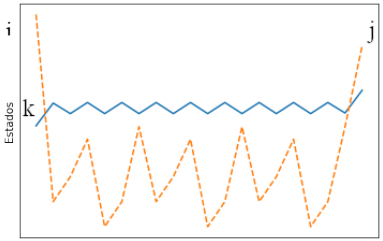
De DFA a expresión regular

Teorema

Sea $A = (Q, \Sigma, \delta, q_0, F)$ un AFD tal que $L = L(A)$: entonces, existe una expresión regular R tal que $L = [[R]]$.

Supóngase que $Q = \{1, 2, ..., n\}$ donde los estados están dados por naturales. Y definamos una expresión regular $R^k_{i,j}$ de la siguiente forma:

El lenguaje de $R^k_{i,j}$ contiene todas las cadenas $w \in \Sigma^*$, tal que w se conforma de símbolos del camino del estado i a j y este camino no tiene nodos intermedios que sean mayores a k. Construiremos $R^k_{i,j}$ utilizando inducción sobre k.



Demostración (Base)

Base: Si $k = 0$, entonces no hay estados intermedios (no hay nodo 0). Por tanto, hay dos caso:

1. Una arcos de i hacia j ($i \neq j$).
2. Un ciclo del nodo i hacia sí mismo ($i = j$).

Si $i \neq j$, sólo 1 es posible. Siendo $a \in \Sigma$, notamos que:

1. Si no hay un símbolo a del estado i a j, entonces $R^0_{i,j} = \emptyset$.
2. Si sólo hay un símbolo a, entonces $R^0_{i,j} = a$.
3. Si se dan varios arcos con símbolos $a_1, a_2, ..., a_k$, entonces $R^0_{i,j} = a_1 + a_2 + ... + a_k$.

Si $i = j$, tenemos lazos hacia un mismo nodo. Podemos temer los casos anteriores, pero como se queda en el mismo nodo, podemos tener también ϵ:

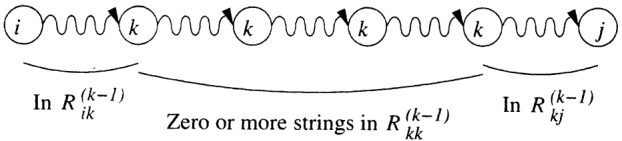
1. $R^0_{i,j} = \emptyset + \epsilon = \epsilon$.
2. $R^0_{i,j} = a + \epsilon$.
3. $R^0_{i,j} = a_1 + a_2 + ... + a_k + \epsilon$.

Demostración (Inducción)

Base: Si $k > 0$, entonces hay dos casos:

1. El camino no pasa por k, por lo que el lenguaje está dado por $R_{i,j}^{k-1}$
2. El camino pasa al menos una vez por k, por lo que se puede dividir en secciones:
   1. De i a k sin pasar por k, el lenguaje será $R_{i,k}^{k-1}$.
   2. De k a j, el lenguaje será $R_{k,j}^{k-1}$.
   3. De k a k (cuando pasa más de una vez por k), el lenguaje será $R_{k,k}^{k-1}$.

   La expresión regular final de i a j estará dada como $R_{i,k}^{k-1} \cdot \left(R_{k,k}^{k-1}\right)^* R_{k,j}^{k-1}$.



Finalmente, tendremos que $R_{i,j}^{k} = R_{i,j}^{k-1} + R_{i,k}^{k-1} \cdot \left(R_{k,k}^{k-1}\right)^* R_{k,j}^{k-1}$.

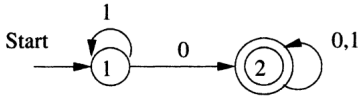
De AFD a regex

Algorithm Conversión de AFD a regex

procedure TO-REGEX($A = (Q, \Sigma, \delta, q_0, F)$)
  $\{1, 2, ..., n\} \leftarrow Q$
  if $i \neq j$ **then** $\quad i, j \in \{1, 2, ..., n\}$
    $R^0_{i,j} \leftarrow \emptyset, a, a_1 + ... + a_k$
  end if
  if $i = j$ **then** $\quad i, j \in \{1, 2, ..., n\}$
    $R^0_{i,i} \leftarrow \epsilon, a + \epsilon, a_1 + ... + a_k + \epsilon$
  end if
  for k **from** 1 **to** n **do**
    $R^k_{i,j} \leftarrow R^{k-1}_{i,j} + R^{k-1}_{i,k} \cdot \left(R^{k-1}_{k,k}\right)^* \cdot R^{k-1}_{k,j}$
  end for
  return $R^n_{1,n}$
end procedure

Ejemplo

Transformemos el siguiente autómata en una expresión regular:



k=0	Resultado
$R^0_{1,1}$	$1+\epsilon$
$R^0_{1,2}$	0
$R^0_{2,1}$	$\emptyset$
$R^0_{2,2}$	$0+1+\epsilon$

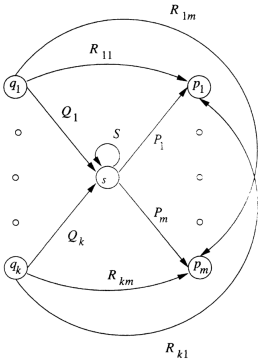
$L(A) = [[1^*0(0+1)^*]]$

k=1	Result.	R
$R^1_{1,1}$	$(1+\epsilon)+(1+\epsilon)(1+\epsilon)^*(1+\epsilon)$	1^*
$R^1_{1,2}$	$0+(1+\epsilon)(1+\epsilon)^*0$	1^*0
$R^1_{2,1}$	$\emptyset+\emptyset(1+\epsilon)^*(1+\epsilon)$	$\emptyset$
$R^1_{2,2}$	$(0+1+\epsilon)+\emptyset(1+\epsilon)^*0$	$0+1+\epsilon$

k=2	Result.	R
$R^2_{1,1}$	$1^*+(1^*0)(0+1+\epsilon)^*\emptyset)$	1^*
$R^2_{1,2}$	$1^*0+(1^*0)(0+1+\epsilon)^*(0+1+\epsilon)$	$1^*0(0+1)^*$
$R^2_{2,1}$	$\emptyset+(0+1+\epsilon)(1+\epsilon)^*\emptyset$	$\emptyset$
$R^2_{2,2}$	$(0+1+\epsilon)+(0+1+\epsilon)(1+\epsilon)^*(0+1+\epsilon)$	$(0+1)^*$

Conversión de AFD a Regex eliminando estados

El método descrito con anterioridad tiene una complejidad de $O(n^3)$ pues tiene que calcular los casos $R_{i,k}, R_{k,k}$ y $R_{k,j}$ para toda $i, j, k = 1, ..., n$.
Otro método para transformar un AFD en una expresión regular es por medio de **Reducción de estados**.

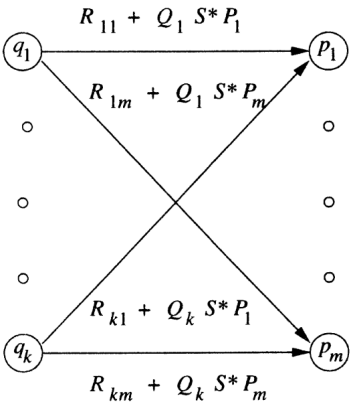


- Sea $s \in Q$ con estados predecesores $q_1, .., q_k$ y sucesores $p_1, ..., p_m$
- De una transición de q_i a s, sea Q_i la regex.
- De una transición de s a p_i, sea P_i la regex.
- De una transición de q_i hacia p_j sea la regex $R_{i,j}$

Eliminación de estados

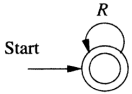
Si de este AFD se elimina el estado s, se tienen las siguientes situaciones:

- Los arcos que incluyen a s se borran.
- Se agregan regex de q_i a s y de s a p_j: $Q_i \cdot S^* \cdot P_J$ es la nueva regex para transición de q_i a p_j.
- La regex $R_{i,j}$ de q_i a p_j también se incluye: $R_{i,j} + Q_i \cdot S^* \cdot P_J$.

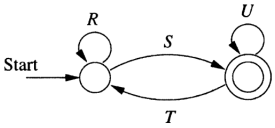


Procedimiento por reducción de estados

- Para todo $q \in Q$ se producen expresiones regulares en sus transiciones.
- Se eliminan todos los estados con el procedimiento anterior, excepto q_0 inicial, y q_F final.
- Si $q_0 \in F$ se tendrá autómata de un estado y la regex será R^*:



- Si $q_0 \neq q_F$, el autómata obtenido tendrá dos estados y la regex será $(R + SU^*T)^*SU^*$:



- La regex final será la suma de todas las expresiones resultado de 2) y 3)

AFD a Regex por reducción de estados

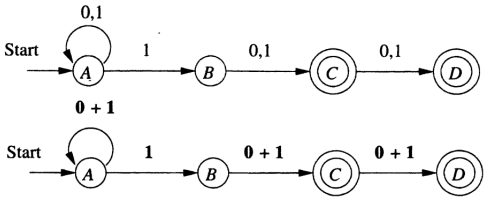
Algorithm Conversión de AFD a regex

procedure $\textsc{To-Regex}(A = (Q, \Sigma, \delta, q_0, F))$
  $A', regex \leftarrow \textsc{Transitions-to-Regex}(A), \emptyset$
  for $q \in Q \setminus \{q_0, q_F\}$ **do**
    $A' \leftarrow \textsc{Eliminate-State}(A')$
  end for
  $R, S, T, U \leftarrow \textsc{Regex}(q_0, q_0), \textsc{Regex}(q_0, q_F), \textsc{Regex}(q_F, q_0), \textsc{Regex}(q_F, q_F)$
  if $q_0 \in F$ **then**
    $regex \leftarrow regex + R^*$
  end if
  if $q_0 \neq q_F$ **then**
    $regex \leftarrow regex + (R + SU^*T)^*SU^*$
  end if
  return *regex*
end procedure

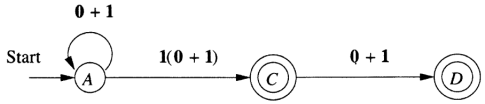
$q_F \in F$

Ejemplo

Considérese el siguiente autómata y su reducción de transiciones a expresiones regulares:

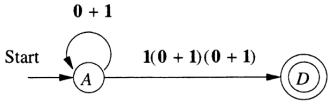


- Se elimina B: predecesor A $Q_A = 1$ y sucesor B $P_C = 0 + 1$, sin transición $A \to B$, $R_{A,B} = \emptyset$ y sin lazo en B, $S = \emptyset$. Por tanto la regex es $\emptyset + 1\emptyset^*(0 + 1) = 1(0 + 1)$

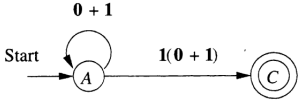


Ejemplo

- Se elimina C, $Q_A = 1(0+1)$, $P_D = 0+1$, $R_{A,D} = \emptyset$ y $S = \emptyset$, por lo que la regex es $\emptyset + 1(0+1)\emptyset^*(0+1) = 1(0+1)(0+1)$



- La regex aquí es $\left((0+1)^* + 1(0+1)^2\emptyset^*\emptyset\right)^* 1(0+1)^2\emptyset^* = (0+1)^*1(0+1)^2$
- Se elimina D, como no hay transiciones subsecuentes, sólo quedan las anteriores:

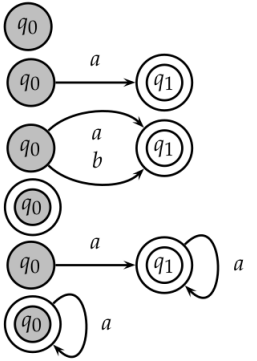


- Auí la regex es $\left((0+1) + 1(0+1)\emptyset^*\emptyset\right)^* 1(0+1)\emptyset^* = (0+1)^*1(0+1)$

La regex final es $(0+1)^*1(0+1)^2 + (0+1)^*1(0+1)$

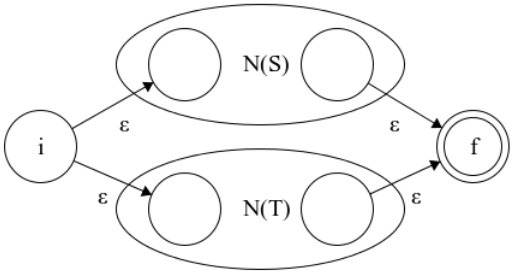
Construcción de autómatas

La construcción de autómatas a partir de algunas expresiones regulares sobre autómatas se puede hacer de la siguiente forma:

FSA, A	$L(A)$
	$\emptyset$
	$\{a\}$
	$\{a, b\}$
	$\{\epsilon\}$
	$a^+ = \{a, aa, aaa, aaaa, \ldots\}$
	$a^* = \{\epsilon, a, aa, aaa, aaaa, \ldots\}$

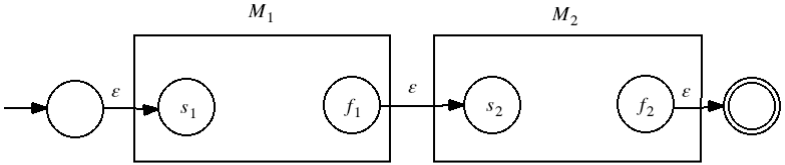
Unión de autómatas

Dado autómatas A_1 y A_2 con lenguajes $L_1 = L(A_1)$ y $L_2 = L(A_2)$, podemos construir un autómata A tal que $L(A) = L_1 \cup L_2$ de la siguiente forma:



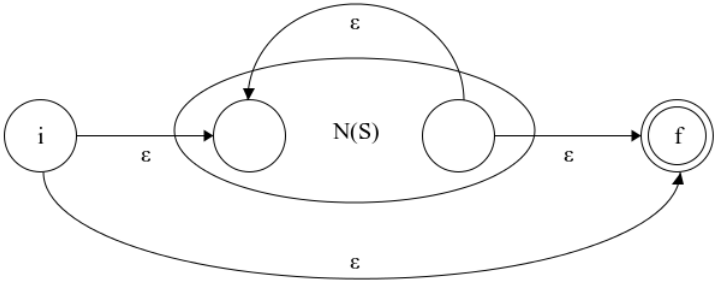
Concatenación de autómatas

Dado autómatas A_1 y A_2 con lenguajes $L_1 = L(A_1)$ y $L_2 = L(A_2)$, podemos construir un autómata A tal que $L(A) = L_1 \cdot L_2$ de la siguiente forma:



Unión de autómatas

Dado un autómata B con lenguaje $L = L(B)$, podemos construir un autómata A tal que $L(A) = L^*$ de la siguiente forma:

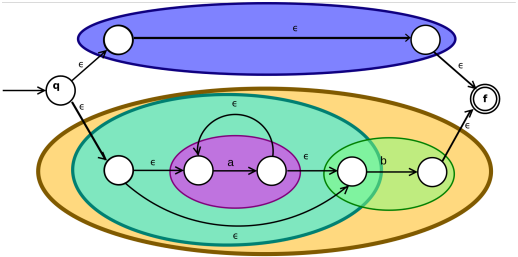


Algoritmo de construcción de Thompson

Con la ayuda de los autómatas finitos, podemos usar las expresiones regulares para hacer búsqueda de patrones dentro de cadenas.

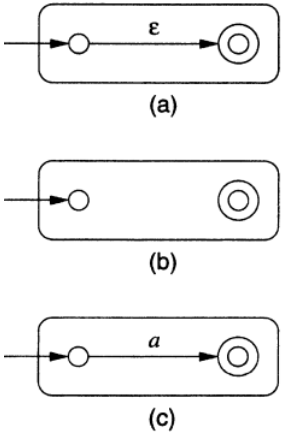
Algoritmo de construcción de Thompson

El algoritmo de construcción de Thompson es un algoritmo que compila una expresión regular y la convierte en un ϵ-AFN.



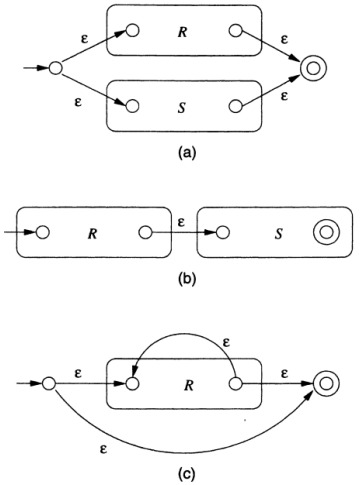
Reglas para pasar de regex a AFN (Base)

- Si $R = \epsilon$, se crea transición ϵ del inicio al fin.
- Si $R = a$, se crea transición de inicio a fin con a.
- Si $R = \emptyset$, no hay transición.



Reglas para pasar de regex a AFN (Recursión)

- Si R y S son dos regex, $R + S$ corresponde al autómata con transiciones ϵ de q hacia el inicio de R y S y de los finales de R y S hacia un nuevo estado final.
- Para $R{\cdot}S$, se agrega una transición ϵ de los finales de R al inicial de S.
- Para R^* se crea un nuevo estado final y uno inicial conectados por una transición ϵ, además se conecta el nuevo inicio al inicio de R y los finales del R al nuevo final. Finalmente, se hace una conexión ϵ del final de R a su inicio.



Algoritmo de construcción de Thompson (Base)

Algorithm Algoritmo de Construcción de Thompson

```
1: procedure THOMPSON-CONSTRUCTION(regex)
2:     if regex = ϵ then
3:         δ ← (q, ϵ, q_f ∈ F)
4:     end if
5:     if regex = ∅ then
6:         Q ← q, q_f
7:         δ ← ∅
8:     end if
9:     if regex = a ∈ Σ then
10:        δ ← (q, a, q_f ∈ F)
11:    end if
12: end procedure
```

Algoritmo de construcción de Thompson (Inducción)

Algorithm Algoritmo de Construcción de Thompson

1: **procedure** THOMPSON-CONSTRUCTION(regex)
2: **if** regex $= R + S$ **then**
3: $\delta \leftarrow (q, \epsilon, q_0), (q, \epsilon, q_0'), (q_R, \epsilon, q_f), (q_S, \epsilon, q_f)$ $q_R \in F_R, q_S \in F_S, q_0, q_0'$ iniciales
4: **end if**
5: **if** regex $= R \cdot S$ **then**
6: $\delta \leftarrow (q_R, \epsilon, q)$ $q_R \in F_R, q \in S$ inicial
7: **end if**
8: **if** regex $= R^*$ **then**
9: $\delta \leftarrow (q, \epsilon, q_f), (q_R, \epsilon, q_0), (q, \epsilon, q_0), (q_R, \epsilon, q_f)$ $q_R \in F_R, q_0$ inicial
10: **end if**
11: **end procedure**

Ejemplo

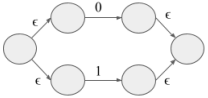
Construiremos un autómata finito a partir de la expresión $(0+1)^*1(0+1)$

1. En primer lugar, usamos los pasos base para 0 y para 1:

$$\delta(q_1, 0) = q_2$$
$$\delta(q_3, 1) = q_4$$

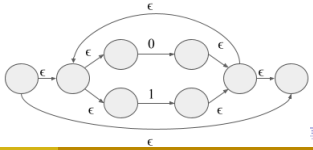
2. Para el caso de $(0+1)$ tenemos que agregar:

$$\delta(q_5, \epsilon) = q_1 \qquad \delta(q_5, \epsilon) = q_3$$
$$\delta(q_2, \epsilon) = q_6 \qquad \delta(q_4, \epsilon) = q_6$$



3. Ahora para $(0+1)^*$:

$$\delta(q_7, \epsilon) = q_8 \qquad \delta(q_6, \epsilon) = q_5$$
$$\delta(q_7, \epsilon) = q_5 \qquad \delta(q_6, \epsilon) = q_8$$



Equivalencia de regex y autómatas finitos

Teorema

Sea $L = [[R]]$ un lenguaje regular, entonces existe un autómata finito A, tal que $L = L(A)$.

Para demostrar esto se usa la construcción de Thompson e inducción sobre las expresiones regulares.
Base: Si la regex es $\epsilon, \emptyset$ o un símbolo $a \in \Sigma$, la base de la construcción de Thompson da el autómata correspondiente, pues es claro que $L(A) = [[R]]$.
Inducción: Tenemos los tres casos:
1) $R + S$, por h.i. sabemos que existe autómata para R y para S tal que $L(A_R) = [[R]]$ y $L(A_S) = [[S]]$, la construcción de Thompson nos da el autómata tal que $L(A_R) \cup L(A_S) = [[R]] \cup [[S]] = [[R + S]]$.
2) $R \cdot S$, sabemos que existe A_R y A_S tal que $L(A_R) = [[R]]$ y $L(A_S) = [[S]]$, la construcción de Thompson nos da el autómata tal que $L(A_R) \cdot L(A_S) = [[R]] \cdot [[S]] = [[R \cdot S]]$.
3) Finalmente, para R^*, sabemos por h.i. que existe A tal que $L(A) = [[R]]$ y por la construcción de Thompson tenemos $L(A)^* = [[R]]^* = [[R^*]]$.

Equivalencia y minimización de autómatas

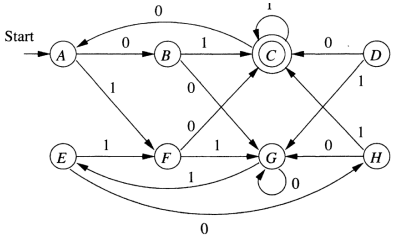
Equivalencia de estados

Estados equivalentes

Dos estados p y q son equivalentes si $\forall w \in \Sigma^*$, $\hat{\delta}(p, w) \in F$ si y sólo si $\hat{\delta}(q, w) \in F$.

Estado distinguible

Decimos que un estado $p \in Q$ es distinguible de $q \in Q$ si $\exists w \in \Sigma^*$ tal que $\hat{\delta}(p, w)$ y $\hat{\delta}(p, w)$ uno es aceptado y el otro no.



- C y G no son equivalentes, pues $\hat{\delta}(C, \epsilon) \in F$.
- A y G no son equivalentes, pues $\hat{\delta}(A, 01) = C \in F$ y $\hat{\delta}(G, 01) = E \notin F$
- A y E son equivalentes.

Algoritmo para encontrar estados distinguibles

Algorithm Algoritmo de llenado de tabla

```
1: procedure TABLE-FILLING(p, q estados)
2:     T ← tabla con estados
3:     if p ∈ F, q ∉ F then                                   (BASE)
4:         T[p, q] ← x                                        {p, q} distinguibles.
5:     else                                                   (INDUCCIÓN)
6:         for a ∈ Σ do
7:             s, r ← δ(p, a), δ(q, a)
8:             if s y r distinguibles then
9:                 T[p, q] ← x                                {p, q} distinguibles.
10:            end if
11:        end for
12:    end if
13: end procedure
```

Ejemplo

Creemos una tabla para el autómata anterior y llenemos los cuadros de los estados distinguibles:

B	X						
C	X	X					
D	X	X	X				
E		X	X	X			
F	X	X	X		X		
G	X	X	X	X	X	X	
H	X		X	X	X	X	X
	A	**B**	**C**	**D**	**E**	**F**	**G**

- Ya que solo $C \in F$ es distinguible de todos los otros estados.
- C, H distinguibles y $\delta(E, 0) = H$, $\delta(F, 0) = C$, E, F distinguibles. C, G dist. y $\delta(D, 0) = C, \delta(H, 0) = G$, así D, H dist.
- E, F dist. y $\delta(G, 1) = E, \delta(A, 1) = F$, ent. G, A distinguibles.
- Al final sólo los pares $\{A, E\}, \{B, H\}$ y $\{D, F\}$ son equivalentes.

Correctud del algoritmo

Teorema

Si dos estados $p, q \in Q$ no son encontrados por el algoritmo anterior (vacío en la tabla), entonces p y q son equivalentes.

Sea $A = (Q, \Sigma, \delta, q_0, F)$ un DFA. Demostramos por inducción sobre la cadena que distingue los estados que el algoritmo encuentra todos los pares distinguibles.

- $w = \epsilon$, entonces, s.p.g., p final y q no, y serían encontrados en la base del algoritmo.
- Si $|w| = n \geq 1$, sea $w = a_1 a_2 \cdots a_n$ la cadena más corta que distingue p y q. Y sean $\delta(p, a_1) = r$, $\delta(q, a_1) = s$ tal que r y s son distinguibles por $a_2 a_3 \cdots a_n$, que tiene longitud menor a la de w, por lo que $\{r, s\}$ es un par distinguible encontrado por el algoritmo. Pero por el paso inductivo del algoritmo $\{p, q\}$ es encontrado y marcado.

Entonces los únicos pares que no encuentra el algoritmo son los pares equivalentes.

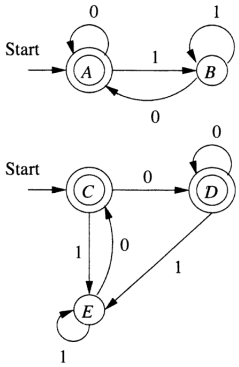
Equivalencia de lenguajes

Algorithm Algoritmo de equivalencia

1: **procedure** EQUIVALENCE(L_1, L_2)
2: $A_1, A_2 \leftarrow$ TO-DFA(L_1), TO-DFA(L_2)
3: $A \leftarrow A_1 \cup A_2$ con estados iniciales q_0, p_0
4: $T \leftarrow$ TABLE-FILLING(A)
5: **if** $(q_0, p_0) \notin T$ **then**
6: **return** True (L_1 y L_2 son equivalentes)
7: **else**
8: **return** False
9: **end if**
10: **end procedure**

Ejemplo

La unión de los Autómatas genera 5 estados:



- A,B y A,E son distinguibles, así como C,E y C,B, además de D,E y D,B
- A,C eq., A,D eq., B,E eq., y C,D eq.
- Ya que A y C iniciales, $L(A_1) = L(A_2)$

B	X			
C		X		
D		X		
E	X		X	X
	A	B	C	D

Equivalencia de estados como relación

Proposición

La equivalencia de estados es una relación de equivalencia.

Reflexividad: Cada estado es equivalente a sí mismo trivialmente.
Simetría: Si p equivalente a q, claramente q equivalente a p.
Transitividad: Sean $\{p, q\}$ y $\{q, r\}$ equivalentes. Sin $\{p, r\}$ son distinguibles entonces $\exists w \in \Sigma^*$ t.q. (s.p.g.) $\hat{\delta}(p, w) \in F$ y $\hat{\delta}(p, w) \notin F$. Se presentan dos casos:

- Si $\hat{\delta}(q, w) \in F$, $\{q, r\}$ es distinguible.
- Si $\hat{\delta}(q, w) \notin F$, p, q es distinguible.

En cualquier caso, hay una contradicción. Por tanto $\{p, r\}$ son equivalentes.

Corolario

Si $\equiv$ representa la equivalencia de estados, entonces $Q/\equiv$ es una partición de Q (espacio de estados).

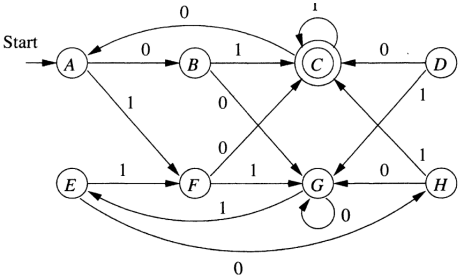
Minimización de AFD

Algorithm Algoritmo de minimización

procedure MINIMIZE($A = (Q, \Sigma, \delta, q_0, F)$)
  $T \leftarrow$TABLE-FILLING(A)
  $Q_{equiv} \leftarrow$EQUIVALENT-STATES(T)
  $Q/\equiv \leftarrow$PARTITION(Q, Q_{equiv})
  $\overline{q_0} \leftarrow [q_0]$
  $\overline{F} \leftarrow [q_f]$
  for $[q] \in Q/\equiv, a \in \Sigma$ **do**
    if $[q] \neq [p]$ **and** $\delta(q, a) \in [p]$ **then**
      $\gamma \leftarrow \delta([q], a) = [p]$
    end if
  end for
end procedure

Ejemplo

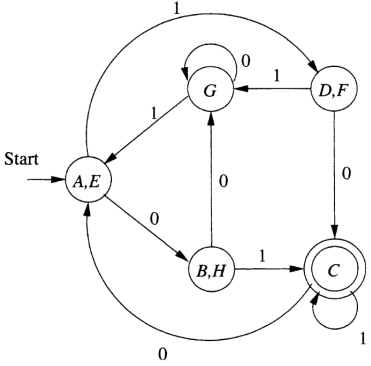
Usaremos el autómata anterior, al que hemos aplicado ya el algoritmo de llenado de tabla que repetimos aquí:



- Sabemos que los estados equivalentes son $\{A, E\}, \{B.H\}$ y $\{D, F\}$.
- $Q/\equiv = \{[A], [B], [D], [C], [G]\}$, con $[A] = \{A, E\}, [B] = \{B.H\}, [D] = \{D, F\}, [C] = \{C\}$ y $[G] = \{G\}$

Ejemplo (continuación)

- $\overline{q_0} = [A] = \{A, E\}$
- $\overline{F} = \{[C]\} = \{\{C\}\}$
- $\delta(A, 0) = B$ y $\delta(E, 0) = H$, ent, $\gamma([A], 0) = [B]$; $\delta(A, 1) = F, \delta(E, 1) = F$, ent. $\gamma([A], 1) = [D]$; $\delta(D, 0) = C, \delta(F, 0) = C$, ent. $\gamma([D], 0) = [C]$, etc...



Demostración del algoritmo

Teorema

Si A es un AFD y M el AFD construido por el método de minimización, entonces M tiene un número menor de estados que cualquier otro ADF equivalente a A.

Sea N un DFA t.q. $L(A) = L(N)$ y $|Q_N| < |Q_M|$. Sea $p \in Q_M$, entonces $\exists w = a_1 a_2 \cdots a_k \in \Sigma^*$ t.q. $\hat{\delta}(q_M, w) = p$ y $\hat{\delta}(q_N, w) = q \in Q_N$. Ya que $L(M) = L(N)$ los estados iniciales $\{q_M, q_N\}$ son indistinguibles.

Y por construcción el sucesor bajo el símbolo a_1 son indistinguibles y así de manera sucesiva. Por tanto, p y q son indistinguibles, lo que contradice el hecho de que $|Q_N| < |Q_M|$. Por tanto debe ser que $|Q_N| = |Q_M|$.

Autómatas finitos y lenguajes regulares

Hemos visto que dado un lenguaje regular L es posible encontrar un autómata finito (ϵ-AFN) que acepte el lenguaje. Asimismo, es posible construir una expresión regular a partir de un atuómata finito (AFD). También hemos visto que es posible convertir un AFN en AFD y viceversa. Por tanto, tenemos que:

Teorema

Dado un lenguaje $L \subseteq \Sigma^*$ son equivalentes:

- L es un lenguaje regular.
- L es aceptado por un autómata finito.
- L es aceptado por un autómata finito determinista.
- L es aceptador por un autómata finito determinista de tamaño mínimo.

Concluimos que los lenguajes regulares son aquellos lenguajes que pueden ser representados por un autómata finito (determinista o no determinista).

Lema del bombeo

Lenguajes regulares

No todos los lenguajes son regulares. Por ejemplo, el siguiente lenguaje no es regular:

$$L = \{0^n 1^n : n > 0\}$$

No se puede diseñar un autómata o una expresión regular que describa este lenguaje.
La razón de no poder describirlo con un **autómata** es que el autómata tendría que guardar **memoria** de cuántos 0's ha visto para poder replicar ese número de 1's.
Pero los autómatas finitos **no cuentan con memoria**.

Una forma de determinar cuándo un lenguaje no es regular es a partir del **lema del bombeo**.

Lema de bombeo para lenguajes regulares

Teorema (Lema del bombeo)

Sea L un lenguaje regular; entonces $\exists n > 0$ (que depende de L) tal que $\forall w \in L$ (con $|w| \geq n$) $w = xyz$ con x, y, z subcadenas que cumplen:

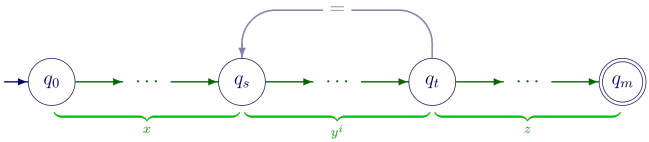
1. $y \neq \epsilon$
2. $|xy| \leq n$
3. $\forall k \geq 0, xy^k z \in L$

Demostración ($w = xyz$)

-**Definición de Autómata:** Sea L un lenguaje regular, por lo que tiene un AFD asociado. Sea $A = (Q, \Sigma, \delta, q_0, F)$ el AFD con menor número de estados tal que $L = L(A)$. Supóngase que el número de estados de A es $|\mathbf{Q}| = \mathbf{n}$.
-**Cadena:** Tómese $w \in L$ tal que $|w| \geq n$ con $w = a_1 a_2 \cdots a_m$, con $m \geq n$ y $a_i \in \Sigma$.
-**Repetición de estados:** $\forall i \in \{0, 1, 2, ..., n\}$ sea $\hat{\delta}(q_0, a_1 a_2 \cdots a_i) = p_i$; nótese que $p_0 = q_0$. Claramente para estos $n+1$ estados no es posible que toda p_i sea distinta, pues A tiene sólo n estados. Por tanto, $\exists 0 \leq i < j \leq n$ t.q. $\mathbf{p_i} = \mathbf{p_j}$.
-**Separación de w:** De aquí, podemos ver que w puede separarse en 3 subcadenas $\mathbf{w = xyz}$ tales que:

- $x = a_1 a_2 \cdots a_i$ (que va de p_0 a p_i)
- $y = a_{i+1} a_{i+2} \cdots a_j$ (que va de p_i a p_j)
- $z = a_{j+1} a_{j+2} \cdots a_m$ (que va de p_j a p_n)

Demostración ($y \neq \epsilon$ y $|xz| \leq n$)



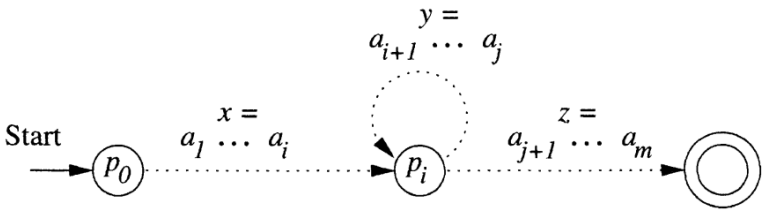
-**Longitudes de subcadenas:** Se deben notar tres casos relevantes:

- Se debe notar que si $i = 0$, entonces $x = \epsilon$.
- Asimismo, si $j = n = m$, entonces $z = \epsilon$.
- Sin embargo $\mathbf{y \neq \epsilon}$, pues hemos mostrado que $i < j$, por lo que $|y| = j - i \neq 0$.

Tenemos $|x| = i$ y $|y| = j - i$, por lo que $|xy| = j - i + i = j$, pero como en la forma en que hemos elegido estos casos, tenemos que $j < n$, por lo que $\mathbf{|xy| \leq n}$.

Demostración ($xy^kz \in L$)

-**Bombeo de y es parte del lenguaje:** Por inducción sobre k (veces que se bombea):
Sea $k = 0$, tal que $xy^kz = xz$, entonces A va de $p_0 = q_0$ por x hasta $p_i = p_j$ y de aquí hacía $p_n \in F$. Por tanto $\mathbf{xz} \in \mathbf{L}$.
Inductivamente, si tomamos $xy^{k+1}z = xy^kyz$, como xy^kz y xyz son aceptados por A, xy^k pasa de p_0 a p_j; por su parte yz va de p_i a $p_n \in F$, ya que $p_i = p_j$, $xy^{k+1}z$ va del estado inicial al final. Por tanto $\mathbf{xy^kz} \in \mathbf{L}, \forall k \geq 0$.



Lenguajes no regulares

Proposición

El lenguaje definido como $L = \{a^n b^n : n > 0\}$ no es un lenguaje regular.

Mostraremos que este lenguaje tiene cadenas que no cumplen el *lema del bombeo*.
Sea n la constante del lema del bombeo y tómese $w = a^n b^n$. Separando $w = xyz$ tal que $y \neq \epsilon$ y $|xz| \leq n$.
Ya que xy es un prefijo de w, ent. $x = a^l$ y $y = a^m$, $l + m \leq n$.
El lema del bombeo nos dice que $xz = xy^0 z \in L$, entonces $a^l b^n$ debería ser parte de L, pero $l \neq n$, por lo que $xz = a^l b^n \notin L$.
Por tanto, $L = \{a^n b^n : n > 0\}$ no es un lenguaje regular.