Procesadores del Lenguaje I

Antonio Falcó

Índice general

Parte I

Preliminares

Capítulo 1

Alfabetos y Lenguajes

El objetivo de esta primera parte es el introducir la notación formal básica absolutamente necesaria para la compresión del resto de temas.

1.1. Cadenas y Lenguajes

Las **cadenas de caracteres**, también llamadas **palabras**, son la piedra angular en la Ciencia de la Computación. El alfabeto sobre el cual las cadenas o palabras se definen puede variar con respecto a la aplicación que se esté tratando. Para nuestros propósitos definiremos un **alfabeto** como un conjunto finito de símbolos que denotaremos usualmente por letras griegas mayúsculas como Σ o Γ.

Ejemplo 1. $\Sigma_1 = \{0, 1\}$.

$\Sigma_2 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$.

$\Gamma = \{0, 1, x, y, z\}$.

Nótese que, puesto que un alfabeto es simplemente un conjunto finito no vacío, dados dos alfabetos Σ_1 y Σ_2 se tiene:

1. $\Sigma_1 \cup \Sigma_2 = \{a | a \in \Sigma_1 \text{ o } a \in \Sigma_2\}$ es un alfabeto.
2. $\Sigma_1 \cap \Sigma_2 = \{a | a \in \Sigma_1 \text{ y } a \in \Sigma_2\}$ es un alfabeto si es distinto del conjunto vacío.
3. $\Sigma_1 \setminus \Sigma_2 = \{a | a \in \Sigma_1 \text{ y } a \notin \Sigma_2\}$ es un alfabeto si es distinto del conjunto vacío.
4. $\Sigma_2 \setminus \Sigma_1 = \{a | a \in \Sigma_2 \text{ y } a \notin \Sigma_1\}$ es un alfabeto si es distinto del conjunto vacío.

Una **cadena o palabra sobre un alfabeto** es una sucesión finita de símbolos sobre ese alfabeto, usualmente escritos uno al lado de otros sin ningún tipo de separación. Si consideramos 01001 esta es una cadena para los alfabetos Σ_1 y Γ. Del mismo modo $abracadabra$ es una cadena del alfabeto Σ_2.

Si w es una cadena en el alfabeto, Σ la **longitud** de w que denotamos por $|w|$ es el número de símbolos que contiene. La cadena de longitud 0 es llamada la **cadena vacía** y se denota por ε. La cadena vacía hace el papel del cero en el sistema númerico.

Ejemplo 2. *Si* $w = 121$ *es una palabra sobre el alfabeto* $\Sigma = \{1, 2, 3\}$ *entonces* $|w| = 3$.

Ejemplo 3. $|\varepsilon| = 0$.

Si w tiene logitud n entonces podemos escribir $w = w_1 w_2 \cdots w_n$ donde cada $w_i \in \Sigma$ para $i = 1, 2, \ldots, n$.

El **inverso** de w, que denotaremos por $w^{\mathcal{R}}$, es la cadena obtenida escribiendo w en el orden inverso, esto es, $w^{\mathcal{R}} = w_n w_{n-1} \cdots w_1$.

Ejemplo 4. *Si* $w = abracadabra$ *entonces* $w^{\mathcal{R}} = arbadacarba$.

Ejemplo 5. *Si* $w = 1221$ *entonces* $w^{\mathcal{R}} = 1221 = w$, *cuando una cadena o palabra cumple que* $w^{\mathcal{R}} = w$ *se le llama* ***palíndromo***.

La cadena z es una **subcadena** si aparece incluida en w.

Ejemplo 6. *La cadena* cad *es una subcadena de* $abracadabra$.

Si tenemos una cadena $x = x_1 x_2 \cdots x_m$ de longitud m y una cadena $y = y_1 y_2 \cdots y_n$ de longitud n, la **concatenación** de x e y, que escribiremos como xy es la cadena obtenida añadiendo y al final de x, esto es,

$$xy = x_1 x_2 \cdots x_m y_1 y_2 \cdots y_n$$

Ejemplo 7. *Consideremos* $x = 0101$ *e* $y = 1111$ *entonces* $xy = 01011111$.

Observése que se cumple

$$|xy| = |x| + |y|,$$

Además, para cualquier cadena x escribiremos por definición

$$\boxed{x = \varepsilon x = x\varepsilon.}$$

En particular se tiene que $|x\varepsilon| = |\varepsilon x| = |x|$. Nótese que la palabra vacía ε con hace con respecto la concatenación el papel del uno con respecto la multiplicación de números naturales.

Para concatenar una cadena consigo misma emplearemos la siguiente notación: Para cualquier cadena x y cada $k \geq 1$ escribiremos

$$\boxed{x^k = \overbrace{xx \cdots x}^{k-\text{veces}}.}$$

Emplearemos, además, el siguiente convenio notacional: Para cualquier cadena finita x escribiremos

$$x^0 = \varepsilon.$$

El **orden lexicográfico** de las cadenas es el mismo que el orden que aparece en los diccionarios. Establecemos en primer lugar un oden sobre el alfabeto Σ y después lo extendemos a las cadenas formadas por los símbolos de dicho alfabeto, de forma que $x < y$, si existe un índice $k \leq \min\{m, n\}$ de forma que $x_1 \cdots x_k = y_1 \cdots y_k$ y $x_{k+1} < y_{k+1}$. Estamos asumiendo que si $k = m$ entonces $x_{k+1} = \varepsilon$.

Ejemplo 8. *Consideremos* $\Sigma = \{0, 1\}$, *donde asumimos que* $0 < 1$. *Entonces*

$$00010100111 < 00010101.$$

Esto se ve mejor si lo escribrimos como sigue:

$$0001010\varepsilon0\varepsilon111 < 0001010\varepsilon1\varepsilon.$$

Ejemplo 9. *Consideremos* $\Sigma = \{a, b, c\}$, *donde asumimos que* $a < b < c$. *Entonces*

$$baca < c,$$

y

$$caba < cabac,$$

ya que podemos escribir

$$caba\varepsilon < cabac.$$

Un **lenguaje** es un conjunto cualquiera de cadenas o palabras definidas sobre un alfabeto finito.

Los lenguajes pueden ser conjuntos muy grandes, como el formado por todas las palabras "correctas" en español o el lenguaje $\{1, 11, 111, 1111, \ldots\}$ formado por todas las cadenas finitas de unos. Hay que destacar que este lenguaje tiene tantos elementos como números naturales. Para demostrarlo basta comprobar que la aplicación definida como

$$n \mapsto 1^n$$

establece una relación biyectiva entre los números naturales $\mathbb{N}$ y las palabras de dicho lenguaje.

Cuando un lenguaje es muy grande es dificil especificar que cadenas o palabras lo componen. La especificación o caracterización de las mismas es una de las tareas principales a las que le vamos a dedicar gran parte de nuestro tiempo.

Dado que un lenguaje es un conjunto de cadenas, se puede tener el lenguaje compuesto por ninguna cadena, el **lenguaje vacío**, y que denotaremos por $\emptyset$. Este no es el mismo lenguaje que el que consta de la cadena vacía $\{\varepsilon\}$.

1.2. Operaciones con lenguajes

Los conceptos de concatenación, potencia e inverso se pueden extender a los lenguajes en general.

Sean L_1 y L_2 dos lenguajes sobre un alfabeto Σ. Se define el **lenguaje concatenación** de L_1 y L_2 como

$$L_1L_2 = L_1 \circ L_2 = \{wx | w \in L_1 \text{ y } x \in L_2\}.$$

En consecuencia, L_1L_2 está formado por todas las cadenas que se forman concatenando cada cadena de L_1 con todas las cadenas de L_2

Ejemplo 10. *Sean* $L_1 = \{00, 11, 1\}$ *y* $L_2 = \{10, 01\}$. *Entonces*

$$L_1L_2 = \{0010, 0001, 1110, 1101, 110, 101\}.$$

Cabe destacar que para formar el lenguaje concatenación no es necesario que L_1 y L_2 sean lenguajes sobre el mismo alfabeto. Si L_1 es un lenguaje sobre Σ_1 y L_2 es un lenguaje sobre Σ_2, entonces L_1L_2 es un lenguaje sobre $\Sigma_1 \cup \Sigma_2$.

Ejemplo 11. *Sean Sean* $L_1 = \{00, 11, 1\}$ *y* $L_2 = \{ab, ba\}$. *Entonces*

$$L_1L_2 = \{00ab, 00ba, 11ab, 11ba, 1ab, 1ba\},$$

es un lenguaje sobre

$$\{0, 1, a, b\} = \{0, 1\} \cup \{a, b\}.$$

Nótese que para cualquier lenguaje no vacío L se cumple:

$$L = L \circ \{\varepsilon\} = \{\varepsilon\} \circ L$$

Al igual que para las cadenas, podemos definir **la potencia** de un lenguaje L mediante la expresión

$$L^k = \overbrace{LL\cdots L}^{k-\text{veces}} = \overbrace{L \circ L \circ \cdots \circ L}^{k-\text{veces}}.$$

para $k \geq 1$. Si $k = 0$, escribiremos

$$L^0 = \{\varepsilon\}.$$

Ejemplo 12. *Si* $L = \{ab\}$, *entonces*

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^1 &= \{ab\} \\ L^2 &= L \circ L = \{abab\} \\ L^3 &= L \circ L^2 = \{ababab\} \end{aligned}$$

En importante tener en cuenta que con esta definición se tiene que

$$\boxed{\emptyset^0 = \{\varepsilon\}.}$$

Puesto que un lenguaje es una colección o subconjunto de cadenas, se puede definir para el mismo la unión, intersección y el concepto de sublenguaje, al igual que se definen para los conjuntos en general.

Si L_1 y L_2 dos lenguajes sobre un alfabeto Σ. Se define el lenguaje **unión** $L_1 \cup L_2$ como el lenguaje formado por las cadenas que pertenecen al menos a uno de los dos lenguajes. Por lo tanto

$$L_1 \cup L_2 = \{w | w \in L_1 \text{ o } w \in L_2\}.$$

La **intersección** de dos lenguajes L_1 y L_2 es el lenguaje

$$L_1 \cap L_2 = \{w | w \in L_1 \text{ y } w \in L_2\}.$$

Es decir está formado por las cadenas que son comunes a ambos lenguajes.

Ejemplo 13. *Consideremos* $L_1 = \{\varepsilon, 0, 1, 10, 11\}$ *y* $L_2 = \{\varepsilon, 1, 0110, 11010\}$. *Entonces*

$$L_1 \cup L_2 = \{\varepsilon, 0, 1, 10, 11, 0110, 11010\}$$

y

$$L_1 \cap L_2 = \{\varepsilon, 1\}.$$

Vamos a hora a introducir una notación que será muy útil para describir lenguajes como $L_1 = \{1, 11, 111, 1111, \ldots\}$ y $L_2 = \{\varepsilon, 1, 11, 111, 1111, \ldots\}$. Nótese que podemos escribir

$$L_1 = \{1\} \cup \{1\}^2 \cup \{1\}^3 \cup \cdots,$$

y

$$L_2 = \{1\}^0 \cup \{1\}^1 \cup \{1\}^2 \cup \{1\}^3 \cup \cdots.$$

En general dado cualquier lenguaje L sobre un alfabeto Σ definimos **la clausura o producto estrella de** L al lenguaje

$$\boxed{L^* = \cup_{n=0}^{\infty} L^n = \{\varepsilon\} \cup L \cup L^2 \cup L^2 \cup \cdots,}$$

y

$$\boxed{L^+ = \cup_{n=1}^{\infty} L^n = L \cup L^2 \cup L^2 \cup \cdots.}$$

Nótese que $L^* = \{\varepsilon\} \cup L^+$.

Ejemplo 14. *Podemos escibir entonces*

$$\{1\}^* = \{\varepsilon, 1, 11, 111, 1111, \ldots\}$$

y

$$\{1\}^+ = \{1, 11, 111, 1111, \ldots\}.$$

Ejemplo 15. *Consideremos* $\Sigma = \{0, 1\}$ *y* $L = \Sigma$. *Entonces* $L^* = \Sigma^* = \{0, 1\}^*$ *está formado por todas las cadenas finitas e infinitas de ceros y unos y la palabra vacía* ε.

Ejemplo 16. *Consideremos* $L = \{\varepsilon\}$. *Entonces,* $L^* = \{\varepsilon\}^* = \{\varepsilon\}$.

Si L es un lenguaje sobre Σ, definimos el lenguaje **complemento** de L sobre Σ como

$$\boxed{L^C = \Sigma^* \setminus L = \{w \in \Sigma^* | w \notin L\}.}$$

Ejemplo 17. *Si* $L = \{\varepsilon\}$, *entonces* $L^C = \Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

Del mismo modo podemos definir para dos lenguajes L_1 y L_2 sobre el alfabeto Σ el lenguaje

$$\boxed{L_1 \setminus L_2 = \{w \in L_1 | w \notin L_2\} = L_1 \cap L_2^c.}$$

Ejemplo 18. *Sean* $L_1 = \{a, ab, ba, aab\}$ *y* $L_2 = \{b, ab, bb, ca\}$. *Entonces* $L_1 \setminus L_2 = \{a, ba, aab\}$.

Tambien podemos extender dado un lenguaje L el lenguaje inverso $L^{\mathcal{R}}$ mediante la siguiente definición

$$\boxed{L^{\mathcal{R}} = \{w^{\mathcal{R}} | w \in L\}.}$$

Ejemplo 19. *Sea* $L = \{0, 1, 01, 001, 0001\}$. *Entonces* $L^{\mathcal{R}} = \{0, 1, 10, 001, 1000\}$.

Si L_1 y L_2 son dos lenguajes sobre el alfabeto Σ y si todas las cadenas de L_1 son también cadenas de L_2, diremos que L_1 es un **sublenguaje** de L_2 y lo denotaremos como

$$\boxed{L_1 \subset L_2.}$$

Ejemplo 20. *Consideremos* $L_1 = \{a, aa, aaa, aaaa\}$ *y* $L_2 = \{a\}^*$. *Entonces* $L_1 \subset L_2$.

Se dice que dos lenguajes L_1 y L_2 son iguales si contienen exactamente las mismas cadenas, es decir son conjuntos iguales y se denota como

$$\boxed{L_1 = L_2.}$$

Teorema 1. *Sean* L_1 *y* L_2 *dos lenguajes sobre un alfabeto* Σ. *Entonces* $L_1 = L_2$ *si y solo si* $L_1 \subset L_2$ *y* $L_2 \subset L_1$.

Demostración. Supongamos en primer lugar que $L_1 = L_2$. Para demostrar en primer lugar que se cumple que $L_1 \subset L_2$, elijamos una cadena genérica x de L_1, como $L_1 = L_2$ entonces x será a su vez una cadena de L_2, luego queda probado que $L_1 \subset L_2$. Veamos ahora que también se cumple que $L_2 \subset L_1$. Elijamos, como antes una cadena genérica y de L_2, del mismo modo que antes y será a su vez una cadena de L_1, lo que prueba que $L_2 \subset L_1$.

Demostremos ahora la implicación inversa. Supongamos pues que se cumple que $L_1 \subset L_2$ y $L_2 \subset L_1$. Esto nos dice que cualquier cadena de L_1 es a su vez una cadena de L_2 y que cualquier cadena de L_2 es a su vez una cadena de L_1. En consecuencia ambos lenguajes comparten todas sus cadenas y por lo tanto deben de ser iguales. □

El Teorema 1.2 nos proporciona una metodología para determinar si dos lenguajes son iguales.

Ejemplo 21. *Demostrar que dados los lenguajes L_1, L_2 y L_2 sobre un alfabeto Σ se cumple que*

$$L_1 \circ (L_2 \cup L_3) = (L_1 \circ L_2) \cup (L_1 \circ L_3).$$

Para demostrar la igualdad necesitamos demostrar en virtud del Teorema 1.2 que

1. $L_1 \circ (L_2 \cup L_3) \subset (L_1 \circ L_2) \cup (L_1 \circ L_3)$ *y*
2. $(L_1 \circ L_2) \cup (L_1 \circ L_3) \subset L_1 \circ (L_2 \cup L_3)$.

Veamos 1. Consideremos un elemento genérico x de $L_1 \circ (L_2 \cup L_3)$ que tiene que ser de la forma $x = wz$ donde $w \in L_1$ y $z = L_2 \cup L_3$. Luego o bien $z \in L_2$ o bien $z \in L_3$. En el primer caso $x \in L_1 \circ L_2$ o bien en el segundo $x \in L_1 \circ L_3$. Esto nos dice que $x \in (L_1 \circ L_2) \cup (L_1 \circ L_3)$. Esto prueba 1.

Para probar 2, consideremos un elemento genérico y de $(L_1 \circ L_2) \cup (L_1 \circ L_3)$. Entonces o bien $y \in L_1 \circ L_2$ o bien $y \in L_1 \circ L_3$. Podemos escribir entonces que que $y = wz$ donde o bien $w \in L_1$ y $z \in L_2$ o bien $w \in L_1$ y $z \in L_3$. En resumen, $y = wz$ siendo $w \in L_1$ y o bien $z \in L_2$ o bien $z \in L_3$. En consecuencia, y está en $L_1 \circ (L_1 \cup L_3)$ queda demostrada la segunda inclusión.

Ejercicios propuestos

1. Sea $\Sigma = \{1\}$. Se puede decir que para todo número natural n hay una palabra $w \in \Sigma^*$ para la cual $|w| = n$? ¿Podemos afirmar que dicha cadena es única? ¿Qué ocurriría si $\Sigma = \{1, 2\}$.

2. ¿ Para una cadena finita w podemos afirmar que

   $$|w^{i+j}| = |w^i| + |w^j|?$$

   Encontrar una expresión para $|w^{i+j}|$ en términos de i, j y $|w|$.

3. Demostrar para cualquier par de cadenas finitas x e y que $(xy)^{\mathcal{R}} = y^{\mathcal{R}} x^{\mathcal{R}}$.

4. Demostrar que para cualquier lenguaje L se cumple que

   $$L^+ = L \circ L^* = L^* \circ L.$$

5. Demostrar que para dos lenguajes L_1 y L_2 se cumple
$$(L_1 \circ L_2)^{\mathcal{R}} = L_2^{\mathcal{R}} \circ L_1^{\mathcal{R}}.$$

6. Sea L un lenguaje. ¿Qué tipo de lenguaje es $A \circ \emptyset$?

7. Sean $L_1 = \{el, mi\}$ y $L_2 = \{caballo, casa, coche\}$ sublenguajes sobre el alfabeto español. Obtener $L_1 \circ L_2$, $L_1 \circ L_1$ y $L_2 \circ L_2$.

8. Sea $L = \{\varepsilon, a\}$. Obtener L^n para $n = 0, 1, 2, 3$. ¿Cuántos elementos tiene A^n para un valor de n arbitrario? ¿Cuáles son las cadenas de L^n para un valor de n arbitrario?

9. Suponer que $L = \{\varepsilon\}$. Obtener L^n para un n arbitrario.

10. Sea $L_1 = \{\varepsilon, ab\}$ y $L_2 = \{cd\}$. ¿Cuántas cadenas hay en $A^n \circ B$ para un n arbitrario?

11. Sean $L_1 = \{a\}$ y $L_2 = \{b\}$. Obtener $A^n \circ B$, $A \circ B^n$ y $(A \circ B)^n$.

12. Sean $L_1 = \{\varepsilon\}$, $L_2 = \{aa, ab, bb\}$, $L_3 = \{\varepsilon, aa, ab\}$ y $L_4 = \emptyset$. Obtener $L_1 \cup L_2$, $L_1 \cup L_3$, $L_2 \cup L_3$, y $L_1 \cap L_2$, $L_1 \cap L_3$, $L_2 \cap L_3$. Suponer que L un lenguaje cualquiera $L \cup L_4$ y $L \cap L_4$.

13. Si L y L_i para $i = 1, 2, 3, \ldots$ son lenguajes sobre un alfabeto Σ, demostrar que
$$L \circ \left(\bigcup_{i=1}^{\infty} L_i \right) = \bigcup_{i=1}^{\infty} (L \circ L_i).$$

14. ¿Bajo qué condiciones podemos afirmar que $L^+ = L^*$?

15. Conocemos que para cualquier lenguaje L se tiene que $\varepsilon \in L^*$. ¿Cuándo podemos afirmar que $\varepsilon \in A^+$.

16. Demostrar que $\{\varepsilon\}^* = \{\varepsilon\} = \{\varepsilon\}^+$.

17. Sean L_1 y L_2 dos lenguajes sobre un alfabeto Σ. Demostrar que $(L_1 \cap L_2)^c = L_1^c \cup L_2^c$ y $(L_1 \cup L_2)^c = L_1^c \cap L_2^c$.

18. Sea $\Sigma = \{a, b, c\}$ y sea $L = \{c^i x c^j | i, j \geq 0\}$, donde $x \in \{\varepsilon, aw, wb\}$ para algún $w \in \Sigma$. ¿Se cumple que $L = \Sigma^*$? ¿Es cierto que $L^2 = \Sigma^*$?

19. Sea $\Sigma = \{a, b, c\}$. Lo siguiente es una definición recursiva de un lenguaje L :

    i $\varepsilon \in L$.

    ii Si $x \in L$, entonces axb y bxa están en L.

    iii Si x e y están en L, entonces xy está en L.

iv No hay nada más en L.

Se pide:

a) Demostrar que

$$L = \{w \in \Sigma^* | w \text{tiene el mismo numero de a's que de b's}\}.$$

b) Si b y ε están en L ¿qué más palabras hay en L?

c) Dar una definición recursiva para que $L \subset \{a, b\}^*$ contenga todas las cadenas que tienen el doble de a's que de b's.

20. Dar una definición recursiva de un palíndromo. (Obsérvese que ε es un palíndromo).

21. Sea $\Sigma = \{a\}$ y $x = a$. Definir las siguientes palabras : $x, xx, xxx, x^3, x^8, x^0$ Indicar sus longitudes.

22. Sea $\Sigma = \{0, 1, 2\}$, $x = 00$, $y = 1$, $z = 210$. Definir las siguientes palabras : $xy, xz, yz, xyz, (xy)^{\mathcal{R}}, x^3, x^2y^2, (xy)^2, (zxx)^3$. Indicar sus longitudes.

23. Describir las palabras pertenecientes a $L = \{0^n 1^n | n \geq 1\}$.

24. Describir las palabras pertenecientes a $L = \{0^i 1^j | 0 \leq i \leq j\}$.

25. Describir formalmente el lenguaje formado por 0's y 1's, en el que hay el doble de 0's que de 1's y todos los 0's van delante de los 1's.

26. Describir formalmente el lenguaje formado por palabras que comienzan y terminan en a teniendo entre medias 3 o más b's.

27. Dado el lenguaje $L = \{a, ab, ba\}$, enumere las palabras que pertenecen a los lenguajes $L^0, L^1 y L^2$.

28. Dados el alfabeto $\Sigma = \{1, 2, 3, a, b, c\}$, y los lenguajes $L_1 = \{1, 2, 3\}$ y $L2 = \{a, b, c\}$, definir los lenguajes $L_1 \cup L_2$, $L_1 L_2$ y $(L_1 L_2)^2$.

29. Sea $L = \{ab, aa, baa\}$. Indicar cuáles de las siguientes palabras pertenecen a L^+ : $abaa, aabab, abaabaaabaa, aaaabaaaa, baaaaabaaaab, baaaaabaa, \varepsilon$.

30. Sean $L_1 = \{a^n b^{n+1} | n \geq 1\}$ y $L_2 = \{w | \text{num. a's} = \text{num. b's}\}$. ¿ Es $L_1 = L_1^*$?. ¿ Y $L2 = L_2^*$?

31. ¿Existe algún lenguaje tal que $(L^*)^c = (L^c)^*$?

32. Demostrar o refutar la igualdad siguiente :

$$(L^*)^{\mathcal{R}} = (L^{\mathcal{R}})^*,$$

para todo lenguaje L.

33. Demostrar que para todo lenguaje L, se verifica $L^*L^* = L^*$.

34. Dado el lenguaje $L = \{a, abb, ba, bbba, b\}$, especifique el conjunto de todas las palabras con una longitud menor que 3 en L^*.

Parte II

Autómatas y Lenguajes

Capítulo 2

Lenguajes Regulares

En esta primer capítulo introduciremos las máquinas teóricas capaces de procesar lenguajes formales. En primer lugar definiremos los Autómatas Finitos Deterministas como máquinas que procesan los llamados Lenguajes Regulares. Demostraremos que la clase de lenguajes regulares es cerrada para la operación de unión de lenguajes. Sin embargo, debido a las particulares características de estos mecanismos nos es imposible demostrar que dicha clase es cerrada para las operaciones de concatenación y clausura o producto estrella de lenguajes.

Para superar este obstáculo definiremos los llamados Autómatas Finitos No Deterministas. Este tipo de máquinas más versátiles tienen la curiosa propiedad que procesan el mismo tipo de lenguajes que los Autómatas Finitos Deterministas, los llamados lenguajes regulares. La versatilidad de los mismos nos permitirá demostrar que el lenguaje que procesan es cerrado para las operaciones de concatenación y clausura o producto estrella de lenguajes.

En una tercera fase, introduciremos unas expresiones de carácter algebraico llamadas expresiones regulares. Demostraremos en primer lugar que si un lenguaje es caracterizado por una de estas expresiones entonces podemos construir un Autómata Finito No Determinista que procesa dicho lenguaje, y en consecuencia es un lenguaje regular. Reciprocamente demostraremos que si un lenguaje es regular, entonces podemos construir lo que se conoce como Autómata Finito No Determinista Generalizado que procesa dicho lenguaje. Finalmente, construiremos a partir de este una expresión regular que caracteriza dicho lenguaje.

En resumen, Autómatas Finitos Deterministas, Autómatas Finitos No Deterministas y Expresiones Regulares caracterizan la misma clase de lenguajes, los Lenguajes Regulares.

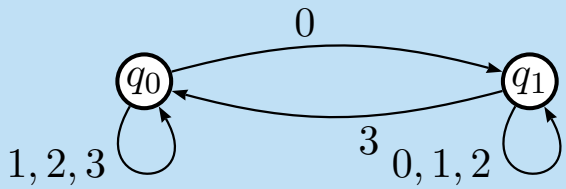
Para concluir, empleando el llamado Lema del Bombeo, construiremos un mecanismo para determinar si un lenguaje dado no es regular.

2.1. Autómatas Finitos

Los autómatas finitos son unos excelentes modelos para representar computadoras con una cantidad extremadamente limitada de memoria. ¿Qué podemos entender por un computador de memoria limitada?

Ejemplo: Una puerta automática

El controlador de la puerta se encuentra siempre en uno de los dos estados "ABIERTA" que denotaremos por q_0 o "CERRADA" que será q_1. Del mismo modo existen cuatro posibles efectos que hacen que el sistema cámbie de estado: La alfombrilla de entrada sufre un "input" que se denotaremos por 0, en el caso de que sea la alfombrilla de salida por 1, si no existe ninguna señal por 2 y si tenemos una señal en ambas alfombrillas la señal la denotaremos por 3. Graficamente el comportamiento de la puerta automática lo podemos representar empleando el grafo siguiente:



O bien se puede representar empleando la tabla de trasición siguiente:

δ	0	1	2	3
q_0	q_1	q_0	q_0	q_0
q_1	q_1	q_1	q_0	q_1

Los autómatas finitos son mecanismos "sin memoria" en consecuencia tienen propiedades muy similares a las llamadas **Cadenas de Markov**, reconocidos objetos para el reconocimiento de patrones.

La figura siguiente representa lo que a partir de ahora denominaremos autómata finito y que denotaremos por M_1 :

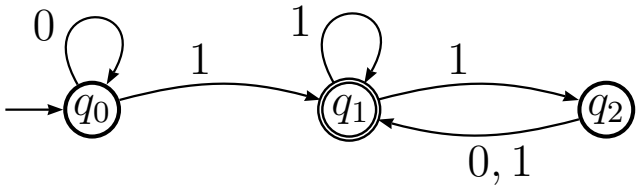


Figura 2.1: Un autómata finito llamado M_1 con tres estados diferentes.

La Figura 2.1 representa el llamado **diagrama de estado** de M_1. Tiene tres estados etiquetados por q_0, q_1 y q_2. El estado inicial q_1 que se mediante una flecha apuntando hacia el y no etiquetada con ningún símbolo. El **estado de aceptación**, q_1 que es el etiquedado con una doble circunferencia. Las flechas etiquetadas entre estados se les llama **transiciones**.

Cuando este autómata recibe como "input" una cadena, como por ejemplo, 1101 el autómata procesa la cadena y produce una respuesta o "output". La respuesta es **aceptar** o **rechazar**. El proceso comienza en el estado inicial de M_1 que es q_0. el autómata recibe el primer símbolo de la cadena de entrada, en este caso 1 de 1101, una vez leido este símbolo M_1 se mueve a lo largo de un estado u otro según le indique el símbolo etiquetado en la transición correspondiente. En el ejemplo que nos ocupa, se moverá al estado q_1 ya que $(q_0, 1) \mapsto q_1$. Empleando recursivamente esta regla seguimos hasta finalizar los símbolos de la cadena de entrada. Si el último estado, es el estado de aceptación, en nuestro ejemplo q_1, entonces aceptamos la cadena y en caso contrario esta es rechazada.

Para nuestro ejemplo podemos escribir:

1. Comenzamos en el estado q_0.
2. Leemos 1 nos desplazamos de q_0 a q_1.
3. Leemos 1 nos desplazamos de q_1 a q_1.
4. Leemos 0 nos desplazamos de q_1 a q_2.
5. Leemos 0 nos desplazamos de q_2 a q_1.
6. Aceptamos 1100 ya que M_1 se encuentra en el estado de aceptación q_1 al final de la cadena de entrada.

Ejercicio 1. *Comprobar que las cadenas*

1. $1, 01, 11, 010101010101$ *son aceptadas por* M_1.
2. $100, 0100, 110000, 0101000000$ *son aceptadas por* M_1
3. $0, 10, 101000$ *son rechazadas por* M_1.
4. *¿Podemos afirmar que* M_1 *acepta cualquier cadena que finaliza con un número par de* 0*'s que siguen al último* 1*?*

2.1.1. Definición formal de un autómata finito

Vamos a emplear la llamada **función de transición** que denotaremos por δ para definir las reglas por las que el autómata se mueve de un estado a otro en función del símbolo recibido. Como podemos ver en la Figura 2.2 si nos encontramos en el estado x y recibimos el sí).mbolo a, entonces el autómata se desplaza al estado y. Esto lo representamos mediante la función δ como $\delta(x, a) = y$ (ver la Figura 2.2).

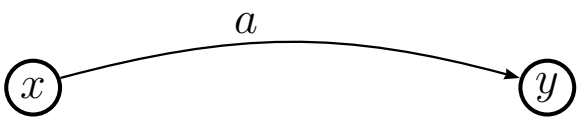


Figura 2.2: La función de transición $\delta(x, a) = y$

Definición 1. *Un **autómata finito** es una* 5*–upla* $M = (Q, \Sigma, \delta, q_0, F)$ *donde:*

1. Q *un un conjunto finito llamados los* ***estados****.*
2. Σ *es un conjunto finito llamado el* ***alfabeto****.*
3. $\delta : Q \times \Sigma \longrightarrow Q$ *es la* ***función de transición****.*
4. $q_0 \in Q$ *es el llamado* ***estado inicial****.*
5. $F \subset Q$ *es el conjunto de* ***estados de aceptación****.*

Ahora podemos describir M_1 (ver la Figura 2.1) de manera mucho más fórmal.

1. $Q = \{q_0, q_1, q_2\}$.
2. $\Sigma = \{0, 1\}$.
3. δ viene determinada por la tabla siguiente:

δ	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_1	q_1

4. q_0 es el estado inicial.
5. $F = \{q_1\}$ es el conjunto de estados de aceptación.

Si A es el conjunto de cadenas que el autómata M acepta, diremos que A es el **lenguaje aceptado o reconocido** por el autómata M, y escribiremos $L(M) = A$.

Un autómata puede reconocer o aceptar diferentes cadenas, pero siempre reconoce o acepta un único lenguaje. Si el autómata no reconoce o acepta ninguna cadena diremos que M reconoce el **lenguaje vacío** $\emptyset$.

En el caso del autómata M_1 si definimos

$$A = \{w | w \text{ contiene al menos un 1 y finaliza con un numero par de 0's que siguen al ultimo 1}\},$$

tenemos que $L(M_1) = A$.

2.1.2. Definición formal de computación

Sea $M = (Q, \Sigma, \delta, q_0, F)$ un autómata finito y sea $w = w_1 w_2 \cdots w_n$ una cadena en el alfabeto Σ. Entonces M **acepta** w si existe una sucesión de estados $r_0, r_1, \ldots, r_n$ en Q que cumplen las siguientes tres condiciones:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ para $i = 0, 1, \ldots, n-1$, y
3. $r_n \in F$.

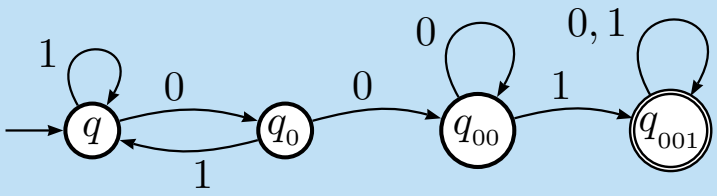
Diremos que M **reconoce el lenguaje** A si $A = \{w | M \text{ acepta } w\}$. Escribiremos entonces que $L(M) = A$.

Definición 2. *A un lenguaje se le llama **lenguaje regular** si es aceptado o reconocido por algún autómata finito.*

Esto nos dice que para cualquier lenguaje regular A se puede construir un autómata finito M de forma que $A = L(M)$.

Ejemplo: El diseño de un autómata finito

Consideremos el lenguaje formado por todas las cadenas del alfabeto $\Sigma = \{0, 1\}$ que contienen la subcadena 001. Para determinar que es un lenguaje regular debemos construir un autómata finito que acepte o reconozca dicho lenguaje. En principio mientras vayan apareciendo unos, no deberíamos cambiar de estado q. La situación cámbia en el momento que aparece el primer cero, en este momento cambíamos a q_0 si seguidamente aparece un uno, volvemos a encontranos en la situación inicial, volvemos a q y en caso contrario nos desplazamos a un nuevo estado q_{00}. Si aparece un uno, podemos finalizar en el estado q_{001}, en caso contrario permanecemos en q_{00}. En resumen obtenemos el automáta



2.1.3. Las operaciones regulares

En las dos secciones precedentes se han introducido y definido los autómatas finitos y los lenguajes regulares. Vamos a comenzar a investigar sus propiedades. Para ello

vamos a necesitar la ayuda una pléyade de tecnicas que emplearemos para diseñar autómatas que puedan reconocer determinados tipos de lenguajes. Estas herramientas incluirán técnicas que nos permitirán demostrar que un determinado lenguaje no es regular (es decir, que están por encima de la capacidad de cómputo de un autómata finito).

Estas herramientas van a incluir tres operaciones que ya conocemos, la unión, la concatenación y la clausura o producto estrella, de lenguajes, que denominaremos a partir de ahora **operaciones regulares**, y que serán empleadas posteriormente para el estudio de los lenguajes regulares.

El concepto de "cerrado bajo la operación"

Si consideramos los números naturales $\mathbb{N} = \{1, 2, \ldots\}$. Decimos que $\mathbb{N}$ es cerrado bajo la operación de multiplicación, ya que el resultado de multiplicar dos números naturales es otro número natural. Por el contrario, dicho conjunto no es cerrado para la división, ya que si, por ejemplo, dividimos 1 entre 3 el resultado es un número racional, que evidentemente, no es un número natural.

Nuestro objetivo consistirá a partir de ahora en demostrar que los lenguajes regulares son cerrados para las operaciones regulares. Para ello en primer lugar demostraremos:

Teorema 2. *La clase de lenguajes regulares es cerrada para la unión. En otras palabras, si L_1 y L_2 son lenguajes reguales, entonces $L_1 \cup L_2$ en un lenguaje regular.*

IDEA DE LA DEMOSTRACIÓN. Partimos de dos lenguajes regulares L_1 y L_2 y tenemos que demostrar que la unión de ambos $L_1 \cup L_2$ es un lenguaje regular. Como L_1 y L_2 son regulares conocemos que existen dos AFD M_1 y M_2 de forma que M_1 reconoce el lenguaje L_1 y M_2 reconoce el lenguaje L_2. Para demostrar que $L_1 \cup L_2$ es un lenguaje regular tenemos que construir un AFD, que llamaremos M de forma que pueda reconocer este lenguaje.

La demostración del mismo es de caracter constructivo, ya que M será construido a partir de M_1 y M_2 del siguiente modo. Dada una cadena cualquiera de $L_1 \cup L_2$ esta será procesada en paralelo por M_1 y M_2 diremos M la acepta si es aceptada o bien por M_1 o bien por M_2 o bien por ambos. Esto nos asegura que M reconoce todas las cadenas del lenguaje unión. □

Demostración. Sea $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ que reconoce a L_1 y sea $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ que reconoce a L_2. Vamos a construir $M = (Q, \Sigma, \delta, q_0, F)$ que reconoce a $L_1 \cup L_2$.

Sean:

1. $Q = Q_1 \times Q_2 = \{(r_1, r_2) : r_1 \in Q_1, r_2 \in Q_2\}$, que es el producto cartesiano de los conjuntos de estados de los autómatas.

2. Σ es el mismo alfabeto que el de M_1 y M_2. En este resultado estamos asumiendo que ambos lenguajes tienen el mismo alfabeto. Esto en general no tiene por que ser cierto y en general tendríamos que suponer que M_1 tiene un alfabeto Σ_1 y M_2 un alfabeto Σ_2. En este caso tomaríamos como $\Sigma = \Sigma_1 \cup \Sigma_2$.

3. La función de transición δ se define como

$$\delta\left((r_1, r_2), a\right) = (\delta_1(r_1, a), \delta_2(r_2, a)).$$

4. $q_0 = (q_1, q_2)$ y

5. $F = \{(r_1, r_2) : r_1 \in F_1 \text{ o } r_2 \in F_2\}$

□

Necesitamos demostrar ahora que la clase de lenguaje regulares es cerrada para la concatenación y la clausura o producto estrella. Para la concatenación y como en el resultado anterior comenzaríamos con dos AFD M_1 y M_2 de forma que $L_1 = L(M_1)$ y $L_2 = L(M_2)$. Pero no podemos construir tan fácilmente un autómata M de forma que $L(M) = L(M_1 \circ M_2)$ ya que dada una cadena, no es tan inmediato el construir un mecanismo que detecte para una cadena que parte está en M_1 y que parte en M_2. Para ello, vamos a introducir, en la sección siguiente, una nueva clase de autómatas finitos, equivalentes a los AFD ya que generaran la misma clase de lenguajes, más versátiles y que nos permitirán demostrar de forma sencilla que la clase de los lenguajes regulares es cerrada para las operaciones regulares.

2.2. Autómatas no deterministas

En un autómata determinista cuando se encuentra en un estado determinado y lee el śimbolo siguiente conocemos cuál va a ser el estado al que nos vamos a mover. En un autómata no determinista, podemos tener diferentes estados a donde movermos (ver la Figura 2.3). Esto, nos proporciona un mecanismo mucho más general, que en particular engloba al modelo determinista. Podemos pues afirmar que todo autómata determinista es no determinista.

A partir de ahora consideraremos Autómatas Finitos Deterministas, AFD para simplificar y Autómatas Finitos No Deterministas, AFN.

En primer lugar cualquier estado de un AFN tiene exactamente una transición para cada símbolo del alfabeto. El AFN como el de la Figura 2.3 viola esta propiedad, el estado q_0 tiene una única transición para el símbolo 0 y dos transiciones para el símbolo 1. Esto se denotaría como $\delta(q_0, 1) = \{q_0, q_1\}$.

En segundo lugar el AFD etiqueta sus transciones con símbolos del alfabeto. Como podemos ver en la Figura 2.3 existe una única transición entre q_1 y q_2 etiquetada con la palabra vacía ε.

¿Cómo computa entonces un AFN?

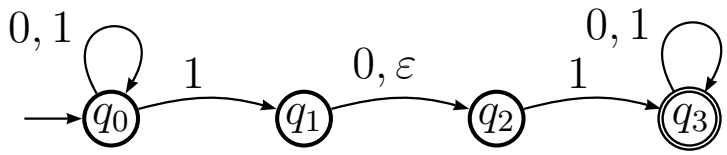


Figura 2.3: Un ejemplo de AFN.

Supongamos que un AFN comienza a procesar una cadena y llega a un estado en el que tenemos diferentes formas de movernos. Por ejemplo, supongamos que nos encontramos en el estado q_0 de la Figura 2.3, que denotaremos a partir de ahora por N_1, y el símbolo que nos llega para procesar es un 1. Después de leer el símbolo el autómata procesa diferentes copias de si mismo y sigue todos los posibles caminos en paralelo. En la Figura 2.4 podemos comprobar como procesaría N_1 la cadena 010110.

Consideremos un AFN definido sobre el alfabeto $\{0\}$. Un alfabeto conteniendo un único símbolo se le llama alfabeto **unitario**. Si nos fijamos en el AFN la Figura 2.5 este acepta todas las cadenas de la forma 0^k donde k es un múltiplo de 2 o 3. Como podemos observar la presencia de flechas etiquetadas con ε es esencial.

2.2.1. Definición formal del un Autómata Finito no Determinista

Para cualquier conjunto Q denotaremos por $\mathcal{P}(Q)$ la colección de todos los subconjuntos posibles de Q, incluido el conjunto vacío $\emptyset$. Al conjunto $\mathcal{P}(Q)$ se le conoce como **conjunto potencia** de Q. Para cualquier alfabeto Σ denotaremos por Σ_ε al conjunto $\Sigma \cup \{\varepsilon\}$. Con todo ello podemos escribir la descripción formal de la función de transición para un AFN como sigue.

Definición 3. *Un autómata finito no determinista es una* 5*–upla* $N = (Q, \Sigma, \delta, q_0, F)$ *donde:*

1. Q *un un conjunto finito llamados los* ***estados****.*
2. Σ *es un conjunto finito llamado el* ***alfabeto****.*
3. $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ *es la* ***función de transición****.*
4. $q_0 \in Q$ *es el llamado* ***estado inicial****.*
5. $F \subset Q$ *es el conjunto de* ***estados de aceptación****.*

Para el AFN de la Figura 2.3 se tiene

1. $Q = \{q_0, q_1, q_2, q_3\}$.
2. $\Sigma = \{0, 1\}$.

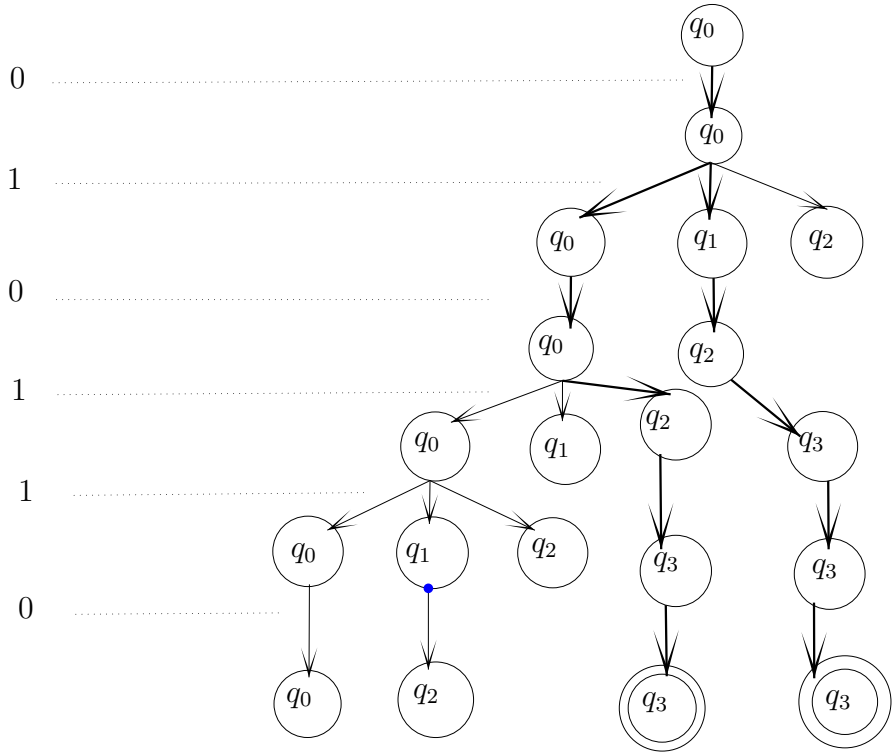


Figura 2.4: El cómputo de la cadena 010110 por N_1

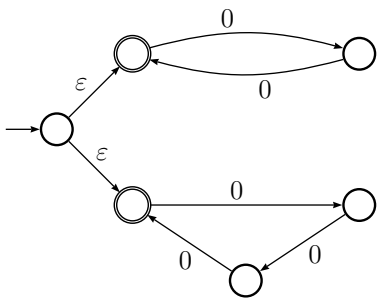


Figura 2.5: Un AFN sobre un alfabeto unitario.

3. δ viene determinada por la tabla siguiente:

δ	0	1	ε
q_0	$\{q_0\}$	$\{q_0, q_1\}$	$\emptyset$
q_1	$\{q_2\}$	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\{q_3\}$	$\emptyset$
q_3	$\{q_3\}$	$\{q_3\}$	$\emptyset$

4. q_0 es el estado inicial.

5. $F = \{q_3\}$ es el conjunto de estados de aceptación.

La definición formal de computación para un AFN es similar a la de un AFD. Sea $N = (Q, \Sigma, \delta, q_0, F)$ un AFN y sea w una cadena sobre el alfabeto Σ. Entonces diremos N **acepta** w si podemos escribir $w = y_1 y_2 \cdots y_m$, donde cada y_i es un elemento del alfabeto Σ_ε, y existe una sucesión finita de estados $r_0, r_1, \ldots, r_m$ en Q para los que se cumplen las tres siguientes condiciones:

1. $r_0 = q_0$,

2. $r_{i+1} \in \delta(r_i, y_{i+1})$ para $i = 0, 1, \ldots, m-1$, y

3. $r_m \in F$.

2.2.2. Equivalencia entre AFNs y AFDs

Los autómatas finitos deterministas y no deterministas reconocen la misma clase de lenguajes. Esto puede parecer sorprendente, ya que a priori parece que los AFN sean más potentes que los AFD. Esta característica es de mucha utilidad ya que en muchas ocasiones es más fácil describir un lenguaje empleando un AFN que su contrapartida determinista.

Diremos que dos autómatas son **equivalentes** si ambos reconocen o aceptan el mismo lenguaje.

Teorema 3. *Cualquier AFN tiene un AFD equivalente.*

IDEA DE LA DEMOSTRACIÓN. Si un lenguaje es reconocido por un AFN, entonces tenemos que demostrar la existencia de un AFD que lo reconoce a su vez. La idea es convertir un AFN en un AFD equivalente que lo simula. ¿Cómo podemos entoncer simular un AFN si pretendemos que se comporte como un AFD? El problema radica en que de cada cada estado pueden salir diferentes flechas con el mismo símbolo del alfabeto, esto es lo que provoca que la imagen de la función de transición sea un conjunto de estados. En consecuencia, tenemos que transformar los conjuntos de estados del AFN en los estados del AFD que tiene que simular. Si por ejemplo el AFN tiene k–estados sabemos que el conjunto potencia de todos los estados del AFN tiene

2^k elementos que constituyen todos los subconjuntos posibles que podemos construir a partir de los estados del AFN original. A partir de esto tendremos que construir una nueva función de transición que nos permita simular el AFD a partir del AFN. □

Demostración. Sea $N = (Q, \Sigma, \delta, q_0, F)$ el AFN que reconoce un lenguaje regular dado L. Construyamos el AFD M que reconoce a L. Consideraremos en primer lugar el caso en el que N no tiene flechas etiquetadas con ε.

Construimos $M = (Q', \Sigma, \delta', q_0', F')$ como sigue:

1. $Q' = 2^Q = \mathcal{P}(Q)$. Cualquier estado de M está formado por un subconjunto de estados de N.

2. Para $R \in Q'$, esto es $R \subseteq Q$, y $a \in \Sigma$ sea

$$\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \text{ para } r \in R\},$$

   otra forma de escribir esta expresión es

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a).$$

3. $q_0' = \{q_0\}$.

4. $F' = \{R \in Q' | R \cap F \neq \emptyset\}$

Veamos ahora el caso en el que existan flechas etiquetadas con ε, para ello necesitamos notación adicional. Para cada estado R de M vamos a definir $\varepsilon(R)$ como la colección de estados que se alcanzan a partir de R a través de transiciones etiquetadas con ε, incluyendo los elementos de R. De manera formal, para $R \subseteq Q$ definimos

$$\varepsilon(R) = \{q \in Q | q \text{ se alcanza a partir de } R \text{ con 0 o mas flechas } \varepsilon\}.$$

Entonces modificamos la función de transición de M añadiendo flechas adicionales sobre todos los estados que son alcanzados a través de transiciones ε a través de cada paso. Reemplazando $\delta(r, a)$ por $\varepsilon(\delta(r, a))$ hace que esto sea efectivo. Por lo tanto

$$\delta'(R, a) = \{q \in Q | q \in \varepsilon(\delta(r, a)) \text{ para } r \in R\}.$$

De forma adicional necesitaremos modificar el estado inicial de M de q_0' a $\varepsilon(q_0')$. Con esto completamos la construcción del AFD M que simula al AFN N. □

Corolario 1. *Un lenguaje es regular si y solo si algún AFN lo reconoce.*

Ejemplo: De un AFN a un AFD

Consideremos un AFN $N = (Q, \{a, b\}, \delta, 1, \{1\})$, siendo $Q = \{1, 2, 3\}$, dado por la Figura 2.6. Para construir el AFD equivalente, conocemos que N tiene 3 estados, en consecuencia

$$\mathcal{P}(Q) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

que es el número de estados del AFD equivalente que denotaremos por D.

Ahora determinaremos el estado inicial y los estados de aceptación. El estado inicial el $\varepsilon(\{1\})$ compuesto por los estados accesibles a través de $\{1\}$ empleando transiciones ε. A partir de la Figura 2.6 obtenemos que

$$\varepsilon(\{1\}) = \{1, 3\}.$$

Además, los estados de aceptación están formados por todos aquellos que contienen el estado de aceptación $\{1\}$, luego

$$F' = \{\emptyset, \{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}.$$

Determinemos ahora la función de transición. Conocemos que el estado $\{2\}$ va al estado $\{2, 3\}$ cuando se recibe como input el carácter a y no podemos ir más allá de 2 y 3 empleando flechas etiquetadas con ε.

El estado $\{1\}$ va a $\emptyset$ con a ya que ninguna flecha sale de el. Va hacia $\{2\}$ con el caracter b.

El estado $\{3\}$ va hacia $\{1, 3\}$ con a, ya que el estado 3 va a 1 y el 1 va a 3 con una flecha con etiqueta ε. El estado $\{3\}$ con b va a $\emptyset$.

El estado $\{1, 2\}$ va a $\{1, 3\}$ con a, ya que 3 con a va a 1 y 1 va a 3 con una flecha con etiqueta ε. El estado $\{1, 2\}$ con b va a $\{2, 3\}$

Continuando obtenemos el diagrama de la Figura 2.9.

Finalmente podemos simplificar la Figura 2.9 observando que no existen flechas apuntando al los estados $\{1\}$ y $\{1, 2\}$, en consecuencia podemos eliminarlos del diagrama sin que afecten al funcionamiento del autómata. Esto viene representado en la Figura 2.8.

2.2.3. La propiedad de clausura bajo operaciones regulares

Teorema 4. *La clase de lenguajes regulares es cerrada bajo la operación de la unión.*

IDEA DE LA DEMOSTRACIÓN. Supongamos que tenemos dos lenguajes regulares L_1 y L_2 y deseamos demostrar que $L_1 \cup L_2$ es regular. La idea es tomar dos AFN N_1 y N_2 que acepte cada uno de los lenguajes y combinar ambos para construir un AFN N que combine a ambos.

La máquina N tiene que aceptar cadenas de ambos lenguajes. Para ello introduciremos un nuevo estado inicial del que partirán dos flechas una al estado inicial de N_1 y otra al correspondiente de N_2 con la etiqueta ε. □

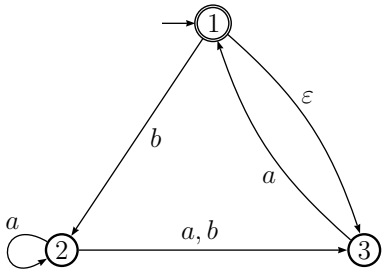


Figura 2.6: El AFN N.

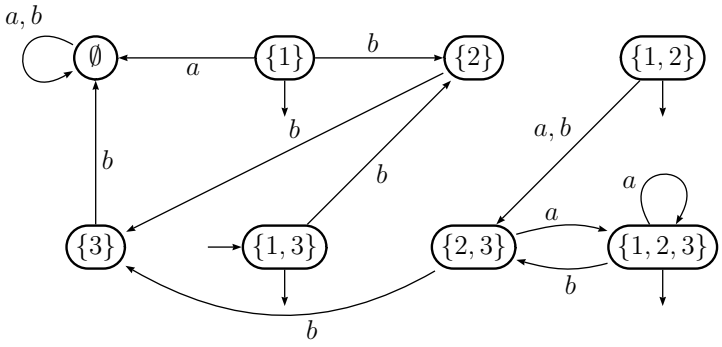


Figura 2.7: El AFD D en este caso marcamos los estados finales con una flecha vertical sin etiquetar.

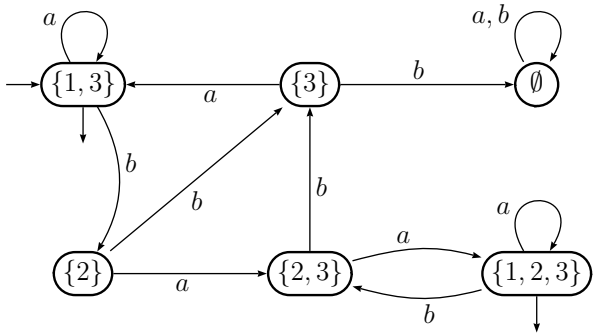


Figura 2.8: El AFD D eliminados los estados que son innecesarios.

Demostración. Sea $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ que reconoce a L_1 y $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ que reconoce a L_2. El siguiente autómata $N = (Q, \Sigma, \delta, q_0, F)$ reconoce a $L_1 \cup L_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$, donde q_0 es un nuevo estado que introducimos.
2. q_0 es el estado inicial de N.
3. $F = F_1 \cup F_2$.
4. Definimos δ para cada $q \in Q$ y $a \in \Sigma$ del modo siguiente

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{si} \quad q \in Q_1 \\ \delta_2(q, a) & \text{si} \quad q \in Q_2 \\ \{q_1, q_2\} & \text{si} \quad q = q_0 \text{ y } a = \varepsilon \\ \emptyset & \text{si} \quad q = q_0 \text{ y } a \neq \varepsilon \end{cases}$$

□

Teorema 5. *La clase de lenguajes regulares es cerrada bajo la operación de la concatenación.*

IDEA DE LA DEMOSTRACIÓN. Supongamos que tenemos ahora dos lenguajes regulares L_1 y L_2 y deseamos probar que $L_1 \circ L_2$ es regular. La idea es tomar dos AFN N_1 y N_2 para cada uno de estos lenguajes y combinarlos para obtener un nuevo AFN N que reconozca a $L_1 \circ L_2$.

Para ello vamos a considerar que primero procesamos a traves de N_1 y ligamos todos los estados finales de este autómata al estado inicial de N_2 mediante flechas etiquetadas con ε. El autómata resultante será N que reconocerá las cadenas de $L_1 \circ L_2$. □

Demostración. Sea $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ que reconoce a L_1 y $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ que reconoce a L_2. El siguiente autómata $N = (Q, \Sigma, \delta, q_0, F)$ reconoce a $L_1 \circ L_2$.

1. $Q = Q_1 \cup Q_2$.
2. $q_0 = q_1$.
3. $F = F_2$.
4. Definimos δ para cada $q \in Q$ y $a \in \Sigma$ del modo siguiente

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{si} \quad q \in Q_1 \text{ y } q \notin F_1 \\ \delta_1(q,a) & \text{si} \quad q \in F_1 \text{ y } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & \text{si} \quad q \in F_1 \text{ y } a = \varepsilon \\ \delta_2(q,a) & \text{si} \quad q \in Q_2. \end{cases}$$

□

Teorema 6. *La clase de lenguajes regulares es cerrada bajo la operación de la clausura o producto estrella.*

IDEA DE LA DEMOSTRACIÓN. Vamos a considerar que tenemos un lenguaje regular L_1 y tenemos que demostrar que su clausura de Kleene o producto estrella L^* es un lenguaje regular. Para ello vamos a considerar un AFN N_1 que reconoce al lenguaje y vamos a modificarlo convenientemente, obteniendo una nueva máquina N para que reconozca a L^*.

Hay que tener en cuenta que N tiene que reconocer a la palabra vacía ε. En consecuencia el estado inicial tiene que ser un estado de aceptación. Esto nos va a lleva a añadir un nuevo estado, que será el estado inicial de N del que partirá una flecha etiquetada con ε al estado inicial de N_1. Para aceptar cadenas de L^* de cada estado final de N_1 partirá una flecha etiquetada con la palabra vacía hacia el estado inicial de N_1.
□

Demostración. Sea $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ que reconoce a L_1. El siguiente autómata $N = (Q, \Sigma, \delta, q_0, F)$ reconoce a L_1^*.

1. $Q = \{q_0\} \cup Q_1$.
2. q_0 es el nuevo estado inicial.
3. $F = \{q_0\} \cup F_1$.
4. Definimos δ para cada $q \in Q$ y $a \in \Sigma$ del modo siguiente

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{si} \quad q \in Q_1 \text{ y } q \notin F_1 \\ \delta_1(q,a) & \text{si} \quad q \in F_1 \text{ y } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & \text{si} \quad q \in F_1 \text{ y } a = \varepsilon \\ \{q_1\} & \text{si} \quad q = q_0 \text{ y } a = \varepsilon \\ \emptyset & \text{si} \quad q = q_0 \text{ y } a \neq \varepsilon \end{cases}$$

□

2.3. Expresiones Regulares

En aritmética empleamos las operaciones $+$ y $\times$ para construir expresiones del tipo

$$(5 + 3) \times 4.$$

Del mismo modo, podemos emplear operaciones regulares para construir expresiones que describan lenguajes, a este tipo de expresiones las denominaremos **expresiones regulares**. Un ejemplo es

$$(0 \cup 1)0^*.$$

El valor de la expresión aritmética es 32. El valor de la expresión regular es un lenguaje. En este caso se trataría de cadenas que comienzan con 0 o 1 y van seguidas de una cadena de 0's que puede ser incluso la cadena vacía ε.

2.3.1. Definición formal de una expresión regular

Diremos que R es una **expresión regular** si R es:

1. a para algún a de un alfabeto Σ.
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, donde R_1 y R_2 son expresiones regulares,
5. $(R_1 R_2) = (R_1 \circ R_2)$, donde R_1 y R_2 son expresiones regulares,
6. $(R_1)^*$ donde R_1 es una expresión regular.

Los puntos 1,2 y 3 representan los lenguajes $\{a\}$, $\{\varepsilon\}$ y $\emptyset$ respectivamente. No hay que confundir $\{\varepsilon\}$ y $\emptyset$ ya que el primer lenguaje solo tiene una cadena, la vacía y el segundo no tiene ninguna cadena, por eso es el **lenguaje vacío**.

2.3.2. La equivalencia con los Autómatas Finitos

Teorema 7. *Un lenguaje es regular si y solo si existe una expresión regular que lo describe.*

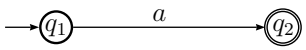
Este teorema tiene dos direcciones, probaremos cada una de ellas por separado.

Lema 1. *Si un lenguaje es descrito por una expresión regular entonces es un lenguaje regular.*

IDEA DE LA DEMOSTRACIÓN. Supongamos que tenemos una expresión regular R que describe un lenguaje $L = L(R)$. Tenemos entonces que convertir R en un AFN N de forma que $L = L(M)$. □

Demostración. Para convertir R en N consideraremos seis casos dentro de la definición formal de una expresión regular:

1. Si $R = a$ para algún $a \in \Sigma$, entonces $L(R) = \{a\}$ y el siguiente AFN reconoce a $L(R)$



   De manera formal $N = \{\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\}\}$, donde $\delta(q_1, a) = \{q_2\}$ y $\delta(r, b) = \emptyset$ para $r \neq q_1$ o $b \neq a$.

2. $R = \varepsilon$. Entonces $L(R) = \{\varepsilon\}$ y el AFN siguiente reconoce a $L(R)$:

   De manera formal $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$, donde $\delta(r, b) = \emptyset$ para cualquier r y b.

3. $R = \emptyset$. Entonces $L(R) = \emptyset$ y el siguiente AFN reconoce a $L(R)$:

   De manera formal $N = (\{q\}, \Sigma, \delta, q, \emptyset)$, donde $\delta(r, b) = \emptyset$ para cualquier r y b.

4. $R = R_1 \cup R_2$.

5. $R = R_1 \circ R_2$.

6. $R = R_1^*$.

Para los casos 4,5 y 6 emplearemos las construcciones dadas en las demostraciones de los Teoremas 2, 6 y 7 respectivamente. □

Lema 2. *Si un lenguaje es regular, entonces se puede describir empleando un expresión regular.*

IDEA DE LA DEMOSTRACIÓN. Vamos a partir la prueba de este resultado en dos partes, empleando un nuevo tipo de máquina llamado Automáta Finito No Determinista Generalizado AFNG. En primer lugar mostraremos como convertir un AFD en un AFNG, y en segundo lugar convertiremos un AFNG en una expresión regular.

Los AFNG's son AFN's en las que *las etiquetas de las flechas de su diagrama de transición están formadas por expresiones regulares*, en lugar de elementos del alfabeto o la palabra vacía ε. Por conveniencia, eligiremos siempre un AFNG que tiene la forma especial siguiente:

- El estado inicial q_{start} tiene flechas hacia cualquier otro estado, pero cualquier otro estado no tiene ninguna flecha sobre el estado inicial.

- Existe unicamente un estado de aceptación q_{accept} y tiene flechas que llegan de cualquier otro estado, pero no existe ninguna que salga hacia cualquier otro estado. El estado inicial es distinto del estado de aceptación.

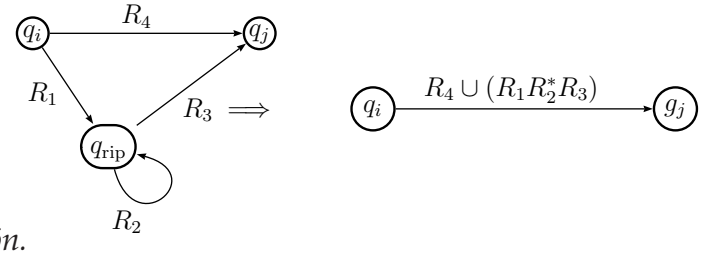
- Excepto para el estado inicial y de aceptación una sola flecha va para cualquier otro estado y una también de cada estado en si mismo.

Podemos convertir facilmente un AFD a un AFGN descrito en la forma especial como sigue: Añadimos un nuevo estado inicial con una flecha etiquetada con ε sobre el antiguo estado inicial, y un nuevo estado de aceptación, al que le llega una fecha etiquetada con ε de cada uno de los antiguos estados de aceptación. Si cualquier flecha tiene multiples etiquetas (o si hay multiples flechas que tienen la misma dirección entre los mismos estados) reemplazamos cada una de ellas por una flecha cuya etiqueta es la unión de todas las etiquetas previas. Finalmente, añadimos flechas etiquetadas $\emptyset$ entre estados que no tengan flechas.

Veamos ahora como convertir un AFNG en una expresión regular. Digamos que el AFNG tiene k–estados. Como q_{start} y q_{accept} tienen que ser diferentes se tiene que $k \geq 2$. Si $k > 2$, construimos el AFNG en su forma equivalente con $k-1$ estados. Este paso se repite hasta reducir a un AFNG con dos estados. Si $k = 2$, el AFNG tiene una única flecha que va de q_{start} a q_{accept}. La etiqueta de esta flecha es una expresión regular equivalente al lenguaje que genera el AFD.

El paso crucial es la construcción del AFNG equivalente cuando $k > 2$. Para realizar este paso seleccionaremos un estado, que será el que desaparezca, para reducir a $k-1$ estados, y tendremos que "parchear" la máquina para que el lenguaje que reconozca siga siendo el mismo. Para ello cualquier estado valdrá, siempre y cuando no se trate de q_{start} y q_{accept}. Esto queda garantizado ya que $k > 2$. Al estado seleccionado lo denotaremos por q_{rip}.

Después de eliminar q_{rip} repararemos la máquina alterando las expresiones regulares que etiquetan cada una de las flechas. Las nuevas etiquetas deberán compensar la ausencia de q_{rip}, añadiendo las expresiones perdidas. La nueva etiqueta del estado q_i al estado q_j es una expresión regular que describe todas las cadenas de q_i a q_j via q_{rip}. Ver el gráfico adjunto:



□

Demostración.

Vamos a escribir formalmente la idea expresada anteriormente. Para facilitar la demostración vamos a introducir la definición formal del tipo de autómata ya introducido. Un AFNG es similar al un AFN excepto en la definición de la función de transición

que en este caso se escribe como

$$\delta : (Q \setminus \{q_{\text{accept}}\}) \times (Q \setminus \{q_{\text{start}}\}) \longrightarrow \mathcal{R}.$$

El símbolo $\mathcal{R}$ denota la colección de todas las expresiones regulares sobre el alfabeto Σ, y q_{start} y q_{accept} son el estado inicial y de aceptación, respectivamente. Si $\delta(q_i, q_j) =$ la etiqueta de la flecha que va del estado q_i al estado q_j es la expresión regular R. El dominio de la función de transición es $(Q \setminus \{q_{\text{accept}}\}) \times (Q \setminus \{q_{\text{start}}\})$ ya que una flecha conecta cualquier estado con cualquier estado, excepto que no hay flechas que vayan a q_{start} o que salgan de q_{accept}.

Definición 4. *Un Autómata Finito No Determinista Generalizado es una* 5*–upla*

$$G = (Q, \sigma, \delta, q_{\text{start}}, q_{\text{accept}})$$

donde

1. Q *es el conjunto finito de estados.*
2. Σ *es el alfabeto de entrada.*
3. $\delta : (Q \setminus \{q_{\text{accept}}\}) \times (Q \setminus \{q_{\text{start}}\}) \longrightarrow \mathcal{R}$ *es la función de transición.*
4. q_{start} *es el estado inicial.*
5. q_{accept} *es el estado de aceptación.*

Un AFNG acepta la cadena w en Σ^* si $w = w_1 w_2 \cdots w_k$ donde cada $w_i \in \Sigma$ y una sucesión de estados $q_0, q_1, \ldots, q_k$ existe de forma que

1. $q_0 = q_{\text{start}}$,
2. $q_k = q_{\text{accept}}$ y
3. para cada i, se tiene que $w_i \in L(R_i)$ donde $R_i = \delta(q_{i-1}, q_i)$; en otras palabras R_i es la expresión que etiqueta la flecha entre q_i y q_j

Retornando a la idea de demostración, sea M el AFD para el lenguaje L. Entonces vamos a transformar M a un AFNG N añadiendo un nuevo estado y un nuevo estado de aceptación y nuevas flechas transacionales si fuese necesario. Vamos a emplear el procedimiento $\text{CONVERT}(G)$, que toma un AFNG y devuelve una expresión regular equivalente. Este procedimiento emplea un argumento **recursivo**, que emplea llamadas a si mismo. No puede darse un bucle infinito ya que el procedimiento se llama a si mismo solo para procesar un AFNG que tiene un estado menos. El caso en el que AFNG tiene exclusivamente dos estados se trata sin emplear el argumento recursivo.

$\text{CONVERT}(G)$:

1. Sea K el número de estados de G.
2. Si $k = 2$, entonces G tiene que tener unicamente un estado de inicio, un estado de aceptación y una flecha uniendólos etiquetadas con una expresión regular R. Delvolver la expresión regular R.
3. Si $k > 2$, seleccionar cualquier estado $q_{\text{rip}} \in Q \setminus \{q_{\text{start}}, q_{\text{accept}}\}$ y sea G' es AFNG $(Q', \sigma, \delta', q_{\text{start}}, q_{\text{accept}})$ donde
$$Q' = Q \setminus \{q_{\text{rip}}\}$$
y para cualquier $q_i \in Q' \setminus \{q_{\text{accept}}\}$ y cualquier $q_j \in Q' \setminus \{q_{\text{start}}\}$ sea
$$\delta'(q_i, q_j) = (R_1 R_2^* R_3) \cup R_4$$
para $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$ y $R_4 = \delta(q_i, q_j)$.
4. Calcular $\text{CONVERT}(G')$ y devolver su valor.

Vamos a demostrar que el procedimiento anterior devuelve el valor adecuado

Enunciado 1. *Para cualquier AFNG G, $\text{CONVERT}(G')$ es equivalente a G.*

Demostraremos este enunciado por inducción en k el número de estados del AFNG.

Base: Demostremos el enunciado para $k = 2$. Si G tiene exclusivamente dos estados, ellos deben de estar unidos por una única flecha, que lleva el estado inicial al estado de aceptación. La expresión regular que etiqueta a esta flecha describe todas las cadenas que para G llegan al estado de aceptación. En consecuencia, esta expresión es equivalente a G.

Paso inductivo: Supongamos el enunciado cierto para $k-1$ estados y empleemos esta hipótesis para demostrar que el enunciado es cierto para k estados.

En primer lugar demostraremos que G y G' reconocen el mismo lenguaje.

Supongamos en primer lugar que G acepta la cadena w. Entonces una rama del proceso de w por G viene determinada por la sucesión finita de estados:

$$q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}}.$$

Si niguno de ellos es el estado a eliminar q_{rip}, de forma clara G' acepta a w. La razón es que cada una de las nuevas expresiones regulares etiquetando las flechas de G' contiene la antigua expresión regular como parte de la unión. Si q_{rip}, aparece eliminando cada ejecución de q_{rip} estado aparece una forma de procesamiento aceptada por G' Los estados q_i y q_j que nos dan un paso de procesado tienen una nueva expresión regular que los liga y que describe todas las cadenas que van de q_i a q_j pasando por q_{rip}.

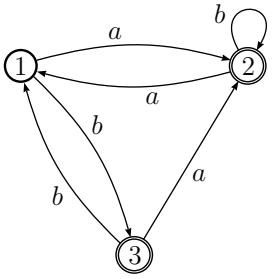
En la otra dirección, supongamos que G' acepta la cadena w. Como cada flecha entre dos estados q_i y q_j en G' describe la colección de cadenas que van de q_i a q_j

directamente o via q_{rip}, G debe de aceptar w. En consecuencia G y G' son equivalentes, esto es, $L(G) = L(G')$.

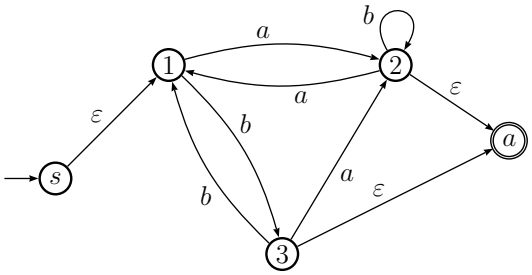
La hipótesis de inducción enuncia que cuando el algoritmo se llama a si mismo recursivamente sobre un input G' el resultado es una expresión regular que es equivalente a G' ya que G' tiene $k - 1$ estados. Luego la expresión regular es equivalente a G y se ha demostrado que el algoritmo funciona correctamente.

Con esto concluimos la demostración del Enunciado 1, del Lema 2 y, finalmente del Teorema 7. □

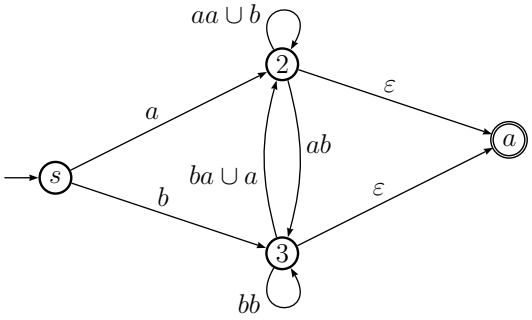
Ejemplo 22. *Construir la expresión regular que caracteriza el lenguaje aceptado por el siguiente AFD:*



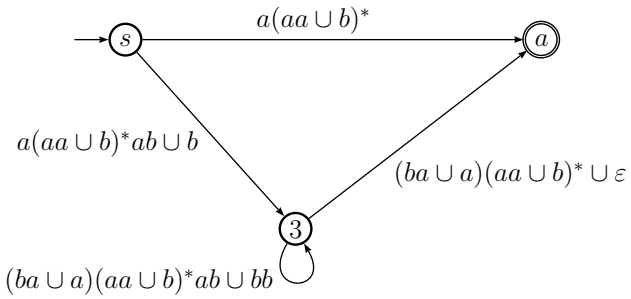
Paso 1:



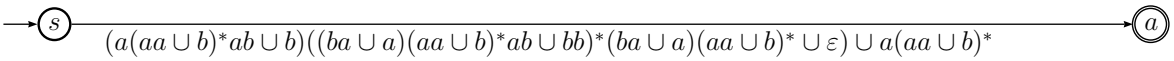
Paso 2:



Paso 3:



Paso 4:



2.4. Lenguajes no regulares

Para comprender la potencia de cálculo de un autómata finito hay que comprender bien sus limitaciones. En esta sección mostraremos como se puede probar que determinados lenguajes no pueden ser reconocidos por un autómata finito.

Consideremos el lenguaje $L_B = \{0^n 1^n | n \geq 0\}$. Si intentamos construir un AFD que reconozca a L_B, descubriremos que la máquina necesitará recordar el número de ceros que ha leido previamente. Debido a que el número de ceros que puede leer en primer lugar es ilimitado, la máquina tendrá que preveer un número ilimitado de posibilidades. Esto queda fuera de una máquina con un número finito de estados.

Ahora, presentaremos un método para probar que lenguajes como L_B no son regulares. Podríamos pensar que el anterior argumento es suficiente para ello ya que el número de ceros no está limitado. Esto no es un argumento riguroso ya que la necesidad de necesitar cantidades no acotadas de memoria no va a ser suficiente para ello. Puede serlo para L_B, pero existen otros lenguajes que parecen necesitar una cantidad ilimitada de memoria pero aún asi son regulares. Por ejemplo consideremos los siguientes lenguajes sobre el alfabeto $\Sigma = \{0, 1\}$:

$$L_C = \{w | w \text{tiene el mismo numero de } 0\text{'s y } 1\text{'s}\}$$

$$L_D = \{w | w \text{tiene el mismo numero de ocurrencias de } 01 \text{ y } 10 \text{ como subcadenas}\}$$

En una primera mirada ninguno de estos lenguajes parece ser regular. Como esperábamos L_C lo es, pero sorprendentemente L_D es regular ! Por lo tanto, nuestra intuición

puede, en algunas ocasiones, llevarnos a equivocación. Esta es la razón por la que necesitamos la ayuda de las demostraciones matemáticas para asegurar los resultados. En esta sección mostraremos como probar que determinados lenguajes no son regulares

2.4.1. El Lema del Bombeo para Lenguajes Regulares

La técnica para probar no regularidad es consecuencia de un resultado acerca de los lenguajes regulares, conocido con el nombre del **Lema del Bombeo**. Este teorema nos dice que todos los lenguajes regulares tienen una propiedad especial. Si podemos demostrar que un lenguaje no la posee, entonces podemos asegurar que dicho lenguaje no es regular. La propiedad en cuestión es que todas las cadenas del lenguaje se pueden **bombear** si tienen como mínimo una determinada longitud llamada **longitud de bombeo**. Esta quiere decir que cada una de estas cadenas contiene una sección que puede ser repetida cualquier número de veces con el resultado de que la cadena resultante siga perteneciendo al lenguaje.

Teorema 8 (Lema del Bombeo)**.** *Si A es un lenguaje regular, entonces existe un número p (llamada* longitud del bombeo*) donde, si s es cualquier cadena de A de forma que $|s| \geq p$ entonces s puede dividirse en tres piezas, $s = xyz$ que satifacen las condiciones siguientes:*

1. *para cada $i \geq 0$, $xy^iz \in A$,*
2. *$|y| > 0$, y*
3. *$|xy| \leq p$.*

Nota 1. *Recordar la notación donde $|s|$ denota la longitud de la cadena s, $y^i = \overbrace{yy \cdot y}^{i-\text{veces}}$, e $y^0 = \varepsilon$. Cuando s es dividida en xyz o bien x o bien z pueden ser ε, pero la condición 2 impone que $y \neq \varepsilon$. Observemos que sin dicha condición el Teorema es cierto de forma trivial. La condición 3 nos dice que las subcadenas x e y juntas han de tener longitud como máximo de p. Esta es una condición extra de caracter técnico, que ocasionalmente puede resultar útil para demostrar que ciertos tipos de lenguajes no son regulares.*

Ejercicios propuestos

1. Considerar los siguientes diagramas de estado de la Figura 2.9 para dos AFD M_1 y M_2. Responder a las cuestiones siguientes:

   a) ¿Cuál es el estado inicial de M_1?

   b) ¿Cuál es el conjunto de estados de aceptación para M_1?

   c) ¿Cuál es el estado inicial de M_2?

   d) ¿Cuál es el conjunto de estados de aceptación para M_2?

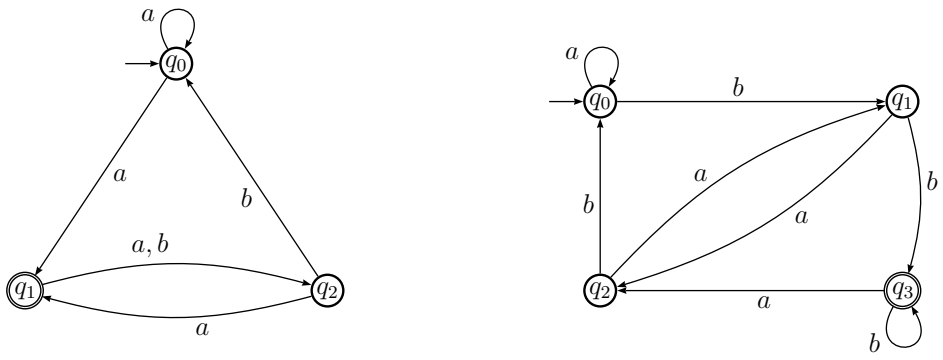


Figura 2.9: Los AFD M_1 y M_2.

e) ¿Qué sucesión de estados produce M_1 al procesar la cadena $aabb$?

f) ¿Acepta M_1 la cadena $aabb$?

g) ¿Acepta M_2 la cadena vacía ε?

2. Da la descripción formal de los autómatas M_1 y M_2 del ejercicio 1.

3. La descripción formal de un AFD M es

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{u, d\}, \delta, q_2, \{q_2\}),$$

donde δ viene descrito por la tabla siguiente:

	u	d
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_1	q_3
q_3	q_2	q_4
q_4	q_3	q_4

Construye el diagrama de estados de este autómata.

4. Da los diagramas de estados de AFD que reconozcan los siguientes lenguajes. En todos los casos el alfabeto es $\{0, 1\}$

a) $\{w|w$ empieza con 0 y acaba con $1\}$.

b) $\{w|w$ contiene al menos 3 unos$\}$.

c) $\{w|w$ contiene la subcadena 0101$\}$.

d) $\{w|w$ tiene al menos longitud 3 y el tercer simbolo es $0\}$.

e) $\{w|w$ empieza con 0 y tiene longitud impar o empieza con 1 y tiene longitud par$\}$.

f) $\{w|w$ no contiene la subcadena 110$\}$.

g) $\{w|w$ la longitud de w es al menos 5$\}$.

h) $\{w|w$ es cualquier cadena excepto 11 y 111$\}$.

i) $\{w|w$ cualquier posicion impar es un 1$\}$.

j) $\{w|w$ contiene al menos dos 0′s y al menos un 1$\}$.

k) $\{\varepsilon, 0\}$.

l) $\{w|w$ contiene un numero par de 0′s o exactamente dos 1′s$\}$.

m) El conjunto vacío.

n) Todas las cadenas excepto la cadena vacía.

5. Construye los AFN con el número de estados que se especifican a continuación y que reconocen cada uno de los siguientes lenguajes.

   a) $\{w|w$ acaba con 00$\}$ con tres estados.

   b) El lenguaje del Ejercicio 4c con cinco estados.

   c) El lenguaje del Ejercicio 4l con seis estados.

   d) El lenguaje $\{0\}$ con dos estados.

   e) El lenguaje $0^*1^*0^*0$ con tres estados.

   f) El lenguaje $\{\varepsilon\}$ con un estado.

   g) El lenguaje 0^* con un estado.

6. Usa la construcción empleada en el Teorema 6 para construir los diagramas de los AFN que reconocen la unión de los lenguajes descritos en:

   a) Ejercicios 4a y 4b.

   b) Ejercicios 4c y 4f.

7. Usa la construcción empleada en el Teorema 7 para construir los diagramas de los AFN que reconocen la concatenación de los lenguajes descritos en:

   a) Ejercicios 4g y 4i.

   b) Ejercicios 4b y 4m.

8. Usa la construcción empleada en el Teorema **??** para construir los diagramas de los AFN que reconocen la clausura de los lenguajes descritos en:

   a) Ejercicio 4b.

   b) Ejercicio 4j.

c) Ejercicio 4m.

9. Demuestra que cualquier AFN puede transformarse en uno equivalente con un único estado de aceptación.

10. *a*) Demuestra que si M es un AFD que reconoce un lenguaje B, intercambiando los estados de aceptación y no–aceptación en M obtenemos un nuevo AFD que reconoce el lenguaje complementario de B, que es el conjunto $\Sigma^* \setminus B = \{w | w \notin B\}$. Concluye que la clase de los lenguajes regulares es cerrada bajo la operación del complemento.

    b) Prueba mediante un ejemplo que si M es un AFN que reconoce un lenguaje C, intercambiando los estados de aceptación y no aceptación en M no necesariamente obtenemos un nuevo AFN que reconoce el lenguaje complementario de C. ¿Podemos afirmar entonces que la clase de lenguajes que reconocen los AFN's es cerrada para la operación del complemento. Explica brevemente tu respuesta.

11. Da un contraejemplo para probar que la construcción siguiente falla para la demostración del Teorema **??** donde se afirma que la clase de los lenguajes regulares es cerrada para la clausura. Supongamos que $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ reconoce A_1. Construye $N = (Q_1, \Sigma, \delta, q_1, F)$ como sigue. Suponemos que N reconoce A_1^*.

    a) Los estados de N son los mismos estados que los de N_1.

    b) El estado inicial de N es el mismo estado inicial de N_1.

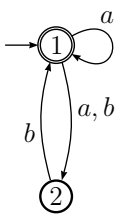
    c) $F = \{q_1\} \cup F_1$. Los estados de aceptación F son los antiguos estados de aceptación más el estado inicial.

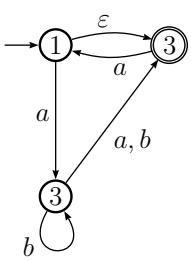
    d) Define δ de forma que para cualquier $q \in Q$ y cualquier $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{si} \quad q \notin F_1 \text{ o } a \neq \varepsilon, \\ \delta_1(q,a) \cup \{q_1\} & \text{si} \quad q \in F_1 \text{ y } a = \varepsilon. \end{cases}$$

    (Indicación: Transforma esta construcción formal en un gráfico.)

12. Emplea la construcción dada en la demostración del Teorema 5 convierte los siguientes AFN en sus equivalentes AFD





13. Encuentra las expresiones regulares que generan los lenguajes del ejercicio 4.

14. Emplea el procedimiento descrito en el Lema **??** para convertir las siguientes expresiones regulares a un AFN.

    a) $(0 \cup 1)^*000(0 \cup 1)^*$.

    b) $(((00)^*(11)) \cup 01)^*$.

    c) $\emptyset^*$.

15. Para cada uno de los lenguajes siguientes, construye dos cadenas que formen parte y dos que no formen parte del mismo. Suponer que el alfabeto es $\Sigma = \{a, b\}$ en todos los casos.

    a) a^*b^*.

    b) $a(ba)^*b$.

    c) $a^* \cup b^*$.

    d) $(aaa)^*$.

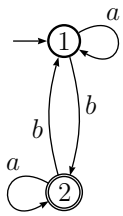
    e) $\Sigma^*a\Sigma^*b\Sigma^*a\Sigma^*$.

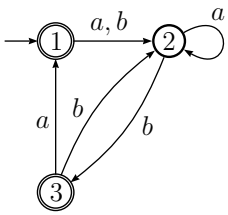
    f) $aba \cup bab$.

    g) $(\varepsilon \cup a)b$.

    h) $(a \cup ba \cup bb)\Sigma^*$.

16. Emplear el procedimiento descrito en el Lema **??** para transformar los siguientes autómatas a expresiones regulares





17. Emplear el lema del bombeo para demostrar que los lenguajes siguientes no son regulares.

    a) $A_1 = \{0^n1^n2^n | n \geq 0\}$.

    b) $A_2 = \{www | w \in \{a, b\}^*\}$.

    c) $A_3 = \{a^{2^n} | n \geq 0\}$.

18. Describe el error en la siguiente demostración de que 0^*1^* no es un lenguaje regular. (Algún error ha de existir ya que 0^*1^* es una expresión regular). La demostración es or reducción al absurdo. Supongamos que 0^*1^* es regular. Sea p la longitud de bombeo para 0^*1^*. Elijamos $s = 0^p1^p$. Conocemos que s está en 0^*1^* pero conocemos que 0^p1^p no se puede bombear. En consecuencia, 0^*1^* no es regular.

Capítulo 3

Lenguajes Libres del Contexto

En este segundo capítulo nos ocuparemos de ampliar la clase lenguajes que son reconocidos por una máquina teórica. Para ello introduciremos, en primer lugar, las llamadas Gramáticas Libres del Contexto que permiten construir los llamados Lenguajes Libres del Contexto. Este clase, mucha más amplia, incluirá a los Lenguajes Regulares, y probaremos la existencia lenguajes no regulares que son lenguajes libres del contexto. Para construir un mecanismo que procese a estos lenguajes, construiremos los llamados Autómatas a Pila y demostraremos que los lenguajes reconocidos por dichas máquinas coinciden con los Lenguajes Libres del Contexto.

Finalmente, al igual que en capítulo anterior, construiremos un mecanismo para determinar si un lenguaje dado no es libre del contexto.

3.1. Gramáticas Libres del Contexto

Una gramática libre del contexto, que podemos denotar por G_1, se puede visualizar mediante una serie de reglas gramáticales como las que mostramos a continuación

$$A \to 0A1$$
$$A \to B$$
$$B \to \#.$$

Una gramática consiste en una collección de **reglas gramaticales o de sustitución,** también llamadas **producciones** Cada regla aparece como una línea en la gramática y comprende un símbolo y una cadena separadas por una flecha. El símbolo es conocido como **variable**. La cadena está compuesta por una serie de variables y otros símbolos llamados **terminales**. Los símbolos que representan variables suelen ser letras mayúsculas. Los símbolos terminales son los análogos al los símbolos del alfabeto de entrada y se suelen representar mediante letras minúsculas, números o símbolos especiales. Una de las variables es designada como la variable de **inicio**. Usualmente ocupa el primer lugar a la izquierda en la primera regla de sustitución.

Por ejemplo la gramática G_1 poseé tres reglas de sustitución, las variables A y B, donde A es la variable de inicio. Los símbolos terminales son $0, 1$ y $\#$.

Podemos emplear una gramática para describir un lenguaje generando cada cadena del mismo del modo siguiente:

1. Escribamos la variable de inicio que usualmente se encuentra el primer lugar a la izquierda en la primera regla de sustitución, excepto que se indique cualquier otra cosa.

2. Encontrar una variable que previamente hallamos escrito y una regla que empieze con dicha variable. Reemplazar la variable previamente escrita con la parte derecha dicha regla . Por ejemplo en G_1 escribimos la variable A, y elegimos la regla $A \to 0\mathbf{A}1$ reemplazamos **A** por $0A1$ y obtenemos $A \to 00A11$.

3. Repetir el paso anterior hasta que no aparezcan variables.

Por ejemplo la gramática G_1 genera la cadena $000\#111$. La sucesión de sustituciones para obtener una cadena se le llama una **derivación**. Una derivación de la cadena $000\#111$ en la gramática G_1 es

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111.$$

Todas las cadenas generadas de esta forma constituyen el **lenguaje de una gramática**. Denotaremos por $L(G_1)$ al lenguaje generado por la gramática G_1. Trabajando sobre la gramática G_1 podemos demostrar que $L(G_1) = \{0^n\#1^n | n \geq 0\}$. Cualquier lenguaje que puede ser generado por una gramática libre del contexto se le llama **Lenguaje Libre del Contexto** Por conveniencia cuando presentemos una gramática libre del contexto simplificaremos la presentación de las reglas que tengan la misma variable en el lado izquierdo, como por ejemplo $A \to 0A1$ y $A \to B$ las escribiremos como $A \to 0A1|B$, empleando el símbolo "|" como "o".

3.1.1. Definicición formal de una Gramática Libre del Contexto

Formalicemos la definición de gramática libre del contexto GLC.

Definición 5. *Una **Gramática Libre del Contexto** es una 4–upla $G = (V, \Sigma, R, S)$ donde*

1. *V es un conjunto finitos llamado cuyos elementos son llamados **variables**.*

2. *Σ es un conjunto finito, disjunto de V, cuyos elementos son llamados **símbolos terminales**.*

3. *R es un conjunto finito de **reglas gramaticales o producciones,** cada una de ellas está compuesta por una variable o por una cadena de variables y símbolos terminales.*

4. *$S \in V$ es el **símbolo inicial**.*

Si u, v y w son cadenas de variables y terminales, y $A \to w$ es una regla gramatical, diremos que uAv **deriva** uwv, y escribiremos $uAv \Rightarrow uwv$. Escribiremos $u \Rightarrow^* v$ si $u = v$ o si existe una sucesión finita $u_1, u_2, \ldots, u_k$ para $k \geq 1$ de forma que

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v.$$

El **lenguaje generado por la gramática** G y que denotaremos por $L(G)$, viene definido del modo siguiente $L(G) = \{w \in \Sigma^* | S \Rightarrow^* w\}$.

3.1.2. Gramáticas sin símbolos inútiles

Diremos que un símbolo A es **útil** para la gramática $G = (V, \Sigma, R, S)$ is existe una derivación de la forma $S \Rightarrow^* uAv \Rightarrow^* w \in \Sigma^*$, lo que nos asegura que w forma parte del lenguaje generado por la gramática. A los símbolos que no son útiles les denominaremos **símbolos inútiles**.

Diremos que A es un **símbolo generador** si $A \Rightarrow^* w$ para alguna cadena $w \in \Sigma^*$. Cualquier símbolo terminal, es decir de Σ, es entonces un símbolo generador.

Diremos que A es un símbolo **accesible** si existe una derivación del tipo $S \Rightarrow^* uAv$.

Consideremos la gramática $G = (\{S, A, B\}, \{a, b\}, R, S)$ donde el conjunto de reglas gramaticales R viene determinado por

$$\begin{aligned} S &\to AB|a \\ A &\to b. \end{aligned}$$

Nótese que la variable B no es un símbolo generador. En consecuencia, prodríamos eliminar la producción $S \to AB$, con lo que las reglas gramaticales se verían reducidas a

$$\begin{aligned} S &\to a \\ A &\to b. \end{aligned}$$

Como A no es un símbolo accesible, podemos eliminar la producción $A \to b$. En consecuencia la gramática G reduce sus reglas a una única producción

$$S \to a.$$

Veamos como calcular tanto los símbolos no generadores como los símbolos no accesibles.

Cómputo de los símbolos no generadores

Paso 1 Cualquier símbolo de Σ es de forma obvia un generador.

Paso Inductivo Supongamos que tenemos una producción $A \to a$ siendo a un generador entonces A es generador. Esta regla incluye el caso $a = \varepsilon$; todas las variables que producen a la palabra vacía son generadoras.

Los símbolos que no aparezcan en la lista así obtenida serán los símbolos no generadores.

Cómputo de los símbolos no accesibles

Paso 1 S es un símbolo accesible.

Paso Inductivo Supongamos que conocemos que A es accesible, si $A \to uBv$, entonces u, B y v son accesibles.

Los símbolos que no aparezcan en la lista así obtenida serán los símbolos no accesibles.

Teorema 9. *Sea $G = (V, \Sigma, R, S)$ una gramática libre del contexto de forma que el lenguaje que genera es distinto del lenguaje $\emptyset$. Sea $G_1 = (V_1, \Sigma_1, R_1, S)$ la gramática obtenida eliminando primero los símbolos no generadores y después los símbolos no accesibles. Entonces G_1 no tiene símbolos inútiles y genera el mismo lenguaje que la gramática G, esto es, $L(G) = L(G_1)$.*

Demostración. Ver Teorema 7.2 en [1] □

En virtud del anterior resultado podemos suponer partir de ahora que todas las gramáticas libres del contexto no poseeran símbolos inútiles, ya que en caso contrario emplearíamos el anterior argumento.

3.1.3. Diseño de Gramáticas Libres del Contexto

Las técnicas que presentaremos a continuación pueden ser útiles empleadas aisladamente o en conjunción cuando nos enfrentemos al problema de construir una GLC.

En primer lugar muchas GLC son la unión de GLC más simples. Si deseamos construir una GLC para un Lenguaje Libre del Contexto LLC desmenbremos la original en piezas más pequeñas que podamos caracterizar. Estas gramáticas individuales pueden facilmente combinarse añadiendo una nueva regla del tipo $S \to S_1|S_2|\cdots|S_k$, donde las variables $S_1, S_2, \ldots, S_k$ son las variables iniciales para las gramáticas individuales. Resolver muchos problemas simples es mejor que resolver un problema más complicado.

Por ejemplo para construir la gramática del lenguaje

$$\{0^n1^n|n \geq 0\} \cup \{1^n0^n|n \geq 0\},$$

construyamos la gramática

$$S_1 \to 0S_11|\varepsilon$$

para el lenguaje

$$\{0^n1^n|n \geq 0\},$$

y la gramática

$$S_2 \to 1S_20|\varepsilon$$

para el lenguaje

$$\{1^n0^n|n \geq 0\}.$$

Finalmente, añadimos la regla

$$S \to S_1|S_2$$

para dar la gramática

$$S \rightarrow S_1|S_2$$
$$S_1 \rightarrow 0S_1 1|\varepsilon$$
$$S_2 \rightarrow 1S_2 0|\varepsilon$$

En segundo lugar, construir una GLC para un lenguaje que parece regular es fácil si previamente se puede construir un AFD para dicho lenguaje. Se puede convertir el AFD en la GLC como sigue: Tomar la variable R_i para cada estado q_i. Añadir la regla $R_i \rightarrow aR_j$ a la GLC si $\delta(q_i, a) = q_j$ es una transición del AFD. Añadir la la regla $R_i \rightarrow \varepsilon$ si q_i es un estado de aceptación del AFD. Tomar R_0 como variable de inicio, donde q_0 es el estado inicial del AFD. Verificar finalmente que la GLC resultante genera el mismo lenguaje que el que el AFD reconoce o acepta.

En tercer lugar, ciertos LLC contienen cadenas con dos subcadenas que están unidas en el sentido que una máquina que reconozca dicho lenguaje necesite recordar una cantidad no acotada de información acerca de una de estas subcadenas. Esta situación se da en el lenguaje $\{0^n 1^n | n \geq 0\}$ debido a que la máquina necesita recordar el número de ceros para verificar que necesita el mismo número de unos. Se puede construir entonces una GLC empleando una regla del tipo $R \rightarrow uRv$, que genera cadenas cuya porción de u's corresponde a la porción correspondiente de v's.

Finalmente, para lenguajes más complejos, las cadenas pueden contener ciertas estructuras que aparecen recursivamente como parte de otras (o de las mismas) estructuras. Esta situación ocurre en el ejemplo siguiente:

Consideremos la gramática $G_2 = (V, \Sigma, R, \langle EXPR \rangle)$. Donde

$$\Sigma = \{a, +, \times, (,)\}$$
$$V = \{\langle EXPR \rangle, \langle TERM \rangle, \langle FACTOR \rangle\}$$

y las reglas son

$$\langle EXPR \rangle \rightarrow \langle EXPR \rangle + \langle TERM \rangle | \langle TERM \rangle$$
$$\langle TERM \rangle \rightarrow \langle TERM \rangle \times \langle FACTOR \rangle | \langle FACTOR \rangle$$
$$\langle FACTOR \rangle \rightarrow (\langle EXPR \rangle) | a.$$

Esta gramática genera cadenas del tipo $a + a \times a$ y $(a + a) \times a$. Nótese que el símbolo a aparece siempre al igual que una expresión entre paréntesis de forma recursiva debido a que

$$\langle FACTOR \rangle \Rightarrow (\langle EXPR \rangle) \Rightarrow (\langle TERM \rangle) \Rightarrow (\langle FACTOR \rangle) \Rightarrow ((\langle EXPR \rangle)).$$

3.1.4. Ambiguedad en las Gramáticas

3.1.5. La Forma Normal de Chomsky

Definición 6. *Una gramática libre del contexto se encuentra escrita en* ***Forma Normal de Chomsky*** *si cualquier regla es del tipo*

$$A \rightarrow BC$$
$$A \rightarrow a$$

donde a es cualquier símbolo terminal y A, B, C son variables, con la excepción de que ni B ni C pueden ser el símbolo inicial. Además, añadiremos la posibilidad de que aparezca la regla $S \to \varepsilon$, donde S es el símbolo inicial.

Teorema 10. *Cualquier lenguaje libre del contexto está generado por una gramática libre del contexto escrita en la Forma Normal de Chomsky.*

IDEA DE LA DEMOSTRACIÓN. Vamos a dar las pautas para escribir cualquier gramática G en la forma normal de Chomsky. La conversión tiene diferentes pasos en donde las reglas que violan las condiciones van a ser sucesivamente reemplazadas por otras equivalentes y que satifacen los requerimientos de la forma normal de Chomsky. En primer lugar, añadiremos un nuevo símbolo inicial. Entonces, eliminaremos todas las reglas ε de la forma $A \to \varepsilon$. También, eliminaremos todas las reglas unitarias de la forma $A \to B$. En ambos casos la gramática será parcheada para asegurarnos de que sigue reconociendo el mismo lenguaje. Finalmente, reconvertiremos las restantes reglas a la forma adecuada. □

Demostración. En primer lugar, añadiremos un nuevo símbolo inicial S_0 y la regla $S_0 \to S$, donde S es el antiguo símbolo inicial. Este cámbio nos garantiza que el símbolo inicial no aparezca en el lado izquierdo de una regla gramatical.

En segundo lugar nos ocuparemos de las reglas ε. Eliminaremos cualquier regla del tipo $A \to \varepsilon$, donde A no sea el símbolo inicial. Entonces, para cada ocurrencia de un A en el lado derecho de una regla, añadiremos una nueva regla en donde este símbolo desaparece. En otras palabras si $R \to uAv$ es una regla donde u y v son cadenas de variables y terminales, añadiremos entonces $R \to uv$. Esto lo realizaremos para cada ocurrencia de un A, en consecuencia la regla $R \to uAvAw$ nos producirá el añadir $R \to uAvw$, $R \to uvAw$ y $R \to uvw$. Si tenemos la regla $R \to A$ añadiremos $R \to \varepsilon$, excepto si ha sido previamente eliminada. Repetiremos todas estas reglas hasta asegurarnos que se han eliminado todas las reglas ε.

En tercer lugar, eliminaremos todas las reglas unitarias. Eliminaremos $A \to B$, entonces cuando aparezca una regla del tipo $B \to u$ añadiremos $A \to u$ excepto si esto ya fue realizado al eliminar una regla unitaria anterior. Al igual que antes u es una cadena de variables y terminales. Repetiremos todos estos pasos hasta eliminar todas las reglas unitarias.

Finalmente, convertiremos todas las restantes reglas a la forma adecuada. Reemplazaremos cada regla $A \to u_1u_2 \cdots u_k$ donde $k \geq 3$ y cada u_i es una variable o símbolo terminal con las reglas $A \to u_1A_1$, $A_1 \to u_2A_2$, $A_2 \to u_3A_3, \ldots, A_{k-2} \to u_{k-1}u_k$. Las A_i's son nuevas variables. Si $k \geq 2$, reemplazaremos cualquier símbolo terminal u_i en la regla precedente con una nueva variable U_i con una nueva variable U_i y añadiremos la regla $U_i \to u_i$.

□

3.2. Autómatas a Pila

En esta sección introduciremos un nuevo tipo de modelo de computación llamado **autómata a pila**. Este autómata es del tipo de un AFN con un componente extra llamado **pila**.

Un autómata a pila, AP a partir de ahora, puede escribir símbolos en la pila y leerlos más tarde. al escribir un símbolo nuevo, los símbolos anteriores se desplazan, por asi decirlo, hacia abajo. Estos también pueden ser borrados o eliminados de la pila, además, cabe destacar que todos los accesos a la pila, tanto de lectura como de escritura, se hacen exclusivamente en la parte superior de la pila. Si escribimos una cierta información en la pila, y más adelante necesitamos acceder a ella debemos eliminar toda la información por encima de ella para poder acceder a la misma.

Como apuntamos anteriormente, los AP pueden ser no deterministas. Esta propiedad es crucial, ya que en contraste con la situación de los automatas finitos, el no determinismo añade potencia a la capacidad de cómputo a este tipo de máquinas.

3.2.1. Definición formal de un Autómata a Pila

La definición formal del autómata a pila coincide con la de un autómata finito, excepto por la apracición de la pila. La pila es un mecanismo que almacena símbolos de un alfabeto dado. La máquina puede emplear diferentes alfabetos para sus entradas o "inputs" y los elementos de la pila. En consecuencia, especificaremos dos alfabetos, el alfabeto de las cadenas de entrada Σ y el alfabeto de pila Γ.

El eje central de cualquier autómata reside en la definición de su función de transición. Recordemos que $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ y $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$. El dominio de la función de transición es $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$. En consecuencia, el estado actual más el símbolo de entrada que leemos más el símbolo de pila que tengamos en la parte superior de la pila determinaran el siguiente movimiento de un autómata a pila.

Para el rango de la función de transición necesitaremos considerar las necesidades del autómata a realizar cuando se encuentra en este tipo de situación. Este podría entrar en un nuevo estado y escribir un nuevo símbolo en la parte superior de la pila. La función δ podría indicar esta acción devolviendo un elemento de Q junto con un elemento de Γ_ε, esto es, un elemento de $Q \times \Gamma_\varepsilon$. Juntando todos estos elementos obtenemos que $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow Q \times \Gamma_\varepsilon$.

Definición 7. *Un **autómata a pila** es una* 6*–upla* $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$*, donde* Q, Σ, Γ, F *son todos conjuntos finitos, y*

1. Q *es el conjunto de estados,*
2. Σ *es el alfabeto de entrada,*
3. Γ *es el alfabeto de pila,*
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow Q \times \Gamma_\varepsilon$ *es la función de transición,*

5. $q_0 \in Q$ *es el estado inicial,*

6. $F \subset Q$ *es el conjunto de estados de aceptación.*

Un autómata a pila $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computa como sigue. Supongamos que la cadena de entrada es $w = w_1 w_2 \cdots w_m$ donde cada $w_i \in \Sigma_\varepsilon$ y que existen una sucesión de estados $r_0, r_1, \ldots r_m \in Q$ y de cadenas $s_0, s_1 \ldots, s_m \in \Gamma^*$ que cumplen las tres condiciones siguientes:

1. $r_0 = q_0$ y $s_0 = \varepsilon$. Esta condición significa que M comienza adecuadamente en el estado inicial y la pila vacía.

2. Para $i = 0, 1, \ldots, m-1$ tenemos $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, donde $s_i = az$ y $s_{i+1} = bz$ para $a, b \in \Gamma_\varepsilon$ y $z \in \Gamma^*$. Esta condición enuncia que M se mueve adecuadamente de acuerdo el estado, la pila y el siguiente símbolo de entrada.

3. $r_m \in F$. Esta condición nos dice que nos encontramos en un estado de aceptación cuando llegamos al símbolo final de la cadena de entrada.

Podemos emplear también un diagrama de estado para describir un AP. Escribiremos $a, b \to c$ para indicar que el autómata está leyendo un símbolo a y que puede reemplazar el símbolo b en la parte superior de la pila por el símbolo c. Cualquiera de los tres símbolos a, b o c pueden ser la palabra vacía ε. Si a es ε, la máquina puede tomar esta transición sin necesidad de leer el símbolo de entrada. Si b es ε, la máquina puede tomar esta transición sin necesidad de leer o reemplazar cualquier símbolo en la pila. Si c es ε, la máquina no puede escribir cualquier símbolo en la pila a lo largo de la transición.

3.2.2. Equivalencia con las Gramáticas Libres del Contexto

Teorema 11. *Un lenguaje es libre del contexto si y solo si es aceptado o reconocido por un autómata a pila.*

Lema 3. *Si un lenguaje es libre del contexto, entonces es reconocido o aceptado por un autómata a pila.*

IDEA DE LA DEMOSTRACIÓN. Sea $L = L(G)$ un lenguaje libre del contexto, donde G es una gramática libre del contexto. Tenemos que construir un AP P de forma que $L(P) = L$.

El AP P que vamos a describir trabaja aceptando una cadena w generada por la gramática G, a partir de la derivación de la misma. Recordemos que una derivación es una secuencia de sustituciones realizadas a partir de las reglas de la misma, y que permiten generar una cadena. Cada paso de la derivación nos proporciona una **cadena intermedia** de variables y terminales. Vamos a construir P para determinar cuando una serie de sustituciones empleando las reglas de G puede llevarnos de la variable inicial a la cadena w.

Una de las dificultades en la comprobación de cuando hay una derivación que permita determinar si una cadena está generada o no por la gramática G El no determinismo de un AP nos va a permitir detectar las sustituciones correctas son posibles. En cada paso de la derivación una de las reglas para una variable particular será seleccionada de forma no determinista, y empleada para sustituir a la misma.

El AP P comenzará a trabajar escribiendo la variable inicial en la pila. Llevará a cabo una serie pasos a través de una serie de cadenas intermedias, realizando una sustitución tras otra. De manera eventual pudiera llegar a una cadena que contenga solo símbolos terminales, lo que nos llevaría a poder afirmar que la cadena puede ser derivada por la gramática. Entonces, P aceptará dicha cadena si es idéntica a la cadena que ha recibido como flujo de entrada.

La implementación de esta estrategia en un AP requiere una idea adicional. Necesitamos ver como el AP almacena cadenas intermedias y va de una a otra. Para ello podríamos emplear la pila para almacenar cada cadena intermedia. Esta estrategia presenta un problema adicional ya que el AP necesita buscar las variables en las cadenas intermedias para poder realizar las sustituciones. Recordemos que el AP solo puede acceder al símbolo que se encuentra en la parte superior de la pila y que puede ser tanto una variable como un símbolo terminal. La forma de sortear este problema es el de poner unicamente parte de la cadena intermedia en la pila: los símbolos que comiezan con la primera variable en la cadena intermedia. Cualquier símbolo terminal que aparezca antes de la primera variable es sustituido inmediatamente con los símbolos de la cadena del flujo de entrada.

Podemos describir el AP P de manera informal como sigue:

1. Ponemos el símbolo de pila $\$$ y la variable de inicio S en la pila.

2. Repetir los siguientes pasos:

   a) Si en la parte superior de la pila tenemos una variable A, seleccionar de forma no determinista una de las reglas de A y sustituir por la cadena que se encuentra en el lado derecho de la regla.

   b) Si en la parte superior tenemos un símbolo terminal a leemos el símbolo siguiente de la cadena del flujo entrada y lo comparamos con a, si coinciden seguimos, en caso contrario rechazamos esta rama no determinista.

   c) Si en la parte superior de la pila está el símbolo $\$$, entramos en el estado de aceptación, y, de forma evidente, aceptamos la cadena de entrada.

□

Demostración. Vamos ahora a dar la demostración formal del Lema 3. Para ello vamos a construir el AP $P = (Q, \Sigma, \Gamma, \delta, q_1, F)$. Vamos a introducir una notación simplificada para declarar la función de transición. Esta misma nos permitirá escribir una cadena entera en la pila en un único paso de la máquina.

Sean p y r estados del AP, sea a de Σ_ε y $s \in \Gamma_\varepsilon$. Deseamos escribir la cadena $u = u_1u_2\cdots u_l$ en la pila al mismo tiempo. Podemos implementar esta acción introduciendo nuevos estados $q_1, q_2, \ldots, q_{l-1}$ y añadir las transiciones siguientes:

$$\begin{array}{lcl} \delta(q,a,s) & \supset & \{(q_1,u_l)\} \\ \delta(q_1,\varepsilon,\varepsilon) & = & \{(q_2,u_{l-1})\} \\ \delta(q_2,\varepsilon,\varepsilon) & = & \{(q_3,u_{l-2})\} \\ & \vdots & \\ \delta(q_{l-1},\varepsilon,\varepsilon) & = & \{(r,u_1)\}. \end{array}$$

Emplearemos la notación $(r,u) \in \delta(q,a,s)$ para decir que cuando el autómata está en el estado q y s es el símbolo que está en la parte superior de la pila el AP puede leer el símbolo a borrar s poner la cadena u en el pila e ir al estado r. La Figura 3.1 muestra esta implementación mediante un diagrama de estado.

Los estados de P son $Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup E$, donde E es el conjunto de estados necesarios para implementar la simplificación antes comentada. El estado inicial es q_{start} y el único estado de aceptación q_{accept}.

La función de transición se define como sigue. Empezamos inicializando la pila con los símbolos \$ y S, es decir implementado el punto 1. de la descripción informal:

$$\delta(q_{\text{start}},\varepsilon,\varepsilon) = \{q_{\text{loop}}, S\$\}.$$

Entonces ponenos todas nuestras transiciones en el paso 2, de nuestra descripción informal.

Para el caso (a) en el que en la parte superior de la pila contiene una variable. Sea

$$\delta(q_{\text{loop}},\varepsilon,A) = \{(q_{\text{loop}},w) | A \to w \text{ es una regla}\}.$$

Finalmente, en el caso (c) cuando el símbolo de final de pila \$ está en la parte superior de la misma:

$$\delta(q_{\text{loop}},\varepsilon,\$) = \{(q_{\text{accept}},\varepsilon)\}.$$

El diagrama de estado es como el que muestra la Figura 3.2

□

Lema 4. *Si un lenguaje es aceptado o reconocido por un autómata a pila entonces es un lenguaje libre del contexto.*

IDEA DE LA DEMOSTRACIÓN. Ahora, tenemos un AP P y deseamos construir una GLC G que genere todas las cadenas que acepte P. En otras palabras, G debe poder generar una cadena si esta cadena hace que el AP se mueva del estado inicial al estado de aceptación.

Para poder llevar a cabo esta tarea, diseñaremos una gramática que hace algo más. Para cada par de estados p y q en P tendremos en la gramática una variable $A_{pq.}$ Esta variable va a generar todas las cadenas tome P desde p con la pila vacía hasta q con la pila vacía. Observesé que estas cadenas nos llevarán de p a q sin mirar los contenidos de la pila en p y q.

Para simplificar nuestra tarea modificaremos P levemente del modo siguiente:

1. Tendrá un único estado de aceptación q_{accept}.

2. Este vacía la pila una vez que es aceptado.

3. Cada transición o bien pone un símbolo en la pila o bien borra un símbolo pero no realiza ambos movimientos al mismo tiempo.

Establecer 1 y 2 es sencillo. Para el caso 3, necesitamos reemplazar cada transición que pone y borra en la pila al mismo tiempo por dos transiciones que nos lleven al nuevo estado.

Para diseñar G de forma que A_{pq} genere todas las cadenas de P que llevan p a q comenzando y acabando con la pila vacía, debemos comprender como opera P sobre estas cadenas. Para cualquiera de estas cadenas x, P se moverá primero sobre x escribiendo en la pila, ya que cualquier movimiento o bien escribe o bien borra y P no puede borrar en una pila vacía. De forma similar el último movimiento sobre x será borrar, ya que la pila tiene que terminar vacía.

Dos posibilidades pueden ocurrir mientras P procesa a x. O bien el símbolo es borrado al final es el símbolo que fue escrito al principio o no. Si fuese así pila estaría vacía al principio y al final del proceso de x por P En caso contrario, el símbolo inicialmente escrito deberá ser borrado en algún punto antes de que finalice el proceso de x, y en consecuencia la pila estará vacía a partir de este punto. Simularemos esta última posibilidad con la regla

$$A_{pq} \rightarrow aA_{rs}b,$$

donde a es el símbolo del flujo de entrada leido en el primer movimiento, b es el símbolo leido en el último movimiento, r es el estado siguiente a p y s el estado que precede a q. Podemos simular esta última posibilidad con la regla

$$A_{pq} \rightarrow A_{pr}A_{sq},$$

donde r es el estado en el que la pila queda vacía. □

Demostración. Supongamos que $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$ y construyamos la gramática G. Las variables de G son $\{A_{pq} | p, q \in Q\}$. La variable de inicio es $A_{q_0,q_{\text{accept}}}$. Describamos las reglas de G :

- Para cada $p, q, r, s \in Q$, $t \in \Gamma$, y $a, b \in \Sigma_\varepsilon$, si $\delta(p, a, \varepsilon)$ contiene (r, t) y $\delta(s, b, t)$ contiene (q, ε) añadamos la regla $Apq \rightarrow aA_{rs}b$ a G.

- Para cada $p, q, r \in Q$, añadamos la regla $A_{p,q} \rightarrow A_{pr}A_{sq}$ a G.

- Finalmente, para cada $p \in Q$ añadamos la regla $A_{pq} \rightarrow \varepsilon$ a G.

Ahora demostraremos que esta construcción funciona demostrando que x está generado por A_{pq} si y solo si x puede ir en P del estado p al estado q con la pila vacía. Consideraremos cada una de estas dos implicaciones por separado.

Enunciado 2. *Si x es generada por A_{pq}, entonces x puede ir en P desde p hasta q con la pila vacía*

Demostraremos este primer enunciado por inducción sobre el número de pasos en la derivación de x empleando A_{pq}.

Base: La derivación tiene un paso

Una derivación con un único paso tiene que emplear una regla cuya parte derecha no contenga variables. Las únicas reglas de G que no tienen reglas en la parte derecha son $A_{pp} \to \varepsilon$. Claramente, la entrada ε lleva p con la pila vacía a p con la pila vacía.

Paso inductivo: Supongamos el enunciado cierto para derivaciones de longitud menores o iguales a k, con $k \geq 1$, y probemos que es cierto para derivaciones de longitud $k+1$.

Supongamos que $A_{pq} \Rightarrow^* x$ con $k+1$ pasos. El primer paso en esta derivación es o bien $A_{pq} \Rightarrow aA_{rs}b$ o $A_{pq} \Rightarrow A_{pr}A_{sq}$. Vamos a tratar estos dos casos por separado

En el primer caso, consideremos la porción y de x que genera A_{rs}, por lo tanto $x = ayb$. Como $A_{rs} \Rightarrow^* y$ tiene menos de k pasos la hipótesis de inducción nos dice que P puede ir de r con la pila vacía a s con la pila vacía. Como $A_{pq} \to aA_{rs}b$ es una regla de G, $\delta(p, a, \varepsilon)$ contiene a (r, t) y $\delta(s, b, t)$ contiene a (q, ε). Luego, si P empieza en p con la pila vacía después de leer el símbolo a puede ir al estado r y escribir t en la pila. Entonces, leyendo la cadena y podemos ir a s y dejar t en la pila. Entonces,después de leer b puede ir al estado q y borrar a t de la pila. En consecuencia, x nos lleva de p con la pila vacía a q con la pila vacía.

En el segundo caso, consideremos las porciones de y y z de x que generan A_{pr} y A_{rq} respectivamente, es decir, $x = yz$. Como $A_{pq} \Rightarrow^* y$ en menos de k pasos y $A_{rs} \Rightarrow^* z$ en menos de k pasos, la hipótesis de inducción nos dice que y puede ir de p con la pila vacía a r con la pila vacía y z puede ir de r a q ambos con la pila vacía. Luego x nos permite ir de p a q con la pila vacía en ambos casos. Esto completa el paso inductivo.

Enunciado 3. *Si x puede ir en P desde p hasta q con la pila vacía, entonces x es generado A_{pq}.*

Demostraremos este enunciado por inducción sobre el número de pasos necesarios para ir de p a q con la pila vacía sobre un flujo de entrada x.

Base: La derivación tiene 0 pasos

Si la derivación tiene 0 pasos, empieza y acaba en el mismo estado, digamos p. Por lo que deberemos demostrar que A_{pp}
$Rightarrow x$. En 0 pasos, P solo tiene tiempo de leer la cadena vacía, luego $x = \varepsilon$. Por construcción, G contiene la regla $A_{pp} \to \varepsilon$, luego esto prueba la base.

Paso inductivo: Supongamos el enunciado cierto para derivaciones de longitud menores o iguales a k, con $k \geq 1$, y probemos que es cierto para derivaciones de longitud $k+1$.

Supongamos que cuando P procesa x vamos de p a q con la pila vacía en $k+1$ pasos. O bien la pila está vacía al comienzo y al final de este proceso o bien está vacía todo el tiempo.

En el primer caso, el símbolo que es escrito al primer movimiento tiene que ser el mismo que es borrado en el último de ellos. Llamemos t a este símbolo. Sea a el primer símbolo del flujo de entrada, b el último símbolo leido, r el estado después del primer movimiento y s el estado anterior al último movimiento. Entonces $\delta(p, a, \varepsilon)$ contiene (r, t) $\delta(s, b, t)$ contiene (q, ε), por lo tanto la regla $A_{p,q} \rightarrow aA_{rs}b$ está en G.

Sea y la porción de x entre a y b, es decir $x = ayb$. La entrada y va de r a s sin necesidad de tocar el símbolo t que está en la pila y por lo tanto podemos ir entre s y r a pila vacía con la entrada y. Como se han eliminado el primer y el último paso de los $k+1$ posibles, nuestra derivación tiene $k-1$ pasos. Por lo que la hipótesis de inducción nos asegura que $A_{r,s} \Rightarrow^* y$. Luego $A_{pq} \Rightarrow^* x$.

En el segundo caso, sea r el estado donde la pila comienza a estar vacía diferente del principio o final del proceso de x por P. Entonces las porciones del proceso de p a r y de r a q cada una de ellas tiene una longitud menor de k pasos. Digamos que y es la porción leida en primer lugar y z la leida en segundo lugar. La hipótesis de inducción nos dice que $A_{pr} \Rightarrow^* y$ y $A_{rq} \Rightarrow^* z$. Como la regla $A_{pq} \rightarrow A_{pr}A_{rq}$ está en G, se cumple que $A_{pq} \Rightarrow^* x$ y la demostración está completa. Esto prueba el Lema 4 y en consecuencia el Teorema 11. □

Como cualquier AFD es una AP que pasa entre estados con la pila vacía, se tiene el siguiente corolario.

Corolario 2. *Cualquier lenguaje regular es libre del contexto*

3.3. Lenguajes No Libres del Contexto

3.3.1. El Lema del bombeo para Lenguajes Libres del Contexto

Teorema 12 (Lema del Bombeo para Lenguajes Libres del Contexto)**.** *Si A es un lenguaje libre del contexto, entonces existe un número p (la longitud de bombeo) donde si s es una cadena cualquiera con $|s| \geq p$, entonces s puede dividirse en 5 subcadenas $s = uvxyz$ que satisfacen las condiciones siguientes:*

1. *Para cada $i \geq 0$, $uv^ixy^iz \in A$,*
2. *$|vy| > 0$ y*
3. *$|vxy| \leq p$.*

Ejemplo: Un autómata a pila

Consideremos el lenguaje $\{0^n 1^n | n \geq 0\}$. Entonces el siguiente AP reconoce este lenguaje.

$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0, 1\}$,

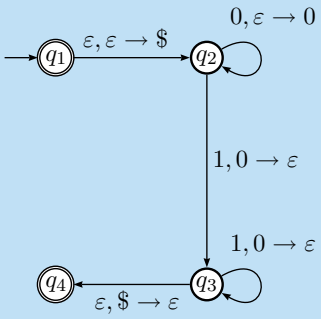
$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, y

δ viene determinada por la tabla siguiente:

Input:	0			1			ε		
Pila:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_3				$\{(q_3, \varepsilon)\}$					
q_4									

También lo podemos representar mediante el diagrama de estado siguiente:



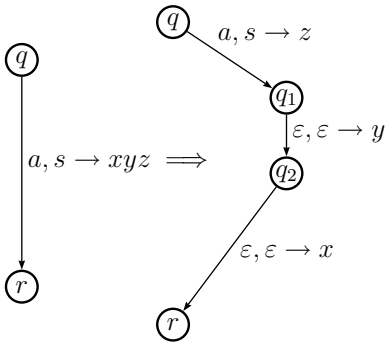


Figura 3.1: Implementación de $(r, xyz) \in \delta(q, a, s)$.

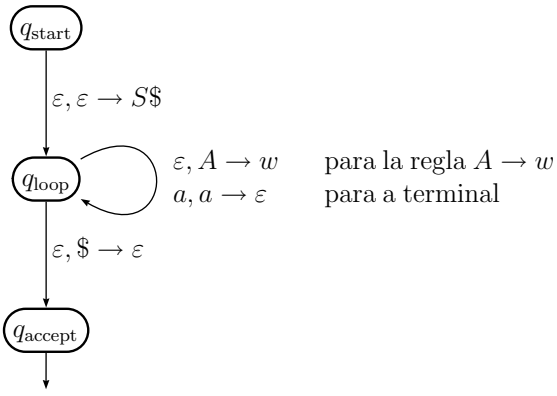


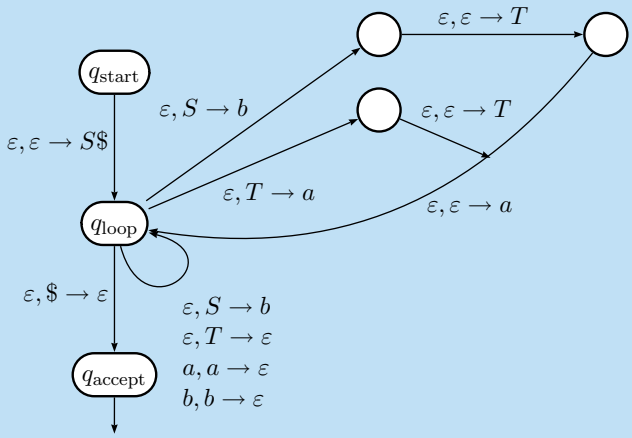
Figura 3.2: Diagrama de estado para P.

Ejemplo del procedimiento descrito en el Lema 3

Construyamos un AP P_1 a partir de la siguiente GLC G

$$S \to aTb|b$$
$$T \to Ta|\varepsilon$$

En el paso 1. establecemos $\varepsilon, \varepsilon \to S\$$. En el paso 2 (a) establecemos a partir de las reglas $S \to b$ y $S \to \varepsilon$ establecemos en primer lugar $\varepsilon, S \to b$ y $\varepsilon, T \to \varepsilon$. En segundo lugar de las reglas $S \to aTb$ y $T \to Ta$ tenemos que emplear el mecanismo que nos permite introducir una cadena entera en la pila. Esto nos produce para el primer caso dos estados nuevos y las transiciones $\varepsilon, S \to b$, $\varepsilon, \varepsilon \to T$ y $\varepsilon, \varepsilon \to a$. En el segundo un estado nuevo y las transiciones $\varepsilon, T \to a$ y $\varepsilon, \varepsilon \to T$. Finalmente, 2(c) nos da las transiciones para los símbolos terminales a y b $a, a \to \varepsilon$ y $b, b \to \varepsilon$. El siguiente gráfico reproduce esta situación:



Bibliografía

[1] Hopcroft J. E., Motwani R. and Ullman J. D. “Introduction to Automata Theory, Languages and Computation”. Addison–Wesley (2001).