Teoría de la Computación

APUNTES ELABORADOS POR

Jorge Eduardo Carrión Viramontes

Instituto Tecnológico de Morelia
Departamento de Sistemas y Computación
Av. Tecnológico # 1500
Col. Santiaguito
Teléfono (443) 312-15-70

EDITORIAL ACD

Catalogación

JORGE EDUARDO CARRIÓN
TEORIA DE LA COMPUTACIÓN
México: *ACD*, 2005
276 pág.; 21 cm.
ISBN: 968-5354-82-0

Editorial *ACD*
17 Sur No. 3105, Col. Volcanes
Tel: 01(222) 4 03 69 89 y telfax: 4 03 69 90
E–mail: informes@editorial–acd.com
Página Web: www.editorial–acd.com

Contenido

Capítulo

1

Conceptos Básicos

Se definen los conceptos fundamentales de símbolo, alfabeto, cadena y lenguaje. Se describen las operaciones y las propiedades más importantes de cada uno de ellos. Estos conceptos son esenciales para los objetivos de la presente obra.

Símbolo

Un símbolo es una representación de un concepto o idea que es perceptible por medio de al menos uno de los sentidos, generalmente la vista. Un símbolo puede ser un signo, un dígito, una letra e incluso un grupo de letras que se utiliza para transmitir el conocimiento en algún lenguaje y que tiene algún significado convencional. Se suele decir, entonces, que símbolo es una representación gráfica de algo.

Ejemplos de símbolos: **0**, **1**, **W**, **OK**, €, **b**, ψ, $\exists$, $\neq$, π, **STOP**, etc.

Alfabeto

Un alfabeto es un conjunto ordenado, finito y no vacío de símbolos.

Ejemplos

- $\sum$ = { **0**, **1** } es un alfabeto.
- El alfabeto griego es: $\sum = \{ \alpha, \beta, \gamma, \delta, \ldots, \omega \}$
- El alfabeto para un programa de cómputo puede ser: $\sum$ = { **APPEND, END**, **FOR**, **GET**, **IF**, … , **XOR** }

Propiedades de los Alfabetos

Los alfabetos permiten las operaciones que son comunes a cualquier otra clase de conjuntos, siempre que el resultado de tal operación no sea un conjunto vacío, es decir: Si $\sum_1$ y $\sum_2$ son dos alfabetos, entonces, los resultados de las siguientes operaciones: $\sum_1 \cup \sum_2$, es un alfabeto; $\sum_1 \cap \sum_2$, es un alfabeto si $\sum_1$ y $\sum_2$ no son disjuntos; $\sum_1 - \sum_2$, es un alfabeto si $\sum_1 \not\subset \sum_2$; $\sum_2 - \sum_1$, es un alfabeto si $\sum_2 \not\subset \sum_1$; y finalmente, $\sum_1 \oplus \sum_2$, es un alfabeto si $\sum_1 \neq \sum_2$.

Dado que no existe un alfabeto universal, porque tendría que ser infinito, y esto contradice la definición, tampoco puede existir el complemento de un alfabeto.

Ejemplo

Sean los alfabetos $\sum_1$ = { **0**, **1**, **2**, **3**, **4** } y $\sum_2$ = { **0**, **2**, **a**, **b** }, entonces tenemos que también son alfabetos los siguientes: $\sum_1 \cup \sum_2$ = { **0**, **1**, **2**, **3**, **4**, **a**, **b** }, $\sum_1 \cap \sum_2$ = { **0**, **2** }, $\sum_1 - \sum_2$ = { **1**, **3**, **4** }, $\sum_2 - \sum_1$ = { **a**, **b** } y $\sum_1 \oplus \sum_2$ = { **1**, **3**, **4**, **a**, **b** }.

Cadena

Una Cadena es una secuencia finita de símbolos de un alfabeto dado, yuxtapuestos uno a continuación de otro en una secuencia determinada. La posición que ocupa cada símbolo dentro de la cadena lo diferencia de las demás ocurrencias del mismo, y por eso cada símbolo puede aparecer numerosas veces dentro de una misma cadena.

Ejemplos

- Sea el alfabeto $\sum$ = { **0**, **1** }, entonces w_1 = **10**, w_2 = **10011** y w_3 = **100100101** son cadenas formadas a partir de ese alfabeto.
- Sea el alfabeto $\sum$ = { **a**, **b**, **c**, **d**, **e** }, x_1 = **bebe**, x_2 = **daba**, x_3 = **becada**, x_4 = **cabe** y x_5 = **c** son cadenas formadas a partir de ese alfabeto.

Cadena Vacía

La cadena vacía se denota por ε y es la cadena que está formada por una secuencia vacía de símbolos de cualquier alfabeto.

Operaciones con Cadenas

Longitud de una Cadena

Si *w* es una cadena, decimos que la longitud de la misma es el número de símbolos que la forman y se denota por |*w*|. No importando cuantas veces aparezca el mismo símbolo en la cadena, cada ocurrencia se cuenta por separado.

Ejemplos

- Sea w_1 = **10011**, entonces $|w_1| = 5$.
- Sea w_2 = **1011010101**, entonces $|w_2| = 10$.
- La longitud de la cadena vacía es cero: $|\varepsilon| = 0$.

Concatenación

Concatenación es la yuxtaposición de dos cadenas, una a continuación de la otra, de tal forma que si *w* y *x* son dos cadenas, la concatenación de *w* con *x* es la cadena que se obtiene de añadir la cadena *x* a la cadena *w*.

La concatenación se denota con el operador de yuxtaposición: $w \cdot x$, pero usualmente se omite el punto, quedando: wx. Además, la yuxtaposición no es conmutativa y en general se tiene que: $wx \neq xw$.

La longitud de la concatenación es igual a la suma de las longitudes de las cadenas individuales: $|wx| = |w| + |x| = |xw|$.

La concatenación de ε con cualquier cadena *w* no modifica a *w*. Es decir, la cadena vacía es el idéntico respecto a la concatenación, ya que: $\varepsilon w = w\varepsilon = w$. Por esto, a la cadena vacía se le conoce también como el *Elemento Neutro* de la Concatenación.

Ejemplos

- Sean *w* = **001** y *x* = **1**, entonces $w \cdot x$ = **0011**, mientras que $x \cdot w$ = **1001**.
- Sean *w* = **ab** y *x* = **bab**, entonces *xw* = **babab**, *wx* = **abbab**.
- Sea *w* = **abba**, entonces εw = **abba** y $w\varepsilon$ = **abba**.

Potencia de Cadenas

Sea *w* una cadena formada a partir de un alfabeto $\sum$, entonces para cualquier $n \geq 0$, se tiene que la enésima potencia de *w* se puede definir recursivamente como:

$$w^n = \begin{cases} \varepsilon & \text{para } n = 0 \\ w\,w^{n-1} & \text{para } n > 0 \end{cases}$$

Observe que en general se tiene que la concatenación de potencias es diferente de la potencia de la concatenación, esto es: $(wx)^n \neq w^n x^n$.

Ejemplos

- Sea *w* = **abc** entonces $w^0 = \varepsilon$, $w^1 = ww^0$ = **abc**ε = **abc**, $w^2 = ww^1$ = **abcabc**, $w^3 = ww^2$ = **abcabcabc**, etc.
- Sea *w* = **0** entonces $w^0 = \varepsilon$, w^1 = **0**, w^2 = **00**, w^3 = **000**, w^4 = **0000**, etc.
- Sea $w = \varepsilon$, entonces $w^0 = w^1 = w^2 = w^3 = \ldots = \varepsilon$.
- Sea *w* = **01** y *x* = **1**, entonces $(wx)^2$ = (**011**)^2 = **011011**, y w^2x^2 = (**01**)^2(**1**)^2 = **010111**.

Prefijo

Sea *w* una cadena formada a partir de un alfabeto $\sum$, entonces, se cumple que existen dos cadenas *x* y *z*, tales que *w* = *xz*, se dice que x es un prefijo de *w*. Además, si $z \neq \varepsilon$ (es decir $x \neq w$), se dice que *x* es un prefijo propio de *w*; en el otro caso, cuando *x* = *w*, se tiene que *x* es llamado prefijo impropio.

Una cadena de longitud n, tiene n prefijos propios distintos y uno impropio.

Ejemplos

- $x_0 = \varepsilon$, $x_1 =$ **c**, $x_2 =$ **co**, $x_3 =$ **cof**, $x_4 =$ **cofr** son prefijos propios de *w* = **cofre**.
- $x_0 = \varepsilon$, $x_1 =$ **a**, $x_2 =$ $\mathbf{a}^2$, $x_3 =$ $\mathbf{a}^3$, $x_4 =$ $\mathbf{a}^4$ son prefijos propios de *w* = $\mathbf{a}^5$.

Sufijo

Sea *w* una cadena formada a partir de un alfabeto $\sum$, entonces, se cumple que existen dos cadenas *x* y *z*, tales que *w* = *xz*, se dice que z es un sufijo de *w*. Además, si $x \neq \varepsilon$ (es decir $z \neq w$), se puede decir que *z* es un sufijo propio de *w*; y similarmente al caso anterior, si *z* = *w*, entonces *z* es un sufijo impropio.

Una cadena de longitud n, tiene n sufijos propios distintos y uno impropio.

Ejemplos

- $z_0 = \varepsilon$, $z_1 =$ **o**, $z_2 =$ **to** y $z_3 =$ **ato** son los sufijos propios de *w* = **gato**, mientras que el sufijo impropio es $z_4 =$ **gato**.
- $z_0 = \varepsilon$, $z_1 =$ **a**, $z_2 =$ **aa**, $z_3 =$ **aaa** y $z_4 =$ **aaaa** son los sufijos propios de *w* = $\mathbf{a}^5$.

Como se ha podido observar en los ejemplos anteriores, la cadena vacía ε siempre es prefijo y sufijo propio de cualquier otra cadena.

Subcadenas

Una cadena *y* es una subcadena de otra cadena *w*, si existen *x* y *z*, no ambas vacías, para las cuales se cumple que *w* = *xyz*. Cualquier prefijo o sufijo propios de *w* también son subcadenas de *w*, en particular, ε es subcadena de cualquier otra cadena.

La cantidad máxima N de subcadenas distintas de una cadena dada se puede determinar con la siguiente fórmula:

$$N = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

Mientras que la cantidad mínima de subcadenas distintas de una cadena es n, para el caso en que la cadena esté formada por puros símbolos iguales.

Ejemplos

- $y_0 = \varepsilon$, $y_1 = \mathbf{g}$, $y_2 = \mathbf{a}$, $y_3 = \mathbf{t}$, $y_4 = \mathbf{o}$, $y_5 = \mathbf{ga}$, $y_6 = \mathbf{at}$, $y_7 = \mathbf{to}$, $y_8 = \mathbf{gat}$ y $y_9 = \mathbf{ato}$ son las 10 subcadenas de $w = \mathbf{gato}$.
- $y_0 = \varepsilon$, $y_1 = \mathbf{p}$, $y_2 = \mathbf{a}$, $y_3 = \mathbf{pa}$, $y_4 = \mathbf{ap}$, $y_5 = \mathbf{pap}$ y $y_6 = \mathbf{apa}$ son las 7 subcadenas distintas de $w = \mathbf{papa}$.
- $y_0 = \varepsilon$, $y_1 = \mathbf{a}$, $y_2 = \mathbf{aa}$, $y_3 = \mathbf{aaa}$, $y_4 = \mathbf{aaaa}$ y $y_5 = \mathbf{aaaaa}$ son las 6 subcadenas distintas de $w = \mathbf{aaaaaa}$.

Inversa de una Cadena

La Inversa de una Cadena *w* es la cadena w^R, tal que es la imagen refleja de *w*, es decir, que equivale a *w* cuando se lee de derecha a izquierda.

Formalmente se define la inversa de *w* de manera recursiva como:

$$w^R = \begin{cases} \varepsilon & \text{si} \quad w = \varepsilon \\ y^R\mathbf{a} & \text{si} \quad w = \mathbf{a}y \end{cases}$$

Donde *y* es una cadena y **a** es un símbolo.

Propiedades de la Inversa

- $\mathbf{a}^R = \mathbf{a}$, donde **a** es un símbolo.
- $(x^R)^R = x$
- Si $x = wy$, entonces $x^R = y^R w^R$
- Una cadena se llama *palíndroma*, cuando es igual a su inversa: $w = w^R$.

Ejemplo

Obtener la inversa de la cadena $w = \mathbf{amor}$, aplicando la definición podemos considerar que $w = \mathbf{a}(\mathbf{mor})$, entonces tenemos que $w^R = (\mathbf{amor})^R = (\mathbf{mor})^R\mathbf{a}$, repitiendo el proceso, se obtiene sucesivamente:

$w^R = (\mathbf{m}(\mathbf{or}))^R\mathbf{a} = ((\mathbf{or})^R\mathbf{m})\mathbf{a} = (\mathbf{o}(\mathbf{r}))^R\mathbf{ma} = (\mathbf{r})^R\mathbf{oma} = (\mathbf{r}\varepsilon)^R\mathbf{oma} = (\varepsilon)^R\mathbf{roma} = \mathbf{roma}$.

Obviamente, para nosotros, es mucho más sencillo en la práctica hacerlo de forma directa que a través de la definición, pero la fórmula recursiva es muy útil para una implementación en un programa de computadora.

Lenguaje

Un lenguaje formal es un conjunto de palabras o cadenas formadas a partir de los símbolos de un alfabeto dado. Un lenguaje puede ser finito o infinito, aunque, como se mencionó antes, el alfabeto de donde se genera debe ser siempre finito.

Todo alfabeto $\sum$ puede ser considerado también, si así se desea, como un lenguaje formado por cadenas que son todas de longitud uno.

Ejemplos

- Sea $\sum$ = { **a**, **b**, **c** }, entonces L = { **a**, **b**, **c** } es un lenguaje finito sobre $\sum$.
- Sea $\sum$ = { **a**, **b** }, entonces L = { ε, **ab**, **abbab**, **abbba**, **bbaa**, **baaba** } es un lenguaje finito sobre $\sum$.
- Sea $\sum$ = { **a** }, entonces L = { ε, **a**, **aa**, **aaa**, **aaaa**, … } es un lenguaje infinito sobre $\sum$ y también se denota en forma compacta como: L = { $\mathbf{a}^n \mid n \geq 0$}.
- Sea $\sum$ = { **0**, **1** }, entonces L = { ε, **0**, **1**, **00**, **11**, **000**, **010**, **101**, **111**, … } es un lenguaje infinito formado por todas las cadenas *palíndromas* de ceros y unos, o sea las cadenas que cumplen que $w = w^R$, esto se denota como: L = { $w \mid w = w^R$ }

Lenguaje Vacío

El lenguaje vacío se denota como $\varnothing$, y es aquél que no contiene ninguna cadena, es decir $\varnothing$ = { }, igual que en teoría de Conjuntos. No se debe confundir el lenguaje vacío con el lenguaje que contiene solamente a la cadena vacía, es decir $\varnothing \neq \{\varepsilon\}$.

Ejemplo

- Sea $\sum$ = { **a**, **b** }, entonces L = { } = $\varnothing$ es un lenguaje vacío sobre $\sum$.

Lenguaje Universal

Definimos al Lenguaje Universal $\sum^*$ como el lenguaje formado por todas las cadenas que se pueden formar a partir del alfabeto $\sum$; a $\sum^*$ también se le conoce como la cerradura de $\sum$. Para cualquier alfabeto $\sum$, el lenguaje universal $\sum^*$ es siempre infinito.

Ejemplos

- El lenguaje L = { $\mathbf{a}^n \mid n \geq 0$ } es el lenguaje universal del alfabeto $\sum$ = { **a** }.
- Sea $\sum^*$ = { ε, **0**, **1**, **00**, **01**, **10**, **11**, **000**, **001**, **010**, **011**, **100**, **101**, **110**, **111**, … }, es el Lenguaje Universal para el alfabeto $\sum$ = { **0**, **1** }.

El conjunto $\sum^*$ es un conjunto numerable, lo que significa que los elementos de $\sum^*$ pueden ordenarse por tamaño y luego, para cadenas del mismo tamaño, ordenarse alfabéticamente (recuérdese que $\sum$ es un conjunto ordenado), para finalmente poder establecer una relación biunívoca de $\sum^*$ con el conjunto de números Naturales.

Ejemplo

- Dado $\sum^*$ = { **0**, **1** }* = { ε, **0**, **1**, **00**, **01**, **10**, **11**, **000**, … }, se puede establecer la siguiente correspondencia biunívoca entre $\sum^*$ y el conjunto de los números naturales: 1 $\leftrightarrow$ ε, 2 $\leftrightarrow$ **0**, 3 $\leftrightarrow$ **1**, 4 $\leftrightarrow$ **00**, 5 $\leftrightarrow$ **01**, 6 $\leftrightarrow$ **10**, 7 $\leftrightarrow$ **11**, 8 $\leftrightarrow$ **000**, etc.

Para un alfabeto $\sum$ que contiene n símbolos, se tiene que $\sum^*$ tiene $n^0 = 1$, cadenas de longitud 0, $n^1 = n$ cadenas de longitud 1, n^2 cadenas de longitud 2, etc. Por lo tanto, es posible determinar el lugar que ocupa determinada cadena dentro del conjunto $\sum^*$.

Ejemplo

- Para el alfabeto $\sum$ = { **0**, **1** }, se tiene que $\sum^*$ contiene 1 cadena de longitud 0, 2 cadenas de longitud 1, 4 cadenas de longitud 2, 8 cadenas de longitud 3, etc. Se puede verificar que la cadena **001** ocupa el noveno lugar del conjunto.
- Sea el alfabeto $\sum$ = { **a**, **b**, **c** }, entonces en $\sum^*$ existe 1 cadena de longitud 0, 3 cadenas de longitud 1, 9 cadenas de longitud 2, 27 cadenas de longitud 3, 81 cadenas de longitud 4, etc. y además, se puede verificar que la cadena **aaaa** ocupa el lugar 41 del conjunto.

Operaciones con Lenguajes

Concatenación de Lenguajes

Sean L_1 y L_2 dos lenguajes cualesquiera, entonces $L_1 \cdot L_2 = \{ w \cdot x \mid w \in L_1, x \in L_2 \}$ es el lenguaje concatenación de L_1 con L_2.

Ejemplos

- Sean los lenguajes L_1 = { **patz**, **came**, **yure**, **cara**, **zita** } y L_2 = { **cuaro** }, entonces la concatenación de L_1 con L_2 es: $L_1 \cdot L_2$ = { **patzcuaro**, **camecuaro**, **yurecuaro**, **caracuaro**, **zitacuaro** }.
- Sean los lenguajes L_1 = { **01**, **21** } y L_2 = { **a**, **ba**, **ca** }, entonces la concatenación de L_1 con L_2 es: $L_1 \cdot L_2$ = { **01a**, **01ba**, **01ca**, **21a**, **21ba**, **21ca**}
- Similarmente se puede obtener $L_2 \cdot L_1$ = { **a01**, **ba01**, **ca01**, **a21**, **ba21**, **ca21**}

En este último ejemplo el alfabeto de L_1 es $\sum_1$ = { **0**, **1**, **2** }, mientras que el alfabeto de L_2 es $\sum_2$ = { **a**, **b**, **c** }, por lo tanto, el alfabeto de $L_1 \cdot L_2$ es la unión de ambos alfabetos: $\sum_1 \cup \sum_2$ = { **0**, **1**, **2**, **a**, **b**, **c** }.

El lenguaje {ε} es idéntico respecto a la concatenación: $L_1 \cdot \{\varepsilon\} = \{\varepsilon\} \cdot L_1 = L_1$.

El lenguaje ∅ es nulo respecto a la concatenación: $L_1 \cdot \varnothing = \varnothing \cdot L_1 = \varnothing$.

La cantidad máxima de cadenas que hay en el lenguaje concatenación es el producto de las cantidades de cadenas en cada uno de ellos, en el segundo ejemplo, se tiene que los lenguajes L_1 y L_2 tienen 2 y 3 cadenas respectivamente, entonces, por lo tanto: $L_1 \cdot L_2$, como $L_2 \cdot L_1$ tienen 6 cadenas distintas, sin embargo en algunos casos puede haber menos cadenas distintas, debido a la generación de cadenas duplicadas.

Potencia de Lenguajes

Sea L un lenguaje sobre el alfabeto ∑, entonces para cualquier n ≥ 0, se tiene que la enésima potencia de L se define recursivamente como:

$$L^n = \begin{cases} \{\varepsilon\} & \textbf{para } n = 0 \\ L\,L^{n-1} & \textbf{para } n > 0 \end{cases}$$

Ejemplos

- Sea L = { **ab** }, entonces L^0 = { ε }, L^1 = { **ab** }, L^2 = { **abab** } y L^3 = { **ababab** }.
- Sea L = { **0**, **1** } entonces L^0 = { ε }, L^1 = { **0**, **1** }, L^2 = { **00**, **01**, **10**, **11** }, etc.
- Sea L = { ε, **a** }, entonces L^0 = { ε }, L^1 = { ε, **a** }, L^2 = { ε, **a**, **aa** }, etc.

Es interesante observar que de la definición se desprende que: $\varnothing^0$ = { ε }, mientras que $\varnothing^n = \varnothing$ para toda n > 0.

Unión, Intersección y Diferencia

Las demás operaciones de conjuntos se aplican igualmente a los lenguajes, es decir: Si L_1 y L_2 son dos lenguajes cualesquiera, entonces $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$, $L_2 - L_1$ y $L_1 \oplus L_2$ también son lenguajes.

Adicionalmente, dado que existe el lenguaje universal ∑*, a también podemos definir a L^C, el complemento de L, como $L^C = \sum^* - L$.

Sublenguaje

A es un sublenguaje de L y se denota como $A \subseteq L$, si para cada $w \in A$ se tiene que también $w \in L$.

En el ejemplo anterior se puede observar una importante propiedad; si un lenguaje contiene a la cadena vacía ε, se cumple que: $L^0 \subseteq L^1 \subseteq L^2 \subseteq L^3 \subseteq \ldots$

Cualquier lenguaje L sobre el alfabeto $\sum$ es un sublenguaje de $\sum^*$.

$\varnothing$ es un sublenguaje de cualquier lenguaje L sin importar el alfabeto $\sum$.

Propiedades de las operaciones de Lenguajes

Sean A, B y C tres lenguajes cualesquiera sobre un alfabeto $\sum$, entonces:

- $A (B \cup C) = AB \cup AC$
- $(B \cup C) A = BA \cup CA$
- $A = B$ sí y sólo sí $A \subseteq B$ y $B \subseteq A$
- en general $A^C B^C \neq (AB)^C$
- en general $A (B \cap C) \neq AB \cap AC$
- en general $A (B - C) \neq AB - AC$

Ejemplos

- Si $A = \{ \varepsilon, \mathbf{0}, \mathbf{1} \}$, $B = \{ \varepsilon \}$ y $C = \{ \mathbf{0} \}$, entonces tenemos: $A(B \cap C) = \varnothing$, mientras que $AB \cap AC = \{ \mathbf{0} \}$.
- Por otro lado, $A(B - C) = \{ \varepsilon, \mathbf{0}, \mathbf{1} \}$, mientras que $AB - AC = \{ \varepsilon, \mathbf{1} \}$

Cerradura de Kleene

Sea L un lenguaje sobre el alfabeto $\sum$, se define a L* como la Cerradura de Kleene o Cerradura Estrella como la unión infinita de todas las potencias de L, incluyendo la potencia cero.

$$L^* = \bigcup_{n=0}^{\infty} L^n$$

Ejemplos

- $L_1 = \{ \mathbf{0}, \mathbf{1} \}$, entonces $L_1^* = L_1^0 \cup L_1^1 \cup L_1^2 \cup \ldots = \{ \varepsilon \} \cup \{ \mathbf{0}, \mathbf{1} \} \cup \{ \mathbf{00}, \mathbf{01}, \mathbf{10}, \mathbf{11} \} \cup \{ \mathbf{000}, \mathbf{001}, \mathbf{010}, \mathbf{011}, \ldots \} \cup \ldots = \{ \varepsilon, \mathbf{0}, \mathbf{1}, \mathbf{00}, \mathbf{01}, \mathbf{10}, \mathbf{11}, \mathbf{000}, \mathbf{001}, \mathbf{010}, \mathbf{011}, \ldots \}$. Obsérvese que si $\sum = \{ \mathbf{0}, \mathbf{1} \}$, entonces $\sum^* = L_1^*$.

- Sea L_2 = { **01**, **10** }, entonces L_2^*= { ε, **01**, **10**, **0101**, **0110**, **1001**, **1010**, **010101**, … }, en este caso $L_2^* \subseteq \Sigma^*$, para Σ = { **0**, **1** },

Los lenguajes L_1 = { ε } y L_2 = ∅, son los únicos cuya cerradura de Kleene es finita, y dado que $\varnothing^0$ = { ε }, se cumple que ∅* = { ε }, y por tanto, para ambos se tiene que: $L_1^* = L_2^*$ = { ε }. También es interesante observar que aunque tengamos que $L_1^* = L_2^*$, esto no significa que L_1 y L_2 tengan que ser iguales.

Cerradura de Positiva

De manera similar, se define la Cerradura Positiva de L, como la unión infinita de todas las potencias de L a partir de uno.

$$L^+ = \bigcup_{n=1}^{\infty} L^n$$

Ejemplo

- Sea L = {**ab**}, entonces L^+ = {**ab**, **abab**, **ababab**, **abababab**, **ababababab**, …}, mientras que L* = { ε, **ab**, **abab**, **ababab**, **abababab**, **ababababab**, …}.

Propiedades de las Cerraduras

Para cualquier lenguaje L, se cumplen las siguientes propiedades:

- $L^* = \{ \varepsilon \} \cup L^+$
- $L^+ = L \cdot L^* = L^* \cdot L = L^* \cdot L^+ = L^+ \cdot L^*$
- $(L^+)^+ = L^+$
- $(L^*)^* = (L^+)^* = (L^*)^+ = L^*$

Si $\varepsilon \in L$, entonces se sigue necesariamente que: $L^+ = L^*$, mientras que si $\varepsilon \notin L$, se cumple forzosamente que: $L^+ = L^* - \{\varepsilon\}$.

Inverso de un Lenguaje

Sea L un lenguaje, se define al inverso de L como $L^R = \{ w^R \mid w \in L \}$.

Ejemplos

- Sea L = { arroz, abad, radar, lamina }, entonces L^R = { zorra, daba, radar, animal }
- Sea L = { **0**, **1**, **101**, **11101** }, entonces L^R = { **0**, **1**, **101**, **10111** }
- Sea L = { ε, **0**, **1**, **00**, **11**, **000**, **010**, **101, 111** }, entonces L^R = L.

Propiedades del Lenguaje Inverso

Sean A y B dos lenguajes, entonces se cumple que:

- $\Sigma^R = \Sigma$
- $(\Sigma^*)^R = \Sigma^*$
- $(A^R)^R = A$
- $(A^*)^R = (A^R)^*$
- $(A \cdot B)^R = B^R \cdot A^R$
- $(A \cup B)^R = A^R \cup B^R$
- $(A \cap B)^R = A^R \cap B^R$

Del hecho de que se cumpla $L = L^R$, no se implica que L esté formado por cadenas palíndromas exclusivamente, un ejemplo de esto sería el lenguaje L = { **01**, **10** }.

Preguntas

a) ¿Cuántas posibles subcadenas tiene una cadena de longitud n?

b) ¿Cuál es el número mínimo de subcadenas distintas de una cadena de longitud n? ¿Cómo deberá ser esa cadena?

c) ¿Bajo qué condiciones se cumple que $L^* = L^+$?

d) ¿Y en que otras condiciones se cumple que $L^+ = L^* - \{ \varepsilon \}$?

e) ¿En que casos el Lenguaje Universal es finito?

f) Si L es un lenguaje finito, ¿Cómo debe ser L^C?

g) ¿Y si L es infinito que podemos decir de L^C?

h) ¿Existe algún lenguaje para el que se cumple que L* es finito?

i) ¿Se cumple que $(L^*)^n = (L^n)^*$?

Ejercicios Capítulo 1

1.1 Sean los alfabetos A = { ψ, η, λ }, y B = { ϕ, λ, θ }, obtener los siguientes alfabetos, si existen: $A \cup B$, $A \cap B$, $A \oplus B$, $A - B$ y $B - A$.

1.2 Sea *w* = **pino**, obtener todos los prefijos y sufijos propios y todas las subcadenas de *w*.

1.3 Encontrar w^2, w^3 y w^R para la cadena *w* = **papa**.

1.4 Sea *x* = **piñata**, obtener todos los prefijos de *x*.

1.5 Sea *y* = **maroma**, obtener todos los sufijos de *y*.

1.6 Obtener todas las subcadenas de z = **banana**.

1.7 Dadas las cadenas anteriores obtener: xy^Rz, y también: z^2x.

1.8 Sea la cadena w = **01110220**, obtener todas las subcadenas distintas de w de longitud menor o igual a 3.

1.9 Sean los lenguajes A = { **a**, **b**, **c** } y B = { **c**, **d**, **e** }, efectuar las siguientes operaciones de lenguajes: $(A \cup B^2)$, $(AB)^*$ y $(BA)^R$.

1.10 Sean los lenguajes L_1 = { ε, **0**, **10**, **11** } y L_2 = { ε, **1**, **01**, **11** } sobre el alfabeto $\sum$ = { **0**, **1** }, obtener: $L_1 \cdot L_2$, $L_2 \cdot L_1$, $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$, $L_2 - L_1$, L_1^*, L_2^* y $L_1 \oplus L_2$.

1.11 Sea L = { ε, **a** }, obtener L^0, L^1, L^2 y L^3.

1.12 Sean L_1 = { **a** } y L_2 = { **b** }, explique como se interpretan los siguientes lenguajes: $L_1^n L_2$, $L_1 L_2^n$ y $(L_1L_2)^n$.

1.13 Sean L_1 = { ε }, L_2 = { **aa**, **ab**, **bb** }, L_3 = { ε, **aa**, **ab** } y $L_4 = \varnothing$, obtener los lenguajes: $L_1 \cup L_2$, $L_1 \cup L_3$, $L_1 \cup L_4$, $L_2 \cup L_4$, $L_1 \cap L_2$, $L_2 \cap L_3$, $L_3 \cap L_4$ y $L_1 \cap L_4$.

1.14 Dados los lenguajes siguientes: A = { **ab**, **b**, **cb** } y B = { **a**, **ba** } obtener los lenguajes que resultan de las operaciones de lenguajes: $(A \cup B^2)$, $(B \cup A)^R$, (AB), $(A^2 \cap BA)$, $(A \oplus B^R)$ y $(A^R - B)^2$.

1.15 Dados los lenguajes: L_1 = { **01**, **11** } y L_2 = { **011**, **101**, **11** } obtener los lenguajes que resultan de las operaciones: $(L_1 \cup L_2)^R$, $(L_2 - L_1)^2$, $(L_1 - L_2)^+$, $(L_1 \cap L_2)^*$, $L_1^R L_2$.

1.16 Sea L = { **a**, **ba** }*, sobre $\sum$ = { **a**, **b** }, obtenga L^C.

Capítulo

2

Lenguajes Regulares

Se define qué es un lenguaje regular y qué son las expresiones regulares, se enuncian los principales teoremas que involucran expresiones regulares y se ilustra su uso en la simplificación de expresiones regulares.

Los Lenguajes Regulares son la primera clase de lenguajes que vamos a considerar, debido a su simplicidad, aunado a las propiedades de cerradura respecto a las principales operaciones de lenguajes, que son unión, concatenación y cerradura de Kleene, y es con base en estas operaciones que daremos la definición:

Definición

Sea $\sum$ un alfabeto, el conjunto de lenguajes regulares sobre $\sum$ se define como sigue:

- $\varnothing$ es un Lenguaje Regular.
- $\{ \varepsilon \}$ es un Lenguaje Regular
- Para toda $\mathbf{a} \in \sum$, { **a** } es un Lenguaje Regular
- Si L_1 y L_2 son dos Lenguajes Regulares, entonces $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^* y L_2^* son todos Lenguajes Regulares
- Ningún otro lenguaje sobre $\sum$ es Regular.

Ejemplo 1

Sea $\sum$ = { **a**, **b** }, entonces de la definición se tiene que:

- $\varnothing$ y $\{ \varepsilon \}$ son Regulares
- { **a** } y { **b** } son Regulares
- { **a**, **b** } es Regular (Unión)
- { **aa** }, { **ab** }, { **bb** } y { **ba** } Son Regulares (Concatenación)
- { **a**, **b**, **ab**, **ba**, **aa**, **bb** } es Regular (Unión)
- $\{ \mathbf{a}^n \mid n \geq 0 \}$ es Regular (Cerradura de Kleene)
- $\{ (\mathbf{ab})^n \mid n \geq 0 \}$ es Regular (Cerradura de Kleene)

- $\{ \mathbf{a}^n \mathbf{b}^m \mid n \geq 0, m \geq 0 \}$ es Regular (Concatenación)
- $\{ (\mathbf{ab})^n \mid n \geq 0 \} \cup \{ (\mathbf{ba})^m \mid m \geq 0 \}$ es Regular (Unión)

Ejemplo 2

Sea $\sum = \{ \mathbf{0}, \mathbf{1}, \mathbf{2} \}$, entonces de la definición se tiene que:

- $\varnothing$ y $\{ \varepsilon \}$ son Regulares
- $\{ \mathbf{0} \}$, $\{ \mathbf{1} \}$ y $\{ \mathbf{2} \}$ son Regulares
- $\{ \mathbf{0}, \mathbf{1} \}$, $\{ \mathbf{0}, \mathbf{2} \}$, $\{ \mathbf{1}, \mathbf{2} \}$, $\{ \mathbf{0}, \mathbf{1}, \mathbf{2} \}$ son Regulares (Unión)
- $\{ \mathbf{01} \}$, $\{ \mathbf{12} \}$, $\{ \mathbf{02} \}$ son Regulares (Concatenación)
- $\{ \mathbf{00}, \mathbf{01} \}$, $\{ \mathbf{10}, \mathbf{12} \}$, $\{ \mathbf{00}, \mathbf{01}, \mathbf{02}, \mathbf{10}, \mathbf{11}, \mathbf{12} \}$ son Regulares (Concatenación)
- $\{ \mathbf{00}, \mathbf{01}, \mathbf{02}, \mathbf{10}, \mathbf{11}, \mathbf{12}, \mathbf{20}, \mathbf{21}, \mathbf{22} \}$ es Regular (Unión)
- $\{ \varepsilon, \mathbf{0}, \mathbf{00}, \mathbf{000}, \mathbf{0000}, \mathbf{00000}, \ldots \}$ es Regular (Cerradura de Kleene)
- $\{ \varepsilon, \mathbf{01}, \mathbf{0101}, \mathbf{010101}, \mathbf{01010101}, \ldots \}$ es Regular (Cerradura de Kleene)
- $\{ \varepsilon, \mathbf{0}, \mathbf{1}, \mathbf{00}, \mathbf{01}, \mathbf{11}, \mathbf{000}, \mathbf{001}, \mathbf{011}, \mathbf{111}, \ldots \}$ es Regular (Concatenación)

Teoremas

- Todos los lenguajes finitos son regulares.
- Si L es regular, entonces $L^C = \sum^* - L$ también es regular.
  Este teorema se demostrará posteriormente en el siguiente capítulo.
- Si L_1 y L_2 son regulares, entonces $L_1 \cap L_2$ también es un lenguaje regular.
  Demostración: Si L_1 y L_2 son regulares, entonces L_1^C y L_2^C son regulares, entonces $L_1^C \cup L_2^C$ es regular y por tanto $(L_1^C \cup L_2^C)^C = L_1 \cap L_2$ es regular.
- Si L_1 y L_2 son regulares, entonces $L_1 - L_2$ también es un lenguaje regular.
  Demostración: Si L_1 y L_2 son regulares, entonces $L_1 \cap L_2^C$ es regular, por los teoremas anteriores y resulta que: $L_1 - L_2 = L_1 \cap L_2^C$, entonces es regular.
- Si L_1 es regular, entonces L_1^R también es un lenguaje regular.
  La demostración se desprende de las propiedades de la inversa de una cadena que se extienden a los lenguajes.

Todas las cadenas de longitud par, formadas a partir de un alfabeto $\sum$, se representan por el lenguaje $(\sum^2)^*$, mientras que las cadenas de longitud impar se representan por $\sum(\sum^2)^*$, por lo tanto: $\sum^* = \sum(\sum^2)^* \cup (\sum^2)^*$

Expresiones Regulares

Podemos simplificar la especificación de un lenguaje regular utilizando una nomenclatura abreviada, llamada *expresión regular*, de tal manera que el lenguaje unitario { **a** }, se denota simplemente como **a**.

Las operaciones de lenguajes regulares se denotan como: **a** ∪ **b**, en vez de { **a**, **b** }; **ab**, en vez de { **ab** }; **a*** en vez de { **a** }* y $\mathbf{a}^+$ en vez de { **a** }$^+$. El objetivo de esto es facilitar la lectura y manipulación algebraica de los lenguajes regulares.

Entonces podemos definir las expresiones regulares recursivamente:

- ∅ y ε son expresiones regulares.
- **a** es una expresión regular para toda **a** ∈ ∑,
- Si **r** y **s** son dos expresiones regulares cualesquiera, entonces **r** ∪ **s**, **r** · **s**, **r*** y **s*** son también expresiones regulares
- Ninguna otra secuencia de símbolos es una expresión regular.

El orden de precedencia de las operaciones en una expresión regular es análogo a las expresiones algebraicas comunes: cerradura (potencia), concatenación y unión, a menos de que se utilicen paréntesis para cambiar ese orden.

Ejemplos

- Determinar las cadenas que pertenecen al lenguaje descrito por la expresión regular: **a***__b__ ∪ **c**, realizando las operaciones en el orden de precedencia, tenemos:
  L = { **b**, **ab**, **aab**, **aaab**, **aaaab**, ... , **c** }

- Determinar como son las cadenas que pertenecen al lenguaje dado por la expresión regular: **c*** (**a** ∪ **bc***)*, sobre el alfabeto ∑ = { **a**, **b**, **c** }.
  Analizando las cadenas que se pueden construir, es posible observar que ninguna de ellas puede contener a la subcadena **ac**.
  L = { ε, **a** , **aa**, **ab**, **abc**, **aab**, **abca**, ..., **b**, **bc**, **bcc**, **bcca**, **bcbc**, ..., **c**, **ca**, **cab**, **cabc**, **caab**, ..., **cc**, **cca**, **ccb**, … }

- Determinar las cadenas que pertenecen al lenguaje descrito por la expresión regular siguiente: L = **c***__a__ ∪ (**bc**)* ∪ **b***.
  Analizando separadamente cada término, obtenemos que:
  L = { **a**, **ca**, **cca**, **ccca**, **cccca**, ..., ε, **bc**, **bcbc**, **bcbcbc**, **bcbcbcbc**, ..., **b**, **bb**, **bbb**, **bbbb**, **bbbbb**, ... }

En ocasiones se requiere que seamos capaces de encontrar una expresión regular que represente a un determinado lenguaje, dada la descripción de las características que cumplen las cadenas pertenecientes a él, como se muestra en los siguientes ejemplos:

Ejemplo 1

Encontrar la expresión regular que representa al lenguaje formado por las cadenas que contienen exactamente dos ceros sobre el alfabeto: $\sum$ = { **0**, **1** }.

Es recomendable que primero describamos este lenguaje por extensión, es decir, listemos algunas de las cadenas que lo forman: L = { **00**, **001**, **010**, **100**, **0011**, **0101**, **0110**, **1001**, **1010**, **1100**, … } una vez que hemos comprendido como son estas cadenas, es relativamente fácil encontrar la expresión regular: L = **1*01*01***, ya que puede haber cualquier cantidad de unos al inicio, luego el primer cero, otra cantidad de unos, el segundo cero y al final cualquier cantidad de unos.

Ejemplo 2

Encontrar la expresión regular que representa al lenguaje formado por las cadenas que contienen cuando mucho una **a**, sobre el alfabeto: $\sum$ = { **a**, **b** }.

Como en el ejemplo anterior, listemos varias de las cadenas que forman al referido lenguaje: L = { ε, **a**, **b**, **ab**, **ba**, **bb**, **abb**, **bab**, **bba**, **bbb**, **abbb**, … } una vez que hemos comprendido como son estas cadenas, es relativamente fácil encontrar la expresión regular: L = **b*** $\cup$ **b*ab***, ya que puede no haber ninguna **a**, o bien, puede haber una, y en este segundo caso, al inicio puede existir cualquier cantidad de **b**s, luego la **a**, y al final cualquier cantidad de **b**s.

Ejemplo 3

Encontrar la expresión regular que representa al lenguaje formado por las cadenas que contienen el sufijo **aba**, sobre el alfabeto: $\sum$ = { **a**, **b** }.

Podemos listar las cadenas que forman este lenguaje: L = { **aba**, **aaba**, **baba**, **aaaba**, **ababa**, **baaba**, **bbaba**, … }, es fácil encontrar la expresión regular si consideramos que antes del sufijo puede haber cualquier cadena del alfabeto $\sum$, es decir, el prefijo es de la forma $\sum$*, y por lo tanto, la expresión regular buscada es: L = (**a** $\cup$ **b**)***aba**.

Teoremas sobre Expresiones Regulares

Sean **r**, **s** y **t** expresiones regulares sobre un alfabeto $\sum$, entonces:

1. $\mathbf{r} \cup \mathbf{s} = \mathbf{s} \cup \mathbf{r}$
2. $\mathbf{r} \cup \varnothing = \varnothing \cup \mathbf{r} = \mathbf{r}$
3. $\mathbf{r} \cup \mathbf{r} = \mathbf{r}$
4. $(\mathbf{r} \cup \mathbf{s}) \cup \mathbf{t} = \mathbf{r} \cup (\mathbf{s} \cup \mathbf{t})$
5. $\mathbf{r}\varepsilon = \varepsilon\mathbf{r} = \mathbf{r}$
6. $\mathbf{r}\varnothing = \varnothing\mathbf{r} = \varnothing$
7. $(\mathbf{rs})\mathbf{t} = \mathbf{r}(\mathbf{st})$
8. $\mathbf{r} (\mathbf{s} \cup \mathbf{t}) = \mathbf{rs} \cup \mathbf{rt}$
9. $(\mathbf{r} \cup \mathbf{s}) \mathbf{t} = \mathbf{rt} \cup \mathbf{st}$
10. $\mathbf{r}^* = (\mathbf{r}^*)^* = (\mathbf{r}^+)^* = (\mathbf{r}^*)^+ = \mathbf{r}^*\mathbf{r}^* = (\varepsilon \cup \mathbf{r})^* = (\varepsilon \cup \mathbf{r})^+ = \varepsilon \cup \mathbf{r}^* = \varepsilon \cup \mathbf{r}^+ = \mathbf{r}^* (\varepsilon \cup \mathbf{r}) = (\varepsilon \cup \mathbf{r}) \mathbf{r}^* = \mathbf{r}^* \cup \mathbf{r} = \mathbf{r}(\mathbf{rr})^* \cup (\mathbf{rr})^*$
11. $(\mathbf{r} \cup \mathbf{s})^* = (\mathbf{r}^* \cup \mathbf{s}^*)^* = (\mathbf{r}^*\mathbf{s}^*)^* = (\mathbf{r}^*\mathbf{s})^* \mathbf{r}^* = \mathbf{r}^* (\mathbf{s}\mathbf{r}^*)^*$
12. $\mathbf{r}(\mathbf{sr})^* = (\mathbf{rs})^* \mathbf{r}$
13. $(\mathbf{r}^*\mathbf{s})^* = \varepsilon \cup (\mathbf{r} \cup \mathbf{s})^*\mathbf{s}$
14. $(\mathbf{r}\mathbf{s}^*)^* = \varepsilon \cup \mathbf{r} (\mathbf{r} \cup \mathbf{s})^*$
15. $\mathbf{r}^+ = \mathbf{r}\mathbf{r}^* = \mathbf{r}^*\mathbf{r} = \mathbf{r}^+\mathbf{r}^* = \mathbf{r}^*\mathbf{r}^+$

Ejemplo 1

Simplificar la siguiente expresión regular: $\mathbf{s} (\varepsilon \cup \mathbf{r})^* (\varepsilon \cup \mathbf{r}) \cup \mathbf{s}$.

Aplicando el teorema número 15, se tiene que: $(\varepsilon \cup \mathbf{r})^* (\varepsilon \cup \mathbf{r}) = (\varepsilon \cup \mathbf{r})^+$, luego, con el teorema 10, se puede observar que $(\varepsilon \cup \mathbf{r})^+ = \mathbf{r}^*$, después, usando el teorema 8, se *factoriza* la **s**, obteniendo: $\mathbf{s}\mathbf{r}^* \cup \mathbf{s} = \mathbf{s} (\mathbf{r}^* \cup \varepsilon)$ y finalmente si aplicamos nuevamente el teorema 10 se llega al resultado deseado:

$$\mathbf{s} (\varepsilon \cup \mathbf{r})^* (\varepsilon \cup \mathbf{r}) \cup \mathbf{s} = \mathbf{s}\mathbf{r}^*$$

Ejemplo 2

Simplificar, si es posible, la siguiente expresión regular: $(\mathbf{a}^*\mathbf{b})^* \cup (\mathbf{a}\mathbf{b}^*)^*$.

Aplicando los teoremas 13 y 14 resulta: $\varepsilon \cup (\mathbf{a} \cup \mathbf{b})^*\mathbf{b} \cup \varepsilon \cup \mathbf{a}(\mathbf{a} \cup \mathbf{b})^*$, sin embargo, los términos donde aparece $(\mathbf{a} \cup \mathbf{b})^*$ no se pueden agrupar, pues en el primer caso la **b**

es un sufijo y en el segundo caso la **a** es prefijo, y como la concatenación no es conmutativa, se concluye que la expresión original ya es la más simple.

Ejemplo 3

Demostrar que si **r** = **s*****t** entonces **r** = **sr** ∪ **t**. Podemos reemplazar **s*** por ε ∪ **s**$^+$, quedando **r** = (ε ∪ **s**$^+$) **t**, ahora reemplazando **s**$^+$ = **ss*** queda: **r** = (ε ∪ **ss***) **t**, aplicando el teorema 9: **r** = ε **t** ∪ **ss*****t**, reemplazando a **r**, **r** = **t** ∪ **sr** y conmutando, resulta finalmente que si **r** = **s*****t** entonces **r** = **sr** ∪ **t**.

Sustitución

En un lenguaje regular L, se puede reemplazar un símbolo **a** por una expresión regular y el resultado es un lenguaje regular, a esta operación le llamamos sustitución.

Ejemplo

Sea el lenguaje regular L = **0*** (**0** ∪ **1**) **1***, entonces, si reemplazamos el **0** por la expresión regular f(**0**) = e_1 y el **1** por la expresión regular f(**1**) = e_2, tenemos el lenguaje: f(L) = e_1* (e_1 ∪ e_2) e_2*, por ejemplo, si sustituimos las expresiones: e_1 = **a** y e_2 = **b***, obtenemos la transformación: f(L) = **a*** (**a** ∪ **b***) (**b***)*, y que simplificando queda: f(L) = **a*** **b*** y que obviamente es un lenguaje regular.

Homomorfismo

Homomorfismo en una substitución en la que h(**a**) tiene una sola cadena asociada a cada símbolo **a**, y por lo tanto se trata de una función invertible. En un homomorfismo se cumplen las siguientes propiedades:

- h(ε) = ε
- h(**a***w*) = h(**a**)h(*w*), para toda **a** ∈ ∑ y *w* ∈ ∑*.

Por ejemplo, si ahora sustituimos las expresiones: h(**0**) = e_1 = **a** y h(**1**) = e_2 = **ba** en L, obtenemos la transformación: h(L) = **a***(**a** ∪ **ba**)(**ba**)*, que es invertible por ser un homomorfismo.

Si aplicamos el homomorfismo sobre la cadena *w* = **00101**, tenemos h(**00101**) = **aabaaba**, mientras que h^{-1}(**aabaaba**) = **00101**.

Si L es regular, h(L) y h^{-1}(L) son también lenguajes regulares, pues se trata de casos particulares de sustituciones.

Cociente

Sean L_1 y L_2 dos lenguajes regulares, entonces definimos el cociente como:

$$L_1/L_2 = \{ x \mid \text{existe } y \in L_2 \text{ tal que } xy \in L_1\}$$

El cociente de dos lenguajes regulares también es regular; más aún, el cociente es regular si L_1 es regular y para cualquier lenguaje arbitrario L_2.

Es interesante resaltar que si $L_1 = L_3 \cdot L_2$, entonces se tiene que $L_1/L_2 = L_3$

Ejemplo

Sean los lenguajes $L_1 = \mathbf{0^*1\ 0^*}$ y $L_2 = \mathbf{1\ 0^*1}$, entonces $L_1/L_2 = \varnothing$, pues ninguna cadena cumple con la definición.

En cambio, si consideramos a los lenguajes: $L_1 = \mathbf{0^*1\ 0^*}$ y $L_2 = \mathbf{0^*1}$, tenemos que: $L_1/L_2 = \mathbf{0^*}$, porque existe y $= \mathbf{1} \in L_2$ tal que $\mathbf{0^*1} \in L_1$.

Análisis de léxico

En la programación de computadoras es necesario tener la seguridad de que los datos de entrada son válidos, por lo que es necesario contar con mecanismos para analizar la validez de la información proporcionada por el usuario. Este problema se reduce al establecimiento de una expresión regular que represente la forma que deben tener los datos de entrada válidos. Las expresiones regulares se utilizan para reconocer los componentes léxicos de un lenguaje, y forma una parte esencial en el proceso de compilación de un programa.

Ejemplo

Se desea determinar la expresión regular que utiliza un compilador de un lenguaje de computación, para verificar que una constante numérica sea válida, la que solamente puede tener algunos de los siguientes símbolos: $\sum = \{$ **.**, **–**, **dígito** $\}$, el punto puede ir en cualquier parte de la cadena que representa al número, mientras que el signo menos solamente puede ir al inicio de ésta y en donde los únicos símbolos que se repiten pertenecen al conjunto: **dígito** $= \{$ **0**, **1**, … , **9**$\}$.

Entonces la expresión regular que nos permite conocer si una cadena numérica es válida es:

$$(\ \varepsilon \cup \mathbf{-}\ )(\ \mathbf{dígito}^+\ (\ \varepsilon \cup \mathbf{.dígito}^*) \cup\ \mathbf{.dígito}^+)$$

Preguntas

a) Si L_1 y L_2 son regulares, ¿Es $L_1 \cap L_2$ regular?

b) Si L_1 y L_2 son regulares, ¿Es $L_1 \oplus L_2$ regular?

c) Si L es finito ¿Es L regular?

d) Si L es infinito, ¿Es L regular?

e) Si L es regular, ¿Es L^C regular?

Ejercicios Capítulo 2

2.1 Obtener una expresión regular para cada uno de los siguientes casos:

a) El lenguaje formado por todas las cadenas de unos y ceros que inician con dos ceros consecutivos.

b) El lenguaje formado por todas las cadenas de unos y ceros que tenga un número de ceros divisible entre tres.

c) El lenguaje formado por todas las cadenas de unos y ceros que tienen al menos dos unos consecutivos.

d) El lenguaje formado por todas las cadenas de unos y ceros que contengan cuando mucho dos ceros.

e) El lenguaje formado por todas las cadenas de unos y ceros que **solamente** tenga una ocurrencia de tres ceros consecutivos.

f) El lenguaje formado por todas las cadenas de dígitos que representen un número entero positivo (base diez) correctamente escrito.

g) El lenguaje formado por todas las cadenas de unos y ceros que tengan longitud menor o igual a 5.

h) El lenguaje formado por todas las cadenas de unos y ceros que no finalicen en **01**.

i) El lenguaje formado por todas las cadenas de unos y ceros que terminen en **1** y no contengan a la subcadena **00**.

j) El lenguaje formado por todas las cadenas de unos y ceros cuya longitud es múltiplo de 5.

k) El lenguaje formado por todas las cadenas de unos y ceros que inicien o terminen en **00** o en **11**.

2.2 Interprete en palabras el significado de cada una de las siguientes expresiones regulares:

a) (**00**)*

b) **0*1***

c) **1**(**0** ∪ **1**)*

d) (**0** ∪ **1**)***00**

e) (**0** ∪ **1**)***10**(**0** ∪ **1**)*

f) **1***01***0**(**0** ∪ **1**)*

2.3 Dada la expresión regular (**ab**)$^+$ ∪ (**cb**)*. Indicar si las siguientes cadenas pertenecen o no al lenguaje que representa:

a) w_1 = **abcb**

b) w_2 = ε

c) w_3 = **cbcbb**

d) w_4 = **ab**

e) w_5 = **abcbcbcb**

2.4 Determinar las cadenas que pertenecen al lenguaje descrito por la expresión regular siguiente: **c***a ∪ (**bc**)* ∪ **b***.

2.5 Dada la expresión regular **a** (**b** ∪ **c**) **a** (**a** ∪ **b** ∪ **c**)* **a**, ¿Cuántas cadenas de longitud 6 representa?

2.6 Simplificar las siguientes expresiones:

a) (ε ∪ **ab**)*

b) **a** (ε ∪ **aa**)* **a** ∪ ε

c) (**a** ∪ ε) **a***b

d) (((**a***a) **b**) ∪ **b**)

e) (ε ∪ **aa**) (ε ∪ **aa**)*

f) (**aa**)* **a** ∪ (**aa**)*

g) (**a** ∪ **b**)* **a** (**a** ∪ **b**)*

h) **a** (ε ∪ **aa**)* (ε ∪ **aa**) ∪ **a**

i) ∅* ∪ **a*** ∪ **b*** ∪ (**a*** ∪ **b***)$^+$

j) ((**a***b*)* · (**b***a*)*)*

k) (**a***b)* ∪ (**b***a)*

l) (ε ∪ **a**$^+$) **bb**$^+$ (ε ∪ **c**)*

m) **y**(ε ∪ **x**$^+$) ∪ (**yy**$^+$(ε ∪ **x**)*)

n) (ε ∪ **x**) (ε ∪ **x**)$^+$ ∪ (ε ∪ **x**) ∪ ∅*

o) (**ba***)* ∪ ε ∪ (**a** ∪ **b**)$^+$

p) (**a** ∪ **b**)(ε ∪ **aa**)*(ε ∪ **aa**) ∪ (**a** ∪ **b**)

q) **a***b ((**a** ∪ **b**) **a***b)* ∪ **a***b

r) (**b***a)* ∪ (**a** ∪ **b**)$^+$**a**

s) (**abc***)* ∪ **ab** ∪ **ab**(**c** ∪ **ab**)$^+$

Capítulo

3

Autómatas Finitos

Se define Diagrama de Transiciones, se define Autómata Finito Determinista. Se presentan las Tablas de Transiciones, se establece una metodología para encontrar el Autómata Mínimo Equivalente y el algoritmo para determinar si dos autómatas dados son equivalentes; finalmente se presentan algunas variantes con valores de salida.

Diagramas de Transiciones

Las expresiones regulares sencillas nos permiten determinar con facilidad si una cadena pertenece o no a un lenguaje dado. Por ejemplo: Si tenemos al lenguaje definido por la expresión regular **a***b*, ésta se puede interpretar como el lenguaje que contiene cualquier cadena que comience con cualquier cantidad de **a**s, seguida por cualquier cantidad de **b**s, como por ejemplo: **aaab**, **abbb**, **a**, **bb**, ε, etc. También nos permite determinar que no pertenecen a este lenguaje cadenas como las siguientes: **abab**, **baba**, **bba**, etc.

Otra técnica que permite identificar si una cadena pertenece o no a un lenguaje dado es el empleo de Diagramas de Transiciones, los cuales son grafos dirigidos a cuyos nodos se les denomina *Estados* y a sus aristas se les llama *Transiciones*, éstas se encuentran etiquetadas con algún símbolo del alfabeto, como se muestra en el ejemplo de la figura 3.1.

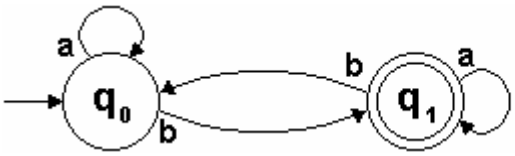


Figura 3.1

Existe un único estado que se le llama *Estado Inicial*, el cual se señala con una flecha, a partir del cual se comienza el reconocimiento de la cadena; cada símbolo leído provoca una transición de un estado a otro, siguiendo la arista etiquetada con éste.

Este proceso se repite hasta agotar la cadena. Si el estado donde finalizamos es un *Estado de Aceptación*, que se identifica por un doble círculo, quiere decir que la cadena analizada pertenece al lenguaje, en caso contrario es rechazada.

Si observamos como se cambia de estado sobre el diagrama de transiciones anterior, con cada uno de los símbolos de la cadena *w* = **aaabab**, para determinar si pertenece o no al lenguaje representado por éste, observaremos que las transiciones realizan los cambios de estado mostrados en la secuencia siguiente:

$$q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_0$$

Y como el estado terminal q_0 no es un estado de aceptación, la cadena anterior es rechazada.

Para entender mejor el proceso de análisis de una cadena, se emplea la siguiente notación, en la que se muestra cada paso como un par formado por el estado actual y la parte de la cadena pendiente de procesar, a este par se le conoce como una Descripción Instantánea (**DI**) de la fase en que se encuentra actualmente el autómata. Para este caso, la **DI** inicial es: (q_0, **a**aabab), se subraya el primer símbolo para indicar que éste es el símbolo que provocará la próxima transición. El paso de una **DI** a la subsiguiente, se denota por medio del siguiente símbolo: ⊢, de tal forma que la secuencia de fases instantáneas del análisis anterior se representa así:

(q_0, **a**aabab) ⊢ (q_0, **a**abab) ⊢ (q_0, **a**bab) ⊢ (q_0, **b**ab) ⊢ (q_1, **a**b) ⊢ (q_1, **b**) ⊢ (q_0, ε)

Una vez agotada la cadena, y a no hay transiciones posibles y el autómata se detiene en el estado q_0, como ya se había indicado, rechazando la cadena.

Una tercera representación gráfica de este proceso se hace considerando a la cadena contenida en una cinta suficientemente grande, y un puntero que inicialmente señala al primer símbolo de la cadena y que se encuentra en el estado inicial q_0.

Conforme realiza las transiciones respectivas, el puntero irá desplazándose hacia la derecha para leer los siguientes símbolos, mientras cambia de estado según corresponda a cada transición, hasta llegar al final de la cadena, indicado, en este caso, por el símbolo especial **$**. Es entonces cuando el análisis concluye, y se determina el estado en el que se finaliza, que como hemos visto, se trata del estado q_0. Esta representación se muestra gráficamente en la siguiente figura:

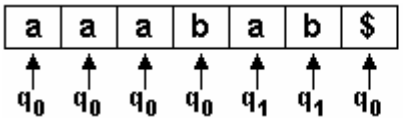


Figura 3.2

Un análisis detallado de todas las cadenas que nos conducen al estado de aceptación q_1 en el diagrama de la Figura 3.1, nos lleva a la conclusión de que éstas deben de poseer una cantidad impar de **b**s para ser aceptadas.

Ejemplo 1

Consideremos la necesidad de construir un diagrama de transiciones para reconocer el lenguaje formado por todas las cadenas provenientes del alfabeto Σ = { **a**, **b** } que terminan en **b**, es fácil visualizar que el diagrama de transiciones debe tener un estado de aceptación q_1, al que se accede cuando aparece el símbolo **b** en la cadena, esperando que sea el último símbolo, pero que se sale de él cuando se lee el símbolo **a**, de esta manera, concebimos el diagrama de transiciones mostrado en la figura 3.3.

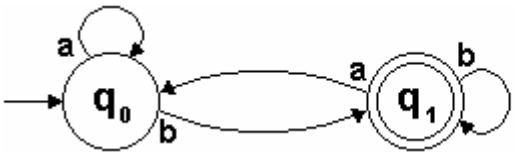


Figura 3.3

Ejemplo 2

El siguiente diagrama de transiciones consta de tres estados, etiquetados por q_0, q_1 y q_2, respectivamente; de los cuales q_1 es el estado de Aceptación. Además hay 6 transiciones, dos para cada estado, etiquetadas con los símbolos **a** y **b**, aunque por comodidad se suele usar una sola flecha para varios símbolos que coinciden en el mismo estado de destino.

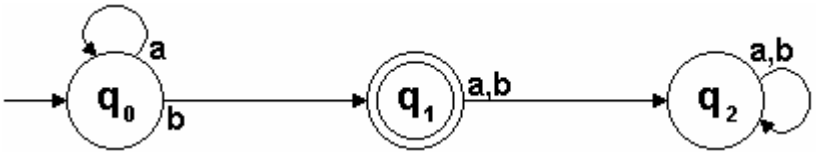


Figura 3.4

Las cadenas formadas por cualquier cantidad de **a**s seguida de una única **b** son aceptadas por el lenguaje que representa el diagrama anterior, en general se puede expresar este lenguaje de la forma L = { $\mathbf{a}^n\mathbf{b}$ | $n \geq 0$ }.

Cualquier otra cadena no será aceptada, dado que se provocará que pase al estado q_2, del cual no podrá salir y dado que no es un estado de Aceptación, se le denomina *Estado no Deseado* o *Estado de Rechazo*.

La expresión regular **a***b nos describe el lenguaje antes citado.

Ejemplo 3

Ahora consideremos el lenguaje (**ab**)*, el cual acepta, entre otras, a la cadena vacía, por lo que es necesario hacer que el Estado Inicial q_0 sea también un Estado de Aceptación.

El diagrama de transiciones debe ser semejante al que se muestra en la figura 3.5:

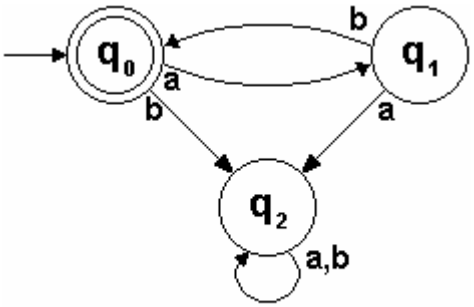


Figura 3.5

Se puede verificar que las cadenas que inician con una **b**, las que tienen dos **a**s o dos **b**s consecutivas pasarán a un estado no deseado y serán rechazadas.

Tabla de Transiciones

Un diagrama de transiciones también puede ser representado de manera tabular de la siguiente forma: colocamos a cada símbolo como encabezado de cada una de las columnas y a cada estado al inicio de cada uno de los renglones. La flecha indica el estado inicial y el asterisco se usa para denotar los estados de aceptación. Dentro de la tabla se coloca el estado siguiente, según corresponda a cada transición. La tabla de transiciones del diagrama del ejemplo anterior se muestra en la tabla 3.1:

	a	**b**
$\rightarrow$*q_0	q_1	q_2
q_1	q_2	q_0
q_2	q_2	q_2

Tabla 3.1

Las tablas de transiciones son muy útiles para realizar programas de computadora que permitan el reconocimiento de lenguajes regulares, pero para cuando se hace el proceso de reconocimiento en forma manual, es preferible utilizar los diagramas de transiciones, tal como lo seguiremos haciendo en los siguientes ejemplos.

Ejemplo 4

Ahora consideremos el lenguaje L = { $(\mathbf{ab})^n \mid n \geq 1$}, el cual se identifica con la expresión regular $(\mathbf{ab})^+$. Para construir el diagrama de transiciones correspondiente que acepte este lenguaje, podemos seguir los siguientes pasos:

Primero construimos un diagrama que acepte una **a**, seguida de una **b**, (por ser **ab** la cadena más pequeña de dicho lenguaje) para llegar al estado de aceptación, así:

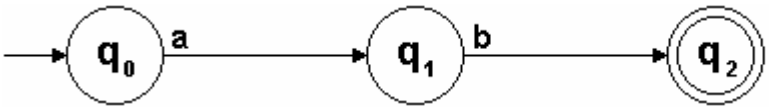


Figura 3.6

Cualquier cadena que empiece con una **b**, o que después de la primera **a** tenga otra **a** debe ser rechazada, pasando a un Estado No Deseado, tal como se muestra en la figura 3.7.

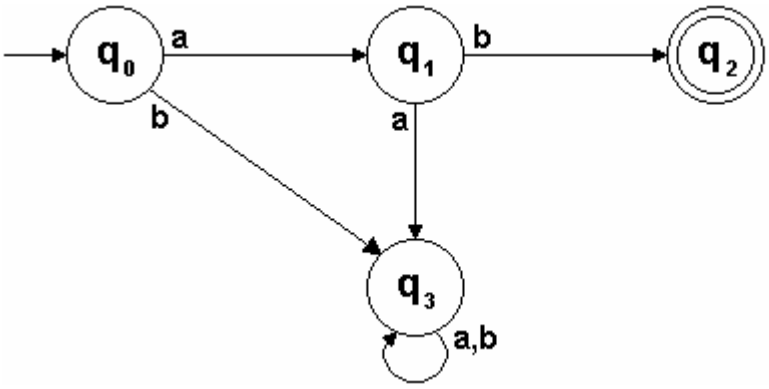


Figura 3.7

Si hay más de dos símbolos en la cadena, éstos tienen que ir en parejas de la forma **ab**, por lo que, si el tercer símbolo es una **b** debemos pasar del estado q_2 al estado de rechazo, mientras que una **a** después de la primera **b**, nos lleva nuevamente al estado q_1, en donde esperamos que el siguiente símbolo sea una **b** para retornar al estado de aceptación. El diagrama completo queda finalmente como se ve en la figura 3.8:

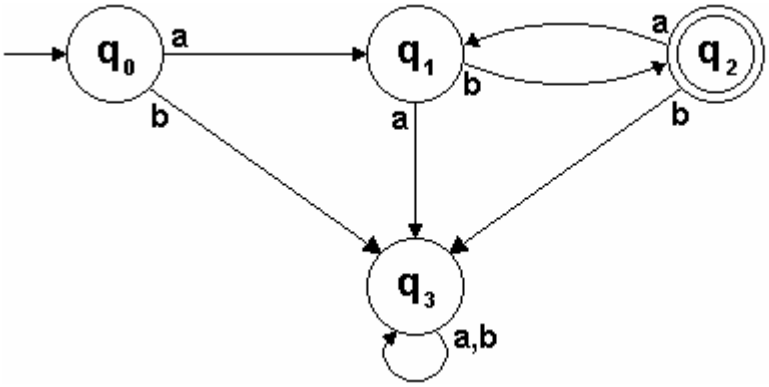


Figura 3.8

Ejemplo 5

Construya el diagrama de transiciones para el reconocimiento de las cadenas del lenguaje en el alfabeto $\sum$ = { **a**, **b** }, que contengan el sufijo (que terminan en) **ba**.

Primeramente hay que analizar la cadena más simple que pertenece a este lenguaje y que es *w* = **ba** y se trazan las transiciones necesarias para llegar a un estado de aceptación, como se muestra en la Figura 3.9.

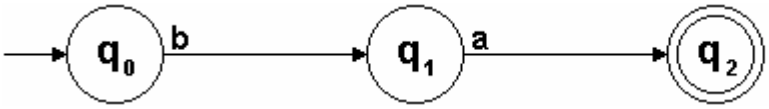


Figura 3.9

Posteriormente se agregan las transiciones faltantes; como falta una para el símbolo **a**, desde el estado q_0, la cual resulta ser un lazo sobre el mismo estado, tal como se indica en la Figura 3.10.

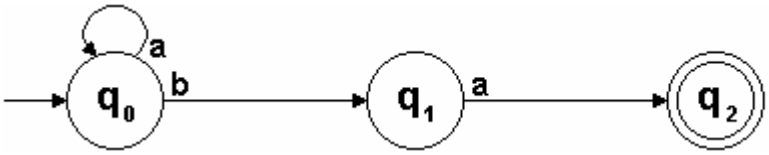


Figura 3.10

Similarmente, para el estado q_1 falta la transición para el símbolo **b**, la cual es también un lazo en ese estado, Figura 3.11.

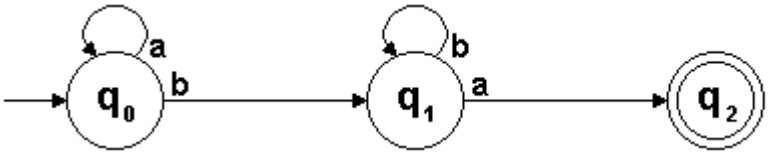


Figura 3.11

Finalmente, las transiciones desde el estado q_2, hay una transición hacia q_0 para el símbolo **a** y hay otra hacia q_1 para el símbolo **b**, quedando el diagrama completo, tal como se aprecia en la Figura 3.12.

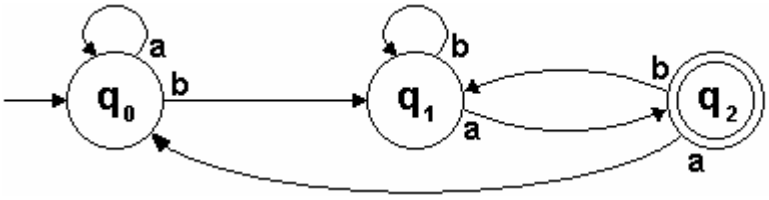


Figura 3.12

Autómata Finito Determinista

Un autómata es el modelo matemático de un sistema. Al modelo matemático que hemos definido por medio de un diagrama de transiciones, que representa a una máquina que pasa de un estado a otro como respuesta a cada uno de los símbolos de

una cadena de entrada, se le llama Autómata Finito Determinista y lo denotamos como **AFD**.

Formalmente se define a un **AFD** por la quíntupla M = (Q, Σ, s, F, δ), donde: Q es un conjunto finito de estados, Σ es el alfabeto de entrada, s $\in$ Q es el estado inicial, F es el subconjunto de Q de los estados de aceptación (F $\subseteq$ Q) y δ es la función de transición (δ: Q $\times$ Σ $\rightarrow$ Q).

La característica principal de un **AFD** es que δ es una función que está definida para todos los posibles estados $q_i \in Q$ y para todos los símbolos $\sigma_j \in \Sigma$. Es decir para cualquier pareja de la forma (q_i, σ_j) siempre existe un único estado siguiente.

Ejemplo 1

Sea M el **AFD** donde Q = { q_0, q_1 }, Σ = { **a**, **b** }, s = q_0, F = { q_0 } y las siguientes transiciones: δ(q_0, **a**) = q_0, δ(q_0, **b**) = q_1; δ(q_1, **a**) = q_1 y δ(q_1, **b**) = q_0.

Esto se representa por medio de la siguiente tabla de transiciones:

δ	**a**	**b**
$\rightarrow$*q_0	q_0	q_1
q_1	q_1	q_0

Tabla 3.2

A partir de la tabla podemos construir el diagrama de transiciones correspondiente a este **AFD**, el cual se muestra en la Figura 3.13:

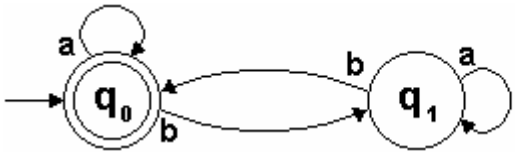


Figura 3.13

Este **AFD** acepta cadenas con una cantidad par de **b**s, para verificarlo, podemos probar algunas cadenas sobre el diagrama, como por ejemplo: w = **abba**, lo que nos resulta en la siguiente secuencia de estados:

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_0 \xrightarrow{a} q_0$$

Es usual que se generalice la definición de la función de transición para cadenas completas no vacías, de la siguiente manera: δ: Q $\times$ Σ^+ $\rightarrow$ Q, por lo que es frecuente

que nos encontremos con expresiones como la siguiente: $\delta(q_0,$ **abba**$) = q_0$, para resumir las transiciones anteriores.

Si probamos el diagrama de transiciones con una cadena que contenga una cantidad impar de **b**s, terminaremos invariablemente en el estado q_1, como por ejemplo la cadena w = **abaa** nos arroja la siguiente secuencia de estados:

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_1 \xrightarrow{a} q_1$$

Ejemplo 2

Sea M el **AFD** definido por la tabla de transiciones siguiente:

δ	**a**	**b**
$\rightarrow$*q_0	q_0	q_1
*q_1	q_0	q_2
*q_2	q_0	q_3
q_3	q_3	q_3

Tabla 3.3

De la información contenida en la tabla se desprende que: Q = { q_0, q_1, q_2, q_3 }, s = q_0, $\sum$ = { **a**, **b** }, F = { q_0, q_1, q_2 } y las transiciones siguientes: $\delta(q_0,$ **a** $) = q_0$, $\delta(q_0,$ **b** $) = q_1$, $\delta(q_1,$ **a** $) = q_0$, $\delta(q_1,$ **b**$) = q_2$, $\delta(q_2,$ **a** $) = q_0$, $\delta(q_2,$ **b** $) = q_3$, $\delta(q_3,$ **a** $) = q_3$ y $\delta(q_3,$ **b**$) = q_3$.

La figura siguiente muestra el diagrama de transiciones correspondiente a este **AFD**, el cual acepta el lenguaje formado por cadenas que no contengan la secuencia **bbb**:

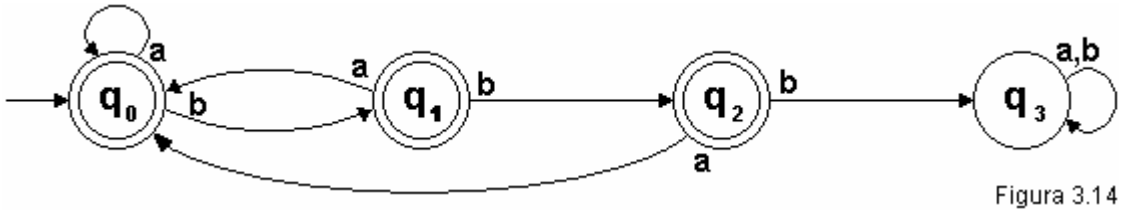


Figura 3.14

Y a continuación se detallan las descripciones instantáneas para la aceptación de la cadena w = **abbab**: $(q_0,$ **abbab**$) \vdash (q_0,$ **bbab**$) \vdash (q_1,$ **bab**$) \vdash (q_2,$ **ab**$) \vdash (q_0,$ **b**$) \vdash (q_1, \varepsilon)$

Ejemplo 3

Sea M el **AFD** donde Q = { q_0, q_1, q_2, q_3 }, $\sum$ = { **0**, **1** }, F = { q_0 }, s = q_0 y por las siguientes transiciones que se relacionan: $\delta($ $q_0,$ **0** $) = q_1$, $\delta($ $q_0,$ **1** $) = q_2$, $\delta($ $q_1,$ **0** $) = q_0$, $\delta($ $q_1,$ **1**$) = q_3$, $\delta($ $q_2,$ **0** $) = q_3$, $\delta($ $q_2,$ **1** $) = q_0$, $\delta($ $q_3,$ **0** $) = q_2$ y $\delta($ $q_3,$ **1**$) = q_1$.

La tabla 3.4 nos representa toda la información del **AFD** antes descrito:

δ	**0**	**1**
$\rightarrow$*q_0	q_1	q_2
q_1	q_0	q_3
q_2	q_3	q_0
q_3	q_2	q_1

Tabla 3.4

Este **AFD**, acepta al lenguaje formado por las cadenas $x \in \{ \mathbf{0}, \mathbf{1} \}^*$, tales que $N_0(x)$ es par y $N_1(x)$ es par, (la función $N_a(x)$ representa la cantidad de veces que la cadena *x* contiene al símbolo **a**). El diagrama de transiciones correspondiente a este **AFD** es el que se muestra en la figura 3.15, a continuación:

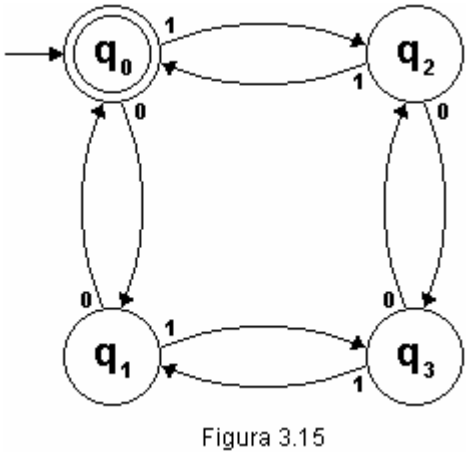


Figura 3.15

AFD Complemento

En el capítulo anterior mencionamos que si L es regular, entonces $L^C = \sum^* - L$ también es regular, ahora veremos una forma muy sencilla de demostrarlo por medio de los autómatas:

Sea M_1 el **AFD** que acepta el lenguaje L a partir del alfabeto $\sum$, formado por el conjunto de estados Q, y de los cuales F es el subconjunto de estados de aceptación y con función de transición δ. Entonces M_2 será el **AFD** que acepte al lenguaje L^C, si está formado por el mismo conjunto de estados Q, el mismo alfabeto la misma función de transición δ, el mismo alfabeto $\sum$, pero cuyo conjunto de estados finales es ahora: $F^C = Q - F$. Es decir, M_2 acepta las cadenas que M_1 rechaza y viceversa.

Ejemplo

La Figura 3.16 nos muestra el diagrama de transiciones del autómata que acepta el lenguaje formado por las cadenas que contengan a la subcadena **bbb**, el cual es el complemento del **AFD** mostrado antes en la figura 3.14.

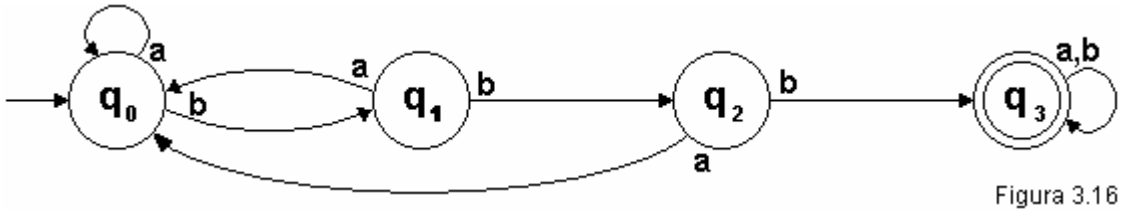


Figura 3.16

Autómatas Equivalentes

Si M es un **AFD**, vamos a denotar como L(M) al lenguaje aceptado por M, es decir, $L(M) = \{ w \mid \delta(q_0,w) \in F\}$, es el conjunto de cadenas que hacen que M pase del estado inicial a un estado de aceptación.

Sean M_1 y M_2 dos **AFD**s, entonces se dice que M_1 y M_2 son equivalentes si se cumple que $L(M_1) = L(M_2)$.

Considere a los **AFD**s M_1 y M_2 sobre el alfabeto $\sum = \{ \mathbf{0}, \mathbf{1} \}$ que se muestran en la figura 3.17, como se puede verificar, ambos aceptan el mismo lenguaje **0*****10***, por lo tanto, son equivalentes.

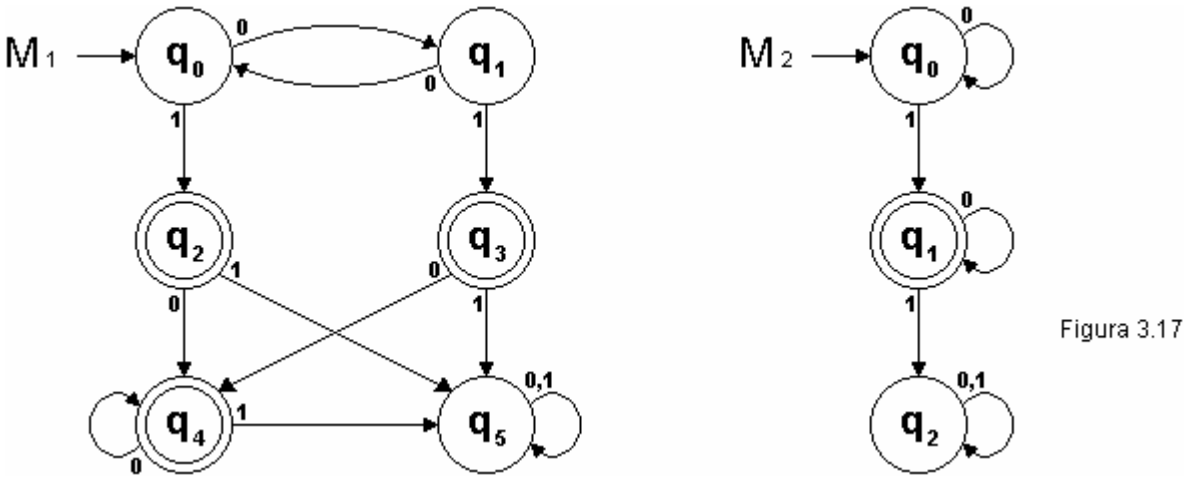


Figura 3.17

A pesar de que pueden existir varios autómatas equivalentes para un lenguaje dado, solamente existe un único **AFD Mínimo** equivalente (de mínimo número de estados), que se considera como la representación óptima para dicho lenguaje, como es el caso del autómata etiquetado como M_2 en este ejemplo.

Estados Accesibles

Dado un **AFD** que tiene n estados, se dice que un estado q_i es accesible si existe una cadena *w*, $|w| < n$, que permite que el autómata llegue a ese estado partiendo del estado inicial q_0.

Es decir que existe *w* tal que $\delta(q_0,w) = q_i$. Gráficamente diríamos que existe una trayectoria desde el estado q_0 al estado q_i formada por menos de n transiciones.

Autómatas conexos

Decimos que un **AFD** es un *Autómata conexo* si todos los estados de Q son accesibles desde el estado inicial q_0.

Dado un autómata no conexo, podemos obtener, a partir de él, un autómata equivalente que sea conexo, simplemente eliminando todos los estados que no sean accesibles desde q_0.

Ejemplo

El siguiente autómata no es conexo, porque contiene dos estados no accesibles: q_2 y q_3, lo que se puede verificar fácilmente:

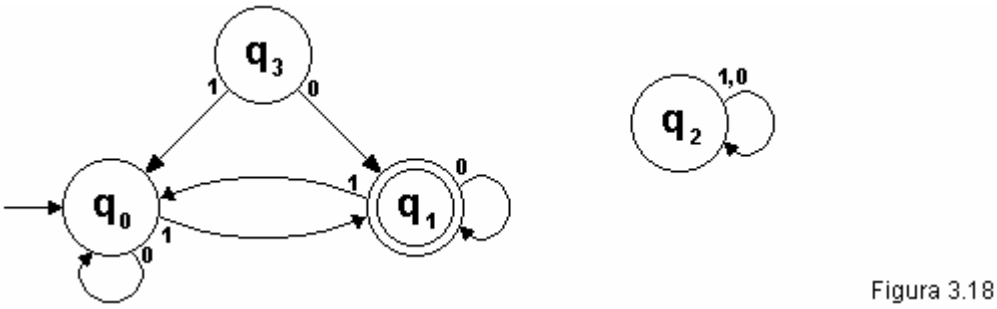


Figura 3.18

Eliminando esos estados queda el **AFD** equivalente que se muestra a continuación:

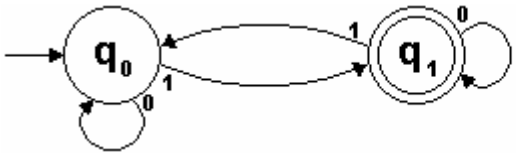


Figura 3.19

Estados Equivalentes

Dado un **AFD**, se dice que dos estados p y q son equivalentes (y se denota como pEq) si para toda cadena $w \in \Sigma^*$, se cumple que: $\delta(p, w) \in F \Leftrightarrow \delta(q, w) \in F$.

AFD Mínimo Equivalente

Dado M, un **AFD** cualquiera, se puede obtener M', el **AFD** con el mínimo número de estados que sea equivalente a M, por medio del siguiente algoritmo:

1. Se obtiene el autómata conexo equivalente, eliminando todos los estados que no son accesibles desde q_0.
2. Se realiza una partición de los estados de Q en dos clases:
   - En la clase C_1 se incluyen a los estados de aceptación, es decir: $C_1 = F$.
   - En la clase C_2 a los demás estados, esto es: $C_2 = Q - F$.
3. Se analiza cada clase C_m con objeto de ver si para todo $q_k \in C_m$ se cumple para cada $\sigma_i \in \sum$ que $\delta(q_k,\sigma_i) \in C_n$, para alguna clase C_n.
4. En caso de que no se cumpla lo anterior para alguna clase C_m, se realiza una partición de esa clase, dividiéndola en subclases que estén formadas por los grupos de estados que satisfagan la condición anterior entre sí.
5. Se repite el paso 3 hasta verificar que todas las clases cumplan la condición requerida, entonces cada clase de estados equivalentes resultante se representará por un solo estado en el **AFD** mínimo.

Ejemplo 1

Considérese al **AFD** M_1 mostrado en la figura 3.17. Es fácil verificar que se trata de un **AFD** conexo, entonces, aplicando el segundo paso del algoritmo, se hace la partición inicial: $C_1 = \{ q_2, q_3, q_4 \}$ y $C_2 = \{ q_0, q_1, q_5 \}$.

Ahora, continuando con el paso 3 del algoritmo, analizamos las transiciones para los tres estados contenidos en la clase C_1:

$\delta(q_2, \mathbf{0}) = q_4 \in C_1$ $\qquad$ $\delta(q_2, \mathbf{1}) = q_5 \in C_2$

$\delta(q_3, \mathbf{0}) = q_4 \in C_1$ $\qquad$ $\delta(q_3, \mathbf{1}) = q_5 \in C_2$

$\delta(q_4, \mathbf{0}) = q_4 \in C_1$ $\qquad$ $\delta(q_4, \mathbf{1}) = q_5 \in C_2$

Para todos los estados de esta clase se cumple la condición de equivalencia requerida en este paso, por lo que no se requiere realizar ninguna partición.

A continuación hacemos el análisis de las transiciones para los tres estados contenidos en la clase C_2:

$\delta(q_0, \mathbf{0}) = q_1 \in C_2$ $\quad\quad$ $\delta(q_0, \mathbf{1}) = q_2 \in C_1$

$\delta(q_1, \mathbf{0}) = q_0 \in C_2$ $\quad\quad$ $\delta(q_1, \mathbf{1}) = q_3 \in C_1$

$\delta(q_5, \mathbf{0}) = q_5 \in C_2$ $\quad\quad$ $\delta(q_5, \mathbf{1}) = q_5 \in C_2$

Aquí observamos que no se satisface la condición exigida para todos los casos, entonces hay que separar los estados de C_2 en dos subclases a saber: $C_3 = \{ q_0, q_1 \}$, dado que ambos se comportan de forma equivalente y $C_4 = \{ q_5 \}$.

Ahora debemos repetir el paso 3, para verificar que se cumple la condición requerida para cada una de las tres clases que ahora tenemos.

Se puede observar que esta partición no afecta los resultados para la clase C_1, puesto que solamente se reemplaza C_2 por C_4 en las transiciones del símbolo **1**.

A continuación se muestra que también se cumple la condición de equivalencia para las otras dos clases:

Para C_3:

$\delta(q_0, \mathbf{0}) = q_1 \in C_3$ $\quad\quad$ $\delta(q_0, \mathbf{1}) = q_2 \in C_1$

$\delta(q_1, \mathbf{0}) = q_0 \in C_3$ $\quad\quad$ $\delta(q_1, \mathbf{1}) = q_3 \in C_1$

Y para C_4 la condición se cumple por necesidad, puesto que tiene un solo estado:

$\delta(q_5, \mathbf{0}) = q_5 \in C_4$ $\quad\quad$ $\delta(q_5, \mathbf{1}) = q_5 \in C_4$

Con esto se concluye que el **AFD** Mínimo solamente consta de tres estados, el cual está representado por la siguiente tabla de transiciones.

δ	**0**	**1**
$\rightarrow C_3$	C_3	C_1
$*C_1$	C_1	C_4
C_4	C_4	C_4

Tabla 3.5

Observe que la clase C_3 que contiene el estado inicial original q_0 se considera el nuevo estado inicial y que la clase C_1, que surgió a partir de los estados de aceptación será el nuevo estado de aceptación.

El diagrama de transiciones correspondiente a este **AFD** mínimo es el mismo que fue representado por el autómata M_2 en la figura 3.17.

Ejemplo 2

Encontrar el **AFD** Mínimo equivalente al mostrado en la figura a continuación:

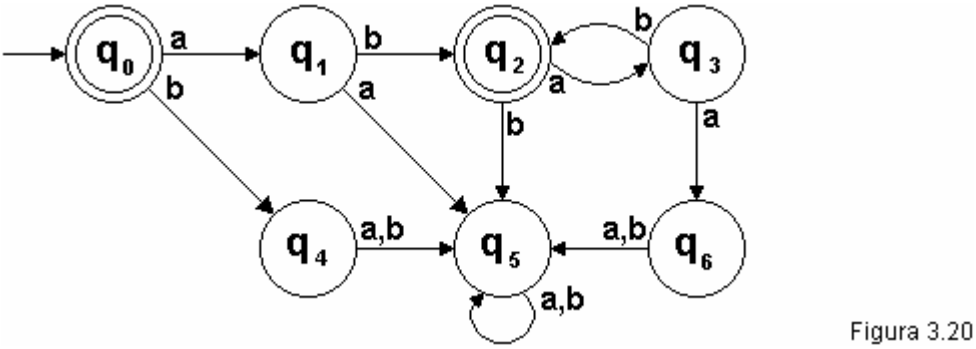


Figura 3.20

Primero verificamos que todos los estados son accesibles, luego realizamos la primera partición de Q, de donde se tiene: $C_1 = \{ q_0, q_2 \}$ y $C_2 = \{ q_1, q_3, q_4, q_5, q_6 \}$, ahora analizamos las transiciones para los estados en C_1:

$\delta(q_0, \mathbf{a}) = q_1 \in C_2$ $\qquad$ $\delta(q_0, \mathbf{b}) = q_4 \in C_2$

$\delta(q_2, \mathbf{a}) = q_3 \in C_2$ $\qquad$ $\delta(q_2, \mathbf{b}) = q_5 \in C_2$

Para todos los estados de esta clase se cumple la condición requerida en el paso 3.

Ahora analizamos las transiciones para los cinco estados de la clase C_2:

$\delta(q_1, \mathbf{a}) = q_5 \in C_2$ $\qquad$ $\delta(q_1, \mathbf{b}) = q_2 \in C_1$

$\delta(q_3, \mathbf{a}) = q_6 \in C_2$ $\qquad$ $\delta(q_3, \mathbf{b}) = q_2 \in C_1$

$\delta(q_4, \mathbf{a}) = q_5 \in C_2$ $\qquad$ $\delta(q_4, \mathbf{b}) = q_5 \in C_2$

$\delta(q_5, \mathbf{a}) = q_5 \in C_2$ $\qquad$ $\delta(q_5, \mathbf{b}) = q_5 \in C_2$

$\delta(q_6, \mathbf{a}) = q_5 \in C_2$ $\qquad$ $\delta(q_6, \mathbf{b}) = q_5 \in C_2$

Podemos ver que no se cumple la condición para todas las transiciones del símbolo **b**, por lo que es necesario dividir a la clase C_2 en dos subclases a saber: $C_3 = \{ q_1, q_3 \}$ y $C_4 = \{ q_4, q_5, q_6 \}$.

Finalmente repetimos la verificación para las tres clases que tenemos ahora; es fácil comprobar que esta partición ya es definitiva y que todas las clases cumplen la condición de equivalencia impuesta por el algoritmo.

Por lo que se concluye que el **AFD** Mínimo equivalente solamente consta de tres estados, el cual está representado por la siguiente tabla de transiciones.

δ	**a**	**b**
$\rightarrow$*C_1	C_3	C_4
C_3	C_4	C_1
C_4	C_4	C_4

Tabla 3.6

Algoritmo para ver si dos AFD son equivalentes

El algoritmo anterior para hallar el **AFD** *Mínimo Equivalente* también se utiliza para verificar si dos **AFD**s cualesquiera son o no equivalentes.

Para ello, simplemente se hace la unión de todas las transiciones de ambos autómatas en una sola tabla de transiciones, y se aplica el algoritmo anterior, si al final resulta que ambos estados iniciales pertenecen a la misma clase de equivalencia significa que ambos autómatas son equivalentes.

Ejemplo

Verifique si es que los dos **AFD**s siguientes son equivalentes o no:

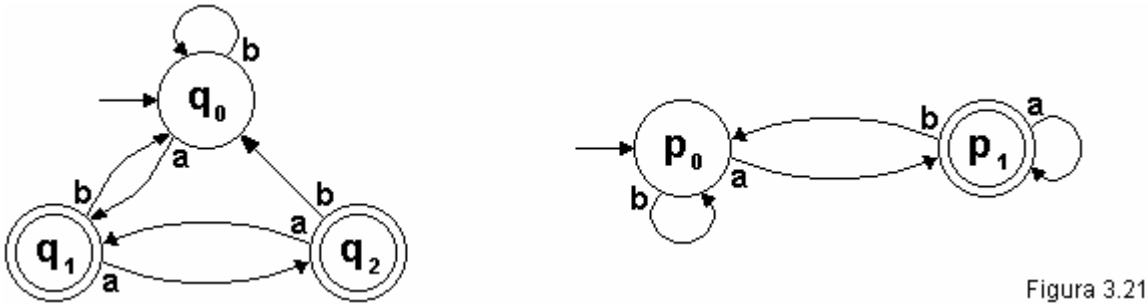


Figura 3.21

Haciendo la unión de las transiciones de ambos autómatas obtenemos la siguiente tabla de transiciones:

δ	**a**	**b**
$\rightarrow q_0$	q_1	q_0
$*q_1$	q_2	q_0
$*q_2$	q_1	q_0
$\rightarrow p_0$	p_1	p_0
$*p_1$	p_1	p_0

Tabla 3.7

Aplicando el algoritmo anterior, se tiene que: $C_1 = \{ q_1, q_2, p_1 \}$ y $C_2 = \{ q_0, p_0 \}$, ahora, analizando las transiciones para los estados en C_1 tenemos:

$\delta(q_1, \mathbf{a}) = q_2 \in C_1$ $\delta(q_1, \mathbf{b}) = q_0 \in C_2$

$\delta(q_2, \mathbf{a}) = q_1 \in C_1$ $\delta(q_2, \mathbf{b}) = q_0 \in C_2$

$\delta(p_1, \mathbf{a}) = p_1 \in C_1$ $\delta(p_1, \mathbf{b}) = p_0 \in C_2$

Y verificando las transiciones de los estados de la clase C_2:

$\delta(q_0, \mathbf{a}) = q_1 \in C_1$ $\delta(q_0, \mathbf{b}) = q_0 \in C_2$

$\delta(p_0, \mathbf{a}) = p_1 \in C_1$ $\delta(p_0, \mathbf{b}) = p_0 \in C_2$

Como ambos estados iniciales q_0 y $p_0 \in C_2$, se concluye que los **AFD**s son equivalentes, ambos aceptan las cadenas del alfabeto $\sum$ = { **a**, **b** } que terminan en **a**.

Máquina de Moore

La *Máquina de Moore* es una variedad de **AFD**, en el cual no existen estados de aceptación propiamente dichos, sino que a cada estado se le asocia un símbolo o una cadena de salida, que generalmente no se relaciona con el alfabeto de entrada.

La respuesta de la máquina de Moore será el símbolo o la cadena asociada con el estado en el que se finalice el análisis de la cadena de entrada.

Ejemplo

La siguiente Máquina de Moore permite calcular el módulo base 3 de cualquier número en formato binario. (La función módulo es el residuo que resulta de dividir el primer argumento entre el segundo) En la parte superior de cada estado se indica el símbolo de salida correspondiente.

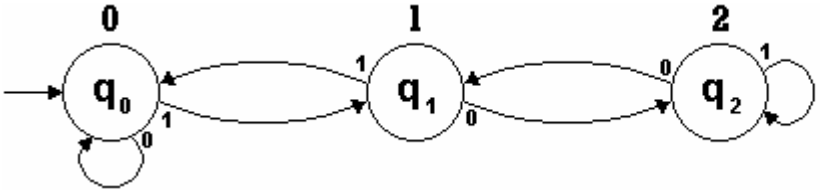


Figura 3.22

Tomando, por ejemplo, el número decimal 19, buscamos su representación binaria, lo que nos da la cadena de entrada: **10011**, al analizarla, veremos que en este caso las **DI**s mostrarán a un tercer elemento, que el símbolo de salida de cada estado:

$(q_0, \underline{\mathbf{1}}\mathbf{0011}, \mathbf{0}) \vdash (q_1, \underline{\mathbf{0}}\mathbf{011}, \mathbf{1}) \vdash (q_2, \underline{\mathbf{0}}\mathbf{11}, \mathbf{2}) \vdash (q_1, \underline{\mathbf{1}}\mathbf{1}, \mathbf{1}) \vdash (q_0, \underline{\mathbf{1}}, \mathbf{0}) \vdash (q_1, \underline{\varepsilon}, \mathbf{1})$

Al agotar la cadena se finaliza en el estado q_1, por lo que el símbolo de salida es **1**, y por lo tanto, se concluye que 19 **Mod** 3 = 1.

Máquina de Mealy

La *Máquina de Mealy* es otra variedad de **AFD**, en la cual tampoco existen estados de aceptación, sino que en este caso, a cada transición se le asocia con un símbolo de salida (que suele pertenecer a un alfabeto distinto al de entrada). Los símbolos de salida se van concatenando, conforme se generan, para formar una cadena de salida de la misma longitud que la cadena de entrada. Cada transición de este tipo de autómatas se denota como: **E/S**, donde **E** es el símbolo de entrada y **S** es el símbolo de salida

Ejemplo

Construir una Máquina de Mealy sobre el alfabeto $\sum$ = { **a**, **b** }, que genere una cadena de salida cuyo símbolo inicial sea una **a**, y que el i-ésimo símbolo sea una **a**, si los símbolos i – 1 e i son diferentes en la cadena de entrada, en caso contrario, el símbolo será una **b**.

Las transiciones iniciales son: $\delta(q_0, \mathbf{a/a}) = q_1$ y $\delta(q_0, \mathbf{b/a}) = q_2$

Las transiciones desde q_1 son: $\delta(q_1, \mathbf{a/b}) = q_1$ y $\delta(q_1, \mathbf{b/a}) = q_2$

Mientras que las transiciones desde q_2 son: $\delta(q_2, \mathbf{a/a}) = q_1$ y $\delta(q_2, \mathbf{b/b}) = q_2$

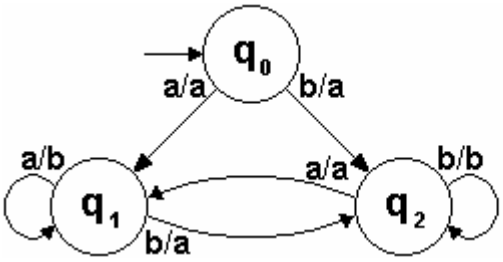


Figura 3.23

La figura 3.23 muestra la construcción de este autómata, donde podemos comprobar que si la cadena de entrada es **aabba**, la salida será: **ababa**, tal como se muestra en la siguiente secuencia de **DI**s.

$(q_0,\underline{\mathbf{a}}\mathbf{abba},\varepsilon) \vdash (q_1,\underline{\mathbf{a}}\mathbf{bba},\mathbf{a}) \vdash (q_1,\underline{\mathbf{b}}\mathbf{ba},\mathbf{ab}) \vdash (q_2,\underline{\mathbf{b}}\mathbf{a},\mathbf{aba}) \vdash (q_2,\underline{\mathbf{a}},\mathbf{abab}) \vdash (q_1,\underline{\varepsilon},\mathbf{ababa})$

Transductor Determinista

Un dispositivo semejante a la Máquina de Mealy es el llamado *Transductor Determinista*, cuyo propósito es el de transformar cadenas de entrada en cadenas de salida, para cada transición agrega a la cadena de salida, cero, uno o varios símbolos,

dependiendo del estado actual y el símbolo de entrada. Cada transición de este tipo de dispositivos se denota por: **E**/**S**, donde **E** es el símbolo de entrada y **S** es la subcadena de salida, que incluso puede ser la cadena vacía.

Ejemplo

Construya un *Transductor Determinista* que genere la cadena $\mathbf{a}^n$, donde n es el número de ocurrencias de la subcadena **ab** en *w*.

Las transiciones desde q_0 son: $\delta(q_0, \mathbf{a}/\varepsilon) = q_1$ y $\delta(q_0, \mathbf{b}/\varepsilon) = q_0$, podemos interpretar al estado q_1 como el estado donde llevamos una **a**, y permanecemos en éste esperando la **b**, entonces, las transiciones desde q_1 tienen que ser: $\delta(q_1, \mathbf{a}/\varepsilon) = q_1$ y $\delta(q_1, \mathbf{b}/\mathbf{a}) = q_0$ es ésta última transición la que agrega una **a** a la cadena de salida.

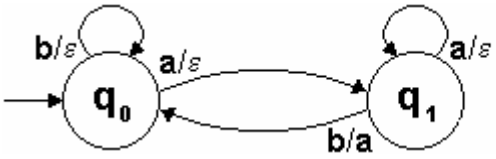


Figura 3.24

La figura 3.24 muestra la construcción de este autómata, con el que podemos comprobar que si la cadena de entrada es **aabab**, la salida será **aa**, tal como lo indica la secuencia de **DI**s correspondiente.

$$(q_0, \underline{\mathbf{a}}\mathbf{abab}, \varepsilon) \vdash (q_1, \underline{\mathbf{a}}\mathbf{bab}, \varepsilon) \vdash (q_1, \underline{\mathbf{b}}\mathbf{ab}, \mathbf{a}) \vdash (q_1, \underline{\mathbf{b}}, \mathbf{a}) \vdash (q_0, \underline{\varepsilon}, \mathbf{aa})$$

Preguntas

a) ¿Bajo qué condiciones se cumple que un **AFD** es mínimo?

b) ¿Para qué casos se cumple que $\varepsilon \in L(M)$, donde M es un **AFD**?

c) ¿Para qué casos se cumple que $L(M) = \varnothing$, donde M es un **AFD**?

d) Dado M, un **AFD** que acepta al lenguaje L(M), ¿Cómo es posible construir un **AFD**, a partir de M, que acepte todas cadenas que sean prefijos de la cadena *w* para todo $w \in L(M)$?

e) Dados M_1 y M_2, dos **AFD**s que aceptan los lenguajes $L(M_1)$ y $L(M_2)$, ¿Cómo es posible construir un **AFD** que acepte el lenguaje $L(M_1) \cap L(M_2)$?

f) Y ¿Cómo es posible construir un **AFD** que acepte el lenguaje $L(M_1) - L(M_2)$?

g) ¿Cómo es posible demostrar que un **AFD** que acepta el lenguaje universal?

h) ¿Qué significa que dos **AFD**s sean equivalentes?

i) ¿Que significa que dos Estados sean Equivalentes?

Ejercicios Capítulo 3

3.1 Construya el diagrama de transición del **AFD** a partir de la tabla 3.8:

δ	**0**	**1**
$\rightarrow$*q_0	q_2	q_1
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_3	q_1

Tabla 3.8

3.2 Para los siguientes ejercicios, construya el diagrama de transición del **AFD** que acepta a cada uno de los lenguajes sobre el alfabeto $\sum$ = { **a**, **b** }:

a) El lenguaje donde toda cadena tiene exactamente dos **b**s.

b) El lenguaje de las cadenas no vacías, donde toda **a** está entre dos **b**s.

c) El lenguaje donde toda cadena contiene el sufijo **aba**.

d) El lenguaje donde ninguna cadena contiene las subcadenas **aa** ni **bb**.

e) El lenguaje donde toda cadena contiene la subcadena **baba**.

f) El lenguaje donde toda cadena contiene por separado a las cadenas **ab** y **ba**.

g) Toda cadena es de longitud impar y contiene una cantidad par de **a**s.

3.3 Para los siguientes ejercicios, construya el diagrama de transición del **AFD** que acepta a cada uno de los lenguajes sobre el alfabeto $\sum$ = { **a**, **b** }:

a) El lenguaje descrito por la Expresión Regular: $\mathbf{a}^+ \cup \mathbf{ba}^*$.

b) El lenguaje descrito por la Expresión Regular: $\mathbf{a}^+\mathbf{ba}^+\mathbf{b}$.

c) El lenguaje descrito por la Expresión Regular: $\mathbf{b}^+\mathbf{a} \cup \mathbf{a}^*$.

d) El lenguaje descrito por la Expresión Regular: $\mathbf{ab}(\mathbf{a} \cup \mathbf{b})^*$.

e) El lenguaje descrito por la Expresión Regular: $(\mathbf{a}^+\mathbf{b} \cup \mathbf{b}^+\mathbf{a})^*$

f) El lenguaje descrito por la Expresión Regular: $\varepsilon \cup \mathbf{ab}^+\mathbf{a}$.

g) El lenguaje descrito por la Expresión Regular: $\varepsilon \cup \mathbf{ab}^+ \cup \mathbf{b}^*\mathbf{a}$.

3.4 Encontrar el **AFD** mínimo equivalente a los **AFD**s cuyos diagramas de transiciones se muestran en las siguientes figuras:

a)

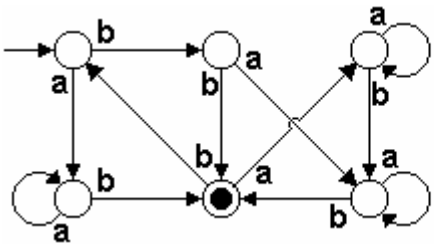


Figura 3.25

b)

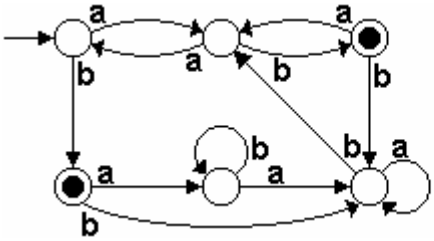


Figura 3.26

c)

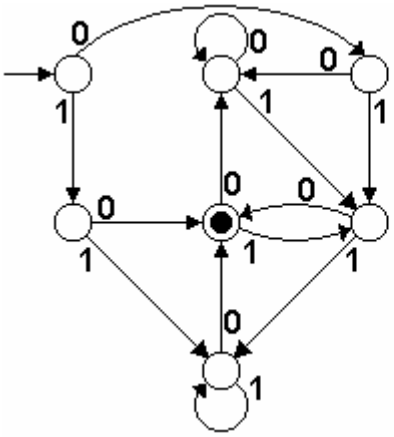


Figura 3.27

d)

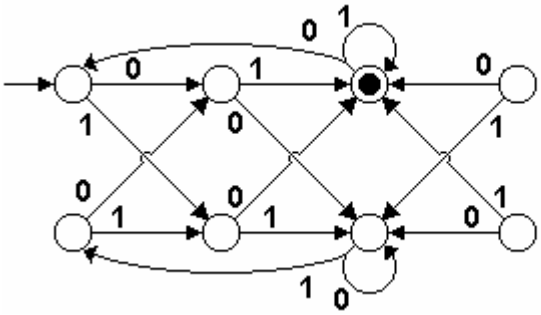


Figura 3.28

e)

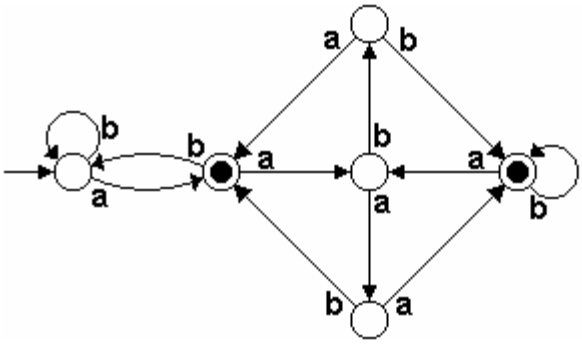


Figura 3.29

3.5 Determine si los **AFD**s mostrados en la Figura 3.30 son o no equivalentes.

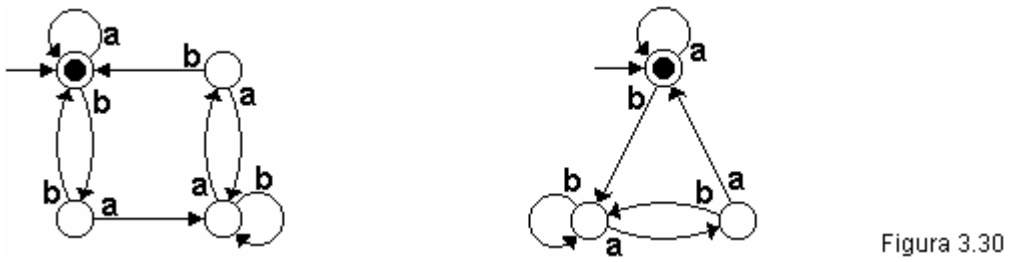


Figura 3.30

3.6 Determine si los **AFD**s mostrados en la Figura 3.31 son o no equivalentes.

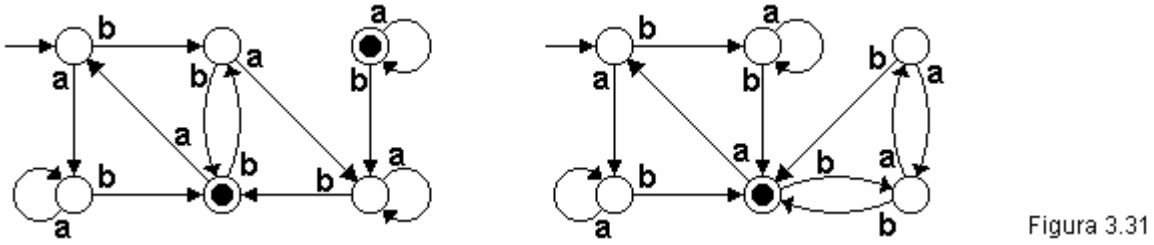


Figura 3.31

3.7 Determine si el **AFD** mostrado en la Figura 3.32 es equivalentes al mostrado en la Figura 3.20.

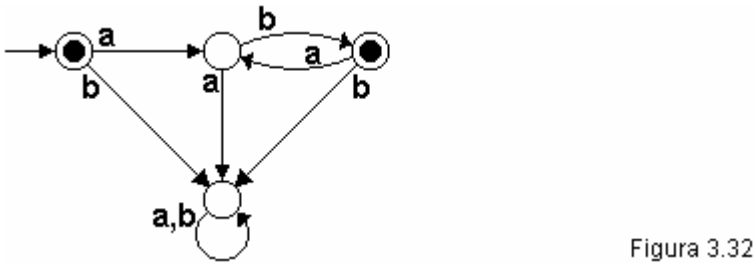


Figura 3.32

3.8 Construir una Máquina de Moore que permita calcular el módulo base 4 de un número en formato binario.

3.9 Construir una Máquina de Moore que permita calcular el módulo base 5 de un número en formato binario.

3.10 A partir de los resultados en los ejercicios 3.8 y 3.9 construya los **AFD**s que acepten los siguientes lenguajes:

a) L = { w $\in$ { **0**, **1**}* | w es un número binario múltiplo de 4 }

b) L = { w $\in$ { **0**, **1**}* | w es un número binario múltiplo de 5 }

3.11 Construir una Máquina de Mealy sobre el alfabeto $\sum$ = { **0**, **1** }, que genere una cadena de salida cuyo símbolo inicial sea un **0**, y que el i-ésimo símbolo sea una **0** si los símbolos i – 1 e i son ambos cero en la cadena de entrada, en caso contrario, el símbolo será un **1**.

3.12 Construya un *Transductor Determinista* sobre el alfabeto $\sum$ = { **0**, **1** }, que genere la cadena $\mathbf{1}^n$, donde n es el número de **ceros** que hay en la cadena *w*.

3.13 Construya un *Transductor Determinista* sobre el alfabeto $\sum$ = { **a**, **b** }, que genere la cadena $\mathbf{a}^n$, donde n es el número de ocurrencias de la subcadena **abb** en *w*.

Capítulo

4

Autómatas Finitos No Deterministas

Se define Autómata Finito No Determinista, se definen las transiciones épsilon y finalmente se define el Autómata Finito Generalizado. Se establece la metodología para encontrar Autómatas no deterministas sin transiciones épsilon y Deterministas equivalentes.

Definición

Autómata Finito No Determinista es aquel que puede tener cero, una o más transiciones distintas para el mismo símbolo de entrada y lo denotamos por **AFN**.

Todo **AFD** puede considerarse como un tipo particular de **AFN**, pero generalmente resulta mucho más sencillo construir e interpretar a un **AFN** que a un **AFD,** tal como se puede observar en los siguientes ejemplos.

Ejemplo 1

En el capítulo anterior habíamos construido un **AFD** para reconocimiento del lenguaje L = (**ab**)$^+$, como se muestra en la Figura 3.8, si excluimos el estado no deseado q_3 y las transiciones que acceden a él, obtenemos como resultado un **AFN** equivalente de apariencia mucho más simple:

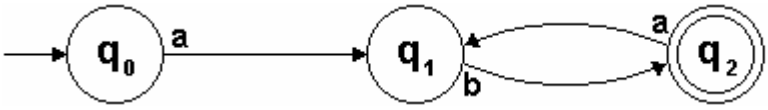


Figura 4.1

Se trata de un **AFN** que es considerado por muchos autores como un **AFD**, pues su comportamiento es absolutamente determinista, con la diferencia de que como carece de algunas transiciones, es posible que el análisis de alguna cadena se vea interrumpido. Si el autómata se detiene sin agotar la cadena, es equivalente a pasar a un estado no deseado, y por lo tanto, el no poder terminar de analizar la cadena significa que ésta no puede ser aceptada. Ésta es una de las grandes diferencias de los **AFN**s respecto de los **AFD**s, mientras que éstos siempre terminan de procesar una cadena dada, los primeros pueden enfrentar situaciones en los que no exista transición alguna y se detengan sin agotar la cadena, rechazándola.

Ejemplo 2

Consideremos ahora un lenguaje muy simple: L = **a** ∪ (**ab**)$^+$, construir el diagrama de transiciones de un **AFD** que acepte este lenguaje es laborioso, tal como se muestra en la siguiente figura:

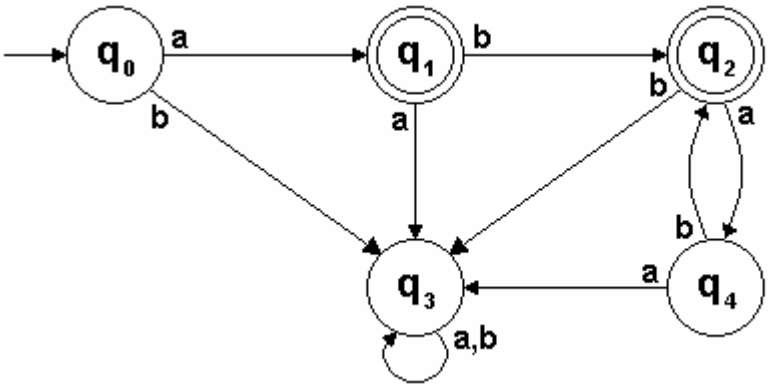


Figura 4.2

Sin embargo, resulta mucho más fácil construir e interpretar el diagrama de transiciones de un **AFN**, tal como se muestra en la siguiente figura:

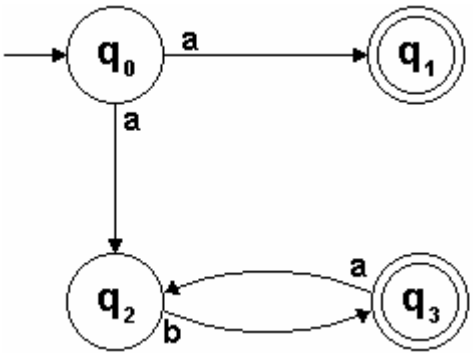


Figura 4.3

Ejemplo 3

Ahora considere al diagrama mostrado en la figura 4.4.

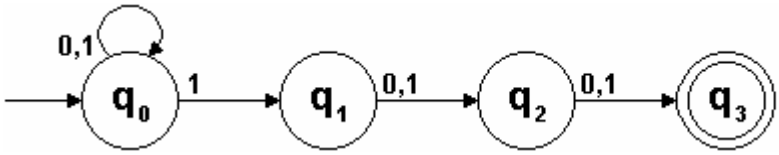


Figura 4.4

Éste diagrama representa a un **AFN** que acepta el lenguaje: L = (**0** ∪ **1**)*__1__(**0** ∪ **1**)^2, que es el lenguaje formado por las cadenas de ceros y unos cuyo antepenúltimo símbolo es un **1**. Por tratarse de un Autómata No Determinista, se puede ver que para el estado q_3 no existen transiciones definidas, mientras que para el estado q_0 hay dos transiciones para el símbolo **1**.

Compare el diagrama anterior con el correspondiente al **AFD** mínimo equivalente que se muestra en la figura 4.5 y saque sus propias conclusiones.

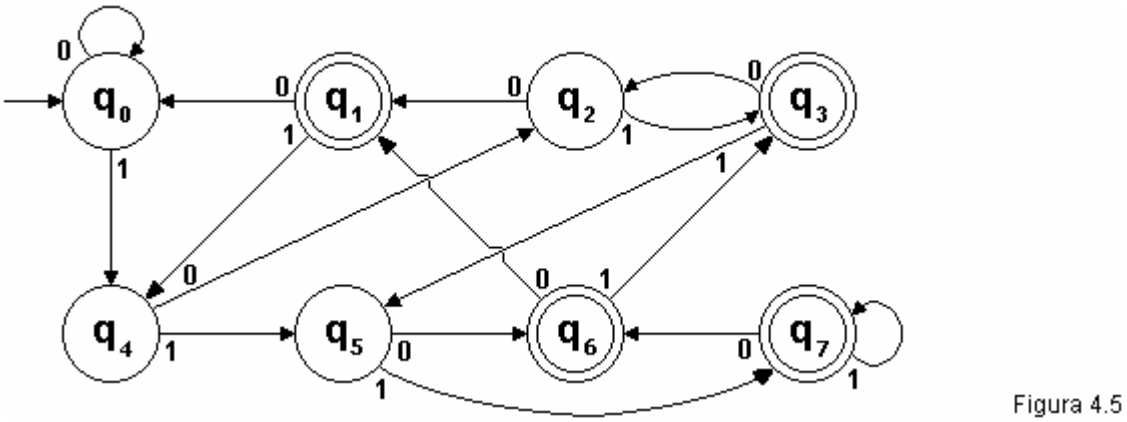


Figura 4.5

Definición formal de AFN

Formalmente se define a un **AFN** por la quíntupla M = (Q, $\sum$, Δ, s, F), donde: Q es un conjunto finito de estados, $\sum$ es el alfabeto de entrada, s $\in$ Q es el estado inicial, F es el subconjunto de Q de los estados de aceptación (F $\subseteq$ Q) y Δ es la función de transición (Δ: Q $\times$ $\sum$ $\rightarrow$ 2^Q), donde se puede apreciar que el contra-dominio de la función es el conjunto potencia de Q, es decir que esta función devuelve conjuntos de estados en vez de un solo estado.

El **AFN** de la Figura 4.4 está definido por: Q = { q_0, q_1, q_2, q_3 }, $\sum$ = { **0**, **1** }, F = { q_3 }, s = q_0 y Δ contiene a las transiciones siguientes:

$\Delta(q_0,\mathbf{0}) = \{q_0\}$ $\quad$ $\Delta(q_0,\mathbf{1}) = \{q_1, q_0\}$

$\Delta(q_1,\mathbf{0}) = \{q_2\}$ $\quad$ $\Delta(q_1,\mathbf{1}) = \{q_2\}$

$\Delta(q_2,\mathbf{0}) = \{q_3\}$ $\quad$ $\Delta(q_2,\mathbf{1}) = \{q_3\}$

O resumiendo, el **AFN** y todos sus elementos se muestran en la siguiente tabla, por lo que podemos afirmar que esta tabla representa completamente al autómata:

Δ	**0**	**1**
$\rightarrow q_0$	$\{q_0\}$	$\{ q_1, q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
$^*q_3$	$\varnothing$	$\varnothing$

Tabla 4.1

Ejemplo 4

El diagrama de transiciones de la Figura 4.6 representa a un **AFN**.

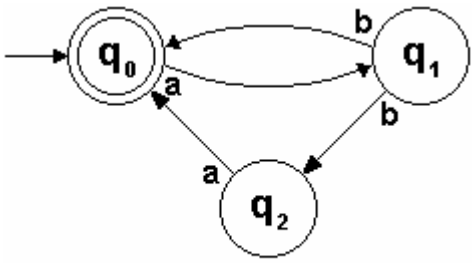


Figura 4.6

Los elementos de dicho **AFN** están contenidos en la tabla de transiciones 4.2:

Δ	**a**	**b**
$\rightarrow$*q_0	$\{q_1\}$	$\varnothing$
q_1	$\varnothing$	$\{q_0,q_2\}$
q_2	$\{q_0\}$	$\varnothing$

Tabla 4.2

El lenguaje que acepta este **AFN** está dado por la expresión (**ab** $\cup$ **aba**)*, esto se visualiza fácilmente analizando las dos trayectorias cíclicas que tenemos para regresar al estado q_0, pasando por el estado q_1.

Para analizar una cadena, se elegirá la opción que más convenga, si se trata de la cadena w_1 = **aba**, pasamos por el estado q_2, pero si la cadena es w_2 = **abab**, entonces seleccionamos la transición que va directamente de q_1 a q_0.

Analicemos este segundo caso, utilizando la función de transición tenemos:

- $\Delta(q_0,\mathbf{a}) = \{q_1\}$
- $\Delta(\{q_1\},\mathbf{b}) = \Delta(q_1,\mathbf{b}) = \{q_0, q_2\}$
- $\Delta(\{q_0,q_2\},\mathbf{a}) = \Delta(\{q_0\},\mathbf{a}) \cup \Delta(\{q_2\},\mathbf{a}) = \Delta(q_0,\mathbf{a}) \cup \Delta(q_2,\mathbf{a}) = \{q_1\} \cup \{q_0\} = \{q_0, q_1\}$
- $\Delta(\{q_0,q_1\},\mathbf{b}) = \Delta(\{q_0\},\mathbf{b}) \cup \Delta(\{q_1\},\mathbf{b}) = \Delta(q_0,\mathbf{b}) \cup \Delta(q_1,\mathbf{b}) = \varnothing \cup \{q_0, q_2\} = \{q_0, q_2\}$.

De esta manera, para simplificar la secuencia de transiciones, resulta más fácil escribir de modo resumido: $\Delta(\{q_0\},\mathbf{abab}) = \{q_0, q_2\}$, la cadena analizada es aceptada debido a que en el conjunto final aparece q_0 que es un estado de aceptación.

Entonces, observamos que es posible extender el dominio de la función de transición original por el siguiente: $\Delta: 2^Q \times \Sigma^+ \rightarrow 2^Q$.

Ejemplo 5

En el diagrama de la Figura 4.7, tenemos que el estado q_0 tiene dos transiciones posibles para el símbolo **0** y otras dos para el símbolo **1**, esto nos da la libertad de elegir la que convenga a una secuencia dada, con el fin de poder llegar al estado de aceptación. No importa si una alternativa no nos conduce al estado de aceptación, mientras exista una que si nos lleve a tal estado.

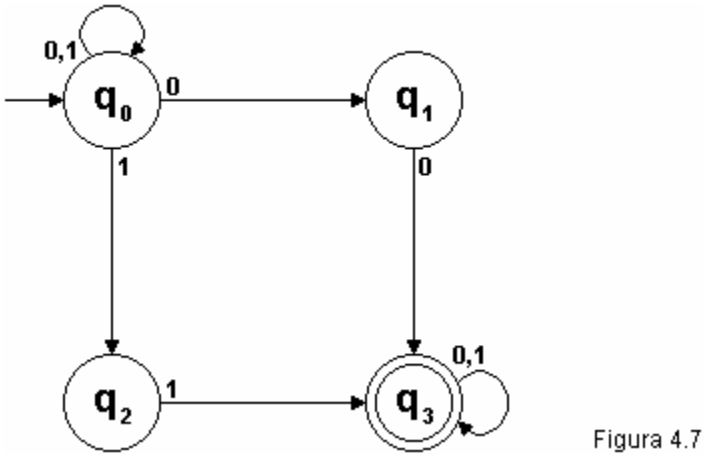


Figura 4.7

Este **AFN** acepta cualquier cadena con un par de ceros consecutivos o con un par de unos consecutivos.

La cadena w = **01001** presenta el siguiente árbol de posibles transiciones, nos basta con que una de cuyas trayectorias nos conduzca al estado de aceptación q_3.

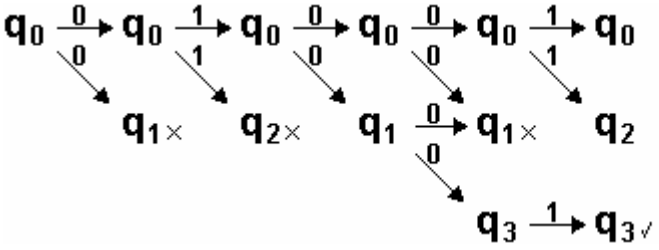


Figura 4.8

Autómatas Equivalentes

Dado que los **AFN**s son más versátiles, es mucho más frecuente encontrar que algunos de ellos son equivalentes y en la mayoría de los casos no es fácil determinar cual de ellos es mejor. En general, se considera que un **AFN** es mejor a otro, si está más cercano al comportamiento determinista, sin importar que tenga más estados, esto se explica porque es más fácil analizar las cadenas con ellos, de hecho, siempre nos convendrá encontrar un **AFD** equivalente para el análisis de cadenas.

Ejemplo

Sean M_1 y M_2 dos **AFN**s sobre el alfabeto Σ = { **a**, **b** } que se muestran en la Figura 4.9. Se puede verificar fácilmente que son equivalentes, porque ambos aceptan el mismo lenguaje $\mathbf{a}^+\mathbf{b}^+$, pero definitivamente M_2 es mejor. ¿Por qué? Porque su comportamiento es determinista, lo que permite analizar fácilmente cualquier cadena, mientras que el no-determinismo de M_1 complica el análisis de cualquier cadena.

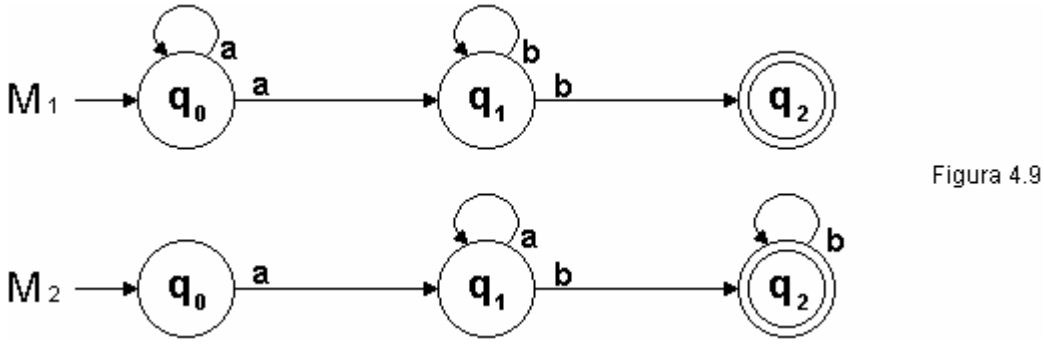


Figura 4.9

Equivalencia entre AFN y AFD

Aunque podemos considerar a un **AFD** dado como un caso particular de los **AFN**s y sabemos que éstos últimos son más versátiles, esto no significa que los **AFN** sean más poderosos que los **AFD**s, esto se confirma demostrando que para cualquier **AFN**, siempre existe un **AFD** equivalente que acepta el mismo lenguaje.

Ejemplo 1

Encontrar un **AFD** equivalente al **AFN** definido por el diagrama de transiciones mostrado en la figura siguiente:

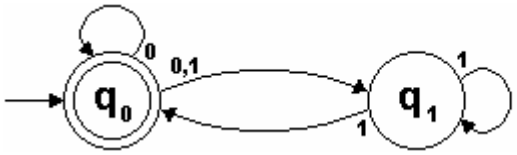


Figura 4.10

Es conveniente, primero, representar al **AFN** por medio de su tabla de transiciones correspondiente, tal como se muestra a continuación:

Δ	**0**	**1**
$\rightarrow$*q_0	{ q_0,q_1}	{q_1}
q_1	$\varnothing$	{q_0,q_1}

Tabla 4.3

Ahora debemos definir a M', el **AFD** equivalente a M, como sigue: cada uno de los posibles estados de M' corresponderá a una combinación de estados de M, de tal forma que Q' = { $[q_0]$, $[q_1]$, $[q_0,q_1]$, $[\varnothing]$ }, donde el estado inicial es s' = $[q_0]$, y los estados de aceptación son todos aquellos que contengan a q_0, que es el estado de aceptación de M, es decir: F' = { $[q_0]$, $[q_0,q_1]$ }. De esta manera, δ queda definido por las transiciones: $\delta([q_0], \mathbf{0}) = [q_0,q_1]$, $\delta([q_0], \mathbf{1}) = [q_1]$, $\delta([q_1], \mathbf{0}) = [\varnothing]$ y $\delta([q_1], \mathbf{1}) = [q_0,q_1]$, las cuales se obtienen directamente de la tabla 4.3.

Además, el estado $[\varnothing]$ corresponde a un estado no deseado, por lo que sus transiciones permanecerán en este estado: $\delta([\varnothing], \mathbf{0}) = [\varnothing]$ y $\delta([\varnothing], \mathbf{1}) = [\varnothing]$.

Finalmente, para determinar las transiciones del estado $[q_0,q_1]$, tenemos que observar que se derivan de las siguientes transiciones en M:

- $\Delta(\{q_0, q_1\}, \mathbf{0}) = \Delta(\{q_0\}, \mathbf{0}) \cup \Delta(\{q_1\}, \mathbf{0}) = \{q_0,q_1\} \cup \varnothing = \{q_0, q_1\}$
- $\Delta(\{q_0, q_1\}, \mathbf{1}) = \Delta(\{q_0\}, \mathbf{1}) \cup \Delta(\{q_1\}, \mathbf{1}) = \{q_1\} \cup \{q_0,q_1\} = \{q_0, q_1\}$

Dando como resultado las transiciones para el estado $[q_0,q_1]$: $\delta([q_0,q_1], \mathbf{0}) = [q_0,q_1]$ y $\delta([q_0,q_1], \mathbf{0}) = [q_0,q_1]$, todo lo anterior se resume en la tabla 4.4.

Aunque la nomenclatura suele parecer, al principio, algo confusa, como último paso, se sugiere renombrar a cada uno de los estados de M' con nombres más simples, por ejemplo, se pueden renombrar así: $p_0 = [q_0]$, $p_1 = [q_1]$, $p_2 = [q_0,q_1]$ y $p_3 = [\varnothing]$, tal como se muestra el resultado en la Tabla 4.5.

δ	**0**	**1**
$\rightarrow$*$[q_0]$	$[q_0,q_1]$	$[q_1]$
$[q_1]$	$[\varnothing]$	$[q_0,q_1]$
*$[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_1]$
$[\varnothing]$	$[\varnothing]$	$[\varnothing]$

Tabla 4.4

	0	**1**
$\rightarrow$*p_0	p_2	p_1
p_1	p_3	p_2
*p_2	p_2	p_2
p_3	p_3	p_3

Tabla 4.5

De esta forma, el diagrama de transiciones del **AFD** equivalente es el mostrado en la figura 4.11. Haciendo un análisis del mismo, podemos ver que este autómata acepta todas las cadenas que tenga unos y ceros, con excepción de *w* = **1** y de todas las cadenas que inicien con el prefijo **10**.

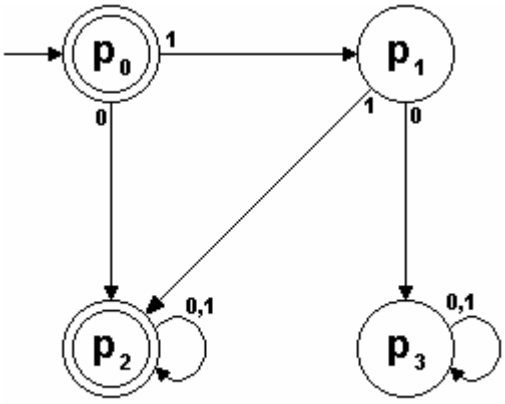


Figura 4.11

Ejemplo 2

Encontrar el **AFD** equivalente del **AFN** visto anteriormente en la figura 4.3 y que acepta el lenguaje **a** ∪ (**ab**)$^+$. Para ello, primero obtenemos su tabla de transiciones:

Δ	**a**	**b**
→q_0	{q_1,q_2}	∅
*q_1	∅	∅
q_2	∅	{q_3}
*q_3	{q_2}	∅

Tabla 4.6

Para obtener la tabla de transiciones del **AFD** equivalente, convertimos solamente la primera fila de la tabla del **AFN**, correspondiente al estado inicial [q_0], posteriormente agregamos a la tabla las filas correspondientes a los estados [q_1,q_2] y [∅], que son los estados hacia donde se dirigen las transiciones de [q_0]. Al poner las transiciones del estado [q_1,q_2], aparece como destino el estado [q_3], que corresponde a la cuarta fila que se agrega a la tabla. Finalmente, al hacer esto, surge como destino el estado [q_2], que será nuestra última fila a considerar, debido a que no surgen nuevos estados:

δ	**a**	**b**
→[q_0]	[q_1,q_2]	[∅]
*[q_1,q_2]	[∅]	[q_3]
[∅]	[∅]	[∅]
*[q_3]	[q_2]	[∅]
[q_2]	[∅]	[q_3]

Tabla 4.7

Entonces el **AFD** equivalente está dado por: Q' = { $[q_0]$, $[q_1,q_2]$, $[q_2]$, $[q_3]$, $[\varnothing]$ }, s' = $[q_0]$ y F' = { $[q_3]$, $[q_1,q_2]$ }, además de las transiciones de la tabla anterior, es fácil verificar que éste es el mismo **AFD** cuyo diagrama de transiciones aparece en la figura 4.2.

Observe que en la Tabla 4.7 no aparece el estado $[q_1]$, ya que éste no es accesible desde $[q_0]$. La forma en la que se construyó esta tabla permite que se obtenga un **AFD** conexo, ya que se excluyen los estados aislados que aparecerían por otros métodos, sin embargo, el procedimiento no nos garantiza obtener al **AFD** mínimo equivalente.

Transiciones épsilon

Las transiciones épsilon son aquéllas que no dependen de ninguna entrada, ni consumen ningún símbolo para efectuarse y se denotan con el símbolo ε.

Ejemplo 1

Los dos Autómatas M_1 y M_2 de la figura siguiente, son equivalentes, pero M_1 emplea una transición ε para acceder al estado de aceptación, de manera no determinista, mientras que M_2 es un **AFD**:

Figura 4.12

Ejemplo 2

El siguiente **AFN** con transiciones épsilon acepta cualquier cantidad de **0**s o ninguna, seguida de cualquier cantidad de **1**s y luego cualquier cantidad de **2**s. Es decir acepta el lenguaje dado por la expresión regular: **0*1*2***.

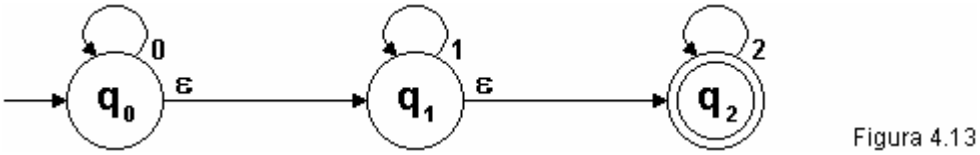


Figura 4.13

La función de transición *d*, de un **AFN** con transiciones épsilon se define de la siguiente manera: d:$Q \times (\sum \cup \varepsilon) \rightarrow 2^Q$, que como podemos ver, es ligeramente distinta a la de la función Δ de los **AFN**s sin transiciones épsilon.

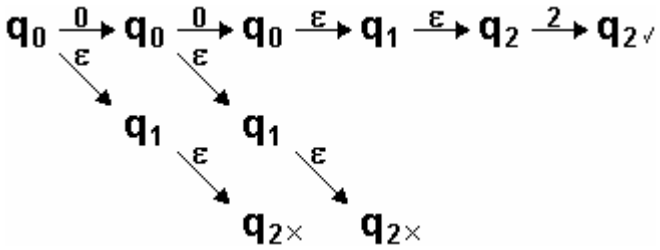
La tabla de transiciones del **AFN** anterior se muestra a continuación, en ella se incluye una columna adicional para las transiciones épsilon, como si fuera un símbolo más:

d	**0**	**1**	**2**	ε
→q_0	{q_0}	∅	∅	{q_1}
q_1	∅	{q_1}	∅	{q_2}
*q_2	∅	∅	{q_2}	∅

Tabla 4.8

De hecho, de forma semejante a lo que hicimos con las funciones de transición anteriores, podemos extender el dominio de esta función así $d:2^Q \times \sum^* \to 2^Q$.

La cadena **002** es aceptada por este **AFN**, porque equivale a **00εε2** y la secuencia de transiciones para este caso sería:



Ejemplo 3

Ahora consideremos el **AFN** M_1, mostrado en la Figura 4.14, que acepta al lenguaje (**ab** ∪ **aba**)*, y el cual es equivalente a M_2, en la Figura 4.15, que es un **AFN** sin transiciones épsilon.

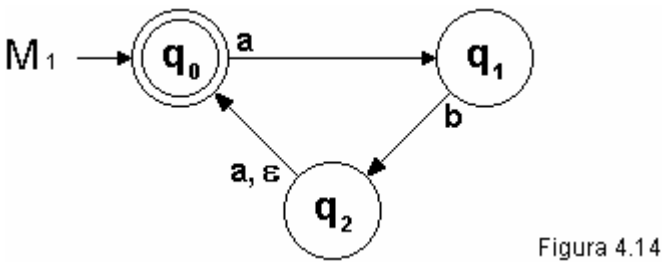


Figura 4.14

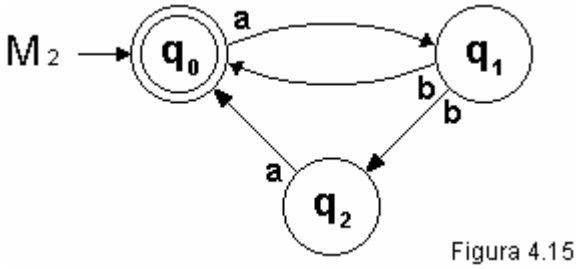


Figura 4.15

Equivalencia entre AFNs con y sin transiciones ε

A continuación veremos una metodología para encontrar un **AFN** equivalente que no tenga transiciones épsilon, para ello, primero debemos definir la *Cerradura épsilon* de un estado q_i de la siguiente manera:

$$C(q_m) = \{ q_n \in Q \mid q_n \text{ es accesible desde } q_m \text{ sin consumir ningún símbolo}\}$$

Por definición q_m siempre está contenido en su propia cerradura $C(q_m)$, ya que no necesita de ningún símbolo para acceder a sí mismo.

Las transiciones del **AFN** equivalente sin transiciones épsilon se definen utilizando la fórmula que sigue, en donde se aplica dos veces la cerradura épsilon, antes y después de cada transición:

$$\Delta(q_m, \sigma) = C(d(C(q_m), \sigma))$$

Ejemplo 1

Considere el **AFN** cuyo diagrama de transiciones se muestra en la Figura 4.16, y que podemos representar por medio de la Tabla 4.9 de la derecha:

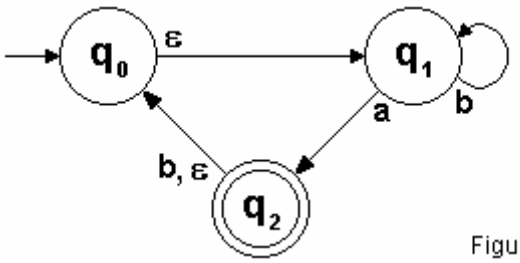


Figura 4.16

d	**a**	**b**	ε
→q_0	∅	∅	$\{q_1\}$
q_1	$\{q_2\}$	$\{q_1\}$	∅
*q_2	∅	$\{q_0\}$	$\{q_0\}$

Tabla 4.9

Podemos aplicar la fórmula de la cerradura desde la tabla o analizando el diagrama, de cualquier forma, se deben obtener las siguientes cerraduras épsilon:

- $C(q_0) = \{q_0,q_1\}$
- $C(q_1) = \{q_1\}$
- $C(q_2) = \{q_0,q_1,q_2\}$

Las transiciones para construir el **AFN** sin transiciones épsilon se obtienen así:

- $\Delta(q_0, \mathbf{a}) = C(d(C(q_0), \mathbf{a})) = C(d(\{q_0,q_1\}, \mathbf{a})) = C(d(q_0, \mathbf{a}) \cup d(q_1, \mathbf{a})) = C(\varnothing \cup \{q_2\}) = C(q_2) = \{q_0,q_1,q_2\}$
- $\Delta(q_0, \mathbf{b}) = C(d(C(q_0), \mathbf{b})) = C(d(\{q_0,q_1\}, \mathbf{b})) = C(d(q_0, \mathbf{b}) \cup d(q_1, \mathbf{b})) = C(\varnothing \cup \{q_1\}) = C(q_1) = \{q_1\}$
- $\Delta(q_1, \mathbf{a}) = C(d(C(q_1), \mathbf{a})) = C(d(q_1, \mathbf{a})) = C(q_2) = \{q_0,q_1,q_2\}$
- $\Delta(q_1, \mathbf{b}) = C(d(C(q_1), \mathbf{b})) = C(d(q_1, \mathbf{b})) = C(q_1) = \{q_1\}$
- $\Delta(q_2, \mathbf{a}) = C(d(C(q_2), \mathbf{a})) = C(d(\{q_0,q_1,q_2\}, \mathbf{a})) = C(d(q_0, \mathbf{a}) \cup d(q_1, \mathbf{a}) \cup d(q_2, \mathbf{a})) = C(\varnothing \cup \{q_2\} \cup \varnothing) = C(q_2) = \{q_0,q_1,q_2\}$
- $\Delta(q_2, \mathbf{b}) = C(d(C(q_2), \mathbf{b})) = C(d(\{q_0,q_1,q_2\}, \mathbf{b})) = C(d(q_0, \mathbf{b}) \cup d(q_1, \mathbf{b}) \cup d(q_2, \mathbf{b})) = C(\varnothing \cup \{q_1\} \cup \{q_0\}) = C(q_1) \cup C(q_0) = \{q_1\} \cup \{q_0,q_1\} = \{q_0,q_1\}$

Resumiendo, las transiciones obtenidas se muestran en la tabla 4.10:

Δ	a	b
→q_0	{q_0,q_1,q_2}	{q_1}
q_1	{q_0,q_1,q_2}	{q_1}
*q_2	{q_0,q_1,q_2}	{q_0,q_1}

Tabla 4.10

Se deja como ejercicio construir el diagrama de transiciones, así como encontrar el **AFD** mínimo equivalente.

Ejemplo 2

Encontrar el **AFN** sin transiciones épsilon equivalente al **AFN** cuyo diagrama de transiciones se muestra en la figura 4.17:

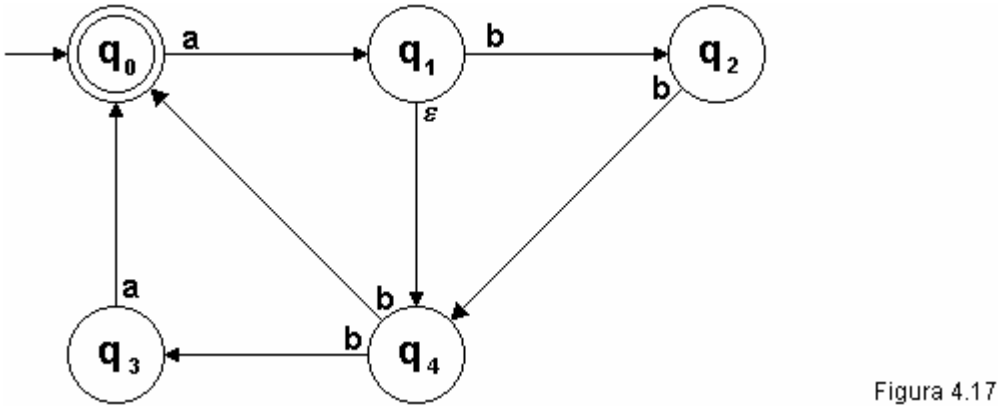


Figura 4.17

Del diagrama anterior podemos obtener las cerraduras épsilon respectivas, la única que no resulta ser una identidad es la correspondiente al estado q_1: $C(q_1) = \{q_1,q_4\}$.

La función Δ del **AFN** sin transiciones épsilon se obtiene con la fórmula antes vista y sus resultados se muestran en la tabla 4.11, siguiente:

Δ	a	b
→*q_0	{q_1, q_4}	∅
q_1	∅	{q_0, q_2, q_3}
q_2	∅	{q_4}
q_3	{q_0}	∅
q_4	∅	{q_0, q_3}

Tabla 4.11

La gráfica del diagrama de transiciones correspondiente se muestra en la figura 4.18 a continuación, obsérvese que todas las transiciones no-épsilon del **AFN** original se

preservan y solamente surgen las nuevas transiciones que se agregan a cambio de las transiciones épsilon que se eliminaron, con objeto de que el **AFN** obtenido sea equivalente.

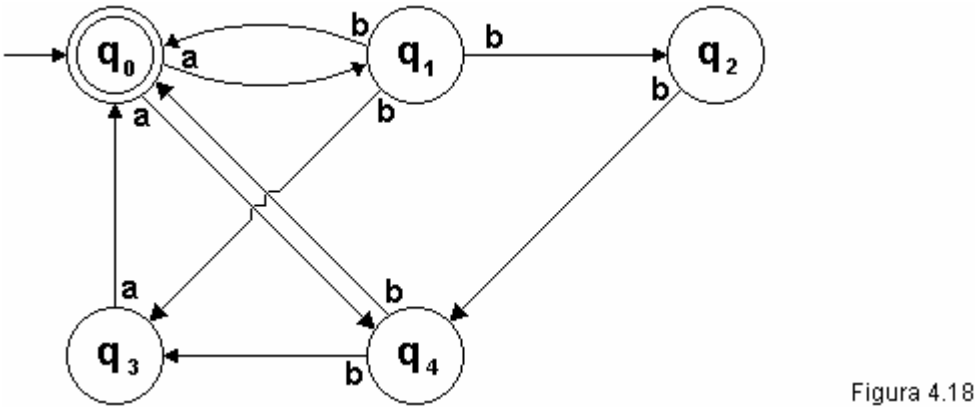


Figura 4.18

En algunas ocasiones, surgen transiciones nuevas que resultan redundantes y en otras ocasiones podemos obtener, como resultado, que algunos estados se queden aislados o bien, que se conviertan en estados no deseados, para reducir estos inconvenientes, recomendamos que antes de aplicar el procedimiento descrito, se empleen los siguientes criterios, cuando sea posible.

Criterios de Simplificación

Criterio 1

Si la única entrada de un estado q_k, sea o no de aceptación, pero que no es el estado inicial, es una transición épsilon, proveniente desde un estado q_n cualquiera, entonces q_k se fusiona a q_n.

Ejemplo

En el siguiente ejemplo se muestra como se aplica el criterio, fusionando el estado q_1 al estado q_0 y el estado q_3 al estado q_4, quedando un **AFN** sin transiciones épsion.

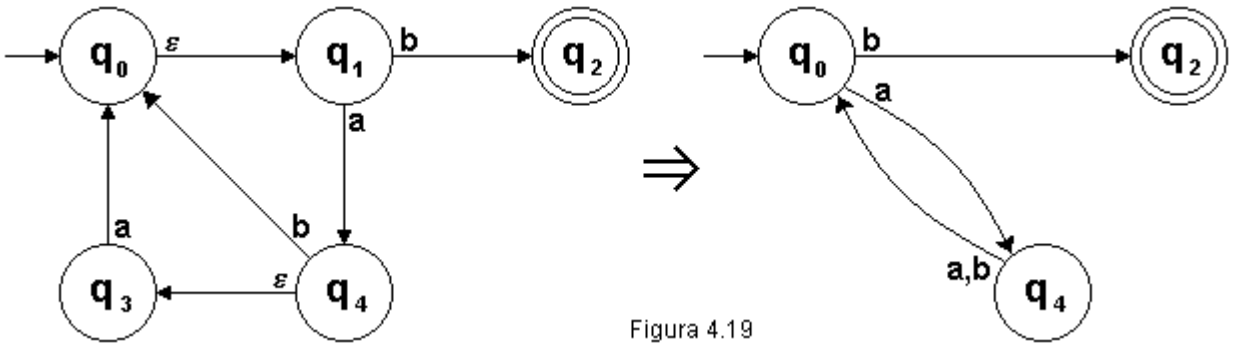


Figura 4.19

Criterio 2

Si la única salida de un estado q_k, que no es de aceptación, es una transición épsilon, hacia un estado q_n cualquiera, entonces q_k se fusiona a q_n, este criterio también es válido cuando ambos estados, q_k y q_n, son de aceptación.

Ejemplo

En el siguiente ejemplo se muestra como se aplica el segundo criterio, fusionando el estado q_2 al estado q_1, quedando un **AFN** sin transiciones épsion.

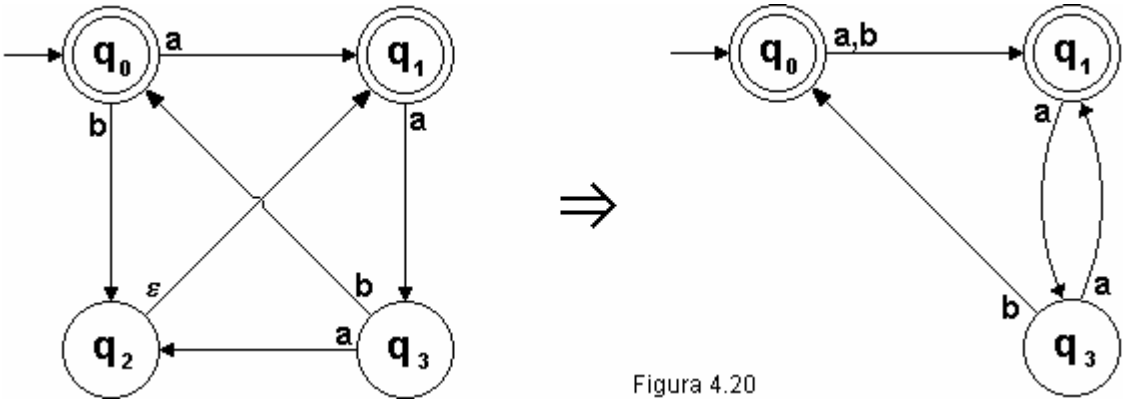


Figura 4.20

Criterio 3

Si un estado de aceptación q_k tiene varias entradas, una de las cuales es una transición épsilon desde el estado q_n, y no tiene salidas de ningún tipo, entonces q_k se subdivide en dos estados de aceptación equivalentes q_k' y q_k", el primero tendrá a la transición épsilon proveniente de q_n como única entrada y el otro a las demás entradas, con objeto de poder aplicar posteriormente el primer criterio y fusionar q_k' a q_n, convirtiéndolo en estado de aceptación.

El resultado final se puede resumir en que la transición épsilon se elimina a cambio de convertir al estado q_n en un estado de aceptación, como se muestra en la Figura 4.21:

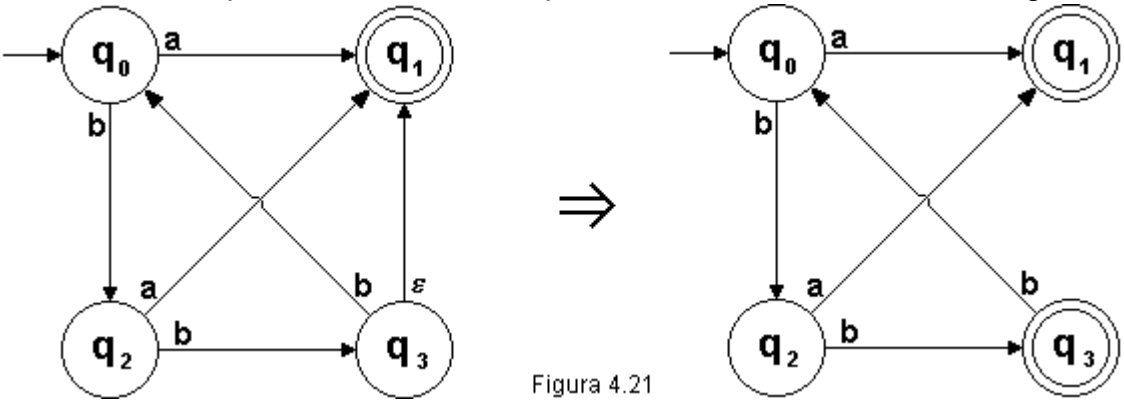


Figura 4.21

Ejemplo 1

Considere el **AFN** cuyo diagrama de transiciones se muestra en la figura 4.22:

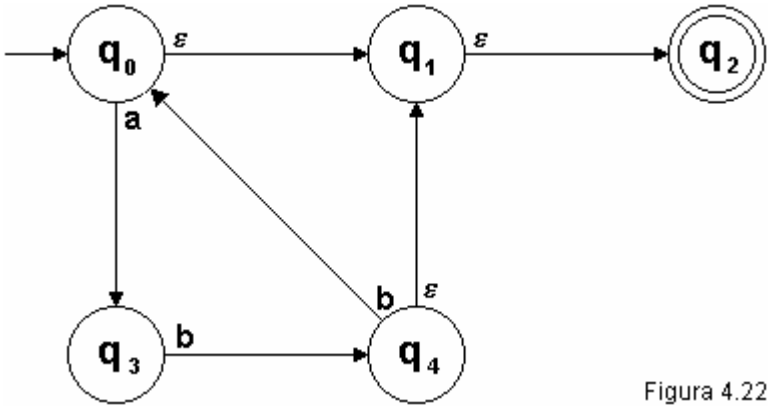


Figura 4.22

Aplicando el criterio 1, fusionamos al estado q_2 con el estado q_1. Posteriormente aplicamos el criterio 3 en q_1, que ahora ya es un estado de aceptación y que no tiene salidas, transformando a los estados q_0 y q_4 en estados de aceptación, q_1 se elimna al quedar aislado cuando se descarta la transición épsilon, resultando el siguiente **AFN** sin transiciones épsilon:

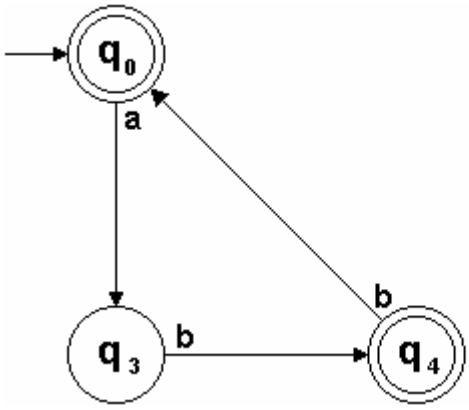


Figura 4.23

Ejemplo 2

Utilice los criterios anteriores para eliminar las transiciones épsilon del **AFN** cuyo diagrama de transiciones se muestra en la Figura 4.24.

Antes de poder aplicar los criterios, veamos que es posible eliminar a los estados q_4 y a q_5, puesto que se trata de estados no deseados. Hacer esto nos abre la posibilidad de aplicar el criterio 2, para fusionar al estado q_3 con el estado q_1, quedando el **AFN** con una transición épsilon, pero que ya no es posible simplificar por medio de los critrerios, y es el que se muestra en la Figura 4.25:

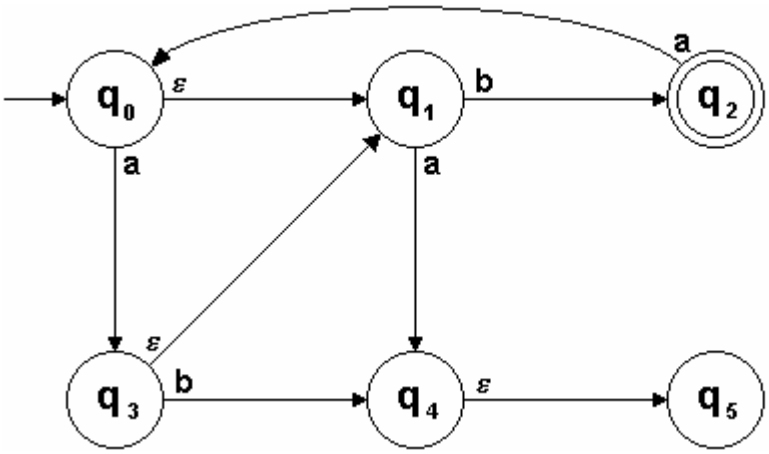


Figura 4.24

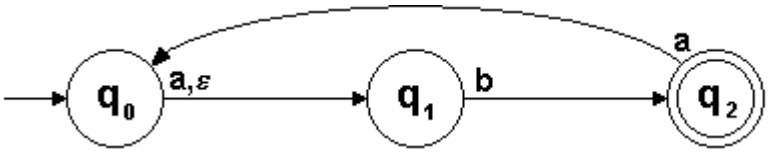


Figura 4.25

Y aplicando el procedimiento antes visto, para eliminar la transición épsilon restante, obtenemos finalmente el **AFN** mostrado en la figura 4.26:

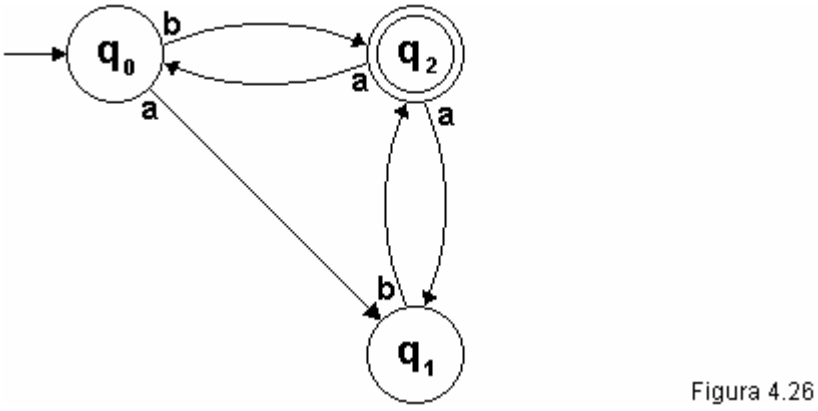


Figura 4.26

Autómatas Finitos Generalizados

En la página 53 se mencionó que la función de transición de los **AFN**s con transiciones épsilon se podía generalizar de la siguiente manera: $d:2^Q \times \Sigma^* \rightarrow 2^Q$. Esto nos permite suponer que también es posible construir autómatas finitos en los que cada transición esté definida por una cadena de símbolos y no por uno solo nada más.

A este tipo de autómatas les denominamos Autómatas Finitos No Deterministas Generalizados (**AFNG**).

Un **AFNG** puede tener cada transición etiquetada por una cadena, e incluso por una expresión regular más compleja, por tanto, debe poder leer cualquier cantidad de símbolos antes de efectuar cada transición.

La figura 4.27 muestra un **AFNG** que acepta el lenguaje regular **ab** (**bab**)* **ba**,

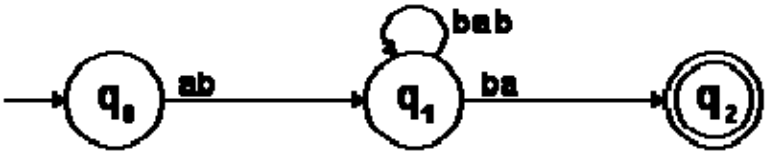


Figura 4.27

Ejemplo Práctico

Hay una Fábula que nos cuenta de un hombre que vivía a la orilla de un río y tenía que atravesarlo para ir a la ciudad. En una ocasión debía llevar un lobo, un borrego y un repollo, para cruzar el río poseía un pequeño bote donde solamente cabe él con uno de los animales o con el repollo, por lo que debía cruzar varias veces el río para pasar a todos ellos al otro lado. Sin embargo, no podía dejar el lobo con el borrego porque se lo comería, por lo mismo, tampoco podía dejar al borrego con el repollo. El objetivo consiste en representar un diagrama de estados que nos permita saber como puede el hombre finalmente cruzar el río sin que el repollo o el borrego sean devorados.

Los posibles estados se denotan como las combinaciones de las cuatro letras separadas por un guión, indicando de cual lado del río se encuentra cada personaje, de este modo tenemos:

Estado inicial: [BRHL- ∅] (cuando los cuatro están del lado donde está la casa)

Estado final: [∅ - BRHL] (cuando los cuatro pasaron al otro lado del río)

Algunos estados intermedios válidos son: [RL – BH], [RHL – B], [L – BRH], etc.

Los estados inaceptables serían: [RH – BL], [BR – HL], [BL – RH] y [HL –BR], porque nos conducirían a estados no deseados como: [RH – L], [B – HL], etc.

A continuación se presenta el diagrama de transiciones correspondiente, sin mostrar los estados inaceptables o no deseados, a fin de darle mayor claridad al mismo, Los símbolos del sistema son: **h** – cuando el hombre viaja solo, **l** – cuando viaja con el lobo, **b** – cuando viaja con el borrego y **r** – cuando viaja con el repollo.

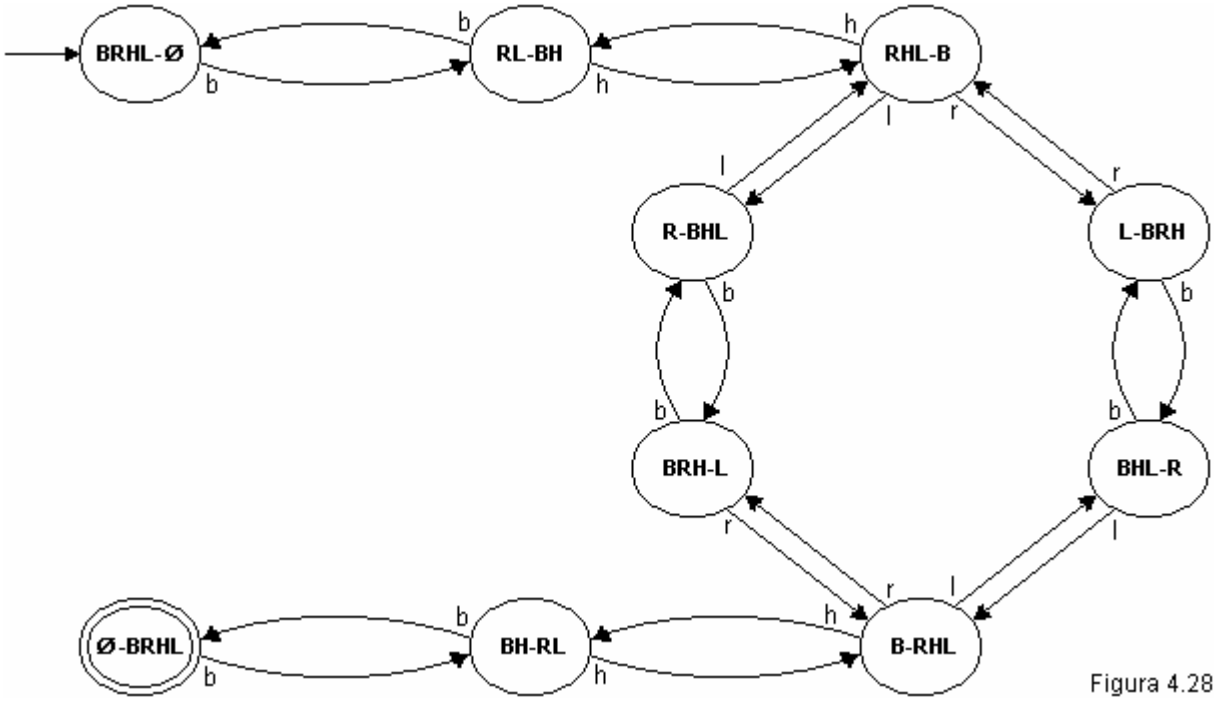


Figura 4.28

Como puede observarse, existen dos soluciones mínimas, y podemos denotarlas como: **bhlbrhb** y **bhrblhb**.

Preguntas

a) Dado M, un **AFD** que acepta al lenguaje L(M), ¿Cómo es posible construir un **AFN**, a partir de M, que acepte todas cadenas que sean sufijos de *w* para todo $w \in$ L(M)?

b) ¿Para qué casos se cumple que un **AFN** con transiciones épsilon acepta la cadena vacía?

c) ¿Existen ventajas o desventajas al emplear **AFN**s en vez de **AFD**s en el reconocimiento de lenguajes?

d) ¿Es más sencillo construir un **AFD** que un **AFN** a partir de una **ER** dada?

Ejercicios Capítulo 4

4.1 Construir un **AFN** que acepte cada uno de los lenguajes regulares siguientes, donde su alfabeto es $\sum$ = { **0**, **1** }:

a) Las cadenas que tienen solamente dos ceros, los cuales están separados por una cadena de unos de longitud múltiplo de 4.

b) Las cadenas en las que el quinto símbolo contado desde el final sea un **1**.

c) Las cadenas que contengan a la secuencia **101**.

d) El lenguaje descrito por la expresión regular (**00**)*(**11**)*

4.2 Construir el **AFN** que acepte el lenguaje (**a** ∪ **b**)* **aba**, encontrar el **AFD** mínimo equivalente.

4.3 Determine si los **AF**s dados en cada uno de los incisos de la siguiente figura, son o no son equivalentes:

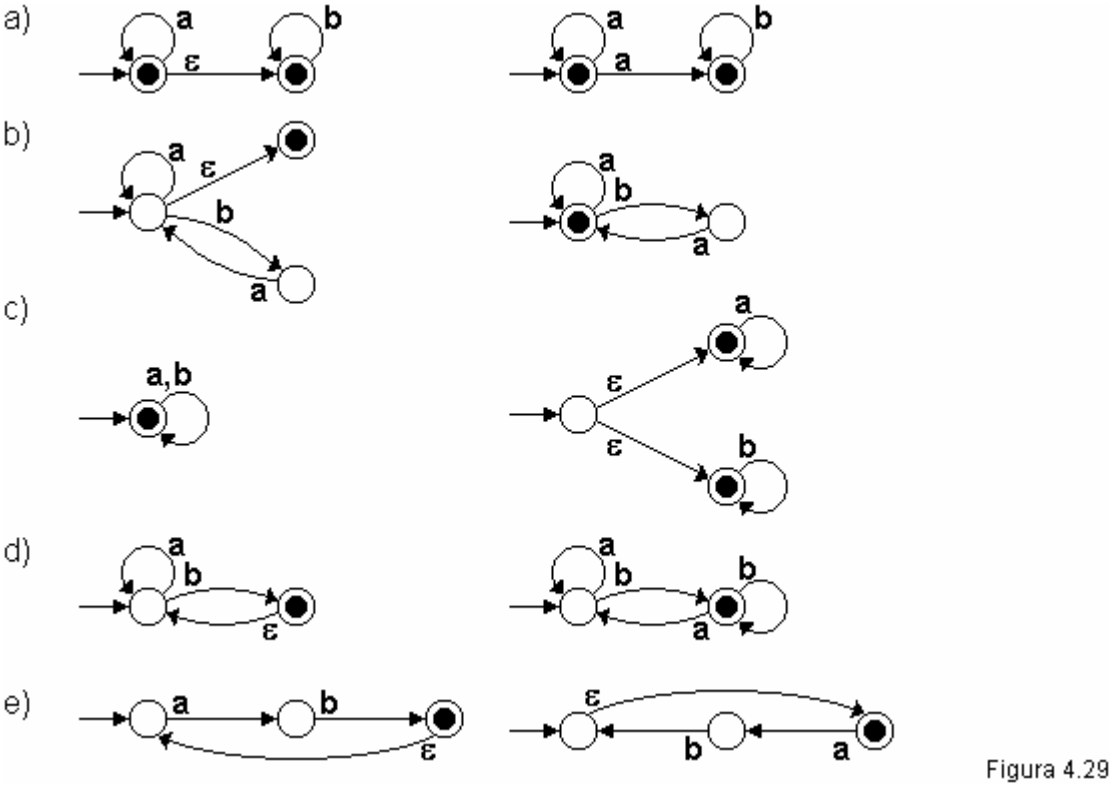


Figura 4.29

4.4 Encontrar el **AFD** mínimo equivalente a cada uno de los **AFN**s siguientes:

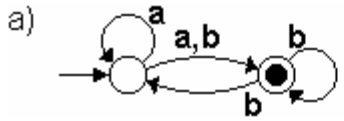


Figura 4.30

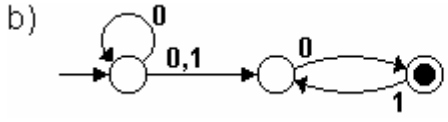


Figura 4.31

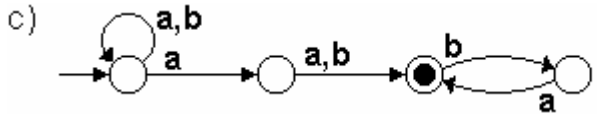


Figura 4.32

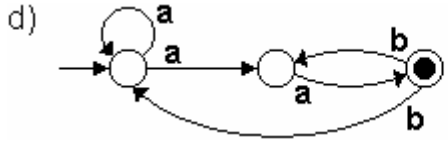


Figura 4.33

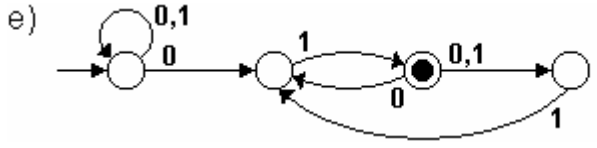


Figura 4.34

4.5 Encontrar el **AFD** mínimo equivalente para los siguientes casos:

a) Al **AFN** representado por la Tabla 4.10

b) Al **AFN** mostrado en la Figura 4.23

c) Al **AFN** mostrado en la Figura 4.26

d) Al **AFN** mostrado en la Figura 4.18

4.6 Encontrar el **AFN** sin transiciones épsilon equivalente al mostrado en la Figura 4.13, posteriormente encontrar el **AFD** mínimo equivalente.

4.7 Encontrar un **AFN** sin transiciones épsilon equivalente, posteriormente encontrar un **AFD** equivalente, finalmente encontrar el **AFD** mínimo equivalente para cada uno de los autómatas mostrados en las figuras siguientes:

a)

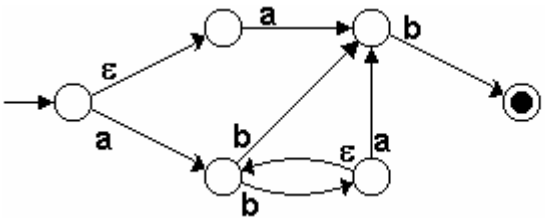


Figura 4.35

b)

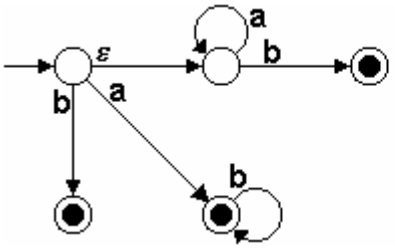


Figura 4.36

c)

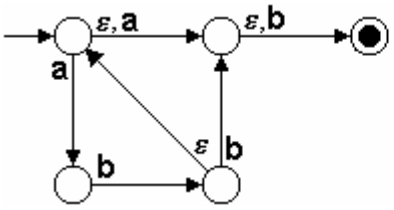


Figura 4.37

d)

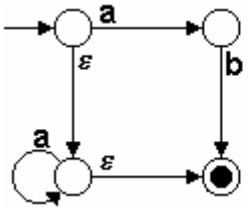


Figura 4.38

e)

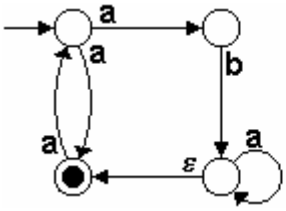


Figura 4.39

f)

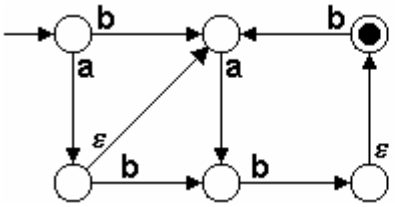


Figura 4.40

g)

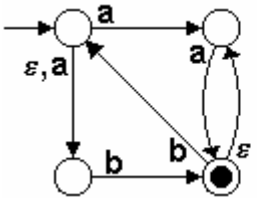


Figura 4.41

h)

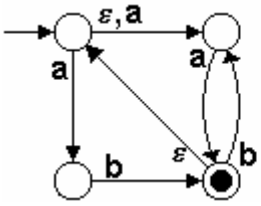


Figura 4.42

i)

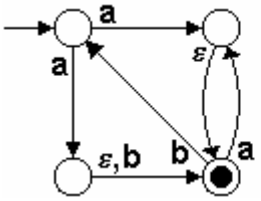


Figura 4.43

j)

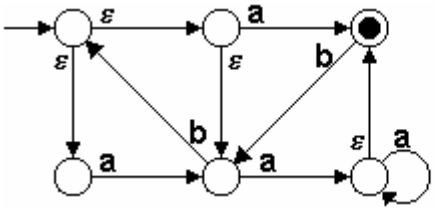


Figura 4.44

4.8 Encontrar un **AFD** equivalente al **AFNG** representado en la siguiente figura:

Figura 4.45

Capítulo

5

Autómatas Finitos y Expresiones Regulares

Se analizan los procedimientos para construir Autómatas, a partir de las expresiones regulares de los lenguajes que deben ser aceptados por ellos y los métodos para encontrar la expresión regular de un lenguaje a partir de un Autómata Finito dado.

Construcción de Autómatas

Para construir autómatas a partir de expresiones regulares, podemos iniciar con los casos más sencillos, los cuales se muestran en la Figura 5.1. Posteriormente podremos construir otros más complejos con base en las operaciones de unión, concatenación y cerradura. Por razones de simplicidad, podemos suponer, sin perder generalidad, que cada autómata en este capítulo deberá poseer un solo estado final.

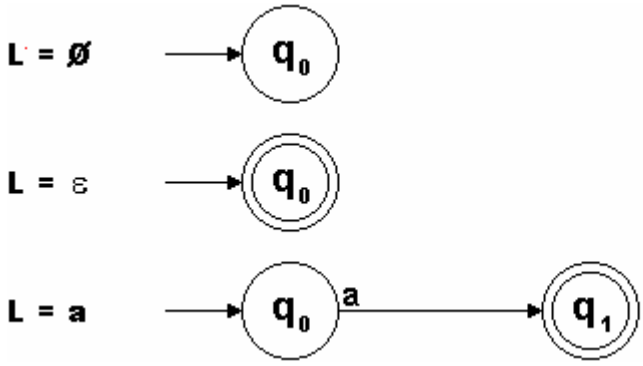


Figura 5.1

Unión

Supongamos que $M_1 = (Q_1, \Sigma_1, \Delta_1, q_0, F_1)$ y $M_2 = (Q_2, \Sigma_2, \Delta_2, p_0, F_2)$, son dos **AFN** que aceptan los lenguajes L_1 y L_2 respectivamente, entonces podemos construir un nuevo **AFN** $M_3 = (Q_3, \Sigma_3, \Delta_3, q_0', F_3)$, que acepte al lenguaje unión, $L_3 = L_1 \cup L_2$, a partir de M_1 y M_2, añadiendo un nuevo estado inicial q_0‘ y dos nuevas transiciones ε, que vayan de

q_0‘ a los estados iniciales q_0 y p_0, respectivamente, así como también añadimos un único estado final q_f’ unido desde q_f y desde p_f por medio de dos transiciones ε.

De esta forma tenemos que M_3 está dado por $Q_3 = Q_1 \cup Q_2 \cup \{ q_0', q_f' \}$, $\Sigma_3 = \Sigma_1 \cup \Sigma_2$, $F_3 = \{q_f'\}$, donde q_0’ es el nuevo estado inicial y Δ_3 se define de tal forma que incluya todas las transiciones de Δ_1 y Δ_2 más las cuatro transiciones épsilon: $\Delta(q_0',\varepsilon) = \{q_0, p_0\}$, $\Delta(q_f,\varepsilon) = \{ q_f' \}$ y $\Delta(p_f,\varepsilon) = \{ q_f' \}$, tal como se ilustra en la Figura 5.2.

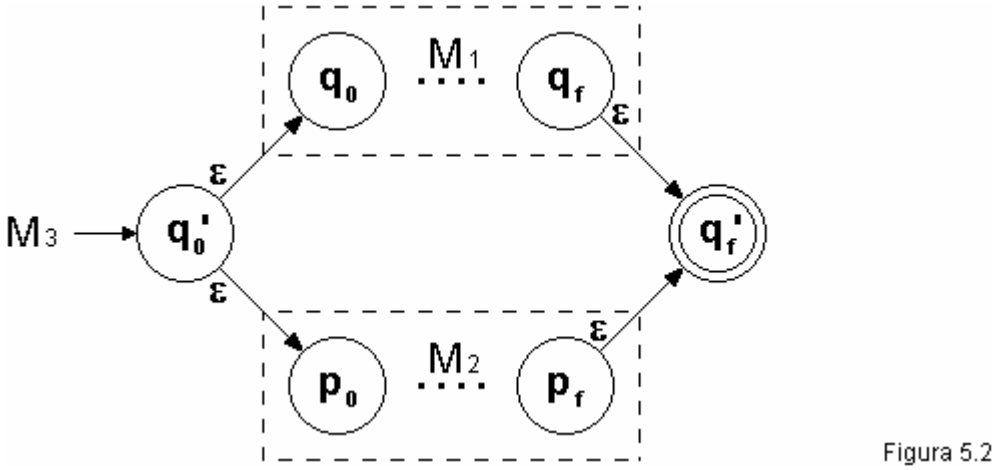


Figura 5.2

Ejemplo

Sean M_1 y M_2, los **AFN**s que aceptan los lenguajes $L(M_1)$ = **ab*** y $L(M_2)$ = **a*b**, respectivamente, entonces el **AFN** M_3, obtenido a partir de M_1 y M_2, acepta el lenguaje $L(M_3)$ = **ab*** ∪ **a*b**, tal como se muestra en la siguiente figura:

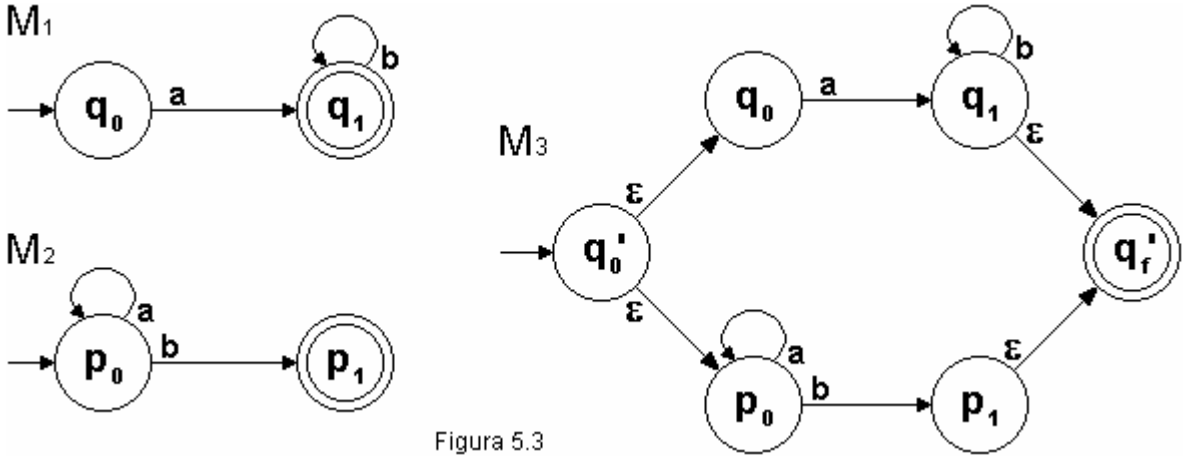


Figura 5.3

Una vez construido el diagrama de transiciones de M_3, se pueden utilizar los criterios vistos en el capítulo anterior, así como el procedimiento de la Cerradura épsilon para eliminar las transiciones épsilon y obtener un **AFN** más sencillo.

Concatenación

Supongamos que $M_1 = (Q_1, \Sigma_1, \Delta_1, q_0, F_1)$ y $M_2 = (Q_2, \Sigma_2, \Delta_2, p_0, F_2)$, son dos **AFN** que aceptan los lenguajes L_1 y L_2 respectivamente, entonces podemos construir un nuevo **AFN** llamado $M_3 = (Q_3, \Sigma_3, \Delta_3, q_0, F_2)$, que acepte al lenguaje $L_3 = L_1 \cdot L_2$, colocando primero a M_1 con su estado inicial q_0 y enseguida se enlaza con M_2, añadiendo una transición ε que vaya del estado de aceptación q_f de M_1, que dejará de serlo, al estado inicial p_0 de M_2, que también deja de serlo.

De esta forma tenemos que el nuevo **AFN** M_3, con alfabeto $\Sigma_3 = \Sigma_1 \cup \Sigma_2$, está dado por $Q_3 = Q_1 \cup Q_2$, donde q_0 es el estado inicial y Δ_3 se define de tal forma que incluya todas las transiciones de Δ_1 y Δ_2 y la nueva transición $\Delta(q_f,\varepsilon) = \{ p_0 \}$, tal como se ilustra en la Figura 5.4.

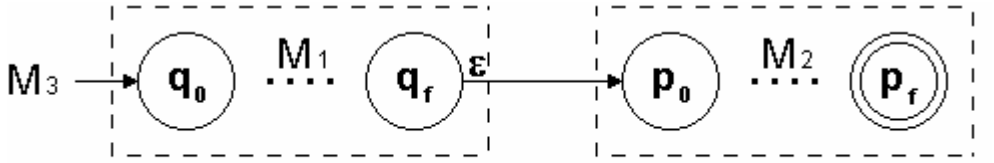


Figura 5.4

Ejemplo

Sean M_1 y M_2, los cuales aceptan los lenguajes $L(M_1) =$ **ab*** y $L(M_2) =$ (**ab**)*, respectivamente, entonces el **AFN** M_3, obtenido a partir de M_1 y M_2, acepta el lenguaje $L(M_3) =$ **ab***(**ab**)*, tal como se muestra en la siguiente figura:

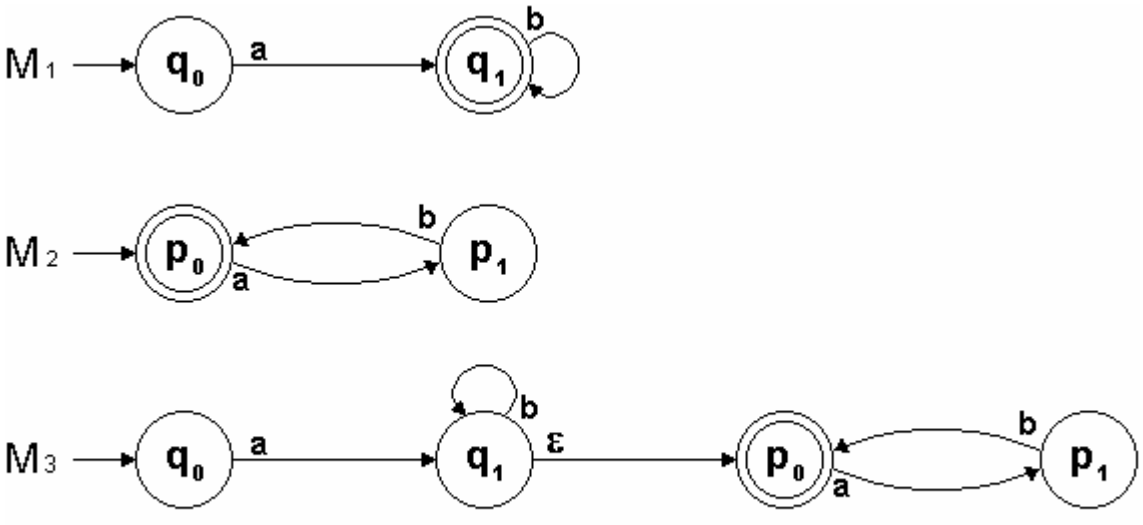


Figura 5.5

Cerradura

Supongamos que $M_1 = (Q_1, \sum_1, \Delta_1, q_0, F_1)$ es un **AFN** que acepta el lenguaje $L(M_1)$, entonces podemos construir un nuevo **AFN** M_3, que acepte al lenguaje $L(M_3) = L(M_1)^*$ agregando un nuevo estado inicial q_0', el cual también será el estado de aceptación (de tal forma que acepte la cadena vacía) y se agrega una transición ε que vaya de q_0' a q_0 (el estado inicial de M_1). Finalmente se cierra el ciclo con otra transición ε desde q_f, el estado de aceptación de M_1, que dejará de serlo, al nuevo estado q_0'.

De esta forma tenemos que el **AFN** $M_3 = (Q_3,\sum_1,\Delta_3,q_0',F_3)$ viene dado por $F_3 = \{ q_0' \}$, $Q_3 = Q_1 \cup \{ q_0' \}$, donde q_0' es el estado inicial y Δ_3 se define de tal forma que incluya todas las transiciones de Δ_1 y las nuevas transiciones $\Delta(q_0',\varepsilon) = \{ q_0 \}$ y $\Delta(q_f,\varepsilon) = \{ q_0' \}$, tal como se ilustra en la Figura 5.6.

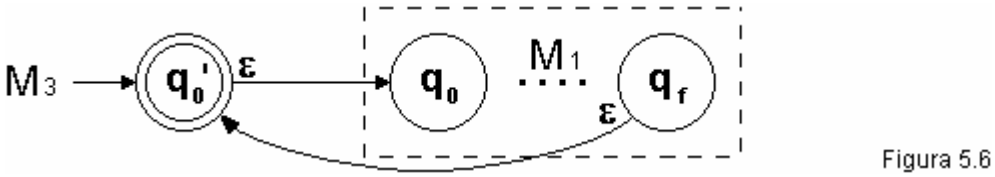


Figura 5.6

Ejemplo

Sean M_1, el cual acepta el lenguaje $L(M_1) = \mathbf{a} \cup \mathbf{b}$, entonces se tiene que el **AFN** M_3, obtenido a partir de M_1, acepta a $L(M_3) = (\mathbf{a} \cup \mathbf{b})^*$, tal como se muestra en la siguiente figura:

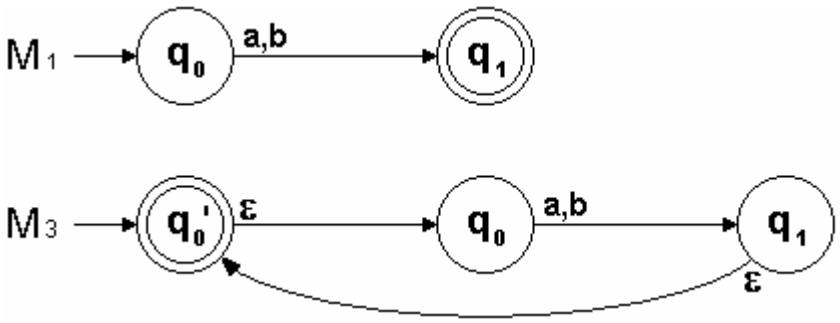


Figura 5.7

Aplicando los criterios 1 y 2, se puede reducir el diagrama de transiciones de M_3 anterior, para obtener el siguiente **AFD** equivalente sin transiciones épsilon:

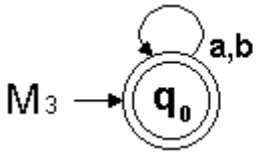


Figura 5.8

En conclusión

Dada cualquier expresión regular, podemos construir un **AFN** que acepte el lenguaje representado por ella, aplicando las técnicas anteriores, por lo tanto, se concluye que todo lenguaje regular es aceptado por algún autómata finito.

Ejemplo 1

Para construir un **AFN** que acepte el lenguaje **010** ∪ **10***, podemos partir de los dos autómatas básicos para cada una de las expresiones que aparecen en la unión, como se ilustra en la Figura 5.9.

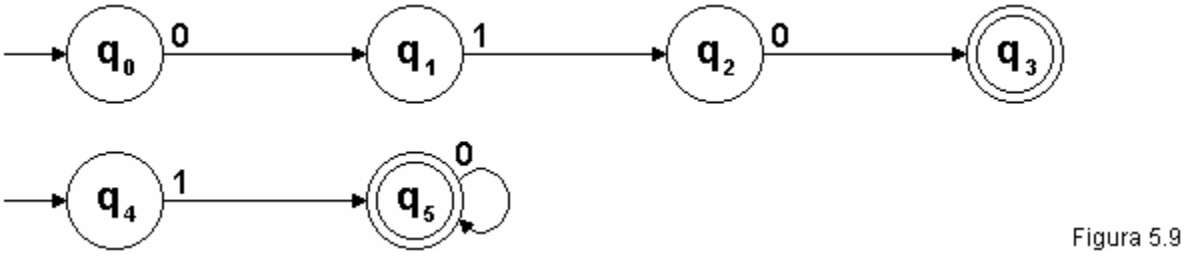


Figura 5.9

Luego construimos el autómata correspondiente a la unión de las dos anteriores, tal como se muestra en la Figura 5.10.

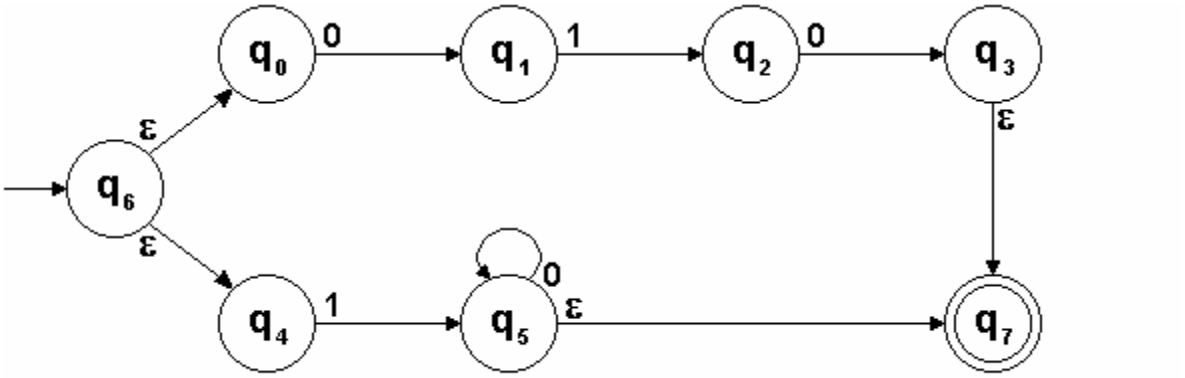


Figura 5.10

Ejemplo 2

Para construir un **AFN** que acepte el lenguaje (**ab** ∪ **b***)***ba** ∪ (**ab**)*, debemos empezar por construir un autómata para la expresión que aparece en el primer paréntesis, luego hacer la cerradura de Kleene, el resultado se concatena con el autómata para el sufijo **ba** y finalmente hacemos la unión con el autómata que acepta a la expresión (**ab**)*, debiendo llegar a un resultado semejante al que se ilustra en la Figura 5.11, se deja como ejercicio la simplificación de este **AFN**, eliminando las transiciones épsilon, para llegar a una representación más compacta.

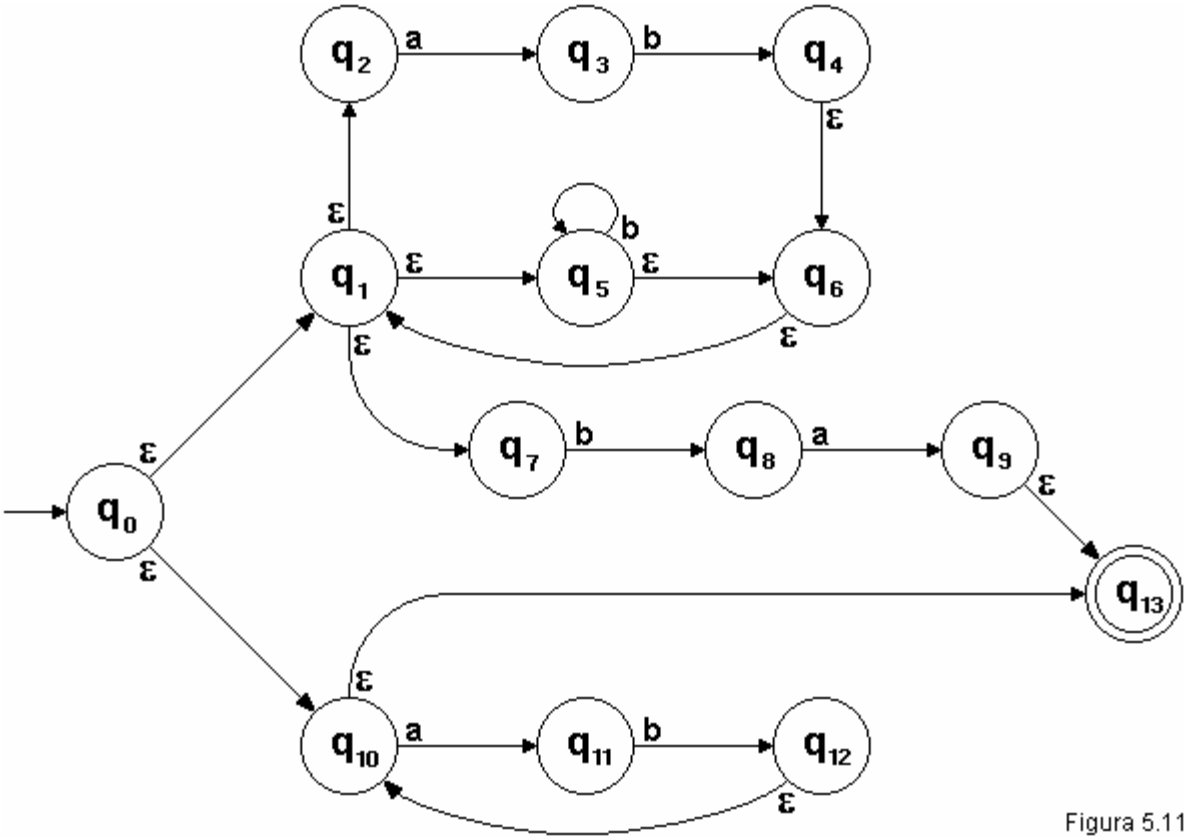


Figura 5.11

Obtención de Expresiones Regulares

Ya vimos que dada una **ER**, es posible construir algún **AFN** que acepte el lenguaje respectivo, ahora vamos a ver, por medio de un método algebraico, que siempre es posible obtener una expresión regular del lenguaje aceptado por M, un **AFN** dado cualquiera. Para ello, primero debemos definir los lenguajes sufijos del lenguaje buscado L(M).

Sea M = (Q, $\sum$, Δ, s, F) un **AFN**, entonces podemos definir a A_i como el conjunto de las cadenas sobre $\sum$ que hacen que M pase desde un estado q_i hasta algún estado de aceptación, esto es:

$$A_i = \{ w \in \sum^* \mid \Delta(q_i, w) \in F \}.$$

Es decir, A_i representa al lenguaje aceptado por el **AF** si suponemos que q_i es el estado inicial, en especial, se tiene que para el estado inicial q_0 se cumple que A_0 representa al leguaje buscado: L(M).

Ejemplo

Obtener la expresión regular del lenguaje aceptado por el **AFD** representado por el **DT** mostrado en la Figura 5.12:

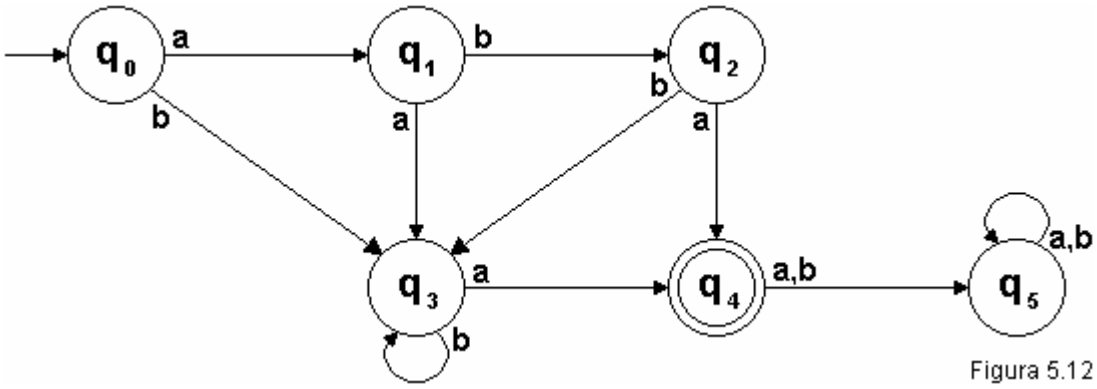


Figura 5.12

Aplicando la definición anterior para los estados de este **AFD**, se obtiene que:

- $A_5 = \varnothing$
- $A_4 = \varepsilon$
- $A_3 = \mathbf{b}^*\mathbf{a}$
- $A_2 = \mathbf{a} \cup \mathbf{b}^+\mathbf{a} = \mathbf{b}^*\mathbf{a}$, etc.

A cada paso podemos observar que la obtención de A_i es más compleja, por lo que se puede apreciar que es más conveniente hacer este mismo análisis algebraicamente.

Si para el estado q_i hay una transición del símbolo **a** hacia el estado q_j y una transición del símbolo **b** hacia el estado q_k, entonces se plantea la siguiente ecuación para el lenguaje A_i: $A_i = \mathbf{a}A_j \cup \mathbf{b}A_k$. Puesto que A_j y A_k son dos sufijos de A_i.

Además, debemos tomar en cuenta que si el estado q_i es un estado de aceptación, entonces hay que agregar el término independiente adicional ε.

Para el **AFD** mostrado en la Figura 5.12, tenemos que se obtiene el siguiente sistema de ecuaciones:

$$A_0 = \mathbf{a}A_1 \cup \mathbf{b}A_3 \qquad A_1 = \mathbf{a}A_3 \cup \mathbf{b}A_2$$

$$A_2 = \mathbf{a}A_4 \cup \mathbf{b}A_3 \qquad A_3 = \mathbf{a}A_4 \cup \mathbf{b}A_3$$

$$A_4 = \varepsilon \cup \mathbf{a}A_5 \cup \mathbf{b}A_5 \qquad A_5 = \mathbf{a}A_5 \cup \mathbf{b}A_5$$

Para resolver este sistema de ecuaciones se parte del hecho que la única solución posible para A_5 es el lenguaje vacío, $\varnothing$, ya que A_5 aparece en todos los términos de la ecuación. (Como una referencia, considere el análogo algebraico x = 2x + 4x, que tiene como única solución a x = 0).

Ahora sustituyendo A_5 en la ecuación de A_4, nos queda simplemente que $A_4 = \varepsilon$, y si reemplazamos este valor en la ecuación de A_3 queda: $A_3 = \mathbf{a} \cup \mathbf{b}A_3$.

Para resolver este tipo de ecuaciones, en las que el mismo elemento A_i aparece en ambos lados de la ecuación, aplicamos el resultado del siguiente lema:

Lema de Arden

Una ecuación de la forma $A = \mathbf{r}A \cup \mathbf{s}$, tiene como solución única a: $A = \mathbf{r}^*\mathbf{s}$, donde **r** y **s** son dos expresiones regulares cualesquiera que no contienen a A.

Entonces, continuando con la solución de nuestro problema, resolvemos la ecuación de A_3 aplicando el lema de Arden y obtenemos que: $A_3 = \mathbf{b}^*\mathbf{a}$.

A partir de ahí se obtiene por substitución: $A_2 = \mathbf{a} \cup \mathbf{b}\mathbf{b}^*\mathbf{a} = \mathbf{a} \cup \mathbf{b}^+\mathbf{a} = \mathbf{b}^*\mathbf{a}$. (No es sorprendente que A_2 sea igual a A_3, pues ambos tienen la misma ecuación, esto se debe a que ambos estados q_2 y q_3 son equivalentes).

Luego obtenemos $A_1 = \mathbf{a}(\mathbf{b}^*\mathbf{a}) \cup \mathbf{b}(\mathbf{b}^*\mathbf{a}) = (\mathbf{a} \cup \mathbf{b})\mathbf{b}^*\mathbf{a}$

Y finalmente, si substituimos A_1 y A_3 en A_0, se tiene que el lenguaje buscado es:

$$L(M) = A_0 = \mathbf{a}(\mathbf{a} \cup \mathbf{b})\mathbf{b}^*\mathbf{a} \cup \mathbf{b}\mathbf{b}^*\mathbf{a} = (\mathbf{a}\mathbf{a} \cup \mathbf{a}\mathbf{b} \cup \mathbf{b})\mathbf{b}^*\mathbf{a}$$

Ejemplo 2

Encontrar la **ER** del lenguaje aceptado por el **AFD** mostrado en la Figura 5.13:

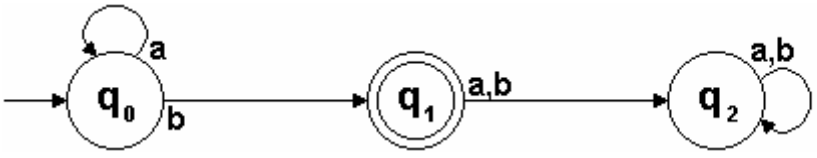


Figura 5.13

A partir del diagrama obtenemos el sistema de ecuaciones siguiente:

$$A_0 = \mathbf{a}A_0 \cup \mathbf{b}A_1$$

$$A_1 = \varepsilon \cup \mathbf{b}A_2 \cup \mathbf{a}A_2$$

$$A_2 = \mathbf{a}A_2 \cup \mathbf{b}A_2$$

Para resolverlo, podemos ver de que: $A_2 = \varnothing$, y a partir de ahí, se obtiene que: $A_1 = \varepsilon$, por tanto queda: $A_0 = \mathbf{a}A_0 \cup \mathbf{b}$, y resolviendo por medio del Lema de Arden, resulta:

$$L(M) = A_0 = \mathbf{a}^*\mathbf{b}.$$

Ejemplo 3

Encontrar la **ER** del lenguaje aceptado por el **AFD** representado por el **DT** mostrado en la figura 5.14:

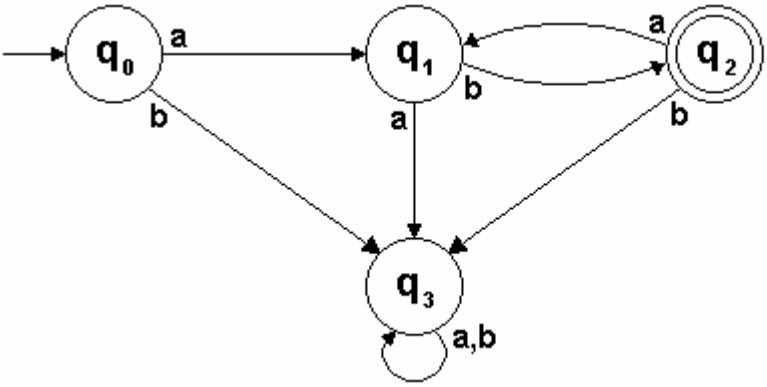


Figura 5.14

Analizando las transiciones en cada uno de los estados del diagrama, obtenemos el siguiente sistema de ecuaciones:

$$A_0 = \mathbf{a}A_1 \cup \mathbf{b}A_3$$
$$A_1 = \mathbf{b}A_2 \cup \mathbf{a}A_3$$
$$A_2 = \varepsilon \cup \mathbf{a}A_1 \cup \mathbf{b}A_3$$
$$A_3 = \mathbf{a}A_3 \cup \mathbf{b}A_3$$

Partiendo del hecho que: $A_3 = \varnothing$, se obtiene que: $A_2 = \varepsilon \cup \mathbf{a}A_1$ y que $A_1 = \mathbf{b}A_2$ por lo tanto $A_2 = \varepsilon \cup \mathbf{ab}A_2$. Esta ecuación se resuelve por medio el Lema de Arden, obteniéndose que $A_2 = (\mathbf{ab})^*$ y posteriormente, sustituyendo esta expresión en A_1, se obtiene el siguiente resultado: $A_1 = \mathbf{b}(\mathbf{ab})^*$ y de ahí se llega a que la expresión buscada está dada por:

$$L(M) = A_0 = \mathbf{a}A_1 = \mathbf{ab}(\mathbf{ab})^* = (\mathbf{ab})^+.$$

Dado que el método algebraico es aplicable igualmente a **AFD**s que a **AFN**s, podemos facilitar su aplicación si desde un inicio descartamos a los estados no deseados que se detecten, ya que no aportan nada a la solución del problema, tal como se observa en el siguiente ejemplo:

Ejemplo 4

Dado el **AFD** de la figura 5.15, encontrar la expresión regular del lenguaje que representa.

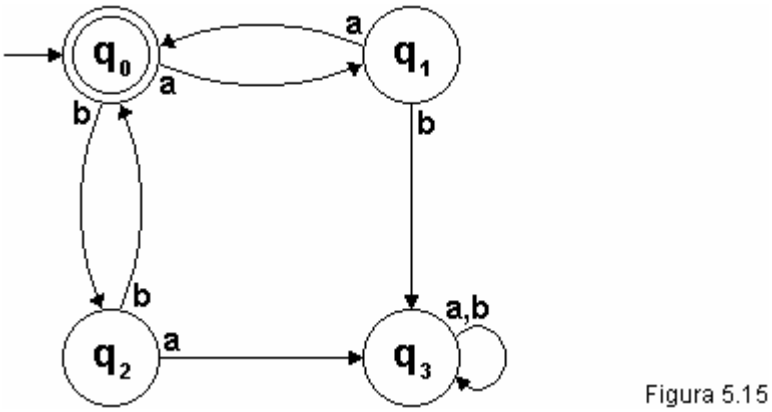


Figura 5.15

Analizando el diagrama, se observa que podemos descartar a q_3, debido a que se trata de un estado no deseado, por lo que se obtiene el siguiente sistema de ecuaciones reducido:

$$A_0 = \varepsilon \cup \mathbf{a}A_1 \cup \mathbf{b}A_2$$

$$A_1 = \mathbf{a}A_0$$

$$A_2 = \mathbf{b}A_0$$

Se deja al lector la tarea de resolverlo, y de que verifique que la respuesta del sistema de ecuaciones anterior es: $A_0 = L(M) = (\mathbf{a}^2 \cup \mathbf{b}^2)^*$.

Ejemplo 5

Dado el **AFN** mostrado en la figura 5.16, encontrar la expresión regular del lenguaje que acepta.

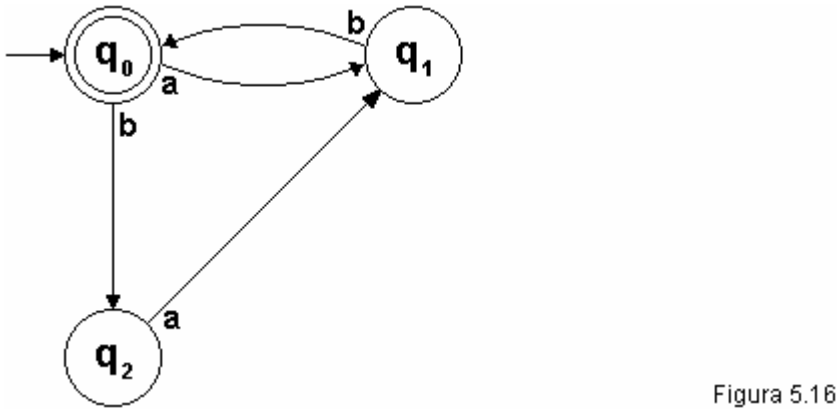


Figura 5.16

Analizando el diagrama, obtenemos el sistema de ecuaciones siguiente:

$$A_0 = \varepsilon \cup \mathbf{a}A_1 \cup \mathbf{b}A_2$$

$$A_1 = \mathbf{b}A_0$$

$$A_2 = \mathbf{a}A_1$$

Se deja al lector la tarea de resolverlo, y que verifique que la respuesta del sistema de ecuaciones es: $A_0 = L(M) = (\mathbf{ab} \cup \mathbf{bab})^*$.

Simplificación de AFNGs

Otro método para obtener la expresión regular a partir de un **AF** que es de especial interés consiste en la simplificación de **AFNG**s por medio de un algoritmo que permite ir eliminando de estado en estado, hasta reducirlo a solamente dos, el inicial y el final y una sola transición entre ellos etiquetada por la expresión regular.

Para facilitar la aplicación de este algoritmo, se parte de los siguientes supuestos:

- El estado inicial no tiene transiciones de entrada, solo de salida.
- El estado de aceptación es único y no tiene transiciones de salida.
- La función de transición dual se define como δ': $Q \times Q \rightarrow \sum^*$.

Algoritmo

Dado un **AF** cualquiera, que acepta un lenguaje L, debemos verificar primero que se satisfacen las condiciones arriba mencionadas, en caso contrario se añaden uno o dos estados más según se requiera.

1. Si el número de estados del **AF** actual es n = 2, la etiqueta de la transición es la expresión regular.
2. Si n > 2, entonces se selecciona un estado intermedio q_i y se elimina, redefiniendo las transiciones de para cada pareja de estados q_j y q_k que tengan transiciones hacia y desde q_i, respectivamente, de la siguiente forma: $\delta'(q_j,q_k) = \mathbf{r}_1 \mathbf{r}_2^* \mathbf{r}_3 \cup \mathbf{r}_4$, donde: $\mathbf{r}_1 = \delta'(q_j,q_i)$, $\mathbf{r}_2 = \delta'(q_i,q_i)$, $\mathbf{r}_3 = \delta'(q_i,q_k)$ y $\mathbf{r}_4 = \delta'(q_j,q_k)$.

   (Se recomienda eliminar preferentemente al estado en el que resulte el menor producto del número de entradas por el número de salidas)
3. Se repite el proceso.

El paso 2 lo podemos representar gráficamente por medio de la siguiente figura:

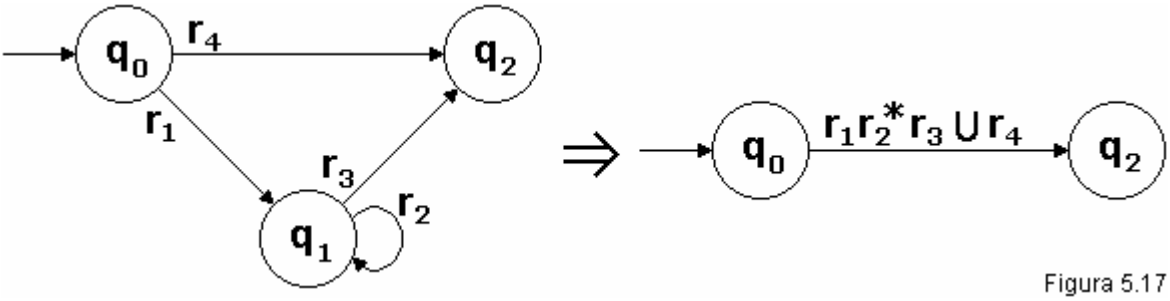


Figura 5.17

Ejemplo

Encontrar la expresión regular del lenguaje aceptado por el siguiente **AFN**:

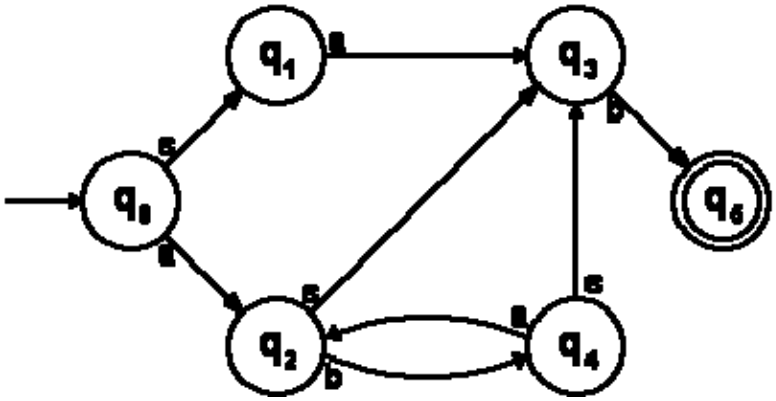


Figura 6.18

Como se cumplen las condiciones exigidas por el algoritmo, podemos elegir uno de los estados para eliminarlo, en este caso, iniciamos quitando a q_1, puesto que solamente tiene una transición de entrada y una de salida.

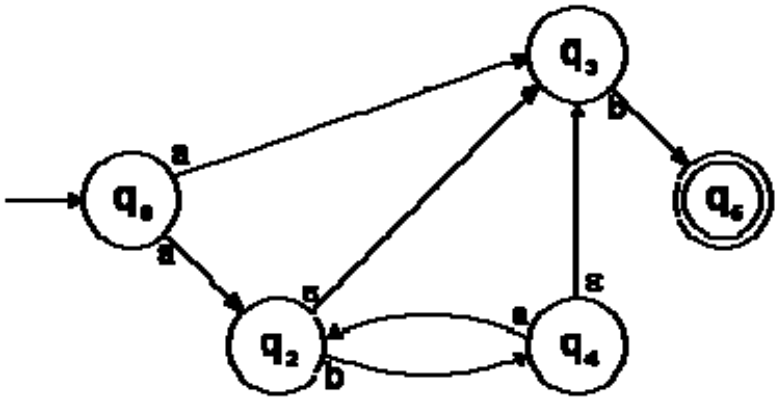


Figura 6.19

Observe que el resultado de eliminar a q_1 es equivalente a la aplicación del criterio 1 para la eliminación de transiciones épsilon, que se vio en el capítulo anterior.

Ahora conviene eliminar a q_4, que tiene dos transiciones de salida por una sola de entrada, por lo que hay que considerar dos combinaciones al aplicar el algoritmo, $\delta'(q_2,q_2) = (\mathbf{ba})^*$ y $\delta'(q_2,q_3) = \varepsilon \cup \mathbf{b}$, tal como se ilustra en la Figura 5.20:

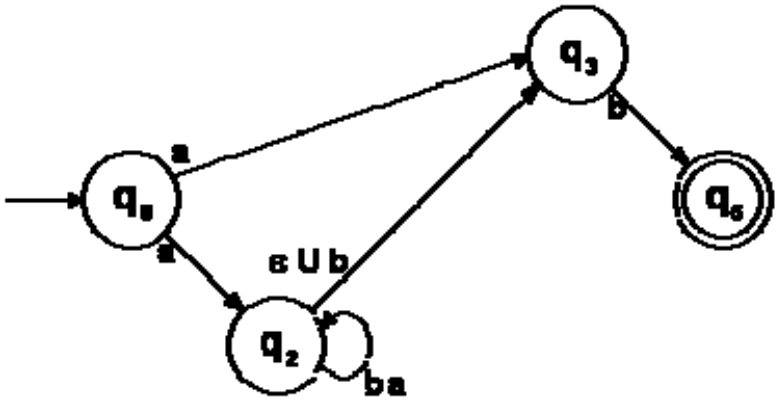


Figura 5.20

A continuación procedemos a eliminamos el estado q_2, que solo tiene una entrada y una salida.

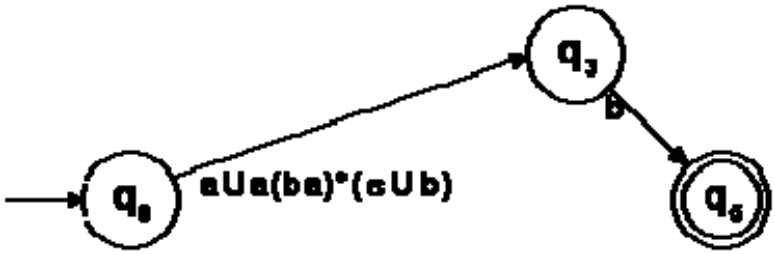


Figura 5.21

Finalmente eliminando el estado q_3 resulta la expresión regular buscada:

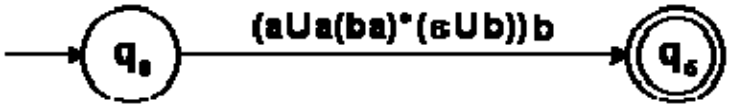


Figura 5.22

Por todo lo expuesto en este capítulo, se puede afirmar que para cada Autómata Finito M, existe una Expresión Regular **r** para la cual L(**r**) = L(M) y viceversa, lo que concluye Kleene enunciando el siguiente teorema:

Teorema de Kleene

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

En la Figura 5.23 se resume gráficamente lo que hemos realizado en estos dos capítulos sobre la convertibilidad que existe entre los **AF**s y las **ER**s. Primeramente, dada cualquier **ER**, podemos construir un Autómata Finito No Determinista con transiciones épsilon, posteriormente podemos encontrar un **AFN** equivalente sin este tipo de transiciones, para que de ahí, encontremos un **AFD** equivalente y por otro

lado, dado un **AFD**, podemos ser capaces de encontrar la Expresión Regular del lenguaje aceptado por éste.

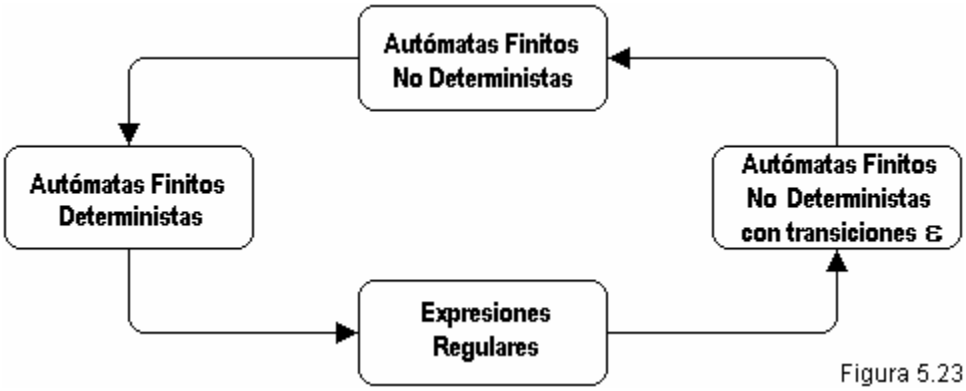


Figura 5.23

Preguntas

a) ¿Existe algún Lenguaje Regular que no pueda ser aceptado por algún **AFN**?

b) ¿Existe algún **AFN** que reconozca a un lenguaje que no sea regular?

c) ¿Es necesario el empleo de las transiciones épsilon para construir **AF**s a partir de una Expresión Regular dada?

d) ¿Por qué, dado un **AFN**, es conveniente encontrar un **AFD** equivalente?

e) ¿Por qué es útil el empleo de los **AFN**s?

f) El método algebraico para encontrar **ER**s ¿es más confiable que el método de la simplificación de **AFNG**s?

g) ¿Por qué se necesita anteponer un estado inicial y de aceptación para construir un **AFN** que acepte la cerradura de Kleene de un lenguaje L?

Ejercicios Capítulo 5

5.1 De acuerdo con las técnicas descritas para la construcción de autómatas, construir los **AFN** que acepten cada uno de los siguientes lenguajes:

a) $(\mathbf{a} \cup \mathbf{b})^* \cup (\mathbf{aba})^+$

b) $((\mathbf{ab} \cup \mathbf{aab})^*\mathbf{a}^*)^*$

c) $(\mathbf{ba} \cup \mathbf{b})^* \cup (\mathbf{bb} \cup \mathbf{a})^*$

d) $((\mathbf{a}^*\mathbf{b}^*\mathbf{a}^*)^*\mathbf{b})^*$

e) $(\mathbf{ab} \cup \mathbf{abb} \cup \mathbf{aba})^*$

5.2 Dados los lenguajes $L_1 = \mathbf{a}$ y $L_2 = \mathbf{ab}^*\mathbf{c}$, construir:

a) Un **AFN** que acepte el Lenguaje $L_1 \cdot L_2 \cup L_1^*$

b) Otro **AFN** que acepte el Lenguaje $L_2 \cdot L_1 \cup L_2^*$

5.3 Sean $M_1 = (Q_1, \Sigma_1, F_1, s_1, \Delta_1)$ y $M_2 = (Q_2, \Sigma_2, F_2, s_2, \Delta_2)$, donde $Q_1 = \{ q_0, q_1, q_2 \}$, $\Sigma_1 = \{\mathbf{a},\mathbf{b}\}$, $F_1 = \{q_0\}$, $s_1 = q_0$, $Q_2 = \{p_0, p_1, p_2, p_3\}$, $\Sigma_2 = \{\mathbf{0},\mathbf{1}\}$, $s_1 = p_0$, $F_2 = \{ p_0, p_1\}$, Δ_1 y Δ_2 dados en la tablas 5.1 y 5.2:

Δ_1	**a**	**b**
$\rightarrow$*q_0	$\{q_1, q_2\}$	$\varnothing$
q_1	$\varnothing$	$\{q_0\}$
q_2	$\{q_0\}$	$\{q_2\}$

Tabla 5.1

Δ_2	**0**	**1**
$\rightarrow$*p_0	$\{p_1\}$	$\varnothing$
*p_1	$\varnothing$	$\{ p_2, p_3\}$
p_2	$\{p_1\}$	$\varnothing$
p_3	$\{p_2\}$	$\varnothing$

Tabla 5.2

a) Obtener un **AFN** que acepte el lenguaje $L(M_1) \cdot L(M_2)$.

b) Obtener un **AFN** que acepte el lenguaje $L(M_2) \cdot L(M_1)^2$.

c) Obtener un **AFN** que acepte el lenguaje $L(M_1)^2 \cup L(M_2)$.

5.4 Dado el **AFN** mostrado en la Figura 5.24, encontrar la expresión regular del lenguaje que acepta.

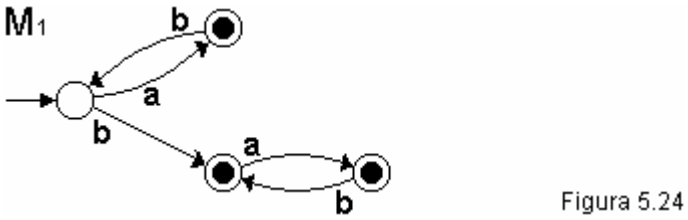


Figura 5.24

5.5 Dado el **AFN** mostrado en la Figura 5.25, encontrar la expresión regular del lenguaje que acepta.

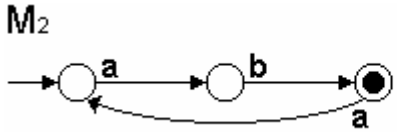


Figura 5.25

5.6 Sean M_1 y M_2, los **AFN**s mostrados en las Figuras 5.24 y 5.25.

a) Obtener un **AFN** que acepte el lenguaje $L(M_1) \cdot L(M_2)$.

b) Obtener un **AFN** que acepte el lenguaje $L(M_2) \cup L(M_1)$.

5.7 Dados los **AFD**s mostrados en las figuras 3.12, 3.13, 3.14, 3.16 y 4.11, obtener la expresión regular del lenguaje que acepta cada uno de ellos.

5.8 Dados los **AFD**s mostrados en las figuras de la 5.26 a la 5.31, encontrar la expresión regular del lenguaje que acepta cada uno de ellos.

a)

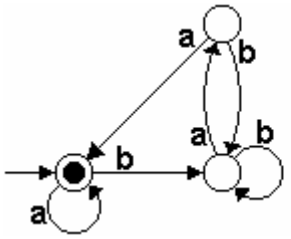


Figura 5.26

b)

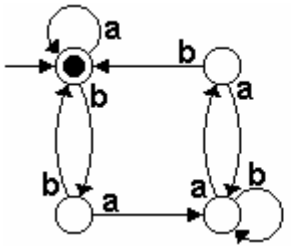


Figura 5.27

c)

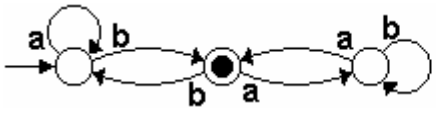


Figura 5.28

d)

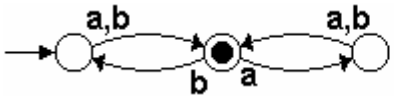


Figura 5.29

e)

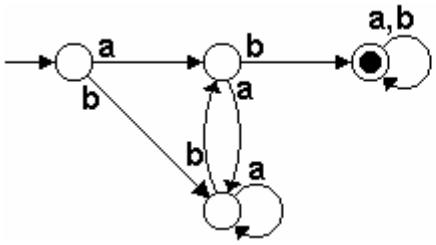


Figura 5.30

f)

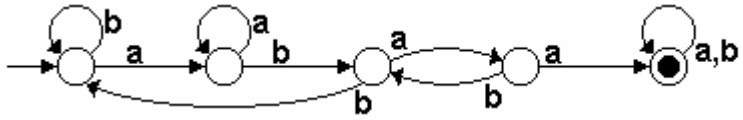


Figura 5.31

5.9 Dados los **AFN**s mostrados en las figuras de la 5.32 a la 5.34, encontrar la expresión regular del lenguaje que acepta cada uno de ellos.

a)

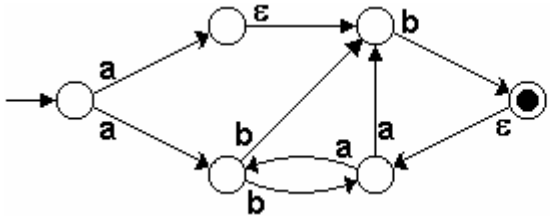


Figura 5.32

b)

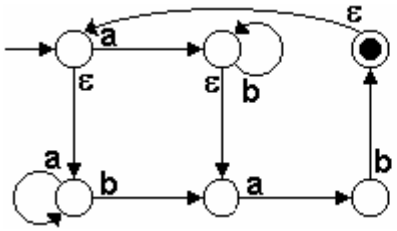


Figura 5.33

c)

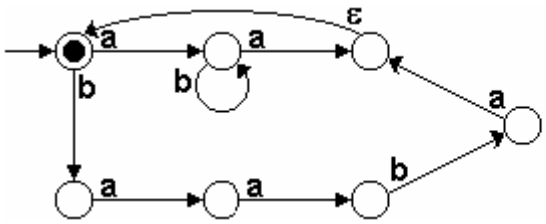


Figura 5.34

Capítulo

6

Gramáticas Regulares

Se define el concepto de Gramática Regular y su equivalencia con las expresiones regulares y con los autómatas finitos. Se introduce el concepto de Gramáticas Regulares Reversas y su relación con el Lenguaje Inverso.

En los capítulos anteriores hemos empleado a los Diagramas de Transiciones como una representación Gráfica de los **AF**s para el reconocimiento de lenguajes, pero éstos también pueden ser utilizados para representar a un dispositivo generador de cadenas para un lenguaje dado; de tal manera que partiendo de una cadena vacía, se le van a ir concatenando símbolos en la medida de que se recorren las distintas transiciones que forman una determinada trayectoria sobre el Diagrama, desde el estado inicial hasta algún estado final (al que ya no llamaríamos de aceptación).

Ejemplo 1

El Diagrama de Transiciones mostrado en la figura 6.1 representa al lenguaje regular $L = \mathbf{a}(\mathbf{a}^* \cup \mathbf{b}^*)\mathbf{b} = \mathbf{a}^+\mathbf{b} \cup \mathbf{a}\mathbf{b}^+$.

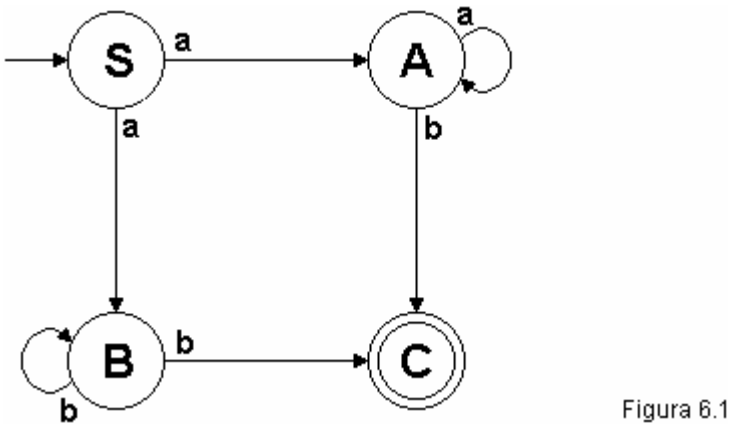


Figura 6.1

Aplicando el método descrito se puede generar cualquier cadena perteneciente al lenguaje citado, tal como se ejemplifica a continuación.

La secuencia de transiciones que genera la cadena *w* = **aaab**, es la siguiente:

$$S \xrightarrow{a} A \xrightarrow{a} A \xrightarrow{a} A \xrightarrow{b} C$$

Obsérvese que se han nombrado a los estados con las letras mayúsculas S, A, B y C, en vez de la notación empleada para los **AF**s, esto se hace para poder introducir una nueva simbología.

Simbólicamente podemos representar la generación de cada uno de los símbolos por medio de las siguientes *Reglas de Sustitución* o *Producciones*:

$$S \rightarrow \mathbf{a}A$$
$$S \rightarrow \mathbf{a}B$$
$$A \rightarrow \mathbf{a}A$$
$$A \rightarrow \mathbf{b}C$$
$$B \rightarrow \mathbf{b}B$$
$$B \rightarrow \mathbf{b}C$$
$$C \rightarrow \varepsilon \text{ (por ser un estado de aceptación)}$$

La primera de estas reglas se interpreta como que la transición del estado S al estado A nos genera el símbolo **a**. La flecha $\rightarrow$ expresa que S "*Es sustituido por*" o *"produce"* la cadena **a**A, de modo que podemos utilizar esta regla cuando tengamos al símbolo S en una cadena, el cual puede ser sustituido por la expresión que aparece a la derecha de la flecha.

El proceso de generar una cadena se llama *Derivación*, y parte de un símbolo inicial representado por S. Por lo tanto, la cadena *w* = **aaab** se puede generar por medio de las reglas anteriores. De este modo, si partimos del símbolo inicial S y aplicamos la primera regla, obtenemos **a**A, y si luego se aplica en dos ocasiones la tercera regla, podemos obtener **aaa**A, luego empleamos la cuarta regla para obtener **aaab**C y finalmente la última regla para terminar en **aaab**.

Este proceso se puede describir de manera compacta así:

$$S \Rightarrow \mathbf{a}A \Rightarrow \mathbf{aa}A \Rightarrow \mathbf{aaa}A \Rightarrow \mathbf{aaab}C \Rightarrow \mathbf{aaab}$$

Donde la doble flecha $\Rightarrow$ significa *genera* o *deriva*. Los símbolos en minúsculas son los elementos de nuestro alfabeto $\sum$ y se les conoce como *Símbolos Terminales*, mientras que las letras mayúsculas se denominan *Símbolos No Terminales* (**SNT**), debido a que solamente aparecen durante el proceso de la derivación, pero no pueden figurar en una cadena finalizada, además, éstos son los símbolos que pueden ser reemplazados aplicando alguna de las reglas de sustitución disponibles.

Asimismo, podemos utilizar la barra vertical | (que significa *o*) para agrupar de forma compacta a las reglas de sustitución de cada **SNT**. Y si además reemplazamos el símbolo C por su valor ε, el conjunto de reglas anteriores se representa así:

$$S \rightarrow \mathbf{a}A \mid \mathbf{a}B$$

$$A \rightarrow \mathbf{a}A \mid \mathbf{b}$$

$$B \rightarrow \mathbf{b}B \mid \mathbf{b}$$

Definición de Gramática

Una Gramática es la cuarteta G = (N, ∑, S, P), donde ∑ es un alfabeto de Símbolos Terminales, N es la colección de Símbolos No Terminales, S es el Símbolo Inicial (No Terminal, S ∈ N) y P es la colección de reglas de sustitución, también llamadas *Producciones*. Son éstas últimas las que nos permiten distinguir a las distintas clases de gramáticas, como se indica a continuación:

Gramática Regular

Una Gramática Regular (**GR**) es una cuarteta G = (N, ∑, S, P), donde las Producciones son de la forma $A \rightarrow w$, donde A ∈ N, mientras que $w \in \sum^* \cdot (\varepsilon \cup N)$, esto es, A es un Símbolo No Terminal, mientras que *w* puede contener varios Símbolos Terminales seguidos de uno o ningún **SNT**, al final de la cadena *w*, como ejemplos, podemos citar a las siguientes producciones:

$$S \rightarrow \mathbf{0110}A \mid B$$

$$A \rightarrow \mathbf{101}A \mid \varepsilon$$

$$B \rightarrow \mathbf{01} \mid \mathbf{0}A$$

Una manera equivalente de denotar a estas producciones es representarlas como Relaciones, así, para el ejemplo anterior, se representan las producciones como:

$$P = \{ (S, \mathbf{0110}A), (S, B), (A, \mathbf{101}A), (A, \varepsilon), (B, \mathbf{01}), (B, \mathbf{0}A) \}$$

Construcción de Gramáticas

Para construir una Gramática Regular, a partir de una expresión regular, se puede optar por construir el Diagrama de Transiciones que representa el **AFN** que acepta al lenguaje regular dado y de ahí obtener las producciones respectivas, como se muestra en los siguientes ejemplos.

Ejemplo 1

Obtener una Gramática Regular para generar al lenguaje regular: L = **a*****b** ∪ **a**. Primero construimos el Diagrama de Transiciones correspondiente al **AFN** que acepta el lenguaje referido, obtenemos el diagrama mostrado en la Figura 6.2:

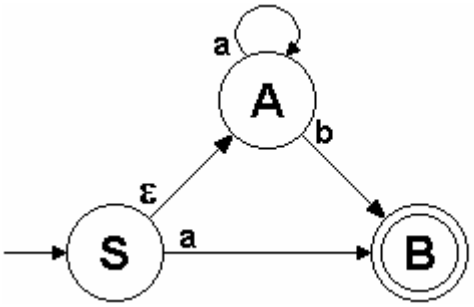


Figura 6.2

Del diagrama anterior se obtiene la gramática:

$$S \rightarrow A \mid \mathbf{a}B$$

$$A \rightarrow \mathbf{a}A \mid \mathbf{b}B$$

$$B \rightarrow \varepsilon$$

Que podemos simplificar al sustituir al **SNT** B, y nos queda:

$$S \rightarrow A \mid \mathbf{a}$$

$$A \rightarrow \mathbf{a}A \mid \mathbf{b}$$

Ejemplo 2

Obtener una Gramática Regular para generar al lenguaje regular siguiente:

$$L = \mathbf{a}^*\mathbf{b} \cup \mathbf{b}^*\mathbf{a}$$

Construyendo el Diagrama de Transiciones respectivo, se puede obtener uno semejante al mostrado en la Figura 6.3:

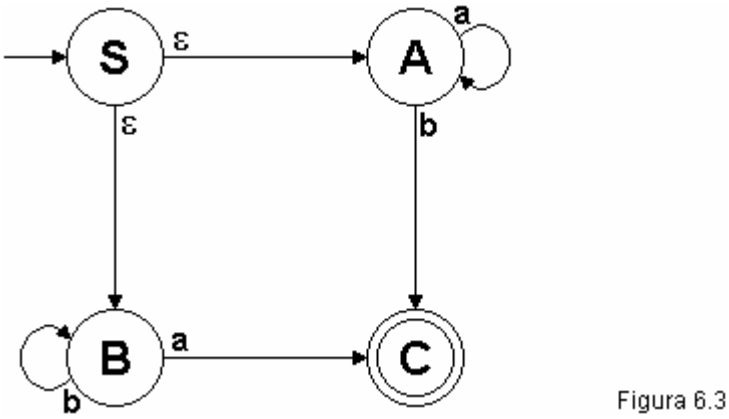


Figura 6.3

Y del diagrama anterior, se obtiene la siguiente gramática:

$$S \rightarrow A \mid B$$
$$A \rightarrow \mathbf{a}A \mid \mathbf{b}C$$
$$B \rightarrow \mathbf{b}B \mid \mathbf{a}C$$
$$C \rightarrow \varepsilon$$

Esta gramática se puede simplificar ligeramente sustituyendo el Símbolo C en las producciones de A y B, con lo que nos queda una gramática equivalente:

$$S \rightarrow A \mid B$$
$$A \rightarrow \mathbf{a}A \mid \mathbf{b}$$
$$B \rightarrow \mathbf{b}B \mid \mathbf{a}$$

Ejemplo 3

Obtener la gramática regular para el lenguaje L = (**a*****b** ∪ **b*****a**)* y utilizarla para generar la cadena *w* = **aaaba**. Construyendo el **DT** respectivo, obtenemos el mostrado en la siguiente figura:

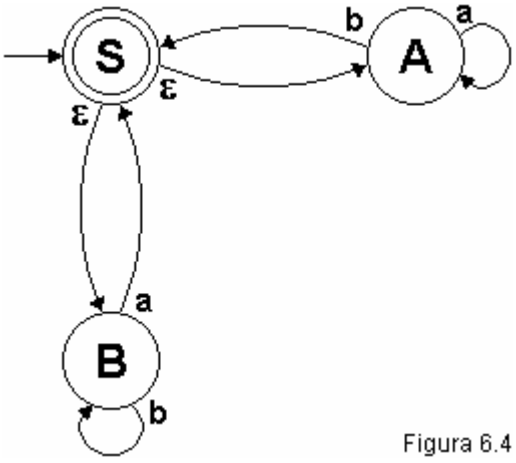


Figura 6.4

Del diagrama anterior se obtiene que la gramática buscada es:

$$S \rightarrow A \mid B \mid \varepsilon$$
$$A \rightarrow \mathbf{a}A \mid \mathbf{b}S$$
$$B \rightarrow \mathbf{b}B \mid \mathbf{a}S$$

Entonces, para generar la cadena solicitada tenemos, entre otras, la siguiente derivación:

$$S \Rightarrow A \Rightarrow \mathbf{a}A \Rightarrow \mathbf{aa}A \Rightarrow \mathbf{aaa}A \Rightarrow \mathbf{aaab}S \Rightarrow \mathbf{aaab}B \Rightarrow \mathbf{aaaba}S \Rightarrow \mathbf{aaaba}$$

En muchas ocasiones, con un poco de experiencia, es posible encontrar una Gramática Regular sin tener que construir el Diagrama de Transiciones respectivo, basta con analizar la expresión regular y determinar cuales reglas debe contener la gramática para generar al lenguaje en cuestión.

Ejemplo 4

Obtener la Gramática Regular sin construir el Diagrama de Transiciones para el lenguaje siguiente:

$$L = \mathbf{ab} (\mathbf{a} \cup \mathbf{b})^* \mathbf{b}$$

Analizando la expresión regular, determinamos que primero necesitamos una producción para inicializar la cadena con el prefijo **ab**:

$$S \rightarrow \mathbf{ab}A$$

Posteriormente debemos tener dos alternativas para concatenar la letra **a** o la **b**, tantas veces como se quieran y una producción final que permita colocar una **b** al término de la cadena:

$$A \rightarrow \mathbf{a}A \mid \mathbf{b}A \mid \mathbf{b}$$

Obtención de Expresiones Regulares

El lenguaje generado por una gramática G se denota como L(G). Una gramática puede ser especificada completamente por medio de sus producciones, así, por ejemplo: $S \rightarrow \mathbf{a}S \mid \mathbf{b}$, especifica la gramática que genera al lenguaje $L(G) = \mathbf{a}^*\mathbf{b}$.

Ejemplo 1

Considérese la gramática regular G = (N, ∑, S, P), donde ∑ = {**a**, **b**}, N = {S, A} y cuyas producciones son:

$$S \rightarrow \mathbf{b}A$$

$$A \rightarrow \mathbf{aa}A \mid \mathbf{b} \mid \varepsilon$$

El Lenguaje generado por esta gramática L(G) contiene todas las cadenas de la forma $\mathbf{ba}^{2n}\mathbf{b}$ o de la forma $\mathbf{ba}^{2n}$, para $n \geq 0$, es decir: $L(G) = \mathbf{b}(\mathbf{a}^2)^*(\mathbf{b} \cup \varepsilon)$.

La técnica más confiable para obtener este resultado consiste en expresar las producciones como ecuaciones, reemplazando las flechas por signos de igual y las

barras por signo de unión, resultando un sistema de ecuaciones idéntico al que se obtiene cuando tenemos el diagrama de transiciones de un **AF**, en este caso:

$$S = \mathbf{b}A$$

$$A = \mathbf{aa}A \cup \mathbf{b} \cup \varepsilon$$

De esta forma, y aplicando el Lema de Arden se tiene que: A = (**aa**)* (**b** ∪ ε), y reemplazando la A en la primera ecuación, se tiene: S = **b**(**aa**)* (**b** ∪ ε.).

Ejemplo 2

Obtener una expresión regular para el lenguaje generado por la gramática dada por las siguientes producciones:

$$S \rightarrow \mathbf{b}A \mid \mathbf{a}B \mid \varepsilon$$

$$A \rightarrow \mathbf{aba}S$$

$$B \rightarrow \mathbf{bab}S$$

Expresando el sistema de ecuaciones respectivo, tenemos que:

$$S = \mathbf{b}A \cup \mathbf{a}B \cup \varepsilon$$

$$A = \mathbf{aba}S$$

$$B = \mathbf{bab}S$$

Reemplazando A y B en S tenemos:

$$S = \mathbf{baba}S \cup \mathbf{abab}S \cup \varepsilon = (\mathbf{baba} \cup \mathbf{abab})S \cup \varepsilon$$

Y aplicando el Lema de Arden resulta finalmente:

$$S = L(G) = (\mathbf{baba} \cup \mathbf{abab})^*$$

Ejemplo 3

Obtener una expresión regular para el lenguaje generado por la gramática regular dada por las producciones siguientes:

$$S \rightarrow \mathbf{a}S \mid \mathbf{b}A \mid \mathbf{b}$$

$$A \rightarrow \mathbf{c}B$$

$$B \rightarrow \mathbf{a}S$$

Expresando las ecuaciones respectivas, tenemos:

$$S = \mathbf{a}S \cup \mathbf{b}A \cup \mathbf{b}$$

$$A = \mathbf{c}B$$

$$B = \mathbf{a}S$$

Reemplazando A y B en S tenemos:

$$S = \mathbf{a}S \cup \mathbf{bca}S \cup \mathbf{b} = (\mathbf{a} \cup \mathbf{bca})S \cup \mathbf{b}$$

Y aplicando el Lema de Arden resulta:

$$S = L(G) = (\mathbf{a} \cup \mathbf{bca})^* \mathbf{b}$$

Ejemplo 4

Obtener una expresión regular para el lenguaje generado por la gramática dada por las producciones:

$$S \rightarrow \mathbf{ab}A \mid B \mid \mathbf{ba}B \mid \varepsilon$$

$$A \rightarrow \mathbf{b}S \mid \mathbf{b}$$

$$B \rightarrow \mathbf{a}S$$

Expresando las ecuaciones respectivas, tenemos:

$$S = \mathbf{ab}A \cup B \cup \mathbf{ba}B \cup \varepsilon$$

$$A = \mathbf{b}S \cup \mathbf{b}$$

$$B = \mathbf{a}S$$

Reemplazando A y B en S tenemos:

$$S = \mathbf{ab}(\mathbf{b}S \cup \mathbf{b}) \cup \mathbf{a}S \cup \mathbf{baa}S \cup \varepsilon$$

Reagrupando los términos y aplicando el Lema de Arden resulta:

$$S = L(G) = (\mathbf{abb} \cup \mathbf{a} \cup \mathbf{baa})^* (\mathbf{abb} \cup \varepsilon)$$

Ejemplo 5

Construir una expresión regular para el lenguaje generado por la gramática dada por las producciones:

$$S \rightarrow \mathbf{a}S \mid \mathbf{b}A$$

$$A \rightarrow \mathbf{b}S$$

Pretender resolver el sistema de ecuaciones resultante nos dará como solución un lenguaje vacío, ya que esta gramática no puede generar ninguna cadena, puesto que no existe ninguna producción terminal (que no tenga símbolos No terminales).

Teorema

Un lenguaje L es regular si y sólo si está generado por una gramática regular.

Consecuentemente, por todo lo que hemos visto, tenemos tres maneras equivalentes de especificación de un lenguaje: las expresiones regulares, los autómatas finitos y las gramáticas regulares.

Gramáticas Regulares Reversas

Una gramática cuyas producciones sean de la forma $A \rightarrow w$, donde $A \in N$, y en donde $w \in (\varepsilon \cup N) \cdot \sum^*$, (esto es: *w* puede iniciar con uno o ningún **SNT** seguidos de ninguno, uno o varios Símbolos Terminales) se le llama Gramática Regular Reversa, puesto que genera las cadenas de un lenguaje regular, pero en sentido contrario, es decir de derecha a izquierda.

Ejemplo

Para obtener la gramática regular reversa que genere el lenguaje $L_1 = \mathbf{a}^*\mathbf{b} \cup \mathbf{b}^*\mathbf{a}$, se requieren de las siguientes producciones:

$$S \rightarrow A\mathbf{a} \mid B\mathbf{b}$$

$$A \rightarrow A\mathbf{b} \mid \varepsilon$$

$$B \rightarrow B\mathbf{a} \mid \varepsilon$$

Para generar la cadena w = **aaab**, tenemos la siguiente derivación, la cual inicia con el último símbolo y se concatenan símbolos a la izquierda, hasta terminar con el primero:

$$S \Rightarrow B\mathbf{b} \Rightarrow B\mathbf{ab} \Rightarrow B\mathbf{aab} \Rightarrow B\mathbf{aaab} \Rightarrow \mathbf{aaab}$$

Compare esta gramática con la siguiente Gramática Regular *Derecha Análoga*:

$$S \rightarrow \mathbf{a}A \mid \mathbf{b}B$$

$$A \rightarrow \mathbf{b}A \mid \varepsilon$$

$$B \rightarrow \mathbf{a}B \mid \varepsilon$$

Y la derivación siguiente:

$$S \Rightarrow \mathbf{b}B \Rightarrow \mathbf{ba}B \Rightarrow \mathbf{baa}B \Rightarrow \mathbf{baaa}B \Rightarrow \mathbf{baaa}$$

Resulta fácil comprobar en este caso que el lenguaje generado es: $L_2 = \mathbf{ab}^* \cup \mathbf{ba}^*$.

Por lo tanto, podemos concluir que una Gramática Reversa genera al lenguaje inverso de la Gramática *Derecha* Análoga, es decir $L_1 = L_2^R$.

Construir el diagrama de transiciones del **AF** que acepte el lenguaje generado por una gramática regular reversa también debe hacerse en sentido contrario, en este caso el símbolo inicial S representa el estado de aceptación y cada producción de la forma $S \rightarrow A\mathbf{a}$, significa que existe una transición del estado A al estado S etiquetada con el símbolo **a**, compare las figuras 6.5 y 6.6, que aceptan a los lenguajes $L_1 = \mathbf{a}^*\mathbf{b} \cup \mathbf{b}^*\mathbf{a}$, y $L_2 = \mathbf{ab}^* \cup \mathbf{ba}^*$, respectivamente.

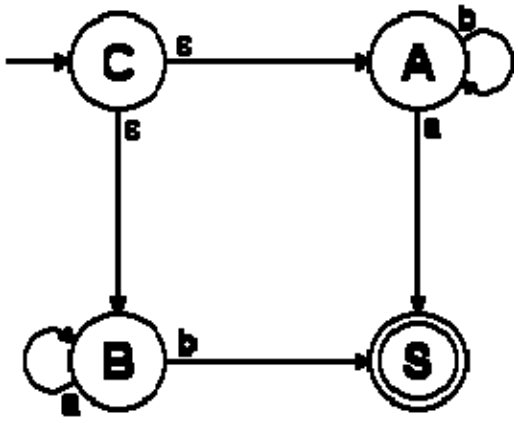


Figura 6.5

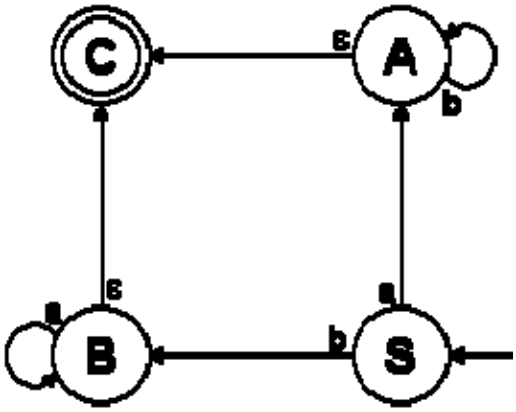


Figura 6.6

De esta manera tenemos una técnica para construir gramáticas regulares y autómatas finitos para un lenguaje dado, si conocemos el análogo para el lenguaje inverso.

Ejemplo

Obtener la expresión regular del lenguaje que genera la siguiente gramática regular reversa:

$$S \rightarrow S\mathbf{a} \mid A\mathbf{b}$$

$$A \rightarrow B\mathbf{b} \mid \mathbf{a}$$

$$B \rightarrow B\mathbf{a} \mid \mathbf{b}$$

Expresando las ecuaciones respectivas, tenemos:

$$S = S\mathbf{a} \cup A\mathbf{b}$$

$$A = B\mathbf{b} \cup \mathbf{a}$$

$$B = B\mathbf{a} \cup \mathbf{b}$$

Ahora, para resolver este sistema de ecuaciones necesitamos la versión dual del Lema de Arden que dice:

Lema Dual de Arden

Una ecuación de la forma $A = A\mathbf{r} \cup \mathbf{s}$, tiene como solución única a: $A = \mathbf{sr}^*$, donde **r** y **s** son dos expresiones regulares cualesquiera que no contienen a A.

Por lo tanto, tenemos que $B = \mathbf{ba}^*$, entonces $A = \mathbf{ba}^*\mathbf{b} \cup \mathbf{a}$ y $S = S\mathbf{a} \cup (\mathbf{ba}^*\mathbf{b} \cup \mathbf{a})\mathbf{b}$, que nos resulta con: $L(G) = S = (\mathbf{ba}^*\mathbf{b} \cup \mathbf{a})\mathbf{ba}^*$.

Propiedades de las Gramáticas Regulares

Unión

Sea G_1, una Gramática Regular con símbolo inicial S_1, que genera al lenguaje L_1 y sea G_2, una Gramática Regular con símbolo inicial S_2, que genera al lenguaje L_2, entonces la gramática G_3 que acepta al lenguaje unión $L_1 \cup L_2$ es una Gramática Regular y se obtiene juntando las producciones de G_1 y G_2, y agregando las siguientes producciones: $S_3 \rightarrow S_1 \mid S_2$.

Ejemplo

Considérese la gramática regular G_1 que genera al lenguaje $L_1 = \mathbf{ba}^*\mathbf{b}$, cuyas producciones son:

$$S_1 \rightarrow \mathbf{b}A_1$$

$$A_1 \rightarrow \mathbf{a}A_1 \mid \mathbf{b}$$

Y la gramática regular G_2 que genera al lenguaje $L_2 = \mathbf{b}^*\mathbf{ab}$, con producciones:

$$S_2 \rightarrow \mathbf{b}S_2 \mid \mathbf{ab}$$

Entonces la gramática regular G_3 que genera al lenguaje $L_3 = \mathbf{ba}^*\mathbf{b} \cup \mathbf{b}^*\mathbf{ab}$, está dada por las producciones:

$$S_3 \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow \mathbf{b}A_1$$

$$A_1 \rightarrow \mathbf{a}A_1 \mid \mathbf{b}$$

$$S_2 \rightarrow \mathbf{b}S_2 \mid \mathbf{ab}$$

Concatenación

Sea G_1, una Gramática Regular con símbolo inicial S_1, que genera al lenguaje L_1 y sea G_2, una Gramática Regular con símbolo inicial S_2, que genera al lenguaje L_2, entonces la gramática G_3, con símbolo inicial S_1, que acepta al lenguaje L_1L_2 es una Gramática Regular y se obtiene juntando las producciones de G_1 y G_2, y modificando las producciones terminales de G_1 de la siguiente manera: Cada producción de la forma: $A \rightarrow w$, se reemplaza por otra de la forma: $A \rightarrow wS_2$.

Ejemplo

Considérese la gramática regular G_1 que genera al lenguaje $L_1 = \mathbf{ba}^+ \cup \mathbf{a}^+\mathbf{b}$, cuyas producciones son:

$$S_1 \rightarrow \mathbf{b}A_1 \mid \mathbf{a}B_1$$
$$A_1 \rightarrow \mathbf{a}A_1 \mid \mathbf{a}$$
$$B_1 \rightarrow \mathbf{a}B_1 \mid \mathbf{b}$$

Y la gramática regular G_2 que genera al lenguaje $L_2 = (\mathbf{a} \cup \mathbf{b})^*$, con producciones:

$$S_2 \rightarrow \mathbf{b}S_2 \mid \mathbf{a}S_2 \mid \varepsilon$$

Entonces la gramática regular G_3 que genera al lenguaje $L_3 = (\mathbf{ba}^+ \cup \mathbf{a}^+\mathbf{b})(\mathbf{a} \cup \mathbf{b})^*$, está dada por las producciones:

$$S_1 \rightarrow \mathbf{b}A_1 \mid \mathbf{a}B_1$$
$$A_1 \rightarrow \mathbf{a}A_1 \mid \mathbf{a}S_2$$
$$B_1 \rightarrow \mathbf{a}B_1 \mid \mathbf{b}S_2$$
$$S_2 \rightarrow \mathbf{b}S_2 \mid \mathbf{a}S_2 \mid \varepsilon$$

Cerradura

Finalmente, sea G_1, una Gramática Regular con símbolo inicial S_1, que genera al lenguaje L_1, entonces la gramática G_3 que acepta al lenguaje $L_3 = L_1^*$ es una Gramática Regular cuyo símbolo inicial es S, y que se obtiene modificando las producciones terminales de G_1 de la siguiente manera: Cada producción de la forma: $A \rightarrow w$, se reemplaza por otra producción de la forma: $A \rightarrow wS$, y agregando las producciones del símbolo inicial: $S \rightarrow S_1 \mid \varepsilon$.

Ejemplo

Considérese la gramática regular G_1 que genera al lenguaje $L_1 = \mathbf{ba}^+ \cup \mathbf{a}$, cuyas producciones son:

$$S_1 \rightarrow \mathbf{b}A_1 \mid \mathbf{a}$$

$$A_1 \rightarrow \mathbf{a}A_1 \mid \mathbf{a}$$

Entonces la gramática regular G_3 que genera al lenguaje $L_3 = (\mathbf{ba}^+ \cup \mathbf{a})^*$, está dada por las producciones:

$$S \rightarrow S_1 \mid \varepsilon$$

$$S_1 \rightarrow \mathbf{b}A_1 \mid \mathbf{a}S$$

$$A_1 \rightarrow \mathbf{a}A_1 \mid \mathbf{a}S$$

Se invita al lector a que compare estos tres ejemplos con los casos vistos en el capítulo anterior relativos a la construcción de **AFN** para que constate el paralelismo que existe entre ambas representaciones, sin embargo, no resulta sencillo construir una **GR** para generar el complemento de un lenguaje generado por una gramática dada.

Preguntas

a) ¿Existe algún Lenguaje Regular que no pueda ser generado por una **GR**?

b) ¿Cómo se puede construir una **GR** que genere el lenguaje que sea la concatenación de dos lenguajes regulares cuyas gramáticas son conocidas?

c) ¿Puede construirse una **GR** para generar un lenguaje que sea la intersección de dos lenguajes regulares?

d) ¿Puede construirse una **GR** para generar el lenguaje complemento de un lenguaje regular dado?

e) ¿Para quiénes son útiles las Gramáticas Regulares Reversas?

Ejercicios Capítulo 6

6.1 Construir un **AFN** que acepte cada uno de los siguientes lenguajes y encontrar una gramática regular que los genere:

a) L(G) = $\mathbf{a}^*\mathbf{bc}$

b) L(G) = ($\mathbf{a} \cup \mathbf{b}$) $\mathbf{c}^*$

c) L(G) = ($\mathbf{a} \cup \mathbf{b}$)* $\mathbf{c}$

d) L(G) = ($\mathbf{a} \cup \mathbf{b}$)* $\mathbf{c}^*$

e) L(G) = $\mathbf{b}^+ \cup \mathbf{a}\ (\ \mathbf{a} \cup \mathbf{b}\ )^+\ \mathbf{b}$

f) L(G) = $\mathbf{a}^+\mathbf{ba} \cup \mathbf{b}(\ \mathbf{a} \cup \mathbf{b}\ )^*$

6.2 Obtener una gramática regular que genere cada uno de los siguientes lenguajes, sin construir el diagrama de transiciones:

a) L(G) = $\mathbf{a}^*\mathbf{baa}$

b) L(G) = $(\ \mathbf{a} \cup \mathbf{ba} \cup \mathbf{ac})^*$

c) L(G) = $\mathbf{a}^*\mathbf{bc}^+ \cup \mathbf{cb}^+$

d) L(G) = $\mathbf{a}^+\mathbf{b}\ (\ \mathbf{b} \cup \mathbf{a}\ )^+$

e) L(G) = $\mathbf{b}^+\mathbf{a} \cup \mathbf{ab}(\ \mathbf{a} \cup \mathbf{b}\ )^*$

6.3 Construir una gramática regular que genere el lenguaje aceptado por el siguiente **AFN**:

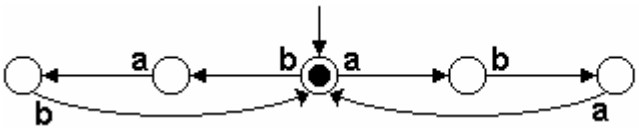


Figura 6.7

6.4 Construir una gramática regular que genere el siguiente lenguaje:

L(G) = { $w \in \{\mathbf{0}, \mathbf{1}\}^*$ | w es la representación binaria de 5, 10, 20, 40, 80, … }

6.5 Construir una gramática regular que genere el lenguaje aceptado por el **AFNG** mostrado en la figura 6.8.

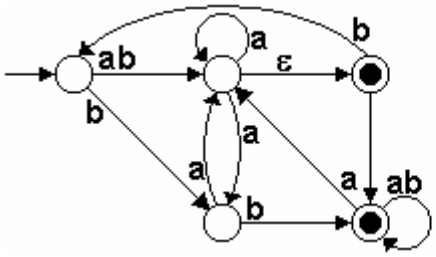


Figura 6.8

6.6 Construir un **AFN** y encontrar la expresión regular para el lenguaje que genera cada una de las siguientes gramáticas regulares:

a) S $\rightarrow$ **b**A | **a**B | **ba**B, A $\rightarrow$ **b**S | **b**, B $\rightarrow$ **a**A

b) S $\rightarrow$ **a**A | **b**B, A $\rightarrow$ **b**S | **b**, B $\rightarrow$ **a**

c) S $\rightarrow$ A | **b**S, A $\rightarrow$ **b**B | **b**C, B $\rightarrow$ **a**C, C $\rightarrow$ **a**

d) S $\rightarrow$ **a**A | **b**B, A $\rightarrow$ **a**A | **b**A | ε, B $\rightarrow$ **a**A | **b**C, C $\rightarrow$ **a**B | **b**C

e) S $\rightarrow$ **a**A | **b**B, A $\rightarrow$ **b**A | **a**S | **b**, B $\rightarrow$ **a**S | **b**C, C $\rightarrow$ **a**B | **b**C

f) S $\rightarrow$ **a**A | **b**C, A $\rightarrow$ **a**A | **b**B, B $\rightarrow$ **a**B | **b**B | **b**, C $\rightarrow$ **a**B | **b**A

g) S $\rightarrow$ **a**C | **b**S, A $\rightarrow$ **a**A | **b**S, B $\rightarrow$ **a**S | **b**A, C $\rightarrow$ **a**B | **b**C | **b**

6.7 Construir un **AFN** y encontrar la expresión regular para el lenguaje que genera la siguiente gramática regular reversa:

$S \rightarrow A\mathbf{b} \mid A\mathbf{a} \mid B\mathbf{b}$, $A \rightarrow S\mathbf{a} \mid \mathbf{b}$, $B \rightarrow S\mathbf{b} \mid \varepsilon$

Capítulo

7

Gramáticas Libres del Contexto

Se define el concepto de Gramática Libre del Contexto, Se analizan diversos algoritmos para la depuración de anomalías en las Gramáticas Libres del Contexto. Se discuten y se analizan las formas normales de Chomsky y de Greibach, así como el algoritmo CYK.

En el capítulo anterior tratamos sobre las gramáticas regulares y surge de manera inmediata una pregunta: ¿Existe otro tipo de gramáticas que nos permitan generar lenguajes que no sean regulares?

Por ejemplo, si consideramos a la gramática siguiente: $S \rightarrow \mathbf{a}S\mathbf{b} \mid \varepsilon$, vemos que no es una Gramática Regular, puesto que no satisface la definición del capítulo anterior; entonces, para saber que lenguaje genera, hagamos un análisis de las cadenas que podemos obtener por medio de esta gramática, para ello, realizamos algunas derivaciones:

$$S \Rightarrow \varepsilon$$

$$S \Rightarrow \mathbf{a}S\mathbf{b} \Rightarrow \mathbf{ab}$$

$$S \Rightarrow \mathbf{a}S\mathbf{b} \Rightarrow \mathbf{aa}S\mathbf{bb} \Rightarrow \mathbf{aabb}$$

$$S \Rightarrow \mathbf{a}S\mathbf{b} \Rightarrow \mathbf{aa}S\mathbf{bb} \Rightarrow \mathbf{aaa}S\mathbf{bbb} \Rightarrow \mathbf{aaabbb}$$

$$S \Rightarrow \mathbf{a}S\mathbf{b} \Rightarrow \mathbf{aa}S\mathbf{bb} \Rightarrow \mathbf{aaa}S\mathbf{bbb} \Rightarrow \mathbf{aaaa}S\mathbf{bbbb} \Rightarrow \mathbf{aaaabbbb}, \text{ etc.}$$

Es fácil llegar a la conclusión que esta gramática genera al lenguaje formado por las cadenas que tienen una cierta cantidad de **a**s seguida por la misma cantidad de **b**s, es decir, del lenguaje siguiente:

$$L = \{ \varepsilon, \mathbf{ab}, \mathbf{aabb}, \mathbf{aaabbb}, \mathbf{aaaabbbb}, \ldots \} = \{ \mathbf{a}^n\mathbf{b}^n \mid n \geq 0 \}$$

No confundir este lenguaje con el Lenguaje Regular $\mathbf{a}^*\mathbf{b}^*$, ya que este nuevo lenguaje solamente contiene a las cadenas de aquel que satisfacen la restricción dada, y es por esto que se necesita de una gramática más poderosa para generarlo.

Definición de Gramática Libre del Contexto

Supongamos que en una gramática se permiten producciones de la forma $A \rightarrow w$, donde *w* puede contener cero, uno o más **SNT** y además se permite que éstos aparezcan en cualquier parte de la cadena *w*. Obviamente ésta no sería una Gramática Regular.

A este tipo de gramática se le denomina *Gramática Libre del Contexto* o también *Gramática Independiente del Contexto* (**GIC**). Formalmente definimos a una **GIC** como la cuarteta G = (N, $\sum$, S, P), donde $\sum$ es un alfabeto, N es la colección de símbolos no terminales, S es el símbolo inicial ($S \in N$) y P es el conjunto de *Producciones*, que son de la forma $A \rightarrow w$, donde $A \in N$ y $w \in (N \cup \sum)^*$.

Se les llama Libres o Independientes del Contexto a este tipo de Gramáticas, debido a que el símbolo no terminal del lado izquierdo de la producción puede ser reemplazado en una derivación cualquiera, sin ningún tipo de condicionamientos.

Dado que esta definición es más amplia, se puede agregar que toda gramática regular también pertenece a la clase de las **GIC**s.

Notación de Backus-Naur

La notación de Backus-Naur es una meta-sintaxis para la representación de las **GIC**s, en particular, se usa para describir las gramáticas de los lenguajes computacionales, tales como el ALGOL, Pascal y otros, en el diseño de compiladores. Dentro de los metasímbolos utilizados por esta notación se encuentran los siguientes:

- Los paréntesis angulares ⟨ y ⟩, para denotar los nombres de los símbolos no terminales.
- La barra vertical **|** para denotar “o”.
- La doble flecha $\Rightarrow$ para denotar las derivaciones.
- Los dobles dos puntos seguidos de igual **::=** se usan para denotar una producción, también se interpreta como “es definido como”. (equivale a la flecha $\rightarrow$)
- Los paréntesis cuadrados o corchetes **[** y **]** se usan para denotar elementos opcionales.
- Las llaves **{** y **}** se usan para términos repetitivos.

El empleo de esta notación está fuera del objetivo de esta obra, nos limitamos a presentarla como un vínculo hacia otras obras especializadas en compiladores.

Ejemplo

Consideremos una gramática muy simplificada de nuestra lengua española:

⟨enunciado⟩ → ⟨sujeto⟩ ⟨verbo⟩ ⟨predicado⟩ ⟨punto⟩
⟨sujeto⟩ → ⟨artículo⟩ ⟨sustantivo común⟩ | ⟨sustantivo propio⟩
⟨verbo⟩ → quiere | compra | come
⟨predicado⟩ → dulces | croquetas | flores
⟨artículo⟩ → El | Un
⟨sustantivo común⟩ → niño | perro
⟨sustantivo propio⟩ → Juanito | María
⟨punto⟩ → **.**

Utilizando las producciones anteriores podemos generar al siguiente enunciado: "*el niño quiere dulces.*", por medio de la siguiente derivación:

⟨enunciado⟩ ⇒ ⟨sujeto⟩ ⟨verbo⟩ ⟨predicado⟩ ⟨punto⟩
⇒ ⟨artículo⟩ ⟨sustantivo común⟩ ⟨verbo⟩ ⟨predicado⟩ ⟨punto⟩
⇒ El ⟨sustantivo común⟩ ⟨verbo⟩ ⟨predicado⟩ ⟨punto⟩
⇒ El niño ⟨verbo⟩ ⟨predicado⟩ ⟨punto⟩
⇒ El niño quiere ⟨predicado⟩ ⟨punto⟩
⇒ El niño quiere dulces ⟨punto⟩ ⇒ El niño quiere dulces.

Pero con esta gramática también se pueden derivar algunos enunciados sin mucho sentido, como por ejemplo: "*Un perro compra dulces.*", O "*Juanito come flores.*", puesto que las reglas gramaticales son reglas sintácticas y no semánticas. Por otro lado, estas pocas reglas son insuficientes para construir otros enunciados con un significado correcto como por ejemplo: "*El niño quiere un perro.*".

Árboles de Derivación

A veces es útil representar la derivación de una cadena por medio de un árbol, esto nos permite visualizar de qué manera se generó dicha cadena. El árbol de derivación se construye colocando al símbolo inicial en la raíz del árbol, y se crea una rama para cada símbolo de la producción que se utilizó para reemplazar ese símbolo, en el mismo orden en que aparecen. Se procede del mismo modo para cada Nodo que contenga a un **SNT**. Los nodos que contengan símbolos terminales no tienen hijos y serán las hojas del árbol.

Ejemplo 1

La derivación de la cadena: "*El niño quiere dulces.*" se puede representar gráficamente por medio del árbol de derivación mostrado en la figura 7.1.

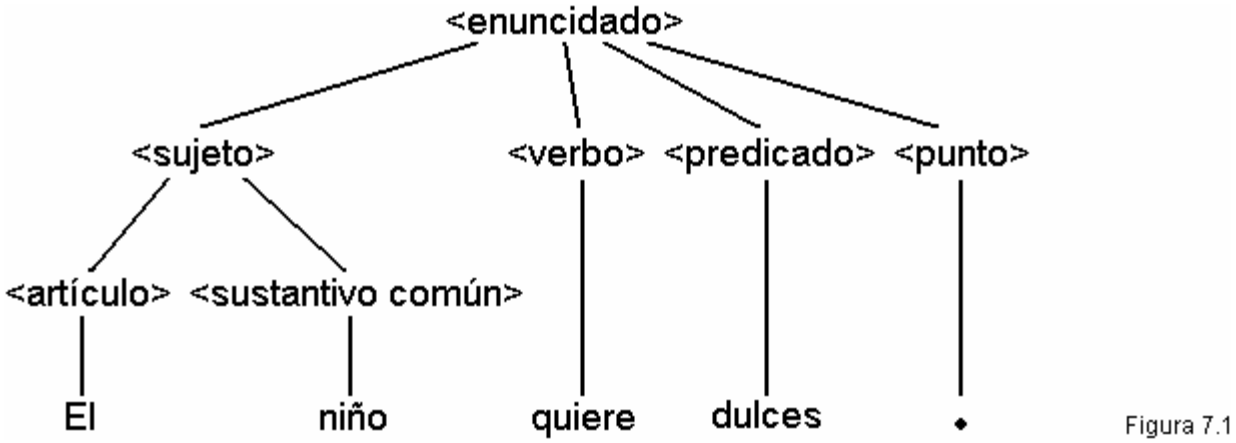


Figura 7.1

Ejemplo 2

Considere la **GIC** siguiente:

$$S \rightarrow AB$$

$$A \rightarrow \mathbf{a}A \mid \mathbf{a}$$

$$B \rightarrow \mathbf{b}B \mid \mathbf{b}$$

A continuación se muestran 3 maneras distintas de derivar a la cadena *w* = **aabbb**:

$$S \Rightarrow AB \Rightarrow A\mathbf{b}B \Rightarrow A\mathbf{bb}B \Rightarrow A\mathbf{bbb} \Rightarrow \mathbf{a}A\mathbf{bbb} \Rightarrow \mathbf{aabbb}$$

$$S \Rightarrow AB \Rightarrow \mathbf{a}AB \Rightarrow \mathbf{aa}B \Rightarrow \mathbf{aab}B \Rightarrow \mathbf{aabb}B \Rightarrow \mathbf{aabbb}$$

$$S \Rightarrow AB \Rightarrow \mathbf{a}AB \Rightarrow \mathbf{a}A\mathbf{b}B \Rightarrow \mathbf{a}A\mathbf{bb}B \Rightarrow \mathbf{a}A\mathbf{bbb} \Rightarrow \mathbf{aabbb}$$

Podemos verificar que el árbol de derivación de esta cadena es el mismo, para todas las derivaciones, y es el que se muestra en la figura 7.2 a continuación:

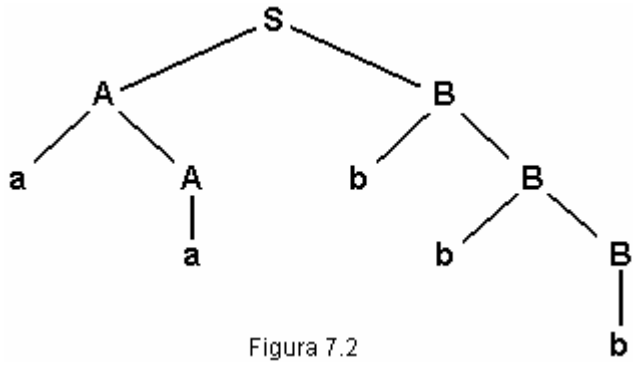


Figura 7.2

Se puede comprobar que esta propiedad se mantiene para cualquier otra cadena que generemos con esta gramática, en que, aún cuando existan distintas derivaciones, encontramos que siempre se tiene una misma representación gráfica para todos los casos. Esta característica nos permite afirmar que la gramática en cuestión no es *Ambigua*.

Ambigüedad

Ahora consideremos a la **GIC** siguiente, por medio de la cual podemos generar cadenas del lenguaje L = { $w \in$ {**a**, **b**}* | $N_a(w) = N_b(w)$ }.

$$S \rightarrow S\mathbf{ab} \mid S\mathbf{ba} \mid \mathbf{b}S\mathbf{a} \mid \mathbf{a}S\mathbf{b} \mid \varepsilon$$

Por ejemplo, la cadena *w* = **ababba** se puede derivar de estas dos formas:

$$S \Rightarrow S\mathbf{ba} \Rightarrow S\mathbf{abba} \Rightarrow S\mathbf{ababba} \Rightarrow \mathbf{ababba}$$

$$S \Rightarrow S\mathbf{ba} \Rightarrow \mathbf{a}S\mathbf{bba} \Rightarrow \mathbf{ab}S\mathbf{abba} \Rightarrow \mathbf{ababba}$$

En este caso, los árboles de derivación obtenidos para cada alternativa, son distintos uno del otro, y sin embargo, ambos producen la misma cadena, como se aprecia en la siguiente figura:

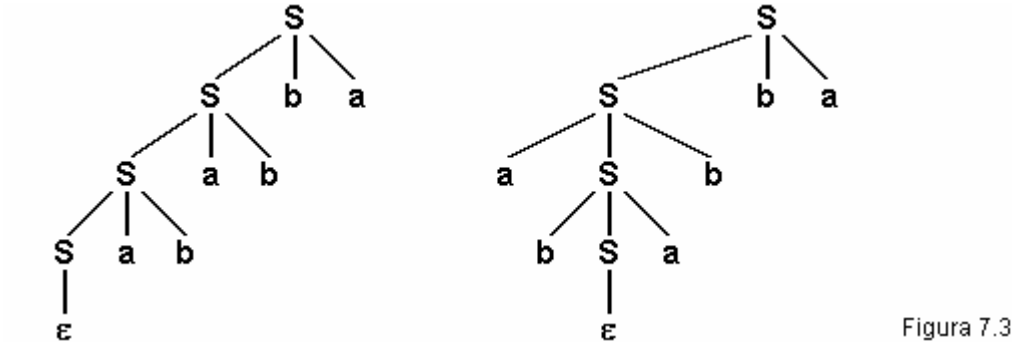


Figura 7.3

Cuando hay dos o más árboles de derivación distintos para una misma cadena, se dice que la gramática en cuestión es *Ambigua*.

Un ejemplo de Ambigüedad en nuestro idioma sería la frase "*Juan vio a una muchacha con un telescopio*", ya que puede interpretarse como que Juan observó a la muchacha por medio del telescopio, pero también se entiende como que la muchacha era quién portaba un telescopio, en estos casos, se requiere buscar otras alternativas equivalentes que estén libres de ambigüedades, por ejemplo, podemos decir "*Juan vio a una muchacha a través de un telescopio*", para la primera interpretación de la frase, o bien, "*Juan vio a una muchacha que portaba un telescopio*", para el segundo caso.

Ejemplo 1

Consideremos una parte de la gramática de un lenguaje de programación:

$$S \rightarrow \textbf{IF } A \textbf{ THEN } S \mid \textbf{IF } A \textbf{ THEN } S \textbf{ ELSE } S \mid \text{otras}$$

Donde A es una expresión lógica y S es una instrucción del programa, entonces podemos generar la siguiente derivación:

$$\textbf{IF } A \textbf{ THEN IF } A \textbf{ THEN } S \textbf{ ELSE } S$$

Aquí se presenta un problema de ambigüedad: ¿El **ELSE** corresponde al primer **IF** o al segundo? Por una convención, se considera que el **ELSE** pertenece al segundo **IF**, pero esto es una regla Semántica, no de sintaxis, para coincidir con ese significado, se debe utilizar la siguiente gramática equivalente, la cual no tiene ambigüedades:

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow \textbf{IF } A \textbf{ THEN } S_1 \textbf{ ELSE } S_1 \mid \text{otras}$$

$$S_2 \rightarrow \textbf{IF } A \textbf{ THEN } S \mid \textbf{IF } A \textbf{ THEN } S_1 \textbf{ ELSE } S_2$$

Ejemplo 2

Considere la siguiente gramática, donde N = {S} y Σ = { **+**, ×, **(**, **)**, $\mathbf{n_1}$, $\mathbf{n_2}$, $\mathbf{n_3}$ }:

$$S \rightarrow S \textbf{ + } S \mid S \times S \mid \textbf{(}S\textbf{)} \mid \mathbf{n_1} \mid \mathbf{n_2} \mid \mathbf{n_3}$$

Con esta **GIC** podemos hacer la derivación que se muestra a continuación, en la que se muestra correctamente que primero se realiza la suma y posteriormente el producto:

$$S \Rightarrow S \times S \Rightarrow \textbf{(}S\textbf{)} \times S \Rightarrow \textbf{(}S \textbf{ + } S\textbf{)} \times S \Rightarrow \textbf{(}\mathbf{n_1} \textbf{ + } S\textbf{)} \times S \Rightarrow \textbf{(}\mathbf{n_1} \textbf{ + } \mathbf{n_2}\textbf{)} \times S \Rightarrow \textbf{(}\mathbf{n_1} \textbf{ + } \mathbf{n_2}\textbf{)}\times\mathbf{n_3}$$

Sin embargo, esta gramática es Ambigua, ya que la cadena $\mathbf{n_1} \textbf{ + } \mathbf{n_2} \times \mathbf{n_3}$ puede ser generada de dos maneras distintas, con diferentes interpretaciones, en un primer caso se trata de la suma de un producto, mientras que en el segundo caso es el producto de una suma:

$$S \Rightarrow S \textbf{ + } S \Rightarrow S \textbf{ + } S \times S \Rightarrow \mathbf{n_1} \textbf{ + } S \times S \Rightarrow \mathbf{n_1} \textbf{ + } \mathbf{n_2} \times S \Rightarrow \mathbf{n_1} \textbf{ + } \mathbf{n_2} \times \mathbf{n_3}$$

$$S \Rightarrow S \times S \Rightarrow S \textbf{ + } S \times S \Rightarrow \mathbf{n_1} \textbf{ + } S \times S \Rightarrow \mathbf{n_1} \textbf{ + } \mathbf{n_2} \times S \Rightarrow \mathbf{n_1} \textbf{ + } \mathbf{n_2} \times \mathbf{n_3}$$

Los árboles de derivación respectivos se muestran en la figura 7.4:

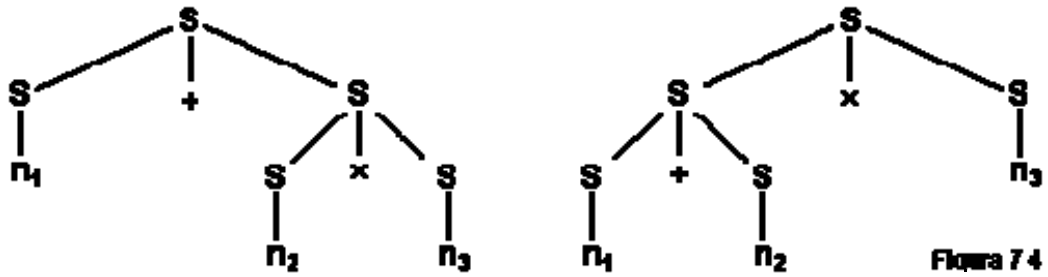


Figura 7.4

En este caso, el emplear un solo **SNT** es la causa de esta ambigüedad, por lo que un recurso generalmente utilizado para encontrar una gramática equivalente no ambigua consiste en la introducción de varios **SNT**s adicionales.

La gramática siguiente es una gramática equivalente no ambigua:

$$S \rightarrow S + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow (S) \mid n_1 \mid n_2 \mid n_3$$

La derivación de la cadena $(n_1 + n_2) \times n_3$ requiere de más pasos:

$$S \Rightarrow T \Rightarrow T \times F \Rightarrow F \times F \Rightarrow (S) \times F \Rightarrow (S + T) \times F \Rightarrow (T + T) \times F \Rightarrow (F + T) \times F$$

$$\Rightarrow (F + F) \times F \Rightarrow (n_1 + F) \times F \Rightarrow (n_1 + n_2) \times F \Rightarrow (n_1 + n_2) \times n_3$$

Sin embargo, la ventaja de esta nueva gramática es que existe una sola forma de derivar la cadena: $n_1 + n_2 \times n_3$, que se interpreta como la suma de un producto, tal como se entiende si respetamos la precedencia de la multiplicación sobre la suma.

$$S \Rightarrow S + T \Rightarrow S + T \times F \Rightarrow S + T \times F \Rightarrow T + T \times F \Rightarrow F + T \times F \Rightarrow F + F \times F$$

$$\Rightarrow n_1 + F \times F \Rightarrow n_1 + n_2 \times F \Rightarrow n_1 + n_2 \times n_3$$

Depuración de Gramáticas Independiente del Contexto

En muchas ocasiones, al obtener el conjunto de producciones que constituyen una gramática, encontramos que algunas producciones pueden resultar superfluas, así como también algunos **SNT**s pueden ser inútiles.

Por eso es necesario contar con una metodología que nos permita eliminar tales producciones y simplificar la gramática sin afectarla. Este proceso no pretende encontrar la gramática óptima, pero si nos permite eliminar las anomalías de construcción gramatical más perjudiciales.

Con el fin de observar la aplicación de los siguientes algoritmos, considérese la siguiente gramática como modelo, la cual presenta las diversas anomalías a detectar y erradicar:

$S \rightarrow \mathbf{a}A \mid B \mid D$

$A \rightarrow \mathbf{a}A \mid \mathbf{b}A \mid B \mid \varepsilon$

$B \rightarrow \mathbf{b} \mid \mathbf{a}F$

$C \rightarrow \mathbf{abd}$

$E \rightarrow \mathbf{c}E$

$F \rightarrow \varepsilon$

Algoritmo 1

El primer paso consiste en sustituir a las producciones épsilon, bajo el siguiente criterio: debemos quitar a todas las producciones épsilon, con la posible excepción de la producción $S \rightarrow \varepsilon$, ya que de presentarse este caso, significa que $\varepsilon \in L(G)$.

Sea la gramática G = (N, ∑, S, P) que contiene producciones épsilon anulables, se puede obtener una gramática equivalente G' = (N', ∑, S, P') que no tenga producciones épsilon, de modo que L(G) = L(G'), aplicando el siguiente criterio:

Para cada producción de la forma $A \rightarrow \varepsilon$, que se desea eliminar, se seleccionan todas las producciones en las que aparece el símbolo A del lado derecho y se substituye al **SNT** A por épsilon, obteniendo las nuevas producciones que se agregarán a la gramática G', a cambio de la eliminación de la producción $A \rightarrow \varepsilon$.

Se repite el procedimiento para las producciones épsilon que surjan de esta substitución, excepto en los casos en que se obtengan duplicados de alguna producción épsilon, que previamente se hubiera eliminado, que ya no se volverán a tomar en cuenta; así como a las producciones del tipo $A \rightarrow A$, que se generen durante este proceso. Y a las que simplemente descartamos de la gramática, por que no contribuyen en nada a la gramática.

Sobre la Gramática Modelo

En nuestro ejemplo existen dos producciones épsilon: $A \rightarrow \varepsilon$ y $F \rightarrow \varepsilon$, ahora analicemos las siguientes derivaciones, en donde interviene el símbolo No Terminal A y se aplica la primera producción épsilon:

$$S \Rightarrow \mathbf{a}A \Rightarrow \mathbf{a}$$

$$A \Rightarrow \mathbf{a}A \Rightarrow \mathbf{a}$$

$$A \Rightarrow \mathbf{b}A \Rightarrow \mathbf{b}$$

Entonces, para eliminar la producción $A \rightarrow \varepsilon$, debemos agregar, a cambio, las tres producciones que surgen de estas tres derivaciones: $S \rightarrow \mathbf{a}$, $A \rightarrow \mathbf{a}$ y $A \rightarrow \mathbf{b}$.

Ahora analicemos la siguiente derivación, en donde interviene el símbolo No Terminal F y se aplica la segunda producción épsilon:

$$B \Rightarrow \mathbf{a}F \Rightarrow \mathbf{a}$$

Por lo tanto, se agrega la producción $B \rightarrow \mathbf{a}$, en sustitución de la producción $F \rightarrow \varepsilon$, la cual se elimina, quedando finalmente la gramática G' equivalente siguiente:

$$S \rightarrow \mathbf{a}A \mid B \mid D \mid \mathbf{a}$$

$$A \rightarrow \mathbf{a}A \mid \mathbf{b}A \mid B \mid \mathbf{a} \mid \mathbf{b}$$

$$B \rightarrow \mathbf{b} \mid \mathbf{a}F \mid \mathbf{a}$$

$$C \rightarrow \mathbf{abd}$$

$$E \rightarrow \mathbf{c}E$$

Consecuentemente, se tiene que N' = { S, A, B, C, E }, mientras que $\sum$ no cambia.

Algoritmo 2

El siguiente paso consiste en eliminar *Símbolos No Terminales Superfluos*, es decir, aquellos **SNT**s que no van a participar en la producción de alguna cadena generada por la gramática en cuestión.

Sea la gramática G = (N, $\sum$, S, P) que contiene *Símbolos No Terminales Superfluos*, podemos obtener una gramática equivalente G' = (N', $\sum$', S, P') que no los tenga, de modo que L(G) = L(G'), por medio del siguiente algoritmo:

1. Se inicializa N' = $\varnothing$ y a P' con todas las producciones de G que sean de la forma: $A \rightarrow w$, donde *w* contiene solamente Símbolos Terminales del lado derecho, con la posible excepción de $S \rightarrow \varepsilon$, es decir, $w \in \sum^*$.
2. Se añaden a N' todos los no terminales que aparezcan en el lado izquierdo de alguna de las producciones que fueron incorporadas a P' en el paso anterior.

3. De entre las producciones restantes de G, se agregan a P' a aquellas que contengan a cualquiera de los **SNT**s ya incluidos en N', además de cualquier otro Símbolo Terminal, es decir, añadir a P' las producciones de la forma $A \rightarrow w$ donde $w \in (N' \cup \sum)^*$.
4. Se repiten los pasos 2 y 3 hasta que ya no se puedan añadir más **SNT**s a N' ni más producciones a P'.
5. Se crea a $\sum'$ con todos los Símbolos Terminales que aparezcan en el lado derecho de alguna de las producciones anteriores.

Sobre la Gramática Modelo

Aplicando el paso 1 de este algoritmo a la gramática modelo obtenida después del primer algoritmo, incorporamos a P' las producciones que tengan solamente símbolos terminales en el lado derecho, y que son las producciones siguientes:

$$S \rightarrow \mathbf{a}$$

$$A \rightarrow \mathbf{a} \mid \mathbf{b}$$

$$B \rightarrow \mathbf{b} \mid \mathbf{a}$$

$$C \rightarrow \mathbf{abd}$$

Entonces agregamos al conjunto N' los Símbolos No Terminales que aparecen del lado izquierdo en las producciones anteriores, es decir: N' = { S, A, B, C }.

Luego, analizando las producciones restantes que contengan a alguno de los No Terminales de N' junto con cualquier terminal, encontramos las cinco producciones siguientes, que se agregarán a P' junto con las 6 anteriores:

$$S \rightarrow \mathbf{a}A \mid B$$

$$A \rightarrow \mathbf{a}A \mid \mathbf{b}A \mid B$$

Como ya no hay más Símbolos No Terminales ni producciones que añadir a nuestra gramática, se da por terminado el proceso creando el conjunto de símbolos terminales: $\sum'$ = { **a**, **b**, **d** }.

Detectamos que los símbolos D, E y F son superfluos (F se volvió superfluo al eliminar la producción $F \rightarrow \varepsilon$) y como consecuencia de lo anterior, desaparece también el Símbolo Terminal **c**, junto con las siguientes producciones: $S \rightarrow D$, $E \rightarrow \mathbf{c}E$ y $B \rightarrow \mathbf{a}F$.

Dado que se trata de una gramática regular, podemos verificar que esta eliminación se relaciona con el hecho de quitar estados no deseados de un autómata finito.

Algoritmo 3

El tercer paso consiste en eliminar las producciones unitarias, las cuales son de la forma $A \rightarrow B$, donde A y B son **SNT**s (y que equivalen a las transiciones épsilon de un autómata finito), ya que estas producciones son consideradas como *improductivas* y es deseable que sean erradicadas.

Sea la gramática G = (N, $\sum$, S, P) que contiene *Producciones Unitarias*, podemos obtener una gramática equivalente G' = (N, $\sum$, S, P') que no las tenga, de modo que L(G) = L(G'), por medio del siguiente criterio:

Sea la producción unitaria $A \rightarrow B$, si las producciones de B son de la forma $B \rightarrow w$, entonces aplicamos la siguiente derivación: $A \Rightarrow B \Rightarrow w$, para cada producción de B y se puede observar que es posible reemplazar a la producción unitaria $A \rightarrow B$ por todas las producciones de la forma: $A \rightarrow w$ que se obtengan de las derivaciones anteriores.

Sobre la Gramática Modelo

En nuestra gramática modelo existen dos producciones unitarias: $S \rightarrow B$ y $A \rightarrow B$.

La eliminación de estas producciones unitarias se hace analizando las derivaciones que haya para todas las producciones del **SNT** B, que en este caso son: $B \rightarrow$ **b** | **a**, por lo tanto, las derivaciones son:

$S \Rightarrow B \Rightarrow$ **b** $\quad\quad$ $S \Rightarrow B \Rightarrow$ **a**

$A \Rightarrow B \Rightarrow$ **b** $\quad\quad$ $A \Rightarrow B \Rightarrow$ **a**

Este procedimiento nos genera las nuevas producciones, que reemplazarán a las producciones unitarias anteriores: $S \rightarrow$ **b** | **a** y $A \rightarrow$ **b** | **a**.

Después de reemplazar las producciones unitarias, la gramática equivalente queda así (omitiendo las producciones duplicadas):

$S \rightarrow$ **a**A | **b** | **a**

$A \rightarrow$ **a**A | **b**A | **a** | **b**

$B \rightarrow$ **b** | **a**

$C \rightarrow$ **abd**

Algoritmo 4

El último paso consiste en eliminar las producciones inútiles, es decir, las que nunca se van a poder aplicar, ya que no son accesibles desde el símbolo inicial S.

Sea la gramática G = (N, $\sum$, S, P) que contiene *Producciones Inútiles*, podemos obtener una gramática equivalente G' = (N', $\sum$', S, P') que no las tenga, de modo que L(G) = L(G'), por medio del siguiente algoritmo:

1. Se inicializan los siguientes conjuntos: N' = {S}, $\sum$' = $\varnothing$ y P' = $\varnothing$
2. Para cada **SNT** A $\in$ N', con producciones A $\rightarrow$ *w*, se hace lo siguiente:
   - Agrega la producción A $\rightarrow$ *w* al conjunto P'
   - Agrega todos los Símbolos No Terminales de *w*, al conjunto N'
   - Agrega todos los Símbolos Terminales de *w*, al conjunto $\sum$'
3. Se repite el paso 2 hasta que ya no se puedan añadir más producciones.

Sobre la Gramática Modelo

Aplicando este algoritmo en la gramática modelo, primero inicializamos al conjunto N' con el símbolo N' = {S}, entonces agregamos a P' las tres producciones que tiene el símbolo S:

$$S \rightarrow \mathbf{a}A \mid \mathbf{b} \mid \mathbf{a}$$

Agregamos ahora los Símbolos Terminales a $\sum$' y No Terminales a N', obteniendo entonces: N' = { S, A } y $\sum$' = { **a**, **b** }, ahora agregamos las producciones existentes para A:

$$A \rightarrow \mathbf{a}A \mid \mathbf{b}A \mid \mathbf{a} \mid \mathbf{b}$$

Como esto ya no modifica ni a N' ni a $\sum$', entonces damos por terminado el proceso, encontrando que las producciones inútiles que desaparecen son:

$$B \rightarrow \mathbf{b} \mid \mathbf{a}$$

$$C \rightarrow \mathbf{abd}$$

Ya que B y C nunca formarán parte de ninguna derivación a partir de S, son consideradas producciones inútiles (y que en un **AF** corresponden a estados inaccesibles) y como consecuencia de lo anterior, también se elimina al símbolo terminal **d** del alfabeto.

Ejemplo 1

Depurar de anomalías la siguiente gramática:

$$S \rightarrow \mathbf{a}A \mid \varepsilon$$
$$A \rightarrow \mathbf{b}A \mid \mathbf{a}B \mid \varepsilon$$
$$B \rightarrow \mathbf{b}B$$

Eliminando la producción $A \rightarrow \varepsilon$ ($S \rightarrow \varepsilon$ permanece) queda entonces como sigue:

$$S \rightarrow \mathbf{a}A \mid \mathbf{a} \mid \varepsilon$$
$$A \rightarrow \mathbf{b}A \mid \mathbf{b} \mid \mathbf{a}B$$
$$B \rightarrow \mathbf{b}B$$

Aplicando el algoritmo 2 se elimina el símbolo B, quedando la gramática como:

$$S \rightarrow \mathbf{a}A \mid \mathbf{a} \mid \varepsilon$$
$$A \rightarrow \mathbf{b}A \mid \mathbf{b}$$

Los algoritmos 3 y 4 no modifican el resultado anterior.

Ejemplo 2

Depurar de anomalías la siguiente gramática:

$$S \rightarrow A \mid A\mathbf{a}$$
$$A \rightarrow B$$
$$B \rightarrow C \mid \mathbf{b}$$
$$C \rightarrow \mathbf{b} \mid \mathbf{ab}$$

El algoritmo 1 no aplica, mientras que el algoritmo 2 nos deja igual la gramática, aplicando el tercer algoritmo nos elimina las producciones unitarias, quedando:

$$S \rightarrow \mathbf{b} \mid \mathbf{ab} \mid A\mathbf{a}$$
$$A \rightarrow \mathbf{b} \mid \mathbf{ab}$$
$$B \rightarrow \mathbf{b} \mid \mathbf{ab}$$
$$C \rightarrow \mathbf{b} \mid \mathbf{ab}$$

Y aplicando el cuarto algoritmo desaparecen los **SNT**s B y C, quedando solo:

$$S \rightarrow \mathbf{b} \mid \mathbf{ab} \mid A\mathbf{a}$$
$$A \rightarrow \mathbf{b} \mid \mathbf{ab}$$

Una simplificación adicional consiste en sustituir al símbolo A en las producciones de S, debido a que contienen terminales solamente, por lo que resulta:

$$S \rightarrow \mathbf{b} \mid \mathbf{ab} \mid \mathbf{ba} \mid \mathbf{aba}$$

Ejemplo 3

Depurar de anomalías la siguiente **GIC**:

$$S \rightarrow A\mathbf{ba}C$$
$$A \rightarrow AB$$
$$B \rightarrow \mathbf{b} \mid \varepsilon$$
$$C \rightarrow D \mid \varepsilon$$
$$D \rightarrow \mathbf{d}$$
$$E \rightarrow \mathbf{a} \mid \varepsilon$$

Eliminando las producciones épsilon, obsérvese que las producciones: $A \rightarrow A$ y $E \rightarrow \varepsilon$ simplemente desaparecen, pues la primera es idéntica, y el símbolo E de la segunda no figura en ninguna producción, por tanto, la **GIC** equivalente queda como sigue:

$$S \rightarrow A\mathbf{ba}C \mid A\mathbf{ba}$$
$$A \rightarrow AB$$
$$B \rightarrow \mathbf{b}$$
$$C \rightarrow D$$
$$D \rightarrow \mathbf{d}$$
$$E \rightarrow \mathbf{a}$$

Ahora aplicando el primer paso del segundo algoritmo, queda como sigue:

$$B \rightarrow \mathbf{b}$$
$$D \rightarrow \mathbf{d}$$
$$E \rightarrow \mathbf{a}$$

Entonces agregamos al conjunto N' los Símbolos No Terminales que aparecen del lado izquierdo en las producciones anteriores: N' = { B, D, E }, lo que nos permite agregar a la producción: $C \rightarrow D$, añadiendo el **SNT** C al conjunto N', pero entonces ya no hay más producciones que añadir y por lo tanto la gramática no existe, ya que no se incluyó el símbolo inicial S en N'. El lenguaje generado por esta **GIC** es $\varnothing$.

Forma Normal de Chomsky[1]

Una Gramática Independiente del Contexto está en la Forma Normal de Chomsky si todas las producciones son de la forma $S \rightarrow \varepsilon$ (donde S es el Símbolo Inicial) o de la forma $A \rightarrow \mathbf{a}$, donde $\mathbf{a} \in \sum$, o de la forma $A \rightarrow BC$; donde A, B y C son **SNT**s, es decir, del lado derecho de las producciones solo se permite que aparezca un solo Símbolo Terminal o dos **SNT**s, con la posible excepción de $S \rightarrow \varepsilon$.

Cualquier **GIC** puede ser transformada a la Forma Normal de Chomsky, primero se tienen que aplicar el procedimiento anterior para depurar la gramática y eliminar las producciones ε, los símbolos inútiles y las producciones unitarias de G.

Ejemplo 1

Sea la **GIC**, sobre el alfabeto Σ = { **a**, **b** }, que genera de manera no ambigua al lenguaje formado por las cadenas no vacías, que contienen la misma cantidad de **a** que **b** y que está libre de anomalías:

$$S \rightarrow \mathbf{b}A \mid \mathbf{a}B$$

$$A \rightarrow \mathbf{b}AA \mid \mathbf{a}S \mid \mathbf{a}$$

$$B \rightarrow \mathbf{a}BB \mid \mathbf{b}S \mid \mathbf{b}$$

Las únicas dos producciones que previamente están normalizadas, y a las cuales no se les debe hacer nada, son: $A \rightarrow \mathbf{a}$ y $B \rightarrow \mathbf{b}$

Ahora debemos definir dos producciones nuevas, para los Símbolos Terminales, que se substituirán en las producciones que no están aún normalizadas:

$$C_a \rightarrow \mathbf{a}$$

$$C_b \rightarrow \mathbf{b}$$

Entonces, podemos expresar las seis producciones restantes así:

$$S \rightarrow C_bA \mid C_aB$$

$$A \rightarrow C_bAA \mid C_aS$$

$$B \rightarrow C_aBB \mid C_bS$$

[1] El lingüista, profesor e intelectual estadounidense Noam Chomsky es fundador de la teoría generativo-transformacional que ha revolucionado la lingüística. Trata la gramática dentro de la teoría general del lenguaje: esto es, Chomsky cree que junto a las reglas gramaticales de cada lengua concreta, existen además unas universales comunes a todas las lenguas, lo que indica que cualquier persona posee la capacidad innata de producir y entender el lenguaje.

Cuatro producciones quedan normalizadas con este cambio, luego, para modificar las producciones con tres Símbolos No terminales, agregamos las producciones:

$$D_1 \rightarrow AA$$

$$D_2 \rightarrow BB$$

Reemplazando estos nuevos Símbolos No Terminales en las dos producciones restantes, obtenemos finalmente la **GIC** equivalente en la **FNCh**:

$$S \rightarrow C_bA \mid C_aB$$

$$A \rightarrow C_bD_1 \mid C_aS \mid \mathbf{a}$$

$$B \rightarrow C_aD_2 \mid C_bS \mid \mathbf{b}$$

$$C_a \rightarrow \mathbf{a}$$

$$C_b \rightarrow \mathbf{b}$$

$$D_1 \rightarrow AA$$

$$D_2 \rightarrow BB$$

Ejemplo 2

Encontrar la **GIC** en **FNCh** equivalente a la siguiente **GIC** simplificada:

$$S \rightarrow \mathbf{a}A \mid \mathbf{a} \mid BA\mathbf{b}$$

$$A \rightarrow \mathbf{ab} \mid \mathbf{a}A\mathbf{b}$$

$$B \rightarrow \mathbf{b}$$

Las únicas dos producciones que previamente están normalizadas son:

$$S \rightarrow \mathbf{a}$$

$$B \rightarrow \mathbf{b}$$

Ahora debemos definir la nueva producción $C_a \rightarrow \mathbf{a}$, para sustituir al símbolo terminal **a**; no se requiere definir una nueva producción para el símbolo **b**, ya que la producción $B \rightarrow \mathbf{b}$ es única para ese No Terminal y se puede utilizar para ser reemplazada en la **GIC**.

Reemplazando los símbolos B y C_a en las cuatro producciones de la **GIC** original que no están normalizadas, obtenemos:

$$S \rightarrow C_aA \mid BAB$$

$$A \rightarrow C_aB \mid C_aAB$$

Dos de ellas aún no están normalizadas, para ello, definimos una producción más:

$$D_1 \rightarrow AB$$

Por lo tanto, la **GIC** equivalente en la **FNCh** queda finalmente así:

$$S \rightarrow C_aA \mid \mathbf{a} \mid BD_1$$

$$A \rightarrow C_aB \mid C_aD_1$$

$$B \rightarrow \mathbf{b}$$

$$C_a \rightarrow \mathbf{a}$$

$$D_1 \rightarrow AB$$

Algoritmo CYK (Cocke, Younger y Kasami)

Sea G una **GIC** en la forma Normal de Chomsky y sea *w* una cadena cualquiera de $\sum^*$, este algoritmo nos permite determinar si dicha cadena puede ser o no generada por esta gramática, y se basa en la idea de construir un árbol de derivación para la cadena *w*, pero en sentido inverso, es decir, partiendo desde las hojas (los símbolos terminales) y tratar de llegar a la raíz (el Símbolo Inicial S), si es posible lograrlo, significa que la cadena puede ser generada por esa gramática.

La ventaja de utilizar la gramática en la **FNCh** es que el árbol de derivación es un árbol binario, y por tanto, tiene el menor número posible de combinaciones.

Ejemplo 1

Determinar si la cadena *w* = **bab** puede ser generada por la siguiente **GIC**, que está en la **FNCh**:

$$S \rightarrow AB \mid BC$$

$$A \rightarrow BA \mid \mathbf{a}$$

$$B \rightarrow CC \mid \mathbf{b}$$

$$C \rightarrow AB \mid \mathbf{a}$$

Como se trata de una cadena de sólo tres símbolos, tenemos solamente dos estructuras de árbol de derivación binarios posibles, las cuales se muestran en la figura 7.5, pero desconocemos los símbolos que ocupa cada uno de los vértices del árbol que no son hoja, por lo que las indicamos con el signo de interrogación. El objetivo es ir reemplazándolos por los **SNT** correspondientes.

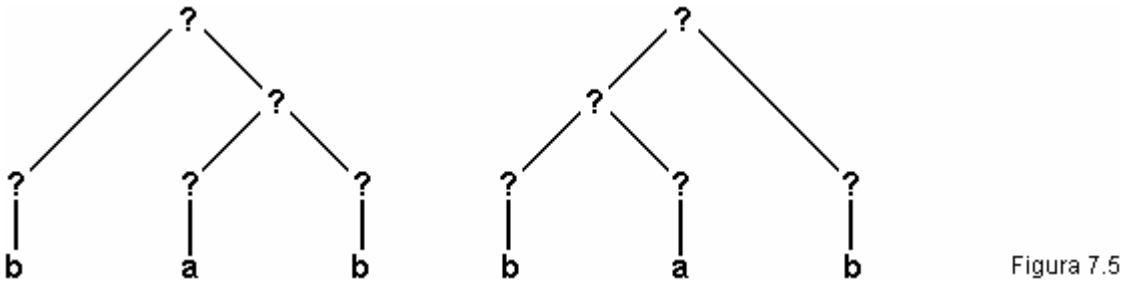


Figura 7.5

Las producciones con símbolos terminales nos permiten subir un nivel en la sustitución de las incógnitas de éstos árboles, algunas de las cuales pueden tener más de una alternativa, como se puede apreciar en la figura 7.6:

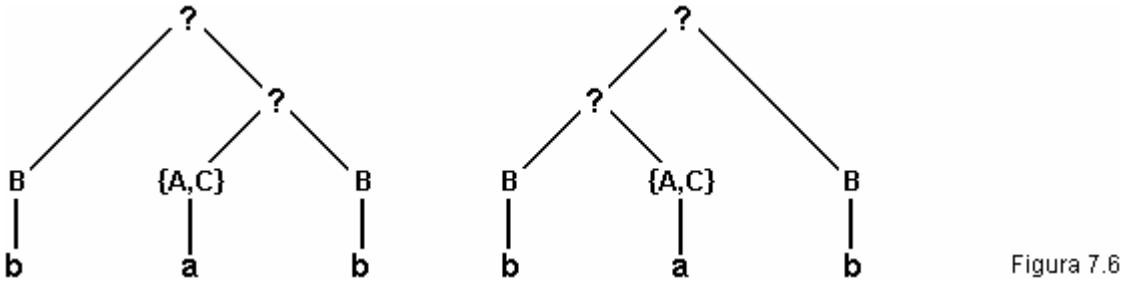


Figura 7.6

Los demás niveles se deberán ir reemplazando con las producciones que nos permitan generar los pares de No Terminales posibles, como se aprecia en la figura 7.7:

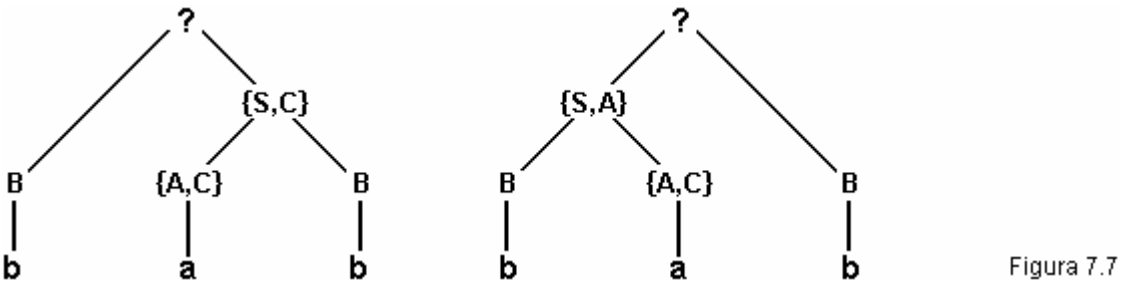


Figura 7.7

Finalmente, en la figura 7.8, llegamos a determinar las posibilidades de la raíz, que en ambos casos contienen al símbolo S, como es posible construir un árbol de derivación a partir del símbolo S, se concluye que esta gramática si genera a la cadena *w* = **bab**.

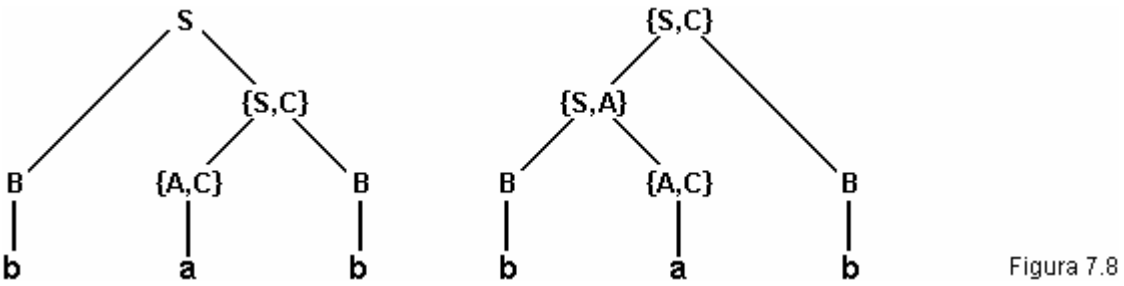


Figura 7.8

Además se puede observar que esta gramática es ambigua, ya que ambas derivaciones contienen al símbolo S en su raíz.

Este procedimiento se vuelve tremendamente engorroso si se pretende dibujar a todos los árboles posibles para cadenas más grandes, una cadena de longitud 4 tiene 5 posibles árboles de derivación, y una cadena de longitud 5 tendrá 14 combinaciones posibles, y así sucesivamente.

En vez de ello, se hace uso del algoritmo **CYK**, con el cual se puede construir la siguiente tabla, de abajo hacia arriba, y de forma piramidal, en la cual se resumen todas las posibles formas de los árboles de derivación:

Los primeros dos pasos, para j = 1 y 2, se asemejan a lo descrito anteriormente, pero resume en cada celda el análisis hecho para ambos árboles:

j = 2		{ S, A }	{ S, C }
j = 1	{ B }	{ A, C }	{ B }
	b	**a**	**b**

Tabla 7.1

Finalmente, para j = 3, la obtención de los elementos que constituyen la celda se obtiene de la unión de ambos árboles, que son las dos maneras de sumar 3:

Primero, para la combinación 1 + 2 (las celdas que se identifican con el símbolo ♦) y luego con 2 + 1, las celdas identificadas por el símbolo ♥. Recordando que las parejas de Símbolos No Terminales siempre se leen de izquierda a derecha, independientemente de que una celda esté en un nivel inferior o superior a la otra.

j = 3		{ S, C }	
j = 2	{ S, A } ♥	{ S, C } ♦	
j = 1	{ B } ♦	{ A, C }	{ B } ♥
	b	**a**	**b**

Tabla 7.2

Ejemplo 2

Ahora queremos determinar si la cadena *w* = **abab** puede ser generada por la misma gramática.

Aplicando el algoritmo, obtenemos la tabla 7.3, ahí podemos observar que la tabla anterior está contenida en ella, solamente aparecen las nuevas celdas al inicio de

cada fila. Para obtener el contenido de la cuarta línea, hay que analizar las tres posibles combinaciones: 1 + 3, 2 + 2 y 3 + 1; indicadas en las celdas por los símbolos ♠, ♥ y ♦, respectivamente. Finalmente, podemos observar que una celda se construye por medio de las celdas inferiores que forman las diagonales convergentes a dicha celda, independientemente de la posición de la misma en la pirámide.

j = 4		{ B }		
j = 3		{ B } ♦	{ S, C } ♠	
j = 2	{ S, C } ♥	{ S, A }	{ S, C } ♥	
j = 1	{ A, C } ♠	{ B }	{ A, C }	{ B } ♦
	a	**b**	**a**	**b**

Tabla 7.3

Dado que S no se encuentra en la parte superior de la tabla, concluimos que *w* no puede ser generado por dicha gramática.

Forma Normal de Greibach

Otra Forma Normal importante en las Gramáticas Independientes del Contexto es la Forma Normal de Greibach (**FNG**), la cual se define así:

Una **GIC** está en **FNG** si todas sus producciones son de la forma: $A \rightarrow \mathbf{a}\alpha$, en donde **a** es un Símbolo Terminal, $\mathbf{a} \in \sum$ y $\alpha \in (\sum \cup N)^*$, con la posible excepción de $S \rightarrow \varepsilon$. Es decir que se requiere que toda producción tenga un símbolo terminal al principio, y por tanto, se incluye de forma excepcional a la producción de la cadena vacía.

Ejemplo 1

Sea la siguiente **GIC** encuentre una equivalente en la **FNG**.

$S \rightarrow \mathbf{b}S \mid A\mathbf{c}S \mid \mathbf{a}$

$A \rightarrow \mathbf{b}A \mid B\mathbf{a}$

$B \rightarrow \mathbf{a}B \mid \mathbf{b}$

Podemos observar que las producciones de B ya están normalizadas, por lo que bastará reemplazar las producciones de B en la segunda producción de A, que inicia con ese símbolo, para normalizarlas:

$A \rightarrow \mathbf{b}A \mid \mathbf{a}B\mathbf{a} \mid \mathbf{ba}$

Similarmente, reemplazando las producciones de A, ya normalizadas, por la inicial de la segunda producción de S tenemos que también quedan normalizadas:

$$S \rightarrow \mathbf{b}S \mid \mathbf{b}A\mathbf{c}S \mid \mathbf{a}B\mathbf{ac}S \mid \mathbf{bac}S \mid \mathbf{a}$$

Reuniendo todas producciones anteriores, el resultado final en la **FNG** es:

$$S \rightarrow \mathbf{b}S \mid \mathbf{b}A\mathbf{c}S \mid \mathbf{a}B\mathbf{ac}S \mid \mathbf{bac}S \mid \mathbf{a}$$

$$A \rightarrow \mathbf{b}A \mid \mathbf{a}B\mathbf{a} \mid \mathbf{ba}$$

$$B \rightarrow \mathbf{a}B \mid \mathbf{b}$$

Teorema de la Recursividad

Si el **SNT** A tiene producciones de la forma $A \rightarrow A\alpha$ y otras producciones de la forma $A \rightarrow \beta$, podemos eliminar las producciones recursivas a la izquierda por medio de la introducción de un nuevo **SNT** A' y entonces reemplazamos las producciones anteriores por las siguientes:

$$A \rightarrow \beta \mid \beta A'$$

$$A' \rightarrow \alpha \mid \alpha A'$$

En ocasiones, encontrar una **GIC** equivalente en la **FNG**, requiere de la aplicación del teorema de la recursividad.

Ejemplo 2

Sea la siguiente **GIC** encuentre una equivalente en la **FNG**.

$$S \rightarrow S\mathbf{b}S \mid S\mathbf{c}S \mid \mathbf{a}$$

Ahora aplicando el teorema de la recursividad, identificamos que β **= a** , mientras que α_1 **=** $\mathbf{b}S$ y α_2 **=** $\mathbf{c}S$, por lo que nos queda:

$$S \rightarrow \mathbf{a}S' \mid \mathbf{a}$$

$$S' \rightarrow \mathbf{b}S \mid \mathbf{c}S \mid \mathbf{b}SS' \mid \mathbf{c}SS'$$

Segunda Forma Normal de Greibach

Una **GIC** está en la **SFNG** si todas sus producciones son de la forma: $A \rightarrow \mathbf{a}\alpha$, donde $\mathbf{a} \in \sum$ y $\alpha \in N^*$. Es decir que se requiere que toda producción tenga un símbolo terminal al principio, seguido de uno, varios o ningún **SNT**.

Una muestra de esta forma es la que se obtuvo en el ejemplo previo.

Ejemplo 3

Sea la siguiente **GIC** en **FNCh**, encuentre una gramática equivalente en la **FNG**.

$S \rightarrow AA \mid \mathbf{a}$

$A \rightarrow SA \mid \mathbf{b}$

Primero reemplazamos las producciones de S en la primera producción para A, (también se hubiera podido reemplazar la A en la producción correspondiente al símbolo S), resultando que:

$A \rightarrow AAA \mid \mathbf{a}A \mid \mathbf{b}$

Ahora aplicamos el teorema de la recursividad, de donde resulta:

$A \rightarrow \mathbf{a}A \mid \mathbf{b} \mid \mathbf{a}AA' \mid \mathbf{b}A'$

$A' \rightarrow AA \mid AAA'$

Ahora reemplazamos la A inicial en la primera producción, quedando normalizada:

$S \rightarrow \mathbf{a}AA \mid \mathbf{b}A \mid \mathbf{a}AA'A \mid \mathbf{b}A'A \mid \mathbf{a}$

Finalmente reemplazamos la A inicial en las producciones de A', quedando:

$A' \rightarrow \mathbf{a}AA \mid \mathbf{b}A \mid \mathbf{a}AA'A \mid \mathbf{b}A'A \mid \mathbf{a}AAA' \mid \mathbf{b}AA' \mid \mathbf{a}AA'AA' \mid \mathbf{b}A'AA'$

Reuniendo los resultados, la gramática equivalente en la segunda **FNG** es:

$S \rightarrow \mathbf{a}AA \mid \mathbf{b}A \mid \mathbf{a}AA'A \mid \mathbf{b}A'A \mid \mathbf{a}$

$A \rightarrow \mathbf{a}A \mid \mathbf{b} \mid \mathbf{a}AA' \mid \mathbf{b}A'$

$A' \rightarrow \mathbf{a}AA \mid \mathbf{b}A \mid \mathbf{a}AA'A \mid \mathbf{b}A'A \mid \mathbf{a}AAA' \mid \mathbf{b}AA' \mid \mathbf{a}AA'AA' \mid \mathbf{b}A'AA'$

Preguntas

a) ¿Es posible que una Gramática Regular sea Ambigua?

b) ¿Es posible encontrar una gramática en la **FNCh** que genere a un **LR**?

c) ¿Se puede considerar que toda Gramática Regular se encuentra en la **FNG**?

d) Sea G una Gramática Regular Inversa. ¿Puede encontrarse una gramática equivalente en la **FNG**?

e) Es posible escribir un algoritmo semejante al **CYK** para gramáticas que no estén en la **FNCh**?

f) ¿Garantiza el algoritmo **CYK** que siempre es posible determinar si una cadena puede ser generada o no por una gramática dada?

Ejercicios Capítulo 7

7.1 Depurar las siguientes gramáticas regulares y encontrar una gramática equivalente libre de anomalías, para cada uno de los siguientes casos:

a) S → **a** | **a**A | B
A → **a**B | **d**A
B → **e**A
C → **b**

b) S → **a**A | **b**A | **a**
A → **a**A | **bb**A | ε

c) S → **ab**A
A → **cc**C
B → **dd** | D
C → **a** | **e**A
D → **f**
E → **g**F

d) S → **a** | **a**A | B
A → B | D | ε
B → A | **b**

e) S → A
A → **a** | B
B → A | ε

7.2 Depurar cada una de las siguientes gramáticas independientes del contexto y encontrar una gramática equivalente libre de anomalías:

a) S → AB
A → **a**A | **ab**B | **a**C**a**
B → **b**A | BB | ε
C → ε
D → **d**B | BCB

b) S → **a**B
A → **bc**CCC | **d**A
B → **a**B | **e**
C → **f**A
D → D**gh**

c) S → A | AA | AAA
A → AB**a** | AC**a** | **a**
B → AB**a** | A**b** | ε
C → C**ab** | CC
D → CD | C**d** | CE**a**
E → **b**

d) S → D | **a**E | **b**CD
A → C**d** | CS**a** | **b**B
B → **a**B | **b**A
C → C**ab** | **c**B
D → **a**A | C**a** | **b**
E → BE**a** | DB**b** | ε

e) S → D | **a**ED | **b**CD
A → C**d** | CS**a** | **b**B
B → **a**B | **b**A
C → C**ab** | **c**B
D → **a**A | E**a** | **b**
E → E**a** | DB**b** | ε

f) S → DB | **a**E | **b**CD
A → C**d** | CS**a** | **b**A
B → **a**B | **b**S | ε
C → C**ab** | **c**AE
D → **a**A | C**a** | **b**
E → BE**a** | D**ab**

g) S → **a** | **a**A | B | C
A → **a**B | ε
B → A**a**
C → **b**CD
D → **ccc**

h) S → **a**A**b** | **c**EB | CE
A → **d**BE | **ee**C
B → **ff** | D
C → **g**FB | **ae**
D → **h**

i) S → C**d** | CS**b** | **b**EA
A → S | **a**E | **a**CD
B → **a**B | **b**SC
C → C**ab** | **a**B
D → **a**A | C**b** | **b**
E → BE**a** | DB**b** | ε

j) S → AC | **b**C | **a**AF
A → S**b** | D**b** | **a**
B → **b**B | E**b**
C → SC | B**a** | ε
D → **b**EB | **a**E
E → B**ba** | **a**E

7.3 Encontrar la gramática en Forma Normal de Chomsky equivalente a cada una de las gramáticas libres de anomalías siguientes:

a) S → AB | **ac** | ε
A → **a**B | **b**B**b**A
B → **b**

b) S → **a**A | **a** | A**b**
A → **a**B**b**
B → **b** | A**a**

c) S → **a**A | B**a** | **b**
A → **a**C | **b**BS
B → B**ab** | **a**
C → AC**a** | S**b**

d) S → **a** | **b**AB
A → **a**S | **b**B | **c**CA
B → **a**S | **b**
C → **b**B | **c**B | C**a**

e) S → A**b**S | **a**B | ε
A → A**b** | C**a** | **b**
B → **a**A | **b**B | **a**
C → C**ab** | **a**B

f) S → **b**A | **a**B | ε
A → **a**B | **b**CS | **b**
B → **a**A | **b**AS | **a**
C → S**a**C | B**a**

7.4 Depurar cada una de las siguientes gramáticas y encontrar una gramática equivalente en la Forma Normal de Chomsky:

a) S → **a**A**b** | **c**EB | CG
A → **d**BH | **eb**C
B → **f** | D
C → **g**EB | **ah**
E → **dc**GGG | **c**E
G → G**am**

b) S → **a**B | A**a**
A → **b**B
B → A | **b**

c) S → AAA | **a** | **a**A
A → B**b** | **a**BS | ε
B → **ba** | **ab**

7.5 Por medio el algoritmo de **CYK**, determina si la cadena *w* = **baabb** puede ser generada por la gramática siguiente:

$$S \rightarrow \mathbf{a} \mid AS$$
$$A \rightarrow AS \mid \mathbf{b}$$
$$B \rightarrow BC \mid \mathbf{a}$$
$$C \rightarrow CB \mid \mathbf{b}$$

7.6 Por medio el algoritmo de **CYK**, determina si la cadena *w* = **babab** puede ser generada por la gramática siguiente:

$$S \rightarrow \mathbf{a} \mid AS$$
$$A \rightarrow AB \mid \mathbf{b}$$
$$B \rightarrow SB \mid \mathbf{b}$$

7.7 Por medio el algoritmo de **CYK**, determina si la cadena *w* = **01101** puede ser generada por la gramática siguiente:

$$S \rightarrow BA \mid \mathbf{0} \mid \mathbf{1}$$
$$A \rightarrow AS \mid \mathbf{1}$$
$$B \rightarrow BS \mid \mathbf{0}$$

7.8 Por medio el algoritmo de **CYK**, determina si la cadena *w* = **11111** puede ser generada por la gramática del problema anterior.

7.9 Por medio el algoritmo de **CYK**, determina si la cadena *w* = **ababb** puede ser generada por la gramática siguiente:

$$S \rightarrow \mathbf{a} \mid SA$$
$$A \rightarrow AB \mid \mathbf{a}$$
$$B \rightarrow BA \mid \mathbf{b}$$

7.10 Consideremos la siguiente **GIC** en la **FNCh**:

$$S \rightarrow AB \mid BC$$
$$A \rightarrow BA \mid \mathbf{a}$$
$$B \rightarrow AC \mid \mathbf{b}$$
$$C \rightarrow CB \mid \mathbf{a}$$

Utilice el algoritmo **CYK** para verificar si las cadenas siguientes pueden ser generadas por esta gramática:

a) w_1 = **aaa**

b) w_2 = **aba**

c) w_3 = **ababa**

d) w_4 = **baaab**

7.11 Consideremos la siguiente **GIC** en la **FNCh**:

$S \rightarrow AB \mid AC \mid \mathbf{a}$
$A \rightarrow BD$
$B \rightarrow CS \mid \mathbf{b}$
$C \rightarrow DC \mid \mathbf{b}$
$D \rightarrow AB \mid \mathbf{a}$

Utilice el algoritmo **CYK** para verificar si las cadenas siguientes pueden ser generadas por esta gramática:

a) w_1 = **bbab**

b) w_2 = **baba**

c) w_3 = **abbba**

d) w_4 = **abaab**

7.12 Encuentre una gramática equivalente en la **FNG** a cada una de las gramáticas siguientes:

a) $S \rightarrow \mathbf{a}S\mathbf{c} \mid S\mathbf{c} \mid \mathbf{b}$

b) $S \rightarrow \mathbf{b}S \mid S\mathbf{a} \mid \mathbf{a} \mid S\mathbf{ba}$

c) $S \rightarrow S\mathbf{a} \mid S\mathbf{b} \mid \mathbf{c}A$
$A \rightarrow A\mathbf{a} \mid \mathbf{a}$

d) $S \rightarrow S\mathbf{a} \mid S\mathbf{b} \mid A\mathbf{ab}$
$A \rightarrow A\mathbf{ba} \mid S\mathbf{a} \mid \mathbf{a}$

e) $S \rightarrow AA \mid \mathbf{0}$
$A \rightarrow SS \mid \mathbf{1}$

f) $S \rightarrow \mathbf{a}A\mathbf{b} \mid \mathbf{c}SB \mid SS$
$A \rightarrow \mathbf{b}BA \mid \mathbf{ba}A$
$B \rightarrow \mathbf{ac} \mid SA$

g) $S \rightarrow ABA \mid \mathbf{b} \mid \mathbf{a}S$
$A \rightarrow B\mathbf{b} \mid \mathbf{a}BS \mid AS$
$B \rightarrow \mathbf{ba} \mid \mathbf{ab}$

Capítulo

8

Lenguajes Libres del Contexto

Se definen los Lenguajes Libres del Contexto y se analizan sus principales propiedades, se define el concepto de Analizador Sintáctico, como una clase especial de Gramáticas Independientes del Contexto.

Lenguajes No Regulares

Llamamos Lenguaje Libre del Contexto o Lenguaje Independiente del Contexto (**LIC**) al lenguaje que puede ser generado por alguna **GIC**. Dado que toda Gramática Regular también es una **GIC**, podemos afirmar que todo Lenguaje Regular es también un **LIC**, sin embargo, no cualquier **LIC** es Regular, por lo que surge la interrogante: ¿Cómo podemos diferenciar un Lenguaje Regular de uno que no lo es?

Por ejemplo, el lenguaje L = { $\mathbf{a}^n\mathbf{b}^n \mid n \geq 0$} visto en el capítulo anterior no es regular, una manera de confirmar esta aseveración es por el hecho de que no es posible construir un **AF** que acepte dicho lenguaje ni tampoco se puede construir una Gramática Regular que lo genere, pero siempre queda la duda ¿No es posible o simplemente no fuimos capaces de hacerlo? Afortunadamente existe una manera directa y certera de probarlo, por medio de la aplicación del *Lema del Rizo*, también conocido como *Lema del Bombeo*.

Lema del Rizo

Sea L un Lenguaje Regular infinito, entonces existe una constante k, de tal forma que cualquier cadena $w \in$ L cuya longitud sea mayor o igual a k, se puede escribir de la forma $w = uvx$, donde $|v| \geq 1$ y $|uv| \leq$ k, se debe cumplir también que todas las cadenas de la forma $uv^m x$ pertenecen a L para todo $m \geq 0$.

Si pudiéramos construir un **AFD** con k estados que acepte a L, y si la cadena es de longitud mayor o igual a k, es forzoso que las transiciones pasen por un mismo estado dos veces antes de llegar al estado de aceptación. El bucle o rizo que se forma para regresar al mismo estado se puede repetir tantas veces como se quiera. Gráficamente esto se puede representar por medio del diagrama mostrado en la siguiente figura:

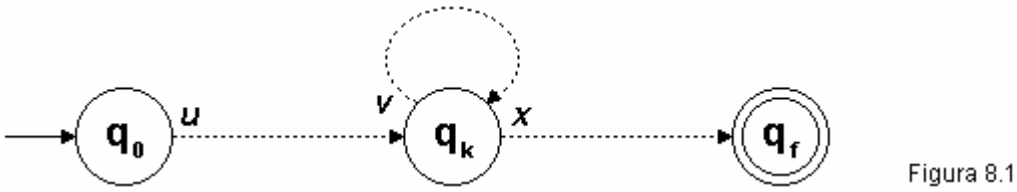


Figura 8.1

Utilizando el lema anterior, se puede demostrar que L = $\mathbf{a}^n\mathbf{b}^n$ no es regular, porque no importa lo grande que se tome a la constante k, siempre existirá una cadena que se rehúse a ser bombeada, porque el citado rizo no existe y por lo tanto, el **AFD** no puede ser construido. Por ejemplo, dada la cadena *w* = **aaabbb**, no se puede encontrar ninguna subcadena *v* en *w*, tal que se satisfaga que cualquier cadena de la forma uv^mx también pertenezca a L.

Si suponemos que *v* = **a**, entonces se tiene que permitir que *w* contenga cualquier cantidad de **a**s, lo que contradice la definición del lenguaje, porque debe ser igual a la de **b**s, un razonamiento similar se sigue si asumimos que *v* = **b**, finalmente, si suponemos que *v* = **ab**, entonces el lenguaje debería aceptar cadenas que contengan subcadenas de la forma **ababab**…, lo cual tampoco es válido.

Este razonamiento se puede aplicar a cadenas de cualquier longitud, por lo tanto, se concluye que el lenguaje L = $\mathbf{a}^n\mathbf{b}^n$ no es regular.

Propiedades de los LICs

La primera propiedad que señalaremos es la de que todo Lenguaje Independiente del Contexto sobre un alfabeto de un solo símbolo es Regular, esto no significa, como veremos en capítulos posteriores, que todo lenguaje sobre un alfabeto de un solo símbolo, tenga que ser Regular.

Ejemplo

La gramática siguiente es una **GIC**:

$$S \rightarrow \mathbf{a}S\mathbf{a} \mid \mathbf{a}S \mid SS \mid \varepsilon$$

Pero genera al lenguaje L = { $\mathbf{a}^n$ | $n \geq 0$ } = **a***, el cual es regular.

Unión

Sean L_1 y L_2 dos Lenguajes Independientes del Contexto, entonces la unión de ellos $L_1 \cup L_2$ también es un **LIC**.

Ejemplo

Obtener la **GIC** que genera el lenguaje $L = \{ \mathbf{a}^n\mathbf{b}^m \mid m = n \text{ o } m = 2n \}$.

Primeramente, podemos observar que L es la unión de los dos lenguajes siguientes: $L_1 = \{ \mathbf{a}^n\mathbf{b}^n \mid n \geq 0 \}$ y $L_2 = \{ \mathbf{a}^n\mathbf{b}^{2n} \mid n \geq 0 \}$, que son generados por las siguientes gramáticas: $S_1 \rightarrow \mathbf{a}S_1\mathbf{b} \mid \varepsilon$ y $S_2 \rightarrow \mathbf{a}S_2\mathbf{bb} \mid \varepsilon$, respectivamente, por lo tanto, la gramática que genera a $L = L_1 \cup L_2$ simplemente requiere agregar una producción inicial que permita elegir entre ambas gramáticas: $S \rightarrow S_1 \mid S_2$**.**

Concatenación

Sean L_1 y L_2 dos Lenguajes Independientes del Contexto, entonces la concatenación $L_1 \cdot L_2$ también es un **LIC**.

Ejemplo

Obtener la **GIC** que genera el lenguaje $L = \{ \mathbf{a}^m\mathbf{b}^n \mid m \geq n \geq 0 \}$.

Si definimos al índice $k = m - n$, podemos reemplazar a m por $k + n$ y representar a L como la concatenación de dos lenguajes: $L_1 = \{ \mathbf{a}^k \mid k \geq 0 \}$ y $L_2 = \{ \mathbf{a}^n\mathbf{b}^n \mid n \geq 0\}$, cuyas gramáticas son $S_1 \rightarrow \mathbf{a}S_1 \mid \varepsilon$ y $S_2 \rightarrow \mathbf{a}S_2\mathbf{b} \mid \varepsilon$, respectivamente; por lo tanto, la gramática que genera a $L = L_1 \cdot L_2$ simplemente requiere una producción que permita concatenar las cadenas generadas por ambas gramáticas: $S \rightarrow S_1S_2$**.**

Cerradura Estrella

Sea L un Lenguaje Independiente del Contexto, entonces L*, la cerradura de Kleene de L, también es un **LIC**.

Ejemplo

Obtener la **GIC** que genera el lenguaje $L = \{ (\mathbf{a}^{m_k}\ \mathbf{b}^{n_k})^k \mid m_k \geq n_k \geq 0, k \geq 0 \}$.

En este caso se considera que $L = L_1^*$, donde $L_1 = \{ \mathbf{a}^m\mathbf{b}^n \mid m \geq n \geq 0 \}$ es el lenguaje visto en el ejemplo anterior, entonces si S_1 es el símbolo inicial de la **GIC** que genera las cadenas de L_1, basta con agregar un par de producciones para el nuevo símbolo inicial, de la forma siguiente: $S \rightarrow S_1S \mid \varepsilon$.

Existen muchos lenguajes semejantes a los vistos en estos ejemplos cuyas gramáticas se pueden obtener por medio de la aplicación de las propiedades anteriores.

Homomorfismo

Sea L un Lenguaje Independiente del Contexto y h es un homomorfismo dado, entonces h(L) también es un **LIC**. Esta propiedad es también válida para la operación inversa $h^{-1}(L)$.

Ejemplo

Obtener la **GIC** que genera el lenguaje L = { $\mathbf{a}^m\mathbf{bc}^m$ | m ≥ 0 }.

En este caso se considera el homomorfismo donde: h(**a**) = **0**, h(**b**) = **01** y h(**c**) = **1**, entonces: h(L) = { $\mathbf{0}^{m+1}\mathbf{1}^{m+1}$ | m ≥ 0 } = { $\mathbf{0}^m\mathbf{1}^m$ | m ≥ 1 }, que un lenguaje del que conocemos su gramática, la cual es S → **0**S**1** | **01**, entonces, aplicando la transformación inversa h^{-1} se obtiene la siguiente gramática para L: S' → **a**S'**c** | **b**.

Lenguajes inherentemente ambiguos

Existen dos tipos de ambigüedad: la ambigüedad que proviene de la forma de la gramática que se está utilizando, y por tanto, es removible y la ambigüedad que es de fondo y que es una característica del lenguaje, no es posible encontrar una gramática que no sea ambigua, porque el lenguaje es *inherentemente ambiguo*.

Ejemplo

El lenguaje L = { $\mathbf{a}^n\mathbf{b}^m\mathbf{c}^m\mathbf{d}^n$ | n ≥ 1, m ≥ 1} ∪ { $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^m\mathbf{d}^m$ | n ≥ 1, m ≥ 1} es un lenguaje Inherentemente ambiguo, puesto que la cadena w = $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n\mathbf{d}^n$, para cualquier valor de n ≥ 1, puede provenir de cualquiera de los dos sublenguajes que lo forman, y por lo tanto, independientemente de la gramática empleada, siempre habrá dos maneras distintas de derivarla.

Intersección

Sean L_1 y L_2 dos Lenguajes Independientes del Contexto, entonces la intersección de ambos, $L_1 \cap L_2$, no necesariamente es un **LIC**. Pero la intersección de un Lenguaje Independiente del Contexto con un Lenguaje Regular siempre es un **LIC**.

Ejemplo 1

Sean los lenguajes L_1 = $\mathbf{a}^*\mathbf{b}^*$ = { $\mathbf{a}^n\mathbf{b}^m$ | n ≥ 0, m ≥ 0} y L_2 = { $\mathbf{a}^n\mathbf{b}^n$ | n ≥ 1 }, entonces la intersección $L_1 \cap L_2$ = { $\mathbf{a}^n\mathbf{b}^n$ | n ≥ 1} es un **LIC**.

Ejemplo 2

Sean los lenguajes $L_1 = \{ \mathbf{a}^n\mathbf{b}^m\mathbf{c}^m \mid n \geq 1, m \geq 1\}$ y $L_2 = \{ \mathbf{a}^n\mathbf{b}^n\mathbf{c}^m \mid n \geq 1, m \geq 1\}$, en este caso, la intersección $L_1 \cap L_2 = \{ \mathbf{a}^n\mathbf{b}^n\mathbf{c}^n \mid n \geq 1\}$ no es un **LIC**, como se demostrará más adelante.

Construcción de las GICs

Dado un Lenguaje Independiente del Contexto, es posible construir una **GIC** que genere dicho Lenguaje, basándose en la estructura de algunas gramáticas conocidas y, en su caso, aplicando las propiedades anteriores.

De esta manera es posible construir una **GIC** que genere cualquier lenguaje de la forma, $L = \{ \mathbf{a}^n\mathbf{b}^m \}$, para distintas relaciones entre m y n, encontrando un homomorfismo con lenguaje $L = \{ \mathbf{a}^n\mathbf{b}^n \}$, y utilizando las transformaciones correspondientes sobre la gramática que genera a este lenguaje, para encontrar la **GIC** que genera al lenguaje en cuestión. De esta forma podemos construir gramáticas para todos los lenguajes que se le asemejen.

Ejemplo

Obtener la **GIC** que genera el lenguaje $L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^+ \mid N_{\mathbf{a}}(w) = 2\, N_{\mathbf{b}}(w) \}$, a partir del hecho que se conoce una gramática no ambigua para generar el lenguaje de las cadenas que contienen la misma cantidad de **a**s que **b**s.

En el capítulo anterior se mencionó que la siguiente gramática permite generar al lenguaje de las cadenas no vacías que contiene la misma cantidad de **a**s que **b**s:

$$S \rightarrow \mathbf{b}A \mid \mathbf{a}B$$
$$A \rightarrow \mathbf{b}AA \mid \mathbf{a}S \mid \mathbf{a}$$
$$B \rightarrow \mathbf{a}BB \mid \mathbf{b}S \mid \mathbf{b}$$

Entonces lo único que tenemos que hacer es buscar la manera para duplicar la cantidad de **a**s, en cualquier cadena generada, lo que nos conduce a la siguiente **GIC**, que genera de manera no ambigua al lenguaje solicitado:

$$S \rightarrow \mathbf{b}AA \mid \mathbf{a}B$$
$$A \rightarrow \mathbf{b}AAA \mid \mathbf{a}S \mid \mathbf{a}$$
$$B \rightarrow \mathbf{a}BB \mid \mathbf{ba}S \mid \mathbf{ab}S \mid \mathbf{ba} \mid \mathbf{ab}$$

Otro tipo de **LIC**s que son muy interesantes corresponde a los que contienen cadenas palíndromas, obtener una gramática para este tipo de lenguajes es muy sencillo, ya que basta con construir la cadena desde los extremos hacia el centro de la misma, finalizando con el símbolo central si la longitud de la cadena es impar o con la cadena vacía para las cadenas de longitud par.

Ejemplo

Construir una gramática que genere al lenguaje L = { ww^R | $w \in$ {**a**, **b**}* }:

Como en este caso todas las cadenas generadas deben ser siempre de longitud par, la gramática sería:

$$S \rightarrow \mathbf{b}S\mathbf{b} \mid \mathbf{a}S\mathbf{a} \mid \varepsilon$$

Teorema del Rizo para los LICs

Sea G una Gramática Independiente del Contexto, entonces existe una constante k, de tal forma que cualquier cadena $w \in$ L(G) cuya longitud sea mayor o igual a k, se puede escribir de la forma $w = uvxyz$, de la forma que v o y son no vacías, se debe cumplir que también que todas las cadenas uv^nxy^nz pertenecen a L(G) para todo $n \geq 0$.

Para demostrar este teorema, basta probar que en la gramática G es posible encontrar una derivación de la forma $A \Rightarrow^* vAy$ que forme parte de la derivación de la cadena w: $S \Rightarrow^* uvxyz$, la cual es posible aplicarla de manera repetitiva las veces que se quiera, para generar todas las cadenas de la forma: uv^nxy^nz. Esta propiedad se conoce como la *Periodicidad* de las **GIC**s.

En otras palabras, para que un lenguaje sea un **LIC** se permita que exista una condicionante entre dos de los símbolos (o subcadenas), como en el caso de $\mathbf{a}^n\mathbf{b}^n$, o bien, que haya varias condicionantes entre varios símbolos, siempre por pares y definidas por índices independientes entre sí, como en el caso de $\mathbf{a}^n\mathbf{b}^m\mathbf{c}^m\mathbf{d}^n$, pero cuando existen condicionantes solapadas como en $\mathbf{a}^n\mathbf{b}^m\mathbf{c}^n\mathbf{d}^m$ o que vinculan a tres o más símbolos como es el caso de $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$, no puede ser un **LIC** y por tanto no existe una **GIC** que pueda generar dicho lenguaje.

Análisis Sintáctico

El análisis sintáctico es el proceso de determinar si una cadena dada puede ser generada por una gramática, para el estudio de este problema, es conveniente construir un árbol de derivación sintáctico, que nos asegure que la determinación sea correcta.

Los analizadores sintácticos de lenguajes de programación suelen hacer un examen simple de izquierda a derecha, viendo un componente léxico de la entrada, a la vez.

La mayoría de los métodos de análisis sintáctico están comprendidos en dos clases, dependiendo del orden en que se construyen los nodos del árbol de derivación pueden ser descendentes o ascendentes.

En el primer caso, se construyen comenzando de la raíz, paso a paso, avanzando hacia las hojas y en cada paso se representa una derivación de la palabra por la izquierda, en el segundo, la construcción se inicia en las hojas y avanza hacia la raíz siguiendo el camino contrario en una derivación por la derecha de la palabra.

Ejemplo

Sea la gramática siguiente:

$$S \rightarrow T \mid \mathbf{ab} \mid \mathbf{c}T\mathbf{c}S$$
$$T \rightarrow \mathbf{d} \mid \mathbf{e} \mid \mathbf{f}$$

La derivación Descendente para la cadena *w* = **cdccecf**, es la siguiente:

$$S \Rightarrow \mathbf{c}T\mathbf{c}S \Rightarrow \mathbf{cdc}S \Rightarrow \mathbf{cdcc}T\mathbf{c}S \Rightarrow \mathbf{cdccec}S \Rightarrow \mathbf{cdccec}T \Rightarrow \mathbf{cdccecf}$$

Y se representa por medio de un árbol en el que se enfatiza la derivación por la izquierda, como se aprecia en la siguiente figura:

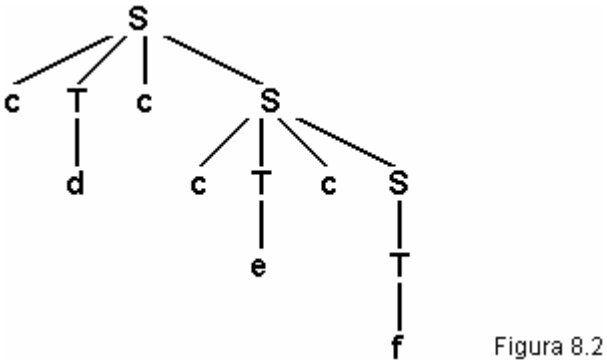


Figura 8.2

Ahora, la derivación Ascendente para la misma cadena es:

$$S \Rightarrow \mathbf{c}T\mathbf{c}S \Rightarrow \mathbf{c}T\mathbf{cc}T\mathbf{c}S \Rightarrow \mathbf{c}T\mathbf{cc}T\mathbf{c}T \Rightarrow \mathbf{c}T\mathbf{cc}T\mathbf{cf} \Rightarrow \mathbf{c}T\mathbf{ccecf} \Rightarrow \mathbf{cdccecf}$$

Y que se representa por medio del mismo árbol, pero enfatizando la derivación por la derecha, como se muestra en la figura siguiente:

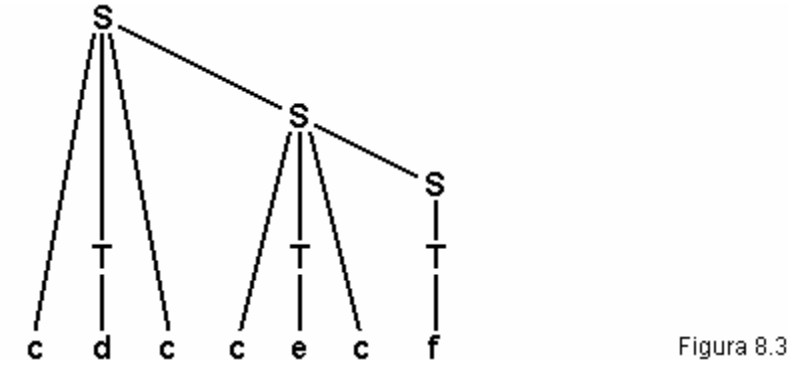


Figura 8.3

Los métodos ascendente y descendente más eficientes trabajan con dos subclases de gramáticas, que se les conoce como las gramáticas **LL** (left to left: leen en la cadena de izquierda a derecha, derivación por la izquierda) y **LR** (left to right: también leen de izquierda a derecha, pero la derivación es por la derecha), ambas son lo suficientemente expresivas como para describir a la mayoría de las construcciones sintácticas de los lenguajes de programación.

Derivaciones por la Derecha y por la Izquierda

Se dice que una cadena *w* perteneciente al lenguaje generado por alguna **GIC**, se ha analizado sintácticamente cuando se conocen uno (o todos) sus árboles de derivación.

Sea G = (N, Σ, P, S) una **GIC**, con las producciones en P numeradas 1, 2, ..., p y sea la cadena $w \in (N \cup \Sigma)^*$, Entonces:

1. Un análisis a la izquierda de *w* es una secuencia de producciones usadas en una derivación más a la izquierda de *w* desde S.
2. Un análisis a la derecha de *w* es el reverso de la secuencia de producciones usadas en una derivación más a la derecha de *w* desde S.

Ejemplo

Sea la cadena w = **aabbc**, y sea la **GIC** siguiente: G = ({ S,A,B,C }, { **a**,**b**,**c** }, P, S) conteniendo las producciones numeradas a continuación:

1. S → **a**A
2. A → **a**B**b**C
3. B → **b**
4. C → **c**

Entonces la derivación por la izquierda es:

$$S \Rightarrow \mathbf{a}A \Rightarrow \mathbf{aa}B\mathbf{b}C \Rightarrow \mathbf{aabb}C \Rightarrow \mathbf{aabbc}$$

Y por tanto, el análisis a la izquierda de **aabbc** está dado por la secuencia: 1234.

Mientras que la derivación por la derecha es:

$$S \Rightarrow \mathbf{a}A \Rightarrow \mathbf{aa}B\mathbf{b}C \Rightarrow \mathbf{aa}B\mathbf{bc} \Rightarrow \mathbf{aabbc}$$

Y por tanto, el análisis a la derecha de **aabbc** es 3421.

Las gramáticas que definen lenguajes de programación frecuentemente requieren de restricciones adicionales, sobre la forma de las reglas de producción, para eficientar el análisis de las cadenas del lenguaje y garantizar que este análisis siempre terminará.

Este tipo de gramáticas son las nombradas Gramáticas LL(k).

Gramáticas LL(k)

Las Gramáticas LL(k) constituyen un subconjunto de las **GIC**s usadas para la construcción de compiladores “top-down”. Éstas permiten un análisis determinista de arriba hacía abajo usando un análisis anticipado de k símbolos.

La notación LL describe la estrategia de análisis de una cadena para determinar si ésta pertenece a un lenguaje o no. La cadena de entrada es revisada de izquierda a derecha y el programa que realiza el análisis genera derivaciones por la izquierda.

Características particulares de una gramática LL(k)

Un analizador “top-down” intenta construir la derivación por la izquierda para una cadena de entrada *w*.

Para decidir, en un momento dado, sobre las reglas que deben considerarse para el No terminal A, se examina por adelantado a la cadena, de modo que se eliminen las reglas de A que no contengan a los símbolos que se están anticipando, hasta determinar de manera inequívoca la regla a aplicar.

Ejemplo 1

Sea la gramática G definida por las siguientes producciones:

$$S \rightarrow \mathbf{a}S \mid \mathbf{c}A$$
$$A \rightarrow \mathbf{b}A \mid \mathbf{c}B \mid \varepsilon$$
$$B \rightarrow \mathbf{c}B \mid \mathbf{a} \mid \varepsilon$$

El analizador intenta construir la derivación por la izquierda para la cadena **acbb**, a partir del símbolo inicial; aquí se presentan dos opciones: $S \Rightarrow \mathbf{a}S$ o $S \Rightarrow \mathbf{c}A$: Al anticipar que el símbolo a generar es una **a**, se elimina la segunda regla ya que no nos conduciría a la generación de **acbb**, quedando solamente: $S \Rightarrow \mathbf{a}S$.

En el siguiente paso, el símbolo anticipado es la **c**, ahora, la única regla de S que tiene inicialmente a la **c**, es la segunda, por lo que esa regla es la que se aplica.

El proceso continúa hasta lograr la derivación por la izquierda de la cadena, en todos los casos nos basta con anticipar un solo símbolo para determinar la producción a emplear, tal como se resume en la siguiente tabla:

Prefijo	Anticipación	Regla a Aplicar	Derivación
ε	**a**	$S \rightarrow \mathbf{a}S$	$S \Rightarrow \mathbf{a}S$
a	**c**	$S \rightarrow \mathbf{c}A$	$\Rightarrow \mathbf{ac}A$
ac	**b**	$A \rightarrow \mathbf{b}A$	$\Rightarrow \mathbf{acb}A$
acb	**b**	$A \rightarrow \mathbf{b}A$	$\Rightarrow \mathbf{acbb}A$
acbb	ε	$A \rightarrow \varepsilon$	$\Rightarrow \mathbf{acbb}$

Tabla 8.1

El significado de “k” en una gramática LL(k)

La cadena de símbolos de anticipación para una producción dada, puede ser de una longitud arbitraria, sin embargo, con la selección de las reglas adecuadas, sólo se requiere anticipar prefijos de longitud máxima k.

El valor k denota que la longitud del prefijo debe ser menor o igual a k. El ejemplo anterior representa a una gramática del tipo LL(1).

Ejemplo 2

El lenguaje L = { $\mathbf{a}^n\mathbf{abc}^n$ | n > 0 }, es generado por la gramática siguiente:

$$S \rightarrow \mathbf{a}S\mathbf{c} \mid \mathbf{aabc}$$

Para seleccionar que regla a utilizar, se requiere una anticipación de por lo menos tres símbolos, ya que las posibilidades para los prefijos son: { **aaa** o **aab** }, entonces, esta gramática es del tipo LL(3).

Para la gramática siguiente, que genera al mismo lenguaje, sólo se requiere una anticipación de dos símbolos:

$$S \rightarrow \mathbf{a}A$$

$$A \rightarrow S\mathbf{c} \mid \mathbf{abc}$$

Ya que para el símbolo no terminal S no se requiere ninguna, y para el símbolo A, se tienen las siguientes dos posibilidades: { **aa** o **ab** }, por lo que esta gramática es del tipo LL(2). Finalmente, la gramática siguiente es del tipo LL(1), ya que sólo se requiere de un símbolo de anticipación.

$$S \rightarrow \mathbf{aa}A\mathbf{c}$$

$$A \rightarrow \mathbf{a}A\mathbf{c} \mid \mathbf{b}$$

El análisis sintáctico nos enseña que no solamente es necesario elegir una gramática que no tenga ambigüedades, sino que también es muy importante el encontrar una gramática que sea del tipo con el valor de k más pequeño posible.

Gramáticas LR(k)

Los analizadores sintácticos LL(k), por su naturaleza predictiva, pueden analizar una clase restringida de lenguajes. Los analizadores sintácticos LR(k) evitan muchos de los problemas de los anteriores, cubriendo una clase mucho más amplia de lenguajes: los Lenguajes Independientes del Contexto Deterministas.

Las gramáticas LR(k) representan un subconjunto de las **GIC**s usadas para la construcción de compiladores “bottom-up”. Éstas permiten un análisis determinista de abajo hacía arriba. La notación LR describe la estrategia de análisis de una cadena para determinar si ésta pertenece a un lenguaje o no. La cadena de entrada es revisada de izquierda a derecha y el programa que realiza el análisis genera derivaciones por la derecha, usando un pre-análisis de k símbolos.

El analizador trata de reducir la cadena de entrada *w* al símbolo inicial S. En un proceso que recorre el árbol de derivación en sentido inverso, y que se conoce como *Reducción*. En términos generales, el analizador acumula los símbolos que va leyendo hasta que éstos sean iguales al lado derecho de alguna producción de la gramática. Al llegar a este punto, el analizador reemplaza (reduce) todos esos símbolos por el No terminal del lado derecho de la producción. Este proceso se repite hasta el punto en que se reduzca toda la cadena al símbolo inicial, indicando con esto que la cadena dada puede ser generada por una derivación por la derecha.

Se dice que una gramática es LR(k) si siempre es posible determinar en forma única el lado derecho de la producción a reducir, teniendo en cuenta el contexto izquierdo (prefijo incluido del lado derecho) y los siguientes k símbolos, de forma no ambigua.

Contextos LR(0)

Se llaman contextos LR(0) (**CLR**) de una producción dada, al conjunto de derivaciones parciales por la derecha que terminan en la aplicación de esa regla. Esto es la cadena *uw* está en el **CLR** de la producción A $\rightarrow$ *w*, si hay una reducción de la cadena *uwv* a S que empieza por reemplazar a w con A. Los contextos son empleados por los analizadores LR(k) para determinar la producción a utilizar en cada caso, para la reducción de una cadena dada.

Ejemplo 1

Sea la gramática G definida por las siguientes producciones:

$$S \rightarrow \mathbf{a}A \mid \mathbf{b}B$$
$$A \rightarrow \mathbf{ab}A \mid \mathbf{b}B$$
$$B \rightarrow \mathbf{b}B\mathbf{c} \mid \mathbf{bc}$$

Los **CLR** para cada una de las 6 producciones de G son los siguientes:

$S \rightarrow \mathbf{a}A$	$\{ \mathbf{a}A \}$
$S \rightarrow \mathbf{b}B$	$\{ \mathbf{b}B \}$
$A \rightarrow \mathbf{ab}A$	$\{ \mathbf{a}(\mathbf{ab})^n A \mid n > 0 \}$
$A \rightarrow \mathbf{b}B$	$\{ \mathbf{a}(\mathbf{ab})^n \mathbf{b}B \mid n \geq 0 \}$
$B \rightarrow \mathbf{b}B\mathbf{c}$	$\{ \mathbf{a}(\mathbf{ab})^n \mathbf{bb}^m B\mathbf{c}$ o $\mathbf{bb}^m B\mathbf{c} \mid n \geq 0, m > 0 \}$
$B \rightarrow \mathbf{bc}$	$\{ \mathbf{a}(\mathbf{ab})^n \mathbf{bb}^m \mathbf{c}$ o $\mathbf{bb}^m \mathbf{c} \mid n \geq 0, m > 0 \}$

Aunque no siempre es sencilla su determinación de los **CLR**, si es más fácil verificarlo.

Ejemplo 2

Sea la gramática G definida por las siguientes producciones:

$$S \rightarrow A$$
$$A \rightarrow T \mid A\mathbf{+}T$$
$$T \rightarrow \mathbf{b} \mid \mathbf{(}A\mathbf{)}$$

Como ejemplo de un proceso de reducción LR(k), se muestra, paso a paso, la reducción de la cadena **(b)+b** hasta concluir con el símbolo inicial S, tal como se presenta a continuación:

Reducción	**Regla a Aplicar**
(b)+b	
(T**)+b**	$T \rightarrow \mathbf{b}$
(A**)+b**	$A \rightarrow T$
T**+b**	$T \rightarrow \mathbf{(}A\mathbf{)}$
A**+b**	$A \rightarrow T$
A**+**T	$T \rightarrow \mathbf{b}$
A	$A \rightarrow A\mathbf{+}T$
S	$S \rightarrow A$

Tabla 8.2

Si invertimos el orden en que se generó la tabla obtenemos la derivación por la derecha de la cadena dada:

$$S \Rightarrow A \Rightarrow A\mathbf{+}T \Rightarrow A\mathbf{+b} \Rightarrow T\mathbf{+b} \Rightarrow \mathbf{(}A\mathbf{)+b} \Rightarrow \mathbf{(}T\mathbf{)+b} \Rightarrow \mathbf{(b)+b}$$

En este tipo de derivaciones se emplea el esquema denominado *Reconocimiento de Patrones*, primero se subdivide la cadena *w* en dos partes *uv*, y se buscan en los patrones las producciones que generen a alguno de los sufijos de *u*, conduciendo a la descomposición de *w* en u_1Av, este procedimiento se repite hasta obtener la reducción de toda la cadena a S.

Preguntas

a) ¿Es posible encontrar una gramática en la **FNCh** que genere a un Lenguaje Regular?

b) Sea *n* un número primo. ¿El lenguaje formado por todas las cadenas cuya longitud es múltiplo de *n* es regular?

c) ¿Para cada gramática independiente de contexto *G* existe un autómata finito no determinista *M* tal que $L(G) = L(M)$?

d) Sea *G* una gramática libre de contexto tal que sólo existe una regla para cada no terminal. ¿Es regular el lenguaje $L(G)$?

e) ¿Podrías explicar por qué los analizadores sintácticos LR(k) cubren una clase mucho más amplia de lenguajes que los analizadores LL(k)?

Ejercicios Capítulo 8

8.1 Construya una **GIC** para generar cada uno de los siguientes lenguajes:

a) $L = \{ \mathbf{a}^n \mathbf{bc}^{n+1} \mid n > 0 \}$

b) $L = \{ \mathbf{a}^m \mathbf{b}^n \mid m > n, n \geq 0 \}$

c) $L = \{ \mathbf{a}^{n+2} \mathbf{b}^n \mid n > 0 \}$

d) $L = \{ \mathbf{a}^{2n} \mathbf{bc}^n \mid n > 0 \}$

e) $L = \{ \mathbf{a}^n \mathbf{b}^m \mathbf{c}^{n+m} \mid n \geq 0 \text{ y } m \geq 0 \}$

f) $L = \{ \mathbf{a}^n \mathbf{b}^m \mathbf{c}^m \mathbf{a}^n \mid n \geq 0, m \geq 0 \}$

g) $L = \{ \mathbf{a}^n \mathbf{b}^n \mathbf{c}^m \mathbf{d}^m \mid n \geq 0 \text{ y } m \geq 0 \}$

h) $L = \{ \mathbf{a}^n \mathbf{b}^{n+m} \mathbf{c}^m \mid n \geq 0 \text{ y } m \geq 0 \}$

i) $L = \{ \mathbf{a}^m \mathbf{b}^n \mid 0 \leq n \leq m \leq 2n \}$

j) $L = \{ \mathbf{a}^n \mathbf{b}^{n+m} \mathbf{c}^{2m} \mid m \geq 0, n \geq 0 \}$

k) $L = \{ \mathbf{a}^m \mathbf{b}^n \mathbf{c}^p \mid m \geq 0, n \geq 0, p \geq 0, m \neq n + p \}$

l) $L = \{ \mathbf{a}^m \mathbf{b}^n \mathbf{c}^p \mid m = n \text{ o } n = p \}$

m) $L = \{ \mathbf{a}^m \mathbf{b}^n \mathbf{c}^p \mid m < n \text{ o } m > p \}$

n) $L = \{ \mathbf{a}^m \mathbf{b}^n \mathbf{c}^p \mid n > m + p \}$

o) $L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w = w^R \}$

8.2 Cada una de las siguientes **GIC**s generan a un Lenguaje Regular, encuentre la Expresión Regular de cada caso y obtenga una Gramática Regular equivalente:

a) $S \rightarrow A\mathbf{ab}B$, $A \rightarrow \mathbf{a}A \mid \mathbf{b}A \mid \varepsilon$, $B \rightarrow B\mathbf{ab} \mid B\mathbf{b} \mid \mathbf{ab} \mid \mathbf{b}$

b) $S \rightarrow SSS \mid \mathbf{a} \mid \mathbf{ab}$

c) S → AAS | **ab** | **aab**, A → **ab** | **ba** | ε

d) S → **a**S**b** | **a**S**a** | **b**S**a** | **b**S**b** | ε

e) S → AB, A → **a**A | A**b** | **a** | **b**, B → **a**B | **b**B | ε

f) S → AA | B, A → AAA | A**b** | **b**A | **a**, B → **b**B | ε

8.3 Identifique el lenguaje generado por cada una de las gramáticas siguientes:

a) S → **a**S**a** | **b**S**b** | ε

b) S → **a**S**a** | **b**S**b** | **a** | **b**

c) S → **a**S**a** | **b**S**b** | A, A → **a**B**b** | **b**B**a**, B → **a**B | **b**B | ε

d) S → **a**S | **a**S**b**S | ε

e) S → **a**S | **b**S | **a**

f) S → SS | **b**S | S**b** | **a**

8.4 Para cada una de las siguientes gramáticas, ¿Cuál es el tipo de analizador sintáctico **LL** más sencillo?

a) S → AB, A → **a**B | **b**A, B → **b**

b) S → BA**b**, A → **aa**A | **ab**B, B → **a**

c) S → **c**AB**c**, A → **a**A**a** | **c**, B → **b**B**b** | **c**

d) S → **aa**A**a** | **ab**B, A → **b**B, B → **a**

e) S → **a**A**b** | **ab**B, A → **a**B, B → **b**

Capítulo

9

Autómatas de Pila

Se define el concepto Lenguaje Independiente del Contexto, se define el Autómata de Pila como mecanismo para la identificación de este tipo de Lenguajes y se muestra su relación con las Gramáticas Independientes del Contexto.

Autómata de Pila Determinista

Un Autómata de Pila es un dispositivo que hace uso de una memoria infinita, la que opera como una pila, de manera semejante a la operación que se realiza en una cocina donde se lavan los platos, una vez secados se amontonan en la pila y cada vez que se requiere de un plato, siempre se toma el que está en la cima, a este tipo de modelos se le llama **UEPS** (última entrada, primera salida).

Este tipo de autómatas pasa a distintos estados de manera similar a un **AF**, pero al mismo tiempo, puede interactuar con la pila, agregando, sustituyendo o quitando símbolos; además el símbolo que se encuentre en la cima de la pila también puede determinar las transiciones del autómata.

El empleo de la pila le permite al autómata la identificación de cadenas para los lenguajes independientes del contexto.

Formalmente definimos a un Autómata de Pila Determinista (**APD**) como una sexteta $M = (Q, \Sigma, \Gamma, s, F, \delta)$, en donde Q es el conjunto de estados, Σ es el alfabeto de entrada, Γ es el alfabeto de la Pila, s es el estado inicial, F es el conjunto de estados de aceptación y δ es el conjunto de Reglas de Transición, de las cuales, existe *cuando mucho* una transición aplicable para cada configuración posible.

Operaciones en la Pila

Las transiciones de un **AP** nos sirven para realizar en la pila las operaciones de apilar, desapilar, reemplazar símbolos e incluso para dejar la pila intacta. Además, estas operaciones pueden ser de dos tipos: las condicionadas o las incondicionales.

Una pila es representada por una cadena de Símbolos de la pila, por convención, se considera que el primer símbolo de la cadena es el que corresponde a la cima de la pila y que el símbolo que se apila, se concatena a la izquierda de la misma.

Para expresar las operaciones en la pila empleamos la siguiente notación:

Operaciones incondicionales

$\varepsilon \rightarrow \varepsilon$ denota que no se hace ninguna operación con la pila, ésta se queda intacta.

$\varepsilon \rightarrow A$ denota que se agrega incondicionalmente el símbolo A en la cima de la pila.

Operaciones condicionadas

$A \rightarrow A$ denota que si en la cima de la pila se encuentra el símbolo A, éste permanece ahí, la pila no se altera.

$A \rightarrow w$ denota que si en la cima de la pila se encuentra el símbolo A, éste será reemplazado por la cadena *w*.

$A \rightarrow \varepsilon$ denota que si en la cima de la pila se encuentra el símbolo A, éste se deberá desapilar, (equivale a reemplazarlo por la cadena vacía)

$A \rightarrow wA$ denota que si en la cima de la pila se encuentra el símbolo A, se apilará sobre éste a la cadena *w*.

Las acciones a realizar por un **APD** se representan por medio de transiciones definidas por la función δ: $Q \times (\sum \cup \varepsilon) \times (\Gamma \cup \varepsilon) \rightarrow Q \times \Gamma^*$. Estas transiciones se pueden interpretar como en los ejemplos siguientes:

La transición $\delta(q_1, \mathbf{a}, A) = (q_2, \varepsilon)$, se aplica cuando el **APD** se encuentra en el estado q_1, tiene el símbolo de entrada **a** y el símbolo A se encuentra en la cima de la pila, entonces se realiza el cambio del estado q_1 al estado q_2 y se quita el símbolo A de la cima de la pila.

Mientras que la transición $\delta(q_2, \mathbf{a}, \varepsilon) = (q_1, \varepsilon)$ cambia del estado q_2 al q_1 con el símbolo de entrada **a**, sin tomar en cuenta a la pila ni actuar sobre ella. Y la siguiente transición: $\delta(q_3, \mathbf{a}, \varepsilon) = (q_1, A)$ cambia del estado q_3 al q_1 con el símbolo de entrada **a** y, sin importar lo que hubiera en la cima de la pila, coloca en la pila al símbolo A.

Ejemplo 1

Sea el **APD** definido por: Q = { q_0, q_1 }, Σ = {**a**, **b**}, Γ = { A }, s = q_0, F = {q_1} y δ está dado por las siguientes transiciones:

$$\delta(q_0, \mathbf{a}, \varepsilon) = (q_0, A)$$

$$\delta(q_0, \mathbf{b}, A) = (q_1, \varepsilon)$$

$$\delta(q_1, \mathbf{b}, A) = (q_1, \varepsilon)$$

Una cadena *w* será aceptada por este **APD** siempre que sea posible llegar al estado de aceptación q_1 y agotar la cadena al mismo tiempo que la pila quede vacía. En cambio, si el autómata se detiene por no existir una transición definida, sin haber agotado la cadena o sin haber vaciado la pila, la cadena será rechazada.

Gráficamente el **APD** anterior lo podemos representar por medio del diagrama de transiciones mostrado en la siguiente figura, donde cada una de las transiciones involucra un símbolo de entrada y una acción sobre la pila:

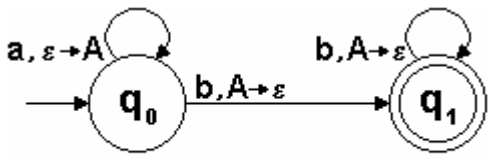


Figura 9.1

Si analizamos detalladamente al **APD** anterior, podemos observar que se apila un símbolo A por cada símbolo de entrada **a**, después, al encontrar un símbolo de entrada **b**, cambia de estado a q_1 y desapila al símbolo A. En el estado q_1 continúa desapilando un símbolo A por cada **b** de entrada, eventualmente se agota la cadena y si se encuentra con la pila vacía, la cadena es aceptada, de este análisis podemos deducir que este **APD** acepta las cadenas del lenguaje siguiente: L = { $\mathbf{a}^n\mathbf{b}^n$ | n > 0 }

El autómata pasa por distintas fases instantáneas, del mismo modo que los cuadros de una película, en la siguiente figura se ilustra como la cadena *w* = **aabb** es aceptada por este autómata.

Figura 9.2

Cada una de las fases se describe por medio de la combinación de los tres elementos del Autómata: el estado actual, el estado actual de la cadena, indicando el símbolo de entrada correspondiente y la cadena que forma la pila, señalando el símbolo ubicado en la cima de la pila.

En notación más compacta y práctica se hace uso del símbolo $\vdash$ para denotar el paso entre dos descripciones instantáneas consecutivas, de este modo podemos mostrar cómo procesa el autómata a la cadena anterior pasando por cada una de las distintas fases instantáneas hasta alcanzar la configuración de aceptación:

$$(q_0, \underline{\mathbf{a}}\mathbf{abb}, \underline{\varepsilon}) \vdash (q_0, \underline{\mathbf{a}}\mathbf{bb}, \underline{A}) \vdash (q_0, \underline{\mathbf{b}}\mathbf{b}, \underline{A}A) \vdash (q_1, \underline{\mathbf{b}}, \underline{A}) \vdash (q_1, \underline{\varepsilon}, \underline{\varepsilon})$$

Las cadenas $w_1 = \mathbf{aab}$, $w_2 = \mathbf{abab}$ y $w_3 = \mathbf{abb}$, no pertenecen al lenguaje $\mathbf{a}^n\mathbf{b}^n$, por lo que no deben ser aceptadas por este autómata, en todos los casos se detiene el autómata en q_1 pero en el primer caso le quedan símbolos en la pila, mientras que en los otros dos casos no se agota la cadena, tal como se ilustra por medio de sus respectivas descripciones instantáneas:

1: $(q_0, \underline{\mathbf{a}}\mathbf{ab}, \underline{\varepsilon}) \vdash (q_0, \underline{\mathbf{a}}\mathbf{b}, \underline{A}) \vdash (q_0, \underline{\mathbf{b}}, \underline{A}A) \vdash (q_1, \underline{\varepsilon}, \underline{A})$

2: $(q_0, \underline{\mathbf{a}}\mathbf{bab}, \underline{\varepsilon}) \vdash (q_0, \underline{\mathbf{b}}\mathbf{ab}, \underline{A}) \vdash (q_1, \underline{\mathbf{a}}\mathbf{b}, \underline{\varepsilon})$

3: $(q_0, \underline{\mathbf{a}}\mathbf{bb}, \underline{\varepsilon}) \vdash (q_0, \underline{\mathbf{b}}\mathbf{b}, \underline{A}) \vdash (q_1, \underline{\mathbf{b}}, \underline{\varepsilon})$

Finalmente, si deseamos que este **AFD** también acepte a la cadena vacía, bastará con marcar a q_0 como estado de aceptación.

Ejemplo 2

Para mostrar que todo autómata finito también puede ser considerado como un caso especial de autómata de pila, que nunca realizará operaciones sobre la pila, considérese el siguiente **AFN**:

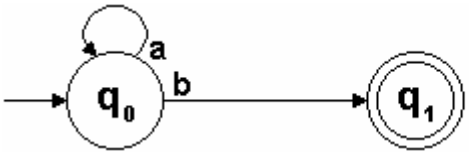


Figura 9.3

El **APD** equivalente estará dado por: $Q = \{q_0, q_1\}$, $\Sigma = \{\mathbf{a}, \mathbf{b}\}$, $\Gamma = \varnothing$, $s = q_0$, $F = \{q_1\}$ y las transiciones siguientes:

$$\delta(q_0, \mathbf{a}, \varepsilon) = (q_0, \varepsilon)$$

$$\delta(q_0, \mathbf{b}, \varepsilon) = (q_1, \varepsilon)$$

Un **APD** que solamente apila y desapila una vez le se llama **APD** *de Una Vuelta*, y sirve para reconocimiento de lenguajes generados por *Gramáticas Lineales*, es decir, de aquellas que contienen cuando mucho un Símbolo No Terminal del lado derecho en cada una de sus producciones. La clase de Gramáticas Regulares se considera un subconjunto de la clase de Gramáticas Lineales.

Ejemplo 3

Construir un **APD** que acepte el Lenguaje L = { $w\mathbf{c}w^R$ | $w \in$ {**a**, **b**}*}, entonces consideremos que: Q = {q_0, q_1}, Σ = { **a**, **b**, **c** }, Γ = { A, B }, s = q_0, F = {q_1} y las transiciones siguientes:

$\delta(q_0, \mathbf{a}, \varepsilon) = (q_0, A)$ $\quad\quad$ $\delta(q_1, \mathbf{a}, A) = (q_1, \varepsilon)$

$\delta(q_0, \mathbf{b}, \varepsilon) = (q_0, B)$ $\quad\quad$ $\delta(q_1, \mathbf{b}, B) = (q_1, \varepsilon)$

$\delta(q_0, \mathbf{c}, \varepsilon) = (q_1, \varepsilon)$

Este **APD** *de Una Vuelta* trabaja de la siguiente manera, primero apila una A por cada **a** y una B por cada **b** de la cadena *w*, al encontrar la **c** cambia de estado sin modificar la pila, y después, desapila una A por cada **a** y una B por cada **b** de la cadena w^R, hasta agotar la cadena y volver a encontrar la pila vacía, tal como se muestra en la siguiente figura:

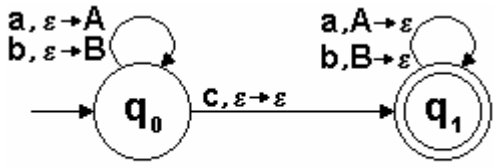


Figura 9.4

Una cadena aceptada por este autómata sería *w* = **abbcbba**, la cual pasa por las siguientes fases instantáneas:

(q_0, **a**bbcbba, ε) ⊢ (q_0, **b**bcbba, A) ⊢ (q_0, **b**cbba, BA) ⊢ (q_0, **c**bba, BBA) ⊢

(q_1, **b**ba, BBA) ⊢ (q_1, **b**a, BA) ⊢ (q_1, **a**, A) ⊢ (q_1, ε, ε)

Si la cadena no es palíndroma, entonces el **APD** se detendrá en q_1 por alguno de los siguientes motivos: que no coincida el símbolo de entrada con el esperado en la cima de la pila, que se vacíe la pila sin haberse agotado la cadena o que se agote la cadena sin vaciarse la cadena.

Autómata de Pila No Determinista

Un Autómata de Pila No Determinista (**APN**) es la sexteta M = (Q, $\sum$, Γ, s, F, Δ), que contiene dos o más transiciones definidas para una configuración dada. Las transiciones para este tipo de autómatas provienen de la siguiente función de transición Δ: $Q \times (\sum \cup \varepsilon) \times (\Gamma \cup \varepsilon) \rightarrow 2^{Q \times \Gamma^*}$. Los autómatas de este tipo son más poderosos que los **APD**s, por lo que permiten el reconocimiento de algunos lenguajes que no pueden ser identificados por medio de ningún **APD**.

Debido a esto, los **LIC**s se subdividen en dos subtipos, por un lado a los lenguajes que pueden ser reconocidos por algún **APD** los identificaremos como **LICD**s, mientras que a los lenguajes que no pueden ser reconocidos por ningún **APD**, los denominaremos como **LICN**s, ya que requieren necesariamente de un **APN** para su análisis. Para mostrar que un **APN** resulta un dispositivo más poderoso que cualquier **APD**, consideremos el siguiente caso:

Ejemplo 1

Resulta imposible construir un **APD** que acepte el Lenguaje L = { ww^R | $w \in$ {**a**,**b**}* }, entonces consideremos como alternativa al **APN**, en el que: Q = { q_0, q_1 }, $\sum$ = { **a**, **b** }, Γ = { A, B }, s = q_0, F = { q_1 } y las transiciones siguientes:

$\Delta(q_0, \mathbf{a}, \varepsilon) = \{(q_0, A)\}$ $\quad$ $\Delta(q_1, \mathbf{a}, A) = \{(q_1, \varepsilon)\}$

$\Delta(q_0, \mathbf{b}, \varepsilon) = \{(q_0, B)\}$ $\quad$ $\Delta(q_1, \mathbf{b}, B) = \{(q_1, \varepsilon)\}$

$\Delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \varepsilon)\}$

Este **APN**, cuyo diagrama de transiciones se muestra en la figura 9.5, es casi idéntico al **APD** presentado en un ejemplo anterior, sin embargo tiene una transición épsilon que permite pasar del estado q_0 al q_1 de forma no determinista.

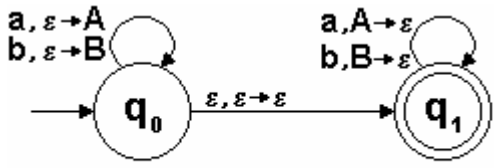


Figura 9.5

Por ejemplo, la cadena *w* = **abba**, pertenece al lenguaje referido y por tanto debe ser aceptada por el autómata, en la siguiente figura se representa este proceso, y podemos observar que existen múltiples trayectorias posibles, en la mayoría de los

casos no se vacía la pila o no se agota la cadena, u ocurren ambas cosas; sin embargo, existe un camino en el que cumplen ambas condiciones, se vacía la pila y se agota la cadena, por lo tanto ésta es aceptada por el **APD**.

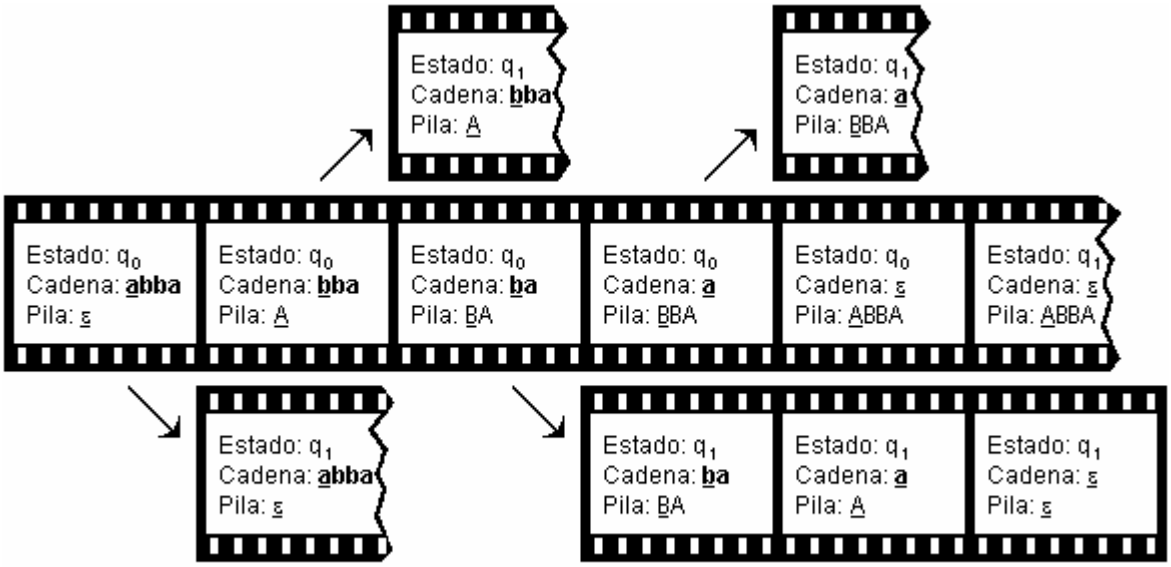


Figura 9.6

Ejemplo 2

Otro caso de un lenguaje que requiere de un **APN** para ser reconocido es el siguiente: $L = \{ \mathbf{a}^m\mathbf{b}^n \mid m > n > 0 \}$, donde: $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{\mathbf{a}, \mathbf{b}\}$, $\Gamma = \{A\}$, $s = q_0$, $F = \{q_2\}$ y las transiciones siguientes:

$\Delta(q_0, \mathbf{a}, \varepsilon) = \{(q_0, A)\}$

$\Delta(q_0, \mathbf{b}, A) = \{(q_1, \varepsilon)\}$

$\Delta(q_1, \mathbf{b}, A) = \{(q_1, \varepsilon)\}$

$\Delta(q_1, \varepsilon, A) = \{(q_2, \varepsilon)\}$

$\Delta(q_2, \varepsilon, A) = \{(q_2, \varepsilon)\}$

Donde las últimas dos transiciones épsilon nos permiten desapilar las As que haya en exceso en la pila, asegurándonos que al menos exista una para poder acceder al estado de aceptación q_2, tal como se muestra en la siguiente figura:

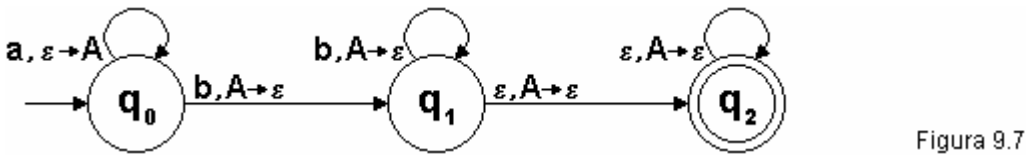


Figura 9.7

No-determinismo Removible

El No Determinismo del ejemplo anterior es removible, ya que en casos como este, se puede utilizar un pequeño truco para construir un **APD** equivalente, consistente en agregar al final de la cadena un símbolo especial (**$**), que indique el final de la cadena, en este caso las transiciones se redefinirían como sigue:

$\delta(q_0, \mathbf{a}, \varepsilon) = \{(q_0, A)\}$ $\quad\quad$ $\delta(q_0, \mathbf{b}, A) = \{(q_1, \varepsilon)\}$

$\delta(q_1, \mathbf{b}, A) = \{(q_1, \varepsilon)\}$ $\quad\quad$ $\delta(q_1, \mathbf{\$}, A) = \{(q_2, \varepsilon)\}$

$\delta(q_2, \varepsilon, A) = \{(q_2, \varepsilon)\}$

Quedando el **APD** modificado como se muestra en la figura a continuación:

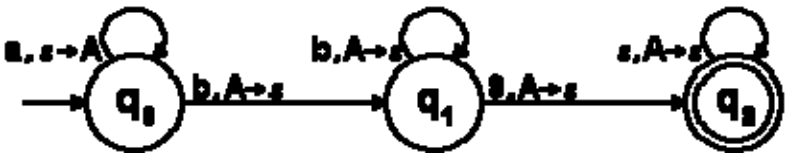


Figura 8.8

De esta manera, el reconocimiento de la cadena *w* = **aaab**, se convierte en la aceptación de la cadena: *w'* = **aaab$**, como se muestra a continuación:

$(q_0,\underline{\mathbf{a}}\mathbf{aab\$},\underline{\varepsilon}) \vdash (q_0,\underline{\mathbf{a}}\mathbf{ab\$},\underline{A}) \vdash (q_0,\underline{\mathbf{a}}\mathbf{b\$},\underline{A}A) \vdash (q_0,\underline{\mathbf{b}}\mathbf{\$},\underline{A}AA) \vdash (q_1,\underline{\mathbf{\$}},\underline{A}A) \vdash (q_2,\underline{\varepsilon},\underline{A}) \vdash (q_2,\underline{\varepsilon},\underline{\varepsilon})$

Además de que se puede omitir la última transición si usamos el criterio anterior.

Autómatas de Pila Condicionados

En los ejemplos precedentes la mayoría de las transiciones son incondicionales, con excepción de las que desapilan, que no pueden serlo; pero en ciertos casos se requiere que todas las transiciones sean condicionadas, para estos casos se suele utilizar una variante de **AP**, tanto determinista como no determinista, en la que todas las transiciones se definen de forma condicionada.

Para ello se requiere de la suposición de que originalmente existe un símbolo en la pila, al que denotaremos por Z, el cual viene a considerarse como el séptimo elemento en la definición de esta variante de **AFD**s: M = (Q, $\sum$, Γ, s, F, δ, Z), o para el caso de los **AFN**s: M = (Q, $\sum$, Γ, s, F, Δ, Z).

Esta suposición es equivalente a considerar un autómata de pila convencional en el que existe una transición inicial de la forma: $\Delta(q_0', \varepsilon, \varepsilon) = (q_0, Z)$, cuyo único fin es el de colocar al símbolo Z en la pila, y rescribiendo todas las transiciones incondicionales de forma condicional.

Los **APN**s condicionados se utilizan para el reconocimiento de lenguajes que requieren identificar el final de la pila, pero sin vaciarla, se suelen construir de tal forma que tengan un único estado de aceptación, al cual se acceda solamente cuando se tenga la certeza de aceptar la cadena; en este estado no es usual que haya transiciones, pero en ciertos casos las puede haber.

Con este enfoque ya no es necesario exigir que la pila se vacíe para aceptar la cadena, lo único que necesitamos verificar es que la cadena se agote y estar en un estado de aceptación cuando esto suceda.

Ejemplo 1

Podemos construir un **APD** condicionado que acepte el Lenguaje L = { $\mathbf{a}^n\mathbf{b}^n$ | n > 0}, a partir del **APD** mostrado previamente en este capítulo: Q = { q_0, q_1, q_2}, s = q_0, F = {q_2}, Γ = { A, Z }, $\sum$ = { **a**, **b** } y las transiciones siguientes:

$\delta(q_0, \mathbf{a}, Z) = (q_0, AZ)$ $\quad\quad$ $\delta(q_0, \mathbf{a}, A) = (q_0, AA)$

$\delta(q_0, \mathbf{b}, A) = (q_1, \varepsilon)$ $\quad\quad$ $\delta(q_1, \mathbf{b}, A) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, Z) = (q_2, Z)$

Este autómata es determinista a pesar de contener una transición épsilon. Obsérvese que en este **APD**, hay que acceder al estado de aceptación cuando se agota la cadena y se encuentra a Z en la pila, y como se dijo antes, no es necesario quitarla de la pila.

Ejemplo 2

Construir un **APD** condicionado, para reconocer el lenguaje: L = { $\mathbf{a}^m\mathbf{b}^n$ | n > m > 0 }.

La solución es muy semejante al anterior, pero con la diferencia que debemos encontrar al símbolo Z antes de agotar la cadena, cuando esto sucede, pasamos al estado q_2, que es el de aceptación, para agotar ahí la cadena, verificando que exclusivamente se acepten **b**s.

Las transiciones necesarias son:

$\delta(q_0, \mathbf{a}, Z) = (q_0, AZ)$ $\quad\quad$ $\delta(q_0, \mathbf{a}, A) = (q_0, AA)$

$\delta(q_0, \mathbf{b}, A) = (q_1, \varepsilon)$ $\quad\quad$ $\delta(q_1, \mathbf{b}, A) = (q_1, \varepsilon)$

$\delta(q_1, \mathbf{b}, Z) = (q_2, Z)$ $\quad\quad$ $\delta(q_2, \mathbf{b}, Z) = (q_2, Z)$

Ejemplo 3

Construir un **APN** condicionado que acepte a: L = { $w \in$ {**a**, **b**}* | $N_{\mathbf{a}}(w) = N_{\mathbf{b}}(w)$ }. El **APN** está dado por: Q = { q_0, q_1 }, s = q_0, Σ = { **a**, **b** }, Γ = { A, B, Z }, F = { q_1 } y las transiciones siguientes:

$\Delta(q_0, \mathbf{a}, Z) = (q_0, AZ)$ $\Delta(q_0, \mathbf{a}, A) = (q_0, AA)$

$\Delta(q_0, \mathbf{b}, Z) = (q_0, BZ)$ $\Delta(q_0, \mathbf{b}, B) = (q_0, BB)$

$\Delta(q_0, \mathbf{b}, A) = (q_0, \varepsilon)$ $\Delta(q_0, \mathbf{a}, B) = (q_0, \varepsilon)$

$\Delta(q_0, \varepsilon, Z) = (q_1, Z)$

Este **APN** trabaja de la siguiente manera, primero apila una A por cada **a**, pero si el símbolo en la cima de la pila es una B, entonces la desapila. Si el símbolo de entrada es una **b**, entonces apila una B, excepto cuando el símbolo en la cima es una A, porque entonces la desapila.

Cada vez que se encuentra Z en la cima de la pila significa que la parte de la cadena que ya fue analizada contiene la misma cantidad de **a**s que de **b**s. Este autómata pasa al estado q_1 de manera No Determinista cada vez que la Z esté en la cima de la pila, sin embargo, solamente cuando la cadena se agote y se encuentre el símbolo Z en la cima de la pila, es pertinente pasar al estado q_1 y aceptar la cadena.

Una cadena aceptada por este autómata sería *w* = **abba**, la cual pasa por la siguiente secuencia de fases instantáneas:

$(q_0, \underline{\mathbf{a}}\mathbf{bba}, \underline{Z}) \vdash (q_0, \underline{\mathbf{b}}\mathbf{ba}, \underline{A}Z) \vdash (q_0, \underline{\mathbf{b}}\mathbf{a}, \underline{Z}) \vdash (q_0, \underline{\mathbf{a}}, \underline{B}Z) \vdash (q_0, \underline{\varepsilon}, \underline{Z}) \vdash (q_1, \underline{\varepsilon}, \underline{Z})$

En la tercera fase existe una transición épsilon hacia la configuración (q_1, **ba**, Z), que no se indica, puesto que no nos sirve para el reconocimiento de la cadena.

Para forzar a que la transición épsilon solamente se utilice cuando la cadena se haya agotado, para aceptarla, accediendo al estado de aceptación, podemos emplear la técnica del símbolo de fin de cadena (**$**), mostrada anteriormente, el **APD** equivalente para esta variante sería:

$\delta(q_0, \mathbf{a}, Z) = (q_0, AZ)$ $\delta(q_0, \mathbf{a}, A) = (q_0, AA)$

$\delta(q_0, \mathbf{b}, Z) = (q_0, BZ)$ $\delta(q_0, \mathbf{b}, B) = (q_0, BB)$

$\delta(q_0, \mathbf{b}, A) = (q_0, \varepsilon)$ $\delta(q_0, \mathbf{a}, B) = (q_0, \varepsilon)$

$\delta(q_0, \mathbf{\$}, Z) = (q_1, Z)$

Y la secuencia de fases instantáneas para aceptar la cadena *w'* = **abba$**, será:

$(q_0, \underline{\mathbf{a}}\mathbf{bba\$}, \underline{Z}) \vdash (q_0, \underline{\mathbf{b}}\mathbf{ba\$}, \underline{A}Z) \vdash (q_0, \underline{\mathbf{b}}\mathbf{a\$}, \underline{Z}) \vdash (q_0, \underline{\mathbf{a}}\mathbf{\$}, \underline{B}Z) \vdash (q_0, \underline{\mathbf{\$}}, \underline{Z}) \vdash (q_1, \underline{\varepsilon}, \underline{Z})$

APD Complemento

Dado M un **APD condicionado**, que acepta a L(M), se puede construir un **APD** M', tal que acepte a L^C, agregando las transiciones necesarias hacia un estado de rechazo, de modo que queden definidas todas las configuraciones posibles y cambiando los estados de aceptación por estados de no-aceptación y viceversa.

Ejemplo

Construya un **APD** para el lenguaje complemento de L = { $\mathbf{a}^n\mathbf{b}^n$ | n > 0}, a partir de **APD** condicionado visto en la página 151, primeramente copiamos las cinco transiciones originales:

$\delta(q_0, \mathbf{a}, Z) = (q_0, AZ)$ $\quad\quad$ $\delta(q_0, \mathbf{a}, A) = (q_0, AA)$

$\delta(q_0, \mathbf{b}, A) = (q_1, \varepsilon)$ $\quad\quad$ $\delta(q_1, \mathbf{b}, A) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, Z) = (q_2, Z)$

Posteriormente, tenemos que agregar las siguientes transiciones hacia q_3, (el estado de rechazo de M)

$\delta(q_0, \mathbf{b}, Z) = (q_3, Z)$ $\quad\quad$ $\delta(q_1, \mathbf{a}, A) = (q_3, \varepsilon)$

$\delta(q_2, \mathbf{b}, Z) = (q_3, Z)$ $\quad\quad$ $\delta(q_2, \mathbf{a}, Z) = (q_3, Z)$

En el estado q_3 desapilamos todas las As y terminamos de aceptar el resto de la cadena de entrada, con las siguientes transiciones:

$\delta(q_3, \varepsilon, A) = (q_3, \varepsilon)$ $\quad\quad$ $\delta(q_3, \mathbf{a}, Z) = (q_3, Z)$

$\delta(q_3, \mathbf{b}, Z) = (q_3, Z)$

Para este **APD** complemento, los estados de aceptación son: F = { q_0, q_1, q_3 }

Construcción de un APN dada una GIC

Para demostrar que cualquier Lenguaje Independiente del Contexto L(G), definido por una **GIC** dada, es aceptado por algún **APN**, describiremos un procedimiento para construir un **APN** que acepte a L(G).

Sea una **GIC** definida por G = (N, $\sum$, S, P), deseamos construir un **APN** que acepte L(G), para ello definimos los elementos del **APN** como sigue: Q = { q_0, q_1 }, Γ = N $\cup$ $\sum$, s = q_0, F = {q_1} y para obtener a Δ se aplican los siguientes criterios:

- Una transición inicial para introducir en la pila el Símbolo No Terminal Inicial:
  $\Delta(q_0, \varepsilon, \varepsilon)$ = { (q_1, S) }
- Para cada producción de la forma A $\rightarrow$ *w*, se define una transición que substituye en la pila al símbolo no terminal A, por la cadena *w*:
  $\Delta(q_1, \varepsilon, A)$ = { (q_1, w) }
- Definimos una transición que desapila el símbolo de la cima de la pila para cada Símbolo Terminal que coincida con el siguiente símbolo de entrada:
  $\Delta(q_1, \mathbf{a}, \mathbf{a})$ = { (q_1, ε) }

Ejemplo

Sea la gramática: S $\rightarrow$ **a**S**a** | **b**S**b** | **c**, que genera al lenguaje L = { *w***c***w*^{R}| *w*$\in${**a**,**b**}*}, aplicando los criterios anteriores, las reglas de transición del autómata son:

$\Delta(q_0, \varepsilon, \varepsilon)$ = { (q_1, S) }

$\Delta(q_1, \varepsilon, S)$ = { (q_1, **a**S**a**), (q_1, **b**S**b**), (q_1, **c**) }

$\Delta(q_1, \mathbf{a}, \mathbf{a}) = \Delta(q_1, \mathbf{b}, \mathbf{b}) = \Delta(q_1, \mathbf{c}, \mathbf{c})$ = { (q_1, ε) }

La cadena w = **abcba** es aceptada mediante la siguiente secuencia:

(q_0, **a**bcba, $\underline{\varepsilon}$) $\vdash$ (q_1, **a**bcba, $\underline{S}$) $\vdash$ (q_1, **a**bcba, **a**S**a**) $\vdash$ (q_1, **b**cba, $\underline{S}$**a**) $\vdash$ (q_1, **b**cba, **b**S**ba**)

$\vdash$ (q_1, **c**ba, $\underline{S}$**ba**) $\vdash$ (q_1, **c**ba, **c**ba) $\vdash$ (q_1, **b**a, **b**a) $\vdash$ (q_1, **a**, **a**) $\vdash$ (q_1, $\underline{\varepsilon}$, $\underline{\varepsilon}$)

El autómata obtenido por medio de esta técnica es un **APN**, a pesar que se trata de un **LICD**, como ya se demostró previamente, por ejemplo, en el tercer paso de la derivación se reemplazó la S por **a**S**a**, puesto que sabíamos que el símbolo de entrada era una **a**, pero el autómata debió reemplazar la S de manera no determinista por las otras dos alternativas también, encontrado más adelante que se trataba de una sustitución infructuosa.

Técnica de Anticipación

En algunos casos podemos modificar a un **APN** para obtener un **APD** equivalente, por medio de la Técnica de Anticipación, que nos permite saber por adelantado cuál es el símbolo de entrada siguiente, para poder seleccionar adecuadamente la transición

requerida. Esta técnica es interesante, pues es la que se emplea en los Analizadores Sintácticos para las gramáticas tipo LL(k).

Aplicando esta técnica, agregamos tres nuevos estados: q_a, q_b y q_c, hacia los cuales se definen las transiciones dependiendo del símbolo anticipado, entonces para definir un **APD** equivalente al **APN** anterior, se tiene que: Q = { q_0, q_1, q_a, q_b, q_c }, Σ = {**a**,**b**,**c**}, Γ = { S, **a**, **b**, **c** } s = q_0, F = {q_1} y las transiciones siguientes:

$\Delta(q_0, \varepsilon, \varepsilon) = (q_1, S)$ $\Delta(q_1, \mathbf{a}, \varepsilon) = (q_a, \varepsilon)$

$\Delta(q_a, \varepsilon, \mathbf{a}) = (q_1, \varepsilon)$ $\Delta(q_1, \mathbf{b}, \varepsilon) = (q_b, \varepsilon)$

$\Delta(q_b, \varepsilon, \mathbf{b}) = (q_1, \varepsilon)$ $\Delta(q_1, \mathbf{c}, \varepsilon) = (q_c, \varepsilon)$

$\Delta(q_c, \varepsilon, \mathbf{c}) = (q_1, \varepsilon)$ $\Delta(q_a, \varepsilon, S) = (q_a, \mathbf{a}S\mathbf{a})$

$\Delta(q_b, \varepsilon, S) = (q_b, \mathbf{b}S\mathbf{b})$ $\Delta(q_c, \varepsilon, S) = (q_c, \mathbf{c})$

La aceptación de la cadena *w* = **abcba** sigue la secuencia siguiente:

$(q_0, \underline{\mathbf{a}}\mathbf{bcba}, \underline{\varepsilon}) \vdash (q_1, \underline{\mathbf{a}}\mathbf{bcba}, \underline{S}) \vdash (q_a, \underline{\mathbf{b}}\mathbf{cba}, \underline{S}) \vdash (q_a, \underline{\mathbf{b}}\mathbf{cba}, \underline{\mathbf{a}}S\mathbf{a}) \vdash (q_1, \underline{\mathbf{b}}\mathbf{cba}, \underline{S}\mathbf{a}) \vdash$

$(q_b, \underline{\mathbf{c}}\mathbf{ba}, \underline{S}\mathbf{a}) \vdash (q_b, \underline{\mathbf{c}}\mathbf{ba}, \underline{\mathbf{b}}S\mathbf{ba}) \vdash (q_1, \underline{\mathbf{c}}\mathbf{ba}, \underline{S}\mathbf{ba}) \vdash (q_c, \underline{\mathbf{b}}\mathbf{a}, \underline{S}\mathbf{ba}) \vdash (q_c, \underline{\mathbf{b}}\mathbf{a}, \underline{\mathbf{c}}\mathbf{ba}) \vdash$

$(q_1, \underline{\mathbf{b}}\mathbf{a}, \underline{\mathbf{b}}\mathbf{a}) \vdash (q_b, \underline{\mathbf{a}}, \underline{\mathbf{b}}\mathbf{a}) \vdash (q_1, \underline{\mathbf{a}}, \underline{\mathbf{a}}) \vdash (q_a, \underline{\varepsilon}, \underline{\mathbf{a}}) \vdash (q_1, \underline{\varepsilon}, \underline{\varepsilon})$

Este tipo de dispositivo se le conoce con el nombre de *Derivador*, ya que nos permite reconstruir fácilmente el árbol de derivación que genera la cadena en cuestión, detectando las reglas que se aplicaron a cada paso.

Por ejemplo, la secuencia anterior nos indica que en el tercer paso se utilizó la producción S $\rightarrow$ **a**S**a**, luego en el sexto se utilizó la producción S $\rightarrow$ **b**S**b** y en el noveno paso se aplicó S $\rightarrow$ **c**, de ahí se obtiene el árbol de derivación mostrado en la figura 9.9:

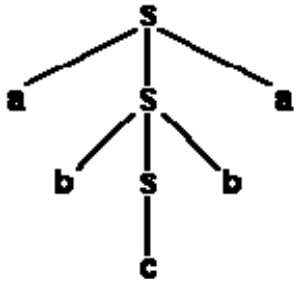


Figura 9.3

Técnica de Postergación

Para poder aplicar la técnica de Anticipación eficazmente y construir un Derivador en los casos donde existen producciones que inician del lado derecho con un símbolo no terminal, debemos buscar una gramática equivalente en la **FNG**, Vg., supongamos que tenemos la siguiente gramática:

$$S \rightarrow A \mid B$$

$$A \rightarrow \mathbf{a}A \mid \mathbf{b}$$

$$B \rightarrow \mathbf{b}B \mid \mathbf{a}$$

Podemos reemplazar las producciones del símbolo S por las producciones de A y de B, tal como se muestra en la siguiente gramática equivalente en **FNG**:

$$S \rightarrow \mathbf{a}A \mid \mathbf{b} \mid \mathbf{b}B \mid \mathbf{a}$$

$$A \rightarrow \mathbf{a}A \mid \mathbf{b}$$

$$B \rightarrow \mathbf{b}B \mid \mathbf{a}$$

Sin embargo, hay varias producciones que inician con el mismo símbolo terminal, por lo que se requiere emplear otra técnica adicional, llamada de Postergación por *Factorización* a la Izquierda, esto reduce el valor de k en las gramáticas LL(k), y que consiste en que las producciones de S se reemplazan por otras equivalentes, donde definimos dos nuevos símbolos no terminales C y D, quedando finalmente la siguiente gramática:

$$S \rightarrow \mathbf{a}C \mid \mathbf{b}D$$

$$A \rightarrow \mathbf{a}A \mid \mathbf{b}$$

$$B \rightarrow \mathbf{b}B \mid \mathbf{a}$$

$$C \rightarrow \mathbf{a}A \mid \mathbf{b} \mid \varepsilon$$

$$D \rightarrow \mathbf{b}B \mid \mathbf{a} \mid \varepsilon$$

Construcción de las GICs

Ahora procederemos a construir la **GIC** que genere el lenguaje aceptado por un **AP** dado. Esta gramática se definirá con símbolos no terminales de la forma $[q_iAq_j]$, con objeto de mantener la referencia de las transiciones de donde proceden. Para aplicar esta metodología, es necesario que trate de un **APD** o un **APN** condicionado, el cual debe vaciar la pila al llegar al estado de aceptación, que debe ser único.

En caso que no se cumpla alguna de estas condiciones, se deberá modificar al autómata a una forma equivalente que las satisfaga, agregándole las transiciones, símbolos No Terminales y estados que se requieran.

Los criterios básicos son:

- El símbolo inicial de la **GIC** es: S = $[q_0Zq_f]$, donde q_f es el estado de aceptación, Z es el símbolo inicial de la pila y q_0 es el estado inicial.
- Las transiciones de la forma: $\Delta(q_i, \mathbf{a}, A) = (q_j, \varepsilon)$, nos generan producciones de la forma: $[q_iAq_j] \rightarrow \mathbf{a}$
- Las transiciones de la forma: $\Delta(q_i, \mathbf{a}, A) = (q_j, B)$, nos dan producciones de la forma: $[q_iAq_m] \rightarrow \mathbf{a}[q_jBq_m]$, para cualquier estado q_m de Q.
- Las transiciones de la forma: $\Delta(q_i, \mathbf{a}, A) = (q_j, BC)$, nos dan producciones de la forma: $[q_iAq_m] \rightarrow \mathbf{a}[q_jBq_n]\,[q_nCq_m]$, para cualesquier estados q_n y q_m de Q.

Ejemplo 1

Construir la **GIC** que genere el lenguaje L = { $\mathbf{a}^n\mathbf{b}^n$ | n > 0 }, aceptado por el **APD** condicionado visto en la página 143, pero vaciando la pila al acceder a q_2, primero, en la **GIC** que vamos a construir se considera como símbolo inicial a S = $[q_0Zq_2]$.

Ahora, iniciando el proceso con las transiciones que decrementan la pila, a saber:

$$\delta(q_0, \mathbf{b}, A) = \{ (q_1, \varepsilon) \}$$

$$\delta(q_1, \mathbf{b}, A) = \{ (q_1, \varepsilon) \}$$

$$\delta(q_1, \varepsilon, Z) = \{ (q_2, \varepsilon) \}$$

Encontramos que éstas últimas nos originan las producciones siguientes:

$$[q_0Aq_1] \rightarrow \mathbf{b}$$

$$[q_1Aq_1] \rightarrow \mathbf{b}$$

$$[q_1Zq_2] \rightarrow \varepsilon$$

Continuando con las que apilan, que son:

$$\delta(q_0, \mathbf{a}, Z) = (q_0, AZ)$$

$$\delta(q_0, \mathbf{a}, A) = (q_0, AA)$$

Observamos que estas transiciones nos producen las siguientes expresiones:

$[q_0Zq_m] \rightarrow \mathbf{a}[q_0Aq_n]\,[q_nZq_m]$, para cualesquier estados q_n y q_m de Q.

$[q_0Aq_m] \rightarrow \mathbf{a}[q_0Aq_n]\,[q_nAq_m]$, para cualesquier estados q_n y q_m de Q.

Esto representa 9 posibles combinaciones, para cada una, la mayoría de las cuales son inútiles, por lo que conviene que determinar los valores de n y m para los que se obtienen transiciones útiles, para ello se toman en cuenta a las producciones anteriormente obtenidas:

Dado que existe $[q_0Aq_1]$, entonces, para el primer caso tenemos que n = 1, y dado que también existe $[q_1Zq_2]$, entonces m = 2, quedando:

$$[q_0Zq_2] \rightarrow \mathbf{a}[q_0Aq_1]\,[q_1Zq_2]$$

Dado que existe $[q_0Aq_1]$, entonces, para el segundo caso tenemos que n = 1, y dado que también existe $[q_1Aq_1]$, entonces m = 1, quedando:

$$[q_0Aq_1] \rightarrow \mathbf{a}[q_0Aq_1]\,[q_1Aq_1]$$

Finalmente, recomendamos hacer un cambio de variables como por ejemplo, reemplazar: $S = [q_0Zq_2]$, $A = [q_0Aq_1]$, $B = [q_1Aq_1]$ y $C = [q_1Zq_2]$, resultando que nuestra gramática está formada por las siguientes producciones:

$$S \rightarrow \mathbf{a}AC$$
$$A \rightarrow \mathbf{b} \mid \mathbf{a}AB$$
$$B \rightarrow \mathbf{b}$$
$$C \rightarrow \varepsilon$$

Y reemplazando a las últimas dos producciones, la gramática equivalente queda:

$$S \rightarrow \mathbf{a}A$$
$$A \rightarrow \mathbf{b} \mid \mathbf{a}A\mathbf{b}$$

Por ejemplo, si aplicamos la gramática precedente para realizar la generación de la cadena *w* = **aabb**, tenemos la siguiente derivación:

$$S \Rightarrow \mathbf{a}A \Rightarrow \mathbf{aa}A\mathbf{b} \Rightarrow \mathbf{aabb}$$

Ejemplo 2

Obtener una **GIC** que genere el lenguaje L = **ab*a**, el cual es aceptado por el **APD**:

Q = { q_0, q_1, q_2 }, Σ = { **a**, **b** }, Γ = { A, Z }, F = { q_2 }, s = q_0 y δ dado por:

$\delta(q_0, \mathbf{a}, Z) = \{ (q_0, AZ) \}$ $\quad$ $\delta(q_0, \mathbf{b}, A) = \{ (q_0, AA) \}$

$\delta(q_0, \mathbf{a}, A) = \{ (q_1, \varepsilon) \}$ $\quad$ $\delta(q_1, \varepsilon, A) = \{ (q_1, \varepsilon) \}$

$\delta(q_1, \varepsilon, Z) = \{ (q_2, \varepsilon) \}$

Ahora ya podemos encontrar la **GIC** correspondiente; para las transiciones que desapilan tenemos:

$$[q_0Aq_1] \rightarrow \mathbf{a}$$

$$[q_1Aq_1] \rightarrow \varepsilon$$

$$[q_1Zq_2] \rightarrow \varepsilon$$

Y para las transiciones que apilan, escribiendo sólo las que son útiles tenemos:

$$[q_0Zq_2] \rightarrow \mathbf{a}[q_0Aq_1]\,[q_1Zq_2]$$

$$[q_0Aq_1] \rightarrow \mathbf{b}[q_0Aq_1]\,[q_1Aq_1]$$

Reemplazando $[q_0Zq_2]$ por S, que es nuestro símbolo inicial, $[q_0Aq_1]$ por A, $[q_1Aq_1]$ por B y $[q_1Zq_2]$ por C tenemos:

$$S \rightarrow \mathbf{a}AC$$

$$A \rightarrow \mathbf{b}AB \mid \mathbf{a}$$

$$B \rightarrow \varepsilon$$

$$C \rightarrow \varepsilon$$

Y eliminando las producciones épsilon, nos resulta la siguiente gramática regular:

$$S \rightarrow \mathbf{a}A$$

$$A \rightarrow \mathbf{b}A \mid \mathbf{a}$$

Usando esta gramática para la generación de la cadena *w* = **abba**, tenemos:

$$S \Rightarrow \mathbf{a}A \Rightarrow \mathbf{ab}A \Rightarrow \mathbf{abb}A \Rightarrow \mathbf{abba}$$

Resulta interesante observar que las gramáticas obtenidas por esta técnica se encuentran en la **FNG**, esto facilita enormemente la generación de las cadenas correspondientes al lenguaje en cuestión, como se puede apreciar en los dos ejemplos previos.

Preguntas

a) Sean M_1 y M_2 dos **APN**s que aceptan los lenguajes L_1 y L_2 respectivamente, ¿Cómo es posible construir un **APN** que acepte al Lenguaje $L_1 \cup L_2$?

b) Sean M_1 y M_2 dos **APN**s que aceptan los lenguajes L_1 y L_2 respectivamente, describa ¿Cómo es posible construir un **APN** que acepte al Lenguaje L_1L_2?

c) Sean M_1 un **APN** que acepta al lenguaje L_1, describa ¿Cómo es posible construir un **APN** que acepte al Lenguaje L_1*?

d) ¿Es posible encontrar un **APN** complemento para cualquier **APN** dado?

e) ¿Cómo diseñarías un autómata de dos pilas para el reconocimiento del siguiente lenguaje: L = { $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n \mid n > 0$ }?

f) ¿Es más poderosos un autómata de dos pilas que uno de una sola?

g) ¿Es posible diseñar un autómata que utilice una cola en vez de una pila para el reconocimiento del lenguaje L = { $ww \mid w \in \{ \mathbf{a}, \mathbf{b} \}^*$ }?

h) ¿Para que situaciones conviene diseñar autómatas de cola en vez de los de pila que hemos visto?

Ejercicios Capítulo 9

9.1 Construya los **APD** para aceptar a cada uno los siguientes lenguajes:

a) L = { $\mathbf{a}^n\mathbf{b}^m\mathbf{c}^n \mid n, m > 0$ }

b) L = { $\mathbf{a}^n\mathbf{b}^{2n} \mid n > 0$}

c) L = { $\mathbf{a}^n\mathbf{b}\mathbf{c}^n \mid n \geq 0$}

d) L = { $\mathbf{a}^n\mathbf{b}^m\mathbf{c}^{n+m} \mid n, m > 0$}

e) L = { $\mathbf{a}^{2n}\mathbf{b}^n \mid n > 0$}

9.2 Construya un **APD** condicionado para aceptar el lenguaje regular L = **ac***ba, pero utilizando la pila en lugar de hacer cambios de estado, de este modo todas las transiciones del **APD** se harán en un solo estado.

9.3 Construya los **APD**s condicionados que reconozcan a los siguientes lenguajes:

a) L = { $\mathbf{a}^n\mathbf{b}^m\mathbf{a}^n \mid n, m > 0$}

b) L = { $\mathbf{a}^n\mathbf{b}^{n+m}\mathbf{a}^m \mid n, m > 0$ }

c) L = { $\mathbf{a}^n\mathbf{b}^m\mathbf{a}^m\mathbf{b}^n \mid n, m > 0$ }

d) L = { $\mathbf{a}^n\mathbf{b}^{n+1} \mid n \geq 0$ }

9.4 Construya los **APN**s que permitan aceptar a cada uno los siguientes lenguajes:

a) L = { $\mathbf{a}^m\mathbf{b}^n \mid m, n \geq 0$ y $m \neq n$}

b) L = { $\mathbf{a}^n\mathbf{b}^m \mid 0 \leq m \leq n$ }

c) L = { $\mathbf{a}^n\mathbf{b}^m \mid 0 \leq n \leq m$ }

d) L = { $\mathbf{a}^m\mathbf{b}^n \mid 0 \leq m \leq n \leq 3m$}

e) L = { $w \in \{ \mathbf{a}, \mathbf{b} \}^* \mid w = w^R$ y $|w|$ es impar }

f) L = { $\mathbf{a}^n w \mid w \in \{ \mathbf{a}, \mathbf{b} \}^*$ y $n \geq |w|$ }

g) L = { $x\mathbf{a}y\mathbf{b} \mid x, y \in \{ \mathbf{a}, \mathbf{b} \}^*, |x| = |y|$ }

h) L = { $w \in \{ \mathbf{a}, \mathbf{b} \}^* \mid w \neq w^R$ }

9.5 Construya los **APN** condicionados que reconozcan a los siguientes lenguajes:

a) $L = \{ w \in \{ \mathbf{a}, \mathbf{b} \}^* \mid N_{\mathbf{a}}(w) = 2N_{\mathbf{b}}(w) \}$

b) $L = \{ w \in \{ \mathbf{a}, \mathbf{b} \}^* \mid N_{\mathbf{a}}(w) < N_{\mathbf{b}}(w) < 2N_{\mathbf{a}}(w) \}$

9.6 Identifique el Lenguaje que acepta el **APN** condicionado: $Q = \{ q_0, q_1, q_2 \}$, $s = q_0$, $F = \{q_2\}$, $\Sigma = \{ \mathbf{a}, \mathbf{b} \}$, $\Gamma = \{ A, B, Z \}$ y las transiciones siguientes:

$\Delta(q_0, \mathbf{a}, Z) = \{ (q_1, A), (q_2, \varepsilon) \}$ $\quad$ $\Delta(q_1, \mathbf{a}, B) = (q_2, \varepsilon)$

$\Delta(q_1, \mathbf{b}, A) = (q_1, B)$ $\quad$ $\Delta(q_1, \mathbf{b}, B) = (q_1, B)$

9.7 Describa el Lenguaje que acepta el **APD** condicionado: $Q = \{ q_0, q_1, q_2 \}$, $s = q_0$, $F = \{q_2\}$, $\Sigma = \{\mathbf{a}, \mathbf{b}\}$, $\Gamma = \{A, Z\}$ y las transiciones siguientes:

$\delta(q_0, \mathbf{a}, Z) = (q_0, AZ)$ $\quad$ $\delta(q_0, \mathbf{a}, A) = (q_0, AA)$

$\delta(q_0, \mathbf{b}, A) = (q_0, \varepsilon)$ $\quad$ $\delta(q_0, \mathbf{b}, Z) = (q_1, AZ)$

$\delta(q_1, \mathbf{b}, A) = (q_1, \varepsilon)$ $\quad$ $\delta(q_1, \varepsilon, Z) = (q_2, \varepsilon)$

9.8 Describa el proceso que realiza el **APD** del problema anterior, con cada una de las cadenas siguientes: w_1 = **abba**, w_2 = **abab** y w_3 = **abbb**.

9.9 Obtenga un **APN** que acepte el lenguaje generado por cada una de las siguientes **GIC**s:

a) $S \rightarrow \mathbf{a}AA$, $A \rightarrow \mathbf{a}A \mid \mathbf{b}S \mid \mathbf{c}$

b) $S \rightarrow \mathbf{ab}S \mid \mathbf{ac}S\mathbf{bb} \mid \mathbf{c}$

c) $S \rightarrow \mathbf{a}ABB \mid \mathbf{b}AA$, $A \rightarrow \mathbf{a}BB \mid \mathbf{b}A$, $B \rightarrow \mathbf{b}BB \mid \mathbf{a}$

9.10 Construya un **APD** equivalente para cada uno de los incisos del problema anterior, utilizando la técnica de la anticipación.

9.11 Obtenga una **GIC** que genera a los lenguajes aceptados por los **AP**s de los ejercicios 9.6 y 9.7, respectivamente.

9.12 Obtenga una **GIC** que genere el lenguaje aceptado por el **APD** definido por: $Q = \{ q_0, q_1, q_2 \}$, $\Sigma = \{ \mathbf{a}, \mathbf{b} \}$, $\Gamma = \{ A, Z \}$, $s = q_0$, $F = \{ q_2 \}$ y δ dado por:

$\delta(q_0, \mathbf{a}, Z) = (q_0, AZ)$ $\quad$ $\delta(q_0, \mathbf{a}, A) = (q_0, A)$

$\delta(q_0, \mathbf{b}, A) = (q_1, \varepsilon)$ $\quad$ $\delta(q_1, \varepsilon, Z) = (q_2, \varepsilon)$

9.13 Obtenga una **GIC** que genere el lenguaje aceptado por el **APN** definido por: $Q = \{ q_0, q_1, q_2 \}$, $\Sigma = \{ \mathbf{a}, \mathbf{b} \}$, $\Gamma = \{ X, Z \}$, $s = q_0$, $F = \{ q_2 \}$ y Δ dado por:

$\Delta(q_0, \mathbf{a}, Z) = (q_0, XZ)$ $\quad$ $\Delta(q_0, \mathbf{b}, X) = (q_0, XX)$

$\Delta(q_0, \mathbf{a}, X) = (q_1, X)$ $\quad$ $\Delta(q_1, \mathbf{b}, X) = (q_1, \varepsilon)$

$\Delta(q_1, \mathbf{a}, Z) = (q_0, Z)$ $\quad$ $\Delta(q_0, \varepsilon, Z) = (q_2, \varepsilon)$

9.14 Obtenga una **GIC** que genere el lenguaje aceptado por el **APN** definido por: Q = { q_0, q_1, q_2 }, s = q_0, F = { q_2 }, Σ = { **a**, **b** }, Γ = { A, Z} y Δ dado por:

$\Delta(q_0, \mathbf{a}, Z) = \{ (q_0, AZ) \}$ $\Delta(q_0, \mathbf{a}, A) = \{ (q_0, AA) \}$

$\Delta(q_0, \mathbf{b}, A) = \{ (q_1, \varepsilon) \}$ $\Delta(q_0, \varepsilon, Z) = \{ (q_2, \varepsilon) \}$

$\Delta(q_1, \mathbf{a}, A) = \{ (q_1, \varepsilon) \}$ $\Delta(q_1, \mathbf{b}, Z) = \{ (q_0, Z) \}$

9.15 Obtenga una **GIC** que genere el lenguaje aceptado por el **APD** definido por: Q = { q_0, q_1, q_2 }, s = q_0, F = { q_2 }, Σ = { **a**, **b** }, Γ = { A, Z} y δ dado por:

$\delta(q_0, \mathbf{a}, Z) = \{ (q_0, AZ) \}$ $\delta(q_0, \mathbf{a}, A) = \{ (q_0, AA) \}$

$\delta(q_0, \mathbf{b}, Z) = \{ (q_1, AZ) \}$ $\delta(q_0, \mathbf{b}, A) = \{ (q_0, \varepsilon) \}$

$\delta(q_1, \mathbf{b}, A) = \{ (q_1, \varepsilon) \}$ $\delta(q_1, \varepsilon, Z) = \{ (q_2, \varepsilon) \}$

9.16 Obtenga una **GIC** que genere el lenguaje aceptado por el **APN** definido por: Q = { q_0, q_1, q_2, q_3}, Σ = { **a**, **b** }, Γ = { A, B, Z}, s = q_0, F = { q_3 } y Δ dado por:

$\Delta(q_0, \mathbf{a}, Z) = (q_1, AZ)$ $\Delta(q_1, \mathbf{a}, A) = \{ (q_1, A), (q_2, \varepsilon) \}$

$\Delta(q_0, \mathbf{b}, Z) = (q_1, BZ)$ $\Delta(q_1, \mathbf{b}, B) = \{ (q_1, B), (q_2, \varepsilon) \}$

$\Delta(q_1, \mathbf{a}, B) = (q_1, B)$ $\Delta(q_1, \mathbf{b}, A) = (q_1, A)$

$\Delta(q_2, \varepsilon, Z) = (q_3, \varepsilon)$

9.17 Obtenga una **GIC** que genere el lenguaje aceptado por el **APN** definido por: Q = { q_0, q_1, q_2 }, s = q_0, F = { q_2 }, Σ = { **a**, **b** }, Γ = { A, Z} y Δ dado por:

$\Delta(q_0, \mathbf{a}, Z) = \{ (q_0, AZ) \}$ $\Delta(q_0, \mathbf{a}, A) = \{ (q_0, AA) \}$

$\Delta(q_0, \mathbf{b}, Z) = \{ (q_1, Z) \}$ $\Delta(q_1, \mathbf{b}, Z) = \{ (q_1, Z) \}$

$\Delta(q_0, \mathbf{b}, A) = \{ (q_0, \varepsilon) \}$ $\Delta(q_1, \varepsilon, Z) = \{ (q_2, \varepsilon) \}$

9.18 Obtenga una **GIC** que genere el lenguaje aceptado por el **APN** definido por: Q = { q_0, q_1, q_2 }, s = q_0, F = { q_2 }, Σ = { **0**, **1** }, Γ = { A, Z} y Δ dado por:

$\Delta(q_0, \mathbf{0}, Z) = \{ (q_0, AZ) \}$ $\Delta(q_0, \mathbf{0}, A) = \{ (q_0, A) \}$

$\Delta(q_0, \mathbf{1}, Z) = \{ (q_1, Z) \}$ $\Delta(q_1, \mathbf{0}, Z) = \{ (q_1, Z) \}$

$\Delta(q_0, \mathbf{1}, A) = \{ (q_0, \varepsilon) \}$ $\Delta(q_1, \varepsilon, Z) = \{ (q_2, \varepsilon) \}$

9.19 Obtenga una **GIC** que genere el lenguaje aceptado por el **APN** definido por: Q = { q_0, q_1, q_2 }, s = q_0, F = { q_2 }, Σ = { **a**, **b** }, Γ = { A, Z} y Δ dado por:

$\Delta(q_0, \mathbf{a}, Z) = \{ (q_0, AZ) \}$ $\Delta(q_0, \mathbf{a}, A) = \{ (q_1, \varepsilon) \}$

$\Delta(q_0, \mathbf{b}, A) = \{ (q_0, AA) \}$ $\Delta(q_1, \mathbf{a}, A) = \{ (q_1, \varepsilon) \}$

$\Delta(q_1, \mathbf{b}, Z) = \{ (q_0, Z) \}$ $\Delta(q_0, \varepsilon, Z) = \{ (q_2, \varepsilon) \}$

9.20 Obtenga una **GIC** que genere el lenguaje aceptado por el **APD** definido por: Q = { q_0, q_1, q_2 }, Σ = { **a**, **b**, **c** }, Γ = { X, Z}, s = q_0, F = { q_2 } y δ dado por:

$\delta(q_0, \mathbf{a}, Z) = (q_0, XZ)$ $\quad$ $\delta(q_0, \mathbf{a}, X) = (q_0, XX)$

$\delta(q_0, \mathbf{b}, Z) = (q_0, XZ)$ $\quad$ $\delta(q_0, \mathbf{b}, X) = (q_0, XX)$

$\delta(q_0, \mathbf{c}, Z) = (q_1, Z)$ $\quad$ $\delta(q_0, \mathbf{c}, X) = (q_1, X)$

$\delta(q_1, \mathbf{a}, X) = (q_1, \varepsilon)$ $\quad$ $\delta(q_1, \mathbf{b}, X) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, Z) = (q_2, \varepsilon)$

Capítulo

10

Máquinas de Turing

Se introduce la noción de Cinta, Se define el concepto de Máquina de Turing, se analiza su aplicación en el cómputo de funciones y su utilización en la decisión de cadenas. Se describen algunas de sus variantes. Se estudian las Máquinas Simples y Compuestas.

La Noción de Cinta

Cuando empleamos un Autómata Finito para el reconocimiento de un lenguaje regular, se debe proporcionar una cadena para que sea analizada; imaginemos ahora que esta cadena de entrada se encuentra escrita sobre una cinta infinita que está subdividida en celdas unitarias, cada una de las cuales es capaz de contener un solo símbolo y que existe un apuntador que inicialmente se encuentra señalando a la celda que contiene al primer símbolo de la cadena.

Cada transición realizada en el **AF** provocará un cambio de estado y hará que el apuntador avance una celda hacia la derecha; tal como se ilustra gráficamente en el ejemplo mostrado en la figura 10.1:

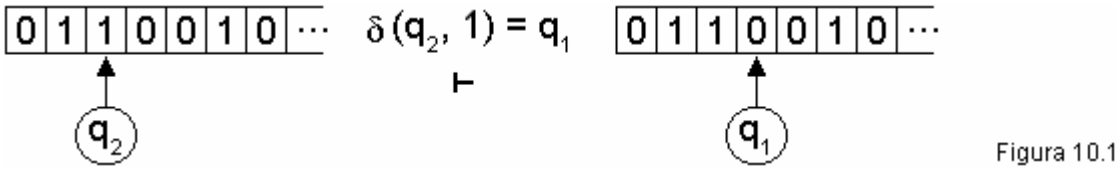


Figura 10.1

Sin embargo, cuando se trata de una transición épsilon, entonces solamente se provoca un cambio de estado, pero el apuntador permanece en la celda que aún no ha sido *leída*, tal como se muestra en la figura 10.2:

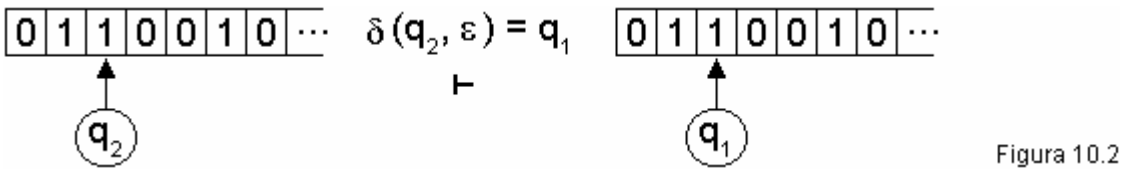


Figura 10.2

Finalmente, se considera que la cadena ha sido agotada cuando el apuntador llega a una celda que está vacía. Este hecho se suele representar por medio de un símbolo

especial como por ejemplo: #, el cual no pertenece al alfabeto $\sum$ y en ese momento el autómata termina su proceso, y es cuando se considera el estado en que se finaliza, para determinar la aceptación o rechazo de la cadena propuesta, tal como se ejemplifica en la figura 10.3.

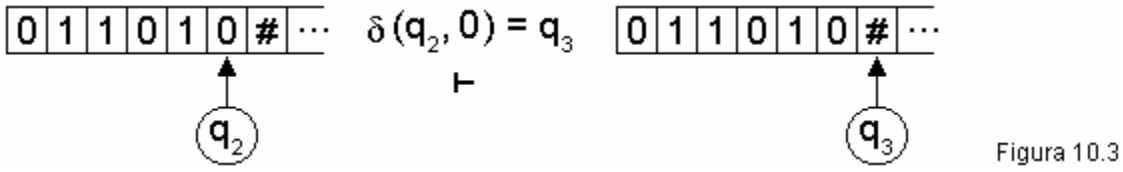


Figura 10.3

Autómata Finito Determinista Bidireccional

Un Autómata Finito Determinista Bidireccional (**AFDB**) es aquel que permite que el apuntador se pueda desplazar tanto hacia la derecha como hacia la izquierda. Se ha demostrado que esta capacidad de movimiento no le agrega ningún poder a este tipo de **AF** respecto de los otros, y en cambio, se pueden presentar situaciones en las que se cicle y nunca alcance el fin de la cadena.

Ejemplo

Sea el **AFDB** definido por: Q = { q_0, q_1, q_2 }, $\sum$ = { **0**, **1** }, s = q_0, F = { q_0, q_1 } y la función δ que contiene a las transiciones siguientes:

$\delta(q_0, \mathbf{0}) = (q_0, R)$ $\quad$ $\delta(q_0, \mathbf{1}) = (q_1, R)$

$\delta(q_1, \mathbf{0}) = (q_1, R)$ $\quad$ $\delta(q_1, \mathbf{1}) = (q_2, L)$

$\delta(q_2, \mathbf{0}) = (q_0, R)$ $\quad$ $\delta(q_2, \mathbf{1}) = (q_2, L)$

Donde se denota con la letra **R** a un desplazamiento hacia la derecha (Right) y con la letra **L** un desplazamiento hacia la Izquierda (Left).

Ahora observemos el proceso que sigue para la aceptación de *w* = **1010010**.

$(q_0,\underline{\mathbf{1}}\mathbf{010010}) \vdash (q_1,\mathbf{1}\underline{\mathbf{0}}\mathbf{10010}) \vdash (q_1,\mathbf{10}\underline{\mathbf{1}}\mathbf{0010}) \vdash (q_2,\mathbf{1}\underline{\mathbf{0}}\mathbf{10010}) \vdash (q_0,\mathbf{10}\underline{\mathbf{1}}\mathbf{0010}) \vdash$
$(q_1,\mathbf{101}\underline{\mathbf{0}}\mathbf{010}) \vdash (q_1,\mathbf{1010}\underline{\mathbf{0}}\mathbf{10}) \vdash (q_1,\mathbf{10100}\underline{\mathbf{1}}\mathbf{0}) \vdash (q_2,\mathbf{1010}\underline{\mathbf{0}}\mathbf{10}) \vdash (q_0,\mathbf{10100}\underline{\mathbf{1}}\mathbf{0}) \vdash$
$(q_1,\mathbf{101001}\underline{\mathbf{0}}) \vdash (q_1,\mathbf{1010010}\underline{\#})$

Este autómata acepta las cadenas que no contengan la subcadena **11**, cada vez que detecta un segundo **1**, regresa para comprobar que el símbolo anterior es **0**, sin embargo, en caso de haber dos **1**s consecutivos, se cicla indefinidamente, como se muestra en la siguiente secuencia:
$(q_0, \underline{\mathbf{0}}\mathbf{11}) \vdash (q_0, \mathbf{0}\underline{\mathbf{1}}\mathbf{1}) \vdash (q_1, \mathbf{01}\underline{\mathbf{1}}) \vdash (q_2, \mathbf{0}\underline{\mathbf{1}}\mathbf{1}) \vdash (q_2, \underline{\mathbf{0}}\mathbf{11}) \vdash (q_0, \mathbf{0}\underline{\mathbf{1}}\mathbf{1}) \vdash (q_1, \mathbf{01}\underline{\mathbf{1}}) \vdash \ldots$

En la figura que se muestra a continuación, tenemos el **AFN** equivalente a este **AFDB** y que acepta al lenguaje: L = ($\varepsilon \cup$ **1**)(**0** $\cup$ **01**)*.

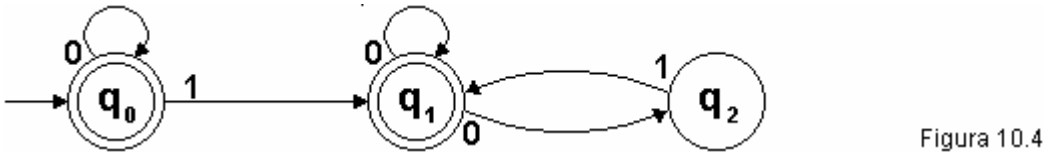


Figura 10.4

La Máquina de Turing[2]

Conceptualmente una Máquina de Turing (**MT**) puede parecer muy semejante al modelo de **AFDB** presentado previamente, pero la pequeña gran diferencia es que este dispositivo tiene la posibilidad de escribir sobre la cinta. Esta capacidad le permite a la **MT** devolver una respuesta, por medio de una cadena de salida.

Formalmente definimos una Máquina de Turing como M = (Q, $\sum$, Γ, s, #, F, δ), en donde Q es el conjunto de estados, $\sum$ es el alfabeto de entrada, Γ es el alfabeto de la Cinta ($\sum \subset \Gamma$), s es el estado inicial, # es el símbolo de vacío (# $\in \Gamma$, pero # $\notin \sum$), F es el conjunto de estados finales y δ es la función de Transición determinista.

En este caso, cada transición provoca tres respuestas, primero un posible cambio de estado, segundo, la escritura de un símbolo sobre la celda actual y, tercero, el desplazamiento del apuntador (también llamado cabezal de lectura-escritura o **CLE**) hacia la celda de la derecha o la celda de la izquierda. Las **MT**s pueden ser no deterministas, pero como esta característica no les da mayor poder, nos limitaremos a estudiar solamente las **MTD**.

Ejemplo 1

Consideremos la **MT** definida por los siguientes elementos: Q = { q_0, q_1, q_2}, $\sum$= {**a**, **b** }, s = q_0, Γ = { **a**, **b**, #}, F = { q_2 } y las transiciones:

$\delta(q_0, \mathbf{a}) = (q_0, \mathbf{a}, R)$
$\delta(q_0, \mathbf{b}) = (q_0, \mathbf{a}, R)$
$\delta(q_0, \#) = (q_1, \#, L)$
$\delta(q_1, \mathbf{a}) = (q_1, \mathbf{a}, L)$
$\delta(q_1, \#) = (q_2, \#, R)$

[2] Alan Mathison Turing (1912-1954), matemático británico y pionero en la teoría de la computación. Nació en Londres y estudió en las universidades de Cambridge y Princeton. En 1936, mientras era todavía un estudiante, publicó un ensayo titulado *On Computable Numbers* (*Sobre números calculables*), con el que contribuyó a la lógica matemática al introducir el concepto teórico de un dispositivo de cálculo que hoy se conoce como la máquina de Turing. El concepto de esta máquina, que podría efectuar teóricamente cualquier cálculo matemático, fue importante en el desarrollo de las computadoras digitales.

Esta Máquina reemplaza una **a** por otra **a** y se desplaza a la derecha, si encuentra una **b** la reemplazará también por una **a** y desplaza la **CLE** a la derecha, cuando encuentre una celda en blanco cambiará de estado y se desplaza a la izquierda, en el estado q_1 recorrerá la cinta a la izquierda, hasta encontrar la celda vacía que se encuentra antes de la cadena, donde cambiará de estado, se moverá a la celda inicial y se parará. La transición $\delta(q_1, \mathbf{b})$ no se define porque jamás se utilizará.

El funcionamiento de esta **MT** para la cadena *w* = **abba**, lo podemos representar por medio de las descripciones instantáneas siguientes (las celdas anteriores y posteriores a la cadena se asumen vacías y solo se indican cuando es necesario):

$(q_0,\underline{\mathbf{a}}\mathbf{bba}) \vdash (q_0,\mathbf{a}\underline{\mathbf{b}}\mathbf{ba}) \vdash (q_0,\mathbf{aa}\underline{\mathbf{b}}\mathbf{a}) \vdash (q_0,\mathbf{aaa}\underline{\mathbf{a}}) \vdash (q_0,\mathbf{aaa}\underline{\#}) \vdash (q_1,\mathbf{aaa}\underline{\mathbf{a}}) \vdash (q_1,\mathbf{aa}\underline{\mathbf{a}}\mathbf{a}) \vdash (q_1,\mathbf{a}\underline{\mathbf{a}}\mathbf{aa}) \vdash (q_1,\underline{\mathbf{a}}\mathbf{aaa}) \vdash (q_1,\underline{\#}\mathbf{aaa}) \vdash (q_2,\underline{\mathbf{a}}\mathbf{aaa})$

O bien, otra notación más compacta sería poniendo el estado antes del símbolo correspondiente a la celda actual:

$q_0\mathbf{abba} \vdash \mathbf{a}q_0\mathbf{bba} \vdash \mathbf{aa}q_0\mathbf{ba} \vdash \mathbf{aaa}q_0\mathbf{a} \vdash \mathbf{aaa}q_0\# \vdash \mathbf{aaa}q_1\mathbf{a} \vdash \mathbf{aa}q_1\mathbf{aa} \vdash \mathbf{a}q_1\mathbf{aaa} \vdash q_1\mathbf{aaaa} \vdash q_1\#\mathbf{aaa} \vdash q_2\mathbf{aaa}$

Ejemplo 2

Consideremos la Máquina de Turing siguiente: Q = $\{q_0, q_1, q_2\}$, Σ = {**0**, **1**}, Γ = {**0**, **1**, **#**}, s = q_0, F = { q_2 } y las transiciones:

$\delta(q_0, \mathbf{0}) = (q_0, \mathbf{1}, R)$	$\delta(q_0, \mathbf{1}) = (q_0, \mathbf{0}, R)$
$\delta(q_0, \#) = (q_1, \#, L)$	$\delta(q_1, \mathbf{0}) = (q_1, \mathbf{0}, L)$
$\delta(q_1, \mathbf{1}) = (q_1, \mathbf{1}, L)$	$\delta(q_1, \#) = (q_2, \#, R)$

Esta Máquina complementa las cadenas sobre el alfabeto Σ hasta encontrar una celda en blanco, entonces regresará hasta el primer símbolo no blanco y parará.

Como en el primer ejemplo, se asume que la **CLE** se ubica sobre el primer símbolo de cadena de entrada. Por ejemplo, para la cadena w = **0110**, tenemos:

$(q_0, \underline{\mathbf{0}}\mathbf{110}) \vdash (q_0, \mathbf{1}\underline{\mathbf{1}}\mathbf{10}) \vdash (q_0, \mathbf{10}\underline{\mathbf{1}}\mathbf{0}) \vdash (q_0, \mathbf{100}\underline{\mathbf{0}}) \vdash (q_0, \mathbf{1001}\underline{\#}) \vdash (q_1, \mathbf{100}\underline{\mathbf{1}}) \vdash (q_1, \mathbf{10}\underline{\mathbf{0}}\mathbf{1})$

$\vdash (q_1, \mathbf{1}\underline{\mathbf{0}}\mathbf{01}) \vdash (q_1, \underline{\mathbf{1}}\mathbf{001}) \vdash (q_1, \underline{\#}\mathbf{1001}) \vdash (q_2, \underline{\mathbf{1}}\mathbf{001})$

Este proceso se puede resumir de la siguiente forma: $(q_0, \underline{\mathbf{0}}\mathbf{110}) \vdash^* (q_2, \underline{\mathbf{1}}\mathbf{001})$. (Donde el asterisco denota que se requieren de varias transiciones para llegar a la descripción de la derecha)

Funciones Turing-Computables

Puesto que se una **MT** puede leer y escribir sobre la cinta, podemos observar que la cadena de entrada también se transforma en una cadena de salida, formada por los símbolos que quedan en la cinta al terminar la ejecución de la misma, para ello, es conveniente construir la **MT** de tal forma que finalice con la **CLE** ubicada en la misma posición donde inició, tal como se hizo en los ejemplos precedentes. La posición inicial puede ser elegida arbitrariamente, en estos ejemplos hemos elegido la celda que contiene al primer símbolo de la cadena, pero puede ser la celda vacía precedente o la que está después del final de la cadena, lo importante es que la celda inicial elegida sea siempre la misma, para que la **MT** trabaje correctamente.

En estos ejemplos, las **MT**s se han construido de tal forma que siempre paren, que siempre alcancen al estado final. A la secuencia de movimientos que nos conducen a una configuración de parada se le llama *Computación*. En forma compacta, expresamos una computación como: $(q_0, w) \vdash^* (q_f, u)$

Si existe una función ***F*** sobre la cadena *w*, tal que ***F***(*w*) = *u*, y si existe una Máquina de Turing para la cual se puede efectuar la computación $(q_0, w) \vdash^* (q_f, u)$, donde $s = q_0$ y $q_f \in F$, para cualquier cadena *w* en el dominio de ***F***, entonces se dice que la función de cadena ***F*** es *Turing-computable*.

Ejemplo 1

Construir una Máquina de Turing para realizar la suma de dos números en base uno, es decir, el número *n* se representa como una cadena de *n* unos: $\mathbf{1}^n$. El proceso a realizar para sumar *n* + *m* consiste en computar: $(q_0, \mathbf{1}^n\mathbf{+1}^m) \vdash^* (q_4, \mathbf{1}^{n+m})$:

$\delta(q_0, \mathbf{1}) = (q_0, \mathbf{1}, R)$ $\quad$ $\delta(q_0, \mathbf{+}) = (q_1, \mathbf{1}, R)$

$\delta(q_1, \mathbf{1}) = (q_1, \mathbf{1}, R)$ $\quad$ $\delta(q_1, \#) = (q_2, \#, L)$

$\delta(q_2, \mathbf{1}) = (q_3, \#, L)$ $\quad$ $\delta(q_3, \mathbf{1}) = (q_3, \mathbf{1}, L)$

$\delta(q_3, \#) = (q_4, \#, R)$

Por ejemplo para sumar 3 + 2 tenemos la siguiente *computación*:

$(q_0, \underline{\mathbf{1}}\mathbf{11+11}) \vdash (q_0, \mathbf{1}\underline{\mathbf{1}}\mathbf{1+11}) \vdash (q_0, \mathbf{11}\underline{\mathbf{1}}\mathbf{+11}) \vdash (q_0, \mathbf{111}\underline{\mathbf{+}}\mathbf{11}) \vdash (q_1, \mathbf{1111}\underline{\mathbf{1}}) \vdash (q_1, \mathbf{11111}\underline{\mathbf{\#}}) \vdash (q_2, \mathbf{11111}\underline{\mathbf{1}}) \vdash (q_3, \mathbf{1111}\underline{\mathbf{1}}\mathbf{\#}) \vdash (q_3, \mathbf{111}\underline{\mathbf{1}}\mathbf{1}) \vdash (q_3, \mathbf{11}\underline{\mathbf{1}}\mathbf{11}) \vdash (q_3, \mathbf{1}\underline{\mathbf{1}}\mathbf{111}) \vdash (q_3, \underline{\mathbf{1}}\mathbf{1111}) \vdash (q_3, \underline{\mathbf{\#}}\mathbf{11111}) \vdash (q_4, \underline{\mathbf{1}}\mathbf{1111})$

Ejemplo 2

Para construir una Máquina de Turing para realizar la resta de dos números, se debe realizar un proceso más elaborado que el del ejemplo anterior, pues debemos ir eliminando un **1** del sustraendo por cada **1** del minuendo y repetir este proceso tantas veces como sea necesario, hasta agotar la cantidad de unos en alguna de las dos partes, considerando la posibilidad de que la diferencia sea negativa. Este proceso se resume entonces en: $(q_0, \mathbf{1}^n\mathbf{-}\mathbf{1}^m) \vdash^* (q_7, \mathbf{1}^{n-m})$, si el resultado es positivo o cero y en: $(q_0, \mathbf{1}^n\mathbf{-}\mathbf{1}^m) \vdash^* (q_7, \mathbf{-}\mathbf{1}^{m-n})$ para el caso contrario.

$\delta(q_0, \mathbf{1}) = (q_0, \mathbf{1}, R)$ $\quad$ $\delta(q_0, \mathbf{-}) = (q_1, \mathbf{-}, R)$ $\quad$ $\delta(q_1, \mathbf{1}) = (q_1, \mathbf{1}, R)$

$\delta(q_1, \#) = (q_2, \#, L)$ $\quad$ $\delta(q_2, \mathbf{1}) = (q_3, \#, L)$ $\quad$ $\delta(q_3, \mathbf{1}) = (q_3, \mathbf{1}, L)$

$\delta(q_3, \mathbf{-}) = (q_3, \mathbf{-}, L)$ $\quad$ $\delta(q_3, \#) = (q_4, \#, R)$ $\quad$ $\delta(q_4, \mathbf{1}) = (q_0, \#, R)$

$\delta(q_4, \mathbf{-}) = (q_5, \mathbf{1}, L)$ $\quad$ $\delta(q_5, \#) = (q_6, \mathbf{-}, L)$ $\quad$ $\delta(q_6, \#) = (q_7, \#, R)$

$\delta(q_2, \mathbf{-}) = (q_6, \#, L)$ $\quad$ $\delta(q_6, \mathbf{1}) = (q_6, \mathbf{1}, L)$

Por ejemplo, para realizar la resta de 3 – 2, iniciamos con **1**11–11 y tenemos que realizar la siguiente secuencia, hasta obtener el resultado de **1**:

$(q_0, \mathbf{1}11\mathbf{-}11) \vdash (q_0, 1\mathbf{1}1\mathbf{-}11) \vdash (q_0, 11\mathbf{1}\mathbf{-}11) \vdash (q_0, 111\mathbf{-}\mathbf{1}1) \vdash (q_1, 111\mathbf{-}1\mathbf{1}) \vdash (q_1, 111\mathbf{-}11\#) \vdash (q_2, 111\mathbf{-}1\mathbf{1}\#) \vdash (q_3, 111\mathbf{-}\mathbf{1}\#\#) \vdash (q_3, 111\mathbf{-}\mathbf{1}1) \vdash (q_3, 11\mathbf{1}\mathbf{-}1) \vdash (q_3, 1\mathbf{1}1\mathbf{-}1) \vdash (q_3, \mathbf{1}11\mathbf{-}1) \vdash (q_3, \#111\mathbf{-}1) \vdash (q_4, \#\mathbf{1}11\mathbf{-}1) \vdash (q_0, \#\#\mathbf{1}1\mathbf{-}1) \vdash (q_0, 1\mathbf{1}\mathbf{-}1) \vdash (q_0, 11\mathbf{-}\mathbf{1}) \vdash (q_1, 11\mathbf{-}1\mathbf{1}) \vdash (q_1, 11\mathbf{-}1\#) \vdash (q_2, 11\mathbf{-}\mathbf{1}\#) \vdash (q_3, 11\mathbf{-}\#\#) \vdash (q_3, 1\mathbf{1}\mathbf{-}) \vdash (q_3, \mathbf{1}1\mathbf{-}) \vdash (q_3, \#11\mathbf{-}) \vdash (q_4, \#\mathbf{1}1\mathbf{-}) \vdash (q_0, \#\#\mathbf{1}\mathbf{-}) \vdash (q_0, 1\mathbf{-}) \vdash (q_1, 1\mathbf{-}\#) \vdash (q_2, 1\mathbf{-}\#) \vdash (q_6, \mathbf{1}\#\#) \vdash (q_6, \#1) \vdash (q_7, \#\mathbf{1})$

Ahora, para restar 2 – 3, la secuencia es como sigue:

$(q_0, \mathbf{1}1\mathbf{-}111) \vdash (q_0, 1\mathbf{1}\mathbf{-}111) \vdash (q_0, 11\mathbf{-}111) \vdash (q_1, 11\mathbf{-}\mathbf{1}11) \vdash (q_1, 11\mathbf{-}1\mathbf{1}1) \vdash (q_1, 11\mathbf{-}11\mathbf{1}) \vdash (q_1, 11\mathbf{-}111\#) \vdash (q_2, 11\mathbf{-}11\mathbf{1}\#) \vdash (q_3, 11\mathbf{-}1\mathbf{1}\#\#) \vdash (q_3, 11\mathbf{-}\mathbf{1}1) \vdash (q_3, 11\mathbf{-}11) \vdash (q_3, 1\mathbf{1}\mathbf{-}11) \vdash (q_3, \mathbf{1}1\mathbf{-}11) \vdash (q_3, \#11\mathbf{-}11) \vdash (q_4, \#\mathbf{1}1\mathbf{-}11) \vdash (q_0, \#\#\mathbf{1}\mathbf{-}11) \vdash (q_0, 1\mathbf{-}11) \vdash (q_1, 1\mathbf{-}\mathbf{1}1) \vdash (q_1, 1\mathbf{-}1\mathbf{1}) \vdash (q_1, 1\mathbf{-}11\#) \vdash (q_2, 1\mathbf{-}1\mathbf{1}\#) \vdash (q_3, 1\mathbf{-}\mathbf{1}\#\#) \vdash (q_3, 1\mathbf{-}\mathbf{1}) \vdash (q_3, \mathbf{1}\mathbf{-}1) \vdash (q_3, \#1\mathbf{-}1) \vdash (q_4, \#\mathbf{1}\mathbf{-}1) \vdash (q_0, \#\#\mathbf{-}1) \vdash (q_1, \mathbf{-}\mathbf{1}\#) \vdash (q_1, \mathbf{-}1\#) \vdash (q_2, \mathbf{-}\mathbf{1}\#) \vdash (q_3, \mathbf{-}\#\#) \vdash (q_3, \#\mathbf{-}) \vdash (q_4, \#\mathbf{-}) \vdash (q_5, \#\mathbf{1}) \vdash (q_6, \#\mathbf{-}1) \vdash (q_7, \#\mathbf{-}1)$

Finalmente, para restar 1 – 1, se tiene la siguiente secuencia:

$(q_0, \underline{\mathbf{1}}\mathbf{-1}) \vdash (q_0, \mathbf{1}\underline{\mathbf{-}}\mathbf{1}) \vdash (q_1, \mathbf{1-}\underline{\mathbf{1}}) \vdash (q_1, \mathbf{1-1}\underline{\#}) \vdash (q_2, \mathbf{1-}\underline{\mathbf{1}}\#) \vdash (q_3, \mathbf{1}\underline{\mathbf{-}}\#\#) \vdash (q_3, \underline{\mathbf{1}}\mathbf{-}) \vdash$

$(q_3, \underline{\#}\mathbf{1-}) \vdash (q_4, \#\underline{\mathbf{1}}\mathbf{-}) \vdash (q_0, \#\#\underline{\mathbf{-}}) \vdash (q_1, \mathbf{-}\underline{\#}) \vdash (q_2, \underline{\mathbf{-}}\#) \vdash (q_6, \underline{\#}\#) \vdash (q_7, \#\underline{\#})$

Ejemplo 3

Constrúyase una Máquina de Turing a la que se le da una cadena no vacía de unos y ceros para que devuelva una cadena con los símbolos ordenados, primero los ceros y a continuación los unos, como se ejemplifica a continuación:

$$(q_0, \underline{\mathbf{1}}\mathbf{00101}) \vdash^* (q_5, \underline{\mathbf{0}}\mathbf{00111})$$

Las transiciones necesarias son:

$\delta(q_0, \mathbf{0}) = (q_0, \mathbf{0}, R)$ $\quad$ $\delta(q_0, \mathbf{1}) = (q_1, \mathbf{1}, R)$ $\quad$ $\delta(q_1, \mathbf{1}) = (q_1, \mathbf{1}, R)$

$\delta(q_1, \mathbf{0}) = (q_2, \mathbf{1}, L)$ $\quad$ $\delta(q_2, \mathbf{1}) = (q_2, \mathbf{1}, L)$ $\quad$ $\delta(q_2, \mathbf{0}) = (q_3, \mathbf{0}, R)$

$\delta(q_2, \#) = (q_3, \#, R)$ $\quad$ $\delta(q_3, \mathbf{1}) = (q_1, \mathbf{0}, R)$ $\quad$ $\delta(q_1, \#) = (q_4, \#, L)$

$\delta(q_0, \#) = (q_4, \#, L)$ $\quad$ $\delta(q_4, \mathbf{0}) = (q_4, \mathbf{0}, L)$ $\quad$ $\delta(q_4, \mathbf{1}) = (q_4, \mathbf{1}, L)$

$\delta(q_4, \#) = (q_5, \#, R)$

Reconocimiento de Lenguajes con Máquinas de Turing

Otra aplicación específica de las **MT** es la determinación de que una cadena dada pertenezca o no a cierto lenguaje estipulado. A esta propiedad se le llama *Decidibilidad*, ya que la **MT** puede *decidir* o *reconocer* si la cadena pertenece o no al lenguaje dado. La **MT** nos hace saber que la cadena pertenece al referido lenguaje cuando llega a un tipo de estado final llamado estado de *Aceptación*, y en caso contrario, nos indica que la cadena no pertenece a dicho lenguaje cuando llega a otro estado final, llamado en este caso, estado de *Rechazo*.

Sea M una **MT** cualquiera, entonces decimos que el lenguaje aceptado por M es:

$$L(M) = \{ w \in \Sigma^* \mid (q_0, w) \vdash^* (q_A, u) \text{ para } q_A \in F\}$$

Ejemplo 1

Construir una Máquina de Turing para decidir el Lenguaje Regular $\mathbf{a}^*\mathbf{b}^+$.

Sea $Q = \{ q_0, q_1, q_A, q_R \}$, $\Sigma = \{ \mathbf{a}, \mathbf{b} \}$, $\Gamma = \{ \mathbf{a}, \mathbf{b}, \# \}$, $s = q_0$, $F = \{ q_A \}$, $R = \{ q_R \}$ y las transiciones:

$\delta(q_0, \mathbf{a}) = (q_0, \mathbf{a}, R)$ $\delta(q_0, \mathbf{b}) = (q_1, \mathbf{b}, R)$

$\delta(q_1, \mathbf{b}) = (q_1, \mathbf{b}, R)$ $\delta(q_1, \#) = (q_A, \#, R)$

Las siguientes transiciones se utilizan en caso de rechazar la cadena:

$\delta(q_0, \#) = (q_R, \#, R)$ $\delta(q_1, \mathbf{a}) = (q_R, \mathbf{a}, R)$

En la Figura 10.5 se compara esta **MT** obtenida con el **AFD** equivalente, ya que se encontrará una gran similitud entre ambos, salvo las transiciones para la # y el hecho de que la **MT** no requiere de *agotar* la cadena para rechazarla.

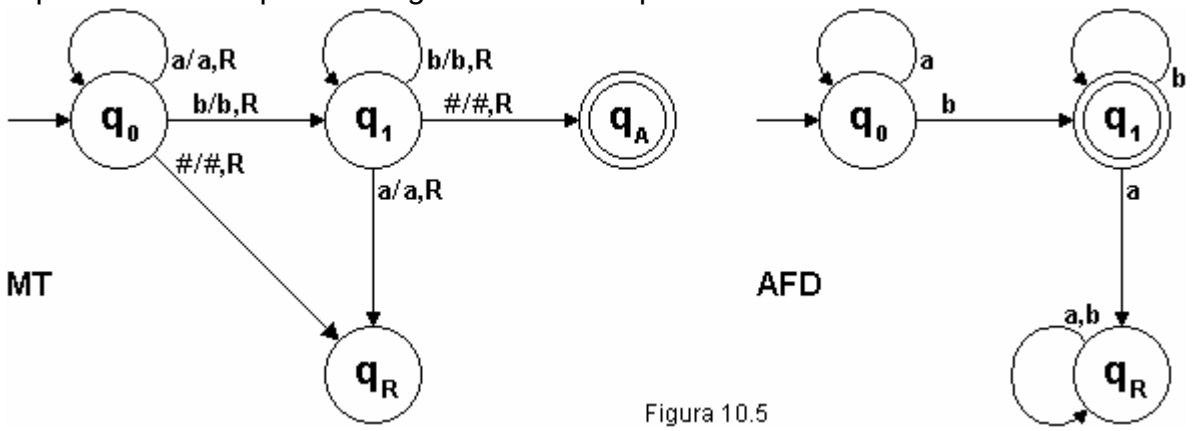


Figura 10.5

Ejemplo 2

El lenguaje dado por la expresión regular $L = ((\mathbf{a} \cup \mathbf{b})^2)^*$, está formado por todas las cadenas de longitud par, y es aceptado por el **AFD** definido por: $Q = \{ q_0, q_1 \}$, $s = q_0$, $\Sigma = \{ \mathbf{a}, \mathbf{b} \}$, $F = \{ q_0 \}$ y las transiciones siguientes:

$\delta(q_0, \mathbf{a}) = q_1$ $\delta(q_0, \mathbf{b}) = q_1$

$\delta(q_1, \mathbf{a}) = q_0$ $\delta(q_1, \mathbf{b}) = q_0$

Para hacer la **MT** equivalente, tenemos que: $Q' = \{ q_0, q_1, q_R, q_A \}$, $s = q_0$, $\Sigma = \{ \mathbf{a}, \mathbf{b} \}$, $\Gamma = \{ \mathbf{a}, \mathbf{b}, \# \}$, $R = \{ q_R \}$, $F' = \{ q_A \}$ y las transiciones siguientes:

$\delta'(q_0, \mathbf{a}) = (q_1, \mathbf{a}, R)$ $\delta'(q_0, \mathbf{b}) = (q_1, \mathbf{b}, R)$

$\delta'(q_1, \mathbf{a}) = (q_0, \mathbf{a}, R)$ $\delta'(q_1, \mathbf{b}) = (q_0, \mathbf{b}, R)$

$\delta'(q_0, \#) = (q_A, \#, R)$ $\delta'(q_1, \#) = (q_R, \#, R)$

En la Figura 10.6 se muestra gráficamente a la **MT** que decide el lenguaje en cuestión:

También es posible construir una **MT** no determinista a partir de un **AFN** dado, sin embargo, esto no es recomendable, siempre es preferible encontrar primero el **AFD** mínimo equivalente antes de construir la **MT** determinista.

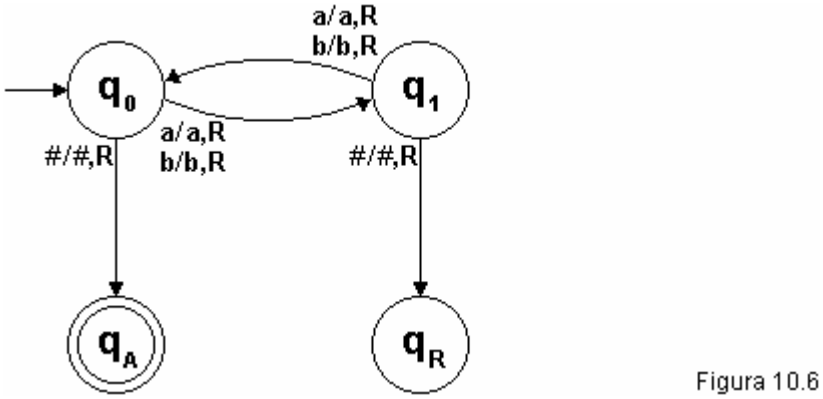


Figura 10.6

Decidibilidad de LICs

En el siguiente capítulo vamos a mostrar una técnica para construir una **MT** a partir de un **APD** o un **APN** dados, por lo pronto, nos limitaremos a afirmar que Todo **LIC** es Decidible y mostrar, con un par de ejemplos, como se puede construir una **MT** para decidir esta clase de lenguajes.

Ejemplo 1

Construir una Máquina de Turing para decidir al siguiente **LIC**: $L = \{ \mathbf{a}^n\mathbf{b}^n \mid n \geq 0 \}$. Este proceso requiere de varias fases, la primera consiste en reemplazar la primera **a** por un símbolo # para luego buscar la última **b** y cambiarla por otro #, con el fin de descartar por parejas. Para ello se usan las siguientes transiciones:

$\delta(q_0, \mathbf{a}) = (q_1, \#, R)$ $\quad\quad$ $\delta(q_1, \mathbf{b}) = (q_1, \mathbf{b}, R)$

$\delta(q_1, \mathbf{a}) = (q_1, \mathbf{a}, R)$ $\quad\quad$ $\delta(q_1, \#) = (q_2, \#, L)$

$\delta(q_2, \mathbf{b}) = (q_3, \#, L)$

En la segunda fase usamos el estado q_3 para retroceder hasta localizar el # escrito previamente y se regresa al estado q_0 para repetir el proceso anterior:

$\delta(q_3, \mathbf{a}) = (q_3, \mathbf{a}, L)$ $\quad\quad$ $\delta(q_3, \mathbf{b}) = (q_3, \mathbf{b}, L)$

$\delta(q_3, \#) = (q_0, \#, R)$

El proceso termina si al regresar al estado q_0 ya no queda ninguna **a** y tampoco ninguna **b**, sino que la cinta está en blanco y encontramos al símbolo #, esta situación también aplica si la cadena inicial es la cadena vacía, por tanto, la transición final debe ser:

$\delta(q_0, \#) = (q_A, \#, L)$

Si la cadena pertenece al lenguaje la cinta quedará en blanco y se finalizará en el estado de aceptación q_A, en caso contrario, la **MT** se utilizará alguna de las tres transiciones para ir al estado de rechazo, dependiendo del caso:

$\delta(q_2, \mathbf{a}) = (q_R, \mathbf{a}, L)$ $\quad\quad$ $\delta(q_0, \mathbf{b}) = (q_R, \mathbf{b}, L)$

$\delta(q_2, \#) = (q_R, \#, L)$

La Máquina de Turing está dada por: $Q = \{q_0, q_1, q_2, q_3, q_A, q_R\}$, $\Sigma = \{\mathbf{a},\mathbf{b}\}$, $\Gamma = \{\mathbf{a}, \mathbf{b}, \#\}$, $s = q_0$, $F = \{q_A\}$, $R = \{q_R\}$ y las 12 transiciones anteriores, por ejemplo, para procesar la cadena w_1 = **aabb**, tenemos las siguientes descripciones instantáneas:

$(q_0, \underline{\mathbf{a}}\mathbf{abb}) \vdash (q_1, \#\underline{\mathbf{a}}\mathbf{bb}) \vdash (q_1, \#\mathbf{a}\underline{\mathbf{b}}\mathbf{b}) \vdash (q_1, \#\mathbf{ab}\underline{\mathbf{b}}) \vdash (q_1, \#\mathbf{abb}\underline{\#}) \vdash (q_2, \#\mathbf{ab}\underline{\mathbf{b}}\#) \vdash$
$(q_3, \#\mathbf{a}\underline{\mathbf{b}}\#) \vdash (q_3, \#\underline{\mathbf{a}}\mathbf{b}\#) \vdash (q_3, \underline{\#}\mathbf{ab}\#) \vdash (q_0, \#\underline{\mathbf{a}}\mathbf{b}\#) \vdash (q_1, \#\underline{\mathbf{b}}\#) \vdash (q_1, \#\mathbf{b}\underline{\#}) \vdash (q_2, \#\underline{\mathbf{b}}\#) \vdash$
$(q_3, \underline{\#}\#) \vdash (q_0, \#\underline{\#}) \vdash (q_A, \underline{\#}\#)$

Al aplicar la **MT** anterior para cadenas que no pertenecen al lenguaje citado, tendremos situaciones como las siguientes, si sobra una **a**: $(q_0, \underline{\mathbf{a}}\mathbf{aabb}) \vdash^* (q_R, \underline{\#}\#)$ o bien si sobran varias **a**s: $(q_0, \underline{\mathbf{a}}\mathbf{aaab}) \vdash^* (q_R, \underline{\mathbf{a}}\mathbf{a})$, mientras que, cuando sobra una o varias **b**s: $(q_0, \underline{\mathbf{a}}\mathbf{abbb}) \vdash^* (q_R, \underline{\#}\mathbf{b})$, además, podemos tener situaciones en las que se encuentra una **a** donde debería haber una **b**, o que hay una **b** donde debería haber una **a** y que también nos conducirán a q_R.

Es frecuente que se considere innecesario definir las transiciones hacia el estado de rechazo, sin perder el concepto de Decidibilidad, sino que, como en el caso de los **AFD**s, se considere como rechazada a la cadena cuando no existen transiciones definidas aplicables, de esta manera nos podemos ahorrar el esfuerzo de enlistar una gran cantidad de transiciones hacia el estado de rechazo.

Ejemplo 2

Construir una Máquina de Turing para decidir el lenguaje $\{ w \in \{ \mathbf{a}, \mathbf{b} \}^* \mid w = w^R \}$. Esta Máquina de Turing está dada por: $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_A\}$, $s = q_0$, $\Sigma = \{\mathbf{a}, \mathbf{b}\}$, $\Gamma = \{ \mathbf{a}, \mathbf{b}, \# \}$, $F = \{ q_A \}$ y las transiciones que se explican a continuación:

Primero se lee el símbolo inicial de la cadena, si se trata de una **a** se borra, pasando al estado q_1 y se utilizan las transiciones de la columna izquierda para verificar si existe una **a** al final de la cadena. Pero si el símbolo inicial es una **b,** se borra pasando al estado q_2, y entonces se emplean las transiciones de la columna derecha para verificar que el símbolo del final de la cadena sea una **b**:

$\delta(q_0, \mathbf{a}) = (q_1, \#, R)$ $\delta(q_0, \mathbf{b}) = (q_2, \#, R)$

$\delta(q_1, \mathbf{a}) = (q_1, \mathbf{a}, R)$ $\delta(q_2, \mathbf{a}) = (q_2, \mathbf{a}, R)$

$\delta(q_1, \mathbf{b}) = (q_1, \mathbf{b}, R)$ $\delta(q_2, \mathbf{b}) = (q_2, \mathbf{b}, R)$

$\delta(q_1, \#) = (q_3, \#, L)$ $\delta(q_2, \#) = (q_4, \#, L)$

$\delta(q_3, \mathbf{a}) = (q_5, \#, L)$ $\delta(q_4, \mathbf{b}) = (q_5, \#, L)$

Si al regresar a la celda anterior la encuentra vacía, significa que se trataba de una palíndroma de longitud impar y se acepta la cadena:

$\delta(q_3, \#) = (q_A, \#, L)$ $\delta(q_4, \#) = (q_A, \#, L)$

El estado q_5 se utilizará para regresar al inicio de la cadena y repetir nuevamente el ciclo de transiciones, pero, si la cinta ya quedó en blanco, se pasará al estado de aceptación, pues se trata de una palíndroma de longitud par:

$\delta(q_5, \mathbf{a}) = (q_5, \mathbf{a}, L)$ $\delta(q_5, \mathbf{b}) = (q_5, \mathbf{b}, L)$

$\delta(q_5, \#) = (q_0, \#, R)$ $\delta(q_0, \#) = (q_A, \#, L)$

Ejemplo 3

Construir una Máquina de Turing para decidir el lenguaje $\{ w \in \{\mathbf{0}, \mathbf{1}\}^* \mid N_0(w)=N_1(w) \}$.

La estrategia que se sigue para construir esta **MT** consiste en leer y borrar el primer símbolo de la cadena, si éste es un **0**, deberá buscar un **1**, si lo encuentra marcarlo y si no, rechazará la cadena. Análogamente, si el símbolo inicial es un **1**, se buscará posteriormente un **0**. Pero si el símbolo inicial es uno previamente marcado, éste simplemente se borra. Este proceso se repite hasta agotar la cadena y es entonces cuando se acepta.

Las transiciones necesarias son:

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

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

$\delta(q_1, \mathbf{x}) = (q_1, \mathbf{x}, R)$ $\delta(q_2, \mathbf{x}) = (q_2, \mathbf{x}, R)$

$\delta(q_1, \#) = (q_R, \#, L)$ $\delta(q_2, \#) = (q_R, \#, L)$

$\delta(q_1, \mathbf{1}) = (q_3, \mathbf{x}, L)$ $\delta(q_2, \mathbf{0}) = (q_3, \mathbf{x}, L)$

$\delta(q_3, \sigma) = (q_3, \sigma, L)$ $\delta(q_3, \#) = (q_0, \#, R)$

$\delta(q_0, \mathbf{x}) = (q_0, \#, R)$ $\delta(q_0, \#) = (q_A, \#, R)$

Donde σ se emplea para representar a cualquier símbolo de la cinta distinto de #.

Otros Lenguajes Decidibles

Finalmente, mostraremos un par de ejemplos de construcción de **MT**s que deciden sobre las cadenas de un lenguaje que no es **LIC**.

Ejemplo 1

De manera análoga al caso presentado en el ejemplo anterior podemos construir una Máquina de Turing para decidir el lenguaje $\{\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n \mid n \geq 0\}$, del que ya se dijo antes no es un **LIC**. Esta Máquina de Turing está dada por: $Q = \{q_0, q_1, q_2, q_3, q_4, q_A\}$, $s = q_0$, $\sum = \{ \mathbf{a}, \mathbf{b} \}$, $\Gamma = \{ \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \# \}$, $F = \{ q_A \}$ y las transiciones siguientes:

$\delta(q_0, \#) = (q_A, \#, L)$ $\quad$ $\delta(q_0, \mathbf{a}) = (q_1, \#, R)$

$\delta(q_1, \mathbf{a}) = (q_1, \mathbf{a}, R)$ $\quad$ $\delta(q_1, \mathbf{d}) = (q_1, \mathbf{d}, R)$

$\delta(q_1, \mathbf{b}) = (q_2, \mathbf{d}, R)$ $\quad$ $\delta(q_2, \mathbf{b}) = (q_2, \mathbf{b}, R)$

$\delta(q_2, \mathbf{c}) = (q_2, \mathbf{c}, R)$ $\quad$ $\delta(q_2, \#) = (q_3, \#, L)$

$\delta(q_3, \mathbf{c}) = (q_4, \#, L)$ $\quad$ $\delta(q_4, \sigma) = (q_4, \sigma, L)$

$\delta(q_4, \#) = (q_0, \#, R)$ $\quad$ $\delta(q_0, \mathbf{d}) = (q_0, \mathbf{d}, R)$

Como ya se dijo, σ representa cualquier símbolo de la cinta distinto de #. En la Figura 10.7 tenemos la representación gráfica del diagrama de transiciones de esta **MT**:

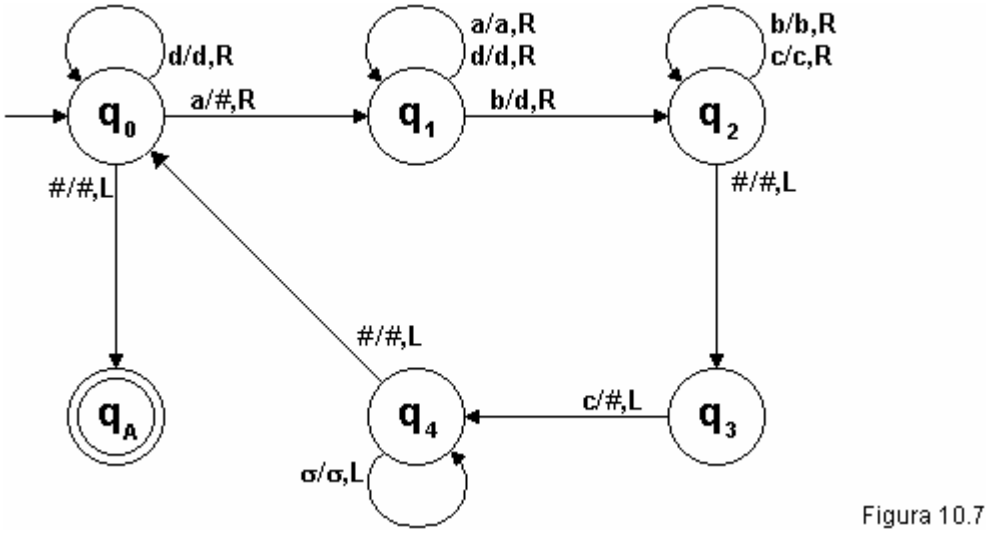


Figura 10.7

Por ejemplo, para aceptar la cadena w_1 = **abc**, tenemos la siguiente secuencia de descripciones instantáneas:

$(q_0, \underline{\mathbf{a}}\mathbf{bc}) \vdash (q_1, \#\underline{\mathbf{b}}\mathbf{c}) \vdash (q_2, \#\mathbf{d}\underline{\mathbf{c}}) \vdash (q_2, \#\mathbf{dc}\underline{\#}) \vdash (q_3, \#\mathbf{d}\underline{\mathbf{c}}\#) \vdash (q_4, \#\underline{\mathbf{d}}\#) \vdash (q_4, \underline{\#}\mathbf{d}\#) \vdash$
$(q_0, \#\underline{\mathbf{d}}\#) \vdash (q_0, \#\mathbf{d}\underline{\#}) \vdash (q_A, \#\underline{\mathbf{d}}\#)$

Aunque lo único necesario es alcanzar el estado de aceptación q_A para saber que la cadena pertenece a un lenguaje, en ocasiones se requiere que la **MT** escriba una respuesta de salida: generalmente un **1** o una **S** para indicar la aceptación y un **0** o una **N** para rechazarla. Por lo tanto, se puede complementar el ejemplo presente con las siguientes transiciones adicionales, que borran el contenido de la cinta y escriben el símbolo **1** para indicar que la cadena es aceptada:

$\delta(q_A, \mathbf{d}) = (q_A, \#, L)$ $\quad\quad$ $\delta(q_A, \#) = (q_6, \mathbf{1}, R)$

$\delta(q_6, \#) = (q_7, \#, L)$

Como se dijo, resulta muy laborioso determinar el conjunto de todas las transiciones necesarias para indicar que la cadena es rechazada, además, para todas las posibles configuraciones se tendría, que limpiar el contenido restante de la cinta y escribir un **0**

Ejemplo 2

Construir una Máquina de Turing para decidir el lenguaje L = { *ww* | *w* ∈ { **a**, **b** }* }. Las transiciones necesarias para que esta **MT** pueda reconocer las cadenas de L se dividen en dos procesos, el primero sirve para la localización de la mitad de la cadena, y verificar si es de longitud par, para ello sustituimos minúsculas por mayúsculas:

$\delta(q_0, \#) = (q_A, \#, R)$ $\quad\quad$ $\delta(q_0, \mathbf{a}) = (q_1, \mathbf{A}, R)$

$\delta(q_0, \mathbf{b}) = (q_1, \mathbf{B}, R)$ $\quad\quad$ $\delta(q_1, \mathbf{a}) = (q_1, \mathbf{a}, R)$

$\delta(q_1, \mathbf{b}) = (q_1, \mathbf{b}, R)$ $\quad\quad$ $\delta(q_1, \#) = (q_2, \#, L)$

$\delta(q_1, \mathbf{A}) = (q_2, \mathbf{A}, L)$ $\quad\quad$ $\delta(q_1, \mathbf{B}) = (q_2, \mathbf{B}, L)$

$\delta(q_2, \mathbf{a}) = (q_3, \mathbf{A}, L)$ $\quad\quad$ $\delta(q_2, \mathbf{b}) = (q_3, \mathbf{B}, L)$

$\delta(q_3, \mathbf{a}) = (q_3, \mathbf{a}, L)$ $\quad\quad$ $\delta(q_3, \mathbf{b}) = (q_3, \mathbf{b}, L)$

$\delta(q_3, \mathbf{A}) = (q_0, \mathbf{A}, R)$ $\quad\quad$ $\delta(q_3, \mathbf{B}) = (q_0, \mathbf{B}, R)$

Una vez concluido este primer proceso, marcamos con el símbolo **x** la celda donde inicia la segunda mitad de la cadena e iniciamos las transiciones que corresponden al segundo paso, y que dependen del símbolo que estaba en la celda marcada:

$\delta(q_0, \mathbf{A}) = (q_4, \mathbf{x}, L)$ $\quad\quad$ $\delta(q_0, \mathbf{B}) = (q_5, \mathbf{x}, L)$

Nos movemos al inicio de la cadena para verificar que el primer símbolo coincide con el que acabamos de marcar en la segunda mitad de la misma, si es así lo borramos y regresamos a la segunda mitad de la cadena, donde continuamos con el proceso de marcado y borrado:

$\delta(q_4, \sigma) = (q_4, \sigma, L)$ $\delta(q_5, \sigma) = (q_5, \sigma, L)$

$\delta(q_4, \#) = (q_6, \#, R)$ $\delta(q_5, \#) = (q_7, \#, R)$

$\delta(q_6, \mathbf{A}) = (q_8, \#, R)$ $\delta(q_7, \mathbf{B}) = (q_8, \#, R)$

$\delta(q_8, \mathbf{A}) = (q_8, \mathbf{A}, R)$ $\delta(q_8, \mathbf{B}) = (q_8, \mathbf{B}, R)$

$\delta(q_8, \mathbf{x}) = (q_9, \mathbf{x}, R)$ $\delta(q_9, \mathbf{x}) = (q_9, \mathbf{x}, R)$

$\delta(q_9, \mathbf{A}) = (q_4, \mathbf{x}, L)$ $\delta(q_9, \mathbf{B}) = (q_5, \mathbf{x}, L)$

Finalmente, para aceptar la cadena se tiene la última transición, cuando hemos borrado la primera mitad de la cadena y solamente quedan **x**s en la cinta:

$$\delta(q_9, \#) = (q_A, \#, R)$$

Variantes de la Máquina de Turing

Hay otras versiones de Máquinas de Turing que, aunque todas son equivalentes al modelo previamente descrito, son relevantes ya que permiten simplificar y eficientar ciertas tareas específicas. A continuación presentamos algunas de estas variantes.

Movimiento Estacionario

La definición original obliga a que la cabeza de lectura-escritura se desplace hacia la izquierda (L) o hacia la derecha (R), a veces es necesario hacer que la cabeza pueda permanecer en la posición original (S), lo que en esencia sintetiza dos transiciones en una sola, tal como se indica en el siguiente ejemplo:

$$\delta(q_0, \mathbf{b}) = (q_1, \mathbf{a}, R) \circ \delta(q_1, \mathbf{a}) = (q_2, \mathbf{a}, L) \Rightarrow \delta(q_0, \mathbf{b}) = (q_2, \mathbf{a}, S)$$

Lo único que hace falta en este caso es agregar la tercera opción en el contradominio de la función de transición: $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{ L, R, S \}$.

Máquinas Multiceldas

Otra modificación interesante es la llamada Máquina de Turing Multicelda, en la cual se considera que la cinta está formada por *n pistas*, por lo tanto, cada celda de la cinta está subdividida en n subceldas, lo que permite almacenar n símbolos por celda, como ejemplo tenemos la siguiente **MT** de tres pistas:

El contenido de cada celda puede ser representado entonces por una tercia ordenada de símbolos, en el ejemplo anterior la celda actual contiene (**0**, **1**, **1**). Como ya se dijo,

estas máquinas no tienen más poder, pero hacen más fácil la solución de ciertos problemas.

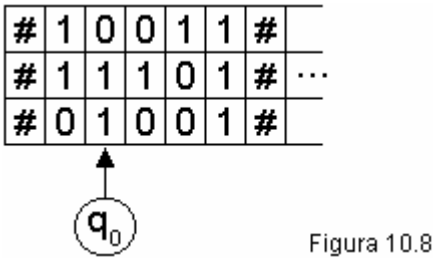


Figura 10.8

Ejemplo 1

Podemos utilizar una **MT** Multicelda para hacer la suma binaria de dos cadenas de unos y ceros de la misma longitud. En la primera pista se escribe la cadena que representa el primer sumando y en la segunda al otro y se realiza la suma sobre la tercer pista, inicialmente en blanco.

Por comodidad, se asume que la cabeza de lectura-escritura se encuentra en el extremo derecho de la cinta. El estado q_0, es para cuando la suma no implica *acarreo*, mientras que q_1 es el estado para la suma con acarreo. El conjunto de transiciones necesario para realizar la suma es:

$\delta(q_0, (\mathbf{0}, \mathbf{0}, \#)) = (q_0, (\mathbf{0}, \mathbf{0}, \mathbf{0}), L)$ $\quad$ $\delta(q_0, (\mathbf{0}, \mathbf{1}, \#)) = (q_0, (\mathbf{0}, \mathbf{1}, \mathbf{1}), L)$

$\delta(q_0, (\mathbf{1}, \mathbf{0}, \#)) = (q_0, (\mathbf{1}, \mathbf{0}, \mathbf{1}), L)$ $\quad$ $\delta(q_0, (\mathbf{1}, \mathbf{1}, \#)) = (q_1, (\mathbf{1}, \mathbf{1}, \mathbf{0}), L)$

$\delta(q_1, (\mathbf{0}, \mathbf{0}, \#)) = (q_0, (\mathbf{0}, \mathbf{0}, \mathbf{1}), L)$ $\quad$ $\delta(q_1, (\mathbf{1}, \mathbf{0}, \#)) = (q_1, (\mathbf{1}, \mathbf{0}, \mathbf{0}), L)$

$\delta(q_1, (\mathbf{0}, \mathbf{1}, \#)) = (q_1, (\mathbf{0}, \mathbf{1}, \mathbf{0}), L)$ $\quad$ $\delta(q_1, (\mathbf{1}, \mathbf{1}, \#)) = (q_1, (\mathbf{1}, \mathbf{1}, \mathbf{1}), L)$

$\delta(q_0, (\#, \#, \#)) = (q_2, (\#, \#, \#), R)$ $\quad$ $\delta(q_1, (\#, \#, \#)) = (q_2, (\mathbf{0}, \mathbf{0}, \mathbf{1}), S)$

Como ejemplo se muestra la siguiente suma, usando las transiciones precedentes:

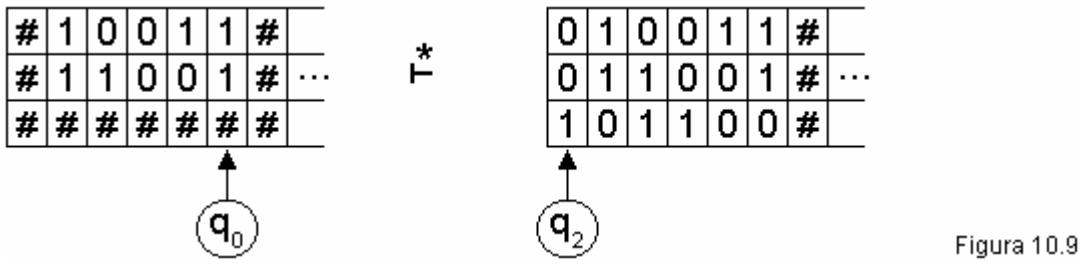


Figura 10.9

Ejemplo 2

Una situación común en la que se emplean las **MT** multiceldas, consiste en hacer el reconocimiento de una cadena sin alterarla, empleando una segunda pista para hacer

el marcaje de las celdas. Con base en lo anterior, construir una **MT** de dos pistas para reconocer las cadenas del siguiente lenguaje: L = { w ∈ {**a**, **b**}* | $N_a(w)=N_b(w)$ }.

Las transiciones necesarias son:

$\delta(q_0, (\mathbf{a}, \#)) = (q_1, (\mathbf{a}, \mathbf{x}), R)$ $\quad$ $\delta(q_0, (\mathbf{b}, \#)) = (q_2, (\mathbf{b}, \mathbf{x}), R)$

$\delta(q_1, (\mathbf{a}, \#)) = (q_1, (\mathbf{a}, \#), R)$ $\quad$ $\delta(q_2, (\mathbf{b}, \#)) = (q_2, (\mathbf{b}, \#), R)$

$\delta(q_1, (\mathbf{b}, \mathbf{y})) = (q_1, (\mathbf{b}, \mathbf{y}), R)$ $\quad$ $\delta(q_2, (\mathbf{a}, \mathbf{y})) = (q_2, (\mathbf{a}, \mathbf{y}), R)$

$\delta(q_1, (\#, \#)) = (q_R, (\#, \#), L)$ $\quad$ $\delta(q_2, (\#, \#)) = (q_R, (\#, \#), L)$

$\delta(q_1, (\mathbf{b}, \#)) = (q_3, (\mathbf{b}, \mathbf{y}), L)$ $\quad$ $\delta(q_2, (\mathbf{a}, \#)) = (q_3, (\mathbf{a}, \mathbf{y}), L)$

$\delta(q_3, (\mathbf{a}, \#)) = (q_3, (\mathbf{a}, \#), L)$ $\quad$ $\delta(q_3, (\mathbf{b}, \#)) = (q_3, (\mathbf{b}, \#), L)$

$\delta(q_3, (\mathbf{a}, \mathbf{y})) = (q_3, (\mathbf{a}, \mathbf{y}), L)$ $\quad$ $\delta(q_3, (\mathbf{b}, \mathbf{y})) = (q_3, (\mathbf{b}, \mathbf{y}), L)$

$\delta(q_3, (\mathbf{a}, \mathbf{x})) = (q_0, (\mathbf{a}, \mathbf{x}), R)$ $\quad$ $\delta(q_3, (\mathbf{b}, \mathbf{x})) = (q_0, (\mathbf{b}, \mathbf{x}), R)$

$\delta(q_0, (\mathbf{a}, \mathbf{y})) = (q_0, (\mathbf{a}, \mathbf{x}), R)$ $\quad$ $\delta(q_0, (\mathbf{b}, \mathbf{y})) = (q_0, (\mathbf{b}, \mathbf{x}), R)$

$\delta(q_0, (\#, \#)) = (q_A, (\#, \#), R)$

Debido a que este tipo de **MT**s no es más eficiente que la correspondiente de una sola pista, no existen otras razones prácticas para su uso, con excepción de las aplicaciones semejantes a la del primer ejemplo.

Máquinas Semi-Acotadas

Otra variante común es la **MT** semi-acotada, es aquella que solamente es infinita en una dirección, mientras que en la otra dirección está acotada por una celda inicial, la cual no se debe intentar traspasar hacia la izquierda, porque se provocaría un error. Debido a esto, conviene que se marque esta celda inicial con un símbolo especial, como por ejemplo, el asterisco, para indicar que nos encontramos en el límite izquierdo de la misma y evitar cualquier transición en esa dirección, tal como se muestra en la Figura 10.10:

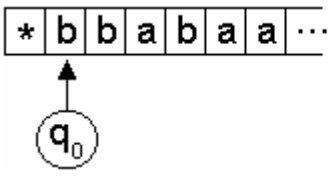


Figura 10.10

Máquinas Multicintas

El modelo multicinta es muy eficiente, éste consiste en una **MT** que contiene varias cintas y cada una de ellas con su cabeza de lectura-escritura, las cuales se mueven de manera independiente. Este modelo es muy flexible, cada transición permite cambiar de estado dependiendo del contenido en las celdas de todas las cintas, escribir un símbolo distinto en cada celda para cada una de las cintas, así como mover la cabeza de lectura-escritura de cada una de las cintas en direcciones independientes, entonces tenemos que: δ: $Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{R,L,S\}^n$.

Ejemplo

Construir una **MT** multicinta para el reconocimiento de cadenas pertenecientes al lenguaje L = { $\mathbf{a}^n\mathbf{b}^n$ | n > 0 }, para ello se emplean las siguientes transiciones:

$\delta(q_0, (\mathbf{a}, \#)) = (q_0, (\mathbf{a}, \mathbf{a}), (R, R))$ $\quad$ $\delta(q_0, (\mathbf{b}, \#)) = (q_1, (\mathbf{b}, \#), (S, L))$

$\delta(q_1, (\mathbf{b}, \mathbf{a})) = (q_1, (\mathbf{b}, \mathbf{a}), (R, L))$ $\quad$ $\delta(q_1, (\#, \#)) = (q_2, (\#, \#), (S, R))$

La operación de esta **MT** la representamos en la Figura 10.11:

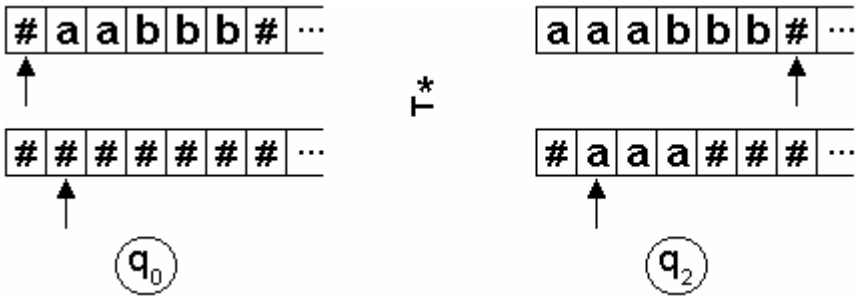


Figura 10.11

Máquinas Multidimensionales

Finalmente, también podemos hablar de la **MT** multidimensional, la cual puede realizar desplazamientos en varias dimensiones, como por ejemplo, en la Figura 10.12 se muestra una **MT** de dos dimensiones, la cual permite los desplazamientos a la izquierda y a la derecha, así como también hacia arriba o hacia abajo (U, D).

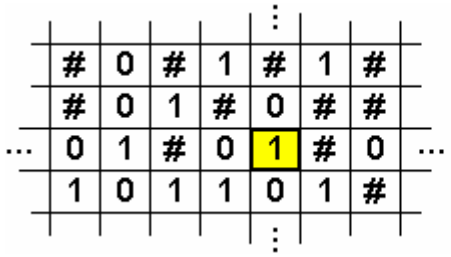


Figura 10.12

Por cada dimensión adicional se deberán considerar dos direcciones más, por ejemplo, frente y atrás (F, B) para la tercera dimensión, y así sucesivamente.

A pesar de ser más eficientes que los modelos anteriores, son poco utilizadas, debido a la complejidad que involucran en su conceptualización, por lo que no incluiremos ningún ejemplo de este modelo.

Autómata de Dos Pilas

Una Máquina de Turing puede también ser simulada por un Autómata Determinista de Dos Pilas, a continuación se explica el procedimiento a seguir:

1. Primero se copia la cadena contenida en la cinta a la primera pila.
2. A continuación se vacía el contenido de la primera pila a la segunda, con objeto de que el símbolo de la cima de la pila coincida con el inicio de la cadena.
3. Ahora se inicia la simulación propiamente: cada transición de la **MT** hacia la derecha se realiza desapilando un símbolo de la segunda pila y pasándolo a la primera.
4. Cada transición de la **MT** hacia la izquierda, por el contrario, se realiza desapilando un símbolo de la primera pila y pasándolo a la segunda. De esta forma el símbolo que apunta la **CLE** en la **MT** será siempre el que se encuentre en la cima de la segunda pila, y los símbolos a la izquierda de ésta serán los contenidos en la primera pila, tal como se ejemplifica en la Figura 10.13:

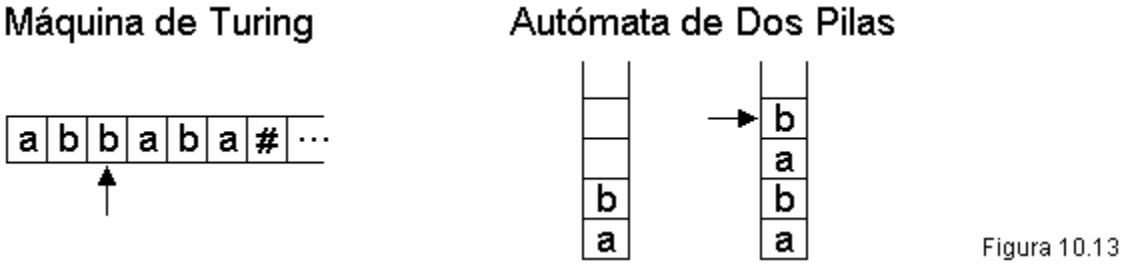


Figura 10.13

5. La escritura de la cinta se realiza simplemente definiendo transiciones en las que se reemplace el símbolo que se desapila (lectura) por el nuevo símbolo que se apila (escritura).

Máquina Universal de Turing

La Máquina Universal de Turing (**MUT**) es una **MT** que nos permite reproducir el funcionamiento de cualquier otra máquina M. Para ello, es necesario establecer un

código que nos permita proporcionar a la **MUT** la información de las transiciones de máquina M, dada bajo las siguientes consideraciones:

Por ejemplo, sea M una **MT** cuyos estados son: Q = { q_1, q_2, …, q_n }, donde s = q_1 y F = { q_2 }, entonces podemos iniciar la codificación de los estados de M, de la siguiente manera: al estado q_1 le asignamos el código **1**, a q_2 le corresponde **11**, q_3 por **111**, y así sucesivamente hasta llegar a q_n representado por $\mathbf{1}^n$.

Similarmente, si los símbolos admitidos por la cinta de M son Γ = { σ_1, …, σ_m }, donde # = σ_1, representamos a cada símbolo σ_i por una cadena de i unos, en este caso, el símbolo # es representado por **1**, mientras que σ_m le corresponde $\mathbf{1}^m$.

Finalmente los desplazamientos a la izquierda se representan por **1** y a la derecha por **11**.

Entonces una transición de M quedará codificada por cinco sucesiones de unos, separadas por un cero, con un cero al principio y otro al final, de tal forma que, por ejemplo, para codificar a la siguiente transición:

$$\delta(q_3, \sigma_1) = (q_2, \sigma_3, R)$$

La representamos mediante la siguiente cadena: **011101011011110110**.

Entonces, la **MUT** se construirá como una máquina de tres cintas; en la primera se colocarán todas las transiciones codificadas de M, una a continuación de otra y la cabeza de lectura-escritura se ubica al inicio de esta cadena.

En la segunda cinta se escribe la codificación del contenido de la cinta de M (la cadena de entrada) como varias secuencias de unos, separadas por ceros, la cabeza de lectura-escritura también se sitúa al inicio de esta cadena.

La tercera cadena contiene la codificación del estado actual, que inicialmente es q_1, representado por un **1**.

El funcionamiento de la **MUT** requiere de muchas transiciones para ejecutar cada transición de M, por ejemplo, primero deberá leer el estado actual en la tercera cinta, a continuación el símbolo actual en la segunda y finalmente buscar la transición correspondiente en la primera cinta; una vez encontrada, modificará el estado actual y el contenido de la cadena en la segunda cinta, según indique la transición, así como buscará el siguiente cero a la derecha o a la izquierda, dependiendo de lo que indique la transición.

La **MUT** aceptará la cadena si M la acepta, y el contenido de la tercera cinta será **11**, que corresponde al estado de aceptación y rechazará la cadena si M la rechaza, de tal forma que la **MUT** se comportará de manera idéntica a M.

Ejemplo

Codifique la **MT** definida por las siguientes transiciones:

$\delta(q_0, \mathbf{a}) = (q_0, \mathbf{a}, R)$ $\quad\quad$ $\delta(q_0, \mathbf{b}) = (q_1, \mathbf{b}, R)$

$\delta(q_1, \mathbf{b}) = (q_1, \mathbf{b}, R)$ $\quad\quad$ $\delta(q_1, \#) = (q_A, \#, R)$

Codificando los diferentes estados, tenemos: $1 - q_0$, $11 - q_A$ y $111 - q_1$.

Ahora, codificando los diferentes símbolos: $1 - \#$, $11 - \mathbf{a}$ y $111 - \mathbf{b}$.

Y como todos los desplazamientos son a la derecha, codificamos solamente: 11 – R.

Por lo tanto, la siguiente cadena representa la codificación solicitada:

0101101011011001011101110111011001110111011101110111001110101101010110

Máquinas de Turing básicas

Antes de pretender construir Máquinas de Turing más compleja, debemos construir una serie de **MT**s básicas, que realicen operaciones muy elementales, fundamentalmente las hay de dos tipos, las que sirven para posicionar la **CLE** y las que sirven para escribir.

Movimientos

La Máquina de Turing que se desplaza una celda a la derecha se denota por la letra R y consiste de tan sólo dos transiciones:

$\delta(q_0, \sigma) = (q_1, \sigma, R)$ $\quad\quad$ $\delta(q_0, \#) = (q_1, \#, R)$

Donde σ representa cualquier símbolo de la cinta distinto del vacío.

Similarmente, la Máquina de Turing que se desplaza una celda a la izquierda se denota por la letra L y también consta de dos transiciones:

$\delta(q_0, \sigma) = (q_1, \sigma, L)$ $\quad\quad$ $\delta(q_0, \#) = (q_1, \#, L)$

Búsqueda de celdas vacías

La siguiente **MT** busca la primera celda en blanco #, ubicada a la derecha de la posición actual, y ahí se para, a esta máquina se le suele referir como $R_{\#}$ y es aplicable a cualquier alfabeto y para cualquier contenido de la cinta.

Esta **MT** está dada por: $Q = \{ q_0, q_1, q_2, q_3 \}$, $s = q_0$, $\Gamma = \Sigma \cup \{\#\}$, $F = \{ q_3 \}$ y las transiciones:

$\delta(q_0, \sigma) = (q_1, \sigma, R)$ $\quad$ $\delta(q_0, \#) = (q_1, \#, R)$

$\delta(q_1, \sigma) = (q_1, \sigma, R)$ $\quad$ $\delta(q_1, \#) = (q_2, \#, L)$

$\delta(q_2, \sigma) = (q_3, \sigma, R)$ $\quad$ $\delta(q_2, \#) = (q_3, \#, R)$

De manera muy similar se puede construir la máquina que realice la búsqueda del primer símbolo blanco a la izquierda de la posición actual: $L_{\#}$, basta reemplazar las L por R y viceversa.

Búsqueda de celdas no vacías

Esta otra **MT** busca el primer símbolo no blanco σ, ubicado a la izquierda de la posición actual, y ahí se para, a esta máquina se le suele referir como $L_{\#'}$ y es aplicable a cualquier alfabeto y para cualquier contenido de la cinta.

Esta **MT** está dada por: $Q = \{ q_0, q_1, q_2, q_3 \}$, $s = q_0$, $\Gamma = \Sigma \cup \{ \# \}$, $F = \{ q_3 \}$ y las transiciones:

$\delta(q_0, \sigma) = (q_1, \sigma, L)$ $\quad$ $\delta(q_0, \#) = (q_1, \#, L)$

$\delta(q_1, \sigma) = (q_2, \sigma, R)$ $\quad$ $\delta(q_1, \#) = (q_1, \#, L)$

$\delta(q_2, \sigma) = (q_3, \sigma, L)$ $\quad$ $\delta(q_2, \#) = (q_3, \#, L)$

De manera muy similar se puede construir la máquina que realice la búsqueda del primer símbolo no blanco a la derecha de la posición actual: $R_{\#'}$, basta reemplazar las L por R y viceversa.

Sin embargo, si se pretende dirigir la búsqueda hacia una dirección en la que el resto de la cinta esté en blanco, ya sea a la derecha del fin de la cadena o a la izquierda de su inicio, la **MT** correría por siempre y nunca pararía.

Búsqueda del símbolo **a**

Siguiendo el mismo orden de ideas, se puede construir una máquina que realice la búsqueda del símbolo **a**, a la izquierda de la posición actual: L_a, o a la derecha de la posición actual: R_a.

Complementariamente, se podrían hacer máquinas que realicen la búsqueda del primer símbolo diferente de **a**, a la izquierda o a la derecha de la posición actual, denotadas respectivamente como: $L_{a'}$ y $R_{a'}$.

Escritura del símbolo **a**

Esta **MT** reemplaza el contenido de la celda donde se encuentra, no importando cual sea, por un símbolo **a** y permanece en esa misma celda; a esta máquina la vamos a designar simplemente por la letra **a**.

Sea otra **MT** dada por: Q = { q_0, q_1, q_2 }, **a**, $\sigma \in \Sigma$, $\Gamma = \Sigma \cup \{\#\}$, s = q_0, F = { q_2 } y

$\delta(q_0, \sigma) = (q_1, \mathbf{a}, R)$ $\quad$ $\delta(q_0, \#) = (q_1, \mathbf{a}, R)$

$\delta(q_1, \sigma) = (q_2, \sigma, L)$ $\quad$ $\delta(q_1, \#) = (q_2, \#, L)$

Máquinas de Turing Compuestas

La combinación de dos o más Máquinas de Turing se realiza bajo la suposición de que ambas comparten la misma cinta, y que al terminar la ejecución de la primera máquina, la cadena de salida contenida en la cinta, será la cadena de entrada para el inicio de la segunda, la posición de la cabeza de lectura será sobre la celda donde terminó la ejecución de la primera máquina.

Ejemplos varios

La composición siguiente: $R_{\#}\mathbf{a}$ es una nueva **MT** que busca el primer blanco a la derecha de la posición actual, lo reemplaza por una **a** y ahí se queda, como se ilustra en la figura 10.14:

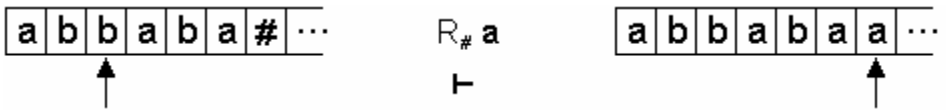


Figura 10.14

Para buscar hacia la derecha de la celda actual a la tercer celda no vacía, utilizamos la composición $R_{\#'}$ $R_{\#'}$ $R_{\#'}$, como se aprecia en la figura 10.15.

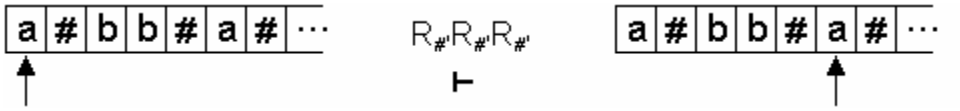


Figura 10.15

Y para buscar hacia la izquierda de la posición actual a la tercera celda vacía, usamos la composición $L_{\#}\, L_{\#}\, L_{\#}$, (Figura 10.16)

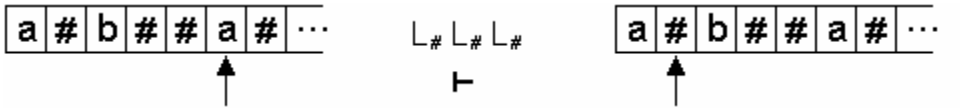


Figura 10.16

La máquina L**a**L**a**L**a**, escribe una secuencia de tres **a**s a la izquierda de la posición inicial (ver figura 10.17)

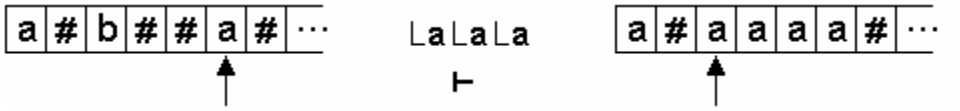


Figura 10.17

Finalmente, la Máquina: $R_{\#}\mathbf{a}L_{\#}R$, busca el primer blanco al final de la cadena, adiciona una **a** y se mueve a la celda inicial de la cadena, ver figura 10.18.

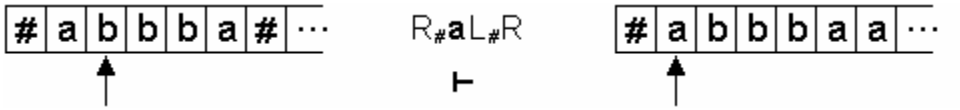


Figura 10.18

Bifurcación

Podemos construir una Máquina de Turing más versátil, que realice una acción si el símbolo encontrado es uno y otra acción distinta si el símbolo leído es otro distinto, como por ejemplo, si halla un **0** deberá escribir un **1**, pero si hay un **1**, entonces escribirá un **0**, la parte esencial de esta máquina condicional es la *Bifurcación*, que consta de las siguientes transiciones:

$\delta(q_0, \mathbf{0}) = (q_1, \mathbf{0}, R)$ $\qquad$ $\delta(q_0, \mathbf{1}) = (q_3, \mathbf{1}, R)$

$\delta(q_1, \mathbf{0}) = (q_2, \mathbf{0}, L)$ $\qquad$ $\delta(q_1, \mathbf{1}) = (q_2, \mathbf{1}, L)$

$\delta(q_3, \mathbf{0}) = (q_4, \mathbf{0}, L)$ $\qquad$ $\delta(q_3, \mathbf{1}) = (q_4, \mathbf{1}, L)$

Como podemos observar, esta máquina no altera el contenido de la cinta, sino que solamente elige uno de los dos estados finales, según el símbolo que se encuentre en la celda actual, si hay un **0** lo indicará terminando en el estado q_2, mientras que si encuentra un **1**, lo señalará finalizando en el estado q_4.

Gráficamente podemos representar la bifurcación mediante el siguiente diagrama:

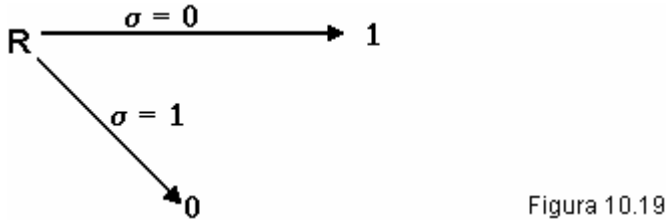


Figura 10.19

Ejemplo

Con esta idea, podemos construir una máquina compuesta como la que se muestra en la Figura 10.20, para que analice una cadena y que produzca su complemento, es decir, que reemplace los unos por ceros y los ceros por unos, como por ejemplo, en la siguiente computación: $(q_0, \underline{\#}\mathbf{0110}) \vdash^* (q_f, \underline{\#}\mathbf{1001})$:

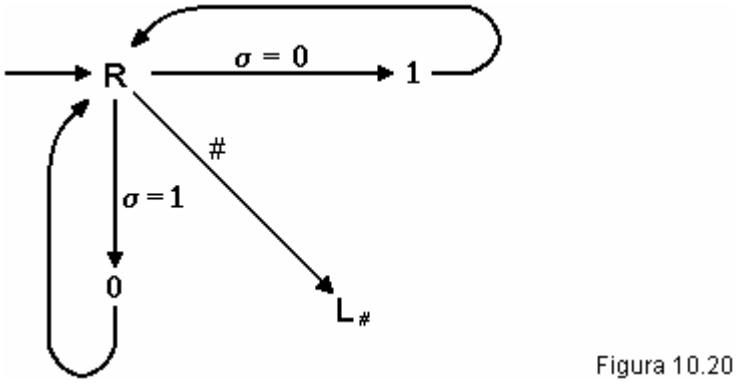


Figura 10.20

Esta representación nos permite construir máquinas más complejas, sin entrar en el detalle de las transiciones, como se muestra en los ejemplos siguientes.

Corrimiento a la Derecha

En la figura 10.21 se representa a una Máquina de Turing que realiza el corrimiento de una cadena a la derecha, es decir, que transforma la cadena $\underline{\#}w\#$ en $\#\underline{\#}w$, la cual se le identifica como S_R (Shift-Right), de manera semejante se puede construir la **MT** S_L, que efectúe el corrimiento a la izquierda de la cadena.

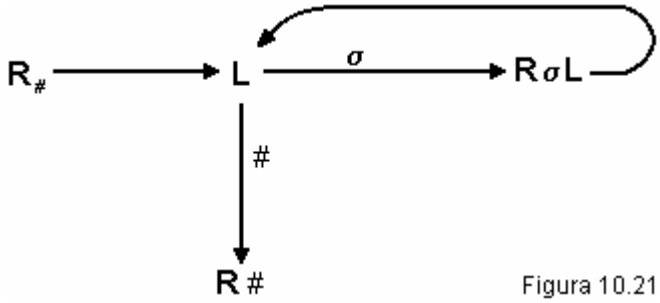


Figura 10.21

Copia de Cadenas

En la siguiente figura se muestra a una Máquina de Turing, denominada C, que realiza la copia de una cadena, es decir, que dada la cadena $w \in \{\mathbf{a}, \mathbf{b}\}^*$, el resultado en la cinta queda como $\underline{\#}w\#w$.

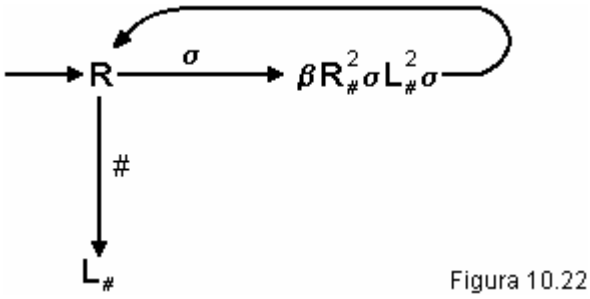


Figura 10.22

Duplicación de Cadenas

Para duplicar una cadena, se realiza la copia con C y luego se elimina el símbolo vacío intermedio, con S_R, para finalmente obtener *ww*, como se ilustra en la siguiente figura:

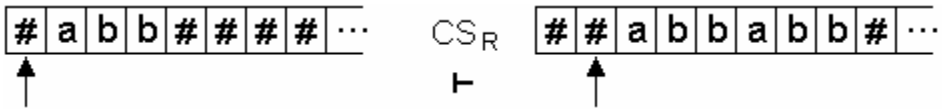


Figura 10.23

Reconocimiento de Cadenas

Por último, en la figura 10.24 se muestra una **MT** que hace el reconocimiento de cadenas de la forma ww^R, (palíndromas de longitud par), supongamos que el contenido inicial de la cinta es $\underline{\#}ww^R\#$, procedemos a comparar los extremos de la cadena para verificar que coinciden, borrando los símbolos identificados, continuando de la misma manera mientras haya coincidencia si el proceso finaliza normalmente, la cinta queda vacía y únicamente se escribe el símbolo **1** (Acepta), para confirmar que la cadena es aceptada, en caso contrario, se termina de borrar la cadena y se escribe el símbolo **0**, para denotar que la cadena es rechazada.

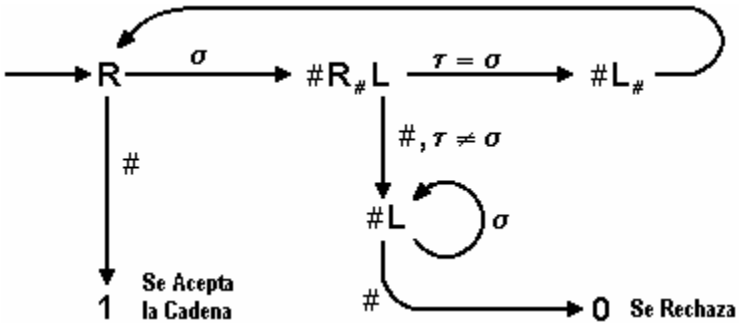


Figura 10.24

Preguntas

a) ¿Es necesario que una **MT** agote la cadena de entrada para aceptarla?

b) ¿Es siempre posible encontrar una **MT** de una cinta equivalente a una **MT** multicinta dada?

c) ¿Se pueden diseñar **MT** que sean multicintas-multiceldas?

d) ¿Se puede diseñar una **MUT** que permita emular a **MT**s multicintas?

e) En un autómata de dos pilas, ¿Qué pasa si una de las pilas se vacía? ¿Cómo se puede simular a una **MT** que se desplaza más allá de los límites de la cadena?

Ejercicios Capítulo 10

10.1 Diseñe y escriba las transiciones de una Máquina de Turing que realice cada una de las siguientes funciones:

a) Dada una cadena de entrada de la forma *w***c***x*, donde *w*, *x* ∈ {**a**, **b**}*, arroje como resultado la cadena *xw*.

b) Que duplique una cadena, es decir, dada la cadena de entrada *w* ∈ {**a**, **b**}*, arroje como resultado *ww*.

c) Dados dos números, en notación unaria, identifique el mínimo; la entrada es una cadena de la forma $\mathbf{1}^n\#\mathbf{1}^m$, con n, m ≥ 0, mientras que la salida es: $\mathbf{1}^m$, si m < n o $\mathbf{1}^n$, en caso contrario.

d) Dada una cadena cualquiera *w* ∈ {**a**, **b**, **c**}*, ordene los símbolos, colocando primero las **a**s, a continuación las **b**s y finalmente las **c**s.

e) Dada una cadena cualquiera *w* ∈ {**a**, **b**}*, nos entregue una cadena de salida que tenga tantos **1**s como veces aparezca la secuencia **ab** dentro de la cadena *w*.

f) Dada una cadena de entrada de la forma $w = \mathbf{1}^n$, n ≥ 0, nos entregue una cadena de salida que tenga la forma $(\mathbf{01})^n$.

g) Dada una cadena de entrada de la forma $w = (\mathbf{ab})^n$, n > 0, nos entregue una cadena de salida que tenga la forma $\mathbf{1}^n$, y nos entregue la cadena **0**, si la entrada no corresponde al formato esperado.

10.2 Diseñe y escriba las transiciones de una Máquina de Turing para realizar un contador binario como sigue:

$$(q0, 0\#) \vdash^* (q0, 1\#) \vdash^* (q0, 10\#) \vdash^* (q0, 11\#) \vdash^* (q0, 100\#) \vdash^* \ldots$$

10.3 Diseñe y escriba las transiciones de una máquina de Turing que iniciando en una cinta en blanco, enumere en orden ascendente las cadenas del siguiente lenguaje: $L = \{ \mathbf{a}^n \mid n > 0 \}$, separadas por un espacio en blanco.

10.4 Diseñe y escriba las transiciones de una máquina de Turing que realice la multiplicación $n \times m$.

10.5 Diseñe y escriba las transiciones de una máquina de Turing que calcule n^2, utilizando el desarrollo de una suma de n impares: $n^2 = 1 + 3 + \ldots + (2n - 1)$.

10.6 Obtenga una definición recursiva de n! y diseñe una **MT** que calcule n!

10.7 Diseñe y escriba las transiciones de una **MT** que permita decidir las cadenas para cada uno de los siguientes lenguajes (omita las transiciones de rechazo):

a) $L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid$ la longitud de w es par$\}$

b) $L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w$ contiene al menos una **a**$\}$

c) $L = (\mathbf{aa} \cup \mathbf{bb})^*$

d) $L = \{\mathbf{a}^n\mathbf{b}^m \mid n, m \geq 0, m \neq n \}$

e) $L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w = w^R \}$

f) $L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w \neq w^R \}$

g) $L = \{ w\mathbf{c}w \mid w \in \{\mathbf{a}, \mathbf{b}\}^* \}$

h) $L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid w$ es una cadena de longitud par, no contiene ninguna **a** en la primera mitad y ninguna **b** en la segunda$\}$

i) $L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid N_{\mathbf{a}}(w) = N_{\mathbf{b}}(w) = N_{\mathbf{c}}(w) \}$

j) $L = \{ \mathbf{a}^n\mathbf{b}^{2n} \mid n \geq 0 \}$

k) $L = \{ \mathbf{a}^m\mathbf{b}^n\mathbf{a}^m\mathbf{b}^n \mid m, n > 0 \}$

l) $L = \{ \mathbf{a}^{2^n} \mid n \geq 0\}$

10.8 Diseñe y escriba las transiciones de una máquina de Turing de tres pistas que haga la resta de dos números binarios, asumiendo que el minuendo es mayor que el sustraendo, es decir, que la diferencia es positiva.

10.9 Diseñe y escriba las transiciones de una máquina de Turing de dos pistas que permita el reconocimiento del lenguaje $L = \{ w\mathbf{c}w \mid w \in \{\mathbf{a}, \mathbf{b}\}^* \}$, deje intacta la cadena original en la primera pista y utilice la segunda pista para ir marcando los símbolos que van siendo comparados.

10.10 Diseñe y escriba las transiciones de una máquina de Turing de dos pistas que permita el reconocimiento del lenguaje $L = \{ ww \mid w \in \{\mathbf{a}, \mathbf{b}\}^* \}$, de manera semejante al problema anterior.

10.11 Diseñe y escriba las transiciones de una máquina de Turing de dos cintas que permita el reconocimiento del lenguaje $L = \{ ww^R \mid w \in \{\mathbf{a}, \mathbf{b}\}^* \}$.

10.12 Diseñe y escriba las transiciones de una máquina de Turing de dos cintas que permita el reconocimiento del lenguaje $L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid N_{\mathbf{a}}(w) = N_{\mathbf{b}}(w) \}$.

10.13 Diseñe y escriba las transiciones de una máquina de Turing de dos cintas que duplique una cadena, es decir, dado $w \in \{\mathbf{a},\mathbf{b}\}^*$, escriba *ww*.

10.14 Diseñe y escriba las transiciones para cada una de las siguientes Máquinas de Turing básicas:

a) La máquina de Turing $L_{\#}$, que busque el primer blanco a la izquierda de la posición actual y que se pare sobre dicha celda.

b) La máquina de Turing $R_{\#'}$, que busca hacia la derecha de la posición actual el primer símbolo no blanco y que se pare sobre dicha celda.

10.15 Por medio de la composición de máquinas básicas, construir cada una de las Máquinas de Turing siguientes:

a) Una Máquina de Turing que acepte las cadenas pertenecientes al siguiente lenguaje L = { w $\in$ { **a**, **b** }* | el tercer símbolo de w es una **a** }.

b) Una Máquina de Turing que acepte las cadenas pertenecientes al siguiente lenguaje L = { w $\in$ { **a**, **b** }* | el segundo y tercer símbolos de w son una **b** }

c) Una Máquina de Turing que acepte las cadenas pertenecientes al siguiente lenguaje L = { w $\in$ { **a**, **b** }* | w contiene a la subcadena **ab** }

d) Una Máquina de Turing que acepte las cadenas pertenecientes al siguiente lenguaje L = { w $\in$ { **a**, **b** }* | w contiene a la subcadena **bba** }

e) La Máquina de Turing S_L, que realice el Corrimiento a la Izquierda (Shift-Left) de la cadena *w*, es decir, que cambie la cadena #*w* en *w*#.

f) Dada una cadena de ceros y unos, ordene los símbolos, colocando primero todos los ceros y a continuación los unos.

g) La máquina de Turing que decida las cadenas pertenecientes al siguiente lenguaje L = { *w***c***w* | w $\in$ {**a**, **b**}* }

Capítulo

11

Gramáticas No Restringidas

Se define el concepto de Gramáticas No Restringidas, se expone la Jerarquía de Chomsky para las Gramáticas, se definen la clase de Lenguajes Enumerables y se establece la equivalencia entre las GNR y las MTs.

Lenguajes Decidibles

En el capítulo anterior estudiamos el caso de las **MT**s que siempre son capaces de aceptar o rechazar cualquier cadena dada de un lenguaje determinado; es decir, estas **MT**s siempre alcanzan un estado de parada, y pueden, por tanto, *decidir* si la cadena pertenece o no al lenguaje en cuestión, es por esto, que a estos lenguajes se les denomina como *Lenguajes Decidibles* (**LD**), todos los Lenguajes Independientes del Contexto (**LIC**s) son Decidibles, y por lo tanto, todos los Lenguajes Regulares (**LR**s) también son **LD**s.

Todo LR es Decidible

En el capítulo anterior se mostraron dos ejemplos de la construcción de **MT**s para el reconocimiento de Lenguajes Regulares.

Ahora expondremos una técnica para construir una **MT** a partir de un **AFD** cualquiera, el procedimiento es como sigue:

Dado el **AFD** compuesto por M = (Q, Σ, s, F, δ), entonces se puede construir la **MT** equivalente como sigue: M' = (Q', Σ, Γ, s, #, F', R, δ'), tal que L(M) = L(M'), en donde: Q' = Q $\cup$ { q_A, q_R }, $\Gamma = \Sigma \cup \{\#\}$, R = {$q_R$} y F' = {$q_A$}.

Las transiciones de la **MT** se obtienen a partir de las transiciones del **AFD**, además de otras más que nos llevan a los estados adicionales q_A y q_R, los cuales nos sirven para decidir si la cadena pertenece o no al lenguaje en cuestión:

$\delta'(q_i, \sigma_j) = (\delta(q_i, \sigma_j), \sigma_j, R)$, para todo $q_i \in Q$ y $\sigma_j \in \Sigma$

$\delta'(q_i, \#) = (q_R, \#, R)$, para todo $q_i \notin F$

$\delta'(q_i, \#) = (q_A, \#, R)$, para todo $q_i \in F$

Todo LICD es Decidible

Ahora veamos como es posible verificar que la clase de los Lenguajes Independientes del Contexto es un subconjunto de la clase de los Lenguajes Decidibles.

Primero vamos a demostrar que todo Lenguaje Independiente del Contexto Determinista es también un Lenguaje Decidible, para ello mostraremos, por medio de un ejemplo, que se puede construir una Máquina de Turing que decida sobre las cadenas de un lenguaje dado y que funciona de manera equivalente al Autómata de Pila Determinista que acepta dicho lenguaje; en este ejemplo se hace uso del símbolo de fin de cadena **$**, para enfatizar el carácter determinista del **APD**:

Ejemplo

Sea M el **APD** dado por Q = { q_0, q_1, q_A }, Σ = { **a**, **b**, **$** }, Γ = {A, B}, F = {q_A} y las transiciones siguientes:

$\delta(q_0, \mathbf{a}, \varepsilon) = (q_0, A)$ $\quad$ $\delta(q_0, \mathbf{b}, A) = (q_1, \varepsilon)$

$\delta(q_1, \mathbf{b}, A) = (q_A, B)$ $\quad$ $\delta(q_1, \mathbf{b}, B) = (q_1, \varepsilon)$

$\delta(q_0, \mathbf{\$}, \varepsilon) = (q_1, \varepsilon)$ $\quad$ $\delta(q_1, \mathbf{\$}, \varepsilon) = (q_A, \varepsilon)$

La estrategia consiste en utilizar una **MT** de dos cintas, en la primera cinta estará la cadena a ser decidida y mientras que la segunda cinta simulará a la pila, que inicialmente estará vacía. El conjunto de estados es idéntico al del **APD**:

Para las transiciones que desapilan tenemos sus equivalentes en la **MT**:

$$\delta(q_0, (\mathbf{b}, A)) = (q_1, (\mathbf{b}, \#), (R, R))$$

$$\delta(q_1, (\mathbf{b}, B)) = (q_1, (\mathbf{b}, \#), (R, R))$$

Ahora para la transición que no modifica el tamaño de la pila la transición de la **MT** no tiene movimiento en la segunda cinta, aunque si cambia el contenido de la celda:

$$\delta(q_1, (\mathbf{b}, A)) = (q_1, (\mathbf{b}, B), (R, S))$$

Mientras que para la transición que apila, se requiere utilizar un estado intermedio para desplazarse una celda hacia la izquierda sobre la cinta inferior (hacia donde crece la pila), no importando el símbolo actual de la celda, para que luego se pueda escribir el símbolo que se agrega a la cima de la pila:

$$\delta(q_0, (\mathbf{a}, \sigma)) = (p, (\mathbf{a}, \sigma), (S, L))$$

$$\delta(p, (\mathbf{a}, \#)) = (q_0, (\mathbf{a}, A), (R, S))$$

Obsérvese que el apuntador de la segunda cinta siempre indica el símbolo de la cima de la pila, que como dijimos, crece hacia la izquierda.

La cadena se acepta cuando ésta se agote y la pila esté vacía:

$$\delta(\ q_0,\ (\#,\ \#)\ ) = (\ q_A,\ (\#,\ \#),\ (S,\ S)\ )$$

$$\delta(\ q_1,\ (\#,\ \#)\ ) = (\ q_A,\ (\#,\ \#),\ (S,\ S)\ )$$

La Decidibilidad exige que siempre podamos determinar cuando una cadena no pertenece al lenguaje en cuestión, para ello podemos introducir todas las transiciones que nos conduzcan a un estado de rechazo para todas las distintas situaciones en que esto sucede, que en este ejemplo son las siguientes:

$$\delta(\ q_1,\ (\mathbf{a},\ \sigma)\ ) = (\ q_R,\ (\mathbf{a},\ \sigma),\ (S,\ S)\ )$$

$$\delta(\ q_0,\ (\mathbf{b},\ \#)\ ) = (\ q_R,\ (\mathbf{b},\ \#),\ (S,\ S)\ )$$

$$\delta(\ q_1,\ (\mathbf{b},\ \#)\ ) = (\ q_R,\ (\mathbf{b},\ \#),\ (S,\ S)\ )$$

$$\delta(\ q_0,\ (\#,\ \mathbf{A})\ ) = (\ q_R,\ (\#,\ \mathbf{A}),\ (S,\ S)\ )$$

$$\delta(\ q_1,\ (\#,\ \mathbf{A})\ ) = (\ q_R,\ (\#,\ \mathbf{A}),\ (S,\ S)\ )$$

$$\delta(\ q_1,\ (\#,\ \mathbf{B})\ ) = (\ q_R,\ (\#,\ \mathbf{B}),\ (S,\ S)\ )$$

Como enlistar todas las situaciones posibles resulta siempre muy tedioso y puede incurrirse muy fácilmente en omisiones, se suele dar por sobreentendido que si la **MT** se detiene en otro estado distinto al de aceptación, la cadena es decidida negativamente, rechazada, como si se hubiera llegado al estado de rechazo.

Todo LICN es Decidible

Para cuando tenemos un **APN**, no es conveniente tratar de obtener una **MT** equivalente que decida al mismo **LICN**, porque es posible que la **MT** así generada nunca se detenga al analizar cadenas que no pertenezcan al lenguaje, debido a todas las posibles situaciones que presenta el no-determinismo y que podrían multiplicarse hasta el infinito, lo que no probaría que es Decidible.

La técnica para demostrar que un **LICN** dado es Decidible y que evita el riesgo de que la **MT** obtenida se cicle, consiste en encontrar una **GIC** que genere al lenguaje en cuestión y demostrar que se puede construir una **MT** que decida si una cadena dada es o no posible generarla por medio de esta gramática, en un número finito de pasos, esto sería equivalente a construir una **MT** que utilice el algoritmo **CYK** con este objetivo, tal como lo vimos en el capítulo 7.

Gramáticas Sensibles al Contexto

Las Gramáticas Sensibles al Contexto son aquellas gramáticas cuyas producciones son de la forma siguiente: $u\mathbf{A}v \rightarrow uwv$, donde u, w y v son cadenas de Símbolos Terminales y No Terminales, posiblemente vacías. Este tipo de gramáticas genera una clase de lenguajes a los que se les denomina *Lenguajes Dependientes del Contexto* o también *Sensibles al Contexto*.

La producción $u\mathbf{A}v \rightarrow uwv$ es esencialmente distinta de la producción: $\mathbf{A} \rightarrow w$, en el sentido de que el No terminal **A** puede ser reemplazado por la cadena w solamente si **A** se encuentra dentro de un contexto donde le precede la cadena u y es seguida por la cadena v, en caso contrario no puede hacerse la sustitución.

Queda claro que si en todas las producciones de una gramática, u y v son ambas cadenas vacías, entonces la gramática en cuestión es *Independiente del Contexto*.

Las Gramáticas Sensibles al Contexto son muy útiles para el estudio de la formación de los Lenguajes Naturales, y que es una de las ramas de la Inteligencia Artificial.

Ejemplo

Se desea construir una Gramática Sensible al Contexto (**GSC**) que permita generar las cadenas que pertenecen al siguiente lenguaje: $L = \{ \mathbf{a}^n\mathbf{b}^n\mathbf{c}^n \mid n \geq 1 \}$.

Una forma interesante para lograrlo es por medio de la siguiente gramática:

$$S \rightarrow \mathbf{abc} \mid \mathbf{a}A\mathbf{bc}$$

$$A\mathbf{b} \rightarrow \mathbf{b}A$$

$$A\mathbf{c} \rightarrow B\mathbf{bcc}$$

$$\mathbf{b}B \rightarrow B\mathbf{b}$$

$$\mathbf{a}B \rightarrow \mathbf{aa} \mid \mathbf{aa}A$$

Observe como es que esta gramática emplea los **SNT**s A y B para agregar de manera ordenada y controlada a los símbolos **c**, **b** y **a**, tal como se ejemplifica en la siguiente derivación:

$\underline{S} \Rightarrow \mathbf{a}\underline{A}\mathbf{bc} \Rightarrow \mathbf{ab}\underline{A}\mathbf{c} \Rightarrow \mathbf{ab}\underline{B}\mathbf{bcc} \Rightarrow \mathbf{a}\underline{B}\mathbf{bbcc} \Rightarrow \mathbf{aa}\underline{A}\mathbf{bbcc} \Rightarrow \mathbf{aab}\underline{A}\mathbf{bcc} \Rightarrow \mathbf{aabb}\underline{A}\mathbf{cc} \Rightarrow$
$\mathbf{aab}\underline{b}B\mathbf{bccc} \Rightarrow \mathbf{aa}\underline{b}B\mathbf{bbccc} \Rightarrow \mathbf{a}\underline{a}B\mathbf{bbbccc} \Rightarrow \mathbf{aaabbbccc}$

Otra gramática equivalente, pero con un enfoque distinto, que nos permite generar las cadenas de este lenguaje, se puede describir inicialmente con un par de transiciones

que nos permitan ir agregando tercias de símbolos **a**, B y C, hasta llegar a la cantidad de n, pero como ya se dijo, no es posible generarlos de tal modo que queden ordenados, entonces utilizamos los **SNT**s B y C en vez de los símbolos terminales **b** y **c**, respectivamente:

$$S \rightarrow \mathbf{a}SBC \mid \mathbf{a}BC$$

A continuación introducimos una producción que reordene a los símbolos no terminales B y C que aparezcan fuera de secuencia:

$$CB \rightarrow BC$$

Ahora requerimos de una serie de producciones que nos permitan realizar las sustituciones de los Símbolos No Terminales por los terminales correspondientes, solamente si están en el orden (contexto) adecuado:

$$\mathbf{a}B \rightarrow \mathbf{ab}$$

$$\mathbf{b}B \rightarrow \mathbf{bb}$$

$$\mathbf{b}C \rightarrow \mathbf{bc}$$

$$\mathbf{c}C \rightarrow \mathbf{cc}$$

Comparemos como funciona esta gramática en la generación de la misma cadena del caso anterior: **aaabbbccc**.

$\underline{S} \Rightarrow \mathbf{a}\underline{S}BC \Rightarrow \mathbf{aa}\underline{S}BCBC \Rightarrow \mathbf{aaa}BCB\underline{CB}C \Rightarrow \mathbf{aaa}B\underline{CB}BCC \Rightarrow \mathbf{aaa}BB\underline{CB}CC \Rightarrow \mathbf{aa}\underline{\mathbf{a}}BBBCCC \Rightarrow \mathbf{aaab}\underline{\mathbf{b}}BBCCC \Rightarrow \mathbf{aaabb}\underline{\mathbf{b}}BCCC \Rightarrow \mathbf{aaabbb}\underline{\mathbf{c}}CCC \Rightarrow \mathbf{aaabbbc}\underline{\mathbf{c}}C \Rightarrow \mathbf{aaabbbccc}$

Si pretendiéramos reemplazar los terminales antes de reordenar los no terminales no se puede concluir la generación de ninguna cadena:

$\underline{S} \Rightarrow \mathbf{a}\underline{S}BC \Rightarrow \mathbf{aa}\underline{B}CBC \Rightarrow \mathbf{aab}\underline{C}BC \Rightarrow \mathbf{aabc}BC$

La producción $CB \rightarrow BC$ no parece ser sensible al contexto, de acuerdo con la forma como se definió más arriba, pero se puede comprobar fácilmente que esta producción es equivalente a un conjunto de cuatro producciones claramente sensibles al contexto:

$$\underline{C}B \rightarrow \underline{BX}B$$

$$X\underline{B} \rightarrow X\underline{Y}$$

$$\underline{X}Y \rightarrow Y$$

$$\underline{Y} \rightarrow \underline{C}$$

Autómatas Lineales Acotados

Los Lenguajes Sensibles al Contexto pueden ser reconocidos por unos dispositivos conocidos como *Autómatas Lineales Acotados* (**ALA**), los cuales son una clase particular de **MT**s, deterministas o no, que satisfacen estas dos condiciones:

- El alfabeto de la cinta contiene dos símbolos especiales **[** y **]**, conocidos como marcadores izquierdo y derecho, respectivamente, de esta forma, el contenido de la cinta se representa inicialmente como **[***w***]**.
- El **ALA** no puede moverse a la izquierda del marcador **[**, ni a la derecha del marcador **]**, así como tampoco escribir ningún otro símbolo sobre ellos, ni tampoco usar esos símbolos en cualquier otra celda intermedia.

Todo LSC es también Decidible

De esta forma, un **ALA** queda restringido a realizar todas sus operaciones únicamente sobre las celdas que ocupaba la cadena original. Finalmente, cabe señalar que dado un **ALA** cualquiera, éste siempre llega a una configuración de parada, por lo que todo **LSC** al ser reconocido por un **ALA** también es Decidible.

Gramáticas No Restringidas

Una *Gramática No Restringida* (**GNR**), también llamada *Gramática de Estructura de Frase*, es una cuádrupla G = (N, $\sum$, S, P), en la que N es un alfabeto de Símbolos No Terminales, $\sum$ es el alfabeto de Símbolos Terminales, S es el Símbolo No Terminal Inicial y P es el conjunto de Producciones de la forma: $\alpha \rightarrow \beta$, donde la parte izquierda contiene una cadena de símbolos terminales y no terminales, $\alpha \in (N \cup \sum)^* N (N \cup \sum)^*$, en la que al menos hay un símbolo no terminal, y la parte derecha está formada por cualquier cadena de símbolos terminales y no terminales $\beta \in (N \cup \sum)^*$.

Esta es la definición más amplia de gramática, incluye a todas las clases previamente mencionadas y ya no hay otra clase de gramáticas que sea superior a ésta.

A la clase de lenguajes generados por una **GNR** se le llaman *Lenguajes Recursivamente Enumerables*, o simplemente, *Lenguajes Enumerables*, dentro de los que se incluye a la clase de los *Lenguajes Decidibles*. Sin embargo, existen *Lenguajes Enumerables* que no son Decidibles, como veremos más adelante.

Primeramemente, exponemos dos casos de Lenguajes decidibles que son generados por sus respectivas **GNR**s, con el fin de entender su funcionamiento.

Ejemplo 1

Sea la Gramática no Restringida (**GNR**) dada por las siguientes producciones:

$$S \rightarrow [\mathbf{a}]$$
$$D] \rightarrow]$$
$$[\rightarrow [D$$
$$[\rightarrow \varepsilon$$
$$] \rightarrow \varepsilon$$
$$D\mathbf{a} \rightarrow \mathbf{aa}D$$

Con esta gramática podemos derivar las cadenas w_1 = **a**, w_2= **aa**, w_3 = **aaaa**, etc. (cadenas **a**s de longitud que es potencia de 2). Ahora, analicemos el comportamiento de esta gramática en la derivación de la cadena w_3 = **aaaa**:

$S \Rightarrow [\mathbf{a}] \Rightarrow [D\mathbf{a}] \Rightarrow [\mathbf{aa}D] \Rightarrow [\mathbf{aa}] \Rightarrow [D\mathbf{aa}] \Rightarrow [\mathbf{aa}D\mathbf{a}] \Rightarrow [\mathbf{aaaa}D] \Rightarrow [\mathbf{aaaa}] \Rightarrow \mathbf{aaaa}$

El lenguaje generado por esta gramática es: $L = \{ \mathbf{a}^{2^n} \mid n \geq 0 \}$, en este caso el símbolo D hace el papel de *Duplicador de símbolos* y solamente lo podemos descartar cuando haya concluido su tarea de duplicar toda la cadena al extremo derecho de la misma y es entonces cuando conviene quitar los corchetes.

Ejemplo 2

Sea la Gramática no Restringida (**GNR**) dada por las producciones:

$$S \rightarrow M]$$
$$] \rightarrow A\mathbf{a}] \mid B\mathbf{b}]$$
$$\mathbf{a}A \rightarrow A\mathbf{a}$$
$$\mathbf{a}B \rightarrow B\mathbf{a}$$
$$\mathbf{b}A \rightarrow A\mathbf{b}$$
$$\mathbf{b}B \rightarrow B\mathbf{b}$$
$$MA \rightarrow \mathbf{a}M$$
$$MB \rightarrow \mathbf{b}M$$
$$] \rightarrow \varepsilon$$
$$M \rightarrow \varepsilon$$

Observemos como funciona esta gramática en la derivación de w = **aabaab**:

$S \Rightarrow M] \Rightarrow MA\mathbf{a}] \Rightarrow \mathbf{a}M\mathbf{a}] \Rightarrow \mathbf{a}M\mathbf{a}A\mathbf{a}] \Rightarrow \mathbf{a}MA\mathbf{aa}] \Rightarrow \mathbf{aa}M\mathbf{aa}] \Rightarrow \mathbf{aa}M\mathbf{aa}B\mathbf{b}] \Rightarrow \mathbf{aa}M\mathbf{a}B\mathbf{ab}]$
$\Rightarrow \mathbf{aa}MB\mathbf{aab}] \Rightarrow \mathbf{aab}M\mathbf{aab}] \Rightarrow \mathbf{aabaab}] \Rightarrow \mathbf{aabaab}$

Esta gramática genera el siguiente lenguaje: L = { $ww \mid w \in \{ \mathbf{a}, \mathbf{b}\}^*$ }, en este caso el símbolo M marca la mitad de la cadena.

Toda MT puede simularse por alguna GNR

Todo Lenguaje Enumerable, es aceptado por alguna **MT** y puede ser generado por alguna **GNR**, para mostrar esta equivalencia, se planteará un procedimiento general que nos permita encontrar una **GNR** que simule la **MT** mediante el siguiente ejemplo:

Ejemplo

Sea la **MT** que acepta el lenguaje regular L = **a***, dada por. Q = { q_0, q_1, q_A }, Σ = { **a** }, Γ = { **a**, **1**, # }, s = q_0, F = { q_A } y las transiciones siguientes:

$$\delta(\ q_0, \mathbf{a}\ ) = (q_0, \mathbf{a}, R\ )$$
$$\delta(\ q_0, \#\ ) = (q_1, \#, R\ )$$
$$\delta(\ q_1, \#\ ) = (q_A, \mathbf{1}, L\ )$$

Primeramente analicemos las descripciones instantáneas que muestran el comportamiento de la **MT** para aceptar la siguiente cadena **aaa**:

$$q_0\mathbf{aaa} \vdash \mathbf{a}q_0\mathbf{aa} \vdash \mathbf{aa}q_0\mathbf{a} \vdash \mathbf{aaa}q_0\# \vdash \mathbf{aaa}\#q_1\# \vdash \mathbf{aaa}q_A\mathbf{\#1}$$

Vamos a construir un conjunto de reglas gramaticales que nos permitan generar invariablemente a la cadena de la configuración final: **aaa**q_A**#1** y luego por medio de diversas producciones tratar de llegar a la configuración inicial: q_0**aaa**. La razón de ir en sentido inverso es que la configuración inicial es única.

Para obtener esas reglas, obsérvese en el primer paso que lo que cambia es: q_0**a** por **a**q_0, mientras que el resto permanece invariable, si el objetivo de la gramática es moverse en el sentido inverso al descrito, resulta evidente que una producción de la forma: **a**$q_0 \rightarrow q_0$**a** realiza dicho cambio en sentido contrario. Esto es válido para cualquier transición a la derecha.

Para ejemplificar las transiciones a la izquierda, hemos definido una transición que cambia el contenido de la cinta para hacerlo más claro, en este caso la subcadena que cambia es #q_1# por q_A**#1**, y entonces la producción necesaria para nuestra gramática deberá tener la forma siguiente: q_A**#1** $\rightarrow$ #q_1#.

El lector deberá ser capaz de deducir como son las producciones para las transiciones estacionarias en caso de haberlas. Aclarado lo anterior, podemos construir una **GNR** que genere a L, por medio de los siguientes pasos:

Paso 1. **Inicialización**, donde se incluyen las producciones que nos permiten regenerar la configuración final del proceso de aceptación, por lo que se deben poder adicionar cualquier símbolo del alfabeto de la cinta Γ y el estado de aceptación q_A, en donde convenga:

$$S \rightarrow [A$$

$$A \rightarrow \mathbf{a}A \mid A\mathbf{a} \mid \#A \mid A\# \mid \mathbf{1}A \mid A\mathbf{1} \mid q_A$$

Paso 2. **Transiciones a la derecha**, como se comentó arriba, se tiene que:

$$\mathbf{a}q_0 \rightarrow q_0\mathbf{a}$$

$$\#q_1 \rightarrow q_0\#$$

Paso 3. **Transiciones a la izquierda**, ya antes comentado, tenemos:

$$q_A\mathbf{\#1} \rightarrow \#q_1\#$$

Paso 4. **Finalización**, Reemplaza por cadenas vacías los símbolos no terminales:

$$\# \rightarrow \varepsilon$$

$$[q_0 \rightarrow \varepsilon$$

Aplicando nuestra gramática para generar la cadena anterior se tiene la siguiente derivación:

$S \Rightarrow [A \Rightarrow [A\mathbf{1} \Rightarrow [A\mathbf{\#1} \Rightarrow [\mathbf{a}A\mathbf{\#1} \Rightarrow [\mathbf{aa}A\mathbf{\#1} \Rightarrow [\mathbf{aaa}A\mathbf{\#1} \Rightarrow [\mathbf{aaa}q_2\mathbf{\#1} \Rightarrow [\mathbf{aaa}\#q_1\# \Rightarrow [\mathbf{aaa}q_0\#\# \Rightarrow [\mathbf{aa}q_0\mathbf{a}\#\# \Rightarrow [\mathbf{a}q_0\mathbf{aa}\#\# \Rightarrow [q_0\mathbf{aaa}\#\# \Rightarrow \mathbf{aa}\#\# \Rightarrow \mathbf{aaa}$

Jerarquía de Chomsky

Chomsky clasificó a las gramáticas en cuatro clases básicas; las gramáticas más simples son las gramáticas del tipo 3, que corresponden a las *Gramáticas Regulares*, las cuales generan a los Lenguajes Regulares y éstos a su vez pueden ser reconocidos por los Autómatas Finitos, tal como se trató en los capítulos iniciales de esta obra.

Un nivel más arriba, siguen las gramáticas del tipo 2, que corresponden a las que ya conocemos como *Gramáticas Independientes del Contexto*, las cuales generan los Lenguajes Independientes del Contexto y que pueden ser reconocidos por medio de

los Autómatas de Pila No Deterministas, como ya se abordó en los capítulos respectivos.

A continuación, siguen las gramáticas del tipo 1, que podemos identificar con las *Gramáticas Sensibles al Contexto*, a las que Chomsky les impone la condición de que todas las producciones en este tipo de gramáticas deben ser de la forma $\alpha \rightarrow \beta$, y deben cumplir que $|\alpha| \leq |\beta|$, esta definición es casi equivalente a la condición expuesta al inicio de este capítulo, con la salvedad de que con esta restricción, Chomsky excluye de este tipo de gramáticas a todas aquellas que generen lenguajes que contienen a la cadena vacía. Siendo estrictos, este es un impedimento para que esta clase contenga completamente a todas las gramáticas regulares e independientes del contexto, para salvar este inconveniente, basta considerar de manera excepcional a la producción $S \rightarrow \varepsilon$, para los lenguajes que tengan a la cadena vacía.

Finalmente tenemos a las gramáticas del tipo 0, que corresponden a las *Gramáticas No Restringidas*, también conocidas como las *Gramáticas Irrestrictas* o *Gramáticas de Estructura de Frase*.

A los lenguajes generados por estas gramáticas se les llama *Lenguajes Recursivamente Enumerables*, o simplemente *Lenguajes Enumerables* y pueden ser aceptados, pero tal vez no decididos, por alguna Máquina de Turing.

Lenguajes Enumerables

En el capítulo anterior mencionamos que una **MT** podía decidir si una cadena pertenecía o no a un lenguaje. Sin embargo, es posible diseñar las **MT**s para que solamente se detengan en el caso que una cadena pertenezca a un determinado lenguaje y se ciclen indefinidamente cuando no, a esta situación se le denomina *Aceptar* a una cadena, incluso con lenguajes decidibles.

En la siguiente figura, se muestran dos representaciones equivalentes de **MT**s que aceptan a un lenguaje formado por las cadenas del alfabeto $\sum$ = { **x**, **y** } que contienen al menos una **x**, la primera **MT** decide, la segunda sólo acepta.

Figura 11.1

Mientras que en la Figura 11.2 se muestra una **MT** que sólo acepta el lenguaje formado por todas las cadenas que *no* empiezan por ***x***.

El punto es que, aunque no hayamos visto un ejemplo aún, existen *Lenguajes Enumerables* que no son, ni pueden ser, Decidibles; sólo son aceptados por alguna **MT**, la cual aceptará a toda cadena que pertenezca al lenguaje en cuestión, pero si la

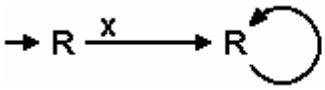


Figura 11.2

cadena que se analiza no pertenece a dicho lenguaje, entonces probablemente la **MT** se cicle indefinidamente y no es posible modificarla para forzar a que se detenga al analizar las cadenas con las que se cicla, ni tampoco es posible encontrar otra **MT** equivalente que no se cicle con aquellas cadenas que deberían ser rechazadas.

Lo desafiante de este problema consiste en que no es posible determinar con certeza en que casos la **MT** se va a ciclar indefinidamente o en que casos se va a tardar demasiado porque esa cadena requiere de un análisis muy complejo, pero es cuestión de esperar más tiempo para que se detenga finalmente.

Esta situación es conocida como el *Problema de Detención* (o Problema de Parada) de las **MT**, el cual retomaremos nuevamente más adelante, dada su importancia. Por ahora, la conclusión que surge de todo lo anterior, es que existe una correspondencia exacta entre las **MT**s, las **GNR**s y los Lenguajes Enumerables.

En la siguiente figura se resume todo lo que hemos hablado hasta ahora sobre las diferentes clases de lenguajes y los distintos dispositivos empleados para su reconocimiento, incluso se hace referencia a que existen lenguajes que no son Enumerables y para los cuales no existe nada que permita reconocerlos.

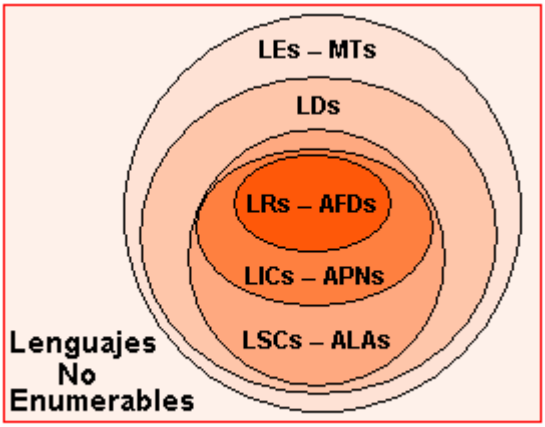


Figura 11.3

La Máquina de Turing como Enumerador

Ya hemos visto que una **MT** puede utilizarse para el reconocimiento de lenguajes, así como también para el cálculo de funciones sobre enteros positivos, un tercer uso de las **MT** consiste en emplearlas como un dispositivo generador de cadenas.

Considere que se puede construir una **MT** que enumere todas las cadenas que pertenecen a un lenguaje dado, para ello imaginemos que se tiene una **MT** multicinta en la que haya una cinta de salida, donde escriba las cadenas que pertenecen a un determinado lenguaje y que únicamente avance hacia la derecha, de modo que una vez escrito el símbolo no puede ser modificado, además se emplea un símbolo especial como por ejemplo el # para separar las distintas cadenas generadas.

El lenguaje generado por la máquina M es el conjunto de cadenas $w \in \Sigma^*$ que aparecen en la cinta de salida separadas por el símbolo especial # y si recordamos que una Gramática es un conjunto de reglas para generar un lenguaje determinado, podemos constatar que en este caso, la **MT** se estará comportando de manera equivalente a una **GNR**.

No se requiere que las cadenas sean generadas una sola vez, ni tampoco en algún orden determinado, pero se puede verificar que la clase de lenguajes generados por alguna **MT** corresponde a la de los *Lenguajes Enumerables*.

Sin embargo, si podemos hacer que el conjunto de cadenas de cierto lenguaje, sea generado de una manera ordenada; generando primero las cadenas de menor tamaño y luego avanzando en orden creciente en extensión; y para las cadenas que sean del mismo tamaño, procedemos a generarlas de acuerdo a un orden alfabético preestablecido de los símbolos que las conforman. Si esto es posible, se dice que ese lenguaje se encuentra en un *Orden Canónico*.

Si un lenguaje puede ser generado en un Orden Canónico, entonces dicho lenguaje pertenece a la clase de los *Lenguajes Recursivos*, que es otro nombre con el que también se conoce a la clase de los *Lenguajes Decidibles*.

Ejemplo 1

Un ejemplo muy sencillo de un lenguaje que puede ser generado en orden canónico es el lenguaje universal: L = { **0**, **1** }* = { ε, **0**, **1**, **00**, **01**, **10**, **11**, **000**, **001**, … }.

Ejemplo 2

Otro ejemplo más interesante es la generación de pares ordenados de la forma (**i**, **j**), pues en este caso, si pretendemos generarlos en el orden siguiente: (**1**,**1**), (**1**,**2**), (**1**,**3**), (**1**,**4**), (**1**,**5**), …, jamás se podrán generar las parejas que contengan un valor de **i** $\neq$ **1**.

Pero si seguimos una estrategia distinta para generar las parejas (**i**, **j**), consistente en generarlas de acuerdo al orden creciente de la suma **i** + **j**, y ordenando las parejas que tengan una misma suma en orden creciente del valor de la **i**, entonces es posible generar de manera previsible cualquier pareja de números enteros.

La secuencia en que se generen sería como sigue: (**1**,**1**), (**1**,**2**), (**2**,**1**), (**1**,**3**), (**2**,**2**), (**3**,**1**), (**1**,**4**), (**2**,**3**), (**3**,**2**), (**4**,**1**), …. , de esta forma, cualquier pareja de la forma (**i**, **j**) puede ser generada y hasta es posible determinar el lugar que ocupará en la secuencia, utilizando la fórmula: (**i** + **j** – 1)(**i** + **j** – 2)/2 + **i**.

De esta manera, se puede ver que es posible establecer una correspondencia biunívoca entre los pares ordenados de enteros positivos con el conjunto de los números naturales, lo que significa que ambos conjuntos son *Equinumerosos*.

Aún hay más, con la técnica anterior se puede confirmar lo que establece el Teorema de Georg Cantor, que dice que los números racionales son tan numerosos como los enteros, simplemente porque cualquier número racional se puede representar como la fracción de dos enteros, de la misma forma que las parejas de enteros ya tratadas, a la clase de conjuntos infinitos *Equinumerosos* con los Números Naturales, se dice que tienen un tamaño Aleph cero: $\aleph_0$.

Tesis de Church

Esta tesis establece que la construcción de una Máquina de Turing que siempre para, es equivalente al concepto formal de un *Algoritmo*, entendido como un proceso, por medio del cual se puede realizar una tarea en un número finito de pasos, y que en ambos casos, lo importante es que siempre se alcanza un resultado.

Esta noción formal nos ha permitido demostrar, en muchos casos, la inexistencia de algunos algoritmos que hasta hace unos cien años no se había podido determinar si existían o no, simplemente porque carecían de las herramientas formales para ello; por ejemplo, en 1900, Hilbert planteó el desafío de 23 problemas matemáticos, entre ellos, el décimo, concerniente a hallar un algoritmo para saber si un polinomio dado

tiene una raíz entera, no fue sino hasta 1970, que se probó formalmente que tal algoritmo no existe.

Replanteemos el problema de la siguiente manera; supongamos que se construye una **MT** que permita reconocer el siguiente lenguaje:

$$L_P = \{ p \mid p \text{ es un polinomio que tiene una raíz entera} \}$$

La manera como debe trabajar esta **MT** requerirá que evalúe al polinomio p sobre la sucesión de enteros: 0, 1, -1, 2, -2, 3, -3, …. Si la raíz existe, eventualmente la encontrará y la ejecución de la **MT** terminará; pero si no existe esa raíz, la **MT** seguirá buscándola para siempre. Evidentemente esta **MT** es una *aceptadora*, pero no es una *decididora*, por lo tanto, concluimos que L_P es un Lenguaje Enumerable pero que no es Decidible.

Por lo tanto, Church llegó a la siguiente conclusión: Un problema es computable si y solo si es Decidible por alguna Máquina de Turing si y solo si existe un algoritmo que permita resolverlo, Además, no se ha probado que exista algún modelo teórico que sea más poderoso que una **MT** y es muy poco probable que pudiera existir, ¡Porque de haberlo, se revolucionaría el diseño de computadoras en su totalidad!

Preguntas

a) ¿La unión de dos lenguajes Decidibles es un Lenguaje Decidible?

b) ¿La concatenación de dos lenguajes Decidibles es un Lenguaje Decidible?

c) Si no existe un algoritmo para resolver un problema dado, significa que ¿no será posible construir una **MT** que decida sobre éste?

d) Un lenguaje que no es Decidible puede ser que tampoco sea aceptado por ninguna **MT**?

Ejercicios Capítulo 11

11.1 Obtener una **GNR** para generar cada uno de los siguientes lenguajes:

a) $L = \{ \mathbf{a}^i\mathbf{b}^j\mathbf{c}^k \mid 0 \leq i < j < k \}$

b) $L = \{ \mathbf{a}^n\mathbf{b}^n\mathbf{c}^n\mathbf{d}^n \mid n \geq 0 \}$

c) $L = \{ \mathbf{a}^n\mathbf{b}^n\mathbf{a}^n\mathbf{b}^n \mid n \geq 0 \}$

d) $L = \{ \mathbf{a}^{3^k} \mid k \geq 0 \}$

e) $L = \{ www \mid w \in \{ \mathbf{a}, \mathbf{b} \} \}$

11.2 Encontrar una **MT determinista** que decida el lenguaje aceptado por el **AFN** mostrado en el diagrama de la figura 11.4:

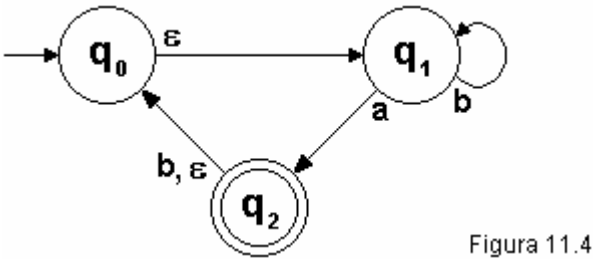


Figura 11.4

11.3 Construya una **MT** que decida el lenguaje aceptado por el **AFD** mostrado en el siguiente diagrama:

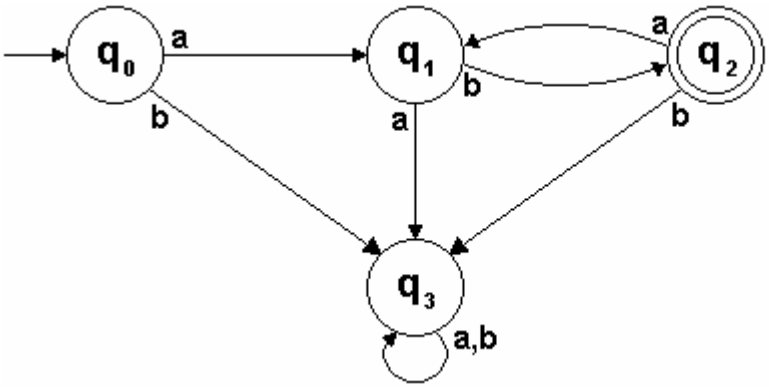


Figura 11.5

11.4 Construya una Máquina de Turing que decida sobre el lenguaje aceptado por el siguiente **APD condicionado**: Q = { q_0, q_1 }, s = q_0, F = { q_1 }, Σ = { **a**, **b** }, Γ = { A, B, Z } y las transiciones siguientes:

$\Delta(q_0, \mathbf{a}, Z) = (q_0, AZ)$ $\quad$ $\Delta(q_0, \mathbf{a}, A) = (q_0, AA)$

$\Delta(q_0, \mathbf{b}, Z) = (q_0, BZ)$ $\quad$ $\Delta(q_0, \mathbf{b}, B) = (q_0, BB)$

$\Delta(q_0, \mathbf{b}, A) = (q_0, \varepsilon)$ $\quad$ $\Delta(q_0, \mathbf{a}, B) = (q_0, \varepsilon)$

$\Delta(q_0, \varepsilon, Z) = (q_1, \varepsilon)$

11.5 Construir una **GNR** que genere el lenguaje aceptado por la **MT** cuyas transiciones son, pruebe la gramática generando la cadena **aabbaa**:

$\delta(\ q_0, \mathbf{a}\ ) = (q_1, \mathbf{x}, R\ )$ $\quad$ $\delta(\ q_1, \mathbf{a}\ ) = (q_0, \mathbf{x}, R\ )$

$\delta(\ q_0, \mathbf{b}\ ) = (q_2, \mathbf{x}, R\ )$ $\quad$ $\delta(\ q_2, \mathbf{b}\ ) = (q_0, \mathbf{x}, R\ )$

$\delta(\ q_0, \#\ ) = (q_3, \#, L\ )$

Capítulo

12

Resolubilidad

Se define Resolubilidad y Se establece una correlación entre los Lenguajes Decidibles y los Problemas Resolubles por medio de Máquinas de Turing. Se plantean ejemplos de problemas irresolubles.

Definición de Resolubilidad

En los capítulos anteriores se mencionó que es posible diseñar una **MT que decida** si una cadena pertenece o no a un determinado lenguaje L, de tal forma que si finaliza en un estado de aceptación, la cadena pertenece a L, pero si termina en un estado de rechazo, la cadena no pertenece a L. A los lenguajes que pueden ser decididos por medio de una **MT**, se les conoce también como *Lenguajes Recursivos*.

Ahora veremos como es que dado un determinado problema, lo podemos replantear de tal forma que se interprete como un Lenguaje y por tanto, que podamos construir una **MT** que acepte dicho lenguaje. A los problemas que puedan replantearse como Lenguajes Decidibles les llamamos *Problemas Resolubles*, mientras que los que corresponden a Lenguajes Enumerables se les identificará como *Irresolubles*.

Reformulación de Problemas

A fin de entender mejor como se hace la reformulación de problemas, recordemos el modelo de la Máquina Universal de Turing vista en el capítulo 10, e imaginemos que es posible diseñar un dispositivo capaz de emular el comportamiento del modelo matemático de cualquier problema. Así es como haremos ahora para replantear varios de los problemas relacionados con autómatas y gramáticas:

¿Un AFD dado acepta a una cadena w?

Para concebir a un dispositivo que nos responda a la pregunta: Sea B, un **AFD** cualquiera, ¿éste acepta a una cadena dada w?, primero necesitamos definir al lenguaje siguiente:

$$L_{AFD} = \{ \langle B, w \rangle \mid B \text{ es un } \textbf{AFD} \text{ que acepta la cadena } w \}$$

Entonces, la pregunta de que si B, un **AFD**, acepta a una cadena w se replantea como la pregunta de que ¿la cadena $\langle B, w\rangle \in L_{AFD}$?

Como se trata de un **AFD**, siempre es posible responder la pregunta si B acepta o no a la cadena w, por lo tanto siempre es posible decir si $\langle B, w\rangle$ pertenece o no a L_{AFD}, por lo que claramente este lenguaje es Decidible.

¿Qué pasaría si en vez de un **AFD** se tratara de un **AFN**? El lenguaje L_{AFN} también es Decidible ya que conocemos un procedimiento para encontrar un **AFD** equivalente a un **AFN** dado.

¿Una GR dada genera a una cadena w?

Ahora queremos concebir un dispositivo que nos responda a la pregunta: Dada G, una **GR**, ¿ésta puede generar a una cadena dada w?, para ello, definimos el lenguaje siguiente:

$$L_{GR} = \{ \langle G, w\rangle \mid G \text{ es una } \textbf{GR} \text{ que genera la cadena } w \}$$

Dado que siempre es posible construir un **AFD** que acepte el mismo lenguaje que es generado por una Gramática Regular dada, entonces el lenguaje L_{GR} es Decidible.

Dado un AFD, ¿acepta al Lenguaje Vacío?

Si deseamos una forma para determinar si un **AFD** dado acepta o no al lenguaje vacío, podemos definir al lenguaje:

$$V_{AFD} = \{ \langle A\rangle \mid A \text{ es un } \textbf{AFD} \text{ donde } L(A) = \varnothing \}$$

Para responder esto, basta con comprobar que el **AFD** no tiene ningún estado de aceptación que sea accesible desde el estado inicial, por tanto V_{AFD} es Decidible.

Dados dos AFDs, ¿son equivalentes?

Para replantear el problema de determinar si dos **AFD**s cualesquiera son equivalentes, definimos el lenguaje siguiente:

$$E_{AFD} = \{ \langle A, B\rangle \mid A \text{ y } B \text{ son dos } \textbf{AFD}\text{s tales que } L(A) = L(B) \}$$

Para probar esto, basta con construir un tercer **AFD** C, tal que acepte el lenguaje que resulte de la diferencia simétrica de los lenguajes aceptados por A y B, es decir que: $L(C) = L(A) \oplus L(B)$. Si resulta que $L(C) = \varnothing$, problema que es Resoluble, entonces esto implica que L(A) = L(B), por lo tanto, también E_{AFD} es Decidible.

¿Una GIC dada genera a una cadena w?

Ahora queremos un dispositivo que nos responda a la pregunta ¿Dada G un **GIC**, ésta puede generar a una cadena dada w?, entonces definimos el lenguaje:

$$L_{GIC} = \{ \langle G, w \rangle \mid G \text{ es una } \textbf{GIC} \text{ que genera la cadena } w \}$$

Este problema es más interesante, pues si pretendemos construir una **MT** que pruebe todas las posibles derivaciones, el problema aparenta ser No Decidible, pero, si encontramos una gramática equivalente, en la forma Normal de Chomsky, se puede probar que si |w| = n, entonces la derivación más larga posible consta de 2n – 1 pasos, y por tanto el lenguaje L_{GIC} si es Decidible.

Dada una GIC, ¿Genera al Lenguaje Vacío?

Si deseamos determinar si G una **GIC** cualquiera, genera al lenguaje vacío, podemos definir al lenguaje:

$$V_{GIC} = \{ \langle G \rangle \mid G \text{ es una } \textbf{GIC} \text{ donde } L(G) = \varnothing \}$$

En este caso, no es factible probar a todas y cada una de las cadenas posibles para ver que ninguna es generada por G, en cambio, es viable determinar si hay Símbolos No Terminales que generan solamente Símbolos Terminales; y después determinar si hay alguna derivación posible hacia esos Símbolos No Terminales desde el Símbolo Inicial, por lo que queda claro que el lenguaje V_{GIC} es Decidible.

Propiedades de los Lenguajes Decidibles

El complemento de un **LD** también es Decidible

Para probar esta propiedad basta con suponer que M_1 es una **MT** que decida el lenguaje $L(M_1)$, entonces es posible construir una máquina M_2, que decida al lenguaje $L(M_2) = L^C(M_1)$, simplemente complementando las salidas de M_1, si la respuesta de M_1 es afirmativa, entonces la de M_2 será negativa y viceversa, tal como se ilustra en la Figura 12.1:

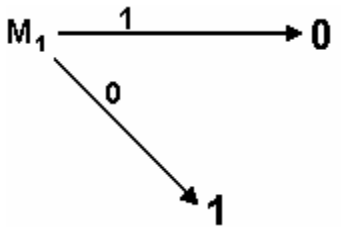


Figura 12.1

Al lector le debe resultar fácil demostrar las siguientes propiedades de los **LD**s:

- La unión de dos Lenguajes Decidibles también es un **LD**.
- La concatenación de dos Lenguajes Decidibles también es un **LD**.
- La Cerradura de Kleene de un Lenguaje Decidible también es un **LD**.
- La intersección de dos Lenguajes Decidibles también es un **LD**.

Problema de la Detención

Hemos visto que si un problema codificado corresponde a un Lenguaje Recursivo, entonces es Resoluble, porque existe un algoritmo capaz de responder *Si* o *No* en todos los casos posibles, pero si tal algoritmo no existe, entonces el problema es Irresoluble, ya que habrá instancias en las que no sea capaz de responder *Si* o *No*, este es el caso de un Lenguaje Enumerable y que no sea Recursivo, para el que habrá algunas cadenas con las cuales la **MT** que lo acepta nunca se detiene. Este es el *Problema de la Detención*, que nos conduce a dividir los problemas en dos clases: los que son computables y los que no lo son.

Con objeto de probar que existe un lenguaje que sea Enumerable, pero que no sea Recursivo, vamos a suponer que esto no es cierto, y que existe, por tanto, una Super Máquina de Turing (**SMT**) que es capaz de determinar siempre si M, una **MT** dada, va a poder aceptar o no a una cadena cualquiera *w*. De esta manera, el problema de la Decidibilidad de cualquier **MT** lo reducimos a la prueba e que esta **SMT** es Decidible.

Por lo tanto, podemos empezar por considerar que la **SMT** debe aceptar al lenguaje siguiente:

$$L_{SMT} = \{ \langle M, w \rangle \mid M \text{ acepta a } w \}$$

Más aún, el resolver el problema de que si L_{SMT} es Decidible, se puede reducir a la demostración de que el lenguaje $L_M = \{ \langle M \rangle \mid M \text{ acepta a } \langle M \rangle \}$ es Decidible, con la ventaja de que es fácil verificar que L_M es Enumerable, ya que el conjunto de todas las **MT** posibles es un conjunto contable, bastaría con listar en orden canónico a todas las cadenas que correspondan a una codificación válida de $\langle M \rangle$.

Ahora bien, si L_M fuera Decidible, el lenguaje complemento $L_{M'}$ también debería ser Decidible, como lo mostramos anteriormente, es decir, si existe M_0 que decida a L_M, deberá existir M_1 que decida al lenguaje: $L_{M'} = \{ \langle M \rangle \mid M \text{ no acepta a } \langle M \rangle \}$.

Entonces, si suponemos que $\langle M_1 \rangle$ pertenece a $L_{M'}$, debería de cumplirse que M_1 no acepta a la cadena $\langle M_1 \rangle$, y por lo tanto, $\langle M_1 \rangle$ no puede pertenecer a $L_{M'}$, con lo que tenemos una contradicción, esto demuestra que nuestra suposición es inválida, y por tanto, $L_{M'}$ no puede ser Decidible y en consecuencia L_M tampoco es Decidible.

De todo lo anterior, se concluye que tanto L_{SMT} como L_M son Lenguajes Enumerables no Decidibles y por consecuencia, se prueba que el Problema de la Detención es Irresoluble.

Diagonalización

Con objeto de encontrar un lenguaje que no sea Enumerable, vamos a imaginar una matriz, en la cual ordenamos todas las posibles cadenas formadas a partir del alfabeto $\sum$ = { **0**, **1** }, encabezando los renglones, y a todas las codificaciones de Máquinas de Turing encabezando las columnas, de modo que cada elemento de la matriz es de la forma (w_i, M_j), cuyo valor es **1**, si la i-ésima cadena es aceptada por la j-ésima **MT**, y es **0** en caso contrario.

Dicha matriz se muestra en la figura 12.2, y sobre la cual, nos concentraremos solamente en los elementos (w_i, M_i), que se encuentran sobre la diagonal principal, en un proceso que se denomina *Diagonalización*:

	M_1	M_2	M_3	M_4	⋯
w_1	1	0	1	0	
w_2	0	0	1	1	⋯
w_3	1	0	0	1	
w_4	0	1	0	1	
⋮		⋮			⋱

Figura 12.2

A partir de los elementos de la diagonal principal de esta matriz construyamos el siguiente lenguaje que llamaremos L_d, definido de la siguiente manera:

$$L_d = \{ w_i \mid (w_i, M_i) = 0 \}$$

De esta manera nos aseguramos que ninguna **MT** acepta al lenguaje L_d, ya que si suponemos que existe M_d, una **MT** que acepta al lenguaje L_d, entonces se debe

cumplir que para toda $w_i \in L_d$, se tiene que $(w_i, M_i) = 0$, por lo que M_d no puede ser ninguna de las M_i, esto significa que existe $w_d \notin L_d$, tal que $(w_d, M_d) = 1$, pero esto indicaría que w_d es aceptada por M_d, es decir que $w_d \in L_d$, lo que contradice todo lo anterior, por lo tanto, no es posible que exista tal **MT** y entonces se concluye que L_d es un lenguaje No Enumerable.

Aunque resulta difícil describir e identificar a un lenguaje No Enumerable, ya que no tiene bases gramaticales, se sabe que existe una infinidad de ellos, porque el conjunto de todos los lenguajes es incontable, equiparables con el conjunto de los números reales, de los que se dice que tienen tamaño Aleph 1: $\aleph_1$, en contraste con el conjunto de las **MT**s, que, como se dijo, es numerable (de tamaño $\aleph_0$).

Propiedades de los Lenguajes Enumerables

Las propiedades de los **LE**s, más interesantes son las siguientes:

- La unión de dos Lenguajes Enumerables L_1 y L_2, es un Lenguaje Enumerable.
- Si L es un Lenguaje Enumerable y no Recursivo, entonces L^C no es un Lenguaje Enumerable, al que se le llama Lenguaje Co-Enumerable.
- Si L y L^C son ambos Lenguajes Enumerables entonces L y L^C son ambos Lenguajes Recursivos (Decidibles).

Reducibilidad

La Reducibilidad es una técnica para demostrar la irresolubilidad de determinados problemas, a partir de su semejanza con problemas que ya han sido probados como irresolubles. Con la reducibilidad se busca reemplazar un problema por otro equivalente más simple, por ejemplo, el problema de llegar a un Museo en una ciudad desconocida se reduce al problema de conseguir un mapa.

Ahora ya sabemos que el problema de detención de las **MT**s es irresoluble, puesto que demostramos que L_{SMT} es no Decidible, mostrando primero que es equivalente a resolver el problema de L_M, el cual encontramos que es Enumerable no Decidible.

Igualmente podemos verificar que el lenguaje $L_{M'}$ es equivalente al lenguaje L_d, por lo que ambos lenguajes son No Enumerables.

La semejanza de L_{SMT} con otros problemas se puede utilizar para demostrar su irresolubilidad, como se ejemplifica a continuación:

Determinar si una MT acepta ∅

Definamos ahora el siguiente lenguaje, que nos permite determinar si M una **MT** dada acepta al lenguaje vacío:

$$V_M = \{ \langle M \rangle \mid L(M) = \varnothing \}$$

Y a su complemento:

$$V_{M'} = \{ \langle M \rangle \mid L(M) \neq \varnothing \}$$

Para ello probaremos primero que $V_{M'}$ es un **LE**, por lo que deberá existir M, una **MT** que acepte los códigos de todas las **MT** que aceptan Lenguajes no vacíos. Dada $\langle M_i \rangle$ como entrada y utilizando un generador de parejas (j, k), podemos verificar que la j-ésima cadena es aceptada por M_i en k pasos, y por lo tanto, M acepta a $\langle M_i \rangle$.

Ya demostramos que $V_{M'}$ es un **LE**, ahora supongamos que es un **LD**, entonces el lenguaje L_M también tendría que ser decidible, porque el problema de demostrar que $V_{M'}$ es Decidible se reduce a que L_M también lo sea, y como ya se probó que no lo es, entonces, $V_{M'}$ no es un **LD**. Adicionalmente podemos concluir que V_M no es un **LE**.

Otros problemas irresolubles

Los siguientes problemas relativos a Máquinas de Turing son irresolubles, porque todos ellos se pueden reducir al hecho probado de que L_M no es Decidible:

- Dada M una **MT** determinada y una cadena *w*, ¿M acepta a *w*?
- Dada M una **MT** cualquiera, ¿M acepta la cadena vacía?
- Dada M una **MT** cualquiera, ¿Existe alguna cadena *w*, que M acepte?
- Dada M una **MT** cualquiera, ¿M se detiene para cada cadena *w*?
- Dadas M_1 y M_2 dos **MT**s cualesquiera, ¿ambas aceptan al mismo lenguaje?
- Dada M una **MT**, ¿Es L(M) un **LR**?, ¿Es L(M) un **LIC**?, ¿Es L(M) un **LD**?

Los siguientes problemas relativos a Gramáticas son irresolubles y se pueden reducir a algunos de los problemas sobre **MT** anteriores:

- Dada G una gramática cualquiera y una cadena *w*, ¿$w \in L(G)$?
- Dada G una gramática determinada y una cadena *w*, ¿$w \in L(G)$?
- Dadas G_1 y G_2 dos gramáticas cualesquiera, ¿$L(G_1) = L(G_2)$?
- Dada G una gramática cualquiera, ¿Se puede determinar si $L(G) = \varnothing$?

Teorema de Rice

De lo dicho anteriormente, se concluye que ninguna propiedad no trivial de las **MT**s es Decidible o sea que todos los problemas no triviales sobre **MT**s son *Irresolubles*. Esto queda plasmado en el siguiente teorema, debido a Rice:

Si la Propiedad C es un subconjunto propio no vacío del conjunto de los lenguajes Enumerables, entonces C es irresoluble si para M, una dada **MT**, se cumple $L(M) \in C$.

Si suponemos que existe R_C la **MT** que decida a C y construyamos a M_0 tal que si M acepta a *x*, entonces $L(M_0) \in C$, si M no acepta a *x*, entonces $L(M_0) \notin C$. Entonces si fuera posible que R_C decida a $L(M_0)$, se reduce a que L_{SMT} pueda decidir a $\langle M, x \rangle$, lo cual se probó que es Irresoluble.

El problema de Correspondencia de Post

Un Sistema de Correspondencia de Post (**SCP**) está compuesto de tres elementos, primero un alfabeto Σ y dos conjuntos de cadenas de Σ^+ del mismo tamaño: A y B, identificados como: $A = \{ a_1, a_2, \ldots , a_k \}$ y $B = \{ b_1, b_2, \ldots , b_k \}$.

Una solución del **SCP** es una secuencia de índices $i_1, i_2, \ldots , i_n$, para los cuales se cumple que existe una cadena $w \in \Sigma^+$, tal que:

$$w = a_{i_1}a_{i_2} \ldots a_{i_n} = b_{i_1}b_{i_2} \ldots b_{i_n}$$

El problema de Correspondencia de Post (**PCP**) consiste en determinar si un **SCP** dado tiene o no una solución. Se puede probar que el **PCP** es irresoluble, porque si fuera resoluble implicaría que también el problema de Detención de las **MT** tendría que ser resoluble, lo que ya se mostró que no es posible.

Una forma de visualizar el **SCP** si agrupamos las cadenas de A y B en bloques semejantes a unas fichas del dominó, de tal forma, que si podemos juntar una serie de fichas que tanto arriba como abajo formen la misma cadena, tendremos una solución del **SCP**:

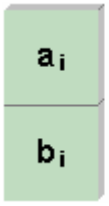


Figura 12.3

Ejemplo 1

Sea el **SCP** dado por Σ = { **a**, **b** }, A = { **a**, **abaaa**, **ab** } y B = { **aaa**, **ab**, **b** }, entonces el **SCP** anterior se puede representar así:

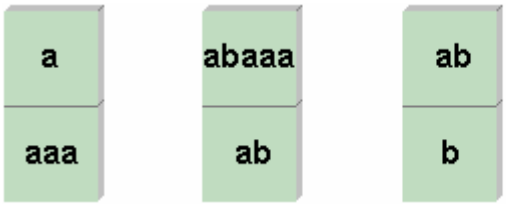


Figura 12.4

De este modo, se puede ver al problema de **SCP** como la manera de colocar bloques de los tres tipos anteriores uno al lado del otro hasta que la cadena formada por la parte superior coincida con la cadena formada por la parte inferior, por lo tanto, la respuesta al problema anterior sería:

Figura 12.5

La solución de este problema viene dada por los índices: $i_1 = 2$, $i_2 = 1$, $i_3 = 1$ y $i_4 = 3$, dado que la cadena: $w = a_2a_1a_1a_3 = b_2b_1b_1b_3 =$ **abaaaaaab**.

Ejemplo 2

Sea el **SCP** dado por Σ = { **a**, **b** }, A = { **ab**, **aba**, **baa** } y B = { **aba**, **baa**, **aa** }, Otra representación interesante de este tipo de problemas es por medio de piezas de un rompecabezas, tal como se ejemplifica en la siguiente figura:

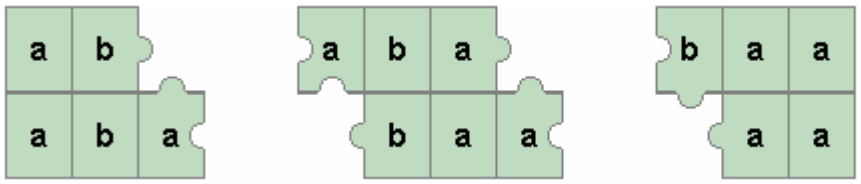


Figura 12.6

Se puede observar que el primer índice forzosamente debe ser $i_1 = 1$, ya que es el único caso que inicia con la misma letra en ambas celdas, a continuación, la única posibilidad consiste en colocar el segundo bloque, ya que debe iniciar con **a** en la parte superior y la cadena inferior debe iniciar con la segunda letra de la cadena superior, quedando así:

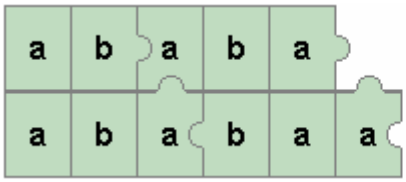


Figura 12.7

Para continuar, se puede observar que el único bloque posible es nuevamente el segundo y este razonamiento se cicla de manera infinita, ya que no hay manera de terminar y por lo tanto este problema no tiene solución.

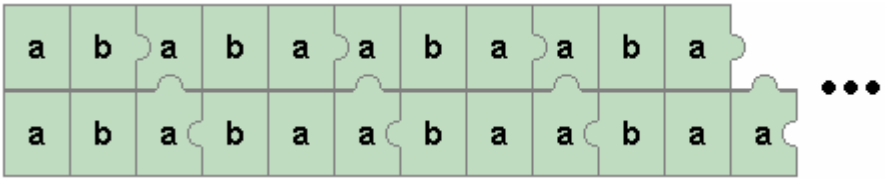


Figura 12.8

Ejemplo 3

Sea el **SCP** dado por Σ = { **0**, **1** }, A = { **100**, **0**, **1** } y B = { **1**, **100**, **00** }, a pesar de que los ejemplos anteriores son sumamente sencillos, no siempre es así, en este caso tenemos un ejemplo un poco más difícil, la representación por medio de piezas de rompecabezas no es simple, como se aprecia en la gráfica de la solución:

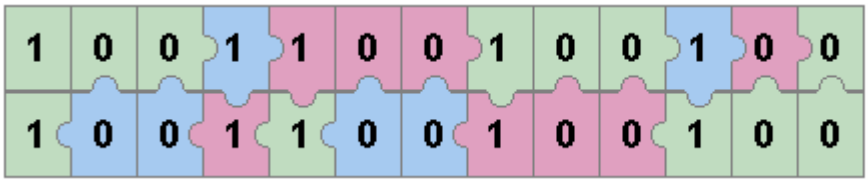


Figura 12.9

Se han documentado casos donde la solución es aún muchísimo más compleja.

El problema de intersección vacía de dos GICs.

Este problema consiste en que dadas dos **GIC**s, G_1 y G_2, determinar si la intersección de los lenguajes generados por ambas es vacía: $L(G_1) \cap L(G_2) = \varnothing$; este problema no tiene solución.

Para probar esto, supongamos el caso en el que $L(G_1) = \{ w\mathbf{c}w^R \mid w \in \{a,b\}^* \}$ y que $L(G_2) = \{ a_{i_1}a_{i_2} \ldots a_{i_n}\mathbf{c}b_{i_n}{}^R \ldots b_{i_2}{}^R b_{i_1}{}^R \mid A = \{ a_1, a_2, \ldots, a_k \}$ y $B = \{ b_1, b_2, \ldots, b_k \}$ es un **SCP**}, entonces se puede verificar que $L(G_1) \cap L(G_2) = \{ w\mathbf{c}w^R \mid w$ es solución del **SCP**}, por lo tanto, resolver $L(G_1) \cap L(G_2) = \varnothing$ es equivalente a resolver el **PCP**, por consiguiente, este problema también es irresoluble.

El problema de Ambigüedad para GICs.

Este problema consiste en que dada G una **GIC**, ¿Es posible determinar si G es *Ambigua*? Este problema no tiene solución, en la prueba de este teorema también se utiliza el resultado del **PCP**.

Preguntas

a) ¿La diferencia de dos lenguajes Decidibles es un Lenguaje Decidible?

b) Si el lenguaje L_d es No Enumerable, ¿Cómo debe ser el lenguaje L_d^C?

c) ¿Es posible que L y L^C sean ambos Lenguajes No Enumerables?

d) Del Teorema de Rice se deduce entonces que ¿no podemos saber nada respecto de ninguna Máquina de Turing?

Ejercicios Capítulo 12

12.1 Probar que los siguientes problemas son resolubles (Decidibles), Sea un **AFD** M y una cadena $w \in \Sigma^*$:

a) ¿Hay una cadena $uw \in L(M)$, para algún $u \in \Sigma^*$?

b) ¿ Hay una cadena $wu \in L(M)$, para algún $u \in \Sigma^*$?

c) ¿Hay una cadena $uwv \in L(M)$, para algunos $u, v \in \Sigma^*$?

d) Sea un **AFD** M y una expresión regular r, ¿Es L(M) = L(r)?

e) Sea una **GIC** G, ¿$\varepsilon \in L(G)$?

12.2 Realizar un análisis e interpretación de los siguientes problemas no resolubles:

a) Dada una MT M, ¿$\varepsilon \in M$?

b) Dada una MT M, ¿Es $L(M) = \varnothing$?

c) Dada una MT M, ¿Es L(M) finito?

12.3 Dados M y N, los dos **AFD**s mostrados en la Figura 12.10 y los conjuntos siguientes, responder a cada una de las preguntas:

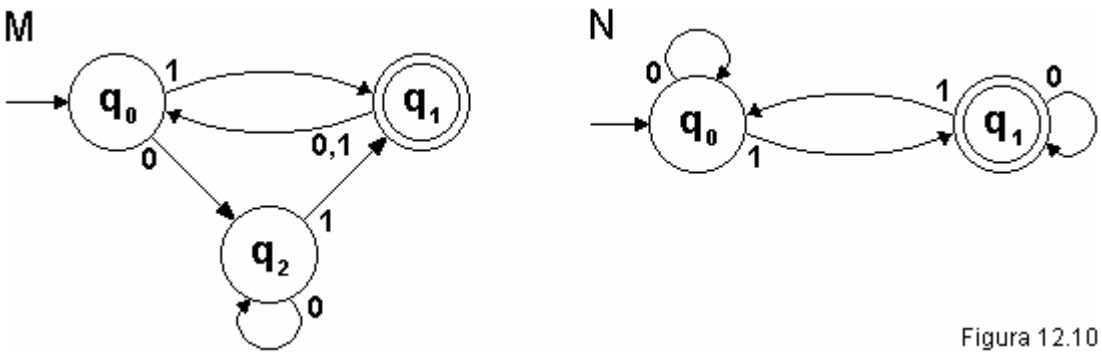


Figura 12.10

- A = { ⟨M,w⟩ | M es un AFD que acepta como entrada la cadena w }
- B = { ⟨M⟩ | M es un AFD y L(M) = ∅ }
- C = { ⟨M,N⟩ | M, N son AFDs y L(M) = L(N) }
- D = { ⟨r,w⟩ | r es una expresión regular que comprende a la cadena w }

a) ¿Está ⟨M,001⟩ en A?

b) ¿Está ⟨M,0111⟩ en A?

c) ¿Está ⟨N,000⟩ en A?

d) ¿Está ⟨M⟩ en B?

e) ¿Está ⟨M,N⟩ en C?

f) ¿Está ⟨N,N⟩ en C?

g) ¿Está ⟨N,0111⟩ en D?

h) ¿Está ⟨0111,0111⟩ en D?

i) ¿Está ⟨01*0,0111⟩ en D?

12.4 Para cada uno de los siguientes sistemas de correspondencia de Post, sobre el alfabeto Σ = { **a**, **b** }, obtener una solución o explicar por qué no existe.

a) A = { **aaa**, **baa** } y B = { **aa**, **abaaa** }

b) A = { **ab**, **bba**, **aba** } y B = { **aba**, **aa**, **bab** }

c) A = { **a**, **bb**, **a** } y B = { **aa**, **b**, **bb** }

d) A = { **a**, **aab**, **abaa** } y B = { **aaa**, **b**, **ab** }

e) A = { **ab**, **ba**, **b**, **ba** } y B = { **a**, **bab**, **aa**, **ab** }

f) A = { **ab**, **aa**, **ab**, **bb** } y B = { **bb**, **ba**, **abb**, **bab** }

g) A = { **aa**, **bb**, **abb** } y B = { **aab**, **ba**, **b** }

h) A = { **ab**, **baa**, **aba** } y B = { **aba**, **aa**, **baa** }

i) A = { **a**, **ba**, **b**, **bba** } y B = { **ab**, **aba**, **aba**, **b** }

j) A = { **b**, **aa**, **bab**, **ab** } y B = { **ba**, **b**, **aa**, **ba** }

k) A = { **abb**, **aba**, **aab** } y B = { **ab**, **ba**, **abab** }

12.5 Decidir si las siguientes instancias de **PCP**, sobre el alfabeto Σ = { **0**, **1** }, tiene solución. Justificar la respuesta en cada caso.

a) A = { **1**, **11**, **011** } y B = { **11**, **10**, **1** }

b) A = { **1**, **11**, **011** } y B = { **11**, **100**, **1** }

c) A = { **01**, **110010**, **1**, **11** } y B = { **0**, **0**, **1111**, **01** }

d) A = { **0**, **01** } y B = { **10**, **1** }

Capítulo

13

Computabilidad

Se definen el concepto de Computabilidad. Se describen las Funciones Iniciales y las Construcciones Primitivas. Se definen Funciones Recursivas de diferentes tipos y las Funciones Características. Se define Incomputabilidad.

El objetivo de este capítulo es identificar cuáles son todas las funciones que pueden calcularse por medio de algún sistema computacional; en otras palabras, diremos que una función es computable, si se puede calcular mediante algún algoritmo, sin necesidad de especificar ningún algoritmo en concreto.

Como punto de partida, comenzaremos por definir una serie de *funciones iniciales*, tan sencillas que no crean dudas en cuanto a su Computabilidad, para luego mostrar que estas funciones pueden *combinarse* para formar otras, cuya Computabilidad se desprenda de las funciones originales.

Para que finalmente obtengamos una colección que contenga a todas las *funciones computables*. Así, si un sistema computacional (lenguaje de programación), abarca todas esas funciones, diremos que el sistema tiene todo el poder posible.

Funciones Iniciales

La única capacidad que se le reconoce inicialmente a nuestro sistema computacional es que sea capaz de *Contar*, mediante el uso de tres tipos de funciones iniciales:

1. Fijar un punto inicial como 0 (Función Cero ζ)
2. Reconocer el número siguiente de cada número natural (Función Sucesor σ).
3. Y la familia de funciones π, son necesarias para cambiar de dimensión entre espacios (Funciones Proyección).

Función Cero

La función *Cero* se representa por ζ. Establece una correspondencia entre una variable vacía y el 0. Es computable porque corresponde al proceso de escribir un cero en un papel en blanco, o registrar un cero en una celda de memoria vacía.

Función Sucesor

Se representa por σ. Establece una correspondencia entre un número natural y su sucesor, de manera que $\sigma(x) = x + 1$ para cada entero $x > 0$. Es computable, porque se conoce desde hace tiempo un proceso computacional para sumar uno a un entero.

Funciones Proyección

La familia de Funciones Proyección se representa por π. Obtiene como salida un componente específico de su tupla de entrada. Utiliza un superíndice para indicar el tamaño de la entrada, y un subíndice para indicar el lugar que ocupa en la tupla el componente que se extrae.

De esta forma se tiene que: $\pi^3_2(7,6,4) = 6$, $\pi^2_1(5,14) = 5$, $\pi^2_2(3,17) = 17$, y como caso especial, $\pi^2_0(6,5) = ()$, es decir, un valor vacío. Al igual que las anteriores, es fácil ver que las funciones Proyección, están en la clase de las *Funciones Computables*.

Construcciones Primitivas

Las *Funciones Iniciales* forman la base de la jerarquía de un grupo de Funciones más extensas llamadas *Funciones Primitivas Recursivas*. Nuestra siguiente tarea es investigar cómo pueden emplearse las Funciones Iniciales para construir otras funciones más complejas.

Combinación

Una forma de construir funciones más complejas a partir de las funciones iniciales es por medio de la combinación, definida como sigue:

Sean dos funciones $f: N^k \rightarrow N^m$ y $g: N^k \rightarrow N^n$, la combinación de ambas se representa por el símbolo $\times$ y es una función: $f \times g: N^k \rightarrow N^{m+n}$.

Es decir, toma entradas en forma de k-tuplas, y produce salidas cuyos primeros m componentes consisten en la salida de f, y los n siguientes, en la salida de g.

Por tanto, la combinación de dos funciones computables produce una función computable.

Ejemplo

$\sigma(5) \times \pi^3_2(1,5,3) = (6,5)$

Composición

Sean dos funciones f: $N^k \rightarrow N^m$ y g: $N^m \rightarrow N^n$, la composición de ambas funciones está definida como una función: f ∘ g: $N^k \rightarrow N^n$.

Es decir, para encontrar la salida de f ∘ g, primero aplicamos f a la entrada, y después la función g, a la salida de f. Por tanto, la composición de dos funciones computables produce una función computable.

Ejemplo

$\zeta \circ \sigma() = 1$, ya que $\zeta()$ es 0, y $\sigma(0)$ es 1, otra manera de representarlo es: $\sigma(\zeta()) = 1$.

Recursividad primitiva

La recursividad primitiva dice de manera general que, si dada f una función cualquiera f: $N^{k+1} \rightarrow N^m$, ésta se puede definir por medio de dos funciones g y h, con g: $N^k \rightarrow N^m$ y h: $N^{k+m+1} \rightarrow N^m$, respectivamente. Entonces decimos que f esta construida a partir de g y h por medio de recursividad primitiva.

Una función construida por recursividad primitiva a partir de funciones computables, debe considerarse computable.

Ejemplo

Podemos definir recursivamente la función Mas de la siguiente manera:

- $Mas(x,0) = \pi^1_1(x) = x$
- $Mas(x,y+1) = \sigma \circ \pi^3_3(x,y,Mas(x,y))$

Es decir, que x + 0 es x, mientras que x + (y + 1) se obtiene (recursivamente) encontrando al sucesor de x + y.

Funciones Recursivas Primitivas

La clase de las funciones que están por encima de las funciones iniciales, es la clase de las funciones recursivas primitivas, que consisten en todas las funciones que se construyen a partir de *Funciones Iniciales* aplicando un número finito de *Combinaciones*, *Composiciones* y *Recursividades Primitivas*.

Esta clase es muy extensa, incluyendo a casi todas las funciones totales que se requieren en las aplicaciones de un computador tradicional. (*Funciones Totales* son

las que están definidas para todos las tuplas de su dominio). A continuación describiremos a las Funciones Recursivas Primitivas más comunes.

Funciones Constantes

Las Funciones Constates producen una salida fija, predeterminada, sin importar cuál sea la entrada. Se representan por K^n_m, donde n indica la dimensión del dominio, y m es el valor de salida de la función. Por ejemplo, K^3_5 establece la correspondencia entre cualquier tupla de tres elementos, y el valor 5.

Es evidente, que las funciones constantes son recursivas primitivas.

Ejemplo

Para producir la función K^0_n se puede aplicar la función sucesor un número n de veces: $K^0_2 = \sigma \circ \sigma \circ \zeta$.

También son recursivas primitivas, las funciones constantes cuyas salidas tienen más de una componente, ya que sólo son combinaciones de funciones del tipo K^n_m.

Ejemplo

La función que establece una relación entre cualquier tripleta, y el par (2,5) está definida por: $K^3_2 \dot{\times} K^3_5$.

Para evitar complicaciones de notación, si el tamaño del dominio queda claro, señalaremos estas funciones como su valor, es decir: $K^3_5 \dot{\Leftrightarrow} 5$ y $K^3_2 \times K^3_5 \dot{\Leftrightarrow} (2,5)$.

Multiplicación

Con base en las funciones constantes y la función Mas, podemos definir a la multiplicación como función recursiva primitiva:

- Mult(x,0) = 0
- Mult(x,y+1) = Mas(x, Mult(x,y))

Exponenciación

De manera semejante, con base en las funciones constantes y la función Mult, podemos definir a la exponenciación como función recursiva primitiva:

- Exp(x,0) = 1
- Exp(x,y+1) = Mult(x,Exp(x,y))

Función Predecesor

Da el valor de su predecesor, excepto cuando la entrada es cero, que no lo varía, dando también cero a la salida.

- Pred(0) = $\zeta()$
- Pred(y+1) = $\pi^2_1(y,Pred(y))$

Resta positiva

En cierta forma, Pred es la función inversa de σ, por lo que puede usarse para construir la resta de la misma manera que $\dot{\sigma}$ se usa para definir la suma. La función Monus(a,b), la representaremos también como a – b.

- Monus(x,0) = x
- Monus(x,y+1) = Pred(Monus(x,y))

Funciones Características

Las funciones características, son aquellas en que sólo tienen dos posibles valores de salida. Son la representación funcional de un problema de decisión, como por ejemplo:

Función Igualdad

La Función Igualdad determina si dos valores son iguales, dando 1 si son iguales, y 0 si son diferentes.

- Eq(x,y) = Monus $\circ$ ($K^2_1 \dot{\times}$ (Mas $\circ$ ((Monus $\circ$ ($\pi^2_2 \dot{\times} \pi^2_1$)) $\times$ (Monus $\circ$ ($\pi^2_1 \dot{\times} \pi^2_2$)))))

O lo que es lo mismo:

- Eq(x,y) = 1 – ((y – x) + (x – y))

Funciones Tabulares

Son funciones que están definidas por una tabla con un número finito de entradas, y sus valores de salida correspondientes. Por ejemplo:

Entrada	Salida
0	3
4	5
otros	2

Tabla 13.1

Estas funciones son recursivas primitivas, y para demostrarlo, primero veremos la Función Característica κ_i que da como salida 1 si la entrada es i, y cero en los demás casos. Esta función κ_i se puede formar con:

- κ_i = Monus(I_i, I_i – 1), donde I_i = Eq((x – i), 0).

De esta forma, vemos que una función tabular no es más que la suma finita del producto de funciones características, constantes y características negadas. Por ejemplo, la función de la tabla anterior, se puede construir mediante:

- Mult(3, κ_0) + Mult(5, κ_4) + Mult(2, Mult($\sim\kappa_0$, $\sim\kappa_4$))

Función Cociente:

La función Coc: $N^2 \rightarrow N$, definida por la parte entera del cociente de sus componentes, o cero en el caso de que el segundo sea cero, es recursiva primitiva y puede construirse de la siguiente forma:

- Coc(0,y) = 0
- Coc(x+1,y) = Coc(x,y) + Eq(x+1, Mult(Coc(x,y), y) + y)

Ejemplo

Para calcular Coc(9, 3), se haría:

Coc(9, 3) = Coc(8+1, 3) = Coc(8, 3) + Eq(9, Mult(Coc(8, 3), 3) + 3)
= 2 + Eq(9, Mult(2, 3) + 3) = 2 + Eq(9, 6 + 3) = 2 + Eq(9, 9) = 2 + 1 = 3

Funciones μ-recursivas

Con lo visto hasta ahora, utilizamos las *Funciones Iniciales* para crear a las Funciones *Recursivas Primitivas*. Pero éstas, no abarcan a todas las funciones *totales computables*. Como ejemplo de esta última afirmación tenemos a la función de Ackermann, que es la función A: $N^2 \rightarrow N$, definida por:

- A(0, y) = y + 1
- A(x+1, 0) = A(x, 1)
- A(x+1, y+1) = A(x, A(x+1, y))

La cual es computable y total, pero no es recursiva primitiva.

De todas formas, no necesitamos un ejemplo de estas funciones para demostrar el siguiente teorema:

Teorema

Existe una función total computable de N a N, que no es recursiva primitiva.

Para demostrarlo, supongamos que es posible representar cada una de las funciones recursivas primitivas, mediante una cadena finita de símbolos, por lo que es posible ordenarlas y hablar así de f_1 la primera, f_2 la segunda, etc.

Si definimos ahora una nueva función $g(n) = f_n(n)+1$. Podemos decir que g es total y computable (se puede calcular a partir de f_n). Sin embargo no es recursiva primitiva, porque si lo fuera, sería un f_m (para algún m), pero entonces, g(m) sería igual a $f_m(m)$, lo cual no puede ser, porque la definimos como: $g(m) = f_m(m) + 1$

La clase de las funciones totales computables, se conoce como la clase de las funciones μ-recursivas.

Funciones Parciales

No todas las funciones tienen la forma: $f: N^m \rightarrow N^n$ (una m-tupla de enteros como entrada, y una n-tupla de enteros como salida), es decir, no siempre la función está definida para todas las m-tuplas. Por ejemplo, la función Coc(*x*, *y*), no es de la forma *f*: $N^2 \rightarrow N$, ya que no está definida para los pares donde el segundo componente sea cero.

Una función parcial de un conjunto X, es aquella cuyo dominio consiste en un subconjunto de X, entonces, la función Coc(x, y) es una función parcial de N^2 ya que su dominio es un subconjunto de N^2. Es posible identificar cualquier función computable como una función parcial de la forma: $f: N^m \rightarrow N^n$

Funciones Recursivas Parciales

Para ampliar nuestro estudio más allá de las funciones totales computables, siempre dentro de las funciones computables, aplicamos la técnica de la minimalización.

Minimalización

La técnica de Minimalización nos permite crear una función $f: N^n \rightarrow N$, a partir de otra función $g: N^{n+1} \rightarrow N$, declarando a f(x) como la menor y, tal que g(x, y) = 0, y donde g(x, z) está definida para todos los enteros no negativos z menores que y.

Esta construcción se representa como $f(x) = \mu y[g(x, y) = 0]$.

Ejemplo

Supongamos que la función g(x,y) está definida por la tabla siguiente y que f(x) se define como $\mu y[g(x, y) = 0]$:

g(0,0) = 2	g(0,0) = 3	g(2,0) = 8	g(3,0) = 2	…
g(0,1) = 3	g(1,1) = 4	g(2,1) = 3	g(3,1) = 6	…
g(0,2) = 1	g(1,2) = 0	g(2,2) = ind.	g(3,2) = 7	…
g(0,3) = 5	g(1,3) = 2	g(2,3) = 6	g(3,3) = 2	…
g(0,4) = 0	g(1,4) = 2	g(2,4) = 0	g(3,4) = 8	…
…	…	…	…	…

Tabla 13.2

Entonces f(0) = 4, puesto que g(0,4) = 0, f(1) = 2, porque g(1,2) = 0 y f(2) no está definida porque antes de encontrar el primer valor igual a cero se encuentra uno que no está definido. En lo que se refiere a f(3), la tabla no nos da suficiente información.

Como se ve en este ejemplo, la minimalización puede producir funciones que no están definidas para ciertas entradas.

Con esta técnica podemos construir la función Coc (cociente entero):

$$Coc(x,y) = \mu t[(x+1) – (Mult(t, y) + y) = 0]$$

- Coc(5,4) = 1, ya que 1 es el menor valor de t, para el que se cumple que:
  $(5 + 1) – (Mult(t,4) + 4) = 0 \Leftrightarrow 6 – (1 \times 4) + 4 \Leftrightarrow 6 – 8 = 0$
- Coc(3,4) = 0, ya que 0 es el menor valor de t, para el que se cumple que:
  $(3 + 1) – (Mult(t,4) + 4) = 0 \Leftrightarrow 4 – (0 \times 4) + 4 \Leftrightarrow 4 – 4 = 0$
- Coc(3,0) no está definida, ya que no hay ningún valor de t, que haga cumplir que:
  $(3 + 1) – (Mult(t,0) + 0) = 0 \Leftrightarrow 4 – 0 \neq 0$.

Habíamos visto que la función Coc, no era *recursiva primitiva*, pero puede construirse mediante la minimalización, por lo que pertenece a la clase de las Funciones *Recursivas Parciales*.

Es evidente, que una función creada mediante la minimalización, es computable. La nueva clase descrita, es decir, la de las funciones recursivas parciales, es la clase de las funciones creadas a partir de funciones iniciales, y aplicando un número finito de combinaciones, composiciones, recursividades primitivas y minimalizaciones.

En la Figura 13.1 se muestra toda la jerarquía de las funciones computables, ya que la clase de las funciones recursivas parciales, contiene a todas las funciones parciales computables.

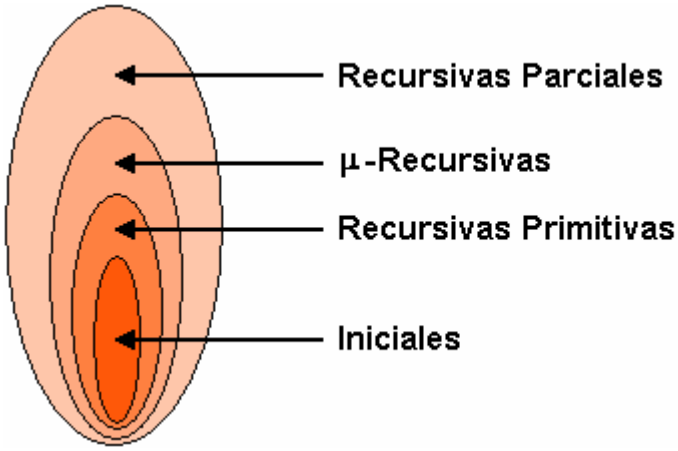


Figura 13.1

Esta tesis se conoce como Tesis de Church (análoga a la forma referida al final del Capítulo 11). Nadie ha encontrado una función parcial que sea computable, pero que no sea recursiva parcial, lo que da validez a esta tesis.

Los problemas cuya solución puede especificarse como una función recursiva parcial son exactamente los mismos que pueden solucionarse mediante máquinas de Turing. Para sustentar una afirmación así es conveniente enunciar un teorema:

Teorema

Todo proceso computacional puede ser realizado por una máquina de Turing si y sólo si es el cálculo de una función recursiva parcial.

Incomputabilidad

No es lo mismo afirmar que todavía no se ha encontrado la solución de un problema, a decir, que se ha demostrado que no puede existir un algoritmo, procedimiento o función que lo resuelva.

Cuando nos referimos a Incomputabilidad, nos referimos a que existen muchos más problemas que soluciones que puedan ser construidas. De esta forma, aunque no se digan cuales, se sabe que existen una infinitud de problemas sin solución.

La única manera eficaz de dar un ejemplo de alguno de estos problemas, y demostrar que no pueden solucionarse, es por reducción al absurdo: Esto es, se demuestra que

no pueden ser ciertas a la vez toda la teoría construida hasta el momento, y la afirmación de que ese problema tiene solución computable.

Preguntas

a) ¿En qué nos basamos al afirmar que existen más problemas que soluciones posibles?

b) ¿Hay funciones parciales que no sean recursivas?

c) ¿Todas las Funciones Totales son Computables?

Ejercicios Capítulo 13

13.1 Calcule el elemento imagen de las siguientes Funciones Recursivas Primitivas:

a) $\sigma \circ \pi^2_1(3, 5)$

b) $\pi^1_1 \circ \sigma(4)$

c) $\sigma \circ \zeta()$

d) $\sigma \circ \zeta \circ \pi^3_0(5, 6, 2)$

13.2 Considere las funciones siguientes y determine, si están bien definidas, cuál es su dominio y su espacio final:

a) $f = \pi^2_1 \circ \sigma$

b) $g = \zeta \circ \sigma \circ \pi^2_2$

c) $h = \sigma \circ \zeta$

d) $k = \sigma \circ \zeta \circ \pi^3_0$

13.3 Calcule el elemento imagen de las siguientes funciones:

a) $\pi^3_2 \times \pi^3_1 (4, 2, 5)$

b) $\pi^3_1 \times \pi^3_2 (4, 2, 5)$

c) $(\sigma \circ \pi^3_2) \times (\zeta \circ \pi^3_0)(4, 2, 5)$

d) $\pi^2_2 \times (\zeta \circ \pi^2_0) \times (\sigma \circ \sigma \circ \pi^2_2) \circ ((\sigma \circ \pi^2_2) \times \pi^2_1) (4, 6)$

13.4 Defina las funciones siguientes de forma tal que se verifiquen las igualdades. Utilice exclusivamente composición, combinación o recursividad primitiva de funciones proyección o funciones constantes.

a) f(2, 3, 4) = (3, 4, 7)

b) g(2,3) = (3, 3, 9)

c) h(2, 3, 4) = (7, 2)

13.5 Dado que el esquema general de definición utilizando recursividad primitiva es: f(x, y) = g(x) y f(x, y + 1) = h(x, y, f(x, y)). Obtenga las expresiones adecuadas de g y h para las funciones Mas(x, y), Mult(x, y) y Exp(x, y).

13.6 Sea A la función de Ackermann, definida previamente en la página 212, demuestre que A(1, z) = z + 2 para todo z.

13.7 Si g(x) = x y h(x, y, z)= z + 2, ¿Que función se obtiene desde g y h por recursión primitiva?

13.8 Mostrar que las siguientes funciones son recursivas primitivas:

a) Abs(x, y) = | x – y |

b) Fact(x) = x!

c) Min(x, y)

d) Max(x, y)

13.9 Construir una función de la forma μ-recursiva de la definición de la función Módulo (Residuo).

Capítulo

14

Complejidad

Se define el concepto de Complejidad. Se estudia el impacto de la duración para tratabilidad de los problemas por medio de Máquinas de Turing.

Un problema es *Decidible* (*Computable* o *Resoluble*) si existe un algoritmo para resolverlo, sin embargo, en la práctica esto no es suficiente, ya que pueden existir otro tipo de limitaciones en cuanto a los recursos de tiempo y espacio necesarios para solucionarlo, por lo que, desde el punto de vista práctico, un problema, a pesar de ser soluble, puede permanecer sin solución.

El objetivo de este capítulo es el de clasificar a los problemas solubles desde el punto de vista de la dificultad intrínseca que presentan para resolverlos.

Complejidad de los cálculos

Primeramente hay que establecer una escala para clasificar los problemas de acuerdo a su complejidad, para ello, debemos aprender a medir la complejidad de los cálculos individuales, a partir de lo cual podemos determinar la complejidad de los algoritmos, y como consecuencia, la complejidad de los problemas.

Comenzamos el estudio formalizando el concepto de cálculo: Un cálculo consiste en un par determinado formado por un algoritmo y una entrada; o bien, desde otro punto de vista: la combinación de una **MT** en concreto y una cadena de entrada en concreto.

Medición de la complejidad

La complejidad de un algoritmo se refiere entonces, a la cantidad de tiempo y espacio requeridos para ejecutarlo, para una entrada determinada. Es posible medir tanto su complejidad temporal como espacial y ambas serán unos números determinados.

El *tiempo* es uno de estos recursos. Llamamos complejidad temporal a la cantidad de tiempo necesaria para efectuar el cálculo: Por ejemplo, la búsqueda binaria de un elemento en una lista ordenada, tiene una complejidad temporal estimada, menor que la búsqueda secuencial, puesto que usualmente se realiza más rápido.

La cantidad de *espacio de almacenamiento* requerido, es otro de los recursos mencionados. Llamamos complejidad espacial, a la cantidad de espacio de almacenamiento requerido para efectuar un cálculo.

La complejidad espacial o temporal de un cálculo, puede variar dependiendo del sistema en donde se ejecute el cálculo (éste puede hacerse en una computadora rapidísima, o con una calculadora de bolsillo. Así mismo, los datos pueden estar codificados en binario o escritos en papel, etc.) Para evitar estas variaciones, es común estudiar la complejidad, en el contexto de un sistema computacional fijo, en este caso, nos referimos a la máquina de Turing.

Complejidad de los cálculos de Máquinas de Turing

Definimos la complejidad temporal de una máquina de Turing, como el número de pasos que se ejecutan durante los cálculos. Por ejemplo, la **MT** de la Figura 14.1, tendrá una complejidad temporal de 5 para un cálculo en el que la cinta contenga originalmente: **#xxx##**, (La cabeza recorrerá un espacio, las tres **x**, y después otro espacio en blanco más). Similarmente, si se inicia con una cinta que contenga solo espacios en blanco: **###**, la complejidad espacial será solamente de 2.

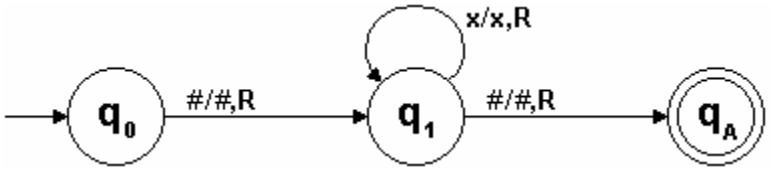


Figura 14.1

En general, las complejidades espacial y temporal son diferentes, sin embargo, no son completamente independientes, ya que en *n* pasos, una **MT** tiene acceso cuando mucho a *n* + 1 posiciones de la cinta. Por tanto, si la complejidad temporal es *n*, entonces la complejidad espacial nunca será mayor que *n* + *1*.

Si no exigimos que la máquina borre la cinta antes de pararse, es posible que dicha máquina acepte una cadena con una complejidad espacial menor que el espacio requerido para contener la cadena. Por ejemplo, si las cadenas aceptadas sólo se caracterizan por comenzar por el símbolo **x**, entonces, una vez examinado el primer símbolo, podemos aceptar o rechazar la cadena sin necesidad de desplazarnos por el resto de posiciones de la cinta que contienen a la cadena, sin embargo, si se exige que se borre la cinta, la complejidad espacial será por lo menos igual a la longitud de su cadena de entrada.

Complejidad de los algoritmos

En general, el uso de un mismo algoritmo para distintas entradas, produce cálculos de complejidades diferentes. Para calcular si un algoritmo es complejo, nos basamos en si los cálculos son complejos o no, entonces, ¿qué ocurre cuando unas entradas nos llevan a cálculos sencillos, y otras generan cálculos complejos?

En este caso, se mantiene fija (y única) la **MT** utilizada y se estudia para todas las entradas posibles. Las complejidades medidas pueden variar enormemente. Uno de los factores que más influyen en esta variación es la longitud de la cadena de entrada, por lo que se deben estudiar por separado las distintas entradas con la misma longitud. Se escoge el peor caso posible dentro de cada una de las familias de entradas de cada longitud y todo esto se sintetiza como una función de la longitud de la entrada.

Complejidad temporal de las máquinas de Turing

Consideremos una máquina de Turing, para determinar si dos cadenas de igual longitud son iguales, para el alfabeto Σ = { **x**, **y**, **z** }. Suponemos que ambas están escritas en la cinta, una detrás de la otra, y separadas por un signo de igual. Por ejemplo, se puede comenzar con una cinta que contenga **#yxxz=yxzx##**, y terminar con **#1##** o con **#0##**, si las cadenas son iguales o distintas respectivamente.

Nuestra **MT** observa el primer símbolo de la primera cadena, y mueve la cabeza hasta el primer símbolo de la segunda cadena. Si son iguales, hace lo mismo con los segundos símbolos de ambas cadenas, y así hasta encontrar una discrepancia o hasta agotar ambas cadenas.

Evidentemente, el tiempo requerido para ejecutar este algoritmo, debe ser función de la longitud de las cadenas de entrada. Así, sin entrar en más detalle, diremos que el tiempo necesario (número de pasos ejecutados) para confirmar que dos cadenas de longitud n son idénticas, es: $T(n) = 2n^2 + 6n + 6$

Esto incluye $(2n^2 + 3n + 1)$ pasos para la comparación, $(3n + 3)$ pasos para mover la cabeza hacia el extremo derecho de la entrada y borrar la cinta de derecha a izquierda y por último un paso para escribir el símbolo **1**.

Es decir, comparar dos cadenas de 4 elementos requerirá una complejidad temporal de 62, mientras que comparar dos cadenas de longitud 10, requeriría de 266 pasos.

Esto no es siempre cierto, ya que sólo nos referimos a la comparación de dos cadenas idénticas, pero si las cadenas son diferentes, la complejidad dependerá del momento en que se encuentre la primera discrepancia, y esto puede ocurrir en un tiempo muy corto. Generalmente se manejan estas variaciones en el rendimiento de un algoritmo, identificando las situaciones de *mejor caso* y *peor caso*. Esto nos garantiza que cualquier aplicación de dicho algoritmo, cae dentro de ese intervalo.

En nuestro ejemplo, el peor caso será cuando las cadenas son idénticas y haya que analizarlas completamente, y el mejor caso, cuando la discrepancia se encuentra en el primer símbolo. Entonces, podemos afirmar que la complejidad se encontrará dentro del rango entre $T_m(n) = 4n + 5$ y $T_p(n) = 2n^2 + 6n + 6$.

Generalmente se acostumbra adoptar un punto de vista pesimista y definir la complejidad temporal de un algoritmo como su rendimiento en el peor caso posible.

Rendimiento medio

Si utilizamos un algoritmo con mucha frecuencia, nos interesará encontrar su rendimiento promedio, más que describir el comportamiento en el peor o en el mejor de los casos.

La estimación del rendimiento promedio no consiste en el promedio aritmético de los peor y mejor casos, sino que se calcula de una forma más precisa, multiplicando la complejidad de cada cálculo posible ($\mathbf{c}_1$, $\mathbf{c}_2$, …, $\mathbf{c}_m$), por la probabilidad de que ocurra dicho cálculo ($\mathbf{p}_1$, $\mathbf{p}_2$, …, $\mathbf{p}_m$) y realizando la sumatoria para todos estos casos, así:

$$T(n) = \sum_{i=1}^{m} p_i c_i$$

Análisis informal de algoritmos

Generalmente, determinar la complejidad temporal de un algoritmo no es tan sencillo, y hemos de conformarnos con una estimación muy aproximada, lo que, sin embargo, suele ser suficiente. Por ejemplo, nos basta con tener una idea aproximada de las complejidades de dos distintas técnicas, para poder evaluar las ventajas relativas entre ambos algoritmos, y tomar una decisión de cual es la más adecuada.

Así, por ejemplo, si queremos calcular la complejidad del algoritmo de ordenación por inserción, debemos suponer que será proporcional al número de veces que se ejecuta

su bucle más interno. De esta forma podríamos determinar que la complejidad temporal del algoritmo será proporcional a $n^2 – n$, donde n es la longitud de la lista a ordenar.

Por supuesto, esto es una aproximación, y no podemos esperar que nos calcule el tiempo exacto de una aplicación del algoritmo. Sin embargo, sí podemos afirmar que si vamos aumentando la longitud de entrada, el tiempo necesario aumentará en proporción a una expresión cuadrática. Es decir, si se duplica el tamaño de la lista, el tiempo aumentará en un factor *aproximadamente* de cuatro.

Estas aproximaciones son de gran utilidad para elegir entre varias estrategias para resolver un problema concreto. Sin embargo, las consideraciones necesarias para juzgar el rendimiento estimado de un algoritmo, varían de acuerdo con la situación concreta. Hay que considerar, por tanto, no sólo los casos peor, mejor y promedio, sino también el ambiente donde se aplicará el algoritmo.

Esta elección debe ser independiente del mecanismo empleado para su ejecución, ya que los algoritmos de complejidad temporal muy grande, no mejoran por el simple hecho de que se ejecuten en una veloz supermáquina.

Complejidad de los problemas

A veces decimos que un problema es difícil cuando nos resulta difícil resolverlo. Sin embargo, *siempre existe una forma difícil de resolver cualquier problema*, por ello, la complejidad de un problema se debe medir considerando la complejidad de la solución más sencilla posible.

Normalmente no es fácil identificar cual es la solución más sencilla, para ello se tendrían que considerar todas las posibles **MT**s conocidas para resolver el problema, y analizar a cada una de ellas bajo todas las entradas posibles, y de esta manera, poder asignar al problema la complejidad del algoritmo que resulte el mejor, hasta ese momento, pues posteriormente puede surgir uno más eficiente. De este modo, la búsqueda de una solución óptima, se puede convertir en una labor sin fin de descubrimientos sucesivos de mejores soluciones.

Mejoramiento de soluciones

En la máquina de Turing que analizamos anteriormente, donde se comparaban dos cadenas de igual longitud, cotejando los primeros elementos de cada cadena,

después los segundos, y así sucesivamente; conseguimos, de este modo, una complejidad de $T(n) = 2n^2 + 6n + 6$.

Podemos producir una solución más rápida, si diseñamos una **MT** que compare dos símbolos cada vez. Esto reduciría aproximadamente a la mitad el número de movimientos. Si continuamos con esa técnica, podríamos construir máquinas que compararan k símbolos a cada vez y obtendríamos una solución que sea aproximadamente 1/k veces mejor.

Esto nos lleva a un sin fin de soluciones cada una más eficiente que la anterior, de hecho, el teorema de aceleración de Blum establece que existen problemas para los cuales es posible mejorar *considerablemente* cualquier solución. Y que en términos generales, dice que:

En el contexto de las máquinas de Turing (y por ende en otros sistemas), para cualquier función μ-recursiva g, existe un problema para el cual, cualquier solución con complejidad temporal t(n), se puede mejorar para obtener una nueva solución con complejidad temporal g(t(n)) para todos, excepto un número finito de valores, a su vez, ésta nueva solución se puede mejorar para obtener una complejidad g(g(t(n))) y así sucesivamente.

Sin embargo, este teorema no ha sido aplicado fuera de las ciencias teóricas, por lo que tiene poca importancia en situaciones prácticas.

Por todo esto, al clasificar a los problemas de acuerdo con su complejidad, nos conformaremos con dar una magnitud: Por ejemplo, el problema de comparación de cadenas, tiene una complejidad que es del orden de una *expresión cuadrática*. Es decir, hemos clasificado la complejidad del problema en la clase de las funciones cuadráticas.

En la mayoría de los casos, esta clasificación será más que suficiente para nuestros propósitos. Una forma útil para medir la complejidad de los problemas consiste en establecer una escala basada en *Clases* de funciones. Una de estas escalas es la de las *Tasas de Crecimiento*.

Tasas de Crecimiento

Dada una función $f: N \rightarrow R^+$, decimos que el Orden de la función f (se denota O(f), leído como *o mayúscula de f*) es el conjunto de todas las funciones $g: N \rightarrow R^+$,

acotadas superiormente por un múltiplo real positivo de f, para valores de n suficientemente grandes.

Hay que notar que O(f) es un conjunto de funciones, y no una sola función y que para cada función g(n) del conjunto, existe un entero n_0, denominado *umbral*, tal que se cumple lo anterior para todo $n \geq n_0$.

$$O(f) = \{ g: N \rightarrow R^+ \mid \exists\, c \in R^+, \exists\, n_0 \in N \mid \forall\, n \geq n_0, g(n) \leq cf(n) \}$$

Dos funciones *f* y *g* son equivalentes si O(*f*) = O(*g*). El conjunto de funciones equivalentes a *f*, se denota como $\Theta(f)$, es decir, "theta mayúscula de *f*". A cada clase $\Theta(f)$, se le llama tasa de crecimiento.

La tasa de crecimiento de cualquier polinomio de grado d, es $\Theta(n^d)$:

$$\Theta\left(\sum_{i=0}^{d} a_i n^i\right) = \Theta\left(n^d\right)$$

Ahora, aunque no podemos identificar una función específica como la complejidad temporal de cualquier problema, sí podemos decir que es posible resolverlo con un algoritmo cuya complejidad se encuentra, por ejemplo, en la clase $\Theta(n^2)$.

Así pues, las tasas de crecimiento pueden proporcionar un esquema de clasificación para determinar la complejidad de los problemas. La complejidad de un problema será de la clase $\Theta(f)$, si se puede resolver mediante un algoritmo de esa complejidad y donde cualquier mejor solución también está en esa clase.

Limitaciones de la escala de tasas de crecimiento

La escala de tasas de crecimiento tiene éxito para clasificar la complejidad de varios problemas en distintas situaciones, sin embargo, si cambiamos de un sistema computacional a otro, es posible que la complejidad temporal corresponda a otra tasa de crecimiento (por ejemplo, un problema puede tener una complejidad cuadrática usando máquinas de Turing de una cinta, pero puede ser lineal si usamos máquinas de Turing de dos cintas).

Por lo tanto, la complejidad de un problema, medida como una tasa de crecimiento, no constituye una propiedad exclusiva del problema, sino que además, depende del sistema computacional subyacente.

En la tabla 14.1 se muestran diferentes funciones de complejidad y el tiempo que consumiría en resolver problemas de diferente tamaño, suponiéndose que cada paso tarda un microsegundo en ejecutarse.

O(n)	n = 10	20	50	100	500	1000
ln n	2.3×10^{-6} seg	3×10^{-6} seg	3.9×10^{-6} seg	4.6×10^{-6} seg	6.2×10^{-6} seg	6.9×10^{-6} seg
N	1×10^{-5} seg	2×10^{-5} seg	5×10^{-5} seg	.0001 seg	.0005 seg	.001 seg
n ln n	2.3×10^{-5} seg	6×10^{-5} seg	2×10^{-4} seg	4.6×10^{-4} seg	.0031 seg	.0069 seg
n^2	.0001 seg	.0004 seg	.0025 seg	.01 seg	.25 seg	1 seg
n^3	.001 seg	.008 seg	.125 seg	1 seg	125 seg	1000 seg
n^5	.1 seg	3.2 seg	5.2 min	2.8 hrs	1 año	31.7 años
2^n	.001 seg	1 seg	35.7 años	4×10^{14} sig	∞	∞
3^n	.059 seg	58 min	2.3×10^{8} sig	1.6×10^{32} sig	∞	∞
n!	3.6 seg	771 siglos	9.6×10^{48} sig	∞	∞	∞

Tabla 14.1

Como podemos ver, existen algunos problemas, que debido a su complejidad, no es posible resolverlos en la práctica, por la magnitud de tiempo tan considerable que requeriría su solución, incluso para valores de n razonablemente pequeños.

Cualquier algoritmo se vuelve inútil cuando el tiempo requerido para su ejecución es demasiado grande, por ello, siempre es necesario estimar de qué orden es la función del tiempo requerido para la ejecución de algunos algoritmos.

Y en los casos en que los problemas sean muy complejos, se debe tratar de encontrar otro algoritmo equivalente que sea más eficiente. La eficiencia entonces, consiste en reemplazar algoritmos realizables por otros cuya función del tiempo se encuentre más arriba de la tabla.

Intratabilidad

Cuando la función de complejidad de un problema depende polinomialmente del tamaño del problema, se dice que el problema es *Tratable*, (esto incluye también a cualquier función logarítmica), pero cuando la complejidad del problema depende exponencialmente de su tamaño, es decir, cuando no se puede limitar a una expresión polinomial, como en el caso de las funciones exponenciales, entonces el problema es considerado como *Intratable*.

El problema de intratabilidad no se resuelve simplemente esperando a que podamos conseguir un equipo de cómputo mucho más poderoso. En la tabla 14.2 se compara el tamaño de un problema que puede ser resuelto en el lapso de una hora, a la velocidad actual de un microsegundo por paso, contra los tamaños de problemas que podrían resolverse con equipos muchas veces más veloces en ese mismo lapso, para las diferentes funciones de complejidad:

	Tamaño del problema a resolver en una hora		
	A la velocidad actual	**100 veces más rápido**	**1000 veces más rápido**
n	3.6×10^9	3.6×10^{11}	3.6×10^{12}
n^2	6×10^4	6×10^5	1.9×10^6
n^3	1533	7114	15326
n^5	82	205	325
2^n	31.7	38.4	41.7
3^n	20	24.2	26.3

Tabla 14.2

Como se puede apreciar, para las funciones exponenciales el incremento en el tamaño de los problemas a resolver es insignificante con respecto al aumento en velocidad del equipo disponible, de ahí su clasificación como *Intratables*.

Ejemplo

Como se mencionó en el capítulo 12, el lenguaje siguiente es Decidible:

$$L_{GIC} = \{ \langle G, w \rangle \mid G \text{ es una } \textbf{GIC} \text{ que genera la cadena } w \}$$

Primero debemos encontrar una gramática en la **FNCh** equivalente, y luego probar todas las derivaciones posibles de longitud 2n – 1 o menos.

El problema radica en que usando esta técnica, el número de derivaciones posibles crece de forma exponencial, por lo que no es tratable de esta manera.

Sin embargo, podemos replantear este problema desde la perspectiva de la Programación Dinámica, cuya idea básica es la de reutilizar soluciones parciales ya obtenidas, y así evitar la repetición innecesaria de muchos pasos. Este criterio es el fundamento del procedimiento que hemos descrito en el capítulo 7 como el algoritmo **CYK**, el cual es Polinomial de orden $O(n^3)$.

Cálculos de tiempo polinomial

Volvemos a estudiar los problemas de reconocimiento de lenguajes, pero desde el punto de vista de su complejidad: Nos interesa saber si el procesamiento de un lenguaje es una tarea práctica, una vez que sabemos que es teóricamente posible. Decimos que una máquina de Turing M, calcula la función f en un tiempo polinomial, si existe un polinomio p(x) tal que para cualquier entrada w (para la cual está definida f(w)), M calcula f(w) en no más de p(|w|) pasos.

Una propiedad importante de las funciones que pueden calcular las máquinas de Turing en tiempo polinomial, es que la *composición* de dos de estas funciones también se calcula en tiempo polinomial.

También es cierto que la clase de las funciones que pueden calcularse en tiempo polinomial con máquinas de Turing de varias cintas, es igual que la de las máquinas de una cinta.

La clase *P*

Definimos que ***P*** es la clase de los lenguajes que las máquinas de Turing pueden decidir en un tiempo polinomial. Es decir, son todos los lenguajes que pueden ser aceptados en un *tiempo razonable*. Otra característica de la clase ***P***, es que permanece estable para una amplia gama de sistemas computacionales.

Teorema

Si una máquina de Turing decide un lenguaje L en tiempo polinomial, entonces existe otra máquina que también lo decide en tiempo polinomial pero que indica su aceptación deteniéndose con la configuración de cinta **#1##** y su rechazo con la configuración **#0##**.

Problemas de decisión

Un problema de decisión es aquel problema que puede expresarse en forma de una pregunta cuya respuesta es *sí* o *no*. Desde esta perspectiva, la clase ***P*** corresponde a la clase de los Problemas de Decisión que pueden resolverse con Máquinas de Turing en tiempo polinomial.

Tanto es así, que con frecuencia consideramos a ***P*** más que una clase de lenguajes Decidibles, como una clase de problemas de decisión.

También vimos que había lenguajes que eran aceptados por máquinas de Turing, pero no eran Decidibles. Deducimos por tanto, que estos lenguajes no se encuentran en la clase ***P***.

La clase *NP*

Definimos que ***NP*** es la clase de los lenguajes que las máquinas de Turing no deterministas pueden aceptar en tiempo polinomial. Por supuesto, ya que cualquier máquina de Turing determinista está contenida en la clase de las máquinas de Turing no deterministas, podemos afirmar que $\boldsymbol{P} \subseteq \boldsymbol{NP}$.

Sin embargo, la cuestión de que si $\boldsymbol{P} = \boldsymbol{NP}$, aún no se ha resuelto. Existen numerosos problemas de reconocimiento de lenguajes que se sabe que pertenecen a ***NP***, pero se desconoce su pertenencia (o no pertenencia) a ***P***. Si descubriéramos que $\boldsymbol{P} = \boldsymbol{NP}$, podríamos afirmar que estos problemas tienen soluciones prácticas. En caso de probarse lo contrario, la posibilidad de encontrar soluciones eficientes, se desvanece.

No se ha podido resolver tampoco la relación entre la aceptación y la decisión de lenguajes en tiempo polinomial en el contexto de las máquinas de Turing no deterministas (Sin embargo, sí se determinó para el caso de la clase ***P***).

Un ejemplo de este tipo de problemas es el conocido como El Problema del Agente Viajero. En él se define un lenguaje L_s que consiste en las cadenas que contienen una ruta válida. Este lenguaje puede *aceptarse* por una máquina no determinista en tiempo polinomial (la máquina se detiene si la ruta es suficientemente corta. En caso contrario, el tiempo crece considerablemente), pero no puede afirmarse que este lenguaje pueda ser *decidido* por una máquina no determinista.

Todos los algoritmos conocidos en la actualidad deben revisar todos los posibles caminos, y este chequeo es exponencial, sin embargo tampoco se ha probado que no exista algún algoritmo tratable para resolver este problema.

Los problemas de clase ***NP***, son verificables polinomialmente, un verificador es un lenguaje dado por: $L = \{ w \mid$ El algoritmo V acepta a $\langle w, c \rangle$ para una cadena $c \}$ el verificador Polinomial es de orden $O(|w|)$, donde la cadena c es la clave para probar que $w \in L$.

Ejemplo

Resulta muy difícil encontrar los factores del número 4,294,967,297, incluso Pierre de Fermat declaró en 1640 que se trataba de un número primo, y no fue sino hasta 1732 que Euler demostró que 6,700,417 y 641 son factores de ese número, lo que nos resulta ahora muy sencillo de verificar.

Si se pudiera demostrar que ***P*** = ***NP***, entonces L_s pertenecería a ***P*** y podría ser decidido en tiempo polinomial y entonces resolveríamos en un tiempo razonable al Problema del Agente Viajero y otros semejantes.

Reducciones Polinomiales

Demostrar que uno de los muchos ejemplos que sabemos que pertenecen a ***NP*** pertenece también a ***P***, sería un pequeño paso hacia la resolución de la relación entre ***P*** y ***NP***. Sin embargo, existe un camino para atacar muchos problemas a la vez. Este método se basa en el concepto de completez-***NP***, que a su vez se apoya en el concepto de las reducciones polinomiales.

Una reducción polinomial (o transformación polinomial) de un lenguaje a otro, es una función que puede ser calculada por una máquina de Turing en tiempo polinomial, y para la cual $w \in L_1$ si y sólo si $f(w) \in L_2$. Si existe una reducción polinomial de L_1 a L_2, decimos que L_1 se reduce a L_2 y lo escribimos como $L_1 \propto L_2$.

De esta forma, si tenemos una máquina M_f, que ejecuta una función f la cual es una reducción polinomial de L_1 a L_2, y si tenemos una segunda máquina M_2, que acepta el lenguaje L_2, entonces el lenguaje que acepta la máquina compuesta M_fM_2 es L_1 (M_f convierte L_1 en L_2, y L_2 es aceptado por M_2)

Además, como M_f se calcula en tiempo polinomial, la complejidad temporal de la máquina compuesta es comparable con la de M_2. Esta afirmación justifica el siguiente teorema:

Teorema

Si $L_1 \propto L_2$, y L_2 está en ***P***, entonces L_1 está en ***P***.

Una de las principales aplicaciones de este teorema es para demostrar la pertenencia (o no pertenencia) a ***P*** de ciertos lenguajes: Si podemos demostrar la reducción polinomial de un lenguaje L_1 a otro L_2, entonces ambos pertenecen (o ninguno pertenece) a ***P***, con solo demostrar que L_2 se halla (o no se halla) en ***P***.

Teorema de Cook

Este teorema, descubierto por S. A. Cook, identifica que existe un lenguaje de la clase ***NP*** (problemas de decisión) al que se puede reducir por reducción polinomial cualquier otro lenguaje en ***NP***.

Desde el descubrimiento del teorema de Cook, se han encontrado que existe un gran número de lenguajes en ***NP*** que son reducciones polinomiales de cualquier otro lenguaje en ***NP***. A esta clase de lenguajes se le conocen como completos-***NP***.

Así, si alguna vez se demuestra que uno de estos lenguajes se halla en ***P***, todos los lenguajes en ***NP***, se hallarán en ***P***, porque la clase de lenguajes completos-***NP*** está formada por los lenguajes más difíciles de la clase **NP**, ver Figura 14.2.

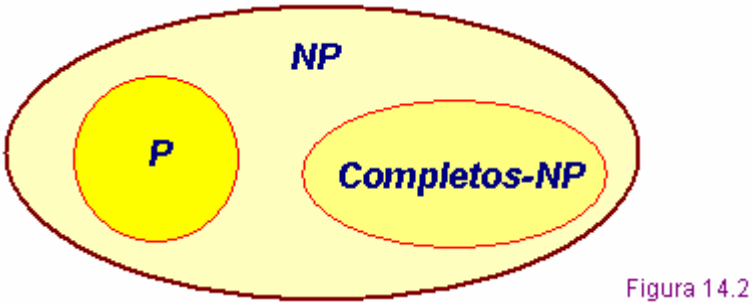


Figura 14.2

Si se demostrara que alguno de estos lenguajes se puede aceptar por una máquina de Turing determinista en tiempo polinomial, entones ***NP*** debería ser igual a ***P***, y muchos de los problemas que ahora no tienen solución práctica (por tener una complejidad que los hace intratables), podrían ser finalmente Tratables.

Al contrario, si se demostrara que algún lenguaje de ***NP*** yace fuera de ***P***, entonces sería inútil seguir buscando soluciones eficientes para dichos problemas. Hasta el momento, no se ha demostrado ninguna de estas opciones.

Comentarios Finales

Vimos los autómatas finitos, que aceptaban los lenguajes regulares. Con el objetivo de desarrollar técnicas más potentes para el reconocimiento de lenguajes, llegamos al estudio de los autómatas de pila y los lenguajes independientes de contexto. Y posteriormente, se estudió las máquinas de Turing, y los lenguajes estructurados por frases donde llegamos al aparente límite identificado por la Tesis de Church-Turing.

Ampliamos el uso de las máquinas de Turing, que dejaron de ser simples dispositivos de reconocimiento de lenguajes para convertirse en máquinas de propósito general que calculaban funciones. De esta forma, vimos que la clase de funciones que era calculable por una máquina de Turing era la clase de las funciones recursivas parciales, que coincide con las funciones calculables con un sencillo lenguaje de programación esencial. Así vimos la equivalencia entre diferentes disciplinas de la tesis de Church-Turing.

Una vez investigada la frontera entre la computabilidad y la no computabilidad, nos limitamos a los problemas que en teoría pueden resolverse mediante algoritmos, y estudiamos la complejidad de la mejor de las soluciones. Esto nos llevó a la clase ***P***, como la clase de los problemas que pueden resolverse por una máquina de Turing determinista en tiempo polinomial. La clase ***NP*** por el contrario, es la clase de los problemas que pueden resolverse por una máquina de Turing no determinista en tiempo polinomial. Una de las preguntas que queda sin resolver es si la clase ***P*** y ***NP*** son idénticas o no.

Preguntas

a) ¿En qué nos basamos al afirmar que existen más problemas que soluciones posibles?

b) ¿Por qué usualmente es más importante estudiar la intratabilidad de problemas en el tiempo, más que en el espacio?

c) ¿En que casos es relevante analizar la complejidad en el espacio?

Ejercicios Capítulo 14

14.1 Dadas las funciones $f(n) = 3n^3 + 2n + 5$ y $g(n) = n^3$, demuestre que $f \in O(g)$.

14.2 Observe la máquina de Turing, con alfabeto Σ = { **x**, **y** }, mostrada en la Figura 14.3, de la página siguiente, describa su comportamiento y señale los valores máximo y mínimo de su complejidad temporal.

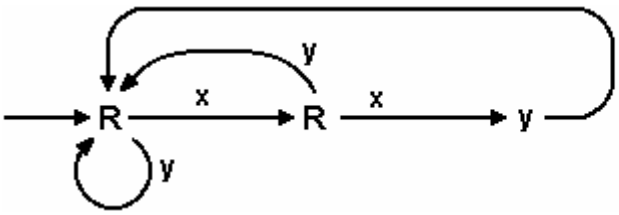


Figura 14.3

14.3 Demuestre que si una MT requiere un espacio de O(f(n)), entonces necesita un tiempo $2^{O(f(n))}$.

14.4 Para cada una de las siguientes funciones, elija la “mejor O” de acuerdo a la tabla que describe la velocidad de crecimiento de la función:

a) $6n^2 + 500$

b) $2n^2 + n^2 \log_2 n$

c) $\lfloor (n^3 + 2n)(n + 5)/n^2 \rfloor$

d) $n^2 2^n + n!$

e) $25n^{3/2} + 5n^2 + 23$

14.5 Utilizar la definición de O(g), para establecer las siguientes relaciones:

a) $n \log_2 n \in O(n^2)$

b) $n^2 \notin O(n \log_2 n)$

c) $n^r \in O(2^n)$

d) $n! \notin O(2^n)$

14.6 Determinar la complejidad temporal de las siguientes Maquinas de Turing:

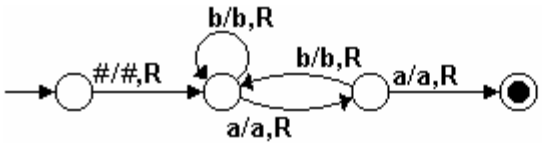


Figura 14.4

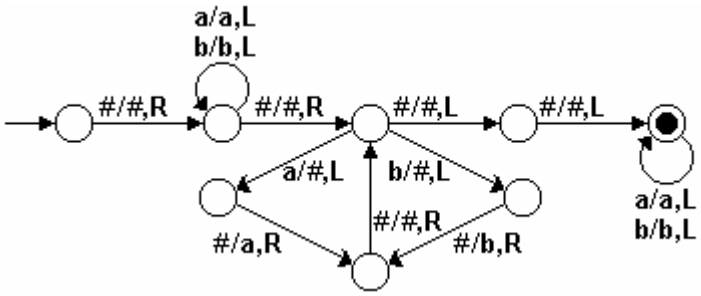


Figura 14.5

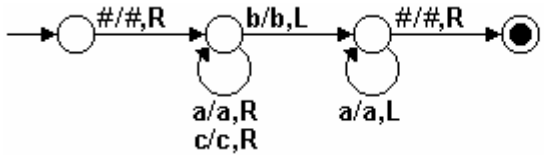


Figura 14.6

Apéndice

A

Glosario de Términos

Alfabeto

Un alfabeto es un conjunto ordenado, finito no vacío de símbolos.

Árbol

Un árbol es un dígrafo que tiene las siguientes propiedades:

- Existe un único vértice llamado raíz, que no tiene predecesores y desde el cual existen trayectorias hacia cada vértice.
- Cada vértice, menos la raíz, tiene un solo predecesor.
- Los sucesores de un vértice se ordenan de izquierda a derecha.
- Los árboles se trazan poniendo la raíz en la parte superior y los arcos en dirección descendente (por eso no se requiere poner la flecha)
- A los vértices que no tienen descendientes se les denomina Hojas.

Árbol de Derivación

Un árbol de derivación es un árbol que representa en forma gráfica la manera como una gramática dada genera a una cadena concreta y tiene las siguientes propiedades:

- La etiqueta de la raíz es S.
- La etiqueta de cualquier vértice que no es hoja es un símbolo no terminal.
- La etiqueta de cualquier vértice hoja es un símbolo terminal o ε.
- Para cada producción de la forma $A \rightarrow w$, que se aplique en una derivación, deberá haber un nodo etiquetado con el **SNT** A, cuyos descendientes correspondan a cada uno de los símbolos que forman la cadena *w* (en el mismo orden).
- La concatenación de los símbolos que haya en cada una de las hojas (leídos de izquierda a derecha) nos proporciona la cadena obtenida en la derivación que representa el árbol.

Autómata Finito

Un autómata finito es un conjunto finito de estados y un conjunto de transiciones entre estado y estado que pueden ocurrir, dependiendo del símbolo de entrada que se elija dentro de un alfabeto $\sum$.

Cadena

Una Cadena es una secuencia finita de símbolos de un alfabeto dado yuxtapuestos uno a continuación de otro en una secuencia determinada.

Cadena Inversa

La Inversa de una Cadena *w* es la cadena w^R, tal que es la imagen refleja de *w*, es decir, que equivale a *w* cuando se lee de derecha a izquierda.

Cerradura de Kleene

Sea L un lenguaje sobre el alfabeto $\sum$, se define a L* como la Cerradura de Kleene o Cerradura Estrella como la unión infinita de todas las potencias de L, incluyendo la potencia cero.

Cerradura Positiva

Se define a L^+, la Cerradura Positiva de L, como la unión infinita de todas las potencias de L a partir de uno.

Ciclo

Un Ciclo es cuando existe una trayectoria de *k* vértices de un grafo: v_1, v_2, …, v_k, tal que $v_1 = v_k$, para $k > 1$.

Concatenación

Concatenación es la yuxtaposición de dos cadenas, una a continuación de la otra, de tal forma que si *w* y *x* son dos cadenas, la concatenación de *w* con *x* es la cadena que se obtiene de añadir la cadena *x* a la cadena *w*.

Diagrama de Transición

Un dígrafo es un diagrama de transición si está asociado a un Autómata Finito de la siguiente manera: Los vértices del dígrafo representan los posibles estados del **AF**,

Una transición desde el estado *q* al estado *p* se representa por un arco etiquetado con el valor del símbolo de entrada que la provoca.

Dígrafo

Un dígrafo es un conjunto de vértices y un conjunto de pares ordenados llamados arcos y se denota como $v \rightarrow w$. Al vértice v se le llama Predecesor (Padre) y al vértice w se le llama Sucesor (Hijo).

Ejemplo: $G = (\{1,2,3,4\}, \{ i \rightarrow j \mid i < j \})$

Estado

Se llama estado a cada una de las distintas configuraciones internas en que puede estar un sistema en un momento dado.

Estado Inicial

Usualmente se denota como q_0 al estado inicial y es en el que se asume se encuentra un Sistema al principio.

Estados Finales

A los estados de aceptación también se les conoce como estados finales.

Grafo

Un Grafo G = { V, E } es un conjunto finito V de vértices o nodos y un conjunto E de pares de vértices conectados, llamados aristas.

Ejemplo: V = { 1, 2, 3, 4, 5 }, E = { (n, m) | n + m = 4 ó n + m = 7 }, en este caso, el par (1, 3) y el par (3, 1) representan a la misma arista.

Gramáticas de Estructura de Frase

Una *gramática de estructura de frase* es aquella cuyo lado izquierdo de las reglas de producción puede estar formado por cualquier cadena no vacía sobre $N \cup \sum$, siempre que contengan al menos un símbolo no terminal. Las producciones son de la forma: $A \rightarrow w$, donde $A \in (N \cup \sum)^* N (N \cup \sum)^*$ y $w \in (N \cup \sum)^*$. Este tipo de gramáticas se conocen también como *Gramáticas No Restringidas* o *Gramáticas del Tipo Cero*.

Gramáticas Sensibles al Contexto

Las *Gramáticas Sensibles al Contexto* son aquellas cuyas producciones son de la forma $A \rightarrow w$, donde $|A| \leq |w|$.

Además, la forma normal de estas gramáticas establece que toda producción debe ser de la forma: $\alpha_1 A \alpha_2 \rightarrow \alpha_1 w \alpha_2$, con $w \neq \varepsilon$. Estas producciones permiten que A sea reemplazado por *w* sólo cuando A aparezca en el contexto de α_1 y α_2.

A estas gramáticas se les llama también *Gramáticas Dependientes del Contexto*.

Hoja

Un vértice de un árbol que no tiene sucesores se denomina hoja.

Lazo

Se llama lazo a un ciclo en un grafo, de longitud uno.

Lenguaje

Un lenguaje formal es un conjunto de palabras o cadenas formadas por símbolos de un alfabeto dado.

Lenguaje Decidible

Un lenguaje Decidible es aquel que puede ser decidido por alguna Máquina de Turing, también se les llama Lenguajes Recursivos.

Lenguaje Enumerable

Un lenguaje Enumerable o Recursivamente Enumerable puede ser aceptado pero no necesariamente decidido por alguna Máquina de Turing.

Lenguaje Universal

Definimos al Lenguaje Universal sobre $\sum$ como el lenguaje formado por todas las cadenas que se pueden formar sobre $\sum$, también se le conoce como la cerradura de $\sum$, y se denota como $\sum^*$. Para cualquier alfabeto, $\sum^*$ es siempre infinito.

Lenguaje Vacío

El lenguaje vacío se denota como $\varnothing$, y es el que no contiene ninguna cadena, es decir $\varnothing = \{\ \}$,

Símbolo

Un símbolo es un signo, dígito, letra o incluso un grupo de letras que se utiliza en algún lenguaje y que tiene algún significado convencional.

Trayectoria

Una trayectoria es una sucesión de *k* vértices: v_1, v_2, …, v_k, tales que exista una arista entre dos elementos sucesivos que los una, (v_i, v_{i+1}), para todo $1 \leq i < k$.

Apéndice

B

Solución a Problemas Planteados

1.1. $A \cup B = \{ \psi, \eta, \lambda, \phi, \theta \}$, $A \cap B = \{ \lambda \}$, $A \oplus B = \{ \psi, \eta, \phi, \theta \}$, $A - B = \{ \psi, \eta \}$ y $B - A = \{ \phi, \theta \}$

1.2. Prefijos propios: ε, **p**, **pi**, **pin**. Sufijos propios: ε, **o**, **no**, **ino**.
Subcadenas: ε, **p**, **i**, **n**, **o**, **pi**, **in**, **no**, **pin**, **ino**.

1.3. w^2 = **papapapa**, w^3 = **papapapapapa** y w^R = **apap**.

1.4. Prefijos: ε, **p**, **pi**, **piñ**, **piña**, **piñat**, **piñata**.

1.5. Sufijos: ε, **a**, **ma**, **oma**, **roma**, **aroma**, **maroma**.

1.6. Subcadenas: ε, **b**, **a**, **n**, **ba**, **an**, **na**, **ban**, **ana**, **nan**, **bana**, **anan**, **nana**, **banan**, **anana**.

1.7. xy^Rz = **piñataamorambanana**, z^2x = **bananabananapiñata**.

1.8. Subcadenas: ε, **0**, **1**, **2**, **01**, **11**, **10**, **02**, **22**, **20**, **011**, **111**, **110**, **102**, **022**, **220**.

1.9. $(A \cup B^2)$ = { **a**, **b**, **c**, **cc**, **dc**, **ec**, **cd**, **dd**, **ed** , **ce**, **de**, **ee** }
$(AB)^*$ = { ε, **ac**, **ad**, **ae**, **bc**, **bd**, **be**, **cc**, **cd**, **ce**, **acac**, **acad**, … }
$(BA)^R$ = { **ac**, **ad**, **ae**, **bc**, **bd**, **be**, **cc**, **cd**, **ce** }.

1.10. $L_1 \cdot L_2$ = { ε, **1**, **01**, **11**, **0**, **001**, **011**, **10**, **101**, **1001**, **1011**, **111**, **1101**, **1111**},
$L_2 \cdot L_1$ = { ε, **0**, **10**, **11**, **1**, **110**, **111**, **01**, **010**, **0110**, **0111**, **1110**, **1111**},
$L_1 \cup L_2$ = { ε, **0**, **10**, **11**, **1**, **01**}, $L_1 \cap L_2$ = { ε, **11**}, $L_1 - L_2$ = { **0**, **10** },
$L_2 - L_1$ = { **1**, **01** }, L_1^* = { ε, **0**, **10**, **11**, **00**, **010**, **011**, **100**, **1010**, **1011**, …},
L_2^* = { ε, **1**, **01**, **11**, **101**, **111**, **011**, **0101**, **0111**, …} y $L_1 \oplus L_2$ = { **0**, **10**, **1**, **01** }

1.11. L^0 = { ε }, L^1 = { ε, **a** }, L^2 = { ε, **a**, **aa** }, L^3 = { ε, **a**, **aa**, **aaa** }.

1.12. { **b**, **ab**, **aab**, **aaab**, … }, { **a**, **ab**, **abb**, **abbb**, … } y { ε, **ab**, **abab**, **ababab**, …}

1.13. $L_1 \cup L_2$ = { ε, **aa**, **ab**, **bb** }, $L_1 \cup L_3$ = { ε, **aa**, **ab** }, $L_1 \cup L_4$ = { ε },
$L_2 \cup L_4$ = {**aa**, **ab**, **bb**}, $L_1 \cap L_2 = \varnothing$, $L_2 \cap L_3$ = {**aa**, **ab**}, $L_3 \cap L_4 = \varnothing$ y $L_1 \cap L_4 = \varnothing$.

1.14. $(A \cup B^2)$ = { **ab**, **b**, **cb**, **aa**, **aba**, **baa**, **baba** }, $(B \cup A)^R$ = { **a**, **ab**, **ba**, **b**, **bc** }, (AB) = { **aba**, **ba**, **cba**, **abba**, **bba**, **cbba** }, $(A^2 \cap BA)$ = { **bab** }, $(A \oplus B^R)$ = { **b**, **cb**, **a** } y $(A^R - B)^2$ = { **bb**, **bbc**, **bcb**, **bcbc** }.

1.15. $(L_1 \cup L_2)^R$ = { **110**, **101**, **11**, **10** }. $(L_2 - L_1)^2$ = {**011011**, **011101**, **101011**, **101101**}. $(L_1 - L_2)^+$ = { **01**, **0101**, **010101**, … }. $(L_1 \cap L_2)^*$ = { ε, **11**, **1111**, **11111**, … }. $L_1{}^R L_2$ = { **10011**, **10101**, **1011**, **11011**, **11101**, **1111**}.

1.16. L^C = { **b**, **ab**, **bb**, **aab**, **abb**, **bab**, **bba**, **bbb**, …}.

2.1. a) **00** (**1** ∪ **0**)*, b) **1***(**01*01*01***)*, c) (**1** ∪ **0**)* **11** (**1** ∪ **0**)*,
d) **1*** ∪ **1*01*** ∪ **1*01*01***, e) **1***(**01**$^+$ ∪ **001**$^+$)* **000** (**1**$^+$**0** ∪ **1**$^+$**00**)***1***,
f) **0** ∪ (**1** ∪ **2** ∪ … ∪ **9**)Σ*, g) (ε ∪ **1** ∪ **0**)^5, h) (**0** ∪ **1**)* (**11** ∪ **0**) ∪ **1** ∪ ε,
i) (**01** ∪ **1**)$^+$, j) ((**0** ∪ **1**) ^{5})*, k) (**11** ∪ **00**)(**1** ∪ **0**)* ∪ (**1** ∪ **0**)*(**11** ∪ **00**).

2.2. a) Las cadenas formadas por una cantidad par de ceros, incluyendo la vacía.
b) Las cadenas que inician con cualquier cantidad (incluso ninguno) de ceros seguida por cualquier cantidad (incluso ninguno) de unos.
c) Las cadenas de ceros y unos que inician con uno.
d) Las cadenas de ceros y unos que terminan con doble cero.
e) Las cadenas de ceros y unos que contienen la subcadena 10.
f) Las cadenas de ceros y unos que contienen al menos dos ceros.

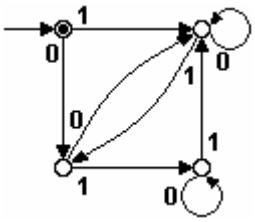
2.3. a) No, b) Si, c) No, d) Si, e) No.

2.4. L = { **a**, **ca**, **cca**, **ccca**, …, ε, **bc**, **bcbc**, **bcbcbc**, …, **b**, **bb**, **bbb**, **bbbb**, … }

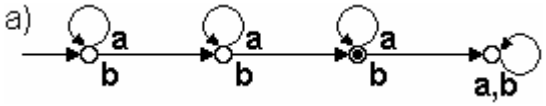
2.5. 18

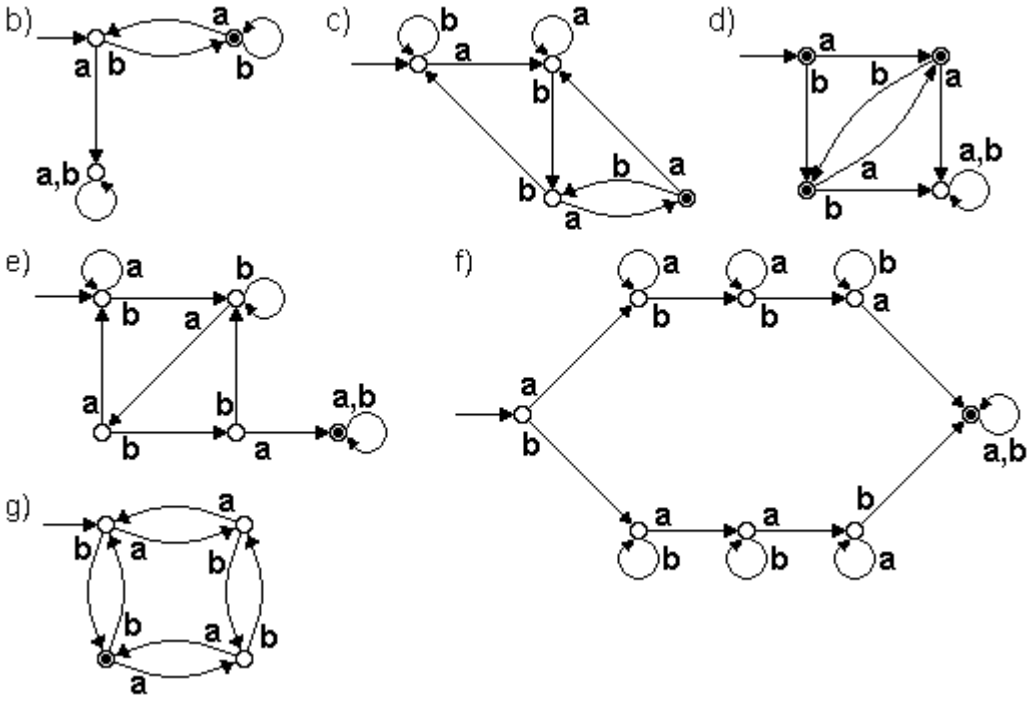
2.6. a) (**ab**)*, b) (**aa**)*, c) **a***, d) **a***, e) (**aa**)*, f) **a***, g) (**b***a**)$^+$**b***, h) **a**(**aa**)*, i) (**a** ∪ **b**)*
j) (**a** ∪ **b**)*, k) (**a** ∪ **b**)*, l) **a***bb**$^+$**c***, m) **y**$^+$**x***, n) **x***, o) (**a** ∪ **b**)*, p) (**a** ∪ **b**) (**aa**)*,
q) (**a** ∪ **ba** ∪ **b**^{2})***b**, r) (**b***a)*, s) (**abc***)*

3.1.

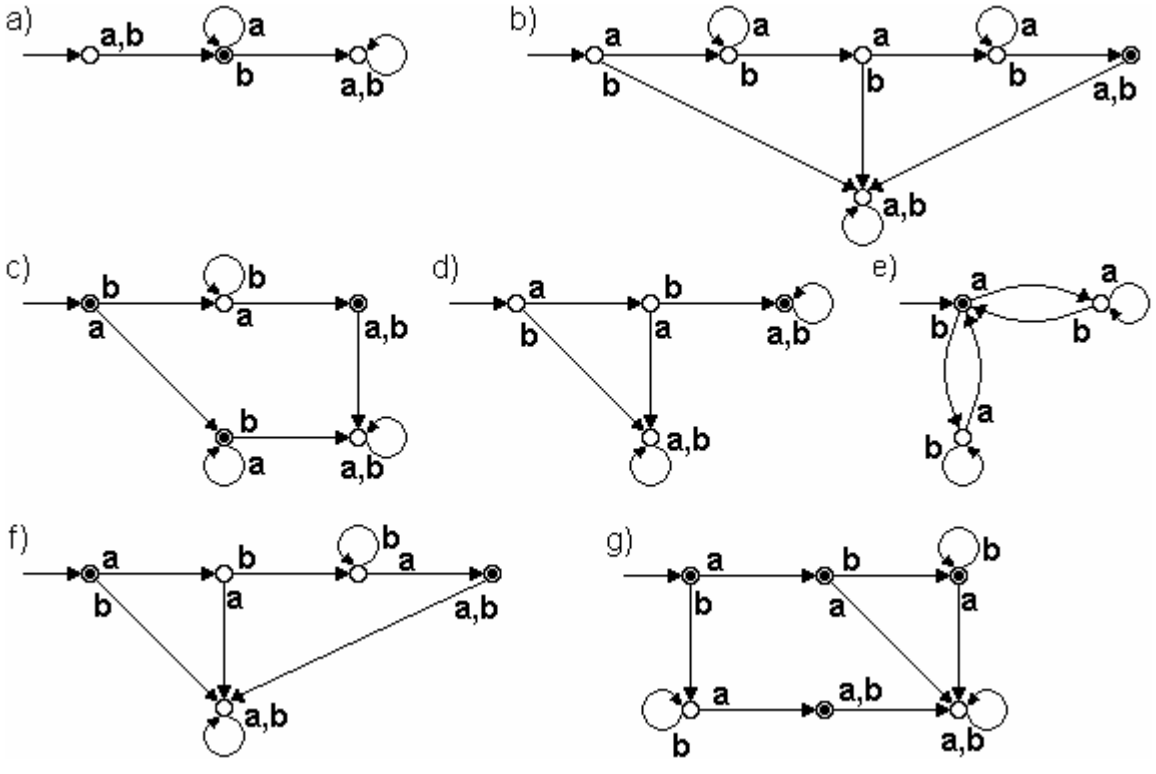


3.2.

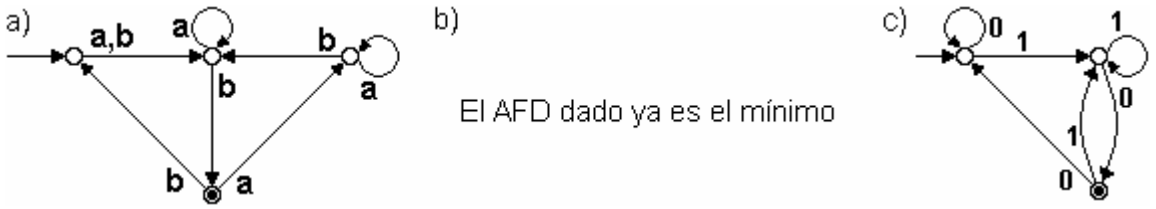




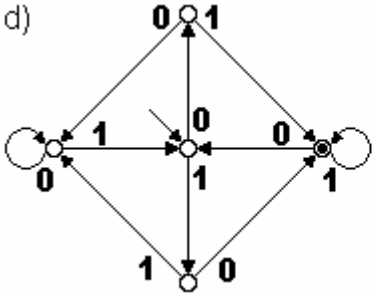
3.3.



3.4.



d)



e)

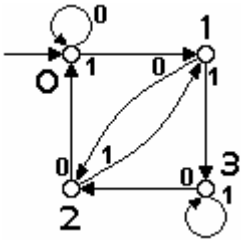
El AFD dado ya es el mínimo

3.5. No son equivalentes.

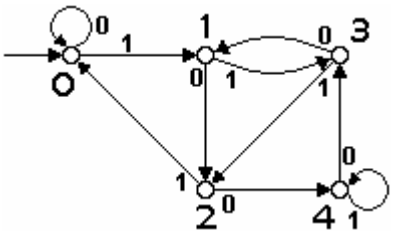
3.6. Si son equivalentes.

3.7. Si son equivalentes.

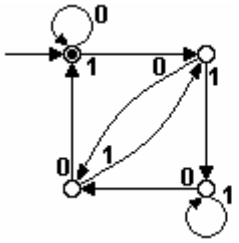
3.8.



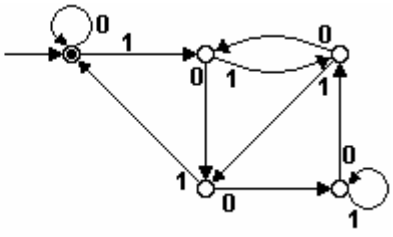
3.9.



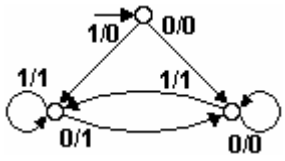
3.10. a)



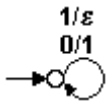
b)

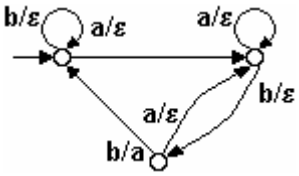


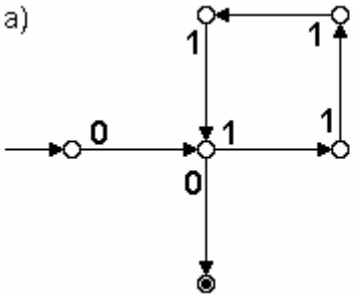
3.11.

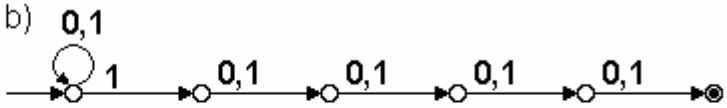


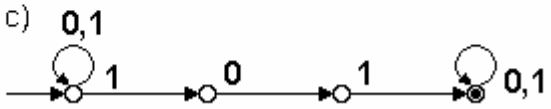
3.12.

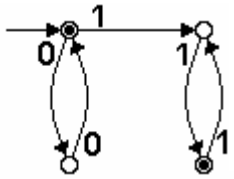


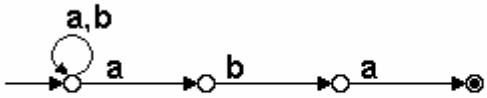
3.13. 

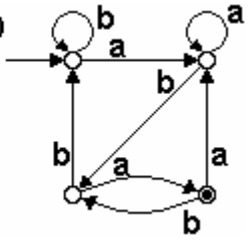
4.1. a) 

b) 

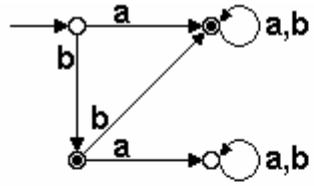
c) 

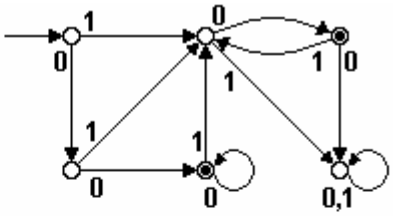
d) 

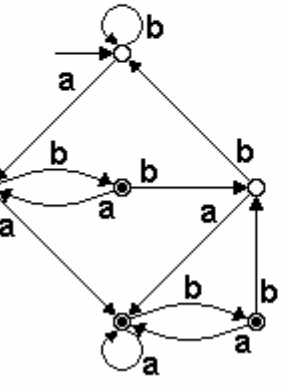
4.2. AFN 

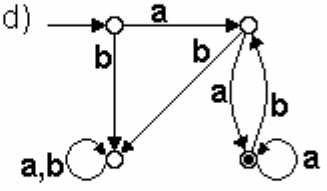
AFD 

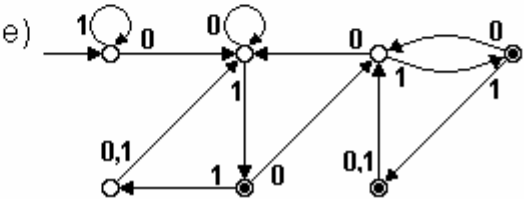
4.3. a) No, b) Sí, c) No, d) Sí, e) No.

4.4. a) 

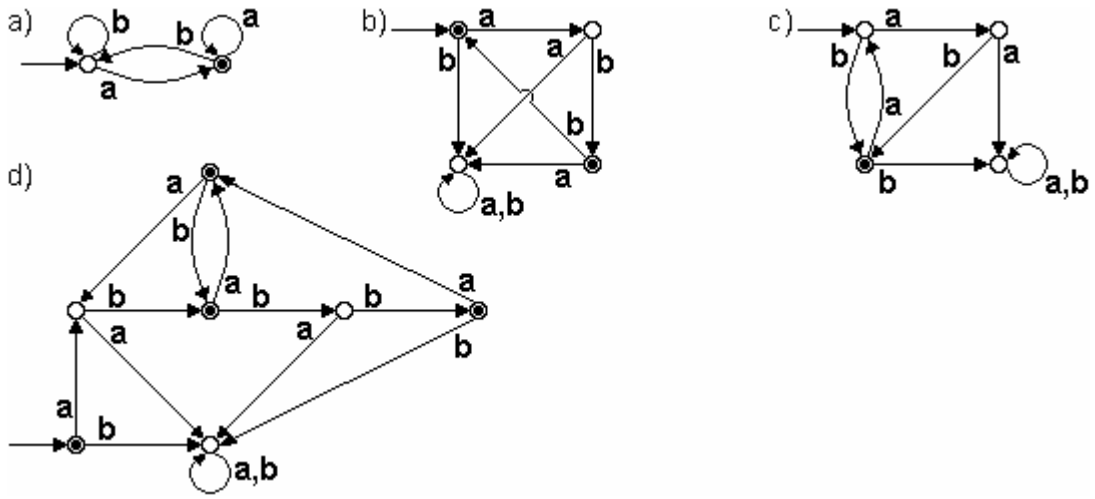
b) 

c) 

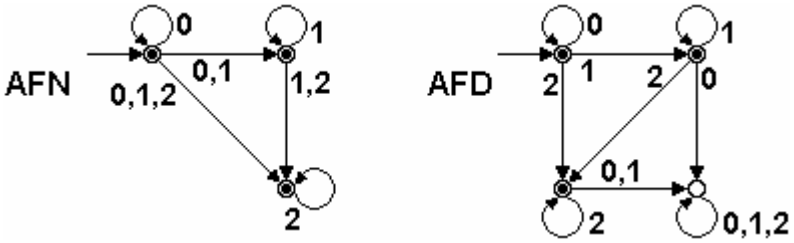
d) 

e) 

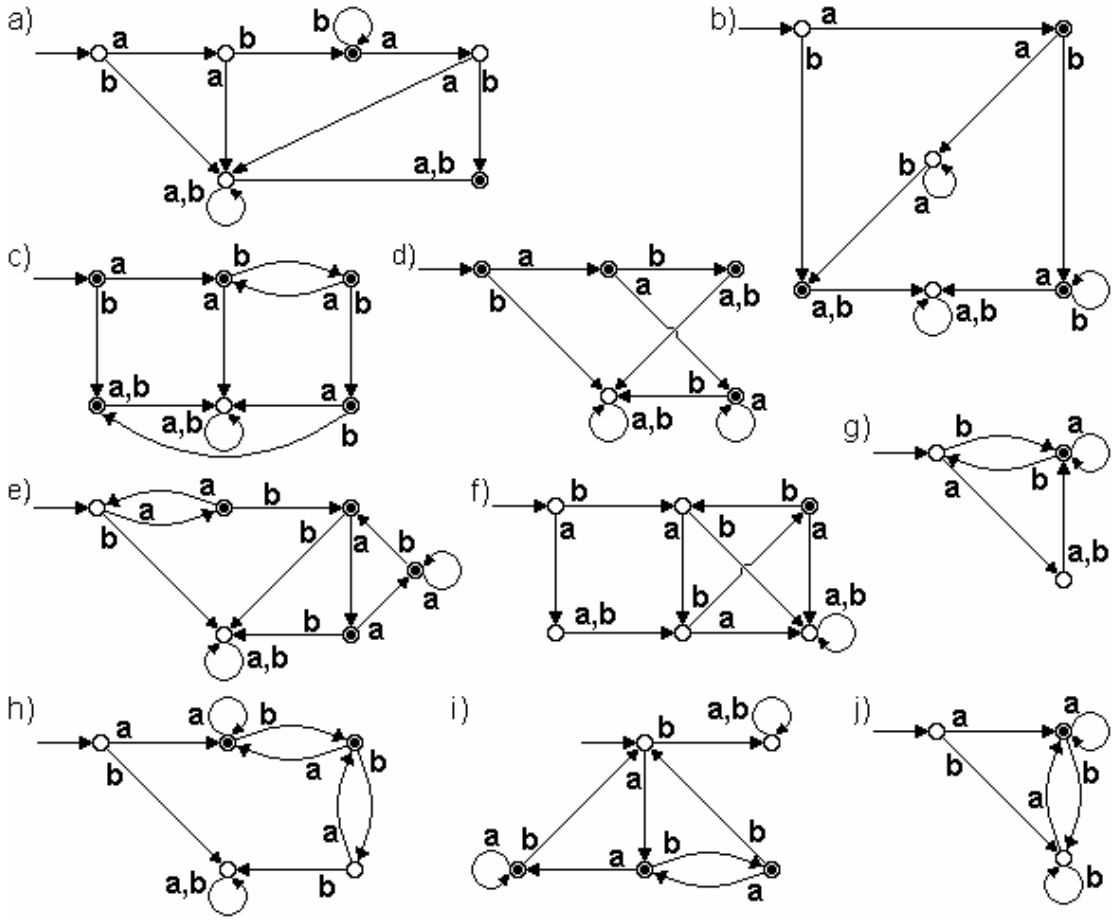
4.5.



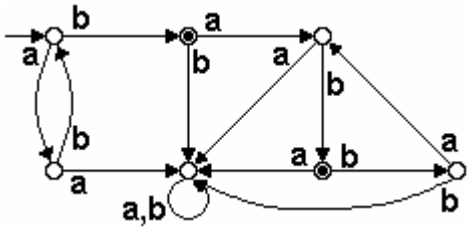
4.6.



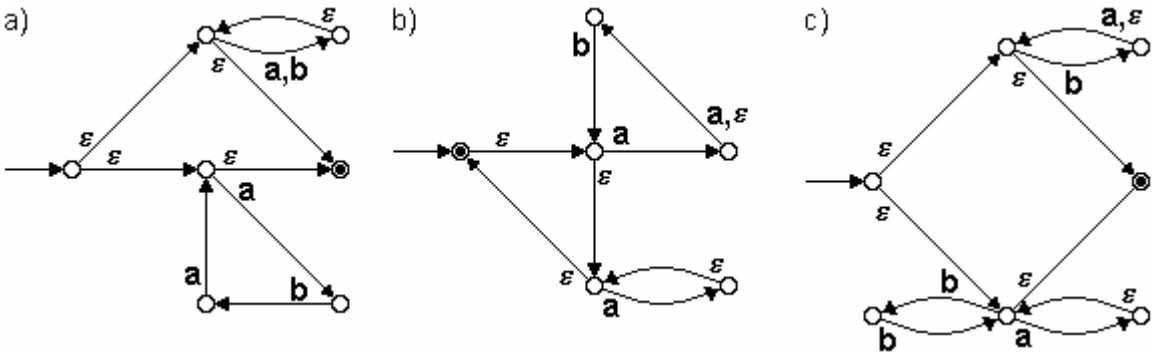
4.7.

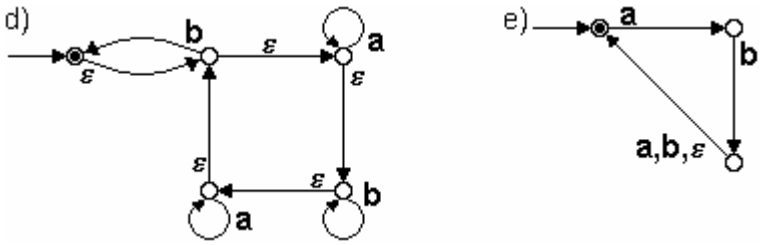


4.8.

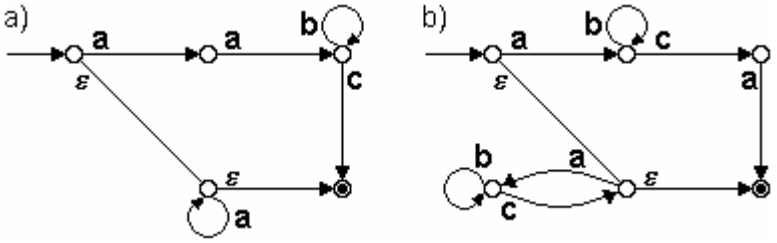


5.1.

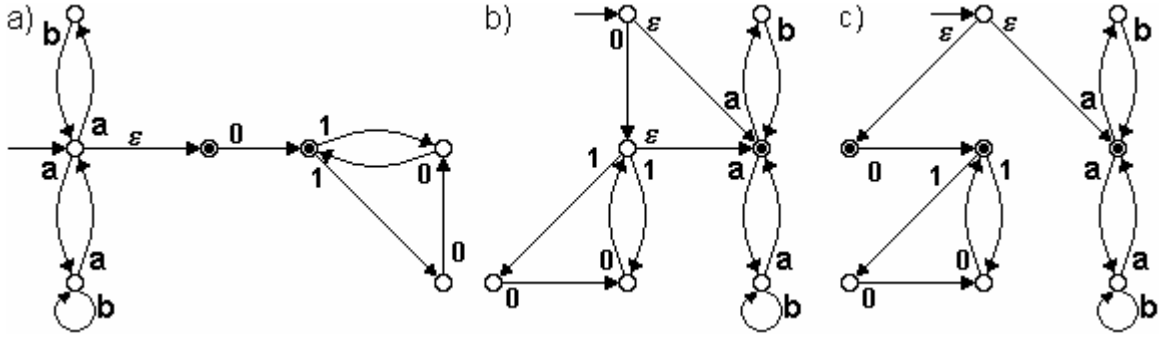




5.2.

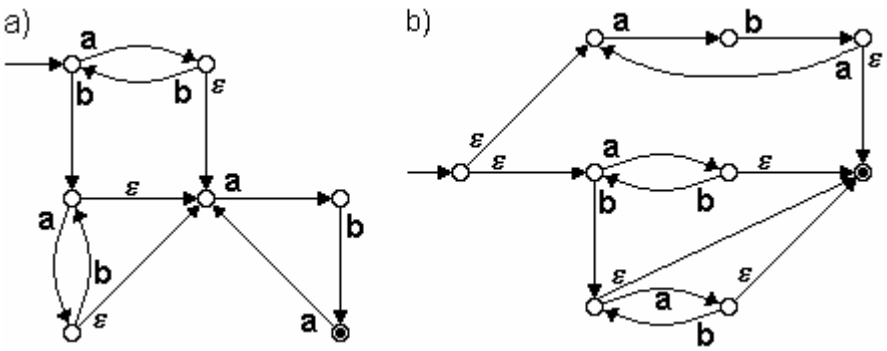


5.3.



5.4. L = (**ab**)* ((**ba**)***b** ∪ (**ba**)$^{+}$ ∪ **a**)

5.5. L = **ab**(**aab**)* = (**aba**)* **ab**

5.6. 

5.7. a) L = (**a** ∪ **b**)***ba**, b) L = (**a** ∪ **ba***b**)*, c) L = (**a** ∪ **ba** ∪ **bba**)*(**bb** ∪ **b** ∪ ε)
d) L = (**a** ∪ **ba** ∪ **bba**)* **bbb** (**a** ∪ **b**)*, e) L = (**11** ∪ **0**)(**1** ∪ **0**)*

5.8. a) L = (**a** ∪ **b**(**b** ∪ **ab**)* **a**^{2})* = (**a** ∪ **b**$^+$(**ab**$^+$)* **a**^{2})* = ((**b**$^+$**a**)* **a**)*
b) L = (**a** ∪ **ba** (**a**^{2} ∪ **b**)* **ab** ∪ **b**^{2})* = (**a** ∪ **b** (**ab***a**)* **b**)*
c) L = **a***b (**ab***a ∪ **ba***b)*
d) L = (**a** ∪ **b**)((**a** ∪ **b**)^2)*
e) L = (**a** ∪ **ba***b)(**ab**)* **b** (**a** ∪ **b**)*
f) L = (**b***a$^+$**b**)(**b**$^+$**a**$^+$**b** ∪ **ab**)* **a**^{2} (**a** ∪ **b**)*

5.9. a) L = (**ab** ∪ **abb**)$^+$
b) L = ((**ab*** ∪ **a***b) **ab**)$^+$
c) L = (**ab***a ∪ **baaba**)*

6.1. a) S → **a**S | **bc**
b) S → **a**A | **b**A, A → **c**A | ε
c) S → **a**S | **b**S | **c**
d) S → **a**S | **b**S | A, A → **c**A | ε
e) S → **b**A | **a**B, A → **b**A | ε, B → **a**C | **b**C, C → **a**C | **b**C | **b**
f) S → **a**A | **b**B, A → **a**A | **ba**, B → **a**B | **b**B | ε

6.2. a) S → **a**S | **baa**
b) S → **a**S | **ba**S | **ac**S | ε
c) S → A | **cb**B, A → **a**A | **bc**C, B → **b**B | ε, C → **c**C | ε
d) S → **a**A, A → **a**A | **b**B, B → **b**C | **a**C, C → **b**C | **a**C | ε
e) S → **b**A | **ab**B, A → **b**A | **a**, B → **a**B | **b**B | ε

6.3. S → **aba**S | **bab**S | ε

6.4. $S \rightarrow \mathbf{101}A$, $A \rightarrow \mathbf{0}A \mid \varepsilon$

6.5. $S \rightarrow \mathbf{ab}A \mid \mathbf{b}C$, $A \rightarrow \mathbf{a}A \mid B \mid \mathbf{a}C$, $B \rightarrow \mathbf{b}S \mid \mathbf{a}D \mid \varepsilon$, $C \rightarrow \mathbf{a}A \mid \mathbf{b}D$, $D \rightarrow \mathbf{a}A \mid \mathbf{ab}D \mid \varepsilon$

6.6. a) $L = (\mathbf{bb} \cup \mathbf{aab} \cup \mathbf{baab})^+$

b) $L = (\mathbf{ab})^* (\mathbf{ab} \cup \mathbf{ba})$

c) $L = \mathbf{b}^* (\mathbf{baa} \cup \mathbf{ba})$

d) $L = (\mathbf{a} \cup \mathbf{b}(\mathbf{b}^+\mathbf{a})^*\mathbf{a}) (\mathbf{a} \cup \mathbf{b})^*$

e) $L = (\mathbf{ab}^*\mathbf{a} \cup \mathbf{b}(\mathbf{b}^+\mathbf{a})^*\mathbf{a})^* \mathbf{ab}^+$

f) $L = (\mathbf{a}^+\mathbf{b} \cup \mathbf{ba} \cup \mathbf{bba}^*\mathbf{b}) (\mathbf{a} \cup \mathbf{b})^* \mathbf{b}$

g) $L = (\mathbf{ab}^*\mathbf{aa} \cup \mathbf{ab}^*\mathbf{aba}^*\mathbf{b} \cup \mathbf{b}) \mathbf{ab}^*\mathbf{b}$

6.7. $L = (\mathbf{bb} \cup \mathbf{ba} \cup \mathbf{b}) (\mathbf{ab} \cup \mathbf{aa} \cup \mathbf{bb})^*$

7.1. a) $S \rightarrow \mathbf{a}$

b) $S \rightarrow \mathbf{a}A \mid \mathbf{b}A \mid \mathbf{a} \mid \mathbf{b}$, $A \rightarrow \mathbf{a}A \mid \mathbf{bb}A \mid \mathbf{a} \mid \mathbf{bb}$

c) $S \rightarrow \mathbf{ab}A$, $A \rightarrow \mathbf{cc}C$, $C \rightarrow \mathbf{a} \mid \mathbf{e}A$

d) $S \rightarrow \mathbf{a} \mid \mathbf{a}A \mid \mathbf{b} \mid \varepsilon$, $A \rightarrow \mathbf{b}$

e) $S \rightarrow \mathbf{a} \mid \varepsilon$

7.2. a) $S \rightarrow AB \mid \mathbf{a}A \mid \mathbf{ab} \mid \mathbf{ab}B \mid \mathbf{aa}$, $A \rightarrow \mathbf{a}A \mid \mathbf{ab} \mid \mathbf{ab}B \mid \mathbf{aa}$, $B \rightarrow \mathbf{b}A \mid BB$

b) $S \rightarrow \mathbf{a}B$, $B \rightarrow \mathbf{a}B \mid \mathbf{e}$

c) $S \rightarrow AB\mathbf{a} \mid A\mathbf{a} \mid \mathbf{a} \mid AA \mid AAA$, $A \rightarrow AB\mathbf{a} \mid A\mathbf{a} \mid \mathbf{a}$, $B \rightarrow AB\mathbf{a} \mid A\mathbf{a} \mid A\mathbf{b}$

d) $S \rightarrow \mathbf{a} \mid \mathbf{b}$

e) $S \rightarrow \mathbf{a}ED \mid \mathbf{a}D \mid E\mathbf{a} \mid \mathbf{a} \mid \mathbf{b}$, $D \rightarrow E\mathbf{a} \mid \mathbf{a} \mid \mathbf{b}$, $E \rightarrow E\mathbf{a} \mid \mathbf{a}$

f) $S \rightarrow DB \mid \mathbf{a}E \mid \mathbf{b}$, $B \rightarrow \mathbf{a}B \mid \mathbf{b}S \mid \mathbf{a}$, $D \rightarrow \mathbf{b}$, $E \rightarrow BE\mathbf{a} \mid E\mathbf{a} \mid D\mathbf{ab}$

g) $S \rightarrow \mathbf{a} \mid \mathbf{a}A \mid A\mathbf{a}$, $A \rightarrow \mathbf{a}B$, $B \rightarrow \mathbf{a} \mid A\mathbf{a}$

h) $S \rightarrow \mathbf{a}A\mathbf{b}$, $A \rightarrow \mathbf{ee}C$, $C \rightarrow \mathbf{ae}$

i) $S \rightarrow \mathbf{b}A$, $A \rightarrow \mathbf{b}A \mid \mathbf{a}$, $D \rightarrow \mathbf{a}A \mid \mathbf{b}$

j) $S \rightarrow AC \mid \mathbf{b}C \mid S\mathbf{b} \mid \mathbf{a} \mid \mathbf{b}$, $A \rightarrow S\mathbf{b} \mid \mathbf{a}$, $C \rightarrow SC \mid AC \mid \mathbf{b}C \mid S\mathbf{b} \mid \mathbf{a} \mid \mathbf{b}$

7.3. a) $S \rightarrow AB \mid C_aC_c \mid \varepsilon$, $A \rightarrow C_aB \mid BD_1$, $B \rightarrow \mathbf{b}$, $C_a \rightarrow \mathbf{a}$, $C_c \rightarrow \mathbf{c}$, $D_1 \rightarrow D_2A$, $D_2 \rightarrow BB$

b) $S \rightarrow C_aA \mid \mathbf{a} \mid AC_b$, $A \rightarrow D_1C_b$, $B \rightarrow \mathbf{b} \mid AC_a$, $C_a \rightarrow \mathbf{a}$, $C_b \rightarrow \mathbf{b}$, $D_1 \rightarrow C_aB$

c) $S \rightarrow C_aA \mid BC_a \mid \mathbf{b}$, $A \rightarrow C_aC \mid D_1S$, $B \rightarrow D_2C_b \mid \mathbf{a}$, $C \rightarrow AD_3 \mid SC_b$, $C_a \rightarrow \mathbf{a}$, $C_b \rightarrow \mathbf{b}$, $D_1 \rightarrow C_bB$, $D_2 \rightarrow BC_a$, $D_3 \rightarrow CC_a$

d) $S \rightarrow \mathbf{a} \mid D_1B$, $A \rightarrow C_aS \mid C_bB \mid D_2A$, $B \rightarrow C_aS \mid \mathbf{b}$, $C \rightarrow C_bB \mid C_cB \mid CC_a$, $C_a \rightarrow \mathbf{a}$, $C_b \rightarrow \mathbf{b}$, $C_c \rightarrow \mathbf{c}$, $D_1 \rightarrow C_bA$, $D_2 \rightarrow C_cC$

e) $S \to D_1S \mid C_aB \mid \varepsilon$, $A \to AC_b \mid CC_a \mid \mathbf{b}$, $B \to C_aA \mid C_bB \mid \mathbf{a}$, $C \to CD_2 \mid C_aB$, $C_a \to \mathbf{a}$, $C_b \to \mathbf{b}$, $D_1 \to AC_b$, $D_2 \to C_aC_b$

f) $S \to C_bA \mid C_aB \mid \varepsilon$, $A \to C_aB \mid D_1S \mid \mathbf{b}$, $B \to C_aA \mid D_2S \mid \mathbf{a}$, $C \to D_3C \mid BC_a$, $C_a \to \mathbf{a}$, $C_b \to \mathbf{b}$, $D_1 \to C_bC$, $D_2 \to C_bA$, $D_3 \to SC_a$

7.4. a) $S \to D_1C_b$, $A \to C_eD_2$, $B \to \mathbf{f}$, $C \to C_aC_h$, $C_a \to \mathbf{a}$, $C_b \to \mathbf{b}$, $C_e \to \mathbf{e}$, $C_h \to \mathbf{h}$, $D_1 \to C_aA$, $D_2 \to C_bC$

b) $S \to C_aB \mid AC_a$, $A \to C_bB$, $B \to C_bB \mid \mathbf{b}$, $C_a \to \mathbf{a}$, $C_b \to \mathbf{b}$

c) $S \to D_1A \mid AA \mid \mathbf{a} \mid C_aA \mid BC_b \mid D_2S \mid \varepsilon$, $A \to BC_b \mid D_2S$, $B \to C_bC_a \mid C_aC_b$, $C_a \to \mathbf{a}$,$C_b \to \mathbf{b}$, $D_1 \to AA$, $D_2 \to C_aB$

7.5. No puede ser generada

		{C}		
	{C}	∅		
{S,A,C}	∅	{B}		
{S,A,C}	∅	{B}	∅	
{A,C}	{S,B}	{S,B}	{A,C}	{A,C}
b	**a**	**a**	**b**	**b**

7.6. No puede ser generada

		{A,B}		
	{S}	{B}		
{A,B}	∅	{A,B}		
{S}	{B}	{S}	{B}	
{A,B}	{S}	{A,B}	{S}	{A,B}
b	**a**	**b**	**a**	**b**

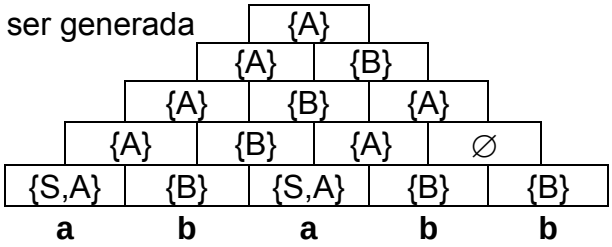
7.7. Si puede ser generada

		{S,B}		
	{S,B}	{A}		
{S,B}	{A}	{A}		
{S,B}	{A}	{A}	{S,B}	
{S,B}	{S,A}	{S,A}	{S,B}	{S,A}
0	**1**	**1**	**0**	**1**

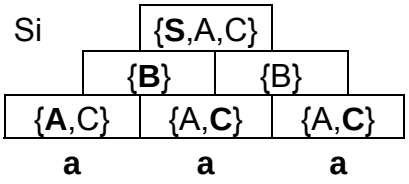
7.8. No puede ser generada

		{A}		
	{A}	{A}		
{A}	{A}	{A}		
{A}	{A}	{A}	{A}	
{S,A}	{S,A}	{S,A}	{S,A}	{S,A}
1	**1**	**1**	**1**	**1**

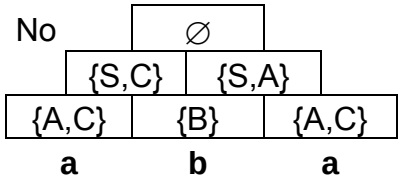
7.9. No puede ser generada



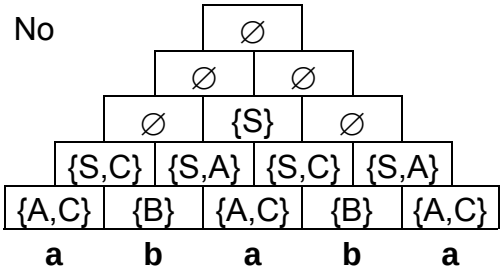
7.10. a) Si



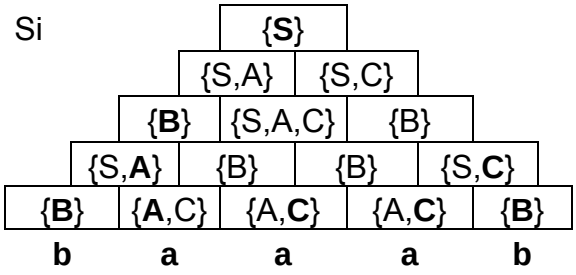
b) No



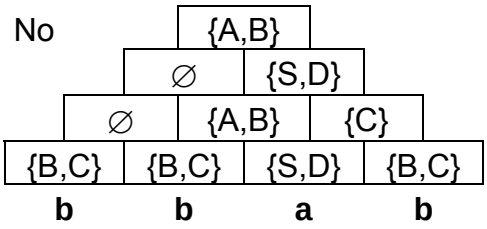
c) No



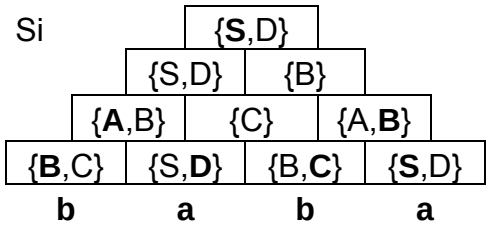
d) Si



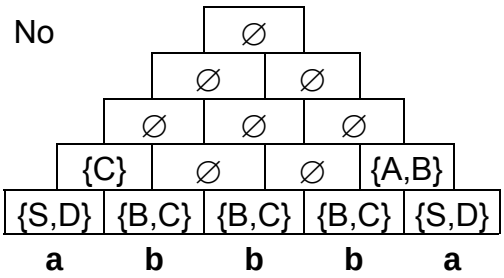
7.11. a) No



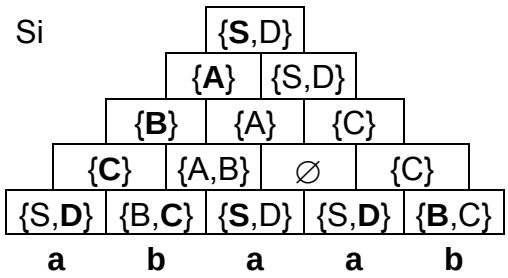
b) Si



c) No



d) Si



7.12. a) S → **a**S**c** | **b** | **a**S**c**S' | **b**S', S'→ **c** | **c**S'

b) S → **b**S | **a** | **b**SS' | **a**S', S'→ **a** | **ba** | **a**S' | **ba**S'

c) S → **c**A | **c**AS', S'→ **a** | **b** | **a**S' | **b**S', A → **a** | **a**A', A' → **a** | **a**A'

d) S → **aab** | **aab**S' | **a**A'**ab** | **a**A'**ab**S', S'→ **a** | **b** | **a**S' | **b**S', A → **a** | **a**A',
A' → **ba** | **aba** | **ab**S'**a** | **ba**A' | **aba**A' | **ab**S'**a**A'

e) S → **1**AS' | **0**S' | **1**A | **0**, S' → **1**AS'AS' | **0**S'AS' | **1**AAS' | **0**AS' | **1**AS'A |
0S'A | **1**AA | **0**A, A → **1**AS'S | **0**S'S | **1**AS | **0**S | **1**

f) S → **a**A**b** | **c**SB | **a**A**b**S' | **c**SBS', S'→ **a**A**b**S | **c**SBS | **a**A**b**S'S | **c**SBS'S |
aA**b**SS' | **c**SBSS' | **a**A**b**S'SS' | **c**SBS'SS',
A → **b**BA | **ba**A, B → **ac** | **a**A**b**A | **c**SBA | **a**A**b**S'A | **c**SBS'A

g) S → **bab**BA | **abb**BA | **a**BSBA | **bab**A'BA | **abb**A'BA | **a**BSA'BA | **b** | **a**S,
A → **bab** | **abb** | **a**BS | **bab**A' | **abb**A' | **a**BSA', B → **ba** | **ab**,
A' → **bab**BA|**abb**BA|**a**BSBA | **bab**A'BA|**abb**A'BA|**a**BSA'BA | **b** | **a**S | **bab**BAA'
| **abb**BAA' | **a**BSBAA' | **bab**A'BAA' | **abb**A'BAA' | **a**BSA'BAA' | **b**A' | **a**SA'

8.1. a) S → **a**S**c** | **bc**

b) S → **a**S**b** | **a**S | **a**

c) S → **a**S**b** | **aa**

d) S → **aa**S**c** | **b**

e) S → **a**S**c** | A, A → **b**A**c** | ε

f) S → **a**S**a** | A, A → **b**A**c** | ε

g) S → AB, A → **a**A**b** | ε, B → **c**B**d** | ε

h) S → AB, A → **a**A**b** | ε, B → **b**B**c** | ε

i) S → **a**S**b** | **aa**S**b** | ε

j) S → AB, A → **a**A**b** | ε, B → **b**B**cc** | ε

k) S → **a**S**c** | A | B**b** | C**c**, A → **a**A**b** | **a**A | **a**, B → **a**B**b** | B**b** | ε**,** C → C**c** | B

l) S → AC | BD, A → **a**A**b** | ε, B → **a**B | ε, C → **c**C | ε, D → **b**D**c** | ε

m)S → AC | B, A → **a**A**b** | A**b** | **b**, B → **a**B**c** | **a**B | **a**C, C → **b**C | ε

n) S → ABC, A → **a**A**b** | ε, B → **b**B | **b**, C → **b**C**c** | ε

o) S → **b**S**b** | **a**S**a** | **a** | **b** | ε

8.2. a) L(G) = (**a** ∪ **b**)***ab**(**ab** ∪ **b**)$^{+}$

b) L(G) = (**a** ∪ **ab**) ((**a** ∪ **ab**)^{2})*

c) L(G) = (**ab** ∪ **ba**)*(**ab** ∪ **aab**)

d) L(G) = ((**a** ∪ **b**)^{2})*. S → **a**A | **b**A | ε, A → **a**S | **b**S.

e) L(G) = **a***(**a** ∪ **b**)**b***(**a** ∪ **b**)*

f) L(G) = ((**b*****ab***)^{2})* ∪ **b***

8.3. a) Palíndromas de longitud par sobre Σ = { **a**, **b** }
b) Palíndromas de longitud impar sobre Σ = { **a**, **b** }
c) Cadenas No Palíndromas sobre Σ = { **a**, **b** }
d) Cadenas cuyos prefijos propios contienen más **a**s que **b**s.
e) L = (**a** $\cup$ **b**)***a**
f) L = (**a** $\cup$ **b***__a__ $\cup$ __ab***)$^+$

8.4. a) LL(1), b) LL(2), c) LL(1), d) LL(2), e) LL(2)

9.1. a) $\delta(q_0,\mathbf{a},\varepsilon) = (q_0,A)$, $\delta(q_0,\mathbf{b},A) = (q_1,A)$, $\delta(q_1,\mathbf{b},\varepsilon) = (q_1, \varepsilon)$, $\delta(q_1,\mathbf{c},A) = (q_2,\varepsilon)$, $\delta(q_2,\mathbf{c},A) = (q_2,\varepsilon)$, $F = \{q_2\}$

b) $\delta(q_0,\mathbf{a},\varepsilon) = (q_0,AA)$, $\delta(q_0,\mathbf{b},A) = (q_1,\varepsilon)$, $\delta(q_1,\mathbf{b},A) = (q_1,\varepsilon)$, $F = \{q_1\}$

c) $\delta(q_0,\mathbf{a},\varepsilon) = (q_0,A)$, $\delta(q_0,\mathbf{b},\varepsilon) = (q_1,\varepsilon)$, $\delta(q_1,\mathbf{c},A) = (q_1, \varepsilon)$, $F = \{q_1\}$

d) $\delta(q_0,\mathbf{a},\varepsilon) = (q_0,A)$, $\delta(q_0,\mathbf{b},A) = (q_1,AA)$, $\delta(q_1,\mathbf{b},\varepsilon) = (q_1,A)$, $\delta(q_1,\mathbf{c},A) = (q_2,\varepsilon)$, $\delta(q_2,\mathbf{c},A) = (q_2,\varepsilon)$, $F = \{q_2\}$

e) $\delta(q_0,\mathbf{a},\varepsilon) = (q_1,A)$, $\delta(q_1,\mathbf{a},\varepsilon) = (q_0,\varepsilon)$, $\delta(q_0,\mathbf{b},A) = (q_2,\varepsilon)$, $\delta(q_2,\mathbf{b},A) = (q_2,\varepsilon)$, $F = \{q_2\}$

9.2. $\delta(q_0,\mathbf{a},Z) = (q_0,A)$, $\delta(q_0,\mathbf{c},A) = (q_0,A)$, $\delta(q_0,\mathbf{b},A) = (q_0,B)$, $\delta(q_0,\mathbf{a},B) = (q_1,\varepsilon)$, $F= \{q_1\}$

9.3. a) $\delta(q_0,\mathbf{a},Z) = (q_0,AZ)$, $\delta(q_0,\mathbf{a},A) = (q_0,AA)$, $\delta(q_0,\mathbf{b},A) = (q_1,A)$, $\delta(q_1,\mathbf{b},A) = (q_1,A)$, $\delta(q_1,\mathbf{a},A) = (q_2,\varepsilon)$, $\delta(q_2,\mathbf{a},A) = (q_2,\varepsilon)$, $\delta(q_2,\varepsilon,Z) = (q_3,\varepsilon)$, $F = \{q_3\}$

b) $\delta(q_0,\mathbf{a},Z) = (q_0,AZ)$, $\delta(q_0,\mathbf{a},A) = (q_0,AA)$, $\delta(q_0,\mathbf{b},A) = (q_1,\varepsilon)$, $\delta(q_1,\mathbf{b},A) = (q_1,\varepsilon)$, $\delta(q_1,\mathbf{b},Z) = (q_2,AZ)$, $\delta(q_2,\mathbf{b},A) = (q_2,AA)$, $\delta(q_2,\mathbf{a},A) = (q_3,\varepsilon)$, $\delta(q_3,\mathbf{a},A) = (q_3,\varepsilon)$, $\delta(q_3,\varepsilon,Z) = (q_4,\varepsilon)$, $F = \{q_4\}$

c) $\delta(q_0,\mathbf{a},Z) = (q_0,AZ)$, $\delta(q_0,\mathbf{a},A) = (q_0,AA)$, $\delta(q_0,\mathbf{b},A)= (q_1,BA)$, $\delta(q_1,\mathbf{b},B)= (q_1,BB)$, $\delta(q_1,\mathbf{a},B) = (q_2,\varepsilon)$, $\delta(q_2,\mathbf{a},B) = (q_2,\varepsilon)$, $\delta(q_2,\mathbf{b},A) = (q_3,\varepsilon)$, $\delta(q_3,\mathbf{b},A) = (q_3,\varepsilon)$, $\delta(q_3,\varepsilon,Z) = (q_4,\varepsilon)$, $F = \{q_4\}$

d) $\delta(q_0,\mathbf{a},Z) = (q_0,AZ)$, $\delta(q_0,\mathbf{a},A) = (q_0,AA)$, $\delta(q_0,\mathbf{b},A) = (q_1,\varepsilon)$, $\delta(q_1,\mathbf{b},A) = (q_1,\varepsilon)$, $\delta(q_0,\mathbf{b},Z) = (q_2,\varepsilon)$, $\delta(q_1,\mathbf{b},Z) = (q_2,\varepsilon)$, $F = \{q_2\}$

9.4. a) $\Delta(q_0,\mathbf{a},\varepsilon) = (q_0,A)$, $\Delta(q_0,\varepsilon,\varepsilon) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{b},A) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{b},\varepsilon) = (q_2,\varepsilon)$, $\Delta(q_2,\mathbf{b},\varepsilon) = (q_2,\varepsilon)$, $\Delta(q_1,\varepsilon,A) = (q_3,\varepsilon)$, $\Delta(q_3,\varepsilon,A) = (q_3,\varepsilon)$, $F = \{q_2,q_3\}$

b) $\Delta(q_0,\mathbf{a},\varepsilon) = (q_0,A)$, $\Delta(q_0,\varepsilon,A) = (q_0,\varepsilon)$, $\Delta(q_0,\mathbf{b},A) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{b},A) = (q_1,\varepsilon)$, $\Delta(q_1,\varepsilon,A) = (q_1,\varepsilon)$, $F = \{q_0,q_1\}$

c) $\Delta(q_0,\mathbf{a},\varepsilon) = (q_0,A)$, $\Delta(q_0,\mathbf{b},A) = (q_1,\varepsilon)$, $\Delta(q_0,\mathbf{b},\varepsilon) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{b},A) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{b},\varepsilon) = (q_1,\varepsilon)$, $F = \{q_0,q_1\}$

d) $\Delta(q_0,\mathbf{a},\varepsilon) = (q_0,A)$, $\Delta(q_0,\varepsilon,\varepsilon) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{b},A) = \{ (q_1,\varepsilon), (q_2,\varepsilon) \}$,
$\Delta(q_2,\mathbf{b},\varepsilon) = \{ (q_1,\varepsilon), (q_3,\varepsilon) \}$, $\Delta(q_3,\mathbf{b},\varepsilon) = (q_1,\varepsilon)$, $F = \{q_1\}$

e) $\Delta(q_0,\mathbf{a},\varepsilon) = \{ (q_0,A), (q_1,\varepsilon) \}$, $\Delta(q_0,\mathbf{b},\varepsilon) = \{(q_0,B),(q_1,\varepsilon)\}$, $\Delta(q_1,\mathbf{b},B) = (q_1,\varepsilon)$,
$\Delta(q_1,\mathbf{a},A) = (q_1,\varepsilon)$, $F = \{q_1\}$

f) $\Delta(q_0,\mathbf{a},\varepsilon) = (q_0,A)$, $\Delta(q_0,\mathbf{b},A) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{a},A) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{b},A) = (q_1,\varepsilon)$,
$\Delta(q_0,\varepsilon,A) = (q_1,\varepsilon)$, $\Delta(q_1,\varepsilon,A) = (q_1,\varepsilon)$, $F = \{q_0,q_1\}$

g) $\Delta(q_0,\mathbf{a},\varepsilon) = \{ (q_0,A), (q_1,\varepsilon) \}$, $\Delta(q_0,\mathbf{b},\varepsilon) = (q_0,A)$, $\Delta(q_1,\mathbf{a},A) = (q_1,\varepsilon)$,
$\Delta(q_1,\mathbf{b},A) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{b},\varepsilon) = (q_2,\varepsilon)$, $F = \{q_2\}$

h) $\Delta(q_0,\mathbf{a},\varepsilon) = \{ (q_0,A), (q_1, \varepsilon) \}$, $\Delta(q_0,\mathbf{b},\varepsilon) = \{ (q_0,B), (q_1,\varepsilon) \}$, $\Delta(q_0,\varepsilon,\varepsilon) = (q_1,\varepsilon)$,
$\Delta(q_1,\mathbf{a},A) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{b},B) = (q_1,\varepsilon)$, $\Delta(q_1,\mathbf{a},B) = (q_2,\varepsilon)$, $\Delta(q_1,\mathbf{b},A) = (q_2,\varepsilon)$,
$\Delta(q_2,\mathbf{a},A) = (q_2,\varepsilon)$, $\Delta(q_2,\mathbf{b},B) = (q_2,\varepsilon)$, $\Delta(q_2,\mathbf{a},B) = (q_2,\varepsilon)$, $\Delta(q_2,\mathbf{b},A) = (q_2,\varepsilon)$

9.5. a) $\Delta(q_0,\mathbf{a},Z)=(q_0,AZ)$, $\Delta(q_0,\mathbf{b},Z)=(q_0,BZ)$, $\Delta(q_0,\mathbf{a},A)=(q_0,AA)$,$\Delta(q_0,\mathbf{b},B)=(q_0,BB)$,
$\Delta(q_0,\mathbf{a},B)= (q_0,C)$, $\Delta(q_0,\mathbf{a},C)= (q_0,\varepsilon)$, $\Delta(q_0,\mathbf{b},C) = (q_0,CB)$, $\Delta(q_0,\mathbf{b},A) = (q_1,\varepsilon)$,
$\Delta(q_1,\varepsilon,A) = (q_0,\varepsilon)$, $\Delta(q_0,\varepsilon,Z) = (q_2,\varepsilon)$, $F = \{q_2\}$

b) $\Delta(q_0,\mathbf{a},Z)=(q_0,AZ)$, $\Delta(q_0,\mathbf{b},Z)=(q_0,BZ)$, $\Delta(q_0,\mathbf{a},A)=(q_0,AA)$, $\Delta(q_0,\mathbf{b},B)=(q_0,BB)$,
$\Delta(q_0,\mathbf{a},B) = \{ (q_0,\varepsilon), (q_0,C) \}$, $\Delta(q_0,\mathbf{a},C) = (q_0,\varepsilon)$, $\Delta(q_0,\mathbf{b},C) = (q_0,CB)$,
$\Delta(q_0,\mathbf{b},A) = \{ (q_0,\varepsilon), (q_1,\varepsilon) \}$, $\Delta(q_1,\varepsilon,A) = (q_0,\varepsilon)$, $\Delta(q_0,\varepsilon,Z) = (q_2,\varepsilon)$, $F = \{q_2\}$

9.6. $L = \mathbf{ab^+a} \cup \mathbf{a}$

9.7. Las cadenas de L están formadas por una subcadena que tiene la misma cantidad de **a**s que **b**s, pero que cada prefijo propio de esa subcadena siempre tendrá una cantidad mayor o igual de **a**s que **b**s, adicionalmene, todas las cadenas de L terminan con el sufijo **bb**.

9.8. w_1: $(q_0, \underline{\mathbf{a}}\mathbf{bba}, \underline{Z}) \vdash (q_0, \underline{\mathbf{b}}\mathbf{ba}, \underline{A}Z) \vdash (q_0, \underline{\mathbf{b}}\mathbf{a}, \underline{Z}) \vdash (q_1, \underline{\mathbf{a}}, \underline{A}Z)$
w_2: $(q_0, \underline{\mathbf{a}}\mathbf{bab}, \underline{Z}) \vdash (q_0, \underline{\mathbf{b}}\mathbf{ab}, \underline{A}Z) \vdash (q_0, \underline{\mathbf{a}}\mathbf{b}, \underline{Z}) \vdash (q_0, \underline{\mathbf{b}}, \underline{A}Z) \vdash (q_0, \underline{\varepsilon}, \underline{Z})$
w_3: $(q_0, \underline{\mathbf{a}}\mathbf{bbb}, \underline{Z}) \vdash (q_0, \underline{\mathbf{b}}\mathbf{bb}, \underline{A}Z) \vdash (q_0, \underline{\mathbf{b}}\mathbf{b}, \underline{Z}) \vdash (q_1,\underline{\mathbf{b}},\underline{A}Z) \vdash (q_1,\underline{\mathbf{b}},\underline{Z}) \vdash (q_2,\underline{\varepsilon},\underline{\varepsilon})$

9.9. a) $\Delta(q_0, \varepsilon, \varepsilon) = \{ (q_1, S) \}$, $\Delta(q_1, \varepsilon, S) = \{ (q_1, \mathbf{a}AA) \}$,
$\Delta(q_1, \varepsilon, A) = \{ (q_1, \mathbf{a}A), (q_1, \mathbf{b}S), (q_1, \mathbf{c}) \}$,
$\Delta(q_1, \mathbf{a}, \mathbf{a}) = \Delta(q_1, \mathbf{b}, \mathbf{b}) = \Delta(q_1, \mathbf{c}, \mathbf{c}) = \{ (q_1, \varepsilon) \}$

b) $\Delta(q_0, \varepsilon, \varepsilon) = \{ (q_1, S) \}$, $\Delta(q_1, \varepsilon, S) = \{ (q_1, \mathbf{ab}S), (q_1, \mathbf{ac}S\mathbf{bb}), (q_1, \mathbf{c}) \}$,
$\Delta(q_1, \mathbf{a}, \mathbf{a}) = \Delta(q_1, \mathbf{b}, \mathbf{b}) = \Delta(q_1, \mathbf{c}, \mathbf{c}) = \{ (q_1, \varepsilon) \}$

c) $\Delta(q_0, \varepsilon, \varepsilon) = \{ (q_1, S) \}$, $\Delta(q_1, \varepsilon, S) = \{ (q_1, \mathbf{a}ABB), (q_1, \mathbf{b}AA) \}$
$\Delta(q_1, \varepsilon, A) = \{ (q_1, \mathbf{a}BB), (q_1, \mathbf{b}A) \}$, $\Delta(q_1, \varepsilon, B) = \{ (q_1, \mathbf{b}BB), (q_1, \mathbf{a}) \}$,
$\Delta(q_1, \mathbf{a}, \mathbf{a}) = \Delta(q_1, \mathbf{b}, \mathbf{b}) = \{ (q_1, \varepsilon) \}$

9.10. a) $\delta(q_0, \varepsilon, \varepsilon) = \{ (q_1, S) \}$, $\delta(q_1, \mathbf{a}, \varepsilon) = (q_a, \varepsilon)$, $\delta(q_a, \varepsilon, \mathbf{a}) = (q_1, \varepsilon)$,
$\delta(q_1, \mathbf{b}, \varepsilon) = (q_b, \varepsilon)$, $\delta(q_b, \varepsilon, \mathbf{b}) = (q_1, \varepsilon)$, $\delta(q_1, \mathbf{c}, \varepsilon) = (q_c, \varepsilon)$,
$\delta(q_c, \varepsilon, \mathbf{c}) = (q_1, \varepsilon)$, $\delta(q_a, \varepsilon, S) = \{ (q_a, \mathbf{a}AA) \}$, $\delta(q_a, \varepsilon, A) = \{ (q_a, \mathbf{a}A) \}$,
$\delta(q_b, \varepsilon, A) = \{ (q_b, \mathbf{b}S) \}$, $\delta(q_c, \varepsilon, A) = \{ (q_c, \mathbf{c}) \}$, $\delta(q_1, \mathbf{\$}, \varepsilon) = (q_2, \varepsilon)$

b) $\delta(q_0, \varepsilon, \varepsilon) = \{ (q_1, S) \}$, $\delta(q_1, \mathbf{a}, \varepsilon) = (q_a, \varepsilon)$, $\delta(q_a, \varepsilon, \mathbf{a}) = (q_1, \varepsilon)$,
$\delta(q_1, \mathbf{b}, \varepsilon) = (q_b, \varepsilon)$, $\delta(q_b, \varepsilon, \mathbf{b}) = (q_1, \varepsilon)$, $\delta(q_1, \mathbf{c}, \varepsilon) = (q_c, \varepsilon)$,
$\delta(q_c, \varepsilon, \mathbf{c}) = (q_1, \varepsilon)$, $\delta(q_a, \varepsilon, S) = \{ (q_a, \mathbf{a}A) \}$, $\delta(q_b, \varepsilon, A) = \{ (q_b, \mathbf{b}S) \}$,
$\delta(q_c, \varepsilon, A) = (q_c, \mathbf{c}S\mathbf{bb}) \}$, $\delta(q_c, \varepsilon, S) = (q_c, \mathbf{c}) \}$, $\delta(q_1, \mathbf{\$}, \varepsilon) = (q_2, \varepsilon)$

c) $\delta(q_0, \varepsilon, \varepsilon) = \{ (q_1, S) \}$, $\delta(q_1, \mathbf{a}, \varepsilon) = (q_a, \varepsilon)$, $\delta(q_a, \varepsilon, \mathbf{a}) = (q_1, \varepsilon)$,
$\delta(q_1, \mathbf{b}, \varepsilon) = (q_b, \varepsilon)$, $\delta(q_b, \varepsilon, \mathbf{b}) = (q_1, \varepsilon)$, $\delta(q_a, \varepsilon, S) = \{ (q_a, \mathbf{a}ABB) \}$,
$\delta(q_a, \varepsilon, A) = \{ (q_a, \mathbf{a}BB) \}$, $\delta(q_a, \varepsilon, B) = \{ (q_a, \mathbf{a}) \}$, $\delta(q_b, \varepsilon, S) = \{ (q_b, \mathbf{b}AA) \}$,
$\delta(q_b, \varepsilon, A) = \{ (q_b, \mathbf{b}A) \}$, $\delta(q_b, \varepsilon, B) = \{ (q_b, \mathbf{b}BB) \}$, $\delta(q_1, \mathbf{\$}, \varepsilon) = (q_2, \varepsilon)$

9.11. a) $S \rightarrow \mathbf{a}A \mid \mathbf{a}$, $A \rightarrow \mathbf{b}B$, $B \rightarrow \mathbf{b}B \mid \mathbf{a}$
b) $S \rightarrow \mathbf{a}AS \mid \mathbf{bb}$, $A \rightarrow \mathbf{a}AA \mid \mathbf{b}$

9.12. $S \rightarrow \mathbf{a}A$, $A \rightarrow \mathbf{b} \mid \mathbf{a}A$

9.13. $S \rightarrow \mathbf{a}AB \mid \varepsilon$, $A \rightarrow \mathbf{b}AC \mid \mathbf{a}C$, $B \rightarrow \mathbf{a}S$, $C \rightarrow \mathbf{b}$

9.14. $S \rightarrow \mathbf{a}AB \mid \varepsilon$, $A \rightarrow \mathbf{a}AC \mid \mathbf{b}$, $B \rightarrow \mathbf{b}S$, $C \rightarrow \mathbf{a}$

9.15. $S \rightarrow \mathbf{a}AS \mid \mathbf{b}BC$, $A \rightarrow \mathbf{a}AA \mid \mathbf{b}$, $B \rightarrow \mathbf{b}$, $C \rightarrow \varepsilon$

9.16. $S \rightarrow \mathbf{a}A \mid \mathbf{b}B$, $A \rightarrow \mathbf{a}A \mid \mathbf{b}A \mid \mathbf{a}$, $B \rightarrow \mathbf{a}B \mid \mathbf{b}B \mid \mathbf{b}$

9.17. $S \rightarrow \mathbf{a}AS \mid \mathbf{b}B$, $A \rightarrow \mathbf{a}AA \mid \mathbf{b}$, $B \rightarrow \mathbf{b}B \mid \varepsilon$

9.18. $S \rightarrow \mathbf{0}AS \mid \mathbf{1}B$, $A \rightarrow \mathbf{0}A \mid \mathbf{1}$, $B \rightarrow \mathbf{0}B \mid \varepsilon$

9.19. $S \rightarrow \mathbf{a}AC \mid \varepsilon$, $A \rightarrow \mathbf{b}AB \mid \mathbf{a}$, $B \rightarrow \mathbf{a}$, $C \rightarrow \mathbf{b}S$

9.20. $S \rightarrow \mathbf{c} \mid \mathbf{a}A \mid \mathbf{b}A$, $A \rightarrow \mathbf{a}A\mathbf{a} \mid \mathbf{a}A\mathbf{b} \mid \mathbf{b}A\mathbf{b} \mid \mathbf{b}A\mathbf{a} \mid \mathbf{ca} \mid \mathbf{cb}$

10.1. a) $\delta(q_0,\sigma) = (q_0,\sigma,R)$, $\delta(q_0,\#) = (q_1,\#,L)$, $\delta(q_1,\mathbf{a}) = (q_2,\#,L)$, $\delta(q_1,\mathbf{b}) = (q_3,\#,L)$,
$\delta(q_1,\mathbf{c}) = (q_4,\#,L)$, $\delta(q_2,\sigma) = (q_2,\sigma,L)$, $\delta(q_2,\#) = (q_0,\mathbf{a},R)$, $\delta(q_3,\sigma) = (q_3,\sigma,L)$,
$\delta(q_3,\#) = (q_0,\mathbf{b},R)$, $\delta(q_4,\sigma) = (q_4,\sigma,L)$, $\delta(q_4,\#) = (q_5,\#,R)$

b) $\delta(q_0,\mathbf{a}) = (q_1,\mathbf{c},R)$, $\delta(q_0,\mathbf{b}) = (q_2,\mathbf{d},R)$, $\delta(q_1,\sigma) = (q_1,\sigma,R)$, $\delta(q_2,\sigma) = (q_2,\sigma,R)$, $\delta(q_1,\#) = (q_3,\mathbf{A},L)$, $\delta(q_2,\#) = (q_3,\mathbf{B},L)$, $\delta(q_3,\mathbf{a}) = (q_3,\mathbf{a},L)$, $\delta(q_3,\mathbf{b}) = (q_3,\mathbf{b},L)$, $\delta(q_3,\mathbf{A}) = (q_3,\mathbf{A},L)$, $\delta(q_3,\mathbf{B}) = (q_3,\mathbf{B},L)$, $\delta(q_3,\mathbf{c}) = (q_0,\mathbf{a},R)$, $\delta(q_3,\mathbf{d}) = (q_0,\mathbf{b},R)$, $\delta(q_0,\mathbf{A}) = (q_4,\mathbf{a},R)$, $\delta(q_0,\mathbf{B}) = (q_4,\mathbf{b},R)$, $\delta(q_4,\mathbf{A}) = (q_4,\mathbf{a},R)$, $\delta(q_4,\mathbf{B}) = (q_4,\mathbf{b},R)$, $\delta(q_4,\#) = (q_5,\#,L)$, $\delta(q_5,\sigma) = (q_5,\sigma,L)$, $\delta(q_5,\#) = (q_6,\#,R)$, $\delta(q_0,\#) = (q_6,\#,R)$

c) $\delta(q_0,\mathbf{1}) = (q_1,\mathbf{0},R)$, $\delta(q_0,\#) = (q_{11},\#,L)$, $\delta(q_1,\mathbf{1}) = (q_1,\mathbf{1},R)$, $\delta(q_1,\#) = (q_1,\#,R)$, $\delta(q_1,\#) = (q_2,\#,L)$, $\delta(q_1,\mathbf{0}) = (q_2,\mathbf{0},L)$, $\delta(q_2,\mathbf{1}) = (q_3,\mathbf{0},L)$, $\delta(q_3,\mathbf{1}) = (q_3,\mathbf{1},L)$, $\delta(q_3,\#) = (q_3,\#,L)$, $\delta(q_3,\mathbf{0}) = (q_0,\#,R)$, $\delta(q_0,\#) = (q_4,\#,R)$, $\delta(q_4,\sigma) = (q_4,\sigma,R)$, $\delta(q_4,\#) = (q_5,\#,L)$, $\delta(q_5,\mathbf{0}) = (q_5,\#,L)$, $\delta(q_5,\mathbf{1}) = (q_5,\#,L)$, $\delta(q_5,\#) = (q_6,\#,L)$, $\delta(q_6,\mathbf{0}) = (q_6,\mathbf{1},L)$, $\delta(q_6,\#) = (q_{11},\#,R)$, $\delta(q_2,\#) = (q_7,\#,L)$, $\delta(q_7,\sigma) = (q_7,\sigma,L)$, $\delta(q_7,\#) = (q_8,\#,R)$, $\delta(q_8,\mathbf{0}) = (q_8,\#,R)$, $\delta(q_8,\mathbf{1}) = (q_8,\#,R)$, $\delta(q_8,\#) = (q_9,\#,R)$, $\delta(q_9,\mathbf{0}) = (q_9,\mathbf{1},R)$, $\delta(q_9,\#) = (q_{10},\#,L)$, $\delta(q_{10},\mathbf{1}) = (q_{10},\mathbf{1},L)$, $\delta(q_{10},\#) = (q_{11},\#,R)$

d) $\delta(q_0,\mathbf{a}) = (q_0,\mathbf{a},R)$, $\delta(q_0,\#) = (q_{10},\#,L)$, $\delta(q_0,\mathbf{b}) = (q_1,\mathbf{b},R)$, $\delta(q_1,\mathbf{b}) = (q_1,\mathbf{b},R)$, $\delta(q_1,\#) = (q_{10},\#,L)$, $\delta(q_0,\mathbf{c}) = (q_2,\mathbf{c},R)$, $\delta(q_1,\mathbf{c}) = (q_2,\mathbf{c},R)$, $\delta(q_2,\mathbf{c}) = (q_2,\mathbf{c},R)$, $\delta(q_2,\#) = (q_{10},\#,L)$, $\delta(q_1,\mathbf{a}) = (q_3,\mathbf{b},L)$, $\delta(q_3,\mathbf{b}) = (q_3,\mathbf{b},L)$, $\delta(q_3,\mathbf{a}) = (q_4,\mathbf{a},R)$, $\delta(q_3,\#) = (q_4,\#,R)$, $\delta(q_4,\mathbf{b}) = (q_1,\mathbf{a},R)$, $\delta(q_2,\mathbf{a}) = (q_5,\mathbf{c},L)$, $\delta(q_5,\mathbf{c}) = (q_5,\mathbf{c},L)$, $\delta(q_5,\mathbf{b}) = (q_6,\mathbf{b},R)$, $\delta(q_6,\mathbf{c}) = (q_3,\mathbf{b},L)$, $\delta(q_5,\mathbf{a}) = (q_7,\mathbf{a},R)$, $\delta(q_5,\#) = (q_7,\#,R)$, $\delta(q_7,\mathbf{c}) = (q_2,\mathbf{a},R)$, $\delta(q_2,\mathbf{b}) = (q_8,\mathbf{c},L)$, $\delta(q_8,\mathbf{c}) = (q_8,\mathbf{c},L)$, $\delta(q_8,\mathbf{b}) = (q_9,\mathbf{b},R)$, $\delta(q_8,\mathbf{a}) = (q_9,\mathbf{a},R)$, $\delta(q_8,\#) = (q_9,\#,R)$, $\delta(q_9,\mathbf{c}) = (q_2,\mathbf{b},R)$, $\delta(q_{10},\sigma) = (q_{10},\sigma,L)$, $\delta(q_{10},\#) = (q_{11},\#,R)$

e) $\delta(q_0,\mathbf{b}) = (q_0,\#,R)$, $\delta(q_0,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_1,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_1,\mathbf{b}) = (q_2,\#,R)$, $\delta(q_2,\sigma) = (q_2,\sigma,R)$, $\delta(q_2,\#) = (q_3,\mathbf{1},L)$, $\delta(q_3,\sigma) = (q_3,\sigma,L)$, $\delta(q_3,\#) = (q_0,\#,R)$, $\delta(q_0,\mathbf{1}) = (q_4,\mathbf{1},L)$, $\delta(q_0,\#) = (q_5,\#,R)$, $\delta(q_4,\#) = (q_5,\#,R)$

f) $\delta(q_0,\mathbf{1}) = (q_1,\#,R)$, $\delta(q_1,\sigma) = (q_1,\sigma,R)$, $\delta(q_1,\#) = (q_2,\mathbf{0},R)$, $\delta(q_2,\#) = (q_3,\mathbf{1},L)$, $\delta(q_3,\sigma) = (q_3,\sigma,L)$, $\delta(q_3,\#) = (q_0,\#,R)$, $\delta(q_0,\mathbf{0}) = (q_4,\mathbf{0},L)$, $\delta(q_0,\#) = (q_5,\#,R)$, $\delta(q_4,\#) = (q_5,\#,R)$

g) $\delta(q_0,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_1,\mathbf{b}) = (q_2,\#,R)$, $\delta(q_2,\sigma) = (q_2,\sigma,R)$, $\delta(q_2,\#) = (q_3,\mathbf{1},L)$, $\delta(q_3,\sigma) = (q_3,\sigma,L)$, $\delta(q_3,\#) = (q_0,\#,R)$, $\delta(q_0,\mathbf{b}) = (q_4,\#,R)$, $\delta(q_1,\mathbf{a}) = (q_4,\#,R)$, $\delta(q_4,\sigma) = (q_4,\#,R)$, $\delta(q_4,\#) = (q_5,\mathbf{0},L)$, $\delta(q_0,\mathbf{1}) = (q_5,\mathbf{1},L)$, $\delta(q_5,\#) = (q_6,\#,R)$

10.2. $\delta(q_0,\mathbf{0}) = (q_2,\mathbf{1},R)$, $\delta(q_2,\#) = (q_0,\#,L)$, $\delta(q_0,\mathbf{1}) = (q_1,\mathbf{0},L)$, $\delta(q_1,\mathbf{1}) = (q_1,\mathbf{0},L)$, $\delta(q_1,\mathbf{0}) = (q_2,\mathbf{1},R)$, $\delta(q_1,\#) = (q_2,\mathbf{1},R)$, $\delta(q_2,\mathbf{0}) = (q_2,\mathbf{0},R)$

10.3. $\delta(q_0,\#) = (q_1,\mathbf{a},R)$, $\delta(q_1,\#) = (q_2,\#,L)$, $\delta(q_2,\mathbf{a}) = (q_2,\mathbf{a},L)$, $\delta(q_2,\#) = (q_3,\#,R)$, $\delta(q_3,\mathbf{a}) = (q_4,\mathbf{0},R)$, $\delta(q_4,\mathbf{a}) = (q_4,\mathbf{a},R)$, $\delta(q_4,\#) = (q_5,\#,R)$, $\delta(q_5,\mathbf{a}) = (q_5,\mathbf{a},R)$,

$\delta(q_5,\#) = (q_6,\mathbf{a},L)$, $\delta(q_6,\mathbf{a}) = (q_6,\mathbf{a},L)$, $\delta(q_6,\#) = (q_6,\#,L)$, $\delta(q_6,\mathbf{0}) = (q_3,\mathbf{a},R)$,
$\delta(q_3,\#) = (q_0,\#,R)$, $\delta(q_0,\mathbf{a}) = (q_0,\mathbf{a},R)$

10.4. $\delta(q_0,\mathbf{1}) = (q_1,\mathbf{0},R)$, $\delta(q_1,\mathbf{1}) = (q_1,\mathbf{1},R)$, $\delta(q_1,\times) = (q_2,\times,R)$, $\delta(q_2,\mathbf{1}) = (q_3,\mathbf{0},R)$,
$\delta(q_3,\mathbf{1}) = (q_3,\mathbf{1},R)$, $\delta(q_3,=) = (q_3,=,R)$, $\delta(q_3,\#) = (q_4,\mathbf{1},L)$, $\delta(q_4,\mathbf{1}) = (q_4,\mathbf{1},L)$,
$\delta(q_4,=) = (q_4,=,L)$, $\delta(q_4,\mathbf{0}) = (q_2,\mathbf{0},R)$, $\delta(q_2,=) = (q_5,=,L)$, $\delta(q_5,\mathbf{0}) = (q_5,\mathbf{1},L)$,
$\delta(q_5,\times) = (q_6,\times,L)$, $\delta(q_6,\mathbf{1}) = (q_6,\mathbf{1},L)$, $\delta(q_6,\mathbf{0}) = (q_0,\mathbf{1},R)$, $\delta(q_0,\times) = (q_7,\mathbf{1},L)$,
$\delta(q_7,\mathbf{1}) = (q_7,\mathbf{1},L)$, $\delta(q_7,\#) = (q_8,\#,R)$, $\delta(q_8,\mathbf{1}) = (q_8,\#,R)$, $\delta(q_8,=) = (q_9,\#,R)$

10.5. $\delta(q_0,\mathbf{1}) = (q_0,\mathbf{1},R)$, $\delta(q_0,\#) = (q_1,\#,L)$, $\delta(q_1,\mathbf{1}) = (q_2,\mathbf{x},L)$, $\delta(q_1,\#) = (q_{10},\#,R)$,
$\delta(q_2,\mathbf{1}) = (q_3,\mathbf{x},R)$, $\delta(q_3,\mathbf{x}) = (q_3,\mathbf{x},R)$, $\delta(q_3,\#) = (q_2,\mathbf{x},L)$, $\delta(q_2,\mathbf{x}) = (q_2,\mathbf{x},L)$,
$\delta(q_2,\#) = (q_4,\#,R)$, $\delta(q_4,\mathbf{x}) = (q_4,\mathbf{x},R)$, $\delta(q_4,\#) = (q_5,\#,L)$, $\delta(q_5,\mathbf{x}) = (q_6,\mathbf{1},L)$,
$\delta(q_6,\mathbf{x}) = (q_7,\mathbf{1},L)$, $\delta(q_7,\mathbf{x}) = (q_8,\mathbf{0},R)$, $\delta(q_8,\mathbf{0}) = (q_8,\mathbf{0},R)$, $\delta(q_8,\mathbf{1}) = (q_8,\mathbf{1},R)$,
$\delta(q_8,\#) = (q_7,\mathbf{1},L)$, $\delta(q_7,\mathbf{0}) = (q_7,\mathbf{0},L)$, $\delta(q_7,\mathbf{1}) = (q_7,\mathbf{1},L)$, $\delta(q_7,\#) = (q_9,\#,R)$,
$\delta(q_9,\mathbf{0}) = (q_9,\mathbf{x},R)$, $\delta(q_9,\mathbf{1}) = (q_5,\mathbf{1},L)$, $\delta(q_6,\#) = (q_{10},\mathbf{1},S)$

10.6. $\delta(q_0,\mathbf{1}) = (q_1,\mathbf{0},R)$, $\delta(q_0,=) = (q_3,=,R)$, $\delta(q_1,\mathbf{1}) = (q_1,\mathbf{1},R)$, $\delta(q_1,=) = (q_1,=,R)$,
$\delta(q_1,\#) = (q_2,\mathbf{1},L)$, $\delta(q_2,\mathbf{1}) = (q_2,\mathbf{1},L)$, $\delta(q_2,=) = (q_2,=,L)$, $\delta(q_2,\ \mathbf{0}) = (q_0,\mathbf{0},R)$,
$\delta(q_3,\mathbf{1}) = (q_3,\mathbf{1},R)$, $\delta(q_3,\ \#) = (q_4,*,L)$, $\delta(q_4,\mathbf{1}) = (q_4,\mathbf{1},L)$, $\delta(q_4,=) = (q_4,=,L)$,
$\delta(q_4,\mathbf{0}) = (q_4,\mathbf{1},L)$, $\delta(q_4,\#) = (q_5,\#,R)$, $\delta(q_5,\ \mathbf{1}) = (q_6,\#,R)$, $\delta(q_6,\mathbf{1}) = (q_7,\#,R)$,
$\delta(q_6,=) = (q_{17},\#,R)$, $\delta(q_7,\mathbf{1}) = (q_8,\mathbf{0},R)$, $\delta(q_7,\ =) = (q_{14},\ =,R)$, $\delta(q_8,\mathbf{1}) = (q_8,\mathbf{1},R)$,
$\delta(q_8,=) = (q_9,=,R)$, $\delta(q_9,\mathbf{1}) = (q_{10},\mathbf{0},R)$, $\delta(q_9,*) = (q_{12},*,L)$, $\delta(q_{10},\mathbf{1})= (q_{10},\mathbf{1},R)$,
$\delta(q_{10},*)= (q_{10},*,R)$, $\delta(q_{10},\#) = (q_{11},\mathbf{1},L)$, $\delta(q_{11},\mathbf{1}) = (q_{11},\mathbf{1},L)$, $\delta(q_{11},*) = (q_{11},*,L)$,
$\delta(q_{11},\mathbf{0})= (q_9,\mathbf{0},R)$, $\delta(q_{12},\mathbf{0}) = (q_{12},\mathbf{1},L)$, $\delta(q_{12},=) = (q_{13},=,L)$, $\delta(q_{13},\mathbf{1}) = (q_{13},\mathbf{1},L)$,
$\delta(q_{13},\mathbf{0})= (q_7,\mathbf{0},R)$, $\delta(q_{14},\mathbf{1})= (q_{14},\mathbf{1},R)$, $\delta(q_{14},*) = (q_{14},\mathbf{1},R)$, $\delta(q_{14},\#) = (q_{15},\#,L)$,
$\delta(q_{15},\mathbf{1}) = (q_{16},*,L)$, $\delta(q_{16},\mathbf{1})= (q_{16},\mathbf{1},L)$, $\delta(q_{16},=)= (q_{16},=,L)$, $\delta(q_{16},\mathbf{0})= (q_{16},\mathbf{1},L)$,
$\delta(q_{16},\#) = (q_6,\#,R)$, $\delta(q_{17},\mathbf{1})= (q_{17},\mathbf{1},R)$, $\delta(q_{17},*) = (q_{18},\#,L)$, $\delta(q_{18},\mathbf{1})= (q_{18},\mathbf{1},L)$,
$\delta(q_{18},\#) = (q_{19},\#,R)$

10.7. a) $\delta(q_0,\sigma) = (q_1,\#,R)$, $\delta(q_1,\sigma) = (q_0,\#,R)$, $\delta(q_0,\#) = (q_2,\mathbf{1},S)$

b) $\delta(q_0,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_0,\mathbf{b}) = (q_0,\#,R)$, $\delta(q_1,\sigma) = (q_1,\#,R)$, $\delta(q_1,\#) = (q_2,\mathbf{1},S)$

c) $\delta(q_0,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_1,\mathbf{a}) = (q_0,\#,R)$, $\delta(q_0,\mathbf{b}) = (q_2,\#,R)$, $\delta(q_2,\mathbf{b}) = (q_0,\#,R)$,
$\delta(q_0,\#) = (q_3,\mathbf{1},S)$

d) $\delta(q_0,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_1,\sigma) = (q_1,\sigma,R)$, $\delta(q_1,\#) = (q_2,\#,L)$, $\delta(q_2,\mathbf{b}) = (q_3,\#,L)$,
$\delta(q_3,\sigma) = (q_3,\sigma,L)$, $\delta(q_3,\#) = (q_0,\#,R)$, $\delta(q_0,\mathbf{b}) = (q_4,\#,R)$, $\delta(q_4,\mathbf{b}) = (q_4,\#,R)$,
$\delta(q_4,\#) = (q_6,\mathbf{1},S)$, $\delta(q_2,\mathbf{a}) = (q_5,\#,L)$, $\delta(q_5,\mathbf{a}) = (q_5,\#,L)$, $\delta(q_5,\#) = (q_6,\mathbf{1},S)$

e) $\delta(q_0,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_1,\sigma) = (q_1,\sigma,R)$, $\delta(q_1,\#) = (q_3,\#,L)$, $\delta(q_3,\mathbf{a}) = (q_5,\#,L)$, $\delta(q_0,\mathbf{b}) = (q_2,\#,R)$, $\delta(q_2,\sigma) = (q_2,\sigma,R)$, $\delta(q_2,\#) = (q_4,\#,L)$, $\delta(q_4,\mathbf{b}) = (q_5,\#,L)$, $\delta(q_5,\sigma) = (q_5,\sigma,L)$, $\delta(q_5,\#) = (q_0,\#,R)$, $\delta(q_0,\#) = (q_6,\mathbf{1},S)$, $\delta(q_3,\#) = (q_6,\mathbf{1},S)$, $\delta(q_4,\#) = (q_6,\mathbf{1},S)$

f) $\delta(q_0,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_1,\sigma) = (q_1,\sigma,R)$, $\delta(q_1,\#) = (q_3,\#,L)$, $\delta(q_3,\mathbf{a}) = (q_5,\#,L)$, $\delta(q_0,\mathbf{b}) = (q_2,\#,R)$, $\delta(q_2,\sigma) = (q_2,\sigma,R)$, $\delta(q_2,\#) = (q_4,\#,L)$, $\delta(q_4,\mathbf{b}) = (q_5,\#,L)$, $\delta(q_5,\sigma) = (q_5,\sigma,L)$, $\delta(q_5,\#) = (q_0,\#,R)$, $\delta(q_4,\mathbf{a}) = (q_6,\#,L)$, $\delta(q_3,\mathbf{b}) = (q_6,\#,L)$, $\delta(q_6,\sigma) = (q_6,\#,L)$, $\delta(q_6,\#) = (q_7,\mathbf{1},S)$

g) $\delta(q_0,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_1,\mathbf{a}) = (q_1,\mathbf{a},R)$, $\delta(q_1,\mathbf{b}) = (q_1,\mathbf{b},R)$, $\delta(q_1,\mathbf{c}) = (q_3,\mathbf{c},R)$, $\delta(q_3,\mathbf{a}) = (q_5,\mathbf{d},L)$, $\delta(q_3,\mathbf{d}) = (q_3,\mathbf{d},R)$, $\delta(q_0,\mathbf{b}) = (q_2,\#,R)$, $\delta(q_2,\mathbf{a}) = (q_2,\mathbf{a},R)$, $\delta(q_2,\mathbf{b}) = (q_2,\mathbf{b},R)$, $\delta(q_2,\mathbf{c}) = (q_4,\mathbf{c},R)$, $\delta(q_4,\mathbf{b}) = (q_5,\mathbf{d},L)$, $\delta(q_5,\mathbf{d}) = (q_5,\mathbf{d},R)$, $\delta(q_5,\sigma) = (q_5,\sigma,L)$, $\delta(q_5,\#) = (q_0,\#,R)$, $\delta(q_0,\mathbf{c}) = (q_6,\#,R)$, $\delta(q_6,\mathbf{d}) = (q_6,\#,R)$, $\delta(q_6,\#) = (q_7,\mathbf{1},S)$

h) $\delta(q_0,\mathbf{b}) = (q_1,\#,R)$, $\delta(q_0,\mathbf{c}) = (q_1,\#,R)$, $\delta(q_1,\sigma) = (q_1,\sigma,R)$, $\delta(q_1,\#) = (q_2,\#,L)$, $\delta(q_2,\mathbf{a}) = (q_3,\#,L)$, $\delta(q_2,\mathbf{c}) = (q_3,\#,L)$, $\delta(q_3,\sigma) = (q_3,\sigma,L)$, $\delta(q_3,\#) = (q_0,\#,R)$, $\delta(q_0,\#) = (q_4,\mathbf{1},S)$

i) $\delta(q_0,\mathbf{a}) = (q_1,\mathbf{d},L)$, $\delta(q_0,\mathbf{b}) = (q_0,\mathbf{b},R)$, $\delta(q_0,\mathbf{c}) = (q_0,\mathbf{c},R)$, $\delta(q_0,\mathbf{d}) = (q_0,\mathbf{d},R)$, $\delta(q_1,\sigma) = (q_1,\sigma,L)$, $\delta(q_1,\#) = (q_2,\#,R)$, $\delta(q_2,\mathbf{a}) = (q_2,\mathbf{a},R)$, $\delta(q_2,\mathbf{b}) = (q_3,\mathbf{d},L)$, $\delta(q_2,\mathbf{c}) = (q_2,\mathbf{c},R)$, $\delta(q_2,\mathbf{d}) = (q_2,\mathbf{d},R)$,$\delta(q_3,\sigma) = (q_3,\sigma,L)$, $\delta(q_3,\#) = (q_4,\#,R)$, $\delta(q_4,\mathbf{a}) = (q_4,\mathbf{a},R)$, $\delta(q_4,\mathbf{b}) = (q_4,\mathbf{b},R)$, $\delta(q_4,\mathbf{c}) = (q_5,\mathbf{d},L)$, $\delta(q_4,\mathbf{d}) = (q_4,\mathbf{d},R)$, $\delta(q_5,\sigma) = (q_5,\sigma,L)$, $\delta(q_5,\#) = (q_0,\#,R)$, $\delta(q_0,\#) = (q_6,\#,L)$, $\delta(q_6,\mathbf{d}) = (q_6,\#,L)$, $\delta(q_6,\#) = (q_7,\mathbf{1},S)$

j) $\delta(q_0,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_1,\sigma) = (q_1,\sigma,R)$, $\delta(q_1,\#) = (q_2,\#,L)$, $\delta(q_2,\mathbf{b}) = (q_3,\#,L)$, $\delta(q_3,\mathbf{b}) = (q_4,\#,L)$, $\delta(q_4,\sigma) = (q_4,\sigma,L)$, $\delta(q_4,\#) = (q_0,\#,R)$, $\delta(q_0,\#) = (q_5,\mathbf{1},S)$

k) $\delta(q_0,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_1,\mathbf{a}) = (q_1,\mathbf{a},R)$, $\delta(q_1,\mathbf{b}) = (q_2,\mathbf{b},R)$, $\delta(q_2,\mathbf{b}) = (q_2,\mathbf{b},R)$, $\delta(q_2,\mathbf{c}) = (q_2,\mathbf{c},R)$, $\delta(q_2,\mathbf{a}) = (q_3,\mathbf{c},L)$, $\delta(q_3,\sigma) = (q_3,\sigma,L)$, $\delta(q_3,\#) = (q_4,\#,R)$, $\delta(q_4,\mathbf{a}) = (q_1,\#,R)$, $\delta(q_4,\mathbf{b}) = (q_5,\#,R)$, $\delta(q_5,\mathbf{b}) = (q_5,\mathbf{b},R)$, $\delta(q_5,\mathbf{c}) = (q_6,\mathbf{c},R)$, $\delta(q_6,\mathbf{c}) = (q_6,\mathbf{c},R)$, $\delta(q_6,\mathbf{b}) = (q_7,\#,L)$, $\delta(q_7,\sigma) = (q_7,\sigma,L)$, $\delta(q_7,\#) = (q_8,\#,R)$, $\delta(q_8,\mathbf{b}) = (q_5,\#,R)$, $\delta(q_8,\mathbf{c}) = (q_9,\mathbf{c},R)$, $\delta(q_9,\mathbf{c}) = (q_9,\mathbf{c},R)$, $\delta(q_9,\#) = (q_{10},\mathbf{1},S)$

l) $\delta(q_0,\mathbf{a}) = (q_1,\mathbf{b},R)$, $\delta(q_1,\mathbf{a}) = (q_2,\mathbf{a},L)$, $\delta(q_2,\mathbf{c}) = (q_2,\mathbf{c},L)$, $\delta(q_2,\mathbf{b}) = (q_3,\mathbf{c},R)$, $\delta(q_3,\mathbf{c}) = (q_3,\mathbf{c},R)$, $\delta(q_3,\mathbf{a}) = (q_2,\mathbf{c},L)$, $\delta(q_2,\#) = (q_1,\#,R)$, $\delta(q_1,\mathbf{c}) = (q_1,\mathbf{b},R)$, $\delta(q_1,\#) = (q_4,\#,L)$, $\delta(q_4,\mathbf{b}) = (q_4,\#,L)$, $\delta(q_4,\#) = (q_5,\mathbf{1},S)$

10.8. $\delta(q_0,(\mathbf{0},\mathbf{0},\#)) = (q_0,(\mathbf{0},\mathbf{0},\mathbf{0}),L)$, $\delta(q_0,(\mathbf{1},\mathbf{0},\#)) = (q_0,(\mathbf{1},\mathbf{0},\mathbf{1}),L)$,
$\delta(q_0,(\mathbf{0},\mathbf{1},\#)) = (q_1,(\mathbf{0},\mathbf{1},\mathbf{1}),L)$, $\delta(q_0,(\mathbf{1},\mathbf{1},\#)) = (q_0,(\mathbf{1},\mathbf{1},\mathbf{0}),L)$,
$\delta(q_1,(\mathbf{0},\mathbf{0},\#)) = (q_1,(\mathbf{0},\mathbf{0},\mathbf{1}),L)$, $\delta(q_1,(\mathbf{1},\mathbf{0},\#)) = (q_0,(\mathbf{1},\mathbf{0},\mathbf{0}),L)$,
$\delta(q_1,(\mathbf{0},\mathbf{1},\#)) = (q_1,(\mathbf{0},\mathbf{1},\mathbf{0}),L)$, $\delta(q_1,(\mathbf{1},\mathbf{1},\#)) = (q_1,(\mathbf{1},\mathbf{1},\mathbf{1}),L)$,
$\delta(q_0,(\#,\#,\#)) = (q_2,(\#,\#,\#),R)$

10.9. $\delta(q_0,(\mathbf{a},\#)) = (q_1,(\mathbf{a},\mathbf{x}),R)$, $\delta(q_1,(\mathbf{a},\#)) = (q_1,(\mathbf{a},\#),R)$, $\delta(q_1,(\mathbf{b},\#)) = (q_1,(\mathbf{b},\#),R)$,
$\delta(q_1,(\mathbf{c},\#)) = (q_3,(\mathbf{c},\#),R)$, $\delta(q_3,(\sigma,\mathbf{y})) = (q_3,(\sigma,\mathbf{y}),R)$, $\delta(q_3,(\mathbf{a},\#)) = (q_5,(\mathbf{a},\mathbf{y}),L)$,
$\delta(q_0,(\mathbf{b},\#)) = (q_2,(\mathbf{b},\mathbf{x}),R)$, $\delta(q_2,(\mathbf{a},\#)) = (q_2,(\mathbf{a},\#),R)$, $\delta(q_2,(\mathbf{b},\#)) = (q_2,(\mathbf{b},\#),R)$,
$\delta(q_2,(\mathbf{c},\#)) = (q_4,(\mathbf{c},\#),R)$, $\delta(q_4,(\sigma,\mathbf{y})) = (q_4,(\sigma,\mathbf{y}),R)$, $\delta(q_4,(\mathbf{b},\#)) = (q_5,(\mathbf{b},\mathbf{y}),L)$,
$\delta(q_5,(\sigma,\mathbf{y})) = (q_5,(\sigma,\mathbf{y}),L)$, $\delta(q_5,(\sigma,\#)) = (q_5,(\sigma,\#),L)$, $\delta(q_5,(\sigma,\mathbf{x})) = (q_0,(\sigma,\mathbf{x}),R)$,
$\delta(q_0,(\mathbf{c},\#)) = (q_6,(\mathbf{c},\#),R)$, $\delta(q_6,(\sigma,\mathbf{y})) = (q_6,(\sigma,\mathbf{y}),R)$, $\delta(q_6,(\#,\#)) = (q_7,(\#,\#),S)$

10.10. $\delta(q_0,(\sigma,\#)) = (q_1,(\sigma,\mathbf{x}),R)$, $\delta(q_1,(\sigma,\#)) = (q_1,(\sigma,\#),R)$, $\delta(q_2,(\#,\#)) = (q_3,(\#,\#),L)$,
$\delta(q_3,(\sigma,\#)) = (q_4,(\sigma,\mathbf{x}),L)$, $\delta(q_4,(\sigma,\#)) = (q_4,(\sigma,\#),L)$, $\delta(q_4,(\sigma,\mathbf{x})) = (q_0,(\sigma,\mathbf{x}),R)$,
$\delta(q_0,(\sigma,\mathbf{x})) = (q_5,(\sigma,\mathbf{y}),L)$, $\delta(q_5,(\sigma,\mathbf{x})) = (q_5,(\sigma,\#),L)$, $\delta(q_5,(\#,\#)) = (q_6,(\#,\#),R)$,
$\delta(q_6,(\mathbf{a},\#)) = (q_7,(\mathbf{a},\mathbf{x}),R)$, $\delta(q_7,(\sigma,\#)) = (q_7,(\sigma,\#),R)$, $\delta(q_7,(\sigma,\mathbf{y})) = (q_7,(\sigma,\mathbf{y}),R)$,
$\delta(q_6,(\mathbf{b},\#)) = (q_8,(\mathbf{b},\mathbf{x}),R)$, $\delta(q_8,(\sigma,\#)) = (q_8,(\sigma,\#),R)$, $\delta(q_8,(\sigma,\mathbf{y})) = (q_8,(\sigma,\mathbf{y}),R)$,
$\delta(q_7,(\mathbf{a},\mathbf{x})) = (q_9,(\mathbf{a},\mathbf{y}),L)$, $\delta(q_8,(\mathbf{b},\mathbf{x})) = (q_9,(\mathbf{b},\mathbf{y}),L)$, $\delta(q_9,(\sigma,\mathbf{y})) = (q_9,(\sigma,\mathbf{y}),L)$,
$\delta(q_9,(\sigma,\#)) = (q_9,(\sigma,\#),L)$, $\delta(q_9,(\sigma,\mathbf{x})) = (q_6,(\sigma,\mathbf{x}),R)$, $\delta(q_6,(\sigma,\mathbf{y})) = (q_{10},(\sigma,\mathbf{y}),R)$,
$\delta(q_{10},(\sigma,\mathbf{y})) = (q_{10},(\sigma,\mathbf{y}),R)$, $\delta(q_{10},(\#,\#)) = (q_{11},(\#,\#),S)$

10.11. $\delta(q_0,(\sigma, \#)) = (q_0,(\sigma, \sigma),(R, R))$, $\delta(q_0,(\#, \#)) = (q_1,(\#, \#),(S, L))$,
$\delta(q_1,(\#, \sigma)) = (q_1,(\#, \sigma),(S, L))$, $\delta(q_1,(\#, \#)) = (q_2,(\#, \#),(L, R))$,
$\delta(q_2,(\sigma, \sigma)) = (q_3,(\sigma, \sigma),(L, R))$, $\delta(q_3,(\sigma, \sigma)) = (q_2,(\sigma, \sigma),(L, R))$,
$\delta(q_2,(\#, \#)) = (q_4,(\#, \#),(R, S))$

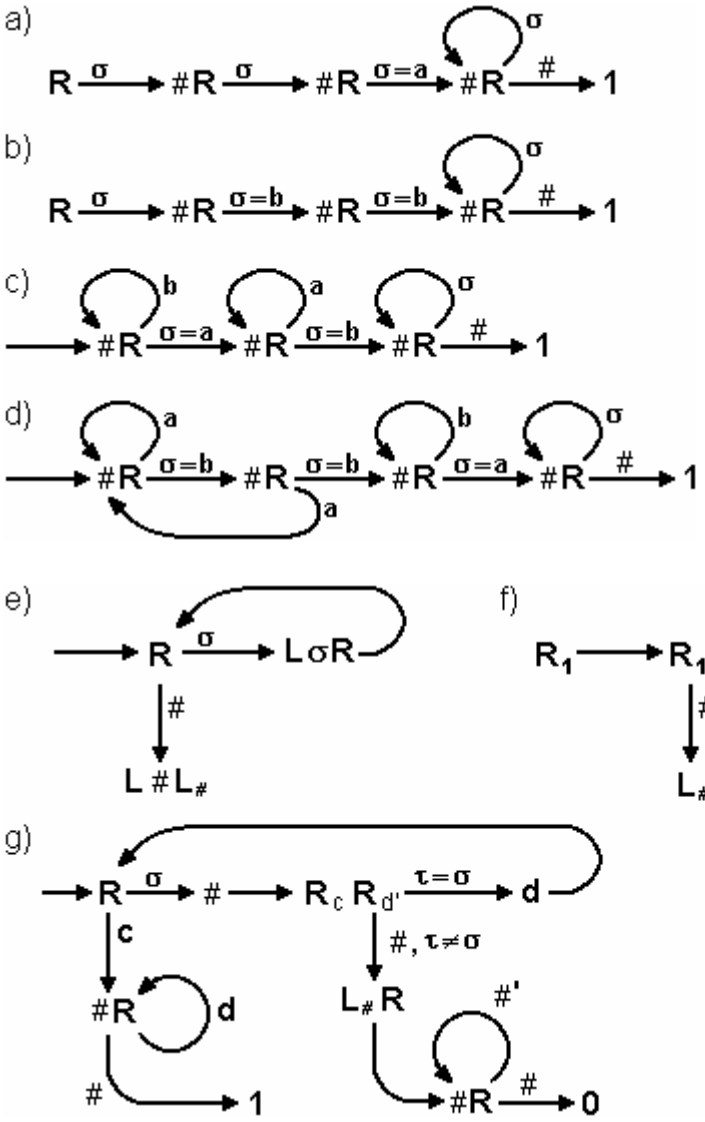
10.12. $\delta(q_0,(\mathbf{a}, \#)) = (q_0,(\mathbf{a}, \mathbf{a}),(R, R))$, $\delta(q_0,(\mathbf{b}, \#)) = (q_0,(\mathbf{b}, \#),(R, S))$,
$\delta(q_0,(\#, \#)) = (q_1,(\#, \#),(L, L))$, $\delta(q_1,(\mathbf{b}, \mathbf{a})) = (q_1,(\mathbf{b}, \#),(L, L))$,
$\delta(q_1,(\mathbf{a}, \mathbf{a})) = (q_1,(\mathbf{a}, \mathbf{a}),(L, S))$, $\delta(q_1,(\mathbf{a}, \#)) = (q_1,(\mathbf{a}, \#),(L, S))$,
$\delta(q_1,(\#, \#)) = (q_2,(\#, \#),(R, S))$

10.13. $\delta(q_0,(\sigma, \#)) = (q_0,(\sigma, \sigma),(R, R))$, $\delta(q_0,(\#, \#)) = (q_1,(\#, \#),(S, L))$,
$\delta(q_1,(\#, \sigma)) = (q_1,(\#, \sigma),(S, L))$, $\delta(q_1,(\#, \#)) = (q_2,(\#, \#),(S, R))$,
$\delta(q_2,(\#, \sigma)) = (q_2,(\sigma, \sigma),(R, R))$, $\delta(q_2,(\#, \#)) = (q_3,(\#, \#),(L, S))$,
$\delta(q_3,(\sigma, \#)) = (q_3,(\sigma, \#),(L, S))$, $\delta(q_3,(\#, \#)) = (q_4,(\#, \#),(R, S))$

10.14. a) $\delta(q_0,\sigma) = (q_1,\sigma,L)$, $\delta(q_0,\#) = (q_1,\#,L)$, $\delta(q_1,\sigma) = (q_1,\sigma,L)$, $\delta(q_1,\#) = (q_2,\#,R)$, $\delta(q_2,\sigma) = (q_3,\sigma,L)$, $\delta(q_2,\#) = (q_3,\#,L)$

b) $\delta(q_0,\sigma) = (q_1,\sigma,R)$, $\delta(q_0,\#) = (q_1,\#,R)$, $\delta(q_1,\sigma) = (q_2,\sigma,L)$, $\delta(q_1,\#) = (q_1,\#,R)$, $\delta(q_2,\sigma) = (q_3,\sigma,R)$, $\delta(q_2,\#) = (q_3,\#,R)$

10.15.



11.1. a) S → **a**SBC | **b**DC; D → **b**DC | EC, E → EC | ε, CB → BC, **a**B → **ab**, **b**B → **bb**, **b**C → **bc**, **c**C → **cc**.

b) S → **a**SBCD | ε, DB → BD, DC → CD, CB → BC, **a**B → **ab**, **b**B → **bb**, **b**C → **bc**, **c**C → **cc**, **c**D → **cd**, **d**D → **dd**.

c) $S \to [E]$, $E \to ABECD \mid \varepsilon$, $BA \to AB$, $DC \to CD$, $[A \to \mathbf{a}$, $\mathbf{a}A \to \mathbf{aa}$, $\mathbf{a}B \to \mathbf{ab}$, $\mathbf{b}B \to \mathbf{bb}$, $D] \to \mathbf{b}$, $D\mathbf{b} \to \mathbf{bb}$, $C\mathbf{b} \to \mathbf{ab}$, $C\mathbf{a} \to \mathbf{aa}$.

d) $S \to [\mathbf{a}]$, $D] \to]$, $] \to \varepsilon$, $[\to [D$, $[\to \varepsilon$, $D\mathbf{a} \to \mathbf{aa}D$.

e) $S \to MN]$, $] \to A\mathbf{a}] \mid B\mathbf{b}]$, $\mathbf{a}A \to A\mathbf{a}$, $\mathbf{a}B \to B\mathbf{a}$, $\mathbf{b}A \to A\mathbf{b}$, $\mathbf{b}B \to B\mathbf{b}$, $M \to \varepsilon$, $NA \to A\mathbf{a}N$, $NB \to B\mathbf{b}N$, $MA \to \mathbf{a}M$, $MB \to \mathbf{b}M$, $N \to \varepsilon$, $] \to \varepsilon$.

11.2. $\delta(q_0,\mathbf{a}) = (q_1,\mathbf{a},R)$, $\delta(q_0,\mathbf{b}) = (q_0,\mathbf{b},R)$, $\delta(q_1,\mathbf{a}) = (q_1,\mathbf{a},R)$, $\delta(q_1,\mathbf{b}) = (q_0,\mathbf{b},R)$, $\delta(q_0,\#) = (q_R,\#,S)$, $\delta(q_1,\#) = (q_A,\#,S)$.

11.3. $\delta(q_0,\mathbf{a}) = (q_1,\mathbf{a},R)$, $\delta(q_1,\mathbf{b}) = (q_2,\mathbf{b},R)$, $\delta(q_2,\mathbf{a}) = (q_1,\mathbf{a},R)$, $\delta(q_2,\#) = (q_A,\#,S)$, $\delta(q_0,\mathbf{b}) = (q_R,\mathbf{b},S)$, $\delta(q_1,\mathbf{a}) = (q_R,\mathbf{a},S)$, $\delta(q_2,\mathbf{b}) = (q_R,\mathbf{b},S)$.

11.4. $\delta(q_0',(\mathbf{a},\#)) = (q_0,(\mathbf{a},Z),(S,S))$, $\delta(q_0',(\mathbf{b},\#)) = (q_0,(\mathbf{b},Z),(S,S))$,
$\delta(q_0,(\mathbf{a},Z)) = (p,(\mathbf{a},Z),(S,R))$, $\delta(q_0,(\mathbf{b},Z)) = (p,(\mathbf{b},Z),(S,R))$,
$\delta(q_0,(\mathbf{a},A)) = (p,(\mathbf{a},A),(S,R))$, $\delta(q_0,(\mathbf{b},B)) = (p,(\mathbf{b},B),(S,R))$,
$\delta(p,(\mathbf{a},\#)) = (q_0,(\mathbf{a},A),(R,S))$, $\delta(p,(\mathbf{a},\#)) = (q_0,(\mathbf{b},B),(R,S))$,
$\delta(q_0,(\mathbf{a},B)) = (q_0,(\mathbf{a},\#),(R,L))$, $\delta(q_0,(\mathbf{b},A)) = (q_0,(\mathbf{b},\#),(R,L))$,
$\delta(q_0,(\#,Z)) = (q_A,(\#,\#),(S,S))$.

11.5. $S \to [A$, $A \to \mathbf{x}A \mid A\mathbf{x} \mid \#A \mid A\# \mid q_3$, $\mathbf{x}q_1 \to q_0\mathbf{a}$, $\mathbf{x}q_2 \to q_0\mathbf{b}$, $\mathbf{x}q_0 \to q_1\mathbf{a} \mid q_2\mathbf{b}$,
$q_3\mathbf{x}\# \to \mathbf{x}q_0\#$, $q_3\#\# \to \#q_0\#$, $\# \to \varepsilon$, $[q_0 \to \varepsilon$
$S \Rightarrow [A \Rightarrow [\mathbf{x}A \Rightarrow [\mathbf{xx}A \Rightarrow [\mathbf{xxx}A \Rightarrow [\mathbf{xxxx}A \Rightarrow [\mathbf{xxxxx}A \Rightarrow [\mathbf{xxxxx}A\mathbf{x} \Rightarrow$
$[\mathbf{xxxxx}A\mathbf{x}\# \Rightarrow [\mathbf{xxxxx}q_3\mathbf{x}\# \Rightarrow [\mathbf{xxxxx}q_0\# \Rightarrow [\mathbf{xxxxx}q_1\mathbf{a}\# \Rightarrow [\mathbf{xxxx}q_0\mathbf{aa}\# \Rightarrow$
$[\mathbf{xxx}q_2\mathbf{baa}\# \Rightarrow [\mathbf{xx}q_0\mathbf{bbaa}\# \Rightarrow [\mathbf{x}q_1\mathbf{abbaa}\# \Rightarrow [q_0\mathbf{aabbaa}\# \Rightarrow \mathbf{aabbaa}$

12.3. a) Si, b) Si, c) No, d) No, e) No, f) Si. g) No, h) Si, i) No.

12.4. a) $i_1 = 1$, $i_2 = 2$, $i_3 = 1$, b) No, c) $i_1 = 1$, $i_2 = 3$, $i_3 = 2$, $i_4 = 2$, d) $i_1 = 3$, $i_2 = 1$, e) No, f) No, g) $i_1 = 1$, $i_2 = 2$, $i_3 = 1$, $i_4 = 3$, h) No, i) $i_1 = 1$, $i_2 = 4$, $i_3 = 2$, j) No, k) No.

12.5. a) $i_1 = 1$, $i_2 = 2$, $i_3 = 3$, $i_4 = 1$, b) No, c) $i_1 = 1$, $i_2 = 3$, $i_3 = 2$, $i_4 = 4$, $i_5 = 4$, $i_6 = 3$, d) No.

13.1. a) $\pi^2_1(3,5) = 3$; $\sigma(3) = 4$
b) $\sigma(4) = 5$; $\pi^1_1(5) = 5$
c) $\zeta() = 0$; $\sigma(0) = 1$
d) $\pi^3_0(5,6,2) = ()$; $\zeta() = 0$; $\sigma(0) = 1$

13.2. f y g no están bien definidas, h y k si lo están.

13.3. a) $\pi^3_2 \times \pi^3_1(4, 2, 5) = (\pi^3_2(4, 2, 5), \pi^3_1(4, 2, 5)) = (2, 4)$
b) $\pi^3_1 \times \pi^3_2(4, 2, 5) = (\pi^3_1(4, 2, 5), \pi^3_2(4, 2, 5)) = (4, 2)$
c) $(\sigma \circ \pi^3_2) \times (\zeta \circ \pi^3_0)(4, 2, 5) = (\sigma \circ \pi^3_2(4, 2, 5), \zeta \circ \pi^3_0(4, 2, 5)) = (3, 0)$
d) $((\sigma \circ \pi^2_2) \times \pi^2_1) (4,6) = ((\sigma \circ \pi^2_2) (4,6) \times \pi^2_1(4,6)) = (7, 4)$, luego
$\pi^2_2 \times (\zeta \circ \pi^2_0) \times (\sigma \circ \sigma \circ \pi^2_2)(7,4) = (\pi^2_2(7,4), \zeta \circ \pi^2_0(7,4), \sigma \circ \sigma \circ \pi^2_2(7,4)) =$
(4, 0, 6)

13.4. a) $f(2, 3, 4) = \pi^3_2 \times \pi^3_3 \times K^3_7 (2, 3, 4) = (3, 4, 7)$
b) $g(2,3) = \pi^2_2 \times \pi^2_2 \times K^3_9 (2,3) = (3, 3, 9)$
c) $h(2, 3, 4) = K^3_7 \times \pi^3_1 (2, 3, 4) = (7, 2)$

13.5. a) $Mas(x, 0) = \pi^1_1(x)$, $Mas(x, y+1) = \sigma \circ \pi^3_3(x, y, Mas(x, y))$
b) $Mult(x, 0) = K^1_0(x)$, $Mult(x, y+1) = Mas \circ (\pi^3_1 \times \pi^3_3)(x, y, Mult(x, y))$
c) $Exp(x, 0) = K^1_1(x)$, $Exp(x, y+1) = Mult \circ (\pi^3_1 \times \pi^3_3)(x, y, Exp(x, y))$

13.6. Caso base: $A(1,0) = A(0,1) = 1 + 1 = 2$, Si es cierto para $z = n$, entonces es cierto para $z = n+1$: $A(1,n+1) = A(0,A(1,n)) = A(0,n+2) = (n+2)+1 = (n+1)+2$.

14.1. Para $n_0 = 2$ y $c = 6$ se verifica que $f(n) \leq cg(n)$, para $n_0 \leq n$.

14.2. $T_m(n) = n + 2$, $T_p(n) = (3n+4)/2$

14.4. a) $O(n^2)$, b) $O(n^2 \ln n)$, c) $O(n^2)$, d) $O(n!)$, e) $O(n^2)$.

Esta obra se imprimió en los talleres de la
EDITORIAL
ACD
17 Sur No. 3105, Col. Volcanes
Puebla, Puebla, México.

La edición consta de 500 ejemplares
más sobrantes para reposición

14.5.