Apuntes de Teoría de Autómatas y Lenguajes Formales

Gloria Martínez Luis A. García

27 de octubre de 2005

Índice general

Capítulo 1

Introducción

Índice General

1.1. Alfabetos y Cadenas.

Un *alfabeto* es un conjunto *finito y no vacío* de elementos denominados *símbolos*. Los alfabetos se definen bien sea por *extensión*, enumeración de sus símbolos, o bien por *comprensión*, enunciando alguna propiedad que todos los símbolos han de cumplir.

Algunos ejemplos de alfabetos son los siguientes:

1. Σ = {*a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z*}, *el alfabeto latino,*
2. Σ = {*0, 1*}, *el alfabeto binario,*
3. $\Sigma = \{a_i \mid i \in \mathcal{I}\}$, *donde* $\mathcal{I}$ *es un conjunto finito, como por ejemplo* $\mathcal{I} = \{x \in N \mid x > 0 \wedge x < 50\}$.

Los dos primeros alfabetos se han definido por extensión, mientras que el tercero se ha definido por comprensión.

Dado un alfabeto una *cadena* de ese alfabeto es una combinación de símbolos de ese alfabeto.

Por ejemplo, con el alfabeto (1) una cadena sería "mar", con el alfabeto (2) sería "0111" y con el alfabeto (3) sería "$a_1a_5a_{49}$*".*

Se define una cadena especial denominada *cadena nula*, denotada por λ, caracterizada por no poseer símbolos.

Se denomina *prefijo* a cualquier secuencia de símbolos de la parte inicial de una cadena, y *sufijo* a cualquier secuencia de símbolos de su parte final.

Por ejemplo, la cadena "0111" tiene como prefijos a las siguientes cadenas: λ*, "0", "01", "011" y "0111" mientras que son sus sufijos las cadenas:* λ*, "1", "11", "111", "0111".*

Existe un gran número de operaciones básicas que se pueden realizar con las cadenas. De entre ellas, se destacan:

Longitud de una cadena Se denomina longitud de una cadena *x*, denotado por |x|, a la operación que consiste en contar el número de símbolos que componen a la cadena *x*. Esta operación cumple las siguientes propiedades:

1. $| \lambda |$ = 0
2. |a| = 1, si *a* es un símbolo del alfabeto
3. Si y = $a_1a_2..a_n$, x = $a_1a_2..a_nb$ y $a_i \in \Sigma, b \in \Sigma$, entonces |y| = n $\wedge$ |x| = n + 1.

Dado un alfabeto Σ, se define Σ^n como el conjunto de cadenas de longitud *n* que se pueden construir con ese alfabeto. Así, por ejemplo, siempre se cumple que sea cual sea el alfabeto Σ, $\Sigma^0 = \{\lambda\}$.

Concatenación Sean *x* e *y* dos cadenas, entonces la cadena resultante de concatenar *x* con *y* es la cadena *xy*, cumpliéndose que si $| x |$ = n y $| y |$ = m entonces $| xy |$ = $| x |$ + $| y |$ = n + m. Cuando se concatena una cadena consigo misma varias veces se emplea la potencia para denotarlo; por ejemplo, la cadena *xxxx* se denota como x^4. Además, como consecuencia del uso de esta notación, resulta que $| x^n |$ = n$| x |$.

Estrella de Kleene Se denomina estrella de Kleene, *cierre reflexivo transitivo* u *operación asterisco* sobre un alfabeto Σ al conjunto de todas las cadenas que se pueden construir a partir de los símbolos del alfabeto Σ; en notación matemática esta definición se expresa así: $\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$.

Cierre transitivo El cierre transitivo u *operación más*, se denota como Σ^+ y se define como $\Sigma^+ = \bigcup_{i>0} \Sigma^i$.

De las definiciones de cierre reflexivo transitivo y de cierre transitivo, se puede deducir que $\Sigma^* = \{\lambda\} \cup \Sigma^+$.

Inversión Dada una cadena $x \in \Sigma^*$, $x = a_1a_2 \ldots a_n$, se define la cadena inversa de *x*, denotada por x^{-1}, a la cadena $a_n \ldots a_2a_1$. Esta operación se puede definir también recursivamente:

1. $\lambda^{-1} = \lambda$
2. $(xa)^{-1} = a(x^{-1})$, con $x \in \Sigma^* \wedge a \in \Sigma$.

De esta definición se desprenden varias propiedades entre las que cabe destacar las siguientes:

1. $a^{-1} = a$ si $a \in \Sigma$, la inversa de un símbolo es el mismo símbolo,
2. $(x^{-1})^{-1} = x$, $x \in \Sigma^*$, la inversa de la inversa es la cadena original,
3. $(x_1 x_2 \dots x_k)^{-1} = x_k^{-1} \dots x_2^{-1} x_1^{-1}$, $x_i \in \Sigma^*$.

1.2. Lenguajes.

Dado un alfabeto Σ se dice que *L* es un *lenguaje* si es un subconjunto de Σ^*, es decir, $L \subseteq \Sigma^*$.

Por ejemplo, sea el alfabeto Σ = {0,1}. L_1 = {0, 01, 011} *es un lenguaje finito sobre el alfabeto* Σ *y* $L_2 = \{0^{2n} \mid n \geq 0\}$ *es otro lenguaje, esta vez con un número de cadenas infinito, sobre el mismo alfabeto.*

Puesto que se ha definido un lenguaje como un conjunto de cadenas, las operaciones que se pueden realizar sobre lenguajes son las mismas que las que se pueden realizar sobre conjuntos. De entre ellas, se destacan las siguientes:

Unión de lenguajes $L_1 \cup L_2 = \{x \in \Sigma^* \mid x \in L_1 \vee x \in L_2\}$.

Cumple las propiedades

1. Asociativa, $L_1 \cup L_2 \cup L_3 = (L_1 \cup L_2) \cup L_3 = L_1 \cup (L_2 \cup L_3)$,
2. Commutativa, $L_1 \cup L_2 = L_2 \cup L_1$, y
3. Elemento neutro, $L \cup \emptyset = \emptyset \cup L = L$.

Intersección de lenguajes $L_1 \cap L_2 = \{x \in \Sigma^* \mid x \in L_1 \wedge x \in L_2\}$.

Cumple las propiedades

1. Asociativa, $L_1 \cap L_2 \cap L_3 = (L_1 \cap L_2) \cap L_3 = L_1 \cap (L_2 \cap L_3)$,
2. Commutativa, $L_1 \cap L_2 = L_2 \cap L_1$, y
3. Elemento neutro, $L \cap \Sigma^* = \Sigma^* \cap L = L$.

Concatenación $L_1 L_2 = \{x \in \Sigma^* \mid x = yz \wedge y \in L_1 \wedge z \in L_2\}$.

Cumple las propiedades

1. Asociativa, $L_1 L_2 L_3 = (L_1 L_2) L_3 = L_1 (L_2 L_3)$, y
2. Elemento neutro, $L\{\lambda\} = \{\lambda\} L = L$.

En la concatenación de lenguajes también se utiliza la notación potencia tal y como se definió en la concatenación de cadenas, es decir, el lenguaje *LLLL* se denota como L^4.

Estrella de Kleene $L^* = \bigcup_{i \geq 0} L^i$.

Cierre transitivo $L^+ = \bigcup_{i > 0} L^i$.

Al aplicar las operaciones *asterisco* y *más* sobre lenguajes, se puede comprobar que se cumplen las siguientes propiedades:

a) $(L^*)^* = L^*$ b) $L^+ = (L^*)L$
c) $(L^+)^* = L^*$ d) $(L^*)^+ = L^*$
e) $(L^+)^+ = L^+$ f) si $L_1 \subseteq L_2$, entonces $L_1^* \subseteq L_2^* \wedge L_1^+ \subseteq L_2^+$

Dependiendo de si λ pertenece o no pertenece a un lenguaje *L* se establece la siguiente relación entre la operación *más* y la operación *asterisco*:

$$L^+ = L^*, \ si \ \lambda \in L \ y \ L^+ = L^* - \{\lambda\}, \ si \ \lambda \notin L.$$

Diferencia de lenguajes $L_1 - L_2 = \{x \in \Sigma^* \mid x \in L_1 \wedge x \notin L_2\}$.

Se cumple que si $L_1 \subseteq L_2$ entonces $L_1 - L_2 = \emptyset$ puesto que todas las cadenas de L_1 pertenecen a L_2, y que si $L_1 \cap L_2 = \emptyset$ entonces $L_1 - L_2 = L_1$, puesto que ambos lenguajes no tienen cadenas en común.

Complemento Sea *L* un lenguaje, entonces el lenguaje complementario de *L*, denotado por $\bar{L}$, se define como $\bar{L} = \Sigma^* - L = \{x \in \Sigma^* \mid x \notin L\}$.

Cumple las propiedades

1. $\bar{\bar{L}} = L$,
2. $L \cup \bar{L} = \Sigma^*$, y
3. $L \cap \bar{L} = \emptyset$.

Partes de un conjunto Una de las operaciones básicas sobre conjuntos consiste en calcular *el conjunto de todos los conjuntos que se pueden formar con los elementos del conjunto original*. Esta operación se denomina *cálculo de las partes de un conjunto* y, si el conjunto se denomina *A*, se denota como *P(A)* o 2^A.

Por ejemplo, si A es el conjunto A = {a,b,c} *entonces el cálculo de las partes del conjunto A,* 2^A *o P(A), es el siguiente:*

$$2^A = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\}$$

Una vez establecida la operación partes de un conjunto queda por destacar cual es el resultado al aplicarla a un conjunto con un número infinito de elementos. Por ejemplo, dado un alfabeto Σ, considérese 2^{Σ^*}. Con esta operación se denota al conjunto de todos los conjuntos que se pueden formar con los elementos del alfabeto Σ, es decir, todos los posibles lenguajes que se pueden formar con las cadenas de símbolos del alfabeto Σ.

La observación anterior permite realizar una consideración interesante. Sea el alfabeto $\Sigma = \{0,1\}$; en el conjunto 2^{Σ^*} estarían, entre otros muchos conjuntos, el conjunto de cadenas que representan el juego de instrucciones de cualquier microprocesador (68000, pentium, athlon, ...etc.). Más aún, si se concatena en una misma cadena todas las instrucciones de un programa escrito en uno de estos juegos de instrucciones, también se obtiene un elemento de 2^{Σ^*}. Se tiene, entonces, que, de entre los elementos de 2^{Σ^*}, un subconjunto sería el formado por todas cadenas que representan programas correctamente escritos en ese juego de instrucciones, y que, además, un subconjunto de este sería el de los que no sólo son correctos desde el punto de vista sintáctico, sino que también se ejecutan de forma correcta.

Inversión Dado un lenguaje L, $L \subseteq \Sigma^*$, se define el lenguaje inverso de L, denotado por L^{-1}, como el conjunto de cadenas que resulta de aplicar la operación de inversión a cada una de las cadenas del lenguaje L, es decir, $L^{-1} = \{x^{-1} \mid x \in L\}$.

Se puede comprobar que esta operación cumple las siguientes propiedades:

a) $(\Sigma^{-1}) = \Sigma$ b) $(L_1L_2)^{-1} = L_2^{-1}L_1^{-1}$
c) $(L^i)^{-1} = (L^{-1})^i$ d) $(L^*)^{-1} = (L^{-1})^*, (L_1 \cup L_2)^{-1} = L_1^{-1} \cup L_2^{-1}$
e) $(\Sigma^*)^{-1} = \Sigma^*$ f) $(L^{-1})^{-1} = L$

1.3. El Concepto de Gramática.

Informalmente una gramática no es más que un mecanismo para generar las cadenas que pertenecen a un lenguaje. Este mecanismo define un conjunto finito de reglas que, aplicadas a partir de un único símbolo inicial, son capaces de generar todas sus cadenas.

Por ejemplo, sea la gramática

$$S \to 0S \mid 0A$$
$$A \to 1A \mid 1$$

En el ejemplo se observan dos tipos de símbolos; por un lado, S y A que se denominan símbolos auxiliares *y, por otro lado, 0 y 1 que se denominan* símbolos terminales. *El objetivo es obtener una cadena aplicando* producciones, *que son las reglas que nos indican cómo substituir los símbolos auxiliares. Las cadenas generadas por una gramática están formadas exclusivamente por símbolos terminales.*

Así, para obtener una cadena se parte de S (símbolo inicial, start*) y se substituye por una de sus dos producciones. No hay ningún tipo de preferencias entre ellas, se puede elegir cualquiera:*

$$S \Rightarrow 0S$$

y, a continuación, hay que proceder del mismo modo y substituir la nueva aparición de S por una cualquiera de sus dos producciones:

$$S \Rightarrow 0\mathbf{S} \Rightarrow 00\mathbf{S} \Rightarrow 000\mathbf{S}$$

En este punto, se podría optar por la segunda producción,

$$S \Rightarrow 0\mathbf{S} \Rightarrow 00\mathbf{S} \Rightarrow 000\mathbf{S} \Rightarrow 0000\mathbf{A}$$

y, por lo tanto, ahora habría que continuar substituyendo el auxiliar A por una de sus dos producciones,

$$S \Rightarrow 0\mathbf{S} \Rightarrow 00\mathbf{S} \Rightarrow 000\mathbf{S} \Rightarrow 0000\mathbf{A} \Rightarrow 00001\mathbf{A} \Rightarrow 000011\mathbf{A} \Rightarrow 0000111$$

La cadena 0000111 está formada exclusivamente por terminales, luego se puede concluir que es una cadena del lenguaje generado por la gramática. A continuación, se formalizan estos conceptos.

Definición 1.1 (Gramática) *Una gramática es una cuádrupla*

$$G = \langle \Sigma_A, \Sigma_T, P, S \rangle$$

donde:

Σ_A *es el denominado alfabeto de símbolos auxiliares,*

Σ_T *es el denominado alfabeto de símbolos terminales,*

P *es el denominado conjunto de producciones. Es un conjunto de pares ordenados (α, β) finito y no vacío,*

$$P = \{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n)\},$$

con $\alpha_i \in (\Sigma_A \cup \Sigma_T)^+ \wedge \beta_i \in (\Sigma_A \cup \Sigma_T)^*, i = 1, 2, \ldots, n$.

S *es un símbolo especial denominado símbolo inicial de la gramática, $S \in \Sigma_A$.*

Se denota por Σ al conjunto $\Sigma_A \cup \Sigma_T$ y se exige que $\Sigma_A \cap \Sigma_T = \emptyset$.

Las producciones se definen formalmente como pares (α, β) en los que a α se le denomina *antecedente* de la producción y a β su *consecuente*. La notación habitual de una producción (α, β) es la expresión $\alpha \to \beta$, que ha sido la utilizada en el ejemplo anterior.

Definición 1.2 *Dos cadenas, θ_1 y θ_2 se encuentran en* relación de derivación directa *dentro de la gramática G, denotado por $\theta_1 \Rightarrow_G \theta_2$, cuando*

$$\theta_1 = \gamma\alpha\delta \wedge \theta_2 = \gamma\beta\delta \wedge (\alpha \to \beta) \in P$$

donde α, β, γ y $\delta \in \Sigma^*$.

Es decir, esta relación expresa cómo conseguir de forma directa una cadena θ_2 a partir de otra cadena θ_1: realizando la sustitución de la cadena α por la cadena β (aplicando una producción de la gramática G) dejando el resto de cadenas sin modificar.

Por ejemplo, en la gramática anterior las cadenas $000S$ *y* $0000A$ *están en relación de derivación directa ya que se puede obtener* $0000A$ *de* $000S$ *en un único paso.*

Definición 1.3 *Dos cadenas,* θ_1 *y* θ_2 *se encuentran en* relación de derivación en n pasos *dentro de la gramática G, cuando existen n cadenas* $\mu_1, \mu_2, \ldots, \mu_n \in \Sigma^*$*, tales que* $\theta_1 \Rightarrow_G \mu_1 \Rightarrow_G \mu_2 \Rightarrow_G \ldots \Rightarrow_G \mu_n \Rightarrow_G \theta_2$.

Esta operación se puede generalizar más aún si no se especifica el número de pasos. En este caso se emplea la notación $\theta_1 \overset{*}{\Rightarrow}_G \theta_2$ cuando $n \geq 0$ (es decir, se utilizan cero, una o más derivaciones directas) y se emplea la notación $\theta_1 \overset{+}{\Rightarrow}_G \theta_2$ cuando $n > 0$ (es decir, cuando se utilizan una o más derivaciones directas).

Definición 1.4 *Dada una cadena* $\theta \in \Sigma^*$*, se dice que* θ *es una* forma sentencial *cuando* $S \overset{*}{\Rightarrow}_G \theta$*. Una cadena* θ *se denomina* sentencia *o* palabra *cuando* θ *es una forma sentencial y todos los símbolos de* θ *son símbolos terminales.*

En el ejemplo anterior, las cadenas $000S$*,* $0000A$ *y* $00001A$ *(entre otras) son formas sentenciales. Sin embargo,* 0000111 *ya es una palabra, puesto que sólo está formada por terminales.*

Definición 1.5 *Se define el* lenguaje asociado a una gramática *como el conjunto de palabras*

$$L(G) = \{x \in \Sigma_T^* \mid S \overset{*}{\Rightarrow}_G x\}.$$

En adelante, y siempre que no exista posibilidad de confusión, se suprimirá la identificación de la gramática como subíndice en las flechas que denotan las relaciones de derivación, tal y como ya se hizo en el desarrollo del ejemplo.

¿Cuáles son los lenguajes generados por las siguientes gramáticas?

(a) $S \to 0S \mid 0A$, $A \to 1A \mid 1$

(b) $S \to 0S \mid 1A$, $A \to 1A \mid \lambda$

(c) $S \to aS \mid bA \mid \lambda$, $A \to bA \mid \lambda$

(d) $S \to aS \mid A$, $A \to bA \mid \lambda$

(e) $S \to 0S1 \mid 01$

(f) $S \to 0S1 \mid \lambda$

(g) $S \to 000S111 \mid 000111$

(h) $S \to 000S111$, $0S1 \to 01$

(i) $S \to X \mid Y$, $Y \to aY \mid A$, $A \to bA \mid \lambda$, $X \to 0B1Y$, $B \to 0B1 \mid \lambda$

Para determinar cuál es el lenguaje generado por una gramática se tiene que calcular cuáles son las cadenas que pueden derivarse a partir de su símbolo inicial. Por ejemplo, en la gramática del apartado a) se puede proceder de la siguiente forma:

$S \Rightarrow 0\mathbf{S} \Rightarrow 00\mathbf{S} \Rightarrow 000\mathbf{S} \Rightarrow \ldots$*, es decir, se puede pensar en aplicar primero n veces la producción* $S \to 0S$*, obteniéndose*

$S \overset{n}{\Rightarrow} 0^n S \Rightarrow \ldots$*, tras lo cual se aplica la producción* $S \to 0A$*, siendo el resultado*

$S \overset{n}{\Rightarrow} 0^n\mathbf{S} \Rightarrow 0^n0A \Rightarrow \ldots$*, tras lo cual se puede aplicar varias veces la producción* $A \to 1A$*,*

$S \overset{n}{\Rightarrow} 0^nS \Rightarrow 0^n0\mathbf{A} \Rightarrow 0^n01\mathbf{A} \Rightarrow 0^n011\mathbf{A} \Rightarrow 0^n0111\mathbf{A} \ldots$

Si se aplica m veces $A \to 1A$ *y ya, por último,* $A \to 1$*, se llega a*

$S \overset{n}{\Rightarrow} 0^nS \Rightarrow 0^n0A \overset{m}{\Rightarrow} 0^n01^m\mathbf{A} \Rightarrow 0^n01^m1.$

Por lo tanto el lenguaje generado por la gramática del apartado a) es el siguiente:

$$L(G_{(a)}) = \{x \in \{0,1\}^* \mid x = 0^n1^m, \text{ con } n, m \geq 1\}$$

Se propone como ejercicio la obtención de los demás lenguajes.

Definición 1.6 *Dos gramáticas se denominan* equivalentes *cuando ambas generan el mismo lenguaje. Es decir, dadas dos gramáticas* G_1 *y* G_2 *entonces* $G_1 \equiv G_2 \Leftrightarrow L(G_1) = L(G_2)$.

Además, una gramática sólo puede generar un lenguaje mientras que un lenguaje puede ser generado por varias gramáticas (de hecho, hay infinitas gramáticas que son capaces de generar un mismo lenguaje).

Se propone como ejercicio determinar cuáles de las gramáticas anteriores son equivalentes.

1.4. Clasificación Jerárquica de las Gramáticas.

Uno de los posibles criterios para clasificar las gramáticas es hacerlo de acuerdo al formato utilizado para describir al conjunto de producciones. Esta clasificación fue establecida por el lingüista Noam Chomsky.

Gramáticas de tipo 3: También denominadas *regulares*. Las hay de dos tipos, según sean el formato de sus producciones:

- *Lineales a la derecha*, con producciones

$$A \to aB$$
$$A \to a$$

- *Lineales a la izquierda*, con producciones

$$A \to Ba$$
$$A \to a$$

En ambos casos, $A, B \in \Sigma_A,\ a \in \Sigma_T^*$.

Hay que destacar que no se permite mezclar en una misma gramática de tipo 3 producciones lineales a la derecha con producciones lineales a la izquierda.

Gramáticas de tipo 2: También denominadas *de contexto libre*. Son las gramáticas caracterizadas porque su conjunto de producciones presenta el siguiente formato:

$$A \to \beta \; donde \; A \in \Sigma_A,\ \beta \in \Sigma^*.$$

En las gramáticas de tipo 2 el símbolo A siempre se puede sustituir por la cadena β *independientemente* del contexto en el que aparezca.

Gramáticas de tipo 1: También se les conoce como *gramáticas sensibles al contexto*. Son aquellas gramáticas cuyas producciones presentan el siguiente formato:

$$\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2 \; donde \; A \in \Sigma_A,\ \alpha_1, \alpha_2 \in \Sigma^*,\ \beta \in \Sigma^*.$$

Es decir, sólo se permite la sustitución del símbolo A por la cadena β cuando el símbolo A aparezca en el *contexto indicado* por la producción, es decir, con la cadena α_1 a la izquierda de A y por la cadena α_2 a su derecha. Nótese la diferencia con las gramáticas de tipo 2.

Gramáticas de tipo 0: Son aquellas gramáticas caracterizadas porque en sus producciones no se establece ningún tipo de restricción respecto a su formato.

Se dice que un lenguaje es de tipo *i* (*i* = 3, 2, 1, 0) si, y sólo si, la gramática de índice más alto que puede generarlo es de tipo *i*.

Además, se verifica que si se denomina L_i a la clase de lenguajes generados por gramáticas de tipo *i* entonces se observa que, de la definición de la jerarquía de gramáticas de Chomsky, se deriva el siguiente enunciado

$$L_3 \subseteq L_2 \subseteq L_1 \subseteq L_0.$$

Aún se puede afinar más y, en los temas sucesivos, se demostrará que estas inclusiones no son propias, es decir, existen lenguajes pertenecientes a la clase L_i que no pertenecen a la clase L_{i+1} (*i* = 0,1,2).

Por lo tanto, se cumple el siguiente enunciado

$$L_3 \subset L_2 \subset L_1 \subset L_0,$$

lo que se podría representar de forma gráfica mediante la siguiente figura:

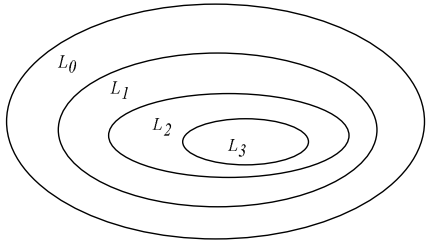


Figura 1.1: Relaciones entre las clases de lenguajes derivadas de la jerarquía de Chomsky.

La jerarquía establecida por Chomsky, además de resultar elegante, permite vertebrar la Teoría de la Computación clasificando las clases de lenguajes en función del número de recursos computacionales necesarios para reconocerlos.

Uno de los objetivos del estudio de los lenguajes formales es asimilar la correspondencia entre lenguajes formales y problemas computacionales, esto es, cualquier lenguaje formal se puede asociar a un problema computacional. Desde este punto de vista, reconocer las cadenas de un lenguaje es equivalente a solucionar un problema. Por lo tanto, la jerarquía de Chomsky realmente lo que permite es clasificar por orden creciente el poder computacional necesario para poder resolver distintas clases de problemas. Y uno de los resultados más espectaculares de esta disciplina lo supone el hecho de que, hoy por hoy, existen problemas que no se pueden resolver aún cuando se utilice el modelo con mayor poder computacional, el correspondiente a los lenguajes de Tipo 0, que modela la capacidad computacional de

cualquier computador existente. Esto es, hay problemas *irresolubles*, hay límites para el concepto de computación tal y como lo entendemos.

También resulta interesante comentar cómo ha ido tomando cuerpo esta teoría ya que, en principio, los principales resultados surgieron de campos muy diversos. Del campo de la Lógica Matemática surgió el modelo de Máquina de Turing, que ocupa el lugar más alto de la jerarquía, ya que es el reconocedor de los lenguajes de Tipo 0. Del campo de la Automática Industrial, surgieron las Autómatas de Control Finito, que ocupan a su vez el lugar más bajo, al corresponderse con los lenguajes de Tipo 3. De avances propios de la Informática, como el desarrollo de los procesadores de lenguaje, surgieron los reconocedores de lenguajes de Tipo 1.

El propio Noam Chomsky no es informático, ni matemático, ni ingeniero. Como ya se ha dicho es un lingüista cuya principal aportación fue el estudio de las gramáticas y su clasificación como parte de la demostración de su teoría *generativa* de las gramáticas [1].

Sin embargo, todos estos elementos tan diversos en su origen, estructuran el modelo existente de la Teoría de la Computación. Los capítulos de este libro se han estructurado de acuerdo a esta jerarquía, estudiando los niveles más trascendentales para nuestros objetivos; así, en la asignatura se verán los diferentes modelos abstractos en tres grandes grupos, correspondientes a los niveles 3, 2 y 0.

1.5. Problemas Propuestos.

1. Sean $X = \{x \mid x \in \mathcal{N} \wedge x\ es\ impar\}$, $Y = \{y \mid y \in \mathcal{N} \wedge y\ es\ primo\}$ y $Z = \{z \mid z \in \mathcal{N} \wedge z\ es\ múltiplo\ de\ tres\}$. Describir cada uno de los siguientes conjuntos:

   $a)\ X \cap Y$ $b)\ X \cap Z$ $c)\ Y \cap Z$ $d)\ Z - Y$
   $e)\ X - Y$ $f)\ X - (Y \cap Z)$ $g)\ (Y \cap Z) - X$ $h)\ (X \cap Y) \cap Z$
   $i)\ X \cup Y$ $j)\ X \cup (Y \cap Z)$

2. Sea X = {a, b, c} . Describir *P(X)* y *P(P(X))*.

3. Un palíndromo puede definirse como una cadena que se lee igual de derecha a izquierda que de derecha a izquierda, o por medio de la siguiente definición:

   a) λ es un palíndromo.

   b) Si *a* es un símbolo, entonces *a* es un palíndromo.

   c) Si *a* es un símbolo, y *x* es un palíndromo, también lo es *axa*.

[1] La teoría de Chomsky se basa en la existencia en el ser humano de estructuras gramaticales propias, esto es, el ser humano nace con la capacidad de aprender un lenguaje porque los esquemas gramáticales forman parte de su cerebro, de tal forma que, a medida que su experiencia le permite mejorar ese conocimiento, adquiere completamente la capacidad de comunicarse en un idioma. Esta teoría se contrapone a las que asumen que el cerebro humano es una "tabla en blanco" al nacer y que sólo mediante su interacción con el entorno adquiere el lenguaje.

   d) Cualquier cadena que no pueda ser formada mediante la aplicación de los pasos (a) a (c) anteriores un número finito de veces no es un palíndromo.

   Demostrar por inducción que las dos definiciones anteriores son equivalentes.

4. Escribir gramáticas que reconozcan cada uno de los siguientes lenguajes, sobre el alfabeto {a,b,c}:

   a) Cadenas que tengan igual número de a's que de b's.

   b) Cadenas de la forma $a^n b^n$, con $n \geq 0$.

   c) Cadenas de la forma $a^* b^*$.

   d) Cadenas de la forma $a^n b^n c^n$, con $n \geq 0$.

5. Sea $\Sigma = \{a, b\}$. La siguiente es una definición recursiva para el lenguaje L.

   - $\lambda \in L$,
   - Si $x \in L$ también $axb \in L$ y $bxa \in L$,
   - Si $x \in L$ e $y \in L$, entonces $xy \in L$,
   - Nada más pertenece a L.

   a) ¿Qué lenguaje es L?

   b) Escribir una gramática que genere L. ¿A qué tipo pertenece?

   c) Dar una definición recursiva para $L_1 \subseteq \{a, b\}^*$, consistente de todas las palabras con doble número de a's que de b's.

6. Describir los lenguajes generados por las siguientes gramáticas:

   $(a)\ S \to 0S1 \mid 01$

   $(b)\ S \to 000S111 \mid 000111$

   $(c)\ S \to aS \mid A$
   $A \to bA \mid \lambda$

   $(d)\ S \to 000S111$
   $0S1 \to 01$

   $(e)\ S \to X \mid Y$
   $X \to 0B1Y$
   $Y \to aY \mid A$
   $B \to 0B1 \mid \lambda$
   $A \to bA \mid \lambda$

7. Escribir una gramática, utilizando el alfabeto $\{+, -, *, /, (,), \texttt{digito}\}$, que genere cadenas que representen cualquier expresión aritmética válida.

Capítulo 2

Autómatas Finitos y Expresiones Regulares

Índice General

2.1. Autómatas Finitos Deterministas.

En esta asignatura se estudiarán distintos tipos de máquinas abstractas con diferente poder computacional. Cada uno de estos tipos de máquinas permiten reconocer lenguajes de distintos tipos y su poder computacional está asociado a cuál es el tipo de lenguajes que pueden reconocer.

El primer tipo de máquinas abstractas que se va a definir son los Autómatas Finitos, que permiten reconocer cadenas pertenecientes a Lenguajes Regulares.

Son los autómatas con menor poder computacional y para reconocer los lenguajes sólo disponen de una cinta de entrada (en la que se escribe una cadena), de un cabezal lector (que lee símbolos de uno en uno y de izquierda a derecha) y de un conjunto finito de reglas

que especifica cómo actuar ante cada símbolo leído de la cinta de entrada. Gráficamente se puede representar por medio de la siguiente figura:

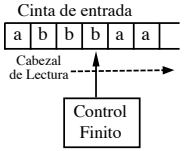


Figura 2.1: Modelo físico de un Autómata Finito.

Definición 2.1 *Un autómata finito determinista, AFD, es una quíntupla* $A = \langle \Sigma, Q, f, q_0, F \rangle$ *donde:*

Σ *es el alfabeto de entrada,*

Q *es un conjunto finito y no vacío de estados,*

f *es una función total definida como*

$$f : Q \times \Sigma \rightarrow Q,$$

q_0 *es el denominado estado inicial,* $q_0 \in Q$,

F *es el conjunto de estados denominados finales,* $F \subseteq Q$.

La forma más generalizada de representación de un AFD es mediante un grafo dirigido. En este grafo los nodos se corresponden con los estados de Q y los arcos representan valores de la función de transición *f*, de forma que si $f(q_s, a) = q_t$ entonces desde el nodo q_s parte un arco etiquetado con el símbolo a hacia el nodo q_t. Los estados de F se representan mediante nodos con doble círculo y el estado inicial se suele representar con una flecha de entrada al nodo que lo representa.

Ejemplo:

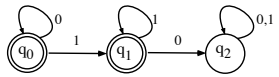
Sea A el siguiente AFD

$$Q = \{q_0, q_1, q_2\} \quad F = \{q_0, q_1\}$$
$$\Sigma = \{0, 1\} \quad q_0 \in Q, \ es\ el\ estado\ inicial$$

en el que la función de transición f se define como:

f	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_2

Este AFD se puede representar mediante el siguiente grafo:



Definición 2.2 *Se define el lenguaje aceptado por un AFD $A = \langle \Sigma, Q, f, q_0, F \rangle$ como el conjunto de cadenas tales que después de ser analizadas el AFD se encuentra en un estado final; es decir,*

$$L(A) = \{x \in \Sigma^* \mid f(q_0, x) \in F\}$$

Ejemplo:

¿Qué lenguajes reconocen los AFD de la figura 2.2?

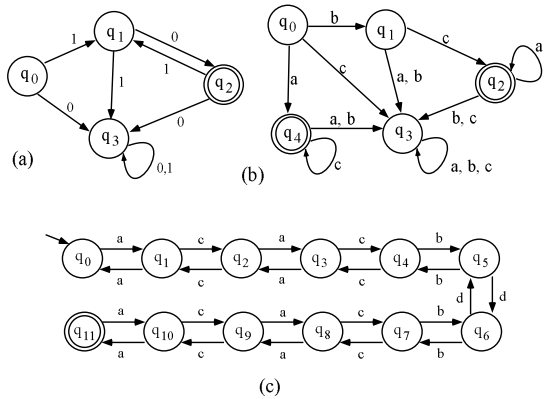


Figura 2.2: ¿Qué lenguajes reconocen estos AFD?

EL AFD del caso (a) reconoce las cadenas que tienen el formato $(10)^n$ con $n \geq 1$, es decir $L_a = \{10, 1010, 101010, 10101010, \ldots\}$.

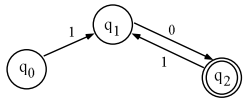
EL AFD del caso (b) acepta las cadenas que presentan el formato bca^n y las cadenas del formato ac^m, con $n, m \geq 0$.

El AFD del caso (c) resulta más difícil de describir de forma general. Algunas de las cadenas que acepta son

$$L_c = \{acacbdbcaca, aaacaaacccbdddddbcaaaca, \ldots\}.$$

Aquellos estados que no sean finales y que, además, una vez que son alcanzados el AFD permanece en ellos, sean cuales sean los símbolos que se lean de la entrada, se denominan *estados sumideros*. Como convenio no suelen mostrarse en el grafo para así poder minimizar el número de arcos del grafo y simplificar su representación gráfica.

Volviendo al ejemplo anterior, en el AFD (a), el estado q_3 es un estado sumidero; por lo tanto, este AFD puede representarse de la forma:



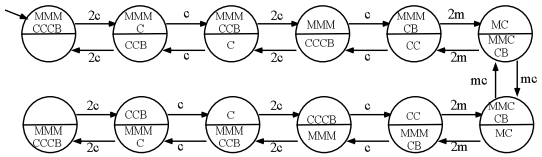
En el AFD del caso (b), el estado q_3 también es un sumidero y el nodo correspondiente y los arcos que se dirigen a él podrían eliminarse.

Para finalizar, se presenta como ejemplo un conocido problema de planificación y cómo resolverlo utilizando un AFD:

Tres caníbales y tres misioneros han de cruzar un río. Sólo disponen de una barca en la que caben como máximo dos personas. ¿Cómo han de cruzar el río de forma que nunca haya más caníbales que misioneros en cualquiera de las orillas?

Para solucionar el problema se identifica cada una de las posibles situaciones viables en las que se pueden encontrar caníbales y misioneros como un estado. En cada estado se representará los caníbales y misioneros que hay en cada orilla, así como la posición de la barca. Así, para indicar que hay 3 misioneros y 1 caníbal en una orilla (`MMMC`*) y 2 caníbales y la barca en la otra orilla (*`CCB`*), se emplea la siguiente notación:*

Para indicar quién ocupa la barca (transiciones entre estados), se emplea la siguiente notación: c *para caníbal,* 2m *dos misioneros,* 2c *dos caníbales y* mc *un misionero y un caníbal. Con esta notación, el conjunto de infinitas soluciones que tiene este problema se puede representar como cualquiera de los caminos que conducen de la situación inicial a la final en el siguiente AFD:*



El AFD diseñado para el problema de los misioneros y los caníbales se puede relacionar con el mostrado en el apartado (c) del ejemplo anterior; el lenguaje L_c se puede asociar a las soluciones de este problema bajo la interpretación *a* por *2c*, *c* por *c*, *b* por *2m* y *d* por *mc*. Por lo tanto, realizando una interpretación adecuada a los símbolos y los estados del AFD se ha podido establecer el lenguaje aceptado por este AFD. En general, el problema de encontrar una interpretación adecuada no es sencillo de solucionar y plantea una serie de cuestiones de orden filosófico/matemático que queda fuera del alcance de este tema.

2.2. Autómatas Finitos No Deterministas.

Un *Autómata Finito No Determinista*, que se denotará como AFN, se caracteriza porque en un estado puede haber más de una transición posible para un mismo símbolo de entrada. Es decir, $\mid f(q,a) \mid \geq 1$ para algún $q \in Q$ y para algún símbolo $a \in \Sigma$.

Definición 2.3 *Un AFN es una quíntupla* $A = \langle \Sigma, Q, f, q_0, F \rangle$ *donde:*

Σ *es el alfabeto de entrada,*

Q *es un conjunto finito y no vacío de estados,*

f *es una función definida de la siguiente forma*

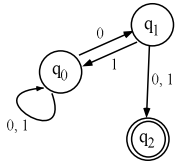
$$f : Q \times \Sigma \to 2^Q,$$

q_0 *es el denominado estado inicial,* $q_0 \in Q$,

F *es el conjunto de estados denominados finales,* $F \subseteq Q$.

Ejemplo:

El siguiente grafo representa un AFN ya que $f(q_0, 0) = \{q_0, q_1\}$ *y* $f(q_1, 1) = \{q_0, q_2\}$*:*



El no determinismo afecta a la forma de entender el comportamiento del AF con una cadena. Si se tiene que $f(q_0, 0) = \{q_0, q_1\}$*, entonces* $f(q_0, 01) = f(f(q_0, 0), 1) = f(\{q_0, q_1\}, 1) = \{q_0, q_2\}$*. Es decir, en general, lo que se obtiene es un conjunto de estados, tanto si se analiza el comportamiento con un símbolo, como si se generaliza para una cadena. Esto afectará a la definición de lenguaje aceptado por un AFN.*

Definición 2.4 *Se define el lenguaje aceptado por un AFN* $A = \langle \Sigma, Q, f, q_0, F \rangle$ *como el conjunto de cadenas tales que después de ser analizadas, el AFN ha podido alcanzar al menos un estado final, es decir,*

$$L(A) = \{x \in \Sigma^* \mid f(q_0, x) \cap F \neq \emptyset\}.$$

En el autómata del ejemplo anterior, L(A) es el conjunto de cadenas cuyo penúltimo símbolo leído de izquierda a derecha es un 0.

Un sencillo análisis de este tipo de autómata permite establecer que si en un autómata de este tipo no hay ninguna transición tal que para un mismo símbolo se disponga de más de un estado destino diferente, entonces este autómata es determinista. Es decir, los AFD se pueden ver como casos particulares de los AFN. Entonces, si se denota por *L(AFD)* al conjunto de todos los lenguajes que se pueden reconocer con autómatas finitos deterministas y se denota por *L(AFN)* al conjunto de todos los lenguajes que se pueden reconocer con autómatas finitos no deterministas, resulta que $L(AFD) \subseteq L(AFN)$.

La cuestión que se plantea es si la inclusión contraria también se cumple, es decir, si para cualquier AFN existe un AFD que reconozca exactamente el mismo lenguaje que el AFN. El siguiente teorema establece que así es, lo que permite concluir que

$$L(AFD) = L(AFN).$$

Teorema 2.1 *Sea L un lenguaje aceptado por un AFN, entonces existe un AFD que también acepta L.*

La demostración de este teorema no se verá, pero la operación de calcular un AFD a partir de un AFN es muy común y, por ello, se han desarrollado varios métodos. El más sencillo y eficiente es el siguiente:

Mínimo número de pasos para construir un AFD a partir de un AFN

1. Se construye una tabla donde cada columna está etiquetada con un símbolo del alfabeto de entrada y cada fila se etiqueta con un conjunto de estados.
2. La primera fila se etiqueta con $\{q_0\}$, estado inicial, y en cada entrada de la tabla (q_0, s_i) se almacena $f(q_0, s_i) = \{p_1, \ldots, p_n\} = P$.
3. Se etiqueta cada fila con cada uno de los conjuntos P que no tengan asociada una fila en la tabla (es decir, con cada uno de los conjuntos P que aparezcan por primera vez en la tabla) y se completa cada una de estas filas con el correspondiente $f(P, s_i)$.
4. Se realiza el paso (3) hasta que no haya en la tabla conjuntos P sin filas asociadas.
5. Se asocia a cada conjunto P que aparezca en la tabla un estado en el nuevo AFD y aquellos que tengan entre sus componentes algún estado final del AFN se considerarán estados finales en el AFD.

Se va a aplicar este método al autómata anterior,

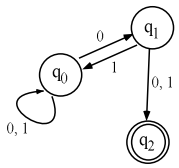


Figura 2.3: AFN a transformar en AFD.

La función de transición de este AFN es la siguiente:

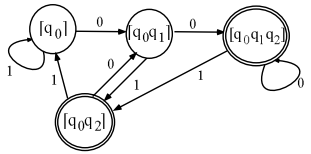
f	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_0, q_2\}$
q_2	$\emptyset$	$\emptyset$

Al aplicar el método anterior al AFN se obtiene el siguiente AFD, cuya función de transición es:

f	0	1
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

Al conjunto $\{q_0\}$ se le denomina $[q_0]$, al $\{q_0, q_1\}$ se le denomina $[q_0, q_1]$, al $\{q_0, q_1, q_2\}$ se le denomina $[q_0, q_1, q_2]$ y a $\{q_0, q_2\}$ se le denomina $[q_0, q_2]$.

Como los conjuntos $\{q_0, q_1, q_2\}$ y $\{q_0, q_2\}$ contienen estados finales del AFN entonces los estados $[q_0, q_1, q_2]$ y $[q_0, q_2]$ serán estados finales en el AFD.



2.3. Autómatas Finitos con Movimientos Libres.

Un Autómata Finito con Movimientos Libres, que se denotará habitualmente como AFλ, se caracteriza porque cada transición está etiquetada con uno o más símbolos del alfabeto de entrada o bien con la cadena vacía.

Definición 2.5 *Un autómata finito con movimientos libres, AFλ, es una quíntupla $A = \langle \Sigma, Q, f, q_0, F \rangle$ donde:*

Σ *es el alfabeto de entrada,*

Q *es un conjunto finito y no vacío de estados,*

f *es una función definida de la siguiente forma*

$$f : Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q,$$

q_0 *es el denominado estado inicial, $q_0 \in Q$,*

F *es el conjunto de estados denominados finales, $F \subseteq Q$.*

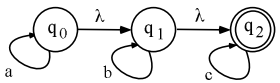
Para poder determinar correctamente cuál es el comportamiento de este tipo de autómata ante una cadena de entrada se necesita establecer una serie de definiciones previas.

Definición 2.6 *Se denomina $\lambda\_clausura(q),\ \forall\ q \in Q$, al conjunto de estados accesibles desde el estado* q *utilizando sólo transiciones λ. Además, se cumple que $q \in \lambda\_clausura(q)$.*

Definición 2.7 *Se denomina $\lambda\_CLAUSURA(P)$, siendo P un conjunto de estados, $P \in 2^Q$, al conjunto de estados formado por todas las λ_clausuras de todos los estados que forman P,*

$$\lambda\_CLAUSURA(P) = \bigcup_{p \in P} \lambda\_clausura(p).$$

Ejemplo:



$$\lambda\_clausura(q_0) = \{q_0, q_1, q_2\}$$
$$\lambda\_clausura(q_1) = \{q_1, q_2\}$$
$$\lambda\_clausura(q_2) = \{q_2\}$$

Para simplificar la notación se utiliza el mismo símbolo de función λ_clausura para referirse tanto a $\lambda\_clausura$ como a $\lambda\_CLAUSURA$.

Este concepto es muy importante porque al analizar el comportamiento del AFλ con una determinada cadena se han de tener en cuenta también las posibles transiciones sin consumir símbolo. La λ_clausura permite determinar a qué estados se puede transitar desde uno dado *sin consumir símbolo*; por lo tanto, es un paso previo al cálculo de la imagen. Y para poder determinar cuál es el lenguaje reconocido por este tipo de autómata es preciso conocer cuál es su comportamiento ante cadenas de símbolos teniendo en cuenta las transiciones λ. Para ello se define una extensión de la función de transición del autómata, a la que se denominará $\hat{f}$, $\hat{f} : Q \times \Sigma \longrightarrow 2^Q$, de forma que $\hat{f}(x)$, siendo x una cadena, se calcula analizando símbolo a símbolo el conjunto de estados alcanzables.

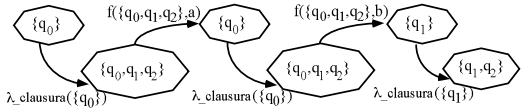
Ejemplo:

Si en el AFλ del ejemplo anterior se quiere calcular $\hat{f}(q_0, ab)$, para hacerlo correctamente hay que seguir el siguiente proceso:

1. *Calcular la $\lambda\_clausura(q_0) = \{q_0, q_1, q_2\}$, ya que se parte del estado q_0,*

2. *Calcular $f(\{q_0, q_1, q_2\}, a) = \{q_0\}$, ya que a es el primer símbolo de la cadena ab.*
3. *Calcular la $\lambda\_clausura$ del resultado del paso anterior; como, casualmente es q_0 ya sabemos que es $\{q_0, q_1, q_2\}$. Este es el primer resultado parcial, con el primer símbolo; para calcular $f(q_0, ab)$ hemos de calcular la imagen de este conjunto de estados con b,*
4. *Calcular $f(\{q_0, q_1, q_2\}, b) = \{q_1\}$,*
5. *Calcular la $\lambda\_clausura$ del resultado del paso anterior: $\lambda\_clausura(q_1) = \{q_1, q_2\}$.*

Todo este proceso se refleja en el siguiente esquema:



Definición 2.8 *Se define el lenguaje aceptado por un AFλ A, $A = \langle \Sigma, Q, f, q_0, F \rangle$, como el conjunto de cadenas tales que, después de ser analizadas, el AFλ ha podido alcanzar al menos un estado final, es decir,*

$$L(A) = \{x \in \Sigma^* \mid \hat{f}(q_0, x) \cap F \neq \emptyset\}$$

De la definición de un AFλ se sigue que un AFN es un caso particular de AFλ en el que no existen transiciones λ. Por lo tanto, si se denota por L(AFλ) al conjunto de todos los lenguajes que pueden ser reconocidos por AFλ, entonces $L(AFN) \subseteq L(AF\lambda)$.

La cuestión que se plantea a continuación es si la inclusión contraria también se cumple; es decir, si para cualquier AFλ existe un AFN que reconozca exactamente el mismo lenguaje que el AFλ.

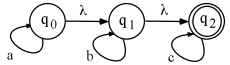
Teorema 2.2 *Sea L un lenguaje aceptado por un AFλ, entonces existe un AFN que también acepta L.*

Al igual que ocurría en la sección anterior, más que demostrar este teorema interesa estudiar un método eficiente para realizar el cálculo de un AFN equivalente a un AFλ dado. En la asignatura se utilizará el siguiente:

1. A partir de la tabla original, se construye una tabla donde cada columna está etiquetada con un símbolo del alfabeto de entrada, *sin tener ya en cuenta a* λ.
2. En esta nueva tabla, cada fila se etiqueta con uno de los estados del AFλ original.
3. Para dar un valor a cada una de las entradas $[q_i, s_j]$ de la tabla hay que realizar el cálculo de $\hat{f}(q_i, s_j)$:
   a) se calcula la λ-clausura de $\{q_i\}$, obteniéndose el conjunto de estados P,
   b) se calcula $f(P, s_j)$, imagen de P con el símbolo j, obteniéndose el conjunto de estados P′, y
   c) se calcula la λ-clausura de P′, y el conjunto de estados obtenido se almacena en la tabla
4. El conjunto de estados finales estará formado por lo mismos estados que lo formaban en el autómata original. Además, si la $\lambda - clausura(q_0)$ contiene algún estado final, q_0 también será un estado final.

   Nota: *si la* $\lambda - clausura(q_0)$ *contiene algún estado final,* q_0 *también será un estado final.*¿por qué? :-)

Ejemplo: Calcular un AFN que reconozca el mismo lenguaje que el siguiente AFλ



La figura 2.4 es un esquema gráfico del procedimiento seguido para rellenar la primera fila de la tabla anterior, la que corresponde a $\{q_0\}$*: se va calculando la imagen de* $\lambda - clausura(q_1)$ *con cada uno de los tres símbolos del alfabeto y, además, se vuelve a calcular la* λ*-clausura del conjunto así obtenido.*

Haciendo lo mismo para $\{q_1\}$ *y* $\{q_2\}$*, se obtiene la siguiente tabla:*

	a	b	c
$\{q_0\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$\{q_1\}$	$\emptyset$	$\{q_1, q_2\}$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$	$\{q_2\}$

que describe el comportamiento del AFN equivalente, que se muestra en la figura 2.5.

En la figura 2.5, además, se ve que $\{q_0\}$ *es estado final, además de* $\{q_2\}$*, porque originalmente* $F = \{q_2\}$ *y, además,* $\lambda\_clausura(q_0) \cap F \neq \emptyset$*.*

Como consecuencia de este teorema y del teorema 2.1, se obtiene el siguiente resultado:

$$\mathbf{L(AFD)} = \mathbf{L(AFN)} = \mathbf{L(AF}\lambda\mathbf{)}.$$

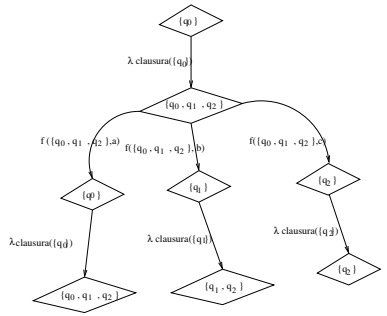


Figura 2.4: Esquema del cálculo de la primera fila de la tabla para el AFN.

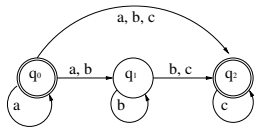
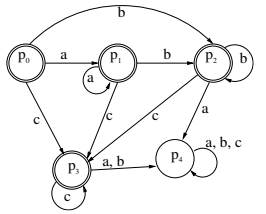


Figura 2.5: AFN equivalente obtenido.

Ejemplo

El AFD siguiente es equivalente a los AFs anteriores. (p_4 *es un estado sumidero):*



Para obtenerlo, basta con aplicar sobre el AFN representado en la figura 2.5 el método descrito en la sección 2.2:

f	a	b	c
$p_0 = \{q_0\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$p_1 = \{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$p_2 = \{q_1, q_2\}$	$\emptyset$	$\{q_1, q_2\}$	$\{q_2\}$
$p_3 = \{q_2\}$	$\emptyset$	$\emptyset$	$\{q_2\}$

2.4. Expresiones Regulares.

Las expresiones regulares permiten denotar lenguajes regulares y su estudio resulta de gran interés, tanto por su capacidad de especificación mediante un número reducido de operadores como por sus aplicaciones prácticas en la construcción de analizadores léxicos.

Definición 2.9 *Dado el alfabeto Σ, una expresión regular será el resultado de la aplicación de algunas (o todas) de las siguientes reglas un número finito de veces:*

1. *El símbolo $\emptyset$ es una expresión regular y denota el lenguaje vacío,*
2. *El símbolo λ es una expresión regular y denota el lenguaje $\{\lambda\}$,*
3. *Si* a $\in \Sigma$ *entonces* a *es una expresión regular que denota el lenguaje* $\{$a$\}$,
4. *Si α y β son expresiones regulares entonces*
   - a) *$\alpha + \beta$ es una expresión regular que denota la unión de los lenguajes denotados por α y por β.*
   - b) *$\alpha\beta$ es una expresión regular que denota la concatenación del lenguaje denotado por α con el denotado por β.*
   - c) *α^* es una expresión regular que denota la clausura del lenguaje denotado por α.*

Ejemplos:

Si $\alpha = a^*$ *entonces* $L(\alpha) = L(a^*) = \{a^i \mid i \geq 0\}$.

Si $\alpha = a^*$ *y* $\beta = b^*$ *entonces* $L(\alpha\beta) = L(a^*b^*) = \{a^i b^j \mid i, j \geq 0\}$.

Si $\alpha = a$ *entonces* $L(\alpha^*\alpha) = L(a^*a) = L(a^*)L(a) = \{a^i \mid i \geq 0\}\{a\} = \{a^i \mid i \geq 1\}$.

Si $\alpha = a$ *entonces* $L(\alpha\alpha^*) = L(aa^*) = L(a)L(a^*) = \{a\}\{a^i \mid i \geq 0\} = \{a^i \mid i \geq 1\}$. *Por lo tanto,* $L(a^*a) = L(aa^*)$.

El último ejemplo sirve para ilustrar la cuestión de qué se entiende por expresiones regulares equivalentes: α y β son dos expresiones regulares equivalentes si $L(\alpha) = L(\beta)$.

En general se va a denotar por α tanto a la expresión regular como al lenguaje que denota, y además, para simplificar su escritura se establece una jerarquía de operadores para evaluar una expresión regular: primero se realizan las clausuras, luego las concatenaciones y por último las uniones.

Así, por ejemplo, la expresión regular $a(a)^*$ se puede escribir aa^*, y $(\alpha\beta) + \gamma$ se puede escribir $\alpha\beta + \gamma$.

Teorema 2.3 (Propiedades de las Expresiones Regulares) *Las siguientes relaciones son ciertas:*

1) $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma = \alpha + \beta + \gamma$
2) $\alpha + \beta = \beta + \alpha$
3) $\alpha + \emptyset = \emptyset + \alpha = \alpha$
4) $\alpha(\beta\gamma) = (\alpha\beta)\gamma = \alpha\beta\gamma$
5) $\alpha\lambda = \lambda\alpha = \alpha$
6) $\alpha\beta + \alpha\gamma = \alpha(\beta + \gamma); \beta\alpha + \gamma\alpha = (\beta + \gamma)\alpha$
7) $\emptyset\alpha = \alpha\emptyset = \emptyset$
8) $\alpha + \alpha = \alpha; \alpha^* = \lambda + \alpha + \alpha^2 + \ldots + \alpha^n + \alpha^*$
9) $\lambda^* = \lambda$
10) $\emptyset^* = \lambda; \emptyset^+ = \emptyset$
11) $\alpha^*\alpha^* = \alpha^*$
12) $(\alpha^*)^* = \alpha^*$
13) $\alpha\alpha^* = \alpha^*\alpha = \alpha^+$
14) $\alpha^* = \lambda + \alpha + \alpha^2 + \ldots + \alpha^n + \alpha^{n+1}\alpha^*$
15) $\alpha^* = \lambda + \alpha\alpha^* = \lambda + \alpha^+$
16) $\alpha^* = (\lambda + \alpha)^{n-1}(\alpha^n)^*$
17) $(\alpha + \beta)^* = (\alpha^* + \beta^*)^* = (\alpha^*\beta^*)^*$
18) $(\alpha + \lambda)^* = \alpha^* + \lambda = \alpha^*$
19) $(\alpha\beta)^*\alpha = \alpha(\beta\alpha)^*$
20) $(\alpha^*\beta)^*\alpha^* = (\alpha + \beta)^*$
21) $(\alpha^*\beta)^* = (\alpha + \beta)^*\beta + \lambda$
22) $(\beta\alpha^*)^* = \beta(\alpha + \beta)^* + \lambda$

En las propiedades anteriores el signo = debe interpretarse como equivalencia; es decir, al expresar que $\alpha + \beta = \beta + \alpha$ se afirma que el lenguaje denotado por la expresión regular $\alpha + \beta$ es equivalente al denotado por la expresión regular $\beta + \alpha$.

Teorema 2.4 (de Análisis) *Sea L un lenguaje aceptado por un AFD, entonces existe una expresión regular que lo denota.*

En la asignatura se utilizará el siguiente método para calcular la expresión regular que denota a un determinado AFD. Se denomina **Regla de Arden**. Para aplicarla, hay que esta-

blecer un sistema de ecuaciones lineales en expresiones regulares y resolverlo. Cada ecuación de un sistema de ecuaciones lineales en expresiones regulares tiene la siguiente forma general

$$X = rX + s$$

en la que r y s son expresiones regulares sobre un alfabeto Σ. Dependiendo de que $\lambda \in r$ o $\lambda \notin r$ se tienen dos soluciones para dicha ecuación:

- $\lambda \notin$ **r** y, entonces, $X = r^*s$. Para comprobarlo, basta ver que

$$X = rX + s = r(r^*s) + s = (rr^* + \lambda)s = r^*s = X.$$

- $\lambda \in$ **r** y, entonces, hay infinitas soluciones $X = r^*(s + t)$, donde t es una expresión regular cualquiera sobre Σ (normalmente, este caso no nos afectará). Este resultado puede comprobarse de la siguiente forma,

$$X = rX + s = r(r^*(s + t)) + s = (rr^* + \lambda)s + rr^*t = (r^+ + \lambda)s + r^+t.$$

Como $r^* = r^+$ si $\lambda \in r$, entonces lo anterior queda,

$$(r^+ + \lambda)s + r^+t = (r^* + \lambda)s + r^*t = r^*(s + t) = X.$$

Método para obtener la expresión regular que denota a un AF dado. El método se basa en el establecimiento de un sistema de ecuaciones lineales en expresiones regulares a partir del AF. Sea A,

$$A = \langle \Sigma, Q, f, q_0, F \rangle$$

un AFD o un AFN; a A se le puede asociar un sistema de ecuaciones lineales en expresiones regulares de la siguiente forma:

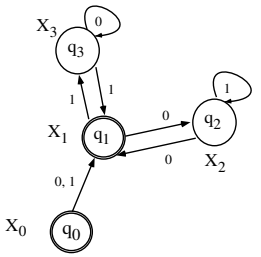
1. Se asocia una variable a cada estado: $\forall q_i \in Q$, a q_i se le asocia X_i.
2. Las ecuaciones se construyen en función de las transiciones: si $q_j \in f(q_i, a)$, entonces en la ecuación de la variable X_i aparece el término aX_j en su parte derecha:

$$X_i = ... + aX_j + ...$$

3. Además, se asocia el término λ a los estados finales: si $q_i \in F$, entonces en la ecuación de la variable X_i aparece λ en su parte derecha: $X_i = ... + \lambda + ...$

El lenguaje reconocido por el AF es la *expresión regular de la variable asociada a su estado inicial*.

Ejemplo Hay que calcular la expresión regular que denota el lenguaje reconocido por el autómata que se muestra en la figura 2.4.



Sobre la figura se ha asociado ya a cada estado su correspondiente variable. De acuerdo al segundo y tercer paso del método, el sistema de ecuaciones lineales que hay que resolver es el siguiente:

$$\begin{aligned} X_0 &= 0X_1 + 1X_1 + \lambda = (0 + 1)X_1 + \lambda \\ X_1 &= 0X_2 + 1X_3 + \lambda \\ X_2 &= 1X_2 + 0X_1 \\ X_3 &= 0X_3 + 1X_1 \end{aligned}$$

Recordemos que la solución de la ecuación general, $X = rX + s$ es $X = r^*s$. En la última ecuación, si identificamos términos con r y s se tiene:

$$X_3 = \underbrace{0}_{r} X_3 + \underbrace{1X_1}_{s}.$$

Por lo tanto,

$$X_3 = 0^*1X_1.$$

De forma similar, se puede operar con X_2,

$$X_2 = \underbrace{1}_{r} X_2 + \underbrace{0X_1}_{s}, \Rightarrow X_2 = 1^*0X_1.$$

Se tienen X_2 y X_3 en función de X_1. Al substituir en la ecuación de X_1 se obtiene

$$X_1 = 0X_2 + 1X_3 + \lambda = 01^*0X_1 + 10^*1X_1 + \lambda = (01^*0 + 10^*1)X_1 + \lambda$$

Agrupando términos se llega a

$$X_1 = \underbrace{(01^*0 + 10^*1)}_{r} X_1 + \underbrace{\lambda}_{s}, \Rightarrow X_1 = (01^*0 + 10^*1)^*\lambda = (01^*0 + 10^*1)^*.$$

y, al substituir en la ecuación de X_0 se obtiene finalmente,

$$\begin{aligned} X_0 &= (0 + 1)X_1 + \lambda = (0 + 1)(01^*0 + 10^*1)^* + \lambda, \\ &\Rightarrow L(AFD) = (0 + 1)(01^*0 + 10^*1)^* + \lambda. \end{aligned}$$

Teorema 2.5 (de Síntesis) *Sea r una expresión regular, entonces existe un AFλ A tal que $L(A) = r$.*

Demostración:

Hay que demostrar que para cualquier expresión regular *r* siempre se puede construir un AFλ que sea capaz de reconocer lenguaje que denota *r*. Una forma de calcular el AFλ adecuado a cada expresión regular consiste en aplicar inducción sobre el número de operadores presentes en la expresión regular.

Para simplificar el proceso de demostración se asume que todos los AFλ contienen un único estado final que no posee transiciones de salida, es decir, $F = \{q_f\}$ *y* $f(q_f, a) = \emptyset\ \forall\ a \in \Sigma$.

Paso Base: En *r* no hay operadores regulares. Entonces *r* sólo puede ser $\emptyset$, λ, $a \in \Sigma$, a los que se puede asociar, respectivamente, los AFλ que se muestran en al figura 2.6.

Figura 2.6: AFλs asociados a $\emptyset$, λ y $a \in \Sigma$.

Hipótesis de Inducción: Supóngase que para cualquier expresión regular con un número de operadores menor o igual que *n* se cumple que existe un AFλ que la reconoce.

Paso de Inducción: Sea *r* con *n+1* operadores, entonces *r* se puede escribir como $r = r_1 + r_2$, $r = r_1 r_2$ o $r = r_1^*$, donde tanto r_1 como r_2 son expresiones regulares que contienen *n* o menos operadores regulares.

Se estudiará cada una de las tres posibles situaciones:

$\mathbf{r = r_1 + r_2}$**:** Como r_1 y r_2 tienen *n* o menos operadores regulares existen, por H.I., dos AFλ A_1 y A_2 tales que $L(A_1) = r_1$ y $L(A_2) = r_2$. Estos autómatas aparecen esquematizados en la figura 2.7. :

Figura 2.7: $L(A_1) = r_1$ y $L(A_2) = r_2$.

El AFλ de la figura 2.8 reconoce $L(A_1) \cup L(A_2)$, por lo tanto $r = r_1 + r_2$.
La definición formal de este autómata es la siguiente:

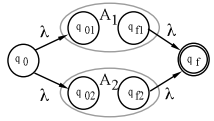


Figura 2.8: AFλ A tal que $L(A) = r_1 + r_2$.

Sea $A_1 = \langle \Sigma_1, Q_1, f_1, q_{01}, \{q_{f1}\}\rangle$ y $A_2 = \langle \Sigma_2, Q_2, f_2, q_{02}, \{q_{f2}\}\rangle$, tales que $Q_1 \cap Q_2 = \emptyset$.
Se define el AFλ

$$A = \langle \Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2 \cup \{q_0, q_f\}, f, q_0, \{q_f\}\rangle$$

en el que la función de transición *f* se define como:

$$\begin{array}{l} f(q_0, \lambda) = \{q_{01}, q_{02}\} \\ f(q, a) = f_1(q, a), \forall\ q \in Q_1, \forall\ a \in \Sigma_1 \cup \{\lambda\} \\ f(q, a) = f_2(q, a), \forall\ q \in Q_2, \forall\ a \in \Sigma_2 \cup \{\lambda\} \\ f(q_{f1}, \lambda) = f(q_{f2}, \lambda) = \{q_f\} \end{array}$$

$\mathbf{r = r_1 r_2}$**:** Como r_1 y r_2 tienen *n* o menos operadores regulares existen, por H.I., dos AFλ A_1 y A_2 tales que $L(A_1) = r_1$ y $L(A_2) = r_2$. Estos autómatas aparecen esquematizados en la figura 2.9.

Figura 2.9: $L(A_1) = r_1$ y $L(A_2) = r_2$.

Si se permite la transición del estado final final de A_1 al inicial del autómata A_2, se obtiene el AFλ de la figura 2.10, que reconoce $L(A_1)L(A_2)$ y, por lo tanto, $r = r_1 r_2$.

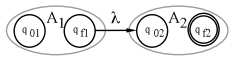


Figura 2.10: AFλ A tal que $L(A) = r_1 r_2$.

Para definir formalmente este autómata, se parte de la definición de los autómatas $A_1 = \langle \Sigma_1, Q_1, f_1, q_{01}, \{q_{f1}\}\rangle$ y $A_2 = \langle \Sigma_2, Q_2, f_2, q_{02}, \{q_{f2}\}\rangle$ tal que $Q_1 \cap Q_2 = \emptyset$. Entonces, se define el AFλ

$$A = \langle \Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2, f, q_{01}, \{q_{f2}\}\rangle$$

en el que la función de transición f se define como:

$$\begin{array}{l} f(q, a) = f_1(q, a), \forall\ q \in Q_1 - \{q_{f1}\}, \forall\ a \in \Sigma_1 \cup \{\lambda\} \\ f(q_{f1}, \lambda) = \{q_{02}\} \\ f(q, a) = f_2(q, a), \forall\ q \in Q_2, \forall\ a \in \Sigma_2 \cup \{\lambda\} \end{array}$$

$\mathbf{r = r_1^*}$: Como r_1 tiene n o menos operadores regulares existe, por H.I., un AFλ A_1 tal que $L(A_1) = r_1$:

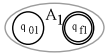


Figura 2.11: AFλ A_1 tal que $L(A_1) = r_1$.

El AFλ de la figura 2.12, reconoce $[L(A_1)]^*$, por lo tanto $r = r_1^*$.

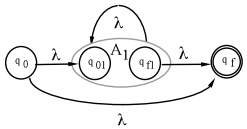


Figura 2.12: AFλ A tal que $L(A) = r_1^*$.

En este caso, para la definición formal, se parte del autómata $A_1 = \langle \Sigma_1, Q_1, f_1, q_{01}, \{q_{f1}\}\rangle$. Y se define el AF$\lambda$

$$A = \langle \Sigma_1, Q_1 \cup \{q_0, q_f\}, f, q_0, \{q_f\}\rangle$$

en el que la función de transición f se define como:

$$f(q_0, \lambda) = f(q_{f1}, \lambda) = \{q_{01}, q_f\}$$
$$f(q, a) = f_1(q, a), \forall\, q \in Q_1 - \{q_{f1}\}, \forall\, a \in \Sigma_1 \cup \{\lambda\}$$

c.q.d.

2.5. Otros Tipos de Autómatas Finitos.

Además de las extensiones vistas (no deterministas, deterministas, movimientos libres, expresiones regulares) existen otra serie de modelos de máquinas virtuales con una capacidad de cálculo equivalente a los autómatas finitos. De entre estos autómatas se van a exponer las principales características de los autómatas con función de salida (Mealy y Moore).

2.5.1. Autómatas Finitos con Función de Salida.

Hasta el momento los AF descritos tienen como salida una señal del tipo acepto/no acepto (según se alcance un estado final o no después de consumir toda la cadena de entrada).

Los autómatas finitos con función de salida son modelos que traducen la cadena de entrada en otra cadena de salida. Si esa traducción se realiza en cada estado entonces se

tiene una máquina de Moore y si la traducción se realiza en cada transición entonces se tiene una máquina de Mealy.

Máquina de Moore.

Es una séxtupla $A = \langle Q, \Sigma, \Delta, f, \gamma, q_0\rangle$ donde Q, Σ, f y q_0 se definen como en un AFD. El conjunto Δ representa al alfabeto de salida y γ es una función que asocia a cada estado un símbolo del alfabeto de salida, es decir, $\gamma : Q \to \Delta$.

La respuesta de A con respecto a la entrada $a_1 a_2 \ldots a_n, n \geq 0$, es

$$\gamma(q_0)\gamma(q_{i1})\gamma(q_{i2})\ldots\gamma(q_{in})$$

donde $q_0, q_{i1}, q_{i2}, \ldots, q_{in}$ es la secuencia de estados tal que $f(q_{ij}, a_{i+1}) = q_{ij+1}$.

Ejemplo:

Escribir una máquina de Moore que determine el resto de dividir un número binario entre 3.

El resto de dividir un número entre 3 será el entero 0, el 1 o el 2. Por lo tanto, se establece una Máquina de Moore con tres estados, q_0, q_1 y q_2, de forma que $\gamma(q_0) = 0$, $\gamma(q_1) = 1$ y $\gamma(q_2) = 2$.

Para determinar las transiciones entre estados se realiza el siguiente análisis:

- *Si a un número binario* i *se le añade un* 0 *por la derecha se obtiene un nuevo entero,* i0*, cuyo valor es el de* i *multiplicado por* 2.
  Por lo tanto, si $i \bmod 3 = r$, entonces $2i \bmod 3 = 2r \bmod 3 = r'$. Cuando se parte de un valor $r = 0$, se obtiene para r' el valor 0; si $r = 1 \Rightarrow r' = 2$ y si $r = 2 \Rightarrow r' = 1$.
- *Si al entero* i *se le añade un* 1 *se obtiene como nuevo valor,* i1*, el anterior multiplicado por* 2 *y añadiéndole* 1.
  Por lo tanto, si $i \bmod 3 = r$, entonces $(2i + 1) \bmod 3 = (2r + 1) \bmod 3 = r'$. Entonces, si $r = 0 \Rightarrow r' = 1$, si $r = 1 \Rightarrow r' = 0$ y si $r = 2 \Rightarrow r' = 2$.

Este análisis permite diseñar la Máquina de Moore de la figura 2.13.

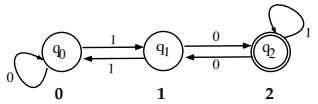


Figura 2.13: Máquina de Moore para calcular el resto de la división entera entre 3.

El comportamiento de esta Máquina de Moore ante la cadena de entrada `110100` *es la cadena de salida* $\gamma(q_0)\gamma(q_1)\gamma(q_0)\gamma(q_0)\gamma(q_1)\gamma(q_2)\gamma(q_1)$*, es decir,* `0100121`.

Al interpretar este resultado se obtiene que `110100` *es el entero 52, y* $52 \bmod 3 = 1$*, que coincide con el último símbolo de salida proporcionado por la máquina de Moore.*

Máquina de Mealy.

Es una séxtupla $A = \langle Q, \Sigma, \Delta, f, \gamma, q_0 \rangle$ donde Q, Σ, f y q_0 se definen como en un AFD. El conjunto Δ representa al alfabeto de salida y γ es una función que asocia a cada transición un símbolo del alfabeto de salida, es decir, $\gamma : Q \times \Sigma \to \Delta$.

La respuesta de A con respecto a la entrada $a_1 a_2 \dots a_n, n \geq 0$, es

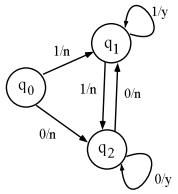
$$\gamma(q_0, a_1)\gamma(q_{i1}, a_2)\gamma(q_{i2}, a_3) \dots \gamma(q_{in-1}, a_n)$$

donde $q_0, q_{i1}, q_{i2}, \dots, q_{in-1}$ es la secuencia de estados tal que $f(q_{ij}, a_{i+1}) = q_{ij+1}$.

Ejemplo:

Escribir una máquina de Mealy que produzca salida 'y' *cuando los dos últimos símbolos leídos de la cadena sean iguales y que produzca salida* 'n' *en otro caso.*

Para realizar esta máquina basta con recordar cual ha sido el último símbolo leído de la cadena de entrada, y si su siguiente símbolo coincide con el anterior, entonces se produce salida 'y'*; en cualquier otra situación la salida será* 'n'.



2.6. Aplicaciones.

Los lenguajes regulares son utilizados en aplicaciones muy diversas. Entre ellas, en esta sección se comentan tres de las más usuales,

- análisis léxico de un compilador (exp. regulares, AF),
- especificación de parámetros de entrada (expresiones regulares),
- ayuda al diseño de circuitos (Mealy, Moore).

2.6.1. Aplicación como ayuda al análisis léxico en un compilador.

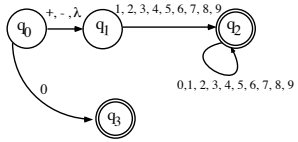
Un compilador es un programa que traduce un código denominado *fuente* en otro código denominado *objeto*. Para realizar esa transformación son necesarias varias etapas. La primera de estas etapas consiste en detectar, de todas las cadenas que componen el código fuente, cuáles son representativas para el compilador. Por ejemplo, qué cadenas representan *constantes* (enteras, reales, lógicas, etc) o bien *palabras reservadas* del lenguaje (`begin`, `end`, {, }, `procedure`, `function`, `main`, `while`, ... etc.). Estas cadenas se representan por medio de expresiones regulares y su detección se realiza por medio de autómatas finitos.

El siguiente ejemplo muestra cómo representar constantes enteras, con o sin signo, utilizando expresiones regulares y cómo reconocer estas constantes por medio de un autómata finito.

Expresión Regular:

$$\texttt{entero} = (' + ' + ' - ' + \lambda)(\texttt{digitonocero})(\texttt{digito})^* + ' 0'$$
$$\texttt{digitonocero} =' 1' + ' 2' + ' 3' + ' 4' + ' 5' + ' 6' + ' 7' + ' 8' + ' 9'$$
$$\texttt{digito} = \texttt{digitonocero} + ' 0'$$

Autómata Finito:



Esta tarea resulta tan habitual que se encuentran disponibles varias herramientas de programación que automatizan la tarea de diseñar los programas que se comporten como reconocedores léxicos. Por ejemplo, la orden *lex* del sistema operativo `UNIX` establece un mecanismo para generar un programa ejecutable que se comporte como un autómata finito mediante la especificación de su expresión regular asociada.

2.6.2. Especificación de parámetros de entrada.

En algunos programas que requieran de la interacción del usuario para buscar ciertos patrones de caracteres sobre un determinado fichero, suele resultar bastante aconsejable permitir la introducción de esos datos mediante expresiones regulares. Por ejemplo, en la mayoría de sistemas operativos se incluye la capacidad de buscar ficheros cuyo nombre presente algunas características expresables mediante expresiones regulares.

En el siguiente ejemplo se observa como listar en el sistema operativo UNIX todos los ficheros cuyo nombre empiece por `'p'` *y su tercer carácter sea un* `'5':` `ls p?5*`. *Este comando da lugar al autómata finito que se muestra en la figura 2.14.*

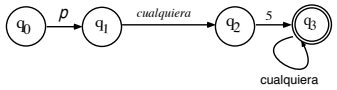


Figura 2.14: Autómata Finito asociado a la expresión `p?5*`

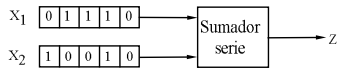
Otro ejemplo de utilización de expresiones regulares en aplicaciones de texto se presenta en el editor de texto `UNIX`, *ed*. Para representar la ejecución de tareas tales como la búsqueda de un texto y su sustitución por otro texto en todo un fichero se utiliza un comando como el siguiente: `s/ bbb*/b`. Al ejecutar este comando se sustituyen las secuencias de dos o más caracteres blancos en el fichero `bbb*` por un único carácter blanco `b`. En este caso el editor *ed* transforma el comando de entrada `s/bbb*/b` en un autómata finito con movimientos libres con el fin de ahorrar memoria al representar el autómata finito.

2.6.3. Ayuda al diseño de circuitos.

Tanto las máquinas de Moore como las máquinas de Mealy se utilizan frecuentemente para diseñar circuitos.

Como ejemplo de este tipo de aplicación se presenta el diseño de un sumador binario serie mediante una máquina de Mealy.

El diagrama de bloques de un sumador binario en serie es el siguiente:



Este diagrama de bloques se puede representar mediante una tabla de estados que contendrá dos estados: q_0 *y* q_1. *El estado* q_0 *representa la condición de acarreo* 0 *mientras que el estado* q_1 *representa la condición de acarreo* 1.

$Estado\ actual$	$x_1x_2 = 00$	01	10	11
q_0	$q_0, 0$	$q_0, 1$	$q_0, 1$	$q_1, 0$
q_1	$q_0, 1$	$q_1, 0$	$q_1, 0$	$q_1, 1$

La entrada $(q_0, 00) = (q_0, 0)$ *representa que si el dígito* x_1 *es* 0, *el dígito* x_2 *es* 0 *y la máquina de Mealy se encuentra en el estado* q_0, *entonces la máquina de Mealy permanece en el estado* q_0 *y produce el dígito de salida* 0.

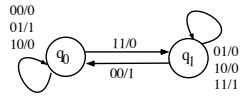


Figura 2.15: Máquina de Mealy para un sumador binario.

2.7. Problemas Propuestos.

1. Simplificar las siguientes expresiones regulares:

   $(a)\ (\lambda + aa)^*$
   $(b)\ (\lambda + aa)(\lambda + aa)^*$
   $(c)\ a(\lambda + aa)^*a + \lambda$
   $(d)\ a(\lambda + aa)^*(\lambda + aa) + a$
   $(e)\ (a + \lambda)a^*b$
   $(f)\ (\lambda + aa)^*(\lambda + aa)a + a$
   $(g)\ (\lambda + aa)(\lambda + aa)^*(\lambda + aa) + (\lambda + aa)$
   $(h)\ (\lambda + aa)(\lambda + aa)^*(ab + b) + (ab + b)$
   $(i)\ (a + b)(\lambda + aa)^*(\lambda + aa) + (a + b)$
   $(j)\ (aa)^*a + (aa)^*$
   $(k)\ a^*b((a + b)a^*b) * + a^*b$
   $(l)\ a^*b((a + b)a^*b)^*(a + b)(aa)^* + a(aa)^* + a^*b((a + b)a^*b)^*$

2. Escribir una Gramática Regular y un AF que reconozca cada uno de los siguientes lenguajes, definidos sobre el alfabeto $\{a, b, c\}$:

   a) Conjunto de cadenas que contengan tres símbolos *c* consecutivos en algún sitio.
   b) Conjunto de todas las cadenas no nulas en las que *bc* no sea una subcadena.
   c) Conjunto de todas las cadenas en las que cada *b* que ocurra vaya inmediatamente seguida por una *c*.
   d) Conjunto de cadenas en las que *abc* sea prefijo y sufijo propio. (Prefijo propio: *abcx*, Sufijo propio: *xabc* , $x \in \Sigma^+$)
   e) Conjunto de cadenas de la forma $(abc)^*$.
   f) Conjunto de cadenas de la forma $V_1V_2 \ldots V_n,\ n \geq 0$, donde $V_i = bc$ o $V_i = ca$.

3. Escribir una Gramática Regular y un AF que reconozca cada uno de los siguientes lenguajes, definidos sobre el alfabeto $\{0, 1\}$:

   a) Conjunto de todas las cadenas acabadas en 00.
   b) Conjunto de cadenas en las que cada bloque de cinco símbolos consecutivos contenga al menos dos ceros.
   c) Conjunto de cadenas tales que su décimo símbolo de derecha a izq. sea un 1. (Nota: escribir un autómata no determinista).

d) Conjunto de cadenas tales que representen una suma binaria de dos sumandos en el que los sumandos y el resultado se leen por columnas de derecha a izquierda y de arriba a abajo. Por ejemplo, la cadena `000011110100001` representa la suma

$$\begin{array}{r} 1100 \\ +\,0110 \\ \hline 10010 \end{array}$$

4. Diseñar un autómata finito cuyo comportamiento se corresponda con el del circuito la figura 2.16, en el que se supone que hay suficiente tiempo entre los cambios en valores de entrada para que se propaguen las señales y la red alcance una configuración estable. Los estados finales se corresponden con salida '1'. Inicialmente $y_1 = y_2 = 0$.

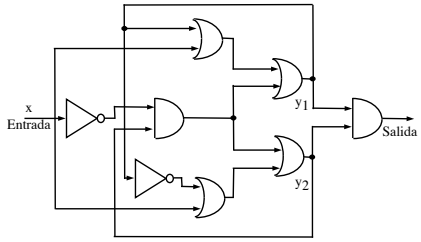


Figura 2.16: Circuito digital a convertir en autómata.

5. Diseñar un autómata finito cuyo comportamiento sea equivalente a la "red neuronal" de la figura 2.17. Los estados finales del autómata deben de corresponderse con la salida "1" de la red.

   Cada neurona tiene sinapsis excitadoras (círculos) e inhibidoras (puntos). Una neurona produce una salida "1" si el número de sinapsis excitadoras con entrada "1" excede el número de sinapsis inhibidoras con entrada "1" por al menos el umbral de la neurona (número del triángulo). Suponer que existe el tiempo suficiente entre los cambios en valores de entrada para que las señales se propaguen y que la red alcance una configuración estable. Inicialmente $y_1 = y_2 = y_3 = 0$.

6. Construir Autómatas Finitos equivalentes a las siguientes expresiones regulares:

   a) $10 + (0 + 11)0^*1$

   b) $01[((10)^* + 111)^* + 0]^*1$

   c) $((0 + 1)(0 + 1))^* + ((0 + 1)(0 + 1)(0 + 1))^*$

7. Construir AFD equivalentes a los siguientes AFN,

$$A = \langle \{p, q, r, s\}, \{0, 1\}, f_1, p, \{s\} \rangle$$

   y

$$M = \langle \{p, q, r, s\}, \{0, 1\}, f_2, p, \{q, s\} \rangle,$$

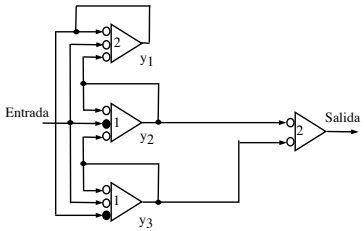


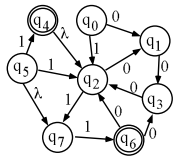
Figura 2.17: Red neuronal a convertir en autómata.

donde f_1 y f_2 vienen dadas por las siguientes tablas:

	0	1
p	$\{p, q\}$	p
q	r	r
r	s	$\emptyset$
s	s	s

	0	1
p	$\{q, s\}$	q
q	r	$\{q, r\}$
r	s	p
s	$\emptyset$	p

8. Construir un AFD que acepte el mismo lenguaje que el siguiente $AF\lambda$:



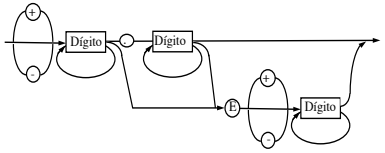
9. Escribir Máquinas de Moore y de Mealy para los siguientes procesos:

   a) Para cadenas de $(0 + 1)^*$; si la cadena acaba en *101*, sacar *A*; si la entrada acaba en *110*, sacar *B*; en cualquier otro caso, sacar *C*.

   b) Para cadenas de $(0 + 1 + 2)^*$ sacar el residuo módulo 5 de la entrada tratada como un número en base 3.

10. Una de las etapas en el desarrollo de un compilador consiste en analizar léxicamente el programa que recibe el compilador con el propósito de localizar nombres de variables, constantes - enteras, reales, carácter, lógicas -, palabras reservadas del lenguaje,... A esta etapa se la denomina "análisis léxico".

    Construir un AF que reconozca constantes con signo enteras y reales con notación en punto fijo o científica. La figura muestra el formato que debe seguir una constante real con signo en notación fija o científica.



11. Demostrar la certeza o falsedad de la siguiente afirmación:

    "Para todo lenguaje regular existe un AF que lo reconoce y que posee un único estado final".

Capítulo 3

Propiedades de los Lenguajes Regulares

Índice General

En este tema se presentan las propiedades que cumplen los lenguajes regulares.

El primer resultado a estudiar, el teorema de Myhill-Nerode, es de especial trascedencia puesto que establece la existencia de un AF mínimo para aceptar un determinado lenguaje regular y, además, permite desarrollar un método para calcularlo.

Además, se presentan los resultados que satisfacen los lenguajes regulares. Estos resultados permitirán saber si un lenguaje dado es o no es regular. El interés de estudiar este tipo de herramientas (lema de bombeo, propiedades de clausura) es muy grande, ya que cuando se determina el tipo más restrictivo al que pertenece un lenguaje, se está determinando el mínimo número de recursos computacionales necesario para reconocer sus cadenas.

3.1. El Teorema de Myhill-Nerode. Minimización de Autómatas Finitos.

Sea D un conjunto, una **relación binaria R** es un subconjunto de $D \times D$, es decir, $R \in D \times D$. Si el par $(a, b) \in R$, entonces se denota como aRb; en caso contrario, se denota como $a\not{R}b$.

Definición 3.1 (Relación Binaria de Equivalencia, RBE) *Una relación binaria se dice de equivalencia -r.b.e.- si, y sólo si:*

1. *cumple la propiedad reflexiva:* $\forall\, x \in D, xRx$,
2. *cumple la propiedad simétrica:* $\forall\, x, y \in D$, *si* xRy *entonces* yRx,
3. *cumple la propiedad transitiva:* $\forall\, x, y, z \in D$, *si* $xRy \wedge yRz$ *entonces* xRz.

Ejemplo:

- $iR_{\leq}j$ *si y sólo si* $i \leq j$ *no es r.b.e., puesto que es reflexiva y transitiva, pero no simétrica.*
- $iR_m j$ *si y sólo si* $(i - j) \bmod m = 0$, *es r.b.e.*

Sea R una relación binaria de equivalencia. Se define la *clase de equivalencia* $[x]_R$ como el conjunto $\{y \in D \mid xRy\}$.

El *índice* de una r.b.e. es el número distinto de clases de equivalencia que contiene.

Sea Σ un alfabeto, se dice que una r.b.e. R, $R \subseteq \Sigma^* \times \Sigma^*$, es invariante a la derecha con respecto a la concatenación si $\forall\, z \in \Sigma^*$ se verifica que xRy implica que $xzRyz$.

Teorema 3.1 (Teorema de Myhill-Nerode) *Las tres afirmaciones siguientes son equivalentes:*

1. *El lenguaje* $L \subseteq \Sigma^*$ *es aceptado por algún AF.*
2. *L es la unión de algunas de las clases de equivalencia de una r.b.e. invariante a la derecha de índice finito.*
3. *Sea* R_L *la r.b.e.*

   $$xR_L y \text{ si y sólo si } \forall\, z \in \Sigma^*, xz \in L \text{ sólo cuando } yz \in L,$$

   entonces R_L *es de índice finito.*

En la demostración del teorema de Myhill-Nerode, que no se verá, se construye un AFD *A* que reconoce un lenguaje regular L según la relación R_L. También se expone que cualquier otra r.b.e. que se establezca sobre un AFD A′ que reconozca el mismo lenguaje va a tener un número de estados que será mayor o igual al número de estados del AFD proporcionado por la demostración del teorema de Myhill-Nerode. Por lo tanto, este autómata tendrá el *número mínimo de estados* que pueda tener cualquier autómata que reconozca el mismo lenguaje.

La pregunta que se establece ahora es, ¿puede existir otro AFD que reconozca este mismo lenguaje que tenga el mismo número de estados que el AFD dado por el teorema de Myhill-Nerode pero que tenga una diferente función de transición?

Teorema 3.2 *El AFD de mínimo número de estados de entre todos los AFD que aceptan un mismo lenguaje regular es único, salvo isomorfismos (renombramiento de estados).*

Ya que el teorema de Myhill-Nerode permite afirmar que hay un único AFD con número mínimo de estados, lo que se plantea a continuación es cómo obtener un método práctico que permita calcularlo.

Sea $\equiv$ la r.b.e. sobre los estados de A tal que $p \equiv q$ si, y sólo si, para cada cadena de entrada *x*, $f(p, x)$ es un estado final si, y sólo si, $f(q, x)$ es un estado final.

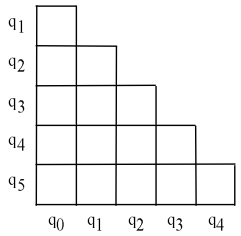
Hay un isomorfismo entre aquellas clases de equivalencia de $\equiv$ que contienen un estado alcanzable desde q_0 para alguna cadena de entrada y los estados del AFD de número mínimo de estados. Por lo tanto, los estados de AFD mínimo se pueden identificar con estas clases.

Se emplea la siguiente notación: si $p \equiv q$ se dice que *p es equivalente a q* y en caso contrario se dice que *p es distinguible de q*.

El cálculo de la relación $\equiv$ se realiza por medio del siguiente método, que se ilustra tomando como ejemplo el AFD presentado en la figura 2.4:

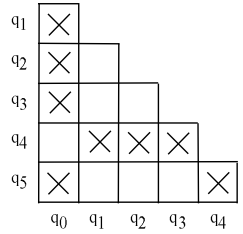
1. Se crea una tabla con $\mid Q \mid -1$ filas y $\mid Q \mid -1$ columnas. Cada fila y cada columna se etiqueta con uno de los estados de *Q*, de forma que si *q* es un estado de *Q* entonces no existen en la tabla entradas correspondientes a parejas *(q,q)*. Asímismo, si *q* y *p* son estados de *Q*, la entrada correspondiente a la pareja *(q,p)* también representa a la pareja *(p,q)*.

   En el ejemplo, la tabla correspondiente sería de la forma,



2. Se marca cada entrada de la tabla que se corresponde con una pareja *(estado final, estado no final)*, pues todas esas parejas se corresponden con pares de estados distinguibles.

En el ejemplo, se tienen las siguientes parejas de estados final/no final,



3. Para cada par de estados *(p,q)* que no se haya analizado hasta el momento, se consideran los pares de estados *(r,s)* tales que $r = f(q, a)$ y $s = f(p, a)$, para cada símbolo de entrada *a*.

   Si los estados *r* y *s* son distinguibles para alguna cadena *x*, entonces los estados *p* y *q* son distinguibles por la cadena *ax*.

   Por lo tanto, si la entrada *(r,s)* está marcada en la tabla, entonces también se marca la entrada *(p,q)*. Si la entrada *(r,s)* no está marcada, entonces el par *(p,q)* se coloca en una lista asociada con la entrada *(r,s)*. Si posteriormente se marca la entrada *(r,s)*, también se marcarán todas las parejas de la lista asociada.

   En el ejemplo, se obtiene lo siguiente:

   - *El análisis del par* (q_0, q_4) *remite al par* (q_1, q_5)*; por lo tanto, se coloca en la lista asociada a dicha entrada.*
   - *El análisis del par* (q_1, q_2) *remite al par* (q_0, q_3)*; por lo tanto, se marca.*
   - *El análisis del par* (q_1, q_3) *remite al par* (q_0, q_2)*; por lo tanto, se marca.*
   - *El análisis del par* (q_1, q_5) *remite al par* (q_0, q_4)*; por lo tanto, no se puede marcar ninguna de las dos entradas.*
   - *El análisis del par* (q_2, q_3) *remite al par* (q_3, q_4)*; por lo tanto, se marca.*
   - *El análisis del par* (q_2, q_5) *remite al par* (q_3, q_4)*; por lo tanto, se marca.*
   - *El análisis del par* (q_3, q_5) *remite al par* (q_2, q_4)*; por lo tanto, se marca.*

Al finalizar este proceso todas aquellas entradas de la tabla que queden vacías identifican parejas de estados equivalentes.

En el ejemplo, los estados q_1 y q_5 son equivalentes, y lo mismo sucede con los estados q_0 y q_4. Se puede comprobar que el autómata obtenido es el presentado en la figura 3.1.

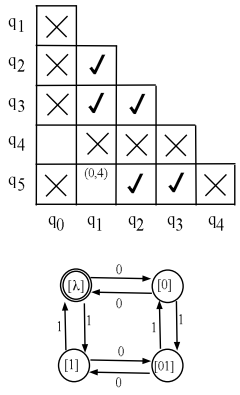


Figura 3.1: Mínimo AFD que reconoce L.

3.2. Lema de Bombeo.

Este lema proporciona una herramienta muy útil para demostrar que ciertos lenguajes **no son regulares**, es decir, que no pueden ser reconocidos por autómatas finitos. Además, como consecuencias adicionales de su enunciado, también resulta útil como referencia teórica para desarrollar algoritmos que respondan a ciertas cuestiones sobre aspectos determinados de autómatas finitos, como por ejemplo, si el lenguaje aceptado por un autómata finito es finito o infinito.

Lema 3.1 (Lema de Bombeo) *Para todo lenguaje regular L existe una constante* n, *dependiente únicamente de L, tal que si* z *es una cadena de L, y se cumple que* $| z | \geq n$, *entonces la cadena* z *se puede descomponer como* z = uvw *tal que:*

1. $| v | \geq 1$,
2. $| uv | \leq n$,
3. *Para todo* $i \geq 0$ *las cadenas* $uv^i w$ *son, todas, cadenas de L.*

Demostración:

Si un lenguaje es regular, entonces es aceptado por un autómata finito determinista, AFD, $A = \langle Q, \Sigma, f, q_0, F \rangle$. Sea |Q| = n, es decir, el AFD tiene *n* estados.

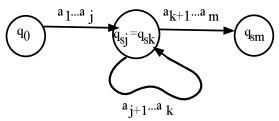
Sea $z = a_1 a_2 \dots a_m$ una cadena de *m* símbolos, tal que $m \geq n$, y supóngase que

$f(q_0, a_1 a_2 \dots a_i) = q_{si}$, donde q_{si} representa el estado que se alcanza después de analizar los primeros a_i símbolos. Según esto, $f(q_0, a_1) = q_{s1}$, $f(q_0, a_1 a_2) = q_{s2}$, ...etc.

Como la cadena z tiene *m* símbolos y el AFD tiene *n* estados distintos, y $m \geq n$, entonces si el AFD A comienza a trabajar con la cadena z, no puede ser que los primeros *n + 1* estados que se suceden en el análisis de la cadena z (q_0, q_{s1}, q_{s2}, ... q_{sn}) sean todos distintos.

Por lo tanto, existen al menos dos enteros, llamémosles *j* y *k*, $1 \leq j < k \leq n$ tales que $q_{sj} = q_{sk}$, es decir, $f(q_0, a_1 \dots a_j) = q_{sj}$, $f(q_0, a_1 \dots a_k) = q_{sk} = q_{sj} \wedge f(q_k, a_{k+1} \dots a_m) = q_{sm}$. Como $j < k$ entonces la subcadena $a_{j+1} \dots a_k$ tiene una longitud mayor o igual que *1*, y como $k \leq n$ entonces su longitud no puede ser mayor que *n*.

Gráficamente, la idea se puede expresar de la forma siguiente:



Si la cadena $a_1 \dots a_m$ pertenece al lenguaje reconocido por el AFD A, entonces $q_{sm} \in F$, y por lo tanto, la cadena $a_1 \dots a_j a_{k+1} \dots a_m$ también pertenece a *L(A)*. En este caso, el bucle que reconoce $a_{j+1} \dots a_k$ no se realiza ninguna vez. Pero también podría darse el caso de realizarlo *i* veces, en cuyo caso la cadena reconocida sería $a_1 \dots a_j (a_{j+1} \dots a_k)^i a_{k+1} \dots a_m$.

¿Qué conclusión se puede sacar de este proceso? Que si se tiene un AFD y una cadena z de una longitud lo suficientemente larga que sea aceptada por el AFD, entonces se puede localizar una subcadena de z tal que esa subcadena se puede repetir tantas veces como se quiera (es decir, se puede "*bombear*") obteniéndose como resultado de este proceso nuevas cadenas que también serán aceptadas por el AFD.

Con esta idea gráfica, para completar la demostración basta con identificar *u* con $a_1 \dots a_j$, *v* con $a_{j+1} \dots a_k$ y *w* con $a_{k+1} \dots a_n$. Como $j < k$ entonces $| v | = k - (j + 1) + 1 = k - j \geq 1$ y como $k \leq n$, entonces $| uv | \leq n$.

c.q.d.

Este lema se aplica para demostrar que un lenguaje *no es regular*, es decir, si un lenguaje no cumple con el lema de bombeo entonces se puede garantizar que no es regular, pero si cumple las condiciones del lema entonces no se puede asegurar si es o no es regular.

Ejemplos:

1. *Demostrar que el lenguaje* $\{a^n b^n \mid n \geq 0\}$ *no es un lenguaje regular.*

Para poder comprobar si se cumplen las condiciones impuestas por el lema de bombeo, se debe localizar una cadena del lenguaje cuya longitud sea mayor que la constante del lema para este lenguaje. Sea t *esa constante. Entonces* $z = a^t b^t$ *pertenece al lenguaje y su longitud es mayor que* t*, por lo tanto* z *se puede escribir como* uvw*, pero ¿cuales son los símbolos de* z *que componen la cadena* v*? Se analizan todos los casos posibles y si al menos uno de estos casos fuese posible, entonces no se podría demostrar que no se cumple el lema para esta cadena:*

a) *Si* v *está compuesta sólo de símbolos* a*, entonces las cadenas* $uv^i w$, $i \geq 2$ *no pertenecen al lenguaje (contienen más* a*'s que* b*'s).*

$$\underbrace{a\ldots}_{u}\underbrace{\overset{(t)}{\ldots}a\ldots}_{v}\underbrace{abb\overset{(t)}{\ldots}b}_{w}$$

b) *Si en la cadena* v *aparece algún símbolo* b*, deja de cumplirse la segunda condición, ya que entonces* $\mid uv \mid > t$*, ya que en la subcadena* u *deberían de estar, al menos, todas los* t *símbolos* a*.*

$$\underbrace{a\ldots}_{u}\underbrace{\overset{(t)}{a}\ldots b\ldots}_{v}\underbrace{b\overset{(t)}{b}\ldots b}_{w}$$

$$\underbrace{\overset{(t)}{a}\ldots ab\ldots}_{u}\underbrace{b\ldots}_{v}\underbrace{\overset{(t)}{\ldots}\ldots b}_{w}$$

Como no hay más opciones posibles para la asignación de símbolos a la cadena v*, entonces este lenguaje no cumple el lema de bombeo y, por lo tanto, no puede ser un lenguaje regular.*

2. *Demostrar que el lenguaje* $L = \{0^p \mid p\ es\ un\ número\ primo\}$ *no es un lenguaje regular.*
Este lenguaje está formado por las cadenas de 0's cuya longitud es un número primo. Sea n *la constante del lema de bombeo y sea* $z = 0^k$ *tal que* k *es un número primo mayor que* n*.*
Como el conjunto de números primos es un conjunto de infinitos elementos se garantiza que ese número primo k *existe, sea cual sea el valor de* n*. Por lo tanto* $z \in L$ *y si L fuese un lenguaje regular entonces deberían de cumplirse las condiciones expuestas por el lema de bombeo, en particular, que*

$$uv^i w \in L\ \forall i \geq 0.$$

Sea $i = k + 1$*; la cadena* $uv^{k+1}w$ *debería pertenecer a L. Pero,*

$$\mid uv^{k+1}w \mid = \mid uv^k vw \mid = \mid uvw \mid + \mid v^k \mid = k + k \mid v \mid = k(1 + \mid v \mid).$$

Es decir, $\mid uv^{k+1}w \mid$ *no es un número primo, puesto que es divisible por* k *y por* $(1 + \mid v \mid)$*. Como consecuencia, el lenguaje L no es un lenguaje regular puesto que no cumple el lema de bombeo.*

3.3. Propiedades de Clausura.

Existen numerosas operaciones que aplicadas a lenguajes regulares dan como resultado otro lenguaje regular. Por ejemplo, la unión de dos lenguajes regulares es un lenguaje regular puesto que si r_1 y r_2 denotan a los lenguajes regulares L_1 y L_2, entonces la expresión regular $r_1 + r_2$ denota al lenguaje $L_1 \cup L_2$. Este tipo de operaciones reciben el nombre de *operaciones de clausura*.

Teorema 3.3 *Los lenguajes regulares son cerrados bajo las operaciones de unión, concatenación, y estrella de Kleene.*

La demostración del teorema anterior se deja propuesto como ejercicio.

Teorema 3.4 *Los lenguajes regulares son cerrados bajo la operación de complementación.*

Demostración:

Sea $A = \langle Q, \Sigma, q_0, f, F\rangle$ un Autómata Finito Determinista tal que L(A) = L. Se construye un AFD $A' = \langle Q, \Sigma, q_0, f, Q - F\rangle$, es decir, A' es un AFD que se diferencia del AFD A en que en A' serán estados finales los que no lo eran en el AFD A y viceversa.

Por lo tanto,

$L(A') = \{x \in \Sigma^* \mid f(q_0, x) \in Q - F\} = \{x \in \Sigma^* \mid f(q_0, x) \notin F\} = \Sigma^* - \{x \in \Sigma^* \mid f(q_0, x) \in F\} = \Sigma^* - L = \bar{L}$.

c.q.d.

Resulta interesante destacar que la condición de que el autómata finito de partida sea determinista es muy importante, puesto que de no ser así, entonces la demostración no sería correcta.

Por ejemplo, si A fuese un AFN tal que $f(q, a) = \{q_j, q_k\}$ y $q_j \in F$, pero $q_k \notin F$, al construir el AF A' se tendría que $q_k \in Q - F$ y por lo tanto sería un estado final de A'. Y como $f(q, a) = \{q_j, q_k\}$ resulta que la cadena a es reconocida por el AF A y por A'. Esto es imposible si el AF A' reconoce el complementario de L(A).

Teorema 3.5 *Los lenguajes regulares son cerrados bajo la operación de intersección.*

Demostración:

Como los lenguajes regulares son cerrados bajo las operaciones de complementación y de unión, entonces si L_1 y L_2 son lenguajes regulares también lo será $\overline{\overline{L_2} \cup \overline{L_1}} = L_1 \cap L_2$.

c.q.d.

¿Cómo se podría construir un AFD que reconociese la intersección de dos lenguajes regulares L_1 y L_2? Un método consistiría en construir el AFD A_1 que reconociese L_1, y el AFD A_2, que reconociese L_2; a partir de ellos se puede construir el AFD A'_1 que reconoce el complementario de L_1 y, posteriormente, el AFD A'_1 que reconoce el complemento de L_1. Con estos, se puede construir el AFD A'_U para reconocer la unión de $L(A'_1)$ y $L(A'_2)$. Para finalizar, se construye el AFD que reconoce el complementario de $L(A'_U)$. Pero este método resulta muy largo.

Este AFD se puede calcular de forma más sencilla aplicando el siguiente método. Sea $A_1 = \langle Q_1, \Sigma, q_0, f_1, F_1 \rangle$ un Autómata Finito Determinista tal que $L(A_1) = L_1$, y sea $A_2 = \langle Q_{12}, \Sigma, p_0, f_2, F_2 \rangle$ otro Autómata Finito Determinista tal que $L(A_2) = L_2$.

Se construye el siguiente AFD $A'' = \langle Q_1 \times Q_2, \Sigma, [q_0, p_0], f, F_1 \times F_2 \rangle$ tal que la función de transición f se define de la forma siguiente:

$$f([q,p],a) = [f_1(q,a), f_2(p,a)].$$

De esta forma el lenguaje reconocido por el AFD A'' es el siguiente,

$$L(A'') = \{x \in \Sigma^* \mid f([q_0,p_0],x) \in F_1 \times F_2\} =$$

$$\{x \in \Sigma^* \mid f_1(q_0,x) \in F_1 \wedge f_2(p_0,x) \in F_2\} = L_1 \cap L_2.$$

Ejemplo de aplicación del teorema:

Sea $L = \{x \in (0+1)^* \mid S(0,x) = S(1,x)\}$. ¿Es L un lenguaje regular?

El lenguaje L está formado por cadenas que tienen el mismo número de 0's que de 1's dispuestos en cualquier posición.

Si L fuese un lenguaje regular entonces al intersectarlo con un lenguaje regular debería dar como resultado un lenguaje regular. El lenguaje 0^*1^* es un lenguaje regular, puesto que es una expresión regular. Por lo tanto si L fuese regular también lo debería ser el lenguaje $L \cap 0^*1^*$, pero este lenguaje es el lenguaje 0^n1^n que ya se ha visto que no es un lenguaje regular.

Por lo tanto, L tampoco puede ser regular.

Teorema 3.6 *Los lenguajes regulares son cerrados bajo la operación de inversión.*

Demostración:

Sea $A = \langle \Delta, Q, f, q_0, F \rangle$ un autómata finito arbitrario y sea el autómata finito $A' = \langle \Sigma, Q', f', q'_0, F' \rangle$, definido como $Q' = Q \cup \{q'_0\}$ y $F' = \{q_0\}$, tal que $q'_0 \notin Q$, es decir, q'_0 es un estado nuevo.

La función de transición f' se define en dos pasos:

1. $f'(q'_0, \lambda) = F$,
2. $q \in f(p,a) \Leftrightarrow p \in f'(q,a), a \in (\Sigma \cup \{\lambda\})$.

Con esta construcción se garantiza que $L(A') = [L(A)]^{-1}$.

Adicionalmente, hay que comentar que el método de construcción de A' se puede simplificar *si en el conjunto de estados finales F sólo hay un estado*, es decir, $F = \{q_f\}$. Entonces, la construcción del autómata finito A' se puede realizar de esta otra forma:

$$Q = Q'$$
$$q'_0 = q_f$$
$$F' = \{q_0\}$$
$$q \in f(p,a) \Leftrightarrow p \in f'(q,a), a \in (\Sigma \cup \{\lambda\})$$

c.q.d.

Ejemplo:

Sea $L = \{x \in 0^m1^n \mid m \leq n\}$. ¿Es L un lenguaje regular?.

Supóngase que L es regular; entonces, también lo debe de ser $L^{-1} = \{1^n0^m \mid m \leq n\}$.

Se define el homomorfismo $g : \{0,1\} \to \{0,1\}^*$ tal que $g(0) = 1$ y $g(1) = 0$.

Como la operación de homomorfismo es una operación de clausura para los lenguajes regulares, entonces el lenguaje $g(L^{-1})$ también debe ser un lenguaje regular . Y, por lo tanto, el lenguaje $L \cap g(L^{-1})$ debe ser regular, pero este lenguaje es

$$L \cap g(L^{-1}) = \{0^m1^n \mid m \leq n\} \cap \{0^n1^m \mid m \leq n\} = \{0^n1^n \mid n \geq 0\}.$$

que no es un lenguaje regular. Se deduce entonces que la suposición inicial, L es un lenguaje regular, no puede ser cierta.

3.4. Algoritmos de Decisión para Lenguajes Regulares.

Un gran número de cuestiones sobre lenguajes regulares se pueden resolver mediante el uso de algoritmos de decisión como, por ejemplo, determinar si un lenguaje regular es vacío, finito o infinito, o bien determinar si una determinada cadena puede ser generada por una determinada gramática regular.

Como se verá en temas siguientes, estas cuestiones no siempre serán resolubles para lenguajes de otros tipos.

Teorema 3.7 *El conjunto de cadenas aceptadas por un autómata finito*

$$A = \langle \Sigma, Q, q_0, f, F \rangle$$

tal que $\mid Q \mid = n$ *es:*

1. *No vacío* $\Leftrightarrow \exists\, x \in L(A) / \mid x \mid < n$.
2. *Infinito* $\Leftrightarrow \exists\, x \in L(A) / n \leq \mid x \mid < 2n$.

Demostración:

1. No vacío $\Leftrightarrow \exists\, x \in L(A) / \mid x \mid < n$.

   "$\Leftarrow$": Obviamente, si existe una cadena que pertenece a L(A), entonces L(A) no es el conjunto vacío.

   "$\Rightarrow$": Si L(A) es no vacío es porque existe al menos una cadena que pertenece a L(A). Sea x esa cadena. Si $\mid x \mid < n$ entonces ya está demostrado el teorema.
   Si $\mid x \mid \geq n$ entonces, por el lema de bombeo, resulta que $x = uvw$, cumpliéndose también que $\forall\, i \geq 0,\ uv^i w \in L(A)$. En particular, si se selecciona $i = 0$, se obtiene la cadena $uv^0 w = uw$ que, como $\mid v \mid \geq 1$, cumple que $\mid x \mid = \mid uvw \mid > \mid uw \mid$ y $uw \in L(A)$. Si $\mid uw \mid < n$, entonces ya se ha conseguido demostrar el teorema. En caso contrario, se vuelve a aplicar este razonamiento.
   Como cada vez que se aplica este razonamiento se obtienen cadenas de menor longitud, tiene que llegar un momento en que se obtenga una cadena cuya longitud sea menor que *n*.

   Como consecuencia, un posible algoritmo para poder afirmar si el lenguaje que reconoce un AF es o no es vacío, consistiría en determinar si hay alguna cadena de longitud menor que *n*, siendo $\mid Q \mid = n$, que pertenezca al lenguaje (nótese que como *n* es un valor fijo, hay un número finito de posibles cadenas que pudieran pertenecer al lenguaje; por lo tanto, se asegura el fin del proceso).

2. Infinito $\Leftrightarrow \exists\, x \in L(A) / n \leq \mid x \mid < 2n$.

   "$\Leftarrow$": Como $\mid x \mid \geq n$, entonces, por el lema de bombeo, se cumple que $\forall\, i \geq 0, uv^i w \in L(A)$. Por lo tanto, se puede afirmar que hay un número infinito de cadenas que pertenecen a $L(A)$.

   "$\Rightarrow$": Se sigue un proceso de razonamiento similar al realizado en el caso "$\Rightarrow$" del apartado anterior, teniendo en cuenta que $\mid v \mid \geq 1$ y $\mid uv \mid \leq n$.

   Como consecuencia, un posible algoritmo para poder afirmar si el lenguaje que reconoce un AF es o no es infinito, consistiría en determinar si hay alguna cadena de longitud menor que *2n* y menor o igual a *n*, siendo $\mid Q \mid = n$, que pertenezca al lenguaje.

c.q.d.

Teorema 3.8 *Existe un algoritmo para determinar si dos autómatas finitos deterministas reconocen el mismo lenguaje.*

Demostración:

Sea A_1 un AFD $\mid L(A_1) = L_1$ y sea A_2 otro AFD $\mid L(A_2) = L_2$. Se construye un AFD A′ que reconozca el lenguaje,

$$L(A') = (L_1 \cap \overline{L}_2) \cup (\overline{L}_1 \cap L_2).$$

Entonces $L_1 = L_2 \Leftrightarrow L(A') = \emptyset$, que, según el teorema 3.7, es una cuestión decidible.

c.q.d.

3.5. Problemas Propuestos.

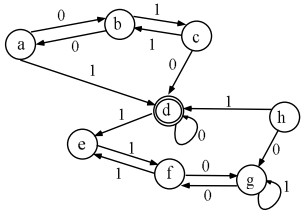
1. ¿Cuáles de los siguientes lenguajes son regulares?. Demostrar la respuesta.

   a) $\{0^{2n} \mid n \geq 1\}$.
   b) $\{0^m 1^n 0^{m+n} \mid m \geq 1 \wedge n \geq 1\}$.
   c) El conjunto de todas las cadenas que no tienen tres ceros consecutivos.
   d) El conjunto de todas las cadenas con igual número de ceros que de unos.
   e) $\{x \mid x \in (0+1)^* \wedge x = x^{-1}\}$.
   f) $\{xwx^{-1} \mid x, w \in (0+1)^+\}$.

2. Calcular el AFD mínimo equivalente a la siguiente Gramática Regular:

$$S \to A \mid 0C \mid 1 \mid \lambda$$
$$A \to 1A \mid 0D_1 \mid 0D_2 \mid 0D_3 \mid 1$$
$$D_1 \to 0B$$
$$D_2 \to 1C$$
$$D_3 \to 0$$
$$B \to 1S \mid 0B \mid 0$$
$$C \to 1A \mid 0C \mid 1$$

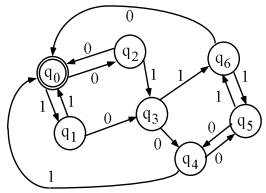
3. Calcular el AFD mínimo equivalente al siguiente AFD:



4. Calcular el AFD mínimo equivalente a la siguiente Gramática Regular:

$$S \to A \mid 0C \mid 1 \mid \lambda$$
$$A \to 1A \mid 0D_1 \mid 0D_2 \mid 0D_3 \mid 1$$
$$D_1 \to 0B$$
$$D_2 \to 1C$$
$$D_3 \to 0$$
$$B \to 1S \mid 0B \mid 0$$
$$C \to 1A \mid 0C \mid 1$$

5. Calcular el AFD mínimo equivalente al siguiente AFD:



6. Calcular el mínimo AFD que reconozca

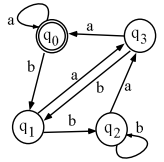
$$L = (ab^* + b^*a)ab\ .$$

7. Calcular el AFD mínimo asociado al AFN definido por la siguiente función de transición:

	0	1
q_0	$\{q_1, q_2\}$	$\{q_3\}$
q_1	$\{q_1, q_4\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$
q_3	$\emptyset$	$\{q_1, q_2, q_4\}$
q_4	$\{q_3\}$	$\{q_4\}$

donde $F = \{q_4\}$.

8. Sea A el AFD de la figura:



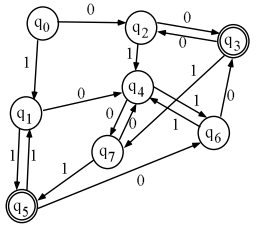
   a) Calcular el AFD mínimo, A', equivalente.

   b) Calcular la expresión regular que denota al conjunto de cadenas reconocidas por dicho autómata.

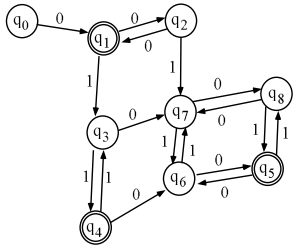
   c) Calcular una Gramática Regular Lineal a la Izquierda que genere las cadenas reconocidas por el AFD mínimo A'.

   d) Escribir un algoritmo que reconozca las mismas cadenas que A'.

9. Calcular el AFD mínimo equivalente al de la siguiente figura.



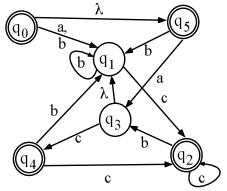
10. Calcular el AFD mínimo equivalente al de la siguiente figura.



11. Resolver el siguiente sistema de ecuaciones:

$$X_1 = aX_2 + bX_3$$
$$X_2 = aX_2 + bX_4 + \lambda$$
$$X_3 = aX_2 + bX_4 + \lambda$$
$$X_4 = bX_4 + \lambda$$

12. Dado el AFλ de la figura,

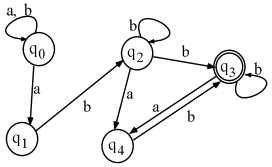


a) Calcular un AFD equivalente.

b) Calcular el AFD mínimo equivalente.

c) Calcular la expresión regular asociada.

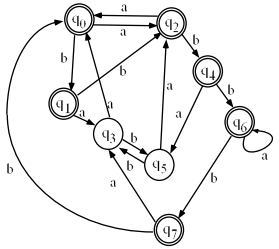
13. Calcular el AFD mínimo equivalente al siguiente AFN.



14. Calcular la expresión regular asociada al siguiente sistema de ecuaciones lineales en expresiones regulares (nota: minimizar el AFD asociado)

$$X_0 = 0X_2 + 1X_1 \quad X_1 = 0X_4 + 1X_5$$
$$X_2 = 0X_3 + 1X_4 \quad X_3 = 0X_2 + 1X_7 + \lambda$$
$$X_4 = 0X_7 + 1X_6 \quad X_5 = 0X_6 + 1X_1 + \lambda$$
$$X_6 = 0X_3 + 1X_4 \quad X_7 = 0X_4 + 1X_5$$

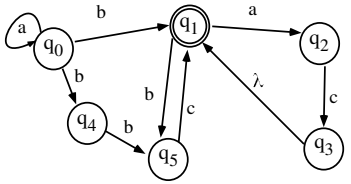
15. Sea A el AFD de la figura:



a) Calcular el AFD de mínimo número de estados que reconoce L(A).

b) Calcular una e.r. que denote a L(A).

16. Sea el AF de la figura:



Calcular

a) el AFN equivalente,

b) el AFD equivalente,

c) el mínimo AFD equivalente, y la expresión regular que denota a L(AF).

17. Escribir una Gramática Regular para el siguiente lenguaje

$$L = \{w \in \{a,b\}^* \mid w \textit{ no contiene la subcadena } aa\}$$

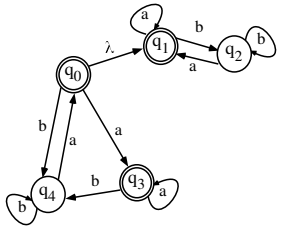
18. Dados L_1 y L_2, lenguajes regulares, demostrar si los siguientes lenguajes son o no son lenguajes regulares:

    a) $L_3 = \{L_1^n L_2^n \mid n \geq 0\}$
    b) $L_4 = L_2^{-1} L_1^{-1}$

19. Calcular

    a) el AFN equivalente,
    b) el AFD equivalente,
    c) el mínimo AFD equivalente, y
    d) la expresión regular que denota a L(AF).

    del AF de la figura:



Capítulo 4

Gramáticas de Contexto Libre y Autómatas de Pila

Índice General

4.1. Gramáticas y Lenguajes de Contexto Libre.

Tal y como se expuso en el tema anterior existen ciertos lenguajes muy usuales que no pueden ser generados por medio de gramáticas regulares. Algunos de estos lenguajes sí pueden ser generados por otro tipo de gramáticas, las denominadas *gramáticas de contexto libre* (también denominadas gramáticas incontextuales, o independientes del contexto). Uno de los ejemplos más típicos consiste en la gramática de contexto libre que genera el lenguaje formado por aquellas cadenas que tengan igual número de paréntesis abiertos que cerrados.

Definición 4.1 *Una Gramática de Contexto Libre (GCL) es una cuádrupla* $G = \langle \Sigma_A, \Sigma_T, P, S \rangle$ *donde*

Σ_A *es un conjunto finito y no vacío de símbolos denominados auxiliares,*

Σ_T *es un conjunto finito de símbolos denominados terminales,* $\Sigma_A \cap \Sigma_T = \emptyset$*,*

S *es el símbolo inicial de la Gramática,* $S \in \Sigma_A$*,*

P *es el conjunto de producciones, donde cada producción es de la forma* $A \rightarrow \beta$ *con* $A \in \Sigma_A$ *y* $\beta \in (\Sigma_T \cup \Sigma_A)^*$*.*

Un lenguaje es *de tipo 2*, también denominado *lenguaje de contexto libre*, si la gramática más restrictiva que puede generarlo, según la jerarquía de Chomsky, es de tipo 2.

Ejemplos de GCL:

- *El lenguaje* $L_1 = \{0^n 1^n \mid n \geq 0\}$ *puede ser generado por medio de la siguiente GCL*

$$S \rightarrow \lambda \mid 0S1$$

Efectivamente, para comprobar que esta gramática genera el lenguaje L_1 *se realiza un análisis parecido al proceso de demostración por inducción:*

Paso base *la cadena de menor longitud que pertenece al lenguaje* L_1 *es* λ*;*

Hipótesis de inducción *supóngase que desde S se puede generar cualquier cadena de longitud* k *que pertenezca a* L_1*;*

Paso de inducción *la siguiente cadena que pertenece al lenguaje* L_1 *de mayor longitud a la de S es la cadena 0S1.*

- *El lenguaje* $L_2 = \{0^n 1^m \mid n \geq m\}$ *puede ser generado por medio de la siguiente GCL*

$$S \rightarrow \lambda \mid 0S \mid 0S1$$

- *El lenguaje* $L_3 = \{w \in (0+1)^* \mid w = w^{-1}\}$ *puede ser generado por medio de la siguiente GCL*

$$S \rightarrow \lambda \mid 0S0 \mid 1S1 \mid 0 \mid 1$$

4.2. Simplificación de GCL.

Aplicando las producciones de la gramática desde el símbolo inicial, se obtienen las cadenas del lenguaje. Una tarea que resulta de gran interés y que, por lo tanto, será una de las primeras tareas que hay que realizar con una GCL es eliminar todas aquellas producciones que no aporten ningún tipo de información válida en la generación de alguna de las cadenas

del lenguaje. Es decir, hay que simplificar la GCL sin alterar el conjunto de cadenas que es capaz de generar.

Cualquier lenguaje de contexto libre, *L*, puede ser generado por medio de una GCL, *G*, que cumpla las siguientes condiciones:

1. Cada símbolo (terminal o auxiliar) de G se emplea en la derivación de alguna cadena de *L*.
2. En el conjunto de producciones de G no existen *producciones unitarias*, es decir, producciones de la forma $A \to B$ donde $A, B \in \Sigma_A$.
3. Si $\lambda \notin L$ entonces en el conjunto de producciones de G no existen *producciones vacías*, es decir, producciones de la forma $A \to \lambda$.

El objetivo de estas condiciones es determinar una GCL en la cual, en cada paso de derivación de una cadena, siempre se introduce información relevante.

Dada una GCL, *G*, se puede construir una GCL, *G'*, tal que *L(G) = L(G')* y en *G'* no hay símbolos inútiles, es decir, que cumple la primera de las condiciones establecidas previamente.

Para construir esta GCL *G'* se aplican dos lemas. El primero de ellos determina el conjunto de símbolos a partir de los cuales se puede obtener una cadena del lenguaje; el segundo lema determina el conjunto de símbolos que pueden aparecer en una forma sentencial de la gramática, es decir, los símbolos que pueden ser alcanzados desde el símbolo inicial de la gramática.

Lema 4.1 (de la derivabilidad) *Dada una GCL, $G = \langle \Sigma_A, \Sigma_T, P, S \rangle$, $L(G) \neq \emptyset$, puede construirse una GCL equivalente $G' = \langle \Sigma'_A, \Sigma_T, P', S \rangle$ en la que se cumple que $\forall\, A \in \Sigma'_A\ \exists\, w \in \Sigma^*_T$ tal que $A \overset{*}{\Rightarrow} w$.*

Para construir la GCL G′ se calcula el conjunto de símbolos derivables por medio de la siguiente fórmula:

Paso Base: $\forall\ A \in \Sigma_A$, $\forall\ w \in \Sigma^*_T$ tal que $A \to w \in P$, entonces se sabe que A es derivable.

Paso Recursivo: Si $(A \to \alpha) \in P$ y si todos los símbolos auxiliares de α son derivables, entonces el símbolo A también es derivable.

Es decir, se inicia el conjunto de símbolos derivables de acuerdo al paso base y, posteriormente, se añaden elementos de acuerdo al paso recursivo hasta que no se detecten nuevos símbolos derivables en el conjunto de derivables, Σ'_A.

Una vez obtenido Σ'_A se define *P'* como el siguiente conjunto

$$P' = \{(A \to \alpha) \in P \mid A \in \Sigma'_A \wedge \alpha \in (\Sigma_T \cup \Sigma'_A)^*\}$$

Lema 4.2 (de la alcanzabilidad) *Dada una GCL, $G = \langle \Sigma_A, \Sigma_T, P, S \rangle$ tal que $L(G) \neq \emptyset$, puede construirse una GCL equivalente $G' = \langle \Sigma'_A, \Sigma'_T, P', S \rangle$ tal que $\forall\ A \in (\Sigma'_A \cup \Sigma'_T)\ \exists\, \alpha, \beta \in (\Sigma'_A \cup \Sigma'_T)^* \mid S \overset{*}{\Rightarrow} \alpha A \beta$.*

Para construir la GCL G′ se calcula el conjunto de símbolos alcanzables por medio del proceso iterativo:

- $\Sigma'_A = \{S\}, \Sigma'_T = \emptyset$,
- $\forall\ (A \to \alpha) \in P$: si $A \in \Sigma'_A$, todos los símbolos auxiliares de α se añaden a Σ'_A y todos sus símbolos terminales se añaden a Σ'_T.

Se define *P′* como el siguiente conjunto

$$P' = \{(A \to \alpha) \in P \mid A \in \Sigma'_A \wedge \alpha \in (\Sigma'_T \cup \Sigma'_A)^*\}$$

Teorema 4.1 *Todo LCL L, $L \neq \emptyset$, puede ser generado por una GCL sin símbolos inútiles.*

La aplicación de los lemas a cada GCL debe realizarse según el orden indicado, puesto que en caso contrario se podría obtener una GCL con símbolos inútiles.

Ejemplo:

Sea G la GCL definida por medio de las siguientes producciones:

$$\{S \to aABC \mid a; A \to a; B \to b; E \to b\}.$$

Si se aplica primero el lema 4.1 se obtiene la GCL

$$\{S \to a; A \to a; B \to b; E \to b\},$$

y al aplicar a continuación el lema 4.2 se obtiene la GCL

$$\{S \to a\}.$$

Si a G se le aplicase primero el lema 4.2 se obtendría la GCL siguiente

$$\{S \to aABC \mid a; A \to a; B \to b\},$$

y al aplicar posteriormente el lema 4.1 la GCL resultante sería la siguiente

$$\{S \to a; A \to a; B \to b\}$$

que contiene los símbolos inútiles {A, B}.

El siguiente paso para simplificar una GCL consiste en eliminar los símbolos anulables.

Si $A \to \lambda \in P$, entonces la aparición del símbolo A en alguna forma sentencial intermedia del proceso de generación de una cadena, x, del lenguaje supone que, más pronto o más tarde, el auxiliar A será sustituido por λ, y que, por lo tanto, dicho símbolo no habrá reportado ninguna utilidad en el proceso de generación de la cadena x.

Teorema 4.2 *Para toda GCL G | L(G) = L, existe una GCL G′ tal que L(G′) = L(G) - {λ} y G′ no contiene símbolos inútiles ni anulables.*

Sea $G = \langle \Sigma_A, \Sigma_T, P, S \rangle$ una GCL. El primer paso consiste en calcular los símbolos anulables, es decir, el conjunto $\mathcal{A} = \{X \in \Sigma_A \mid X \overset{*}{\Rightarrow} \lambda\}$. Este conjunto se puede calcular por medio del siguiente proceso iterativo:

- $A_1 = \{X \in \Sigma_A \mid X \to \lambda) \in P\}$,
- $A_{i+1} = A_i \cup \{X \in \Sigma_A \mid \exists\, \alpha \in A_i^+ : (X \to \alpha) \in P\}, \forall\, i \geq 1$.

El proceso finaliza cuando para algún valor de *k* se cumple que $A_{k+1} = A_k$.

Sea $G_2 = \langle \Sigma_A, \Sigma_T, P_2, S \rangle$ la GCL tal que el conjunto de producciones P_2 se construye a partir de P y del conjunto A de la siguiente forma:

$(X \to \beta) \in P_2 \Leftrightarrow \beta \neq \lambda \wedge \exists\, \alpha \in (\Sigma_A \cup \Sigma_T)^+ : (X \to \alpha) \in P \wedge \beta$ se puede obtener de α eliminando ninguna, una o más apariciones de ninguno, uno o más símbolos del conjunto de símbolos anulables $\mathcal{A}$.

Una vez obtenida G_2 se le aplica el teorema 4.1 para eliminar los símbolos inútiles[1], obteniéndose la GCL buscada.

Ejemplo:

Sea G la GCL dada por el siguiente conjunto de producciones:

$$S \to ABC \mid Da$$
$$A \to aAb \mid \lambda$$
$$B \to bBc \mid \lambda$$
$$C \to cCa \mid ca$$
$$D \to AB \mid a$$

El conjunto de símbolos anulables, A, se calcula de acuerdo al método indicado:

$$A_1 = \{A, B\},\ A_2 = \{A, B, D\},\ A_3 = A_2 = \mathcal{A}$$

[1] Si bien en la práctica suele aplicarse este paso primero, se debe tener en cuenta que tras eliminar las transiciones λ es posible que vuelvan a aparecer símbolos inútiles.

El conjunto de producciones P_2 será el siguiente:

$$S \to ABC \mid AC \mid BC \mid C \mid Da \mid a$$
$$A \to aAb \mid ab$$
$$B \to bBc \mid bc$$
$$C \to cCa \mid ca$$
$$D \to AB \mid A \mid B \mid a$$

Corolario 4.1 *Dada una GCL G puede construirse una GCL G′ equivalente a G tal que no tenga producciones λ excepto cuando λ ∈ L(G) en cuyo caso S′ → λ es la única producción en la que aparece λ y además S′ no aparece en el consecuente de ninguna otra regla de producción.*

Efectivamente, si $G = \langle \Sigma_A, \Sigma_T, P, S \rangle$ es una GCL, tal que $\lambda \in L(G)$, entonces se construye la gramática $G_2 = \langle \Sigma_A, \Sigma_T, P_2, S \rangle$ siguiendo el método expuesto en la demostración del teorema 4.2.

La relación entre G y G_2 es $L(G_2) = L(G) - \{\lambda\}$. Para construir la gramática G′ del corolario se define G′ como $\langle \Sigma'_A, \Sigma_T, P', S' \rangle$ tal que $\Sigma'_A = \Sigma_A \cup \{S'\},\ S' \in \Sigma_A,\ y\ P' = P_2 \cup \{S' \to \lambda \mid S\}$.

El último paso para simplificar una GCL consiste en eliminar las producciones unitarias, es decir aquellas producciones de la forma $A \to B$, ya que cuando aparecen en una derivación lo único que introducen es un cambio de nombre del auxiliar.

Teorema 4.3 *Todo LCL L, tal que λ ∉ L, se puede generar por una GCL G que no contiene producciones unitarias ni símbolos inútiles ni anulables.*

Sea $G = \langle \Sigma_A, \Sigma_T, P, S \rangle$ una GCL tal que L(G) = L y en G no existen producciones λ. El teorema 4.2 asegura que existe. El nuevo conjunto de producciones se forma de la manera siguiente:

Si $(A \to B) \in P$ y $A, B \in \Sigma_A$, entonces esta producción se elimina del nuevo conjunto de producciones y se introducen las siguientes nuevas producciones: $(A \to \gamma) \mid \gamma$ es la primera forma sentencial que se obtiene a partir de B, al aplicar producciones de P, y que cumple que $\mid \gamma \mid \geq 2 \vee \gamma \in \Sigma_T$.

Una vez obtenida G_1 se eliminan los símbolos inútiles[2] obteniéndose la GCL buscada.

[2] Recuérdese lo comentado en el pie de página 1.

4.3. Autómatas de Pila.

La tarea de reconocimiento de los LCL se realiza por medio de un tipo de autómata denominado *autómata de pila*, AP.

4.3.1. Autómatas de Pila No Deterministas.

Definición 4.2 *Un Autómata de Pila es una séptupla*

$$A = \langle \Sigma, Q, \Gamma, f, q_0, Z_0, F \rangle$$

donde

Σ *es el alfabeto de entrada,*

Q *es el conjunto de estados, que es finito y no vacío,*

Γ *es el alfabeto de la pila,*

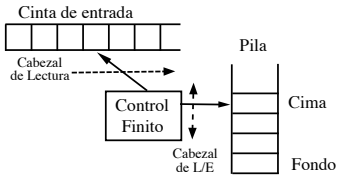
q_0 *es el estado inicial,*

Z_0 *es un símbolo especial, denominado* fondo de pila, $Z_0 \in \Gamma$,

F *es el conjunto de estados finales,* $F \subseteq Q$,

f *es la función de transición,* $f : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \longrightarrow 2^{Q \times \Gamma^*}$.

Una descripción informal de tal autómata es la representada en la siguiente figura:



El autómata de pila consta de una cinta de entrada dividida en celdas, cada una de las cuales puede almacenar un sólo símbolo. El acceso a cada celda se realiza por medio del cabezal lector, cuyo movimiento siempre es de izquierda a derecha. El autómata posee un cabezal adicional de lectura/escritura que sólo puede leer o escribir sobre la cima de una pila. Por eso en este autómata se dispone de dos alfabetos de símbolos, el alfabeto de la cinta de entrada, Σ, y el de la pila, Γ.

El funcionamiento de este autómata es el siguiente: dado un *símbolo de la cinta de entrada* o bien la cadena vacía[3] (que equivale a no leer el símbolo que se encuentra bajo el cabezal de lectura), el *estado actual* del autómata (especificado en el control finito) y el símbolo que esté en la *cima de la pila*, entonces este autómata

1. cambia de estado,
2. elimina el símbolo de la cima de la pila y apila ninguno, uno o varios símbolos en la misma, y
3. mueve el cabezal de lectura sobre la cinta de entrada una celda a la derecha, siempre y cuando se hubiese leído un símbolo de la misma.

Es decir, dada la transición $f(q, a, X) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_m, \gamma_m)\}$ el autómata de pila, en el supuesto que se seleccione para ejecución la acción $f(q, a, X) = (p_i, \gamma_i)$, con $\gamma_i = RYZ$, realiza los siguientes cambios:

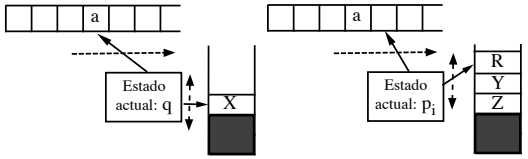


Figura 4.1: Efecto de realizar una transición genérica en un AP.

Como se observa en la figura 4.1, se transita desde el estado *q* al estado p_i (que pasa a ser el estado actual) el cabezal de lectura se desplaza una celda a la derecha y en la pila desaparece el símbolo *X* de la cima y en su lugar se introducen los símbolos *Z*, *Y* y *R*, en este orden.

El comportamiento es similar en el caso de ejecutar una transición del tipo $f(q, \lambda, X) = \{..., (p_i, \gamma_i), ...\}$, con la salvedad de que, en este caso, el cabezal de lectura de la cinta *no* se hubiera desplazado hacia la derecha y permanecería sobre el símbolo *a*.

Para poder describir el comportamiento del AP sin necesidad de tener que especificar de forma gráfica su evolución, se suelen utilizar las *descripciones instantáneas*; en el caso de un AP, una descripción instantánea es un elemento del conjunto $Q \times \Sigma^* \times \Gamma^*$. Así, por ejemplo, la descripción instantánea (q, w, α) $(\in Q \times \Sigma^* \times \Gamma^*)$ representa la situación ilustrada en la figura 4.2, en la que la cadena de entrada es *x*, de la cual falta todavía por analizar el sufijo *w*. El cabezal de lectura está sobre el primer símbolo de *w*, el estado actual del autómata es *q* y el contenido de la pila es α.

[3] Más adelante, se remarcará esta cuestión, pero nótese que esta posibilidad de no leer el símbolo bajo el cabezal hace que el comportamiento de unl autómata de pila sea no determinista por propia definición.

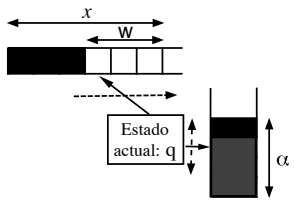


Figura 4.2: Representación gráfica de la DI (q, w, α).

Se dice que una descripción instantánea I_1 *alcanza* a otra descripción instantánea I_2 *en un sólo paso*, y se denota como $I_1 \vdash I_2$, cuando se cumplen las siguientes condiciones:

$$I_1 = (q, aw, X\gamma) \wedge$$
$$I_2 = (p, w, \delta\gamma) \wedge$$
$$(p, \delta) \in f(q, a, X) \mid \ a \in (\Sigma \cup \{\lambda\}) \wedge X \in \Gamma$$

Se dice que una descripción instantánea I_1 *alcanza* a otra descripción instantánea I_2, y se denota como $I_1 \overset{*}{\vdash} I_2$, si $\exists\, n$ descripciones instantáneas auxiliares $DI_1, DI_2, \ldots, DI_n$ tal que a través de ellas se pueda alcanzar I_2 desde I_1, es decir,

$$I_1 = DI_1 \vdash DI_2 \vdash \ldots \vdash DI_n = I_2.$$

Con estas definiciones ya se está en condiciones de poder establecer cuál es el lenguaje aceptado por un AP, en el que hay que diferenciar dos casos.

Definición 4.3 *Se define el* **lenguaje aceptado por estado final** *de un AP A al conjunto*

$$L(A) = \{x \in \Sigma^* \mid (q_0, x, Z_0) \overset{*}{\vdash} (p, \lambda, \gamma) \ tal\ que\ p \in F\}.$$

Se define el **lenguaje aceptado por pila vacía** *de un AP A al conjunto*

$$N(A) = \{x \in \Sigma^* \mid (q_0, x, Z_0) \overset{*}{\vdash} (p, \lambda, \lambda)\}.$$

Se observa que en la definición de N(A), no importa el estado al que se llegue al analizar la cadena x. Es suficiente con que, al finalizar su análisis, la pila quede vacía.

Ejemplo:

Sea A el siguiente AP:

$$A = \langle \{a, b\}, \{q_0\}, \{S, A, B\}, f, q_0, S, \emptyset \rangle$$

donde la función de transición f se define de la forma siguiente:

$$f(q_0, a, S) = \{(q_0, SB), (q_0, ASB)\}$$
$$f(q_0, \lambda, S) = \{(q_0, \lambda)\}$$
$$f(q_0, b, B) = \{(q_0, \lambda)\}$$
$$f(q_0, a, A) = \{(q_0, \lambda)\}$$

Como se observa, en este AP el conjunto de estados finales es $\emptyset$. Por lo tanto, este AP reconoce el lenguaje $\emptyset$ por el criterio de estado final.

El lenguaje reconocido por este autómata por el criterio de pila vacía es $N(A) = \{a^m b^n \mid m \geq n\}$.

Para comprobar que este AP reconoce la cadena aaabb *se van a especificar todas las posibles transiciones que se pueden realizar con este autómata; el resultado se muestra en la figura 4.3.1. Como al menos una secuencia de transiciones consigue vaciar la pila y consumir toda la cadena de entrada, entonces la cadena* aaabb *será aceptada.*

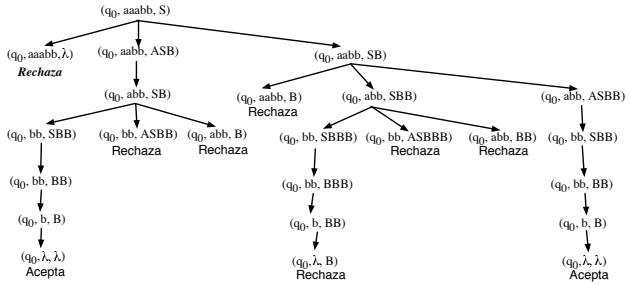


Figura 4.3: Análisis de todas las posibles transiciones que puede realizar el autómata A sobre la cadena *aaabb*.

4.3.2. Autómatas de Pila Deterministas.

El último ejemplo de la subsección anterior, pone de manifiesto lo que la definición de un autómata de pila implica un *comportamiento no determinista*. Es posible diseñar Autómatas de Pila con comportamiento determinista, si bien, tal y como se verá, no se puede establecer una equivalencia entre un APND y un APD.

De forma general, un autómata de pila determinista, APD, es un AP en el que es aplicable una, y sólo una, transición en cada instante. Sin embargo, esta posibilidad no basta para excluir el comportamiento no determinista, ya que puede darse la situación de que, desde un mismo estado y con un mismo símbolo en la cima de la pila, se puedan realizar acciones diferentes si existen transiciones con lectura de símbolo o sin lectura de símbolo, es decir,