Índice general

Capítulo 1

Propiedades cerradas y algoritmos de decisión

No todos los lenguajes son regulares y las herramientas para trabajar (reconocer, expresar y generar) sobre los lenguajes regulares estan limitadas por el número finito de estados o de clases de equivalencia. Por lo que un lenguaje no regular (tipo 2 al tipo 0) no tiene la posibilidad de modelarse efectivamente con herramientas regulares (AF y ER), y por tanto se necesita identificar el tipo de lenguaje para determinar su solución (el modelo). Este tema pretende mostrar características y propiedades para identificar cuando un lenguaje es o no regular.

1.1. Lema del Bombeo o Potencias para Conjuntos Regulares

Es un método para demostrar que ciertos lenguajes NO son regulares. Si L es regular y M = (Q,Σ,δ,q_0,F) es un AFD y L = L(M) con cierto número de estados n.
Consideremos $a_1, a_2, ...a_m, m \geq n$ (m símbolos) y sea $\widehat{\delta}(q_o, a_1a_2...a_i) = q_i$ para i = 1,2,...,m (necesariamente se pasa por un estado 2 o más veces). No es posible que todos los q_i sean distintos ya que solo hay n estados. Por lo tanto hay 2 enteros j y k, tal que $0 \leq j < k \leq m$, si q_m está en F, entonces $a_1, a_2, ...a_m$ esta en L(M) pero también lo está $a_1, a_2, ...a_j(a_{j+1}...a_k)^i a_{k+1}...a_m$ con $i \geq 0$, i es el número de veces que se repite el ciclo o lazo de q_j a q_k como muestra la figura 1.1 (es decir, se concatena i veces la subcadena $a_{j+1}...a_k$ y la cadena resultante sigue estando en el lenguaje)

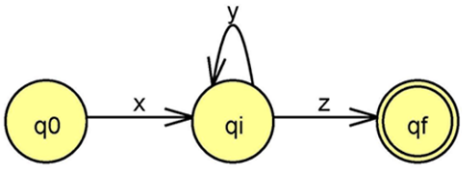

Figura 1.1: Bombeando la cadena $w^i = xy^iz$

Teorema 1. ***Lema del Bombeo*** [1]***:*** *Si L es un conjunto regular entonces hay una constante n tal que para w en L y $|w| \geq n$ podemos escribir $w = xyz$ tal que:*

1. *$y \neq \lambda$ o $|y| \geq 1$, la cadena a bombear y no es vacia, al menos tiene un símbolo.*
2. *$|xy| \leq n$, el tamaño del prefijo xy de w no es más grande que n*
3. *Para todo i entero, xy^iz pertenece a L, para cualquier valor entero mayor o igual al cero la cadena bombeada sigue perteneciendo al lenguaje.*

Es decir, siempre es posible encontrar una cadena no vacia y, no demasiado lejos del comienzo de w que se pueda "bombear"(repetir i veces) y la cadena resultante pertenece también al lenguaje L.
Se debe usar el lema para demostrar por contradicción que un Lenguaje L no es regular, de la siguiente forma:

1. Asumo que L es regular
2. Seleccionar un w adecuado que cumpla con las condiciones del lema (1 y 2) y que pertenezca a L (el lenguaje debe depender del valor n, constante del Lema).
3. Bombear y salir de L (conseguir una contradicción para la condición 3 del lema, es decir encontrar un solo valor de i para bombear la cadena y sacar el resultado del lenguaje L).
4. Al tener la contradicción, deducir que lo que se asume al principio es falso, por tanto el lenguaje L no es regular.

Nota: el x,y,z no son fijos, ni tienen longitudes fijas sólo están limitados por las condiciones (1 y 2) del lema y la definición del lenguaje.
Ejemplos: Dados los siguientes lenguajes desmostrar si son o no lenguajes regulares

- L = $\{ 0^j1^j \mid j \geq 0 \}$

  Se asume que L es un Lenguaje Regular, y se aplica el Lema del Bombeo. Se selecciona w = 0^n1^n, que cumple con que w = xyz, $|w| \geq n$ y $|xy| \leq n$
  Se va a bombear la cadena y que esta en 0^n por las condiciones del Lema, cumpliendo con la siguiente desigualdad $1 \leq |y| \leq n$. Por tanto, la cadena w quedaría de la siguiente forma:
  w = $0^{n-|y|}0^{|y|}1^n$, partiendo de que w = xyz, donde $0^{n-|y|}$ es la cadena x, $0^{|y|}$ es la cadena y, y el resto 1^n es la cadena z
  Para bombear solo hay que concatenar tantas veces como indique la potencia de la cadena ($w^i = xy^iz$), quedando de la siguiente forma: $w^{\mathbf{i}} = 0^{n-|y|}0^{|y|*\mathbf{i}}1^n$
  Ahora debe conseguirse al menos un valor de i para el cual la cadena resultante no esté en el lenguaje. Se considera un valor de i = 2: $0^{n-|y|}0^{\mathbf{2}*|y|}1^n = 0^{n-|y|+\mathbf{2}*|y|}1^n = 0^{n+|y|}1^n$
  Para todos los valores posible de |y|, $1 \leq |y| \leq n$, el w bombeado sale del lenguaje L, ya que el número de ceros siempre es mayor que el número de unos ($1+n \leq |y|+n \leq n+n$). Esto es una contradicción ya que el número de ceros debe ser igual al número de unos para todo i ≥ 0, por tanto la suposición es falsa y por tanto el lenguaje L NO es Regular.

[1] Bombear significa repetir i veces la concatenación de una cadena, generalmente se indica el bombeo con el exponente de la potencia de la subcadena, por ejemplo a^3 = aaa

- L = { XX | X en $(a+b)^*$ }

  Se asume que L es un Lenguaje Regular, y se aplica el Lema del Bombeo. Se selecciona w = a^nba^nb, que cumple con que w = xyz, $|w| \geq n$ y $|xy| \leq n$
  Se va a bombear la cadena y que esta en a^n por las condiciones del Lema, cumpliendo con la siguiente desigualdad $1 \leq |y| \leq n$. Por tanto, la cadena w quedaría de la siguiente forma:
  w = $a^{n-|y|}a^{|y|}ba^nb$, partiendo de que w = xyz, donde $a^{n-|y|}$ esta en la cadena x, $a^{|y|}$ esta en la cadena y, y el resto ba^nb esta en la cadena z
  Para bombear solo hay que multiplicar por un valor de i a la potencia de la cadena ($w^i = xy^iz$), quedando de la siguiente forma: $w^{\mathbf{i}} = a^{n-|y|}a^{|y|*\mathbf{i}}ba^nb$
  Ahora debe conseguirse al menos un valor de i para el cual la cadena resultante no esté en el lenguaje. Se considera un valor de i = 0: ($w^0 = xy^0z$) por tanto $w^{\mathbf{0}} = a^{n-|y|}a^{|y|*\mathbf{0}}ba^nb = a^{n-|y|}ba^nb$
  Para todos los valores posible de |y|, $1 \leq |y| \leq n$, el w bombeado sale del lenguaje L, ya que la primera subcadena X = $a^{n-|y|}b$ siempre es diferente a la segunda subcadena X = a^nb, por el número de símbolos a (los rangos son: $n-1 \leq n-|y| \leq n-n$). Esto es una contradicción ya que ambas subcadenas X deben ser iguales, para todo i ≥ 0, por tanto la suposición inicial es falsa y por tanto el lenguaje L NO es Regular.

1.2. Propiedades de clausura de los Lenguajes regulares

También llamadas operaciones cerradas, son aquellas que se aplican sobre los lenguajes regulares y el resultado sigue siendo un lenguaje regular. Estas propiedades describen el comportamiento de estos lenguajes, permite conocer construcciones para operar con sus modelos y se pueden utilizar para demostrar que un lenguaje es regular o no es regular

Teorema 1.2.1. *Los conjuntos regulares son cerrados bajo unión, concatenación y Clausura de Kleene. También son cerrados bajo:*

- *Complemento*
- *Intersección*
- *Diferencia*
- *Reflexión o reflejo*
- *Sustituciones, homomorfismo y homomorfismo inverso*
- *Cociente de conjuntos arbitrarios*

Para las operaciones de unión, concatenación y clausura de Kleene son cerradas por definición de las expresiones regulares. Para el resto de las operaciones es necesario ver como se aplican sobre conjuntos regulares:

1. **Los conjuntos regulares son cerrados bajo complemento**
   Sea L un lenguaje regular y L $\subseteq \Sigma^*$, entonces también Σ^* - L es regular. Sea L tal que L = L(M) con M = (Q,Σ, δ, q_0,F) donde M es AFD y sea L $\subseteq \Sigma^*$, se asume el mismo alfabeto

para $\overline{L}$ y se construye el autómata M' tal que L(M') = $\overline{L}$. Asi M' = (Q,Σ, δ, q_0,Q-F), M' acepta a w si y solo si w está en Σ^* - L. La figura 1.2 muestra un AFD y su complemento, lo cual se logra complementando el conjunto F de M, es decir en M' se acepta lo que en M no se aceptaba y M' no acepta lo que aceptaba M.

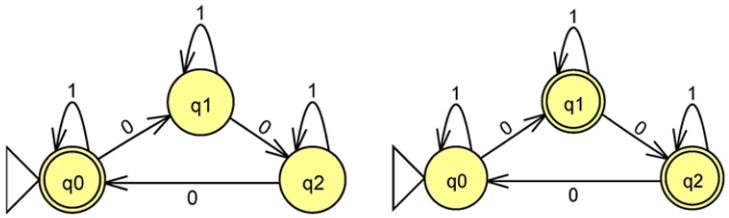

Figura 1.2: Complemento de un AFD: Cadenas binarias que no tienen un número de 0s múltiplo de 3

2. **Los conjuntos regulares son cerrados bajo intersección**

Por la ley de Morgan [2], la intersección se expresa en términos de unión y complemento por lo tanto la operación de la intersección es cerrada. Sin embargo, aplicar tantos operadores (unión y tres complementos) es ineficiente, existe una alternativa para realizar la operación sobre el modelo de AF denominada Construcción de producto [3]:
Construcción de producto: Sea $L_1 = L(M_1), M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ y $L_2 = L(M_2), M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, se construye M' = $(Q_1 x Q_2, \Sigma, \delta, [q_1, q_2], F_1 x F_2)$, $\forall p_1$ en Q_1, p_2 en Q_2 y a en Σ, hacer $\delta([p_1, p_2], a) = [\delta_1(p_1, a), \delta_2(p_2, a)]$. La construcción se aplica en: la Figura 1.3 muestra el AFD de los lenguajes L_1 (número impar de ceros) y L_2 (número par de unos), la Figura 1.4 muestra el autómata de la intersección de los dos lenguajes L_1 y L_2 a través de sus Autómatas determinísticos, y el cuadro 1.1 muestra la tabla con la construcción de producto.

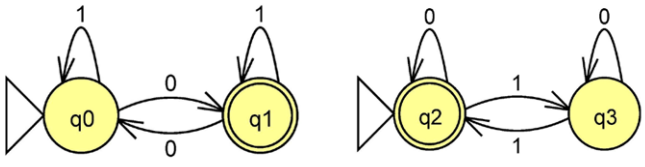

Figura 1.3: M_1 y M_2 reconocen respectivamente a L_1 (número impar de ceros) y L_2 (número par de unos)

3. **Los conjuntos regulares son cerrados bajo diferencia**

[2] Ley de Morgan / Diagramas de Venn: $L_1 \bigcap L_2 = \overline{\overline{L_1} \bigcup \overline{L_2}}$
[3] debido a que se aplica el producto cartesiano de los conjuntos Q y F de cada autómata

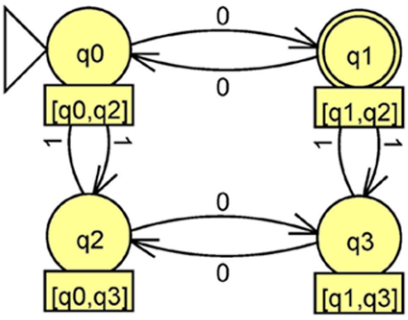

Figura 1.4: AFD que reconoce el lenguajes de las cadenas binarias con un número impar de ceros y un número par de unos (intersección)

Cuadro 1.1: Tabla de la Construcción de producto para obtener el AFD que reconoce el lenguajes de las cadenas binarias con un número impar de ceros y un número par de unos (intersección)

Etiqueta en Fig 1.4	δ Producto	0	1
q_0	$\rightarrow$ [q_0,q_2]	[q_1,q_2]	[q_0,q_3]
q_1	* [q_1,q_2]	[q_0,q_2]	[q_1,q_3]
q_2	[q_0,q_3]	[q_1,q_3]	[q_0,q_2]
q_3	[q_1,q_3]	[q_0,q_3]	[q_1,q_2]

L_1 y L_2 son lenguajes regulares entonces $L_1 - L_2$ también los son (conjunto de cadenas que pertenecen a L_1 y no pertenecen a L_2). Por teoría de conjuntos, la diferencia[4] puede expresarse en terminos de intersección y complemento las cuales son cerradas por tanto la operación de diferencia es cerrada también.

4. **Los conjuntos regulares son cerrados bajo reflexión o reverso**

Dada una cadena w = $a_1, a_2, ...a_n$ su reflexión se escribe al revés $a_n, a_{n-1}, ...a_1$. Se denota w^R a la refleja (reverso) de w. Su construcción se puede dar a partir del AF y de su ER:

- Dado un Autómata A, se procede de la siguiente forma (ver figura 1.5):

  - Invertir todos los arcos del diagrama de A. Si δ(q,a) = p, entonces δ_N(p,a) = q
  - Convertir el estado inicial de A en el único de aceptación
  - Crear un nuevo estado inicial p_0, con transiciones λ a todos los estados de aceptación de A. En el caso donde |F| = 1 entonces solo bastaría con colocar dicho estado como el inicial.

  La figura 1.5 muestra el AFD del lenguaje denotado por la expresión regular (bb*a)*aa*ba* y su reflejo.

[4] Diferencia de conjunto $L_1 - L_2 = L \bigcap \overline{L_2}$

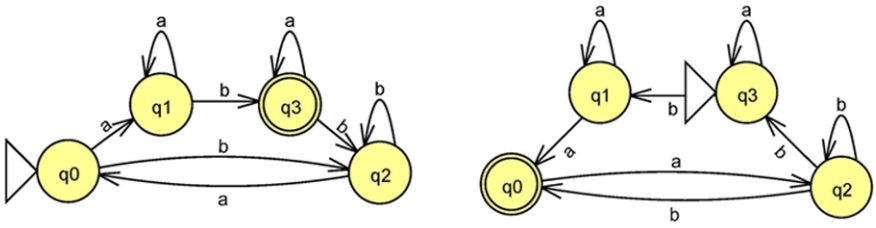

Figura 1.5: AFD que reconce a (bb*a)*aa*ba* y el AFD con la operación del reflejo

- Dado la Expresión regular que denota al lenguaje, se procede de la siguiente forma:
  - Caso Base:
    $L(E^R) = (L(E))^R$, $E^R = E$, $\phi^R = \phi$, $a^R = a$.
  - Paso Inductivo:
    E = $E_1 + E_2$, entonces $E^R = E_1^R + E_2^R$
    E = $E_1 E_2$, entonces $E^R = E_2^R . E_1^R$
    E = E_1^*, entonces $E^R = (E_1^R)^*$

  La figura 1.5 muestra la expresión regular (bb*a)*aa*ba* y su reflejo sería a*ba*a(ab*b)*

5. **Los conjuntos regulares son cerrados bajo sustituciones, homomorfismo y homomorfismo inverso**

   La sustitución (en conjuntos regulares) consiste en reemplazar o mapear los símbolos de un lenguaje a un subconjunto de otro lenguaje (generalmente denotado por expresiones regulares). Una sustitución f es un mapeo de un alfabeto Σ en un subconjunto de Δ^* para algun alfabeto Δ. La función f también puede extenderse a cadenas (por concatenación):

   $a)$ $f(\lambda) = \lambda$

   $b)$ $f(xa) = f(x)f(a)$

   La función f también puede extenderse a lenguajes: $f(L) = \bigcup_{xenL} f(x)$
   Por ejemplo: Suponga las siguientes funciones de sustitución f(0) = a y f(1) = b^*. Los alfabetos Σ = {0,1} y Δ' = {a,b^*}, donde $\Delta' \subseteq \Delta^*$, Δ = {a,b}. Se aplica sobre cadenas en Σ^*:
   f(010) = f(01).f(0) = f(01)a = f(0).f(1).a = ab^*a
   f(1101) = b^* b^* a b^* = b^* a b^*
   Dado un lenguaje L = 0^* (0+1) 1^* , se aplica f(L):
   f(L) = a^* (a + b^*) $(b^*)^*$ = a^* (a + b^*) b^*

   El homomorfismo, o sustitución h, es un caso particular de sustitución sobre funcio-

nes (inyectiva y sobreyectiva [5]) que mapean por cada símbolo a en Σ una sóla cadena en Δ^*. También es llamado sustitución uno a uno, su definición es la siguiente:

h(x) = { h(x) | h(x) en Δ^*, x en Σ }
Por definición $h(\lambda) = \lambda$.

La imagen homomorfica inversa de un lenguaje L y de una cadena se define como: $h^{-1}(L)$ = { x | h(x) en L }
$h^{-1}(w)$ = { x | h(x) = w }
Por ejemplo: Dados los alfabetos Σ = { 0,1 } y Δ = { a, b }, donde se define un conjunto Δ' = {aa, aba} $\subseteq \Delta^*$, Se define el homomorfismo h(0) = aa y h(1) aba, se tienen las siguientes cadenas y lenguaje:

- h(010) = aaabaaa
- h^{-1}(abaaa) = 10
- $L_1 = (01)^*$, $h(L_1) = (aaaba)^*$
- L_2 = (ab + ba)*a, $h^{-1}(L_2) = 1$, Se trata de buscar las cadenas en Σ a las cuales se les aplica el homomorfismo h y estan en L_2 = {a, aba, baa, ababa, abbaa, baaba, babaa,...}.

Los conjuntos regulares son cerrados bajo homomorfismo y homomorfismo inverso por tanto se puede aplicar sobre los autómatas finitos. La construcción dada una función homomorfica, se aplica sobre un AFD M = (Q,Σ,δ,q_0,F) que acepta L y sea h un homomorfismo de Δ en Σ^* ($h : \Delta \to \Sigma^*$) se construye una M' = (Q,Δ,δ',q_0,F) donde δ'(q,a) = δ(q,h(a)) para q en Q y a en Δ.

Ejemplo: Σ = { a,b } y Δ = { 0, 1 }, L = (ab+ba)*a
h: $\Delta \to \Sigma^*$, h(0) = aa y h(1) = aba

La máquina M = ($\{q_0, q_1, q_2, q_3\}$,Σ,δ,q_0,$\{q_1\}$) de la figura 1.6 reconoce a L. La máquina de la figura 1.7 muestra es M' = ($\{q_0, q_1, q_2, q_3\}$,Δ,δ',q_0,$\{q_1\}$) con $h^{-1} : \Sigma^* \to \Delta$.
Cada transición aplica la construcción, por ejemplo para $\delta'(q_0,0) = \delta(q_0,h(0)) = \delta(q_0,aa) = \delta(\delta(q_0,a),a) = \delta(q_1,a) = q_3$

Se puede usar cualquiera de estas propiedades de clausura para demostrar que un lenguaje NO es regular con el lema del bombeo. Por ejemplo si se quiere demostrar que un lenguaje L no es regluar y es dificil aplicar el lema del bombeo entonces podemos aplicarlo para el complemento, bajo sustitución, etc. que no es regular y L tampoco lo será pues son operaciones cerradas. Las operaciones cerradas se utilizan para demostrar que algo es regular o que no es regular. El homomorfismo se aplica para transformar algo que no se conoce en algo conocido que se sabe que no es regular, por ejemplo: Demostrar

[5] Dado una función h:A $\to$ B, h es *inyectiva* si $a_1 \neq a_2$, $h(a_1) \neq h(a_2)$; h es *sobreyectiva* si todo elemento de B es imagen de alguno de A

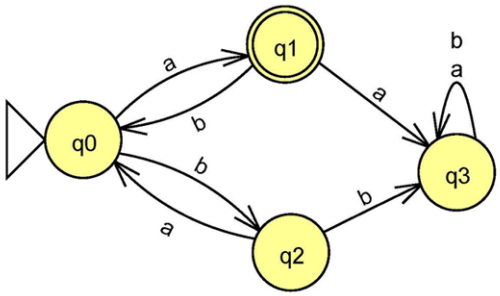

Figura 1.6: AFD que reconoce (ab + ba)*a

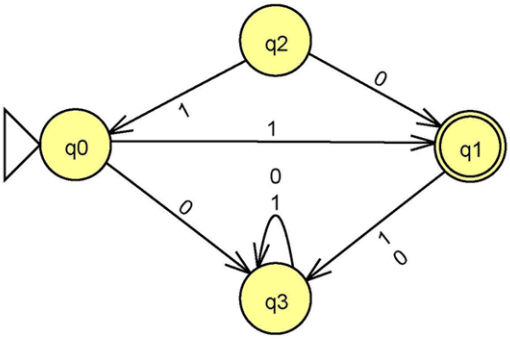

Figura 1.7: AFD con homorfismo inverso h^{-1} de la Figura 1.6

que $\{a^n ba^n | n \geq 1\}$ no es regular sabiendo que $\{0^n 1^n | n \geq 1\}$ no es regular.
$\{0^n 1^n | n \geq 1\}$ no es regular por el Lema del Bombeo. Para transformar $a^n ba^n$ en $0^n 1^n$ se realiza una reducción con operaciones cerradas del lenguaje L = $a^n ba^n$ a $0^n 1^n$, de la siguiente forma:
Σ = { a,b }, Δ = { 0, 1 } y Δ' = { x, y, z }
$a^n ba^n = a^n baa^{n-1}$
Se define el homomorfirmo $h_1 : \Delta' \rightarrow \Sigma^*$, asi: $h_1(x)$ = a, $h_1(y)$ = ba y $h_1(z)$ = a.
Al aplicar el homomorfismo inverso h_1^{-1} queda $h_1^{-1}(L) = (x+z)^n y(x+z)^{n-1} \rightarrow h_1^{-1}(L) \bigcap x^* yz^* = x^n yz^{n-1}$.
Se define el homomorfismo $h_2 : \Delta' \rightarrow \Delta^*$, asi: $h_2(x) = 0$, $h_2(y) = 1$ y $h_2(z) = 1$. Al aplicarlo $h_2(h_1^{-1}(L)) = 0^n 1^n$, ya que $h_2((x+z)^n y(x+z)^{n-1}) = 0^n 11^{n-1} = 0^n 1^n$.
Como se han aplicado solamente operaciones cerradas sobre los lenguajes regulares sobre un conjunto que se supone es regular $a^n ba^n$, y se ha llegado a $0^n 1^n$ que se sabe que No es regular, entonces es falso el supuesto de $a^n ba^n$ y por tanto se demuestra que NO es regular.

6. **Los conjuntos regulares son cerrados bajo Cociente de conjuntos arbitrarios**

1.3. Algoritmos de decisión sobre conjuntos regulares

Se debe recordar que el lenguaje típico que interesa es infinito, pues para los lenguajes finitos (que son regulares) no hay ninguna complicación para resolver problemas. Por tanto, hacer una pregunta sobre alguna propiedad de un conjunto infinito realmente no es posible, a menos que se realice sobre su *representación finita* (AF o ER). Las siguientes propiedades se pueden expresar como problemas de decisión:

1. Vacuidad: ¿El lenguaje L es vacío?
2. Pertenencia: ¿La cadena w pertenece al lenguaje L?
3. Equivalencia: ¿Dos autómatas finitos reconocen el mismo lenguaje L?

Las preguntas anteriores se refieren al lenguaje, pero se trata de buscar un **algorimo decidible** [6] que dé respuesta (si o no) al problema de decisión. Por tanto no puede aplicarse sobre el lenguaje infinito (expresado como conjunto) y se utiliza en su lugar los modelos finitos del lenguaje (ER y AF para lenguajes regulares). Para eso se muestran como pueden resolverse estos problemas con un algoritmo decidible al utilizar el autómata finito o la expresión regular que reconocen o denotan al lenguaje.

1.3.1. Vacuidad: ¿El lenguaje L es vacío?

Se evalua la posibilidad de que L = ϕ sobre la representación de L:

- Si es un Autómata finito
  Si existe algun camino desde q_0 a un estado de aceptacion el lenguaje no es vacio. Caso contrario (todos los estados de aceptación estan desconectados) entonces el lenguaje L es vacio. En teoría de grafos existen varios algoritmos que permiten detectar caminos entre un par de nodos (estado inicial y algunos de los estados finales) del grafo.
- Si es una Expresión regular
  - Convertir la ER en un AFND con transiciones nulas y luego transformalo con la construcción de subconjuntos en un AFD y proceder como se indica anteriormente.
  - Aplicar las siguientes reglas recursivas a la ER:
    Base: denota el lenguaje vacio, no denotan al lenguaje vacio
    Inductivo: R es una expresión regular y si:
    1. R = $R_1 + R_2$, entonces L(R) es vacio si y solo si L(R_1) y L(R_2) son vacio
    2. R = R_1R_2, entonces L(R) es vacio si y solo si L(R_1) es vacio o L(R_2) es vacio
    3. R = R_1^*, entonces L(R) no es vacio ya que siempre incluye por lo menos a la cadena vacia
    4. R = (R_1), entonces L(R) es vacio si y solo si R_1 es vacio.

[6] Un algoritmo decidible es un algoritmo que funciona para toda instancia del problema y que siempre termina para dar una respuesta

1.3.2. Pertenencia: ¿La cadena w pertenece al lenguaje L?

Dada una cadena w en Σ^* y un lenguaje L, cómo determinar si ¿w pertenece a L?. La cadena w se representa explicitamente pues es de tamaño finito. El lenguaje L se representa a través de una ER o de un AF, que son modelos finitos de lenguajes regulares. Si se tiene el AFD que reconoce el lenguaje se simula el funcionamiento al recibir la cadena w comenzando desde q_0, si finaliza en un estado de aceptación la respuesta es si y en caso contrario la respuesta es no (ver en el tema 1 el tópico de programación basada en autómatas). Si se trata de un AFND-λ se puede aplicar el algoritmo de backtracking para llevar las multiples opciones en la ejecución, o también hacer un algoritmo recursivo de δ, o bien hacer la transformación a un AFD equivalente para ejecutarlo deterministicamente.

1.3.3. Equivalencia: ¿Dos autómatas finitos reconocen el mismo lenguaje L?

Comprobación de estados equivalentes Dos estados p y q son equivalentes si para todo w en Σ^*, $\widehat{\delta}$(p,w) y $\widehat{\delta}$(q,w) estan en un estado de aceptación (no necesariamente el mismo estado). Dos estados p y q son estados equivalentes si llego a un estado final procesando la misma cadena w en ambos casos, por lo que decimos que no hay ninguna distinción haber estado en q o haber estado en p. Si dos estados no son equivalentes decimos que son distinguibles. Entonces p es distinguible de q si existe por lo menos una cadena w tal que $\widehat{\delta}$(p,w) es un estado de aceptación y $\widehat{\delta}$(q,w) no es un estado de aceptación y viceversa.

Algoritmo para buscar pares de estados distinguibles recursivamente .

Precondición = {Debe aplicarse sobre un AFD}
Estructura de datos = Matriz de |Q|-1 x |Q|-1 (eliminar la diagonal). Cada posición de la matriz contiene una X para indicar que son pares distinguible (vacío en caso contrario) y además contiene una lista de pares de estados de los cuales depende la condición de distinguible.

1. Para p en F y q en Q-F marque (p,q) como distinguible. Recordar que $\widehat{\delta}(p, \lambda) = p$ por definición para todo p en Q, por tanto si p esta en F y q esta en Q-F entonces son distinguibles porque la cadena vacía λ los distingue.
2. Para cada par (p,q) en FxF o en (Q-F)x(Q-F) no marcado como distinguible, hacer:
   a) Si para algún a en Σ, el par (δ(p,a),δ(q,a)) esta marcado entonces marque (p,q). Recursivamente marque todos los pares en la lista asociada a (p,q).
   b) En caso contrario (no hay ningún par marcado), entonces colocar (p,q) en la lista de (δ(p,a),δ(q,a)) excepto cuando δ(p,a) = δ(q,a) para todo a en Σ.

Ejemplo: Dado el AFD de la Figura 1.8 se muestra la tabla de estados distinguibles en el cuadro 1.2. La ejecución del algoritmo es la siguiente:

1. Paso 1 del algoritmo: F = {c} y Q-F = {a,b,d,e,f,g,h}. Todos los pares FxQ-F son distinguibles.
2. Paso 2 del algoritmo: Para cada par (p,q) en FxF o en (Q-F)x(Q-F) no marcado como distinguible. [7]

[7] El orden para revisar las celdas de la tabla en el paso 2 del algoritmo no esta establecido, debe revisarse todos los pares. Cualquier orden debe dar el mismo resultado la diferencia estaría en la lista asociada

- (a,b): 1 distingue, δ(a,1) = f y δ(b,1) = c, (f,c) marcado
- (a,d): 0 distingue, δ(a,0) = b y δ(d,0) = c, (b,c) marcado
- (b,d): cualquier simbolo en Σ (0 y 1) distingue, por ejemplo 0: δ(b,0) = g y δ(d,0) = c, (g,c) marcado
- (a,e): ningun símbolo de Σ distingue. Se agrega en la casilla del par no revisado como distinguible (b,h) (ya que δ(a,0) = b y δ(e,0) = h) el primer nodo de su lista el par (a,e). Cuando este par (b,h) se marque dintingible entonces se revisa su lista asociada y se marcaría el par (a,e) (Paso 2.b del algoritmo, ver tabla 1.2).
- (b,e): 1 distingue, δ(b,1) = c y δ(e,1) = f, (f,c) marcado
- (d,e): 0 distingue, δ(d,0) = c y δ(e,0) = h, (h,c) marcado
- (a,f): 0 distingue, δ(a,0) = b y δ(f,0) = c, (b,c) marcado
- (b,f): cualquier simbolo en Σ (0 y 1) distingue, por ejemplo 0: δ(b,0) = g y δ(f,0) = c, g no ϵ F y c ϵ F
- (d,f): ningun símbolo de Σ distingue, pues son el mismo valor en ambos casos (δ(d,1) = δ(f,1) = g y δ(d,0) = δ(f,0) = c), excepción que no crea lista asociada (Paso 2.b del algoritmo).
- (e,f): 0 distingue, δ(e,0) = h y δ(f,0) = c, (h,c) marcado
- (a,g): 1 distingue, δ(a,1) = f y δ(g,1) = e, (e,f) esta marcado como distinguible por tanto (a,g) también (2.a del algoritmo).
- (b,g): 1 distingue, δ(b,1) = c y δ(g,1) = e, (e,c) marcado
- (d,g): 0 distingue, δ(d,0) = c y δ(g,0) = g, (g,c) marcado
- (e,g): 1 distingue, δ(e,1) = f y δ(g,1) = e, (e,f) esta marcado como distinguible por tanto (e,g) también (paso 2.a del algoritmo).
- (f,g): 0 distingue, δ(f,0) = c y δ(g,0) = g, (g,c) marcado
- (a,h): 1 distingue, δ(a,1) = f y δ(h,1) = c, (f,c) marcado
- (b,h): ningun símbolo de Σ distingue, pues son el mismo valor en ambos casos (δ(b,1) = δ(h,1) = c y δ(b,0) = δ(h,0) = g), excepción que no crea lista asociada (Paso 2.b del algoritmo).
- (d,h): 0 distingue, δ(d,0) = c y δ(h,0) = g, (g,c) marcado
- (e,h): 1 distingue, δ(e,1) = f y δ(h,1) = c, (f,c) marcado
- (f,h): cualquier símbolo en Σ (0 y 1) distingue, por ejemplo 0: δ(f,0) = c y δ(h,0) = g, (g,c) marcado
- (g,h): 1 distingue, δ(g,1) = e y δ(h,1) = c, (e,c) marcado

Los pares de estados que no pueden marcarse al terminar el algoritmo no se pueden distinguir y por tanto son pares equivalente. En la tabla 1.2 se observan los pares (a,e), (d,f), (b,h) como equivalentes.

Utilidad de la tabla de estados distinguibles

- Comprobación de la equivalencia de lenguajes regulares. Dado 2 lenguajes L y M se llevan sus representaciones a un AFD. Se aplica el algoritmo para determinar los estados distinguibles a la unión de los estados de los dos autómatas. Luego se prueba si los estados iniciales de ambos AFD originales son equivalentes. Si son equivalentes entonces L = M y en

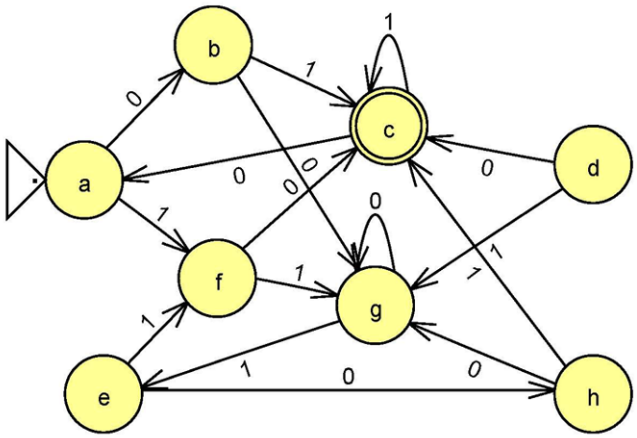

Figura 1.8: AFD para aplicar el algoritmo de estados distinguibles

Cuadro 1.2: Tabla de estados distinguibles del AFD de la figura 1.8

b	X						
c	X	X					
d	X	X	X				
e	$\rightarrow$ //	X	X	X			
f	X	X	X	$\rightarrow$ //	X		
g	X	X	X	X	X	X	
h	X	$\rightarrow$ (a,e)$\rightarrow$ //	X	X	X	X	X
	a	b	c	d	e	f	g

caso contrario L $\neq$ M. Por ejemplo ver figura 1.9 donde se muestra 2 autómatas para determinar si son equivalentes se construye la tabla de estados distinguibles del cuadro 1.3.

Los estados iniciales son A y C, el par (A,C) no es distinguible por lo tanto es equivalente, es decir ambos autómatas reconocen el mismo lenguaje [8]. También es posible determinar si dos autómatas son equivalentes (aceptan el mismo lenguaje) por las propiedades de los lenguajes regulares: Sean M_1 y M_2 dos AFD que aceptan L_1 y L_2 respectivamente, se puede construir un M_3 que acepte $(L_1 \bigcap \overline{L_2}) \bigcup (\overline{L_1} \bigcap L_2)$ si L(M3) = ϕ, entonces $L_1 = L_2$. Otra forma sería minimizar ambos autómatas y obtener como resultado el mismo autómata, ya que el mínimo es único.

- Minimización de un AFD
  El AFD mínimo es único para cada lenguaje [9]. Existe un algoritmo para construir

[8] Recordar que la definición de equivalentes quiere decir que dos estados son equivalentes si para w en $\Sigma*$ el comportamiento es el mismo, es decir si se acepta o no se acepta en ambos

[9] Conjunto de los AFDs equivalente que reconcen a un lenguaje es un conjunto bien ordenado

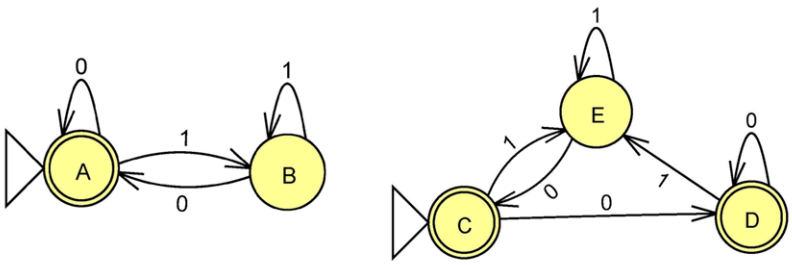

Figura 1.9: Algoritmo de estados distinguibles para determinar equivalencia de autómatas

Cuadro 1.3: Tabla de estados distinguibles de los autómatas de la figura 1.9

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

el AFD mínimo a partir de la tabla de estados dinguibles:

1. Eliminar cualquier estado que no sea accesible desde el estado inicial
2. Se agrupan los estados equivalentes, es decir se crean a partir de las particiones.
3. Construir el autómata mínimo utilizando las particiones como estados y las transiciones entre las particiones considerando uno de los estados de cada partición para todo a en Σ.

Sea un AFD M = (Q,Σ,δ,q_0,F) tal que L(M) es un conjunto regular. Se establece la relación de equivalencia $\equiv$ (reflexiva, simétrica y transitiva) definida sobre los estados de M, tal que p $\equiv$ q si y solo si, para todo w en Σ^*, $\widehat{\delta}(p,w)$ está en F, si y solo si $\widehat{\delta}(q,w)$ está en F.
En el ejemplo de la Figura 1.8 las clases de equivalencia son: [e,a], [h,b], [f,d], [c] y [g], el AFD mínimo puede verse en la Figura 1.10. En el ejemplo de la Figura 1.9 las clases de equivalencia son: [A,D,C] y [B,E] y el autómata determinista mínimo es el primer autómata [10].

1.4. Preguntas y respuestas, ejercicios resueltos y propuestos

1.4.1. Preguntas y respuestas

1. Porqué el lema del bombeo no sirve para demostrar que un lenguaje si es regular

[10] Es decir, en la Figura 1.9 se trata de minizar el segundo autómata, quedan las particiones [C,D] y [E], y su construcción daría un autómata con la misma estructura que el primer autómata, donde los estados A = [C,D] y B = [E] tendría entre ellos las mismas transiciones. De esta forma también se demuestra que son equivalente, se minimizan ambos y se comparan las estructura del AFD

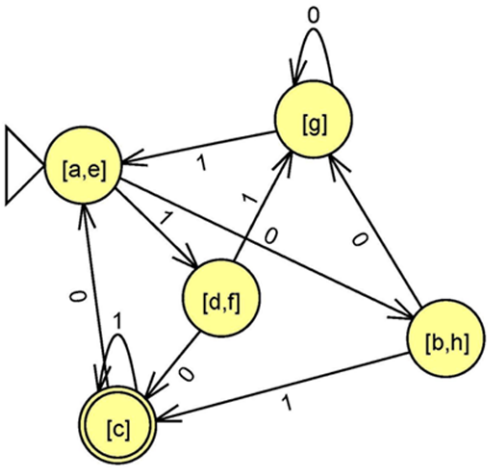

Figura 1.10: AFD Mínimo del lenguaje que reconoce la figura 1.8

2. Cómo demuestro que un lenguaje si es regular
3. Qué logro al demostrar la regularidad de un lenguaje
4. Para qué utilizo las propiedades de clausura de los lenguajes regulares
5. Utilidad de minimizar un autómata

1.4.2. Ejercicios propuestos

1. Contestar verdadero (V) o falso (F):

   a) Si un autómata *finito* pudiera tener *infinitos* estados entonces podría reconocer lenguajes NO regulares.

   b) El lenguaje L generado por el alfabeto binario tal que sus cadenas son de la forma 0^n1^k donde n,k ≥ 0, es un lenguaje NO regular.

   c) Si la aplicación del Lema del Bombeo para lenguajes regulares falla (no hay contradicción para todas las cadenas) entonces el Lenguaje es regular.

   d) Los autómatas finitos con salida pueden procesar lenguajes NO regulares.

   e) Todo subconjunto de un lenguaje regular es regular

   f) Si L es regular, también lo es el lenguaje formado por { w, w en L y w^R en L }

   g) La unión o intersección de dos lenguajes NO regulares no puede ser regular

   h) Si L es cualquier lenguaje, no necesariamente regular, con un alfabeto de un solo símbolo entonces L* es regular.

2. Demuestre si los siguientes lenguajes son o no regulares:

   - { ww^R | w en (a+b)* } R = reverso
   - { 0^n1^m | n $\leq$ m }

- { $1^n w$ | w en (0+1)*, n $\geq$ 1 }
- { xx | x en (0+1)* }
- { 0^j | j = i2, i $\geq$ 0 }
- {$a^i b^j a^k$ | k >i + j ; k,j,i $\geq$ 0 }
- { x | x en (a + b + c)* y N_a(x) <N_b(x) y N_a(x) <N_c(x) } Donde el N_a(x) es el número de veces que el símbolo a aparece en la cadena x
- { w | w en (a + b)* y tiene una cantidad de a's divisible por 3 }
- { w | w en (a + b)* y el número de a´s = número de b´s }

3. Dado dos lenguajes regulares L_1 y L_2 diga y justifique si es posible tener un algoritmo para las siguientes preguntas:

   a) Determinar si existe alguna cadena w en Σ^* que no pertenezca ni a L_1 ni a L_2.

   b) ¿L_1 y L_2 son uno el complemento del otro?

   c) ¿$L_1^* = L_2$?

   d) ¿L_1 = Prefijo(L_2)?

   e) ¿$L_1 = L_2$?

4. Explique óomo encontraría un algoritmo de decisión para determinar, si dado dos AF M_1 y M_2 ¿existen cadenas que no sean aceptadas por ninguno de ellos?

5. Escriba un algoritmo de "decisión"que dada una expresión regular R para denotar a un lenguaje regular L en el alfabeto, determinar si ¿ L = Σ^* ?

6. Encuentre los autómatas finitos con un número de estados mínimo (usando el algoritmo de la tabla de estados distinguibles) que acepten el mismo lenguaje de cada uno de los autómatas de la figura 1.11

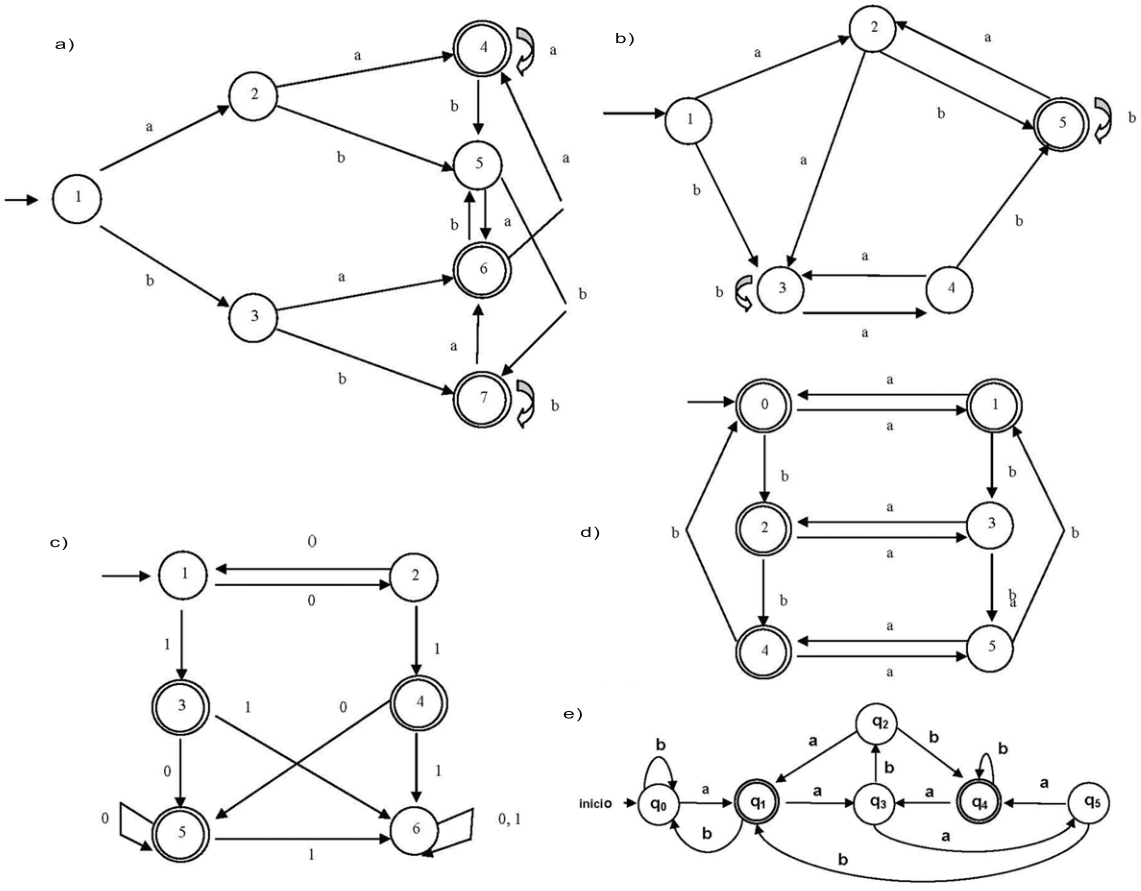

Figura 1.11: Diagramas de transición de AFD para minimizar