

Teoría de la Computación

Lenguajes, autómatas, gramáticas

Rodrigo De Castro Korgi

Ph.D. en Matemáticas

University of Illinois, U.S.A.

Departamento de Matemáticas

Universidad Nacional de Colombia, Bogotá

Teoría de la Computación

Lenguajes, autómatas, gramáticas

© UNIVERSIDAD NACIONAL DE COLOMBIA
FACULTAD DE CIENCIAS

Juan Manuel Tejeiro, Decano
Natalia Ruiz, Vicedecana Académica

Gustavo Rubiano, Director de Publicaciones

© Rodrigo De Castro Korgi
Profesor Asociado
Departamento de Matemáticas

Primera edición, 2004
Diagramación en \LaTeX realizada por el autor

Impresión:
UNIBIBLOS
Universidad Nacional de Colombia
Bogotá D.C., 2004

Índice general

Prólogo	1
Introducción. ¿Qué es la Teoría de la Computación?	3
1. Alfabetos, cadenas y lenguajes	5
1.1. Alfabetos y cadenas	5
1.2. Concatenación de cadenas	7
1.3. Potencias de una cadena	8
1.4. Longitud de una cadena	8
1.5. Reflexión o inversa de una cadena	9
1.6. Subcadenas, prefijos y sufijos	9
1.7. Lenguajes	10
1.8. Operaciones entre lenguajes	11
1.9. Concatenación de lenguajes	12
1.10. Potencias de un lenguaje	14
1.11. La clausura de Kleene de un lenguaje	14
1.12. Reflexión o inverso de un lenguaje	17
1.13. Lenguajes regulares	18
1.14. Expresiones regulares	19
2. Autómatas finitos	25
2.1. Autómatas finitos deterministas (AFD)	25
2.2. Diagrama de transiciones de un autómatas	28
2.3. Diseño de autómatas	30
2.4. Autómatas finitos no-deterministas (AFN)	33
2.5. Equivalencia computacional entre los AFD y los AFN	38
2.6. Autómatas con transiciones λ (AFN- λ)	43

2.7.	Equivalencia computacional entre los AFN- λ y los AFN . . .	47
2.8.	Teorema de Kleene. Parte I	49
2.9.	Ejemplos de la parte I del Teorema de Kleene	52
2.10.	Lema de Arden	55
2.11.	Teorema de Kleene. Parte II	57
2.12.	Ejemplos de la parte II del Teorema de Kleene	58
3.	Otras propiedades de los lenguajes regulares	63
3.1.	Lema de bombeo	63
3.2.	Propiedades de clausura	67
3.3.	Propiedades de clausura para autómatas	69
3.4.	Homomorfismos \bowtie	72
3.5.	Imagen inversa de un homomorfismo \bowtie	74
3.6.	Algoritmos de decisión	75
4.	Lenguajes y gramáticas independientes del contexto	81
4.1.	Gramáticas generativas	81
4.2.	Gramáticas independientes del contexto	82
4.3.	Árbol de una derivación	88
4.4.	Gramáticas ambiguas	91
4.5.	Gramáticas para lenguajes de programación	94
4.6.	Gramáticas para lenguajes naturales \bowtie	96
4.7.	Gramáticas regulares	98
4.8.	Eliminación de las variables inútiles	102
4.9.	Eliminación de las producciones λ	107
4.10.	Eliminación de las producciones unitarias	110
4.11.	Forma Normal de Chomsky (FNC)	113
4.12.	Forma Normal de Greibach (FNG) \bowtie	120
4.13.	Lema de bombeo para LIC	125
4.14.	Propiedades de clausura de los LIC	130
4.15.	Algoritmos de decisión para GIC	135
5.	Autómatas con pila	143
5.1.	Autómatas con Pila Deterministas (AFPD)	143
5.2.	Autómatas con pila no-deterministas (AFPN)	150
5.3.	Aceptación por pila vacía	154
5.4.	Autómatas con pila y LIC. Parte I.	157
5.5.	Autómatas con pila y LIC. Parte II. \bowtie	160

6. Máquinas de Turing	167
6.1. Máquinas de Turing como aceptadoras de lenguajes	167
6.2. Subrutinas o macros	174
6.3. Máquinas de Turing como calculadoras de funciones	176
6.4. Máquinas de Turing como generadoras de lenguajes	179
6.5. Variaciones del modelo estándar de MT	180
6.5.1. Estado de aceptación único	180
6.5.2. Máquina de Turing con cinta dividida en pistas	181
6.5.3. Máquina de Turing con múltiples cintas	181
6.5.4. Máquinas de Turing no-deterministas (MTN)	183
6.6. Simulación de autómatas por medio de máquinas de Turing	186
6.6.1. Simulación de autómatas	186
6.6.2. Simulación de autómatas con pila	186
6.7. Autómatas con dos pilas (AF2P) ✕	188
6.8. Propiedades de clausura de los lenguajes RE y de los lenguajes recursivos	193
6.9. Máquinas de Turing, computadores, algoritmos y la tesis de Church-Turing	197
6.9.1. Máquinas de Turing y algoritmos	198
6.9.2. Máquinas de Turing y computadores	199
7. Problemas indecidibles	201
7.1. Codificación y enumeración de máquinas de Turing	201
7.2. Máquina de Turing universal	206
7.3. Algoritmos de aceptación para lenguajes RE	209
7.4. Lenguajes que no son RE	211
7.5. Lenguajes RE no recursivos	212
7.6. Problemas indecidibles o irresolubles	215
Bibliografía	221

Prólogo

Este libro contiene lo *mínimo* que los estudiantes de las carreras de ingeniería de sistemas y de matemáticas deberían saber sobre los fundamentos matemáticos de la teoría de la computación. Está basado en el material de clase utilizado por el autor durante los últimos años en la Universidad Nacional de Colombia, sede de Bogotá.

A estudiantes y profesores

El libro está escrito tanto para estudiantes de matemáticas —quienes, es de suponer, tienen más experiencia con razonamientos abstractos y demostraciones— como para estudiantes de ingeniería. Es el profesor quien debe establecer el tono del curso, enfatizando ya sea el rigor matemático o una presentación más intuitiva y práctica. Los resultados están presentados en forma de teoremas, corolarios y lemas, con sus respectivas demostraciones; éstas pueden omitirse, si así lo estima el profesor. En los cursos dirigidos a estudiantes de ingeniería de sistemas, el énfasis debe residir —tanto por parte del profesor como por parte del estudiante— en los ejemplos y ejercicios prácticos; hay que resaltar más el *significado* de los enunciados que sus demostraciones formales. El libro contiene gran cantidad de ejemplos y problemas resueltos, con aplicaciones o ilustraciones directas de la teoría.

Como prerrequisito, es imprescindible que el estudiante haya tomado al menos un curso de matemáticas discretas en el que se haya familiarizado con las nociones básicas y la notación de la teoría intuitiva de conjuntos, grafos, inducción matemática y lógica elemental. La experiencia previa en programación es muy útil pero, de ninguna manera, necesaria.

El material se presenta en secciones relativamente cortas, lo que permite alguna flexibilidad en la selección de los tópicos del curso. Así, si el tiempo

disponible no es holgado, o no es posible avanzar con la velocidad suficiente, podrían suprimirse las secciones demarcadas con el símbolo ✕.

Casi todas las secciones poseen ejercicios, de variada dificultad; los más difíciles están precedidos de un símbolo de admiración ! y podrían ser considerados opcionales. Es responsabilidad del estudiante resolver los ejercicios que sean asignados por el profesor. La única manera de aprender y asimilar las ideas y técnicas presentadas en la clase es trabajar seria y completamente los ejercicios.

Material de apoyo en la red

Versiones preliminares de estas notas aparecieron, de forma incompleta, en el curso virtual de Teoría de la Computación perteneciente al programa *Universidad Virtual* de la Universidad Nacional de Colombia. Es la intención del autor mantener y actualizar permanentemente la versión virtual interactiva de este curso, con material de apoyo como temas y ejercicios nuevos, corrección de errores, software, enlaces a otras páginas Web, etc. Se puede acceder libremente al curso virtual en el portal

<http://www.virtual.unal.edu.co/>

siguiendo los enlaces: Cursos–Facultad de Ciencias–Matemáticas–Teoría de la Computación.

Agradecimientos

Durante la elaboración de estas notas he recibido por parte de estudiantes atentos muchas observaciones útiles que han ayudado a mejorar sustancialmente la presentación. Quiero expresarles mis agradecimientos a todos ellos, demasiado numerosos para mencionarlos individualmente.

La primera versión del curso virtual fue realizada con la ayuda del estudiante de posgrado Adolfo Reyes, a quien expreso mi gratitud y reconocimiento. Para la preparación de la presente versión tuve la suerte de contar con la colaboración del estudiante de matemáticas Camilo Cubides, con quien estoy muy agradecido por la calidad y seriedad de su trabajo.

Finalmente, quiero agradecer a Gustavo Rubiano, Director de las oficina de publicaciones de la Facultad de Ciencias, por su continuo apoyo y su cooperación desinteresada.

Introducción

¿Qué es la Teoría de la Computación?

La Teoría de la Computación estudia modelos abstractos de los dispositivos concretos que conocemos como computadores, y analiza lo que se puede y no se puede hacer con ellos. Este estudio teórico se inició varias décadas antes de la aparición de los primeros computadores reales y continúa creciendo, a medida que la computación incrementa su sofisticación.

Entre los muchos tópicos que conforman la teoría de la computación, sólo tendremos la oportunidad de tratar someramente los dos siguientes:

Modelos de computación. Las investigaciones en este campo comenzaron en la década de los 30 del siglo XX con el trabajo del lógico norteamericano Alonzo Church (1903–1995) y del matemático británico Alan Turing (1912–1954). Church introdujo el formalismo conocido como cálculo- λ y enunció la tesis —hoy conocida como tesis de Church— de que las funciones efectivamente computables, es decir, computables por *cualquier* método computacional concebible, son exactamente las funciones λ -computables. En contraste con el enfoque más abstracto de Church, Turing (quien fue alumno doctoral de Church en la universidad de Princeton) propuso un modelo concreto de máquina computadora, hoy conocida como la *máquina de Turing*, capaz de simular las acciones de *cualquier* otro dispositivo físico de computación secuencial.

Curiosamente, las propuestas de Church y Turing se publicaron exactamente en el mismo año: 1936. Los dos formalismos resultaron ser equivalentes y, desde entonces, se han propuesto muchos otros modelos de computación. Como todos han resultado ser equivalentes entre sí, ha ganado

aceptación universal la tesis de Church-Turing: no hay modelo de computación más general ni poderoso que la máquina de Turing.

En los años 40 y 50 del siglo XX se adelantaron investigaciones sobre máquinas de Turing con capacidad restringida, surgiendo así la noción de *máquina de estado finito* o *autómata finito* (“autómata” es sinónimo de “máquina de cómputo automático”). Los autómatas han resultado ser modelos muy útiles para el diseño de diversos tipos de software y hardware.

Lenguajes y gramáticas formales. Una línea investigativa, aparentemente alejada de los modelos de computación, surgió con los estudios del lingüista norteamericano Noam Chomsky¹. Chomsky introdujo en 1956 la noción de *gramática generativa* con el propósito de describir los lenguajes naturales como el español, el inglés, el francés, etc. Chomsky clasificó las gramáticas en cuatro tipos, dependiendo de la forma de sus *producciones*, que son las reglas que utiliza una gramática para generar palabras o cadenas de símbolos. Pocos años después se estableció que hay una estrecha relación entre autómatas y gramáticas: los lenguajes de la llamada *jerarquía de Chomsky* corresponden a los lenguajes que pueden ser reconocidos por tipos especiales de autómatas.

La interacción entre los autómatas (mecanismos para *procesar* cadenas de símbolos) y las gramáticas (mecanismos para *generar* cadenas de símbolos) es una fuente de resultados profundos y significativos. Desde la aparición de los influyentes textos de Hopcroft y Ullman ([HU1], 1969) y ([HU2], 1979), un curso semestral básico de teoría de la computación se ha centrado en el estudio de autómatas y gramáticas. Estas notas de clase reflejan esa tradición.

¹En el año 2002, la Universidad Nacional de Colombia otorgó el doctorado *Honoris Causa* a Noam Chomsky.

Capítulo 1

Alfabetos, cadenas y lenguajes

De manera muy amplia podría decirse que la computación es la manipulación de secuencias de símbolos. Pero el número de símbolos disponibles en cualquier mecanismo de cómputo es finito y todos los objetos usados como entradas o salidas (inputs/outputs) deben ser identificados en un tiempo finito. Desde el punto de vista teórico esto impone dos restricciones básicas: el conjunto de símbolos (alfabeto) debe ser finito y se deben considerar únicamente cadenas (secuencias de símbolos) de longitud finita. Surgen así los ingredientes esenciales de una teoría abstracta de la computación: *alfabetos* y *cadenas*. Los conjuntos de cadenas (ya sean finitos o infinitos) se denominarán *lenguajes*.

1.1. Alfabetos y cadenas

Un **alfabeto** es un conjunto finito no vacío cuyos elementos se llaman **símbolos**. Denotamos un alfabeto arbitrario con la letra Σ .

Una **cadena** o **palabra** sobre un alfabeto Σ es cualquier sucesión (o secuencia) finita de elementos de Σ . Admitimos la existencia de una única cadena que no tiene símbolos, la cual se denomina **cadena vacía** y se denota con λ . La cadena vacía desempeña, en la teoría de la computación, un papel similar al del conjunto vacío \emptyset en la teoría de conjuntos.

Ejemplo Sea $\Sigma = \{a, b\}$ el alfabeto que consta de los dos símbolos a y b . Las siguientes son cadenas sobre Σ :

aba
ababaaa
aaaab.

Obsérvese que $aba \neq aab$. El orden de los símbolos en una cadena es significativo ya que las cadenas se definen como *sucesiones*, es decir, conjuntos *secuencialmente ordenados*.

Ejemplo El alfabeto $\Sigma = \{0, 1\}$ se conoce como *alfabeto binario*. Las cadenas sobre este alfabeto son secuencias finitas de ceros y unos, llamadas *secuencias binarias*, tales como

001
1011
001000001.

Ejemplo $\Sigma = \{a, b, c, \dots, x, y, z, A, B, C, \dots, X, Y, Z\}$, el alfabeto del idioma castellano. Las palabras oficiales del castellano (las que aparecen en el diccionario DRA) son cadenas sobre Σ .

Ejemplo El alfabeto utilizado por muchos de los llamados *lenguajes de programación* (como Pascal o C) es el conjunto de caracteres ASCII (o un subconjunto de él) que incluye, por lo general, las letras mayúsculas y minúsculas, los símbolos de puntuación y los símbolos matemáticos disponibles en los teclados estándares.

El conjunto de *todas* las cadenas sobre un alfabeto Σ , incluyendo la cadena vacía, se denota por Σ^* .

Ejemplo Sea $\Sigma = \{a, b, c\}$, entonces

$$\Sigma^* = \{\lambda, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, abc, baa, \dots\}.$$

En la siguiente tabla aparece la notación corrientemente utilizada en la teoría de la computación. De ser necesario, se emplean subíndices.

Notación usada en la teoría de la computación	
Σ, Γ	denotan alfabetos.
Σ^*	denota el conjunto de todas las cadenas que se pueden formar con los símbolos del alfabeto Σ .
a, b, c, d, e, \dots	denotan símbolos de un alfabeto.
u, v, w, x, y, z, \dots	denotan cadenas, es decir, sucesiones finitas de
$\alpha, \beta, \gamma, \dots$	símbolos de un alfabeto.
λ	denota la cadena vacía, es decir, la única cadena que no tiene símbolos.
$A, B, C, \dots, L, M, N, \dots$	denotan lenguajes (definidos más adelante).

- ✎ Algunos autores denotan la cadena vacía con la letra griega ε . Preferimos denotarla con λ porque ε tiende a confundirse con el símbolo \in usado para la relación de pertenencia.
- ✎ Si bien un alfabeto Σ es un conjunto finito, Σ^* es siempre un conjunto infinito (enumerable). En el caso más simple, Σ contiene solo un símbolo, $\Sigma = \{a\}$, y $\Sigma^* = \{\lambda, a, aa, aaa, aaaa, aaaaa, \dots\}$.
- ✎ Hay que distinguir entre los siguientes cuatro objetos, que son todos diferentes entre sí: \emptyset , λ , $\{\emptyset\}$ y $\{\lambda\}$.
- ✎ La mayor parte de la teoría de la computación se hace con referencia a un alfabeto Σ fijo (pero arbitrario).

1.2. Concatenación de cadenas

Dado un alfabeto Σ y dos cadenas $u, v \in \Sigma^*$, la **concatenación de u y v** se denota como $u \cdot v$ o simplemente uv y se define descriptivamente así:

1. Si $v = \lambda$, entonces $u \cdot \lambda = \lambda \cdot u = u$. Es decir, la concatenación de cualquier cadena u con la cadena vacía, a izquierda o a derecha, es igual a u .
2. Si $u = a_1 a_2 \cdots a_n$, $v = b_1 b_2 \cdots b_m$, entonces

$$u \cdot v = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

Es decir, $u \cdot v$ es la cadena formada escribiendo los símbolos de u y a continuación los símbolos de v .

La concatenación de cadenas se puede definir inductiva o recursivamente de la siguiente manera. Si $u, v \in \Sigma^*$, $a \in \Sigma$, entonces

1. $u \cdot \lambda = \lambda \cdot u = u$.
2. $u \cdot (va) = (u \cdot v)a$.

Propiedad. La concatenación de cadenas es una operación asociativa. Es decir, si $u, v, w \in \Sigma^*$, entonces

$$(uv)w = u(vw).$$

Demostración. Se puede hacer escribiendo explícitamente las cadenas u , v , w y usando la definición descriptiva de concatenación. También se puede dar una demostración inductiva usando la definición recursiva de concatenación (ejercicio opcional). □

1.3. Potencias de una cadena

Dada $u \in \Sigma^*$ y $n \in \mathbb{N}$, se define (descriptivamente) u^n en la siguiente forma

$$\begin{aligned} u^0 &= \lambda, \\ u^n &= \underbrace{uu \cdots u}_{n \text{ veces}}. \end{aligned}$$

Como ejercicio, el estudiante puede dar una definición recursiva de u^n .

1.4. Longitud de una cadena

La **longitud** de una cadena $u \in \Sigma^*$ se denota $|u|$ y se define como el número de símbolos de u (contando los símbolos repetidos). Es decir,

$$|u| = \begin{cases} 0, & \text{si } u = \lambda, \\ n, & \text{si } u = a_1 a_2 \cdots a_n. \end{cases}$$

Ejemplo $|aba| = 3$, $|baaa| = 4$.

Ejemplo Si $w \in \Sigma^*$, $n, m \in \mathbb{N}$, demostrar que $|w^{n+m}| = |w^n| + |w^m|$. Esta no es una propiedad realmente importante (¡no la usaremos nunca en este libro!); la presentamos aquí para enfatizar los conceptos involucrados e ilustrar los razonamientos estrictos.

Solución. Caso $n, m \geq 1$. $|w^{n+m}| = |\underbrace{ww \cdots w}_{n+m \text{ veces}}| = (n+m)|w|$. Por otro lado,

$$|w^n| + |w^m| = |\underbrace{ww \cdots w}_{n \text{ veces}}| + |\underbrace{ww \cdots w}_{m \text{ veces}}| = n|w| + m|w|.$$

Caso $n = 0, m \geq 1$. $|w^{n+m}| = |w^{0+m}| = |w^m|$. Por otro lado,

$$|w^n| + |w^m| = |w^0| + |w^m| = |\lambda| + |w^m| = 0 + |w^m| = |w^m|.$$

Caso $m = 0, n \geq 1$. Similar al caso anterior.

Caso $n = 0, m = 0$. $|w^{n+m}| = |w^{0+0}| = |\lambda| = 0$. Por otro lado,

$$|w^n| + |w^m| = |w^0| + |w^0| = |\lambda| + |\lambda| = 0 + 0 = 0.$$

1.5. Reflexión o inversa de una cadena

La **reflexión** o **inversa** de una cadena $u \in \Sigma^*$ se denota u^R y se define descriptivamente así:

$$u^R = \begin{cases} \lambda, & \text{si } u = \lambda, \\ a_n \cdots a_2 a_1, & \text{si } u = a_1 a_2 \cdots a_n. \end{cases}$$

De la definición se observa claramente que la reflexión de la reflexión de una cadena es la misma cadena, es decir,

$$(u^R)^R = u, \quad \text{para } u \in \Sigma^*.$$

 Algunos autores escriben u^{-1} en lugar de u^R para denotar la reflexión de una cadena u .

Ejercicios de la sección 1.5

- ① Dar una definición recursiva de u^R .
- ② Si $u, v \in \Sigma^*$, demostrar que $(uv)^R = v^R u^R$. Generalizar esta propiedad a la concatenación de n cadenas.

1.6. Subcadenas, prefijos y sufijos

Una cadena v es una **subcadena** o una **subpalabra** de u si existen cadenas x, y tales que $u = xvy$. Nótese que x o y pueden ser λ y, por lo tanto, la cadena vacía es una subcadena de cualquier cadena y toda cadena es subcadena de sí misma.

Un **prefijo** de u es una cadena v tal que $u = vw$ para alguna cadena $w \in \Sigma^*$. Se dice que v es un **prefijo propio** si $v \neq u$.

Similarmente, un **sufijo** de u es una cadena v tal que $u = wv$ para alguna cadena $w \in \Sigma^*$. Se dice que v es un **sufijo propio** si $v \neq u$.

Obsérvese que λ es un prefijo y un sufijo de toda cadena u ya que $u\lambda = \lambda u = u$. Por la misma razón, toda cadena u es prefijo y sufijo de sí misma.

Ejemplo Sean $\Sigma = \{a, b, c, d\}$ y $u = bcbaadb$.

Prefijos de u :
 λ
 b
 bc
 bcb
 $bcba$
 $bcbaa$
 $bcbaad$
 $bcbaadb$
Sufijos de u :
 λ
 b
 db
 adb
 $aadb$
 $baadb$
 $cbaadb$
 $bcbaadb$

1.7. Lenguajes

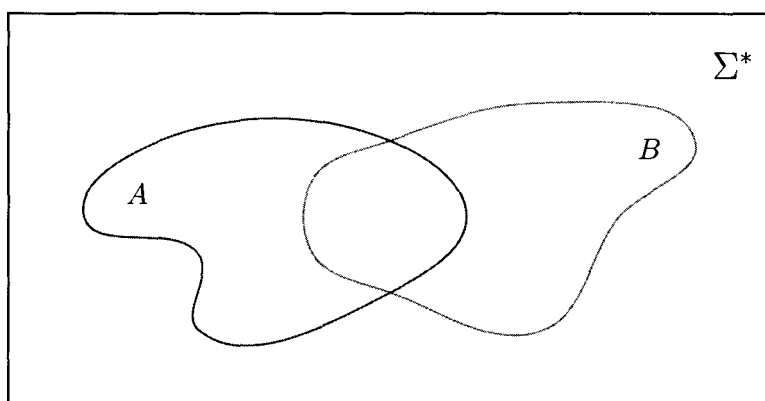
Un **lenguaje** L sobre un alfabeto Σ es un subconjunto de Σ^* , es decir $L \subseteq \Sigma^*$.

Casos extremos:

$L = \emptyset$, lenguaje vacío.

$L = \Sigma^*$, lenguaje de todas las cadenas sobre Σ .

Todo lenguaje L satisface $\emptyset \subseteq L \subseteq \Sigma^*$, y puede ser finito o infinito. Los lenguajes se denotan con letras mayúsculas $A, B, C, \dots, L, M, N, \dots$. En la siguiente gráfica se visualizan dos lenguajes A y B sobre Σ .



Ejemplos


Los siguientes son ejemplos de lenguajes sobre los alfabetos especificados.

- $\Sigma = \{a, b, c\}$. $L = \{a, aba, aca\}$.
- $\Sigma = \{a, b, c\}$. $L = \{a, aa, aaa, \dots\} = \{a^n : n \geq 1\}$.

- $\Sigma = \{a, b, c\}$. $L = \{\lambda, aa, aba, ab^2a, ab^3a, \dots\} = \{ab^n a : n \geq 0\} \cup \{\lambda\}$.
- $\Sigma = \{a, b, c, \dots, x, y, z, A, B, C, \dots, X, Y, Z\}$. $L = \{u \in \Sigma^* : u \text{ aparece en el diccionario español DRA}\}$. L es un lenguaje finito.
- $\Sigma = \{a, b, c\}$. $L = \{u \in \Sigma^* : u \text{ no contiene el símbolo } c\}$. Por ejemplo, $abbaab \in L$ pero $abbcaa \notin L$.
- $\Sigma = \{0, 1\}$. $L =$ conjunto de todas las secuencias binarias que contienen un número impar de unos.
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. El conjunto \mathbb{N} de los números naturales se puede definir como un lenguaje sobre Σ , en la siguiente forma:

$$\mathbb{N} = \{u \in \Sigma^* : u = 0 \text{ ó } 0 \text{ no es un prefijo de } u\}.$$

Como ejercicio, el estudiante puede definir el conjunto de los enteros $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ como un lenguaje sobre un alfabeto adecuado.

 El concepto abstracto de “lenguaje”, tal como se ha definido, no es exactamente la misma noción utilizada en la expresión “lenguaje de programación”. Para precisar la relación entre estos conceptos, consideremos el alfabeto Σ de los caracteres ASCII. Un programa en C o en Pascal, por ejemplo, es simplemente una cadena de símbolos de Σ y, por lo tanto, un conjunto de programas es un lenguaje (en el sentido formal definido en esta sección).

1.8. Operaciones entre lenguajes

Puesto que los lenguajes sobre Σ son subconjuntos de Σ^* , las operaciones usuales entre conjuntos son también operaciones válidas entre lenguajes. Así, si A y B son lenguajes sobre Σ (es decir $A, B \subseteq \Sigma^*$), entonces los siguientes también son lenguajes sobre Σ :

$A \cup B$	Unión
$A \cap B$	Intersección
$A - B$	Diferencia
$\overline{A} = \Sigma^* - A$	Complemento

Estas operaciones entre lenguajes se llaman *operaciones conjuntistas* o *booleanas* para distinguirlas de las *operaciones lingüísticas* (concatenación, potencia, inverso, clausura) que son extensiones a los lenguajes de las operaciones entre cadenas.

1.9. Concatenación de lenguajes

La **concatenación** de dos lenguajes A y B sobre Σ , notada $A \cdot B$ o simplemente AB se define como

$$AB = \{uv : u \in A, v \in B\}.$$

En general, $AB \neq BA$.

Ejemplo Si $\Sigma = \{a, b, c\}$, $A = \{a, ab, ac\}$, $B = \{b, b^2\}$, entonces

$$\begin{aligned} AB &= \{ab, ab^2, ab^2, ab^3, acb, acb^2\}. \\ BA &= \{ba, bab, bac, b^2a, b^2ab, b^2ac\}. \end{aligned}$$

Ejemplo Si $\Sigma = \{a, b, c\}$, $A = \{ba, bc\}$, $B = \{b^n : n \geq 0\}$, entonces

$$\begin{aligned} AB &= \{bab^n : n \geq 0\} \cup \{bcb^n : n \geq 0\}. \\ BA &= \{b^nba : n \geq 0\} \cup \{b^nb c : n \geq 0\} \\ &= \{b^{n+1}a : n \geq 0\} \cup \{b^{n+1}c : n \geq 0\} \\ &= \{b^na : n \geq 1\} \cup \{b^nc : n \geq 1\}. \end{aligned}$$

Propiedades de la concatenación de lenguajes. Sean A, B, C lenguajes sobre Σ , es decir $A, B, C \subseteq \Sigma^*$. Entonces

1. $A \cdot \emptyset = \emptyset \cdot A = \emptyset$.
2. $A \cdot \{\lambda\} = \{\lambda\} \cdot A = A$.
3. Propiedad Asociativa,

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C.$$

4. Distributividad de la concatenación con respecto a la unión,

$$\begin{aligned} A \cdot (B \cup C) &= A \cdot B \cup A \cdot C. \\ (B \cup C) \cdot A &= B \cdot A \cup C \cdot A. \end{aligned}$$

5. Propiedad distributiva generalizada. Si $\{B_i\}_{i \in I}$ es una familia cualquiera de lenguajes sobre Σ , entonces

$$\begin{aligned} A \cdot \bigcup_{i \in I} B_i &= \bigcup_{i \in I} (A \cdot B_i), \\ \left(\bigcup_{i \in I} B_i \right) \cdot A &= \bigcup_{i \in I} (B_i \cdot A). \end{aligned}$$

Demostración.

1. $A \cdot \emptyset = \{uv : u \in A, v \in \emptyset\} = \emptyset$.
2. $A \cdot \{\lambda\} = \{uv : u \in A, v \in \{\lambda\}\} = \{u : u \in A\} = A$.
3. Se sigue de la asociatividad de la concatenación de cadenas.
4. Caso particular de la propiedad general, demostrada a continuación.
5. Demostración de la igualdad $A \cdot \bigcup_{i \in I} B_i = \bigcup_{i \in I} (A \cdot B_i)$:

$$\begin{aligned}
 x \in A \cdot \bigcup_{i \in I} B_i &\iff x = u \cdot v, \quad \text{con } u \in A \text{ \& } v \in \bigcup_{i \in I} B_i \\
 &\iff x = u \cdot v, \quad \text{con } u \in A \text{ \& } v \in B_j, \text{ para algún } j \in I \\
 &\iff x \in A \cdot B_j, \quad \text{para algún } j \in I \\
 &\iff x \in \bigcup_{i \in I} (A \cdot B_i).
 \end{aligned}$$

La igualdad $\left(\bigcup_{i \in I} B_i\right) \cdot A = \bigcup_{i \in I} (B_i \cdot A)$ se demuestra de forma similar. \square

- ☞ La propiedad asociativa permite escribir concatenaciones de tres o más lenguajes sin necesidad de usar paréntesis.
- ☞ En general, no se cumple que $A \cdot (B \cap C) = A \cdot B \cap A \cdot C$. Es decir, la concatenación no es distributiva con respecto a la intersección. Contraejemplo: $A = \{a, \lambda\}$, $B = \{\lambda\}$, $C = \{a\}$. Se tiene:

$$A \cdot (B \cap C) = \{a, \lambda\} \cdot \emptyset = \emptyset.$$

Por otro lado,

$$A \cdot B \cap A \cdot C = \{a, \lambda\} \cdot \{\lambda\} \cap \{a, \lambda\} \cdot \{a\} = \{a, \lambda\} \cap \{a^2, a\} = \{a\}.$$

Ejercicios de la sección 1.9

- ① Dar un ejemplo de un alfabeto Σ y dos lenguajes diferentes A, B sobre Σ tales que $AB = BA$.
- ② Una de las dos contencencias siguientes es verdadera y la otra es falsa. Demostrar o refutar, según sea el caso:
 - (i) $A \cdot (B \cap C) \subseteq A \cdot B \cap A \cdot C$.
 - (ii) $A \cdot B \cap A \cdot C \subseteq A \cdot (B \cap C)$.

1.10. Potencias de un lenguaje

Dado un lenguaje A sobre Σ , ($A \subseteq \Sigma^*$), y un número natural $n \in \mathbb{N}$, se define A^n en la siguiente forma

$$A^0 = \{\lambda\},$$

$$A^n = \underbrace{AA \cdots A}_{n \text{ veces}} = \{u_1 \cdots u_n : u_i \in A, \text{ para todo } i, 1 \leq i \leq n\}.$$

De esta forma, A^2 es el conjunto de las concatenaciones dobles de cadenas de A , A^3 está formado por las concatenaciones triples y, en general, A^n es el conjunto de todas las concatenaciones de n cadenas de A , de todas las formas posibles. Como ejercicio, el estudiante puede dar una definición recursiva de A^n .

1.11. La clausura de Kleene de un lenguaje

La **clausura de Kleene** o **estrella de Kleene** o simplemente la **estrella** de un lenguaje A , $A \subseteq \Sigma^*$, es la unión de todas las potencias de A y se denota por A^* .

(Descripción 1)

$$A^* = \bigcup_{i \geq 0} A^i = A^0 \cup A^1 \cup A^2 \cup \cdots \cup A^n \cdots$$

Según la definición de las potencias de un lenguaje, A^* consta de todas las concatenaciones de cadenas de A consigo mismas, de todas las formas posibles. Tenemos así una útil descripción de A^* :

(Descripción 2)

$$\begin{aligned} A^* &= \text{conjunto de } \textit{todas} \text{ las concatenaciones} \\ &\quad \text{de cadenas de } A, \text{ incluyendo } \lambda \\ &= \{u_1 \cdots u_n : u_i \in A, \quad n \geq 0\} \end{aligned}$$

De manera similar se define la **clausura positiva** de un lenguaje A , $A \subseteq \Sigma^*$, denotada por A^+ .

$$A^+ = \bigcup_{i \geq 1} A^i = A^1 \cup A^2 \cup \cdots \cup A^n \cdots$$

A^+ se puede describir de la siguiente manera

$$\begin{aligned} A^+ &= \text{conjunto de } \textit{todas} \text{ las concatenaciones de cadenas de } A \\ &= \{u_1 \cdots u_n : u_i \in A, \quad n \geq 1\} \end{aligned}$$

Obsérvese que $A^* = A^+ \cup \{\lambda\}$ y que $A^* = A^+$ si y solamente si $\lambda \in A$.

Propiedades de $*$ y $+$. Sea A un lenguaje sobre Σ , es decir, $A \subseteq \Sigma^*$.

1. $A^+ = A^* \cdot A = A \cdot A^*$.
2. $A^* \cdot A^* = A^*$.
3. $(A^*)^n = A^*$, para todo $n \geq 1$.
4. $(A^*)^* = A^*$.
5. $A^+ \cdot A^+ \subseteq A^+$.
6. $(A^*)^+ = A^*$.
7. $(A^+)^* = A^*$.
8. $(A^+)^+ = A^+$.
9. Si A y B son lenguajes sobre Σ^* , entonces $(A \cup B)^* = (A^* B^*)^*$.

Demostración.

$$\begin{aligned}
 1. \quad A \cdot A^* &= A \cdot (A^0 \cup A^1 \cup A^2 \cup \dots) \\
 &= A^1 \cup A^2 \cup A^3 \cup \dots \\
 &= A^+.
 \end{aligned}$$

Similarmente se demuestra que $A^* \cdot A = A^+$.

2. Si $x \in A^* \cdot A^*$, entonces $x = u \cdot v$, con $u \in A^*$, $v \in A^*$. De modo que, $x = u \cdot v$, con $u = u_1 u_2 \cdots u_n$, $u_i \in A$, $n \geq 0$ y $v = v_1 v_2 \cdots v_m$, $v_i \in A$, $m \geq 0$.

De donde

$$x = u \cdot v = u_1 \cdot u_2 \cdots u_n \cdot v_1 \cdot v_2 \cdots v_m.$$

con $u_i \in A$, $v_i \in A$, $n \geq 0$. Por lo tanto, x es una concatenación de $n + m$ cadenas de A . Así que $x \in A^*$.

Recíprocamente, si $x \in A^*$, entonces $x = x \cdot \lambda \in A^* \cdot A^*$. Esto prueba la igualdad de los conjuntos $A^* \cdot A^*$ y A^* .

3. Se sigue de la propiedad anterior.

$$\begin{aligned}
 4. \quad (A^*)^* &= (A^*)^0 \cup (A^*)^1 \cup (A^*)^2 \cup \dots \\
 &= \{\lambda\} \cup A^* \cup A^* \cup A^* \cup \dots \\
 &= A^*.
 \end{aligned}$$

5. La demostración de esta propiedad es similar a la de la propiedad 2, pero con la restricción $m, n \geq 1$. En general, no se tiene la igualdad $A^+ \cdot A^+ = A^+$; más adelante se mostrará un contraejemplo.

$$\begin{aligned} 6. \quad (A^*)^+ &= (A^*)^1 \cup (A^*)^2 \cup (A^*)^3 \cup \dots \\ &= A^* \cup A^* \cup A^* \cup \dots \\ &= A^*. \end{aligned}$$

$$\begin{aligned} 7. \quad (A^+)^* &= (A^+)^0 \cup (A^+)^1 \cup (A^+)^2 \cup \dots \\ &= \{\lambda\} \cup A^+ \cup A^+ A^+ \cup \dots \\ &= A^* \cup (\text{conjuntos contenidos en } A^+) \\ &= A^*. \end{aligned}$$

$$\begin{aligned} 8. \quad (A^+)^+ &= (A^+)^1 \cup (A^+)^2 \cup (A^+)^3 \cup \dots, \\ &= A^+ \cup (\text{conjuntos contenidos en } A^+) \\ &= A^+. \end{aligned}$$

9. Según la Descripción 2, el lenguaje $(A \cup B)^*$ está formado por las concatenaciones de cadenas de A consigo mismas, las concatenaciones de cadenas de B consigo mismas y las concatenaciones de cadenas de A con cadenas de B , de todas las formas posibles y en cualquier orden. Si se usa también la Descripción 2, se observa que eso mismo se obtiene al efectuar $(A^* B^*)^*$. \square

☞ **Contraejemplo de $A^+ \cdot A^+ = A^+$.** Sea $\Sigma = \{a, b\}$, $A = \{a\}$. Se tiene

$$A^+ = A^1 \cup A^2 \cup \dots = \{a\} \cup \{aa\} \cup \{aaa\} \cup \dots = \{a^n : n \geq 1\}.$$

Por otro lado,

$$\begin{aligned} A^+ \cdot A^+ &= \{a, a^2, a^3, \dots\} \cdot \{a, a^2, a^3, \dots\} = \{a^2, a^3, a^4, \dots\} \\ &= \{a^n : n \geq 2\}. \end{aligned}$$

☞ Según las definiciones dadas, Σ^* tiene dos significados:

Σ^* = conjunto de las cadenas sobre el alfabeto Σ .

Σ^* = conjunto de todas las concatenaciones de cadenas de Σ .

No hay conflicto de notaciones porque las dos definiciones anteriores de Σ^* dan lugar al mismo conjunto.

1.12. Reflexión o inverso de un lenguaje

Dado A un lenguaje sobre Σ , se define A^R de la siguiente forma:

$$A^R = \{u^R : u \in A\}.$$

A^R se denomina la **reflexión** o el **inverso** de A .

Propiedades. Sean A y B lenguajes sobre Σ (es decir, $A, B \subseteq \Sigma^*$).

1. $(A \cdot B)^R = B^R \cdot A^R$.
2. $(A \cup B)^R = A^R \cup B^R$.
3. $(A \cap B)^R = A^R \cap B^R$.
4. $(A^R)^R = A$.
5. $(A^*)^R = (A^R)^*$.
6. $(A^+)^R = (A^R)^+$.

Demostración. Demostraremos las propiedades 1 y 5; las demás se dejan como ejercicio para el estudiante.

1. $x \in (A \cdot B)^R \iff x = u^R, \text{ donde } u \in A \cdot B$
 $\iff x = u^R, \text{ donde } u = vw, v \in A, w \in B$
 $\iff x = (vw)^R, \text{ donde } v \in A, w \in B$
 $\iff x = w^R v^R, \text{ donde } v \in A, w \in B$
 $\iff x \in B^R \cdot A^R$.

5. $x \in (A^*)^R \iff x = u^R, \text{ donde } u \in A^*$
 $\iff x = (u_1 \cdot u_2 \cdots u_n)^R, \text{ donde los } u_i \in A, n \geq 0$
 $\iff x = u_n^R \cdot u_{n-1}^R \cdots u_1^R, \text{ donde los } u_i \in A, n \geq 0$
 $\iff x \in (A^R)^*$.

Ejercicios de la sección 1.12

- ① Demostrar las propiedades 2, 3, 4 y 6 de la reflexión de cadenas.
- ② ¿Se pueden generalizar las propiedades 2 y 3 anteriores para uniones e intersecciones arbitrarias, respectivamente?

1.13. Lenguajes regulares

Los **lenguajes regulares** sobre un alfabeto dado Σ son todos los lenguajes que se pueden formar a partir de los lenguajes básicos \emptyset , $\{\lambda\}$, $\{a\}$, $a \in \Sigma$, por medio de las operaciones de unión, concatenación y estrella de Kleene.

A continuación presentamos una definición recursiva de los lenguajes regulares. Sea Σ un alfabeto.

1. \emptyset , $\{\lambda\}$ y $\{a\}$, para cada $a \in \Sigma$, son lenguajes regulares sobre Σ . Estos son los denominados lenguajes regulares básicos.
2. Si A y B son lenguajes regulares sobre Σ , también lo son

$$\begin{array}{ll} A \cup B & (\text{unión}), \\ A \cdot B & (\text{concatenación}), \\ A^* & (\text{estrella de Kleene}). \end{array}$$

Obsérvese que tanto Σ como Σ^* son lenguajes regulares sobre Σ . La unión, la concatenación y la estrella de Kleene se denominan **operaciones regulares**.

Ejemplos Sea $\Sigma = \{a, b\}$. Los siguientes son lenguajes regulares sobre Σ :

1. El lenguaje A de todas las cadenas que tienen exactamente una a :

$$A = \{b\}^* \cdot \{a\} \cdot \{b\}^*.$$

2. El lenguaje B de todas las cadenas que comienzan con b :

$$B = \{b\} \cdot \{a, b\}^*.$$

3. El lenguaje C de todas las cadenas que contienen la cadena ba :

$$C = \{a, b\}^* \cdot \{ba\} \cdot \{a, b\}^*.$$

$$4. (\{a\} \cup \{b\}^*) \cdot \{a\}.$$

$$5. [(\{a\}^* \cup \{b\}^*) \cdot \{b\}]^*.$$

Es importante observar que *todo lenguaje finito* $L = \{w_1, w_2, \dots, w_n\}$ es regular ya que L se puede obtener con uniones y concatenaciones:

$$L = \{w_1\} \cup \{w_2\} \cup \dots \cup \{w_n\},$$

y cada w_i es la concatenación de un número finito de símbolos, $w_i = a_1 a_2 \dots a_k$; por lo tanto, $\{w_i\} = \{a_1\} \cdot \{a_2\} \dots \{a_k\}$.

1.14. Expresiones regulares

Con el propósito de simplificar la descripción de los lenguajes regulares se definen las llamadas expresiones regulares.

La siguiente es la definición recursiva de las **expresiones regulares** sobre un alfabeto Σ dado.

1. Expresiones regulares básicas:

- \emptyset es una expresión regular que representa al lenguaje \emptyset .
- λ es una expresión regular que representa al lenguaje $\{\lambda\}$.
- a es una expresión regular que representa al lenguaje $\{a\}$, $a \in \Sigma$.

2. Si R y S son expresiones regulares sobre Σ , también lo son:

$$\begin{aligned} (R)(S) \\ (R \cup S) \\ (R)^* \end{aligned}$$

$(R)(S)$ representa la concatenación de los lenguajes representados por R y S ; $(R \cup S)$ representa su unión, y $(R)^*$ representa la clausura de Kleene del lenguaje representado por R . Los paréntesis (y) son símbolos de agrupación y se pueden omitir si no hay peligro de ambigüedad.

Para una expresión regular R cualquiera se utiliza en ocasiones la siguiente notación:

$$L(R) := \text{lenguaje representado por } R.$$

Utilizando esta notación y la definición recursiva de expresión regular podemos escribir las siguientes igualdades en las que R y S son expresiones regulares arbitrarias:

$$\begin{aligned} L(\emptyset) &= \emptyset. \\ L(\lambda) &= \{\lambda\}. \\ L(a) &= \{a\}, \quad a \in \Sigma. \\ L(RS) &= L(R)L(S). \\ L(R \cup S) &= L(R) \cup L(S). \\ L(R^*) &= L(R)^*. \end{aligned}$$

Ejemplo

Dado el alfabeto $\Sigma = \{a, b, c\}$,

$$(a \cup b^*)a^*(bc)^*$$

es una expresión regular que representa al lenguaje

$$(\{a\} \cup \{b\})^* \cdot \{a\}^* \cdot \{bc\}^*.$$

Ejemplo Dado el alfabeto $\Sigma = \{a, b\}$,

$$(\lambda \cup a)^*(a \cup b)^*(ba)^*$$

es una expresión regular que representa al lenguaje

$$(\{\lambda\} \cup \{a\})^* \cdot (\{a\} \cup \{b\})^* \cdot \{ba\}^*.$$

Ejemplos Podemos representar los tres primeros lenguajes de la sección 1.13 con expresiones regulares:

1. El lenguaje A de todas las cadenas que tienen exactamente una a :

$$A = b^*ab^*.$$

2. El lenguaje B de todas las cadenas que comienzan con b :

$$B = b(a \cup b)^*.$$

3. El lenguaje C de todas las cadenas que contienen la cadena ba :

$$C = (a \cup b)^*ba(a \cup b)^*.$$

☛ La representación de lenguajes regulares por medio de expresiones regulares no es única. Es posible que haya varias expresiones regulares diferentes para el mismo lenguaje. Por ejemplo, $b(a \cup b)^*$ y $b(b \cup a)^*$ representan el mismo lenguaje.

Otro ejemplo: las dos expresiones regulares $(a \cup b)^*$ y $(a^*b^*)^*$ representan el mismo lenguaje por la propiedad 9 de la sección 1.11.

Ejemplos Encontrar expresiones regulares que representen los siguientes lenguajes, definidos sobre el alfabeto $\Sigma = \{a, b\}$:

1. Lenguaje de todas las cadenas que comienzan con el símbolo b y terminan con el símbolo a .

Solución. $b(a \cup b)^*a$.

2. Lenguaje de todas las cadenas que tienen un número par de símbolos (cadenas de longitud par).

Solución. $(aa \cup ab \cup ba \cup bb)^*$. Otra expresión regular para este lenguaje es $[(a \cup b)(a \cup b)]^*$.

3. Lenguaje de todas las cadenas que tienen un número par de a es.

Soluciones:

$$\begin{aligned} &b^*(b^*ab^*ab^*)^*. \\ &(ab^*a \cup b)^*. \\ &(b^*ab^*ab^*)^* \cup b^*. \end{aligned}$$

Ejemplos Encontrar expresiones regulares que representen los siguientes lenguajes, definidos sobre el alfabeto $\Sigma = \{0, 1\}$:

1. Lenguaje de todas las cadenas que tienen exactamente dos ceros.

Solución. $1^*01^*01^*$.

2. Lenguaje de todas las cadenas cuyo penúltimo símbolo, de izquierda a derecha, es un 0.

Solución. $(0 \cup 1)^*0(0 \cup 1)$. Usando la propiedad distributiva obtenemos otra expresión regular para este lenguaje: $(0 \cup 1)^*00 \cup (0 \cup 1)^*01$.

Ejemplo Sea $\Sigma = \{0, 1\}$. Encontrar una expresión regular que represente el lenguaje de todas las cadenas que no contienen dos ceros consecutivos.

Solución. La condición de que no haya dos ceros consecutivos implica que todo cero debe estar seguido necesariamente de un uno, excepto un cero al final de la cadena. Por lo tanto, las cadenas de este lenguaje se obtienen concatenando unos con bloques 01, de todas las formas posibles. Hay que tener en cuenta, además, que la cadena puede terminar ya sea en 1 o en 0. A partir de este análisis, llegamos a la expresión regular

$$(1 \cup 01)^* \cup (1 \cup 01)^*0.$$

Usando la propiedad distributiva, obtenemos otra expresión para este lenguaje: $(1 \cup 01)^*(\lambda \cup 0)$.

Ejemplo Sea $\Sigma = \{a, b, c\}$. Encontrar una expresión regular que represente el lenguaje de todas las cadenas que no contienen la cadena bc .

Solución. Una b puede estar seguida solamente de otra b o de una a , mientras que las a s y las c s pueden estar seguidas de cualquier símbolo. Teniendo en cuenta todas las restricciones y posibilidades, arribamos a la siguiente expresión regular:

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

La condición de que no aparezca la cadena bc significa que una c puede estar precedida solamente de una a y de otra c . Siguiendo esta descripción, obtenemos otra expresión regular para el lenguaje en cuestión:

$$c^*(b \cup ac^*)^*.$$

Ejercicios de la sección 1.14

- ① Encontrar expresiones regulares para los lenguajes descritos a continuación:
 - (i) $\Sigma = \{0, 1, 2\}$. Lenguaje de todas las cadenas que comienzan con 2 y terminan con 1.
 - (ii) $\Sigma = \{a, b, c\}$. Lenguaje de todas las cadenas que tienen un número par de símbolos.
 - (iii) $\Sigma = \{a, b\}$. Lenguaje de todas las cadenas que tienen un número impar de símbolos.
 - (iv) $\Sigma = \{a, b, c\}$. Lenguaje de todas las cadenas que tienen un número impar de símbolos.
 - (v) $\Sigma = \{a, b\}$. Lenguaje de todas las cadenas que tienen un número impar de a s.
 - (vi) $\Sigma = \{a, b\}$. Lenguaje de todas las cadenas que tienen la cadena ab un número par de veces.
 - (vii) $\Sigma = \{a, b\}$. Lenguaje de todas las cadenas que tienen un número par de a s o un número impar de b s.
 - (viii) $\Sigma = \{0, 1, 2\}$. Lenguaje de todas las cadenas que no contienen dos unos consecutivos.
- ② Encontrar expresiones regulares para los siguientes lenguajes definidos sobre el alfabeto $\Sigma = \{0, 1\}$:
 - (i) Lenguaje de todas las cadenas que tienen por lo menos un 0 y por lo menos un 1.

- (ii) Lenguaje de todas las cadenas que tienen a lo sumo dos ceros consecutivos.
- (iii) Lenguaje de todas las cadenas cuyo quinto símbolo, de izquierda a derecha, es un 1.
- (iv) Lenguaje de todas las cadenas de longitud par ≥ 2 formadas por ceros y unos alternados.
- (v) Lenguaje de todas las cadenas cuya longitud es ≥ 4 .
- (vi) Lenguaje de todas las cadenas de longitud impar que tienen unos únicamente en las posiciones impares.
- (vii) Lenguaje de todas las cadenas cuya longitud es un múltiplo de tres.
- (viii) Lenguaje de todas las cadenas que no contienen tres ceros consecutivos.
- (ix) Lenguaje de todas las cadenas que no contienen cuatro ceros consecutivos.
- !(x) Lenguaje de todas las cadenas que no contienen la subcadena 101.

✎ No todos los lenguajes sobre un alfabeto dado Σ son regulares. Más adelante se mostrará que el lenguaje

$$L = \{\lambda, ab, aabb, aaabbb, \dots\} = \{a^n b^n : n \geq 0\}$$

sobre $\Sigma = \{a, b\}$ no se puede representar por medio de una expresión regular, y por lo tanto, no es un lenguaje regular.

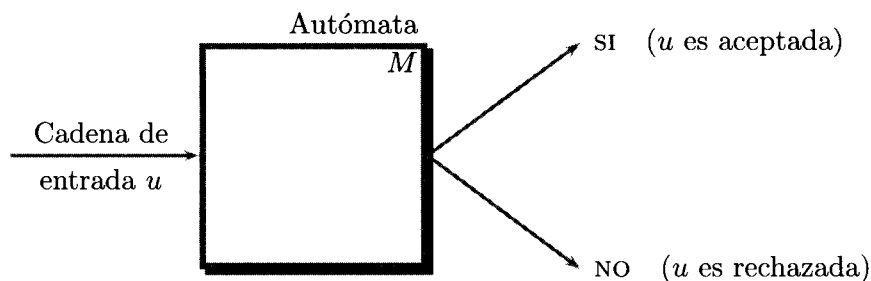
Capítulo 2

Autómatas finitos

Los autómatas son máquinas abstractas con capacidad de computación. Históricamente, su estudio se originó con la llamada “máquina de Turing”, que fue propuesta en 1936 por el matemático británico Alan Turing (1912–1954) con el propósito de precisar las características y las limitaciones de un dispositivo de computación mecánica. En los años 40 y 50 del siglo XX se adelantaron investigaciones sobre máquinas de Turing con capacidad restringida, lo que dio lugar a la noción de autómata finito.

2.1. Autómatas finitos deterministas (AFD)

Los **autómatas finitos** son máquinas abstractas que procesan cadenas de entrada, las cuales son aceptadas o rechazadas:



El autómata actúa leyendo los símbolos escritos sobre una cinta semi-infinita, dividida en celdas o casillas, sobre la cual se escribe una cadena de entrada u , un símbolo por casilla. El autómata posee una **unidad de control** (también llamada **cabeza lectora**, **control finito** o **unidad de**

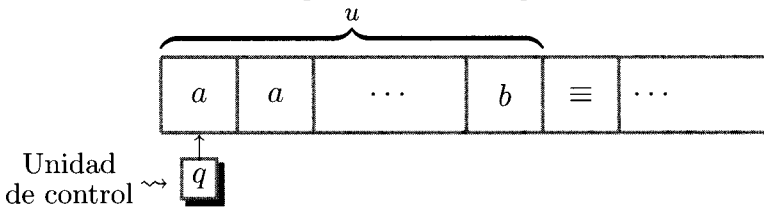
memoria) que tiene un número finito de configuraciones internas, llamadas **estados del autómata**. Entre los estados de un autómata se destacan el **estado inicial** y los **estados finales** o **estados de aceptación**.

Formalmente, un autómata finito M está definido por cinco parámetros o componentes, $M = (\Sigma, Q, q_0, F, \delta)$, a saber:

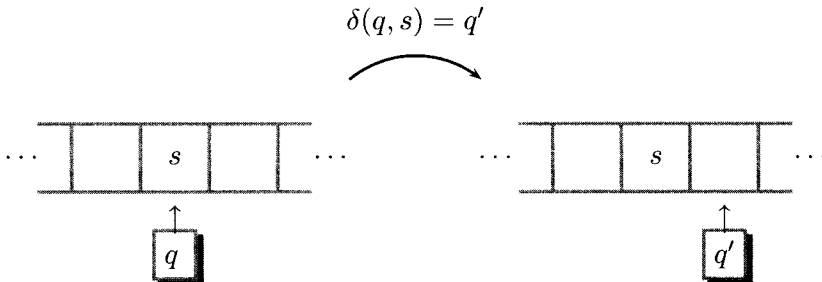
1. Un alfabeto Σ , llamado alfabeto de cinta. Todas las cadenas que procesa M pertenecen a Σ^* .
2. $Q = \{q_0, q_1, \dots, q_n\}$, conjunto de estados internos del autómata.
3. $q_0 \in Q$, estado inicial.
4. $F \subseteq Q$, conjunto de estados finales o de aceptación. $F \neq \emptyset$.
5. La función de transición del autómata

$$\begin{aligned} \delta : Q \times \Sigma &\longrightarrow Q \\ (q, s) &\longmapsto \delta(q, s) \end{aligned}$$

Una cadena de entrada u se coloca en la cinta de tal manera que el primer símbolo de u ocupa la primera casilla de la cinta. La unidad de control está inicialmente en el estado q_0 escaneando la primera casilla:



La función de transición δ indica el estado al cual pasa el control finito, dependiendo del símbolo escaneado y de su estado actual. Así, $\delta(q, s) = q'$ significa que, en presencia del símbolo s , la unidad de control pasa del estado q al estado q' y se desplaza hacia la derecha. Esta acción constituye un **paso computacional**:



Puesto que la función δ está definida para toda combinación estado-símbolo, una cadena de entrada cualquiera es procesada completamente, hasta que la unidad de control encuentra la primera casilla vacía.

La unidad de control de un autómata siempre se desplaza hacia la derecha; no puede retornar ni escribir símbolos sobre la cinta.

Ejemplo Consideremos el autómata definido por los siguientes cinco componentes:

$$\Sigma = \{a, b\}.$$

$$Q = \{q_0, q_1, q_2\}.$$

q_0 : estado inicial.

$$F = \{q_0, q_2\}, \text{ estados de aceptación.}$$

Función de transición δ :

δ	a	b
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_1	q_1

$$\delta(q_0, a) = q_0$$

$$\delta(q_0, b) = q_1$$

$$\delta(q_1, a) = q_1$$

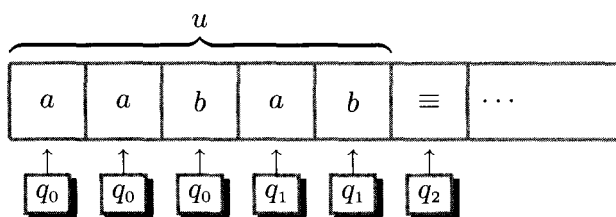
$$\delta(q_1, b) = q_2$$

$$\delta(q_2, a) = q_1$$

$$\delta(q_2, b) = q_1.$$

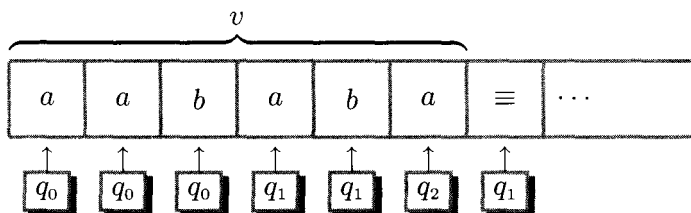
Vamos a ilustrar el procesamiento de dos cadenas de entrada.

1. $u = aabab$.



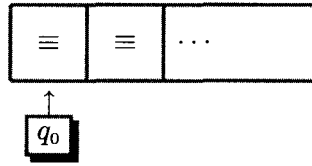
Como q_2 es un estado de aceptación, la cadena de entrada u es aceptada.

2. $v = aababa$.



Puesto que q_1 no es un estado de aceptación, la entrada v es rechazada.

Caso especial: la cadena λ es la cadena de entrada.



Como q_0 es un estado de aceptación, la cadena λ es aceptada.

En general se tiene lo siguiente: la cadena vacía λ es aceptada por un autómata M si y solamente si el estado inicial q_0 de M también es un estado de aceptación.

Los autómatas finitos descritos anteriormente se denominan **autómatas finitos deterministas** (AFD) ya que para cada estado q y para cada símbolo $a \in \Sigma$, la función de transición $\delta(q, a)$ siempre está definida. Es decir, la función de transición δ *determina completa y unívocamente* la acción que el autómata realiza cuando la unidad de control se encuentra en un estado q leyendo un símbolo s sobre la cinta.

Dado un autómata M , el **lenguaje aceptado o reconocido** por M se denota $L(M)$ y se define por

$$L(M) \quad := \quad \{u \in \Sigma^* : M \text{ termina el procesamiento de la cadena de entrada } u \text{ en un estado } q \in F\}.$$

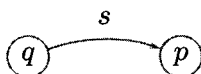
2.2. Diagrama de transiciones de un autómata

Un autómata finito se puede representar por medio de un grafo dirigido y etiquetado. Recuerdese que un **grafo** es un conjunto de vértices o nodos unidos por arcos o conectores; si los arcos tienen tanto dirección como etiquetas, el grafo se denomina **grafo dirigido y etiquetado** o **digrafo etiquetado**.

El digrafo etiquetado de un autómata se obtiene siguiendo las siguientes convenciones:

- Los vértices o nodos son los estados del autómata.
- El estado q se representa por: $\bigcirc q$
- El estado inicial q_0 se representa por: $\succ \bigcirc q_0$

- Un estado final q se representa por: \textcircled{q}
- La transición $\delta = (q, s) = p$ se representa en la forma



Dicho grafo se denomina **diagrama de transiciones del autómata** y es muy útil para hacer el seguimiento completo del procesamiento de una cadena de entrada. Una cadena u es aceptada si existe una trayectoria etiquetada con los símbolos de u , que comienza en el estado q_0 y termina en un estado de aceptación.

Ejemplo Diagrama de transiciones del autómata presentado en la sección anterior.

$\Sigma = \{a, b\}$.

$Q = \{q_0, q_1, q_2\}$.

q_0 : estado inicial.

$F = \{q_0, q_2\}$, estados de aceptación.

Función de transición δ :

δ	a	b
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_1	q_1

$$\delta(q_0, a) = q_0$$

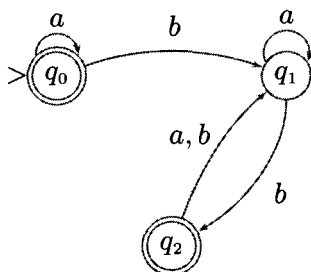
$$\delta(q_0, b) = q_1$$

$$\delta(q_1, a) = q_1$$

$$\delta(q_1, b) = q_2$$

$$\delta(q_2, a) = q_1$$

$$\delta(q_2, b) = q_1$$



Examinando el diagrama de transiciones podemos observar fácilmente que la entrada $aaababbb$ es aceptada mientras que $aabaaba$ es rechazada.

2.3. Diseño de autómatas

Para autómatas deterministas se adopta la siguiente convención adicional con respecto a los diagramas de transiciones: se supone que los arcos no dibujados explícitamente conducen a un estado “limbo” de no-aceptación. Es decir, en el diagrama de transiciones se indican únicamente los arcos que intervengan en trayectorias de aceptación. Esto permite simplificar considerablemente los diagramas.

En este capítulo abordaremos dos tipos de problemas:

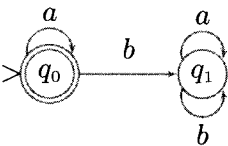
1. Dado un lenguaje regular L diseñar un autómata finito M que acepte o reconozca a L , es decir, tal que $L(M) = L$.

2. Dado un autómata M determinar el lenguaje aceptado por M .

Más adelante se demostrará, en toda su generalidad, que estos problemas *siempre* tienen solución. Consideremos inicialmente problemas del primer tipo.

Ejemplo

$L = a^* = \{\lambda, a, a^2, a^3, \dots\}$. AFD M tal que $L(M) = L$:

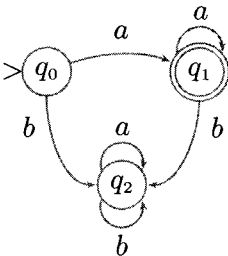


Versión simplificada:

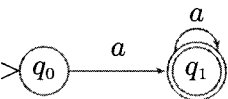


Ejemplo

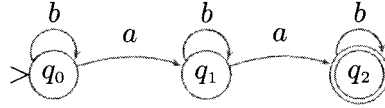
$L = a^+ = \{a, a^2, a^3, \dots\}$. AFD M tal que $L(M) = L$:



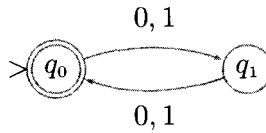
Versión simplificada:



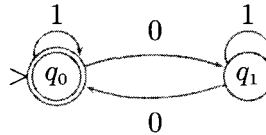
Ejemplo $\Sigma = \{a, b\}$. L = lenguaje de las cadenas que contienen exactamente dos a 's = $b^*ab^*ab^*$. AFD M tal que $L(M) = L$:



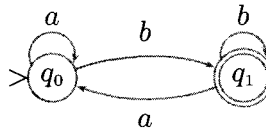
Ejemplo $\Sigma = \{0, 1\}$. L = lenguaje de las cadenas sobre Σ que tienen un número par de símbolos (cadenas de longitud par). AFD M tal que $L(M) = L$:



Ejemplo $\Sigma = \{0, 1\}$. L = lenguaje de las cadenas sobre Σ que contienen un número par de ceros. AFD M tal que $L(M) = L$:



Ejemplo $\Sigma = \{a, b\}$. L = lenguaje de las cadenas sobre Σ que terminan en b . AFD M tal que $L(M) = L$:



Ejercicios de la sección 2.3

① Diseñar autómatas finitos deterministas que acepten los siguientes lenguajes:

- (i) $\Sigma = \{0, 1\}$. L = lenguaje de las cadenas sobre Σ de longitud impar.

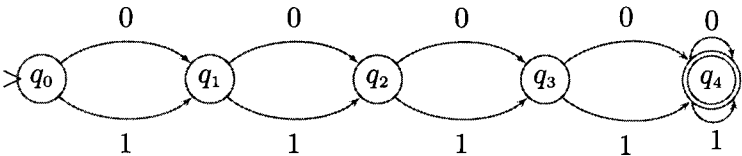
- (ii) $\Sigma = \{0, 1\}$. L = lenguaje de las cadenas sobre Σ que contienen un número impar de unos.
- (iii) $\Sigma = \{a, b\}$. $L = ab^+$.
- (iv) $\Sigma = \{a, b\}$. $L = ab^* \cup ab^*a$.
- (v) $\Sigma = \{0, 1\}$. $L = (0 \cup 10)^*$.
- (vi) $\Sigma = \{0, 1\}$. $L = (01 \cup 10)^*$.
- (vii) $\Sigma = \{0, 1\}$. Lenguaje de todas las cadenas que no contienen dos unos consecutivos.
- (viii) $\Sigma = \{a, b\}$. $L = \{a^{2i}b^{3j} : i, j \geq 0\}$.
- (ix) $\Sigma = \{a, b\}$. L = lenguaje de las cadenas sobre Σ que contienen un número par de *aes* y un número par de *bes*. Ayuda: utilizar 4 estados.
- (x) $\Sigma = \{a, b\}$. Para cada combinación de las condiciones “par” e “impar” y de las conectivas “o” e “y”, diseñar un AFD que acepte el lenguaje L definido por

L = lenguaje de las cadenas con un número par/impar de *aes* y/o un número par/impar de *bes*.

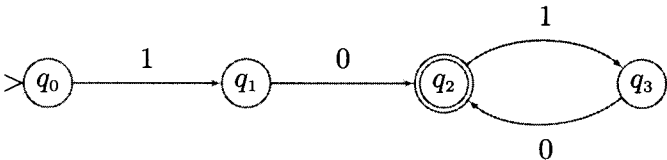
Ayuda: utilizar el autómata de 4 estados diseñado en el ejercicio anterior, modificando adecuadamente el conjunto de estados finales.

- ② Determinar los lenguajes aceptados por los siguientes AFD. Describir los lenguajes ya sea por medio de una propiedad característica o de una expresión regular.

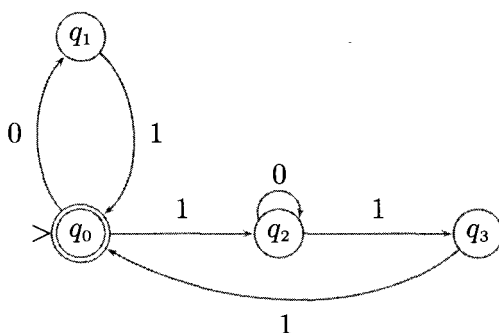
(i)



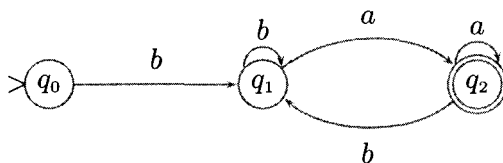
(ii)



(iii)



(iv)



2.4. Autómatas finitos no-deterministas (AFN)

Los **autómatas finitos no-deterministas** (AFN) se asemejan a los AFD, excepto por el hecho de que para cada estado $q \in Q$ y cada $a \in \Sigma$, la transición $\delta(q, a)$ puede consistir en más de un estado o puede no estar definida. Concretamente, un AFN está definido por $M = (\Sigma, Q, q_0, F, \Delta)$ donde:

1. Σ es el alfabeto de cinta.
2. Q es un conjunto (finito) de estados internos.
3. $q_0 \in Q$ es el estado inicial.
4. $\emptyset \neq F \subseteq Q$ es el conjunto de estados finales o estados de aceptación.
- 5.

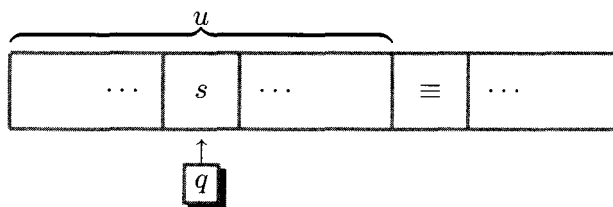
$$\begin{aligned} \Delta : Q \times \Sigma &\longrightarrow \wp(Q) \\ (q, s) &\longmapsto \Delta(q, s) = \{q_{i_1}, q_{i_2}, \dots, q_{i_k}\} \end{aligned}$$

donde $\wp(Q)$ es el conjunto de subconjunto de Q .

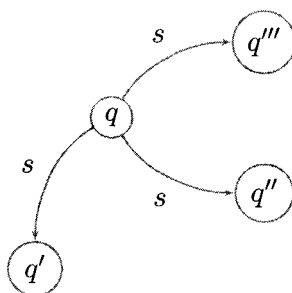
El significado de $\Delta(q, s) = \{q_{i_1}, q_{i_2}, \dots, q_{i_k}\}$ es el siguiente: estando en el estado q , en presencia del símbolo s , la unidad de control puede pasar (aleatoriamente) a uno cualquiera de los estados $q_{i_1}, q_{i_2}, \dots, q_{i_k}$, después de lo cual se desplaza a la derecha.

Puede suceder que $\Delta(q, s) = \emptyset$, lo cual significa que, si durante el procesamiento de una cadena de entrada u , M ingresa al estado q leyendo sobre la cinta el símbolo s , el cómputo se aborta.

Cómputo abortado:



La noción de diagrama de transiciones para un AFN se define de manera análoga al caso AFD, pero puede suceder que desde un mismo nodo (estado) salgan dos o más arcos con la misma etiqueta:



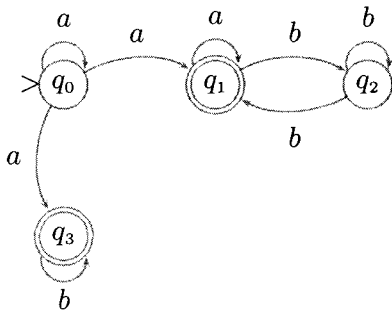
Un AFN M puede procesar una cadena de entrada $u \in \Sigma^*$ de varias maneras. Sobre el diagrama de transiciones del autómata, esto significa que pueden existir varias trayectorias, desde el estado q_0 , etiquetadas con los símbolos de u .

La siguiente es la noción de aceptación para autómatas no-deterministas:

$L(M) = \text{lenguaje aceptado o reconocido por } M$ $= \{u \in \Sigma^* : \text{existe por lo menos un cómputo completo de } u \text{ que termina en un estado } q \in F\}$

Es decir, para que una cadena u sea aceptada, debe existir algún cómputo en el que u sea procesada completamente y que finalice estando M en un estado de aceptación.

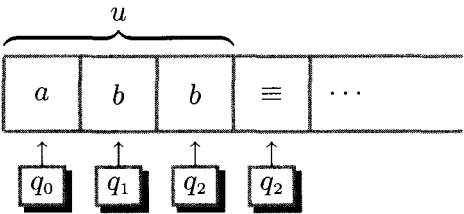
Ejemplo Sea M el siguiente AFN:



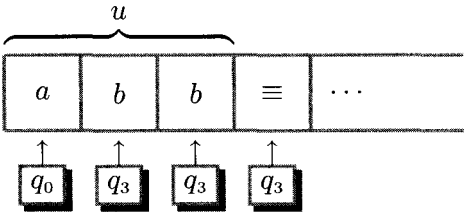
Δ	a	b
q_0	$\{q_0, q_1, q_3\}$	\emptyset
q_1	$\{q_1\}$	$\{q_2\}$
q_2	\emptyset	$\{q_1, q_2\}$
q_3	\emptyset	$\{q_3\}$

Para la cadena de entrada $u = abb$, existen cómputos que conducen al rechazo, cómputos abortados y cómputos que terminan en estados de aceptación. Según la definición de lenguaje aceptado, $u \in L(M)$.

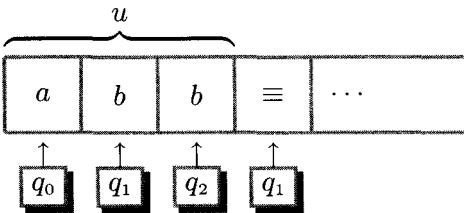
Cómputo de rechazo:



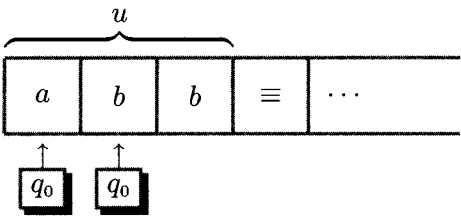
Cómputo de aceptación:



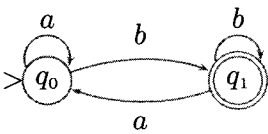
Otro cómputo de aceptación:



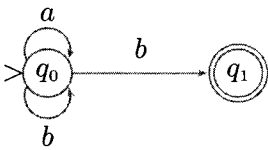
Cómputo abortado:



Ejemplo En el último ejemplo de la sección 2.3 se diseñó el siguiente AFD que acepta el lenguaje de las cadenas sobre $\Sigma = \{a, b\}$ que terminan en b :

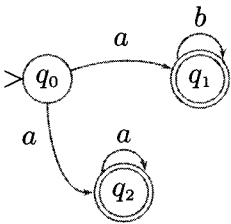


Un AFN que acepta el mismo lenguaje y que es, tal vez, más fácil de concebir, es el siguiente:

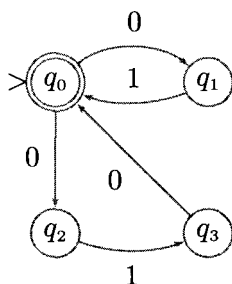


Este autómata se asemeja a la expresión regular $(a \cup b)^*b$.

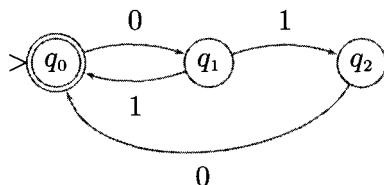
Ejemplo Considérese el lenguaje $L = ab^* \cup a^+$ sobre el alfabeto $\Sigma = \{a, b\}$. El siguiente AFN M satisface $L(M) = L$.



Ejemplo $\Sigma = \{0, 1\}$, $L = (01 \cup 010)^*$. El siguiente AFN acepta a L .



Otro AFN que acepta el mismo lenguaje y que tiene sólo tres estados es el siguiente:



Ejercicios de la sección 2.4

Diseñar autómatas AFD o AFN que acepten los siguientes lenguajes:

- ① $\Sigma = \{a, b\}$. $L = ab^+a^*$.
- ② $\Sigma = \{a, b\}$. $L = a(a \cup ab)^*$.
- ③ $\Sigma = \{a, b, c\}$. $L = a^*b^*c^*$.
- ④ $\Sigma = \{0, 1, 2\}$. $L =$ lenguaje de las cadenas sobre Σ que comienzan con 0 y terminan con 2.
- ⑤ $\Sigma = \{0, 1\}$. Lenguaje de las cadenas de longitud par ≥ 2 formadas por ceros y unos alternados.
- ⑥ $\Sigma = \{0, 1\}$. Lenguaje de las cadenas que tienen a lo sumo dos ceros consecutivos.
- ⑦ $\Sigma = \{0, 1\}$. Lenguaje de las cadenas de longitud impar que tienen unos únicamente en las posiciones impares.
- ⑧ $\Sigma = \{a, b, c\}$. $L =$ lenguaje de las cadenas sobre Σ que contienen la cadena bc .
- ⑨ $\Sigma = \{a, b, c\}$. $L =$ lenguaje de las cadenas sobre Σ que *no* contienen la cadena bc . En el último ejemplo de la sección 1.14 se presentaron dos expresiones regulares para L . Nota: ¿se puede construir un AFD con sólo dos estados para aceptar este lenguaje!

2.5. Equivalencia computacional entre los AFD y los AFN

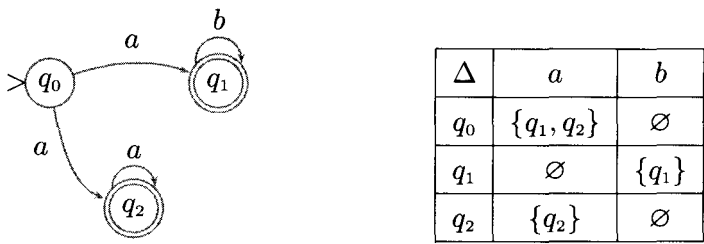
En esta sección se mostrará que los modelos AFD y AFN son computacionalmente equivalentes. En primer lugar, es fácil ver que un AFD $M = (\Sigma, Q, q_0, F, \delta)$ puede ser considerado como un AFN $M' = (\Sigma, Q, q_0, F, \Delta)$ definiendo $\Delta(q, a) = \{\delta(q, a)\}$ para cada $q \in Q$ y cada $a \in \Sigma$. Para la afirmación recíproca tenemos el siguiente teorema.

2.5.1 Teorema. *Dado un AFN $M = (\Sigma, Q, q_0, F, \Delta)$ se puede construir un AFD M' **equivalente a M** , es decir, tal que $L(M) = L(M')$.*

Este teorema, cuya demostración se dará en detalle más adelante, establece que el no-determinismo se puede eliminar. Dicho de otra manera, los autómatas deterministas y los no-deterministas aceptan los mismos lenguajes. La idea de la demostración consiste en considerar cada conjunto de estados $\{p_1, \dots, p_j\}$, que aparezca en la tabla de la función Δ del autómata no-determinista, como un *único* estado del nuevo autómata determinista. La tabla de Δ se completa hasta que no aparezcan nuevas combinaciones de estados. Los estados de aceptación del nuevo autómata son los conjuntos de estados en los que aparece *por lo menos* un estado de aceptación del autómata original. El siguiente ejemplo ilustra el procedimiento.

Ejemplo

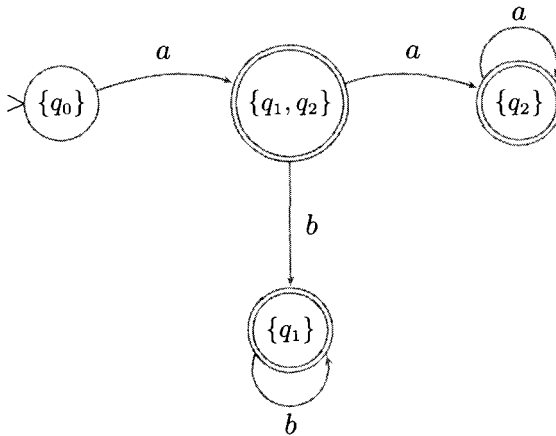
Consideremos el AFN M , presentado en la sección 2.4, que acepta el lenguaje $L(M) = ab^* \cup a^+$ sobre $\Sigma = \{a, b\}$:



El nuevo AFD M' construido a partir de M tiene un estado más, $\{q_1, q_2\}$, y su función de transición δ tiene el siguiente aspecto:

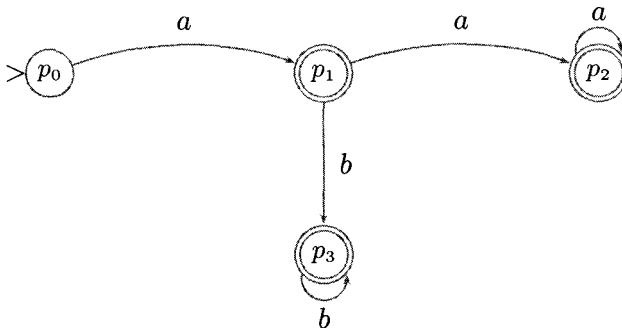
δ	a	b
q_0	$\{q_1, q_2\}$	\emptyset
q_1	\emptyset	$\{q_1\}$
q_2	$\{q_2\}$	\emptyset
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_1\}$

El diagrama de transiciones de este autómata es:



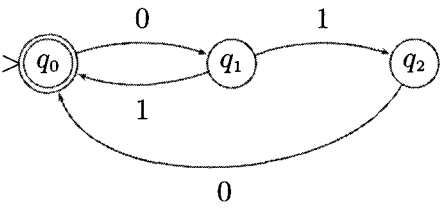
Los estados de aceptación son aquéllos en los que aparezcan q_1 ó q_2 , que son los estados de aceptación del autómata original.

Para mayor simplicidad, podemos cambiar los nombres de los estados del nuevo autómata:



Ejemplo

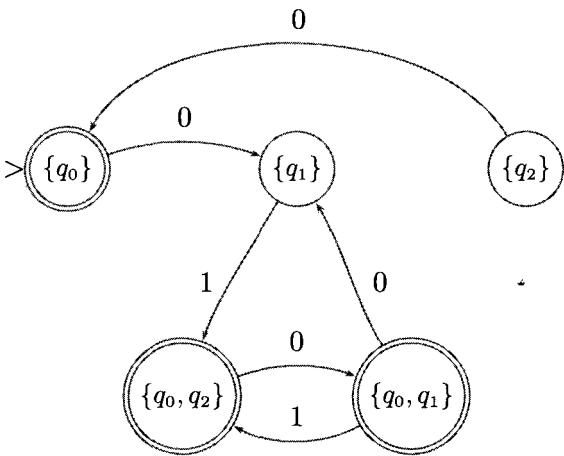
El siguiente AFN M , presentado en la sección 2.4, acepta el lenguaje $L = (01 \cup 010)^*$ sobre $\Sigma = \{0, 1\}$.



La tabla de la función de transición de M se extiende para completar la función δ del nuevo AFN:

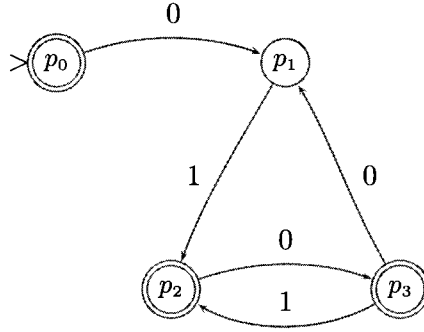
δ	0	1
q_0	$\{q_1\}$	\emptyset
q_1	\emptyset	$\{q_0, q_2\}$
q_2	$\{q_0\}$	\emptyset
$\{q_0, q_2\}$	$\{q_0, q_1\}$	\emptyset
$\{q_0, q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$

El diagrama de transiciones del nuevo autómatas es:



Los estados de aceptación son aquéllos en los que aparezca q_0 ya que q_0 es el único estado de aceptación del autómatas original.

Puesto que el nuevo estado $\{q_2\}$ no interviene en la aceptación de cadenas, el autómatas se puede simplificar en la siguiente forma:



Para la demostración del Teorema 2.5.1, conviene extender la definición de la función de transición, tanto de los autómatas deterministas como de los no-deterministas.

2.5.2 Definición. Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD. La función de transición $\delta : Q \times \Sigma \rightarrow Q$ se extiende a una función $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ por medio de la siguiente definición recursiva:

$$\begin{cases} \hat{\delta}(q, \lambda) = q, & q \in Q, \\ \hat{\delta}(q, a) = \delta(q, a), & q \in Q, a \in \Sigma, \\ \hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a), & q \in Q, a \in \Sigma, w \in \Sigma^*. \end{cases}$$

Según esta definición, para una cadena de entrada $w \in \Sigma^*$, $\hat{\delta}(q_0, w)$ es el estado en el que el autómata termina el procesamiento de w . Por lo tanto, podemos describir el lenguaje aceptado por M de la siguiente forma:

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

Notación. Sin peligro de ambigüedad, la función extendida $\hat{\delta}(q, w)$ se notará simplemente $\delta(q, w)$.

2.5.3 Definición. Sea $M = (\Sigma, Q, q_0, F, \Delta)$ un AFN. La función de transición $\Delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ se extiende inicialmente a conjuntos de estados. Para $a \in \Sigma$ y $S \subseteq F$ se define

$$\Delta(S, a) := \bigcup_{q \in S} \Delta(q, a).$$

Luego se extiende Δ a una función $\hat{\Delta} : Q \times \Sigma^* \longrightarrow \wp(Q)$, de manera similar a como se hace para los AFD. Recursivamente,

$$\begin{cases} \hat{\Delta}(q, \lambda) = \{q\}, & q \in Q, \\ \hat{\Delta}(q, a) = \Delta(q, a), & q \in Q, a \in \Sigma, \\ \hat{\Delta}(q, wa) = \Delta(\hat{\Delta}(q, w), a) = \bigcup_{p \in \hat{\Delta}(q, w)} \Delta(p, a), & q \in Q, a \in \Sigma, w \in \Sigma^*. \end{cases}$$

Según esta definición, para una cadena de entrada $w \in \Sigma^*$, $\hat{\Delta}(q_0, w)$ es el conjunto de los posibles estados en los que terminan los cálculos *completos* de w . Si el cómputo se aborta durante el procesamiento de w , se tendría $\hat{\Delta}(q_0, w) = \emptyset$. Usando la función extendida $\hat{\Delta}$, el lenguaje aceptado por M se puede describir de la siguiente forma:

$$L(M) = \{w \in \Sigma^* : \hat{\Delta}(q_0, w) \text{ contiene un estado de aceptación}\}.$$

Notación. Sin peligro de ambigüedad, la función extendida $\hat{\Delta}(q, w)$ se notará simplemente $\Delta(q, w)$.

Demostración del Teorema 2.5.1:

Dado el AFN $M = (\Sigma, Q, q_0, F, \Delta)$, construimos el AFD M' así:

$$M' = (\Sigma, \wp(Q), \{q_0\}, F', \delta)$$

donde

$$\begin{aligned} \delta : \wp(Q) \times \Sigma &\longrightarrow \wp(Q) \\ (S, a) &\longmapsto \delta(S, a) := \Delta(S, a). \end{aligned}$$

$$F' = \{S \subseteq \wp(Q) : S \cap F \neq \emptyset\}.$$

Se demostrará que $L(M) = L(M')$ probando que, para toda cadena $w \in \Sigma^*$, $\delta(\{q_0\}, w) = \Delta(q_0, w)$. Esta igualdad se demostrará por inducción sobre w . Para $w = \lambda$, claramente se tiene $\delta(\{q_0\}, \lambda) = \Delta(q_0, \lambda) = \{q_0\}$. Para $w = a$, $a \in \Sigma$, se tiene

$$\delta(\{q_0\}, a) = \Delta(\{q_0\}, a) = \Delta(q_0, a).$$

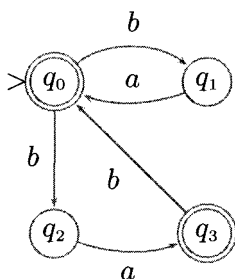
Supóngase (hipótesis de inducción) que $\delta(\{q_0\}, w) = \Delta(q_0, w)$, y que $a \in \Sigma$.

$$\begin{aligned} \delta(\{q_0\}, wa) &= \delta(\delta(\{q_0\}, w), a) && \text{(definición de la extensión de } \delta) \\ &= \delta(\Delta(\{q_0\}, w), a) && \text{(hipótesis de inducción)} \\ &= \Delta(\Delta(\{q_0\}, w), a) && \text{(definición de } \delta) \\ &= \Delta(\{q_0\}, wa) && \text{(definición de la extensión de } \Delta) \\ &= \Delta(q_0, wa) && \text{(definición de la extensión de } \Delta). \quad \square \end{aligned}$$

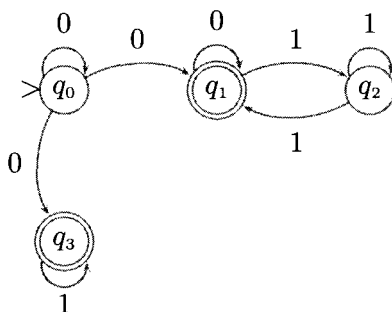
Ejercicios de la sección 2.5

Diseñar AFD equivalentes a los siguientes AFN:

①



②



2.6. Autómatas con transiciones λ (AFN- λ)

Un **autómata finito con transiciones λ** (AFN- λ) es un autómata no-determinista $M = (\Sigma, Q, q_0, F, \Delta)$ en el que la función de transición está definida como:

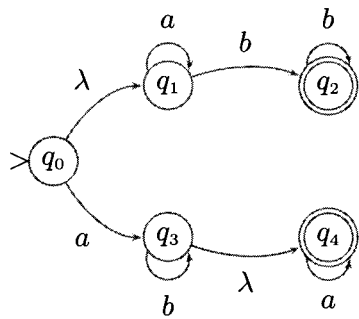
$$\Delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \mathcal{P}(Q).$$

La transición $\Delta(q, \lambda) = \{p_{i_1}, \dots, p_{i_n}\}$, llamada **transición λ** , **transición nula** o **transición espontánea**, tiene el siguiente significado computacional: estando en el estado q , el autómata puede cambiar a uno cualquiera de los estados p_{i_1}, \dots, p_{i_n} , independientemente del símbolo leído y sin mover la unidad de control. Dicho de otra manera, las transiciones λ permiten al autómata cambiar internamente de estado sin procesar o “consumir” el símbolo leído sobre la cinta.

En el diagrama del autómata, las transiciones λ dan lugar a arcos con etiquetas λ . Una cadena de entrada w es aceptada por un AFN- λ si existe por lo menos una trayectoria, desde el estado q_0 , cuyas etiquetas son exactamente los símbolos de w , intercalados con cero, uno o más λ s.

En los autómatas AFN- λ , al igual que en los AFN, puede haber múltiples cálculos para una misma cadena de entrada, así como cálculos abortados.

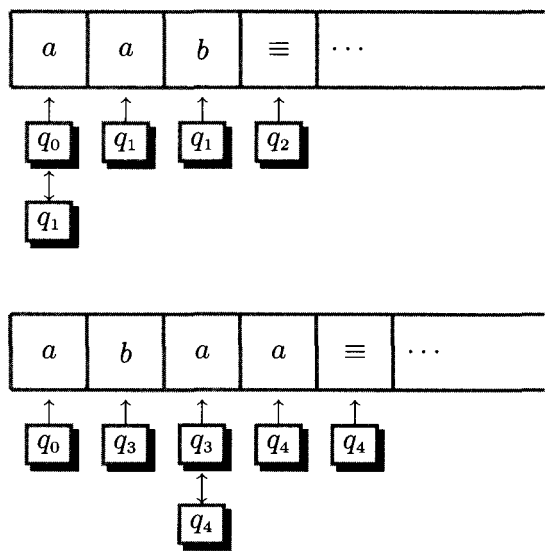
Ejemplo M :



Ejemplos de cadenas aceptadas por M :

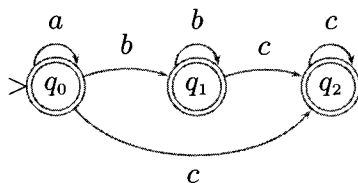
- $u = aab$
- $v = abaa$
- $w = abbaa$

Cálculos de aceptación de $u = aab$ y $v = abaa$:

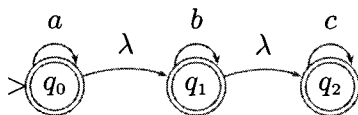


Los AFN- λ permiten aún más libertad en el diseño de autómatas, especialmente cuando hay numerosas concatenaciones.

Ejemplo $\Sigma = \{a, b, c\}$. $L = a^*b^*c^*$. AFD que acepta a L :



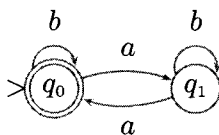
AFN- λ que acepta a L :



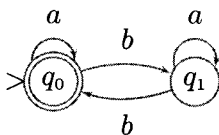
Este autómata se asemeja a la expresión regular $a^*b^*c^*$: las concatenaciones han sido reemplazadas por transiciones λ .

Ejemplo $\Sigma = \{a, b\}$. $L =$ lenguaje de todas las cadenas sobre Σ que tienen un número par de a es o un número par de b es.

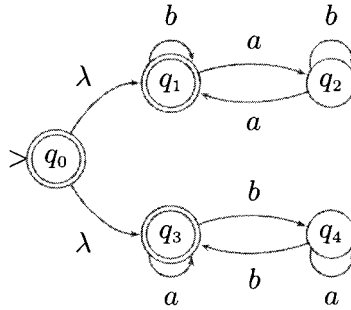
AFD que acepta el lenguaje de las cadenas con un número par de a es:



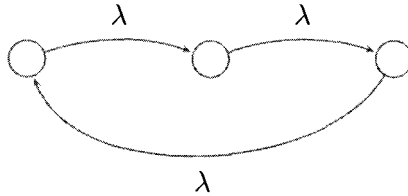
AFD que acepta el lenguaje de las cadenas con un número par de b es:



AFN- λ que acepta el lenguaje de las cadenas con un número par de a es o un número par de b es:



A diferencia de los AFD y los AFN, en los AFN- λ pueden existir “cómputos infinitos”, es decir cómputos que nunca terminan. Esto puede suceder si el autómata ingresa a un estado desde el cual haya varias transiciones λ encadenadas que retornen al mismo estado, como por ejemplo:



Ejercicios de la sección 2.6

Diseñar AFN- λ que acepten los siguientes lenguajes:

- ① $(ab \cup b)^*ab^*$, sobre $\Sigma = \{a, b\}$.
- ② $a(a \cup c)^*b^+$, sobre $\Sigma = \{a, b, c\}$.
- ③ $ab^* \cup ba^* \cup b(ab \cup ba)^*$, sobre $\Sigma = \{a, b\}$.
- ④ $ab^*ba^*b(ab \cup ba)^*$, sobre $\Sigma = \{a, b\}$.
- ⑤ $(0 \cup 010)^*0^*(01 \cup 10)^*$, sobre $\Sigma = \{0, 1\}$.
- ⑥ $0^+1(010)^*(01 \cup 10)^*1^+$, sobre $\Sigma = \{0, 1\}$.
- ⑦ $\Sigma = \{a, b\}$. L = lenguaje de todas las cadenas sobre Σ que tienen un número par de a 's y un número par de b 's.

2.7. Equivalencia computacional entre los AFN- λ y los AFN

En esta sección se mostrará que el modelo AFN- λ es computacionalmente equivalente al modelo AFN. O dicho más gráficamente, las transiciones λ se pueden eliminar, añadiendo transiciones que las simulen, sin alterar el lenguaje aceptado.

En primer lugar, un AFN $M = (\Sigma, Q, q_0, F, \Delta)$ puede ser considerado como un AFN- λ en el que, simplemente, hay *cero* transiciones λ . Para la afirmación recíproca tenemos el siguiente teorema.

2.7.1 Teorema. *Dado un AFN- λ $M = (\Sigma, Q, q_0, F, \Delta)$, se puede construir un AFN M' equivalente a M , es decir, tal que $L(M) = L(M')$.*

Bosquejo de la demostración. Para construir M' a partir de M se requiere la noción de **λ -clausura de un estado**. Para un estado $q \in Q$, la λ -clausura de q , notada $\lambda[q]$, es el conjunto de estados de M a los que se puede llegar desde q por 0, 1 o más transiciones λ . Nótese que, en general, $\lambda[q] \neq \Delta(q, \lambda)$. Por definición, $q \in \lambda[q]$. La λ -clausura de un conjunto de estados $\{q_1, \dots, q_k\}$ se define por:

$$\lambda[\{q_1, \dots, q_k\}] := \lambda[q_1] \cup \dots \cup \lambda[q_k].$$

Además, $\lambda[\emptyset] := \emptyset$. Sea $M' = (\Sigma, Q, q_0, F', \Delta')$ donde

$$\begin{aligned} \Delta' : Q \times \Sigma &\longrightarrow \wp(Q) \\ (q, a) &\longmapsto \Delta'(q, a) := \lambda[\Delta(\lambda[q], a)]. \end{aligned}$$

M' simula así las transiciones λ de M teniendo en cuenta todas las posibles trayectorias. F' se define como:

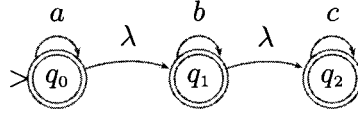
$$F' = \{q \in Q : \lambda[q] \text{ contiene al menos un estado de aceptación}\}.$$

Es decir, los estados de aceptación de M' incluyen los estados de aceptación de M y aquellos estados desde los cuales se puede llegar a un estado de aceptación por medio de una o más transiciones λ . \square

Como se puede apreciar, la construcción de M' a partir de M es puramente algorítmica.

Ejemplo

Vamos a ilustrar el anterior algoritmo con el AFN- λ M , presentado en el segundo ejemplo de la sección 2.6.



$L(M) = a^*b^*c^*$. Las λ -clausuras de los estados vienen dadas por:

$$\lambda[q_0] = \{q_0, q_1, q_2\}.$$

$$\lambda[q_1] = \{q_1, q_2\}.$$

$$\lambda[q_2] = \{q_2\}.$$

La función de transición $\Delta' : Q \times \{a, b, c\} \rightarrow \mathcal{P}(\{q_0, q_1, q_2\})$ es:

$$\Delta'(q_0, a) = \lambda[\Delta(\lambda[q_0], a)] = \lambda[\Delta(\{q_0, q_1, q_2\}, a)] = \lambda[\{q_0\}] = \{q_0, q_1, q_2\}.$$

$$\Delta'(q_0, b) = \lambda[\Delta(\lambda[q_0], b)] = \lambda[\Delta(\{q_0, q_1, q_2\}, b)] = \lambda[\{q_1\}] = \{q_1, q_2\}.$$

$$\Delta'(q_0, c) = \lambda[\Delta(\lambda[q_0], c)] = \lambda[\Delta(\{q_0, q_1, q_2\}, c)] = \lambda[\{q_2\}] = \{q_2\}.$$

$$\Delta'(q_1, a) = \lambda[\Delta(\lambda[q_1], a)] = \lambda[\Delta(\{q_1, q_2\}, a)] = \lambda[\emptyset] = \emptyset.$$

$$\Delta'(q_1, b) = \lambda[\Delta(\lambda[q_1], b)] = \lambda[\Delta(\{q_1, q_2\}, b)] = \lambda[\{q_1\}] = \{q_1, q_2\}.$$

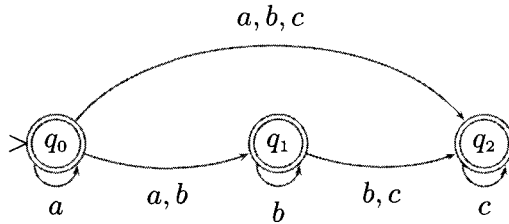
$$\Delta'(q_1, c) = \lambda[\Delta(\lambda[q_1], c)] = \lambda[\Delta(\{q_1, q_2\}, c)] = \lambda[\{q_2\}] = \{q_2\}.$$

$$\Delta'(q_2, a) = \lambda[\Delta(\lambda[q_2], a)] = \lambda[\Delta(\{q_2\}, a)] = \lambda[\emptyset] = \emptyset.$$

$$\Delta'(q_2, b) = \lambda[\Delta(\lambda[q_2], b)] = \lambda[\Delta(\{q_2\}, b)] = \lambda[\emptyset] = \emptyset.$$

$$\Delta'(q_2, c) = \lambda[\Delta(\lambda[q_2], c)] = \lambda[\Delta(\{q_2\}, c)] = \lambda[\{q_2\}] = \{q_2\}.$$

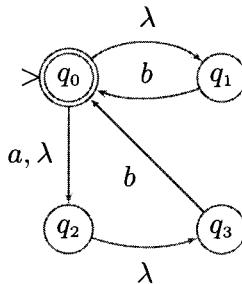
El autómata M' así obtenido es el siguiente:



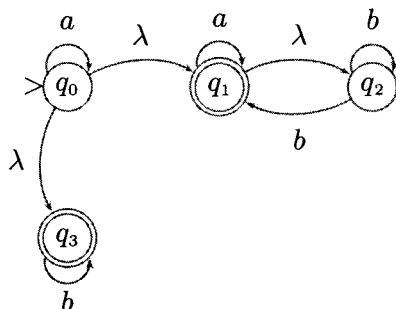
Ejercicios de la sección 2.7

Construir AFN equivalentes a los siguientes AFN- λ :

①



②



2.8. Teorema de Kleene. Parte I

En las secciones anteriores se ha mostrado la equivalencia computacional de los modelos AFD, AFN y AFN- λ , lo cual puede ser descrito en la forma:

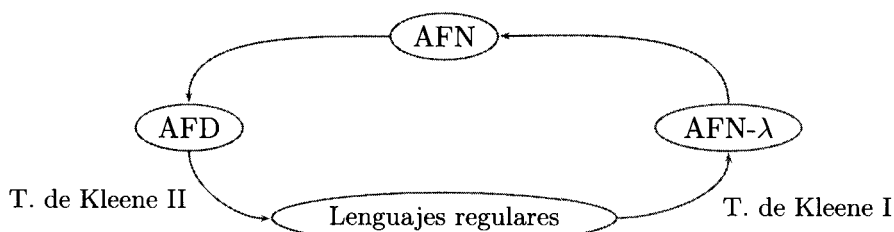
$$\text{AFD} \equiv \text{AFN} \equiv \text{AFN-}\lambda$$

Esto quiere decir que para cada autómata de uno de estos tres modelos se pueden construir autómatas equivalentes en los otros modelos. Por lo tanto, los modelos AFD, AFN y AFN- λ aceptan exactamente los mismos lenguajes. El Teorema de Kleene establece que tales lenguajes son precisamente los lenguajes regulares.

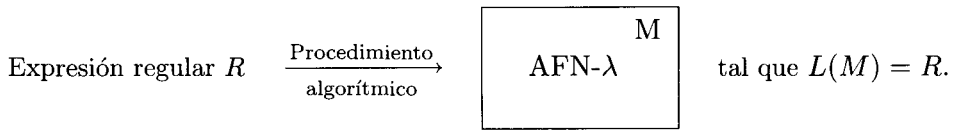
2.8.1. Teorema de Kleene. *Un lenguaje es regular si y sólo si es aceptado por un autómata finito (AFD o AFN o AFN- λ).*

Para demostrar el teorema consideraremos las dos direcciones por separado. Primero demostraremos que para un lenguaje regular L dado existe un AFN- λ tal que $L(M) = L$. En la sección 2.11 demostraremos que, a partir de un AFD M , se puede encontrar una expresión regular R tal que $L(M) = R$. En ambas direcciones las demostraciones son constructivas.

Las construcciones de este capítulo se pueden presentar así:



Parte I. Dada una expresión regular R sobre un alfabeto Σ , se puede construir un AFN- λ M tal que el lenguaje aceptado por M sea exactamente el lenguaje representado por R .



Demostración. Puesto que la definición de las expresiones regulares se hace recursivamente, la demostración se lleva a cabo razonando por inducción sobre R . Para las expresiones regulares básicas, podemos construir fácilmente autómatas que acepten los lenguajes representados. Así, el autómata



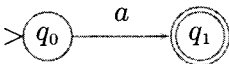
acepta el lenguaje \emptyset , es decir, el lenguaje representado por la expresión regular $R = \emptyset$.

El autómata



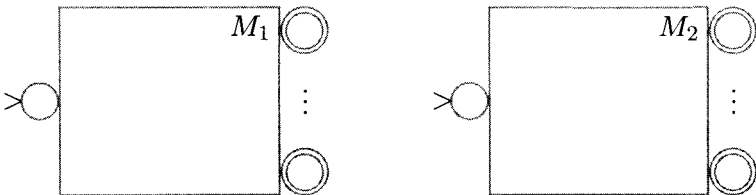
acepta el lenguaje $\{\lambda\}$, es decir, el lenguaje representado por la expresión regular $R = \lambda$.

El autómata



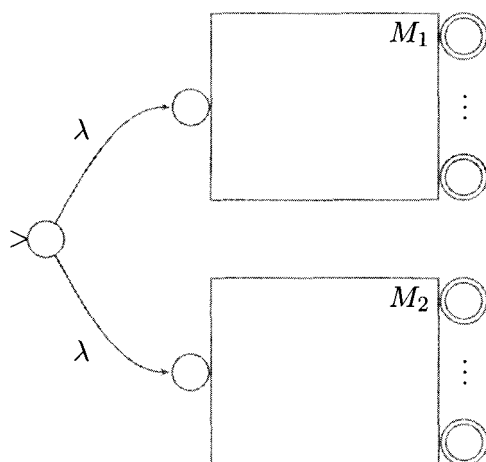
acepta el lenguaje $\{a\}$, $a \in \Sigma$, es decir, el lenguaje representado por la expresión regular $R = a$.

Paso inductivo: supóngase que para las expresiones regulares R y S existen AFN- λ M_1 y M_2 tales que $L(M_1) = R$ y $L(M_2) = S$. Esquemáticamente vamos a presentar los autómatas M_1 y M_2 en la siguiente forma:

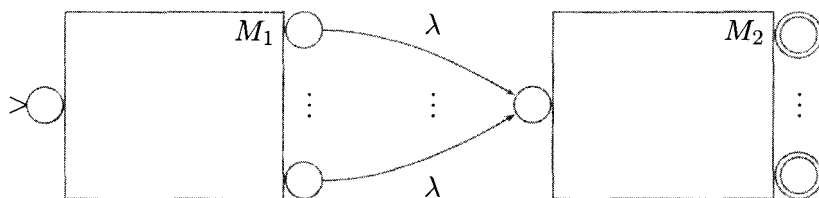


Los estados finales o de aceptación se dibujan a la derecha, pero cabe advertir que el estado inicial puede ser también un estado de aceptación. Obviando ese detalle, podemos ahora obtener AFN- λ que acepten los lenguajes $R \cup S$, RS y R^* .

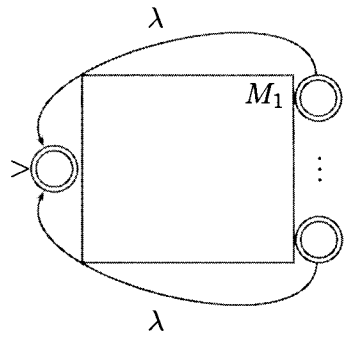
- Autómata que acepta $R \cup S$. Los autómatas M_1 y M_2 se conectan en paralelo y los estados finales del nuevo autómata son los estados finales de M_1 junto con los de M_2 :



- Autómata que acepta RS . Los autómatas M_1 y M_2 se conectan en serie y los estados finales del nuevo autómata son únicamente los estados finales de M_2 :



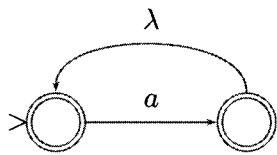
- Autómata que acepta R^* . Los estados finales del nuevo autómata son los estados finales de M_1 junto con el estado inicial.



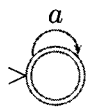
Esto concluye la demostración de la parte I del Teorema de Kleene. En la siguiente sección se presentan ejemplos concretos del procedimiento utilizado en la demostración. □

2.9. Ejemplos de la parte I del Teorema de Kleene

De acuerdo con las construcciones presentadas en la demostración de la parte I del Teorema de Kleene, un AFN- λ que acepta el lenguaje a^* es:



Para simplificar las próximas construcciones utilizaremos, en su lugar, el bucle de un estado:



Ejemplo

 Vamos a utilizar el procedimiento del teorema para construir un AFN- λ que acepte el lenguaje $a^*(ab \cup ba)^* \cup a(b \cup a^*)$ sobre el alfabeto $\{a, b\}$.

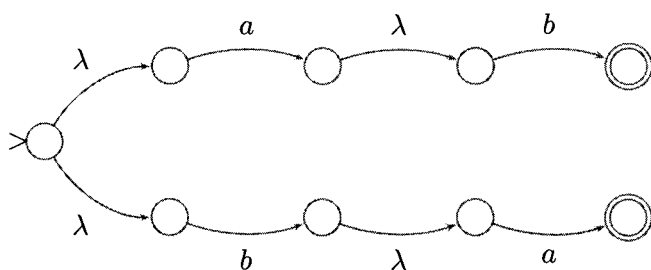
Autómata que acepta ab :



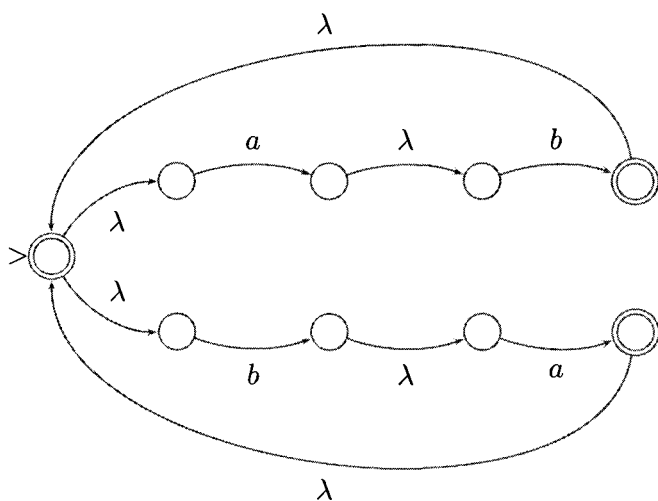
Autómata que acepta ba :



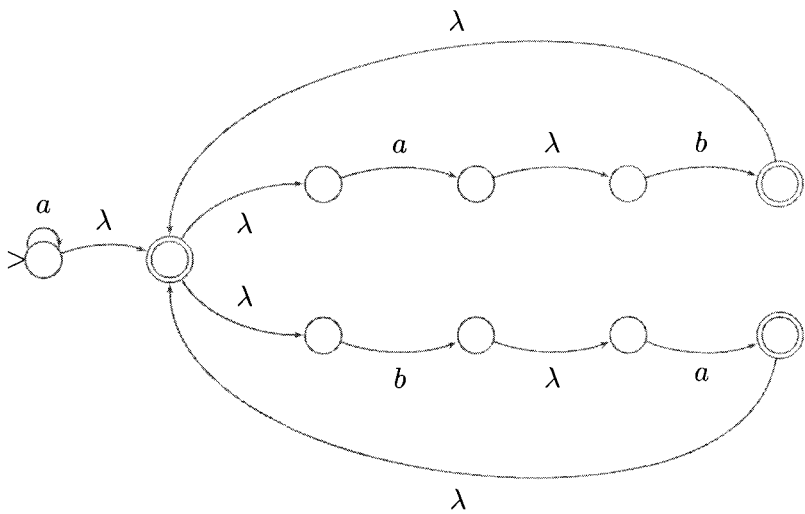
Autómata que acepta $ab \cup ba$:



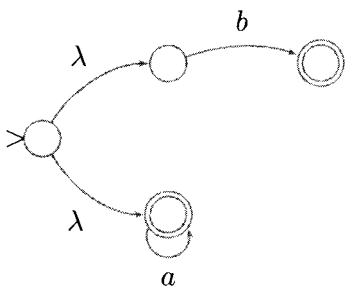
Autómata que acepta $(ab \cup ba)^*$:



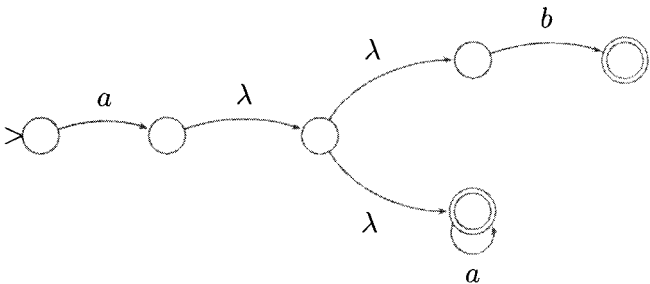
Autómata que acepta $a^*(ab \cup ba)^*$:



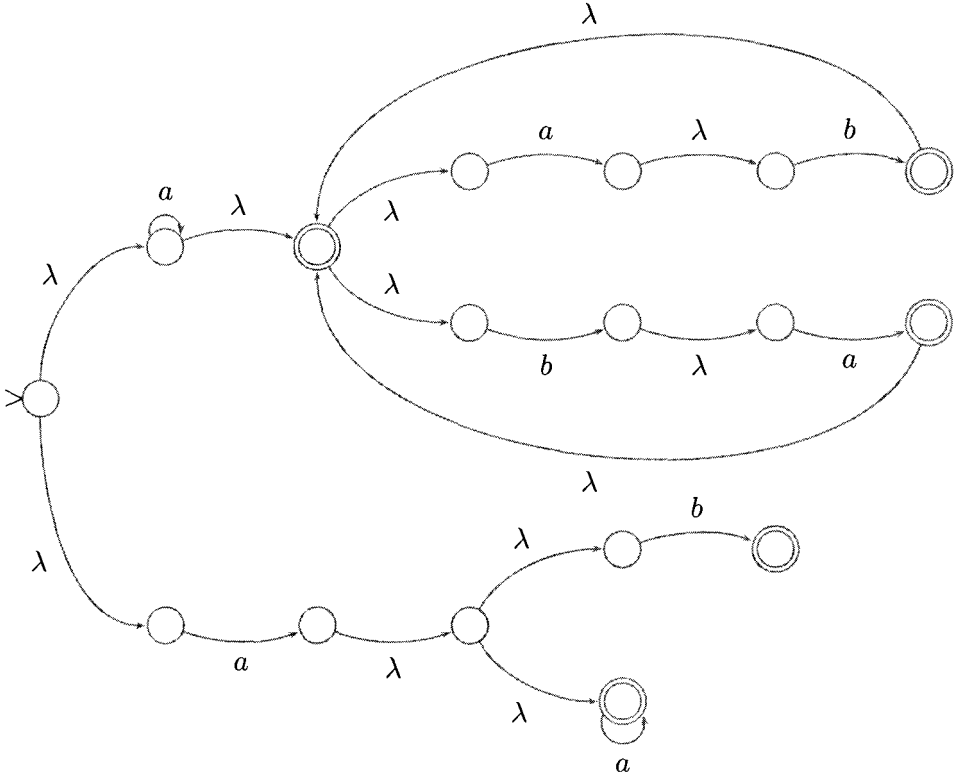
Autómata que acepta $b \cup a^*$:



Autómata que acepta $a(b \cup a^*)$:



Autómata que acepta $a^*(ab \cup ba)^* \cup a(b \cup a^*)$:



Ejercicios de la sección 2.9

Diseñar autómatas AFN- λ que acepten los siguientes lenguajes sobre el alfabeto $\Sigma = \{a, b, c\}$:

- ① $a^*(b \cup ab^* \cup ab^*a)c^* \cup (a \cup b)(a \cup ac)^*$.
- ② $c^*a(a \cup ba)^*(abc)^* \cup c^*(a \cup cb^*c)$.
- ③ $(ac)^* \cup a(a \cup ab^*a) \cup (abc)^*(cba)^* \cup (c \cup ab \cup ba \cup ca)^*(ca \cup cb)^*$.

2.10. Lema de Arden

Vamos a utilizar el siguiente resultado, conocido como “lema de Arden”, para demostrar la segunda parte del Teorema de Kleene.

2.10.1. Lema de Arden. Si A y B son lenguajes sobre un alfabeto Σ y $\lambda \notin A$, entonces la ecuación $X = AX \cup B$ tiene una única solución dada por $X = A^*B$.

Demostración. Si X es una solución de $X = AX \cup B$, entonces $B \subseteq AX \cup B = X$. También se tiene $AX \subseteq X$; a partir de esta contención y usando inducción sobre n , se puede demostrar que $A^n X \subseteq X$ para todo $n \in \mathbb{N}$. Por lo tanto

$$A^n B \subseteq A^n X \subseteq X$$

para todo $n \in \mathbb{N}$. Así que

$$A^*B = \left(\bigcup_{n \geq 0} A^n \right) B = \bigcup_{n \geq 0} A^n B \subseteq X.$$

Esto muestra que toda solución de $X = AX \cup B$ contiene a A^*B y es fácil verificar que, de hecho, A^*B es una solución:

$$A(A^*B) \cup B = A^+B \cup B = (A^+ \cup \lambda)B = A^*B.$$

Para la unicidad, demostraremos que si $A^*B \cup C$, con $C \cap A^*B = \emptyset$, es una solución de la ecuación, entonces $C = \emptyset$.

$$\begin{aligned} A^*B \cup C &= A(A^*B \cup C) \cup B \\ &= A^+B \cup AC \cup B \\ &= (A^+ \cup \lambda)B \cup AC \\ &= A^*B \cup AC. \end{aligned}$$

Intersectando con C ambos lados de la anterior igualdad, se tiene:

$$\begin{aligned} (A^*B \cap C) \cup C &= (A^*B \cap C) \cup (AC \cap C), \\ C &= AC \cap C. \end{aligned}$$

Por lo tanto, $C \subseteq AC$. Si se tuviera $C \neq \emptyset$, existiría una cadena $u \in C$ de longitud mínima. Entonces $u = vw$, con $v \in A$, $w \in C$. Como $\lambda \notin A$, $v \neq \lambda$; por consiguiente $|w| < |u|$. Esta contradicción muestra que necesariamente $C = \emptyset$, tal como se quería. \square

Ejemplo La ecuación $X = aX \cup b^*ab$ tiene solución única $X = a^*b^*ab$.

Ejemplo La ecuación $X = a^2X \cup b^+X \cup ab$ se puede escribir en la forma $X = (a^2 \cup b^+)X \cup ab$. Por el lema de Arden la ecuación tiene solución única $X = (a^2 \cup b^+)^*ab$.

Ejemplo La ecuación $X = ab^2X \cup aX \cup a^*b \cup b^*a$ se puede escribir como $X = (ab^2 \cup a)X \cup (a^*b \cup b^*a)$. Por lema de Arden la ecuación tiene solución única $X = (ab^2 \cup a)^*(a^*b \cup b^*a)$.

Ejercicios de la sección 2.10

① Encontrar las soluciones (únicas) de las siguientes ecuaciones:

(i) $X = aX \cup bX$.

(ii) $X = aX \cup b^*ab \cup bX \cup a^*$.

!② Demostrar de si $\lambda \in A$, entonces Y es una solución de la ecuación $X = AX \cup B$ si y solo si $Y = A^*(B \cup D)$ para algún $D \subseteq \Sigma^*$.

2.11. Teorema de Kleene. Parte II

En esta sección demostraremos que para todo AFN $M = (\Sigma, Q, q_0, F, \Delta)$ existe una expresión regular R tal que $L(M) = R$.

Un autómata tiene un único estado inicial pero cambiando dicho estado surgen nuevos autómatas. Para cada $q_i \in Q$, sea M_i el autómata que coincide con M pero con estado inicial q_i ; más precisamente, $M_i = (\Sigma, Q, q_i, F, \Delta)$. Al lenguaje aceptado por M_i lo denotaremos A_i ; es decir, $L(M_i) = A_i$. En particular, $A_0 = L(M)$. Puesto que los estados de aceptación no se han alterado, se tiene que

$$A_i = \{w \in \Sigma^* : \Delta(q_i, w) \cap F \neq \emptyset\}.$$

Cada A_i se puede escribir como

$$(2.1) \quad A_i = \begin{cases} \bigcup_{a \in \Sigma} \{aA_j : q_j \in \Delta(q_i, a)\}, & \text{si } q_i \notin F, \\ \bigcup_{a \in \Sigma} \{aA_j : q_j \in \Delta(q_i, a)\} \cup \lambda. & \text{si } q_i \in F. \end{cases}$$

Si $Q = \{q_0, q_1, \dots, q_n\}$, las igualdades de la forma (2.1) dan lugar a un sistema de $n + 1$ ecuaciones con $n + 1$ incógnitas (los A_i):

$$\begin{cases} A_0 = C_{01}A_1 \cup C_{02}A_2 \cup \dots \cup C_{0n}A_n & (\cup \lambda) \\ A_1 = C_{11}A_1 \cup C_{12}A_2 \cup \dots \cup C_{1n}A_n & (\cup \lambda) \\ \vdots \\ A_n = C_{n1}A_1 \cup C_{n2}A_2 \cup \dots \cup C_{nn}A_n & (\cup \lambda) \end{cases}$$

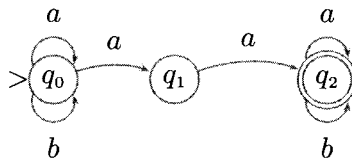
donde cada coeficiente C_{ij} o es \emptyset o es un símbolo de Σ . El término λ se añade a una ecuación solamente si el estado correspondiente es un estado de aceptación.

Utilizando sucesivas veces el lema de Arden, se puede mostrar que este sistema de ecuaciones siempre se puede solucionar y su solución es única. En efecto, comenzando con la última ecuación, se escribe A_n en términos de los demás A_i , para lo cual se usa el lema de Arden si es necesario. Este valor de A_n se reemplaza en las demás ecuaciones y el sistema se reduce a n ecuaciones con n incógnitas. Similarmente, A_{n-1} se escribe en términos de los demás A_i , usando el lema de Arden si es necesario, y tal valor se reemplaza en las ecuaciones restantes. Prosiguiendo de esta manera, el sistema original se reduce a una sola ecuación cuya incógnita es precisamente A_0 . Esta ecuación se soluciona recurriendo una vez más al lema de Arden. Puesto que los coeficientes C_{ij} diferentes de \emptyset son símbolos de Σ , se obtiene una expresión regular R tal que $L(M) = A_0 = R$.

2.12. Ejemplos de la parte II del Teorema de Kleene

A continuación ilustraremos el procedimiento de la sección 2.11 para encontrar $L(M)$ a partir de un AFN $M = (\Sigma, Q, q_0, F, \Delta)$ dado.

Ejemplo Considérese el siguiente AFN M :



Por simple inspección sabemos que $L(M) = (a \cup b)^* a^2 (a \cup b)^*$, pero utilizaremos el método descrito para encontrar explícitamente $L(M)$.

El sistema de ecuaciones asociado con el autómata M es:

$$\begin{cases} (1) & A_0 = aA_0 \cup bA_0 \cup aA_1 \\ (2) & A_1 = aA_2 \\ (3) & A_2 = aA_2 \cup bA_2 \cup \lambda. \end{cases}$$

La ecuación (3) se puede escribir como

$$(4) \quad A_2 = (a \cup b)A_2 \cup \lambda.$$

Aplicando el lema de Arden en (4):

$$(5) \quad A_2 = (a \cup b)^* \lambda = (a \cup b)^*.$$