

Autómatas finitos y lenguajes regulares

Enric Sesa i Nogueras

PID_00216890

Índice

Introducción	5
Objetivos	6
1. Autómatas finitos deterministas y lenguajes regulares	7
1.1. ¿Qué es un autómata finito?	7
1.1.1. Autómatas finitos que reconocen lenguajes	7
1.1.2. Autómatas finitos y algoritmos	11
1.2. Definición formal de un autómata finito determinista (DFA)	11
1.3. Definición formal del lenguaje reconocido por un DFA	13
1.4. Lenguajes regulares y lenguajes reconocidos por DFA	14
1.4.1. Ejemplos de lenguajes regulares	14
1.5. Lenguajes no regulares	17
2. Autómatas finitos indeterministas	19
2.1. ¿Qué es un autómata finito indeterminista (NFA)?	19
2.2. Aceptación de una palabra por un NFA	20
2.3. Ejemplos paradigmáticos de NFA	21
2.4. Definición formal de un NFA	23
2.5. Definición formal del lenguaje reconocido por un NFA	25
2.6. Determinización de autómatas indeterministas	26
2.7. El precio de la determinización	30
3. Operaciones con autómatas finitos	32
3.1. Autómata finito para el lenguaje complementario	32
3.2. Autómata finito para el lenguaje intersección:	
autómata producto cartesiano	33
3.3. Autómata finito para el lenguaje unión	36
3.3.1. Unión por producto cartesiano	36
3.3.2. Unión por adjunción	37
3.4. Autómata finito para la concatenación	39
3.5. Autómata finito para el cierre positivo	42
3.6. Autómata finito para el cierre de Kleene	44
3.7. Autómata finito para el lenguaje inverso	45
4. Minimización de autómatas finitos	47
4.1. Equivalencia de estados	48
4.2. El autómata cociente	49
4.3. Cómo debe construirse el autómata cociente	50
4.4. Unicidad del autómata mínimo	57

5. Expresiones regulares	58
5.1. Definición de <i>expresión regular</i>	58
5.2. Relación entre los lenguajes regulares y las expresiones regulares	59
5.2.1. El teorema de Kleene	59
5.2.2. Construcción del autómata finito que reconoce el lenguaje asociado a una expresión regular	60
5.2.3. Determinación del lenguaje reconocido por un autómata finito: el lema de Arden	62
6. Determinación de la no-regularidad de un lenguaje:	
el lema del bombeo	68
6.1. Una condición necesaria para la regularidad	68
6.2. Una condición suficiente para la no-regularidad	69
Resumen	72
Ejercicios de autoevaluación	75
Solucionario	77
Glosario	94
Bibliografía	94

Introducción

Los autómatas finitos son las “máquinas formales” más simples que la informática teórica considera. Sin embargo, esta simplicidad teórica no los hace ni menos interesantes ni menos útiles, ya sea en los campos más propios de la informática teórica como en los de la informática más práctica.


En el campo de la informática teórica, los autómatas finitos suponen un medio excelente para introducir algunos conceptos, de los cuales esta disciplina hará un uso bastante importante: el concepto de estado, de transición, de máquina reconocedora de un lenguaje, de indeterminismo, etc.

En los campos de la informática menos teórica, los autómatas finitos deterministas reconocedores de lenguajes tienen aplicaciones prácticas muy notables: los compiladores de lenguajes de programación los utilizan para analizar el texto que leen y extraer los elementos léxicos que lo forman; los editores de textos y herramientas similares los utilizan para localizar palabras y modelos dentro del texto que se edita.

Fuera del ámbito restringido del reconocimiento de lenguajes, los autómatas finitos deterministas tienen incontables aplicaciones: en diagramas de transiciones de estados para estudiar y modelar el comportamiento dinámico de los objetos de una clase, en el análisis y la programación orientada a objetos (OO), en el diseño de sistemas digitales, etc.

Un poco más lejos de la informática, los autómatas finitos deterministas son ampliamente utilizados en el control industrial, por ejemplo.

Los **autómatas finitos**, y todo lo que los rodea de manera más inmediata, los **lenguajes regulares** y las **expresiones regulares** son el tema central de este módulo didáctico. El enfoque es eminentemente práctico, a pesar de que en algún momento haremos una breve incursión teórica.

Todos los aspectos tratados están ilustrados con un ejemplo, como mínimo. Intentad no avanzar si antes no los habéis entendido perfectamente. Una buena parte de estos ejemplos tienen una dificultad parecida a la que posteriormente encontraréis en los ejercicios de autoevaluación. Una vez los hayáis entendido, intentad rehacerlos por vuestra cuenta, como si fuesen un ejercicio más. De este modo, la autoevaluación os será aún más provechosa. 

Cuando paséis la última página de este módulo, recordad que los autómatas finitos y los lenguajes regulares sólo son una puerta de entrada al mundo de la informática teórica. Dentro de este mundo quedan aún muchos y muy interesantes aspectos.


Objetivos

El estudio de este módulo didáctico debe permitir al estudiante alcanzar los siguientes objetivos:

1. Comprender la naturaleza propia de los autómatas finitos como reconocedores de lenguajes.
2. Entender los tipos de máquinas y de algoritmos que configuran estas herramientas teóricas.
3. Captar la estrecha relación que se establece entre los autómatas finitos, los lenguajes reguladores y las expresiones regulares.
4. Percibir las posibilidades teóricas del indeterminismo como medio para describir un cálculo.
5. Conocer las propiedades de cierre más importantes de la clase de los lenguajes regulares y utilizarlas para construir sobre el papel autómatas finitos deterministas que reconozcan lenguajes de una complejidad moderada.
6. Percibir la posibilidad de mecanizar con un ordenador los procesos involucrados en la obtención de un autómata finito determinista.
7. Conocer la existencia de lenguajes que no son regulares.
8. Razonar sobre la no-regularidad de algunos lenguajes no regulares.

1. Autómatas finitos deterministas y lenguajes regulares

1.1. ¿Qué es un autómata finito?

Los **autómatas finitos** son el modelo matemático de los sistemas que presentan las siguientes características: 

- 1) En cada momento el sistema se encuentra en un estado, y el conjunto total de estados diferentes en los que se puede encontrar el sistema es finito.
- 2) Pueden responder a un número finito de acontecimientos diferentes.
- 3) El estado en el que se encuentra el sistema resume toda la información referente a todos los acontecimientos pasados.
- 4) La respuesta a un acontecimiento sólo se determina en función del acontecimiento y del estado en que se encuentra el sistema.

Los siguientes son ejemplos de sistemas como los que acabamos de describir:

- Un interruptor (mecánico, electrónico, etc.) biestable. El sistema se puede encontrar en dos estados diferentes (encendido o apagado), y sólo responde al acontecimiento “pulsación” que provoca un cambio de estado y una acción sobre lo que el interruptor controla.
- Un ascensor (o su mecanismo de control). Cada estado se corresponde con un piso y un sentido (subida o bajada). Los acontecimientos a los que puede responder son las peticiones, que provocan cambios de estado (cambio de piso y de sentido).
- Un ordenador también puede ser concebido como un sistema de estas características: cada posible configuración del procesador, de la memoria y de los dispositivos de entrada y salida se corresponden con un estado diferente. Los acontecimientos provienen de los dispositivos de entrada, de las señales de reloj del mismo procesador, etc.

La informática teórica...


... no configura los ordenadores con autómatas finitos, aunque parezca que podría hacerlo. La informática teórica modeliza los ordenadores como dispositivos con una cantidad ilimitada de memoria. Los modelos teóricos de los ordenadores reciben el nombre de *máquinas de Turing*.

1.1.1. Autómatas finitos que reconocen lenguajes

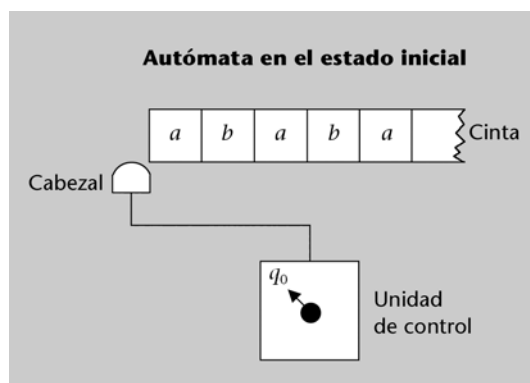
En esta asignatura consideraremos los autómatas finitos como reconocedores de lenguajes. Para nosotros, la tarea modelizada por un autómata finito será responder a la pregunta de si una palabra pertenece a un lenguaje o no.

Cómo hay que imaginarse un autómata finito

Si bien es importante tener presente que los autómatas finitos reconocedores de lenguajes son sólo modelos matemáticos, también es importante que seamos capaces de imaginarlos de una forma que nos facilite la comprensión de su funcionamiento.

La forma más habitual de hacerlo consiste en imaginar a los autómatas como máquinas que constan de una unidad central con un cabezal capaz de leer una cinta sobre la cual se han escrito, de izquierda a derecha, los símbolos de la palabra que se intenta reconocer. 

Inicialmente, la unidad de control está en un estado que denominamos **estado inicial** (q_0), y el cabezal está totalmente a la izquierda de la cinta, sin haber leído todavía ninguno de los símbolos que contiene.

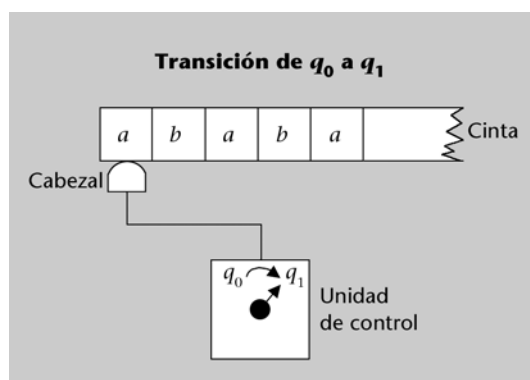


En cada unidad de tiempo, el cabezal se desplaza una posición hacia la derecha y lee el símbolo que se ubica exactamente debajo. La unidad de control determina un nuevo estado a partir del actual y del símbolo leído.

El cambio de estado que se produce cuando se lee un símbolo se denomina **transición**.

Unidad de tiempo

Podéis entender que las unidades de tiempo están marcadas por el "tic" de un reloj interno. Sin embargo, el concepto *tiempo* no es importante para entender lo que es un autómata finito.



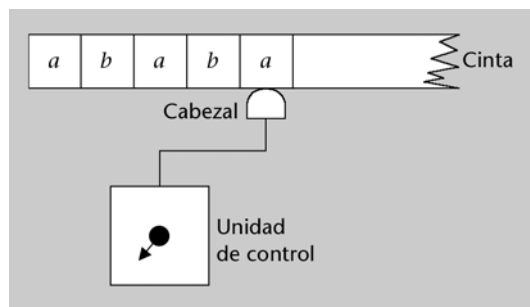
Los estados de la máquina están divididos en dos categorías, los estados llamados **aceptadores** o **finales**, y los estados llamados **no aceptadores**:

a) Cuando el estado en que está la máquina es un estado aceptador, esto significa que la cadena de símbolos que va desde el inicio de la cinta hasta el símbolo que queda exactamente por encima del cabezal forma una palabra que la máquina reconoce como perteneciente a L .

b) Si el estado no es aceptador, la palabra no se reconoce como perteneciente a L .

Finalmente, el cabezal quedará situado debajo del último símbolo de la cinta y ya no avanzará más:

a) Si en este momento la máquina queda en un estado aceptador, esto significa que la palabra entera *abab* ha sido reconocida como una palabra de L .



b) En caso contrario –en caso de que el último estado no sea un estado aceptador–, la palabra no es de L .

Diremos que la palabra ha sido **aceptada** o **rechazada**, respectivamente.

Representación gráfica de autómatas finitos

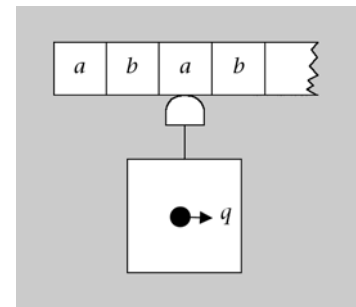
Lo más habitual es que, si las medidas lo permiten, los autómatas finitos se representen gráficamente. La notación gráfica que utilizaremos en este módulo será la siguiente:

1) Los **estados** se representan con círculos, que, si es preciso, llevan en su interior el nombre que se quiere dar al estado para identificarlo.

2) El **estado inicial** se indicará con una pequeña flecha que incida sobre éste.

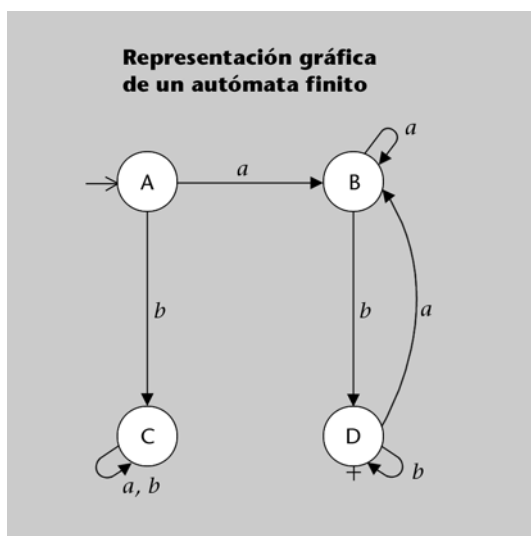
3) Los **estados aceptadores** se indicarán con una pequeña cruz que salga de los mismos.

4) Las **posibles transiciones**, en función de los símbolos leídos, se indicarán con flechas que van de un estado a otro (o a sí mismo). Estas flechas se etiquetarán con el símbolo que provoca el cambio.



Si el estado q es un estado aceptador, la palabra *abab* se reconoce como una palabra de L .

En la figura siguiente se representa gráficamente un autómata finito:



En esta figura...

... podemos observar los siguientes aspectos:

- El estado inicial es el que se denomina A.
- El autómata sólo tiene un estado aceptador, que se denomina D.
- Si está en el estado A y el símbolo leído es *a*, el autómata evoluciona hacia el estado B, mientras que si el símbolo leído es *b*, evoluciona hacia el estado C.
- Si está en el estado B y el símbolo leído es *a*, evoluciona hacia B (se queda donde estaba), mientras que si el símbolo leído es *b*, evoluciona hacia el estado D.
- Para los estados C y D, la interpretación de las transiciones sería similar.

Procesamiento de una palabra por parte de un autómata

Acabaremos este apartado viendo cómo el autómata que acabamos de presentar efectúa el procesamiento de la palabra $w = aabab$:

- 1) Inicialmente el autómata está en el estado inicial A.
- 2) Cuando se lee el símbolo *a*, el autómata evoluciona hacia el estado B.
- 3) Cuando se lee el segundo símbolo *a*, el autómata evoluciona desde B hasta B.
- 4) Cuando se lee el símbolo *b*, el autómata evoluciona desde B hasta D. En este momento se encuentra en un estado aceptador. Esto significa que la parte de la palabra que ha sido procesada hasta ese momento (*aab*) pertenece a *L*.
- 5) Cuando lee el cuarto símbolo (*a*), el autómata evoluciona desde D hasta B. El autómata ha regresado a un estado no aceptador porque la palabra *aaba* no pertenece al lenguaje *L*.
- 6) Finalmente, el autómata lee el último símbolo (*b*) y evoluciona desde B hasta D. En este momento, la palabra ya está completamente procesada y, dado que D es un estado aceptador, también ha sido reconocida como perteneciente a *L*.

Es importante notar que para decir que una palabra es aceptada no es suficiente con que durante su procesamiento el autómata haya pasado por algún estado aceptador. Es necesario que el último estado sea aceptador, independientemente del hecho de si alguno de los anteriores también lo era o no.

No es difícil darse cuenta de que el autómata del ejemplo reconoce el lenguaje de las palabras que empiezan por *a* y acaban en *b*, es decir, $L = \{w \mid \exists x \in \Sigma^* (w = axb)\}$.

Fijaos en que...

... si el autómata pasa por algún estado aceptador antes de acabar el *procesamiento*, esto significa que la palabra que se está procesando contiene subpalabras que pertenecen a *L*.

Cuando decimos...

... que un autómata reconoce un lenguaje, esto significa que reconoce cualquier palabra que pertenezca al lenguaje y que rechaza cualquiera que no pertenezca.


1.1.2. Autómatas finitos y algoritmos

Cualquier autómata finito se puede interpretar como un algoritmo de la manera siguiente:

```
algoritmo autómata_finito;
  var
    c : símbolo;
    e : estado;
    ...
  fvar
    e := estado_inicial;
  mientras hay_símbolos_para_leer() hacer
    c := leer_símbolo();
    e := determinar_nuevo_estado(e, c)
  fmientras;
  si estado_aceptador(e)
    entonces notificar("Palabra aceptada")
    sino notificar("Palabra rechazada")
  fsi
falgoritmo
```

Los autómatas finitos son un modelo de aquellos algoritmos –es decir, se corresponden con ellos– que especifican un cálculo, que se puede realizar con una cantidad de memoria independiente de la longitud de la entrada.

Independientemente de la longitud de la palabra de entrada, la cantidad de memoria que debe consumirse para realizar el procesamiento de la palabra es siempre la misma.

Aquellos lenguajes que, para poder dictaminar si una palabra les pertenece o no, requieren algoritmos que utilizan una cantidad de memoria que depende de la longitud de la palabra, no pueden ser reconocidos por autómatas finitos. 

Observad que...

... interpretado como algoritmo, un autómata finito se corresponde con un típico recorrido de una secuencia: cada símbolo determina un nuevo estado y, al final, se verifica si el estado resultante es aceptador o no lo es.

1.2. Definición formal de un autómata finito determinista (DFA)

Los autómatas con un número finito de estados, que tienen un único estado inicial, y para cada estado una, y sólo una, transición para cada símbolo, se denominan **autómatas finitos deterministas**. Nosotros utilizaremos el acrónimo DFA (*Deterministic Finite Automaton*) para referirnos a ellos.

El acrónimo DFA...

... está mucho más extendido que AFD (autómata finito determinista), a pesar de que este último puede encontrarse en algunos textos.

Una pequeña reflexión sobre lo que acabamos de exponer con respecto a cómo se puede imaginar un DFA y la forma como realiza el procesamiento de una palabra nos permite darnos cuenta de que para poderlo definir completamente es necesario conocer los siguientes elementos: !

- Cuál es el conjunto de estados.
- Cuál de estos estados es el inicial.
- Cuáles de estos estados son aceptadores.
- Cuál es el alfabeto de la palabra que se escribe en la cinta.
- La manera de determinar el nuevo estado en función del símbolo leído y del estado actual.

Así pues, un autómata finito determinista (DFA) se define como un quintuplo de la forma $(Q, \Sigma, q_0, F, \delta)$, donde:

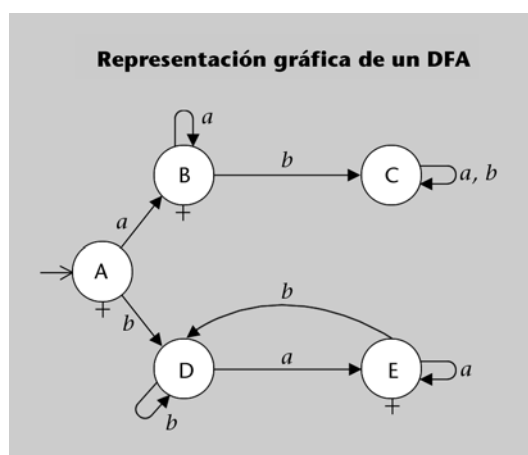
- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada.
- q_0 es el estado inicial (siempre debe ocurrir que $q_0 \in Q$).
- F es el conjunto de estados aceptadores (siempre debe ocurrir que $F \subset Q$).
- δ , llamada *función transición*, es una aplicación de la forma:

$$\delta: Q \times \Sigma \rightarrow Q.$$

En la **función de transición** (δ), dado un estado y un símbolo, el resultado es otro estado.

Así, $\delta(q_i, a) = q_j$ significa que en el estado q_i y en presencia de un símbolo a , el autómata evoluciona hacia el estado q_j .

Como ejemplo de definición de un DFA consideramos el siguiente autómata (sobre $\Sigma = \{a, b\}$):



Su definición formal sería $A = (Q, \Sigma, q_0, F, \delta)$, con:

- $Q = \{A, B, C, D, E\}$.
- $\Sigma = \{a, b\}$.
- $q_0 = A$.
- $F = \{A, B, E\}$.
- δ , definida como indica la siguiente tabla:

δ	a	b
$\rightarrow \dagger A$	B	D
$\dagger B$	B	C
C	C	C
D	E	D
$\dagger E$	E	D

Los símbolos \rightarrow y \dagger ...

... junto con el nombre de un estado, indican que éste es inicial o aceptador, respectivamente.

Gracias a que el número de estados y el número de símbolos de Σ es finito, la función de transición (δ) se puede especificar tabularmente como acabamos de mostrar.

Actividad

1.1. Escribid un algoritmo que lea una palabra carácter a carácter y simule el comportamiento del autómata del ejemplo anterior, indicando al final si la palabra es aceptada o no.


1.3. Definición formal del lenguaje reconocido por un DFA

Para dar una definición precisa de lo que se entiende por *lenguaje aceptado por un autómata*, se comienza por definir una nueva función, $\tilde{\delta}$, a partir de la función de transición δ :

$$\tilde{\delta} : Q \times \Sigma^* \rightarrow Q,$$

definida como:

$$\tilde{\delta}(q, w) = \begin{cases} q & \text{si } w = \lambda \\ \delta(\tilde{\delta}(q_0, x), a) & \text{si } w = xa. \end{cases}$$

Esta función se denomina **función de transición extendida**, y su propósito es extender a las palabras (de ahí su nombre) la función de transición. 


En la función de transición extendida,...

... dado un estado y una palabra, el resultado es otro estado.

Utilizando la fórmula de la función transición extendida $\tilde{\delta}$, que acabamos de describir, es bastante fácil definir el lenguaje reconocido por un autómata finito determinista (DFA).

Dado un DFA $A = \{Q, \Sigma, q_0, F, \delta\}$, el lenguaje reconocido por A , notado $L(A)$, se define como:


$$L(A) = \{w \in \Sigma^* \mid \tilde{\delta}(q_0, w) \in F\}.$$

Fijaos en que $L(A)$ es, sencillamente, el conjunto de las palabras que, a partir del estado inicial, provocan que el autómata evolucione hasta alguno de los estados aceptadores. 

1.4. Lenguajes regulares y lenguajes reconocidos por DFA

Lenguaje regular y *lenguaje reconocido* por un DFA son expresiones sinónimas.

Un lenguaje es un **lenguaje regular** si, y sólo si, existe un DFA que lo reconoce.

Los lenguajes regulares son aquellos que pueden ser reconocidos utilizando algoritmos que requieren una cantidad de memoria independiente de la longitud de las palabras. 

1.4.1. Ejemplos de lenguajes regulares

Muchos lenguajes conocidos, como Σ^* , Σ^+ , \emptyset o $\{\lambda\}$, son lenguajes regulares. En este subapartado mostraremos los autómatas finitos deterministas (DFA) que los reconocen, con el doble propósito de demostrar su regularidad y de familiarizarnos con ellos.

Para demostrar que un lenguaje es regular, es suficiente con exhibir un autómata que lo reconozca, porque, por definición, un lenguaje es regular si es reconocido por algún DFA.

no olvidéis...

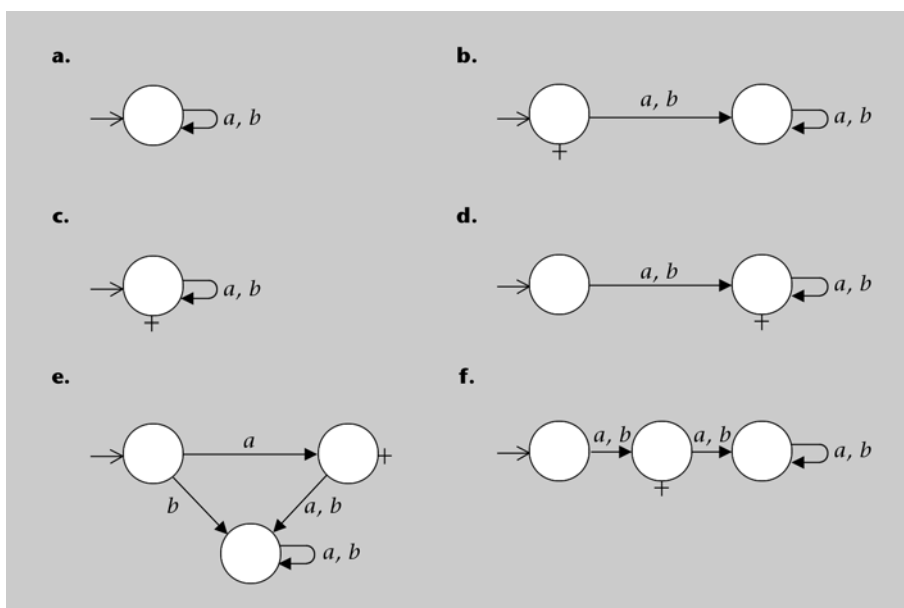
... que un autómata reconoce L si acepta todas las palabras que pertenecen a L y rechaza todas las que no le pertenecen.


Procedimientos de corrección probada

Estrictamente, cuando se exhibe un autómata para demostrar que un determinado lenguaje es regular, también es preciso demostrar que el lenguaje reconocido por el autómata es exactamente el que dice que es. Sin embargo, este tipo de demostraciones suelen ser muy complejas, de manera que en la mayoría de ocasiones se prescinde de ellas. Para suplirlas se construye el autómata siguiendo unos procedimientos de corrección demostrada, de manera que se garantice, siempre que no se cometan errores, que el autómata que se construye reconoce, efectivamente, el lenguaje que se pretende que se reconozca. Veremos estos procedimientos de corrección probada en posteriores apartados.

A continuación presentamos algunos ejemplos de lenguajes regulares, juntamente con el autómata que los reconoce:

- a) $L = \emptyset$ es regular y el DFA que lo reconoce es el que veis en el diagrama a.
- b) $L = \{\lambda\}$ es regular y el DFA que lo reconoce es el que veis en el diagrama b.
- c) $L = \Sigma^*$ es regular y el DFA que lo reconoce es el que veis en el diagrama c.
- d) $L = \Sigma^+$ es regular y el DFA que lo reconoce es el que veis en el diagrama d.
- e) $L = \{a\}$ es regular y el DFA que lo reconoce es el que veis en el diagrama e.
- f) $L = \Sigma = \{a, b\}$ es regular y el DFA que lo reconoce es el que veis en el diagrama f.



Es importante que prestéis atención a los siguientes detalles: 

1) $L = \emptyset$ es aceptado por un autómata sin ningún estado aceptador. Es evidente que un autómata sin ningún estado aceptador no puede aceptar ninguna palabra. Por consiguiente, reconoce \emptyset .

2) Cuando ocurre que $\lambda \in L$, entonces el estado inicial del autómata también es aceptador. Esto es así porque, cuando está en el estado inicial, el autómata aún no ha procesado ningún símbolo, pero precisamente la palabra sin ningún símbolo es λ .

Siempre que $\lambda \in L...$

... el estado inicial también es uno de los estados aceptadores.

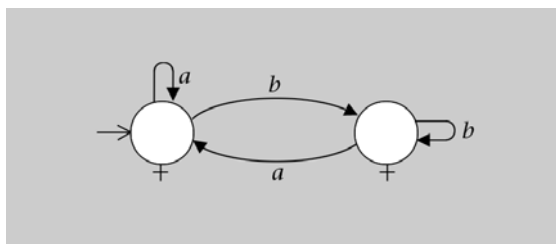
3) En algunos autómatas hay un estado no aceptador del cual no se puede salir nunca (por ejemplo, en el autómata que reconoce $\{\lambda\}$, en el que reconoce $\{a\}$ y en el que reconoce \emptyset). Este estado se denomina **estado pozo**.

Cuando una palabra...

... provoca que el autómata evolucione hasta un estado pozo, esta palabra será rechazada con total seguridad.

Muchas veces hablamos del “autómata que reconoce un lenguaje” como si para cada lenguaje regular hubiera un único autómata que lo reconociera. No obstante, la realidad es otra: cada lenguaje regular es reconocido por una infinidad de autómatas diferentes.

Por ejemplo, el lenguaje regular $L = \Sigma^*$ también es reconocido por el siguiente DFA:



Por tanto, cuando nos referimos al autómata que reconoce un lenguaje, siempre haremos alusión a “uno de los autómatas que reconoce el lenguaje”.

Actividad

1.2. Representad otros autómatas que reconozcan los lenguajes anteriores.

A continuación mostraremos ejemplos de autómatas que reconocen otros lenguajes regulares. Observadlos y no perdáis la oportunidad de entenderlos:

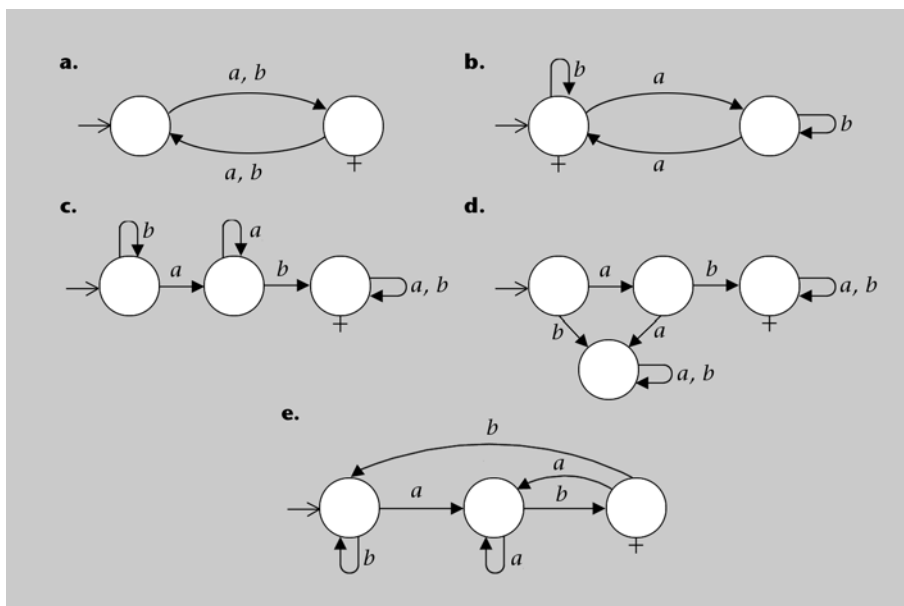
1) Autómata que reconoce el lenguaje de las palabras de longitud impar, $L = \{w \mid |w| = 2n + 1, n \geq 0\}$: consultad el diagrama **a** en la figura siguiente.

2) Autómata que reconoce el lenguaje de las palabras con un número par de símbolos a , $L = \{w \mid |w|_a = 2n, n \geq 0\}$: consultad el diagrama **b** en la figura siguiente.

3) Autómata que reconoce el lenguaje de las palabras que contienen la subpalabra ab , $L = \{w \mid \exists x, y \in \Sigma^* (w = xaby)\}$: consultad el diagrama **c** en la figura siguiente.

4) Autómata que reconoce el lenguaje de las palabras que empiezan por el prefijo ab , $L = \{w \mid \exists x \in \Sigma^* (w = abx)\}$: consultad el diagrama **d** en la figura siguiente.

5) Autómata que reconoce el lenguaje de las palabras que acaban con el sufijo ab , $L = \{w \mid \exists x \in \Sigma^* (w = xab)\}$: consultad el diagrama e en la figura siguiente.



1.5. Lenguajes no regulares

No todos los lenguajes son regulares. Lo cierto es que los lenguajes regulares representan una minoría (infinita) entre todos los lenguajes. Para razonar sobre la existencia de lenguajes que no son regulares, es preciso hacerlo sobre la existencia de lenguajes que no pueden ser reconocidos por ningún DFA. **!**

Concretamente, veremos que hay lenguajes que para ser reconocidos necesitarían autómatas con un número infinito de estados. Uno de estos lenguajes es el de las palabras de la forma $a^n b^n$ con $n \geq 0$ ($L = \{a^n b^n \mid n \geq 0\} = \{a\}^n \{b\}^n$).

Prestad atención al hecho de que para que una palabra pertenezca a L se necesita que el número de símbolos b coincida, exactamente, con el número de símbolos a . Así pues, es indispensable recordar el número de símbolos a para verificar que el número de símbolos b , que se encuentran a continuación, sea exactamente el mismo.

Sin embargo, ¿cómo puede un autómata finito recordar algo? La única respuesta posible es que lo hace mediante sus estados. Efectivamente, cada estado representa el recuerdo de toda la historia pasada que es relevante para determinar la pertenencia de una palabra al lenguaje. **!**

En el caso del lenguaje $L = \{a^n b^n \mid n \geq 0\}$, el número de símbolos a que se han leído es relevante. En consecuencia, sería preciso asignar un estado a cada número posible. Sin embargo, esto es imposible, porque el número de estados debe ser finito.

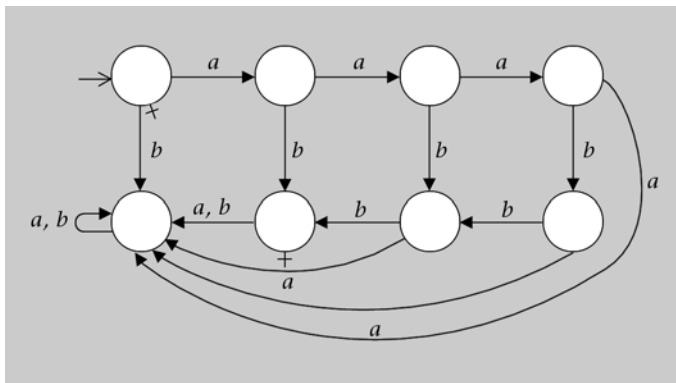
Memoria necesaria para recordar una palabra

Recordad que los lenguajes regulares son aquellos que se pueden reconocer utilizando algoritmos que requieren una cantidad de memoria que es siempre la misma, con independencia de la palabra de entrada. Pues bien, para reconocer el lenguaje $L = \{a^n b^n \mid n \geq 0\}$, es preciso recordar el número de a , para poder dictaminar si la cantidad de b que siguen a las a es igual o no. La cantidad de memoria necesaria para recordar un número, será mayor cuanto mayor sea el número (hasta 225 con 8 bits, hasta 65.535 con 16 bits, etc.). Por consiguiente, la cantidad de memoria necesaria no es, en este caso, independiente de la palabra.

Así pues, ningún autómatas con un número finito de estados puede reconocer $L = \{a^n b^n \mid n \geq 0\}$ y, en consecuencia, L no es regular.

Podéis razonar...

... de manera análoga que lenguajes como $L = \{w \mid |w|_a = |w|_b\}$ o $L = \{w \mid w = w^R\}$ no son regulares.



Autómata que reconoce $L = \{\lambda, ab, aabb, aaabbb\}$

Éste es el menor autómata que reconoce este lenguaje. Prestad atención a lo que pasaría si se quisiera extender para que también aceptara $aaaabbbb$, etc.

Actividad

1.3. Razonando en términos de la cantidad de memoria que se necesita para reconocer los lenguajes, proponed ejemplos de lenguajes que no sean regulares.

2. Autómatas finitos indeterministas

2.1. ¿Qué es un autómata finito indeterminista (NFA)?

Construir directamente el autómata finito determinista (DFA) que reconoce un determinado lenguaje regular no siempre es una tarea sencilla. Para lenguajes relativamente simples, que pueden ser reconocidos por autómatas con un reducido número de estados, esta construcción puede llegar a ser posible. Para lenguajes menos simples, la tarea puede ser muy complicada o, posiblemente, inabordable.

Los **autómatas finitos indeterministas** son una forma especial de autómatas finitos que forman parte del conjunto de herramientas que pueden utilizarse para facilitar la tarea de construcción de DFA. Los llamaremos **NFA** (del inglés *Non-deterministic Finite Automaton*).

El acrónimo NFA...

... está mucho más extendido que **AFI** (autómata finito indeterminista) o **AFN** (autómata finito no-determinista), a pesar de que algunos textos utilizan estos últimos.

En este apartado veremos que:

- 1) Construir un NFA es más simple que construir un DFA.
- 2) Existe un algoritmo que convierte cualquier NFA en un DFA equivalente (que reconoce el mismo lenguaje).

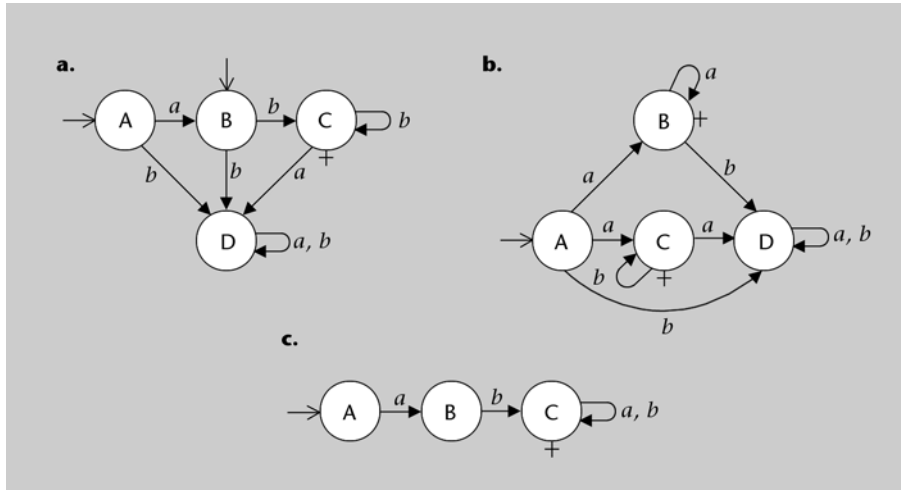
Un autómata es indeterminista cuando se da alguna de estas tres condiciones:

- a) Tiene más de un estado inicial.
- b) Tiene transiciones definidas de forma múltiple.
- c) Tiene transiciones no definidas.

Algunos ejemplos de autómatas indeterministas serían los siguientes (observad los diagramas en la página siguiente):

- 1) El autómata del diagrama **a** tiene dos estados iniciales (A y B).
- 2) El autómata del diagrama **b** tiene transiciones definidas de forma múltiple. En concreto, desde el estado A, con el símbolo *a* hay dos transiciones posibles: una hacia el estado B y otra hacia el estado C.

3) Como último ejemplo, considerad el autómata del diagrama c. Notad que en este autómata hay transiciones no definidas. En concreto, para el estado A no se produce una transición definida para el símbolo b , y para el estado B no hay transición definida para el símbolo a .



2.2. Aceptación de una palabra por un NFA

El proceso de reconocimiento efectuado por un NFA* es similar al que realiza un DFA, pero teniendo en cuenta las siguientes consideraciones: !

* Un NFA sólo es un modelo matemático. Si lo imaginamos como una máquina que efectúa un proceso es para entenderlo mejor.

- 1) Si el autómata tiene más de un estado inicial, entonces el reconocimiento comienza en cualquiera de éstos.
- 2) Si en un estado y para un determinado símbolo hay más de una transición definida, se aplica cualquiera de éstas.
- 3) Si en un estado y para un determinado símbolo no hay transición definida, entonces el proceso de reconocimiento se detiene y la palabra no se acepta.

Está claro que si entendemos un NFA como un autómata finito con capacidad de elección, entonces ocurrirá que algunas palabras serán aceptadas algunas veces y rechazadas otras, en función de cuál haya sido la elección.

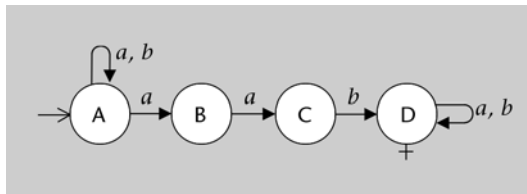
Fijaos en la definición de *palabra aceptada por un NFA* que presentamos a continuación.

Un NFA se puede entender...

... como un autómata que en algunas circunstancias (más de un estado inicial o transición definida de forma múltiple) tiene capacidad de elección.

Una palabra es aceptada por un NFA cuando hay alguna secuencia de transiciones que la hacen evolucionar hasta un estado aceptador. En otras palabras: una palabra es aceptada cuando es posible que lleve al autómata a un estado aceptador.

Por ejemplo, si consideramos el NFA de la siguiente figura:



y la palabra *abaabb*, podemos ver que:

- Si ante el primer símbolo de la palabra el autómatas opta por la transición hacia el estado B, entonces para el segundo símbolo (*b*) no habrá ninguna transición definida.
- Si ante el primer símbolo de la palabra el autómatas opta por no cambiar de estado (transición de A hacia A), entonces la palabra puede hacer evolucionar el autómatas hasta el estado D. La secuencia completa sería AABCDD.

Debemos concluir, pues, que la palabra *abaabb* es aceptada por el NFA del ejemplo anterior porque, como mínimo, hay una secuencia de transiciones que lo lleva a un estado aceptador. Fijaos en que el hecho de que algunas secuencias de transiciones no lo lleven a un estado aceptador no tiene la menor importancia.

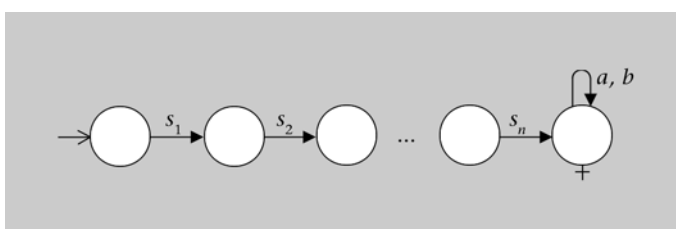
Por el contrario, palabras como *bbb* o *abab* no son reconocidas por este NFA, porque para éstas no hay ninguna secuencia de transiciones que lo lleven a un estado aceptador.

2.3. Ejemplos paradigmáticos de NFA

Desde el punto de vista de la construcción, es mucho más simple construir un NFA que un DFA.

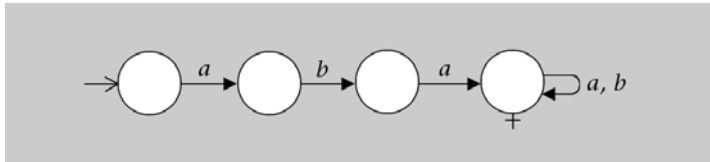
Existen cuatro tipos de lenguajes para los cuales es muy fácil construir los autómatas indeterministas que los reconocen. Veámoslos:

1) Lenguajes que se definen como “palabras que empiezan por un determinado prefijo”. Si el prefijo que caracteriza las palabras del lenguaje es $s_1s_2...s_n$ ($s_i \in \Sigma$), entonces el autómatas indeterminista que reconoce este lenguaje adopta la forma que se muestra a continuación.



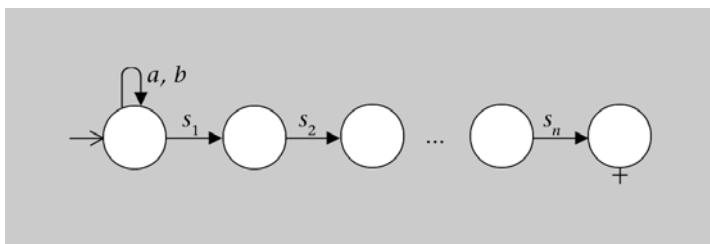
Ejemplo

El autómata indeterminista que reconoce el lenguaje de las palabras que empiezan por el prefijo *aba* ($L = \{w \mid \exists x \in \Sigma^*(w = a b a x)\}$) es el que se representa en la siguiente figura:



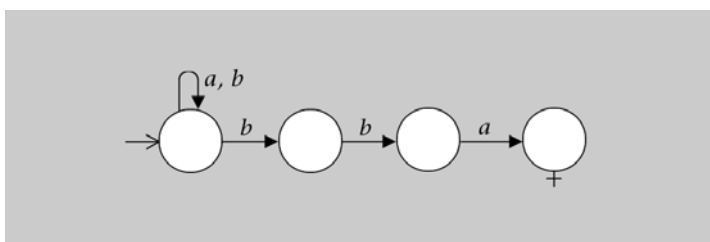
Para entender que, efectivamente, este autómata reconoce el lenguaje indicado, fijaos en que los tres primeros estados sólo “dejan pasar” el prefijo *aba*. Una vez se garantiza la presencia del prefijo deseado, el resto es intrascendente –la palabra es aceptada.

2) Lenguajes que se definen como “palabras que acaban con un determinado sufijo”. Si $s_1 s_2 \dots s_n$ ($s_i \in \Sigma$) es el sufijo que caracteriza las palabras del lenguaje, entonces el autómata indeterminista que reconoce este lenguaje presenta la siguiente forma:



Ejemplo

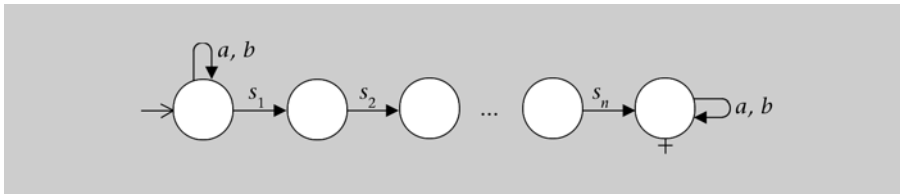
Considerad como ejemplo el lenguaje de todas las palabras que acaban en *bba* ($L = \{w \mid \exists x \in \Sigma^*(w = x b b a)\}$). El autómata indeterminista que lo reconoce es el que representamos a continuación:



Para entender que, efectivamente, este autómata reconoce el lenguaje indicado, notad que para llegar al estado aceptador la secuencia *bba* debe estar presente en la palabra. Sin embargo, fijaos en el hecho de que este estado aceptador no tiene ninguna transición definida y, por tanto, si después de llegar (con *bba*) hay algún símbolo más, el proceso de reconocimiento se detiene y la palabra no se acepta. En conclusión, la secuencia *bba* debe estar al final de la palabra para que ésta sea aceptada.

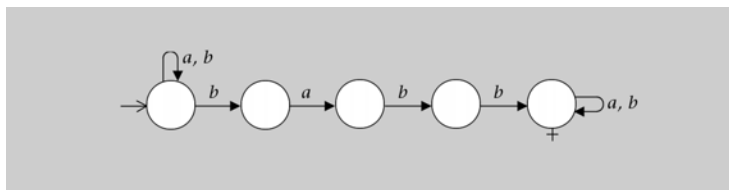
3) Lenguajes que se definen como “palabras que contienen una determinada subpalabra”. En este caso, si $s_1 s_2 \dots s_n$ ($s_i \in \Sigma$) es el factor que deben contener to-

das las palabras del lenguaje, entonces el autómata indeterminista que lo reconoce adopta la forma que se muestra a continuación:



Ejemplo

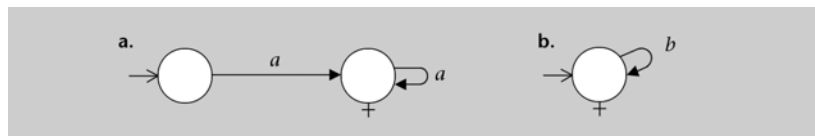
El autómata indeterminista que reconoce el lenguaje de todas las palabras que contienen el factor *babb* ($L = \{w \mid \exists x, y \in \Sigma^*(w = x b a b b y)\}$) es el siguiente:



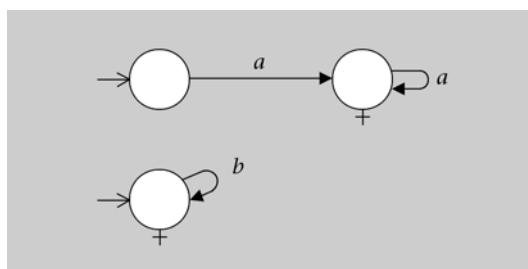
4) Lenguajes que se definen en términos de la unión de otros lenguajes. En este caso, se necesita un NFA (o un DFA) para cada uno de los lenguajes, pero se entenderá que el conjunto es un único NFA con más de un estado inicial.

Ejemplo

Por ejemplo, el lenguaje $L_1 = \{a^n \mid n \geq 1\}$ (cadenas de *a*; como mínimo, una) es reconocido por el autómata indeterminista **a** de la figura, y el lenguaje $L_2 = \{b^p \mid p \geq 0\}$ cadenas de *b* y λ) es reconocido por el autómata indeterminista **b** de la figura:



Pues bien, el lenguaje $L_1 \cup L_2$ es reconocido por el NFA que representamos a continuación:



2.4. Definición formal de un NFA

Un NFA es un quintuplo de la forma $(Q, \Sigma, I, F, \delta)$, donde:


- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada.

La función de transición...

... en el caso indeterminista, dado un estado y un símbolo, da como resultado un conjunto de estados (que puede ser vacío).

- I es el conjunto de los estados iniciales ($I \subset Q$).
- F es el conjunto de estados aceptadores ($F \subset Q$).
- δ es la función de transición y se define de la siguiente manera:

$$\delta: Q \times \Sigma \rightarrow P(Q).$$

Debéis recordar que las **diferencias entre los DFA y los NFA** son las siguientes: 

1) Un DFA tiene un único estado inicial, mientras que un NFA puede tener más de uno.

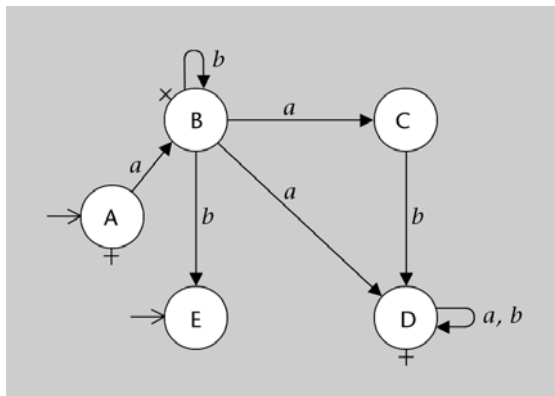
2) En un DFA, la función de transición determina un nuevo estado para cada pareja estado-símbolo; mientras que en un NFA, para cada pareja estado-símbolo, tenemos un conjunto de estados.

$P(Q)$...

... es el conjunto de todos los subconjuntos de Q . Por ejemplo, si $Q = \{A, B, C\}$, entonces $P(Q) = \{\emptyset, \{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\}\}$.

También se le denomina **conjunto de las partes de Q** .

Por ejemplo, para el NFA de la siguiente figura:



tendríamos que: $Q = \{A, B, C, D, E\}$, $\Sigma = \{a, b\}$, $I = \{A, E\}$ y $F = \{A, B, D\}$.

Así, podemos representar nuestro NFA en forma de tabla de transición como la siguiente:

δ	a	b
$\rightarrow \dagger A$	$\{B\}$	\emptyset
$\dagger B$	$\{C, D\}$	$\{B, E\}$
C	\emptyset	$\{D\}$
$\dagger D$	$\{D\}$	$\{D\}$
$\rightarrow E$	\emptyset	\emptyset

2.5. Definición formal del lenguaje reconocido por un NFA

Como paso previo a la definición formal del lenguaje aceptado por un autómata finito determinista (DFA), se ha definido la función de transición extendida ($\tilde{\delta}$). Para definir el lenguaje aceptado por un autómata finito indeterminista (NFA), se procederá de la misma forma.

Hemos visto los lenguajes reconocidos por los DFA en el subapartado 1.3. de este módulo didáctico.



A partir de δ se define la función de transición extendida:

$$(\tilde{\delta}:)P(Q) \times \Sigma^* \rightarrow P(Q)$$

del siguiente modo:

$$\tilde{\delta}(p, w) = \begin{cases} p; & \text{si } w = \lambda, \\ \bigcup_{q \in \tilde{\delta}(p, x)} \delta(q, a); & \text{si } w = xa. \end{cases}$$

En la función de transición extendida,...

... dado un conjunto de estados y una palabra, el resultado es el conjunto de estados a los que la palabra permite llegar desde los estados dados.

Para definir el lenguaje reconocido, nos interesa saber el conjunto de estados al que se llega desde los estados iniciales.


Dado un NFA $A = \{Q, \Sigma, I, F, \delta\}$, el lenguaje reconocido por A se define de la siguiente manera:

$$L(A) = \{w \in \Sigma^* \mid \exists q_f \in F (q_f \in \tilde{\delta}(I, w))\}.$$

Es decir, el lenguaje reconocido es el conjunto de las palabras que desde los estados iniciales (I) pueden llevar al autómata hasta alguno de sus estados finales ($q_f \in F$).

Hemos visto que los lenguajes reconocidos por los autómatas finitos deterministas (DFA) son los llamados *lenguajes regulares*. En este momento, podemos preguntarnos cuáles son los lenguajes reconocidos por los autómatas finitos indeterministas (NFA).

Los lenguajes que los autómatas finitos indeterministas pueden reconocer son exactamente los mismos que se pueden reconocer utilizando autómatas deterministas: los lenguajes regulares.

Así pues, podemos decir que en el caso de los autómatas finitos, el indeterminismo no aporta la capacidad de reconocer más lenguajes, sino la simplicidad de construirlos. 

2.6. Determinización de autómatas indeterministas

Los DFA y los NFA son equivalentes: !

- 1) Por un lado, los DFA pueden considerarse un caso particular de los NFA.
- 2) Por otro lado, cualquier NFA puede convertirse en un DFA que acepta el mismo lenguaje mediante un procedimiento algorítmico que se denomina **determinización**.

Aunque los DFA y los NFA sean equivalentes en el sentido de poder hacer las mismas cosas (reconocer los mismos lenguajes), no son equivalentes en otros sentidos:

- 1) Por un lado, ya hemos visto que la construcción de los NFA es más simple.
- 2) Sin embargo, por otro lado, el indeterminismo es una cualidad poco deseable, ya que no se quieren implementar los “mecanismos que escogen una opción entre varias”.

Por estas razones, en la mayoría de las ocasiones los autómatas se creen indeterministas, se determinizan, y se utiliza su versión determinizada. !

Un DFA...

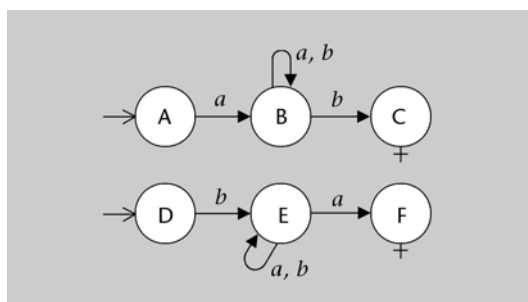
... se puede considerar un NFA con un único estado inicial y con una función de transición que siempre da como resultado un único estado.

El indeterminismo debe entenderse como una herramienta conceptual.

La idea clave que guía el **proceso de determinización de un NFA** es la siguiente: cada estado del autómata determinista se corresponde con un subconjunto de los estados del autómata indeterminista. Concretamente:

- El estado inicial del autómata determinista es el conjunto de los estados iniciales del indeterminista (I).
- Los estados aceptadores del determinista son aquellos que contienen algún aceptador del indeterminista.
- La función de transición del determinista tiene en consideración todas las transiciones definidas para cada uno de los estados del autómata indeterminista que componen un estado del autómata determinista.

Para ver con un ejemplo lo que acabamos de exponer, considerad el siguiente autómata indeterminista:



Veamos la determinización de este autómata paso a paso:

1) En primer lugar, el estado inicial del autónoma determinista se corresponderá con los estados iniciales del indeterminista (A y D); observad el diagrama **a**.

2) La función de transición del estado $\{A, D\}$ con el símbolo a debe recoger las transiciones tanto del estado A como del estado D con este símbolo. Concretamente, la transición para A con a es hacia el estado B ($\delta(A, a) = \{B\}$) y, para D con a no está definida ($\delta(D, a) = \emptyset$). Así, la transición para $\{A, D\}$ con a es hacia el estado $\{B\} \cup \emptyset = \{B\}$; observad el diagrama **b**.

3) Para el estado $\{A, D\}$ y el símbolo b se procede de la misma forma: $\delta(A, b) = \emptyset$ y $\delta(D, b) = \{E\}$. Así, la transición para $\{A, D\}$ con b es hacia el estado $\emptyset \cup \{E\} = \{E\}$; observad el diagrama **c**.

4) El mismo proceso que se ha seguido con el estado $\{A, D\}$ se debe seguir con los nuevos estados que han aparecido: $\{B\}$ y $\{E\}$.

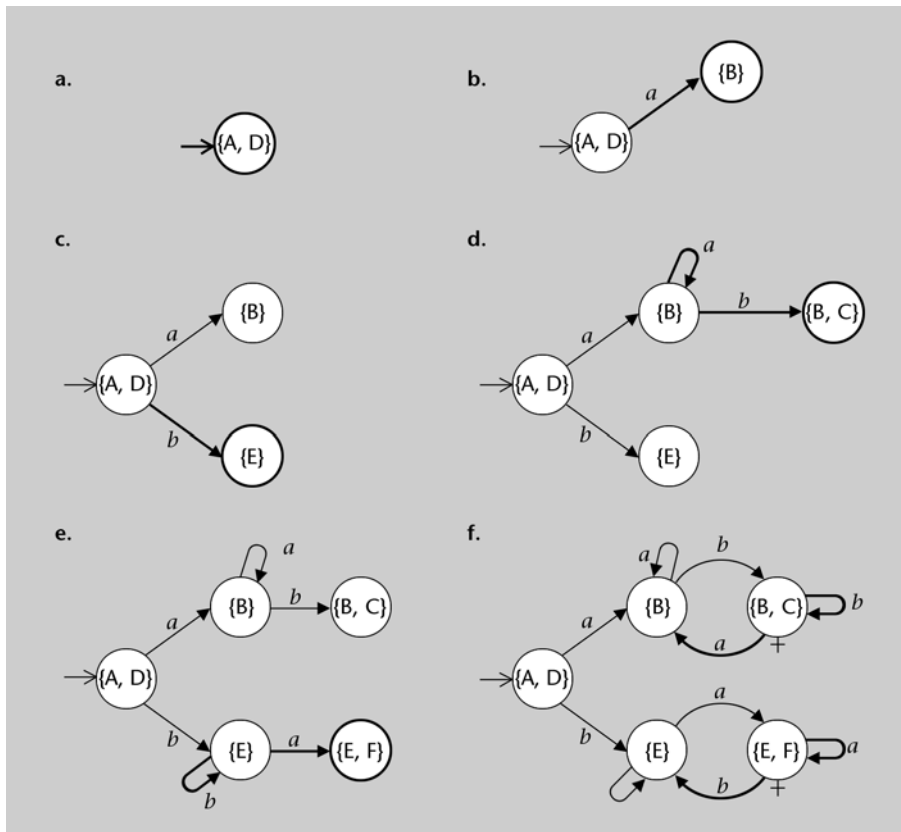
- En relación con $\{B\}$, tenemos que $\delta(B, a) = \{B\}$ y que $\delta(B, b) = \{B, C\}$, como podéis apreciar en la figura. Fijaos en que $\{B\}$ es un estado que ya había aparecido antes (es, precisamente, el que analizamos); observad el diagrama **d**.
- Del mismo modo, para $\{E\}$ tenemos que $\delta(E, a) = \{E, F\}$ y que $\delta(E, b) = \{E\}$; observad el diagrama **e**.

5) El mismo proceso se repite para los estados $\{B, C\}$ y $\{E, F\}$. Concretamente, y como podéis observar en el diagrama **f**, tenemos que:

- $\delta(B, a) = \{B\}$ y $\delta(C, a) = \emptyset$. Así, la transición para $\{B, C\}$ con a es hacia $\{B\}$.
- $\delta(B, b) = \{B, C\}$ y $\delta(C, b) = \emptyset$. Así, la transición para $\{B, C\}$ con b es hacia $\{B, C\}$.
- $\delta(E, a) = \{E, F\}$ y $\delta(F, a) = \emptyset$. Así, la transición para $\{E, F\}$ con a es hacia $\{E, F\}$.
- $\delta(E, b) = \{E\}$ y $\delta(F, b) = \emptyset$. Así, la transición para $\{E, F\}$ con b es hacia $\{E\}$.

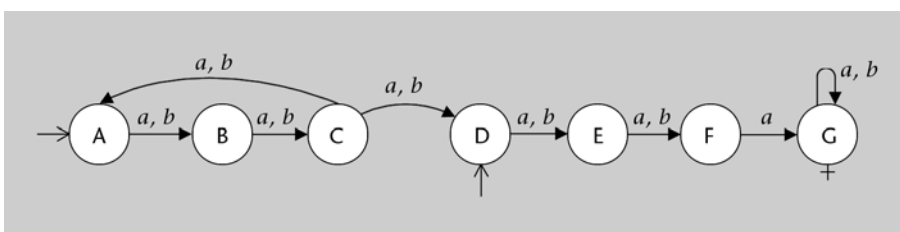
6) En este momento ya no queda ningún nuevo estado por considerar. El autómata resultante es la versión determinista del autómata indeterminista original.

Para concluir, prestad atención al hecho de que $\{B, C\}$ y $\{E, F\}$ han sido marcados como estados aceptadores. Esto es así porque contienen estados aceptadores del autómata indeterminista ($C \in \{B, C\}$ y $F \in \{E, F\}$).



Por regla general, la determinización no se hace construyendo gráficamente el autómata determinista resultante. Lo más habitual es construir la tabla de la función de transición del DFA a partir de la tabla de la función de transición del NFA. **!**

Como ejemplo de construcción de una tabla de la función de transición del DFA, vemos la determinización del autómata que representamos a continuación:



que tiene por función de transición la que se expone en la tabla siguiente:

δ	a	b
$\rightarrow A$	{B}	{B}
B	{C}	{C}
C	{A, D}	{A, D}
$\rightarrow D$	{E}	{E}
E	{F}	{F}
F	{G}	\emptyset
$\dagger G$	{G}	{G}

Iniciamos la construcción de la tabla de la función de transición del autómata determinista a partir del que será su estado inicial: el estado unión de los estados iniciales del autómata indeterminista.

δ	<i>a</i>	<i>b</i>
$\rightarrow\{A, D\}$	{B, E}	{B, E}

Sólo ha sido necesario “unir” las filas correspondientes al estado A y al estado D. A continuación veremos una representación desarrollada:

$\rightarrow A$	{B}	{B}
$\rightarrow D$	{E}	{E}
$\rightarrow\{A, D\}$	{B, E}	{B, E}

Han aparecido dos estados nuevos, {B, E} i {C, F}. Es necesario incorporarlos a la tabla de la función de transición del autómata determinista y calcular sus transiciones:

δ	<i>a</i>	<i>b</i>
$\rightarrow\{A, D\}$	{B, E}	{B, E}
{B, E}	{C, F}	{C, F}
{C, F}	{A, D, G}	{A, D}

Es mejor...

... tener los conjuntos de estados ordenados por orden alfabético, porque eso facilita su comparación y se evitan errores.


El estado {A, D, G} resultante de la unión de las transiciones de los estados C y F con el símbolo *a* no ha aparecido antes en la tabla. Lo incorporamos y obtenemos lo siguiente:

δ	<i>a</i>	<i>b</i>
$\rightarrow\{A, D\}$	{B, E}	{B, E}
{B, E}	{C, F}	{C, F}
{C, F}	{A, D, G}	{A, D}
$\dagger\{A, D, G\}$	{B, E, G}	{B, E, G}

Fijaos en que {A, D, G} es un estado final. Lo es porque contiene G, que es un estado final del autómata indeterminista.


El proceso de incorporar a la tabla los estados nuevos y calcular sus transiciones a partir de la tabla de transiciones del autómata indeterminista se repite hasta que ya no aparece ninguno nuevo más:

δ	a	b
$\rightarrow\{A, D\}$	$\{B, E\}$	$\{B, E\}$
$\{B, E\}$	$\{C, F\}$	$\{C, F\}$
$\{C, F\}$	$\{A, D, G\}$	$\{A, D\}$
$\dagger\{A, D, G\}$	$\{B, E, G\}$	$\{B, E, G\}$
$\dagger\{B, E, G\}$	$\{C, F, G\}$	$\{C, F, G\}$
$\dagger\{C, F, G\}$	$\{A, D, G\}$	$\{A, D, G\}$

No demostraremos formalmente que el lenguaje reconocido por el autómata resultante de la determinización es el mismo que el reconocido por el autómata determinista. Sin embargo, es fácil entender que, efectivamente, los dos autómatas reconocen exactamente el mismo lenguaje. Fijaos en los siguientes puntos: 

1) Si en el autómata determinista una palabra provoca que se evolucione hasta un estado $\{p_1, \dots, p_n\}$, esto significa que en el autómata indeterminista la misma palabra podría hacerlo evolucionar hasta cualquiera de los estados p_1, \dots, p_n .

2) En el autómata determinista, la aceptación se produce cuando se llega a un estado que contiene algunos de los aceptadores del indeterminista. Sin embargo, teniendo en cuenta el punto anterior, si una palabra provoca la evolución hasta un estado aceptador del autómata determinista, esto significa que podría hacer que el autómata indeterminista evolucionara hasta un estado aceptador.

Así pues, podemos darnos cuenta de que las palabras que podrían conseguir que el autómata indeterminista evolucionara hacia un estado aceptador son las mismas que hacen evolucionar al determinista hacia este estado. 

2.7. El precio de la determinización

¿Qué relación se establece entre el número de estados de un autómata indeterminista y el número de estados del autómata determinista obtenido a partir del primero mediante el proceso de determinización?

La respuesta es simple: cada estado del autómata determinista es un conjunto de estados del autómata indeterminista. En otras palabras, cada estado del determinista es un subconjunto del conjunto de estados del indeterminista.

Si el autómata indeterminista tiene N estados, el determinista puede llegar a tener 2^N . El crecimiento del número de estados puede llegar a ser exponencial. Éste es el precio que se deberá pagar para eliminar el indeterminismo.

Un conjunto con N elementos tiene 2^N subconjuntos diferentes.

Sin embargo, hay que tener en cuenta que el elevado número de estados que la determinización puede llegar a suponer no es imputable al proceso de determinización en sí mismo, sino al lenguaje reconocido. Hay algunos lenguajes que sólo pueden ser reconocidos con autómatas deterministas que tienen un número de estados del orden de 2^N , si N es el número de estados del autómata indeterminista que los reconoce.

Un ejemplo de estos lenguajes es $\Sigma^* \{a\} \Sigma^n$. Para un n dado, el autómata indeterminista que lo reconoce tiene $n + 2$ estados, mientras que el menor autómata determinista que también lo reconoce tiene 2^{n+1} .


El crecimiento exponencial es el que tiene lugar en el peor de los casos. No siempre tiene lugar.

En otro apartado estudiaremos un algoritmo que permite minimizar el número de estados de un autómata determinista. Este algoritmo nos ayudará a garantizar que los autómatas que construimos presentan los estados mínimos indispensables.

Consultad el subapartado 4.4. de este módulo didáctico.



3. Operaciones con autómatas finitos

¿La unión de dos lenguajes regulares es también un lenguaje regular? ¿Y la intersección? ¿Y la concatenación? En este apartado damos una respuesta positiva a todas estas cuestiones mostrando que: 

a) Si L es un lenguaje regular, entonces L^c , L^R , L^+ , y L^* también son lenguajes regulares.

b) Si L_1 y L_2 son lenguajes regulares, entonces $(L_1 \cap L_2)$, $(L_1 \cup L_2)$ y $(L_1 L_2)$ también son lenguajes regulares.

Tened cuidado con el sentido de la implicación:

a) Que L^+ o L^* sean lenguajes regulares no significa que L lo sea.

b) Que $(L_1 \cap L_2)$ o $(L_1 \cup L_2)$ o $(L_1 L_2)$ sean regulares no significa que L_1 o L_2 también lo sean.

Se dice que los lenguajes regulares son cerrados para las operaciones de complementación, intersección, unión, concatenación, cierres y inversión. *Cerrados* significa que, si se les aplican estas operaciones, los resultados continúan siendo lenguajes regulares.

En todos los casos, la estrategia será la misma: mostrar cómo se construye el autómata que reconoce el lenguaje a partir de los autómatas que reconocen los lenguajes originales.

Por ejemplo, para mostrar que la intersección de dos lenguajes regulares, L_1 y L_2 es regular, veremos cómo podemos construir un autómata que reconozca el lenguaje $L_1 \cap L_2$, a partir de los autómatas que reconocen los lenguajes L_1 y L_2 .

Notad el aspecto constructivo de esta forma de proceder: no sólo se muestran las propiedades interesantes de los lenguajes regulares, sino que también se proporcionan herramientas para construir autómatas a partir de otros autómatas. Estas herramientas resultan muy útiles porque permiten construir autómatas de gran complejidad a partir de autómatas más simples.

No olvidéis que...

... para demostrar que un lenguaje es regular, es suficiente con tener un autómata que lo reconozca.

3.1. Autómata finito para el lenguaje complementario

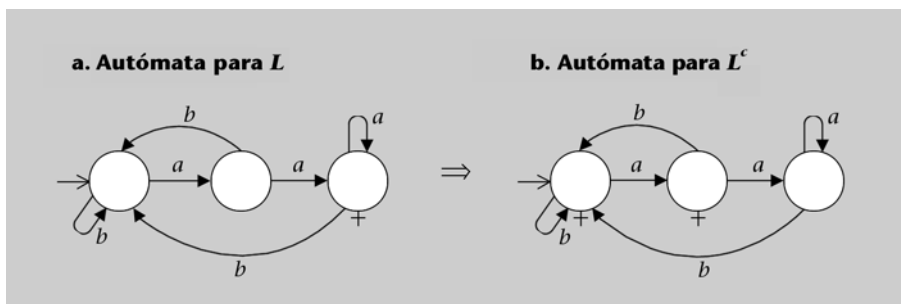
Si L es un lenguaje regular, su complementario, L^c , también lo es.

Si el lenguaje L es reconocido por el DFA $A = (Q, \Sigma, q_0, F, \delta)$, entonces el lenguaje L^c es reconocido por el DFA $A^c = (Q, \Sigma, q_0, Q - F, \delta)$.

Recordad que $L^c = \{w \mid w \notin L\}$.

Si en un DFA se intercambian los estados aceptadores por los no aceptadores y viceversa, se obtiene otro DFA que reconoce el complementario del lenguaje aceptado por el primero.

Por ejemplo, del autómata que reconoce el lenguaje de las palabras que acaban en aa ($L = \{w \mid \exists x \in \Sigma^* ((w = xaa))\}$) se pasa al autómata que reconoce el lenguaje de las palabras que no acaban en aa ($L^c = \{w \mid \forall x \in \Sigma^* ((w \neq xaa))\}$), intercambiando los estados aceptadores por los no aceptadores y los no aceptadores por los aceptadores, como muestra la figura siguiente:



Este modelo sólo puede aplicarse a autómatas deterministas (DFA). Cuando se aplica a autómatas indeterministas (NFA), puede dar resultados incorrectos. !

Actividad

3.1. Hallad un NFA tal que, si lo complementáis, no reconozca el complementario del lenguaje que reconocía antes de complementarlo.

3.2. Autómata finito para el lenguaje intersección: autómata producto cartesiano

Si L_1 y L_2 son lenguajes regulares, su intersección $L_1 \cap L_2$ también es un lenguaje regular.

No olvidéis que
 $L_1 \cap L_2 = \{w \mid (w \in L_1) \wedge (w \in L_2)\}$

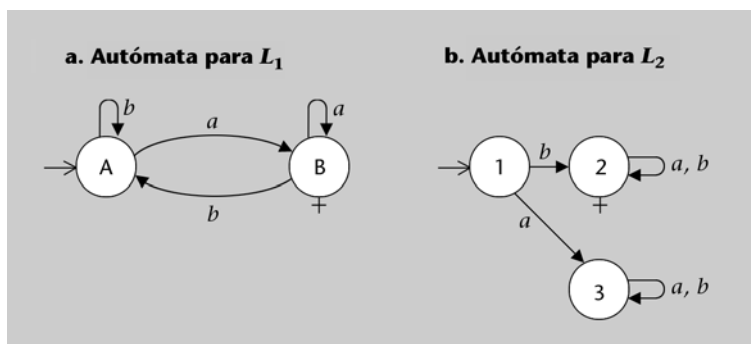
Si el lenguaje L_1 es reconocido por el DFA $A_1 = (Q_1, \Sigma, q_1, F_1, \delta_1)$ y el lenguaje L_2 es reconocido por el DFA $A_2 = (Q_2, \Sigma, q_2, F_2, \delta_2)$; entonces, el lenguaje $L_1 \cap L_2$ es reconocido por el autómata $A = (Q_1 \times Q_2, \Sigma, \langle q_1, q_2 \rangle, F_1 \times F_2, \delta)$ con $\delta: (Q_1 \times Q_2 \times \Sigma \rightarrow Q_1 \times Q_2)$, definida como $\delta(\langle p, q \rangle, a) = \langle \delta_1(p, a), \delta_2(q, a) \rangle$.

El autómata A se denomina **autómata intersección de A_1 y A_2** . Se obtiene a partir del producto cartesiano de los conjuntos de los estados de A_1 y A_2 .

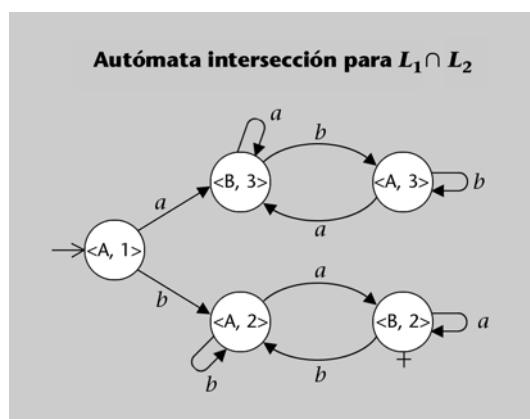
El autómata intersección de A_1 y A_2 simula el reconocimiento de una palabra simultáneamente por A_1 y por A_2 . Fijaos en que: **!**

- Cada estado de A es un par formado por un estado de A_1 y un estado de A_2 . Cuando A está en el estado $\langle p, q \rangle$, esto equivale a decir que A_1 está en el estado p y que A_2 está en el estado q .
- El estado inicial de A es el par formado por el estado inicial de A_1 y el estado inicial de A_2 .
- Los estados aceptadores de A son los pares formados por un estado aceptador de A_1 y un estado aceptador de A_2 . De esta manera, A acepta una palabra si, y sólo si, tanto A_1 como A_2 la aceptan.
- Desde el estado $\langle p, q \rangle$ y con un símbolo cualquiera de Σ , el autómata A evoluciona hacia un estado que tiene, como primer componente, el estado hacia el cual evolucionaría A_1 desde p con aquel símbolo y, como segundo componente, el estado hacia el cual evolucionaría A_2 desde q con el mismo símbolo.

Con el siguiente ejemplo veremos cómo se puede llevar a la práctica la construcción del autómata intersección.



A partir de los autómatas de la figura anterior, que reconocen los lenguajes $L_1 = \{w \mid \exists x \in \Sigma^* (w = xa)\}$ (palabras que acaban en a) y $L_2 = \{w \mid \exists x \in \Sigma^* (w = bx)\}$ (palabras que empiezan en b), se obtiene el siguiente autómata intersección:



Este autómata* reconoce el lenguaje de las palabras que empiezan por b y acaban con a ($L_1 = \{w \mid \exists x \in \Sigma^* (w = bxa)\}$).

* Sin embargo, no se trata del autómata con menos número de estados que reconoce este lenguaje.

Al igual que en el caso de la determinización, la construcción del autómata intersección no suele hacerse gráficamente, sino que lo más habitual es construir la tabla de la función de transición del autómata A a partir de las tablas de las funciones de transición de los autómatas A_1 y A_2 .

Por ejemplo, los autómatas que reconocen los lenguajes $L_1 = \{w \mid \exists x y \in \Sigma^* (w = xaa y)\}$ (palabras que contienen el factor aa) y $L_2 = \{w \mid \exists x \in \Sigma^* (w = xba)\}$ (palabras que acaban con el sufijo ba) disponen de las siguientes tablas para las respectivas funciones de transición:

δ_1	a	b
$\rightarrow A$	B	A
B	C	A
$\dagger C$	C	C

δ_2	a	b
$\rightarrow 1$	1	2
2	3	2
$\dagger 3$	1	2

La construcción del autómata intersección se inicia a partir de su estado inicial: el par formado por el estado inicial del primer autómata y por el estado inicial del segundo.

δ	a	b
$\rightarrow \langle A, 1 \rangle$	$\langle B, 1 \rangle$	$\langle A, 2 \rangle$

Las transiciones se obtienen de las filas de los estado A y 1:


$\rightarrow A$	B	A	
$\rightarrow 1$	1	2	
$\rightarrow \langle A, 1 \rangle$	$\langle B, 1 \rangle$	$\langle A, 2 \rangle$	

Los nuevos estados $\langle B, 1 \rangle$ y $\langle A, 2 \rangle$ se incorporan a la tabla y se calculan sus transiciones:

δ	a	b
$\rightarrow \langle A, 1 \rangle$	$\langle B, 1 \rangle$	$\langle A, 2 \rangle$
$\langle B, 1 \rangle$	$\langle C, 1 \rangle$	$\langle A, 2 \rangle$
$\langle A, 2 \rangle$	$\langle B, 3 \rangle$	$\langle A, 2 \rangle$

Los nuevos estados que aparecen se van añadiendo a la tabla y se calculan sus transiciones hasta que ya no aparecen más:

δ	a	b
$\rightarrow \langle A, 1 \rangle$	$\langle B, 1 \rangle$	$\langle A, 2 \rangle$
$\langle B, 1 \rangle$	$\langle C, 1 \rangle$	$\langle A, 2 \rangle$
$\langle A, 2 \rangle$	$\langle B, 3 \rangle$	$\langle A, 2 \rangle$
$\langle C, 1 \rangle$	$\langle C, 1 \rangle$	$\langle C, 2 \rangle$
$\langle B, 3 \rangle$	$\langle C, 1 \rangle$	$\langle A, 2 \rangle$
$\langle C, 2 \rangle$	$\langle C, 3 \rangle$	$\langle C, 2 \rangle$
$\dagger \langle C, 3 \rangle$	$\langle C, 1 \rangle$	$\langle C, 2 \rangle$

La construcción del producto cartesiano también se podría aplicar, sin ninguna modificación, si alguno –o todos– los autómatas fuesen indeterministas. De todos modos, es recomendable minimizarlos antes para evitar la proliferación de estados. 

3.3. Autómata finito para el lenguaje unión


Si L_1 y L_2 son lenguajes regulares, su unión, $L_1 \cup L_2$, también es regular.

3.3.1. Unión por producto cartesiano

Si el lenguaje L_1 es reconocido por el DFA $A_1 = (Q_1, \Sigma, q_1, F_1, \delta)$ y el lenguaje L_2 es reconocido por el DFA $A_2 = (Q_2, \Sigma, q_2, F_2, \delta)$, entonces el lenguaje $L_1 \cup L_2$ es reconocido por el autómata $A = (Q_1 \times Q_2, \Sigma, \langle q_1, q_2 \rangle, (F_1 \times Q_2) \cup (Q_1 \times F_2), \delta)$ con $\delta: Q_1 \times Q_2 \times \Sigma \rightarrow Q_1 \times Q_2$, definida como $\delta(\langle p, q \rangle, a) = \langle \delta_1(p, a), \delta_2(q, a) \rangle$.

El autómata A se denomina **autómata unión de A_1 y A_2** , que también se obtiene a partir del producto cartesiano de los estados de A_1 y A_2 .

Recordad que $L_1 \cup L_2 = \{w \mid (w \in L_1) \vee (w \in L_2)\}$.

La única diferencia entre el autómata intersección y el autómata unión son los estados aceptadores: 

a) En el caso de la intersección, son estados aceptadores los pares en los que cada componente es un estado aceptador.

b) En el caso de la unión, son estados aceptadores los pares en los que alguno de los dos componentes es un estado aceptador.

La razón de esta diferencia es simple: para que una palabra pertenezca a la unión es suficiente con que pertenezca a alguno de los dos lenguajes y, por consiguiente, es suficiente con que uno de los dos autómatas, A_1 o A_2 , la acepte. Pues bien, cuando el autómata unión llega a un estado tal que alguno de los dos componentes es un estado aceptador, significa que alguno de los dos autómatas, A_1 o A_2 , ha llegado a un estado aceptador.

Al igual que en las otras ocasiones, expondremos un caso práctico. Recordemos las tablas de las funciones de transición de los autómatas que reconocen los lenguajes $L_1 = \{w \mid \exists x, y \in \Sigma^* (w = x a a y)\}$ (palabras que contienen el factor aa) y $L_2 = \{w \mid \exists x \in \Sigma^* (w = x b a)\}$ (palabras que acaban con el sufijo ba) que ya hemos utilizado en el ejemplo de la intersección:


δ_1	a	b
$\rightarrow A$	B	A
B	C	A
$\dagger C$	C	C

δ_2	a	b
$\rightarrow 1$	1	2
2	3	2
$\dagger 3$	1	2

Para construir la tabla del autómata unión se procede exactamente de la misma forma que en el caso de la intersección. La única diferencia reside en el hecho de saber qué estados son considerados aceptadores:

δ	a	b
$\rightarrow \langle A, 1 \rangle$	$\langle B, 1 \rangle$	$\langle A, 2 \rangle$
$\langle B, 1 \rangle$	$\langle C, 1 \rangle$	$\langle A, 2 \rangle$
$\langle A, 2 \rangle$	$\langle B, 3 \rangle$	$\langle A, 2 \rangle$
$\dagger \langle C, 1 \rangle$	$\langle C, 1 \rangle$	$\langle C, 2 \rangle$
$\dagger \langle B, 3 \rangle$	$\langle C, 1 \rangle$	$\langle A, 2 \rangle$
$\dagger \langle C, 2 \rangle$	$\langle C, 3 \rangle$	$\langle C, 2 \rangle$
$\dagger \langle C, 3 \rangle$	$\langle C, 1 \rangle$	$\langle C, 2 \rangle$

Al igual que en el caso de la intersección, este método también puede aplicarse a NFA.

Sin embargo, hay un segundo método, que explicamos a continuación, que es especialmente interesante cuando alguno de los autómatas que se deben unir es indeterminista (o todos lo son). 

3.3.2. Unión por adjunción

La adjunción es un método alternativo al producto cartesiano para obtener el autómata unión.

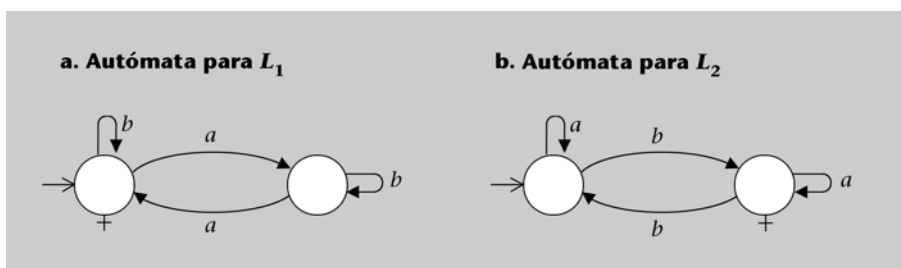
Si el lenguaje L_1 es reconocido por el autómata $A_1 = (Q_1, \Sigma, I_1, F_1, \delta_1)$ y el lenguaje L_2 es reconocido por el autómata $A_2 = (Q_2, \Sigma, I_2, F_2, \delta_2)$, entonces el lenguaje $L_1 \cup L_2$ es reconocido por el NFA $A = (Q_1 \cup Q_2, \Sigma, I_1 \cup I_2, F_1 \cup F_2, \delta)$ con δ definida de la siguiente forma:

$$\delta(p, a) = \begin{cases} \delta_1(p, a) & \text{si } p \in Q_1, \\ \delta_2(p, a) & \text{si } p \in Q_2. \end{cases}$$

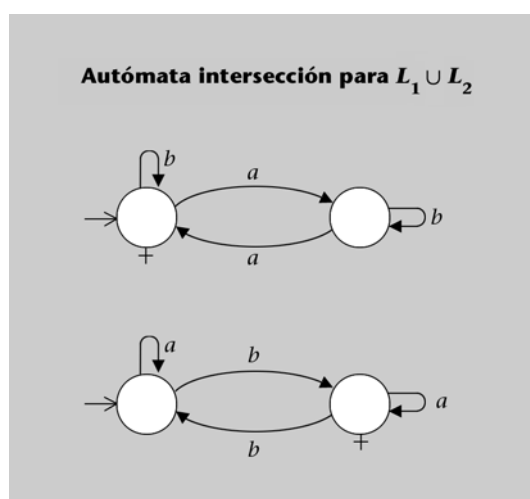
Este modo de obtener un autómata que reconozca la unión de dos lenguajes se denomina **adjunción**.

La adjunción únicamente consiste en considerar que en lugar de dos autómatas se tiene uno. Tanto si los autómatas de partida son deterministas como si no lo son, el resultado es siempre un autómata indeterminista. **!**


Por ejemplo, los siguientes autómatas reconocen los lenguajes $L_1 = \{w \mid |w|_a = 2n, n \geq 0\}$ (palabras con un número par de símbolos a) y $L_2 = \{w \mid |w|_b = 2n + 1, n \geq 0\}$ (palabras con un número impar de símbolos b):



Pues bien, un autómata indeterminista que reconoce $L_1 \cup L_2$ es el siguiente:



Para decidir cuál de los dos métodos para obtener el autómata que reconoce el lenguaje unión es preferible, se debe tener en cuenta que la adjunción introduce indeterminismo. Por consiguiente, y como regla general, podemos decir

que si los autómatas de partida son deterministas, entonces el mejor es el método basado en el producto cartesiano. Si alguno de los autómatas es indeterminista, entonces es preferible usar el método basado en la adjunción. 


3.4. Autómata finito para la concatenación

Si L_1 y L_2 son lenguajes regulares, L_1L_2 también es un lenguaje regular.

A diferencia de los casos precedentes, comenzamos explicando cómo hay que proceder para obtener el autómata que reconoce L_1L_2 , dados los autómatas que reconocen L_1 y L_2 . Posteriormente formalizaremos la construcción utilizada.

La idea que guía la obtención del autómata que reconoce L_1L_2 es muy simple: las palabras de L_1L_2 tienen una primera parte de L_1 y una segunda parte de L_2 . Pues bien, lo que haremos será aprovechar el primer autómata para reconocer la primera parte y, cuando el reconocimiento se produzca, saltaremos al autómata que reconoce L_2 e intentaremos aceptar, utilizando este segundo autómata, una palabra de L_2 . Si finalmente el autómata que reconoce L_2 llega a un estado aceptador, esto significará que se ha aceptado una palabra de L_1 primero y una palabra de L_2 , después; en otros términos, una palabra de L_1L_2 .


Para concatenar los dos autómatas, añadiremos transiciones a todos los estados del primer autómata que desemboquen en estados aceptadores. Las nuevas transiciones, con los mismos símbolos que las originales, irán a parar a los estados iniciales del autómata que reconoce el segundo lenguaje.

En relación con los estados aceptadores y con los estados iniciales, deberemos tener en cuenta lo siguiente: 

a) Los estados aceptadores serán los de L_2 .

b) Los estados iniciales serán:

- Los de L_1 , cuando $\lambda \notin L_1$.
- Los de L_1 y los de L_2 , cuando $\lambda \in L_1$.

Si A_1 y A_2 son los autómatas que reconocen L_1 y L_2 , respectivamente, procederemos de la siguiente manera: 

1) Conectaremos los estados que preceden a los aceptadores de A_1 , con los iniciales de A_2 : si f es un estado aceptador de A_1 , y hay una transición que va, con el símbolo a , de un estado p a un estado f , entonces añadiremos transiciones que vayan con el símbolo a desde p hasta todos los estados iniciales del autómata A_2 .

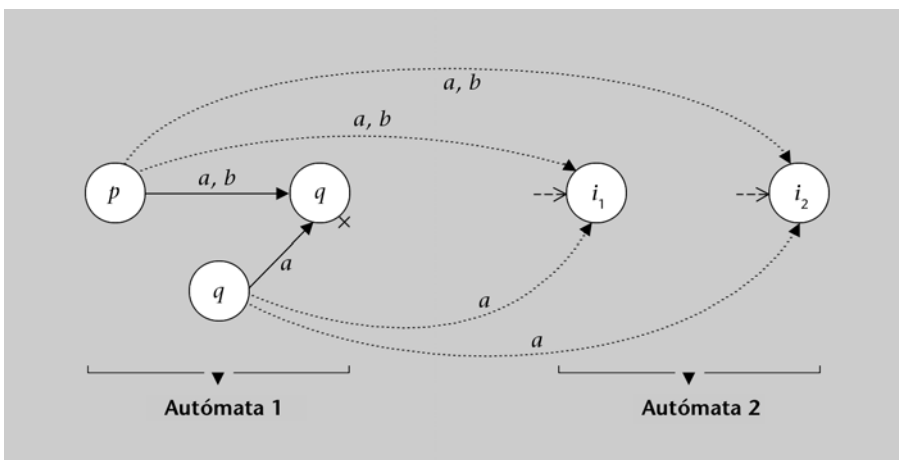
2) En relación con qué estados son los iniciales, tenemos:

a) Si $\lambda \in L_1$ (algún estado inicial de A_1 también es aceptador), entonces dejamos como estados iniciales del nuevo autómata tanto los de A_1 como los de A_2 .

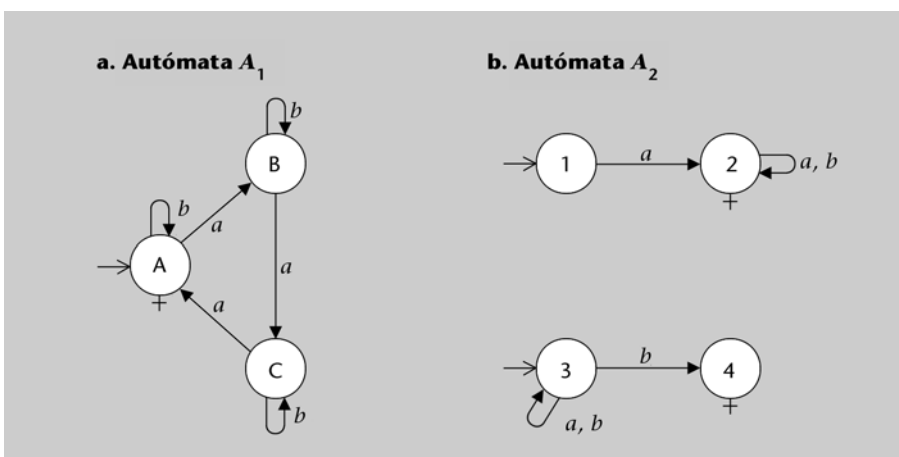
b) Si $\lambda \notin L_1$, entonces no consideramos los estados iniciales de A_2 como iniciales del nuevo autómata (dejamos sólo los iniciales de A_1).

3) Consideremos estados aceptadores sólo los estados aceptadores del autómata A_2 . Los estados aceptadores de A_1 dejan de serlo.

La siguiente figura representa la concatenación de dos autómatas:



Un ejemplo nos permitirá ilustrar todo el proceso. Consideremos A_1 y A_2 como los autómatas que reconocen lenguajes $L_1 = \{w \mid |w|_a = 3n, n \geq 0\}$ (palabras con un número de símbolos a múltiplo de 3) y $L_2 = \{w \mid \exists x \in \Sigma^* (w = ax \vee w = xb)\}$ (palabras que comienzan con el símbolo a o que acaban con el símbolo b).



Comenzaremos determinando cuáles son los estados del primer autómata que preceden a los estados aceptadores, y con qué símbolos:

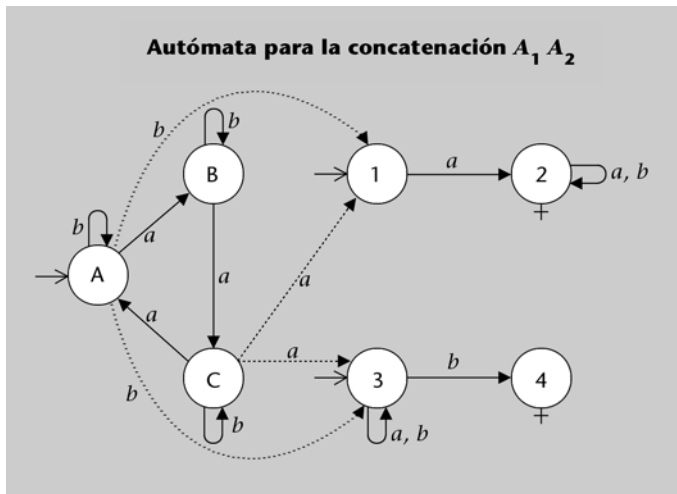
- Estado A: se da una transición desde A hasta A (estado aceptador) con b .
- Estado C: se da una transición desde C hasta A (estado aceptador) con a .

Ahora debemos añadir transiciones que permitan, con los mismos símbolos, llegar a los estados iniciales del segundo autómata (1 y 3):

- Una transición de A a 1, con el símbolo b .
- Una transición de A a 3, con el símbolo b .
- Una transición de C a 1, con el símbolo a .
- Una transición de C a 3, con el símbolo a .

Para acabar, sólo nos queda ver qué estados deben ser iniciales y cuáles aceptadores. En relación con los estados iniciales, debemos preguntarnos si $\lambda \in L_1$. Dado que la respuesta es afirmativa (el estado inicial también es aceptador), entonces tanto los estados iniciales de A_1 como los de A_2 deben ser estados iniciales del nuevo autómata. Los estados finales son sólo los del segundo autómata.

El resultado final se representa en la siguiente figura:



La construcción que acabamos de ver se define formalmente como exponemos a continuación:

Sean $A_1 = (Q_1, \Sigma, I_1, F_1, \delta_1)$ y $A_2 = (Q_2, \Sigma, I_2, F_2, \delta_2)$, que son autómatas que reconocen L_1 y L_2 , respectivamente. Entonces, el lenguaje $L_1 L_2$ es reconocido por el autómata $A = (Q_1 \cup Q_2, \Sigma, I, F_2, \delta)$ con:

- $$I = \begin{cases} I_1; & \text{si } I_1 \cap F_1 = \emptyset (\lambda \notin L_1) \\ I_1 \cup I_2; & \text{si } I_1 \cap F_1 \neq \emptyset (\lambda \in L_1) \end{cases}$$
- $$\delta(q, a) = \begin{cases} \delta_1(q, a) \cup I_2; & \text{si } (q \in Q_1) \text{ y } (F_1 \cap \delta_1(q, a) \neq \emptyset), \\ \delta_1(q, a); & \text{si } (q \in Q_1) \text{ y } (F_1 \cap \delta_1(q, a) = \emptyset), \\ \delta_2(q, a); & \text{si } q \in Q_2. \end{cases}$$

No es preciso que los autómatas involucrados en el proceso de concatenación sean indeterministas. Si alguno de los autómatas fuera indeterminista, sólo sería

necesario considerar que su conjunto de estados iniciales es $\{q_0\}$ y que su función de transición determina conjuntos de un único estado.

El autómata resultante de la concatenación siempre es indeterminista, independientemente de si los autómatas originales lo eran o no. **!**

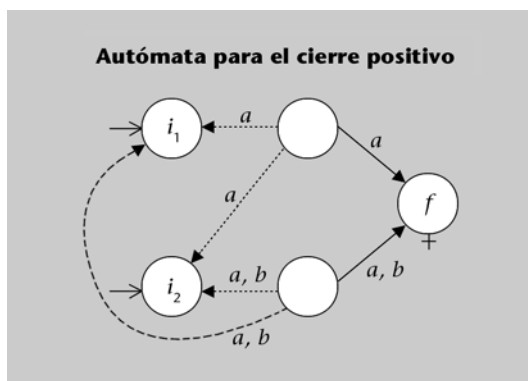
3.5. Autómata finito para el cierre positivo

Si L es un lenguaje regular, su cierre positivo L^+ también es un lenguaje regular.

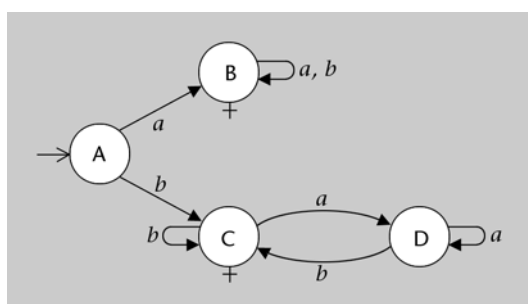
Para el caso del cierre positivo, la idea es esencialmente la misma que para el caso de la concatenación. Dado que sólo se involucra un único autómata, éste se concatena consigo mismo.

Si f es un estado aceptador y hay una transición que va con el símbolo a , de un estado p al estado f , entonces añadimos transiciones que, con el símbolo a , vayan de p a todos los estados iniciales.

Esto se representa gráficamente en la siguiente figura:



Por ejemplo, el siguiente autómata reconoce el lenguaje $L = \{w \mid \exists x \in \Sigma^*(w = ax \vee w = xb)\}$ (palabras que empiezan por a o que acaban en b):



Para construir el autómata que reconoce L^+ , a partir del que reconoce L , se dan los siguientes pasos:

1) Se empieza detectando qué estados preceden a los aceptadores y con qué símbolos:

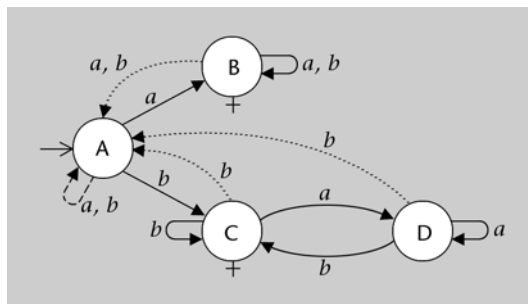
- Estado A: hay una transición desde A hasta B, con el símbolo a .
- Estado A: hay una transición desde A hasta C, con el símbolo b .
- Estado B: hay una transición desde B hasta B (aceptador), con el símbolo a .
- Estado B: hay una transición desde B hasta B (aceptador), con el símbolo b .
- Estado C: hay una transición desde C hasta C (aceptador), con el símbolo b .
- Estado D: hay una transición desde D hasta C, con el símbolo b .

2) Para acabar, se añaden las transiciones que permiten llegar a los estados iniciales del autómata (A) con los mismos símbolos:

- Una transición de A a A, con el símbolo a .
- Una transición de A a A, con el símbolo b .
- Una transición de B a A, con el símbolo a .
- Una transición de B a A, con el símbolo b .
- Una transición de C a A, con el símbolo b .
- Una transición de D a A, con el símbolo b .

No es necesario hacer nada más porque, en todos los casos, los estados iniciales y aceptadores son los mismos.

La siguiente figura muestra el autómata que reconoce L^+ :



Formalmente, la construcción que hemos visto se define como exponemos a continuación:

Sea $A = (Q, \Sigma, I, F, \delta)$ el autómata que reconoce el lenguaje L . Entonces el lenguaje L^+ es reconocido por el autómata $A^+ = (Q, \Sigma, I, F, \delta^+)$ con:

$$\delta^+(q, a) = \begin{cases} \delta(q, a) \cup I; & \text{si } \delta(q, a) \cap F \neq \emptyset, \\ \delta(q, a); & \text{si } \delta(q, a) \cap F = \emptyset. \end{cases}$$

Al igual que en el caso de la concatenación, no es necesario que el autómata sea indeterminista. Sin embargo, el resultado siempre será indeterminista, independientemente del autómata original. !

3.6. Autómata finito para el cierre de Kleene

Si L es un lenguaje regular, su cierre de Kleene, L^* , también lo es.

Recordad que hemos visto el cierre de Kleene en el subapartado 3.4.1. del módulo "Alfabetos, palabras y lenguajes". !

Antes de plantear cómo se construye el autómata que reconoce L^* a partir del autómata que reconoce L , recordemos algunos aspectos esenciales de la relación entre L^* y L^+ :

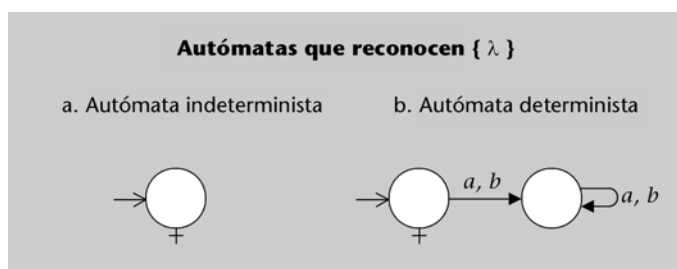
1) $L^* = L^+ \cup \{\lambda\}$.

2) $\lambda \in L \Leftrightarrow L^* = L^+$ (de hecho, lo que nos interesa es $\lambda \in L \Rightarrow L^* = L^+$).

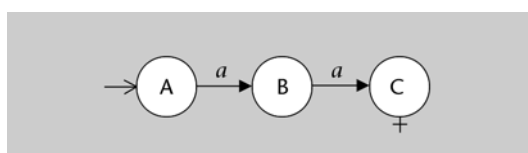
Pues bien, la tarea será simple porque nos podemos encontrar ante las dos situaciones siguientes: !

1) Si $\lambda \in L$, entonces el mismo autómata que reconoce a L^+ también reconoce a L^* , porque $L^* = L^+$. Entonces deberemos proceder de la misma manera que en el caso del cierre positivo.

2) Si $\lambda \notin L$, entonces la diferencia entre L^* y L^+ es λ . Sólo será necesario unir el autómata que reconoce L^+ con el que reconoce $\{\lambda\}$.



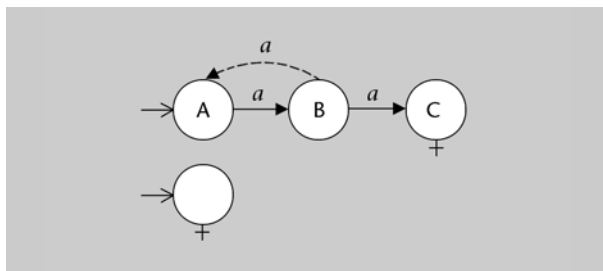
Como ejemplo, considerad el siguiente autómata, que reconoce el lenguaje $L = \{aa\}$:



Puesto que $\lambda \notin L$, primero construimos al autómata que reconoce L^+ procediendo tal como se ha visto en el subapartado anterior:

a) El estado B precede al estado C con el símbolo a ; y ningún otro estado precede al estado C con ningún símbolo. Por tanto, sólo será necesario añadir una transición de B hacia A, con el símbolo a .

b) Finalmente, sólo es necesario unir este autómata con el que reconoce $\{\lambda\}$. Dado que el autómata que reconoce L^+ ya es indeterminista, no nos preocuparemos por el hecho de introducir un poco más de indeterminismo, de modo que optaremos por la adjunción como se representa en la siguiente figura:



Formalmente, esta construcción se define de la siguiente forma:

Si $A = (Q, \Sigma, I, F, \delta)$ es el autómata que reconoce el lenguaje L , entonces el lenguaje L^* es reconocido por el autómata $A^* = (Q \cup \{q_0\}, \Sigma, I \cup \{q_0\}, F \cup \{q_0\}, \delta^*)$ con:

$$\delta^*(q, a) = \begin{cases} \emptyset; & \text{si } q = q_0, \\ \delta(q, a) \cup I; & \text{si } (q \in Q) \text{ y } (F \cap \delta(q, a) \neq \emptyset), \\ \delta(q, a); & \text{si } (q \in Q) \text{ y } (F \cap \delta(q, a) = \emptyset). \end{cases}$$

3.7. Autómata finito para el lenguaje inverso

Si L es un lenguaje regular, su inverso, L^R , también lo es.

En este caso, posiblemente la idea intuitiva que se tendría consistiría en construir un nuevo autómata, intercambiando los papeles de los estados aceptadores y de los estados iniciales (los estados aceptadores pasarían a ser estados iniciales, y los estados iniciales pasarían a ser estados aceptadores) e invirtiendo el sentido de las transiciones.

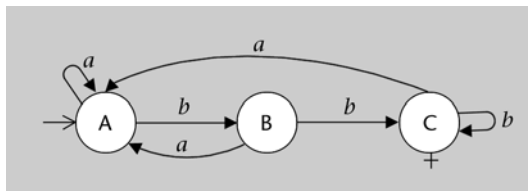
Pues bien, esta idea intuitiva sería correcta.

Recordad el concepto de inversión que se ha visto en el subapartado 3.3. del módulo "Alfabetos, palabras y lenguajes".

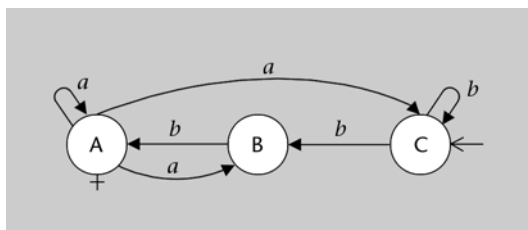
Sea $A = (Q, \Sigma, I, F, \delta)$ el autómata que reconoce al lenguaje L . Entonces el lenguaje L^R es reconocido por el autómata $A^R = (Q, \Sigma, I, F, \delta^R)$ con:

$$\delta^R(q, a) = \{p \in Q \mid q \in \delta(p, a)\}.$$

Por ejemplo, a partir del siguiente autómata:



que reconoce el lenguaje $L = \{w \mid \exists x \in \Sigma^*(w = xbb)\}$ (palabras que acaban con el sufijo bb), se obtiene el siguiente autómata:



intercambiando el estado inicial por el aceptador, el aceptador por el inicial e intercambiando el sentido de las transiciones. Este autómata reconoce el lenguaje $L^R = \{w \mid \exists x \in \Sigma^*(w = bbx)\}$ (palabras que empiezan por el prefijo bb).

Este método puede provocar la introducción de indeterminismo. !


Actividad

3.2. Hallad un autómata indeterminista tal que, si se le aplica el método de inversión, se convierte en un autómata determinista.


4. Minimización de autómatas finitos

En este apartado nos ocuparemos del problema de encontrar versiones mínimas de los DFA.

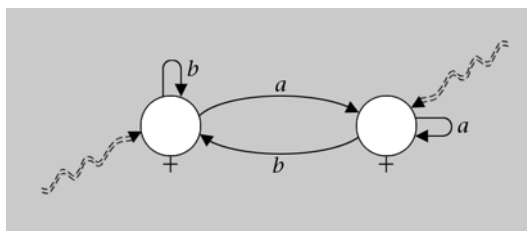
La versión **mínima de un DFA** es otro DFA que reconoce exactamente el mismo lenguaje que el primero, pero que lo hace con el menor número posible de estados.

Minimizar el número de estados de un DFA es muchas veces una necesidad incuestionable. Sólo es necesario recordar los dos hechos siguientes: 

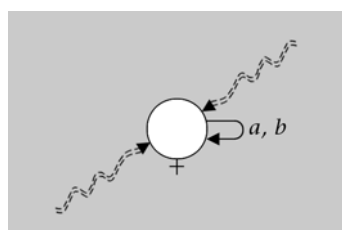
- 1) La determinización puede provocar que se pase de N a 2^N estados.
- 2) La introducción de indeterminismo es insalvable cuando se utilizan algunos de los métodos que hemos visto en el apartado anterior.

Una propiedad muy interesante de los autómatas mínimos es que son únicos. Si bien hay una infinidad de autómatas diferentes que reconocen un mismo lenguaje regular, sólo hay uno que tenga el menor número posible de estados. 

La idea subyacente en el algoritmo de minimización que veremos es que, en un DFA, puede haber grupos de DFA que se comporten de la misma manera. Un grupo de estados que tengan el mismo comportamiento se puede reducir a un único estado. Por ejemplo, si en un DFA aparece una parte como la siguiente:

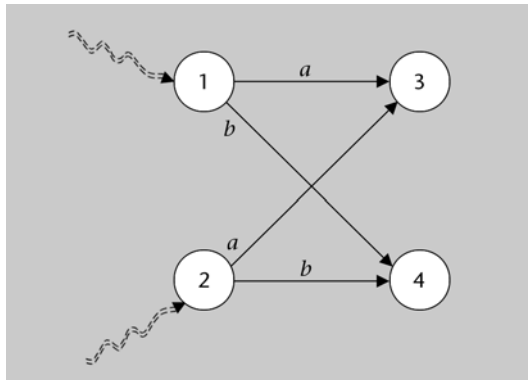


es indiferente la manera de llegar a estos estados y a cuál de los dos se llegue. Una vez se ha llegado, nunca se abandona una situación de estado aceptador y, por consiguiente, sería suficiente con:

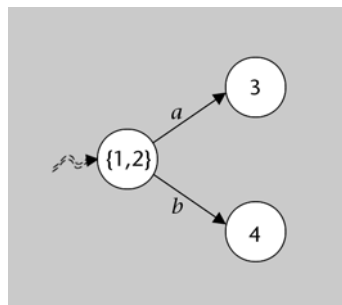


Una situación parecida se puede dar cuando en un DFA hay más de un estado con la función de pozo, porque todos pueden reducirse a uno solo.

Otra situación de estados que se comportan de la misma manera podría ser la siguiente:



Los estados 1 y 2 se comportan igual (con el símbolo a , transición al estado 3; con el símbolo b , transición al estado 4). Por esta razón pueden ser sustituidos por un único estado, como podéis observar en la siguiente figura:



4.1. Equivalencia de estados

Sean $A = (Q, \Sigma, q_0, F, \delta)$ y $p, q \in Q$ un par de estados de este DFA.

Se dice que p y q son **estados equivalentes** si, y sólo si, una palabra desde p hace evolucionar el autómata hasta un estado aceptador; entonces desde q también lo hace evolucionar hasta un estado aceptador.

Otra manera de decir lo mismo es que p y q son estados equivalentes si, y sólo si, el conjunto de las palabras que son aceptadas desde p es idéntico al de las palabras que son aceptadas desde q .

Cuando p y q son equivalentes, esto se denota $p \equiv q$. Formalmente:


$$p \equiv q \text{ ssi } \forall w \in \Sigma^* [\delta(p, w) \in F \leftrightarrow \delta(q, w) \in F].$$

Todos los estados pozo...

... tienen el mismo comportamiento: no aceptan indefinidamente.

A menudo los estados equivalentes también se denominan *estados indistinguibles*.

Fijaos en que, cuando dos estados son equivalentes, es lo mismo reconocer a partir de uno que reconocer a partir del otro: si una palabra es aceptada desde un estado, también será aceptada desde el otro, aunque quizá en estados aceptadores diferentes. De la misma forma, si desde uno de éstos se rechaza una palabra, entonces también será rechazada desde el otro.

\equiv es una relación de equivalencia (es reflexiva, simétrica y transitiva) e induce en Q una partición en clases. Cada grupo de estados equivalentes forma una **clase**, y cada clase puede ser sustituida por un único estado. 

Dos estados,...

... uno aceptador y otro no aceptador, nunca pueden ser equivalentes, porque como mínimo para la palabra λ tienen un comportamiento diferente: el primero acepta y el segundo rechaza.

4.2. El autómata cociente

Sea $A = (Q, \Sigma, q_0, F, \delta)$ un DFA. El **autómata cociente de A respecto de \equiv** se denomina \bar{A} y se define como:

$$\bar{A} = (\bar{Q}, \Sigma, \bar{q}_0, \bar{F}, \bar{\delta}),$$

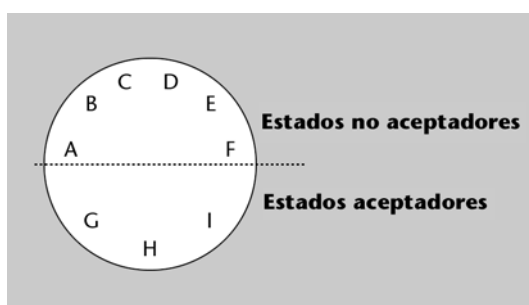
donde:

- \bar{Q} es el cociente de Q para \equiv (el conjunto de las clases inducidas por \equiv en Q).
- \bar{q}_0 es la clase a la que pertenece q_0 (la clase del estado inicial).
- \bar{F} es el conjunto de las clases a las que pertenecen los elementos de F .
- $\bar{\delta} : \bar{Q} \times \Sigma \rightarrow \bar{Q}$ definida como: $\bar{\delta}(\bar{q}, a) = \overline{\delta(q, a)}$ (las clases se comportan como se comportan sus representantes).

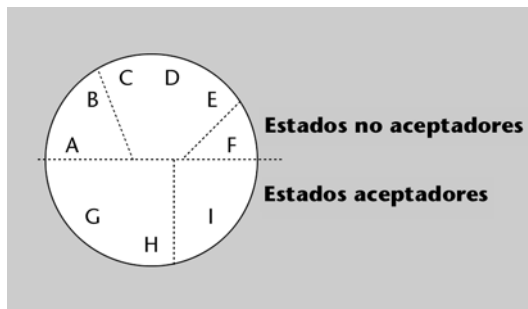
Nota

\bar{q} es la clase que tiene q por representante. El **representante de una clase** es un elemento cualquiera de ésta que lo representa. Dado que todos los elementos de una misma clase son equivalentes, cualquier elemento de ésta puede hacer el papel de representante.

Por ejemplo, supongamos que tenemos un autómata finito determinista con un conjunto de estados $Q = \{A, B, C, D, E, F, G, H, I\}$ donde $q_0 = A$ y $F = \{G, H, I\}$, tal como lo representamos a continuación:



Supongamos, también, que \equiv induce una partición como la siguiente:



Entonces:

- $\bar{Q} = \{\{A, B\}, \{C, D, E\}, \{F\}, \{G, H\}, \{I\}\} = \{\bar{A}, \bar{C}, \bar{F}, \bar{G}, \bar{I}\}$.
- $\bar{q}_0 \{A, B\} = \bar{A}$.
- $\bar{F} \{G, H\}, \{I\} = \{\bar{G}, \bar{I}\}$.
- $\bar{\delta}(\bar{A}, a) = \overline{\delta(A, a)}$, $\bar{\delta}(\bar{A}, b) = \overline{\delta(A, b)}$, $\bar{\delta}(\bar{C}, a) = \overline{\delta(C, a)}$, ... (por ejemplo, si $\delta(A, a) = E$, entonces $\bar{\delta}(\bar{A}, a) = \bar{C}$ porque $E \in \bar{C} = \{C, D, E\}$). Fijaos en que $\bar{C} = \bar{D}$ porque representan la misma clase).

Para evitar confusiones...

... escogeremos siempre como representante de una clase su primer elemento (en el orden en que los hayamos escrito, intentando siempre escribirlos en orden alfabético).

Si A es un autómata finito determinista (DFA) que reconoce L y \bar{A} es el autómata cociente de A , entonces \bar{A} es el autómata con el menor número de estados que reconoce L .

4.3. Cómo debe construirse el autómata cociente

La única dificultad en la construcción de \bar{A} radica en la construcción de \bar{Q} . Prestad atención al hecho de que para construir \bar{Q} es necesario determinar qué estados son equivalentes entre sí. Sin embargo, lamentablemente, la definición de \equiv ($p \equiv q$ ssi $\forall w \in \Sigma^* [\bar{\delta}(p, w) \in F \leftrightarrow \bar{\delta}(q, w) \in F]$) involucra un número infinito de comprobaciones ($\forall w \in \Sigma^*$).


Un método...

... que involucrara un número infinito de comprobaciones no sería un método algorítmico, porque cualquier algoritmo debe involucrar únicamente un número finito de pasos.

A continuación veremos un método que permite construir Q en un número finito de pasos. Este método está basado en una restricción de \equiv llamada **k-equivalencia** (\equiv_k).

Dados p y $q \in Q$ se denota:

- $p \equiv_0 q$ ssi $p \in F \leftrightarrow q \in F$.
- $p \equiv_k q$ ssi $\begin{cases} p \equiv_{k-1} q, \\ \forall a \in \Sigma [\bar{\delta}(p, a) \equiv_{k-1} \bar{\delta}(q, a)] \end{cases}$

Así, tenemos que: 

1) Dos estados son **estados cero-equivalentes** (\equiv_0) si, y sólo si, los dos son aceptadores o los dos son no aceptadores.

2) Dos estados son **estados k -equivalentes** (\equiv_k , $k > 0$) si, y sólo si, los dos son $(k - 1)$ -equivalentes y, para cada símbolo del alfabeto, sus transiciones también son $(k - 1)$ -equivalentes.

La k -equivalencia

La k -equivalencia de estados no es más que la equivalencia que hemos definido antes (\equiv), pero restringida a palabras de longitud no superior a k . Así, la definición formal de la equivalencia es:


$$p \equiv q \text{ ssi } \forall w \in \Sigma^* [\tilde{\delta}(p, w) \in F \leftrightarrow \tilde{\delta}(q, w) \in F],$$

otra definición de la k -equivalencia que evidencia la limitación en la longitud de las palabras es:

$$p \equiv_k q \text{ ssi } \forall w \in \Sigma^* |w| \leq k [\tilde{\delta}(p, w) \in F \leftrightarrow \tilde{\delta}(q, w) \in F].$$

Dado que se trata de una relación de equivalencia, \equiv_k también induce en \bar{Q} una partición.

Debéis tener en cuenta que a partir de un determinado número k , el conjunto de clases que induce \equiv_k es el mismo que induce \equiv . El número k está limitado superiormente por $|Q| - 1$ ($k \leq |Q| - 1$).

En la práctica, el hecho de que a partir de un cierto número no superior a $|Q| - 1$ el conjunto de clases inducido por \equiv_k sea el mismo que el inducido por \equiv significa que \equiv_k se puede utilizar para calcular \bar{Q} en un número finito de pasos. 

La misma definición recursiva de \equiv_k permite elaborar el algoritmo que se utiliza para obtener \bar{Q} , que es el siguiente:

algoritmo cálculo de \bar{Q} ;

/*entrada: un DFA $(Q, \Sigma, q_0, F, \delta)^*$ /*

/*salida \bar{Q} (conjunto de las clases de equivalencia)/*

calcular las clases inducidas a Q por \equiv_0

$k := 0$

repetir

$k := k + 1$

calcular las clases inducidas por \equiv_k a partir de las inducidas

por \equiv_{k-1}

hasta que el conjunto de clases inducidas por \equiv_k sea igual

al de las clases inducidas por \equiv_{k-1}

retorna (el conjunto de clases inducidas por \equiv_k)

falgoritmo

Fijaos...

... en que el límite para k ($k \leq |Q| - 1$) depende del número de estados del autómata que se considere.

Según su definición...

... las clases inducidas por \equiv_0 son dos: la que contiene todos los estados aceptadores y la que contiene todos los no aceptadores.

Para acabar de entender el funcionamiento del algoritmo, es necesario ver cómo se calcula un conjunto de particiones a partir de lo anterior:

Cálculo de las clases inducidas por \equiv_k a partir de las inducidas por \equiv_{k-1}

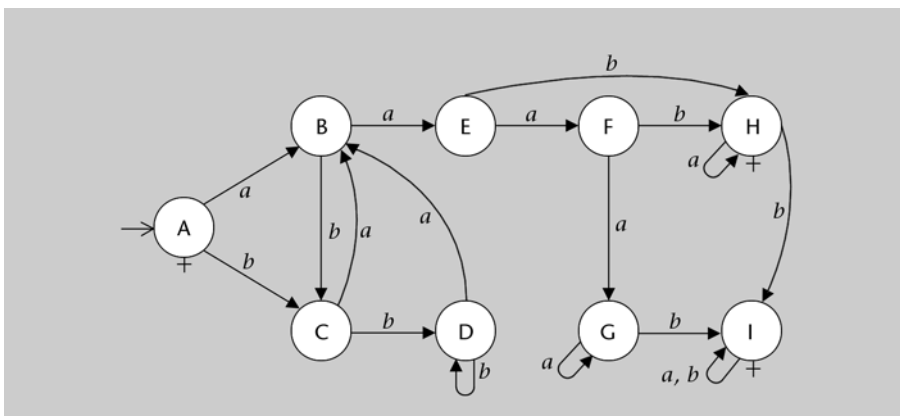
```

para cada clase  $C$  hacer
  para cada estado  $p$  dentro de la clase  $C$  hacer
    para cada símbolo  $a \in \Sigma$  hacer
      calcular la clase a la que pertenece  $\delta(p, a)$ 
    fpara
  fpara
  si todos los estados de la clase tienen sus transiciones equivalentes
    entonces
      la clase  $C$  permanecerá inalterada
    sino
      la clase  $C$  se dividirá en tantas clases como grupos de estados
      con transiciones equivalentes /*pero no ahora, al final*/
  fsi
fpara
Las clases que se deben dividir se dividen en este momento
  
```

Como en otras ocasiones, algunos ejemplos nos permitirán acabar de fijar las ideas.

Ejemplos de minimización de un autómata

Ejemplo 1. Supongamos el siguiente autómata:



Ahora buscamos las clases inducidas por $\equiv_0, \equiv_1, \equiv_2, \dots$

a) Clases inducidas por \equiv_0 . El conjunto de clases inducidas por \equiv_0 es el resultado de separar los estados aceptadores de los no aceptadores: $\bar{A} = \{A, H, I\}$, $\bar{B} = \{B, C, D, E, F, G\}$.

b) Clases inducidas por \equiv_1 . Para determinar las clases inducidas por \equiv_1 se calculan las clases de cada transición:

- Para la clase \bar{A} :
 - Para el estado A: $\delta(A, a) = B \in \bar{B}$ y $\delta(A, b) = C \in \bar{B}$.
 - Para el estado H: $\delta(H, a) = H \in \bar{A}$ y $\delta(H, b) = I \in \bar{A}$.
 - Para el estado I: $\delta(I, a) = I \in \bar{A}$ y $\delta(I, b) = I \in \bar{A}$.

Las transiciones para los estados H e I son equivalentes entre sí, pero no lo son respecto a las del estado A. Por consiguiente, los estados H e I formarán parte de una clase diferente respecto de la de A.

- Para la clase \bar{B} :
 - Para el estado B: $\delta(B, a) = E \in \bar{B}$ y $\delta(B, b) = C \in \bar{B}$.
 - Para el estado C: $\delta(C, a) = B \in \bar{B}$ y $\delta(C, b) = D \in \bar{B}$.
 - Para el estado D: $\delta(D, a) = B \in \bar{B}$ y $\delta(D, b) = D \in \bar{B}$.
 - Para el estado E: $\delta(E, a) = F \in \bar{B}$ y $\delta(E, b) = H \in \bar{A}$.
 - Para el estado F: $\delta(F, a) = G \in \bar{B}$ y $\delta(F, b) = H \in \bar{A}$.
 - Para el estado G: $\delta(G, a) = G \in \bar{B}$ y $\delta(G, b) = I \in \bar{A}$.

Las transiciones para los estados B, C y D son equivalentes entre sí. Las de los estados E, F y G también lo son. Así, la clase \bar{B} se dividirá en dos más.

Por tanto, las clases inducidas por \equiv_1 son las siguientes: $\bar{A} = \{A\}$, $\bar{H} = \{H, I\}$, $\bar{B} = \{B, C, D\}$, $\bar{E} = \{E, F, G\}$.

c) Repetimos el mismo proceso para determinar las clases inducidas por \equiv_2 :

- Para la clase \bar{A} : no es necesario hacer nada, ya que esta clase no se dividirá.
- Para la clase \bar{H} :
 - $\delta(H, a) \in \bar{H}$ y $\delta(H, b) \in \bar{H}$.
 - $\delta(I, a) \in \bar{H}$ y $\delta(I, b) \in \bar{H}$.

En esta iteración, la clase no se dividirá.

- Para la clase \bar{B} :
 - $\delta(B, a) \in \bar{E}$ y $\delta(B, b) \in \bar{B}$.
 - $\delta(C, a) \in \bar{B}$ y $\delta(C, b) \in \bar{B}$.
 - $\delta(D, a) \in \bar{B}$ y $\delta(D, b) \in \bar{B}$.

En esta iteración, esta clase se dividirá en dos, una con el estado B y otra con los estados C y D.

Nota

A partir de este punto, el ejemplo se desarrolla de una forma un poco más detallada.

- Para la clase \bar{E} :
 - $\delta(E, a) \in \bar{E}$ y $\delta(E, b) \in \bar{H}$.
 - $\delta(F, a) \in \bar{E}$ y $\delta(F, b) \in \bar{H}$.
 - $\delta(G, a) \in \bar{E}$ y $\delta(G, b) \in \bar{H}$.

En esta iteración, esta clase no se dividirá.

Así, las clases inducidas por \equiv_2 son las siguientes: $\bar{A} = \{A\}$, $\bar{H} = \{H, I\}$, $\bar{B} = \{B\}$, $\bar{C} = \{C, D\}$, $\bar{E} = \{E, F, G\}$.

d) El paso siguiente será calcular las clases inducidas por \equiv_3 :

- Para la clase \bar{A} : no es necesario hacer nada, ya que esta clase no se dividirá.
- Para la clase \bar{H} :
 - $\delta(H, a) \in \bar{H}$ y $\delta(H, b) \in \bar{H}$.
 - $\delta(I, a) \in \bar{H}$ y $\delta(I, b) \in \bar{H}$.

En esta iteración, la clase no se dividirá.

- Para la clase \bar{B} : no es necesario hacer nada, ya que esta clase no se dividirá.
- Para la clase \bar{C} :
 - $\delta(C, a) \in \bar{B}$ y $\delta(C, b) \in \bar{C}$.
 - $\delta(D, a) \in \bar{B}$ y $\delta(D, b) \in \bar{C}$.

En esta iteración, esta clase no se dividirá.

Para la clase \bar{E} :

- $\delta(E, a) \in \bar{E}$ y $\delta(E, b) \in \bar{H}$.
- $\delta(F, a) \in \bar{E}$ y $\delta(F, b) \in \bar{H}$.
- $\delta(G, a) \in \bar{E}$ y $\delta(G, b) \in \bar{H}$.

En esta iteración, esta clase no se dividirá.

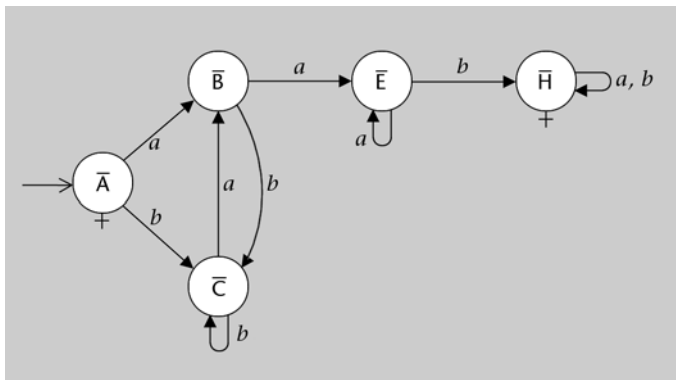
Por tanto, las clases inducidas por \equiv_3 son $\bar{A} = \{A\}$, $\bar{H} = \{H, I\}$, $\bar{B} = \{B\}$, $\bar{C} = \{C, D\}$, $\bar{E} = \{E, F, G\}$.

No hay ninguna diferencia entre las clases inducidas por \equiv_2 y por \equiv_3 , que en ambos casos son: $\bar{A} = \{A\}$, $\bar{H} = \{H, I\}$, $\bar{B} = \{B\}$, $\bar{C} = \{C, D\}$, $\bar{E} = \{E, F, G\}$. Esto indica que el proceso ya se ha acabado y que las clases inducidas por \equiv_2 son las mismas que las inducidas por \equiv_3 .

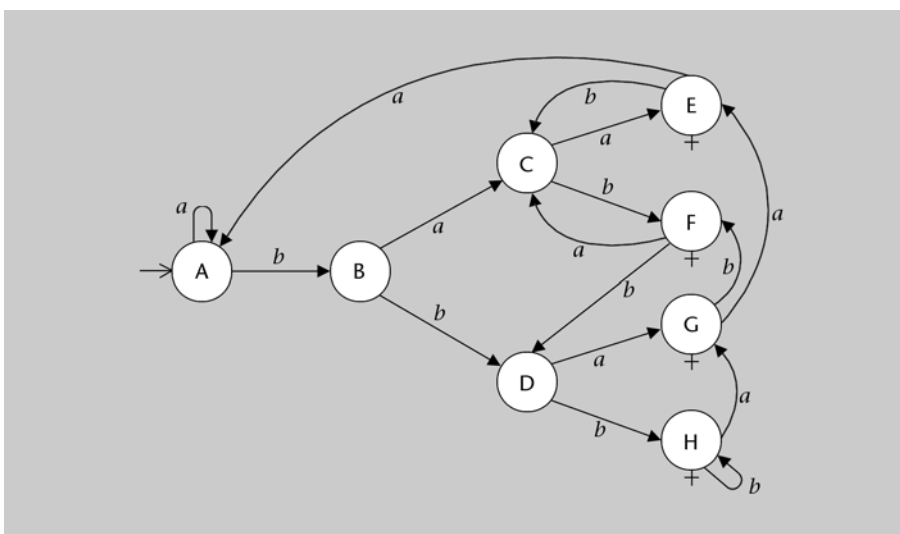
Así pues, el autómata resultante es $\bar{A} = (\bar{Q}, \Sigma, \bar{q}_0, \bar{F}, \bar{\delta})$ con: $\bar{Q} = \{\bar{A}, \bar{B}, \bar{C}, \bar{E}, \bar{H}\}$, $\bar{q}_0 = \bar{A}$, $\bar{F} = \{\bar{A}, \bar{H}\}$, y la tabla de transición será la siguiente:

$\bar{\delta}$	a	b
$\rightarrow \bar{A}$	\bar{B}	\bar{C}
\bar{B}	\bar{E}	\bar{C}
\bar{C}	\bar{B}	\bar{C}
$\dagger \bar{E}$	\bar{E}	\bar{H}
$\dagger \bar{H}$	\bar{H}	\bar{H}

Así pues, la representación gráfica de este autómata es:



Ejemplo 2. Supongamos el siguiente autómata:



Como en el ejemplo anterior, calculamos las clases inducidas por \equiv_0 , \equiv_1 y \equiv_2 :

a) Las clases inducidas por \equiv_0 son $\bar{A} = \{A, B, C, D\}$, $\bar{E} = \{E, F, G, H\}$.

b) Ahora calculamos las clases inducidas por \equiv_1 :

- Para la clase \bar{A} :

- $\delta(A, a) \in \bar{A}, \delta(A, b) \in \bar{A}.$
- $\delta(B, a) \in \bar{A}, \delta(B, b) \in \bar{A}.$
- $\delta(C, a) \in \bar{E}, \delta(C, b) \in \bar{E}.$
- $\delta(D, a) \in \bar{E}, \delta(D, b) \in \bar{E}.$

Esta clase se dividirá en dos.

- Para la clase \bar{E} :

- $\delta(E, a) \in \bar{A}, \delta(E, b) \in \bar{A}.$
- $\delta(F, a) \in \bar{A}, \delta(F, b) \in \bar{A}.$
- $\delta(G, a) \in \bar{E}, \delta(G, b) \in \bar{E}.$
- $\delta(H, a) \in \bar{E}, \delta(H, b) \in \bar{E}.$

Esta clase se dividirá en dos.

Así, las clases inducidas por \equiv_1 son $\bar{A} = \{A, B\}$, $\bar{C} = \{C, D\}$, $\bar{E} = \{E, F\}$, $\bar{G} = \{G, H\}$.

c) Calculamos las clases inducidas por \equiv_2 :

- Para la clase \bar{A} :

- $\delta(A, a) \in \bar{A}, \delta(A, b) \in \bar{A}.$
- $\delta(B, a) \in \bar{C}, \delta(B, b) \in \bar{C}.$

Esta clase se dividirá en dos.

- Para la clase \bar{C} :

- $\delta(C, a) \in \bar{E}, \delta(C, b) \in \bar{E}.$
- $\delta(D, a) \in \bar{G}, \delta(D, b) \in \bar{G}.$

Esta clase se dividirá en dos.

- Para la clase \bar{E} :

- $\delta(E, a) \in \bar{A}, \delta(E, b) \in \bar{A}.$
- $\delta(F, a) \in \bar{A}, \delta(F, b) \in \bar{A}.$


Esta clase se dividirá en dos.

- Para la clase \bar{G} :

- $\delta(G, a) \in \bar{E}, \delta(G, b) \in \bar{E}.$
- $\delta(H, a) \in \bar{G}, \delta(H, b) \in \bar{G}.$

Esta clase se dividirá en dos.


Por tanto, las clases inducidas por \equiv_2 son las siguientes: $\bar{A} = \{A\}$, $\bar{B} = \{B\}$, $\bar{C} = \{C\}$, $\bar{D} = \{D\}$, $\bar{E} = \{E\}$, $\bar{F} = \{F\}$, $\bar{G} = \{G\}$, $\bar{H} = \{H\}$.

Se ha llegado a una situación en la que hay tantas clases como estados. Esto significa que no existen estados equivalentes entre sí y que, por tanto, el autómata no se puede minimizar. El autómata original ya era mínimo. 

4.4. Unicidad del autómata mínimo

Muchos autómatas diferentes pueden reconocer un mismo lenguaje regular, a pesar de que todos tendrán en común un mismo autómata mínimo.

Dado un lenguaje regular, el autómata mínimo que lo reconoce es único.

Esta propiedad del autómata mínimo tiene las dos aplicaciones inmediatas siguientes: 

1) Para saber si dos autómatas A_1 y A_2 reconocen el mismo lenguaje, los minimizamos y comparamos los autómatas mínimos resultantes. Si son iguales, ya podemos afirmar que A_1 y A_2 reconocen el mismo lenguaje.

2) Para saber si dos lenguajes regulares, L_1 y L_2 , son el mismo ($L_1 = L_2$), construimos los autómatas que los reconocen, los minimizamos y comparamos los autómatas mínimos resultantes. Si son iguales, ya podemos afirmar que L_1 y L_2 también lo son.

5. Expresiones regulares

5.1. Definición de *expresión regular*


Las expresiones regulares proporcionan una manera simple, y al mismo tiempo precisa, de describir lenguajes regulares.

Si Σ es un alfabeto, una **expresión regular sobre Σ** se define, recursivamente, de la siguiente manera:

- 1) \emptyset y λ son expresiones regulares.
- 2) Cada símbolo de Σ es una expresión regular.
- 3) Si e_1 y e_2 son expresiones regulares, entonces:
 - $(e_1 + e_2)$ es una expresión regular.
 - $(e_1 \cdot e_2)$ es una expresión regular.
- 4) Si e es una expresión regular, entonces e^* es una expresión regular.
- 5) Ninguna otra es una expresión regular.

Para simplificar la escritura adoptaremos las siguientes convenciones: 

- 1) La prioridad de $*$ es la mayor y la de $+$ es la menor.
- 2) El punto (\cdot) se puede omitir.
- 3) Los paréntesis que no sean estrictamente necesarios para la correcta lectura se pueden omitir.

Cada expresión regular describe un lenguaje. Si denominamos $L(e)$ al lenguaje descrito por la expresión regular e , entonces: 


- 1) $L(\emptyset) = \emptyset$.
- 2) $L(\lambda) = \{\lambda\}$.
- 3) $L(a) = \{a\}$ ($\forall a \in \Sigma$).
- 4) $L(e_1 + e_2) = L(e_1) + L(e_2)$.

Fijaos en que...

... al definir las expresiones regulares no se menciona ni la complementación ni la intersección.

$$5) L(e_1 \cdot e_2) = L(e_1) \cdot L(e_2).$$


$$6) L(e^*) = [L(e)]^*.$$

Dado que utilizaremos las expresiones regulares con el propósito de describir lenguajes, no recurriremos a la notación $L(e)$ para referirnos al lenguaje descrito por e . Sencillamente, utilizaremos e para referirnos al lenguaje que e describe. 

Algunos ejemplos de expresiones regulares son los que podemos encontrar a continuación:

- El lenguaje descrito por $a + b$ es $\{a\} \cup \{b\} = \{a, b\} = \Sigma$.
- El lenguaje descrito por $(a + b)^*$ es $\{a, b\}^* = \Sigma^*$.
- El lenguaje descrito por $a(a + b)^*$ es $\{a\}\Sigma^*$.
- El lenguaje descrito por $a(a + b)^* + b(a + b)^*$ es $\{a\}\Sigma^* \cup \{b\}\Sigma^* = \Sigma^+$.

Si tenemos en cuenta que la concatenación es distributiva respecto de la unión, —es decir, $(L_1 \cup L_2)L_3 = L_1L_3 \cup L_2L_3$; $L_1(L_2 \cup L_3) = L_1L_2 \cup L_1L_3$ —, veremos que, utilizando el símbolo $+$ para designar la unión, las expresiones regulares se pueden manipular de la forma algebraica habitual.

 La distributividad de la concatenación respecto de la unión se ha estudiado en el subapartado 2.1.2. del módulo “Alfabetos, palabras y lenguajes”.

El símbolo $+$

Aunque estrictamente hablando el símbolo $+$, para designar el cierre positivo de Kleene, no forma parte del conjunto de símbolos que se pueden usar en las expresiones regulares, escribiremos e^+ en lugar de ee^* y de e^*e siempre que lo consideremos conveniente. De la misma manera, escribiremos e^2 en lugar de ee , e^3 en lugar de eee , etc.

Por ejemplo:

$$\begin{aligned} b((a + b)^*a + (a + b)^*b)a &= (b(a + b)^*a + b(a + b)^*b)a = \\ &= b(a + b)^*aa + b(a + b)^*ba = b(a + b)^*(aa + ba) = \\ &= b(a + b)^*(a + b)a = b(a + b)^+a. \end{aligned}$$

5.2. Relación entre los lenguajes regulares y las expresiones regulares


5.2.1. El teorema de Kleene

Las expresiones regulares y los lenguajes regulares están muy vinculados, tal como pone de manifiesto el teorema de Kleene: un lenguaje es regular si, y sólo si, es descrito por una expresión regular.

Dado que un lenguaje es regular si, y sólo si, es reconocido por algún autómata finito, la relación entre lenguajes regulares y expresiones regulares también puede expresarse como exponemos a continuación:

Si L es un lenguaje, entonces existe un autómata que reconoce L si, y sólo si, existe una expresión regular que describe L :

$$\forall L [\exists A = (Q, \Sigma, q_0, F, \delta) L(A) = L \Leftrightarrow \exists e, \text{ expresión regular, } L(e) = L].$$

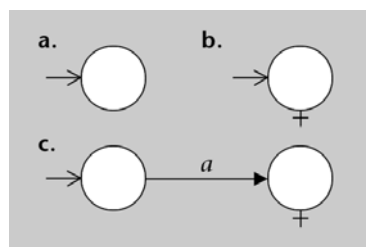
La demostración del teorema de Kleene entre autómatas finitos y expresiones regulares es muy interesante, porque nos mostrará los siguientes procedimientos: 

- 1) Construir el autómata que reconoce el lenguaje descrito por una expresión regular dada (sentido \Leftarrow de la implicación).
- 2) Hallar la expresión regular que describe el lenguaje reconocido por un autómata (sentido \Rightarrow de la implicación).

5.2.2. Construcción del autómata finito que reconoce el lenguaje asociado a una expresión regular

Si tenemos en cuenta que:

- 1) El lenguaje \emptyset es reconocido por el autómata **a** de la figura.
- 2) El lenguaje $\{\lambda\}$ es reconocido por el autómata **b** de la figura.
- 3) El lenguaje $\{a\}$ es reconocido por el autómata **c** de la figura.

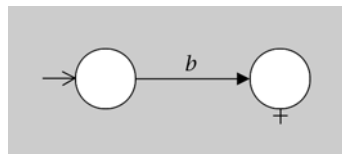


Y también tenemos en cuenta los procedimientos descritos en este mismo módulo para construir los autómatas de la unión, la concatenación y el cierre de Kleene de lenguajes regulares; podemos ver el proceso que debemos seguir para construir el autómata que reconoce el lenguaje descrito por una expresión regular.

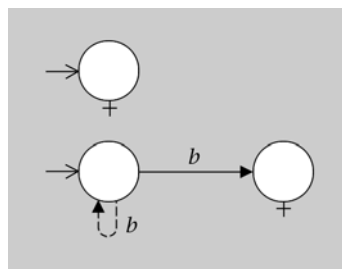
A partir de los autómatas que reconocen \emptyset , $\{\lambda\}$, $\{a\}$ y $\{b\}$, se construye el autómata del lenguaje descrito por cualquier expresión regular, utilizando los procedimientos de unión, concatenación y cierre de Kleene de autómatas.

Por ejemplo, para encontrar el autómata que reconoce el lenguaje descrito por la expresión regular $b^*(a + b)a$, se procedería de la forma siguiente:

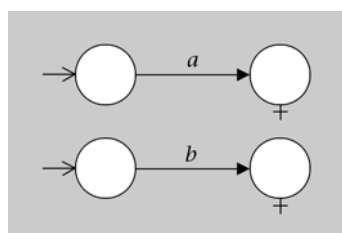
1) El autómata que reconoce b es el siguiente:



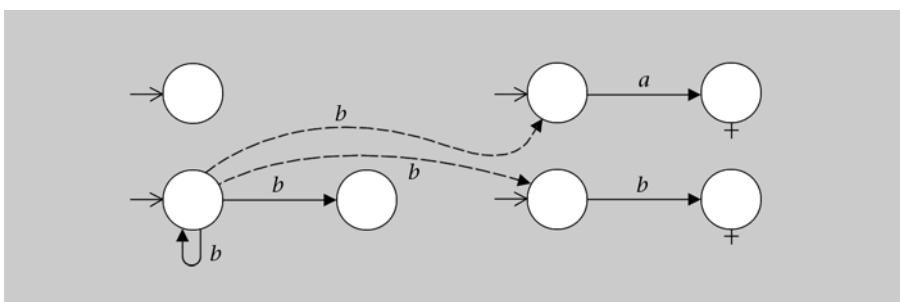
2) A partir de éste, se encuentra el que reconoce b^* :



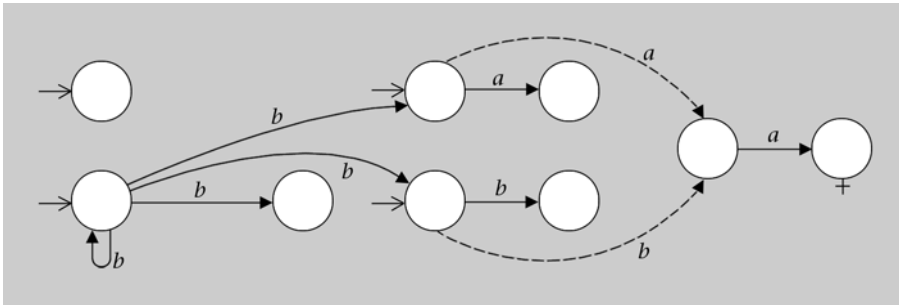
3) El autómata que reconoce $a + b$ (a partir de los autómatas que reconocen a y b , respectivamente) es el de la figura siguiente:



4) A partir de los dos anteriores se encuentra el que reconoce $b^*(a + b)$:



5) A partir del que reconoce a y del que reconoce $b^*(a + b)$ se encontraría el que reconoce $b^*(a + b)a$:



6) El último paso, que aquí omitimos, sería la determinización y minimización del resultado obtenido.

Esta manera de proceder para construir el autómata que reconoce el lenguaje descrito por una expresión regular es muy tediosa si se debe desarrollar manualmente, mientras que es muy fácil de programar para que la desarrolle un ordenador.

5.2.3. Determinación del lenguaje reconocido por un autómata finito: el lema de Arden

En este apartado veremos cómo se puede encontrar una expresión regular que describa el lenguaje reconocido por un autómata finito. Antes, y para preparar el terreno, estudiaremos el lema de Arden, que nos permitirá resolver ecuaciones lineales de lenguajes.

El **lema de Arden** afirma que la ecuación $X = AX + B$, donde A y B son dos lenguajes cualesquiera:

- 1) Da como resultado el lenguaje A^*B .
- 2) El lenguaje A^*B es la menor solución. Cualquier otra solución contendrá A^*B .
- 3) Si $\lambda \notin A$, entonces A^*B es la única solución de la ecuación.


Fijaos en que...

... aunque lo aplicaremos a los lenguajes regulares, el lema de Arden se aplica a cualesquiera lenguajes no necesariamente regulares.

Para demostrar que A^*B es una solución de la ecuación $X = AX + B$, es suficiente con sustituir X por A^*B en la ecuación y verificar su igualdad. Entonces, haciendo la sustitución tenemos $A^*B = AA^*B + B$. A partir de aquí operaremos siempre el miembro derecho de la igualdad:

- $A^*B = A^*B + B$, ya que $AA^* = A^*$.
- $A^*B = (A^+ + \lambda)B$, sacando el factor común B .
- $A^*B = A^*B$, porque $A^+ + \lambda = A^*$.

Así, queda probado que A^*B es una solución de la ecuación.

Las otras dos propiedades de la solución, que se trata de la menor solución y que si $\lambda \notin A$ se trata de la única solución, no las demostraremos aquí. 

Si la ecuación es $X = XA + B$, entonces la solución es BA^* , y ésta cumple las mismas propiedades que en el caso de A^*B .

Como ejemplo, considerad la ecuación $X = (a + ab)X + aa$. Aquí $A = a + ab$ y $B = aa$. Según el lema de Arden, una solución de esta ecuación es $X = (a + ab)^*aa$. Dado que $\lambda \notin (a + ab)$, ya se puede afirmar que esta solución es la única solución que existe.


Tened cuidado con las ecuaciones de la forma $X = AX$. La solución para estas ecuaciones es $X = \emptyset$ y no $X = A^*$. Fijaos en que $X = AX$ es lo mismo que $X = AX + \emptyset$ y que, por tanto, su solución debe ser $A^*\emptyset = \emptyset$. La ecuación que tiene por solución A^* es $X = AX + \lambda$.

Utilizando el lema de Arden, se puede encontrar la expresión regular que describe el lenguaje reconocido por un autómata. Será necesario, en primer lugar, definir el lenguaje asociado a un estado de un autómata.

Dado un DFA $A = (Q, \Sigma, q_0, F, \delta)$, L_i denota el **lenguaje de las palabras que son reconocibles por el autómata desde el estado q_i** . L_i se define como $L_i = \{w \mid \tilde{\delta}(q_i, w) \in F\}$.

El lenguaje L_i ...

... es el conjunto de palabras que serían reconocidas por el autómata si el estado inicial fuera q_i en lugar de q_0 .

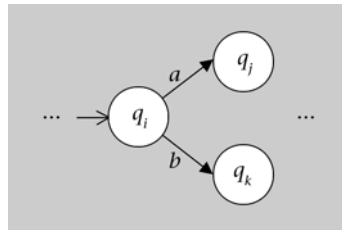
Notad que: 

1) El lenguaje reconocido por A es L_0 ($L_0 = L(A)$). Sólo es necesario recordar que el lenguaje aceptado por un DFA se define como el conjunto de palabras que llevan al autómata a un estado aceptador a partir de un estado inicial ($L(A) = \{w \in \Sigma \mid \tilde{\delta}(q_0, w) \in F\}$).

2) $\lambda \in L_i \Leftrightarrow q_i \in F$. Si λ es reconocible desde un estado q_i , entonces este estado debe ser un estado aceptador (λ no provoca ninguna transición). Por otra parte, ya sabemos que, si un estado aceptador se comporta como un estado inicial, la palabra λ es reconocida.

El paso siguiente consiste en ver que el lenguaje asociado a cada estado se puede expresar en forma de ecuación, en la que intervienen los estados sucesores

y los símbolos que etiquetan las transiciones hacia los estados sucesores, como muestra la siguiente figura:



La ecuación resultante sería la siguiente:

$$L_i = aL_j + bL_k.$$

Si q_i es un estado aceptador ($q_i \in F$), entonces:

$$L_i = aL_j + bL_k + \lambda.$$

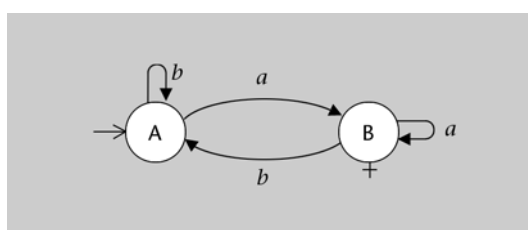
¿Cuál es la razón de estas ecuaciones? La ecuación $L_i = aL_j + bL_k$ responde a la pregunta “¿cuáles son las palabras reconocidas desde el estado q_i ?”, poniendo de manifiesto que las palabras reconocidas desde el estado q_i son aquellas que “empiezan por a , y el resto deben ser reconocidas desde el estado j ” y aquellas que “empiezan por b , y el resto deben ser reconocidas a partir del estado k ”. Si $q_i \in F$, entonces λ también es reconocida, lo cual explica la segunda ecuación.

El tercer y último paso consiste en darse cuenta que cualquier DFA da lugar a un sistema de ecuaciones lineales en el que intervienen los lenguajes asociados a cada estado.

Cuando se resuelve el sistema y se encuentra la expresión regular que describe el lenguaje asociado al estado inicial, entonces describe, también, el lenguaje reconocido por el autómata, ya que $L_0 = L(A)$.

Ejemplos de utilización del lema de Arden

Ejemplo 1. Queremos encontrar la expresión regular que describe el lenguaje reconocido por el autómata representado en el diagrama que podemos ver a continuación:



En primer lugar, encontramos el sistema de ecuaciones que describen el autómata. Éste es el siguiente:

$$\begin{cases} L_A = bL_A + aL_B \\ L_B = aL_B + bL_A + \lambda \end{cases}$$

Después comenzamos a resolver el sistema, teniendo en cuenta que nuestro objetivo es L_A (A es el estado inicial del autómata).

La ecuación $L_B = aL_B + bL_A + \lambda$ tiene por solución $L_B = a^*(bL_A + \lambda)$. Fijaos en que para encontrar esta solución sólo es necesario aplicar el lema de Arden (L_B es la incógnita, a es A y $bL_A + \lambda$ es B).

Sustituimos L_B por $a^*(bL_A + \lambda)$ en la primera ecuación y obtenemos la expresión que vemos a continuación:

$$L_A = bL_A + aa^*(bL_A + \lambda).$$

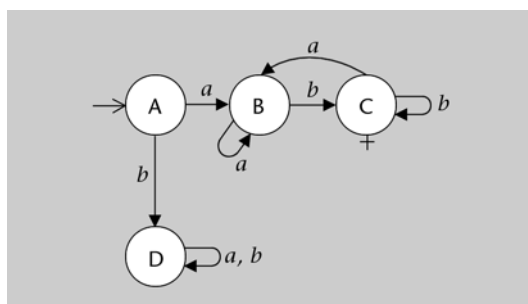
Ahora es necesario que operemos para que el lema de Arden sea aplicable a la ecuación: $L_A = bL_A + aa^*(bL_A + \lambda)$; distribuyendo aa^* queda $L_A = bL_A + aa^*bL_A + aa^*\lambda$ y haciendo L_A factor común de bL_A y de aa^*bL_A , $L_A = (b + aa^*b)L_A + aa^*\lambda$. En este momento ya podemos aplicar el lema de Arden: $L_A = (b + aa^*b)^*L_A + aa^*\lambda$.

Así pues, el lenguaje reconocido por el autómata dado es el descrito por $(b + aa^*b)^*aa^*$.

Simplificación del resultado

La solución hallada puede simplificarse un poco: $b + aa^*b$ es lo mismo que $b + a^*b$ y esto es lo mismo que $(\lambda + a^*)b$. Dado que $\lambda + a^*$ es a^* , tenemos que $(\lambda + a^*)b$ es a^*b . De esta manera, el lenguaje reconocido por el autómata dado también está descrito por la expresión regular: $(a^*b)^*aa^*$.

Ejemplo 2. Queremos hallar la expresión regular que describe el lenguaje reconocido por el autómata representado en el diagrama que encontramos a continuación:



El sistema de ecuaciones que describe al autómata es:

$$\begin{cases} L_A = aL_B + bL_D, \\ L_B = aL_B + bL_C, \\ L_C = aL_B + bL_C + \lambda, \\ L_D = (a + b)L_D. \end{cases}$$

Fijaos en que la solución para L_D es $L_D = \emptyset$. No es necesario llegar a aplicar el lema de Arden para descubrir este resultado; es suficiente con constatar que D es un estado pozo, de manera que ninguna palabra puede ser reconocida a partir de sí misma.

De la ecuación $L_C = aL_B + bL_C + \lambda$, obtenemos la solución $L_C = b^*(aL_B + \lambda)$.

Sustituyendo en la ecuación $L_B = aL_B + bL_C$, queda así: $L_B = aL_B + bb^*(aL_B + \lambda)$. Y manipulándola algebraicamente, $L_B = (a + bb^*a)L_B + bb^*$. Simplificando un poco antes de resolverla, queda $L_B = b^*aL_B + b^*$. Finalmente, aplicando el lema de Arden, obtenemos $L_B = (b^*a)^*b^*$.


El último paso es resolver $L_A = aL_B + bL_D$, la ecuación que describe el lenguaje reconocido por el autómata. Si se sustituye L_B por $(b^*a)^*b^*$ y L_D por \emptyset , queda $L_A = a(b^*a)^*b^* + b\emptyset = a(b^*a)^*b^*$. Así pues, el lenguaje reconocido por el autómata queda descrito por la expresión regular $a(b^*a)^*b^*$.

Simplificación del resultado

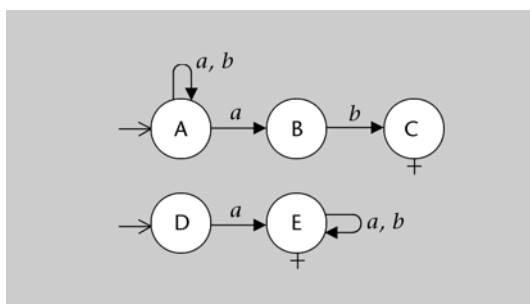
De nuevo, el resultado puede simplificarse un poco. La expresión regular $a(b^*a)^*b^*$ también se puede escribir $a(b^*a)^*bb^*$. Si consideramos la parte final de esta expresión, veremos que b y b^* pueden intercambiar sus posiciones, ya que $bb^* = b^*b = b^+$, de manera que la expresión también la podemos escribir $a(b^*a)^*b^+b$. Si ahora nos centramos en la parte central de esta expresión regular, $(b^*a)^*b^+$, podremos ver que es $(a + b)^*$ (no es difícil ver que cualquier palabra de Σ^* puede escribirse como la concatenación de palabras de b^+a con una palabra de b^+ . Sólo debemos pensar que la palabra de Σ^* se corta cada vez que aparece una a , y da lugar a una secuencia de palabras de b^+a . La palabra de b^+ del final recoge todos los símbolos b después del último símbolo a). Así pues, el lenguaje reconocido por el autómata dado también está descrito por la expresión regular $a(a + b)^*b$ (palabras que empiezan con un símbolo a y acaban con un símbolo b).

Si tenemos que escoger...

... entre aplicar el método a un autómata indeterminista o a su versión determinizada, es mejor aplicarlo al autómata indeterminista, porque dará lugar a un sistema con menos ecuaciones.

Este mismo método también es aplicable a autómatas indeterministas. Sólo deberemos tener en cuenta que el lenguaje reconocido por el autómata será la unión de los lenguajes asociados a los estados iniciales. 

Ejemplo 3. Queremos hallar la expresión regular que describe el lenguaje reconocido por:



El sistema de ecuaciones es el siguiente:

$$\begin{cases} L_A = (a + b)L_A + aL_B. \\ L_B = bL_C. \\ L_C = \lambda. \\ L_D = aL_E. \\ L_E = (a + b)L_E + \lambda. \end{cases}$$


El hecho de que algunos estados tengan transiciones indefinidas simplifica la tarea de resolución:

- $L_E = (a + b)^*$.
- $L_D = aL_E = a(a + b)^*$ (no ha sido necesario aplicar el lema de Arden. La sustitución ha sido suficiente).
- $L_B = bL_C = b\lambda = b$ (de nuevo la sustitución ha sido suficiente).
- $L_A = (a + b)L_A + aL_B = (a + b)L_A + ab = (a + b)^*ab$.

El lenguaje reconocido por el autómata es el descrito por $L_A + L_D$ (unión de los lenguajes asociados a los estados iniciales); $L_A + L_D = (a + b)^*ab + a(a + b)^*$ (palabras que acaban con ab o que empiezan con a).

6. Determinación de la no-regularidad de un lenguaje: el lema del bombeo

6.1. Una condición necesaria para la regularidad

Ya se ha comentado, en puntos anteriores de este mismo módulo, que no todos los lenguajes sobre Σ^* son regulares. Sin embargo, no disponemos de ninguna herramienta formal que nos permita razonar de forma general, sobre la regularidad o no-regularidad de un lenguaje. En este último apartado expon-dremos el *lema del bombeo*, que nos permitirá: 

- 1) Entender la naturaleza repetitiva de las palabras de los lenguajes regulares.
- 2) Razonar sobre la no-regularidad de algunos lenguajes.

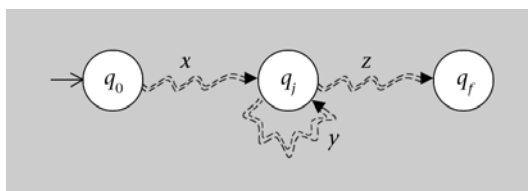
Enunciado informalmente, el **lema del bombeo** afirma que si L es un lenguaje regular y $w \in L$ es una palabra lo bastante larga, entonces es inevitable que pase lo siguiente: la palabra w se puede descomponer en tres subpalabras xyz ($w = xyz$), de manera que $xy^2z \in L$, $xy^3z \in L$, ..., $xy^iz \in L$ ($\forall i$ $xy^iz \in L$). En otras palabras, si w es suficientemente larga, entonces todas las palabras resultantes de repetir tantas veces como se quiera una determinada parte central continúan perteneciendo a L .

El lema del bombeo...

... recibe este nombre porque afirma que la parte central (y) de la palabra $w = xyz \in L$ se puede hacer crecer tanto como se quiera sin que esto afecte a la pertenencia de la palabra resultante al lenguaje L . Este "hacer crecer" la parte central se denomina *bombeo de la palabra*.


- Sea L regular y $A = (Q, \Sigma, q_0, F, \delta)$ el DFA mínimo que lo reconoce.
- Sea N el número de estados de A ($N = |Q|$).
- Sea $w \in L$ una palabra de longitud superior a N ($m = |w| \geq N$).

Dado que w tiene más símbolos que estados tiene A , cuando es reconocido, debe pasar por algún estado más de una vez. Denominamos q_i al estado por el que el autómatas pasa más de una vez durante el reconocimiento de w . Gráficamente, lo podemos representar de la siguiente manera:



Para reconocer...

... una palabra de m símbolos, el autómatas debe efectuar m transiciones (una por cada símbolo) que involucran $m + 1$ estados.

El reconocimiento de w se puede dividir en tres etapas: 

- 1) Primera etapa: desde q_0 hasta q_i .
- 2) Segunda etapa: desde q_i hasta q_i .
- 3) Tercera etapa: desde q_i hasta q_f .

En la figura anterior, hemos denominado x , y y z a las palabras involucradas en la primera, en la segunda y en la tercera etapas, respectivamente. Fijaos en que $w = xyz$ y que: $\tilde{\delta}(q_0, x) = q_j$, $\tilde{\delta}(q_j, y) = q_j$, $\tilde{\delta}(q_j, z) = q_f \in F$.

Es importante darse cuenta de que desde el estado q_j la palabra y hace que el autómata vuelva al estado q_j . Esto provoca que:

- $\tilde{\delta}(q_j, \lambda) = q_j$ (por la definición de $\tilde{\delta}$; si no hay entrada, el autómata no efectúa ninguna transición).
- $\tilde{\delta}(q_j, yy) = \tilde{\delta}(\tilde{\delta}(q_j, y), y) = \tilde{\delta}(q_j, y) = q_j$.
- $\tilde{\delta}(q_j, yyy) = \tilde{\delta}(\tilde{\delta}(\tilde{\delta}(q_j, yy), y), y) = \tilde{\delta}(q_j, y) = q_j$.

Y, en general, $\forall i \geq 0 \tilde{\delta}(q_j, y^i) = q_j$.

Así pues, podemos concluir que $\forall i \geq 0 \tilde{\delta}(q_0, xy^iz) = \tilde{\delta}(\tilde{\delta}(q_0, x), y^iz) = \tilde{\delta}(q_j, y^iz) = \tilde{\delta}(\tilde{\delta}(q_j, y^i), z) = \tilde{\delta}(q_j, z) = q_f \in F$. El autómata A reconoce todas las palabras de la forma xy^iz con $i \geq 0$.


Resumiendo, se acaba de demostrar que si w es suficientemente larga, entonces:

$$w = xyz \in L \Rightarrow xy^iz \in L \quad \forall i \geq 0.$$

Enunciado formalmente, el lema del bombeo dice que, si L es un lenguaje regular ($L \in \text{REG}$), entonces pasa que:

$$\exists N \geq 1 \quad \forall w \in L \quad [|w| \geq N \rightarrow \exists x, y, z \\ (w = xyz \wedge |xy| \leq N \wedge |y| \geq 1 \wedge \forall i \geq 0 (xy^iz \in L))].$$

6.2. Una condición suficiente para la no-regularidad

Fijaos en que el lema del bombeo es una condición necesaria de regularidad. Esto significa que explica algo que pasa en los lenguajes regulares, pero que también puede pasar en lenguajes que no lo son: 

1) Si se detecta que en un lenguaje todas las palabras a partir de una determinada longitud son “abombables” ($\forall i \geq 0 xy^iz \in L$), esto no significa que el lenguaje sea necesariamente regular.

2) Por otro lado, si se detecta que en un determinado lenguaje es posible encontrar palabras de todas las longitudes posibles que no son “abombables”

$(\exists i \geq 0 \ xy^iz \notin L)$, entonces ya se puede afirmar con toda seguridad que el lenguaje no es regular.

Este último planteamiento, que nos permite afirmar con seguridad la no-regularidad de un lenguaje, es el **contrarrecíproco del lema del bombeo** y lo enunciamos a continuación:


Si a un lenguaje L le pasa que:

$$\forall N \geq 1 \ \exists w \in L \ [|w| \geq N \wedge \forall x, y, z \ (w = xyz \wedge |x| \leq N \wedge |y| \geq 1 \rightarrow \rightarrow \exists i \geq 0 \ (xy^iz \notin L))],$$

entonces L no es un lenguaje regular.

Recordad

... que, cuando estudiabais lógica, $A \rightarrow B$ era lo mismo que $\neg B \rightarrow \neg A$ (si A es suficiente para B , entonces B es necesario para A). Esta ley ($A \rightarrow B \vdash \neg B \rightarrow \neg A$) se conoce con el nombre de *ley del contrarrecíproco*.

Estudiando atentamente esta condición suficiente de no-regularidad, descubriremos que para demostrar que un lenguaje L no es regular es necesario: 

- 1) Para cada posible longitud, encontrar una palabra de aquella longitud o superior.
- 2) Para cada una de las palabras, considerar todas las posibles factorizaciones de esta palabra en tres subpalabras, x , y y z , que cumplan las condiciones $|x| \leq N$ y $|y| \geq 1$.
- 3) Para cada una de las posibles factorizaciones de cada una de las palabras, encontrar una constante i tal que $xy^iz \notin L$.

Si esto se consigue, ya se puede afirmar que L no es regular.

Ejemplos de demostraciones de no-regularidad de lenguajes

Ejemplo 1. Demostramos la no-regularidad del lenguaje $L = \{w \mid |w|_a = |w|_b\}$.

a) Para cada longitud, es necesario encontrar una palabra del lenguaje que tenga, como mínimo, aquella longitud. Es necesario tener en cuenta que no puede ser cualquier palabra, sino que debe ser una palabra que permita realizar los dos pasos siguientes. Esta parte es la más complicada, ya que si las palabras no están bien escogidas, los dos puntos siguientes fracasarán.

Fijaos también en que el número de palabras que debemos encontrar es infinito (una por cada longitud a partir de 1). Daremos las palabras como función de N (la longitud mínima).

En este caso concreto, las palabras que se escogerán son $a^N b^N$ (para $N = 1$ la palabra es ab , para $N = 2$ es $aabb$, etc.).

b) Para cada una de las palabras se deben considerar todas las posibles descomposiciones en tres subpalabras de la forma xyz , en las cuales $|x| \leq N$ y $|y| \geq 1$.

En el caso concreto de palabras que presentan la forma $a^N b^N$ (las palabras que hemos escogido en el apartado anterior), las subpalabras x , y y z serán de la forma $x = a^p$, $y = a^q$, $z = a^{N-p-q} b^N$, con $p + q \leq N$ y $q \geq 1$.

Resumiendo, cada palabra $a^N b^N$ se expresa como $a^p a^q a^{N-p-q} b^N$.

Fijaos en que las palabras se han escogido de manera que el prefijo xy sólo contenga símbolos a .

c) Para cada descomposición es necesario hallar una constante i tal que $xy^i z \notin L$. No es necesario que la constante sea la misma para cada descomposición; pero en este caso sería suficiente con que consideremos únicamente la constante $i = 2$, ya que $xy^2 z = a^p a^{2q} a^{N-p-q} b^N = a^{p+2q+N-p-q} b^N = a^{N+q} b^N$.

Las palabras de la forma $a^{N+q} b^N$ no son de L , ya que $N + q \neq N$, porque $q \geq 1$.

Se habrían obtenido resultados similares considerando cualquier constante $i \neq 1$.

Se acaba de demostrar, pues, que L no es un lenguaje regular.

Ejemplo 2. Demostramos la no-regularidad de $L = \{a^n \mid n \text{ es un número primo}\}$.

a) Para cada posible longitud N consideraremos la palabra $a^{f(N)}$, donde $f(N)$ es cualquier número primo que sea superior a N .

b) Las descomposiciones de las palabras $a^{f(N)}$ en tres subpalabras, x , y y z , en las que $|x| \leq N$ y $|y| \geq 1$, deben ser todas de la forma $x = a^p$, $y = a^q$, $z = a^{f(N)-p-q}$.

c) Si para cada descomposición de cada palabra consideramos la constante $i = f(N) + 1$, tendremos que la palabra $xy^i z$ no es de L . Veámoslo:

$$xy^i z = xy^{f(N)+1} z = a^p a^{q(f(N)+1)} a^{f(N)-p-q} = a^{f(N)(q+1)}.$$

Resulta obvio que el número $f(N)(q+1)$ no es primo, porque como podemos observar tiene, como mínimo, dos factores que no son ni él mismo ni 1.

Concluye así la demostración de que L no es un lenguaje regular.

Las palabras...

... siempre se escogen de manera que todas las descomposiciones que se deben considerar posteriormente tengan la forma más simple posible.

Resumen

Hemos comenzado este módulo didáctico explicando que los **autómatas finitos deterministas** son los reconocedores de los lenguajes denominados *regulares*. La relación entre los autómatas finitos deterministas y los lenguajes regulares es tan estrecha que un lenguaje es regular si, y sólo si, hay un autómata finito determinista que lo reconoce.

Los autómatas finitos deterministas son el modelo que la informática teórica da para aquellos algoritmos que pueden efectuar la tarea que les es propia, utilizando una cantidad fija de memoria, con independencia total del tamaño de la entrada.

Los **autómatas finitos indeterministas** son una herramienta conceptual que facilita el diseño de los deterministas. Los lenguajes reconocidos por los autómatas finitos indeterministas son, exactamente, los mismos que los reconocidos por los deterministas, y hay un procedimiento algorítmico que permite construir, para cualquier autómata indeterminista, un autómata determinista equivalente, que reconoce el mismo lenguaje. Como contrapartida negativa, el procedimiento de determinización puede provocar, en algunos casos, un crecimiento exponencial en el número de estados.

Los lenguajes regulares son cerrados para las operaciones conjuntistas habituales (complementación, unión e intersección), la concatenación, los cierres y las inversiones. Este hecho se pone de manifiesto de manera constructiva cuando se dan procedimientos que permiten construir autómatas finitos que reconocen el complementario, la unión, etc. de lenguajes regulares a partir de los autómatas finitos que los reconocen.

Cualquier lenguaje regular puede ser reconocido por infinidad de autómatas finitos deterministas. Sin embargo, el autómata finito determinista que lo reconoce, con el menor número de estados posible, es único, y hay un algoritmo que permite encontrarlo. La unicidad del **autómata mínimo** permite determinar si dos autómatas reconocen el mismo lenguaje, así como determinar si dos lenguajes son exactamente el mismo o no.

Las **expresiones regulares** proporcionan una manera simple, y al mismo tiempo precisa, de describir lenguajes regulares. Además, se pueden manipular de la manera algebraica habitual. La relación que se establece entre los autómatas finitos y las expresiones regulares se pone de manifiesto demostrando que, para cualquier expresión regular, hay un autómata finito que reconoce el lenguaje que describe, y que cada autómata finito reconoce un lenguaje que se puede describir con una expresión regular.

Para determinar la expresión regular que describe el lenguaje reconocido por un autómata finito, se dispone del **lema de Arden**, que permite solucionar ecuaciones lineales con lenguajes.

El **lema del bombeo** pone de manifiesto la naturaleza repetitiva de las palabras de los lenguajes regulares. Este lema expresa una condición necesaria para la regularidad de un lenguaje. Su contrarrecíproco expresa una condición suficiente de no-regularidad que se puede utilizar para demostrar que determinados lenguajes no son regulares.

Ejercicios de autoevaluación

1. Dad DFA que reconozcan los siguientes lenguajes definidos sobre el alfabeto $\{a, b\}$:

- $L = \{abba\}$.
- $L = \{abba, ababba\}$.
- $L = \{a\}^* = \{a^n \mid n \geq 0\} = \{w \mid |w|_b = 0\}$.
- $L = \{a\}^+$.
- $L = \{w \mid |w| \geq 3\}$.
- $L = \{w \mid |w| < 4\}$.
- $L = \{w \mid |w|_a = 3\}$.
- $L = \{w \mid |w|_b = 4n, n \geq 0\}$.
- $L = \{w \mid |w|_b = 4n, n > 0\}$.
- $L = \{w \mid \exists x, y \in \Sigma^* (w = xy \wedge |y| = 3 \wedge |y|_a \geq 2)\}$ (palabras de longitud mínima 3 que tienen en los tres últimos símbolos dos a como mínimo).
- Palabras sobre $\Sigma = \{0, 1\}$ que codifican en binario un número que es múltiplo de 5. (Pista 1: un estado por cada posible resto de la división por 5. Pista 2: en binario, añadir un 0 a la derecha significa multiplicar por 2, y añadir un 1 significa multiplicar por 2 y sumar 1).

2. Dad NFA que reconozcan los siguientes lenguajes:

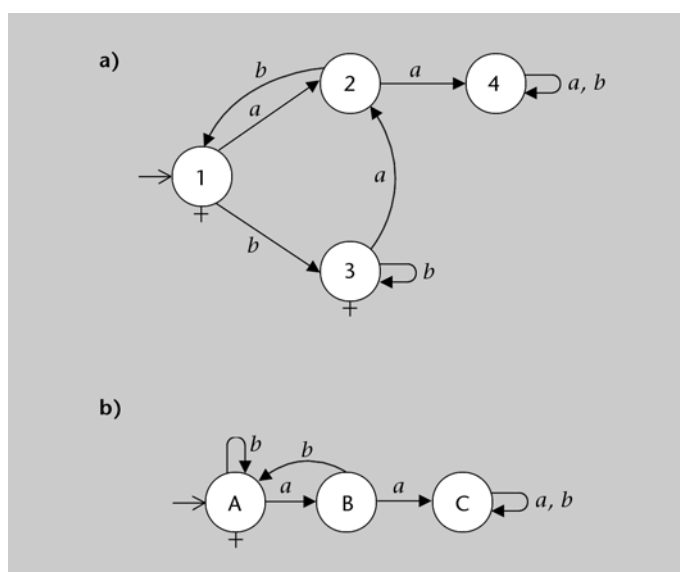
- $L = \{abba\}$.
- $L = \{abba, ababba\}$.
- $L = \{w \mid \exists x, y \in \Sigma^* (w = xy \wedge \exists z \in \Sigma^2 (y = az))\} = \Sigma^* \{a\} \{a, b\}^2$ (palabras cuyo antepenúltimo símbolo es a).

3. Determinizad los NFA que se dan como soluciones del ejercicio 2.

4. Dad DFA con el mínimo número de estados que reconozcan cada uno de los siguientes lenguajes regulares. No los intentéis encontrar “directamente”. Tratad de reducir cada problema a un conjunto de subproblemas (complementaciones, uniones, concatenaciones, etc.) más simples y utilizad las operaciones con autómatas que hemos visto en el apartado 3 para construir la solución final:

- Palabras con un número par de a y un número par de b .
- Palabras que, si contienen la cadena aba , entonces también contienen la cadena bab (si no contienen aba , no es necesario que contengan bab).
- Palabras en las que cada símbolo que ocupa una posición múltiplo de 3 es una b .
- Palabras en las que toda subpalabra de tres símbolos contiene dos a como mínimo.
- Palabras en las que todo prefijo tiene un número par de a o un número par de b .

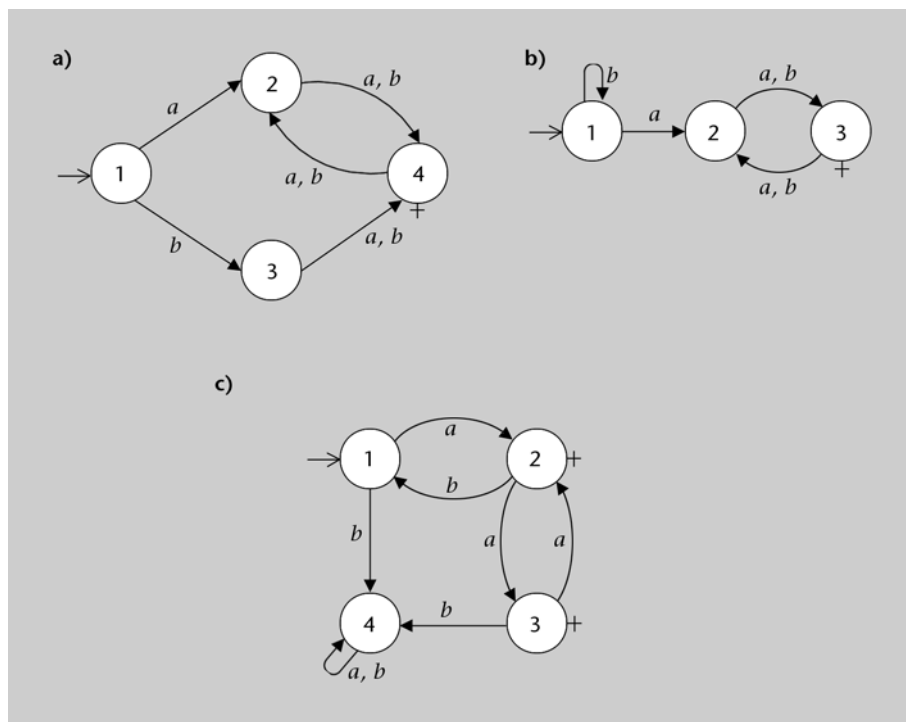
5. Demostrad que los autómatas siguientes reconocen exactamente el mismo lenguaje, y que éste es $(b^* + (ab)^*)^+$:



6. Construid los NFA que reconocen los lenguajes descritos por las siguientes expresiones regulares:

- $a^*ba^*ab^*$.
- $((aab^* + a^4b)^*)$.

7. Hallad expresiones regulares que describan los lenguajes reconocidos por los siguientes autómatas finitos:



8. Demostrad la no-regularidad de los siguientes lenguajes:

- a) $L = \{a^m b^n a^{m+n} \mid m, n \geq 1\}$.
- b) $L = \{w \mid w = w^R\}$ (lenguaje de todas las palabras palíndromas).
- c) $L = \{a^n \mid n \text{ no es un número primo}\}$.

9. Razonad sobre la veracidad o la falsedad de las afirmaciones que se dan a continuación. En el caso de que sean verdaderas, dad ejemplos de ello:

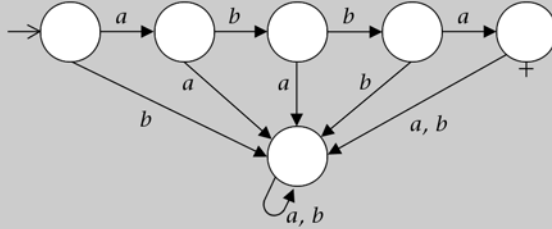
- a) Un lenguaje regular puede tener un subconjunto que sea un lenguaje no regular.
- b) La estrella de Kleene de un lenguaje regular puede ser no regular.
- c) La intersección de dos lenguajes no regulares puede ser regular.

Solucionario

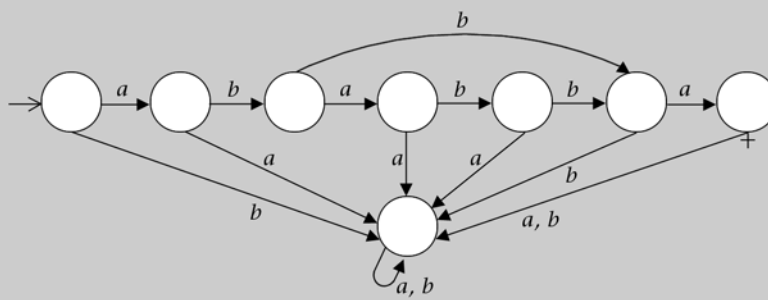
Ejercicios de autoevaluación

1. La solución de los apartados del a al g de este ejercicio se representan en el siguiente gráfico:

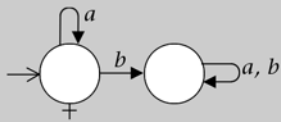
a)



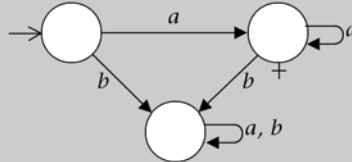
b)



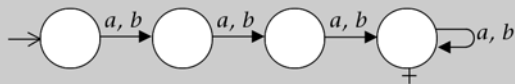
c)



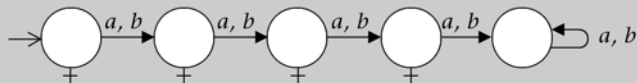
d)



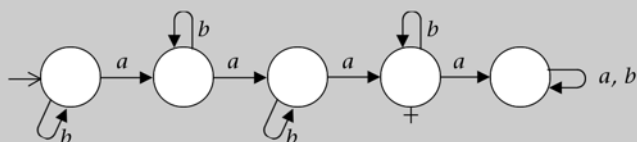
e)



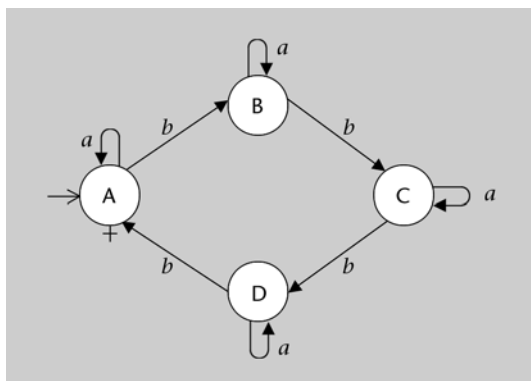
f)



g)



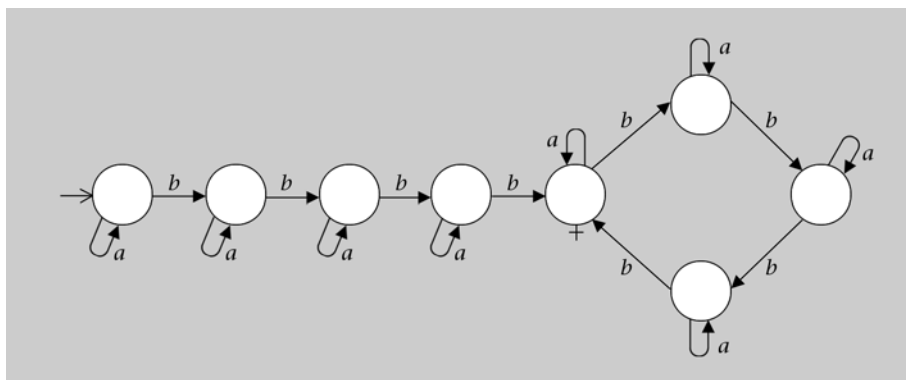
h) Para este lenguaje, el DFA que lo reconoce es:



Fijaos con qué situación se corresponde cada estado:

- Estado A: la palabra analizada hasta ahora tiene $4n$ ($n \geq 0$) b .
- Estado B: la palabra analizada hasta ahora tiene $4n + 1$ ($n \geq 0$) b .
- Estado C: la palabra analizada hasta ahora tiene $4n + 2$ ($n \geq 0$) b .
- Estado D: la palabra analizada hasta ahora tiene $4n + 3$ ($n \geq 0$) b .

i) Para este lenguaje, el DFA que lo reconoce es:



Fijaos en que es necesario haber “procesado” cuatro b antes de poder entrar en el ciclo que se ha visto en el apartado anterior.

j) Para encontrar al autómatas que reconoce este lenguaje, se puede proceder de la siguiente manera: las posibles terminaciones de una palabra cuando se consideran los tres últimos símbolos son aaa , aab , aba , abb , baa , bab , bba y bbb . A cada una de estas terminaciones le haremos corresponder un estado. Para recordar la correspondencia entre terminaciones y estados, a éstos los llamaremos AAA, AAB, ABA, ABB, BAA, BAB, BBA y BBB. De estos ocho estados, cuatro se corresponden con terminaciones con dos o más a : AAA, AAB, ABA y BAA. Estos estados serán finales.

Las transiciones entre estos estados son más fáciles de determinar. Por ejemplo, si a la terminación aba se le añade un símbolo a , se convierte en la terminación $abaa$. Sin embargo, dado que sólo estamos interesados en los tres últimos símbolos, la terminación $abaa$ la vemos como baa . Así pues, desde el estado ABA se pasa al estado BAA con un símbolo a . Para el resto de los estados y símbolos se puede razonar de la misma forma y obtener una tabla como la siguiente:

δ	a	b
†AAA	AAA	AAB
†AAB	ABA	ABB
†ABA	BAA	BAB
ABB	BBA	BBB

(Continúa en la página siguiente.)

δ	<i>a</i>	<i>b</i>
†BAA	AAA	AAB
BAB	ABA	ABB
BBA	BAA	BAB
BBB	BBA	BBB

Sin embargo, no todas las palabras tienen como mínimo tres símbolos: hay palabras de un símbolo (*a* y *b*), de dos (*aa*, *ab*, *ba* y *bb*) y de ninguno (λ). A cada una de estas palabras le haremos corresponder un estado, que denominaremos A, B, AA, BA, BB y L. El estado L (correspondiente a λ) será el estado inicial.

Finalmente, la tabla de transición completa de todo el autómata queda como exponemos a continuación:

δ	<i>a</i>	<i>b</i>
→L	A	B
A	AA	AB
B	BA	BB
AA	AAA	AAB
AB	ABA	ABB
BA	BAA	BAB
BB	BBA	BBB
†AAA	AAA	AAB
†AAB	ABA	ABB
†ABA	BAA	BAB
ABB	BBA	BBB
†BAA	AAA	AAB
BAB	ABA	ABB
BBA	BAA	BAB
BBB	BBA	BBB

k) La solución de este problema se obtiene razonando de la siguiente forma: en función del resto de la división por 5, las palabras se pueden clasificar en cinco clases: palabras de valor $5n$, de valor $5n + 1$, de valor $5n + 2$, de valor $5n + 3$ y de valor $5n + 4$. Consideraremos un estado para cada una de estas clases, a los cuales llamaremos R_0 , R_1 , R_2 , R_3 y R_4 , respectivamente. El estado R_0 será aceptador porque es el que se corresponde con las palabras que tienen un valor decimal múltiplo de 5. Para determinar las transiciones se razona de la siguiente forma: si a un número binario se le añade un 0 a la derecha, su valor decimal se multiplica por 2; y si se añade un 1, su valor decimal se multiplica por 2 y se incrementa en una unidad. Entonces:

- A un múltiplo de 5 se le añade un 0 a la derecha: $(5n) \cdot 2 = 5(2n)$ (múltiplo de 5).
- A un múltiplo de 5 se le añade un 1 a la derecha: $(5n) \cdot 2 + 1 = 5(2n) + 1$ (resto 1).

Esto significa que para el estado R_0 , las transiciones serán:

δ	0	1
† R_0	R_0	R_1

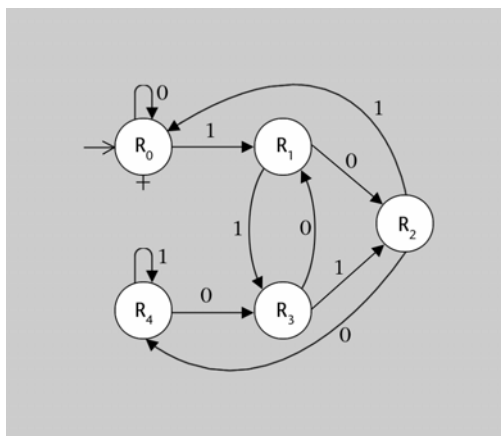
Para el resto de estados se procede del mismo modo:

- $(5n + 1) \cdot 2 = 5(2n) + 2$ (resto 2).
- $(5n + 1) \cdot 2 + 1 = 5(2n) + 3$ (resto 3).
- $(5n + 2) \cdot 2 = 5(2n) + 4$ (resto 4).
- $(5n + 2) \cdot 2 + 1 = 5(2n) + 5 = 5(2n + 1)$ (resto 0).
- $(5n + 3) \cdot 2 = 5(2n) + 6 = 5(2n + 1) + 1$ (resto 1).
- $(5n + 3) \cdot 2 + 1 = 5(2n) + 7 = 5(2n + 1) + 2$ (resto 2).
- $(5n + 4) \cdot 2 = 5(2n) + 8 = 5(2n + 1) + 3$ (resto 3).
- $(5n + 4) \cdot 2 + 1 = 5(2n) + 9 = 5(2n + 1) + 4$ (resto 4).

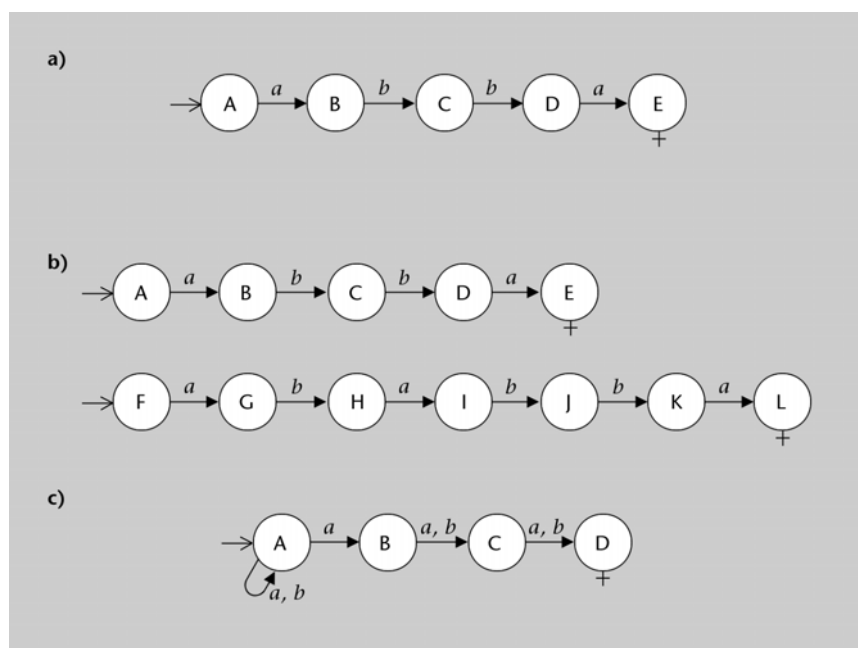
Así, la tabla de la función de transición queda de la siguiente forma:

δ	0	1
$\rightarrow \dagger R_0$	R_0	R_1
R_1	R_2	R_3
R_2	R_4	R_0
R_3	R_1	R_2
R_4	R_3	R_4

Gráficamente, la representamos así:



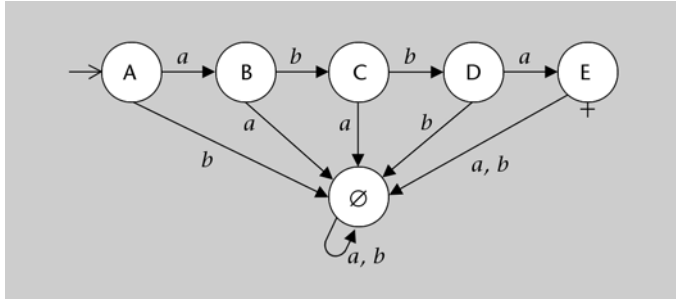
2. Las soluciones de los apartados a, b y c de este ejercicio se representan en el siguiente gráfico:



3.

a) En todos los casos en los que la única fuente de indeterminismo es la existencia de transiciones no definidas (sin más de un estado inicial y sin transiciones definidas de forma múltiple), la determinización consiste en añadir un estado pozo al que se dirigirán todas las transiciones que antes no estaban definidas.

Así, la figura resultante será la siguiente:



b) La tabla de transición del autómata indeterminista y la del determinista son, respectivamente:

δ	a	b
$\rightarrow A$	B	\emptyset
B	\emptyset	C
C	\emptyset	D
D	E	\emptyset
$\dagger E$	\emptyset	\emptyset
$\rightarrow F$	G	\emptyset
G	\emptyset	H
H	I	\emptyset
I	\emptyset	J
J	\emptyset	K
K	L	\emptyset
$\dagger L$	\emptyset	\emptyset

δ	a	b
$\rightarrow AF$	BG	\emptyset
BG	\emptyset	CH
CH	I	D
I	\emptyset	J
D	E	?
J	\emptyset	K
$\dagger E$	\emptyset	\emptyset
K	L	\emptyset
$\dagger L$	\emptyset	\emptyset

Notación

Para simplificar la notación, los conjuntos de estados se han notado con la secuencia ordenada alfabéticamente de los estados que los componen. Por ejemplo, el estado $\{A, F\}$ se ha notado AF, y el estado $\{C, H\}$ se ha notado CH. En las próximas soluciones también utilizaremos esta notación.

Tened presente que en el autómata determinizado, el estado \emptyset es un estado pozo.

c) La tabla de transición del autómata determinista es la siguiente:

δ	a	b
$\rightarrow A$	AB	A
AB	ABC	AC
ABC	ABCD	ACD
AC	ABD	AD
$\dagger ABCD$	ABCD	ACD
$\dagger ACD$	ABD	AD

(Continúa en la página siguiente.)

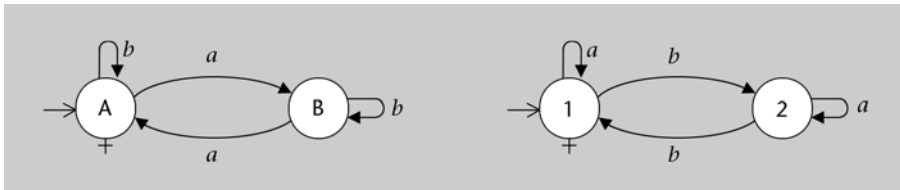
δ	a	b
$\rightarrow \dagger ABD$	ABC	AC
$\dagger AD$	AB	A

4.

a) Claramente, este lenguaje se puede obtener por la intersección del lenguaje de las palabras con un número par de b con el lenguaje de las palabras con un número par de a :

$$L = L_1 \cap L_2 \text{ amb } L_1 = \{w \mid |w|_a = 2n \ (n \geq 0)\} \text{ y } L_2 = \{w \mid |w|_b = 2p \ (p \geq 0)\}.$$

Los DFA que reconocen L_1 y L_2 son, respectivamente, los siguientes:



Las tablas de transición de estos autómatas son, respectivamente:

δ_1	a	b
$\rightarrow \dagger A$	B	A
B	A	B

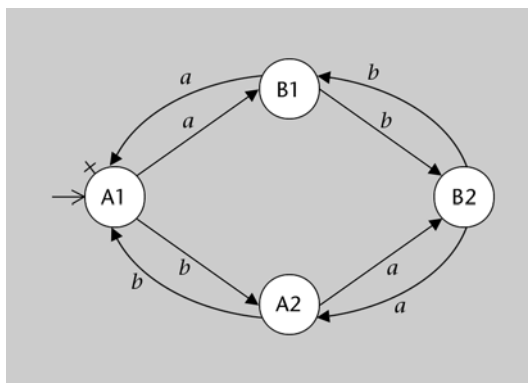
δ_2	a	b
$\rightarrow \dagger 1$	1	2
2	2	1

La tabla de transición del autómata intersección es la siguiente:

δ	a	b
$\rightarrow \dagger A1$	B1	A2
B1	A1	B2
A2	B2	A1
B2	A2	B1

Fijaos en que, por simplicidad, se ha escrito A2 en lugar de $\langle A, 2 \rangle$, B2 en lugar de $\langle B, 2 \rangle$, etc.

Gráficamente, el autómata que reconoce L , y que ya es mínimo, es el siguiente:



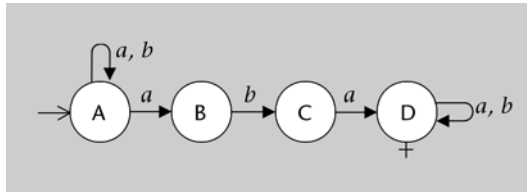
b) En primer lugar, es necesario darse cuenta de que las palabras de este lenguaje se definen utilizando una implicación: “si contienen la cadena aba , entonces contienen la cadena bab ”. O, lo que es lo mismo (equivalencia deductiva): “no contienen la cadena aba o contienen la cadena bab ”. Esto permite apreciar que $L = L_1^c \cup L_2$, donde:

- $L_1 = \{w \mid \exists x, y \in \Sigma^* (w = xabxy)\} = \Sigma^* \{aba\} \Sigma^*$.
- $L_2 = \{w \mid \exists x, y \in \Sigma^* (w = xbabxy)\} = \Sigma^* \{bab\} \Sigma^*$.

Para resolver el problema, se construirá primero un autómata para L_1^c y después otro para L_2 . La solución será el resultado de su unión:

- Construcción del autómata que reconoce L_1^c .

Un NFA que reconoce L_1 es el siguiente:

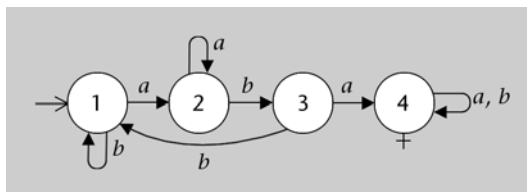


Dado que será necesario complementar al autómata, primero se determiniza. La tabla de la función de transición del autómata determinizado es:

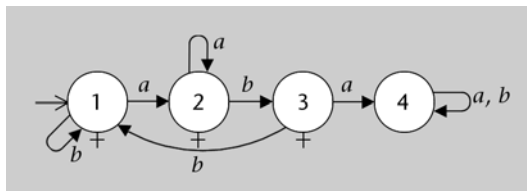
	δ	a	b
(1)	$\rightarrow A$	AB	A
(2)	AB	AB	AC
(3)	AC	ABD	A
(4)	$\dagger ABD$	ABD	ACD
(5)	$\dagger ACD$	ABD	AD
(6)	$\dagger AD$	ABD	AD

Fijaos en que los estados 4, 5 y 6 son equivalentes (no se puede salir de un estado aceptador \rightarrow 4, 5 ó 6 \leftarrow cuando se ha entrado en él). Por esta razón, se hará desaparecer los estados 5 y 6 sólo se mantendrá el estado 4. Gráficamente, representamos el autómata de la manera siguiente:

Para facilitar la representación gráfica de la tabla de transición, hemos asignado un número a cada estado.

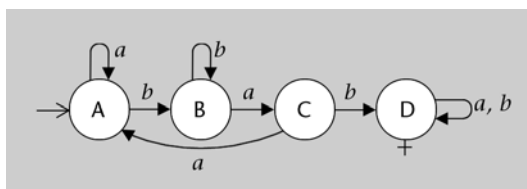


Complementándolo (es decir, intercambiando aceptadores por no aceptadores y no aceptadores por aceptadores), obtenemos el siguiente autómata:



- Construcción del autómata que reconoce L_2 .

Procediendo de manera similar a la que nos ha permitido obtener un autómata que reconoce L_1 , se obtiene el DFA que reconoce L_2 siguiente:



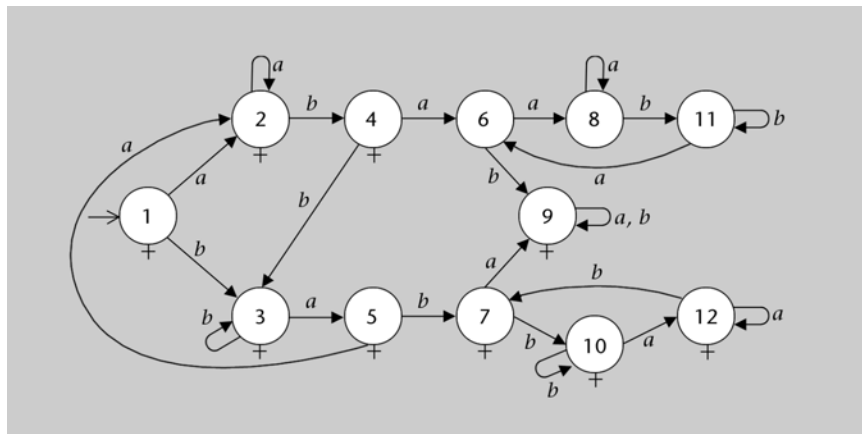
- Construcción del autómata que reconoce $L_1^c \cup L_2$.

Dado que los autómatas que se tienen para L_1^c y para L_2 son deterministas, se utilizará el método de unión por producto cartesiano.

La tabla de transiciones del autómata unión es la siguiente:

	δ	a	b
(1)	$\rightarrow \dagger A1$	A2	B1
(2)	$\dagger A2$	A2	B3
(3)	$\dagger B1$	C2	B1
(4)	$\dagger B3$	C4	B1
(5)	$\dagger C2$	A2	D3
(6)	C4	A4	D4
(7)	$\dagger D3$	D4	D1
(8)	A4	A4	B4
(9)	$\dagger D4$	D4	D4
(10)	$\dagger D1$	D2	D1
(11)	B4	C4	B4
(12)	$\dagger D2$	D2	D3

Gráficamente, representamos el autómata que reconoce la unión de los lenguajes L_1^c y L_2 de la siguiente manera:



No es difícil ver que los estados 7, 9, 10 y 12 son equivalentes. Cuando se minimice el autómata, esto será aún más evidente.

- Minimización del resultado.

$\equiv_0: \{\{1, 2, 3, 4, 5, 7, 9, 10, 12\}, \{6, 8, 11\}\}.$

$\equiv_1: \{\{1, 2, 3, 5, 7, 9, 10, 12\}, \{4\}, \{6\}, \{8, 11\}\}.$

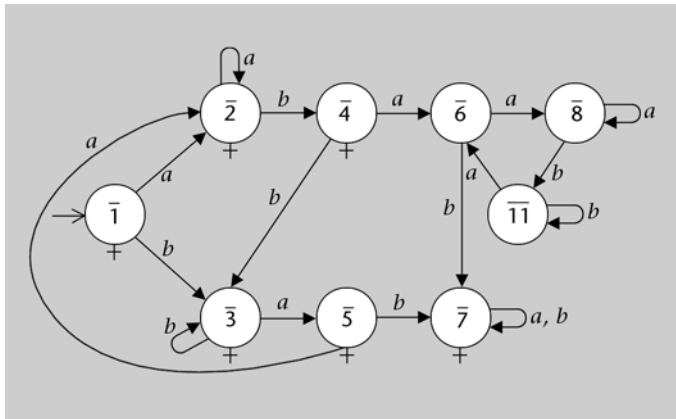
$\equiv_2: \{\{1, 3, 5, 7, 9, 10, 12\}, \{2\}, \{4\}, \{6\}, \{8\}, \{11\}\}.$

$\equiv_3: \{\{1, 5\}, \{3, 7, 9, 10, 12\}, \{2\}, \{4\}, \{6\}, \{8\}, \{11\}\}.$

$\equiv_4: \{\{1, 5\}, \{3\}, \{7, 9, 10, 12\}, \{2\}, \{4\}, \{6\}, \{8\}, \{11\}\}.$

$\equiv_5: \{\{1\}, \{5\}, \{3\}, \{7, 9, 10, 12\}, \{2\}, \{4\}, \{6\}, \{8\}, \{11\}\}.$

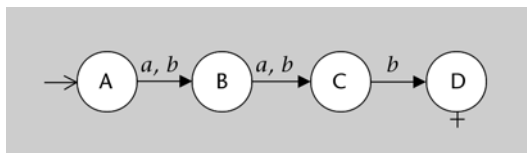
Gráficamente, se representa el autómata fruto de la minimización del resultado de la siguiente forma:



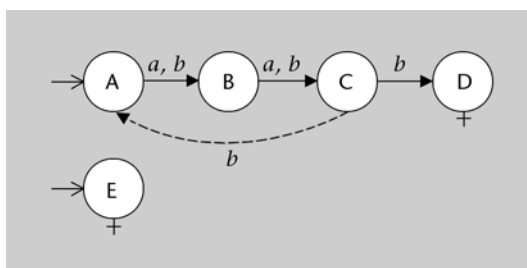
c) El hecho de que cada posición múltiplo de 3 esté ocupada por un símbolo b se puede conseguir utilizando el lenguaje $(\Sigma^2\{b\})^*$ (dos símbolos cualesquiera y una b , dos símbolos más y una b , dos símbolos más y una b ...). Pero todas las palabras de este lenguaje tienen longitud múltiplo de 3, de manera que no incluye palabras como $aaba$, aa , $abbaabaa$, que son del lenguaje para el que se quiere construir el autómata, porque todas tienen las posiciones múltiplos de 3 ocupadas por símbolos b . El problema quedará finalmente solucionado concatenando $(\Sigma^2\{b\})^*$ con $\{\lambda, a, b, aa, ab, ba, bb\}$ (para permitir todas las posibles terminaciones). Así, será necesario construir un autómata para el lenguaje $L = (\Sigma^2\{b\})^*\{\lambda, a, b, aa, ab, ba, bb\}$.

- Construcción de un autómata que reconozca $(\Sigma^2\{b\})^*$.

Un autómata indeterminista que reconoce el lenguaje $\Sigma^2\{b\}$ es el que podéis ver a continuación:

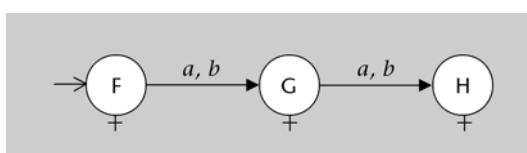


A partir de éste, construimos uno que reconoce $(\Sigma^2\{b\})^*$ y le añadimos un estado inicial y final para poder aceptar $\{\lambda\}$:



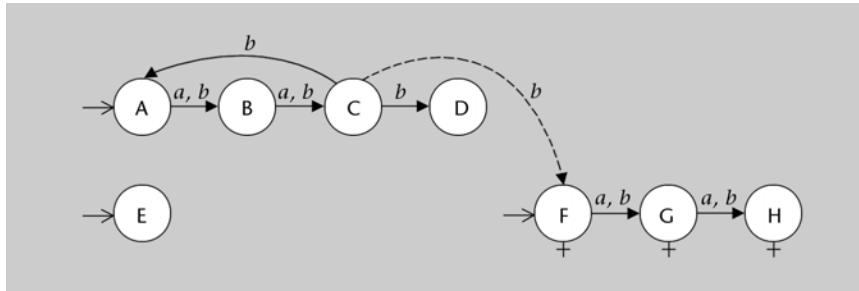
- Construcción de un autómata que reconozca $\{\lambda, a, b, aa, ab, ba, bb\}$.

Un autómata finito que reconoce este lenguaje es el que podéis ver a continuación:



- Construcción del autómata concatenación de los dos anteriores.

Aplicando el método de concatenación se obtiene el autómata siguiente:



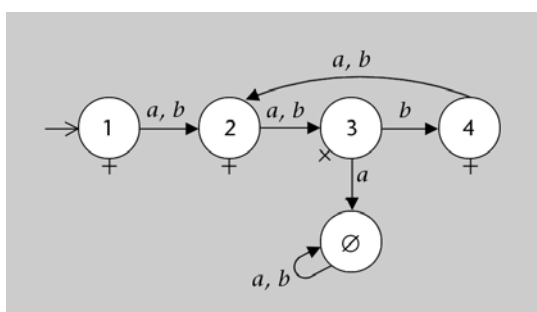
Fijaos en que se mantiene el estado F como estado inicial porque el primer autómata reconoce λ . Fijaos también en que los estados D y E dejan de ser aceptadores.

Las tablas de transición del autómata y de su versión determinizada son, respectivamente:

δ	a	b
$\rightarrow A$	B	B
B	C	C
C	\emptyset	ADF
D	\emptyset	\emptyset
$\rightarrow E$	\emptyset	\emptyset
$\rightarrow \dagger F$	G	G
$\dagger G$	H	H
$\dagger H$	\emptyset	\emptyset

	δ	a	b
(1)	$\rightarrow \dagger AEF$	BG	BG
(2)	$\dagger BG$	CH	CH
(3)	$\dagger CH$	\emptyset	ADF
(4)	$\dagger ADF$	BG	BG

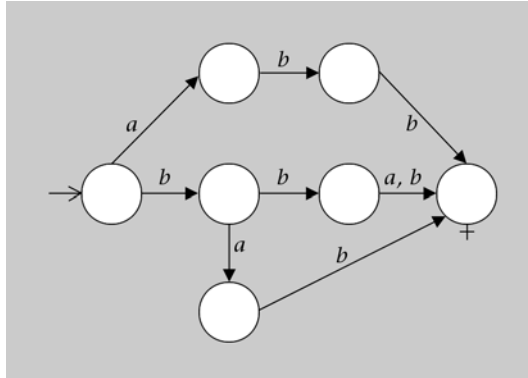
Gráficamente, obtenemos el siguiente autómata, que ya es mínimo:



d) Para resolver este problema comenzaremos considerando el complementario del lenguaje solicitado. Este complementario es “palabras con alguna subpalabra de tres símbolos que contiene menos de dos a (cero o una)”. Así, tenemos: $L^c = \Sigma^*\{bbb, abb, bab, bba\}\Sigma^*$.

- Construcción de un autómata que reconoce $\{bbb, abb, bab, bba\}$.

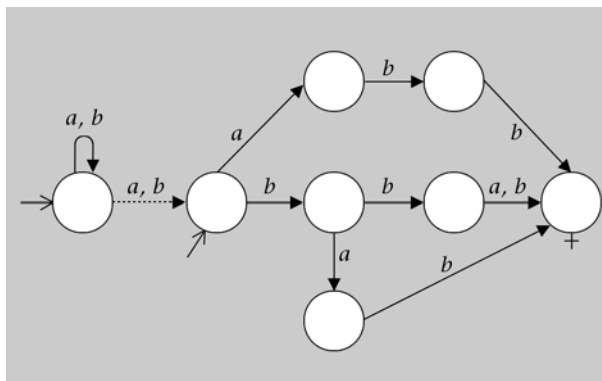
Este autómata será el siguiente:



Evidentemente, hay muchas otras posibilidades, pero ésta tiene un número razonable de estados.

- Construcción de un el autómata que reconoce $\Sigma^*\{bbb, abb, bab, bba\}$.

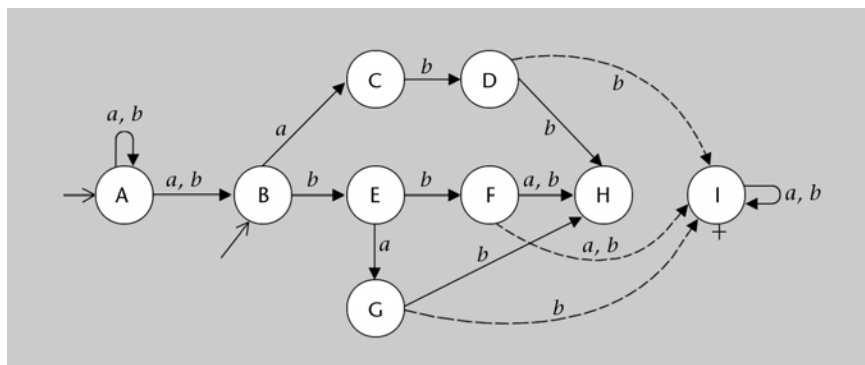
Sólo es necesario concatenar un autómata que acepte Σ^* con el anterior:



Fijaos en que se mantienen los estados iniciales de los dos autómatas porque el primero acepta la palabra λ .

- Construcción del autómata que reconoce $\Sigma^*\{bbb, abb, bab, bba\}\Sigma^*$.

Concatenamos el autómata anterior con el que reconoce Σ^* :



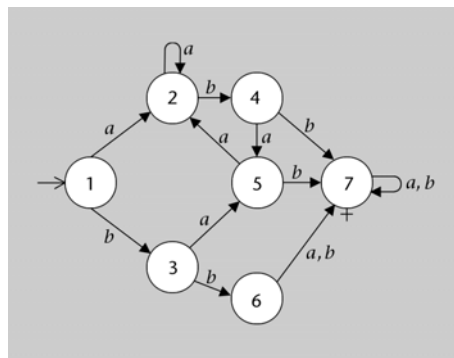
Fijaos en que el primer autómata no acepta la palabra λ y que, por tanto, los únicos estados iniciales son los suyos.

- Determinizar y complementar.

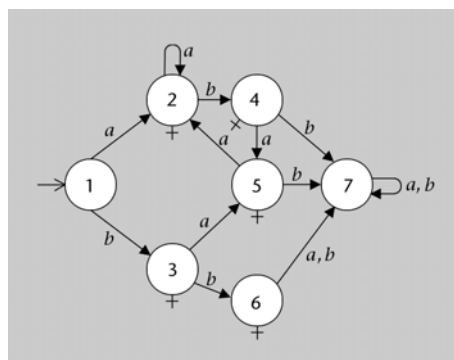
La tabla de transiciones del autómata determinista que se ha obtenido del anterior es:

	δ	a	b
(1)	$\rightarrow AB$	ABC	ABE
(2)	ABC	ABC	ABDE
(3)	ABE	ABCG	ABEF
(4)	ABDE	ABCG	ABEFHI
(5)	ABCG	ABC	ABDEHI
(6)	ABEF	ABCGHI	ABEFHI
(7)	$\dagger ABEFHI$	ABCGHI	ABEFHI
(8)	$\dagger ABDEHI$	ABCGI	ABEFHI
(9)	$\dagger ABCGHI$	ABCI	ABDEHI
(10)	$\dagger ABCGI$	ABCI	ABDEHI
(11)	$\dagger ABCI$	ABCI	ABDEI
(12)	$\dagger ABDEI$	ABCGI	ABEFHI

Fijaos en que los seis estados finales son equivalentes y que, en consecuencia, se pueden reducir a uno solo, tal como se muestra en la figura siguiente:



Intercambiando estados aceptadores y no-aceptadores queda el autómata siguiente:



- Minimización del autómata resultante.

\equiv_0 : $\{\{1, 2, 3, 4, 5, 6\}, \{7\}\}$.

\equiv_1 : $\{\{1, 2, 3\}, \{4, 5\}, \{6\}, \{7\}\}$.

\equiv_2 : $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}\}$.

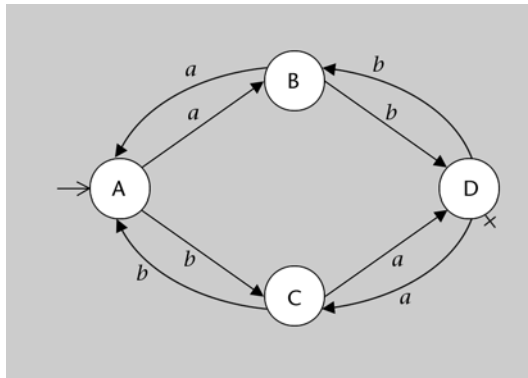
El autómata que se ha encontrado ya era mínimo.

e) Este problema también se resolverá considerando, en primer lugar, el complementario del lenguaje que debe ser reconocido. Este complementario es el siguiente: “palabras que

tienen un prefijo con un número impar de símbolos a y un número impar de símbolos b ". Así, $L^c = L_1 \Sigma^*$, donde $L_1 = \{w \mid |w|_a = 2n + 1 \wedge |w|_b = 2p + 1, n \geq 0, p \geq 0\}$.

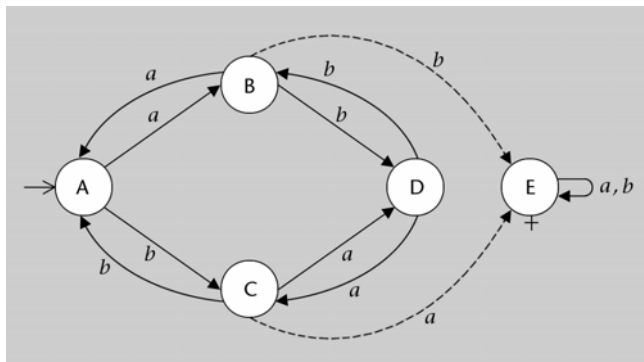
- Construcción de un autómata que reconozca L_1 .

El autómata que reconoce L_1 será el siguiente:



- Construcción de un autómata para $L_1 \Sigma^*$.

Sólo es necesario concatenar el autómata anterior con el que reconoce Σ^* :

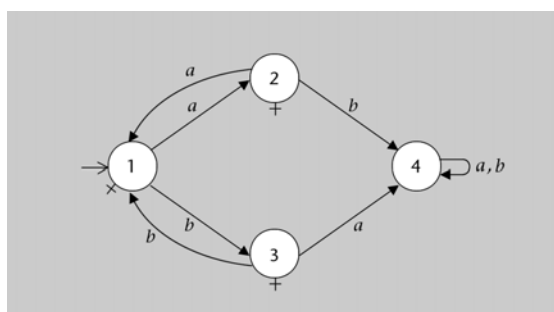


- Determinizar y complementar.

La tabla de transición del autómata determinista que se ha obtenido a partir del anterior es:

	δ	a	b
(1)	$\rightarrow A$	B	C
(2)	B	A	DE
(3)	C	DE	A
(4)	$\dagger DE$	CE	BE
(5)	$\dagger CE$	DE	AE
(6)	$\dagger BE$	AE	DE
(7)	$\dagger AE$	BE	CE

Los cuatro estados finales son equivalentes y, por tanto, se podrán reducir a un único estado. Una vez complementado, queda el gráfico siguiente, que ya es mínimo:



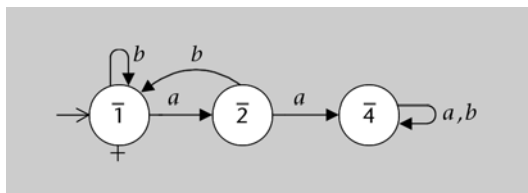
5. El problema se dividirá en dos partes:

- a) En primer lugar, se demostrará que los dos autómatas reconocen el mismo lenguaje minimizándolos y comprobando que los autómatas mínimos resultantes son el mismo.
- b) En segundo lugar, se demostrará que el autómata que reconoce el lenguaje asociado a la expresión regular es (en su versión mínima) el mismo que el que se ha encontrado en la primera parte.

a) Comencemos por la primera parte:

- Minimización del primer autómata.

- Las clases inducidas por \equiv_0 son: $\{1, 3\}$, $\{2, 4\}$.
- Las clases inducidas por \equiv_1 son: $\{1, 3\}$, $\{2\}$, $\{4\}$.
- Las clases inducidas por \equiv_2 son las mismas que las inducidas por \equiv_1 . Por tanto, el autómata mínimo es:

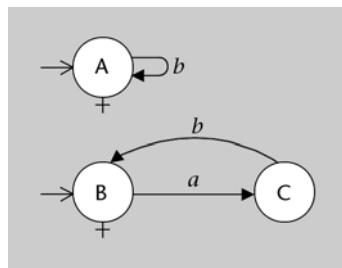


Si exceptuamos los nombres de los estados, este autómata coincide con el segundo. En consecuencia, ya queda demostrado que el lenguaje reconocido es exactamente el mismo, sin necesidad de minimizar el segundo autómata.

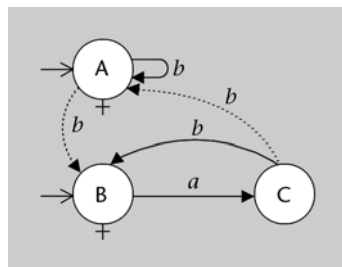
b) Ahora demostraremos la segunda parte:

- Obtención del autómata que reconoce el lenguaje descrito por $(b^* + (ab)^*)^+$.

Un autómata que reconoce $b^* + (ab)^*$ es:



Aplicando el método de construcción del autómata que reconoce el cierre positivo, se obtiene el siguiente autómata:



La tabla de transiciones resultante de la determinización de este autómata se encuentra a continuación y se puede observar que, si exceptuamos los nombres de los estados, este autómata coincide con el autómata mínimo que se ha encontrado en el apartado anterior:

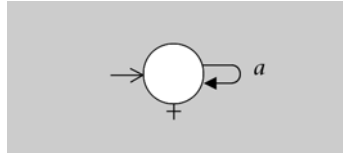
	δ	a	b
(1)	$\rightarrow AB$	C	AB
(2)	C	\emptyset	AB
(3)	\emptyset	\emptyset	\emptyset

6.

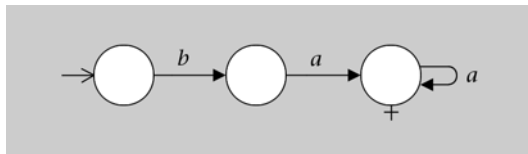
a) La expresión regular dada ($a^*ba^*ab^*$) se puede reescribir $a^*ba^+b^*$, de manera que un NFA que reconozca el lenguaje descrito se puede obtener por concatenación de los NFA que reconocen los lenguajes descritos por a^* , ba^+ y b^* .

Estos NFA son los siguientes:

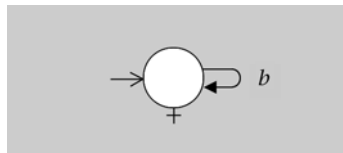
- NFA para a^* :



- NFA para ba^+ :

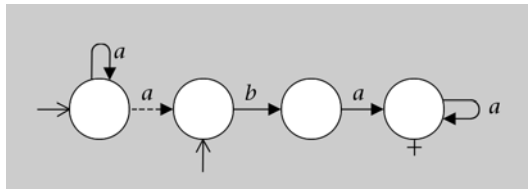


- NFA para b^* :

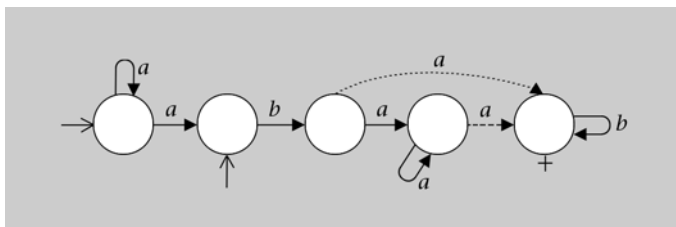


Una vez tenemos los NFA, comenzamos a concatenarlos:

- Concatenación a^* y ba^+ :

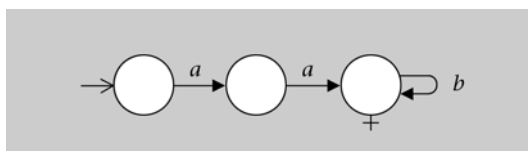


- Concatenación a^* , ba^+ y b^* :

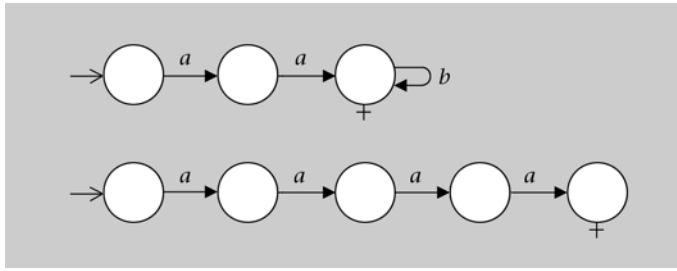


b) Al igual que en el caso anterior, se procederá encontrando NFA para las subexpresiones más simples; y después, aprovechando las operaciones conocidas sobre autómatas, se construirá el NFA que reconoce la totalidad de la expresión regular dada:

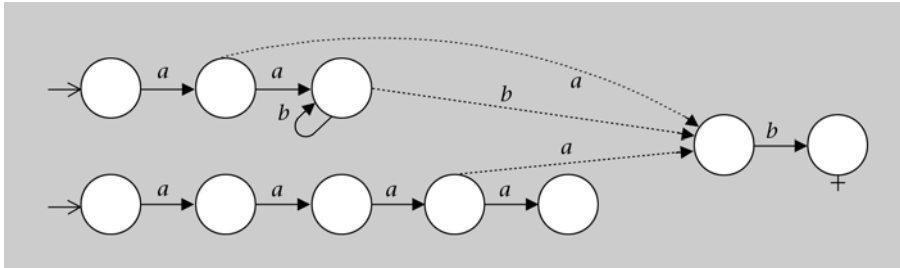
- NFA para aab^* :



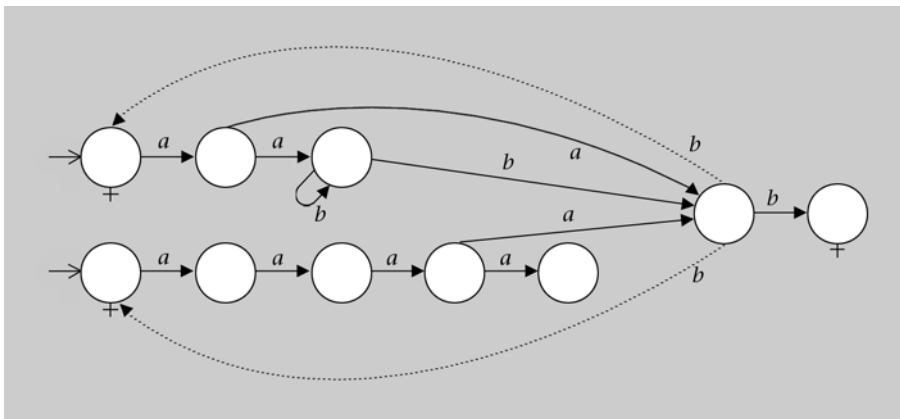
- NFA para $aab^* + a^4$:



- NFA para $(aab^* + a^4)b$:



- NFA para $((aab^* + a^4)b)^*$:



7.

a) El sistema de ecuaciones asociado al autómata dado es:

$$\begin{cases} L_1 = aL_2 + bL_3. \\ L_2 = (a + b)L_4. \\ L_3 = (a + b)L_4. \\ L_4 = (a + b)L_2 + \lambda. \end{cases}$$

Para resolver bien el sistema y llegar a la solución deseada (L_1), se debe ser cuidadoso con el orden de las sustituciones:

- $L_4 = (a + b)L_2 + \lambda = (a + b)(a + b)L_4 + \lambda = [(a + b)(a + b)]^* \lambda$.
- $L_2 = (a + b)L_4 = (a + b)[(a + b)(a + b)]^* \lambda$.
- $L_3 = (a + b)L_4 = (a + b)[(a + b)(a + b)]^* \lambda = L_2$.
- $L_1 = aL_2 + bL_3 = aL_2 + bL_2 = (a + b)L_2 = (a + b)(a + b)[(a + b)(a + b)]^* \lambda = [(a + b)(a + b)]^* \lambda = (\Sigma^2)^+ \lambda$.

b) El sistema de ecuaciones asociado al autómata es:

$$\begin{cases} L_1 = bL_1 + aL_2. \\ L_2 = (a + b)L_3. \\ L_4 = (a + b)L_2 + \lambda. \end{cases}$$

La solución es la siguiente:

- $L_2 = (a+b)L_3 = (a+b)[(a+b)L_2 + \lambda] = (a+b)^2L_2 + (a+b) = ((a+b)^2)^*(a+b)$.
- $L_1 = bL_1 + aL_2 = bL_1 + a((a+b)^2)^*(a+b) = b^*((a+b)^2)^*(a+b) = b^*a(\Sigma^2)^*\Sigma = b^*a\Sigma^{2n+1}, n \geq 0$.

c) El sistema de ecuaciones asociado al autómata es:

$$\begin{cases} L_1 = aL_2 + bL_4. \\ L_2 = aL_3 + bL_1 + \lambda. \\ L_3 = aL_2 + bL_4 + \lambda. \\ L_4 = (a+b)L_4. \end{cases}$$

La solución será la siguiente:

- $L_4 = (a+b)\emptyset = \emptyset$ (y no $L_4 = (a+b)^*$).
- $L_3 = aL_2 + bL_4 + \lambda = aL_2 + b\emptyset + \lambda = aL_2 + \lambda$.
- $L_2 = aL_3 + bL_1 + \lambda = a(aL_2 + \lambda) + bL_1 + \lambda = aaL_2 + a + bL_1 + \lambda = (a^2)^*(a + bL_1 + \lambda)$.
- $L_1 = aL_2 + bL_4 = a(aa)^*(a + bL_1 + \lambda) + b\emptyset = a(a^2)^*(a + bL_1 + \lambda) = a(a^2)^*a + a(a^2)^*bL_1 + a(a^2)^* = [a(a^2)^*b]^*[a(a^2)^*a + a(a^2)^*] = [a(a^2)^*b]^*a^+$.

Fijaos en que $a(a^2)^*a + a(a^2)^* = a^+$. Una posible demostración sería la siguiente:

- $a(a^2)^*a = aa^{2n}a, n \geq 0$, o, lo que es lo mismo, $a^{2(n+1)}, n \geq 0$ (secuencias con un número par, no cero, de a).
- $a(a^2)^* = aa^{2n}, n \geq 0$, o, lo que es lo mismo, $a^{2n+1}, n \geq 0$ (secuencias con un número impar de a).

8.

a)

- Para cada longitud N consideramos una palabra de L . En este caso, la palabra considerada será a^Nba^{N+1} .
- Para cada palabra consideraremos todas las posibles descomposiciones:

$$x = a^p, y = a^q, z = a^{N-p-q}ba^{N+1}, \text{ con } p + q \leq N \text{ y } q \geq 1.$$

Así, cada palabra a^Nba^{N+1} se expresa como $a^pa^qa^{N-p-q}ba^{N+1}$.

- Hallamos una constante que haga que $xy^iz \notin L$. En este caso se puede utilizar cualquier valor que no sea 1. Por ejemplo, con $i = 0$ tendríamos que:

$$xy^iz = xy^0z = a^pa^{N-p-q}ba^{N+1} = a^{N-q}ba^{N+1}$$

Dado que $N - q \neq N + 1$, pasa que $a^{N-q}ba^{N+1} \notin L$, con lo cual queda demostrada la no-regularidad de L .

b)

- Para cada longitud N consideramos una palabra de L . Consideraremos a^Nba^N .
- Para cada palabra consideramos todas las posibles descomposiciones:

$$x = a^p, y = a^q, z = a^{N-p-q}ba^N, \text{ con } p + q \leq N \text{ y } q \geq 1.$$

Así, cada palabra a^Nba^N , se expresa como $a^pa^qa^{N-p-q}ba^N$.

- Encontramos una constante i que haga que $xy^iz \notin L$.

En este caso, se puede utilizar cualquier valor que no sea 1. Por ejemplo, con $i = 2$ tendríamos que:

$$xy^iz = xy^2z = a^pa^2qa^{N-p-q}ba^N = a^{N+q}ba^N.$$

Dado que $N + q \neq N + 1$, pasa que $a^{N+q}ba^N$ no es una palabra palíndroma, con lo que queda demostrada la no-regularidad del lenguaje.

c) En el segundo ejemplo de aplicación del lema del bombeo se ha demostrado que el lenguaje $\{a^n \mid n \text{ es un número primo}\}$ no es un lenguaje regular. Está claro que este lenguaje es el

Recordad que...

... "todas las posibles descomposiciones" hace referencia a todas aquéllas de la forma xyz donde $|xy| \leq N$ y $|y| \geq 1$.

complementario del que ahora se da para demostrar su no-regularidad. Pues bien, cuando un lenguaje no es regular, su complementario tampoco lo es.

La demostración de esta afirmación es una sencilla aplicación de la ley del contrarrecíproco: L es un lenguaje regular $\rightarrow L^c$ es un lenguaje regular. L^c no es un lenguaje regular $\rightarrow L$ no es un lenguaje regular.

9.

a) La afirmación es cierta. Sólo hay que considerar como ejemplos los lenguajes que ya se ha demostrado que no son regulares. Todos aquellos lenguajes son subconjuntos de Σ^* . Así pues, Σ^* , que es un lenguaje regular, tiene subconjuntos (como $L = \{w \mid w = w^R\}$) que no son regulares.

Otros lenguajes regulares, como por ejemplo $\{a\}^* \{b\}^*$, tienen subconjuntos que no lo son, como $\{a\}^n \{b\}^n, n \geq 0$.

b) La afirmación es falsa. La estrella de Kleene de un lenguaje regular es siempre otro lenguaje regular, porque dado un lenguaje regular, siempre se puede encontrar un autómata que lo reconoce y, a partir de éste, un autómata que reconoce su estrella de Kleene.

c) Esta afirmación es cierta. Es suficiente con que los lenguajes sean disyuntos. De esta manera, su intersección será \emptyset y, como ya sabemos, \emptyset es un lenguaje regular. Un par de lenguajes no regulares que son disyuntos son $L_1 = \{w \mid |w| \text{ es un número primo}\}$ y $L_2 = \{w \mid |w| \text{ no es un número primo}\}$.

La ley del contrarrecíproco
 $A \rightarrow B \dashv\vdash \neg B \rightarrow \neg A$.

Recordad que...

... si L es regular, L^c también lo es, porque siempre se puede encontrar un autómata que reconoce L^c a partir del autómata que reconoce L .

Glosario

autómata

Máquina o algoritmo (o su modelo formal) que reconoce un lenguaje (respondiendo a la cuestión de si una palabra arbitraria pertenece a él o no).

DFA

Acrónimo de *Deterministic Finite Automaton* ('autómata finito determinista').

estado

Configuración en la que se puede encontrar un autómata.

expresión (regular)

Expresión que describe un lenguaje regular utilizando sólo los operadores + (unión), \cdot (concatenación) y * (estrella de Kleene).

lema de Arden

Lema de D.N. Arden que afirma que la solución de la ecuación $L = AL + B$ es A^*B .

lema del bombeo

Resultado obtenido por Y. Bar-Hillel, M. Perles y E. Shamir que pone de manifiesto la naturaleza repetitiva de las palabras de los lenguajes regulares. Se trata de una condición necesaria de regularidad. Cuando se considera su contrarrecíproco, se convierte en una condición suficiente de no-regularidad que se puede utilizar para demostrar que determinados lenguajes no son regulares.

NFA

Acrónimo de *Nondeterministic Finite Automaton* ('autómata finito indeterminista').

regular (lenguaje)

Lenguaje que puede ser reconocido por algún DFA.

transición

En un autómata, cambio de estado.

Bibliografía

Bibliografía complementaria

Brookshear, J.G. (1993). *Teoría de la computación. Lenguajes formales, autómatas y complejidad*. Addison-Wesley Iberoamericana.

Hopcroft, J.E.; Ullman, J.D. (2002). *Introducción a la teoría de autómatas, lenguajes y computación*. México: CECSA. (2ª edición)

Isasi, P.; Martínez, P.; Borrajo, D. (1997). *Lenguajes, gramáticas y autómatas. Un enfoque práctico*. Madrid: Addison-Wesley Iberoamericana.