TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES

(TALF)

GUÍA BÁSICA PARA EL DISEÑO DE AUTÓMATAS FINITOS DETERMINISTAS (AFDs)

Ana Pradera, Juan Manuel Serrano
Sergio Saugar, Mónica Robledo
César Alfaro, Javier Gómez
María Teresa González de Lena, Gema Gutiérrez

Noviembre de 2024

Contenido

5 AFDs DE UN LENGUAJE Y DE SU COMPLEMENTARIO

- $L = \{w \in \{a, b\}^* \ : \ w \text{ no contiene } ba\}$

6 AFDs PARA ALGUNOS LENGUAJES DISTINGUIDOS

- $L = \emptyset = \{\}$
- $L = \{\lambda\}$
- $L = \Sigma = \{a, b\}$
- $L = \Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, ...\}$
- $L = \Sigma^+ = \Sigma^* - \{\lambda\} = \{a, b, aa, ab, ...\}$

7 AFDs PARA LENGUAJES CON CARACTERÍSTICAS TÍPICAS

- Longitud de las palabras
  - $L = \{w \in \Sigma^* : |w| \text{ MOD } 2 = 0\}$
  - $L = \{w \in \Sigma^* : |w| \text{ MOD } 2 = 1\}$
  - $L = \{w \in \Sigma^* : |w| \text{ MOD } m = 0\}$
  - $L = \{w \in \Sigma^* : |w| \text{ MOD } m \neq 0\}$
  - $L = \{w \in \Sigma^* : |w| \leq m\}$
  - $L = \{w \in \Sigma^* : |w| = m\}$
  - $L = \{w \in \Sigma^* : |w| > m\}$
  - $L = \{w \in \Sigma^* : |w| \neq m\}$

- Apariciones de un cierto símbolo
  - $L = \{w \in \Sigma^* : n_a(w) \text{ MOD } 2 = 0\}$
  - $L = \{w \in \Sigma^* : n_a(w) \leq m\}$
  - $L = \{w \in \Sigma^* : n_a(w) \text{ MOD } m = 0\}$
  - $L = \{w \in \Sigma^* : n_a(w) \neq m\}$
- Comienzo de las palabras
  - $L = \{abaw \ : \ w \in \Sigma^*\}$
  - $L = \Sigma^* - \{aaw \ : \ w \in \Sigma^*\}$
- Final de las palabras
  - $L = \{waa \ : \ w \in \Sigma^*\}$
  - $L = \Sigma^* - \{waba \ : \ w \in \Sigma^*\}$
- Contenido de las palabras
  - $L = \{w_1 aaw_2 \ : \ w_1, w_2 \in \Sigma^*\}$
  - $L = \Sigma^* - \{w_1 aabw_2 \ : \ w_1, w_2 \in \Sigma^*\}$

8 AFDs PARA LENGUAJES COMPUESTOS (AFD PRODUCTO)

- Construcción del AFD producto
- $L = \{w \in \{a, b\}^* : w \text{ contiene } ba \textbf{ y } \text{empieza por } b\}$
- $L = \{w \in \{a, b\}^* : w \text{ contiene } ba \textbf{ o } \text{empieza por } b\}$
- $L = \{w \in \{a, b\}^* : w \text{ contiene } ba \textbf{ pero no } \text{empieza por } b\}$

- $L = \{w \in \{a,b\}^* : \textbf{si } w \text{ contiene } ba, \textbf{ entonces } \text{empieza por } b\}$

9 EJEMPLOS ADICIONALES

- $L = \{w \in \{a,b\}^+ : |w| \text{ MOD } 3 = 0\}$
- $L = \{w \in \{a,b\}^* : n_{ab}(w) \text{ MOD } 2 = 0\}$
- $L = \{w \in \{a,b\}^+ \ : \ w \text{ no contiene dos b's juntas aisladas}\}$
- $L = \{w \in \{a,b\}^* : n_a(w) \text{ MOD } 2 = 0 \textbf{ y } n_b(w) \text{ MOD } 3 = 0\}$
- $L = \{w \in \{a,b\}^* : n_a(w) \text{ MOD } 2 = 0 \textbf{ o } n_b(w) \text{ MOD } 3 = 0\}$
- $L = \{w \in \{a,b,c\}^* : n_b(w) \text{ MOD } 2 = 1, w \text{ no acaba en } ac\}$
- $L = \{a^n x : n > 0, x \in \{a,b,c\}^*, x \text{ no contiene } ccc\}$
- $L = \{awa \ : \ w \in \{a,b,c\}^*\}$
- $L = \{w \in \{a,b\}^* \ : \ w \text{ empieza por } b \text{ y si ... entonces...}\}$

10 BIBLIOGRAFÍA

11 DISTRIBUCIÓN

INTRODUCCIÓN

- El objetivo de estas notas es proporcionar, por medio de ejemplos, algunas pautas básicas para el diseño de Autómatas Finitos Deterministas (AFDs).
- Dado un lenguaje regular L, el problema consiste en construir un AFD A que reconozca *exactamente* ese lenguaje (es decir, acepte *todas* las palabras de L y rechace *cualquier* palabra que no pertenezca a L), de forma que se tenga $L(A) = L$, donde $L(A)$ es el lenguaje reconocido por A.
- El problema anterior no tiene una solución única puesto que AFDs distintos pueden reconocer un mismo lenguaje (en cuyo caso se dice que los autómatas son equivalentes). Una vez diseñado el AFD, se puede comprobar si la solución propuesta es óptima aplicando el conocido como *algoritmo de minimización* (ver bibliografía).

- El material que se presenta a continuación es fruto de la impartición de la materia *"Teoría de Autómatas y Lenguajes Formales"* en distintos grados universitarios del ámbito de la Informática.
- Se dirige a estudiantes tales que, además de manejar los conceptos básicos relativos a lenguajes y autómatas, conocen con cierta profundidad las características y el funcionamiento de los *autómatas finitos deterministas*.
- En lo que sigue:
  - Se recuerdan en primer lugar los **conceptos fundamentales** relativos a AFDs.
  - A continuación se detallan, ilustradas con un ejemplo, las **pautas generales para el diseño de AFDs**, así como los principales **errores y mejoras** en su diseño.
  - Los apartados siguientes explican el diseño de AFDs para **lenguajes distinguidos o con características típicas** y para **lenguajes compuestos** a partir de otros.
  - Se termina con algunos **ejemplos adicionales**.

AFDs: CONCEPTOS FUNDAMENTALES

Definición (Autómata Finito Determinista, AFD)

Un Autómata Finito Determinista (AFD) es una quíntupla (Q, Σ, f, q_0, F) donde:

- Q es un conjunto no vacío y *finito* de elementos denominados estados (internos).
- Σ es un alfabeto (conjunto no vacío y finito de símbolos) denominado alfabeto de entrada.
- $f : Q \times \Sigma \to Q$ es una *aplicación* denominada función de transición, que asocia a *todo* par $(q, a) \in Q \times \Sigma$ un y *solo un* estado de Q.
- $q_0 \in Q$ es un estado distinguido denominado estado inicial.
- $F \subseteq Q$ es un subconjunto de estados denominados estados finales.

Representación de AFDs

Los AFDs se pueden representar, de forma equivalente, mediante *tablas* o mediante *grafos (dirigidos)*. En lo que sigue usaremos grafos, en los que:

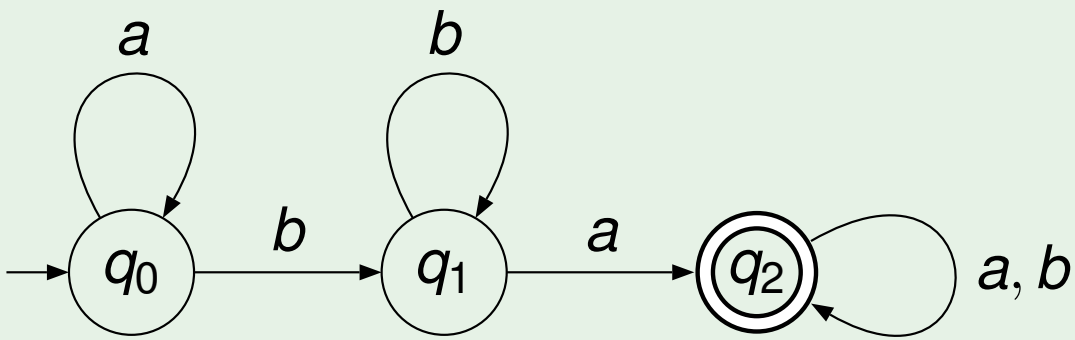
- Cada uno de los estados de Q constituye un nodo del grafo, que se representa mediante el nombre del estado rodeado de un círculo.
- El estado inicial se destaca señalándolo con una flecha incidente, $\rightarrow$, sin origen.
- Los estados finales se marcan rodeándolos mediante un círculo adicional.
- Cada transición $f(q_i, a_k) = q_j$ constituye un arco dirigido desde el nodo q_i hacia el nodo q_j etiquetado con el símbolo a_k.

Funcionamiento de un AFD (descripción informal)

Los AFDs aceptan o rechazan palabras (secuencias finitas de cero o más símbolos) construidas sobre su alfabeto de entrada Σ procesándolas como sigue:

- Inicio: el AFD se encuentra en su estado inicial q_0 y con el cabezal de lectura (que se mueve símbolo a símbolo y de izquierda a derecha) situado en el símbolo más a la izquierda de la palabra que se quiere procesar.
- Mientras haya símbolos por leer:
  - Se lee el símbolo sobre el que está situado el cabezal y se avanza este una posición a la derecha.
  - Se cambia o se mantiene el estado según indique la función de transición f, dependiendo del estado actual y del símbolo leído.
- Fin: el proceso termina una vez leídos todos los símbolos, y:
  - La palabra se acepta si el estado en el que se encuentra el autómata en ese momento es un estado final (pertenece a F).
  - Se rechaza en caso contrario.

Ejemplo



El AFD anterior tiene tres estados: q_0 es el inicial y q_2 es el único final, que además es un estado *trampa* (todas sus transiciones se dirigen a sí mismo). Su alfabeto de entrada es $\Sigma = \{a, b\}$, y:

- Rechaza palabras como λ, a, aa, b o abb: tras procesar cualquiera de las tres primeras el autómata queda en el estado q_0, mientras que tanto b como abb llevan a q_1, y ninguno de estos dos estados es final.
- Acepta palabras como ba, baa, aba o $abbaba$, ya que todas ellas dejan la máquina en el estado q_2, que es final.

PAUTAS GENERALES PARA EL DISEÑO DE AFDs

- En este apartado se proponen algunas pautas generales para el diseño de un AFD que reconozca un lenguaje dado L, es decir, para diseñar un AFD A tal que $L(A) = L$, donde $L(A)$ es el conjunto de todas las palabras aceptadas por A.
- Cada uno de los pasos expuestos se ilustra con su aplicación al diseño de un AFD para el lenguaje

$$L = \{w_1 baw_2 \mid w_1, w_2 \in \{a, b\}^*\},$$

lenguaje formado por todas las palabras construidas mediante a's y b's que contienen la cadena ba.

- El AFD diseñado debe por tanto aceptar cualquier palabra de $\{a, b\}^*$ que contenga ba y rechazar cualquier palabra que no contenga ba.

PASO 1. Análisis del lenguaje de partida

- Para poder diseñar un AFD para L es imprescindible entender correctamente cómo son las palabras que pertenecen al lenguaje.
- Conviene para ello buscar palabras significativas (una de ellas será siempre la palabra vacía λ) tanto pertenecientes como no pertenecientes a L, que servirán además para hacer pruebas posteriormente en el PASO 5.

Ejemplo (AFD para $L = \{w_1 baw_2 \ : \ w_1, w_2 \in \{a, b\}^*\}$ (1/5))

- Algunas palabras $x \in L$: ba (con $w_1 = w_2 = \lambda$), baa, bab, $bababb$ (con $w_1 = \lambda, w_2 \neq \lambda$), aba, bba, $aaba$ (con $w_1 \neq \lambda, w_2 = \lambda$) o $abaa$, $abab$, $abababb$ (con $w_1, w_2 \neq \lambda$).
- Algunas palabras $x \notin L$: cualquier palabra de la forma $a^m b^n$ con $m, n \geq 0$, lo cual incluye a la palabra vacía λ (con $m = n = 0$).

PASO 2. Diseño del conjunto de estados

- Se debe crear un estado para cada posible situación que sea significativa a la hora de decidir (en el momento, o con posterioridad) si la palabra que se está procesando debe o no aceptarse.
- Cada estado tendrá por lo tanto asociada una semántica describiendo las propiedades específicas que se cumplen al llegar a él, propiedades que determinarán su comportamiento tanto a la hora de aceptar o no palabras como a la hora de procesar nuevos símbolos. Por ello, es una buena práctica elegir los nombres de los estados en consonancia con su semántica.
- Como todo AFD tiene un estado inicial, estado en el que se encuentra la máquina al empezar a funcionar, la semántica de uno de los estados creados tiene que ser compatible con la situación “no se ha leído ningún símbolo”.

Ejemplo (AFD para $L = \{w_1 baw_2 \ : \ w_1, w_2 \in \{a, b\}^*\}$ (2/5))

Dado que lo que se pretende es aceptar aquellas palabras que contienen *ba*, es necesario controlar la aparición de esta última cadena, para lo cual hay que distinguir las siguientes situaciones:

1. Que se haya procesado la cadena *ba* (sin importar lo leído antes o después), situación descrita mediante un estado cuyo nombre podría ser $q_{..ba..}$.
2. Que no se haya procesado la cadena *ba* pero que lo último leído sea una *b* (ya que esta podría producir posteriormente la aparición de *ba*). Esta situación requiere de un estado que podría denotarse $q_{..b}$.
3. Que no se dé ninguna de las dos situaciones anteriores (ni se ha procesado *ba* ni lo último leído es una *b*), dando lugar a un estado que podría denotarse $q_{\neq}$.

PASO 3. Determinación del estado inicial y de los estados finales

Estado inicial

- El estado inicial, estado en el que se encuentra el AFD al comenzar a funcionar, será aquel cuya semántica incluya la situación “no se ha leído ningún símbolo”.

Estados finales

- La condición final/no final de los estados se determina en función de su semántica: deberán ser finales todos aquellos, y nada más que ellos, cuya semántica describa palabras que pertenecen al lenguaje L.
- En particular, el estado inicial será final si y solo si el lenguaje L contiene la palabra vacía λ.

Ejemplo (AFD para $L = \{w_1 baw_2 \ : \ w_1, w_2 \in \{a, b\}^*\}$ (3/5))

Teniendo en cuenta la semántica y los nombres propuestos en el punto anterior, es fácil establecer lo siguiente:

- El estado inicial debe ser $q_{\neq}$, puesto que el caso "no se ha leído ningún símbolo" encaja dentro de la descripción "no se ha procesado *ba* y lo último leído no es *b*".
- El único que debe ser final es $q_{..ba..}$, puesto que el autómata solo debe aceptar aquellas palabras que contienen *ba*.
- En particular, el estado inicial $q_{\neq}$ no debe ser final puesto que la palabra vacía no pertenece a L (y debe por tanto rechazarse).

PASO 4. Cálculo de la función de transición

Las transiciones de los estados (una y solo una para cada símbolo del alfabeto) se determinan en función de su semántica.

Ejemplo (AFD para $L = \{w_1 baw_2 \ : \ w_1, w_2 \in \{a, b\}^*\}$ (4/5))

Teniendo en cuenta la semántica propuesta previamente:

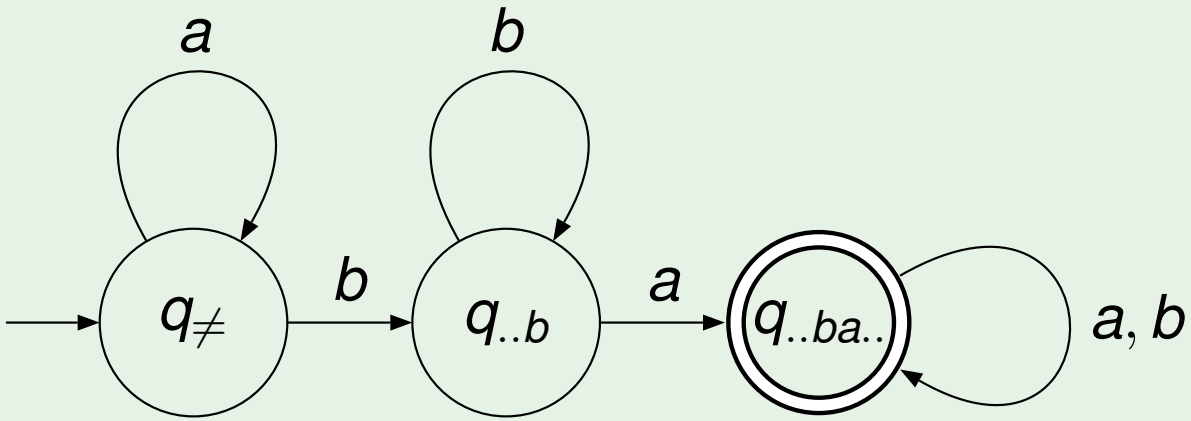
- Desde $q_{\neq}$: la transición a debe ser un bucle (tras leer una a la semántica del estado se mantiene) mientras que la transición b debe ir a $q_{..b}$ (se verifica que lo último leído es una b).
- Desde $q_{..b}$: la lectura de una b no altera la situación, por lo que debe dar lugar a un bucle, mientras que la lectura de una a produce la cadena ba y debe llevar por lo tanto al estado $q_{..ba..}$.
- Las dos transiciones de $q_{..ba..}$ son bucles (una vez que la palabra contiene ba da igual lo que venga detrás, seguirá conteniendo ba). Se trata de un estado trampa.

Ejemplo (AFD para $L = \{w_1 baw_2 \ : \ w_1, w_2 \in \{a, b\}^*\}$ (5/5))

Uniendo todo lo anterior, un posible AFD para el lenguaje

$$L = \{w_1 baw_2 \ : \ w_1, w_2 \in \{a, b\}^*\}$$

es



Observe que se trata del mismo AFD que el propuesto previamente aquí pero con nombres de estados acordes con su semántica.

PASO 5. Comprobaciones básicas

Una vez terminado el diseño de un AFD A con el objetivo de reconocer un cierto lenguaje L, conviene hacer pruebas con palabras *significativas* que pertenecen y no pertenecen a L (ver PASO 1), comprobando que las primeras son aceptadas por el AFD diseñado y las segundas son rechazadas.

Evidentemente, superar estas pruebas con las palabras elegidas no demuestra que se cumpla $L(A) = L$. Para ello habría que hacer una demostración rigurosa, válida para cualquier palabra construida sobre el alfabeto de entrada. Este tipo de demostraciones están fuera del alcance de esta guía.

En el siguiente apartado se enumeran e ilustran los errores que se pueden cometer al diseñar un AFD.

POSIBLES ERRORES Y MEJORAS

- En este apartado se exponen los errores que se pueden cometer a la hora de diseñar un AFD:
  - Lo diseñado no es un AFD.
  - Es un AFD pero es incompleto y/o incorrecto.

  así como las posibles mejoras:
  - Es un AFD completo y correcto pero es mejorable porque no es el mínimo.

- Los errores y mejoras anteriores se ilustran con el lenguaje utilizado en el segundo apartado de esta guía, formado por todas las palabras de $\{a, b\}^*$ que contienen la cadena ba:

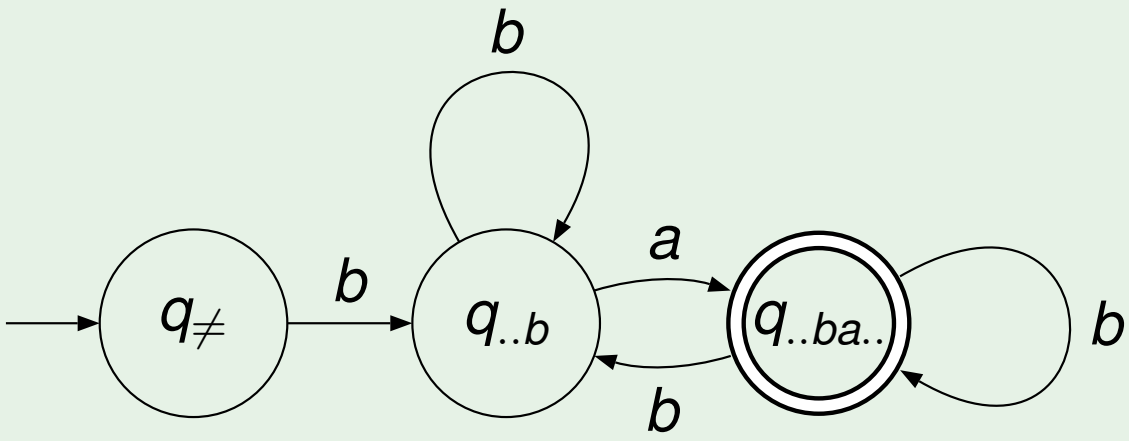
$$L = \{w_1 baw_2 \mid w_1, w_2 \in \{a, b\}^*\}$$

No es un AFD

Lo primero que hay que comprobar al intentar diseñar un AFD es que lo diseñado sea efectivamente un AFD. Para ello, hay que asegurar que se cumplen las siguientes condiciones:

1. El autómata cuenta con *un único estado inicial*.
2. El automáta cuenta con *al menos un estado final* (de otro modo solo reconocerá el lenguaje vacío).
3. *No faltan* transiciones: desde cada estado debe partir una transición para cada uno de los símbolos del alfabeto.
4. *No sobran* transiciones: desde cada estado debe partir *solo* una transición para cada uno de los símbolos del alfabeto.

Ejemplo (AFD para $L = \{w_1 baw_2 \mid w_1, w_2 \in \{a, b\}^*\}$)



Propuesta errónea, lo anterior NO es un AFD:

- Faltan las transiciones "a" tanto desde $q_{\neq}$ como desde $q_{..ba..}$.
- Sobra una de las dos transiciones "b" que parten de $q_{..ba..}$.

Es un AFD pero es incompleto y/o incorrecto

Al diseñar un AFD A para reconocer un cierto lenguaje L, lo que se pretende obtener es $L(A) = L$, equivalente a que se cumpla tanto $L(A) \subseteq L$ como $L \subseteq L(A)$, por lo que se pueden cometer dos tipos de errores (o ambos a la vez):

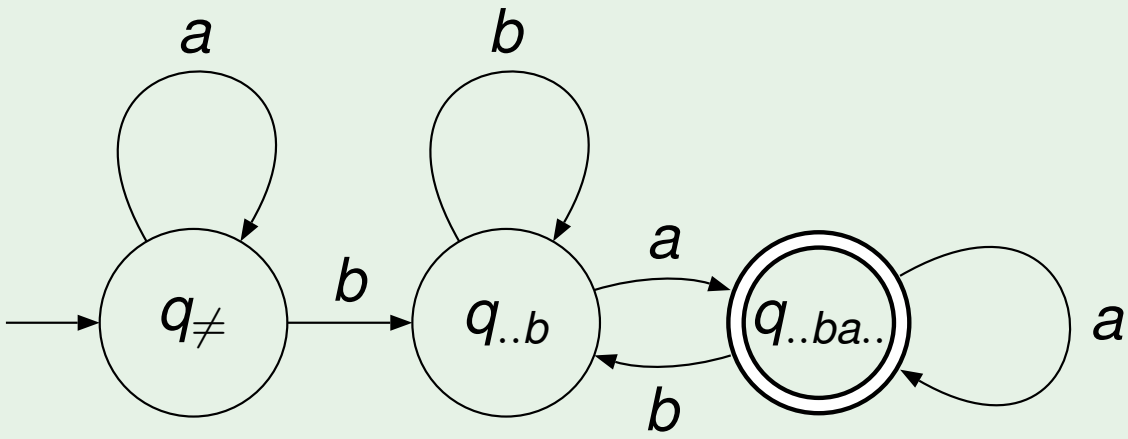
- Que A no reconozca palabras que sí debería reconocer (porque pertenecen a L), es decir, existe alguna $x \in L$ tal que $x \notin L(A)$:

  ¡$L \not\subseteq L(A)$, faltan palabras, A es incompleto!

- Que A reconozca palabras que no debería reconocer (porque no pertenecen a L), es decir, existe alguna $x \in L(A)$ tal que $x \notin L$:

  ¡$L(A) \not\subseteq L$, sobran palabras, A es incorrecto!

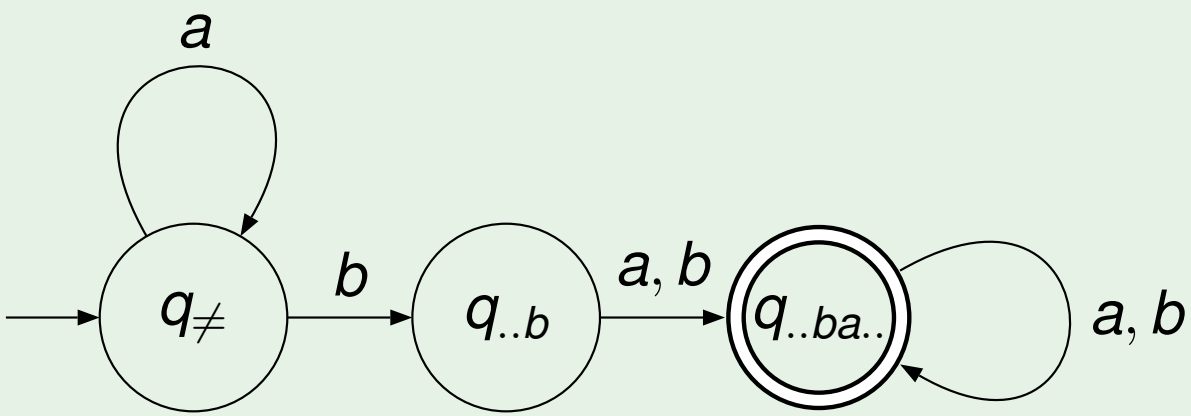
Ejemplo (AFD para $L = \{w_1 baw_2 \mid w_1, w_2 \in \{a, b\}^*\}$)



Propuesta errónea, es un AFD correcto pero INCOMPLETO:

Es un AFD y es correcto (todas las palabras que acepta contienen *ba* y por lo tanto pertenecen a *L*), pero ¡faltan palabras!: por ejemplo, rechaza palabras de *L* como *bab*, *abababab*, etc, y debería aceptarlas (puesto que contienen *ba*). Motivo: la transición *b* de $q_{..ba..}$ no respeta la semántica de los estados (si una palabra contiene *ba*, siempre seguirá conteniendo *ba*, por lo que todas sus transiciones, incluida la de *b*, deberían ser bucles).

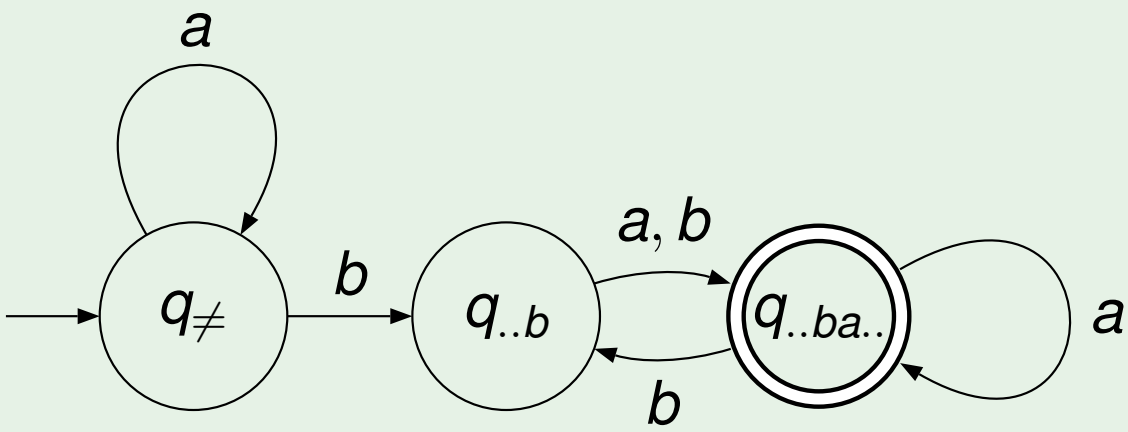
Ejemplo (AFD para $L = \{w_1 baw_2 \mid w_1, w_2 \in \{a, b\}^*\}$)



Propuesta errónea, es un AFD completo pero INCORRECTO:

Es un AFD y es completo (acepta todas las palabras que contienen *ba*), pero ¡sobran palabras!: por ejemplo, acepta palabras como *bb*, *abb*, etc, que debería rechazar (puesto que no contienen *ba*). Motivo: la transición *b* de $q_{..b}$ no respeta la semántica de los estados (si lo último leído es una *b* y no se ha leído *ba*, la lectura de una nueva *b* no altera la situación, por lo que debería ser un bucle).

Ejemplo (AFD para $L = \{w_1 baw_2 \mid w_1, w_2 \in \{a, b\}^*\}$)



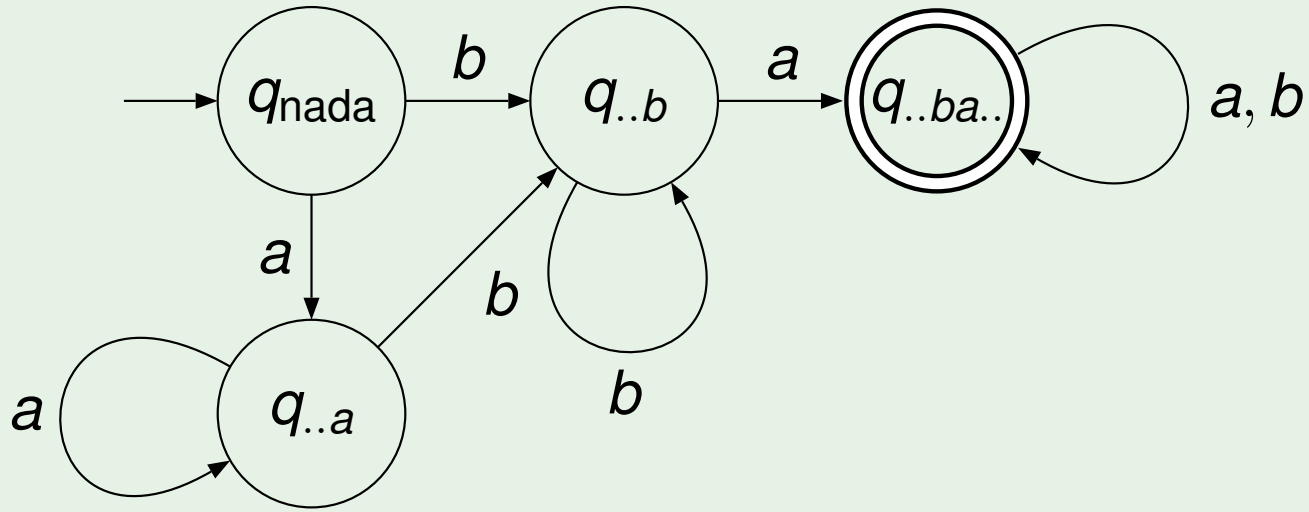
Propuesta errónea, es un AFD INCOMPLETO e INCORRECTO:

Es un AFD pero no es ni completo (rechaza palabras que contienen *ba*, como *bab*, *abab*, etc) ni correcto (acepta palabras que no contienen *ba*, como *bb*, *aabb*, etc).

Es un AFD correcto y completo pero mejorable

- Todo lenguaje regular se puede reconocer mediante varios AFDs distintos, equivalentes entre sí.
- Por razones de eficiencia, es aconsejable trabajar con aquel (único salvo en lo que respecta a los nombres de los estados) que tiene el menor número posible de estados, denominado el AFD mínimo.
- Existe un algoritmo -fuera del alcance de esta guía-, el algoritmo de minimización de AFDs, que no solo determina si un AFD es o no el mínimo, sino que además, en el caso de que no lo sea, facilita el AFD mínimo equivalente al dado.

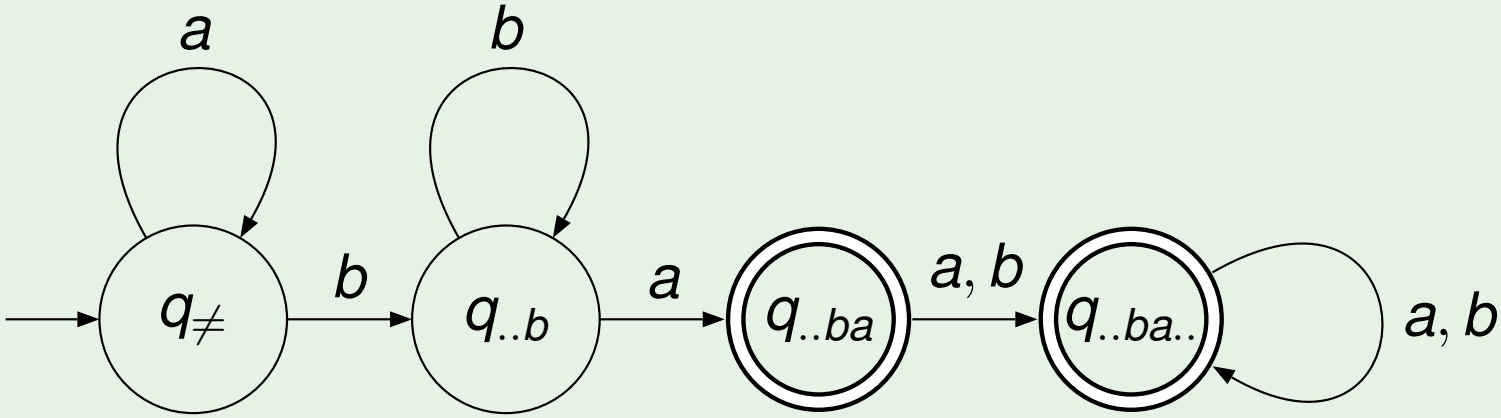
Ejemplo (AFD para $L = \{w_1 baw_2 \mid w_1, w_2 \in \{a, b\}^*\}$)



Mejorable: AFD completo y correcto pero que NO ES MÍNIMO.

Aplicando el algoritmo de minimización se comprueba que los estados q_{nada} y $q_{..a}$ son equivalentes y pueden por lo tanto fundirse en uno, dando lugar al AFD construido previamente aquí, que es el mínimo. En efecto, intuitivamente se ve que para detectar la cadena *ba* es indiferente si hay o no *a*'s delante de la primera *b*.

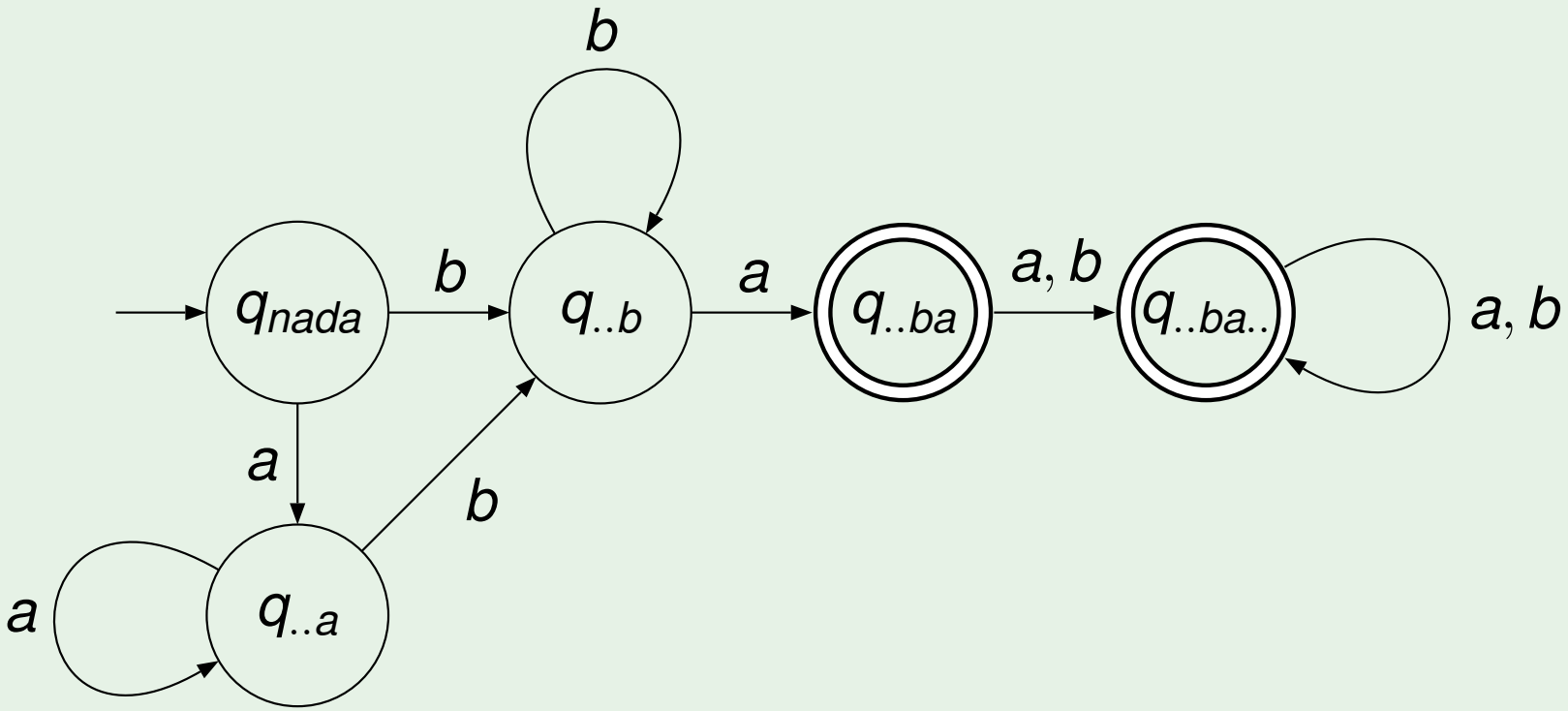
Ejemplo (AFD para $L = \{w_1 baw_2 \mid w_1, w_2 \in \{a, b\}^*\}$)



Mejorable: AFD completo y correcto pero que NO ES MÍNIMO.

Aplicando el algoritmo de minimización se comprueba que los estados $q_{..ba}$ y $q_{..ba..}$ son equivalentes y pueden por lo tanto fundirse en uno, dando lugar al AFD mínimo visto previamente. En efecto, las palabras que acaban en ba (estado $q_{..ba}$) no son más que un caso particular de las que contienen ba ($q_{..ba..}$), que es lo que aquí interesa.

Ejemplo (AFD para $L = \{w_1 baw_2 \mid w_1, w_2 \in \{a, b\}^*\}$)



Mejorable: AFD completo y correcto pero que NO ES MÍNIMO.

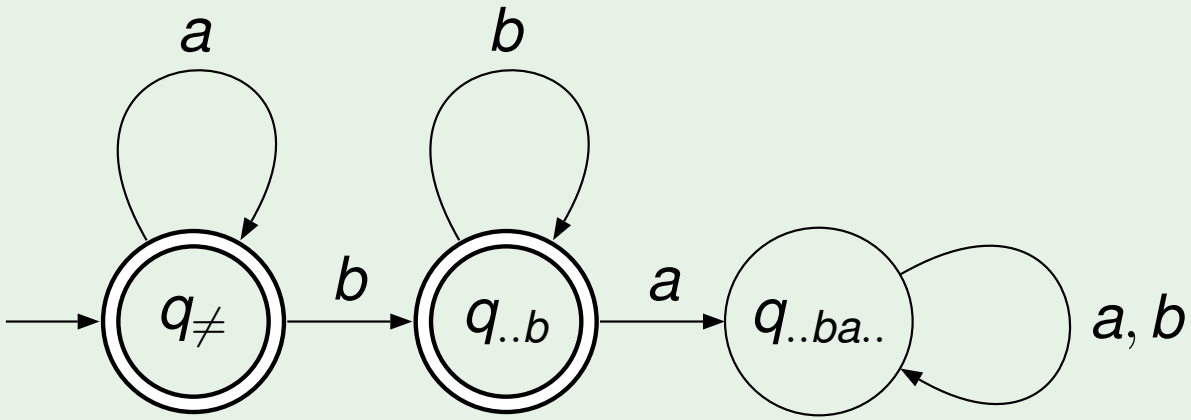
Aplicando el algoritmo de minimización se comprueba que tanto los estados q_{nada} y $q_{..a}$, por un lado, como $q_{..ba}$ y $q_{..ba..}$, por otro lado, son equivalentes entre sí y pueden por lo tanto fundirse, dando lugar al AFD mínimo visto previamente.

AFDs DE UN LENGUAJE Y DE SU COMPLEMENTARIO

- En ocasiones, dado un lenguaje L sobre un alfabeto Σ, puede resultar más sencillo diseñar un AFD para su complementario, $\overline{L} = \Sigma^* - L$, que para L.
- En casos como estos, resulta adecuado diseñar un AFD para $\overline{L}$ y obtener directamente un AFD para L con solo convertir los estados finales en no finales y los no finales en finales.
- OJO: lo anterior **no es válido** para el caso de los Autómatas Finitos No Deterministas (AFNDs) que se estudian más adelante.

Ejemplo

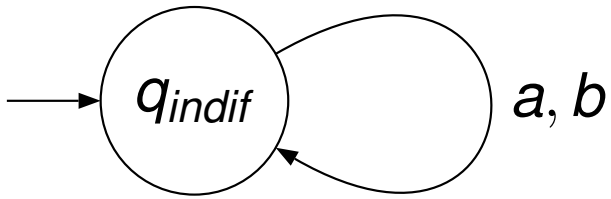
Suponga que necesita diseñar un AFD que reconozca el lenguaje formado por todas las palabras de $\{a, b\}^*$ que *no* contienen la cadena *ba*. Una forma de hacerlo es diseñar un AFD para su complementario, formado por todas las palabras de $\{a, b\}^*$ que sí contienen la cadena *ba*, y luego simplemente convertir sus estados finales en no finales y viceversa. Por lo tanto, partiendo del AFD diseñado previamente, se obtiene fácilmente el siguiente AFD, que reconoce el conjunto de palabras que *no* contienen la cadena *ba* y que podría describirse matemáticamente como $L = \{a^m b^n \ : \ m, n \geq 0\}$:



AFDs PARA ALGUNOS LENGUAJES DISTINGUIDOS, $\Sigma = \{a, b\}$

$L = \emptyset = \{\}$ (lenguaje vacío)

- Todas las palabras deben ser tratadas por igual, por lo que basta un único estado, con semántica "lo leído es indiferente".
- Ese único estado es necesariamente el inicial y no es final puesto que L no contiene λ (el lenguaje vacío no contiene ninguna palabra).
- Sus transiciones no pueden ser más que bucles.



$L = \{\lambda\}$ (lenguaje que solo contiene la palabra vacía)

- Es necesario distinguir la palabra vacía, de longitud 0, de todas las demás, que tienen longitud mayor o igual que 1.
- Se necesitan por lo tanto dos estados:
  - Estado q_0, con semántica "no se ha leído ningún símbolo".
  - Estado q_{1^+}, con semántica "se ha leído al menos un símbolo".
- Estado inicial: q_0.
- Estados finales:
  - q_0 es final puesto que L contiene λ.
  - q_{1^+} no es final puesto que el autómata debe rechazar cualquier palabra con longitud no nula.

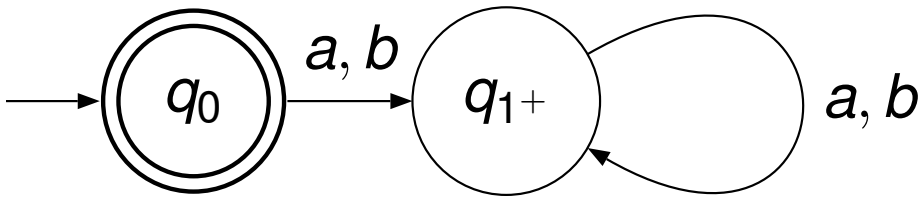
- Transiciones desde q_0:

  Para cualquier símbolo que se lea, resultará cierto que “se ha leído al menos un símbolo”, por lo que hay que transitar al estado q_{1+}.

- Transiciones desde q_{1+}:

  Para cualquier símbolo que se lea se sigue cumpliendo “se ha leído al menos un símbolo”, por lo que todas sus transiciones son bucles (q_{1+} es un estado trampa).

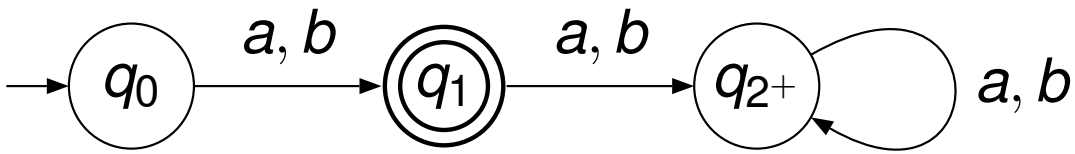
- Se obtiene por lo tanto el siguiente AFD:



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

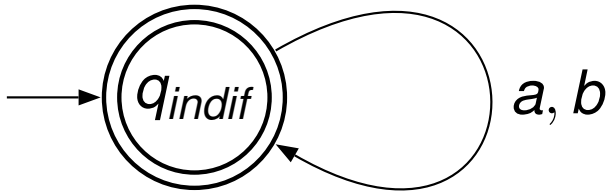
- Es necesario distinguir las palabras de tamaño 1 (a y b), del resto de palabras. Pero entre estas últimas hay una especial, la palabra vacía, puesto que, a diferencia de las demás, si se le añade un símbolo se obtiene una palabra de tamaño 1.
- Se necesitan por lo tanto tres estados:
  - Estado q_0, con semántica "no se ha leído ningún símbolo".
  - Estado q_1, con semántica "se ha leído exactamente un símbolo".
  - Estado q_{2^+}, con semántica "se han leído dos o más símbolos".
- Estado inicial: q_0.
- Estados finales:
  - q_0 no es final puesto que L no contiene λ.
  - q_1 es final puesto que el autómata debe aceptar cualquier palabra con longitud 1.
  - q_{2^+} no es final puesto que el autómata debe rechazar cualquier palabra con longitud distinta de 1.

- Transiciones desde q_0: para cualquier símbolo que se lea hay que ir a q_1, puesto que se ha leído exactamente un símbolo.
- Transiciones desde q_1: para cualquier símbolo que se lea hay que ir a q_{2+}, ya que se cumple "se han leído dos o más símbolos".
- Transiciones desde q_{2+}: para cualquier símbolo que se lea se sigue cumpliendo "se han leído dos o más símbolos", por lo que todas sus transiciones son bucles (q_{2+} es un estado trampa).
- Teniendo en cuenta todo lo anterior, se obtiene el AFD siguiente:



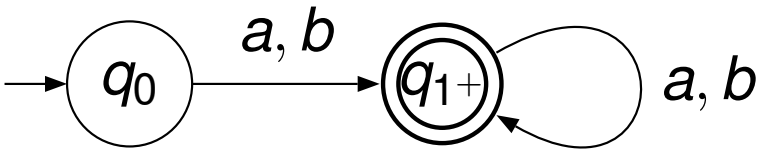
$L = \Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, ...\}$ (lenguaje universal)

- Σ^* es el complementario del lenguaje vacío discutido antes, por lo que basta con convertir en final el estado no final.



$L = \Sigma^+ = \Sigma^* - \{\lambda\} = \{a, b, aa, ab, ...\}$ (lenguaje universal positivo)

- Σ^+ es el complementario del lenguaje $\{\lambda\}$ discutido previamente, por lo que basta con intercambiar estados finales y no finales.



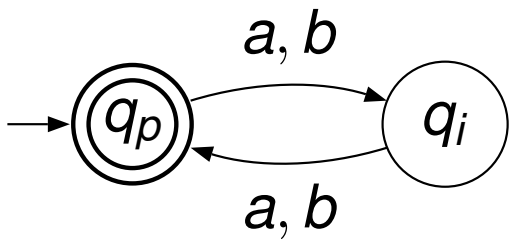
AFDs PARA LENGUAJES CON CARACTERÍSTICAS TÍPICAS, $\Sigma = \{a, b\}$

Longitud de las palabras ($|w|$ = longitud de la palabra w)

$L = \{w \in \Sigma^* : |w| \text{ MOD } 2 = 0\}$ (palabras con longitud par)

- Es necesario distinguir entre dos clases de palabras: las que tienen longitud par (0, 2, 4, ...) y el resto, que tienen longitud impar (1, 3, 5, ...).
- Se necesitan por lo tanto dos estados:
  - Estado q_p, con semántica "palabra con longitud par".
  - Estado q_i, con semántica "palabra con longitud impar"
- Estado inicial: q_p (0 símbolos leídos y 0 es par).

- Estados finales:
  - q_p es final puesto que L contiene λ así como cualquier otra palabra de longitud par.
  - q_i no es final puesto que el autómata debe rechazar cualquier palabra con longitud impar.
- Transiciones desde q_p: para cualquier símbolo que se lea hay que ir a q_i, puesto que cualquier palabra con longitud par, tras la lectura de un símbolo, pasa a tener tamaño impar.
- Transiciones desde q_i: por motivos similares, para cualquier símbolo que se lea en q_i hay que transitar a q_p.

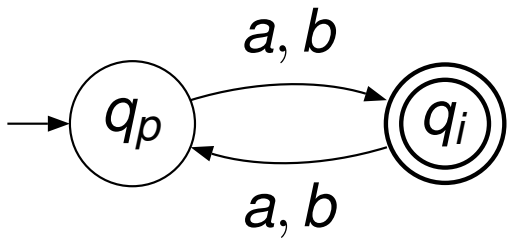


$L = \{w \in \Sigma^* : |w| \text{ MOD } 2 = 1\}$ (palabras con longitud impar)

- La longitud de una palabra solo puede ser par o impar, por lo que este lenguaje es el complementario del anterior:

$$\{w \in \Sigma^* : |w| \text{ MOD } 2 = 1\} = \Sigma^* - \{w \in \Sigma^* : |w| \text{ MOD } 2 = 0\}$$

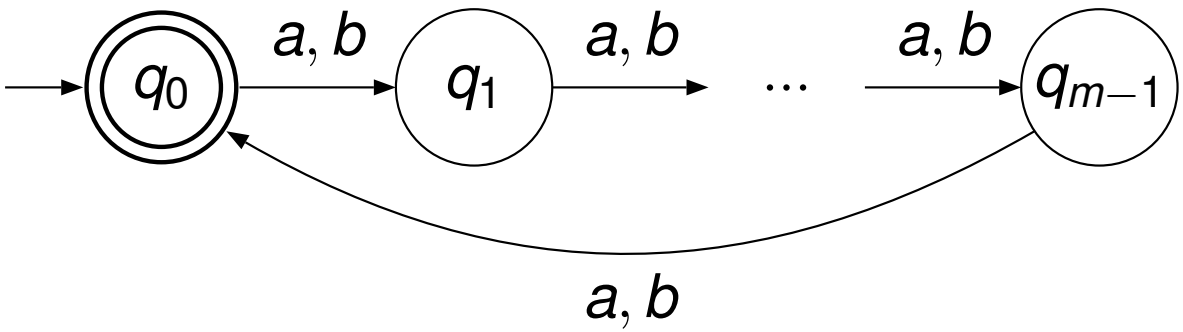
- Para obtener un AFD para el conjunto de palabras con longitud impar basta por lo tanto con intercambiar la condición de final/no final de los estados del AFD diseñado en el punto anterior, obteniéndose como resultado el AFD siguiente:



$L = \{w \in \Sigma^* : |w| \text{ MOD } m = 0\}$
(palabras cuya longitud es un múltiplo de m, $m \in \{2, 3, 4, ...\}$)

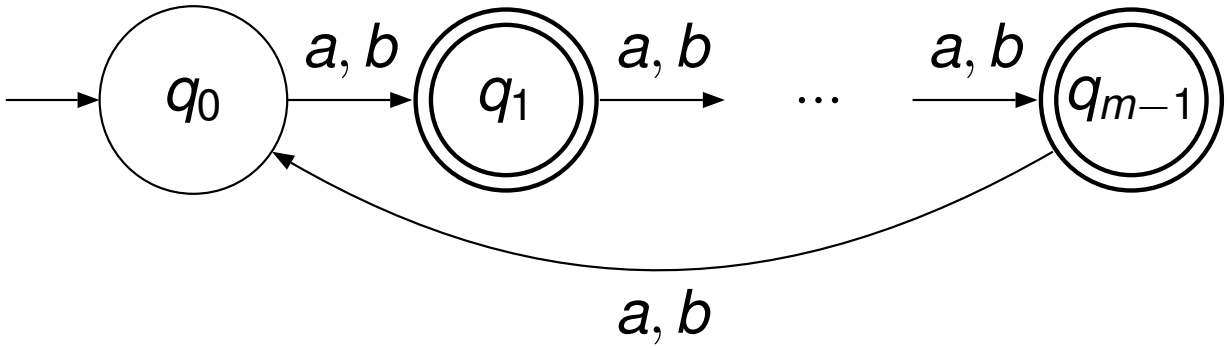
- Con $m = 2$ se recupera el conjunto formado por las palabras con longitud par estudiado previamente.
- Con $m = 3$: conjunto formado por todas las palabras con longitudes 0, 3, 6, 9, etc.
- Con $m = 4$: conjunto formado por todas las palabras con longitudes 0, 4, 8, 12, etc.
- En general, será necesario llevar la cuenta de los símbolos leídos, módulo m, para lo que se necesitarán m estados q_i, $i \in \{0, 1, ..., m-1\}$, con semántica "se han leído i símbolos (módulo m)", de forma que al estado q_i se llegará tras haber leído i símbolos, o $m + i$ símbolos, o $2m + i$ símbolos, o $3m + i$ símbolos, etc.

- El estado inicial será q_0, ya que es aquel en el que se han leído $0, m, 2m, 3m, \ldots$ símbolos, lo cual incluye el caso "no se ha leído ningún símbolo".
- El único estado final será el estado inicial q_0, puesto que es aquel al que llegan todas las palabras con tamaño múltiplo de m.
- Las transiciones avanzan todas ellas de cada estado q_i a su siguiente, q_{i+1}, salvo las del estado q_{m-1}, que vuelven al estado inicial q_0 (se ha leído un número de símbolos múltiplo de m).
- En definitiva, se obtiene el AFD siguiente:



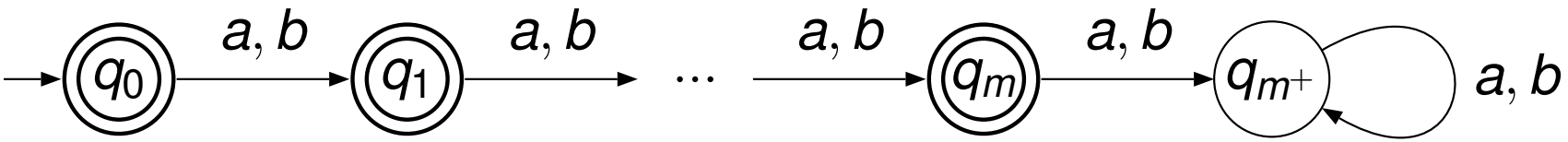
$L = \{w \in \Sigma^* : |w| \text{ MOD } m \neq 0\}$
(palabras cuya longitud no es un múltiplo de m, $m \in \{2, 3, 4, ...\}$)

- Este lenguaje es el complementario del anterior, por lo que basta con invertir los estados finales y no finales del AFD diseñado previamente.



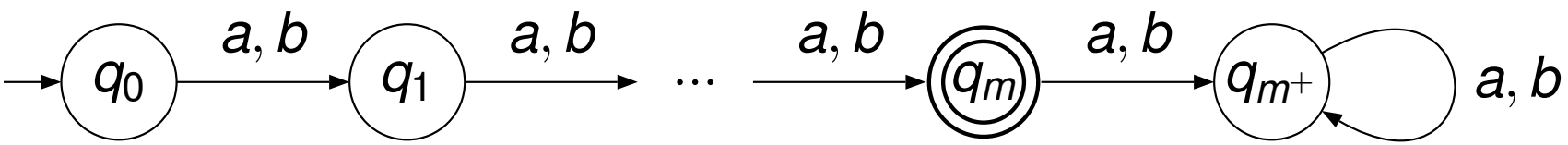
$L = \{w \in \Sigma^* : |w| \leq m\}$ (palabras con longitud menor o igual que $m \in \{0, 1, 2, 3, ...\}$)

- Con $m = 0$ se recupera el lenguaje $L = \{\lambda\}$.
- Con $m = 3$: conjunto formado por todas las palabras con longitudes 0, 1, 2 o 3.
- En general, será necesario llevar la cuenta de los símbolos leídos hasta llegar a m, para lo que se necesitarán $m + 1$ estados q_i, $i \in \{0, 1, ..., m\}$, con semántica "se han leído i símbolos", todos ellos finales, y un estado adicional, trampa y no final, q_{m^+}, "se han leído más de m símbolos".



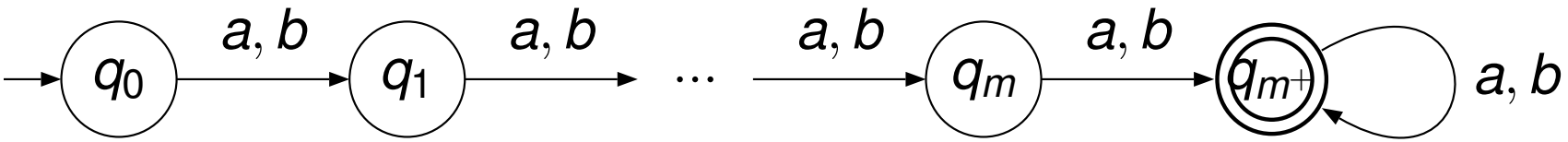
$L = \{w \in \Sigma^* : |w| = m\}$ (palabras con longitud igual a $m \in \{0, 1, 2, 3, ...\}$)

- Mismos estados que para el caso anterior (longitud menor o igual que m):
  - q_i, $i \in \{0, 1, ..., m\}$, "se han leído i símbolos".
  - q_{m^+}, "se han leído más de m símbolos".
- Sin embargo, ahora el único estado final debe ser q_m.



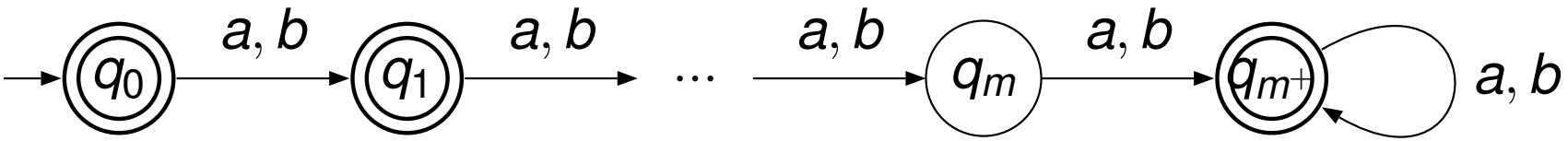
$L = \{w \in \Sigma^* : |w| > m\}$
(palabras con longitud mayor que $m \in \{0, 1, 2, 3, ...\}$)

Este lenguaje es el complementario de $\{w \in \Sigma^* : |w| \leq m\}$:



$L = \{w \in \Sigma^* : |w| \neq m\}$
(palabras con longitud distinta a $m \in \{0, 1, 2, 3, ...\}$)

Este lenguaje es el complementario de $\{w \in \Sigma^* : |w| = m\}$:



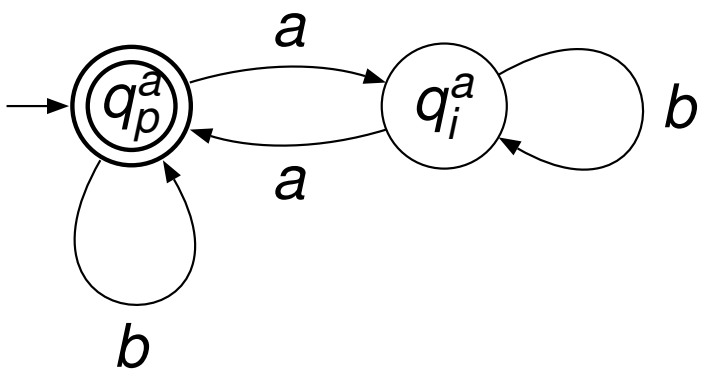
Apariciones de un cierto símbolo ($n_a(w)$ = número de apariciones de $a \in \Sigma$ en la palabra w)

- Estos AFDs se diseñan de forma similar a los descritos previamente en los que intervenía la longitud de las palabras.

- La diferencia fundamental es que ahora la semántica de los estados no viene determinada por la lectura de cualquier símbolo sino solo por la del símbolo cuyas apariciones hay que tener en cuenta.

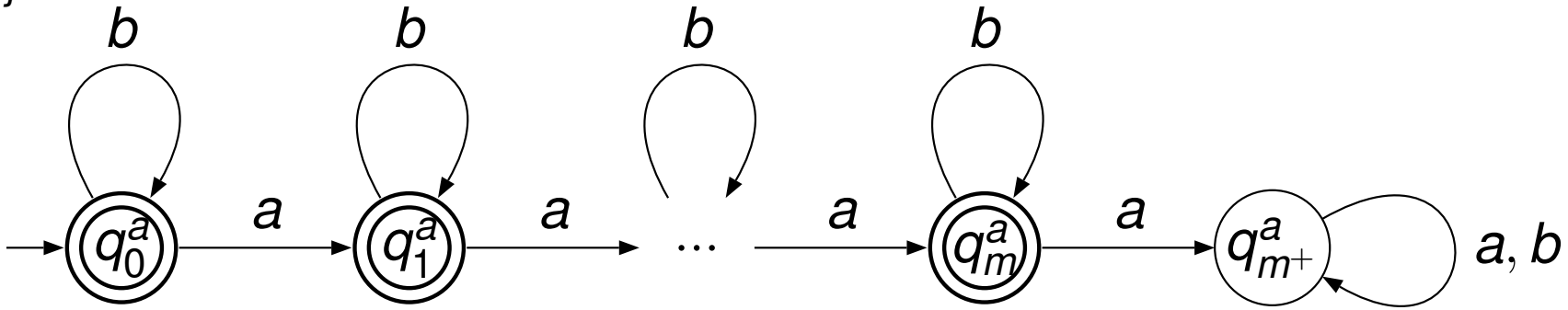
$L = \{w \in \Sigma^* : n_a(w) \text{ MOD } 2 = 0\}$ (palabras con un número par de *a*'s)

- Similar al AFD para palabras con longitud par: solo hay dos tipos de palabras, las que tienen un número par de *a*'s (entre las que se encuentra λ, puesto que tiene cero *a*'s) y las que tienen un número impar de *a*'s. El estado final (e inicial) q_p^a acepta a las primeras, mientras que q_i^a rechaza a las segundas.
- Solo se cambia de estado al leer una *a*, puesto que la lectura de una *b* no afecta.



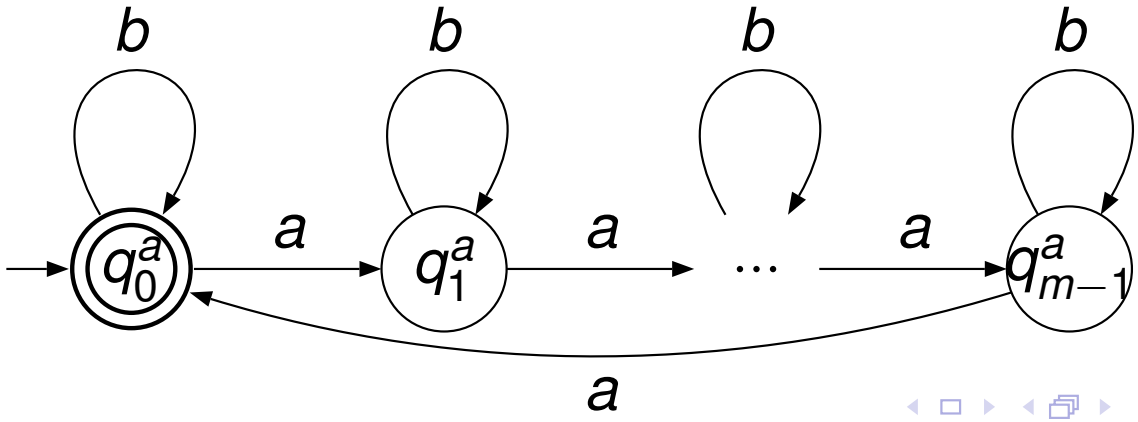
$L = \{w \in \Sigma^* : n_a(w) \leq m\}$, $m \in \{0, 1, 2, 3, ...\}$

$q_j^a =$ "se han leído j símbolos a", con $j \in \{0, ..., m\}$.



$L = \{w \in \Sigma^* : n_a(w) \text{ MOD } m = 0\}$, $m \in \{2, 3, 4, ...\}$

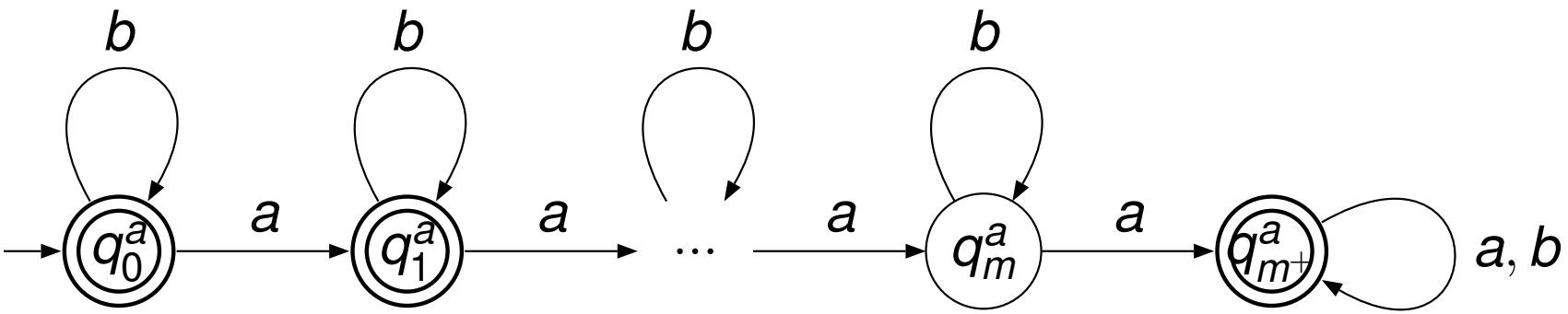
$q_j^a =$ "se han leído j símbolos a (módulo m)".



$L = \{w \in \Sigma^* : n_a(w) \neq m\}, m \in \{0, 1, 2, ...\}$

$q_j^a =$ "se han leído j símbolos a", con $j \in \{0, ..., m\}$

$q_{m^+}^a =$ "se han leído más de m símbolos a".



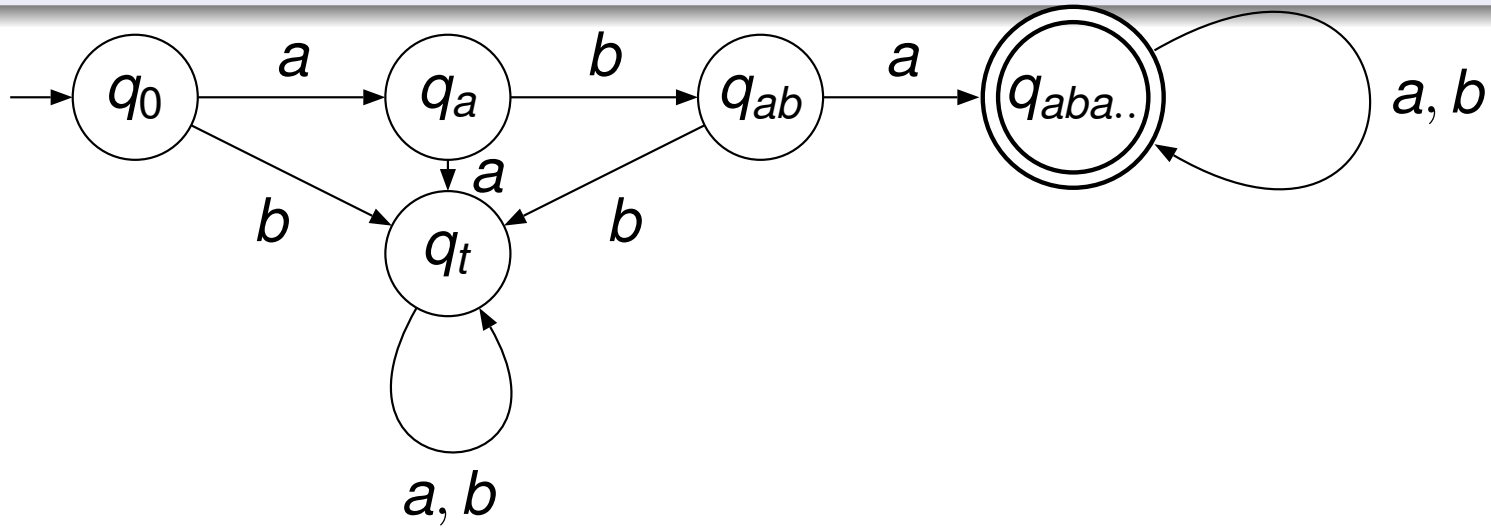
Comienzo de las palabras

- Se trata de lenguajes compuestos exclusivamente por las palabras que empiezan, o las que no empiezan, por una cierta cadena, por ejemplo:
  - "palabras que empiezan por *aba*".
  - "palabras que no empiezan por *aa*".
- Los segundos son los complementarios de los primeros, por lo que nos limitamos a discutir aquellos lenguajes que empiezan por una cadena no vacía x, que se pueden describir matemáticamente como sigue:

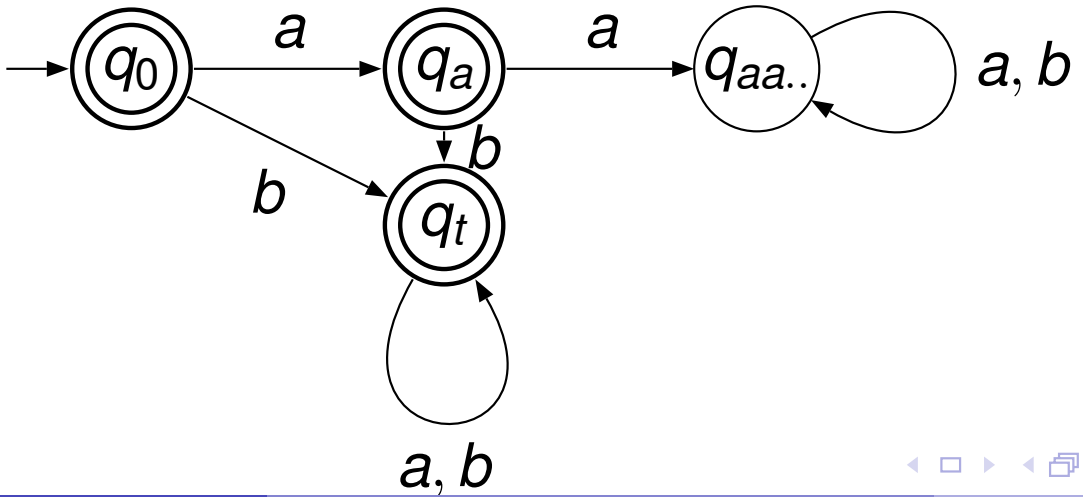
$$L_{x..} = \{xw \ : \ w \in \Sigma^*\}, \text{ con } x = a_1...a_m \in \Sigma^+ (m \geq 1)$$

- Para reconocer las palabras que empiezan con la cadena $x = a_1 ... a_m$ serán necesarios los siguientes $m + 2$ estados:
  - q_0 (no se ha leído nada)
  - q_{a_1} (se ha leído solo a_1).
  - $q_{a_1 a_2}$ (se ha leído solo $a_1 a_2$).
  - ...
  - $q_{x..}$ (la palabra empieza por x).
  - q_t (ninguna de las situaciones anteriores: la palabra no puede empezar por x)
- El estado q_0 es claramente el inicial y será no final puesto que evidentemente la palabra vacía no pertenece a L ($x \neq \lambda$).
- Del resto de estados, el único final debe ser $q_{x..}$ (las palabras aceptadas deben empezar necesariamente por x).
- Los estados $q_{x..}$ y q_t serán ambos estados trampa (se debe aceptar/rechazar cualquier palabra que empiece/no pueda empezar por x).
- Del resto de transiciones, de q_0 a $q_{x..}$ se procesa la palabra x, y las otras irán todas ellas al estado trampa q_t.

$L = \{abaw : w \in \Sigma^*\}$ (palabras que empiezan por *aba*)



$L = \Sigma^* - \{aaw : w \in \Sigma^*\}$ (palabras que no empiezan por *aa*)



Final de las palabras

- Se trata de lenguajes compuestos exclusivamente por las palabras que terminan, o las que no terminan, por una cierta cadena, por ejemplo:
  - "palabras que terminan por *aa*".
  - "palabras que no terminan por *aba*".
- Los segundos son los complementarios de los primeros, por lo que nos limitamos a discutir aquellos lenguajes que terminan por una cadena no vacía x:

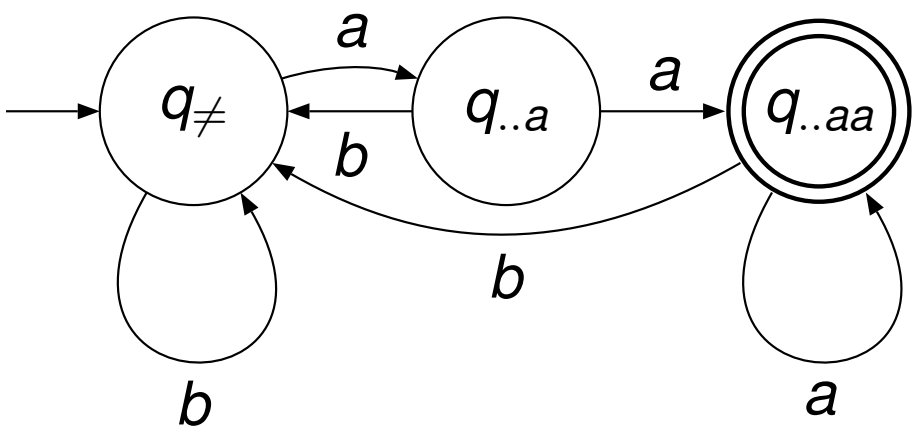
$$L_{..x} = \{wx \ : \ w \in \Sigma^*\}, \text{ con } x = a_1...a_m \in \Sigma^+ (m \geq 1)$$

- Para reconocer las palabras que terminan por $x = a_1 ... a_m$ serán necesarios los siguientes $m + 1$ estados:
  - $q_{..a_1}$ (lo último leído es a_1).
  - $q_{..a_1 a_2}$ (lo último leído es $a_1 a_2$).
  - ...
  - $q_{..x}$ (lo último leído es x).
  - $q_{\neq}$ (ninguna de las situaciones anteriores)

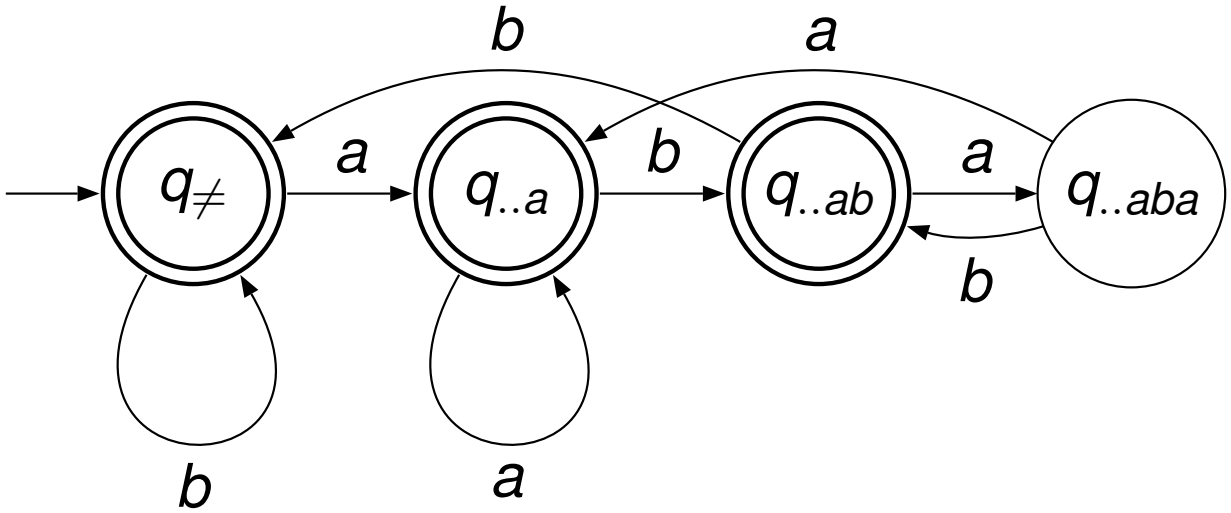
- El estado $q_{\neq}$ es el inicial (incluye el caso "no se ha leído nada") y no es final puesto que evidentemente la palabra vacía no termina por $x \neq \lambda$.
- Del resto de estados, el único final será $q_{..x}$ (las palabras aceptadas deben terminar necesariamente por x).
- Las transiciones de los estados se establecen en función de su semántica y del símbolo leído, entendiendo que tienen preferencia los estados con mayor número de símbolos (por ejemplo, como se muestra a continuación, desde el estado $q_{..aa}$, leyendo una a, se irá al propio $q_{..aa}$ en vez de a $q_{..a}$).

$L = \{waa : w \in \Sigma^*\}$ (palabras que terminan por aa)

- Estados: $q_{\neq}$ (inicial, lo último leído no es a, ni por lo tanto aa), $q_{..a}$ (lo último leído es a y no es aa) y $q_{..aa}$ (final, lo último leído es aa).
- Transiciones:
  - Desde $q_{\neq}$ leyendo una a se debe pasar a $q_{..a}$ (lo último leído es a), mientras que la lectura de una b mantiene el AFD en el estado inicial (lo último leído no es a).
  - Desde $q_{..a}$ leyendo una a se debe pasar a $q_{..aa}$ (puesto que lo último leído es aa), mientras que la lectura de una b provoca la vuelta a $q_{\neq}$ (lo último leído no es a).
  - Desde $q_{..aa}$ la lectura de una a no altera el estado (sigue siendo cierto que lo último leído es aa), mientras que una b provoca la vuelta a $q_{\neq}$ (lo último leído no es a).



$L = \Sigma^* - \{waba : w \in \Sigma^*\}$ (palabras que no terminan por *aba*)



Contenido de las palabras

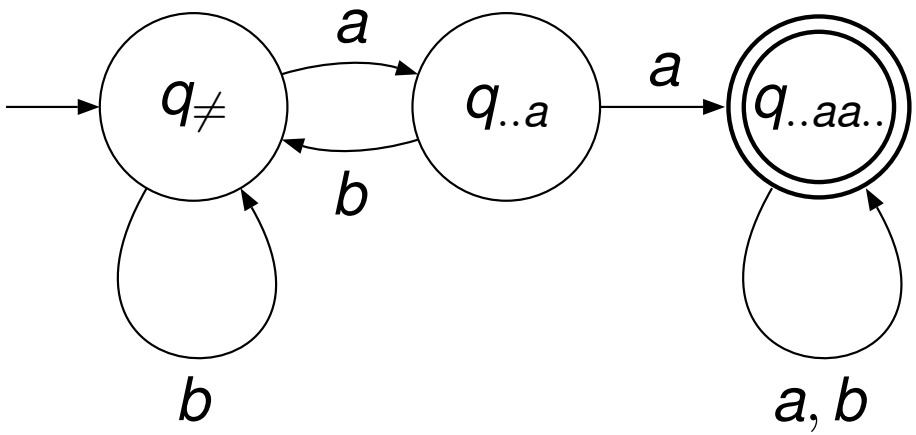
- Se trata de lenguajes compuestos exclusivamente por las palabras que contienen, o las que no contienen, una cierta cadena, por ejemplo:
  - "palabras que contienen *aa*".
  - "palabras que no contienen *aab*".
- Los segundos son los complementarios de los primeros, por lo que nos limitamos a discutir aquellos lenguajes que contienen una cadena no vacía x:

$$L_{..x..} = \{w_1 x w_2 \ : \ w_1, w_2 \in \Sigma^*\}, \text{ con } x = a_1 ... a_m \in \Sigma^+ (m \geq 1)$$

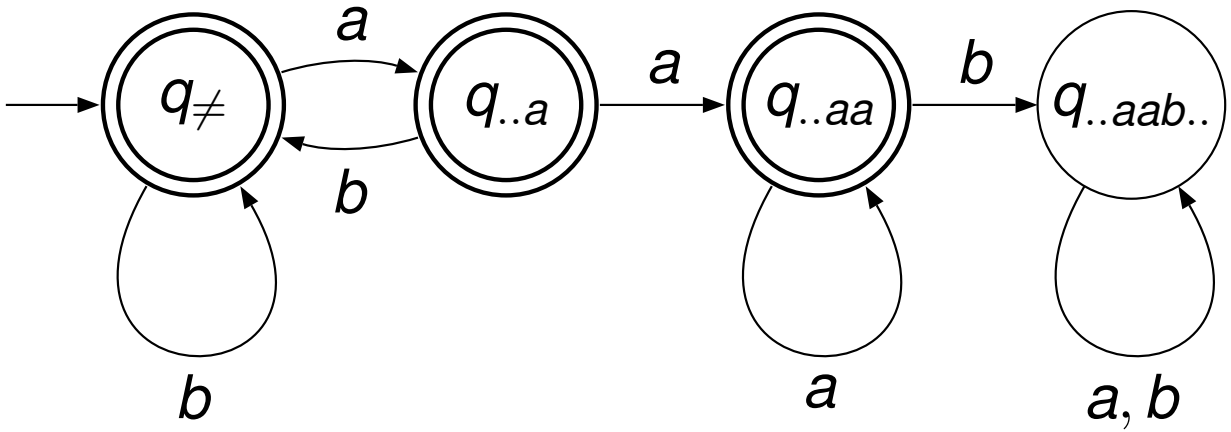
- Los AFDs para estos lenguajes se diseñan igual que los del apartado anterior (lenguajes que terminan por x) pero cambiando la semántica y por lo tanto el comportamiento del estado $q_{..x}$, que pasa a ser el estado *trampa* "$q_{..x..}$, la palabra contiene x".

$L = \{w_1 aaw_2 \ : \ w_1, w_2 \in \Sigma^*\}$ (palabras que contienen aa)

- Estados: $q_{\neq}$ (inicial, lo último leído no es a ni aa), $q_{..a}$ (lo último leído es a) y $q_{..aa..}$ (final, la palabra contiene aa).
- Transiciones:
  - Desde $q_{\neq}$ leyendo una a se debe pasar a $q_{..a}$ (lo último leído es a), mientras que la lectura de una b mantiene el AFD en el estado inicial (lo último leído no es a ni aa).
  - Desde $q_{..a}$ leyendo una a se debe pasar a $q_{..aa..}$ (la palabra contiene aa), mientras que la lectura de una b lleva al estado $q_{\neq}$ (lo último leído no es a ni aa).
  - $q_{..aa..}$ es un estado trampa puesto que cualquier palabra que haya llegado allí contiene aa.



$L = \Sigma^* - \{w_1 aabw_2 \ : \ w_1, w_2 \in \Sigma^*\}$
(palabras que no contienen *aab*)



AFDs PARA LENGUAJES COMPUESTOS (AFD PRODUCTO)

- A menudo aparecen lenguajes formados por palabras que deben cumplir no una sino dos características a la vez:

$$L = \{w \in \Sigma^* : w \text{ cumple } C_1 \textbf{ y } C_2\}$$

- Estos lenguajes pueden verse como la intersección de otros dos lenguajes, donde cada uno de ellos contiene las palabras que cumplen una de las dos características:

$$L = L_1 \cap L_2$$

siendo

$$L_1 = \{w \in \Sigma^* : w \text{ cumple } C_1\}$$
$$L_2 = \{w \in \Sigma^* : w \text{ cumple } C_2\}$$

- De manera similar, en ocasiones aparecen lenguajes compuestos por palabras que deben cumplir al menos una de dos posibles características dadas (pueden ser las dos):

$$L = \{w \in \Sigma^* : w \text{ cumple } C_1 \textbf{ o } C_2\}$$

- Estos lenguajes pueden verse como la unión de otros dos lenguajes, donde cada uno de ellos contiene las palabras que cumplen una de las dos características:

$$L = L_1 \cup L_2$$

siendo

$$L_1 = \{w \in \Sigma^* : w \text{ cumple } C_1\}$$
$$L_2 = \{w \in \Sigma^* : w \text{ cumple } C_2\}$$

- En general, las características C_1 y C_2 se pueden combinar no solamente mediante la conjunción o la disyunción, sino mediante cualquier operación lógica. Por ejemplo:

$$L = \{w \in \Sigma^* : w \text{ cumple } C_1 \textbf{ pero no } C_2\}$$
$$L = \{w \in \Sigma^* : w \text{ cumple } C_1 \textbf{ o } C_2 \textbf{ pero no ambas } \}$$
$$L = \{w \in \Sigma^* : \textbf{ si } w \text{ cumple } C_1 \textbf{ entonces } \text{ cumple } C_2\}$$

- Los lenguajes resultantes se pueden definir composicionalmente de manera conjuntista en base a los lenguajes asociados a las condiciones C_1 y C_2. Para los ejemplos anteriores:

$$L = L_1 - L_2 = L_1 \cap \overline{L_2}$$
$$L = L_1 \cap \overline{L_2} \cup L_2 \cap \overline{L_1}$$
$$L = \overline{L_1} \cup L_2$$

- El diseño de un AFD para un lenguaje que resulta, como en los casos anteriores, de la combinación de otros dos, debe seguir los cinco pasos generales especificados en el segundo apartado de esta guía, pero en general es recomendable hacerlo partiendo de los AFDs diseñados para los dos lenguajes que se combinan, construyendo a partir de ellos lo que se conoce como el autómata producto:
  - Los pasos primero y último, en los que, respectivamente, se analiza el lenguaje de partida y se hacen comprobaciones básicas, se mantienen.
  - Las pasos intermedios construyen el autómata producto en base a los AFDs de los lenguajes componentes.
- En lo que sigue se detalla el proceso de construcción del AFD producto, que se ilustra a continuación con varios ejemplos.

Construcción del AFD producto

Sea L un lenguaje que se puede expresar mediante la combinación (intersección, unión, diferencia, ...) de otros dos lenguajes, L_1 y L_2. Un AFD que reconoce L es el AFD producto de A_1 y A_2, siendo estos AFDs que reconocen L_1 y L_2. Una vez analizado el lenguaje de partida en el PASO 1, el AFD producto se construye como sigue.

PASO 2. Diseño del conjunto de estados

- En primer lugar se diseñan, por separado, los estados de A_1 y A_2 necesarios para reconocer los lenguajes L_1 y L_2.
- Si Q_1 y Q_2 son los conjuntos de estados obtenidos, el conjunto de estados del AFD producto es $Q_1 \times Q2$, que, como se explica a continuación, contiene todas las posibles combinaciones de un estado de Q_1 con uno de Q_2.

1. Sean $q_{11}, \cdots, q_{1n_1}$ los n_1 estados del AFD A_1.
2. Sean $q^{21}, \cdots, q^{2n_2}$ los n_2 estados del AFD A_2.
3. El conjunto de estados del AFD producto para L estará compuesto por los $n_1 \times n_2$ estados resultantes de combinar cada estado de A_1 con cada uno de los estados de A_2:
   - La combinación de q_{11} con los n_2 estados de A_2 dará lugar a los n_2 estados $q_{11}^{21} \cdots q_{11}^{2n_2}$.
   - $\cdots$
   - La combinación de q_{1n_1} con los n_2 estados de A_2 dará lugar a los n_2 estados $q_{1n_1}^{21} \cdots q_{1n_1}^{2n_2}$.

   de forma que la semántica de cualquier estado q_i^j será

$$q_i^j = \text{se cumple tanto } i \text{ como } j$$

Observación

Aunque el proceso de construcción anterior siempre produce $n_1 \times n_2$ estados, *no todos ellos serán imprescindibles*, es decir, el AFD producto resultante no es necesariamente el mínimo, ya que:

- Algunos estados podrán no tenerse en cuenta al resultar inaccesibles (cuando no existe ningún camino en el grafo que permita llegar a ellos partiendo del estado inicial) y ser por lo tanto prescindibles. Estos estados siempre serán los mismos, independientemente de la operación con la que se estén combinando los lenguajes L_1 y L_2, y se podrán detectar:
  - Informalmente en el momento de su construcción, al constatar que presentan una semántica contradictoria.
  - O bien, una vez construido el AFD, aplicando el algoritmo de minimización, que comprobará su inaccesibilidad y los eliminará.
- Además, algunos estados podrán resultar equivalentes entre sí, aunque esto sí depende de cómo se estén combinando los lenguajes (ver PASO 3).

PASO 3. Determinación del estado inicial y de los finales

El estado inicial será el (único) estado del AFD producto construido *a partir de los estados iniciales de* A_1 *y* A_2.

Los estados finales se determinan atendiendo a la forma en la que se combinan los lenguajes componentes. Por ejemplo, siendo $F_1 \subseteq Q_1$ y $F_2 \subseteq Q_2$ los conjuntos de estados finales de los autómatas A_1 y A_2:

- Si es $L = L_1 \cap L_2$ (se cumple C_1 **y** C_2), el conjunto de estados finales del AFD producto será $F_1 \times F_2$, es decir, serán finales todos aquellos estados q_i^j tales que *sus dos componentes, i y j, se correspondan con estados finales*.

- Si es $L = L_1 \cup L_2$ (se cumple C_1 **o** C_2), el conjunto de estados finales del AFD producto será $(F_1 \times Q_2) \cup (Q_1 \times F_2)$, es decir, serán finales todos aquellos estados q_i^j tales que *al menos una de sus dos componentes -pueden ser ambas- se corresponda con un estado final*.
- Si es $L = L_1 - L_2$ (se cumple C_1 **pero no** C_2), el conjunto de estados finales del AFD producto será $F_1 \times \overline{F_2}$, siendo $\overline{F_2} = Q_2 - F_2$, es decir, serán finales todos aquellos estados q_i^j tales que *la componente procedente de L_1, i, es final y la de L_2, j, no lo es*.
- Si es $L = \overline{L_1} \cup L_2$ (**si** se cumple C_1, **entonces** se cumple C_2) el conjunto de estados finales del AFD producto será $(\overline{F_1} \times Q_2) \cup (Q_1 \times F_2)$, es decir, serán finales todos aquellos estados q_i^j tales que *o bien la componente procedente de A_1 no es final, o bien la componente procedente de A_2 sí es final* (pueden ser ambas cosas).

Observación (1/2)

- Algunos grupos de estados podrán fundirse en un único estado por resultar equivalentes entre sí.
- Estas equivalencias dependerán de la combinación que se esté realizando entre L_1 y L_2, por lo que solo se podrán detectar después de determinar qué estados son finales:
  - Informalmente, analizando y comparando sus semánticas.
  - O bien, una vez construido el AFD, aplicando el algoritmo de minimización, que construirá las clases de equivalencia pertinentes.

Observación (2/2)

Por ejemplo:

- Si es $L = L_1 \cap L_2$ (se cumple C_1 **y** C_2): en particular -aunque no solo- serán equivalentes entre sí todos aquellos estados cuya semántica indique que *al menos una* de las dos condiciones, C_1 o C_2 (o ambas), *ni se cumple ni se podrá cumplir nunca, pase lo que pase a continuación*, y se fundirán en un único estado, no final y trampa.
- Si es $L = L_1 \cup L_2$ (se cumple C_1 **o** C_2): en particular -aunque no solo- serán equivalentes entre sí aquellos estados cuya semántica *asegure el cumplimiento de al menos una de las dos condiciones dadas (o las dos), pase lo que pase a continuación*. Todos los estados que cumplan lo anterior se fundirán en un único estado, final y trampa.

PASO 4. Cálculo de la función de transición

Las transiciones de los estados (una y solo una para cada símbolo del alfabeto) se determinan en función de su semántica, también reflejada en las transiciones de los AFDs para L_1 y L_2:

desde un estado cualquiera q_i^j del autómata producto (construido a partir del estado q_i del AFD de L_1 y del estado q_j del AFD de L_2), leyendo el símbolo $a \in \Sigma$, se transitará al estado $q_{i_a}^{j_a}$, siendo $i_a = f_1(q_i, a), j_a = f_2(q_j, a)$ y siendo f_1 y f_2 las funciones de transición de los AFDs de L_1 y L_2, respectivamente.

Observación

- La construcción de un AFD producto es por lo tanto independiente de la operación (unión, intersección, etc) con la que se combinan los lenguajes componentes salvo en lo que atañe a la forma en la que se identifican los estados finales y los posibles estados equivalentes.

- La técnica empleada puede extenderse fácilmente a lenguajes descritos mediante la composición de más de dos lenguajes: por ejemplo, si $L = L_1 \cap L_2 \cap L_3$ y los AFDs para L_1, L_2 y L_3 tienen, respectivamente, n_1, n_2 y n_3 estados, el AFD producto para L tendría $n_1 \times n_2 \times n_3$ estados potenciales (aunque muchos de ellos resultarán inaccesibles o equivalentes entre sí).

$L = \{w \in \{a,b\}^* : w \text{ contiene } ba \textbf{ y } \text{empieza por } b\}$

A continuación se discute en detalle el diseño de un AFD para el lenguaje

$$L = \{w \in \{a,b\}^* : w \text{ contiene } ba \textbf{ y } \text{empieza por } b\}$$

que se puede ver como la intersección de otros dos lenguajes, L_1 y L_2,

$$L = L_1 \cap L_2$$

donde

$$L_1 = \{w \in \{a,b\}^* : w \text{ contiene } ba\}$$
$$L_2 = \{w \in \{a,b\}^* : w \text{ empieza por } b\}$$

y se puede por lo tanto resolver mediante el AFD producto.

PASO 1. Análisis del lenguaje de partida

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **y** empieza por $b\}$ (1/7))

- Algunas palabras $x \in L$:

$$ba, bba, baa, bab, bbaa, bbba, baba, babb, baaa, bbaabbba, ...$$

- Algunas palabras $x \notin L$:
  - cualquier palabra de la forma ay con $y \in \{a,b\}^*$, como por ejemplo

$$a, ab, aa, aba, aabab, ...$$

  - cualquier palabra de la forma b^n con $n \geq 0$, como por ejemplo

$$\lambda, b, bb, bbb, ...$$

PASO 2. Diseño del conjunto de estados

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **y** empieza por $b\}$ (2/7))

1. La característica "w contiene ba" requiere 3 estados: $q_{..b}$ (lo último es b), $q_{..ba..}$ (contiene ba) y $q_{\neq}$ (ninguna de las anteriores).
2. La característica "w empieza por b" requiere otros 3 estados: q^0 (no se ha leído nada), $q^{b..}$ (empieza por b) y q^t (ninguna de las anteriores).
3. Los estados obtenidos combinando ambas características serán por lo tanto los siguientes $3 \times 3 = 9$ estados:

$$\begin{array}{ccc} q^0_{..b} & q^{b..}_{..b} & q^t_{..b} \\ q^0_{..ba..} & q^{b..}_{..ba..} & q^t_{..ba..} \\ q^0_{\neq} & q^{b..}_{\neq} & q^t_{\neq} \end{array}$$

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **y** empieza por $b\}$ (3/7))

De los 9 estados anteriores, los siguientes son *inaccesibles* y por lo tanto se pueden descartar:

- El estado $q^0_{..b}$ es inaccesible ya que su semántica es "lo último leído es b y no se ha leído nada", afirmaciones evidentemente contradictorias.
- El estado $q^0_{..ba..}$ se debe descartar por el mismo motivo: es imposible que se dé simultáneamente "contiene ba" y "no se ha leído nada".
- El estado $q^{b..}_{\neq}$ será también inaccesible: si una palabra empieza por b, entonces o bien lo último leído es b (cosa que no puede ocurrir en $q_{\neq}$) o bien lo último leído es a, en cuyo caso en algún momento se ha tenido la secuencia ba (lo cual también es contradictorio con $q_{\neq}$).

PASO 3. Determinación del estado inicial y de los finales

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **y** empieza por $b\}$ (4/7))

- Estado inicial: $q^0_{\neq}$ puesto que $q_{\neq}$ y q_0 son claramente los estados iniciales de los AFDs para L_1 y L_2, respectivamente.
- Estados finales: el único estado final es $q^{b..}_{..ba..}$, puesto que es el único construido a partir de dos estados finales ($F_1 \times F_2$).
- Los tres estados que contienen q^t son *equivalentes entre sí* y se podrían fundir en un único estado, no final y trampa, ya que q^t indica que la palabra es no vacía y empieza por algo distinto de b (en nuestro caso, empieza por a), y todas esas palabras, contengan o no ba, deberán siempre rechazarse puesto que *nunca* podrán cumplir la condición C_2 (empieza por b).

PASO 4. Cálculo de la función de transición

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **y** empieza por $b\}$ (5/7))

- Desde el estado inicial $q^0_{\neq}$, la transición a debe ir al estado $q^t_{\neq}$, puesto que sigue siendo cierto $q_{\neq}$ (ni lo último es b ni se ha procesado ba) pero q^0 (no se ha leído nada) deja de ser cierto y pasa a ser q^t (no empieza por b).

- Desde el estado inicial $q^0_{\neq}$, la transición b debe ir al estado $q^{b..}_{..b}$, puesto que se cumple que lo último leído es una b y que la palabra empieza por b.

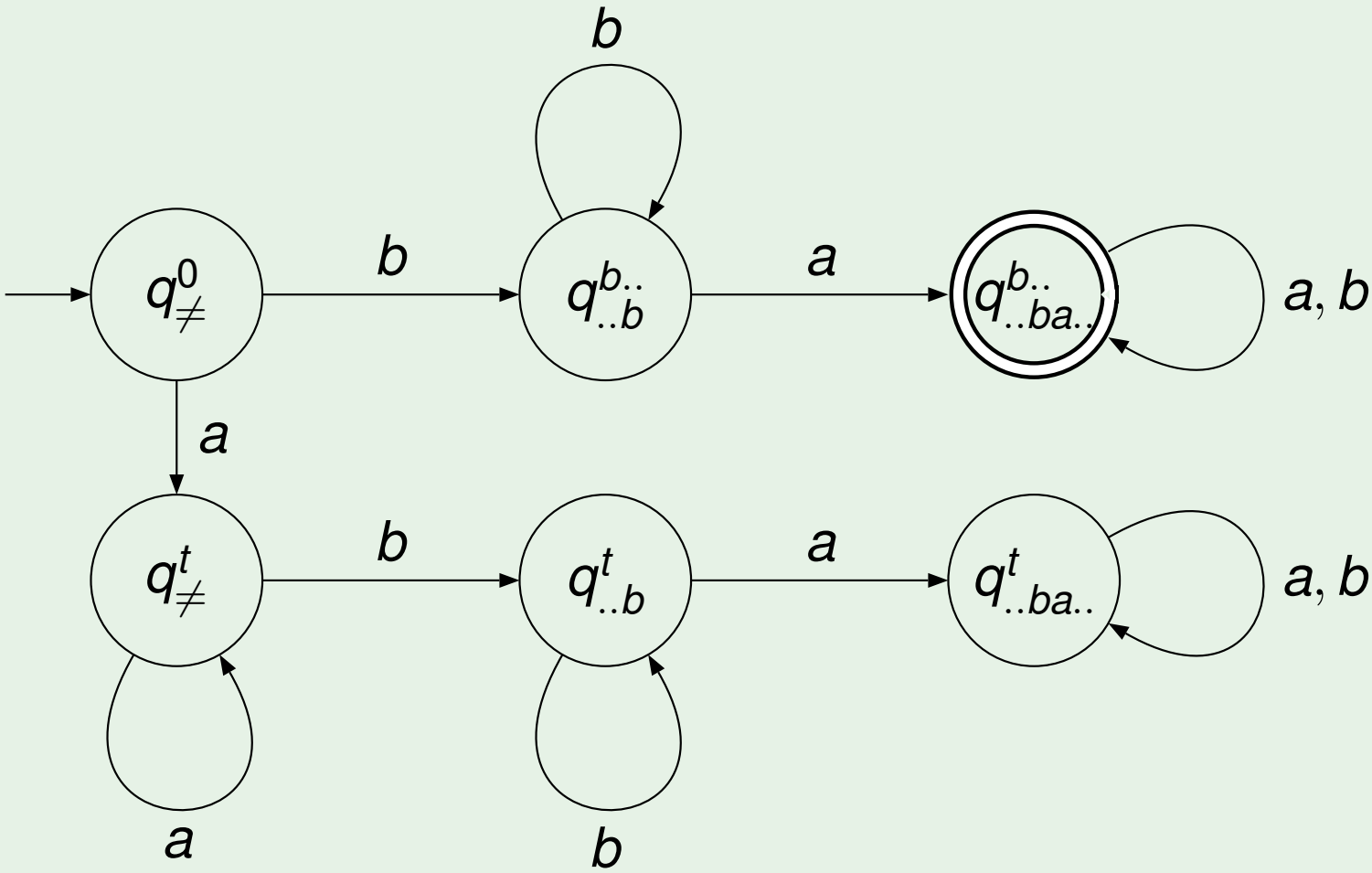
- Las transiciones correspondientes al resto de estados se razonan de forma similar, según van apareciendo, hasta que finalmente se obtiene el AFD cuyo grafo se dibuja a continuación.

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **y** empieza por $b\}$ (6/7))

Teniendo en cuenta todo lo anterior, un posible AFD para el lenguaje

$$L = \{w \in \{a,b\}^* : w \text{ contiene } ba \textbf{ y } \text{empieza por } b\}$$

es

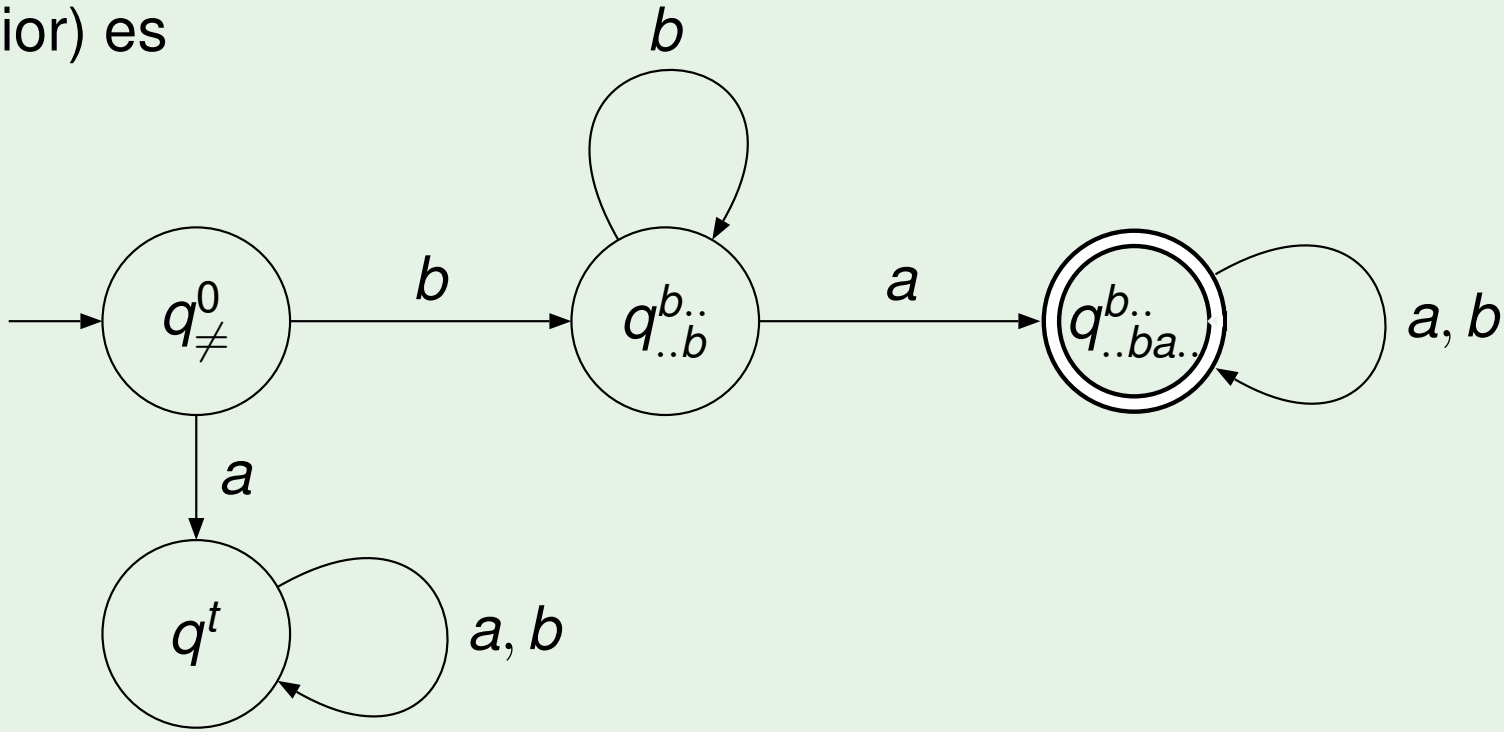


Observación

- En el AFD anterior se han obviado los 3 estados inaccesibles mencionados previamente.
- Se han mantenido por el momento los 3 estados equivalentes.

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **y** empieza por $b\}$ (7/7))

El AFD mínimo (obtenido aplicando el algoritmo de minimización al AFD anterior) es



$L = \{w \in \{a,b\}^* : w \text{ contiene } ba \textbf{ o } \text{empieza por } b\}$

A continuación se discute el diseño de un AFD para el lenguaje

$$L = \{w \in \{a,b\}^* : w \text{ contiene } ba \textbf{ o } \text{empieza por } b\}$$

que se puede ver como $L = L_1 \cup L_2$ donde

$$L_1 = \{w \in \{a,b\}^* : w \text{ contiene } ba\}$$
$$L_2 = \{w \in \{a,b\}^* : w \text{ empieza por } b\}$$

Dado que el lenguaje es una combinación de los mismos lenguajes L_1 y L_2 del ejemplo anterior, la construcción del autómata producto de este último se puede reutilizar prácticamente por completo, salvo en lo referente a estados finales y estados equivalentes.

PASO 1. Análisis del lenguaje de partida

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **o** empieza por $b\}$ (1/5))

- Algunas palabras $x \in L$:
  - cualquier palabra de la forma $x_1 bax_2$ con $x_1, x_2 \in \{a,b\}^*$, como por ejemplo
    $$ba, aba, bba, baa, bab, aababa, ...$$
  - cualquier palabra de la forma by con $y \in \{a,b\}^*$, como por ejemplo
    $$b, bb, ba, bba, bbb, bbbba, baabaab, ...$$
- Algunas palabras $x \notin L$:
  - la palabra vacía λ.
  - cualquier palabra de la forma $a^m b^n$ con $m > 0, n \geq 0$, como por ejemplo
    $$a, aa, ab, abb, aaab, ...$$

PASO 2. Diseño del conjunto de estados

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **o** empieza por $b\}$ (2/5))

- El conjunto de estados se construye igual que para la intersección, obteniéndose los mismos $3 \times 3 = 9$ estados compuestos:

$$\begin{array}{ccc} q^0_{..b} & q^{b..}_{..b} & q^t_{..b} \\ q^0_{..ba..} & q^{b..}_{..ba..} & q^t_{..ba..} \\ q^0_{\neq} & q^{b..}_{\neq} & q^t_{\neq} \end{array}$$

  donde cada q^j_i indica que "se cumple tanto i como j".

- De los anteriores, y por los motivos explicados previamente, los estados $q^0_{..b}$, $q^0_{..ba..}$ y $q^{b..}_{\neq}$ se pueden ignorar al ser *inaccesibles*.

PASO 3. Determinación del estado inicial y de los finales

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **o** empieza por $b\}$ (3/5))

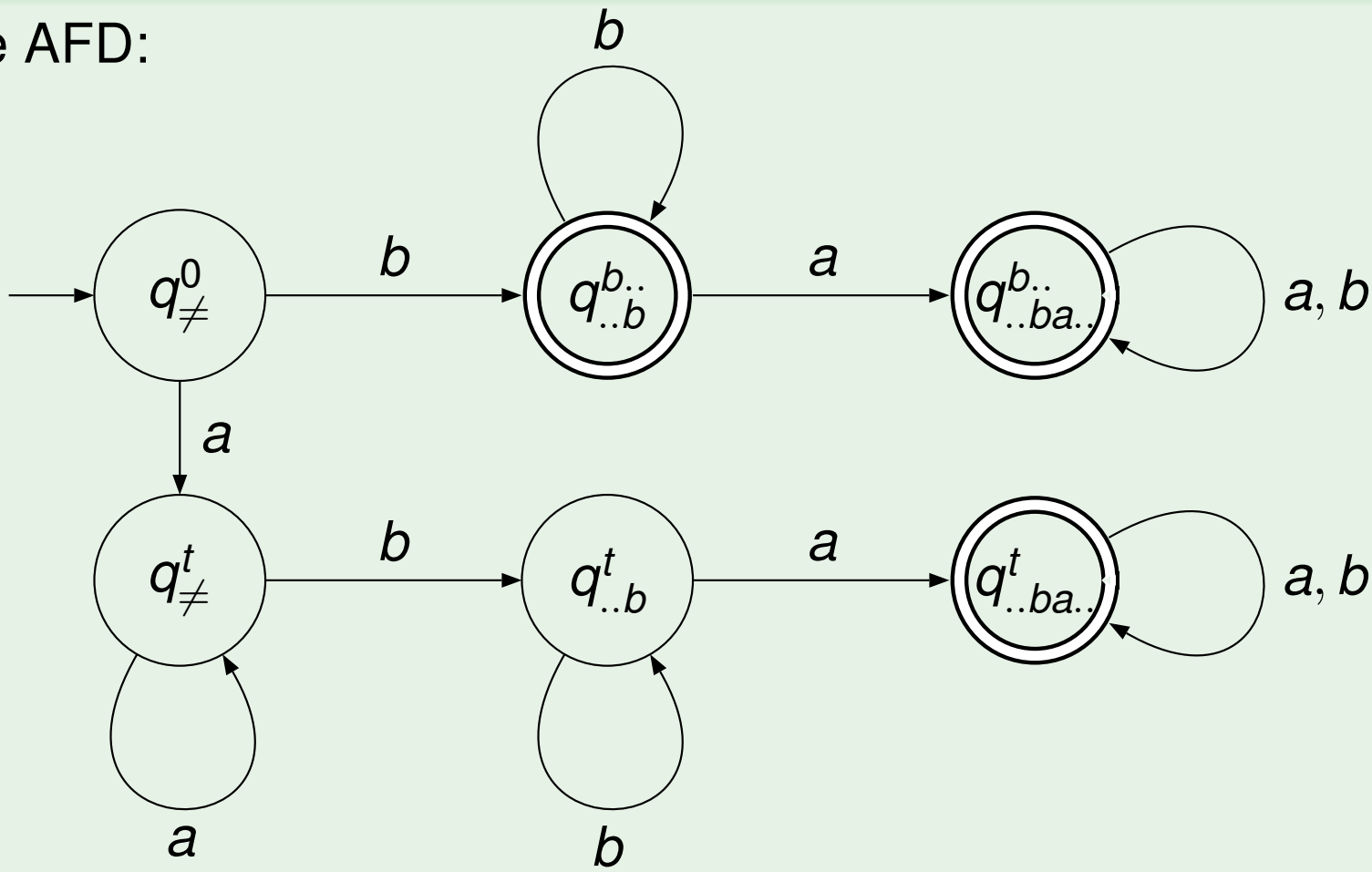
- Estado inicial: $q^0_{\neq}$, puesto que $q_{\neq}$ y q_0 son claramente los estados iniciales de los AFDs A_1 y A_2, respectivamente.
- Estados finales: $q^{b..}_{..b}$, $q^{b..}_{..ba..}$ y $q^{t}_{..ba..}$, puesto que son aquellos en los que al menos una de sus dos componentes (las dos en el caso de $q^{b..}_{..ba..}$) procede de un estado final de los AFDs A_1 y A_2 (es decir, los que pertenecen a $(F_1 \times Q_2) \cup (Q_1 \times F_2)$).
- Los estados ahora equivalentes entre sí serán $q^{b..}_{..b}$, $q^{b..}_{..ba..}$ y $q^{t}_{..ba..}$, puesto que todos ellos garantizan que, pase lo que pase a continuación, al menos una de las dos condiciones se cumplirá (las dos en el caso de $q^{b..}_{..ba..}$).

PASO 4. Cálculo de la función de transición

Exactamente igual que para el caso de la intersección.

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **o** empieza por $b\}$ (4/5))

Un posible AFD:

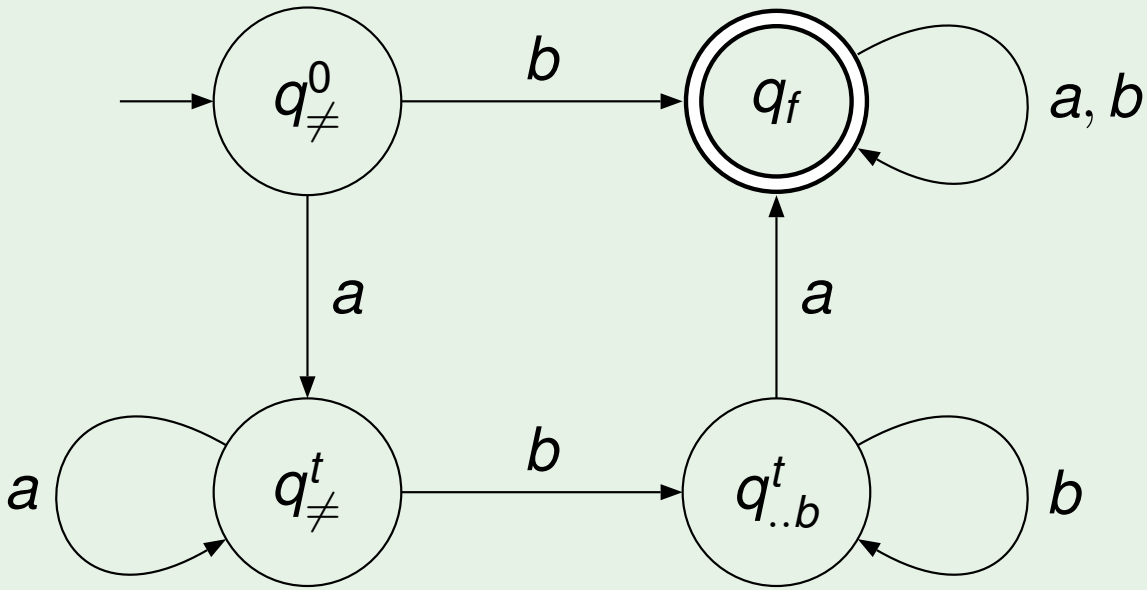


Observación

- En el AFD anterior se han obviado los 3 estados inaccesibles mencionados previamente.
- Se han mantenido por el momento los 3 estados equivalentes.

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **o** empieza por $b\}$ (5/5))

El AFD mínimo (obtenido aplicando el algoritmo de minimización al AFD anterior y llamando q_f al estado final fusionado) es



$L = \{w \in \{a,b\}^* : w \text{ contiene } ba \textbf{ pero no } \text{empieza por } b\}$

A continuación se discute el diseño de un AFD para el lenguaje

$$L = \{w \in \{a,b\}^* : w \text{ contiene } ba \textbf{ pero no } \text{empieza por } b\}$$

que se puede ver como la diferencia $L = L_1 - L_2$ donde

$$L_1 = \{w \in \{a,b\}^* : w \text{ contiene } ba\}$$
$$L_2 = \{w \in \{a,b\}^* : w \text{ empieza por } b\}$$

es decir, una nueva combinación de los mismos lenguajes L_1 y L_2 de los ejemplos anteriores, por lo que gran parte del trabajo hecho (todo menos finales y equivalencias) se puede reutilizar.

PASO 1. Análisis del lenguaje de partida

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **pero no** empieza por $b\}$ (1/5))

- Algunas palabras $x \in L$:
  - La de menor longitud será aba.
  - En general, cualquier palabra de la forma ax_1bax_2 con $x_1, x_2 \in \{a,b\}^*$, como por ejemplo

$$aba, aaba, abba, abab, aaabbba, \ldots$$

- Algunas palabras $x \notin L$:
  - Cualquier palabra que no contenga ba, es decir, cualquier palabra $a^n b^m$, con $n, m \geq 0$, entre ellas la palabra vacía λ.
  - Cualquier palabra empezando por b, es decir, cualquier palabra $bx, x \in \{a,b\}^*$.

PASO 2. Diseño del conjunto de estados

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **pero no** empieza por $b\}$ (2/5))

- El conjunto de estados se construye igual que para la intersección o unión, obteniéndose los mismos $3 \times 3 = 9$ estados compuestos:

$$\begin{array}{ccc} q^0_{..b} & q^{b..}_{..b} & q^t_{..b} \\ q^0_{..ba..} & q^{b..}_{..ba..} & q^t_{..ba..} \\ q^0_{\neq} & q^{b..}_{\neq} & q^t_{\neq} \end{array}$$

  donde cada q^j_i indica que "se cumple tanto i como j".

- De los anteriores, y por los motivos explicados previamente, los estados $q^0_{..b}$, $q^0_{..ba..}$ y $q^{b..}_{\neq}$ se pueden ignorar al ser *inaccesibles*.

PASO 3. Determinación del estado inicial y de los finales

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **pero no** empieza por $b\}$ (3/5))

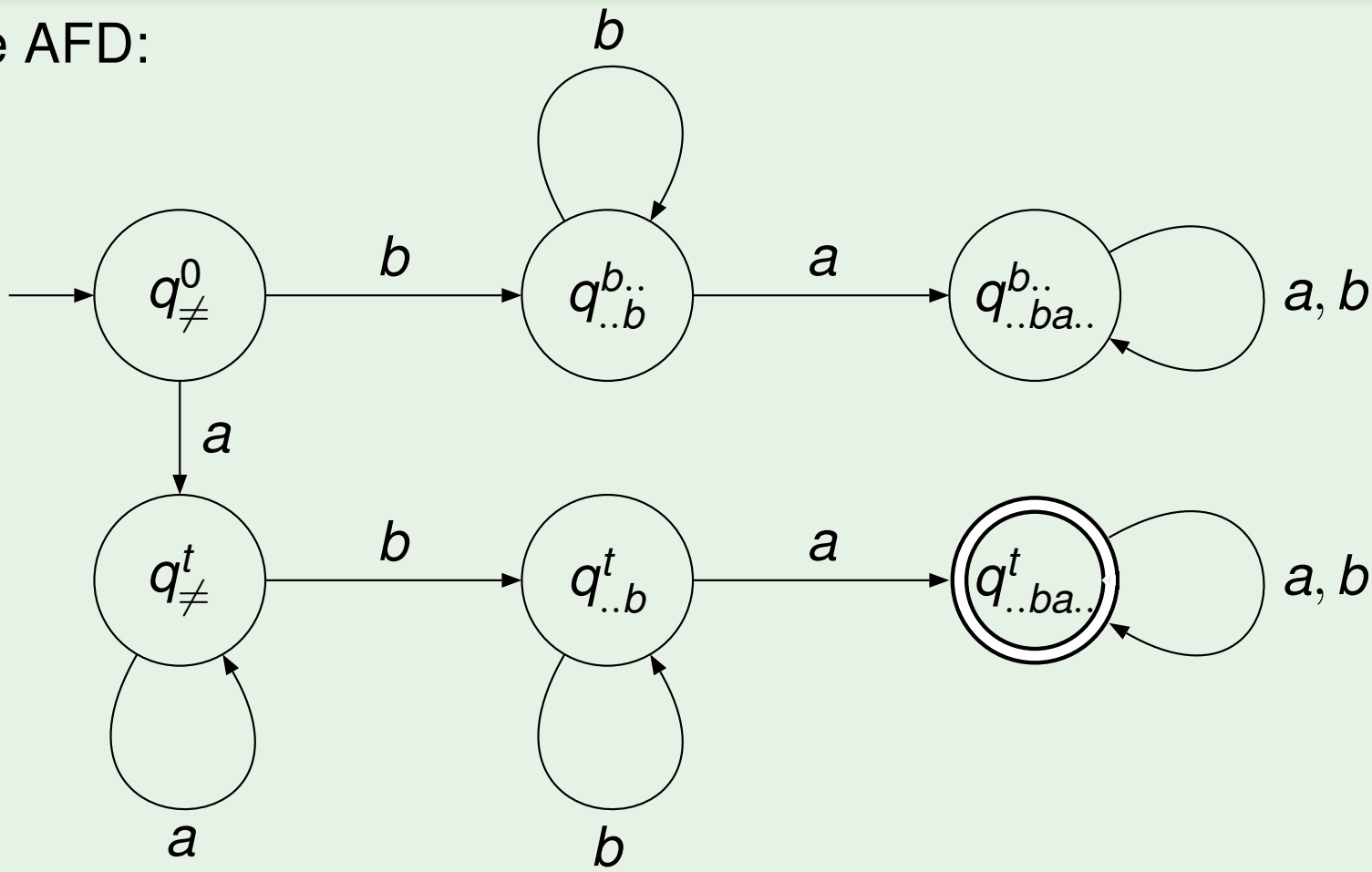
- Estado inicial: $q^0_{\neq}$, puesto que $q_{\neq}$ y q_0 son claramente los estados iniciales de los AFDs A_1 y A_2, respectivamente.
- Estados finales: ahora el único estado final será $q^t_{..ba..}$, puesto que es el único estado accesible con la primera componente de F_1 (contiene ba) y la segunda de $\overline{F_2}$ (no empieza por b) ($F_1 \times \overline{F_2}$).
- Los estados ahora equivalentes entre sí serán $q^{b..}_{..b}$ y $q^{b..}_{..ba..}$, puesto que en ambos casos la palabra empieza por b y deberá por lo tanto rechazarse, pase lo que pase después.

PASO 4. Cálculo de la función de transición

Exactamente igual que para los dos ejemplos anteriores.

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **pero no** empieza por $b\}$)

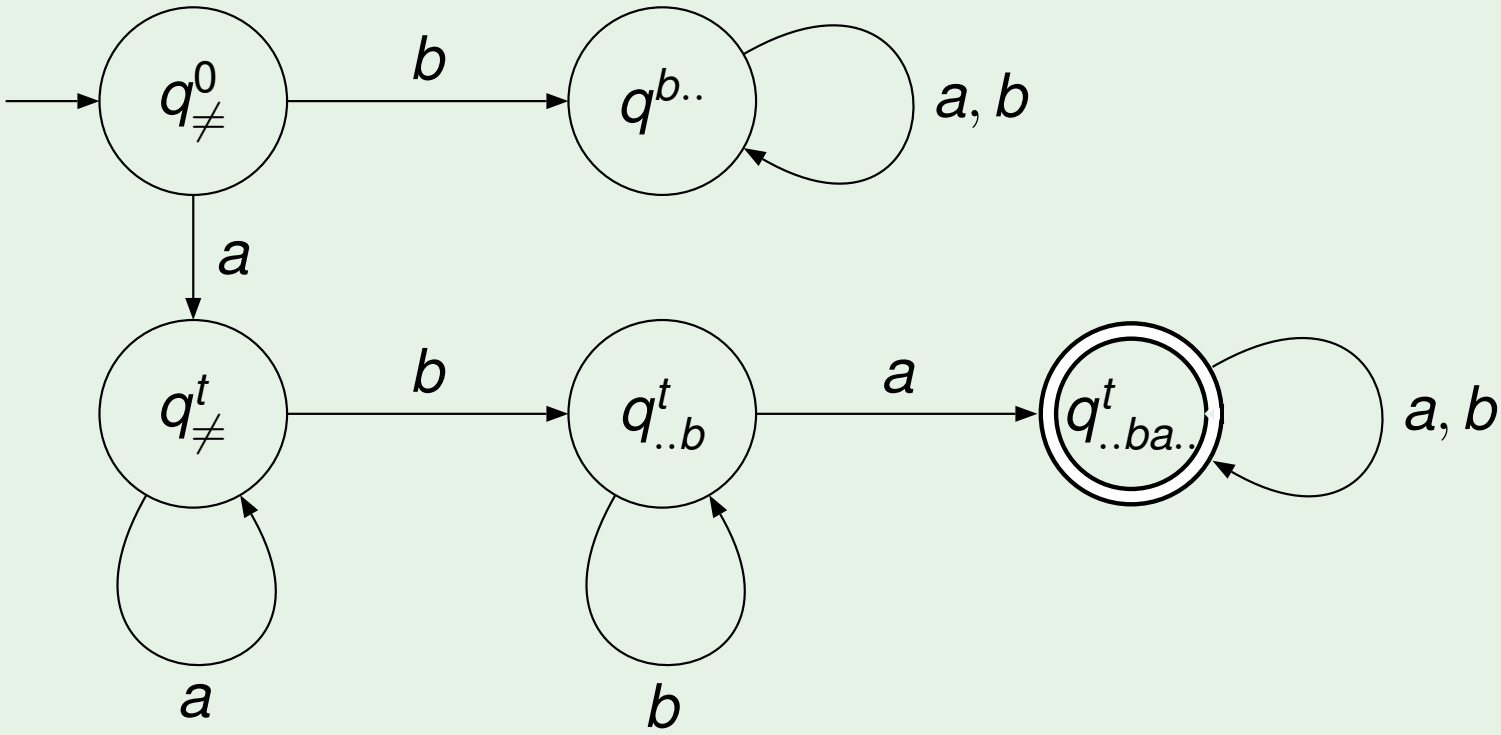
Un posible AFD:



En el AFD anterior se han obviado los 3 estados inaccesibles pero se han mantenido los 2 estados equivalentes.

Ejemplo ($\{w \in \{a,b\}^* : w$ contiene ba **pero no** empieza por $b\}$)

El AFD mínimo (obtenido aplicando el algoritmo de minimización al AFD anterior y llamando $q^{b..}$ al estado trampa no final fusionado) es



$L = \{w \in \{a,b\}^* : \textbf{si } w \text{ contiene } ba, \textbf{ entonces } \text{empieza por } b\}$

A continuación se discute el diseño de un AFD para el lenguaje

$$L = \{w \in \{a,b\}^* : \textbf{si } w \text{ contiene } ba, \textbf{ entonces } \text{empieza por } b\}$$

que se puede ver como la unión $L = \overline{L_1} \cup L_2$ donde

$$L_1 = \{w \in \{a,b\}^* : w \text{ contiene } ba\}$$
$$L_2 = \{w \in \{a,b\}^* : w \text{ empieza por } b\}$$

es decir, una nueva combinación de los mismos lenguajes L_1 y L_2 de los ejemplos anteriores, con la particularidad además de que se trata del complementario del último ejemplo, por lo que se podría resolver a partir de él aplicando lo explicado aqui.

PASO 1. Análisis del lenguaje de partida

Ejemplo
($\{w \in \{a,b\}^* :$ **si** w contiene ba, **entonces** empieza por $b\}$)

- Algunas palabras $x \in L$:
  - Cualquier palabra que *no* contenga ba, es decir, cualquier palabra $a^m b^n$ con $m, n \geq 0$, entre ellas la palabra vacía λ.
  - Cualquier palabra que contenga ba siempre que empiece por b, es decir, además de ba, cualquier palabra de la forma $bx_1 bax_2$ con $x_1, x_2 \in \{a,b\}^*$, como por ejemplo

$$bba, baba, bbba, bbaa, bbab, \ldots$$

- Algunas palabras $x \notin L$: cualquiera que contenga ba y empiece por a, como por ejemplo

$$aba, abaa, abab, aaba, abba, aababbab, \ldots$$

PASO 2. Diseño del conjunto de estados

Ejemplo
($\{w \in \{a,b\}^* :$ **si** w contiene ba, **entonces** empieza por $b\}$)

- El conjunto de estados se construye igual que para la intersección o unión, obteniéndose los mismos $3 \times 3 = 9$ estados compuestos:

$$\begin{array}{ccc} q^0_{..b} & q^{b..}_{..b} & q^t_{..b} \\ q^0_{..ba..} & q^{b..}_{..ba..} & q^t_{..ba..} \\ q^0_{\neq} & q^{b..}_{\neq} & q^t_{\neq} \end{array}$$

  donde cada q^j_i indica que "se cumple tanto i como j".

- De los anteriores, y por los motivos explicados previamente, los estados $q^0_{..b}$, $q^0_{..ba..}$ y $q^{b..}_{\neq}$ se pueden ignorar al ser *inaccesibles*.

PASO 3. Determinación del estado inicial y de los finales

Ejemplo
($\{w \in \{a,b\}^* :$ **si** w contiene ba, **entonces** empieza por $b\}$)

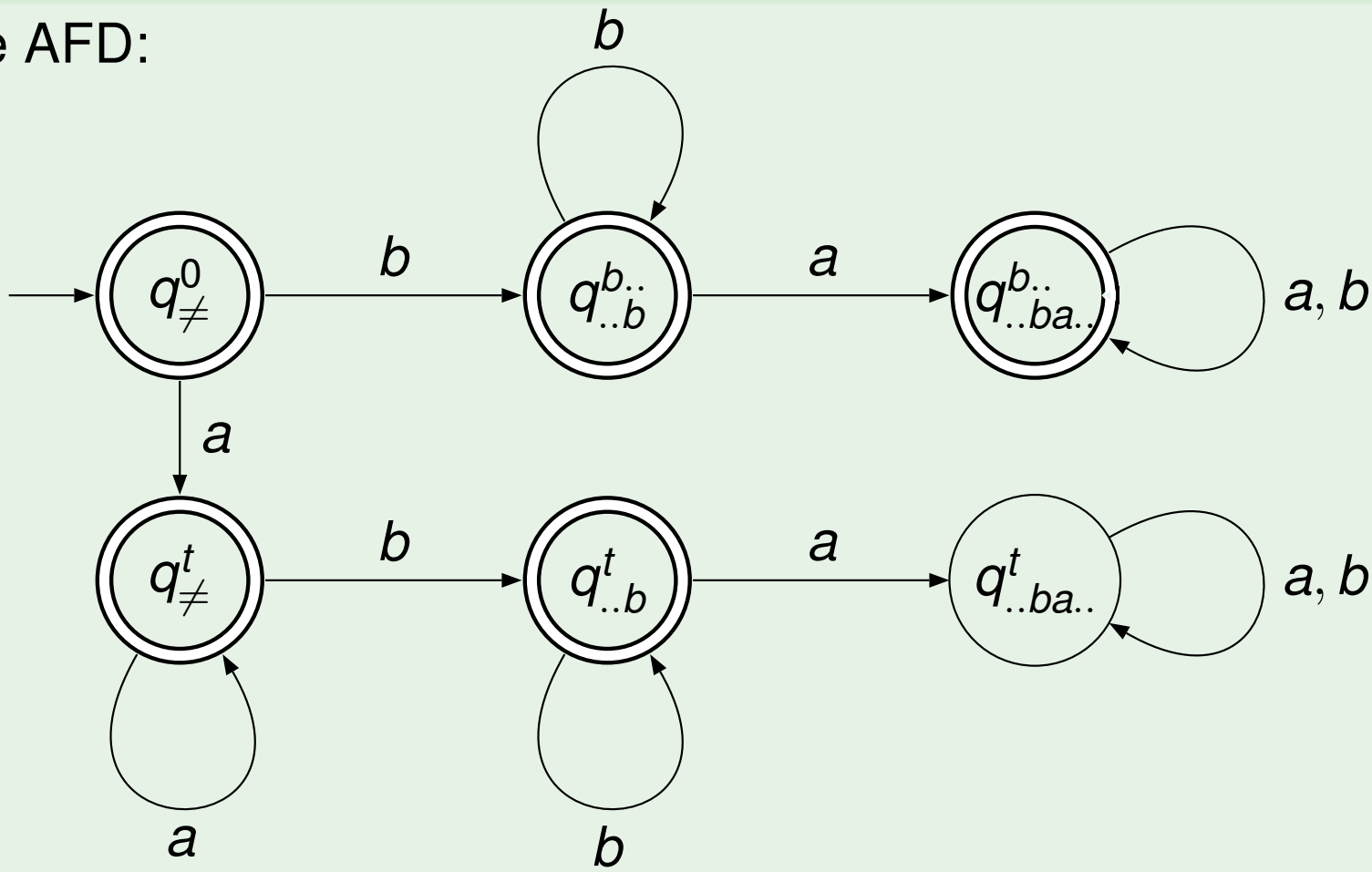
- Estado inicial: $q^0_{\neq}$, puesto que $q_{\neq}$ y q_0 son claramente los estados iniciales de los AFDs A_1 y A_2, respectivamente.
- Estados finales: los que no contienen ba, que son $q^0_{\neq}$, $q^t_{\neq}$, $q^{b,}_{..b}$ y $q^t_{..b}$, más los que empiezan por b, que son $q^{b,}_{..b}$ (que ya está en el grupo anterior) y $q^{b,}_{..ba..}$ $((\overline{F_1} \times Q_2) \cup (Q_1 \times F_2))$
- Los estados ahora equivalentes entre sí serán $q^{b,}_{..b}$ y $q^{b,}_{..ba..}$, puesto que en ambos casos la palabra empieza por b y deberá por lo tanto aceptarse, pase lo que pase después.

PASO 4. Cálculo de la función de transición

Exactamente igual que para los ejemplos anteriores.

Ejemplo ({**si** *w* contiene *ba*, **entonces** empieza por *b*})

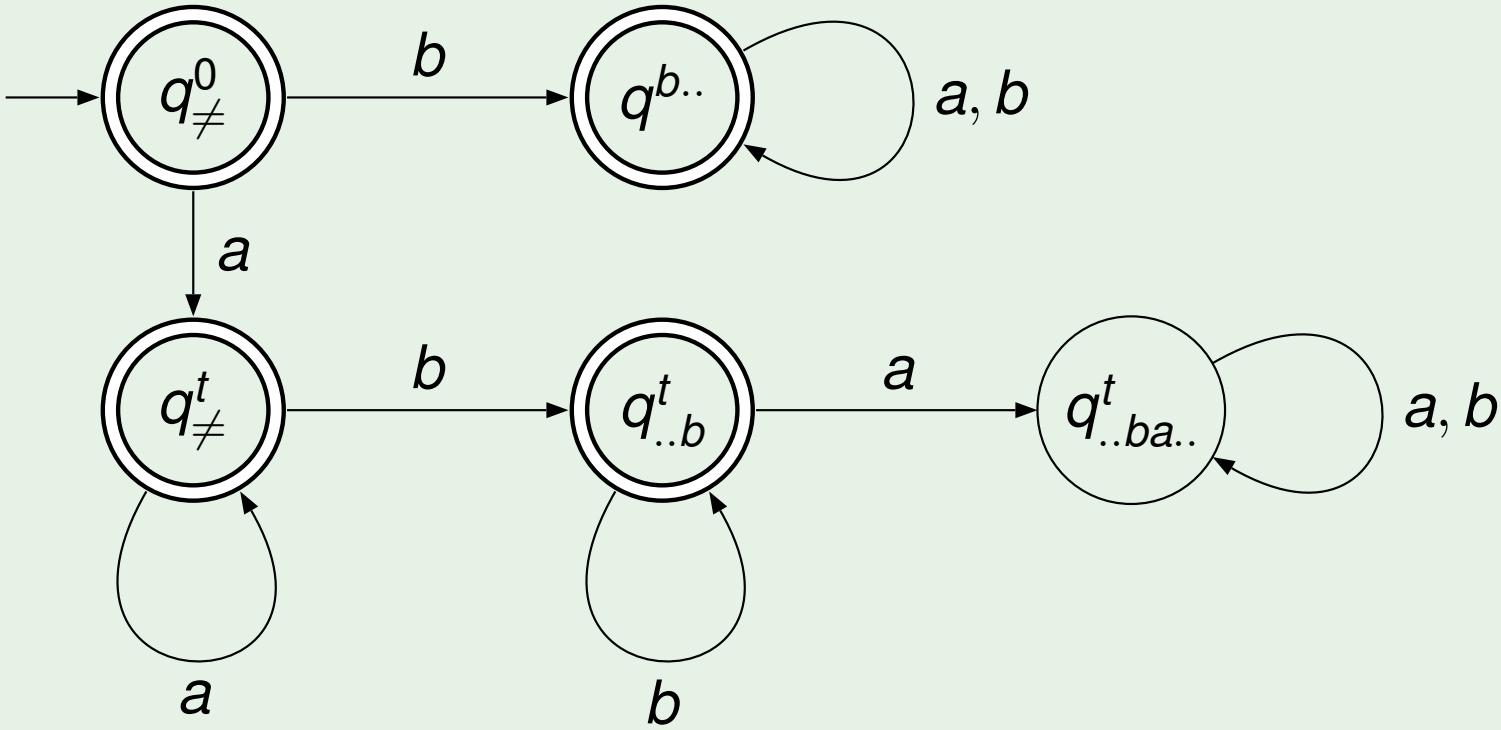
Un posible AFD:



En el AFD anterior se han obviado los 3 estados inaccesibles pero se han mantenido los 2 estados equivalentes.

Ejemplo
($\{w \in \{a,b\}^* :$ **si** w contiene ba, **entonces** empieza por $b\}$)

El AFD mínimo (obtenido aplicando el algoritmo de minimización al AFD anterior y llamando $q^{b..}$ al estado trampa final fusionado) es



EJEMPLOS ADICIONALES

- A continuación se discute en detalle el diseño de AFDs para algunos lenguajes regulares.

- Muchos de ellos incluyen combinaciones o modificaciones de algunas de las características típicas estudiadas previamente, por lo que sus diseños se basan en ejemplos discutidos antes.

$$L = \{w \in \{a, b\}^+ : |w| \text{ MOD } 3 = 0\}$$

Descripción y ejemplos:

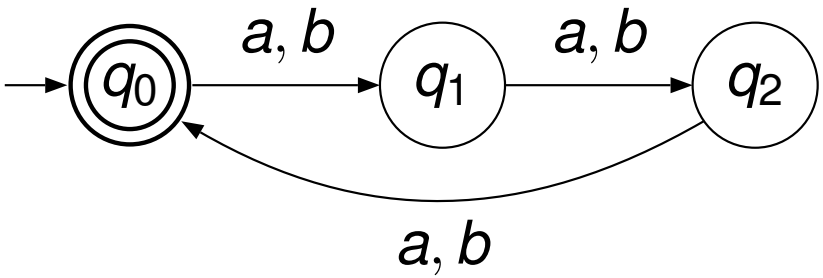
- L está formado por aquellas palabras de $\{a, b\}^+$ cuya longitud es un múltiplo de 3 $(0, 3, 6, 9, ...)$.
- La palabra vacía no está en $\{a, b\}^+$ (cierre universal *positivo* de $\{a, b\}$), por lo que no pertenece al lenguaje.
- Ejemplos de palabras que pertenecen a L:

$$a^{3n}(n \geq 1), b^{3n}(n \geq 1) \in L$$
$$aab, aba, baa, bba, aaabab, aabbba, \cdots \in L$$

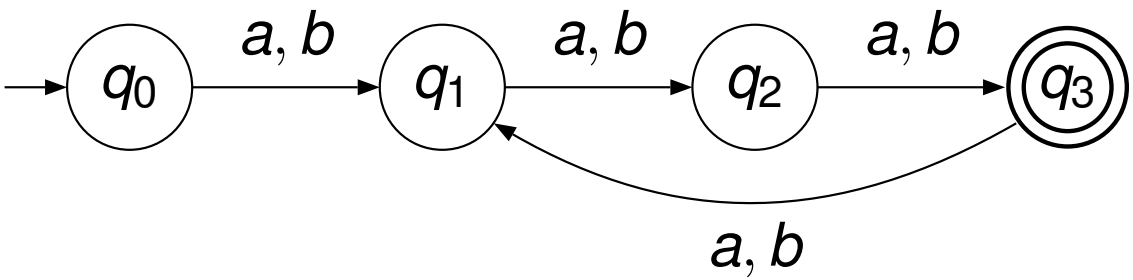
- Ejemplos de palabras que no pertenecen a L:

$$\lambda, a, b, aa, ab, bb, bbbb, baab, babab, \cdots \notin L$$

- Un AFD para $L = \{w \in \{a, b\}^* : |w| \text{ MOD } 3 = 0\}$ tendría los estados q_i, "se han leído i símbolos (módulo 3)", $i \in \{0, 1, 2\}$ (ver discusión en apartado anterior) y sería



- Para excluir a la palabra vacía habría que añadir un estado adicional, q_3, que pasaría a ser el único estado final:



$$L = \{w \in \{a, b\}^* : n_{ab}(w) \text{ MOD } 2 = 0\}$$

Descripción y ejemplos:

- L está formado por aquellas palabras de $\{a, b\}^*$ que contienen un número par de ocurrencias de la cadena ab.
- La palabra vacía contiene cero ocurrencias de ab, por lo que pertenece al lenguaje.
- Ejemplos de palabras que pertenecen a L:

$$\lambda, (ab)^{2n}, a^n, b^n (n \geq 0) \cdots \in L$$
$$ba, aa, bba, abaab, babbab, \cdots \in L$$

- Ejemplos de palabras que no pertenecen a L:

$$ab, abb, aab, bab, abbabab \cdots \notin L$$

Estados:

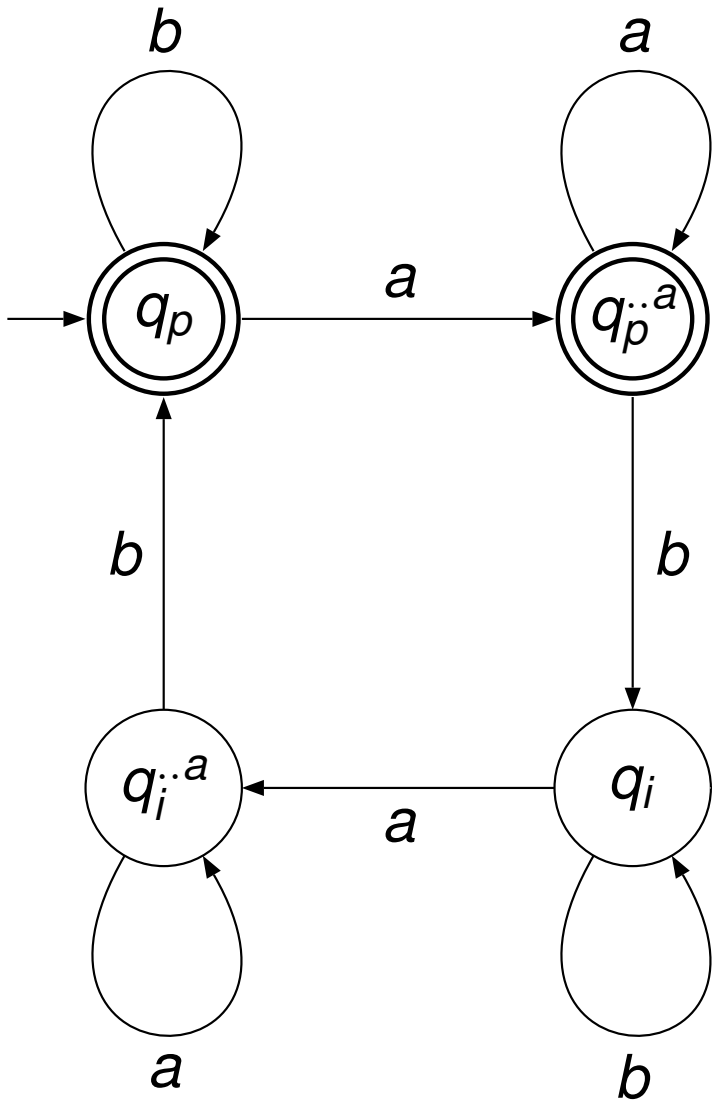
- La información a tener en cuenta es en este caso la siguiente:
  1. Si el número de ocurrencias de *ab*'s es par o impar.
  2. Si el último símbolo leído es o no una *a*, ya que, si lo es, la lectura a continuación de una *b* formaría una ocurrencia de *ab*.

- La combinación de los dos datos anteriores supone la creación de $2 \times 2 = 4$ estados: $q_p, q_p^{..a}, q_i, q_i^{..a}$, donde la *p* y la *i* indican número par/impar de *ab*'s y *..a* indica que lo último leído es una *a*:
  - $q_p =$ "número par de *ab*'s, lo último leído no es *a*".
  - $q_p^{..a} =$ "número par de *ab*'s, lo último leído es *a*".
  - $q_i =$ "número impar de *ab*'s, lo último leído no es *a*".
  - $q_i^{..a} =$ "número impar de *ab*'s, lo último leído es *a*".

- Estado inicial: q_p (incluye "no se ha leído nada": cero ocurrencias de *ab*'s, lo último leído no es una *a*).
- Estados finales: q_p y $q_p^{\cdot a}$ (ambos aseguran un número par de *ab*'s).

Transiciones:

- Si estando en q_p se lee una *b*, la situación no cambia puesto que sigue habiendo un número par de *ab*'s y lo último leído sigue sin ser una *a*. Hay que añadir por lo tanto un bucle *b* sobre q_p.
- Si en q_p se lee una *a* hay que crear el estado $q_p^{\cdot a}$, puesto que el número de *ab*'s no cambia (sigue siendo par) pero es necesario recordar que lo último leído ha sido una *a*.
- Transiciones de $q_p^{\cdot a}$: leyendo una *a* la situación no cambia (bucle *a* sobre $q_p^{\cdot a}$), mientras que con la lectura de una *b* lo último leído deja de ser una *a* y además, al producirse una nueva ocurrencia de *ab*, el número de estas pasa a ser impar (transición con *b* a q_i).
- Siguiendo con el razonamiento anterior, basado en la semántica de los estados, se llega al AFD siguiente:


b
a
a
q_p
$q_p{\cdot}a$
b
b
$q_i{\cdot}a$
q_i
a
a
b

Lenguaje $L \subseteq \{a, b\}^+$ formado por aquellas palabras tales que si contienen b's, el número de b's consecutivas siempre es 1 o estrictamente mayor que 2 (pero nunca 2)

Descripción y ejemplos:

- Dado que L está contenido en $\{a, b\}^+$ (cierre universal *positivo*), el lenguaje no contiene la palabra vacía.
- Ejemplos de palabras que pertenecen a L:

$$a^n(n \geq 1), b, b^n(n \geq 3), abab, abbba, ababbbbaaabbb \in L$$

- Ejemplos de palabras que no pertenecen a L:

$$\lambda, bb, bba, abb, ababba, ababbaaabbb \notin L$$

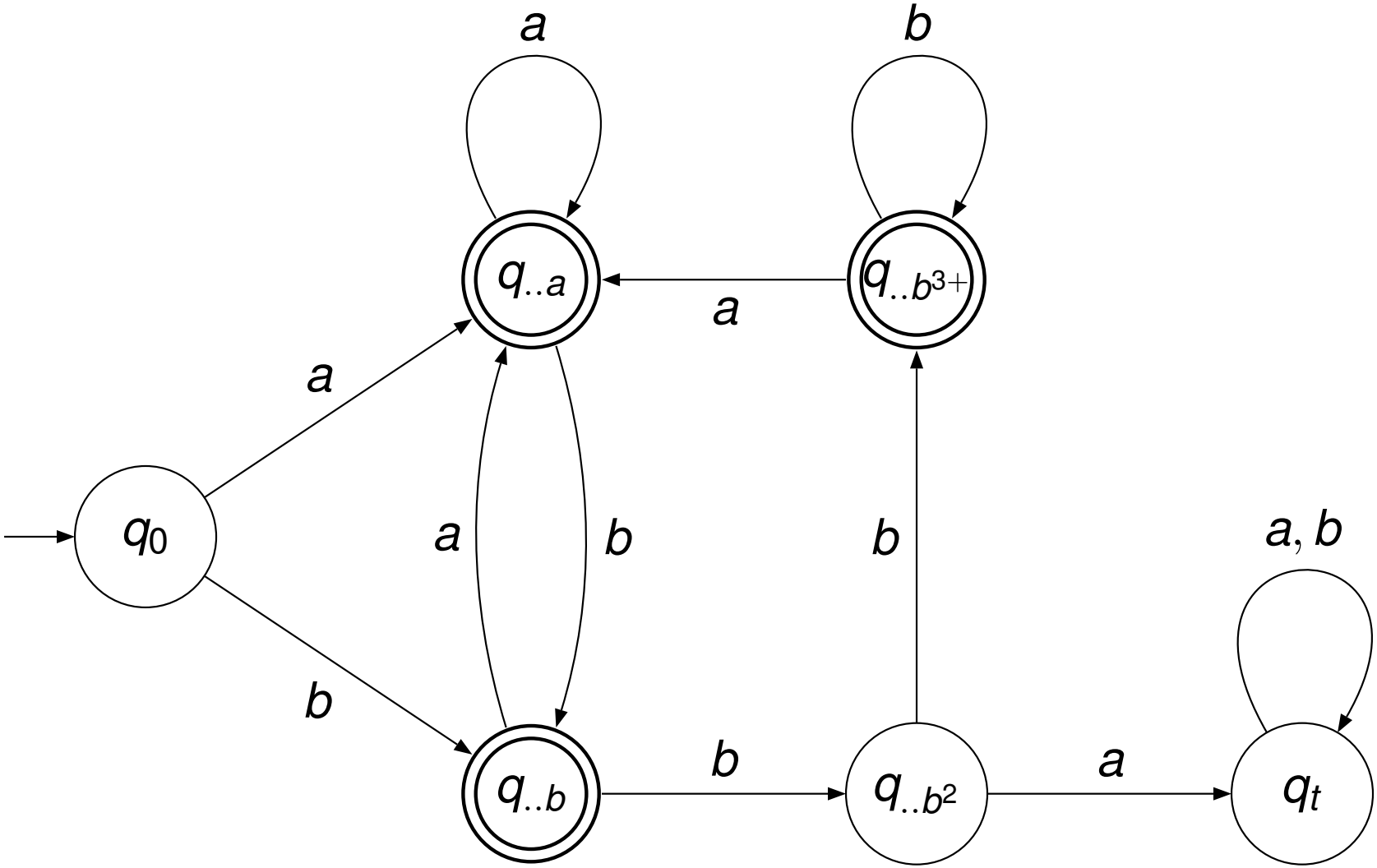
Estados:

- Para diseñar un AFD para L hay que recordar qué es lo último que se ha leído para evitar admitir dos b's consecutivas aisladas. Para ello se consideran los siguientes estados:
  - q_0: no se ha leído ningún símbolo.
  - $q_{..a}$: lo último leído es a (es necesario distinguir entre este estado y el anterior debido a que $\lambda \notin L$; si no fuese así, estos dos estados podrían fundirse en uno solo, con semántica "lo último leído no es b").
  - $q_{..b}$: lo último leído es una b (y solo una b).
  - $q_{..b^2}$: lo último leído es bb (y solo dos b's).
  - $q_{..b^{3+}}$: lo último leído es b^n ($n \geq 3$).
- Será necesario además disponer de un estado trampa (no final), q_t, al que habrá que ir en el momento en el que se compruebe la existencia de exactamente dos b's consecutivas aisladas.

- Estado inicial: q_0
- Estados finales:
  - q_0 no puede ser final puesto que $\lambda \notin L$.
  - $q_{..b^2}$ no debe ser final puesto que al llegar a este estado la palabra acaba con dos b's (y solo dos) consecutivas.
  - q_t tampoco deber ser final, puesto que se llegará a él precisamente cuando se detecten dos b's aisladas.
  - $q_{..a}$, $q_{..b}$ y $q_{..b^{3+}}$ deben ser finales.

Transiciones:

Para decidir las transiciones basta con tener en cuenta la semántica de cada uno de los estados, llegándose al AFD cuyo grafo se incluye a continuación.


a
b
$q_{..a}$
$q_{..b^{3+}}$
a
a
b
q_0
a
b
b
a, b
b
b
a
$q_{..b}$
$q_{..b^2}$
q_t

$$L = \{w \in \{a,b\}^* : n_a(w) \text{ MOD } 2 = 0 \textbf{ y } n_b(w) \text{ MOD } 3 = 0\}$$

Descripción y ejemplos:

- L está formado por aquellas palabras de $\{a,b\}^*$ que contienen un número de a's múltiplo de 2 (es decir, par: $0, 2, 4, 6, ...$) **y** un número de b's múltiplo de 3 ($0, 3, 6, 9, ...$), por lo que se trata de un ejemplo típico de intersección de dos lenguajes: $L = L_1 \cap L_2$, $L_1 = \{w \in \{a,b\}^* : n_a(w) \text{ MOD } 2 = 0\}$, $L_2 = \{w \in \{a,b\}^* : n_b(w) \text{ MOD } 3 = 0\}$.
- Ejemplos de palabras que pertenecen a L:

  λ (contiene cero a's y cero b's), $a^{2n}(n \geq 0), b^{3n}(n \geq 0) \in L$
  $ababb, babba, babab, bbbaa, abababa, aabbbaa, \cdots \in L$

- Ejemplos de palabras que no pertenecen a L:

  $$a, aaa, b, bb, bbbb, aab, bababa, \cdots \notin L$$

Estados:

- Para diseñar el AFD producto para L hay que tener en cuenta:
  1. El número de a's leídas (módulo 2), para lo que se necesitan 2 estados, q_i, con $i \in \{0, 1\}$.
  2. El número de b's leídas (módulo 3), para lo que se necesitan 3 estados q^j, con $j \in \{0, 1, 2\}$.
- La combinación de los estados anteriores supone la creación de $2 \times 3 = 6$ estados, que podrían denotarse

$$q_i^j, i \in \{0, 1\}, j \in \{0, 1, 2\},$$

  con semántica "se han leído i a's (módulo 2) y j b's (módulo 3)".
- Ningún estado es inaccesible (las dos características son independientes y por lo tanto nunca serán contradictorias).
- Estado inicial: q_0^0 (cero ocurrencias -módulo 2- de a y cero ocurrencias -módulo 3- de b).

- Estados finales: el único final es el inicial, q_0^0, puesto que es el único en el que se cumplen a la vez las dos condiciones buscadas (número par de *a*'s y número de *b*'s múltiplo de 3).
- Ningún estado garantiza que *nunca* se vaya a cumplir una de las dos características, puesto que todo depende de lo que se lea a continuación, y por lo tanto no hay estados equivalentes que se puedan fundir en un trampa no final.

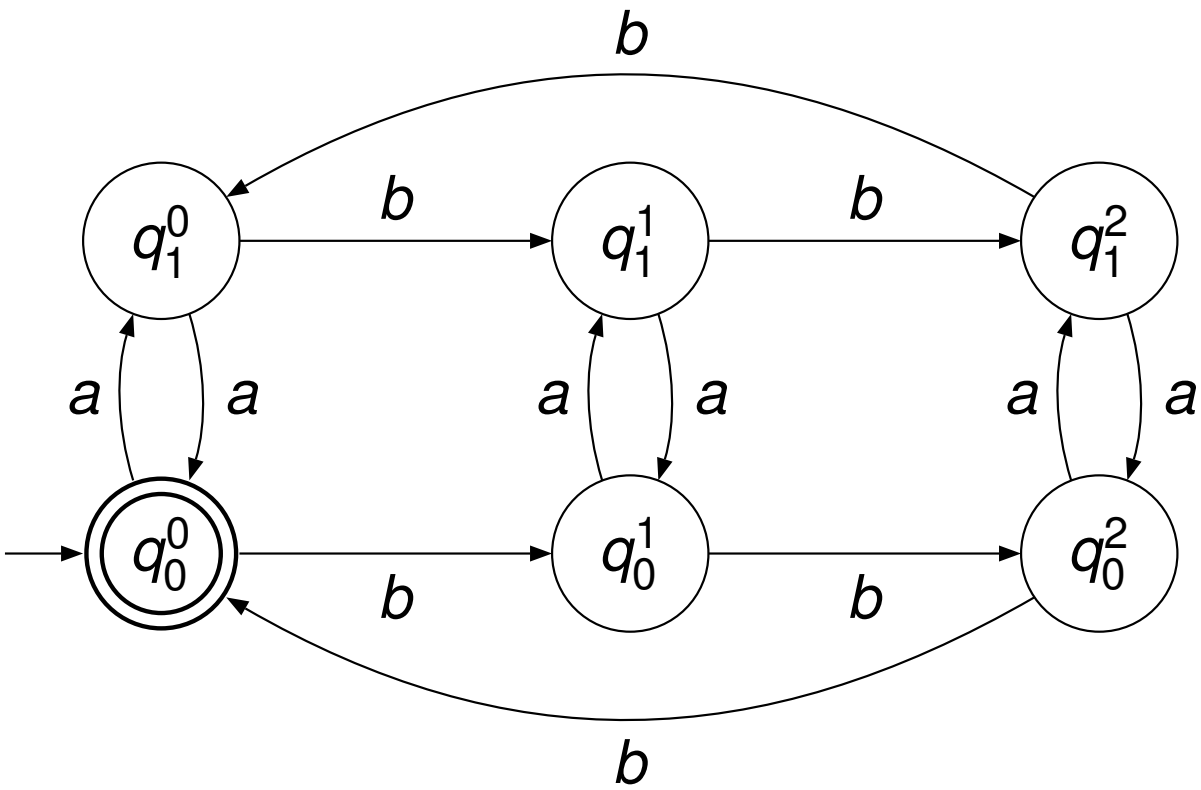
Transiciones:

- Desde el estado inicial q_0^0 con la *a* se transita al estado q_1^0 (puesto que se suma una *a*) y con la *b* a q_0^1 (se suma una *b*).
- Desde q_1^0, con la *a* se vuelve al estado q_0^0 (el número de *a*'s pasa de impar a par y el número de *b*'s no varía), y con la *b* se pasa a q_1^1 (se suma una *b*).
- Desde q_0^1, con la *a* se pasa al estado q_1^1 y con la *b* se pasa a q_0^2.
- El resto de transiciones se contruyen con el mismo razonamiento, basado en la semántica de los estados.

De todo lo anterior se deduce que un AFD para el lenguaje

$$L = \{w \in \{a, b\}^* : n_a(w) \text{ MOD } 2 = 0 \textbf{ y } n_b(w) \text{ MOD } 3 = 0\}$$

es (compruebe que es el mínimo aplicando el algoritmo de minimización)



$$L = \{w \in \{a,b\}^* : n_a(w) \text{ MOD } 2 = 0 \textbf{ o } n_b(w) \text{ MOD } 3 = 0\}$$

Descripción y ejemplos:

- L está formado por aquellas palabras de $\{a,b\}^*$ que contienen un número de a's múltiplo de 2 (es decir, par: $0, 2, 4, 6, ...$) **o** un número de b's múltiplo de 3 ($0, 3, 6, 9, ...$), por lo que se trata de un ejemplo típico de unión de dos lenguajes: $L = L_1 \cup L_2$,
  $L_1 = \{w \in \{a,b\}^* : n_a(w) \text{ MOD } 2 = 0\}$,
  $L_2 = \{w \in \{a,b\}^* : n_b(w) \text{ MOD } 3 = 0\}$.
- Ejemplos de palabras que pertenecen a L:

$$\lambda, a^n(n \geq 0), b^n(n \geq 0), aba, babb, ababb, aababa, ... \in L$$

- Ejemplos de palabras que no pertenecen a L:

$$ab, abb, abaa, ababa, a^{3n}bb(n > 0), b^{5m}a^{3n}(m, n > 0), ... \notin L$$

Estados y transiciones:

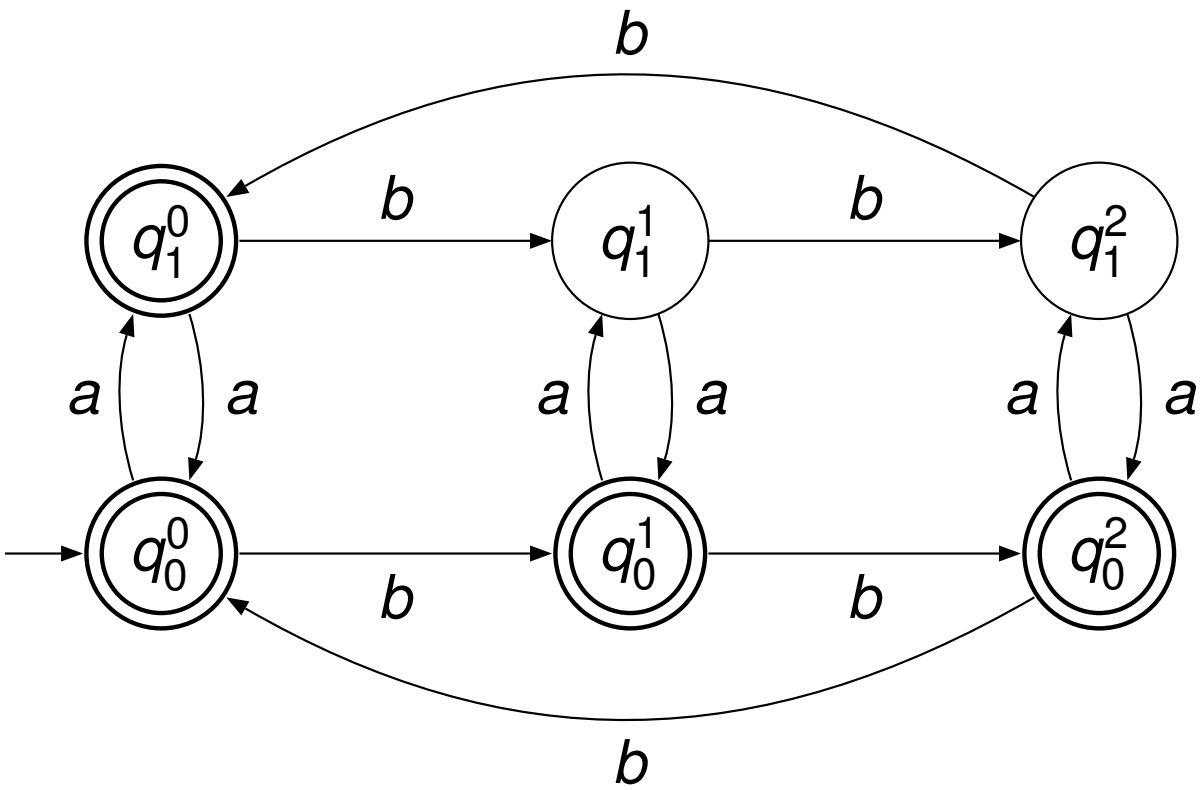
Atendiendo a lo explicado para el caso general de lenguajes que resultan ser la combinación de otros dos, un AFD para este lenguaje se construye exactamente igual que el del ejemplo anterior (que era la intersección de los dos mismos lenguajes), salvo en lo que respecta a:

1. Los *estados finales*, que ahora serán todos aquellos en los que al menos una de sus dos componentes se corresponda con un estado final de los autómatas para L_1 o L_2, y por lo tanto serán finales q_0^0, q_0^1, q_0^2 y q_1^0.
2. Los posibles *estados equivalentes entre sí*, pero no los hay: en particular, ninguno de los estados garantiza el cumplimiento de alguna de las dos características, dado que todo depende de lo que se lea a continuación, por lo que no hay estados equivalentes que se puedan fundir en un único estado trampa final.

De todo lo anterior se deduce que un AFD para el lenguaje

$$L = \{w \in \{a, b\}^* : n_a(w) \text{ MOD } 2 = 0 \textbf{ o } n_b(w) \text{ MOD } 3 = 0\}$$

es (compruebe que es el mínimo aplicando el algoritmo de minimización)



$L = \{w \in \{a, b, c\}^* : n_b(w) \text{ MOD } 2 = 1, w \text{ no acaba en } ac\}$

Descripción y ejemplos:

- L está formado por aquellas palabras de $\{a, b, c\}^*$ que contienen un número impar de b's y que además no acaban en ac, por lo que se trata nuevamente de un ejemplo claro de intersección de lenguajes.
- La palabra vacía no pertenece al lenguaje, puesto que, aunque no acaba en ac, contiene un número par (cero) de b's.
- Ejemplos de palabras que pertenecen a L:

$$b^{2n+1}, bca^n cc, bc^n bba(n \geq 0), \cdots \in L$$
$$ba, bca, bcacbaab, babacb, \cdots \in L$$

- Ejemplos de palabras que no pertenecen a L:

$$b^{2n}(n \geq 0), bac, ac, acbaab, bcbac \cdots \notin L$$

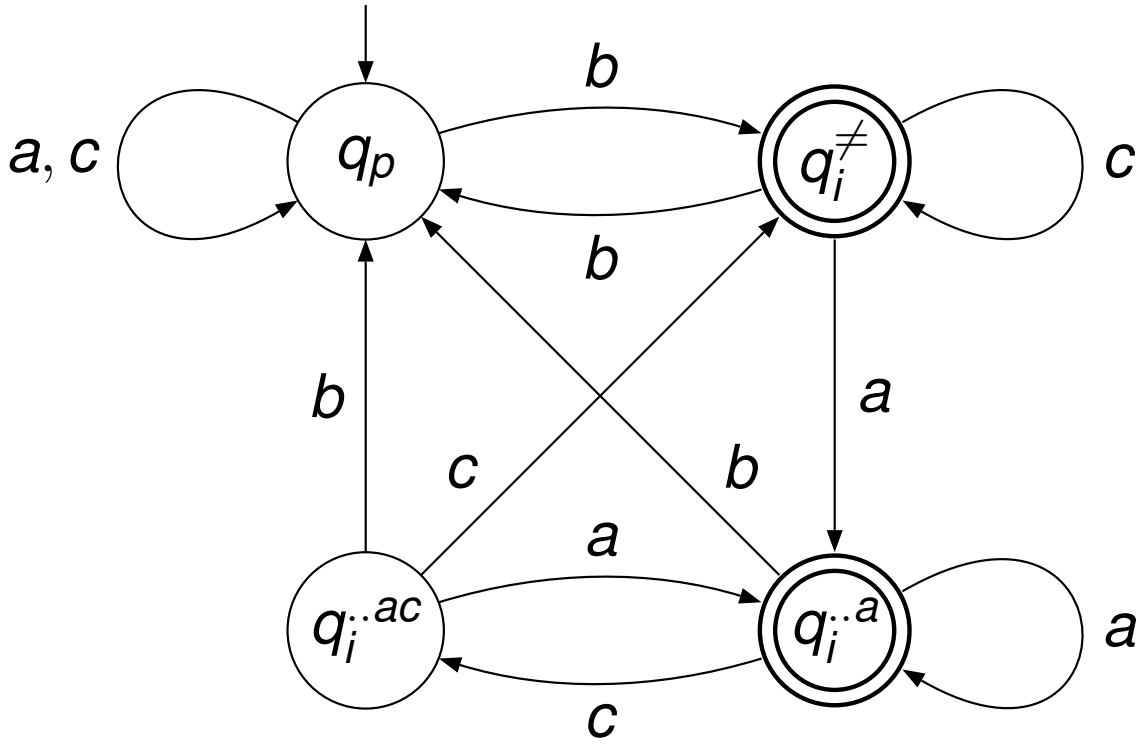
Estados:

- La información a tener en cuenta es en este caso la siguiente:
  1. Si el número de ocurrencias de *b*'s es par o impar, para lo que se necesitan dos estados, q_p y q_i.
  2. Si lo último leído es *a*, *ac* o cualquier otra cosa distinta a ambas: $q^{..a}$, $q^{..ac}$, $q^{\neq}$.
- La combinación de los dos datos anteriores supondría la creación de $2 \times 3 = 6$ estados. Sin embargo, todos los estados conteniendo q_p son equivalentes, ya que si el número de *b*'s es par, para aceptar será necesario leer al menos una *b* más, por lo que lo último leído es indiferente.
- Los estados necesarios serían por lo tanto solo 4 (fusionando los tres q_p^{algo} en q_p):
  - q_p = "número par de *b*'s" (estado inicial).
  - $q_i^{..a}$ = "número impar de *b*'s, lo último leído es *a*" (final).
  - $q_i^{..ac}$ = "número impar de *b*'s, lo último leído es *ac*".
  - $q_i^{\neq}$ = "número impar de *b*'s, lo último leído no es ni *a* ni *ac*" (final).

De todo lo anterior se deduce que un AFD para el lenguaje

$$L = \{w \in \{a, b, c\}^* : n_b(w) \text{ MOD } 2 = 1, w \text{ no acaba en } ac\}$$

es (compruebe que es el mínimo aplicando el algoritmo de minimización)



$L = \{a^n x : n > 0, x \in \{a, b, c\}^*, x \text{ no contiene } ccc\}$

Descripción y ejemplos:

- Todas las palabras deben empezar necesariamente por a (puesto que es $n > 0$), lo cual excluye, en particular, a la palabra vacía.
- Después de la a's iniciales, se admite cualquier palabra que no contenga ccc.
- Ejemplos de palabras que pertenecen a L:

$$a^n (n \geq 1), ab, ac, accb, aacbcc \in L$$

- Ejemplos de palabras que no pertenecen a L:

$$\lambda, b, c, bac, bacccca, acccc, abccca, abcacccb \notin L$$

Analizando el lenguaje dado con más detenimiento, se puede observar lo siguiente:

- La n es prescindible, es decir, bastaría con escribir $L = \{ax : ...\}$, puesto que nada impide a la x posterior empezar con tantas a's como se requieran.
- La condición "no contiene ccc" es aplicable no solo al trozo x sino a cualquier palabra de L, puesto que no afecta a la a previa (otra cosa sería que en lugar de empezar por a se exigiese empezar por c).
- Como consecuencia de lo anterior, el lenguaje L puede verse como $L = L_1 \cap L_2$, siendo

$$L_1 = \{ax : x \in \{a, b, c\}^*\}$$
$$L_2 = \{w \in \{a, b, c\}^* : w \text{ no contiene } ccc\}$$

Estados:

Para diseñar un AFD para $L = L_1 \cap L_2$ hay por lo tanto que controlar que las palabras empiecen por *a* y que no contengan la subcadena *ccc*. Las dos características anteriores han sido discutidas, por separado, en apartados anteriores:

- Para construir un AFD para L_1 se necesitan los estados q_0 (no se ha leído nada), $q_{a..}$ (empieza por *a*) y q_t (ninguna de las anteriores).
- Para construir un AFD para L_2 se necesitan los estados $q^{..c}$ (lo último es una *c* y solo una), $q^{..cc}$ (lo último es *cc* y solo dos), $q^{..ccc..}$ (contiene *ccc*) y $q^{\neq}$ (ninguna de las anteriores).
- Para obtener un AFD para *L* se puede construir el AFD producto de los dos anteriores, formado por $3 \times 4 = 12$ estados.

De los 12 posibles estados, solo 5 de ellos serán de utilidad. En efecto (compruebe lo que sigue construyendo el AFD completo y aplicándole posteriormente el algoritmo de minimización):

- Los estados $q_0^{..c}$, $q_0^{..cc}$ y $q_0^{..ccc..}$ serán claramente inaccesibles puesto que presentan una semántica contradictoria (recuerde que q_0 indica que no se ha leído todavía nada).
- Los estados que aseguren que alguna de las dos condiciones *nunca* se podrá cumplir, independientemente de lo que venga después, serán todos ellos equivalentes entre sí y podrán fundirse en un único estado, no final y trampa. En este grupo están los 4 estados que contienen q_t (puesto que una vez en ellos es imposible que la palabra procesada empiece por a) y el único que queda construido a partir de $q^{..ccc..}$, que es $q_{a..}^{..ccc..}$ (de los otros dos uno es inaccesible, $q_0^{..ccc..}$, y el otro, $q_t^{..ccc..}$, ya está incluido en el grupo por proceder también de q_t).

Quedarían por lo tanto los siguientes estados:

- $q_0^{\neq}$: no se ha leído ningún símbolo, que será el estado *inicial* y será *no final*.
- $q_{a..}^{\neq}$: la palabra empieza por *a* y lo último leído no es *c* (*final*).
- $q_{a..}^{:c}$: la palabra empieza por *a* y lo último leído es una *c* (y solo una *c*) (*final*).
- $q_{a..}^{:cc}$: la palabra empieza por *a* y lo último leído es *cc* (y solo dos *c*'s) (*final*).
- q_t: estado trampa válido tanto para las palabras que empiezan por algo distinto a *a* como para aquellas que contienen *ccc* (*no final*).

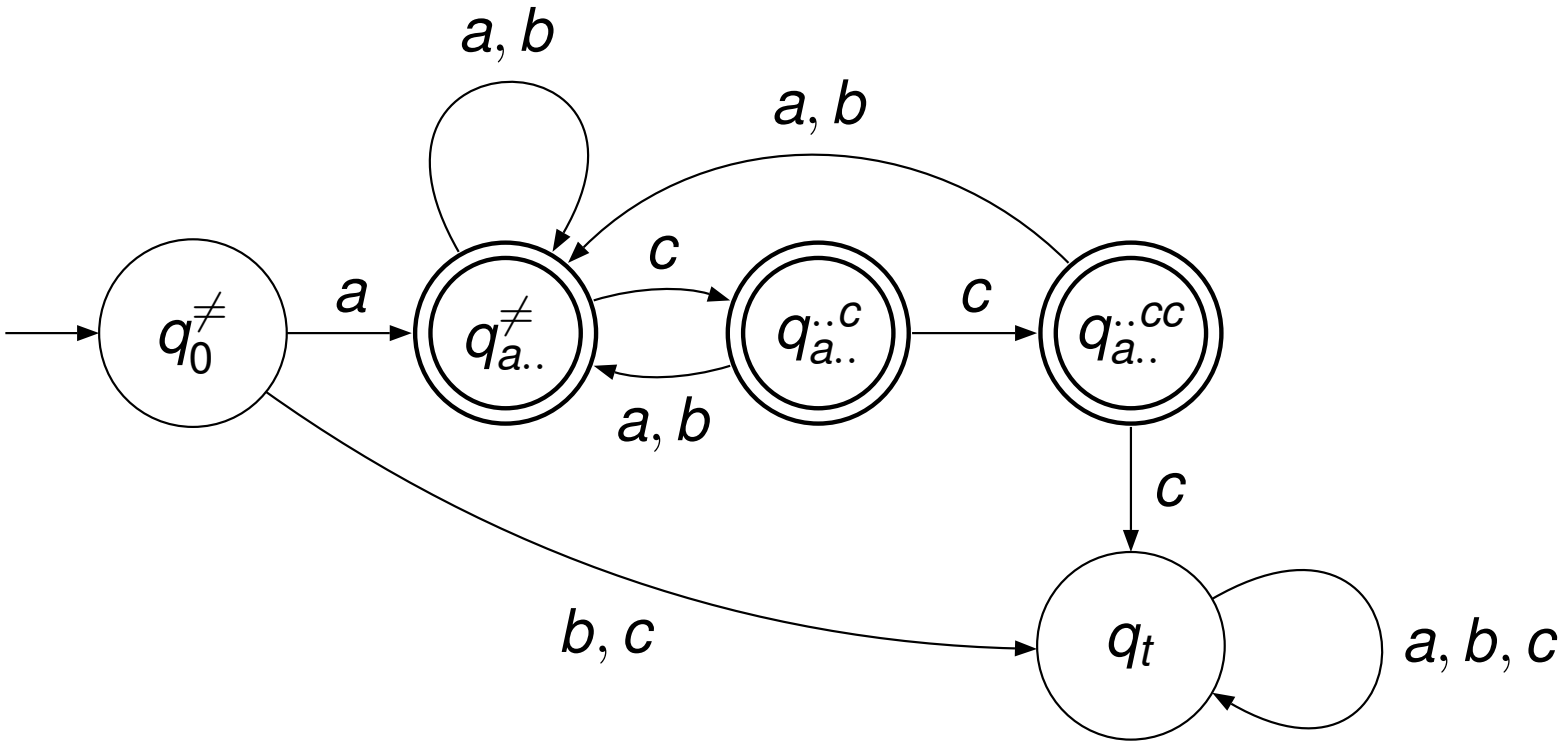
Transiciones:

Para decidir las transiciones basta con tener en cuenta la semántica de cada uno de los estados, llegándose al AFD cuyo grafo se incluye a continuación (y que es el mínimo).

De todo lo anterior se deduce que un AFD para el lenguaje

$$L = \{a^n x : n > 0, x \in \{a, b, c\}^*, x \text{ no contiene } ccc\}$$

es (compruebe que es el mínimo aplicando el algoritmo de minimización)



$L = \{awa \ : \ w \in \{a, b, c\}^*\}$

Descripción y ejemplos:

- Todas las palabras deben empezar y acabar necesariamente por a, lo cual excluye, en particular, a la palabra vacía.
- La a inicial y la final deben de ser distintas, por lo que la palabra a no pertenece a L.
- No hay restricciones entre las dos a's (puede haber cualquier cantidad de símbolos del alfabeto $\{a, b, c\}$, incluido ninguno).
- Ejemplos de palabras que pertenecen a L:

$$aa, aaa, aba, aca, aabca, acbaa, abba, \ldots \in L$$

- Ejemplos de palabras que no pertenecen a L:

$$\lambda, a, b, c, ab, ba, ac, ca, abb, abc, cba, \ldots \notin L$$

Estados:

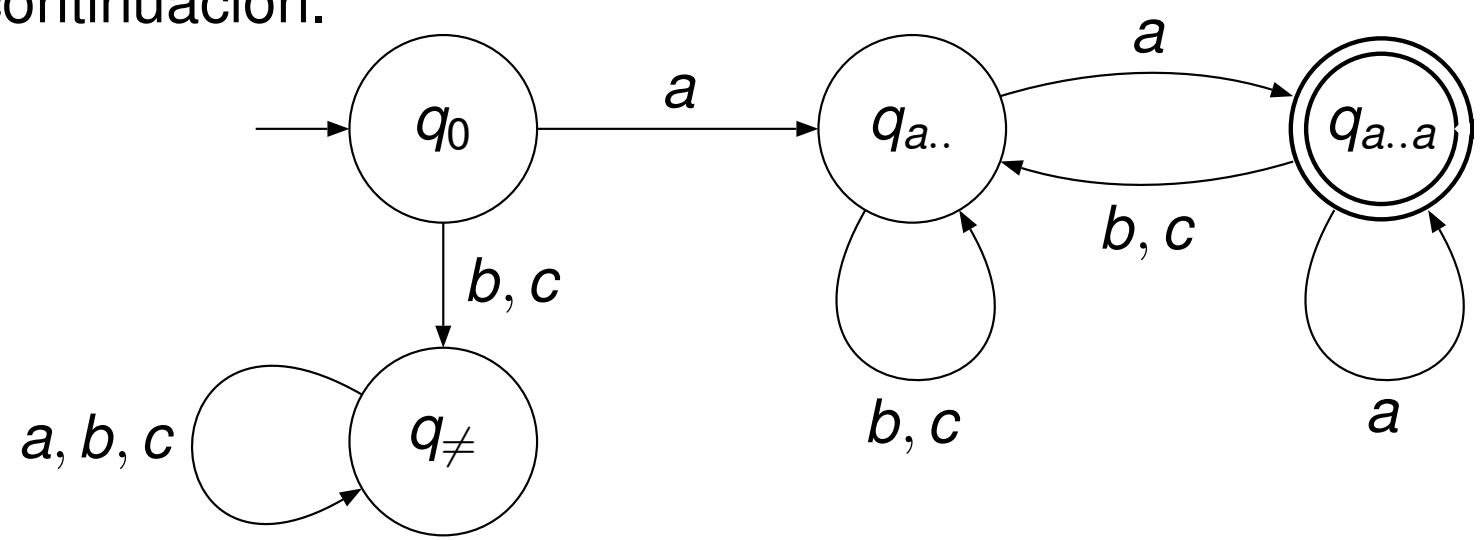
Para diseñar un AFD para L hay que controlar que las palabras empiecen y terminen por a (siendo ambas a's distintas entre sí). Las dos características anteriores han sido discutidas, por separado, en apartados anteriores. Se trata ahora por lo tanto de combinarlas, para lo que se consideran los siguientes estados:

- q_0: no se ha leído ningún símbolo.
- $q_{a..}$: la palabra empieza por a pero no termina con otra a distinta.
- $q_{a..a}$: la palabra empieza por a y lo último leído es una a distinta a la inicial.
- $q_{\neq}$: ninguna de las anteriores (la palabra empieza por algo distinto a a).

- Estado inicial: q_0
- Estados finales:
  - q_0 no puede ser final puesto que $\lambda \notin L$.
  - Del resto, de acuerdo con su semántica, el único que debe ser final es $q_{a..a}$.

Transiciones:

Para decidir las transiciones basta con tener en cuenta la semántica de cada uno de los estados, llegándose al AFD cuyo grafo se incluye a continuación.



Observación

- El ejemplo anterior se ha resuelto razonando directamente sobre las características significativas que permiten -o permitirán decidir más adelante- si la palabra que se está procesando debe aceptarse o rechazarse.
- Podría haberse resuelto, equivalentemente, viendo el lenguaje dado como la *intersección* de otros y aplicando la técnica mencionada previamente para esos casos. En efecto, si se denota $L_1 = \{ax : x \in \{a, b, c\}^*\}$ y $L_2 = \{xa : x \in \{a, b, c\}^*\}$, resulta

$$L = (L_1 \cap L_2) - \{a\} = L_1 \cap L_2 \cap \overline{\{a\}}$$

lo cual daría lugar a un AFD con $3 \times 2 \times 3 = 18$ estados, resultando muchos de ellos claramente inaccesibles y otros equivalentes entre sí.

$L = \{w \in \{a, b\}^* \ : \ w$ empieza por b y si $(n_a(w)$ MOD $2 = 0)$ entonces w termina por $b\}$

Descripción y ejemplos:

- Todas las palabras deben empezar necesariamente por b, lo cual excluye, en particular, a la palabra vacía.
- Si las palabras tienen un número par de a's, entonces deben acabar por b.
- No se menciona el caso de las palabras que contienen un número impar de a's, por lo que no hay restricciones sobre su terminación.
- Ejemplos de palabras que pertenecen a L:

$$b, bb, ba, bab, baaba, baabab, babab, baabaab, \ldots \cdots \in L$$

- Ejemplos de palabras que no pertenecen a L:

$$\lambda, a, aa, ab, aab, baa, baba, bbabbaaa, \cdots \notin L$$

- L puede verse como una intersección de lenguajes, $L = L_1 \cap L_2$, siendo

$$L_1 = \{w \in \{a,b\}^* \ : \ w \text{ empieza por } b\}$$
$$L_2 = \{w \in \{a,b\}^* \ : \ (n_a(w) \text{ MOD } 2 = 0) \rightarrow (w \text{ termina por } b)\}$$

- Dado que $p \rightarrow q \equiv \neg p \vee q$, L_2, a su vez, puede verse como una unión de lenguajes, $L_2 = L_{21} \cup L_{22}$, siendo

$$L_{21} = \{w \in \{a,b\}^* \ : \ n_a(w) \text{ MOD } 2 \neq 0\}$$
$$L_{22} = \{w \in \{a,b\}^* \ : \ w \text{ termina por } b\}$$

- En definitiva:

$$L = L_1 \cap \Big(L_{21} \cup L_{22} \Big)$$

y se pueden aplicar las técnicas para el diseño de autómatas producto discutidas con anterioridad.

Estados:

- Para construir un AFD para L_1 se necesitan los estados q_0 (no se ha leído nada), $q_{b..}$ (empieza por b) y q_t (ninguna de las anteriores).
- Para construir un AFD para L_2, al tratarse de una unión, hay que analizar qué estados son necesarios para cada una de sus componentes y combinarlos:
  - Para L_{21} se necesitan los estados q^{pa} (número par de a's) y q^{ia} (número impar de a's).
  - Para L_{22} se necesitan los estados $q^{..b}$ (lo último es b) y $q^{\neq}$ (lo último no es b).
  - Para $L_2 = L_{21} \cup L_{22}$ se obtienen por lo tanto los $2 \times 2 = 4$ estados $q^{j,k}$ con $j \in \{pa, ia\}$ y $k \in \{..b, \neq\}$, pero los dos con $j = ia$ transitarían a los mismos estados tanto con a (ambos a $q^{pa,\neq}$) como con b (ambos a $q^{ia,..b}$), por lo que son equivalentes entre sí. Llamando q^{ia} a la fusión de estos dos estados, quedan 3 estados para L_2: $q^{pa,\neq}$, $q^{pa,..b}$ y q^{ia}.

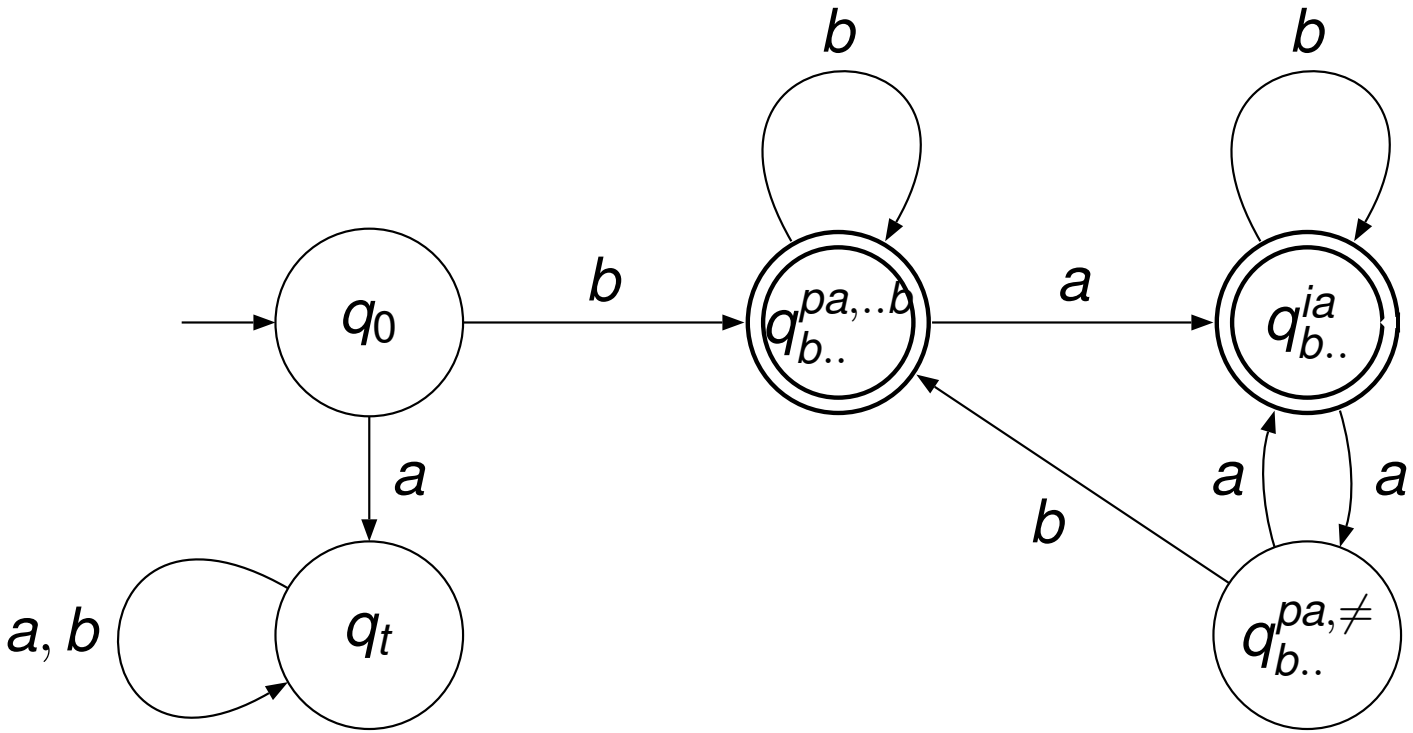
Para obtener un AFD para L se construye el AFD producto de los dos anteriores, formado por $3 \times 3 = 9$ estados, de los cuales solo 5 serán de utilidad. En efecto (compruebe lo que sigue construyendo el AFD completo y aplicándole posteriormente el algoritmo de minimización):

- El estado q_0 de L_1, que indica que no se ha leído nada, solo es compatible con el estado $q^{pa,\neq}$ de L_2 (los otros dos estados de L_2 presentan una semántica contradictoria con q_0). Sea $\mathbf{q_0}$ este estado.
- El estado q_t de L_1 indica que la palabra empieza por algo distinto de b, asegurando así que la primera condición necesaria para pertenecer a L *nunca* se podrá cumplir, independientemente de lo que venga después. En consecuencia, los estados resultantes de combinarlo con los tres de L_2 serán equivalentes entre sí y podrán fundirse en un único estado, no final y trampa. Sea $\mathbf{q_t}$ este estado.

- Estado inicial: q_0
- Estados finales: aquellos que aseguren la primera condición ($b..$) y *alguna* de las otras dos: *ia* o bien $..b$.

Transiciones:

Basta con tener en cuenta la semántica de los estados, obteniéndose el AFD cuyo grafo se incluye a continuación:



BIBLIOGRAFÍA

- Este material docente está basado en fuentes diversas, en particular en los libros y recursos que se citan a continuación, de los que proceden muchos de los ejercicios que se discuten aquí.

- Se nutre también del trabajo de nuestros antiguos compañeros Jorge Martín Corujo, Holger Billhardt y José Miguel Buenaposada, a los que expresamos desde aquí nuestro agradecimiento.

- P. Linz, An Introduction to Formal Languages and Automata. Jones and Barlett Publishers, 4th ed., 2006.
- J.E. Hopcroft, R. Motwani, J.D. Ullman, Introducción a la Teoría de Autómatas, Lenguajes y Computación, Addison-Wesley Iberoamericana, 3ª edición, 2007.
- M. Alfonseca, J. Sancho, M. Martínez Orga, Teoría de Lenguajes, Gramáticas y Autómatas, Ediciones RAEC 1997.
- P. Isasi, P. Martínez, D. Borrajo. Lenguajes, gramáticas y autómatas. Un enfoque práctico. Addison-Wesley, 1997.
- M. Alfonseca, E. Alfonseca, A. Ortega, Teoría de Autómatas y Lenguajes formales, Mc. Graw Hill DL, 2007.

DISTRIBUCIÓN

TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES

(TALF)

GUÍA BÁSICA PARA EL DISEÑO DE AUTÓMATAS FINITOS NO DETERMINISTAS (AFNDs)

Ana Pradera, Juan Manuel Serrano
Sergio Saugar, Mónica Robledo
César Alfaro, Javier Gómez
María Teresa González de Lena, Gema Gutiérrez

Noviembre de 2024

INTRODUCCIÓN

- El objetivo de estas notas es proporcionar, por medio de ejemplos, algunas pautas básicas para el diseño de Autómatas Finitos No Deterministas (AFNDs).
- Dado un lenguaje regular L, el problema consiste en construir un AFND A que reconozca *exactamente* ese lenguaje (es decir, acepte *todas* las palabras de L y rechace *cualquier* palabra que no pertenezca a L), de forma que se tenga $L(A) = L$, donde $L(A)$ es el lenguaje reconocido por A.
- Los AFNDs, aunque presentan algunas características propias, son similares a los AFDs (Autómatas Finitos Deterministas), por lo que su diseño debe seguir las mismas pautas (véase la "*Guía básica para el diseño de AFDs*" de esta misma colección) pero sacando partido de sus particularidades.

- El material que se presenta a continuación es fruto de la impartición de la materia *"Teoría de Autómatas y Lenguajes Formales"* en distintos grados universitarios del ámbito de la Informática.
- Se dirige a estudiantes tales que, además de manejar los conceptos básicos relativos a lenguajes y autómatas, conocen con cierta profundidad las características y el funcionamiento de los *autómatas finitos*, tanto deterministas como no deterministas, así como las técnicas básicas de *diseño* de los primeros.
- En lo que sigue:
  - Se recuerdan en primer lugar los **conceptos fundamentales** relativos a AFNDs.
  - A continuación se detallan algunas de las **técnicas específicas para su diseño**, ilustradas con ejemplos.
  - Se termina con algunos **ejemplos adicionales**.

AFNDs: CONCEPTOS FUNDAMENTALES

- En este apartado se exponen brevemente las características y la representación de AFNDs, así como una descripción informal de su funcionamiento (consulte la bibliografía para más detalles).
- Los AFNDs son similares a los AFDs salvo en lo que respecta a su comportamiento no determinista: el cambio de un estado interno a otro NO está unívocamente determinado por el estado actual y el siguiente símbolo pendiente de leer en el dispositivo de entrada.
- Como se recuerda a continuación, el no determinismo se expresa modificando la definición de la *función de transición* de estos autómatas, lo cual se traduce en tres características fundamentales que diferencian a los AFNDs de los AFDs.

Definición (Autómata Finito No Determinista, AFND)

Un Autómata Finito No Determinista (AFND) es una quíntupla (Q, Σ, f, q_0, F) donde:

- Q es un conjunto no vacío y finito de elementos denominados estados internos.
- Σ es un alfabeto denominado alfabeto de entrada.
- $f : Q \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q)$ es una aplicación denominada función de transición, donde $\mathcal{P}(Q)$, también denotado mediante 2^Q, es el conjunto potencia o conjunto de partes de Q (conjunto formado por todos los posibles subconjuntos de Q).
- $q_0 \in Q$ es un estado distinguido denominado estado inicial.
- $F \subseteq Q$ es un subconjunto de estados denominados estados finales.

Características principales de un AFND

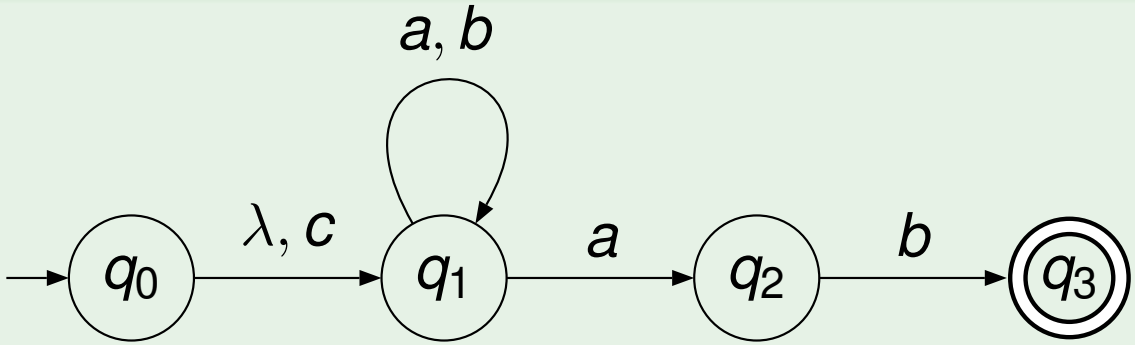
1. Los pares <estado, símbolo de entrada> pueden tener asociado más de un estado siguiente. Por ejemplo, puede ser $f(q, a) = \{p, q, r\}$, indicando que desde el estado q y leyendo el símbolo a se puede pasar a cualquiera de los estados p, q o r.
2. Los estados pueden transitar por sí mismos, sin necesidad de leer ningún símbolo de la cinta de entrada. Estas transiciones se llaman transiciones λ (*lambda*). Por ejemplo, puede ser $f(q, \lambda) = \{r, s\}$, indicando que desde el estado q se puede pasar tanto a r como a s sin consumir ningún símbolo de entrada.
3. Los estados no tienen por qué tener definido su comportamiento ante cualquier símbolo de entrada. Por ejemplo, puede ser $f(q, a) = \emptyset = \{\}$, indicando que no hay ninguna transición definida para el estado q cuando se lee el símbolo de entrada a.

Representación de AFNDs

Los AFNDs se pueden representar, de forma equivalente, mediante *tablas* o mediante *grafos (dirigidos)*. En lo que sigue usaremos grafos, en los que:

- Cada uno de los estados de Q constituye un nodo del grafo, que se representa mediante el nombre del estado rodeado de un círculo.
- El estado inicial se destaca señalándolo con una flecha incidente, $\rightarrow$, sin origen.
- Los estados finales se marcan rodeándolos mediante un círculo adicional.
- Cada transición de la forma $f(q_i, a) = \{q_{i1}, \ldots, q_{in_i}\}$, $i, i1, \ldots, in_i \in \{0, \ldots, n\}$, $a \in \Sigma \cup \{\lambda\}$, da lugar a n_i arcos dirigidos desde el nodo q_i hacia cada uno de los nodos $q_{i1}, \ldots, q_{in_i}$, etiquetados todos ellos con el símbolo a.

Ejemplo (1/4)



Este AFND tiene cuatro estados: q_0 es el inicial y q_3 es el único final. Su alfabeto de entrada es $\Sigma = \{a, b, c\}$, y presenta las tres ▸ características específicas de los AFNDs (no permitidas en AFDs):

1. El estado q_1, mediante el símbolo a, puede tanto permanecer en el propio q_1 como transitar a q_2: $f(q_1, a) = \{q_1, q_2\}$.
2. El estado q_0 dispone de una transición λ mediante la que puede transitar a q_1 sin consumir ningún símbolo: $f(q_0, \lambda) = \{q_1\}$.
3. Hay estados (en este caso, todos ellos) a los que les faltan transiciones: a q_0 le faltan las transiciones con a y b, a q_1 le falta la transición c, a q_2 le faltan a y c y a q_3 le faltan todas.

Funcionamiento de un AFND (descripción informal)

Los AFNDs aceptan o rechazan palabras construidas sobre su alfabeto de entrada Σ (una palabra sobre Σ es una secuencia finita de cero o más símbolos de Σ) procesándolas como sigue:

- Partiendo del estado inicial, se intentan todas las posibles formas de procesar la palabra de entrada, empezando por la izquierda y avanzando símbolo a símbolo, teniendo en cuenta que las transiciones λ pueden realizarse tantas veces como se quiera a lo largo del proceso (sin necesidad de consumir ningún símbolo de la entrada).
- La respuesta del AFND será afirmativa cuando *alguno* de los intentos anteriores sea capaz de procesar la palabra de entrada completa y además, tras hacerlo, la máquina quede en un estado final. Será negativa en cualquier otro caso.

Observación

- Para que la respuesta de un AFND sea afirmativa basta con que *uno* de los intentos de procesamiento de la palabra de entrada sea exitoso, es decir, no importa que otros intentos no lleguen a procesar la palabra completa o que, aunque sí puedan hacerlo, acaben en un estado no final.
- Por lo tanto, la respuesta de un AFND será negativa sólo cuando resulte que *todos* los intentos de procesamiento de la palabra fracasan, bien porque no pueden procesar la palabra completa o bien porque, aunque son capaces de procesarla entera, terminan en un estado no final.

Ejemplo (2/4)

Ejemplos de palabras *rechazadas* por el AFND anterior:

- Rechaza palabras como *c* o *cb*: solo puede procesarlas de una forma, $q_0 \xrightarrow{c} q_1$ y $q_0 \xrightarrow{c} q_1 \xrightarrow{b} q_1$, respectivamente, terminando en ambos casos en q_1, estado que no es final.
- Rechaza palabras como *a* o *cba*, que pueden procesarse de varias formas distintas pero siempre acaban en un estado no final (q_1 o q_2):
  - Para procesar *a*, se puede hacer $q_0 \xrightarrow{\lambda} q_1 \xrightarrow{a} q_1$ o bien $q_0 \xrightarrow{\lambda} q_1 \xrightarrow{a} q_2$.
  - Para procesar *cba*, se puede hacer $q_0 \xrightarrow{c} q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_1$ o bien $q_0 \xrightarrow{c} q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_2$.
- Rechaza palabras como *cc*, *ac* o *bac*, en estos casos porque no es capaz de procesarlas de forma completa (llega hasta q_1 o q_2 pero ninguno de los dos puede procesar el siguiente símbolo *c*).

Ejemplo (3/4)

Ejemplos de palabras *aceptadas* por el AFND anterior:

- Acepta palabras como *ab*, *cab*, *aab* o *cabab*, ya que para todas ellas encuentra una forma de procesar la palabra que termina en el estado final q_3 (aunque otros posibles caminos no puedan procesar todos los símbolos o acaben en estado no final).
- Por ejemplo, existen dos posibilidades para procesar *ab*: $q_0 \xrightarrow{\lambda} q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_1$ y $q_0 \xrightarrow{\lambda} q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_3$, y como la segunda acaba en estado final, el AFND la acepta.

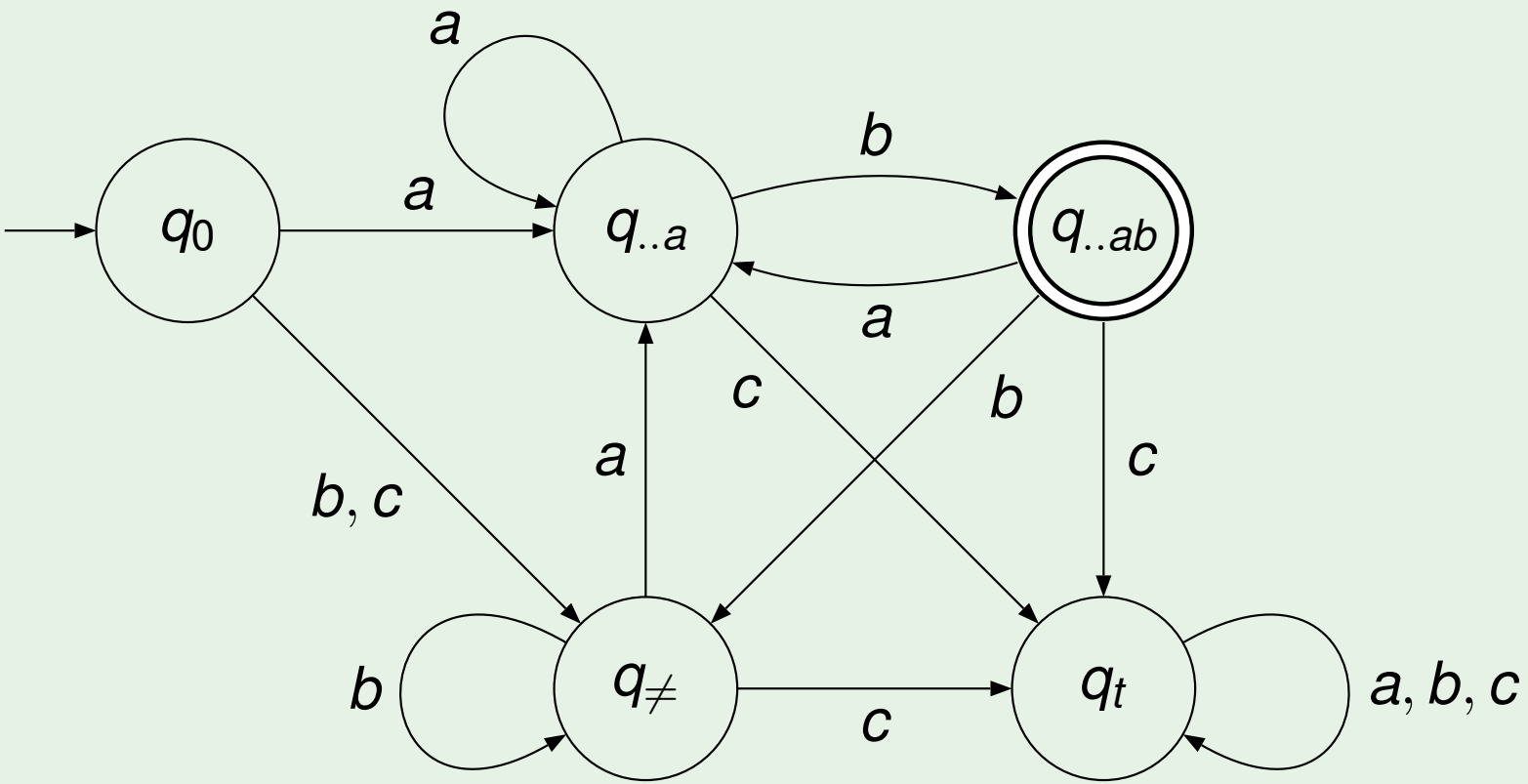
El autómata acepta palabras acabadas con la cadena *ab* (necesaria para llegar al único estado final, q_3), precedida de tantas *a*'s y *b*'s como se quiera (debido al bucle con *a* y *b* de q_1), y empezando, opcionalmente gracias a la transición λ, con una única *c*:

$$L(A) = \{xab : x \in \{a,b\}^*\} \cup \{cxab : x \in \{a,b\}^*\} = \{\lambda, c\}\{a,b\}^*\{ab\}$$

Aunque los AFNDs tienen la misma capacidad expresiva que los AFDs (describen exactamente la misma clase de lenguajes, los lenguajes regulares, ver bibliografía), son en general más compactos y más fáciles de diseñar que los AFDs (ver ejemplos a continuación).

Ejemplo (4/4)

Compare el AFND anterior con el AFD (mínimo) equivalente a él:



ALGUNAS TÉCNICAS PARA EL DISEÑO DE AFNDs

Para diseñar un AFND se deben seguir las pautas generales indicadas para el diseño de AFDs (ver "Guía básica para el diseño de AFDs") pero sacando partido del ▸ no determinismo, que permite:

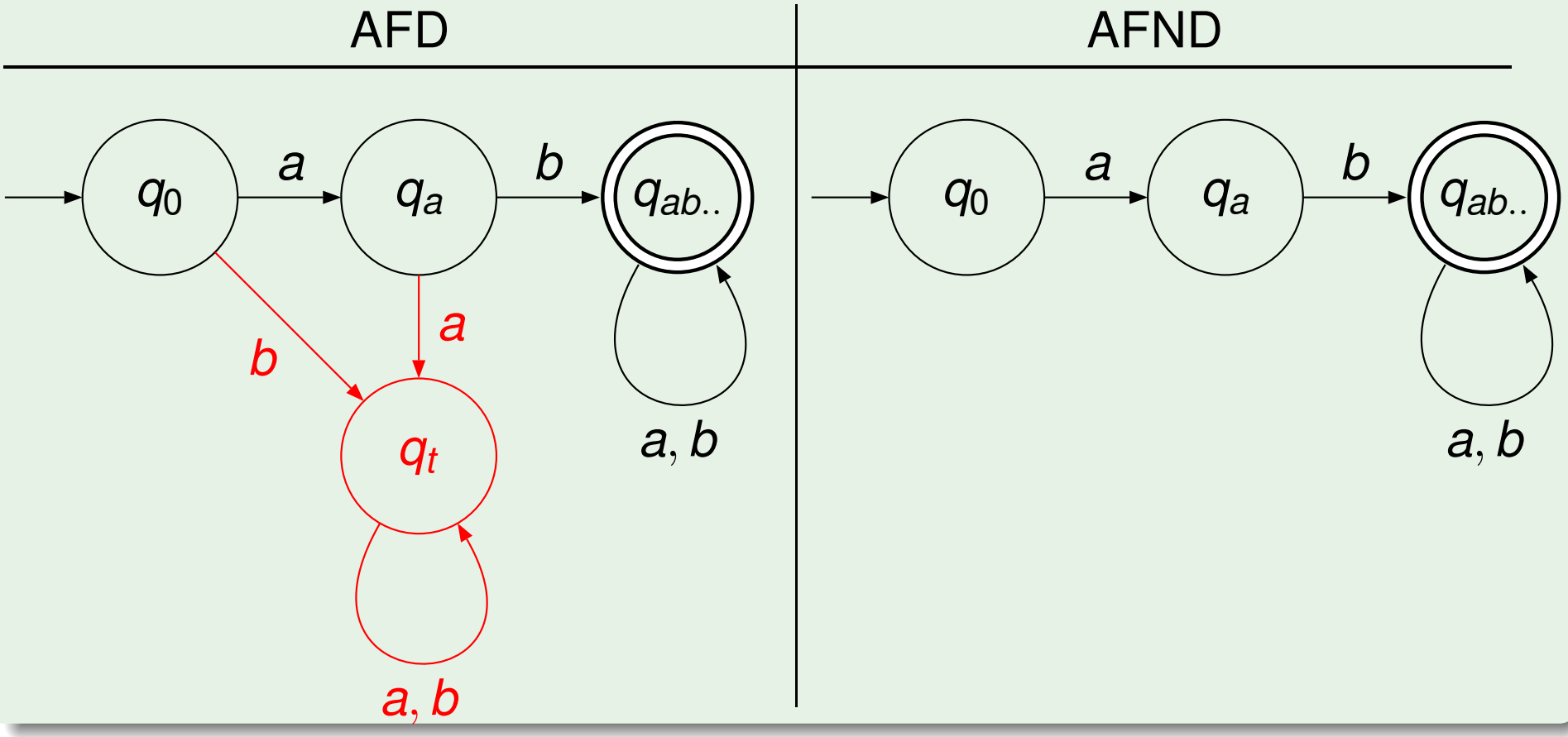
1. Omitir ciertas transiciones (p.ej. $f(q, a) = \{\}$) para *rechazar palabras*, prescindiendo de los *estado trampa no finales*.
2. Usar transiciones no deterministas (p.ej. $f(q, a) = \{p, q, r\}$) para *simultanear* de forma sencilla varias alternativas.
3. Usar transiciones λ (p.ej. $f(q, \lambda) = \{r, s\}$) para transitar entre estados sin consumir ningún símbolo, lo cual facilita la lectura de *símbolos opcionales* así como el diseño de *uniones, concatenaciones y cierres de lenguajes*.

A continuación se incluyen ejemplos ilustrando una o varias de estas tres estrategias.

Omisión de transiciones

Ejemplo (Palabras que empiezan por ...)

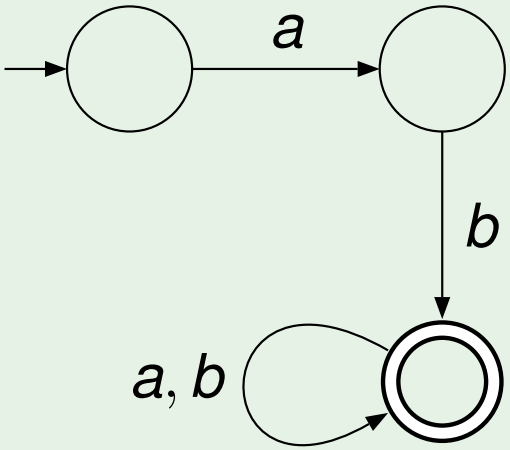
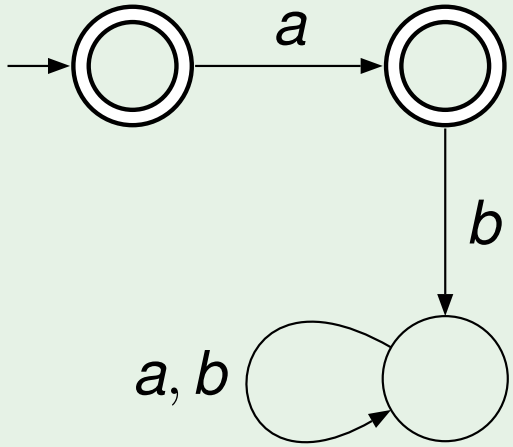
$$L = \{abw : w \in \{a, b\}^*\}$$



Observación

A diferencia de lo que ocurre con los AFDs (ver guía de diseño), la conversión de los estados finales en no finales, y viceversa, en un AFND, *no garantiza* la obtención del lenguaje complementario.

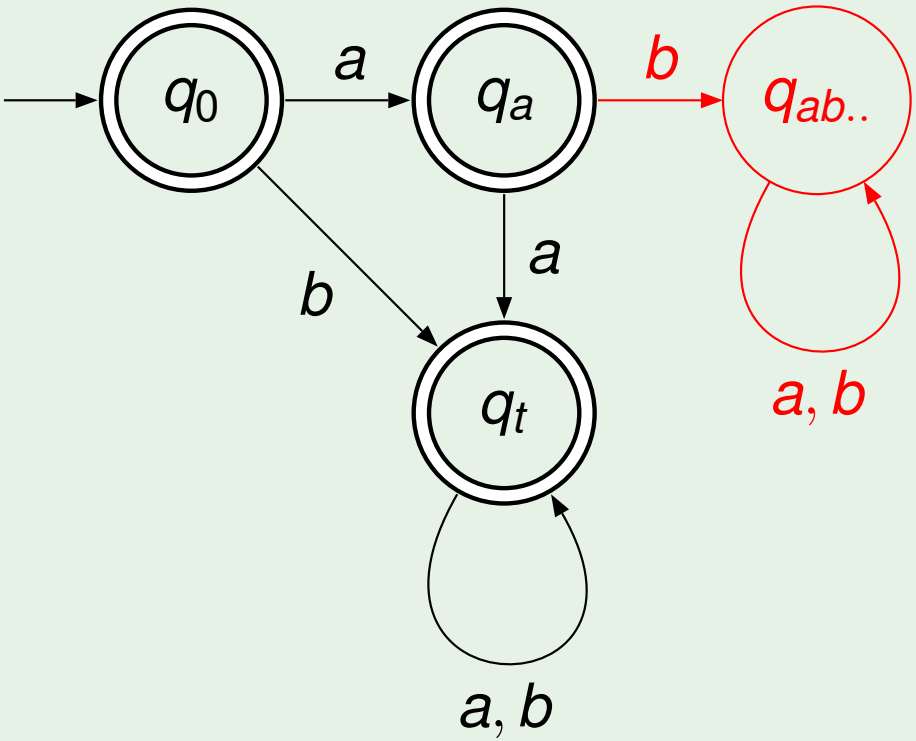
Ejemplo

AFND A	AFND B (intercambiando finales)
	
$L(A) = \{abw : w \in \{a,b\}^*\}$	$L(B) = \{\lambda, a\}$ ¡¡ $L(B) \neq \overline{L(A)}$!!

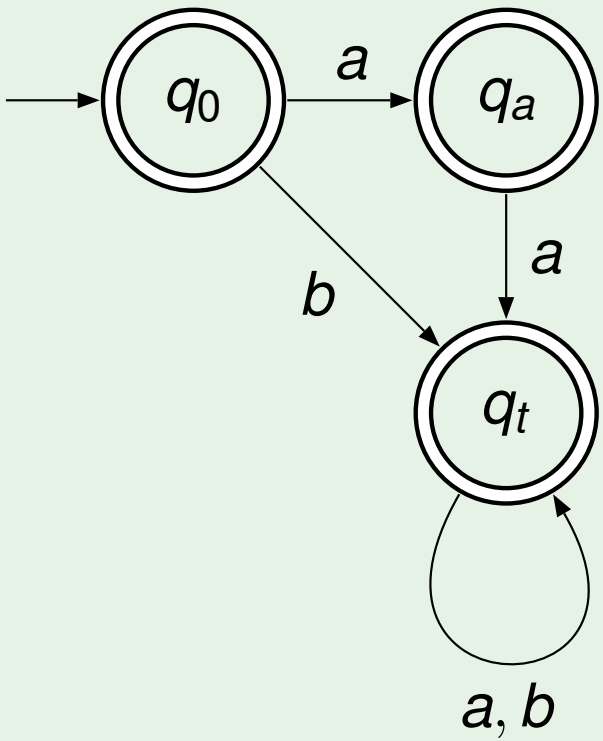
Ejemplo (Palabras que *no* empiezan por ...)

$$L = \{a, b\}^* - \{abw : w \in \{a, b\}^*\}$$

AFD



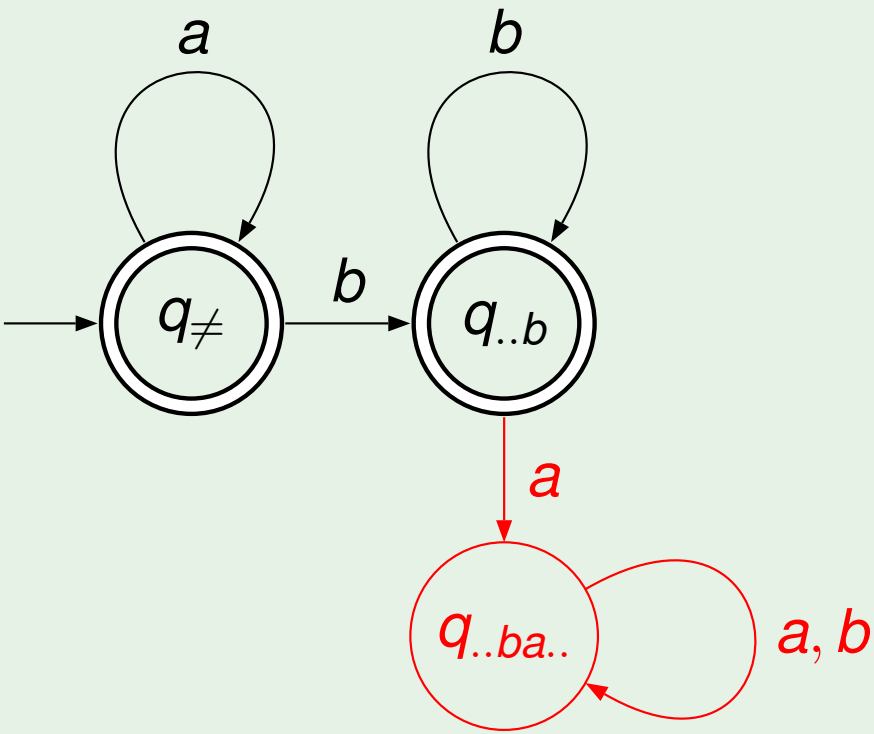
AFND



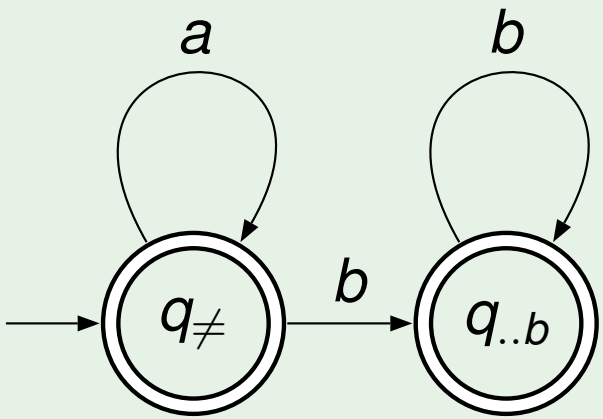
Ejemplo (Palabras que *no* contienen ...)

$$L = \{a, b\}^* - \{w_1 baw_2 : w_1, w_2 \in \{a, b\}^*\}$$

AFD



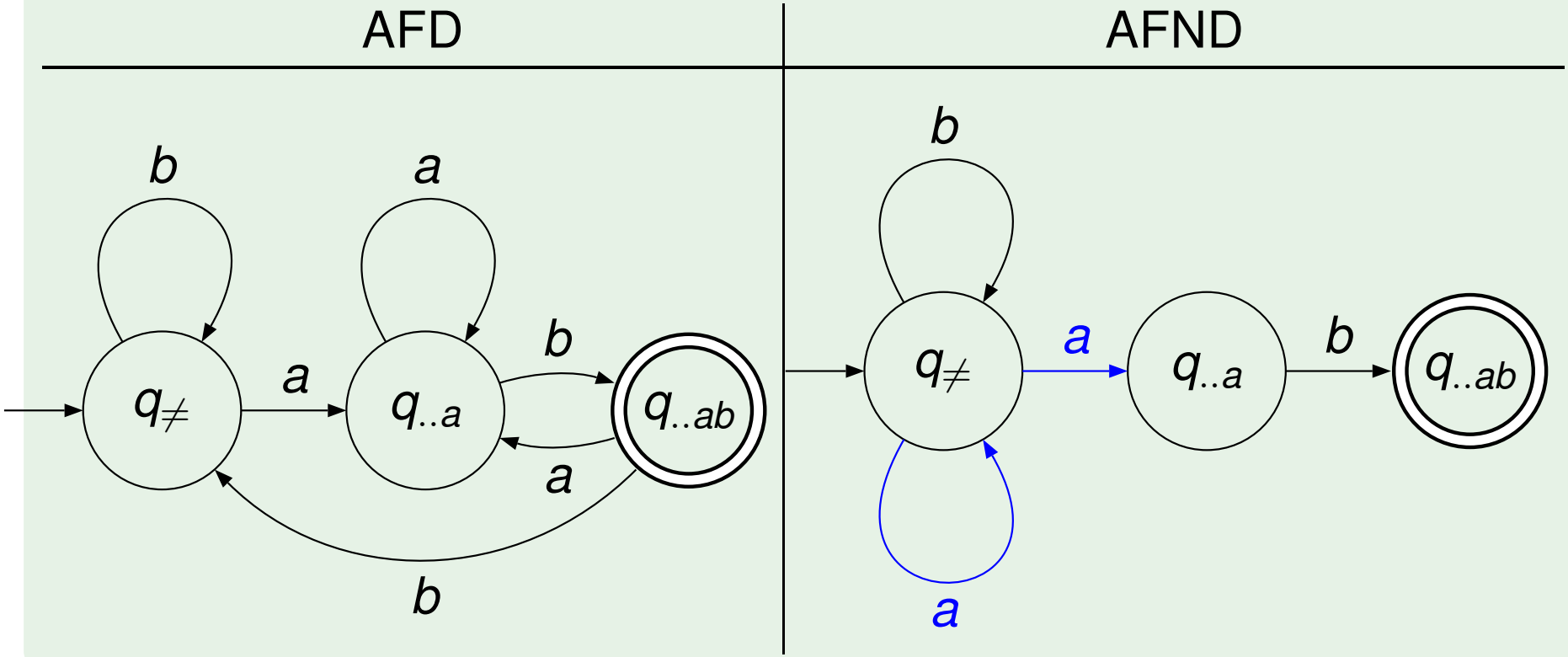
AFND



Uso de transiciones no deterministas

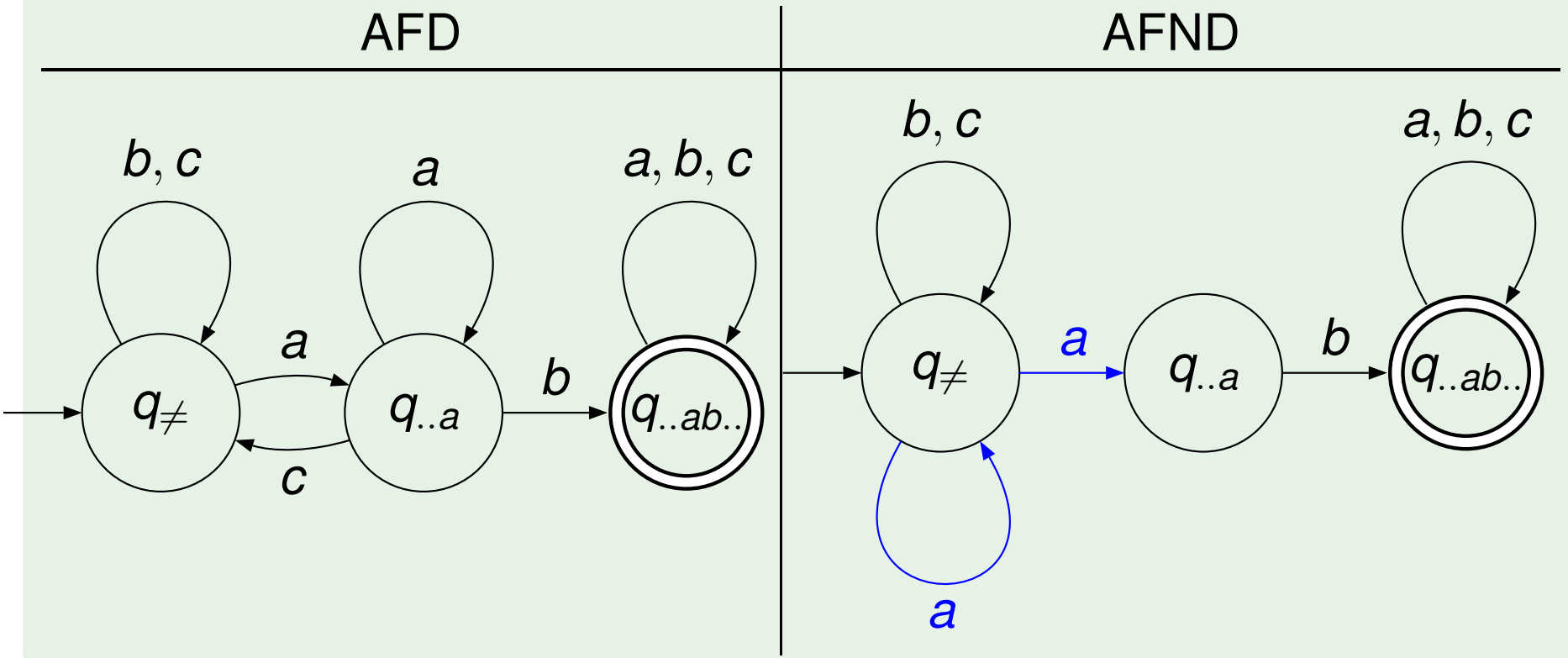
Ejemplo (Palabras que terminan por ...)

$$L = \{wab : w \in \{a, b\}^*\}$$



Ejemplo (Palabras que contienen ...)

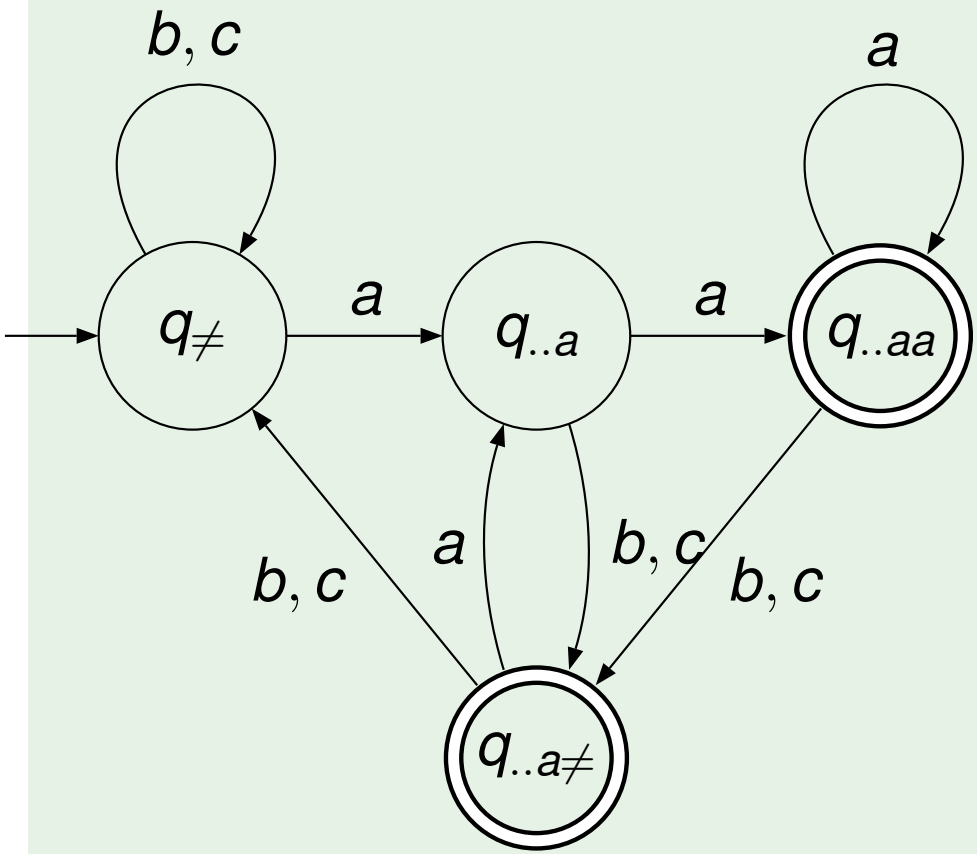
$$L = \{w_1 abw_2 : w_1, w_2 \in \{a, b, c\}^*\}$$



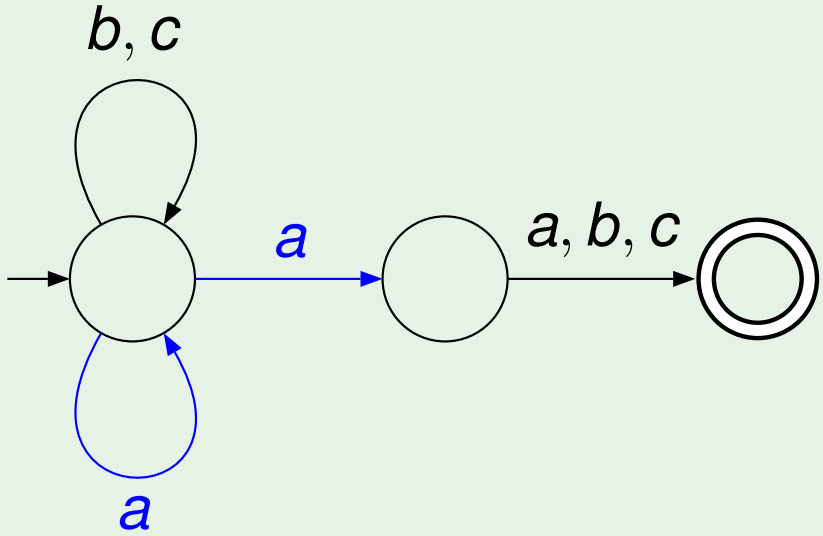
Ejemplo (Palabras cuyo penúltimo símbolo es ...)

$$L = \{w_1 a w_2 : w_1, w_2 \in \{a, b, c\}^*, |w_2| = 1\}$$

AFD



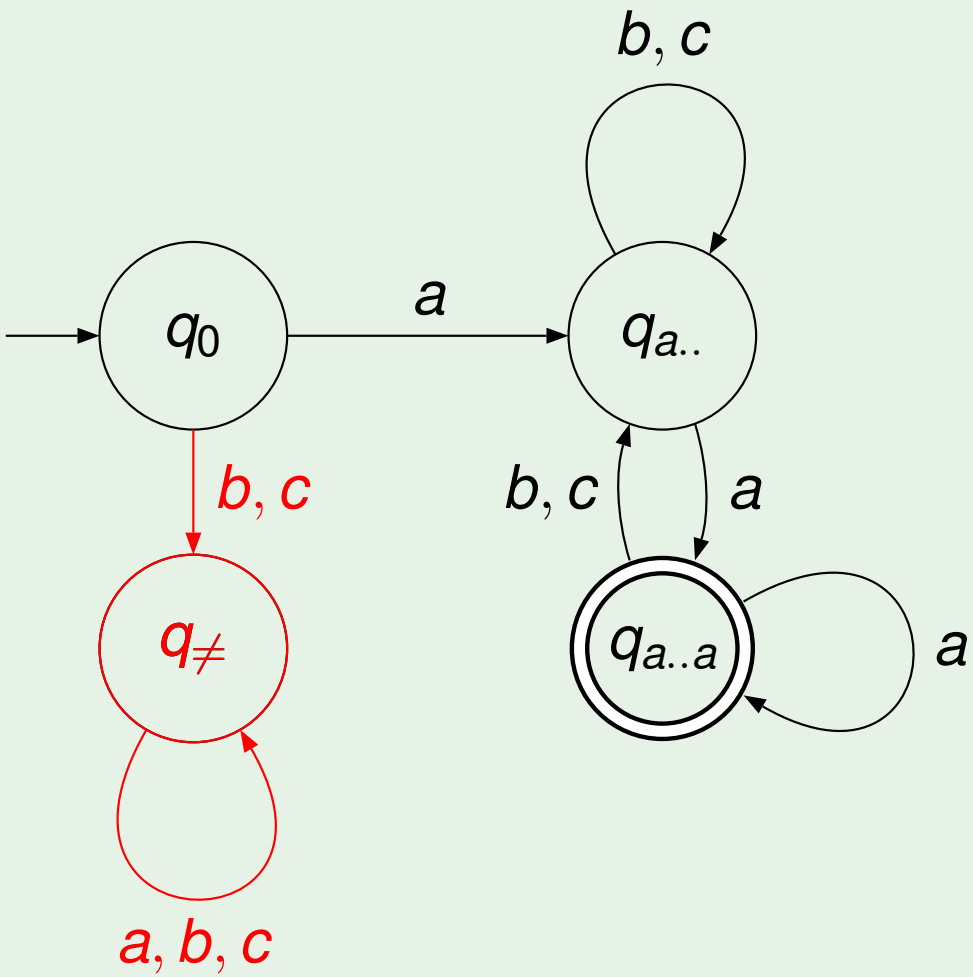
AFND



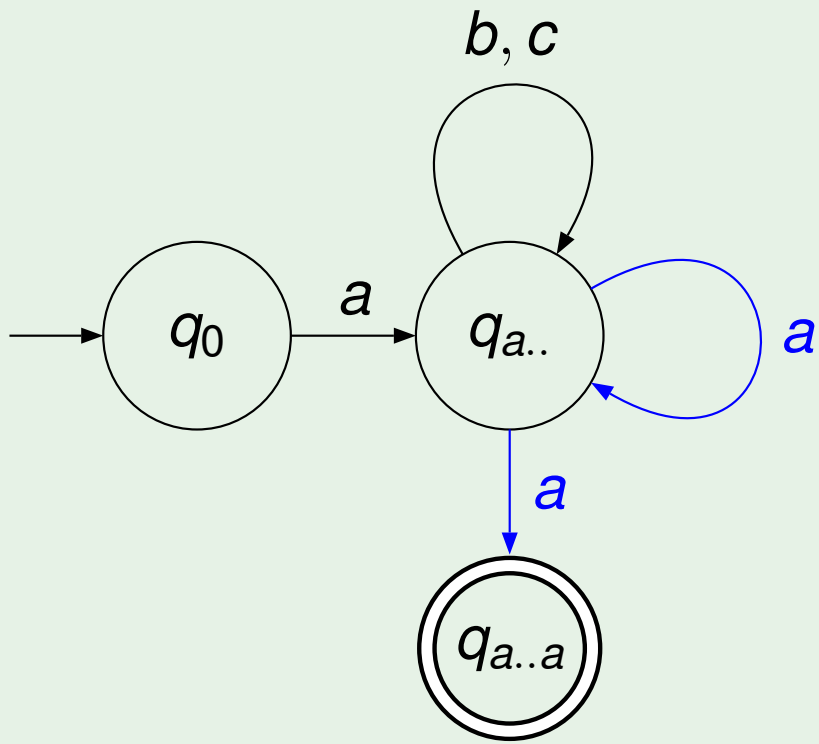
Ejemplo (Palabras $|x| > 1$ que empiezan y terminan por ...)

$$L = \{awa \ : \ w \in \{a, b, c\}^*\}$$

AFD (ver guía diseño AFDs)



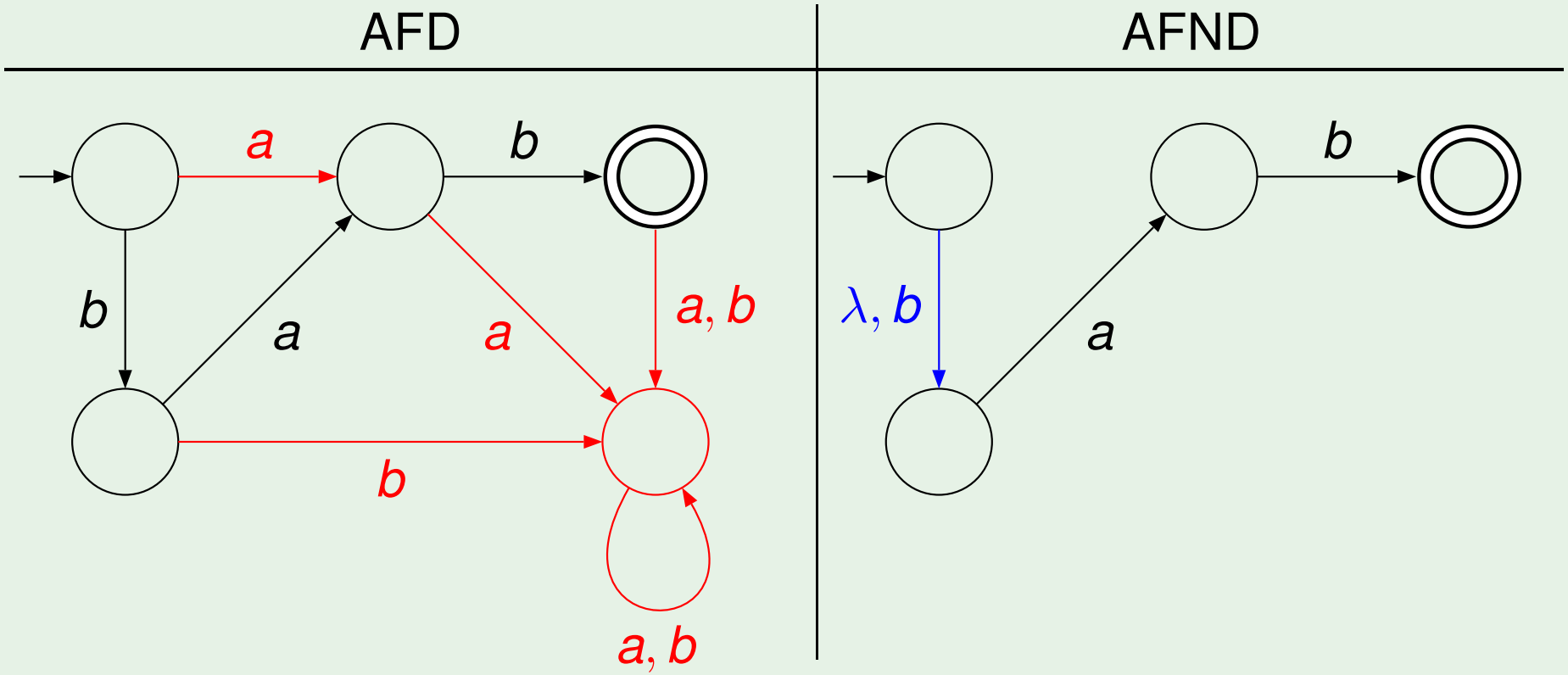
AFND



Uso de transiciones λ

AFNDs para lenguajes con símbolos opcionales

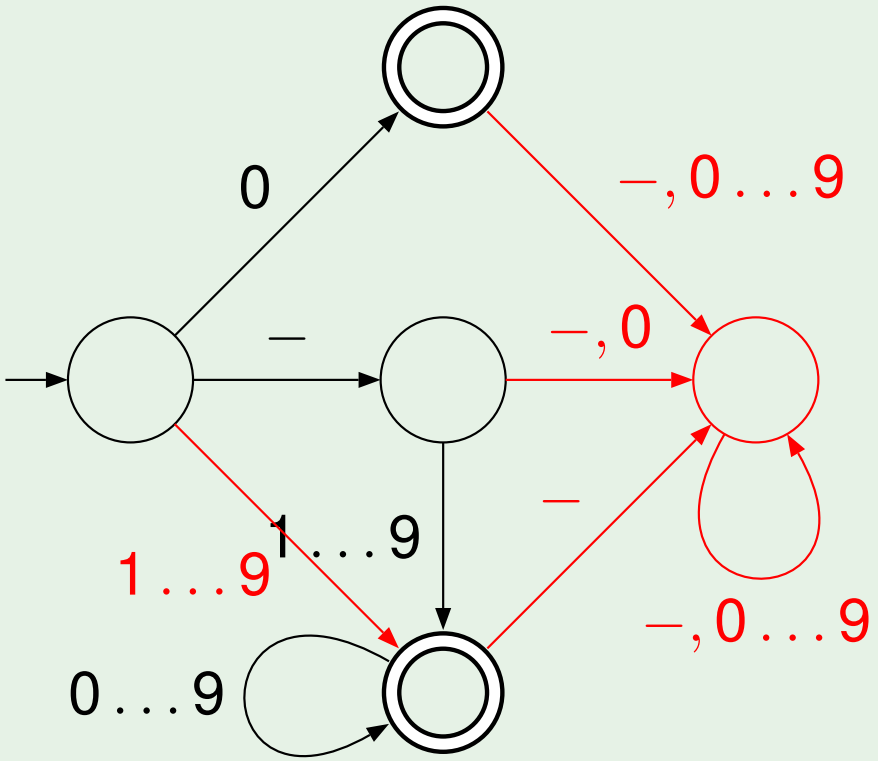
Ejemplo ($L = \{b^m ab : m \leq 1\} = \{\lambda, b\}\{ab\} = \{ab, bab\}$)



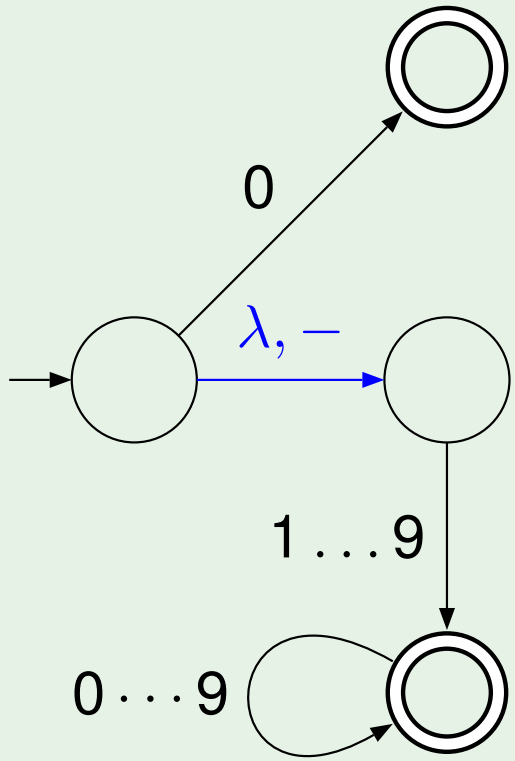
Ejemplo ($\mathbb{Z}$, conjunto de los números enteros)

$$L = \mathbb{Z} = \{\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots\} \subseteq \{-, 0, 1, \cdots, 9\}^*$$

AFD

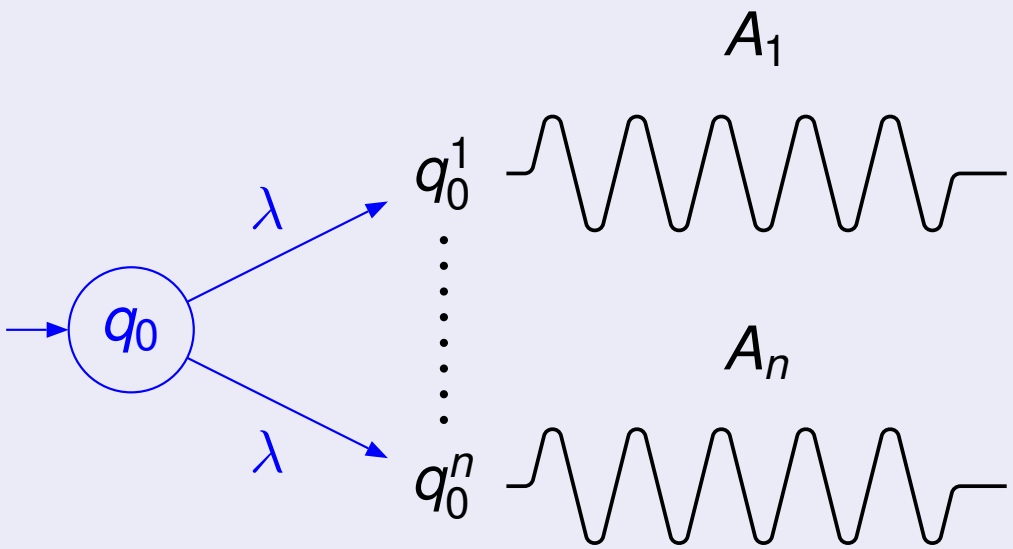


AFND



AFNDs para la unión de lenguajes

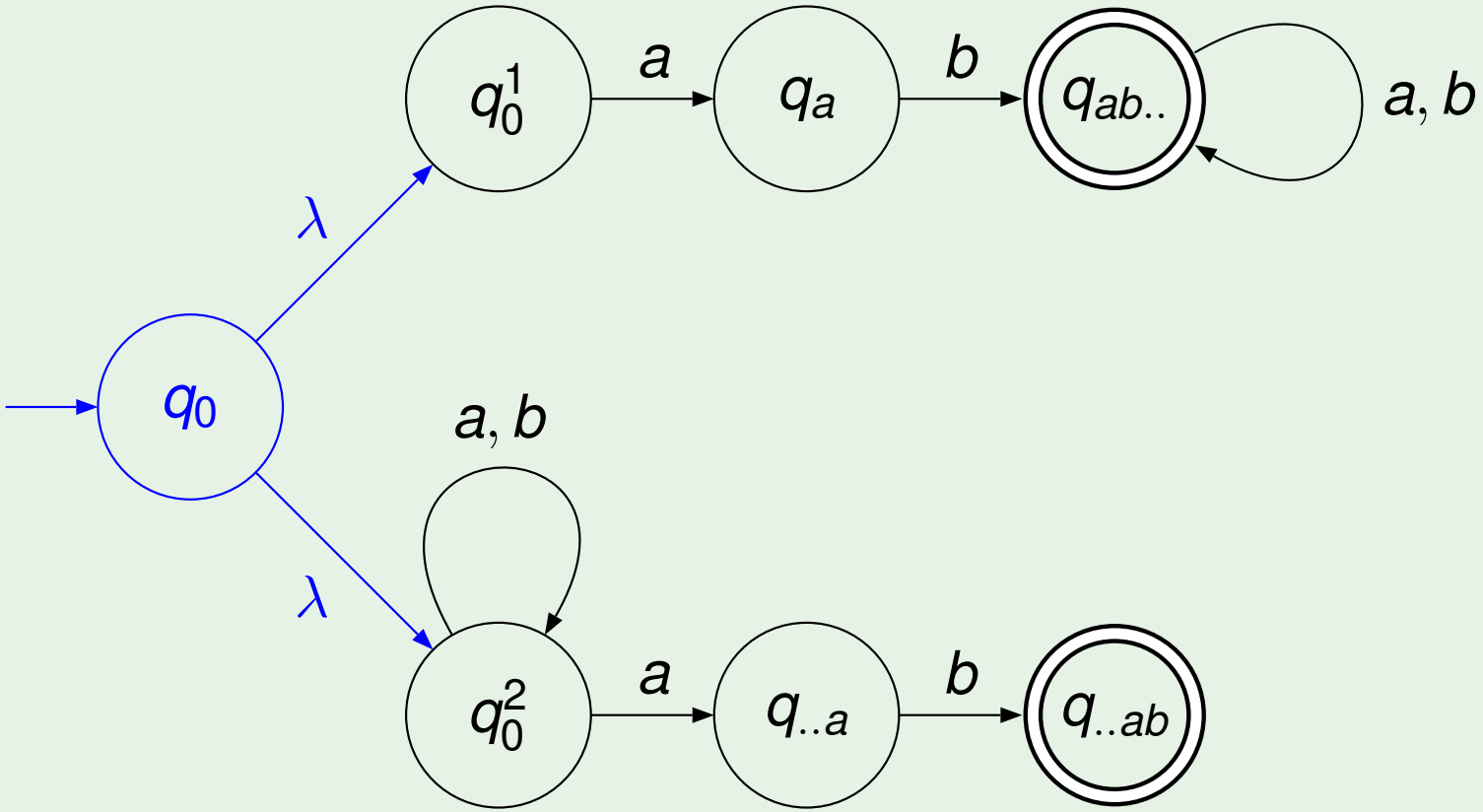
Si un lenguaje L se puede expresar como la unión de otros, $L = L_1 \cup \ldots \cup L_n$, una forma de obtener un AFND para L es construir AFs para $L_1 \ldots L_n$ (sean $A_1 \ldots A_n$ dichos autómatas, con estados iniciales $q_0^1 \ldots q_0^n$) y componerlos de la siguiente forma:



El AFND así obtenido no estará necesariamente entre los más sencillos posibles (este será el caso, por ejemplo, si algunos de los lenguajes L_i tienen partes comunes).

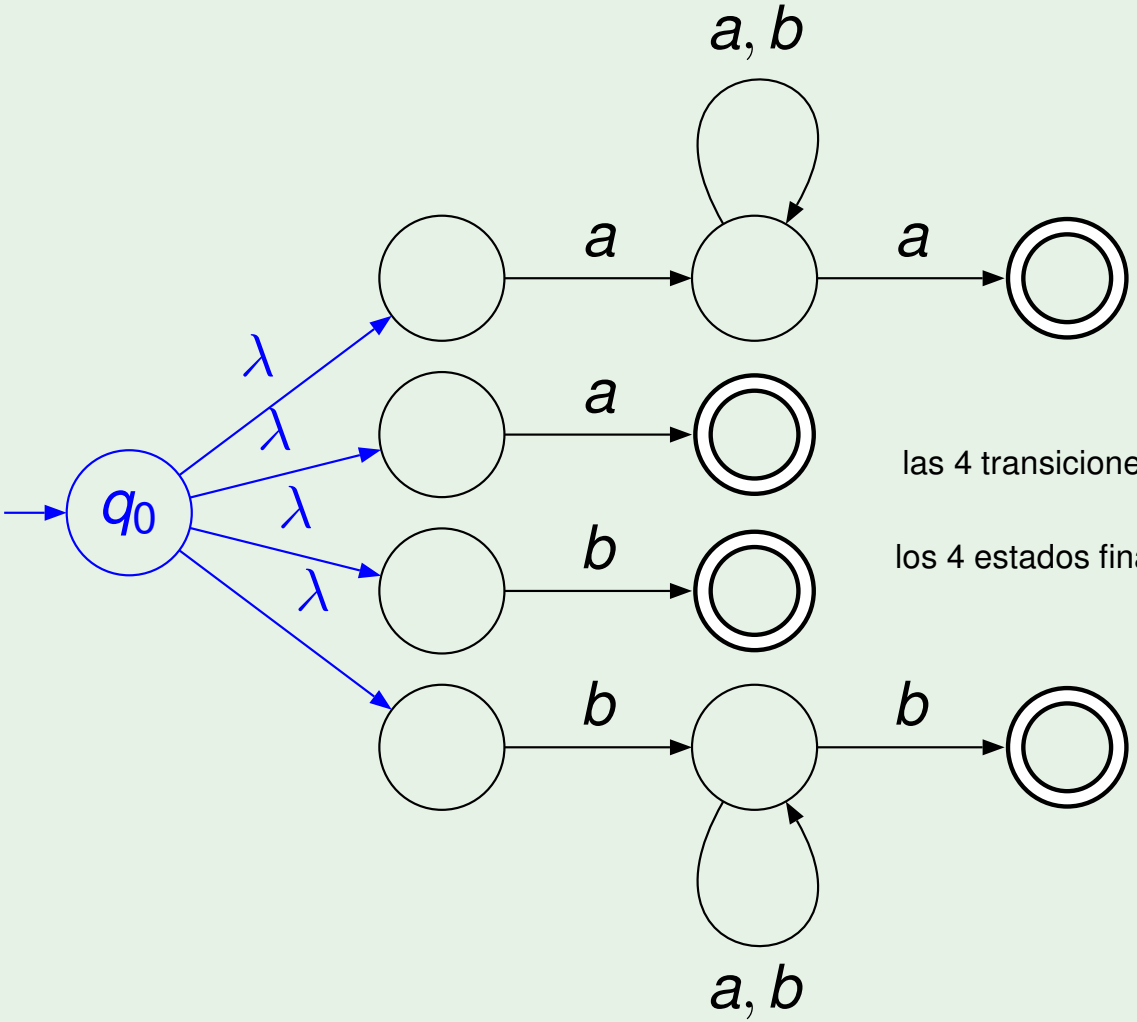
Ejemplo (Palabras que empiezan *o* terminan por *ab*)

$$L = L_1 \cup L_2 = \{abw : w \in \{a, b\}^*\} \cup \{wab : w \in \{a, b\}^*\}$$



Ejemplo (Palabras de $\{a, b\}^+$ que empiezan y terminan por el mismo símbolo)

$$L = \{awa : w \in \{a, b\}^*\} \cup \{a\} \cup \{b\} \cup \{bwb : w \in \{a, b\}^*\}$$



las 4 transiciones λ se podrían suprimir

los 4 estados finales podrían fundirse en uno

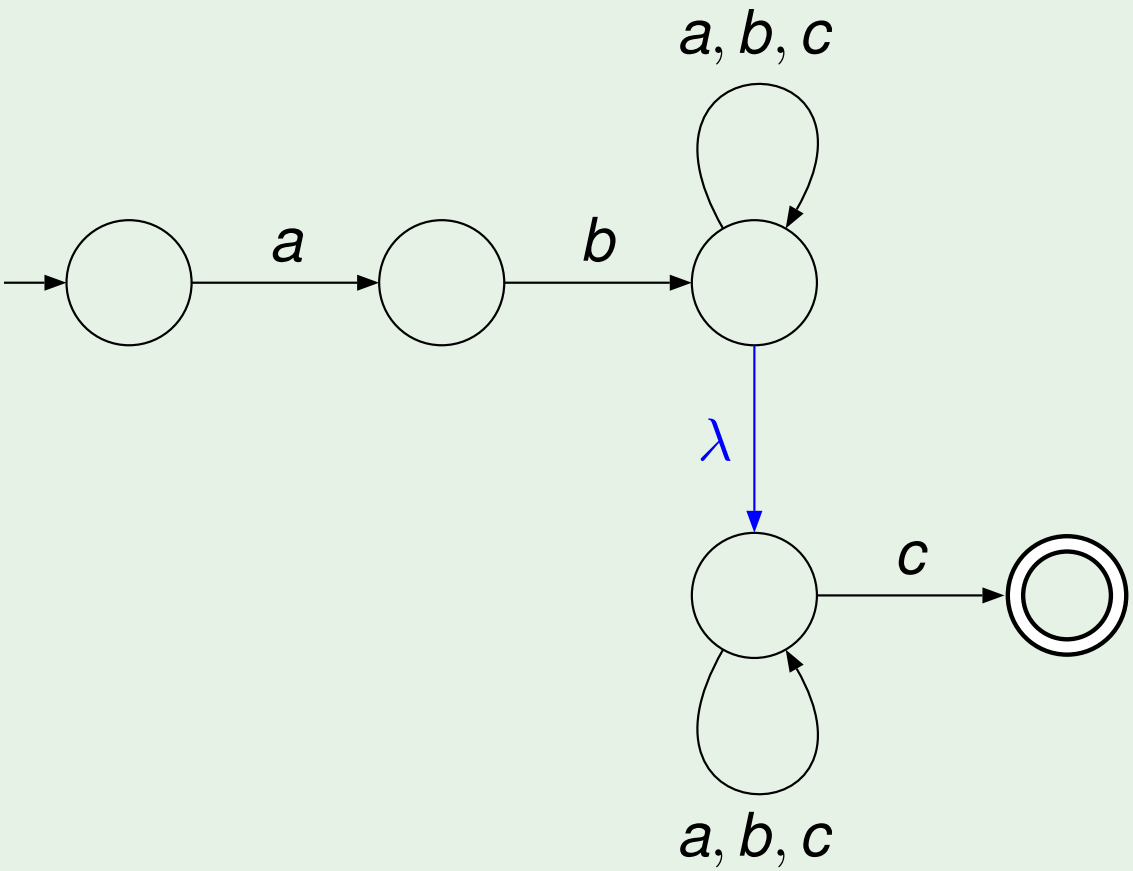
AFNDs para la concatenación de lenguajes

Si un lenguaje L se puede expresar como la concatenación de otros, $L = L_1 L_2$, una forma sencilla de obtener un AFND para L es la siguiente:

1. Construir AFs para L_1 y L_2, A_1 y A_2.
2. Construir un nuevo AF A combinando los dos anteriores como sigue:
   - Quitando la condición de finales a todos los estados de A_1 que sean finales.
   - Añadiendo una transición λ desde cada uno de los anteriores hasta el estado inicial de A_2, que deja de ser inicial.
3. Claramente A es un AFND tal que $L(A) = L$, aunque no estará necesariamente entre los más sencillos (véanse los comentarios a este respecto en algunos de los ejemplos que se proponen a continuación).

Ejemplo (Palabras de $\{a, b, c\}^*$ que empiezan por ab y terminan por c)

$$L = L_1 L_2 = \{abx : x \in \{a, b, c\}^*\}\{xc : x \in \{a, b, c\}^*\}$$



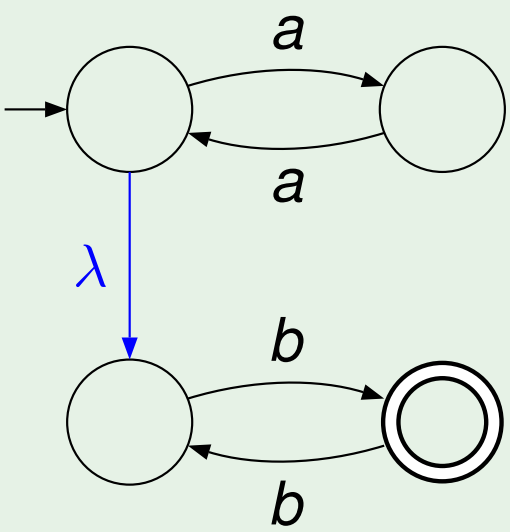
A_1 (su estado final deja de serlo)

A_2 (su estado inicial deja de serlo)

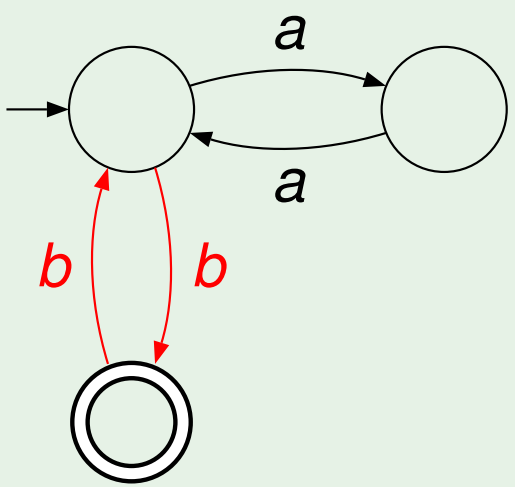
La transición λ es superflua: el último estado de A_1 podría fundirse con el primero de A_2

Ejemplo ($L = \{a^{2n} : n \geq 0\}\{b^{2n+1} : n \geq 0\}$)

AFND CORRECTO



AFND ¡INCORRECTO!

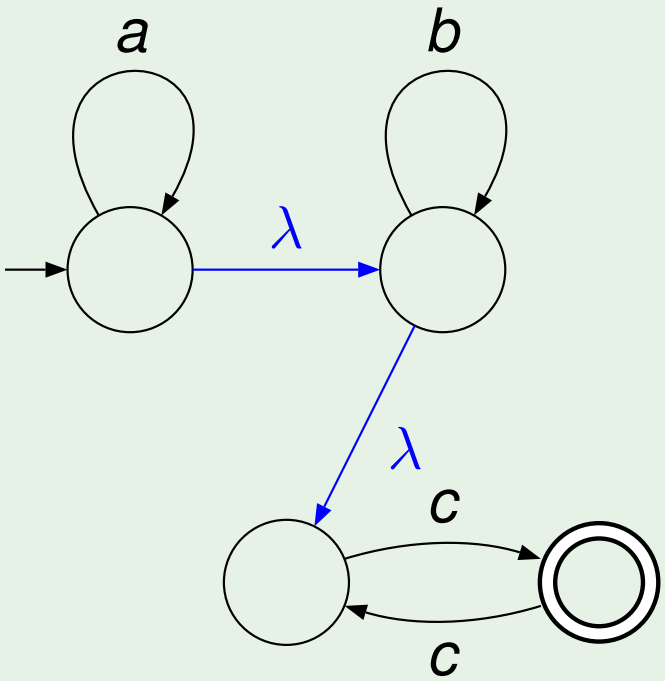


¡¡Acepta, p.ej., *bbaab*!!

La transición λ es necesaria

Ejemplo ($L = \{a^m b^n c^{2p+1} : m, n, p \geq 0\}$)

$$L = (L_1 L_2) L_3 = \Big(\{a^m : m \geq 0\}\{b^n : n \geq 0\}\Big)\{c^{2p+1} : p \geq 0\}$$



$A_1 A_2$ (sus dos estados finales dejan de serlo)

A_3 (su estado inicial deja de serlo)

Las transiciones λ son necesarias

AFNDs para cierres de Kleene de lenguajes

Dado un lenguaje L, una forma de obtener un AFND para su cierre de Kleene positivo, L^+, formado por todas las posibles concatenaciones de una o más palabras de L, es la siguiente:

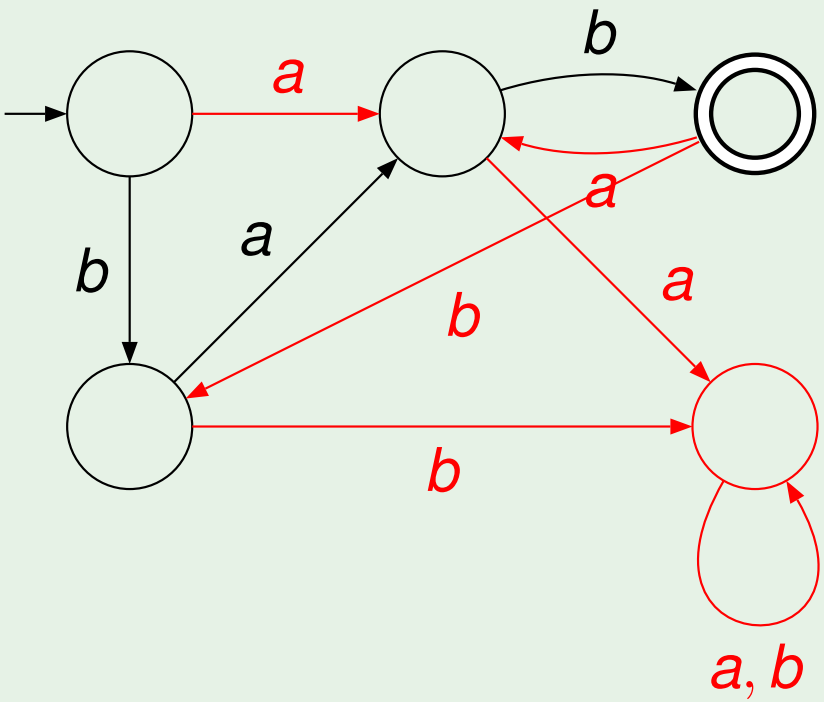
1. Construir un AFND para L, A.
2. Construir un nuevo AFND, A', añadiendo una transición λ desde cada uno de los estados finales de A hasta su estado inicial.
3. Claramente A' será un AFND tal que $L(A') = L^+$, aunque no estará necesariamente entre los más sencillos.

En cuanto a la obtención de un AFND para el cierre de Kleene, $L^* = L^+ \cup \{\lambda\}$, de un lenguaje L dado:

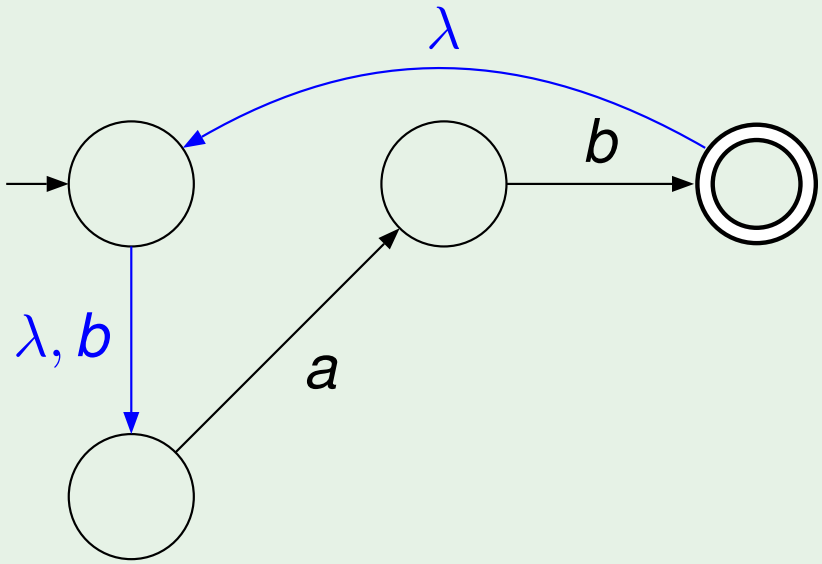
- Si $\lambda \in L$, entonces $L^+ = L^*$ y por lo tanto el AFND A' anterior también describe L^*.
- En caso contrario, bastará con convertir el estado inicial de A' en final para obtener un AFND A'' tal que $L(A'') = L^*$.

Ejemplo ($L = \{b^m ab : m \leq 1\} = \{\lambda, b\}\{ab\} = \{ab, bab\}$)

AFD para L^+

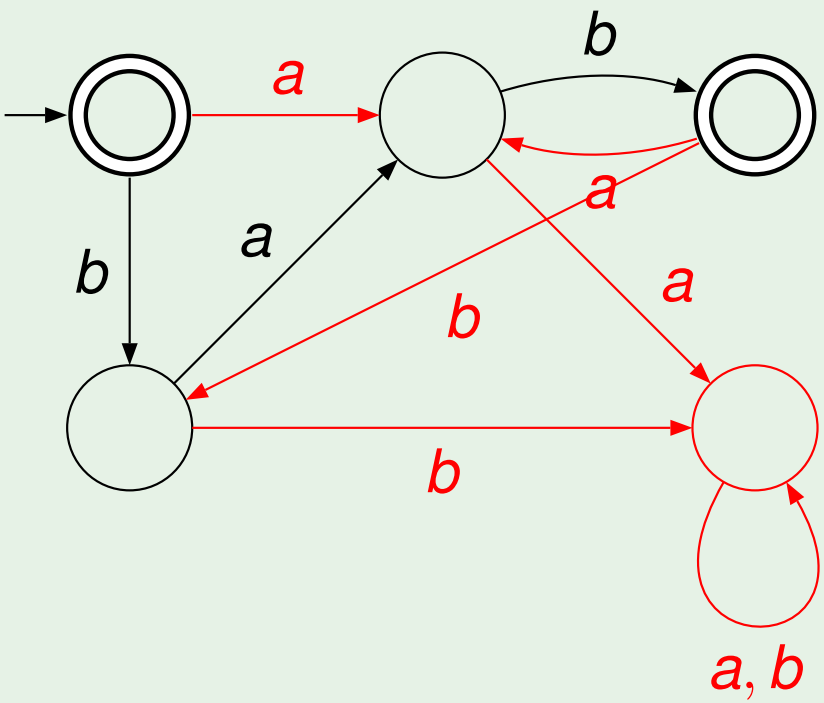


AFND para L^+

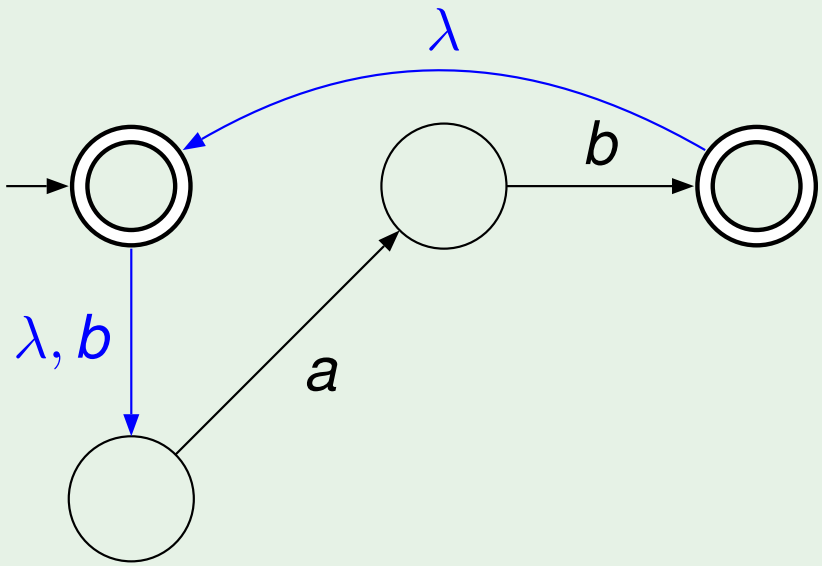


Ejemplo ($L = \{b^m ab : m \leq 1\} = \{\lambda, b\}\{ab\} = \{ab, bab\}$)

AFD para L^*



AFND para L^*

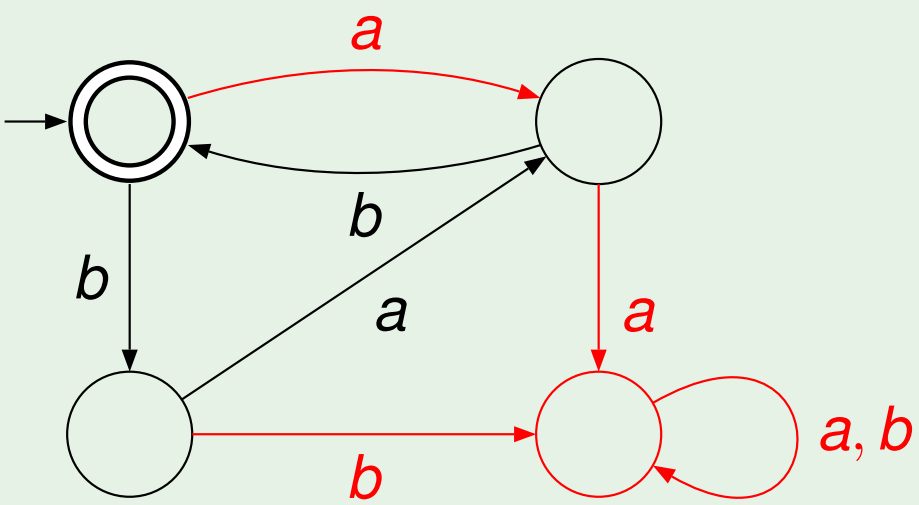


Observación

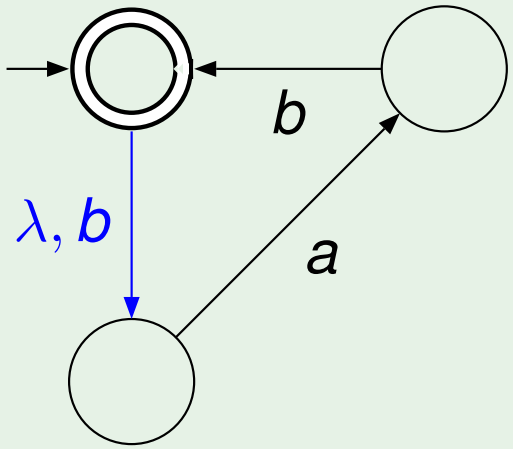
Los autómatas para L^* obtenidos de los propuestos para L^+ convirtiendo en finales sus estados iniciales, no son necesariamente los mejores: los que se proponen a continuación son equivalentes a los del último ejemplo, pero más sencillos.

Ejemplo ($L = \{b^m ab : m \leq 1\} = \{\lambda, b\}\{ab\} = \{ab, bab\}$)

AFD para L^*



AFND para L^*



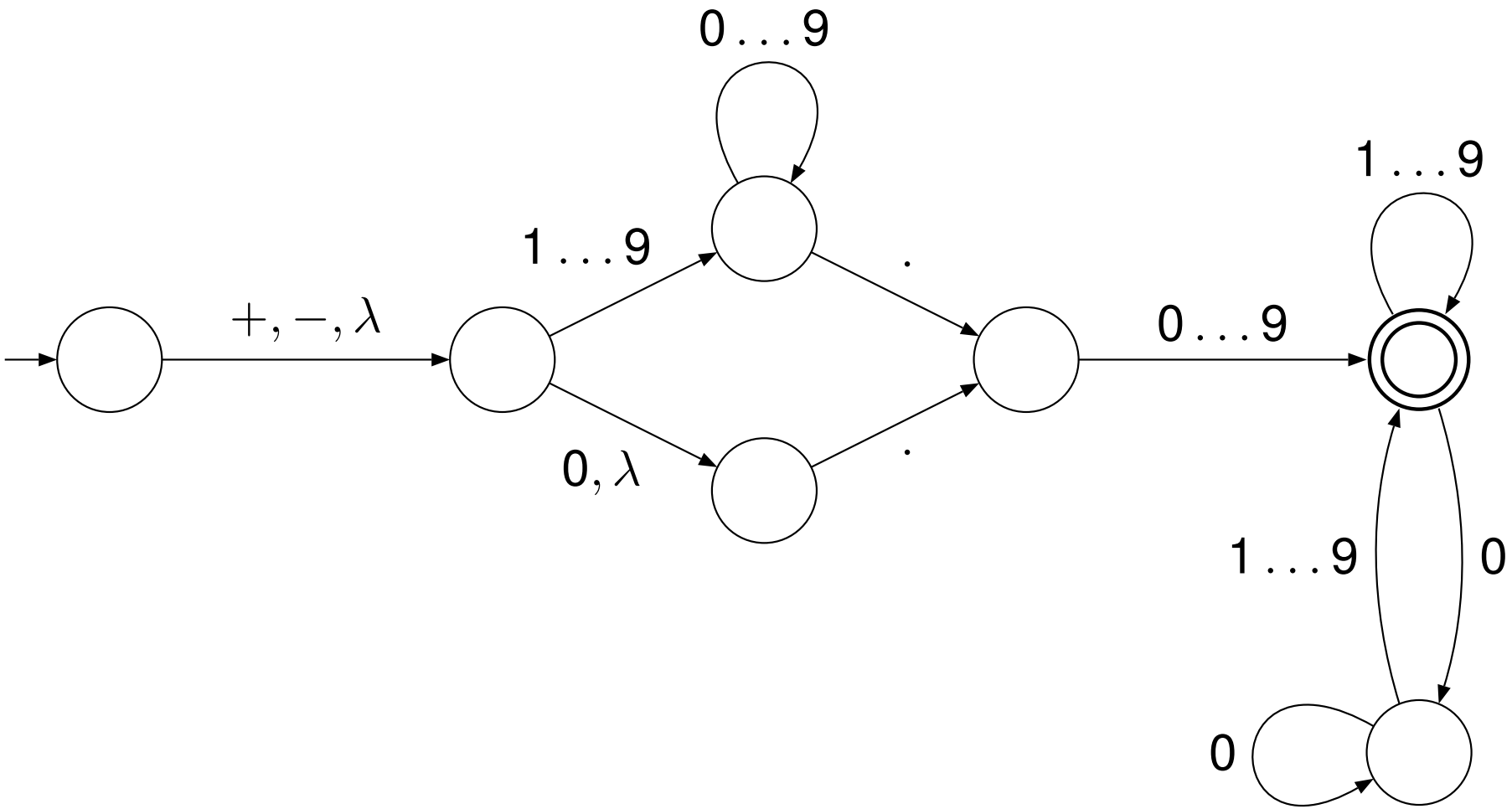
EJEMPLOS ADICIONALES

Números decimales

Diseño de un AFND para números decimales con el siguiente formato:

- Los números pueden empezar, opcionalmente, con un signo $+$ o un signo $-$.
- Su parte entera debe ser una cadena de dígitos que puede ser vacía pero que no puede empezar por el dígito 0 (salvo que solo conste de un 0).
- A continuación debe aparecer un punto.
- Por último, su parte decimal debe ser una cadena no vacía de dígitos que no puede acabar con el dígito 0 (salvo que solo conste de un 0).

Por ejemplo, $+ \cdot 0$, $0 \cdot 0$, $-0 \cdot 01$, $3 \cdot 14$, $+20 \cdot 0$, $0 \cdot 0007 \in L(A)$, pero sin embargo $\cdot$, $0\cdot$, $+01 \cdot 3$, -35 , $+3\cdot$, $\cdot 10$, $-3 \cdot 00 \notin L(A)$.



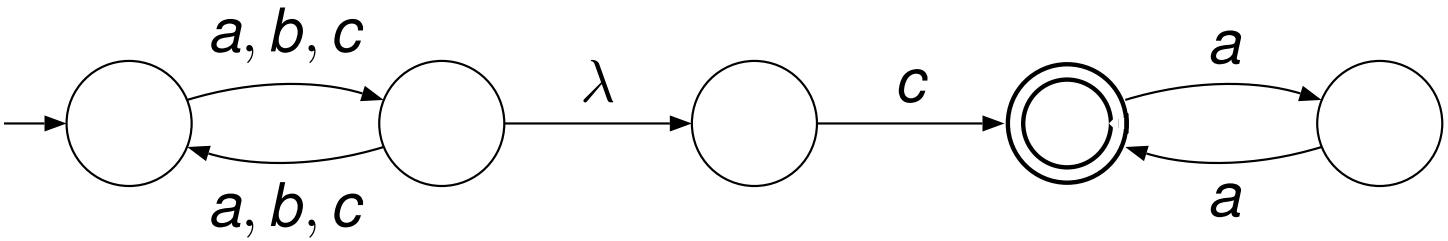
$$L = \{x = wca^{2n} : n \geq 0, w \in \{a, b, c\}^*, |x| \text{ MOD } 2 = 0\}$$

- El lenguaje L está formado por todas las palabras de $\{a, b, c\}^*$ con tamaño par que terminan por una cadena perteneciente al lenguaje $\{ca^{2n} : n \geq 0\}$. Por ejemplo:
  - $ac, cc, bbcc, acaa, babcaaaa, acaaaaaa, ... \in L$
  - $\lambda, a, c, bb, aacb, caa, caaa, ... \notin L$
- L se puede ver como la *concatenación* de otros dos lenguajes, $L = L_1 L_2$, siendo

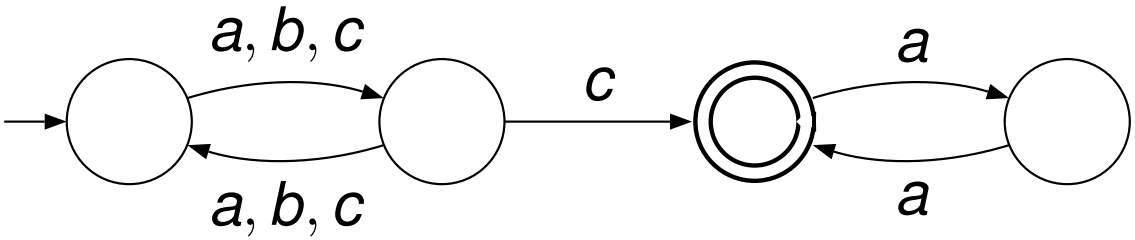
$$L_1 = \{w \in \{a, b, c\}^* : |w| \text{ MOD } 2 = 1\}$$
$$L_2 = \{ca^{2n} : n \geq 0\}$$

En efecto, como las palabras de L_2 tienen siempre tamaño impar, las de L_1 también tienen que tener tamaño impar para que el tamaño total sea par.

Un AFND para L, obtenido concatenando mediante una transición λ los AFNDs correspondientes a L_1 y L_2, y quitando la condición de final al estado final de L_1, es el siguiente:



En este caso la transición λ es innecesaria, por lo que un AFND equivalente, más sencillo, es:

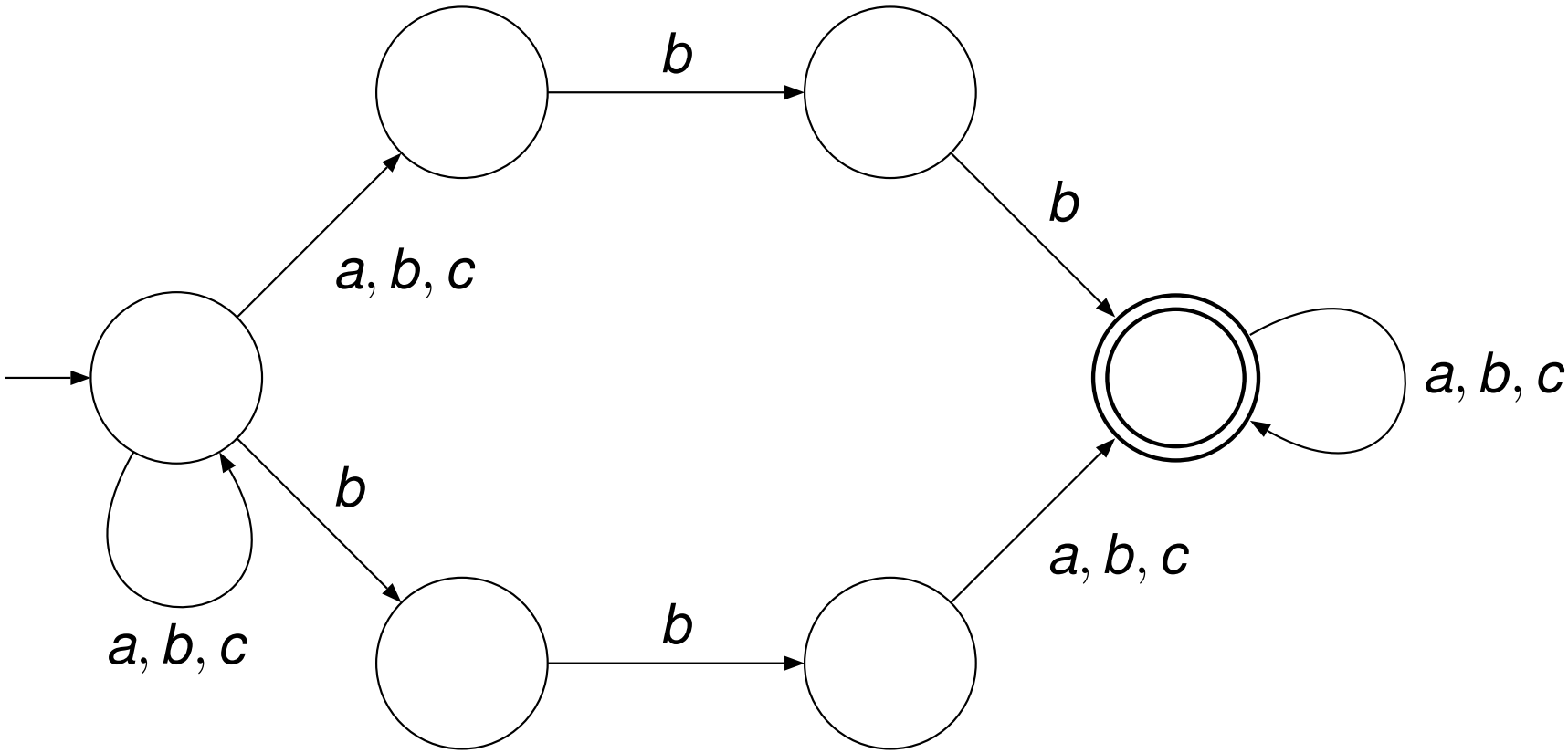


$$L = \{x = x_1 bbx_2 : x_1, x_2 \in \{a, b, c\}^*, |x| > 2\}$$

- El lenguaje L está formado por todas las palabras de $\{a, b, c\}^*$ con tamaño mayor o igual que 3 que contienen al menos dos b's consecutivas. Por ejemplo:
  - $abb, bbb, bbc, acbbbcc, babbbcbb, \ldots \in L$
  - $\lambda, a, b, ab, bb, abc, baabcb, \ldots \notin L$
- Como las palabras de L constan de al menos 3 símbolos y contienen bb, tienen que tener al menos un símbolo o bien delante o bien detrás de bb. Por ello, el lenguaje L se puede ver como la *unión* de otros dos, $L = L_1 \cup L_2$, siendo

$$L_1 = \{x_1 bbx_2 : x_1 \in \{a, b, c\}^+, x_2 \in \{a, b, c\}^*\}$$
$$L_2 = \{x_1 bbx_2 : x_1 \in \{a, b, c\}^*, x_2 \in \{a, b, c\}^+\}$$

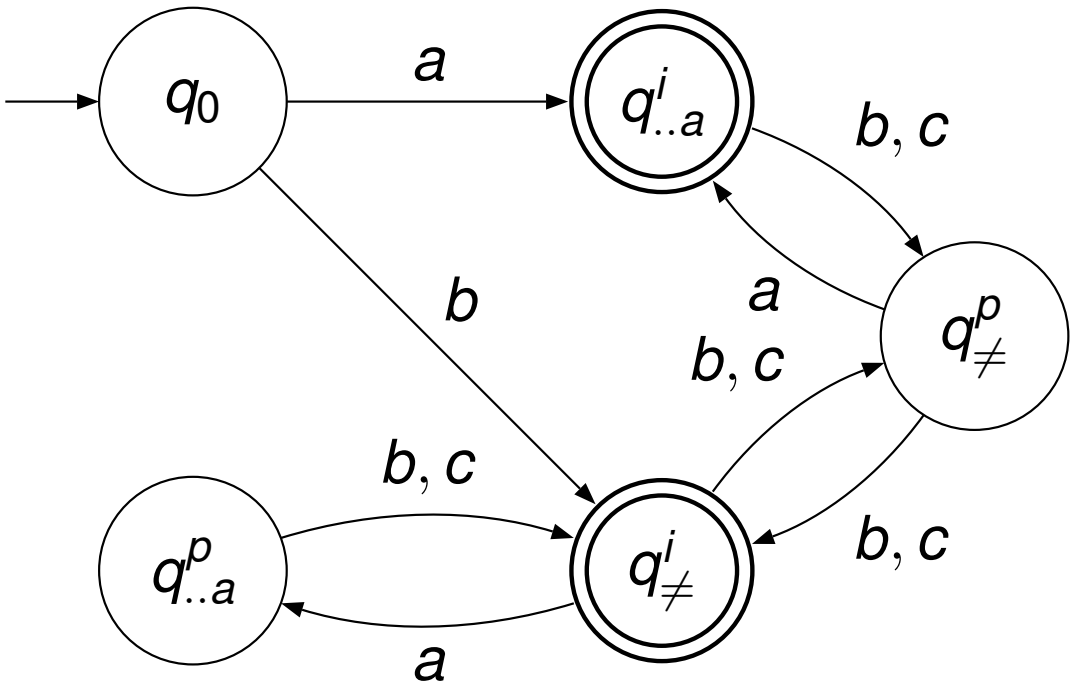
Un AFND para $L = L_1 \cup L_2$, obviando las transiciones λ iniciales, innecesarias en este caso, y fundiendo los dos estados finales en uno, sería el siguiente:



$L = \{w \in \{a,b,c\}^* : |w| \text{ MOD } 2 = 1\} - \{cx : x \in \{a,b,c\}^*\} - \{x_1 aax_2 : x_1, x_2 \in \{a,b,c\}^*\}$

- El lenguaje L está formado por todas las palabras de $\{a,b,c\}^*$ con longitud impar que no empiezan por c y no contienen dos a's consecutivas. Por ejemplo:
  - $a, b, abb, abc, aba, bac, acbac, ... \in L$
  - $\lambda, c, ca, cca, aab, aaa, abbc, ... \notin L$
- Para el diseño de un AFND para L se consideran los siguientes estados:
  - q_0: todavía no se ha leído nada (y por lo tanto no se puede leer una c).
  - $q^i_{..a}$: lo último leído es una a, longitud impar.
  - $q^p_{..a}$: lo último leído es una a, longitud par.
  - $q^i_{\neq}$: lo último leído no es una a, longitud impar.
  - $q^p_{\neq}$: lo último leído no es una a, longitud par.

Un AFND para L con los estados anteriores:

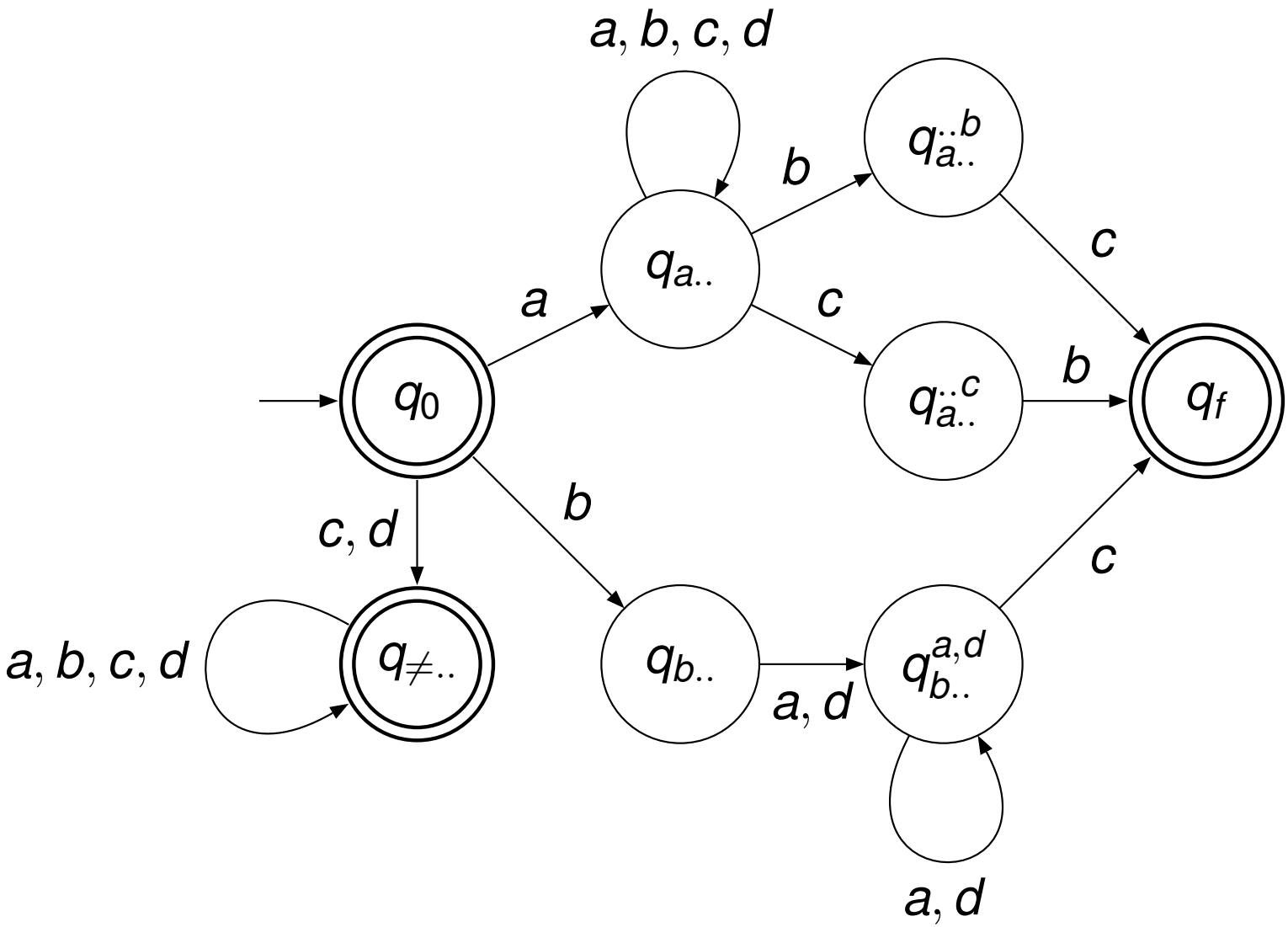


$L = \{w \in \{a, b, c, d\}^* : \text{si } \cdots \text{ entonces } \cdots$

Diseño de un AFND para el lenguaje $L \subseteq \{a, b, c, d\}^*$ formado por todas las palabras que cumplen las siguientes condiciones:

1. Si la palabra empieza por a, entonces termina por bc o cb.
2. Si la palabra empieza por b, entonces termina por c y entre ambos símbolos solo pueden aparecer secuencias $x \in \{a, d\}^+$.

De lo anterior se deduce que si la palabra es vacía o empieza por cualquier otra símbolo (c o d), no hay ninguna restricción.


a, b, c, d
b
$q^{..b}_{a..}$
c
$q_{a..}$
a
c
q_0
$q^{..c}_{a..}$
b
q_f
c, d
b
c
a, b, c, d
$q_{\neq..}$
$q_{b..}$
$q^{a,d}_{b..}$
a, d
a, d

BIBLIOGRAFÍA

- Este material docente está basado en fuentes diversas, en particular en los libros y recursos que se citan a continuación, de los que proceden muchos de los ejercicios que se discuten aquí.

- Se nutre también del trabajo de nuestros antiguos compañeros Jorge Martín Corujo, Holger Billhardt y José Miguel Buenaposada, a los que expresamos desde aquí nuestro agradecimiento.

- P. Linz, An Introduction to Formal Languages and Automata. Jones and Barlett Publishers, 4th ed., 2006.
- J.E. Hopcroft, R. Motwani, J.D. Ullman, Introducción a la Teoría de Autómatas, Lenguajes y Computación, Addison-Wesley Iberoamericana, 3ª edición, 2007.
- M. Alfonseca, J. Sancho, M. Martínez Orga, Teoría de Lenguajes, Gramáticas y Autómatas, Ediciones RAEC 1997.
- P. Isasi, P. Martínez, D. Borrajo. Lenguajes, gramáticas y autómatas. Un enfoque práctico. Addison-Wesley, 1997.
- M. Alfonseca, E. Alfonseca, A. Ortega, Teoría de Autómatas y Lenguajes formales, Mc. Graw Hill DL, 2007.

DISTRIBUCIÓN

TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES

(TALF)

GUÍA BÁSICA PARA EL DISEÑO DE EXPRESIONES REGULARES (ERs)

Ana Pradera, Juan Manuel Serrano
Sergio Saugar, Mónica Robledo
César Alfaro, Javier Gómez
María Teresa González de Lena, Gema Gutiérrez

Noviembre de 2024

1. INTRODUCCIÓN
2. ERs: CONCEPTOS FUNDAMENTALES
   - Definición
   - Lenguaje descrito por una ER
3. ERs PARA ALGUNOS LENGUAJES DISTINGUIDOS
   - $L = \emptyset = \{\}$
   - $L = \{\lambda\}$
   - $L = \Sigma = \{a, b\}$
   - $L = \Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, ...\}$
   - $L = \Sigma^+ = \Sigma^* - \{\lambda\} = \{a, b, aa, ab, ...\}$
4. ERs PARA LENGUAJES CON CARACTERÍSTICAS TÍPICAS
   - Longitud de las palabras
     - $L = \{w \in \Sigma^* : |w| = m\}$
     - $L = \{w \in \Sigma^* : |w| \leq m\}$
     - $L = \{w \in \Sigma^* : |w| \geq m\}$
     - $L = \{w \in \Sigma^* : |w| \neq m\}$
     - $L = \{w \in \Sigma^* : |w| \text{ MOD } 2 = 0\}$
     - $L = \{w \in \Sigma^* : |w| \text{ MOD } m = 0\}$

- $L = \{w \in \Sigma^* : |w| \text{ MOD } 2 = 1\}$
- $L = \{w \in \Sigma^* : |w| \text{ MOD } m = i\}$
- $L = \{w \in \Sigma^* : |w| \text{ MOD } m \neq 0\}$

- Apariciones de un cierto símbolo
  - $L = \{w \in \Sigma^* : n_a(w) = m\}$
  - $L = \{w \in \Sigma^* : n_a(w) \leq m\}$
  - $L = \{w \in \Sigma^* : n_a(w) \geq m\}$
  - $L = \{w \in \Sigma^* : n_a(w) \neq m\}$
  - $L = \{w \in \Sigma^* : n_a(w) \text{ MOD } 2 = 0\}$
  - $L = \{w \in \Sigma^* : n_a(w) \text{ MOD } m = 0\}$
  - $L = \{w \in \Sigma^* : n_a(w) \text{ MOD } 2 = 1\}$
  - $L = \{w \in \Sigma^* : n_a(w) \text{ MOD } m = i\}$
  - $L = \{w \in \Sigma^* : n_a(w) \text{ MOD } m \neq 0\}$
- Comienzo de las palabras
  - Discusión general
  - $L = \{abaw \ : \ w \in \Sigma^*\}$
  - $L = \Sigma^* - \{aaw \ : \ w \in \Sigma^*\}$
- Final de las palabras
  - Discusión general
  - $L = \{waa \ : \ w \in \Sigma^*\}$

- $L = \Sigma^* - \{waba \ : \ w \in \Sigma^*\}$

- Contenido de las palabras
  - Discusión general
  - $L = \{w_1 abw_2 \ : \ w_1, w_2 \in \Sigma^*\}$
  - $L = \Sigma^* - \{w_1 abw_2 \ : \ w_1, w_2 \in \Sigma^*\}$
  - $L = \Sigma^* - \{w_1 aaw_2 \ : \ w_1, w_2 \in \Sigma^*\}$
  - $L = \Sigma^* - \{w_1 aabw_2 \ : \ w_1, w_2 \in \Sigma^*\}$

5 EJEMPLOS ADICIONALES

- $L = \{w \in \{a, b\}^+ : |w| \text{ MOD } 3 = 0\}$
- $L = \{a, b, c\}^* - \{abaw \ : \ w \in \{a, b, c\}^*\}$
- $L = \{a, b, c\}^+ - \{wab \ : \ w \in \{a, b, c\}^*\}$
- $L = \{w \in \{a, b, c\}^* \ : \ w \text{ no termina en } abab\}$
- $L = \{a, b, c\}^* - \{w_1 abw_2 \ : \ w_1, w_2 \in \{a, b, c\}^*\}$
- $L = \{w \in \{a, b\}^* : n_{ab}(w) \text{ MOD } 2 = 0\}$
- $L = \{w \in \{a, b, c\}^* : w \text{ no contiene dos b's juntas aisladas}\}$
- $L = \{a^n x : n > 0, x \in \{a, b, c\}^*, x \text{ no contiene } ccc\}$
- $L = \{w \in \{a, b, c\}^* : n_b(w) \text{ MOD } 2 = 1 \text{ y } w \text{ no acaba en } ac\}$
- $L = \{w \in \{a, b, c\}^* : n_b(w) \text{ MOD } 2 = 1 \text{ o } w \text{ no acaba en } ac\}$
- $L = \{w \in \{a, b, c\}^* : n_a(w) \geq 1, w \text{ contiene } bb\}$
- $L = \{w \in \{a, b, c\}^* : n_a(w) \geq 2, w \text{ no contiene } bb\}$

- $L = \{w \in \{a, b, 1\}^* : w$ empieza por letra y no contiene ...$\}$
- $L = \{c^p w c^q : p \leq 1, q \geq 0, w \in \{a, b\}^+, (p + q) \geq 1, (p + q)$ MOD $2 = 0\}$

6 BIBLIOGRAFÍA

7 DISTRIBUCIÓN

INTRODUCCIÓN

- El objetivo de estas notas es proporcionar, por medio de ejemplos, algunas pautas básicas para el diseño de Expresiones Regulares (ERs).

- Dado un lenguaje regular L, el problema consiste en diseñar una ER r que reconozca ese lenguaje, es decir, tal que el lenguaje descrito por r, $L(r)$, coincida con L: $L(r) = L$.

- El problema anterior no tiene una solución única puesto que ERs distintas pueden describir un mismo lenguaje (en cuyo caso se dice que las expresiones son *equivalentes*).

- El material que se presenta a continuación es fruto de la impartición de la materia *"Teoría de Autómatas y Lenguajes Formales"* en distintos grados universitarios del ámbito de la Informática.
- Se dirige a estudiantes tales que, además de manejar los conceptos básicos relativos a lenguajes formales, conocen con cierta profundidad las características y el funcionamiento de las *expresiones regulares*.
- En lo que sigue:
  - Se recuerdan en primer lugar los **conceptos fundamentales** relativos a ERs.
  - A continuación se discute, ilustrándolo con ejemplos, el diseño de ERs para **lenguajes distinguidos o con características típicas** (relativas a la longitud de sus palabras, las apariciones de un cierto símbolo o bien al comienzo, final o el contenido de las palabras).
  - Se termina con algunos **ejemplos adicionales**.

ERs: CONCEPTOS FUNDAMENTALES

Definición (Expresión Regular, ER)

Dado un alfabeto (conjunto finito y no vacío de símbolos) Σ, una expresión regular (ER) sobre Σ se define de forma recursiva como sigue:

1. Los símbolos $\emptyset$, λ y a, este último para cualquier $a \in \Sigma$, son expresiones regulares (conocidas como *expresiones regulares primitivas*).
2. Si r_1 y r_2 son expresiones regulares, entonces $r_1 + r_2$, $r_1 r_2$, r_1^* y (r_1) también son expresiones regulares.
3. No hay más expresiones regulares que las construidas mediante las dos reglas anteriores.

Definición (Lenguaje descrito por una ER (1/2))

Dada una ER r definida sobre un alfabeto Σ, se llama lenguaje descrito o reconocido por r, y se denota $L(r)$, al lenguaje definido de forma recursiva como sigue:

- Si $r = \emptyset$, entonces $L(r) = \{\} = \emptyset$
  (el lenguaje vacío).
- Si $r = \lambda$, entonces $L(r) = \{\lambda\}$
  (el lenguaje compuesto exclusivamente por la palabra vacía).
- Para todo símbolo $a \in \Sigma$, si $r = a$ entonces $L(r) = \{a\}$
  (el lenguaje compuesto exclusivamente por la palabra a).

Definición (Lenguaje descrito por una ER (2/2))

- Si $r = r_1 + r_2$, siendo r_1, r_2 expresiones regulares, entonces $L(r) = L(r_1) \cup L(r_2)$
  (la *unión* de los lenguajes $L(r_1)$ y $L(r_2)$).
- Si $r = r_1 r_2$, siendo r_1, r_2 expresiones regulares, entonces $L(r) = L(r_1)L(r_2)$
  (la *concatenación* de los lenguajes $L(r_1)$ y $L(r_2)$).
- Si $r = r_1^*$, siendo r_1 una expresión regular, entonces $L(r) = L(r_1)^*$
  (el cierre de Kleene del lenguaje $L(r_1)$).
- Si $r = (r_1)$, siendo r_1 una expresión regular, entonces $L(r) = L(r_1)$
  (el mismo lenguaje).

Observación (Reglas de precedencia)

Recuerde que las posibles ambigüedades a la hora de interpretar expresiones regulares se resuelven mediante las siguientes reglas de precedencia entre operadores:

- El operador * (cierre) es el de mayor precedencia.
- El siguiente es el operador de concatenación.
- El de menor precedencia es el operador $+$ (unión).

Ejemplo

Aplicando las reglas anteriores, la ER $r = a + bc^*$ debe entenderse como $a + (b(c)^*)$, puesto que primero se ha de aplicar el operador de cierre, $(c)^*$, luego el de concatenación, $(b(c)^*)$, y por último el de unión, obteniéndose finalmente la expresión $a + (b(c)^*)$, con lenguaje asociado $L(r) = L(a) \cup L(b)L(c)^* = \{a\} \cup \{b\}\{\lambda, c, cc, ccc, ...\} = \{a, b, bc, bcc, bccc, ...\}$.

ERs PARA ALGUNOS LENGUAJES DISTINGUIDOS, $\Sigma = \{a, b\}$

$L = \emptyset = \{\}$ (lenguaje vacío)

Por la definición de $L(r)$, la ER tal que $L(r) = \emptyset$ (es decir, la ER que describe el lenguaje vacío) es

$$r = \emptyset$$

$L = \{\lambda\}$ (lenguaje que solo contiene la palabra vacía)

De nuevo por definición, la ER tal que $L(r) = \{\lambda\}$ (es decir, la ER que describe el lenguaje compuesto exclusivamente por la palabra vacía) es

$$r = \lambda$$

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

La ER que describe el lenguaje formado solo por los símbolos a y b es

$$r = a + b$$

En efecto, de acuerdo con la definición resulta
$L(a + b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$.

$L = \Sigma^* = \{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, ...\}$
(lenguaje universal)

Para describir el lenguaje $\{a, b\}^*$, formado por todas las palabras sobre $\{a, b\}$, incluida la palabra vacía, basta la expresión

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

En efecto: $L((a + b)^*) = (L(a + b))^* = \{a, b\}^*$.

$L = \Sigma^+ = \Sigma^* - \{\lambda\} = \{a, b, aa, ab, ...\}$ (lenguaje universal positivo)

El lenguaje $\{a, b\}^+$ contiene todas las palabras *no vacías* sobre Σ, por lo que para construir una expresión regular que lo describa hay que imponer la presencia de al menos un símbolo -cualquiera- del alfabeto, lo cual se traduce en la expresión $a + b$, y concatenar dicho símbolo con palabras de cualquier tamaño, incluida la palabra vacía, lo cual se consigue con $(a + b)^*$. Por lo tanto, una ER que describe $\{a, b\}^+$ es

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

o, equivalentemente,

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

ERs PARA LENGUAJES CON CARACTERÍSTICAS TÍPICAS, $\Sigma = \{a, b\}$

Longitud de las palabras ($|w|$ = longitud de la palabra w)

$L = \{w \in \Sigma^* : |w| = m\}$
(palabras con longitud igual a $m \in \{0, 1, 2, 3, ...\}$)

- Con $m = 0$ se recupera el lenguaje $L = \{\lambda\}$, descrito por $r = \lambda$.
- Con $m = 1$ se recupera el lenguaje $L = \Sigma = \{a, b\}$, descrito por $r = a + b$.
- Para $m \geq 2$, basta con concatenar m símbolos, donde cada símbolo puede ser cualquiera de los símbolos del alfabeto:

$$r = \underbrace{(a + b) \cdots (a + b)}_{m \text{ veces}}$$

$L = \{w \in \Sigma^* : |w| \leq m\}$ (palabras con longitud menor o igual que $m \in \{0, 1, 2, 3, ...\}$)

Las palabras con longitud menor o igual que un cierto m son aquellas con tamaños $0, 1, \cdots, m$, por lo que se tiene:

$$L = \bigcup_{i=0}^{m} \{w \in \Sigma^* : |w| = i\}$$

Basta entonces con aplicar lo visto en el punto anterior para obtener la siguiente ER para describir L en el caso $m \geq 2$:

$$r = \underbrace{\lambda}_{i=0} + \underbrace{(a+b)}_{i=1} + \cdots + \underbrace{(a+b)\cdots(a+b)}_{m \text{ veces } (i=m)}$$

Otra posible ER, equivalente, sería

$$r = \underbrace{(\lambda + a + b)\cdots(\lambda + a + b)}_{m \text{ veces}}$$

$L = \{w \in \Sigma^* : |w| \geq m\}$
(palabras con longitud mayor o igual que $m \in \{0, 1, 2, 3, ...\}$)

Para describir todas las palabras con longitud mayor o igual que un cierto m basta con colocar m símbolos -cualesquiera- del alfabeto y concatenar (antes o después) una palabra cualquiera -incluida la vacía- sobre Σ, mediante $(a + b)^*$.

$$r = \underbrace{(a + b) \cdots (a + b)}_{m \text{ veces}}(a + b)^*$$

o, equivalentemente,

$$r = (a + b)^* \underbrace{(a + b) \cdots (a + b)}_{m \text{ veces}}$$

$L = \{w \in \Sigma^* : |w| \neq m\}$
(palabras con longitud distinta a $m \in \{0, 1, 2, 3, ...\}$)

Dado que

$$\{w \in \Sigma^* : |w| \neq m\} = \{w \in \Sigma^* : |w| \leq m-1\} \cup \{w \in \Sigma^* : |w| \geq m+1\}$$

una ER para el lenguaje dado sería la unión de dos de las ERs propuestas en puntos anteriores:

$$r = \lambda + (a+b) + \cdots + \underbrace{(a+b)\cdots(a+b)}_{m-1 \text{ veces}} + \underbrace{(a+b)\cdots(a+b)}_{m+1 \text{ veces}}(a+b)^*$$

$L = \{w \in \Sigma^* : |w| \text{ MOD } 2 = 0\}$ (palabras con longitud par)

Para conseguir palabras con longitud par basta con colocar los símbolos de dos en dos, repitiendo tantas veces como se quiera (incluida ninguna):

$$r = \Big((a+b)(a+b)\Big)^*$$

lo cual es equivalente a

$$r = \Big(aa + ab + ba + bb\Big)^*$$

$L = \{w \in \Sigma^* : |w| \text{ MOD } m = 0\}$
(palabras cuya longitud es un múltiplo de m, $m \in \{2, 3, 4, ...\}$)

- Con $m = 2$ se recupera el conjunto formado por las palabras con longitud par estudiado previamente.
- Con $m = 3$: conjunto formado por todas las palabras con longitudes 0, 3, 6, 9, etc.
- En general, basta con colocar símbolos de m en m, tantas veces como se quiera (incluida ninguna).

$$r = \Big(\underbrace{(a+b)\cdots(a+b)}_{m \text{ veces}} \Big)^*$$

$L = \{w \in \Sigma^* : |w| \text{ MOD } 2 = 1\}$ (palabras con longitud impar)

Las palabras con longitud impar se construyen fácilmente añadiendo un símbolo a cualquier palabra de longitud par, y por lo tanto, teniendo en cuenta lo visto anteriormente, una ER que describe estas palabras sería

$$r = \underbrace{(a+b)}_{\text{un símbolo}} \underbrace{\Big((a+b)(a+b)\Big)^*}_{\text{long. par}} = \underbrace{\Big((a+b)(a+b)\Big)^*}_{\text{long. par}} \underbrace{(a+b)}_{\text{un símbolo}}$$

o también, equivalentemente,

$$r = (a+b)\Big(aa+ab+ba+bb\Big)^* = \Big(aa+ab+ba+bb\Big)^*(a+b)$$

$$L = \{w \in \Sigma^* : |w| \text{ MOD } m = i\}, i \in \{0, , \cdots, m-1\}$$

- Con $i = 0$ se recupera el conjunto de palabras con longitud múltiplo de m.
- Con $m = 2, i = 1$ se recupera el conjunto formado por las palabras con longitud impar.
- Con $m = 3, i = 1$: conjunto formado por todas las palabras con longitudes 1, 4, 7, 10, etc.
- Con $m = 3, i = 2$: conjunto formado por todas las palabras con longitudes 2, 5, 8, 11, etc.
- En general, basta con colocar símbolos de m en m, tantas veces como se quiera (incluida ninguna), y añadir i símbolos más.

$$r = \Big(\underbrace{(a+b)\cdots(a+b)}_{m \text{ veces}} \Big)^* \underbrace{(a+b)\cdots(a+b)}_{i \text{ veces}}$$

$L = \{w \in \Sigma^* : |w| \text{ MOD } m \neq 0\}$
(palabras cuya longitud no es un múltiplo de m, $m \in \{2, 3, 4, ...\}$)

- $|w| \text{ MOD } m \neq 0$ es equivalente a $|w| \text{ MOD } m = i$ para algún $i \in \{1, \cdots, m-1\}$, por lo que se tiene:

$$L = \bigcup_{i=1}^{m-1} \{w \in \Sigma^* : |w| \text{ MOD } m = i\}$$

- Basta entonces con aplicar lo visto en el punto anterior para obtener la siguiente ER para describir L:

$$r = \Big(\underbrace{(a+b)\cdots(a+b)}_{m \text{ veces}} \Big)^* \Big(\underbrace{(a+b)}_{i=1} + \cdots + \underbrace{(a+b)\cdots(a+b)}_{i=m-1} \Big)$$

Apariciones de un cierto símbolo ($n_a(w)$ = número de apariciones de $a \in \Sigma$ en la palabra w)

- Estas ERs se diseñan de forma similar a las descritas previamente en las que intervenía la longitud de las palabras.

- La diferencia fundamental es que ahora no hay que contar las apariciones de cualquier símbolo sino solo las del símbolo dado.

$$L = \{w \in \Sigma^* : n_a(w) = m\},\ m \in \{0, 1, 2, ...\}$$

- Si $m = 0$, L está formado por todas aquellas palabras que no contienen ninguna a, por lo que la ER que lo describe sería

$$r = b^*$$

- Si $m \geq 1$, hay que colocar exactamente m a's, pero teniendo en cuenta que estas pueden aparecer en cualquier lugar, es decir, que puede haber cualquier cantidad de b's entre las a's, al principio de la palabra o al final de la misma. Así, una posible ER describiendo L sería:

$$r = \underbrace{b^*ab^* \cdots b^*ab^*}_{m\ a's}$$

$L = \{w \in \Sigma^* : n_a(w) \leq m\}, m \in \{0, 1, 2, 3, ...\}$

Las palabras con una cantidad de a's menor o igual que un cierto m son aquellas que tienen exactamente o 0, o 1 o $\cdots$ m a's, por lo que se tiene:

$$L = \bigcup_{i=0}^{m} \{w \in \Sigma^* : n_a(w) = i\}$$

Basta entonces con aplicar lo visto en el punto anterior para obtener la siguiente ER para describir L:

$$r = \underbrace{b^*}_{0\ a's} + \underbrace{b^*ab^*}_{1\ a} + \cdots + \underbrace{b^*ab^* \cdots b^*ab^*}_{m\ a's}$$

Otra posible ER, equivalente, sería

$$r = \underbrace{b^*(a+\lambda)b^* \cdots b^*(a+\lambda)b^*}_{m\ (a+\lambda)'s}$$

$L = \{w \in \Sigma^* : n_a(w) \geq m\}, m \in \{0, 1, 2, 3, ...\}$

Para describir todas las palabras con una cantidad de *a*'s mayor o igual que un cierto *m* basta con colocar *m* *a*'s, rodeadas de cualquier cantidad de *b*'s, y concatenar el resultado con una palabra cualquiera -incluida la vacía- sobre Σ (que se podría colocar tanto al principio de la ER como al final).

$$r = \underbrace{b^*ab^* \cdots b^*ab^*}_{m\ a's}(a+b)^* = (a+b)^* \underbrace{b^*ab^* \cdots b^*ab^*}_{m\ a's}$$

Otra posible ER, equivalente a la anterior, sería

$$r = \underbrace{(a+b)^*a(a+b)^* \cdots (a+b)^*a(a+b)^*}_{m\ a's}$$

(*m* *a*'s obligatorias, rodeadas de palabras cualesquiera sobre el alfabeto, que podrán contener, o no, más *a*'s).

$L = \{w \in \Sigma^* : n_a(w) \neq m\}, m \in \{0, 1, 2, ...\}$

Dado que

$$\begin{aligned} \{w \in \Sigma^* : n_a(w) \neq m\} &= \{w \in \Sigma^* : n_a(w) \leq m-1\} \\ &\cup \{w \in \Sigma^* : n_a(w) \geq m+1\} \end{aligned}$$

una ER para el lenguaje dado sería la unión de dos de las ERs propuestas en puntos anteriores:

$$\begin{aligned} r = & \underbrace{b^*}_{0\ a's} + \underbrace{b^*ab^*}_{1\ a} + \cdots + \underbrace{b^*ab^* \cdots b^*ab^*}_{m-1\ a's} \\ & + \underbrace{b^*ab^* \cdots b^*ab^*}_{m+1\ a's}(a+b)^* \end{aligned}$$

$L = \{w \in \Sigma^* : n_a(w) \text{ MOD } 2 = 0\}$

De forma similar al caso de las palabras de longitud par, para describir palabras que contengan un número par de *a*'s bastará con colocar siempre las *a*'s de dos en dos, tantas veces como se quiera, aunque rodeadas de tantas *b*'s como se quiera y en cualquier lugar (antes, después, o entre las *a*'s). Una primera aproximación sería $(b^*ab^*ab^*)^*$.

La ER anterior es correcta (todas sus palabras tienen un número par de *a*'s) pero es incompleta, porque, salvo la palabra vacía λ, no permite generar ninguna otra palabra con cero *a*'s, como *b*, *bb*, etc, que claramente forman parte de *L*. Lo anterior se consigue de varias formas, equivalentes, como por ejemplo

$$r = b^* + \Big(\underbrace{b^*ab^*ab^*}_{2a's}\Big)^* = \Big(\underbrace{b^*ab^*a}_{2a's}\Big)^* b^* = b^*\Big(\underbrace{ab^*ab^*}_{2a's}\Big)^* = \Big(b + ab^*a\Big)^*$$

$$L = \{w \in \Sigma^* : n_a(w) \text{ MOD } m = 0\},\ m \in \{2, 3, 4, ...\}$$

Este lenguaje contiene todas las palabras con un número de a's múltiplo de m. Con $m = 2$ se recupera el caso anterior. En general, basta con colocar a's de m en m, tantas veces como se quiera, sin olvidar incluir aquellas que tienen cero a's.

$$r = b^* + \Big(\underbrace{b^*ab^* \cdots b^*ab^*}_{m\ a's}\Big)^* = \Big(\underbrace{b^*ab^* \cdots b^*ab^*}_{m\ a's}\Big)^* + b^*$$

o, equivalentemente,

$$r = b^*\Big(\underbrace{ab^* \cdots b^*ab^*}_{m\ a's}\Big)^* = \Big(\underbrace{b^*ab^* \cdots b^*a}_{m\ a's}\Big)^* b^* = \Big(b + \underbrace{ab^* \cdots b^*a}_{m\ a's}\Big)^*$$

$$L = \{w \in \Sigma^* : n_a(w) \text{ MOD } 2 = 1\}$$

Una forma de obtener palabras con un número impar de *a*'s es partir de las palabras con un número par de *a*'s, cuya ER se ha comentado previamente, y añadir una *a* adicional, rodeada de tantas *b*'s como se quiera:

$$r = \underbrace{(b^*ab^*ab^*)^*}_{\text{nº par de } a's} \underbrace{b^*ab^*}_{1\ a}$$

o, equivalentemente,

$$r = \underbrace{b^*ab^*}_{1\ a} \underbrace{(b^*ab^*ab^*)^*}_{\text{nº par de } a's}$$

$$L = \{w \in \Sigma^* : n_a(w) \text{ MOD } m = i\}, i \in \{1, \cdots, m-1\}$$

- El caso $i = 0$, que genera el conjunto de palabras con número de a's múltiplo de m, se ha tratado anteriormente.
- Con $m = 2, i = 1$ se recupera el conjunto formado por las palabras con un número de a's impar.
- Con $m = 3, i = 1$: conjunto formado por todas las palabras con 1, 4, 7, 10, ... a's.
- En general, para $i \neq 0$, basta con colocar a's de m en m, tantas veces como se quiera (incluida ninguna), y añadir i símbolos a adicionales, permitiendo siempre la colocación de tantas b's como se desee delante, en medio o detrás de las a's.

$$r = (\underbrace{b^*ab^* \cdots b^*ab^*}_{m\ a's})^* \underbrace{b^*ab^* \cdots b^*ab^*}_{i\ a's}$$

$$= \underbrace{b^*ab^* \cdots b^*ab^*}_{i\ a's}(\underbrace{b^*ab^* \cdots b^*ab^*}_{m\ a's})^*$$

$L = \{w \in \Sigma^* : n_a(w) \text{ MOD } m \neq 0\}$

- $n_a(w)$ MOD $m \neq 0$ es equivalente a $n_a(w)$ MOD $m = i$ para algún $i \in \{1, \cdots, m-1\}$, por lo que se tiene:

$$L = \bigcup_{i=1}^{m-1} \{w \in \Sigma^* : n_a(w) \text{ MOD } m = i\}$$

- Basta entonces con aplicar lo visto en el punto anterior para obtener la siguiente ER para describir L:

$$r = (\underbrace{b^*ab^* \cdots b^*ab^*}_{m\ a's})^* \left(\underbrace{b^*ab^*}_{i=1} + \cdots + \underbrace{b^*ab^* \cdots b^*ab^*}_{i=m-1} \right)$$

Comienzo de las palabras

- Se trata de lenguajes compuestos exclusivamente por las palabras que empiezan, o las que no empiezan, por una cierta cadena, por ejemplo:
  - "palabras que empiezan por *aba*".
  - "palabras que no empiezan por *aa*".
- En el caso general: los primeros se pueden escribir matemáticamente como

$$L_{x..} = \{xw \ : \ w \in \Sigma^*\}, \text{ con } x = a_1...a_m \in \Sigma^+ (m \geq 1)$$

donde $L_{x..}$ es el lenguaje formado por todas las palabras de Σ^* que empiezan por x, mientras que los segundos son sus complementarios, $\Sigma^* - L_{x..}$, lenguajes formados por todas las palabras de Σ^* que no empiezan por x.

- El diseño de ERs para los lenguajes del tipo $L_{x..}$ es trivial, puesto que basta con colocar la palabra x y concatenar a continuación una palabra cualquiera del lenguaje Σ^*. Es decir, si $\Sigma = \{a, b\}$, una ER que describe $L_{x..}$ es

$$r = x(a + b)^*$$

- El diseño de ERs para sus complementarios, formados por todas las palabras que *no* empiezan por una cierta palabra, no es tan fácil. Una posible solución es describir estos lenguajes como uniones de lenguajes que *sí* empiezan por ciertas palabras, y aplicar la técnica anterior para diseñar ERs para estos últimos.

- Ambos casos se ilustran a continuación mediante ejemplos.

$L = \{abaw \ : \ w \in \Sigma^*\}$
(palabras que empiezan por aba)

$$r = aba(a + b)^*$$

$L = \Sigma^* - \{aaw \ : \ w \in \Sigma^*\}$
(palabras que no empiezan por aa)

Con $\Sigma = \{a, b\}$, las palabras que no empiezan por aa tienen que pertenecer necesariamente a uno de los siguientes lenguajes:

- El lenguaje formado por todas las palabras de Σ^* que empiezan por b, que se puede describir mediante la ER

$$r_1 = b(a + b)^*$$

- El lenguaje formado por todas las palabras de Σ^* que empiezan por *a* pero no por *aa*. Este lenguaje incluye a todas aquellas palabras que empiezan por *ab*, pero no hay que olvidar a la palabra *a*, que también empieza por *a* y no por *aa*. Este conjunto de palabras se puede describir mediante la ER

$$r_2 = ab(a+b)^* + a$$

- Una última palabra que desde luego no empieza por *aa* es la palabra vacía, descrita mediante la ER

$$r_3 = \lambda$$

- Cualquier palabra formada por *a*'s y *b*'s que no empieza por *aa* tiene que pertenecer a alguno de los tres grupos anteriores.

Por tanto, una ER que describe el lenguaje *L* sería

$$\begin{aligned} r = r_1 + r_2 + r_3 &= b(a+b)^* + ab(a+b)^* + a + \lambda \\ &= (b+ab)(a+b)^* + a + \lambda \end{aligned}$$

Final de las palabras

- Se trata de lenguajes compuestos exclusivamente por las palabras que terminan, o las que no terminan, por una cierta cadena, por ejemplo:
  - "palabras que terminan por *aa*".
  - "palabras que no terminan por *aba*".
- En el caso general: los primeros se pueden escribir matemáticamente como

$$L_{..x} = \{wx \ : \ w \in \Sigma^*\}, \text{ con } x = a_1...a_m \in \Sigma^+ (m \geq 1)$$

donde $L_{..x}$ es el lenguaje formado por todas las palabras de Σ^* que terminan por x, mientras que los segundos son sus complementarios, $\Sigma^* - L_{..x}$, lenguajes formados por todas las palabras de Σ^* que no terminan por x.

- El diseño de ERs para los lenguajes del tipo $L_{..x}$ es trivial, puesto que basta con colocar la palabra x y concatenar delante de ella una palabra cualquiera del lenguaje Σ^*. Es decir, si $\Sigma = \{a, b\}$, una ER que describe $L_{..x}$ es

$$r = (a + b)^* x$$

- Al igual que ocurre con el diseño de palabras del tipo "*no empiezan por*", una posible forma de diseñar ERs para lenguajes formados por palabras que "*no terminan por*" es describirlos como uniones de lenguajes que *sí* terminan por ciertas palabras, y aplicar la técnica anterior para diseñar ERs para estos últimos.

- Ambos casos se ilustran a continuación mediante ejemplos.

$L = \{waa : w \in \Sigma^*\}$ (palabras que terminan por aa)

$$r = (a + b)^* aa$$

$L = \Sigma^* - \{waba : w \in \Sigma^*\}$
(palabras que no terminan por aba)

Con $\Sigma = \{a, b\}$, las palabras que no terminan por aba tienen que pertenecer necesariamente a uno de los siguientes lenguajes:

- El lenguaje formado por todas las palabras de Σ^* que terminan por b, que se puede describir mediante la ER

$$r_1 = (a + b)^* b$$

- El lenguaje formado por todas las palabras de Σ^* que terminan por *a* pero no por *ba*. Este lenguaje incluye a todas aquellas palabras que terminan por *aa* así como a la palabra *a*. Este conjunto de palabras se puede describir mediante la ER

$$r_2 = (a+b)^* aa + a$$

- El lenguaje formado por todas las palabras de Σ^* que terminan por *ba* pero no por *aba*. Este lenguaje incluye a todas aquellas palabras que terminan por *bba* así como a la palabra *ba*. Este conjunto de palabras se puede describir mediante la ER

$$r_3 = (a+b)^* bba + ba$$

- Una última palabra que desde luego no termina por *aba* es la palabra vacía, descrita mediante la ER

$$r_4 = \lambda$$

- Cualquier palabra formada por *a*'s y *b*'s que no termina por *aba* tiene que pertenecer a alguno de los cuatro grupos anteriores.

Por tanto, una ER que describe el lenguaje *L* sería

$$\begin{aligned} r &= r_1 + r_2 + r_3 + r_4 \\ &= (a+b)^* b + (a+b)^* aa + a + (a+b)^* bba + ba + \lambda \\ &= (a+b)^* (b + aa + bba) + a + ba + \lambda \end{aligned}$$

Contenido de las palabras

- Se trata de lenguajes compuestos exclusivamente por las palabras que contienen, o las que no contienen, a una cierta cadena, por ejemplo:
  - "palabras que contienen a *ab*".
  - "palabras que no contienen a *aab*".
- En el caso general: los primeros se pueden escribir matemáticamente como

$$L_{..x..} = \{w_1 x w_2 \ : \ w_1, w_2 \in \Sigma^*\}, \text{ con } x = a_1 ... a_m \in \Sigma^+ (m \geq 1)$$

donde $L_{..x..}$ es el lenguaje formado por todas las palabras de Σ^* que contienen a x, mientras que los segundos son sus complementarios, $\Sigma^* - L_{..x..}$, lenguajes formados por todas las palabras de Σ^* que no contienen a x.

- El diseño de ERs para los lenguajes del tipo $L_{..x..}$ es trivial, puesto que basta con colocar la palabra x y concatenar delante y detrás de ella sendas palabras cualesquiera del lenguaje Σ^*. Es decir, si $\Sigma = \{a, b\}$, una ER que describe $L_{..x..}$ es

$$r = (a + b)^* x (a + b)^*$$

- El diseño directo de ERs para lenguajes formados por palabras del tipo "*no contienen*" no siempre es tan fácil, como se ilustra a continuación con algunos ejemplos. Sí es más sencillo en cambio diseñar autómatas finitos para estos lenguajes, por lo que una forma alternativa para obtener una ER es diseñar un autómata finito y transformar dicho autómata en su ER equivalente aplicando alguno de los algoritmos existentes para este cometido.

$L = \{w_1 abw_2 \ : \ w_1, w_2 \in \Sigma^*\}$ (palabras que contienen a ab)

$$r = (a + b)^* ab(a + b)^*$$

$L = \Sigma^* - \{w_1 abw_2 \ : \ w_1, w_2 \in \Sigma^*\}$
(palabras que no contienen a ab)

Las palabras que no contienen a ab, siendo $\Sigma = \{a, b\}$, son aquellas en las que las b's solo pueden estar delante de las a's. Como puede haber cualquier número tanto de b's como de a's, una ER que describe L es

$$r = b^* a^*$$

$L = \Sigma^* - \{w_1 aaw_2 \ : \ w_1, w_2 \in \Sigma^*\}$ (palabras que no contienen dos *a*'s consecutivas)

Para describir el conjunto de palabras que no contienen *aa* hay que tener en cuenta lo siguiente:

- El símbolo *b* no interviene en la restricción, por lo que las *b*'s se pueden aceptar sin problemas.
- En cuanto al símbolo *a*, la forma de evitar que haya dos *a*'s consecutivas es proteger cada *a* con una *b* justo detrás de ella, es decir, no aceptar *a*'s sino *ab*'s.
- Como las *b*'s y las *ab*'s anteriores se pueden mezclar y repetir tantas veces como se quiera, una posible ER sería

$$r_1 = (b + ab)^*$$

- Claramente ninguna de las palabras descritas por la ER anterior contiene *aa* (la ER es correcta), pero sin embargo faltan palabras que sí deberían estar (la ER es incompleta), en particular palabras acabadas en *a* como por ejemplo $a, ba, abba, \ldots$. El problema es que r_1 solo contempla palabras acabadas en *b*, cuando no tiene por qué ser así. Una ER para las palabras que no contienen *aa* y acaban en *a* sería

$$r_2 = (b + ab)^* a$$

- Uniendo ambos casos, una ER para *L* sería

$$\begin{aligned} r &= r_1 + r_2 \\ &= (b + ab)^* + (b + ab)^* a \\ &= (b + ab)^* (\lambda + a) \end{aligned}$$

$L = \Sigma^* - \{w_1 aabw_2 \ : \ w_1, w_2 \in \Sigma^*\}$
(palabras que no contienen a *aab*)

Para describir el conjunto de palabras que no contienen a *aab* hay que tener en cuenta lo siguiente:

- El símbolo *b* se puede aceptar sin problemas.
- En cuanto al símbolo *a*, para evitar *aab* es necesario evitar que haya más de una *a* seguida (salvo que se trate del final de la palabra, ver punto más adelante). Para ello es necesario proteger cada aparición de una *a* con una *b* posterior, admitiendo *ab* en lugar de *a*.
- Como las *b*'s y las *ab*'s anteriores se pueden mezclar y repetir tantas veces como se quiera, una posible ER sería

$$r_1 = (b + ab)^*$$

- Claramente ninguna de las palabras descritas por la ER anterior contiene *aab* (la ER es correcta) pero sin embargo faltan aquellas que no contienen *aab* pero acaban en *a*, como por ejemplo *a*, *ba*, *abbaaa*, ... (la ER es incompleta). Una ER para las palabras que no contienen *aab* pero acaban con tantas *a*'s como se quiera, al menos una, sería

$$r_2 = (b + ab)^* aa^*$$

- Uniendo ambos casos, una ER para L sería

$$\begin{aligned} r &= r_1 + r_2 \\ &= (b + ab)^* + (b + ab)^* aa^* \\ &= (b + ab)^* (\lambda + aa^*) \\ &= (b + ab)^* a^* \end{aligned}$$

EJEMPLOS ADICIONALES

- A continuación se discute en detalle el diseño de expresiones regulares para algunos lenguajes regulares adicionales.

- En general, estas expresiones se diseñan adaptando o combinando algunas de las características típicas estudiadas previamente.

$L = \{w \in \{a,b\}^+ : |w| \text{ MOD } 3 = 0\}$

- L está formado por aquellas palabras de $\{a,b\}^*$ cuya longitud es un múltiplo de 3 $(0,3,6,9,...)$ pero excluyendo la de tamaño 0, la palabra vacía, puesto que $\{a,b\}^+$ es el cierre universal *positivo* de $\{a,b\}$.
- Basta entonces con aplicar el caso general visto anteriormente, colocando los símbolos de 3 en 3, pero obligando a que haya al menos 3. Una posible ER que describe el lenguaje dado es por lo tanto la siguiente:

$$r = \Big((a+b)(a+b)(a+b)\Big)^*(a+b)(a+b)(a+b)$$
$$= (a+b)(a+b)(a+b)\Big((a+b)(a+b)(a+b)\Big)^*$$

$L = \{a, b, c\}^* - \{abaw \ : \ w \in \{a, b, c\}^*\}$

L contiene a todas aquellas palabras formadas por a's, b's y c's que *no* empiezan por aba. Estas palabras tienen que pertenecer necesariamente a uno de los siguientes conjuntos:

- Palabra vacía λ.
- Palabras que empiezan por b o por c, descritas mediante $b(a+b+c)^* + c(a+b+c)^* = (b+c)(a+b+c)^*$.
- Palabras que empiezan por a pero no por ab, que son la palabra a más todas aquellas palabras que empiezan por aa o por ac: $a + (aa + ac)(a+b+c)^*$.
- Palabras que empiezan por ab pero no por aba, que son la propia ab más todas aquellas que empiezan por abb o por abc: $ab + (abb + abc)(a+b+c)^*$.

Uniendo todos los casos anteriores se obtiene la ER:

$$r = \lambda + a + ab + (b + c + aa + ac + abb + abc)(a+b+c)^*$$

$$L = \{a, b, c\}^+ - \{wab : w \in \{a, b, c\}^*\}$$

L contiene a todas aquellas palabras no vacías formadas por a's, b's y c's que *no* terminan por ab. Estas palabras tienen que pertenecer necesariamente a uno de los siguientes conjuntos:

- Palabras que terminan por a o por c, descritas mediante $(a+b+c)^*a + (a+b+c)^*c = (a+b+c)^*(a+c)$.
- Palabras que terminan por b pero no por ab, que son la palabra b más todas aquellas palabras que terminan por bb o por cb: $b + (a+b+c)^*(bb+cb)$.

Uniendo los dos casos anteriores se obtiene la ER:

$$r = b + (a+b+c)^*(a+c+bb+cb)$$

$L = \{w \in \{a, b, c\}^* \ : \ w \text{ no termina en } abab\}$

Las palabras de $\{a, b, c\}^*$ que no terminan en $abab$ tienen que estar necesariamente en alguno de los siguientes casos:

- Palabra vacía λ.
- Palabras que no terminan en b, y por lo tanto terminan en a o en c, descritas mediante

$$(a + b + c)^*(a + c)$$

- Palabras que terminan en b pero no en ab, y por lo tanto terminan en bb o en cb, sin olvidar a la palabra b:

$$b + (a + b + c)^*(bb + cb)$$

- Palabras que terminan en *ab* pero no en *bab*, y por lo tanto terminan en *aab* o en *cab*, sin olvidar a la palabra *ab*:

$$ab + (a + b + c)^*(aab + cab)$$

- Palabras que terminan en *bab* pero no en *abab*, y por lo tanto terminan en *bbab* o en *cbab*, sin olvidar a la palabra *bab*:

$$bab + (a + b + c)^*(bbab + cbab)$$

Uniendo todos los casos anteriores se obtiene la ER:

$$\begin{aligned} r \quad = \quad & \lambda + b + ab + bab \\ & + (a + b + c)^*(a + c + bb + cb + aab + cab + bbab + cbab) \end{aligned}$$

$$L = \{a, b, c\}^* - \{w_1 abw_2 \ : \ w_1, w_2 \in \{a, b, c\}^*\}$$

L consta de todas aquellas palabras formadas por a's, b's y c's que *no* contienen a ab. Conviene tener en cuenta lo siguiente:

- Para evitar la cadena ab es necesario proteger cada posible aparición de un símbolo a con otro símbolo posterior distinto a la b, que en este caso solo pueden ser a's o c's, por lo que se aceptarán cadenas a^*c.
- Las b's y c's se pueden colocar entonces sin problemas.
- Las cadenas anteriores se pueden repetir y mezclar tantas veces como se desee, lo cual se describe con la ER $(a^*c + b + c)^*$, que es equivalente a $(a^*c + b)^*$ puesto que $a^*c + c = a^*c$.
- A la ER anterior le faltan las palabras tales que, además de no contener ab, acaban con una o más a's.

Una posible ER para el lenguaje L sería entonces:

$$r = (a^*c + b)^*a^*$$

$L = \{w \in \{a,b\}^* : n_{ab}(w) \text{ MOD } 2 = 0\}$

L está formado por aquellas palabras de $\{a,b\}^*$ que contienen un número par de ocurrencias de la cadena ab, como por ejemplo $\lambda, (ab)^{2n}, a^n, b^n (n \geq 0), ba, aa, bba, abaab, babbaba, \cdots \in L$, que serán:

- Aquellas que no contienen ninguna cadena ab, que son tales que las b's solo pueden estar delante de las a's:

$$r_1 = b^* a^*$$

- Aquellas que contienen un número par no nulo de ab's, para lo cual basta con colocar las ab's de dos en dos, tantas veces como se quiera, teniendo en cuenta que las a's y las b's pueden ser más de una. Una primera aproximación para este subconjunto sería:

$$r_2 = \left(a^* \underbrace{ab}\, b^* a^* \underbrace{ab}\, b^* \right)^*$$

- La ER anterior es correcta (todas las palabras que define contienen un número par de *ab*'s) pero no es completa, porque todas sus palabras, salvo la vacía, empiezan necesariamente por *a* y terminan necesariamente por *b*, cosa que no tiene por qué ocurrir: por ejemplo, tanto *babab* como *ababaaa* pertenecen a L pero no pertenecen a $L(r_2)$. Este error se subsana permitiendo *b*'s delante y *a*'s detrás:

$$r_2 = b^* \left(a^* \underbrace{ab}\, b^* a^* \underbrace{ab}\, b^* \right)^* a^*$$

- Basta ahora con unir las dos expresiones regulares anteriores, constatando que la segunda incluye a la primera, para obtener la siguiente ER, que describe el lenguaje L dado:

$$r = r_1 + r_2 = b^* a^* + b^* \left(a^* abb^* a^* abb^* \right)^* a^*$$
$$= b^* \left(a^* abb^* a^* abb^* \right)^* a^*$$

Lenguaje $L \subseteq \{a, b, c\}^*$ formado por aquellas palabras tales que si contienen b's, el número de b's consecutivas siempre es 1 o estrictamente mayor que 2 (pero nunca 2)

Ejemplos:

- Ejemplos de palabras que pertenecen a L:

  $a^n, c^n (n \geq 0), b, b^n (n \geq 3), abcb, abbbacc, abcbbbbaaabbb \in L$

- Ejemplos de palabras que no pertenecen a L:

  $bb, bba, abcccbb, abcbba, cbabbaaabbcccc \notin L$

- No hay problema ni con las *a*'s ni con las *c*'s.
- Las *b*'s nunca pueden aparecer en grupos de exactamente dos. Para conseguir lo anterior:
  - Puede haber apariciones individuales de *b*'s pero siempre que vayan protegidas con otra letra distinta posterior (en este caso *a* o *c*) para evitar la aparición de dos *b*'s consecutivas.
  - Puede haber ristras de *b*'s pero siempre que tengan tamaño igual o superior a 3.
- Como las cadenas anteriores se pueden mezclar y repetir tantas veces como se quiera, una posible ER sería

$$r_1 = (a + c + ba + bc + bbbb^*)^*$$

- La expresión anterior es correcta (no contiene ninguna palabra con dos *b*'s juntas aisladas) pero es incompleta, porque faltan todas aquellas palabras construidas como indica r_1 pero acabadas en *b* (las de r_1 acaban todas, salvo la palabra vacía, en *a*, en *c* o con una ristra de tres o más *b*'s). El conjunto de palabras que faltan se puede describir mediante la ER

$$r_2 = (a + c + ba + bc + bbbb^*)^* b$$

- Uniendo las dos expresiones anteriores se obtiene la siguiente ER, que describe el lenguaje *L* dado:

$$\begin{aligned} r &= r_1 + r_2 \\ &= (a + c + ba + bc + bbbb^*)^* + (a + c + ba + bc + bbbb^*)^* b \\ &= (a + c + ba + bc + bbbb^*)^* (\lambda + b) \end{aligned}$$

$L = \{a^n x : n > 0, x \in \{a, b, c\}^*, x \text{ no contiene } ccc\}$

Descripción y ejemplos:

- Todas las palabras deben empezar necesariamente por a (puesto que es $n > 0$), lo cual excluye, en particular, a la palabra vacía.
- Después de la a inicial, se admite cualquier palabra que no contenga a ccc, por lo que el lenguaje dado es igual a $\{ax : x \in \{a, b, c\}^*, x \text{ no contiene } ccc\}$.
- Ejemplos de palabras que pertenecen a L:

$$a^n (n \geq 1), ab, ac, accb, aacbcc \in L$$

- Ejemplos de palabras que no pertenecen a L:

$$\lambda, b, c, bac, acccc, abccca, abcacccb \notin L$$

Para diseñar una ER para L hay que controlar que las palabras empiecen por a y que no contengan a la subcadena ccc. Las dos características anteriores han sido discutidas, por separado, en apartados anteriores. Se trata ahora por lo tanto de combinarlas.

- La primera condición se consigue trivialmente colocando el símbolo a al comienzo de la ER.
- Para cumplir la segunda condición es necesario concatenar al símbolo anterior una ER que describa todas las palabras que no contienen ccc, para lo cual:
  - No hay problema ni con la a ni con la b.
  - El símbolo c se puede colocar de uno en uno o como máximo de dos en dos, por lo que hay que proteger esas apariciones con un símbolo distinto (en este caso a o b) justo a continuación, salvo que se trate del final de la palabra.
- En definitiva, una ER para L podría ser:

$$r = a\Big(a + b + c(a + b) + cc(a + b)\Big)^*(\lambda + c + cc)$$

$$= a\Big((\lambda + c + cc)(a + b)\Big)^*(\lambda + c + cc)$$

$L = \{w \in \{a, b, c\}^* : n_b(w) \text{ MOD } 2 = 1 \text{ y } w \text{ no acaba en } ac\}$

Descripción y ejemplos:

- L está formado por aquellas palabras de $\{a, b, c\}^*$ que contienen un número impar de b's y que además no acaban en ac.
- La palabra vacía no pertenece al lenguaje, puesto que, aunque no acaba en ac, contiene un número par (cero) de b's.
- Ejemplos de palabras que pertenecen a L:

$$b^{2n+1}, bca^n cc, bc^n bba (n \geq 0), \cdots \in L$$
$$ba, bca, bcacbaab, babacb, \cdots \in L$$

- Ejemplos de palabras que no pertenecen a L:

$$b^{2n}(n \geq 0), bac, ac, acbaab, bcbac \cdots \notin L$$

- Dado que uno de los dos aspectos a tener en cuenta es si el número de *b*'s es par o impar, consideramos en primer lugar las dos ERs siguientes, construidas fácilmente de acuerdo con lo visto previamente:
  - Palabras de $\{a, b, c\}^*$ con un número par de *b*'s:

$$r_{bp} = \Big((a+c)^* b (a+c)^* b (a+c)^*\Big)^* (a+c)^*$$

  - Palabras de $\{a, b, c\}^*$ con un número impar de *b*'s:

$$r_{bi} = \Big((a+c)^* b (a+c)^* b (a+c)^*\Big)^* (a+c)^* b (a+c)^*$$

- La segunda restricción, que las palabras no acaben en *ac*, implica, como se ha visto en un apartado anterior, que las palabras deben acabar en *a*, en *b* o en *c*, excluyendo en este último caso las acabadas en *ac*.

- Aparecen por lo tanto los siguientes casos:
  - Palabras con un número impar de *b*'s acabadas en *a*:

$$r_{bi}a$$

  - Palabras con un número impar de *b*'s acabadas en *b*. Para conseguir un número total de *b*'s impar, delante de la *b* final debe haber un número par de *b*'s:

$$r_{bp}b$$

  - Palabras con un número impar de *b*'s acabadas en *bc* o en *cc* (se excluye la palabra *c* ya que, aunque acaba en *c* y no en *ac*, no contiene un número impar de *b*'s):

$$r_{bp}bc + r_{bi}cc$$

- En definitiva, una ER para *L* podría ser:

$$r = r_{bi}a + r_{bp}b + r_{bp}bc + r_{bi}cc = r_{bi}(a + cc) + r_{bp}(b + bc)$$

$L = \{w \in \{a, b, c\}^* : n_b(w) \text{ MOD } 2 = 1 \text{ o } w \text{ no acaba en } ac\}$

- Este ejemplo es similar al anterior salvo que ahora las condiciones van unidas por una disyunción (**o**) en lugar de por una conjunción (**y**), por lo que el lenguaje se puede ver como la *unión* de dos lenguajes:

$$\begin{aligned} L &= L_1 \cup L_2 \\ &= \{w \in \{a, b, c\}^* : n_b(w) \text{ MOD } 2 = 1\} \\ &\cup \{w \in \{a, b, c\}^* : w \text{ no acaba en } ac\} \end{aligned}$$

- Estos casos son más sencillos puesto que basta con diseñar, por separado, ERs para cada uno de los lenguajes que forman la unión y posteriormente sumarlas:

$$\Big(L = L_1 \cup L_2, L_1 = L(r_1), L_2 = L(r_2)\Big) \Rightarrow L = L(r) \text{ con } r = r_1 + r_2$$

- Una ER para L_1, de acuerdo con lo visto anteriormente, sería

$$r_1 = \Big((a+c)^* b (a+c)^* b (a+c)^*\Big)^* (a+c)^* b (a+c)^*$$

- Para diseñar una ER para L_2 hay que tener en cuenta que las palabras de $\{a, b, c\}^*$ que no acaban en ac son aquellas que acaban en a, en b, en bc o en cc, sin olvidar a la palabra vacía y a la palabra c:

$$r_2 = \lambda + c + (a+b+c)^* \Big(a + b + bc + cc\Big)$$

- Por lo tanto, una ER que describe el lenguaje L dado sería

$$\begin{aligned} r &= r_1 + r_2 \\ &= \Big((a+c)^* b (a+c)^* b (a+c)^*\Big)^* (a+c)^* b (a+c)^* \\ &+ \lambda + c + (a+b+c)^* \Big(a + b + bc + cc\Big) \end{aligned}$$

$L = \{w \in \{a, b, c\}^* : n_a(w) \geq 1, w \text{ contiene } bb\}$

Las palabras de L deben contener al menos una a y al menos dos b's seguidas, por lo que la ER debe imponer la aparición tanto de a como de bb, pero teniendo en cuenta lo siguiente:

- Las cadenas obligatorias a y bb pueden ir precedidas, seguidas o separadas por símbolos cualesquiera del alfabeto:

$$r_1 = (a + b + c)^* a (a + b + c)^* bb (a + b + c)^*$$

- Las cadenas obligatorias a y bb no tienen por qué aparecer en ese orden (observe por ejemplo que la palabra bba, que pertenece a L, no está en el lenguaje descrito por r_1), por lo que es necesario tener en cuenta además la ER

$$r_2 = (a + b + c)^* bb (a + b + c)^* a (a + b + c)^*$$

Una posible ER describiendo L sería entonces la unión de las dos ERs anteriores:

$$
\begin{aligned}
r &= r_1 + r_2 \\
&= (a+b+c)^* a (a+b+c)^* bb (a+b+c)^* \\
&\quad + (a+b+c)^* bb (a+b+c)^* a (a+b+c)^* \\
&= (a+b+c)^* \Big(a(a+b+c)^* bb + bb(a+b+c)^* a \Big) (a+b+c)^*
\end{aligned}
$$

$$L = \{w \in \{a, b, c\}^* : n_a(w) \geq 2, w \text{ no contiene } bb\}$$

Las palabras de L deben contener al menos dos a's y no deben contener dos b's seguidas:

- La condición de contener al menos dos a's se consigue colocando dos a's obligatorias en la ER, pero teniendo en cuenta que esas dos a's pueden ir rodeadas de otras palabras de $\{a, b, c\}^*$ siempre que cumplan la otra condición, es decir, no contengan más de dos b's seguidas.
- Una posible ER que describe L sería por lo tanto

$$r = r_1 a r_1 a r_1$$

donde r_1 describe el lenguaje

$$L_1 = \{w \in \{a, b, c\}^* : w \text{ no contiene } bb\}$$

Una posible ER describiendo L_1 se diseña teniendo en cuenta lo siguiente:

- No hay problema ni con la a ni con la c.
- Para evitar bb, el símbolo b se debe colocar siempre protegido con otro símbolo posterior distinto (en este caso una a o una c), salvo que se trate del último símbolo de la palabra.
- Una ER para L_1 podría ser entonces:

$$r_1 = \Big(a + c + b(a + c)\Big)^* (\lambda + b)$$

En definitiva, una ER para el lenguaje L dado sería

$$r = r_1 a r_1 a r_1, \text{ con } r_1 = \Big(a + c + b(a + c)\Big)^* (\lambda + b)$$

Lenguaje $L \subseteq \{a, b, 1\}^*$ formado por aquellas palabras que empiezan por letra y no contienen ni dos letras consecutivas ni cadenas de 1's consecutivos de longitud impar.

Entendiendo que a y b son letras y que 1 no lo es:

- Ejemplos de palabras que pertenecen a L:

$$a, b, a11, a11b, b1111b11, a11b11a, a111111 \in L$$

- Ejemplos de palabras que no pertenecen a L:

$$\lambda, 1, 11, 11a, aa, ba, a1b11, b111, a11bab11 \notin L$$

Para diseñar una ER que describa el lenguaje L hay que tener en cuenta lo siguiente:

- La condición relativa al comienzo de las palabras implica que la ER pedida debe ser de la forma $r = (a+b)r_1$, siendo r_1 una ER que no puede empezar por letra (para evitar dos letras consecutivas al comienzo) y cumpliendo el resto de condiciones.
- Además de no poder empezar por letra, r_1 no puede contener dos letras seguidas ni ristras de 1's de longitud impar. Lo anterior significa que las letras tienen que ir protegidas (por delante) por 1's y que los 1's solo se pueden poner de dos en dos, todo ello en el orden en el que se quiera y tantas veces como se quiera:

$$r_1 = (11a + 11b + 11)^* = \Big(11(a+b+\lambda)\Big)^*$$

- En definitiva, una posible ER describiendo L sería

$$r = (a+b)r_1 = (a+b)\Big(11(a+b+\lambda)\Big)^*$$

$L = \{c^p w c^q : p \leq 1, q \geq 0, w \in \{a,b\}^+, (p+q) \geq 1, (p+q) \text{ MOD } 2 = 0\}$

- Las subpalabras $w \in \{a,b\}^+$ se representan claramente, como se ha visto previamente, mediante la ER $(a+b)(a+b)^*$.
- La condición $p \leq 1$ significa que p solo puede tomar dos valores, $p = 0$ o $p = 1$. Veamos ambos casos.
- Si $p = 0$, como $c^0 = \lambda$ y $\lambda w = w$, las palabras de L empezarán por la subpalabra w. Por otro lado, las condiciones $(p+q) \geq 1$ y $(p+q)$ MOD $2 = 0$ se convierten en $q \geq 1$ y q MOD $2 = 0$, es decir, las c's del final deben aparecer un número par y no nulo de veces. Una ER para este caso sería

$$r_0 = (a+b)(a+b)^* cc(cc)^*$$

- Si $p = 1$ las palabras de L empezarán por la subpalabra cw. Por otro lado, las condiciones $(p + q) \geq 1$ y $(p + q)$ MOD $2 = 0$ se convierten en $q \geq 0$ y $(1 + q)$ MOD $2 = 0$, es decir, las c's del final deben aparecer un número impar de veces. Una ER para este caso sería

$$r_1 = c(a + b)(a + b)^* c(cc)^*$$

- En definitiva, teniendo en cuenta los dos posibles casos, una posible ER para el lenguaje L dado sería

$$\begin{aligned} r &= r_0 + r_1 \\ &= (a + b)(a + b)^* cc(cc)^* + c(a + b)(a + b)^* c(cc)^* \end{aligned}$$

BIBLIOGRAFÍA

- Este material docente está basado en fuentes diversas, en particular en los libros y recursos que se citan a continuación, de los que proceden muchos de los ejercicios que se discuten aquí.

- Se nutre también del trabajo de nuestros antiguos compañeros Jorge Martín Corujo, Holger Billhardt y José Miguel Buenaposada, a los que expresamos desde aquí nuestro agradecimiento.

- P. Linz, An Introduction to Formal Languages and Automata. Jones and Barlett Publishers, 4th ed., 2006.
- J.E. Hopcroft, R. Motwani, J.D. Ullman, Introducción a la Teoría de Autómatas, Lenguajes y Computación, Addison-Wesley Iberoamericana, 3ª edición, 2007.
- M. Alfonseca, J. Sancho, M. Martínez Orga, Teoría de Lenguajes, Gramáticas y Autómatas, Ediciones RAEC 1997.
- P. Isasi, P. Martínez, D. Borrajo. Lenguajes, gramáticas y autómatas. Un enfoque práctico. Addison-Wesley, 1997.
- M. Alfonseca, E. Alfonseca, A. Ortega, Teoría de Autómatas y Lenguajes formales, Mc. Graw Hill DL, 2007.

DISTRIBUCIÓN