eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Lenguajes, Computación y Sistemas Inteligentes

Grado en Ingeniería Informática de Gestión y Sistemas de Información

Escuela de Ingeniería de Bilbao (UPV/EHU)

2^o curso

Curso académico 2023-2024

Tema 8: Autómatas finitos: Lenguajes regulares

JOSÉ GAINTZARAIN IBARMIA

Departamento de Lenguajes y Sistemas Informáticos

Última actualización: 10 - 11 - 2023

Índice general

Índice de figuras

Índice de tablas

Tema 8

Autómatas finitos: Lenguajes regulares

8.1.

Introducción

Un AF define un lenguaje: el lenguaje formado por las palabras para las cuales el AF responde afirmativamente. Dicho de otra forma, el lenguaje de las palabras aceptadas por el AF.

Dado un lenguaje $\mathcal{L}$, un AF construido para ese lenguaje es un algoritmo mediante el cual se formaliza una estrategia para decidir si una palabra dada pertenece al lenguaje $\mathcal{L}$.

Además de conocer una estrategia para decidir si una palabra dada pertenece a $\mathcal{L}$, también es interesante conocer la estructura general de las palabras que pertenecen a $\mathcal{L}$. Se presentará un método para calcular dicha estructura general a partir de un AF. Veremos que la estructura de las palabras reconocidas por un AF se puede representar mediante una expresión regular. Habitualmente, abreviaremos "expresión regular" como ER.

Cada expresión regular (ER) representa un lenguaje, pero, adicionalmente, comprobaremos que a cada expresión regular le corresponde un AF. La conclusión es que los AF y las expresiones regulares son formalismos equivalentes: todo lenguaje definible mediante un AF es también definible mediante una expresión regular y, del mismo modo, todo lenguaje definible mediante una expresión regular es también definible mediante un AF. Se presentará un método de transformación para cada sentido: de AF a ER y de ER a AF.

Los lenguajes representables mediante las ER, es decir, mediante los AF, reciben el nombre de lenguajes regulares (LR). La duda que se plantea en este momento es si todos los lenguajes definibles sobre un alfabeto dado $\mathbb{A}$, son regulares. Haciendo uso de las ER y teniendo en cuenta que $\mathbb{H}^*$ es enumerable para cualquier alfabeto $\mathbb{H}$ y que $2^{\mathbb{A}^*}$ no es enumerable, vamos a poder probar que hay lenguajes definidos sobre el alfabeto $\mathbb{A}$ que no son regulares.

Dado un lenguaje regular $\mathcal{L}$, es interesante disponer de un algoritmo o una estrategia que sirva para ir generando palabras de $\mathcal{L}$. Un AF asociado a $\mathcal{L}$, sirve para, dada una palabra w, decidir si w pertenece a $\mathcal{L}$. Una ER asociada a $\mathcal{L}$ nos indica cuál es la estructura general de las palabras de $\mathcal{L}$. Ni los AF ni las ER están pensadas para generar palabras de $\mathcal{L}$ paso a paso. La estrategia necesaria para generar palabras de $\mathcal{L}$ se formaliza mediante una gramática. En el caso de las gramáticas necesarias para formalizar lenguajes definibles mediante un AF o una

ER, reciben el nombre de gramáticas regulares (GR).

Las gramáticas regulares y los AF son formalismos equivalentes: todo lenguaje definible mediante un AF es también definible mediante una gramática regular y, del mismo modo, todo lenguaje definible mediante una gramática regular es también definible mediante un AF. Se presentará un método de transformación para cada sentido: de AF a GR y de GR a AF.

Los AF, las ER y las GR son formalismos equivalentes que sirven para formalizar los LR: los AF se utilizan para verificar o comprobar si una palabra pertenece a un LR determinado, las ER se utilizan para expresar o documentar la estructura de las palabras que pertenecen a un LR determinado y las GR se utilizan para generar o derivar palabras de un LR determinado.

Pero algunos lenguajes no pueden ser representados mediante los AF —y tampoco mediante las ER y las GR, porque las ER y las GR son formalismos equivalentes a los AF. Al final de este tema, se probará, de manera formal, que hay lenguajes que no son regulares, es decir, que hay lenguajes no representables mediante los AF, las ER y las GR. Como consecuencia de este resultado teórico importante, se abre la necesidad de máquinas o formalismos con mayor capacidad: los autómatas de pila (AP) y las máquinas de Turing (MT).

Abreviaturas utilizadas en esta introducción:

- AF: Autómata Finito
- ER: Expresión Regular
- LR: Lenguaje Regular
- GR: Gramática Regular
- AP: Autómata de pila
- MT: Máquina de Turing

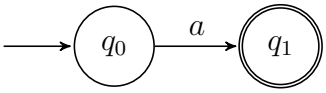
8.2.

Autómatas finitos y expresiones regulares

En este apartado se presentará un método que, dado un autómata finito, permite obtener de manera sistemática la expresión regular correspondiente que muestra la estructura de las palabras pertenecientes al lenguaje definido por el autómata finito.

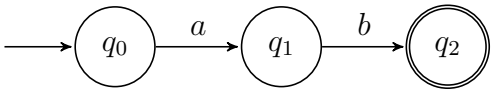
8.2.1 Información que aporta un AF sobre el lenguaje que define

Consideremos el siguiente AF:



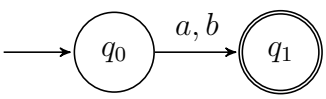
El lenguaje aceptado por ese AF es $\{a\varepsilon\}$. Por tanto, es un lenguaje que solo contiene una palabra. Un arco con un símbolo representa una palabra formada por un símbolo.

Sea el siguiente AF:

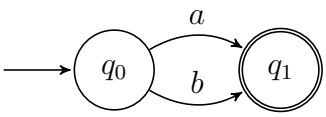


El lenguaje aceptado por ese AF es $\{a\varepsilon\} \circ \{b\varepsilon\}$, es decir $\{ab\varepsilon\}$. Por tanto, es también, un lenguaje de una sola palabra. Nótese que arcos consecutivos representan la concatenación.

Analizamos ahora el AF que se muestra a continuación:

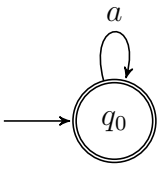


Ese AF es equivalente a este otro AF:



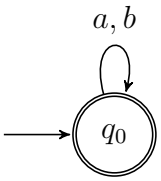
El lenguaje aceptado por ese AF es $\{a\varepsilon\} \cup \{b\varepsilon\}$, es decir, $\{a\varepsilon, b\varepsilon\}$. Es un lenguaje que contiene dos palabras. Cada arco es una palabra y mediante arcos que van de un mismo estado a otro mismo estado, se representan distintas palabras del lenguaje. La operación asociada es la unión de conjuntos: $\cup$.

El siguiente AF tiene un único estado y un bucle sobre ese estado:

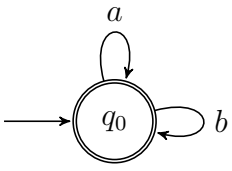


El bucle ofrece la opción de atravesarlo cero veces, una vez, dos veces, tres veces y así sin límite. Consecuentemente, las palabras ε (cero veces), $a\varepsilon$ (una vez), $aa\varepsilon$ (dos veces), $aaa\varepsilon$ (tres veces) etc. pertenecen al lenguaje: $\{\varepsilon, a\varepsilon, aa\varepsilon, aaa\varepsilon, \dots\}$. Un bucle representa la clausura universal. Por tanto, la manera formal de expresar el lenguaje definido por ese AF es $\{a\varepsilon\}^*$.

También el siguiente AF tiene un único estado, pero el bucle lleva dos símbolos:

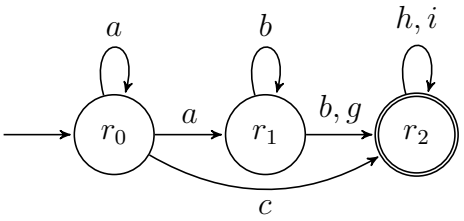


Ese AF es equivalente al siguiente AF:



En este autómata podemos elegir a y b todas las veces que queramos, incluso mezcládo las elecciones. Así, por ejemplo, $aaa\varepsilon$, $bbbbb\varepsilon$, ε, $baaba\varepsilon$ y $aaaba\varepsilon$ son palabras aceptadas por ese AF. La manera formal de expresar el lenguaje definido por ese AF es $(\{a\varepsilon\} \cup \{b\varepsilon\})^*$.

Cojamos un AF en el que aparecen todas las opciones que hemos considerado hasta ahora:



Las palabras del lenguaje asociado son aquellas cuya estructura permite terminar en r_2 con todos los símbolos consumidos tras haber partido desde r_0. Teniendo en cuenta lo visto en los ejemplos previos, tenemos lo siguiente:

$$\underbrace{((\{a\varepsilon\}^*) \circ \{a\varepsilon\} \circ (\{b\varepsilon\}^*) \circ (\underbrace{\{b\varepsilon\} \cup \{g\varepsilon\}}_{\substack{\text{dos}\\\text{alternativas}}}) \circ ((\underbrace{\{h\varepsilon\} \cup \{i\varepsilon\}}_{\substack{\text{dos}\\\text{alternativas}}})^*))}_{\text{recorrido superior}}$$

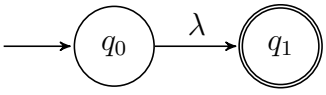
$$\cup$$

$$\underbrace{((\{a\varepsilon\}^*) \circ \{c\varepsilon\} \circ ((\underbrace{\{h\varepsilon\} \cup \{i\varepsilon\}}_{\substack{\text{dos}\\\text{alternativas}}})^*))}_{\text{recorrido inferior}}$$

Hay dos recorridos posibles desde r_0 a r_2. En el recorrido superior, las palabras tienen al menos una a al principio y, opcionalmente, pueden tener más apariciones de a. Después del bloque formado por al menos una a, se tiene un bloque formado por cero o más apariciones de b. A continuación, una b o una g es obligatoria. Al final se tiene un bloque compuesto por apariciones de h y de i. La longitud de este último bloque puede ser cero. Por ejemplo, $aabbb\varepsilon$, $aaag\varepsilon$, $aabbghhih\varepsilon$, $aabbgh\varepsilon$, $aabbgii\varepsilon$ y $ag\varepsilon$ tienen esa estructura. En el recorrido inferior, las palabras tienen un bloque inicial formado por cero o más apariciones de a. Luego, una c. Para terminar el recorrido inferior, se tiene un bloque compuesto por apariciones de h y de i, pero el bloque puede ser vacío. Por ejemplo, $c\varepsilon$, $aaac\varepsilon$, $aachhhh\varepsilon$, $aacii\varepsilon$, $aacihi\varepsilon$ y $chh\varepsilon$ se adecúan a la estructura del recorrido inferior. En cambio, palabras de la forma $bbb\varepsilon$, $gg\varepsilon$, $iiii\varepsilon$ o $cichhg\varepsilon$ no se adecúan ni al recorrido superior ni al inferior.

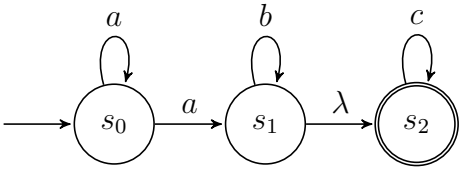
Para finalizar esta exposición de casos posibles, vamos a tratar dos opciones especiales: que un arco esté etiquetado mediante λ y que no haya arco entre dos estados.

Sea el siguiente AF:



La transición vacía —etiquetada mediante λ— indica que no hay símbolos y, consecuentemente, el lenguaje correspondiente es el lenguaje que solo contiene la palabra vacía: $\{\varepsilon\}$.

Consideremos un caso con más transiciones:



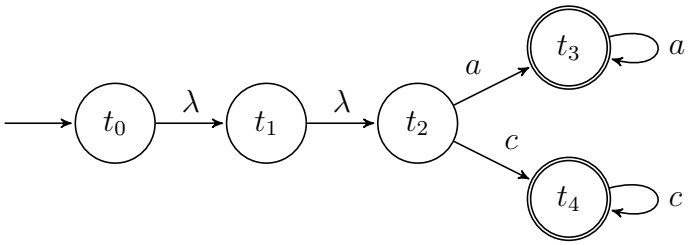
El lenguaje asociado se puede expresar de la siguiente forma:

$$(\{a\varepsilon\}^*) \circ \{a\varepsilon\} \circ (\{b\varepsilon\}^*) \circ \{\varepsilon\} \circ (\{c\varepsilon\}^*)$$

Como el lenguaje $\{\varepsilon\}$ es elemento neutro para la concatenación $\circ$, la expresión se puede simplificar:

$$(\{a\varepsilon\}^*) \circ \{a\varepsilon\} \circ (\{b\varepsilon\}^*) \circ (\{c\varepsilon\}^*)$$

El siguiente caso tiene dos transiciones vacías:



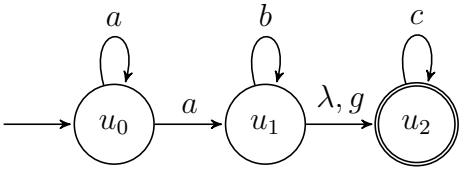
El lenguaje asociado se puede expresar de la siguiente forma:

$$\{\varepsilon\} \circ \{\varepsilon\} \circ ((\{a\varepsilon\} \circ (\{a\varepsilon\}^*)) \cup (\{c\varepsilon\} \circ (\{c\varepsilon\}^*)))$$

Al ser $\{\varepsilon\}$ elemento neutro para la concatenación $\circ$, la expresión se puede simplificar:

$$(\{a\varepsilon\} \circ (\{a\varepsilon\}^*)) \cup (\{c\varepsilon\} \circ (\{c\varepsilon\}^*))$$

Si tenemos el siguiente AF:

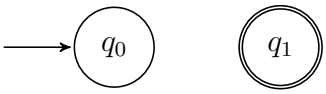


El lenguaje asociado se puede expresar de la siguiente forma:

$$(\{a\varepsilon\}^*) \circ \{a\varepsilon\} \circ (\{b\varepsilon\}^*) \circ (\{\varepsilon\} \cup \{g\varepsilon\}) \circ (\{c\varepsilon\}^*)$$

Como el lenguaje $\{\varepsilon\}$ no es elemento neutro para la unión $\cup$, la expresión no se puede simplificar.

Finalmente, consideramos el siguiente AF:



Puesto que no es posible llegar a q_1 partiendo desde q_0, no hay ninguna palabra en el lenguaje asociado a ese AF. Por tanto, ese AF representa el lenguaje vacío: $\varnothing$.

8.2.2 Expresiones regulares

En el apartado anterior se muestra, de manera informal, que cada símbolo α —del alfabeto— que aparece en un arco, representa el lenguaje formado por la palabra que solo contiene ese símbolo: $\{\alpha\varepsilon\}$. A partir de ese tipo de lenguajes básicos, los arcos en secuencia representan la concatenación ($\circ$) de lenguajes, los arcos en paralelo representan la unión ($\cup$) de lenguajes y los bucles representan la clausura universal ($*$). Los bucles, además, han de ser insertados en la secuencia de arcos mediante la concatenación. Por tanto, solo son necesarias esas tres operaciones para ir construyendo la expresión que muestra la estructura de las palabras que pertenecen al lenguaje asociado al AF. En cuanto a lenguajes necesarios para formular esas expresiones, solo se necesitan los lenguajes básicos que constan de una única palabra. Habrá un lenguaje básico $\{\alpha\varepsilon\}$ por cada símbolo α del alfabeto. Adicionalmente, se necesitan el lenguaje $\{\varepsilon\}$ para tratar con las transiciones etiquetadas con λ y el lenguaje $\varnothing$ para representar el lenguaje vacío.

Puesto que solo se requieren los lenguajes básicos mencionados que únicamente contienen una palabra formada por un único símbolo y los lenguajes adicionales $\{\varepsilon\}$ y $\varnothing$, en las expresiones regulares se simplifica la notación. En vez de escribir $\{\alpha\varepsilon\}$ se escribirá solo α, en vez de escribir $\{\varepsilon\}$ se escribirá solo ε. En lo que concierne a las operaciones, también se simplificará la notación: en vez de $\cup$ se escribirá $+$, la concatenación $\circ$ no se representará y la clausura universal $*$ se mantendrá tal cual. Además, se establecen unas prioridades entre los operadores. Ello hará que a veces los paréntesis no sean necesarios. Las prioridades, de mayor a menor son estas: La clausura universal (*), la concatenación (sin operador) y la unión ($+$). Esas prioridades se pueden modificar mediante el uso de paréntesis.

8.2.2.1 Definición formal de expresiones regulares

Dado un alfabeto $\mathbb{A}$, la noción de expresión regular sobre el alfabeto $\mathbb{A}$ se define recursivamente de la siguiente forma:

- $\varnothing$ es una expresión regular definida sobre $\mathbb{A}$.
- ε es una expresión regular definida sobre $\mathbb{A}$.
- Para cada símbolo α que pertenece al alfabeto $\mathbb{A}$, α es una expresión regular definida sobre $\mathbb{A}$.
- Si R es una expresión regular definida sobre $\mathbb{A}$, entonces (R^*) es una expresión regular definida sobre $\mathbb{A}$: clausura universal de R, es decir, R repetido todas las veces que se quiera.
- Si R_1 y R_2 son expresiones regulares definidas sobre $\mathbb{A}$, entonces (R_1R_2) es una expresión regular definida sobre $\mathbb{A}$: concatenación de R_1 y R_2.
- Si R_1 y R_2 son expresiones regulares definidas sobre $\mathbb{A}$, entonces $(R_1 + R_2)$ es una expresión regular definida sobre $\mathbb{A}$: unión o disyunción de R_1 y R_2.

Si el alfabeto es $\mathbb{A} = \{a, b, c\}$, tenemos que por ejemplo, a es una ER, que $(((bc)a)a)$ es una ER, que $(a(b^*))$ es una ER, que $(a((cb)^*))$ es una ER, que $((a((cb)^*)) + ((c(cc))^*))$ es una ER, que $((a((cb)^*)) + \varepsilon)$ es una ER y que $((a(c + b))^*)$ es una ER.

Los paréntesis complican, a veces, la escritura de las ER. Pero teniendo en cuenta que la concatenación es asociativa —es decir, $(R_1(R_2R_3)) \equiv ((R_1R_2)R_3)$— los paréntesis internos generados por la concatenación se pueden eliminar, dejando $(R_1R_2R_3)$. Teniendo en cuenta que $*$ tiene más prioridad que la concatenación y que $+$, la expresión (α^*) se puede simplificar como α^*, siendo α un símbolo del alfabeto. Pero, para una expresión regular R que no sea de la forma α —siendo α un símbolo del alfabeto—, en general no es posible quitar los paréntesis de (R^*). Por otro lado, teniendo en cuenta que la concatenación tiene más prioridad que $+$, $((R_1R_2) + R_3)$ se puede simplificar como $(R_1R_2 + R_3)$ y $(R_1 + (R_2R_3))$ se puede simplificar como $(R_1 + R_2R_3)$. Finalmente, los paréntesis externos también se pueden quitar.

Tras estas normas que permiten prescindir de algunos paréntesis, en vez de $(((bc)a)a)$ podemos escribir $bcaa$, en vez de $(a(b^*))$ podemos escribir ab^*, en vez de $(a((cb)^*))$ podemos escribir $a(cb)^*$, en vez de $((a((cb)^*)) + ((c(cc))^*))$ podemos escribir $a(cb)^* + (ccc)^*$, en vez de $((a((cb)^*))+\varepsilon)$ podemos ecribir $a(cb)^*+\varepsilon$ y en vez de $((a(c+b))^*)$ podemos escribir $(a(c+b))^*$. Estos ejemplos muestran que algunos paréntesis son imprescindibles.

8.2.2.2 Definición formal de lenguaje regular y ejemplos de lenguajes regulares

Dado un alfabeto $\mathbb{A}$, un lenguaje $\mathcal{L}$ definido sobre el alfabeto $\mathbb{A}$ es regular si es posible representar la estructura de todas las palabras de $\mathcal{L}$ mediante una expresión regular definida sobre el alfabeto $\mathbb{A}$.

A continuación, se muestran algunos ejemplos de lenguajes regulares. En cada caso, se da una expresión regular que expresa la estructura de todas las palabras del lenguaje en cuestión. Al formular una expresión que exprese la estructura de todas las palabras del lenguaje, queda demostrado que el lenguaje es regular. Por tanto, una manera de demostrar que un lenguaje es regular, es dando una expresión regular que exprese la estructura de todas las palabras del lenguaje.

En todos los ejemplos, consideramos el alfabeto $\mathbb{A} = \{a, b, c\}$.

1. El lenguaje L_1 formado por todas las palabras que no contienen ninguna aparición ni de b ni de c, es un lenguaje regular. Una manera de formular el lenguaje L_1 es la siguiente:

   $$L_1 = \{w | w \in \mathbb{A}^* \wedge |w|_b = 0 \wedge |w|_c = 0\}$$

   También es posible definirlo de la siguiente forma:

   $$L_1 = \{w | w \in \mathbb{A}^* \wedge |w| = |w|_a\}$$

   Y también es posible definirlo de la siguiente forma:

   $$L_1 = \{w | w \in \mathbb{A}^* \wedge \exists k(k \in \mathbb{N} \wedge w = (a\varepsilon)^k)\}$$

   Pero esas tres formalizaciones no sirven para demostrar que L_1 es regular. La expresión regular correspondiente a L_1 es la siguiente:

   $$a^*$$

   Esa expresión regular indica que la estructura de cualquier palabra de L_1 es una secuencia formada por cero o más repeticiones del símbolo a.

   Al haber dado una expresión regular que caracteriza todas las palabras de L_1, queda probado que L_1 es un lenguaje regular.

2. El lenguaje L_2 formado por todas las palabras no vacías formadas por cualquier número de repeticiones de a, es un lenguaje regular. L_2 puede definirse como $L_1 \setminus \{\varepsilon\}$. Como las palabras de L_2 han de tener al menos una a, la expresión regular correspondiente es la siguiente:

   $$aa^*$$

   Esa expresión regular indica que las palabras de L_2 tienen al menos una a y luego cualquier número de repeticiones —cero o más— de a.

3. El lenguaje $L_3 = \{a\varepsilon, b\varepsilon, c\varepsilon\}$, es regular, ya que la estructura de las palabras de L_3 puede ser expresada mediante la siguiente expresión regular:

   $$a + b + c$$

   El significado es que las palbras de L_3 están formadas por una a o por una b o por una c.

4. El lenguaje universal $L_4 = \mathbb{A}^*$, formado por todas las palabras que se pueden generar a partir del alfabeto $\mathbb{A}$, es regular porque la estructura de las palabras de L_4 puede ser expresada mediante la siguiente expresión regular:

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

Esa expresión regular indica que las palabras de L_4 están formadas por cualquier número de repeticiones —cero o más— de a, b y c y combinando las repeticiones de cualquier forma. Por ejemplo, $aaa\varepsilon$ pertenece a L_4 pero también pertenecen a L_4 las palabras $bbcaaacb\varepsilon$ y $cccbc\varepsilon$. Nótese que para expresar la estructura de las palabras de L_4, también sirven, por ejemplo, las expresiones regulares $(c + b + a)^*$, $(c + a + b)^*$ y $(b + a + c)^*$. Con $+$ el orden no es relevante.

5. El lenguaje L_5 formado por todas las palabras que no contienen ninguna b ni ninguna c y en las que el número de apariciones de a es par (ε, $aa\varepsilon$, $aaaa\varepsilon$, $aaaaaa\varepsilon$, etc.) es regular. Una manera de formular el lenguaje L_5 es la siguiente:

$$L_5 = \{w | w \in \mathbb{A}^* \wedge \exists k(k \in \mathbb{N} \wedge w = (aa\varepsilon)^k)\}$$

Otra manera de formular el lenguaje L_5 es la siguiente:

$$L_5 = \{w | w \in \mathbb{A}^* \wedge |w| = |w|_a \wedge |w| \ mod \ 2 = 0\}$$

Esas dos formalizaciones no prueban que L_5 es regular. La expresión regular que expresa la estructura de las palabras de L_5 es la siguiente:

$$(aa)^*$$

Esa expresión regular indica que las repeticiones de a siempre van de dos en dos: cero, dos, cuatro, seis, ocho, etc.

6. A continuación vamos a probar que el lenguaje L_6 formado por todas las palabras que empiezan por a es regular. Una manera de definir formalmente ese lenguaje es la siguiente:

$$L_6 = \{w | w \in \mathbb{A}^* \wedge \exists u(u \in \mathbb{A}^* \wedge w = au)\}$$

Otra manera de formular el lenguaje L_6 es la siguiente:

$$L_6 = \{w | w \in \mathbb{A}^* \wedge |w| \geq 1 \wedge w(1) = a\}$$

Pero para probar que es regular tenemos que dar una manera de definir L_6 mediante una expresión regular. La siguiente expresión regular expresa la estructura de todas las palabras de L_6:

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

Mediante esa expresión regular se indica que las palabras de L_6 tienen al principio una a y, luego, cualquier número de repeticiones de a, b y c combinados de cualquier manera.

7. A continuación vamos a probar que el lenguaje L_7 formado por todas las palabras que terminan en b es regular. Una manera de definir formalmente este lenguaje es la siguiente:

$$L_7 = \{w | w \in \mathbb{A}^* \wedge \exists u(u \in \mathbb{A}^* \wedge w = ub)\}$$

Otra manera de formular el lenguaje L_7 es la siguiente:

$$L_7 = \{w | w \in \mathbb{A}^* \wedge |w| \geq 1 \wedge w(|w|) = b\}$$

Pero esas definiciones no sirven para probar que L_7 es regular. Para probar que es regular tenemos que dar una expresión regular que expresa la estructura de todas las palbras de L_7. La siguiente expresión regular expresa la estructura de todas las palabras de L_7:

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

Mediante esa expresión regular se indica que las palabras de L_7 tienen, al principio, cualquier número de repeticiones de a, b y c combinados de cualquier manera y, para terminar, una b.

8. A continuación consideramos el lenguaje L_8 formado por todas las palabras que empiezan por a y terminan en b es regular. Una manera de definir formalmente este lenguaje es la siguiente:

$$L_8 = \{w | w \in \mathbb{A}^* \wedge \exists u(u \in \mathbb{A}^* \wedge w = aub)\}$$

Otra manera de formular el lenguaje L_8 es la siguiente:

$$L_8 = \{w | w \in \mathbb{A}^* \wedge |w| \geq 2 \wedge w(1) = a \wedge w(|w|) = b\}$$

La siguiente expresión regular expresa la estructura de todas las palabras de L_8:

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

Mediante esa expresión regular se indica que las palabras de L_8 tienen, al principio, una a, luego, cualquier número de repeticiones de a, b y c combinados de cualquier manera y, para terminar, una b.

9. El lenguaje L_9 está formado por todas las palabras que no contienen ninguna c y en las que todas las apariciones de a (si hay alguna) están en el lado izquierdo y todas las apariciones de b (si hay alguna) aparecen en el lado derecho. Por ejemplo, las siguientes palabras pertenecen a L_9: ε, $aaaa\varepsilon$, $bbb\varepsilon$, $aabbb\varepsilon$. Una manera de definir formalmente este lenguaje es la siguiente:

$$L_9 = \{w | w \in \mathbb{A}^* \wedge \exists u, v(u \in \mathbb{A}^* \wedge v \in \mathbb{A}^* \wedge |u| = |u|_a \wedge |v| = |v|_b \wedge w = uv)\}$$

Otra manera de definir formalmente este lenguaje es la siguiente:

$$L_9 = \{w | w \in \mathbb{A}^* \wedge \exists j, k(j \in \mathbb{N} \wedge k \in \mathbb{N} \wedge w = (a\varepsilon)^j \circ (b\varepsilon)^k)\}$$

El lenguaje L_9 es regular porque la siguiente expresión regular expresa la estructura de todas las palabras de L_9:

$$a^*b^*$$

Mediante esa expresión regular se indica que las palabras de L_9 tienen, al principio, cualquier número de repeticiones de a —cero o más— y, para terminar, cualquier número de repeticiones de b —cero o más.

10. Sea el lenguaje L_{10} formado por todas las palabras que no contienen ninguna c y en las que las apariciones de a y las apariciones de b no están mezcladas (ε, $aaaa\varepsilon$, $bbb\varepsilon$, $aabbb\varepsilon$, $bbbaaaa\varepsilon$, etc.). Una manera de definir formalmente este lenguaje es la siguiente:

$$L_{10} = \{w | w \in \mathbb{A}^* \wedge \exists u, v (u \in \mathbb{A}^* \wedge v \in \mathbb{A}^* \wedge |u| = |u|_a \wedge |v| = |v|_b \wedge (w = uv \vee w = vu))\}$$

Otra manera de definir formalmente este lenguaje es la siguiente:

$$L_{10} = \{w | w \in \mathbb{A}^* \wedge \exists j, k (j \in \mathbb{N} \wedge k \in \mathbb{N} \wedge (w = (a\varepsilon)^j \circ (b\varepsilon)^k \vee w = (b\varepsilon)^k \circ (a\varepsilon)^j))\}$$

La siguiente expresión regular expresa la estructura de todas las palabras de L_{10}:

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

Mediante esa expresión regular se indica que las palabras de L_{10} tienen, al principio, cualquier número de repeticiones de a —cero o más— y, para terminar, cualquier número de repeticiones de b —cero o más— o, alternativamente, tienen, al principio, cualquier número de repeticiones de b y, para terminar, cualquier número de repeticiones de a.

11. Consideremos el lenguaje L_{11} formado por todas las palabras que contienen exactamente dos apariciones de a y cualquier cantidad de apariciones de b y c. Por ejemplo, las siguientes palabras pertenecen a L_{10}: ε, $aa\varepsilon$, $abacc\varepsilon$, $bbbaabbb\varepsilon$, $cabbbabb\varepsilon$ y $babbbca\varepsilon$. Una manera de definir formalmente este lenguaje es la siguiente:

$$L_{11} = \{w | w \in A^* \wedge |w|_a = 2\}$$

La siguiente expresión regular expresa la estructura de todas las palabras de L_{11}:

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

Mediante esa expresión regular se indica que las palabras de L_{11} tienen, al principio, cualquier número de repeticiones de b y c combibados de cualquier manera, luego una a, a continuación, cualquier número de repeticiones de b y c combibados de cualquier manera, de nuevo una a y, para terminar, cualquier número de repeticiones de b y c combibados de cualquier manera.

12. Sea el lenguaje L_{12} formado por todas las palabras en las que el número de apariciones de a es par. Por ejemplo, las siguientes palabras pertenecen a L_{12}: ε, $aa\varepsilon$, $abacc\varepsilon$, $aaaa\varepsilon$, $ababaabbbcaa\varepsilon$. Una manera de definir formalmente este lenguaje es la siguiente:

$$L_{12} = \{w | w \in A^* \wedge |w|_a \; mod \; 2 = 0\}$$

Para probar que es un lenguaje regular tenemos que dar una expresión regular que represente la estructura de todas las palabras de L_{12}. Una opción es la siguiente:

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

La parte $((b+c)^*a(b+c)^*a(b+c)^*)^*$ representa a ε y todas las palabras que tienen un número par de apariciones de a mayor o igual que 2. Pero esa expresión no incluye las palabras que no teniendo ninguna a, sí tienen al menos una b o una c. Es decir, palabras como $ccc\varepsilon$, $cbbcb\varepsilon$ y $b\varepsilon$. Puesto que esas palabras también pertenecen al lenguaje L_{12}, hace falta poner la parte $(b+c)^*$. Nótese que la expresión $(b+c)^*$ vuelve a incluir a la palabra vacía ε pero eso no es ningún problema porque en los conjuntos las repeticiones desaparecen y de cada elemento solo se mantiene una copia.

13. Sea el lenguaje L_{13} definido de la siguiente manera:

$$L_{13} = \{w \mid w \in \mathbb{A}^* \wedge \exists \alpha, \beta, u(\alpha \in \mathbb{A} \wedge \beta \in \mathbb{A} \wedge u \in \mathbb{A}^* \wedge |w|_\alpha = 2 \wedge w = \alpha\beta\alpha u)\}$$

Otra posible definición:

$$L_{13} = \{w \mid w \in \mathbb{A}^* \wedge |w| \geq 3 \wedge w(1) = w(3) \wedge |w|_{w(1)} = 2\}$$

En L_{13} se tienen las palabras en las que el símbolo de la primera posición vuelve a aparecer por segunda y última vez en la tercera posición. Esto conlleva que la longitud de las palabras es al menos 3. Por ejemplo, las siguientes palabras pertenecen al lenguaje: $ababcbbc\varepsilon$, $babaaaa\varepsilon$, $bcb\varepsilon$ y $bcbaaaac\varepsilon$. En cambio, estas otras no: ε, $a\varepsilon$, $bb\varepsilon$, $abaabbcc\varepsilon$, $ccc\varepsilon$, $abaa\varepsilon$ y $abcbba\varepsilon$.

La siguiente expresión regular representa la estructura de las palabras de L_{13}:

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

14. Sea el lenguaje L_{14} definido de la siguiente manera:

$$\begin{aligned} L_{14} = \{w \mid\; & w \in \mathbb{A}^* \wedge |w| \geq 3 \wedge |w| \bmod 3 = 0 \\ & \forall k((k \in I\!N \wedge 1 \leq k \leq |w| \wedge k \bmod 3 = 0) \rightarrow \\ & \quad ((w(k) = b \vee w(k) = c) \wedge w(k-1) = a \wedge \\ & \quad (w(k-2) = b \vee w(k-2) = c)))\} \end{aligned}$$

En L_{14} se tienen las palabras cuya longitud es múltiplo de 3 —es decir, 0, 3, 6, 9, 12 etc.— y que, además, en cada posición que es múltiplo de 3, se tiene b o c, en la anterior a y dos posiciones antes b o c. Por ejemplo, las siguientes palabras pertenecen al lenguaje: $\underline{cac}bab\varepsilon$, $\underline{cab}bab\underline{bab}\varepsilon$, $bac\varepsilon$, $\underline{cab}cac\varepsilon$ y $\underline{bac}cac\underline{bac}\varepsilon$. En cambio, estas otras no: ε, $a\varepsilon$, $bb\varepsilon$,

$aab\varepsilon$, $cccc\varepsilon$, $aabbaaa\varepsilon$, $abbacccaa\varepsilon$, $abc\varepsilon$ y $abbcacccbaaba\varepsilon$. En los ejemplos de palabras del lenguaje, se ha utilizado el subrayado para diferenciar bloques de tres elementos y mejorar la legibilidad.

La siguiente expresión regular representa la estructura de las palabras de L_{14}:

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

15. Sea el lenguaje L_{15} definido de la siguiente manera:

$$\begin{aligned} L_{15} = \{w \mid\ & w \in \mathbb{A}^* \wedge |w|_a \geq 1 \wedge |w|_b \geq 1 \wedge |w|_c \geq 1 \wedge \\ & \exists u, v, x (u \in \mathbb{A}^* \wedge v \in \mathbb{A}^* \wedge x \in \mathbb{A}^* \wedge |u|_a = 0 \wedge |v|_a = |v| \wedge \\ & |x|_a = 0 \wedge w = uvx)\} \end{aligned}$$

En L_{15} se tienen las palabras en las que todas las apariciones de a están en posiciones contiguas, pero además, tanto a como b como c aparecen al menos una vez. Por ejemplo, las siguientes palabras pertenecen al lenguaje: $cbbaaccbc\varepsilon$, $cab\varepsilon$, $bac\varepsilon$, $caab\varepsilon$, $aaabcbb\varepsilon$, $bcbaaaa\varepsilon$ y $bbaaaacbc\varepsilon$. En cambio, estas otras no: ε, $a\varepsilon$, $bb\varepsilon$, $aab\varepsilon$, $cccc\varepsilon$, $aabbaaa\varepsilon$, $abc\varepsilon$ y $accbccacbca\varepsilon$.

La siguiente expresión regular representa la estructura de las palabras de L_{15}:

$$\begin{aligned} & (b+c)^*bc(b+c)^*aa^*(b+c)^*+ \\ & (b+c)^*cb(b+c)^*aa^*(b+c)^*+ \\ & (b+c)^*b(b+c)^*aa^*(b+c)^*c(b+c)^*+ \\ & (b+c)^*c(b+c)^*aa^*(b+c)^*b(b+c)^*+ \\ & (b+c)^*aa^*(b+c)^*bc(b+c)^*+ \\ & (b+c)^*aa^*(b+c)^*cb(b+c)^* \end{aligned}$$

En esa expresión regular, aa^* representa un bloque formado por una o más repeticiones de a. Recordemos que es equivalente a $a(a^*)$.

16. Sea el lenguaje L_{16} definido de la siguiente manera:

$$L_{16} = \{w \mid w \in \mathbb{A}^* \wedge |w| \geq 4 \wedge (w(1) = b \vee w(1) = c) \wedge |w|_a \geq 3 \wedge w(|w|) = a\}$$

En L_{16} se tienen las palabras cuya longitud es al menos 4 y empiezan con b o c, tienen al menos tres apariciones de a y terminan con a. Por ejemplo, las siguientes palabras pertenecen al lenguaje: $baaa\varepsilon$, $ccabaabba\varepsilon$, $bababa\varepsilon$ y $cbbbabaabaaaa\varepsilon$. En cambio, estas otras no: ε, $a\varepsilon$, $bb\varepsilon$, $aab\varepsilon$, $cccc\varepsilon$, $aabbaaa\varepsilon$, $abbacccaa\varepsilon$, $abc\varepsilon$ y $caaaab\varepsilon$.

La siguiente expresión regular representa la estructura de las palabras de L_{16}:

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

8.2.3 Método para calcular la expresión regular correspondiente a un AF

En el apartado 8.2.1 de la página 13, se muestra a través de ejemplos, la relación directa entre la estructura de un AF y la estructura de las palabras aceptadas por el AF. A partir de observar esa relación, se ha definido la noción de expresión regular —apartado 8.2.2 de la página 17— que, de manera precisa formaliza la estructura de todas las palabras de un lenguaje aceptado por un AF.

En este apartado se presenta un método que, dado un AF, de manera sistemática permite obtener una expresión regular que expresa la estructura de todas las palabras aceptadas por el AF en cuestión. Este método va eliminando estados del AF y generando una expresión regular. Cuando se hayan eliminado todos los estados del AF original, se habrá construido una ER que expresa la estructura de todas las palabras aceptadas por el AF original.

8.2.3.1 Pautas para eliminar estados de un AF y construir una ER

Dado un AF, el primer paso es añadir un estado q_{ini} y un estado q_{fin}:

- q_{ini} será el nuevo estado inial del AF y habrá que poner una transición con etiqueta ε desde q_{ini} al estado inicial del AF original.
- q_{fin} será el único estado que tenga doble círculo y habrá que poner una transición con etiqueta ε desde cada estado que tenía doble círculo en el AF original al estado q_{fin}.

El segundo paso es sustituir todas las apariciones de λ por ε. De esta forma todas las etiquetas serán expresiones regulares. Recordemos que λ no es una expresión regular pero ε sí.

El tercer paso es sustituir las comas por $+$, tal como se muestra en la figura 8.2.1 de la página 26.

Una vez que se han añadido los estados q_{ini} y q_{fin}, se han sustituido las apariciones de λ por ε y se han sustituido las comas por $+$, hay que ir eliminando los estados del nuevo AF de uno en uno. Al final solo han de quedar los estados q_{ini} y q_{fin} con una transición que va desde q_{ini} a q_{fin} y que estará etiquetada con una ER que represente la estructura de todas las palabras aceptadas por el AF original.

Informalmente, la idea es que cuando se elimina un estado, la información de todos los caminos que pasaban por ese estado ha de mantenerse. Dicho de otra forma, los caminos y la información asociada a cada camino han de mantenerse. Para ello, esa información se representa mediante expresiones regulares.

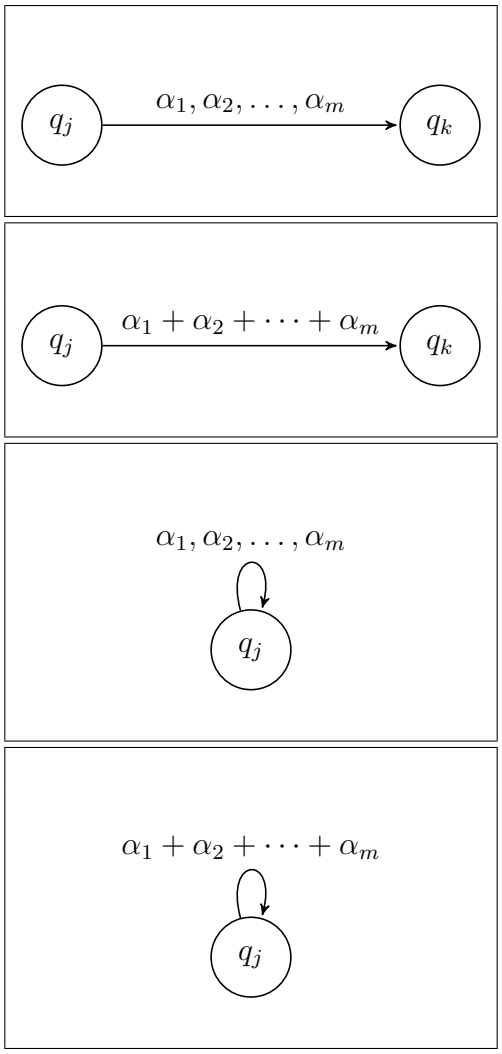


Figura 8.2.1. Sustitución de comas por $+$.

En la figura 8.2.2 de la página 28 se muestra una situación general que incluye tres estados q_j, q_k y q_ℓ. En esa situación, se indica la ER que surgiría al eliminar q_k.

En la figura 8.2.3 de la página 28 se muestra una situación que incluye dos estados q_j y q_k. Las transiciones van de q_j a q_k y de q_k a q_j. Es por tanto, como el caso general pero considerando que q_j y q_ℓ son el mismo estado.

En la figura 8.2.4 de la página 62 se muestra una situación general que incluye tres estados q_j, q_k y q_ℓ. En este caso, q_k no tiene bucle y ello es equivalente a tener un bucle etiquetado con $\varnothing$.

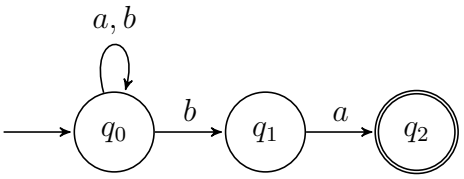
En la figura 8.2.5 de la página 63 se muestra una situación que incluye dos estados q_j y q_k. Las transiciones van de q_j a q_k y de q_k a q_j y q_k no tienen bucle. Es por tanto, como el caso general sin bucle pero considerando que q_j y q_ℓ son el mismo estado.

En la figura 8.2.6 de la página 64 se muestra una situación que incluye dos estados q_j y q_k. Las transiciones van de q_j a q_k y de q_k a q_k pero de q_k no hay acceso a ningún otro estado. La ausencia de transición representa, en general, una transición con $\varnothing$ como etiqueta. En este caso se puede interpretar como que existe una transición desde q_k a q_j con etiqueta $\varnothing$. Pero al concatenar $\varnothing$ con cualquier otra expresión, todo se vuelve $\varnothing$. En este caso hay que eliminar q_k sin más.

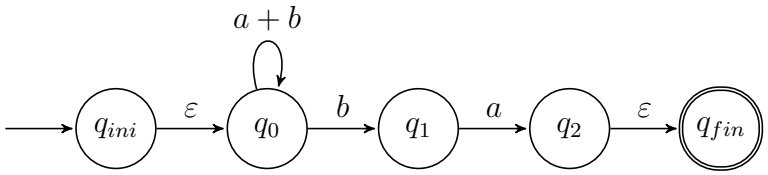
8.2.4 Ejemplos de cálculo de la ER correspondiente a un AF

8.2.4.1 Ejemplo 1: de AF a ER

Consideramos el siguiente AF:



Primero añadimos q_{ini} y q_{fin}, sustituimos λ por ε (en este ejemplo λ no aparece) y sustituimos comas por $+$:



A continuación se han de eliminar de uno en uno los estados que provienen del AF original. Por tanto, no se pueden eliminar q_{ini} y q_{fin}. Cualquier orden es bueno para ir eliminando los

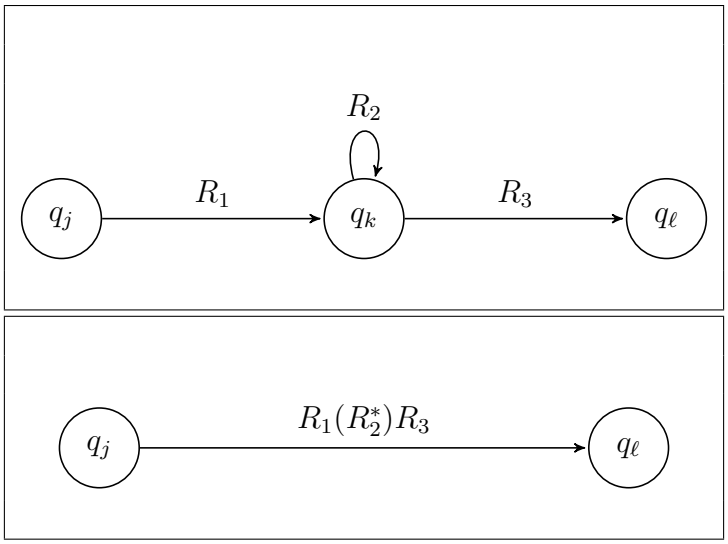


Figura 8.2.2. ER correspondiente a la eliminación del estado q_k: caso general.

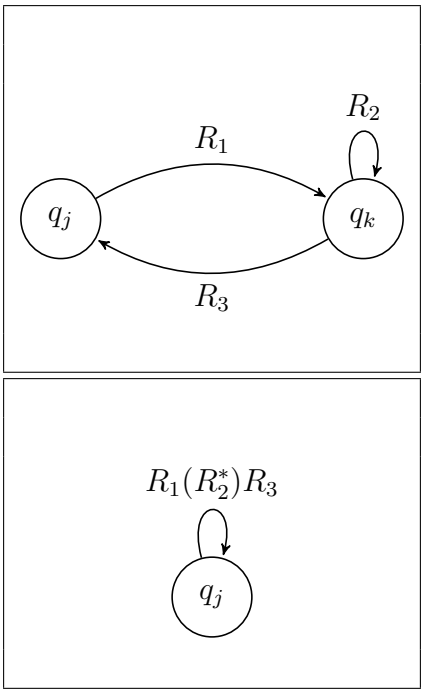
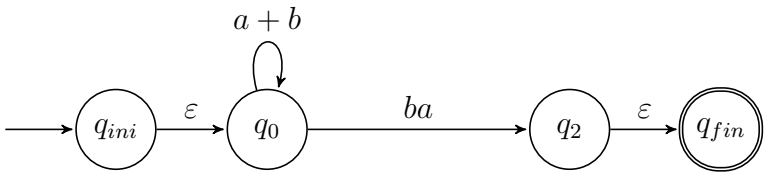


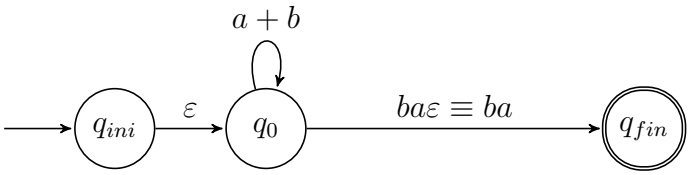
Figura 8.2.3. ER correspondiente a la eliminación del estado q_k: siendo q_j punto de partida y llegada.

estados pero lo más conveniente es ir quitando aquellos estados más fáciles de eliminar, es decir, aquellos estados por los que pasan menos caminos y los caminos son sencillos. Para empezar vamos a eliminar q_1:



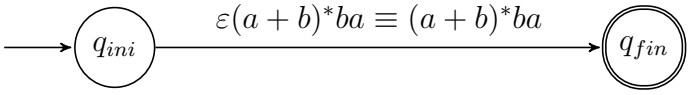
Ha desaparecido q_1 pero se mantiene el camino que pasaba por q_1. La nueva transición recoge —mediante la ER ba— la información de ese camino.

Ahora se va a eliminar q_2 pero hay que conservar el camino que pasa por q_2:



Cuando en una ER aparece ε concatenada, se puede simplificar la ER eliminando ε. Al eliminar q_2 ha surgido la ER $ba\varepsilon$. Puesto que ε aparece concatenada, se puede eliminar.

Para terminar, se va a eliminar q_0:

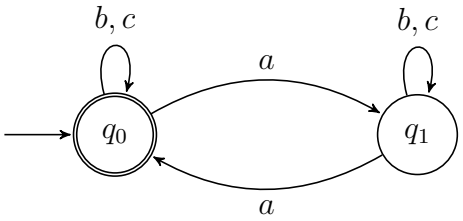


Al eliminar q_0 ha surgido la ER $\varepsilon(a+b)^*ba$. Puesto que ε aparece concatenada, se puede eliminar y queda $(a+b)^*ba$.

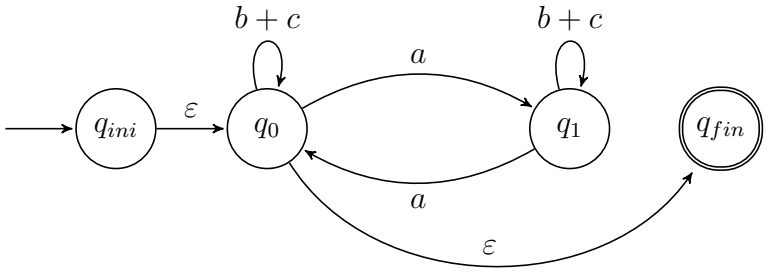
Puesto que ya se han eliminado todos los estados del AF original, la ER que representa la estructura de todas las palbras aceptadas por dicho AF es $(a+b)^*ba$. Por tanto, las palabras de este lenguaje tendrán un bloque inicial formado por cualquier número de apariciones de a y b mezcladas de cualquier forma y, para terminar, se ha de tener una aparición de b y una de a en ese orden. Dicho de otra forma, tenemos el lenguaje de las palabras definidas sobre el alfabeto $\{a, b\}$ que terminan con la secuencia ba.

8.2.4.2 Ejemplo 2: de AF a ER

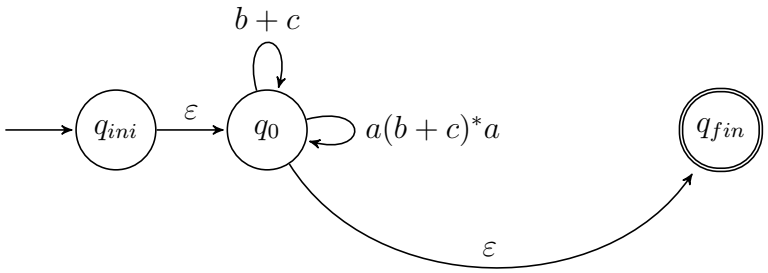
Consideramos el siguiente AF:



Primero añadimos q_{ini} y q_{fin}, sustituimos λ por ε (en este ejemplo λ tampoco aparece) y sustituimos comas por $+$:

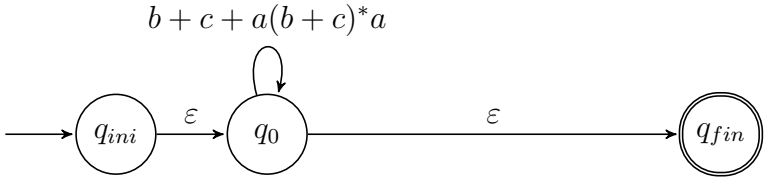


A continuación se han de eliminar de uno en uno los estados que provienen del AF original pero no q_{ini} y q_{fin}. Cualquier orden es bueno para ir eliminando los estados pero lo más conveniente es ir quitando aquellos estados más fáciles de eliminar, es decir, aquellos estados por los que pasan menos caminos y los caminos son sencillos. Para empezar vamos a eliminar q_1:

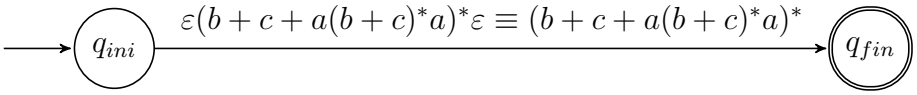


Ha desaparecido q_1 pero se mantiene el camino que pasaba por q_1. El nuevo bucle en q_0 recoge —mediante la ER $a(b+c)^*a$— la información de ese camino.

Puesto que en q_0 hay dos bucles, eso es lo mismo que tener un bucle con las expresiones separadas por comas, pero aquí en vez de comas hay que poner $+$. Por tanto, tenemos lo siguiente:



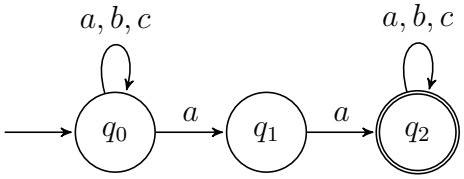
Ahora se eliminará q_0:



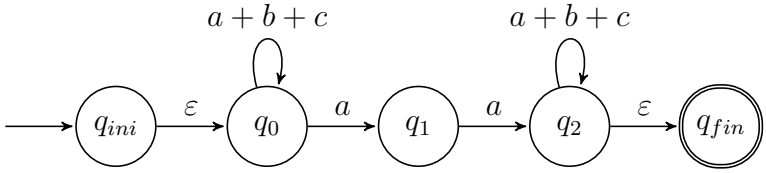
La ER resultante es $\varepsilon(b+c+a(b+c)^*a)^*\varepsilon$, pero cuando ε aparece concatenada, se puede eliminar y queda la ER $(b+c+a(b+c)^*a)^*$.

8.2.4.3 Ejemplo 3: de AF a ER

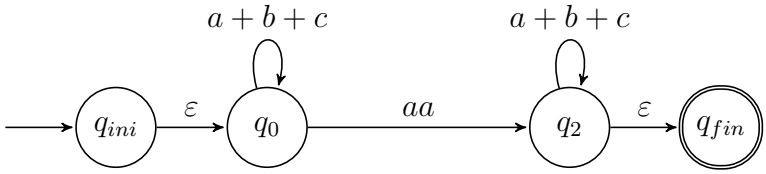
Consideramos el siguiente AF:



Primero añadimos q_{ini} y q_{fin}, sustituimos λ por ε (en este ejemplo λ no aparece) y sustituimos comas por $+$:

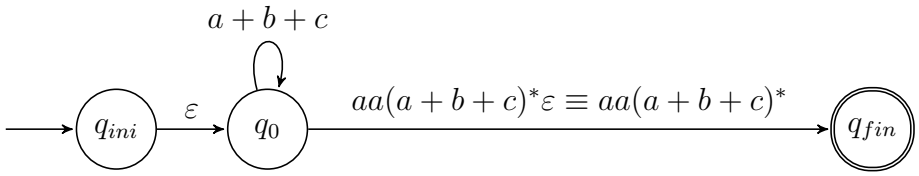


A continuación se han de eliminar de uno en uno los estados que provienen del AF original. Por tanto, no se pueden eliminar q_{ini} y q_{fin}. Cualquier orden es bueno para ir eliminando los estados pero lo más conveniente es ir quitando aquellos estados más fáciles de eliminar, es decir, aquellos estados por los que pasan menos caminos y los caminos son sencillos. Para empezar vamos a eliminar q_1:



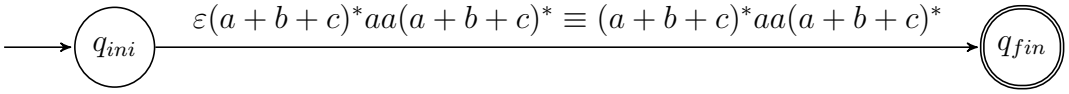
Ha desaparecido q_1 pero se mantiene el camino que pasaba por q_1. La nueva transición recoge —mediante la ER aa— la información de ese camino.

Ahora se va a eliminar q_2 pero hay que conservar el camino que pasa por q_2:



Cuando en una ER aparece ε concatenada, se puede simplificar la ER eliminando ε. Al eliminar q_2 ha surgido la ER $aa(a+b+c)^*\varepsilon$. Puesto que ε aparece concatenada, se puede eliminar y queda $aa(a+b+c)^*$.

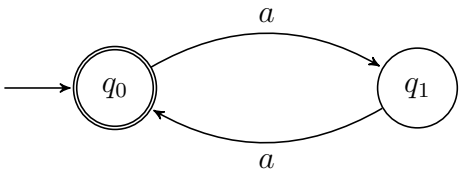
Para terminar, se va a eliminar q_0:



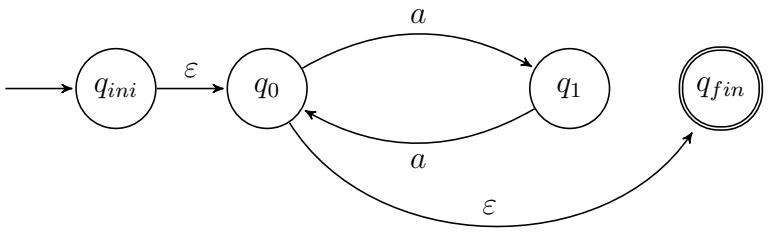
Al eliminar q_0 ha surgido la ER $\varepsilon(a+b+c)^*aa(a+b+c)^*$. Puesto que ε aparece concatenada, se puede eliminar y queda $(a+b+c)^*aa(a+b+c)^*$.

8.2.4.4 Ejemplo 4: de AF a ER

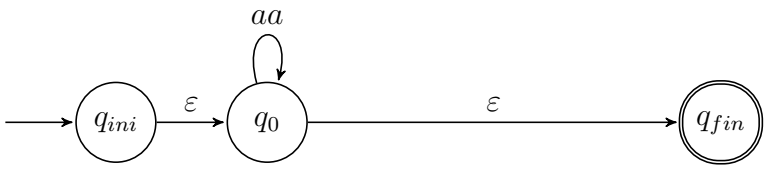
Consideramos el siguiente AF:



Primero añadimos q_{ini} y q_{fin}, sustituimos λ por ε (en este ejemplo λ tampoco aparece) y sustituimos comas por $+$ (no hay ninguna coma):

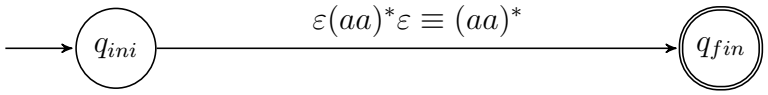


A continuación se han de eliminar de uno en uno los estados que provienen del AF original pero no q_{ini} y q_{fin}. Cualquier orden es bueno para ir eliminando los estados pero lo más conveniente es ir quitando aquellos estados más fáciles de eliminar, es decir, aquellos estados por los que pasan menos caminos y los caminos son sencillos. Para empezar vamos a eliminar q_1:



Ha desaparecido q_1 pero se mantiene el camino que pasaba por q_1. El nuevo bucle en q_0 recoge —mediante la ER aa— la información de ese camino.

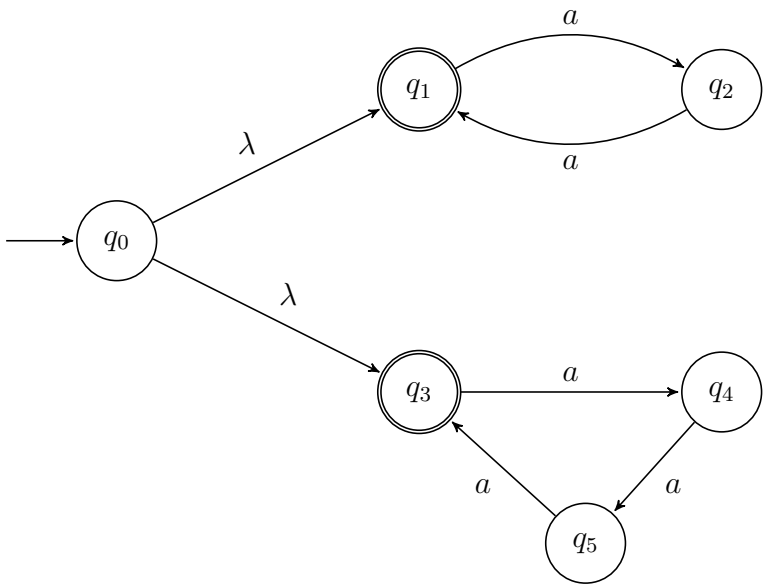
Ahora se eliminará q_0:



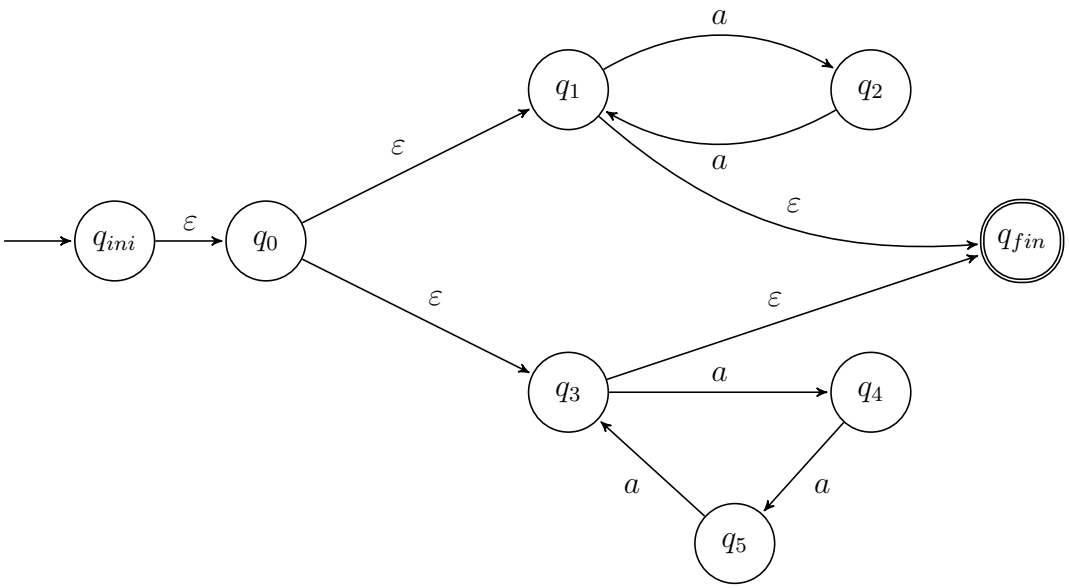
La ER resultante es $\varepsilon(aa)^*\varepsilon$, pero cuando ε aparece concatenada, se puede eliminar y queda la ER $(aa)^*$.

8.2.4.5 Ejemplo 5: de AF a ER

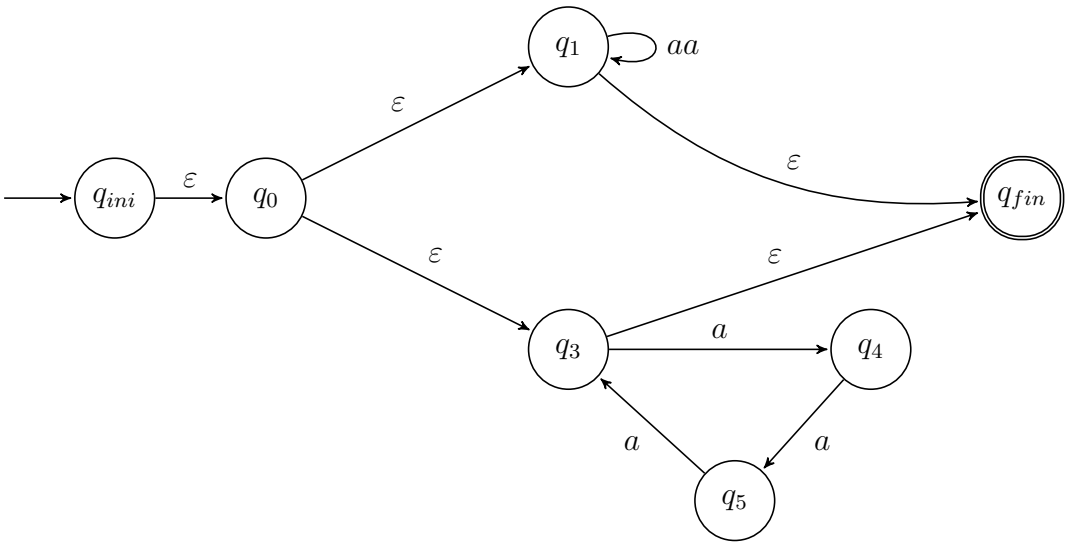
Consideramos el siguiente AF:



Primero añadimos q_{ini} y q_{fin}, sustituimos λ por ε y sustituimos comas por $+$ (no hay ninguna coma):

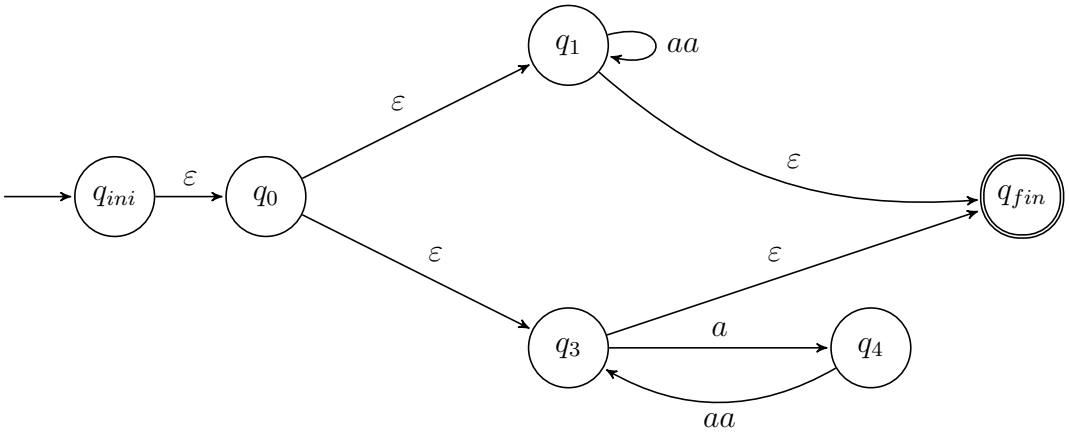


A continuación se han de eliminar de uno en uno los estados que provienen del AF original pero no q_{ini} y q_{fin}. Cualquier orden es bueno para ir eliminando los estados pero lo más conveniente es ir quitando aquellos estados más fáciles de eliminar, es decir, aquellos estados por los que pasan menos caminos y los caminos sean sencillos. Para empezar vamos a eliminar q_2:



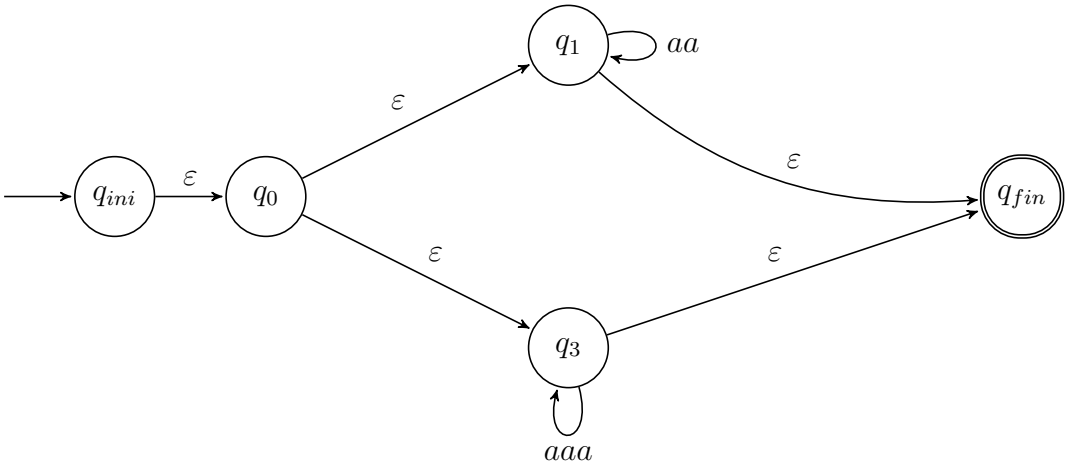
Ha desaparecido q_2 pero se mantiene el camino que pasaba por q_2. El nuevo bucle en q_1 recoge —mediante la ER aa— la información de ese camino.

Ahora se eliminará q_5:



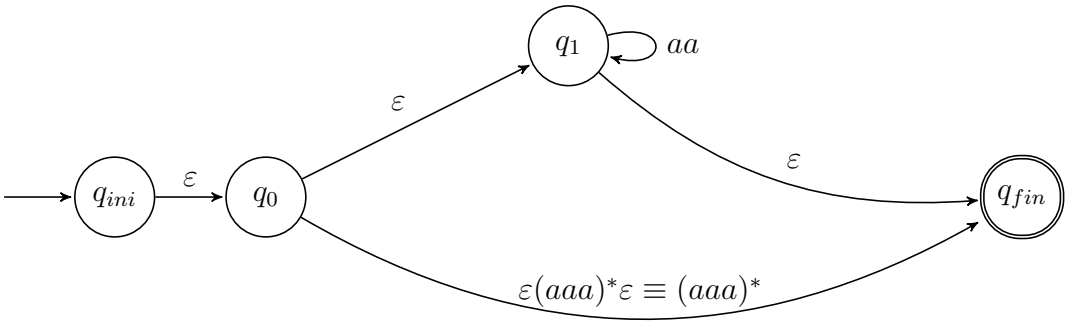
Ha desaparecido q_5 pero se mantiene el camino que pasaba por q_5. El arco que va de q_4 a q_3 recoge —mediante la ER aa— la información de ese camino.

Ahora se eliminará q_4:



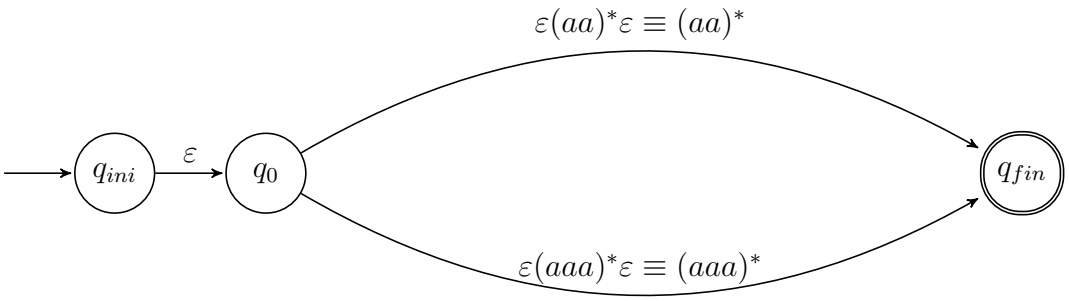
Ha desaparecido q_4 pero se mantiene el camino que pasaba por q_4. El bucle de q_3 recoge —mediante la ER aaa— la información de ese camino.

Ahora se eliminará q_3:



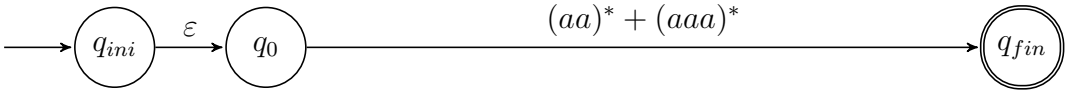
Ha desaparecido q_3 pero se mantiene el camino que pasaba por q_3. El arco que va de q_0 a q_{final} recoge —mediante la ER $\varepsilon(aaa)^*\varepsilon$— la información de ese camino. Puesto que cuando ε aparece en concatenación se puede eliminar, la expresión resultante es $(aaa)^*$.

Ahora se eliminará q_1:

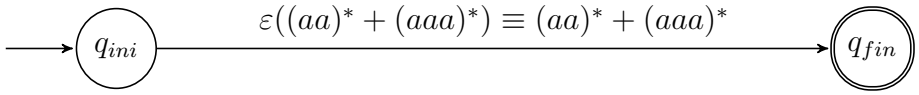


Ha desaparecido q_1 pero se mantiene el camino que pasaba por q_1. El arco superior que va de q_0 a q_{fin} recoge —mediante la ER $\varepsilon(aa)^*\varepsilon$— la información de ese camino. Puesto que cuando ε aparece en concatenación se puede eliminar, la expresión resultante es $(aa)^*$.

Los dos arcos que van de q_0 a q_{final} se juntan en un único arco con una única etiqueta que junta las etiquetas $(aa)^*$ y $(aaa)^*$ mediante $+$:



Ahora se eliminará q_0:

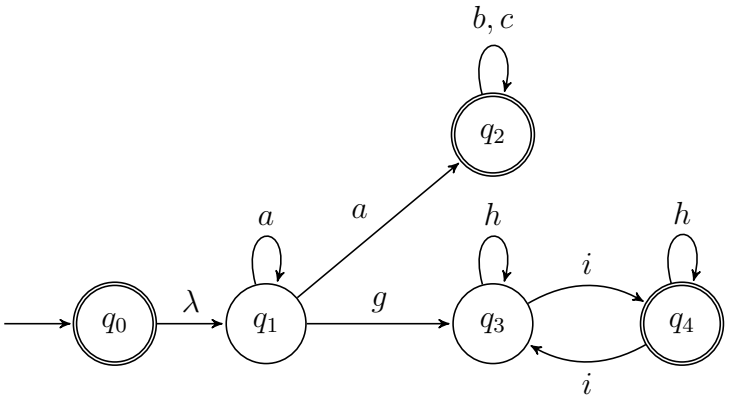


Ha desaparecido q_0 pero se mantiene el camino que pasaba por q_0. El arco que va de q_{ini} a q_{fin} recoge —mediante la ER $\varepsilon((aa)^* + (aaa)^*)$— la información de ese camino. Puesto que cuando ε aparece en concatenación se puede eliminar, la expresión resultante es $(aa)^* + (aaa)^*$.

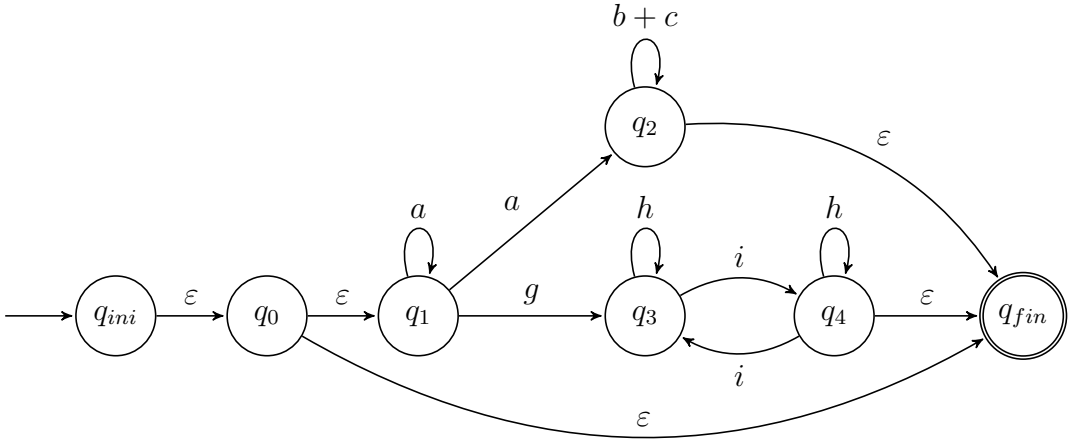
Se han eliminado todos los estados del AF original y, por tanto, la ER definitiva es $(aa)^* + (aaa)^*$.

8.2.4.6 Ejemplo 6: de AF a ER

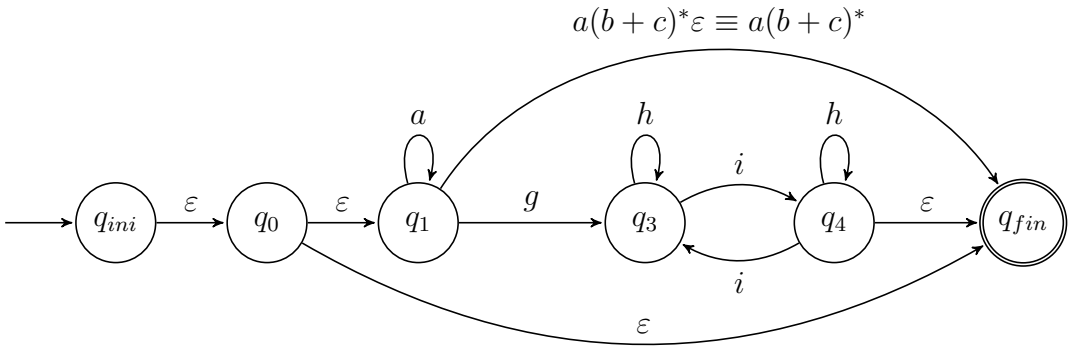
Consideramos el siguiente AF:



Primero añadimos q_{ini} y q_{fin}, sustituimos λ por ε y sustituimos comas por $+$:

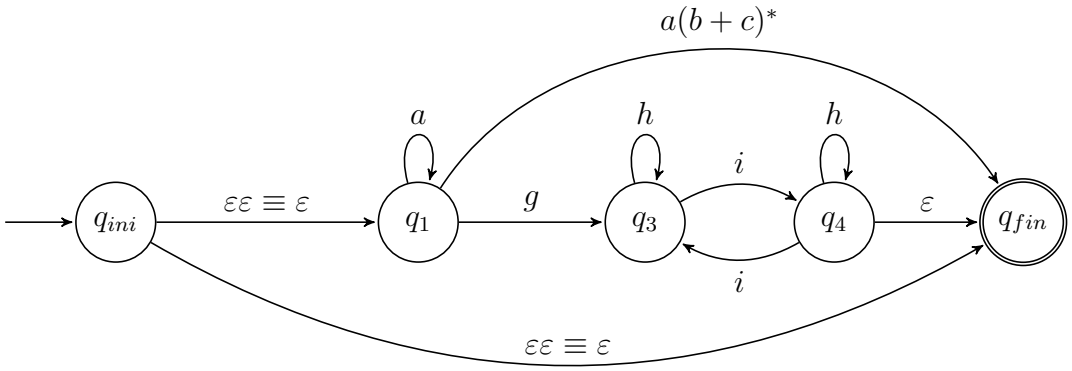


A continuación se han de eliminar de uno en uno los estados que provienen del AF original pero no q_{ini} y q_{fin}. Cualquier orden es bueno para ir eliminando los estados pero lo más conveniente es ir quitando aquellos estados más fáciles de eliminar, es decir, aquellos estados por los que pasan menos caminos y los caminos sean sencillos. Para empezar vamos a eliminar q_2:



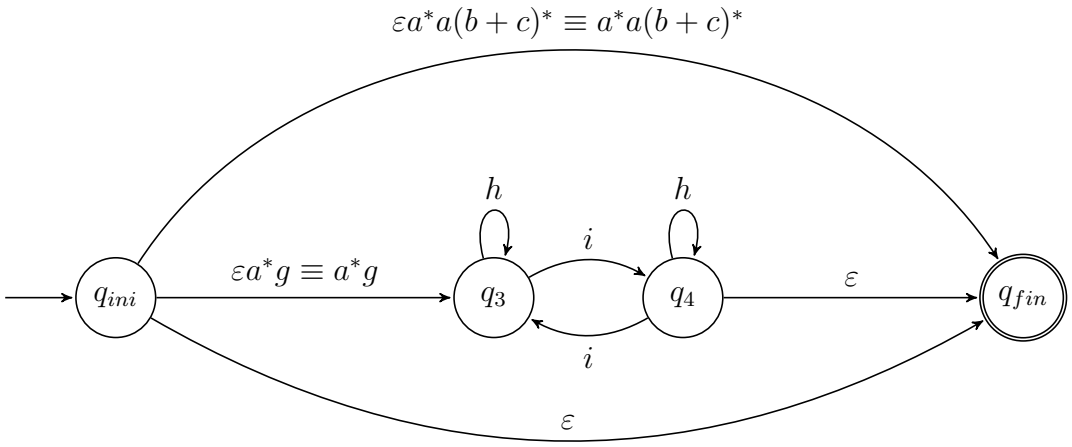
Ha desaparecido q_2 pero se mantiene el camino que pasaba por q_2. El arco que va de q_1 a q_{fin} recoge —mediante la ER $a(b+c)^*\varepsilon$— la información de ese camino. Puesto que cuando ε aparece en concatenación se puede eliminar, la expresión resultante es $a(b+c)^*$.

Ahora eliminamos q_0:



Ha desaparecido q_0 pero se mantienen los dos caminos que pasaban por q_0. El arco que va de q_{ini} a q_1 recoge —mediante la ER $\varepsilon\varepsilon$— la información de uno de los dos caminos y el arco que va de q_{ini} a q_{fin} recoge —mediante la ER $\varepsilon\varepsilon$— la información del otro camino. Puesto que cuando ε aparece repetida en concatenación se puede eliminar una de las copias, la expresión resultante es ε en ambos casos.

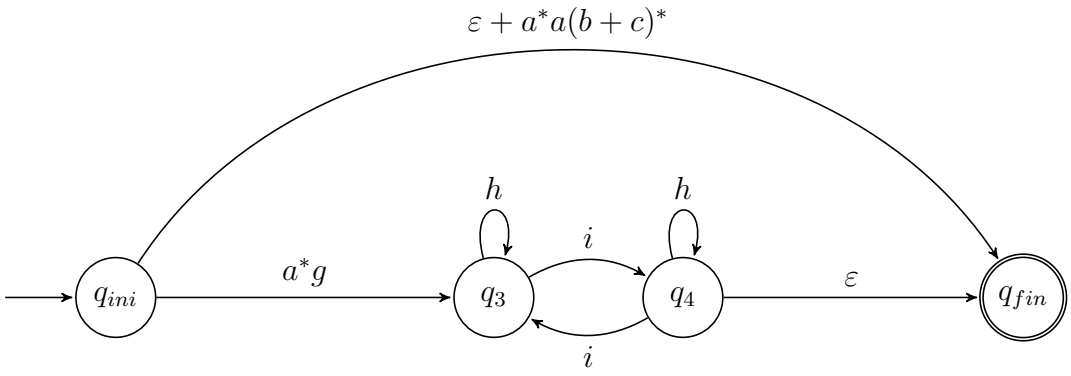
Ahora eliminamos q_1:



Ha desaparecido q_1 pero se mantienen los dos caminos que pasaban por q_1. El arco que va de q_{ini} a q_3 recoge —mediante la ER $\varepsilon a^* g$— la información de uno de los dos caminos y el arco que va de q_{ini} a q_{fin} recoge —mediante la ER $\varepsilon a^* a(b+c)^*$— la información del otro camino. Puesto que cuando ε aparece en concatenación se puede eliminar, la expresiones resultantes son

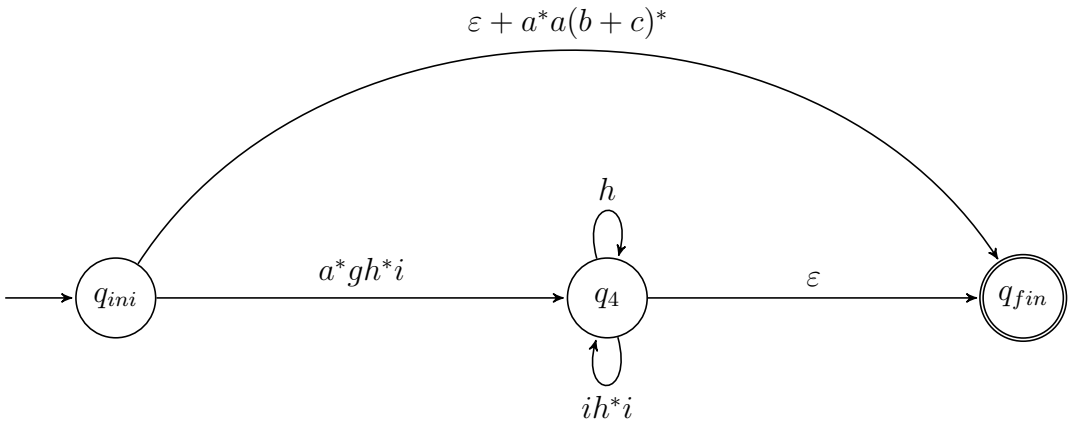
a^*g y $a^*a(b+c)^*$.

Además, ahora tenemos dos arcos que van de q_{ini} a q_{fin}. Vamos a poner un único arco juntando las dos ER con el operador $+$:



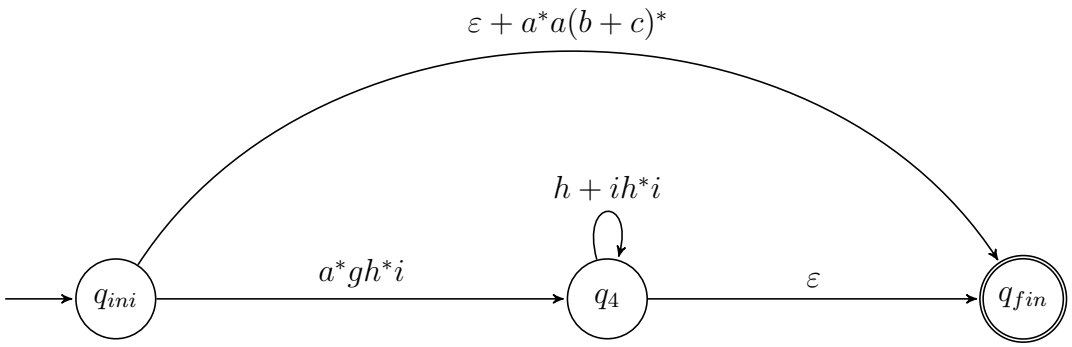
Nótese que en la ER $\varepsilon + a^*a(b+c)^*$, la expresión ε no aparece concatenada sino que unida mediante $+$. Debido a ello, no se puede eliminar.

Ahora eliminamos q_3:

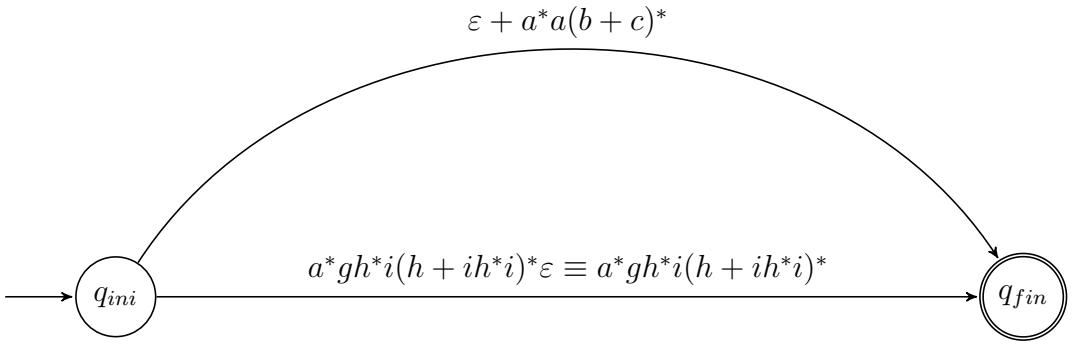


Ha desaparecido q_3 pero se mantienen los dos caminos que pasaban por q_3. El arco que va de q_{ini} a q_4 recoge —mediante la ER a^*gh^*i— la información de uno de los dos caminos y el nuevo bucle que va de q_4 a q_4 recoge —mediante la ER ih^*i— la información del otro camino.

Ahora tenemos dos bucles en q_4. Vamos a poner un único bucle juntando las dos ER con el operador $+$:

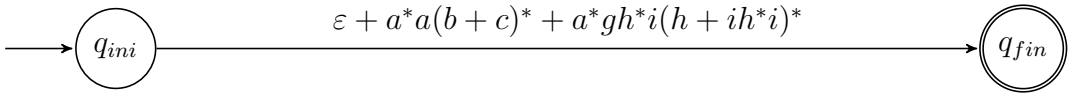


Finalmente, eliminamos q_4:



Ha desaparecido q_4 pero se mantiene el camino que pasaba por q_4. El arco que va de q_{ini} a q_{fin} recoge —mediante la ER $a^*gh^*i(h + ih^*i)^*\varepsilon$— la información de uno de ese camino. Puesto que ε aparece concatenada, se puede eliminar. Queda la siguiente ER: $a^*gh^*i(h+ih^*i)^*$.

Ahora tenemos dos arcos que van de q_{ini} a q_{fin}. Vamos a poner un único arco juntando las dos ER con el operador $+$:

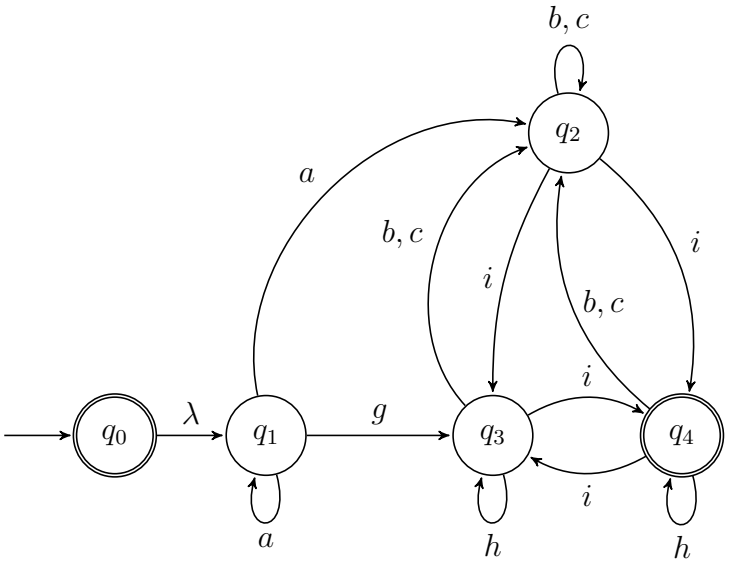


La ER resultante es una manera de representar la estructura de todas las palabras aceptadas por el AF de partida:

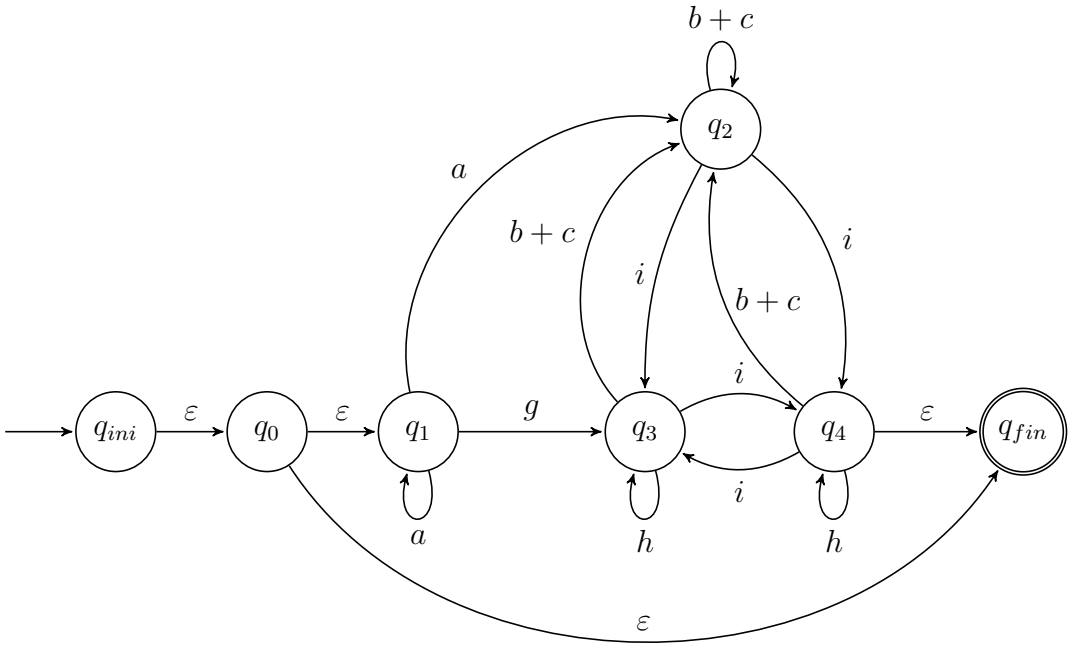
$$\varepsilon + (a^*a(b+c)^*) + (a^*gh^*i(h + ih^*i)^*)$$

8.2.4.7 Ejemplo 7: de AF a ER

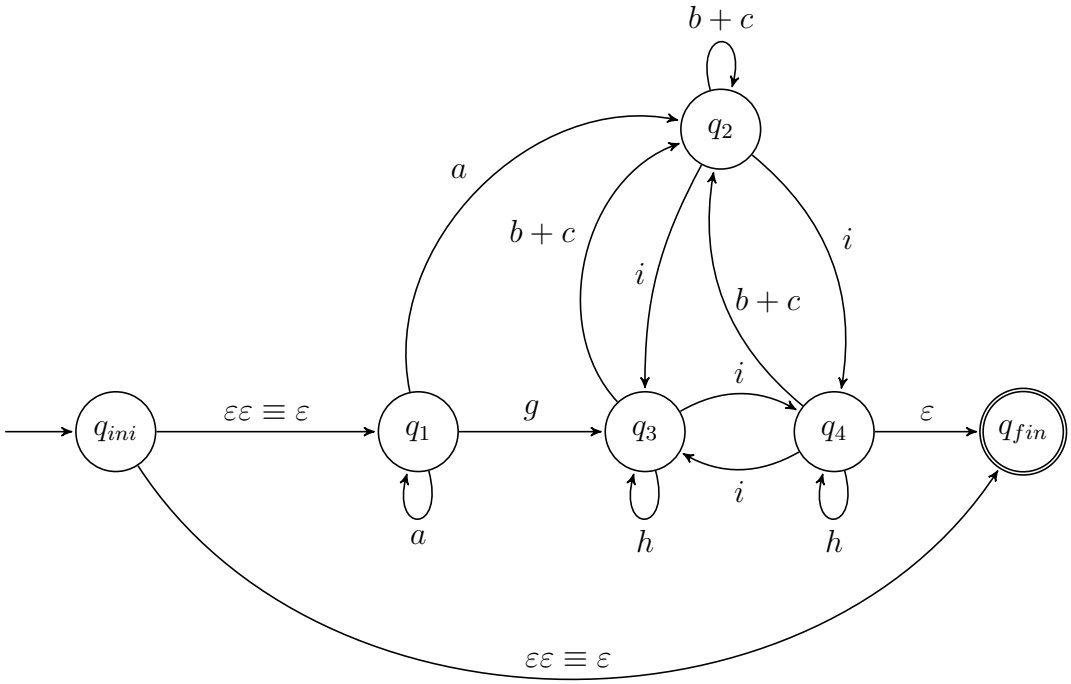
Consideramos el siguiente AF:



Primero añadimos q_{ini} y q_{fin}, sustituimos λ por ε y sustituimos comas por $+$:

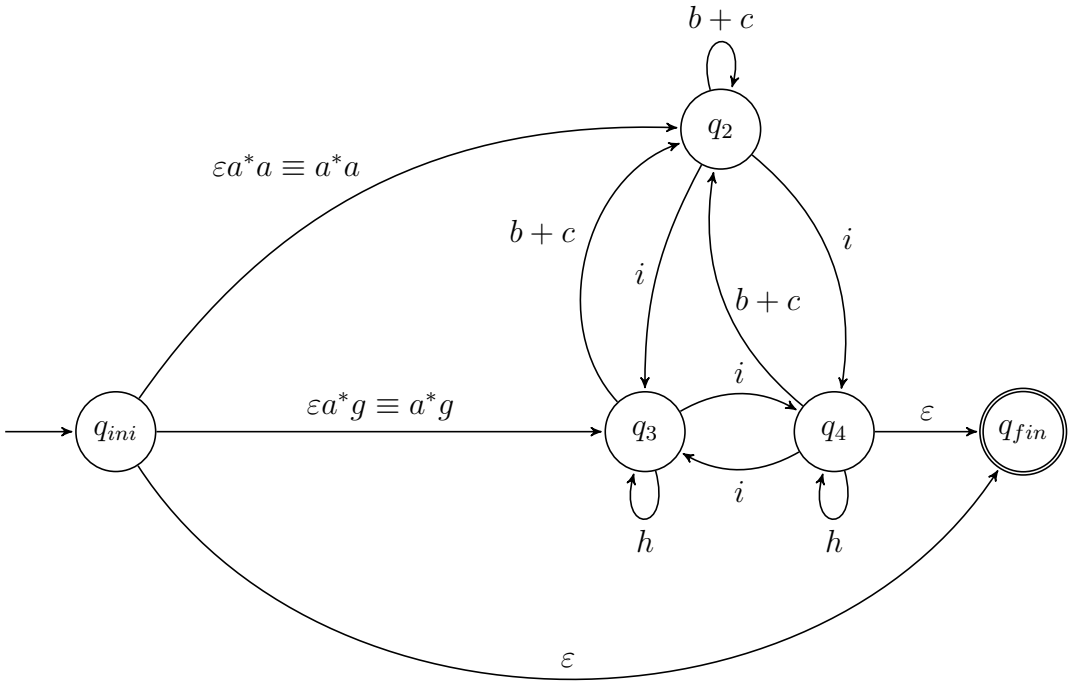


A continuación se han de eliminar de uno en uno los estados que provienen del AF original pero no q_{ini} y q_{fin}. Cualquier orden es bueno para ir eliminando los estados pero lo más conveniente es ir quitando aquellos estados más fáciles de eliminar, es decir, aquellos estados por los que pasan menos caminos y los caminos sean sencillos. Para empezar vamos a eliminar q_0:



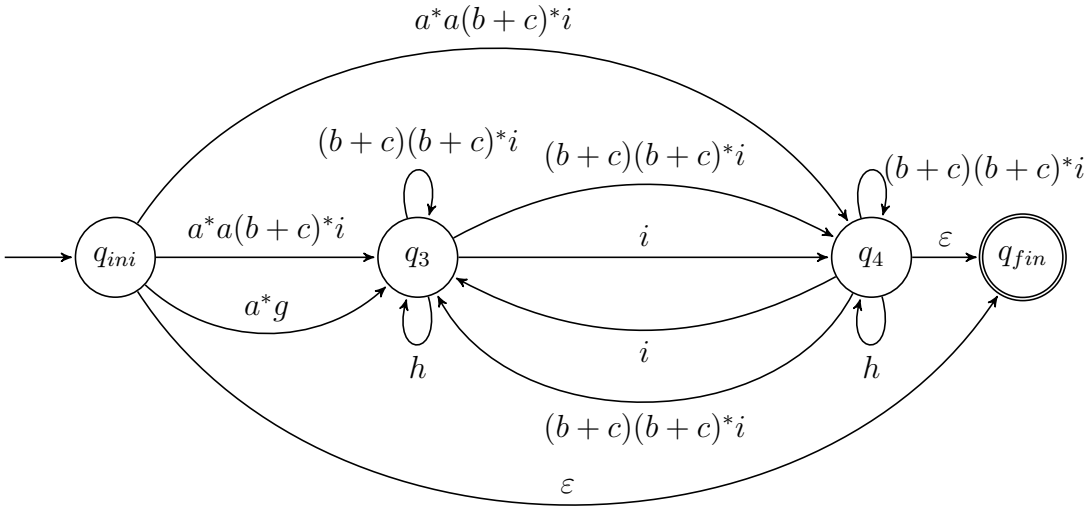
Ha desaparecido q_0 pero se mantienen los dos caminos que pasaban por q_0. El arco que va de q_{ini} a q_1 recoge —mediante la ER $\varepsilon\varepsilon$— la información de uno de los dos caminos y el arco que va de q_{ini} a q_{fin} recoge —mediante la ER $\varepsilon\varepsilon$— la información del otro camino. Puesto que cuando ε aparece repetida en concatenación se puede eliminar una de las copias, la expresión resultante es ε en ambos casos.

A continuación, se eliminará q_1:



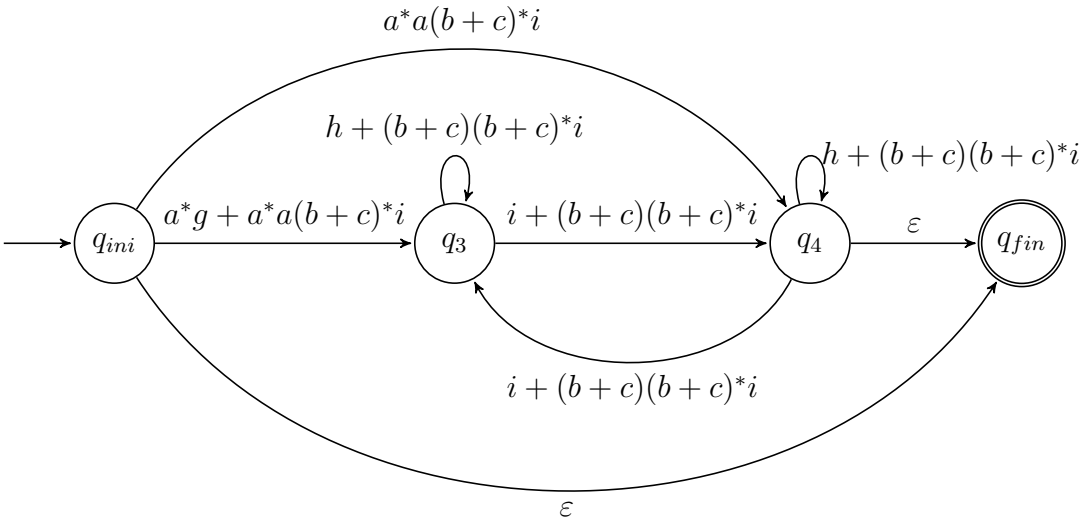
Ha desaparecido q_1 pero se mantienen los dos caminos que pasaban por q_1. El arco que va de q_{ini} a q_2 recoge —mediante la ER εa^*a— la información de uno de los dos caminos y el arco que va de q_{ini} a q_3 recoge —mediante la ER εa^*g— la información del otro camino. Puesto que cuando ε aparece en concatenación se puede eliminar, la expresiones resultantes son a^*a y a^*g.

A continuación, se eliminará q_2:

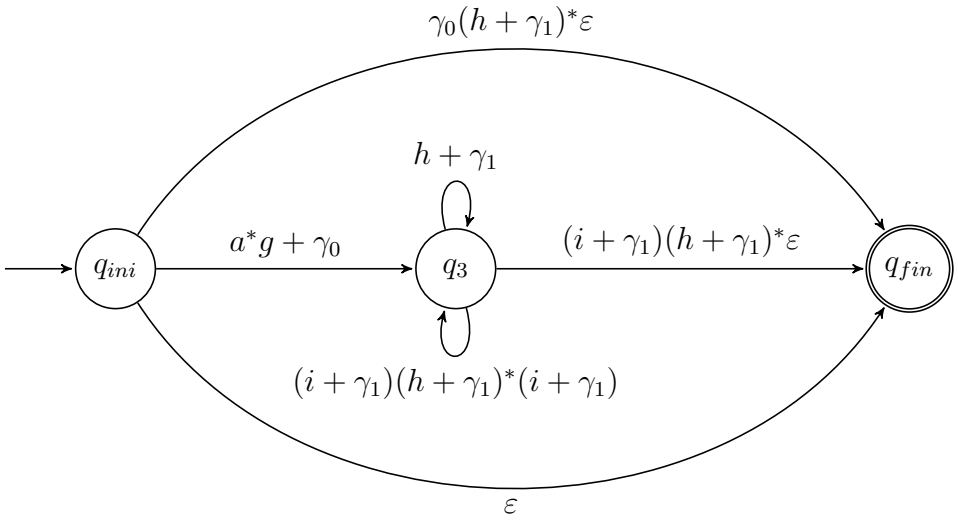


Ha desaparecido q_2 pero se mantienen los seis caminos que pasaban por q_2. El arco que va de q_{ini} a q_3 recoge —mediante la ER $a^*a(b+c)^*i$— la información de uno de los seis caminos; el arco que va de q_{ini} a q_4 recoge —mediante la ER $a^*a(b+c)^*i$— la información del segundo camino; el arco que va de q_3 a q_4 con ER $(b+c)(b+c)^*i$ recoge la información del tercer camino; el arco que va de q_4 a q_3 con ER $(b+c)(b+c)^*i$ recoge la información del cuarto camino. Los restantes dos caminos se corresponden con los bucles definidos sobre q_3 y q_4 con ER $(b+c)(b+c)^*i$.

A continuación se unifican los arcos y los bucles que tienen la misma trayectoria:



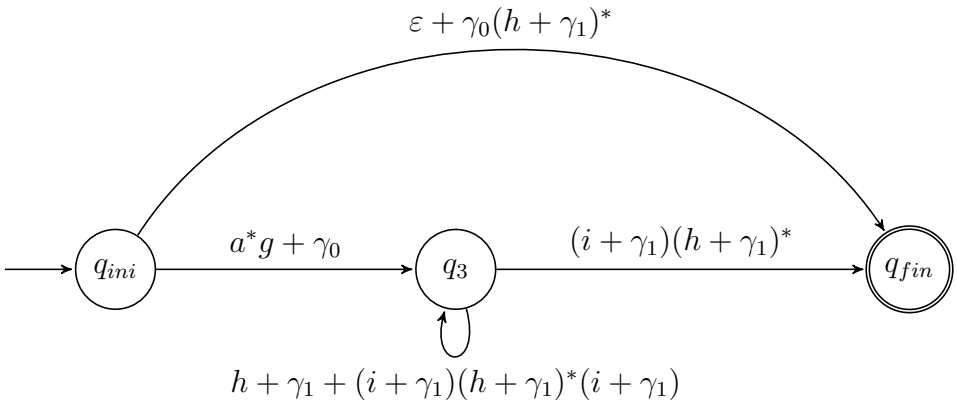
A continuación, se eliminará q_4:



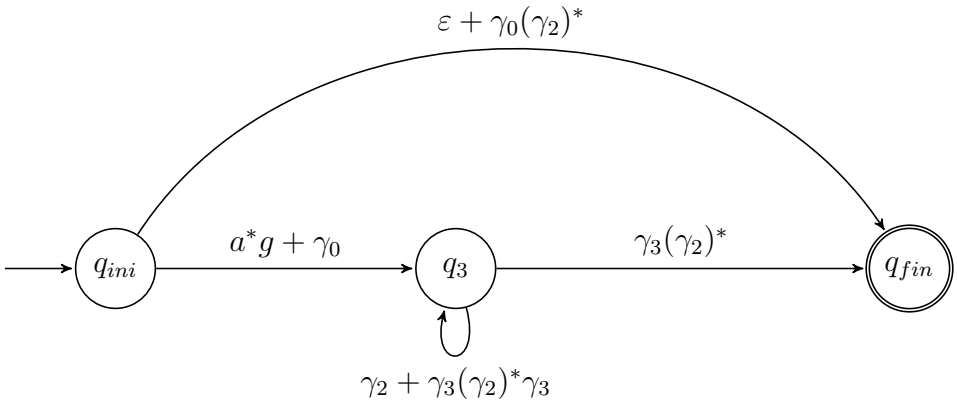
donde $\gamma_0 \equiv a^*a(b+c)^*i$ y $\gamma_1 \equiv (b+c)(b+c)^*i$.

Ha desaparecido q_4 pero se mantienen los tres caminos que pasaban por q_4. El arco que va de q_{ini} a q_{fin} recoge —mediante la ER $\gamma_0(h+\gamma_1)^*\varepsilon$— la información de uno de los tres caminos; el arco que va de q_3 a q_{fin} recoge —mediante la ER $(i+\gamma_1)(h+\gamma_1)^*\varepsilon$— la información del segundo camino; el bucle sobre q_3 con ER $(i+\gamma_1)(h+\gamma_1)^*(i+\gamma_1)$ recoge la información del tercer camino.

A continuación se unifican los arcos y los bucles que tienen la misma trayectoria y se eliminan las apariciones de ε cuando esta aparezca concatenada a otra ER:

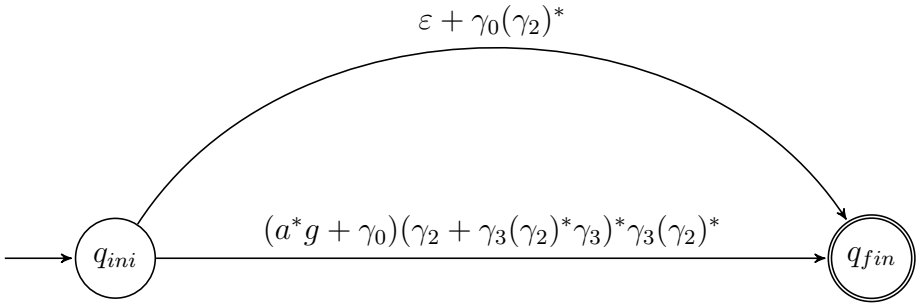


Para simplificar las etiquetas de los arcos, vamos a introducir más abreviaturas:



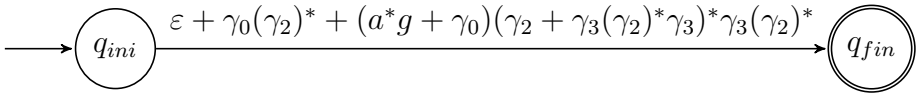
donde $\gamma_2 \equiv h + \gamma_1$ y $\gamma_3 \equiv i + \gamma_1$.

Finalmente, se elimina q_3:



Ha desaparecido q_3 pero se mantiene el camino que pasaba por q_3. El arco que va de q_{ini} a q_{fin} recoge —mediante la ER $(a^*g + \gamma_0)(\gamma_2 + \gamma_3(\gamma_2)^*\gamma_3)^*\gamma_3(\gamma_2)^*$— la información de ese camino.

A continuación se unifican los dos arcos que van de q_{ini} a q_{fin}:



Ahí se puede ver la ER que representa la estructura de todas las palabras aceptadas por el AF de partida. Las abreviaturas son las siguientes:

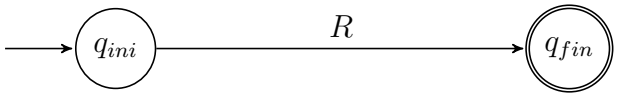
- $\gamma_0 \equiv a^*a(b + c)^*i$
- $\gamma_1 \equiv (b + c)(b + c)^*i$
- $\gamma_2 \equiv h + \gamma_1$
- $\gamma_3 \equiv i + \gamma_1$

8.2.5 Método para calcular el AF correspondiente a una expresión regular

En este apartado se presenta un método que, dada una ER, de manera sistemática permite obtener un AF que acepta las palabras que tienen la estructura formalizada mediante la ER en cuestión. Este método parte de un AF que tiene solo dos estados —el inicial q_{ini} y el estado final q_{fin}— y un arco entre esos dos estados, etiquetado con la ER de entrada. A partir de ahí, va generando estados del AF y descomponiendo la expresión regular. Cuando todos los arcos estén etiquetados por símbolos del alfabeto o por λ, se tendrá el AF que se quería obtener.

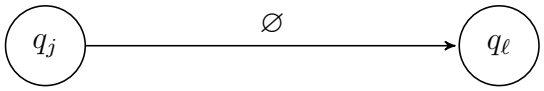
8.2.5.1 Pautas para generar estados de un AF a partir de una ER

Dada una ER R, el primer paso es crear un AF con un estado inicial q_{ini} y un estado final o aceptador q_{fin}. Además, habrá que poner una transición, con R como etiqueta, desde q_{ini} a q_{fin}.



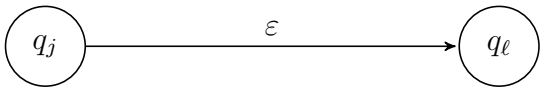
A continuación, habrá que ir aplicando los siguientes esquemas generales hasta que cada transición o arco esté etiquetado por un único símbolo del alfabeto $\mathbb{A}$ o por λ. En estos esquemas generales, se consideran dos estados q_j y q_ℓ y un arco entre ellos, etiquetado mediante una ER R:

1. Si tenemos que R es $\varnothing$

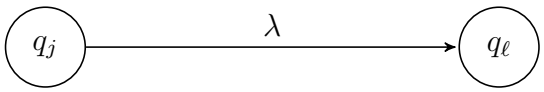


   entonces, se ha de eliminar el arco entre q_j y q_ℓ:

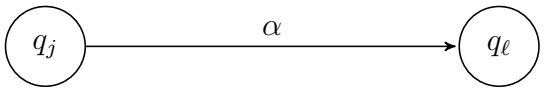
2. Si tenemos que R es ε



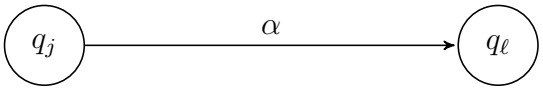
   entonces, se ha de sustituir ε por λ:



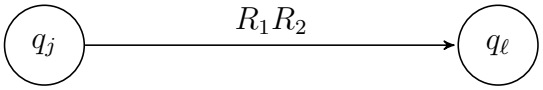
3. Si tenemos que R es un símbolo α del alfabeto $\mathbb{A}$



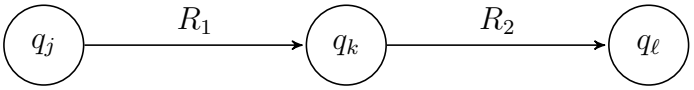
   entonces, se ha de mantener α:



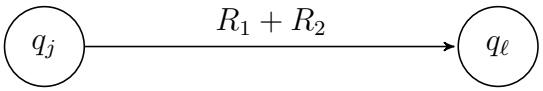
4. Si tenemos que R es la concatenación de dos expresiones regulares R_1 y R_2, es decir, $R = R_1R_2$



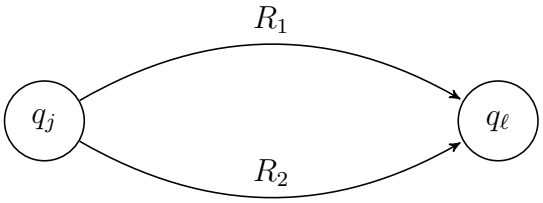
entonces, se ha de añadir un estado q_k entre q_j y q_ℓ y se ha de sustituir el arco que va de q_j a q_ℓ por dos arcos, uno de q_j a q_k con etiqueta R_1 y otro q_k a q_ℓ con etiqueta R_2:



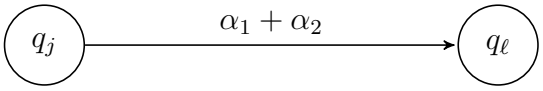
5. Si tenemos que R es la unión de dos expresiones regulares R_1 y R_2, es decir, $R = R_1+R_2$



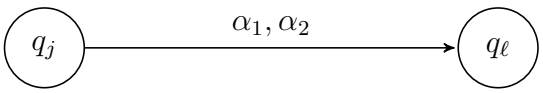
entonces, se ha de sustituir el arco que va de q_j a q_ℓ con etiqueta $R_1 + R_2$ por dos arcos que van de q_j a q_ℓ, uno con etiqueta R_1 y el otro con etiqueta R_2:



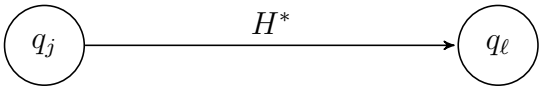
Recordemos que si las expresiones R_1 y R_2 fuesen dos símbolos α_1 y α_2 del alfabeto,



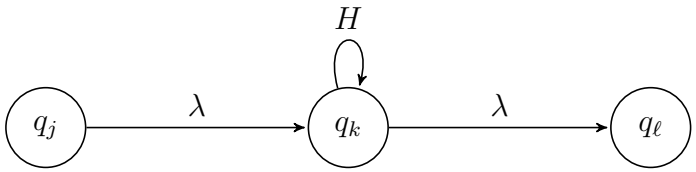
entonces se pondría un único arco y α_1 y α_2 aparecerían separados por una coma:



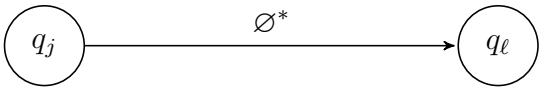
6. Si tenemos que R es la clausura universal de otra expresión regular H, es decir, $R = H^*$



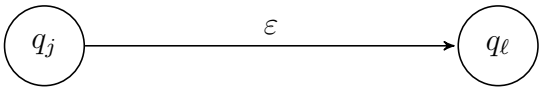
entonces, se ha de añadir un estado q_k entre q_j y q_ℓ y se ha de sustituir el arco que va de q_j a q_ℓ por dos arcos y un bucle. En concreto, un arco de q_j a q_k con etiqueta λ, otro arco de q_k a q_ℓ con etiqueta λ y un bucle sobre q_k con etiqueta H:



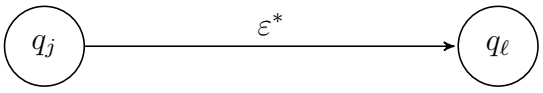
Si H fuese $\varnothing$ o ε, es decir, $R = \varnothing^*$ o $R = \varepsilon^*$, entonces sería suficiente con sustituir $\varnothing^*$ o ε^* por ε. Por tanto, si tenemos



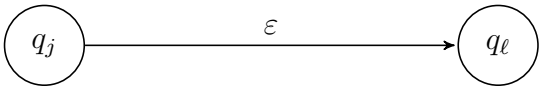
nos quedará



Y si tenemos

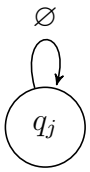


nos quedará



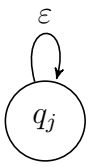
A continuación, se adaptan los casos anteriores a la situación en la que los estados q_j y q_ℓ son el mismo estado. Por tanto, se tiene solo un estado q_j con un bucle etiquetado mediante una ER R:

7. Si tenemos que R es $\varnothing$



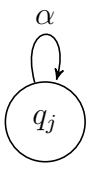
entonces, se ha de eliminar el bucle de q_j:

8. Si tenemos que R es ε

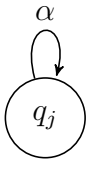


entonces, se ha de eliminar el bucle de q_j:

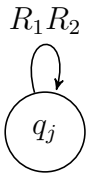
9. Si tenemos que R es un símbolo α del alfabeto $\mathbb{A}$



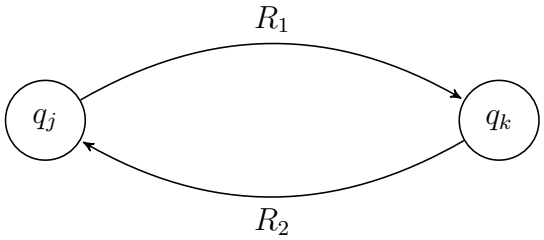
entonces, se ha de mantener el bucle con α:



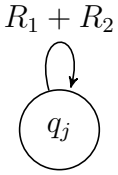
10. Si tenemos que R es la concatenación de dos expresiones regulares R_1 y R_2, es decir, $R = R_1R_2$



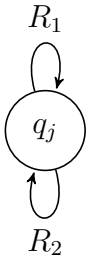
entonces, se ha de añadir un estado q_k y se ha de sustituir el bucle de q_j por dos arcos. En concreto, un arco de q_j a q_k con etiqueta R_1 y un arco de q_k a q_j con etiqueta R_2:



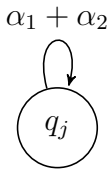
11. Si tenemos que R es la unión de dos expresiones regulares R_1 y R_2, es decir, $R = R_1+R_2$



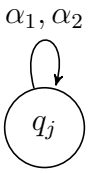
entonces, se ha de sustituir el bucle sobre q_j con etiqueta $R_1 + R_2$ por dos bucles sobre q_j, uno con etiqueta R_1 y el otro con etiqueta R_2:



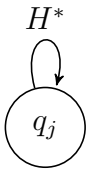
Recordemos que si las expresiones R_1 y R_2 fuesen dos símbolos α_1 y α_2 del alfabeto,



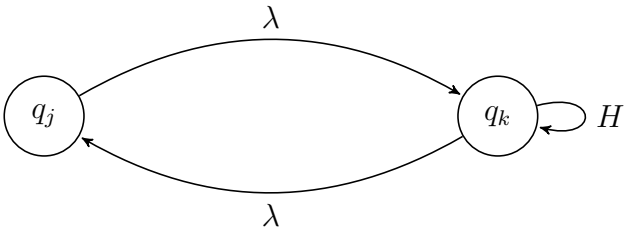
entonces se pondría un único bucle y α_1 y α_2 aparecerían separados por una coma:



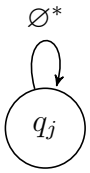
12. Si tenemos que R es la clausura universal de otra expresión regular H, es decir, $R = H^*$



entonces, se ha de añadir un estado q_k y se ha de sustituir el bucle de q_j por dos arcos y un bucle. En concreto, un arco de q_j a q_k con etiqueta λ, un arco de q_k a q_j con etiqueta λ y un bucle sobre q_k con etiqueta H:

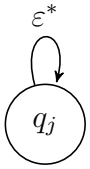


Si H fuese $\varnothing$ o ε, es decir, $R = \varnothing^*$ o $R = \varepsilon^*$, entonces sería suficiente con eliminar el bucle. Por tanto, si tenemos



nos quedará

Y si tenemos



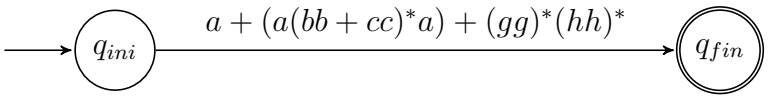
nos quedará

8.2.6 Ejemplos de cálculo de la AF correspondiente a una ER

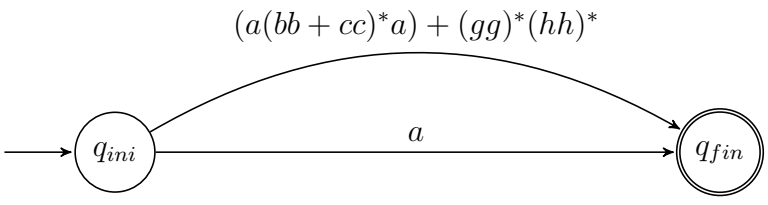
8.2.6.1 Ejemplo 1: de ER a AF

Se desea diseñar un AF para la siguiente expresión regular: $R = a+(a(bb+cc)^*a)+(gg)^*(hh)^*$.

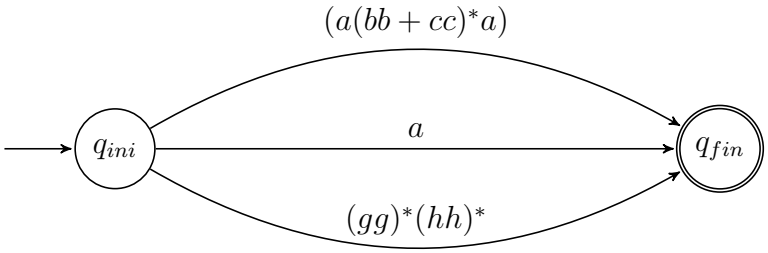
El primer paso es crear un AF con un estado inicial q_{ini} y un estado final o aceptador q_{fin} y con una transición, cuya etiqueta es R, desde q_{ini} a q_{fin}.



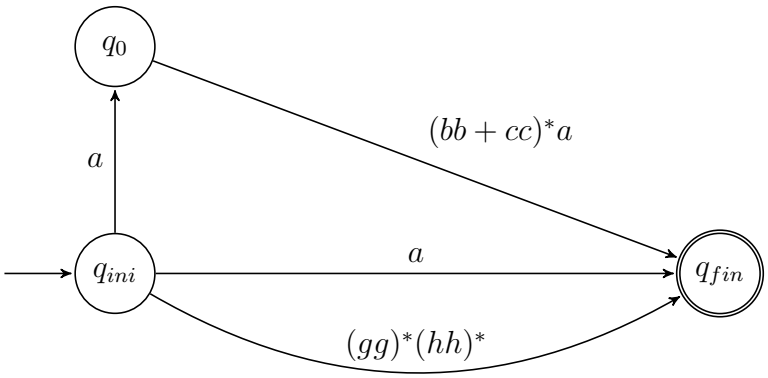
A continuación, consideramos $R_1 = a$ y $R_2 = (a(bb + cc)^*a) + (gg)^*(hh)^*$ y aplicamos el esquema correspondiente a $R_1 + R_2$:



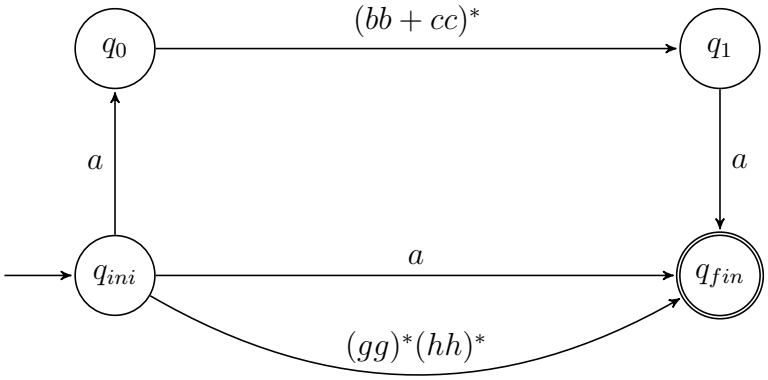
A continuación, consideramos $R_1 = (a(bb+cc)^*a)$ y $R_2 = (gg)^*(hh)^*$ y volvemos a aplicar el esquema correspondiente a $R_1 + R_2$:



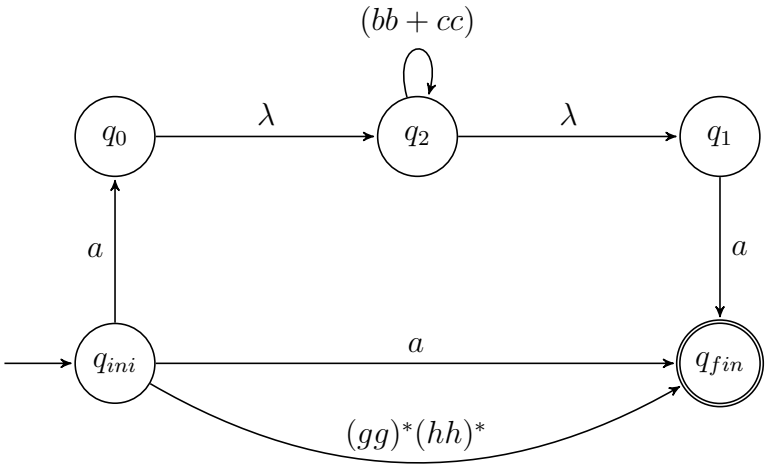
A continuación, consideramos $R_1 = a$ y $R_2 = (bb + cc)^*a$ y aplicamos el esquema correspondiente a R_1R_2:



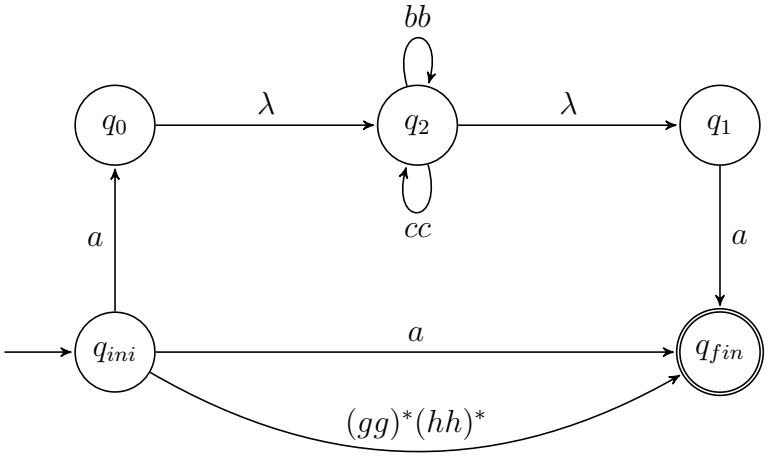
A continuación, consideramos $R_1 = (bb + cc)^*$ y $R_2 = a$ y volvemos a aplicar el esquema correspondiente a R_1R_2:



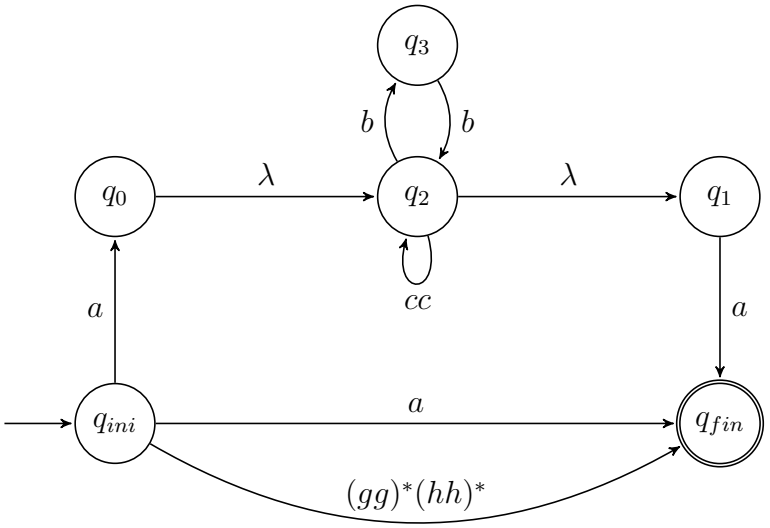
A continuación, consideramos $H^* = (bb + cc)^*$ y aplicamos el esquema correspondiente a H^*:



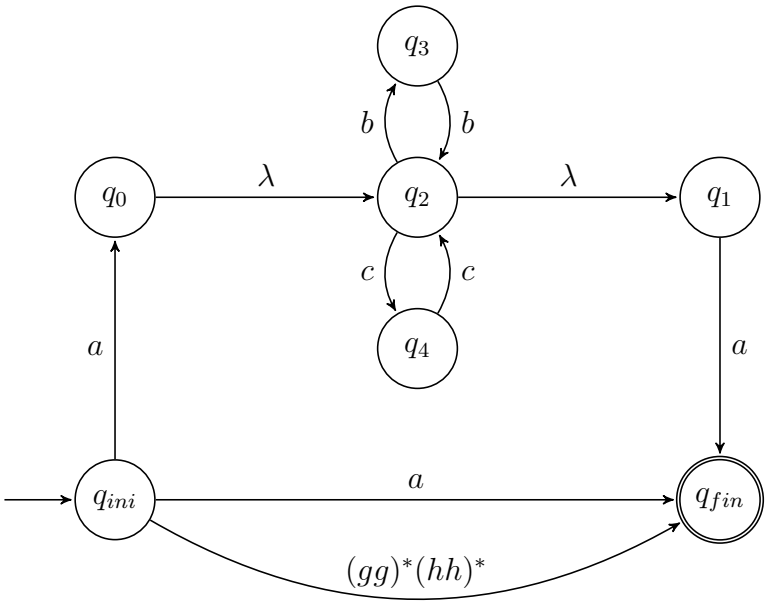
A continuación, consideramos $R_1 = bb$ y $R_2 = cc$ y aplicamos el esquema correspondiente a $R_1 + R_2$:



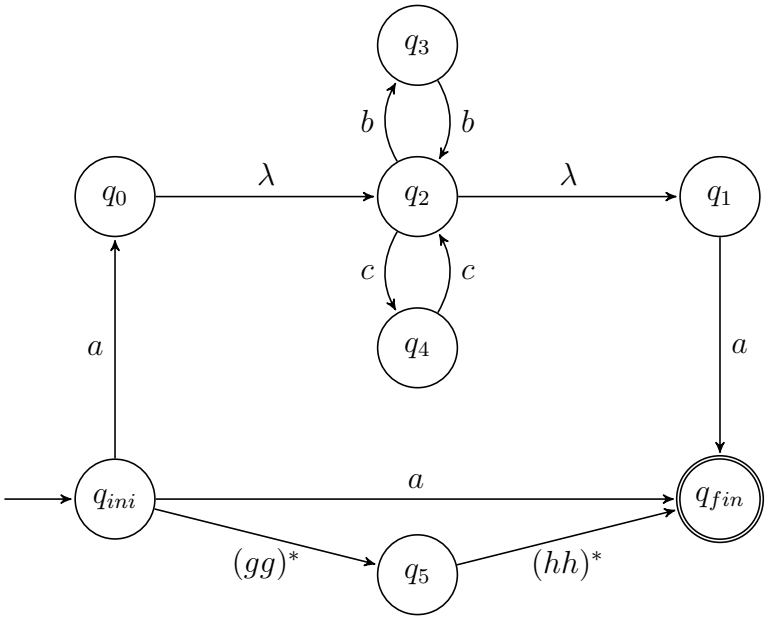
A continuación, consideramos $R_1 = b$ y $R_2 = b$ y aplicamos el esquema correspondiente a $R_1 R_2$:



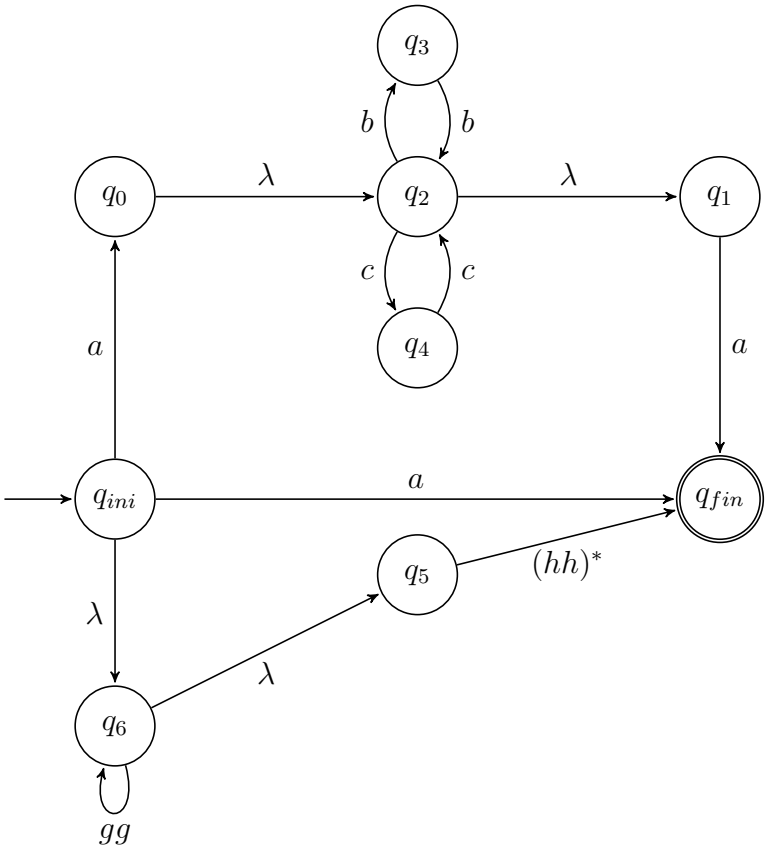
A continuación, consideramos $R_1 = c$ y $R_2 = c$ y aplicamos el esquema correspondiente a R_1R_2:



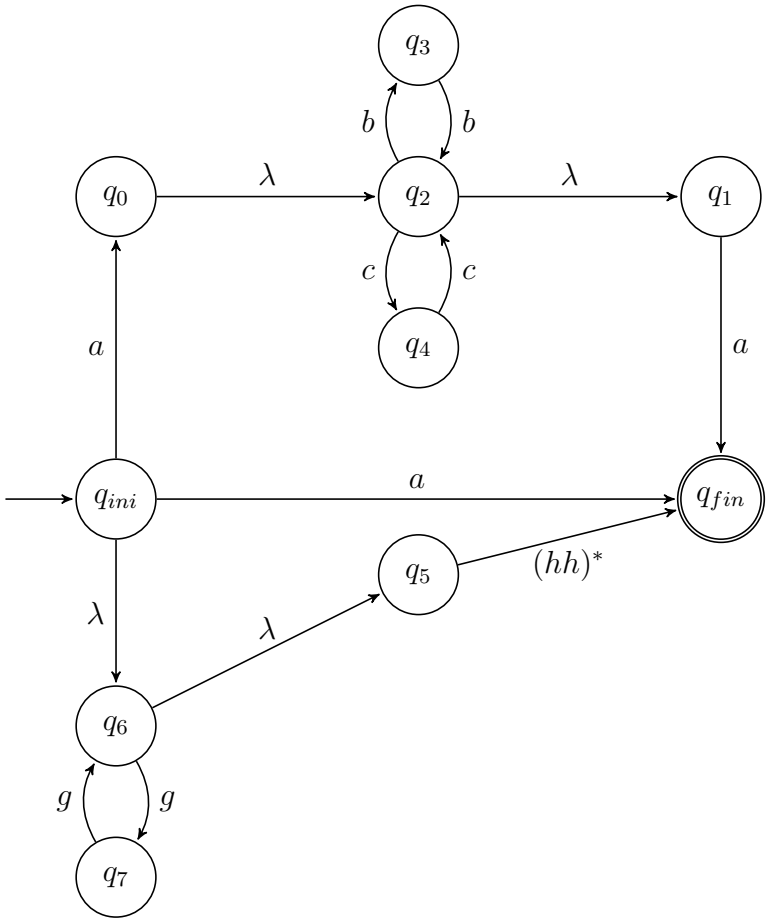
A continuación, consideramos $R_1 = (gg)^*$ y $R_2 = (hh)^*$ y aplicamos el esquema correspondiente a R_1R_2:



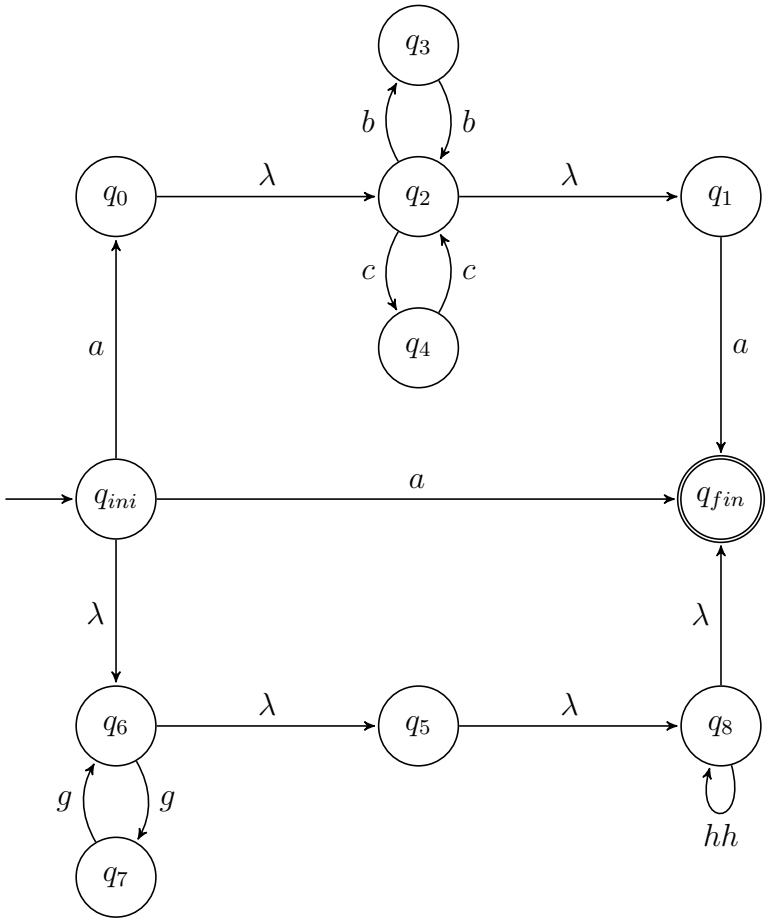
A continuación, consideramos $H = (gg)^*$ y aplicamos el esquema correspondiente a H^*:



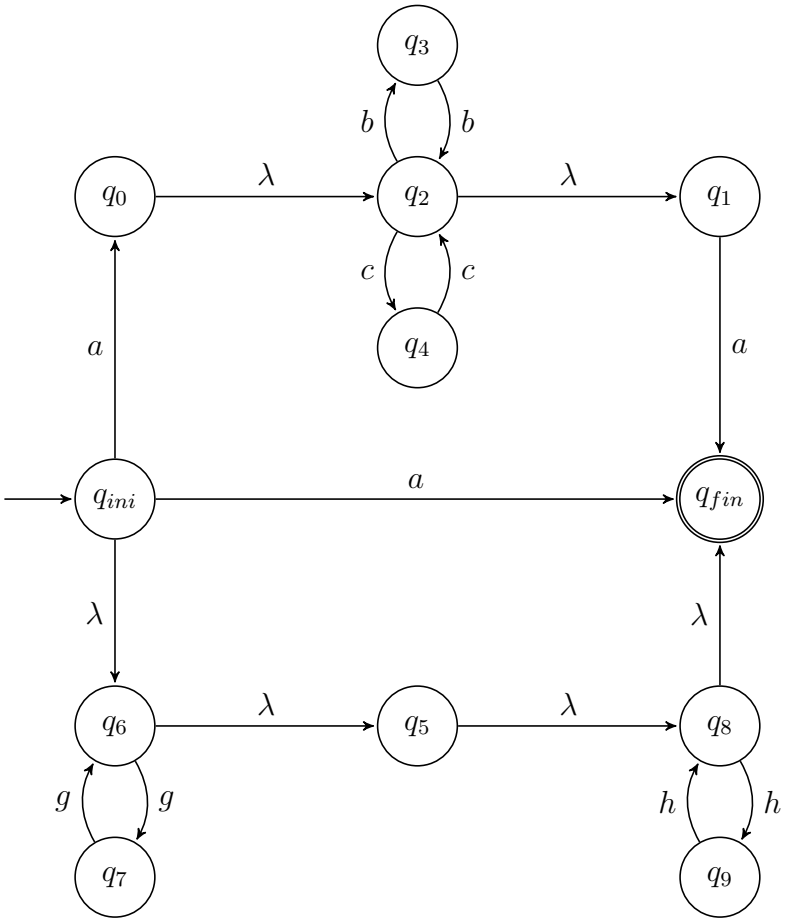
A continuación, consideramos $R_1 = g$ y $R_2 = g$ y aplicamos el esquema correspondiente a R_1R_2:



A continuación, consideramos $H^* = (hh)^*$ y aplicamos el esquema correspondiente a H^*:



A continuación, consideramos $R_1 = h$ y $R_2 = h$ y aplicamos el esquema correspondiente a R_1R_2:



8.2.7 Utilidad de las expresiones regulares

Las expresiones regulares sirven para representar la estructura de las palabras que cumplen una determinada propiedad. Consecuentemente, las ER se utilizan en búsqueda de patrones en cadenas de caracteres.

Por ejemplo, si en una colección de textos queremos encontrar referencias a *Handel*, el primer problema con el que nos encontramos es que dependiendo del autor del texto o del idioma del texto, podemos encontrarnos con tres versiones diferentes: Handel, Händel o Haendel. Estas tres opciones se representan fácilmente mediante la siguiente ER: *H(a + ä + ae)ndel*.

Tenemos un caso parecido para la palabra castellana ‘máximo’. Algunos editores de texto —generalmente editores pensados exclusivamente para el inglés— no permiten tíldes. Debido a ello, si en una colección de textos queremos encontrar la palabra ‘máximo’, el primer problema con el que tenemos es que dependiendo del autor del texto, podemos encontrarnos con dos versiones diferentes: máximo y maximo. Estas dos opciones se representarían mediante la siguiente ER: *m(á + a)ximo*.

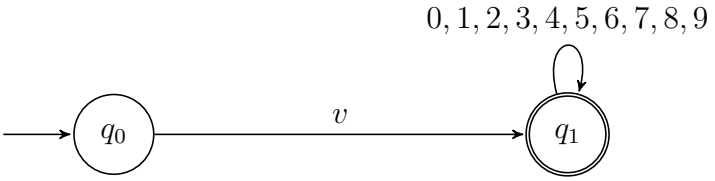
Otro caso es el de la palabra 'septiembre', puesto que también se da por válida la versión 'setiembre'. La ER *se(ε + p)tiembre* representaría las dos posibilidades. Además, en este caso se puede apreciar la utilidad de ε.

Como último ejemplo de este tipo, vamos a mencionar el caso de las formas del subjuntivo 'hubiéramos' y 'hubiésemos', que tienen el mismo significado. La ER *hubi(é + e)(ra + se)mos* representaría todas las opciones, incluidas las apariciones incorrectas que no llevan tílde.

Las ER también se utilizan en el diseño de compiladores, puesto que sirven para establecer reglas que han de cumplir los nombres de las variables o los nombres de las funciones de un lenguaje de programación. Por ejemplo, si tenemos un lenguaje de programación en el que las variables han de empezar con la letra 'v' y luego tener una secuencia de dígitos, la ER que representaría esa estructura sería la siguiente:

$$v(0+1+2+3+4+5+6+7+8+9)^*$$

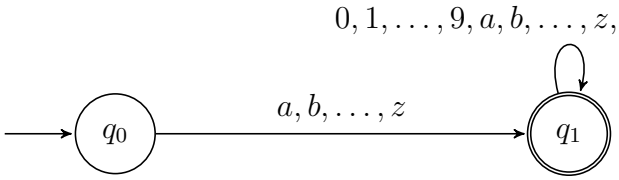
Y el AF sería el siguiente:



Si las variables han de empezar con una letra minúscula y luego pueden tener una secuencia de letras minúsculas, letras mayúsculas, dígitos y el carácter _, la ER que representaría esa estructura tendría la siguiente forma:

$$(a+b+\cdots+z)(0+1+\cdots+9+a+b+\cdots+z+A+B+\cdots+Z+\_)^*$$

Y el AF sería el siguiente:



Realmente, habría que escribir todos los dígitos, todas las letras minúsculas y todas las letras mayúsculas. Se han utilizado puntos suspensivos para abreviar.

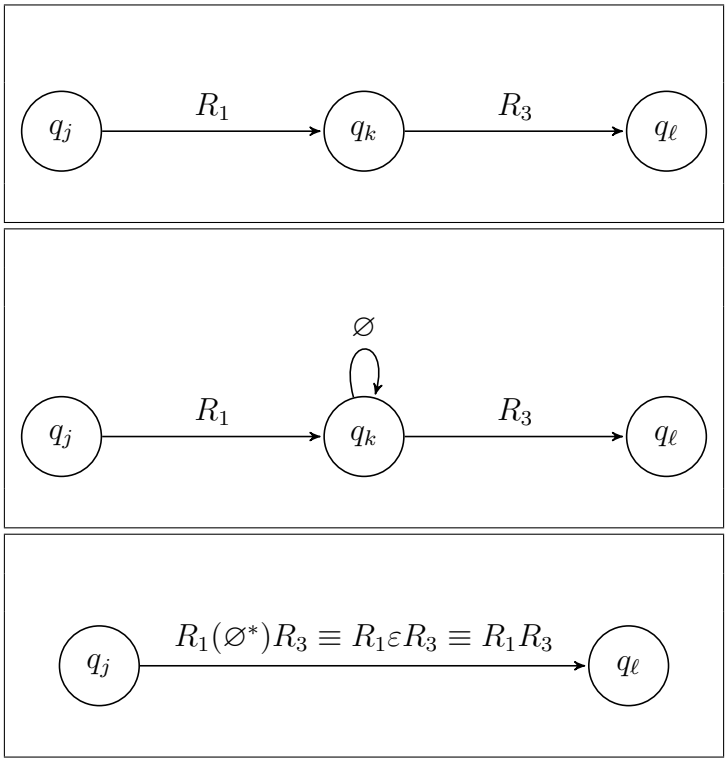


Figura 8.2.4. ER correspondiente a la eliminación del estado q_k: caso general sin bucle.

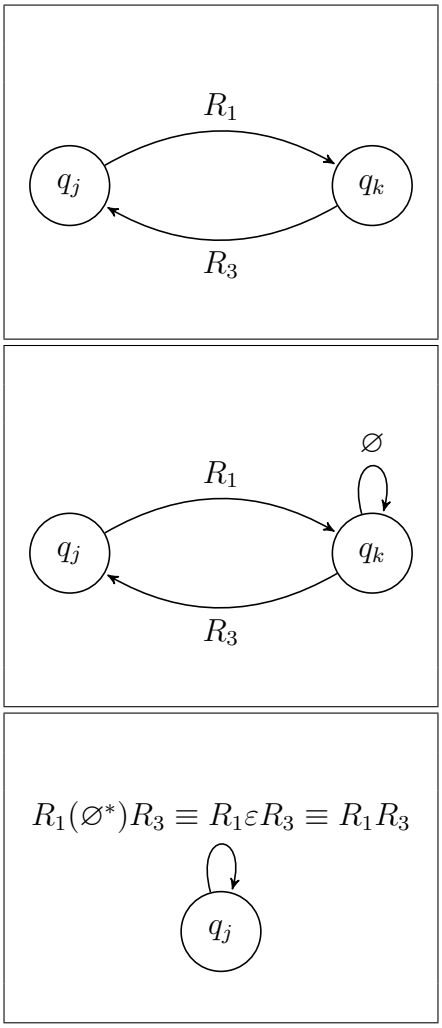


Figura 8.2.5. ER correspondiente a la eliminación del estado q_k: siendo q_j punto de partida y llegada y q_k sin bucle.

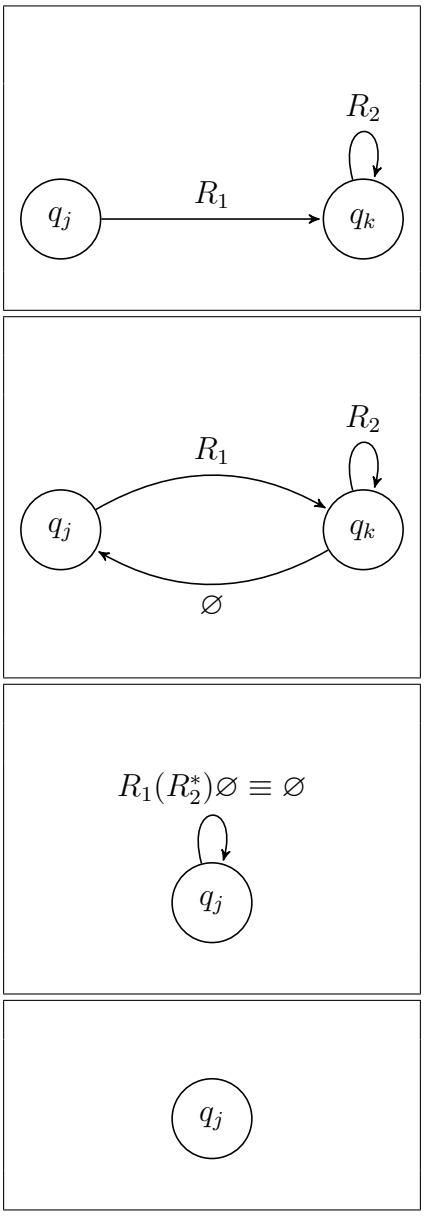


Figura 8.2.6. ER correspondiente a la eliminación del estado q_k, cuando desde q_k no hay acceso a ningún otro estado.

8.3.

Autómatas finitos y gramáticas regulares

En este apartado se presentará un método que, dado un autómata finito, permite obtener de manera sistemática la gramática regular correspondiente que sirve para generar palabras pertenecientes al lenguaje definido por el autómata finito.

8.3.1 Información que aporta un AF sobre cómo generar palabras del lenguaje que define

Dado un AF, ese AF ofrece la información necesaria para poder generar las palabras del lenguaje correspondiente. Lo que ocurre es que esa información no está expresada de tal forma que sea fácil ir generando todas las palabras del lenguaje asociado de manera sistemática. Por tanto, se va a presentar un algoritmo que, a partir del AF, obtenga un conjunto de reglas de producción de palabras. Dichas reglas ofrecerán una manera sistemática para ir generando o produciendo todas las palabras del lenguaje.

8.3.2 Método para calcular la gramática regular correspondiente a un AF

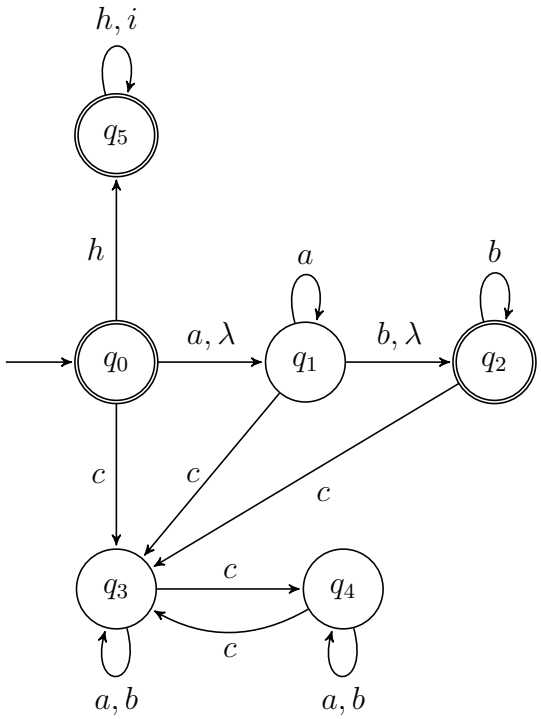
El objetivo es poder formular reglas que sirvan para generar o producir palabras del lenguaje asociado a un AF.

8.3.2.1 Formular las reglas partiendo del estado inicial y averiguando cómo avanzar (inadecuado)

Dado un AF, la primera intuición podría consistir en partir del estado inicial e ir eligiendo las transiciones disponibles con la meta puesta en terminar en un estado final o aceptador. Es importante recordar que una palabra construida de esa forma, será del lenguaje asociado al AF si

se ha terminado en un estado aceptador.

Consideremos el siguiente AF:



El alfabeto es $\mathbb{A} = \{a, b, c, h, i\}$ y el lenguaje asociado a ese AF está formado por tres tipos de palabras:

- La palabra vacía ε.
- Las palabras que tienen un bloque inicial que contiene solo apariciones de a —cero o más apariciones de a— y un bloque final que contiene solo apariciones de b —cero o más apariciones de b.
- Las palabras que empiezan con h y luego tienen un bloque formado por repeticiones de h e i —cero o más repeticiones y mezclados de cualquier manera.

El símbolo c no puede aparecer.

Teniendo en cuenta la idea intuitiva mostrada previamente, para generar una palabra, tenemos que seguir las alternativas que ofrecen las transiciones del AF. Analizamos tres ejemplos:

- Partimos de q_0; pasamos a q_1, para lo cual añadimos una a; luego mediante el bucle de q_1 añadimos, por ejemplo, a tres veces; a continuación pasamos a q_2, para lo cual añadimos una b; y en q_2 mediante el bucle añadimos b, por ejemplo, dos veces. Hemos construido la palabra $aaaabbb\varepsilon$. Esa misma palabra se puede generar también de la siguiente manera:

partimos de q_0; pasamos a q_1 sin añadir a porque tenemos la opción λ; luego mediante el bucle de q_1 añadimos a cuatro veces; a continuación pasamos a q_2 sin añadir b porque tenemos la opción λ; y en q_2 mediante el bucle añadimos b tres veces.

- Partimos de q_0 y nos quedamos en q_0 porque q_0 es un estado aceptador. De esa forma generamos la palabra vacía ε.
- Partimos de q_0, pasamos a q_5, para lo cual añadimos una h, luego, mediante el bucle de q_5 añadimos, por ejemplo, i, i, h, i y h. Hemos construido la palabra $hiihih\varepsilon$.

Diferentes recorridos hechos de esa manera irán generando las infinitas palabras que pertenecen al lenguaje asociado.

Las reglas de producción sirven para formalizar esos recorridos. El planteamiento desarrollado mediante ese ejemplo se puede representar mediante las siguientes reglas de producción:

1. $Q_0 \to \varepsilon$	8. $Q_1 \to Q_2$	15. $Q_3 \to cQ_4$
2. $Q_0 \to aQ_1$	9. $Q_1 \to cQ_3$	16. $Q_4 \to aQ_4$
3. $Q_0 \to Q_1$	10. $Q_2 \to \varepsilon$	17. $Q_4 \to bQ_4$
4. $Q_0 \to cQ_3$	11. $Q_2 \to bQ_2$	18. $Q_4 \to cQ_3$
5. $Q_0 \to hQ_5$	12. $Q_2 \to cQ_3$	19. $Q_5 \to \varepsilon$
6. $Q_1 \to aQ_1$	13. $Q_3 \to aQ_3$	20. $Q_5 \to hQ_5$
7. $Q_1 \to bQ_2$	14. $Q_3 \to bQ_3$	21. $Q_5 \to iQ_5$

Esas reglas de producción se han formulado tomando Q_0 como punto de referencia inicial y representando todos los pasos que se pueden ir dando. Indican cómo ir generando palabras del lenguaje asociado al AF. Q_0 ha de ser siempre el punto de partida. Para poder terminar la construcción de una palabra, es necesario llegar a Q_0, Q_2 o Q_5, puesto que esos tres casos ofrecen la posibilidad de generar ε y terminar. Por ejemplo, para construir la palabra $aaaabbb\varepsilon$ se puede seguir el siguiente proceso de producción:

$$Q_0 \to aQ_1 \to aaQ_1 \to aaaQ_1 \to aaaaQ_1 \to aaaabQ_2 \to$$

$$aaaabbQ_2 \to aaaabbbQ_2 \to aaaabbb\varepsilon$$

Pero también este otro:

$$Q_0 \to Q_1 \to aQ_1 \to aaQ_1 \to aaaQ_1 \to aaaaQ_1 \to aaaaQ_2 \to$$

$$aaaabQ_2 \to aaaabbQ_2 \to aaaabbbQ_2 \to aaaabbb\varepsilon$$

Y hay más opciones, en concreto, dos más.

La construcción de la palabra ε sigue el siguiente proceso de producción:

$$Q_0 \to \varepsilon$$

La construcción de la palabra $hiihih\varepsilon$ sigue el siguiente proceso de producción:

$$Q_0 \to hQ_5 \to hiQ_5 \to hiiQ_5 \to hiihQ_5 \to hiihiQ_5 \to hiihihQ_5 \to hiihih\varepsilon$$

La idea general es que uno parte de Q_0 y puede ir eligiendo las opciones disponibles e ir generando una palabra del lenguaje. Si por ejemplo, partimos de Q_0, aplicamos las reglas 2, 6, 9, 9 y 8, habremos generado la palabra $abbb\varepsilon$.

Este enfoque en el que las reglas de producción se obtienen partiendo de Q_0 y recogiendo las distintas posibilidades de avanzar, tiene un problema cuando se tienen estados desde los cuales no es posible alcanzar ningún estado aceptador. En el AF considerado, desde los estados q_3 y q_4 no es posible llegar a ningún estado aceptador (q_0, q_2, q_5). El AF puede tener miles de estados y no es sencillo averiguar qué caminos conducen a un estado aceptedor y cuáles no. Si en el AF considerado partimos de q_0 y pasamos a q_3 añadiendo c y luego vamos a q_4 añadiendo otra c y seguimos añadiendo símbolos porque no conseguimos llegar a un estado aceptador, nos encontramos inmersos en un proceso infinito que no genera ninguna palabra del lenguaje asociado. Pero, en general, no seremos conscientes de que estamos inmersos en un proceso infinito.

Por ejemplo, podemos partir de Q_0 y seguir el siguiente proceso de construcción:

$$Q_0 \to aQ_1 \to aaQ_1 \to aacQ_3 \to aaccQ_4 \to aaccaQ_4 \to aaccabQ_4 \to aaccabcQ_3 \to \ldots$$

Puesto que el avance se realiza, en general, sin saber hacia dónde vamos, sin darnos cuenta nos hemos metido en un proceso infinito que no lleva a la construcción de ninguna palabra.

La conclusión de todo esto es que la idea intuitiva de formular las reglas de producción a avanzando a partir de q_0 y averiguando qué caminos hay, no es del todo adecuada.

8.3.2.2 Formular las reglas partiendo de los estados aceptadores y retrocediendo (adecuado)

En este enfoque se parte de los estados finales o estados aceptadores y se va retrocediendo para ir calculando todos aquellos caminos que desembocan en estos estados finales.

Retomamos el AF del apartado anterior. El primer paso es formular las reglas de los estados aceptadores:

$$Q_0 \to \varepsilon$$
$$Q_2 \to \varepsilon$$
$$Q_5 \to \varepsilon$$

A continuación, se averigua desde dónde son alcanzables Q_0, Q_2 y Q_5. En concreto, Q_0 no es alcanzable desde ningún estado, pero Q_2 y Q_5 sí:

$$Q_1 \to bQ_2$$
$$Q_1 \to Q_2$$
$$Q_2 \to bQ_2$$
$$Q_0 \to hQ_5$$
$$Q_5 \to hQ_5$$
$$Q_5 \to iQ_5$$

Es decir, Q_2 es alcanzable desde Q_1 con b, desde Q_1 sin ningún símbolo y desde Q_2 también con b. En cuanto a Q_5, este es alcanzable desde Q_0 con h, desde el propio Q_5 con h y también desde el propio Q_5 con i.

Puesto que Q_1 ha aparecido en el lado izquierdo de una regla, hay que averiguar desde dónde es alcanzable Q_1:

$$Q_0 \to aQ_1$$
$$Q_0 \to Q_1$$
$$Q_1 \to aQ_1$$

Es decir, Q_1 es alcanzable desde Q_0 con a, desde Q_0 sin ningún símbolo y desde el propio Q_1 con a.

Puesto que no hay más estados no tratados en el lado izquierdo de ninguna regla, hemos terminado. Las reglas de producción —reordenadas— son las siguientes:

1. $Q_0 \to \varepsilon$
2. $Q_0 \to aQ_1$
3. $Q_0 \to Q_1$
4. $Q_0 \to hQ_5$
5. $Q_1 \to aQ_1$
6. $Q_1 \to bQ_2$
7. $Q_1 \to Q_2$
8. $Q_2 \to \varepsilon$
9. $Q_2 \to bQ_2$
10. $Q_5 \to \varepsilon$
11. $Q_5 \to hQ_5$
12. $Q_5 \to iQ_5$

Ahora se tienen solo 12 reglas de producción, han desaparecido las restantes 9 que estaban asociadas a los estados Q_3 y Q_4, desde los cuales ningún estado aceptador es alcanzable.

Con estas 12 reglas, podemos partir de Q_0 e ir generando palabras del lenguaje. Las reglas de producción representan un algoritmo para producir palabras. El algoritmo representado mediante esas reglas es indeterminista, porque en algunos puntos hay varias opciones para avanzar y hay que elegir una de ellas. El algoritmo no nos indica cuál elegir. Es decir, el camino no está determinado de antemano. Pero cualquier elección que hagamos cuando haya más de una opción será adecuada para seguir construyendo una palabra. Por tanto, estamos ante un **indeterminismo angelical**. Cuando se tiene indeterminismo no angelical, suele ser necesario utilizar la técnica de retroceso o "backtracking" porque algunas elecciones realizadas pueden ser erróneas, pero cuando el indeterminismo es angelical el "backtraking" no es necesario porque cualquier elección realizada es acertada.

8.3.3 Definición formal de gramática regular

El ejemplo del apartado anterior contiene todos los casos que pueden aparecer en un AF: transiciones no vacías, transiciones vacías, bucles, etc. Al calcular las reglas de producción se ha visto que han surgido tres tipos de reglas: las que tienen ε en la parte derecha, las que en el lado derecho tienen un símbolo del alfabeto y un símbolo que representa un estado y, finalmente, las que en el lado derecho tienen solo un símbolo que representa un estado. Teniendo esto en cuenta, se define el concepto de gramática regular.

Una gramática regular es una cuádrupla de la forma (N, T, P, S) donde:

- N es el conjunto de los símbolos no terminales.
- T es el conjunto de los símbolos terminales.
- P es el conjunto de las reglas de producción.
- S es el símbolo no terminal inicial. Por tanto, $S \in N$.

Además, las reglas de producción de P han de tener alguna de las siguientes tres formas:

- $X \to \varepsilon$.
- $X \to \alpha V$.
- $X \to V$.

donde ε es la palabra vacía, α es un símbolo terminal de T y los elementos X y V son símbolos no terminales de N. Además, X y V podrían ser el mismo símbolo no terminal.

El símbolo λ nunca aparece en una gramática regular. Cuando en un AF se tiene una transición con λ, la regla de producción correspondiente tendrá la forma $X \to V$.

Volviendo al AF del apartado anterior, la gramática correspondiente sería una cuádrupla (N, T, P, S) donde:

- $N = \{Q_0, Q_1, Q_2, Q_3, Q_4, Q_5\}$.
- $T = \{a, b, c, h, i\}$.
- P es el conjunto formado por las siguientes reglas de producción:

1. $Q_0 \to \varepsilon$	7. $Q_1 \to Q_2$
2. $Q_0 \to aQ_1$	8. $Q_2 \to \varepsilon$
3. $Q_0 \to Q_1$	9. $Q_2 \to bQ_2$
4. $Q_0 \to hQ_5$	10. $Q_5 \to \varepsilon$
5. $Q_1 \to aQ_1$	11. $Q_5 \to hQ_5$
6. $Q_1 \to bQ_2$	12. $Q_5 \to iQ_5$

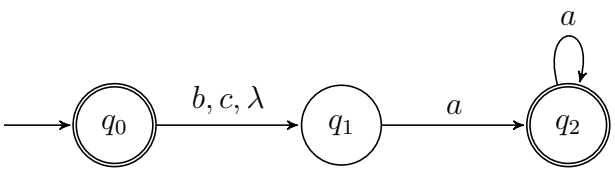
- S es Q_0.

Nótese que el símbolo terminal c y los símbolos no terminales Q_3 y Q_4 no aparecen en ninguna regla de producción.

8.3.4 Ejemplos de cálculo de la GR correspondiente a un AF

8.3.4.1 Ejemplo 1: de AF a GR

Consideramos el siguiente AF:



El primer paso es formular las reglas de los estados aceptadores:

$$Q_0 \to \varepsilon$$
$$Q_2 \to \varepsilon$$

A continuación, se averigua desde dónde son alcanzables Q_0 y Q_2. En concreto, Q_0 no es alcanzable desde ningún estado, pero Q_2 sí:

$$Q_1 \to aQ_2$$
$$Q_2 \to aQ_2$$

Es decir, Q_2 es alcanzable desde Q_1 con a y desde Q_2 con a.

Puesto que Q_1 ha aparecido en el lado izquierdo de una regla, hay que averiguar desde dónde es alcanzable Q_1:

$$Q_0 \to bQ_1$$
$$Q_0 \to cQ_1$$
$$Q_0 \to Q_1$$

Es decir, Q_1 es alcanzable desde Q_0 con b, desde Q_0 con c y desde Q_0 sin ningún símbolo.

Las reglas de producción —reordenadas— son las siguientes:

1. $Q_0 \to \varepsilon$
2. $Q_0 \to bQ_1$
3. $Q_0 \to cQ_1$
4. $Q_0 \to Q_1$
5. $Q_1 \to aQ_2$
6. $Q_2 \to \varepsilon$
7. $Q_2 \to aQ_2$

La gramática correspondiente sería una cuádrupla (N, T, P, S) donde:

- $N = \{Q_0, Q_1, Q_2\}$.
- $T = \{a, b, c\}$.
- P es el conjunto formado por las siguientes reglas de producción:

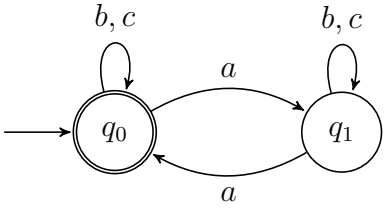
$$\begin{array}{ll} 1.\ Q_0 \to \varepsilon & 5.\ Q_1 \to aQ_2 \\ 2.\ Q_0 \to bQ_1 & 6.\ Q_2 \to \varepsilon \\ 3.\ Q_0 \to cQ_1 & 7.\ Q_2 \to aQ_2 \\ 4.\ \ Q_0 \to Q_1 & \end{array}$$

- S es Q_0.

Nótese que λ nunca aparece en una gramática regular. La λ que aparece en el AF ha dado lugar a la regla de producción $Q_0 \to Q_1$.

8.3.4.2 Ejemplo 2: de AF a GR

Consideramos el siguiente AF:



El primer paso es formular las reglas de los estados aceptadores. En este caso, solo Q_0:

$$Q_0 \to \varepsilon$$

A continuación, se averigua desde dónde es alcanzable Q_0:

$$\begin{array}{l} Q_0 \to bQ_0 \\ Q_0 \to cQ_0 \\ Q_1 \to aQ_0 \end{array}$$

Es decir, Q_0 es alcanzable desde Q_0 con b y con c y desde Q_1 con a.

Puesto que Q_1 ha aparecido en el lado izquierdo de una regla, hay que averiguar desde dónde es alcanzable Q_1:

$$\begin{array}{l} Q_0 \to aQ_1 \\ Q_1 \to bQ_1 \\ Q_1 \to cQ_1 \end{array}$$

Es decir, Q_1 es alcanzable desde Q_0 con a y desde Q_1 con b y c.

Las reglas de producción —reordenadas— son las siguientes:

1. $Q_0 \to \varepsilon$	5. $Q_1 \to aQ_0$
2. $Q_0 \to bQ_0$	6. $Q_1 \to bQ_1$
3. $Q_0 \to cQ_0$	7. $Q_1 \to cQ_1$
4. $Q_0 \to aQ_1$	

La gramática correspondiente sería una cuádrupla (N, T, P, S) donde:

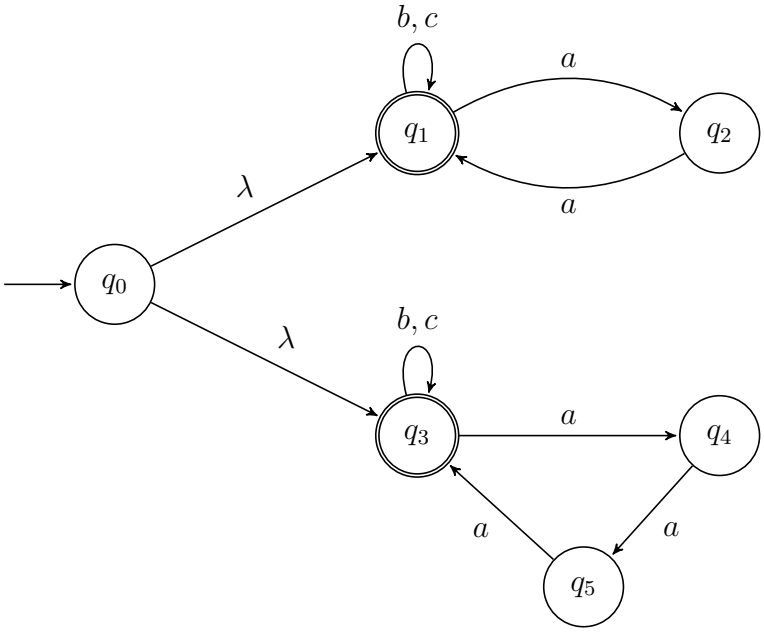
- $N = \{Q_0, Q_1\}$.
- $T = \{a, b, c\}$.
- P es el conjunto formado por las siguientes reglas de producción:

1. $Q_0 \to \varepsilon$	5. $Q_1 \to aQ_0$
2. $Q_0 \to bQ_0$	6. $Q_1 \to bQ_1$
3. $Q_0 \to cQ_0$	7. $Q_1 \to cQ_1$
4. $Q_0 \to aQ_1$	

- S es Q_0.

8.3.4.3 Ejemplo 3: de AF a GR

Consideramos el siguiente AF:



El primer paso es formular las reglas de los estados aceptadores. En este caso, solo Q_0:

$$Q_1 \to \varepsilon$$
$$Q_3 \to \varepsilon$$

A continuación, se averigua desde dónde son alcanzables Q_1 y Q_3:

$$Q_0 \to Q_1$$
$$Q_1 \to bQ_1$$
$$Q_1 \to cQ_1$$
$$Q_2 \to aQ_1$$
$$Q_0 \to Q_3$$
$$Q_3 \to bQ_3$$
$$Q_3 \to cQ_3$$
$$Q_5 \to aQ_3$$

Es decir, Q_1 es alcanzable desde Q_0 sin ningún símbolo, desde Q_1 con b y con c y desde Q_2 con a. Por su parte, Q_3 es alcanzable desde Q_0 sin ningún símbolo, desde Q_3 con b y con c y desde Q_5 con a

Puesto que Q_0, Q_2 y Q_5 han aparecido en el lado izquierdo de una regla, hay que averiguar desde dónde son alcanzables esos tres estados:

$$Q_1 \to aQ_2$$
$$Q_4 \to aQ_5$$

Es decir, Q_0 no es alcanzable desde ningún estado, Q_2 es alcanzable desde Q_1 con a y Q_5 es alcanzable desde Q_4 con a.

Puesto que Q_4 ha aparecido en el lado izquierdo de una regla, hay que averiguar desde dónde es alcanzable Q_4:

$$Q_3 \to aQ_4$$

Es decir, Q_4 es alcanzable desde Q_3 con a.

La gramática correspondiente sería una cuádrupla (N, T, P, S) donde:

- $N = \{Q_0, Q_1, Q_2, Q_3, Q_4, Q_5\}$.
- $T = \{a, b, c\}$.
- P es el conjunto formado por las siguientes reglas de producción:

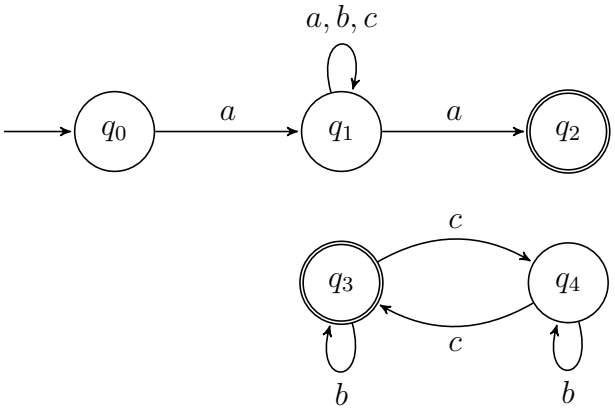
1. $Q_0 \to Q_1$
2. $Q_0 \to Q_3$
3. $Q_1 \to \varepsilon$
4. $Q_1 \to aQ_2$
5. $Q_1 \to bQ_1$
6. $Q_1 \to cQ_1$
7. $Q_2 \to aQ_1$
8. $Q_3 \to \varepsilon$
9. $Q_3 \to aQ_4$
10. $Q_3 \to bQ_3$
11. $Q_3 \to cQ_3$
12. $Q_4 \to aQ_5$
13. $Q_5 \to aQ_3$

- S es Q_0.

Las dos apariciones de λ en el AF, han dado lugar a las reglas de producción $Q_0 \to Q_1$ y $Q_0 \to Q_3$.

8.3.4.4 Ejemplo 4: de AF a GR

Consideramos el siguiente AF:



El primer paso es formular las reglas de los estados aceptadores. En este caso, solo Q_0:

$$Q_2 \to \varepsilon$$
$$Q_3 \to \varepsilon$$

A continuación, se averigua desde dónde son alcanzables Q_2 y Q_3:

$$Q_1 \to aQ_2$$
$$Q_3 \to bQ_3$$
$$Q_4 \to cQ_3$$

Es decir, Q_2 es alcanzable desde Q_1 con a. Por su parte, Q_3 es alcanzable desde Q_3 con b y desde Q_4 con c.

Puesto que Q_1 y Q_4 han aparecido en el lado izquierdo de una regla y todavía no han sido analizados, hay que averiguar desde dónde son alcanzables Q_1 y Q_4:

$$Q_0 \to aQ_1$$
$$Q_1 \to aQ_1$$
$$Q_1 \to bQ_1$$
$$Q_1 \to cQ_1$$
$$Q_3 \to cQ_4$$
$$Q_4 \to bQ_4$$

Es decir, Q_1 es alcanzable desde Q_0 con a y desde Q_1 con a, con b y con c. Por su parte, Q_4 es alcanzable desde Q_3 con c y desde Q_4 con b.

Las reglas de producción —reordenadas— son las siguientes:
La gramática correspondiente sería una cuádrupla (N, T, P, S) donde:

- $N = \{Q_0, Q_1, Q_2, Q_3, Q_4\}$.
- $T = \{a, b, c\}$.
- P es el conjunto formado por las siguientes reglas de producción:

1. $Q_0 \to aQ_1$	5. $Q_1 \to cQ_1$	9. $Q_3 \to cQ_4$
2. $Q_1 \to aQ_1$	6. $Q_2 \to \varepsilon$	10. $Q_4 \to bQ_4$
3. $Q_1 \to aQ_2$	7. $Q_3 \to \varepsilon$	11. $Q_4 \to cQ_3$
4. $Q_1 \to bQ_1$	8. $Q_3 \to bQ_3$	

- S es Q_0.

Nótese que los estados q_3 y q_4 del AF son inalcanzables desde q_0 pero, como q_3 es un estado aceptador, se generan reglas de producción en las que aparecen Q_3 y Q_4. Si ninguno de esos dos estados hubiese sido aceptador, no hubiese surgido ninguna regla de producción en la que apareciese Q_3 o Q_4.

8.3.5 Método para calcular el AF correspondiente a una gramática regular

En este apartado se presenta un método que, dada una GR, de manera sistemática permite obtener un AF que acepta las palabras generadas mediante la GR en cuestión.

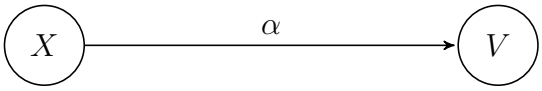
8.3.5.1 Pautas para generar un AF a partir de una GR

Dada una GR $G = (N, T, P, S)$, el AF correspondiente será de la forma $(Q, \mathbb{A}, \tau_{\mathbb{A}}, \tau_{\lambda}, \sigma, Y)$ donde:

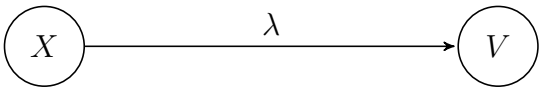
- $Q = N$
- $\mathbb{A} = T$
- $\tau_{\mathbb{A}} = \{(X, \alpha, V) \mid (X \to \alpha V) \in P\}$
- $\tau_{\lambda} = \{(X, V) \mid (X \to V) \in P\}$
- $\sigma = S$

- $Y = \{X \mid (X \to \varepsilon) \in P\}$

Por tanto, cada símbolo no terminal da lugar a un estado del AF. Cada símbolo terminal da lugar a un símbolo del alfabeto. Las reglas de producción de la forma $X \to \alpha V$ —con $\alpha \in T$, $X \in N$ y $V \in N$— dan lugar a transiciones no vacías de la forma



Las reglas de producción de la forma $X \to V$ —con $X \in N$ y $V \in N$— dan lugar a transiciones vacías de la forma



El estado inicial σ del AF será el símbolo inicial S de la GR. El conjunto Y de estados aceptadores del AF estará formado por aquellos símbolos X de N para los cuales exista una regla de producción de la forma $X \to \varepsilon$.

8.3.6 Ejemplos de cálculo del AF correspondiente a una GR

8.3.6.1 Ejemplo 1: de GR a AF

Se desea diseñar un AF $(Q, \mathbb{A}, \tau_{\mathbb{A}}, \tau_{\lambda}, \sigma, Y)$ para la siguiente gramática regular (N, T, P, S):

- $N = \{Z_0, Z_1, Z_2, Z_3, Z_4\}$.
- $T = \{a, b, c\}$.
- P es el conjunto formado por las siguientes reglas de producción:

1. $Z_0 \to aZ_1$	7. $Z_2 \to aZ_3$	13. $Z_4 \to aZ_4$
2. $Z_0 \to Z_2$	8. $Z_2 \to bZ_2$	14. $Z_4 \to bZ_4$
3. $Z_0 \to cZ_4$	9. $Z_2 \to cZ_2$	15. $Z_4 \to cZ_4$
4. $Z_1 \to \varepsilon$	10. $Z_3 \to aZ_2$	16. $Z_4 \to cZ_5$
5. $Z_1 \to aZ_1$	11. $Z_3 \to bZ_3$	17. $Z_5 \to \varepsilon$
6. $Z_2 \to \varepsilon$	12. $Z_3 \to cZ_3$	

- S es Z_0.

Los componentes del AF serán los siguientes:

- $Q = N = \{Z_0, Z_1, Z_2, Z_3, Z_4\}$.
- $\mathbb{A} = T = \{a, b, c\}$.

- Transiciones no vacías:

$$\tau_{\mathbb{A}} = \{(Z_0, a, Z_1), (Z_0, c, Z_4), (Z_1, a, Z_1), (Z_2, a, Z_3), (Z_2, b, Z_2), (Z_2, c, Z_2), (Z_3, a, Z_2), (Z_3, b, Z_3), (Z_3, c, Z_3), (Z_4, a, Z_4), (Z_4, b, Z_4), (Z_4, c, Z_4), (Z_4, c, Z_5)\}$$

  Conjunto obtenido a partir de las reglas de producción que tienen la forma $X \to \alpha V$.

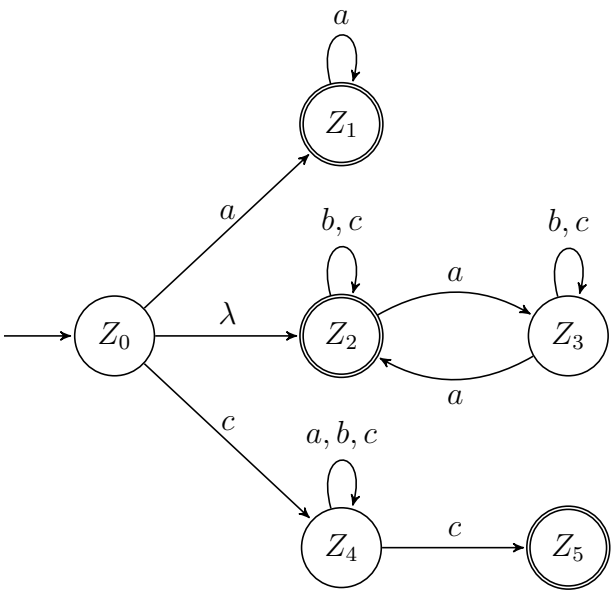
- Transiciones vacías:

$$\tau_{\lambda} = \{(Z_0, Z_2)\}$$

  Conjunto obtenido a partir de las reglas de producción que tienen la forma $X \to V$.

- $\sigma = S = Z_0$.

- $Y = \{Z_1, Z_2, Z_5\}$. Conjunto obtenido a partir de las reglas de producción que tienen la forma $X \to \varepsilon$.



8.3.7 Utilidad de las gramáticas regulares

Las gramáticas regulares sirven para producir palabras que cumplen una determinada propiedad. También sirven para comprobar si una palabra cumple una determinada propiedad. Para ello habrá que averiguar si la palabra puede ser producida teniendo en cuenta las reglas de producción de la gramática regular en cuestión. Habitualmente, las GR se utilizan para definir tipos de datos. Un tipo de dato es una colección de valores que cumplen una determinada propiedad, es decir, tienen una determinada estructura.

8.3.7.1 Tipo de datos $Cero$

Por ejemplo, podemos definir el tipo de datos $Cero$ que solo contiene el cero:

$$1.\ Cero \to \varepsilon$$

El valor 0 se representa ahí mediante la palabra vacía. El símbolo inicial es el único símbolo que se tiene: $Cero$. El conjunto de símbolos no terminales es $\{Cero\}$ y el conjunto de símbolos terminales es vacío.

Para obtener el número 0, es decir, la palabra ε, se sigue el siguiente proceso de producción:

$$Cero \to^1 \varepsilon$$

8.3.7.2 Tipo de datos Nat (números naturales)

También podemos definir el tipo de datos Nat formado por los números naturales (0, 1, 2, 3, etc.). En esta definición se utiliza el tipo $Cero$ definido previamente:

$$1.\ Nat \to Cero$$
$$2.\ Nat \to aNat$$
$$3.\ Cero \to \varepsilon$$

El valor 0 se representa ahí mediante la palabra vacía. Los valores positivos se representan mediante palabras formadas por repeticiones de a. Por ejemplo, el 4 se representa como $aaaa\varepsilon$. El símbolo inicial es Nat. El conjunto de símbolos no terminales es $\{Nat, Cero\}$ y el conjunto de símbolos terminales es $\{a\}$.

Para obtener el número 4, es decir, la palabra $aaaa\varepsilon$, se sigue el siguiente proceso de producción:

$$Nat \to^2 aNat \to^2 aaNat \to^2 aaaNat \to^1 aaaaCero \to^3 aaaa\varepsilon$$

Para obtener el número 0, es decir, la palabra ε, se sigue el siguiente proceso de producción:

$$Nat \to^1 Cero \to^3 \varepsilon$$

8.3.7.3 Tipo de datos Pos (números naturales positivos)

El tipo de datos Pos formado por los números naturales positivos (1, 2, 3, etc.), se puede definir de la siguiente forma:

$$1.\ Pos \to aCero$$
$$2.\ Pos \to aPos$$
$$3.\ Cero \to \varepsilon$$

El símbolo inicial es Pos. El conjunto de símbolos no terminales es $\{Pos, Cero\}$ y el conjunto de símbolos terminales es $\{a\}$.

El número positivo 1, es decir, la palabra $a\varepsilon$ se obtiene de la siguiente forma:

$$Pos \rightarrow^1 aCero \rightarrow^3 a\varepsilon$$

El 4, es decir, la palabra $aaaa\varepsilon$ se obtiene de la siguiente forma:

$$Pos \rightarrow^2 aPos \rightarrow^2 aaPos \rightarrow^2 aaaPos \rightarrow^1 aaaaCero \rightarrow^3 aaaa\varepsilon$$

Por tanto, la última a se añade utilizando la regla $Pos \rightarrow aCero$.

8.3.7.4 Tipo de datos Nat (números naturales, definición alternativa)

Una vez definido el tipo Pos, es posible redefinir el tipo Nat de la siguiente forma:

1. $Nat \rightarrow Cero$
2. $Nat \rightarrow Pos$
3. $Cero \rightarrow \varepsilon$
4. $Pos \rightarrow aCero$
5. $Pos \rightarrow aPos$

El símbolo inicial es Nat. El conjunto de símbolos no terminales es $\{Nat, Cero, Pos\}$ y el conjunto de símbolos terminales es $\{a\}$.

El 4, es decir, la palabra $aaaa\varepsilon$ se obtiene de la siguiente forma:

$$Nat \rightarrow^2 Pos \rightarrow^5 aPos \rightarrow^5 aaPos \rightarrow^5 aaaPos \rightarrow^4 aaaaCero \rightarrow^3 aaaa\varepsilon$$

El 0, es decir, la palabra ε se obtiene de la siguiente forma:

$$Nat \rightarrow^1 Cero \rightarrow^3 \varepsilon$$

8.3.7.5 Tipos de datos Par e $Impar$ (números naturales pares e impares)

El tipo de datos Par formado por los números naturales pares (0, 2, 4, etc.), se puede definir de la siguiente forma:

1. $Par \rightarrow Cero$
2. $Par \rightarrow aImpar$
3. $Cero \rightarrow \varepsilon$
4. $Impar \rightarrow aPar$

El símbolo inicial es Par. El conjunto de símbolos no terminales es $\{Par, Cero, Impar\}$ y el conjunto de símbolos terminales es $\{a\}$.

El 0, es decir, la palabra ε se obtiene de la siguiente forma:

$$Par \rightarrow^1 Cero \rightarrow^3 \varepsilon$$

El 4, es decir, la palabra $aaaa\varepsilon$ se obtiene de la siguiente forma:

$$Par \rightarrow^2 aImpar \rightarrow^4 aaPar \rightarrow^2 aaaImpar \rightarrow^4 aaaaPar \rightarrow^1 aaaaCero \rightarrow^3 aaaa\varepsilon$$

Por tanto, la última a se añade utilizando la regla $Impar \rightarrow aPar$.

En esa gramática regular el símbolo inicial es Par y la gramática sirve para generar los números pares. Si quisiésemos definir el tipo de los números impares, nos servirían esas mismas reglas pero tendríamos que generar una gramática con $Impar$ como símbolo inicial.

8.3.7.6 Tipo de datos Neg (números naturales negativos)

El tipo de datos Neg formado por los números enteros negativos (-1, -2, -3, etc.), se puede definir de la siguiente forma:

1. $Neg \rightarrow bPos$
2. $Pos \rightarrow aCero$
3. $Pos \rightarrow aPos$
4. $Cero \rightarrow \varepsilon$

Es decir, un número entero negativo es un número entero positivo precedido por un símbolo diferente (por ejemplo, b) que representa el signo negativo.

El símbolo inicial es Neg. El conjunto de símbolos no terminales es $\{Neg, Pos, Cero\}$ y el conjunto de símbolos terminales es $\{a, b\}$.

El -4, es decir, la palabra $baaaa\varepsilon$ se obtiene de la siguiente forma:

$$Neg \rightarrow^1 bPos \rightarrow^3 baPos \rightarrow^3 baaPos \rightarrow^3 baaaPos \rightarrow^2 baaaaCero \rightarrow^4 baaaa\varepsilon$$

El -1, es decir, la palabra $ba\varepsilon$ se obtiene de la siguiente forma:

$$Neg \rightarrow^1 bPos \rightarrow^2 baCero \rightarrow^4 ba\varepsilon$$

8.3.7.7 Tipo de datos Ent (números enteros)

El tipo de datos Ent formado por los números enteros ($\ldots, -3, -2, -1, 0, 1, 2, 3 \ldots$), se puede definir de la siguiente forma:

1. $Ent \to Cero$
2. $Ent \to Pos$
3. $Ent \to Neg$
4. $Cero \to \varepsilon$
5. $Pos \to aCero$
6. $Pos \to aPos$
7. $Neg \to bPos$

Es decir, un número entero es o bien cero, o bien un número entero positivo o si no, un número entero negativo.

El símbolo inicial es Ent. El conjunto de símbolos no terminales es $\{Ent, Cero, Pos, Neg\}$ y el conjunto de símbolos terminales es $\{a, b\}$.

El 4, es decir, la palabra $aaaa\varepsilon$ se obtiene de la siguiente forma:

$$Ent \to^2 Pos \to^6 aPos \to^6 aaPos \to^6 aaaPos \to^5 aaaaCero \to^4 aaaa\varepsilon$$

El -4, es decir, la palabra $baaaa\varepsilon$ se obtiene de la siguiente forma:

$$Ent \to^3 Neg \to^7 bPos \to^6 baPos \to^6 baaPos \to^6 baaaPos \to^5 baaaaCero \to^4 baaaa\varepsilon$$

8.3.7.8 Tipo de datos Dec (números naturales en el sistema decimal)

También es posible definir una gramática regular que genera los número naturales en el formato decimal habitual:

1. $Dec \to 0Fin$
2. $Dec \to 1C$
3. $Dec \to 2C$
4. $Dec \to 3C$
5. $Dec \to 4C$
6. $Dec \to 5C$
7. $Dec \to 6C$
8. $Dec \to 7C$
9. $Dec \to 8C$
10. $Dec \to 9C$
11. $C \to Fin$
12. $C \to 0C$
13. $C \to 1C$
14. $C \to 2C$
15. $C \to 3C$
16. $C \to 4C$
17. $C \to 5C$
18. $C \to 6C$
19. $C \to 7C$
20. $C \to 8C$
21. $C \to 9C$
22. $Fin \to \varepsilon$

El símbolo inicial es Dec. El conjunto de símbolos no terminales es $\{Dec, C, Fin\}$ y el conjunto de símbolos terminales es $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Esa gramática impide que haya más de un 0 al principio de un número. Por ejemplo, la palabra 00087902ε no puede ser generada mediante esa gramática, pero la palabra 87902ε sí. De manera similar, la palabra 0000ε no puede ser generada mediante esa gramática, pero la palabra 0ε sí.

8.3.7.9 Tipo de datos $Color$

El tipo de datos $Color$ formado por los colores amarillo, blanco, marrón, negro, rojo, rosa, verde y azul, se puede definir de la siguiente forma:

1. $Color \rightarrow aFin$
2. $Color \rightarrow bFin$
3. $Color \rightarrow mFin$
4. $Color \rightarrow nFin$
5. $Color \rightarrow rFin$
6. $Color \rightarrow sFin$
7. $Color \rightarrow vFin$
8. $Color \rightarrow zFin$
9. $Fin \rightarrow \varepsilon$

Por tanto, cada color es representado mediante un símbolo. Realmente, se crea una palabra de un único símbolo.

El símbolo inicial es $Color$. El conjunto de símbolos no terminales es $\{Color, Fin\}$ y el conjunto de símbolos terminales es $\{a, b, m, n, r, s, v, z\}$.

La palabra correspondiente al color amarillo, es decir, la palabra $a\varepsilon$ se obtiene de la siguiente forma:

$$Color \rightarrow^5 aFin \rightarrow^1 a\varepsilon$$

La palabra correspondiente al color blanco, es decir, la palabra $b\varepsilon$ se obtiene de la siguiente forma:

$$Color \rightarrow^8 bFin \rightarrow^1 b\varepsilon$$

Para generar la palabra correspondiente al color deseado, hay que elegir la regla de producción adecuada.

8.4.

Lenguajes no regulares

La pregunta que surge ante la equivalencia entre autómatas finitos y lenguajes regulares es la siguiente ¿Todos los lenguajes son regulares? o planteándolo de otro modo ¿para todo lenguaje $\mathcal{L}$ perteneciente a $2^{\mathbb{A}^*}$, existe algún autómata finito que es capaz de definir $\mathcal{L}$? La respuesta es que no.

8.4.1 Existen lenguajes no regulares

En este apartado se va a probar que para cualquier alfabeto $\mathbb{A}$, existen lenguajes no regulares definidos sobre $\mathbb{A}$.

Consideramos un alfabeto cualquiera $\mathbb{A}$ y el alfabeto $\mathbb{B} = \mathbb{A} \cup \{\varepsilon, \varnothing, {}^*, +, (,)\}$.

Todos los lenguajes regulares definibles sobre $\mathbb{A}$ se pueden representar mediante expresiones regulares definidas sobre $\mathbb{A}$, es decir, mediante palabras pertenecientes a $\mathbb{B}^*$. El conjunto $\mathbb{B}^*$ es enumerable pero $2^{\mathbb{A}^*}$ no es enumerable. Esto quiere decir que hay más elementos (lenguajes) en $2^{\mathbb{A}^*}$ que elementos (palabras) en $\mathbb{B}^*$. Por consiguiente, tenemos que hay lenguajes de $2^{\mathbb{A}^*}$ que no son expresables mediante palabras de $\mathbb{B}^*$. Esos lenguajes de $2^{\mathbb{A}^*}$ que no son expresables mediante palabras de $\mathbb{B}^*$ no son regulares. Por tanto, hay lenguajes de $2^{\mathbb{A}^*}$ que no son regulares.

8.4.2 Lenguajes incontextuales: autómatas con pila

Los autómatas con pila (AP) son una extensión de los AF. En concreto, la idea es añadir una memoria adicional con las características de una pila. Una pila solo permite dos operaciones básicas: Poner un elemento en la cima (apilar) y quitar el elemento de la cima (desapilar). Por tanto, no se puede acceder directamente a los elementos que no están en la cima de la pila. La única manera de acceder a un elemento x que no está en la cima es ir quitando elementos hasta que x quede en la cima, pero los elementos que se han quitado para poder llegar a x se pierden. Consecuentemente, la memoria añadida (la pila) tiene limitaciones pero incluso con

esas limitaciones, es posible definir lenguajes que no son definibles mediante los AF.

Las transiciones de un AP pueden ser de varios tipos. A continuación se presentan los tipos de transiciones utilizados en este documento y el significado:

- α/N donde $\alpha \in \mathbb{A}$: Si se lee α en el dispositivo de entrada, no hacer nada en la pila.
- λ/N: cambiar de estado sin leer desde el dispositivo de entrada y no hacer nada en la pila.
- $\alpha/A/\gamma$ donde $\alpha \in \mathbb{A}$ y $\gamma \in \mathbb{A}$: Si se lee α en el dispositivo de entrada, apilar γ en la pila.
- $\alpha/D/\gamma$ donde $\alpha \in \mathbb{A}$ y $\gamma \in \mathbb{A}$: Si se lee α en el dispositivo de entrada, desapilar el elemento de la cima si es γ. Si la pila está vacía o el elemento de la cima no es γ, el AP se detiene y responde "No".
- $\alpha_1, \alpha_2, \alpha_3/A/\gamma_1, \gamma_2, \gamma_3$ donde $\alpha_1, \alpha_2, \alpha_3, \gamma_1, \gamma_2, \gamma_3 \in \mathbb{A}$: Si se lee α_1 en el dispositivo de entrada, apilar γ_1 en la pila; Si se lee α_2 en el dispositivo de entrada, apilar γ_2 en la pila; Si se lee α_3 en el dispositivo de entrada, apilar γ_3 en la pila.
- $\alpha_1, \alpha_2, \alpha_3/D/\gamma_1, \gamma_2, \gamma_3$ donde $\alpha_1, \alpha_2, \alpha_3, \gamma_1, \gamma_2, \gamma_3 \in \mathbb{A}$: Si se lee α_1 en el dispositivo de entrada, desapilar el elemento de la cima si es γ_1; Si se lee α_2 en el dispositivo de entrada, desapilar el elemento de la cima si es γ_2; Si se lee α_3 en el dispositivo de entrada, desapilar el elemento de la cima si es γ_3. Si la pila está vacía o el elemento de la cima no es el esperado (γ_1, γ_2 o γ_3, según el caso), el AP se detiene y responde "No".

Para que un AP responda "Sí", se tienen que cumplir las siguientes condiciones:

- Que el AP termine en un estado aceptador (estado con doble círculo).
- Que se haya leído toda la palabra.
- Que se hayan podido realizar todas las operaciones en la pila. Es decir, siempre que se haya intentado desapilar el elemento de la cima de la pila, que la pila no estuviera vacía y que el elemento de la cima fuese el elemento esperado (o adecuado).
- Que al final la pila quede vacía.

Al principio la pila siempre estará vacía.

En los AP, las configuraciones tienen tres componentes: (w, q_i, p) donde w es el trozo de palabra que todavía no se ha leído, q_i es el estado actual y p es la pila.

Un AP puede ser determinista o no determinista. En el caso de los AF, el determinismo y el no determinismo son equivalentes, es decir, el no determinismo no añade expresividad o poder de definir más lenguajes. En cambio, en el caso de los AP, el no determinismo añade expresividad o poder de definir más lenguajes. De esta forma, hay lenguajes que se pueden definir

mediante un AP no determinista pero que no pueden ser definidos mediante un AP determinista. Por otro lado, todo lenguaje definible mediante un AP determinista es también definible mediante un AP no determinista. Los lenguajes definibles mediante un AP determinista reciben el nombre de **lenguajes incontextuales deterministas**. Los lenguajes definibles mediante un AP no determinista reciben el nombre de **lenguajes incontextuales**. Los lenguajes incontextuales deterministas forman un subconjunto estricto de los lenguajes incontextuales.

Los lenguajes incontextuales reciben también el nombre de **lenguajes independientes del contexto**.

Recordemos que en un autómata no determinista (con o sin pila), dada una palabra hay, en general, más de una manera de moverse por los estados, es decir, más de una computación posible. Pero es suficiente con que exista una computación que termine cumpliendo los criterios para responder "Sí" para que la palabra en cuestión sea aceptada.

8.4.2.1 Ejemplo: Lenguaje formado por las palabras de la forma $(a\varepsilon)^k \cdot (b\varepsilon)^k$ con $k \in \mathbb{N}$

Vamos a considerar ahora un lenguaje para el cual no se puede construir un autómata finito:

$$G_1 = \{w \mid w \in \mathbb{A}^* \wedge \exists k(k \in \mathbb{N} \wedge w = (a\varepsilon)^k \cdot (b\varepsilon)^k)\}$$

El lenguaje infinito G_1 está definido sobre el alfabeto $\mathbb{A} = \{a, b, c\}$ y lo forman las palabras en las que la primera mitad es una secuencia de a-s y la segunda mitad es una secuencia de b-s, y donde esas dos secuencias son de la misma longitud. Por tanto, $G_1 = \{\varepsilon, ab\varepsilon, aabb\varepsilon, aaabbb\varepsilon, aaaabbbb\varepsilon, \dots\}$. El lenguaje G_1 es un sublenguaje del lenguaje L_9 del apartado 8.2.2.2 de la página 18. Aunque L_9 es regular, G_1 no es regular. Un detalle importante es el hecho de que una palabra de G_1 puede tener cualquier número de apariciones de a (0, 1, 2, 3, etc.) seguidos del mismo número de apariciones de b. Los autómatas finitos implementan la memoria mediante un número finito de estados y, por tanto, la memoria es finita. Los AF son capaces de recordar ciertas informaciones. Por ejemplo, es posible definir un autómata finito que es capaz de recordar si el número de apariciones de a es par. También es posible definir un autómata finito que es capaz de comprobar si el número de apariciones de a es 20 y el número de apariciones de b es 20 —o cualquier otra constante que ha de ser conocida antes de definir el AF. Pero el problema en el lenguaje G_1 es que antes de definir el AF no se conoce el número de apariciones de a y de b. El número de apariciones de a y de b varia de palabra a palabra, ese número no es siempre el mismo. Consecuentemente, hay que contar el número exacto de apariciones de a para luego comprobar que el número de apariciones de b coincide, y un autómata finito no es capaz de hacer eso considerando que ese número de apariciones de a y de b no es conocido de antemano y puede ser cualquier número (0, 1, 2, 3, etc.).

Por tanto, el lenguaje G_1 no es regular. Para el lenguaje G_1 se necesita el autómata determinista con pila (APD) que aparece en la figura 8.4.1 de la página 91. Ese autómata parte con

la pila vacía y va apilando, en la pila, las apariciones iniciales de a. Cuando aparece la primera b, va desapilando una a por cada b que se lee. De esa manera, se comprueba si la palabra está formada por una secuencia de apariciones de a seguida por una secuencia de apariciones de b de igual longitud. Si la longitud de la secuencia de apariciones de a coincide con la longitud de la secuencia de apariciones de b, la pila quedará vacía cuando se lea la última b desde el dispositivo de entrada de datos y se sabrá que la palabra tiene la forma $(a\varepsilon)^k \cdot (b\varepsilon)^k$ con $k \in \mathbb{N}$. Por otra parte, las razones por las que una palabra no tenga la forma $(a\varepsilon)^k \cdot (b\varepsilon)^k$ con $k \in \mathbb{N}$ pueden ser varias: que la palabra tenga menos apariciones de a que de b ($aabbbb\varepsilon$); que la palabra tenga más apariciones de a que de b ($aaaabbb\varepsilon$); que, independientemente del número de apariciones de a y b, las apariciones de a y b no estén en el orden adecuado ($aabbab\varepsilon$, $bababb\varepsilon$); que aparezca c ($aaabbbc\varepsilon$, $accaabcb\varepsilon$). Nótese que, si en cualquier momento aparece una c o si estando en q_1 aparece a, el autómata se bloquea y responde con un "No". Como este autómata con pila es determinista, dada una palabra hay una única manera de moverse por los estados. Para poder realizar la operación de desapilar una a, la pila no puede estar vacía y, además, el primer símbolo a leer en el dispositivo de entrada ha de ser b. Si estando en q_1 todavía hay más apariciones de b en el dispositivo de entrada pero la pila está vacía, el autómata se bloquea porque no puede realizar la operación de desapilar una a y la respuesta final será "No". Para que el autómata con pila responda "Sí", se ha de poder leer toda la palabra de la entrada y que la pila quede vacía.

A modo de ejemplo, a continuación se muestra la manera de proceder del APD en cuestión, para cuatro palabras concretas. Recordemos que las configuraciones tienen tres componentes: (w, q_i, p) donde w es el trozo de palabra que todavía no se ha leído, q_i es el estado actual y p es la pila. En este ejemplo, la pila se representa como una lista donde el elemento de la cima es el elemento del extremo izquierdo:

1. $(aabb\varepsilon, q_0, [\,]) \to (abb\varepsilon, q_0, [a]) \to (bb\varepsilon, q_0, [a, a]) \to (b\varepsilon, q_1, [a]) \to (\varepsilon, q_1, [\,])$. Primero se han apilado las a-s. Luego se ha desapilado una a por cada b que se ha leído. El APD respode que "Sí" porque se ha acabado en un estado de doble círculo y tras leer la última b y desapilar la a de la pila, la pila ha quedado vacía. Por tanto, se cumplen las condiciones necesarias para poder afirmar que $aabb\varepsilon$ pertenece a G_1.

2. $(aaabb\varepsilon, q_0, [\,]) \to (aabb\varepsilon, q_0, [a]) \to (abb\varepsilon, q_0, [a, a]) \to (bb\varepsilon, q_0, [a, a, a]) \to (b\varepsilon, q_1, [a, a]) \to (\varepsilon, q_1, [a])$. En este caso no se ha acabado bien: tras leer la última b y desapilar una copia de a de la pila, la pila no ha quedado vacía. Por tanto, $aabb\varepsilon$ no pertenece a G_1.

3. $(aabbb\varepsilon, q_0, [\,]) \to (abbb\varepsilon, q_0, [a]) \to (bbb\varepsilon, q_0, [a, a]) \to (bb\varepsilon, q_0, [a]) \to (b\varepsilon, q_1, [\,])$. Tampoco en este caso se ha acabado bien: tras leer la última b no es posible desapilar una copia de a de la pila. Por tanto, $aabbb\varepsilon$ no pertenece a G_1.

4. $(aacbb\varepsilon, q_0, [\,]) \to (acbb\varepsilon, q_0, [a]) \to (cbb\varepsilon, q_0, [a, a])$. Tampoco en este caso se ha acabado bien: tras leer c, el autómata se detiene. No se ha conseguido leer toda la palabra y terminar con la pila vacía. Por tanto, $aacbb\varepsilon$ no pertenece a G_1.

8.4.2.2 Ejemplo: Lenguaje formado por las palabras de la forma $(a\varepsilon)^i \cdot (b\varepsilon)^j \cdot (c\varepsilon)^k$ con $i, j, k \in \mathbb{N}$ e $i = j$ o $i = k$

Vamos a considerar ahora otro lenguaje definido sobre el alfabeto $\mathbb{A} = \{a, b, c\}$ para el cual no se puede construir un autómata finito:

$$G_2 = \{w \mid w \in \mathbb{A}^* \wedge \exists i, j, k (i, j, k \in \mathbb{N} \wedge w = (a\varepsilon)^i \cdot (b\varepsilon)^j \cdot (c\varepsilon)^k) \wedge (i = j \vee i = k)\}$$

Es decir, G_2 contiene palabras como ε, $aaabbb\varepsilon$, $aaabbbc\varepsilon$, $aaacccc\varepsilon$, $aaabcccc\varepsilon$ y $aabbcc\varepsilon$. Pero no contiene palabras como $abbcccc\varepsilon$ y $aab\varepsilon$.

En la figura 8.4.2 de la página 91, se muestra un autómata no determinista con pila (AP) que sirve para reconocer el lenguaje G_2. Ese autómata parte con la pila vacía y va apilando las apariciones iniciales de a en la pila hasta que aparezca b o c. Al aparecer b o c tiene dos opciones:

(a) Seguir el camino q_1, q_2 e ir desapilando una copia de a por cada b que se lee. Si al terminar las apariciones de b se termina también de desapilar las copias de a en la pila, ya no hace falta comprobar el número de apariciones de c porque ya se sabe que el número de apariciones de a y el número de apariciones de b coinciden.

(b) Seguir el camino q_3, q_4 e ir leyendo las apariciones de b sin desapilar ninguna a y cuando aparezca la primera c, ir desapilando una a por cada c que se lee. Si al terminar con las apariciones de c se termina también de desapilar las copias de a, se sabe que el número de apariciones de a y el número de apariciones de c coinciden.

El primer camino es para comprobar si el número de a-s coincide con el número de b-s. El segundo camino es para comprobar si el número de a-s coincide con el número de c-s. El asunto es que al aparecer la primera b, no se sabe si apostar por comparar el número de a-s con el número de b-s o apostar por comparar el número de a-s con el número de c-s. He ahí el indeterminismo. No se nos indica qué camino concreto seguir; se nos muestran las opciones posibles.

El lenguaje G_2 no puede ser definido ni mediante un AF ni mediante un AP determinista. Es necesario un AP no determinista.

Por ejemplo, para la palabra $aabcc\varepsilon$ hay varias posibilidades:

1. $$(aabcc\varepsilon, q_0, [\,]) \to (abcc\varepsilon, q_0, [a]) \to (bcc\varepsilon, q_0, [a, a]) \to (bcc\varepsilon, q_3, [a, a]) \to$$
$$(cc\varepsilon, q_3, [a, a]) \to (cc\varepsilon, q_4, [a, a]) \to (c\varepsilon, q_4, [a]) \to (\varepsilon, q_4, [\,])$$

   Primero se han apilado las a-s. Al aparecer b, había que elegir entre apostar por comparar el número de a-s con el número de b-s o apostar por comparar el número de a-s con el número de c-s. Se ha apostado por comparar el número de a-s con el número de c-s y se ha acertado. Se han leído las b-s sin hacer nada en la pila y luego se ha desapilado una a

por cada c que se ha leído. El AP respode que "Sí" porque se ha acabado en un estado de doble círculo y tras leer la última c y desapilar la a de la pila, la pila ha quedado vacía. Por tanto, se cumplen las condiciones necesarias para poder afirmar que $aabcc\varepsilon$ pertenece a G_2.

2. $$(aabcc\varepsilon, q_0, [\,]) \to (abcc\varepsilon, q_0, [a]) \to (bcc\varepsilon, q_0, [a, a]) \to (bcc\varepsilon, q_1, [a, a]) \to (cc\varepsilon, q_1, [a]) \to (cc\varepsilon, q_2, [a]) \to (c\varepsilon, q_2, [a]) \to (\varepsilon, q_2, [a])$$

   Como en el caso anterior, primero se han apilado las a-s. Al aparecer b, había que elegir entre apostar por comparar el número de a-s con el número de b-s o apostar por comparar el número de a-s con el número de c-s. En esta ocasión, se ha apostado por comparar el número de a-s con el número de b-s y no se ha acertado. Se ha desapilado una a cuando se ha leído la primera b. Luego, se ha pasado al estado q_2 sin leer nada desde el dispositivo de entrada. En el estado q_2, se han leído las dos c-s sin modificar la pila. Cuando se ha terminado la palabra, el AP estaba en un estado con doble círculo pero la pila no estaba vacía. Por tanto, al terminar de leer toda la palabra no se cumplen las condiciones necesarias para poder asegurar que $aabcc\varepsilon$ pertenece a G_2.

3. $$(aabcc\varepsilon, q_0, [\,]) \to (abcc\varepsilon, q_0, [a]) \to (abcc\varepsilon, q_3, [a]) \to (abcc\varepsilon, q_4, [a])$$

   En este caso, se ha apilado una a en la pila y, en vez de continuar en q_0 y apilar la segunda a, se ha optado por pasar al estado q_3 sin leer nada desde el dispositivo de entrada. La decisión ha sido desacertada porque se ha pasado al estado q_3 demasiado rápido. Como en el estado q_3 no se admiten a-s, la única posibilidad es pasar a q_4. En q_4 no se admiten a-s y el autómata se ha bloqueado, es decir, el proceso ha terminado pero no se cumplen las condiciones necesarias para poder asegurar que $aabcc\varepsilon$ pertenece a G_2.

Hay más casos en los que el recorrido acaba mal, pero como en un caso acaba bien, sabemos que la palabra $aabcc\varepsilon$ pertenece al lenguage G_2.

Si consideramos la palabra $aaab\varepsilon$, todas las computaciones que parten desde la configuración $(q_0, aaab\varepsilon, [\,])$ terminan mal y, consecuentemente, se puede asegurar que $aaab\varepsilon$ no pertenece a G_2.

Puesto que este autómata con pila es no determinista, dada una palabra hay más de una manera de moverse por los estados.

8.4.2.3 Ejemplo: Lenguaje formado por las palabras de la forma $v \cdot v^R$ con $v \in \mathbb{A}^*$

Vamos a considerar ahora un tercer lenguaje definido sobre el alfabeto $\mathbb{A} = \{a, b, c\}$ para el cual no se puede construir un autómata finito:

$$G_3 = \{w \mid w \in \mathbb{A}^* \wedge \exists v(v \in \mathbb{A}^* \wedge w = v \cdot v^R)\}$$

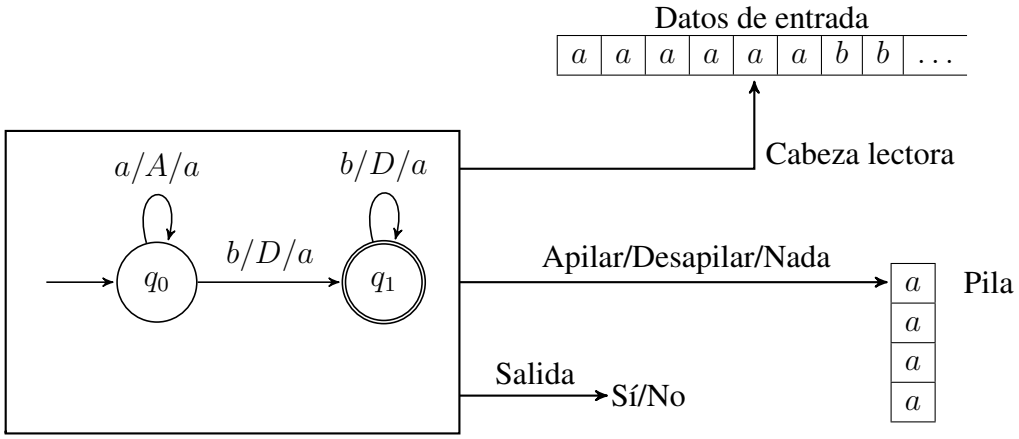


Figura 8.4.1. Autómata de pila —determinista— para palabras de la forma $(a\varepsilon)^k \cdot (b\varepsilon)^k$ con $k \in \mathbb{N}$. Ejemplo del apartado 8.4.2.1 de la página 87.

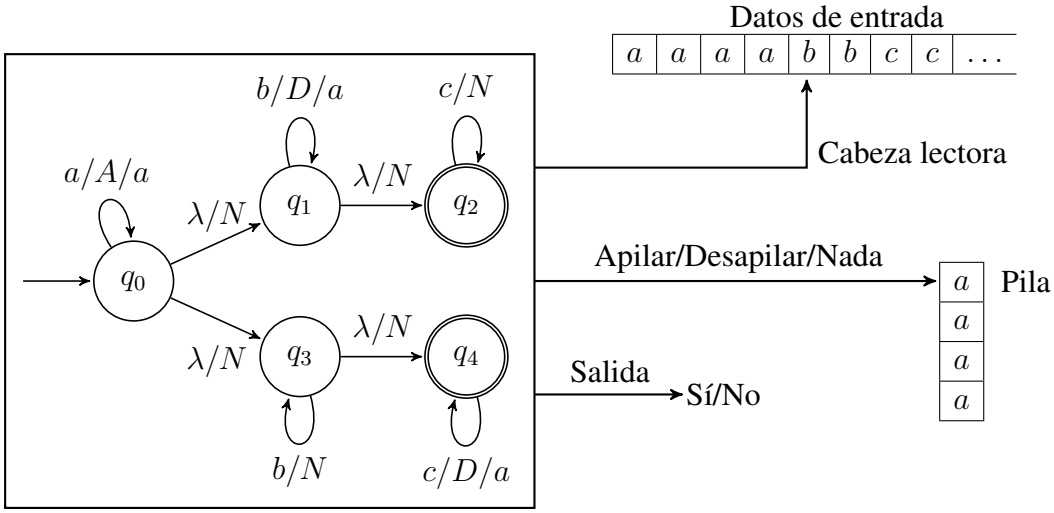


Figura 8.4.2. Autómata de pila —no determinista— para palabras de la forma $(a\varepsilon)^i \cdot (b\varepsilon)^j \cdot (c\varepsilon)^k$ con $i, j, k \in \mathbb{N}$ e $i = j$ o $i = k$. Ejemplo del apartado 8.4.2.2 de la página 89.

G_3 está formado por las palabras en las que la segunda mitad es la inversa de la primera mitad. G_3 contiene palabras como ε, $\underline{aa}aa\varepsilon$, $\underline{abc}cba\varepsilon$, $\underline{ab}ba\varepsilon$ y $\underline{ccbaba}ababcc\varepsilon$. Pero no contiene palabras como $aaca\varepsilon$, $abcabc\varepsilon$ y $aab\varepsilon$.

En la figura 8.4.3 de la página 98, se muestra un autómata no determinista con pila que sirve para reconocer el lenguaje G_3. Ese autómata parte con la pila vacía y va apilando los símbolos en la pila hasta llegar a la mitad de la palabra. En este autómata con pila la idea para decidir si una palabra tiene la forma $v \cdot v^R$ es ir guardando los símbolos de v en la pila y después ir quitándolos de la pila según se va leyendo v^R. Es importante darse cuenta de que los símbolos que forman v se guardan en la pila de tal forma que el último componente de v quede en la cima. Ese último componente de v ha de ser igual al primer componente de v^R.

Puesto que ese autómata con pila es no determinista, dada una palabra hay más de una manera de moverse por los estados. Una de esas maneras de moverse por los estados según se va leyendo la palabra de entrada acaba bien si al parar la máquina se cumple que se ha terminado en un estado final, se ha leído toda la palabra, se han podido realizar todas las operaciones de quitar (o desapilar) y la pila está vacía. Para poder realizar la operación de quitar un elemento de la pila, la pila no puede ser vacía y el primer símbolo de la palabra actual y el símbolo de la cima de la pila han de coincidir. Es suficiente con que en una de esas posibles maneras de moverse por los estados se acabe bien para poder asegurar que la palabra pertenece a G_3. En las transiciones, N indica que no hay que hacer nada en la pila, A indica que hay que poner en la cima de la pila (apilar) el símbolo que se acaba de leer y D indica que hay que quitar (desapilar) el símbolo de la cima de la pila, si coincide con el primer símbolo del trozo de palabra que queda por leer.

Por ejemplo, para la palabra $acca\varepsilon$ hay varias posibilidades:

1. $$(acca\varepsilon, q_0, [\,]) \to (cca\varepsilon, q_0, [a]) \to (ca\varepsilon, q_0, [c, a]) \to (ca\varepsilon, q_1, [c, a]) \to (a\varepsilon, q_1, [a]) \to (\varepsilon, q_1, [\,])$$

   Primero se han apilado las a y c. En cada momento había que elegir entre apostar por quedarse en q_0 o apostar por pasar a q_1. Se ha pasado al estado q_1 en el momento apropiado. Una vez en q_1, se ha podido desapilar un elemento de la pila por cada elemento leído desde la entrada porque coincidían. El AP respode que "Sí" porque se ha acabado en un estado de doble círculo y tras leer la última a y desapilar la a de la pila, la pila ha quedado vacía. Por tanto, se cumplen las condiciones necesarias para poder afirmar que $acca\varepsilon$ pertenece a G_3.

2. $$(acca\varepsilon, q_0, [\,]) \to (cca\varepsilon, q_0, [a]) \to (cca\varepsilon, q_1, [a])$$

   En este caso ha ido mal porque se ha pasado al estado q_1 demasiado rápido y a continuación se tiene una c en la palabra y una a en la pila. Como solo se puede quitar un elemento de la pila si el primer símbolo de la palabra es igual, el autómata se bloquea. Por

tanto, en esta computación no se cumplen las condiciones necesarias para poder asegurar que la palabra pertenece al lenguaje G_3.

3.

$$(accaε, q_0, [\,]) \to (ccaε, q_0, [a]) \to (caε, q_0, [c, a]) \to (aε, q_0, [c, c, a]) \to (aε, q_1, [c, c, a])$$

En este caso ha ido mal porque se ha pasado al estado q_1 demasiado tarde y a continuación se tiene una a en la palabra y una c en la pila. Como solo se puede quitar un elemento de la pila si el primer símbolo de la palabra es igual, el autómata se bloquea. No se cumplen las condiciones necesarias para poder asegurar que la palabra pertenece al lenguaje G_3.

4. ...

Hay más casos en los que el recorrido acaba mal, pero como ya hemos visto que en un caso acaba bien, sabemos que la palabra $accaε$ pertenece al lenguage G_3.

Si cogemos la palabra $bbbaε$, todas las maneras de moverse por los estados terminan mal y se podrá asegurar que $bbbaε$ no pertenece al lenguaje G_3.

8.4.3 Utilidad de los autómatas con pila

Una de las utilidades tanto de los lenguajes regulares como de los independientes del contexto, y por consiguiente de los autómatas finitos y de los autómatas con pila, es en el área de la teoría de la computación, ya que pueden ser utilizados para estudiar algunos aspectos de la computabilidad. Pero también son útiles en campos más prácticos. En concreto, los AF se utilizan en analizadores léxicos —que han de analizar si un nombre de variable o de función, etc. es adecuado— mientras que los AP se utilizan en analizadores sintácticos. Por tanto, sirven para construir compiladores, analizar estructuras y definir la sintáxis de los lenguajes de programación. Por ejemplo, los AP se pueden utilizar para averiguar si en una expresión todos los paréntesis que se abren, se cierran luego adecuadamente. En este ejemplo concreto, la idea sería apilar los paréntesis que se abren y cada vez que se cierre un paréntesis, desapilar un paréntesis que está almacenado en la pila.

El tipo de gramática que corresponde a los AP es el de las gramáticas libres de contexto o incontextuales. También estas gramáticas son cuádruplas de la forma (N, T, P, S). En estas gramáticas las reglas de producción son de la forma $X \to \Omega$, donde Ω es una cadena cualquiera formada por símbolos no terminales de N y símbolos terminales de T. Por tanto, $\Omega \in (N \cup T)^*$. Las reglas de producción se aplican sin tener en cuenta el contexto, es decir, lo que rodea.

A continuación se muestran algunos ejemplos de gramáticas incontextuales:

- La siguiente gramática genera el lenguaje G_1 del apartado 8.4.2.1 de la página 87:

1. $S \to H$
2. $S \to VH$
3. $H \to \varepsilon$
4. $V \to ab$
5. $V \to aVb$

El símbolo inicial es S. El conjunto de símbolos no terminales es $\{S, V, H\}$ y el conjunto de símbolos terminales es $\{a, b\}$.

- La siguiente gramática libre de contexto genera el lenguaje formado por todas las palabras palíndromas definibles sobre el alfabeto $\{a, b\}$:

1. $S \to H$
2. $S \to VH$
3. $V \to aVa$
4. $V \to aa$
5. $V \to bVb$
6. $V \to bb$
7. $H \to \varepsilon$

El símbolo inicial es S. El conjunto de símbolos no terminales es $\{S, H, V\}$ y el conjunto de símbolos terminales es $\{a, b\}$.

- La siguiente gramática libre de contexto genera el lenguaje formado por todas las palabras que representan expresiones aritméticas definibles sobre el alfabeto $\{x, y, z, +, -, *, /, (,)\}$:

1. $S \to VH$
2. $V \to x$
3. $V \to y$
4. $V \to z$
5. $V \to V + V$
6. $V \to V - V$
7. $V \to V * V$
8. $V \to V/V$
9. $V \to (V)$
10. $H \to \varepsilon$

El símbolo inicial es S. El conjunto de símbolos no terminales es $\{S, V, H\}$ y el conjunto de símbolos terminales es $\{x, y, z, +, -, *, /, (,)\}$.

Esa gramática produce expresiones como la siguiente:

$$(x + y) * x - z * y/(x + x)\varepsilon$$

8.4.4 Lenguajes no reconocibles mediante autómatas con pila

Tal como se indicado el apartado 8.4.2 de la página 85, en los autómatas finitos con pila, solo se pueden realizar dos operaciones sobre la pila: Poner un elemento en la cima y quitar el elemento de la cima. Por tanto, no se puede acceder directamente a los elementos que no están en la cima de la pila. La única manera de acceder a un elemento x que no está en la cima es ir

quitando elementos hasta que x quede en la cima. Esta limitación en el manejo de pilas hace que la capacidad de cálculo de los autómatas con pilas sea limitada.

A continuación se muestran ejemplos de lenguajes que no pueden ser definidos por ningún autómata con pila y que, por tanto, no son lenguajes incontextuales:

- $F_1 = \{w \mid w \in \mathbb{A}^* \wedge \exists k(k \in \mathbb{N} \wedge w = (a\varepsilon)^k \cdot (b\varepsilon)^k \cdot (c\varepsilon)^k)\}$. Es decir, F_1 contiene palabras como ε, $abc\varepsilon$, $aabbcc\varepsilon$, $aaabbbccc\varepsilon$, etc. Pero no contiene palabras como $abbccc\varepsilon$, $aab\varepsilon$, $bcaabbba\varepsilon$, etc. Aunque este lenguaje es parecido a G_2, en G_2 había que comparar el número de a-s con el número de b-s o con el número de c-s pero no con los dos. En F_1 hay que comparar el número de a-s con el número de b-s y con el número de c-s, con los dos. Por ello, si guardamos las a-s en la pila y luego las borramos al ir leyendo las b-s, cuando lleguemos a las c-s ya no se sabe cuántas a-s había porque se han borrado de la pila.

- $F_2 = \{w \mid w \in \mathbb{A}^* \wedge \exists i, j, k(i, j, k \in \mathbb{N} \wedge (0 \leq i \leq j \leq k) \wedge w = (a\varepsilon)^i \cdot (b\varepsilon)^j \cdot (c\varepsilon)^k)\}$. El lenguaje F_2 contiene palabras como ε, $abc\varepsilon$, $aabbcc\varepsilon$, $bbccc\varepsilon$ y $abbccc\varepsilon$. Pero no contiene palabras como $aaabbccc\varepsilon$, $aab\varepsilon$ y $bbacccb\varepsilon$. El problema de este lenguaje es similar al comentado en F_1.

- $F_3 = \{w \in A^* \mid \exists v(v \in A^* \wedge w = v \cdot v)\}$. F_3 contiene palabras como ε, $\underline{aa}aa\varepsilon$, $\underline{abc}abc\varepsilon$, $\underline{ab}ab\varepsilon$ y $\underline{ccbaba}ccbaba\varepsilon$. Pero no contiene palabras como $aacaa\varepsilon$, $abccba\varepsilon$ y $aab\varepsilon$. Debido al orden en el que se guardan los símbolos en la pila y debido a que solo se puede tener acceso al elemento que está en la cima, no hay manera de saber si la palabra consta de dos trozos repetidos utilizando un autómata con pila.

Los lenguajes de estos tres ejemplos son **lenguajes dependientes del contexto** y los autómatas correspondientes son los **autómatas linealmente acotados**.

A modo de ejemplo, la gramática dependiente del cotexto que corresponde al lenguaje F_1 es la siguiente:

1. $S \to \varepsilon$	4. $X \to aXHc$
2. $S \to X\varepsilon$	5. $cH \to Hc$
3. $X \to abc$	6. $bH \to bb$

El símbolo inicial es S. El conjunto de símbolos no terminales es $\{S, X, H\}$ y el conjunto de símbolos terminales es $\{a, b, c\}$. Observando las reglas de producción 5 y 6, se puede apreciar que en el lado izquierdo de la regla no hay solo un símbolo. En concreto, en las reglas 5 y 6, podemos decir que el símbolo H muestra un comportamiento distinto dependiendo de lo que le rodea a H (en este caso, dependiendo del símbolo no terminal que le precede). Por tanto, dependiendo del contexto de H, hay que aplicar una regla u otra. Puesto que el entorno o el contexto influye a la hora de elegir qué regla aplicar, estamos ante una gramática dependiente

del contexto. A continuación, se muestra una posible manera de producir la palabra $aaabbbcccε$ perteneciente al lenguaje F_1:

$$\underline{S} \to^2 \underline{X}\varepsilon \to^4 a\underline{X}Hc\varepsilon \to^4 aa\underline{X}HcHc\varepsilon \to^3 aaab\underline{cH}cHc\varepsilon \to^5 aaab\underline{bH}ccHc\varepsilon \to^6 aaabbc\underline{cH}c\varepsilon \to^5 aaabbc\underline{bH}cc\varepsilon \to^5 aaabb\underline{bH}ccc\varepsilon \to^6 aaabbbccc\varepsilon$$

En cada paso, se ha subrayado la subcadena a la que se le va a aplicar una regla de producción.

Los autómatas más generales son las **máquinas de Turing**. Los lenguajes asociados a las máquinas de Turing son los **lenguajes recursivamente enumerables**. Algunos de los lenguajes recursivamente enumerables son **decidibles**. Estos lenguajes decidibles reciben también el nombre de **lenguajes recursivos**. Aquellos lenguajes que son recursivamente enumerables pero no son recursivos, son conocidos como **semi-decidibles o parcialmente decidibles**. Los lenguajes decidibles son aquellos para los cuales se tiene una máquina de Turing total (siempre responde), mientras que para los lenguajes recursivamente enumerables que no son decidibles (por tanto, son semi-decidibles) se tiene una máquina de Turing parcial (a veces no responde).

Pero, tal como se mostrará en el siguiente tema, hay lenguajes que no son recursivamente enumerables y, por tanto, las máquinas de Turing no sirven para esos lenguajes. De hecho, para esos lenguajes que quedan fuera del conjunto de lenguajes recursivamente enumerables, no es posible definir ningún tipo de autómata.

En la figura 8.4.4 de la página 99, se muestra la relación entre los distintos tipos de lenguajes mencionados hasta ahora. Los lenguajes regulares forman un subconjunto de los lenguajes incontextuales deterministas y estos, a su vez, forman un subconjunto de los lenguajes incontextuales. Los lenguajes incontextuales forman un subconjunto de los lenguajes contextuales y estos, por su parte, forman un subconjunto de los lenguajes recursivos (o decidibles). Finalmente, los leguajes recursivos constituyen un subconjunto del conjunto formado por los lenguajes recursivamente enumerables o lenguajes semidecidibles o lenguajes reconocibles. Los lenguajes regulares son los definibles mediante autómatas finitos, los lenguajes incontextuales deterministas son los definibles mediante autómatas deterministas con pila, los lenguajes incontextuales son los definibles mediante autómatas no deterministas con pila. Los lenguajes contextuales requieren de los autómatas linealmente acotados. Los lenguajes recursivos o decidibles requieren de máquinas de Turing totales mientras que para entrar en el conjunto de lenguajes recursivamente enumerables —o semidecidibles o reconocibles—, se necesita tener una máquina de Turing que puede ser total o parcial. Nótese que un autómata finito es un autómata con pila en el que no se utiliza la pila, es decir, podemos considerar que un AF es un AP en el que todas las transiciones son de la forma λ/N o de la forma α/N, donde $\alpha \in \mathbb{A}$ y N significa no hacer nada en la pila (ni apilar ni desapilar). De la misma forma, los AP son casos particulares de los autómatas linealmente acotados y estos, a su vez, son casos particulares de las máquinas de Turing totales.

Existen lenguajes que quedan fuera de esa clasificación. Son lenguajes que ni siquiera son semidideciblesdecidibles o reconocibles. No es posible diseñar ningún tipo de autómata para esos lenguajes. Son los lenguajes *no reconocibles*.

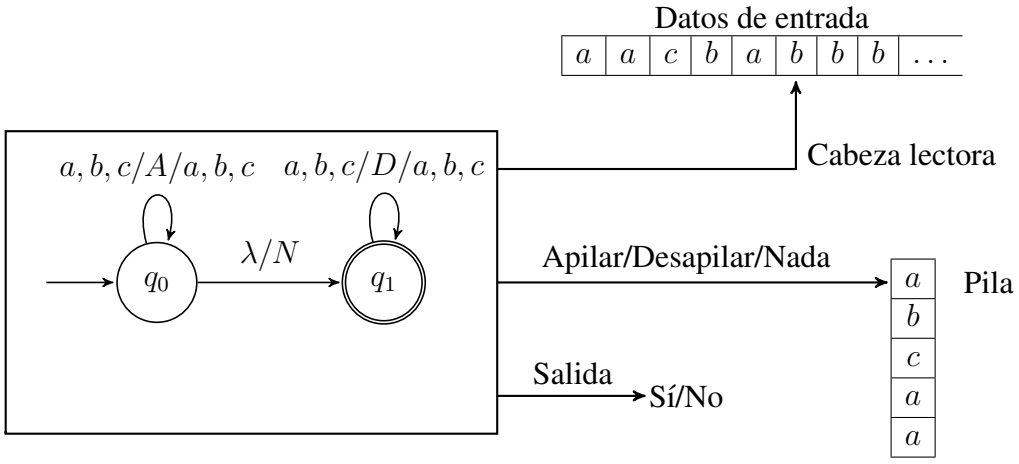


Figura 8.4.3. Autómata de pila —no determinista— para palabras de la forma $v \cdot v^R$ con $v \in \mathbb{A}^*$. Ejemplo del apartado 8.4.2.3 de la página 90.

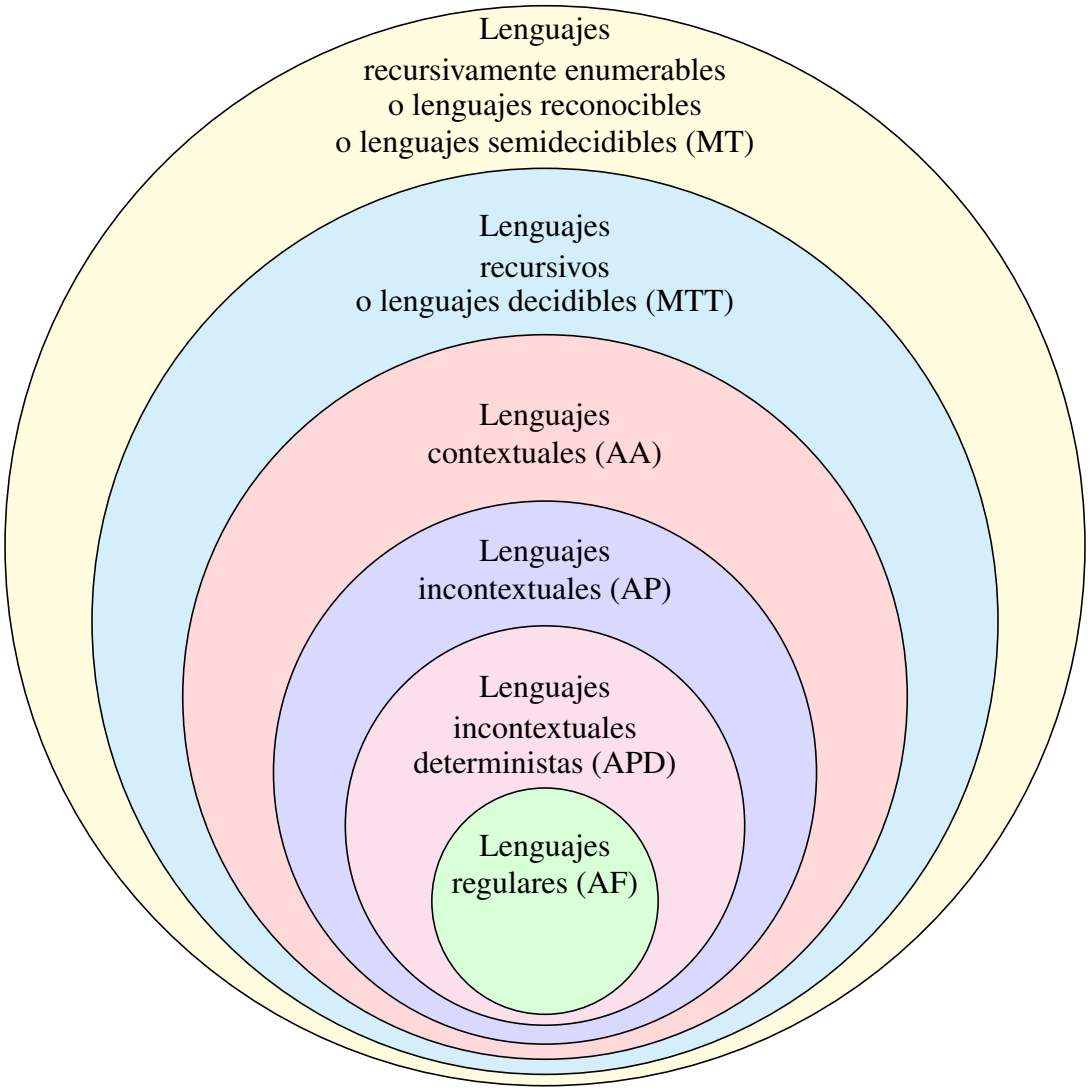


Figura 8.4.4. Clasificación de lenguajes definidos sobre un alfabeto $\mathbb{A}$ dependiendo del tipo de autómata que requieren: máquina de Turing (MT), máquina de Turing total (MTT), autómata linealmente acotado (AA), autómata con pila (AP), autómata determinista con pila (APD) o autómata finito (AF). Existen lenguajes de $\mathbb{2}^{\mathbb{A}^*}$ que quedan fuera de estos conjuntos porque ningún tipo de autómata sirve para ellos: son los lenguajes no reconocibles.