

**Versión del: July 12, 2018**

Olivia Gutú

# Primer curso en teoría de autómatas y lenguajes formales

2.<sup>a</sup> edición

---

## Contenido

---

<b>1</b>	<b>Alfabetos, cadenas y lenguajes . . . . .</b>	1
1.1	Notación básica . . . . .	2
1.2	Alfabetos y cadenas . . . . .	2
1.3	Lenguajes . . . . .	5
<b>2</b>	<b>Autómatas finitos deterministas . . . . .</b>	9
2.1	Definición de autómatas finitos deterministas . . . . .	10
2.2	Lenguaje de un autómata finito determinista . . . . .	14
2.3	Construcción del autómata de la intersección . . . . .	19
<b>3</b>	<b>Autómatas no-deterministas . . . . .</b>	23
3.1	Autómatas finito no-deterministas . . . . .	24
3.2	Autómatas con transiciones instantáneas . . . . .	29
3.3	Equivalencia de los autómatas finitos . . . . .	34
<b>4</b>	<b>Lenguajes regulares . . . . .</b>	43
4.1	Expresiones regulares . . . . .	43
4.2	De autómatas finitos a expresiones regulares . . . . .	48
4.3	De expresiones regulares a autómatas finitos . . . . .	51
4.4	Propiedades de los lenguajes regulares . . . . .	54
<b>5</b>	<b>Autómatas a pila . . . . .</b>	59
5.1	Definición de autómata a pila . . . . .	60
5.2	Aceptación por pila vacía y por estado de aceptación . . . . .	67
5.3	Equivalencia entre aceptación por estados y por pila vacía . . . . .	71

5.4	Autómatas a pila deterministas y lenguajes regulares.....	75
<b>6</b>	<b>Gramáticas libres de contexto .....</b>	<b>79</b>
6.1	Definición de gramática libre de contexto .....	80
6.2	Derivaciones y árboles de derivación .....	86
6.3	Gramáticas ambiguas .....	92
<b>7</b>	<b>Lenguajes libres de contexto .....</b>	<b>99</b>
7.1	De gramáticas a autómatas a pila .....	99
7.2	De autómatas a pila a gramáticas .....	103
7.3	Lema de bombeo para los lenguajes libres de contexto.....	113
7.4	Propiedades de los lenguajes libres de contexto .....	117
7.5	Algoritmo de Cocke-Younger-Kasami .....	123
<b>A</b>	<b>Forma normal de Chomsky .....</b>	<b>127</b>
<b>B</b>	<b>Minimización de un autómata finito determinista .....</b>	<b>137</b>
B.1	Estados equivalentes .....	138
B.2	Construcción del autómata mínimo .....	143
	<b>Referencias .....</b>	<b>147</b>

# CAPÍTULO 1

---

## Alfabetos, cadenas y lenguajes

---

En este capítulo se presenta la terminología formal y elemental que se emplea a lo largo del texto comenzando en la primera sección con la notación básica. En la segunda sección se introduce la noción abstracta de *cadena* como una una yuxtaposición de símbolos dentro de un conjunto llamado *alfabeto*, la cual en situaciones específicas puede representar muchas cosas: un texto, un programa, una secuencia finita de señales, una sucesión finita de movimientos de una ficha dentro de un juego, etcétera. En la tercera sección se establece el concepto de *lenguaje*, algunas nociones relacionadas y propiedades esenciales. Aunque un lenguaje teóricamente es simplemente un conjunto de cadenas, en la práctica un problema concreto como «decidir si una palabra está en un texto» o «decidir si un programa tiene la sintaxis correcta» se puede caracterizar mediante un lenguaje, eso se verá en capítulos posteriores.

Se asume que el lector está familiarizado con las nociones de conjunto, par ordenado, función, relación, así como con la lógica de primer orden y el razonamiento matemático a nivel elemental. Sin embargo, dentro de la segunda sección se incluye un breve recordatorio sobre el principio de inducción matemática, pues es el razonamiento en que se basan prácticamente todas las demostraciones formales del texto.

## 1.1 Notación básica

Sean  $A$  y  $B$  dos conjuntos cualesquiera. La notación estándar de conjuntos es la que se usa de aquí en adelante:

unión de conjuntos	$A \cup B = \{x : x \in A \text{ ó } x \in B\}$
intersección de conjuntos	$A \cap B = \{x : x \in A \text{ y } x \in B\}$
diferencia de conjuntos	$A \setminus B = \{x : x \in A \text{ y } x \notin B\}$
complemento de un conjunto	$A^c = \{x : x \notin A\}$
espacio producto	$A \times B = \{(a, b) : a \in A \text{ y } b \in B\}$

Sean  $m$  y  $n$  números en  $\{0, 1, 2, 3, \dots\}$ . A lo largo del texto, se escribe  $n \geq m$  si  $n \in \{m, m+1, m+2, \dots\}$  y  $n > m$  si  $n \in \{m+1, m+2, \dots\}$ .

Se denota por  $f : X \rightarrow Y$  a una función  $f$  con dominio en  $X$  e imagen contenida en  $Y$ . El símbolo  $f(x)$  —se lee  $f$  evaluada en  $x$ — es un elemento de  $Y$ ; si se escribe  $f(x) = y$  significa que  $f$  le asigna a  $x \in X$  el valor de  $y \in Y$ .

## 1.2 Alfabetos y cadenas

Un *alfabeto* es un conjunto finito no vacío. Por convención se denota a los alfabetos con la letra  $\Sigma$ . A los elementos de un alfabeto se les llama *símbolos*.

**Ejemplo 1.** El conjunto  $\{0, 1\}$  es un alfabeto, al cual se le conoce como *alfabeto binario*. El conjunto de símbolos de texto plano (sin formato, universalmente legible) es un alfabeto. El conjunto de números naturales no es un alfabeto.

Dado un alfabeto  $\Sigma$ , una *cadena* de  $\Sigma$  es la yuxtaposición finita de símbolos de  $\Sigma$ . La *cadena vacía* de  $\Sigma$  es la yuxtaposición de 0 símbolos de  $\Sigma$ , se denota siempre por  $\varepsilon$ , independientemente de cual sea el alfabeto. Por supuesto, se entiende que  $\varepsilon$  no es un símbolo de  $\Sigma$ . Sea  $n$  un entero no negativo y  $a$  un símbolo del alfabeto. A veces se usa  $a^n$  para denotar a la yuxtaposición de  $n$  símbolos iguales a  $a$ , es decir:

$$a^n = \underbrace{aa \cdots a}_{n \text{ veces}}.$$

Por convención  $a^0$  se define como la cadena vacía. La *longitud* de una cadena  $w$  es el número de posiciones ocupadas por los símbolos que constituyen la cadena.

Se denota a la longitud de  $w$ , por  $|w|$ . La cadena vacía de cualquier alfabeto tiene por definición longitud cero, *i. e.*  $|\varepsilon| = 0$ .

**Ejemplo 2.** La secuencia **holá** es una cadena de longitud 4. La secuencia **011011** es una cadena del alfabeto binario de longitud 6. La secuencia infinita **010101 ···** no es una cadena del alfabeto binario, tampoco lo es la secuencia **00210**.

Dado un alfabeto, se pueden formar infinitas cadenas con los elementos de ese alfabeto. El conjunto constituido por todas las posibles cadenas formadas con elementos de un alfabeto dado, será referido muchas veces a lo largo del texto, así que se le dará un nombre especial:

**Definición.** Sea  $\Sigma$  un alfabeto. La *clausura* de  $\Sigma$  —se denota por  $\Sigma^*$ — es el conjunto de todas las cadenas tales que cada uno de su símbolos está  $\Sigma$ , incluyendo a la cadena vacía.

**Ejemplo 3.**  $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 010, 011, 101, \dots\}$ .

Sea  $\Sigma$  un alfabeto, se define la *concatenación* de dos cadenas  $x$  e  $y$  de  $\Sigma^*$  mediante la operación binaria  $\cdot : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  dada de la siguiente manera: si  $x \neq \varepsilon$  e  $y \neq \varepsilon$  entonces  $x \cdot y$  se define como la yuxtaposición de la cadena  $x$  con la cadena  $y$ , en otras palabras,

$$x \cdot y = xy;$$

además, para cualquier cadena  $x \in \Sigma^*$  se define  $x \cdot \varepsilon = \varepsilon \cdot x = x$ . Obviamente, para cualesquiera tres cadenas  $x, y$  y  $z$  de  $\Sigma^*$ ,  $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ , luego la expresión  $x \cdot y \cdot z$  está bien definida. Por tanto, la concatenación finita de cadenas también está bien definida.

Si  $w = x \cdot y$  es una cadena de un alfabeto, es fácil verificar que

$$|w| = |x| + |y|.$$

En el resto del documento, se abusa de la notación y se escribe simplemente  $xy$  en lugar de  $x \cdot y$ . Así pues, se debe entender que  $x\varepsilon$  es la cadena  $x$ , no debe de confundirse con la simple yuxtaposición de  $x$  con  $\varepsilon$ , puesto que visto de esta forma,  $x\varepsilon$  no estaría en la clausura del alfabeto. Si  $w$  es una cadena cualquiera, por convención  $w^0 = \varepsilon$  y como es natural, se entiende que:

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

Durante todo el texto se darán argumentos para evidenciar y justificar predicados sobre números naturales, es decir, enunciados de la forma:

$$\text{Para todo } n > 0 \text{ se satisface } P(n). \quad (1.1)$$

Aquí  $P(n)$  representa a una afirmación que depende de  $n$ . Ejemplos de predicados sobre números naturales son los siguientes «para todo  $n > 0$ ,  $\varepsilon^n = \varepsilon$ », «para todo  $n > 0$  se cumple que  $|x^n| \geq |x^{n-1}|$ », etcétera. Un método *ad hoc* para verificar predicados sobre números naturales es el siguiente:

**Principio de inducción matemática.** Para probar (1.1) se procede como sigue:

- 1.<sup>o</sup> se verifica  $P(1)$ ,
- 2.<sup>o</sup> se demuestra que  $P(n)$  implica  $P(n + 1)$ .

A la premisa del 2.<sup>o</sup> paso se le llama *hipótesis de inducción*.

Si  $P(1)$  es verdadera y se comprueba que  $P(n)$  implica  $P(n + 1)$  entonces se deduce que  $P(2)$  es verdadera, y por tanto también lo es  $P(3)$  y así sucesivamente. Por supuesto, se puede comenzar el proceso inductivo con  $n = 0$ ; en este caso la prueba es válida para predicados sobre el conjunto  $\{0, 1, 2, 3, \dots\}$ . En general, se puede ajustar el principio de inducción matemática para demostrar que  $P(n)$  es cierta para todo número natural  $n \geq m$ , pues simplemente  $P(m)$  es probado primero y después el 2.<sup>o</sup> paso.

### **Lista de ejercicios de la sección 1.2**

*Ejercicio 1.* Pruebe que si  $w = x^n$  para alguna cadena  $x$  y un  $n > 0$ , entonces  $|w| = n|x|$ .

*Ejercicio 2.* Sea  $\Sigma$  un alfabeto y  $w$  una cadena en  $\Sigma^*$ . Se dice que  $x$  es una *subcadena* de  $w$  si existen cadenas  $y$  y  $z$  en  $\Sigma^*$  tales que  $w = yxz$ . Verifique  $|x| \leq |w|$ .

*Ejercicio 3.* Sea  $\Sigma$  un alfabeto con  $m$  elementos. ¿Cuántas cadenas de longitud  $n$  hay en  $\Sigma^*$ ?

## 1.3 Lenguajes

Sea  $\Sigma$  un alfabeto. Se dice que  $L$  es un *lenguaje* de un alfabeto  $\Sigma$  si es un subconjunto de  $\Sigma^*$ .

**Ejemplo 4.** El conjunto de cadenas binarias con un número par de ceros y un número impar de unos es un lenguaje del alfabeto binario. Los conjuntos  $\{\varepsilon\}$  y el conjunto vacío  $\emptyset$  son obviamente lenguajes de cualquier alfabeto.

Sean  $L$  y  $M$  lenguajes con alfabeto  $\Sigma$ . Como  $L$  y  $M$  son conjuntos, podemos hablar de *unión*, *intersección* y *diferencia* entre  $L$  y  $M$ , incluso también de *complemento* del lenguaje  $L$ . Sin embargo, para el caso particular de lenguajes se puede definir una nueva operación a partir de la noción de concatenación de dos cadenas:

**Definición.** La *concatenación* de dos lenguajes  $L$  y  $M$  de un mismo alfabeto es el conjunto:

$$LM = \{xy : x \in L \text{ y } y \in M\}.$$

Note que, por definición de concatenación de cadenas, el conjunto  $LM$  es también un lenguaje con alfabeto  $\Sigma$ .

**Ejemplo 5.** Si  $L = \{0, 11\}$  y  $M = \{0001, 111\}$  es fácil verificar que:

$$LM = \{00001, 0111, 110001, 11111\}.$$

Podemos «hacer las cuentas» con una tabla:

L		
0	0 · 0001	0 · 111
11	11 · 0001	11 · 111
	0001	111 M

A lo largo del texto, reiteradamente se habla de «probar que dos lenguajes son iguales». Un lenguaje, desde el punto de vista puramente matemático, es simplemente un conjunto numerable. Si  $L$  y  $M$  son dos lenguajes y se quiere verificar que  $L = M$ , basta demostrar que  $L \subset M$  y  $M \subset L$ , es decir, se han de demostrar las siguientes afirmaciones:

- si  $x \in L$  entonces  $x \in M$ , y
- si  $x \in M$  entonces  $x \in L$ .

Un ejemplo —como muchos otros que aparecen después— es la prueba de la siguiente propiedad.

**Propiedad 1.** Sean  $L$ ,  $M$  y  $N$  lenguajes con un mismo alfabeto. Se cumple que:

$$(LM)N = L(MN).$$

*Prueba.* Sea  $w \in (LM)N$ , entonces  $w = w_1z$ , donde  $w_1 \in LM$  y  $z \in N$ . Además,  $w_1 = xy$  con  $x \in L$  e  $y \in M$ . Por lo que  $w = xw_2$ , donde  $w_2 = yz \in MN$ , luego  $w \in L(MN)$ . Con esto se prueba que  $(LM)N \subset L(MN)$ . Para ver que  $L(MN) \subset (LM)N$ , se razona de la misma forma.  $\square$

Si  $L$ ,  $M$  y  $N$  son lenguajes con un mismo alfabeto, por la propiedad 1, la expresión  $LMN$  está bien definida, y por tanto, la concatenación finita de lenguajes con un mismo alfabeto está también bien definida. Sea  $L$  un lenguaje con alfabeto  $\Sigma$ . Si  $n$  es un número natural, se escribe  $L^n$  para denotar la concatenación de  $L$  con él mismo  $n$  veces. Es decir:

$$L^n = \underbrace{LL\cdots L}_{n \text{ veces}}.$$

Además, por convención  $L^0$  denota al conjunto  $\{\varepsilon\}$ . El lenguaje de todos estos lenguajes, es un concepto relevante, como veremos más adelante:

**Definición.** La *clausura de Kleene* —en lo que sigue, simplemente *clausura*— de un lenguaje  $L$  es el conjunto:

$$L^* = \bigcup_{n=0}^{\infty} L^n.$$

Naturalmente, la clausura de un lenguaje con alfabeto  $\Sigma$  es también un lenguaje con alfabeto  $\Sigma$ .

**Ejemplo 6.**  $\{0, 11\}^* = \{\varepsilon, 0, 11, 00, 011, 110, 1111, 000, 0011, 0110, 1100, \dots\}$ .

**Ejemplo 7.** Sea  $L = \{\varepsilon, 0, 00, 000, 0000, \dots\}$ . Para todo  $n$  natural,  $L^n = L$  lo que implica que  $L^* = L$ .

**Ejemplo 8.** Ya que  $\emptyset^0 = \{\varepsilon\}$  y para todo natural  $n$  se cumple que  $\emptyset^n = \emptyset$  y por tanto  $\emptyset^* = \{\varepsilon\}$ .

### **Lista de ejercicios de la sección 1.3**

*Ejercicio 4.* Demuestre que para cualquier lenguaje  $L$ ,  $L\emptyset = \emptyset L = \emptyset$ .

*Ejercicio 5.* Sea  $L = \{\varepsilon, 0, 00, 000, 0000, \dots\}$  y  $M = \{\varepsilon, 1, 11, 111, 1111, \dots\}$ . ¿Quién es el conjunto  $(LM)^*$ ?

*Ejercicio 6.* Sean  $L$  y  $M_n$ ,  $n \geq 0$  lenguajes de un mismo alfabeto. Verifica que:

$$L \bigcup_{n=0}^{\infty} M_n = \bigcup_{n=0}^{\infty} LM_n.$$

*Ejercicio 7.* Sean  $L$ ,  $M$  y  $N$  lenguajes cualesquiera, todos con un mismo alfabeto. Pruebe o de un contraejemplo de cada una de las siguientes afirmaciones:

1.  $L(M \cap N) = LM \cap LN$ ;
2.  $L \cap (MN) = (L \cap M)(L \cap N)$ ;
3.  $LM = ML$ ;
4.  $LL^* = L^*L$ .
5. Para todo  $n$  natural,  $(LM)^n = L^n M^n$ .
6. Para todo  $n$  y  $m$  naturales,  $(L^n)^m = (L^m)^n$ .

*Ejercicio 8.* Verifique que si  $\varepsilon \notin L$  entonces  $L^*L = L^* \setminus \{\varepsilon\}$ .

*Ejercicio 9.* Demuestre que para todo lenguaje  $L$ ,  $(L^*)^* = L^*$ . Sugerencia: verifique primero que para todo  $n$  natural,  $(L^*)^n = L^*$ .

*Ejercicio 10.* Sean  $L$  y  $M$  dos lenguajes, demuestre que  $(L \cup M)^* = (L^*M^*)^*$ .

*Ejercicio 11.* Prueba el *lema de Arden*. Considere la ecuación entre lenguajes

$$X = XM \cup N \tag{1.2}$$

con  $X$  desconocida. Se tiene lo siguiente:

1.  $X_0 = NM^*$  es solución de la ecuación (1.2).
2. Si  $L$  es otra solución de (1.2) entonces  $X_0 \subset L$ . Esto es,  $X_0$  es la solución «más pequeña» de (1.2).
3. Si  $\varepsilon \in M$  entonces para cualquier lenguaje  $S$ ,  $X_S = (N \cup S)M^*$  es solución de (1.2). Sugerencia: note que si  $\varepsilon \in M$  entonces  $M^*M = M^*$ .
4. Si  $\varepsilon \notin M$  entonces  $X_0$  es la única solución de (1.2). Sugerencia: sea  $L$  otra solución de (1.2), pruebe que para todo  $n > 0$ ,  $L = LM^{n+1} \cup N \bigcup_{i=0}^n M^i$ ; pruebe que si  $w \in L$  con  $|w| = n$  entonces  $w$  no puede estar en  $LM^{n+1}$  y por tanto tiene que estar en  $N \bigcup_{i=0}^n M^i \subset X_0$ .



# CAPÍTULO 2

---

## Autómatas finitos deterministas

---

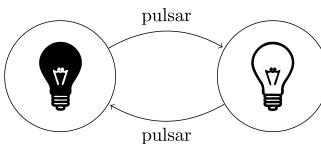
En este capítulo se estudia a los autómatas finitos deterministas (AFD), concepto que sirve de modelo para procesos donde existe un número finito de estados y de señales que indican el cambio de estado. La transición de un estado a otro se hace de forma determinista, lo que significa que dado un estado y una señal existe solo una posibilidad de estado de llegada. A cada sistema de estados y señales le corresponde un lenguaje de cadenas de señales asociado a un problema específico.

La definición de un AFD se establece en la primera sección junto con algunos ejemplos informales. Todo AFD tiene asociado un lenguaje, el cual a su vez puede estar ligado a un problema concreto. En la segunda sección se exponen dos maneras equivalentes de definir al lenguaje asociado a un AFD . En la tercera sección finalmente se establece un algoritmo que nos permite construir autómatas más complicados a partir de autómatas simples a través de la «intersección» de dos autómatas.

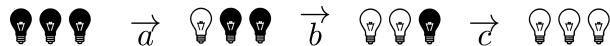
El estudio formal de los autómatas finitos deterministas se remite a [15], aunque la definición habitual que aparece en los libros de texto, *c. f.* [11] y [23] —que es la que se presenta aquí— se introdujo de manera independiente en [12], [17] y [18].

## 2.1 Definición de autómatas finitos deterministas

Un autómata finito determinista es sistema con una cantidad finita de estados y señales, donde se pasa de un estado a otro al recibir una señal. Por ejemplo, supóngase que la idea es modelar el comportamiento de un interruptor típico de luz. El foco solo puede estar «prendido» o «apagado» (los estados). En cualquier caso, si se pulsa el interruptor (la señal) el foco dejará de estar prendido para estar apagado o dejará de estar apagado para estar prendido. Este sistema se puede representar fácilmente con un dibujito:



Se supone ahora que queremos complicar un poco el sistema, se tienen tres interruptores de luz asociados cada uno a un foco. Se tienen entonces tres señales ( $a$ ,  $b$  y  $c$ ), cada una corresponde a pulsar el interruptor de un foco diferente. Evidentemente hay  $2^3 = 8$  estados, uno por cada una de las siguientes posibilidades: todos los focos apagados, todos prendidos, el primero prendido y pero los demás apagados, etcétera. En un inicio todos los focos se encuentran en el estado tres-focos-apagados y se supone que se quieren distinguir aquellas secuencias finitas de señales que conducen al estado tres-focos-prendidos. Estas secuencias finitas de señales se pueden representar por medio de cadenas en  $\{a, b, c\}^*$ . Por ejemplo las cadenas  $abc$ ,  $baacbcbac$ , son algunas de las cadenas que representan secuencias tales que, a partir del estado de inicio tres-focos-apagados, después de ser ejecutada toda la secuencia de señales se llega a al estado tres-focos-prendidos:



El cambio de estado al recibir cada una de las señales se puede representar en una tabla (figura 2.1) que contiene toda la información. Esta tabla es un caso particular de un AFD, la definición general se expone con precisión en el cuadro azul de abajo. El conjunto de las cadenas que representan las secuencias buscadas es justamente el lenguaje asociado al autómata. La caracterización de este lenguaje se verá en la siguiente sección.

**Definición AFD.** Un *autómata finito determinista* es una quíntupla  $(Q, \Sigma, \delta, q_0, F)$ , donde:

$Q$  es un conjunto finito no vacío de *estados*;

$\Sigma$  es un conjunto finito no vacío de *símbolos de entrada*;

$\delta$  es una función con dominio  $Q \times \Sigma$  y con imagen contenida en  $Q$ , es decir, para todo estado  $q$  y símbolo de entrada  $a$ ,  $\delta(q, a)$  pertenece a  $Q$ . A  $\delta$  se le llama *función de transición*;

$q_0$  es un elemento de  $Q$ , llamado *estado inicial*;

$F$  es un subconjunto de  $Q$  de *estados finales o de aceptación*.

En principio el conjunto  $F$  puede ser  $\emptyset$  y además  $q_0$  puede estar en  $F$ . Una manera de representar a un AFD  $(Q, \Sigma, \delta, q_0, F)$  es a través de *tablas de transición*. Si  $Q = \{q_0, q_1, \dots, q_n\}$  y además  $\Sigma = \{a_1, a_2, \dots, a_m\}$ , se escribe:

	$a_1$	$\dots$	$a_m$
$\rightarrow q_0$	$\delta(q_0, a_1)$	$\dots$	$\delta(q_0, a_m)$
$\vdots$	$\vdots$		$\vdots$
$*q_n$	$\delta(q_n, a_1)$	$\dots$	$\delta(q_n, a_m)$

La flecha  $\rightarrow$  antes de un estado indica que se trata del estado inicial. A todos los estados de aceptación se les coloca un asterisco antes, por ejemplo, en esta tabla se indica que  $q_n$  es un estado de aceptación.

**Ejemplo 9.** El sistema de tres focos expuesto al inicio de la sección define claramente un AFD que se puede describir mediante la siguiente tabla de transición de la figura 2.1. El conjunto  $Q$  tiene 8 estados, escritos (y descritos) a través de los dibujitos de los tres focos (aunque se les podría haber llamado simplemente  $q_0, q_1, \dots, q_7$ , respectivamente).  $\Sigma$  es el conjunto  $\{a, b, c\}$ , el estado inicial  $q_0$  es el de tres-focos-apagados y el conjunto de estados de aceptación  $F$  tiene un solo elemento: el estado de tres-focos-prendidos.

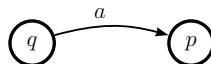
Existe otra manera, a veces muy útil, de representar gráficamente a un autómata finito determinista. Esta representación se llama *diagrama de transición* y se establece de la siguiente manera. Cada estado de  $q$  se representa con un nodo, a los nodos correspondientes a los estados de aceptación se les dibuja además un círculo concéntrico, al nodo que representa estado inicial  $q_0$  se le dibuja una flecha apuntando hacia él en la parte izquierda:



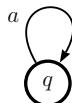
	<i>a</i>	<i>b</i>	<i>c</i>
$\rightarrow$	3 lightbulbs	2 lightbulbs	3 lightbulbs
	2 lightbulbs	3 lightbulbs	2 lightbulbs
	3 lightbulbs	2 lightbulbs	3 lightbulbs
	2 lightbulbs	3 lightbulbs	2 lightbulbs
	3 lightbulbs	2 lightbulbs	3 lightbulbs
	2 lightbulbs	3 lightbulbs	2 lightbulbs
*	3 lightbulbs	2 lightbulbs	3 lightbulbs
	2 lightbulbs	3 lightbulbs	2 lightbulbs

**Fig. 2.1** Tabla de transición del autómata de los tres focos.

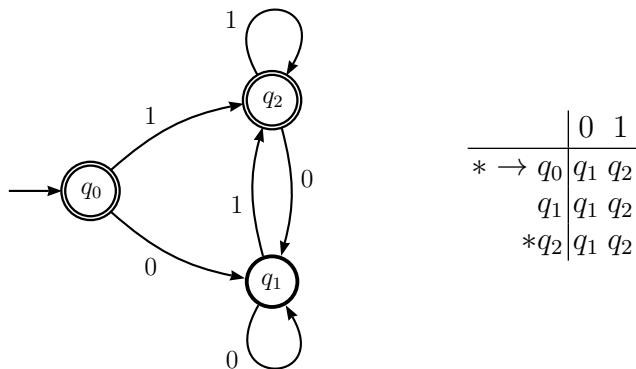
La expresión  $\delta(q, a) = p$  se representa mediante un dibujo en el cual se unen los nodos de  $q$  y  $p$  con una flechita etiquetada con  $a$ :



Si  $p = q$  se dibuja así, ver figura 2.2:

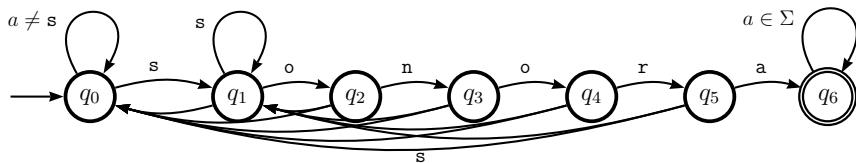


**Ejemplo 10.** La tarea es diseñar un AFD que reconozca la palabra **sonora** en un texto plano. Se propone el autómata con conjunto de estados  $\{q_0, q_1, \dots, q_6\}$  donde  $q_0$  es el estado inicial y para  $i = 1, \dots, 6$ , el estado  $q_i$  simboliza la situación «se ha leído la cadena de longitud  $i$  (**s**, **so**, **son**, **sono**, **sonor** y **sonora**, respectivamente)»; el alfabeto de símbolos de entrada es el conjunto de símbolos admitidos en un texto plano; la función de transición  $\delta$  se representa en la figura 2.3, el conjunto de estados de aceptación es evidentemente  $\{q_6\}$ . Se ha de notar que para  $i = 0, 1, \dots, 5$ ,  $\delta(q_i, \mathbf{s}) = q_1$  y que estando en  $q_i$  no se pasa a  $q_{i+1}$  al



**Fig. 2.2** Ejemplo de diagrama y tabla de transición de un mismo autómata.

menos que se haya leído una letra que contribuya a la lectura de **sonora**, de otro modo se pasa a  $q_0$ .



**Fig. 2.3** Autómata que lee sonora.

### Lista de ejercicios de la sección 2.1

*Ejercicio 12.* Diseñe un AFD que entre todas las cadenas binarias distinga únicamente aquellas que terminen en 1001.

*Ejercicio 13.* Diseñe un AFD que distinga solo a las cadenas de  $\{a, b, c\}^*$  de longitud mayor a 3 y menor a 6.

*Ejercicio 14.* Construya un AFD que reconozca solamente aquellos textos que *no* contienen a la palabra **sonora**.

*Ejercicio 15.* Diseñe un AFD que entre todas las cadenas de texto plano distinga únicamente a las que comienzan con **hola**.

*Ejercicio 16.* Para cada  $n > 0$  fijo, construya un AFD de  $n$  estados que distinga solamente a las cadenas de  $\{a\}^*$  de la forma  $a^{kn}$  donde  $k \geq 0$ .

*Ejercicio 17.* Considere una máquina despachadora de bebidas. Esta máquina solo vende dos tipos de bebida: agua purificada y horchata. El agua vale 15 pesos y la horchata vale 20 pesos. La máquina solo acepta monedas de 5 y de 10 pesos y no da cambio. Diseñe un AFD que modele el conteo de las monedas que el cliente está insertando. Ya que la máquina no da cambio, los estados de aceptación serán aquellos en el que el cliente haya insertado una cantidad suficiente para comprar  $n$  aguas y  $m$  horchatas (por supuesto  $m$  ó  $n$  pueden ser cero, pero no ambas a la vez, es decir si el cliente no inserta nada, no se estaría en un estado de aceptación). Por ejemplo, si el cliente introduce en total 25 pesos no se estaría en un estado de aceptación, ya que se podría comprar una horchata pero sobrarían 5 pesos. Si el cliente inserta en total 55 pesos, sí se estaría en un estado de aceptación ya que se podrían comprar 2 horchatas y un agua purificada exactamente.

*Ejercicio 18.* Busque en la literatura el algoritmo de Knuth-Morris-Pratts (KMP) sobre *pattern matching*. Consulte por ejemplo <https://www.cs.princeton.edu/~rs/AlgsDS07/21PatternMatching.pdf>. Este algoritmo está basado en la construcción de un AFD. Estudie e implemente el KMP para cadenas en el alfabeto binario.

## 2.2 Lenguaje de un autómata finito determinista

Hasta ahora se ha hablado de que «el autómata ha leído la cadena tal ...», «la cadena ha sido distinguida por el autómata ...», etc., para darle sentido formal a estas frases a continuación se define el proceso de lectura de una cadena de símbolos de entrada es mediante la relación binaria de duplas  $(q, w) \in Q \times \Sigma^*$ . Dos duplas están relacionadas, se escribe:

$$(q, x) \vdash (p, y) \tag{2.1}$$

si  $x = ay$  y  $p = \delta(q, a)$ . La dupla  $(q, x)$  se interpreta como «se está en el estado  $q$  y falta por leer  $x$ ». La relación (2.1) significa que «al leer el primer símbolo

*a* de la cadena  $x = ay$ , se pasa al estado  $p$  a partir de  $q$  y resta por leer  $y$ . Para representar varios movimientos en el autómata es pertinente considerar la clausura reflexiva y transitiva de la relación binaria  $\vdash$ , esto es:

**Definición**  $\vdash^*$ .

- R. INICIAL  $(q, x) \vdash (p, y)$  implica  $(q, x) \vdash^* (p, y)$   
en 1 movimiento;
  - R. INDUCTIVA si  $(q, x) \vdash^* (r, z)$  en  $n$  movimientos y  
 $(r, z) \vdash (p, y)$  entonces  $(q, x) \vdash^* (p, y)$   
en  $n + 1$  movimientos.
- Por convención,  $(q, x) \vdash^* (q, x)$  en 0 movimientos.

**Ejemplo 11.** El proceso de lectura de la cadena **sonora** a partir de cualquier estado  $p \neq q_6$  en el autómata el ejemplo 10, con la notación anterior de las duplas relacionadas es:

$$(p, \text{sonora}) \vdash (q_1, \text{onora}) \vdash (q_2, \text{nora}) \vdash (q_3, \text{ora}) \vdash (q_4, \text{ra}) \vdash (q_5, \text{a}) \vdash (q_6, \varepsilon).$$

Por otro lado, si  $w$  es cualquier cadena de texto plano:

$$(q_6, w) \vdash^* (q_6, \varepsilon).$$

**Definición L(A).** Sea  $A$  un AFD. Al conjunto  $L(A)$  de cadenas  $w \in \Sigma^*$  tales que  $(q_0, w) \vdash^* (q, \varepsilon)$  para algún estado  $q \in F$  se le llama *lenguaje aceptado por A*.

Esto es,  $L(A)$  es el conjunto de cadenas de símbolos de entrada tales que si a partir de  $q_0$  se comienza su lectura, al término del proceso se llega a algún estado de aceptación.

**Ejemplo 12.** Consideremos las siguientes variantes de nuestro AFD de «prendido y apagado»; vamos a definir cuatro autómatas, todos con conjunto de estados  $Q = \{q_a, q_p\}$ , conjunto de señales  $\Sigma = \{a\}$ , función de transición tal que  $\delta(q_a, a) = q_p$ ,  $\delta(q_p, a) = q_a$ , estado inicial  $\{q_a\}$ , pero cada uno con el siguiente respectivo conjunto de estados de aceptación:  $F = \emptyset$ ,  $F = \{q_a\}$ ,  $F = \{q_p\}$  y  $F = \{q_a, q_p\}$ . El lenguaje aceptado por cada uno de los autómatas dependen, por supuesto, de quien es  $F$ . En la siguiente tabla se exponen los casos:

$F$	lenguaje	razón
$\emptyset$	$\emptyset$	No hay cadenas para las que se llegue a un estado de aceptación a partir del estado inicial $q_a$ , pues ¡no hay estados de aceptación!
$\{q_a\}$	$\{a^{2m} : m \geq 0\}$	Se requiere un número par de señales para ir de $q_a$ a $q_a$ . Notar que $(q_a, \varepsilon) \xrightarrow{*} (q_a, \varepsilon)$ en 0 movimientos por lo que $\varepsilon$ pertenece al lenguaje aceptado por el autómata.
$\{q_p\}$	$\{a^{2m+1} : m \geq 0\}$	Un número impar de señales se requiere para llegar del estado $q_a$ al estado $q_p$ .
$\{q_a, q_p\}$	$\Sigma^*$	Es obvio, pues todos los estados son de aceptación.

Dada una cadena  $w \in \Sigma^*$  existe un único camino que describe el proceso de lectura de  $w$  en  $A$  a partir de un estado  $p$ . Por tanto, para cualquier cadena  $w$ , existe un único proceso de lectura  $(p, w) \xrightarrow{*} (q, \varepsilon)$ . El último y único estado de llegada  $q$  define una función que depende de  $q$  y de  $w$ , llamada función de transición extendida mediante la cual es posible definir equivalentemente a  $L(A)$  como sigue:

**Definición L(A).** La *función de transición extendida*  $\hat{\delta}$  de un autómata finito determinista  $A = (Q, \Sigma, \delta, q_0, F)$  es una aplicación con dominio en  $Q \times \Sigma^*$  y con imagen contenida en  $Q$  tal que para todo  $q \in Q$ ,  $x \in \Sigma^*$  y  $a \in \Sigma$ :

- R. INICIAL  $\hat{\delta}(q, \varepsilon) = q$ ;
- R. INDUCTIVA  $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ .

Con esta notación, el *lenguaje aceptado por A* está dado por:

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

Como es de esperarse, se cumple efectivamente lo siguiente:

**Propiedad 2.** Sean  $A$  un AFD y  $\delta$  su función de transición. Para toda cadena  $w$  de símbolos de entrada y cualquier estado  $q$  se cumple que:

$$(q, w) \xrightarrow{*} (\hat{\delta}(q, w), \varepsilon).$$

La demostración se deja como ejercicio para el lector [ejercicio 22]. En particular,  $\hat{\delta}(q_0, w) \in F$  si y solo si  $(q_0, w) \vdash^* (p, \varepsilon)$  para algún estado de aceptación  $p$ . Es decir, las dos nociones de lenguaje aceptado por un autómata finito determinista presentadas en esta sección coinciden. Por otro lado, si se parte de un lenguaje  $L$  y se diseña un autómata finito determinista  $A$  que acepte a  $L$  como lenguaje, el diseño es correcto si  $L(A) = L$ . En otras palabras, se deben evitar los:

*Falsos negativos:* se ha de verificar que si  $w \in L$  entonces  $\hat{\delta}(q_0, w) \in F$ , las cadenas de  $L$  llegan a un estado de aceptación a partir del estado inicial; y  
*Falsos positivos:* se tiene que checar que si  $\hat{\delta}(q_0, w) \in F$  entonces  $w \in L$ , las cadenas que llegan a un estado de aceptación a partir del estado inicial, deben por fuerza estar en  $L$ .

**Ejemplo 13.** Sea  $A = (Q, \Sigma, \delta, q_0, F)$  el AFD descrito en el ejemplo 10. El lenguaje aceptado por  $A$  es el conjunto, que denotaremos por  $L_{\text{son}}$ , de cadenas de texto plano que contienen a la subcadena **sonora**. Sea  $w = x\text{sonor}az$  y  $p = \hat{\delta}(q_0, x)$  entonces, por el ejemplo 11 y el ejercicio 26:

$$(q_0, x\text{sonor}az) \vdash^* (p, \text{sonor}az) \vdash^* (q_6, z) \vdash^* (q_6, \varepsilon).$$

Por tanto  $w \in L(G)$ . La otra implicación se prueba inductivamente sobre la longitud de la cadena, específicamente se prueba la siguiente afirmación para  $n \geq 0$ .

$$P(n) : \text{si } |w| \leq n \text{ y } \hat{\delta}(q_0, w) = q_6, \text{sonora es subcadena de } w.$$

La afirmación  $P(0)$  se cumple trivialmente pues la premisa es falsa. Se supone cierta  $P(n)$  (hipótesis de inducción). Sea  $w$  tal que  $|w| = n + 1$  y  $\hat{\delta}(q_0, w) = q_6$ . Se supone por contradicción que  $w$  no contiene a **sonora**. En el peor de los casos,  $w$  podría contener a  $y_1 = s$ ,  $y_2 = so$ ,  $y_3 = son$ ,  $y_4 = sono$  y a  $y_5 = sonor$ ; de lo contrario se llegaría a la contradicción  $\hat{\delta}(q_0, w) = q_0$ . Se escinde a la cadena  $w$  en la forma  $xy_iz$ , donde  $y_i$  es la última subcadena de este tipo que aparece en  $w$ . Por hipótesis de inducción como  $|x| \leq n$  se tiene que  $p = \hat{\delta}(q_0, x) \neq q_6$ , pues de otra forma **sonora** sería subcadena de  $x$  y por tanto de  $w$ . Luego, para  $i = 1, \dots, 5$ :

$$(q_0, xy_iz) \vdash^* (p, y_iz) \vdash^* (q_i, z).$$

Supongamos que  $z \neq \varepsilon$ , de lo contrario llegaríamos directamente a la contradicción  $\hat{\delta}(q_0, w) = q_i \neq q_6$ . Sea  $z = au$ , donde  $a \in \Sigma$ . Si  $y_i = y_5$ ,  $a$  no puede ser ni **s**, porque se supone que  $y_1$  no está en  $z$ ; ni **a**, porque entonces **sonora** sería

subcadena de  $w$ . Además,  $u$  no continene a  $y_1 = \mathbf{s}$ , luego:

$$(q_5, z) \vdash (q_0, u) \vdash^* (q_0, \varepsilon).$$

De nuevo se llega a una contradicción. Los casos  $y_i = y_1, \dots, y_4$ , se razonan de manera análoga. En conclusión  $L_{\text{son}} = L(A)$ .

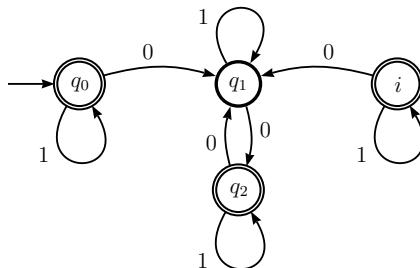
### **Lista de ejercicios de la sección 2.2**

*Ejercicio 19.* Sea  $A = (Q, \Sigma, \delta, q_0, F)$  un AFD y sea  $A^c = (Q, \Sigma, \delta, q_0, Q \setminus F)$ . Pruebe que  $L(A^c) = \Sigma^* \setminus L(A)$ .

*Ejercicio 20.* Sea  $A$  el AFD de la figura 2.2. Verifique que:

$$L(A) = \{\varepsilon\} \cup \{x1 : x \text{ es una cadena binaria}\}.$$

*Ejercicio 21.* Se dice que un estado  $i$  de un AFD es *inaccesible* si no existe alguna cadena de símbolos de entrada  $w$  tal que  $\hat{\delta}(q_0, w) = i$ . Por ejemplo, el estado  $i$  del siguiente autómata es inaccesible, no hay manera de llegar a él partiendo del estado inicial.



1. Sean  $A = (Q, \Sigma, \delta, q_0, F)$  un AFD y sea  $I$  el conjunto de estados inaccesibles de  $A$ . Sea  $A'$  el AFD  $(Q \setminus I, \Sigma, \gamma, q_0, F \setminus I)$ , donde  $\gamma(q, a) = \delta(q, a)$ , para todo  $q \in Q \setminus I$  y  $a \in \Sigma$ . Pruebe que  $L(A) = L(A')$ .
2. Escriba un algoritmo para encontrar todos los estados inaccesibles de un AFD. Sugerencia: escribe un algoritmo (regla inicial y regla inductiva) para encontrar el conjunto de los estados *accesibles* y luego considere el complemento de ese conjunto.

*Ejercicio 22.* Pruebe la propiedad 2.

*Ejercicio 23.* Pruebe que  $L_a = \{a^{2m+1} : m \geq 0\}$  es el lenguaje aceptado por el autómata del ejemplo 12 con  $F = \{q_p\}$ .

*Ejercicio 24.* Considere el AP de la figura 2.5. Demuestre que este autómata acepta como lenguaje al conjunto de cadenas binarias con un número impar de ceros y un número impar de unos. Sugerencia: usar inducción simultánea de forma análoga a la sugerencia propuesta para resolver el ejercicio 23.

*Ejercicio 25.* Verifique que los autómatas que usted diseñó en el ejercicio 16 son correctos.

*Ejercicio 26.* Sean  $A$  un AFD con función de transición  $\delta$ , sean  $x$  y  $y$  cadenas de símbolos de entrada y sean  $q$  y  $p$  estados de  $A$ . Pruebe que para cualquier cadena de símbolos de entrada  $w$ :

$$(q, x) \xrightarrow{*} (p, y) \text{ implica } (q, xw) \xrightarrow{*} (p, yw).$$

Sugerencia: razoné por inducción sobre el número de movimientos.

*Ejercicio 27.* A partir del ejercicio 26, pruebe que para cualquier cadena de símbolos de entrada  $w$ ,  $\hat{\delta}(q, xw) = \hat{\delta}(\hat{\delta}(q, x), w)$ .

## 2.3 Construcción del autómata de la intersección

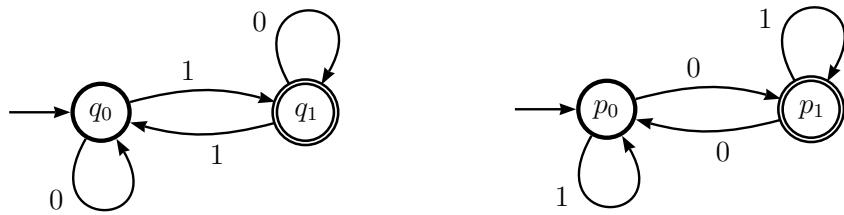
Dados dos AFD  $A$  y  $B$  se puede construir fácilmente un nuevo AFD  $C$  que acepte a  $L(A) \cap L(B)$  como lenguaje. Para aterrizar el problema, se consideran los autómatas  $A$  y  $B$  descritos en la figura 2.4. Una cadena es aceptada por los dos autómatas si y solo si al recorrer ambos simultáneamente se llega a los estados  $q_1$  y  $p_1$ , respectivamente. Este recorrido simultáneo puede ser representado naturalmente a través de pares ordenados. Por ejemplo, para leer simultáneamente la cadena 0111 se comienza en el par-estado  $(q_0, p_0)$ . Se recibe la señal 0: el autómata  $A$  indica permanecer en  $q_0$  y el autómata  $B$  indica cambio de estado a  $p_1$ ; en resumen se cambia al par-estado  $(q_0, p_1)$  al leer 0. En nuestra notación de duplas se tendría:

$$((q_0, p_0), 0111) \vdash ((q_0, p_1), 111)$$

y así sucesivamente el recorrido simultáneo se completa como sigue:

$$((q_0, p_1), 111) \vdash ((q_1, p_1), 11) \vdash ((q_0, p_1), 1) \vdash ((q_1, p_1), \varepsilon).$$

La cadena será aceptada si y solo si el último par-estado está constituido por dos estados de aceptación, en este caso es  $(q_1, p_1)$ . La construcción precisa y en general se establece en (2.2).



**Fig. 2.4** Autómata  $A$  que acepta como lenguaje al conjunto de cadenas binarias con una cantidad impar de unos y autómata  $B$  que acepta como lenguaje al conjunto de cadenas binarias con una cantidad impar de ceros.

**Algoritmo AFD de  $L(A) \cap L(B)$ .** Sea  $A = (Q, \Sigma, \delta, q_0, F)$  y sea  $B = (P, \Sigma, \gamma, p_0, H)$  dos AFD. Un autómata que acepta a  $L(A) \cap L(B)$  como lenguaje es el autómata

$$C = (Q \times P, \Sigma, \eta, (q_0, p_0), F \times H)$$

con función de transición:

$$\eta((q, p), a) = (\hat{\delta}(q, a), \hat{\gamma}(p, a)). \quad (2.2)$$

Para justificar formalmente que  $C$  es el autómata que se está buscando, se ha de probar primero que, para todo  $n \geq 0$ :

$$P(n) : \text{si } |w| = n \text{ entonces } \hat{\eta}((q_0, p_0), w) = (\hat{\delta}(q_0, w), \hat{\gamma}(p_0, w)).$$

La afirmación  $P(0)$  se cumple trivialmente. Supongamos cierta  $P(n)$ . Si  $|x| = n$ , se tiene:

$$\begin{aligned} \hat{\eta}((q_0, p_0), xa) &= \eta(\hat{\eta}((q_0, p_0), x), a) \\ &= \eta\left(\left(\hat{\delta}(q_0, x), \hat{\gamma}(p_0, x)\right), a\right) \\ &= \left(\delta\left(\hat{\delta}(q_0, x), a\right), \gamma\left(\hat{\gamma}(p_0, x), a\right)\right) \\ &= \left(\hat{\delta}(q_0, xa), \hat{\gamma}(p_0, xa)\right). \end{aligned}$$

Con esto se demuestra  $P(n+1)$ . Resta verificar que  $L(C) = L(A) \cap L(B)$ , pero esto es fácil, ya que  $w$  está en  $L(C)$  si y solo si  $\hat{\eta}((q_0, p_0), w)$  pertenece a  $F \times H$  si y solo si  $\hat{\delta}(q_0, w) \in F$  y  $\hat{\gamma}(p_0, w) \in H$  si y solo si  $w \in L(A)$  y  $w \in L(B)$ .

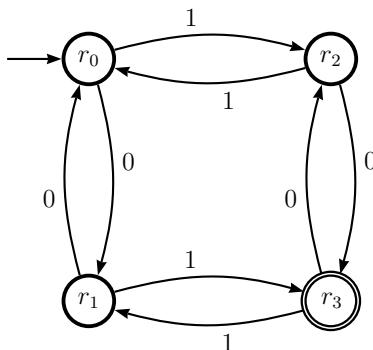
**Ejemplo 14.** Sean  $A = (Q, \Sigma, \delta, q_0, F)$  y  $B = (P, \Sigma, \gamma, p_0, H)$  los autómatas descritos mediante en los diagramas de la figura 2.4. El autómata  $A$  acepta el conjunto de cadenas binarias con una cantidad impar de unos y el autómata  $B$  acepta el conjunto de cadenas binarias con una cantidad impar de ceros. Siguiendo el procedimiento descrito anteriormente, se tiene que:

$$Q \times P = \{r_0 = (q_0, p_0), r_1 = (q_0, p_1), r_2 = (q_1, p_0), r_3 = (q_1, p_1)\}.$$

La construcción anterior arroja lo siguiente tabla de transición para  $\eta$ :

	0	1
$\rightarrow r_0$	$(\delta(q_0, 0), \gamma(p_0, 0)) = (q_0, p_1) = r_1$	$(\delta(q_0, 1), \gamma(p_0, 1)) = (q_1, p_0) = r_2$
$r_1$	$(\delta(q_0, 0), \gamma(p_1, 0)) = (q_0, p_0) = r_0$	$(\delta(q_0, 1), \gamma(p_1, 1)) = (q_1, p_1) = r_3$
$r_2$	$(\delta(q_1, 0), \gamma(p_0, 0)) = (q_1, p_1) = r_3$	$(\delta(q_1, 1), \gamma(p_0, 1)) = (q_0, p_0) = r_0$
$*r_3$	$(\delta(q_1, 0), \gamma(p_1, 0)) = (q_1, p_0) = r_2$	$(\delta(q_1, 1), \gamma(p_1, 1)) = (q_0, p_1) = r_1$

El diagrama de transición del autómata  $C$  definido mediante la construcción anterior se representa en la figura 2.5.



**Fig. 2.5** AFD que acepta al conjunto de cadenas binarias con una cantidad impar de ceros y una cantidad impar de unos.

Note que los estados del AFD de la figura 2.5 tienen significados por sí mismos, por ejemplo,  $r_0$  es el estado de «lectura de una cantidad par de unos y par de ceros», los otros tres estados completan las otras tres posibilidades de lectura.

### **Lista de ejercicios de la sección 2.3**

*Ejercicio 28.* Sean  $A$  y  $B$  dos AFD con un mismo alfabeto de símbolos de entrada. Construya un autómata finito determinista que acepte a  $L(A) \cup L(B)$  como lenguaje. Sugerencia: tome en cuenta el ejercicio 19 y las leyes de De Morgan.

*Ejercicio 29.* Diseñe un AFD que acepte a  $L(A) \setminus L(B)$  como lenguaje dados dos AFD  $A$  y  $B$ .

*Ejercicio 30.* Defina un AFD que acepte como lenguaje al conjunto de cadenas que no contienen a la palabra reservada `then` pero que contienen a la palabra reservada `else`.

*Ejercicio 31.* Diseñe un AFD que acepte como lenguaje al conjunto de cadenas binarias que contengan a 10001 como subcadena y que terminen en 0101.

*Ejercicio 32.* Diseñe un AFD que acepte como lenguaje al conjunto de cadenas en el alfabeto  $\{a, b, c\}$  que comience con  $abc$  o con  $cba$  y que además no contengan a las palabras  $ba, baba, bababa, \dots$

*Ejercicio 33.* Diseñe un AFD que acepte como lenguaje al conjunto de cadenas de texto plano que contengan a las palabras **azul**, **amarillo** y **rojo**.

*Ejercicio 34.* Sean  $A$ ,  $B$  y  $C$  como en la definición (2.2). Demuestre que para toda cadena  $w$  y cualesquiera estados  $q$  y  $p$  de  $A$  y  $B$ , respectivamente, se cumple que:

$$((q_0, p_0), w) \xrightarrow{*} ((q, p), \varepsilon) \text{ si y solo si } (q_0, w) \xrightarrow{*} (q, \varepsilon) \text{ y } (p_0, w) \xrightarrow{*} (p, \varepsilon).$$

*Ejercicio 35.* Sean  $A_1, A_2, \dots, A_n$  AFD con un mismo alfabeto de entrada. Define un autómata  $C$  que acepte como lenguaje al conjunto:

$$L(A_1) \cap L(A_2) \cap \dots \cap L(A_n).$$

*Ejercicio 36.* A partir de la construcción del ejercicio 35, escriba un programa que reciba  $n$  AFD con un mismo alfabeto y regrese el AFD que acepta como lenguaje la intersección de los lenguajes de los autómatas dados. Haga otro programa con la misma entrada pero que regrese un AFD que acepte como lenguaje la unión de los lenguajes de los autómatas dados.

# CAPÍTULO 3

---

## Autómatas no-deterministas

---

En este capítulo se estudian otro tipo sistemas de estados y señales en la misma línea del capítulo anterior. La diferencia fundamental es que las transiciones no son deterministas, en el sentido de que dado un estado y una señal pueden haber muchos estados de llegada. Por consiguiente, existen varias posibilidades de lectura de una cadena, lo cual aporta flexibilidad al momento de querer modelar procesos con un número finito de estados y señales.

Este capítulo se divide en tres secciones. En la primera sección se estudian los llamados autómatas finitos no-deterministas (AFND). Para estos sistemas, dado un estado y una señal de entrada, la función de transición regresa un conjunto de estados. En la segunda sección se establece un tipo de autómatas no-deterministas más general, estos sistemas admiten adicionalmente transiciones instantáneas (AFND- $\varepsilon$ ), esto es, estando en un estado es posible pasar a otros sin necesidad de gastar símbolos de entrada. En la tercera y última sección se presenta un resultado que afirma que todos los tipos de autómatas finitos estudiados hasta el momento son en algún sentido equivalentes. Las definiciones de este capítulo y el teorema de equivalencia corresponde de origen a [20].

### 3.1 Autómatas finito no-deterministas

Un autómata finito no-determinista es un sistema con un número finito de estados y señales pero que —a diferencia de los autómatas deterministas— admite más de un estado de llegada una vez recibida una señal. Esto es, la función de transición  $\delta$  le asigna a un par estado-señal  $(q, a)$  un conjunto:

$$\delta(q, a) = \{p_1, \dots, p_m\}$$

donde cada estado  $p_i$ ,  $i = 1, 2, \dots, m$ , representa un posible estado de llegada. Además  $\delta(q, a)$  pudiera ser el conjunto vacío. Formalmente:

**Definición AFND.** Un *autómata finito no-determinista* es una quíntupla  $(Q, \Sigma, \delta, q_0, F)$  donde:

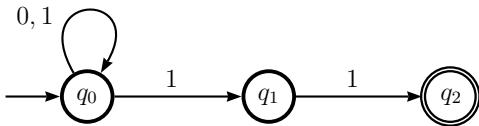
- $Q$  es un conjunto finito no vacío de *estados*;
- $\Sigma$  es un conjunto finito no vacío de *símbolos de entrada*;
- $\delta$  es la *función de transición* la cual recibe un estado  $q$  y símbolo de entrada  $a$  y regresa un subconjunto de  $Q$ ;
- $q_0$  es el *estado inicial* perteneciente a  $Q$ ;
- $F$  es el subconjunto de *estados de aceptación* de  $Q$ .

La tabla de transición de un AFND es igual a la de uno determinista. Respecto al diagrama de transición, la única diferencia es que las transiciones del tipo  $\delta(q, a) = \emptyset$  se omiten. Dos duplas de  $Q \times \Sigma^*$  están relacionadas, esto es  $(q, x) \vdash (p, y)$ , si  $x = ay$  y además  $p \in \delta(q, a)$ . Esto representa un posible movimiento en el autómata. Para representar más de un movimiento, se considera la relación  $\vdash^*$  definida igual que en el caso determinista.

**Ejemplo 15.** Sea  $A = (Q, \Sigma, \delta, q_0, F)$  el autómata descrito mediante el diagrama de transición que se muestra en la figura 3.1. En este caso  $Q = \{q_0, q_1, q_2\}$ ,  $\Sigma = \{0, 1\}$ ,  $F = \{q_2\}$  y además:

$$\begin{array}{ll} \delta(q_0, 0) = \{q_0\} & \delta(q_0, 1) = \{q_0, q_1\} \\ \delta(q_1, 0) = \emptyset & \delta(q_1, 1) = \{q_2\} \\ \delta(q_2, 0) = \emptyset & \delta(q_2, 1) = \emptyset \end{array}$$

La transición  $\delta(q_0, 1) = \{q_0, q_1\}$  significa que estando en  $q_0$  al recibir 1 hay dos posibilidades: quedarse en  $q_0$  o pasar al estado  $q_1$ . La transición  $\delta(q_1, 1) = \{q_2\}$  se interpreta como estando en  $q_1$ , al recibir 1 se pasa con seguridad a  $q_2$ . Por ejemplo, la cadena 00011 puede ser leída de muchas formas. Un camino consiste en permanecer en  $q_0$  durante toda la lectura:



**Fig. 3.1** AFND que acepta a las cadenas del alfabeto binario que terminan en 11.

$$(q_0, 00011) \vdash (q_0, 0011) \vdash (q_0, 011) \vdash (q_0, 11) \vdash (q_0, 1) \vdash (q_0, \varepsilon).$$

Otro camino puede ser, por ejemplo, la siguiente ruta:

$$(q_0, 00011) \vdash (q_0, 0011) \vdash (q_0, 011) \vdash (q_0, 11) \vdash (q_1, 1) \vdash (q_2, \varepsilon).$$

Finalmente, la transición  $\delta(q_2, 0) = \emptyset$  se puede interpretar como «estando en  $q_2$  si se recibe la señal 1 entonces no se va a ningún estado». Por ejemplo, un posible camino de lectura para la cadena 11100 es el siguiente:

$$(q_0, 11100) \vdash (q_1, 1100) \vdash (q_2, 100).$$

Por este camino ya no se puede avanzar y concluir el proceso de lectura de 11100. Se tienen a la vista las siguientes disimilitudes respecto a los autómatas deterministas. Dada una cadena  $w$  y un estado  $q$ :

- en un AFD siempre existe un camino  $(q, w) \stackrel{*}{\vdash} (p, \varepsilon)$ . Es decir, la cadena siempre se puede terminar de leer a partir de cualquier estado  $q$ , a diferencia de un AFND. Por ejemplo, la dupla  $(q_1, 0101)$  no está relacionada con ninguna otra dupla distinta;
- un AFND puede tener varios caminos de recorrido de lectura de una cadena, a diferencia de un AFD.

Igual que en el caso de un AFD se tiene que una cadena de símbolos de entrada  $w$  es distinguida por un AFND si y solo si existe un camino:

$$(q_0, w) \stackrel{*}{\vdash} (q, \varepsilon)$$

donde  $q$  es un estado de aceptación. Por ejemplo, el autómata del ejemplo 15 acepta como lenguaje al conjunto de cadenas binarias que terminan en 11. Alternativamente, también en este caso se puede definir el lenguaje aceptado por un autómata mediante una función de transición extendida.

**Definición L(A).** Sea  $q$  es un estado y sean  $x$  y  $a$ , respectivamente, una cadena y un símbolo de entrada. La *función de transición extendida*  $\hat{\delta}$  de un AFND  $A = (Q, \Sigma, \delta, q_0, F)$  se define inductivamente:

- R. INICIAL  $\hat{\delta}(q, \varepsilon) = \{q\}$ ;
- R. INDUCTIVA  $\hat{\delta}(q, x) = \{p_1, \dots, p_m\}$  implica:

$$\hat{\delta}(q, xa) = \bigcup_{i=1}^m \delta(p_i, a).$$

Si  $\hat{\delta}(q, x) = \emptyset$  entonces  $\hat{\delta}(q, xa) = \emptyset$ . Al conjunto

$$L(A) = \{w \in \Sigma^* : \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

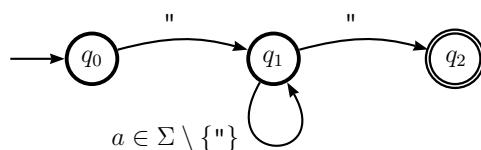
se le llama *lenguaje aceptado por A*.

Con esta definición de función de transición extendida, se cumple la siguiente propiedad [ejercicio 39].

**Propiedad 3.** Sea  $\delta$  la función de transición de un AFND. Para todo estado  $q$  y cadena de símbolos de entrada  $w$ :

$$\hat{\delta}(q, w) = \{r \in Q : (q, w) \vdash^* (r, \varepsilon)\}.$$

**Ejemplo 16.** Sea  $A$  el AFND con alfabeto  $\Sigma$  de símbolos de texto plano:



Es fácil reconocer el siguiente recorrido:

$$(q_0, "hola_mundo") \vdash (q_1, hola_mundo) \vdash^* (q_1, ") \vdash (q_2, \varepsilon).$$

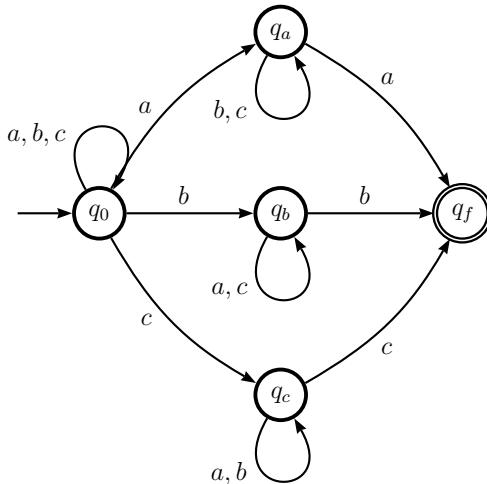
Aunque el autómata es no-determinista, este es el único camino que a partir de la dupla  $(q_0, "hola_mundo")$  el proceso se concluye con la lectura completa de la cadena *"hola\_mundo"*. Por tanto,  $\hat{\delta}(q_0, "hola_mundo") = \{q_2\}$ . Por otro lado,  $\hat{\delta}(q_0, hola_mundo) = \emptyset$ , pues de entrada  $\hat{\delta}(q_0, h) = \emptyset$ . Este autómata distingue las frases escritas entre comillas, sin que aparezcan comillas en el interior [ejercicio

43]. Note que por ejemplo la cadena "mira" a no es aceptada aunque se llegue al estado de aceptación  $q_2$ . En este caso el único recorrido a partir del inicio es:

$$(q_0, \text{"mira"} a) \vdash (q_1, \text{"mira"} a) \vdash^* (q_1, \text{"a}) \vdash (q_2, \text{a}).$$

La cadena no es aceptada pues el proceso se paró antes de terminar la lectura de la cadena completa, pues  $\delta(q_2, \text{a}) = \emptyset$ .

**Ejemplo 17.** Sea  $\Sigma = \{a, b, c\}$  un alfabeto. Se está interesado en diseñar un autómata que distinga aquellas cadenas en  $\Sigma^*$  cuya última entrada aparezca *al menos* dos veces. Por ejemplo, las cadenas *aabaca*, *babbcc* y *bccb* deben ser distinguidas, mientras las cadenas *bbccbba*, *aaabbac* no. Se propone el siguiente autómata finito no-determinista:



Se supone que existe una cadena aceptada  $w = xa$  con  $x \in \{b, c\}^*$ . Es decir la última entrada  $a$  de  $w$  no aparece al menos dos veces pero es aceptada por el autómata. Entonces,  $(q_0, xa) \vdash^* (r, a)$  para algún estado  $r \neq q_a$ , pues  $x$  no contiene símbolos  $a$ . A partir de  $(r, a)$  es imposible llegar a  $q_f$ , pues el único estado que cuando recibe  $a$  se cambia a  $q_f$  es precisamente  $q_a$ . Se llega a una contradicción. Si  $w = xb$  o  $w = xc$  se razona de igual manera.

Sea  $w$  una cadena cuya última entrada aparece al menos dos veces en la cadena, digamos  $w = xaya$ , donde  $y \in \{b, c\}^*$  y  $x \in \Sigma^*$ . Para este caso, se ha preferido usar la notación de duplas, pues la representación es muy clara, ya que evidentemente existe el camino:

$$(q_0, xaya) \xrightarrow{*} (q_0, aya) \xrightarrow{*} (q_a, ya) \xrightarrow{*} (q_a, a) \xrightarrow{*} (q_f, \varepsilon).$$

Si la última entrada es  $b$  o  $c$  se razona de la misma forma.

### **Lista de ejercicios de la sección 3.1**

*Ejercicio 37.* Para cada uno de los incisos, diseña un AFND que acepte únicamente cadenas binarias tales que:

1. terminen en 0011;
2. contengan al menos tres unos;
3. comiencen en 110 y acaben en 0110;
4. contengan la subcadena 1010.

*Ejercicio 38.* Modifique al AFND del ejemplo 16 para que distinga aquellas cadenas que contengan al menos una frase entre comillas.

*Ejercicio 39.* Pruebe la propiedad 3. Sugerencia: demuestre por inducción, que para todo  $n \geq 0$ ,

$$P(n): \text{si } |w| = n \text{ y } r \in \hat{\delta}(q, w) \text{ entonces } (q, w) \xrightarrow{*} (r, \varepsilon).$$

$$Q(n): \text{si } (q, w) \xrightarrow{*} (r, \varepsilon) \text{ en } n \text{ movimientos entonces } r \in \hat{\delta}(q, w).$$

*Ejercicio 40.* Demuestre el ejercicio 26 suponiendo que el autómata  $A$  es no-determinista. Concluya que si  $\hat{\delta}(q, x) = \{p_1, \dots, p_m\}$ , para algunos estados  $q, p_1, p_2, \dots, p_m$  y cadena de símbolos de entrada  $x$ , entonces para cualquier cadena de símbolos de entrada  $w$ :

$$\hat{\delta}(q, xw) = \hat{\delta}(p_1, w) \cup \dots \cup \hat{\delta}(p_m, w).$$

*Ejercicio 41.* Sea  $A$  el autómata del ejemplo 15, verifique que  $L(A)$  es el conjunto de cadenas binarias que terminan en 11.

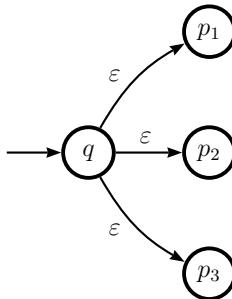
*Ejercicio 42.* Diseña un AFND que acepte como lenguaje el conjunto de números naturales que terminan en 3, 6 o 9.

*Ejercicio 43.* Sea  $A$  el autómata del ejemplo 16, pruebe que  $L(A)$  es el conjunto de cadenas de un texto plano de la forma  $w = "x"$  donde  $x \in (\Sigma \setminus \{"\})^*$ .

*Ejercicio 44.* Demuestre que si un lenguaje es aceptado por un AFD entonces es aceptado por un AFND.

### 3.2 Autómatas con transiciones instantáneas

Como ya se ha visto antes, la cadena  $\varepsilon abbaac\varepsilon cca$  es igual a la cadena  $abbaaccca$ . A veces, en el diseño de un autómata, es más fácil pensar que  $\varepsilon$  es también una entrada de la cadena. El siguiente diagrama:



se podría interpretar como: a partir del estado  $q$  es posible ir a los estados  $p_1$ ,  $p_2$  o  $p_3$  «instantáneamente», sin haber recibido algún símbolo de entrada. La definición formal de un *autómata finito no-determinista con transiciones instantáneas*  $A = (Q, \Sigma, \delta, q_0, F)$  —abreviado AFND- $\varepsilon$ — es exactamente igual a la de un autómata finito no-determinista, salvo que el dominio de la función  $\delta$  es:

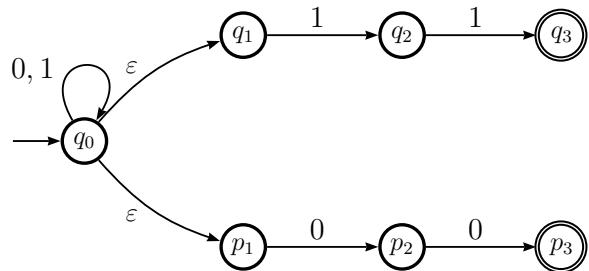
$$Q \times \Sigma \cup \{\varepsilon\}.$$

Al igual que antes, dos duplas de  $Q \times \Sigma^*$  están relacionadas, esto es:

$$(q, x) \vdash (p, y),$$

si  $x = ay$  y  $p \in \delta(q, a)$ . Tome en cuenta, que para autómatas con transiciones instantáneas,  $a$  puede ser la cadena vacía, en este caso se tendría  $(q, y) \vdash (p, y)$ . Al igual que en los casos anteriores, para representar más de un movimiento, se considera la relación  $\vdash^*$ .

**Ejemplo 18.** Consideremos el AFND- $\varepsilon$  que se muestra en la figura 3.2. La transición  $\delta(q_0, \varepsilon) = \{q_1, p_1\}$  se interpreta como sigue: a partir del estado inicial  $q_0$  se puede cambiar instantáneamente al estado  $q_1$  o al estado  $p_1$  sin necesidad de haber recibido señal alguna. El resto de las transiciones se interpretan igual que en un AFND. Note que tan solo la cadena  $w = 1$  admite cuatro caminos de lectura que llevan a una dupla de la forma  $(q, \varepsilon)$ :



**Fig. 3.2** Autómata que acepta a las cadenas binarias que acaban en 11 o en 00.

$$\begin{aligned}
 (q_0, 1) &\vdash (q_0, \varepsilon) \\
 &\vdash (q_0, \varepsilon) \vdash (q_1, \varepsilon) \\
 &\vdash (q_0, \varepsilon) \vdash (p_1, \varepsilon) \\
 &\vdash (q_1, 1) \vdash (q_2, \varepsilon)
 \end{aligned}$$

Como es de esperarse, una cadena es distinguida por el AFND- $\varepsilon$  si existe un camino tal que a partir del estado inicial, al recorrer el autómata se llega a un estado de aceptación. En este autómata, las transiciones instantáneas que parten de  $q_0$  nos abren la posibilidad de distinguir a las cadenas que terminan en 11 o que acaban en 00.

Para definir a la función de transición  $\hat{\delta}$  se requiere primero establecer al conjunto de estados a los que se llega de forma instantánea a partir de un estado.

**Definición clau( $q$ ).** Dado un autómata finito no-determinista con transiciones instantáneas y función de transición  $\delta$ , se define la *clausura respecto a  $\varepsilon$*  de un estado  $q$  —se denotará  $\text{clau}(q)$ — de manera inductiva:

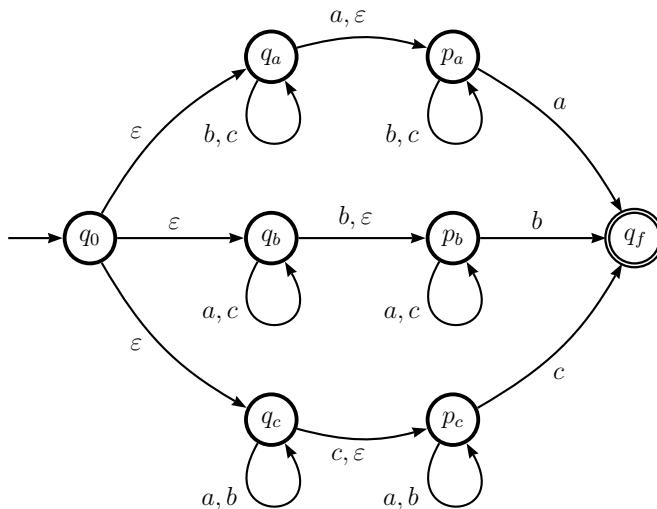
- R. INICIAL  $q \in \text{clau}(q)$ ;
- R. INDUCTIVA si  $p \in \text{clau}(q)$  entonces  $\delta(p, \varepsilon) \subset \text{clau}(q)$ .

Como es de esperarse, se tiene la siguiente propiedad [ejercicio 47]:

**Propiedad 4.** Sea  $\delta$  la función de transición de un AFND- $\varepsilon$  con conjunto de estados  $Q$ . Para todo estado  $q \in Q$ :

$$\text{clau}(q) = \{r \in Q : (q, \varepsilon) \xrightarrow{*} (r, \varepsilon)\}.$$

**Ejemplo 19.** Sea  $\Sigma = \{a, b, c\}$ . Se está interesado en diseñar un AFND- $\varepsilon$  que distinga aquellas cadenas en  $\Sigma^*$  cuya última entrada aparezca *a lo más* dos veces. Por ejemplo, las cadenas *caccba*, *babc* y *bcaccc* deben ser distinguidas, mientras las cadenas *bcacaaa*, *ababcb* no. Se propone el autómata descrito mediante el siguiente diagrama de transición [ejercicio 50]:



El cálculo de  $\text{clau}(q_0)$  siguiendo la definición del recuadro sería de la siguiente manera:

n.º de iteración	estados en $\text{clau}(q_0)$	total acumulado
0	$q_0$	$\{q_0\}$
1	$\delta(q_0, \varepsilon) = \{q_a, q_b, q_c\}$	$\{q_0, q_a, q_b, q_c\}$
2	$\delta(q_a, \varepsilon) = \{p_a\}$ $\delta(q_b, \varepsilon) = \{p_b\}$ $\delta(q_c, \varepsilon) = \{p_c\}$	$\{q_0, q_a, q_b, q_c, p_a, p_b, p_c\}$

Por tanto,  $\text{clau}(q_0) = \{q_0, q_a, q_b, q_c, p_a, p_b, p_c\}$ . De la misma forma, se puede deducir que para  $i = a, b, c$ ,  $\text{clau}(q_i) = \{q_i, p_i\}$  y  $\text{clau}(p_i) = \{p_i\}$ . Por último, es fácil verificar que  $\text{clau}(q_f) = \{q_f\}$ .

**Definición L(A).** Sea  $A$  un AFND- $\varepsilon$ . Si  $q$  es un estado y además  $x$  y  $a$  son, respectivamente, una cadena y un símbolo de entrada, la *función de transición extendida* se define inductivamente como sigue:

- R. INICIAL  $\hat{\delta}(q, \varepsilon) = \text{clau}(q);$
- R. INDUCTIVA si  $\hat{\delta}(q, x) = \{p_1, \dots, p_m\}$  entonces:

$$\hat{\delta}(q, xa) = \bigcup_{j=1}^k \text{clau}(r_j),$$

$$\text{donde } \bigcup_{i=1}^m \delta(p_i, a) = \{r_1, \dots, r_k\}.$$

Si  $\hat{\delta}(q, x) = \emptyset$ , se define  $\hat{\delta}(q, xa) = \emptyset$ . El *lenguaje aceptado* por  $A$  es el conjunto:

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

También, como se ha de esperar, en este caso se cumple la propiedad 3 para AFND- $\varepsilon$ . Es decir, para todo estado  $q$  y cadena de símbolos de entrada  $w$ ,  $\hat{\delta}(q, w)$  es el conjunto de estados  $r \in Q$  tales que  $(q, w) \xrightarrow{*} (r, \varepsilon)$  [ejercicio 47]. Por tanto, también en el caso de los AFND- $\varepsilon$  se tiene que  $L(A)$  es el conjunto de cadenas  $w \in \Sigma^*$  tales que  $(q, w) \xrightarrow{*} (p, \varepsilon)$  donde  $p$  es un estado de aceptación.

**Ejemplo 20.** Considere el autómata de la figura 3.2. Para calcular  $\hat{\delta}(q_0, 100)$  siguiendo la definición formal se procede como a continuación. Se tiene por supuesto que  $\hat{\delta}(q_0, \varepsilon) = \text{clau}(q_0) = \{q_0, q_1, p_1\}$ . Los pasos de lectura de la cadena 100 siguiendo la regla inductiva se explican en la siguiente tabla:

$a$	$\bigcup_{i=1}^m \delta(p_i, a)$	$\hat{\delta}(q_0, x) = \{p_1, \dots, p_m\}$
		$\hat{\delta}(q_0, \varepsilon) = \{q_0, q_1, p_1\}$
1	$\{q_0, q_2\}$	$\hat{\delta}(q_0, \varepsilon 1) = \{q_0, q_1, p_1, q_2\}$
0	$\{q_0, p_2\}$	$\hat{\delta}(q_0, \varepsilon 10) = \{q_0, q_1, p_1, p_2\}$
0	$\{q_0, p_2, p_3\}$	$\hat{\delta}(q_0, \varepsilon 100) = \{q_0, q_1, p_1, p_2, p_3\}$

Por tanto, la cadena 100 sería aceptada, pues el estado de aceptación  $p_3$  pertenece a  $\hat{\delta}(q_0, \varepsilon 100)$ . Se ha de notar que a diferencia de los autómatas sin transiciones instantáneas  $|\delta(q_0, 1)| \neq |\hat{\delta}(q_0, 1)|$ , pues  $\delta(q_0, 1) = \{q_0\}$  y sin embargo la tabla anterior indica que  $\hat{\delta}(q_0, 1) = \{q_0, q_1, p_1, q_2\}$ .

**Ejemplo 21.** Si  $A$  el AFND- $\varepsilon$  del ejemplo 18 entonces  $L(A)$  es el conjunto de cadenas binarias que acaban en 00 o en 11. Note que  $(q_0, x) \vdash^* (q_0, \varepsilon)$ , para toda cadena binaria  $x$ , entonces existen los siguientes recorridos para las cadenas de la forma  $x11$  y  $x00$ :

$$(q_0, x11) \vdash^* (q_0, 11) \vdash (q_1, 11) \vdash (q_2, 1) \vdash (q_3, \varepsilon),$$

$$(q_0, x00) \vdash^* (q_0, 00) \vdash (p_1, 00) \vdash (p_2, 0) \vdash (p_3, \varepsilon).$$

Para ver que son el único tipo de cadenas aceptadas, se supone lo contrario, es decir se aceptan cadenas que acaban 10 o 01. Si aceptaran cadenas del tipo  $w = x10$ , al hacer el cálculo preciso se llegaría a que  $\hat{\delta}(q_0, x10)$  no tiene algún estado de aceptación, independientemente de la cadena  $x$ , lo que lleva a una contradicción. De forma similar se razona para  $w = x01$ .

### Lista de ejercicios de la sección 3.2

*Ejercicio 45.* Sea  $L = \{ab, cd\}$  un lenguaje del alfabeto  $\{a, b, c, d\}$ . Diseñe un AFND- $\varepsilon$  que acepte a  $L^*$  como lenguaje.

*Ejercicio 46.* Diseñe un AFND- $\varepsilon$  que admita un recorrido infinito a partir de  $(q_0, w)$  para alguna cadena  $w$ . ¿Es posible hacer esto si el autómata no tiene transiciones instantáneas?

*Ejercicio 47.* Demuestre por inducción sobre la longitud de  $w$  la propiedad 3 pero suponiendo que el autómata admite transiciones instantáneas. Note que el caso  $w = \varepsilon$  es de hecho la propiedad 4.

*Ejercicio 48.* Sea  $\delta$  la función de transición de un AFND. Verifique que para todo estado  $q$  y símbolo de entrada  $a$ ,  $\hat{\delta}(q, a) = \delta(q, a)$ . De un ejemplo de que esto no ocurre con los AFND- $\varepsilon$ .

*Ejercicio 49.* Sea  $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$  y sea  $L = D^* \setminus \{\varepsilon\}$ . Diseñe un AFND- $\varepsilon$  que acepte como lenguaje al conjunto constituido por los siguientes valores numéricos y «símbolos»:

1. todas las cadenas en  $L$ ;
2. las cadenas de la siguiente forma, donde  $x, y$  y  $z$  están en  $L$ :  

$$\begin{array}{ccccc} -x.y & -x.yez & -xez & -x.ye-z & -xe-z \\ x.y & x.yez & xez & x.ye-z & xe-z \end{array}$$
3. las cadenas  $-\text{Inf}$  y  $+\text{Inf}$  (los dos infinitos); y
4. la cadena  $\text{NaN}$  (de *not a number* en inglés).

*Ejercicio 50.* Sea  $A$  el AFND- $\varepsilon$  del ejemplo 19. Demuestre que  $L(A)$  es el conjunto de cadenas cuya última entrada aparece a lo más dos veces.

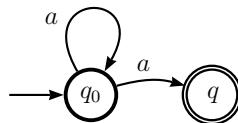
*Ejercicio 51.* Sean  $p$  y  $q$  dos estados de un AFND- $\varepsilon$ . Verifique que si  $p \in \text{clau}(q)$  entonces  $\text{clau}(p) \subset \text{clau}(q)$ .

*Ejercicio 52.* Demuestre el ejercicio 26 suponiendo que el autómata  $A$  es un AFND- $\varepsilon$ . Pruebe además que para un AFND- $\varepsilon$ , si  $(q, xw) \vdash^* (p, yw)$  para *alguna* cadena de símbolos de entrada  $w$ , entonces  $(q, x) \vdash^* (p, y)$ . Sugerencia: razoné por inducción sobre los movimientos respecto a  $\vdash^*$ .

### 3.3 Equivalencia de los autómatas finitos

El objetivo de esta sección es demostrar que ni el concepto de no-determinismo ni el de transiciones instantáneas implican la existencia de una clase de lenguajes más general a la clase de lenguajes aceptados por los autómatas finitos deterministas. En otras palabras, las tres máquinas computacionales son equivalentes.

En primer lugar, dado un AFND  $A_N$  siempre se puede encontrar un AFD  $A_D$  que acepte el mismo lenguaje que  $A_N$ . Por ejemplo, considere el autómata:



La transición no-determinista  $\delta_N(q_0, a) = \{q_0, q\}$  se puede pensar como determinista si le damos al conjunto  $\{q_0, q\}$  el significado de la situación «estoy en  $q$  ó en  $r$ ». Es decir, el conjunto  $\{q_0, q\}$  a su vez es un estado. Estando en el estado  $\{q_0, q\}$  de  $A_D$  si recibo  $a$ , es natural pasar al estado dado por el conjunto:

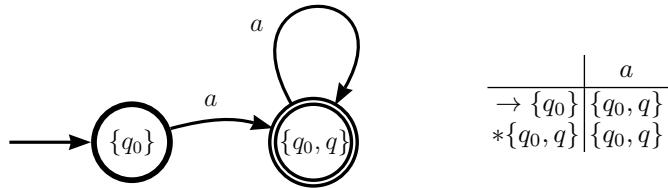
$$\delta_N(q_0, a) \cup \delta_N(q, a).$$

Es decir, el estado en  $A_D$  que representa a todos los posibles estados de llegada de  $A_N$  partiendo de  $q_0$  ó  $q$  al recibir  $a$ . La idea, por tanto, es que los estados de  $A_D$  sean cada uno de los subconjuntos de  $Q_N$ , pero habrá muchos que sean estados inaccesibles. Para evitar hacer el cálculo de  $\delta_D$  para estados inaccesibles, se puede proceder a su cálculo de forma inductiva. Claramente el estado inicial de  $A_D$  es  $\{q_0\}$ . A partir de ahí se encuentra  $\delta_D(\{q_0\}, a) = \{q_0, q\}$ . Por tanto, el

estado  $\{q_0, q\}$  es accesible en  $A_D$ . A su vez,  $\delta_D(\{q_0, q\}, a) = \{q_0, q\}$ . Como este estado ya estaba considerado antes, se concluye el proceso inductivo:

$$\begin{array}{c|c} \xrightarrow{\quad} \{q_0\} & | \\ a & | \\ \hline \end{array} \quad \begin{array}{c|c} \xrightarrow{\quad} \{q_0\} & | \\ \{q_0, q\} & | \\ a & | \\ \hline \end{array} \quad \begin{array}{c|c} \xrightarrow{\quad} \{q_0\} & | \\ \{q_0, q\} & | \\ \{q_0, q\} & | \\ a & | \\ \hline \end{array} \quad \begin{array}{c|c} \xrightarrow{\quad} \{q_0\} & | \\ \{q_0, q\} & | \\ \{q_0, q\} & | \\ a & | \\ \hline \end{array}$$

Finalmente, un estado de  $A_D$  es de aceptación si contiene al menos un estado de aceptación de  $A_N$ . Por tanto, el  $A_D$  que se buscaba es:



El procedimiento general se establece de la siguiente manera.

**Algoritmo AFND→AFD.** Sea  $A_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$  un AFND. Se construye un AFD  $A_D = (Q_D, \Sigma, \delta_D, p_0, F_D)$  tal que  $L(A_N) = L(A_D)$  como sigue:

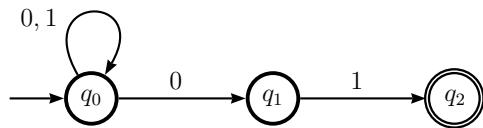
- R. INICIAL  $p_0 = \{q_0\} \in Q_D$  es el estado inicial.
- R. INDUCTIVA para cada  $S \in Q_D$  y símbolo de entrada  $a$ :

$$\delta_D(S, a) = \bigcup_{q \in S} \delta_N(q, a) \in Q_D.$$

Además,  $F_D = \{S \in Q_D : S \cap F_N \neq \emptyset\}$ .

(3.1)

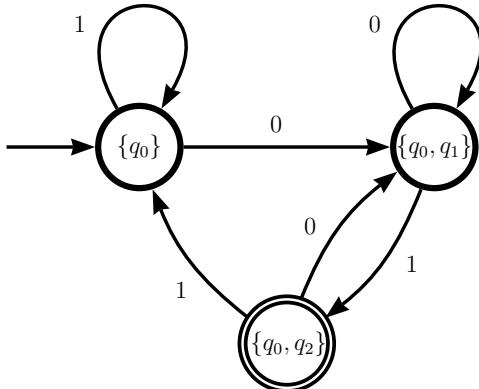
**Ejemplo 22.** Sea  $A_N$  el AFND descrito mediante el siguiente diagrama de transición:



Al proceder según (3.1) se obtiene la siguiente tabla de transición:

	0	1
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

Si  $p_0 = \{q_0\}$ ,  $p_1 = \{q_0, q_1\}$  y  $p_2 = \{q_0, q_2\}$ , el diagrama de transición correspondiente a  $A_D$  es:



En este ejemplo y el anterior, el número de estados de  $A_D$  es igual al de  $A_N$ . Sin embargo, es posible que la construcción nos deje como saldo un crecimiento enorme en el número de estados pudiendo llegar a  $2^{|Q_N|}$ .

Por otro lado, a partir de un AFND- $\varepsilon$   $A_E$  se puede construir similarmente un AFD  $A_D$  que acepte el mismo lenguaje que  $A_E$ . Con este fin y siguiendo la filosofía del procedimiento anterior, es necesario considerar a la *clausura* de un conjunto —estado del determinista—  $S$ , naturalmente definida por:

$$\text{clau}(S) = \bigcup_{p \in S} \text{clau}(p).$$

**Algoritmo AFND- $\varepsilon$ →AFD.** Sea  $A_E = (Q_E, \Sigma, \delta_E, q_0, F_E)$  un AFND- $\varepsilon$ . Se construye  $A_D = (Q_D, \Sigma, \delta_D, p_0, F_D)$  un AFD tal que  $L(A_E) = L(A_D)$  como sigue:

- R. INICIAL clau( $q_0$ )  $\in Q_D$  es el estado inicial.
- R. INDUCTIVA para cada  $S \in Q_D$  y símbolo de entrada  $a$ :

$$\delta_D(S, a) = \text{clau}(S_a) \in Q_D.$$

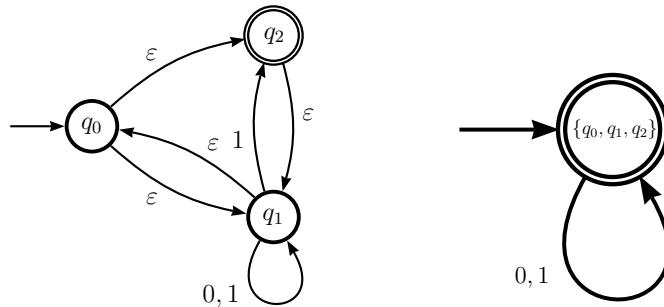
donde

$$S_a = \bigcup_{p \in S} \delta_E(p, a)$$

Finalmente,  $F_D = \{S \in Q_D : S \cap F_E \neq \emptyset\}$ .

(3.2)

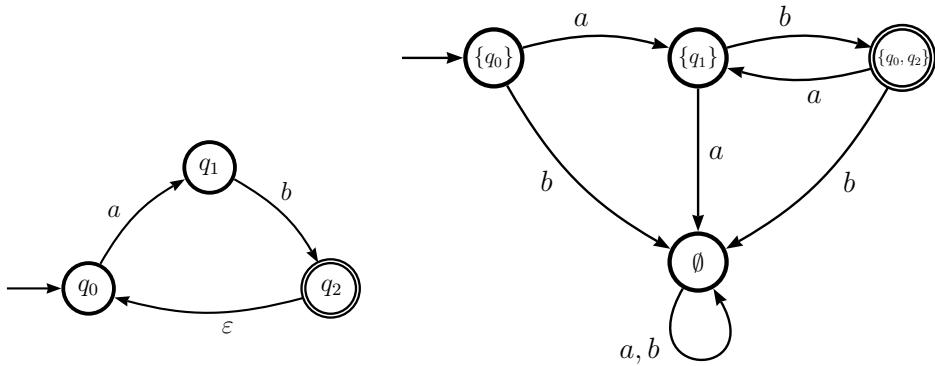
**Ejemplo 23.** El algoritmo (3.2) transforma el AFND- $\varepsilon$  de la izquierda, en el AFD de la derecha en una sola iteración.



El cálculo se detalla en la siguiente tabla:

$S$	$\bigcup_{p \in S} \delta_E(p, 0)$	$\text{clau}(S_0)$	$\bigcup_{p \in S} \delta_E(p, 1)$	$\text{clau}(S_1)$
$* \rightarrow \{q_0, q_1, q_2\}$	$\{q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

**Ejemplo 24.** En el siguiente ejemplo se ilustra el caso en que un estado de  $A_D$  sea el conjunto vacío. El AFD de la derecha es el resultado de aplicarle al AFND- $\varepsilon$  de la izquierda el algoritmo (3.2). Los detalles se muestran en la tabla de abajo.



$S$	$\left  \bigcup_{p \in S} \delta_E(p, a) \right $	$\text{clau}(S_a)$	$\left  \bigcup_{p \in S} \delta_E(p, b) \right $	$\text{clau}(S_b)$
$\rightarrow \{q_0\}$	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
$\{q_1\}$	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_0, q_2\}$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$*\{q_0, q_2\}$	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$

Hasta el momento, se ha visto como eliminar el no-determinismo en un autómata finito. Para cerrar el círculo, son suficientes las siguientes observaciones, ver figura 3:

- Un AFD  $A_D$  con función de transición  $\delta_D$  se puede transformar trivialmente en un AFND  $A_N$  que acepta a  $L(A_D)$  como lenguaje, modificando únicamente su función de transición por [ejercicio 54]:

$$\delta_N(q, a) = \{\delta_D(q, a)\}.$$

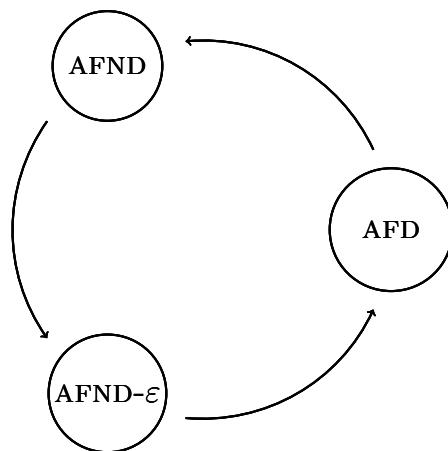
- De igual manera, un AFND  $A_N$  se puede ver fácilmente como un autómata con transiciones instantáneas  $A_E$ , sin alterar el lenguaje de aceptación, definiendo

para cualquier  $\delta_E(q, \varepsilon) = \emptyset$  [ejercicio 55]. En particular, el algoritmo (3.1) es un caso particular del algoritmo (3.2).

- Finalmente, si  $A_E$  es un AFND- $\varepsilon$  y  $A_D$  es el AFD de la construcción (3.2), entonces para todo  $n \geq 0$  [ejercicio 56],

$$P(n) : \text{para } |w| = n, \hat{\delta}_D(\text{clau}(q_0), w) = \hat{\delta}_E(q_0, w). \quad (3.3)$$

Por tanto,  $L(A_E) = L(A_D)$ . Por tanto, el algoritmo (3.2) cumple exitosamente su objetivo, y por consecuencia también el algoritmo (3.1).



**Fig. 3.3** Equivalencia entre autómatas finitos.

En conclusión:

**Teorema 1.** *Sea  $L$  un lenguaje cualquiera. Las siguientes afirmaciones son equivalentes:*

- existe un AFD que acepta a  $L$  como lenguaje;
- existe un AFND que acepta a  $L$  como lenguaje;
- existe un AFND- $\varepsilon$  que acepta a  $L$  como lenguaje.

### **Lista de ejercicios de la sección 3.3**

*Ejercicio 53.* Transforme en AFD los autómatas de los ejemplos 16, 17, 18 y 19.

*Ejercicio 54.* Sea  $A_D = (Q, \Sigma, \delta_D, q_0, F)$  un AFD que acepta a  $L$  como lenguaje. Consideré el AFND  $A_N = (Q, \Sigma, \delta_N, q_0, F)$  donde para cada estado  $q$  y símbolo de entrada  $a$ ,  $\delta_N(q, a) = \{\hat{\delta}_D(q, a)\}$ . Verifique que

$$L(A_N) = L(A_D).$$

Sugerencia: pruebe por inducción sobre la longitud de  $w$ , que para toda cadena de símbolos de entrada  $w$ ,  $\hat{\delta}_N(q_0, w) = \{\hat{\delta}_D(q_0, w)\}$ .

*Ejercicio 55.* Sea  $A_N = (Q, \Sigma, \delta_E, q_0, F)$  un AFND. Consideré también el AFND- $\varepsilon$   $A_E = (Q, \Sigma, \delta_E, q_0, F)$  con función de transición definida para cada  $q \in Q$  por:

$$\begin{aligned}\delta_E(q, a) &= \delta_N(q, a), \text{ si } a \in \Sigma; \\ \delta_E(q, \varepsilon) &= \emptyset.\end{aligned}$$

Verifique que  $L(A_E) = L(A_N)$ .

*Ejercicio 56.* Demuestre por inducción la afirmación (3.3).

*Ejercicio 57.* Sea  $Q$  el conjunto de estados de un AFND- $\varepsilon$ . Demuestre que para cualquier subconjunto  $S$  de  $Q$ :

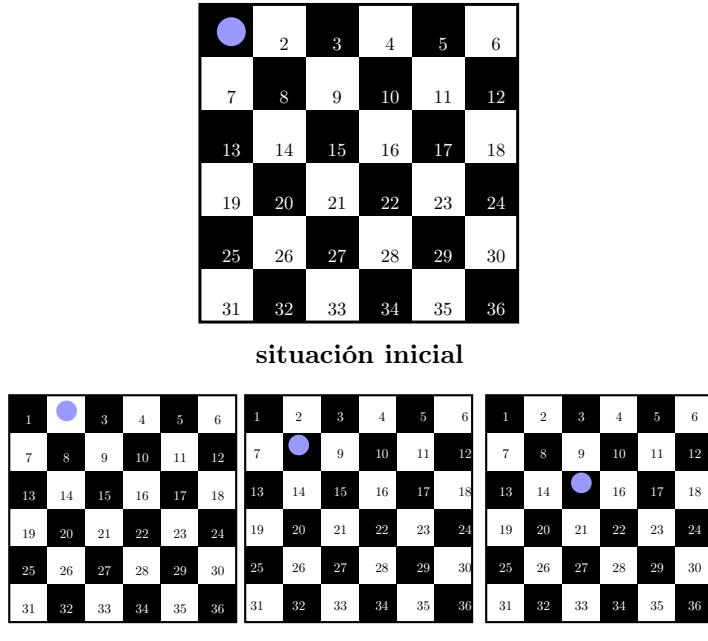
$$\text{clau}(\text{clau}(S)) = \text{clau}(S).$$

*Ejercicio 58.* Considere los siguientes movimientos permitidos de una ficha en un tablero de 36 casillas numeradas:

**b** moverse a la casilla blanca adyacente.

**n** moverse a la casilla negra adyacente.

Se cuentan las adyacentes en diagonal. Por ejemplo una ficha desde la casilla 1, se puede mover hasta a la casilla 15 ejecutando la secuencia de movimientos permitidos **b**, **n**, **n**:



situación inicial

casilla 1 → casilla 2 → casilla 8 → casilla 15

Cada secuencia de movimientos permitidos, puede arrojar muchas otras jugadas. Por ejemplo, la misma secuencia **b, n, n** nos puede llevar a realizar los siguientes cambios de casilla:

casilla 1 → casilla 7 → casilla 13 → casilla 20

Diseñe un AFD (sí: ¡determinista!) que acepte como lenguaje al conjunto de cadenas que representen secuencias finitas de movimientos permitidos tales que exista la posibilidad de llegar a la casilla 36 a partir de la casilla 1. Por ejemplo, con la secuencia **b, b, b, b, b, b** es imposible llegar a la casilla 36 pese a las múltiples opciones de movimiento.

Sugerencia: genere los estados con la misma filosofía con la que se establecieron los estados del autómata determinista del algoritmo (3.1).



# CAPÍTULO 4

---

## Lenguajes regulares

---

En este capítulo se presenta el concepto de expresión regular (ER), el cual fue introducido por S. C. Kleene [14] en los años cincuenta con el fin de describir eventos que pueden representarse mediante modelos de autómatas finitos. Desde entonces hasta nuestros días, las expresiones regulares y extensiones de ellas han sido utilizadas en muchas herramientas de Unix o lenguajes de programación como Perl y Python [1, p. 189].

Este capítulo se divide en tres secciones. En la primera sección se definen las expresiones y los lenguajes regulares. En la segunda y la tercera sección se demuestra que un lenguaje es regular si y solo si es aceptado por un ADF. Estas ideas de equivalencia pertenecen también a Kleene, pero la construcción de un AFD a partir de una ER que se presenta aquí corresponde a [16].

### 4.1 Expresiones regulares

Una ER es una cadena de símbolos que a su vez representa a un conjunto de cadenas. Por ejemplo, en Unix la cadena  $ab^*$  es una notación corta para el conjunto:

$$\{a\}\{b\}^* = \{a, ab, abb, \dots\}.$$

Otro ejemplo es la expresión  $baca|vaca$  la cual significa **baca** o **vaca**; es decir la expresión **baca|vaca** también en este caso se puede entender como una notación

simplificada para el conjunto  $\{\text{baca}, \text{vaca}\}$ . En general, las expresiones regulares se forman a partir de expresiones primitivas simples mediante las operaciones de concatenación, unión y clausura. La definición matemática se establece de forma inductiva, a partir de un alfabeto dado:

**Definición ER.** Dado un alfabeto  $\Sigma$ , se genera el conjunto de *expresiones regulares* asociada a  $\Sigma$  de manera inductiva a través de las siguientes reglas:

- R. INICIAL los símbolos  $\varepsilon$  y  $\emptyset$  son expresiones regulares,  
si  $a \in \Sigma$  entonces  $a$  es una expresión regular.
- R. INDUCTIVA si  $E$  y  $F$  son expresiones regulares entonces  
también lo son:

$$E|F, EF, (E) \text{ y } E^*.$$

**Ejemplo 25.** Las expresiones  $(00|11)^*$ ,  $00^*(0|1)^*11$  son expresiones regulares del alfabeto binario. La expresión  $1|$  no es regular.

El conjunto (lenguaje) que representa una expresión regular también se define de forma inductiva a partir de las operaciones concatenación, unión y clausura de lenguajes que se estudiaron en el capítulo 1.

**Definición  $L(E)$ .** Si  $E$  es una expresión regular, se denota por  $L(E)$  al *lenguaje aceptado por  $E$* . El lenguaje aceptado por cada expresión regular asociada a un alfabeto se establece también inductivamente:

- R. INICIAL  $L(\varepsilon) = \{\varepsilon\}$ ,  $L(\emptyset) = \emptyset$  y  $L(a) = \{a\}$ ;
- R. INDUCTIVA si  $E$  y  $F$  son expresiones regulares,

$$L(E|F) = L(E) \cup L(F),$$

$$L(EF) = L(E)L(F),$$

$$L((E)) = L(E),$$

$$L(E^*) = L(E)^*.$$

Se dice que un lenguaje  $L \subset \Sigma^*$  es un *lenguaje regular* si existe una expresión regular  $E$  tal que  $L(E) = L$ .

Para evitar exceso en el uso de paréntesis se toman en cuenta las siguientes convenciones:

- el operador  $*$  tiene la mayor precedencia y es asociativo a la izquierda,
- la concatenación tiene la segunda precedencia y es asociativa a la izquierda,
- la unión  $|$  tiene la menor precedencia y es asociativa a la izquierda.

**Ejemplo 26.**  $L(aa^*) = \{a, aa, aaa, aaaa, \dots\}$ . Además,  $L((0|1)*11)$  es el conjunto de cadenas binarias que acaban en 11.

**Ejemplo 27.** Se desea encontrar una expresión regular  $E$  sobre el alfabeto binario, tal que  $L(E) = \{w : w \text{ consta de ceros y unos alternados}\}$ . La expresión regular  $01(01)^*$  acepta como lenguaje al conjunto de cadenas con ceros y unos alternados, pero solo los que comienzan en 0 y terminan en 1, en otras palabras:

$$L(01(01)^*) = \{01, 0101, 010101, \dots\}.$$

Para obtener todas las cadenas de ceros y unos alternados es preciso considerar la expresión regular que incluya las otras tres posibilidades:

$$01(01)^*|10(10)^*|010(10)^*|101(01)^*.$$

Por supuesto, pueden existir dos expresiones regulares  $E$  y  $F$  que no son exactamente iguales (mismos símbolos, en el mismo orden) que acepten el mismo lenguaje. En este sentido, las expresiones en cuestión se pueden considerar una sola:

**Definición  $E=F$ .** Se dice que dos expresiones regulares  $E$  y  $F$  de un mismo alfabeto son *equivalentes* —se escribe  $E = F$ — si

$$L(E) = L(F).$$

Una propiedad de las expresiones regulares es que se puede hacer cierta álgebra con ellas a partir de la noción de equivalencia. En el cuadro 4.1 se presentan algunas equivalencias básicas, —llamadas axiomas de Salomaa—. Acompañados a estos «axiomas» vienen las siguientes *reglas de inferencia*:

**SUBSTITUCIÓN:** Sea  $E$  un ER que contiene a la ER  $J$  como subcadena. Sea  $G$  la ER que resulta de substituir cada aparición de  $J$  por otra ER  $I$ . Si  $I = J$  y  $E = H$  entonces se puede inferir que  $G = H$  y  $G = E$ .

**SOLUCIÓN DE ECUACIONES:** Sea  $F$  tal que  $\varepsilon \notin L(F)$ . Entonces de la ecuación  $E = EF|H$  se puede inferir la solución  $E = HF^*$ , c. f. ejercicio 11.

Los axiomas de Salomaa son fácilmente verificables a partir de las propiedades algebraicas de los lenguajes vistos en el primer capítulo [ejercicio 61]. Además tanto la regla de sustitución como la regla de solución de ecuaciones infieren equivalencias ciertas. Más aún, un hecho no trivial es que cualquier equivalencia cierta entre dos expresiones regulares se deduce (en una cantidad finita de equivalencias intermedias) a partir de los axiomas de Salomaa y las reglas de inferencia [21].

$E (F H) = (E F) H$ $E F = F E$ $E E = E$ $(E F)H = EH FH$ $E^* = \varepsilon E^*E$	$E(FH) = (EF)H$ $\varepsilon E = E$ $\emptyset E = \emptyset$ $E(F H) = EF EH$ $E^* = (\varepsilon E)^*$ $E \emptyset = E$
---	---

**Cuadro 4.1** Axiomas de Salomaa.

**Ejemplo 28.** La ER  $0|01^*$  es equivalente a la ER  $01^*$ . Expresando los lenguajes de aceptación en forma explícita, es fácil darse cuenta que ambas aceptan el lenguaje  $\{0, 01, 011, 0111, \dots\}$ . Sin embargo, esta equivalencia también se puede verificar a partir de los axiomas de Salomaa y las reglas de inferencia de forma puramente algebraica:

$$\begin{aligned} 1^* &= \varepsilon|1^*1 \\ &= (\varepsilon|\varepsilon)|1^*1 \\ &= \varepsilon|(\varepsilon|1^*1) \\ &= \varepsilon|1^* \end{aligned}$$

Por tanto,  $01^* = 0(\varepsilon|1^*) = 0\varepsilon|01^* = 0|01^*$ . Ya que  $0\varepsilon = 0$  [ejercicio 63] finalmente se tiene que  $01^* = 0|01^*$ .

**Ejemplo 29.** Unix tiene su propia notación para el manejo de expresiones regulares, además, desde un inicio se han ido agregado *expresiones regulares extendidas*. Algunas son:

$E^+$  es la expresión regular extendida equivalente a  $EE^*$ . El operador  $+$  tiene la misma precedencia y asociatividad que el operador clausura.

$E^?$  es la expresión regular extendida equivalente a  $E|\varepsilon$ . También tiene la misma precedencia y asociatividad que el operador clausura.

$[abc]$  es la expresión regular extendida equivalente a  $a|b|c$ .

$[a-z]$  es la expresión regular extendida equivalente a  $a|b|c|\dots|z$ . Esto aplica siempre y cuando se trate de secuencias lógicas como letras mayúsculas o dígitos.

Por ejemplo, la expresión regular extendida  $[A-Za-z]$  acepta como lenguaje al conjunto de todas las letras en inglés, mayúsculas y minúsculas. La expresión regular  $bo?ulevard$  es equivalente a la expresión regular  $boulevard|bulevard$ . *Advertencia:* las expresiones regulares son fundamentalmente distintas de los conceptos de `regex` o `regexp` con las que se suelen denotar a las expresiones regulares extendidas en Unix, Python, Perl, entre otros. Estas nociones aceptan un conjunto de lenguajes más amplio que las expresiones regulares. De hecho, muchas expresiones que se encuentran en casi todas las librerías de este tipo proporcionan una potencia expresiva muy superior a la de los lenguajes regulares. Por ejemplo la `regex`  $([01]^*)\1$  acepta como lenguaje el conjunto de cadenas binarias de la forma  $ww$  el cual no es regular [ejercicio 75].

### Listado de ejercicios de la sección 4.1

*Ejercicio 59.* Para cada uno de los siguientes incisos, encuentre una ER que acepte el lenguaje formado por el conjunto de cadenas binarias tales que:

1. comienzan en 010 y acaban en 11;
2. tienen longitud  $3n$ ,  $n > 0$ ;
3. son de la forma  $0^n1$  o  $01^n$ ,  $n > 0$ .

*Ejercicio 60.* Verifica los axiomas de Salomaa.

*Ejercicio 61.* Prueba que si  $L(E) \subset L(F)$  para dos ER  $E$  y  $F$ , entonces  $E|F = F$ .

*Ejercicio 62.* Verifique:

1.  $a|a(\varepsilon|aa)^*(\varepsilon|aa) = a(aa)^*$ ,
2.  $(0+1)^*01(\emptyset+\emptyset)(0+1)^*01)^* = (0+1)^*01$ .

*Ejercicio 63.* A partir de las reglas de inferencia y los axiomas de Salomaa, prueba que  $E\varepsilon = E$  y que  $E\emptyset = \emptyset$  para cualquier ER  $E$ .

*Ejercicio 64.* Suponga que está trabajando en un lenguaje que no es sensible a las mayúsculas y minúsculas. Escriba una expresión regular en la notación de Unix descrita en el ejemplo 29 que acepte como lenguaje cualquier texto que contenga a la palabra reservada `then` en cualquiera de sus formas: `Then`, `THen`, `then`, etcétera.

*Ejercicio 65.* Busque en Internet sobre las `regex` en Python. ¿Qué tipo de cadenas representa la siguiente:

$$(0[1-9] | [12] [0-9] | 3[01])([- / .])(0[1-9] | 1[012]) \backslash 2((?:19|20)\d\d) ?$$

*Ejercicio 66.* Para cada una de las siguientes ER encuentre el lenguaje que acepta:

1.  $c(a^*b^*)^*$ ,
2.  $(a|b)^*ab(\emptyset|\emptyset(a|b)^*ab)^*c$ ,
3.  $a^*(c|\emptyset|cb^*)|(a^*)^*cb^*$ .

*Ejercicio 67.* Entre a <https://regexcrossword.com/>. Inscríbase. Juegue.

*Ejercicio 68.* Trate de resolver en línea el rompecabezas hexagonal <http://rampion.github.io/RegHex/>. Este *hexagonal puzzle* es una parte del MIT Mystery Hunt del año 2013 —el cual incluye muchos problemas bastante difíciles. ¡Vea qué tan lejos llega! Nota: si se rompe mucho la cabeza puede consultar la respuesta en [http://www.mit.edu/~puzzle/2013/coinheist.com/rubik/a\\_regular\\_crossword/answer/index.html](http://www.mit.edu/~puzzle/2013/coinheist.com/rubik/a_regular_crossword/answer/index.html). Comprueba que es correcta.

## 4.2 De autómatas finitos a expresiones regulares

En esta sección se demuestra que a partir de un autómata finito determinista se puede construir una expresión regular que acepte el mismo lenguaje. El procedimiento que se presenta se conoce en la literatura como *método de la clausura transitiva o algoritmo de Keene*. Este método tiene importancia teórica por sí mismo y se puede establecer con las herramientas vistas hasta el momento, pero no es particularmente bueno en la práctica pues como se verá en la demostración del teorema 2, puede arrojar expresiones regulares del orden de  $4^n$  términos si  $n$  es el número de estados del autómata dado.

Sea  $\{1, 2, \dots, n\}$  el conjunto de estados de  $A$ , donde 1 es el estado inicial. Se denota por  $R_{ij}^{(k)}$  a una ER que acepta al conjunto de cadenas leídas al recorrer del estado  $i$  al estado  $j$  en  $A$  por un camino con todos los estados intermedios  $\leq k$ ; los estados  $i$  y  $j$  por definición no son intermedios, por tanto no hay condición sobre ellos. La ER  $E$  que se busca es la «unión» de todas las ER  $R_{1j}^{(n)}$  donde  $j$  recorre a todos los estados de aceptación, esto es, si  $F = \{j_1, \dots, j_m\}$

entonces  $E = R_{1j_1}^{(n)} | R_{1j_2}^{(n)} | \dots | R_{1j_m}^{(n)}$ . La construcción de las ER  $R_{ij}^{(k)}$  se establece por inducción sobre  $k$ .

**REGLA INICIAL** sea  $k = 0$ , ya que no existen estados menores o iguales a 0, entonces los caminos que recorren las cadenas aceptadas por  $R_{ij}^{(0)}$  no tienen estados intermediarios. Por tanto, tales cadenas tienen longitud 0 o 1; o bien, no existe ninguna cadena aceptada por  $R_{ij}^{(0)}$  (no hay lazos entre  $i$  y  $j$ ). En el cuadro 4.2, se concretan los casos, se ha de notar que  $\varepsilon$  es aceptada si  $i = j$ , independientemente si hay arcos o no.

**REGLA INDUCTIVA** sea  $k \geq 1$ , hay solo dos posibles casos a considerar para las cadenas que constituyen al lenguaje aceptado por  $R_{ij}^{(k)}$ .

- El camino que recorre la cadena no pasa por el estado  $k$ , en este caso la cadena pertenece al lenguaje aceptado por  $R_{ij}^{(k-1)}$ .
- El camino que recorre la cadena pasa al menos una vez por el estado  $k$ . En este caso el camino comienza en  $i$  hasta que topa por primera vez con  $k$ , esa subcadena está en el lenguaje aceptado por  $R_{ik}^{(k-1)}$ . Luego la cadena inicial se recorre del estado  $k$  hasta que topa otra vez con  $k$ , de nuevo de  $k$  a  $k$ , y así sucesivamente hasta que llega por última vez en su recorrido al estado  $k$ , cada subcadena —leída de  $k$  hasta  $k$ — se encuentra en el lenguaje aceptado por  $R_{kk}^{(k-1)}$ , luego la concatenación de todas se encuentra en el lenguaje aceptado por  $(R_{kk}^{(k-1)})^*$ . Note que si la cadena inicial topa una sola vez en su recorrido con el estado  $k$ , la subcadena recorrida de  $k$  a  $k$  es la cadena vacía. Finalmente el recorrido va desde el estado  $k$  hasta el estado  $j$  y la subcadena correspondiente está en el lenguaje aceptado por  $R_{kj}^{(k-1)}$ . Por tanto, la cadena inicial pertenece al lenguaje aceptado por  $R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^*R_{kj}^{(k-1)}$ .

De los dos casos especificados en la regla inductiva, se llega a que:

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} | R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^*R_{kj}^{(k-1)}.$$

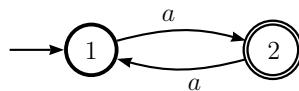
función de transición	$R_{ij}^{(0)}$	c. aceptadas
$\delta(i, a) = j, \forall a \in \{a_1, \dots, a_r : r > 0\}, i \neq j$	$a\_1   \dots   a\_r$	$a_1, a_2, \dots, a_r$
$\delta(i, a) = i, \forall a \in \{a_1, \dots, a_r : r > 0\}$	$\varepsilon   a\_1   \dots   a\_r$	$\varepsilon, a_1, \dots, a_r$
No hay arcos que unan a $i$ y a $j, i \neq j$	$\emptyset$	ninguna
No hay arcos que unan a $i$ y a $j, i = j$	$\varepsilon$	$\varepsilon$

**Cuadro 4.2** Relación de  $\delta$  con las expresiones  $R_{ij}$ .

El método de la clausura transitiva demuestra el siguiente:

**Teorema 2.** Si  $A$  es un AFD entonces existe una ER que acepta el mismo lenguaje que  $A$ .

**Ejemplo 30.** Considere el autómata de «prendido y apagado» del ejemplo 12 con  $F = \{q_p\}$ . Con el cambio de notación de arriba se tiene  $1 \leftarrow q_a$  y  $2 \leftarrow q_p$  y por tanto el autómata se ve de la siguiente forma:



En el siguiente cuadro se escribe el resultado de aplicar el algoritmo establecido en la prueba del teorema 2 para encontrar a la expresión regular  $R_{12}^{(2)}$ :

$k = 0$	$k = 1$	$k = 2$
$R_{11}^{(0)} = \varepsilon$	$R_{11}^{(1)} = \varepsilon$	$R_{11}^{(2)} = \varepsilon   a(\varepsilon   aa)*a$
$R_{12}^{(0)} = a$	$R_{12}^{(1)} = a$	$R_{12}^{(2)} = a   a(\varepsilon   aa)*(\varepsilon   aa)$
$R_{21}^{(0)} = a$	$R_{21}^{(1)} = a$	$R_{21}^{(2)} = a   (\varepsilon   aa)(\varepsilon   aa)*a$
$R_{22}^{(0)} = \varepsilon$	$R_{22}^{(1)} = \varepsilon   aa$	$R_{22}^{(2)} = (\varepsilon   aa)   (\varepsilon   aa)(\varepsilon   aa)*(\varepsilon   aa)$

Por tanto la expresión regular buscada es  $R_{12}^{(2)} = a | a(\varepsilon | aa)*(\varepsilon | aa) = a(aa)*$ .

### **Lista de ejercicios de la sección 4.2**

*Ejercicio 69.* Siguiendo el algoritmo que se establece en la prueba del teorema 2 encuentre una expresión regular para el autómata de la figura 2.5.

*Ejercicio 70.* Busque y estudie en la literatura otras opciones para transformar un AFD a una ER, por ejemplo:

1. El método algebraico de Brzozowski.
2. El algoritmo por eliminación de estados.
3. Usando el lema de Arden.

### 4.3 De expresiones regulares a autómatas finitos

En esta sección se estudia un algoritmo para determinar un autómata finito no-determinista con transiciones instantáneas a partir de una expresión regular. El procedimiento que se presenta a través de la prueba del siguiente teorema se conoce en la literatura como *la construcción de Thompson* o *algoritmo de McNaughton-Yamada*. Existen construcciones directas  $ER \rightarrow AFD$  pero se ha de echar mano de técnicas que por el momento están fuera del alcance de este texto.

**Teorema 3.** *Dada una ER se puede construir un AFND- $\varepsilon$  que acepta el mismo lenguaje.*

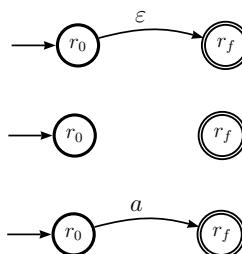
*Prueba.* La demostración se hace por inducción sobre la iteración en que se forman las expresiones regulares. Concretamente se prueba la siguiente afirmación.

$P(n)$ : dada una expresión regular que se construye en una iteración menor o igual a  $n$ , existe un autómata finito no-determinista con transiciones instantáneas que acepta su mismo lenguaje tal que:

- tiene exactamente un estado de aceptación,
- sin arcos que terminen en el estado inicial,
- sin arcos que salgan del estado de aceptación.

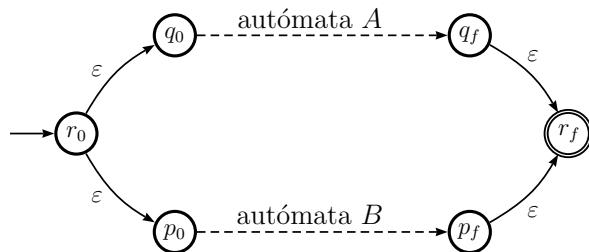
(4.1)

Las expresiones regulares que se forman en la iteración 1 son  $\varepsilon, \emptyset$  y  $a$ , si  $a \in \Sigma$ . Para probar  $P(1)$  se proponen, respectivamente, los siguientes autómatas:

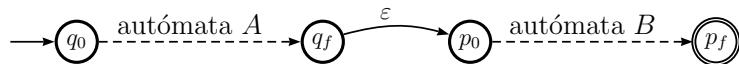


**Fig. 4.1** Autómatas que aceptan  $L(\varepsilon)$ ,  $L(\emptyset)$  y  $L(a)$ , respectivamente.

Sean  $A$  y  $B$  dos autómatas que satisfacen los tres puntos de  $P(n)$  y que aceptan, respectivamente, el mismo lenguaje que  $E$  y  $F$  (hipótesis de inducción). En la iteración  $n + 1$ , las expresiones regulares son de la forma  $E|F$ ,  $EF$ ,  $E^*$  y  $(E)$ , donde  $E$  y  $F$  son expresiones regulares formadas en una iteración menor o igual a  $n$ . Se consideran los autómatas descritos en las figuras 4.2, 4.3 y 4.4 para las primeras tres expresiones regulares. Finalmente, para la expresión regular  $(E)$ , el propio autómata  $A$  sirve. Con esto se prueba  $P(n + 1)$ .  $\square$

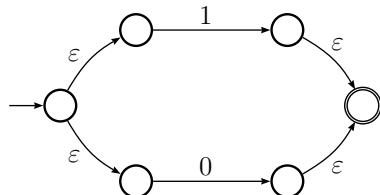


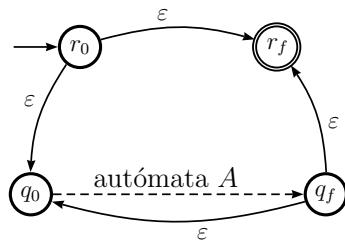
**Fig. 4.2** Autómata que acepta a  $L(A) \cup L(B)$ .



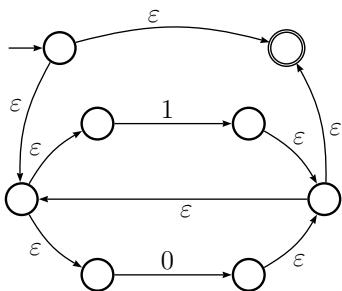
**Fig. 4.3** Autómata que acepta a  $L(A)L(B)$ .

**Ejemplo 31.** Sea  $E = (0|1)^*1$ . Según la construcción anterior, el autómata que acepta a  $L(0|1)$  como lenguaje es:

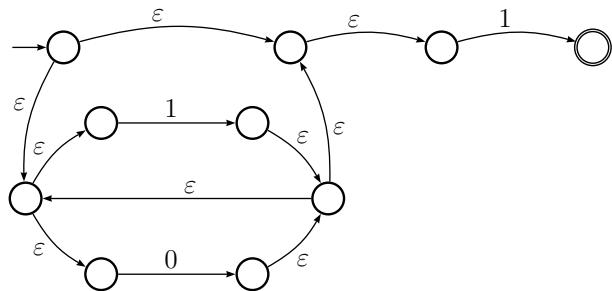


**Fig. 4.4** Autómata que acepta a  $L(A)^*$ .

A partir del autómata anterior se determina el autómata que acepta a  $L((0|1)*)$  como lenguaje:



Finalmente, el autómata que acepta a  $L(E)$  queda así:



### **Lista de ejercicios de la sección 4.3**

*Ejercicio 71.* Con el algoritmo descrito en esta sección, encuentre un autómata finito no-determinista con transiciones instantáneas que acepte el mismo lenguaje que la expresión regular  $0(10)^*|1(10)^*$ .

*Ejercicio 72.* En la demostración del teorema 3 se hace a través de la construcción de AFND- $\varepsilon$  con un solo estado de aceptación, sin arcos que terminen en el estado inicial y sin arcos que salgan del estado de aceptación. ¿Son necesarias estas condiciones?

## **4.4 Propiedades de los lenguajes regulares**

En esta sección se presentan dos caracterizaciones de los lenguajes regulares además de un criterio para verificar que un lenguaje no es regular conocido en la literatura como lema de bombeo para lenguajes regulares. La primera caracterización es el siguiente teorema el cual básicamente ya ha sido probado:

**Teorema 4.** *Sea  $L$  un lenguaje. Las siguientes afirmaciones son equivalentes:*

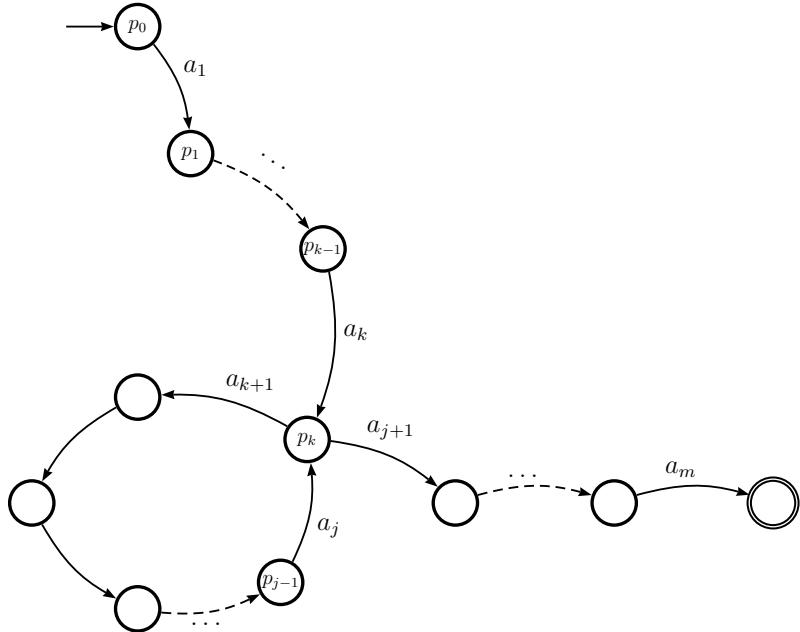
- $L$  es aceptado por un AFD.
- $L$  es un lenguaje regular.

*Prueba.* En la sección 4.2 se vio que si  $A$  era un autómata finito determinista que acepta a  $L$  como lenguaje, se puede construir una expresión regular  $E$  que también acepta a  $L$  como lenguaje. Por otro lado, en la sección 4.3 se vio que dada una expresión regular  $E$  que acepta a  $L$  como lenguaje, existe un autómata finito no-determinista con transiciones que también acepta a  $L$  como lenguaje. Por el teorema 1, existe entonces un autómata finito determinista que acepta a  $L$  como lenguaje.  $\square$

A continuación se verá como esta caracterización nos da una pauta para establecer un mecanismo para comprobar que un leguaje no es regular. Sea  $L$  un lenguaje regular, por el teorema 4 existe un AFD  $A = (Q, \Sigma, \delta, q_0, F)$  tal que  $L(A) = L$ . Sea  $n$  el número de estados de  $A$  y sea  $w = a_1a_2\cdots a_m$ , con  $m > n$  una cadena en  $L$ . Sea  $p_0 = q_0$  y

$$p_i = \hat{\delta}(q_0, a_1 \cdots a_i), \quad \text{para } i \geq 1.$$

Ya que  $A$  tiene un número  $n$  de estados, no es posible que los  $n + 1$  estados  $p_i$  sean todos diferentes. Por tanto existen índices  $k$  y  $j$  en  $\{0, 1, \dots, n\}$  tales que  $p_k = p_j$ , con  $k < j$ . Es decir, la lectura de la cadena  $w$  incluye un ciclo:



La cadena  $w$  se puede dividir naturalmente como  $w = xyz$ , donde:

$$\begin{aligned}x &= a_1 \cdots a_k \\y &= a_{k+1} \cdots a_j \\z &= a_{j+1} \cdots a_m.\end{aligned}$$

Se puede deducir que:

- La cadena  $y$  no puede ser vacía, pues  $k$  es diferente de  $j$ .
- La longitud de  $xy$  es menor o igual al número de estados.
- Para todo  $r \geq 0$ , la cadena  $xy^r z$  también está en  $L$ , pues el ciclo obliga que al finalizar la lectura de  $y$  a partir de  $p_k$  se regrese al mismo estado  $p_j = p_k$  y por tanto, para todo  $r \geq 0$ :

$$(p_0, xy^r z) \stackrel{*}{\vdash} (p_k, y^r z) \stackrel{*}{\vdash} (p_j, z) \stackrel{*}{\vdash} (\hat{\delta}(p_0, w), \varepsilon).$$

En resumen, se ha probado lo siguiente:

**Teorema 5 (lema de bombeo para lenguajes regulares).** *Sea  $L$  un lenguaje regular. Entonces existe un  $n > 0$ , tal que para toda cadena  $w \in L$  con  $|w| > n$  existe una descomposición de  $w$  de la forma  $w = xyz$  que satisface:*

- B<sub>1</sub>.  $y \neq \varepsilon$ ,*
- B<sub>2</sub>.  $|xy| \leq n$ ,*
- B<sub>3</sub>. para todo  $r \geq 0$ , la cadena  $xy^r z$  está en  $L$ .*

**Ejemplo 32.** Sea  $L_a = \{a^{2m+1} : m \geq 0\}$  el lenguaje asociado al autómata del ejemplo 12 con  $F = \{q_p\}$ . En este caso, la constante del lema de bombeo es  $n = 2$ . Para cada cadena  $w = a^{2m+1}$  con  $m \geq 0$ , la descomposición  $x = \varepsilon$ ,  $y = aa$  y  $z = a^{2(m-1)+1}$  satisface trivialmente  $B_1$ ,  $B_2$  y  $B_3$ .

**Observación.** El lema de bombeo es equivalente a la siguiente afirmación: si para cualquier  $n > 0$  existe una cadena  $w \in L$  con  $|w| > n$  tal que para cualquier descomposición de  $w$  al menos de una de las afirmaciones  $B_1$ ,  $B_2$  o  $B_3$  no se cumple, entonces  $L$  no es un lenguaje regular.

**Ejemplo 33.** A continuación se verifica que el lenguaje  $L_{01} = \{0^n 1^n : n > 0\}$  no es regular. Sea  $n$  un número natural y  $w = 0^n 1^n$ . Si se cumple  $B_2$ , entonces  $x$  y  $y$  contienen solo ceros y todos los unos están en  $z$ . Si también se satisface  $B_1$ ,  $y$  tiene que tener al menos un 0. Por tanto,  $xy^r z \notin L_{01}$  para  $r > 1$ , es decir, no se cumple  $B_3$ . Por tanto,  $L_{01}$  no es un lenguaje regular.

El lema de bombeo no caracteriza a los lenguajes regulares. Es decir, hay lenguajes que no son regulares para los cuales existe un  $n$  tal que para cualquier cadena  $w$  con  $|w| > n$  hay una descomposición de  $w$  que satisface las afirmaciones  $B_1$ ,  $B_2$  y  $B_3$  [ejercicio 76]. Sin embargo, una herramienta poderosa que caracteriza a los lenguajes regulares es el *teorema de Myhill-Nerode*. Sin estudiar a fondo este resultado, se explica brevemente en lo que resta de esta sección. Sea  $L$  un lenguaje de  $\Sigma$ , se dice que dos cadenas  $x$  y  $y$  de  $\Sigma^*$  son  $L$ -equivalentes, se escribe  $x \equiv y$ , si para toda cadena  $w \in \Sigma^*$ :

$$xw \in L \quad \text{si y solo si} \quad yw \in L.$$

Es fácil ver que  $\equiv$  define una relación de equivalencia en  $\Sigma^*$ . Esto es, para cualesquiera cadenas  $x, y$  y  $z$  en  $\Sigma^*$ :

- $x \equiv x$ ,
- $x \equiv y$  implica que  $y \equiv x$ ,
- si  $x \equiv z$  y  $z \equiv y$  entonces  $x \equiv y$ .

El *índice* de  $L$  se define como la cardinalidad del conjunto de clases de equivalencia en  $\Sigma^*$  respecto a  $\equiv$ , esto es, el índice de  $L$  es la cardinalidad del conjunto:

$$\Sigma^*/\equiv = \{[x] : x \in \Sigma^*\}$$

donde  $[x] = \{y \in \Sigma^* : x \equiv y\}$ .

**Teorema 6 (MyHill-Nerode).** *Un lenguaje es regular si y solo si su índice es finito*

**Ejemplo 34.** El lenguaje  $L_a = \{a^{2m+1} : m \geq 0\} \subset \{a\}^*$  del ejemplo 32 tiene índice 2. De hecho,  $\{a\}^*/\equiv = \{[a], [aa]\}$ . Para calcular  $[a]$  se deben encontrar todas las  $y \in \{a\}^*$  tales que para todo  $w \in \{a\}^*$ :

$$aw = a^{2k+1} \text{ para algún } k \geq 0 \quad \text{si y solo si} \quad yw = a^{2n+1} \text{ para algún } n \geq 0.$$

Por tanto,  $[a]$  es justamente el conjunto  $L_a$  y de igual forma se puede verificar que  $[aa]$  es el complemento de  $L_a$ .

**Ejemplo 35.** El índice del lenguaje  $L_{01} = \{0^n 1^n : n > 0\}$  del ejemplo 33 es  $\infty$ , ya que si  $n \neq m$  entonces  $0^n 1^n$  está en  $L_{01}$  pero  $0^m 1^n$  no está en  $L_{01}$ . Es decir, si  $n \neq m$  entonces  $[0^m] \neq [0^n]$  por tanto el conjunto  $\{0, 1\}^*/\equiv$  contiene al conjunto infinito  $\{[0^n] : n > 0\}$ .

#### Lista de ejercicios de la sección 4.4

*Ejercicio 73.* Sea  $L_{\text{son}} \subset \Sigma^*$  el lenguaje del ejemplo 10 y el ejemplo 13 donde  $\Sigma$  es el conjunto de símbolos admitidos en un texto plano.

1. Encuentre un  $n$  natural tal que para cualquier cadena  $w \in L_{\text{son}}$  con longitud mayor a  $n$  se cumplan  $B_1$ ,  $B_2$  y  $B_3$  del lema de bombeo.
2. Verifique que el índice de  $L_{\text{son}}$  es 7. Sugerencia: considere a la relación de  $L_{\text{son}}$ -equivalencia  $\equiv$ . Compruebe que

$$\Sigma^*/\equiv = \{[\varepsilon], [\text{s}], [\text{so}], [\text{son}], [\text{sono}], [\text{sonor}], [\text{sonora}]\}.$$

*Ejercicio 74.* Encuentre el índice del lenguaje de cadenas binarias con una cantidad impar de ceros y una cantidad impar de unos cf. ejemplo 14. Sugerencia: considere las clases de equivalencia  $[010]$ ,  $[101]$ ,  $[\varepsilon]$  y  $[01]$ .

*Ejercicio 75.* Demuestre que los siguientes lenguajes no son regulares utilizando el lema de bombeo.

1.  $L_{ww}$  conjunto de cadenas binarias de la forma  $ww$ .
2. El conjunto  $L_{[]}^{\square}$  de cadenas de corchetes equilibrados definido a través de las siguientes reglas:

R. INICIAL  $\varepsilon$  está en  $L_{[]}^{\square}$

R. INDUCTIVA si  $x$  y  $y$  están en  $L_{[]}^{\square}$  entonces también lo están  $[x]$  y  $xy$ .

3. El conjunto formado por cadenas de ceros y de unos con una cantidad desigual de ceros y de unos. Sugerencia: para cada  $n$  natural, considere la cadena  $w = 0^n 1^n 1^n!$ .

*Ejercicio 76.* Sea  $L = \{0^n 1^m 2^m : n \geq 1, m \geq 0\} \cup \{1^m 2^k : m \geq 0, k \geq 0\}$ . Verifique que para cualquier cadena  $w$  con  $|w| \geq 1$  existe una descomposición de  $w$  que satisface las afirmaciones  $B_1$ ,  $B_2$  y  $B_3$  del lema de bombeo. Sugerencia: descomponga a la cadena  $w$  en  $x = \varepsilon$ ,  $y$  el primer símbolo de  $w$  y  $z$  el resto. Compruebe que  $L$  no es regular usando el teorema de MyHill-Nerode.

*Ejercicio 77.* Pruebe que si  $L$  y  $M$  son lenguajes regulares, entonces también lo son los lenguajes  $L \cup M$ ,  $L \cap M$ ,  $L \setminus M$ ,  $LM$ , y  $L^*$ .

*Ejercicio 78. Problemas de decisión de lenguajes regulares.* Sea  $A$  un autómata finito determinista con  $n$  estados. Demuestre que:

1.  $L(A) \neq \emptyset$  si y solo si existe una cadena  $w$  con  $|w| < n$  que es aceptada,
  2.  $L(A)$  es infinito si y solo si existe una cadena aceptada  $w$  con  $n \leq |w| < 2n$ .
- En base a esto, diseñe un algoritmo que reciba como entrada un AFD y regrese un «sí» o un «no» dependiendo si el lenguaje que acepta es vacío o no. Haga lo mismo pero que la respuesta sea en función de que el lenguaje sea infinito o no.

*Ejercicio 79.* Busque en la literatura la demostración del teorema e MyHill-Nerode. Vuelva a escribirlo con sus propias palabras razonando cada paso.

*Ejercicio 80.* Demuestre que el índice de un lenguaje regular es igual al número de estados de su AFD mínimo asociado (descrito en el apéndice B).

# CAPÍTULO 5

---

## Autómatas a pila

---

En este capítulo se define y estudia el concepto de autómata a pila (AP), *push-down automaton* en inglés, el cual tiene su origen en los años sesenta, concretamente en los trabajos de Oettinger [19], Fisher [9] y Schutzenberger [22]. El concepto de autómata es más general que el de autómata finito (determinista o no, con transiciones o no), en el sentido de que existen lenguajes que son aceptados por un autómata a pila y que no son aceptados por algún autómata finito. Un ejemplo importante es el lenguaje de los corchetes equilibrados [ejercicio 75,] el cual corresponde a una versión simplificada del problema de verificar si un código tiene las aperturas y cierres equilibrados. Así que está más que justificada la introducción de autómatas más generales, pues ejemplos simples e importantes no son abarcados por los autómatas vistos hasta ahora.

En la primera sección de este capítulo se estudian de manera intuitiva los conceptos de aceptación de una cadena, movimientos de lectura y lenguaje aceptado por un autómata a pila. En particular, se plantea que a diferencia de los autómatas vistos anteriormente, hay dos maneras de distinguir una cadena: por estado de aceptación —como antes— y por «pila vacía»; todos estos conceptos se abordan de manera formal en la segunda y tercera sección. Finalmente, en la última parte se tratan los autómatas a pila deterministas (APD) y su relación con los lenguajes regulares.

## 5.1 Definición de autómata a pila

En esta sección se definen y ejemplifican sistemas con una cantidad finita de estados y señales llamados autómatas a pila. Estos autómatas son muy parecidos a los autómatas de los capítulos anteriores solo que además tienen una estructura de pila que controla el almacenamiento de las señales recibidas. Las limitaciones de los AFD se pusieron de manifiesto en el capítulo anterior. Por ejemplo se probó que el inocente lenguaje

$$L_{01} = \{0^n 1^n : n > 0\}$$

no es regular. Si el lector intentara construir un AFD para el conjunto  $L_{01}$  se daría cuenta que el problema es que se requiere saber si el número de ceros leídos al principio coincide con el número de unos que se lean enseguida; con el fin de decidir el cambio a un estado de aceptación, lo cual no es posible. Sin embargo, sí es posible construir un AP asociado a este lenguaje [ejercicio 90] ya que en la pila permite entre otras cosas almacenar la información de las señales recibidas.

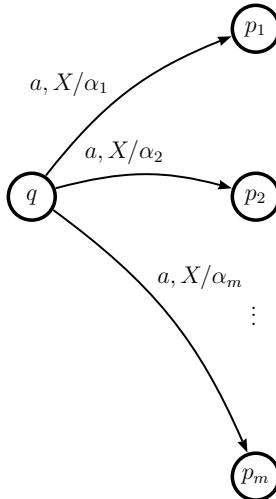
Un AP tiene un conjunto de estados, un alfabeto de símbolos de entrada, un estado inicial y un conjunto de estados de aceptación, todos ellos con la misma interpretación que antes. Además aparecen en escena dos objetos nuevos:

- un *alfabeto de pila*, esto es, un conjunto finito  $\Gamma$  de símbolos que se usan en el manejo de la pila. El alfabeto de pila suele contener al alfabeto de símbolos de entrada;
- un *símbolo inicial de la pila* el cual es un símbolo del alfabeto de pila que sirve de «tope», es decir, los movimientos en la pila se efectúan por encima de él, es por eso que para distinguirlo no debe ser un símbolo del alfabeto de símbolos de entrada, se denotará en lo que sigue como  $Z_0$ .

La función de transición indica cambio de estado y movimiento de pila dados un estado, un símbolo de entrada y un símbolo del alfabeto de pila. Concretamente, es una función que a cada tripleta  $(q, a, X)$  con  $q \in Q$ ,  $a \in \Sigma \cup \{\varepsilon\}$  y  $X \in \Gamma$ , le asigna un conjunto finito de pares ordenados:

$$\delta(q, a, X) = \{(p_1, \alpha_1), (p_2, \alpha_2), \dots, (p_m, \alpha_m)\}, \quad (5.1)$$

donde  $p_i \in Q$  y  $\alpha_i \in \Gamma^*$  para todo  $i = 1, 2, \dots, m$ ; ver figura 5.1 para su representación en diagramas de transición. El proceso de lectura se describe más adelante. La definición precisa es la siguiente:



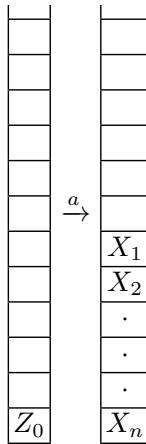
**Fig. 5.1** Representación de (5.1) a través de diagramas de transición.

**Definición AP.** Un *autómata a pila*  $A$  es una séptupla  $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  donde:

- $Q$  es un conjunto finito no vacío de *estados*;
- $\Sigma$  es un conjunto finito no vacío de *símbolos de entrada*;
- $\Gamma$  es un conjunto finito no vacío llamado *alfabeto de pila*;
- $\delta$  es una función que a cada tripleta  $(q, a, \dot{X})$  en  $Q \times \Sigma \cup \{\varepsilon\} \times \Gamma^*$  le asigna un conjunto finito de pares ordenados en  $Q \times \Gamma^*$  llamada *función de transición*;
- $q_0$  es el *estado inicial* perteneciente a  $Q$ ;
- $Z_0$  es un elemento de  $\Gamma$  llamado *símbolo inicial de la pila* que no pertenece a  $\Sigma$ ;
- $F$  es un subconjunto de  $Q$  de *estados de aceptación*.

Por otro lado, al igual que antes, la idea es que cada AP tenga asociado un lenguaje. A diferencia de los autómatas vistos anteriormente, para el caso de los AP hay dos formas de distinguir una cadena: *por estado de aceptación* —nada nuevo hasta aquí— y *por pila vacía*. En ambos casos, el proceso de lectura es el mismo, solo cambia el criterio de aceptación. Este proceso comienza con la siguiente situación inicial: en el estado  $q_0$ , la cadena  $w \in \Sigma^*$  sin leer y la pila con únicamente el símbolo  $Z_0$ . Supongamos que  $w = ab$  y que  $(p, \alpha) \in \delta(q_0, a, Z_0)$

donde  $\alpha = X_1X_2 \cdots X_n \in \Gamma^*$ . En este caso, se permite el siguiente cambio en el estado y la pila al leer  $a$ : se pasa del estado  $q_0$  al estado  $p$  y en la pila se reemplaza el símbolo  $Z_0$  por  $\alpha$ , apilando los elementos de  $\alpha$  empezando por  $X_n$  y terminando por  $X_1$ , de tal suerte que el símbolo  $X_1$  queda hasta arriba de la pila; ver figura 5.2.

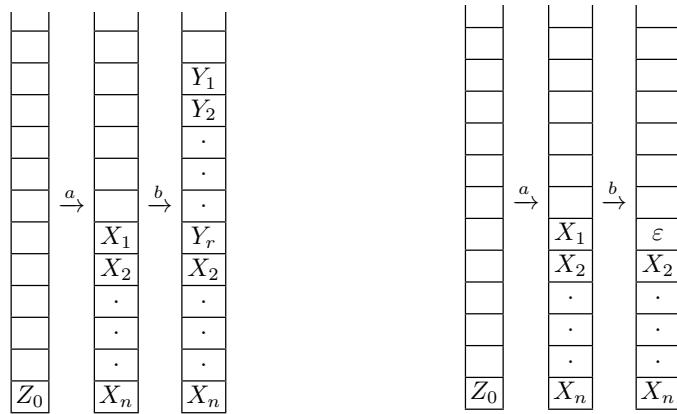


**Fig. 5.2** Cambio de pila al leer  $a$  a través de la transición  $(p, X_1X_2 \cdots X_n) \in \delta(q_0, a, Z_0)$ .

Después de leer  $a$ , se tiene el siguiente escenario: se está en el estado  $p$ ,  $X_1$  hasta arriba de la pila y  $b$  es la próxima señal a leer. Ahora supongamos que se tiene la transición  $(q, \beta) \in \delta(p, b, X_1)$  —si  $\delta(p, b, X_1) = \emptyset$  entonces el proceso de lectura no puede continuar—. En este caso, al leer  $b$ , se cambia del estado  $p$  al estado  $q$  y se reemplaza el símbolo  $X_1$  por la cadena  $\beta = Y_1Y_2 \cdots Y_r$  de tal forma que  $Y_1$  queda hasta arriba de la pila. Si  $\beta$  fuera la cadena vacía, el reemplazo de  $X_1$  por  $\beta = \varepsilon$  se puede interpretar como que  $X_1$  «se borra» de la pila. Con estos dos movimientos se completa la lectura de la cadena  $ab$ ; ver figura 5.3.

En general la ecuación (5.1) define  $m$  posibles movimientos al leer el símbolo  $a$  a partir de un estado  $q$  con símbolo  $X$  hasta arriba de la pila. Nótese que un AP es de naturaleza no-determinista. Además, como ya se ha dicho antes, la definición de un AP permite definir naturalmente dos formas de aceptación de una cadena:

- una cadena es aceptada por estado de aceptación si a partir de la situación inicial existe una forma de leerla de tal suerte que se llegue a un estado de

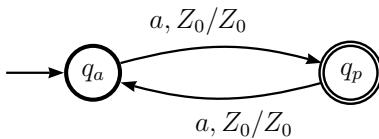


**Fig. 5.3** Cambio de pila en la lectura de la cadena  $ab$ . Reemplazo inicial de la pila y dos posibles casos para el segundo:  $\beta = Y_1Y_2 \dots Y_r$  y  $\beta = \epsilon$ .

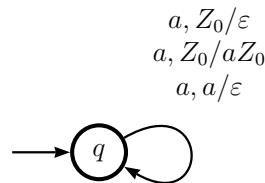
aceptación al concluir la lectura —independientemente de la situación de la pila al final de la lectura—;

- una cadena es aceptada por pila vacía si a partir de la situación inicial existe una forma de lectura para la cual la pila se vacía (incluso  $Z_0$  se remueve) al terminar la lectura —independientemente si se ha llegado a un estado de aceptación al concluir la lectura—.

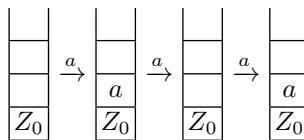
**Ejemplo 36.** Sea  $L_a = \{a^{2m+1} : m \geq 0\}$  el lenguaje aceptado el AFD del ejemplo 12 con  $F = \{q_p\}$ . A partir de un AFD es posible diseñar un AP con criterio de distinción de cadena por estado de aceptación, basta simplemente añadir artificialmente un alfabeto de la pila  $\Gamma = \{Z_0, a\}$  de tal forma que las transiciones mantengan a la pila constante (*cf.* teorema 9):



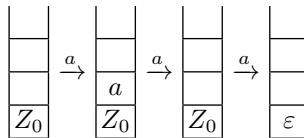
El movimiento de los estados al leer una cadena es igual a la del autómata determinista. La pila permanece constante, pues en cada movimiento se reemplaza  $Z_0$  por  $Z_0$ . Por otra parte,  $L$  puede ser aceptado por un AP con criterio de distinción de cadena por pila vacía. En este caso, los estados de aceptación son irrelevantes y es posible concentrarse en los movimientos en la pila:



Este AP admite más de un camino de lectura para una misma cadena ¡incluso con un solo estado! Por ejemplo, a la cadena  $aaa$  puede ser leída de al menos dos formas:



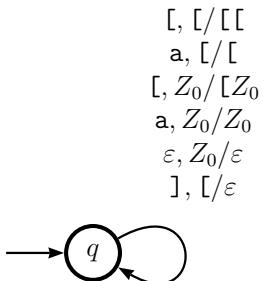
o bien:



**Ejemplo 37.** Sea  $L_{[a]}$  el lenguaje de las cadenas en el alfabeto  $\{a, [, ]\}$  con corchetes equilibrados definido mediante la siguiente regla inductiva:

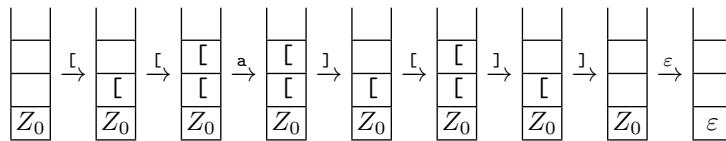
- R. INICIAL  $\varepsilon$  y  $a$  están en  $L_{[a]}$
- R. INDUCTIVA si  $x$  y  $y$  están en  $L_{[a]}$  entonces también lo están  $[x]$  y  $xy$ .

Mediante el lema de bombeo para lenguajes regulares [teorema 16] es posible verificar que  $L_{[a]}$  no es un lenguaje regular. Sin embargo se propone el siguiente AP con alfabeto de pila  $\{Z_0, a, [, ]\}$ :



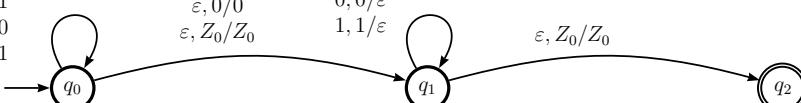
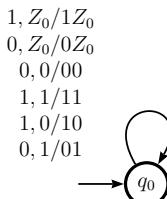
Los primeros 4 lazos, de arriba para abajo, tienen la función de acumular en la pila los corchetes abiertos leídos y de ignorar los símbolos **a**. Por otro lado, el último lazo borra el corchete abierto que se encuentra hasta arriba de la pila si se lee a continuación un corchete cerrado. Una vez leído el primer bloque de corchetes abiertos seguido del primer bloque de corchetes cerrados, el ciclo comienza de nuevo con la lectura de un segundo bloque de corchetes abiertos, ignorando los símbolos **a**, y así sucesivamente. La cadena tiene los corchetes equilibrados si (y solo si) un proceso de lectura conduce al final a que la pila contenga solo el símbolo  $Z_0$ . En ese caso se procede con la eliminación del símbolo  $Z_0$  por medio del penúltimo lazo. Veamos algunos ejemplos de lectura:

- En la siguiente figura se muestra el movimiento de la pila en un proceso de lectura de la cadena  $[[a] []]$  el cual conduce a la aceptación de esta cadena por pila vacía:



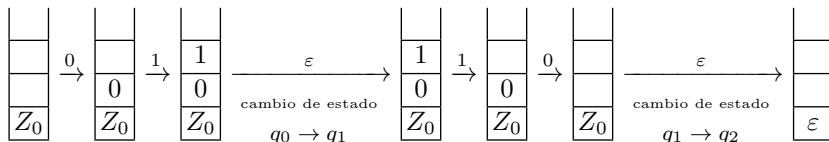
- La cadena  $] [$  no puede ser aceptada, pues no hay lazos que me permitan leer ni siquiera al primer símbolo  $]$  estando  $Z_0$  hasta arriba de la pila. Por tanto, la cadena no puede ser leída y en particular, no puede ser aceptada.
- Para la cadena  $[a] [$  existe un camino de lectura para la subcadena de inicio  $[a]$  tal que al terminar la lectura el símbolo  $Z_0$  se encuentra hasta arriba de la pila. El siguiente símbolo a leer es  $[$ , el cual no puede ser leído pues al igual que el inciso anterior, no hay lazos que lo permitan. Por tanto, el proceso de lectura se termina. La cadena  $[a] [$  no es aceptada, pues el proceso de lectura anterior es con el que más lejos se llega.

**Ejemplo 38.** Sea  $L_{x\bar{x}}$  el lenguaje formado por todas las cadenas binarias de la forma  $x\bar{x}$  donde  $\bar{x}$  es el reflejo de  $x$ . Por ejemplo, las cadenas  $\varepsilon$ , 110011, 01000010 son aceptadas por  $L_{x\bar{x}}$  mientras las cadenas 0, 101, 0010 no. Se propone el siguiente AP:



Estando en  $q_0$ , sin importar que símbolo esté hasta arriba de la pila, se almacena la información de las señales leídas. En cualquier momento se puede suponer que se ha leído justo la mitad de la cadena, hecho representado por cambiarse del estado  $q_0$  al estado  $q_1$  de forma instantánea independientemente de lo que se encuentre arriba de la pila. Estando en  $q_1$ , si 0 (resp. un 1) está hasta arriba de la pila y se lee otro 0 (resp. otro 1), entonces se elimina de la pila. Todo esto sucede sin moverse de estado; así sucesivamente. Si al final de la lectura de la cadena, se está en  $q_1$  y  $Z_0$  se encuentra hasta arriba de la pila, significa que la cadena es de la forma  $x\bar{x}$ ; en este caso es posible el cambio de estado de  $q_1$  al estado de aceptación  $q_2$ . Algunos ejemplos de lectura son los siguientes:

- La siguiente figura muestra un proceso de lectura de la cadena 0110 que conduce a un estado de aceptación:



- Para leer la cadena 010 hay muchas posibilidades. Una de ellas es quedarse en  $q_0$  hasta el final de la lectura y la pila almacenando todas las señales recibidas. Es posible también moverse a  $q_1$  en cualquier momento, esto es, antes de leer la primera señal, después de leer el primer 0, después de leer 01 o después de leer 010. En ningún caso es posible efectuar algún movimiento después, pues los lazos de  $q_1$  no lo permiten. Por tanto, esta cadena no es aceptada.

En los ejemplos anteriores se han expuesto de manera intuitiva las nociones de aceptación por pila vacía y por estado de aceptación, todo esto se formaliza en la siguiente sección.

### **Lista de ejercicios de la sección 5.1**

*Ejercicio 81.* Diseñe un AP que acepte por pila vacía al lenguaje de textos planos que contengan sentencias anidadas de la forma `begin ... end`.

*Ejercicio 82.* Diseñe un AP que acepte por pila vacía al conjunto de palíndromos del alfabeto  $\{a, b, c\}$ .

*Ejercicio 83.* Diseñe un AP de un solo estado que acepte por pila vacía al lenguaje  $L_{x\bar{x}}$  del ejemplo 38.

*Ejercicio 84.* A partir de un AFD  $A$  construya un AP de un solo estado que acepte por pila vacía a  $L(A)$ . Sugerencia: considere a los estados de  $A$  como símbolos de la pila de tal forma que los movimientos de la pila sean determinados por la función de transición de  $A$ .

## 5.2 Aceptación por pila vacía y por estado de aceptación

Para definir formalmente el lenguaje aceptado por un AP se va a echar mano de triplets que contienen la información del estado, la cadena y la pila en cada paso del proceso de lectura de una cadena de manera similar a las duplas de los AFD, AFND, AFND- $\varepsilon$  contienen la información del estado y la cadena en un paso del proceso de lectura. Sea  $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  un AP . Dos triplets están relacionadas, se escribe:

$$(q, w, \gamma) \vdash (p, z, \eta) \quad (5.2)$$

si  $w = az$  ( $a$  puede ser la cadena vacía),  $\gamma = X\beta$ ,  $\eta = \alpha\beta$  y  $(p, \alpha) \in \delta(q, a, X)$ . La triplete  $(q, w, X\beta)$  representa el momento en que «se está en el estado  $q$ , falta por leer  $w$  y  $X$  se encuentra hasta arriba de la pila». La transición  $(p, \alpha) \in \delta(q, a, X)$  indica el movimiento del estado  $q$  al estado  $p$ , lectura de  $a$  (o transición instantánea) y reemplazo de  $X$  por  $\alpha$  en la pila. Al igual que con las duplas de los autómatas de los capítulos anteriores, se tiene la notación útil de varios movimientos a través de la clausura de  $\vdash$ :

### Definición $\vdash^*$ .

- R. INICIAL  $(q, w, \gamma) \vdash (p, z, \eta)$  implica  $(q, w, \gamma) \vdash^* (p, z, \eta)$   
en 1 movimiento;
  - R. INDUCTIVA si  $(q, w, \gamma) \vdash^* (p, z, \eta)$  en  $n$  movimientos y  
 $(p, z, \eta) \vdash (r, u, \beta)$  entonces  $(q, w, \gamma) \vdash^* (r, u, \beta)$   
en  $n + 1$  movimientos.
- Por convención,  $(q, w, \gamma) \vdash^* (q, w, \gamma)$  en 0 movimientos.

Con la notación de movimientos de triplets se definen de forma natural los lenguajes aceptados por un AP ya sea por estado de aceptación o por pila vacía como sigue:

**Definición L(A) y N(A).** Sea  $A$  un AP. Al conjunto  $L(A)$  de cadenas  $w \in \Sigma^*$  tales que  $(q_0, w, Z_0) \vdash^* (q, \varepsilon, \beta)$  para  $q \in F$  y  $\beta \in \Gamma^*$  se le llama *lenguaje aceptado por A por estado de aceptación*. Al conjunto  $N(A)$  de  $w \in \Sigma^*$  tales que  $(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$  para  $q \in Q$  se le llama *lenguaje aceptado por A por pila vacía*.

**Ejemplo 39.** Sea  $A$  el autómata de un estado del ejemplo 36. Un camino de lectura de la cadena  $aaa$  es el siguiente:

$$(q, aaa, Z_0) \vdash (q, aa, aZ_0) \vdash (q, a, Z_0) \vdash (q, \varepsilon, aZ_0)$$

por tanto  $(q, aaa, Z_0) \stackrel{*}{\vdash} (q, \varepsilon, aZ_0)$ . Sin embargo, la cadena  $aaa \in N(A)$  pues  $(q, aaa, Z_0) \stackrel{*}{\vdash} (q, \varepsilon, \varepsilon)$ , de hecho:

$$(q, aaa, Z_0) \vdash (q, aa, aZ_0) \vdash (q, a, Z_0) \vdash (q, \varepsilon, \varepsilon).$$

**Ejemplo 40.** Sea  $A$  el autómata del ejemplo 37. La lectura de la cadena  $[[a] []]$  que comprueba que  $[[a] []] \in N(A)$  con notación de tripletas es:

$$\begin{aligned} (q, [[a] []], Z_0) &\vdash (q, [a] [], [Z_0]) \vdash (q, a) [], [[Z_0]] \vdash (q, ] [], [[Z_0]] \vdash \\ &\vdash (q, []], [Z_0]) \vdash (q, ]], [[Z_0]] \vdash (q, ], [Z_0]) \vdash (q, \varepsilon, Z_0) \vdash (q, \varepsilon, \varepsilon). \end{aligned}$$

**Ejemplo 41.** Sea  $A$  el autómata del ejemplo 38. La lectura de la cadena  $0110$  que comprueba que  $0110 \in L(A)$  con notación de tripletas es:

$$\begin{aligned} (q_0, 0110, Z_0) &\vdash (q_0, 110, 0Z_0) \vdash (q_0, 10, 10Z_0) \vdash \\ &\vdash (q_1, 10, 10Z_0) \vdash (q_1, 0, 0Z_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_2, \varepsilon, Z_0). \end{aligned}$$

Por otro lado, para los AFND- $\varepsilon$  [ejercicio 52] se tiene la siguiente propiedad de las duplas:

- si  $(q, x) \stackrel{*}{\vdash} (p, y)$  entonces  $(q, xw) \stackrel{*}{\vdash} (p, yw)$ , para cualquier cadena de símbolos de entrada  $w$ , esto quiere decir que es posible *añadir* cualquier cadena  $w$  a la derecha del segundo elemento en ambos lados de la relación;
- si  $(q, xw) \stackrel{*}{\vdash} (p, yw)$  para alguna cadena de símbolos de entrada  $w$ , entonces  $(q, x) \stackrel{*}{\vdash} (p, y)$ , es decir se puede *eliminar*  $w$  del segundo elemento en ambos lados de la relación.

En el caso de los AP se tiene una generalización de lo anterior pero parcialmente exitosa. La primera parte se cumple impecablemente, pues también vale *añadir* cadenas a la derecha del tercer elemento (la pila) en ambos lados de la relación entre tripletas. Sin embargo la segunda parte se cumple parcialmente pues en general no es cierto que  $(q, x, \alpha\gamma) \vdash^* (p, y, \beta\gamma)$  implique que  $(q, x, \alpha) \vdash^* (p, y, \beta)$  [ejercicio 89]. En resumen:

**Propiedad 5.** Sean  $p$  y  $q$  estados,  $x$  y  $y$  cadenas de símbolos de entrada y sean  $\alpha$  y  $\beta$  cadenas del alfabeto de pila, todos de un mismo AP:

- si  $(q, x, \alpha) \vdash^* (p, y, \beta)$  entonces  $(q, xw, \alpha\gamma) \vdash^* (p, yw, \beta\gamma)$ , para toda cadena de símbolos de entrada  $w$  y cadena del alfabeto de pila  $\gamma$ .
- si  $(q, xw, \alpha) \vdash^* (p, yw, \beta)$  para alguna cadena de símbolos de entrada  $w$ , entonces  $(q, x, \alpha) \vdash^* (p, y, \beta)$ .

*Prueba.* La demostración de la primera parte se hace por inducción sobre el número de movimientos de la relación binaria  $\vdash^*$ . En concreto se prueba la siguiente afirmación para todo  $n$  natural:

$P(n) : \text{si } (q, x, \alpha) \vdash^* (p, y, \beta) \text{ en } n \text{ movimientos, entonces para cualquier}$   
 $\text{cadena de símbolos de entrada } w \text{ y cualquier cadena del alfabeto de pila}$   
 $\gamma, (q, xw, \alpha\gamma) \vdash^* (p, yw, \beta\gamma) \text{ en } n \text{ movimientos.}$

Si  $(q, x, \alpha) \vdash (p, y, \beta)$  entonces existen  $a \in \Sigma \cup \{\varepsilon\}$ ,  $z \in \Sigma^*$ ,  $X \in \Gamma$ ,  $\eta, \xi \in \Gamma^*$  tales que  $x = az$ ,  $y = z, \alpha = X\xi$ ,  $\beta = \eta\xi$  y además  $(p, \eta) \in \delta(q, a, X)$ . Luego para cualquier cadena  $w \in \Sigma^*$  y  $\gamma \in \Gamma^*$ :

$$(q, xw, \alpha\gamma) \vdash (q, azw, X\xi\gamma) \vdash (p, zw, \eta\xi\gamma) = (p, yw, \beta\gamma).$$

Con esto se demuestra  $P(1)$ . Se supone ahora  $P(n)$  (hipótesis de inducción). Si  $(q, x, \alpha) \vdash^* (p, y, \beta)$  en  $n + 1$  movimientos, entonces existen cadenas  $z \in \Sigma^*$  y  $\eta \in \Gamma^*$  tales que  $(q, x, \alpha) \vdash (p, z, \eta)$  en  $n$  movimientos y  $(p, z, \eta) \vdash (p, y, \beta)$  en 1 movimiento. Por hipótesis de inducción  $(q, xw, \alpha\gamma) \vdash^* (p, zw, \eta\gamma)$  en  $n$  movimientos. Además  $(p, zw, \eta\gamma) \vdash^* (p, yw, \beta\gamma)$  ya que se ha probado  $P(1)$ . Luego  $(q, xw, \alpha\gamma) \vdash^* (p, yw, \beta\gamma)$  en  $n + 1$  movimientos. De forma análoga se demuestra la segunda parte.  $\square$

**Ejemplo 42.** Sea  $A$  el AP del ejemplo 38 y  $x$  una cadena binaria. Mediante la propiedad 5 es posible verificar que  $x\bar{x} \in L(A)$ . Primero se ha de probar que:

$$(q_0, x\bar{x}, Z_0) \stackrel{*}{\vdash} (q_0, \bar{x}, \bar{x}Z_0). \quad (5.3)$$

De hecho, para todo  $n$  natural se cumple la siguiente afirmación más general:

$$P(n) : \text{si } |x| = n, (q_0, x\bar{x}, \alpha) \stackrel{*}{\vdash} (q_0, \bar{x}, \bar{x}\alpha) \text{ para toda } \alpha \in \Gamma^*.$$

La afirmación  $P(1)$  es verdadera, pues si  $\alpha = X\beta$ ,  $(q_0, 00, X\beta) \stackrel{*}{\vdash} (q_0, 0, 0X\beta)$  por la definición del autómata, sin importar si  $X$  es 0, 1 o  $Z_0$ . Lo mismo si  $x = 1$ . Por la propiedad 5 y la hipótesis de inducción  $(q_0, x\bar{x}0, \alpha) \stackrel{*}{\vdash} (q_0, \bar{x}0, \bar{x}\alpha)$  para cualquier  $\alpha \in \Gamma^*$ . Por tanto si  $\gamma \in \Gamma^*$ :

$$(q_0, 0x\bar{x}0, \gamma) \stackrel{*}{\vdash} (q_0, x\bar{x}0, 0\gamma) \stackrel{*}{\vdash} (q_0, \bar{x}0, \bar{x}0\gamma).$$

Análogamente se tiene que  $(q_0, 1x\bar{x}1, \gamma) \stackrel{*}{\vdash} (q_0, \bar{x}1, \bar{x}1\gamma)$ . Con esto se prueba  $P(n+1)$ . Se ha de notar que fue necesario considerar un caso más general con el fin de comprobar (5.3) pues de otra manera no parece natural el paso inductivo. Finalmente, lo que el lector comprobará en el ejercicio 87 y (5.3) implican que:

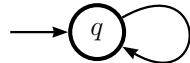
$$(q_0, x\bar{x}, Z_0) \stackrel{*}{\vdash} (q_0, \bar{x}, \bar{x}Z_0) \vdash (q_1, \bar{x}, \bar{x}Z_0) \stackrel{*}{\vdash} (q_1, \varepsilon, Z_0) \vdash (q_2, \varepsilon, Z_0).$$

## **Lista de ejercicios de la sección 5.2**

*Ejercicio 85.* Sea  $A$  el AP de un solo estado del ejemplo 36. Pruebe que efectivamente  $N(A)$  es el conjunto  $L_a = \{a^{2m+1} : m \geq 0\}$ .

*Ejercicio 86.* Considere el siguiente AP:

$$\begin{array}{l} [, [], /[], \\ [, Z_0 / [Z_0 \\ \varepsilon, Z_0 / \varepsilon \\ ], [/ \varepsilon \end{array}$$



Verifique que este autómata acepta como lenguaje al conjunto  $L_{\square}$  de cadenas de corchetes equilibrados definido en el ejercicio 75.

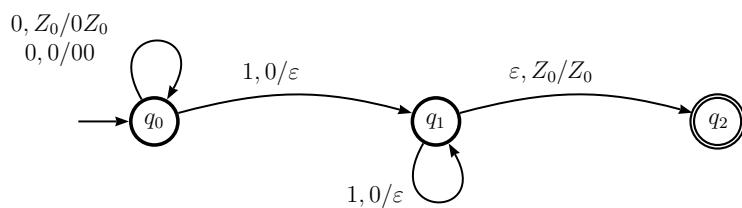
*Ejercicio 87.* Sea  $A$  el AP del ejemplo 38. Pruebe por inducción que para cualquier cadena binaria  $x$  se cumple que  $(q_1, \bar{x}, \bar{x}Z_0) \stackrel{*}{\vdash} (q_1, \varepsilon, Z_0)$ .

*Ejercicio 88.* Para el AP del ejemplo 37 demuestre que para todo  $n$  y  $m$  en los naturales:

$$(q, [^n \mathbf{a}^m]^n, Z_0) \xrightarrow{*} (q, \varepsilon, \varepsilon).$$

*Ejercicio 89.* De un ejemplo de AP donde se muestre que en general no es cierto que  $(q, x, \alpha\gamma) \xrightarrow{*} (p, y, \beta\gamma)$  implica  $(q, x, \alpha) \xrightarrow{*} (p, y, \beta)$ .

*Ejercicio 90.* Pruebe que  $L(A)$  es el conjunto  $L_{01} = \{0^n 1^n : n > 0\}$  donde  $A$  es el siguiente AP:



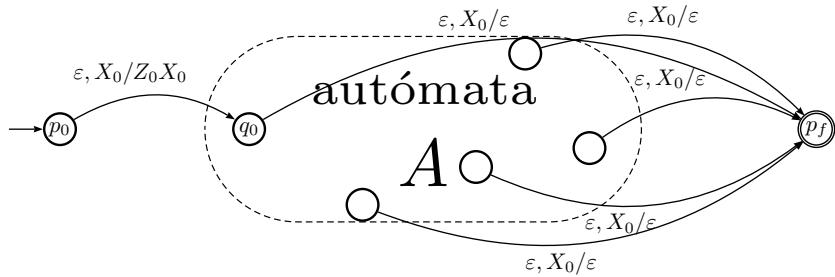
*Ejercicio 91.* Demuestre que el AP que construyó en el ejercicio 84 es correcto.

### 5.3 Equivalencia entre aceptación por estados y por pila vacía

Sea  $\Sigma$  un alfabeto de símbolos de entrada y sea  $L$  un lenguaje en  $\Sigma^*$ . Si  $A$  es un AP tal que  $L = N(A)$  es fácil construir, a partir de  $A$ , otro AP  $B$  tal que  $L = L(B)$ . Sean  $q_0$  y  $Z_0$  el estado inicial y el símbolo inicial de la pila de  $A$ , respectivamente. A  $B$  se le dota de un nuevo estado inicial y un nuevo símbolo inicial del alfabeto de pila, digamos  $p_0$  y  $X_0$ , respectivamente, además de la inclusión de un estado de aceptación  $p_f$ . La idea fundamental sobre las transiciones de  $B$  se puede resumir en tres pasos (ver figura 5.4):

- pasar instantáneamente de  $p_0$  a  $q_0$  y sustituir  $X_0$  por  $Z_0 X_0$ , se está entonces en la situación inicial de  $A$ ;
- emular al autómata  $A$ ;
- a partir de cualquier estado de  $A$  si  $X_0$  se encuentra hasta arriba de la pila —lo que significa que la pila de  $A$  se vació—, se pasa al estado  $p_f$ .

Formalmente:



**Fig. 5.4** Construcción de  $B$  a partir de  $A$  tal que  $L(B) = N(A)$ .

**Algoritmo A→B: L(B)=N(A).** Sea  $A = (Q, \Sigma, \Gamma, \delta_A, q_0, Z_0, \emptyset)$  un AP tal que  $L = N(A)$ . Se define  $B$  como el AP:

$$(Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_B, p_0, X_0, \{p_f\})$$

donde la función de transición  $\delta_B$  se define como sigue:

- $\delta_B(p_0, \varepsilon, X_0) = \{(q_0, Z_0 X_0)\}$ ;
  - si  $(p, \alpha) \in \delta_A(q, a, X)$  entonces  $(p, \alpha) \in \delta_B(q, a, X)$  para todo  $p, q \in Q$ ,  $a \in \Sigma \cup \{\varepsilon\}$  y  $X \in \Gamma$ ;
  - $(p_f, \varepsilon) \in \delta_B(q, \varepsilon, X_0)$  para todo  $q \in Q$ .
- Se cumple que  $L = L(B)$ .

La incorporación del nuevo símbolo inicial de la pila  $X_0$  es importante, pues de otra manera se pudiera llegar para una cadena de aceptación  $w \in N(A)$  a un camino de la forma  $(p_0, w, Z_0) \xrightarrow{B}^* (q, \varepsilon, \varepsilon)$  y entonces no sería posible pasar al estado de aceptación  $q_f$  pues los movimientos en la pila no admiten, por definición, transiciones instantáneas. Resumiendo:

**Teorema 7.** *Sea  $L$  un lenguaje. Si  $L = N(A)$  para algún AP  $A$  entonces existe otro AP  $B$  tal que  $L = L(B)$ .*

Prueba. Se probará que para toda cadena  $w \in \Sigma^*$ :

$$(p_0, w, X_0) \xrightarrow{B}^* (p_f, \varepsilon, \varepsilon) \quad (5.4)$$

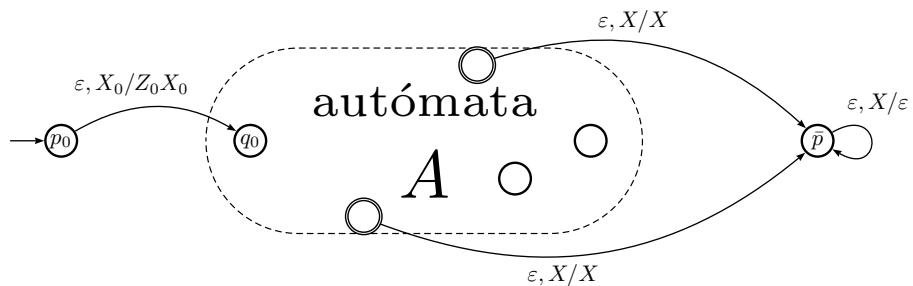
si y solo si existe  $q \in Q$  que satisface:

$$(q_0, w, Z_0) \xrightarrow{A}^* (q, \varepsilon, \varepsilon). \quad (5.5)$$

Se supone que se cumple (5.4). Se tiene que  $(p_0, w, X_0) \xrightarrow{B} (q_0, w, Z_0 X_0)$  es el único movimiento que se puede hacer a partir de  $(p_0, w, X_0)$ . Por tanto  $(q_0, w, Z_0 X_0) \xrightarrow{B}^* (p_f, \varepsilon, \varepsilon)$ . La única manera de llegar a  $(p_f, \varepsilon, \varepsilon)$  es a través del movimiento  $(q, \varepsilon, X_0) \xrightarrow{B} (p_f, \varepsilon, \varepsilon)$  para algún  $q \in Q$ . Por tanto  $(q_0, w, Z_0 X_0) \xrightarrow{B}^* (q, \varepsilon, X_0)$ . Por definición de  $B$  se tiene que  $(q_0, w, Z_0) \xrightarrow{B}^* (q, \varepsilon, \varepsilon)$ . Por otro lado, si (5.5) es cierto, entonces por definición de  $B$ ,  $(q_0, w, Z_0) \xrightarrow{B}^* (q, \varepsilon, \varepsilon)$  para algún  $q \in Q$ . Por la propiedad 5  $(q_0, w, Z_0 X_0) \xrightarrow{B}^* (q, \varepsilon, X_0)$ , luego  $(p_0, w, X_0) \xrightarrow{B} (q_0, w, Z_0 X_0) \xrightarrow{B}^* (q, \varepsilon, X_0)$ . Es decir  $(p_0, w, X_0) \xrightarrow{B}^* (q, \varepsilon, X_0)$   $\square$

Por otro lado, si  $A$  es un AP tal que  $L = L(A)$ , a partir de  $A$  es posible construir otro AP  $B$  tal que  $L = N(B)$ . De igual manera de antes, a  $B$  se le dota de un nuevo estado inicial y un nuevo símbolo inicial del alfabeto de pila,  $p_0$  y  $X_0$ , respectivamente. Además se incluye un nuevo estado  $\bar{p}$  de «vaciado de pila» al que se llegará de cualquier estado de aceptación de  $A$ . La idea fundamental sobre las transiciones de  $B$  se resumen en cuatro pasos (ver figura 5.5):

- pasar instantáneamente de  $p_0$  a  $q_0$  y sustituir  $X_0$  por  $Z_0 X_0$ , se está entonces en la situación inicial de  $A$ ;
- emular al autómata  $A$ ;
- de cualquier estado de aceptación  $q$  se pasa instantáneamente al estado  $\bar{p}$  sin que la pila se vea modificada;
- estando en  $\bar{p}$  la pila se vacía.



**Fig. 5.5** Construcción de  $B$  a partir de  $A$  tal que  $N(B) = L(A)$ .

Más precisamente:

**Algoritmo A→B: N(B)=L(A).** Sea  $A = (Q, \Sigma, \Gamma, \delta_A, q_0, Z_0, F)$  un AP tal que  $L = L(A)$ . Se define  $B$  como el AP:

$$(Q \cup \{p_0, \bar{p}\}, \Sigma, \Gamma \cup \{X_0\}, \delta_B, p_0, X_0, \emptyset)$$

donde la función de transición  $\delta_B$  se define como sigue:

- $\delta_B(p_0, \varepsilon, X_0) = \{(q_0, Z_0 X_0)\};$
- si  $(p, \alpha) \in \delta_A(q, a, X)$  entonces  $(p, \alpha) \in \delta_B(q, a, X)$  para todo  $p, q \in Q$ ,  $a \in \Sigma \cup \{\varepsilon\}$  y  $X \in \Gamma$ ;
- para todo  $q \in F$  y  $X \in \Gamma \cup \{X_0\}$ ,  $(\bar{p}, X) \in \delta_B(q, \varepsilon, X);$
- para todo  $X \in \Gamma \cup \{X_0\}$ ,  $\delta_B(\bar{p}, \varepsilon, X) = \{(\bar{p}, \epsilon)\}.$

Se cumple que  $L = N(B)$ .

Por tanto, podemos deducir el siguiente teorema, los detalles se dejan como ejercicio para el lector [ejercicio 96]:

**Teorema 8.** *Sea  $L$  un lenguaje. Si  $L = L(A)$  para algún AP  $A$  entonces existe otro AP  $B$  tal que  $L = N(B)$ .*

### Listado de ejercicios de la sección 5.3

*Ejercicio 92.* Diseñe dos AP diferentes que acepten por pila vacía al lenguaje aceptado por el AFD de «los tres focos» descrito en la figura 2.1.

*Ejercicio 93.* Diseñe dos AP diferentes que acepten por pila vacía al lenguaje aceptado por el AFND del ejemplo 16.

*Ejercicio 94.* Diseñe un AP que acepte como lenguaje por pila vacía al conjunto de cadenas de símbolos de texto plano con llaves (`{,}`) equilibradas y que comiencen con `\begin{document}` y terminen con `\end{document}`. Transforma este AP en otro que acepte el mismo lenguaje pero por estado de aceptación.

*Ejercicio 95.* A partir del autómata del ejercicio 86 construye un AP que acepte el mismo lenguaje pero por estado de aceptación.

*Ejercicio 96.* Complete los detalles para deducir el teorema 8.

## 5.4 Autómatas a pila deterministas y lenguajes regulares

En esta sección se estudia la noción de determinismo en un AP y su relación con los lenguajes regulares. La idea es establecer una definición que permita un solo camino de movimientos de lectura a partir de una tripleta cualquiera. Es importante señalar que una sola ruta no excluye necesariamente a las transiciones instantáneas como se podría pensar en primera instancia. Para asegurar unicidad en el camino de lectura es preciso que a partir de cualquier tripleta  $(q, az, X\beta)$  solo exista una posibilidad de movimiento, esto es, o bien  $\delta(q, a, X)$  es vacío —en este caso  $\delta(q, \varepsilon, X)$  pudiera tener un elemento o ser vacío también— o bien contiene un solo elemento; si es este último caso entonces no debe existir transiciones instantáneas estando en  $q$  y  $X$  hasta arriba de la pila, esto es,  $\delta(q, \varepsilon, X)$  tiene que ser vacío para evitar la posibilidad de otro movimiento a partir de  $(q, az, X\beta)$ . En otras palabras:

**Definición APD.** Se dice que un AP  $A = (Q, \Sigma, \Gamma, \delta_A, q_0, Z_0, F)$  es *determinista* si:

- para cualquier  $q \in Q$  y  $a \in \Sigma \cup \{\varepsilon\}$  el conjunto  $\delta(q, a, X)$  tiene a lo más un elemento;
- si  $\delta(q, a, X) \neq \emptyset$  para algún  $a \in \Sigma$  entonces  $\delta(q, \varepsilon, X) = \emptyset$ .

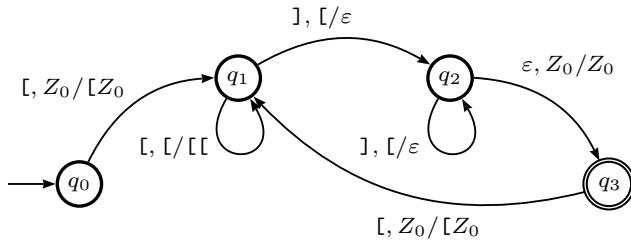
**Ejemplo 43.** El AP de un solo estado del ejemplo 36 no es un APD, pues  $\delta(q, a, Z_0) = \{(q, \varepsilon), (q, aZ_0)\}$  con lo que no se satisface la primera condición de determinismo. Por ejemplo a partir de la tripleta  $(q, a, Z_0)$  hay dos posibles movimientos determinados por  $\delta(q, a, Z_0)$ :

$$(q, a, Z_0) \vdash (q, \varepsilon, aZ_0)$$

y

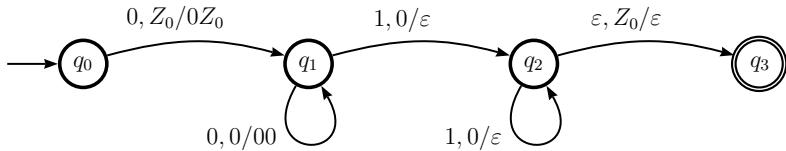
$$(q, a, Z_0) \vdash (q, a, \varepsilon).$$

**Ejemplo 44.** El AP del ejercicio 86 tampoco es determinista pues aunque se tiene que  $\delta(q, \square, Z_0) \neq \emptyset$  también es cierto que  $\delta(q, \varepsilon, Z_0) \neq \emptyset$  por lo que no se cumple la segunda condición determinismo. Este AP acepta como lenguaje al conjunto  $L_{\square}$  de cadenas de corchetes equilibrados definido en el ejercicio 75. Este lenguaje sin embargo puede ser aceptado por el siguiente APD por estado de aceptación:



Sin embargo, no existe un APD que acepte este lenguaje por pila vacía. Si existiera existiría un solo camino de lectura de la cadena  $\square\square$ :  $(q_0, \square\square, Z_0) \xrightarrow{*} (q, \varepsilon, \varepsilon)$  para algún estado  $q$ . Luego  $(q_0, \square\square, Z_0) \xrightarrow{*} (q, \square, \varepsilon)$  siendo este el único camino de lectura de  $\square\square$  y por tanto esta cadena no sería aceptada.

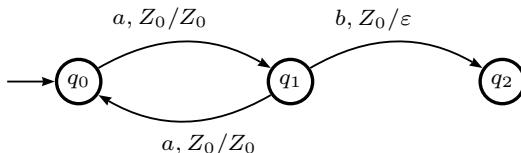
**Ejemplo 45.** Es fácil verificar que el siguiente autómata es un APD, incluso con transiciones instantáneas:



Este autómata acepta al lenguaje  $L_{01} = \{0^n 1^n : n > 0\}$  tanto por pila vacía como por estado de aceptación, pues se ha de notar que para todo  $n > 0$ :

$$(q_0, 0^n 1^n, Z_0) \xrightarrow{*} (q_3, \varepsilon, \varepsilon).$$

**Ejemplo 46.** Considere el diagrama de transición:



Este autómata corresponde a un APD que acepta por pila vacía al lenguaje regular  $L_a\{b\} = \{a^{2m+1}b : m \geq 0\}$ , con único camino de aceptación para una cadena  $a^{2m+1}b$ :

$$(q_0, a^{2m+1}b, Z_0) \xrightarrow{} (q_1, a^{2m}b, Z_0) \xrightarrow{} (q_0, a^{2m-1}b, Z_0) \xrightarrow{*} (q_1, b, Z_0) \xrightarrow{} (q_2, \varepsilon, \varepsilon).$$

A diferencia de la relación entre los AFD, AFND y los AFND- $\varepsilon$ , existen lenguajes aceptados un AP que no es aceptado por algún APD. Un ejemplo es el lenguaje  $L_{x\bar{x}}$  del ejemplo 38. La prueba formal es difícil pero la idea es que el no-determinismo es esencial para establecer donde comienza la reflexión de la cadena [11]. Sin embargo si  $L$  es un lenguaje regular es fácil construir un APD que lo acepte como lenguaje [ejercicio 98]:

**Teorema 9.** Si  $L$  es un lenguaje regular entonces existe un APD  $B$  tal que  $L(B) = L$ .

Es importante señalar que existen lenguajes regulares que no pueden ser aceptados por un APD por pila vacía. Un ejemplo es el lenguaje  $L_0 = \{0^n : n > 0\}$ . De hecho, si 0 fuera aceptada por pila vacía por un APD se tendría para algún estado  $p$ :

$$(q_0, 0, Z_0) \stackrel{*}{\vdash} (p, \varepsilon, \varepsilon)$$

y este sería el único camino. Por tanto,  $(q_0, 00, Z_0) \stackrel{*}{\vdash} (p, 0, \varepsilon)$  y ya no sería posible avanzar más; en otras palabras la cadena 00 no sería aceptada.

Por el teorema 9, el lenguaje regular  $L_0$  es un ejemplo de un lenguaje que es aceptado por un APD por estado de aceptación pero no por un APD por pila vacía. Esto significa que no se tiene un teorema análogo al teorema 8 para autómatas a pila deterministas. Sin embargo el teorema 7 sí se cumple en el caso determinista [ejercicio 99]. La figura 5.6 resume lo expuesto hasta ahora:

#### **Lista de ejercicios de la sección 5.4**

*Ejercicio 97.* Diseñe un APD que acepte los siguientes lenguajes por estado de aceptación o por pila vacía:

1.  $\{0^n 1^m : n \leq m\}$ ,
2.  $\{0^n 1^m : n \geq m\}$ .

*Ejercicio 98.* Sea  $A$  un AFD. Construye a partir de  $A$  un APD que acepte el mismo lenguaje que  $A$ . Sugerencia: defina la función de transición de  $B$  como  $\delta_B(q, a, Z_0) = \{(\delta_A(q, a), Z_0)\}$  donde  $\delta_A$  es la función de transición de  $A$ .

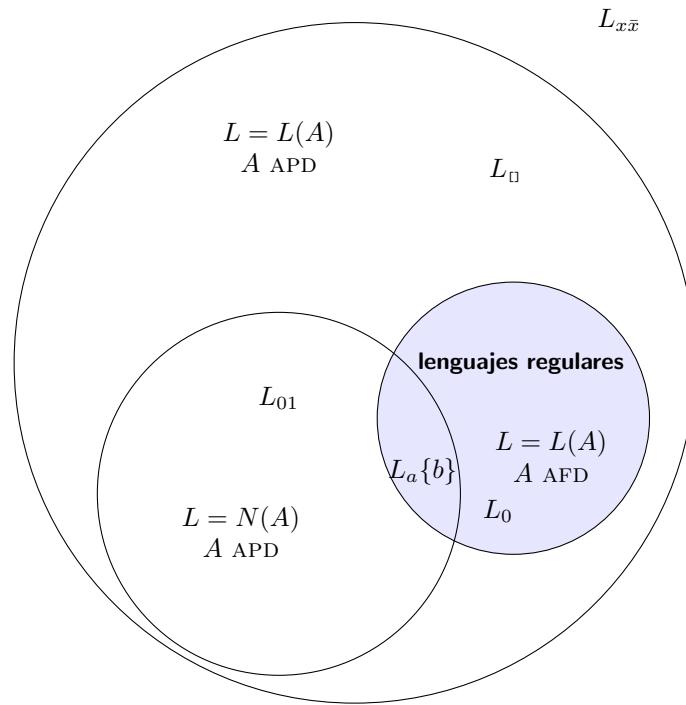
*Ejercicio 99.* Pruebe que si un lenguaje es aceptado por un APD por pila vacía, entonces es aceptado por un APD por estado de aceptación. Sugerencia: copie el esquema de la prueba del teorema 7.

*Ejercicio 100.* ¿Porqué la prueba del teorema 8 no puede ser adaptada para demostrar la afirmación falsa «si un lenguaje es aceptado por estado de aceptación por un APD entonces es aceptado por pila vacía por otro APD»?

$$L = L(A) = N(B)$$

$A$  AP

$B$  AP



**Fig. 5.6** Relación entre lenguajes en función de la aceptación por autómatas.

*Ejercicio 101.* Sea  $A$  un APD y sea  $L = L(A)$ . Considere el conjunto:

$$L_\diamond = \{x\diamond : x \in L\}$$

donde  $\diamond$  es un símbolo que no pertenece al conjunto de símbolos de entrada de  $A$ . Demuestre que existe un APD  $B$  tal que  $L_\diamond = N(B)$ .

# CAPÍTULO 6

---

## Gramáticas libres de contexto

---

En este capítulo se estudian las gramáticas libres de contexto (GLC), concepto introducido en la literatura por el lingüista, filósofo y activista Noam Chomsky a mediados de los años cincuenta [4] y que permiten describir la sintaxis de una gran parte de los lenguajes de programación. En las dos primeras secciones de este capítulo se establecen los elementos básicos de una gramática libre de contexto: su definición formal y el lenguaje que acepta. En los ejercicios se tocan de manera superficial las nociones de gramática regular y gramática sin restricciones con el fin de dar un poco de perspectiva al tema. En la tercera sección se estudian los árboles de derivación y su relación con la noción de no-ambigüedad de una gramática, problema identificado por Cantor [3] y Floyd [10] a principios de los años sesenta. La no-ambigüedad de una gramática está relacionada con el buen funcionamiento del analizador sintáctico el cual es un programa que analiza la validez de una cadena de símbolos de acuerdo a las reglas de una gramática formal.

## 6.1 Definición de gramática libre de contexto

Una GLC es un concepto abstracto relacionado con la noción de *sintaxis* la cual, según el *Diccionario de la lengua española* [7], tiene la siguiente acepción:

Parte de la gramática que estudia el modo en que se combinan las palabras y los grupos que estas forman para expresar significados, así como las relaciones que se establecen entre todas esas unidades.

La idea específica es establecer el «modo» a través de un conjunto finito de reglas —que se llaman *producciones*— las cuales definen un conjunto de cadenas válidas de un *alfabeto*, por ejemplo, secuencias correctas de un lenguaje de programación específico. Los grupos formados por las cadenas válidas, es decir, las «unidades» a las que se refiere la definición de sintaxis, se representan por medio de *variables sintácticas*. Dentro de las variables sintácticas hay una que representa al conjunto total de cadenas válidas llamada *símbolo inicial*. Formalmente:

**Definición GLC.** Una *gramática libre de contexto*  $G$  es una cuádrupla  $(T, V, S, P)$  donde:

$T$  es un conjunto finito no vacío llamado *alfabeto de símbolos terminales*;

$V$  es un conjunto finito no vacío de *variables sintácticas*;

$S$  es un elemento de  $V$  llamado *símbolo inicial*;

$P$  es una lista de expresiones —llamadas *producciones*— de la forma:

$$A \rightarrow \alpha$$

donde  $A$  es una variable sintáctica y  $\alpha \in (T \cup V)^*$ . A la letra  $A$  se le llama *base* de la producción y a  $\alpha$  *cuerpo* de la producción. Si se tienen varias producciones con la misma base  $A$  pero con diferentes cuerpos  $\alpha_1, \alpha_2, \dots, \alpha_n$  entonces se escribe:

$$A \rightarrow \alpha_1 | \alpha_2 | \cdots | \alpha_n.$$

Como se explica en los siguientes ejemplos, a partir de las producciones, cada variable sintáctica genera un lenguaje formado por cadenas de símbolos terminales. El conjunto de cadenas generadas por el símbolo inicial  $S$  será el lenguaje aceptado por la GLC  $G$ . En la siguiente sección se establece la definición precisa.

**Ejemplo 47.** Sea  $G$  la GLC con conjunto de variables sintácticas  $\{S\}$  con alfabeto de símbolos terminales  $T = \{0, 1\}$  y lista de producciones:

$$\begin{array}{l} S \rightarrow 01 \\ | \\ 0S1 \end{array}$$

Las producciones son una forma simplificada de escribir el siguiente proceso inductivo:

- R. INICIAL la cadena 01 es generada por  $S$ ;
- R. INDUTIVA si  $x$  es generada por  $S$  entonces  $0x1$  es generada por  $S$ .

Por tanto, en la primera iteración se genera la cadena 01, en la segunda 0011, y así sucesivamente en la  $n$ -ésima iteración el símbolo inicial  $S$  genera a la cadena  $0^n1^n$ . El conjunto generado por  $S$  es el conjunto  $L_{01} = \{0^n1^n : n > 0\}$ .

**Ejemplo 48.** En este ejemplo se presenta una GLC simplificada para ciertas secuencias del tipo IF-THEN-ELSE. Sea  $G = \{T, V, S, P\}$  la GLC con  $T$  igual al conjunto de símbolos de texto plano, conjunto de variables sintácticas  $V = \{S, C, E\}$  y con conjunto  $P$ :

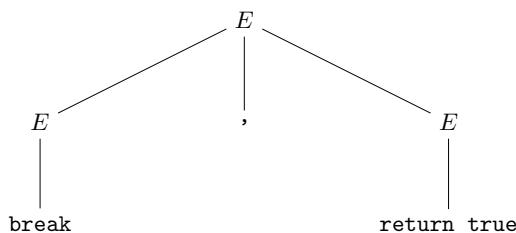
$$\begin{array}{l} S \rightarrow \text{IF } C \text{ THEN } S \\ | \\ \text{IF } C \text{ THEN } S \text{ ELSE } S \\ | \\ E \\ E \rightarrow \text{print "ok"} \\ | \\ \text{break} \\ | \\ \text{return true} \\ | \\ \text{return false} \\ | \\ E, E \\ C \rightarrow \text{false} \\ | \\ \text{true} \end{array}$$

Las producciones  $C \rightarrow \text{false} | \text{true}$  se traducen como que la variable sintáctica  $C$  genera a las cadenas `false` y `true`. Por otro lado, la variable sintáctica  $E$  genera a todas las posibles secuencias finitas de sentencias de ejecución que se pueden formar a partir de las sentencias básicas `print "ok"`, `break`, `return true` y `return false`. De hecho, las producciones con base  $E$  son una forma simplificada de escribir el siguiente proceso inductivo:

- R. INICIAL las cadenas `print "ok"`, `break`, `return true` y `return false` son generadas por  $E$ ;

R. INDUCTIVA (producción  $E \rightarrow E, E$ ) si  $x$  es generada por  $E$  y  $y$  es generada por  $E$ , entonces  $w = x, y$  es generada por  $E$ .

Por ejemplo la cadena `break,return true` es generada en dos iteraciones. Este proceso se puede representar con el siguiente árbol:



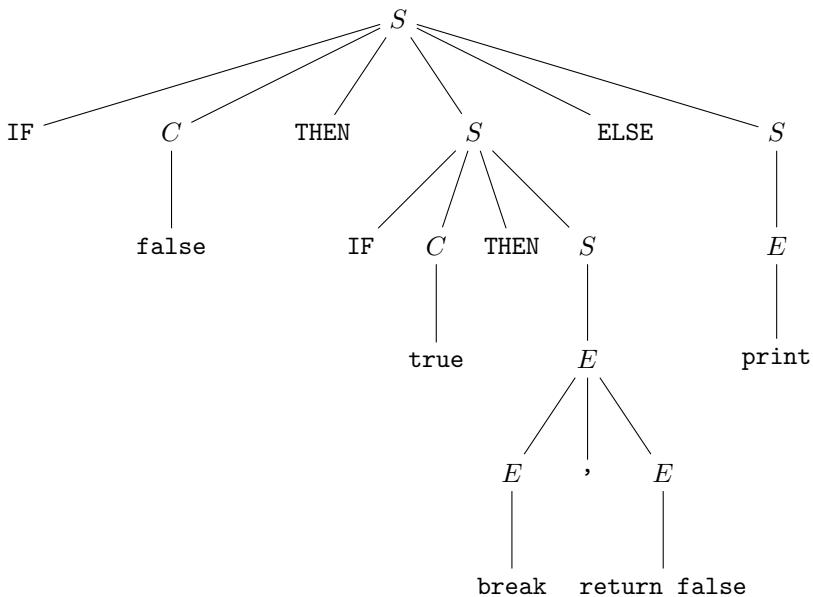
El símbolo inicial  $S$  genera las cadenas que corresponden a las sentencias del tipo IF-THEN-ELSE correctas con las condiciones generadas por  $C$  y las sentencias de ejecución generadas con  $E$ . Por ejemplo,  $S$  genera la cadena que representa el siguiente programa:

```

IF false THEN
  IF true THEN
    break,
    return false
  ELSE
    print
  
```

El proceso iterativo se puede representar con un árbol donde la distancia de la palabra a la raíz  $S$  es igual al número de iteraciones que son necesarias para generar o derivar la palabra a partir de la raíz, ver figura 6.1.

**Ejemplo 49.** La forma Backus-Naur (BNF, por sus siglas en inglés) es una sintaxis popular para describir una GLC. Cada uno de las variables sintácticas se escribe entre corchetes angulares. En las producciones se sustituye  $\rightarrow$  por  $::=$ . Los símbolos terminales son los valores que no están encerrados entre corchetes angulares y que pueden incluir secuencias de escape —por ejemplo, `\n`, `\f`, `\x30`, `\x42`, etcétera— asociadas a un código ASCII. La barra vertical o pleca se mantiene con el mismo sentido que en la notación de la definición formal. El siguiente ejemplo corresponde a la sintaxis de un comando `duplica` tipo `shell` que recibe un número entero:



**Fig. 6.1** Ejemplo de árbol asociado a la GLC para secuencias del tipo IF-THEN-ELSE.

```

<duplica>      ::= duplica<espacioblanco><número><findelínea>
<espacioblanco> ::= \s
                   | \s<espacioblanco>
<número>        ::= <dígito>
                   | <dígito><número>
<dígito>         ::= 0|1|2|3|4|5|6|7|8|9
<findelínea>    ::= \n
                   | <espacioblanco>\n
  
```

Existen muchas variantes de la sintaxis anterior por ejemplo la forma extendida de Backus-Naur (EBNF por sus siglas en inglés) la cual proporciona una serie de mejoras y simplificaciones. Al requerir comillas alrededor de valores literales que representan símbolos terminales, es posible prescindir de los corchetes angulares. Además se usan comas para definir la concatenación. Por lo tanto, podemos expresar nuestro ejemplo anterior de una forma más simple en formato EBNF:

```

duplica      = "duplica", espacioblanco, número, findelínea
espacioblanco = "\s", {"\s"}
número       = dígito, {dígito}
dígito        = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
findelínea   = "\n", [espacioblanco]

```

Si **a** es un valor que representa un símbolo terminal o una variable sintáctica, la producción **variable** = **{a}** significa que la **variable** genera al conjunto de cero o más repeticiones de **a**, por tanto tiene asociada a la expresión regular **a\***. En el ejemplo, la producción **espacioblanco**=""\s", {"\s"} denota el hecho que **espacioblanco** genera a las cadenas aceptadas por la expresión regular **\s(\s)\***. A su vez, la producción **variable** = **[a]** tiene asociada a la expresión regular **a|ε** (o **a?** en notación de Unix). En el ejemplo, la producción **findelínea**="\n", [espacioblanco] denota el hecho que **findelínea** genera las cadenas asociadas a la expresión regular **\n(espacioblanco?)**. Otra variante es la forma aumentada de Backus-Naur (ABNF por sus siglas en inglés) la cual permite, entre otras cosas, usar notación simplificada para rango de valores y hacer repeticiones específicas y hacer comentarios después de ;. La ABNF usa la diagonal / en lugar de la pleca y no requiere comas para la concatenación, solo hace falta añadir un espacio. Así por ejemplo, la producción en EBNF:

```
dígito      = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
```

se escribiría en ABNF:

```
dígito      = "0"/"1"/"2"/"3"/"4"/"5"/"6"/"7"/"8"/"9"
```

y equivalentemente:

```
dígito      = %x30-39 ; dígitos del 0 al 9
```

Otro ejemplo es:

```

letras      = %x41-5A
            / %x61-7A ; letras A-Z y a-z
cualquiera = %x01-7F ; cualquier carácter ASCII salvo NULL

```

El formato ABNF también tiene una notación simplificada para repetir elementos, de hecho **\*a** se usa en lugar de **{a}** que como vimos significa cero o más repeticiones de **a**. En general **n\*m\*a** significa «de **n** (por defecto cero) hasta **m** (por defecto infinitas) repeticiones». Además **n\*n\*a** se abrevia **na** (e. g. **3dígito**

representa a los números de 3 dígitos). Siguiendo con el ejemplo, la producción en formato EBNF:

```
findelínea      = "\n", [espacioblanco]
```

se puede escribir en forma muy limpia en formato ABNF:

```
findelínea      = "\n" 0*1espacioblanco
```

Como comentario adicional queda decir que una producción en formato ABNF del tipo:

```
token          = "then"
```

significa que la variable sintáctica `then` genera por defecto las cadenas `then`, `THen`, `THEn`, `tHeN`, etcétera. Para que solo genere `then` se escribe:

```
token          = %s"then" ; caso sensible
```

En resumen, la gramática quedaría en forma ABNF así:

```
duplica        = %s"duplica" espacioblanco número findelínea
espacioblanco  = 1*"s"
número         = 1*dígito
dígito          = %x30-39
findelínea     = "\n" 0*1espacioblanco
```

Otras cuestiones útiles que simplifican la escritura como la agrupación de términos también son posibles en el formato ABNF, aquí se han presentado solo algunas de las especificaciones para dar una idea al lector.

### **Lista de ejercicios de la sección 6.1**

*Ejercicio 102.* Para cada uno de los siguientes conjuntos construye una GLC que los genere:

1.  $L_a = \{a^{2m+1} : m \geq 0\}$ .
2. El conjunto de cadenas binarias con una cantidad impar de ceros y una cantidad impar de unos.
3.  $L_{\text{son}}$  el lenguaje del ejemplo 10.
4.  $L_{x\bar{x}}$  el lenguaje del ejemplo 38.

### 5. $L_{[a]}$ ejemplo 37.

*Ejercicio 103.* Para cada uno de los siguientes conjuntos de cadenas, construye una GLC que los genere:

1.  $\{0^m 1^n 2^n : n, m > 0\}$ ,
2.  $\{0^n 1^n 2^m : n, m > 0\}$ ,
3.  $\{0^n 1^m 2^k : n = m \text{ ó } m = k\}$
4.  $\{0^n 1^m 0^{2n} : n > 0, m > 0\}$ .

*Ejercicio 104.* Diseñe una GLC que genere cadenas de texto que correspondan a secuencias válidas de una gramática escrita en formato BNF.

*Ejercicio 105.* Busque en Internet la GLC correspondiente a las secuencias válidas de JSON (*JavaScript Object Notation*). Escríbala en formato ABNF. Genere algunas cadenas correctas según la sintaxis de JSON.

## 6.2 Derivaciones y árboles de derivación

Sea  $G = (T, V, S, P)$  una GLC. El objetivo de esta sección es definir de manera formal el lenguaje aceptado por  $G$ , que como se vio en la sección anterior es el conjunto generado por  $S$ . Con este fin se establece primero la versión formal del sistema de sustituciones de la sección anterior. Se define la relación binaria  $\Rightarrow$  entre dos cadenas de  $(T \cup V)^*$  de la siguiente forma:

**Definición**  $\Rightarrow$  y  $\stackrel{*}{\Rightarrow}$ . Si  $A \rightarrow \gamma$  es una producción de  $P$  entonces para cualesquiera dos elementos  $\alpha, \beta \in (T \cup V)^*$ :

$$\alpha A \beta \Rightarrow \alpha \gamma \beta.$$

En este caso se lee « $\alpha A \beta$  deriva a  $\alpha \gamma \beta$ ». En particular si  $A \rightarrow \gamma$  es una producción de  $P$  entonces  $A \Rightarrow \gamma$ . La clausura de  $\Rightarrow$  es a su vez una relación binaria sobre  $(T \cup V)^*$  que se establece de forma inductiva como sigue:

- R. INICIAL si  $\alpha \Rightarrow \beta$  entonces  $\alpha \stackrel{*}{\Rightarrow} \beta$  en 1 paso;
- R. INDUCTIVA si  $\alpha \Rightarrow \beta$  y  $\beta \stackrel{*}{\Rightarrow} \gamma$  en  $n$  pasos, entonces  $\alpha \stackrel{*}{\Rightarrow} \gamma$  en  $n + 1$  pasos.

Por convención,  $\alpha \stackrel{*}{\Rightarrow} \alpha$  en 0 pasos. A una expresión del tipo  $\alpha \stackrel{*}{\Rightarrow} \beta$  se le llama *derivación*.

Con esta notación es posible definir con precisión al lenguaje aceptado por  $G$  como el conjunto de cadenas generadas por  $S$ :

**Definición L(G).** Sea  $G$  una GLC. Al conjunto  $L(G)$  de cadenas  $w$  de símbolos terminales tales que  $S \xrightarrow{*} w$  se le llama *lenguaje aceptado por G*.

**Ejemplo 50.** Sea  $G$  una GLC con producciones:

$$\begin{array}{l} S \rightarrow S + S \\ | \quad S \times S \\ | \quad a \\ | \quad b \\ | \quad c \end{array}$$

La cadena  $a + a \times a \in L(G)$  pues  $S \xrightarrow{*} a + a \times a$ . La siguiente tabla detalla el porqué esto es cierto:

$\alpha A \beta \Rightarrow \alpha \gamma \beta$	$  A \rightarrow \gamma$	$\alpha$	$\beta$
$S \Rightarrow S + S$	$  S \rightarrow S + S$	$\varepsilon$	$\varepsilon$
$S + S \Rightarrow S + S \times S$	$  S \rightarrow S \times S$	$S +$	$\varepsilon$
$S + S \times S \Rightarrow a + S \times S$	$  S \rightarrow a$	$\varepsilon$	$+S \times S$
$a + S \times S \Rightarrow a + a \times S$	$  S \rightarrow a$	$a +$	$\times S$
$a + a \times S \Rightarrow a + a \times a$	$  S \rightarrow a$	$a + a \times$	$\varepsilon$

De manera práctica se puede pensar simplemente como una secuencia de sustituciones que se pueden escribir así:

$$S \Rightarrow S + S \Rightarrow S + S \times S \Rightarrow a + S \times S \Rightarrow a + a \times S \Rightarrow a + a \times a.$$

**Ejemplo 51.** Sea  $G$  la GLC del ejemplo 47. El lenguaje aceptado por  $G$  es el conjunto de cadenas que deriva  $S$ , en este caso,  $L(G) = L_{01} = \{0^n 1^n : n > 0\}$ . Para todo  $n > 0$  se cumple que:

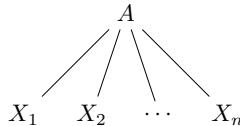
$$P(n) : \text{si } w = 0^n 1^n \text{ si y solo si } S \xrightarrow{*} w \text{ en } n \text{ pasos.}$$

Sea  $w = 01$ , se tiene que  $S \rightarrow 01$  y por tanto  $S \xrightarrow{*} 01$  en 1 paso. Por otro lado, si  $S \xrightarrow{*} w$  en 1 paso entonces  $S \rightarrow w$ , luego  $w = 01$ . En resumen, se cumple  $P(1)$ .

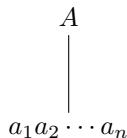
Se supone cierta  $P(n)$  (hipótesis de inducción). Sea  $w = 0^{n+1}1^{n+1}$  por hipótesis de inducción  $S \xrightarrow{*} 0^n1^n$  en  $n$  pasos, luego  $0S1 \xrightarrow{*} 00^n1^n1$ , ver ejercicio 106. Por definición de  $G$ ,  $S \Rightarrow 0S1$ . Por tanto  $S \xrightarrow{*} 0^{n+1}1^{n+1}$  en  $n+1$  pasos. Por otro lado, si  $S \xrightarrow{*} w$  en  $n+1$  pasos, entonces  $S \Rightarrow 0S1 \xrightarrow{*} w$  en  $n+1$  pasos y por tanto  $w = 0x1$  y  $S \xrightarrow{*} x$  en  $n$  pasos. Por hipótesis de inducción  $x = 0^n1^n$  y eso implica que  $w = 0^{n+1}1^{n+1}$ .

Como se vio en la sección anterior existe una forma gráfica de representar las derivaciones por medio de árboles. Estos árboles conocidos como *árboles de derivación* (*parse trees* en inglés) es una estructura de datos que en el marco de la teoría de los compiladores representa al programa fuente. La definición formal es la siguiente. Un árbol de derivación para una GLC  $G = (T, V, S, P)$  es un árbol tal que:

- cada nodo interior —nodos con hijos— se etiqueta con una variable sintáctica;
- cada hoja —nodos sin hijos— se etiqueta o bien por una variable sintáctica o por un símbolo terminal o por  $\varepsilon$ . Sin embargo, si una hoja es etiquetada por  $\varepsilon$  entonces tiene que ser el único hijo de su padre;
- un nodo interior es etiquetado con  $A$  y sus hijos  $X_1, X_2, \dots, X_n$ , haciendo la asignación de izquierda a derecha, lo anterior se representa como:



si y solo si  $A \rightarrow X_1X_2 \dots X_n$  es una producción de  $P$ . Si  $A \rightarrow a_1a_2 \dots a_n$  con cada  $a_i \in T$ ,  $i = 1, 2, \dots, n$  se dibuja en este texto de forma simplificada como sigue:



- a la etiqueta del primer nodo se le llama *raíz* del árbol.

A la concatenación de las etiquetas de las hojas de un árbol de derivación, de izquierda a derecha, se le llama *rendimiento* del árbol. Particularmente importantes son los rendimientos  $w$  formados por símbolos terminales de árboles de derivación cuya raíz es el símbolo inicial  $S$ , pues en ese caso  $S \xrightarrow{*} w$  [propiedad 6].

Más aún, dada una derivación  $A \xrightarrow{*} w$  se puede construir un árbol de derivación con raíz  $A$  y rendimiento  $w$  [ejercicio 112]. En conclusión:

**Teorema 10.** *Sea  $G$  una GLC con símbolo inicial  $S$ . Son equivalentes:*

- $S \xrightarrow{*} w$ ;
- existe un árbol de derivación con raíz  $S$  y rendimiento  $w$ .

A cada nodo  $X$  de un árbol de derivación se le puede asignar su distancia a la raíz  $A$ , esto es, el número de nodos conectados que constituyen el camino de  $X$  a  $A$  menos 1. Se define la *altura* del árbol como el máximo sobre el conjunto de distancias de cada nodo a la raíz.

**Ejemplo 52.** Un árbol que tiene solo a su raíz tiene altura 0. El árbol de la figura 6.1 tiene altura 5.

Sin lugar a duda, estos conceptos permiten comprobaciones más elaboradas. Una muestra es el siguiente ejemplo.

**Ejemplo 53.** Sea  $G$  la GLC con producciones:

$$\begin{array}{lcl} S & \rightarrow & \varepsilon \\ & | & \\ & SS & \\ & | & \\ & [S] & \end{array}$$

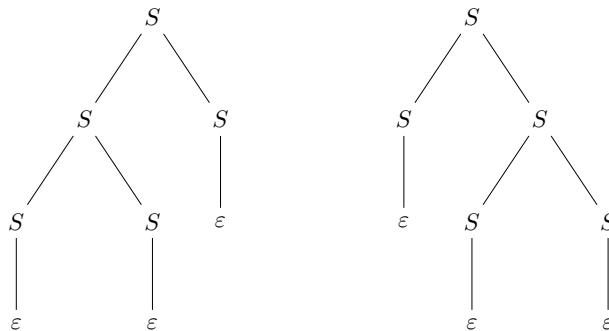
A una cadena de derivación concreta  $S \xrightarrow{*} \varepsilon$  se le puede asignar un árbol de derivación con raíz  $S$  y rendimiento  $\varepsilon$ . Por ejemplo a la cadena  $S \Rightarrow \varepsilon$  le corresponde naturalmente el árbol:



y viceversa, al árbol de arriba le corresponde la cadena  $S \Rightarrow \varepsilon$ . Note que no hay una sola manera de asignarle un árbol al hecho « $S \xrightarrow{*} \varepsilon$ » incluso fijando la secuencia de derivaciones. Por ejemplo, dentro de esta secuencia de derivaciones:

$$S \Rightarrow SS \Rightarrow SSS \Rightarrow S\varepsilon S \Rightarrow \varepsilon\varepsilon S \Rightarrow \varepsilon\varepsilon\varepsilon,$$

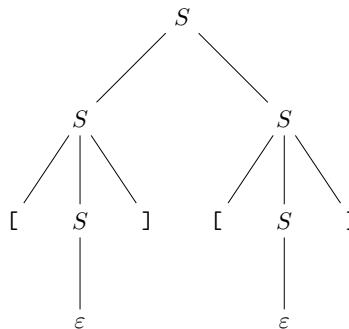
la derivación  $SS \Rightarrow SSS$  arroja dos posibilidades de ramificación, pues no queda claro si se ha reemplazado la primera  $S$  o la segunda  $S$ , por tanto se tienen dos árboles de derivación:



Consideré ahora la secuencia de derivaciones:

$$S \Rightarrow SS \Rightarrow [S]S \Rightarrow [S][S] \Rightarrow [\varepsilon][S] \Rightarrow [\varepsilon][\varepsilon].$$

A esta secuencia se le puede asignar de manera natural el árbol de derivación:



y viceversa, al árbol de arriba se le puede asignar la secuencia de derivaciones de arriba, pero además otras, por ejemplo:

$$S \Rightarrow SS \Rightarrow [S]S \Rightarrow [\varepsilon]S \Rightarrow [\varepsilon][S] \Rightarrow [\varepsilon][\varepsilon]$$

ó:

$$S \Rightarrow SS \Rightarrow S[S] \Rightarrow S[\varepsilon] \Rightarrow [S][\varepsilon] \Rightarrow [\varepsilon][\varepsilon]$$

### **Lista de ejercicios de la sección 6.2**

*Ejercicio 106.* Pruebe que  $A \xrightarrow{*} \gamma$  si y solo si para cualesquiera  $\alpha, \beta \in (T \cup V)^*$ ,  $\alpha A \beta \xrightarrow{*} \alpha \gamma \beta$ . Encuentre un contraejemplo para demostrar que en general no es cierto que si  $\alpha A \beta \xrightarrow{*} \alpha \gamma \beta$  para algunas cadenas  $\alpha$  y  $\beta$  entonces  $A \xrightarrow{*} \gamma$ .

*Ejercicio 107.* Compruebe usando derivaciones que las GLC que construyó en el ejercicio 102 son correctas.

*Ejercicio 108.* Sean  $G_1$  y  $G_2$  dos GLC. Si  $L = L(G_1)$  y  $M = L(G_2)$  diseñe una nueva GLC que acepte, respectivamente, como lenguaje a:

1.  $L \cup M$ ,
2.  $LM$ ,
3.  $L^*$ .

¿Es posible construir una GLC que acepte a  $L \cap M$  como lenguaje?

*Ejercicio 109.* Una *gramática regular* (por la derecha) es una GLC que admite solo producciones de la forma:

$$\begin{array}{ll} A & \rightarrow aB \\ C & \rightarrow b \\ D & \rightarrow \varepsilon \end{array}$$

donde  $A, B, C, D$  son variables sintácticas y  $a, b$  son símbolos terminales. Encuentre una gramática regular para los siguientes conjuntos:

1. El conjunto de cadenas binarias que terminan en 111.
2.  $L_a = \{a^{2m+1} : m \geq 0\}$ .
3. El conjunto de cadenas binarias con una cantidad impar de ceros y una cantidad impar de unos.
4.  $L_{\text{son}}$  el lenguaje del ejemplo 10.

*Ejercicio 110.* Demuestre que un lenguaje es regular si y solo si es aceptado por una gramática regular. Sugerencia: busque en la literatura la construcción de un AFND- $\varepsilon$  a partir de una gramática regular y viceversa.

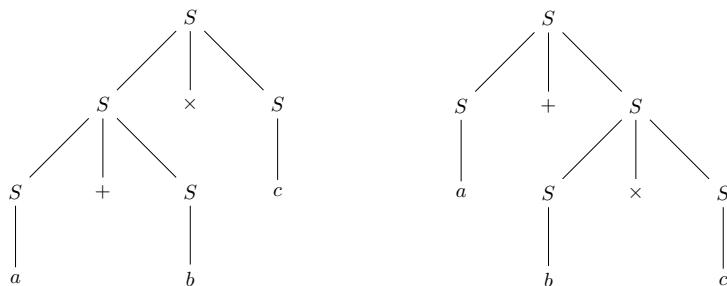
*Ejercicio 111.* Una *gramática sin restricciones* es una tupla  $G = (T, V, S, P)$  con  $T, V$  y  $S$  igual que en la definición de una GLC pero tal que  $P$  admite producciones más generales de la forma  $\alpha \rightarrow \beta$ , donde  $\alpha$  y  $\beta$  son cadenas de  $(T \cup V)^*$  y  $\alpha$  contiene al menos una variable sintáctica. El lenguaje aceptado por una gramática sin restricciones es el conjunto de cadenas de símbolos terminales que se derivan de  $S$ . Se dice que un lenguaje es *recursivamente enumerable* si existe una gramática sin restricciones que lo acepta como lenguaje. De un ejemplo para demostrar que para gramáticas sin restricciones, en general *no* es cierto que  $S \Rightarrow xAy \xrightarrow{*} w \in T^*$  implique que  $w$  sea de la forma  $w = xzy$  para alguna cadena de símbolos terminales  $z$ .

## 6.3 Gramáticas ambiguas

A partir de una gramática  $G$  asociada a un lenguaje de programación, es posible establecer un analizador sintáctico. Esto es, un programa que recibe un programa  $w$  y regresa —en caso que  $w$  sea un programa sintácticamente correcto— «el» árbol de derivación correspondiente a  $w$ . El árbol de derivación asociado a una cadena  $w \in L(G)$  representa una orden clara que debe ser ejecutada. Por tanto, lo deseable es que a cada cadena  $w$  le corresponda un solo árbol de derivación. Cuando esto no ocurre, se dice que la gramática es ambigua:

**Definición.** Sea  $G$  una GLC. Se dice que  $G$  es *ambigua* si existe una cadena  $w$  en  $L(G)$  con dos o más árboles de derivación con rendimiento  $w$ .

**Ejemplo 54.** La gramática del ejemplo 50 es ambigua, puesto que existen dos árboles de derivación para la cadena  $a + b \times c$ :



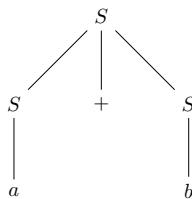
La orden asociada al primer árbol es «ejecuta  $a + b$  y al resultado multiplícalo por  $c$ » mientras la del segundo es «ejecuta  $b \times c$  y luego ejecuta la suma  $a$  más el resultado anterior». Por supuesto  $(1+2) \times 3 = 9$  no es igual a  $1 + (2 \times 3) = 7$ . En otras palabras, dos árboles de derivación para la cadena-orden  $a + b \times c$  significa que la orden a ejecutar no es precisa.

**Ejemplo 55.** El teorema 10 pudiera sugerir que si existen dos o más secuencias de derivaciones  $S \xrightarrow{*} w$  entonces existen dos o más árboles de derivación distintos con raíz  $S$  y rendimiento  $w$ . Sin embargo, una cadena  $w$  puede ser derivada a partir de  $S$  y existir un solo árbol de derivación correspondiente. Por ejemplo, para la GLC del ejemplo 50 existen dos secuencias de derivaciones para la cadena  $a + b$ :

$$S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + b, \text{ y}$$

$$S \Rightarrow S + S \Rightarrow S + b \Rightarrow a + b.$$

No obstante, existe un solo árbol de derivación para esta cadena:



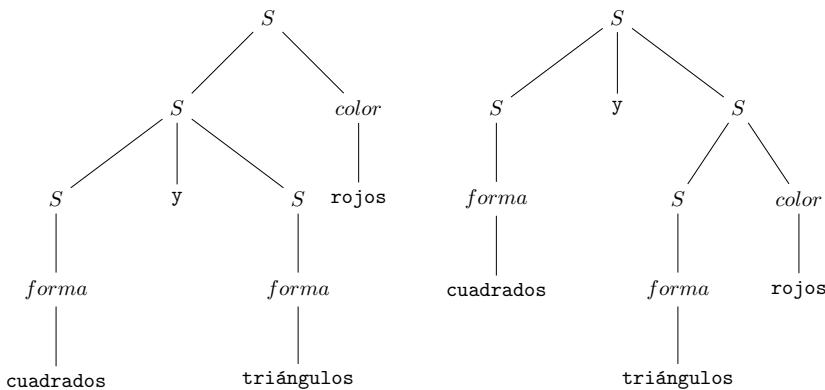
**Ejemplo 56.** Sea una gramática con producciones:

$S$	$\rightarrow S \text{ y } S$
	$S \text{ color}$
	$forma$
$color$	$\rightarrow \text{rojos}$
	$\text{verdes}$
	$\text{amarillos}$
$forma$	$\rightarrow \text{cuadrados}$
	$\text{triángulos}$
	$\text{círculos}$

Esta gramática es ambigua puesto que existen dos árboles de derivación para la cadena **cuadrados y triángulos rojos**, como se muestra en la figura 6.2. Cada árbol tiene un significado distinto; el primero se interpreta como «‘cuadrados y triángulos’ rojos» y el segundo como «‘cuadrados’ y ‘triángulos rojos’».

Hasta ahora se ha visto que es deseable que exista para una cadena un solo árbol de derivación. Además se ha visto que pueden existir varias secuencias de derivaciones para una misma cadena pero con un solo árbol de derivación. Lo que se busca ahora es dar una versión del teorema 10 que caracterice cuando una cadena tiene asociado un solo árbol de derivación. Con este fin se consideran aquellas derivaciones donde el reemplazo se hace considerando la variable sintáctica que se encuentra *más-a-la-izquierda*. Como primer paso se ha de tomar en cuenta la propiedad 6.

**Propiedad 6.** A partir de un árbol de derivación con raíz  $S$  y rendimiento  $w$  se puede construir una secuencia de derivaciones más-a-la-izquierda  $S \xrightarrow{*} w$ .

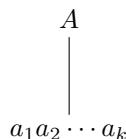


**Fig. 6.2** Dos árboles de derivación con el mismo rendimiento

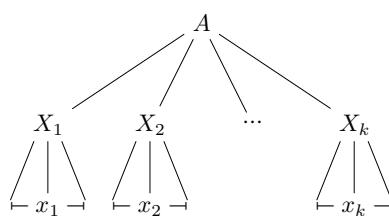
Prueba. Esta propiedad es un caso particular de la siguiente afirmación sobre números naturales:

$P(n)$  : si existe un árbol de derivación con raíz  $A$  y rendimiento  $w$  y altura menor o igual a  $n$ , para una variable sintáctica  $A$ , entonces existe una derivación más-a-la-izquierda  $A \xrightarrow{*} w$ .

Si el árbol es de altura 1 y  $w = a_1 a_2 \cdots a_k$  entonces es de la forma:



por tanto  $A \rightarrow a_1 a_2 \cdots a_k$  debe ser una producción de la GLC, luego  $A \rightarrow w$  es de forma trivial una derivación más-a-la-izquierda de un solo paso, es decir, se satisface  $P(1)$ . Se supone cierta  $P(n)$  (hipótesis de inducción) y que existe un árbol con raíz  $A$  y rendimiento  $w$  de altura  $n + 1$ . Este árbol es de la forma:



donde  $w = x_1x_2 \cdots x_k$  y cada árbol de derivación con raíz  $X_i$  y rendimiento  $x_i$  tiene una altura menor o igual a  $n$ . La hipótesis de inducción nos garantiza la existencia de una derivación más-a-la-izquierda  $X_i \stackrel{*}{\Rightarrow} x_i$ . Si el árbol de derivación con raíz  $X_i$  es de altura 0, significa que  $X_i = x_i$  es un símbolo terminal, en este caso  $X_i \stackrel{*}{\Rightarrow} x_i$  en 0 pasos. Por otro lado  $A \stackrel{*}{\Rightarrow} X_1X_2 \cdots X_k$  es obviamente una derivación más-a-la-izquierda. Por tanto, se tiene la siguiente secuencia de derivaciones más-a-la-izquierda:

$$A \stackrel{*}{\Rightarrow} X_1X_2 \cdots X_k \stackrel{*}{\Rightarrow} x_1X_2 \cdots X_k \stackrel{*}{\Rightarrow} x_1x_2 \cdots X_k \stackrel{*}{\Rightarrow} \cdots \stackrel{*}{\Rightarrow} x_1x_2 \cdots x_k = w.$$

Con esto se prueba  $P(n + 1)$ . □

Como consecuencia del teorema 10 y la propiedad 6 se tiene la siguiente propiedad:

**Propiedad 7.** Si existe una secuencia de derivaciones  $S \stackrel{*}{\Rightarrow} w$  entonces existe una secuencia de derivaciones más-a-la-izquierda  $S \stackrel{*}{\Rightarrow} w$ .

**Ejemplo 57.** Se consideran los árboles de derivación del ejemplo 54. La derivación más-a-la-izquierda asociada al árbol de la izquierda es la siguiente:

$$S \Rightarrow S \times S \Rightarrow S + S \times S \Rightarrow a + S \times S \Rightarrow a + b \times S \Rightarrow a + b \times c.$$

La derivación más-a-la-izquierda asociada al árbol de la derecha a su vez es:

$$S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S \times S \Rightarrow a + b \times S \Rightarrow a + b \times c.$$

Se ha de notar que la construcción de la prueba de la propiedad 6 arroja *distintas derivaciones más-a-la-izquierda si se parte de árboles de derivación distintos*. Además, dos derivaciones más-a-la-izquierda conducen a la construcción de dos árboles de derivación distintos [ejercicio 112]. Resumiendo:

**Teorema 11.** Para una GLC con símbolo inicial  $S$ , las siguientes afirmaciones son equivalentes para una cadena  $w$  de símbolos terminales:

- existe una única derivación más-a-la-izquierda  $S \stackrel{*}{\Rightarrow} w$ ;
- existe un único árbol de derivación con raíz  $S$  y rendimiento  $w$ .

Lamentablemente no existe un algoritmo que reciba una GLC ambigua y regrese una GLC no-ambigua que acepte el mismo lenguaje, más aún, se sabe que

no existe algún algoritmo que diga siquiera si una GLC dada es ambigua o no. Por otro lado, es bien conocida la existencia de lenguajes aceptados por una GLC —llamados *inherentemente ambiguos*— para los cuales es imposible construir una gramática no-ambigua. Un ejemplo de este tipo de lenguajes es el conjunto  $\{0^n 1^m 2^k : n = m \text{ ó } m = k\}$  [6, p. 301]. No obstante, en los casos particulares es posible eliminar la ambigüedad como se muestra en el siguiente ejemplo:

**Ejemplo 58.** Sea  $G$  la GLC del ejemplo 53. Claramente  $G$  es ambigua, puesto que existen una infinidad de derivaciones más-a-la-izquierda para la cadena vacía:

$$S \Rightarrow \varepsilon, S \Rightarrow SS \Rightarrow \varepsilon S \Rightarrow \varepsilon \varepsilon, \text{ etcétera}$$

Se propone la GLC  $G'$ :

$$\begin{array}{rcl} S & \rightarrow & \varepsilon \\ & | & \\ & [S]S & \end{array}$$

Es fácil verificar que  $L(G) = L(G')$  [ejercicio 113]. Además  $G'$  es no-ambigua, pues para todo  $n > 0$  se satisface la siguiente afirmación:

$P(n)$ : si  $S \xrightarrow{*} w$  en una cantidad de pasos menor o igual a  $n$ , entonces existe una única derivación más-a-la-izquierda que deriva  $w$  a partir de  $S$ .

La prueba de  $P(1)$  es trivial, pues la única cadena derivada en 1 paso es la cadena vacía, a través de la única derivación  $S \Rightarrow \varepsilon$ . Se supone que se cumple  $P(n)$  (hipótesis de inducción) y que existe una derivación  $S \xrightarrow{*} w$  de  $n+1$  pasos. Por la propiedad 7 se puede suponer que es una derivación más-a-la-izquierda. Esta derivación tiene que ser de la siguiente forma:

$$S \Rightarrow [S]S \xrightarrow{*} [x]y = w, \quad (6.1)$$

donde se involucra a las derivaciones más-a-la-izquierda  $S \xrightarrow{*} x$  y  $S \xrightarrow{*} y$  de un número de pasos menos o igual a  $n$ . Por hipótesis de inducción esas derivaciones más-a-la-izquierda son únicas, luego la derivación (6.1) también lo es.

En la práctica las gramáticas no son tan simples, pero existen algunos métodos que se utilizan en casos particulares para remover ambigüedad. Estas técnicas incluyen añadir reglas de precedencia y jerarquía para obligar una sola interpretación. El lenguaje de la nueva gramática no es exactamente el mismo, pero para fines de dejar claro el significado de la orden de ejecución es suficiente. El siguiente ejemplo tiene como fin ilustrar estos puntos.

**Ejemplo 59.** Se considera la GLC con lista de producciones de  $P$ :

$$\begin{array}{lcl} S & \rightarrow & S \times S \\ & | & S + S \\ & | & N \\ N & \rightarrow & ON \\ & | & 1N \\ & | & 0 \\ & | & 1 \end{array}$$

Existen dos problemas principalmente en esta GLC. El primero es con respecto a la precedencia de los operadores  $\times$  y  $+$  análogamente a lo que se vio en el ejemplo 54; si se siguen las reglas aritméticas usuales el operador  $\times$  tiene que tener precedencia sobre  $+$ . El segundo es respecto a la secuencia de operaciones con la misma precedencia. Por ejemplo, la cadena  $0 + 1 + 1$  refleja una orden de ejecución imprecisa aunque el resultado sea el mismo, se ha de indicar con exactitud la manera en que se va a efectuar la suma. De forma convencional operaciones de este tipo se hacen de izquierda a derecha. Por otro lado, existe la posibilidad de que primero se requieran sumar dos términos y luego multiplicar el resultado por otro; en ambos casos una solución natural es añadir paréntesis al conjunto de símbolos terminales para indicar precedencia. La GLC no-ambigua que se sugiere es la siguiente, la justificación se deja como ejercicio para el lector [ejercicio 115]:

$$\begin{array}{lcl} S & \rightarrow & T \\ & | & S + T \\ T & \rightarrow & F \\ & | & T \times F \\ F & \rightarrow & N \\ & | & (S) \\ N & \rightarrow & ON \\ & | & 1N \\ & | & 0 \\ & | & 1 \end{array}$$

En el siguiente capítulo se establece una relación entre los APD y las GLC que no son inherentemente ambiguas. Más precisamente, la demostración del teorema 12 constituye una construcción de una gramática no-ambigua a partir de un APD.

### **Lista de ejercicios de la sección 6.3**

*Ejercicio 112.* Argumente porqué si  $A \xrightarrow{*} w$  para alguna variable sintáctica  $A$  y cadena  $w$ , existe un árbol de derivación con raíz  $A$  y rendimiento  $w$ . Además justifique el hecho de que dos derivaciones más-a-la-izquierda arrojan dos árboles de derivación distintos.

*Ejercicio 113.* Sea  $G'$  la GLC del ejemplo 58 y  $G$  la GLC del ejemplo 53. Prueba que  $L(G) = L(G')$ .

*Ejercicio 114.* Verifique que la siguiente GLC para el lenguaje  $L_{\bar{x}\bar{x}}$  es no-ambigua:

$$\begin{array}{lcl} S & \rightarrow & 0S0 \\ & | & 1S1 \\ & | & 00 \\ & | & 11 \end{array}$$

*Ejercicio 115.* Argumente porqué la GLC sugerida en el ejemplo 59 es no-ambigua.

*Ejercicio 116.* Desambigüe la GLC con producciones:

$$\begin{array}{lcl} S & \rightarrow & [S] \\ & | & (S) \\ & | & \{S\} \\ & | & SS \\ & | & \varepsilon \end{array}$$

*Ejercicio 117.* Decida si la GLC del ejemplo 48 es ambigua o no. De no serlo, desambíguela.

*Ejercicio 118.* Dedida si la siguiente GLC es ambigua o no:

$$\begin{array}{lcl} S & \rightarrow & 01 \\ & | & 10 \\ & | & 1S1 \\ & | & S11 \\ & | & 11S \\ & | & 0S0 \\ & | & S00 \\ & | & 00S \end{array}$$

# CAPÍTULO 7

---

## Lenguajes libres de contexto

---

En este capítulo se estudian las propiedades de los lenguajes libres de contexto, eso es, los lenguajes aceptados por una GLC. Las dos primeras secciones se reservan para presentar la equivalencia entre los autómatas a pila y una gramáticas libre de contexto [8]. Además se establecen las relaciones de los autómatas a pila deterministas con las gramáticas no-ambiguas. La tercera sección comienza con un teorema que resume los resultados de las secciones anteriores pero principalmente se centra en el estudio de un lema de bombeo para lenguajes libres de contexto [2], que al igual que en el caso de los lenguajes regulares su importancia radica en dar un criterio útil para verificar que un lenguaje no es libre de contexto. En la última sección se presenta el algoritmo CYK [5, 13, 24] para decidir si una cadena pertenece o no a una gramática dada.

### 7.1 De gramáticas a autómatas a pila

Sea  $G$  una GLC. En esta sección se establece un método para construir un AP tal que  $N(A) = L(G)$ . La idea es que el autómata a pila simule las derivaciones

más-a-la-izquierda de  $G$ . Antes de establecer el algoritmo general, en lo que se muestra a continuación se concreta esta idea a través de un ejemplo sencillo. Se considera concretamente la GLC con producciones:

$$\begin{array}{l} S \rightarrow A + A \\ A \rightarrow AA \\ | \\ 0 \\ | \\ 1 \end{array}$$

Evidentemente la cadena  $10 + 1$  es aceptada por esta GLC, esto se puede verificar a través de una secuencia de derivaciones más-a-la-izquierda:

$$S \Rightarrow A + A \Rightarrow AA + A \Rightarrow 1A + A \Rightarrow 10 + A \Rightarrow 10 + 1. \quad (7.1)$$

El plan es definir un AP de un solo estado  $q$  con alfabeto de símbolos de entrada igual al conjunto de símbolos terminales, en este caso  $\{0, 1, +\}$ , y alfabeto de pila igual al conjunto de símbolos terminales unión con el conjunto de variables sintácticas, en este caso  $\{0, 1, +, A, S\}$ . Las cadenas derivadas de  $S$  deben ser las mismas que aquellas que sean aceptadas por pila vacía. Se define el símbolo inicial de la pila como  $S$  de tal suerte que la descripción instantánea inicial es  $(q_0, 10 + 1, S)$ . Se desea que cada reemplazo efectuado en las derivaciones más-a-la-izquierda sean las que determinen los cambios en la pila, siempre que una variable sintáctica sea la que se encuentre hasta arriba de la pila. Por ejemplo, a la primera derivación de (7.1),  $S \Rightarrow A + A$ , se le asocia el movimiento de pila:

$$(q, 10 + 1, S) \vdash (q, 10 + 1, A + A)$$

así sucesivamente a las derivaciones siguientes  $A + A \Rightarrow AA + A \Rightarrow 1A + A$  se le asocian los movimientos de pila:

$$(q, 10 + 1, A + A) \vdash (q, 10 + 1, AA + A) \vdash (q, 10 + 1, 1A + A).$$

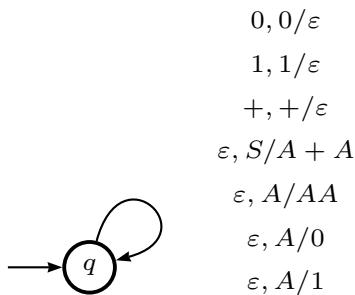
En el momento que aparece un símbolo terminal hasta arriba de la pila y este coincide con el primer elemento de la cadena, se eliminan ambos:

$$(q, 10 + 1, 1A + A) \vdash (q, 0 + 1, A + A).$$

Así sucesivamente se continúan con los movimientos de la pila asociados a las derivaciones más a la izquierda:

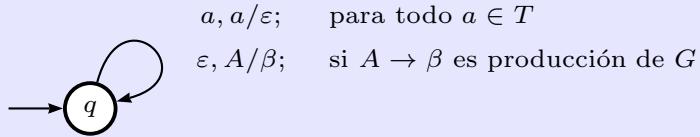
$$(q, 10 + 1, S) \vdash^* (q, 0 + 1, A + A) \vdash (q, 0 + 1, 0 + A) \vdash^* (q, 1, A) \vdash (q, 1, 1) \vdash (q, \varepsilon, \varepsilon).$$

En conclusión, el autómata que se busca es el siguiente:

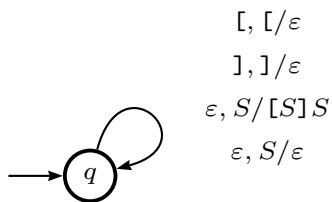


En general se tiene el siguiente algoritmo:

**Algoritmo GLP → AP.** A partir de una GLC  $G = (T, V, S, P)$  se construye un AP  $A = (\{q\}, T, T \cup V, \delta, q, S)$  que acepta del mismo lenguaje que  $G$  por pila vacía como sigue:



**Ejemplo 60.** Si se parte de la GLC del ejemplo 58 el AP que acepta el mismo lenguaje por pila vacía es:



En conclusión:

**Teorema 12.** Si  $L$  es un lenguaje aceptado por una GLC entonces es aceptado un AP por pila vacía.

*Prueba.* Sea  $G$  una GLC y sea  $A$  el AP de la construcción del cuadro de arriba. Se probará que en efecto  $N(A) = L(G)$ . Dada una cadena de derivaciones más-a-la-izquierda  $S \Rightarrow \gamma_1 \Rightarrow \gamma_2 \cdots \Rightarrow \gamma_k = w$  primero se demuestra inductivamente la siguiente afirmación para  $n = 1, 2, \dots, k$ :

$P(n)$ : si  $\gamma_n = x_n \alpha_n$  y  $x_n y_n = w$ , donde  $x_n$  es una cadena de símbolos terminales y la cadena  $\alpha_n$  comienza con una variable sintáctica o  $\alpha_n = \varepsilon$ , entonces  $(q, w, S) \stackrel{*}{\vdash} (q, y_n, \alpha_n)$ .

La demostración  $P(1)$  es fácil, pues en este caso  $\gamma_1 = S$  y solo se admiten las descomposiciones  $S = \varepsilon S$  y  $w = \varepsilon w$ , es decir  $x_1 = \varepsilon$ ,  $\alpha_1 = S$  y  $y_1 = w$ , por tanto  $P(1)$  es cierta, pues  $(q, w, S) \stackrel{*}{\vdash} (q, y_1, \alpha_1) = (q, w, S)$  en 0 pasos. Se supone que se cumple  $P(n)$  (hipótesis de inducción) y que  $\gamma_{n+1} = x_{n+1} \alpha_{n+1}$  y  $x_{n+1} y_{n+1} = w$  donde  $x_{n+1}$  es una cadena de símbolos terminales y  $\alpha_{n+1}$  comienza con una variable sintáctica o es la cadena vacía. Sea  $\gamma_n = x_n \alpha_n$  donde  $x_n$  es una cadena de símbolos terminales. Ya que  $\gamma_n \Rightarrow \gamma_{n+1}$ , existen dos casos a considerar:

- Si  $\alpha_n = A\xi$  entonces  $x_n A\xi \Rightarrow x_{n+1} \alpha_{n+1} = x_n \beta\xi$  donde  $A \rightarrow \beta$  es una producción de  $G$ . Luego existe una cadena de símbolos terminales  $\eta$  tal que  $x_{n+1} = x_n \eta$  y además la cadena  $y_n = \eta y_{n+1}$  satisface  $x_n y_n = w$ . Por hipótesis de inducción:

$$(q, w, S) \stackrel{*}{\vdash} (q, y_n, \alpha_n) = (q, \eta y_{n+1}, A\xi) \vdash (q, \eta y_{n+1}, \beta\xi) = (q, \eta y_{n+1}, \eta \alpha_{n+1}).$$

Por definición del AP,  $(q, \eta y_{n+1}, \eta \alpha_{n+1}) \stackrel{*}{\vdash} (q, y_{n+1}, \alpha_{n+1})$ , por tanto se tiene finalmente que  $(q, w, S) \stackrel{*}{\vdash} (q, y_{n+1}, \alpha_{n+1})$ .

- Si  $\alpha_n = \varepsilon$  significa que  $\gamma_n = w$ . Luego  $\gamma_{n+1} = w$  y por tanto  $\alpha_{n+1} = \varepsilon$  y  $y_{n+1} = y_n = \varepsilon$ . Por hipótesis de inducción se tiene que:

$$(q, w, S) \stackrel{*}{\vdash} (q, y_n, \alpha_n) = (q, y_{n+1}, \alpha_{n+1}) = (q, \varepsilon, \varepsilon).$$

Con esto se demuestra que toda cadena aceptada por la GLC  $G$  es aceptada por pila vacía por el AP  $A$ . Resta demostrar que toda cadena aceptada por pila vacía por  $A$  es aceptada por  $G$ . Con este fin se prueba la siguiente afirmación sobre números naturales, para toda variable sintáctica  $X$  y cadena de símbolos terminales  $w$ :

$Q(n)$ : si  $(q, w, X) \stackrel{*}{\vdash} (q, \varepsilon, \varepsilon)$  en una cantidad menor o igual a  $n$  movimientos, entonces  $X \stackrel{*}{\Rightarrow} w$ .

Si  $(q, w, X) \vdash (q, \varepsilon, \varepsilon)$  significa que  $X \rightarrow \varepsilon$  es una producción de  $G$  y  $w = \varepsilon$ . Se cumple trivialmente  $Q(1)$  pues  $X \stackrel{*}{\Rightarrow} \varepsilon$ . Se supone cierta  $Q(n)$  (hipótesis de

inducción) y que  $(q, w, X) \xrightarrow{*} (q, \varepsilon, \varepsilon)$  en  $n+1$  movimientos. El primer movimiento involucra por fuerza una producción del tipo  $X \rightarrow X_1X_2 \cdots X_k$ , es decir:

$$(q, w, X) \xrightarrow{} (q, w, X_1X_2 \cdots X_k) \xrightarrow{*} (q, \varepsilon, \varepsilon).$$

Sea  $w = x_1x_2 \cdots x_k$  una descomposición de  $w$  tal que  $(q, x_i x_{i+1} \cdots x_k, X_i) \xrightarrow{*} (q, x_{i+1} \cdots x_k, \varepsilon)$ ; en otras palabras, para cada  $i = 1, 2, \dots, k$ , la cadena  $x_i$  es la subcadena de  $w$  que se lee al eliminar  $X_i$  de la pila en el proceso de lectura en una cantidad de movimientos menor o igual a  $n$ . Por la propiedad 5 se tiene que  $(q, x_i, X_i) \xrightarrow{*} (q, \varepsilon, \varepsilon)$ . Si  $X_i$  es un símbolo terminal, por fuerza  $x_i = X_i$  y por tanto  $X_i \xrightarrow{*} x_i$  en 0 pasos. Por otro lado, si  $X_i$  es una variable sintáctica, por hipótesis de inducción  $X_i \xrightarrow{*} x_i$ . En conclusión:

$$X \Rightarrow X_1X_2 \cdots X_k \xrightarrow{*} x_1x_2 \cdots x_k = w.$$

□

### **Lista de ejercicios de la sección 7.1**

*Ejercicio 119.* Para cada una de las GLC vistas en el capítulo anterior, construye con el algoritmo visto en esta sección un AP que acepte el mismo lenguaje.

*Ejercicio 120.* De un ejemplo de una GLC no-ambigua y regular para la cual el procedimiento visto en esta sección arroje un AP que no-determinista.

*Ejercicio 121.* Sea  $G$  una GLC no-ambigua. Pruebe o de un contraejemplo de las siguientes afirmaciones:

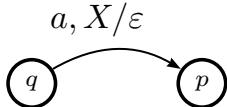
1. existe un APD que acepta a  $L(G)$  por pila vacía;
2. existe un APD que acepta a  $L(G)$  por estado de aceptación.

Repita el ejercicio pero bajo el supuesto que  $G$  es una GLC no-ambigua y regular.

## **7.2 De autómatas a pila a gramáticas**

En esta sección se establece un procedimiento general para construir una GLC a partir de un AP que acepta un lenguaje  $L$  por pila vacía. Además este método arroja una GLC no-ambigua si se parte de un APD. Dados un símbolo  $X$  del alfabeto de pila y dos estados  $p$  y  $q$ , la idea es asignarle una variable sintáctica  $[qXp]$  que derive las cadenas que son leídas al pasar de  $q$  a  $p$  eliminando a  $X$

del tope de la pila. Las producciones asociadas a cada variable sintáctica se establecen a partir de las transiciones que definen a  $A$ . Si  $A$  tiene el lazo:

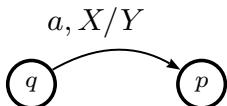


entonces para cualquier cadena del alfabeto de pila  $\alpha$  se tiene que:

$$(q, a, X\alpha) \vdash (p, \varepsilon, \alpha)$$

y por tanto al leer  $a$  se pasa de  $q$  a  $p$  y se elimina  $X$  del tope de la pila. Luego a la variable sintáctica  $[qXp]$  se le asigna la producción:

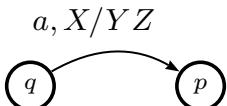
$$[qXp] \rightarrow a.$$



es un lazo de  $A$ , entonces cualquier estado  $r$ ,  $(p, x, Y\alpha) \stackrel{*}{\vdash} (r, \varepsilon, \alpha)$  implica  $(q, ax, X\alpha) \stackrel{*}{\vdash} (r, \varepsilon, \alpha)$  pues  $(q, ax, X\alpha) \vdash (p, x, Y\alpha)$ . En otras palabras, si  $x$  es leída al eliminar a  $Y$  del tope de la pila y pasar de  $q$  a  $r$  entonces al leer  $ax$  se borra  $X$  de hasta arriba de la pila y se pasa de  $q$  a  $r$ . Por tanto, se considera para cada estado  $r$  la producción:

$$[qXr] \rightarrow a[pYr].$$

Si  $A$  tiene el lazo:



para cualesquiera dos estados  $r_1$  y  $r_2$ , si al leer  $x_1$  se pasa  $p$  a  $r_1$  y se elimina  $Y$  del tope de la pila y al leer  $x_2$  se pasa de  $r_1$  a  $r_2$  y se elimina  $Z$  de hasta arriba de la pila, entonces al leer  $ax_1x_2$  se pasa de  $q$  a  $r_2$  y se elimina a  $X$  del tope de la pila, pues para toda cadena del alfabeto de pila  $\alpha$ :

$$(q, ax_1x_2, X\alpha) \vdash (p, x_1x_2, YZ\alpha) \stackrel{*}{\vdash} (r_1, x_2, Z\alpha) \stackrel{*}{\vdash} (r_2, \varepsilon, \alpha).$$

Por tanto, se ha de tomar en cuenta la producción  $[qXr_2] \rightarrow a[pYr_1][r_1Zr_2]$ . Así sucesivamente, el lenguaje aceptado por la GLC debería ser la unión de los lenguajes derivados por todas las variables sintácticas de la forma  $[q_0Z_0q]$  donde  $q$  es cualquier estado; esto es, el conjunto cadenas que leídas a partir de  $q_0$  llegan a un estado  $q$  eliminando a  $Z_0$  de hasta arriba de la pila, y por tanto, vaciando la pila, en otras palabras, las cadenas aceptadas por  $A$ .

Lo mismo se tiene en cada paso si consideramos transiciones instantáneas, esto es para  $a = \varepsilon$ .

**Ejemplo 61.** Sea  $A$  el autómata de un solo estado del ejemplo 36. Con las ideas anteriores se deduce la siguiente tabla de transformación:

lazos	producciones
$a, a/\varepsilon$	 $[qaq] \rightarrow a$
$a, Z_0/\varepsilon$	 $[qZ_0q] \rightarrow a$
$a, Z_0/aZ_0$	 $[qZ_0q] \rightarrow a[qaq][qZ_0q]$

Por tanto, se tiene la GLC con símbolo inicial  $[qZ_0q]$ :

$$\begin{array}{ll} [qZ_0q] & \rightarrow a[qaq][qZ_0q] \\ & | \\ & a \\ [qaq] & \rightarrow a \end{array}$$

Esta GLC acepta por supuesto el conjunto de cadenas de  $\{a\}^*$  con longitud impar, por ejemplo, la cadena  $aaa$  puede ser derivada a través del siguiente camino más-a-la-izquierda:

$$[qZ_0q] \Rightarrow a[qaq][qZ_0q] \Rightarrow aa[qZ_0q] \Rightarrow aaa.$$

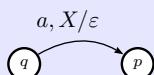
La construcción general es la siguiente:

**Algoritmo AP→GLC.** A partir de un AP  $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$  que acepta a  $L$  por pila vacía se construye una GLC  $G = (T, V, S, P)$  que acepta a  $L$  de la siguiente manera. El conjunto de variables sintácticas  $V$  se conforma por  $S$  y todos los símbolos de la forma  $[qXp]$  donde  $p$  y  $q$  son estados de  $Q$  y  $X$  es un símbolo del alfabeto de pila  $\Gamma$ . El conjunto de producciones se definen como sigue:<sup>1</sup>

- para cada estado  $q$  se incluye la producción:

$$S \rightarrow [q_0 Z_0 q];$$

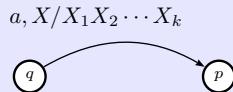
- si



es un lazo de  $A$ , donde  $a \in \Sigma \cup \{\epsilon\}$ , se agrega la producción:

$$[qXp] \rightarrow a;$$

- si



es un lazo de  $A$  donde  $a \in \Sigma \cup \{\epsilon\}$  y algún  $k > 0$ , entonces para cada lista de estados  $r_1, r_2, \dots, r_k$  se añade la producción:

$$[qXr_k] \rightarrow a[pX_1r_1][r_1X_2r_2]\cdots[r_{k-1}X_kr_k].$$

Como es de esperarse, para todo símbolo del alfabeto de la pila  $X$  y para toda cadena del alfabeto de entrada  $w$  y estados  $p$  y  $q$ :

$$[qXp] \xrightarrow{*} w \text{ si y solo si } (q, w, X) \vdash^{*} (p, \epsilon, \epsilon). \quad (7.2)$$

En particular, se tiene el siguiente teorema:

**Teorema 13.** Si  $L$  es un lenguaje aceptado por un AP por pila vacía entonces es aceptado por una GLC.

---

<sup>1</sup> Muchas de las variables sintácticas se pueden eliminar después por no ser viables, como se muestra en el ejemplo 62.

*Prueba.* La propiedad (7.2) implica que la construcción propuesta es exitosa [ejercicio 122]. A continuación se demuestra (7.2) de forma inductiva en dos pasos.

$P(n)$ : si  $[qXp] \xrightarrow{*} w$  en una cantidad de pasos menor o igual a  $n$ , entonces  $(q, w, X) \vdash^* (q, \varepsilon, \varepsilon)$ .

Si  $[qXp] \xrightarrow{*} w$  en un paso, significa que existe la producción  $[qXp] \rightarrow w$ . Por tanto,  $w = a y$  ( $p, \varepsilon \in \delta(q, a, X)$  con  $a \in \Sigma \cup \{\varepsilon\}$ ); luego  $(q, a, X) \vdash (p, \varepsilon, \varepsilon)$ . Con esto se prueba  $P(1)$ . Se supone que se cumple  $P(n)$  (hipótesis de inducción) y que  $[qXp] \xrightarrow{*} w$  en  $n + 1$  pasos. Por fuerza, la derivación debe ser de la forma:

$$[qXp] \Rightarrow a[r_0X_1r_1][r_1X_2r_2] \cdots [r_{k-1}X_kp] \xrightarrow{*} w,$$

para  $a \in \Sigma \cup \{\varepsilon\}$  algunos estados  $r_0, r_1, \dots, r_{k-1}$  y símbolos del alfabeto de la pila  $X_1, X_2, \dots, X_k$ . Sea  $w = ax_1x_2 \cdots x_k$  una descomposición de  $w$  tal que para cada  $i = 1, 2, \dots, k$ ,

$$[r_{i-1}X_ir_i] \xrightarrow{*} x_i,$$

donde  $r_k = p$ . Por hipótesis de inducción  $(r_{i-1}, x_i, X_i) \vdash^* (r_i, \varepsilon, \varepsilon)$ . Por tanto se tiene que para cada  $i = 1, 2, \dots, k$ :

$$(r_{i-1}, x_ix_{i+1} \cdots x_k, X_iX_{i+1} \cdots X_k) \vdash^* (r_i, x_{i+1} \cdots x_k, X_{i+1} \cdots X_k) \quad (7.3)$$

Por otro lado, ya que  $(r_0, X_1X_2 \cdots X_k)$  está en  $\delta(q, a, X)$ , por (7.3):

$$\begin{aligned} (q, ax_1x_2 \cdots x_k, X) &\vdash (r_0, x_1x_2 \cdots x_k, X_1X_2 \cdots X_k) \\ &\vdash^* (r_1, x_2x_3 \cdots x_k, X_2X_3 \cdots X_k) \\ &\vdash^* (r_2, x_3x_4 \cdots x_k, X_3X_4 \cdots X_k) \\ &\vdots \\ &\vdash^* (r_{k-1}, x_k, X_k) \\ &\vdash^* (r_k, \varepsilon, \varepsilon) = (p, \varepsilon, \varepsilon) \end{aligned}$$

Con esto se prueba  $P(n + 1)$ . Para completar la demostración se ha de probar la siguiente afirmación sobre números naturales.

$Q(n)$ : si  $(q, w, X\beta) \vdash^* (q, \varepsilon, \beta)$  para alguna cadena  $\beta$  del alfabeto de pila, en una cantidad de movimientos menor o igual a  $n$ , entonces  $[qXp] \xrightarrow{*} w$ .

Si  $(q, w, X\beta) \stackrel{*}{\vdash} (q, \varepsilon, \beta)$  en un movimiento para alguna cadena  $\beta$ , entonces  $w = a$ , donde  $a \in \Sigma \cup \{\varepsilon\}$  y además  $(q, \varepsilon)$  debe pertenecer a  $\delta(p, a, X)$  lo que implica que  $[qXp] \rightarrow a$  es una producción de  $G$ , en particular  $[qXp] \xrightarrow{*} a$ . Con esto se verifica  $Q(1)$ . Se supone cierta  $Q(n)$  (hipótesis de inducción) y además que  $(q, w, X\beta) \stackrel{*}{\vdash} (q, \varepsilon, \beta)$  en  $n + 1$  movimientos para alguna cadena  $\beta$ . Entonces existe un estado  $r_0$  y símbolos del alfabeto de pila  $X_1, X_2, \dots, X_k$  tales que:

$$(q, w, X\beta) \vdash (r_0, x, X_1X_2 \cdots X_k\beta) \stackrel{*}{\vdash} (p, \varepsilon, \beta), \quad (7.4)$$

donde  $w = ax$  con  $a \in \Sigma \cup \{\varepsilon\}$ . Ya que  $(r_0, X_1X_2 \cdots X_k)$  pertenece a  $\delta(q, a, X)$ , para cualquier lista de estados  $r_1, r_2, \dots, r_k$ ,

$$[qXr_k] \rightarrow a[r_0X_1r_1][r_1X_2r_2] \cdots [r_{k-1}X_kr_k]. \quad (7.5)$$

En particular, se cumple (7.5) para  $r_k = p$ . Por (7.4) existe una descomposición de  $x$ ,  $x = x_1x_2 \cdots x_k$  y una lista de estados  $r_1, r_2, \dots, r_{k-1}$  tales que:

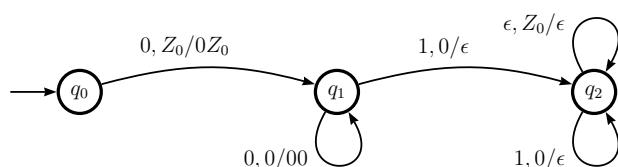
$$\begin{aligned} (r_0, x_1, X_1X_2X_3 \cdots X_k\beta) &\xrightarrow{*} (r_1, \varepsilon, X_2X_3X_4 \cdots X_k\beta), \\ (r_1, x_2, X_2X_3 \cdots X_k\beta) &\xrightarrow{*} (r_2, \varepsilon, X_3X_4 \cdots X_k\beta), \\ &\vdots \\ (r_k, x_k, X_k\beta) &\xrightarrow{*} (p, \varepsilon, \beta). \end{aligned} \quad (7.6)$$

Por hipótesis de inducción se tiene que:

$$\begin{aligned} [r_0X_1r_1] &\xrightarrow{*} x_1, \\ [r_1X_2r_2] &\xrightarrow{*} x_2, \\ &\vdots \\ [r_{k-1}X_kp] &\xrightarrow{*} x_k. \end{aligned}$$

Aplicando (7.5) para  $r_k = p$  se concluye que  $[qXp] \xrightarrow{*} ax_1x_2 \cdots x_k = w$ . Con esto se prueba  $Q(n+1)$ .  $\square$

**Ejemplo 62.** Sea  $A$  el autómata a pila:



Según el instructivo anterior las variables sintácticas de la GLC buscada deben ser:

$$S, [q_0Xq_0], [q_0Xq_1], [q_0Xq_2], [q_1Xq_0], [q_1Xq_1], [q_1Xq_2], [q_2Xq_0], [q_2Xq_1], [q_2Xq_2],$$

donde  $X = \{0, 1, Z_0\}$ , es decir,  $V$  en principio consta de  $3^2 \times 3 + 1$  variables. Sin embargo, muchas de ellas son símbolos *no-generadores*, es decir, símbolos que no derivan alguna cadena de símbolos terminales. En algunos casos se puede verificar a ojo tomando en cuenta (7.2). Se pueden eliminar de inmediatos las variables  $[q_0Xq_0]$ ,  $[q_1Xq_0]$ ,  $[q_2Xq_0]$  y  $[q_2Xq_1]$ , para  $X \in \{0, 1, Z_0\}$ , es decir, salen del juego antes de empezarlo 12 variables sintácticas. La razón de la eliminación es la misma para cada uno de los cuatro tipos de variable sintáctica. Por ejemplo, las variables del tipo  $[q_0Xq_0]$  se eliminan porque simplemente es imposible llegar de  $q_0$  a  $q_0$ , luego es absurdo pretender además que exista una cadena  $w \in \Sigma^*$  tal que  $(q_0, w, X) \xrightarrow{*} (q_0, \varepsilon, \varepsilon)$ . Más adelante, en la construcción de producciones, se eliminan también todas aquellas producciones que involucren estas variables.

Las primeras producciones que se generan son las del tipo  $S \rightarrow [q_0Z_0q]$  donde  $q \in \{q_0, q_1, q_2\}$ . Como la variable  $[q_0Z_0q_0]$  se ha eliminado, solo quedan dos producciones de este tipo:

$$\begin{array}{l} S \rightarrow [q_0Z_0q_1] \\ | [q_0Z_0q_2] \end{array}$$

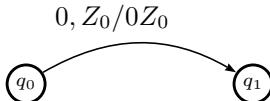
En el cuadro 7.1 se muestran las producciones que se derivan a partir de la información proporcionada por la función de transición dando como resultado la GLC siguiente:

$$\begin{array}{llll} S & \rightarrow [q_0Z_0q_1] & [q_10q_2] & \rightarrow 0[q_10q_1][q_10q_2] \\ & | [q_0Z_0q_2] & & | 0[q_10q_2][q_20q_2] \\ [q_0Z_0q_1] & \rightarrow 0[q_10q_1][q_1Z_0q_1] & & | 1 \\ [q_0Z_0q_2] & \rightarrow 0[q_10q_1][q_1Z_0q_2] & [q_20q_2] & \rightarrow 1 \\ & | 0[q_10q_2][q_2Z_0q_2] & [q_2Z_0q_2] & \rightarrow \varepsilon \\ [q_10q_1] & \rightarrow 0[q_10q_1][q_10q_1] & & \end{array}$$

Por último, las variables  $[q_10q_1]$  y  $[q_0Z_0q_1]$  son símbolos no-generadores. Por tanto, al borrar a todas las producciones que las involucran, la GLC que resulta finalmente es:

$$\begin{array}{llll} S & \rightarrow [q_0Z_0q_2] & [q_20q_2] & \rightarrow 1 \\ [q_0Z_0q_2] & \rightarrow 0[q_10q_2][q_2Z_0q_2] & [q_2Z_0q_2] & \rightarrow \varepsilon \\ [q_10q_2] & \rightarrow 0[q_10q_2][q_20q_2] & | 1 & \end{array}$$

lazos



producciones

lista	producción	¿sobrevive?
$q_0, q_0$	$[q_0 Z_0 q_0] \rightarrow 0[q_1 0 q_0][q_0 Z_0 q_0]$	no
$q_0, q_1$	$[q_0 Z_0 q_1] \rightarrow 0[q_1 0 q_0][q_0 Z_0 q_1]$	no
$q_0, q_2$	$[q_0 Z_0 q_2] \rightarrow 0[q_1 0 q_0][q_0 Z_0 q_2]$	no
$q_1, q_1$	$[q_0 Z_0 q_1] \rightarrow 0[q_1 0 q_1][q_1 Z_0 q_1]$	sí
$q_1, q_0$	$[q_0 Z_0 q_0] \rightarrow 0[q_1 0 q_1][q_1 Z_0 q_0]$	no
$q_1, q_2$	$[q_0 Z_0 q_2] \rightarrow 0[q_1 0 q_1][q_1 Z_0 q_2]$	sí
$q_2, q_2$	$[q_0 Z_0 q_2] \rightarrow 0[q_1 0 q_2][q_2 Z_0 q_2]$	sí
$q_2, q_1$	$[q_0 Z_0 q_1] \rightarrow 0[q_1 0 q_2][q_2 Z_0 q_1]$	no
$q_2, q_0$	$[q_0 Z_0 q_0] \rightarrow 0[q_1 0 q_2][q_2 Z_0 q_0]$	no

0, 0/00



lista	producción	¿sobrevive?
$q_0, q_0$	$[q_1 0 q_0] \rightarrow 0[q_1 0 q_0][q_0 0 q_0]$	no
$q_0, q_1$	$[q_1 0 q_1] \rightarrow 0[q_1 0 q_0][q_0 0 q_1]$	no
$q_0, q_2$	$[q_1 0 q_2] \rightarrow 0[q_1 0 q_0][q_0 0 q_2]$	no
$q_1, q_1$	$[q_1 0 q_1] \rightarrow 0[q_1 0 q_1][q_1 0 q_1]$	sí
$q_1, q_0$	$[q_1 0 q_0] \rightarrow 0[q_1 0 q_1][q_1 0 q_0]$	no
$q_1, q_2$	$[q_1 0 q_2] \rightarrow 0[q_1 0 q_1][q_1 0 q_2]$	sí
$q_2, q_2$	$[q_1 0 q_2] \rightarrow 0[q_1 0 q_2][q_2 0 q_2]$	sí
$q_2, q_1$	$[q_1 0 q_1] \rightarrow 0[q_1 0 q_2][q_2 0 q_1]$	no
$q_2, q_0$	$[q_1 0 q_0] \rightarrow 0[q_1 0 q_2][q_2 0 q_0]$	no

1, 0/ $\varepsilon$  $[q_1 0 q_2] \rightarrow 1$  $\varepsilon, Z_0/\varepsilon$  $[q_2 Z_0 q_2] \rightarrow \varepsilon$ 1, 0/ $\varepsilon$  $[q_2 0 q_2] \rightarrow 1$ 

**Cuadro 7.1** Cálculo de producciones a partir de la función de transición.

En lo que resta de la sección se tratará de convencer al lector de que la construcción vista para transformar un AP en una GLC arroja una GLC no-ambigua cuando el autómata a pila es un APD. La idea fundamental es que si una cadena  $w$  es aceptada por un APD por pila vacía entonces la secuencia de movimientos  $(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$  es única y por tanto existe una única secuencia de derivaciones más-a-la-izquierda  $[q_0 Z_0 q] \xrightarrow{*} w$ .

Sea  $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$  un APD y sea  $w$  una cadena aceptada por la GLC  $G = (T, V, S, P)$  dada por el algoritmo visto en esta sección. El símbolo inicial  $S$  en principio está relacionado con todas las variables sintácticas de la forma  $[q_0 Z_0 p]$  donde  $p \in Q$ , sin embargo, solo puede existir un estado  $p$  tal que  $[q_0 Z_0 p] \xrightarrow{*} w$ . De hecho, si se supone que hay dos estados  $p_1$  y  $p_2$  con esta propiedad, entonces por (7.2) se tendría que  $(q_0, w, Z_0) \vdash^* (p_1, \varepsilon, \varepsilon)$  y  $(q_0, w, Z_0) \vdash^* (p_2, \varepsilon, \varepsilon)$ . Como a partir de  $(q_0, w, Z_0)$  hay un solo camino a seguir que lee la cadena  $w$ , por fuerza  $p_1 = p_2$ . En conclusión, existe una sola derivación más-a-la-izquierda de la forma  $S \Rightarrow [q_0 Z_0 p]$ . La afirmación  $Q'(n)$  de abajo implica en particular que si  $w$  es una cadena aceptada por pila vacía entonces existe una única derivación más-a-la-izquierda  $[q_0 Z_0 p] \xrightarrow{*} w$ . Por lo anterior, existe entonces una única derivación más-a-la-izquierda  $S \xrightarrow{*} w$ . El teorema 11 implica que la gramática  $G$  es no-ambigua. Resta razonar por qué  $Q'$  es cierta para todo  $n > 0$ .

$Q'(n) : \text{si } (q, w, X\beta) \vdash^* (q, \varepsilon, \beta) \text{ para alguna cadena } \beta \text{ del alfabeto de pila,}$   
 $\quad \text{en una única secuencia de movimientos en cantidad menor o igual a } n,$   
 $\quad \text{entonces existe una única derivación más-a-la-izquierda } [qXp] \xrightarrow{*} w.$

Primero se ha de notar que  $Q'$  es una versión más específica que  $Q$ , por lo que se dan a continuación argumentos puntuales para probar  $Q'$  y por tanto el teorema 14. La afirmación  $Q'(1)$  es cierta, pues un solo movimiento  $(q, w, X\beta) \vdash (p, \varepsilon, \beta)$  arroja una derivación más-a-la-izquierda de la forma  $[qXp] \Rightarrow a$  o bien de la forma  $[qXp] \Rightarrow \varepsilon$ , pero no ambas, pues el autómata es determinista. Se supone cierta  $Q'(n)$  (hipótesis de inducción) y que existe una única secuencia de  $n + 1$  movimientos de la forma  $(q, w, X\beta) \vdash^* (p, \varepsilon, \beta)$  para alguna cadena  $\beta$ , en particular, el  $r_0$  y los símbolos  $X_1, X_2, \dots, X_k$  de (7.4) son únicos. Se infiere entonces la existencia de la producción (7.5), en principio para cualquier lista de estados  $r_1, r_2, \dots, r_k$  y en particular para  $r_k = p$ . De nuevo, como la secuencia (7.4) es única, la descomposición  $x = x_1 x_2 \cdots x_k$  y la lista de estados  $r_1, r_2, \dots, r_{k-1}$  que satisface (7.6) son únicas de igual manera. Por tanto, la secuencia de derivaciones más-a-la-izquierda:

$$\begin{aligned}
 [qXp] &\Rightarrow a[r_0X_1r_1][r_1X_2r_2]\cdots[r_{k-1}X_kp] \\
 &\stackrel{*}{\Rightarrow} ax_1[r_1X_2r_2]\cdots[r_{k-1}X_kp] \\
 &\stackrel{*}{\Rightarrow} ax_1x_2\cdots[r_{k-1}X_kp] \\
 &\vdots \\
 &\stackrel{*}{\Rightarrow} ax_1x_2\cdots x_k = w
 \end{aligned}$$

constituye la única derivación más-a-la-izquierda  $[qXp] \stackrel{*}{\Rightarrow} w$ . Con esto se prueba  $Q'(n+1)$ .

Por otro lado, si  $L$  es un lenguaje aceptado por un APD  $A$  por estado de aceptación entonces existe otro APD que acepta al lenguaje  $L_\diamond = \{x\diamond : x \in L\}$  por pila vacía [ejercicio 101] ( $\diamond$  es un símbolo que no pertenece al conjunto de símbolos de entrada de  $A$ ). A partir de la GLC no ambigüa que existe para  $L_\diamond$  es trivial construir otra para  $L$ . En resumen:

**Teorema 14.** Si  $L$  es un lenguaje aceptado por un APD —por pila vacía o por estado de aceptación— entonces es aceptado por una GLC no-ambigua.

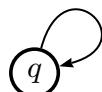
### Listado de ejercicios de la sección 7.2

*Ejercicio 122.* Pruebe que la propiedad (7.2) implica el teorema 13.

*Ejercicio 123.* Dado un AP con  $n$  estados que acepta a  $L$  por estado de aceptación construya otro AP pero de un solo estado que acepte a  $L$  por pila vacía.

*Ejercicio 124.* Usando el algoritmo visto en esta sección convierte el siguiente AP en una GLC:

$$\begin{array}{l}
 a, Z_0/aZ_0 \\
 b, a/\varepsilon \\
 \varepsilon, Z_0/\varepsilon
 \end{array}$$



### 7.3 Lema de bombeo para los lenguajes libres de contexto

Se dice que  $L$  es un *lenguaje libre de contexto* si existe una GLC que lo acepta como lenguaje. Por los teoremas 12, 13, 7 y 8 se tiene la siguiente caracterización de los lenguajes libres de contexto:

**Teorema 15.** *Sea  $L$  un lenguaje. Las siguientes afirmaciones son equivalentes:*

- *$L$  es aceptado por un AP por estado de aceptación.*
- *$L$  es aceptado por un AP por pila vacía.*
- *$L$  es un lenguaje libre de contexto.*

Lo que sigue ahora es establecer y demostrar un lema de bombeo para lenguajes libres de contexto, de la misma forma que en el caso de los lenguajes regulares, es una herramienta para probar que un lenguaje *no* es libre de contexto. Con este fin, es necesario considerar primero la llamada *forma normal de Chomsky* (FNC) de una GLC:

**Definición FNC.** Se dice que una GLC  $G = (T, V, S, P)$  está en *forma normal de Chomsky* si todos sus símbolos son útiles —esto es, variables sintácticas o símbolos terminales  $X$  tales que existe una derivación de la forma  $S \xrightarrow{*} \alpha X \beta \xrightarrow{*} w$ , para algunas  $\alpha$  y  $\beta$  en  $(T \cup V)^*$  y  $w \in T^*$ — y además todas sus producciones son de algunas de las siguientes dos formas:

$$\begin{array}{ll} A & \rightarrow BC \\ D & \rightarrow a \end{array}$$

donde  $A, B, C$  y  $D$  son variables sintácticas y  $a$  es un símbolo terminal.

La clave del lema de bombeo se basa en dos hechos:

- Si una GLC acepta un lenguaje  $L$  que contiene al menos una cadena distinta de  $\varepsilon$  entonces se puede construir otra GLC  $G$  pero en FNC tal que  $L(G) = L \setminus \{\varepsilon\}$  [teorema 18].
- Si  $\Delta$  es un árbol de derivación de con raíz  $S$  y rendimiento  $w \in T^*$  de una GLC en FNC tal que la longitud del camino más largo de  $\Delta$  es  $k$  entonces [ejercicio 125]:

$$|w| \leq 2^{k-1}.$$

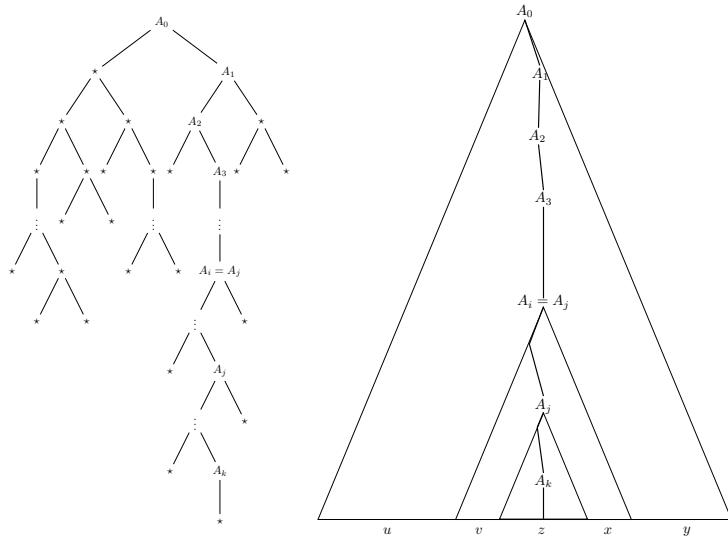
Sea  $L$  un lenguaje libre de contexto. Al igual que en el caso de los lenguajes regulares, lo que se pretende es dado un lenguaje libre de contexto  $L$ , asignarle un  $n > 0$  de tal forma que para toda  $w \in L$  con  $|w| > n$  exista una descomposición de  $w$  de alguna forma donde algunas de las subcadenas de  $w$  se puedan «bombar». Por esta razón, sin perder generalidad se puede suponer que  $L \neq \emptyset$  y no contiene a la cadena vacía, pues esta última no tiene longitud mayor a  $n > 0$ . Por tanto, por el primer punto de arriba se puede suponer que existe una gramática  $G$  en FNC que acepta a  $L$  como lenguaje. Sea  $m$  el número de variables sintácticas de  $G$  y se propone:

$$n = 2^m$$

Sea  $w$  es una cadena de símbolos terminales tal que  $|w| > n$ . Por el segundo punto de arriba, todo árbol de derivación cuyo camino más largo tenga longitud  $m$  o menos tienen que derivar cadenas de longitud a lo más  $2^{m-1} = \frac{n}{2}$ . Por tanto, los árboles de derivación con rendimiento  $w$  tienen que tener un camino de al menos longitud  $m + 1$ . Sea  $\Delta$  un árbol de derivación con rendimiento  $w$  y sea  $k \geq m$  tal que  $k + 1$  es la longitud del camino más largo en  $\Delta$  (se fija un camino más largo). Entonces, existen  $A_0, A_1, \dots, A_k$  etiquetas de  $k + 1$  nodos representando las variables sintácticas correspondientes al camino más largo. Como solo hay  $m$  diferentes variables sintácticas existen al menos dos que se repiten dentro de las últimas  $m + 1$  variables, es decir, existen índices  $i$  y  $j$  tales que  $k - m \leq i < j < k$  y  $A_i = A_j$ . Se considera la descomposición de  $w = uvzxy$ , donde (ver figura 7.1):

- la cadena  $z$  es la subcadena de  $w$  que se deriva del subárbol de  $\Delta$  con raíz  $A_j$ ;
- las cadenas  $v$  y  $x$  son las subcadenas del rendimiento del subárbol de  $\Delta$  con raíz  $A_i$  que se encuentran a la izquierda y derecha, respectivamente, de  $z$ ;
- las cadenas  $u$  y  $y$  son las subcadenas de  $w$  que se encuentran a la izquierda de  $v$  y la derecha de  $x$ , respectivamente.

Primero se ha de notar que como no hay producciones unitarias se cumple que  $vx \neq \varepsilon$ . Por otro lado, ya que  $k - i \leq m$ , entonces la longitud del camino más largo en el subárbol de  $\Delta$  con raíz  $A_i$  no puede pasar de  $m + 1$ , por tanto, la longitud de cualquier rendimiento del subárbol de  $\Delta$  con raíz  $A_i$  no puede pasar de  $2^m = n$ . En particular,  $|vzx| \leq n$ . Finalmente, al considerar la igualdad  $A_i = A_j = A$  es posible construir nuevos árboles de derivación a partir de  $\Delta$ . Se puede, por ejemplo, reemplazar el subárbol de  $\Delta$  con raíz  $A_i$  con rendimiento  $uzx$ , por el subárbol de  $\Delta$  con raíz  $A_j$  que deriva a  $z$ . Luego este nuevo árbol tiene raíz  $S$  y rendimiento  $uzy$ . Por tanto,  $uv^0zx^0y \in L$ . Más aún, si se reemplaza el subárbol  $\Delta$  con raíz  $A_j$  por el subárbol de  $\Delta$  con raíz  $A_i$ , entonces el nuevo árbol tiene como rendimiento a la cadena  $uv^2zx^2y$ . Así sucesivamente se pueden



**Fig. 7.1** Árbol de derivación con rendimiento  $w = uvzxy$ .

construir árboles de derivación con raíz  $S$  y rendimiento  $uv^r zx^r y$  para cualquier  $r \geq 3$ . En resumen se tiene:

**Teorema 16 (lema de bombeo para lenguajes libres de contexto).** *Sea  $L$  un lenguaje libre de contexto. Entonces existe un  $n > 0$ , tal que para toda cadena  $w \in L$  con  $|w| > n$  existe una descomposición de  $w$  de la forma  $w = uvzxy$  que satisface:*

- B<sub>1</sub>.*  $vx \neq \varepsilon$ ,
- B<sub>2</sub>.*  $|vzx| \leq n$ ,
- B<sub>3</sub>.* para todo  $r \geq 0$ , la cadena  $uv^r zx^r y$  está en  $L$ .

**Ejemplo 63.** Sea  $L_{012} = \{0^n 1^n 2^n : n > 0\}$ . Razonando por el absurdo, se supone que  $L_{012}$  es un lenguaje libre de contexto y por tanto existe un  $n > 0$  tal que toda cadena  $w$  con  $|w| > n$  admite una descomposición de la forma  $w = uvzxy$  que satisface  $B_1$ ,  $B_2$  y  $B_3$ . Sea  $w = 0^n 1^n 2^n$ . Si  $|vzx| \leq n$  y  $vx \neq \varepsilon$ , entonces  $vzx$  no puede contener a la vez el símbolo 0 y el símbolo 2, pues entre el último 0 y el primer 2 hay justamente  $n$  posiciones. Existen entonces dos situaciones posibles. Si  $vzx$  no contiene al símbolo 2, entonces  $vx$  consiste de símbolos 0 y 1 además

contiene al menos uno de estos símbolos, pues  $vx \neq \varepsilon$ . Luego  $uv^0zx^0y = uzy$  no está en  $L_{012}$ , pues tiene  $n$  símbolos 2 y menos de  $n$  símbolos 0 ó 1. Análogamente se razona si  $uwx$  no contiene el símbolo 0.

### **Lista de ejercicios de la sección 7.3**

*Ejercicio 125.* Sea  $G = (T, V, S, P)$  una GLC en FNC y sea  $\Delta$  un árbol de derivación de  $G$  con raíz  $S$  y rendimiento  $w \in T^*$ . Si la longitud del camino —esto es, el número de nodos conectados menos 1— más largo de  $\Delta$  es  $k$  entonces  $|w| \leq 2^{k-1}$ . Sugerencia: razonar por inducción sobre  $k$ .

*Ejercicio 126.* Sean  $A, B$  y  $C$  dados por el algoritmo (7.7). Pruebe que efectivamente  $L(C) = L(A) \cap L(B)$ . Sugerencia: pruebe inductivamente que  $(q_0, w, Z_0) \vdash_A^* (q, \varepsilon, \beta)$  y  $(p_0, w) \vdash_B^* (p, \varepsilon)$  si y solo si  $((q_0, p_0), w, Z_0) \vdash_C^* ((q, p), \varepsilon, \beta)$ .

*Ejercicio 127.* Pruebe que los siguientes lenguajes no son libres de contexto.

1.  $\{a^i b^j c^i d^j : i \geq 1, j \geq i\}$ ,
2.  $\{ww : w \text{ es una cadena binaria}\}$ ,
3.  $\{0^n 1^n 2^i : i \leq n\}$ .

*Ejercicio 128.* Una *gramática dependiente de contexto* es una cuádrupla  $(T, V, S, P)$  donde  $T$  es el conjunto de símbolos terminales,  $V$  es el conjunto de variables sintácticas,  $S \in V$  es el símbolo inicial, pero el conjunto de producciones  $P$  de la forma:

$$\alpha \rightarrow \beta, \quad |\alpha| \leq |\beta|$$

donde  $\alpha$  y  $\beta$  son cadenas de  $(T \cup V)^*$  y  $\alpha$  contiene al menos una variable sintáctica. Al igual que antes, el lenguaje aceptado por una gramática dependiente de contexto es el conjunto de cadenas de símbolos terminales  $w$  tales que  $S \xrightarrow{*} w$ , en este caso se le llama *lenguaje dependiente de contexto*. Un ejemplo de gramática dependiente de contexto es la gramática  $G$  dada por:

$$\begin{array}{ll} S & \rightarrow 0SBC \\ & | \\ & 0BC \\ CB & \rightarrow BC \\ 1B & \rightarrow 11 \\ 0B & \rightarrow 01 \\ 1C & \rightarrow 12 \\ 2C & \rightarrow 22 \end{array}$$

Demuestre que  $L(G) = L_{012}$ , es decir el conjunto  $L_{012}$  del ejemplo 63 es dependiente de contexto pero no es libre de contexto. Note que por definición, una gramática dependiente de contexto no admite producciones de la forma  $A \rightarrow \varepsilon$ . Lo que sí se cumple es que *si  $L$  es libre de contexto y no contiene a la cadena vacía entonces es dependiente de contexto*.

*Ejercicio 129. Problemas de decisión de lenguajes libres de contexto.* Sea  $G$  una GLC en FNC con  $m$  variables sintácticas. Pruebe que:

1.  $L(G) \neq \emptyset$  si y solo si existe una cadena  $w$  con  $|w| \leq 2^m$  que es aceptada.
2.  $L(G)$  es infinito si y solo si existe  $w \in L(G)$  con  $2^m < |w| \leq 2^{m+1}$ .

En base a esto, diseñe un algoritmo que reciba como entrada una gramática libre de contexto y regrese un «sí» o un «no» dependiendo si el lenguaje que acepta es vacío o no. Haga lo mismo pero que la respuesta sea en función de si el lenguaje es infinito o no.

## 7.4 Propiedades de los lenguajes libres de contexto

En el capítulo 2 se presenta una construcción de un AFD para la intersección de dos lenguajes regulares, en particular la intersección de dos lenguajes regulares es también un lenguaje regular. La misma pregunta es natural plantearse para dos lenguajes libres de contexto  $L$  y  $M$ . El teorema 15 pudiera sugerir la existencia de un algoritmo análogo al (2.2) para la construcción de un AFD correspondiente a la intersección de dos lenguajes regulares. El primer problema surge al tratar de construir un nuevo alfabeto de pila a partir de otros dos dados. De hecho, esa construcción es imposible, pues en general *no es cierto que la intersección de dos lenguajes libres de contexto  $L \cap M$  sea libre de contexto*. Como muestra se tiene al lenguaje del ejemplo 63 el cual se ha demostrado que no es libre de contexto aunque de hecho es la intersección de dos que sí lo son [ejercicio 103]:

$$\{0^n 1^n 2^n : n > 0\} = \{0^n 1^n 2^m : n, m > 0\} \cap \{0^m 1^n 2^n : n, m > 0\}.$$

En particular, por las leyes de De Morgan *no es cierto que  $L \setminus M$  sea libre de contexto* [ejercicio 130]. Sin embargo, se tiene el siguiente teorema:

**Teorema 17.** *Si  $L$  es un lenguaje libre de contexto y  $M$  es un lenguaje regular, entonces  $L \cap M$  es un lenguaje libre de contexto.*

La demostración es básicamente la justificación del siguiente algoritmo que, a partir una GLC que acepta a un lenguaje  $L$  como lenguaje y un AFD que acepta a un lenguaje  $M$ , regresa una GLC que acepta a  $L \cap M$  como lenguaje.

**Algoritmo GLC de  $L(G) \cap L(A)$  para G GLC y A AFD.** Sea  $G = (\Sigma, V, S, P)$  una GLC y  $A = (Q, \Sigma, q_0, \delta, F)$  un AFD. Se construye una GLC

$$\hat{G} = (\Sigma, \hat{V}, \hat{S}, \hat{P})$$

que acepta a  $L(G) \cap L(A)$  como lenguaje como sigue. Para cada conjunto de estados  $p$  y  $q$  y símbolo  $X \in \Sigma \cup V$  se le asocia una variable sintáctica  $[qXp]$ , de tal suerte que  $\hat{V}$  consiste en  $\hat{S}$  y todas las posibles variables sintácticas asociadas. Además, el conjunto de producciones  $\hat{P}$  se constituye de la siguiente manera:

- para cada  $q \in F$  se añade la producción  $\hat{S} \rightarrow [q_0Sq]$  a  $\hat{P}$ ;
- si  $X \rightarrow X_1X_2 \cdots X_k$  es una producción de  $G$  entonces para cada lista de estados  $p, r_1, r_2, \dots, r_{k-1}, q$  se añade a  $\hat{P}$  la producción:

$$[pXq] \rightarrow [pX_1r_1][r_1X_2r_2] \cdots [r_{k-1}X_kq];$$

- para todo símbolo de entrada  $a \in \Sigma$  y par de estados  $p$  y  $q$  tales que  $\delta(p, a) = q$  se añade a  $\hat{P}$  la producción:

$$[paq] \rightarrow a.$$

(7.7)

*Prueba.* Sea  $G = (\Sigma, V, S, P)$  una GLC,  $A = (Q, \Sigma, q_0, \delta, F)$  un AFD y sea  $\hat{G}$  la GLC que se construye mediante el algoritmo (7.7). Se prueba a continuación que  $L(\hat{G}) = L(G) \cap L(A)$ . Sea  $w = a_1a_2 \cdots a_n \in L(G) \cap L(A)$ . Ya que  $w \in L(G)$  se tiene que  $S \xrightarrow{*} a_1a_2 \cdots a_n$ . Por la definición de  $\hat{G}$  se tiene que [ejercicio 131]:

$$\hat{S} \Rightarrow [q_0Sq_n] \xrightarrow{*} [q_0a_1q_1][q_1a_2q_2][q_2a_3q_3] \cdots [q_{n-1}a_nq_n].$$

donde  $q_n \in F$  y  $q_1, q_2, \dots, q_{n-1}$  son estados cualesquiera de  $A$ . Ya que  $w \in L(A)$  podemos tomar los estados  $q_1, q_2, \dots, q_{n-1}$  tales que:

$$\delta(q_i, a_{i+1}) = q_{i+1}, \quad i = 0, 1, \dots, n-1$$

y por tanto para  $i = 0, 1, \dots, n-1$ , la producción  $[q_ia_{i+1}q_{i+1}] \rightarrow a_{i+1}$  está en  $\hat{P}$  y en particular  $[q_ia_{i+1}q_{i+1}] \Rightarrow a_{i+1}$ . Luego se tiene las derivaciones:

$$\begin{aligned}
\hat{S} &\xrightarrow{*} [q_0a_1q_1][q_1a_2q_2][q_2a_3q_3] \cdots [q_{n-1}a_nq_n] \\
&\Rightarrow a_1[q_1a_2q_2][q_2a_3q_3] \cdots [q_{n-1}a_nq_n] \\
&\Rightarrow a_1a_2[q_2a_3q_3] \\
&\Rightarrow a_1a_2a_3 \cdots [q_{n-1}a_nq_n] \\
&\quad \vdots \\
&\Rightarrow a_1a_2a_3 \cdots a_n
\end{aligned}$$

En otras palabras  $\hat{S} \xrightarrow{*} w$ . Supongamos ahora que  $w \in L(\hat{G})$ . Entonces existe un estado de aceptación  $q \in F$  tal que:

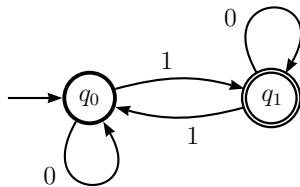
$$\hat{S} \Rightarrow [q_0Sq] \xrightarrow{*} w.$$

Por el ejercicio 132 se tiene que  $\hat{\delta}(q_0, w) = q$  y además  $S \xrightarrow{*} w$ . Por tanto  $w \in L(A)$  y  $w \in L(G)$ .  $\square$

**Ejemplo 64.** Sea  $L$  el lenguaje libre de contexto  $L_{01} = \{0^n1^n : n > 0\}$  y  $M$  el lenguaje regular de cadenas binarias con un número impar de unos. El lenguaje  $L \cap M$  es por tanto el lenguaje de cadenas binarias de la forma  $0^{2k+1}1^{2k+1}$  para  $k = 0, 1, \dots$ . El lenguaje  $L$  es aceptado por la GLC  $G$  del ejemplo 47:

$$\begin{array}{l}
S \rightarrow 01 \\
| \quad 0S1
\end{array}$$

A su vez, el lenguaje  $M$  es aceptado por el AFD  $A$  del ejemplo 14:



A continuación se aplica el algoritmo (7.7) para encontrar una GLC  $\hat{G}$  que acepte a  $L \cap M$  como lenguaje. En primer lugar  $V \cup \Sigma = \{S, 0, 1\}$  y por tanto el conjunto de variables sintácticas  $\hat{V}$  consiste en  $\hat{S}$  y las 24 variables de expuestas en la siguiente tabla:

$[q_0Sq_0]$	$[q_00q_0]$	$[q_01q_0]$
$[q_0Sq_1]$	$[q_00q_1]$	$[q_01q_1]$
$[q_1Sq_0]$	$[q_10q_0]$	$[q_11q_0]$
$[q_1Sq_1]$	$[q_10q_1]$	$[q_11q_1]$

Las producciones de  $\hat{P}$  se añaden a continuación poco a poco. En primer lugar se considera la producción:

$$\hat{S} \rightarrow [q_0 S q_1]$$

Además, la producción  $S \rightarrow 01$  genera la siguiente lista de producciones; en este caso  $X_1 = 0$  y  $X_1 = 1$ :

	lista de estados correspondiente
$[q_0 S q_0] \rightarrow [q_0 0 q_0][q_0 1 q_0]$	$q_0, q_0, q_0$
$[q_0 0 q_1][q_1 1 q_0]$	$q_0, q_1, q_0$
$[q_1 S q_0] \rightarrow [q_1 0 q_0][q_0 1 q_0]$	$q_1, q_0, q_0$
$[q_1 0 q_1][q_1 1 q_0]$	$q_1, q_1, q_0$
$[q_0 S q_1] \rightarrow [q_0 0 q_0][q_0 1 q_1]$	$q_0, q_0, q_1$
$[q_0 0 q_1][q_1 1 q_1]$	$q_0, q_1, q_1$
$[q_1 S q_1] \rightarrow [q_1 0 q_0][q_0 1 q_1]$	$q_1, q_0, q_1$
$[q_1 0 q_1][q_1 1 q_1]$	$q_1, q_1, q_1$

A su vez, la producción  $S \rightarrow 0S1$  genera las siguientes producciones; en notación del algoritmo  $X_1 = 0, X_2 = S$  y  $X_3 = 1$ :

	lista de estados correspondiente
$[q_0 S q_0] \rightarrow [q_0 0 q_0][q_0 S q_0][q_0 1 q_0]$	$q_0, q_0, q_0, q_0$
$[q_0 0 q_1][q_1 S q_0][q_0 1 q_0]$	$q_0, q_1, q_0, q_0$
$[q_0 0 q_0][q_0 S q_1][q_1 1 q_0]$	$q_0, q_0, q_1, q_0$
$[q_0 0 q_1][q_1 S q_1][q_1 1 q_0]$	$q_0, q_1, q_1, q_0$
$[q_1 S q_0] \rightarrow [q_1 0 q_0][q_0 S q_0][q_0 1 q_0]$	$q_1, q_0, q_0, q_0$
$[q_1 0 q_1][q_1 S q_0][q_0 1 q_0]$	$q_1, q_1, q_0, q_0$
$[q_1 0 q_0][q_0 S q_1][q_1 1 q_0]$	$q_1, q_0, q_1, q_0$
$[q_1 0 q_1][q_1 S q_1][q_1 1 q_0]$	$q_1, q_1, q_1, q_0$
$[q_0 S q_1] \rightarrow [q_0 0 q_0][q_0 S q_0][q_0 1 q_1]$	$q_0, q_0, q_0, q_1$
$[q_0 0 q_1][q_1 S q_0][q_0 1 q_1]$	$q_0, q_1, q_0, q_1$
$[q_0 0 q_0][q_0 S q_1][q_1 1 q_1]$	$q_0, q_0, q_1, q_1$
$[q_0 0 q_1][q_1 S q_1][q_1 1 q_1]$	$q_0, q_1, q_1, q_1$
$[q_1 S q_1] \rightarrow [q_1 0 q_0][q_0 S q_0][q_0 1 q_1]$	$q_1, q_0, q_0, q_1$
$[q_1 0 q_1][q_1 S q_0][q_0 1 q_1]$	$q_1, q_1, q_0, q_1$
$[q_1 0 q_0][q_0 S q_1][q_1 1 q_1]$	$q_1, q_0, q_1, q_1$
$[q_1 0 q_1][q_1 S q_1][q_1 1 q_1]$	$q_1, q_1, q_1, q_1$

Finalmente, el AFD arroja las siguientes producciones:

$$\begin{aligned}[q_00q_0] &\rightarrow 0 \\ [q_01q_1] &\rightarrow 1 \\ [q_11q_0] &\rightarrow 1 \\ [q_10q_1] &\rightarrow 0\end{aligned}$$

Después de quitar la basura (los símbolos inútiles) la gramática resultante es:

$$\begin{aligned}\hat{S} &\rightarrow [q_0Sq_1] \\ [q_0Sq_1] &\rightarrow [q_00q_0][q_01q_1] \\ &\quad | \\ &\quad [q_00q_0][q_0Sq_0][q_01q_1] \\ [q_0Sq_0] &\rightarrow [q_00q_0][q_0Sq_1][q_11q_0] \\ [q_00q_0] &\rightarrow 0 \\ [q_01q_1] &\rightarrow 1 \\ [q_11q_0] &\rightarrow 1 \\ [q_10q_1] &\rightarrow 0\end{aligned}$$

Así pues la cadena  $w = 000111$  se deriva de la GLC dada mediante la secuencia de derivaciones:

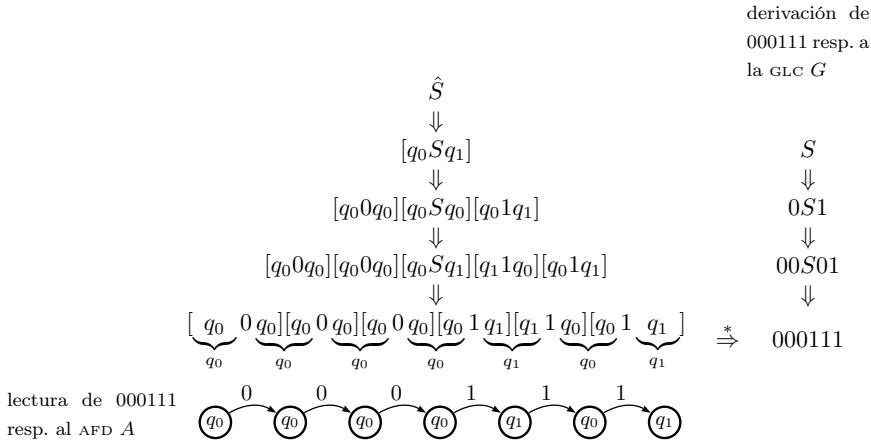
$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$$

Por otro lado, la lectura de la cadena 000111 a través del AFD pasa por la siguiente secuencia de estados (ver el dibujo de arriba):

$$(q_0, 000111) \vdash (q_0, 00111) \vdash (q_0, 0111) \vdash (q_0, 111) \vdash (q_1, 11) \vdash (q_0, 1) \vdash (q_1, \varepsilon).$$

La derivación de la cadena 000111 en la nueva GLC  $\hat{G}$  que se ha construido refleja ambos caminos, tanto para derivar la cadena a partir de  $G$  como para la lectura a través del AFD  $A$ , ver figura 7.2.

Para concluir, note que a partir de dos GLC para  $L$  y  $M$  respectivamente, es fácil construir otra para  $L \cup M$ ,  $LM$  y  $L^*$  [ejercicio 108], por tanto, si  $L$  y  $M$  son libres de contexto, también lo son  $L \cup M$ ,  $LM$  y  $L^*$ .



**Fig. 7.2** Derivación de la cadena  $w = 000111$  respecto a la GLC  $\hat{G}$  vista en relación a la derivación de  $w$  respecto a la GLC  $G$  y la lectura de  $w$  respecto al AFD  $A$ .

### Lista de ejercicios de la sección 7.4

*Ejercicio 130.* Pruebe que si  $L$  y  $M$  son libres de contexto, no necesariamente  $L \setminus M$  es libre de contexto. Sugerencia: razoné por contradicción.

*Ejercicio 131.* Sea  $G = (\Sigma, V, S, P)$  una GLC,  $A = (Q, \Sigma, q_0, \delta, F)$  un AFD y sea  $\hat{G}$  la GLC que se construye mediante el algoritmo (7.7). Sean  $a_1, a_2 \dots a_n$  símbolos en  $\Sigma$ . Pruebe que si

$$S \xrightarrow{*} a_1 a_2 \dots a_n$$

entonces para todo par de estados  $p$  y  $q$  de  $A$  y para toda lista de estados  $r_1, r_2, \dots, r_{n-1}$  se tiene que:

$$[qSp] \xrightarrow{*} [qa_1r_1][r_1a_2r_2][r_2a_3r_3] \dots [r_{n-1}a_np].$$

Sugerencia: razoné por inducción sobre el número de pasos de  $S \xrightarrow{*} a_1 a_2 \dots a_n$ .

*Ejercicio 132.* Sea  $G = (\Sigma, V, S, P)$  una GLC,  $A = (Q, \Sigma, q_0, \delta, F)$  un AFD y sea  $\hat{G}$  la GLC que se construye mediante el algoritmo (7.7). Demuestre que para todo par de estados  $p$  y  $q$  y símbolo  $X \in \Sigma \cup V$  si  $[pXq] \xrightarrow{*} w \in \Sigma^*$  entonces  $\hat{\delta}(p, w) = q$ . Si además  $X$  es una variable sintáctica, entonces  $X \xrightarrow{*} w$  respecto a la GLC  $G$ . Sugerencia: razoné por inducción sobre número de pasos de  $[pXq] \xrightarrow{*} w$ .

## 7.5 Algoritmo de Cocke-Younger-Kasami

A continuación se presenta un procedimiento conocido en la literatura como *algoritmo CYK*. Dada una GLC en FNC y una cadena  $w$ , el algoritmo CYK regresa un «sí» o un «no» dependiendo si  $w \in L(G)$ .

**Algoritmo CYK.** Sea  $G$  una gramática en FNC con símbolo inicial  $S$  y sea  $w = a_1a_2 \cdots a_n$  una cadena. El algoritmo CYK parte de una media tabla indexada como la siguiente:

$n$	$(1,n)$								
:	:	$(2,n)$							
3	$(1,3)$	:	$(3,n)$						
2	$(1,2)$	$(2,3)$	:			$\ddots$			
1	$(1,1)$	$(2,2)$	$(3,3)$	$\ddots$			$(n,n)$		
	$a_1$	$a_2$	$a_3$	$\cdots$	$\cdots$			$a_n$	

A cada espacio en la tabla con índice  $(i, j)$  se le asigna el conjunto de variables sintácticas  $V_{ii} = \{A : A \xrightarrow{*} a_i a_{i+1} \cdots a_j\}$  los cuales se establecen de forma inductiva:

- R. INICIAL para  $1 \leq i \leq n$  se define  $V_{ii} = \{A : A \rightarrow a_i\}$ ;
- R. INDUCTIVA se supone que se han calculado los conjuntos hasta cierto renglón  $m$ ; se definen los  $V_{ij}$  del renglón  $m + 1$  de la siguiente forma. Para  $k = i, i + 1, \dots, j - 1$ , si:
  - $B \in V_{ik}$ ,
  - $C \in V_{(k+1)j}$ , y
  - $A \rightarrow BC$  es una producción de  $G$ .
 entonces  $A \in V_{ij}$ .

El símbolo inicial  $S$  pertenece a  $V_{1n}$  si y solo si  $S \xrightarrow{*} w$ .

Es importante señalar que el paso inductivo del algoritmo CYK es necesario y suficiente, esto es:  $A \in V_{ij}$  si y solo si existe  $k = i, i + 1, \dots, j - 1$  y variables sintácticas  $B \in V_{ik}$  y  $C \in V_{(k+1)j}$  tales que  $A \rightarrow BC$  es una producción de la gramática  $G$ . De hecho, si  $A \xrightarrow{*} a_i a_{i+1} \cdots a_j$  es una derivación de más de un paso, al estar la gramática en FNC tiene que comenzar con  $A \Rightarrow BC$ , para algunas

variables  $B$  y  $C$  y por tanto existe un  $k$  entre  $i$  y  $j - 1$  tal que  $B \xrightarrow{*} a_i a_{i+1} \cdots a_k$  y  $C \xrightarrow{*} a_{k+1} a_{k+2} \cdots a_j$ . La otra implicación es trivial.

Como comentario final, note que la cadena  $a_i a_{i+1} \cdots a_j$  en la definición del conjunto  $V_{ij}$  tiene longitud  $j - i + 1$ . Por tanto, primer renglón le corresponde los conjuntos de la forma  $V_{ii}$ , las cadenas asociadas a los conjuntos  $V_{ii}$  tienen longitud 1; las cadenas que definen a los conjuntos  $V_{ij}$  del segundo renglón tienen longitud 2; y así sucesivamente hasta llegar a la única cadena de longitud  $n$  que define el conjunto  $V_{1n}$  en el renglón  $n$ .

**Ejemplo 65.** Considere la siguiente GLC en FNC que acepta al conjunto de corchetes equilibrados  $L_{\square}$  [ejercicio 75]:

$$\begin{array}{ll} S & \rightarrow AC \\ & | \\ & AB \\ & | \\ & AD \\ & | \\ & AE \\ B & \rightarrow SC \\ D & \rightarrow CS \\ E & \rightarrow BS \\ A & \rightarrow [ \\ C & \rightarrow ] \end{array}$$

Sea  $w = [\square]$  una cadena de longitud 4. En este ejemplo  $a_1 = [$ ,  $a_2 = \square$ ,  $a_3 = ]$  y  $a_4 = ]$ . El primer renglón de la tabla indizada se llena a partir de la regla inicial:

<table border="1" style="border-collapse: collapse; width: 100%; height: 100%;"> <tr><td></td><td></td><td></td><td></td></tr> <tr><td>4</td><td>(1,4)</td><td></td><td></td></tr> <tr><td>3</td><td>(1,3)</td><td>(2,4)</td><td></td></tr> <tr><td>2</td><td>(1,2)</td><td>(2,3)</td><td>(3,4)</td></tr> <tr><td>1</td><td>(1,1)</td><td>(2,2)</td><td>(3,3)</td><td>(4,4)</td></tr> <tr><td></td><td>[</td><td>[</td><td>]</td><td>]</td></tr> </table>					4	(1,4)			3	(1,3)	(2,4)		2	(1,2)	(2,3)	(3,4)	1	(1,1)	(2,2)	(3,3)	(4,4)		[	[	]	]	→	<table border="1" style="border-collapse: collapse; width: 100%; height: 100%;"> <tr><td></td><td></td><td></td><td></td></tr> <tr><td>4</td><td>(1,4)</td><td></td><td></td></tr> <tr><td>3</td><td>(1,3)</td><td>(2,4)</td><td></td></tr> <tr><td>2</td><td>(1,2)</td><td>(2,3)</td><td>(3,4)</td></tr> <tr><td>1</td><td>A</td><td>A</td><td>C</td><td>C</td></tr> <tr><td></td><td>[</td><td>[</td><td>]</td><td>]</td></tr> </table>					4	(1,4)			3	(1,3)	(2,4)		2	(1,2)	(2,3)	(3,4)	1	A	A	C	C		[	[	]	]
4	(1,4)																																																					
3	(1,3)	(2,4)																																																				
2	(1,2)	(2,3)	(3,4)																																																			
1	(1,1)	(2,2)	(3,3)	(4,4)																																																		
	[	[	]	]																																																		
4	(1,4)																																																					
3	(1,3)	(2,4)																																																				
2	(1,2)	(2,3)	(3,4)																																																			
1	A	A	C	C																																																		
	[	[	]	]																																																		

El proceso de la regla inductiva se explica en el cuadro 7.3. Puesto que  $S$  se encuentra en la casilla (1, 4) se concluye que  $[\square] \in L(G)$ .

### Listado de ejercicios de la sección 7.5

**Ejercicio 133.** Escriba un programa que reciba como entrada un texto y diga si ese texto incluye o no a la palabra **sonora**.

$j - i + 1$	$V_{ij} k $	$V_{ik}$	$ V_{(k+1)j} $	producción	tabla																									
2	$V_{12} 1  V_{11} = \{A\}   V_{22} = \{A\}  $ ninguna $V_{23} 2  V_{22} = \{A\}   V_{33} = \{C\}   S \rightarrow AC$ $V_{34} 3  V_{33} = \{C\}   V_{44} = \{C\}  $ ninguna				<table border="1"> <tr><td>4</td><td>(1,4)</td><td></td><td></td><td></td></tr> <tr><td>3</td><td>(1,3)</td><td>(2,4)</td><td></td><td></td></tr> <tr><td>2</td><td><math>\emptyset</math></td><td><math>S</math></td><td><math>\emptyset</math></td><td></td></tr> <tr><td>1</td><td>A</td><td>A</td><td>C</td><td>C</td></tr> <tr><td></td><td>[</td><td>[</td><td>]</td><td>]</td></tr> </table>	4	(1,4)				3	(1,3)	(2,4)			2	$\emptyset$	$S$	$\emptyset$		1	A	A	C	C		[	[	]	]
4	(1,4)																													
3	(1,3)	(2,4)																												
2	$\emptyset$	$S$	$\emptyset$																											
1	A	A	C	C																										
	[	[	]	]																										
3	$V_{13} 1  V_{11} = \{A\}   V_{23} = \{S\}  $ ninguna $V_{13} 2  V_{12} = \emptyset   V_{33} = \{C\}  $ ninguna $V_{24} 2  V_{22} = \{A\}   V_{34} = \emptyset  $ ninguna $V_{24} 3  V_{23} = \{S\}   V_{44} = \{C\}   B \rightarrow SC$				<table border="1"> <tr><td>4</td><td>(1,4)</td><td></td><td></td><td></td></tr> <tr><td>3</td><td><math>\emptyset</math></td><td>B</td><td></td><td></td></tr> <tr><td>2</td><td><math>\emptyset</math></td><td><math>S</math></td><td><math>\emptyset</math></td><td></td></tr> <tr><td>1</td><td>A</td><td>A</td><td>C</td><td>C</td></tr> <tr><td></td><td>[</td><td>[</td><td>]</td><td>]</td></tr> </table>	4	(1,4)				3	$\emptyset$	B			2	$\emptyset$	$S$	$\emptyset$		1	A	A	C	C		[	[	]	]
4	(1,4)																													
3	$\emptyset$	B																												
2	$\emptyset$	$S$	$\emptyset$																											
1	A	A	C	C																										
	[	[	]	]																										
4	$V_{14} 1  V_{11} = \{A\}   V_{24} = \{B\}   S \rightarrow AB$ $V_{14} 2  V_{12} = \emptyset   V_{34} = \emptyset  $ ninguna $V_{14} 3  V_{13} = \emptyset   V_{44} = \emptyset  $ ninguna				<table border="1"> <tr><td>4</td><td><math>S</math></td><td></td><td></td><td></td></tr> <tr><td>3</td><td><math>\emptyset</math></td><td>B</td><td></td><td></td></tr> <tr><td>2</td><td><math>\emptyset</math></td><td><math>S</math></td><td><math>\emptyset</math></td><td></td></tr> <tr><td>1</td><td>A</td><td>A</td><td>C</td><td>C</td></tr> <tr><td></td><td>[</td><td>[</td><td>]</td><td>]</td></tr> </table>	4	$S$				3	$\emptyset$	B			2	$\emptyset$	$S$	$\emptyset$		1	A	A	C	C		[	[	]	]
4	$S$																													
3	$\emptyset$	B																												
2	$\emptyset$	$S$	$\emptyset$																											
1	A	A	C	C																										
	[	[	]	]																										

**Fig. 7.3** Aplicación de la regla inductiva del algoritmo CYK para la gramática del ejemplo 65.

*Ejercicio 134.* Escriba un programa que compruebe si un texto tiene los paréntesis, los corchetes y las llaves equilibradas.



# APÉNDICE A

---

## Forma normal de Chomsky

---

En este apéndice se expone a detalle un procedimiento seguro para transformar una GLC en FNC. El esquema general es el siguiente:

**Algoritmo para transformar una GLC en FNC.** Sea  $G$  una GLC. A partir de  $G$  se construye una GLC  $G'$  en FNC tal que  $L(G') = L(G) \setminus \{\varepsilon\}$  como sigue:

**1.<sup>er</sup> paso:**

- eliminación de producciones vacías,
- eliminación de producciones unitarias,
- eliminación de símbolos inútiles.

**2.<sup>o</sup> paso:** conseguir que en todos los cuerpos de longitud mayor o igual a 2 solamente aparezcan variables;

**3.<sup>er</sup> paso:** descomponer los cuerpos de longitud mayor o igual a 3 en una cascada de producciones de solo dos variables.

(A.1)

Cada paso del procedimiento (A.1) se especifica en las siguientes subsecciones. Para ejemplificar cada parte importante se parte de la gramática:

$$\begin{array}{ccccccc}
 S & \rightarrow & 0A0 & A & \rightarrow & C & B \rightarrow S \\
 | & & 1B1 & & | & A & | \varepsilon \\
 | & & BB & & & &
 \end{array} \tag{A.2}$$

### 1.<sup>er</sup> paso

#### Eliminación de producciones vacías

Dada  $G$  una GLC, la tarea en esta parte es construir una nueva GLC  $G'$  sin producciones de la forma  $A \rightarrow \varepsilon$  (llamadas *producciones vacías*) tal que  $L(G') = L(G) \setminus \{\varepsilon\}$ . Si  $G$  es la GLC (A.2) y se remueve inocentemente a la producción  $C \rightarrow \varepsilon$ , la gramática resultante no cumple con lo requerido, porque por ejemplo la cadena 00 no sería aceptada, de hecho la producción  $C \rightarrow \varepsilon$  influye directamente en la producción  $A \rightarrow C$  y finalmente en la producción  $S \rightarrow 0A0$ . El método para la construcción de la nueva GLC reconoce primero a las variables que producen a la cadena vacía y después hace ajustes en todas las producciones con el fin de no modificar el lenguaje aceptado, salvo la pertenencia de la cadena vacía, por supuesto. El procedimiento se detalla a continuación.

Lo primero es encontrar el subconjunto de variables sintácticas que sean *símbolos anulables*. Se dice que una variable  $A$  es un símbolo anulable si  $A \xrightarrow{*} \varepsilon$ . Si  $G$  es una GLC entonces un algoritmo iterativo para encontrar todos los símbolos anulables es el siguiente:

##### Algoritmo para encontrar los símbolos anulables.

- R. INICIAL si  $A \rightarrow \varepsilon$  es una producción de  $G$ , entonces  $G$  es un símbolo anulable;
- R. INDUTIVA si  $B \rightarrow C_1C_2 \cdots C_k$  es una producción de  $G$  donde cada  $C_i$  es un símbolo anulable para  $i = 1, 2, \dots, k$ , entonces  $B$  es anulable.

(A.3)

**Ejemplo 66.** La siguiente tabla indica el resultado de la búsqueda de símbolos anulables de la gramática (A.2) por medio del algoritmo (A.3):

símbolos anulables	número de iteración
$C$	1
$A$	2
$B$	3
$S$	4

Por tanto, el conjunto de símbolos anulables de la gramática (A.2) es  $\{C, A, B, S\}$ , es decir, en este caso todas las variables sintácticas son símbolos anulables.

Se está ahora en posición de establecer el procedimiento para la construcción de la nueva GLC sin producciones vacías, a partir de una GLC  $G = (T, V, S, P)$ :

**Algoritmo para eliminar producciones vacías.**

- 1.<sup>o</sup> Se encuentra el conjunto de símbolos anulables de  $G$ .
- 2.<sup>o</sup> Se construye  $G' = (V, T, S, P')$ , donde el conjunto de producciones  $P'$  se determina como sigue. Para cada producción

$$A \rightarrow X_1 X_2 \cdots X_k, \quad k \geq 1,$$

sea  $m$  la cantidad de símbolos  $X_i$  que son símbolos anulables. Si  $m \neq k$ , la nueva gramática tendrá  $2^m$  versiones de esta producción y si  $m = k$ ,  $2^m - 1$  versiones. Cada versión es de la forma

$$A \rightarrow \alpha_i,$$

donde las  $\alpha_i$  se establecen a partir de considerar todas las posibles combinaciones que resultan de eliminar de  $X_1 X_2 \cdots X_k$  ninguno, algunos (o posiblemente todos, en el caso  $m \neq k$ ) de los símbolos anulables.

- 3.<sup>o</sup> Se eliminan las producciones  $A \rightarrow \varepsilon$ .

(A.4)

**Ejemplo 67.** A continuación se construye una gramática sin producciones vacías a partir de la gramática (A.2), siguiendo el procedimiento (A.4). El conjunto

$$\{C, A, B, S\}$$

de símbolos anulables de la gramática (A.2) ha sido encontrado en el ejemplo 66, con ello se concluye el primer paso. El siguiente paso se especifica en la siguiente tabla:

producción	$k$	$m$	versiones generadas
$S \rightarrow 0A0$	3	1	$S \rightarrow 0A0 \mid 00$
$S \rightarrow 1B1$	3	1	$S \rightarrow 1B1 \mid 11$
$S \rightarrow BB$	2	2	$S \rightarrow BB \mid B \mid B$
$A \rightarrow C$	1	1	$A \rightarrow C$
$B \rightarrow S$	1	1	$B \rightarrow S$
$B \rightarrow A$	1	1	$B \rightarrow A$
$C \rightarrow S$	1	1	$C \rightarrow S$
$C \rightarrow \varepsilon$	1	0	$C \rightarrow \varepsilon$

Para el último paso, solo se tiene que eliminar la producción  $C \rightarrow \varepsilon$ . En resumen, después de descartar producciones que se repiten, la gramática resultante es:

$$\begin{array}{lllll}
 S & \rightarrow 0A0 & A & \rightarrow C & B \rightarrow S \\
 & | & & | & \\
 & 00 & & A & \\
 & | & & | & \\
 & 1B1 & & 11 & \\
 & | & & | & \\
 & BB & & BB & \\
 & | & & | & \\
 & B & & B &
 \end{array} \tag{A.5}$$

## Eliminación de producciones unitarias

Una *producción unitaria* es una producción de la forma

$$A \rightarrow B,$$

donde las variables  $A$  y  $B$  son sintácticas. El objetivo de esta parte es dada una gramática  $G = (V, T, S, P)$  construir otra gramática  $G'$  sin producciones unitarias tal que

$$L(G) = L(G').$$

El proceso preciso se expone a continuación.

**Algoritmo para eliminar producciones unitarias.**

1.<sup>o</sup> Se encuentra el conjunto de los *pares unitarios*  $(A, B)$  de  $G$ , es decir el conjunto de pares de variables sintácticas  $(A, B)$  tales que  $A \xrightarrow{*} B$ , mediante el siguiente algoritmo:

- R. INICIAL  $(A, A)$  es un par unitario, pues siempre se cumple que  $A \xrightarrow{*} A$  en cero pasos;
- R. INDUCTIVA si  $(A, B)$  es un par unitario y  $B \rightarrow C$  es una producción unitaria, entonces  $(A, C)$  es un par unitario.

2.<sup>o</sup> Se construye  $G' = (V, T, S, P')$ , donde el conjunto de producciones  $P'$  se determina como sigue. A cada par unitario  $(A, B)$  le corresponden las producciones del tipo

$$A \rightarrow \alpha$$

siempre que  $B \rightarrow \alpha$  sea una producción *no* unitaria de  $P$ . El conjunto de producciones  $P'$  se conforma con la unión de todas las producciones producidas a partir de cada par unitario.

(A.6)

**Ejemplo 68.** En este ejemplo se construye una gramática sin producciones unitarias a partir de la gramática (A.5). El primer paso consiste en encontrar todos los pares unitarios. La siguiente tabla muestra el proceso de búsqueda a través del algoritmo iterativo del primer punto del procedimiento (A.6):

pares unitarios	número de iteración
$(A, A), (B, B), (C, C), (S, S)$	1
$(A, C), (B, S), (B, A), (C, S), (S, B)$	2
$(A, S), (B, C), (C, B), (S, A)$	3
$(A, B), (C, A), (S, C)$	4

El conjunto de producciones que le corresponde a cada par unitario, según el segundo paso del procedimiento (A.6) se expone en la siguiente tabla:

par unitario	producciones asociadas
(A, A)	ninguna
(B, B)	ninguna
(C, C)	ninguna
(S, S)	$S \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$
(A, C)	ninguna
(B, S)	$B \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$
(B, A)	ninguna
(C, S)	$C \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$
(S, B)	ninguna
(A, S)	$A \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$
(B, C)	ninguna
(C, B)	ninguna
(S, A)	ninguna
(A, B)	ninguna
(C, A)	ninguna
(S, C)	ninguna

La gramática resultante es:

$$\begin{array}{lllll}
 S & \rightarrow 0A0 & B & \rightarrow 0A0 & C & \rightarrow 0A0 & A & \rightarrow 0A0 \\
 | & 00 & | & 00 & | & 00 & | & 00 \\
 | & 1B1 & | & 1B1 & | & 1B1 & | & 1B1 \\
 | & 11 & | & 11 & | & 11 & | & 11 \\
 | & BB & | & BB & | & BB & | & BB
 \end{array} \tag{A.7}$$

## Eliminación de símbolos inútiles

Sea  $G = (V, T, P, S)$  una gramática tal que  $L(G) \neq \emptyset$ . Se dice que una variable sintáctica o símbolo terminal es un *símbolo útil* si existe una cadena  $w \in T^*$  tal que

$$S \xrightarrow{*} \alpha X \beta \xrightarrow{*} w,$$

para algunas cadenas  $\alpha$  y  $\beta$  en  $(T \cup V)^*$ . Una variable sintáctica o un símbolo terminal se dice que es *inútil* si no es útil. A continuación se establece la estrategia general para eliminar de forma segura los símbolos inútiles de la GLC dada  $G$ .

**Algoritmo para eliminar los símbolos inútiles.**

1.<sup>o</sup> Se encuentran todos los símbolos *generadores* de  $G$ , es decir, aquellos símbolos  $X$  tales que  $X \xrightarrow{*} w$ , para alguna cadena de símbolos terminales  $w$ . El conjunto de símbolos generadores se puede hallar a través del algoritmo iterativo:

- R. INICIAL todo símbolo terminal es generador;
- R. INDUCTIVA si  $A \rightarrow \alpha$  es una producción de  $G$  y cada símbolo de  $\alpha$  es generador entonces  $A$  es un símbolo generador también.

2.<sup>o</sup> Se construye la gramática  $G' = (V', T, S, P')$  eliminando de  $V$  los símbolos no-generadores y también las producciones de  $P$  que involucran a uno o más símbolos no-generadores. Se ha de notar que el símbolo  $S$  no se elimina puesto que es claramente generador, pues se ha supuesto que  $L(G) \neq \emptyset$ .

3.<sup>o</sup> Se localizan todos los símbolos *alcanzables* de  $G'$ , es decir, aquellos símbolos  $X$  para los cuales existe una derivación de la forma  $S \xrightarrow{*} \alpha X \beta$ , para algunas  $\alpha$  y  $\beta$ . Se pueden encontrar los símbolos alcanzables de  $G'$  a través del siguiente algoritmo:

- R. INICIAL el símbolo  $S$  es alcanzable;
- R. INDUCTIVA si  $A$  es alcanzable y  $A \rightarrow \alpha$  está en  $P'$ , entonces todos los símbolos de  $\alpha$  son alcanzables también.

4.<sup>o</sup> Se eliminan todos los símbolos no-alcanzables de  $V'$  y  $T'$  y se eliminan todas las producciones de  $P'$  que involucren a uno o más símbolos no-alcanzables.

(A.8)

**Ejemplo 69.** El objetivo del ejemplo es construir una gramática libre de contexto sin símbolos inútiles que acepte el mismo lenguaje que la gramática (A.7) a través del procedimiento (A.8). Lo primero que se debe hacer es encontrar el conjunto de símbolos generadores de (A.7), según el algoritmo establecido en el primer punto de (A.8). Esta tarea se resume en la siguiente tabla:

símbolos generadores	número de iteración
0, 1	1
$S, B, C, A$	2

Es decir, en este caso, todos los símbolos de  $V$  y  $T$  son generadores, por tanto al aplicar el segundo paso la gramática resultante  $G'$  es igual que  $G$ . Lo que sigue es encontrar el conjunto de símbolos alcanzables de  $G'$  siguiendo el algoritmo del tercer punto de (A.8):

símbolos alcanzables	número de iteración
$S$	1
$0, A, 1, B$	2

En otras palabras, el único símbolo no-alcanzable es  $C$ . Por tanto, según el cuarto punto de (A.8), la gramática buscada es:

$$\begin{array}{llll}
 S & \rightarrow 0AO & B & \rightarrow 0AO \\
 | & 00 & | & 00 \\
 | & 1B1 & | & 1B1 \\
 | & 11 & | & 11 \\
 | & BB & | & BB
 \end{array} \quad \rightarrow 0AO \quad A \rightarrow 0AO$$

(A.9)

## 2.<sup>o</sup> paso

Se supone ahora que se parte de una gramática sin producciones vacías, unitarias y sin símbolos inútiles. Para cada símbolo terminal  $a$  que aparezca en un cuerpo de longitud mayor o igual a 2 se crea una nueva variable sintáctica  $A$  y una producción  $A \rightarrow a$ . En cada cuerpo de producción donde aparezca  $a$ , se reemplaza por la nueva variable  $A$ . Este simple procedimiento se ilustra en el siguiente ejemplo.

**Ejemplo 70.** Se considera la gramática (A.9). Al símbolo terminal 0 se le asocia una nueva variable  $O$  y al símbolo terminal 1 la nueva variable  $I$ . La gramática resultante es:

$$\begin{array}{llll}
 S & \rightarrow OAO & B & \rightarrow OAO \\
 | & OO & | & OO \\
 | & IBI & | & IBI \\
 | & II & | & II \\
 | & BB & | & BB
 \end{array} \quad \rightarrow OAO \quad O \rightarrow 0 \quad I \rightarrow 1$$

(A.10)

### 3.º paso

Se parte ahora de una gramática que tiene únicamente producciones de la forma  $A \rightarrow a$ ,  $A \rightarrow BC$  y

$$A \rightarrow B_1 B_2 \cdots B_k \quad (\text{A.11})$$

donde  $k \geq 3$ . Para cada producción tipo (A.11), se introducen  $r = k - 2$  nuevas variables sintácticas  $C_1, C_2, \dots, C_r$  y se reemplaza (A.11) por las producciones:

$$\begin{aligned} A &\rightarrow B_1 C_1 \\ C_1 &\rightarrow B_2 C_2 \\ &\vdots \\ C_r &\rightarrow B_{k-1} B_k \end{aligned}$$

**Ejemplo 71.** El objetivo de este ejemplo es aplicar el 3.º paso a la gramática (A.10). Asociada a la producción  $S \rightarrow OAO$  se crea una nueva variable  $X$  y se sustituye la producción por las producciones  $S \rightarrow OX$  y  $X \rightarrow AO$ . De igual forma, asociada a la producción  $S \rightarrow IBI$  se crea una nueva variable  $Y$  y se sustituye esta producción por las producciones  $S \rightarrow IY$  y  $Y \rightarrow BI$ . Así sucesivamente se razona con las otras producciones cuyo cuerpo tiene longitud mayor o igual a 3. La GLC resultante es:

$$\begin{array}{llllllll} S & \rightarrow OX & B & \rightarrow OX & A & \rightarrow OX & X & \rightarrow AO & O \rightarrow 0 \\ | & OO & | & OO & | & OO & Y & \rightarrow BI & I \rightarrow 1 \\ | & IY & | & IY & | & IY & & & \\ | & II & | & II & | & II & & & \\ | & BB & | & BB & | & BB & & & \end{array}$$

Hasta aquí se ha expuesto el procedimiento para transformar una GLC en FNC. Estos pasos son la esencia de la demostración del siguiente teorema, los detalles se pueden consultar en [11].

**Teorema 18.** *Sea  $G$  una GLC tal que  $L(G)$  contiene al menos una cadena distinta de  $\varepsilon$ . Entonces por medio del procedimiento (A.1) se puede construir una gramática  $G'$  en FNC tal que  $L(G') = L(G) \setminus \{\varepsilon\}$ .*

**Lista de ejercicios del apéndice**

*Ejercicio 135.* Pruebe que una variable sintáctica es un símbolo anulable si y solo si es reconocido por el algoritmo (A.3).

*Ejercicio 136.* Demuestre que el algoritmo iterativo del primer punto del procedimiento (A.6) encuentra efectivamente el conjunto de pares unitarios de la gramática dada.

*Ejercicio 137.* En [11, §7.1.1 y §7.1.2] se demuestra que la gramática resultante de aplicar el procedimiento (A.8) a la GLC dada  $G$  admite el mismo lenguaje que  $G$  y además no tiene símbolos inútiles. Un detalle importante es el orden de eliminación: primero símbolos no-generadores y después símbolos no-alcanzables. Si se invierte el orden de eliminación es posible que la gramática resultante no admita el mismo lenguaje que la GLC dada. Encuentre un ejemplo.

## APÉNDICE B

---

### Minimización de un autómata finito determinista

---

Se dice que dos autómatas finitos deterministas  $A$  y  $B$  son *equivalentes* si aceptan el mismo lenguaje. El objetivo de este apéndice es establecer un algoritmo que dado  $A = (Q, \Sigma, \delta, q_0, F)$ , encuentre un autómata equivalente pero con la menor cantidad de estados posibles. La idea es agrupar a los estados en clases de equivalencia. Cada clase de equivalencia reúne a estados que desempeñan la misma función dentro del autómata en el siguiente sentido: se dice que dos estados  $p$  y  $q$  de  $A$  son *equivalentes*, se escribe  $p \equiv q$ , si para cualquier cadena  $w$  de símbolos de entrada:

$$\hat{\delta}(p, w) \in F \text{ si y solo si } \hat{\delta}(q, w) \in F.$$

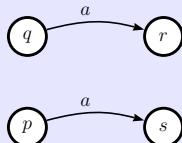
Obviamente, el símbolo  $\equiv$  define una relación de equivalencia, el lector puede verificarlo fácilmente. En lo que sigue, se denota por  $[q]$  al conjunto de estados equivalentes a  $q$ , es decir,  $[q] = \{p : p \equiv q\}$ . La construcción precisa del autómata minimizado se expone más adelante, antes se presenta un método para encontrar las clases de equivalencia.

## B.1 Estados equivalentes

El *algoritmo por llenado de tabla* tiene como fin encontrar pares llamados *distinguibles*. Si  $A$  es un autómata finito determinista con función de transición  $\delta$  y conjunto de estados de aceptación  $F$ , el algoritmo por llenado de tabla se establece mediante el siguiente proceso inductivo:

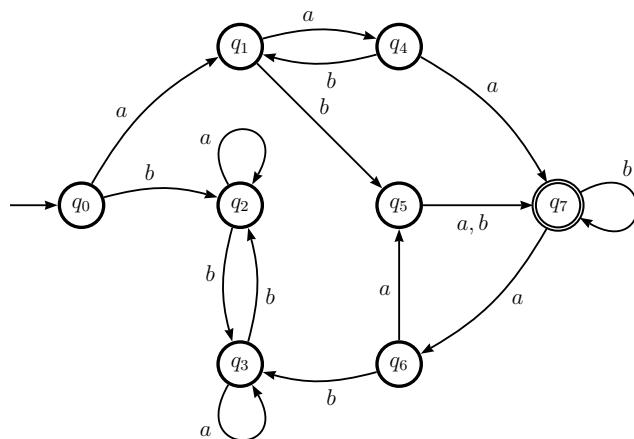
### Algoritmo por llenado de tabla.

- R. INICIAL si  $p \in F$  y  $q \notin F$  entonces  $p$  y  $q$  es un par distinguible;
- R. INDUCTIVA si  $r = \delta(p, a)$  y  $s = \delta(q, a)$  es un par distinguible entonces  $p$  y  $q$  también lo es.



El nombre del algoritmo se justifica en el ejemplo 72. Encontrar el conjunto de todos los pares distinguibles es parte del proceso de minimización de un autómata, pues los pares distinguibles serán exactamente aquellos que *no* son equivalentes, como lo establece el teorema 19.

**Ejemplo 72.** Considere el siguiente autómata:



Una forma de visualizar el procedimiento para encontrar pares distinguibles es partir de una tabla como esta:

$q_1$							
$q_2$							
$q_3$							
$q_4$							
$q_5$							
$q_6$							
$q_7$							
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$

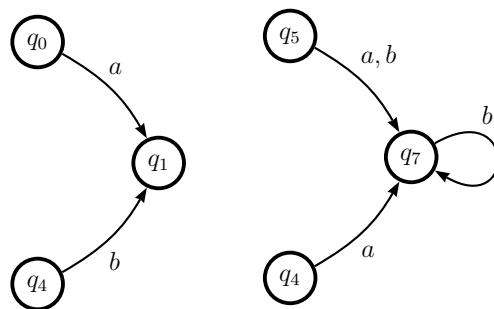
La idea es ir tachando cada espacio en blanco si el par correspondiente es distingible. Por ejemplo, la regla inicial el algoritmo distingue a los estados aceptación —en este caso únicamente  $q_7$ — de los estados que no son de aceptación. Se tachan entonces los cuadros correspondientes a los pares distinguibles:

$q_1$							
$q_2$							
$q_3$							
$q_4$							
$q_5$							
$q_6$							
$q_7$	$\times_{(1)}$	$\times_{(1)}$	$\times_{(1)}$	$\times_{(1)}$	$\times_{(1)}$	$\times_{(1)}$	
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$

(B.1)

La etiqueta (1) se indica que se trata de la primera iteración la cual por supuesto corresponde a la regla inicial. De la información de la tabla B.1 y de la regla inductiva, se deducen nuevos pares distinguibles. Por ejemplo, de saber que  $q_7$  y  $q_1$  es un par distingible se deduce que:  $q_0$  y  $q_4$  es un par distingible,  $q_0$  y  $q_5$  es un par distingible,  $q_4$  y  $q_5$  es un par distingible y finalmente que  $q_4$  y  $q_7$  es un par distingible, aunque esto último ya se sabía (ver figura B.1).

Al recorrer toda la información de la tabla B.1, la regla inductiva en la segunda iteración arroja las cruces etiquetadas con (2). El resultado es la siguiente tabla:



**Fig. B.1** Transiciones que llegan a  $q_1$  y a  $q_7$ .

$q_1$							
$q_2$							
$q_3$							
$q_4$	$\times_{(2)}$	$\times_{(2)}$	$\times_{(2)}$	$\times_{(2)}$			
$q_5$	$\times_{(2)}$	$\times_{(2)}$	$\times_{(2)}$	$\times_{(2)}$	$\times_{(2)}$		
$q_6$					$\times_{(2)}$	$\times_{(2)}$	
$q_7$	$\times_{(1)}$						
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$

Se aplica de nuevo la regla inductiva, a partir de los nuevos pares tachados etiquetados con (2). La tabla que se genera es la siguiente:

$q_1$	$\times_{(3)}$						
$q_2$		$\times_{(3)}$					
$q_3$		$\times_{(3)}$					
$q_4$	$\times_{(2)}$	$\times_{(2)}$	$\times_{(2)}$	$\times_{(2)}$			
$q_5$	$\times_{(2)}$	$\times_{(2)}$	$\times_{(2)}$	$\times_{(2)}$	$\times_{(2)}$		
$q_6$	$\times_{(3)}$	$\times_{(3)}$	$\times_{(3)}$	$\times_{(3)}$	$\times_{(2)}$	$\times_{(2)}$	
$q_7$	$\times_{(1)}$						
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$

Se repite el procedimiento con los pares etiquetados con (3). Los pares etiquetados con (4) no aportan nuevos pares distinguibles. Finalmente, la tabla queda así:

$q_1$	$X_{(3)}$						
$q_2$	$X_{(4)}$	$X_{(3)}$					
$q_3$	$X_{(4)}$	$X_{(3)}$					
$q_4$	$X_{(2)}$	$X_{(2)}$	$X_{(2)}$	$X_{(2)}$			
$q_5$	$X_{(2)}$	$X_{(2)}$	$X_{(2)}$	$X_{(2)}$	$X_{(2)}$		
$q_6$	$X_{(3)}$	$X_{(3)}$	$X_{(3)}$	$X_{(3)}$	$X_{(2)}$	$X_{(2)}$	
$q_7$	$X_{(1)}$						
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$

El resultado de la tabla nos indica que  $q_2$  y  $q_3$  no son distinguibles, mientras cualquier otro par es distingüible. El siguiente teorema señala que el algoritmo de llenado de tabla encuentra efectivamente los pares equivalentes.

**Teorema 19.** *Dos estados de un autómata finito determinista no son equivalentes si y solo si son distinguibles por el algoritmo de llenado de tabla.*

Prueba. Sean  $A = (Q, \Sigma, \delta, q_0, F)$  y

$$W(p, q) = \{w \in \Sigma^* : \hat{\delta}(p, w) \in F \text{ y } \hat{\delta}(q, w) \notin F, \text{ o al revés}\}.$$

Se ha de notar que dos estados  $p$  y  $q$  son equivalentes si y solo si  $W(p, q) = \emptyset$ , o lo que es lo mismo,  $p$  y  $q$  no son equivalentes si y solo si existe una cadena en  $W(p, q)$ . Se prueba a continuación el siguiente enunciado sobre números naturales, y con esta demostración particularmente se prueba el teorema.

$P(n)$ : el algoritmo distingue por primera vez a dos estados  $p$  y  $q$  de  $A$  en una iteración menor o igual a  $n$  si y solo si existe una cadena  $w_0 \in W(p, q)$  de longitud  $n - 1$  tal que, para toda cadena  $w \in W(p, q)$ ,  $|w_0| \leq |w|$ .

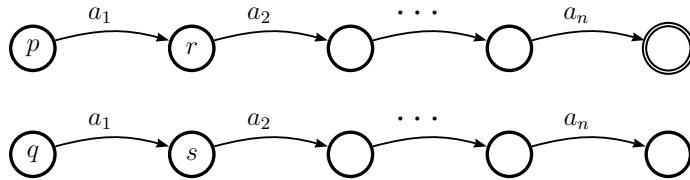
Se supone que el algoritmo distingue a  $p$  de  $q$  en la primera iteración, eso significa que alguno de los estados  $p$  o  $q$  está en  $F$ , pero no ambos. Basta tomar  $w = \varepsilon$ . Ahora se supone que existe una cadena  $w_0 \in W(p, q)$  de longitud 0. Entonces,  $w$  tiene que ser necesariamente la cadena vacía. Por tanto, alguno de los estados  $\hat{\delta}(p, \varepsilon) = p$  o  $\hat{\delta}(q, \varepsilon) = q$  está en  $F$ , pero no ambos. Luego, el algoritmo distingue a  $p$  y a  $q$  en la primera iteración. Con esto se demuestra  $P(1)$ .

Se supone cierta  $P(n)$  (hipótesis de inducción). Sean  $p$  y  $q$  un par que se distingue en la iteración  $n + 1$  por primera vez. Existen dos estados  $r$  y  $s$  distinguibles por primera vez en la iteración  $n$  y existe un símbolo de entrada  $a_1$

tales que (ver figura B.2)  $r = \delta(p, a_1)$  y  $s = \delta(q, a_1)$ . Por hipótesis de inducción, existe una cadena  $x_0 \in W(r, s)$  de longitud  $n - 1$  tal que  $|x_0| \leq |x|$ , para toda  $x \in W(r, s)$ . Sea  $w_0 = a_1 x_0$ . Note que,  $\hat{\delta}(r, x_0) = \hat{\delta}(p, w_0)$ . De igual modo,  $\hat{\delta}(s, x_0) = \hat{\delta}(q, w_0)$ . Por tanto,  $w_0 \in W(p, q)$ . Resta probar que para toda cadena  $w \in W(p, q)$ ,  $|w_0| \leq |w|$ . Se razona por contradicción: sea

$$W'(p, q) = \{z \in W(p, q) : |z| < |w_0| = n\} \neq \emptyset,$$

y  $z_0$  una cadena en  $W'(p, q)$  tal que  $|z_0| \leq |z|$ , para todo  $z \in W'(p, q)$ . Si  $w$  es una cadena en  $W(p, q) \setminus W'(p, q)$ , entonces  $|w_0| \leq |w|$ . Como  $z_0 \in W'(p, q)$ , entonces  $|z_0| < |w_0|$ . Por tanto  $|z_0| < |w|$ . En conclusión,  $z_0 \in W(p, q)$  y  $|z_0| \leq |w|$  para todo  $w \in W(p, q)$ . Por hipótesis de inducción, el algoritmo distingue a  $p$  de  $q$  en la iteración  $|z_0| + 1 < n + 1$ , por tanto  $p$  y  $q$  no se distinguen por primera vez en la iteración  $n + 1$  y se llega a una contradicción.



**Fig. B.2** Movimientos en un autómata de pares distinguibles.

Se supone ahora que existe una cadena  $w_0 = a_1 a_2 \dots a_n \in W(p, q)$  de longitud  $n$  tal que  $n \leq |w|$ , para toda cadena  $w \in W(p, q)$ . Sean  $r = \delta(p, a_1)$ ,  $s = \delta(q, a_1)$  y  $x_0 = a_2 \dots a_n$ . Puesto que  $\hat{\delta}(r, x_0) = \hat{\delta}(p, w_0)$  y  $\hat{\delta}(s, x_0) = \hat{\delta}(q, w_0)$  entonces  $x_0 \in W(r, s)$ . Además, para todo  $x \in W(r, s)$ ,  $|x_0| \leq |x|$ . Si existiera algún  $x \in W(r, s)$  tal que  $|x| < |x_0|$ , entonces la cadena  $a_1 x$  estaría en  $W(p, q)$  y sería más pequeña que  $w_0$  y esto no puede ser. Por tanto, por hipótesis de inducción, los estados  $r$  y  $s$  don distinguibles en la iteración  $n$ , luego  $p$  y  $q$  son distinguidos en la iteración  $n + 1$ .  $\square$

**Ejemplo 73.** Las clases de equivalencia que resultan de aplicar el algoritmo de llenado de tabla al autómata del ejemplo 72 son:  $[q_i] = \{q_i\}$  para  $i = 0, 1, 4, 5, 6, 7$  y  $[q_2] = [q_3] = \{q_2, q_3\}$ .

## B.2 Construcción del autómata mínimo

En esta sección, se establece un procedimiento para, a partir de un autómata finito determinista  $A$ , encontrar un autómata con el menor número de estados que acepta a  $L(A)$  como lenguaje.

**Definición.** Sea  $A = (Q, \Sigma, \delta, q_0, F)$  un autómata finito determinista. Se define el *autómata mínimo* asociado a  $A$  como el autómata:

$$A_{\min} = (Q/\equiv, \Sigma, \delta_{\min}, [q_0], F_{\min}),$$

con función de transición:

$$\delta_{\min}([q], a) = [\delta(q, a)]. \quad (\text{B.2})$$

La función  $\delta_{\min}$  está bien definida [ejercicio 140]. Las clases de equivalencia de:

$$F_{\min} = \{[q] : q \in F\} \quad (\text{B.3})$$

solo contienen elementos de  $F$ , ya que en la primera iteración del algoritmo por llenado de tabla, se distinguen los estados de aceptación de los que no lo son. En otras palabras, si  $[p] \in F_{\min}$ , entonces  $[p] = \{q \in Q : q \equiv p\} \subset F$ . Se cumple además la siguiente propiedad [ejercicio 139].

**Propiedad 8.** Sea  $A = (Q, \Sigma, \delta, q_0, F)$  un autómata finito determinista. Para toda cadena  $w \in \Sigma^*$  y para todo estado  $q$ ,

$$\hat{\delta}_{\min}([q], w) = [\hat{\delta}(q, w)].$$

**Teorema 20.** Sea  $A = (Q, \Sigma, \delta, q_0, F)$  un autómata finito determinista sin estados inaccesibles. Se afirma que:

- I. El autómata  $A_{\min}$  acepta el mismo lenguaje que  $A$ .
- II. Si  $B = (P, \Sigma, \gamma, p_0, H)$  es otro autómata finito determinista tal que acepta el mismo lenguaje que  $A$ , entonces  $|Q/\equiv| \leq |P|$ .

*Prueba.* Para probar que  $L(A_{\min}) = L(A)$ , es suficiente ver que  $\hat{\delta}_{\min}([q_0], w)$  está en  $F_{\min}$  si y solo si  $\hat{\delta}(q_0, w)$  está en  $F$ . Pero este hecho es una consecuencia trivial de la propiedad 8, pues  $\hat{\delta}_{\min}([q_0], w) = [\hat{\delta}(q_0, w)]$ .

Resta demostrar II, se prueba primero que cada estado  $s \in Q/\equiv$  es equivalente a algún estado  $r \in B$ . Los estados  $[q_0] \in Q/\equiv$  y  $p_0 \in P$  son equivalentes puesto que  $L(A_{\min}) = L(B)$  [ejercicio 138]. Sea  $\eta$  la función de transición de  $A_{\min} \cup B$ , por el ejercicio 140, para cada símbolo de entrada  $a$ ,  $\eta([q_0], a) \in Q/\equiv$  es equivalente a  $\eta(p_0, a) \in P$ . Razonando de la misma forma, todos los sucesores de  $\eta([q_0], a)$  son equivalentes a algún estado de  $P$ . Luego, cada estado de  $Q/\equiv$  es equivalente a un estado de  $P$ . Por otro lado, si  $|Q/\equiv|$  fuera mayor que  $|P|$  existirían dos estados de  $Q/\equiv$  que son equivalentes a un mismo estado de  $P$ , lo que contradice la definición del conjunto  $Q/\equiv$ .  $\square$

En resumen, dado un autómata finito determinista  $A$  se construye  $A_{\min}$  a través de los siguientes pasos:

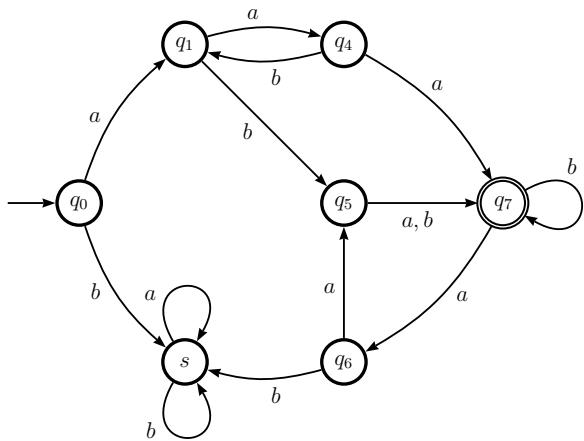
- se encuentra el conjunto de estados inaccesibles del autómata  $A$ ,
- se considera un nuevo  $A' = (Q', \Sigma, \delta', q_0, F')$  sin estados inaccesibles,
- se aplica el algoritmo por llenado a  $A'$  para encontrar  $Q_{\min} = Q'/\equiv$ ,
- se calcula  $\delta_{\min}$  y  $F_{\min}$  a partir de la función de transición  $\delta'$  y el conjunto  $F'$  según las definiciones (B.2) y (B.3),
- finalmente,  $A_{\min} = (Q/\equiv, \Sigma, \delta_{\min}, [q_0], F_{\min})$ .

**Ejemplo 74.** Sea  $A$  el autómata del ejemplo 72. Como el conjunto de estados inaccesibles de  $A$  es vacío, se ignora el segundo paso, en otras palabras  $A = A'$ . Como se calculó en el ejemplo 73,  $Q/\equiv$  consta de las clases de equivalencia  $[q_i] = \{q_i\}$  para  $i = 0, 1, 4, 5, 6, 7$  y  $[s] = \{q_2, q_3\}$ . Por tanto, los estados  $q_2$  y  $q_3$  son reemplazados por un único estado  $s$ . El diagrama de transición correspondiente a  $A_{\min}$  se muestra en la figura B.3.

## Lista de ejercicios del apéndice B

*Ejercicio 138.* Sean  $A = (Q, \Sigma, \delta, q_0, F)$  y  $B = (P, \Sigma, \gamma, p_0, H)$  dos autómatas finitos deterministas. Sea  $A \cup B$  el autómata con conjunto de estados  $Q \cup P$ , alfabeto de símbolos de entrada  $\Sigma$ , estado inicial  $q_0$ , conjunto de estados de aceptación  $F \cup H$  y función de transición  $\eta : (Q \cup P) \times \Sigma \rightarrow (Q \cup P)$  definida para toda entrada  $a$  por  $\delta(r, a)$  si  $r \in Q$  y  $\gamma(r, a)$  si  $r \in P$ . Demuestre que  $q_0$  y  $p_0$  son equivalentes si y solo si  $L(A) = L(B)$ .

*Ejercicio 139.* Demuestre la propiedad 8.



**Fig. B.3** Autómata del ejemplo 72 minimizado.

*Ejercicio 140.* Sea  $A$  un autómata finito determinista con función de transición  $\delta$  y alfabeto  $\Sigma$ . Demuestre que si  $p$  y  $q$  son estados equivalentes entonces  $r = \delta(p, a)$  y  $s = \delta(q, a)$  también lo son, para cualquier  $a \in \Sigma$ . ¿Porqué esto implica que  $\delta_{\min}$  de la ecuación (B.2) está bien definida?

*Ejercicio 141.* En la demostración del teorema 20, ¿dónde se usó el hecho de que el autómata  $A$  no debe tener estados inaccesibles?



---

## Referencias

---

1. A. V. Aho, M. S. Lam, R. Sethi y J. D. Ullman, *Compiladores: principios, técnicas y herramientas*, 2.<sup>a</sup> ed., Pearson y Addison Wesley, 2008.
2. Y. Bar-Hillel, M. Perles y E. Shamir, «On formal properties of simple phrase-structure grammars», *Z. Phonetik. Sprachwiss. Kommunikationsforsch.*, vol. 14, 1961, 143–172.
3. D. C. Cantor, «On the ambiguity problem of backus systems», *J. ACM*, vol. 9, núm. 4, 1962, 477–479.
4. N. Chomsky, «Three models for the description of languages», *IRE Trans. on Information Theory*, vol. 2, núm. 3, 1956, 113–124.
5. J. Cocke y J. T. Schwartz, «Programming languages and their compilers», reporte técnico, Courant Institute of Mathematical Sciences, 1970.
6. M. D. Davis, R. Sigal y E. J. Weyuker, *Computability, complexity, and languages*, Academic Press, New York, NY, USA, 2003.
7. A. de Academias de la Lengua Española, *Diccionario de la lengua española*, 23.<sup>a</sup> ed., Espasa, Madrid, España, 2014.
8. J. Evey, «Application of pushdown store machines», *Proc. Fall Joint Computer Conference*, 1963, 215–227.
9. P. C. Fischer, «On computability by certain classes of restricted turing machines», *Proc. Fourth Annl. Symposium on Switching Circuit Theory and Logical Desing*, 1963, 23–32.
10. R. W. Floyd, «On ambiguity in phrase-structure languages», *Comm. ACM*, vol. 5, núm. 10, 1962, 526–534.
11. J. E. Hopcroft y J. D. Ullman, *Introduction to automata theory, languages and computation*, 3.<sup>a</sup> ed., Addison-Wesley, Reading, Mass., 2007.
12. D. A. Huffman, «The synthesis of sequential switching circuits», *J. Franklin Inst.*, vol. 257, núm. 3, 1954, 161–190.
13. T. Kasami, «An efficient recognition and syntax-analysis algorithm for context-free languages», reporte técnico, Air Force Cambridge Research Lab, 1965.
14. S. C. Kleene, *Automata studies*, cap. «Representation of events in nerve nets and finite automata», 3–42, Princeton Univ. Press, 1956.
15. W. S. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity», *Bull. Math. Biophysics*, vol. 5, 1943, 115–133.

16. R. McNaughton y H. Yamada, «Regular expressions and state graphs for automata», *IEEE Trans. Electronic Computers*, vol. 9, núm. 1, 1960, 39–47.
17. G. H. Mealy, «A method for synthesizing sequential circuits», *Bell System Technical J.*, vol. 34, núm. 5, 1955, 1045–1079.
18. E. F. Moore, *Automata studies*, cap. «Gedanken experiments on sequential machines», 129–153, Princeton Univ. Press, 1956.
19. A. G. Oettinger, «Automatic syntactic analysis and the pushdown store», *Proc. Symposia on Applied Math.*, vol. 12, 1961, 104–129.
20. M. O. Rabin y D. Scott, «Finite automata and their decision problems», *IBM J. Research and Development*, vol. 3, núm. 2, 1959, 115–125.
21. A. Salomaa, «Two complete axiom systems for the algebra of regular events», *J. ACM*, vol. 13, núm. 1, 1966, 158–169.
22. M. P. Schutzenberger, «On context-free languages and pushdown automata», *Information and Control*, vol. 6, núm. 3, 1963, 246–264.
23. M. Sipser, *Introduction to the theory of computation*, 3.<sup>a</sup> ed., Course Tech., 2012.
24. D. H. Younger, «Recognition and parsing of context-free languages in time  $n^3$ », *Information and Control*, vol. 10, núm. 2, 1967, 189–208.