



UNIVERSIDAD  
DE GRANADA

---

# Modelos de Computación

*Temario*

Ismael Sallami Moreno

*Recursos Ingeniería Informática y Ade*

## Licencia

Este trabajo está bajo una Licencia  
Creative Commons BY-NC-ND 4.0.

Permisos: Se permite compartir, copiar  
y redistribuir el material en cualquier  
medio o formato.

Condiciones: Es necesario dar crédito  
adecuado, proporcionar un enlace a la  
licencia e indicar si se han realizado  
cambios. No se permite usar el material  
con fines comerciales ni distribuir  
material modificado.



# Modelos de Computación

Ismael Sallami Moreno



# Índice general

<b>I Teoría</b>	<b>7</b>
1 Introducción a la Computación	9
1.1 Fundamentos y Orígenes . . . . .	9
1.2 Elementos Básicos: Lenguajes Formales . . . . .	10
1.3 Operaciones Esenciales . . . . .	11
1.4 Generación de Lenguajes . . . . .	12
2 Autómatas Finitos y Lenguajes Regulares	15
2.1 Fundamentos y Aplicaciones de los Autómatas Finitos . . . . .	15
2.2 Autómata Finito Determinista (AFD) . . . . .	17
2.3 Tipos de Autómatas Finitos y Equivalencias . . . . .	19
2.4 Expresiones Regulares (ER) y El Teorema de Kleene . . . . .	22
2.5 Gramáticas Regulares (GR) . . . . .	25
3 Relaciones de Ejercicios	29
3.1 Lenguajes y Gramáticas . . . . .	29
3.2 Cálculo de Gramáticas . . . . .	41
3.3 Autómatas Finitos . . . . .	46
4 Tipo Test	63
4.1 Tema 1 . . . . .	63
4.2 Tema 2 . . . . .	66
4.3 Tema 3 . . . . .	69
4.4 Tema 4 . . . . .	76
4.5 Tema 5 . . . . .	79
4.6 Tema 6 . . . . .	83

<b>II Exámenes</b>	<b>89</b>
<b>1 Parcial I</b>	<b>91</b>
<b>1.1 Examen I . . . . .</b>	<b>91</b>
<b>1.2 Examen II . . . . .</b>	<b>95</b>

# **Parte I**

## **Teoría**



# Capítulo 1

---

## Introducción a la Computación

---

### 1.1 Fundamentos y Orígenes

#### 1.1.1 Preguntas Clave: Definición, problemas resolubles, y Modelos de Computación

La Ciencia de la Computación se fundamenta en la búsqueda de respuestas a preguntas trascendentales sobre los límites de la resolución automática de problemas. Entre las cuestiones centrales se encuentran:

- **¿Qué puede ser resuelto de forma automática?** Esta pregunta indaga sobre la existencia de algoritmos para problemas dados, es decir, procedimientos mecánicos sistemáticos que garantizan una terminación para cualquier entrada válida.
- **¿Qué puede ser resuelto de forma eficiente?** Más allá de la mera resolubilidad, este interrogante aborda la viabilidad práctica de los algoritmos en términos de recursos computacionales como tiempo y espacio.
- **¿Qué estructuras son comunes en la computación con símbolos y cómo pueden ser procesadas?** Esta línea de investigación se enfoca en el estudio de patrones en conjuntos de cadenas y en los mecanismos formales para su manipulación.

Para abordar estas preguntas, se desarrollan **Modelos de Computación**. Estos modelos son abstracciones matemáticas de un dispositivo de cómputo, que permiten analizar sus capacidades y limitaciones de manera formal. Un concepto clave en este ámbito es el de **autómata**, concebido como un sistema algebraico que procesa información. Un autómata, en su forma más general, se define por sus conjuntos de estados, señales de entrada y señales de salida, junto con las operaciones que gobiernan sus transiciones y respuestas. El estudio de estos modelos, como las Máquinas de Turing y sus simplificaciones (autómatas finitos, autómatas con pila), constituye la base de la Teoría de la Computabilidad y de Lenguajes Formales.

#### 1.1.2 Breve Historia: Desde precursores hasta Autómatas y el Problema de la Parada

Los fundamentos de la computación se remontan a los trabajos de lógicos y matemáticos como Russell, Hilbert y Boole, quienes sentaron las bases del formalismo matemático moderno. En las décadas de 1930 y 1940, figuras como Alan Turing y Alonzo Church formalizaron la noción de "computabilidad". Turing, en particular, propuso un modelo teórico, la **Máquina de Turing**, que se considera la definición de un dispositivo de cómputo universal.

En su búsqueda por resolver los 23 problemas propuestos por David Hilbert, la comunidad científica demostró que ciertos problemas son **indecidibles**, es decir, no admiten una solución algorítmica.

general. El ejemplo canónico es el **Problema de la Parada (Halting Problem)**.

**Definición 1.1 (Problema de la Parada).** No existe un programa de ordenador universal, llamémoslo  $\text{Stops}(P, x)$ , que pueda tomar como entrada cualquier otro programa  $P$  y unos datos  $x$  y determinar, en un tiempo finito, si  $P$  terminará su ejecución con la entrada  $x$  o si entrará en un bucle infinito.

La imposibilidad de resolver este problema fue un resultado trascendental que estableció los límites fundamentales de lo que puede ser computado.

Paralelamente, el desarrollo de los primeros lenguajes de programación como FORTRAN, COBOL y LISP en los años 50 impulsó el estudio de la sintaxis. En esta década, Noam Chomsky, junto con Michael Rabin y Dana Scott, formalizó la teoría de los **Autómatas y Lenguajes Formales**, estableciendo una jerarquía de gramáticas y máquinas que reconocen distintas clases de patrones.

## 1.2 Elementos Básicos: Lenguajes Formales

### 1.2.1 Alfabetos

La teoría de lenguajes formales se construye a partir de conceptos básicos. El más fundamental es el de alfabeto.

**Definición 1.2 (Alfabeto).** Un **alfabeto** es un conjunto **finito** y no vacío de elementos denominados **símbolos** o **letras**.

*Nota 1.1* . Se denotan los alfabetos con letras mayúsculas como  $\Sigma, A, B$ , etc., y sus símbolos con letras minúsculas como  $a, b, c$  o números.

**Ejemplo 1.1** . Algunos alfabetos comunes son:

- El alfabeto binario:  $\Sigma = \{0, 1\}$ .
- El alfabeto de letras minúsculas:  $\Sigma = \{a, b, c, \dots, z\}$ .
- Un alfabeto de vectores:  $B = \{\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 1, 1 \rangle\}$ .

### 1.2.2 Palabras

**Definición 1.3 (Palabra).** Una **palabra** (también llamada **cadena** o **string**) sobre un alfabeto  $A$  es una sucesión finita de símbolos de  $A$ . La **longitud** de una palabra  $u$ , denotada por  $|u|$ , es el número de símbolos que la componen.

*Nota 1.2* . Las palabras se denotan con letras minúsculas como  $u, v, w, x, y, z$ . El conjunto de todas las palabras posibles sobre un alfabeto  $A$  se denota como  $A^*$ .

**Definición 1.4 (Palabra Vacía).** La **palabra vacía**, denotada por  $\epsilon$  (o a veces  $\lambda$ ), es la única palabra de longitud cero,  $|\epsilon| = 0$ . Es un elemento de  $A^*$  para cualquier alfabeto  $A$ .

*Nota 1.3* . El conjunto de todas las palabras no vacías sobre un alfabeto  $A$  se denota como  $A^+$ . Por lo tanto,  $A^+ = A^* \setminus \{\epsilon\}$ .

El conjunto de palabras  $A^*$  con la operación de concatenación forma una estructura algebraica conocida como monoide, donde la palabra vacía  $\epsilon$  es el elemento neutro.

### 1.2.3 Lenguajes

**Definición 1.5 (Lenguaje).** Un **lenguaje**  $L$  sobre un alfabeto  $A$  es cualquier subconjunto de  $A^*$ , es decir,  $L \subseteq A^*$ . Las palabras que pertenecen al lenguaje se denominan a menudo cadenas del lenguaje.<sup>o</sup> "frases".

**Ejemplo 1.2 .** Dado el alfabeto  $A = \{a, b\}$ :

- $L_1 = \{a, b, \epsilon\}$  es un lenguaje finito con tres palabras.
- $L_2 = \{a^i b^i \mid i \geq 0\}$  es un lenguaje infinito que contiene palabras con igual número de a's seguidas de b's.
- $L_3 = \{uu^{-1} \mid u \in \{a, b\}^*\}$  es el lenguaje de los palíndromos de longitud par.
- $L_4 = \{a^{n^2} \mid n \geq 1\}$  es el lenguaje de cadenas de a's cuya longitud es un cuadrado perfecto.

Una propiedad fundamental del conjunto de todos los lenguajes sobre un alfabeto no vacío es su **no numerabilidad**. Mientras que el conjunto de todas las palabras  $A^*$  es numerable, el conjunto de todos sus subconjuntos (es decir, el conjunto de todos los lenguajes,  $\mathcal{P}(A^*)$ ) es no numerable. Esto tiene una profunda implicación: existen más lenguajes que programas para describirlos o reconocerlos, lo que demuestra que debe haber problemas (lenguajes) que no son computacionalmente resolubles.

## 1.3 Operaciones Esenciales

### 1.3.1 Sobre Palabras

Existen varias operaciones fundamentales que se pueden aplicar a las palabras.

**Definición 1.6 (Concatenación).** Dados  $u = a_1 \dots a_n$  y  $v = b_1 \dots b_m$  dos palabras sobre un alfabeto  $A$ , su **concatenación**, denotada  $uv$ , es la palabra  $a_1 \dots a_n b_1 \dots b_m$ .

La concatenación es asociativa y tiene a  $\epsilon$  como elemento neutro ( $u\epsilon = \epsilon u = u$ ).

**Definición 1.7 (Iteración).** La **iteración n-ésima** de una palabra  $u$ , denotada  $u^n$ , es la concatenación de  $u$  consigo misma  $n$  veces. Se define formalmente como  $u^0 = \epsilon$  y  $u^{i+1} = u^i u$  para  $i \geq 0$ .

**Definición 1.8 (Palabra Inversa).** La **palabra inversa** de  $u = a_1 \dots a_n$ , denotada  $u^{-1}$ , es la palabra  $a_n \dots a_1$ .

*Nota 1.4 .* La operación de inversión no es un homomorfismo. Por ejemplo, si  $f(u) = u^{-1}$ , entonces  $f(uv) = (uv)^{-1} = v^{-1}u^{-1}$ , que en general es distinto de  $f(u)f(v) = u^{-1}v^{-1}$ .

### 1.3.2 Sobre Lenguajes

Al ser los lenguajes conjuntos de palabras, heredan las operaciones de conjuntos como la **unión** ( $L_1 \cup L_2$ ), **intersección** ( $L_1 \cap L_2$ ) y **complementario** ( $\bar{L} = A^* \setminus L$ ). Adicionalmente, se definen operaciones específicas.

**Definición 1.9 (Concatenación de Lenguajes).** La **concatenación** de dos lenguajes  $L_1$  y  $L_2$  es el lenguaje  $L_1L_2 = \{uv \mid u \in L_1, v \in L_2\}$ .

Esta operación es asociativa y tiene como elemento neutro el lenguaje  $\{\epsilon\}$ .

**Definición 1.10 (Clausura de Kleene y Clausura Positiva).** Dado un lenguaje  $L$ , su **clausura de Kleene** (o estrella), denotada  $L^*$ , es la unión de todas sus potencias:

$$L^* = \bigcup_{i \geq 0} L^i = L^0 \cup L^1 \cup L^2 \cup \dots$$

La **clausura positiva**, denotada  $L^+$ , se define de manera similar pero excluyendo la potencia cero:

$$L^+ = \bigcup_{i \geq 1} L^i = L^1 \cup L^2 \cup \dots$$

donde  $L^0 = \{\epsilon\}$  y  $L^{i+1} = LL^i$ .

Se cumple que  $L^+ = L^*$  si y solo si  $\epsilon \in L$ . Si  $\epsilon \notin L$ , entonces  $L^* = L^+ \cup \{\epsilon\}$ . Si  $L$  es un lenguaje no vacío,  $L^*$  siempre es infinito.

## 1.4 Generación de Lenguajes

### 1.4.1 Gramáticas: Definición y Lenguaje Generado

Los lenguajes, especialmente los infinitos, requieren un mecanismo finito para su descripción. Las gramáticas generativas, introducidas por Noam Chomsky, son uno de dichos mecanismos.

**Definición 1.11 (Gramática Generativa).** Una **gramática generativa** es una tupla  $G = (V, T, P, S)$ , donde:

- $V$  es un alfabeto finito de **variables** o símbolos no terminales.
- $T$  es un alfabeto finito de **símbolos terminales**, disjunto de  $V$ .
- $P$  es un conjunto finito de **reglas de producción** de la forma  $\alpha \rightarrow \beta$ , donde  $\alpha, \beta \in (V \cup T)^*$  y  $\alpha$  contiene al menos una variable.
- $S \in V$  es el **símbolo inicial** o axioma.

Una palabra  $\beta$  se **deriva** de  $\alpha$  en un paso ( $\alpha \Rightarrow \beta$ ) si existe una regla  $\gamma \rightarrow \phi \in P$  y  $\alpha$  puede escribirse como  $\alpha_1\gamma\alpha_2$ , de tal forma que  $\beta = \alpha_1\phi\alpha_2$ . La relación de derivación en múltiples pasos se denota por  $\Rightarrow^*$ .

**Definición 1.12 (Lenguaje Generado).** El **lenguaje generado** por una gramática  $G$ , denotado  $L(G)$ , es el conjunto de todas las palabras compuestas exclusivamente por símbolos terminales que pueden derivarse a partir del símbolo inicial  $S$ :

$$L(G) = \{u \in T^* \mid S \Rightarrow^* u\}$$

**Ejemplo 1.3 .** Sea la gramática  $G = (\{S\}, \{a, b\}, \{S \rightarrow aSb, S \rightarrow \epsilon\}, S)$ . Esta gramática genera el lenguaje  $L(G) = \{a^i b^i \mid i \geq 0\}$ . Por ejemplo, para generar  $aabb$ :  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aab\epsilon b = aabb$ .

### 1.4.2 Jerarquía de Chomsky

Noam Chomsky clasificó las gramáticas en cuatro tipos, estableciendo una jerarquía basada en las restricciones impuestas sobre sus reglas de producción.

**Definición 1.13 (Jerarquía de Chomsky).** La jerarquía de Chomsky se define de la siguiente manera:

- **Tipo 0 (Sin restricciones):** Corresponde a cualquier gramática generativa. Estas gramáticas generan los lenguajes **recursivamente enumerables**, que son aquellos reconocibles por una Máquina de Turing.
- **Tipo 1 (Dependientes del contexto):** Todas las producciones son de la forma  $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ , con  $A \in V$ ,  $\alpha_1, \alpha_2, \beta \in (V \cup T)^*$  y  $\beta \neq \epsilon$ . Se permite la regla  $S \rightarrow \epsilon$  si  $S$  no aparece en la parte derecha de ninguna regla. Generan los lenguajes **dependientes del contexto**. Un ejemplo es el lenguaje  $\{a^n b^n c^n \mid n \geq 1\}$ .
- **Tipo 2 (Independientes del contexto):** Todas las producciones son de la forma  $A \rightarrow \alpha$ , con  $A \in V$  y  $\alpha \in (V \cup T)^*$ . Generan los lenguajes **independientes del contexto**.
- **Tipo 3 (Regulares):** Todas las producciones son de la forma  $A \rightarrow uB$  o  $A \rightarrow u$ , donde  $A, B \in V$  y  $u \in T^*$ . Generan los **lenguajes regulares**.

Esta jerarquía establece una relación de inclusión estricta entre las familias de lenguajes generados, denotadas por  $\mathcal{L}_i$  para el tipo  $i$ :

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$$

. Por ejemplo, todo lenguaje regular es también independiente del contexto, pero no a la inversa.



# Autómatas Finitos y Lenguajes Regulares

---

## 2.1 Fundamentos y Aplicaciones de los Autómatas Finitos

### 2.1.1 Contexto y Relevancia

#### *Importancia de los Autómatas Finitos (AF) en la computación*

Los Autómatas Finitos (AF) constituyen modelos matemáticos esenciales que residen en el núcleo teórico de la Ciencia de la Computación. El estudio de los autómatas es fundamental para investigar qué puede resolverse de forma automática y qué estructuras, basadas en palabras y símbolos, pueden procesarse eficientemente en un ordenador. El concepto de autómata surge en diversos problemas asociados con la informática, los sistemas de redes y la teoría de control. La estructura matemática de los autómatas se basa en argumentos intuitivos que reflejan la entidad de los autómatas reales (no necesariamente físicos).

- Aplicaciones en análisis léxico (compiladores): Una aplicación práctica crucial de los AF se encuentra en la construcción de analizadores léxicos (lexers) dentro de los compiladores. Un analizador léxico tiene la función de reconocer secuencias de caracteres que forman *tokens* (como palabras clave o identificadores), un problema que se mapea directamente al reconocimiento de lenguajes regulares.
- Verificación de circuitos digitales: Los AF son ampliamente utilizados en el desarrollo de software destinado al diseño y la verificación de circuitos digitales. Esto es pertinente porque los autómatas modelan sistemas que operan en un número finito de estados diferentes.
- Análisis de patrones de texto (ej. correos electrónicos): Los autómatas finitos se emplean extensamente en software para analizar grandes volúmenes de texto en busca de patrones específicos, tales como palabras, estructuras o formatos predefinidos (por ejemplo, en páginas web o para encontrar direcciones de correo electrónico). También son útiles para comprobar la corrección de cualquier sistema que posea un número finito de estados distintos, como los protocolos de comunicación.

*Nota 2.1 .* El Marco Algebraico de los Autómatas. La investigación de las estructuras algebraicas sugeridas por el concepto de autómata permite desarrollar una teoría profunda. Estos modelos, incluidos los autómatas y las bases de datos, se manifiestan como estructuras algebraicas de múltiples tipos (*many-sorted algebraic structures*). El tratamiento de estas estructuras permite estudiar el comportamiento y la estructura de los autómatas reales.

## 2.1.2 Nociones Preliminares

Para comprender un autómata, se requiere un entendimiento previo de los lenguajes formales. Un lenguaje  $L$  sobre un alfabeto  $A$  se define formalmente como un subconjunto del conjunto de todas las cadenas posibles sobre  $A$ , denotado como  $A^*$ . El autómata tiene la función de aceptar las palabras  $u \in A^*$  que pertenecen a  $L$ .

### Ejemplo introductorio mediante un diagrama de transición

Un Autómata Finito Determinista (AFD)  $M = (Q, A, \delta, q_0, F)$  se define a través de un conjunto de estados ( $Q$ ), un alfabeto de entrada ( $A$ ), una función de transición ( $\delta$ ), un estado inicial ( $q_0$ ) y un conjunto de estados finales ( $F$ ). Un diagrama de transición proporciona una representación visual e intuitiva de esta estructura.

Consideremos un AFD  $M$  sobre el alfabeto  $A = \{0, 1\}$  cuyo lenguaje aceptado,  $L(M)$ , consiste en todas las palabras que contienen la subcadena **00**.

**Ejemplo 2.1 (AFD: Reconocimiento de Subcadena 00).** El autómata se construye con tres estados:  $q_0$  (inicial),  $q_1$  (se leyó un '0'), y  $q_2$  (se leyó '00' y es final).

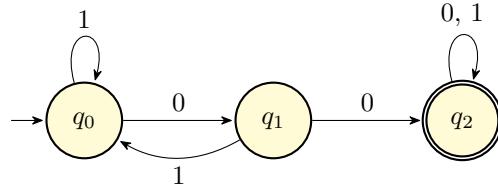


Figura 2.1: Autómata Finito Determinista que acepta el lenguaje de palabras que contienen la subcadena 00.

Proceso de Cálculo: La aceptación de una palabra  $u$  se define a través de una relación de cálculo ( $C_i \vdash C_{i+1}$ ), donde  $C_i = (q, v)$  es una configuración (estado  $q$ , cadena restante  $v$ ).

- Aceptación de **100**: El autómata comienza en  $(q_0, 100)$  y termina en un estado final  $q_2$  con la cadena vacía  $\varepsilon$ :

$$(q_0, 100) \vdash (q_0, 00) \vdash (q_1, 0) \vdash (q_2, \varepsilon)$$

Dado que  $q_2$  es un estado final, **100** es aceptada.

- Rechazo de **101**: El proceso termina en  $q_0$ , que no es un estado final, resultando en el rechazo:

$$(q_0, 101) \vdash (q_0, 01) \vdash (q_1, 1) \vdash (q_0, \varepsilon)$$

El lenguaje aceptado  $L(M)$  es el conjunto de todas las palabras  $u \in A^*$  tales que, partiendo del estado inicial  $q_0$  y consumiendo  $u$ , el autómata finaliza en algún estado  $q \in F$ . Para este ejemplo,  $L(M) = \{u_100u_2 \in \{0, 1\}^* \mid u_1, u_2 \in \{0, 1\}^*\}$ .

## 2.2 Autómata Finito Determinista (AFD)

### 2.2.1 Definición Formal

El Autómata Finito Determinista (AFD) sirve como el modelo fundamental más sencillo para estudiar la computabilidad y los lenguajes regulares. Su naturaleza determinista radica en que, dado un estado y un símbolo de entrada, el estado siguiente está únicamente definido.

**La quíntupla**  $M = (Q, A, \delta, q_0, F)$

Un AFD se define formalmente como una quíntupla:

$$M = (Q, A, \delta, q_0, F)$$

donde cada componente juega un papel crucial en la definición de la máquina de estados:

- $Q$ : Es el conjunto finito de estados del autómata.
- $A$ : Es el alfabeto o conjunto de entrada, que debe ser un conjunto finito de símbolos o letras.
- $\delta$ : Es la función de transición. Para un AFD, esta es una aplicación  $\delta : Q \times A \rightarrow Q$ . La función  $\delta$  especifica la transición de un estado a otro de manera única para cada par (estado, símbolo).
- $q_0$ : Es el estado inicial, un elemento distinguido  $q_0 \in Q$ .
- $F$ : Es el conjunto de estados finales o de aceptación, siendo un subconjunto de  $Q$ ,  $F \subseteq Q$ .

### 2.2.2 Representación

Los AFD pueden representarse de diferentes maneras, lo que facilita su análisis y comprensión, incluyendo la forma analítica, las tablas o los gráficos (*plots*).

#### Diagramas de Transición

Los diagramas de transición (o gráficos de estados) son la forma más intuitiva de representar un autómata. En esta representación, los estados se dibujan como círculos, las transiciones  $\delta(q, a) = p$  como flechas dirigidas desde  $q$  a  $p$  etiquetadas con  $a$ , el estado inicial se marca con una flecha entrante no etiquetada, y los estados finales se denotan con un doble círculo.

**Ejemplo 2.2 (Diagrama de Transición (AFD para subcadena 00)).** Utilizamos la representación de AFD para el lenguaje  $L$  de palabras que contienen la subcadena 00.

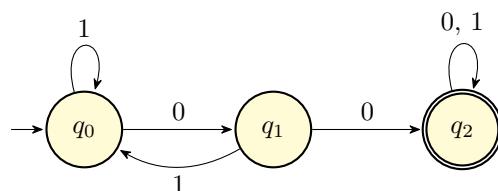


Figura 2.2: Diagrama de Transición para el AFD que acepta 00.

### Tablas de Transición

Las tablas de transición son una representación tabular y analítica de la función  $\delta$ , especialmente útil para la implementación algorítmica y la conversión entre tipos de autómatas.

**Ejemplo 2.3 (Tabla de Transición).** La tabla para el AFD de la Figura 2.2 es:

Estado	0	1
$\rightarrow q_0$	$q_1$	$q_0$
$q_1$	$q_2$	$q_0$
$*q_2$	$q_2$	$q_2$

Cuadro 2.1: *Tabla de transición del AFD para el lenguaje que contiene 00.*

### 2.2.3 Proceso de Cálculo

El proceso de cálculo de un AFD describe cómo evoluciona la máquina al procesar una palabra de entrada.

#### Configuración o Descripción Instantánea

Una configuración o descripción instantánea se define como un par  $(q, u) \in Q \times A^*$ . Representa el estado actual del cómputo, donde  $q$  es el estado en el que se encuentra el autómata y  $u$  es la porción restante de la palabra de entrada. La configuración inicial para una palabra  $u$  es siempre  $(q_0, u)$ .

#### Relación de un paso de cálculo ( $\vdash$ ) y de cálculo extendida ( $\vdash^*$ )

La relación de un paso de cálculo ( $\vdash$ ) describe una única transición en el autómata:

$$(q, au) \vdash (p, u) \iff \delta(q, a) = p$$

donde  $q, p \in Q$ ,  $a \in A$  (símbolo leído), y  $u \in A^*$  (cadena restante). Dado que el AFD es determinista, de una configuración se puede pasar a un máximo de una configuración en un solo paso de cálculo.

La relación de cálculo extendida ( $\vdash^*$ ) denota una secuencia finita de cero o más pasos de cálculo. La existencia de una sucesión de configuraciones  $C_0, \dots, C_n$  tal que  $C_0 \vdash C_1 \vdash \dots \vdash C_n$  se denota como  $(q, u) \vdash^* (p, v)$ .

#### Función de Transición Extendida ( $\delta^*$ )

Para simplificar la notación de trayectorias, se define la función de transición extendida  $\delta^* : Q \times A^* \rightarrow Q$ . Esta función devuelve el estado final al que se llega después de leer una palabra completa  $u$ , partiendo de un estado  $q$ .

**Definición 2.1 (Función de Transición Extendida  $\delta^*$ ).** La función  $\delta^*$  se define recursivamente como:

- 1) Base:  $\delta^*(q, \varepsilon) = q$ , para  $q \in Q$ .

- 2) Recursión:  $\delta^*(q, au) = \delta^*(\delta(q, a), u)$ , para  $q \in Q$ ,  $a \in A$ ,  $u \in A^*$ .

### 2.2.4 Aceptación de Palabras

La capacidad de un AFD de reconocer un patrón o estructura se define mediante el lenguaje que acepta.

#### *Definición de Lenguaje Aceptado $L(M)$ por un AFD*

Una palabra  $u \in A^*$  es aceptada si el autómata, comenzando en el estado inicial  $q_0$  y consumiendo toda la palabra, finaliza en un estado de aceptación  $q \in F$ . El conjunto de todas estas palabras aceptadas constituye el lenguaje  $L(M)$ .

El lenguaje  $L(M)$  puede expresarse formalmente de dos formas equivalentes:

- 1) Usando la relación de cálculo extendida:

$$L(M) = \{u \in A^* \mid \exists q \in F \text{ tal que } (q_0, u) \vdash^* (q, \varepsilon)\}$$

- 2) Usando la función de transición extendida:

$$L(M) = \{u \in A^* \mid \delta^*(q_0, u) \in F\}$$

Esta segunda expresión es inmediata y se utiliza frecuentemente en la teoría.

**Ejemplo 2.4 (Aplicación de  $\delta^*$ ).** Si consideramos el AFD del ejemplo anterior (AFD para subcadena 00), el cálculo de  $\delta^*$  para la palabra `modern100` es:

$$\delta^*(q_0, 100) = \delta^*(\delta(q_0, 1), 00) = \delta^*(q_0, 00) = \delta^*(\delta(q_0, 0), 0) = \delta^*(q_1, 0) = q_2$$

Como  $\delta^*(q_0, 100) = q_2$  y  $q_2 \in F$ , la palabra es aceptada.

*Nota 2.2 .* Propiedad de Clausura. Todos los lenguajes que son aceptados por un AFD son también aceptados por un Autómata Finito No Determinista (AFND), ya que todo AFD se considera un caso especial de AFND.

## 2.3 Tipos de Autómatas Finitos y Equivalencias

El estudio de los Autómatas Finitos (AF) requiere examinar extensiones del modelo determinista que facilitan la modelización y la composición. Los AFND y los AFN- $\epsilon$  son formalmente más potentes en su expresividad, aunque se demuestra que no reconocen una clase de lenguajes mayor que los AFD.

### 2.3.1 Autómata Finito No Determinista (AFND)

Mientras que un Autómata Finito Determinista (AFD) tiene una única transición definida para cada par (estado, símbolo), el Autómata Finito No Determinista (AFND) introduce la posibilidad de que, dada una entrada, la máquina pueda evolucionar a múltiples estados siguientes o a ninguno.

**Definición formal con función de transición a conjuntos de estados:**  $\delta : Q \times A \rightarrow \mathcal{P}(Q)$

Un AFND se define formalmente como una quíntupla  $M = (Q, A, \delta, q_0, F)$ , similar al AFD, pero con una diferencia crítica en su función de transición.

**Definición 2.2 (Autómata Finito No Determinista (AFND)).** Un AFND es una quíntupla  $M = (Q, A, \delta, q_0, F)$ , donde:

- $Q$  es el conjunto finito de estados.
- $A$  es el alfabeto de entrada.
- $\delta$  es la función de transición que mapea un par (estado, símbolo) a un conjunto de estados:

$$\delta : Q \times A \rightarrow \mathcal{P}(Q)$$

donde  $\mathcal{P}(Q)$  es el conjunto potencia de  $Q$  (todos los subconjuntos de  $Q$ ).

- $q_0 \in Q$  es el estado inicial.
- $F \subseteq Q$  es el conjunto de estados finales.

El no determinismo implica que la relación de cálculo  $((q, au) \vdash (p, u))$  se cumple si  $p \in \delta(q, a)$ , permitiendo que haya múltiples caminos de cómputo para una misma palabra de entrada, o incluso ninguno. Una palabra es aceptada si existe al menos un camino de cálculo que consume toda la palabra y termina en un estado de aceptación.

**Ejemplo 2.5 (AFND con Múltiples Transiciones).** Sea un AFND  $M$  sobre  $A = \{0, 1\}$  que acepta palabras que contienen la subcadena **001**. Observamos que desde  $q_0$ , al leer **0**, la transición no es única:

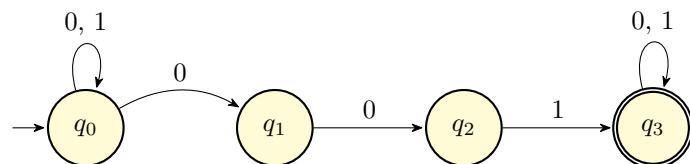


Figura 2.3: AFND para el lenguaje que contiene la subcadena 001. Desde  $q_0$  con el símbolo 0 se puede ir a  $q_0$  o a  $q_1$ , ilustrando la no determinismo.

### 2.3.2 AFND con Transiciones Nulas (AFN- $\epsilon$ )

Los AFND se extienden aún más al permitir movimientos entre estados que no consumen ningún símbolo de la cadena de entrada. Estos son los AFND con transiciones nulas o *épsilon* (AFN- $\epsilon$ ).

**Inclusión del símbolo de palabra vacía ( $\epsilon$ ) en el alfabeto para las transiciones**

Un AFN- $\epsilon$  es una quíntupla  $M = (Q, A, \delta, q_0, F)$  donde la función de transición  $\delta$  incluye la palabra vacía  $\epsilon$  en su dominio:

$$\delta : Q \times (A \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$$

Las transiciones  $\epsilon$  permiten que la descripción instantánea cambie de estado sin avanzar la posición de lectura en la cadena de entrada. Esto resulta extremadamente útil para construir autómatas que simulan operaciones de lenguajes formales, como la unión ( $r_1 + r_2$ ) o la clausura de Kleene ( $r_1^*$ ), ya

que se pueden conectar sub-autómatas utilizando estas transiciones sin afectar la semántica del lenguaje reconocido.

**Ejemplo 2.6 (Utilidad de  $\epsilon$  en la Unión).** Para modelar la unión de dos lenguajes,  $L_1$  y  $L_2$ , se puede crear un nuevo estado inicial  $q_0$  y añadir transiciones  $\epsilon$  a los estados iniciales de los autómatas de  $L_1$  y  $L_2$ .

### 2.3.3 Conversión AFND → AFD

A pesar de la mayor complejidad operativa de los AFND, la clase de lenguajes que aceptan es exactamente la misma que la de los AFD. Esta equivalencia se establece mediante un algoritmo constructivo.

#### *Presentación del Método de los Subconjuntos como algoritmo de equivalencia*

El algoritmo para convertir un AFND  $M$  en un AFD equivalente  $M'$  se basa en la idea de que cada estado en el AFD  $M'$  representa un conjunto de posibles estados en los que el AFND  $M$  podría encontrarse después de leer un prefijo de la entrada. Este procedimiento se conoce como el Método de los Subconjuntos.

Si  $M = (Q, A, \delta_N, q_0, F_N)$  es el AFND, el AFD equivalente resultante es  $M' = (Q', A, \delta_D, q'_0, F_D)$ , donde:

- $Q'$  es un subconjunto de  $\mathcal{P}(Q)$ . Los estados de  $M'$  son subconjuntos de estados de  $M$ .
- El estado inicial  $q'_0 = \{q_0\}$  (o  $\text{Cl}(q_0)$  si hubiera  $\epsilon$ -transiciones).
- Un estado  $P \in Q'$  es final en  $M'$  ( $P \in F_D$ ) si y solo si contiene al menos un estado final del AFND, es decir,  $P \cap F_N \neq \emptyset$ .
- La función de transición determinista  $\delta_D$  se define de la siguiente manera, para un estado  $P \subseteq Q$  y un símbolo  $a \in A$ :

$$\delta_D(P, a) = \bigcup_{q \in P} \delta_N(q, a)$$

El AFD  $M'$  se construye explorando progresivamente los nuevos conjuntos de estados alcanzables a partir de  $q'_0$ .

### 2.3.4 Conversión AFN- $\epsilon$ → AFND

La eliminación de las transiciones  $\epsilon$  es un paso crucial en la práctica, ya que simplifica el autómata y permite, en última instancia, la conversión al modelo AFD (a menudo, la conversión de AFN- $\epsilon$  a AFD se realiza en un solo paso, combinando este proceso con el método de los subconjuntos).

#### *Cálculo de la $\epsilon$ -Clausura para la eliminación de transiciones nulas*

Para eliminar las transiciones  $\epsilon$ , se debe definir la  $\epsilon$ -Clausura (Cl), que captura todos los estados alcanzables desde un estado dado utilizando únicamente transiciones nulas.

**Definición 2.3 ( $\epsilon$ -Clausura).** Dado un AFN- $\epsilon$   $M = (Q, A, \delta, q_0, F)$ :

- La  $\epsilon$ -Clausura de un estado  $q$ , denotada  $\text{Cl}(q)$ , es el conjunto de todos los estados  $p$  tales que existe un camino de cero o más pasos de  $\epsilon$ -transición desde  $q$  hasta  $p$ .

- La  $\epsilon$ -Clausura de un conjunto de estados  $P \subseteq Q$ , denotada  $\text{Cl}(P)$ , es la unión de las  $\epsilon$ -Clausuras de todos los estados en  $P$ :

$$\text{Cl}(P) = \bigcup_{q \in P} \text{Cl}(q)$$

La  $\epsilon$ -Clausura es esencial para definir la función de transición del autómata resultante  $M'$  (que será un AFND sin  $\epsilon$ -transiciones o un AFD). La nueva función de transición,  $\delta'$ , se calcula utilizando la  $\epsilon$ -Clausura en cada paso de lectura, asegurando que se contabilicen todos los movimientos nulos previos y posteriores a la lectura del símbolo.

Formalmente, si  $P$  es un conjunto de estados del AFN- $\epsilon$ , la transición a partir de un símbolo  $a$  en el autómata equivalente se calcula mediante la función  $\delta^*$  extendida:

$$\delta^*(P, a) = \text{Cl} \left( \bigcup_{q \in P} \delta(q, a) \right)$$

Esta  $\delta^*$  (aplicada al AFN- $\epsilon$ ) proporciona directamente el conjunto de estados en el AFD equivalente  $M'$ .

*Nota 2.3 . Teorema de Equivalencia.* La existencia de conversiones algorítmicas entre los tres modelos ( $\text{AFD} \leftrightarrow \text{AFND} \leftrightarrow \text{AFN-}\epsilon$ ) demuestra que todos aceptan exactamente la misma clase de lenguajes, conocida como la clase de Lenguajes Regulares. Esto es un resultado fundamental en la teoría de la computación.

## 2.4 Expresiones Regulares (ER) y El Teorema de Kleene

La clase de los lenguajes aceptados por Autómatas Finitos (AFD, AFND, AFN- $\epsilon$ ) se denomina Lenguajes Regulares ( $\mathcal{L}_3$ ). Una caracterización fundamental de esta clase es su representación mediante Expresiones Regulares (ER). La equivalencia entre estas dos notaciones constituye el célebre Teorema de Kleene.

### 2.4.1 Concepto y Operadores

#### *Definición de Expresión Regular*

Una Expresión Regular sobre un alfabeto  $A$  es una cadena de caracteres que describe un conjunto (lenguaje) de palabras mediante una definición recursiva.

**Definición 2.4 (Expresión Regular (ER)).** Sea  $A$  un alfabeto. Una Expresión Regular  $r$  sobre  $A$  es definida recursivamente como:

- 1) La expresión  $\emptyset$  (o  $/0$ ) es una ER que denota el lenguaje vacío,  $L(\emptyset) = \emptyset$ .
- 2) La expresión  $\epsilon$  es una ER que denota el lenguaje que contiene únicamente la palabra vacía,  $L(\epsilon) = \{\epsilon\}$ .
- 3) Para cada símbolo  $a \in A$ , la expresión  $a$  es una ER que denota el lenguaje que contiene únicamente la palabra  $a$ ,  $L(a) = \{a\}$ .
- 4) Si  $r_1$  y  $r_2$  son ERs, entonces la Unión ( $r_1 + r_2$ ), la Concatenación ( $r_1 r_2$ ) y la Clausura de Kleene

$(r_1^*)$  son también ERs.

### Operaciones fundamentales

Los operadores de las ERs corresponden directamente a operaciones sobre los lenguajes formales:

- Unión (o suma) (+): Si  $L_1 = L(r_1)$  y  $L_2 = L(r_2)$ , la unión de  $r_1$  y  $r_2$ , denotada  $r_1 + r_2$ , representa la unión de los lenguajes:  $L(r_1 + r_2) = L_1 \cup L_2$ . Es conmutativa y asociativa.
- Concatenación (yuxtaposición): La concatenación  $r_1 r_2$  denota el conjunto de palabras formadas al concatenar cualquier palabra de  $L_1$  con cualquier palabra de  $L_2$ :  $L(r_1 r_2) = \{uv \mid u \in L_1, v \in L_2\}$ . La concatenación es asociativa y  $\epsilon$  es su elemento neutro ( $r\epsilon = r$ ).
- Clausura de Kleene (\*): La clausura de Kleene de una expresión  $r$ , denotada  $r^*$ , representa la iteración del lenguaje asociado  $L(r)$ .

$$L(r)^* = \bigcup_{i \geq 0} L(r)^i$$

donde  $L(r)^0 = \{\epsilon\}$ . Esta operación garantiza la aceptación de cero o más repeticiones de patrones definidos por  $r$ .

**Ejemplo 2.7 (Expresiones Regulares).** Sea  $A = \{0, 1\}$ .

- 1) La expresión  $(0+1)^*$  denota todas las palabras posibles sobre  $A$ ,  $A^*$ .
- 2) La expresión  $1*(01*01*)^*1^*$  denota el lenguaje de palabras donde el número de ceros es par, incluyendo la palabra vacía y palabras compuestas únicamente de unos. (Nota: La versión simplificada en la fuente es  $1*(01*01*)^*$ , que acepta palabras que inician con 1s y tienen un número par de 0s, con 1s intercalados, y debe cerrarse con  $1^*$  o ser ajustada para la definición completa del lenguaje).
- 3) La expresión  $(0+1)^*0110(0+1)^*$  denota el lenguaje de palabras que contienen la subcadena 0110.

### 2.4.2 Conversión Autómata Finito → Expresión Regular (Teorema de Kleene, Parte I)

La primera parte del Teorema de Kleene afirma que, si un lenguaje  $L$  es aceptado por un Autómata Finito Determinista (AFD), entonces  $L$  puede ser expresado mediante una Expresión Regular. El método constructivo para obtener la ER asociada a un AFD  $M = (Q, A, \delta, q_1, F)$  se basa en la eliminación de estados de forma recursiva.

Asumimos que los estados de  $M$  están numerados  $Q = \{q_1, q_2, \dots, q_n\}$ .

#### Definición del conjunto $R_{ij}^k$

El cálculo se articula alrededor de la definición de los conjuntos  $R_{ij}^k$ , donde  $k$  representa un límite superior para los índices de los estados intermedios permitidos en un camino.

**Definición 2.5 (Conjunto de Palabras  $R_{ij}^k$ ).**  $R_{ij}^k$  es el conjunto de todas las palabras  $u \in A^*$  tales que, si el autómata comienza en el estado  $q_i$  y finaliza en el estado  $q_j$  después de leer  $u$ , todos los estados intermedios por los que pasa el autómata (al leer cualquier prefijo propio de  $u$ ) tienen

un índice (numeración) menor o igual a  $k$ .

El valor de  $k$  varía desde 0 hasta  $n$ , donde  $n = |Q|$ .

### **Algoritmo recursivo y fórmula de cálculo**

Para calcular  $R_{ij}^k$ , se divide el conjunto de caminos de  $q_i$  a  $q_j$  (con estados intermedios  $\leq k$ ) en dos categorías:

- 1) Caminos que no pasan por el estado  $q_k$ . Estos caminos ya estaban incluidos en  $R_{ij}^{k-1}$ .
- 2) Caminos que pasan por  $q_k$  una o más veces.

Los caminos del tipo 2 se descomponen en cuatro partes, usando el estado  $q_k$  como punto de bifurcación:

- 1) Un camino de  $q_i$  a  $q_k$  que solo usa estados intermedios  $\leq k-1$  ( $R_{ik}^{k-1}$ ).
- 2) Cero o más bucles en  $q_k$  (de  $q_k$  a  $q_k$ ) que solo usan estados intermedios  $\leq k-1$  ( $R_{kk}^{k-1}$ ) $^*$ .
- 3) Un camino final de  $q_k$  a  $q_j$  que solo usa estados intermedios  $\leq k-1$  ( $R_{kj}^{k-1}$ ).

La fórmula de recurrencia para el conjunto  $R_{ij}^k$  es:

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

De esta definición se deriva directamente la fórmula para las expresiones regulares asociadas  $r_{ij}^k$ :

$$r_{ij}^k = r_{ij}^{k-1} + r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1} \quad (2.1)$$

La base de la recursión,  $r_{ij}^0$ , se calcula directamente a partir de las transiciones directas entre  $q_i$  y  $q_j$ :  $r_{ij}^0 = \sum\{a \in A \mid \delta(q_i, a) = q_j\}$ . Si  $i = j$ , se incluye  $\epsilon$  en la suma.

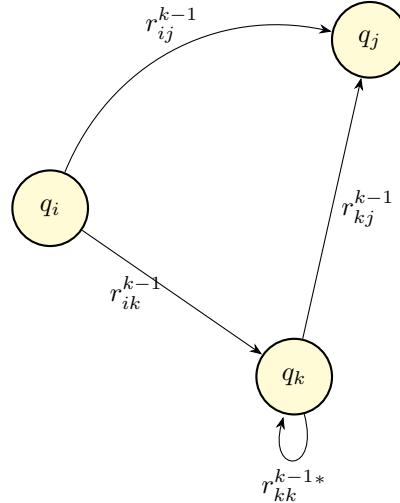


Figura 2.4: Representación esquemática de la fórmula de recurrencia  $r_{ij}^k$ .

### **Obtención de la Expresión Regular final para $L(M)$**

Una vez que la recursión alcanza  $k = n$  (donde  $n$  es el número total de estados),  $R_{1j}^n$  (asumiendo que  $q_1$  es el estado inicial) representa el conjunto de todas las palabras que llevan de  $q_1$  a  $q_j$  sin

restricciones en los estados intermedios.

El lenguaje  $L(M)$  es la unión de todos los lenguajes  $R_{1j}^n$  para cada estado final  $q_j \in F$ .

Si  $F = \{q_{j_1}, q_{j_2}, \dots, q_{j_m}\}$ , la expresión regular final  $r_{L(M)}$  es:

$$r_{L(M)} = r_{1j_1}^n + r_{1j_2}^n + \dots + r_{1j_m}^n$$

*Nota 2.4 . Lema de Arden.* Un método alternativo, a menudo más práctico para AFD pequeños, es el uso del Lema de Arden, que resuelve sistemas de ecuaciones lineales de expresiones regulares de la forma  $q_i = \sum_j a_j q_j + \epsilon$ . La solución para una ecuación de la forma  $R = RQ + P$  (donde  $R, Q, P$  son ERs y  $\epsilon \notin L(Q)$ ) es  $R = PQ^*$ .

### 2.4.3 Conversión Expresión Regular → Autómata Finito (Teorema de Kleene, Parte II)

La segunda parte del Teorema de Kleene establece que, dado cualquier lenguaje definido por una Expresión Regular  $r$ , existe un Autómata Finito que lo acepta. Esta transformación es esencial, ya que permite que los lenguajes, a menudo especificados inicialmente por su patrón (ER), sean implementados y reconocidos por un algoritmo (Autómata Finito).

La demostración de esta parte se realiza mediante una construcción recursiva que genera un Autómata Finito No Determinista con Transiciones Nulas (AFN- $\epsilon$ ) para cada operador.

#### Breve mención a la Construcción de Thompson

El algoritmo constructivo más conocido para esta conversión es la Construcción de Thompson. Este método establece reglas para construir un AFN- $\epsilon$  con un único estado inicial y un único estado final para cada caso base (símbolo  $a$ ,  $\epsilon$ ,  $\emptyset$ ) y para cada operación fundamental (unión, concatenación, clausura de Kleene).

Por ejemplo, la construcción para la unión de dos expresiones  $r_1$  y  $r_2$  (representadas por autómatas  $M_1$  y  $M_2$ ) se logra introduciendo un nuevo estado inicial,  $q_0$ , y añadiendo transiciones  $\epsilon$  desde  $q_0$  a los estados iniciales de  $M_1$  y  $M_2$ , resultando en un autómata compuesto que acepta la unión  $L(r_1) \cup L(r_2)$ .

La existencia de esta construcción recursiva es suficiente para demostrar la potencia equivalente de los tres modelos (AFD, AFND, ER) en la aceptación de la clase de Lenguajes Regulares.

## 2.5 Gramáticas Regulares (GR)

Las Gramáticas Regulares (GR), también conocidas como gramáticas de Tipo 3, representan la clase de generadores de lenguajes más restringida en la Jerarquía de Chomsky. Su importancia radica en que definen formalmente la clase de los Lenguajes Regulares, estableciendo la equivalencia generativa con los Autómatas Finitos.

### 2.5.1 Definición

Una Gramática Generativa se define formalmente como una cuádrupla  $G = (V, T, P, S)$ , donde  $V$  son las variables (símbolos no terminales),  $T$  son los símbolos terminales,  $P$  es el conjunto finito de reglas de producción, y  $S$  es el símbolo de partida.

Las Gramáticas Regulares imponen estrictas restricciones en la forma de las reglas de producción  $P$ .

**Definición 2.6 (Gramática de Tipo 3 o Regular).** Una Gramática  $G = (V, T, P, S)$  es de Tipo 3 (o Regular) si todas sus reglas de producción cumplen la forma Lineal por la Derecha o la forma Lineal por la Izquierda.

### *Gramáticas Lineales por la Derecha*

En una Gramática Lineal por la Derecha (GLD), el símbolo no terminal (variable) aparece únicamente como el último símbolo en la parte derecha de la producción, si es que aparece.

**Definición 2.7 (Gramática Lineal por la Derecha (GLD)).** Todas las reglas de producción  $P$  deben tener una de las siguientes dos formas:

- 1)  $A \rightarrow uB$
- 2)  $A \rightarrow u$

donde  $A, B \in V$  (variables) y  $u \in T^*$  (una cadena de cero o más símbolos terminales).

**Ejemplo 2.8 (GLD).** La gramática  $G$  con reglas  $S \rightarrow 0A$ ,  $A \rightarrow 10A$ ,  $A \rightarrow \varepsilon$  es una GLD.

- $S \rightarrow 0A$ :  $u = 0$ ,  $B = A$ .
- $A \rightarrow 10A$ :  $u = 10$ ,  $B = A$ .
- $A \rightarrow \varepsilon$ :  $u = \varepsilon$  (la palabra vacía).

Esta gramática genera el lenguaje  $L = \{0(10)^*\}$ .

### *Gramáticas Lineales por la Izquierda*

En una Gramática Lineal por la Izquierda (GLI), el símbolo no terminal (variable) aparece únicamente como el primer símbolo en la parte derecha de la producción, si es que aparece.

**Definición 2.8 (Gramática Lineal por la Izquierda (GLI)).** Todas las reglas de producción  $P$  deben tener una de las siguientes dos formas:

- 1)  $A \rightarrow Bu$
- 2)  $A \rightarrow u$

donde  $A, B \in V$  (variables) y  $u \in T^*$  (una cadena de cero o más símbolos terminales).

**Ejemplo 2.9 (GLI).** La gramática  $G$  con reglas  $S \rightarrow S10$  y  $S \rightarrow 0$  es una GLI.

- $S \rightarrow S10$ :  $B = S$ ,  $u = 10$ .
- $S \rightarrow 0$ :  $u = 0$ .

Esta gramática genera el lenguaje  $L = \{0(10)^*\}$ , el mismo lenguaje que el ejemplo de la GLD.

*Nota 2.5 . Restricción Crucial.* Una gramática que contiene una mezcla de reglas lineales por la derecha y reglas lineales por la izquierda (por ejemplo,  $A \rightarrow aB$  y  $C \rightarrow DC$ ) no es una gramática regular, a menos que el lenguaje generado sea trivial.

## 2.5.2 Equivalencia

### *Relación entre Autómatas Finitos y Gramáticas Regulares*

La relación entre los Autómatas Finitos (AF) y las Gramáticas Regulares (GR) es de equivalencia total: cualquier lenguaje reconocido por un AF puede ser generado por una GR, y viceversa.

**Teorema 2.1 (Equivalencia AF  $\leftrightarrow$  GR).** Un lenguaje  $L$  es aceptado por un Autómata Finito Determinista (AFD) si y solo si  $L$  es generado por una Gramática Regular.

Las construcciones algorítmicas que prueban esta equivalencia son directas y sistemáticas:

- 1) AFD  $\rightarrow$  GR (Lineal por la Derecha): Dada la definición de un AFD  $M = (Q, A, \delta, q_0, F)$ , se construye una GLD  $G = (Q, A, P, q_0)$  donde los estados  $Q$  actúan como variables.
  - Por cada transición  $\delta(p, a) = q$  en  $M$ , se crea la producción  $p \rightarrow aq$  en  $G$ .
  - Por cada estado final  $p \in F$ , se añade la producción  $p \rightarrow \varepsilon$ .
- 2) GR (Lineal por la Derecha)  $\rightarrow$  AFND: Dada una GLD  $G$ , se puede construir directamente un Autómata Finito No Determinista (AFND) o un AFND con  $\epsilon$ -transiciones que acepte exactamente el mismo lenguaje. Los estados del autómata se basan en las variables y los prefijos de la parte derecha de las reglas.
- 3) GLD  $\leftrightarrow$  GLI: Se demuestra que los lenguajes generados por GLD y GLI son idénticos. La conversión se realiza mediante la inversión de los autómatas asociados. Un lenguaje  $L$  generado por una GLI se relaciona con el AFD que acepta  $L^{-1}$  (el lenguaje inverso).

## 2.5.3 Conclusión

### *El Teorema de Kleene como la unión de modelos equivalentes*

El Teorema de Kleene, junto con la demostración de la equivalencia con las Gramáticas Regulares, establece un pilar fundamental en la Teoría de la Computación, unificando cuatro formalismos distintos capaces de describir la misma clase de lenguajes.

El conjunto de modelos equivalentes que definen esta clase es:

$$\text{AFD} \leftrightarrow \text{AFND} \leftrightarrow \text{ER} \leftrightarrow \text{GR}$$

Cualquier problema que pueda resolverse mediante uno de estos modelos tiene una solución equivalente en los otros tres. Por ejemplo, si se puede especificar un patrón de texto con una Expresión Regular (ER), se puede construir un AFD que lo reconozca y una Gramática Regular que lo genere.

### *Definición de la clase de los Lenguajes Regulares*

La clase de los Lenguajes Regulares, denotada como  $\mathcal{L}_3$  (Lenguajes de Tipo 3 en la Jerarquía de Chomsky), se define como el conjunto de todos los lenguajes que pueden ser descritos por cualquiera de los formalismos equivalentes discutidos.

**Definición 2.9 (Clase de los Lenguajes Regulares ( $\mathcal{L}_3$ )).** La clase  $\mathcal{L}_3$  es el conjunto de lenguajes que cumplen cualquiera de las siguientes condiciones equivalentes:

- Son aceptados por un Autómata Finito Determinista (AFD).
- Son aceptados por un Autómata Finito No Determinista (AFND o AFN- $\epsilon$ ).
- Son descritos por una Expresión Regular (ER).
- Son generados por una Gramática Regular (GLD o GLI).

La clase  $\mathcal{L}_3$  está contenida estrictamente en la clase de los Lenguajes Independientes del Contexto ( $\mathcal{L}_2$ ), es decir,  $\mathcal{L}_3 \subseteq \mathcal{L}_2$ . Un lenguaje no puede ser regular si, por ejemplo, requiere contar dos cantidades no adyacentes de forma independiente (como  $L = \{a^n b^n \mid n \geq 0\}$ ), ya que esto excede la capacidad de memoria finita de un autómata regular. Aunque esto se haya visto en la unidad anterior, es importante reiterarlo aquí.

## Capítulo 3

---

# Relaciones de Ejercicios

---

## 3.1 Lenguajes y Gramáticas

**Ejercicio 3.1.1.** Descripción de lenguajes generados por gramáticas.

- a) Describir el lenguaje generado por la siguiente gramática:

$$S \rightarrow XYX$$

$$X \rightarrow aX \mid bX \mid \epsilon$$

$$Y \rightarrow bbb$$

*Solución 3.1.1* (Ejercicio 1.a). El lenguaje generado por la gramática está compuesto por cadenas que tienen la forma:

- 1) Una secuencia de cero o más a o b (generada por X).
- 2) Seguido por bbb (generado por Y).
- 3) Seguido nuevamente por una secuencia de cero o más a o b (generada por X).

Por lo tanto, el lenguaje generado es:

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

Donde  $w_1$  y  $w_2$  son cadenas arbitrarias (incluyendo la cadena vacía) formadas por los símbolos a y b. Otra forma es demostrándolo mediante doble inclusión (manera más matemática).

- b) Describir el lenguaje generado por la siguiente gramática:

$$S \rightarrow aX$$

$$X \rightarrow aX \mid bX \mid \epsilon$$

*Solución 3.1.1* (Ejercicio 1.b). El lenguaje generado por la gramática está compuesto por cadenas que tienen la forma:

- 1) Una a inicial (generada por  $S$ ).
- 2) Seguido por una secuencia de cero o más a o b (generada por  $X$ ).

Por lo tanto, el lenguaje generado es:

$$L = \{a \ w \mid w \in \{a, b\}^*\} \text{ ó } L = \{v \in \{a, b\}^* : bbb \in v\}$$

Donde  $w$  es una cadena arbitraria (incluyendo la cadena vacía) formada por los símbolos a y b. Otra forma de demostrarlo es mediante doble inclusión.

- c) Describir el lenguaje generado por la siguiente gramática:

$$S \rightarrow XaXaX$$

$$X \rightarrow aX \mid bX \mid \epsilon$$

*Solución 3.1.1 (Ejercicio 1.c).* El lenguaje generado por la gramática está compuesto por cadenas que tienen la forma:

- 1) Una secuencia de cero o más  $a$  o  $b$  (generada por  $X$ ).
- 2) Seguido por una  $a$ .
- 3) Seguido nuevamente por una secuencia de cero o más  $a$  o  $b$  (generada por  $X$ ).
- 4) Seguido por otra  $a$ .
- 5) Seguido nuevamente por una secuencia de cero o más  $a$  o  $b$  (generada por  $X$ ).

Por lo tanto, el lenguaje generado es:

$$L = \{w_1 a w_2 a w_3 \mid w_1, w_2, w_3 \in \{a, b\}^*\}$$

Donde  $w_1$ ,  $w_2$  y  $w_3$  son cadenas arbitrarias (incluyendo la cadena vacía) formadas por los símbolos  $a$  y  $b$ . Otra forma de demostrarlo es mediante doble inclusión.

- d) Describir el lenguaje generado por la siguiente gramática:

$$S \rightarrow SS \mid XaXaX \mid \epsilon$$

$$X \rightarrow bX \mid \epsilon$$

*Solución 3.1.1 (Ejercicio 1.d).* El lenguaje generado por la gramática está compuesto por cadenas que tienen las siguientes características:

- 1) La gramática permite generar la cadena vacía ( $\epsilon$ ).
- 2) También permite generar cadenas de la forma  $w_1 a w_2 a w_3$ , donde  $w_1$ ,  $w_2$ , y  $w_3$  son cadenas formadas únicamente por el símbolo  $b$  (generadas por  $X$ ).
- 3) Además, permite concatenar arbitrariamente las cadenas generadas en los puntos anteriores debido a la regla  $S \rightarrow SS$ .

Encontramos que el número de  $a$  es par.

Por lo tanto, el lenguaje generado es:

$$L = \{\epsilon\} \cup \{w_1 a w_2 a w_3 \mid w_1, w_2, w_3 \in \{b\}^*\} \cup \{uv \mid u, v \in L\}$$

Donde  $w_1$ ,  $w_2$ , y  $w_3$  son cadenas arbitrarias (incluyendo la cadena vacía) formadas por el símbolo  $b$ , y  $u, v$  son cadenas generadas por la gramática. Otra forma de demostrarlo es mediante doble inclusión.

Otra forma vista en clase es:

$$L_{aux} = \{vawax \mid v, w, x \in \{b^i \mid i \in \mathbb{N}\}\} \cup \{\epsilon\}$$

Demostración por doble inclusión:

- Primera inclusión ( $L \subseteq R$ ):

Sea  $w \in L$ . Según las reglas de la gramática,  $w$  puede ser:

- La cadena vacía ( $\epsilon$ ), que claramente pertenece a  $R$ .
- Una cadena de la forma  $w_1 a w_2 a w_3 a$ , donde  $w_1, w_2, w_3 \in \{b\}^*$ . Estas cadenas también pertenecen a  $R$  por definición.
- Una concatenación de cadenas en  $L$  (por la regla  $S \rightarrow SS$ ). Si  $u, v \in L$ , entonces

$uv \in R$  porque  $R$  es cerrado bajo concatenación.

Por lo tanto,  $w \in R$ , y se cumple que  $L \subseteq R$ .

- Segunda inclusión ( $R \subseteq L$ ):

Sea  $w \in R$ . Según la definición de  $R$ ,  $w$  puede ser:

- La cadena vacía ( $\epsilon$ ), que claramente puede ser generada por la gramática.
- Una cadena de la forma  $w_1 a w_2 a w_3 a$ , donde  $w_1, w_2, w_3 \in \{b\}^*$ . Estas cadenas pueden ser generadas por la regla  $S \rightarrow XaXaX$  y  $X \rightarrow bX \mid \epsilon$ .
- Una concatenación de cadenas en  $R$ . Si  $u, v \in R$ , entonces  $uv \in L$  porque la regla  $S \rightarrow SS$  permite concatenar cadenas generadas por la gramática.

Por lo tanto,  $w \in L$ , y se cumple que  $R \subseteq L$ .

Dado que  $L \subseteq R$  y  $R \subseteq L$ , se concluye que  $L = R$ .

**Ejercicio 3.1.2.** Determinar lenguajes.

- a) Dada la gramática  $G = (\{S, A\}, \{a, b\}, P, S)$  donde:

$$P = \{S \rightarrow abAS, abA \rightarrow baab, S \rightarrow a, A \rightarrow b\}$$

Determinar el lenguaje que genera.

*Solución 3.1.2 (Ejercicio 2.a).* Cada vez que aplicamos  $S \rightarrow abAS$  generamos un bloque  $abA$  adicional y dejamos un  $S$  al final para poder repetir la expansión. Tras  $m$  aplicaciones de  $S \rightarrow abAS$  obtenemos la forma  $(abA)^m S$ .

Cada bloque  $abA$  puede convertirse o bien en  $baab$  aplicando la regla  $abA \rightarrow baab$ , o bien en  $abb$  aplicando primero  $A \rightarrow b$  (porque  $abA \Rightarrow abb$ ). Finalmente  $S \rightarrow a$ . Por tanto, cada bloque se convierte en  $baab$  o en  $abb$  y al final queda una  $a$ .

De aquí se deduce la forma general de las cadenas generadas:

$$L(G) = \{xa \mid x \in \{baab, abb\}^*\},$$

es decir, en notación de expresiones regulares:

$$L(G) = (baab \mid abb)^* a.$$

Prueba formal (dos sentidos)

- 1)  $L(G) \subseteq (baab \mid abb)^* a$

Tras  $m$  aplicaciones de  $S \rightarrow abAS$  se tiene  $(abA)^m S$  (prueba por inducción sobre  $m$ : base  $m = 0$  trivial; paso: si  $S \Rightarrow (abA)^k S$  entonces aplicando  $S \rightarrow abAS$  al  $S$  final obtenemos  $(abA)^{k+1} S$ ).

Para cada uno de los  $m$  factores  $abA$  podemos aplicar  $abA \rightarrow baab$  (obteniendo  $baab$ ) o bien aplicar  $A \rightarrow b$  (obteniendo  $abb$ ). Por tanto, la parte antes de la última  $S$  es una concatenación de  $baab$  y  $abb$ .

Finalmente  $S \rightarrow a$ . Por tanto, toda cadena derivable tiene la forma (bloques  $baab$  o  $abb$ ) seguida de  $a$ .

- 2)  $(baab \mid abb)^* a \subseteq L(G)$

Sea  $w = b_1 b_2 \cdots b_m a$  con cada  $b_i \in \{baab, abb\}$ .

Expandimos  $S$   $m$  veces con  $S \rightarrow abAS$  para obtener  $(abA)^m S$ .

Para cada  $i$ : si  $b_i = baab$  aplicamos la regla  $abA \rightarrow baab$  sobre el  $i$ -ésimo factor; si  $b_i = abb$  aplicamos  $A \rightarrow b$  en ese factor (convirtiendo  $abA$  en  $abb$ ).

Finalmente aplicamos  $S \rightarrow a$ . Eso produce exactamente  $w$ . Por tanto, cualquier cadena del lado derecho puede derivarse.

- b) Sea la gramática  $G = (V, T, P, S)$  donde:

$$V = \{\langle \text{numero} \rangle, \langle \text{digito} \rangle\}$$

$$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$S = \langle \text{numero} \rangle$$

- $\langle \text{numero} \rangle \rightarrow \langle \text{numero} \rangle \langle \text{digito} \rangle$
- $\langle \text{numero} \rangle \rightarrow \langle \text{digito} \rangle$
- $\langle \text{digito} \rangle \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Determinar el lenguaje que genera.

*Solución 3.1.2 (Ejercicio 2.b).* El lenguaje generado por la gramática está compuesto por cadenas que tienen las siguientes características:

- 1) La gramática permite generar cadenas formadas por uno o más dígitos, ya que:
  - $\langle \text{numero} \rangle \rightarrow \langle \text{numero} \rangle \langle \text{digito} \rangle$  permite construir cadenas de longitud arbitraria añadiendo dígitos.
  - $\langle \text{numero} \rangle \rightarrow \langle \text{digito} \rangle$  permite terminar la construcción con un único dígito.
- 2) Cada dígito es uno de los símbolos terminales  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ , según la regla  $\langle \text{digito} \rangle \rightarrow 0 \mid 1 \mid \dots \mid 9$ .

Por lo tanto, el lenguaje generado es el conjunto de todas las cadenas no vacías de dígitos, es decir:

$$L = \{w \mid w \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^+\}.$$

En notación de expresiones regulares, el lenguaje puede escribirse como:

$$L = [0 - 9]^+.$$

Otra forma que quedaría mejor vista en clase es:

$$L = \{0^i n \mid i \in \mathbb{N} \cup \{0\}, n \in \mathbb{N} \cup \{0\}\} \text{ ó } L = \{au : u \in \mathbb{N} \mid a \in \{0^*\}\}$$

- c) Sea la gramática  $G = (\{A, S\}, \{a, b\}, S, P)$  donde las reglas de producción son:

- $S \rightarrow aS$
- $S \rightarrow aA$
- $A \rightarrow bA$
- $A \rightarrow b$

Determinar el lenguaje generado por la gramática.

*Solución 3.1.2 (Ejercicio 2.c).* El lenguaje generado por la gramática está compuesto por cadenas que tienen las siguientes características:

- 1) La gramática permite generar cadenas que comienzan con uno o más símbolos  $a$ , ya que:
  - $S \rightarrow aS$  permite añadir un número arbitrario de  $a$  al principio.
  - $S \rightarrow aA$  permite terminar la secuencia de  $a$  y pasar a generar  $b$ .
- 2) Despues de la secuencia de  $a$ , la gramática genera uno o más símbolos  $b$ , ya que:
  - $A \rightarrow bA$  permite añadir un número arbitrario de  $b$ .
  - $A \rightarrow b$  permite terminar la secuencia de  $b$ .

Por lo tanto, el lenguaje generado es el conjunto de todas las cadenas que consisten en una secuencia no vacía de  $a$  seguida de una secuencia no vacía de  $b$ , es decir:

$$L = \{a^n b^m \mid n \geq 1, m \geq 1\}.$$

En notación de expresiones regulares, el lenguaje puede escribirse como:

$$L = a^+ b^+.$$

**Ejercicio 3.1.3.** Gramáticas de tipo 2 y tipo 3.

- a) Encontrar una gramática de tipo 2 para el lenguaje de palabras en las que el número de  $b$  no es tres. Determinar si el lenguaje generado es de tipo 3.

*Solución 3.1.3. a.* Para ello nos sirve esta gramática:

$$\begin{aligned} S &\rightarrow aS \mid bT \mid \varepsilon \\ T &\rightarrow aT \mid bX \mid \varepsilon \\ X &\rightarrow aX \mid bY \\ Y &\rightarrow aY \mid bZ \\ Z &\rightarrow abz \mid \varepsilon \end{aligned}$$

Se puede afirmar que, como el lenguaje es regular, también es de tipo 3.

- b) Encontrar una gramática de tipo 2 para el lenguaje de palabras que tienen 2 ó 3  $b$ . Determinar si el lenguaje generado es de tipo 3.

*Solución 3.1.3. b.* En este caso se puede usar la gramática:

$$\begin{aligned} S &\rightarrow aS \mid bX \\ X &\rightarrow aX \mid bY \\ Y &\rightarrow aY \mid bZ \mid \varepsilon \\ Z &\rightarrow aZ \mid \varepsilon \end{aligned}$$

*Todas las de tipo 3 son de tipo 2.*

**Ejercicio 3.1.4.** Gramáticas para lenguajes específicos.

- a) Encontrar una gramática de tipo 2 para el lenguaje de palabras que no contienen la subcadena  $ab$ . Determinar si el lenguaje generado es de tipo 3.

*Solución 3.1.4. a.*

$$\begin{aligned} S &\rightarrow aA \mid bS \mid \varepsilon \\ A &\rightarrow aA \mid \varepsilon \end{aligned}$$

- b) Encontrar una gramática de tipo 2 para el lenguaje de palabras que no contienen la subcadena  $baa$ . Determinar si el lenguaje generado es de tipo 3.

*Solución 3.1.4. b.*

$$\begin{aligned} S &\rightarrow aS \mid bB\varepsilon \\ B &\rightarrow bB \mid abB \mid a \mid \varepsilon \end{aligned}$$

De manera análoga a los demás ejercicios al ser lenguajes regulares, podemos afirmar que es de tipo 3 y todas las de tipo 3 están incluidas en las de tipo 2.

**Ejercicio 3.1.5.** Lenguaje con más  $a$  que  $b$ . Encontrar una gramática libre de contexto que genere el lenguaje sobre el alfabeto  $\{a, b\}$  de las palabras que tienen más  $a$  que  $b$  (al menos una más).

*Solución 3.1.5.* Para generar el lenguaje de palabras sobre el alfabeto  $\{a, b\}$  que tienen más  $a$  que  $b$  (al menos una más), podemos usar la siguiente gramática libre de contexto:

$$\begin{aligned} S &\rightarrow XaX \\ X &\rightarrow aXb \mid bXa \mid XX \mid aX \mid \varepsilon \end{aligned}$$

De esta manera garantizamos que desde el inicio hay al menos una  $a$  extra.

**Ejercicio 3.1.6.** Gramáticas regulares o libres de contexto.

- a) Encontrar, si es posible, una gramática regular (o, si no es posible, una gramática libre de contexto) que genere el lenguaje  $L$  sobre el alfabeto  $\{a, b\}$  tal que  $u \in L$  si, y solamente si,  $u$  no contiene dos símbolos  $b$  consecutivos.

*Solución 3.1.6. a).* Una gramática válida sería:

$$\begin{aligned} S &\rightarrow aS \mid bB \mid \varepsilon \\ B &\rightarrow aS \mid \varepsilon \end{aligned}$$

- b) Encontrar, si es posible, una gramática regular (o, si no es posible, una gramática libre de contexto) que genere el lenguaje  $L$  sobre el alfabeto  $\{a, b\}$  tal que  $u \in L$  si, y solamente si,  $u$  contiene dos símbolos  $b$  consecutivos.

*Solución 3.1.6. b).* Una gramática válida sería:

$$\begin{aligned} S &\rightarrow aS \mid bS \mid bbX \\ X &\rightarrow aX \mid bX \mid \varepsilon \end{aligned}$$

**Ejercicio 3.1.7.** Propiedades de lenguajes.

- a) Encontrar, si es posible, una gramática regular (o, si no es posible, una gramática libre de contexto) que genere el lenguaje  $L$  sobre el alfabeto  $\{a, b\}$  tal que  $u \in L$  si, y solamente si,  $u$  contiene un número impar de símbolos  $a$ .

*Solución 3.1.7. a).* Una solución de tipo 3 sería:

$$\begin{aligned} S &\rightarrow aX \mid bY \\ X &\rightarrow B_1 \mid aY \mid \varepsilon \\ B_1 &\rightarrow bB_1 \mid X \\ Y &\rightarrow B_2 \mid aX \\ B_2 &\rightarrow bBa \mid Y \end{aligned}$$

- b) Encontrar, si es posible, una gramática regular (o, si no es posible, una gramática libre de contexto) que genere el lenguaje  $L$  sobre el alfabeto  $\{a, b\}$  tal que  $u \in L$  si, y solamente si,  $u$  no contiene el mismo número de símbolos  $a$  que de símbolos  $b$ .

*Solución 3.1.7. b).* No se puede hacer de tipo 3, ya que hay que tener en cuenta números de  $a$  y  $b$  que no son finitos.

$$\begin{aligned} S &\rightarrow AaA \mid BbB \\ A &\rightarrow AaA \mid X \\ B &\rightarrow BbB \mid X \\ X &\rightarrow aXbX \mid bXaX \mid \varepsilon \end{aligned}$$

**Ejercicio 3.1.8.** Gramáticas para palabras con restricciones.

- a) Dado el alfabeto  $A = \{a, b\}$ , determinar si es posible encontrar una gramática libre de contexto que genere las palabras de longitud impar, y mayor o igual que 3, tales que la primera letra coincida con la letra central de la palabra.

*Solución 3.1.8. a).* Sí, es posible construir una gramática libre de contexto (Tipo 2) para el lenguaje  $L$  sobre el alfabeto  $A = \{a, b\}$ , que genera palabras  $w$  de longitud impar ( $\geq 3$ ) donde la primera letra coincide con la central.

$$\begin{aligned} S &\rightarrow aAb \mid bBa \mid aAa \mid bBb \\ A &\rightarrow a \mid aAb \mid bAa \mid aAa \mid bAb \\ B &\rightarrow b \mid aBb \mid bBa \mid aBa \mid bBb \end{aligned}$$

- b) Dado el alfabeto  $A = \{a, b\}$ , determinar si es posible encontrar una gramática libre de contexto que genere las palabras de longitud par, y mayor o igual que 2, tales que las dos letras centrales coincidan.

*Solución 3.1.8. b).* Sí, es posible encontrar una gramática libre de contexto (Tipo 2) que genere el lenguaje descrito.

$$\begin{aligned} S &\rightarrow ASA \mid B \\ A &\rightarrow a \mid b \\ B &\rightarrow bb \mid aa \end{aligned}$$

**Ejercicio 3.1.9.** Regularidad de un lenguaje. Determinar si el lenguaje generado por la gramática

$$\begin{aligned} S &\rightarrow SS \\ S &\rightarrow XXX \\ X &\rightarrow aX \mid Xa \mid b \end{aligned}$$

es regular. Justificar la respuesta.

*Solución 3.1.9.* Para justificalo vamos a servirnos de la siguiente gramática, definiendo el lenguaje:

$$\begin{aligned}
 S &\rightarrow aS \mid bS_1 \\
 S_1 &\rightarrow aS_1 \mid bS_2 \\
 S_2 &\rightarrow aS_2 \mid bS_3 \\
 S_3 &\rightarrow aS_3 \mid bS_1 \mid \varepsilon
 \end{aligned}$$

Siendo el lenguaje que genera:

$$L(G) = \{u \mid u \in \{a, b\}^* \mid Nb(u) = 3m, m \in \mathbb{N}^*\}$$

**Ejercicio 3.1.10.** Numerabilidad de un lenguaje. Dado un lenguaje  $L$  sobre un alfabeto  $A$ , ¿es  $L^*$  siempre numerable? ¿nunca lo es? ¿o puede serlo unas veces sí y otras, no? Proporcionar ejemplos en este último caso.

*Solución 3.1.10.* Para resolver esta cuestión, primero debemos recordar la definición de lenguaje y de conjunto numerable.

Un lenguaje  $L$  sobre un alfabeto  $A$  es, por definición, un subconjunto del conjunto de todas las palabras que se pueden formar sobre  $A$ , denotado como  $A^*$ . Es decir,  $L \subseteq A^*$ .

Un conjunto se considera numerable si existe una aplicación inyectiva de dicho conjunto en el conjunto de los números naturales.

Para cualquier alfabeto finito  $A$ , el conjunto  $A^*$  (el conjunto de todas las palabras posibles sobre  $A$ ) es siempre numerable. Esto se puede demostrar asignando un número único a cada palabra.

Ahora, analicemos  $L^*$ . La clausura de Kleene de un lenguaje  $L$ , denotada como  $L^*$ , se define como la unión de todas las potencias de  $L$ :

$$L^* = \bigcup_{i \geq 0} L^i = L^0 \cup L^1 \cup L^2 \cup \dots$$

donde  $L^0 = \{\epsilon\}$  y  $L^{i+1} = L^i L$ .

Dado que  $L$  es un lenguaje sobre el alfabeto  $A$ , todas las palabras en  $L$  están compuestas por símbolos de  $A$ . Por lo tanto, cualquier palabra formada por la concatenación de palabras de  $L$  (que es lo que constituye  $L^*$ ) también será una palabra sobre el alfabeto  $A$ . Esto significa que  $L^*$  es también un lenguaje sobre  $A$  y, por definición, es un subconjunto de  $A^*$ .

$$L \subseteq A^* \implies L^* \subseteq (A^*)^* = A^*$$

Como hemos establecido que  $A^*$  es siempre numerable, y cualquier subconjunto de un conjunto numerable es también numerable, podemos concluir que  $L^*$  es siempre numerable.

Por lo tanto, la respuesta a la pregunta es que  $L^*$  es siempre numerable.

**Ejercicio 3.1.11.** Propiedades de  $L^*$ . Dado un lenguaje  $L$  sobre un alfabeto  $A$ , caracterizar cuándo  $L^* = L$ . Es decir, dar un conjunto de propiedades sobre  $L$  de manera que  $L$  cumpla esas propiedades si y sólo si  $L^* = L$ .

*Solución 3.1.11.* Un lenguaje  $L$  sobre un alfabeto  $A$  es igual a su clausura de Kleene,  $L^*$ , si y solo

si cumple las dos propiedades siguientes:

- 1) La palabra vacía debe pertenecer al lenguaje:  $\epsilon \in L$ .
- 2) El lenguaje debe ser cerrado bajo la operación de concatenación: Para cualesquiera dos palabras  $u, v \in L$ , su concatenación  $uv$  también debe pertenecer a  $L$ .

Formalmente, la caracterización es:

$$L = L^* \iff (\epsilon \in L) \wedge (\forall u, v \in L \implies uv \in L)$$

#### Demostración

Demostrarímos la equivalencia en ambas direcciones.

( $\Rightarrow$ ) **Si  $L = L^*$ , entonces  $L$  cumple las dos propiedades.** Suponemos que  $L = L^*$ . Debemos probar que  $\epsilon \in L$  y que  $L$  es cerrado para la concatenación.

- 1) Por la definición de la clausura de Kleene,  $L^* = \bigcup_{i \geq 0} L^i = L^0 \cup L^1 \cup L^2 \cup \dots$ . El término  $L^0$  se define como el conjunto que contiene únicamente la palabra vacía, es decir,  $L^0 = \{\epsilon\}$ . Como  $\epsilon \in L^0$  y  $L^0 \subseteq L^*$ , se tiene que  $\epsilon \in L^*$ . Dado que hemos supuesto  $L = L^*$ , se concluye que  $\epsilon \in L$ .
- 2) Sean  $u, v$  dos palabras cualesquiera en  $L$ . Como  $L = L^*$ , entonces también se cumple que  $u, v \in L^*$ . Por definición de concatenación de lenguajes, la palabra  $uv$  pertenece al lenguaje  $L \cdot L = L^2$ . A su vez,  $L^2$  es un subconjunto de  $L^*$ . Por lo tanto,  $uv \in L^*$ . Y como  $L = L^*$ , concluimos que  $uv \in L$ .

Esto demuestra que si  $L = L^*$ , el lenguaje  $L$  contiene la palabra vacía y es cerrado bajo concatenación.

( $\Leftarrow$ ) **Si  $L$  contiene la palabra vacía y es cerrado bajo concatenación, entonces  $L = L^*$ .** Ahora suponemos que  $\epsilon \in L$  y que para todo  $u, v \in L$ , se cumple que  $uv \in L$ . Debemos demostrar que  $L = L^*$  mediante una doble inclusión.

- 1) Demostración de  $L \subseteq L^*$ : Esta inclusión es siempre cierta por la propia definición de la clausura de Kleene, ya que  $L = L^1$  y  $L^1 \subseteq \bigcup_{i \geq 0} L^i = L^*$ .
- 2) Demostración de  $L^* \subseteq L$ : Tomemos una palabra cualquiera  $w \in L^*$ .
  - Si  $w = \epsilon$ , entonces por la primera hipótesis ( $\epsilon \in L$ ), tenemos que  $w \in L$ .
  - Si  $w \neq \epsilon$ , por la definición de  $L^*$ , existe un número natural  $n \geq 1$  tal que  $w \in L^n$ . Esto significa que  $w$  se puede escribir como la concatenación de  $n$  palabras de  $L$ :  $w = a_1 a_2 \dots a_n$ , donde  $a_i \in L$  para todo  $i = 1, \dots, n$ .

Como  $L$  es cerrado bajo concatenación (segunda hipótesis), la concatenación de dos de sus elementos vuelve a estar en  $L$ . Aplicando esta propiedad de forma repetida, tenemos que  $a_1 a_2 \in L$ , luego  $(a_1 a_2) a_3 \in L$ , y así sucesivamente hasta concluir que  $a_1 a_2 \dots a_n = w \in L$ .

Como hemos demostrado ambas inclusiones ( $L \subseteq L^*$  y  $L^* \subseteq L$ ), se concluye que si  $L$  contiene a  $\epsilon$  y es cerrado bajo concatenación, entonces  $L^* = L$ .

**Ejercicio 3.1.12.** Igualdad de homomorfismos. Dados dos homomorfismos  $f : A^* \rightarrow B^*$ ,  $g : A^* \rightarrow B^*$ , se dice que son iguales si  $f(x) = g(x)$ ,  $\forall x \in A^*$ . ¿Existe un procedimiento algorítmico para comprobar si dos homomorfismos son iguales?

*Solución 3.1.12.* Sí, existe un procedimiento algorítmico para comprobar si dos homomorfismos son iguales.

Un homomorfismo  $h : A^* \rightarrow B^*$  queda completamente determinado por las imágenes de los símbolos del alfabeto  $A$ . Es decir, para cualquier palabra  $u = a_1 a_2 \dots a_n$ , su imagen es  $h(u) = h(a_1)h(a_2) \dots h(a_n)$ .

Por lo tanto, para determinar si dos homomorfismos  $f$  y  $g$  son iguales, basta con comprobar si sus imágenes coinciden para cada uno de los símbolos del alfabeto  $A$ . Dado que el alfabeto  $A$  es un conjunto finito por definición, este procedimiento consiste en un número finito de comparaciones y, por lo tanto, es algorítmico.

El algoritmo es el siguiente:

- Para cada símbolo  $a$  en el alfabeto  $A$ :
  - 1) Calcular  $f(a)$  y  $g(a)$ .
  - 2) Comparar si  $f(a) = g(a)$ .
  - 3) Si para algún  $a$  se encuentra que  $f(a) \neq g(a)$ , los homomorfismos son diferentes.
  - 4) Si se comprueba que  $f(a) = g(a)$  para todos los símbolos  $a \in A$ , entonces los homomorfismos son iguales.

**Ejercicio 3.1.13.** Lenguajes  $S_i$  y  $C_i$ . Sea  $L \subseteq A^*$  un lenguaje arbitrario. Sea  $C_0 = L$  y definamos los lenguajes  $S_i$  y  $C_i$ , para todo  $i \geq 1$ , por  $S_i = C_{i-1}^+$  y  $C_i = \overline{S_i}$ .

- a) ¿Es  $S_1$  siempre, nunca o a veces igual a  $C_2$ ? Justificar la respuesta.
- b) Demostrar que  $S_2 = C_3$ , cualquiera que sea  $L$ . (Pista: Demostrar que  $C_2$  es cerrado para la concatenación).

*Solución 3.1.13.* Este ejercicio requiere la aplicación de las operaciones de clausura positiva ( $L^+$ ) y complemento ( $\overline{L}$ ) a lenguajes.

Las definiciones dadas son: Sea  $L \subseteq A^*$  un lenguaje arbitrario.

- 1)  $C_0 = L$
- 2)  $S_i = C_{i-1}^+$  para  $i \geq 1$  .
- 3)  $C_i = \overline{S_i}$  para  $i \geq 1$  .

Recordamos que la clausura positiva  $L^+$  es la unión de todas las potencias de  $L$  con exponente mayor o igual a 1 ( $L^+ = \bigcup_{i \geq 1} L^i$ ), y que  $L^+$  es el conjunto de concatenaciones finitas de palabras en  $L$  excluyendo la palabra vacía  $\epsilon$ . Si  $\epsilon \in L$ , entonces  $L^+ = L^*$ .

- a) ¿Es  $S_1$  siempre, nunca o a veces igual a  $C_2$ ?

Para resolver esto, primero definimos  $S_1$  y  $C_2$ :

- 1) Cálculo de  $S_1$ :  $S_1 = C_0^+ = L^+$ .
- 2) Cálculo de  $C_2$ :

$$\begin{aligned} C_1 &= \overline{S_1} = \overline{L^+}, \\ S_2 &= C_1^+ = (\overline{L^+})^+, \\ C_2 &= \overline{S_2} = \overline{(\overline{L^+})^+}. \end{aligned}$$

Comparamos  $S_1 = L^+$  y  $C_2 = \overline{(\overline{L^+})^+}$ . La igualdad se cumplirá a veces, dependiendo de si el lenguaje  $P = \overline{L^+}$  genera el universo de palabras  $A^*$  o solo el lenguaje vacío bajo la operación de clausura positiva.

Casos de Ejemplo:

- Caso 1:  $S_1 = C_2$  (A veces) Sea  $A = \{a\}$  y sea  $L = \{a\}$ .
  - 1)  $S_1 = L^+ = \{a\}^+ = \{a^i \mid i \geq 1\}$ .

- 2)  $C_1 = \overline{S_1} = A^* \setminus \{a\}^+$ . Dado que  $A^* = \{a\}^* = \{\epsilon\} \cup \{a\}^+$ , tenemos  $C_1 = \{\epsilon\}$ .
- 3)  $S_2 = C_1^+ = \{\epsilon\}^+$ . Como  $\epsilon \in \{\epsilon\}$ , la clausura positiva es igual a la clausura de Kleene:  $S_2 = \{\epsilon\}^* = \{\epsilon\}$ .
- 4)  $C_2 = \overline{S_2} = \overline{\{\epsilon\}} = A^* \setminus \{\epsilon\} = \{a\}^+$ .

En este caso,  $\mathbf{S}_1 = \mathbf{C}_2$ , ya que ambos son iguales a  $\{a\}^+$ .

- Caso 2:  $S_1 \neq C_2$  (A veces) Sea  $A = \{a, b\}$  y sea  $L = \{ab\}$ .

- 1)  $S_1 = L^+ = \{(ab)^i \mid i \geq 1\}$ .
- 2)  $C_1 = \overline{L^+} = A^* \setminus L^+$ . Dado que  $A$  no está vacío,  $C_1$  contiene  $a, b, \epsilon, aa, bb$ , etc. Puesto que  $a \in C_1$  y  $b \in C_1$ , la clausura de Kleene de  $C_1$  genera todo  $A^*$ . Como  $\epsilon \in C_1$ , se cumple que  $C_1^+ = C_1^*$ . Por lo tanto,  $S_2 = C_1^+ = A^*$ .
- 3)  $C_2 = \overline{S_2} = \overline{A^*} = \emptyset$  (el lenguaje vacío).

En este caso,  $\mathbf{S}_1 \neq \mathbf{C}_2$ , ya que  $S_1 = \{(ab)^i \mid i \geq 1\} \neq \emptyset = C_2$ .

Conclusión:  $S_1$  es a veces igual a  $C_2$ .

- b) Demostrar que  $S_2 = C_3$ , cualquiera que sea  $L$ .

Demostración:

Para demostrar que  $S_2 = C_3$ , seguimos los siguientes pasos:

1. Definimos  $C_3$ :

$$C_3 = \overline{S_3}, \quad S_3 = C_2^+ \implies C_3 = \overline{C_2^+}.$$

2. Dado que  $C_2 = \overline{S_2}$ , si demostramos que  $C_2 = C_2^+$ , se sigue inmediatamente que:

$$C_3 = \overline{C_2^+} = \overline{C_2} = S_2.$$

3. Para que un lenguaje  $X$  sea igual a su clausura positiva ( $X = X^+$ ), se requiere que:

- $X$  sea cerrado bajo concatenación ( $XX \subseteq X$ ).
- $X$  no contenga la palabra vacía  $\epsilon$  ( $\epsilon \notin X$ ).

Paso I: Demostrar que  $C_2$  es cerrado bajo concatenación.

Definimos  $P = \overline{L^+}$  y  $C_2 = \overline{P^+}$ . Demostrar que  $C_2$  es cerrado bajo concatenación significa que si  $u \in C_2$  y  $v \in C_2$ , entonces  $uv \in C_2$ . Esto es equivalente a demostrar que si  $u \notin P^+$  y  $v \notin P^+$ , entonces  $uv \notin P^+$ .

Por definición,  $C_2$  es el conjunto de todas las palabras en  $A^*$  que no pueden ser generadas mediante la concatenación de una o más palabras del lenguaje  $P$ . Supongamos, por contradicción, que  $C_2$  no es cerrado bajo concatenación. Entonces, existen  $u, v \in C_2$  tales que  $uv \notin C_2$ . Si  $uv \notin C_2$ , entonces  $uv \in \overline{C_2} = S_2 = P^+$ . Esto implica que  $uv$  se puede descomponer como una concatenación de una o más palabras de  $P$ :  $uv = p_1 p_2 \dots p_k$ , donde  $p_i \in P$  y  $k \geq 1$ .

Dado que  $u \in C_2$ ,  $u$  no puede ser formado completamente por una subcadena inicial de  $p_1 p_2 \dots p_k$ . Esto contradice la hipótesis de que  $u \in C_2$ . Por lo tanto,  $C_2$  es cerrado bajo concatenación.

Paso II: Verificar que  $\epsilon \notin C_2$ .

Necesitamos determinar si  $\epsilon \in C_2 = \overline{S_2}$ , lo que es equivalente a determinar si  $\epsilon \notin S_2$ . Dado que  $S_2 = P^+$  y  $P = \overline{L^+}$ ,  $\epsilon \in P^+$  si y solo si  $\epsilon \in P$ . Esto implica:

$$\epsilon \in P \iff \epsilon \in \overline{L^+} \iff \epsilon \notin L^+.$$

- Si  $\epsilon \notin L$ , entonces  $\epsilon \notin L^+$ , lo que implica  $\epsilon \in P$ . Por lo tanto,  $\epsilon \in S_2$ , y  $\epsilon \notin C_2$ .
- Si  $\epsilon \in L$ , entonces  $\epsilon \in L^+$ , lo que implica  $\epsilon \notin P$ . Por lo tanto,  $\epsilon \notin S_2$ , y  $\epsilon \in C_2$ .

Conclusión:

Dado que  $C_2$  es cerrado bajo concatenación y cumple las condiciones necesarias, se sigue

que  $C_2 = C_2^+$ . Por lo tanto:

$$S_2 = \overline{C_2} = \overline{C_2^+} = C_3.$$

Esto demuestra que  $S_2 = C_3$ , cualquiera que sea  $L$ .

**Ejercicio 3.1.14.** Numerabilidad de lenguajes finitos. Demostrar que, para todo alfabeto  $A$ , el conjunto de los lenguajes finitos sobre dicho alfabeto es numerable.

*Solución 3.1.14.* Para demostrar que el conjunto de los lenguajes finitos sobre un alfabeto  $A$  es numerable, podemos establecer una correspondencia entre cada lenguaje finito y los números naturales.

- 1) El conjunto de todas las palabras  $A^*$  es numerable: Dado que un alfabeto  $A$  es finito, el conjunto de todas las palabras finitas que se pueden formar con sus símbolos,  $A^*$ , es numerable. Esto significa que podemos enumerar todas las palabras posibles:  $w_0, w_1, w_2, \dots$
- 2) Representación de lenguajes finitos: Un lenguaje finito es un subconjunto finito de  $A^*$ . Por lo tanto, cualquier lenguaje finito  $L$  puede representarse como un conjunto de palabras de esa enumeración, por ejemplo,  $\{w_{i_1}, w_{i_2}, \dots, w_{i_k}\}$ .
- 3) Construcción de una aplicación inyectiva: Podemos asignar un número natural único a cada lenguaje finito. Una manera de hacerlo es asociar a cada lenguaje finito  $L = \{w_{i_1}, w_{i_2}, \dots, w_{i_k}\}$  el número natural  $n = 2^{i_1} + 2^{i_2} + \dots + 2^{i_k}$ .
  - Por las propiedades de la representación binaria, cada número natural  $n$  corresponde a un único conjunto de exponentes, y por lo tanto, a un único lenguaje finito.
- 4) Conclusión de numerabilidad: Al existir una aplicación inyectiva del conjunto de los lenguajes finitos en el conjunto de los números naturales, se demuestra que el conjunto de todos los lenguajes finitos sobre  $A$  es numerable.

## 3.2 Cálculo de Gramáticas

**Ejercicio 3.2.1.** Calcula, de forma razonada, gramáticas que generen cada uno de los siguientes lenguajes:

– SENCILLOS

- $\{u \in \{0,1\}^* \text{ tales que } |u| \leq 4\}$
- Palabras con 0's y 1's que no contengan dos 1's consecutivos y que empiecen por un 1 y terminen por dos 0's.
- El conjunto vacío.
- El lenguaje formado por los números naturales.
- $\{a^n \in \{a,b\}^* \text{ con } n \geq 0\} \cup \{a^n b^n \in \{a,b\}^* \text{ con } n \geq 0\}$
- $\{a^n b^{2n} c^m \in \{a,b,c\}^* \text{ con } n, m > 0\}$
- $\{a^n b^m a^n \in \{a,b\}^* \text{ con } m, n \geq 0\}$
- Palabras con 0's y 1's que contengan la subcadena 00 y 11.
- Palíndromos formados con las letras a y b.

– DIFICULTAD MEDIA

- $\{uv \in \{0,1\}^* \text{ tales que } u^{-1} \text{ es un prefijo de } v\}$

*Solución 3.2.1. media.a.*

$$\begin{aligned} S &\rightarrow XY \\ X &\rightarrow 0X0 \mid 1X1 \mid \varepsilon \\ Y &\rightarrow 1Y \mid 0Y \mid \varepsilon \end{aligned}$$

Esta gramática no es de tipo 3, ya que las reglas de producción no cumplen con las restricciones de las gramáticas regulares, por ende, es de tipo 2.

- $\{ucv \in \{a,b,c\}^* \text{ tales que } u \text{ y } v \text{ tienen la misma longitud}\}$

*Solución 3.2.1. media.b.*

$$\begin{aligned} S &\rightarrow c \\ S &\rightarrow aSa \mid aSb \mid aSc \mid bSa \mid bSb \mid bSc \mid cSa \mid cSb \mid cSc \end{aligned}$$

- $\{u1^n \in \{0,1\}^* \text{ donde } |u| = n\}$

*Solución 3.2.1. media.c.*

$$S \rightarrow 1S1 \mid 0S0 \mid \varepsilon$$

Esta gramática genera cadenas de la forma  $u1^n$ , donde  $|u| = n$ , ya que cada vez que se añade un 1 a la derecha, se añade un símbolo correspondiente a  $u$  a la izquierda. La gramática no es de tipo 3, ya que las reglas de producción no cumplen con las restricciones de las gramáticas regulares, por lo que es de tipo 2.

- $\{a^n b^n a^{n+1} \in \{a,b\}^* \text{ con } n \geq 0\}$

*Solución 3.2.1. media.d.*

$$\begin{aligned} S &\rightarrow aSb \\ S &\rightarrow aA \\ A &\rightarrow aA \\ A &\rightarrow a \end{aligned}$$

Esta gramática genera cadenas de la forma  $a^n b^n a^{n+1}$ , donde  $n \geq 0$ . La primera regla  $S \rightarrow aSb$  asegura que el número de  $a$  y  $b$  en las primeras dos partes de la cadena sea igual. La regla  $S \rightarrow aA$  transfiere el control a  $A$ , que genera la parte final  $a^{n+1}$ . La gramática no es de tipo 3, ya que las reglas de producción no cumplen con las restricciones de las gramáticas regulares, por lo que es de tipo 2.

– DIFÍCILES

- a)  $\{a^n b^m c^k \text{ tales que } k = m + n\}$

*Solución 3.2.1. difícil.a.*

$$S \rightarrow aSbC \mid \varepsilon$$

$$C \rightarrow aC \mid bC \mid \varepsilon$$

Esta gramática genera cadenas de la forma  $a^n b^m c^k$  tales que  $k = m + n$ . La regla  $S \rightarrow aSbC$  asegura que por cada  $a$  y  $b$  generados, se genera un símbolo adicional en  $C$ . La regla  $C \rightarrow aC \mid bC \mid \varepsilon$  permite completar la parte final de la cadena con cualquier combinación de  $a$  y  $b$ .

La gramática no es de tipo 3, ya que las reglas de producción no cumplen con las restricciones de las gramáticas regulares, por lo que es de tipo 2.

- b) Palabras que son múltiplos de 7 en binario.

*Solución 3.2.1. difícil.b.* Para generar palabras que son múltiplos de 7 en binario, podemos construir una gramática que simule un autómata finito determinista (AFD) con 7 estados, donde cada estado representa el residuo de la división del número binario leído hasta el momento entre 7. El estado inicial es el residuo 0, y el estado final también es el residuo 0. Como residuo entendemos el resultado de realizar la operación del módulo 7.

- $S$ : residuo 0 (inicio y final)
- $A$ : residuo 1
- $B$ : residuo 2
- $C$ : residuo 3
- $D$ : residuo 4
- $E$ : residuo 5
- $F$ : residuo 6

$$S \rightarrow 0S \mid 1A \mid \varepsilon$$

$$A \rightarrow 0B \mid 1C$$

$$B \rightarrow 0D \mid 1E$$

$$C \rightarrow 0F \mid 1S$$

$$D \rightarrow 0A \mid 1B$$

$$E \rightarrow 0C \mid 1D$$

$$F \rightarrow 0E \mid 1F$$

En esta gramática:

- $S$  es el estado inicial (residuo 0).
- $A, B, C, D, E, F, G$  representan los estados correspondientes a los residuos 1, 2, 3, 4, 5, y 6, respectivamente.
- Las reglas de producción simulan las transiciones del AFD según el residuo actual

y el siguiente bit leído.

Esta gramática no es de tipo 3, ya que las reglas de producción no cumplen con las restricciones de las gramáticas regulares, por lo que es de tipo 2.

Para entenderlo mejor debemos de tener en cuenta cual es la ecuación que genera el siguiente estado:

Si el número actual (en decimal) es  $n$ , al leer un bit  $b \in \{0, 1\}$  el nuevo número es:

$$n' = 2n + b.$$

Por tanto, el nuevo residuo es:

$$r' = (2 \cdot r + b) \pmod{7},$$

donde  $r$  es el residuo actual ( $r = n \pmod{7}$ ).

Estado	Residuo r	Ejemplo (binario) n	al leer 0: $2n \rightarrow$ residuo	nueva letra	al leer 1: $2n + 1 \rightarrow$ residuo	nueva letra
S	0	$\varepsilon$ (vacía)	$2 \cdot 0 = 0 \pmod{7} = 0$	S	$2 \cdot 0 + 1 = 1 \pmod{7} = 1$	A
A	1	1	$2 \cdot 1 = 2 \pmod{7} = 2$	B	$2 \cdot 1 + 1 = 3 \pmod{7} = 3$	C
B	2	10	$2 \cdot 2 = 4 \pmod{7} = 4$	D	$2 \cdot 2 + 1 = 5 \pmod{7} = 5$	E
C	3	11	$2 \cdot 3 = 6 \pmod{7} = 6$	F	$2 \cdot 3 + 1 = 7 \pmod{7} = 0$	S
D	4	100	$2 \cdot 4 = 8 \pmod{7} = 1$	A	$2 \cdot 4 + 1 = 9 \pmod{7} = 2$	B
E	5	101	$2 \cdot 5 = 10 \pmod{7} = 3$	C	$2 \cdot 5 + 1 = 11 \pmod{7} = 4$	D
F	6	110	$2 \cdot 6 = 12 \pmod{7} = 5$	E	$2 \cdot 6 + 1 = 13 \pmod{7} = 6$	F

– EXTREMADAMENTE DIFÍCILES (no son libres de contexto)

- a)  $\{ww \text{ con } w \in \{0, 1\}^*\}$

*Solución 3.2.1. extremadamente difícil.a.* El lenguaje  $\{ww \mid w \in \{0, 1\}^*\}$  no es libre de contexto, ya que requiere comparar dos partes arbitrariamente largas de una cadena para verificar que son iguales. Esto se puede demostrar utilizando el lema de bombeo para lenguajes libres de contexto.

Sin embargo, podemos describir una gramática de tipo 0 (gramática general) que genera este lenguaje. Una posible gramática es:

$$\begin{aligned} S &\rightarrow X \# X \\ X &\rightarrow 0X0 \mid 1X1 \mid \epsilon \end{aligned}$$

En esta gramática:

- $S \rightarrow X \# X$  asegura que la cadena generada tiene la forma  $w \# w$ , donde  $w$  es una cadena de ceros y unos.
- $X \rightarrow 0X0$  y  $X \rightarrow 1X1$  generan cadenas de ceros y unos de forma recursiva, asegurando que las dos partes de la cadena son idénticas.
- $X \rightarrow \epsilon$  permite terminar la generación de la cadena.

El símbolo  $\#$  es un marcador que separa las dos partes de la cadena. Este lenguaje no es regular ni libre de contexto, pero puede ser generado por una gramática de tipo 0.

- b)  $\{a^{n^2} \in \{a\}^* \text{ con } n \geq 0\}$

*Solución 3.2.1. extremadamente difícil.b.* El lenguaje  $\{a^{n^2} \mid n \geq 0\}$  no es libre de contexto, ya que requiere contar el número de símbolos  $a$  y verificar que este número es un cuadrado perfecto. Esto no puede lograrse con una gramática libre de contexto.

Sin embargo, podemos describir una gramática de tipo 0 (gramática general) que genera este lenguaje. Una posible gramática es:

$$\begin{aligned} S &\rightarrow A\#A \\ A &\rightarrow aA \mid \epsilon \end{aligned}$$

Esta gramática utiliza un marcador  $\#$  para separar las partes de la cadena y asegura que el número total de  $a$  generados corresponde a un cuadrado perfecto. La gramática no es regular ni libre de contexto, pero puede ser generada por una gramática de tipo 0.

- c)  $\{a^p \in \{a\}^*\text{ con }p\text{ primo}\}$

*Solución 3.2.1. extremadamente difícil.c.* El lenguaje  $\{a^p \mid p \text{ es primo}\}$  no es libre de contexto ni regular, ya que requiere verificar que el número de símbolos  $a$  es un número primo, lo cual no puede lograrse con una gramática libre de contexto.

Sin embargo, podemos describir una gramática de tipo 0 (gramática general) que genera este lenguaje. Una posible gramática es:

$$\begin{aligned} S &\rightarrow aS \mid P \\ P &\rightarrow a^2 \mid a^3 \mid a^5 \mid a^7 \mid \dots \end{aligned}$$

En esta gramática:

- $S$  genera cadenas de longitud arbitraria.
- $P$  genera cadenas cuya longitud corresponde a un número primo.

La gramática no es regular ni libre de contexto, pero puede ser generada por una gramática de tipo 0. Podemos ver que es infinita, por ende podemos concluir con que aunque el lenguaje  $\{a^p \mid p \text{ primo}\}$  no es regular y por tanto no es libre de contexto, es decidable<sup>1</sup>, por lo que pertenece a la clase recursiva; existe por tanto alguna gramática tipo-0 o máquina de Turing<sup>2</sup> que lo reconozca, pero no hay una gramática regular ni libre de contexto que lo genere.

- d)  $\{a^n b^m \in \{a, b\}^* \text{ con } n \leq m^2\}$

*Solución 3.2.1. extremadamente difícil.d.* El lenguaje  $\{a^n b^m \mid n \leq m^2\}$  no es libre de contexto, ya que requiere comparar dos partes de una cadena y verificar que una es menor o igual al cuadrado de la otra. Esto no puede lograrse con una gramática libre de contexto.

Sin embargo, podemos describir una gramática de tipo 0 (gramática general) que genera este lenguaje. Una posible gramática es:

$$S \rightarrow aSbb \mid \epsilon$$

En esta gramática:

---

<sup>1</sup>Un lenguaje es decidable si existe un algoritmo que, dado cualquier elemento del alfabeto, puede determinar en un tiempo finito si pertenece o no al lenguaje.

<sup>2</sup>Una máquina de Turing es un modelo computacional abstracto que consiste en una cinta infinita dividida en celdas, un cabezal de lectura/escritura que se mueve sobre la cinta, y un conjunto de estados que determinan las transiciones basadas en el símbolo leído. Es capaz de simular cualquier algoritmo y se utiliza para definir formalmente la computabilidad.

- $S \rightarrow aSbb$  asegura que por cada símbolo  $a$  generado, se generan dos símbolos  $b$ , lo que garantiza que  $n \leq m^2$ .
- $S \rightarrow \epsilon$  permite terminar la generación de la cadena.

La gramática no es regular ni libre de contexto, pero puede ser generada por una gramática de tipo 0.

### 3.3 Autómatas Finitos

**Ejercicio 3.3.1.** Considera el siguiente Autómata Finito Determinista (AFD)  $M = (Q, A, \delta, q_0, F)$ , donde:

- $Q = \{q_0, q_1, q_2\}$ ,
- $A = \{0, 1\}$
- La función de transición viene dada por:

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

- $F = \{q_2\}$

Describir informalmente el lenguaje aceptado.

*Solución 3.3.1.* El lenguaje aceptado por el Autómata Finito Determinista (AFD)  $M$  es el conjunto de todas las palabras sobre el alfabeto  $A = \{0, 1\}$  que contienen la subcadena **00**. Este lenguaje se describe formalmente como  $L = \{u_100u_2 \mid u_1, u_2 \in \{0, 1\}^*\}$ .

1) Ruta de Aceptación:

- 1) El estado  $q_0$  es el estado inicial, y representa no haber encontrado aún la subcadena, o que el último símbolo leído no forma parte de una "0" potencial.
- 2) Al leer '0' desde  $q_0$ , se alcanza  $q_1$ , lo que significa que se ha leído el primer '0' de la secuencia buscada ( $\delta(q_0, 0) = q_1$ ).
- 3) Al leer otro '0' desde  $q_1$ , se alcanza el estado final  $q_2$ , confirmando la aparición de la subcadena "00" ( $\delta(q_1, 0) = q_2$ ).
- 4)  $q_2$  es un estado final absorbente (de trampa), ya que cualquier símbolo de entrada posterior lo mantiene en  $q_2$  ( $\delta(q_2, 0) = q_2, \delta(q_2, 1) = q_2$ ), garantizando que la palabra es aceptada una vez que se detecta "00".

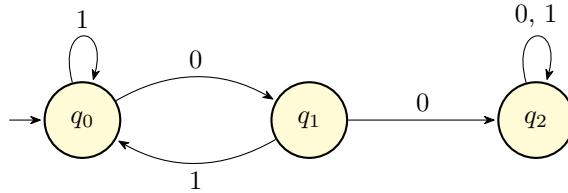


Figura 3.1: Diagrama de Transición del AFD del Ejercicio 1.

**Ejercicio 3.3.2.** Dado el AFD

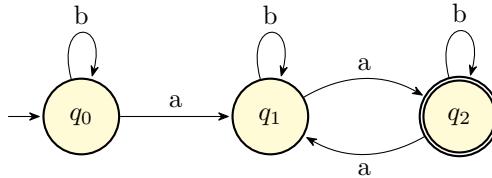


Figura 3.2: Diagrama de Transición para el Ejercicio 2.

describir el lenguaje aceptado por dicho autómata.

*Solución 3.3.2.* El lenguaje aceptado por el Autómata Finito Determinista (AFD) es el conjunto de todas las palabras sobre el alfabeto  $\{a, b\}$  que contienen un número **par** de ocurrencias del símbolo **a**, siendo dicho número mayor que cero.

Formalmente, el lenguaje  $L$  es:

$$L = \{u \in \{a, b\}^* \mid N_a(u) \text{ es par}, N_a(u) > 0\}$$

El autómata utiliza  $q_0$  para contar 0 'a's,  $q_1$  para contar un número impar de 'a's, y  $q_2$  (el estado final) para contar un número par y positivo de 'a's.

**Ejercicio 3.3.3.** Dibujar AFDs que acepten los siguientes lenguajes con alfabeto  $\{0, 1\}$ :

- 1) El lenguaje vacío,  $\emptyset$ .
- 2) El lenguaje formado por la palabra vacía, o sea,  $\{\varepsilon\}$ .
- 3) El lenguaje formado por la palabra 01, o sea,  $\{01\}$ .
- 4) El lenguaje  $\{11, 00\}$ .
- 5) El lenguaje  $\{(01)^i \mid i \geq 0\}$ .
- 6) El lenguaje formado por las cadenas con 0's y 1's donde el número de unos es divisible por 3.

*Solución 3.3.3.* A continuación, se dibujan los Autómatas Finitos Deterministas (AFD) que aceptan cada uno de los lenguajes dados sobre el alfabeto  $\Sigma = \{0, 1\}$ .

- 1) El lenguaje vacío,  $\emptyset$ .

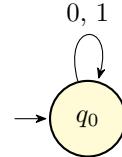


Figura 3.3: AFD para  $L = \emptyset$ .

- 2) El lenguaje formado por la palabra vacía,  $\{\varepsilon\}$ .

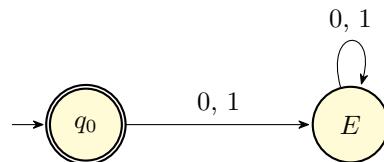


Figura 3.4: AFD para  $L = \{\varepsilon\}$ .

- 3) El lenguaje  $\{01\}$ .

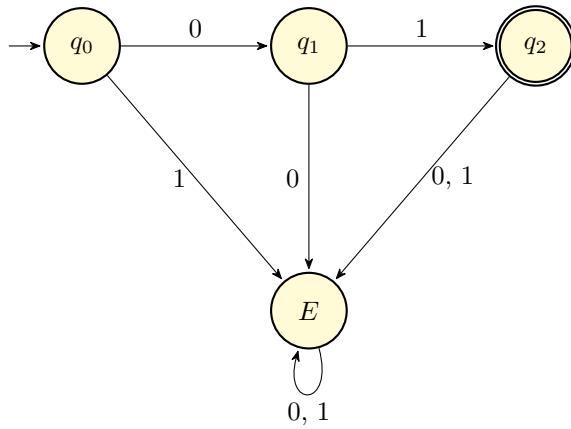
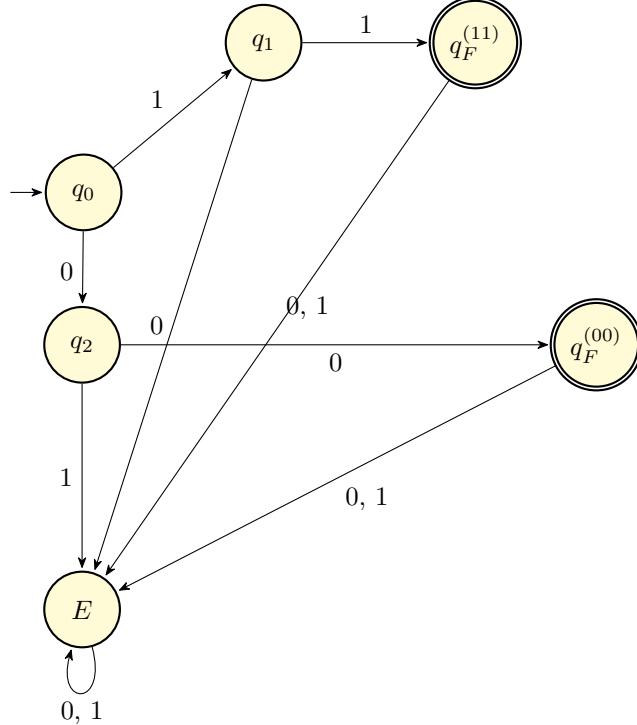
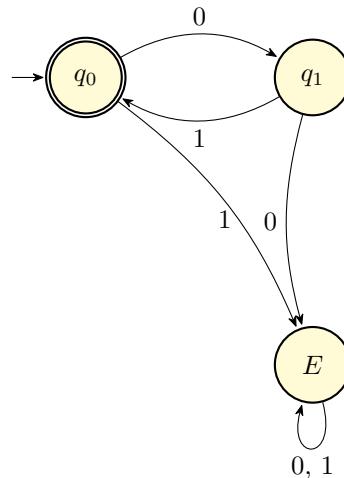
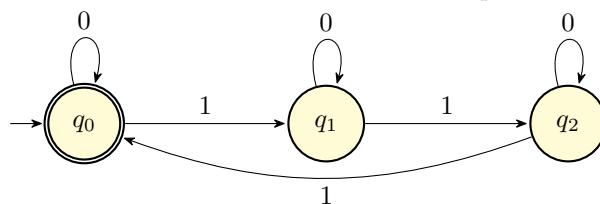


Figura 3.5: AFD para  $L = \{01\}$ .4) El lenguaje  $\{11, 00\}$ .Figura 3.6: AFD para  $L = \{11, 00\}$ .5) El lenguaje  $\{(01)^i \mid i \geq 0\}$ .Figura 3.7: AFD para  $L = \{(01)^i \mid i \geq 0\}$ .

6) El lenguaje de cadenas donde el número de unos es divisible por 3.

Figura 3.8: AFD para  $L = \{w \mid N_1(w) \equiv 0 \pmod{3}\}$ .

**Ejercicio 3.3.4.** Obtener a partir de la gramática regular  $G = (\{S, B\}, \{1, 0\}, P, S)$ , con  $P = \{S \rightarrow 110B, B \rightarrow 1B, B \rightarrow 0B, B \rightarrow \varepsilon\}$ , un AFND que reconozca el lenguaje generado por esa gramática.

*Solución 3.3.4.* Para la resolución de este ejercicio, seguiremos los pasos necesarios para construir un Autómata Finito No Determinista (AFND) a partir de la gramática regular dada.

1) Definición del Lenguaje Generado ( $L(G)$ ):

- La variable  $B$  tiene las producciones  $B \rightarrow 1B$  y  $B \rightarrow 0B$ , que permiten generar cualquier secuencia finita de 1's y 0's, y  $B \rightarrow \varepsilon$ , que permite terminar la derivación. Por lo tanto, el lenguaje generado por  $B$  es  $L(B) = \{1, 0\}^* = \Sigma^*$ .
- La producción inicial  $S \rightarrow 110B$  fuerza a que toda palabra comience con la subcadena 110.
- Así, el lenguaje generado es  $L(G) = \{110w \mid w \in \{1, 0\}^*\} = 110(1 + 0)^*$ .

2) Especificación del AFND:

- 1) Alfabeto ( $\Sigma$ ):  $\Sigma = \{1, 0\}$ , tomado de la gramática  $G$ .

2) Conjunto de Estados ( $Q$ ):

- Utilizaremos un conjunto de estados que incluya los no terminales de la gramática ( $S$  y  $B$ ) y los estados intermedios necesarios para procesar las cadenas terminales de longitud mayor que uno.
- Necesitamos estados intermedios para la producción  $S \rightarrow 110B$ .
- Denominaremos el estado inicial  $q_0$  (correspondiente a  $S$ ).

Definimos  $Q = \{q_0, q_1, q_2, q_3\}$ . Asociaremos  $q_0 \equiv S$  y  $q_3 \equiv B$ .

3) Estado Inicial ( $q_0$ ): El estado inicial es  $q_0$ , correspondiente al símbolo de partida  $S$ .

$$q_{inicial} = q_0.$$

4) Estados Finales ( $F$ ):

- Un estado  $A$  es final si existe una producción  $A \rightarrow u$ , donde  $u \in T^*$ . La producción  $B \rightarrow \varepsilon$  (donde  $u = \varepsilon$ ) indica que el estado  $B$  debe ser un estado final.

$$F = \{q_3\}.$$

3) Construcción de la Función de Transición ( $\delta$ ):

- Producción  $S \rightarrow 110B$  (Ruta principal):

$$\begin{aligned} S \rightarrow 110B &\implies \delta(q_0, 1) = \{q_1\} \\ &\delta(q_1, 1) = \{q_2\} \\ &\delta(q_2, 0) = \{q_3\} \end{aligned}$$

- Producciones  $B \rightarrow 1B$  y  $B \rightarrow 0B$  (Ciclo):

$$\begin{aligned} B \rightarrow 1B &\implies \delta(q_3, 1) = \{q_3\} \\ B \rightarrow 0B &\implies \delta(q_3, 0) = \{q_3\} \end{aligned}$$

- Producción  $B \rightarrow \varepsilon$  (Aceptación): Esta producción permite que la derivación termine en el estado  $B$  ( $q_3$ ), lo que confirma que  $q_3$  debe ser un estado de aceptación.

El AFND resultante es  $M = (\{q_0, q_1, q_2, q_3\}, \{1, 0\}, \delta, q_0, \{q_3\})$ , donde la función de transición

$\delta$  está definida por:

$$\delta = \begin{cases} (q_0, 1) \rightarrow \{q_1\} \\ (q_1, 1) \rightarrow \{q_2\} \\ (q_2, 0) \rightarrow \{q_3\} \\ (q_3, 0) \rightarrow \{q_3\} \\ (q_3, 1) \rightarrow \{q_3\} \\ \text{resto} \rightarrow \emptyset \end{cases}$$

- 4) Representación Gráfica del AFND:

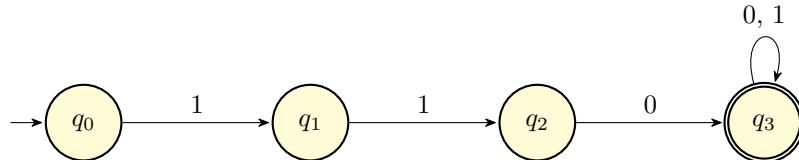


Figura 3.9: Autómata Finito No Determinista (AFND) que reconoce el lenguaje  $L = 110(1 + 0)^*$ , generado por la gramática  $G$ .

**Ejercicio 3.3.5.** Dada la gramática regular  $G = (\{S\}, \{1, 0\}, P, S)$ , con  $P = \{S \rightarrow S10, S \rightarrow 0\}$ , obtener un AFD que reconozca el lenguaje generado por esa gramática.

*Solución 3.3.5.* Para la resolución de este ejercicio se seguirán los siguientes pasos:

- 1) Identificación y Análisis de la Gramática

La gramática  $G = (\{S\}, \{1, 0\}, P, S)$  es una Gramática Lineal por la Izquierda, ya que todas las producciones son de la forma  $A \rightarrow Bw$  o  $A \rightarrow w$ , donde  $w \in T^*$  y  $B \in V$  (en este caso,  $S \rightarrow S10$  y  $S \rightarrow 0$ ).

- 2) Determinación del Lenguaje ( $L(G)$ ): La producción base  $S \rightarrow 0$  genera la palabra más corta, 0. La producción recursiva  $S \rightarrow S10$  permite pre-concatenar la secuencia 10 a cualquier palabra derivable desde  $S$ .

$$S \Rightarrow S10 \Rightarrow S1010 \Rightarrow \dots \Rightarrow S(10)^n \Rightarrow 0(10)^n, \quad n \geq 0$$

Por lo tanto, el lenguaje generado es  $L(G) = \{0(10)^n \mid n \geq 0\}$ .

- 3) Método de Conversión: Inversión de Lenguajes

Dado que la gramática es lineal por la izquierda, el método algorítmico más riguroso implica tres pasos:

- 1) Invertir las producciones para obtener una gramática lineal por la derecha ( $G'$ ).
- 2) Construir un AFND  $M'$  para  $L(G')$ .
- 3) Invertir  $M'$  para obtener un AFND que reconozca  $L(G)$ . Finalmente, se determiniza y se completa para obtener el AFD.

Por ello nos queda:

- 1) Inversión de la Gramática ( $G'$ )

Invertimos las cadenas terminales ( $\alpha$ ) en el lado derecho de las producciones  $A \rightarrow \alpha$  para obtener una gramática lineal por la derecha  $G'$ :

$$P' = \{S \rightarrow 01S, S \rightarrow 0\}$$

Esta nueva gramática  $G'$  genera el lenguaje inverso:  $L(G') = L(G)^{-1} = \{(01)^n 0 \mid$

$$n \geq 0\}.$$

**2) Construcción del AFND  $M'$  para  $L(G')$**

A partir de la gramática lineal por la derecha  $G'$ , construimos un AFND  $M' = (Q', \Sigma, \delta', q'_0, F')$ .

- Estados ( $Q'$ ): Se requieren estados para el símbolo de partida  $S$  y los símbolos intermedios de las producciones con  $|u| \geq 2$ . Definimos  $q_0 \equiv S$ . Necesitamos un estado intermedio ( $q_1$ ) y un estado final ( $q_2$ ).  $Q' = \{q_0, q_1, q_2\}$ .
- Estado Inicial ( $q'_0$ ):  $q'_0 = q_0$ .
- Estados Finales ( $F'$ ): La producción  $S \rightarrow 0$  implica que  $S$  puede derivar directamente a un terminal. Por lo tanto, necesitamos un estado final ( $q_2$ ) alcanzable desde  $S$  con 0.  $F' = \{q_2\}$ .

**3) Transiciones ( $\delta'$ ):**

- $S \rightarrow 0$ :  $\delta'(q_0, 0) = \{q_2\}$ .
- $S \rightarrow 01S$ : Se consumen 0 y 1 para volver a  $S$  ( $q_0$ ).

$$q_0 \xrightarrow{0} q_1 \quad y \quad q_1 \xrightarrow{1} q_0$$

**4) Inversión del Autómata ( $M''$  para  $L(G)$ )**

Para obtener un autómata  $M''$  que reconozca  $L(G) = L(G')^{-1}$ , se invierte  $M'$ : el estado inicial  $q'_0$  se convierte en el único estado final  $F'' = \{q_0\}$ , los estados finales  $F' = \{q_2\}$  se convierten en el (los) estado(s) inicial(es), y todas las transiciones se invierten.

- Estados:  $Q'' = \{q_0, q_1, q_2\}$ .
- Estado Inicial ( $q''_0$ ):  $q_2$ .
- Estados Finales ( $F''$ ):  $q_0$ .

**5) Transiciones Invertidas ( $\delta''$ ):**

- $q_0 \xrightarrow{0} q_2$  (en  $M'$ ):  $q_2 \xrightarrow{0} q_0$  (en  $M''$ )
- $q_0 \xrightarrow{0} q_1$  (en  $M'$ ):  $q_1 \xrightarrow{0} q_0$  (en  $M''$ )
- $q_1 \xrightarrow{1} q_0$  (en  $M'$ ):  $q_0 \xrightarrow{1} q_1$  (en  $M''$ )

**6) Obtención del AFD Final ( $M$ )**

Para obtener el AFD  $M$ , completamos  $M''$  asegurando que la función de transición  $\delta$  esté definida para todos los estados y símbolos. Introducimos un estado de error  $E$ .  $M = (Q, \Sigma, \delta, q_{in}, F)$ , donde:

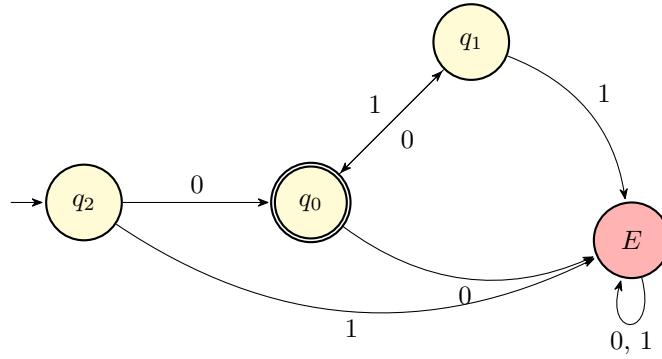
$$Q = \{q_0, q_1, q_2, E\} \quad \Sigma = \{0, 1\} \quad q_{in} = q_2 \quad F = \{q_0\}$$

**7) Función de Transición Final ( $\delta$ ):**

$\delta$	0	1
<b>q<sub>2</sub></b>	$q_0$	$E$
<b>q<sub>0</sub></b>	$E$	$q_1$
<b>q<sub>1</sub></b>	$q_0$	$E$
<b>E</b>	$E$	$E$

**4) Representación Gráfica del AFD**

Representamos el Autómata Finito Determinista  $M$  que reconoce  $L = 0(10)^*$ .

Figura 3.10: Autómata Finito Determinista (AFD) para el lenguaje  $L = 0(10)^*$ .

**Ejercicio 3.3.6.** Construir un Autómata Finito No Determinista (AFND) que acepte las cadenas  $u \in \{0, 1\}^*$  que contengan la subcadena 010. Construir un Autómata Finito No Determinista que acepte las cadenas  $u \in \{0, 1\}^*$  que contengan la subcadena 110. Obtener un AFD que acepte las cadenas  $u \in \{0, 1\}^*$  que contengan simultáneamente las subcadenas 010 y 110.

*Solución 3.3.6.* Partiendo de los AFNDs para las subcadenas 010 y 110 vamos a unirlos rellenando las transiciones que faltan para que sean AFDs. La resolución del grafo se encuentra en la figura 3.11. Se debe de llenar las transiciones faltantes para que sea un AFD.

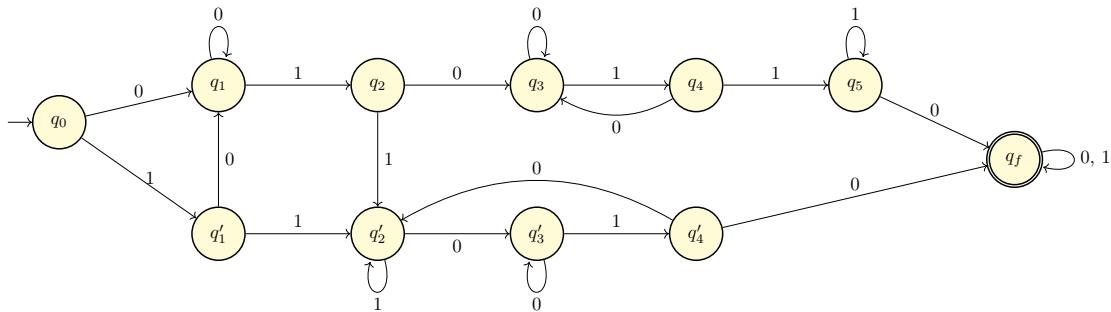


Figura 3.11: Diagrama del AFD para el lenguaje que contiene simultáneamente las subcadenas 010 y 110.

*Solución 3.3.6.* Otra resolución, pero más compleja y extensa, siendo a la vez más metódica es aplicar el producto cartesiano de autómatas.

- 1) Construir AFND<sub>1</sub> para  $L_1 = \{u \in \{0, 1\}^* \mid u \text{ contiene } 010\}$ .
- 2) Construir AFND<sub>2</sub> para  $L_2 = \{u \in \{0, 1\}^* \mid u \text{ contiene } 110\}$ .
- 3) Obtener AFD para  $L = L_1 \cap L_2$ .

#### 1) Construcción del AFND para $L_1$ (subcadena 010)

Para construir un Autómata Finito No Determinista (AFND) que acepte cualquier cadena que contenga la subcadena  $W = 010$ , permitimos transiciones no deterministas en el estado inicial que "adivinan" el comienzo de la subcadena.

Definimos AFND<sub>1</sub> =  $(Q_1, \Sigma, \delta_1, q_0, F_1)$ :

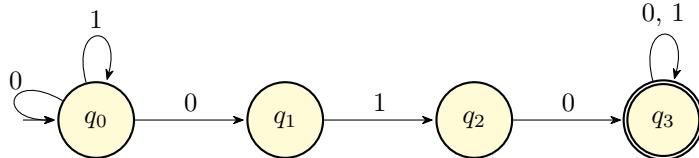
- $Q_1 = \{q_0, q_1, q_2, q_3\}$ , donde los estados representan el prefijo más largo de 010 visto de manera exitosa:  $\varepsilon, 0, 01, 010$ .
- $q_0$  es el estado inicial.
- $F_1 = \{q_3\}$  es el estado final (alcanzar  $q_3$  significa que 010 ha sido encontrado).

La función de transición  $\delta_1$  se centra en el camino  $q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3$ .

Transiciones Clave (No Deterministas):

- Prefijo Arbitrario ( $u_1$ ): En  $q_0$ , el autómata puede leer cualquier símbolo y permanecer en el estado inicial, o intentar comenzar el patrón.  $\delta_1(q_0, a) = \{q_0\}$  para  $a \in \{0, 1\}$ .
- Inicio del Patrón:  $\delta_1(q_0, 0)$  también incluye  $\{q_1\}$ .
- Consumo del Patrón:  $\delta_1(q_1, 1) = \{q_2\}$   $\delta_1(q_2, 0) = \{q_3\}$
- Sufijo Arbitrario ( $u_2$ ): Una vez alcanzado el estado final, cualquier símbolo mantiene el estado de aceptación.  $\delta_1(q_3, a) = \{q_3\}$  para  $a \in \{0, 1\}$ .

Esta estructura coincide con el método estándar de construcción de AFNDs para la subcadena  $W$ .



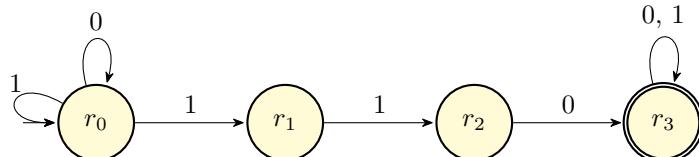
## 2) Construcción del AFND para $L_2$ (subcadena 110)

Procedemos de manera análoga para  $L_2$ , que contiene la subcadena  $W' = 110$ .

Definimos  $AFND_2 = (Q_2, \Sigma, \delta_2, r_0, F_2)$ :

- $Q_2 = \{r_0, r_1, r_2, r_3\}$ , donde los estados representan el prefijo más largo de 110 visto exitosamente.
- $r_0$  es el estado inicial.
- $F_2 = \{r_3\}$ .

La función de transición  $\delta_2$  se centra en el camino  $r_0 \xrightarrow{1} r_1 \xrightarrow{1} r_2 \xrightarrow{0} r_3$ .



## 3) Obtención del AFD para $L_1 \cap L_2$

El lenguaje  $L = L_1 \cap L_2$  (cadenas que contienen simultáneamente 010 y 110) es regular, ya que la familia de lenguajes regulares es cerrada bajo la operación de intersección. Para construir un AFD que acepte  $L$ , utilizamos el método del autómata producto sobre las versiones deterministas (o la lógica determinista) de  $AFND_1$  y  $AFND_2$ .

### 1) Determinización de $AFND_1$ y $AFND_2$

Antes de formar el producto, definimos las funciones de transición deterministas (DFAs)  $M'_1$  y  $M'_2$ . Estos DFAs deben rastrear el prefijo más largo de la subcadena que se ha visto, volviendo al estado adecuado tras un fallo.

DFA  $M'_1$  (Rastreo de 010):  $Q'_1 = \{q_0, q_1, q_2, q_3\}$ .

$$\begin{array}{ll}
 \delta'_1(q_0, 0) = q_1, & \delta'_1(q_0, 1) = q_0 \\
 \delta'_1(q_1, 0) = q_1, & \delta'_1(q_1, 1) = q_2 \\
 \delta'_1(q_2, 0) = q_3, & \delta'_1(q_2, 1) = q_0 \\
 \delta'_1(q_3, 0) = q_3, & \delta'_1(q_3, 1) = q_3
 \end{array}$$

DFA  $M'_2$  (Rastreo de 110):  $Q'_2 = \{r_0, r_1, r_2, r_3\}$ .

$$\begin{array}{ll} \delta'_2(r_0, 0) = r_0, & \delta'_2(r_0, 1) = r_1 \\ \delta'_2(r_1, 0) = r_0, & \delta'_2(r_1, 1) = r_2 \\ \delta'_2(r_2, 0) = r_3, & \delta'_2(r_2, 1) = r_2 \\ \delta'_2(r_3, 0) = r_3, & \delta'_2(r_3, 1) = r_3 \end{array}$$

**2)** Construcción del Autómata Producto *AFD*

El autómata producto es  $M = (Q, \Sigma, \delta, s_{00}, F)$ , donde:

- $Q = Q'_1 \times Q'_2$ . (Máximo  $4 \times 4 = 16$  estados).
- $s_{00} = (q_0, r_0)$ .
- $\delta((q_i, r_j), a) = (\delta'_1(q_i, a), \delta'_2(r_j, a))$ .
- $F = \{(q_i, r_j) \in Q \mid q_i \in F'_1 \text{ y } r_j \in F'_2\} = \{(q_3, r_3)\}$ .

Realizamos la exploración de estados accesibles y transiciones. Notaremos el estado  $(q_i, r_j)$  como  $S_{ij}$ .

Estado $S_{ij} = (q_i, r_j)$	Descripción	0	1
$S_{00} = (q_0, r_0)$	Inicial	$S_{10}$	$S_{01}$
$S_{10} = (q_1, r_0)$	Visto 0	$S_{10}$	$S_{21}$
$S_{01} = (q_0, r_1)$	Visto 1	$S_{10}$	$S_{02}$
$S_{21} = (q_2, r_1)$	Visto 01/1	$S_{30}$	$S_{02}$
$S_{02} = (q_0, r_2)$	Visto 11	$S_{10}$	$S_{02}$
<i>Estados avanzados</i>			
$S_{30} = (q_3, r_0)$	L1 completo	$S_{30}$	$S_{31}$
$S_{31} = (q_3, r_1)$	L1 completo	$S_{30}$	$S_{32}$
$S_{32} = (q_3, r_2)$	L1 completo, visto 11	<b><math>S_{33}</math></b>	$S_{32}$
$S_{22} = (q_2, r_2)$	Visto 01/11	<b><math>S_{33}</math></b>	$S_{02}$
$S_{11} = (q_1, r_1)$	Visto 0/1	$S_{10}$	$S_{22}$
$S_{12} = (q_1, r_2)$	Visto 0/11	$S_{10}$	$S_{22}$
<b><math>S_{33} = (q_3, r_3)</math></b>	<b>L1 y L2 completos</b>	<b><math>S_{33}</math></b>	<b><math>S_{33}</math></b>

Cuadro 3.1: Tabla de transiciones del AFD producto para el lenguaje que contiene simultáneamente las subcadenas 010 y 110.

Observamos que los estados  $S_{11} = (q_1, r_1)$  y  $S_{12} = (q_1, r_2)$  se comportan de manera idéntica al transicionar (ambos van a  $S_{10}$  con 0 y a  $S_{22}$  con 1). Para obtener un AFD resultante más conciso, los fusionamos en un único estado  $S_{1*}$ , resultando en 11 estados distinguibles.

El grafo es equivalente al de la otra solución (Ver figura 3.11).

Observaciones sobre el AFD

Estrategia de Aceptación: El autómata solo alcanza el estado final  $\mathbf{S}_{33} = (q_3, r_3)$  cuando ambas condiciones se han cumplido: el primer componente ( $q_3$ ) confirma la subcadena 010 y el segundo componente ( $r_3$ ) confirma la subcadena 110.

Rutas Críticas: La transición clave es aquella que completa la segunda subcadena mientras la primera ya está completa, o aquella que completa ambas simultáneamente. Por ejemplo:

- Si se lee una cadena que termina en ..,010 pero que anteriormente contenía 11:

- el autómata pasa a un estado  $S_{q_3, r_j}$  y luego debe alcanzar  $r_3$ .
- La cadena 11010. Empieza en  $S_{00}$ .  $1 \rightarrow S_{01}$ .  $1 \rightarrow S_{02}$ .  $0 \rightarrow S_{10}$ .  $1 \rightarrow S_{21}$ .  $0 \rightarrow S_{30}$ . L1 está completa en  $S_{30}$ . Si leemos 10110: ..,11  $\rightarrow S_{32}$ .  $S_{32} \xrightarrow{0} S_{33}$ . L2 se completa aquí.

Correspondencia Producciones-Transiciones: Dado que este lenguaje  $L$  es regular (al ser la intersección de dos lenguajes regulares), sabemos que existe una Gramática Regular de Tipo 3 que lo genera (lineal por la derecha o lineal por la izquierda). Cada transición  $\delta(A, a) = B$  en este AFD corresponde a una regla de producción  $A \rightarrow aB$  en una Gramática Lineal por la Derecha, y el estado final  $S_{33}$  generaría la producción  $S_{33} \rightarrow \varepsilon$ .

Por ejemplo, las transiciones deterministas del AFD implican las siguientes producciones iniciales:

- $S_{00} \rightarrow 0S_{10}$
- $S_{00} \rightarrow 1S_{01}$
- $S_{33} \rightarrow 0S_{33}$
- $S_{33} \rightarrow 1S_{33}$
- $S_{33} \rightarrow \varepsilon$  (Producción de aceptación).

El AFD obtenido es un modelo riguroso y completo para el lenguaje  $L_1 \cap L_2$ .

Acto seguido se debe de representar el AFD obtenido, en este caso se establece que la solución es la misma que la otra solución (Ver figura 3.11)<sup>3</sup>.

**Ejercicio 3.3.7.** Construir un AFD que acepte el lenguaje generado por la siguiente gramática:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aA \\ A &\rightarrow c \\ B &\rightarrow bBb \\ B &\rightarrow b \end{aligned}$$

*Solución 3.3.7.* La solución es el AFD de la figura 3.12.

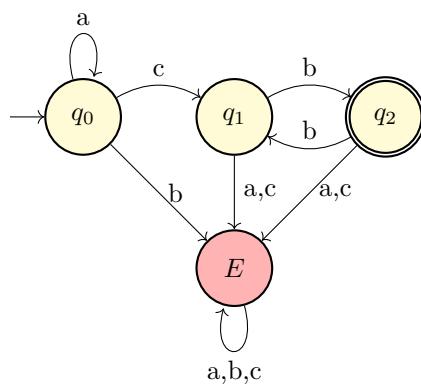


Figura 3.12: Diagrama del AFD resultante del ejercicio 7.

**Ejercicio 3.3.8.** Construir un AFD que acepte el lenguaje  $L \subseteq \{a, b, c\}^*$  de todas las palabras con un número impar de ocurrencias de la subcadena  $abc$ .

<sup>3</sup>El autómata es diferente.

*Solución 3.3.8.* El autómata resultante es el de la figura 3.13.

- $q_0$ : Estado inicial. No se ha encontrado la subcadena  $abc$  (0 ocurrencias, par), o bien se ha encontrado un número par de veces.
- $q_1$ : Se ha encontrado una 'a', esperando 'b'.
- $q_2$ : Se ha encontrado 'ab', esperando 'c'.
- $q_3$ : Estado de aceptación. Se ha encontrado la subcadena 'abc' un número impar de veces.
- $q_4$ : Estado de aceptación. Se puede tener una de las otras repetidas.
- $q_5$ : Estado de aceptación. Se puede tener una de las otras repetidas, pero si recibe una 'c' vuelve a  $q_0$ .

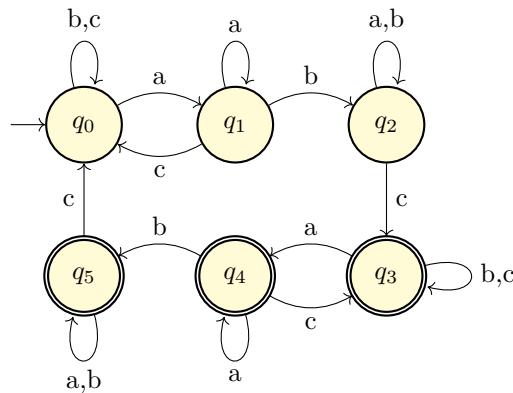


Figura 3.13: Diagrama del AFD resultante del ejercicio 8.

**Ejercicio 3.3.9.** Sea  $L$  el lenguaje de todas las palabras sobre el alfabeto  $\{0, 1\}$  que no contienen dos 1s que estén separados por un número impar de símbolos. Describir un AFD que acepte este lenguaje.

*Solución 3.3.9.* El autómata resultante es el de la figura 3.14.

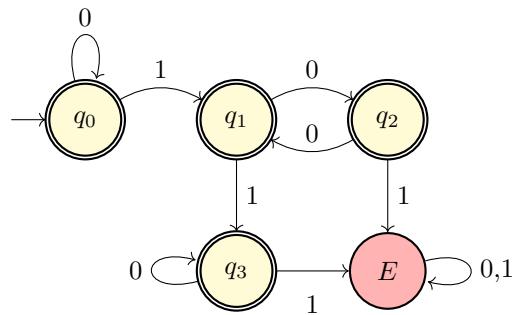


Figura 3.14: Diagrama del AFD resultante del ejercicio 9.

**Ejercicio 3.3.10.** Dada la expresión regular  $(a + \varepsilon)b^*$ , encontrar un AFND asociado y, a partir de este, calcular un AFD que acepte el lenguaje.

*Solución 3.3.10.* El autómata DFND resultante es el de la figura 3.15. El cual podemos simplificar consiguiendo el AFND de la figura 3.16. A partir de este AFND podemos obtener el AFD de la figura 3.17.

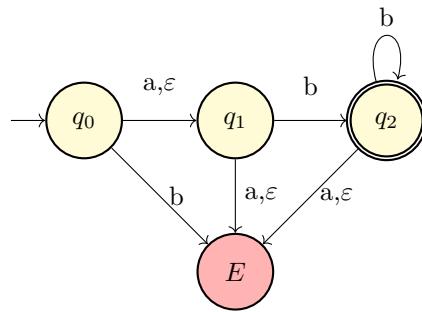


Figura 3.17: Diagrama del AFD resultante del ejercicio 10.

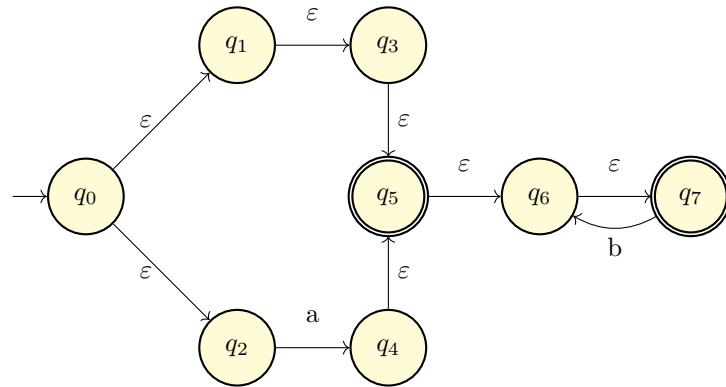


Figura 3.15: Diagrama del AFND resultante del ejercicio 10.

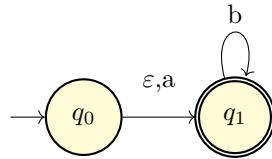


Figura 3.16: Diagrama del AFND resultante del ejercicio 10 simplificado.

**Ejercicio 3.3.11.** Obtener una expresión regular para el lenguaje complementario al aceptado por la gramática

$$S \rightarrow abA \mid B \mid baB \mid \varepsilon, \quad A \rightarrow bS \mid b, \quad B \rightarrow aS$$

*Pista.* Construir un AFD asociado.

*Solución 3.3.11.*

**Ejercicio 3.3.12.** Dar expresiones regulares para los lenguajes sobre el alfabeto  $\{a, b\}$  dados por las siguientes condiciones:

- 1) Palabras que no contienen la subcadena  $a$ .

*Solución 3.3.12. 1).* La expresión regular es  $b^*$ .

- 2) Palabras que no contienen la subcadena  $ab$ .

*Solución 3.3.12. 2).* La expresión regular es  $b^*a^*$ .

- 3) Palabras que no contienen la subcadena  $aba$ .

*Solución 3.3.12. 3).* Para determinar la expresión regular vamos a seguir el proceso de representar el lenguaje mediante un AFD y luego aplicar el método de eliminación de estados

para obtener la expresión regular, usando para ello el Lema de Arden si es necesario. Vemos que el AFD resultante es el de la figura 3.18.

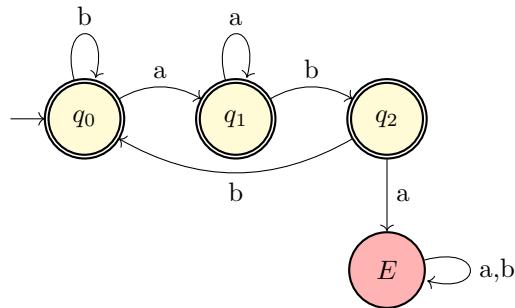


Figura 3.18: Diagrama del AFD resultante del ejercicio 12.

A continuación, aplicamos el método del sistema.

$$\begin{cases} q_0 = bq_0 + aq_1 + \varepsilon \\ q_1 = aq_1 + bq_2 + \varepsilon \\ q_2 = bq_0 + aE + \varepsilon \\ E = aE + bE + \varepsilon = (a+b)\varepsilon + \emptyset \end{cases}$$

Aplicando el lema de Arden obtenemos que:

$$E = \emptyset(a+b)^* = \emptyset$$

Sustituyendo en las ecuaciones anteriores obtenemos:

$$\begin{cases} q_0 = bq_0 + aq_1 + \varepsilon \\ q_1 = aq_1 + bq_2 + \varepsilon \\ q_2 = bq_0 + \varepsilon \end{cases}$$

Sustituimos  $q_2$  y nos queda:

$$\begin{cases} q_0 = bq_0 + aq_1 + \varepsilon \\ q_1 = aq_1 + b(bq_0 + \varepsilon) + \varepsilon \end{cases}$$

Aplicando el Lema de Arden quedaría:

$$q_1 = a^*(b(bq_0 + \varepsilon) + \varepsilon)$$

Sustituimos  $q_1$  en la primera ecuación y nos queda:

$$\begin{aligned} q_0 &= bq_0 + aa^*(b(bq_0 + \varepsilon) + \varepsilon) + \varepsilon = \\ &\quad \text{Aplicamos que } aa^* = a^+ \\ q_0 &= bq_0 + a^+(bbq_0 + b\varepsilon + \varepsilon) + \varepsilon = \\ q_0 &= [b + a^+bb]q_0 + a^+[b + \varepsilon] + \varepsilon \stackrel{\text{Arden}}{=} \\ q_0 &= [b + a^+bb]^* + [a^+[b + \varepsilon]] \end{aligned}$$

Por ende, podemos concluir que la expresión regular asociada al autómata de la figura 3.18 es  $[b + a^+bb]^* + [a^+[b + \varepsilon] + \varepsilon]$ .

**Ejercicio 3.3.13.** Determinar si el lenguaje generado por la siguiente gramática es regular:

$$S \rightarrow AabB, \quad A \rightarrow aA, A \rightarrow bA, A \rightarrow \varepsilon, \quad B \rightarrow Bab, B \rightarrow Bb, B \rightarrow ab, B \rightarrow b$$

En caso de que lo sea, encontrar una expresión regular asociada.

*Solución 3.3.13.* Podemos extraer la expresión regular:

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

**Ejercicio 3.3.14.** Sobre el alfabeto  $A = \{0, 1\}$  realizar las siguientes tareas:

- 1) Describir un autómata finito determinista que acepte todas las palabras que contengan a 011 o a 010 (o las dos) como subcadenas.

*Solución 3.3.14. 1).* La solución es el AFD de la figura 3.19.

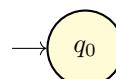


Figura 3.19: Diagrama del AFD resultante del ejercicio 14a.

- 2) Describir un autómata finito determinista que acepte todas las palabras que empiecen por 01 y terminen por 01.
- 3) Dar una expresión regular para el conjunto de las palabras en las que hay dos ceros separados por un número de símbolos que es múltiplo de 4 (los símbolos que separan los ceros pueden ser ceros y puede haber otros símbolos delante o detrás de estos dos ceros).
- 4) Dar una expresión regular para las palabras en las que el número de ceros es divisible por 4.

**Ejercicio 3.3.15.** Construye una gramática regular que genere el siguiente lenguaje:

$$L_1 = \{u \in \{0, 1\}^* \mid \text{el número de 1's y de 0's es impar}\}$$

**Ejercicio 3.3.16.** Encuentra una expresión regular que represente el siguiente lenguaje:

$$L_2 = \{0^n 1^m \mid n \geq 1, m \geq 0, n \text{ múltiplo de } 3 \text{ y } m \text{ es par}\}$$

**Ejercicio 3.3.17.** Diseña un autómata finito determinista que reconozca el siguiente lenguaje:

$$L_3 = \{u \in \{0, 1\}^* \mid \text{el número de 1's no es múltiplo de } 3 \text{ y el número de 0's es par}\}$$

**Ejercicio 3.3.18.** Dar una expresión regular para el lenguaje aceptado por el siguiente autómata:

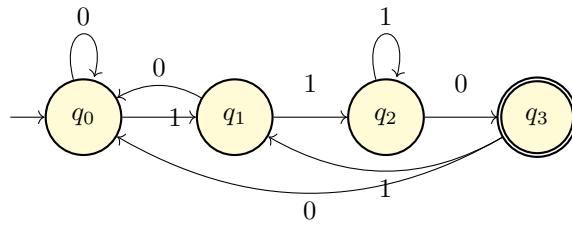


Figura 3.21: Diagrama del AFD resultante del ejercicio 19.

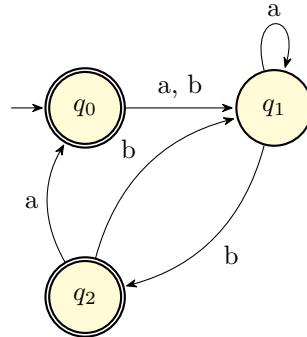


Figura 3.20: Diagrama de Transición para el Ejercicio 18.

**Ejercicio 3.3.19.** Dado el lenguaje

$$L = \{u110 \mid u \in \{1, 0\}^*\}$$

encontrar la expresión regular, la gramática lineal por la derecha, la gramática lineal por la izquierda y el AFD asociado.

*Solución 3.3.19.* La expresión regular asociada es la siguiente:

$$(0 + 1)^* 110$$

La gramática lineal por la derecha es  $G = (\{S, A\}, \{0, 1\}, P, S)$  donde el conjunto de producciones P son:

$$\begin{aligned} S &\rightarrow 0S \mid 1S \mid A \\ A &\rightarrow 110 \end{aligned}$$

La gramática lineal por la izquierda es  $G = (\{S, B\}, \{0, 1\}, P', S)$ , donde el conjunto de producciones P' son:

$$\begin{aligned} S &\rightarrow B110 \\ B &\rightarrow B0 \mid B1 \mid \epsilon \end{aligned}$$

Y el AFD asociado es el de la figura 3.21.

Donde los estados son:

- $q_0$ : Estado inicial. No se ha leído ningún símbolo o se han leído símbolos que no forman parte de la subcadena "110".
- $q_1$ : Se ha leído un '1', esperando '1'.
- $q_2$ : Se han leído "11", esperando '0'.
- $q_3$ : Estado de aceptación. Se ha leído la subcadena "110". Puede volver a  $q_1$  o  $q_0$  para buscar más ocurrencias.

**Ejercicio 3.3.20.** Dado un AFD, determinar el proceso que habría que seguir para construir una Gramática lineal por la izquierda capaz de generar el Lenguaje aceptado por dicho autómata.

**Ejercicio 3.3.21.** Construir un autómata finito determinista que acepte el lenguaje de todas las palabras sobre el alfabeto  $\{0, 1\}$  que no contengan la subcadena 001. Construir una gramática regular por la izquierda a partir de dicho autómata.

**Ejercicio 3.3.22.** Sea  $B_n = \{a^k \mid k \text{ es múltiplo de } n\}$ . Demostrar que  $B_n$  es regular para todo  $n$ .

**Ejercicio 3.3.23.** Sea  $L \subseteq A^*$  y define la relación  $\equiv$  en  $A^*$  como sigue: para  $u, v \in A^*$ , se tiene  $u \equiv v$  si y sólo si para toda  $z \in A^*$ ,  $(uz \in L \Leftrightarrow vz \in L)$ .

- Demostrar que  $\equiv$  es una relación de equivalencia.
- Calcular las clases de equivalencia de  $L = \{a^i b^i \mid i \geq 0\}$ .
- Calcular las clases de equivalencia de  $L = \{a^i b^j \mid i, j \geq 0\}$ .
- Demostrar que  $L$  es aceptado por un autómata finito determinista si y sólo si el número de clases de equivalencia es finito.
- ¿Qué relación existe entre el número de clases de equivalencia y el autómata finito minimal que acepta  $L$ ?

**Ejercicio 3.3.24.** Decimos que  $u$  es un prefijo de  $v$  si existe  $w$  tal que  $uw = v$ . Decimos que  $u$  es un prefijo propio de  $v$  si además  $u \neq v$  y  $u \neq \varepsilon$ . Demostrar que si  $L$  es regular, también lo son los lenguajes

- 1) NOPREFIJO( $L$ ) =  $\{u \in L \mid \text{ningún prefijo propio de } u \text{ pertenece a } L\}$
- 2) NOEXTENSION( $L$ ) =  $\{u \in L \mid u \text{ no es un prefijo propio de ninguna palabra de } L\}$

**Ejercicio 3.3.25.** Dada una palabra  $u = a_1 \dots a_n \in A^*$ , se llama  $\text{Per}(u)$  al conjunto

$$\text{Per}(u) = \{a_{\sigma(1)}, \dots, a_{\sigma(n)} \mid \sigma \text{ es una permutación de } \{1, \dots, n\}\}$$

Dado un lenguaje  $L$ , se llama  $\text{Per}(L) = \bigcup_{u \in L} \text{Per}(u)$ . Dar expresiones regulares y autómatas minimales para  $\text{Per}(L)$  en los siguientes casos:

- 1)  $L = (00 + 1)^*$
- 2)  $L = (0 + 1)^* 0$
- 3)  $L = (01)^*$

¿Es posible que, siendo  $L$  regular,  $\text{Per}(L)$  no lo sea?



## Capítulo 4

---

# Tipo Test

---

*Observación.* Se han eliminado los tipo test repetidos.

### 4.1 Tema 1

- 1) Si un lenguaje es generado por una gramática dependiente del contexto, entonces dicho lenguaje no es independiente del contexto.

**Falso.** La jerarquía de Chomsky establece una relación de inclusión estricta donde los lenguajes regulares  $\subset$  lenguajes independientes del contexto  $\subset$  lenguajes dependientes del contexto  $\subset$  lenguajes recursivamente enumerables. Por tanto, todo lenguaje independiente del contexto es, por definición, también dependiente del contexto. Que un lenguaje sea generado por una gramática dependiente del contexto no excluye la posibilidad de que exista una gramática independiente del contexto (más restrictiva) que también lo genere.

- 2) Los alfabetos tienen siempre un número finito de elementos, pero los lenguajes, incluso si el alfabeto tiene sólo un símbolo, tienen infinitas palabras.

**Falso.** Si bien es cierto que un alfabeto  $\Sigma$  es un conjunto finito y no vacío de símbolos, la segunda parte de la afirmación es incorrecta. Un lenguaje  $L$  se define como un subconjunto de  $\Sigma^*$ . Aunque el universo de todas las palabras posibles  $\Sigma^*$  sea infinito (salvo que  $\Sigma = \emptyset$ , lo cual no es estándar), un lenguaje específico  $L$  puede ser finito. Ejemplo: Si  $\Sigma = \{a\}$ , el lenguaje  $L = \{aa, aaa\}$  es finito.

- 3) Si  $L$  es un lenguaje no vacío, entonces  $L^*$  es infinito.

**Verdadera.** En base a la definición, se sabe que si es no vacío existe un  $u \in L$ , pero existe un caso excepcional. Si el lenguaje  $L$  contiene únicamente la palabra vacía, es decir,  $L = \{\epsilon\}$ , entonces su cierre de Kleene es  $L^* = \{\epsilon\}^* = \{\epsilon\}$ . En este caso,  $L$  no es vacío (tiene un elemento) y  $L^*$  es finito (cardinalidad 1).

- 4) Todo lenguaje con un número finito de palabras es regular e independiente del contexto.

**Verdadero.** Los lenguajes finitos son un subconjunto de los lenguajes regulares. Dado que todo lenguaje finito puede expresarse como la unión finita de sus cadenas constituyentes (y la unión es una operación cerrada en los lenguajes regulares), es regular. Además, como los lenguajes regulares son un subconjunto propio de los independientes del contexto, cualquier lenguaje finito es también independiente del contexto.

- 5) Si  $L$  es un lenguaje, entonces siempre  $L^*$  es distinto de  $L^+$ .

**Falso.** Las definiciones son  $L^* = \bigcup_{i \geq 0} L^i$  y  $L^+ = \bigcup_{i \geq 1} L^i$ . La diferencia entre ambos conjuntos es la palabra vacía  $\epsilon$  correspondiente a  $L^0$ . Sin embargo, si el lenguaje original  $L$  ya contiene a  $\epsilon$ , entonces  $\epsilon \in L \subseteq L^+$ . En consecuencia,  $L^* = L^+ \cup \{\epsilon\} = L^+$ . Por tanto, si  $\epsilon \in L$ ,  $L^* = L^+$ .

- 6)  $L \cdot \emptyset = L$

**Falso.** La concatenación de un lenguaje  $L$  con el conjunto vacío se define como  $L \cdot \emptyset = \{xy \mid x \in L, y \in \emptyset\}$ . Dado que no existe ningún elemento  $y$  en el conjunto vacío, no es posible formar ninguna pareja para la concatenación. El resultado es el conjunto vacío:  $L \cdot \emptyset = \emptyset$ . El elemento neutro de la concatenación es  $\{\epsilon\}$ , no  $\emptyset$ .

- 7) Si  $A$  es un alfabeto, la aplicación que transforma cada palabra  $u \in A^*$  en su inversa es un homomorfismo de  $A^*$  en  $A^*$ .

**Falso.** Para que una aplicación  $h$  sea un homomorfismo, debe cumplir que  $h(xy) = h(x)h(y)$ . La operación de inversa cumple  $(xy)^R = y^Rx^R$ . En general, salvo que el alfabeto tenga un solo símbolo (comutativo),  $y^Rx^R \neq x^Ry^R$ . Por tanto, la inversión no conserva la estructura de la concatenación en el orden original.

- 8) Si  $\epsilon \in L$ , entonces  $L^+ = L^*$ .

**Verdadero.** Por definición,  $L^* = L^+ \cup \{\epsilon\}$ . Si la palabra vacía  $\epsilon$  ya pertenece a  $L$ , entonces cualquier concatenación de elementos de  $L$  que defina  $L^+$  incluirá el caso de una sola repetición de  $\epsilon$ , o simplemente por inclusión de conjuntos  $\epsilon \in L \implies \epsilon \in L^+$ . Al estar  $\epsilon$  presente en  $L^+$ , la unión con  $\{\epsilon\}$  no añade elementos nuevos, resultando en la igualdad de los conjuntos.

- 9) La transformación que a cada palabra sobre  $\{0,1\}^*$  le añade 00 al principio y 11 al final es un homomorfismo.

**Falso.** Definamos la transformación como  $f(w) = 00w11$ . Para ser homomorfismo, debería cumplir  $f(uv) = f(u)f(v)$ . Calculando ambos lados:  $f(uv) = 00uv11$ , mientras que  $f(u)f(v) = (00u11)(00v11) = 00u1100v11$ . Evidentemente,  $00uv11 \neq 00u1100v11$  para palabras genéricas, por lo que no preserva la operación de concatenación.

- 10) Se puede construir un programa que tenga como entrada un programa y unos datos y que siempre nos diga si el programa leído termina para esos datos.

**Falso.** Esto describe el Problema de la Parada (Halting Problem). Alan Turing demostró que este problema es indecidible. No existe un algoritmo general (Máquina de Turing) que pueda determinar correctamente para cualquier par programa-entrada si la ejecución terminará o entrará en un bucle infinito.

- 11) La cabecera (prefijos) del lenguaje  $L$  siempre incluye a  $L$ .

**Verdadero.** Se define el conjunto de prefijos de un lenguaje  $L$  como  $\text{Pref}(L) = \{u \mid \exists v \in \Sigma^*, uv \in L\}$ . Para cualquier palabra  $w \in L$ , podemos elegir  $v = \epsilon$  (la palabra vacía). Como  $w\epsilon = w$  y  $w \in L$ , se cumple la condición. Por tanto, toda palabra de  $L$  es prefijo de sí misma y  $L \subseteq \text{Pref}(L)$ .

- 12) Un lenguaje nunca puede ser igual a su inverso.

**Falso.** Existen lenguajes que son idénticos a su inverso, concretamente aquellos formados por palíndromos o lenguajes donde por cada palabra  $w$ , su inversa  $w^R$  también está en el lenguaje. Un ejemplo trivial es el lenguaje  $L = \{0, 1, 00, 11\}$ , donde  $L = L^R$ . Otro ejemplo es  $\Sigma^*$ , que es igual a su inverso.

- 13) La aplicación que transforma cada palabra  $u$  sobre el alfabeto  $\{0,1\}$  en  $u^3$  (interpretado como  $u^3$ ) es un homomorfismo.

**Falso.** Asumiendo que la transformación es  $f(u) = uuu$ . Para ser homomorfismo debe cumplir  $f(uv) = f(u)f(v)$ . Tenemos  $f(uv) = uvuvuv$ . Por otro lado,  $f(u)f(v) = uuuuvvv$ .

*En general,  $uvuvuv \neq uuvvvv$  (a menos que  $u$  y  $v$  commuten, lo cual no es cierto para todo el alfabeto  $\{0, 1\}$ ).*

- 14) El lenguaje que contiene sólo la palabra vacía es el elemento neutro para la concatenación de lenguajes.

**Verdadero.** *Sea  $L$  un lenguaje cualquiera. La concatenación  $L \cdot \{\epsilon\} = \{w \cdot \epsilon \mid w \in L\} = \{w \mid w \in L\} = L$ . De igual forma  $\{\epsilon\} \cdot L = L$ . Por tanto, el lenguaje  $\{\epsilon\}$  actúa como la identidad o elemento neutro en la estructura monoide de los lenguajes con la operación concatenación.*

- 15) Hay lenguajes con un número infinito de palabras que no son regulares.

**Verdadero.** *La regularidad no depende de la finitud. Existen lenguajes infinitos que requieren memoria ilimitada o estructuras tipo pila para ser reconocidos, escapando de la capacidad de los Autómatas Finitos. El ejemplo clásico es  $L = \{0^n 1^n \mid n \geq 0\}$ , que es infinito y es Independiente del Contexto, pero no Regular (demostrable por el Lema de Bombeo).*

- 16) Si un lenguaje tiene un conjunto infinito de palabras sabemos que no es regular.

**Falso.** *La infinitud no implica no regularidad. El lenguaje universal  $\Sigma^*$  sobre un alfabeto no vacío es infinito y es trivialmente regular (reconocido por un autómata de un solo estado de aceptación con transiciones a sí mismo). Otro ejemplo es  $L = \{a^n \mid n \geq 0\}$ .*

- 17) Si  $L$  es un lenguaje finito, entonces su cabecera ( $CAB(L)$ ) también será finita.

**Verdadero.** *Un lenguaje finito tiene un número finito de palabras, y cada palabra tiene una longitud finita. El número de prefijos (cabecera) de una palabra de longitud  $n$  es  $n + 1$ . La unión de un número finito de conjuntos finitos (los prefijos de cada palabra de  $L$ ) resulta necesariamente en un conjunto finito.*

- 18) El conjunto de palabras sobre un alfabeto dado con la operación de concatenación tiene una estructura de monoide.

**Verdadero.** *Un monoide es una estructura algebraica con una operación asociativa y un elemento neutro. En  $\Sigma^*$ : 1) La concatenación es cerrada (el resultado es una palabra de  $\Sigma^*$ ). 2) Es asociativa:  $(uv)w = u(vw)$ . 3) Existe elemento neutro: la palabra vacía  $\epsilon$ , tal que  $w\epsilon = \epsilon w = w$ .*

- 19) La transformación entre el conjunto de palabras del alfabeto  $\{0, 1\}$  que duplica cada símbolo (la palabra 011 se transforma en 001111) es un homomorfismo.

**Verdadero.** *Definamos el homomorfismo  $h$  mediante las imágenes de los símbolos del alfabeto:  $h(0) = 00$  y  $h(1) = 11$ . La extensión de  $h$  a palabras cumple  $h(a_1 a_2 \dots a_n) = h(a_1)h(a_2) \dots h(a_n)$ . Si tomamos  $u$  y  $v$ , la imagen de su concatenación  $h(uv)$  será la secuencia de símbolos duplicados de  $u$  seguida de los de  $v$ , lo cual es exactamente  $h(u)h(v)$ .*

- 20) Si  $f$  es un homomorfismo entre palabras del alfabeto  $A_1$  en palabras del alfabeto de  $A_2$ , entonces si conocemos  $f(a)$  para cada  $a \in A_1$  se puede calcular  $f(u)$  para cada palabra  $u \in A_1^*$ .

**Verdadero.** *Esta es una propiedad fundamental de los homomorfismos en monoides libres. Un homomorfismo queda unívocamente determinado por su actuación sobre los generadores del monoide (los símbolos del alfabeto). Para cualquier palabra  $u = x_1 x_2 \dots x_k$ , se tiene  $f(u) = f(x_1)f(x_2) \dots f(x_k)$ .*

- 21) Si  $f$  es un homomorfismo, entonces necesariamente se verifica  $f(\epsilon) = \epsilon$ .

**Verdadero.** *En cualquier homomorfismo de monoides, la identidad debe mapearse a la identidad. Formalmente:  $f(w) = f(w\epsilon) = f(w)f(\epsilon)$ . Si asumimos que estamos en un*

monoide libre con la propiedad de cancelación (o simplemente por longitud de palabras  $|f(w)| = |f(w)| + |f(\epsilon)|$ ), se deduce que  $f(\epsilon)$  debe ser la palabra vacía  $\epsilon$ .

- 22) Si  $A$  es un alfabeto, entonces  $A^+$  no incluye nunca la palabra vacía.

**Verdadero.** Un alfabeto  $A$  se define como un conjunto de símbolos. Formalmente, los símbolos tienen longitud 1.  $A^+$  se define como la unión de  $A^1 \cup A^2 \cup \dots$ . Dado que la concatenación de símbolos de longitud  $\geq 1$  siempre produce cadenas de longitud  $\geq 1$ , ninguna combinación en  $A^+$  puede tener longitud 0. Por tanto,  $\epsilon \notin A^+$ .

- 23) Es posible diseñar un algoritmo que lea un lenguaje cualquiera sobre el alfabeto  $\{0,1\}$  y nos diga si es regular o no.

**Falso.** Esta afirmación es falsa por varias razones fundamentales en Teoría de la Computación. Primero, "un lenguaje cualquiera" puede no ser finitamente descriptible. Segundo, si restringimos la entrada a lenguajes definidos por Gramáticas o Máquinas de Turing, el Teorema de Rice establece que cualquier propiedad no trivial de los lenguajes recursivamente enumerables (como "ser regular") es indecidible.

## 4.2 Tema 2

- 1) Si  $r$  y  $s$  son expresiones regulares, tenemos que siempre se verifica que  $(rs)^* = r^*s^*$

**Falso.** El lenguaje  $(rs)^*$  genera repeticiones de la concatenación  $rs$  (ej.  $\epsilon, rs, rsrs, \dots$ ), mientras que  $r^*s^*$  genera cualquier número de  $r$ 's seguido de cualquier número de  $s$ 's. Por ejemplo, la cadena  $r$  pertenece a  $r^*s^*$  pero no a  $(rs)^*$ , a menos que  $s = \epsilon$ .

- 2) Si  $r$  y  $s$  son expresiones regulares, tenemos que siempre se verifica que  $(r+s)^* = r^* + s^*$

**Falso.** El lenguaje  $(r+s)^*$  permite mezclar símbolos de  $r$  y  $s$  en cualquier orden (ej.  $rs, sr, rrs$ ), mientras que  $r^* + s^*$  representa la unión de cadenas formadas solo por  $r$  o solo por  $s$ . La cadena  $rs$  no pertenece a  $r^* + s^*$ .

- 3) Si  $r_1$  y  $r_2$  son expresiones regulares, tales que su lenguaje asociado contiene la palabra vacía, entonces  $(r_1r_2)^* = (r_2r_1)^*$

**Verdadero.** Si  $\epsilon \in L(r_1)$  y  $\epsilon \in L(r_2)$ , entonces  $r_1 \subseteq r_1r_2$  y  $r_2 \subseteq r_1r_2$ , lo que implica que  $r_1 + r_2 \subseteq r_1r_2$ . Esto lleva a que  $(r_1r_2)^*$  cubra todas las combinaciones posibles de  $r_1$  y  $r_2$ , siendo equivalente a  $(r_1 + r_2)^*$ . Por simetría,  $(r_2r_1)^*$  también equivale a  $(r_1 + r_2)^*$ , haciendo la igualdad cierta.

- 4) Si  $r$  y  $s$  son expresiones regulares, tenemos que siempre se verifica que  $(r+\epsilon)^+ = r^*$

**Verdadero.** La operación clausura positiva  $(L)^+$  es  $LL^*$ . Dado que la expresión contiene  $\epsilon$ ,  $(r+\epsilon)^+$  incluye la cadena vacía. La repetición una o más veces de un conjunto que contiene  $\epsilon$  y  $r$  es equivalente a repetir  $r$  cero o más veces, es decir,  $r^*$ .

- 5) Si  $r$  y  $s$  son expresiones regulares, tenemos que siempre se verifica que  $r(r+s)^* = (r+s)^*r$

**Falso.** La expresión de la izquierda genera cadenas que obligatoriamente comienzan por  $r$ , mientras que la de la derecha genera cadenas que obligatoriamente terminan por  $r$ . Por ejemplo, la cadena  $rs$  pertenece al lado izquierdo pero no al derecho.

- 6) Si  $r_1$  y  $r_2$  son expresiones regulares, entonces  $r_1^*r_2^* \subseteq (r_1r_2)^*$  en el sentido de que los lenguajes asociados están incluidos.

**Falso.** El lado izquierdo permite cadenas como  $rr$  (dos  $r_1$  sin  $r_2$ ). El lado derecho  $(r_1r_2)^*$  obliga a que cada aparición de  $r_1$  sea seguida inmediatamente por  $r_2$ . Por tanto,  $rr \notin (r_1r_2)^*$ .

- 7) Si  $r_1$ ,  $r_2$  y  $r_3$  son expresiones regulares, entonces  $(r_1 + r_2)^*r_3 = r_1^*r_3 + r_2^*r_3$

**Falso.** El lado izquierdo permite intercalar  $r_1$  y  $r_2$  antes de  $r_3$  (ej.  $r_1r_2r_3$ ). El lado derecho obliga a elegir o bien una secuencia exclusiva de  $r_1$  o bien una de  $r_2$  antes de  $r_3$ . La distributividad no aplica sobre la clausura de Kleene de una suma.

- 8) Si  $r_1$  y  $r_2$  son expresiones regulares entonces:  $(r_1^*r_2^*)^* = (r_1 + r_2)^*$

**Verdadero.** Esta es una identidad fundamental.  $(r_1^*r_2^*)^*$  permite concatenar bloques de  $r_1^*$  y  $r_2^*$  indefinidamente, lo cual es suficiente para generar cualquier secuencia mezclada de  $r_1$  y  $r_2$ , cubriendo exactamente el mismo lenguaje que  $(r_1 + r_2)^*$ .

- 9) Si  $r$  es una expresión regular, entonces  $r^*r^* = r^*$ .

**Verdadero.** Concatenar dos clausuras de Kleene idénticas equivale a una sola clausura, ya que  $L^*L^* = L^*$ . Cualquier cadena formada por la concatenación de dos cadenas de  $r^*$  sigue siendo una secuencia de elementos de  $r$ , por lo tanto, pertenece a  $r^*$ .

- 10) Si  $r$  es una expresión regular, entonces  $r\emptyset = r + \emptyset$

**Falso.** La concatenación con el conjunto vacío es el conjunto vacío ( $r\emptyset = \emptyset$ ), ya que no es posible formar una cadena si una de las partes no contiene elementos. La unión con el vacío es el elemento neutro ( $r + \emptyset = r$ ). Por tanto,  $\emptyset \neq r$  (salvo si  $r = \emptyset$ ).

- 11) Si  $r$  es una expresión regular, entonces se verifica que  $r^*\epsilon = r^+\epsilon$

**Falso.** La concatenación con  $\epsilon$  es la identidad. La expresión se reduce a  $r^* = r^+$ . Esto es falso porque  $r^*$  siempre contiene la palabra vacía  $\epsilon$ , mientras que  $r^+$  no necesariamente la contiene (solo si  $\epsilon \in r$ ).

- 12) Si  $r_1$  y  $r_2$  son expresiones regulares, entonces siempre  $r_1(r_2r_1)^* = (r_1r_2)^*r_1$

**Verdadero.** Esta es la identidad de desplazamiento de la clausura de Kleene. Ambas expresiones describen secuencias que comienzan con  $r_1$ , alternan entre  $r_2$  y  $r_1$ , y terminan con  $r_1$  (ej.  $r_1r_2r_1r_2r_1$ ).

- 13) Si  $r$  es una expresión regular, entonces  $(rr)^* \subseteq r^*$ .

**Verdadero.**  $(rr)^*$  genera un número par de concatenaciones de  $r$  ( $r^0, r^2, r^4 \dots$ ), mientras que  $r^*$  genera cualquier número ( $r^0, r^1, r^2 \dots$ ). El conjunto de cadenas con repeticiones pares es un subconjunto del total de repeticiones.

- 14) Si  $r_1$  y  $r_2$  son expresiones regulares, tales que su lenguaje asociado contiene la palabra vacía, entonces  $(r_1r_2)^* = (r_1 + r_2)^*$ .

**Verdadero.** Bajo la condición de que  $\epsilon \in L(r_1)$  y  $\epsilon \in L(r_2)$ , la concatenación  $r_1r_2$  incluye a  $r_1$  (pues  $r_1 \cdot \epsilon$ ) y a  $r_2$  (pues  $\epsilon \cdot r_2$ ). Al aplicar la clausura, se pueden generar todas las combinaciones de la unión.

- 15) Si  $r_1, r_2, r_3$  son expresiones regulares, entonces  $r_1(r_2^* + r_3^*) = r_1r_2^* + r_1r_3^*$ .

**Verdadero.** La concatenación es distributiva respecto a la unión (suma) en las expresiones regulares.  $A(B + C) = AB + AC$ .

- 16) La demostración de que la clase de lenguajes aceptados por los autómatas no deterministas es la misma que la aceptada por los autómatas deterministas, se basa en dado un autómata no determinista construir uno determinista que, ante una palabra de entrada, explore todas las posibles opciones que puede seguir el no determinista.

**Verdadero.** Se refiere al algoritmo de construcción de subconjuntos. El AFD resultante simula en paralelo todos los caminos posibles del AFND, manteniendo en cada paso el conjunto de estados en los que podría estar el autómata no determinista.

- 17) Un autómata finito puede ser determinista y no-determinista a la vez.

**Verdadero.** *Formalmente, la definición de AFD es un caso particular de la definición de AFND. Todo autómata determinista es, por definición, también un autómata no determinista que casualmente no tiene transiciones  $\epsilon$  ni múltiples transiciones para un mismo símbolo.*

- 18) Para transformar un autómata que acepta el lenguaje  $L$  en uno que acepte  $L^*$ , basta unir los estados finales con el inicial mediante transiciones nulas.

**Falso.** *No basta con unir los finales al inicial. Primero, el lenguaje  $L^*$  debe aceptar  $\epsilon$ , por lo que el estado inicial debe ser final o se debe crear uno nuevo. Segundo, si el estado inicial original tenía transiciones entrantes, hacer este bucle puede aceptar cadenas que no están en  $L^*$  (se crean caminos indeseados). La construcción correcta implica un nuevo estado inicial  $q'_0$  final y transiciones  $\epsilon$  adecuadas.*

- 19) Para pasar de un autómata que acepte el lenguaje asociado a  $r$  a uno que acepte  $r^*$  basta con unir con transiciones nulas sus estados finales con el estado inicial.

**Falso.** *Misma justificación que la pregunta 21. Es necesario introducir un nuevo estado inicial para asegurar la aceptación de la palabra vacía sin alterar el flujo interno del autómata original y evitar bucles erróneos.*

- 20) Existe un lenguaje reconocido por un AFD y no generado por una gramática independiente del contexto.

**Falso.** *Los lenguajes regulares (reconocidos por AFD) son un subconjunto estricto de los lenguajes libres de contexto (generados por GIC). Todo lenguaje regular puede ser generado por una GIC.*

- 21) Existen lenguajes aceptados por AFD que no pueden ser aceptados por AF no determinísticos.

**Falso.** *Los AFD y los AFND tienen exactamente el mismo poder expresivo. Ambos reconocen la clase de los lenguajes regulares.*

- 22) La clausura de un lenguaje aceptado por un AFD puede ser representado con una expresión regular.

**Verdadero.** *La clase de los lenguajes regulares es cerrada bajo la operación estrella de Kleene. Si  $L$  es regular,  $L^*$  es regular y, por el Teorema de Kleene, todo lenguaje regular puede representarse mediante una expresión regular.*

- 23) Un lenguaje representado por una expresión regular siempre puede ser reconocido por un AF no determinista.

**Verdadero.** *Es parte fundamental del Teorema de Kleene. Existen algoritmos estándar (como la construcción de Thompson) para convertir cualquier expresión regular en un AFND- $\epsilon$  equivalente.*

- 24) Todo lenguaje regular puede ser generado por una gramática libre de contexto.

**Verdadero.** *Como los lenguajes regulares son un subconjunto de los libres de contexto, siempre existe una gramática regular (que es un tipo de GIC) que genera dicho lenguaje.*

- 25) Un lenguaje con un número finito de palabras siempre puede ser reconocido por un AF no determinista.

**Verdadero.** *Todo lenguaje finito es un lenguaje regular. Se puede construir trivialmente un autómata (incluso un AFD tipo "trie") que acepte exactamente esas palabras y ninguna más.*

- 26) Todo autómata finito determinista de  $n$  estados, cuyo alfabeto  $A$  contiene  $m$  símbolos debe tener  $m * n$  transiciones.

**Verdadero.** Por definición formal de AFD, la función de transición  $\delta : Q \times \Sigma \rightarrow Q$  es una función total. Debe haber exactamente una transición definida para cada par (estado, símbolo), resultando en  $m \times n$  transiciones.

- 27) Para que un autómata con pila sea determinista es necesario que no tenga transiciones nulas.  
**Falso.** Un autómata con pila determinista (APD) puede tener transiciones nulas ( $\epsilon$ -transiciones), siempre que no exista ambigüedad. Es decir, si un estado tiene una transición  $\epsilon$ , no puede tener ninguna otra transición con un símbolo de entrada para esa misma configuración de pila.
- 28) Si  $r_1$  y  $r_2$  son expresiones regulares, entonces siempre se tiene que  $(r_1 + r_2)^* = (r_1^* r_2)^* r_1^*$ .  
**Verdadero.** Esta es una identidad conocida. El lado derecho describe cualquier secuencia de bloques donde cada bloque termina en  $r_2$  (precedido por cualquier cantidad de  $r_1$ ), finalizando la cadena global con cualquier cantidad de  $r_1$ . Esto cubre todas las posibles mezclas de  $r_1$  y  $r_2$ , igual que  $(r_1 + r_2)^*$ .
- 29) Si un lenguaje es infinito no se puede encontrar una expresión regular que lo represente.  
**Falso.** La mayoría de las expresiones regulares de interés representan lenguajes infinitos (gracias al operador estrella de Kleene). Por ejemplo,  $a^*$  representa un lenguaje infinito y es una expresión regular válida.
- 30) Si  $r_1$  y  $r_2$  son expresiones regulares, entonces se verifica que  $(r_1 + \epsilon)^+ r_2^+ = r_1^+ (r_2 + \epsilon)^+$ .  
**Falso.** Analicemos los lenguajes:  $(r_1 + \epsilon)^+$  equivale a  $r_1^*$ . Así, el lado izquierdo es  $r_1^* r_2^+$  (cadenas de  $r_1$  seguidas de al menos una  $r_2$ ). El lado derecho es  $r_1^+ r_2^*$  (al menos una  $r_1$  seguida de cadenas de  $r_2$ ). Son conjuntos diferentes (ej.  $r_2$  está en el izquierdo pero no en el derecho).
- 31) El conjunto de palabras sobre el alfabeto  $\{0, 1\}$  tales que eliminando los tres últimos símbolos, en la palabra resultante no aparece el patrón 0011 es un lenguaje regular.  
**Verdadero.** Los lenguajes regulares son cerrados bajo la operación de cociente (eliminación de sufijos finitos) y diferencia. El lenguaje de palabras que contienen "0011" es regular. Su complemento (no contienen "0011") es regular. Concatenar con  $\Sigma\Sigma\Sigma$  (los tres últimos) mantiene la regularidad.
- 32) El lenguaje formado por las cadenas sobre  $\{0, 1\}$  que tienen un número impar de 0 y un número par de 1 no es regular.  
**Falso.** Este lenguaje es regular. Se puede construir un AFD simple con 4 estados que controlan la paridad de los ceros y unos (Par-Par, Par-Impar, Impar-Par, Impar-Impar) para reconocerlo.

## 4.3 Tema 3

- 1) El lema de bombeo puede usarse para demostrar que un lenguaje determinado es regular.  
**Falso.** El lema de bombeo es una condición necesaria pero no suficiente para la regularidad. Se utiliza mediante demostración por reducción al absurdo para probar que un lenguaje NO es regular, pero el cumplimiento de la condición no garantiza que el lenguaje sea regular.
- 2) Todo lenguaje con un número finito de palabras es regular.  
**Verdadero.** Un lenguaje finito es la unión finita de lenguajes unitarios (que contienen una sola palabra), los cuales son regulares. Dado que los lenguajes regulares son cerrados bajo la unión, cualquier lenguaje finito es regular.

- 3) La intersección de lenguajes regulares es siempre regular.

**Verdadero.** *La clase de los lenguajes regulares es cerrada bajo la operación de intersección. Esto se puede demostrar constructivamente mediante el autómata producto de los dos autómatas finitos deterministas correspondientes.*

- 4) La demostración del lema de bombeo se basa en que si leemos una palabra de longitud mayor o igual al número de estados del autómata, entonces en el camino que se recorre en el diagrama de transición se produce un ciclo.

**Verdadero.** *Se fundamenta en el principio del palomar: si una cadena tiene una longitud n mayor o igual al número de estados  $|Q|$ , el camino debe visitar  $n + 1$  estados, forzando la repetición de al menos un estado y formando un ciclo que puede ser bombeado.*

- 5) Es más fácil determinar si una palabra pertenece a un lenguaje regular cuando éste viene dado por una expresión regular que cuando viene dado por un autómata finito determinista.

**Falso.** *El mecanismo computacional más eficiente y directo para verificar la pertenencia de una palabra es el Autómata Finito Determinista (AFD), que lo hace en tiempo lineal  $O(n)$ . Las expresiones regulares suelen requerir una conversión previa a autómata o algoritmos de backtracking menos eficientes.*

- 6) En la demostración de que todo autómata finito tiene una expresión regular que representa el mismo lenguaje, el conjunto  $R_{ij}^k$  se define como el lenguaje de todas las palabras que llevan al autómata del estado  $q_i$  al estado  $q_j$  pasando por el estado número k,  $q_k$ .

**Falso.** *Se trata del conjunto de todas las palabras que llevan al autómata del estado  $q_i$  al estado  $q_j$  con numeración menor o igual a k. No es obligatorio que pasen por el estado  $q_k$ , sino que pueden pasar por cualquier estado con índice hasta k.*

- 7) El conjunto de todas las expresiones regulares es un lenguaje regular.

**Falso.** *El conjunto de cadenas que conforman las expresiones regulares válidas requiere el balanceo de paréntesis (estructura anidada), lo cual es una característica de los lenguajes independientes del contexto (Tipo 2), no de los regulares (Tipo 3).*

- 8) A partir de la demostración de que si R es regular y L un lenguaje cualquiera, entonces  $R/L$  es regular, se puede obtener un algoritmo para construir el autómata asociado a  $R/L$ .

**Verdadero.** *El cociente de un lenguaje regular por cualquier lenguaje es regular. El autómata se construye identificando qué estados del AFD original pueden llegar a un estado final consumiendo alguna cadena de L y marcándolos como finales en el nuevo autómata.*

- 9) En un autómata finito no-determinista, si intercambio entre sí los estados finales y no finales obtengo un autómata que acepta el lenguaje complementario.

**Falso.** *El intercambio de estados finales y no finales solo genera el lenguaje complementario en Autómatas Finitos Deterministas (AFD). En el caso no determinista (AFND), es necesario determinizar el autómata antes de realizar la inversión de estados.*

- 10) Si en un autómata finito no hay estados distinguibles de nivel 2, ya no puede haber estados distinguibles de nivel 4.

**Falso.** *La indistinguibilidad a nivel k no implica indistinguibilidad a nivel k + n. Es posible que dos estados se comporten igual para todas las cadenas de longitud 2, pero que diverjan para cadenas de longitud mayor (por ejemplo, longitud 4), revelando su distinguibilidad en etapas posteriores.*

- 11) Todo lenguaje generado por una gramática lineal por la derecha es también generado por una gramática lineal por la izquierda.

**Verdadero.** Las gramáticas lineales por la derecha y las lineales por la izquierda generan exactamente la misma clase de lenguajes: los lenguajes regulares. Por tanto, siempre existe una gramática equivalente del otro tipo.

- 12) Un autómata finito determinista sin estados inaccesibles ni indistinguibles es minimal.  
**Verdadero.** Un AFD es mínimo (tiene el menor número de estados posible para su lenguaje) si y solo si no contiene estados inaccesibles (inalcanzables desde el inicio) y no existen pares de estados equivalentes (indistinguibles).
- 13) Si  $L$  es una lenguaje sobre el alfabeto  $A$ , entonces  $CAB(L)$  es siempre igual al cociente  $L/A^*$ .  
**Verdadero.**  $CAB(L)$  denota el conjunto de prefijos de  $L$ . El cociente por la derecha  $L/A^*$  se define como  $\{x \mid \exists y \in A^*, xy \in L\}$ , lo cual corresponde exactamente a la definición de los prefijos de las palabras de  $L$ .
- 14) El lenguaje de las palabras sobre  $\{0,1\}$  en las que la diferencia entre el número de ceros y unos es impar es regular.  
**Verdadero.** Este lenguaje puede ser reconocido por un autómata finito que controle la paridad de la diferencia (módulo 2). Los estados representarían si la diferencia actual es par o impar, lo cual requiere memoria finita.
- 15) En un autómata finito cualquiera, si las transiciones dan lugar a un ciclo, entonces el lenguaje aceptado es infinito.  
**Falso.** La existencia de un ciclo es condición necesaria pero no suficiente para la infinitud. El ciclo debe ser accesible desde el estado inicial y, además, debe existir un camino desde el ciclo hacia un estado final. Si el ciclo está en una parte "muerta" o no conduce a aceptación, el lenguaje puede ser finito.
- 16) La expresión recursiva que se emplea para obtener la expresión regular asociada a un autómata finito determinista es:  $r_{ij}^k = r_{ij}^{k-1} + r_{i(k-1)}^{k-1}(r_{(k-1)(k-1)}^{k-1})^*r_{(k-1)j}^{k-1}$ .  
**Falso.** La fórmula correcta del algoritmo de Kleene utiliza el estado  $k$  como pivote en la iteración  $k$ , no el estado  $k - 1$ . La expresión correcta es  $r_{ij}^k = r_{ij}^{k-1} + r_{ik}^{k-1}(r_{kk}^{k-1})^*r_{kj}^{k-1}$ . La fórmula presentada en el enunciado tiene los índices desplazados incorrectamente.
- 17) Cuando se construye la expresión regular asociada a un autómata finito determinista,  $r_{ii}^0$  no puede ser nunca vacío.  
**Verdadero.** El término  $r_{ii}^0$  representa los caminos de longitud 0 o 1 de  $i$  a  $i$  sin estados intermedios. Como siempre es posible ir de un estado a sí mismo con la cadena vacía  $\epsilon$ , el lenguaje  $r_{ii}^0$  contiene al menos a  $\epsilon$  y nunca es el conjunto vacío.
- 18) El conjunto de las palabras  $\{u0011v^{-1} : u, v \in \{0,1\}^*\}$  es regular.  
**Verdadero.** Asumiendo que la notación denota el cociente por la derecha o una operación relacionada con prefijos/sufijos sobre un lenguaje regular (definido por el patrón 0011), las propiedades de clausura (cociente, concatenación) aseguran que el resultado sea regular.
- 19) Si  $L$  es un lenguaje finito, entonces su complementario es siempre regular.  
**Verdadero.** Si  $L$  es finito, es regular. La clase de lenguajes regulares es cerrada bajo la complementación. Por tanto, el complemento de un lenguaje finito es regular (y co-finito).
- 20) En un autómata finito determinista la relación de indistinguibilidad es una relación de equivalencia.  
**Verdadero.** La relación de indistinguibilidad (o equivalencia de Myhill-Nerode) cumple las propiedades reflexiva, simétrica y transitiva, particionando el conjunto de estados en clases

de equivalencia.

- 21) En un autómata finito determinista siempre debe de existir, al menos, un estado de error.  
**Falso.** *No es obligatorio. Un estado de error (pozo) solo es necesario si la función de transición debe ser total y existen transiciones que llevan al rechazo permanente. Si el lenguaje es  $\Sigma^*$  o el autómata está definido parcialmente (aunque formalmente un AFD es total), no "debe" existir siempre explícitamente un estado de error específico si la lógica no lo requiere.*
- 22) El conjunto de los números en binario que son múltiplos de 7 es regular.  
**Verdadero.** *La divisibilidad por un número entero constante  $k$  en cualquier base es una propiedad regular. Se puede construir un AFD donde los estados representan el resto de la división por 7 (estados 0 a 6).*
- 23) Hay situaciones en las que los estados inaccesibles de un AFD cumplen una función específica.  
**Falso.** *Los estados inaccesibles no pueden ser alcanzados por ninguna cadena de entrada desde el estado inicial, por lo que no afectan en absoluto al lenguaje aceptado por el autómata y son irrelevantes para la función de reconocimiento.*
- 24) Si  $R$  es un lenguaje regular y  $L$  un lenguaje independiente del contexto, entonces  $R/L$  es regular.  
**Verdadero.** *Esta es una propiedad fuerte de los lenguajes regulares: el cociente de un lenguaje regular por **cualquier** lenguaje (sea regular, libre de contexto o arbitrario) resulta siempre en un lenguaje regular.*
- 25) Si en un autómata dos estados son distinguibles de nivel  $n$ , entonces serán distinguibles de nivel  $m$  para todo  $m \geq n$ .  
**Verdadero.** *Ser distinguible de nivel  $n$  significa que existe una palabra de longitud  $\leq n$  (o exactamente  $n$ , según definición, pero implica existencia en el conjunto de palabras) que los distingue. Si tal palabra existe, sigue existiendo y siendo válida para distinguir a un "nivel" superior que abarca longitudes mayores.*
- 26) Si  $h$  es un homomorfismo y  $h(L)$  no es regular, podemos concluir que  $L$  no es regular.  
**Falso.** *Los lenguajes regulares son cerrados bajo homomorfismos, por lo que si  $L$  fuera regular,  $h(L)$  tendría que ser regular. Por contraposición, si  $h(L)$  NO es regular, entonces  $L$  NO puede ser regular. (Nota: El enunciado dice "podemos concluir que  $L$  no es regular", lo cual es lógicamente **Verdadero** por contraposición:  $\text{Reg}(L) \implies \text{Reg}(h(L))$  equivale a  $\neg\text{Reg}(h(L)) \implies \neg\text{Reg}(L)$ ). Corrección interpretativa: A veces se confunde con la inversa. Si  $h(L)$  es regular, no implica nada sobre  $L$ . Pero si el resultado NO es regular, la fuente no podía serlo.*
- 27) El lenguaje de todas las palabras en las que los tres primeros símbolos son iguales a los tres últimos es regular.  
**Verdadero.** *Es un lenguaje que impone condiciones finitas sobre prefijos y sufijos de longitud fija. Puede ser reconocido por un AFD que recuerde los tres primeros símbolos y verifique los tres últimos mediante el estado actual. Una explicación más formal sería construyendo una aplicación biyectiva  $f : \{1, 2, \dots, n^3\} \rightarrow A \times A \times A$ , ya que suponemos que tenemos una posibilidad finita de combinaciones.*
- 28) Si un lenguaje verifica la condición que aparece en el lema de bombeo para lenguajes regulares, ya no hay forma de demostrar que no es regular.  
**Falso.** *El cumplimiento del lema de bombeo no garantiza regularidad. Existen lenguajes no*

regulares que satisfacen el lema. Aún se podría demostrar que no es regular utilizando el teorema de Myhill-Nerode o propiedades de clausura.

- 29) Si  $f$  es un homomorfismo entre alfabetos  $f : A_1^* \rightarrow A_2^*$  y  $L \subseteq A_1^*$  no es regular, podemos concluir que  $f(L)$  tampoco es regular.

**Falso.** Un lenguaje no regular puede transformarse en regular mediante un homomorfismo. Por ejemplo, un homomorfismo que mapee todos los símbolos a una constante o a la cadena vacía puede convertir un lenguaje complejo en uno regular trivial.

- 30) Todo lenguaje que cumple la condición del lema de bombeo para lenguajes regulares puede ser aceptado por un autómata finito no determinista.

**Falso.** Los autómatas finitos (deterministas o no) solo aceptan lenguajes regulares. Dado que existen lenguajes no regulares que cumplen el lema de bombeo, estos no pueden ser aceptados por ningún autómata finito.

- 31) No existe algoritmo para saber si el lenguaje generado por una gramática regular es finito.

**Falso.** Sí existe. Dado que una gramática regular equivale a un autómata finito, se puede verificar la finitud comprobando si existen ciclos en el grafo del autómata que sean alcanzables desde el inicio y que puedan alcanzar un estado final.

- 32) Dos autómatas finitos deterministas con diferente número de estados y que aceptan el lenguaje vacío tienen el mismo número de estados finales.

**Falso.** Un autómata que acepta el vacío puede tener 0 estados finales, o puede tener estados finales que sean inaccesibles. Por tanto, el número de estados finales no tiene por qué ser igual (aunque usualmente es 0 en la versión mínima, versiones no mínimas pueden diferir).

- 33) Si  $A$  es un alfabeto y  $L$  un lenguaje cualquiera distinto del vacío, entonces se verifica que  $A^*/L = A^*$ .

**Verdadero.** El cociente  $A^*/L$  contiene todas las cadenas  $x$  tales que  $xy \in A^*$  para algún  $y \in L$ . Como la concatenación de cualquier palabra con otra siempre está en  $A^*$ , solo necesitamos que exista al menos una  $y \in L$ . Si  $L \neq \emptyset$ , entonces para todo  $x \in A^*$ , existe tal  $y$ , por lo que el resultado es todo  $A^*$ .

- 34) Si  $R_{ij}^k$  son los lenguajes que se usan en la construcción de una expresión regular a partir de una autómata finito, siempre se verifica que  $R_{ij}^{j-1}R_{jk}^{j-1} \subseteq R_{ik}^j$ .

**Verdadero.** El término  $R_{ik}^j$  incluye todos los caminos de  $i$  a  $k$  pasando por nodos intermedios  $\{1..j\}$ . La concatenación  $R_{ij}^{j-1}R_{jk}^{j-1}$  representa un camino que va de  $i$  a  $j$  (sin pasar por  $j$  antes) y luego de  $j$  a  $k$  (sin pasar por  $j$  después). Este es un camino válido específico dentro del conjunto más general  $R_{ik}^j$ .

- 35) El lema de bombeo es útil para demostrar que la intersección de dos lenguajes regulares no es regular.

**Falso.** La premisa es incorrecta: la intersección de dos lenguajes regulares ES siempre regular, por lo que el lema de bombeo nunca podría demostrar lo contrario.

- 36) Existe un algoritmo para determinar si el lenguaje generado por una gramática regular es infinito.

**Verdadero.** Se puede convertir la gramática a un AFD y comprobar la existencia de ciclos útiles (accesibles y co-accesibles a estados finales).

- 37) Existe un algoritmo para determinar si el lenguaje generado por una gramática regular es finito o infinito.

**Verdadero.** Es el mismo procedimiento mencionado anteriormente (detección de ciclos en el autómata asociado).

- 38) La intersección de dos lenguajes regulares da lugar a un lenguaje independiente del contexto.

**Verdadero.** La intersección de dos regulares es un lenguaje regular. Dado que la clase de lenguajes regulares es un subconjunto propio de los lenguajes independientes del contexto, el resultado es, por definición, también un lenguaje independiente del contexto.

- 39) Si un lenguaje es infinito no se puede encontrar una expresión regular que lo represente.

**Falso.** Muchos lenguajes infinitos son regulares (por ejemplo,  $a^*$ ) y tienen representaciones mediante expresiones regulares.

- 40) En un autómata finito determinista sin estados inaccesibles la relación de indistinguibilidad entre los estados es una relación de equivalencia.

**Verdadero.** Independientemente de la accesibilidad, la relación de Myhill-Nerode es siempre de equivalencia. La condición de "sin estados inaccesibles" es relevante para la minimización, pero no altera la naturaleza algebraica de la relación.

- 41) En un autómata finito determinista, si no hay dos estados que sean indistinguibles entre sí, entonces el autómata es minimal.

**Falso.** Para ser minimal, además de no tener estados indistinguibles, el autómata no debe tener estados inaccesibles. Se puede tener un autómata sin indistinguibles pero con estados inalcanzables que sobran.

- 42) Dada una gramática lineal por la derecha, siempre existe otra gramática lineal por la izquierda que acepte el mismo lenguaje.

**Verdadero.** Ambos formalismos generan exactamente la clase de los lenguajes regulares, por lo que son equivalentes en poder expresivo.

- 43) Si  $R$  es un lenguaje regular y  $L$  un lenguaje cualquiera, entonces  $R/L$  es siempre un lenguaje regular.

**Verdadero.** Como se mencionó anteriormente, la clase regular es cerrada bajo el cociente con cualquier lenguaje arbitrario.

- 44) Si un lenguaje cumple la condición del lema de bombeo para conjuntos regulares no nos asegura que sea un lenguaje regular.

**Verdadero.** El lema de bombeo no es una condición suficiente. Existen lenguajes no regulares que satisfacen la propiedad de bombeo.

- 45) Existe un algoritmo para determinar si los lenguajes generados por dos gramáticas regulares son iguales o no.

**Verdadero.** Se pueden convertir ambas gramáticas a AFDs, minimizarlos y comprobar si son isomorfos. La equivalencia de lenguajes regulares es decidible.

- 46) El conjunto de cadenas aceptado por un autómata finito no determinista con transiciones nulas no puede ser generado por una gramática independiente del contexto.

**Falso.** Un AFND con transiciones nulas acepta un lenguaje regular. Todo lenguaje regular es independiente del contexto, por lo que ciertamente puede ser generado por una gramática independiente del contexto.

- 47) El lenguaje resultado de la unión de dos lenguajes regulares con un número infinito de palabras puede ser representado mediante una expresión regular.

**Verdadero.** La unión de dos lenguajes regulares es regular, independientemente de si son finitos o infinitos. Todo lenguaje regular puede representarse mediante una expresión regular.

- 48) Una expresión regular siempre representa a un lenguaje que puede ser generado por una gramática independiente del contexto.

**Verdadero.** Las expresiones regulares denotan lenguajes regulares, los cuales son un subconjunto de los lenguajes independientes del contexto.

- 49) Existe un algoritmo para comprobar si son iguales los lenguajes aceptados por dos autómatas finitos diferentes.

**Verdadero.** Es un problema decidable. Se puede verificar si la diferencia simétrica de los lenguajes  $(L(A_1) \cap \overline{L(A_2)}) \cup (\overline{L(A_1)} \cap L(A_2))$  es vacía.

- 50) Si en un autómata finito no determinista intercambio entre sí los estados finales y no finales obtengo un autómata que acepta el lenguaje complementario del aceptado por el autómata original.

**Falso.** Esta propiedad solo es válida para Autómatas Finitos Deterministas. En un AFND, el intercambio de estados no garantiza la complementación correcta debido a la naturaleza de los caminos múltiples.

- 51) Si  $L$  es un lenguaje regular, entonces el lenguaje  $LL^{-1}$  es también regular.

**Verdadero.** Esta expresión denota el cociente  $L/L$ . Dado que los regulares son cerrados bajo cociente con cualquier lenguaje (incluso consigo mismos), el resultado es regular.

- 52) El lema de bombeo para lenguajes regulares es útil para demostrar que un lenguaje determinado no es regular.

**Verdadero.** Es su aplicación principal: demostrar la no regularidad mediante contradicción.

- 53) Si un lenguaje tiene un conjunto infinito de palabras sabemos que no es regular.

**Falso.** La infinitud no implica no regularidad. El lenguaje de todas las cadenas  $\Sigma^*$  es infinito y es regular.

- 54) Un autómata finito determinista sin estados inaccesibles ni indistinguibles es minimal.

**Verdadero.** Cumple las dos condiciones necesarias y suficientes para la minimalidad: todos los estados son útiles (accesibles) y todos son necesarios (distinguibles).

- 55) El conjunto de las palabras  $\{u0011v^{-1} : u, v \in \{0,1\}^*\}$  es regular.

**Verdadero.** Interpretando la operación como un cociente o manipulación de prefijos sobre un patrón regular, las propiedades de clausura mantienen la regularidad.

- 56) Existe un algoritmo para determinar si el lenguaje generado por una gramática regular es infinito.

**Verdadero.** Verificación de ciclos en el autómata correspondiente.

- 57) Para cada autómata finito no determinista  $M$  existe una gramática independiente de contexto  $G$  tal que  $L(M) = L(G)$ .

**Verdadero.** El lenguaje  $L(M)$  es regular. Todo lenguaje regular es libre de contexto, por lo que existe una gramática  $G$  (que puede ser regular o lineal) que lo genera.

- 58) El lenguaje formado por las cadenas sobre  $\{0,1\}$  que tienen un número impar de 0 y un número par de 1 no es regular.

**Falso.** Es regular. Se puede construir un AFD con 4 estados representando las combinaciones de paridad (par/par, par/impar, impar/par, impar/impar) para reconocerlo.

- 59) Si  $L$  es un lenguaje regular, entonces la cabecera de  $L$  ( $CAB(L)$ ) es siempre regular.
- Verdadero.** *CAB( $L$ ) o el conjunto de prefijos de un lenguaje regular es siempre regular. Se obtiene haciendo finales todos los estados desde los que se puede alcanzar un estado final original en el AFD.*
- 60) En un autómata finito determinista, si no hay dos estados que sean indistinguibles entre sí, entonces el autómata es minimal.
- Falso.** *Falta la condición de no tener estados inaccesibles. Un autómata puede tener estados todos distinguibles entre sí, pero si alguno es inalcanzable desde el inicio, no es minimal.*
- 61) La intersección de dos lenguajes regulares da lugar a un lenguaje independiente del contexto.
- Verdadero.** *Da lugar a un lenguaje regular, y todo regular es independiente del contexto.*
- 62) Si un lenguaje es infinito no se puede encontrar una expresión regular que lo represente.
- Falso.** *Las expresiones regulares con el operador estrella de Kleene (\*) representan precisamente lenguajes infinitos (e.g.,  $a^*$  representa un conjunto infinito de 'a's).*

## 4.4 Tema 4

- 1) Si un lenguaje de tipo 2 viene generado por una gramática ambigua, siempre puedo encontrar una gramática no ambigua que genere el mismo lenguaje.
- Falso.** *Existen los denominados lenguajes inherentemente ambiguos. Para estos lenguajes, todas las gramáticas independientes del contexto posibles que los generan son ambiguas, por lo que es teóricamente imposible hallar una gramática no ambigua para ellos.*
- 2) En una gramática de tipo 2 ambigua no puede existir una palabra generada con un único árbol de derivación.
- Falso.** *La definición de ambigüedad en una gramática exige únicamente la existencia de al menos una palabra con más de un árbol de derivación distinto. Esto no excluye la posibilidad de que existan otras palabras dentro del mismo lenguaje que posean un único árbol de derivación.*
- 3) Dada una gramática independiente del contexto, siempre se puede construir una gramática sin transiciones nulas ni unitarias que genere exactamente el mismo lenguaje que la gramática original.
- Falso.** *Si el lenguaje original  $L$  contiene la cadena vacía ( $\epsilon$ ), la eliminación estándar de producciones nulas genera una gramática para  $L \setminus \{\epsilon\}$ . Para generar exactamente  $L$  (incluyendo  $\epsilon$ ), es imprescindible mantener al menos una producción nula asociada al símbolo inicial ( $S \rightarrow \epsilon$ ), por lo que no es posible eliminar absolutamente todas las transiciones nulas sin alterar el lenguaje.*
- 4) Una gramática independiente del contexto es ambigua si existe una palabra que puede ser generada con dos cadenas de derivación distintas.
- Falso.** *La existencia de dos cadenas de derivación distintas no implica ambigüedad si estas corresponden al mismo árbol de derivación (por ejemplo, una derivación por la izquierda y otra por la derecha). La condición necesaria para la ambigüedad es la existencia de dos árboles de sintaxis distintos o, equivalentemente, dos derivaciones por la izquierda distintas para la misma palabra.*
- 5) Un lenguaje inherentemente ambiguo puede ser generado por una gramática ambigua.
- Verdadero.** *Por definición, un lenguaje es inherentemente ambiguo si todas las gramáticas*

que lo generan son ambiguas. Por tanto, necesariamente existe una gramática ambigua que lo genera.

- 6) El lenguaje de las palabras sobre  $\{0,1\}$  con un número impar de ceros es independiente del contexto.

**Verdadero.** *Este lenguaje es regular (reconocible por un autómata finito de dos estados). Dado que el conjunto de los lenguajes regulares es un subconjunto propio de los lenguajes independientes del contexto ( $\mathcal{L}_3 \subset \mathcal{L}_2$ ), el lenguaje es también independiente del contexto.*

- 7) Si en una producción de una gramática independiente del contexto, uno de los símbolos que contiene es útil, entonces la producción es útil.

**Falso.** *La utilidad de una producción es una propiedad integral. Para que una producción sea útil, debe ser alcanzable desde el símbolo inicial y todos sus símbolos deben ser capaces de derivar terminales. La utilidad de un símbolo aislado no garantiza que la producción en su conjunto cumpla estas condiciones (por ejemplo, podría contener otro símbolo no generativo).*

- 8) Todo árbol de derivación de una palabra en una gramática independiente del contexto está asociado a una única derivación por la izquierda.

**Verdadero.** *Existe un isomorfismo (una correspondencia biunívoca) entre el conjunto de árboles de derivación y el conjunto de derivaciones por la izquierda. Un árbol de derivación determina unívocamente el orden de expansión de las variables en una derivación "left-most".*

- 9) Para poder aplicar el algoritmo que hemos visto para transformar una gramática a forma normal de Greibach, la gramática tiene que estar en forma normal de Chomsky necesariamente.

**Falso.** *Si está en forma normal de Chomsky (FNC), es más sencillo aplicar el algoritmo para convertirla a forma normal de Greibach (FNG), pero no es un requisito indispensable. Basta con tener una gramática sin producciones nulas ni unitarias, y aplicar lo siguiente: Para toda producción  $A \rightarrow \alpha$ , cambiar todo símbolo terminal a que aparezca en  $\alpha$  a partir del segundo símbolo por  $C_a$  (Si  $a$  aparece al principio de  $\alpha$  este símbolo terminal no se cambia).*

- 10) Sólo hay una derivación por la derecha asociada a un árbol de derivación.

**Verdadero.** *Al igual que con las derivaciones por la izquierda, existe una correspondencia uno a uno entre un árbol de derivación y su derivación canónica por la derecha (right-most derivation).*

- 11) Si una gramática independiente del contexto no tiene producciones nulas ni unitarias, entonces si  $u$  es una palabra de longitud  $n$  generada por la gramática, su derivación se obtiene en un número de pasos no superior a  $2n - 1$ .

**Verdadero.** *En el "peor caso" de expansión (Forma Normal de Chomsky), cada paso añade un terminal o divide una variable en dos, requiriendo exactamente  $2n - 1$  pasos. Otras producciones que generen múltiples terminales o variables en un solo paso reducirán este número, por lo que  $2n - 1$  actúa como una cota superior estricta.*

- 12) Existe un lenguaje con un número finito de palabras que no puede ser generado por una gramática libre de contexto.

**Falso.** *Todo lenguaje finito es un lenguaje regular (puede construirse un autómata finito que reconozca exactamente esas palabras). Dado que todos los lenguajes regulares son libres de contexto, no existe ningún lenguaje finito que escape a la capacidad generativa de una gramática libre de contexto.*

- 13) La gramática compuesta por las reglas de producción  $S \rightarrow AA$ ,  $A \rightarrow aSa$ ,  $A \rightarrow a$  no es

ambigua.

**Falso.** La gramática es ambigua. Consideremos la cadena  $aaaa$ . Puede generarse mediante  $S \rightarrow AA \rightarrow aA \rightarrow a(aSa) \dots$  o mediante  $S \rightarrow AA \rightarrow (aSa)A \dots$ . Estas derivaciones dan lugar a árboles de estructura distinta, probando la ambigüedad.

- 14) Un lenguaje libre de contexto es inherentemente ambiguo si existe una gramática ambigua que lo genera.  
**Falso.** La existencia de una gramática ambigua no implica la ambigüedad inherente del lenguaje. Un lenguaje es inherentemente ambiguo solo si **todas** las gramáticas que lo generan son ambiguas. A menudo es posible sanear una gramática ambigua para obtener una no ambigua para el mismo lenguaje.
- 15) La gramática compuesta por las reglas de producción  $S \rightarrow A$ ,  $A \rightarrow aSa$ ,  $A \rightarrow a$  es ambigua.  
**Falso.** Esta gramática genera el lenguaje  $\{a^{2n+1} : n \geq 0\}$ . La estructura es lineal y determinista: para generar una cadena de longitud específica, existe una única secuencia de aplicaciones de reglas. Al haber un único árbol para cada palabra, la gramática no es ambigua.
- 16) Para generar una palabra de longitud  $n$  en una gramática en forma normal de Chomsky hacen falta exactamente  $2n - 1$  pasos de derivación.  
**Verdadero.** En FNC, las producciones son de la forma  $A \rightarrow BC$  (aumentan la longitud de la forma sentencial en 1 variable) o  $A \rightarrow a$  (convierten variable en terminal). Para una cadena de  $n$  terminales, se requieren  $n - 1$  aplicaciones de reglas tipo  $A \rightarrow BC$  para crear  $n$  variables, y  $n$  aplicaciones de  $A \rightarrow a$  para terminarlas. Total:  $(n - 1) + n = 2n - 1$ .
- 17) Es imposible que una gramática esté en forma normal de Chomsky y Greibach al mismo tiempo.  
**Falso.** Es posible en casos triviales o específicos. Por ejemplo, una gramática con solo reglas de la forma  $S \rightarrow a$  cumple la definición de FNC ( $A \rightarrow a$ ) y también la de FNG ( $A \rightarrow a\alpha$ , donde  $\alpha = \epsilon$ ).
- 18) En una gramática independiente del contexto, si una palabra de longitud  $n$  es generada, entonces el número de pasos de derivación que se emplean debe de ser menor o igual a  $2n - 1$ .  
**Falso.** Esta cota solo es válida para gramáticas simplificadas (sin producciones unitarias ni nulas) o en formas normales. En una gramática general con ciclos unitarios (e.g.,  $A \rightarrow B, B \rightarrow A$ ) o producciones nulas, la derivación puede tener una longitud arbitraria o incluso infinita para la misma palabra.
- 19) El algoritmo que pasa una gramática a forma normal de Greibach produce siempre el mismo resultado con independencia de cómo se numeren las variables.  
**Falso.** El algoritmo se basa en un ordenamiento arbitrario de las variables no terminales ( $A_1, < A_2 < \dots < A_m$ ) para eliminar la recursividad por la izquierda. Cambiar este orden altera las sustituciones realizadas y, por tanto, modifica el conjunto final de reglas de producción, aunque el lenguaje generado sea el mismo.
- 20) La gramática compuesta por las siguientes reglas de producción  $\{S \rightarrow A|BA|SS, B \rightarrow a|b, A \rightarrow a\}$  es ambigua.  
**Verdadero.** La cadena  $aa$  puede generarse mediante  $S \rightarrow SS \rightarrow AA \rightarrow aa$  o mediante  $S \rightarrow BA \rightarrow aA \rightarrow aa$  (dado que  $B \rightarrow a$ ). Al existir dos árboles de derivación distintos para  $aa$ , es ambigua.
- 21) Si una palabra de longitud  $n$  es generada por una gramática en forma normal de Greibach, entonces lo es con  $n$  pasos de derivación exactamente.

**Verdadero.** En FNG, toda producción es de la forma  $A \rightarrow a\alpha$ , donde  $a$  es un terminal. Cada paso de derivación introduce exactamente un símbolo terminal en la cadena. Por consiguiente, para generar una cadena de  $n$  terminales, se requieren exactamente  $n$  pasos.

- 22) En una gramática independiente del contexto puede existir una palabra que es generada con dos derivaciones por la izquierda distintas que tienen el mismo árbol de derivación.  
**Falso.** Por la propiedad de biyección entre árboles y derivaciones izquierdas, es imposible tener dos derivaciones por la izquierda distintas que correspondan al mismo árbol. Si las derivaciones izquierdas difieren, los árboles asociados necesariamente difieren.
- 23) Una gramática independiente del contexto genera un lenguaje que puede ser representado por una expresión regular.  
**Falso.** Las gramáticas independientes del contexto tienen una capacidad generativa superior a las expresiones regulares. Existen lenguajes libres de contexto (como  $\{a^n b^n\}$ ) que no son regulares y, por tanto, no pueden ser representados mediante expresiones regulares.
- 24) Para cada autómata finito no determinista  $M$  existe una gramática independiente de contexto  $G$  tal que  $L(M) = L(G)$ .  
**Verdadero.** Un autómata finito reconoce un lenguaje regular. Todo lenguaje regular es también independiente del contexto. Específicamente, se puede construir una gramática regular (lineal por la derecha) que simule las transiciones del autómata.
- 25) Para que un autómata con pila sea determinista es necesario que no tenga transiciones nulas.  
**Falso.** Un autómata con pila determinista (DPDA) permite transiciones nulas ( $\lambda$ -transiciones), siempre que se cumpla la condición de determinismo: si en una configuración dada es posible una transición nula, no debe ser posible ninguna otra transición (por lectura de entrada) desde esa misma configuración.
- 26) El conjunto de cadenas generado por una gramática independiente del contexto en forma normal de Greibach puede ser reconocido por un autómata finito no determinista con transiciones nulas.  
**Falso.** Estar en Forma Normal de Greibach no reduce la capacidad generativa de la gramática; sigue generando un lenguaje independiente del contexto (posiblemente no regular). Los autómatas finitos (incluso AFND- $\epsilon$ ) solo pueden reconocer lenguajes regulares.
- 27) La intersección de dos lenguajes regulares da lugar a un lenguaje independiente del contexto.  
**Verdadero.** La clase de los lenguajes regulares es cerrada bajo la intersección ( $Reg \cap Reg = Reg$ ). Dado que todo lenguaje regular es independiente del contexto, el resultado pertenece a esta clase.
- 28) Si  $L_1$  y  $L_2$  son independientes del contexto, no podemos asegurar que  $L_1 \cap L_2$  también lo sea.  
**Verdadero.** La clase de lenguajes independientes del contexto no es cerrada bajo la intersección. La intersección de dos lenguajes libres de contexto puede dar lugar a un lenguaje sensible al contexto que no sea libre de contexto (ejemplo clásico:  $\{a^n b^n c^n\}$ ).

## 4.5 Tema 5

- 1) La clase de los lenguajes aceptados por los autómatas con pila deterministas es igual a la clase de los lenguajes generados por las gramáticas de tipo 2.  
**Falso.** Las gramáticas de tipo 2 generan la clase completa de los Lenguajes Independientes del Contexto (LIC). Los Autómatas con Pila Deterministas (APD) reconocen únicamente

*un subconjunto propio de estos, conocidos como Lenguajes Independientes del Contexto Deterministas (LICD). Para reconocer la clase completa de los LIC (Tipo 2) es necesario un Autómata con Pila No Determinista (APND).*

- 2) Una palabra es aceptada por un autómata con pila por el criterio de pila vacía si en algún momento, cuando leemos esta palabra, la pila se queda sin ningún símbolo, con independencia de la cantidad de símbolos que hayamos leído de la palabra de entrada.

**Falso.** *El criterio de aceptación por pila vacía exige que la pila se vacíe únicamente después de haber consumido la totalidad de la cadena de entrada. Si la pila se vacía antes de terminar de leer la entrada, el autómata se bloquea y la cadena no es aceptada (salvo que pueda realizar transiciones nulas para consumirla, pero la condición de vaciado es al final del procesamiento).*

- 3) Un autómata con pila siempre acepta el mismo lenguaje por los criterios de pila vacía y de estados finales.

**Falso.** *Aunque los dos criterios de aceptación son equivalentes en cuanto a su capacidad expresiva (la clase de lenguajes que pueden definir es la misma), un autómata con pila M específico generalmente acepta lenguajes distintos bajo cada criterio ( $L(M) \neq N(M)$ ) a menos que haya sido diseñado explícitamente para que coincidan.*

- 4) Todo lenguaje aceptado por un autómata con pila determinista por el criterio de estados finales es también aceptado por una autómata con pila determinista por el criterio de pila vacía.

**Falso.** *Para que un lenguaje aceptado por un APD por estados finales (un LICD estándar) pueda ser aceptado por un APD por pila vacía, el lenguaje debe cumplir necesariamente la propiedad prefijo. Si no la cumple, no existe un APD que lo reconozca por vaciado de pila.*

- 5) Para que un autómata con pila sea determinista es suficiente que desde cada configuración se pueda obtener, a lo más, otra configuración en un paso de cálculo.

**Verdadero.** *Esta es una definición dinámica equivalente de determinismo. Implica que para cualquier par estado-símbolo en la cima de la pila, no existe ambigüedad entre realizar una transición consumiendo una entrada o una transición nula ( $\epsilon$ ), y que cada transición tiene un único destino definido.*

- 6) Si un lenguaje de tipo 2 verifica la propiedad prefijo y es aceptado por un autómata con pila determinista por el criterio de estados finales, entonces también es aceptado por un autómata con pila determinista por el criterio de pila vacía.

**Verdadero.** *Existe un teorema que establece que un lenguaje puede ser aceptado por un APD por pila vacía si y solo si es un lenguaje determinista (aceptado por APD por estados finales) y además posee la propiedad prefijo.*

- 7) Para todo autómata con pila existe otro autómata con pila que acepta el mismo lenguaje y tiene un solo estado.

**Verdadero.** *Es posible convertir cualquier autómata con pila en una gramática libre de contexto equivalente, y posteriormente construir un autómata con pila a partir de dicha gramática utilizando el método estándar que emplea un único estado (donde la lógica reside enteramente en la pila).*

- 8) En un autómata con pila determinista no puede haber transiciones nulas.

**Falso.** *Un APD puede tener transiciones nulas ( $\epsilon$ -transiciones) siempre y cuando no generen no determinismo. Esto significa que si existe una transición  $\delta(q, \epsilon, X)$ , no puede existir*

ninguna transición  $\delta(q, a, X)$  para ningún símbolo de entrada  $a \in \Sigma$ .

- 9) Si  $L$  es independiente del contexto determinista y  $\$ \notin L$  entonces  $L.\{\$\}$  es aceptado por un autómata con pila determinista por el criterio de pila vacía.

**Verdadero.** Añadir un símbolo de fin de cadena único (end-marker) garantiza que el lenguaje resultante cumpla la propiedad prefijo, permitiendo así que un autómata determinista vacíe su pila al detectar dicho símbolo.

- 10) El conjunto de las palabras  $\{u0011u^{-1} : u \in \{0,1\}^*\}$  es libre del contexto determinista.

**Falso.** Este lenguaje representa palíndromos con un marcador central "0011". Sin embargo, dado que  $u$  está formado por  $\{0,1\}$ , el marcador "0011" podría aparecer dentro de la propia  $u$ , introduciendo ambigüedad sobre dónde está el centro de la cadena. El autómata no puede determinar deterministicamente cuándo empezar a verificar la segunda mitad  $u^{-1}$ .

- 11) En la construcción de una gramática independiente del contexto a partir de un autómata con pila, la variable  $[p, X, q]$  genera todas las palabras que llevan al autómata desde el estado  $p$  al estado  $q$  sustituyendo  $X$  por el símbolo inicial de la pila.

**Falso.** La variable  $[p, X, q]$  genera las cadenas que permiten transitar del estado  $p$  al estado  $q$  con el efecto neto de **eliminar** (desapilar) el símbolo  $X$  de la pila, no de sustituirlo por el símbolo inicial.

- 12) Todo autómata con pila determinista que acepta un lenguaje por pila vacía se puede transformar en otro autómata determinista que acepte el mismo lenguaje por el criterio de estados finales.

**Verdadero.** La clase de lenguajes aceptados por APD por pila vacía es un subconjunto estricto de los aceptados por estados finales (aquellos con propiedad prefijo). Siempre podemos convertir el primero en el segundo añadiendo un estado final específico al que se transita cuando la pila detecta un fondo de pila nuevo.

- 13) Para que un lenguaje independiente del contexto sea determinista ha de verificar la propiedad prefijo.

**Falso.** La definición estándar de Lenguaje Independiente del Contexto Determinista es aquel aceptado por un APD por **estados finales**. Esta definición no requiere la propiedad prefijo. La propiedad prefijo solo es necesaria para la aceptación por pila vacía.

- 14) El lenguaje compuesto por las instrucciones completas del lenguaje SQL cumplen la propiedad prefijo.

**Falso.** La propiedad prefijo implica que ningún prefijo propio de una palabra del lenguaje pertenece al lenguaje. En SQL, una instrucción válida (ej. 'SELECT \* FROM T') puede ser prefijo de otra instrucción válida más compleja (ej. 'SELECT \* FROM T WHERE id=1'), violando así esta propiedad (asumiendo que no se considera el delimitador de fin de instrucción ';' como parte estricta de la sintaxis interna que se evalúa o si se permiten instrucciones anidadas).

- 15) En el algoritmo para pasar un autómata con pila a gramática que hemos visto, si el autómata tiene 3 estados, entonces la transición  $(p, XYZU) \in \delta(q, \epsilon, H)$  da lugar a  $4^3$  producciones.

**Falso.** El número de producciones generadas es  $|Q|^m$ , donde  $|Q|$  es el número de estados y  $m$  es el número de símbolos apilados. En este caso,  $|Q| = 3$  y se apilan 4 símbolos ( $X, Y, Z, U$ ). Por tanto, se generan  $3^4 = 81$  producciones, no  $4^3 = 64$ .

- 16) El lenguaje  $\{0^i 1^k 2^i : i, k \geq 0\}$  es independiente del contexto determinista.

**Verdadero.** Un APD puede reconocer este lenguaje: apila símbolos para los 0, cambia de

*estado al ver 1 (consumiéndolos sin modificar la pila o ignorándolos mediante cambios de estado), y finalmente desapila los símbolos almacenados al leer los 2. El determinismo se mantiene porque los símbolos de entrada son distintos y ordenados.*

- 17) Si tenemos un lenguaje L aceptado por un Autómata con Pila por el criterio de estados finales, podemos encontrar otro AP que reconozca L por el criterio de pila vacía.  
**Verdadero.** *Para autómatas con pila no deterministas (el caso general), los criterios de aceptación por pila vacía y por estados finales son equivalentes. Existe un algoritmo constructivo para transformar uno en otro.*
- 18) La propiedad prefijo no tiene ninguna relación con el hecho de que un lenguaje sea aceptado por un autómata con pila determinista por estados finales.  
**Verdadero.** *Un lenguaje puede ser LIC determinista (aceptado por estados finales) independientemente de si cumple o no la propiedad prefijo. La propiedad prefijo es relevante solo para la aceptación determinista por pila vacía.*
- 19) Para toda gramática libre de contexto G siempre se puede encontrar un autómata con pila que acepte el lenguaje generado por G.  
**Verdadero.** *Es un resultado fundamental de la teoría de autómatas: la clase de lenguajes generados por Gramáticas Libres de Contexto es exactamente la misma que la clase de lenguajes aceptados por Autómatas con Pila.*
- 20) Si un lenguaje independiente del contexto cumple la propiedad prefijo, entonces puede ser aceptado por un autómata con pila determinista por el criterio de pila vacía.  
**Falso.** *El hecho de cumplir la propiedad prefijo no convierte a un lenguaje en determinista. Existen lenguajes inherentemente ambiguos (no deterministas) que tienen la propiedad prefijo. La afirmación correcta es: "Si un lenguaje es determinista y cumple la propiedad prefijo, entonces es aceptado por un APD por pila vacía".*
- 21) La descripción instantánea de un autómata con pila nos permite saber el estado activo, lo que queda por leer de la cadena de entrada, lo que se ha consumido de la cadena de entrada y lo que nos queda en la pila.  
**Falso.** *La descripción instantánea se define formalmente como la tripla  $(q, w, \alpha)$ , donde  $q$  es el estado actual,  $w$  es la cadena de entrada **restante** y  $\alpha$  es el contenido de la pila. No contiene explícitamente la información de la parte de la cadena que ya ha sido consumida.*
- 22) Un autómata finito determinista se puede convertir en un autómata con pila que acepta el mismo lenguaje por el criterio de pila vacía.  
**Verdadero.** *Los lenguajes regulares (reconocidos por AFD) son un subconjunto de los lenguajes libres de contexto. Un autómata con pila puede simular un AFD ignorando la pila y, mediante no determinismo, vaciar la pila al final de la cadena para aceptar (o determinísticamente si el lenguaje regular tiene propiedad prefijo).*
- 23) El conjunto de cadenas generado por una gramática libre de contexto en forma normal de Greibach puede ser reconocido por un autómata finito no determinista con transiciones nulas.  
**Falso.** *La Forma Normal de Greibach genera Lenguajes Independientes del Contexto, los cuales son una superclase estricta de los lenguajes regulares. Un autómata finito (incluso no determinista) solo puede reconocer lenguajes regulares, por lo que no puede reconocer todos los lenguajes generados por gramáticas en FNG.*
- 24) Puede existir un lenguaje con pila determinista que no sea aceptado por un autómata con pila determinista por el criterio de estados finales.

**Falso.** Por definición, la clase de los Lenguajes Independientes del Contexto Deterministas está formada por aquellos lenguajes que son aceptados por algún Autómata con Pila Determinista bajo el criterio de estados finales.

- 25) Existe un algoritmo para transformar una gramática regular G en un autómata con pila que acepte las cadenas del lenguaje generado por G por el criterio de pila vacía.

**Verdadero.** Dado que una gramática regular genera un lenguaje regular, y los lenguajes regulares son un subconjunto de los libres de contexto, existe un autómata con pila que lo acepta. Específicamente, se puede convertir a un AP que acepta por pila vacía.

- 26) Para que un lenguaje sea aceptado por una autómata con pila determinista por el criterio de pila vacía tiene que verificar la propiedad prefijo.

**Verdadero.** Esta es una condición necesaria y suficiente (junto con ser determinista) establecida en la teoría de APDs. Si un lenguaje tiene una palabra que es prefijo de otra, un APD vaciaría la pila con el prefijo y no podría continuar procesando la palabra más larga, ya que se habría bloqueado.

- 27) Si tenemos un autómata con pila en el que  $(p, \epsilon) \in \delta(q, a, C)$ , entonces para construir una gramática independiente del contexto que genere el mismo lenguaje que acepta el autómata, debemos de añadir la producción  $[p, C, q] \rightarrow a$  (según el procedimiento visto en clase).

**Falso.** En el algoritmo estándar de conversión, si la transición va de  $q$  a  $p$  leyendo  $a$  y desapilando  $C$  (transición  $a \epsilon$  en pila), la producción generada debe ser  $[q, C, p] \rightarrow a$ . El enunciado invierte los estados en la variable de la gramática ( $[p, C, q]$  en lugar de  $[q, C, p]$ ).

- 28) El lenguaje  $L = \{u \in \{0, 1\}^*: u = u^{-1}\}$  es independiente del contexto, pero no determinista.

**Verdadero.** El lenguaje corresponde a los palíndromos sobre el alfabeto  $\{0, 1\}$ . Es un ejemplo clásico de lenguaje libre de contexto que es inherentemente no determinista, ya que el autómata no puede saber determinísticamente cuándo ha alcanzado la mitad de la cadena para comenzar a verificar la coincidencia con la pila.

## 4.6 Tema 6

- 1) La intersección de lenguajes libres de contexto es siempre libre de contexto.

**Falso.** Los lenguajes libres de contexto (LLC) no son cerrados bajo la intersección. Un contraejemplo clásico es la intersección de  $L_1 = \{a^n b^n c^m \mid n, m \geq 0\}$  y  $L_2 = \{a^m b^n c^n \mid n, m \geq 0\}$ . Ambos son LLC, pero su intersección es  $L = \{a^n b^n c^n \mid n \geq 0\}$ , el cual es un lenguaje dependiente del contexto.

- 2) Existe un algoritmo para determinar si una palabra es generada por una gramática independiente del contexto.

**Verdadero.** Este es el problema de la pertenencia (membership problem). Existen algoritmos eficientes para decidir esto, siendo el más destacado el algoritmo CYK (Cocke-Younger-Kasami), que opera en tiempo  $O(n^3)$ , o el algoritmo de Earley.

- 3) El lenguaje  $\{a^i b^j c^i d^i : i, j \geq 0\}$  es independiente del contexto.

**Falso.** Este lenguaje presenta una dependencia triple entre los símbolos  $a$ ,  $c$  y  $d$  ( $i$  aparece tres veces). Un autómata con pila no puede verificar tres conteos simultáneos que dependen de la misma variable si están separados (la pila solo permite acceso LIFO). Al aplicar el lema de bombeo, si intentamos bombear la sección de  $a^i$ , necesariamente desajustamos la cantidad de  $c^i$  o  $d^i$  de manera que no se puede compensar con solo dos segmentos bombeables  $v$  y  $x$ .

- 4) Existe un algoritmo para determinar si una gramática independiente del contexto es ambigua.

**Falso.** *El problema de la ambigüedad para Gramáticas Independientes del Contexto (GIC) es indecidible. No existe (ni puede existir) un algoritmo general que determine si una GIC arbitraria es ambigua.*

- 5) Existe un algoritmo para comprobar cuando dos gramáticas libres de contexto generan el mismo lenguaje.

**Falso.** *El problema de la equivalencia entre dos GIC ( $L(G_1) = L(G_2)$ ) es un problema indecidible. No se puede determinar algorítmicamente si dos gramáticas distintas generan el mismo conjunto de cadenas.*

- 6) El lenguaje  $L = \{0^i 1^j 2^k : 1 \leq i \leq j \leq k\}$  es independiente del contexto.

**Falso.** *Las restricciones  $i \leq j$  y  $j \leq k$  encadenadas requieren comparar tres cantidades secuenciales. Un autómata con pila tendría que apilar las 0s para compararlas con las 1s, pero al desapilar para verificar  $i \leq j$ , pierde la información de la cantidad de 1s necesaria para verificar  $j \leq k$ .*

- 7) Si el lenguaje  $L$  es independiente del contexto, entonces  $L^{-1}$  es independiente del contexto.

**Verdadero.** *Los lenguajes independientes del contexto son cerrados bajo la operación de reverso (o imagen especular). Si existe una GIC que genera  $L$ , se puede construir una GIC para  $L^R$  simplemente invirtiendo el lado derecho de todas las producciones ( $A \rightarrow \alpha$  pasa a ser  $A \rightarrow \alpha^R$ ).*

- 8) Existe un algoritmo que permite determinar si una gramática independiente del contexto genera un lenguaje finito o infinito.

**Verdadero.** *Es decidable. Se puede verificar construyendo el grafo de dependencias de las variables (eliminando previamente producciones inútiles). Si el grafo contiene algún ciclo desde el cual se puedan generar terminales, el lenguaje es infinito; de lo contrario, es finito.*

- 9) En el algoritmo de Earley, la presencia del registro (2,5, A, CD, adS) implica que a partir de CD se puede generar la subcadena de la palabra de entrada que va del carácter 3 al 5.

**Verdadero.** *Asumiendo la notación estándar de elementos de Earley o la implícita en la asignatura donde los índices marcan los límites entre caracteres: un ítem con intervalo (2,5) indica que la parte de la producción ya analizada (o la que se está intentando casar, en este caso derivada de CD) abarca desde la posición inmediatamente posterior al índice 2 (carácter 3) hasta el índice 5 (carácter 5).*

- 10) Existe un algoritmo para comprobar si el lenguaje generado por una gramática libre de contexto es regular.

**Falso.** *Determinar si un Lenguaje Independiente del Contexto es en realidad un Lenguaje Regular es un problema indecidible. No existe un algoritmo general para verificar si  $L(G)$  puede ser generado por un autómata finito.*

- 11) El algoritmo de Earley se puede aplicar a cualquier gramática independiente del contexto (sin producciones nulas ni unitarias).

**Verdadero.** *El algoritmo de Earley es un analizador universal capaz de trabajar con cualquier GIC. Aunque la eliminación de producciones nulas y unitarias puede simplificar o mejorar la eficiencia de ciertas implementaciones, el algoritmo en su forma general soporta cualquier GIC, incluyendo gramáticas ambiguas.*

- 12) El conjunto de palabras  $\{a^n b^n c^i : i \leq n\}$  es independiente del contexto.

**Falso.** *Para verificar que el número de a's es igual al de b's ( $n = n$ ), necesitamos usar la*

pila. Una vez procesadas las  $b$ 's, la pila estaría vacía. No tendríamos forma de recordar el valor de  $n$  para verificar posteriormente que  $i \leq n$  con un autómata con pila estándar (sin contadores adicionales).

- 13) Si  $L_1$  y  $L_2$  son independientes del contexto, entonces  $L_1 - L_2$  es siempre independiente del contexto.

**Falso.** La resta de conjuntos se define como  $L_1 \cap L_2^c$ . Los LLC no son cerrados bajo complementación ni bajo intersección. Por tanto, la diferencia no asegura la preservación de la propiedad de independencia del contexto.

- 14) Hay lenguajes que no son independientes del contexto y si verifican la condición que aparece en el lema de bombeo para lenguajes independientes del contexto.

**Verdadero.** El lema de bombeo es una condición necesaria pero no suficiente. Existen lenguajes complejos (no libres de contexto) que, casualmente, cumplen la estructura de bombeo  $uv^iwx^i$  y sin ser generables por una GIC.

- 15) El conjunto de palabras  $\{u011u : u \in \{0,1\}^*\}$  es independiente del contexto.

**Falso.** Este lenguaje tiene la estructura de copia  $ww$  (con un separador fijo). Un autómata con pila no puede reconocer cadenas de la forma  $uu$  (o  $u...u$ ) de manera general, ya que lo primero que entra en la pila es lo último en salir, lo que facilita reconocer reversos  $uu^R$ , pero no copias directas.

- 16) El conjunto de palabras que contienen la subcadena 011 es independiente del contexto.

**Verdadero.** Este es un lenguaje regular (se puede construir una expresión regular o un autómata finito para "cadenas que contienen 011"). Dado que el conjunto de los lenguajes regulares es un subconjunto propio de los lenguajes independientes del contexto, la afirmación es cierta.

- 17) En el algoritmo de Cocke-Younger-Kasami calculamos los conjuntos  $V_{ij}$  que son las variables que generan la subcadena de la palabra de entrada que va desde el símbolo en la posición  $i$  al símbolo en la posición  $j$ .

**Verdadero.** Esta es la definición fundamental de la tabla dinámica en CYK. La celda  $V_{ij}$  (o  $X_{ij}$ , dependiendo de la notación: inicio-fin o inicio-longitud) almacena el conjunto de variables no terminales  $A$  tales que  $A \Rightarrow^* w_{i...j}$ .

- 18) Un lenguaje puede cumplir la negación de la condición que aparece en el lema de bombeo para lenguajes independientes del contexto y ser regular.

**Falso.** Si un lenguaje es regular, es automáticamente independiente del contexto. Por lo tanto, debe cumplir el lema de bombeo para lenguajes independientes del contexto. No puede cumplir su negación.

- 19) Si  $L_1$  y  $L_2$  son lenguajes independientes de contexto, entonces  $(L_1L_2 \cup L_1)^*$  es independiente de contexto.

**Verdadero.** Los lenguajes independientes del contexto son cerrados bajo las operaciones de unión ( $\cup$ ), concatenación ( $L_1L_2$ ) y clausura de Kleene (\*). Como la expresión está formada únicamente por estas operaciones sobre LLCs, el resultado es un LLC.

- 20) Existe un algoritmo para determinar si una palabra  $u$  tiene más de un árbol de derivación en una gramática independiente del contexto  $G$ .

**Verdadero.** Aunque la ambigüedad de la gramática en general es indecidible, verificar si una palabra específica  $u$  tiene más de un árbol es decidible. Se puede realizar ejecutando un parser (como Earley o CYK) y contando las trazas válidas de derivación para esa cadena.

- 21) La intersección de dos lenguajes independientes de contexto con un número finito de palabras produce siempre un lenguaje regular.
- Verdadero.** *Un lenguaje con un número finito de palabras es, por definición, un lenguaje regular. La intersección de dos lenguajes finitos da como resultado otro lenguaje finito, el cual sigue siendo regular (y por ende, libre de contexto).*
- 22) El complementario de un lenguaje con un número finito de palabras es siempre libre de contexto.
- Verdadero.** *Un lenguaje finito es regular. El complementario de un lenguaje regular es siempre regular. Dado que todo lenguaje regular es libre de contexto, su complementario es libre de contexto.*
- 23) Todo lenguaje aceptado por un autómata con pila por el criterio de estados finales cumple la condición que aparece en el lema de bombeo para lenguajes libres de contexto.
- Verdadero.** *Un lenguaje aceptado por un autómata con pila (ya sea por vaciado de pila o por estado final) es un Lenguaje Independiente del Contexto. Por teorema, todo LLC debe satisfacer las condiciones necesarias del Lema de Bombeo para LLC.*
- 24) Existe un algoritmo para determinar si un autómata con pila es determinista.
- Verdadero.** *Determinar si un autómata dado es determinista es una comprobación sintáctica sobre su función de transición  $\delta$ : verificar que para cada par (estado, símbolo de entrada/ $\epsilon$ , símbolo de pila) exista a lo sumo una transición posible. (Nota: Esto es distinto a decidir si un lenguaje es determinista, lo cual es más complejo).*
- 25) La demostración del lema de bombeo para lenguajes independientes del contexto se basa en que si las palabras superan una longitud determinada, entonces en el árbol de derivación debe de aparecer una variable como descendiente de ella misma.
- Verdadero.** *La prueba utiliza el principio del palomar aplicado a la altura del árbol de derivación. Si la cadena es suficientemente larga, la altura del árbol excede el número de variables no terminales  $|V|$ , forzando la repetición de al menos una variable en un camino desde la raíz a una hoja, lo que permite el "bombeo".*
- 26) La unión de dos lenguajes independientes contexto puede ser siempre aceptada por un autómata con pila.
- Verdadero.** *Los LLC son cerrados bajo la unión. La unión de dos LLC es un LLC, y por definición, todo LLC es aceptado por algún autómata con pila.*
- 27) El complementario de un lenguaje libre de contexto con una cantidad finita de palabras no tiene porque producir otro lenguaje libre de contexto.
- Falso.** *Un lenguaje finito es regular. Su complemento es regular. Todo regular es libre de contexto. Por tanto, el complemento de un lenguaje finito **siempre** es libre de contexto.*
- 28) El lema de bombeo para lenguajes libres de contexto es útil para demostrar que un lenguaje determinado no es libre de contexto.
- Verdadero.** *Es su uso principal. Se utiliza mediante demostración por contradicción (reducción al absurdo) para probar que un lenguaje no es LLC al mostrar que no cumple la condición necesaria de bombeo.*
- 29) La intersección de dos lenguajes independientes del contexto da lugar a un lenguaje aceptado por un autómata con pila determinista.
- Falso.** *La intersección de dos LLC ni siquiera garantiza ser un LLC, mucho menos un Lenguaje Independiente del Contexto Determinista.*

- 30) En el algoritmo de Cocke-Younger-Kasami si  $A \in V_{1,2}$  y  $B \in V_{3,2}$  y  $C \rightarrow AB$ , podemos deducir que  $C \in V_{1,4}$ .

**Verdadero.** *Interpretando la notación  $V_{i,l}$  donde  $i$  es el índice de inicio y  $l$  es la longitud:  $A$  cubre desde 1 con longitud 2 (posiciones 1,2).  $B$  cubre desde 3 con longitud 2 (posiciones 3,4). La concatenación  $AB$  cubre desde 1 con longitud  $2 + 2 = 4$ . Por tanto,  $C$  que genera  $AB$  debe estar en la celda  $V_{1,4}$ .*

- 31) No existe un algoritmo que nos diga si son iguales los lenguajes generados por dos gramáticas independientes del contexto  $G_1$  y  $G_2$ .

**Verdadero.** *El problema de la equivalencia de gramáticas libres de contexto es indecidible.*

- 32) La intersección de dos lenguajes infinitos da lugar a un lenguaje independiente del contexto.

**Falso.** *El hecho de que sean infinitos no cambia la propiedad de clausura. La intersección de LLCs (finitos o infinitos) no es cerrada. Ejemplo: intersección de  $\{a^n b^n c^*\}$  y  $\{a^* b^n c^n\}$  es  $\{a^n b^n c^n\}$ , que no es LLC.*

- 33) Si  $L_1$  y  $L_2$  son independientes del contexto, no podemos asegurar que  $L_1 \cap L_2$  también lo sea.

**Verdadero.** *Correcto. Al no ser una operación cerrada, la intersección podría resultar en un LLC o en un lenguaje que no lo es. No se puede "asegurar" que el resultado sea LLC en todos los casos.*

- 34) Si un lenguaje satisface la condición necesaria del lema de bombeo para lenguajes regulares, entonces también tiene que satisfacer la condición necesaria del lema de bombeo para lenguajes independientes del contexto.

**Verdadero.** *El bombeo regular es un caso particular del bombeo libre de contexto (donde una de las dos partes bombeables es vacía, o visto de otro modo, la estructura lineal es un caso simple de estructura de árbol). Si  $w = xyz$  se puede bombear como  $xy^iz$ , esto se ajusta a la forma  $uv^iwx^iy$  (por ejemplo, haciendo  $u = x, v = y, w = z$  y  $x, y$  vacíos en la notación de LLC, respetando las longitudes). Formalmente, la clase de lenguajes "bombeables linealmente" está contenida en la clase de lenguajes "bombeables estructuralmente".*



# **Parte II**

## **Exámenes**



# Capítulo 1

---

## Parcial I

---

### 1.1 Examen I

#### Datos Examen

- 2021-2022

**Ejercicio 1.1.1.** Encontrar una gramática que genere el lenguaje:  $L_1 = \{a^n b^m c^k \in \{a, b, c\}^* \mid n, m, k \in \mathbb{N} \cup \{0\}, |n - m| = k\}$

#### Solución 1.1.1. 1. Análisis del Lenguaje

La condición principal del lenguaje es  $|n - m| = k$ . Esta ecuación de valor absoluto se puede dividir en dos casos mutuamente excluyentes (excepto cuando  $n = m$  y  $k = 0$ ):

- **Caso 1:**  $n \geq m$

En esta situación, el valor absoluto  $|n - m|$  es  $n - m$ . La condición se convierte en:  $n - m = k$ . Podemos despejar  $n$ :  $n = m + k$ . Las cadenas de este sub-lenguaje, que llamaremos  $L_A$ , tienen la forma:  $a^{m+k} b^m c^k$ .

- **Caso 2:**  $n < m$

En esta situación, el valor absoluto  $|n - m|$  es  $-(n - m) = m - n$ . La condición se convierte en:  $m - n = k$ . Podemos despejar  $m$ :  $m = n + k$ . Las cadenas de este sub-lenguaje, que llamaremos  $L_B$ , tienen la forma:  $a^n b^{n+k} c^k$ .

El lenguaje  $L_1$  es la unión de estos dos sub-lenguajes:  $L_1 = L_A \cup L_B$ .

Dado que los lenguajes libres de contexto (generados por gramáticas) son cerrados bajo la operación de unión, podemos diseñar una gramática para  $L_A$  y otra para  $L_B$ , y luego combinarlas usando un nuevo símbolo inicial  $S$ .

La regla inicial será:  $S \rightarrow S_A \mid S_B$  Donde  $S_A$  es el símbolo inicial para la gramática de  $L_A$  y  $S_B$  es el símbolo inicial para la gramática de  $L_B$ .

#### 2. Gramática para $L_A$ (Caso 1: $n \geq m$ )

Buscamos generar el lenguaje  $L_A = \{a^{m+k} b^m c^k \mid m, k \geq 0\}$ . Podemos reescribir la forma de la cadena como:  $a^k a^m b^m c^k$ .

Esta estructura nos muestra dos relaciones de conteo anidadas:

- 1) Una relación externa:  $k$   $a$ 's al principio se corresponden con  $k$   $c$ 's al final ( $a^k \dots c^k$ ).
- 2) Una relación interna:  $m$   $a$ 's se corresponden con  $m$   $b$ 's ( $a^m b^m$ ).

Podemos construir la gramática de la siguiente manera:

- Usamos  $S_A$  para generar la parte externa ( $a^k \dots c^k$ ). Cada vez que añadimos una  $a$ , también añadimos una  $c$ .

- Cuando dejamos de generar  $a$ 's y  $c$ 's, pasamos a un nuevo símbolo (llamémoslo  $M$ ) que generará la parte interna ( $a^m b^m$ ).

Las producciones para  $L_A$  son:

$$\begin{aligned} S_A &\rightarrow aS_Ac \mid M \\ M &\rightarrow aMb \mid \epsilon \end{aligned}$$

**Ejemplo de derivación para  $a^3b^1c^2$  ( $m = 1, k = 2$ ):**

$$S_A \rightarrow aS_Ac \rightarrow a(aS_Ac)c \rightarrow aaS_Acc \rightarrow aaMcc \rightarrow aa(aMb)cc \rightarrow aa(aeb)cc = a^3b^1c^2$$

### 3. Gramática para $L_B$ (Caso 2: $n < m$ )

Buscamos generar el lenguaje  $L_B = \{a^n b^{n+k} c^k \mid n, k \geq 0\}$ . Podemos reescribir la forma de la cadena como:  $a^n b^n b^k c^k$ .

Esta estructura nos muestra dos relaciones de conteo secuenciales (una después de la otra):

- 1) Una primera parte:  $n$   $a$ 's seguidas de  $n$   $b$ 's ( $a^n b^n$ ).
- 2) Una segunda parte:  $k$   $b$ 's seguidas de  $k$   $c$ 's ( $b^k c^k$ ).

Este lenguaje  $L_B$  es la concatenación de dos lenguajes libres de contexto más simples.

- Podemos usar un símbolo  $S_B$  que se derive en  $YZ$ .
- $Y$  generará la primera parte:  $\{a^n b^n \mid n \geq 0\}$ .
- $Z$  generará la segunda parte:  $\{b^k c^k \mid k \geq 0\}$ .

Las producciones para  $L_B$  son:

$$\begin{aligned} S_B &\rightarrow YZ \\ Y &\rightarrow aYb \mid \epsilon \\ Z &\rightarrow bZc \mid \epsilon \end{aligned}$$

**Ejemplo de derivación para  $a^1b^3c^2$  ( $n = 1, k = 2$ ):**

$$S_B \rightarrow YZ \rightarrow (aYb)Z \rightarrow (aeb)Z \rightarrow abZ \rightarrow ab(bZc) \rightarrow ab(b(bb\epsilon cc)) = a^1b^3c^2$$

### 4. Gramática Final (Unificada)

Ahora, combinamos todas las reglas.

- Símbolo inicial:  $S$
- Variables (No terminales):  $\{S, S_A, S_B, M, Y, Z\}$
- Terminales:  $\{a, b, c\}$

Producciones ( $P$ ):

$$\begin{aligned} S &\rightarrow S_A \mid S_B \\ S_A &\rightarrow aS_Ac \mid M \\ S_B &\rightarrow MZ \\ M &\rightarrow aMb \mid \epsilon \\ Z &\rightarrow bZc \mid \epsilon \end{aligned}$$

**Ejercicio 1.1.2.** Dar gramáticas que acepten los siguientes lenguajes:

- 1)  $L_1 = \{u \in \{0,1\}^* \mid u^{-1} = \bar{u}\}$ , donde  $u^{-1}$  representa el complemento de  $u$ , es decir, cambiando 0's por 1's y viceversa.
- 2)  $L_2 = \{a^n b^m \mid n, m \in \mathbb{N} \cup \{0\}, 3m \geq n \geq 2m\}$

*Solución 1.1.2.* Esta es una resolución detallada del ejercicio.

### Análisis del Lenguaje $L_1$

El lenguaje se define como  $L_1 = \{u \in \{0,1\}^* \mid u^{-1} = \bar{u}\}$ . Vamos a desglosar las condiciones:

- $u \in \{0,1\}^*$ :  $u$  es una cadena (string) formada por 0s y 1s, de cualquier longitud, incluida la longitud 0 (la cadena vacía,  $\epsilon$ ).
- $u^{-1}$ : Es la **reversa** de la cadena  $u$ . Si  $u = w_1 w_2 \dots w_n$ , entonces  $u^{-1} = w_n \dots w_2 w_1$ .
- $\bar{u}$ : Es el **complemento** de la cadena  $u$ . Si  $u = w_1 w_2 \dots w_n$ , entonces  $\bar{u} = \overline{w_1 w_2} \dots \overline{w_n}$ , donde  $\overline{0} = 1$  y  $\overline{1} = 0$ .

La condición  $u^{-1} = \bar{u}$  significa que la cadena reversa de  $u$  debe ser idéntica a la cadena complemento de  $u$ .

### Análisis de la Longitud de las Cadenas

Veamos qué pasa con cadenas de diferentes longitudes:

- **Longitud 0** ( $u = \epsilon$ ):

$$u^{-1} = \epsilon, \quad \bar{u} = \epsilon$$

$u^{-1} = \bar{u}$  se cumple. Por lo tanto,  $\epsilon \in L_1$ .

- **Longitud 1** ( $u = w_1$ ): La condición es  $w_1 = \overline{w_1}$ .

- Si  $w_1 = 0$ , la condición es  $0 = \overline{0} = 1$  (Falso).
- Si  $w_1 = 1$ , la condición es  $1 = \overline{1} = 0$  (Falso).

Ninguna cadena de longitud 1 pertenece a  $L_1$ .

- **Longitud 2** ( $u = w_1 w_2$ ): La condición es  $w_2 = \overline{w_1}$  y  $w_1 = \overline{w_2}$ .

- Si  $w_1 = 0$ ,  $w_2 = \overline{0} = 1$ . La cadena es  $u = 01$ .  
(Check:  $u^{-1} = 10$ ,  $\bar{u} = \overline{01} = 10$ . Se cumple).
- Si  $w_1 = 1$ ,  $w_2 = \overline{1} = 0$ . La cadena es  $u = 10$ .  
(Check:  $u^{-1} = 01$ ,  $\bar{u} = \overline{10} = 01$ . Se cumple).

Las cadenas 01 y 10 pertenecen a  $L_1$ .

- **Longitud 3** ( $u = w_1 w_2 w_3$ ): Las condiciones son:  $w_3 = \overline{w_1}$  y  $w_2 = \overline{w_2}$  y  $w_1 = \overline{w_3}$ .

- La condición central,  $w_2 = \overline{w_2}$ , es la misma que vimos para longitud 1. Es imposible.

Ninguna cadena de longitud 3 pertenece a  $L_1$ .

**Conclusión:** El lenguaje  $L_1$  solo contiene cadenas de longitud par.

### Diseño de la Gramática

Necesitamos una gramática que genere cadenas desde "fuera hacia adentro", asegurando que los pares de caracteres externos (primero y último) sean complementarios.

Sea  $S$  nuestro símbolo inicial.

- **Caso Base:** La cadena más corta (y la única de longitud impar, que es 0) es la cadena vacía  $\epsilon$ . Esta debe ser nuestra regla de terminación.

$$S \rightarrow \epsilon$$

- **Paso Recursivo:** Si tenemos una cadena  $u$  que ya está en  $L_1$  (es decir,  $u$  es generable desde  $S$ ), podemos formar una cadena nueva y más larga poniéndole caracteres complementarios en los extremos.

- Si  $u \in L_1$ , entonces  $0u1$  también debe estar en  $L_1$ .
- Del mismo modo, si  $u \in L_1$ , entonces  $1u0$  también debe estar en  $L_1$ .

Esto nos da las reglas:

$$S \rightarrow 0S1 \quad \text{y} \quad S \rightarrow 1S0$$

### *Gramática Final*

Combinando las reglas obtenidas, la gramática  $G = (V, T, P, S)$  es:

- **Variables (No terminales):**  $\{S\}$
- **Terminales:**  $\{0, 1\}$
- **Símbolo Inicial:**  $S$
- **Producciones:**

$$S \rightarrow 0S1$$

$$S \rightarrow 1S0$$

$$S \rightarrow \epsilon$$

### *Ejemplos de Derivación*

- **Para generar  $\epsilon$ :**

$$S \rightarrow \epsilon$$

- **Para generar 01:**

$$S \rightarrow 0S1 \rightarrow 0\epsilon 1 = 01$$

- **Para generar 10:**

$$S \rightarrow 1S0 \rightarrow 1\epsilon 0 = 10$$

- **Para generar 0011:**

$$S \rightarrow 0S1 \rightarrow 0(0S1)1 \rightarrow 00\epsilon 11 = 0011$$

- **Para generar 1100:**

$$S \rightarrow 1S0 \rightarrow 1(1S0)0 \rightarrow 11\epsilon 00 = 1100$$

## 1.2 Examen II

### Datos Examen

- 2025-2026

**Ejercicio 1.2.1.** Dado el lenguaje  $L = \{w \in \{a,b\}^* \mid \text{los cambios de } a \text{ a } b \text{ solo ocurren cuando hay un número par de } a's, \text{ y los cambios de } b \text{ a } a \text{ solo ocurren cuando hay un número impar de } b's\}$ :

- 1) Diseñar el Autómata Finito Determinista (AFD) que reconozca el lenguaje  $L$ .

Para ello debemos de proporcionar los estados necesarios, así como la función de transición, el estado inicial y los estados de aceptación. Para la resolución del ejercicio, nos serviremos del producto cartesiano de las posibilidades que estamos barajando.

Estados:

- $q_0$ : estado inicial, número par de  $a$ 's y de  $b$ 's.
- $q_1$ : número impar de  $a$ 's y par de  $b$ 's.
- $q_2$ : número impar de  $a$ 's e impar de  $b$ 's.
- $q_3$ : número par de  $a$ 's e impar de  $b$ 's.

Funciones de transición:

- $\delta(q_0, a) = q_1$
- $\delta(q_0, b) = q_3$
- $\delta(q_1, a) = q_0$
- $\delta(q_1, b) = E$
- $\delta(q_2, a) = q_3$
- $\delta(q_2, b) = E$
- $\delta(q_3, a) = q_2$
- $\delta(q_3, b) = q_0$

Debemos de añadir el estado de Error.

Los estados de aceptación serán aquellos en los que se cumplan las condiciones del lenguaje, es decir, todos, debido a que los cambios que no se puedan producir van a error, por ende, el único que no puede ser final es el estado de error.

Por otro lado, podríamos pensar que si en  $q_0$  recibo una  $a$ , como el estado de las  $b$ 's, es par no se puede producir el cambio, pero no debemos de confundirnos, ya que no había ninguna  $b$  para cambiar  $a$  a, por lo que no hay problema en recibir una  $a$  en ese estado.

Por lo tanto el AFD queda definido como el de la figura 1.1.

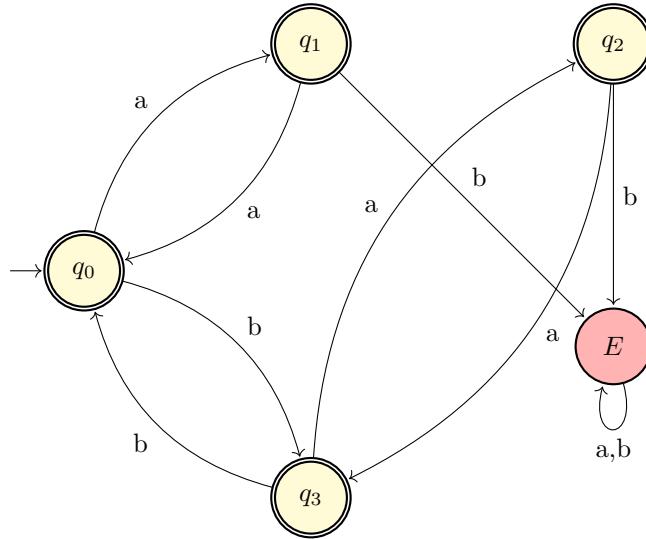


Figura 1.1: Autómata Finito Determinista (AFD) para el lenguaje  $L$ .

Tras esta resolución, podemos pensar que es correcta, pero no estamos teniendo en cuenta que no estamos tratando el punto de partida, es decir, si ha recibido una a o un b, parte fundamental, ya que si vemos que pasa de  $q_0$  a  $q_3$  y vuelve, tiene un número par de b's y puede cambiar a a, esto no debería de poderse, de manera que debemos de añadir más estados para tener en cuenta el primer símbolo que recibimos.

Por lo tanto, los estados quedarían de la siguiente manera:

- $q_{00}$ : estado inicial, número par de a's y de b's, primer símbolo  $a$ .
- $q_{01}$ : número impar de a's y par de b's, primer símbolo  $a$ .
- $q_{02}$ : número impar de a's e impar de b's, primer símbolo  $a$ .
- $q_{03}$ : número par de a's e impar de b's, primer símbolo  $a$ .
- $q_{10}$ : estado inicial, número par de a's y de b's, primer símbolo  $b$ .
- $q_{11}$ : número impar de a's y par de b's, primer símbolo  $b$ .
- $q_{12}$ : número impar de a's e impar de b's, primer símbolo  $b$ .
- $q_{13}$ : número par de a's e impar de b's, primer símbolo  $b$ .

Funciones de transición:

- $\delta(q_{00}, a) = q_{01}$
- $\delta(q_{00}, b) = q_{10}$
- $\delta(q_{01}, a) = q_{00}$
- $\delta(q_{01}, b) = q_{11}$
- $\delta(q_{02}, a) = q_{03}$
- $\delta(q_{02}, b) = q_{12}$
- $\delta(q_{03}, a) = q_{02}$
- $\delta(q_{03}, b) = q_{13}$
- $\delta(q_{10}, a) = q_{11}$
- $\delta(q_{10}, b) = q_{00}$
- $\delta(q_{11}, a) = q_{10}$
- $\delta(q_{11}, b) = q_{12}$
- $\delta(q_{12}, a) = q_{13}$
- $\delta(q_{12}, b) = q_{02}$
- $\delta(q_{13}, a) = q_{12}$
- $\delta(q_{13}, b) = q_{03}$

Los estados de aceptación serán aquellos en los que se cumplan las condiciones del lenguaje,

es decir, todos, debido a que los cambios que no se puedan producir van a error, por ende, el único que no puede ser final es el estado de error.

Por lo tanto el AFD queda definido mucho más complejo.

Otra forma es verlo de manera directa. Para ello vemos el grafo de la figura 1.2.

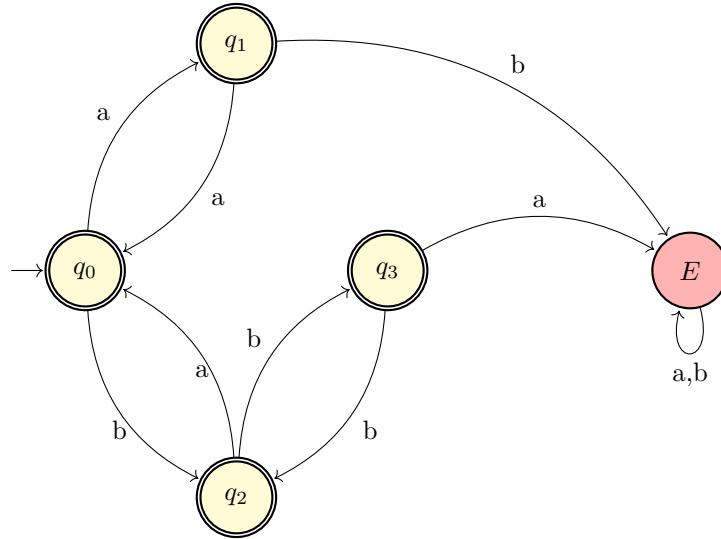


Figura 1.2: Autómata Finito Determinista (AFD) para el lenguaje  $L$ . Versión simplificada de manera directa

- 2) Dar la expresión regular que describe el lenguaje  $L$ .

Aplicamos de antemano que  $E = \emptyset$

Por lo tanto, la ER que describe el lenguaje  $L$  es la siguiente:

$$\begin{aligned} q_0 &= aq_1 + bq_2 + \epsilon \\ q_1 &= aq_0 + Eb + \epsilon \\ q_2 &= bq_3 + aq_0 + \epsilon \\ q_3 &= bq_2 + Ea + \epsilon \end{aligned}$$

Resolviendo el sistemas nos queda:

$$(aa + abaaa)^* + b(bb + ab)^* + abaa + ba + bb + \epsilon$$

- 3) Proporcionar la gramática lineal por la derecha que genera el lenguaje  $L$  (Se ha hecho con el AFD de la figura 1.1).

Para construir la gramática lineal por la derecha, tomamos como base el AFD diseñado en el apartado (a). Cada estado del AFD se convierte en un símbolo no terminal de la gramática, y las transiciones entre estados se convierten en producciones. De manera que nos quedan las siguientes producciones:

$$\begin{aligned} q_0 &\rightarrow aq_1 \mid bq_3 \mid \epsilon \\ q_1 &\rightarrow aq_0 \mid \epsilon \\ q_2 &\rightarrow aq_3 \mid \epsilon \\ q_3 &\rightarrow bq_0 \mid aq_2 \mid \epsilon \end{aligned}$$

*Nota 1.1.* Si se nos pidiera la gramática lineal por la izquierda, deberíamos invertir el AFD y construir la gramática a partir de ese AFD invertido, acto seguido invertimos el orden de las producciones para conseguir que sea lineal por la izquierda.

**Ejercicio 1.2.2.** Dar las gramáticas que generan los siguientes lenguajes:

- 1) El lenguaje de todos los números en binario múltiplos de 4.

La resolución de este ejercicio es algorítmica y para ello vamos a explicarlo de cara a conseguir los múltiplos de cualquiera  $k \mid k \in \mathbb{N}$ .

Para resolver este problema, utilizamos el concepto de aritmética modular para construir un autómata finito determinista (AFD) que reconozca números binarios múltiplos de  $k$ . A continuación, se describe el procedimiento general y se aplica específicamente para  $k = 4$ .

**Concepto central:** Un número  $N$  es múltiplo de  $k$  si y solo si  $N \pmod{k} = 0$ . El AFD leerá el número binario bit a bit y usará sus estados para representar el resto de la división por  $k$  del número leído hasta ese momento. Si al finalizar la lectura el autómata está en el estado que representa el resto 0, la cadena será aceptada.

**Construcción del AFD:**

- **Estados ( $Q$ ):** Se necesitan  $k$  estados, donde cada estado  $q_r$  representa el resto  $r$  al dividir el número leído hasta el momento por  $k$ . Es decir,  $Q = \{q_0, q_1, \dots, q_{k-1}\}$ .
- **Estado inicial ( $q_0$ ):** El estado inicial es  $q_0$ , que representa el resto 0 antes de leer cualquier bit.
- **Estados finales ( $F$ ):** El único estado final es  $q_0$ , ya que aceptamos la cadena solo si el resto final es 0.
- **Función de transición ( $\delta$ ):** La transición entre estados se calcula como:

$$\delta(q_r, b) = q_s, \quad \text{donde } s = (2r + b) \pmod{k}$$

Aquí,  $r$  es el resto actual,  $b$  es el bit leído (0 o 1), y  $s$  es el nuevo resto.

**Ejemplo: Múltiplos de 4 ( $k = 4$ )**

- **Estados:**  $Q = \{q_0, q_1, q_2, q_3\}$
- **Estado inicial:**  $q_0$
- **Estado final:**  $F = \{q_0\}$
- **Transiciones:** Calculamos las transiciones usando la fórmula  $s = (2r + b) \pmod{4}$ .

Estado actual ( $q_r$ )	Leer $b = 0$	$\rightarrow s = (2r + 0) \pmod{4}$	Leer $b = 1$	$\rightarrow s = (2r + 1) \pmod{4}$
$q_0$ (resto 0)	$q_0$	$(0 + 0) \pmod{4} = 0$	$q_1$	$(0 + 1) \pmod{4} = 1$
$q_1$ (resto 1)	$q_2$	$(2 + 0) \pmod{4} = 2$	$q_3$	$(2 + 1) \pmod{4} = 3$
$q_2$ (resto 2)	$q_0$	$(4 + 0) \pmod{4} = 0$	$q_1$	$(4 + 1) \pmod{4} = 1$
$q_3$ (resto 3)	$q_2$	$(6 + 0) \pmod{4} = 2$	$q_3$	$(6 + 1) \pmod{4} = 3$

Cuadro 1.1: *Transiciones del AFD para múltiplos de 4.*

**Observación:** Este AFD acepta cadenas que terminan en 00 (o son 0). Esto se deduce automáticamente del método general, ya que para llegar al estado  $q_0$  (final) después de leer una cadena, el número debe ser divisible por 4.

**Ejemplo de derivación:**

- Cadena 1100 (valor 12, múltiplo de 4):

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_3 \xrightarrow{0} q_2 \xrightarrow{0} q_0 \quad (\text{Aceptada})$$

- Cadena 101 (valor 5, no múltiplo de 4):

$$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_1 \quad (\text{Rechazada})$$

**Ejemplo 2: Múltiplos de 3 ( $k = 3$ )**

- **Estados:**  $Q = \{q_0, q_1, q_2\}$
- **Estado inicial:**  $q_0$
- **Estado final:**  $F = \{q_0\}$
- **Transiciones:** Calculamos las transiciones usando la fórmula  $s = (2r + b) \pmod{3}$ .

Estado actual ( $q_r$ )	Leer $b = 0$	$\rightarrow s = (2r + 0) \pmod{3}$	Leer $b = 1$	$\rightarrow s = (2r + 1) \pmod{3}$
$q_0$ (resto 0)	$q_0$	$(0 + 0) \pmod{3} = 0$	$q_1$	$(0 + 1) \pmod{3} = 1$
$q_1$ (resto 1)	$q_2$	$(2 + 0) \pmod{3} = 2$	$q_0$	$(2 + 1) \pmod{3} = 0$
$q_2$ (resto 2)	$q_1$	$(4 + 0) \pmod{3} = 1$	$q_2$	$(4 + 1) \pmod{3} = 2$

Cuadro 1.2: *Transiciones del AFD para múltiplos de 3.*

**Prueba:**

- Cadena 110 (valor 6, múltiplo de 3):

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \quad (\text{Aceptada})$$

- Cadena 101 (valor 5, no múltiplo de 3):

$$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_2 \quad (\text{Rechazada})$$

Retomando el caso de los múltiplos de 4, la gramática lineal por la derecha correspondiente se calcularía mediante las funciones de transición del AFD diseñado, resultando en las siguientes producciones:

$$\begin{aligned} q_0 &\rightarrow 0q_0 \mid 1q_1 \mid \epsilon \\ q_1 &\rightarrow 0q_2 \mid 1q_3 \\ q_2 &\rightarrow 0q_0 \mid 1q_1 \\ q_3 &\rightarrow 0q_2 \mid 1q_3 \end{aligned}$$

De nuevo debemos de tener en cuenta la *Nota* anterior sobre la gramática lineal por la izquierda.

- 2) El lenguaje de todas las palabras pertenecientes a  $\{a, b, c\}$  tales que el número de  $a$ 's es par si el número de  $c$ 's es par.

Para resolver este problema, definimos cuatro estados basados en la paridad del número de  $a$ 's y  $c$ 's leídos hasta el momento:

- $q_{ee}$ : número par de  $a$ 's y número par de  $c$ 's.
- $q_{eo}$ : número par de  $a$ 's y número impar de  $c$ 's.
- $q_{oe}$ : número impar de  $a$ 's y número par de  $c$ 's.
- $q_{oo}$ : número impar de  $a$ 's y número impar de  $c$ 's.

De manera análoga a como venimos resolviendo los ejercicios anteriores, definimos las

transiciones entre estos estados basándose en la lectura de los caracteres  $a$ ,  $b$  y  $c$ :

$$\begin{array}{lll} \delta(q_{ee}, a) = q_{oe} & \delta(q_{ee}, b) = q_{ee} & \delta(q_{ee}, c) = q_{eo} \\ \delta(q_{eo}, a) = q_{oo} & \delta(q_{eo}, b) = q_{eo} & \delta(q_{eo}, c) = q_{ee} \\ \delta(q_{oe}, a) = q_{ee} & \delta(q_{oe}, b) = q_{oe} & \delta(q_{oe}, c) = q_{oo} \\ \delta(q_{oo}, a) = q_{eo} & \delta(q_{oo}, b) = q_{oo} & \delta(q_{oo}, c) = q_{oe} \end{array}$$

Podemos ver que la columna correspondiente a la lectura de  $b$ 's no afecta al estado, ya que los  $b$ 's no influyen en las condiciones del lenguaje.

Debemos de definir el estado inicial y los estados de aceptación:

- Estado inicial:  $q_{ee}$  (antes de leer cualquier símbolo, tenemos 0  $a$ 's y 0  $c$ 's, ambos pares).
- Estados de aceptación:  $q_{ee}$ ,  $q_{eo}$  y  $q_{oo}$  (estos estados cumplen la condición del lenguaje: si el número de  $c$ 's es par, el número de  $a$ 's también es par). El único que no es válido es  $q_{oe}$ , donde hay un número impar de  $a$ 's y un número par de  $c$ 's, lo cual viola la condición del lenguaje.

Dibujamos el AFD correspondiente, el cual es el de la figura 1.3.

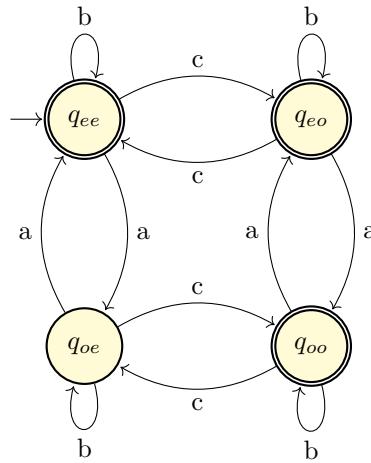


Figura 1.3: Autómata Finito Determinista (AFD) para el lenguaje del ejercicio 2.

A partir de este AFD, podemos construir la gramática lineal por la derecha correspondiente<sup>1</sup>:

$$\begin{aligned} q_{ee} &\rightarrow aq_{oe} \mid bq_{ee} \mid cq_{eo} \mid \epsilon \\ q_{eo} &\rightarrow aq_{oo} \mid bq_{eo} \mid cq_{ee} \\ q_{oe} &\rightarrow aq_{ee} \mid bq_{oe} \mid cq_{oo} \\ q_{oo} &\rightarrow aq_{eo} \mid bq_{oo} \mid cq_{oe} \end{aligned}$$

- 3) El lenguaje de todos los palíndromos pertenecientes a  $\{0,1\}^*$  en los que el número de 1's es igual a  $3k + 1$  para algún  $k \in \mathbb{N}$ .

Lo que nos pide el ejercicio son las palabras de 1's y 0's que son palíndromos y con 1's en número de la forma  $3k + 1$ , como pueden ser 1, 4, 7, 10, ... Debemos de saber que nos pide exactamente. Lo que nos pide es los múltiplos de 3 más 1. Podemos verlo de la siguiente manera.

<sup>1</sup>No es necesario dibujar el AFD, aquí se incluye para hacer el ejercicio más completo, con tener las funciones de transición podemos conseguirla.

- $S_0$ : constructor de palíndromos con número de 1's múltiplo de 3, por lo tanto el resto es 0.
- $S_1$ : constructor de palíndromos con número de 1's de la forma  $3k + 1$ , por lo tanto el resto es 1.
- $S_2$ : constructor de palíndromos con número de 1's de la forma  $3k + 2$ , por lo tanto el resto es 2.

Partimos del punto de que los 0's no afectan al conteo de 1's.

- Para construir  $S_0$  en cuanto al número de 1's:

$$S_0 \rightarrow 1S_?1$$

Añadimos 2 1's y queremos que:

$$(\text{Resto del centro} + 2) \pmod{3} = 0$$

Vemos que funciona con  $S_1$ :

$$S_0 \rightarrow 1S_11$$

- Para construir  $S_1$  en cuanto al número de 1's:

$$S_1 \rightarrow 1S_?1$$

Añadimos 2 1's y queremos que:

$$(\text{Resto del centro} + 2) \pmod{3} = 1$$

Vemos que funciona con  $S_2$ :

$$S_1 \rightarrow 1S_21$$

- Para construir  $S_2$  en cuanto al número de 1's:

$$S_2 \rightarrow 1S_?1$$

Añadimos 2 1's y queremos que:

$$(\text{Resto del centro} + 2) \pmod{3} = 2$$

Vemos que funciona con  $S_0$ :

$$S_2 \rightarrow 1S_01$$

Nos queda definir que palabras son el centro:

- $\varepsilon$  tiene 0 1's por lo que nos lleva a  $S_0 \Rightarrow S_0 \rightarrow \varepsilon$
- 0 tiene 0 1's por lo que nos lleva a  $S_0 \Rightarrow S_0 \rightarrow 0$
- 1 tiene 1 1's por lo que nos lleva a  $S_1 \Rightarrow S_1 \rightarrow 1$

**IMPORTANTE:** Debemos de pensar que estábamos generando los palíndromos con 1's múltiplo de 3, por lo tanto para completar lo que el ejercicio nos pide, debemos de añadir las cadenas de manera que siempre se añada un 1 cumpliendo la condición de palíndromo. Asimismo, debemos de eliminar la cadena vacía de  $S_0$  ya que no cumple la condición del número de 1's.

Por lo tanto, podemos concluir con que las producciones de la gramática son:

$$\begin{aligned}S_0 &\rightarrow 1S_11 \mid 0S_00 \mid 1 \\S_1 &\rightarrow 1S_21 \mid 0S_10 \mid 101 \mid 11 \\S_2 &\rightarrow 1S_01 \mid 0S_20 \mid 10101 \mid 111\end{aligned}$$

---

# Bibliografía

---

- [1] Ismael Sallami Moreno, **Estudiante del Doble Grado en Ingeniería Informática + ADE**, Universidad de Granada.