UNIVERSIDAD DE GRANADA

Modelos de Computación

Temario

Ismael Sallami Moreno

Recursos Ingeniería Informática y Ade

Licencia

Modelos de Computación

Ismael Sallami Moreno

Índice general

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 çomputabilidad". 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 `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 Σ, 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: $\Sigma = \{a, b, c, \ldots, z\}$.
- 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 ϵ (o a veces λ), 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 ϵ 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 çadenas del lenguaje.º "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 ϵ 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 $\alpha \to \beta$, donde $\alpha, \beta \in (V \cup T)^*$ y α contiene al menos una variable.
- $S \in V$ es el **símbolo inicial** o axioma.

Una palabra β se **deriva** de α en un paso ($\alpha \Rightarrow \beta$) si existe una regla $\gamma \to \phi \in P$ y α 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 $G = (\{S\}, \{a, b\}, \{S \to aSb, S \to \epsilon\}, S)$. 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 $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$, con $A \in V$, $\alpha_1, \alpha_2, \beta \in (V \cup T)^*$ y $\beta \neq \epsilon$. Se permite la regla $S \to \epsilon$ 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 $A \to \alpha$, 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 $A \to uB$ o $A \to u$, 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.

Capítulo 2

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 (δ), 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).

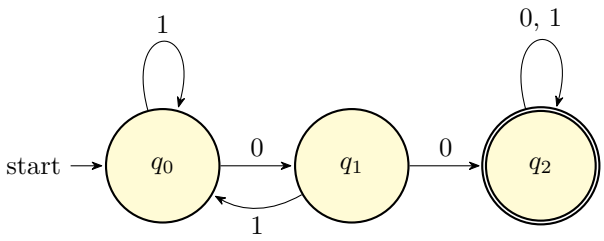


Figura 2.1: *Autómata Finito Determinista que acepta el lenguaje de palabras que contienen la subcadena 00.*

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 ε:

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

  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, $L(M) = \{u_1 00 u_2 \in \{0,1\}^* \mid u_1, u_2 \in \{0,1\}^*\}$.

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 unívocamente 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.
- δ: Es la función de transición. Para un AFD, esta es una aplicación $\delta : Q \times A \to Q$. La función δ 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.

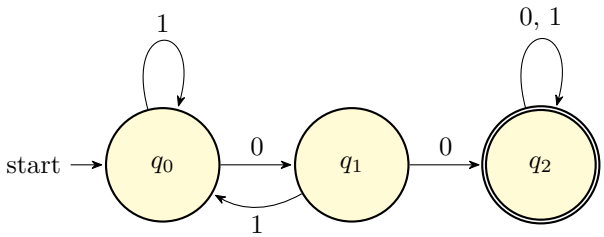


Figura 2.2: *Diagrama de Transición para el AFD que acepta* 00.

Tablas de Transición

Las tablas de transición son una representación tabular y analítica de la función δ, 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
$\star q_2$	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 $C_0, \ldots, C_n$ tal que $C_0 \vdash C_1 \vdash \cdots \vdash C_n$ se denota como $(q, u) \vdash^* (p, v)$.

Función de Transición Extendida (δ^*)

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 δ^*). La función δ^* 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 δ^*). Si consideramos el AFD del ejemplo anterior (AFD para subcadena 00), el cálculo de δ^* para la palabra `modern`100 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-ϵ 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: $\delta : Q \times A \to \mathcal{P}(Q)$

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.
- δ es la función de transición que mapea un par (estado, símbolo) a un conjunto de estados:

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

  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:

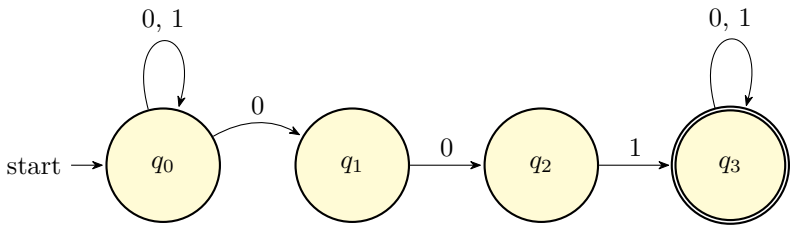


Figura 2.3: *AFND para el lenguaje que contiene la subcadena 001. Desde q_0 con el símbolo 0 se puede ir a q_0 o a q_1, ilustrando la no determinismo.*

2.3.2 AFND con Transiciones Nulas (AFN-ϵ)

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-ϵ).

Inclusión del símbolo de palabra vacía (ϵ) en el alfabeto para las transiciones

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

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

Las transiciones ϵ 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 ϵ 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 ϵ 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 ϵ-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 δ_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-ϵ → AFND

La eliminación de las transiciones ϵ 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-ϵ a AFD se realiza en un solo paso, combinando este proceso con el método de los subconjuntos).

Cálculo de la ϵ-Clausura para la eliminación de transiciones nulas

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

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

- La ϵ-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 ϵ-transición desde q hasta p.

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

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

La ϵ-Clausura es esencial para definir la función de transición del autómata resultante M' (que será un AFND sin ϵ-transiciones o un AFD). La nueva función de transición, δ', se calcula utilizando la ϵ-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-ϵ, la transición a partir de un símbolo a en el autómata equivalente se calcula mediante la función δ^* extendida:

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

Esta δ^* (aplicada al AFN-ϵ) 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 (AFD $\leftrightarrow$ AFND $\leftrightarrow$ AFN-ϵ) 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-ϵ) 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 $/\mathbf{0}$) es una ER que denota el lenguaje vacío, $L(\emptyset) = \emptyset$.
2) La expresión ϵ es una ER que denota el lenguaje que contiene únicamente la palabra vacía, $L(\epsilon) = \{\varepsilon\}$.
3) Para cada símbolo $a \in A$, la expresión $\mathbf{a}$ es una ER que denota el lenguaje que contiene únicamente la palabra a, $L(\mathbf{a}) = \{a\}$.

4) Si r_1 y r_2 son ERs, entonces la Unión (r_1+r_2), la Concatenación (r_1r_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_1r_2 denota el conjunto de palabras formadas al concatenar cualquier palabra de L_1 con cualquier palabra de L_2: $L(r_1r_2) = \{uv \mid u \in L_1, v \in L_2\}$. La concatenación es asociativa y ϵ 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 $L(r)^0 = \{\varepsilon\}$. 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 $Q = \{q_1, q_2, \ldots, q_n\}$.

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:

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

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 ϵ en la suma.

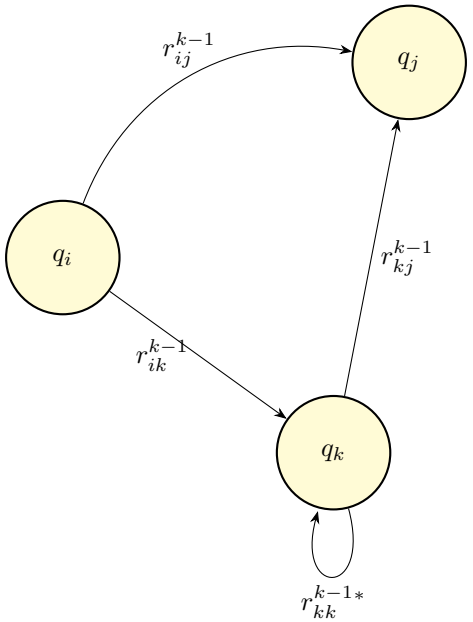


Figura 2.4: *Representación esquemática de la fórmula de recurrencia r_{ij}^k.*

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^n_{1j} (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^n_{1j} para cada estado final $q_j \in F$.

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

$$r_{L(M)} = r^n_{1j_1} + r^n_{1j_2} + \cdots + r^n_{1j_m}$$

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-ϵ) 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-ϵ con un único estado inicial y un único estado final para cada caso base (símbolo a, ϵ, $\emptyset$) 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 ϵ 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) $A \to uB$
2) $A \to u$

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 $S \to 0A$, $A \to 10A$, $A \to \varepsilon$ es una GLD.

- $S \to 0A$: $u = 0$, $B = A$.
- $A \to 10A$: $u = 10$, $B = A$.
- $A \to \varepsilon$: $u = \varepsilon$ (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) $A \to Bu$
2) $A \to u$

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 $S \to S10$ y $S \to 0$ es una GLI.

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

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, $A \to aB$ y $C \to DC$) 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 $\to$ 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 $p \to aq$ en G.
   - Por cada estado final $p \in F$, se añade la producción $p \to \varepsilon$.
2) GR (Lineal por la Derecha) $\to$ AFND: Dada una GLD G, se puede construir directamente un Autómata Finito No Determinista (AFND) o un AFND con ϵ-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-ϵ).
- 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 Relación Tema 1: Modelos de Computación

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

a) Describir el lenguaje generado por la siguiente gramática:

$$S \to XYX$$

$$X \to aX \mid bX \mid \epsilon$$

$$Y \to bbb$$

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:

$$S \to aX$$

$$X \to aX \mid bX \mid \epsilon$$

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:

$$S \to XaXaX$$

$$X \to aX \mid bX \mid \epsilon$$

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:

$$S \to SS \mid XaXaX \mid \epsilon$$

$$X \to bX \mid \epsilon$$

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 (ϵ).
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 $S \to SS$.

Encontramos que el númeri 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:

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

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 (ϵ), 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 $S \to SS$). 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 (ϵ), 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 $S \to XaXaX$ y $X \to bX \mid \epsilon$.
  - Una concatenación de cadenas en R. Si $u, v \in R$, entonces $uv \in L$ porque la regla $S \to SS$ 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:

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

Determinar el lenguaje que genera.

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

Cada bloque abA puede convertirse o bien en $baab$ aplicando la regla $abA \to baab$, o bien en abb aplicando primero $A \to b$ (porque $abA \Rightarrow abb$). Finalmente $S \to a$. 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 $S \to abAS$ se tiene $(abA)^m S$ (prueba por inducción sobre m: base $m = 0$ trivial; paso: si $S \Rightarrow (abA)^k S$ entonces aplicando $S \to abAS$ al S final obtenemos $(abA)^{k+1} S$).

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

   Finalmente $S \to a$. 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 $S \to abAS$ para obtener $(abA)^m S$.

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

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

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

$$V = \{\langle numero\rangle, \langle digito\rangle\}$$
$$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$
$$S = \langle numero\rangle$$

- $\langle numero\rangle \to \langle numero\rangle\langle digito\rangle$
- $\langle numero\rangle \to \langle digito\rangle$
- $\langle digito\rangle \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

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:
   - $\langle numero\rangle \to \langle numero\rangle\langle digito\rangle$ permite construir cadenas de longitud arbitraria añadiendo dígitos.
   - $\langle numero\rangle \to \langle digito\rangle$ 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 $\langle digito\rangle \to 0 \mid 1 \mid \ldots \mid 9$.

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:

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

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

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

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:

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

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.

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

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.

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

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:

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

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:

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

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:

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

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:

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

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.

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

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 (≥ 3) donde la primera letra coincide con la central.

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

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.

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

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

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

es regular. Justificar la respuesta.

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

$$S \to aS \mid bS_1$$
$$S_1 \to aS_1 \mid bS_2$$
$$S_2 \to aS_2 \mid bS_3$$
$$S_3 \to aS_3 \mid bS_1 \mid \varepsilon$$

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:

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

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

Demostraremos 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, $L^* = \bigcup_{i \geq 0} L^i = L^0 \cup L^1 \cup L^2 \cup \ldots$. 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: $w = a_1 a_2 \ldots a_n$, donde $a_i \in L$ para todo $i = 1, \ldots, n$.

   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 $a_1 a_2 \ldots a_n = w \in L$.

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

Ejercicio 3.1.12. *Igualdad de homomorfismos. Dados dos homomorfismos $f : A^* \to B^*$, $g : A^* \to B^*$, 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 $h : A^* \to B^*$ 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 ϵ. 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, $\mathbf{S_1 = C_2}$, 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, ϵ, 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, $\mathbf{S_1 \neq C_2}$, 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 ϵ ($\epsilon \notin X$).

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: $w_0, w_1, w_2, \ldots$.
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, $\{w_{i_1}, w_{i_2}, \ldots, w_{i_k}\}$.
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 $L = \{w_{i_1}, w_{i_2}, \ldots, w_{i_k}\}$ el número natural $n = 2^{i_1} + 2^{i_2} + \cdots + 2^{i_k}$.
   - 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 Relación de problemas 1 bis

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

- *SENCILLOS*
  - *a)* $\{u \in \{0,1\}^* \text{ tales que } |u| \leq 4\}$
  - *b) 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.*
  - *c) El conjunto vacío.*
  - *d) El lenguaje formado por los números naturales.*
  - *e)* $\{a^n \in \{a,b\}^* \text{ con } n \geq 0\} \cup \{a^n b^n \in \{a,b\}^* \text{ con } n \geq 0\}$
  - *f)* $\{a^n b^{2n} c^m \in \{a,b,c\}^* \text{ con } n, m > 0\}$
  - *g)* $\{a^n b^m a^n \in \{a,b\}^* \text{ con } m, n \geq 0\}$
  - *h) Palabras con 0's y 1's que contengan la subcadena 00 y 11.*
  - *i) Palíndromos formados con las letras a y b.*
- *DIFICULTAD MEDIA*
  - *a)* $\{uv \in \{0,1\}^* \text{ tales que } u^{-1} \text{ es un prefijo de } v\}$
    Solución 3.2.1. media.a.

$$S \to XY$$
$$X \to 0X0 \mid 1X1 \mid \varepsilon$$
$$Y \to 1Y \mid 0Y \mid \varepsilon$$

    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.
  - *b)* $\{ucv \in \{a,b,c\}^* \text{ tales que } u \text{ y } v \text{ tienen la misma longitud}\}$
    Solución 3.2.1. media.b.

$$S \to c$$
$$S \to aSa \mid aSb \mid aSc \mid bSa \mid bSb \mid bSc \mid cSa \mid cSb \mid cSc$$

  - *c)* $\{u1^n \in \{0,1\}^* \text{ donde } |u| = n\}$
    Solución 3.2.1. media.c.

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

    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.
  - *d)* $\{a^n b^n a^{n+1} \in \{a,b\}^* \text{ con } n \geq 0\}$
    Solución 3.2.1. media.d.

$$S \to aSb$$
$$S \to aA$$
$$A \to aA$$
$$A \to a$$

Esta gramática genera cadenas de la forma $a^n b^n a^{n+1}$, donde $n \geq 0$. La primera regla $S \to aSb$ asegura que el número de a y b en las primeras dos partes de la cadena sea igual. La regla $S \to aA$ 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)* $\{a^n b^m c^k$ *tales que* $k = m + n\}$
    Solución 3.2.1. difícil.a.

$$S \to aSbC \mid \varepsilon$$
$$C \to aC \mid bC \mid \varepsilon$$

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

$$S \to 0S \mid 1A \mid \varepsilon$$
$$A \to 0B \mid 1C$$
$$B \to 0D \mid 1E$$
$$C \to 0F \mid 1S$$
$$D \to 0A \mid 1B$$
$$E \to 0C \mid 1D$$
$$F \to 0E \mid 1F$$

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:

$$r' = (2 \cdot r + b) \mod 7,$$

donde r es el residuo actual ($r = n \mod 7$).

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

    $$S \to X\#X$$
    $$X \to 0X0 \mid 1X1 \mid \epsilon$$

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

$$S \to A\#A$$
$$A \to aA \mid \epsilon$$

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:

$$S \to aS \mid P$$
$$P \to a^2 \mid a^3 \mid a^5 \mid a^7 \mid \dots$$

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 decidible[1], por lo que pertenece a la clase recursiva; existe por tanto alguna gramática tipo-0 o máquina de Turing[2] 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:

$$S \to aSbb \mid \epsilon$$

En esta gramática:

[1] Un lenguaje es decidible si existe un algoritmo que, dado cualquier elemento del alfabeto, puede determinar en un tiempo finito si pertenece o no al lenguaje.

[2] 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.

- $S \to aSbb$ asegura que por cada símbolo a generado, se generan dos símbolos b, lo que garantiza que $n \leq m^2$.
- $S \to \epsilon$ 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 Relación de Problemas 2: 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:*

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

- $F = \{q_2\}$

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 $L = \{u_1 00 u_2 \mid u_1, u_2 \in \{0, 1\}^*\}$.

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".

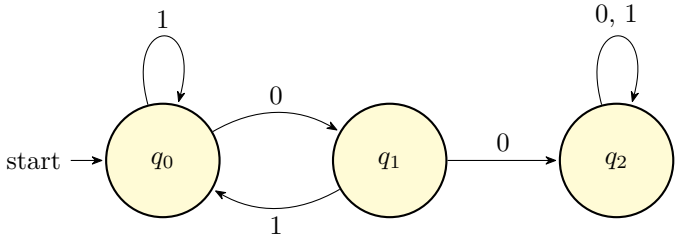


Figura 3.1: *Diagrama de Transición del AFD del Ejercicio 1.*

Ejercicio 3.3.2. *Dado el AFD*

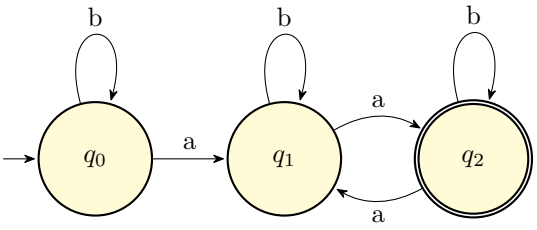


Figura 3.2: *Diagrama de Transición para el Ejercicio 2.*

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:

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

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, $\emptyset$.

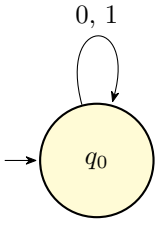


Figura 3.3: *AFD para* $L = \emptyset$*.*

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

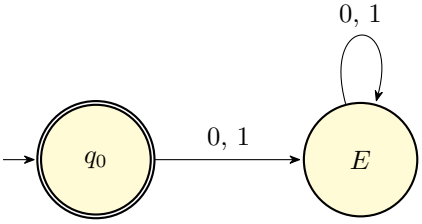


Figura 3.4: *AFD para* $L = \{\varepsilon\}$.

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

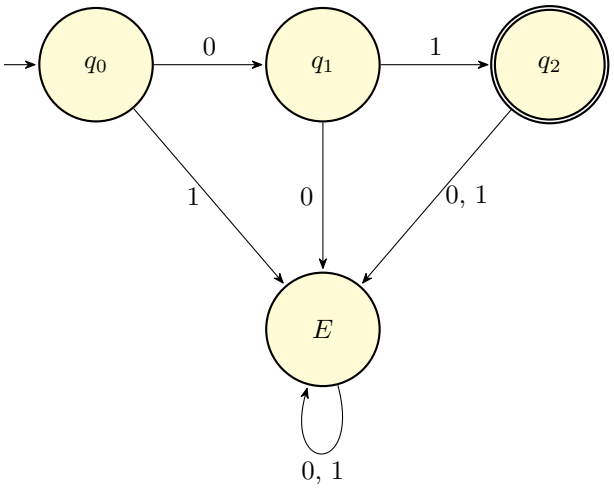


Figura 3.5: *AFD para* $L = \{01\}$.

4) El lenguaje $\{11, 00\}$.

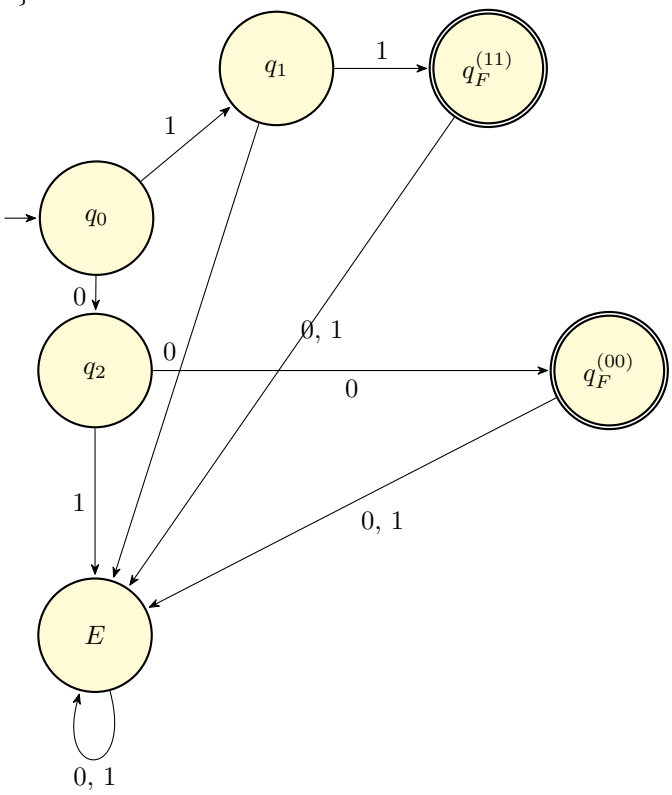


Figura 3.6: *AFD para* $L = \{11, 00\}$.

5) El lenguaje $\{(01)^i \mid i \geq 0\}$.

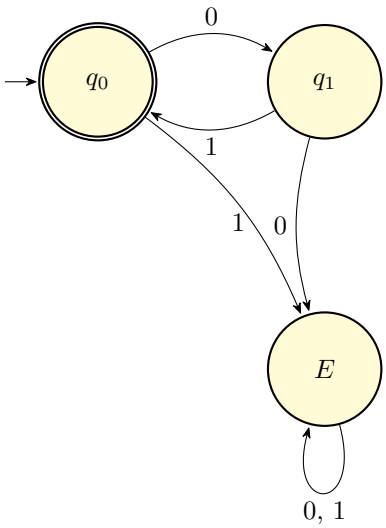


Figura 3.7: *AFD para* $L = \{(01)^i \mid i \geq 0\}$.

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

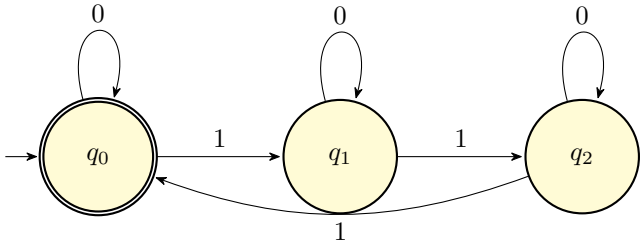


Figura 3.8: *AFD para* $L = \{w \mid N_1(w) \equiv 0 \pmod 3\}$.

Ejercicio 3.3.4. *Obtener a partir de la gramática regular* $G = (\{S, B\}, \{1, 0\}, P, S)$*, con* $P = \{S \to 110B, B \to 1B, B \to 0B, B \to \varepsilon\}$*, 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 $B \to 1B$ y $B \to 0B$, que permiten generar cualquier secuencia finita de 1's y 0's, y $B \to \varepsilon$, que permite terminar la derivación. Por lo tanto, el lenguaje generado por B es $L(B) = \{1, 0\}^* = \Sigma^*$.
   - La producción inicial $S \to 110B$ 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 (Σ): $\Sigma = \{1, 0\}$, 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 $S \to 110B$.
      - 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 $A \to u$, donde $u \in T^*$. La producción $B \to \varepsilon$ (donde $u = \varepsilon$) indica que el estado B debe ser un estado final.

   $F = \{q_3\}$.

3) Construcción de la Función de Transición (δ):
   - Producción $S \to 110B$ (Ruta principal):

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

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

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

   - Producción $B \to \varepsilon$ (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 δ está definida por:

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

4) Representación Gráfica del AFND:

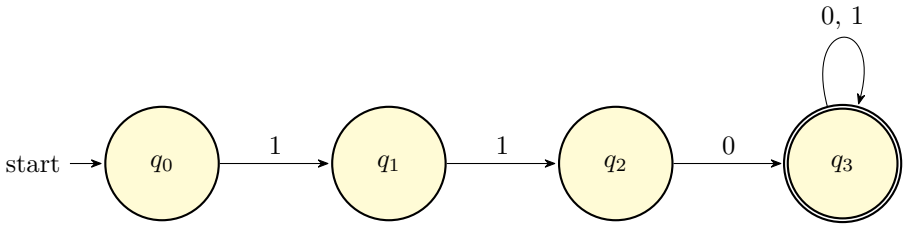


Figura 3.9: *Autómata Finito No Determinista (AFND) que reconoce el lenguaje $L = 110(1+0)^*$, generado por la gramática G.*

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

Solución 3.3.5. Para la resolución de este ejercicio se sguirán lo 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 $A \to Bw$ o $A \to w$, donde $w \in T^*$ y $B \in V$ (en este caso, $S \to S10$ y $S \to 0$).

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

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

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

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

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

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

   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 (δ''):
   - $q_0 \xrightarrow{0} q_2$ (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 δ 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 (δ):

δ	0	1
q_2	q_0	E
q_0	E	q_1
q_1	q_0	E
E	E	E

4) Representación Gráfica del AFD
   Representamos el Autómata Finito Determinista M que reconoce $L = 0(10)^*$.

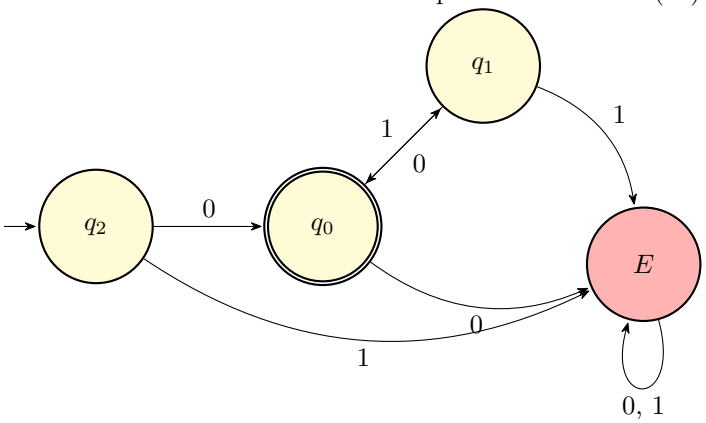


Figura 3.10: *Autómata Finito Determinista (AFD) para el lenguaje* $L = 0(10)^*$.

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. El ejercicio requiere tres partes:

1) Construir $AFND_1$ para $L_1 = \{u \in \{0,1\}^* \mid u \text{ contiene } 010\}$.
2) Construir $AFND_2$ 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)

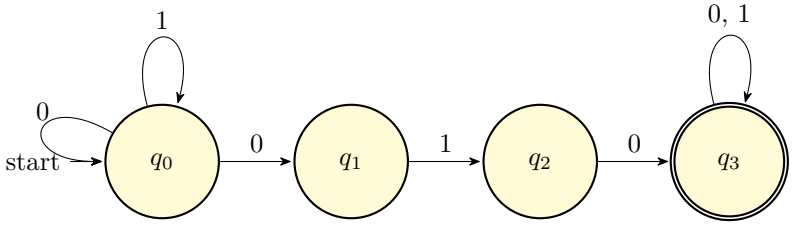
Para construir un Autómata Finito No Determinista (AFND) que acepte cualquier cadena que contenga la subcadena $W = 010$, permitimos transiciones no deterministas en el estado inicial que "adivinan" el comienzo de la subcadena.
Definimos $AFND_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$:

- $Q_1 = \{q_0, q_1, q_2, q_3\}$, 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 δ_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 $a \in \{0, 1\}$.

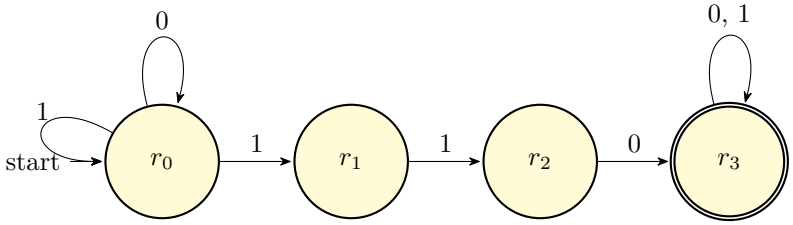
Esta estructura coincide con el método estándar de construcción de AFNDs para la subcadena W.



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 δ_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): $Q_1' = \{q_0, q_1, q_2, q_3\}$.

$$\begin{aligned} \delta_1'(q_0,0) &= q_1, & \delta_1'(q_0,1) &= q_0 \\ \delta_1'(q_1,0) &= q_1, & \delta_1'(q_1,1) &= q_2 \\ \delta_1'(q_2,0) &= q_3, & \delta_1'(q_2,1) &= q_0 \\ \delta_1'(q_3,0) &= q_3, & \delta_1'(q_3,1) &= q_3 \end{aligned}$$

DFA M_2' (Rastreo de 110): $Q_2' = \{r_0, r_1, r_2, r_3\}$.

$$\begin{aligned} \delta_2'(r_0,0) &= r_0, & \delta_2'(r_0,1) &= r_1 \\ \delta_2'(r_1,0) &= r_0, & \delta_2'(r_1,1) &= r_2 \\ \delta_2'(r_2,0) &= r_3, & \delta_2'(r_2,1) &= r_2 \\ \delta_2'(r_3,0) &= r_3, & \delta_2'(r_3,1) &= r_3 \end{aligned}$$

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}.

Tabla de Transiciones del AFD Producto M

Estado $S_{ij} = (q_i, r_j)$	Estatus	Transición con 0	Transición con 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}
Exploración profunda de estados			
$S_{30} = (q_3, r_0)$	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	$\mathbf{S_{33}}$	S_{32}
$S_{22} = (q_2, r_2)$	Visto 01/11	$\mathbf{S_{33}}$	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}
$\mathbf{S_{33}} = (\mathbf{q_3}, \mathbf{r_3})$	L1 & L2 Completos	$\mathbf{S_{33}}$	$\mathbf{S_{33}}$

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 $\mathbf{S_{33}} = (q_3, r_3)$

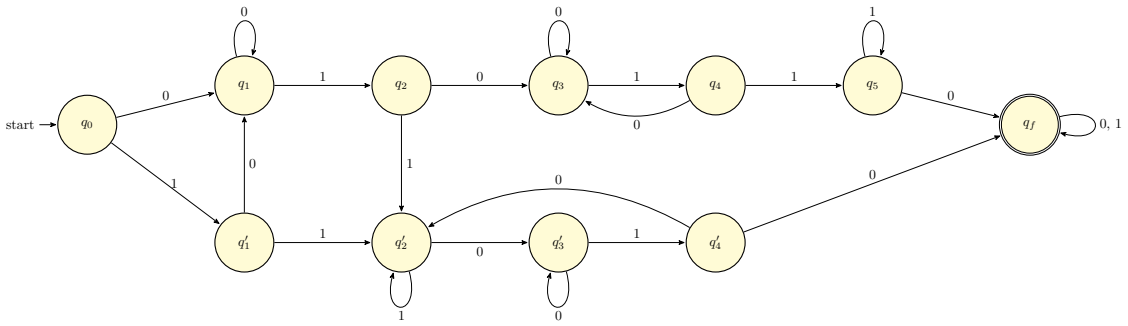


Figura 3.11: *Diagrama del AFD para el lenguaje que contiene simultáneamente las subcadenas* 010 *y* 110.

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}. $1 \to S_{01}$. $1 \to S_{02}$. $0 \to S_{10}$. $1 \to S_{21}$. $0 \to S_{30}$. L1 está completa en S_{30}. Si leemos 10110: ..,$11 \to S_{32}$. $\mathbf{S_{32}} \xrightarrow{\mathbf{0}} \mathbf{S_{33}}$. 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 $A \to aB$ en una Gramática Lineal por la Derecha, y el estado final S_{33} generaría la producción $S_{33} \to \varepsilon$.
Por ejemplo, las transiciones deterministas del AFD implican las siguientes producciones iniciales:

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

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

Solución 3.3.6. Otra solucion es la que se va a describir en esta parte.

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.

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

$$S \to AB$$
$$A \to aA$$
$$A \to c$$
$$B \to bBb$$
$$B \to b$$

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.*

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.*

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.*

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

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

Pista. Construir un AFD asociado.

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.
2) Palabras que no contienen la subcadena ab.
3) Palabras que no contienen la subcadena aba.

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

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

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

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.
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:*

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

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

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

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

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

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

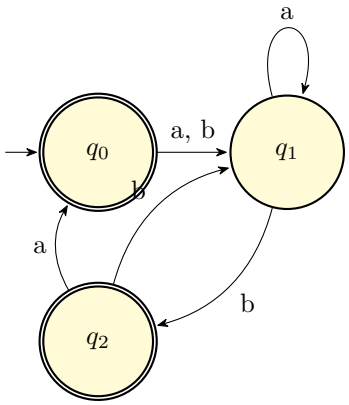


Figura 3.12: *Diagrama de Transición para el Ejercicio 18.*

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.

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. *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 3.3.24. *Dada una palabra* $u = a_1 \dots a_n \in A^*$*, se llama* $Per(u)$ *al conjunto*

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

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

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

¿Es posible que, siendo L regular, Per(L) no lo sea?

Bibliografía

[1] Ismael Sallami Moreno, **Estudiante del Doble Grado en Ingeniería Informática + ADE**, Universidad de Granada, 2025.