



TECNOLÓGICO NACIONAL DE MEXICO

INSTITUTO TECNOLÓGICO DE IZTAPALAPA

LENGUAJES Y AUTOMATAS I

INTEGRANTES:

ARREDONDO FLORES ALEXA KETZALI - 181080005

BOHORQUEZ LOPEZ MIGUEL ANGEL - 181080006

RODRIGUEZ GARCIA JESUS - 181080027

“ APUNTES - LENGUAJES Y AUTÓMATAS ”



TEORÍA DE LA COMPUTACIÓN

¿Cuáles son las capacidades y limitaciones fundamentales de las computadoras?

En cada una de las tres áreas (autómatas, computabilidad y complejidad) esta pregunta se interpreta de manera diferente y las respuestas varían según la interpretación.

Teoría de la Complejidad

Los problemas de la computadora vienen en diferentes variedades; algunos son fáciles y otros difíciles.

¿Qué hace que algunos problemas sean computacionalmente difíciles y otros fáciles?

Ésta es la cuestión central de la teoría de la complejidad. Sorprendentemente, no sabemos la respuesta, aunque se ha investigado intensamente durante más de 40 años. En un logro importante de la teoría de la complejidad hasta ahora, los investigadores han descubierto un elegante esquema para clasificar los problemas de acuerdo con su dificultad computacional.

Tiene varias opciones cuando se enfrenta a un problema que parece ser difícil de calcular.

Primero, al comprender qué aspecto del problema está en la raíz de la dificultad, es posible que pueda modificarlo para que el problema se resuelva más fácilmente.

En segundo lugar, es posible que pueda conformarse con una solución menos que perfecta al problema. En ciertos casos, encontrar soluciones que solo se aproximen a la perfecta es relativamente fácil.

En tercer lugar, algunos problemas son difíciles solo en el peor de los casos, pero fáciles la mayor parte del tiempo. Dependiendo de la aplicación, es posible que se sienta satisfecho con un procedimiento que ocasionalmente es lento pero que generalmente se ejecuta rápidamente.

Por último, puede considerar tipos alternativos de cálculo, como el cálculo aleatorio, que puede acelerar determinadas tareas.

Teoría de Autómata

La teoría de los autómatas se ocupa de las definiciones y propiedades de los modelos matemáticos de computación. Estos modelos juegan un papel en varias áreas aplicadas de la informática.

Uno de estos modelos son los Autómatas Finitos, utilizados en:

- Procesamiento de textos.
- Compiladores.
- Diseño de Hardware.

Otro modelo son las Gramáticas Libres de Contexto, usadas en:

- Lenguajes de programación.
- Inteligencia artificial.

Computabilidad

Está muy relacionado con la Teoría de la Complejidad, ya que introduce varios de los conceptos que esta área utiliza. Tiene como finalidad principal la clasificación de diferentes problemas, así como formalizar el concepto de computar.

Nociones matemáticas y terminología.

NOCIONES Y TERMINOLOGÍA MATEMÁTICA

Conjuntos

Un **conjunto** es un grupo de objetos representados como una unidad. Los conjuntos pueden contener cualquier tipo de objeto, incluidos números, símbolos e incluso otros conjuntos. Los objetos de un conjunto se denominan elementos o miembros. Los conjuntos pueden describirse formalmente de varias formas.

$$S = \{1, 2, 2, 3, 4\}$$

El **conjunto con cero miembros** se llama conjunto vacío y está escrito \emptyset . Un conjunto con un miembro a veces se denomina conjunto singleton, y un conjunto con dos miembros se llama par desordenado.

$$H = \{ \}$$

Como suele ocurrir en matemáticas, una imagen ayuda a aclarar un concepto. Para los conjuntos, usamos un tipo de imagen llamada Diagrama de Venn. Representa conjuntos como regiones encerradas por líneas circulares.

Secuencias y Tuples

Una **secuencia** de objetos es una lista de estos objetos en algún orden. Por lo general, designamos una secuencia escribiendo la lista entre paréntesis. Por ejemplo, la secuencia 7, 21, 57 estaría escrito (7, 21, 57).

Los conjuntos y secuencias pueden aparecer como elementos de otros conjuntos y secuencias.

Funciones y Relaciones

Las funciones son fundamentales para las matemáticas. Una función es un objeto que establece una relación entrada-salida. Una función toma una entrada y produce una salida. En cada función, la misma entrada siempre produce la misma salida. Si F es una función cuyo valor de salida es segundo cuando el valor de entrada es una, nosotros escribimos.

$$f(a) = b$$

Un predicado o propiedad es una función cuyo rango es {VERDADERO FALSO}

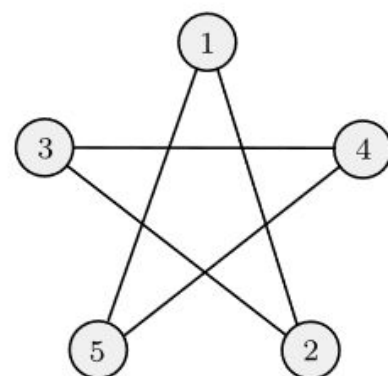
El conjunto de posibles entradas a la función se denomina DOMINIO, las salidas de una función provienen de un conjunto llamado RANGO. La notación para decir que f es una función con dominio D y rango R es:

$$f: D \rightarrow R$$

Gráficos

Un grafo no dirigido, o simplemente un gráfico, es un conjunto de puntos con líneas que conectan algunos de los puntos. Los puntos se llaman nodos o vértices y las líneas se llaman bordes. Un gráfico dirigido tiene flechas en lugar de líneas.

El número de aristas es un nodo particular es un grado de ese nodo, en la imagen anterior todos los nodos tienen 2 grados.



Cadenas y Lenguajes

Las cadenas de caracteres son bloques de construcción fundamentales en la informática. El alfabeto sobre el que se definen las cadenas puede variar según la

aplicación. Para nuestros propósitos, definimos alfabeto a cualquier conjunto finito no vacío. Los miembros del alfabeto son los símbolos del alfabeto. Generalmente usamos letras griegas Σ y Γ .

Lógica Booleana

La lógica booleana es un sistema matemático construido alrededor de los dos valores CIERTO y FALSO. Los valores CIERTO y FALSO se llaman los Valores booleanos y a menudo están representados por los valores 1 y 0.

La Ley distributiva para AND y OR resulta útil cuando manipulamos expresiones booleanas

Definición: describir los objetos y las nociones que usamos. Una definición puede ser simple, como en la definición de conjunto dado anteriormente en este capítulo, o complejo como en la definición de seguridad en un sistema criptográfico.

Una prueba es un argumento lógico convincente de que una declaración es verdadera. En matemáticas, un argumento debe ser hermético; es decir, convincente en un sentido absoluto.

Un teorema es una afirmación matemática probada como cierta. Generalmente nos reservamos el uso de esa palabra para declaraciones de especial interés. Tales declaraciones se llaman lemas. Un teorema o su demostración pueden permitirnos concluir fácilmente que otros enunciados relacionados son verdaderos. Estas declaraciones se llaman corolarios del teorema.

DEFINICIONES, TEOREMAS Y PRUEBAS

Los teoremas y las pruebas son el corazón y alma de las matemáticas, y las definiciones son el espíritu.

Definiciones: Describen los objetos y nociones que usamos.

Prueba: Es un argumento lógico convincente que una formación es verdadera.

Teorema: Es un enunciado matemático que su prueba es totalmente verdad.

Encontrar pruebas

La única forma de determinar la verdad o falsedad de un enunciado matemático es con una prueba matemática.

Tipos de prueba

Varios tipos de argumentos surgen con frecuencia en las demostraciones matemáticas. A continuación, describimos algunos que ocurren a menudo en la teoría de la computación. Tengamos en cuenta que una prueba puede contener más de un tipo de argumento porque la prueba puede contener varias subpruebas diferentes.

Prueba por construcción

Muchos teoremas establecen que existe un tipo particular de objeto. Una forma de probar tal teorema es demostrando cómo construir el objeto. Esta técnica es una prueba para la construcción.

Prueba por contradicción

En una forma común de argumento para probar un teorema, asumimos que el teorema es falso y luego mostramos que esta suposición conduce a una consecuencia obviamente falsa, llamada contradicción. Usamos este tipo de razonamiento con frecuencia en la vida cotidiana.

Prueba por inducción

La prueba por inducción es un método avanzado que se usa para mostrar que todos los elemento

1ra UNIDAD - INTRODUCCIÓN A LA TEORÍA DE LENGUAJES FORMALES

Alfabeto

Un alfabeto es un conjunto de símbolos finito y no vacío de elementos llamados símbolos o letras. Es una agrupación, que se lee con un orden determinado, de las gráficas utilizadas para representar el lenguaje que sirve de sistema de comunicación, un grupo de letras estructurado bajo un orden específico aceptado a nivel general en el marco de una lengua

Convencionalmente, utilizados el símbolo Σ (sumatoria) para designar un alfabeto. Entre los alfabetos más comunes se incluyen los siguientes:

- $\Sigma = \{0,1\}$, el alfabeto binario
- $\Sigma = \{a, b, \dots, z\}$, es el conjunto de todas las letras minúsculas

- El conjunto de todos los caracteres ASCII

Cadena

Una cadena de caracteres (que también se denomina en ocasiones palabra) es una secuencia finita de símbolos seleccionados de algún alfabeto.

Una cadena o palabra es una secuencia finita de símbolos que pertenecen a un alfabeto y comúnmente se denota con la letra.

Ejemplo: si $\Sigma = \{0,1\}$, entonces $\Sigma^1 = \{0,1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$, $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$, etc.

La cadena vacía: La cadena vacía es aquella cadena que presenta cero apariciones de símbolos. Esta cadena, designada por ϵ , es una cadena que puede construirse en cualquier alfabeto

Ejemplo: observe que $\Sigma^0 = \{\epsilon\}$, independientemente de cuál sea el alfabeto Σ . Es decir, ϵ es la única cadena cuya longitud es 0.

Lenguajes

Un conjunto de cadenas, todas ellas seleccionadas de un Σ^* , donde Σ es un determinado alfabeto se denomina lenguaje. Ya que estas pueden ser cualquier cadena que cumpla con lo siguiente, está formada por los símbolos. Los lenguajes habituales pueden interpretarse como conjuntos de cadenas.

Ejemplo: Sería el inglés, donde la colección de las palabras correctas inglesas es un conjunto de cadenas del alfabeto que consta de todas las letras.

Ejemplo: Es el lenguaje C, o cualquier otro lenguaje de programación, donde los programas correctos son un subconjunto de las posibles cadenas que pueden formarse a partir del alfabeto del lenguaje.

Tipos de Lenguajes

Lenguajes Declarativos: Es fundamentalmente lenguajes de órdenes, dominados por Sentencias que expresan “lo que hay que hacer” en vez de “cómo hacerlo”.

Lenguajes de Alto Nivel: Son los más utilizados como lenguajes de programación permiten que los algoritmos se expresen en un nivel y estilo de escritura fácilmente legible y comprensible por otros programadores.

Lenguaje ensamblador: Es el programa en que se realiza la tracción de un programa escrito en un programa escrito en ensamblador y lo pasa a lenguaje

máquina. Directa o no directa de la traducción en que las instrucciones no son más que instrucciones que ejecuta la computadora.

Lenguaje máquina: Es como la máquina interpreta lo que nosotros queremos hacer es una lectura de 0 y 1 es decir binarios de un conjunto infinito tienen una propiedad.

Estructura de un traductor

Un traductor es un programa que tiene como entrada un texto escrito en un lenguaje (lenguaje fuente) y como salida produce un texto escrito en un lenguaje (lenguaje objeto) que preserva el significado de origen.

Fases de un compilador

Los compiladores son programas de computadora que traducen de un lenguaje a otro. Un compilador toma como su entrada un programa escrito en lenguaje fuente y produce un programa equivalente escrito en lenguaje objeto.

Un compilador se compone internamente de varias etapas, o fases, que realizan operaciones lógicas.

- **Analizador léxico:** Lee la secuencia de caracteres de izquierda a derecha del programa fuente y agrupa las secuencias de caracteres en unidades con significado propio (componentes léxicos o “tokens” en inglés).

Las palabras clave, identificadores, operadores, constantes numéricas, signos de puntuación como separadores de sentencias, llaves, parentesis, etc., son diversas clasificaciones de componentes léxicos.

- **Análisis sintáctico:** Determina si la secuencia de componentes léxicos sigue la sintaxis del lenguaje y obtiene la estructura jerárquica del programa en forma de árbol, donde los nodos son construcciones de alto nivel del lenguaje.

Se determinan las relaciones estructurales entre los componentes léxicos, esto es semejante a realizar el análisis gramatical sobre una frase en lenguaje natural. La estructura sintáctica la definiremos mediante las gramáticas independientes del contexto.

- **Análisis semántico:** Realiza las comprobaciones necesarias sobre el árbol sintáctico para determinar el correcto significado del programa.

Las tareas básicas a realizar son: La verificación e inferencia de tipos en asignaciones y expresiones, la declaración del tipo de variables y funciones antes de su uso, el correcto uso de operadores, el ámbito de las variables y la correcta llamada a funciones.

Nos limitaremos al análisis semántico estático (en tiempo de compilación), donde es necesario hacer uso de la Tabla de símbolos, como estructura de datos para almacenar información sobre los identificadores que van surgiendo a lo largo del programa. El análisis semántico suele agregar atributos (como tipos de datos) a la estructura del árbol semántico.

- **Generación y optimización de código intermedio:** La optimización consiste en la calibración del árbol sintáctico donde ya no aparecen construcciones de alto nivel. Generando un código mejorado, ya no estructurado, más fácil de traducir directamente a código ensamblador o máquina, compuesto de un código de tres direcciones (cada instrucción tiene un operador, y la dirección de dos operandos y un lugar donde guardar el resultado), también conocida como código intermedio.
- **Generador de código objeto:** Toma como entrada la representación intermedia y genera el código objeto. La optimización depende de la máquina, es necesario conocer el conjunto de instrucciones, la representación de los datos (número de bytes), modos de direccionamiento, número y propósito de registros, jerarquía de memoria, encauzamientos, etc.

Suelen implementarse a mano, y son complejos porque la generación de un buen código objeto requiere la consideración de muchos casos particulares.

- **Tabla de Símbolos:** Es una estructura tipo diccionario con operaciones de inserción, borrado y búsqueda, que almacena información sobre los símbolos que van apareciendo a lo largo del programa como son: – los identificadores (variables y funciones) – Etiquetas – tipos definidos por el usuario (arreglos, registros, etc.)

Además almacena el tipo de dato, método de paso de parámetros, tipo de retorno y de argumentos de una función, el ámbito de referencia de identificadores y la dirección de memoria. Interacciona tanto con el

analizador léxico, sintáctico y semántico que introducen información conforme se procesa la entrada. La fase de generación de código y optimización también la usan.

- **Gestor de errores:** Detecta e informa de errores que se produzcan durante la fase de análisis. Debe generar mensajes significativos y reanudar la traducción.

Encuentra errores: – En tiempo de compilación: errores léxicos (ortográficos), sintácticos (construcciones incorrectas) y semánticos (p.ej. errores de tipo) – En tiempo de ejecución: direccionamiento de vectores fuera de rango, divisiones por cero, etc. – De especificación/diseño: compilan correctamente pero no realizan lo que el programador desea.

Se tratarán sólo errores estáticos (en tiempo de compilación). Respecto a los errores en tiempo de ejecución, es necesario que el traductor genere código para la comprobación de errores específicos, su adecuado tratamiento y los mecanismos de tratamiento de excepciones para que el programa se continúe ejecutando específica.

2da UNIDAD - EXPRESIONES REGULARES

Representación Finita

Dado un problema de Programación Semi-Infinita, si se puede obtener una representación finita del conjunto factible, pueden funcionar para resolver el problema de los métodos de programación con finitas restricciones.

En matemáticas, un conjunto finito es un conjunto que tiene un número finito de elementos.

Por ejemplo: {2, 4, 6, 8, 10} es un conjunto finito con cinco elementos. La cardinalidad o número de elementos de un conjunto finito es igual a un número natural.

Si un conjunto no es finito, entonces es infinito. Por ejemplo, el conjunto $N = \{1, 2, 3, \dots\}$ de los números naturales es infinito. Todo conjunto finito es un conjunto numerable, puesto que sus elementos pueden contarse, pero la recíproca es falsa: existen conjuntos numerables que no son finitos (como el propio N).

En la primera parte se utilizan los sistemas lineales infinitos que pueden ser reducidos a un sistema finito equivalente, dándose además condiciones suficientes y métodos para efectuar tal reducción. En la segunda parte se encuentran diferentes procedimientos de obtención de la representación finita del conjunto de soluciones de un sistema infinito dado.

Gramáticas

Es un conjunto finito de reglas que describen toda la secuencia de símbolos pertenecientes a un lenguaje específico L . Dos gramáticas que describen el mismo lenguaje se llaman gramáticas equivalentes.

Una gramática es una **estructura algebraica** formada por cuatro elementos fundamentales:

$$G = \{ VN, VT, S, P \}$$

donde

- VN es el conjunto de elementos **No Terminales**
- VT es el conjunto de elementos **Terminales**
- S es el **Símbolo inicial** de la gramática
- P es el conjunto de **Reglas de Producción**

Todas las cadenas del lenguaje definidas por la gramática están formadas con símbolos del *vocabulario terminal* VT . El vocabulario terminal se define por enumeración de los símbolos terminales.

El *vocabulario no terminal* VN es el conjunto de símbolos introducidos como elementos auxiliares para la definición de la gramática, y que no figuran en las sentencias del lenguaje.

La intersección entre el vocabulario terminal y no terminal es el conjunto vacío:

$$\{VN\} \cap \{VT\} = \{\emptyset\}$$

La unión entre el vocabulario terminal y no terminal es el vocabulario.

$$\{VN\} \cup \{VT\} = \{V\}$$

En ocasiones es importante distinguir si un determinado vocabulario incluye o no la cadena vacía, indicándose respectivamente con superíndice $+$ o superíndice $*$, tal como se muestra a continuación:

$$V^+ = V - \{\emptyset\} \quad V^* = V + \{\emptyset\}$$

El *símbolo inicial* S es un símbolo no terminal a partir del cual se aplican las reglas de la gramática para obtener las distintas cadenas del lenguaje.

Las *producciones* P son las reglas que se aplican desde el símbolo inicial para obtener las cadenas del lenguaje. El conjunto de producciones P se define por medio de la enumeración de las distintas producciones, en forma de reglas o por medio de un metalenguaje.

Árboles de Derivación

Un árbol de derivación permite mostrar gráficamente cómo se puede derivar cualquier cadena de un lenguaje a partir del símbolo distinguido de una gramática que genera ese lenguaje.

Un árbol es un conjunto de puntos, llamados nodos, unidos por líneas, llamadas arcos. Un arco conecta dos nodos distintos. Para ser un árbol un conjunto de nodos y arcos debe satisfacer ciertas propiedades:

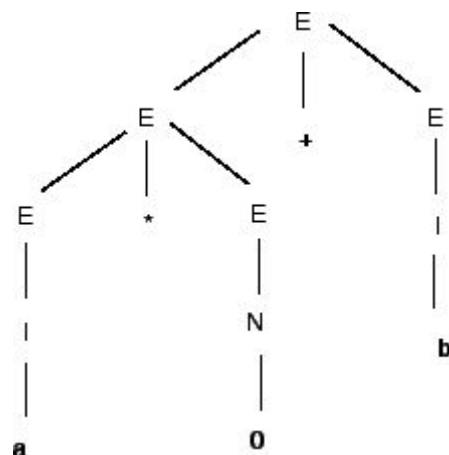
- Hay un único nodo distinguido, llamado raíz (se dibuja en la parte superior) que no tiene arcos incidentes.
- Todo nodo c excepto el nodo raíz está conectado con un arco a otro nodo k , llamado el padre de c (c es el hijo de k). El padre de un nodo, se dibuja por encima del nodo.
- Todos los nodos están conectados al nodo raíz mediante un único camino.
- Los nodos que no tienen hijos se denominan hojas, el resto de los nodos se denominan nodos interiores.

$$\Sigma = \{a, b, 0, 1, +, *, -, /, (,)\}$$

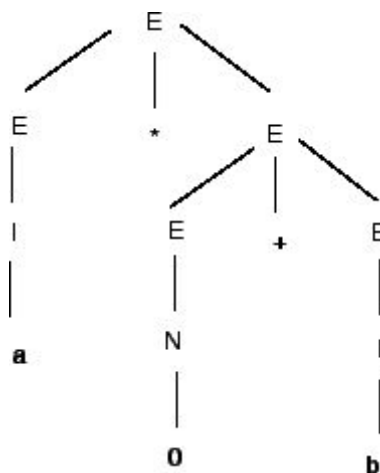
$$E \rightarrow E + E \mid E * E \mid E - E \mid E / E \mid (E) \mid I \mid N$$

$$N \rightarrow 0 \mid 1 \mid N0 \mid N1$$

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$



$a * 0 + b$ interpretado como $(a * 0) + b$

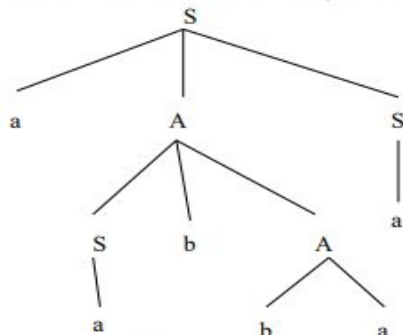


$a * 0 + b$ interpretado como $a * (0 + b)$

Se llama **derivación por la izquierda** asociada a un árbol a aquella en la que siempre se deriva primero la primera variable (más a la izquierda) que aparece en la palabra. Se llama **derivación por la derecha** asociada a un árbol a aquella en la que siempre se deriva primero la última variable (más a la derecha) que aparece en la palabra.

Ejemplo

Dado el árbol de la palabra *aabbbaa*



Derivación por la izquierda:

$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbAS \Rightarrow aabbbaa$

Derivación por la derecha:

$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbaa \Rightarrow aabbbaa$

La derivación más a la izquierda de un árbol se obtiene recorriéndolo en profundidad y de izquierda a derecha.

La derivación más a la derecha de un árbol se obtiene recorriéndolo en profundidad y de derecha a izquierda.

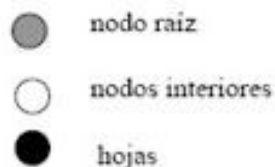
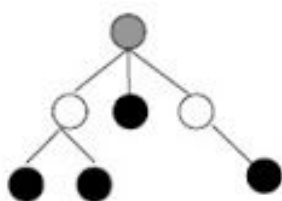
En una derivación en general, seguimos el orden que queremos.

Se dan las siguientes correspondencias:



El árbol de derivación tiene las siguientes propiedades:

- el nodo raíz está rotulado con el símbolo distinguido de la gramática;
- cada hoja corresponde a un símbolo terminal o un símbolo no terminal;
- cada nodo interior corresponde a un símbolo no terminal.



Para cada cadena del lenguaje generado por una gramática es posible construir (al menos) un árbol de derivación, en el cual cada hoja tiene como título uno de los símbolos de la cadena.

Gramáticas regulares

Son una gramática formal (N, Σ, P, S) que pueden ser clasificadas como regular izquierda y regular derecha. Estas gramáticas solo pueden generar a los lenguajes regulares de manera similar a los autómatas finitos y las expresiones regulares.

Una **gramática regular derecha** es aquella cuyas reglas de producción P son de la siguiente forma:

1. $A \rightarrow \alpha$, donde A es un símbolo no-terminal en N y α uno terminal en Σ
2. $A \rightarrow \alpha B$, donde A y B pertenecen a N y α pertenece a Σ
3. $A \rightarrow \varepsilon$, donde A pertenece a N .

Análogamente, en una **gramática regular izquierda**, las reglas son de la siguiente forma:

1. $A \rightarrow \alpha$, donde A es un símbolo no-terminal en N y α uno terminal en Σ
2. $A \rightarrow B\alpha$, donde A y B pertenecen a N y α pertenece a Σ
3. $A \rightarrow \varepsilon$, donde A pertenece a N .

Las gramáticas formales definen un lenguaje describiendo cómo se pueden generar las cadenas del lenguaje. Una gramática formal es una cuádrupla:

$$G = (N, T, P, S)$$

Donde:

N es un conjunto finito de símbolos no terminales

T es un conjunto finito de símbolos terminales $N \cap T = \emptyset$

P es un conjunto finito de producciones

Cada producción de P tiene la forma

$$\alpha \rightarrow \beta, \alpha = \phi A \rho \text{ y } \beta = \phi \omega \rho$$

$\phi, \omega, \rho \in (N \cup T)^*$ y A es S ó $A \in N$

- S es el símbolo distinguido o axioma $S \notin (N \cup T)$

Restringiendo los formatos de producciones permitidas en una gramática, se pueden especificar cuatro tipos de gramáticas (tipo 0, 1, 2 y 3) y sus correspondientes clases de lenguajes.

3ra UNIDAD - AUTÓMATAS FINITOS

Autómata Finito

Es un modelo computacional que realiza cálculos en forma automática sobre una entrada para producir una salida (máquina de estado finito).

Este modelo está conformado por un alfabeto, un conjunto de estados y un conjunto de transiciones entre dichos estados. Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de caracteres pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.

La finalidad de los autómatas finitos es la de reconocer lenguajes regulares, que corresponden a los lenguajes formales más simples según la Jerarquía de Chomsky.

Formalmente, un autómata finito es una 5-tupla $(Q, \Sigma, q_0, \delta, F)$ donde:

- Q es un conjunto finito de estados;
- Σ es un alfabeto finito;
- $q_0 \in Q$ es el estado inicial;
- $\delta: Q \times \Sigma \rightarrow Q$ es una función de transición;
- $F \subseteq Q$ es un conjunto de estados finales o de aceptación.

Autómata Finito Determinista

Un autómata finito determinista (AFD) es un autómata finito que además es un sistema determinista; es decir, para cada estado $q \in Q$ en que se encuentre el

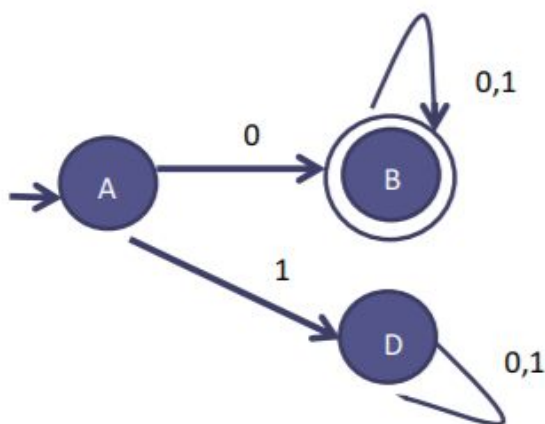
autómata, y con cualquier símbolo $a \in \Sigma$ del alfabeto leído, existe siempre a lo más una transición posible $\delta(q,a)$.

En un AFD no pueden darse ninguno de estos dos casos:

- Que existan dos transiciones del tipo $\delta(q,a)=q_1$ y $\delta(q,a)=q_2$, siendo $q_1 \neq q_2$;
- Que existan transiciones del tipo $\delta(q,\varepsilon)$, salvo que q sea un estado final, sin transiciones hacia otros estados.

Ejemplo: Obtenga un AFD dado el siguiente lenguaje definido en el alfabeto $\Sigma = \{0,1\}$. El conjunto de cadenas que inician en "0".

SOLUCIÓN



Autómata Finito No Determinista

Un autómata finito no determinista (AFND) es aquel que, a diferencia de los autómatas finitos deterministas, posee al menos un estado $q \in Q$, tal que para un símbolo $a \in \Sigma$ del alfabeto, existe más de una transición $\delta(q,a)$ posible.

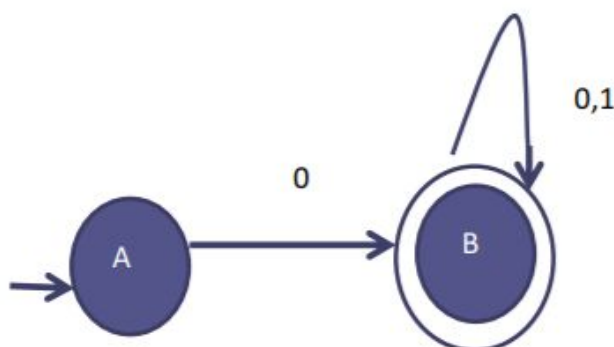
Haciendo la analogía con los AFDs, en un AFND puede darse cualquiera de estos dos casos:

- Que existan transiciones del tipo $\delta(q,a)=q_1$ y $\delta(q,a)=q_2$, siendo $q_1 \neq q_2$;
- Que existan transiciones del tipo $\delta(q,\varepsilon)$, siendo q un estado no-final, o bien un estado final pero con transiciones hacia otros estados.

Cuando se cumple el segundo caso, se dice que el autómata es un autómata finito no determinista con transiciones vacías o transiciones ε (abreviado AFND- ε). Estas transiciones permiten al autómata cambiar de estado sin procesar ningún símbolo de entrada.

Ejemplo: Obtenga un AFND dado el siguiente lenguaje definido en el alfabeto $\Sigma = \{0,1\}$. El conjunto de cadenas que inician en "0".

SOLUCIÓN



Teorema de Myhill-Nerode

El teorema de Myhill-Nerode proporciona una condición necesaria y suficiente para que un lenguaje sea regular. El teorema lleva el nombre de John Myhill y Anil Nerode, quienes lo demostraron en la Universidad de Chicago en 1958.

El teorema de Myhill-Nerode se puede utilizar para demostrar que un lenguaje L es regular demostrando que el número de clases de equivalencia de RL es finito. Esto puede hacerse mediante un análisis de caso exhaustivo en el que, a partir de la cadena vacía, se utilizan extensiones distintivas para encontrar clases de equivalencia adicionales hasta que no se puedan encontrar más.

Minimización

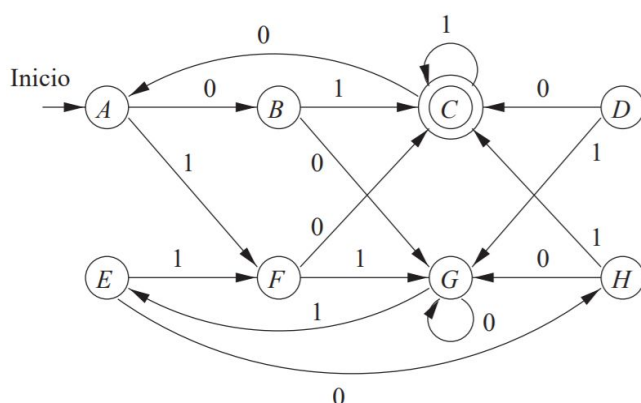
Un AFD está minimizado, si todos sus estados son distinguibles y alcanzables. Un algoritmo de minimización de AFD es el siguiente:

- Eliminar los estados inaccesibles del autómata.
- Construir una tabla con todos los pares (p, q) de estados restantes.
- Marcar en la tabla aquellas entradas donde un estado es final y el otro es no-final, es decir, aquellos pares de estados que son claramente distinguibles.
- Para cada par (p, q) y cada símbolo a del alfabeto, tal que $r = \delta(p, a)$ y $s = \delta(q, a)$:

- Si (r, s) ya ha sido marcado, entonces p y q también son distinguibles, por lo tanto marcar la entrada (p, q) .
- De lo contrario, colocar (p, q) en una lista asociada a la entrada (r, s) .
- Agrupar los pares de estados no marcados.

Luego del tercer paso, si la tabla creada queda completamente marcada, entonces el AFD inicial ya era mínimo.

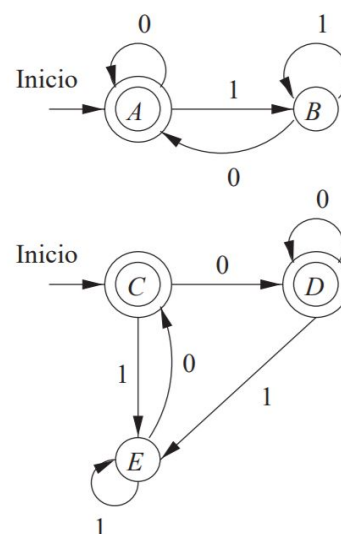
Ejemplos:



B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

Consideremos la tabla de la Figura anterior, donde hemos determinado los estados equivalentes y distinguibles correspondientes al autómata mostrado anteriormente. La partición de los estados en bloques equivalentes es $\{A, E\}$, $\{B, H\}$, $\{C\}$, $\{D, F\}$, $\{G\}$. Observe que los tres pares de estados que son equivalentes se incluyen en un mismo bloque, mientras que los estados que son distinguibles de los restantes estados se incluyen cada uno en un bloque distinto.

Para el siguiente autómata, la partición es $\{A, C, D\}$, $\{B, E\}$. Este ejemplo muestra que podemos tener más dos estados en un bloque. Parece fortuito que A , C y D puedan estar en un mismo bloque, porque todos los pares que forman entre los tres son equivalentes, y ninguno de ellos es equivalente a cualquier otro estado. Sin embargo, como veremos en el siguiente teorema que vamos a demostrar, esta situación está garantizada por la definición de "equivalencia" de estados.



Conversión de RE a FA

Para convertir el RE en FA, usaremos un método llamado método de subconjunto. Este método se utiliza para obtener FA a partir de la expresión regular dada. Este método se da a continuación:

Paso 1: Diseñe un diagrama de transición para una expresión regular dada, utilizando NFA con movimientos ϵ .

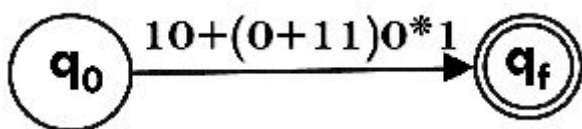
Paso 2: Convierta este NFA con ϵ en NFA sin ϵ .

Paso 3: Convierta el NFA obtenido en DFA equivalente.

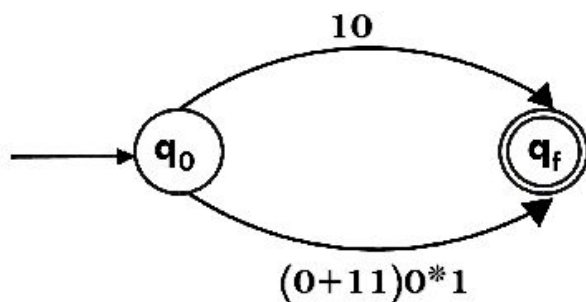
Ejemplo 1: Diseñe un FA a partir de la expresión regular dada $10 + (0 + 11)0^*1$.

Solución: Primero construiremos el diagrama de transición para una expresión regular dada.

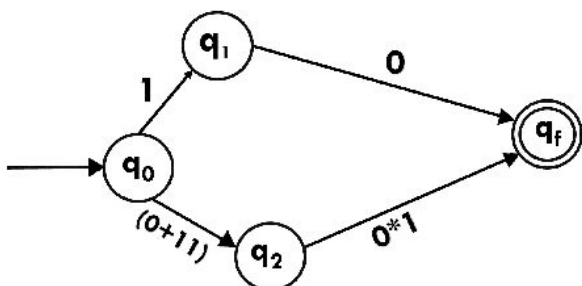
Paso 1:



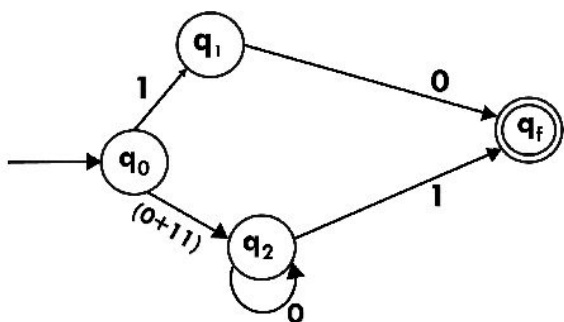
Paso 2:



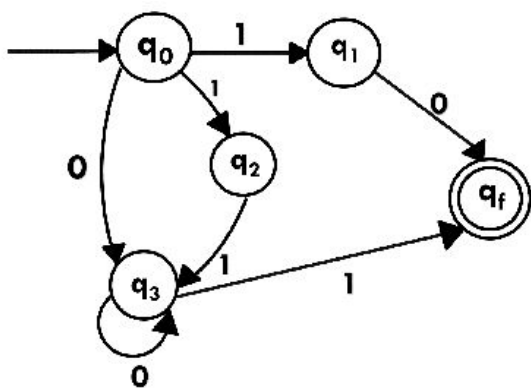
Paso 3:



Paso 4:



Paso 5:



Ahora tenemos NFA sin ϵ . Ahora lo convertiremos en DFA requerido para eso, primero escribiremos una tabla de transición para este NFA.

Estado	0	1
$\rightarrow q_0$	q_3	$\{q_1, q_2\}$
q_1	q_f	\emptyset
q_2	\emptyset	q_3
q_3	q_3	q_f
$* q_f$	\emptyset	\emptyset

El DFA equivalente será:

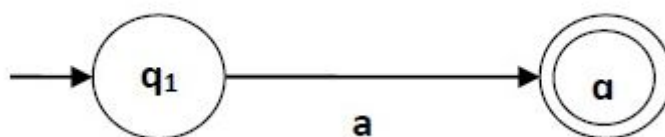
Estado	0	1
$\rightarrow [q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_1]$	$[q_f]$	ϕ
$[q_2]$	ϕ	$[q_3]$
$[q_3]$	$[q_3]$	$[q_f]$
$[q_1, q_2]$	$[q_f]$	$[q_f]$
$* [q_f]$	ϕ	ϕ

Conversión de FA a RE

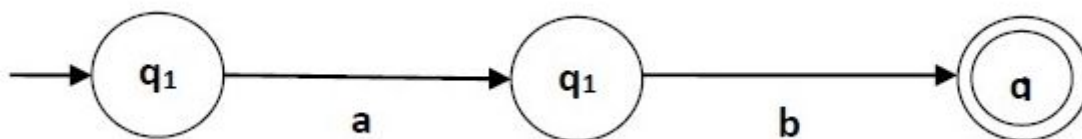
Podemos usar la construcción de Thompson para encontrar un autómata finito a partir de una expresión regular. Reduciremos la expresión regular a expresiones regulares más pequeñas y las convertiremos a NFA y finalmente a DFA.

Algunas expresiones RA básicas son las siguientes:

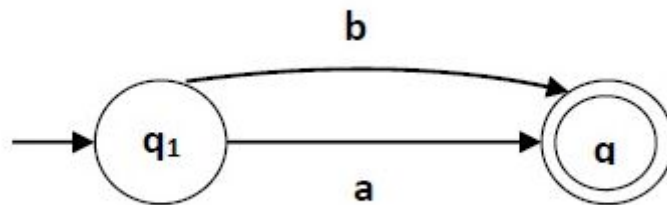
Caso 1 - Para una expresión regular 'a', podemos construir el siguiente FA -



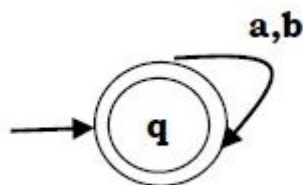
Caso 2 - Para una expresión regular 'ab', podemos construir el siguiente FA -



Caso 3 - Para una expresión regular (a + b), podemos construir el siguiente FA -



Caso 4 - Para una expresión regular $(a + b)^*$, podemos construir el siguiente FA -



Método

Paso 1: Construya un NFA con movimientos nulos a partir de la expresión regular dada.

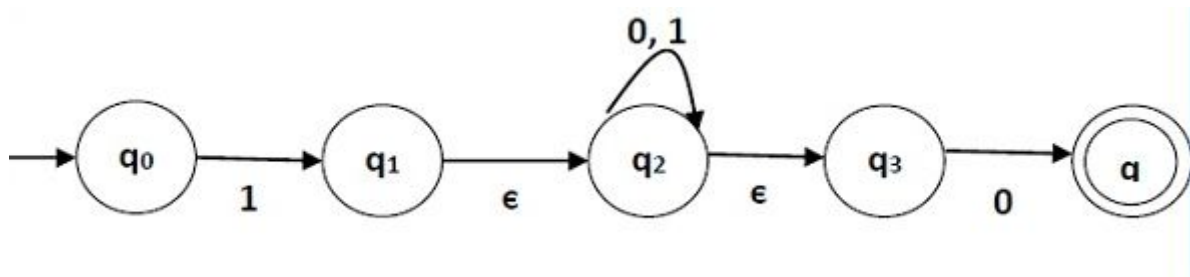
Paso 2: Elimine la transición nula de la NFA y conviértela en su DFA equivalente.

Problema

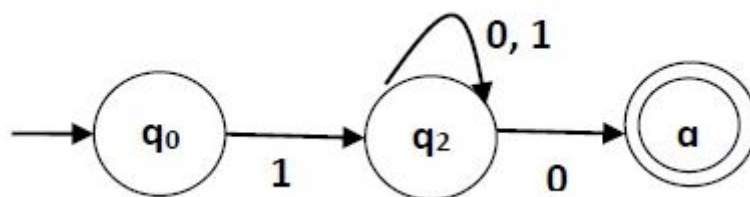
Convierta el siguiente RA en su DFA equivalente - $1(0 + 1)^*0$

Solución

Concatenamos tres expresiones "1", " $(0 + 1)^*$ " y "0"



Ahora eliminaremos las transiciones ϵ . Después de eliminar las transiciones ϵ del NDFA, obtenemos lo siguiente:

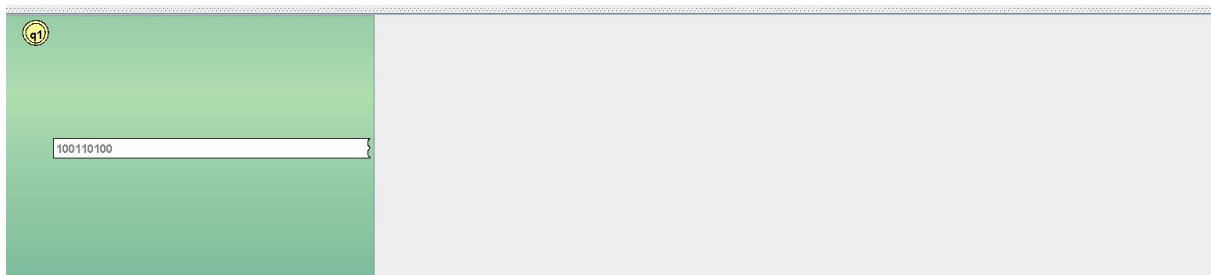
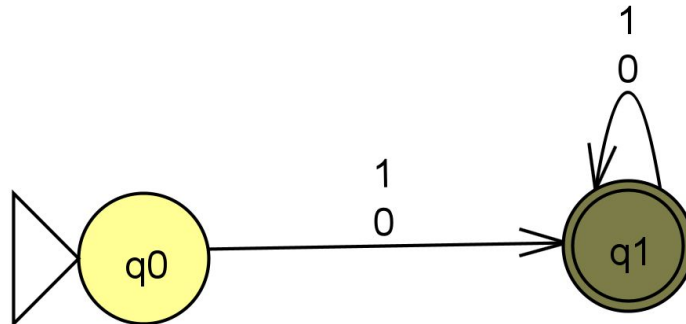


Parte práctica

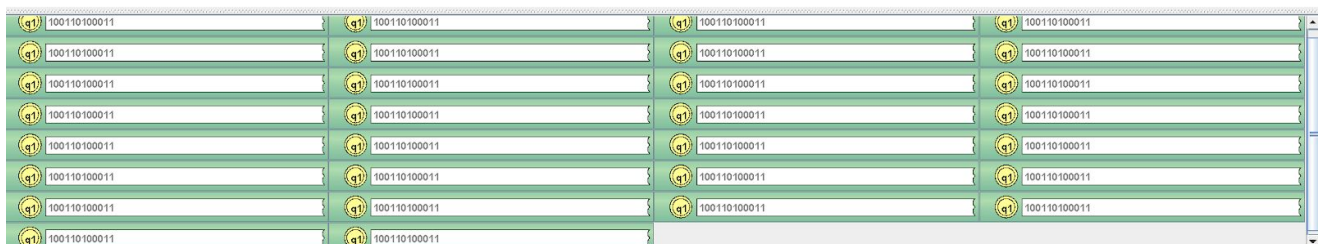
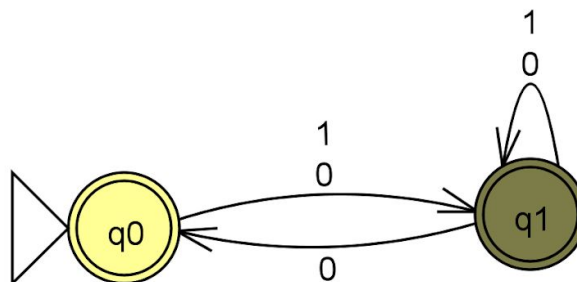
Ejercicios - Capítulo 2

2.2.4. Describa los AFD que acepten los siguientes lenguajes con el alfabeto $\{0,1\}$:

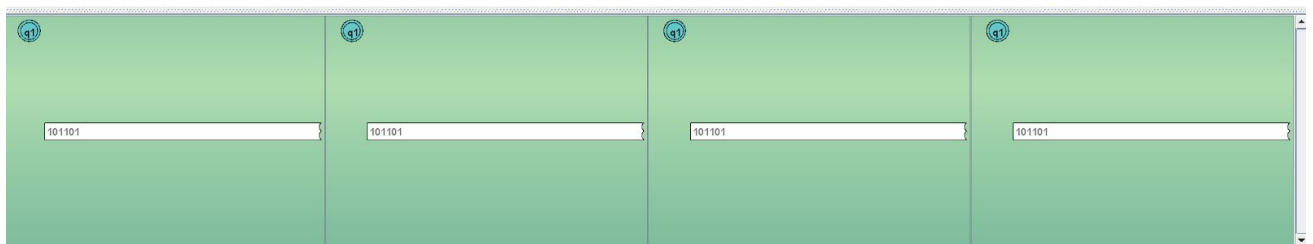
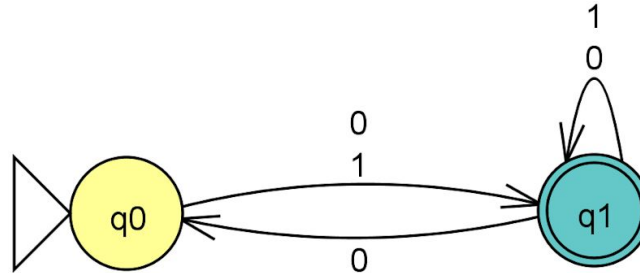
- a) El conjunto de todas las cadenas que terminan en 00.



- b) El conjunto de todas las cadenas con tres ceros consecutivos (no necesariamente al final).

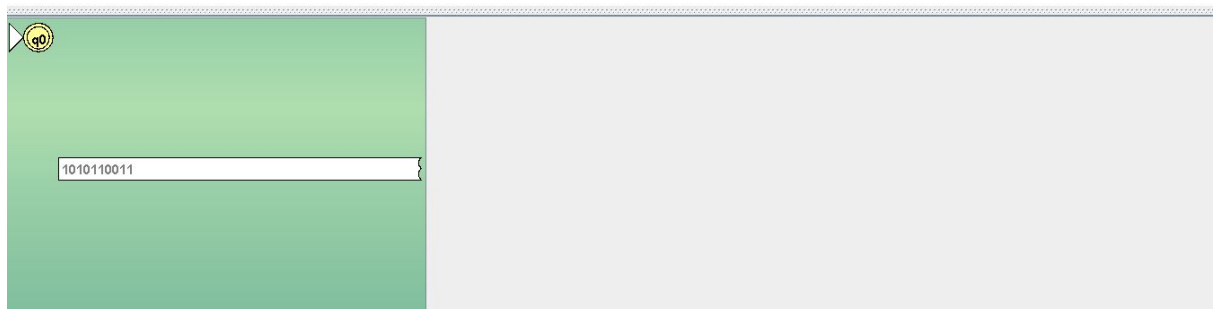
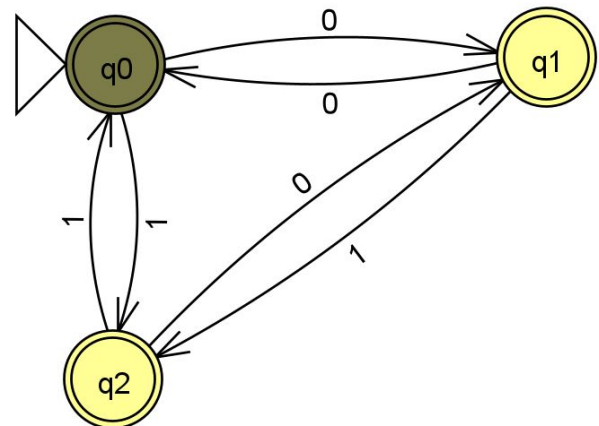


c) El conjunto de todas las cadenas que contengan la subcadena 011.

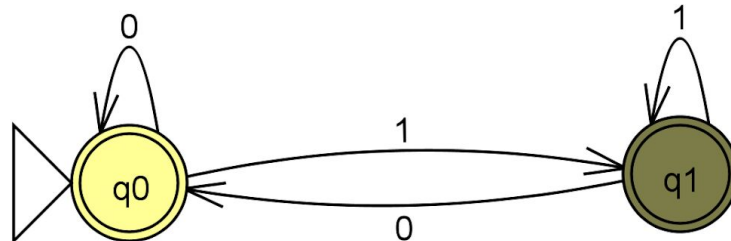


2.2.5. Describa los AFD que acepta los siguientes lenguajes con el alfabeto {0,1}:

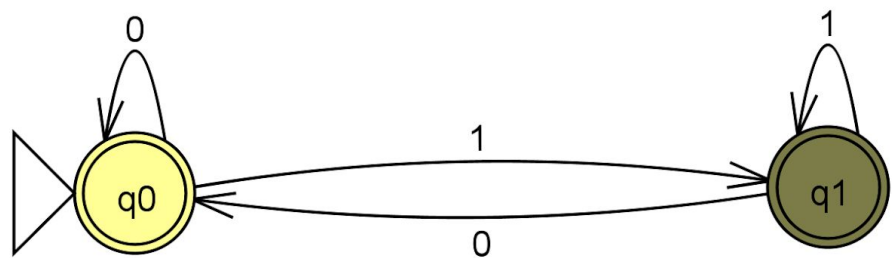
a) El conjunto de todas las cadenas tales que cada bloque de cinco símbolos consecutivos contenga al menos dos ceros.



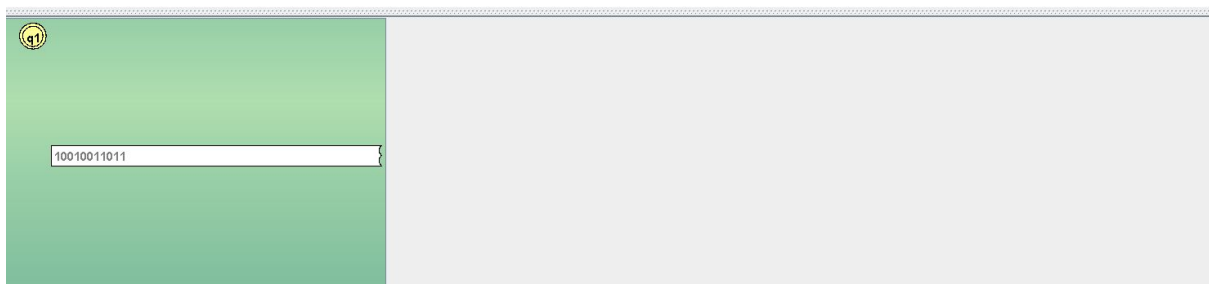
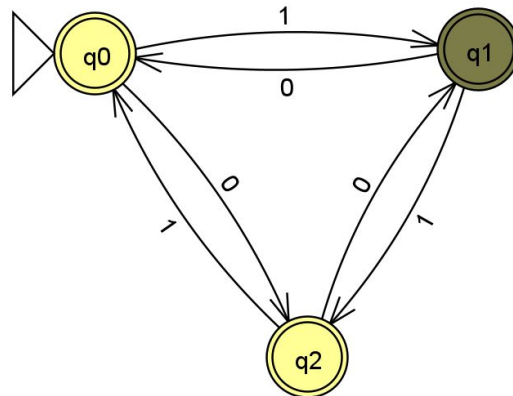
- b) El conjunto de todas las cadenas cuyo símbolo en la décima posición respecto del extremo derecho sea un 1.



- c) El conjunto de cadenas que empiecen o terminen (o ambas cosas) con 01.

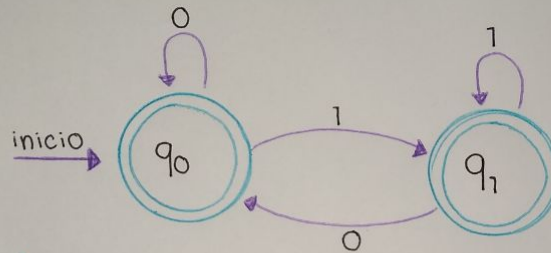


- d) El conjunto de las cadenas tales que el número de ceros es divisible por cinco y el número de unos es divisible por 3.



2.2.7. Sea A un AFD y q un estado concreto de A , tal que $\delta(q, a) = q$ para todos los símbolos a de entrada. Demuestre por inducción sobre la longitud de la entrada que para todas las cadenas de entrada w , se cumple que $\delta(q, w) = q$.

q_0 = empiecen o terminen en 01



$\hat{\delta}(q_0, w)$ para prefijo w de 101001 comenzando por ϵ y aumentando progresivamente.

- $\hat{\delta}(q_0, \epsilon) = q_0$
- $\hat{\delta}(q_0, 1) = \delta(\hat{\delta}(q_0, \epsilon), 1) = \delta(q_0, 1) = q_1$
- $\hat{\delta}(q_0, 10) = \delta(\hat{\delta}(q_0, 1), 0) = \delta(q_1, 0) = q_0$
- $\hat{\delta}(q_0, 101) = \delta(\hat{\delta}(q_0, 10), 1) = \delta(q_0, 1) = q_1$
- $\hat{\delta}(q_0, 1010) = \delta(\hat{\delta}(q_0, 101), 0) = \delta(q_1, 0) = q_0$
- $\hat{\delta}(q_0, 10100) = \delta(\hat{\delta}(q_0, 1010), 0) = \delta(q_0, 0) = q_0$
- $\hat{\delta}(q_0, 101001) = \delta(\hat{\delta}(q_0, 10100), 1) = \delta(q_0, 1) = q_1$

$$\begin{aligned}\hat{\delta}(q, a) &= q \\ \delta(q, a) &= \delta(\hat{\delta}(q, a), a) = \delta(q, a) = q \\ \hat{\delta}(q, w) &= \delta(\hat{\delta}(q, w), w) = \delta(q, w) = q \\ \hat{\delta}(q, \epsilon) &= q \\ \delta(q, w) &= \delta(\hat{\delta}(q, \epsilon), w) = \delta(q, w) = q\end{aligned}$$

$$\begin{aligned}w &= w_1 w_2 \\ \delta(q, w) &= q \\ \hat{\delta}(q, w_1) &= \delta(\hat{\delta}(q, w), w_1) = \delta(q, w_1) = q \\ \hat{\delta}(q, w_1 w_2) &= \delta(\hat{\delta}(q, w_1), w_2) = \delta(q, w_2) = q\end{aligned}$$

2.2.8. Sea A un AFD y a un símbolo de entrada particular de A , tal que para todos los estados q de A tenemos que $\delta(q, a) = q$.

- a) Demuestre por inducción sobre n que para todo $n \geq 0$, $\delta(q, a^n) = q$, donde a^n es la cadena formada por n símbolos a .

Ejercicio: 2.2.8

$$\delta(q, a) = q \quad a^n = 101110...$$

$$\hat{\delta}(q_0, \epsilon) = q_0$$

$$\hat{\delta}(q_0, 1) = \delta(\hat{\delta}(q_0, \epsilon), 1) = \delta(q_0, 1) = q_1$$

$$\hat{\delta}(q_0, 10) = \delta(\hat{\delta}(q_0, 1), 0) = \delta(q_1, 0) = q_0$$

$$\hat{\delta}(q_0, 101) = \delta(\hat{\delta}(q_0, 10), 1) = \delta(q_0, 1) = q_1$$

$$\hat{\delta}(q_0, 1011) = \delta(\hat{\delta}(q_0, 101), 1) = \delta(q_1, 1) = q_1$$

$$\hat{\delta}(q_0, 10111) = \delta(\hat{\delta}(q_0, 1011), 1) = \delta(q_1, 1) = q_1$$

$$\hat{\delta}(q_0, 101110) = \delta(\hat{\delta}(q_0, 10111), 0) = \delta(q_1, 0) = q_0$$

- b) Demuestre que $\{a\}^* \subseteq L(A)$ o $\{a\}^* \cap L(A) = \emptyset$.

b) $\hat{\delta}(q_0, \epsilon) = \{q_0\}$

$$\hat{\delta}(q_0, a) = \delta(q_0, a) = \{q_0, q_1\}$$

$$\hat{\delta}(q_0, a^*) = \delta(q_0, a) \cup \delta(q_1, a) = \{q_0, q_1, q_2\} = \emptyset$$

$$\{q_0, q_1\}$$

$$\hat{\delta}(q_0, a^*) = \delta(q_0, a) \cup \delta(q_1, a) = \{q_0, q_1, q_2\} \subseteq \emptyset$$

$$\emptyset = \{q_0, q_1\}$$

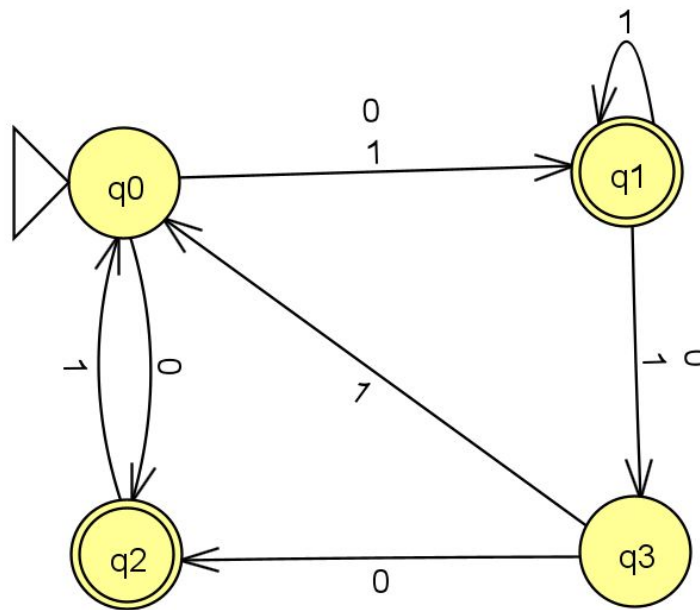
$$\hat{\delta}(q_0, a^*) = \delta(q_0, a) \cup \delta(q_1, a) = \{q_0, q_1, q_2\} \subseteq \emptyset$$

$$= \{q_0, q_1\}$$

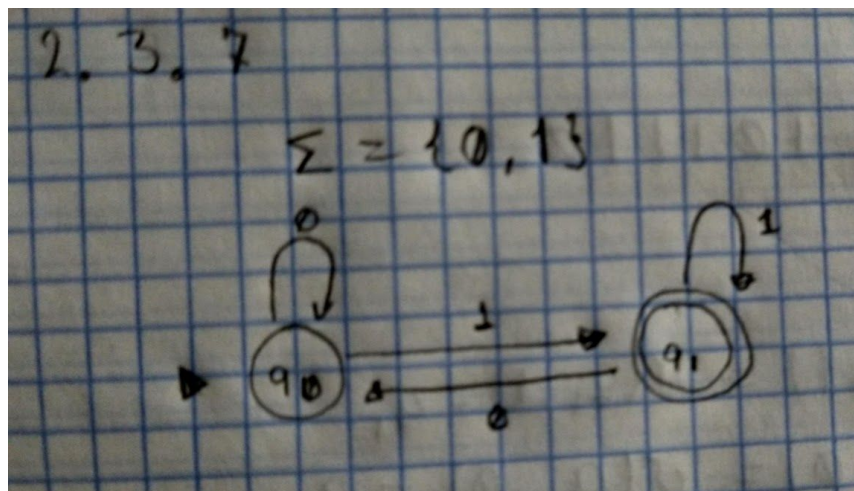
$$\hat{\delta}(q, a^*) = \delta(q_0, a) \cup \delta(q_1, a) = \{q_0\} = \{q_0, q_1\}$$

2.3.2. Convierta en un AFD el siguiente AFN:

	0	1
$\rightarrow p$	$\{q, s\}$	$\{q\}$
$*q$	$\{r\}$	$\{q, r\}$
r	$\{s\}$	$\{p\}$
$*s$	\emptyset	$\{p\}$

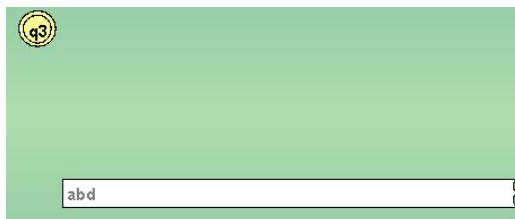
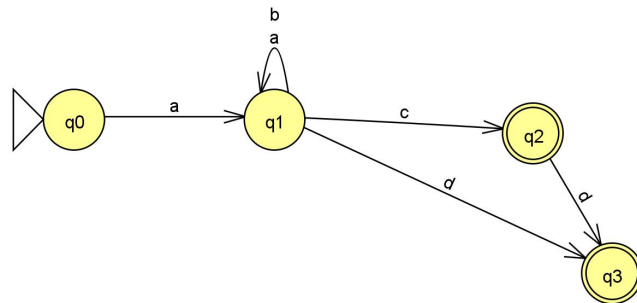
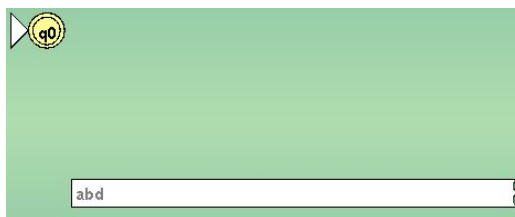
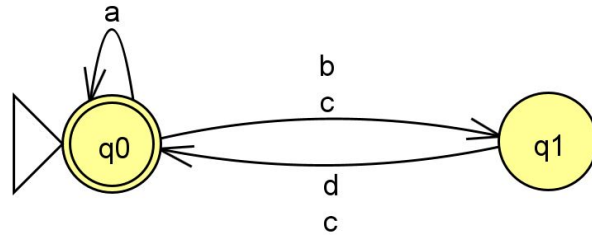
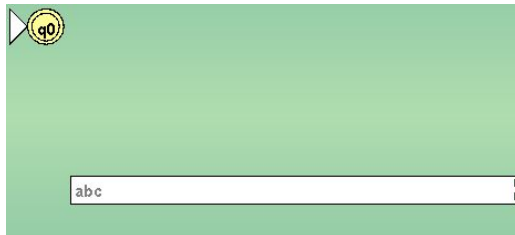


2.3.7. En el *Ejemplo 2.13* hemos establecido que el AFN N se encuentra en el estado q_i , para $i = 1, 2, \dots, n$, después de leer la secuencia de entrada w si y sólo si el símbolo i -ésimo del final de w es 1. Demuestre esta afirmación.

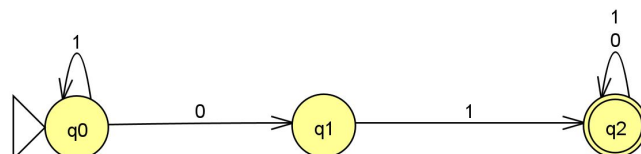


2.4.1. Diseñe un AFN para reconocer los siguientes conjuntos de cadenas.

a) abc, abd y $aacd$. Suponga que el alfabeto es $\{a, b, c, d\}$.



b) $0101, 101$ y 011 .



q2

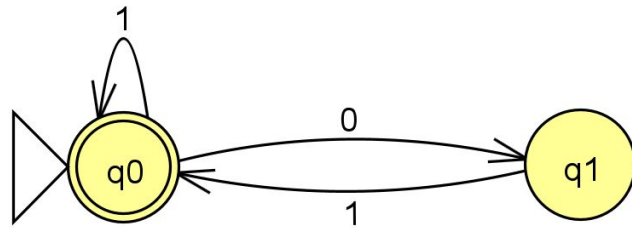
101

q2

011

q0

0101



q0

101

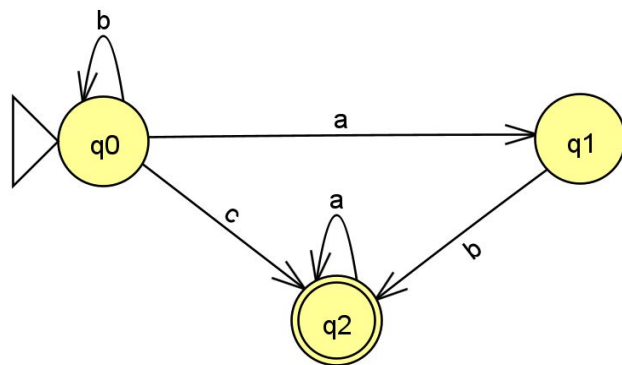
q0

011

c) ab, bc y ca . Suponga que el alfabeto es $\{a,b,c\}$.

q1

ab

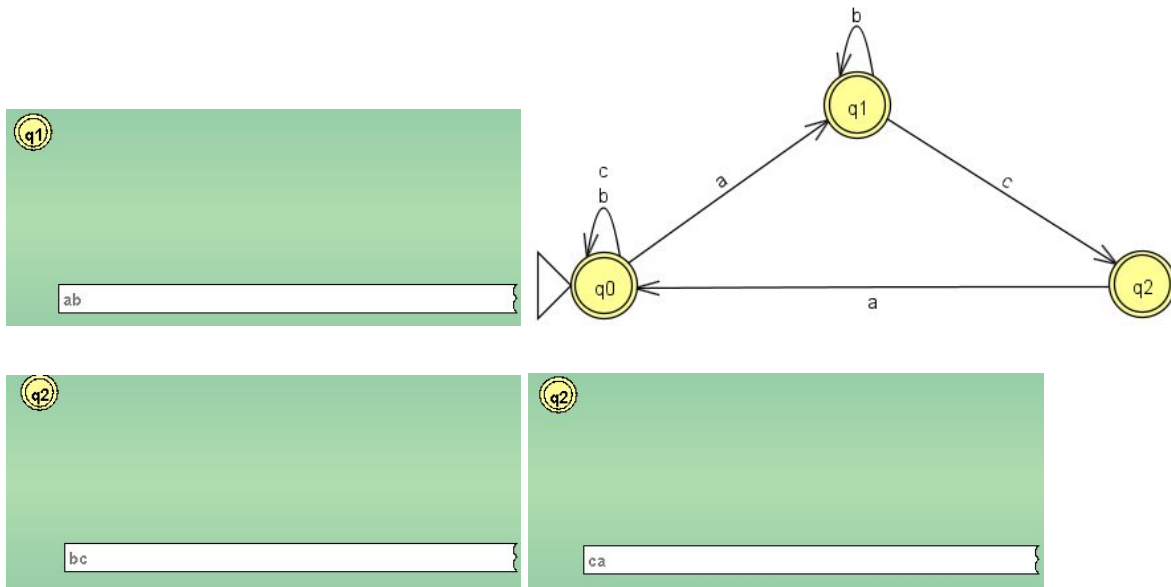


q2

bc

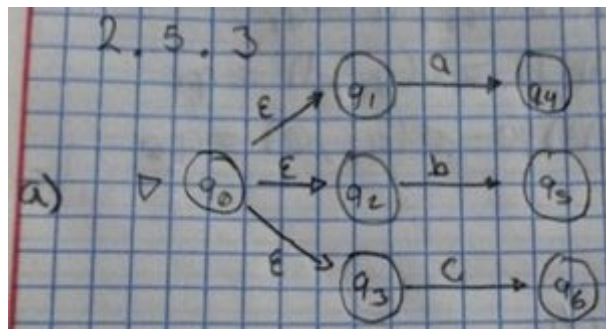
q2

ca

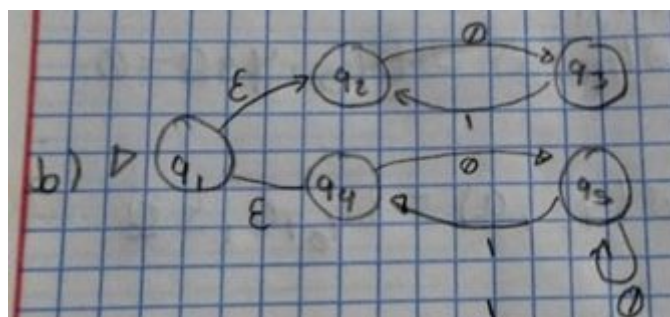


2.5.3. Diseñe un AFN- ϵ para cada uno de los siguientes lenguajes. Intente emplear transiciones- ϵ para simplificar su diseño.

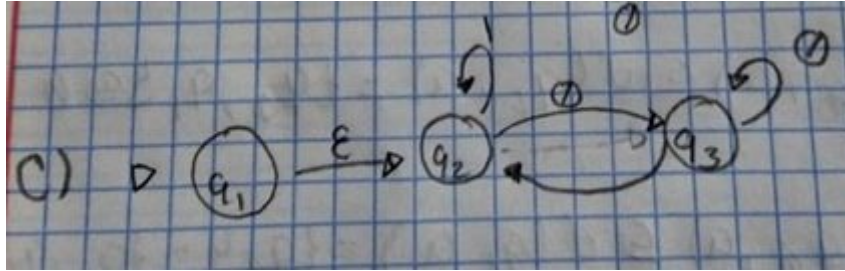
- a) El conjunto de cadenas formado por cero o más letras a seguidas de cero o más letras b, seguida de cero o más letras c.



- b) El conjunto de cadenas que constan de la subcadena 01 repetida una o más veces o de la subcadena 010 repetida una o más veces.



- c) El conjunto de cadenas formadas por ceros y unos tales que al menos una de las diez posiciones es un 1.



Ejercicios - Capítulo 3

3.1.1. Escriba expresiones regulares para los siguientes lenguajes:

- a) El conjunto de cadenas del alfabeto {a,b,c} que contienen al menos una a y al menos una b.

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

- b) El conjunto de cadenas formadas por 0s y 1s cuyo décimo símbolo por la derecha sea 1.

b) $(0^*11^*0)^*$

- c) El conjunto de cadenas formadas por 0s y 1s con al menos una pareja de 1s consecutivos.

c) $(0+10)^*(11+\epsilon)(0+10)^*$

3.1.2. Escriba expresiones regulares para los siguientes lenguajes:

- a) El conjunto de todas las cadenas formadas por ceros y unos tales que cada pareja de 0s adyacentes aparece antes que cualquier pareja de 1s adyacentes.

$$a) (10+0)^*(01+1)^*(\epsilon+1)$$

- b) El conjunto de cadenas formadas por ceros y unos cuyo número de ceros es divisible por cinco.

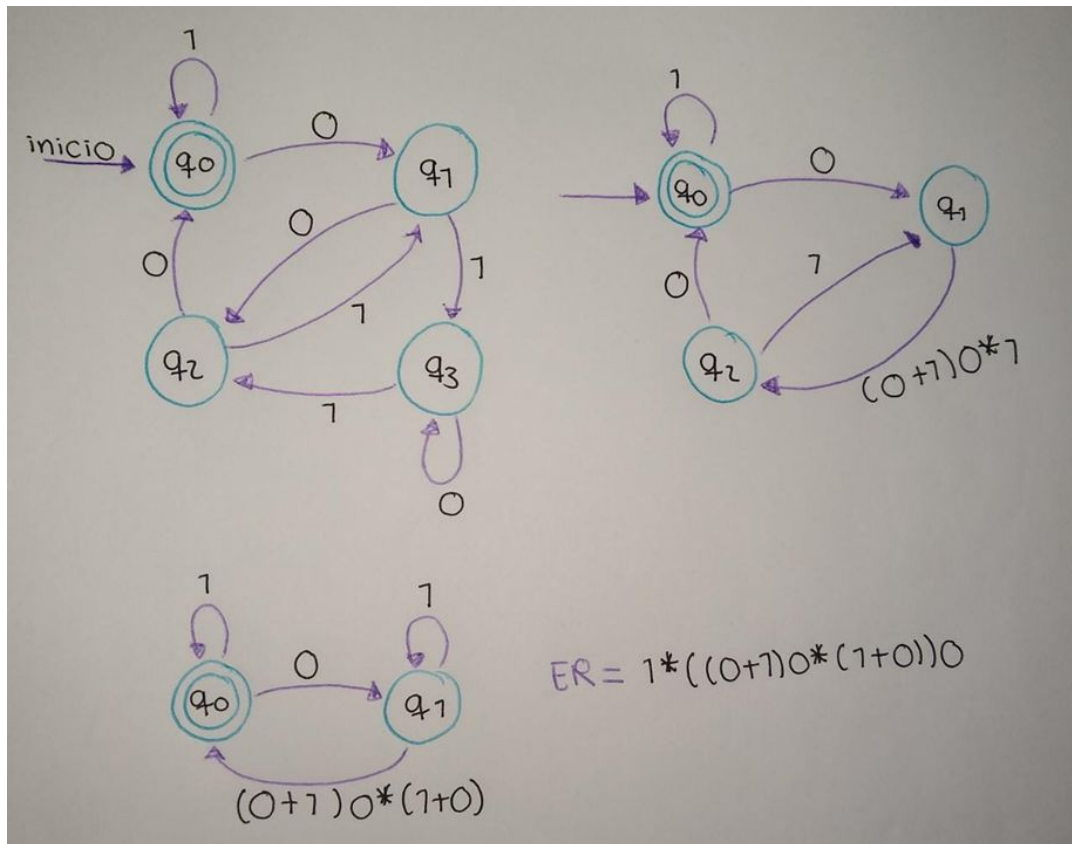
$$b) 0(5)+11*0(5)$$

3.1.4. Proporcione las descripciones informales de los lenguajes correspondientes a las siguientes expresiones regulares:

- a) $(1+\epsilon)(00^*1)^*0^*$. El conjunto de todas las cadenas formadas por 0s y 1s que empiecen con 1 y contenga al menos 1 par de 0s antes de un 1.
- b) $(0^*1^*)^*000(0+1)^*$. El conjunto de todas las cadenas formadas por 0s y 1s cuyo números de 0s sea impar.
- c) $(0+10)^*1^*$. El conjunto de todas las cadenas formadas por 0s y 1s que empiecen con un 0 o 1 y terminen en 01.

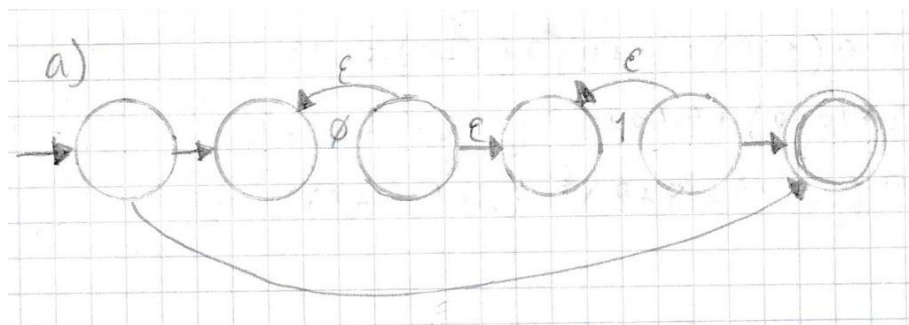
3.2.3. Convierta el siguiente AFD en una expresión regular utilizando la técnica de eliminación de estados de la Sección 3.2.2.

	0	1
$\rightarrow *p$	s	p
q	p	s
r	r	q
s	q	r

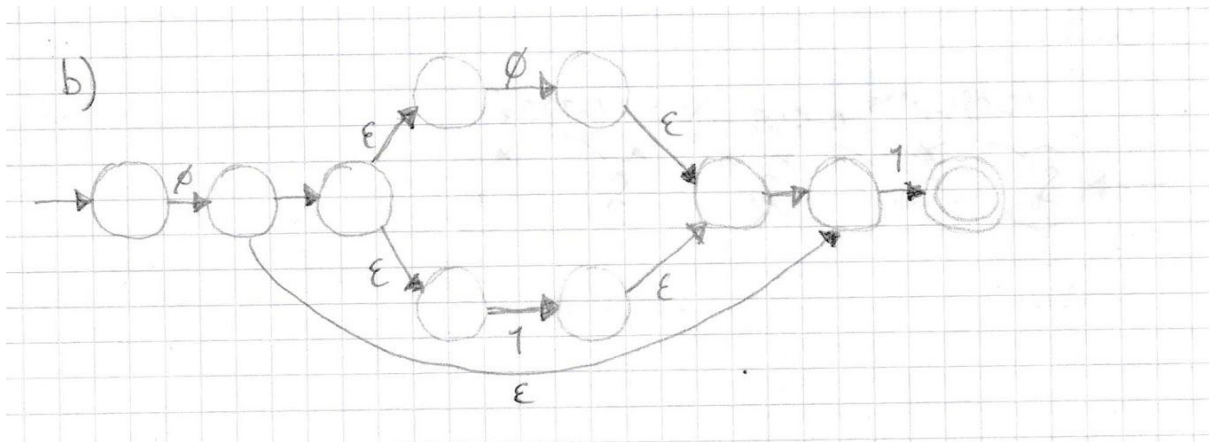


3.2.4. Convierta las siguientes expresiones regulares en un AFN con transiciones- ϵ .

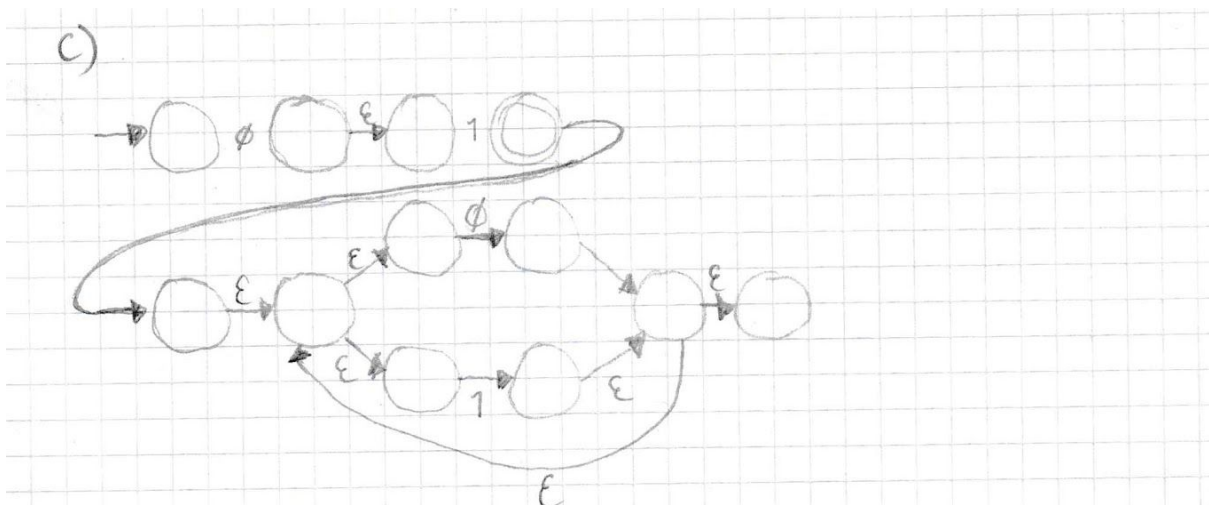
a) 01^* .



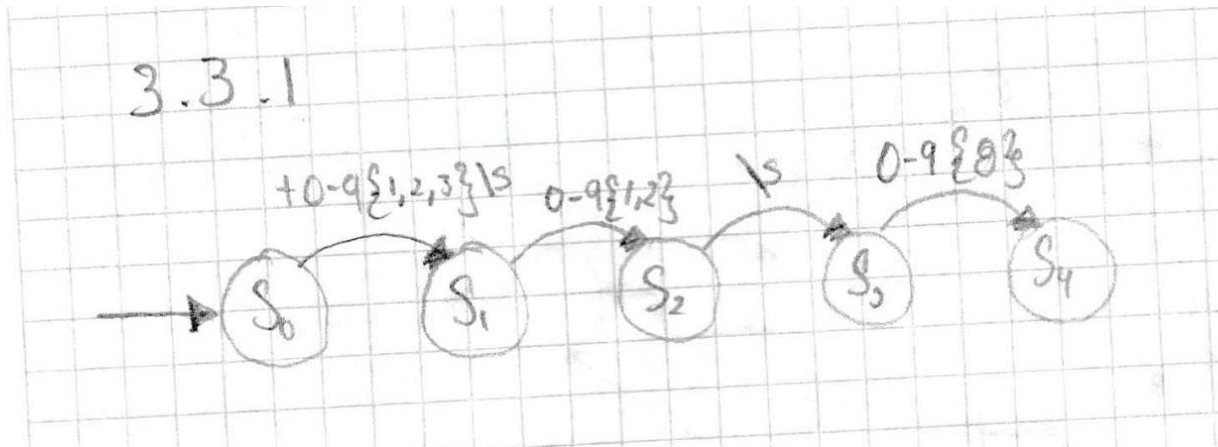
b) $(0+1)01$.



c) $00(0+1)^*$.



3.3.1. Obtenga una expresión regular que describa números de teléfono en todas las variantes que pueda imaginar. Tenga en cuenta los números internacionales así como el hecho de que los distintos países emplean una cantidad diferente de dígitos para los códigos de área de los números de teléfono locales.



3.4.2. Demuestre si cada una de las siguientes proposiciones acerca de expresiones regulares es verdadera o falsa.

- a) $(R+S)^* = R^* + S^*$. -Es verdadera
- b) $(RS+R)^*R = R(SR+R)^*$. -Es verdadera
- c) $(RS+R)^*RS = (RR^*S)^*$. -Es falsa
- d) $(R+S)^*S = (R^*S)^*$. -Es verdadera
- e) $S(RS+S)^*R = RR^*S(RR^*S)^*$. -Es falsa

3.4.3. En el Ejemplo 3.6 hemos desarrollado la expresión regular

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

3.4.3

$$(S+R)^*R(S+R)+(S+R)^*R(S+R)(S+R)$$

1) $R(R+S)^*(S+R)+R(R+S)^*(S+R)(S+R)$

2) $(S+1)R(S+1)+(S+R)^*R(S+1)(S+1)$



Anónimo. (S.F). Árboles de derivación. Recuperado el 26 de octubre de 2020, de <ftp://decsai.ugr.es/pub/utai/other/smc/docencia/tr4.pdf>

Wikipedia. (2020). Gramática Regular. Recuperado el 02 de noviembre de 2020, de https://es.wikipedia.org/wiki/Gram%C3%A1tica_regular#:~:text=En%20inform%C3%A1tica%20una%20gram%C3%A1tica%20regular,finitos%20y%20las%20expresiones%20regulares.

LetySystem's. (S.F). Gramática Regular. Recuperado el 02 de noviembre de, <https://letysystem.wordpress.com/gramatica-regular/>

Wikipedia. (2020). Autómata finito. Recuperado el 02 de noviembre de 2020. De [https://es.wikipedia.org/wiki/Aut%C3%B3mata_finito#:~:text=Un%20aut%C3%B3mata%20finito%20\(AF\)%20o,un%20conjunto%20de%20estados%20finales.](https://es.wikipedia.org/wiki/Aut%C3%B3mata_finito#:~:text=Un%20aut%C3%B3mata%20finito%20(AF)%20o,un%20conjunto%20de%20estados%20finales.)

Ojeda Toche L. (2017). Autómatas deterministas y no deterministas (Ejercicios). Recuperado el 02 de noviembre de 2020, de <https://core.ac.uk/download/pdf/154797605.pdf>

Wikipedia. (2020). Myhill-Nerode theorem. Recuperado el 05 de noviembre de 2020, de https://en.wikipedia.org/wiki/Myhill%E2%80%93Nerode_theorem

Anónimo. (S.F). Minimización de estados. Recuperado el 05 de noviembre de 2020, de <https://sites.google.com/site/tautomatasfinal/minimizaci>

Anónimo. (S.F). Conversion of RE to FA. Recuperado el 12 de noviembre de 2020, de <https://www.javatpoint.com/automata-conversion-of-re-to-fa#:~:text=To%20convert%20the%20RE%20to,using%20NFA%20with%20CE%20moves.>

Anónimo. (S.F). Construction of an FA from an RE. Recuperado el 16 de noviembre de 2020, de https://www.tutorialspoint.com/automata_theory/constructing_fa_from_re.htm





Hopcroft, J., Motwani, R. and Ullman, J., 2008. *Introducción A La Teoría De Autómatas Lenguajes Y Computación*. 3rd ed. Madrid: Miguel Martín-Romo.

