Notas de Lenguajes Formales y Autómatas

Luis Eduardo Gamboa Guzmán

Universidad Michoacana de San Nicolás de Hidalgo
Facultad de Ingeniería Eléctrica

19 de mayo de 2021

Índice general

Sobre este documento

Este documento es una recopilación de conceptos y ejercicios obtenidos mayormente de [Hopcroft 2001] y [Hopcroft 1979]. Los cambios realizados incluyen traducción, notación, reordenamiento y expansión del material para hacerlo más accesible.

Muchos ejercicios han sido desarrollados enteramente por el autor de esta recopilación. Otros tantos han sido modificados de los expuestos en la bibliografía para hacerlos más claros y utilizando los conceptos que abarca el curso.

Este documento ha sido desarrollado utilizando LaTeX.

Capítulo 1

Lenguajes Regulares

1.1. Introducción

La teoría de autómatas es el estudio de dispositivos de cómputo abstractos o "máquinas".

Antes de que hubiera computadoras, Alan Turing estudio en 1930 una máquina abstracta que tenía todas las capacidades de las computadoras de hoy, al menos en lo referente a lo que pueden computar. *Barrera entre lo que puede hacer y no.*

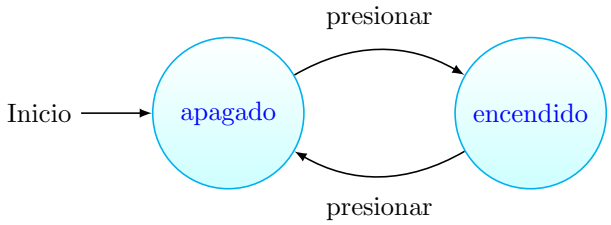
En los años 1940s y 1950s máquinas más sencillas que hoy llamamos *autómata finito*, fueron estudiadas con el fin de modelar funciones cerebrales, pero resultaron sumamente útiles para otros propósitos:

- Software para diseñar y verificar el comportamiento de circuitos digitales.
- Analizador léxico de un compilador
- Software para búsqueda en texto, incluyendo páginas web.
- Software para verificar sistemas.

También en los años 1950s, Noam Chomsky comenzó el estudio de *gramáticas*, que no son máquinas estrictamente hablando, pero sí tienen una relación muy cercana con los autómatas y son la base de componentes de software importantes como los compiladores.

1.2. Definiciones

Un sistema puede ser visto como un conjunto finito de "estados".



Estados: apagado, encendido
Estado inicial: apagado
Arcos: entrada al sistema

Alfabetos

Un alfabeto es un conjunto finito no vacío de símbolos, normalmente denotado por Σ:

$$\Sigma = \{0, 1\} \qquad \text{(binario)}$$
$$\Sigma = \{a, b, c, d, e, \ldots, z\} \qquad \text{(letras minúsculas)}$$

Cadenas

Una cadena o "palabra"'es una secuencia finita de símbolos de algún alfabeto. Ejemplo: 01101 es una cadena del alfabeto $\Sigma = \{0, 1\}$, 111 es otra cadena de ese alfabeto.

Cadena vacía

Es una cadena con cero ocurrencias de símbolos. Se denota ε y es una cadena de cualquier alfabeto.

Tamaño de cadena

Número de posiciones para símbolos en la cadena. 01101 tiene longitud 5. Es coloquialmente aceptado el uso de "número de símbolos" a pesar de ser incorrecto. En la cadena 01101 sólo hay 2 símbolos pero 5 posiciones para símbolos. La longitud de la cadena w se denota $|w|$, de este modo $|011| = 3, |abcd| = 4, |01101| = 5$ y $|\varepsilon| = 0$

Potencias de un alfabeto

Si Σ es un alfabeto, podemos expresar el conjunto de todas las cadenas de cierta longitud utilizando notación exponencial. Así Σ^k es el conjunto de cadenas

de longitud k para el alfabeto Σ. Si $\Sigma = \{0, 1\}$ entonces:

$$\Sigma^0 = \{\varepsilon\} \neq \varepsilon$$
$$\Sigma^1 = \{0, 1\} \neq \Sigma$$
$$\Sigma^2 = \{00, 01, 10, 11\}$$
$$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

Cerradura de Kleene

El conjunto de todas las cadenas sobre un alfabeto Σ se denota Σ^* y está definido como:

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

Ejemplo:

$$\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$$

Ejercicios:

- $\{a\}^* =$
- $\{a, b\}^* =$
- $\{a, b, c\}^* =$

Cerradura Positiva

Denotada por Σ^+, es el conjunto de todas las cadenas sobre un alfabeto excluyendo la cadena vacía:

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

de donde se deduce la propiedad $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

Concatenación

Sean x e y cadenas, entonces xy denota la concatenación de x e y, esto es, la cadena formada por una copia de x seguida de una copia de y:

$$\text{Si } x = a_1a_2a_3\cdots a_i, \quad y = b_1b_2b_3\cdots b_j, \text{entonces } xy = a_1a_2a_3\cdots a_ib_1b_2b_3\cdots b_j$$

Para cualquier cadena w, se cumple $\varepsilon w = w\varepsilon = w$, ε es el elemento identidad (o elemento neutro) para la concatenación (así como 0 lo es para la suma $x+0 = x$).

Lenguajes

Un conjunto de cadenas escogidas de Σ^* para algún alfabeto Σ en particular es llamado lenguaje. Formalmente, si Σ es un alfabeto y se cumple que $L \subseteq \Sigma^*$, entonces L es un lenguaje sobre Σ.

Ejemplos:

1. El lenguaje de todas las cadenas de n ceros seguidas por n unos para $n \geq 0$, esto es, $\{\varepsilon, 01, 0011, 000111, \ldots\} = \{0^n 1^n \mid n \geq 0\}$

2. Las cadenas con igual número de 0s y 1s: {ε,01, 10, 0011, 0101, 1001, 1010, 1100, ...}

3. Σ^* es un lenguaje sobre cualquier alfabeto Σ.

4. $\emptyset$, el lenguaje vacío, es un lenguaje sobre cualquier alfabeto. Hay que resaltar que $\emptyset \neq \{\varepsilon\}$ ya que $\emptyset$ es un lenguaje que no tiene cadenas y $\{\varepsilon\}$ es un lenguaje con una cadena.

Problema: dada una cadena w en Σ^*, decidir si pertenece o no a L.

- La respuesta es tan simple como sí o no. Sin embargo, en ocasiones se quiere computar o transformar una entrada.
  Ejemplo: Un compilador de C recibe una "cadena" ASCII y se le pide que decida si la cadena pertenece o no a L_C (conjunto de programas en C válidos). Sin embargo, no sólo decide, sino también crea tablas de símbolos, árboles y convierte a código máquina.

- Especial importancia a técnicas que prueban que ciertos problemas no pueden ser resueltos en menos tiempo que exponencial respecto al tamaño de la entrada.

Resulta que determinar "sí o no" es tan difícil como "resolver": si es difícil determinar si $w \in L_x$ entonces no es fácil traducir de L_x a código objeto.

1.3. Autómatas Finitos Deterministas

Un autómata determinista es un autómata que no puede estar en más de un estado a la vez. Es decir, para cada entrada hay un y solamente un estado al cual se puede hacer una transición desde el estado actual.

Normalmente se abrevia AFD o DFA (por sus siglas en inglés.

Definición

Un autómata finito determinista $A = (Q, \Sigma, \delta, q_0, F)$ consiste de:

- Un conjunto finito de estados Q.

- Un conjunto de símbolos de entrada Σ.

- Una función de transición que toma como argumentos un estado y un símbolo de entrada y regresa un estado:
  $\delta(q, a) = p$ es el estado p tal que existe un arco con etiqueta a desde q hacia p
- Un estado inicial dentro de los estados Q, normalmente q_0
- Un conjunto de estados finales (o de aceptación) F, tal que $F \subseteq Q$

1.3.1. Procesamiento de cadenas por un AFD

¿Cómo decide un AFD si acepta o no una secuencia de símbolos? El lenguaje del AFD es el conjunto de todas las cadenas que acepta.

Suponga que $a_1a_2 \cdots a_n$ es una secuencia de símbolos, empezamos en el AFD por el estado q_0 y consultamos la función de transición δ. Esto nos lleva a $\delta(q_0, a_1)$ para determinar el estado al que se entra al procesar el símbolo a_1 y continuamos de esta manera hasta encontrar $\delta(q_x, a_n) = q_y$. Si q_y es un miembro de F entonces se acepta $a_1a_2 \cdots a_n$, en caso contrario se rechaza.
Ejemplo: diseñar un AFD que acepte todas las cadenas de 0s y 1s que contienen la secuencia 01 en algún lugar, esto es el lenguaje:

$$L = \{w \mid w \text{ es de la forma } x01y \text{ para cualesquiera cadenas } x \text{ e } y \text{ consistentes de 0s y 1s}\}$$

de aquí podemos observar que las cadenas 01, 001, 011, 010, 11010 pertenecen al lenguaje, mientras que las cadenas ε, 0, 11, 0000, 11100 no pertenecen.

- $\Sigma = \{0, 1\}$
- q_0 es nuestro estado inicial
- ¿Cómo definir los demás estados?

  1. Estado final q_1: ¿ya se procesó 01? entonces acepta todo lo que llegue. $\delta(q_1, 0) = q_1,\ \delta(q_1, 1) = q_1$
  2. Estado q_2: no se ha procesado 01 pero la entrada más reciente es 0, si se recibe un 1 se habrá procesado 01. $\delta(q_2, 0) = q_2,\ \delta(q_2, 1) = q_1$
  3. Estado inicial q_0: No se ha procesado 01 y su entrada más reciente es 1 o nada, por lo tanto, no puede aceptar la cadena a menos que vea primero un 0 y luego un 1. $\delta(q_0, 0) = q_2,\ \delta(q_0, 1) = q_0$

1.3.2. Diagrama de transición de un AFD

Sea $A = (Q, \Sigma, \delta, q_0, F)$:

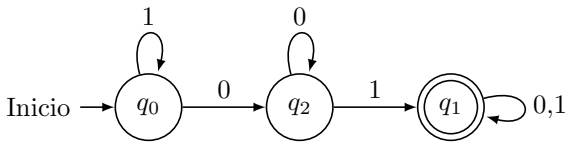
a) Para cada estado en Q hay un nodo.

b) Para cada estado q en Q y cada símbolo a en Σ, asumamos que $\delta(q, a) = p$, entonces el diagrama tiene un arco del nodo q al nodo p con etiqueta a. Es posible agrupar múltiples símbolos en un mismo arco.

c) Hay una flecha hacia q_0 con etiqueta Inicio.

d) Los nodos de los estados de aceptación (en F) se marcan con doble círculo.

Por ejemplo, el diagrama del autómata propuesto con anterioridad sería:



1.3.3. Tabla de transición de un AFD

Es posible representar un AFD por medio de una tabla, siguiendo estos criterios:

a) Los renglones corresponden a los estados.

b) Las columnas corresponden a los símbolos de entrada.

c) El valor en la tabla para el renglón q con entrada a es el estado dado por $\delta(q, a)$

d) El estado inicial se denota con una flecha.

e) Los estados finales se señalan con un asterisco.

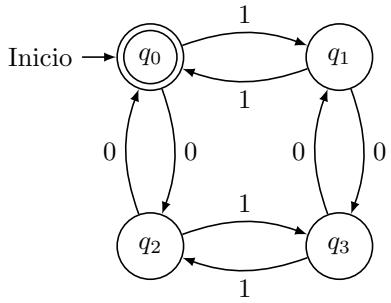
La tabla de transición del autómata anterior es:

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

1.3.4. Función de Transición Extendida

De manera similar a la función de transición $\delta(q, a) = p$, que determina que el autómata hará una transición desde q hacia p cuando procese el símbolo a, la función de transición extendida $\hat{\delta}(q, w) = p$ nos indica que el autómata se encontrará en el estado p después de procesar la cadena w a partir del estado q.

Considere el siguiente autómata:



	0	1
$* \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

cuyo lenguaje es $L = \{w \mid w$ tiene un número par de 0s y 1s$\}$. Ahora suponga que la cadena de entrada es 110101, puesto que consta de un número par de 0s y 1s se espera que la cadena esté en el lenguaje. Expresado con la función de transición extendida, procesar la cadena desde el estado inicial q_0, debería llevar al autómata al estado final q_0, esto es:

$$\hat{\delta}(q_0, 110101) = q_0$$

Para verificarlo, hay que computar $\hat{\delta}(q_0, w)$ para cada prefijo de 110101, empezando por ε e incrementando símbolo a símbolo hasta completar la cadena:

$$\begin{aligned}
\hat{\delta}(q_0, \varepsilon) &= q_0 \\
\hat{\delta}(q_0, 1) &= \delta(\hat{\delta}(q_0, \varepsilon), 1) && = \delta(q_0, 1) = q_1 \\
\hat{\delta}(q_0, 11) &= \delta(\hat{\delta}(q_0, 1), 1) && = \delta(q_1, 1) = q_0 \\
\hat{\delta}(q_0, 110) &= \delta(\hat{\delta}(q_0, 11), 0) && = \delta(q_0, 0) = q_2 \\
\hat{\delta}(q_0, 1101) &= \delta(\hat{\delta}(q_0, 110), 1) && = \delta(q_2, 1) = q_3 \\
\hat{\delta}(q_0, 11010) &= \delta(\hat{\delta}(q_0, 1101), 0) && = \delta(q_3, 0) = q_1 \\
\hat{\delta}(q_0, 110101) &= \delta(\hat{\delta}(q_0, 11010), 1) && = \delta(q_1, 1) = q_0
\end{aligned}$$

El Lenguaje de un Autómata Finito Determinista

El lenguaje de un AFD $A = \{Q, \Sigma, \delta, q_0, F\}$ es denotado $L(A)$ y está definido por:

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \in F\}$$

Si $L = L(A)$ para algún AFD A, entonces L es un lenguaje regular.

1.3.5. Ejercicios

Proponga AFDs que acepten los siguientes lenguajes:

1. a^*b
2. $(ab)^*$

3. $(ab)^+$
4. $(abc)^*$
5. $(abc)^+$
6. $(abcd)^*$
7. $(abcd)^+$
8. Todas las cadenas que terminen en ba para $\Sigma = \{a, b, c\}$

Utilizando el autómata descrito en la sección 1.3.4, calcule $\hat{\delta}(q_0, w)$ para las cadenas w:

9. 0000
10. 0100
11. 0101011101

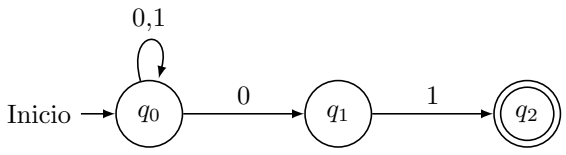
1.4. Autómatas Finitos No Deterministas

Un autómata finito no determinista (AFN) tiene la característica de estar en varios estados a la vez, comúnmente expresada como una habilidad de "adivinar" algo sobre la entrada.

En comparación con los autómatas finitos deterministas, los AFNs tienen las siguientes características:

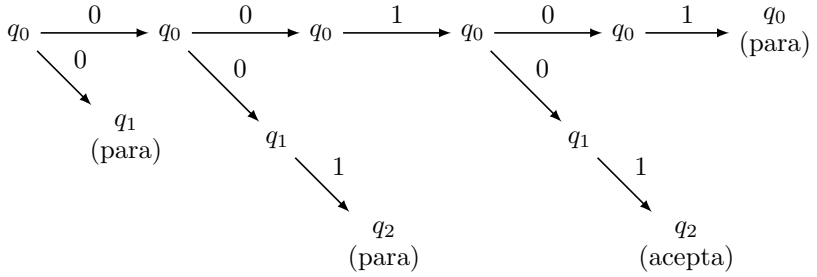
- Un AFN acepta exactamente lo mismo que un AFD, es decir, lenguajes regulares.
- Los AFN son más sencillos de diseñar y siempre se pueden convertir a un AFD.
- La diferencia entre un AFN y un AFD está en la definición de δ: en un AFN, δ es una función que recibe un estado y un símbolo, pero regresa un *conjunto* (posiblemente vacío) de estados. En comparación, δ en un AFD regresa siempre un único estado.

Ejemplo: AFN que acepta todas las cadenas que terminan en 01 para $\Sigma = \{0, 1\}$:



note como al procesar el símbolo 0 en el estado q_0 es posible mantenerse en q_0 o efectuar una transición al estado q_1. Además, no es necesario que cada estado especifique una transición para cada símbolo del alfabeto. El autómata se detendrá al no encontrar una transición válida o al haber procesado la cadena en su totalidad.

En la siguiente figura se muestran todas las posibles formas en que el autómata procesaría la cadena 00101:



como puede observarse, sólo uno de estos caminos lleva a la aceptación de la cadena. Esto nos indica que la cadena es aceptada por el autómata, independientemente del número de posibles caminos que llevarían a que el AFN se detenga.

1.4.1. Definición de un AFN

Un AFN N se define como $N = \{Q, \Sigma, \delta, q_0, F\}$ donde:

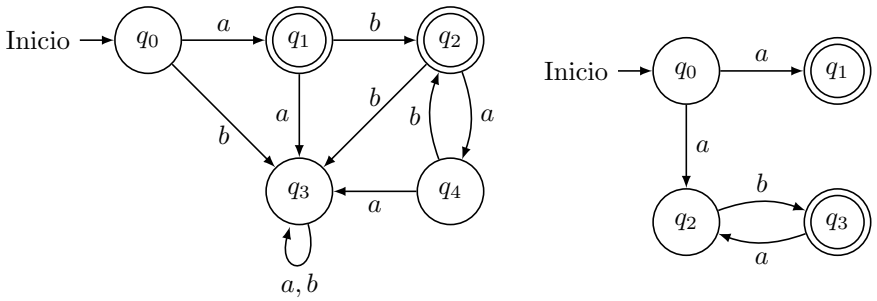
- Q es el conjunto finito de estados.
- Σ es el conjunto finito de símbolos de entrada.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.
- δ es la función de transición que recibe un estado $q \in Q$ y un símbolo $a \in \Sigma$, y regresa un *subconjunto* de Q.

El AFN descrito con anterioridad se especifica formalmente como:

$$(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\}),$$

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

Ejemplo: comparación entre un autómata finito determinista y su contraparte no determinista, ambos aceptan el lenguaje $a \cup (ab)^+$



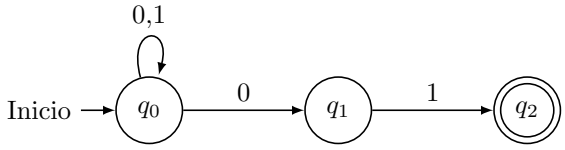
AFD AFN

1.4.2. Función de Transición Extendida

La función de transición δ se extiende a una función $\hat{\delta}(q, w)$ que recibe un estado q y una cadena de símbolos w, y regresa el conjunto de estados en los que está el AFN si comienza en q y procesa w.
Base: $\hat{\delta}(q, \varepsilon) = \{q\}$
Inducción: suponga que $w = xa$ donde a es el símbolo final de w y x el resto. Además suponga que $\hat{\delta}(q, x) = \{p_1, p_2, p_3, \cdots, p_k\}$ y

$$\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, r_3, \cdots, r_m\}$$

entonces $\hat{\delta}(q, w) = \{r_1, r_2, r_3, \cdots, r_m\}$.
Ejemplo: Usar $\hat{\delta}$ para procesar 00101 por el AFN



$$\begin{aligned}
\hat{\delta}(q_0, \varepsilon) &= \{q_0\} \\
\hat{\delta}(q_0, 0) &= \delta(q_0, 0) &&= \{q_0, q_1\} \\
\hat{\delta}(q_0, 00) &= \delta(q_0, 0) \cup \delta(q_1, 0) &&= \{q_0, q_1\} \cup \phi = \{q_0, q_1\} \\
\hat{\delta}(q_0, 001) &= \delta(q_0, 1) \cup \delta(q_1, 1) &&= \{q_0\} \cup \{q_2\} = \{q_0, q_2\} \\
\hat{\delta}(q_0, 0010) &= \delta(q_0, 0) \cup \delta(q_2, 0) &&= \{q_0, q_1\} \cup \phi = \{q_0, q_1\} \\
\hat{\delta}(q_0, 00101) &= \delta(q_0, 1) \cup \delta(q_1, 1) &&= \{q_0\} \cup \{q_2\} = \{q_0, q_2\}
\end{aligned}$$

dado que $\hat{\delta}(q_0, 00101) = \{q_0, q_2\}$ y $\{q_0, q_2\} \cap F \neq \phi$, entonces la cadena 00101 es aceptada por el AFN.

1.4.3. El Lenguaje de un AFN

El hecho de que algunas "opciones" al procesar w no lleven a un estado final o no lleguen a algún estado (ϕ) no indica que w no debería ser aceptada por el AFN.

Formalmente, si $N = (Q, \Sigma, \delta, q_0, F)$ es un AFN, entonces:

$$L(N) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \phi\}$$

1.4.4. Equivalencia de AFD y AFN

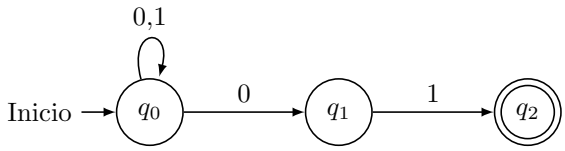
Tanto los autómatas finitos deterministas (AFD) como los no deterministas (AFN) son capaces de reconocer lenguajes regulares. Partiendo de esta premisa, podemos considerar que para cada AFN existe un AFD que reconoce el mismo lenguaje.

La construcción comienza por un AFN $N = \{Q_N, \Sigma, \delta_N, q_0, F_N\}$ y la meta es la descripción de un AFD $D = \{Q_D, \Sigma, \delta_D, \{q_0\}, F_D\}$ tal que $L(D) = L(N)$:

- Q_D es el conjunto de subconjuntos de Q_N, si Q_N tiene n estados, entonces Q_D tiene 2^n estados. Comúnmente, no todos son necesarios.
- F_D es el conjunto de subconjuntos S de Q_N tal que $S \cap F_N \neq \phi$: es decir, F_D al igual que Q_D, está formado por subconjuntos de Q_N con la diferencia que cada uno de estos subconjuntos incluye al menos un estado de aceptación en F_N.
- Para cada conjunto $S \subseteq Q_N$ y cada símbolo a en Σ, la función δ_D se construye:

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$$

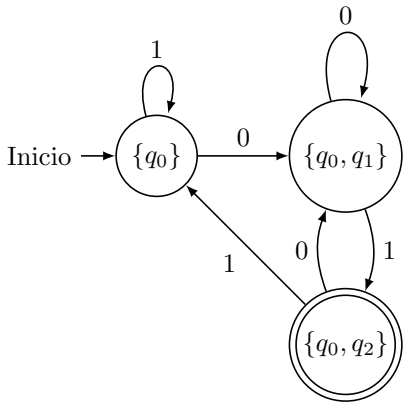
Ejemplo: Construir el AFD equivalente al AFN



La función δ_D del AFD equivalente estaría entonces definida por:

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

y el AFD resultante después de eliminar los estados que no se utilizan es:



1.4.5. Ejercicios

Diseñe AFN que reconozcan los siguientes lenguajes:

1. $(ab)^+$
2. $(ab)^*$
3. $(abc)^+$
4. $(abc)^*$
5. Terminan en ba para $\Sigma = \{a, b, c\}$
6. $a^+ \cup b^+ \cup (ab)^+$

Convertir a AFD los siguientes AFN:

7.

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

8.

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

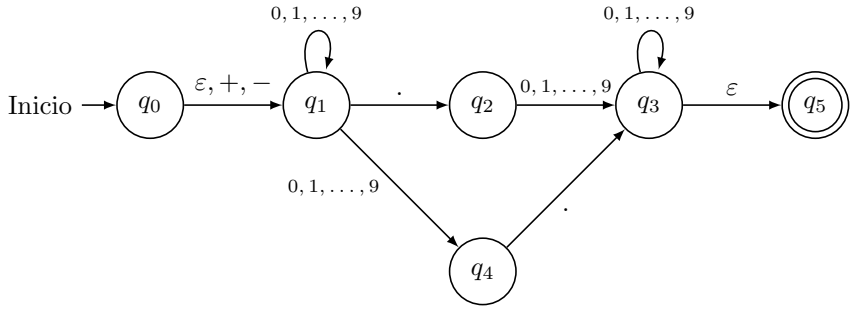
9.

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

1.5. Autómatas Finitos con Transiciones Epsilon

Existen lenguajes formados por cadenas que contienen símbolos o subcadenas opcionales. Diseñar un AFD o un AFN para dichos lenguajes puede llegar a ser complicado. Considere el formato para números de punto flotante (excluyendo el exponente):

1. Un signo + o - opcional,
2. una cadena de digitos,
3. un punto decimal, y
4. otra cadena de digitos.
5. Además, la cadena 2 o la cadena 4 pueden ser vacías, pero al menos una debe ser no vacía.



que reconoce cadenas como:

+3.05	3.05
-3.05	.05
+3.	+.05
-3.	-.05

1.5.1. Definición de un AFN con transiciones epsilon

Se representa de igual manera que un AFN, con la excepción de que la función de transición debe incluir información sobre las transiciones ε. Ahora la función δ toma como argumentos:

1. Un estado en Q, y
2. un miembro de $\Sigma \cup \{\varepsilon\}$

El autómata descrito en la figura anterior se define como:

$$E = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{., +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \delta, q_0, \{q_5\})$$

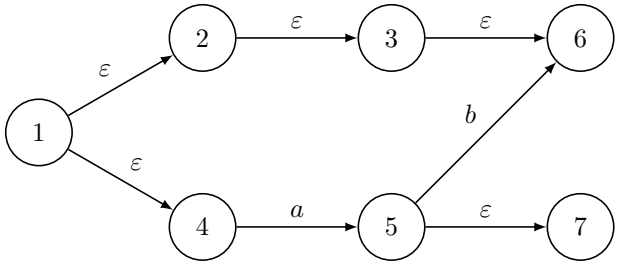
donde δ es:

	ε	$+,-$	$0,1,\ldots,9$	.
$\rightarrow q_0$	$\{q_1\}$	$\{q_1\}$	ϕ	ϕ
q_1	ϕ	ϕ	$\{q_1, q_4\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_3\}$	ϕ
q_3	$\{q_5\}$	ϕ	$\{q_3\}$	ϕ
q_4	ϕ	ϕ	ϕ	$\{q_3\}$
$*q_5$	ϕ	ϕ	ϕ	ϕ

1.5.2. Cerradura epsilon

Base: $q \in ECLOSE(q)$
Inducción: Si $p \in ECLOSE(q)$ y hay una transición de p a r con etiqueta ε, entonces $r \in ECLOSE(q)$. En otras palabras, si $p \in ECLOSE(q)$, entonces $ECLOSE(q)$ contiene todos los estados de $\delta(p, \varepsilon)$.

Ejemplo, considere el siguiente conjunto de estados y transiciones:



entonces, $ECLOSE(5) = \{5, 7\}$ pues 5 es parte de su propia cerradura epsilon y existe una transición epsilon desde 5 a 7. De igual forma, $ECLOSE(1) = \{1, 2, 3, 4, 6\}$, donde se incluye al propio estado y a todos aquellos estados a los que es posible alcanzar haciendo únicamente transiciones ε.

1.5.3. Función de transición extendida

Base: $\hat{\delta}(q, \varepsilon) = ECLOSE(q)$
Inducción: suponga que w es de la forma xa, donde a es el último símbolo de w y $a \neq \varepsilon$, calculamos $\hat{\delta}(q, w)$ de la siguiente forma:

- Sea $\hat{\delta}(q, x) = \{p_1, p_2, \ldots, p_k\}$, y
- $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \ldots, r_m\}$,
- entonces, $\hat{\delta}(q, w) = \bigcup_{j=1}^{m} ECLOSE(r_j)$

Ejemplo, calcular $\hat{\delta}(q_0, 5.6)$ para el AFN-ε descrito en la sección 1.5:

1. $\hat{\delta}(q_0, \varepsilon) = ECLOSE(q_0) = \{q_0, q_1\}$
2. $\hat{\delta}(q_0, 5) = ECLOSE(q_1) \cup ECLOSE(q_4) = \{q_1, q_4\}$
   $\longrightarrow \delta(q_0, 5) \cup \delta(q_1, 5) = \{q_1, q_4\}$
3. $\hat{\delta}(q_0, 5.) = ECLOSE(q_2) \cup ECLOSE(q_3) = \{q_2, q_3, q_5\}$
   $\longrightarrow \delta(q_1, .) \cup \delta(q_4, .) = \{q_2, q_3\}$
4. $\hat{\delta}(q_0, 5.6) = ECLOSE(q_3) = \{q_3, q_5\}$
   $\longrightarrow \delta(q_2, 6) \cup \delta(q_3, 6) \cup \delta(q_5, 6) = \{q_3\}$

1.5.4. Equivalencia de AFD y AFN con transiciones epsilon

Dado cualquier AFN-ε E podemos encontrar un AFD D que acepta el mismo lenguaje que E. El proceso es muy similar a la conversión de AFN a AFD, pero se deben incluir las transiciones ε.

Sea $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$, entonces el AFD equivalente

$$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$$

es definido como:

1. Q_D es el conjunto de subconjuntos de Q_E.
2. $q_D = ECLOSE(q_0)$
3. F_D es el conjunto de estados de Q_D que contienen al menos un estado de aceptación de E:

$$F_D = \{S \mid S \in Q_D, S \cap F_E \neq \phi\}$$

4. $\delta_D(S, a)$ se calcula para todo $a \in \Sigma$ y todo $S \in Q_D$:

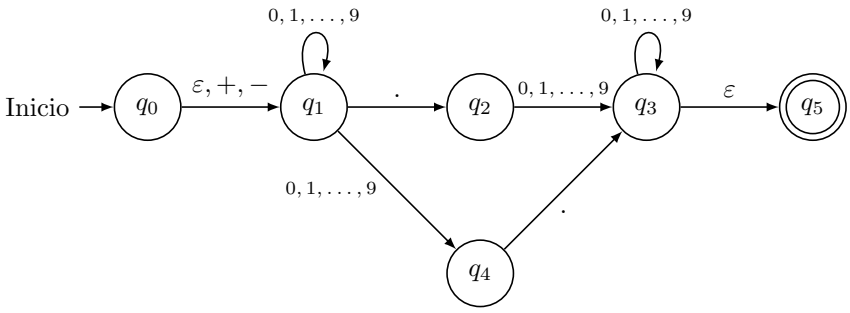
   a) Sea $S = \{p_1, p_2, p_3, \ldots, p_k\}$

   b) Calcular $\bigcup_{i=1}^{k} \delta_E(p_i, a) = \{r_1, r_2, \ldots, r_m\}$

   c) $\delta_D(S, a) = \bigcup_{j=1}^{m} ECLOSE(r_j)$

   es decir, para cualquier estado S de D es necesario procesar la unión de las funciones de transición del AFN-ε para cada uno de los estados (de E) que forman S. Y posteriormente con cada uno de los estados resultantes de dicha unión, se calcula la unión de cerraduras epsilon.

Ejemplo: convertir el siguiente AFN-ε a AFD



El estado inicial de E es q_0, por lo que el estado inicial de D es $ECLOSE(q_0) = \{q_0, q_1\}$.

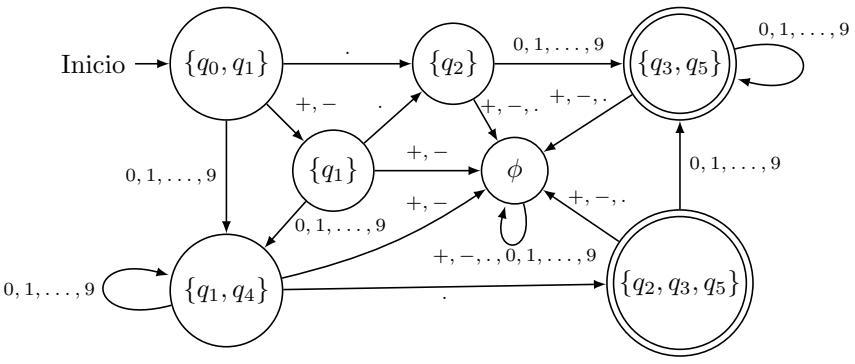
Primero comencemos por obtener la cerradura epsilon de cada estado de E:

$$
\begin{aligned}
ECLOSE(q_0) &= \{q_0, q_1\} \\
ECLOSE(q_1) &= \{q_1\} \\
ECLOSE(q_2) &= \{q_2\} \\
ECLOSE(q_3) &= \{q_3, q_5\} \\
ECLOSE(q_4) &= \{q_4\} \\
ECLOSE(q_5) &= \{q_5\}
\end{aligned}
$$

y ahora procedemos a construir la tabla para las transiciones de D:

δ_D	$+, -$	$.$	$0, 1, \ldots, 9$
$\rightarrow \{q_0, q_1\}$	$\{q_1\}$	$\{q_2\}$	$\{q_1, q_4\}$
$\{q_1\}$	ϕ	$\{q_2\}$	$\{q_1, q_4\}$
$\{q_2\}$	ϕ	ϕ	$\{q_3, q_5\}$
$\{q_1, q_4\}$	ϕ	$\{q_2, q_3, q_5\}$	$\{q_1, q_4\}$
$*\{q_3, q_5\}$	ϕ	ϕ	$\{q_3, q_5\}$
$*\{q_2, q_3, q_5\}$	ϕ	ϕ	$\{q_3, q_5\}$
ϕ	ϕ	ϕ	ϕ

y el autómata resultante es:



1.5.5. Ejercicios

1. Construir los siguientes AFN-ε:

   a) $a^*b^*c^*$, es decir, cero o más a seguida de cero o más b, seguidas de cero o más c

   b) 01 repetido una o más veces, o 010 repetido una o más veces.

2. Calcular las cerraduras epsilon para todos los estados de:

	ε	a	b	c
$\rightarrow p$	ϕ	$\{p\}$	$\{q\}$	$\{r\}$
q	$\{p\}$	$\{q\}$	$\{r\}$	ϕ
$*r$	$\{q\}$	$\{r\}$	ϕ	$\{p\}$

3. Calcule $\hat{\delta}(q_0, +3.1416)$ para el AFN-ε de la sección 1.5.

4. Convierta de AFN-ε a AFD:

   a)

	ε	a	b	c
$\rightarrow p$	ϕ	$\{p\}$	$\{q\}$	$\{r\}$
q	$\{p\}$	$\{q\}$	$\{r\}$	ϕ
$*r$	$\{q\}$	$\{r\}$	ϕ	$\{p\}$

   b)

	ε	a	b	c
$\rightarrow p$	$\{q, r\}$	ϕ	$\{q\}$	$\{r\}$
q	ϕ	$\{p\}$	$\{r\}$	$\{p, q\}$
$*r$	ϕ	ϕ	ϕ	ϕ

Capítulo 2

Expresiones Regulares

Las expresiones regulares nos ofrecen algo que los autómatas no: una forma declarativa para expresar las cadenas que se pretende aceptar.

Las expresiones regulares definen exactamente el mismo tipo de lenguaje que aceptan los autómatas, es decir, lenguajes regulares.

2.1. Operaciones de las expresiones regulares

La *unión* de dos lenguajes, $L \cup M$, es el conjunto de cadenas que pertenecen a L, a M o a ambos:

$$L \cup M = \{w \mid w \in L \vee w \in M\}$$

por ejemplo, si $L = \{001, 10, 111\}$ y $M = \{\varepsilon, 001\}$ entonces $L \cup M = \{\varepsilon, 10, 001, 111\}$.

La *concatenación* de lenguajes L y M es el conjunto de cadenas que pueden ser formadas tomando cada cadena en L y concatenándole cada cadena en M:

$$LM = \{lm \mid l \in L, m \in M\}$$

que utilizando el ejemplo anterior resultaría en $LM = \{001, 001001, 10, 10001, 111, 111001\}$

La *cerradura de Kleene* de un lenguaje L, denotada L^*, es el conjunto de cadenas que se forma concatenando múltiples veces las cadenas de L. Formalmente:

$$L^* = \bigcup_{i \geq 0} L^i, \text{ donde } L^0 = \{\varepsilon\}, L^1 = L \text{ y } L^i = \underbrace{LL \cdots L}_{i} \text{ para } i > 1$$

Por ejemplo, si $L = \{0, 11\}$ entonces L^* consiste de todas las cadenas de 0 y 1 tales que los 1 vienen en pares consecutivos. En este caso, las cadenas ε, 0, 00, 11, 0011, 011, 011011, 11, 110 pertenecen a L^*, pero no es así para 01011 y 101.

2.2. Construcción de expresiones regulares

Para cada expresión regular E se describe el lenguaje $L(E)$ que representa:
Base:

- Las constantes ε y $\emptyset$ son expresiones regulares que denotan los lenguajes $\{\varepsilon\}$ y $\emptyset$, respectivamente. Formalmente, $L(\varepsilon) = \{\varepsilon\}$ y $L(\emptyset) = \emptyset$
- Si a es un símbolo, entonces a es una expresión regular que denota el lenguaje $L(a) = \{a\}$
- Una variable E representa cualquier lenguaje.

Inducción:

- Si E y F son expresiones regulares, entonces $E + F$ es una expresión regular denotando $L(E + F) = L(E) \cup L(F)$
- Si E y F son expresiones regulares, entonces EF es una expresión regular que denota $L(EF) = L(E)L(F)$
- Si E es una expresión regular, entonces E^* es una expresión regular que denota $L(E^*) = (L(E))^*$
- Si E es una expresión regular, entonces (E) es una expresión regular que denota el mismo lenguaje que E, es decir, $L(E) = L((E))$

Ejemplo: escribir una expresión regular que denote el conjunto de cadenas de 0 y 1 intercalados.

- Primero veamos algunos ejemplos de cadenas: 0, 1, 01, 10, 0101, 1010 forman parte del lenguaje mientras que 00, 11, 011 no lo son.
- El conjunto de cadenas $\varepsilon, 01, 0101, 010101, \ldots$ puede ser denotado con la expresión $(01)^*$. Cabe resaltar que $(01)^*$ no es lo mismo que 01^* pues el operador * tiene precedencia sobre la concatenación.
- De igual manera las cadenas $\varepsilon, 10, 1010, 101010, \ldots$ son expresadas por medio de $(10)^*$
- Así, el lenguaje que combine ambos patrones será descrito por la expresión $(01)^* + (10)^*$
- Sin embargo, dicho lenguaje no reconoce las cadenas de longitud impar, como 01010 o 101. Mismas que serían reconocidas por la expresión regular $0(10)^* + 1(01)^*$
- De este modo, la expresión que denota este lenguaje es: $(01)^* + (10)^* + 0(10)^* + 1(01)^*$

2.2.1. Precedencia de operador

El orden en que se aplican las operaciones sigue la siguiente precedencia:

- $(,)$
- *,$^+$, potencia
- concatenación
- $+$

las operaciones con la misma precedencia se aplican de derecha a izquierda. En base a esto, podemos determinar que $ab^+ = (a)(b^+) \neq (ab)^+$ y que $ab^* + a = ((a)(b^*)) + (a)$.

2.2.2. Ejercicios

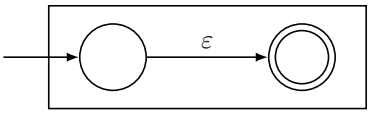
Escriba expresiones regulares para los siguientes regulares

1. El conjunto de cadenas sobre $\Sigma = \{a, b, c\}$ que contienen al menos una a y al menos una b.
2. El conjunto de cadenas de 0 y 1 cuyo décimo símbolo es 1.
3. El conjunto de cadenas de 0 y 1 de longitud a lo más de 5.
4. El lenguaje formado por 0 y 1 de las cadenas que no finalizan en 01.
5. El conjunto de cadenas de 0 y 1 que terminan en 1 y no contienen la subcadena 00.
6. El conjunto de cadenas de 0 y 1 cuya longitud es múltiplo de 5.
7. El conjunto de cadenas de 0 y 1 con sólo un par de 1 consecutivos.
8. El conjunto de cadenas de 0 y 1 con a lo más un par de 1 consecutivos.
9. El conjunto de cadenas de 0 y 1 tales que cada par de 0 aparece antes que cada par de 1.
10. El conjunto de cadenas de 0 y 1 con número de 0 divisible por 3

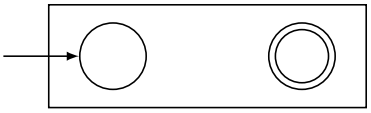
2.3. Conversión de expresiones regulares a autómatas

Teorema: todo lenguaje definido por una expresión regular es también definido por un autómata finito.
Base:

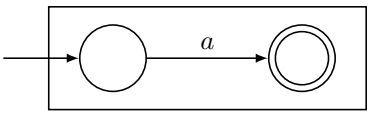
a) La expresión regular ε es:



b) Maneja la expresión $\emptyset$

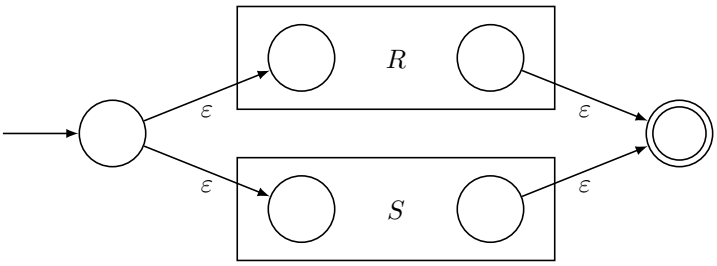


c) Maneja la expresión a:

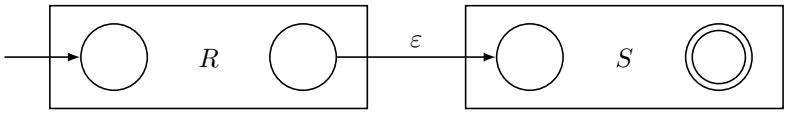


Inducción:

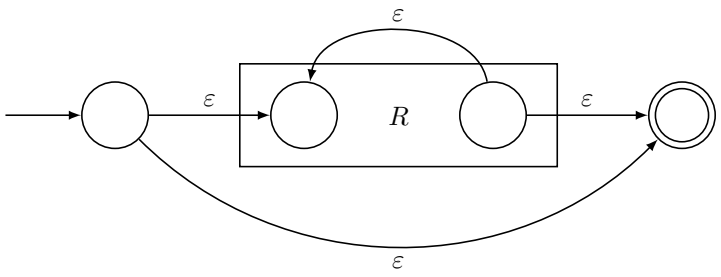
a) La expresión regular $R + S$ para expresiones R y S más pequeñas:



b) La expresión regular RS para expresiones R y S más pequeñas:



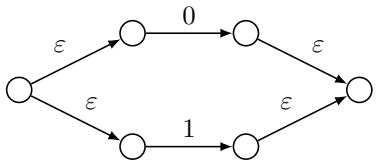
c) La expresión R^* para alguna expresión más pequeña R:



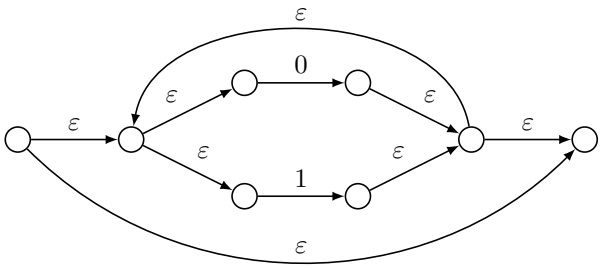
d) Si la expresión es (R) para algúna R, el autómata para R es el autómata para (R).

Ejemplo: convertir $(0+1)^*1(0+1)$ a AFN-ε.

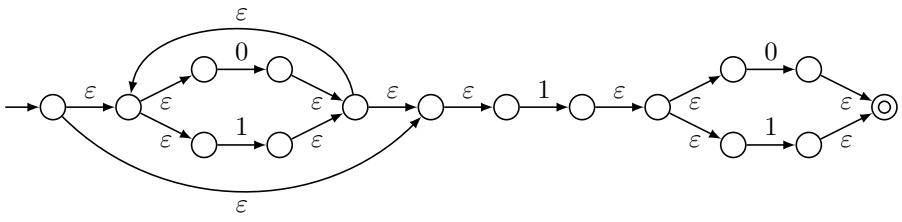
Primeramente, construimos el sub-autómata que reconoce $0+1$:



posteriormente, definimos el sub-autómata que reconoce $(0+1)^*$:



y, finalmente, procedemos a realizar la concatenación de los sub-autómatas que representan las sub-expresiones y así obtener el AFN-ε de $(0+1)^*1(0+1)$:



2.3.1. Ejercicios

Construya el autómata que reconoce el lenguaje descrito por las expresiones regulares:

1. 01^*
2. $(0+1)01$
3. $00(0+1)^*$
4. $(0+1)^3$
5. $(\varepsilon+0+1)^3$
6. $(0+1)^+$
7. $(0+1)^*(11+0)+1+\varepsilon$
8. $(01+1)^+$
9. $((0+1)^3)^*$
10. $((0+1)^*)^3$

2.4. Leyes algebraicas para expresiones regulares

Las expresiones regulares cumplen con las siguientes leyes:

- **Conmutatividad:**
  $L+M=M+L$
  La concatenación no es conmutativa, ej. $01 \neq 10$
- **Asociatividad:**
  $L+M+N=(L+M)+N=L+(M+N)$
  $LMN=(LM)N=L(MN)$
- **Identidad:**
  $\emptyset+L=L+\emptyset=L$
  $\varepsilon L=L\varepsilon=L$
- **Absorción:**
  $\emptyset L=L\emptyset=\emptyset$
- **Distributivas:**
  $L(M+N)=LM+LN$
  $(M+N)L=ML+NL$
- **Idempotencia:**
  $L+L=L$

- **Cerraduras:**
  $(L^*)^* = L^* = (L + \varepsilon)^*$
  $\emptyset^* = \varepsilon$
  $L^+ = LL^* = L^*L$
  $L^* = L^+ + \varepsilon = L^+ + L^*$
  $(L + M)^* = (L^* + M^*)^* = (L^*M^*)^*$
  $(L^*M)^* = \varepsilon + (L + M)^*M$
  $(LM^*)^* = \varepsilon + L(L + M)^*$
  $L^+ + L^* = L^*$
  $L + L^* = L^*$
  $L + (L + M)^* = (L + M)^*$
  $L^* + (L + M)^* = (L + M)^*$
  $L^n + L^+ = L^+$ para $n \geq 1$
  $L^n + L^* = L^*$

2.4.1. Ejercicios

Aplicando las propiedades de las expresiones regulares, simplifique las siguientes expresiones:

1. $s(\varepsilon + r)^*(\varepsilon + r) + s$
2. $(a^*b)^* + (ab^*)^*$
3. $(\varepsilon + ab)^*$
4. $a(\varepsilon + aa)^*a + \varepsilon$
5. $(a + \varepsilon)a^*b$
6. $(((a^*a)b) + b)$
7. $\emptyset^* + a^* + b^* + (a^* + b^*)^+$
8. $(\varepsilon + a^+)bb^+(\varepsilon + c)^*$
9. $y(\varepsilon + x^+) + (yy^+(\varepsilon + x)^*)$
10. $(\varepsilon + x)(\varepsilon + x)^+ + (\varepsilon + x) + \emptyset^*$
11. $(a + b)(\varepsilon + aa)^*(\varepsilon + aa) + (a + b)$

Estudie como se utilizan las expresiones regulares en:

12. Lenguaje C: scanf y regex.h
13. UNIX: comandos grep y find

Capítulo 3

Lenguajes Independientes del Contexto

Considere el lenguaje de las palíndromas sobre $\{0, 1\}$, que contiene cadenas como $00, 0110, 001100, 10100101$. Para este lenguaje simplemente no existe una expresión regular o un autómata finito que lo describa o reconozca.

De aquí que este lenguaje no puede ser catalogado como un lenguaje regular. Sin embargo, determinar si una cadena pertenece o no a este lenguaje es una tarea que fácilmente puede realizarse con una máquina distinta.

3.1. Gramáticas independientes del contexto

Definición:

- Hay un conjunto finito de símbolos que forman las cadenas del lenguaje. Este alfabeto se conoce como terminales o símbolos terminales.
- Hay un conjunto finito de variables, llamadas no-terminales o categorías sintácticas. Cada variable representa un lenguaje.
- Una de las variables representa el lenguaje que se está definiendo, llamado el símbolo inicial.
- Hay un conjunto finito de producciones o reglas que representan la definición recursiva del lenguaje. Cada producción consiste de:
  - Una variable que está siendo definida por la producción (cabeza de la producción).
  - El símbolo de producción $\rightarrow$
  - Una cadena de cero o más terminales y variables (cuerpo de la producción), que representa una manera de formar cadenas en el lenguaje de la variable de la cabeza.

Ejemplo 1: el lenguaje de las palíndromas sobre $\{0, 1\}$ está definido como:

$$
\begin{aligned}
&P \to \varepsilon \\
&P \to 0 \\
&P \to 1 \\
&P \to 0P0 \\
&P \to 1P1
\end{aligned}
$$

Ejemplo 2: describir una GIC que represente expresiones con operaciones de suma y multiplicación. Los argumentos son identificadores que comienzan con a o b seguidos de cualquier cadena formada con los símbolos $\{a, b, 0, 1\}$, es decir, las cadenas del lenguaje $(a + b)(a + b + 0 + 1)^*$.

$$
\begin{aligned}
&E \to I \\
&E \to E + E \\
&E \to E * E \\
&E \to (E) \\
&I \to a \\
&I \to b \\
&I \to Ia \\
&I \to Ib \\
&I \to I0 \\
&I \to I1
\end{aligned}
$$

que puede ser expresado de forma corta como:

$$
\begin{aligned}
E &\to I \mid E + E \mid E * E \mid (E) \\
I &\to a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

podemos verificar que las siguientes cadenas pueden ser producidas por la gramática propuesta: $a, b, b0, b00, a0, ab01, a + b00, (a + b00), a * (a + b00)$

Ejemplo 3: describir una GIC cuyo lenguaje sea el de los prototipos en lenguaje C (restringido a nombres o identificadores en $(a + b)(a + b + 0 + 1)^*$).

$$
\begin{aligned}
&P \to T\ I(A); \\
&I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
&A \to T\ I \\
&A \to A, A \\
&T \to int \mid double \mid float \mid char
\end{aligned}
$$

Ejemplo 4: definir una gramática independiente del contexto para la construcción de una sentencia *while* con condiciones $>, <, >=, <=$

$$
\begin{aligned}
&W \to while(C)\{X\}\\
&C \to E > E \mid E < E \mid E >= E \mid E <= E\\
&X \to E;\\
&X \to XX
\end{aligned}
$$

3.1.1. Ejercicios

1. $\{0^n 1^n \mid n \geq 1\}$
2. $\{0^n 1^m \mid m = n \vee m = 2n\}$
3. $\{0^m 1^n \mid m > n, n \geq 0\}$
4. $\{a^n b^m c^{n+m} \mid m \geq 0, n \geq 0\}$
5. $\{a^n b^{n+m} c^m \mid m \geq 0, n \geq 0\}$
6. $\{a^n b^n c^m d^m \mid m \geq 0, n \geq 0\}$
7. $\{a^n b^m \mid 0 \leq n \leq m \leq 2n\}$
8. $\{a^m b^n c^p \mid m = n \vee n = p\}$

3.2. Derivación

Sea $\alpha A \beta$ una cadena de terminales y variables, siendo A una variable. Esto es, α y β son cadenas en $(V \cup T)^*$ y $A \in V$.

Sea $A \to \gamma$ una producción de G, entonces decimos que $\alpha A \beta \underset{G}{\Longrightarrow} \alpha\gamma\beta$, si se asume G conocido entonces sólo se expresa $\alpha A \beta \Longrightarrow \alpha\gamma\beta$. Un paso de derivación reemplaza cualquier variable en cualquier lugar de la cadena por el cuerpo de una de sus producciones. Para indicar cero o más pasos de derivación se usa $\overset{*}{\Longrightarrow}$.

Ejemplo: la inferencia de que $a * (a + b00)$ está en el lenguaje de

$$
\begin{aligned}
E &\to I \mid E + E \mid E * E \mid (E)\\
I &\to a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

se refleja en una derivación de la cadena empezando por E:

$$
\begin{aligned}
E &\Longrightarrow E * E \Longrightarrow I * E \Longrightarrow a * E \Longrightarrow a * (E) \Longrightarrow a * (E + E)\\
&\Longrightarrow a * (I + E) \Longrightarrow a * (a + E) \Longrightarrow a * (a + I)\\
&\Longrightarrow a * (a + I0) \Longrightarrow a * (a + I00) \Longrightarrow a * (a + b00)
\end{aligned}
$$

Note como no hay un único "camino" de derivación.

3.2.1. Derivación por la izquierda y por la derecha

Para restringir el número de opciones que tenemos al derivar una cadena, es común requerir que en cada paso se reemplace la variable más a la izquierda por una de sus producciones. Esta derivación es llamada derivación por la izquierda y se indica por medio de $\underset{lm}{\Longrightarrow}$ y $\overset{*}{\underset{lm}{\Longrightarrow}}$.

De manera análoga, la derivación por la derecha reemplaza la variable más a la derecha por una de sus producciones y se indica por medio de $\underset{rm}{\Longrightarrow}$ y $\overset{*}{\underset{rm}{\Longrightarrow}}$.

Ejemplo: derivar por la izquierda y por la derecha de la gramática

$$E \to I \mid E + E \mid E * E \mid (E)$$
$$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

para $a * (b1 + a00)$:

- Por la izquierda:

$$\begin{aligned} E &\underset{lm}{\Longrightarrow} E * E \underset{lm}{\Longrightarrow} I * E \underset{lm}{\Longrightarrow} a * E \underset{lm}{\Longrightarrow} a * (E) \underset{lm}{\Longrightarrow} a * (E + E) \\ &\underset{lm}{\Longrightarrow} a * (I + E) \underset{lm}{\Longrightarrow} a * (I1 + E) \underset{lm}{\Longrightarrow} a * (b1 + E) \underset{lm}{\Longrightarrow} a * (b1 + I) \\ &\underset{lm}{\Longrightarrow} a * (b1 + I0) \underset{lm}{\Longrightarrow} a * (b1 + I00) \underset{lm}{\Longrightarrow} a * (b1 + a00) \end{aligned}$$

- Por la derecha:

$$\begin{aligned} E &\underset{rm}{\Longrightarrow} E * E \underset{rm}{\Longrightarrow} E * (E) \underset{rm}{\Longrightarrow} E * (E + E) \underset{rm}{\Longrightarrow} E * (E + I) \\ &\underset{rm}{\Longrightarrow} E * (E + I0) \underset{rm}{\Longrightarrow} E * (E + I00) \underset{rm}{\Longrightarrow} E * (E + a00) \\ &\underset{rm}{\Longrightarrow} E * (I + a00) \underset{rm}{\Longrightarrow} E * (I1 + a00) \underset{rm}{\Longrightarrow} E * (b1 + a00) \\ &\underset{rm}{\Longrightarrow} I * (b1 + a00) \underset{rm}{\Longrightarrow} a * (b1 + a00) \end{aligned}$$

3.2.2. Ejercicios

Dar derivaciones por la izquierda y la derecha usando la gramática

$$S \to A1B$$
$$A \to 0A \mid \varepsilon$$
$$B \to 0B \mid 1B \mid \varepsilon$$

para las cadenas:

1. 00101
2. 1001
3. 00011

3.3. Árboles de derivación

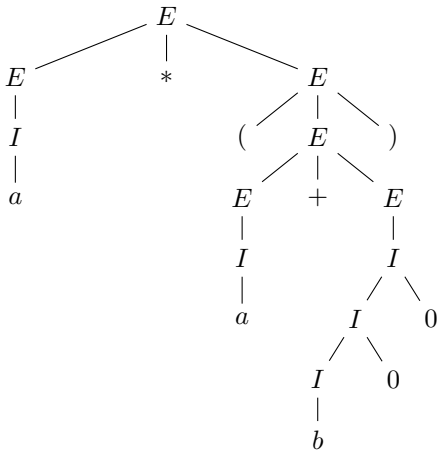
Sea $G = (V, T, P, S)$ una gramática independiente del contexto, el árbol de derivación es un árbol en el que:

1. Cada nodo interior tiene como etiqueta una variable en V
2. Cada hoja está etiquetada como una variable, un terminal o ε. Si la hoja es ε entonces debe ser el único hijo de su padre.
3. Si un nodo interior está etiquetado A y sus hijos son $X_1, X_2, \ldots, X_k$, respectivamente desde la izquierda, entonces $A \to X_1 X_2 \ldots X_k$ es una producción en P.

Ejemplo: Dada la gramática

$$E \to I \mid E + E \mid E * E \mid (E)$$
$$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

construya el árbol de derivación que muestre que $a * (a + b00)$ pertenece al lenguaje reconocido por E:



3.3.1. Ejercicios

Construya el árbol de derivación usando la gramática

$$S \to A1B$$
$$A \to 0A \mid \varepsilon$$
$$B \to 0B \mid 1B \mid \varepsilon$$

para las cadenas:

1. 00101

2. 1001

3. 00011

3.4. Ambigüedad

Considere la forma sentencial $E + E * E$, que tiene dos derivaciones desde E:

$$E \Longrightarrow E + E \Longrightarrow E + E * E$$
$$E \Longrightarrow E * E \Longrightarrow E + E * E$$

con sus respectivos árboles de derivación:

Si $G = (V, T, P, S)$ reconoce al menos una cadena w en T^* para la cual es posible encontrar dos árboles de derivación distintos, cada uno con raíz S, entonces se dice que G es ambigua. Del mismo modo, si cada cadena tiene a lo más un árbol de derivación, entonces la gramática no es ambigua.

3.4.1. Quitar la ambigüedad

No es posible ni siquiera dar un algoritmo que pueda decidir si una gramática independiente del contexto es ambigua. Más aún, hay lenguajes independientes del contexto que únicamente pueden ser producidos por gramáticas independientes del contexto ambiguas, para estos lenguajes, quitar la ambigüedad es imposible.

3.5. Autómatas de pila

Los lenguajes independientes del contexto tienen un tipo de autómata que los define, llamado autómata de pila. Estos autómatas son una "extensión" de los AFN-ε. En esencia, son autómatas finitos con una pila que les permite almacenar datos.

La presencia de una pila indica que el autómata puede "recordar" una cantidad infinita de información, pero sólo puede tener acceso a ella por medio de una pila que opera de forma LIFO (del inglés, *Last In, First Out*).

3.5.1. Definición

Un autómata de pila $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ tiene 7 componentes:

- Q: conjunto finito de estados del autómata
- Σ: conjunto finito de símbolos de entrada
- Γ: alfabeto de la pila
- δ: función de transición de la forma $\delta(q, a, X)$ donde,
  - q es un estado de Q
  - a es un símbolo en Σ o $a = \varepsilon$
  - X es un símbolo de la pila

  El valor de regreso de la función es el par (p, γ) donde:
  - p es el estado al que se lleva al autómata
  - si $\gamma = \varepsilon$ entonces se hace una operación pop a la pila.
  - si $\gamma = X$ entonces no se altera la pila
  - si $\gamma = YZ$ entonces X se reemplaza por Z y Y queda en el tope (push).
- q_0: estado inicial
- Z_0: símbolo inicial de la pila
- F: conjunto de estados de aceptación

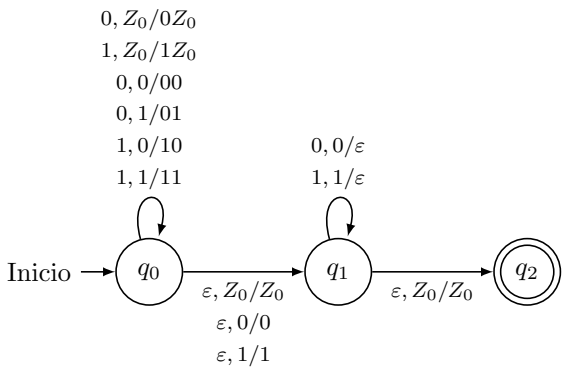
Ejemplo: diseñar un autómata de pila que acepte $\{ww^R \mid w \in (0+1)^*\}$, esto es $S \to 0S0 \mid 1S1 \mid \varepsilon$

1. Comenzar en el estado q_0 que simboliza el estado "no hemos llegado a la mitad". Mientras que estemos en q_0 leemos símbolos y los almacenamos en la pila.
2. q_1 simboliza la presunción de haber llegado a la mitad, aquí comparamos el símbolo leido con el tope de la pila, si coinciden se extrae el símbolo (pop) sino coincide este "ramal" se detiene (asumimos mal).
3. Si vaciamos la pila en q_1, entonces aceptamos la cadena hasta este punto (q_2)

$$\begin{aligned}
\delta(q_0, 0, Z_0) &= \{(q_0, 0Z_0)\} & \delta(q_0, \varepsilon, Z_0) &= \{(q_1, Z_0)\} \\
\delta(q_0, 1, Z_0) &= \{(q_0, 1Z_0)\} & \delta(q_0, \varepsilon, 0) &= \{(q_1, 0)\} \\
\delta(q_0, 0, 0) &= \{(q_0, 00)\} & \delta(q_0, \varepsilon, 1) &= \{(q_1, 1)\} \\
\delta(q_0, 0, 1) &= \{(q_0, 01)\} & \delta(q_1, 0, 0) &= \{(q_1, \varepsilon)\} \\
\delta(q_0, 1, 0) &= \{(q_0, 10)\} & \delta(q_1, 1, 1) &= \{(q_1, \varepsilon)\} \\
\delta(q_0, 1, 1) &= \{(q_0, 11)\} & \delta(q_1, \varepsilon, Z_0) &= \{(q_2, Z_0)\}
\end{aligned}$$

3.5.2. Representación gráfica

Un arco con etiqueta $a, X/\beta$ del estado q al estado p, indica que $\delta(q, a, X)$ contiene el par (p, β), sin restringir a que existan más pares.



3.5.3. Procesar cadenas con un autómata de pila

A continuación se muestra el proceso que seguiría el autómata para aceptar la cadena 1111, por brevedad se omiten los caminos que detienen el autómata sin aceptar la cadena:

$$\begin{aligned}(q_0, 1111, Z_0) &\longrightarrow (q_0, 111, 1Z_0) \longrightarrow (q_0, 11, 11Z_0) \\ &\longrightarrow (q_1, 11, 11Z_0) \longrightarrow (q_1, 1, 1Z_0) \longrightarrow (q_1, \varepsilon, Z_0) \\ &\longrightarrow (q_2, \varepsilon, Z_0)\end{aligned}$$

3.5.4. Ejercicios

Diseñe autómatas de pila para

1. $\{0^n 1^n \mid n \geq 1\}$
2. $\{0^m 1^n \mid m > n, n \geq 0\}$
3. $\{a^n b^m c^{n+m} \mid n \geq 0, m \geq 0\}$
4. $\{a^n b^{n+m} c^m \mid n \geq 0, m \geq 0\}$
5. $\{a^n b^n c^m d^m \mid n \geq 0, m \geq 0\}$
6. $\{a^n b^m \mid 0 \leq n \leq m \leq 2n\}$
7. $\{a^m b^n c^p \mid m = n \ \vee \ n = p\}$

3.6. Conversión de gramáticas a autómatas de pila

Dada una gramática independiente del contexto G, es posible construir un autómata de pila P que simule las derivaciones por la izquierda de G. El autómata P no tendrá estados finales, en lugar de esto, reconocerá una cadena de entrada cuando la cadena sea consumida en su totalidad y la pila quede vacía, es decir, ε.

En cualquier gramática, todas las producciones que no son cadenas terminales pueden ser representadas por $xA\alpha$, donde x es una cadena de terminales a la izquierda de la primer variable A y α es la cadena de terminales y variables a la derecha de A. La idea es que la pila contenga la expresión $A\alpha$, y la subcadena de terminales x habrá sido consumida de la cadena de entrada w.

Sea $G = (V, T, P, S)$ entonces $P = (\{q\}, T, V \cup T, \delta, q, S)$ es un autómata que reconoce por pila vacía. Note que el símbolo inicial de la pila es S, pero esto no indica que está vacía pues $S \neq \varepsilon$.

1. Para cada variable A

$$\delta(q, \varepsilon, A) = \{(q, \beta) \mid A \to \beta \text{ es una producción de } G\}$$

2. Para cada terminal a

$$\delta(q, a, a) = \{(q, \varepsilon)\}$$

Ejemplo: convertir la gramática

$$E \to I \mid E * E \mid E + E \mid (E)$$
$$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

a un autómata de pila.

Primero $P = (\{q\}, \Sigma, \Gamma, \delta, q, E)$, donde el alfabeto de entrada $\Sigma = \{a, b, 0, 1, *, +, (,)\}$ y el alfabeto de la pila $\Gamma = \{a, b, 0, 1, *, +, (,), I, E\}$. Ahora δ estaría definido como:

$$\delta(q, \varepsilon, E) = \{(q, I), (q, E * E), (q, E + E), (q, (E))\}$$
$$\delta(q, \varepsilon, I) = \{(q, a), (q, b), (q, Ia), (q, Ib), (q, I0), (q, I1)\}$$
$$\delta(q, a, a) = \{(q, \varepsilon)\} \qquad \delta(q, *, *) = \{(q, \varepsilon)\}$$
$$\delta(q, b, b) = \{(q, \varepsilon)\} \qquad \delta(q, +, +) = \{(q, \varepsilon)\}$$
$$\delta(q, 0, 0) = \{(q, \varepsilon)\} \qquad \delta(q, (, () = \{(q, \varepsilon)\}$$
$$\delta(q, 1, 1) = \{(q, \varepsilon)\} \qquad \delta(q,),)) = \{(q, \varepsilon)\}$$

El autómata resultante consumiría, por ejemplo, la cadena $a * (a + b00)$ de la siguiente manera:

$$\begin{aligned}(q, a*(a+b00), E) &\vdash (q, a*(a+b00), E*E) \vdash (q, a*(a+b00), I*E)\\ &\vdash (q, a*(a+b00), a*E) \vdash (q, *(a+b00), *E) \vdash (q, (a+b00), E)\\ &\vdash (q, (a+b00), (E)) \vdash (q, a+b00), E)) \vdash (q, a+b00), E+E))\\ &\vdash (q, a+b00), I+E)) \vdash (q, a+b00), a+E)) \vdash (q, +b00), +E))\\ &\vdash (q, b00), E)) \vdash (q, b00), I)) \vdash (q, b00), I0)) \vdash (q, b00), I00))\\ &\vdash (q, b00), b00)) \vdash (q, 00), 00)) \vdash (q, 0), 0)) \vdash (q,),)) \vdash (q, \varepsilon, \varepsilon)\end{aligned}$$

3.6.1. Ejercicios

Convierta a autómata de pila las siguientes gramáticas:

1. $$S \to 0S1 \mid A$$
   $$A \to 1A0 \mid S \mid \varepsilon$$

2. $$S \to aAA$$
   $$A \to aS \mid bS \mid a$$

Procese las cadenas:

3. 01101001 con el autómata resultante del ejercicio 1.
4. abaaaa con el autómata resultante del ejercicio 2.

Capítulo 4

Máquinas de Turing

La simple pregunta ¿qué lenguajes pueden ser definidos por un dispositivo computacional? nos lleva a cuestionarnos también ¿qué pueden hacer las computadoras?

Veamos como ejemplo, un código en C con el que se pretende encontrar un contraejemplo para el Último Teorema de Fermat: "no existen tres enteros positivos x, y, z que satisfacen la ecuación $x^n + y^n = z^n$ para $n > 2$".

```
int expo(int x, int n) /* regresa el valor de x a la n */
{
    int res=1;
    for(int j=1; j<=n; j++)
        res *= x;
    return res;
}
int main()
{
    int n, total, x, y, z;

    scanf("%d",&n);
    total=3;
    while(1)
    {
        for(x=1; x<=total-2; x++)
        {
            for(y=1; y<=total-1; y++)
            {
                z=total-x-y;
                if(expo(x,n) + expo(y,n) == expo(z,n))
                    printf("%d %d %d\n",x,y,z);
            }
        }
        total++;
    }
}
```

Ahora consideremos, que en lugar de imprimir los valores para x, y, z cuando los encuentre, el programa simplemente imprimiría la cadena `"Hello, world!"`. Generalizando a cualquier programa, podemos definir el *problema hola mundo*, en el que dado un programa y su entrada queremos determinar si el programa imprime `"Hello, world!"` o no.

El problema hola mundo, es un problema de *decisión* cuyas únicas respuestas son *sí* o *no*. Y aunque a primera instancia parece sencillo, el problema hola mundo es *indecidible*, es decir, para algunas entradas no podrá dar una respuesta.

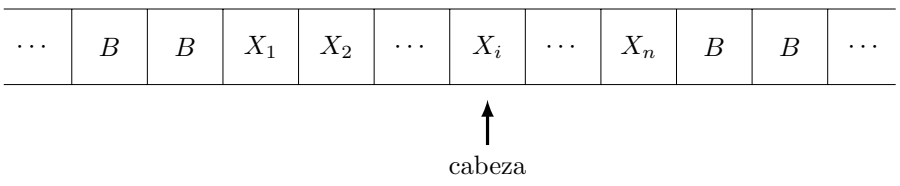
4.1. Máquina de Turing

En 1936 Alan M. Turing propuso una máquina modelo para "todas las posibles computaciones", que hoy se conoce simplemente como máquina de Turing.

La máquina de Turing cuenta con una cinta de capacidad infinita y una cabeza o lector de la cinta. Las transiciones que efectúa la máquina describen el movimiento de la cabeza lectora y las modificaciones al contenido de la cinta.

Formalmente, una Máquina de Turing se define como $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ donde:

- Q es el conjunto finito de estados (control)
- Σ es el conjunto finito de símbolos de entrada
- Γ es el conjunto de símbolos de la cinta, en el entendido que $\Sigma \subset \Gamma$
- δ es la función de transición $\delta(q, X) = (p, Y, D)$ donde:
  - q: estado actual
  - X: símbolo leído de la cinta en la posición actual
  - p: siguiente estado
  - Y: símbolo en Γ que se escribe en la posición actual de la cinta
  - D: dirección L o R, indicando la dirección en la que se mueve la cabeza
- q_0 es el estado inicial
- B es el símbolo "en blanco" tal que $B \in \Gamma$ y $B \notin \Sigma$
- F es el conjunto de estados finales



Ejemplo: construya una Máquina de Turing que reconozca el lenguaje de las cadenas con uno o más 0 seguidos de igual número de 1, es decir, $\{0^n1^n \mid n \geq 1\}$. La cadena de entrada se encuentra en la cinta y la cabeza justo debajo del primer símbolo.

La máquina que se propone en este ejemplo, reemplazará cada 0 por una X y cada 1 correspondiente por una Y, entonces, si la cadena pertenece al lenguaje la cinta deberá contener una cadena en $\{X^nY^n \mid n \geq 1\}$. La presencia de algún símbolo adicional, sea 0 o 1, indicará que la cadena original debe ser rechazada. Los estados podrán definirse del siguiente modo:

- q_0: si encuentra un 0 lo marca y va a q_1, si acabó de procesar todos los ceros (encontró una Y) entonces va a q_3.
- q_1: avanza a la derecha hasta encontrar un 1 (sin alterar 0 y Y), lo marca y pasa a q_2.
- q_2: avanza a la izquierda hasta encontrar una X (saltando 0 y Y), se mueve a la derecha y cambia a q_0.
- q_3: consume todas las Y avanzando a la derecha hasta encontrar un blanco y pasa a q_4
- q_4: Aceptar

Entonces $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$, donde δ está definido por:

	0	1	X	Y	B
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
q_4					

Ejemplo 2: procese las cadenas 0011 y 0010 con la Máquina de Turing descrita con anterioridad.

- Procesar 0011

$$\begin{aligned} q_00011 &\vdash Xq_1011 \vdash X0q_111 \vdash Xq_20Y1 \vdash q_2X0Y1 \vdash Xq_00Y1 \\ &\vdash XXq_1Y1 \vdash XXYq_11 \vdash XXq_2YY \vdash Xq_2XYY \vdash XXq_0YY \\ &\vdash XXYq_3Y \vdash XXYYq_3B \vdash XXYYBq_4B \end{aligned}$$

  dado que llega al estado final, la MT se detiene y acepta la cadena.

- Procesar 0010

$$\begin{aligned} q_00010 &\vdash Xq_1010 \vdash X0q_110 \vdash Xq_20Y0 \vdash q_2X0Y0 \vdash Xq_00Y0 \\ &\vdash XXq_1Y0 \vdash XXYq_10 \vdash XXY0q_1B \end{aligned}$$

  dado que no hay transición definida para q_1 para el símbolo B, la MT se detiene fuera de un estado final y rechaza la cadena.

4.1.1. Ejercicios

Diseñe Máquinas de Turing que procesen cadenas de la forma indicada. Una vez diseñada su MT, utilice `alana` para verificar que su comportamiento sea el correcto.

1. Aceptar el lenguaje $0^n1^n2^n$ para $n \geq 0$

2. Computar en unario $a + b$. Al inicio la cinta contiene una cadena de la forma 0^a10^b, al término la cinta deberá contener una cadena de la forma $0^a10^b20^{a+b}$

3. Computar en unario ab. Al inicio la cinta contiene una cadena de la forma 0^a10^b, al término la cinta deberá contener una cadena de la forma $0^a10^b20^{ab}$

4. Computar en binario $a + b$. En la cinta se tendrá $(0 + 1)^+2(0 + 1)^+$ y se deberá dejar como $r3a2b$ donde las cadenas $a \in (0 + 1)^+, b \in (0 + 1)^+$ y $r \in (0 + 1)^+$ es la cadena resultante de efectuar la suma en binario de las cadenas a y b

Bibliografía

[Hopcroft 2001] Hopcroft, John E., *Introduction to automata theory, languages and computation*, 2nd edition, Addison-Wesley, 2001

[Hopcroft 1979] Hopcroft, John E., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979